Procedural Generation
City Modeling

Artur Alkaim

Instituto Superior Técnico, Universidade de Lisboa
arturalkaim@tecnico.ulisboa.pt
http://tecnico.ulisboa.pt/

Abstract. The existing graphic creation tools are geared for manual use. Unfortunately, the manual production of large amounts of complex architectural forms is very time consuming. Procedural generation of these forms is one of the approaches which considerably speed up this process. This approach consists in the algorithmic construction of these forms through diferent techniques like shape grammars, L-Systems, etc. This project aims to explore this approach and apply it to the procedural modeling of cities by the development of a tool that, in connection with the Rosetta tool, will provide a new mean for the users to explore this approach. (. . .)

Keywords:
Procedural Generation, City, Modeling

1 Introduction (2/3pgs)

As technology evolves and people get new and more powerful devices, they want to take advantage of that with more detailed and complex contents to have more realistic experiences. And this is observable in the graphic contents. With more de definition of the screens and the computational power of the machines beating records, the graphic content have to follow up that characteristics in quantity as well as in quality. The issue is that the manual content generation takes a long work time from architects and designers to achieve this quality, thus implying high costs.

This graphic contents are mainly used for enterteinment, both in the gaming and movie industries, but it is also used in a lot more different areas. The fields of architecture and design, for instance, use this technique to experiment and model new designs, from smal objects like a plate to buildings or even entire cities. So also in this field they face also the problems that raises from the modelling of really big sets of objects and forms manualy. This work focus on this problem of content creation for the fields of architecture and design.

The obvious answer to this problem of manual content creation costs is to contract more architects or designers to each project to increase the production, but experience have shown that this solution is not scalable, that means that

double the number of architects or designers working in a project will not double their overall productivity. And this solution have a big impact on costs, that would take immediately out of the market new producers with less resources.

A solution for this problem is the use of generative design. That is a design method that is based on a programming approach which allows architects and designers to model complex shapes with significantly less effort.

There is an area of research that addresses this problem with *Procedural Content Generation*, or *Procedural Generation*. This have applications for example in the creation of large and/or complex scenarios for games and movies or the creation of models to use for simulation of cities. This examples involves the generation of large amounts of forms that would be impracticable with the manual approach to content creation.

2 Overview

This Section will provide an overview over this topic providing background (. . .). The Section 3 will address the objectives for this thesis work. Section 4 will explore different related works to this program. Section 5 will describe the architecture of the proposed solution. Section 6 will explain how our results will be evaluated and we will conclude on Section 7.

2.1 Procedural Generation

Procedural Generation is the algorithmic generation of content instead of the usual manual creation of content. This can be applied in almost all forms of content, but is mostly used in graphics creation and sound (music and synthetic speech).

The key property of procedural generation is that in describes the data entity, be it geometry, texture or effect, in terms of a sequence of generation instructions rather than as a static block of data [10]. This allows the production of big volumes of detailed, and high quality graphic content without the costs, both in time and price, of manual content creation.

2.2 Overview - Procedural Modelling Techniques

2.2.1 Fractals A fractal is defined in [6] as "a geometrically complex object, the complexity of which arises through the repetition of a given form over a range of scales". This concept is observed in some forms that exist in nature. From trees, mountains, coastlines to the network of neurons on a human cortex can be seen as examples of fractals. Natural shapes tend to be irregular and fragmented and exhibit a complexity incomparable to regular geometry [13]. In [6] is proposed to think of fractals as a new form of symmetry, *Dilation Symmetry*, which is when an object is invariant over a change of scale. This invariance migth be only qualitatively and not exact. For instance, a river network exhibit dilation symmetry if *zooming in* in some part looks the same as the whole image. As this

example, many others show dilated symmetry. As clouds, tree branches and some vegetables as shown in Figure 2. These examples are fractals.

Fig. 2: Fractals in Nature

This idea was applied in maths with the evolution of a new area in this science called fractal mathematics. The objective of this field is to describe this very complex shapes. With really simple rules as repeating a substitution pattern.

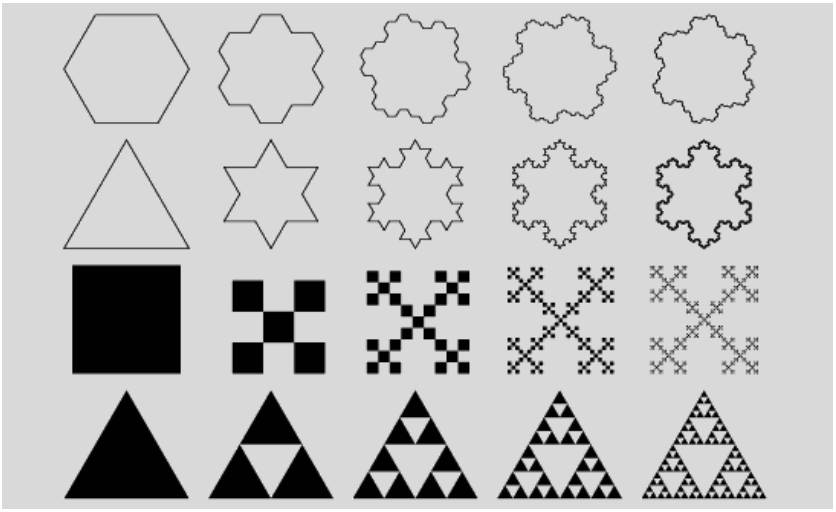


Fig. 3: Geometric Fractals

In the Figure 3 there are four examples of Geometric Fractals, with the first five iterations of each one. All of them are built by the substitution of a part of the image by another one.

The example of the second row is known as the Koch snowflake. In this example, at each iteration, all the line segments are replaced by four segments

with 1/3 of the size of the original one with the two in the middle being placed in a angle forming a equilateral triangle with the original line that is removed.

It's clear that the detail that is presented in each iteration increases as the scale changes. To try to mesure this evolution there is the idea of fractal dimension in which the detail in a pattern changes in comparison with the scale in which it is measured ($Fractal\_dimension$).

A fractal shape is defined by a recursive algorithm and successive recursions create more detail. There is no theoretic limit to the recursion size and with this a fractal is infinitely detailed.

2.2.2 Cellular Automaton It's a model of a system of cells within a grid with a determined shape, each of this cells can be on one of a finite set of states. It evolves during a finite amount of time steps with a set of simple rules according with the state of the neighbouring cells. The neighbourhood of the cell can be defined in many different ways, the most common is the use of the adjacent cells.

In the case where each cell have two possible states and the next generation state depends only on the previous state of the cell and the two immediate neighbors is called an *elementary cellular automaton*. In this case we have $2^3 = 8$ possible patterns for a neighborhood and $2^8 = 256$ sets of possible different rules. This rules are reffered by their *Wolfram code*, defined by Wolfram.

A common initial state for yhis elementary cellular automata is a random line. But to able to compare the results between rules and get clean results other option is to start with a line with zeros except the middle cell with one. Applying this second option and the following set of rules (the rule 30):

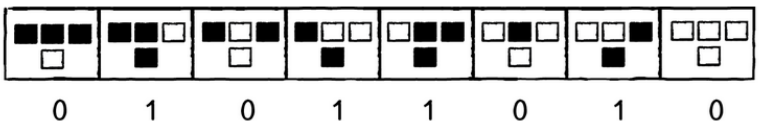


Fig. 4: Example Production Rules[16]

we get the pattern in the Figure 5 that represents the evolution of this Cellular automaton over generations:

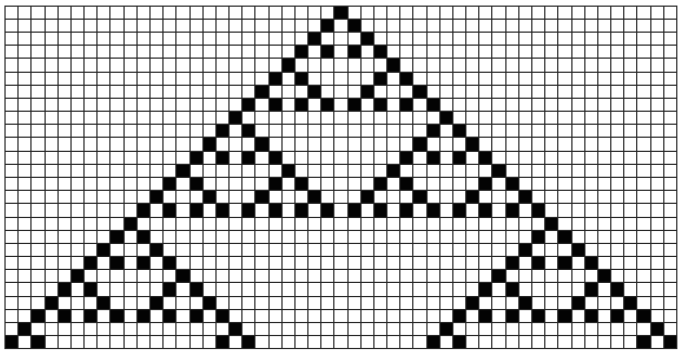


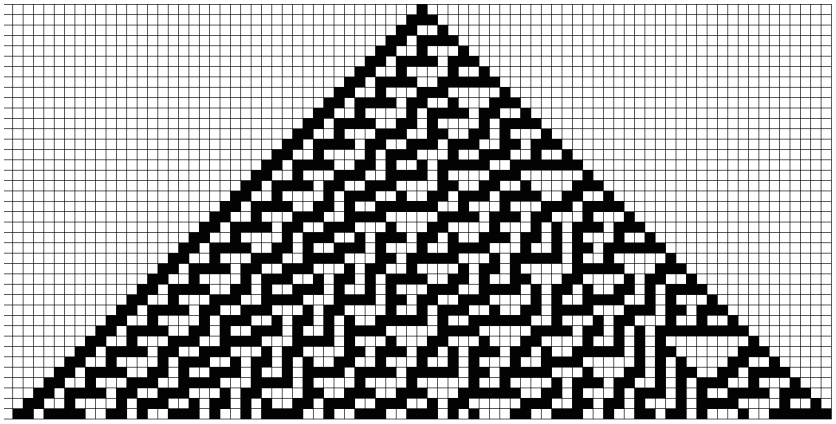
Fig. 5: Sierpiński Triangle, rule 90

In Figure 5 each line represents an iteration of the system with the application of the rules. With this set of rules a Sierpiński triangle is reproduced.

Cellular automata are used mainly to model phenomena that occurs in the physical world, most of them can only express the basic idea of a phenomenon but some are accurate enough to be able to make predictions.

In this context, cellular automata are used to model natural shapes and textures, the Figure 6a shows a natural texture on a Textile Cone Snail that looks like the patterns formed with the cellular automaton in the Figure 6b.

a)



b)

Fig. 7: Example of the representation of natural patterns with cellular automata. a) Natural Shell, image from [16]. b) Pattern formed with the rule 30.

2.2.3 L-Systems Lindenmayer Systems (L-Systems) are a class of string rewriting mechanisms, originally developed by Lindenmayer as a mathematical theory for plant development.

One L-System is a type of a formal grammar and a string rewriting system that is capable of describe the behaviour of plant cells and model the growth processes of plant development.

It consists of two different parts, one axiom and a set of production rules. The axiom is the starting point of the system, acting as a seed. Then is't applied in this seed the set of production rules, that change the initial string and producing other strings. This is an iterative process, so after the production of a larger set of strings, the rules can be applied to each one of them wish grows the size of the set even larger.

This L-Systems are used to produce natural growth of vegetation (Figure 8), and the generation of Fractals.

Fig. 8: Trees with L-Systems

In this process, each symbol is associated with a production rule. For instance having $\{F, +, -\}$ for the alphabet and *production* $\{F \rightarrow F+F--F+F\}$. From a starting axiom *aba*, and the application of the rules we have:

$$F \tag{1}$$

$$F + F - -F + F \tag{2}$$

$$F+F--F+F\ +\ F+F--F+F\ -\ -\ F+F--F+F\ +\ F+F--F+F \quad (3)$$

This is an example of the evolution of one system where the production is applied in (1) that turns into $F + F - -F + F$. In Note that the space between the symbols are just for readability.

All the symbols are assigned with a geometric meaning. The notion of a turtle with a pen, as proposed in [4], with the symbols being interpreted as moving instructions to the turtle, is a simple way to understand. If "F" means forward and the symbols "+" and "-" are interpreted as rotations counter-clockwise and clockwise respectively by a predefined angle.

Using the given example, and setting the angle for the rotation to 60° the result is Figure 9.

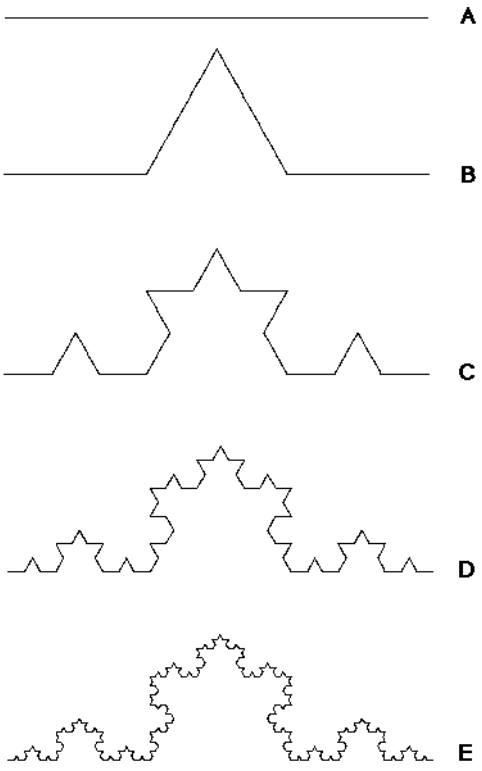
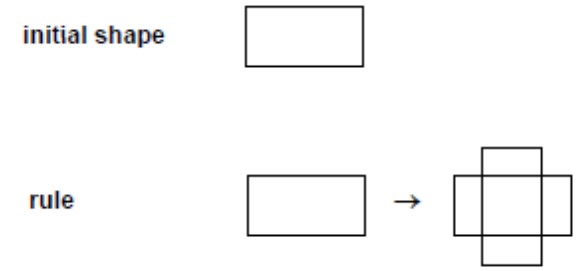


Fig. 9

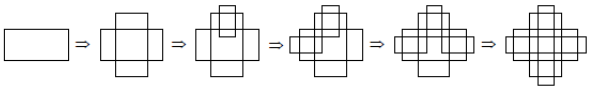
2.2.4 Shape Grammars Shape Grammars can be considered grammars for design. In stead of having symbols or letters as components of the alphabet, it has shapes that can be in 2D or 3D. And have production rules that are composed by this shapes, and specify the evolution of the system. With this process, similar to the L-Systems explained before, the shape starts from a seed, i.e. a usually simple shape and can evolve to one big and/or complex shape.

The process is performed in two steps, the recognition of a shape and the replacement according to the rules that are previously defined.

The Figure 11 exemplify one shape grammar, with one rule and the evolution of the application of this rule to the shapes iteratively. In this image, it's shown that from very simple initial shape, can be generated a complex from with a few iterations.



a)



b)

Fig. 11: a) Grammar Tiles b) Recursion steps

This is applied to the generation of buildings in the CityEngine 4.1 system, using 3D blocks for the main form, and 2D shapes to design the facades.

The Figure 12 shows a simple building that I modelled using CityEngine and it's CGA Shape Grammar. But CGA is powerful enough to model much more complex buildings like the Figure 13.

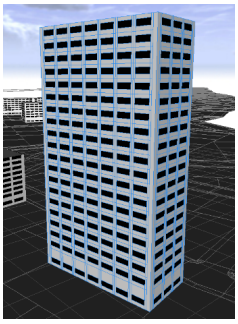


Fig. 12: Simple Building

Fig. 13: Complex Building [14]

2.2.5 Tiling A common solution to give realism to 3D objects is the application of textures. One problem is that this textures takes a lot of memory space. To work around this problem an easy solution is to have a small texture piece, a tile, and repeat throughout the objects. And this is called tiling. Or as defined in [19]: "A plane-filling arrangement of plane figures or its generalisation to higher dimensions.". This means, the result of constructing a plane from a finite set of "tiles".

If this technique is use naively commonly results in not very homogeneous textures, it depends much on the set of tiles that are used. If the borders of all the tiles are the same the result is always homogeneous. But if the borders are very different the chances to have a not uniform texture rises. This example, Figure 14, from [3] shows how a bad structured tiling system produces a not homogeneous texture.

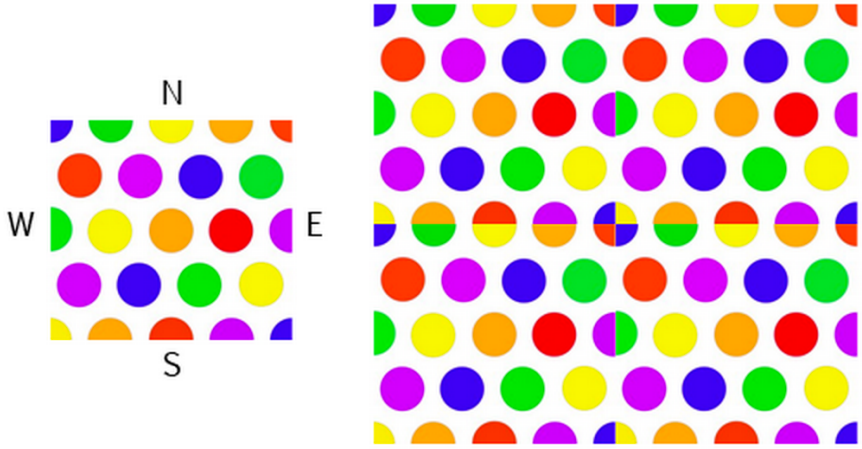
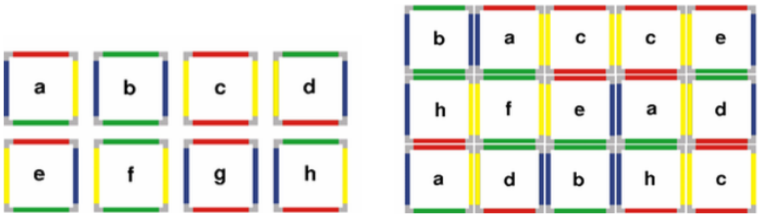


Fig. 14: One tile and an irregular pattern

To make uniform planes, the boundary of each tile must be coherent, i. e., the borders of connecting tiles have to match. Given a single tile, the so-called first corona is the set of all tiles that have a common boundary point with the tile (including the original tile itself). From that, the simple method to create a homogeneous texture is to connect each tile with one that belongs to it's first corona.

The easy, most simple solution is to make sure that all the tiles have the same borders all around. With this property it's guaranteed that any created texture will be homogeneous. But this solution doesn't provide much irregularity and the results present repeating patterns.

Wang Tiles [5]. It is a solution named after Mr Hao Wang that predicted that tiling was not possible. It received his name, not only for him being wrong, but because the prove that this is possible uses much of the work he did trying to prove the impossibility. This process allows tiling with an arbitrary number of different vertical and horizontal borders and from that calculate the set of tiles that are needed to create a full texture without inconsistencies.



a) b)

Fig. 16: a) Eight Wang Tiles b) Portion of a plane

So the process assign colors to the tiles borders and then matching colored borders are aligned forming a plane.

As you might have noticed, the inner content of the tiles are not a problem. As we are trying to create the uniform textures by arranging this smaller pieces only the borders matter, so we can create a set of tiles with the same borders and whatever inner content we want. With this technique the result can be much more irregular.

Corner Tiles. [11] One alternative to Wang tiles with the points to be coloured.

The vanilla wang tiles have problems in the diagonals that are not taken into account. They are the confrontation between four tiles which leads to less homogeneous texture if we the borders don't match. By using the corners, the problem goes to the sides, that despite being larger, are only the confrontation between two tiles and therefore it leads to a more homogeneous texture.

2.2.6 Noise "To generate irregular procedural textures, we need an irregular primitive function, usually called noise" [6]. It's a pseudorandom function that gave the goal to break the monotony of a pattern and make it look more random. Perlin Noise is the most known and used noise function. It was created by Ken Perlin, for the movie Tron 1982 with the aiming to generate natural looking textures.

The psedorandom property is important and a true random function like *white noise* would not do the job. If we generate a texture based on white noise the pattern would change every time it's generated and we would like that the it stays the same, frame after frame. This is achieved with the use of inputs for this functions that with the same input returns always the same output sequence.

The properties of an ideal *noise* functions are as follows [6]:

- *noise* is a repeatable pseudorandomm function of its inputs
- *noise* has a known range, namely, from -1 to 1.
- *noise* is band-limited, with a maximum frequency of about 1.
- *noise* doesn't exhibit obvious periodicities or regular patterns. Such pseudorandom functions are always periodic, but the period can be made very long and therefore the periodicity is not conspicuous.
- *noise* is *stationary* - that is, its statistical character should be translationally invariant
- *noise* is *isotropic* - that is, its statistical character should be rotationally invariant

With this noise function, it's generated a sequence of values that are interpolated to generate a coherent noise. With the application of *turbulence* that is composition of several layers of this noise with different frequencies and amplitudes forming a coherent noise. This layers are called *Octaves* and the ratio between amplitude and frequency of the layers can be expressed as a constant known as *persistence* [9]. With the result we can create a texture that looks natural and with fractal like structure.

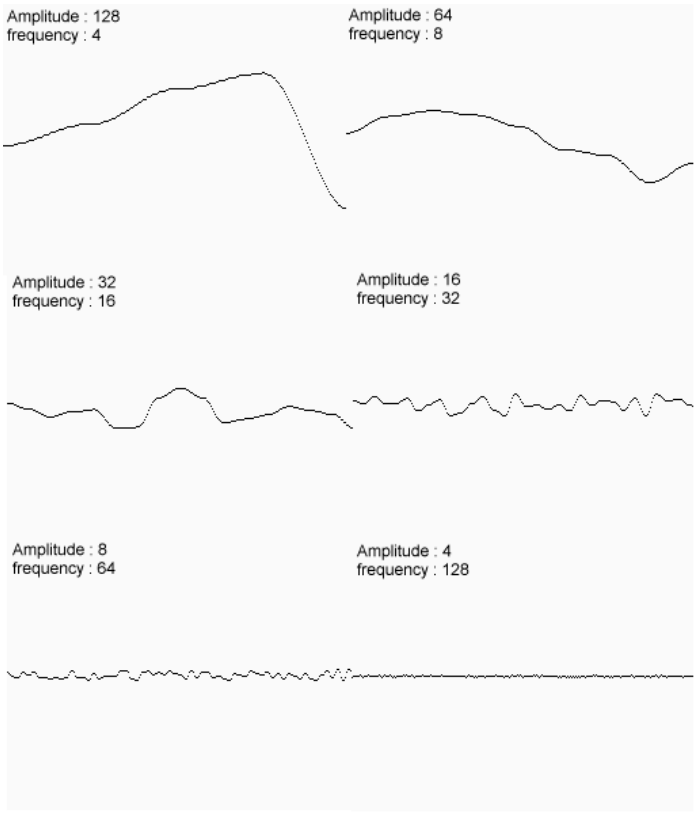


Fig. 17: Different noise functions

For instance, the Figure 17 shows the result of the interpolation over six noise functions with different frequencies and different amplitudes. And the sum of all this functions is the exibithed in the Figure 18 [2].

Noise can also be used to generate planes. The method used is the same as the 1D problem but we have to generate a lot more data points that after are interpolated as a plane. This results in noisy images that are often used to model clouds, smoke and other textures with similar visual properties, Figure 20. Another application for this technique is the generation of height maps like the one in figure .

Another application for Noise planes are for *object placement* on a grid. By creating a noise plane with the same size of the grid, with each cell of the grid corresponding to one pixel of noise, the object placement is done by choosing each object for each cell according to the noise value. Figure 21 shows a city which the buildings where placed with the use of noises.

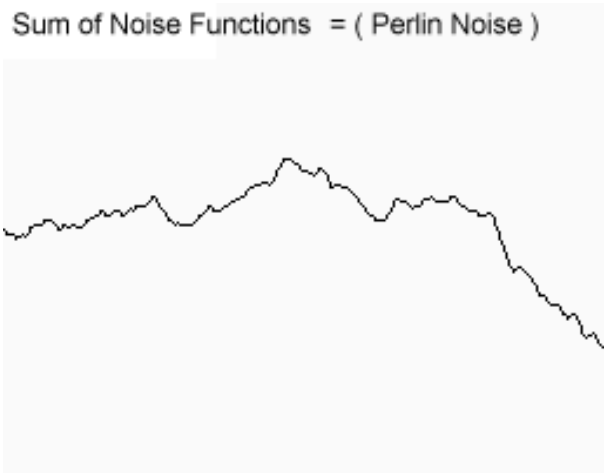


Fig. 18: “You may even imagine that it looks a little like a mountain range.”

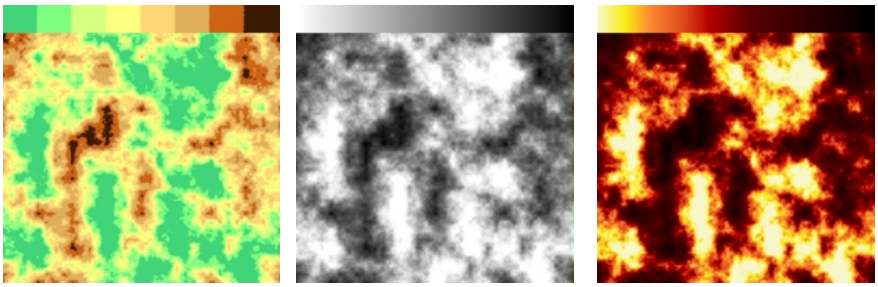


Fig. 20: Gradient maped textures from [1]

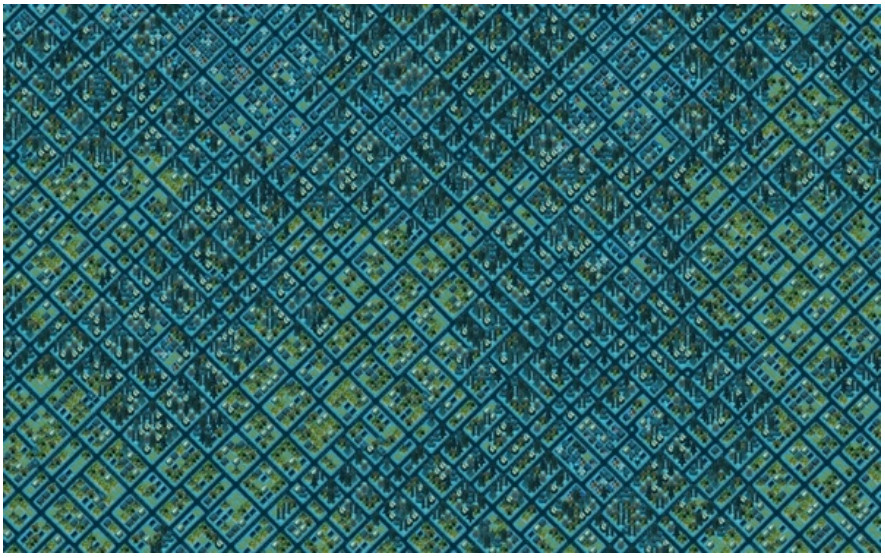


Fig. 21: Objects Placed from a noise plane

2.2.7 Voxels [NOT DONE] ... "Voxel is a combination of "volume" and "pixel" ". i. e. the equivalent to pixels in 3D. In other words, if a pixel is a tiny square that represents a part of an model in a plane (picture), a voxel is a cube that represents a part of a model in a space. Because we usually what we want to model have three dimensions, it's easier to visualize this models with voxels. We can see the world as a gigantic 3D matrix, and the voxels represent what is inside each point in the matrix.

Given that this voxels are just a way to store information, we still need a way to visualize this information. And this is the part with this approach. How to visualize surfaces out of this 3D boxes. Since most of the hardware has evolved and beed adapted to work with textured triangles, and also a big volume of the research is arround textured triangles problems. (...?)

2.3 Architectural Styles

Most buildings or other man made structures are classified by it's *Architectural Style*. An architectural style is characterized by the features that make a building notable and historically identifiable. This style features are related to form, building materials, construction methods, etc. and change trough different places and over the years.

Even with this constant change in the styles there are some rules that we have to followed. The most obvious one is the chronological time. If we are modelling a city from the 16th century it's wrong if we add buildings with styles form the 19th century. And we have to think also in a simmilar way with the places, if it's an asian city we do not add buildings with a known european layout.

And this correctness in relation to the architectural styles is also an important concern to this work.

2.4 Urban Planinng [TODO]

3 Objectives (1pg) [NOT REVIEWED]

The world population has grown at a faster and faster pace, which is causing overcrowding and lack of resources in various locations of the globe. For example in the Emirates, the economic boom supported by the oil business is creating lots of work oportunities, thus thousands of people from everywere in the world are moving there. However the country was not prepared to accommodate the growing population currently living there. To solve this problem of overpopulation that occurs there, and in other countries like China, new big cities are being built in the desert completely from scratch. An example of these cities is the city of maasdar in Abu Dhabi . This city designed by Foster and Partners and currently being built in the desert have a total of $6km^2$. Who is responsible for a project of this size can not afford to make a mistake in choosing the location or city setting. To test these factors is necessary to create models , only in contrast to the relatively small cost of creating manually using traditional techniques , model a single building with some variations , nestad situations have to be modeled an entire city and even variations additional . This design phase is therefore very costly in time and effort required to enable them to obtain the best possible result.

The objectives of this project are to study the existing approaches to Procedural Generation and provide a solutions to the kind of problems, like the new city in Abu Dabi stated above, through the implementation of mechanisms for procedural generation of architectural form, with application to the generation of large amounts of forms, such as of urban environments, buildings and ornaments efficiently and according to a particular architectural style models.

With the large amount of very different techniques, explained in Section 2.2, one important part of this work is to know where to apply each techinque to obtain better results.

4 Related Work (17pgs)

As an active research field, there is a lot of work being done in this area. In this section I give an overview of the related work that has been carried out on procedural generation of cities. The generation of cities can be viewd as the sum of a few parts. The generation of terrains, road networks, buildings and other man made structures and cities in general.

The first step, the *generation of terrains* is often left out of procedural generation of cities, because there is also lot of research focused only on this area, mainly to produce models of nature. In this work I will not evaluate specificaly this factor but make specific comments when relevant.

The *road network* is a fundamental step for the generation of any city. It gives the city it's structure and defines the overall look. Most of the road networks are much different from each others and this makes the generalization dificult, but we can see some patterns when we analyze a road network from real cities. And this patterns are important for us if we want to have procedures that mimic this complex patterns. Some cities a have tightly structured grid network, like New York or a concentric radial pattern like Paris and others are purelly organic with the network presenting a nearlly random pattern. And we also have all the shades of grey in between.

Next step is the *generation of building*. This is also not easy, since that there is no limit to the number of different buildings that can be modeled. Since buildings naturally are different from each other both in functionality and style, this is a complex problem to model. One widelly adopted solution is to use group buildings by functionality. The most common groups are *Commercial*, *Residential* and *Industrial* buildings. After this we have to care only about modelling different styles adapted to each functional group. This part is also hard because this "style" can be anything. The architectural styles, as explained in the Section 2.3, have influences from a lot of different sources.

The last step is about the generation of the city itself, mainly join all the problems stated above and find a realistic layout for the city and also the generation of the ornamentations for the city like trees and parks. Where to put each building is an important question that can have big impacts in the overall look of the models. Also how to group buildings and create neighborhoods, the function of the building is crucial, industrial buildings usualy are far from the residential ones, the same way that the comercial buildings are close together and also close to the residential areas.

4.1 CityEngine [15] [14]

It's a three-dimensional (3D) modeling software developed by Procedural Inc. (now part of the Esri RD Center). It's specialized in the generation of 3D urban environments. With the procedural modeling approach, CityEngine enables an efficient creation of detailed and large-scale 3D city models with a lot of control from the user.

4.1.1 RoadNetwork The first part to procedurally generate a city is to create a road network to become a backbone of the city and provide an overall structure. For that, CityEngine receives as input maps such as land-water boundaries and population density. From that input a network of highways is created to connect the areas off high density population, and small roads connect to the highways. This growth process continues until the average area of each lot is the desired one. The system have a default value, bat it can be set by the user to a different one.

To implement this growth process, it's used an L-System, that computes the road network.

Fig. 22: Road Map growth

The Figure 22 shows the evolution of this process in a map of Manhattan. The first two on the top shows the process in different phases during the process, the middle line is the result of the process and the bottom line is the real map of Manhattan for comparison.

4.1.2 Buildings To implement the generation of buildings, they created the CGA Shape.

CGA Shape is a Shape Grammar that was introduced by Pascal Muller, Peter Wonka and others, in a paper called "Procedural Modeling of Buildings"[15]. It is defined as "a novel shape grammar for the procedural modelling of CG architecture, produces building shells with high visual quality and geometric detail." To do so, this grammar uses a group of well defined production rules.

This tool allows the user to model buildings with an high control and in different ways. It can be done by text, writing production rules from a shape grammar or with a visual language like Grasshopper 3D, that is nice for simple works but it's impossible to work with a slightly more complex work.

Mass Modeling To model a building the first step is to create a mass model of the entire building by assembling basic shapes. With scaling, translation rotation and split applied to basic shapes namely I, L, H, U and T as shown in the Figure **??**.

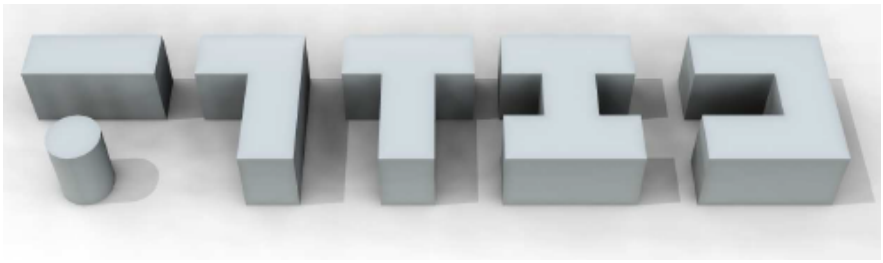


Fig. 23: caption

The next step is to add the roof, from a set of basic roof shapes or general L-Systems.

After that, with the application of the grammar rules in the created mass, it's possible to create complexity to the level that is desired, being able to produce high complex buildings like the one in the following picture.

4.1.3 Cities The result can be an city like Figure 24, with approximately 26000 buildings.

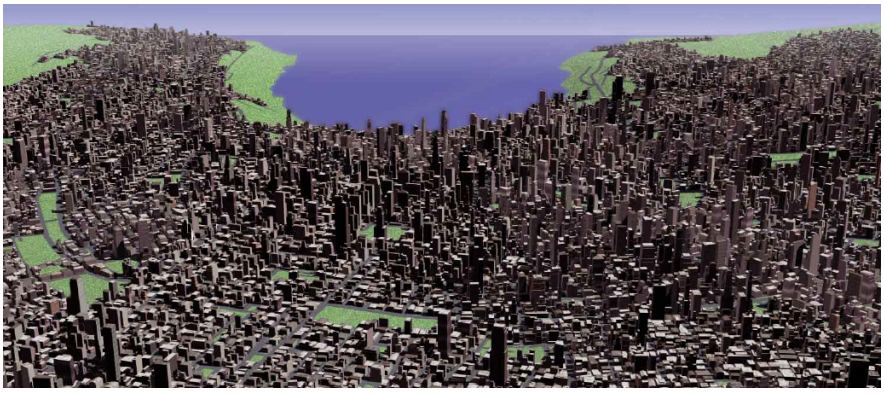


Fig. 24: City with approximately 26000 buildings.

City Engine results can be imported by Maya, to achieve better results. Like the Figure 25, that represents a 'virtual' Manhattan.

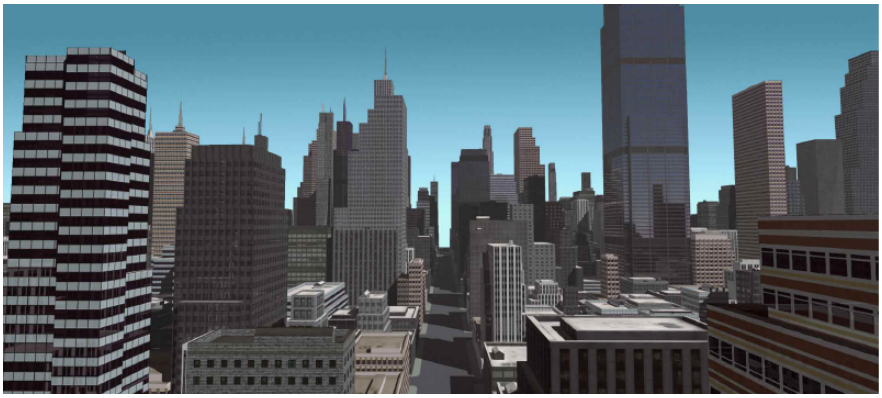


Fig. 25: City rendered with Maya.

4.2 Undiscovered City

In [8] Stefan Greuter et al. presented a system that generates in Real-time pseudo infinite virtual cities which can be interactively explored from a first person perspective. In their approach "all geometrical components of the city are generated as they are encountered by the user." As shown in the Figure 26 only the part of city that is inside the viewing range is generate.

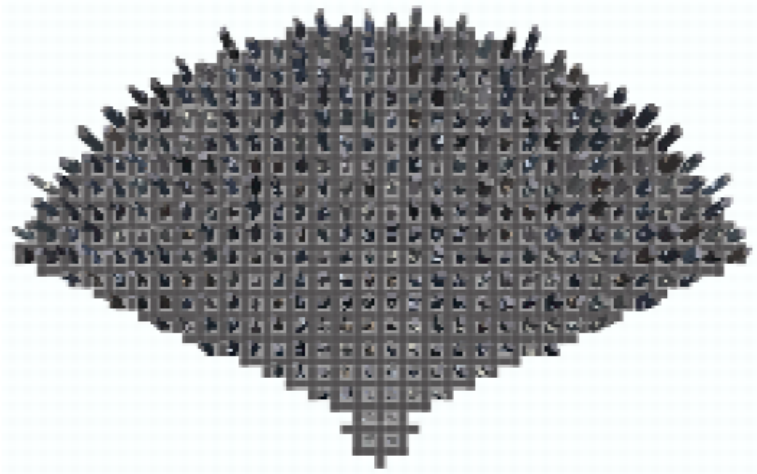


Fig. 26: Viewing Range

4.2.1 Road Network The system uses a 2D grid that divide the terrain into square cells. The cells represent proxies for the content that will be procedurally generated. Before the content of each cell is generated, the potential visibility of it is tested, and after that, only the visible cells are filled with content.

After that the roads are created in a uniform grid pattern. This grid does not feel very natural, and in the continuation of the work, this system evolved into a more realistic one with the join of some of the grids to create a less uniform distribution of the buildings.

4.2.2 Buildings To compute the form and appearance of each building, it is used a "single 32 bit pseudo random number generator seed. The random sequence determines building properties such as width, height and number of floors." Similar sequences of number result in similar buildings. To avoid that, it is used a a hash function to convert each cell position into a seed.

To generate a building is first is generated a floor plan. To do so, it's randomly selected and merged a set of regular polygons and rectangles, then this is extruded. This is an iterative process, that creates sections from the top to the bottom, by adding more shapes to the the initial shape and extruding as shown in the Figure **??**. Starting from the left, first there is a simple polygon, that is merged with a rectangle and after extrusion, forms the first block that will be the top of the building. After that, another extrusion is made to generate the next block followed by the merge of a rectangle to the floor shape and the generation of a new block and so on.

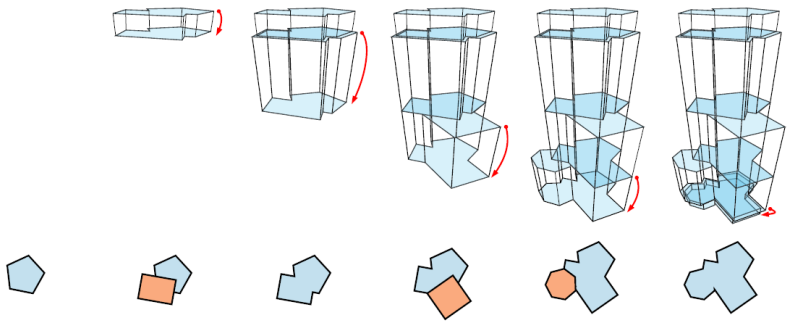


Fig. 27: buildings

With the application of this method very complex architectural forms can be generated, depending only on which forms are selected and the order that is used to merge them.

4.3 CityGen

CityGen [9] it's an interactive system that aims to "rapidly create the urban geometry typical of a modern city". The users can interact and control the

generation process. The system, like others, is able to generate road networks that act as foundations to the model. It also can generate buildings but can not achieve the complexity and realism of other systems.

4.3.1 Road Network CityGen divided this problem in two steps. First the generation of the *Primary Road Network*, and after that, the *Secondary Road Network*. This two steps use different methods to generate the roads. Undirected planar graphs are used to represent all roads. Two graphs for the Primary roads and one for each zone to store the secondary roads.

Primary Road Generation The primary road network uses two graphs, one high level graph that correspond directly to the primary road intersections. It represents the topological structure of the city by it's primary roads, and connections between them. The user is allowed to manipulate this high level graph, to change the high level structure ("topography of the primary road network") of the city .
There is also the low level graph that is generated from the other one, and defines the real path that the roads have in the terrain. It has the same nodes as the first graph and many more, that indicate the points on the terrain which the road passes.
To generate the low level graph it is used "sampling, plotting and interpolation processes".

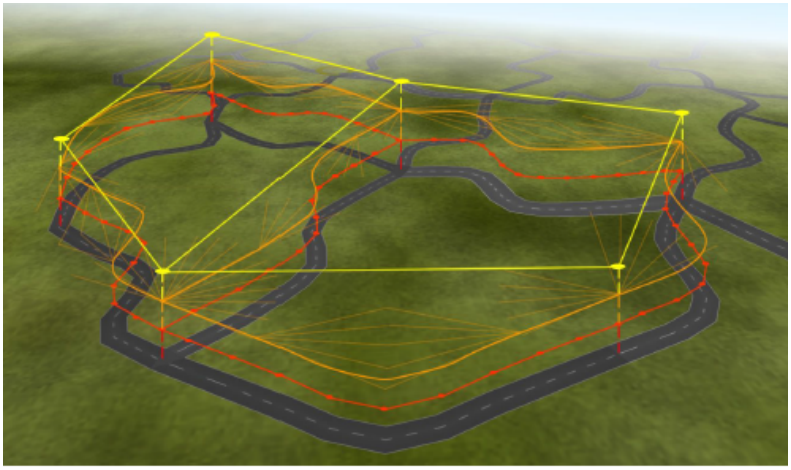


Fig. 28: The lighter graph is the High level graph, and the evolution to the darker Low-level graph

Secondary Road Generation The author defined city cells as districts, that are the areas of terrain that are enclosed by primary roads. The secondary road network is generated

inside this cells using a growth based algorithm similar to the L-Systems technique as shown in Figure 29.

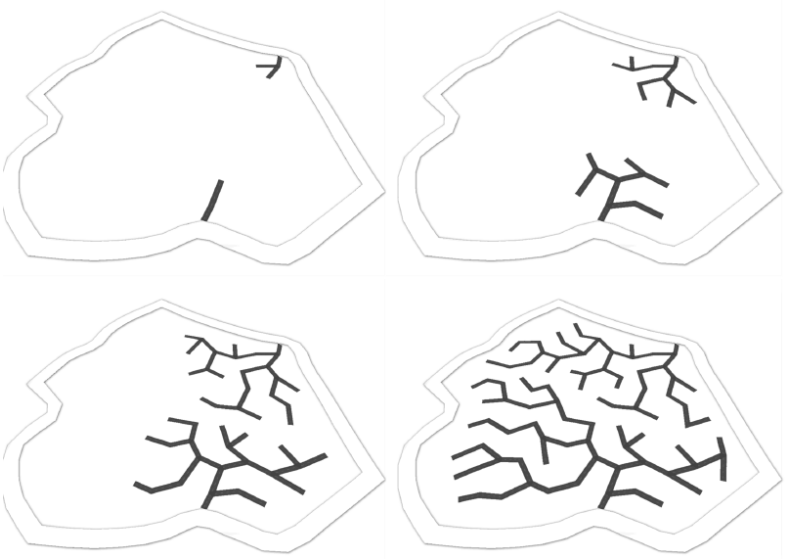


Fig. 29

4.3.2 Buildings This system generates buildings also. Each building is created in lots that are identified after the extraction of enclosed regions, called blocks, from the secondary graph. Lots that don't have direct access to the roads are excluded.

Based on the type of block the building footprints are created. After that building geometry is generated by extruding the footprint. The height of each is determined by a height parameter and a noise factor that can be also manipulated. A block is shown in the Figure 30, with only primitive shapes.

With this primitive shapes, CityGen uses "advanced materials with shaders to simulate additional geometry".

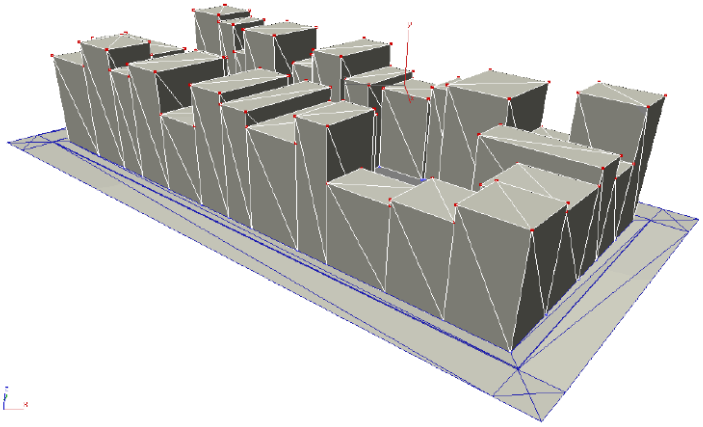


Fig. 30: Primitive Shape Buildings

4.4 Inverse Design [NOT DONE]

In [18] it is presented a different solution from the others already presented.

It's described a framework that enables high level control of the modelling process. It "provides a mechanism to interactively edit urban models". They apply inverse design to solve the problem of output control. From an existing model, the user can specify high level indicators that describe the desired output and the system change the underlying rules and parameters to get the result as close as possible to the desired.

As the Figure 31 shows, the user can change the "low level" parameters and the "high level" indicators to control the final output of the system.

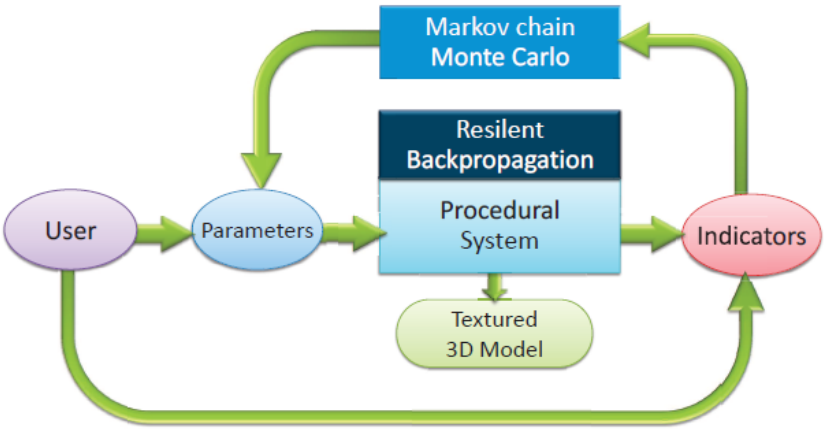


Fig. 31: System Pipeline [18]

This framework uses the indicators as a goal to optimize the parameters. To calculate the values for the parameters they used a version of Monte Carlo Markov Chains (or MCMC) and Resilient Back Propagation.

They implemented a urban procedural engine "similar to previous city-level procedural modelling work". It was inspired by urban planners, that use the place types concept to represent coherent design patterns of buildings and streets.

With this approach, the authors claim two main advantages, *Abstraction* and *Interactivity*. They argue that this solution allows urban planners and designers to work at a high level of abstraction, enabling users to manipulate place types, parameters and indicators to create the 3D model they want without wasting time with low level tasks as implementation of low level rules or parameter tuning.

At the same time this approach enables the users to interactively manipulate very complex indicator targets with a "sophisticated enough" methodology to map target indicators to input low level parameters.

With their teilored version of MCMC and back-propagation they are able to support complex indicators "enabling control beyond global shape, sush as by high-level semantics and indicators", and by considering the procedural model as a black box there aren't any limitations to the used grammar.

(. . . ?)

4.5 CityBuilder [NOT DONE]

CityBuilder is a system introduced by Watson et. al in [12] and [17] "Procedural City Modeling" and "Procedural Modeling of Land Use in Cities".

It aims to be self automated to minimize necessary input, only needs the terrain description. Although it allows some other input from the user to give some interaction and control.

To achieve that, it uses agent based simulation to create a system of agents and behaviours that can model specific entities of a city as developers, planning authorities and road builders. The set of rules for each agent is small to achieve a simple behaviour. With that, they what to make their "model extendible not only in regard to the types of structures that are produced but also in describing the social and cultural influences prevalent in all cities."

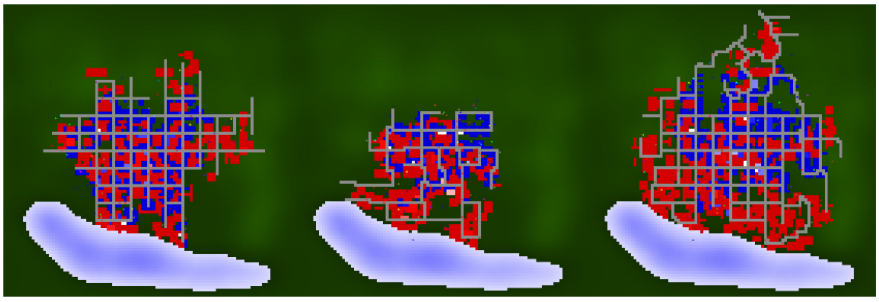


Fig. 32: Different Road Netowrks

4.5.1 Road Network The road network is designed by the agents based on the input terrain description wish will describe the height map and forbidden areas for roads, as water.

The user can specify that the roads must follow a pre-established gridded pattern, or give the freedom to network to grow more organic. The image presents this two options and a combination of the two with a centre with a gridded pattern ant organic surroundings.

There are two types of road building agents, the *extenders* and the *connectors*:

extenders The *extenders* search the area around exiting developments to look for areas that are not being served by any road to expand the current road network.

connectors The *connectors* roam trough existing roads and sampling random points in the road network within a given radius. It tries to reach that point trough the road via a BSF. If it cannot reach or the distance needed is over a threshold value, the agent tries to create a road between these two points.

4.5.2 Buildings This system does not develop buildings, but it's developer agents generate parcels and specify the use of the land. They can identify "at least nine different types of land usages". They perform the role of urban developers that buy land, request planing permission, build and sell. They track the usage of their lands and specify the parameters to the buildings that can be build there.

4.5.3 The City This system develop an evolving conceptual model of one city, that can represent the growth and evolution of a city through the time.

Building Envelopes

4.6 Building Envelopes [NOT DONE]

Sabri Gokmen in [7] presents a way to create envelope systems for buildings. In this case he had a approach that was inspired in Gotheam morphology and leaf venetian patterns.

This article explains form as described by Johann Wolfgang von Goethe in the late eighteenth century. He started by working on annual plants through the work of Linneaus. Goethe considered the external properties of plants to be the result of an internal principle, idea that was influenced by the prior topological work by Linneaus that classified plants according to their physical characteristics. For Goethe this external properties are not constant and change over time according to environmental conditions.

Forms in architecture are described as a "topological entity following an overall schema or as a replication of an exiting type that appears fixed (Garcia, 2010)". It says that architects often work with topologies for various buildings and variations are achieved using topological operations. Adding to this is the idea of parametric design to create "smooth variable systems". This parametric top-down systems are able to control the overall behaviour of design and have been used to evaluate and adapt performance based approaches in design solutions. "However this systems are However these systems are ineffective to provide a morphogenetic approach towards design."

Because morphology considers form as the result of a bottom-up process, it is a better method to model growth. Because the form is not defined from start and is the result of the growth process.

Goethe mixed this two ideas namely performance based and morphogenetic approaches. The parametric property is not throughout the system but applied in parts that grow with a morphogenetic approach.

Leaf venetian patterns: There isn't one certain theory about the guiding principles that support the leaf patterns, there are many theories that try to explain it. One of them is called canalization theory that observes this patterns really as a distribution network, so it depends on the concentration/distributions of auxin producers to efficiently distribute auxin throughout the plant.

Computation of leaf venetian patterns: The growth algorithm that is presented in this paper is based on the canalization theory. It starts by generating a density map that guides de distribution of auxin sources, that is the second step. Finally it generates the leaf venetian patterns from the auxin source map.

The generation of the venetian patterns starts by setting root nodes that will be the points from which the patterns will start growing. Then "at each time step the closest vein node to each auxin source will be defined. Then these nodes will grow towards the average direction influenced by the auxin sources".

4.7 Scene City:

[**?**]

Scene City is an addon for the 3d application Blender.

"It varies the colors, materials and procedural elements enough to produce a reasonable approximation of a city at a distance, but realism is not a goal, and it does not operate on real data at all." (http://vterrain.org/Culture/BldCity/Proc/)

4.8 ghostTown:

http://kilad.net/site/ http://www.kilad.net/GTForum/

It's a script plugin for 3DS Max from Autodesk. "Ghost Town is a script that procedurally generates cities and urban environments in only a few clicks, and features a number of options. Such as low or high poly buildings, road layouts, vehicles, trees, facades and an easy to use material and texture system." (http://cgi.tutsplus.com/tutorials/create-a-detailed-city-with-3d-studio-max-ghost-town–cg-10090)

4.9 Skyscraper:

http://www.skyscrapersim.com/index.shtml Standalone

"Skyscraper aims to be a fully-featured, modular, 3D realtime building simulator, powered by the Scalable Building Simulator (SBS) engine. The main feature SBS provides is a very elaborate and realistic elevator simulator, but also simulates general building features such as walls, floors, stairs, shaftwork and more. Many more things are planned, including gaming support (single and network multiplayer), and a graphical building designer. Skyscraper is written in C++ and uses the OGRE graphics engine, Bullet for collisions and physics, FMOD for sound, the wxWidgets GUI library, and is multiplatform. The current versions aim for a future 2.0 release." from the official site.

4.10 Blended Cities:

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

"Blended Cities is an open-source city generator for Blender. it allows to create quickly a large amount of streets and buildings, with various shapes. B.C. fights against squared things : curved streets and odd or cylindric buildings can be created simply, so you can create old towns, not only modern cities."

5 Architecture (2/3pgs) [Working]

To model a city, there are different points that need to be care about differently. I define four layers for the generation of a city model, the first is the *ground*, the generation of the terrain with the elevations and lower areas, possibly with water, in form of a height map. The next layer is the elaboration of an *urbanistic plan ??* that defines the areas of a city, the number of higher density zones and the distribuition of them throughout the terrain. With that, we can start the *road network* generation by defining where the main roads will be placed as they are built connecting this high density zones. With this high level road network we have the city structure that will be filled in by smaller networks that will model the neighborhoods and after that the lots. The last layer are the *buildings* that will be based on the lots previously defined. This lots have to be classified by types, as residential, comercial or industrial.

5.1 Ground

The ground will be generated with the use of height maps. This height maps could be user input if, like the example of the chinese city, the goal is to generate a city over a specific real place or it can be randomly generated through the use of noise planes.

This heigth maps will be generated form noise planes, explained in Section 2.2.6. This will allow the creation of natural look height planes that will provide a realistic foundation for the city that will be modeled upon.

5.2 Urbanistic Plan

This phase will start by defining the number of high density areas (HDAs) and next to place them in the terrain.

The number of HDAs will be estimated by the desired size of the city. Then I have to place this points on the terrain and to get variability this distribution will be made randomly but according to some constrains. This constrains are minimum distance between HDAs and

With the high density areas placed on the terrain, this points act as nodes in a graph that and they are connected by main roads. This provides the primary structure for the remaining road network to be built upon.

5.3 Road Network

The generation of the road network will be made by the use of growth based techniques like L-Systems that will make possible to achieve good results fast.

5.4 Building

The buildings will be randomly generated from a set of predefined rules, that will take into account the type of building and the architectural style that will define each building.

6 Evaluation (1/2pgs) [NOT DONE]

To evaluate this work I will be focus on three points, Realism, Scale and Efficiency. First for realism, th

Evaluation method taken from [10] to use as an inspiration

1. Realism – Does the generated city look like a real city?
2. Scale – Is the urban landscape at the scale of a city?
3. Variation – Can the city generation system recreate the variation of road networks and buildings found in real cities or is the output homogeneous?
4. Input – What is the minimal input data required to generate basic output and what input data is required for the best output?

5. Efficiency – How long does it take to create the examples shown and on what hardware are they generated? How computational efficient is the algorithm?
6. Control – Can the user influence city generation and receive immediate feedback on their actions? Is there a tactile intuitive method of control available or is the control restricted? To what degree can the user influence the generation results?
7. Real-time – Can the generated city be viewed in real-time? Are there any rendering optimisation techniques applied to enable real-time exploration?

7 Conclusions [NOT DONE]

Wrap up what you wrote.

References

1. How to use perlin noise in your games. http://devmag.org.za/2009/04/25/perlin-noise/. Accessed: 2015-01-15.
2. Perlin noise. http://freespace.virgin.net/hugo.elias/models/m_perlin.htm. Accessed: 2014-10-16.
3. Procedural world blogspot. http://procworld.blogspot.pt/. Accessed: 2014-10-16.
4. H Abelson. Aa disessa. *Turtle geometry*, 1982.
5. MF Cohen, J Shade, S Hiller, and O Deussen. *Wang tiles for image and texture generation*. 2003.
6. David S Ebert, Forest Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and modeling: a procedural approach*. 2002.
7. Sabri Gokmen. A Morphogenetic Approach for Performative Building Envelope Systems Using Leaf Venetian Patterns. 1:497–506, 2013.
8. Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. GRAPHITE 2003 Real-time Procedural Generation of ‘ Pseudo Infinite ’ Cities. 2003.
9. George Kelly. An Interactive System for Procedural City Generation. 2008.
10. George Kelly and Hugh Mccabe. A Survey of Procedural Techniques for City Generation.
11. Ares Lagae and Philip Dutré. An alternative for Wang tiles: Colored edges versus colored corners. *ACM Transactions on Graphics*, 25(4):1442–1459, October 2006.
12. Thomas Lechner, Ben Watson, and Uri Wilensky. Procedural city modeling. *In 1st Midwestern Graphics Conference*, 2003.
13. Benoit B Mandelbrot, Dann E Passoja, and Alvin J Paullay. Fractal character of fracture surfaces of metals. 1984.
14. Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM SIGGRAPH 2006 Papers on - SIGGRAPH ’06*, page 614, 2006.
15. Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’01*, pages 301–308, 2001.
16. Daniel Shiffman. *The Nature of Code*. 2012.
17. Thomas Lechner, Ben Watson, Pin Ren, Uri Wilensky, Seth Tisue and Martin Felsen. Procedural Modeling of Land Use in Cities. 2004.
18. Carlos A Vanegas, Ignacio Garcia-dorado, Daniel G Aliaga, Paul Waddell, and U C Berkeley. Inverse Design of Urban Procedural Models. 2009.
19. Eric W. Weisstein. Tiling. From MathWorld–A Wolfram Web Resource.http://mathworld.wolfram.com/Tiling.html. Accessed: 2014-10-16.