Procedural Generation

Your Thesis subtitle

Artur Alkaim and António Leitão

Instituto Superior Técnico, Universidade de Lisboa
{arturalkaim,antonio.menezes.leitao}@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. The obvious answer to this problem 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.

Most computer-aided design (CAD) application provide progamming languages for generative design, programs writen in these languages have very limited portability and are not pedagogical and most of them are poorly designed

or obsolete, reasons that create barriers to adherence to this approach by users that normally are not used to code [13].

(Talk aboud DrRacket and how user friendlly it is. . . .)

The tool Rosetta is an extensible IDE for generative design, based on DrRacket and targeted at architects and designers [9] created to address this problem by providing a user friendly integrated development environment (IDE) and a set of different languages.

There is an area of research that addresses this problem with *Procedural Content Generation*, or *Procedural Generation*.

2 Overview

2.1 Procedural Generation

This Section will provide an overview over this topic providing backgroud. 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 **??**.

Procedural Generation is the algorithmic generation of content in stead 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 [6]. This allows anyone with less resources to produce high detailed, and high quality content.

2.2 Overview - Procedural Modelling Techniques

Fractals A fractal is defined in [3] 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 [10]. In [3] is proposed to think of fractals as a new form of symmetry, *Dilation Symmetry*, which is 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.

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. 2: Fractals in Nature

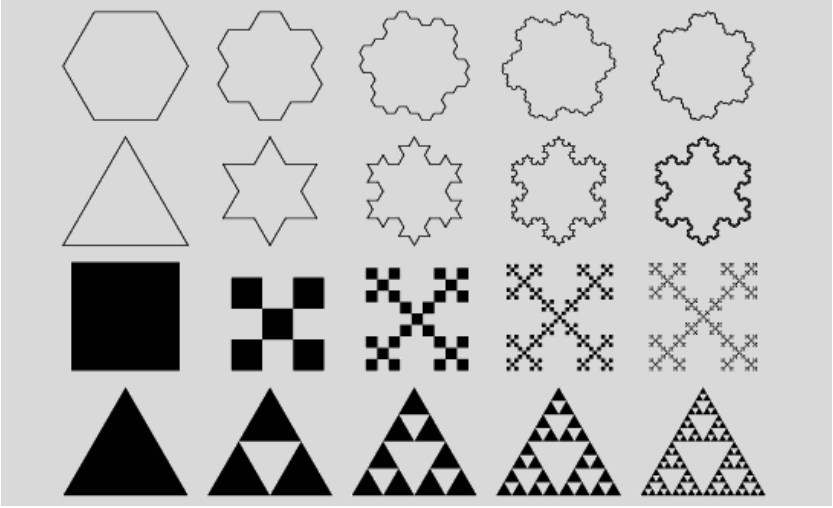

Fig. 3: Geometric Fractals

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.

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.

Starting from a line with zeros except the middle cell with one and the following set of rules:

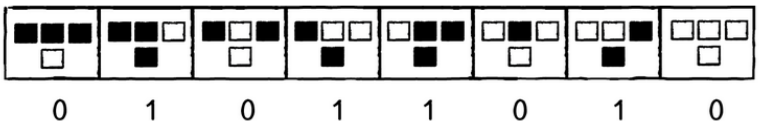

Fig. 4: Example Production Rules[14]

The Figure 5 image represents the evolution of one Cellular automaton over time. In this case, each cell can have one of two states, black or white. Starting from a white line except the middle cell that is black and the presented set of rules.

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.

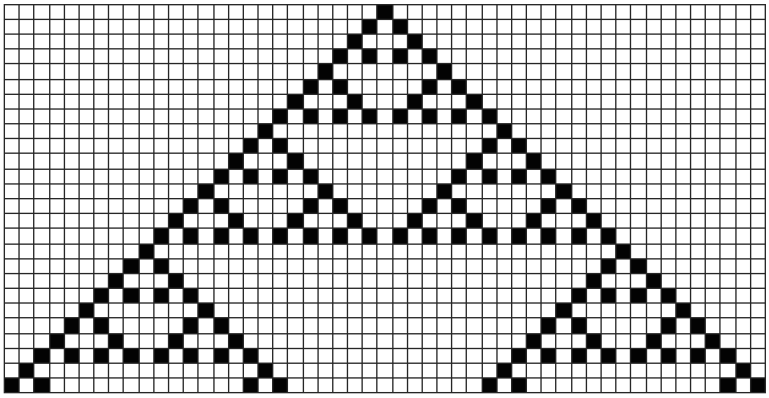

Fig. 5: Sierpiński Triangle

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 6), and the generation of Fractals.

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:

$$F \tag{1}$$

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

$$F+F--F+F \ + \ F+F--F+F \ - \ - \ F+F--F+F \ + \ F+F--F+F \tag{3}$$

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 [2], with the symbols being interpreted as moving instructions to the turtle, is a simple way to understand. If "F" means forward

Fig. 6: Trees with L-Systems

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 7.

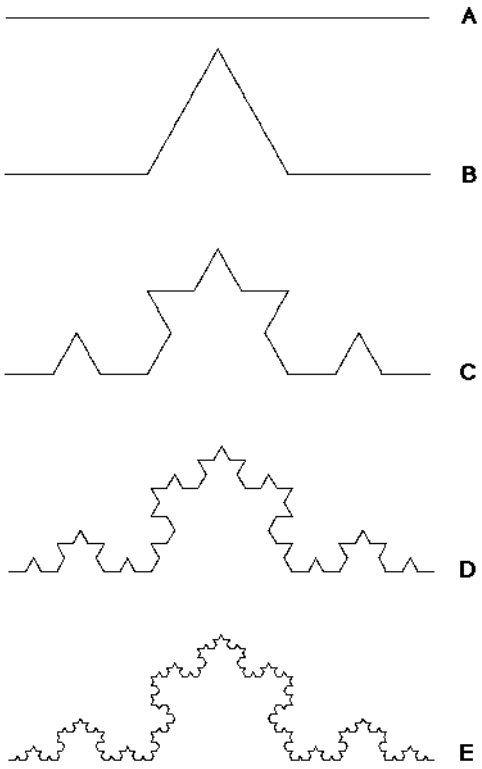

Fig. 7

Shape Grammars Shape Grammars can be considered grammars for design. In stead of having symbols or letters as components of the alphabet, it have 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 following images 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.

This is applied to the generation of buildings in the CityEngine system, using 3D blocks.

The figure on the right 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 one at the bottom.

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. 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 Wolfram: "A plane-filling arrangement of plane figures or its generalisation to higher dimensions.". This means, the result 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 from [1] shows how a bad structured tiling system produces a not homogeneous texture.

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 is a solution named after Mr Hao Wang, that predicted that tiling was not possible. 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.

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 [7]: 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.

Genetic tile generation: "The bottom line for me is, Wang tiles are amazing things until you try to use them seriously. They work great for stuff you can synthesise from the ground up. If you are trying to mix samples from real life, get ready for some trouble." `http://procworld.blogspot.pt/2013/01/tile-genetics.html`

Noise "To generate irregular procedural textures, we need an irregular primitive function, usually called noise" [3]. 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. With this pseudorandom function, it's generated a sequence of values that are interpolated to generate a coherent noise. With the composition of several layers of this noise it's build a texture that look natural and with fractal like structure.

For instance, the image above shows the result of six noise functions with different frequencies and amplitudes. And the sum of all this functions is the following.

"You may even imagine that it looks a little like a mountain range."

Source: `http://freespace.virgin.net/hugo.elias/models/m_perlin.htm`

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 characterisics 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.

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 **??**.

3 Objectives (1pg) [NOT DONE]

Clearly explain the project objectives.

The objectives of this project is to study the existing approaches to Procedural Generation and to develop a new tool that will be able to generate large amouts of forms efficiently and according to a particular architectural style .

Design and implementation of mechanisms for procedural generation of architectural form , with possible application to the generation of models of urban environments , buildings and ornaments. Implementation of these mechanisms in the Generative Design tool Rosetta.

4 Related Work (17pgs)

4.1 CityEngine [12] [11]

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.

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.

The Figure 8 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.

Fig. 8: Road Map growth

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"[12]. 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 **??**.

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.

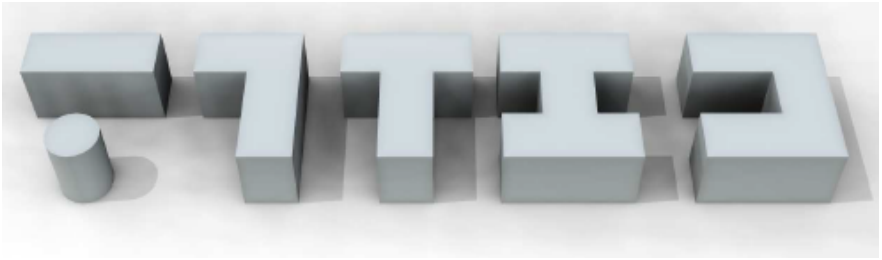

Fig. 9: caption

Cities The result can be an city like Figure 10, with approximately 26000 buildings.

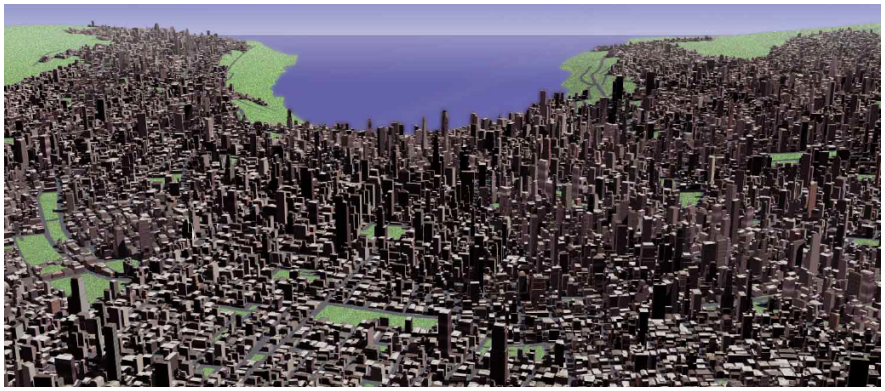

Fig. 10: City with approximately 26000 buildings.

City Engine results can be imported by Maya, to achieve better results. Like the Figure 11, that represents a 'virtual' Manhattan.

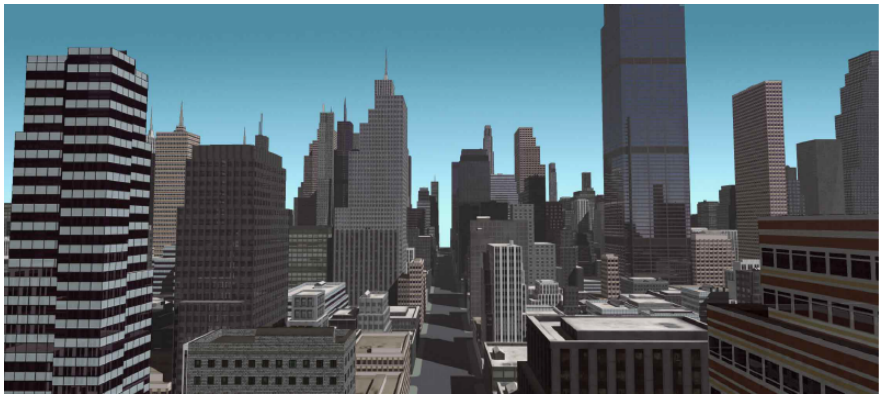

Fig. 11: City rendered with Maya.

4.2 Undiscovered City

In the "Real-time Procedural Generation of 'Pseudo Infinite' Cities" paper, from 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. "All geometrical components of the city are generated as they are encountered by the user." As shown in the following image only the part of city that is inside the viewing range is generate.

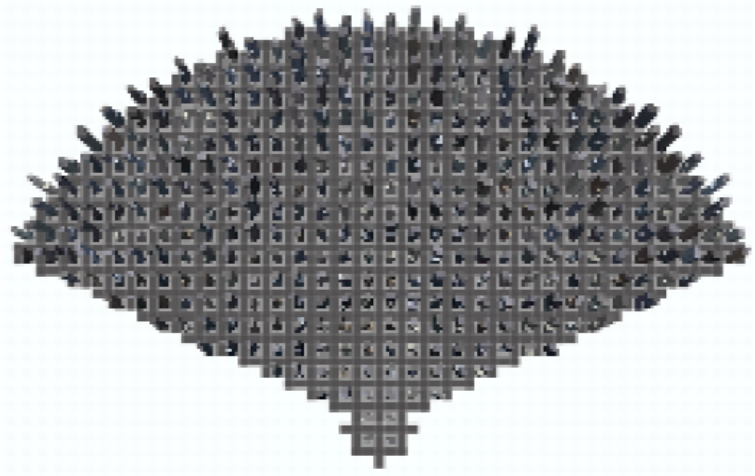

Fig. 12: buildings

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.

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 **??**.

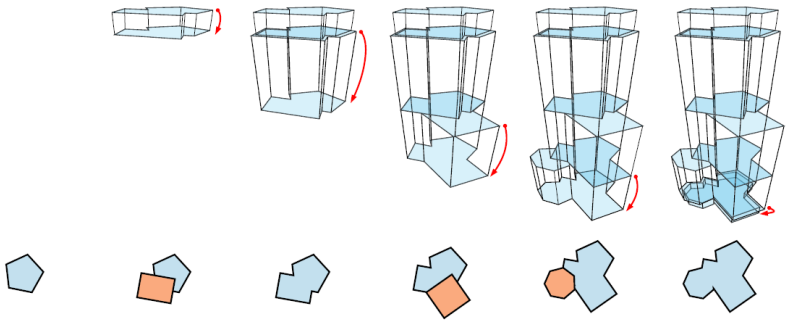

Fig. 13: buildings

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. After that is merged a rectangle to the floor shape and generated a new block and so on.

4.3 CityGen

CityGen [5] 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.

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 have 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".

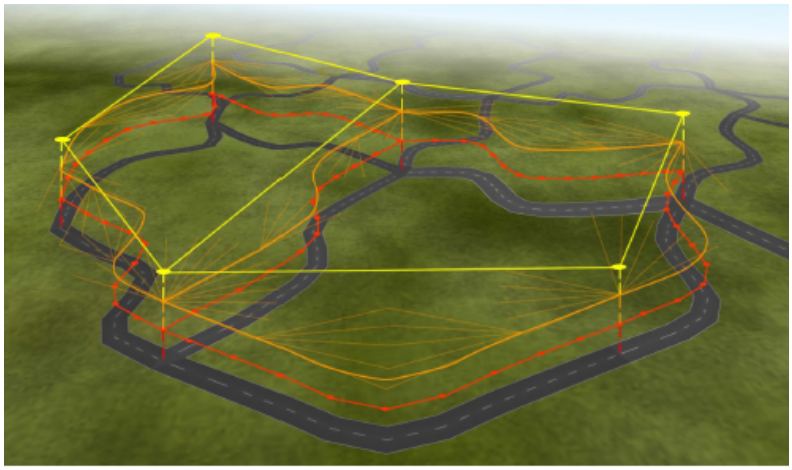

Fig. 14: The lighter graph is the High level graph, and the evolution to the darker Low-level graph

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 15.

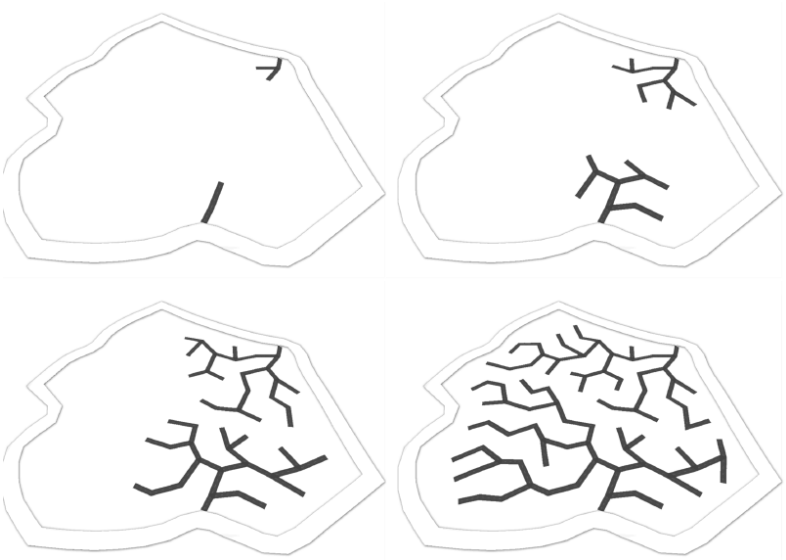

Fig. 15

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 16, with only primitive shapes.

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

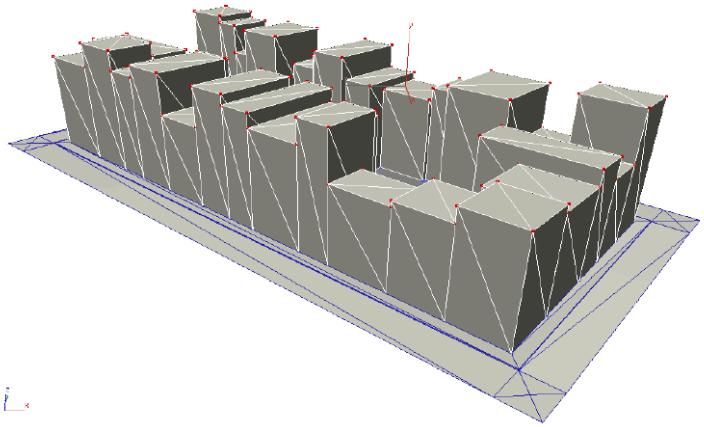

Fig. 16: Primitive Shape Buildings

4.4 Inverse Design [NOT DONE]

In [16] 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 17 shows, the user can change the "low level" parameters and the "high level" indicators to control the final output of the system.

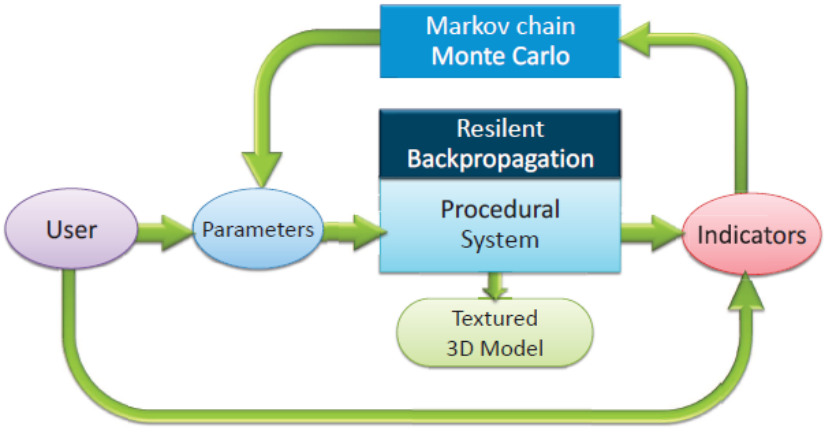

Fig. 17: System Pipeline [16]

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 [8] and [15] "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."

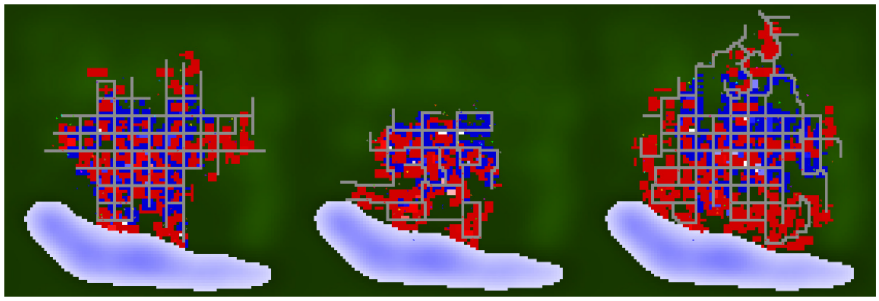

Fig. 18: Different Road Netowrks

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.

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.

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 [4] 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) [NOT DONE]

Your proposed architecture. Can have lots of pictures and bullet points so it is easy to understand.

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

[6]

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. Procedural world blogspot. `http://procworld.blogspot.pt/`. Accessed: 2014-10-16.
2. H Abelson. Aa disessa. *Turtle geometry*, 1982.
3. David S Ebert, Forest Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and modeling: a procedural approach*. 2002.
4. Sabri Gokmen. A Morphogenetic Approach for Performative Building Envelope Systems Using Leaf Venetian Patterns. 1:497–506, 2013.
5. George Kelly. An Interactive System for Procedural City Generation. 2008.
6. George Kelly and Hugh Mccabe. A Survey of Procedural Techniques for City Generation.
7. 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.
8. Thomas Lechner, Ben Watson, and Uri Wilensky. Procedural city modeling. *In 1st Midwestern Graphics Conference*, 2003.
9. Menezes Leit. Programação para Arquitectura. 2012.
10. Benoit B Mandelbrot, Dann E Passoja, and Alvin J Paullay. Fractal character of fracture surfaces of metals. 1984.
11. 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.
12. 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.
13. Pedro Palma Ramos and António Menezes Leitão. Implementing Python for DrRacket. In Maria João Varanda Pereira, José Paulo Leal, and Alberto Simões, editors, *3rd Symposium on Languages, Applications and Technologies*, volume 38 of *OpenAccess Series in Informatics (OASIcs)*, pages 127–141, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
14. Daniel Shiffman. *The Nature of Code*. 2012.
15. Thomas Lechner, Ben Watson, Pin Ren, Uri Wilensky, Seth Tisue and Martin Felsen. Procedural Modeling of Land Use in Cities. 2004.
16. Carlos A Vanegas, Ignacio Garcia-dorado, Daniel G Aliaga, Paul Waddell, and U C Berkeley. Inverse Design of Urban Procedural Models. 2009.