ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

"MIRRORER" AS A PLUG-IN FOR URBAN DESIGN

M.Sc. Thesis by
Betül TUNCER

Department : Informatics

Programme : Architectural Design Computing

Thesis Supervisor: Assoc.Prof. Dr. Sinan Mert ŞENER

MAY 2009

ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

"MIRRORER" AS A PLUG-IN FOR URBAN DESIGN

M.Sc. Thesis by
Betül TUNCER
(523071003)

Date of submission : 04 May 2009
Date of defence examination: 21 July 2009

Supervisor (Chairman) : Assoc. Prof. Dr.Sinan Mert ŞENER (İTÜ)

Members of the Examining Committee : Prof. Dr. Gülen ÇAĞDAS (İTÜ)
Assoc. Prof. Dr. Birgül ÇOLAKOĞLU (YTÜ)

AUGUST 2009

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ŞEHIR PLANLAMASI TASARIMI İÇİN 'YANSITICI'

YÜKSEK LİSANS TEZİ
Betül TUNCER
(523071003)

Tezin Enstitüye Verildiği Tarih : 04 Mayıs 2009
Tezin Savunulduğu Tarih : 21 Temmuz 2009

Tez Danışmanı : Doç.Dr. Sinan Mert ŞENER (İTÜ)
Diğer Jüri Üyeleri : Prof. Dr. Gülen ÇAĞDAS (İTÜ)
Doç. Dr. Birgül ÇOLAKOĞLU (YTÜ)

AĞUSTOS 2009

FOREWORD

I would like to express my deep appreciation and thanks for my advisor Assoc. Prof. Dr. Sinan M. Sener. I would also like to extend my special thanks to my friend for many years, Ömer Kırkağaçlıoğlu and Burak Özer for their computer expertise. Finally I would like to thank to my family, Umay Tuncer, Erdogan Tuncer and soon to be family, Ilkay Orbey for their patience and support during this process.

May 2009

Betül TUNCER

Architectural Design Computing

TABLE OF CONTENTS

Page

LIST OF TABLES **Page**

LIST OF FIGURES

Page

LIST OF FIGURES (cont.) Page

“MIRRORER” AS A PLUG-IN FOR URBAN DESIGN

SUMMARY

In this study, a plug-in is developed for an early phase site planning process. An urban block is chosen as a reference point and the plug-in is designed to generate design alternatives referring to its built environment.

In the first chapter, the subject of the thesis, objectives, scope and method of study are explained.

In the second chapter historic background of artificial intelligence, birth of knowledge based systems from artificial intelligence studies are revealed since the plug-in developed is planned to integrates both knowledge based systems and artificial intelligence in the following phases of development. On the other hand, shape grammars are explained to make it easier for the reader of this thesis to understand the field work consists of shape grammars which are explained in chapter four. Following the historic background information of approaches used in the study, design necessities are explained. Site has been taken into consideration in relation to city and different approaches to city for the sake of creating a healthy society. Establishment of visual harmony is perceived as responding to built environment. Symmetry is used to achieve this response and the idea of symmetry creating a visual harmony is supported by quotes from Palladio.

In the thrid chapter, implications of knowledge based systems are presented to acknowledge the reader about computational power of such systems therefore basing the system on knowledge is beneficial in terms of effectivness and speed in generation.

In the fourth chapter, field work regarding urban morphology generation is explained through its computer implications. These projects are selected depending on different aspects that links the project with this thesis and the plug-in that is developed. The projects explained in this chapter will include CityEngine, The Melinkov Grammar, CityZoom, Urban Design with Patterns and Shape Grammars and Smart Solutions for Spatial Planning (SSSP).

In the fifth chapter, the proposed plug-in is explained in detail. This chapter consists of what the user needs to understand for using the projected plug-in properly, the features of the projected plug-in. In the following sections of the chapter, two applied approaches of the projected plug-in will be explained.

The sixth chapter will conclude the thesis with general comments regarding the process shaping the plug-in, its potentials and restrictions along with a comparison of two applications.

ŞEHİR PLANLAMASI TASARIMI İÇİN 'YANSITICI'

ÖZET

Bu tez çalışmasında, vaziyet planı tasarlanmasının erken fazlarında kullanılmak üzere 3D Studio Max ve Catia Programlarında kullanılabilen bir modül geliştirilmiştir. Başlangıç noktası olarak bir şehir bloğu ele alınmış ve modül, çevresine uyum gösteren tasarım alternatifleri üretmesi için geliştirilmiştir.

Birinci bölümde, çalışma konusu, hedefler kapsam ve yöntem açıklanmıştır.

Ikinci bölümde yapay zeka, bilgi tabanlı sistemlerin yapay zeka çalışmalarından doğuşu ve biçim gramerlerinden bahsedilmiştir. Bunun nedeni, geliştirilen modülun ileriki zamanlarda yapay zekadan ve bilgi tabanlı sistemlerden yararlanabilecek olmasıdır. Biçim gramerleri ise, dördüncü bölümde açıklanan şehir planlamasına yönelik geliştirilen programların bir kısmının tabanını oluşturduğu için ve açıklanan bu programların okuyucu tarafından rahatlıkla anlaşılabilmesi açısından ele alınmıştır. Çalışmada yararlanılan yaklaşım ve sistemler tanıtıldıktan sonra çalışmaya zemin hazırlayan tasarım gereksinimleri açıklanmıştır. Bu kısımda, sağlıklı bir şehir oluşturmak üzere geliştirilen şehir planlaması yaklaşımları açıklanmıştır. Son olarak, simetri genel anlamıyla ve Palladio'nun mimari yaklaşımı ile açıklanmıştır.

Üçüncü bölümde bilgi tabanlı sistemler ve uygulamaları okuyucuyu bilgi tabanlı sistemlerin hesaplama gücü, üretken sistemlerdeki hızı ve verimliliği konusunda bilgilendirmek üzere açıklanmıştır.

Dördüncü bölümde kent tasarım alternatifleri üretmek üzere geliştirilen programlardan bahsedilmektedir. Her program, belirli bir özelliği ile bu tez kapsamında geliştirilen modül ile ilişkilendirilmiş olup bahsedilen programlar şunlardır: CityEngine, The Melinkov Grammar, CityZoom, Urban Design with Patterns and Shape Grammars ve Smart Solutions for Spatial Planning (SSSP).

Beşinci bölümde, modül detaylı bir şekilde açıklanmıştır. Bu bölüm, kullanıcının modülu doğru kullanabilmesi için yapması gerekenleri, modülun özelliklerini, kısıtlamarını ve potansiyelini içerir. Öte yandan tasarlanan modül ile hayata geçirilen iki farklı yaklaşım, uygulama aşamasında birbirinden farklılık gösterdiği için her iki yaklaşım ayrı ayrı açıklanıp arasındaki farklılıklardan bahsedilmiştir.

Altıncı bölüm, tezi sonuçlandırıp, modülü oluşturan süreçle ilgili genel yorumlar yapıp, potansiyellerini, kısıtlamalarını ve gerçekleştirilen iki uygulamanın karşılaştırmasını içermektedir.

1. INTRODUCTION

The most effective way in every kind of problem is to understand the problem itself and starting from the right end. This way, problems can be solved before they get complicated. A well-planned process is always the key. Therefore, a working computational product can make problems easy to solve since every computational product consists of a planned process. This is true in design problems as well. CAD systems that provide the designer with alternatives used at early design phases would initiate a well elaborated conceptual design leading to a well developed complete design. The benefits of relying on an alternative generating computational power are elimination of human error, calculation of high number of alternatives, which would be time consuming for a human designer.

Today, responsive environments are being developed. Many design projects consist of a set of sensors either responding to humanly actions such as movement or climatic conditions such as heat or humidity. However, there are not enough design projects that respond to their built environment even at the most basic level, which in this study is considered as the visual level. Taking the environment into consideration has almost been forgotten among mechanically responsive devices, robotized and iconized buildings so that today, it is harder to give a neighborhood a character because every building block in a neighborhood is disconnected. Therefore, an alternative generation system that generates an urban block in response to its built environment would help initiate neighborhoods that are visually in harmony.

1.1 Purpose of the Thesis

The aim of this study is to draw attention to ways in which urban blocks can respond to built environment. This thesis claims that the face of the city is important and the built environment should be valuable enough to look upon, mimic and even to learn from. The purpose is to manifest the fact that the current conditions do not suit the claim.

Another important objective of this thesis is to provide collaboration between the designer and computation since computational power is inevitable if the designer is dealing with high number of possibilities.

The last goal of this thesis is to create a new platform, elaborate a scale at which both architects and planners meet, initiate their designs therefore use the same design language which would also support the idea of establishment of character of a city.

1.2 Scope

Many efforts to design beautiful and healthy cities had been spent throughout the history. However, all of them had imposed a grand plan to be applied on a city. This thesis is related to city design at a scale of urban block. The urban block is perceived as the building block of a city. If the building block is well developed, it consists of a very high potential to become a well developed city as well. The method applied in the thesis does not suggest the only way to make a well developed city stemming from a well developed urban block. However, it goes as far as drawing attention to subject at hand. Defining a certain solution is beyond the scope of this thesis.

1.3 Method

The method used in this study will be to develop a plug-in suggesting a way to design an urban block that responds to its built environment. A literal approach will be followed in responding to the environment. Simply, the plug-in will mimic its environment to generate forms in elevation by taking mirror images of its neighbor across the street and placing the 2D image on the side border of the site. The plug-in will compute new site design alternatives by converting the 2D image into 3D by

giving it depth values. This process will be applied on all four sides of the site generating plan alternatives with elements merging inwards from all sides.

2. BACKGROUND

2.1 History Artificial Intelligence, Knowledge Based Systems and Shape Grammar

Computation has strong contributions to many sectors as well as construction and design. Computation is used for static load calculations, heating and cooling calculations, budget calculations. It has helped the sector in terms of speed, efficiency and reliability. However, in case of a design process, computation has only been a tool, which is used as a means of drafting and form defining. The solutions to design problems have not been addressed by computational products even today. Design problem is still being evaluated in the designer's mind as well as the strategies towards an architectural solution and they are not translated to computational products. Today, computational software can only help the design process at specific steps and can only help the designer to further iterate his thoughts rather than providing him with an ultimate solution.

An alternative program will be developed along with this research which helps the architect with a set of possible site plan solutions in response to given environmental conditions of the site. This chapter will be delving into theoretical approaches such as Knowledge Based Systems, Artificial Intelligence and Shape Grammars.

2.2 Birth of Knowledge Based Systems from Artificial Intelligence Efforts

Debates on Artificial Intelligence which will be referred to as AI henceforth, have started with the first computer trials in the 19th century. Professor Charles Babbage developed and idea for "analytical engine" which was not realized until Doron Swade proved that it in fact worked in 1993. The analytical engine was in fact held today's computational properties within. Lady Ada Lovelace, who is known as the first programmer, stated that the engine is capable of playing chess or compose a

musical piece. However, it was also stated that the engine is only capable of mimic what has been coded, within the boundaries specified by the programmer. This explanation forecasts today's debates on AI (Russel et, al. 1995).

The first computer was designed by Alan Turing in 1940 in order to solve the enigma code, which was the way by which the Germans communicated during World War II. Other computers named Colossus (1943), Z-3 (1941), ABC (1942) have been further designed to meet daily needs. On the other hand, a distinctly unique approach on computational computers was developed. Artificial neural networks developed by Warren McCulloch and Walter Pitts opposed computational computers stating that computers are able to search through neural network which is also known as the first AI research in history (1943). However, in 1951, Marvin Minsky and Dean Edmonds were the first ones to apply artificial neural networks and find out that the technology at hand was not sufficient to help neural networks to compete with computational computers.

In 1950, Alan Turing reports that he rejects AI that is being developed by computers (Turing, 1995)**.** Referring back to Babbage's work, he states that AI that has been developed on computers are only capable of being fast by means of electric. However, he states that human mind also consists of chemical reactions between neurons which are absent in the case of AI that is provided by computational computers. He agrees with Lovelance and states that AI is only capable of mimic human actions within the boarders defined by humans. He believes that the way in which AI could develop is through computer that have the capability to learn instead of think.

By IBM's trade performance in 1950s, the number of programmable computation experiments increased. The programmers were mostly interested in computers that could mimic human strategies by playing games such as chess and that have the ability to talk and recognize shapes and patterns.

On the contrary to what Turing and Lovelance's suggest, in 1957, Herbert Simon suggested that computers that can think, learn and create will be achieved. He proposed that computer will be able to achieve whatever a human professional can. In 1959 Arthur Samuel developed a program that could learn by the moves made in a chess game. With this program he established techniques on a computer that learns from defined rules in opposition to neural networks. Although this approach

could be applied to some cases, it was still limited and could not help achieving AI's intentions. In 1961, Newell, Simon and Sawn developed the General Problem Solver which simulated human approach and strategies while solving certain logic problems. This program worked in general problem but it was not sufficient enough for cases that needed professional knowledge. Therefore, the program could not be used widely. Hubert Dreyfus on the other hand, thought that Simon's approach underestimated human intelligence. He stated that human intelligence is complicated at a level that we still can not understand even with scientific methods in cognitive science. Human intelligence can not only be represented through problem solving strategies.

AI developments are still in progress. Prior experiments were defined as limited. New approaches are being developed aiming at a solution that is not limited. One of those researches roots back to 1940s: neural networks. Minsky developed the first artificial neural network in 1951 and in 1969 stated with Papert that artificial neural networks were also limited in learning. Most probably this limitation could be due to technical capabilities of the time and early stages of research therefore simplicity of the networks. After 1980's, more complicated networks have been developed and got the hopes up for improved AI. Another approach to AI is through natural evolution. Genetic algorithms were first put forward by John Holland in 1970s. Natural evolution's simulation is coded to computers allowing for programs that can keep growing by themselves. The latest AI research puts intelligent agent approach forward. In this approach, visual and audio capabilities are being coded into computers. Computers are expected to sense their environments, response to their environments while learning from their environments.

Although AI is not sufficient enough to meet today's expectations of it, AI helped some other very useful category to be established: knowledge based systems. As mentioned before, general problem solver was not sufficient for specific problems that required professional knowledge. Since general problem solver was poor in solving problems in its application field, it was referred to run with weak methods. Knowledge based systems then started to develop to compensate for General Problem Solver's weakness.

Knowledge based systems require that knowledge is represented on the computer and that new methods for knowledge based reasoning is developed. Edward Feigenbaum proposed an expert system which would operate like a professional of the field and put forward a human performance. After its success in the 1980s, a whole new market launched (Russel & Norvig, 1995).

It would be untrue to state that knowledge based systems support the improvement of AI. On the contrary, knowledge based systems therefore expert systems initiated a whole new category diverting from the targets of AI. Knowledge based systems should be studied in itself as a whole entity.

2.3 Shape Grammar

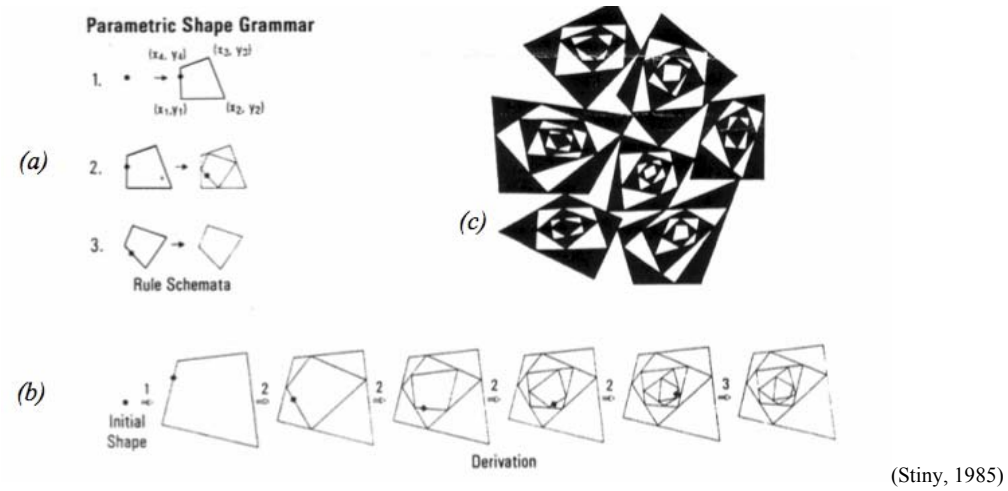
Although expert systems initiated a new market and has been in trend since 1980s, when examined from a specific profession's view of stand like design, expert systems can not provide the designer with a certain and true solution. Expert systems can only help as an assistant at certain steps of the design process. The reason for expert systems in design not being as efficient as they are in other professions is the complicated nature of design and design problems. The major handicap of design is that it is relative. Both the problems leading to the solution and the alternatives studied towards the solution and the solution itself could be considered relatively different. Design problems do not carry certainty and can not be understood at once. Usually the designer starts from a point and finds out about different aspects of the problem he is dealing with as he experiments and evaluates it. Defining a design problem is complicated. A design problem carries a high potential to be under evaluated or mis-defined. Therefore they are referred to as *ill defined problems.* Since dealing with all aspects of the problem at the same time with a general strategy does not ease the situation, various approaches to design problems have been developed with time.

One of the major approaches to design development is shape grammars. Shape Grammar was introduced by George Stiny and James Gips in 1972. They were published in Environment and Planning B in 1976. The paper was called "Two Exercises in Formal Composition". The paper focused mainly on formal spatial concerns and it consisted of two exercises. The first exercise "consists of exploring the possibilities for arranging certain spatial elements in some ordered way." The

second exercise " consists of beginning with a given arrangement or arrangements of certain spatial elements and constructing or identifying additional arrangements of these elements that are in the same style."

Two types of grammars were introduced by Stiny: standard or non parametric and parametric shape grammars. In both cases, a set of rules are applied and an initial shape is transformed to derive new shapes. In the rules to be applied, the shape on the left hand side of the arrow determines the shape or the part of the shape that will be replaced or transformed. The shape on the right hand side of the arrow presents the state of the shape after the rule has been applied. In the following figure, a shows the rules for "a" non parametric shape grammar, "b" shows derivation of rules and "c" shows the end result that is generated by applying the rules repeatedly (Stiny, 1985).

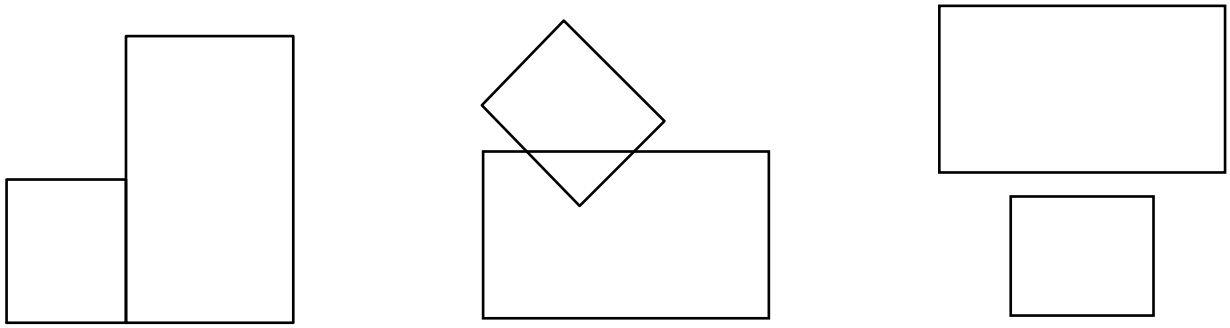
On the other hand, a parametric shape grammar consists of other parameters where the values can be adjusted. In the following figure, rules for "a" parametric shape grammar, "b" shows derivation of rules and "c" shows the end result that is generated by applying the rules repeatedly (Stiny, 1985).



(Stiny, 1985)

Figure 2.1: a) Rules of a Parametric Shape Grammar b) Derivation of Rules c) Result Generated by Application of Rules

Applying shape grammars takes various steps such as definition of shapes, definition of spatial relations, definition of rules, definition of shape grammars and production of design alternatives. Shapes are defined as the basic components of grammars and designs. They show variations from a two dimensional straight line –or a curly line- to three dimensional prisms. Spatial relation defines the arrangement of selected shapes. Below, three alternative spatial relations are presented for a selected set of shapes that consists of one larger and one smaller rectangle. As shown in the figure, the shapes can stand next to each other aligning on a shared edge, they can intersect each other or stand apart with a certain distance.



www.mit.edu/~4.184/lecture1_terry/lecture1slides_final.PPT - 2001-02-02

Figure 2.2: Spatial Relations

The rules define the circumstances under which the spatial relations between the shapes are applied. These operations are defined by Boolean operations. Boolean operations are derived from Boolean Algebra developed in 1854 by George Boole with his book called the Laws of Thought. These operations are simply translated into the modeling language as union, intersection and difference operations. Below, is an example where the shape set, spatial relation, the rules and the design alternatives are shown together.

Shapes: A , B
Spatial Relation: A + B
Rules: A --- A+ B
B --- A+ B

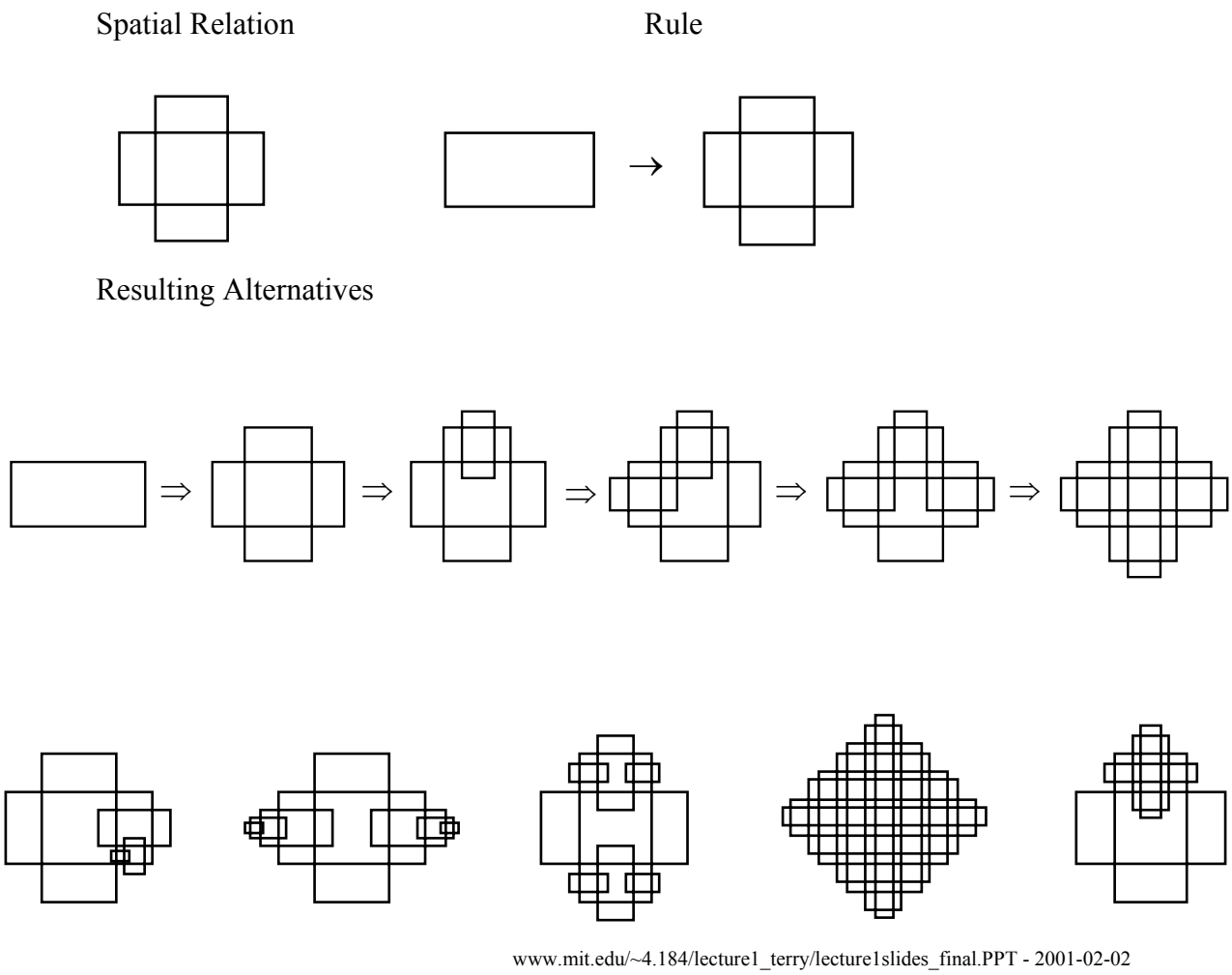
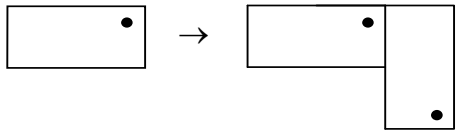


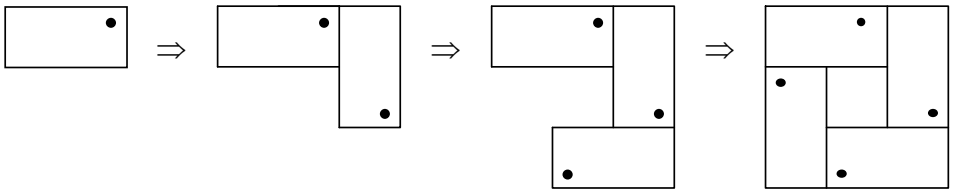
Figure 2.3: Initial shape set, spatial relation, rule and the results of application of the rule on the shape

As mentioned previously, shape grammars have expanded to include labels as well. A label is a symbol placed on the shape and it tells the orientation of the shape as well as how the rule will be applied. The following figures present possible outcomes of a simple labeled rule.

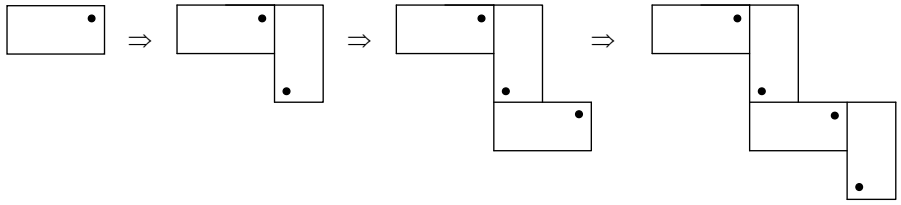
The Labeled Rule



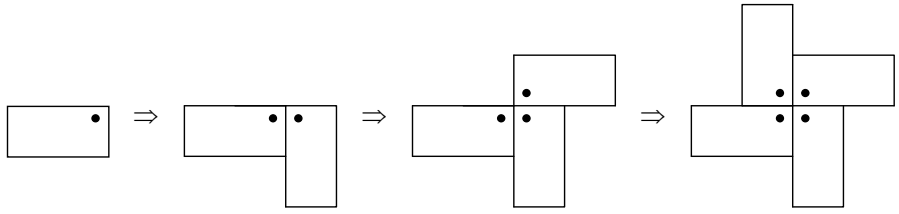
Derivation 1



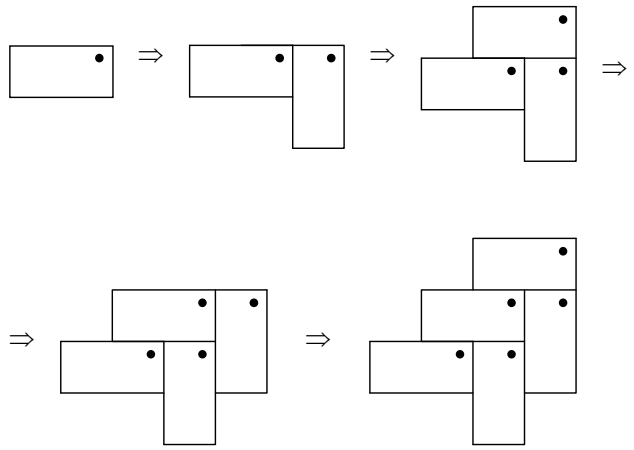
Derivation 2



Derivation 3



Derivation 4



www.mit.edu/~4.184/lecture1_terry/lecture1slides_final.PPT - 2001-02-02

Figure 2.4: Labelled Rule and its Derivations

Beginning with the introduction of the idea of shape grammars, many grammars generating various architectural style have been developed such as Palladian Grammar (Stiny and Mitchell 1978), Mughul Gardens (Stiny and Mitchell 1980), Prairie Houses in the style of Frank Lloyd Wright (Koning and Eizenberg, 1981), Greek Meander Patterns (Knight, 1986), and Queen Anne Houses (Flemming, 1987). Such studies done by local academicians are A Shape Grammar: The Language of Turkish Houses by Gülen Çağdaş, A Shape Grammar Algorithm and Educational Software to Analyze Classiz Ottoman Mosques by Sinan Mert Şener, Emine Görgül and Design by Grammar: An İnterpretation and Generation of Vernacular Hayat Houses in a Contemporary Context by Birgül Çolakoğlu.

Having first introduced the idea of shape grammars in 1972, the idea of computational shape grammars took three years to arrive, in 1975. Stiny presented formal definitions and algorithms for the system. In time, shape grammar system expanded and branched out to include labels, parameters and weights. Eventually, these expansions evolved into shape grammar interpreters. However sophisticated or useful in theory computational shape grammars are, they have not been useful for non-programmers. This issue was addressed by Tapia (1996;1998). Tapia consists of a simple visual interface and act as a general two-dimensional shape grammar interpreter (Knignt, T. W. 1998). The reason why shape grammar is explained in this section is that many urban morphology generating software tools

which will be explained in detail embed shape grammars and it is necessary to understand the essence of shape grammars in order to understand the field work to be explained in the following chapters.

2.4 Design Necessity

2.4.1 City Scale

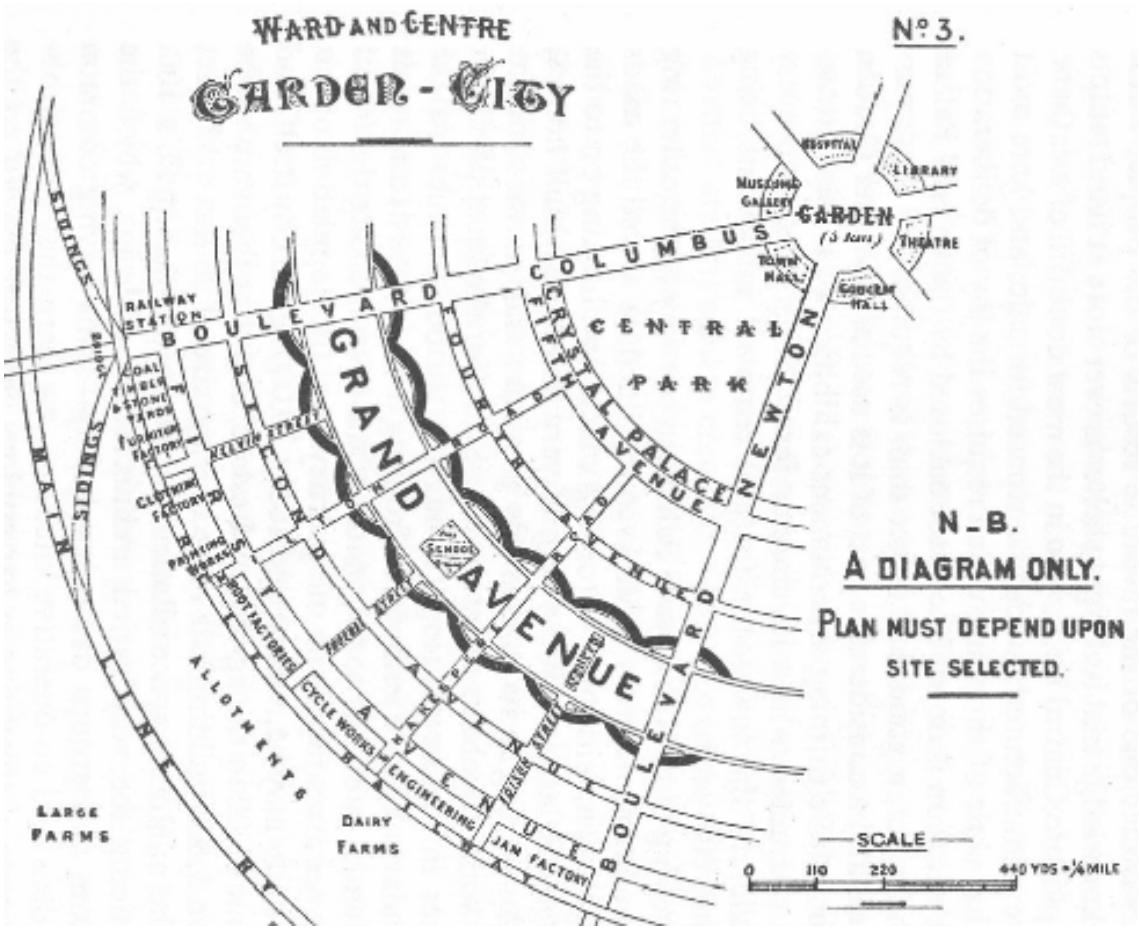
“Reform the environment, stop trying to reform the people. They will reform themselves if the environment is right.” Buckminister Fuller had stated in 1960 to underline the importance and effect of built environment on its inhabitants.

In mid 20th century, the belief supported that a city should be organized, ordered and planned in such a way that the life quality of its inhabitants is improved by the organization of the physical environment. The strongest implementation of such a planned city would be Paris. From 1853 to 1870, George- Eugene Haussmann worked eagerly to re-shape Paris. Haussmann destroyed 12000 structures, demolished 333 miles of meandering old roadways and replaced them with 85 miles of new direct roadways which cut through the city. He had landmarks erected in the city such as the Opera and the Halles Centrales. He also managed to improve the unhealthy water and sewage system and security on the streets by increasing the number of streetlights. Parks equivalent of the ones in London and New York had been established as well. However, the most significant feature of Haussmann’s planning had been wide boulevards which were mostly angled cutting through the streets and linked at diagonals. Angled boulevards had been implemented to serve several functions. They allowed for a fast deployment of troop in case of revolts, evacuated the troublesome neighborhoods and connected the city’s new train terminals more effectively. The boulevards also improved transformation of goods within the city since beginning with the industrial age, the economy had shifted from small artisans working close by their homes to industrialized system which had spread widely in the city. The final feature of the new city plan was that it had created arrondisments so that the city could be evaluated as rings which grew outwards creating a matrix (Fillller, M. 1991).

http://www.idealcity.org.au/pic-1b-paris.html

Figure 2.5 : Radial planning evident with Arc de Triomphe.

On the other hand, after the industrial revolution, cities became so dense that new ideas for ideal cities started to initiate. As the pace of the city increased as well as pollution, inhabitants longed for peaceful environments for their families. The cities started to be decentralized. People preferred living on the outskirts of the city and commute to work in the city. Such cities had been named as edge or garden cities. Garden Cities offered a solution to city's problems through decentralizing the city. Ebenzer Howard, who developed the idea of Garden City, envisioned cities similar to English villages with addition of railroads and small scale industry. The basic unit of these cities are families living in their own houses. The houses were spread on planted streets which converged together towards the center. (Curtis. W. J. R. 1996) Garden City consisted of cemeteries, stone quarries, reservoirs and water falls, on the outer most ring of the city, the municipal canal and the land which are given to the inhabitants by the government to grow plants and vegetables on the next inner ring, inter municipal railway, large farms and green areas in the next inner ring and finally the central city was located. The reason for such a new idea of a city had been well explained by Howard: " clean streets with free countryside all around, a belt of fine gardens and orchards, so that from every point in the city one can reach pure air, the grass and the distant horizon."



http://www.idealcity.org.au/town_planning-4-garden_city.html

last visited: 01.04.2009

Figure 2.6 : Howard's Garden City model.

On the contrary, such organized and well planned cities, over the past 20 years, have been manifested. The researches have focused on "the fact that simple systems were manifested with a level of complexity that was completely unknown, went some way to explain why more complex systems, which were often built from such simpler elements, were entirely unpredictable, in fact even chaotic." (Batty, M. Longely, P. 1994). The weather, cities and stock exchange are only a few of such chaotic examples. Traditional assumption is that systems are linear. They are expected to change in a gradual way which makes them predictable. However organized were the planning of the garden cities, their emergence could not have been predicted (Batty, M. Longely, P. 1994). This is the reason in fact, systems are actually not as simple as they are described. They show discontinuous behavior and this is why emergence of garden cities could not be predicted.

This study will address the issue of city design by offering yet another approach by recommending to start designing from the most basic unit of the city, urban block. It will suggest a bottom up solution rather than a top down solution where how people will live is dictated.

2.4.2 Site

The scale of this thesis is bounded with the scale of site since site is perceived as the building block of a city. A city's character could be established by iterations and multiple applications of a well developed site approach.

Kevin Lynch states in his book Site Planning that "*every site, natural or man-made, is to some degree unique, a connected web of things and activities. That web imposes limitations and offers possibilities. Any plan, however radical, maintains some continuity with the preexisting locale. Understanding a locality demands time and effort. The skilled site planner suffers a constant anxiety about the 'spirit of the place'*."

In this thesis, the matter of continuity is delved into, through continuous visual harmony by using reflected images of the neighborhood as a method. Reflection images used although does not aim at the absolute aesthetic solution, it aims at establishing a visual harmony on the application site. By visual harmony, a consistency in physical form of architecture is sought after. This method is not followed to prove that whatever exists represents true aesthetics or true architecture, therefore it should be mimicked. This method is applied to establish consistency of visual form and act as a manifesto, when applied to most sites in a given city, which underlines the fact that appreciation of aesthetic values is degrading.

Kevin Lynch states that site design is a learning process in which a coherent system of form, client, program and site gradually emerges. The learning process which afterwards leads to a concept design proposal is initiated by the site. Site brings with itself under estimated requirements. These requirements need to be addressed if a new design that is in harmony with its neighborhood is desired. They need to be well blended with the needs of the program as well to meet the requirements of the client.

Every site is stated to have "genus loci" referring to the spirit of the site including its feel, historic background and contemporary profile referring to its social status of

inhabitants and characteristics of the program held within such as industrial, public, residential etc. Site plan regulates activities and the objects that will take place on the site. Site is supposed to aim at ethic and aesthetic goals to establish environmental consciousness and improve its inhabitants quality of life (Lynch, K. 2000).

It is very important to underline the fact that this thesis does not suggest that the method of using reflected images is the only proposed way to meet site or program requirements or improving the life quality of its inhabitants. This method is solely used to underline the fact that it is important to respond to surroundings and take it into consideration.

2.4.3 Symmetry

Symmetry is described by the Oxford Dictionary as " a geometrical or other regularity that is possessed by a mathematical object and is characterized by the operations that leave the object invariant." The word has its roots in Latin, *symmetria*, and it is described as "agreement in dimension, proportion and arrangement."

Symmetry is easy to explain through everyday life examples as well. The human body consists of an important level of symmetry. When cut down from the middle, the two halves would be symmetric. The internal organs within a human body such as lungs and kidneys are symmetric. Symmetry is also found in nature. Plants reveal substantial level of symmetry as well.

Today, mathematicians have elaborated on symmetry so much that it is now possible to point out several types of symmetry : reflectional symmetry, rotational symmetry, translational symmetry. The kind of symmetry will be taken into consideration within this thesis will be reflectional symmetry which is also referred to as mirror image symmetry. It means that will be viewed as the same if viewed with respect to its mirror image. Simply put, an image is flipped around an imaginary line which is called the axis and copied on the other side of the axis to form the mirror image. In this thesis, symmetry will be explained further from an architectural point of view since the reason why symmetry has been chosen as a mean to create harmony is architectural, not mathematical.

Types of symmetry in terms of architecture is grouped into nine categories by Kelli Nipper and Shannon Umberger: Bilateral Symmetry, Rotational and Reflectional Symmetry, Cylindrical Symmetry, Chiral Symmetry, Similarity Symmetry, Helical Symmetry, Translational Symmetry, Spherical Symmetry and Combinations of Symmetry.

Bilateral Symmetry is stated as the most common type of symmetry in architecture. In bilateral symmetry, there is only one vertical reflection axis through the middle of the architectural element. The following figure shows bilateral symmetry in Eiffel Tower.

Figure 2.7 : Eiffel Tower.

Rotational and Reflectional Symmetry, there is a center point which is the point of rotation and the point where reflected images intersect. The figure below, shows an example to this kind of symmetry with the Pentagon located in Washington D.C

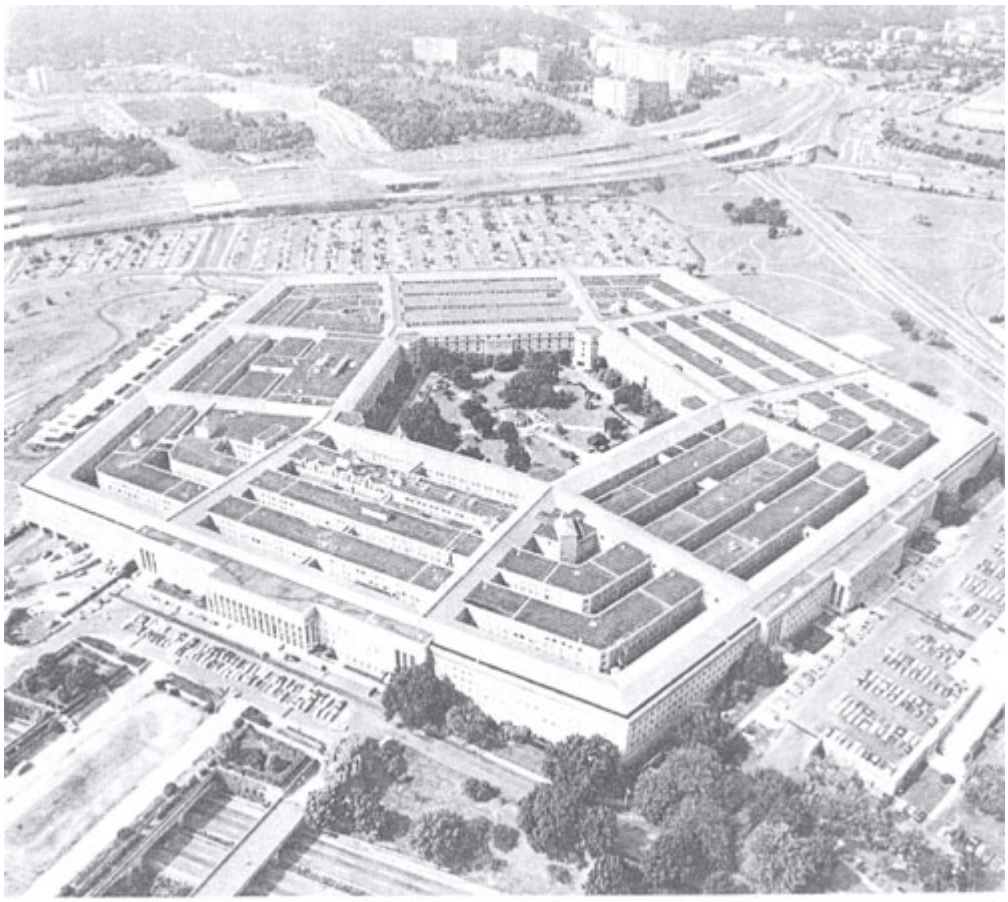


Figure 2.8 : Pentagon.

Another type of symmetry worth showing an example of is Similarity Symmetry. It's established by scaling the same shape by a scale factor so that the shape remains the same but only changes in size. The following figure shows this kind of symmetry with Sydney Opera House and it should be noticed that Similarity Symmetry helps establishing a unity of composition.

Figure 2.9 : Sydney Opera House.

The last type of symmetry that will be exemplified is Chiral Symmetry also in which the type of symmetry used in this thesis falls. In Chiral Symmetry a mirror is established between two separate objects. This move emphasizes the space between the objects. Although the intention is not to emphasize the space in between the objects in the case of symmetry used in this thesis, it corresponds to this category operation wise. The following image of Piazza of Saint Peter's in Italy, emphasizes a horizontal axis between two structures.

Figure 2.10: Piazza of Saint Peter's.

Rest of the types of symmetry mentioned consists of examples such as United States Capitol Building for Cylindrical Symmetry, Guggenheim Museum for Spiral or Helical Symmetry, Lloyds Building for Translational Symmetry, Newton's Cenotaph for Spherical Symmetry and Taj Mahal for Combinations of Symmetry.

Importance given to symmetry will be explained further through Palladio's words. The first quote that follows, does not directly mention symmetry, importance of unity and wholeness is implied. *"And although variety and things new may please every one, yet they ought not to be done contrary to the percepts of art, and contrary to that which reason dictates; whence one sees, that although the ancients did vary, yet they never departed from the universal and necessary rules of art, as shall be seen in my book of antiquities"*

The following quote speaks about the symmetry of the rooms in a Palladian Villa. *"The rooms ought to be distributed on each site of the entry and hall; and it is to be observed, that those on the right correspond to those on the left, that so the fabrick*

may be the same in one place as in the other." It is clearly expressed that the unity of fabric is important. It is proposed that through unity of fabric, a unity in visual perception would be achieved.

In the next and final quote, symmetry is proposed to be used to achieve visual continuity as well as improved level of comfort in climate wise*." The windows on the right had ought to correspond to those on the left, and those above directly over them that are below; and the doors likewise ought to be directly over one another, that the void may be over the void, and the solid upon the solid, and all face one another, so that standing at one end of the house one may see the other, which affords both beauty and cool air in summer, besides other conveniences."*

Following Palladio's approach to symmetry, this thesis aims at establishing a visual harmony and unity through the use of symmetry. The following figures represent a conceptual model of the way in which symmetry is used in this thesis. The model establishes reflectional symmetry by surrounding the site with a mirror which mimics the modeled facades across the street.

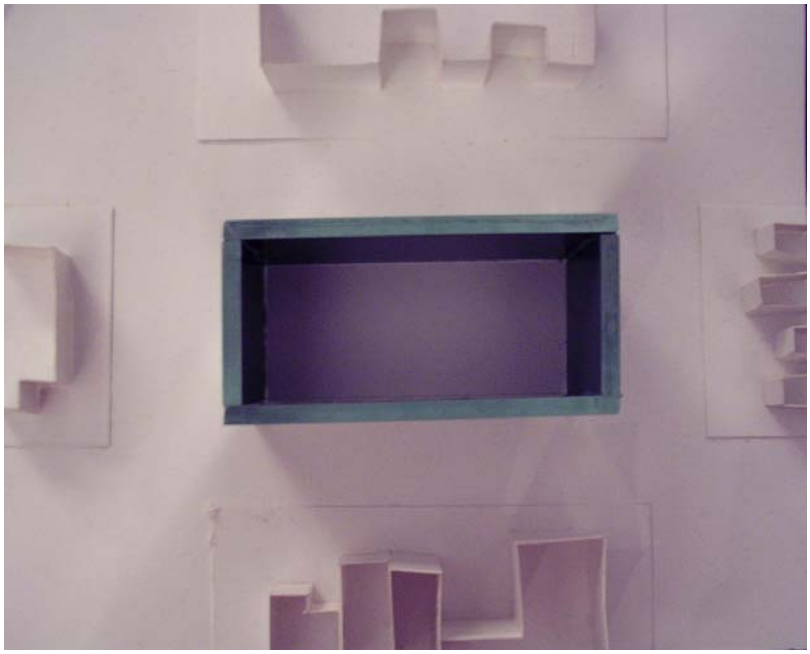


Figure 2.11 : Top view of a model showing conceptual ideas of symmetry applied to an urban block.

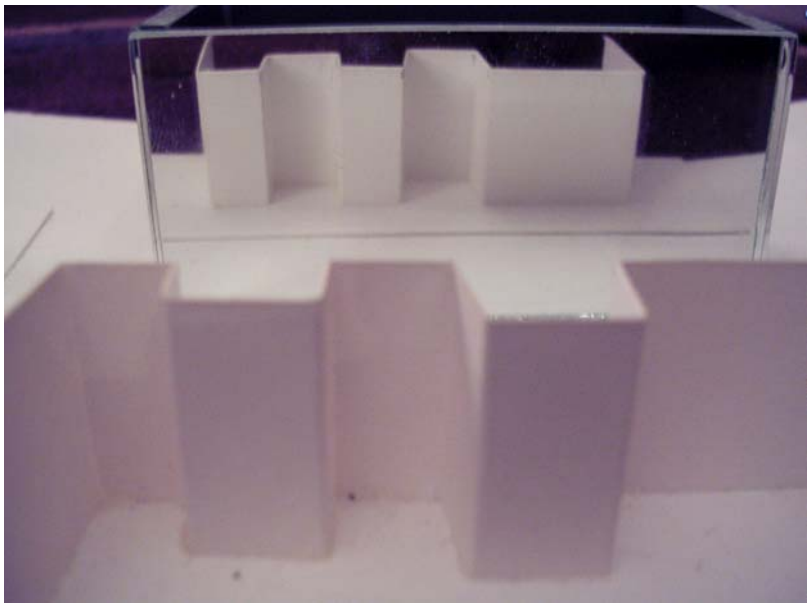


Figure 2.12 : Side view of a model showing conceptual ideas of symmetry applied to an urban block.

2.5 Conclusion

Attempts to improve quailty of life have been made at city scale by numerous planners. In this thesis, it is advocated that the character of a city should not be imposed with a grand plan but rather, should be adopted starting with the simplest urban unit, a block, which is reffered to as the building site.

On the other hand, symmetry has been used by architects to achieve harmony in structures. Several types of symmetry used by architects and implications has been presented in this chapter.

In the following chapters, a plug-in which generates site design alternatives in response to its built environment using symmetry has been explained in detail. The proposed plug-in flow as well as two different implications of the proposal will be presented with results obtained.

3. KNOWLEDGE BASED SYSTEM IMPLICATIONS

3.1 Implications in General

In this section, knowledge based systems that have been discussed previously will be supported with applied examples to state the fact that knowledge based systems are applicable to a variety of scales at a variety of design phases from early phases until the end product therefore; it is a suitable pick to work at urban scale at an early design phase as well.

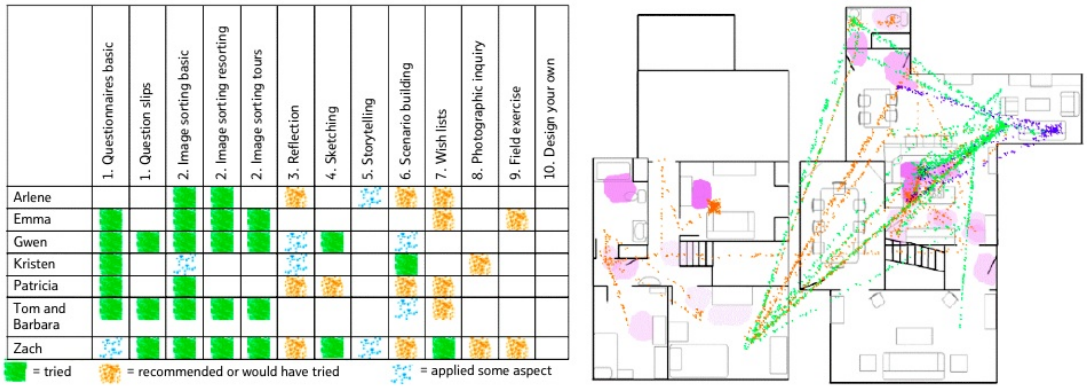
3.2 Open Source Building, OSB

OSB, originally Open Source Building, has been developed in MIT Laboratory. The project aims at developing new design strategies applied onto a prototype, House_n Project. The major goal of the project in long term is to alter how a house is built in the next 10 to 15 years time.

Open Source Building initiates a new approach to design using affordable computation, internet, wireless communication and production. This new design tool offer opportunities for a non designer to be able to design his own house without needing an architect. Open Source Building achieves it through the use of a knowledge based system running in the background.

The new design tool is designed to store the architect's values and knowledge which will support the non- designer with complicated design problems. Open source Building works in four stages: Preference engine, design engine, design

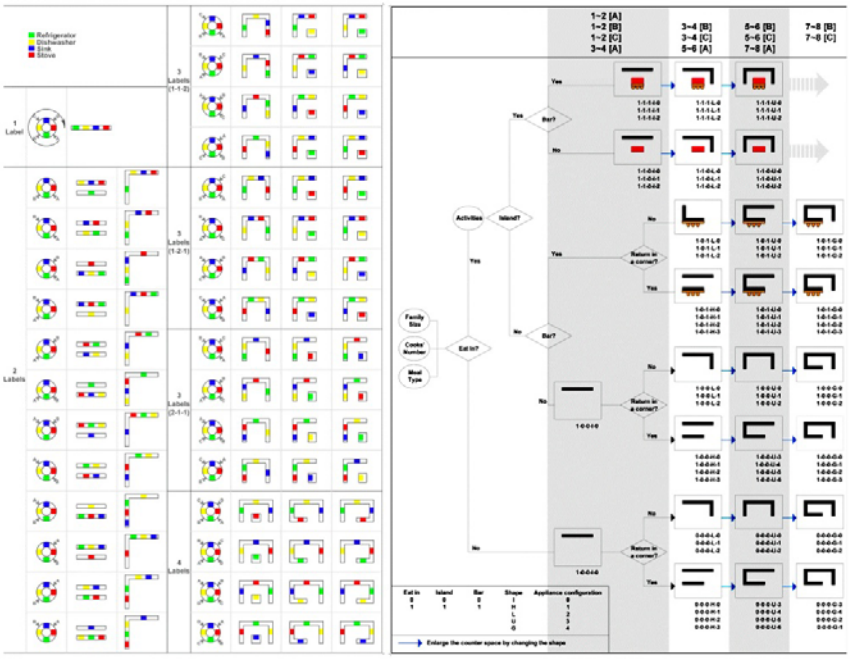
The preference engine consists of specific questions in order to understand the needs of a client. Some of the questions include ones such as " do you like to cook?" or "how many individuals live in your home?". As a result of these set of questions, the tool builds a user profile including family size, budget, aesthetic values and range of activities (Larson, K. et.al 2004). On the other hand, the system is designed to avoid stereotype answers through the usage of sensors placed in the furniture. The sensors generate the activity pattern of the household. The generated activity pattern is to be used to generate an initial plan in the design engine.



(Larson , K. et.al)

Figure 3.1 : Left:Results of the questionare. Right: Diagramatic representation of a family's movements.

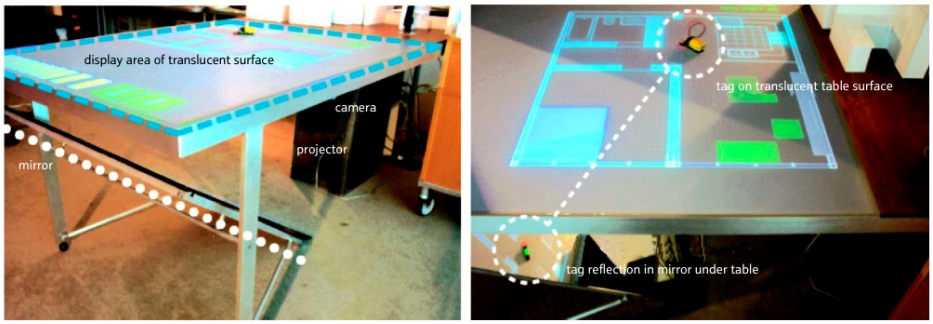
The design engine builds on Siza's system which had been developed in 1970s by. Duarte developed a system based on mentioned work of Alvaro Siza. Siza had aimed to increase user participation in design process of mass housing at Malagueira. Siza applied implicit rules to generate house plans in an effort to accommodate possible custom needs as well. The application of the rules allowed for 35 different plan layouts to be generated. Duarte has translated Siza's rules computationally into a shape grammar. Xiaoyi Ma who is a member of the House _n team, had created a program building up from Duarte's work. The program that Ma created generates kitchen designs *using an exhaustive database of design typologies for kitchen room shapes, functional arrangements, appliance configuration as well as the parametric rules for their configuration* (Larson, K. et. al, 2004). Another typology was created to include family profiles, eating style and cooking patterns. An initial plan is generated by an algorithm which intersects the results of the preference engine and architectural context constraints and universal guidelines.



(Larson , K. et.al)

Figure 3.2 Kitchen types and transformations.

With the completion of the initial plan generation phase, Open Source Building aims to provide the client with an opportunity to costumize the plan. A digital table which has visual LED tags embedded is used to accomplish this task. The customer is asked to move around the infill of the module according to his personal will while the digital plan is constantly being updated as the customer makes changes on the plan.



(Larson , K. et.al)

Figure 3.3 Digital Table.

A computational critic system has been developed for the project by Reid Williams. The algorithm makes the system learn by example instead of rules. An architect trains the system by rating various floor plans and the algorithm encodes the biases used by the architect to teach the critic. The customer aiming to

customize an initial plan is asked to select an architectural critic. As the customer makes his moves on the plan, the critic rates the updated plan and the last move made as “acceptable” , “unacceptable” or “ unrated”. For the moves which received the command “unacceptable” the critic also gives a brief explanation of how to fix the problem.

As a result, Open Source Building is capable of leading a client to design her own house with the knowledge and the database it owns.

3.3 The Architect’s Collaborator, TAC

The Architect’s Collaborator developed by Kimberle Koile at the Massachusetts Institute of Technology in 2001. TAC is as a design support system which aims at realizing experiential qualities as physical forms represented by the computer.

Koile, intends to represent qualities such as openness of privacy by *experiential qualities*. On the other hand, the phrase *physical form* stands for elements such as walls, windows and doors. TAC uses the physical form to establish the experiential quality that is requested by the architect and the customer.

The following paragraph is used to simulate a scenario which will show how the program is envisioned being used in Kolie’s thesis:

“*Imagine that you’re sketching a design, pen in hand. You tell the computer near you that the design is for a client who wants a house with main living spaces that feel open to one another, but private with respect to the front door. You have ideas about physical forms that manifest feelings of privacy and other that manifest feelings of openness, and you’re translating those ideas into lines and annotations on a page. You stop to assess your latest sketch and ask the computer for its comments. It shows you regions defined by your proposed physical forms. It shows you what is visible from each region and from the front door. You notice that a large portion of the living room is visible from the front door, so you add a screen; the computer shows you what is now visible with a screen in place. You also notice that the stair blocks the view of the dining room from the living room. You ask the computer to increase the visual openness of the dining room without sacrificing privacy with respect to the front door or ease of access to the stair. It suggests two*

other locations for the stair, showing you how the visual openness, privacy and access are affected by each location. It shows you several other possible stair locations, but points out that in these cases, while visual openness increases, privacy of the dining room decreases, and the stair is not as easily accessed. You like one of the first two proposed locations, accept it, and continues sketching and consulting with your computer."

TAC is capable of performing the necessary actions to fulfill the requirements of the scenario above through the following steps (Kolie, K.. 2001) :

1 displays regions defined by the physical form.

2 computes and displays visual openness measurements for all regions

3 computes and displays visual openness measurements for the living region from the front door, with and without the screen

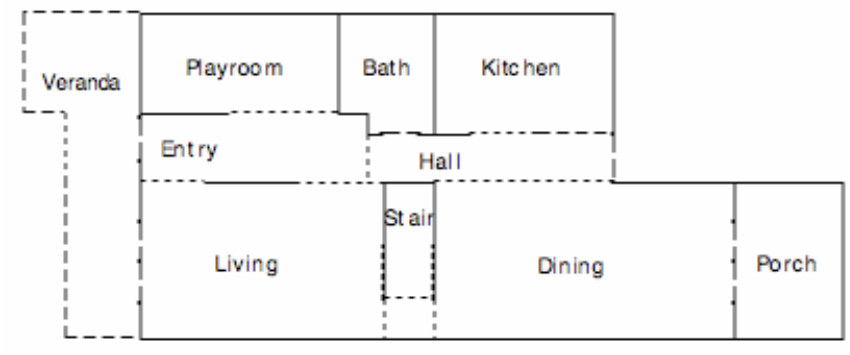
4 proposes repair suggestions for increasing visual openness between dining and living

5 creates new design by carrying out repair suggestions

6 evaluates all goals to check for solutions

7 displays rejected designs.

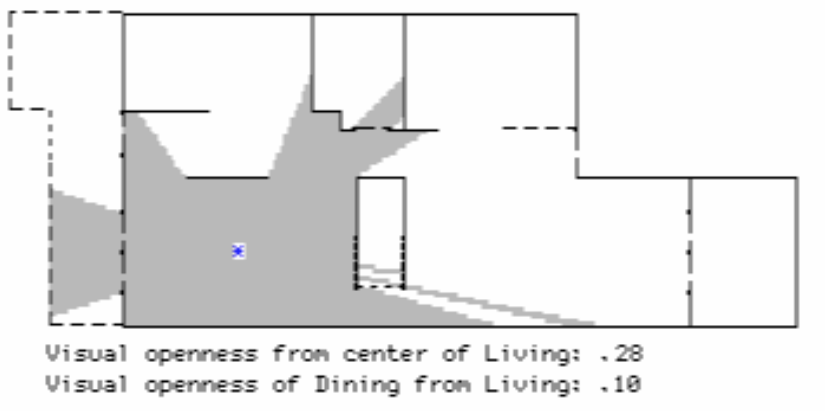
The process is initiated with TAC displaying region names on the floor plan. In this stage the floor plan is expressed in a very symbolic manner with basic representations. All that is needed to be expressed are the voids and the solids in the design. Therefore, wall thickness' are not taken into consideration and the windows are not drawn and only represented by voids and dashed lines.



(Kolie , K., 2001)

Figure 3.4 TAC Computes and displays the regions created by the physical form.

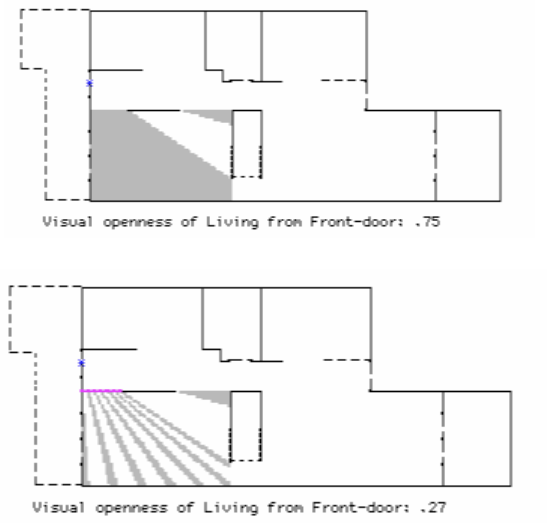
The next step that TAC is supposed to take is to calculate visual openness measurements from the view point that the designer specifies which is represented by a star in the drawing. TAC hatches the visual parts of the design from the specified view point as well as presenting the designer a calculated numeric value.



(Kolie , K., 2001)

Figure 3.5 Area that is visible from the living room.

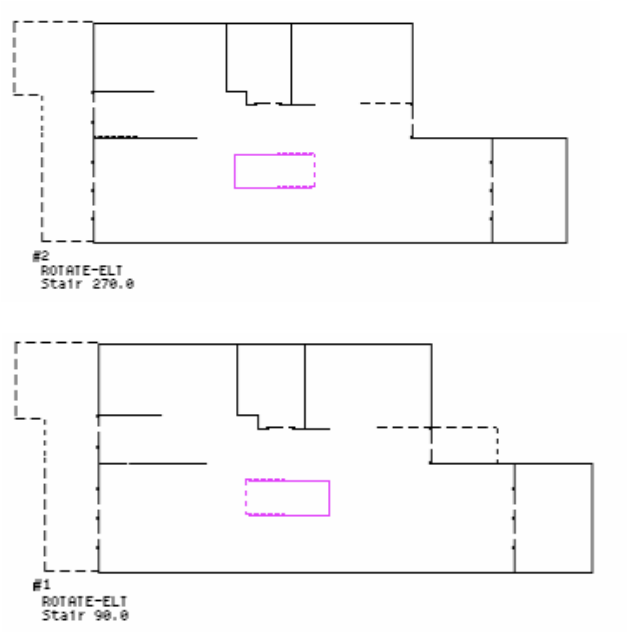
The third step takes measurements from the front door towards the living room both with a screen and without a screen since a large portion of the living room is seen from the front door and the designer prefers reducing visibility to increase privacy.



(Kolie , K., 2001)

Figure 3.6 Top: Visibility from the front door without a screen. Bottom: visibility from the front door with a screen.

TAC applies the fourth step to increase visual openness of dining room from living room which is obscured by the stairs. To achieve this, the program proposes repair moves. The goal of this step is to keep the stairs easily accessible and to keep the privacy states of the dining room and the living room with respect to the front door but maintaining a visual openness between living room and dining room. TAC suggests to rotate the stairs for 90 or 270 degrees or to move the stairs to any of six edges. Then, TAC creates eight new designs. TAC only proposes two of them as new design and stores other six alternatives as rejected designs since they do not meet the goals of the designer.

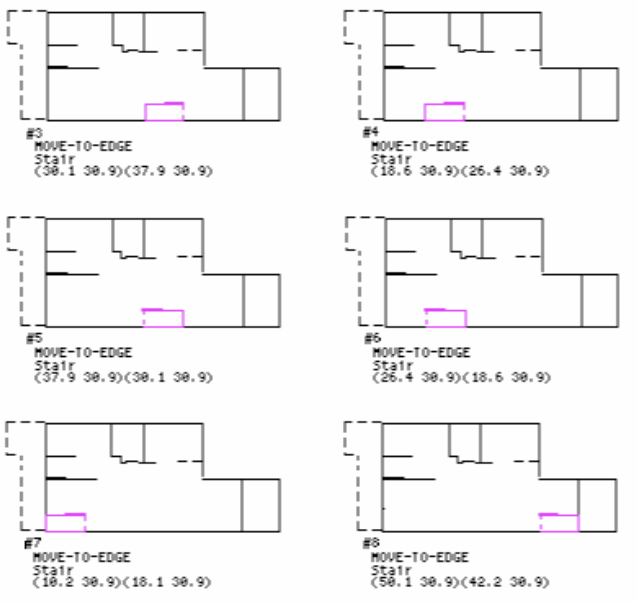


(Kolie , K., 2001)

Figure 3.7 Two alternatives for increasing visual openness of dining room from living room by rotating the stairs.

In the next step, TAC evaluates all goals that have been calculated and achieved previous to rotation of the stairs and approves that rotating the stair not also meets the goals of the design but also improves visual openness measurements as well.

TAC is capable of storing and presenting the rejected designs for the review of the designer



(Kolie , K., 2001)

Figure 3.8 Rejected designs with the stairs on the edge.

The Architect's Collaborator has been introduced in a very general scope since the details of the program is beyond the scope and intentions of this thesis. The intention with introducing TAC is to present the reader with another expert system which aims to translate vague terms such as private and open into physical form and support its proposals with calculations.

3.4 Conclusion

Open Source Building and TAC has been introduced as examples of knowledge based systems their application fields and possibilities of such systems. Different scale of goals are picked consciously to convince the idea that knowledge based systems are applicable to any scale from designing a new home to making adjustments to design elements to achieve abstract goals. In the following chapters of this thesis, another knowledge based system that works at an urban scale will be presented in detail.

4. APPROACHES FOR NEW CITY DESIGN GENERATION

Computer Aided Design (CAD) tools are being ubiquously used for creating and modifying drawings on computers. Autodesk has introduced AutoCAD in 1982 and made it possible to work with CAD on personal computers.

On the other hand, Geographical Information System (GIS) is another computer software which links geographical information with descriptive information. ESRI has developed a leading GIS tool called ArcGIS which is an integrated collection allowing the user to analyze, map, manage geographical information.

Although CAD tools have been used widely they have only been used as a mean to draw rather than design. A designer needs information that a CAD tool can not provide. Therefore, a demand towards integration CAD and GIS software has increased. This demand has lead ESRI to develop ArcGIS for AutoCAD displays geographic reference in a drafting environment. (Larive, M. et, al. 2005)

Some other softwares regarding urban modeling are Virtual Reality (VR) and Virtualized Reality which uses different methods to generate reality based 3D city models. However none of them support designing of a city from scratch or understand the consequences of a change on the urban plan. Softwares that provide the user with tools to generate a new city and analyze it will be explained in this chapter along with theoretical approaches used to achieve such softwares. And another alternative to such softwares that generate new urban models is developed in form of a plug-in within this thesis and will be explained in detail in the next chapter.

The article named Automatic Generation of Urban Zones written by Mathieu Larive, Yann Dupuy and Veronique Gaildrant elaborates different approaches used during various stages of a virtual city generation. The article distinguishes the various steps of a virtual city generation as generation of Urban Zones, Road Networks, Blocks, Lots, Exteriors, Building Plan, Furnished Building. Explanation of every category involves at least two different approaches including City Engine which will be explained in detail in the following sections.

An urban zone is defined as limits of a city. This information is obtained through a geographical or a socio statistical data map. A geographical data may consist of information on altitude, hydrograph and vegetation. On the other hand, a socio statistical map may consist of population density, street patterns and elevation data maps (Larive, M. et, al. 2005).

A road network is explained in two approaches. One of the approaches is called the L-Systems. Parish and Müller developed a system and named it ideal successors. The parameters of these successors are not instantiated. They are instantiated when the global goal function calls them in order to satisfy its goals. Then the system checks to ensure that these parameters do not violate restrictions of the local goals. If it does violate any restrictions, the parameters are re-adjusted in accordance with local goals. The second approach is called Declarative Modeling of Line Segments. In this method, urban elements that can be combined to obtain more complex urban elements. This method is incremental. The user starts with a vague, approximate sketch and defines some main features. The system develops a proposal based on the sketch. Then the user refines the proposal (Larive, M. et, al. 2005).

A block is defined as a connected surface delimited by roads or streets in the article. The hierarchical organization starts with the division of the city into blocks. Blocks are represented with convex polygons so that advantages of the less complex geometric algorithm are taken. (Larive, M. et, al. 2005)

A lot is defined as the surface of the same mailing address. Lot is the surface on which buildings will be developed in the following steps. A Geometric Partition method and a Pattern Guided Partition method is explained. The Geometric Partition method works as a recursive process and it divides the bigger lot on the longer edge. The process terminates when the surface of the largest lot is under a user defined threshold. The Pattern Guided Partition method begins with a random point on a discretized surface on the block. The system then, searches for another point so that it can form a new segment which is in accordance with the defined pattern. (Larive, M. et, al. 2005)

Buildings are also taken into consideration within the hierarchy of the system elaborated in the article. The variables used to generate buildings are the shape, height and orientation of the building as well as the offset from the limit of the lot. The article discusses four approaches to building generation. In Shape Grammar approach, it is proposed that a library of architectural elements from shape grammars

are generated. Then, they are to be used to visualize assembly drawings with 3D scenes. The second approach described in the article is Declarative Modeling. It is based on a learning process classifying solutions so that it proposes the user characteristic solutions that are restricted in number. The third approach defined is Split Grammars. Split Grammars are based on concept of a shape and they are parametric. A second kind of grammar which is called the control grammar is used to check the propagation of the split grammar's attributes. The last approach explained in the article is Geometric Modeling. In this approach, AGETIM which is an integrated software tool which enables 3D environment generation with infrared, electromagnetic and acoustic characterization. The module allows the user to use predefined templates. This way, buildings created have variable heights and roof forms are automatically adapted to their environment in terms of style, orientation and texture (Larive, M. et, al. 2005).

In terms of building plan generation step, two sets of work are presented. In Maculet R. Archipels's work, building design is introduced as a process of constraint satisfaction. Spatial constrains have been configured through the study of representation of the spatial knowledge in architecture. The second work described in the article belongs to Charman P. The work is described as an extension that takes geometric specificity of a layout problem into account which uses a new filtering method called semi-geometrical arc-coherency (Larive, M. et, al. 2005).

There is no tool that takes all the seven steps into consideration. However, CityEngine which is a recent city generation software consists of the first five steps, from scratch to building exteriors and will be explained in detail in the next section. The authors of the article state that only some of the seven stages presented in the article have been studied extensively and the block generation and lot generation have been studied less. In this case it could be stated that this thesis addresses to exact step which needs more study in the hierarchical pipeline and offers a simple approach to generate an urban block.

4.1 City Engine

Procedural INC has developed The CityEngine as a city modeling software. The software is developed by a team that consists of Pascal Müller, Simon Schubiger,

Dominik Tarolli, Matthias Specht, Sefan Müller, Andreas Ulmer, Simon Haegler, Basil Weber, Jan Halatsch as well as other researchers such as Peter Wonka.

The CityEngine is capable of growing streets, manage land parcels, generate buildings, edit facades and use variation. Developers of the software state that the rule based modeling with the computer generated architectural shape grammar is a groundbreaking new technology; and that it enables an iterative and flexible workflow. The street grow tool that offers street patterns such as grid, circular or organic and also takes topography into account will be explained in detail in the following sections. Urban layouts are designed via street grow tool. A method called tensor action is used.

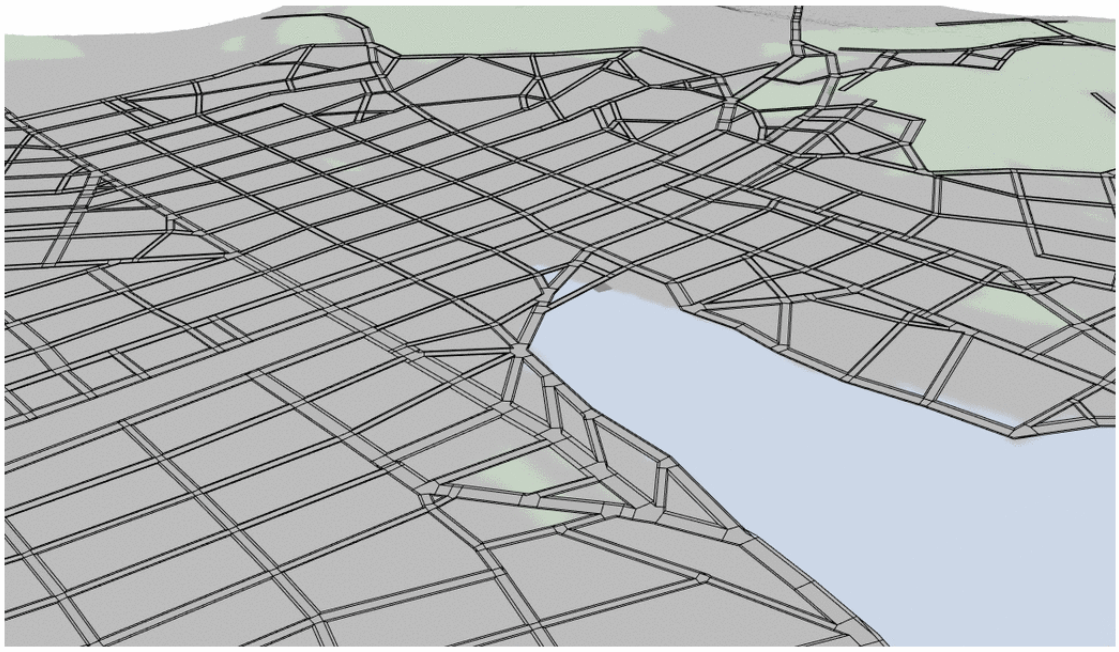


Figure 4.1 : Road Map

The developers also state that the world's first shape grammar implementation is the core of the CityEngine. The software provides the user with a scripting language which is specialized for architecture. Mass, elements, proportions, rhythms and materials can be varied through the use of simple language.

The following section will go through the process of how the CityEngine creates a street network, how it uses shape grammar rules to divide the plots and generate a single block which is then applied onto the plots followed by a variation step. The latter parts will focus on academic research behind the idea of procedural modeling

of cities and procedural modeling of buildings based on rules, shape grammars and user interaction.

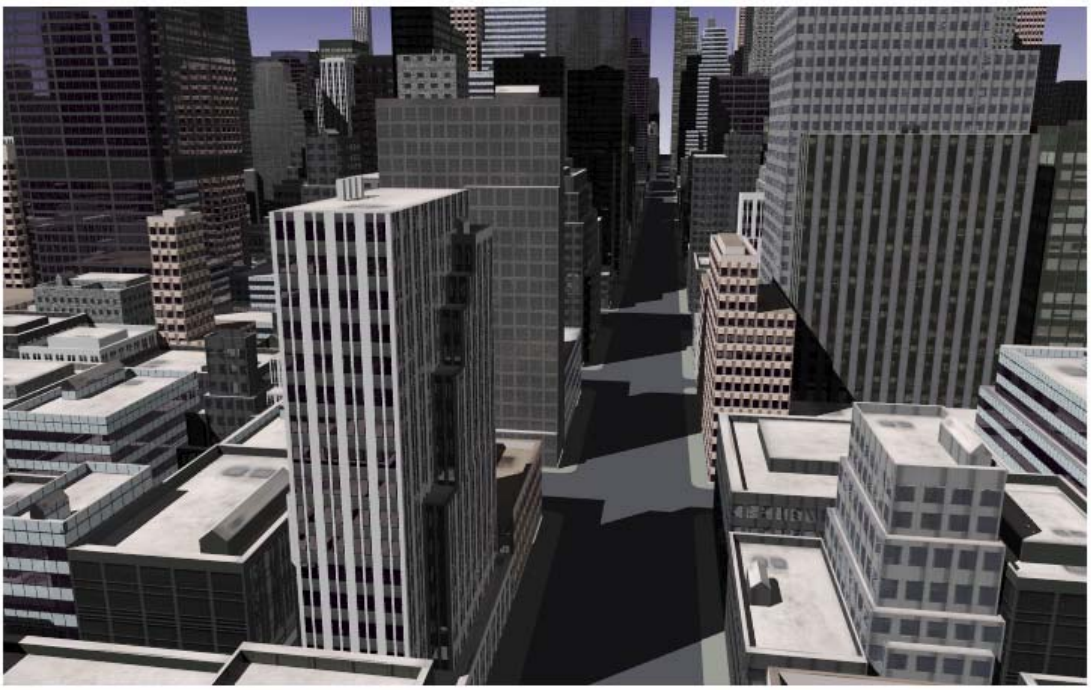


Figure 4.2 : A city generated by CityEngine

4.1.1Procedural Modelling of Cities

Pascal Müller and Yoav I H Parish has summarized the flow of the software in the article called Procedural Modeling of Cities. The article states that in order to create a virtual city, a road map and city building blocks need to be generated. The approach used to generate road maps is based on extension of L-Systems. To initiate the generation process, some sort of image input is needed. This input could be land-water boundaries or population density. Based on the input, the system defines constrains -since streets can not run on water- and generates a highway and street map dividing the space in between the streets into lots on which buildings will be placed. The software builds cities from scratch and the user can enter models manually or the rules can be extended according to his needs as well which makes the software very flexible in terms of user interaction allowing the user to make the decision to take the control or leave it to the software.

The generation of a city is simplified and reduced to generation of road networks and generation of buildings. Generation of road networks depend on image data provided

by the user and the data collected by the software developing group regarding to streets patterns of large cities such as New York, Paris, Tokyo and London.

After the user feeds the system with an image data, branching L-System generates the road network which consists of highways and streets. In the next step, the areas left in between the streets are divided into allotments on which the buildings will be placed. L-Systems are used again for the next step in order to generate solid building shapes. The final step of the system is a parser that visualizes the generated design. The following figure represents the flow of the software where the black boxes represents the result of the corresponding white box.

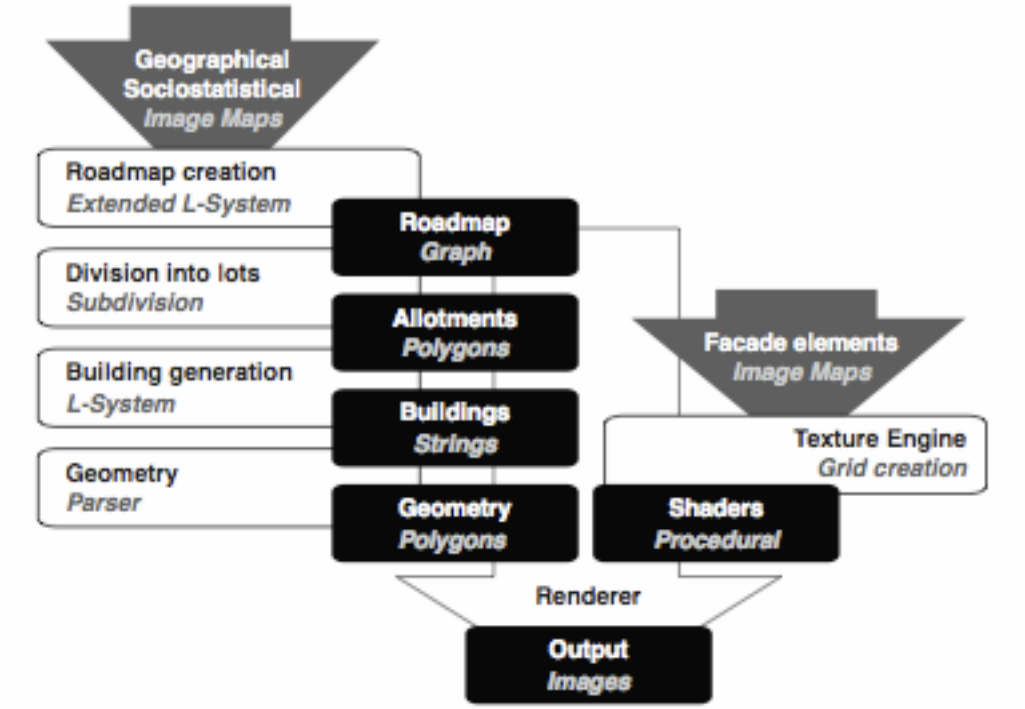


Figure 4.3 : Flow of software

The image input data which initiates the system and the behavior of the L-System generating the road network. Such images can be fed in as drawings done by the user or scanned in images and they can be categorized as either geographical or socio-statistical maps. Geographical maps consist of elevation and land, water or vegetation maps. Socio-statistical maps consist of population density, zone maps, street maps and height maps. In the figure below, water elevation and population density maps and a generated roadmap alternative is shown.

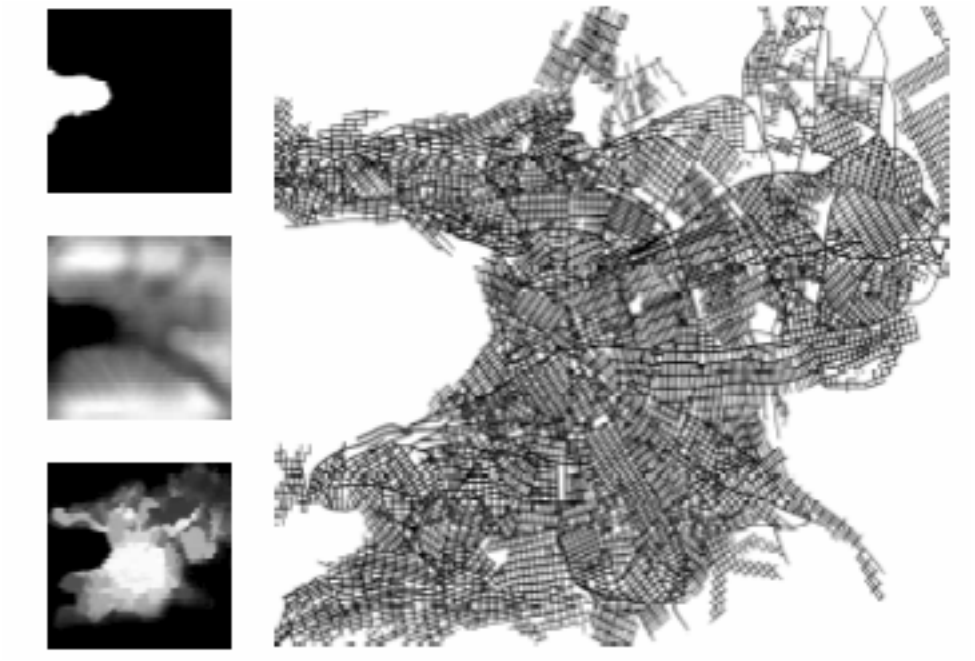


Figure 4.4 : Street map generated using elevation and population density maps.

Population maps invoke the L-System that generates road networks. Road networks are the medium by which populations transport and Pascal Müller and Yoav I H Parish state that the population density of a city is influenced by the creation of streets. They also state that not all roads have an impact on population. Therefore, there has been a distinction made between roads: highways and streets. Highways are described as connecting areas with high population concentration. On the other hand streets, work in between the highways are covered by streets providing the neighborhoods transportation.

After the road map is generated, rest of the land is divided into smaller areas. Allotments are defined by geometrical subdivision of those areas. Then the second type of L-Systems generate the buildings themselves to be placed on the allotments.

The last step of the software is the façade texture generation. Every façade is turned into a grid using shape grammar rules and then they are assigned textures and 3D architectural models such as doors and windows are placed to necessary opening.

4.1.2 Street Map Creation

To create street maps, L-System which is a set of production rules that create a parallel string mechanism. Each string is made up of a number of modules which are

translated into commands. The parameters are stored within the commands of these modules. The article states that since it is difficult to re-write a rule every time a parameter changes, the software developers decided to let L-System create a generic template at each step which is called the ideal successor.

Since changing a parameter means rewriting the rule a generic template system is designed to avoid it, setting of parameters and modifications are turned into external functions. First of these functions is called the ideal successor. Then two other functions are applied to an existing string of module. These functions are called “global Goals” and “local Constraints”. When the L-System returns the ideal successor, the parameters of the modules are unassigned and the other two functions are called. The image below summarizes the relation between these three functions.

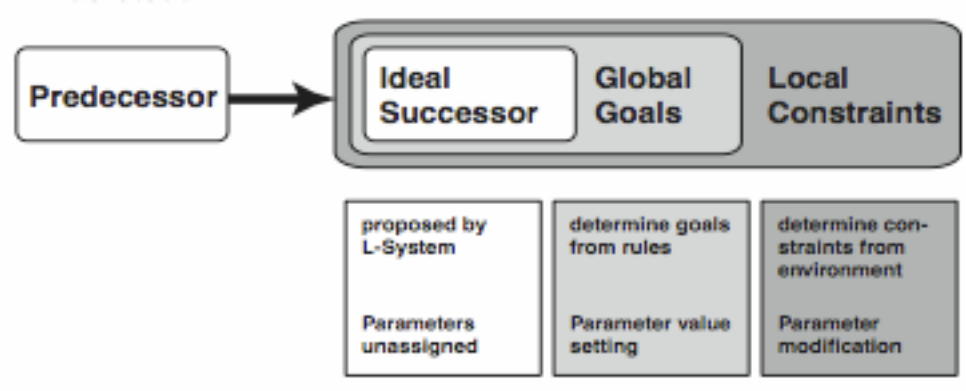


Figure 4.5 : Relation between external functions.

After the ideal successor is called, the global Goals function which determines the parameter values of the dominant global goals is called. This function generate street patterns according to population density. As described before, one of the two types of road networks, highways connect highly dense areas in terms of population. In order to generate highways, the system needs to locate densely populated areas. To achieve this task, “every road shoots a number of rays radically within a preset radius. Along this ray, samples of the population density are taken from the population density map. the population at every sample point on the ray is weighted with the inverse distance to the road and summed up. The direction with the largest sum is chosen for continuing the growth”. This process is represented in the figure below.

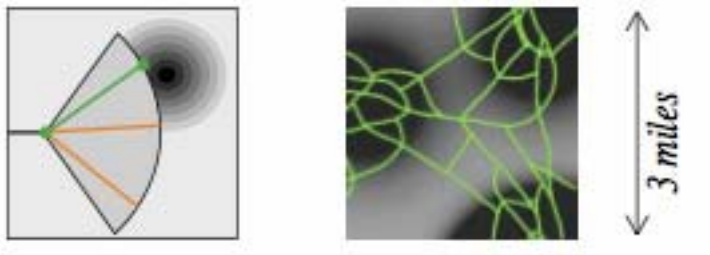


Figure 4.6 : Detecting population density to map highways.

Then the last function, local Constraints, is called. In this function, the parameters that are set by the global Goals function are checked to ensure that they do not violate a restriction set by local Constraints. It checks if constraints regarding to land, water, park boundaries, elevations and crossing of streets are violated. The local Constraint checks if the road segments terminate inside or crosses an illegal area such as water. It searches for intersections with other roads. If the system detects a road segment that ends in the water or a similar illegal area, it adjusts the values in a couple of ways. First, it trims the segment length so that it fits inside the specified boundary. It rotates the segment within a maximal angle until it fits inside the specified boundary. The final step of the local Constraint control is about the highway generation. The only member of the road network which is allowed to cross water is highway. If such a case takes place, the system flags the highway segment that crosses water and later the user is allowed to replace the highway with a bridge or a tunnel. The function also searches for intersections and crossing of the streets. If the system detects two streets intersecting, it generates a crossing. If a street segment ends close to an existing crossing, the system extends the street to reach the crossing. And finally if the street segment ends close to a possible intersection, the system extends street to form an intersection. This function is developed to adjust the necessary parameters. If it detects that a parameter violates a restriction, it searches for alternative parameters to suit the restrictions and if the system returns a FAILED state flag if it can not find a suitable parameter and deletes the module subsequently.

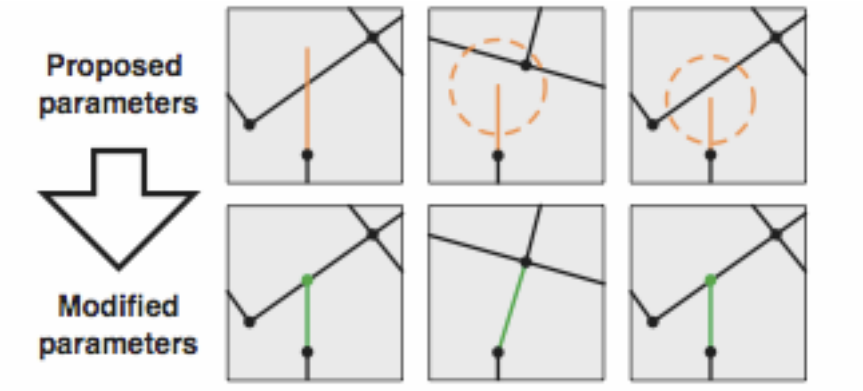


Figure 4.7 : Correcting intersections and crossings by local Constraints.

The software developers have created a set of selection of patterns to apply on the street map. They are grouped as Basic Rule, The New York (checker) Rule, Paris (radial) Rule, San Francisco Rule.

The Basic Rule is the one which does not consist of a superimposed pattern. The roads grow following the population density. This rule represents the natural growth of transportation network and relates to central areas of historic cities. The rules applied within this rule, are the restrictions used by the following type of rules by narrowing down the options of branch angles.

The New York Rule is described by the developers as to follow a given global or local angle and maximum length and width of a single block. This is a very common pattern where the highways and streets intersect each other at right angles forming rectangular urban blocks.

Paris Rule is described as that the highways follow a radial track around a center that can be either calculated from the input data or set manually.

San Francisco Rule is described as that it allows the pattern to form by streets and highways following the route of the least elevation. The developers of the rule state that the roads on different height levels are connected by smaller streets which follow the steepest elevation and are short. Below is a chart showing the pattern name, pattern description and an example image.

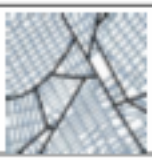
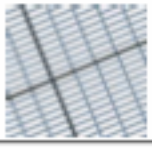
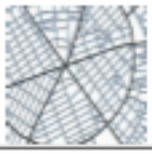
Pattern name	Pattern	Example
Basic	No superimposed pattern.	
New York	Rectangular Raster	
Paris	Radial to center	
San Francisco	Elevation min or max	see figure 6

Figure 4.8 : Preset street patterns.

Below is the example of San Francisco Rule. The elevation map is shown and the resulting highway map is projected onto the elevated geometry.

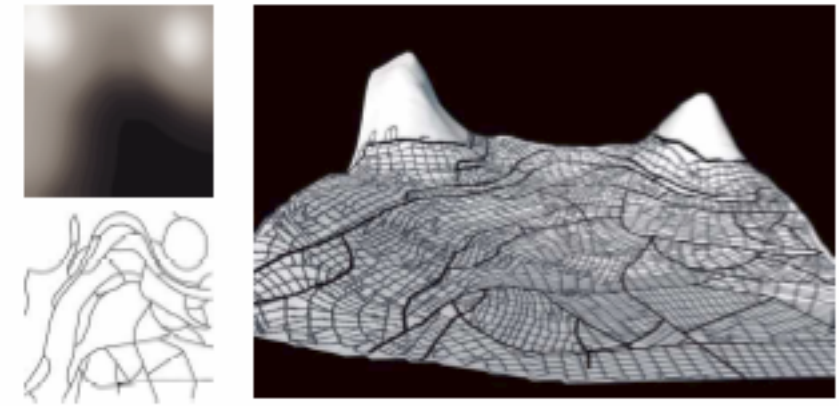


Figure 4.9 : San Francisco rule applied.

It is also possible to apply more than one rule for a location. All the restrictions are evaluated and the proposed parameters are summed and weighted according to the image input.

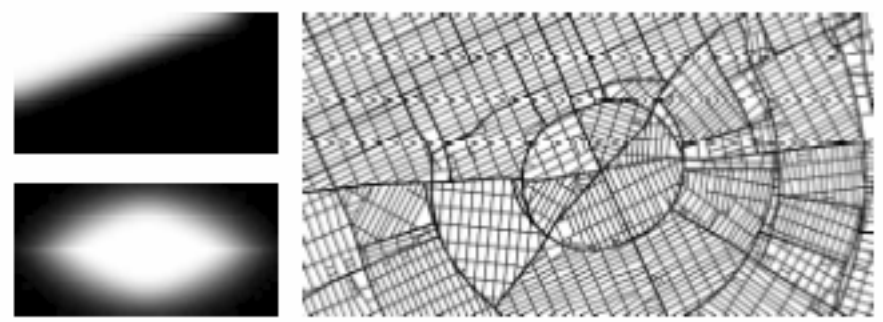


Figure 4.10 : Two patterns used at the same time.

The following figure shows initial stages of visual generation of Manhattan, the final stage of the generated model and the real map of Manhattan in order to be able to compare the system's logic with the evolution of a real city. It can be observed from the image that the oldest part of Manhattan is generated using the Basic rule which does not impose a predefined pattern and the new parts of Manhattan are generated using The New York Rule which generates a grid and rectangular urban blocks. It can also be observed that the proposed bridges are very close to where the bridges are actually placed today.

Figure 4.11 : Generation of road map for Manhattan and actual map of Manhattan

4.1.3 Modelling the Buildings

After the road map is generated, the next step the system generates the allotments on which the buildings will be placed. The area that is in between the streets is called a “block” by the developers. The system divides these blocks into smaller geometries by a recursive algorithm which divides the longest edge until the subdivided lots are under the threshold specified by the user. These subdivided geometries are then called “allotments”. After the allotments are created, the ones that do not have direct access to a street and the ones that are too small are discarded from the system. Below is the figure that shows a street map that is generated, the blocks that are created and final the lots that are generated respectively.

Figure 4.12 : Street map, blocks and lots generated by CityEngine.

The buildings to be placed on the lots are generated as the next step. The type of buildings possible are categorized as skyscrapers, commercial buildings and residential houses according to zoning rules. The developers state that the buildings are generated from an arbitrary ground plan which is the bounding box of the lot. The final shape of the building is determined by its ground plan which is transformed by interpreting the output of the L-System. Therefore the final building form can be defined as reinterpretation or refinement of the initial lot geometry.

In this step of the pipeline, the rules first create mass model of a building and then continue to generate the façade and the last move is to add details for windows, doors and ornaments.

The simplest construction method takes a box as the basic primitive and applies scaling, translation or split rules to create mass models. Below is a figure representing basic small mass models that can be established by applying rules to a simple box with an addition of a cylinder. (Müller, P. et, al. 2007)

Figure 4.13 : Primitive shapes used to generate buildings.

An applied example of building generation using CityEngine is demonstrated by Petronas Towers in Malaysia. This example uses the rotation tool applied on a box and an array of cylinders.

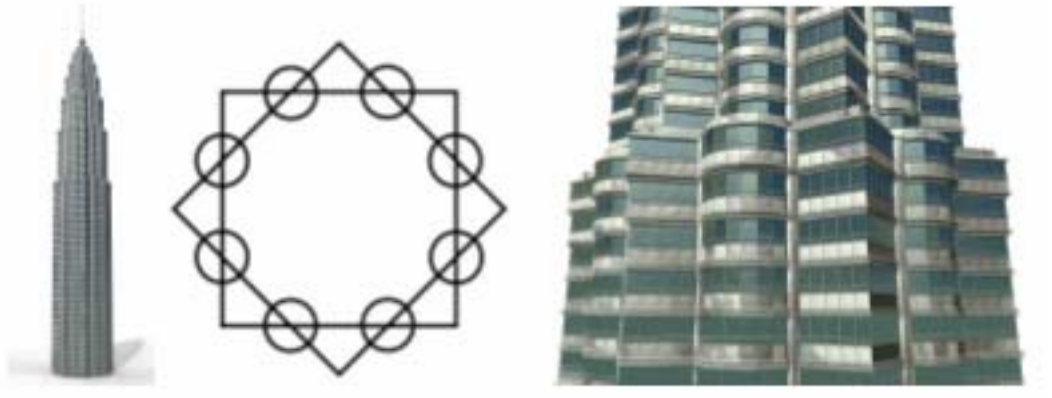


Figure 4.14 : Demonstration of Patronas Towers.

A building can be generated in two ways. First, the lot defines the axiom of the grammar. The mass model is then placed on the lot and it undergoes a set of rules. The rules that can be applied are scale, translation, rotation and split operations with the restriction that the mass model can not go beyond the limits of the lot. The second way that a building can be generated is more restrictive. This method takes information from a GIS database or imports an existing architectural model and can only make minor changes to it (Müller, P. et, al. 2007).

This step also integrates the addition of a roof. Roof types can be one of the following gambrel, cone, gabled, hipped, cross-gable and mansard as shown in the following figure.

Figure 4.15 : Placing of roofs.

4.1.4 Façade Generation

In this step of the software, the user is able to generate a façade based on a given 2D image. This step is capable of detect repetitions, derive a top-down subdivision schema and infer shape grammar rule sets from complex images (Müller, P. et, al. SIGGRAPH 2007). In this step, an input image stemming from ground-based on areal imagery is transformed into a textured 3D model. To achieve this, the authors state that a top-down hierarchical subdivision analogous to splitting rules is used.

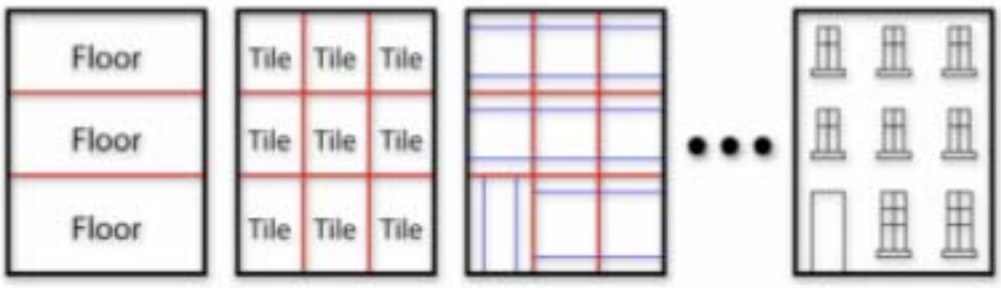


Figure 4.16 : Subdividing a façade.

This step consists of sub-steps named a determination of façade structure, subdivision of façade tiles, matching 3D elements and editing and rule extraction.
First, a 2D image is fed into the system so that it can subdivide it into floors and tiles. The algorithm detects similar image regions using mutual information (MI). Then, a data structure called Irreducible Façade (IF) is created so that information about symmetry on the façade is encoded. Lastly, IF is analyzed to detect further subdivision of the tiles (Müller, P. et, al. SIGGRAPH 2007).

Next, detected tiles are subdivided into smaller regions. The algorithm recursively selects the best dividing line within the tile. In the figure below, the blue lines illustrate the division of the tiles hierarchically.

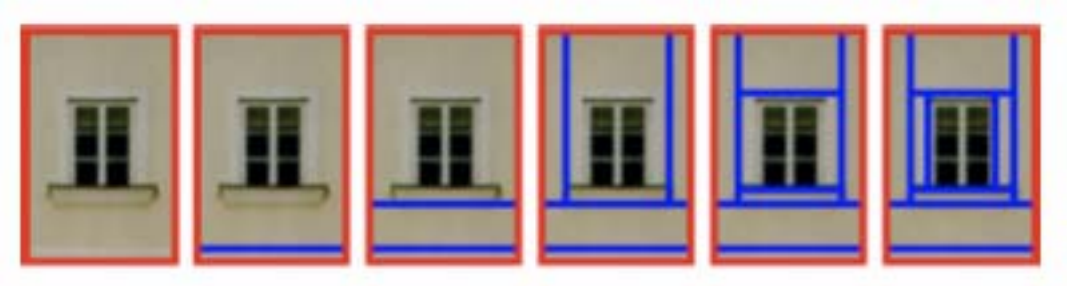


Figure 4.17 : Subdividing the tiles.

As mentioned before, the next sub-step is to match 3D elements. This is achieved by grouping the results of subdivision process. In this stage, MEL Script is taken advantage of to match the elements of the library consisting of architectural modules with the clustered subdivisions of a tile. The figure below show 18 elements of the 3D library consisting of 150 elements in total.

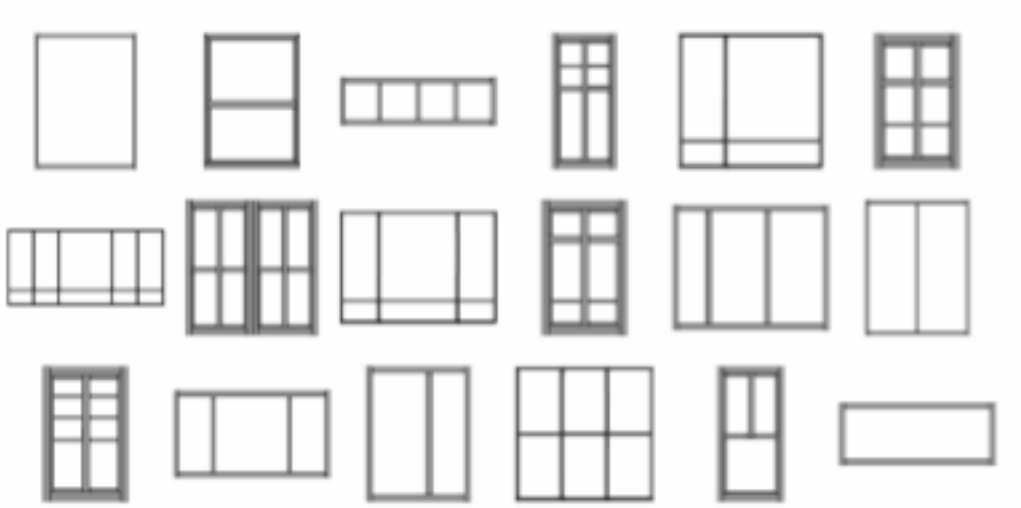


Figure 4.18 : Architectural 3D elements in the library.

In the last stage, the resulting façade gains depth values. Up till this point, it has been encoded as shape tree without depth information. The user can select clusters interactively to define their depth value. Below, the figure represents the input image, 2D generation of the façade and the end result with depth value of the elements of the façade (Müller, P. et, al. SIGGRAPH 2007).

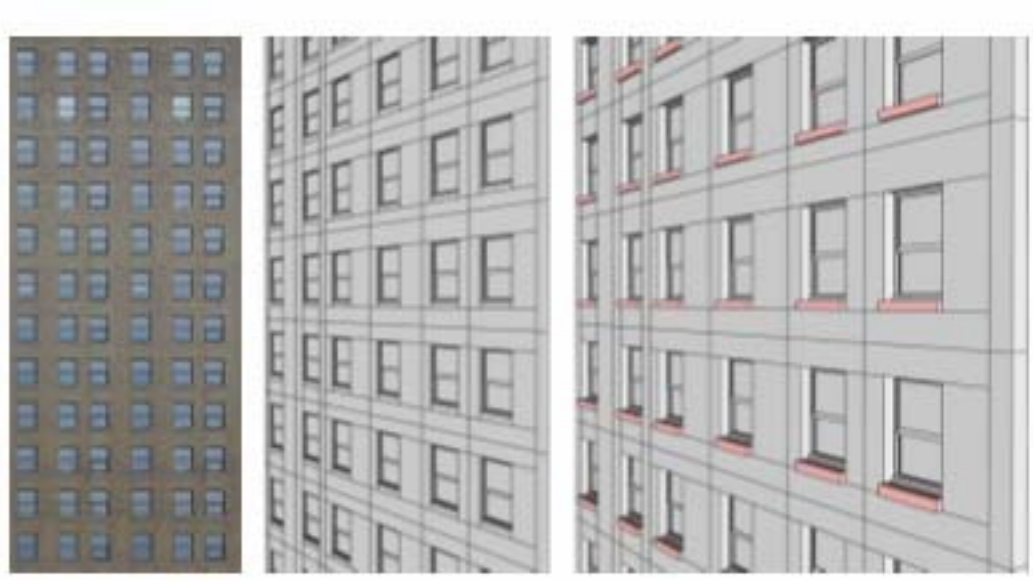


Figure 4.19 : 2D image input, generated façade and façade with depth value.

4.2 The Melinkov Grammar

The Melinkov Grammar is developed by Daniel Cardoso to generate architectural elements and urban morphologies stemming from the Russian architect, Konstantin Melinkov's design. The author of the article, The Melinkov Grammar, describes his work as having a particular generative logic or having consistent exploration of a set of simple formal and geometric themes.

Konstantin Melinkov's most particular work is his own house that is completed in 1929. The building consists of three levels of intersecting two cylindrical towers. He has located the staircase that connects the 3 levels of the house is located at the intersection of two cylinders. The partitions are result of either radial lines that are in the rooms or of the projection of the missing wall of the cylinder which are in the public areas. Another feature of the design is that one of the cylinders have a window on the flattened side. This feature helps to differentiate the two cylinders through the quality of light.

Computer implication of this grammar generates designs semi-autonomously satisfying user defined constraints. The constraints that can be defined by the user include the site's boundary, collision avoidance, a series of pre-existences and adaptation of the elements to maximize views (Cardoso, D. 2007).

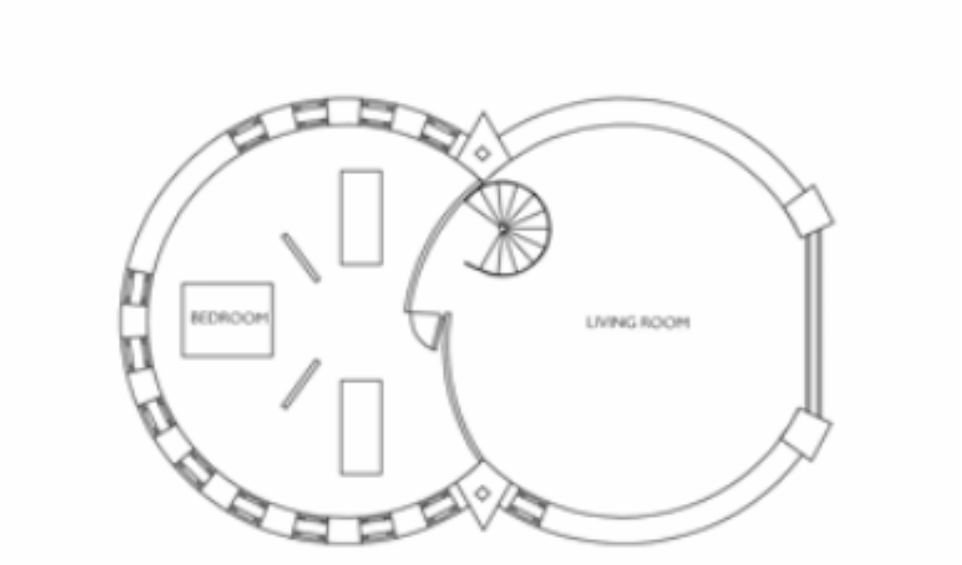


Figure 4.20 : Konstantin Melinkov's own house.

The grammar consists of two rules. The first group of rules is responsible for connections between the primitive types. It consists of cylinders divided at 180 degrees and at 120 degrees creating two and three sections respectively.

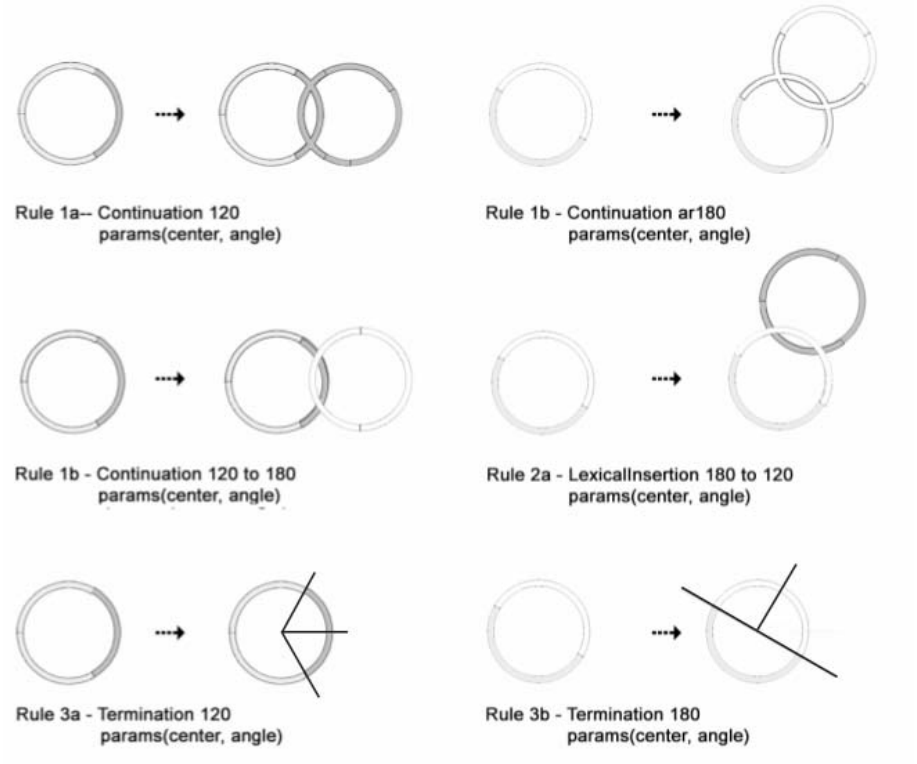


Figure 4.21 : Rules

The second group of rules is responsible for adding windows, doors and staircases. A shape can only be detailed only if it has no more possible transformations to go through. These two groups of rules repeat themselves and when the first one terminates the second one starts repeating itself until there are no more shapes to detail, then the whole process terminates.

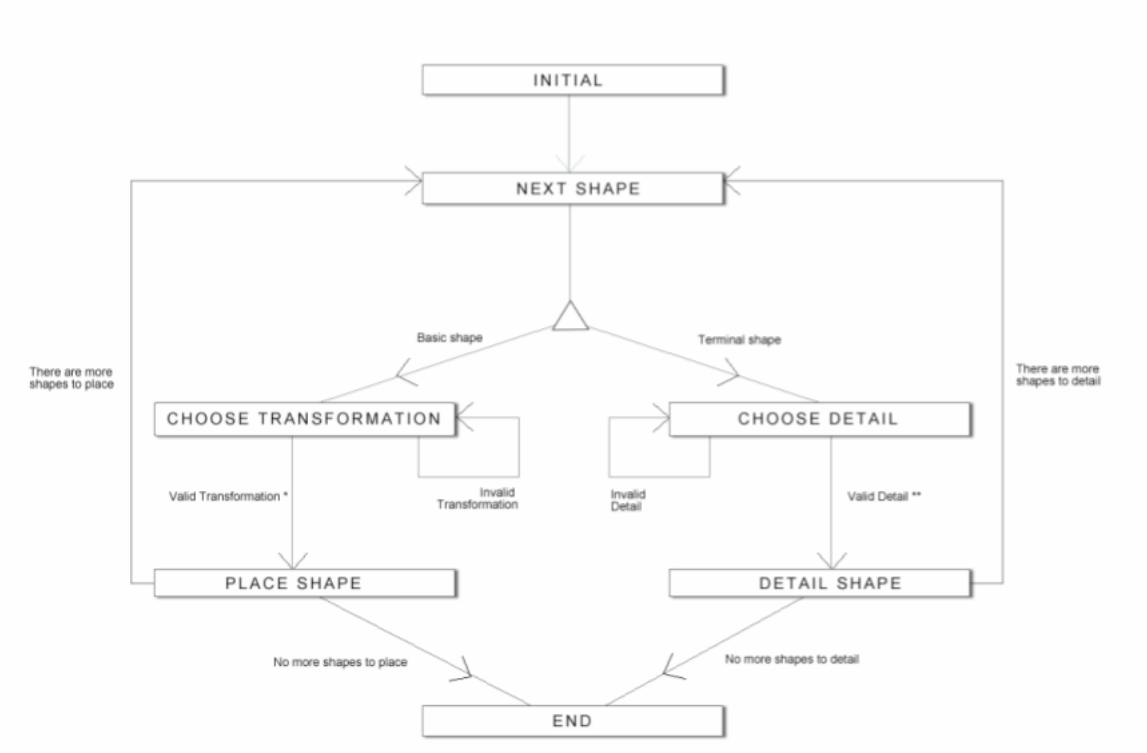


Figure 4.22 : Process flow.

This rule takes view constraints into consideration. This means that one of the two cylinders that are too close and face each other is unlikely to get a large window. Other constraints embedded in the system are a collision constraint which avoids a shape being drawn onto a previously drawn shape, an optional radius constraint which avoids shapes to grow out of a specified range, an obstacle constraints which avoids the shapes to hit a predefined object defined on the site and a boundary constraint which prevents the design to grow out of a defined boundary. Below, is a figure showing an example derived by the grammar by a boundary constraint (Cardoso, D. 2007).

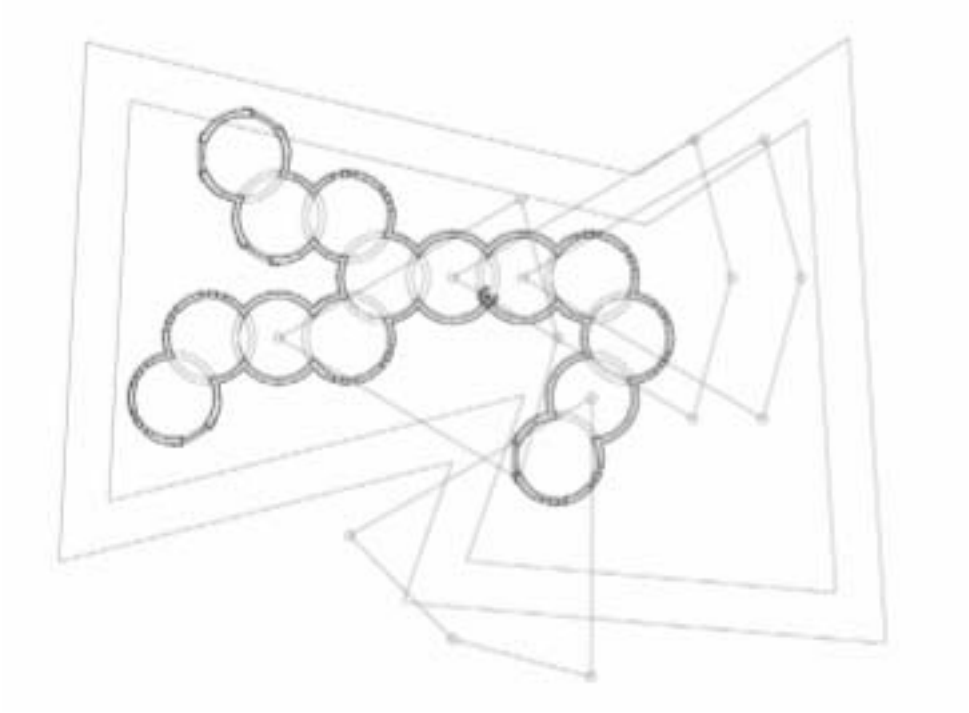


Figure 4.23 : Example generated with a boundary constraint.

Both architectural and urban designs can be achieved through using Melinkov Grammar. The architectural designs can be as detailed as having doors, windows and stair cases where a generated city morphology help the user visualize a network of regions which are shown in the following figures respectively.

Figure 4.24 : Architectural design alternative generated with Melinkov Grammar.

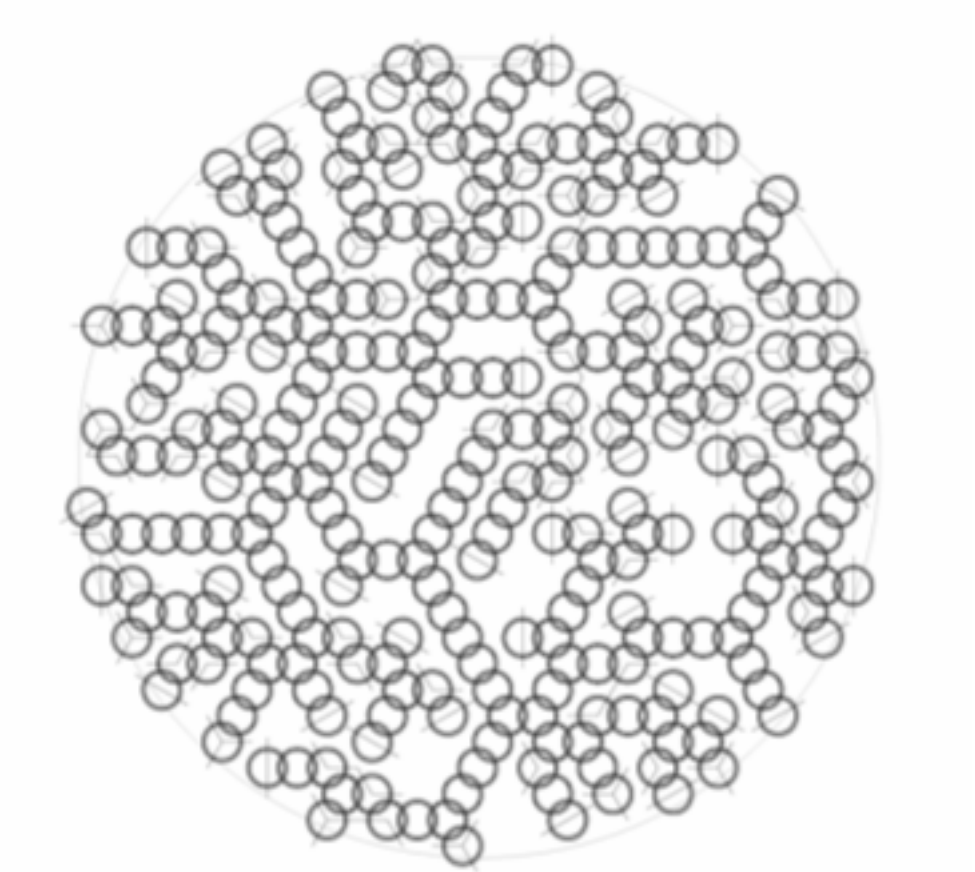


Figure 4.25 : Urban morphology alternative generated with Melinkov Grammar.

This process has not been developed directly for generating new urban morphologies. It only can be utilized when a network has to be established without considering building block allotments. When compared to the outcome of this thesis, this approach offers a substantial method to generate a city network however it does not further develop the idea and stays rather as an abstract idea.

4.3 CityZoom

CityZoom is a software which is used by both planners and architects in terms of issues such as solar radiation, luminance, environmental comfort, sky lines, view obstruction and privacy.

The software is a full decision support system for urban design. The software offers the users an interactive model where various building performances can operate. It's main tool is a graphical editor consisting of urban features. Input data can be obtained through various sources such as free hand drawings, aerial pictures or CAD

files. The software is capable of generate a city model depending on regulations which are entered through Urban Regulations Editor (Turkienicz, B. et, al. 1999).

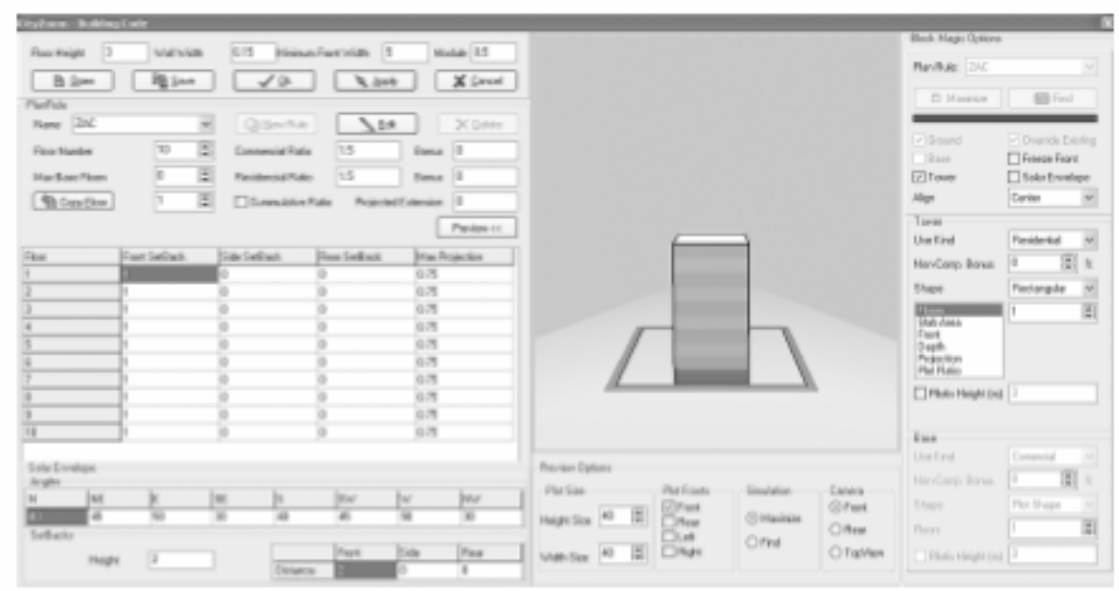


Figure 4.26 : Urban Regulations Editor.

The user can set Master Plan parameters consisting of number of floors, certain ratios regarding the allocation of types of function and minimum setbacks. The editor can also investigate environmental comfort issues by making use of Solar Envelope technique which makes the buildings subject to sunshine and the physical boundaries of surrounding properties. The way that the parameters for the measures dictates the buildings final size and form. “Planning for insolation is essential in establishing the visual and thermal comfort, ie., the benefits to be obtained from the sun in and around the buildings.” (Turkienicz, B. et, al. 1999). The user can set obstruction angles for every plot. These angles are projected onto edges of every plot and generates a boundary. As a result, buildings within this boundary are not allowed to drop shadows over the neighboring buildings during specified times of the year. After the parameters are set and calculations are made, the results of the simulations are visualized as 3D graphics as well as numeric data and graphs.

Another feature of the software is the tool called Mosaic. This tool visualizes the information stemming from performance studies. It applies an orthogonal grid and divides the specified are into uniform cells. Each cell can be assigned an attribute such as building footprint or height and these cells would hold a numeric value. The cells in the grid then takes numerous colors depending on the value it holds

representing the intensity of the attribute the cell is holding. This tool works as visual analysis tool which helps the user to determine patterns or clusters of a specified attribute.

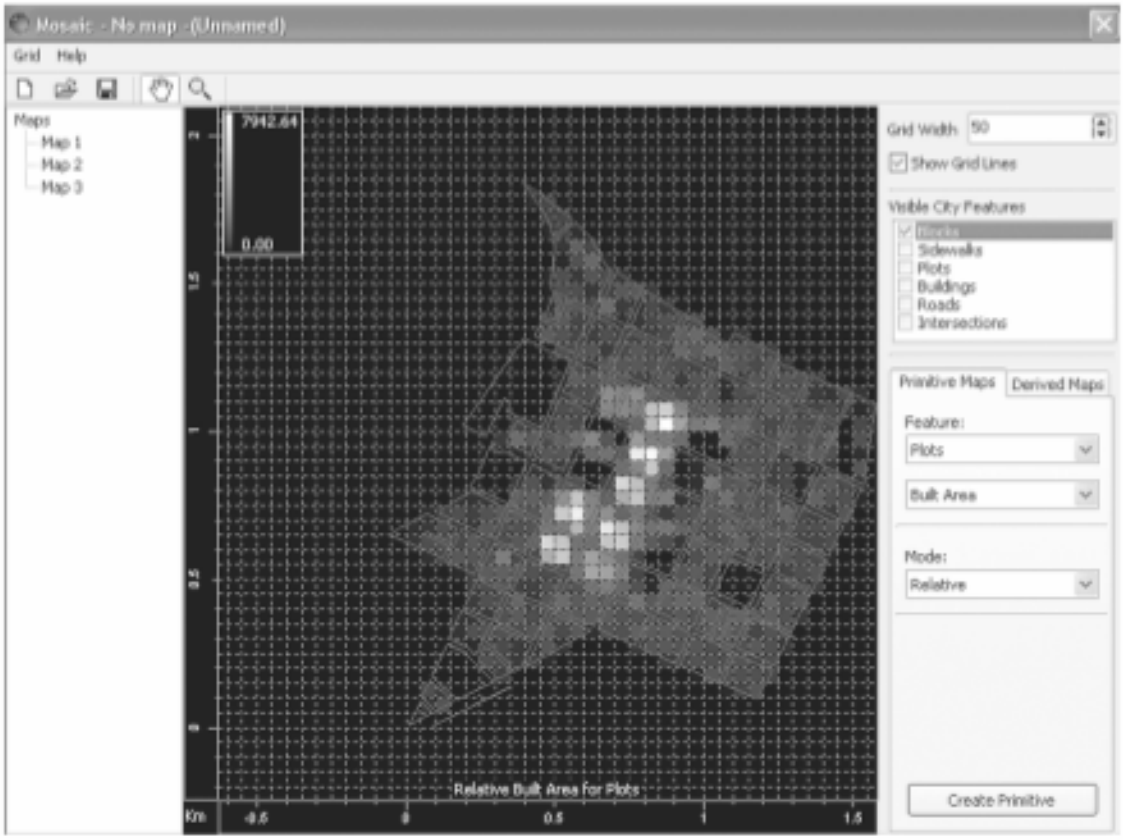


Figure 4.27 : Mosaic

CityZoom is a software which enables the user to visualize a city according to regulations and modify it after running various performance models on environmental comfort issues. However, this software does not provide the user the ability to generate a city from scratch like other tools explained before.

4.4 Urban Design with Patterns and Shape Grammars

A design studio developed by Jose N.Beiraroı and Jose P. Duarte has implemented urban plans by using Alexander's pattern language and Stiny's shape grammar. The study has resulted in proving that use of patterns help to tie the urban plan to a specific development vision and shape grammars are successful in generation alternative design solutions that match varying programs (Duarte, J. P, Beiraoı, B. 2009). The article written on the studio process state that the students were challenged to design developing areas in Portugal and that the designed areas needed

to consist of an urban program and be flexible design solutions. Several generated design solutions will be explained in this section since this particular study links to this thesis in terms of having a bottom up approach and dealing with generation of an urban block in some of the projects.

The first design alternative follows a process of subdividing polygons similar to Stiny's Chinese lattice windows which are represented in the following figure both shape wise and applied in a territory.

Figure 4.28 : Areal photo of existing urban structure and generation rules.

The last rule in the second image belongs to plot division. The plots are drawn perpendicularly to the main street. The parameter "w" is a multiple of 3 meters so that the generated plot is suitable for building. The next figure will show where activity nodes and new centralities were placed through another set of rules.

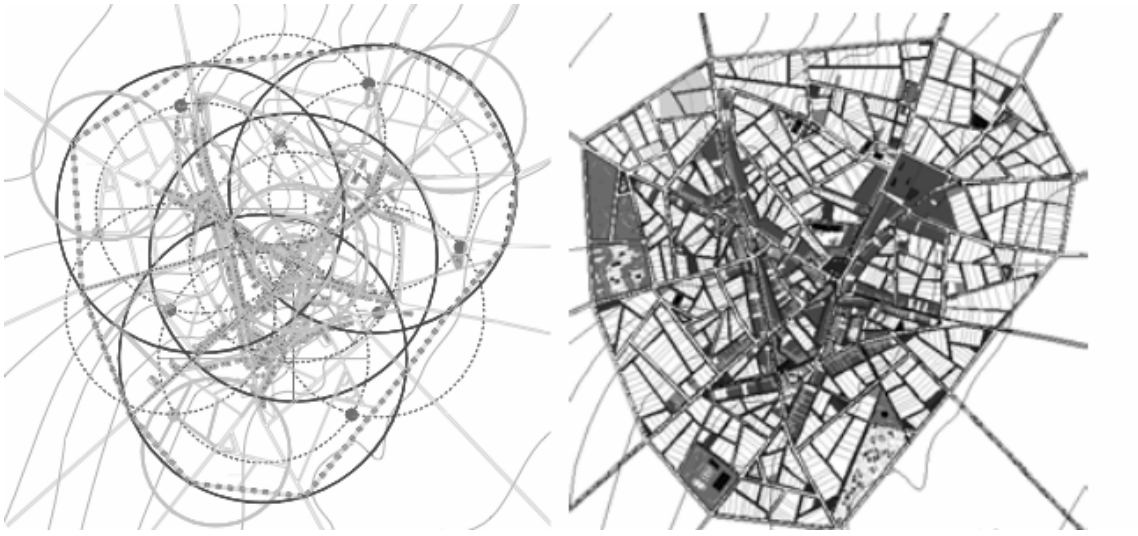


Figure 4.29 : Rules applied for activity nodes.

The second design alternative creates an urban tissue by repeatedly adding clusters of four city blocks. Two different blocks sizes called TA and TB are used which have the same width but different lengths. Addition of blocks follow three rules. First, two orthogonal axes define the initial reference for the generation sequence. Then, blocks facing each other by their longest side should keep a distance of 12 meters and finally blocks keep a distance of 10 meters when a shorter side faces the longest one. The following figure shows applications of these three rules.

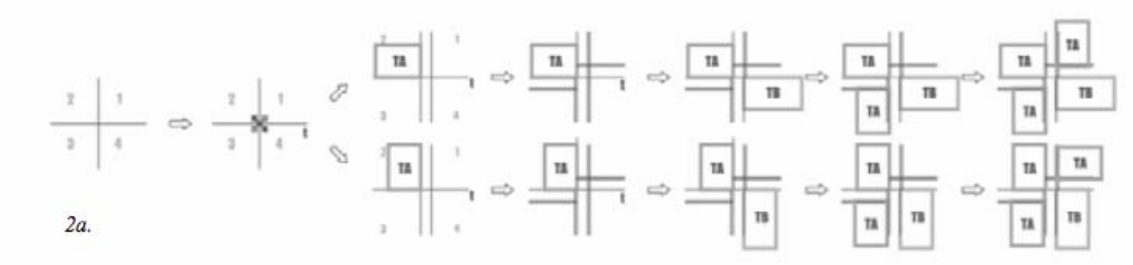


Figure 4.30 : Creation of an urban block.

Then the students in the studio project have developed a rule for adding clusters to which an example is represented below.

Figure 4.31 : Clustering of urban blocks.

Recursive application of the rule within a predefined set of main streets applied along morphological features like water lines result in new urban designs. Again, in this project the students have created activity nodes and they have land marked such nodes by applying subtraction rule which deletes one of the blocks and creates a public square.

It is stated in the article that the results proved, the rule based design approach helped the students to deal with complexity and flexibility issues. This is one of the issues that connects this work with this theses' approach. On the other hand, this study

presents a bottom up fashion of creating an urban design supporting the approach of this thesis. Especially the second project of this study is the final point that strongly links the study with the thesis since it also takes an urban block as the starting point of design. This theses however, only elaborates creation of an urban block in 3D and does not go beyond nor creates a shape grammar for urban blocks to become clusters and a final urban design at this point.

4.5 Smart Solutions for Spatial Planning

SSSP is a collaboration between University of East London Center for Evolutionary Computer and Architecture and Aedas R&D. It is a knowledge transfer partnership therefore attempts to break down the barriers between disciplines which are urban designers, architects and computational designers. The project attempts to build a digital chain of different scales of urban designing process. It simulates an urban planning workflow and deals with it from urban scale to building scale.

First, site analysis application developed calculates the shortest distance between pedestrian routes and places. Potential active routes can be visualized from a given place.

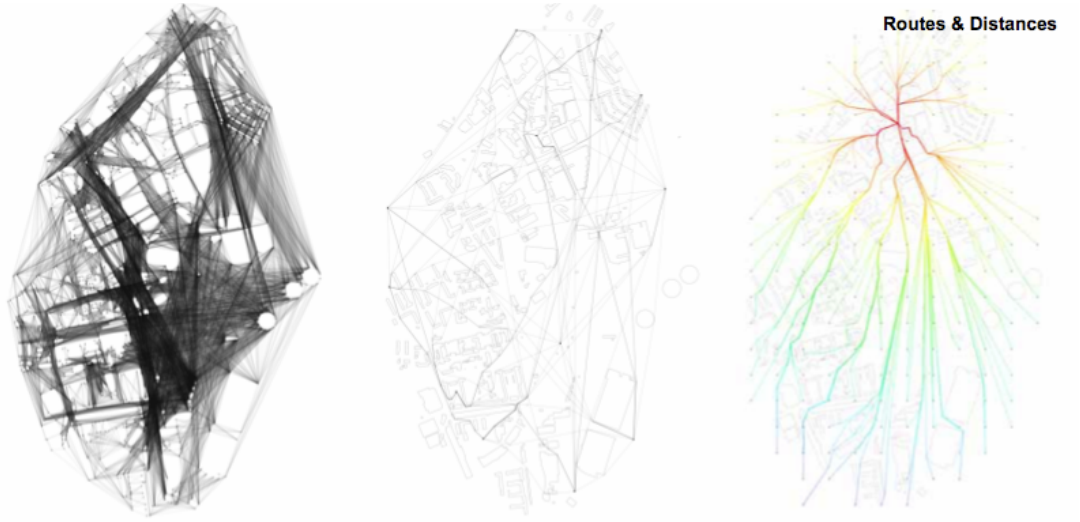


Figure 4.32 : Active routes on the site.

Second application developed deals with access points on the site. Dots on the image refer to marked places and the line in red color means zero minutes of walking and blue color indicates maximum amount of walking which is 20 minutes from and to

the marked places. Improvement of walking time is developed by offering new shortcuts on the site.

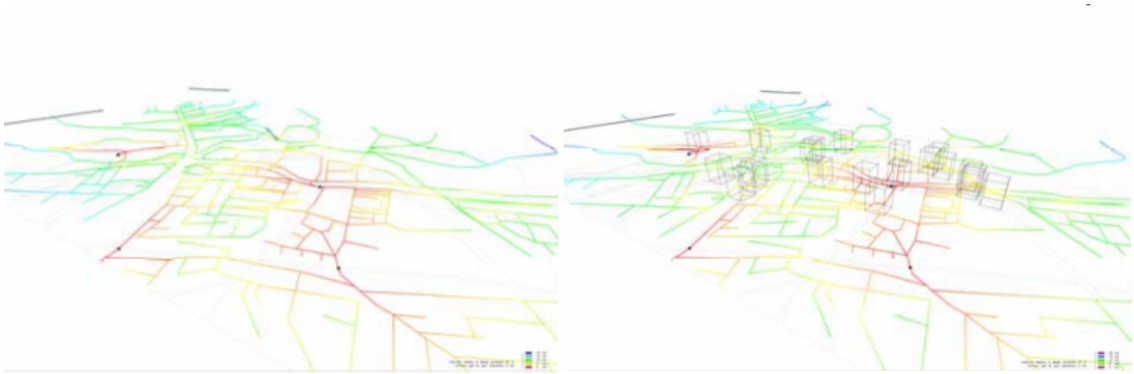


Figure 4.33 : Walking distances from specified places.

After analyzing the site for accessibility and creation of new routes, important places on the site represented with red dots which are shopping places restaurants and cafes are connected through minimum distance of main roads minimizing the time to travel and paved area. If the designer chooses to change the location of these places, the program generates a new road network.

After calculating sustainable main road networks, the next application generates minor roads. The application works with the main roads and divides the site into a series of new city blocks. These blocks are automatically proportioned so that reasonable allotments are generated for building development and open spaces. The program searches for a structure of minor roads recursively where the user can choose an alternative or let the program search for a better alternative.

Figure 4.34 : Block allotment

In the next application, different types of buildings and open spaces arrange themselves according to preferred conditions. The quantity in preferences of these buildings and open spaces are set up by the designer who is trying to create a balance between regulations and local planning policies. Different colors represent different building types and open spaces. Colored circles move around the site until they find themselves a place fulfilling conditions like access to public transport, nearness to main road, water edge or key access point within the site. Additionally, one can visualize the density by massing scenarios by showing the potential volumes at each location. When the designer changes the parameters, the alternatives will be updated automatically.

Figure 4.35 : Land use.

Finally on building scale, building outlines that take into account aspects like climatic conditions, building depth and height, distinction between public and private realm and distribution of shops and flats on different levels within the block.

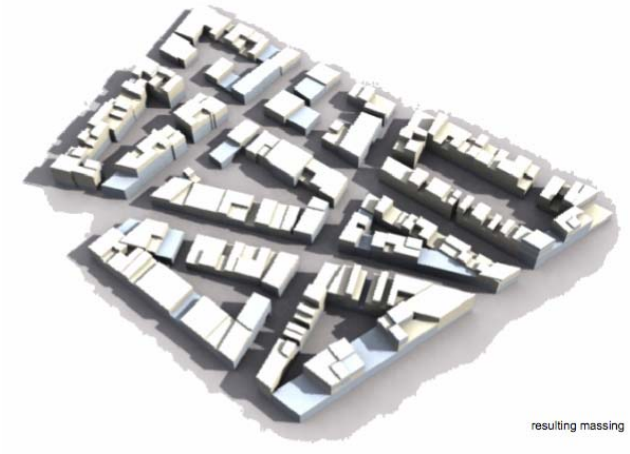


Figure 4.36 : Mass model of the site.

SSP is a new set of tool allows architects and planners experiment sustainability issues, development and regeneration issues quickly and accurately. This project is related with the outcome of this thesis by two points. First, although very crudely, this project takes building block generation process into account. The block generation phase is highly optimized in terms of transportation and creation of substantial and reasonably dimensioned allotments. Second, it has already achieved one of the intentions of this thesis which has been previously stated as bringing architects and urban designers together. On the other hand, this project contrasts with the outcome of this thesis in not mimicking any existent pattern.

4.6 Conclusion

As it can be observed in this chapter, there has been many efforts to form new cities using computation. Most of such efforts combine Shape Grammars. And all of them are composed of Rule Based Systems, similar to this thesis. However, this thesis is unique and differentiates by focusing only on block generation which is rarely seen in the approaches explained above. CityEngine has attempted to rationalize generation of urban blocks by a recursive division algorithm but its top-down approach makes the results predictable. However, this thesis' scope is not as wide as the CityEngine's where it deals with a city from scratch to façade detail, the "Mirrorer" only deals with an urban block and allotment in terms of CityEngine.

This thesis takes urban block generation into consideration in a bottom-up approach similar to the Melinkov Grammar where the end result is complex although the process and the initial shapes are very simple. However, this thesis differentiates from the Melinkov Grammar approach in that the Melinkov Grammar is a 2D generation tool where the "Mirrorer" generates solid urban blocks in 3D rather than a 2D city network.

On the other hand, CityZoom shows a distinctly diverse approach compared to the outcome of this thesis. It is highly oriented to enable the user visualize a city according to regulations and modify it as a result of a set of performance models. Thus, finding a balance between design needs and regulation requirements is the higher purpose for this software rather than offering a system that creates a city from scratch as it is intended with this thesis.

Urban Design with Patterns and Shape Grammars Studio does not offer a computational product but it is a suitable example showing that rule based systems create a more flexible process in terms of dealing with complex design problems such as a city design. This study has been explained within this thesis to present as an experiment that proves efficiency of Rule Based Systems as well as because it consists of approaches developed to generate an urban block at a level which is not present in other models explained.

Smart Solutions for Spatial Planning is another software developed which takes generation of a city into consideration from urban scale to building scale including a step to generate urban block alternatives. The software presents an approach on building block generation depending on generation of allotments with reasonable dimensions for development of buildings and open spaces following a recursive search method. Next, building masses are created taking climatic considerations into account and making a distinction between public and private realm. This step of the software relates to this thesis in terms of intention to create urban blocks with building masses but differentiates with its methods. SSSP uses a recursive search method in dividing the plot into allotments depending on optimization rules and creates building masses taking environmental and social conditions into account which is similar to the approach to city in this thesis in terms of they both take environment into account, however at a different level.

As a result, the presented models should provide a background of fieldwork related to this thesis for the reader as well as the points where the outcome of this thesis can be compared and contrasted to such related work by the reader. It should also provide ideas regarding to which direction the next level of follow up research should head.

5. “MIRRORER” AS A PLUG-IN FOR URBAN DESIGN

5.1 Overview

The design problem of this thesis which is to create a process which mimics environment building block elevations to and alter depth values to create unique plan alternatives has been addressed to by a theoretical, proposed process design two different realized implicated approaches.

5.2 Proposed Process

This plug-in is proposed to work as a plug-in for 3D Studio Max 9.0. It Is designed to help the designer in the early site development phase in a certain way, with a certain method. The plug-in only deals with urban block. It takes mirror images of each neighbour building on the other side of the street and reflects it on the corresponding elevation. This process results in a block that has four different elevations which are reflections of the buildings that are across the street. Although creating symmetric streets is not exciting, this method yields to many creative results in plan when the reflected images on the site gain gradual depth values. As a result, the mirror images on four sides of the site merging towards the center of the site have the potential to end up as a whole one building as well as multiple building blocks with courtyards.

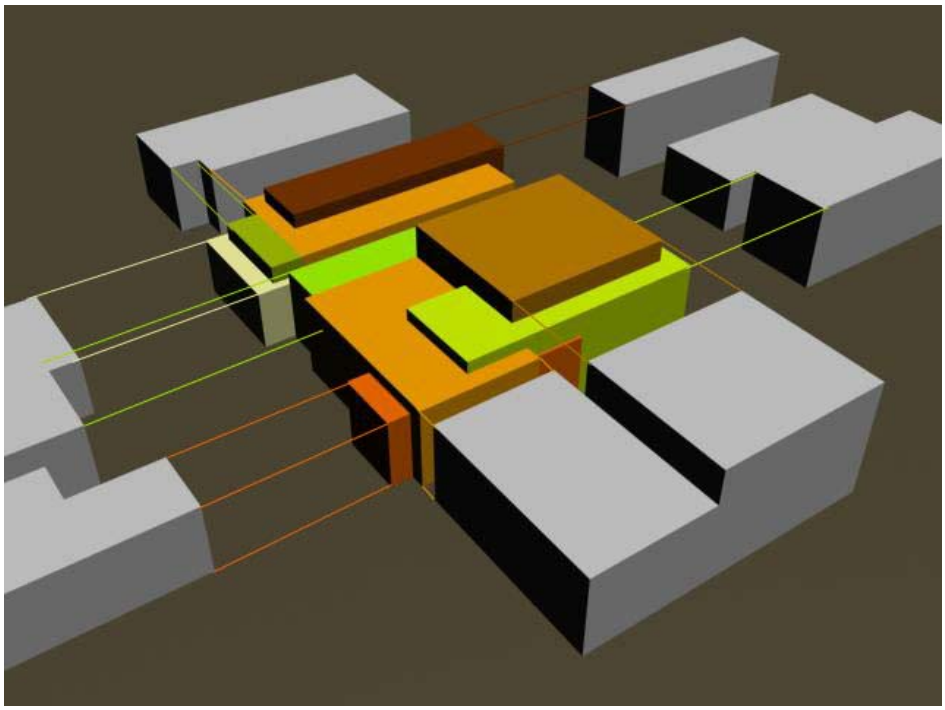


Figure 5.1 : Possible results of the plug-in of an urban block.

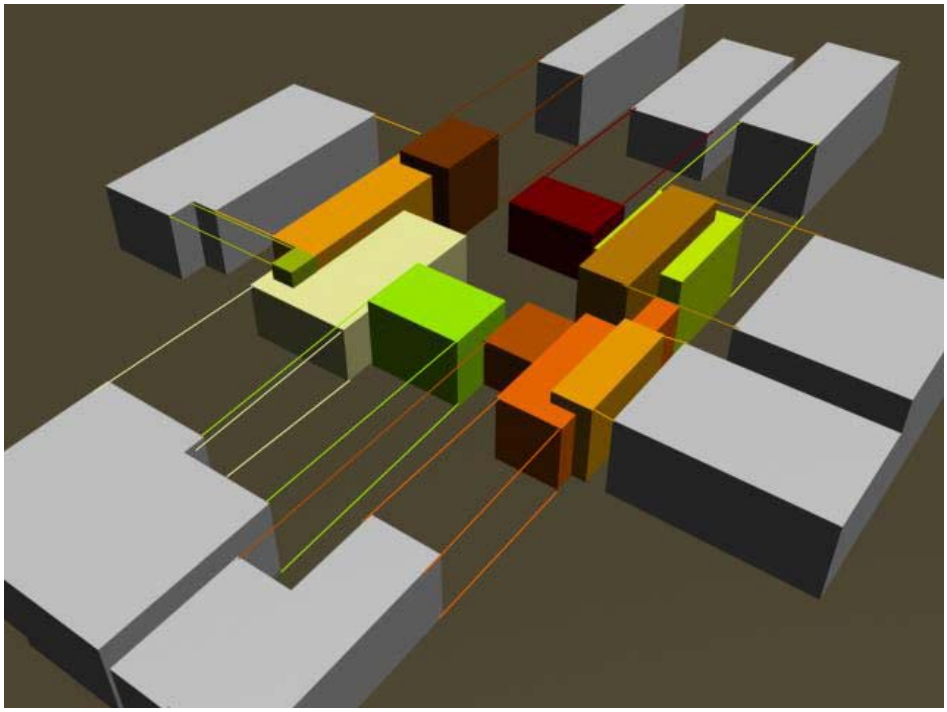


Figure 5.2 : Possible results of the plug-in of an urban block

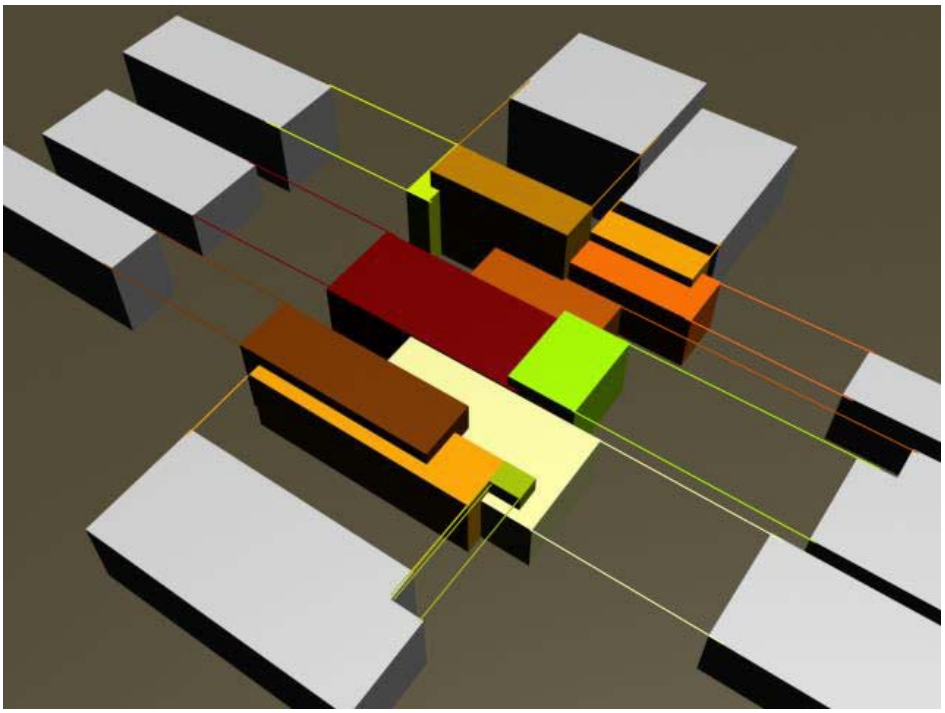


Figure 5.3 : Possible results of the plug-in of an urban block.

5.3 User Requirements

The user is required to have the site represented as ‘shape’ and name it as ‘site’ so that when the plug-in is run, it knows according to which axis to take the mirror images around which means that the images are reflected and gathered around the correct shape in the scene. Then the user is required to draw in the neighbor buildings which will be the source of the mirrored images. These buildings should be represented with ‘box’s only for two reasons. First, the plug-in does not need any detailed drawings. Therefore any windows, doors or ornaments drawn in would be unnecessary and cumbersome to compute. The plug-in only needs the boundaries of the buildings. Second, the plug-in needs to know the difference between the axis around which to rotate the images and the images that will be mirrored. Therefore, the images that will be mirrored are supposed to have some distinctions compared to the ‘site’. This is the reason why the buildings need to be represented with ‘boxes’ rather than ‘shapes’ in the drawing. This way, the plug-in knows which images to mirror.

5.4 Plug-in Flow

As it was briefly described previously, the plug-in is designed to take mirror images of what it sees across the street from the site, onto four different sides of the site and generate creative results by combining all possible depth values that each building can take. Initially, the plug-in reflects the mirror image of the neighbour buildings across the street on each side of the site with an initial depth value which is defined by the user. Here, the user needs to take the height and width values of the building into consideration and define the initial depth value accordingly. The plug-in will assume that the user will enter a valid proportion value in terms of statics of the building and use this value among the alternatives. This means that if the initial depth proportion does not correspond to the static values of the design, some generated alternatives among the end result set will be invalid and may lead to inconsistencies in further phases of the project.

5.4.1Setting the Increment Value

In the next step, the plug-in needs another value before it starts to compute all the combinations that can take place on the site. This value is the increment value. An increment value is the value by which the initial depth value will be increased for each combination. Initially this increment value is calculated by the plug-in as 1/10th of each side of the site. 1/10th of the side that lies on the x axis is calculated and defined as the increment value of those blocks placed on the other side of the site that lies on the y axis since those blocks will be gaining depth following the distance of the side that lies on the x axis. Similarly, /10th of the side that lies on the y axis is calculated and defined as the increment value of those blocks placed on the other side of the site that lies on the x axis since those blocks will be gaining depth following the distance of the side that lies on the y axis . This is the minimum value that the initial value can take and this value can be increased by the user. However, it can not be decreased because decreasing the increment value would result in more alternatives which would not have great distictions being generated. The time that it requires to compute a set of alternatives increases as the number of alternatives in the set increases. It is best to optimize the relation between the alternative number and the time it will take to calculate which leads to optimization of the increment value. If the increment value is too high, the plug-in will need less time to generate an

alternative set; but some potentially good alternatives will be skipped because of a rough generation approach caused by a too high of an increment value.

5.4.2 Filters

In the following step, the plug-in generates all the alternatives starting with the initial depth values and by applying the increment value. However, it would be too cumbersome to store all the alternatives. Therefore, the plug-in runs a set of filters in alliance with the designer's needs and the buildng funcion's and stores the alternatives which pass the filter. Thus, the set of alternatives become more effective and since the unnecessary alternatives are eliminated, the set holds less space on the computer. Applying filters makes it possible to eliminate an alternative that might be intriguing to the designer. Therefore, the filters should be applied carefully so that potentially good alternatives are not eliminated before the designer takes it into consideration.

The first filter helps to preserve symmetry between a given elevation and the buildings across the street. As depth values of buildings on the site increase the building blocks start to merge towards the center of the site and get past each other at a certain point getting closer to the opposite end of the site. Once the depth value arrives at the other end of the site, the building is flush with the reflected building blocks of that particular side of the site, ruining the symmetry. The plug-in filters the results to prevent symmetry for all sides of the site. The plug-in achieves this by offseting the site boundary inwards so that the building blocks can travel only so far into the site. This process can be visualised as a smaller rectangle placed into a larger rectangle with the same proportions or represented as follows:

Shape limit:
x1y1, x2y2, x3y3, x4y4
x1+a=x2=x3 x1=x4
y1=y2 y2-b=y3=y4

Shape limit m:
c<a x1m+cx2m=x3m
d>b x1m=x4m
y1m=y2m y2m-d=y3m=y4m

The user has the opportunity to enter how many building blocks the plug-in should allow on the site. According to the needs of the design, the designer might prefer to

avoid building blocks in pieces but as a whole larger unit which is not intersected with outside passages. This way, the plug-in only saves the alternatives that consists of specified number of building blocks and eliminates the one that do not meet the number of buildings that is required.

Another filter that may be applied to generated results, is “approach limit” . This filter defines the maximum distance that a building block can approach to one and another. The maximum distance depends on the type of building that is to be designe. If the design is for an administration building where amount of daylight filterin into the building is essential for the well being of employers, the approach limit is greater compared to an art gallery where lack of daylight can be compansated with artificial light or even such a position is desired for the sake of an exhibition.

Another useful filter that can be applied is the one that allows the user to define the total footprint area needed. Every project has its requirements and limits. The most common limit of a project is the total construction area. As the depth values of building blocks on the site increase, the total area of footprint on the site increases as well. If this filter is activated and a value is defined, the plug-in will only save the alternatives that meet the square meter requirement.

Another set of filters allows the plug-in to relate to knowledge based systems by offering a library which consists of architectural standards related to specific types such as public, residential, administrative ect. It allows the user to choose the right values for the minimum distance between two specific type of buildings and recommended proportion of open space vs built space for a given type of building.

The last feature of the plug-in is that it alllows the designer to double the number of alternatives by clicking the ‘inverse’ option which subtacts the built area from the site and extrudes the result. This operation allows for the inverse area of an alternative to be extruded giving the designer opportunity of exploring new related options.

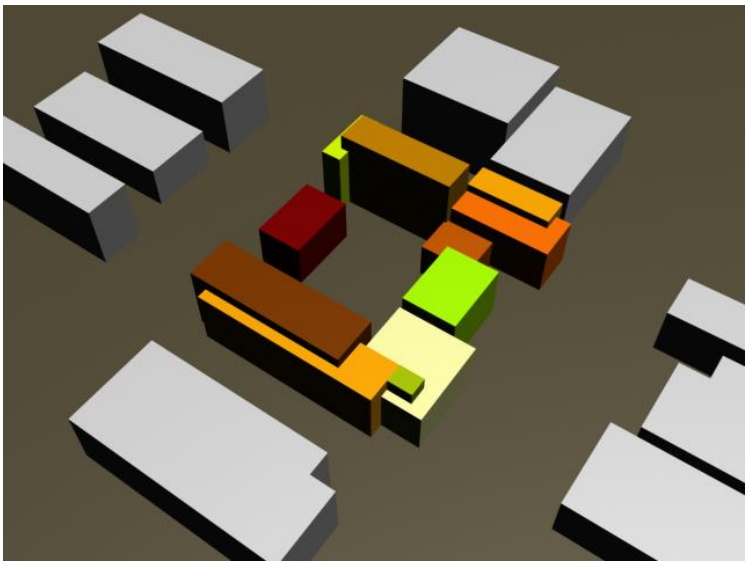


Figure 5.4 : Possible positive area results of the plug-in of an urban block.

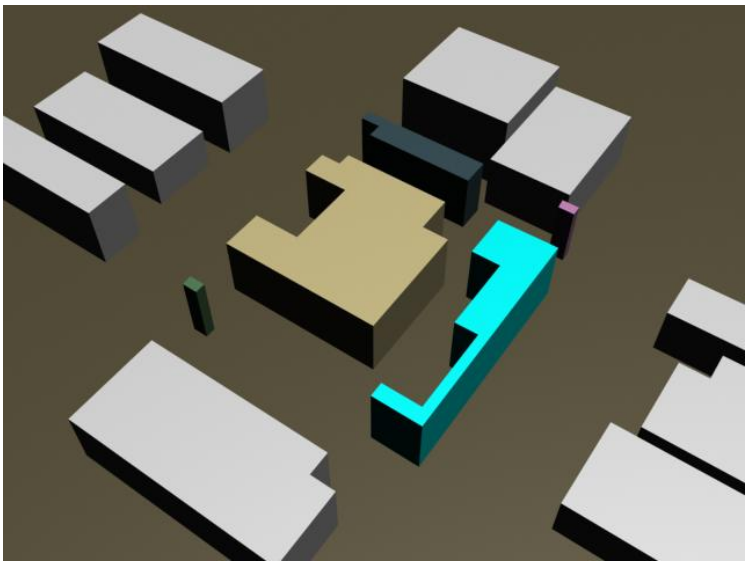


Figure 5.5 : Possible negative area results of the plug-in of an urban block.

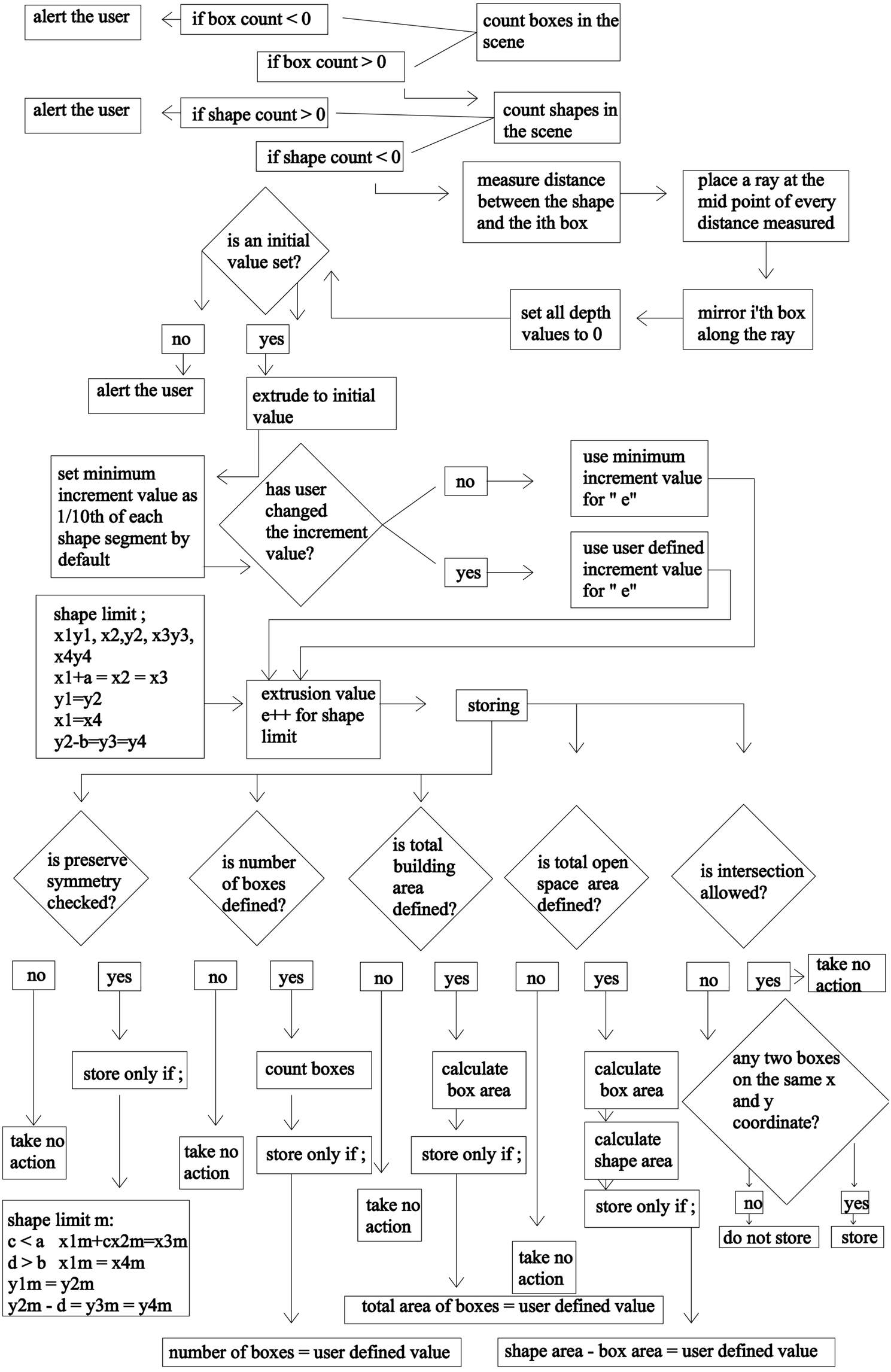


Figure 5.6 : Proposed plug-inflow

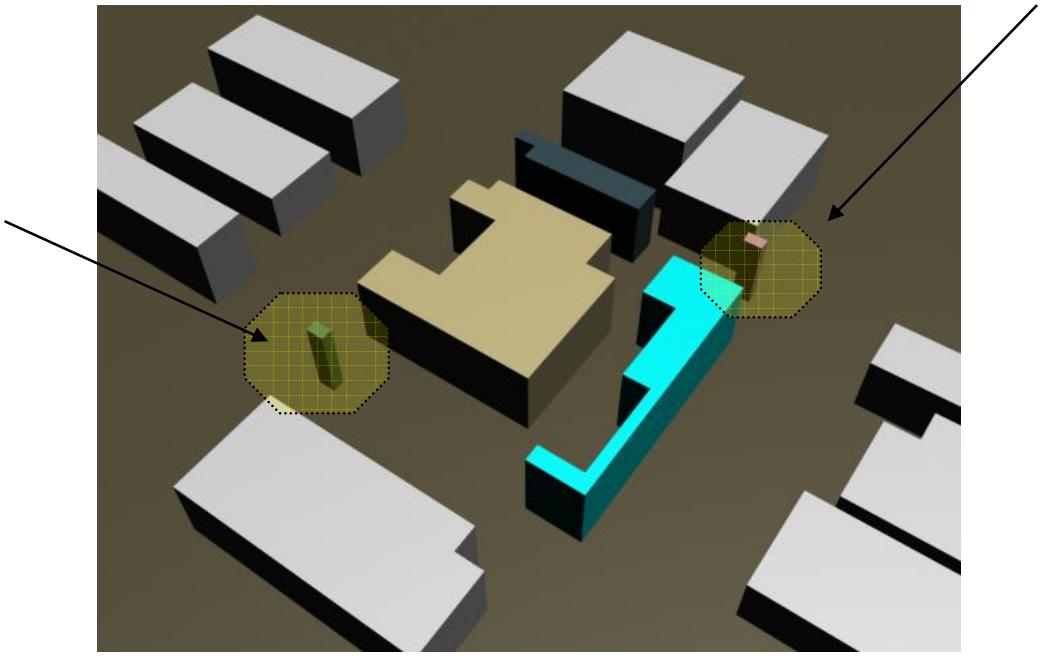


Figure 5.7 : Possible negative area results of the plug-in of an urban block with not preffered building pieces on two opposite corners.

Another handicap of the 'inverse' option is that the generated buildig blocks all have the same preset extrusion level because this particular process does not have anything to refer to in terms of height. In the future, the users will be able to set rules for extrusion during 'inverse' process and set the height values manually.

5.5 Approach 1: MAXScript

Autodesk 3D Studio Max is promoted as production-proven 3D modeling, animation and rendering solution for games, film, television and digital publishing by Autodesk. Along with 3D modeling, animation and rendering features, MAXScript is also used by many users. MaxScript is an open interface for customizing and scripting 3ds Max. Most features of 3ds Max can be accessed by scripting. Operations can be automated and user interface can be extended.

In this thesis, the design problem and proposed process of achieving it are primarily attempted to be established by using Autodesk 3D Studio Max software. MAXScript has been used to generate design alternatives and animation has been used to store the generated design alternatives where all can be seen consequently and compared easily.

5.5.1 Script Functions

In this section, the functions created and called into the script to generate design alternatives will be explained and linked to their proper positions within the script.
The site on which the environment building elevations will be mirrored is required to be drawn as a rectangle and as the very first object in the scene so that it can be located with the specific called the “getLocationobj rect” function without ever needing to distinguish the site among other drawn objects.

The next step defined allows for detecting the side on which the environment building mass stand relative to the site which is drawn as a rectangle and in the area surrounded by the environment building mass. The four sides of the site has been assigned numbers to. If the object is found on the upper part of the site, it is assigned under region 1. If the object is found on the lower side of the site, it is assigned under region 3. If the object is found on the right side of the site, it is assigned under region 2. If the object is found on the left side of the site, it is assigned under region 4. This list will be called into the script during mirroring process and simplify this step. The command will read as “mirror region 4 along the specified axis” rather than finding every single object and determining along which axes it needs to be mirrored about.

The following function called “getBackKnotsobj” finds from which quadrant the building mass object has been mirrored from and holds the index of rear two vertex points which depend on the root quadrant. This infromation needs to be used in the main script during the process of listing all the combinations possible when altering depth values of the mirrored building mass.

The next function called “generateCombMatrix n value” determines how many environment buildings are present in the scene, finds out how many different values that a mirrored building mass can take and holds a matrix of those possible values. This function will be called into the main script after the locations of the rear vertices of the mirrored object are found and they need to be re-positioned according to the matrix created by this function.

The next function takes the arrays in every quadrant matrix along with the site limits. Then it saves each combination as a frame in the animation.

The next functions called as a part of the main script into which functions mentioned earlier are called. The following piece puts the objects into arrays according to their quadrant number expained above.

The following four pieces of the main script plays similar roles hwever in different directions. The first one takes the corresponding array just created and mirrores it on the positive y direction. Then the second takes the corresponding array just created and mirrores it on the negative y direction. The third one takes the corresponding array just created and mirrores it on the positive x direction. The last one takes the corresponding array just created and mirrores it on the negative x direction.

The final part of the main script adds the mirrored objects the neccessary extrusion value. After this step, it displays the combinations as an animation.

A figure representing the relations between the functions and the script flow is shown below. The reader of this thesis can find the whole set of functions and the code as labeled in the figure in the appendix section.

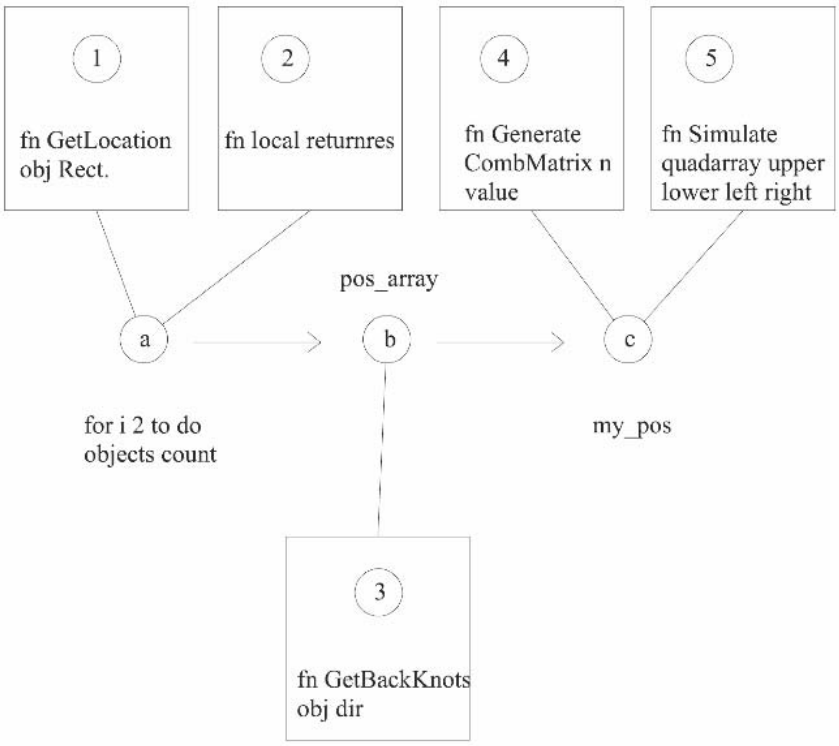


Figure 5.8 : Max function relations

5.5.2 Steps and Examples From a Generated Design

As explained along with the functions, the user needs to draw the site as a rectange first. Then the environment buildings should be drawn as shapes and orthagonally only then extrusion modifier has to be added as the next step.

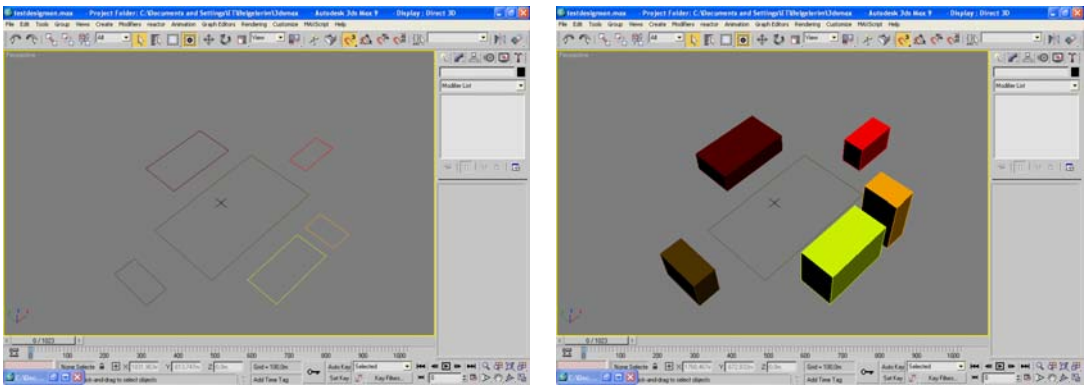


Figure 5.9 : Preparing the scene to work with MAXScript.

Then the script file needs to be opened and from the file menu of the script, evaluate all tab needs to be clicked.

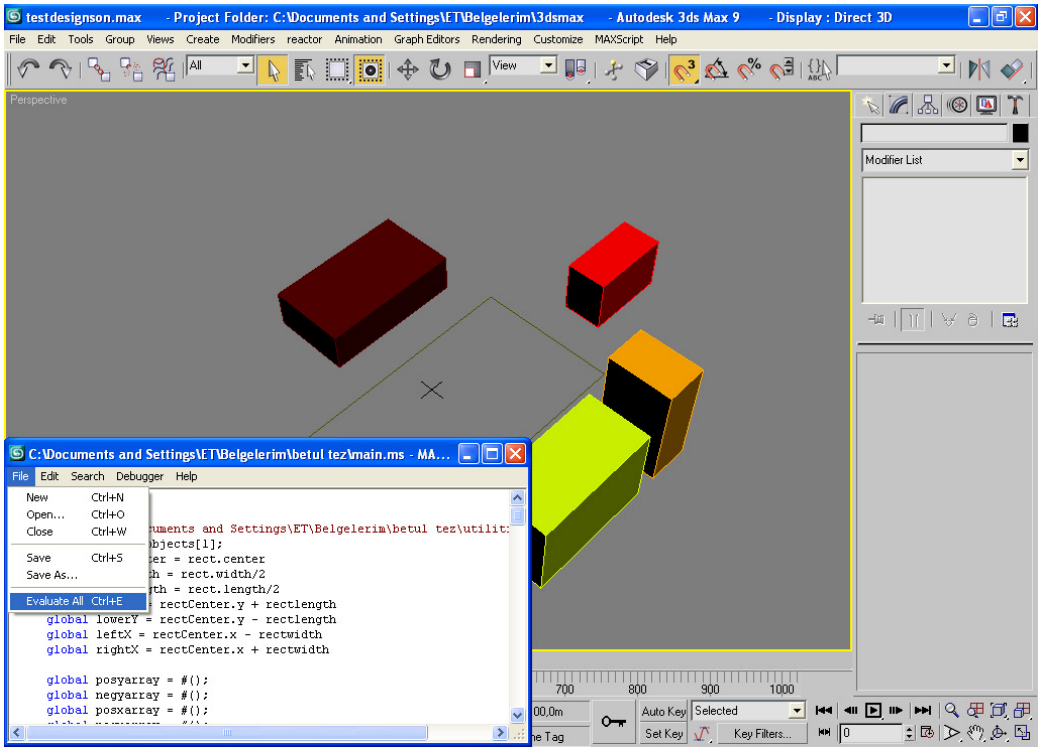


Figure 5.10 : Running MAXScript.

The plugin takes several seconds to proccess the script and creates the animation and the first alternative which is the exact mirror of environment buildings are displayed.

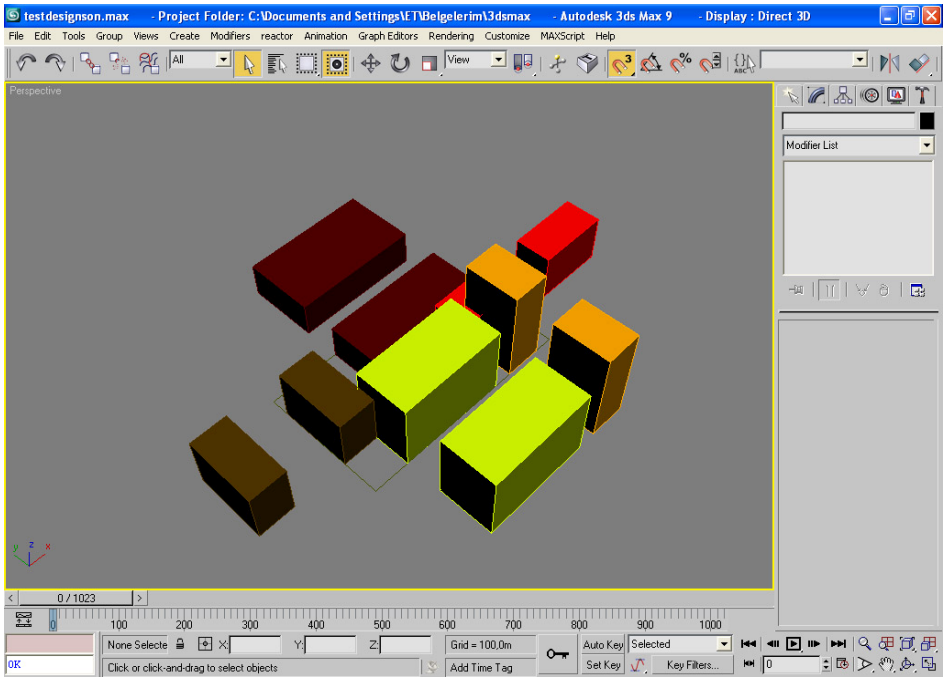


Figure 5.11 : Results of the plug-in of an urban block generated with MAXScript.

As the user drags the time line scroll bar in the animation part to the lower side of the scene, all the 1024 generated alternatives can be viewed.

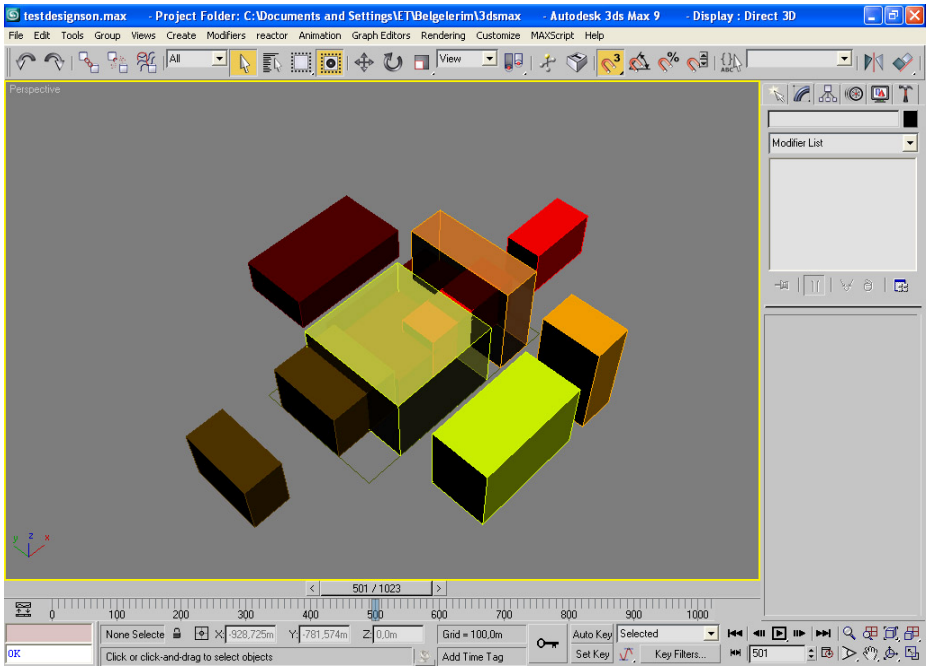


Figure 5.12 : Results of the plug-in of an urban block generated with MAXScript.

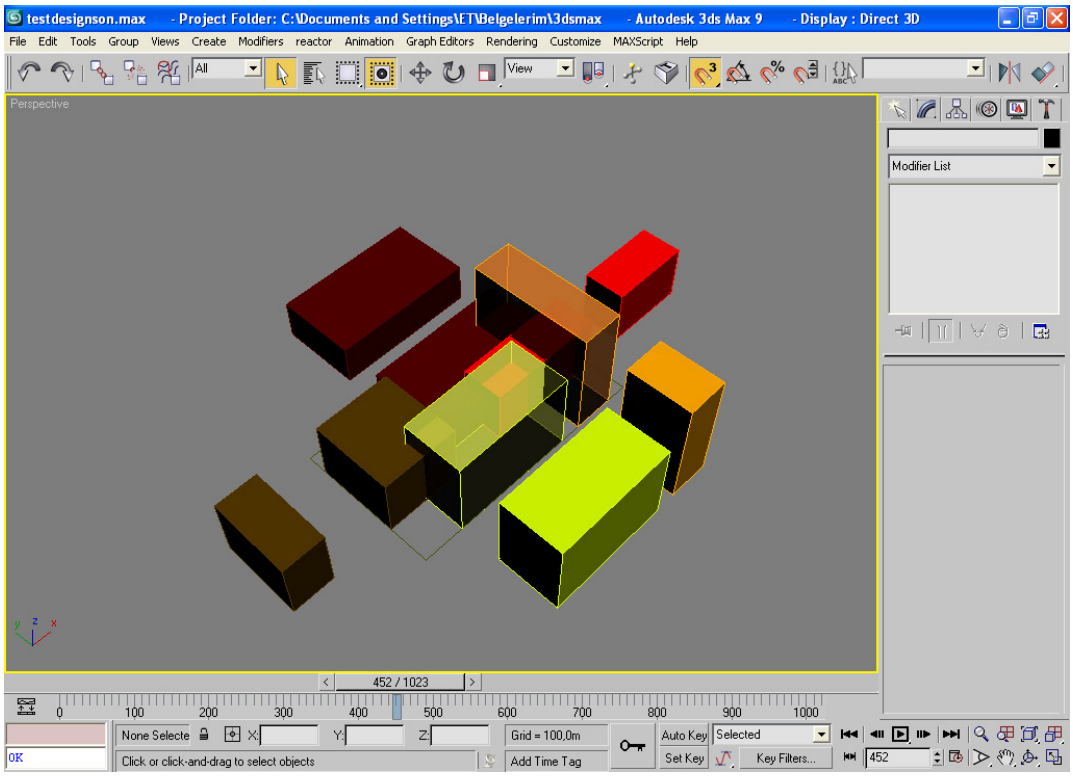


Figure 5.13 : results of the plug-in of an urban block generated with MAXScript.

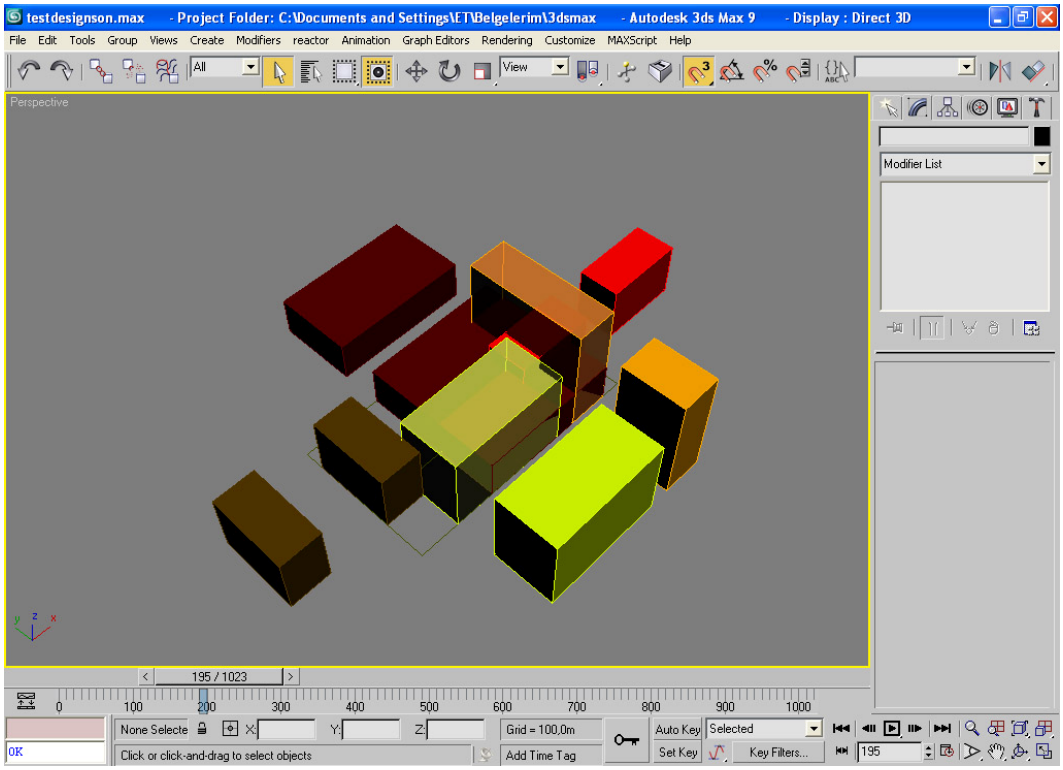


Figure 5.14 : Results of the plug-in of an urban block generated with MAXScript.

5.5.3 Advantage and Disadvantage of MAXScripted Approach

Although 3D Studio Max is a widely used software among architects and designers, in this case, it has not allowed for much flexibility in terms of being able to manipulate the parameters in such a way that the set of alternatives serve to the specified purpose.
Another disadvantage of Max run plug-in is that as the number of building increases, it hits the computational boundaries. Since at the moment there is no way of eliminating undesired alternatives, the plug-in generates and saves all the possible alternatives causing the set to contain too high numbers of alternatives. As the possible design number increases, errors ocur and the software might terminate.
Along with some disadvantages, being able to save all the generated alternatives in an animation allows for a compact way of storing the models and being able to browse among them easily.

5.6 Approach 2: Catia & Excel

The second approach to the problem has been accomplished through using Microsoft Excel to generate and store values necessary to be able to draw the mass models; and Catia has been used to visualize the model by linking the Excel Design Table. Below is the file created in Excel where the user can create an imaginary environment by using randomly generated values, adjust the proportions of the buildings, how they are distributed on the edge or enter real dimensions of a measured environment manually; the user can also generate all the possible depth values that the recently created urban block elements can take as well as the minimum and the maximum value range between which the depth values are picked.

5.6.1 Excel Design Table in Detail

The first column in the Excel Design Table, belongs to titles defining which entity will be affected if the consequent row value is changed. This column is divided into three groups: height and with values of the environment and the buildings to be generated, the depth value of the buildings to be generated and the depth value of the environment buildings. These groups and their features will be explained in detail. The second column belongs to the numeric representation of the values that Catia

will read and draw in the final stage. The third column belongs to the numeric representation of the decision weather to generate the corresponding value randomly or to enter it manually. If the box reads 0, the corresponding entity's value needs to be entered manually. If the box reads 1, the corresponding entity's value will be generated randomly and will update and change upon hitting F9 or clicking anywhere on the design table. The fourth column enables the user to enter values manually if the column number three is marked as 0. Column number five stores the values entered in the fourth column and sends them to column number two if, column number three is marked as 0. Column number six and seven define the minimum and the maximum value that an entity can get when a random value needs to be assigned. Column number eight reads and holds the values generated randomly and sends these values to column number two when column number three is marked as 1.

Since width and height values will remain the same and the only value that will show variation is the depth of the generated elements, the column is divided into three groups: height and width value of both environment and mirrored objects, depth value of generated objects and depth value of environment buildings. The first group in the first column refers to width and height values of the environment buildings and the elements of the urban block to be generated. A differentiation of these dimensions has not been made since the elements of the generated urban block will be mirrored images of the environment therefore be the same and will not be subject to any change. The values in this part of the column can be manually entered or randomly generated. However, if they are generated randomly, the values need to be copied and pasted to column number four and the value 1which means generate values randomly in column number three needs to be switched to 0 which means manual input. Otherwise, the width and height values of the environment buildings will be changed every time the table is updated. Randomness is needed in building the environment buildings for only once. When the environment buildings are created, the values need to be stabilized and the only changing value needs to be the depth value of generated buildings.

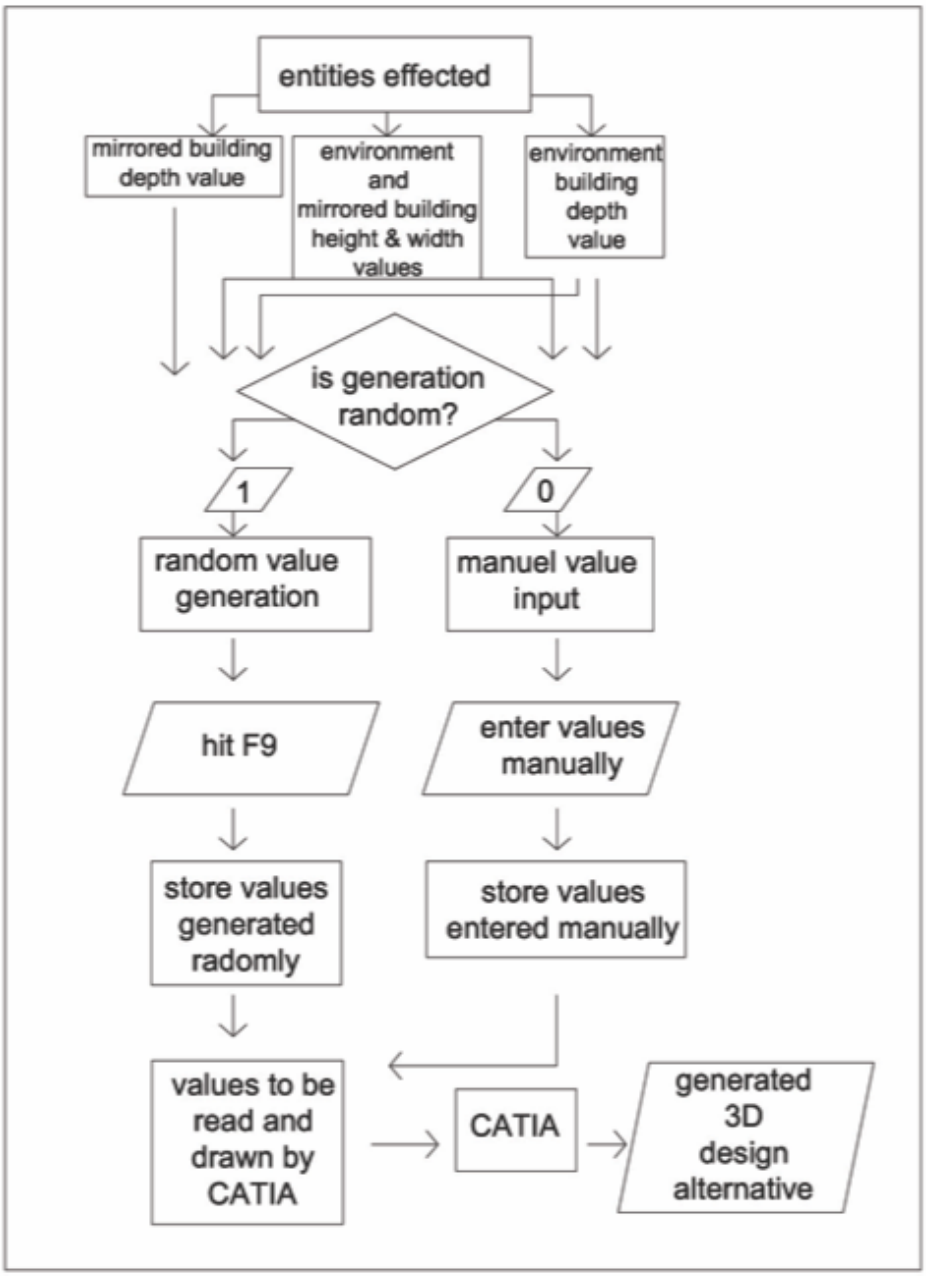


Figure 5.15 : System Flow

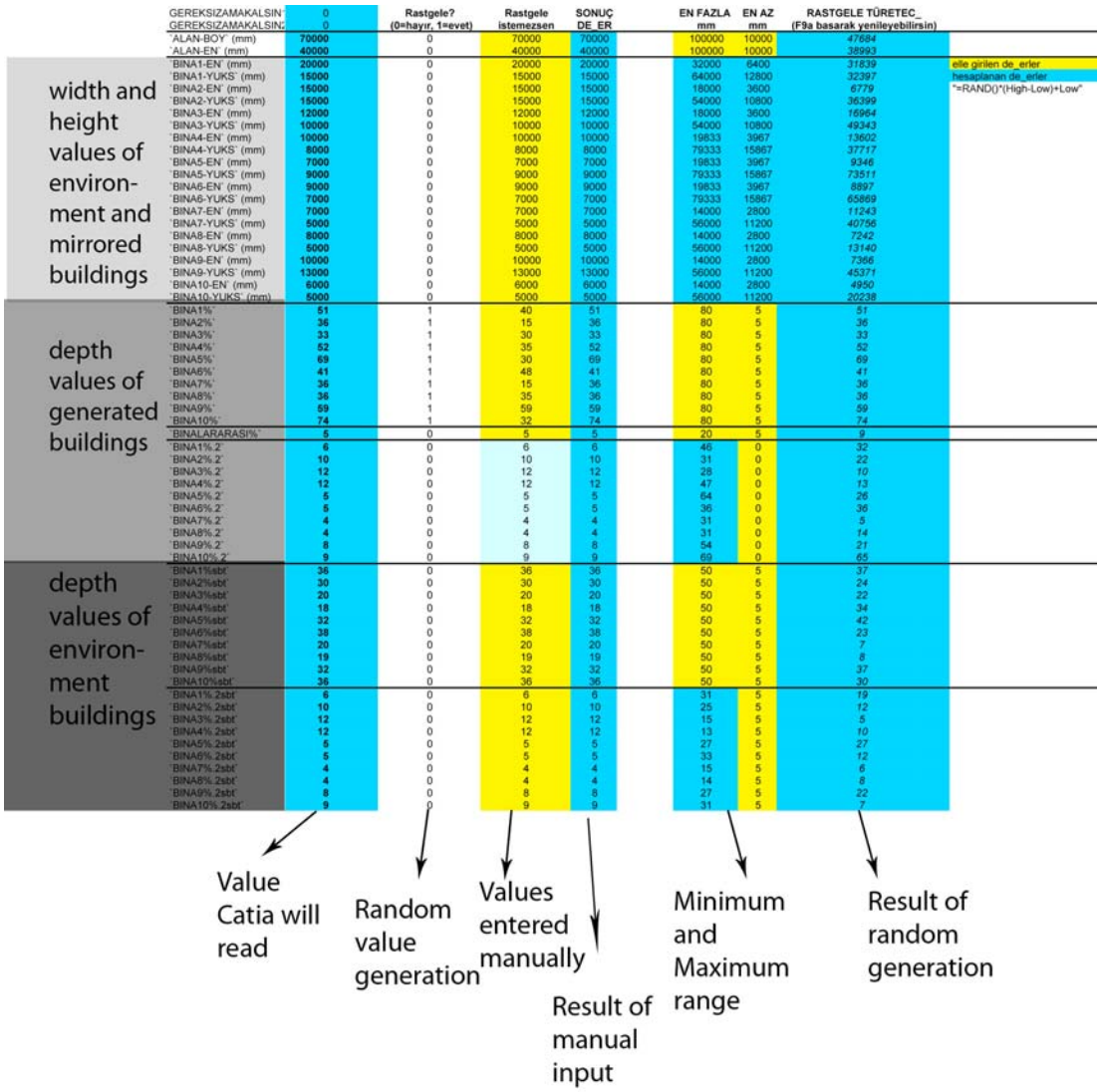


Figure 5.16 : Excel Design Table

The second group in the first column is the width value of generated/mirrored objects. This column has a minimum and maximum range linked located at column number six and seven, from which a value is picked randomly upon hitting F9 or clicking anywhere on the table to refresh the table. The third group consists of depth values of the surrounding buildings. These values can be generated randomly or entered manually as well. If they are generated randomly, the values need to be copied and pasted to column number four and the value 1which means generate values randomly in column number three needs to be switched to 0 which means manual input. As explained before, this helps an imaginary random environment to be created and then stabilized so that the environment building dimensions do not change during generation of the new urban block.

When the designer changes the values as explained and needs to visualize the parameters that are entered, Catia needs to be fired with the specific Catia file that is linked to Excel and consists of a pre-designed scene. Then the Excel file needs to be saved in order to Catia be able to detect the changes in the design table. Once the Excel file is saved and the window is switched to Catia, it will bring up a message box stating that changes have been made to Excel Design File. Once the user closes this box and hits the update button, Catia reads from the file, calculates and draws a new scene. The next step of the process is to go back to the Excel file and hit F9 or double click anywhere so that the randomly changing depth values of the new blocks to be generated are updated. Once the user updates the file, saves it and switches to Catia, the same message box stating that changes has been don to the design table will show up and then the user can hit the update button to visualize the new scene. Below is the flow of the process summarized graphically with a flow chart followed by some examples of generated design alternatives and explanations of how they are accomplished. In the next section.

5.6.2 Applied Examples and Steps

a b	0 0	Random? (0=no, 1=yes)	Manual	Result Value	Max mm	Min mm	Random (F9)
`site-d.` (mm)	**70000**	0	70000	70000	100000	10000	*72599*
`site-w.` (mm)	**40000**	0	40000	40000	100000	10000	*24271*
`bldg.1-w.` (mm)	**18219**	1	12841	18219	32000	6400	*18219*
`bldg.1-h.` (mm)	**28422**	1	20495	28422	64000	12800	*28422*
`bldg.2-w.` (mm)	**14606**	1	8205	14606	18000	3600	*14606*
`bldg.2-h.` (mm)	**38987**	1	49229	38987	54000	10800	*38987*
`bldg.3-w.` (mm)	**5985**	1	13060	5985	18000	3600	*5985*
`bldg.3-h.` (mm)	**45317**	1	45985	45317	54000	10800	*45317*
`bldg.4-w.` (mm)	**7619**	1	4629	7619	19833	3967	*7619*
`bldg.4-h.` (mm)	**35660**	1	37920	35660	39667	7933	*35660*
`bldg.5-w.` (mm)	**6278**	1	12260	6278	19833	3967	*6278*
`bldg.5-h.` (mm)	**29094**	1	19781	29094	39667	7933	*29094*
`bldg.6-w.` (mm)	**8370**	1	4327	8370	19833	3967	*8370*
`bldg.6-h.` (mm)	**33739**	1	18856	33739	39667	7933	*33739*
`bldg.7-w.` (mm)	**10620**	1	8476	10620	14000	2800	*10620*
`bldg.7-h.` (mm)	**23177**	1	6539	23177	28000	5600	*23177*
`bldg.8-w.` (mm)	**11728**	1	11306	11728	14000	2800	*11728*
`bldg.8-h.` (mm)	**12372**	1	14684	12372	42000	8400	*12372*
`bldg.9-w.` (mm)	**5061**	1	2960	5061	14000	2800	*5061*
`bldg.9-h.` (mm)	**27490**	1	7239	27490	28000	5600	*27490*
`bldg.10-w.` (mm)	**9830**	1	13253	9830	14000	2800	*9830*
`bldg.10-h.` (mm)	**21263**	1	5919	21263	28000	5600	*21263*
`bldg.1%`	**46**	1	40	46	80	20	*46*
`bldg.2%`	**73**	1	15	73	80	20	*73*
`bldg.3%`	**58**	1	30	58	80	20	*58*
`bldg.4%`	**43**	1	35	43	80	20	*43*
`bldg.5%`	**31**	1	30	31	80	20	*31*
`bldg.6%`	**32**	1	48	32	80	20	*32*
`bldg.7%`	**71**	1	15	71	80	20	*71*
`bldg.8%`	**43**	1	35	43	80	20	*43*
`bldg.9%`	**42**	1	59	42	80	20	*42*
`bldg.10%`	**60**	1	32	60	80	20	*60*

`bldg.in btw%`	**5**	0	5	5	20	5	*5*
`bldg.1%.2`	**6**	0	6	6	41	0	*11*
`bldg.2%.2`	**10**	0	10	10	68	0	*34*
`bldg.3%.2`	**12**	0	12	12	53	0	*1*
`bldg.4%.2`	**12**	0	12	12	38	0	*22*
`bldg.5%.2`	**5**	0	5	5	26	0	*10*
`bldg.6%.2`	**5**	0	5	5	27	0	*12*
`bldg.7%.2`	**4**	0	4	4	66	0	*31*
`bldg.8%.2`	**4**	0	4	4	38	0	*10*
`bldg.9%.2`	**8**	0	8	8	37	0	*21*
`bldg.10%.2`	**9**	0	9	9	55	0	*43*
`bldg.1%env.`	**36**	0	36	36	50	5	*20*
`bldg.2%env.`	**30**	0	30	30	50	5	*26*
`bldg.3%env.`	**20**	0	20	20	50	5	*25*
`bldg.4%env.`	**18**	0	18	18	50	5	*46*
`bldg.5%env.`	**32**	0	32	32	50	5	*10*
`bldg.6%env.`	**38**	0	38	38	50	5	*12*
`bldg.7%env.`	**20**	0	20	20	50	5	*46*
`bldg.8%env.`	**39**	0	39	39	50	5	*13*
`bldg.9%env.`	**32**	0	32	32	50	5	*31*
`bldg.10%env.`	**36**	0	36	36	50	5	*30*
`bldg.1%.2env.`	**6**	0	6	6	31	5	*15*
`bldg.2%.2env.`	**10**	0	10	10	25	5	*10*
`bldg.3%.2env.`	**12**	0	12	12	15	5	*10*
`bldg.4%.2env.`	**12**	0	12	12	13	5	*7*
`bldg.5%.2env.`	**5**	0	5	5	27	5	*14*
`bldg.6%.2env.`	**5**	0	5	5	33	5	*5*
`bldg.7%.2env.`	**4**	0	4	4	15	5	*6*
`bldg.8%.2env.`	**4**	0	4	4	34	5	*16*
`bldg.9%.2env.`	**8**	0	8	8	27	5	*25*
`bldg.10%.2env.`	**9**	0	9	9	31	5	*13*

Table 5.1 : Generation of Environment Blocks Randomly in Excel

The image above, shows the first step of the design process where the environment needs to be created with random values. Therefore, column number three needs to read 1 for the entities which need random values. Once the corresponding rows of this column read 1, the user needs to update the table in order to obtain randomly generated values. Then the values in column two should be copied and "special pasted" and "value" option should be chosen so that only values are replaced, not the formula.

a	0	**Random? (0=no, 1=yes)**	**Manual**	**Result**	**Max**	**Min**	**Random**
b	0			**Value**	**mm**	**mm**	**(F9)**
`site-d.` (mm)	**70000**	0	70000	70000	100000	10000	*92467*
`site-w.` (mm)	**40000**	0	40000	40000	100000	10000	*30491*
`bldg.1-w.` (mm)	**18219**	0	18219	18219	32000	6400	*12866*
`bldg.1-h.` (mm)	**28422**	0	28422	28422	64000	12800	*55691*
`bldg.2-w.` (mm)	**14606**	0	14606	14606	18000	3600	*8425*
`bldg.2-h.` (mm)	**38987**	0	38987	38987	54000	10800	*20633*
`bldg.3-w.` (mm)	**5985**	0	5985	5985	18000	3600	*11797*
`bldg.3-h.` (mm)	**45317**	0	45317	45317	54000	10800	*35175*
`bldg.4-w.` (mm)	**7619**	0	7619	7619	19833	3967	*11809*
`bldg.4-h.` (mm)	**35660**	0	35660	35660	39667	7933	*14830*
`bldg.5-w.` (mm)	**6278**	0	6278	6278	19833	3967	*15684*
`bldg.5-h.` (mm)	**29094**	0	29094	29094	39667	7933	*37041*
`bldg.6-w.` (mm)	**8370**	0	8370	8370	19833	3967	*15034*
`bldg.6-h.` (mm)	**33739**	0	33739	33739	39667	7933	*21549*
`bldg.7-w.` (mm)	**10620**	0	10620	10620	14000	2800	*3094*
`bldg.7-h.` (mm)	**23177**	0	23177	23177	28000	5600	*14649*
`bldg.8-w.` (mm)	**11728**	0	11728	11728	14000	2800	*11290*
`bldg.8-h.` (mm)	**12372**	0	12372	12372	42000	8400	*23816*
`bldg.9-w.` (mm)	**5061**	0	5061	5061	14000	2800	*4855*
`bldg.9-h.` (mm)	**27490**	0	27490	27490	28000	5600	*24068*
`bldg.10-w.` (mm)	**9830**	0	9830	9830	14000	2800	*7570*

`bldg.10-h.` (mm)	**21263**	0	21263	21263	28000	5600	*7678*
`bldg.1%`	**48**	1	40	48	80	20	*48*
`bldg.2%`	**69**	1	15	69	80	20	*69*
`bldg.3%`	**47**	1	30	47	80	20	*47*
`bldg.4%`	**25**	1	35	25	80	20	*25*
`bldg.5%`	**68**	1	30	68	80	20	*68*
`bldg.6%`	**34**	1	48	34	80	20	*34*
`bldg.7%`	**60**	1	15	60	80	20	*60*
`bldg.8%`	**24**	1	35	24	80	20	*24*
`bldg.9%`	**32**	1	59	32	80	20	*32*
`bldg.10%`	**53**	1	32	53	80	20	*53*
`bldg.in btw%`	**5**	0	5	5	20	5	*15*
`bldg.1%.2`	**6**	0	6	6	43	0	*24*
`bldg.2%.2`	**10**	0	10	10	64	0	*50*
`bldg.3%.2`	**12**	0	12	12	42	0	*27*
`bldg.4%.2`	**12**	0	12	12	20	0	*16*
`bldg.5%.2`	**5**	0	5	5	63	0	*32*
`bldg.6%.2`	**5**	0	5	5	29	0	*9*
`bldg.7%.2`	**4**	0	4	4	55	0	*50*
`bldg.8%.2`	**4**	0	4	4	19	0	*19*
`bldg.9%.2`	**8**	0	8	8	27	0	*18*
`bldg.10%.2`	**9**	0	9	9	48	0	*40*
`bldg.1%env.`	**36**	0	36	36	50	5	*29*
`bldg.2%env.`	**30**	0	30	30	50	5	*25*
`bldg.3%env.`	**20**	0	20	20	50	5	*37*
`bldg.4%env.`	**18**	0	18	18	50	5	*34*
`bldg.5%env.`	**32**	0	32	32	50	5	*35*
`bldg.6%env.`	**38**	0	38	38	50	5	*39*
`bldg.7%env.`	**20**	0	20	20	50	5	*48*
`bldg.8%env.`	**39**	0	39	39	50	5	*22*
`bldg.9%env.`	**32**	0	32	32	50	5	*19*
`bldg.10%env.`	**36**	0	36	36	50	5	*15*
`bldg.1%.2env.`	**6**	0	6	6	31	5	*7*
`bldg.2%.2env.`	**10**	0	10	10	25	5	*15*
`bldg.3%.2env.`	**12**	0	12	12	15	5	*12*
`bldg.4%.2env.`	**12**	0	12	12	13	5	*12*
`bldg.5%.2env.`	**5**	0	5	5	27	5	*24*
`bldg.6%.2env.`	**5**	0	5	5	33	5	*31*
`bldg.7%.2env.`	**4**	0	4	4	15	5	*8*
`bldg.8%.2env.`	**4**	0	4	4	34	5	*9*
`bldg.9%.2env.`	**8**	0	8	8	27	5	*19*
`bldg.10%.2env.`	**9**	0	9	9	31	5	*27*

Table 5.2 : Stabilizing Generated Environment Buildings in Excel

Then the corresponding rows in column three needs to be turned into 0 so that column number two takes the values in column number five where the results for manually enter values are held and those values are read by Catia. In this case, depth values of the environment buildings are not generated randomly and it can be observed through a comparison in column three in the image above.

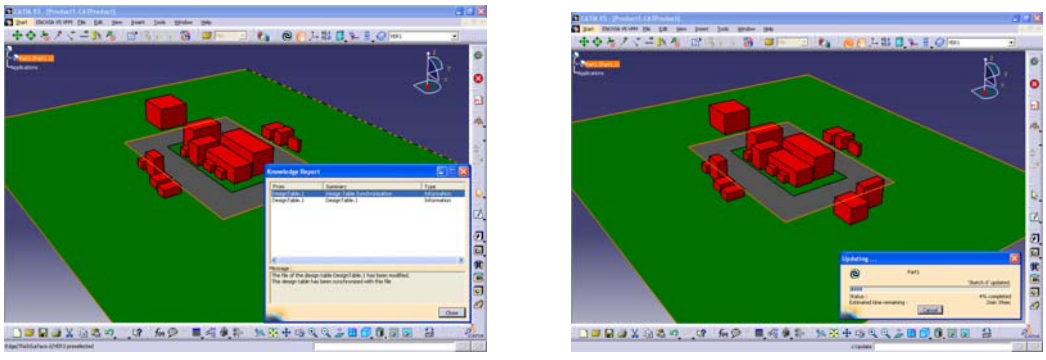


Figure 5.17 : Linking Excel Design Table to Catia and Scene Update.

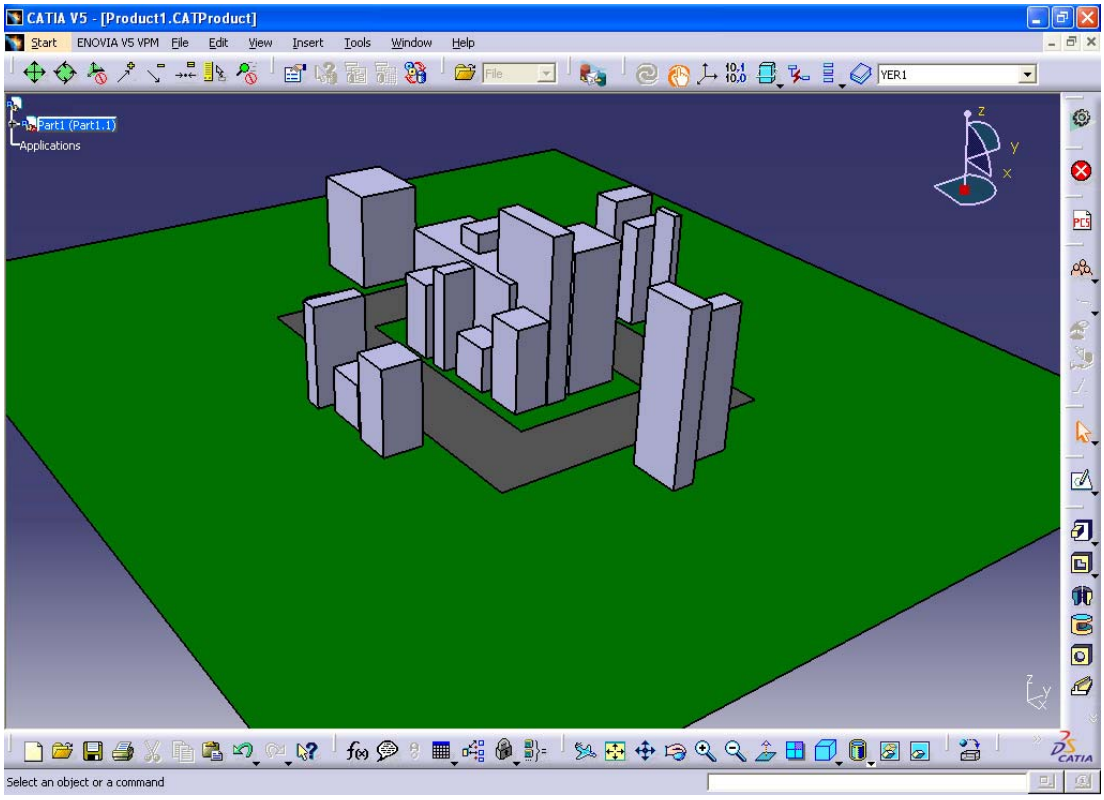


Figure 5.18 : Possible results of the plug-in of an urban block generated in Catia.

The images above show the steps after the Excel file has been saved. When the user switches back to Catia, it brings up a box where the program states that changes has been made to the Excel file and the final image shows the scene generated after the user has hit the update button.

The next table shows regeneration of random depth values of the mass objects to be generated on the new urban block. This step is accomplished by simply switching back to the Excel file and hitting either F9 or clicking anywhere on the table.

a b	0 0	Random? (0=no, 1=yes)	Manual	Result Value	Max mm	Min mm	Random (F9)
`site-d.`	**70000**	0	70000	70000	100000	10000	*39072*
`site-w.`	**40000**	0	40000	40000	100000	10000	*68761*
`bldg.1-w.`	**18219**	0	18219	18219	32000	6400	*15250*
`bldg.1-h.`	**28422**	0	28422	28422	64000	12800	*24478*
`bldg.2-w.`	**14606**	0	14606	14606	18000	3600	*5803*
`bldg.2-h.`	**38987**	0	38987	38987	54000	10800	*40520*
`bldg.3-w.`	**5985**	0	5985	5985	18000	3600	*14303*
`bldg.3-h.`	**45317**	0	45317	45317	54000	10800	*13483*
`bldg.4-w.`	**7619**	0	7619	7619	19833	3967	*5388*
`bldg.4-h.`	**35660**	0	35660	35660	39667	7933	*33918*
`bldg.5-w.`	**6278**	0	6278	6278	19833	3967	*18768*
`bldg.5-h.`	**29094**	0	29094	29094	39667	7933	*11404*

`bldg.6-w.`	**8370**	0	8370	8370	19833	3967	*8164*
`bldg.6-h.`	**33739**	0	33739	33739	39667	7933	*38651*
`bldg.7-w.`	**10620**	0	10620	10620	14000	2800	*12745*
`bldg.7-h.`	**23177**	0	23177	23177	28000	5600	*23276*
`bldg.8-w.`	**11728**	0	11728	11728	14000	2800	*3994*
`bldg.8-h.`	**12372**	0	12372	12372	42000	8400	*38223*
`bldg.9-w.`	**5061**	0	5061	5061	14000	2800	*4169*
`bldg.9-h.`	**27490**	0	27490	27490	28000	5600	*6157*
`bldg.10-w.`	**9830**	0	9830	9830	14000	2800	*9703*
`bldg.10-h.`	**21263**	0	21263	21263	28000	5600	*9138*
`bldg.1%`	**42**	1	40	42	80	20	*42*
`bldg.2%`	**25**	1	15	25	80	20	*25*
`bldg.3%`	**44**	1	30	44	80	20	*44*
`bldg.4%`	**37**	1	35	37	80	20	*37*
`bldg.5%`	**55**	1	30	55	80	20	*55*
`bldg.6%`	**79**	1	48	79	80	20	*79*
`bldg.7%`	**54**	1	15	54	80	20	*54*
`bldg.8%`	**76**	1	35	76	80	20	*76*
`bldg.9%`	**75**	1	59	75	80	20	*75*
`bldg.10%`	**70**	1	32	70	80	20	*70*
`bldg.in btw%`	**5**	0	5	5	20	5	*14*
`bldg.1%.2`	**6**	0	6	6	37	0	*31*
`bldg.2%.2`	**10**	0	10	10	20	0	*1*
`bldg.3%.2`	**12**	0	12	12	39	0	*30*
`bldg.4%.2`	**12**	0	12	12	32	0	*29*
`bldg.5%.2`	**5**	0	5	5	50	0	*36*
`bldg.6%.2`	**5**	0	5	5	74	0	*7*
`bldg.7%.2`	**4**	0	4	4	49	0	*20*
`bldg.8%.2`	**4**	0	4	4	71	0	*8*
`bldg.9%.2`	**8**	0	8	8	70	0	*23*
`bldg.10%.2`	**9**	0	9	9	65	0	*20*
`bldg.1%env.`	**36**	0	36	36	50	5	*33*
`bldg.2%env.`	**30**	0	30	30	50	5	*21*
`bldg.3%env.`	**20**	0	20	20	50	5	*18*
`bldg.4%env.`	**18**	0	18	18	50	5	*18*
`bldg.5%env.`	**32**	0	32	32	50	5	*25*
`bldg.6%env.`	**38**	0	38	38	50	5	*44*
`bldg.7%env.`	**20**	0	20	20	50	5	*10*
`bldg.8%env.`	**39**	0	39	39	50	5	*16*
`bldg.9%env.`	**32**	0	32	32	50	5	*13*
`bldg.10%env`	**36**	0	36	36	50	5	*28*
`bldg.1%.2env`	**6**	0	6	6	31	5	*21*
`bldg.2%.2env`	**10**	0	10	10	25	5	*7*
`bldg.3%.2env`	**12**	0	12	12	15	5	*14*
`bldg.4%.2env`	**12**	0	12	12	13	5	*11*
`bldg.5%.2env`	**5**	0	5	5	27	5	*21*
`bldg.6%.2env`	**5**	0	5	5	33	5	*27*
`bldg.7%.2env`	**4**	0	4	4	15	5	*8*
`bldg.8%.2env`	**4**	0	4	4	34	5	*8*
`bldg.9%.2env`	**8**	0	8	8	27	5	*25*
`bldg.10%.2env`	**9**	0	9	9	31	5	*21*

Table 5.3 : Generating depth varied alternatives in Excel.

When the user saves this file and switches back to Catia, the same message box appears and upon hitting the update button, the following scene is created.

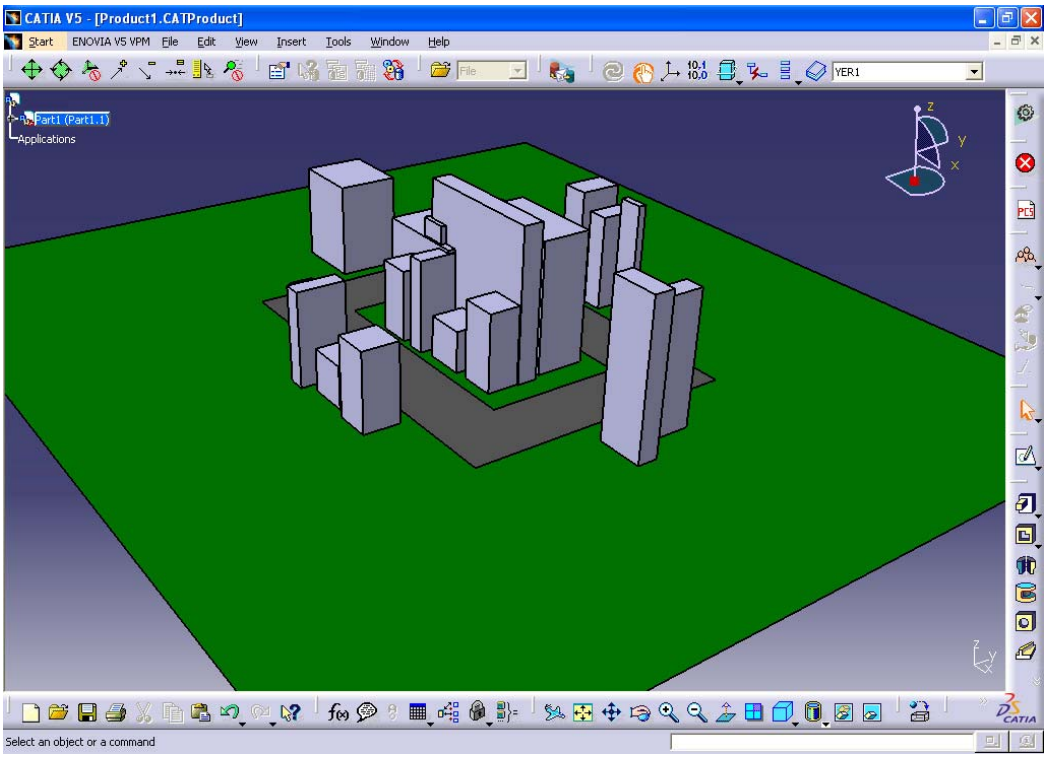


Figure 5.19 : Possible results of the plug-in of an urban block generated in Catia

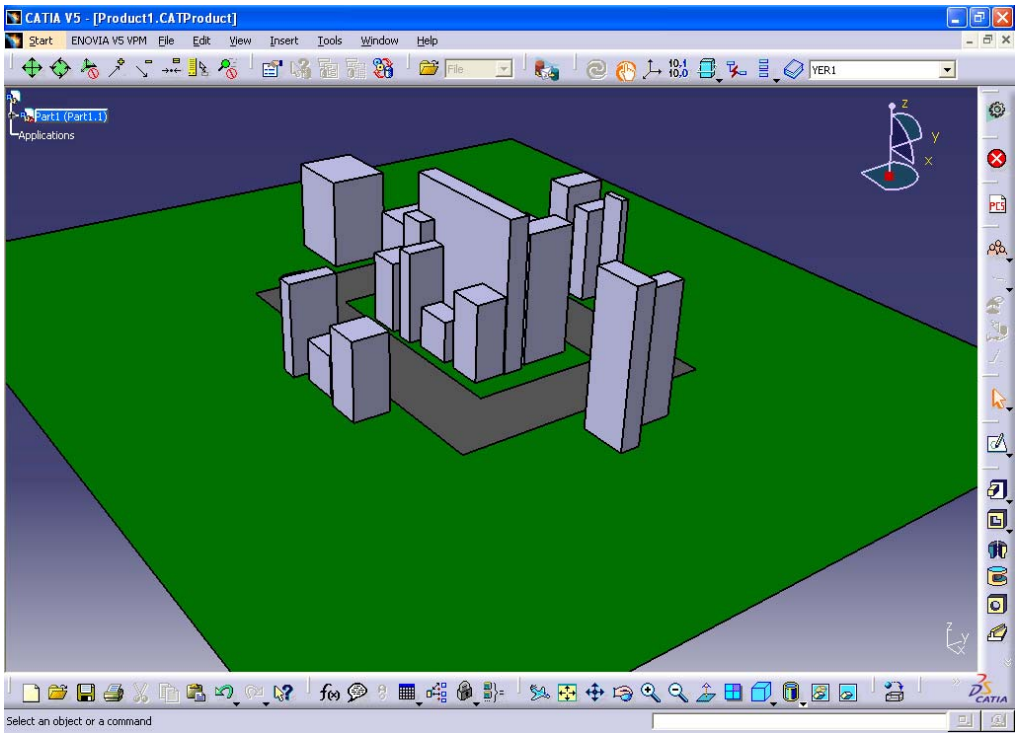


Figure 5.20 : Possible results of the plug-in of an urban block generated in Catia.

When compared with the previously generated scene, it can be observed that the environment buildings, the width and height values of the mirrored buildings in the

generated urban block remain the same while depth values of the mirrored buildings has been changed.

a b	0 0	Random? (0=no, 1=yes)	Manual	Result Value	Max mm	Min mm	Random (F9)
`site-d.` (mm)	**70000**	0	70000	70000	100000	10000	*22780*
`site-w.` (mm)	**40000**	0	40000	40000	100000	10000	*80535*
`bldg.1-w.` (mm)	**18219**	0	18219	18219	32000	6400	*9515*
`bldg.1-h.` (mm)	**28422**	0	28422	28422	64000	12800	*20474*
`bldg.2-w.` (mm)	**14606**	0	14606	14606	18000	3600	*6067*
`bldg.2-h.` (mm)	**38987**	0	38987	38987	54000	10800	*51315*
`bldg.3-w.` (mm)	**5985**	0	5985	5985	18000	3600	*13626*
`bldg.3-h.` (mm)	**45317**	0	45317	45317	54000	10800	*50228*
`bldg.4-w.` (mm)	**7619**	0	7619	7619	19833	3967	*13406*
`bldg.4-h.` (mm)	**35660**	0	35660	35660	39667	7933	*30368*
`bldg.5-w.` (mm)	**6278**	0	6278	6278	19833	3967	*8034*
`bldg.5-h.` (mm)	**29094**	0	29094	29094	39667	7933	*38424*
`bldg.6-w.` (mm)	**8370**	0	8370	8370	19833	3967	*13656*
`bldg.6-h.` (mm)	**33739**	0	33739	33739	39667	7933	*18710*
`bldg.7-w.` (mm)	**10620**	0	10620	10620	14000	2800	*8026*
`bldg.7-h.` (mm)	**23177**	0	23177	23177	28000	5600	*6545*
`bldg.8-w.` (mm)	**11728**	0	11728	11728	14000	2800	*8275*
`bldg.8-h.` (mm)	**12372**	0	12372	12372	42000	8400	*21879*
`bldg.9-w.` (mm)	**5061**	0	5061	5061	14000	2800	*3405*
`bldg.9-h.` (mm)	**27490**	0	27490	27490	28000	5600	*5605*
`bldg.10-w.` (mm)	**9830**	0	9830	9830	14000	2800	*6391*
`bldg.10-h.` (mm)	**21263**	0	21263	21263	28000	5600	*21894*
`bldg.1%`	**23**	1	40	23	30	20	*23*
`bldg.2%`	**28**	1	15	28	30	20	*28*
`bldg.3%`	**23**	1	30	23	30	20	*23*
`bldg.4%`	**22**	1	35	22	30	20	*22*
`bldg.5%`	**25**	1	30	25	30	20	*25*
`bldg.6%`	**30**	1	48	30	30	20	*30*
`bldg.7%`	**26**	1	15	26	30	20	*26*
`bldg.8%`	**25**	1	35	25	30	20	*25*
`bldg.9%`	**26**	1	59	26	30	20	*26*
`bldg.10%`	**21**	1	32	21	30	20	*21*
`bldg.in btw%`	**5**	0	5	5	20	5	*13*
`bldg.1%.2`	**6**	0	6	6	18	0	*9*
`bldg.2%.2`	**10**	0	10	10	23	0	*19*
`bldg.3%.2`	**12**	0	12	12	18	0	*16*
`bldg.4%.2`	**12**	0	12	12	17	0	*14*
`bldg.5%.2`	**5**	0	5	5	20	0	*11*
`bldg.6%.2`	**5**	0	5	5	25	0	*25*
`bldg.7%.2`	**4**	0	4	4	21	0	*13*
`bldg.8%.2`	**4**	0	4	4	20	0	*14*
`bldg.9%.2`	**8**	0	8	8	21	0	*14*
`bldg.10%.2`	**9**	0	9	9	16	0	*4*
`bldg.1%env.`	**36**	0	36	36	50	5	*47*
`bldg.2%env.`	**30**	0	30	30	50	5	*17*
`bldg.3%env.`	**20**	0	20	20	50	5	*16*
`bldg.4%env.`	**18**	0	18	18	50	5	*47*

`bldg.5%env.`	**32**	0	32	32	50	5	*32*
`bldg.6%env.`	**38**	0	38	38	50	5	*32*
`bldg.7%env.`	**20**	0	20	20	50	5	*26*
`bldg.8%env.`	**39**	0	39	39	50	5	*5*
`bldg.9%env.`	**32**	0	32	32	50	5	*45*
`bldg.10%env.`	**36**	0	36	36	50	5	*14*
`bldg.1%.2env.`	**6**	0	6	6	31	5	*20*
`bldg.2%.2env.`	**10**	0	10	10	25	5	*19*
`bldg.3%.2env.`	**12**	0	12	12	15	5	*10*
`bldg.4%.2env.`	**12**	0	12	12	13	5	*8*
`bldg.5%.2env.`	**5**	0	5	5	27	5	*23*
`bldg.6%.2env.`	**5**	0	5	5	33	5	*28*
`bldg.7%.2env.`	**4**	0	4	4	15	5	*9*
`bldg.8%.2env.`	**4**	0	4	4	34	5	*33*
`bldg.9%.2env.`	**8**	0	8	8	27	5	*20*
`bldg.10%.2env.`	**9**	0	9	9	31	5	*17*

Table 5.4 : Generating depth varied alternatives in Excel with Min&Max depth range altered.

In the image above, the circled area which corresponds to the sixth and seventh columns have been altered. This section is responsible from defining the minimum and the maximum value that a recently generated mass can get in terms of percentage of the corresponding side site. In this case, the maximum percentage that a depth value can get is 30 and the minimum percentage that a depth value can get is 20. This change is expected to generate new building blocks that have less depth values therefore create a partial new urban block with more open space available.

The following two images are different views of the alternative generated after altering the minimum and maximum depth values of new blocks to be generated. It can be noticed that this alternative consists of building blocks with an inner open space in the middle. As expected, an alternative with more open space are and less building block is generated. It is also possible to generate such an alternative without changing the minimum and maximum range. However, it would take more trial and error because there are a lot more possibilities. When narrowing the minimum maximum range the user also decreases the number of alternatives and increases the chances to generate an alternative that suits his purpose.

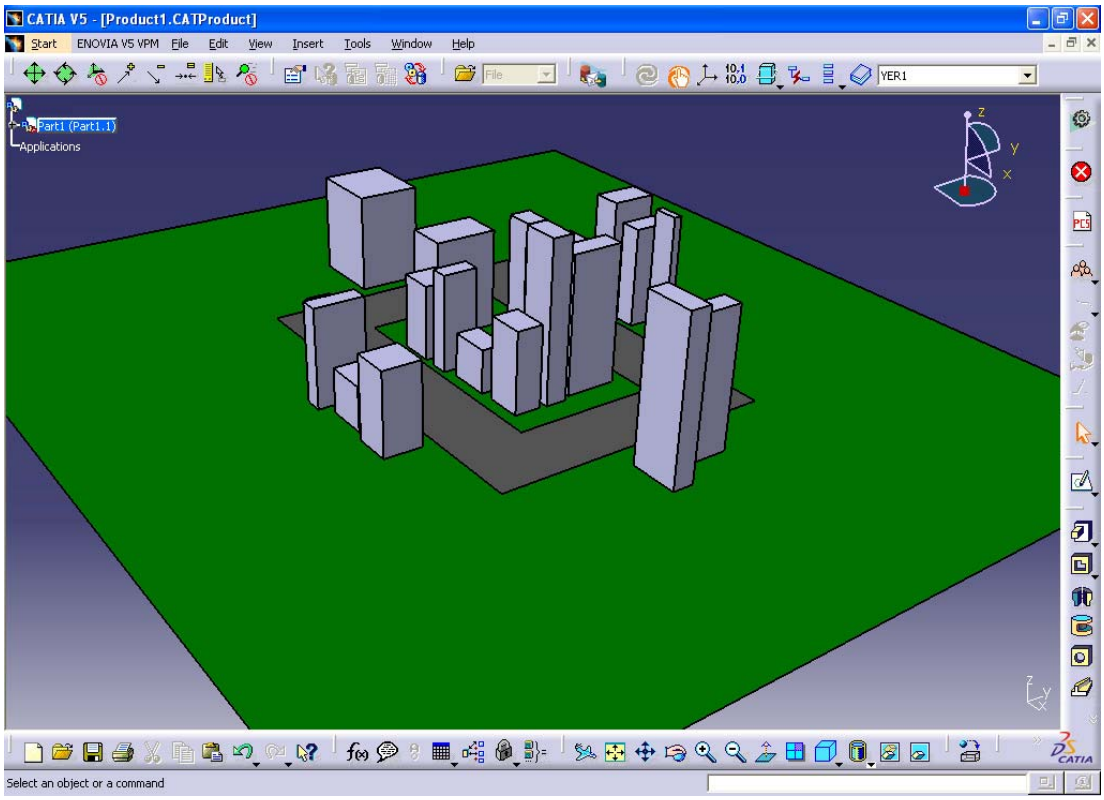


Figure 5.21 : Possible results of the plug-in of an urban block generated in Catia.

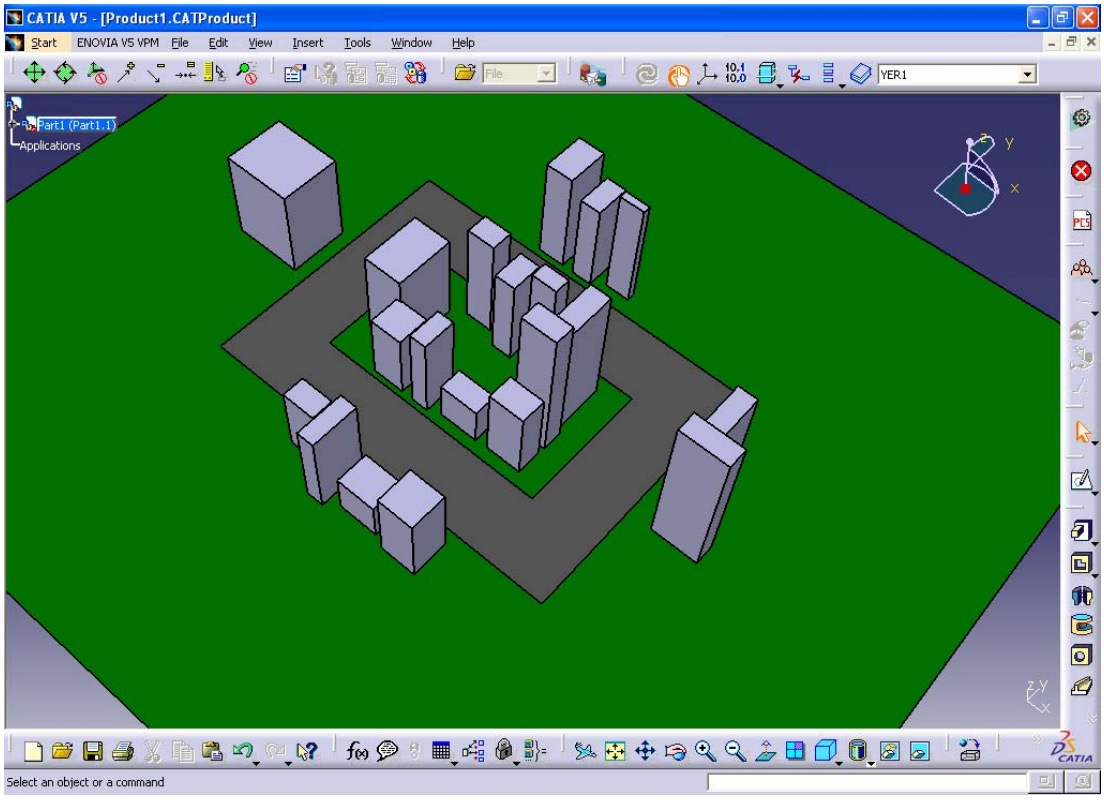


Figure 5.22 : Possible results of the plug-in of an urban block generated in Catia.

a	0	Random? (0=no,	Manual	Result	Max	Min	Random
b	0	1=yes)		Value	mm	mm	(F9)
`site-d.` (mm)	**70000**	0	70000	70000	100000	10000	*87567*
`site-w.` (mm)	**40000**	0	40000	40000	100000	10000	*54974*
`bldg.1-w.` (mm)	**18219**	0	18219	18219	32000	6400	*29228*
`bldg.1-h.` (mm)	**28422**	0	28422	28422	64000	12800	*53216*
`bldg.2-w.` (mm)	**14606**	0	14606	14606	18000	3600	*13395*
`bldg.2-h.` (mm)	**38987**	0	38987	38987	54000	10800	*43454*
`bldg.3-w.` (mm)	**5985**	0	5985	5985	18000	3600	*12487*
`bldg.3-h.` (mm)	**45317**	0	45317	45317	54000	10800	*20645*
`bldg.4-w.` (mm)	**7619**	0	7619	7619	19833	3967	*9103*
`bldg.4-h.` (mm)	**35660**	0	35660	35660	39667	7933	*31324*
`bldg.5-w.` (mm)	**6278**	0	6278	6278	19833	3967	*7244*
`bldg.5-h.` (mm)	**29094**	0	29094	29094	39667	7933	*25208*
`bldg.6-w.` (mm)	**8370**	0	8370	8370	19833	3967	*13014*
`bldg.6-h.` (mm)	**33739**	0	33739	33739	39667	7933	*9038*
`bldg.7-w.` (mm)	**10620**	0	10620	10620	14000	2800	*4483*
`bldg.7-h.` (mm)	**23177**	0	23177	23177	28000	5600	*10503*
`bldg.8-w.` (mm)	**11728**	0	11728	11728	14000	2800	*11268*
`bldg.8-h.` (mm)	**12372**	0	12372	12372	42000	8400	*27159*
`bldg.9-w.` (mm)	**5061**	0	5061	5061	14000	2800	*7878*
`bldg.9-h.` (mm)	**27490**	0	27490	27490	28000	5600	*23198*
`bldg.10-w.` (mm)	**9830**	0	9830	9830	14000	2800	*11007*
`bldg.10-h.` (mm)	**21263**	0	21263	21263	28000	5600	*17203*
`bldg.1%`	**15**	1	40	15	30	5	*15*
`bldg.2%`	**8**	1	15	8	30	5	*8*
`bldg.3%`	**14**	1	30	14	30	5	*14*
`bldg.4%`	**16**	1	35	16	30	5	*16*
`bldg.5%`	**29**	1	30	29	30	5	*29*
`bldg.6%`	**14**	1	48	14	30	5	*14*
`bldg.7%`	**8**	1	15	8	30	5	*8*
`bldg.8%`	**7**	1	35	7	30	5	*7*
`bldg.9%`	**13**	1	59	13	30	5	*13*
`bldg.10%`	**16**	1	32	16	30	5	*16*
`bldg.in btw%`	**5**	0	5	5	20	5	*9*
`bldg.1%.2`	**6**	0	6	6	10	0	*2*
`bldg.2%.2`	**10**	0	10	10	3	0	*2*
`bldg.3%.2`	**12**	0	12	12	9	0	*3*
`bldg.4%.2`	**12**	0	12	12	11	0	*4*
`bldg.5%.2`	**5**	0	5	5	24	0	*7*
`bldg.6%.2`	**5**	0	5	5	9	0	*8*
`bldg.7%.2`	**4**	0	4	4	3	0	*0*
`bldg.8%.2`	**4**	0	4	4	2	0	*2*
`bldg.9%.2`	**8**	0	8	8	8	0	*5*
`bldg.10%.2`	**9**	0	9	9	11	0	*7*
`bldg.1%env.`	**36**	0	36	36	50	5	*13*
`bldg.2%env.`	**30**	0	30	30	50	5	*12*
`bldg.3%env.`	**20**	0	20	20	50	5	*19*
`bldg.4%env.`	**18**	0	18	18	50	5	*31*
`bldg.5%env.`	**32**	0	32	32	50	5	*33*
`bldg.6%env.`	**38**	0	38	38	50	5	*10*
`bldg.7%env.`	**20**	0	20	20	50	5	*26*
`bldg.8%env.`	**39**	0	39	39	50	5	*8*
`bldg.9%env.`	**32**	0	32	32	50	5	*17*
`bldg.10%env.`	**36**	0	36	36	50	5	*6*
`bldg.1%.2env.`	**6**	0	6	6	31	5	*8*
`bldg.2%.2env.`	**10**	0	10	10	25	5	*9*
`bldg.3%.2env.`	**12**	0	12	12	15	5	*11*
`bldg.4%.2env.`	**12**	0	12	12	13	5	*6*
`bldg.5%.2env.`	**5**	0	5	5	27	5	*23*
`bldg.6%.2env.`	**5**	0	5	5	33	5	*20*
`bldg.7%.2env.`	**4**	0	4	4	15	5	*9*
`bldg.8%.2env.`	**4**	0	4	4	34	5	*9*
`bldg.9%.2env.`	**8**	0	8	8	27	5	*9*
`bldg.10%.2env.`	**9**	0	9	9	31	5	*13*

Table 5.5 : Generating depth varied alternatives in Excel with a very small Min. value.

In the example above, the minimum percentage value that a depth value of a new generated mass can get is defined as 5%. This change is expected to cause unacceptable building proportions to be formed since it will allow for very small depth values to be assigned.

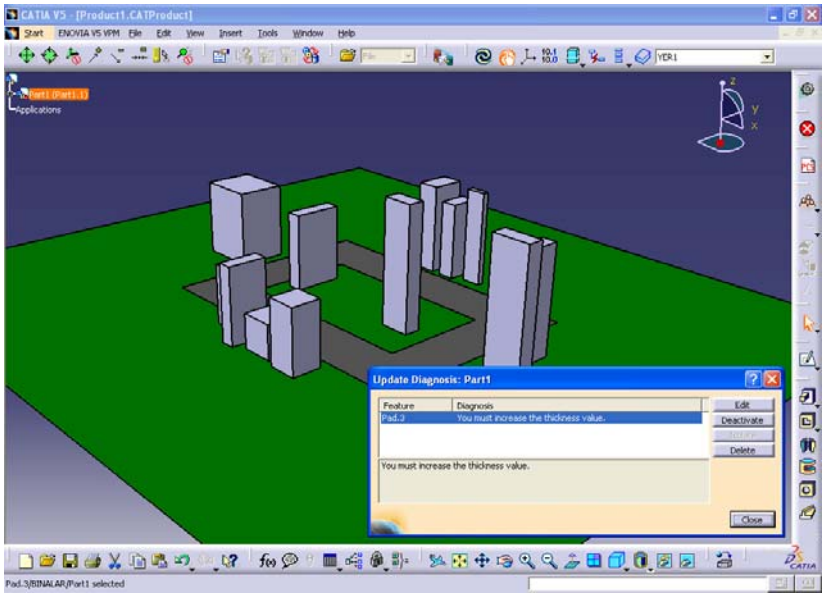


Figure 5.23 : Error during generation

The image above shows where staying close to the minimum value causes an error as many of the mirrored buildings are lost. The system pops up an error message box stating that the user should increase the depth value.

The table below is a second trial to generate an alternative with minimum depth value of %5 for the mirrored buildings. The following figure shows the generated alternative where all the buildings are present on the site; but at least two of them are not in acceptable depth values.

a b	0 0	Random? (0=no, 1=yes)	Manual	Result Value	Max mm	Min mm	Random (F9)
`site-d.` (mm)	**70000**	0	70000	70000	100000	10000	*42078*
`site-w.` (mm)	**40000**	0	40000	40000	100000	10000	*69796*
`bldg.1-w.` (mm)	**18219**	0	18219	18219	32000	6400	*13158*
`bldg.1-h.` (mm)	**28422**	0	28422	28422	64000	12800	*55972*
`bldg.2-w.` (mm)	**14606**	0	14606	14606	18000	3600	*8481*
`bldg.2-h.` (mm)	**38987**	0	38987	38987	54000	10800	*29991*
`bldg.3-w.` (mm)	**5985**	0	5985	5985	18000	3600	*3656*
`bldg.3-h.` (mm)	**45317**	0	45317	45317	54000	10800	*35115*
`bldg.4-w.` (mm)	**7619**	0	7619	7619	19833	3967	*17749*
`bldg.4-h.` (mm)	**35660**	0	35660	35660	39667	7933	*36232*
`bldg.5-w.` (mm)	**6278**	0	6278	6278	19833	3967	*8602*
`bldg.5-h.`	**29094**	0	29094	29094	39667	7933	*15070*

(mm)							
`bldg.6-w.`	**8370**	0	8370	8370	19833	3967	*17305*
(mm)							
`bldg.6-h.`	**33739**	0	33739	33739	39667	7933	*18187*
(mm)							
`bldg.7-w.`	**10620**	0	10620	10620	14000	2800	*8330*
(mm)							
`bldg.7-h.`	**23177**	0	23177	23177	28000	5600	*18864*
(mm)							
`bldg.8-w.`	**11728**	0	11728	11728	14000	2800	*9562*
(mm)							
`bldg.8-h.`	**12372**	0	12372	12372	42000	8400	*22382*
(mm)							
`bldg.9-w.`	**5061**	0	5061	5061	14000	2800	*4168*
(mm)							
`bldg.9-h.`	**27490**	0	27490	27490	28000	5600	*20612*
(mm)							
`bldg.10-w.`	**9830**	0	9830	9830	14000	2800	*3812*
(mm)							
`bldg.10-h.`	**21263**	0	21263	21263	28000	5600	*13397*
(mm)							
`bldg.1%`	**20**	1	40	20	30	5	*20*
`bldg.2%`	**30**	1	15	30	30	5	*30*
`bldg.3%`	**20**	1	30	20	30	5	*20*
`bldg.4%`	**7**	1	35	7	30	5	*7*
`bldg.5%`	**9**	1	30	9	30	5	*9*
`bldg.6%`	**19**	1	48	19	30	5	*19*
`bldg.7%`	**21**	1	15	21	30	5	*21*
`bldg.8%`	**21**	1	35	21	30	5	*21*
`bldg.9%`	**13**	1	59	13	30	5	*13*
`bldg.10%`	**18**	1	32	18	30	5	*18*
`bldg.in btw%`	**5**	0	5	5	20	5	*14*
`bldg.1%.2`	**6**	0	6	6	15	0	*7*
`bldg.2%.2`	**10**	0	10	10	25	0	*5*
`bldg.3%.2`	**12**	0	12	12	15	0	*13*
`bldg.4%.2`	**12**	0	12	12	2	0	*0*
`bldg.5%.2`	**5**	0	5	5	4	0	*1*
`bldg.6%.2`	**5**	0	5	5	14	0	*10*
`bldg.7%.2`	**4**	0	4	4	16	0	*15*
`bldg.8%.2`	**4**	0	4	4	16	0	*4*
`bldg.9%.2`	**8**	0	8	8	8	0	*4*
`bldg.10%.2`	**9**	0	9	9	13	0	*3*
`bldg.1%env.`	**36**	0	36	36	50	5	*39*
`bldg.2%env.`	**30**	0	30	30	50	5	*25*
`bldg.3%env.`	**20**	0	20	20	50	5	*42*
`bldg.4%env.`	**18**	0	18	18	50	5	*23*
`bldg.5%env.`	**32**	0	32	32	50	5	*14*
`bldg.6%env.`	**38**	0	38	38	50	5	*8*
`bldg.7%env.`	**20**	0	20	20	50	5	*27*
`bldg.8%env.`	**39**	0	39	39	50	5	*38*
`bldg.9%env.`	**32**	0	32	32	50	5	*11*
`bldg.10%env.`	**36**	0	36	36	50	5	*29*
`bldg.1%.2env.`	**6**	0	6	6	31	5	*8*
`bldg.2%.2env.`	**10**	0	10	10	25	5	*8*
`bldg.3%.2env.`	**12**	0	12	12	15	5	*7*
`bldg.4%.2env.`	**12**	0	12	12	13	5	*7*
`bldg.5%.2env.`	**5**	0	5	5	27	5	*7*
`bldg.6%.2env`	**5**	0	5	5	33	5	*25*
`bldg.7%.2env`	**4**	0	4	4	15	5	*14*
`bldg.8%.2env`	**4**	0	4	4	34	5	*12*
`bldg.9%.2env`	**8**	0	8	8	27	5	*23*
`bldg.10%.2env.`	**9**	0	9	9	31	5	*29*

Table 5.6 : Generating depth varied alternatives in Excel with a very small Min value.

The following image above shows the generated design alternative according to visualization of the design table just explained. As expected, recently generated alternative offers at least two unacceptable solutions. They are too narrow for their height values because the minimum value defined makes it possible for too small values to be assigned. Such alternatives are unlikely to be accepted by engineers or designers.

The examples above are demonstrated to explain how design alternatives addressing to this thesis' problem and also that Catia & Excel approach has its filters embedded within the Excel Design Table. The user has the opportunity to alter the values and proportions in a way to meet specific design needs and restrictions. Both the design problem, the flexibilities and restrictions of the process need to be well understood in order to eliminate such unacceptable possibilities being generated.

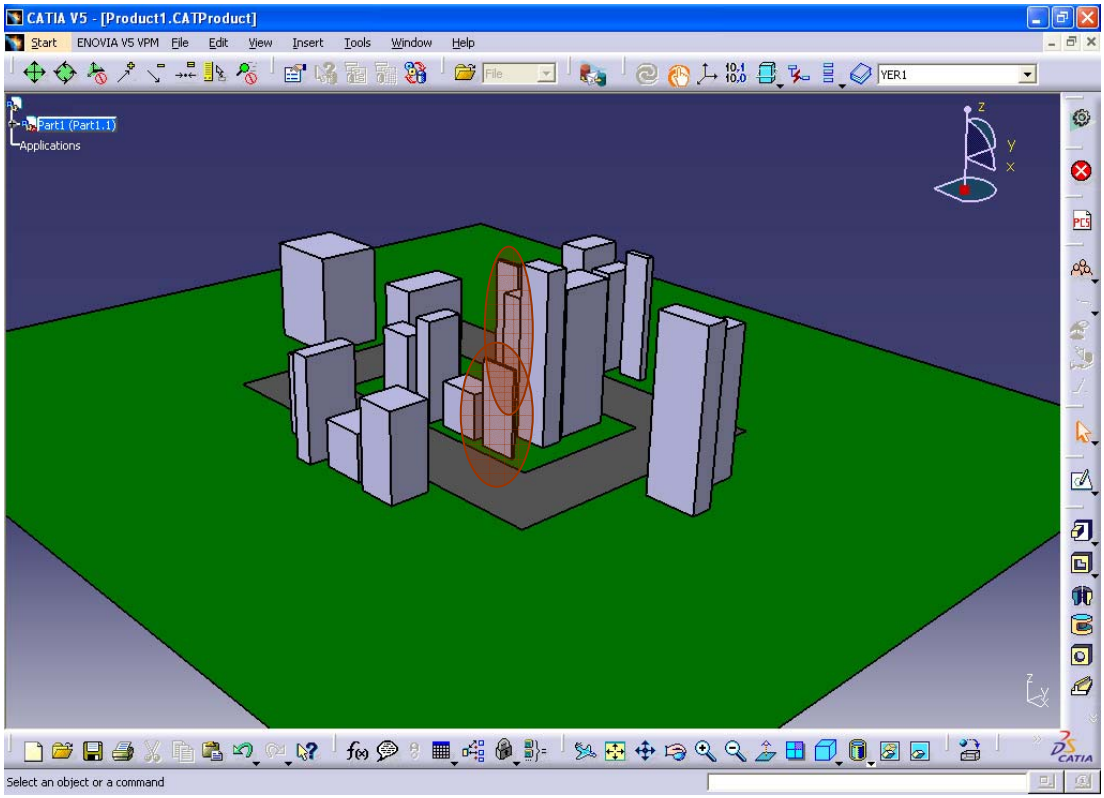


Figure 5.24 : Possible unacceptable result of the plug-in of an urban block generated in Catia.

5.6.3 Advantages and Disadvantages of Catia & Excel Approach

Although architects are not very familiar to the software Catia, it will only be used to read the values in the Excel Design Table and visualize the design model. The user intensely needs to understand and evaluate the values to be put into the Excel file in order to create alternatives that suits his design needs. After elaborating the Excel file, Catia file merely needs to be updated which prevents the user loosing time figuring out the interface.

The plug-in does not provide the user with modulated filters allowing him to eliminate alternatives which do not serve his needs.. However, the Excel Design Table allows for flexibility to some extend and some filters are embedded within. To be able to make most of the flexibility embedded in the plug-in parametric features and relations of the desired design needs to be well translated into the table.

The process develops as a trial and error system. It is difficult to predict possible alternatives since the process is not visual but parametric. This could resemble a disadvantage for architects who work intensely on visual objects.

The last disadvantage on the solution established through Catia & Excel, is that at the moment there is no way to put all the possible alternatives together and save. Since the increment concept is not applied in this solution, the entities can take any dimension value between a specified range which makes the number of design alternatives incredibly high.

6. CONCLUSION

Although this plug-in is addressing issues about site approach, it is not yet fully elaborated to the point which it justifies its potentials. At the moment it acts as an initiator for creating harmonic designs. This research is also powerful in the sense that it has the potential to revitilize urban areas currently not very favorable. This method is claimed to have the potential to create harmonic solutions after a set of refinement steps, even when applied onto unfavorable sites. The initial idea has been elaborated verbally first, and two attempts to realize the model have been made. The attempts have been made in two different modeling software : Catia and 3D Studio Max.

The main reason by appliying this certain approach on site computationally, is to benefit the pace of computation. There is always some way to optimize a process. Although this plug-in is designed to run fast, as soon as method of coding has improved further, it would reach better time records. The status quo shows that the first process the plug-in runs is to compute all the possibilities given the restrictions prior to running the plug-in. Then what it does is to filter the alternatives to generate a result set of alternatives targeting to fulfill the user's definitions. However, there is a much faster way to achieve it. Instead of having the plug-in draw all the possible combinations, and filtering the result with filters at the end, the restrictions could be embedded in the script so that unwanted results would not be drawn at all. Although it would result in faster design alternative generation process, the time elapsed to write the script would be longer and more complicated increasing the risk of making errors. The status quo should be percieved as a starting point where the plug-in works even cumbersomely to a degree and it can always be optimized to run more effectively in the future.

There aslo are some technical lacks in the proposed plug-in and two applied models. Currently, the plug-in works orthanogally only. This means that the plug-in will not be running properly when faced with curves.

For future development, artificial intelligence will be introduced to the plug-in. It will have the option to prepare a set of alternatives under the name of ‘proposed alternatives’ by keeping track of statistical information for specified number of times for a certain user. The plug-in will be able to track the alternatives that have been chosen to be evaluated for further design potentials and the citeria that has developed the specific alternative. Therefore it will be able to decide which criteria to apply and choose under experienced situations and add to its database the relations between the situations and chosen criteria for future use.

On the other hand, Knowledge Based Systems are planned to be embedded in the progressing future works. Descriptive Knowledge Modules will be developed conssiting of semantic information such as Neufert standards. This module will help the designer eliminate the design alternatives by relevance to chosen type of module such as residential, public or commercial all of which have different set of dimensional standards.

The proposed plug-in has been realized in two ways. They have been developed in order to help the architect or the city planner with creative design alternatives that are also in visual harmony with its built environment. The generated solutions are desired to force the user and the readers of this thesis to put emphasis on responding to environment when designing; and also discover the potentials of the point where the architect and the city planner meet and share a platform which helps them to use the same language. The field work research made shows that there have been similar efforts to establish a system to generate a new city for purposes that are both similar and different than of this thesis’. CityEngine, CityZoom, The Melinkov Grammar, Smart Solutions for Spatial Planning and a studio developed by Jose Duarte have been explained in order to provide the reader with related work.

During this part of the research, a comparison between the field work and the outcome of this thesis has been made to realize the advantages and disadvantages of the model developed. This will lead the direction in which the new model will progress in the future by feeding it with technical and conceptual ideas.

It has been observed that most of the efforts combine Shape Grammars and all of them are composed of Rule Based Systems, similar to this thesis. However, this thesis is unique and differentiates by focusing only on block generation which is rarely seen in the approaches explained.

CityEngine has presented a different method in defining an urban block by applying a recursive algorithm on the site boundary defined by roads established by applying various steps provided by the software. CityEngine also differs the way in which it approaches to design of a city : top-down. This thesis follows a bottom-up approach which makes the result unpredictable, unlike CityEngine.

This thesis takes urban block generation into consideration in a bottom-up approach similar to the Melinkov Grammar where the end result is complex although the process and the initial shapes are very simple. However, this thesis differentiates from the Melinkov Grammar approach in that the Melinkov Grammar is a 2D generation tool where the “Mirrorer” generates solid urban blocks in 3D rather than a 2D city network.

CityZoom is highly oriented to enable the user visualize a city according to regualtions and modify it as a result of a set of performance models. It aims at finding a balance between design needs and regulation requirements. The software does not offer a system that creates a city from scratch as it is intended with this thesis.

Urban Design with Patterns and Shape Grammars Studio does not offer a computational product. This study has been explained within this thesis to present as an experiment that proves efficiency of Rule Based Systems as well as because it consists of approaches developed to generate an urban block at a level which is not present in other models explained.

Smart Solutions for Spatial Planning is another software developed which takes generation of a city into consideration from urban scale to building scale including a step to generate urban block alternatives. The software presents an approach on building block generation depending on generation of allotments with reasonable dimensions for development of buildings and open spaces following a recursive search method. Building masses are created taking climatic considerations into account and making a distictions between public and private realm. This step of the software relates to this thesis in terms of intention to create urban blocks with building masses but differentiates with its methods. SSSP uses a recursive search method in dividing the plot into allotments depending on optimization rules. Hewever, this thesis simply mimics the environment to create a new design. It creates building masses taking environmental and social conditions into account

which is similar to the approach to city in this thesis in terms of they both take environment into account, however at a different level.

When both outcomes of this theses are compared, Catia solution proves to be more flexible and inherently embeds filters similar to ones mentioned in the proposed process. However, this feature is not valid for the implementation developed with MaxScript. Although Catia interface is not widely used among architects unlike it is with 3D Studio Max, the benefits of being able to limit and get results related to the designer's target overcomes interface problems. Another disadvantage of Catia solution compared to MaxScript is that there is no systematic way of saving and storing the generated design alternatives where on the other hand, the MaxScript solution creates an animation by combining all the alternatives allowing the user to search through all the results within the same file.

In the future, further studies regarding a comparison of the two implementations is possible by experimenting user-plug-in interaction on a group of students. The outcome of such a study will be more descriptive of the advantages and disadvantages of the models and be more decisive on which approach will be used in the next phases of the future studies related to this thesis.

REFERENCES

Agarwal, M. & Cagan J., (1998). A blend of different tastes: the language of coffeemakers *Environment and Planning B: Planning and Design*. 25, 205-226.

Brown, K. N. & Cagan J., (1996). Grammatical Design and Bounded Creatibility *EDRC Report 24-124-96 Engineering Design Research Center*, Carnegie-Mellon University, Pittsburgh, PA

Batty, M., Longley, P., (1994). *The Fractal City* Academic Press, San Diego, CA and London.

Cardoso, D. & **Nagakura, T.,** (2007). The Melinkov Grammar. Retrieved on August, 18, 2009 from Formal Design Knowledge and Programmed Constructs: http://mit.edu/dcardoso/www/4.564/melnikov_paper.pdf

Chase, C, S (1999). Grammar Based Design: Issues for User Interaction Models. *Media and Design Process*, proceedings of ACADIA '99. 198-210.

Chase C. S., (1997). Emergence, Creativity and Computational Tractability, *Workshop Proceedings, Interactive Systems for Supporting the Emergence of Concepts and Ideas.* CHI ’97, Atlanta, Georgia. Retrieved January 10, 2009 from http://bashfull.lut.ac.uk/chi-wshop.

Curtis. J. R. W., (1996). *Modern Architecture Since 1900* (3rd Edition) Phaidon Press Limited, London.

Çağdaş. G., (1996). A Shape Grammar: The Language of Traditional Turkish Houses. *Environment and Planning B: Planning and Design.* 23, 443-464.

Çolakoğlu, B., (2005). Design by Grammar: An Interpretation and Generation of Vernacular Hayat Houses in a Contemporary Context. *Environment and Planning B: Planning and Design*. 32, 141-149.

Duarte, J. P. & Beirão N.J (2009) Grammars of Design and Grammars for Designing: Grammar Based Patterns for Urban Design. *Proceedings of CAAD Futures '09 Conference*, Montreal, Canada..

Filler, M., (1991, March 24). Architecture View; Baron Haussmann, Urban Designer Par Excellence. *The New York Times.* Retrieved on January, 5, 2009 from http://www.nytimes.com/1991/03/24/arts/architecture-view-baron-haussmann-urban-designer-par-excellence.html

Gips, J., (1999). Computer Implementations of Shape Grammars. *Workshop on Shape Computation.,* MIT.

Koile, K., (2001). The Architect's Collaborator: Toward Intelligent Tools for Conceptual Design. Ph.D. dissertation MIT EECS Department, MIT.

Krishnamurti, R. & Stouffs, R., (1997) Spatial change: continuity, reversibility, and emergent shapes. *Environment and Planning B: Planning and Design*, 24, 359-384.

Krishnamurti, R. & Stouffs R., (1993) Spatial Grammars : Motivation, Comparison and New Results. Proceedings of *The Fifth International Conference on Computer-Aided Architectural Design Futures*, Pittsburgh, Pennsylvania, US. 57-74.

Knight, T. W., (1998)**.** Shape grammars. *Environment and Planning B: Planning and Design,* Anniversary Issue, 86–91.

Knight, T. W., (1998). Designing a Shape Grammar: Problems of Predictability. Proceedings of the 5th International *Artificial Intelligence in Design Conference*. 499-516.

Kwon, Young Doo, (2003). ArchiDNA: A Generative System for Shape Configuration. Thesis of Master of Science in Architecture, University of Washington.

Larson, K. & Intille, S. & Mcleish, T.J., (2004). Open Source Building-Reinventing Places of Living. *BT Technology Journal.* 22, 4.

Müller, P. & Parish, Y. (2001). Procedural Modeling of Cities. Proceedings of the *International Conference on Computer Graphics and Interactive Techniques,* ACM Siggraph, Los Angeles, CA, USA. 301-308.

Müller, P. & Wonka, P. & Haegler, S. & Ulmer, A. & Gool Van, L., (2006). Procedural Modeling of Buildings. *International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2006 Papers.* 614-623.

Müller, P. & Zeng, G. & Wonka, P. & Gool Van, L., (2006). Image-based Procedural Modeling of Facades. ACM Transactions on Graphics (TOG). 26, 3.

Larive, M. & **Dupuy, Y. & Gaildrat, V**., (2005). Automatic Generation of Urban Zones. Proceedings of *WSCG'2005*. 9-12.

Okçu, S., (2003). Software Developed for Livingroom Furnishing, Graduate Thesis, Istanbul Technical University.

Russell, S. & Norvig, P., (1995). *Artificial Intelligence: A Modern Approach*. Prince Hall, Englewood Cliffs, New Jersey.

Stiny, G., (1978). The Palladian grammar. *Environment and Planning B: Planning and Design*. 5, 5-18.

Stiny, G., (1985). Computing with form and Meaning in Architecture. *Journal of Architectural Education.* 39, 7-19.

Şener. S. M., (2008). A Shape Grammar Algorithm and Educational Software to Analyze Classic Ottoman Mosques. *ITU A | Z.* 5 ,1,12-30.

Turing, A, M., (1995). Computing Machinery and Intelligent. *Computers and Thoughts* (pp-11-35). MIT Press Cambridge, MA, USA

Turkienicz, B. & Golçalves, B. B. & Grazziotin, P., (2008). CityZoom : A Visualization Tool for the Assessment of Planning Regulations. *International Journal of Architectural Computing*. 6, 79-95.

url 1: Terry Knight Lecture Slides. Retrieved on July, 10, 2009 from www.mit.edu/~4.184/lecture1_terry/lecture1slides_final.PPT - 2001-02-02

url 2: Smart Solutions for Spatial Planning: Pedestrian Network Connectivity. Retrieved on July, 10, 2009 from http://www.vimeo.com/2023898

APPENDIX

Function 1:

```
fn getLocation obj rect =
```

Function 2:

```
(
local returnres
    rectCenter = rect.center
    objCenter = obj.center
    width = rect.width/2
    length = rect.length/2
    upperY = rectCenter.y + length
    lowerY = rectCenter.y - length
    leftX = rectCenter.x - width
    rightX = rectCenter.x + width
    if objCenter.y > upperY then returnres = 1
    else if objCenter.y < lowerY then returnres = 3
    else if objCenter.x > rightX then returnres = 2
    else if objCenter.x < leftX do (returnres = 4)
    return returnres
)
```

Function 3:

```
fn getBackKnots obj dir =
( local vertexnum = numknots obj;
    local vertexs = #(1,0);
    local vertexcount = 0;
    while vertexs[2] == 0 do
    ( local vertex = 1;
        local point = getknotpoint obj 1 1;
        if(vertexcount == 1) and (vertexs[1] == 1) do
        ( vertex = 2;
            point = getknotpoint obj 1 2;
        )
        for i in 1 to vertexnum do
        ( if(i != vertexs[1]) do
            ( temppoint = getknotpoint obj 1 i;
                if (dir == 1) then
                ( if temppoint.y < point.y do
                    ( point = temppoint;
                        vertex = i;
                    )
                )
                else if (dir == 2) then
                ( if temppoint.x < point.x do
                    ( point = temppoint;
```

```
                    vertex = i;
                )
            )
            else if (dir == 3) then
            ( if temppoint.y > point.y do
                ( point = temppoint;
                    vertex = i;
                )
            )
            else if (dir == 4) do
            ( if temppoint.x > point.x do
                ( point = temppoint;
                    vertex = i;
                )
            )
        )
    )
    vertexs[vertexcount + 1] = vertex;
    vertexcount = vertexcount + 1;
)
return vertexs
)
```

Function 4:

```
fn generateCombMatrix n value =
( local matrix = #();
    local prevrow = #();
    local rowcount = 1;
    local rowcountlimit = (value+1)^n;
    for k in 1 to n do
    ( prevrow[k] = 0;
    )
    while rowcount < rowcountlimit do
    (  local colcount = 1;
        local row = (deepcopy prevrow);
        for i in 1 to n do
        ( if (mod rowcount colcount) == 0 then
            ( row[i] = (mod (prevrow[i]+1) (value + 1));
            )
            colcount = colcount * (value+1);
        )
        append matrix row;
        prevrow = (deepcopy row);
        rowcount = rowcount + 1;
    )
    return matrix
)
```

Function 5:

```
fn simulate quadarray upper lower left right =
( local numbuild = 0;
    local knotlist = #();
    local incr = #(((upper-lower)/7), ((right-left)/7));
    local extrudeset = #();
    for i in 1 to quadarray.count do
    ( numbuild = numbuild + quadarray[i].count;
    )
    combmatrix = (generatecombmatrix numbuild 3)
    for i in 1 to quadarray.count do
    ( for j in 1 to quadarray[i].count do
        ( local obj = quadarray[i][j];
            append extrudeset obj.modifiers[2];
            deletemodifier obj 2;
            converttosplineshape obj;
            animatevertex obj #all;
            tempknots = getbackknots obj i;
            append knotlist tempknots;
        )
    )
    prevcomb = #()
    for i in 1 to numbuild do (append prevcomb 0)
  animate on
```

```
( animationrange = interval 0 combmatrix.count
    for i in 1 to combmatrix.count do
    (at time i
        ( curcomb = combmatrix[i];
            local indexmap = 1;
            for j in 1 to quadarray.count do
            ( for k in 1 to quadarray[j].count do
                (local incramount = (curcomb[indexmap] - prevcomb[indexmap]);
                    local incrvector;
                    if j == 1 then
                    ( incrvector = [0, -incramount*incr[1],0];
                    )
                    else if j == 2 then
                    ( incrvector = [-incramount*incr[2],0,0];
                    )
                    else if j == 3 then
                    ( incrvector = [0, incramount*incr[1],0];
                    )
                    else
                    ( incrvector = [incramount*incr[2],0,0];
                    ) quadarray[j][k][4][8][2 + ((knotlist[indexmap][1] - 1)*3)].value =
(quadarray[j][k][4][8][2 + ((knotlist[indexmap][1] - 1)*3)].value + incrvector);
                    quadarray[j][k][4][8][2 + ((knotlist[indexmap][2] - 1)*3)].value =
(quadarray[j][k][4][8][2 + ((knotlist[indexmap][2] - 1)*3)].value + incrvector);
                    indexmap = indexmap + 1;
```

```
                    )

                ) prevcomb = curcomb;

            ) ) )

    local index = 1

    for i in 1 to quadarray.count do

    ( for j in 1 to quadarray[i].count do

        ( addmodifier quadarray[i][j] extrudeset[index];

            index = index + 1;

        )

    )

)
```

Code Step A :

```
for i in 2 to objects.count do

    (

        if (getLocation objects[i] rect) == 1 then

        (

            append posyarray objects[i];

        )

        else if (getLocation objects[i] rect) == 2 then

        (

            append posxarray objects[i];

        )

        else if (getLocation objects[i] rect) == 3 then

        (
```

```
            append negyarray objects[i];
        )
        else
        (
            append negxarray objects[i];
        )
    )
```

Code Step B:

```
if posyarray.count > 0 do
    (
        local min = posyarray[1].min.y
        for i in 2 to posyarray.count do
        ( local temppoint = posyarray[i].min
            if temppoint.y < min do (min=temppoint.y)
        )
        posyaxis = upperY + ((min - upperY)/2)
    )
 if negyarray.count > 0 do
    ( local maximum = negyarray[1].max.y
        for i in 2 to negyarray.count do
        ( local temppoint = negyarray[i].max
            if temppoint.y > maximum do (maximum=temppoint.y)
        )
        negyaxis = lowerY - ((lowerY - maximum)/2)
```

```
    )
if posxarray.count > 0 do
    ( local min = posxarray[1].min.x
        for i in 2 to posxarray.count do
        ( local temppoint = posxarray[i].min
            if temppoint.x < min do (min=temppoint.x)
        )
        posxaxis = rightX + ((min - rightX)/2)
    )
if negxarray.count > 0 do
    ( local maximum = negxarray[1].max.x
        for i in 2 to negxarray.count do
        (
            temppoint = negxarray[i].max
            if temppoint.x > maximum do (maximum=temppoint.x)
        )
        negxaxis = leftX - ((leftX-maximum)/2)
    )
```

Code Step C:

```
( maxops.cloneNodes posyarray newNodes:&myposy
        for obj in myposy do
        ( local theMod=mirror mirror_axis:1
            addmodifier obj theMod
            theMod.mirror_center.position=[0,posyaxis,0]*inverse(obj.transform)      )
```

```
	)
	( maxops.cloneNodes negyarray newNodes:&mynegy
		for obj in mynegy do
		( local theMod=mirror mirror_axis:1
			addmodifier obj theMod
			theMod.mirror_center.position=[0,negyaxis,0]*inverse(obj.transform)
		)
	)   ( maxops.cloneNodes posxarray newNodes:&myposx
		for obj in myposx do
		( local theMod=mirror mirror_axis:0
			addmodifier obj theMod
			theMod.mirror_center.position=[posxaxis,0,0]*inverse(obj.transform)
		)
	)
	( maxops.cloneNodes negxarray newNodes:&mynegx
		for obj in mynegx do
		( local theMod=mirror mirror_axis:0
			addmodifier obj theMod
			theMod.mirror_center.position=[negxaxis,0,0]*inverse(obj.transform)
		)
)
tempmatrix = #(myposy,myposx,mynegy,mynegx);
	simulate tempmatrix upperY lowery leftx rightx;
)
```

CURRICULUM VITALE

Betül Tuncer was born in Istanbul in 1984. She graduated from the Koç School in 2002. In the same year she started her undergraduate degree at the University of Virginia School of Architecture in Charllotesville, Virginia. She received BS in Architecture degree in 2006. In the same year she started practicing architecture. In September 2007 she started her graduate degree in Architectural Design Computing in Istanbul Technical University.