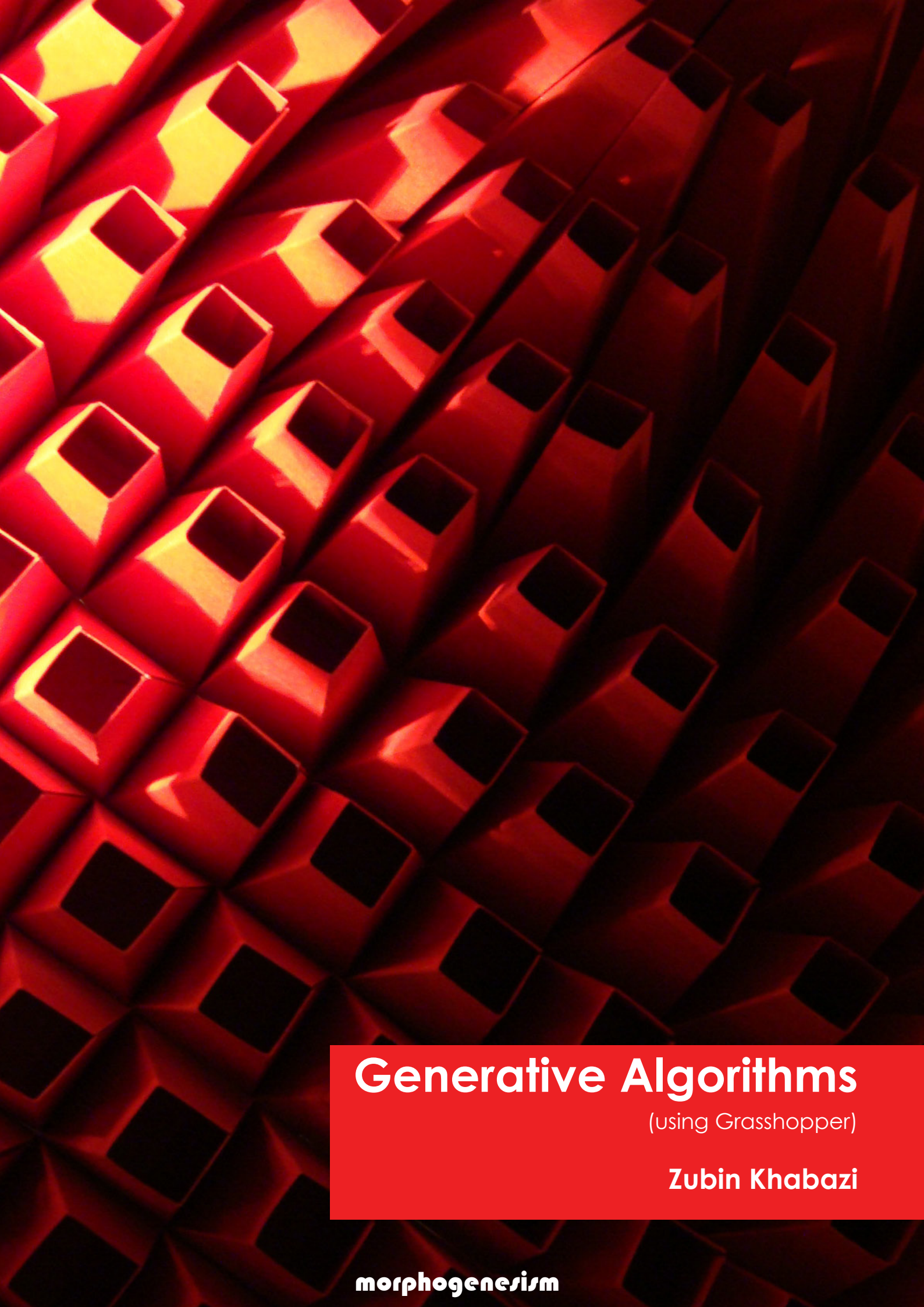


Generative Algorithms

(using Grasshopper)

Zubin Khabazi

morphogenesism

Generative Algorithms

(using Grasshopper)

Zubin Khabazi

morphogenesism

www.morphogenesism.com
zubin.khabazi@gmail.com

Intro

Algorithmic design and Grasshopper are developing rapidly. This is really interesting for design industry and the growing area of research and investigation in design practice. You are reading the third edition of the 'Generative Algorithms' which has been tried to be updated with most of the features in Grasshopper 0.8.0066. The book now has more to offer for those who like to learn designing with algorithms. I hope you enjoy reading it and learning Grasshopper.

I am sorry if there are grammatical or dictation or editorial errors in the text. This is because of the nature of the non-commercial publication, my english and MS Word which auto-corrects ! me.

Acknowledgements

I would like to thank Bob McNeel and Scott Davidson in Robert McNeel and Associates. I would like to thank David Rutten for his inspirations, innovations and supports as well; I also like to thank Michael Hensel, Achim Menges and Mike Weinstock from AA; Many thanks to Dr.Toni Kotnik and Stylianos Dritsas for their computation, scripting and advanced geometry courses in AA. I am extremely grateful to the students, architects and designers who contacted me and shared their knowledge with me from the first publication of the Generative Algorithms up to now.

Zubin Khabazi
August 2012

contents

PART ONE
DESIGN

PART TWO
FABRICATION

PART THREE
ANALYSIS

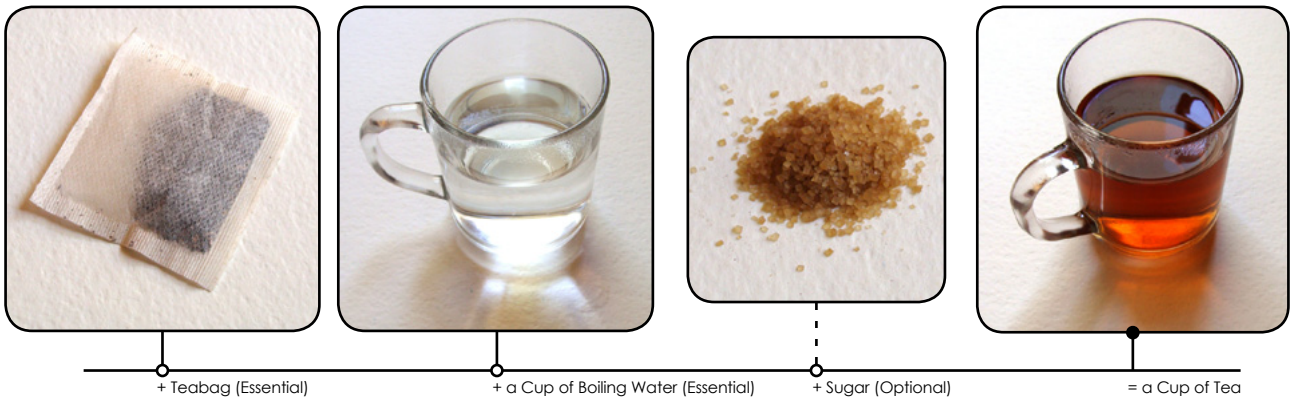
CHAPTER ONE

GENERATIVE ALGORITHMS

Chapter one
Generative Algorithms

It is widely discussed, criticised, attempted and somehow admitted that contemporary architecture as other areas of human activities like media, entertainment, science and technology, is dominated by computers and 'Computation' paradigm. In Design Industry, computers were first appeared as helping tools for facilitation of manual tasks which started the procedure of 'Computerization' through utilization of PC's and CAD software (Terzidis, 2006) in offices (which strongly affected the design industry). While the notion of computerization was the first step, utilization of computers is now certainly evolved into the era of 'Computation' in design processes (which tremendously affected the 'design thinking') (Menges, 2010). In this sense, computation refers to the act of calculation and reasoning in the information processing. It involves certain techniques and methods which deal with the subjects, processes and tasks that could be done through information processing and even raises the question of Computability and Incomputability (Flake, 1998).

Contemporary designers are dealing with 'Algorithms' as the model of computation to do their design tasks. An Algorithm is a set of rules and instructions in a step by step procedure to calculate, process data and do a defined task (more studies_Wikipedia: Algorithm). For any piece of data as input, an algorithm will perform its predefined operations and calculate the result. In this sense, a design algorithm will also provide a design output if being fed by relevant input information. While in conventional design systems, there were various parameters (i.e. Site, Program, Building Type, Facilities, Beauty, Structure …) which should be considered during the design process, in algorithmic processes it is attempted to transfer these parameters (input information) into algorithms to generate design solutions. What is currently known as Algorithmic, Parametric or Generative design software (plug-in/ Add-on/…) is the platform to do such design processes in computers via CAD software.



A (cup of) Tea Making Algorithm

Design Algorithms gather various types of information and in order to fit the needs of designers, produce 'Geometry' as output. To be able to accomplish this task, the marriage between algorithms (Computation) and geometry was necessary. This marriage happened in 'Computational Geometry'. Computational Geometry is a branch of computer science which uses algorithms to solve problems with geometrical aspects and outputs (De Berg, et al. 2000). For example triangulation of a polygon needs an algorithm to process data and the product is geometry. 'Generative Algorithms' (basically design algorithms) utilize computational geometry to produce design products. This book investigates how one can design these generative algorithms and what is needed to set up and run such algorithmic design processes.

DESIGN

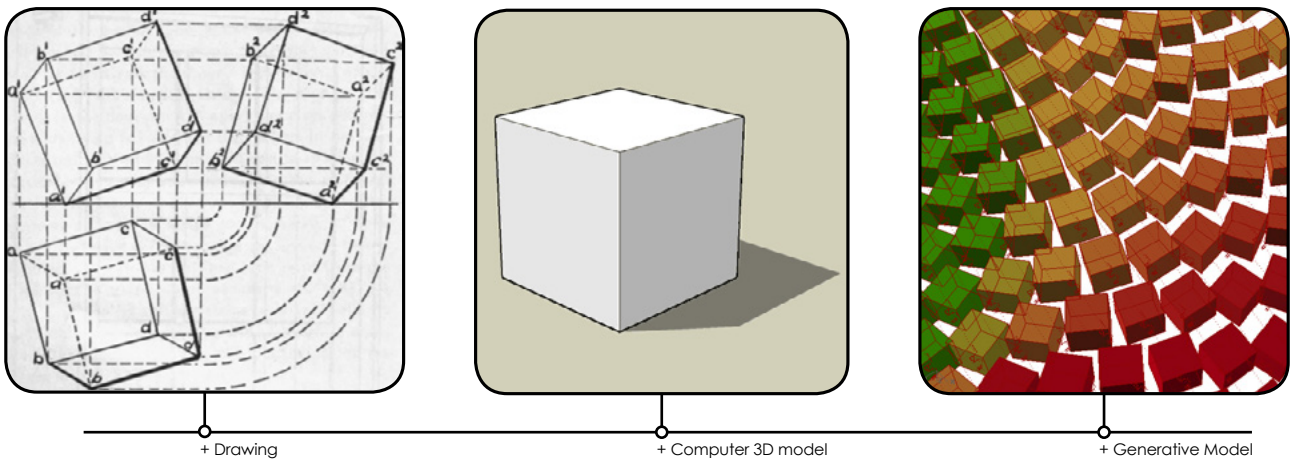
1_1_Design

The introduction of Computer-Aided Design (CAD) was one of the most prominent technological advancements in design practice. It helped designers in their drafting tasks and also enhanced modelling capabilities. Utilization of computer software in design, affected architecture even in style and in its early attempts yielded Blob Architecture (also known as Blobitecture).

CAD software made it possible to deal with more complex geometrical problems than classical geometry in Euclidean space. This was informally called Advanced Geometry which was capable of dealing with:

- Drawing, Modifying and editing various types of objects (lines, polygons, polylines …)
- Controlling the quality, shape, size and properties of curves, surfaces, volumes
- Free-form curves and surfaces (NURBS, Bezier, Meshes …)
- Boolean operations
- Complex Transformations
- Intersection, Trim and various editing features on objects
- Free-form editing, Conversion, Morphing and other complex operations
- Light, Material, Rendering and presentation of objects

The next step forward, when CAD software gained access to scripting and algorithmic tools, it became possible to 'code' or 'script' geometry with lots of design potential. All possible operations and commands in CAD become part of a code which could be applied to large number of objects in the design field. Computer aided design evolved to Generative design.



Cube Drawing in different time slots

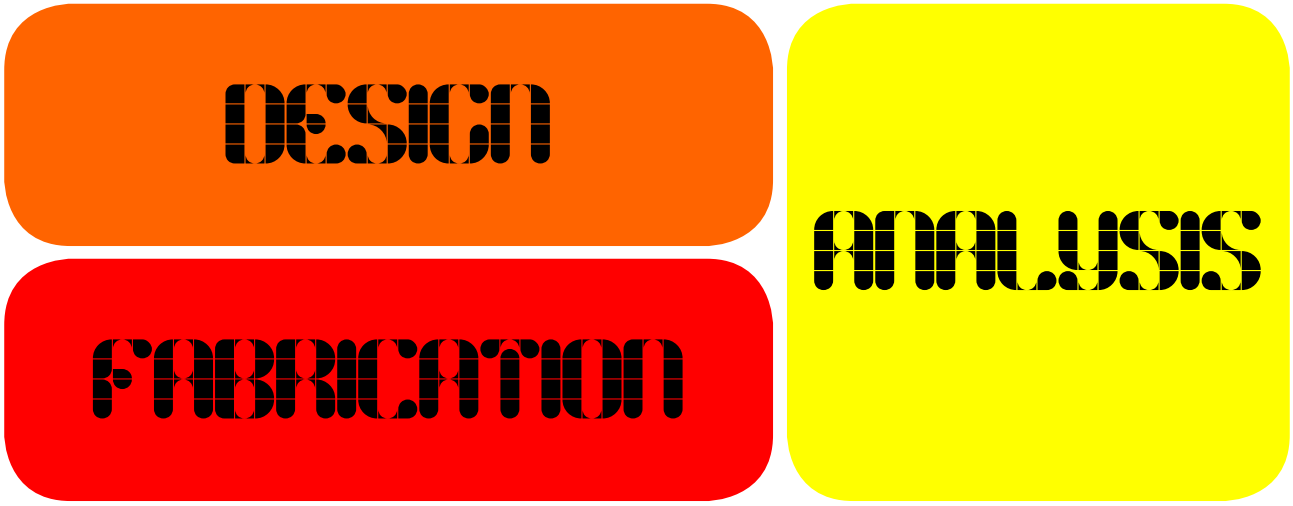
1_2_Fabrication

Design, manipulation and control of geometry of complex objects became possible in CAD software and computer in general. But the design by itself is not what architects and designers looking for. They also want to build what they design. From the moment that Blob architecture was presented in journals it raised the question of realization (construction per se). Is it possible to build these complicated, curvy, blobby objects? Current answer to the question of design and realization is much more elaborated than the age of Blob architecture.

Introduction of Computer-Aided Manufacturing (CAM) coupled the term Construction with Fabrication. While CAD and Algorithmic design features enabled architects to design complicated forms, CAM made it possible to build such complex products by using digital fabrication machineries. Various techniques have developed to utilize CAD/CAM technologies for fabrication of architecture. Different types of CNC machines with multiple heads, drills and beams helped architects to digitally fabricate their products in pieces and assemble them together to realize digitally-made objects. It is now becoming common to set up fabrication strategies of the project from early stages of design and embed their considerations in design algorithms from scratch (Menges, 2008):

- What is the material technology and material system of the project
- What is the potential, needs and necessities of employed material systems
- What is the technique of fabrication
- What is the machinery and its limitations and potentials which should be used
- What is the geometry, properties and vulnerabilities of the fabricated pieces
- What is the technique of attachments, joints, extra pieces, …
- What is needed for transportation and site preparation
- What is needed for assembly

Digitally designed architecture become informed by its fabrication necessities and the design product even optimized in order to accommodate such properties to avoid further complication or changes in the fabrication phase.



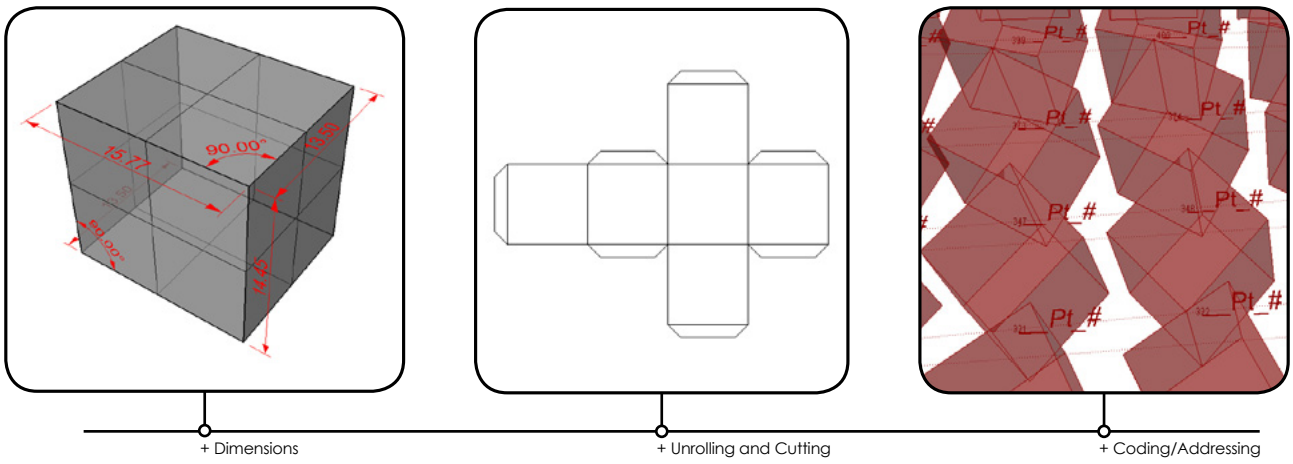
1_3_Analysis

Moving from design to fabrication encompasses a critical point: any architectural product, building, is a physical entity which lives in nature. Nature has many different phenomena, life/ environment rules and its currents and flux. It has forces and also continues processes of energy and material transformation. Any physical product in nature should deal with natural forces and its cycles of energy and material transfer. Building deals with these parameters in different ways:

- Building should have material to cover it from outside
- Building should deal with natural forces mainly through its structural capabilities
- Building should preserve itself from natural decaying factors
- Building should create comfortable internal climate, keep out heat/cold
- Building should warm up/cool down the internal weather
- Building should have light and electricity, gas and energy
- Building should prepare water, get rid of waste and sewage
- Building should be able to be absorbed by nature (recycled) after death

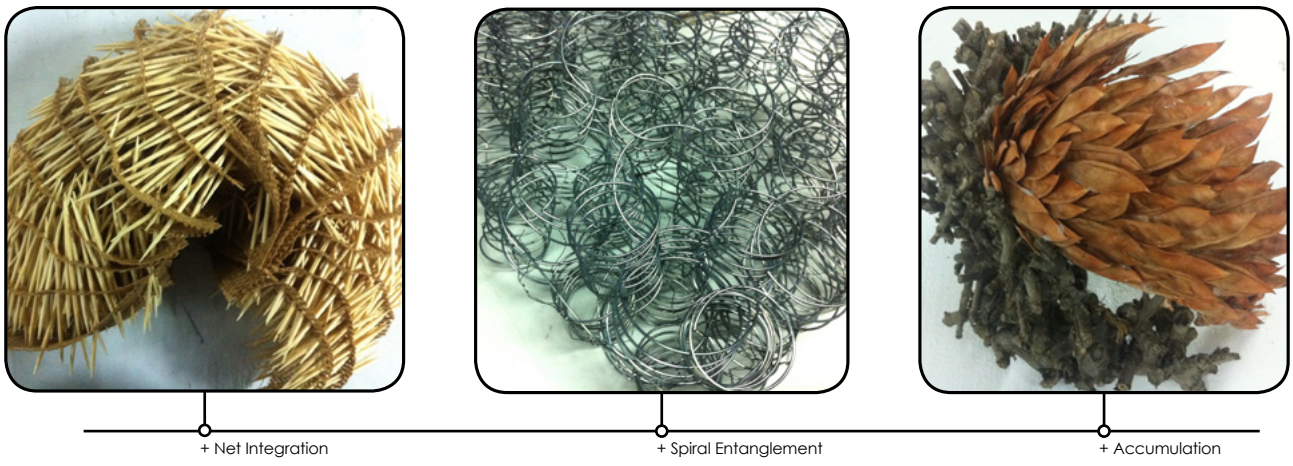
It is pretty much clear that building has material properties to deal with its persistence in nature and have energy and material transfer to maintain the life of its inhabitants. In conventional design strategies, it was 'Engineering' responsibility to deal with these subjects, usually after design stage. There are 'Active Strategies' which usually needs lots of energy, without consideration of recycling or reusing, to handle energy matters. There are structural considerations, to enforce a building, yet separate from architectural design.

After development of green architecture under Sustainable Development Paradigm, architects tried to move towards 'Passive Strategies' which encompass less energy consumption and more reliance on material properties and intelligent usage of available natural resources (Hensel, 2008). There is a growing interest to develop structurally informed architectural designs to consider structure as an integral part of the design process with less usage of material and more intelligent combination of existing systems; The same for material and energy informed systems.



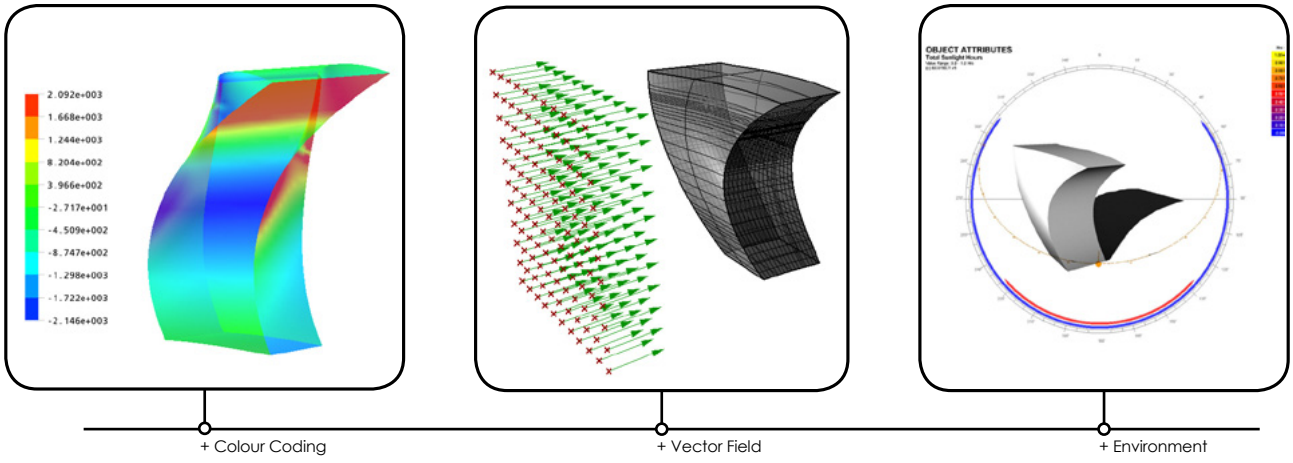
A Cube From Fabrication Point of view

Any Object in design has other different aspects if one looks at it from a Fabrication point of view. In this sense, the method of fabrication is as important as object by itself.



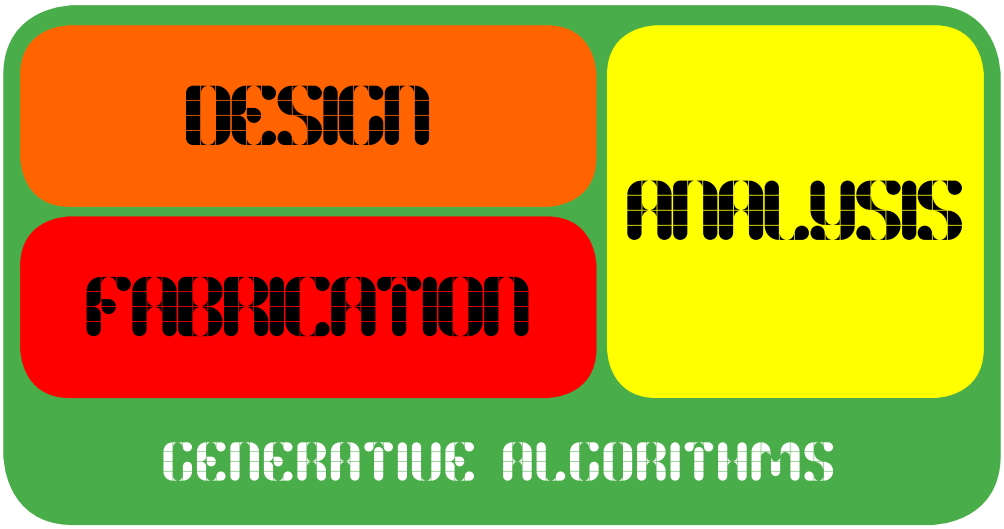
Looking at objects by their Material and Systemic Performance

Considering Fabrication and behaviour of any designed system, it should be noticed that material systems that are employed for the design have various potentials that could be implemented in design development stage which can add extra features to the product's quality.



A Cube from Analysis Point of view

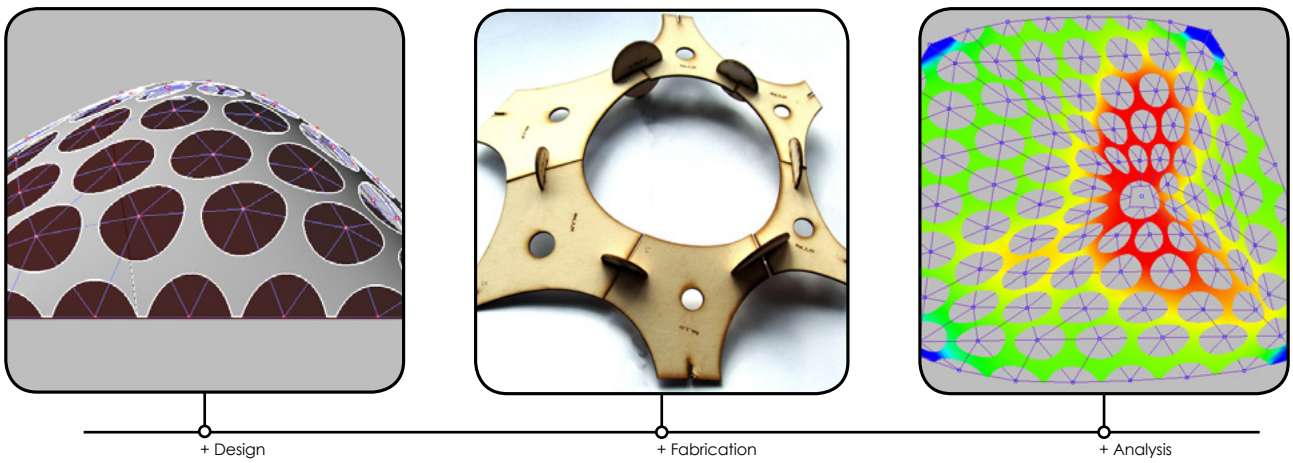
Any Object (in addition to its design features and fabrication neccessities) has material and energy behaviour in nature that should be considered and implemented in design Algorithms.



Contemporary algorithmic architecture tries to implement these ideas in early design stage (Leah, 2009). It becomes important to consider all these environmental properties and material effects in design, to develop material systems which have the potential to mediate environment and avoid postponing it to further engineering modifications. It does not mean that engineering work is omitted from design, yet there are several software, plug-in and add-on which can help designers to deal with physical aspects of buildings, and new type of engineering (i.e. Bio-Engineering) seems to be needed.

The analysis might start from environmental analysis (Sun exposure and Shade, Wind, Rain …), Structural Behaviour, Material Behaviour, Energy Consumption and so on which can help the improvement of design in different ways, but it could turn into a dynamic feedback loop which affects the design process continuously. These feedback loops could be considered as contemporary ways of criticising architecture to enhance its behaviour and get more successful results. This is the state of Self-Criticising in architectural design process.

'Generative Algorithms' is an experimental book to dive into algorithms with Design, Fabrication and Analysis aspects. These three main parts are presented with less theoretical descriptions and more emphasis on practical experiments, yet It has been tried to mix theory with experimentation, so it is not a software tutorial, but a book which helps to set up design strategies with algorithmic methods. The platform for this experimentation is **Rhino+Grasshopper**.



A Process of Design | Fabrication | Analysis (Porous Shell Project/Zubin Khabazi, morphogenesism)

CHAPTER TWO

PLATFORM

Chapter Two
Platform

Architectural design has its medium in order to deliver thoughts. There are sketches, diagrams, drawings and details usually on paper with different drawing tools or printed out of CAD software. There are study models, (architectural) models and detail models using various materials like foam and cardboard to represent the reality in small scale physical models or in 3D digital representations. Algorithmic Design needs its own medium as well; This medium (Here Grasshopper plug-in for Rhino environment) should provide facilities to deal with algorithms with geometrical operations.

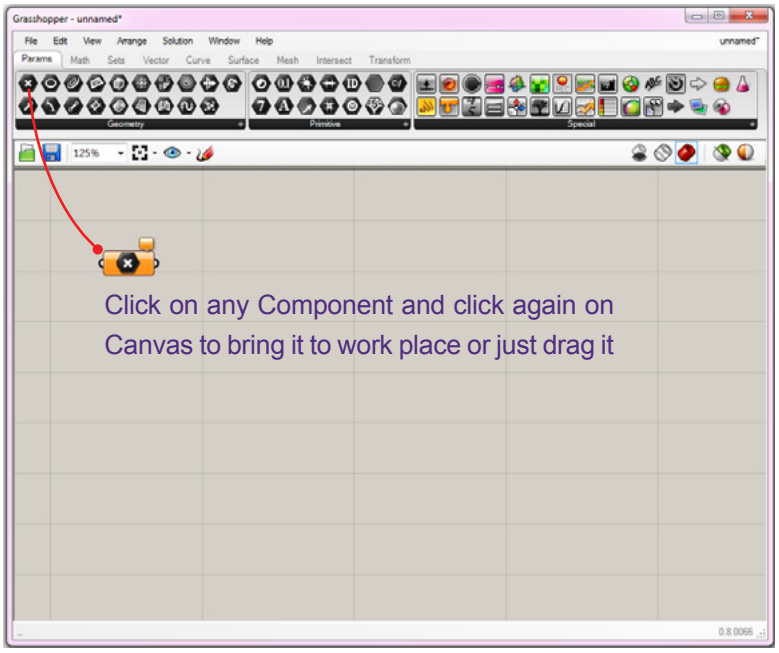
Algorithmic design has two main sides, one is 'Algorithm' and another one is 'Geometry'. Algorithm, like a recipe, manages and processes data, gathers input and provides desired output. Geometry is the ingredients where algorithms apply the recipe to them, and create the output product. Algorithmic design tools and any design medium in this field should provide facilities for both sides.

2_1_Basics of Grasshopper

2_1_1_Interface, Workplace

In contrast to the scripting platforms for algorithmic design, Grasshopper has a visual interface in which development of an algorithm could be seen like a flowchart. Beside other usual Windows menus, there are two important parts in the Grasshopper interface: Component Tabs and Canvas. Component Tabs provide all elements which are needed for algorithm or geometry purposes and Canvas is the work place, where to put Components and set up design algorithms. You can click on any component in any tab and click again on canvas to bring it to work place or just drag it to the canvas.

< Grasshopper Build 0.8.0066
(22 January 2012)
The one which is used in the book.



< Component Tabs

< Canvas

2_1_2_Components

There are different types of Components in Grasshopper which are divided in component tabs based on their geometry or function properties (i.e. to be a surface or to do transformations). Functions, commands and geometry operations are sorted in these components so the design algorithm should be set up by them.

To make everything simple, most of the components can do one of these functions:

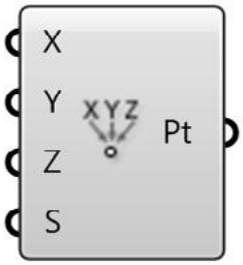
1. Provide data

2. Manipulate and Modify data

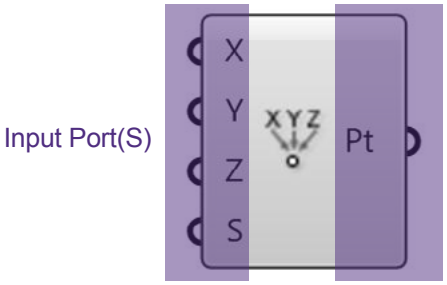
3. Draw Objects (Geometry)

4. Modify Objects

In this book all components are inserted in the text, using <> to address them clearly, like <Point>.



<Point> component

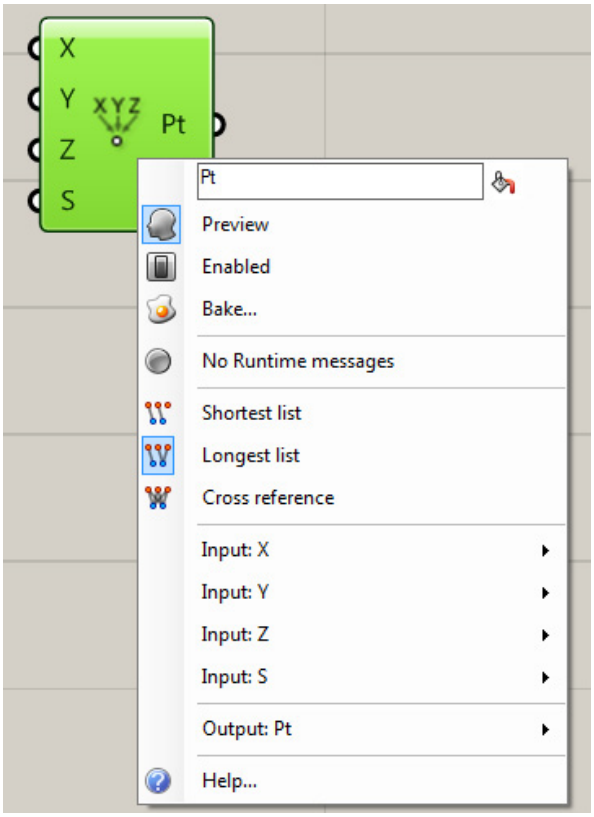


A normal component has two sets of ports: input and output. From the input (left side) it receives data and in output (right side) it provides the result. Some components are different because of the nature of the work that they perform.

2_1_3_Useful Features

Context pop-up menu

Right click on component will open up a menu which has some of the controlling features of it. The last part of the menu also offers a Help option which has some basic descriptions of the component's function. The first option let you set the name of the component if you like to change and customize it.



Preview

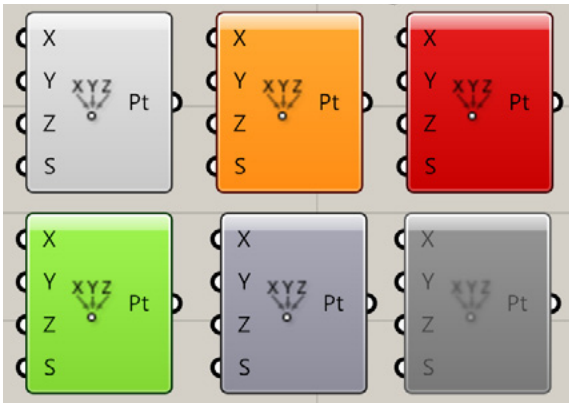
All components that produce objects in Rhino have 'Preview' option in their menu. It can be used to hide or unhide geometries in workplace. Any unchecked preview (Hidden output) turns the component colour to dark grey. Preview option can be used to hide undesired geometries like base points and lines in complex models to avoid distraction. This option in complex models helps to process data faster, so please hide your base geometries when you do not need them to be seen.

Enabled

If you do not need the function of a component at any stage temporarily, you can uncheck the 'Enabled' part so the component turns into dead grey and it does not provide output. This option helps in design process when you are testing various components to see which one suits your work. So before deleting any, you can make disable/enable without too much CPU usage for the benefit of the rest of algorithm.

Colour Coding

There is a colour coding system inside Grasshopper which shows components working status. Any grey component means there is no problem and the data defined correctly/the component works correctly. The orange shows warning and it means there is a problem that should be solved but the component still works. The red component means error and the component does not work in this situation. The source of error should be found and solved in order to make component work properly. The green colour means this component is selected. The geometry which is associated with this component also turns into green in Rhino viewport. Dark/Dead grey means the Preview/Enabled is unchecked.



Component's Colour Coding.

Type-In Component Search / Add

If you know the name of the component you want to use, or if you want to search it faster than shuffling components' tabs, you can double-click on the canvas and type-in the name of the component to bring it on. For those who used to work with keyboard entries, this would be a cool trick!

2_2_Basics of a Design Algorithm

An Algorithm is a set of tasks in order. It takes information, process data and generates result. Usually a component also takes some data from one/multiple source and gives the result back (a very small algorithm!). To set up a design algorithm, it is needed to provide data by relevant components, connect components together in the order of the task which they wanted to perform and get the result. So a design algorithm in Grasshopper is comprised of multiple components with their logical connectivity.

2_2_1_Input Data

Any architectural design starts with the analysis of various types of data (Site Analysis, Program, Structure …). This is the same for Algorithmic Design. The first step in Algorithmic design is to provide data but this data might be a bit different from the conventional one. In Algorithmic design, data usually introduced to algorithm as something sensible and understandable by computer which might be geometry, numerical data, text and image.

The first tab in Grasshopper interface is Params where it mostly dedicated to the components which can provide input data. This input data could be geometry (objects) which are available in Rhino and can be imported into Grasshopper by components in the Geometry section, predefined numerical or text values that can be set by components in Primitive section, or other mixed and different types of data from Special section of the Params Tab.

Predefined Static Data

There are various components in Params>Primitive which provide facilities to set predefined static data in design algorithm. For example a <Number> can be used to set one/multiple real number(s) for further applications. Or there is <String> that can be used to introduce some text to the canvas.

Defining external geometries

One of the most important resources of input data is geometry (drawn objects) from Rhino workplace. It could be a point, a curve, a surface, a drawing (plan, section, ...) up to multiple complex objects from any source that exists in Rhino. Any Geometry in Grasshopper needs a component in canvas to work with and for this purpose there are various components in Params tab, Geometry section to define external object.

After bringing the proper geometry component to the canvas, define a Rhino object by right-click on the component (context menu) and use "set one ... / set multiple … " to assign abject(s) to the component. Here the geometry from Rhino workplace should be selected and assigned. By introducing an object/multiple objects to a component it becomes Grasshopper object which can be used for any design purpose.

Let's have a simple example:

We have three points in Rhino viewport and we want to draw a triangle by these points. First we should introduce these points in Grasshopper. We need three <point> components from Params > Geometry > Point and for each we should go to their context menu (right click) and select 'set one point' and then select the point from Rhino viewport.

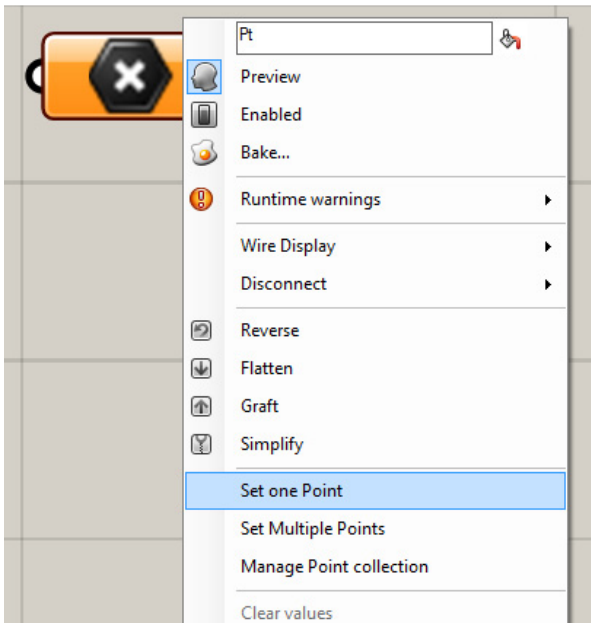


Fig.2.1. Set one point from Rhino in Grasshopper <Point> component.

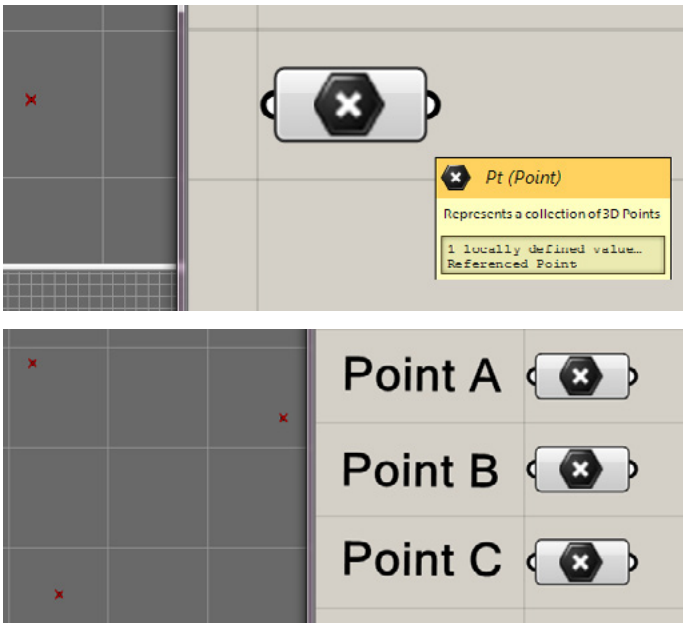


Fig.2.2. Grasshopper canvas and a point defined which turned to a red cross (x) in Rhino workplace. In the next step, Points A, B and C are all defined as three external points in canvas.

These are input data for generation of a triangle and in order to draw it, the triangle algorithm should be set up.

2_2_2_making Algorithms: Component Connectivity

The input data should be processed by the algorithm. In order to set up an algorithm, components need to be connected to each other to do a task in collaboration. Each component performs a specific task on its given data and provides the result that is needed for the next step. Components should be connected in the order that is desired for the performance of the task so the design algorithm would get shape little by little.

Going back to the example, now if you go to the Curve tab, in the Primitive section you will see a <line> component. Drag it to the canvas. Then connect <point A> to the 'A' port of the <line> and <point B> to the 'B' port (to connect components, just click on the semi-circle at the right side of <point> and drag it up to the other semi-circle on the target (A/B input port of the <line>). You can see that Rhino draws a line between these points.

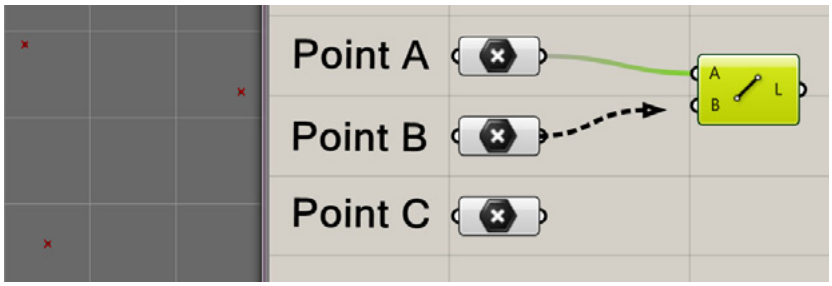
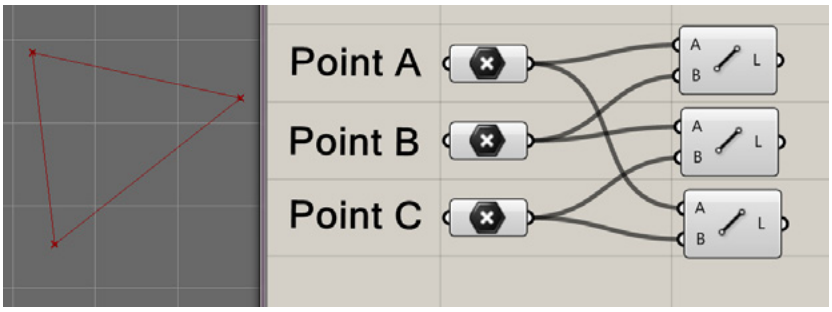
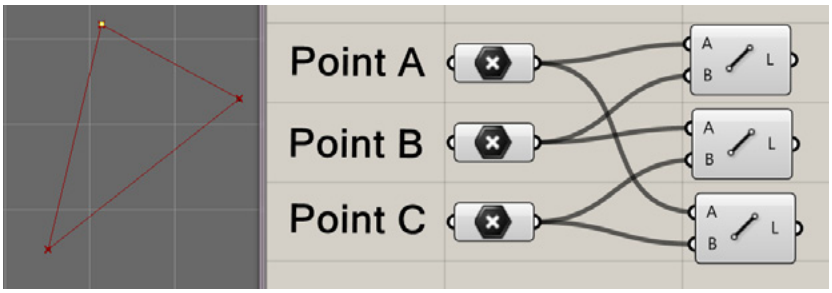


Fig.2.3. Connecting <point> components to a <line> component by dragging from output of the <point> to the input of <line>.



Now add another <line> component for <point B> and <point C>. Do it again for <point C> and <point A> with the third <line> component. Yes! There is a triangle in Rhino viewport. As you see in this example, any component can be used more than once as a source of data.



Now if you change the position of points manually in Rhino viewport, position of points in Grasshopper (X ones) and resultant triangle will change accordingly While lines between points (triangle) remain.

As you can see in this very first example, Grasshopper algorithm made it possible to manipulate points in Rhino and still have triangle between them. This happens because the triangle is now an algorithmic triangle which is generated based on the relations between elements of design rather than being drawn manually. The algorithmic relations define how geometries should be generated in design field.

Let's be Generative:

Now while the (Topological) definition of a triangle is set in the algorithm, it can be used for more triangle generation. When the output of such definition depends on the input data, one can provide enough input data to generate tens of triangles, hundreds, thousands, millions... .

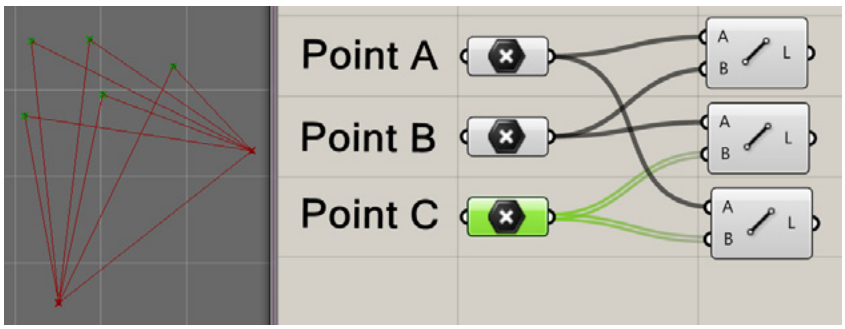
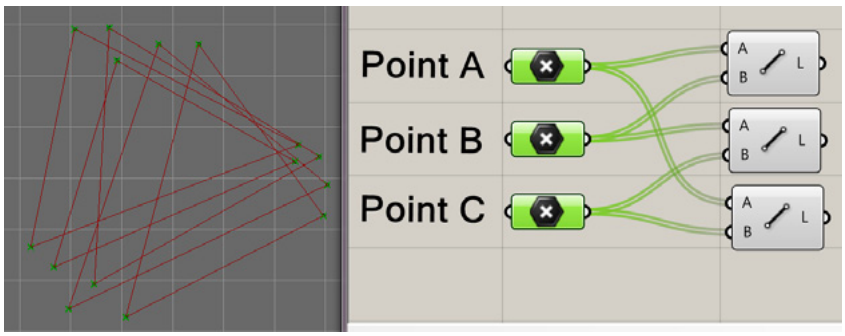


Fig.2.4. Additional Points to the <Point C> component would generate more triangles by the same definition.



Additional points to all <Point> components would result in multiple separate triangles in workplace, still using the same definition with the same amount of components and effort.

2_2_3_Output

Any component which provides geometry could be the desired output. In a series of components which are connected to each other, it usually rests in the right side of the algorithm. If the design is finished, one can use 'Bake…' option from components menu to literally bake it in Rhino, so it would turn into a selectable, real! geometry in Rhino. Notice that any further change in design algorithm would not take effect on the baked model.

2_3_Basics of Data Management

2_3_1_Data Type

Design with algorithms encompasses dealing with different types of data (Objects, Numbers, Texts, Booleans …). It is important to notice that any component should be fed by relevant data type. If a component needs coordinates, it should be fed by coordinate data and if it needs numbers, by numerical data type. Grasshopper sometimes substitutes data types in order to prevent errors. For instance if a component needs a point coordinate and fed by a Plane, it uses the plane Origin as the point coordinate to avoid error. Holding mouse over each port of the component will show a tool-tip that reflects the type of data which is needed/provided by that port of component.

2_3_2_Data Bifurcation: Multiple connections

It is always possible to use the output data of any component for more than one use. So the output can be connected to more than one component at any time and data will be copied. It is also the same for Input of data but with some considerations. Sometimes a component might be fed by more than one source of data. Imagine in the above example you want to draw two lines from <point A> to <point B> and <point C>. You can use two different <line> components or you can use one <line> and attach both point B and point C as the second point to the <line>. To do this, you need to hold Shift key when you want to connect the second source of data to a component, otherwise Grasshopper would substitute it. When you hold shift, the arrow of the mouse turns into green with a tiny (+) icon while normally it is grey. You can also use Ctrl key to disconnect a component from another one (normally you can disconnect a component using context menu). In this case the arrow of the mouse appears in red with a tiny (-) icon.

Connecting more than one component to an input port might cause some unexpected issues in your design. This needs more knowledge about data management in algorithmic design with Grasshopper which you will gain trough practice.

2_3_3_Data Matching

Grasshopper components usually work with lists of data (multiple inputs) instead of just one input and that's why it is generative. But when you are dealing with multiple inputs there might be the situation that the quantity of data from different sources does not match each other. This causes a situation that you need to Match various input data to get the desired result.

Look at this example:

There are two different point sets, each with seven points. Two <point> components are imported and points are stored in them using 'set multiple points'. All upper points are stored in one component and all lower ones in another component as well. As you see, by connecting these two sets of points to a <line>, seven lines are generated between them.

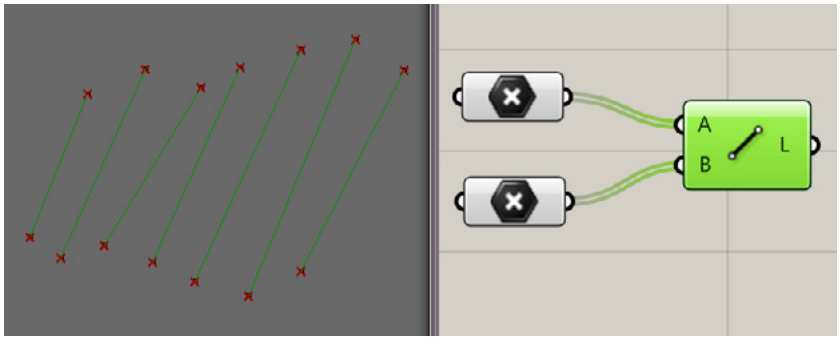


Fig.2.5. Multiple point sets and generating lines by them.

But what would happen if the number of points was not equal in two point (data) sets? In the example below there are 7 points in top row and 10 points in the bottom. Here 'Data matching' is needed to solve this issue. If you have a look at the context menu of the component you see there are three options called:

Shortest list
Longest list
Cross reference

You can see that the shortest list option uses the smallest list of data to make lines, and the longest list uses the largest data set while uses an item of the smaller list more than once. The cross reference option connects any possible two points from lists together. So it combines each item of the first list to all items of the second list. Notice that this option is memory consuming and sometimes takes a while for the scene to upgrade changes, sometimes causes crash if there are huge amount of data in each data list.

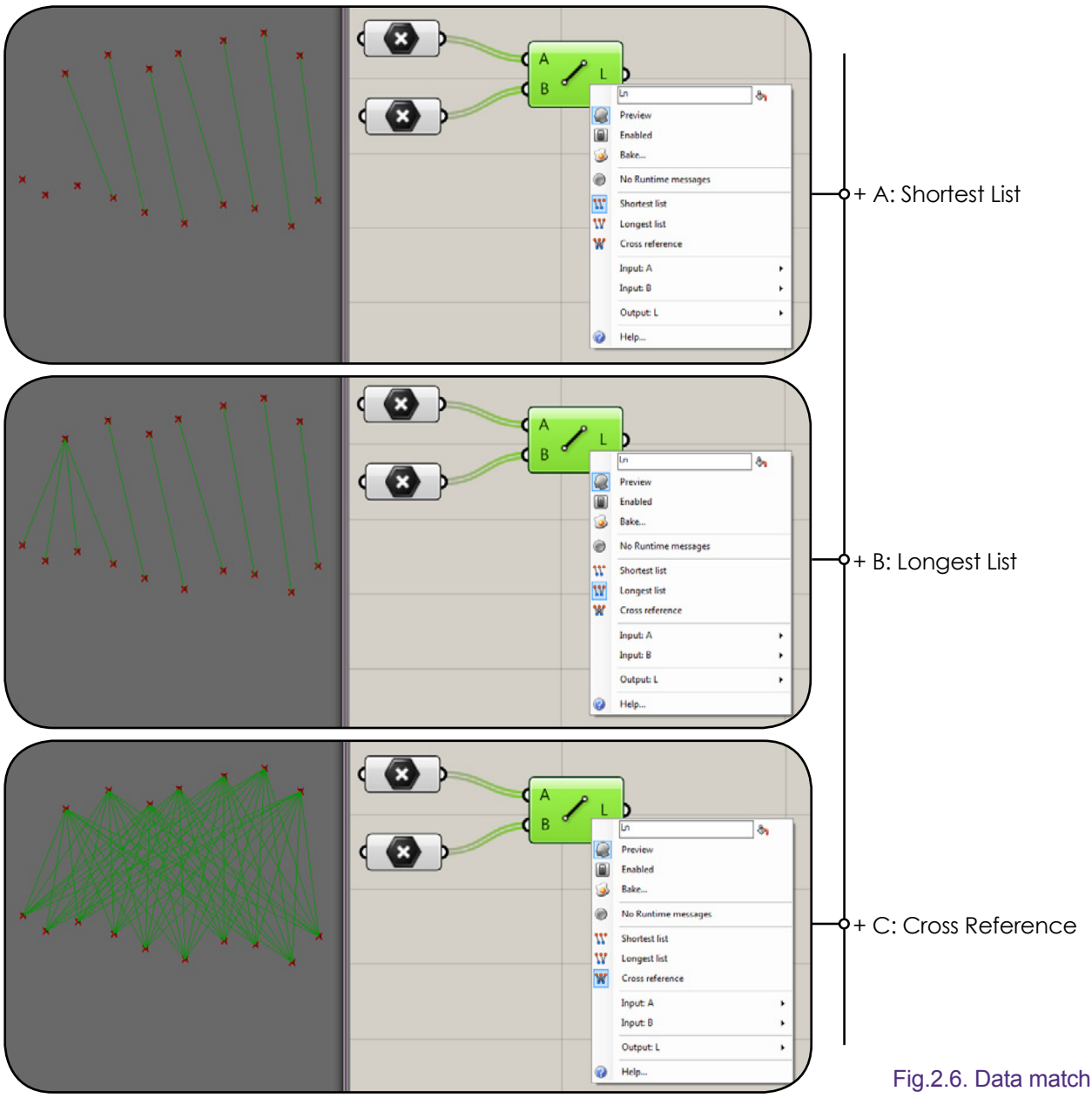


Fig.2.6. Data matching

2_3_4_ Data Tree Control Panel

Working with Grasshopper data lists, it should be considered that data is not 'Flat' and in a single list all the times and it might be divided or 'Branched' in various data branches, making a 'Data Tree'. Data trees are described later but notice that options like 'Graft' and 'Flatten' are in components Input/Output menu to control this level of data management. If you realize that your algorithm has components which are connected to each other by dashed lines (in Fancy Wire mode) then you can make sure that data in these components are divided into various branches and some data tree management options might be needed to get the desired result. You will learn it in following chapters.

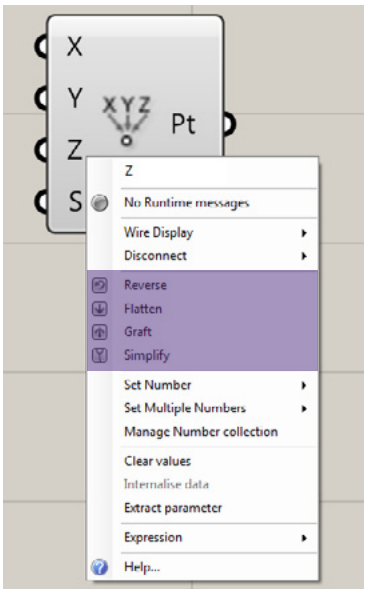


Fig.2.7. Data Tree/Branch control panel

PART ONE
DESIGN

CHAPTER THREE

GENERATING GEOMETRY

Chapter Three
Generating Geometry

Part one of the book will focus on [Generative] Design. As discussed a little bit in chapter two, the idea behind generative (algorithmic) design is to discover the relations and rules of objects in design field and set up this rules as design algorithms to be able to generate design products. Since these rules work with algorithmic processes, they need input and provide output, and it mentioned that based on the input data one provides, the algorithm would generate output, no matter hundreds or thousands. It is just the matter of data which should be processed.

Data is the basic ingredients to be analysed and considered in any design practice. In Generative design, data should be converted into values which are recognizable by algorithms. These include Numerical data, Strings, Booleans.... Designer needs to provide various data types for design algorithms.

3_1_Sketching by Numbers

Algorithms start with math and numbers. Numbers are hidden codes of the universe. Numbers and math are language of nature, they are everywhere. There are numerical values, numerical sequences and domains in Grasshopper which should be explored to start sketching through computation.

3_1_1_Numerical Value(s)

There are components which can provide one or multiple numerical value(s).

The most useful number generator is <Number slider> or simply <Slider> component (Params > Special > Number slider) that generates one number which is adjustable manually. It could be integer, real, odd, even and with limited lower and upper values. You can set these by 'Edit' part of the context menu. For setting one or a group of fixed number(s) you can go to the Params > Primitive > Integer / Number to set one/multiple integer/real number(s). There are other numerical components like <Digit Scroller> which generates an adjustable number.

Numerical Sets:

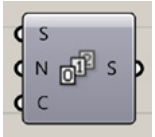
Series of numbers

We can produce a list of discrete numbers by <series> component (Sets > Sequence > Series). This component produces a list of numbers which starts from the 'first number' and grows by the 'step size' and the number of values in the series can be set to limit the amount of values.

(first:0/step:1/No:100): 0, 1, 2, 3, … , 99

(first:2/step:2/No:50): 2, 4, 6, 8, … , 100

(first:10/step:10/No:1000): 10, 20, 30, 40, … , 10000



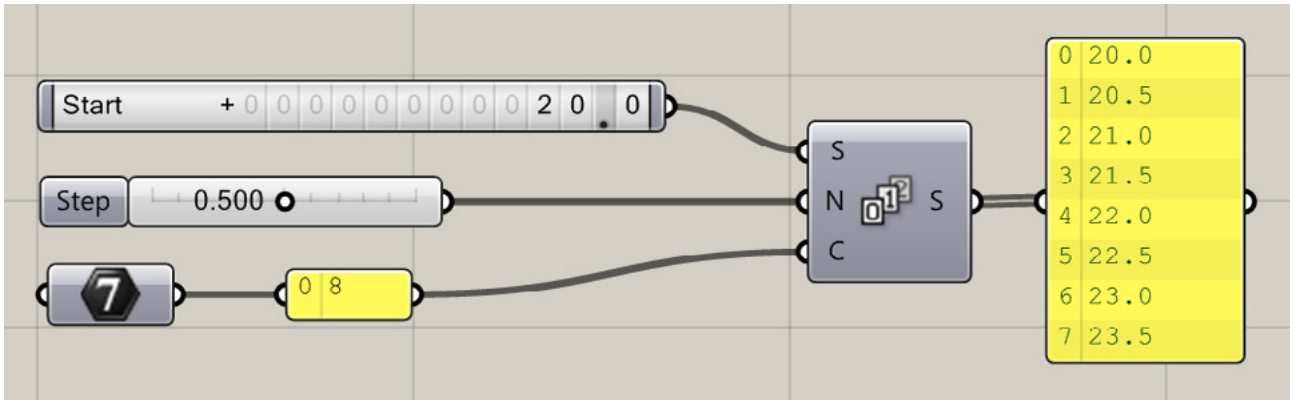


Fig.3.1. A <Series> component has been set to generate a series of numbers start from 20, and grow by the step size of 0.5. The number of values in the series set to 8 to generate 8 numbers (index 0 to 7). The yellow components are <Panel> from Params > Special which show the contents of any component attached.

Domain

Domains provide all real numbers between a lower and upper limits. There are one dimensional and two dimensional domains and various components to create and work with them. Domains by themselves do not provide numbers. They are just extremes, with upper and lower limits.

Range of Numbers in a Domain

There are uncountable beauties in math. One of them is that between any two real numbers, there are infinite real numbers. There are infinite numbers between 1 and 2. There are also infinite numbers between 1 and 1.000000001. Having upper and lower values of a numerical domain, it is possible to divide it by evenly spaced steps and produce a range of numbers. With a defined domain, one can set the steps between lower and upper limits to get a range of numbers (Sets > Sequences > Range).

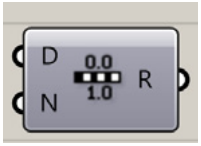
Any numeric domain (i.e. from 1 to 10) can be divided into parts to create numbers:

1, 1.5, 2, …, 10

1, 2, 3, … , 10

1, 2.5, 5, … , 10

1, 5, 10



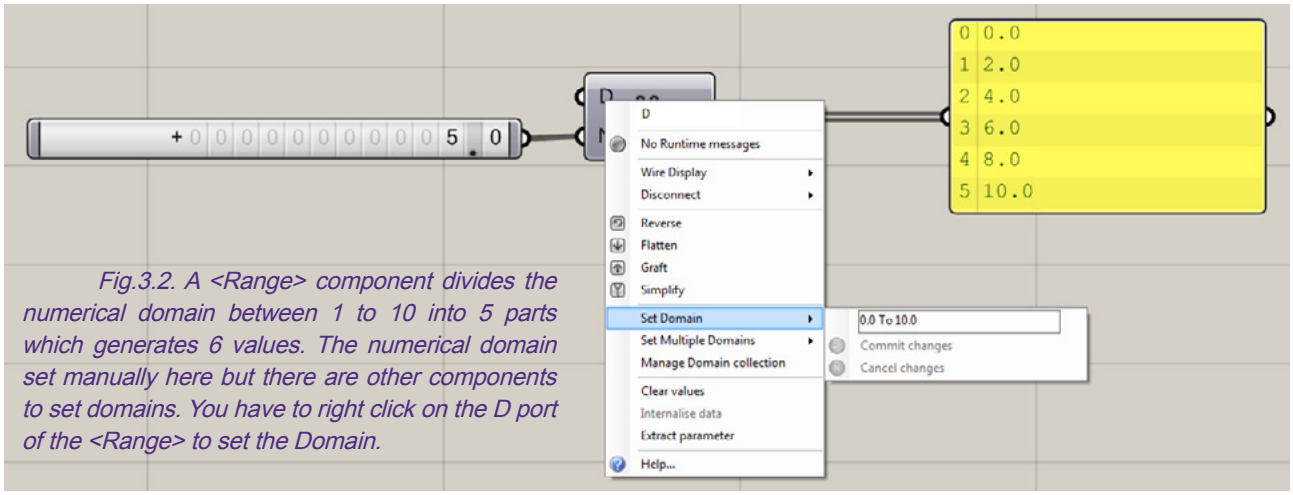


Fig.3.2. A <Range> component divides the numerical domain between 1 to 10 into 5 parts which generates 6 values. The numerical domain set manually here but there are other components to set domains. You have to right click on the D port of the <Range> to set the Domain.

3_1_2_Points and Point Grids

Points are among the basic elements for generating geometries in Generative Algorithms. As points mark a specific position in space, they can be start points of curves, centre of circles, origin of planes and so on. In Grasshopper we can generate points with various approaches. Let's see how we can relate numerical data types and point geometries.

- We can simply pick a point/bunch of points from Rhino and introduce them to workplace by <point> component (Params > Geometry > point) and use them for any purpose (These points could be adjusted and moved manually in Rhino scene and affect the whole project. Examples on chapter_2).
- We can produce points by their coordinate values. To do so, we need a <Point XYZ > component (Vector > Point > Point XYZ) and feed coordinates of the points by numbers.
- We can make various types of point grids like <grid hexagonal> from Vector > Grids.
- We can extract points from other geometries in different ways like endpoints, midpoints, etc.
- Sometimes we can use planes (origins) and vectors (tips) as points to start other geometries. There are also other options to generate points in Vector > Points.

You have seen the very first example of making points in chapter_2 but let's have a look at how we can produce points and point grids by numerical values.

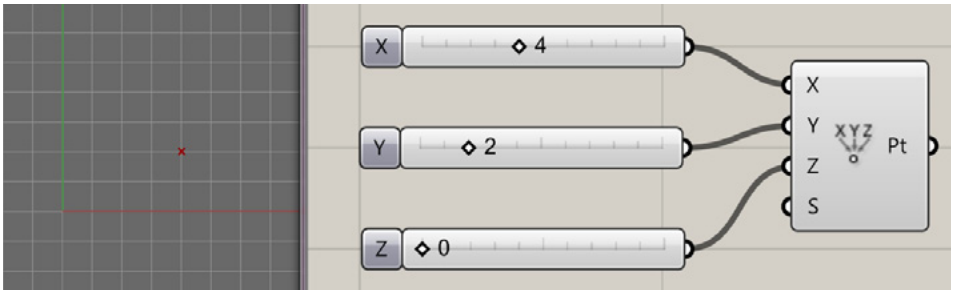
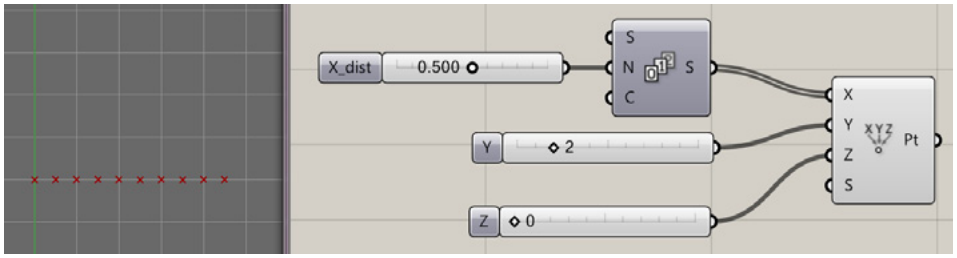
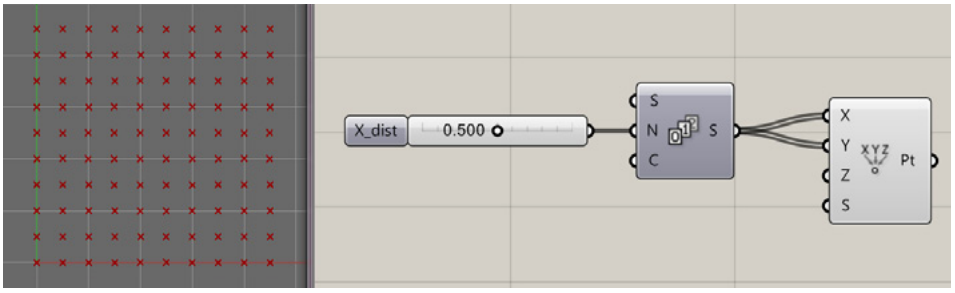


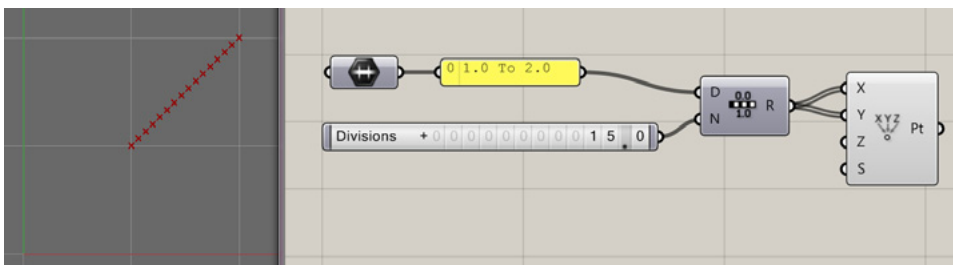
Fig.3.3. Feeding a <Point XYZ> component by three <number slider> to generate a point by manually defined X,Y and Z coordinates.



Generating series of points using a <Series> component to provide series of numbers instead of one number.



Generating a grid of points by <series> and <Point XYZ > components. The data match of the <Point XYZ> set into Cross Reference to make the grid.



Dividing the numerical domain of 1 to 2 by 15 using a <Range> and feeding a <Point XYZ> component with 'Longest list' data match. The <domain> component is in the Params > Primitive.

3_1_3_Operations and Functions

Predefined numerical components in Grasshopper might be insufficient to generate objects. Although it is possible to generate numerical sequences, but how one can calculate the Sine of these numbers or other math operations? Math operations are simple and straightforward, available in Math > Operators. Functions are components which are capable of performing math functions in Grasshopper (Math > Script). A function component should be fed with relevant data (not always numeric but also Boolean, String) and it performs a user defined function on the input data. To define the function you can right-click on the (F) port of the component and type it or go to the Expression Editor. Expression Editor has many predefined functions as well as a library to select from.

Pay attention to the name of variables you use in your expression and the associated data you match to the function component!

Math functions

A Mathematical Graph shows how data are linked together. Graph for a function *f* is all pairs of point coordinates (x, *f*(x)) which is presented in graphic. Let's draw some function graphs.

What is the Math Graph for all x values from -3 to 3 for the function $f(x)=x^2$?

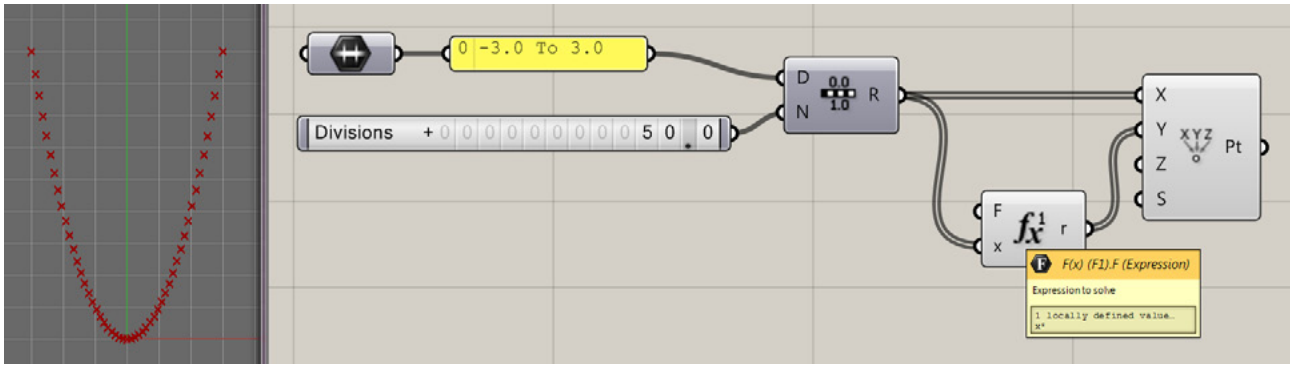


Fig.3.4. What we need is to define x values, calculate f(x) and draw the Graph. Since we have the lower and upper limits of the X values, we can define a one dimensional domain from -3 to 3 and divide it by a <Range> component to get X values in between which should be used for a <Point XYZ>. To find Y values which are f(x), a <Function> is needed to calculate x^2. The function is defined in the F port of the component. if you right click on F port you can find an Expression Editor option, in which you can set your user functions. The graph shows these pairs of numbers (x, f(x)) as points.

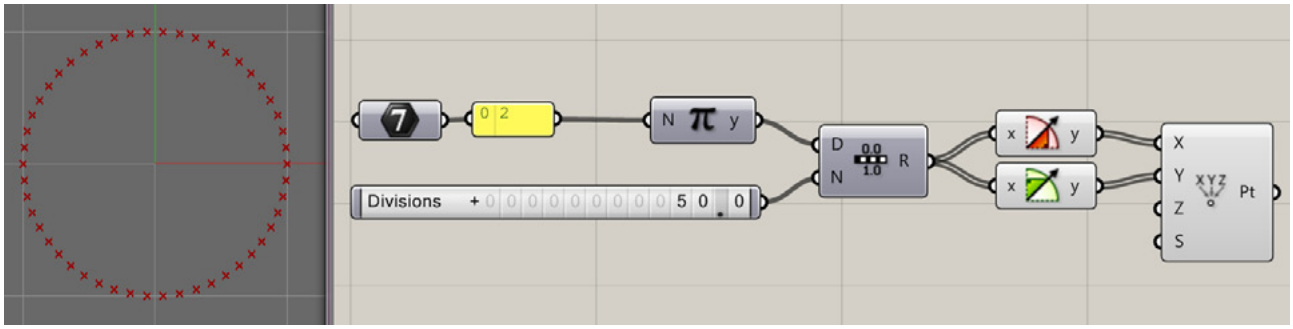


*Fig.3.5. To make it a bit more complicated, we can draw a circle by point coordinates. The mathematical definition of a circle is X=r * Cos(t) and Y=r * Sin(t) while (r) is the radius and (t) is a range of numbers from 0 to 2Pi. All (t) values are provided by a domain from 0 to 2Pi divided by a <Range> to calculate X and Y values in radian. Number of segments in <Range> confirms how many points we need to produce the circle. <Sin> and <Cos> operations are in the Math>Trig.*

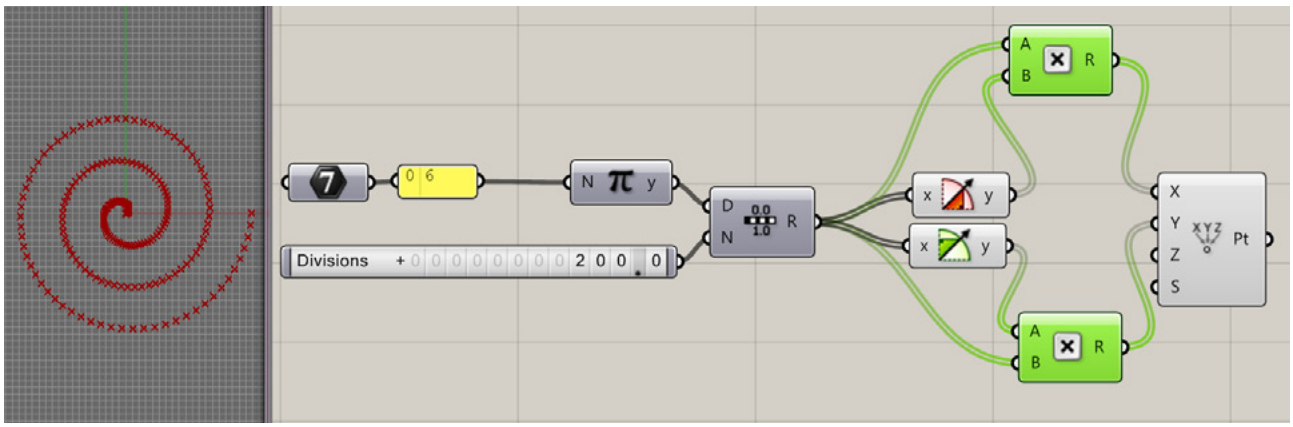


Fig.3.6. A <range> of numbers from 0 to 6Pi is divided into 100 parts. The resultant numerical values are used to feed the <Point XYZ> component through the following math functions:

*X = t * Cos(t)*

*Y = t * Sin(t)*

Since the Cos(t) and Sin(t) have multiplied by (t) again, we can see that points started to move away from the center and they have generated a spiral form.

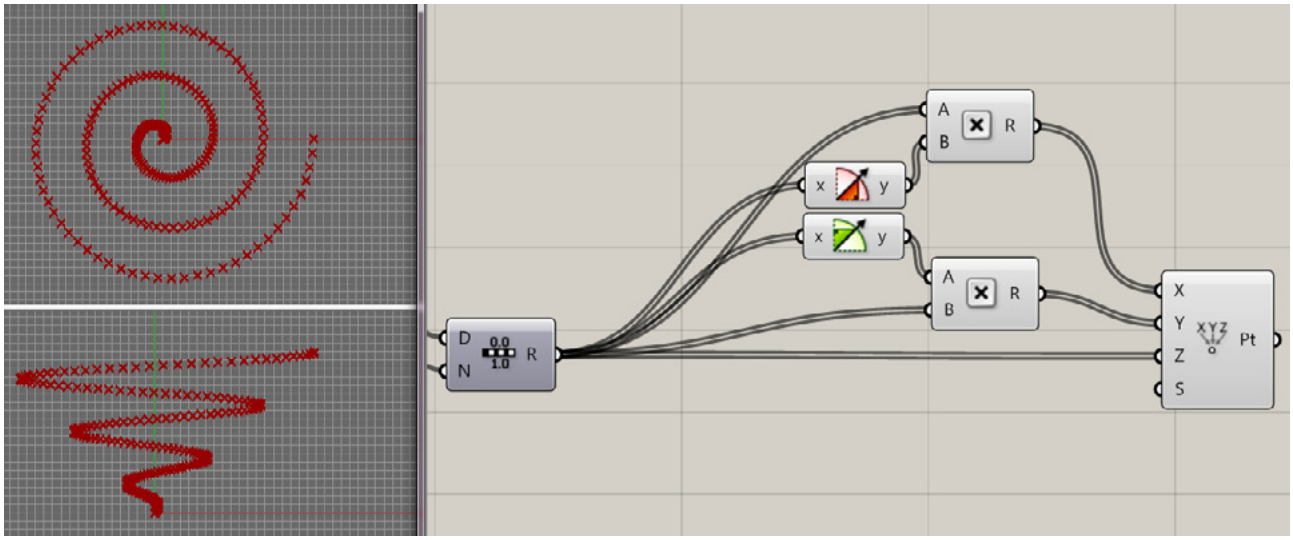
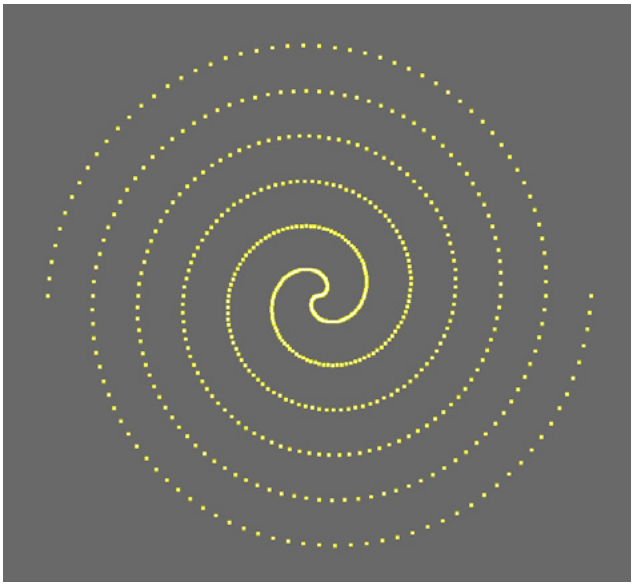


Fig.3.7. So far all Z values of points were 0. Using the same values of <Range> component as Z values of points would cause them to grow and we can convert the Spiral to a Helix.

There are various types of spirals like Fermat's spiral, Cornu spiral, Hyperbolic spiral, Lituus spiral and so on. All can be formulated and implemented in functions to draw with Grasshopper. Playing around math functions could be endless. You can find various mathematical resources to match your data sets with them. Here the idea of math operations and functions which can change the original set of data is important for further design purposes.



3_1_4_ Random values

Contemporary design encompasses complexity and controlled randomness. Designers do not like to follow order of number series or classical geometries. Adding a controlled randomness will increase the complexity of project in a desirable format. There are various ways to deal with random values in Grasshopper.

We can use Vector > Grids> Populate 2D/3D to generate a population of randomly distributed points in a defined boundary. This would help to set up some design elements in random positions.

So far we managed to draw a circle by its mathematical definition. In such circles there were points across the boundary which were evenly distributed. What if we wanted to distribute these points in random distances?

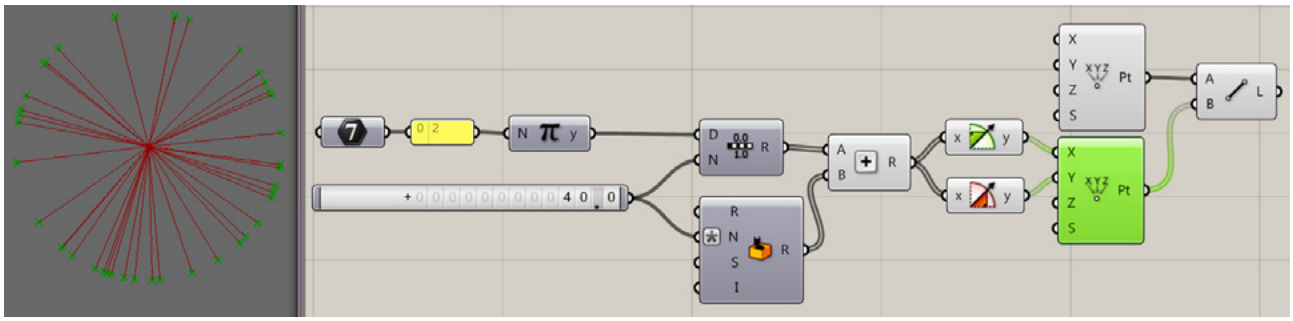
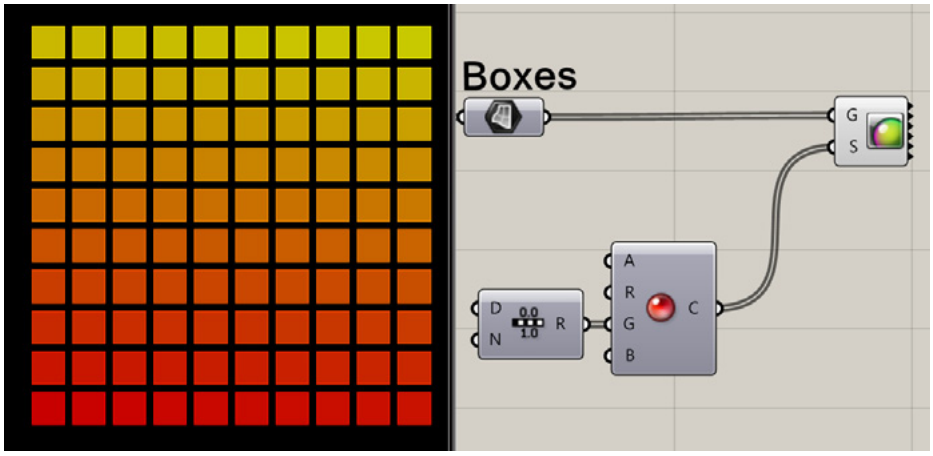


Fig.3.8. Following the same concept as circle generation, here a random value has been added to the numerical values which are going to feed sine/cosine operations. As it is clear in the scene, points that make the circle are distributed in random distances because the numerical values are now changed from their original state of even division. Note that the number of items for the <Random> component is added by one in order to produce the same amount of numbers as <Range> component (N+1 defined internally in N port of <Random> and marked by a star (*) to make sure there is something extra in the component).



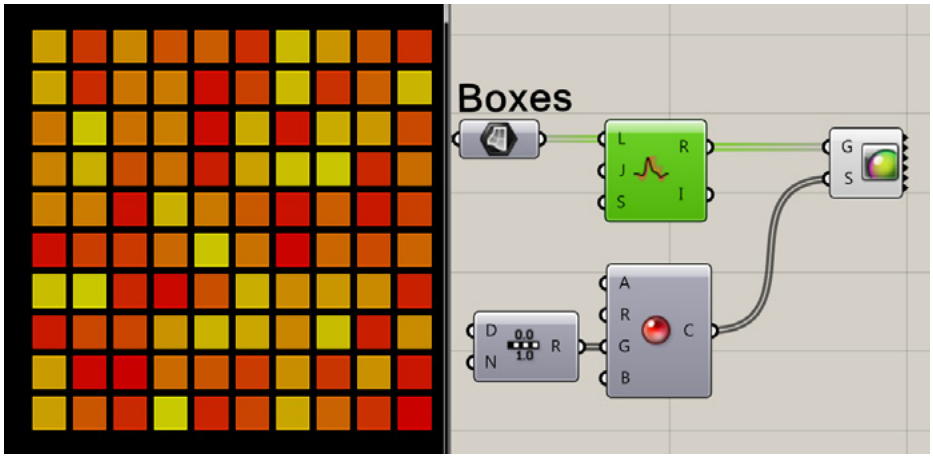


Fig.3.9. In this example, there are couple of boxes and it has been aimed to make a custom (colour) preview in order to show the application clearly. In the first image, all boxes are in an order which makes them accept a range of colours from top-right corner to bottom-left corner one after each other. But in the second example, a <Jitter> component has been added after boxes. This component shuffles a list of data so distribute items randomly between each other. As a result boxes' colours are in a random order although they are still in the same position and they accept the same colour range. (Components for colour are <Custom Preview> (Params>Special) and <Colour RGB> (Vector>Colour)).

3_2_ Reasoning

3_2_1_ Making Decisions

Generative Design is not limited to linear progress of production, it sometimes needs decisions and critical thinking. Based on the design situation, designer sometimes needs to limit some actions, branch out progression for different conditions and so on. As in generation, decision also should be done through progress of data.

Data is not limited to Numbers. There are other data types which are useful for different purposes in programming and algorithms. If we want to decide whether to do a function or not then we need conditional statements in programming ('If' statements). In conditionals, we check if a statement meets certain criteria or not, and based on that, perform a function or not. This needs an specific data type which is called Boolean.

3_2_2_ Boolean Data types

A response to a conditional 'question' is a simple yes or no. in algorithms we use Boolean data to represent these responses. Boolean data types are only True (yes) or False (no) values. If the statement meets the criteria, the response is True, otherwise False. With Booleans, we can talk to the algorithm to perform functions or not, to select some part of the objects, to bifurcate the progression of algorithms and so on.

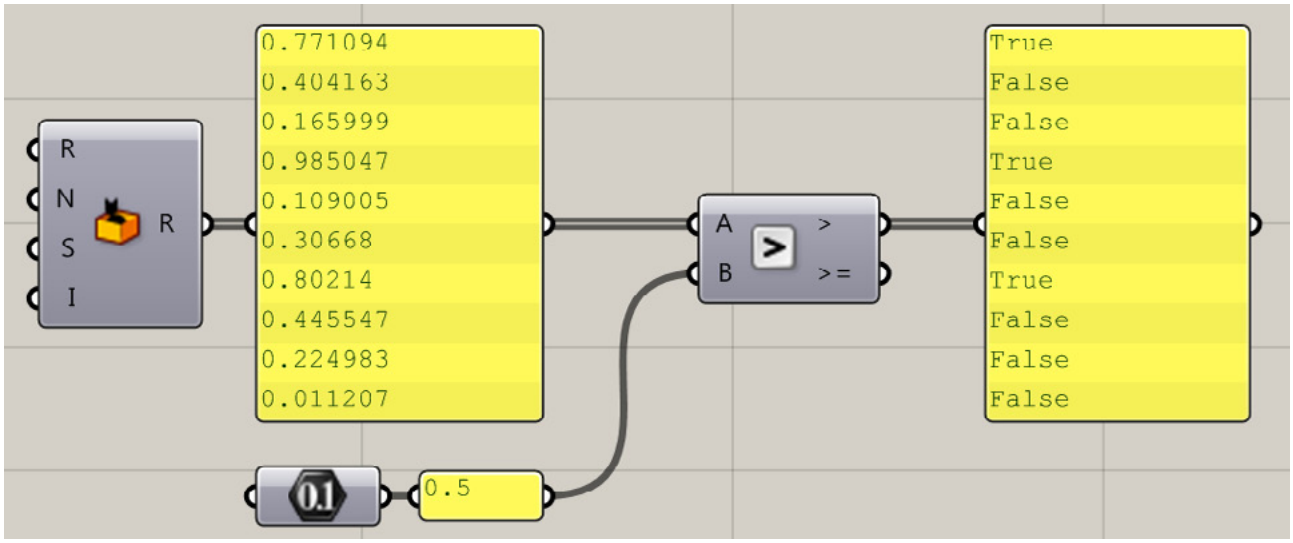


Fig.3.10. There are ten <random> values. We want to see if these random values are bigger than 0.5 or not! If yes, then we want to run other function by those values, others should be omitted. Using a <Larger> component (Scalar>Operators) was made it possible to compare all random numbers with 0.5. Whenever random numbers meet the criterion, the <Larger> passes 'True' for values bigger than 0.5, otherwise 'False'.

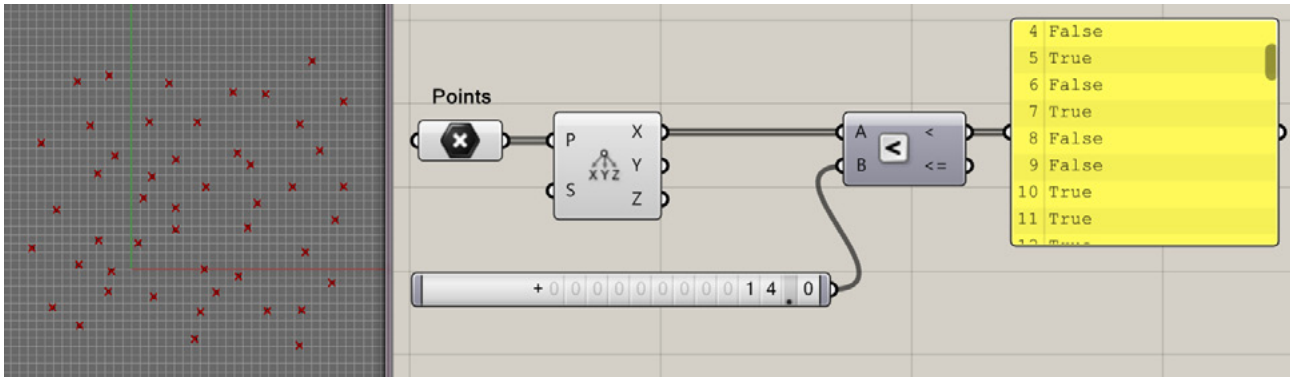


Fig.3.11. Let's try this concept in the context of geometry. The question is about points. There are bunch of points in the workplace and we want to find those points that their X coordinates are less than 14. We need to compare X coordinates of points with 14 using a <Smaller> component. We have True in the <panel> for points which their X<14. Point coordinates are extracted by <Decompose> from Vector>Point (The reverse of <Point XYZ>).

So as you can see in these examples, there are possibilities to examine criteria by numeric values and get Boolean data as result. But sometimes, we want to see if the situation meets different criteria, and we want to decide based on the result of them. For example based on the above example, we want to find those points that their X coordinate is less than 5 but their Y coordinate is more than 3. We know how to find points with their X coordinates less than 5 and points with their Y coordinates more than 3. How we could find points that meet both of these criteria? To find the result, we need to operate on the result of both functions: these are Boolean Operations. There are bunch of these operations under the Math > Boolean. They help to work and decide with more Boolean values.

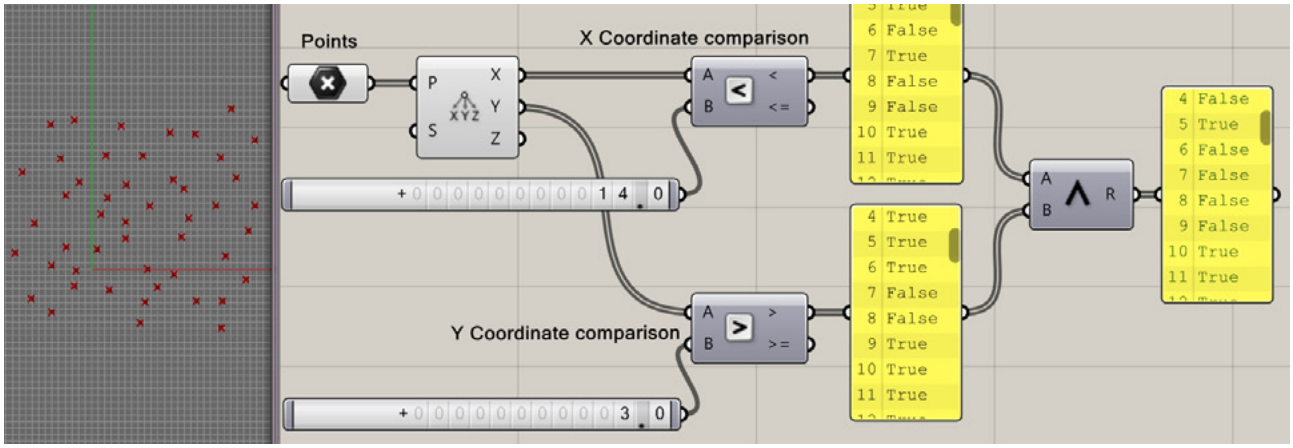


Fig.3.12. Here both concepts are combined. Points' X coordinates compared with 15 and Y coordinates compared with 3 separately. Using a <Gate And> component (Math > Boolean > Gate And) will perform Boolean conjunction on them. The result would be True when both input Boolean values are True, otherwise it would be False. Those points which meet both criteria will pass True.

There are multiple Boolean operators that you can use and combine many of them to create your criteria, make decisions and set up your design based on these decisions. Combination of these operations help to control what is desirable in design and what is not and should be omitted. But before going deep into these operations, let's see what we can do with these Boolean data.

3_3_ Data Manipulation 1_ Data Lists

3_3_1_ Culling Lists

There are many reasons that we might want to select some of the items from a given data list and do not apply a function to all. To do this, we either need to select some of the specific items from a list or omit other items. There are different ways to achieve this but let's start with omitting or Culling lists of data as data manipulation factors.

There are different <Cull> components in Grasshopper Set's tab which help to omit some part of data from a data list (any type of data). While <cull Nth> omit every N item of a given list of data, <cull pattern> takes a pattern of Boolean values (True/False) and cull a list of data, based on this pattern, means any item of the list that associates with True value in Boolean list passes and those that associate with False, omit from the list.

If the number of values in the data list and Boolean list are the same, each item of the data list being evaluated by the same item in the Boolean list. But you can define a simple pattern of Boolean values (like False/False/True/True which is predefined in the component) and <cull> component would repeat the same pattern for all items of the data list.

Distance example

I am thinking of selecting some points from a point set based on their distance to a reference point. Both point set and the reference point are defined by <point> component. First of all what I need is a <distance> component (Vector > Point > Distance) that measures the distance between points and the reference and as a result it provides a list of numbers (distances). I compared these distances by a user defined number (<number slider>) with a <Smaller> component. This comparison generates Boolean values as output (True/False) to show whether the value is smaller (True) or bigger (False) than the <Slider>. I am going to use these Boolean values to feed the <Cull pattern> component.

As mentioned before, <Cull pattern> component takes a list of generic data and a list of Boolean data and omits those members of the generic list of data who associate with 'False' value of the Boolean list. So in this case the output of the <Call pattern> component is a set of points that associate with True values which means they are closer than the specified number shown on the <number slider> to the reference point. To show them better I just connected them to the reference point by a <line>.

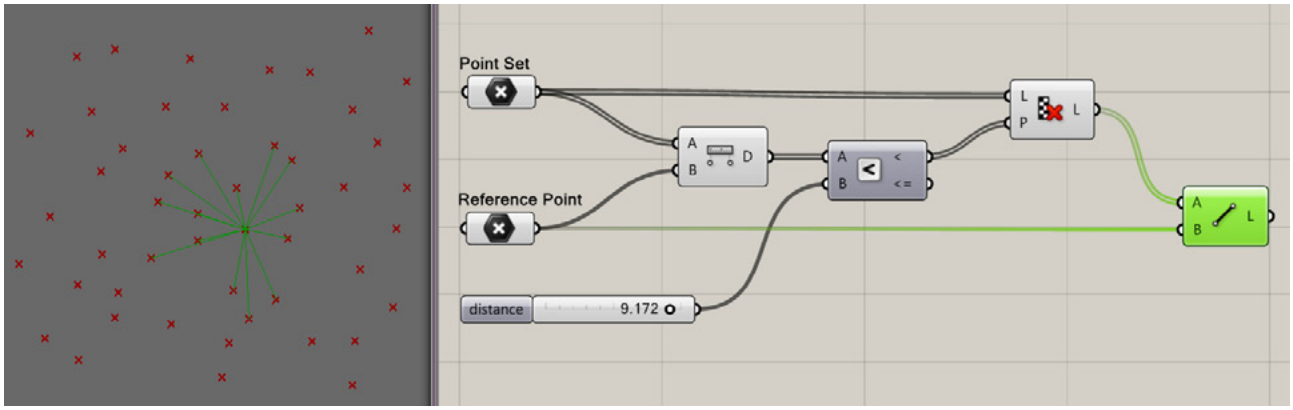


Fig.3.13. Selection of points from a point set by their distance to a reference point, using <Cull pattern> component.

Height example

There are bunch of points which are associated with the topography lines close to an antenna somewhere in the world! The idea is to select points higher than 25 meters in topography lines and also closer than 650 meters to the antenna.

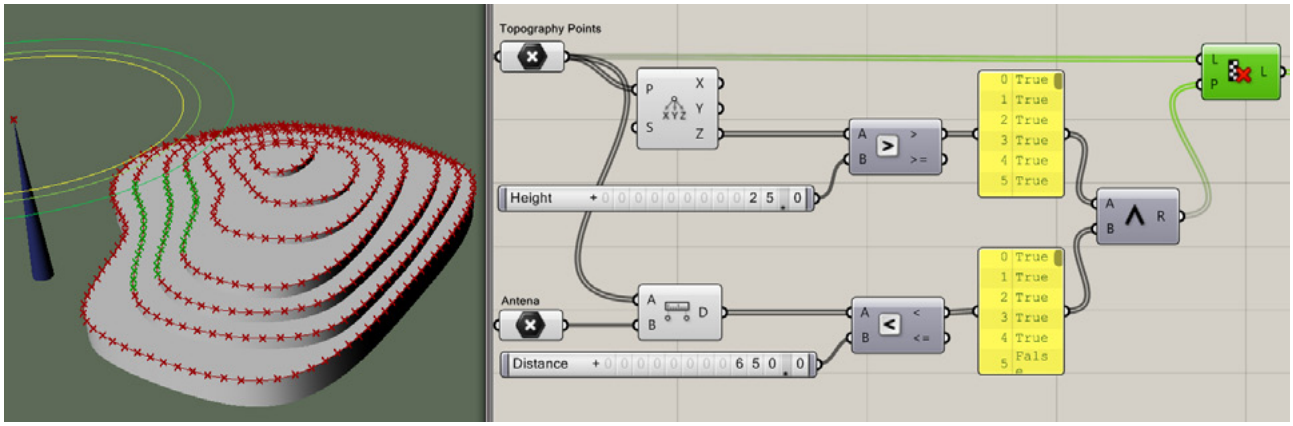


Fig.3.14. Since there are two different criteria, points should be compared first by their height (Z coordinates) to the predefined height and then their distances should be calculated from the antenna to select those which are closer. By using a <Gate And> Component, we are sure that points passed by the <Cull Pattern> component meet both criteria, higher than 25 and closer than 650m to the antenna. These points are selected for further design purposes.

3_3_2_ Data List Management

It is clear now that one of the basics of the algorithmic design is data and data management. Data lists might be any sort of data like numbers, points, geometries and so on. Any geometrical operation easily provide a data list as a results. It is important to control this data lists, in order to get desired output. This needs data manipulation.

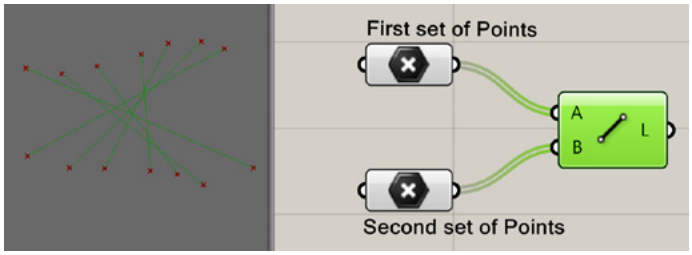


Fig.3.15. There are two lists of points and we want to connect them together by lines but when I connect them we see that the order of points in lists are not the same and what we get is not what we want. Even in this very simple example we see that some sort of data management is needed in order to get the appropriate result.

There are various components in Grasshopper which help to control and manage data lists but for now, looking at the Sets>Lists. There are multiple components that can be used for data manipulation. It is possible to extract one item from a data list by its index number, extract part of a list by the lower and upper index numbers, reverse the order of data in a list and so on. Look at some examples:

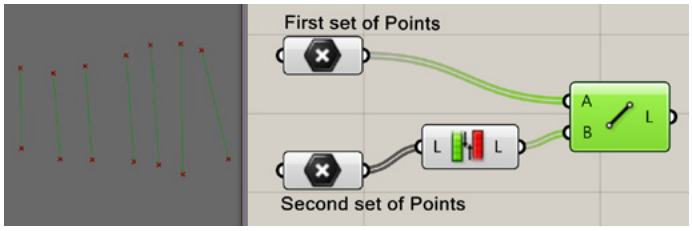


Fig.3.16. Although a very simple situation, to solve the problem of the disordered point lists, here one of them are <Reverse>d and then connected to the <Line> and you see that lines are now connected without intersecting each other.

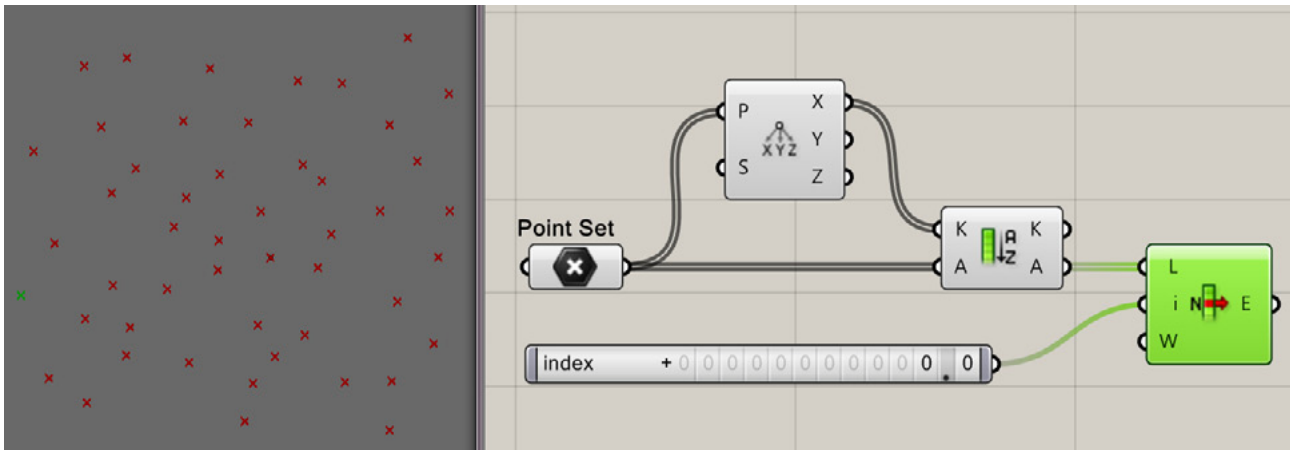


Fig.3.17. Here there is a list of points. I want to select the point with lowest X coordinate. As I said before, a <point decompose> component gives us the coordinates of points. What I need to do is to find the minimum X value of all X values of points. To achieve that I need to sort all these X coordinates from lowest to highest to find the minimum (the first item in the list). This is what <Sort List> will do for me. Basically <sort> component, sorts a list (or multiple lists) of data based on a numeric data list as sortable keys, so when it sorts numbers, the associated data could be sorted as well. So here I sorted all points with their X coordinates as Key data. What I need is to select the first item of this list. I need an <Item> component which extracts an item form a list by its index number. Since the first item (index 0) has the minimum X value, I extracted index 0 from the list and the output of the component would be the point with the minimum X value in the point set.

How can I get the point with highest X coordinate? We can select the last item in the data list by extracting its index number. But what if the number of points in the data lists change? Then we should reset the index number. What if we want it for 1000 different data lists? Then we should change the index number every time? Here it would be better to reverse the list of points to select the first item of the list again. Then we would be sure that this point has the higher X value in the list!

There is a data list of circles which was generated by another algorithmic process and they are basic geometries to create a cylinder. The agenda is: there are two types of cylinders; Normal cylinders for circles smaller than 1 meter in diameter and porous cylinders for bigger circles. How we should set up our algorithm to be able to generate cylinders by the above criteria?

Here we need to bifurcate data and the <dispatch> component will do that for us. This component needs a Boolean data list for evaluation to bifurcate data into two lists (associate with True/False). The criterion is circles' radius.

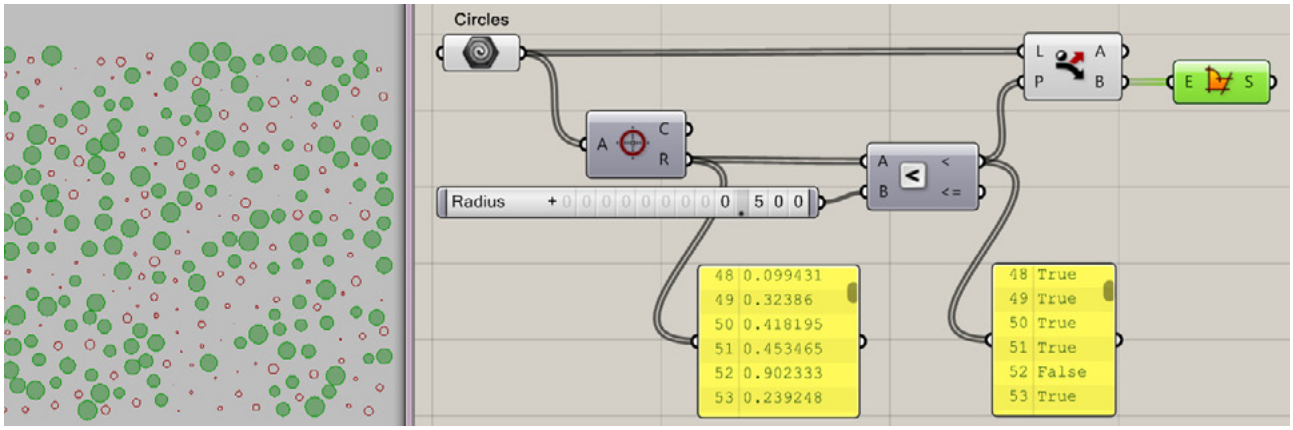


Fig.3.18. Circles' radius are calculated by a <Center> component (Curve > Analysis) and compered to the defined value of 0.5 (radius). The resultant Boolean data list has been used to dispatch data. So the diameter of circles in the A output of the <dispatch> are smaller than 1 meter and the B output represent bigger circles for further design applications. For the purpose of visual clarity, bigger circles are converted to surfaces.

Combination of various data manipulation components are usually needed to get the desired data for design applications. Since all data manipulations are for design purposes let's do a more interesting design experiment.

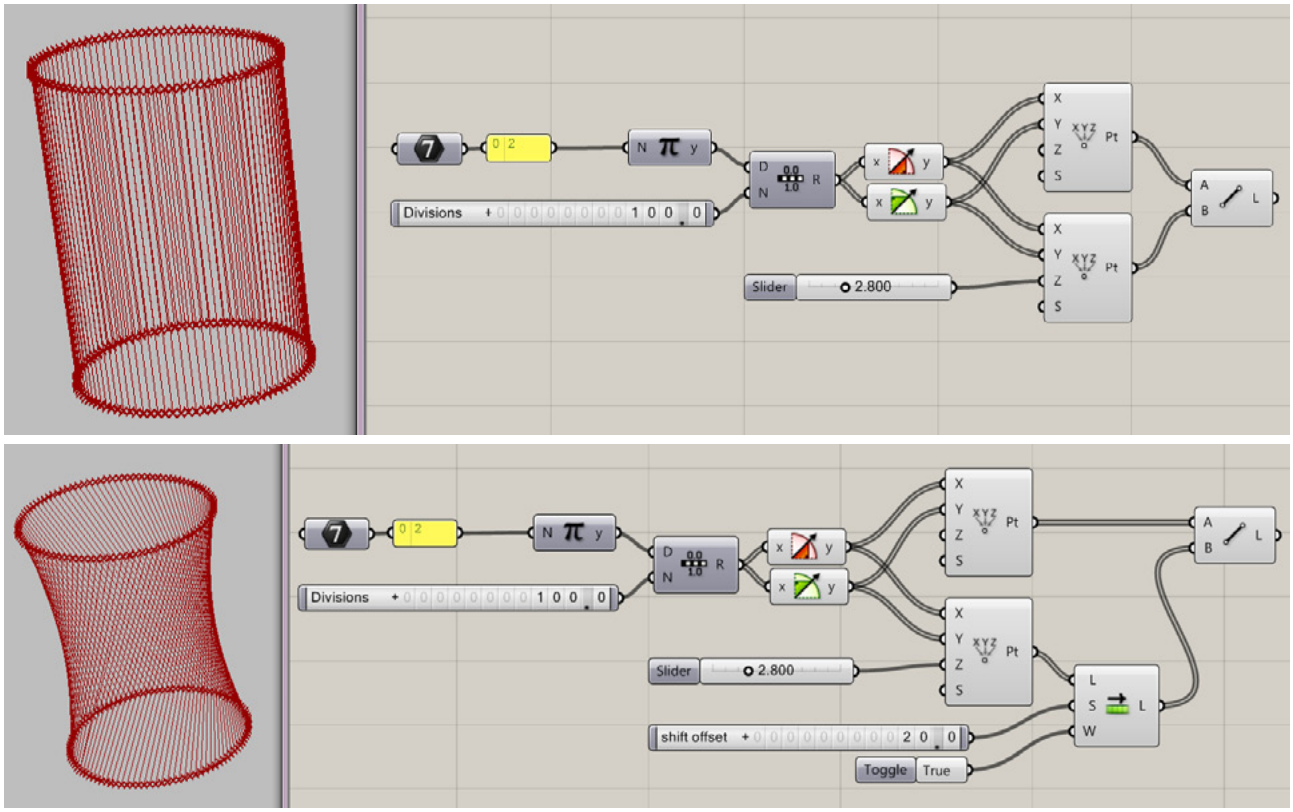


Fig.3.19. Two circles have been drawn while the second one has higher Z values and all points are connected by a <Line>. Using a <Shift> component would shift points in their list which can help us to generate a simple ruled surface.

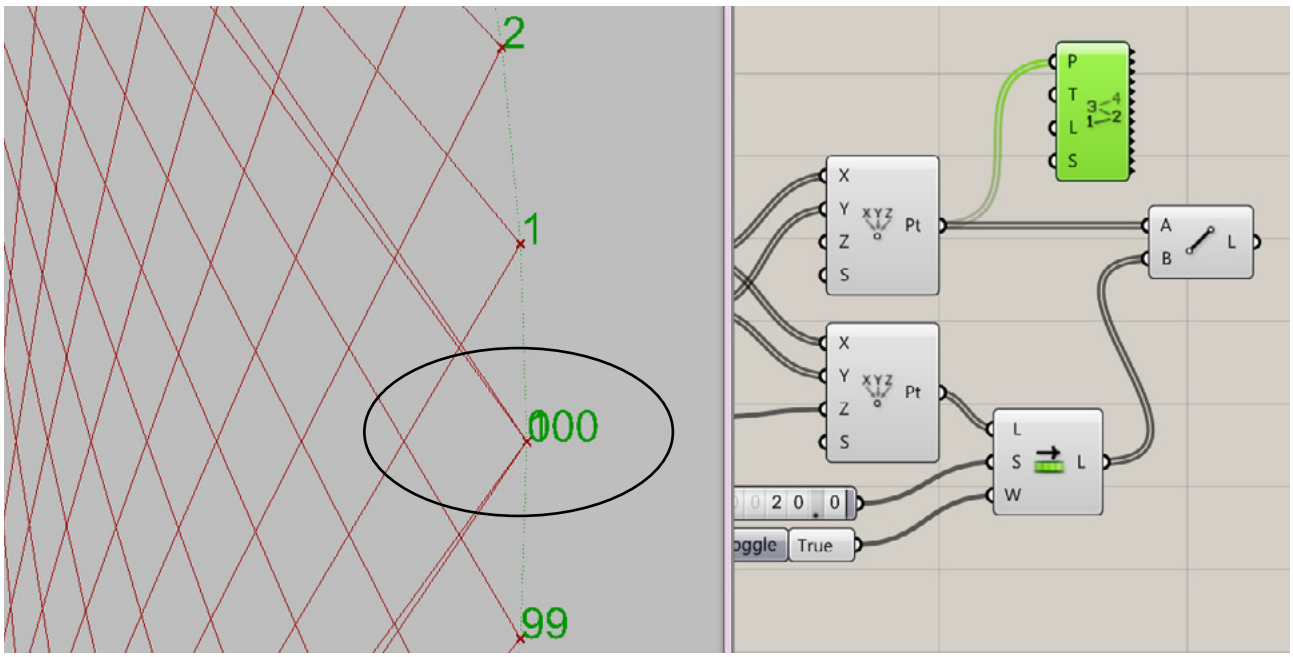


Fig.3.20. Even in a simple example like this there are issues to be solved. Looking closer to the geometry, we see that there are four lines coming to the first point instead of two. The reason is that while we generated points by numerical values, we have the first and the last points (0 and 2Pi) in the same position. So this little error in design (and not in the algorithm) appears. The first or the last point should be omitted in this case.

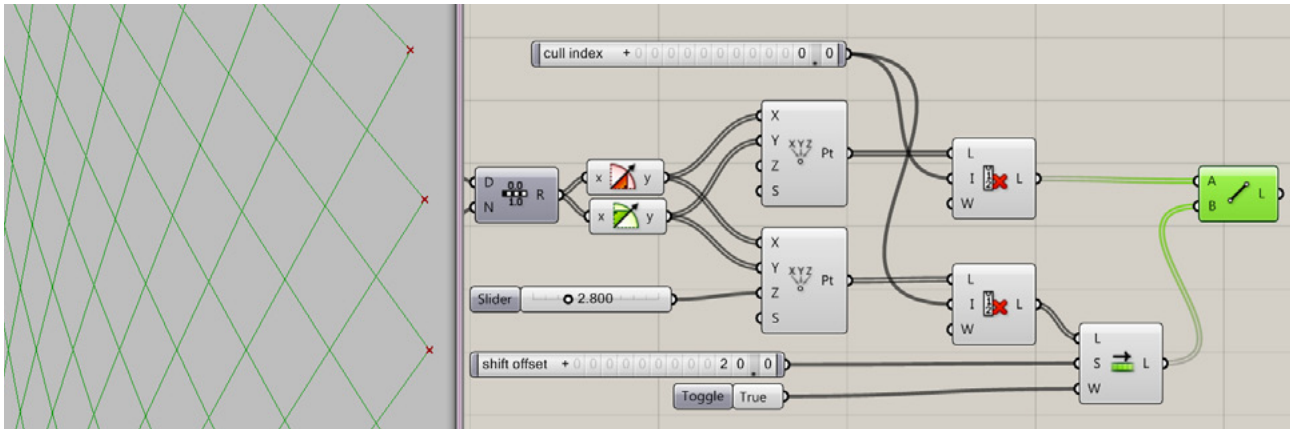


Fig.3.21. We might not know the index number of the last item to remove it (we need to extract it somehow) but we are always sure that the first item has the index number 0. Using a <Cull index> component by index=0, here the first point of both lists has been removed. Now if we follow the rest of the algorithm, we should have a ruled surface with no geometrical issues.

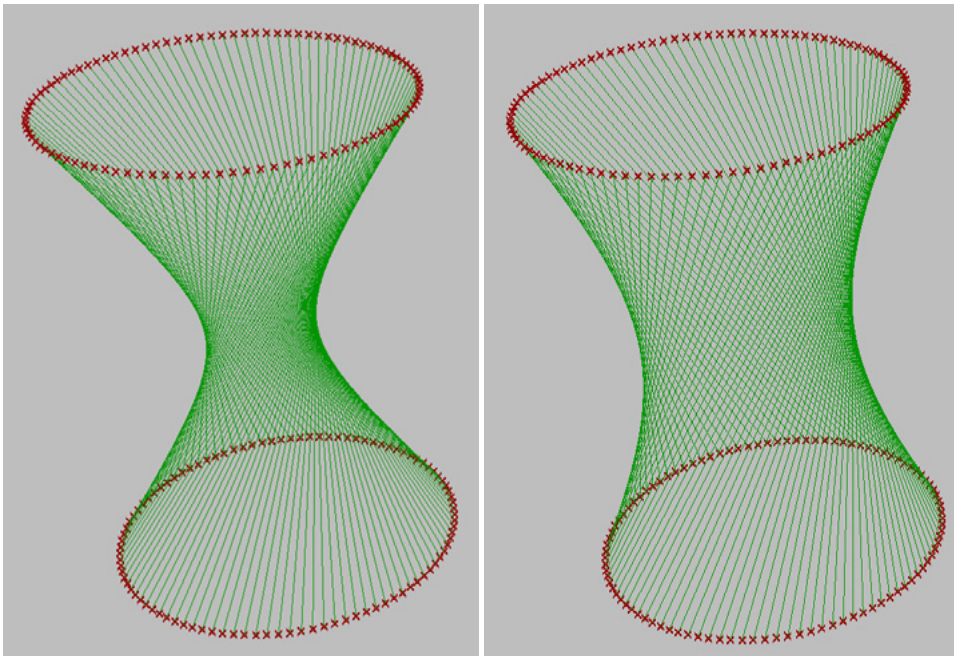


Fig.3.22. Two variations of the ruled surface which are achieved by the change of Shift Offset value of the <Shift>.

3_4_ On Tessellation and Tiling

Tiling, Tessellation and Geometric Patterns are all relate to the idea of covering. They usually encompass a process of subdividing a surface into small parts and then repeat a pattern, tile or template to cover it. In all cases, repetition of the shape should have no overlap and no gap in between. There are various types of tessellation based on different tiles, but there exists only three types of regular tessellation with regular triangles, squares and hexagons. There are also semi-regular and non-regular tessellations as well.

The history of architecture has a broad range of tiling examples which are interesting to study and use for contemporary design applications. Tessellation and Tiling are among the possible design issues with Generative Algorithms and in Grasshopper. It is possible to set up rules to generate a tile and apply these rules to the area which wanted to be tessellated, or there is a potential to design a motif and then proliferate it as a pattern in design space.

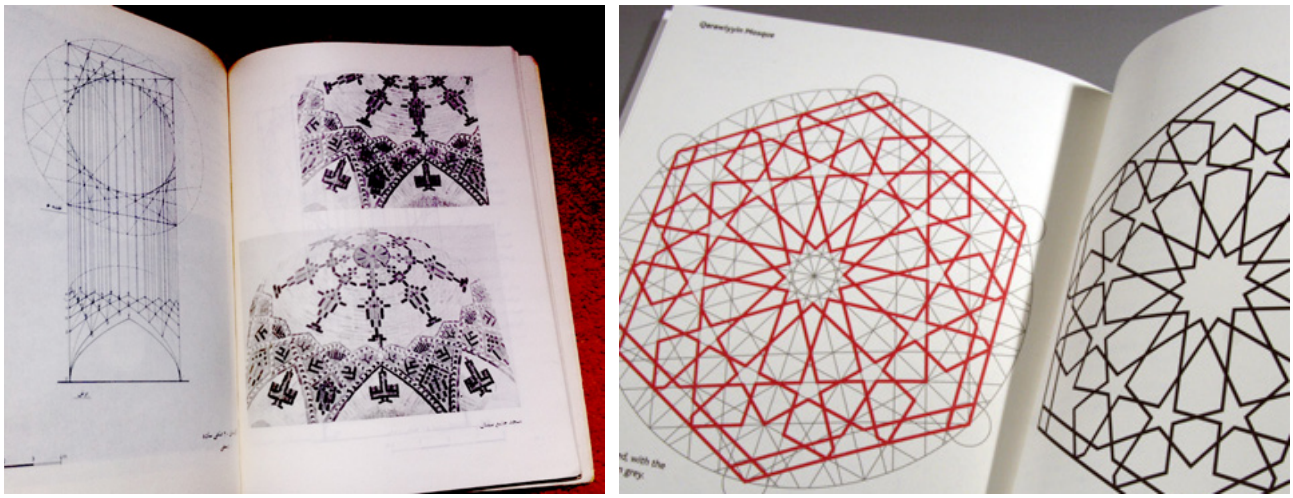


Fig.3.23. Geometrical studies and extracting the logic of a pattern by simple geometries.

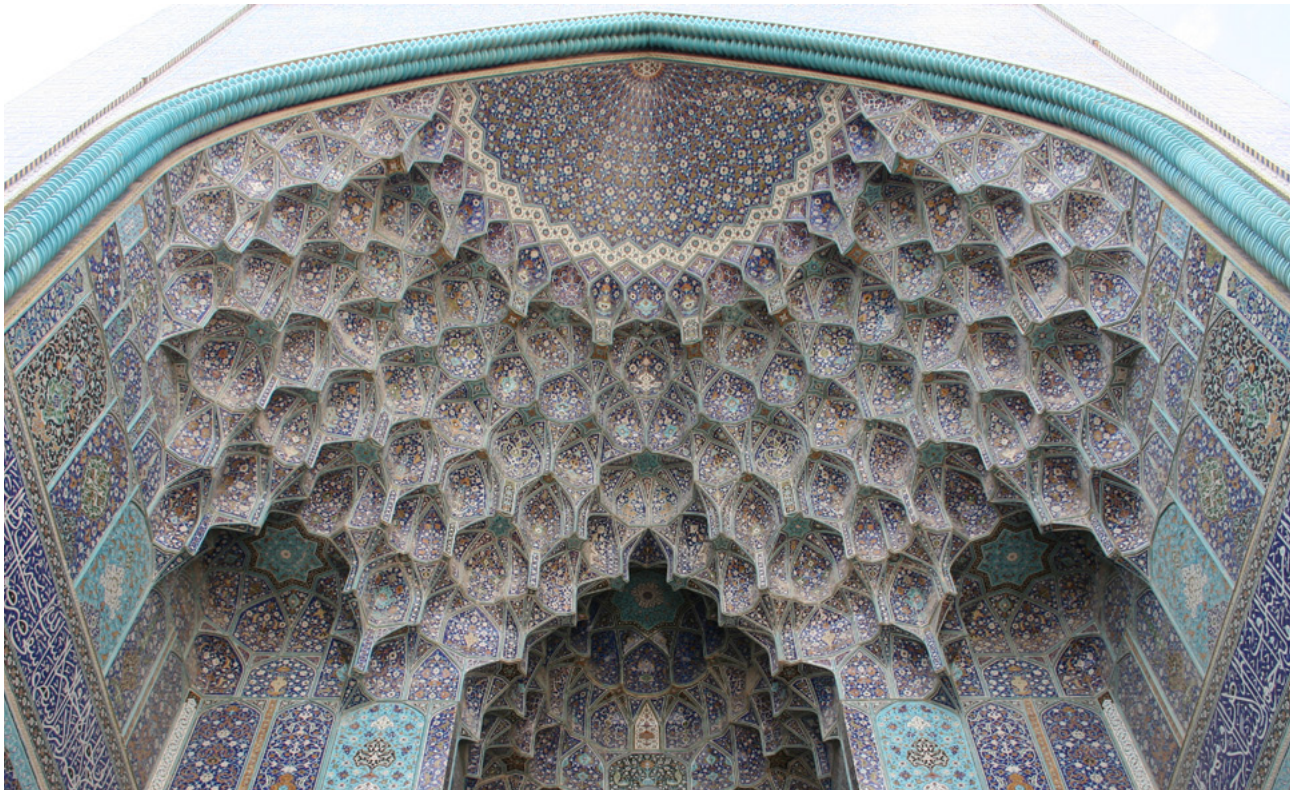


Fig.3.24. "Muqarnas", Complex geometries of Isfahan's Royal Mosque's Muqarnas and tile work comprises of patterns which are repeated in the space.

Fig.3.25. Dome of the Lotfollah mosque in Isfahan with complex tileworks under the dome and on walls as well. Using simple geometrical rules resulted in a a highly complex systems in the coverage of the dome. (Image source: Wikipedia)

There are various methods and approaches to design tiling. It is possible to use a motif and repeat it like fabric. Having a predefined pattern, one can use transformations to repeat it in the design space. The other way is to extract underlying geometrical rules of points, lines … and then set up the design, using those underlying rules as generative force of design. This option sounds interesting for this part of the book.

Sketching a pattern

Here there are some sketches of a simple repetitive pattern which I decided to generate with basic elements like points and lines. Basically the pattern is generated out of some lines which connect certain points in space. So by generating these points in a desired order, it is possible to connect them together by lines and generate base lines of tiling.

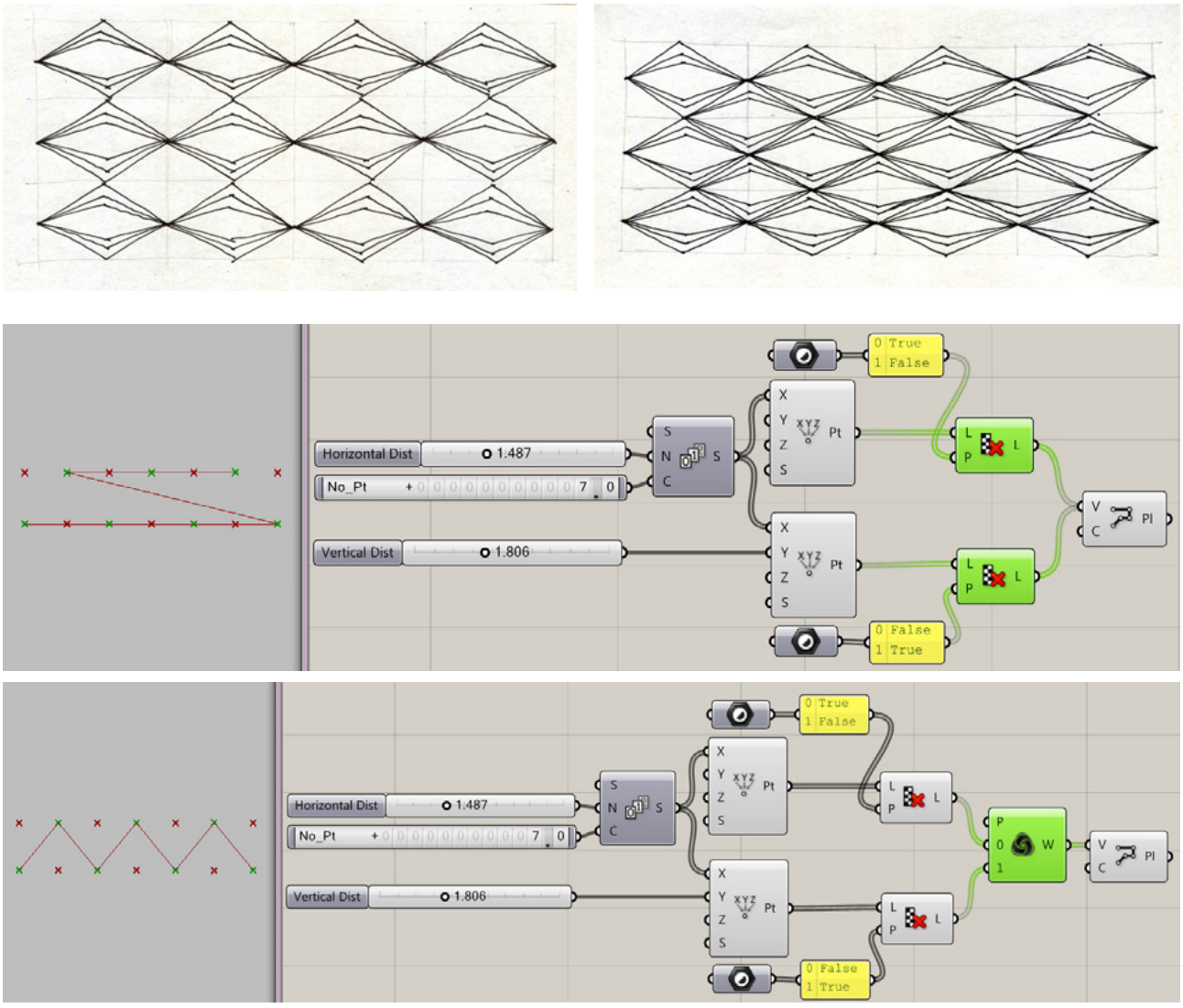


Fig.3.26. The idea is to generate points as base geometries and then add lines. I started my definition by a <series> which makes it possible to control the number of values (here number of points) and the step size (here Horizontal distance between points). To create the 'zigzag' form of the lines, I need two rows of points. I generated the second row of points here with Y values come from another <number slider> which makes it possible to control the vertical distance between points later on. In the next step, I have omitted every other point with <cull pattern> to have zigzag pattern in data. But if you connect both <Cull> components (by holding shift key) to a <Poly line> component from Curve > Spline, you will see that a Z shape line would be the result. This is because the order of points is from the first list and then the second list. But they have to be sorted like this: 1st_pt of 1st row, 1st _pt of 2nd row, 2nd_pt of 1st row, 2nd_pt of 2nd row, …. The component that sorts points in a way which I described is <Weave> (Logic > List). It takes data from multiple resources and sorts them based on a pattern which should be defined in its P port (True/False). The result is a list of sorted data and when you connect it to a <Pline> you will see that the first zigzag line is generated.

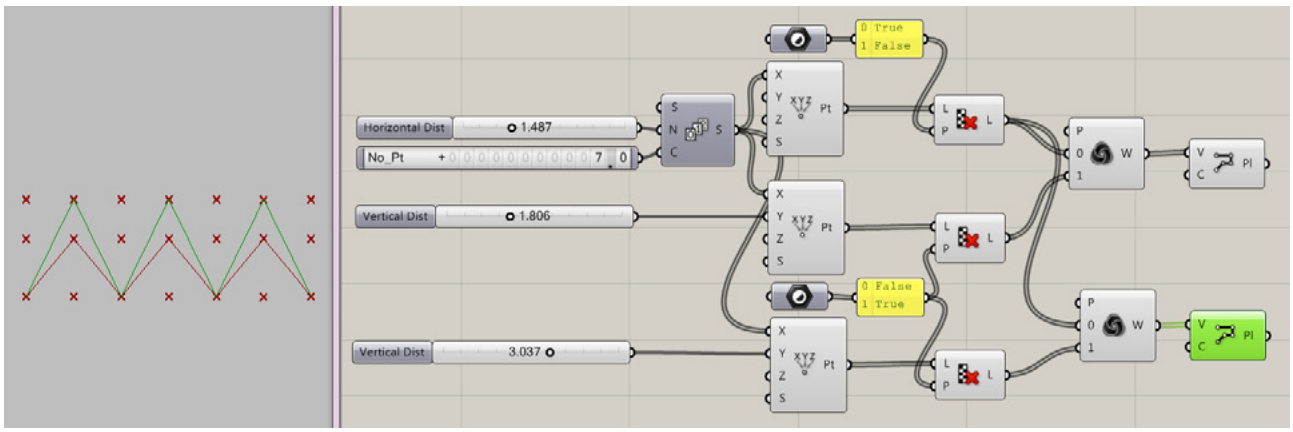
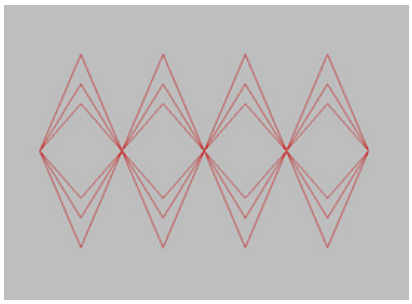


Fig.3.27. Although there are better ways, since they need higher levels of data manipulation, Here with the same concept, I generated the third row of points, and with another <weave> and <Pline> components, I have drawn second zigzag line of the pattern. The process is the same for the rest of the algorithm.



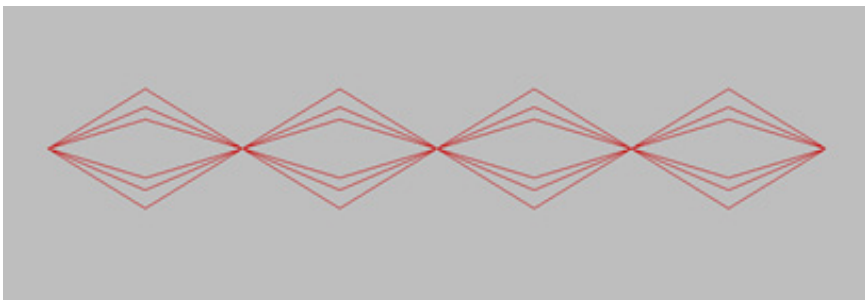


Fig.3.28. There are various ways to control and change the motif and then the pattern. You can change the way you generate your base points or cull your lists of data. The result could be different patterns of intersecting lines which could be the generative geometry to produce complex models. Here the above sketches are converted into a generative model and this tiling has been used to develop a 3D tiling example with extrusion.

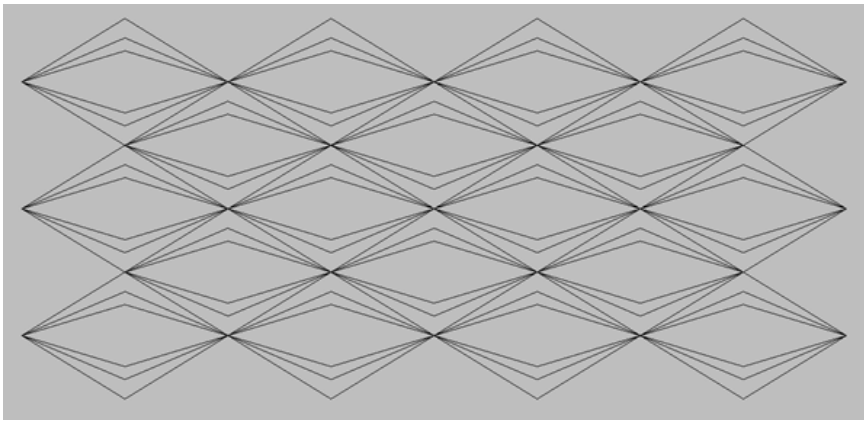
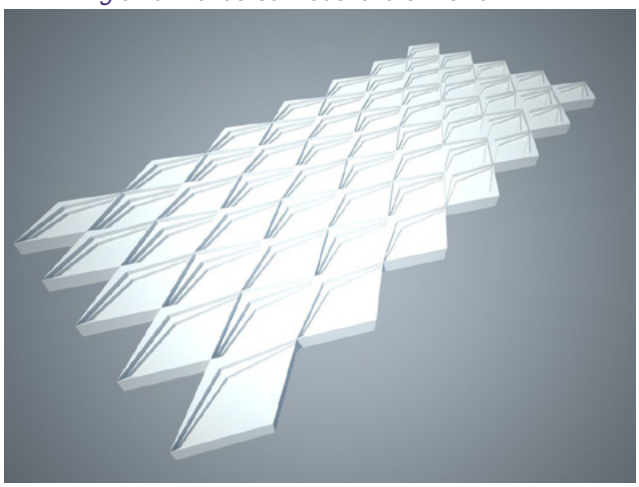
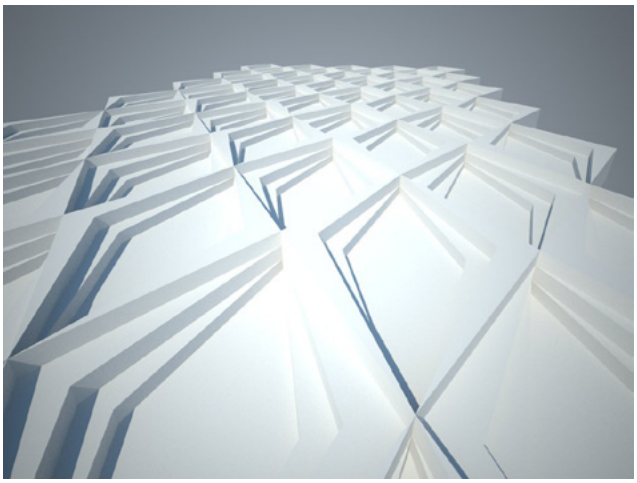


Fig.3.29. Rendered Model of the Tilework.





Tessellation by a repetitive Motif

While in the previous example the basic rules for generating a pattern has been developed, here the idea is to set up a motif and then repeat it in the design field. The motif by itself is simple and achievable through very basic rules we have worked with so far.

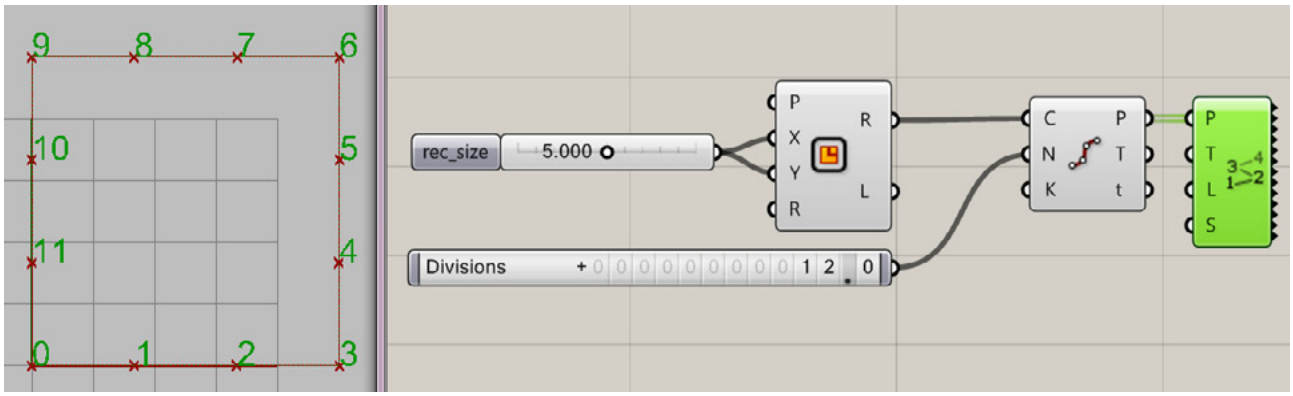


Fig.3.30. The very first step is to define the boundary of the motif. It is like a cell. The <Rectangle> component is used to generate this cell (Curve>Primitive). This rectangle has been divided by a <Divide Curve> to generate some points across its boundary (Curve>Division).

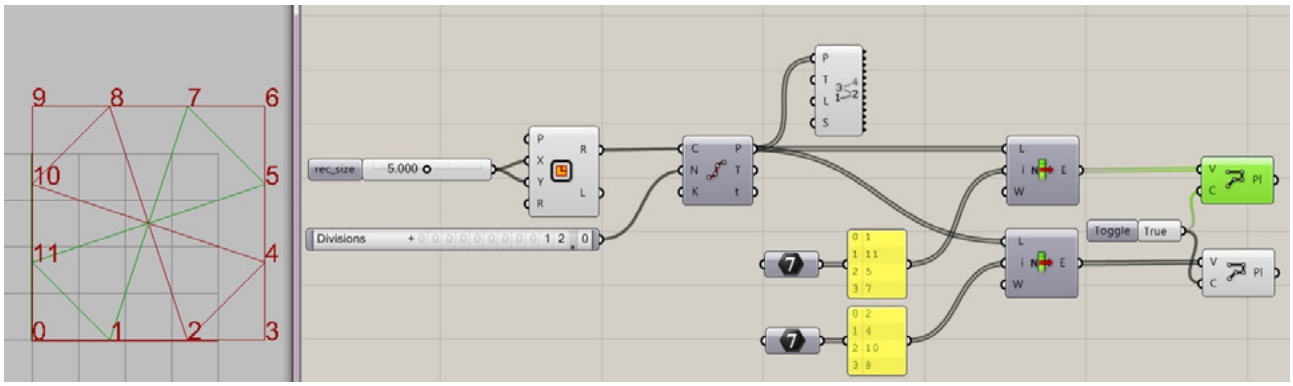


Fig.3.31. The idea is to select points on the rectangle by a predefined order and then connect them together by <Polyline> to generate the basic shape of the motif.

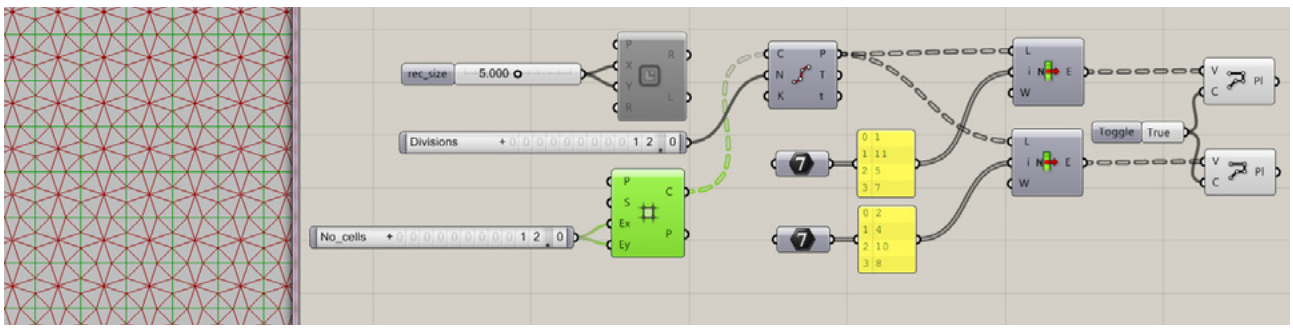


Fig.3.32. Since the motif is generated based on a rectangular cell, it should be possible to change this cell and get the same result. Here a <Grid Square> from Vector>Grids has been used to generate large number of cells instead of one and all these cells are substituted with the previous <Rectangle>. You see that the motif is populated in all cells. Great! This is generative!

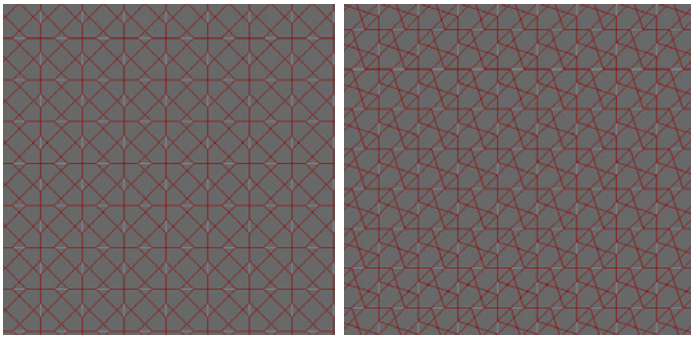


Fig.3.33. Before finalizing the tessellation algorithm, you can start to play around the motif to see what different results you can get. This could happen by changing the numerical order of the indices used to select points.

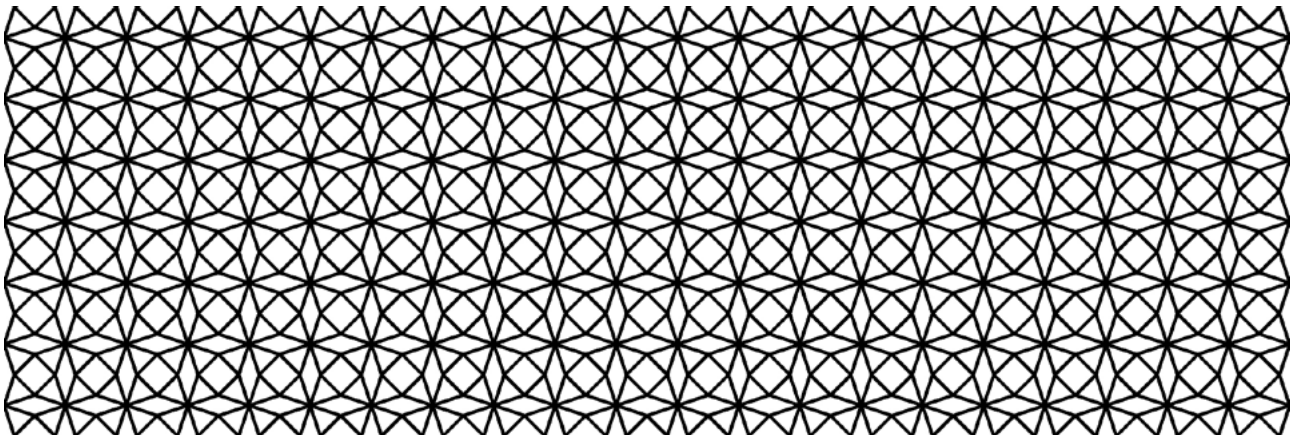
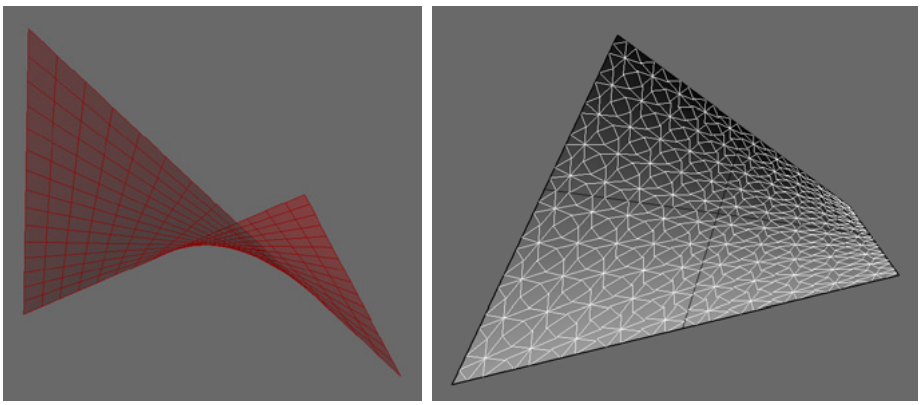
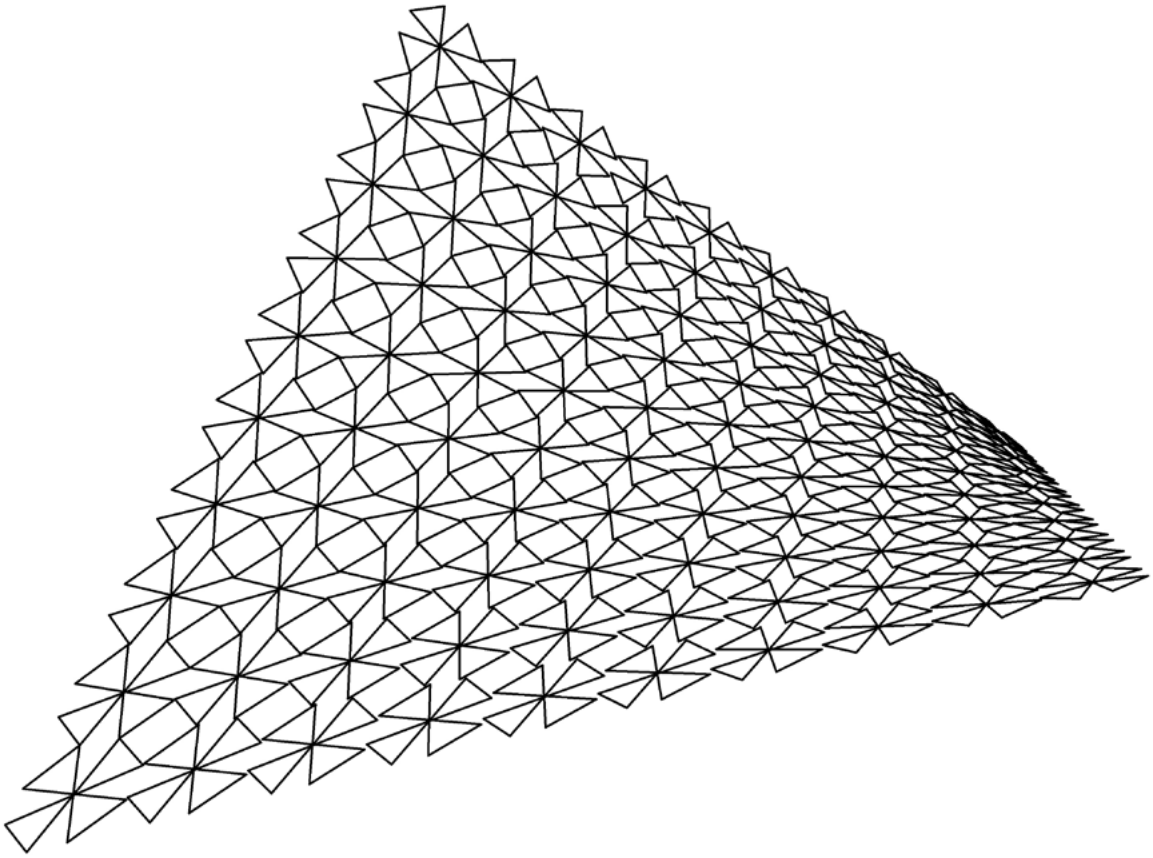


Fig.3.34. Tessellation by a repetitive motif

Fig.3.35. Further in the design process, one can assume that any 3D rectangular net could be a place to populate this motif and get various results out of the same process.





Chapter Four

Transformation

Chapter Four
Transformation

While the first step in algorithmic design is 'Generate Geometry', the next step is to modify those generated or already existing ones and approach the design by modifying objects. Transformations are essential operations to do so. They enable us to get variations from the initial simple objects, help us to (re-) scale and orientate objects, move (translate), copy, rotate and mirror them, or may help in accumulation of objects or other complex operations. There are different types of transformations like Planar Transformation, Spatial, Affine, Projective transformations and so on. While Linear Transformations map straight lines into straight lines, Non-Linear Transformations map straight lines into curves. Also when Planar Transformations happen in a plane, Spatial Transformations are happening in 3D space. In terms of the shape change, transformations like translation, rotation, and reflection keep the original shape but scale and shear change the original state. In addition to translation, rotation and reflection there are different types of shear and non-uniform scale transformations in 3D space, also spiral and helical transformations and projections which make more variations in 3D environments.

In order to transform objects, conceptually we need to move and orientate objects or part of them like their vertices or cage corners in the space. Transformations are become possible by using vectors and planes as basics of these mathematical/geometrical operations. We are not going to discuss geometrical rules of transformations and their mathematical logic here, but first let's have a look at vectors and planes because we need them for our work.

Fig.4.1. Transformation is a great potential to generate complex forms from simple objects. Nature has some of the greatest examples of transformation in its creatures.

4_1_Vectors and Planes

Vector is a mathematical/geometrical entity that has direction, magnitude (or length) and sense. It starts from a point, goes towards another point with certain length and specific straight direction. Vectors have wide usage in different fields like geometry and transformations.

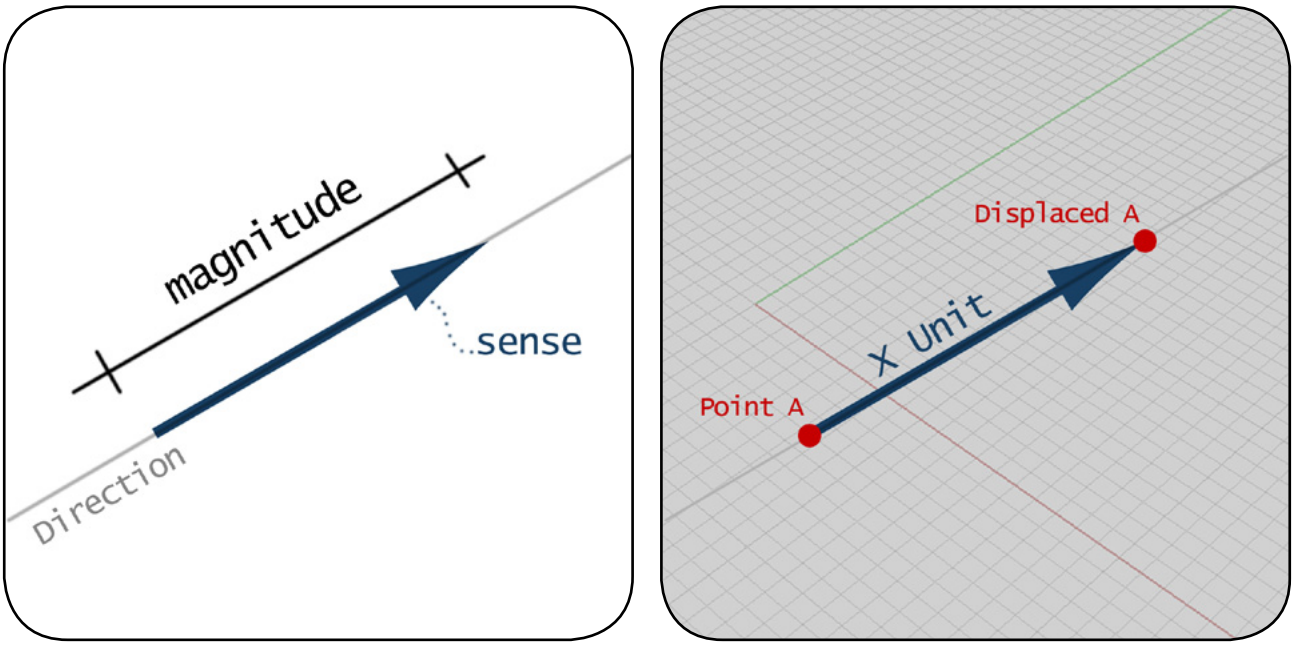


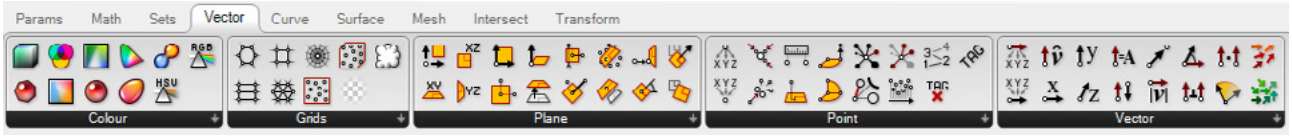
Fig.4.2. A: Basic elements of a Vector, B: point displacement with a vector.

Simply if we have a point and a vector, this vector can displace the point with the distance of vector's magnitude and towards its direction to create a new position for it. We use this simple concept to generate new objects or to move, scale, project, mirror and do other transformations in 3D environment.

Planes are another useful set of geometrical entities that we can describe them as infinite flat surfaces which have origin points. For example Construction Planes in Rhino (CPlane) are planes. We can use these planes to put our geometries on them and do some transformations based on their orientation and origin. For example in 3D space, we cannot orientate an abject on a vector! we need a plane to be able to put geometry on it.

Vectors have direction and magnitude but planes have orientation and origin. So they are two different types of constructs that can help us to create, modify, transform and articulate objects in space.

Grasshopper has some of the basic vectors and planes as predefined components. These are including X, Y and Z unit vectors and XY, XZ, and YZ planes. There are couple of other components to produce and modify them which we will talk about them in our experiments.



4_2_Curves

Curves as 1-Dimentional geometries have multiple functions in design. There are different types of curves which include simple Line, Polyline, Arc, Poly-Arc, NURBS or Bezier curves, Circles, Rectangles, Polygons and…. Like points, curves can be used as the base geometry for construction of so many different objects. A curve can be extruded along another curve to make a surface, connect different curves together and make surfaces and solids, or can be used to distribute other objects along its internal positions.

The Birth of a Star!

To do some experiments with curves and vectors, here the idea is to generate some lines which start from points of a grid, but end up at the one external point. We can assume bunch of converging lines would appear.

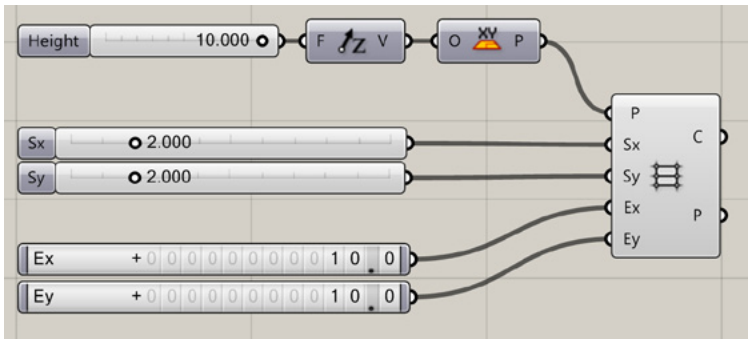


Fig.4.3. Using a <Grid rectangular> (Vector > Grids) which produces a grid of cells and also points, we can control the number of cells/points in X and Y direction and the distance between points (cell sizes) in this component.

By introducing an <XY plane> component (Vector > Plane) it becomes possible to modify the position of the grid, based on its plane. Here I displaced it in Z direction by a <Unit Z> component (Vector > Vector).

If I introduce an external point and connect it to the points of the gridwith a <line>, I would have bunch of lines with different length. But how we can draw these lines with the same length? We need to change the way we draw lines. A <line SDL> component (Curve > Primitive) draws a line by Start point, Direction, and Length of the line. So the length of line is controllable in this component. Here I have start points from the grid, and I want to define the length of lines manually. What about the direction? The direction of lines should be towards the external point. To define the direction, here I need bunch of vectors, each corresponding to a line to define its direction in 3D space. Having start and the end points of lines, we can define vectors by these two points as well, using a <Vector 2pt>.

All these lines can be used for generation of some pyramids to continue our experiments. To generate these pyramids, we need to add a polygon at the end of each line and extrude it towards the end point of the line. Polygons need planes to mount on. These planes should be at the start of lines and perpendicular to their direction.

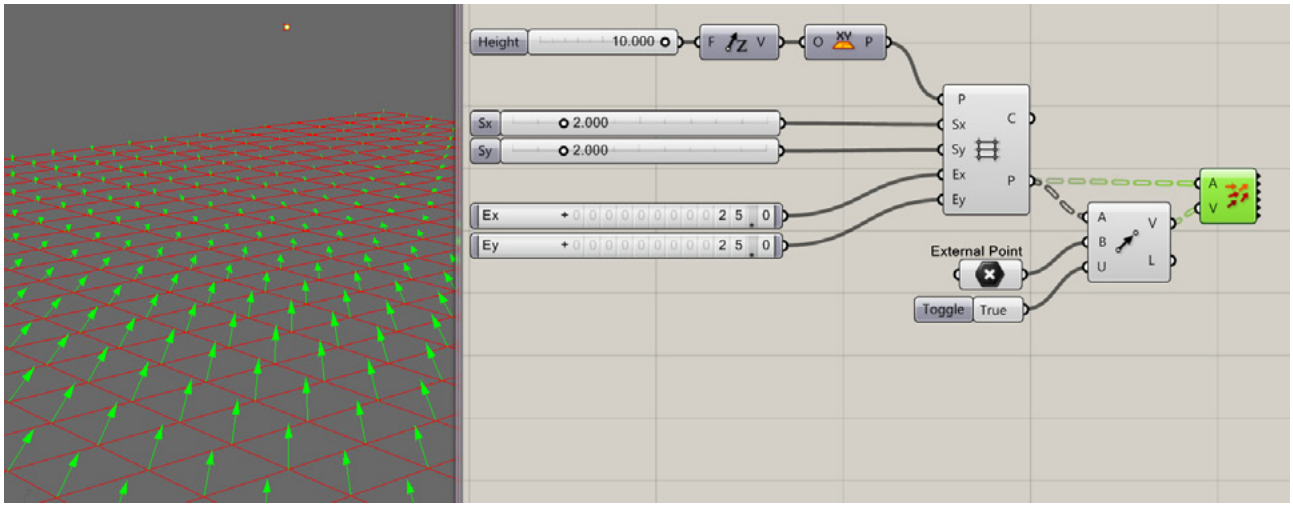


Fig.4.4. Creating vectors from the start point toward the grid points by <Vector 2pt> component (Vector > Vector). We can use a <Vector Display> to visualize generated vectors in viewport.

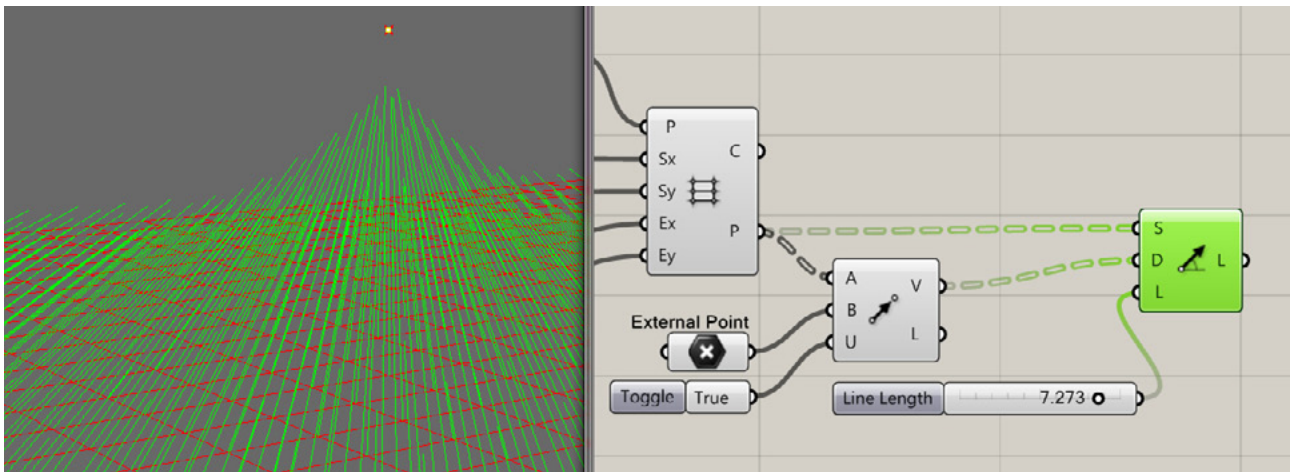


Fig.4.5. The <line SDL> component generates bunch of lines from the grid point towards the external point that seems spread out into space. Changing the position of external point or properties of the grid would result in the quality of this spreading.

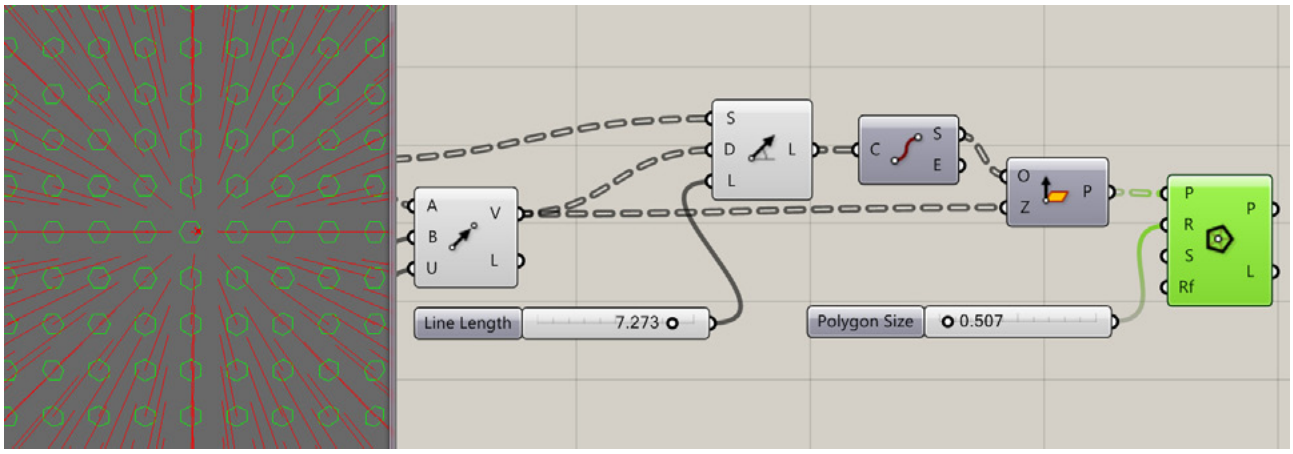


Fig.4.6. By using an <end points> component (Curve > Analysis) and using 'start points' as 'origin points' for a set of planes I could generate my base planes. Here I used <Plane Normal> component (Vector> Plane) which produces a plane by an origin point and a Z direction vector for the plane. Same vectors of line directions have been used as normal vectors. With a <Polygon> component we would have a set of polygons at the start point of each line and perpendicular to it.

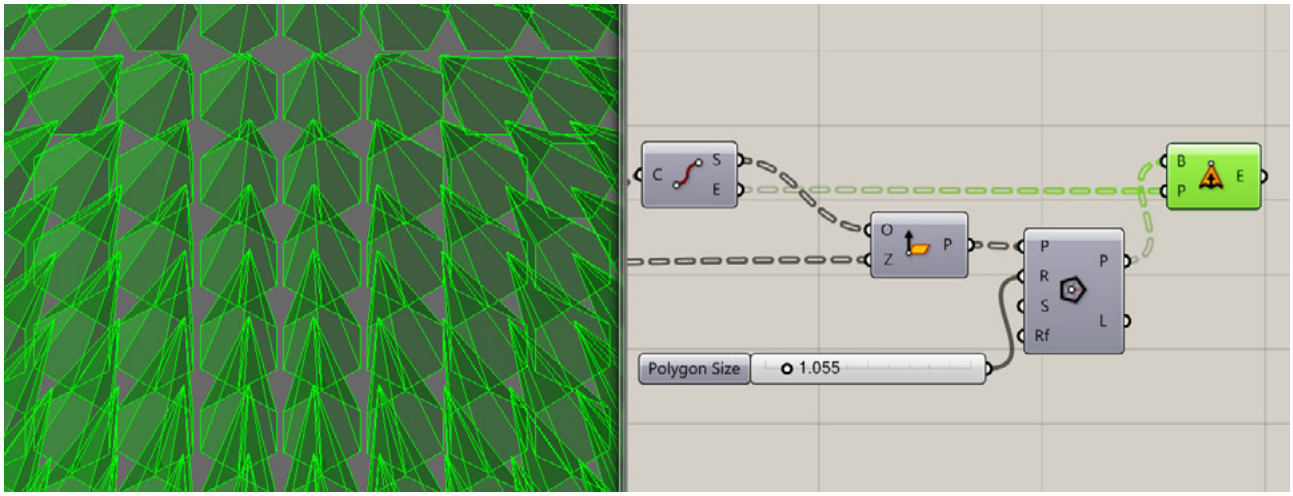


Fig.4.7. In the last step, I used an <Extrude Point> component (Surface>Freeform) and I attached lines' end points as the points towards which I wanted my polygons to extrude.

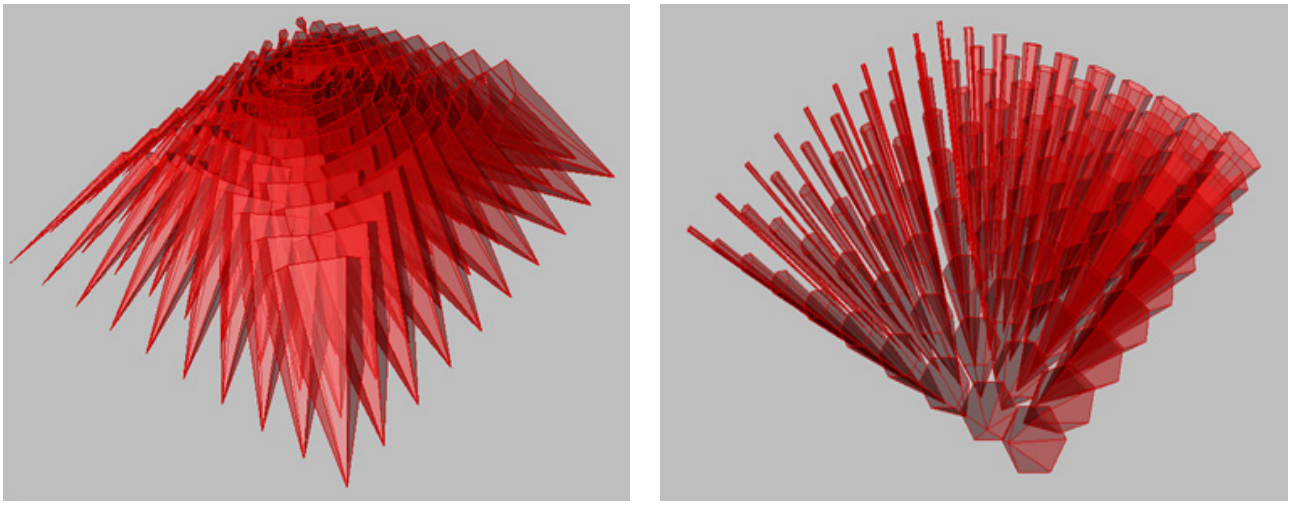


Fig.4.8. It is possible to change the size of polygons gradually using series of numbers instead of <number slider>. It is also possible to change the limit of line length (<Line SDL>) from –x to x so you can change the direction of the lines and also pyramids with negative values. Finally you can bake and render a star shape geometry after these simple experimentations with curves and vectors.

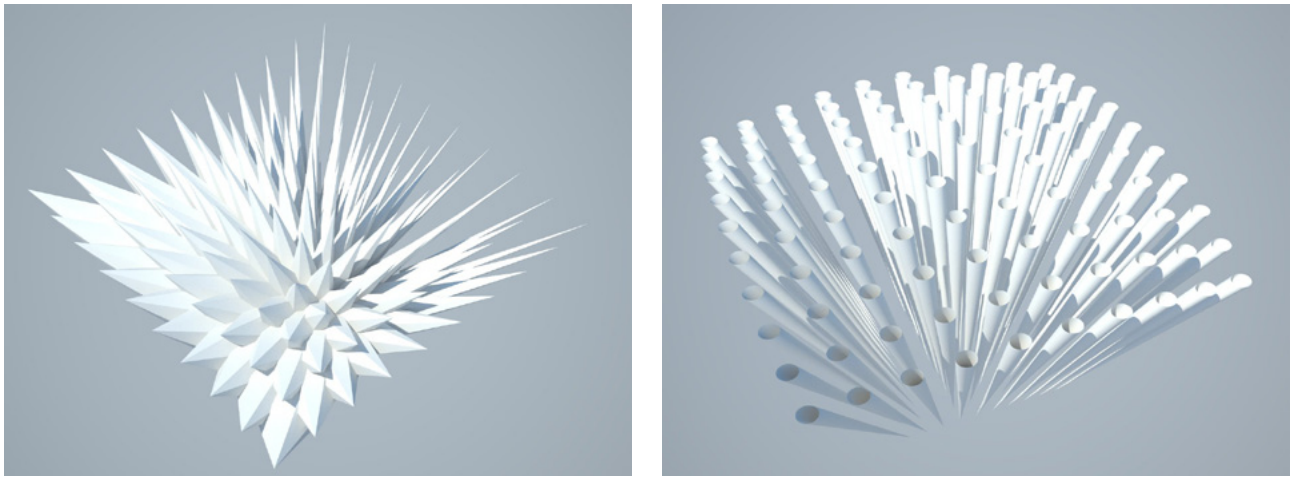
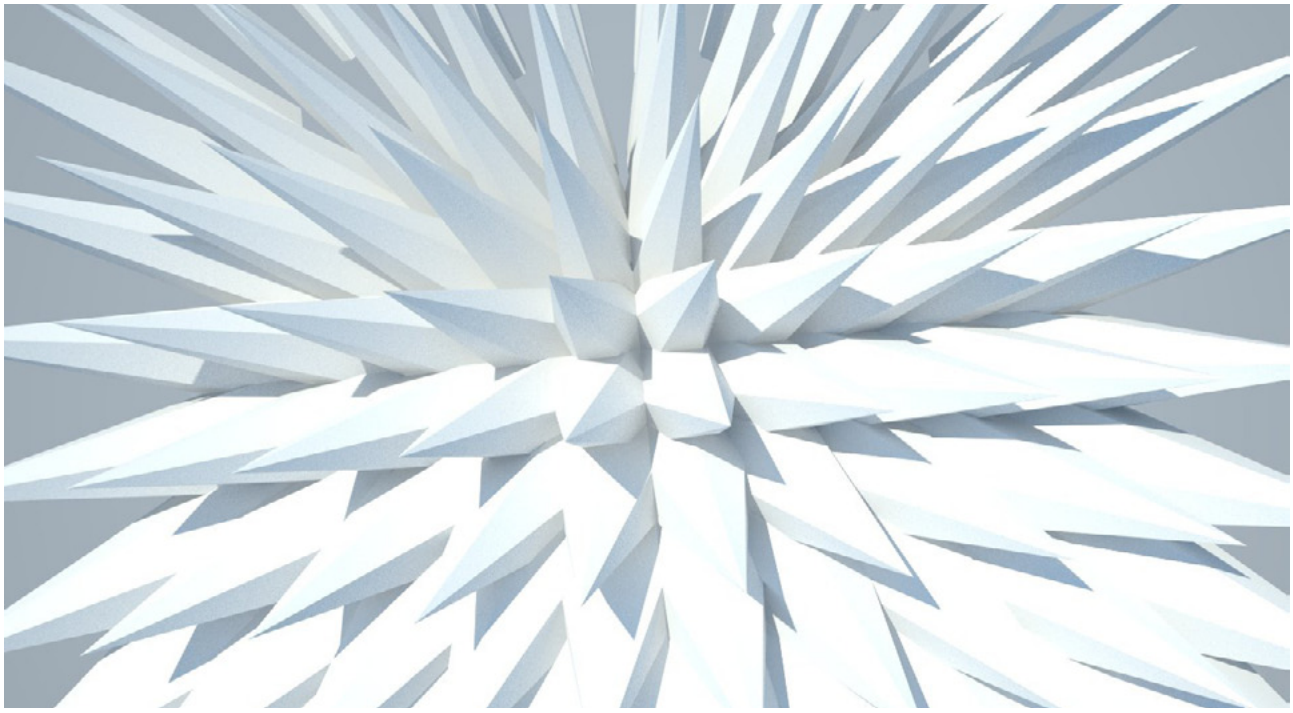


Fig.4.9. Final model (Rendered with Polygon and also circle as the profle curve for extrusion.



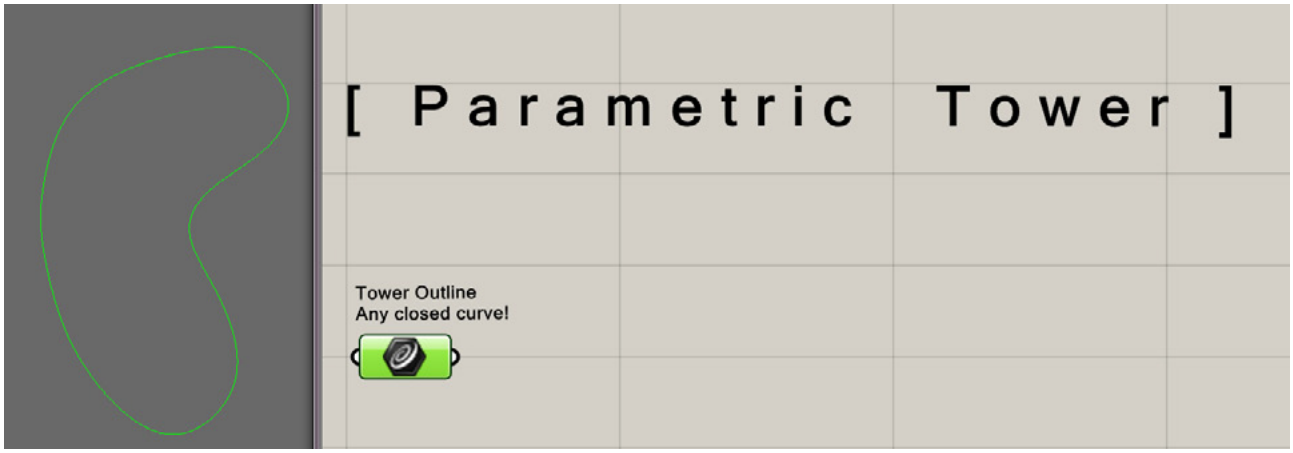
4_3_On Parametric Towers

One of the features of algorithmic design is its potential to sketch ideas relatively fast. By understanding some concepts of geometry and algorithm procedures it becomes easy to visualize early design ideas and sketches with algorithms, create variations of design and push one further for design development. One of these subjects which have been experimented a lot is Parametric Towers. Since a tower literally represents a stack of floors on top of each other, it is possible to follow this idea, with various modifying features in algorithmic design. The following example tries to cover this subject with transformation operations.

_Parametric Tower

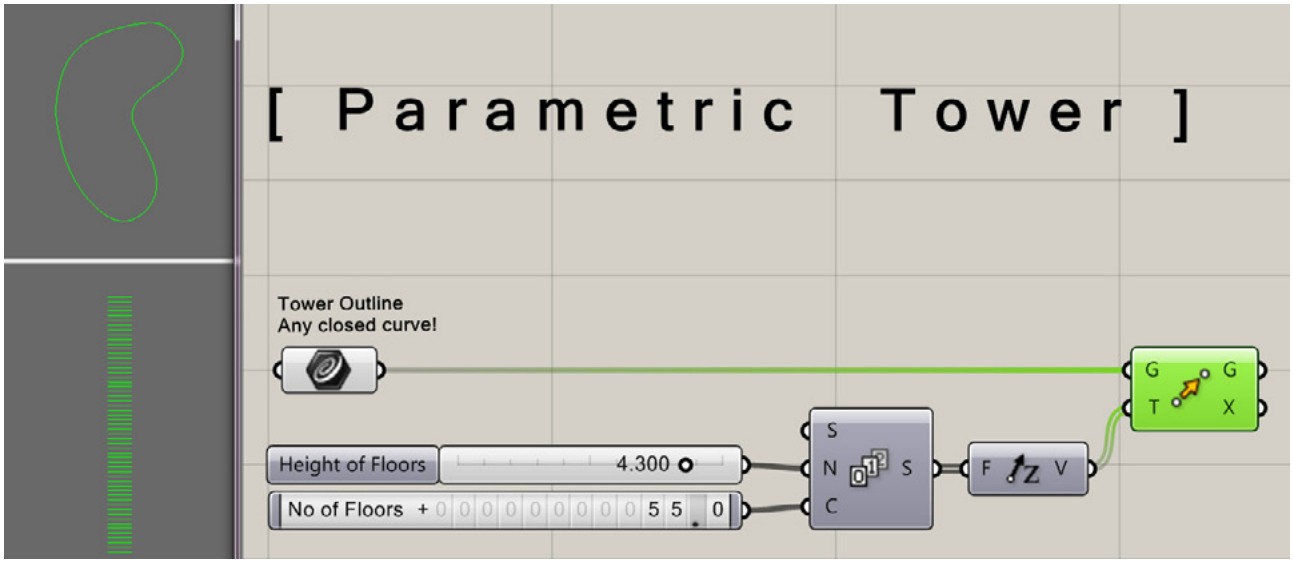
_Tower outline

Although Mies's modernist towers have rectangular outline, it is now (technologically, structurally, systematically …) possible to design a tower with different forms and outlines. This outline represents the periphery of the floors which wanted to be copied above itself in order to generate floors of the tower. Any type of closed curve could be the tower outline.



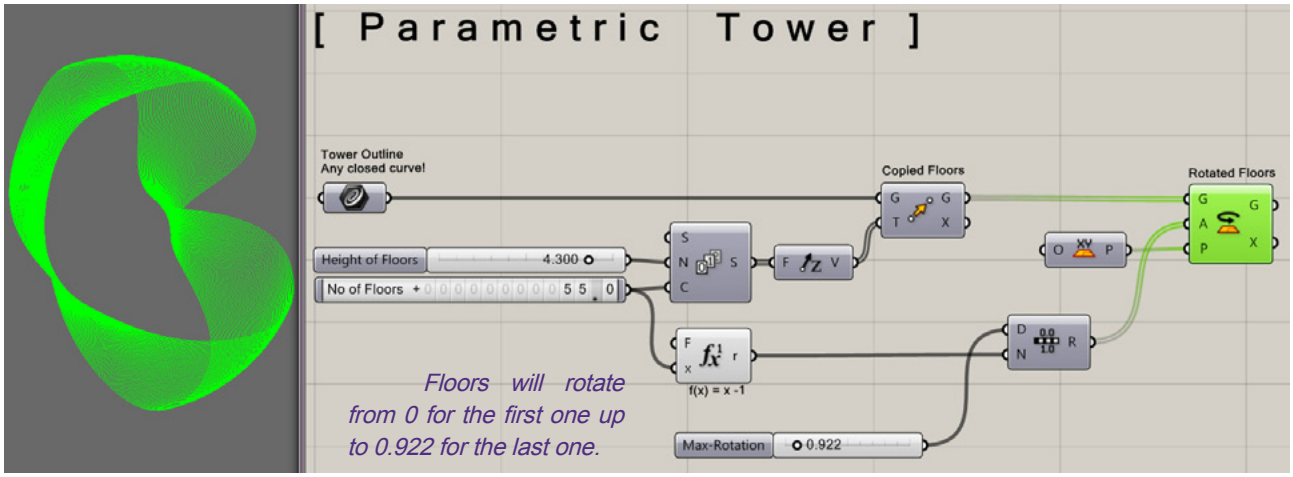
_Stacking Floors

A <Move> component (Transform > Euclidean) can translate an object in space with a vector. it can move (copy) object in various positions in space by using couple of vectors. So using a <Move> component with various Z vectors would result in the stacking of floors on top of each other.



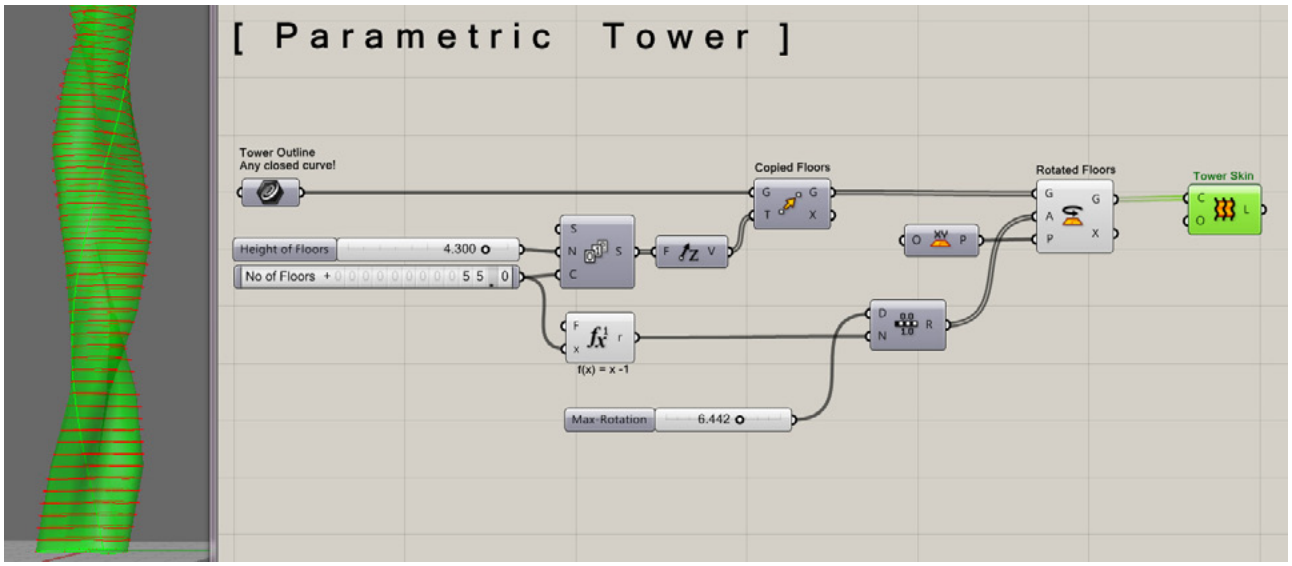
_Rotating

A <Rotate> component rotates an object around the center of a plane. Floors could be the target of rotation as well. It is possible to rotate floors gradually. In order to define the angel of rotation for all floors, there should be one numerical value for each, so the number of rotation angles comes from the number of floors (but -1 since <Range> provides one extra number).



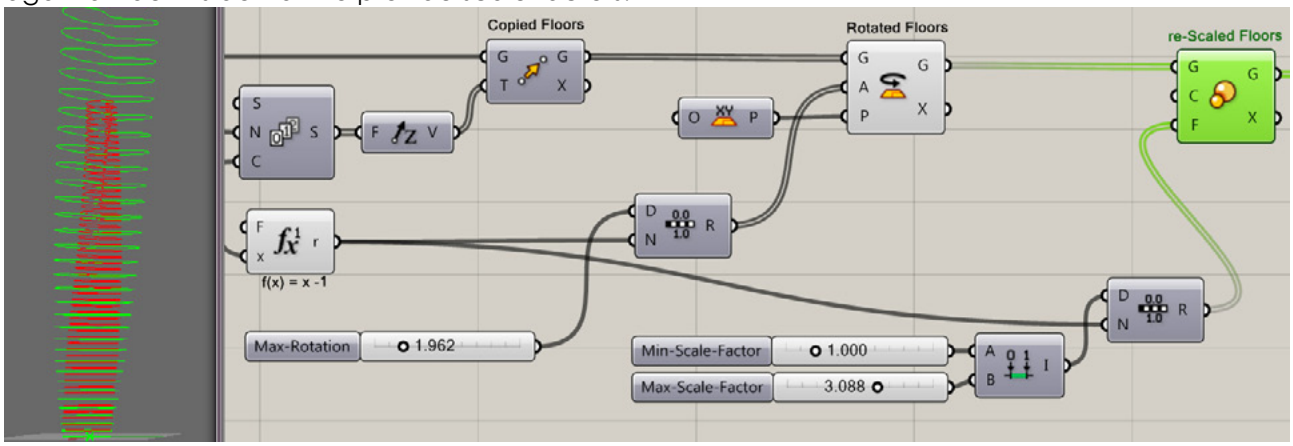
_Skin

Lofting all copied and rotated floors would generate the skin of the tower for better preview.

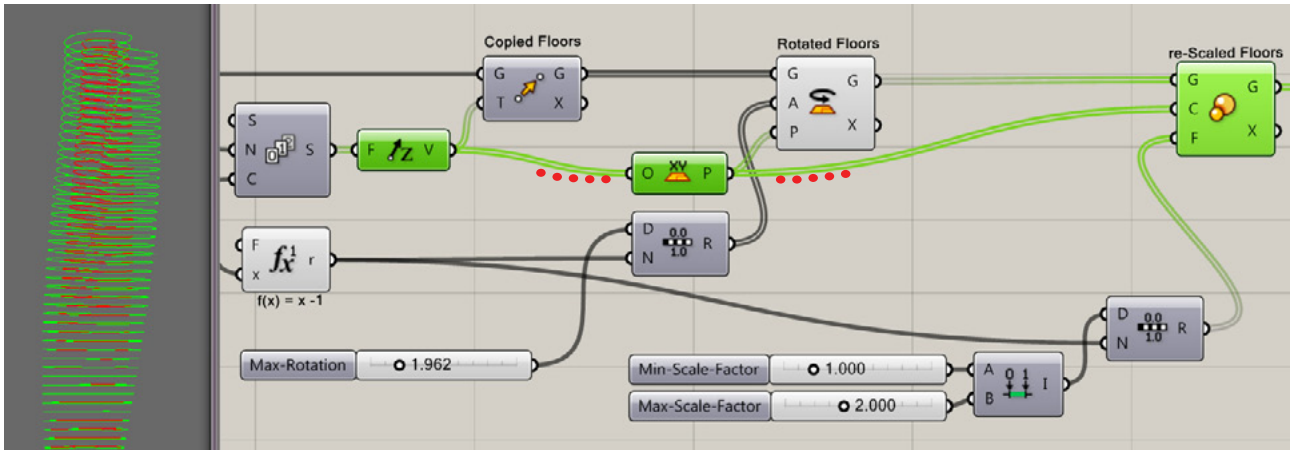


_(re)-Scaling

It would be interesting as well to change the size of floors gradually by using a <Scale> (Transform > Affine). It is possible to define a one dimensional domain to set the min/max of scaling values and again divide this domain to provide scale factors.

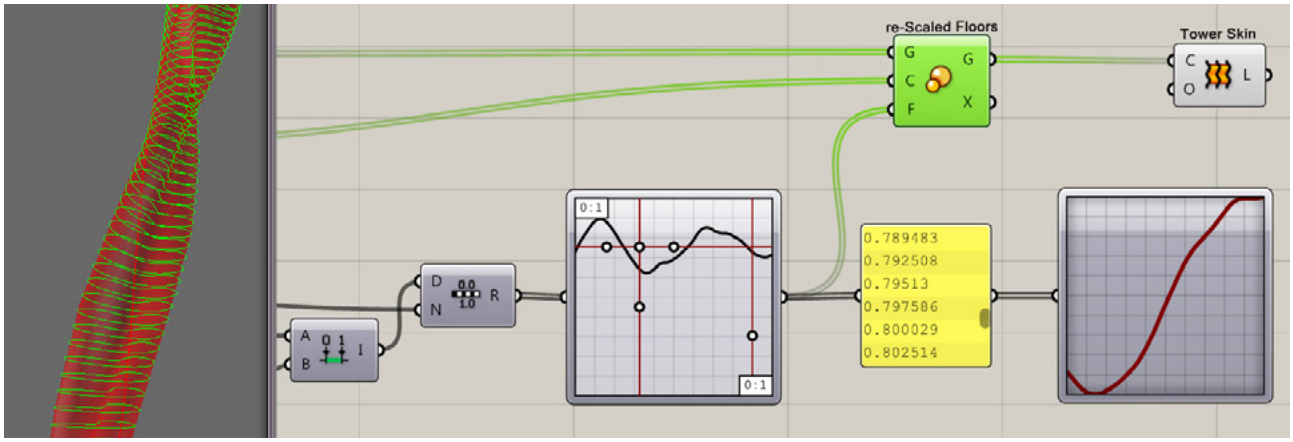


The problem with scale is that it changes the size of tower because it scales each floor by using a centre of scaling which is the base plane's origin here. To retain the size of tower, there should be one plane at the level of each floor to resize it in a constant position.



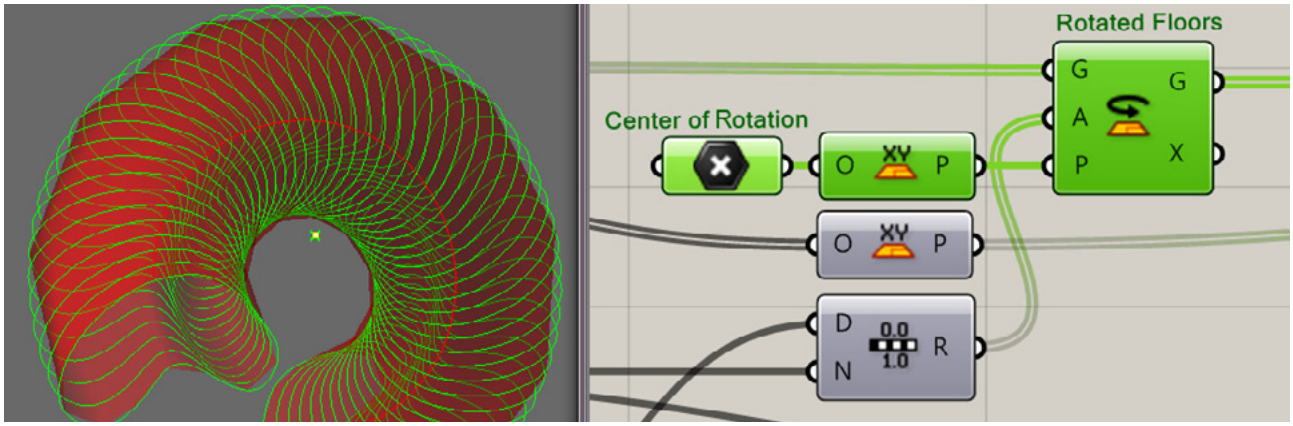
_Re-(re)-Scaling

Although it is possible to change the scale of floors from small to big, one might like to change the scale not in this linear fashion. It might be desirable to grow bigger in the middle of the tower and then become smaller again. Or other types of possibilities for re-scaling. Here a <Graph Mapper> introduced between scale factors and <scale> (Params > Special). A <Graph Mapper> can redistribute numerical values based on a math function, so all scale factors could be changed based on various graph types available in the component.

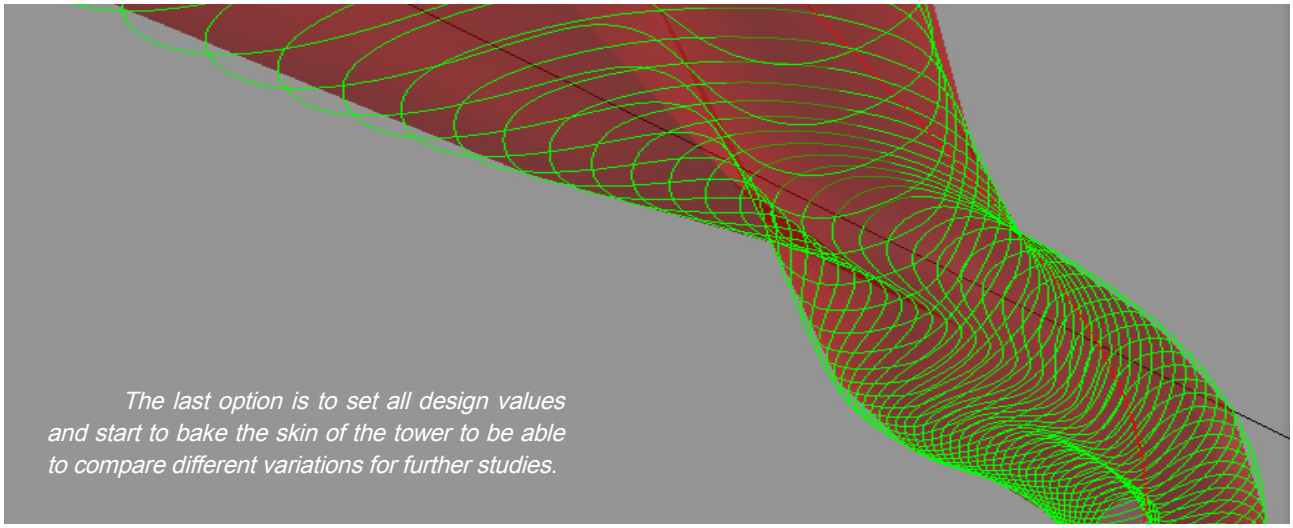


_Rotation Axis

Although the outline curve was drawn around the Origin and the rotation happened around the Z axis of the world XY plane, but it is possible to change the axis of this rotation and rotate floors around other axis to get deviated floors from the central axis of the outline.



_Dancing, Moving, Turning ... Tower!



The last option is to set all design values and start to bake the skin of the tower to be able to compare different variations for further studies.

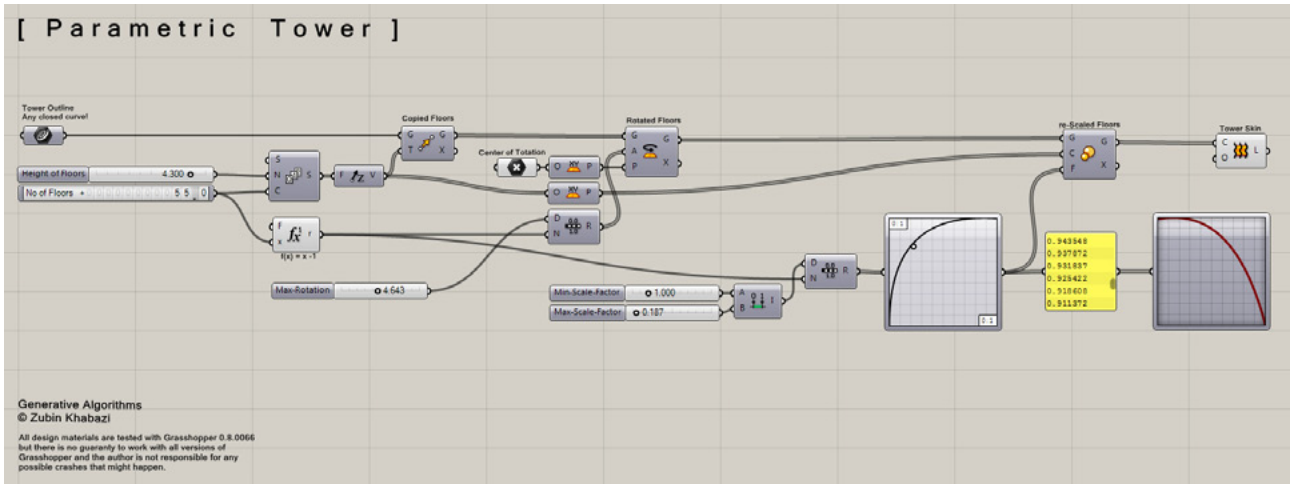
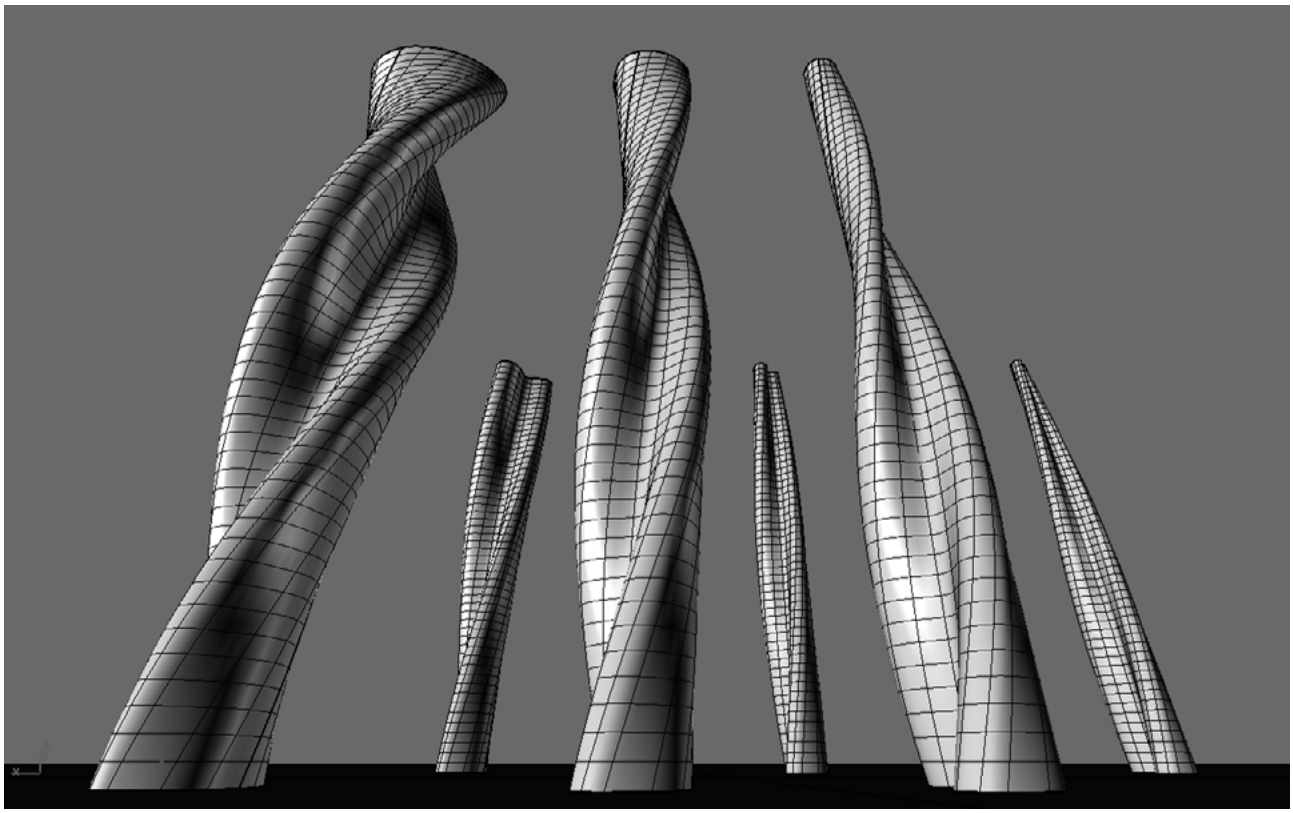


Fig.4.10. Parametric Tower Algorithm



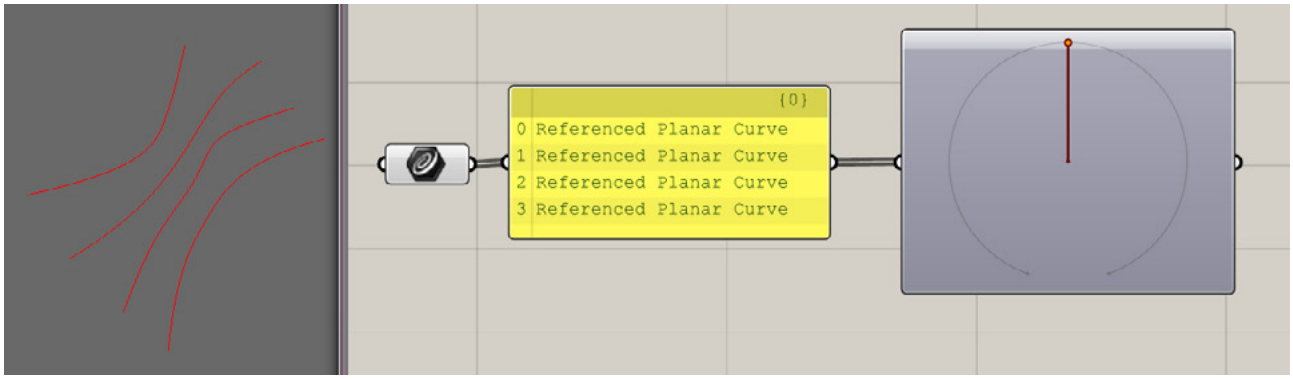
4_4_Data Manipulation 2 _ Data Trees

Data is structured with 'Branches' in Grasshopper and data branches together make 'Data Tree'. So far we managed to work with data lists. A data list is a series of objects in an order. These include various data types like numbers, points or curves. In the concept of data trees, each data list could be a branch, so a data tree is a bunch of data lists.

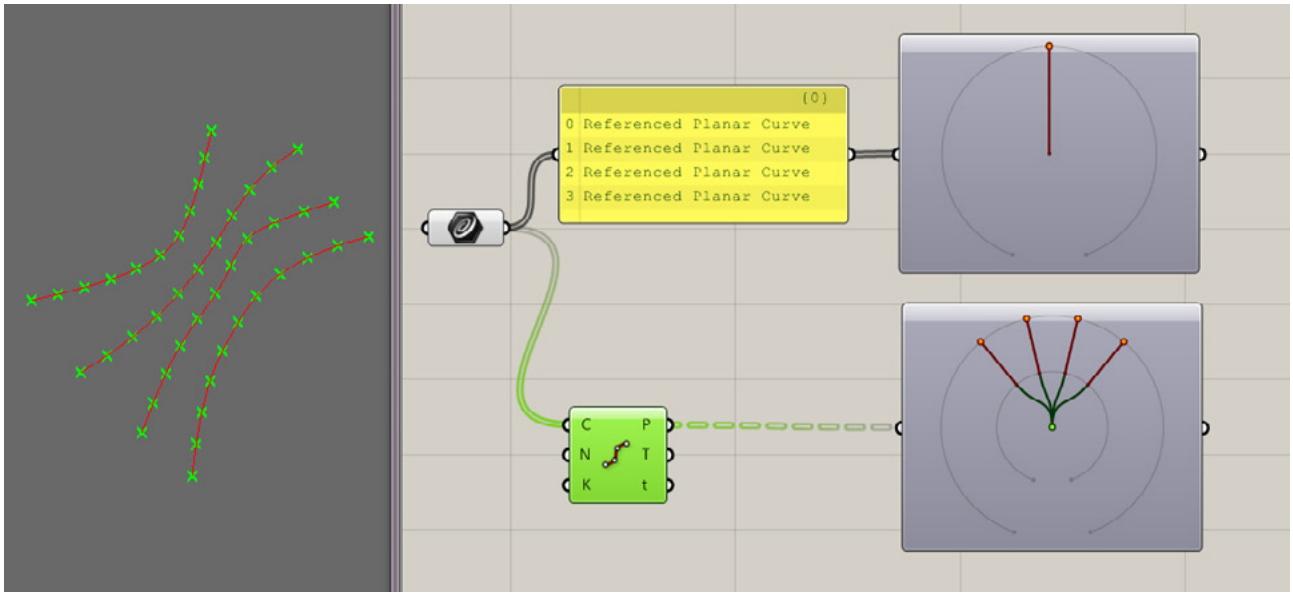
Some components provide data in data trees, some of them need to be fed by data in separate branches and some of them need data without branching. Designing with Grasshopper needs this level of data manipulation, to get desired outputs out of complex data/design situations.

Data Tree

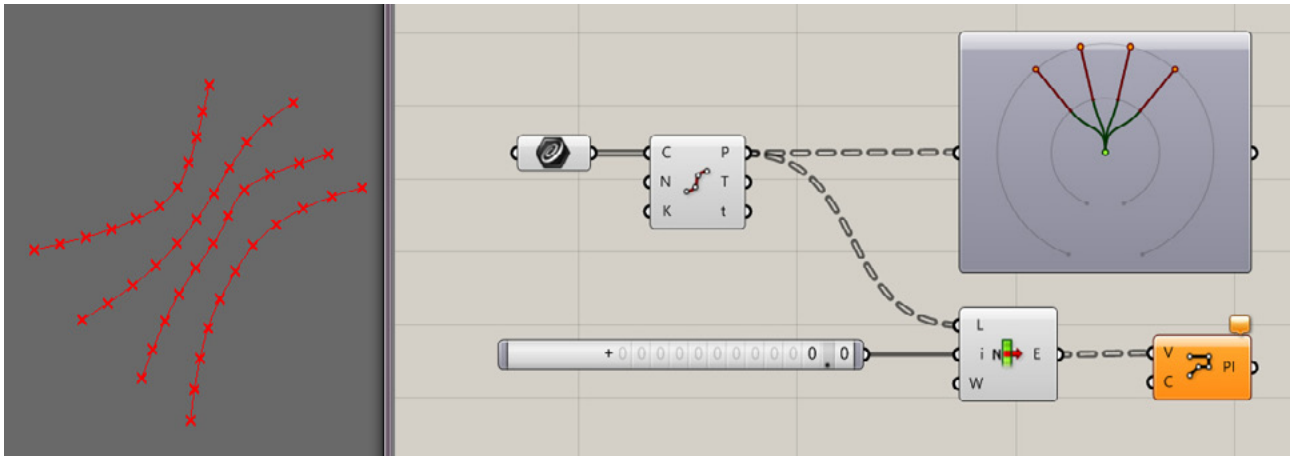
There are 4 different curves presented in a <Curve> component. Connecting this component to a <Param Viewer> from Params > Special, we can see that data is sorted in one data branch. If we select item index 0 from the list of curves, the first curve would be selected.



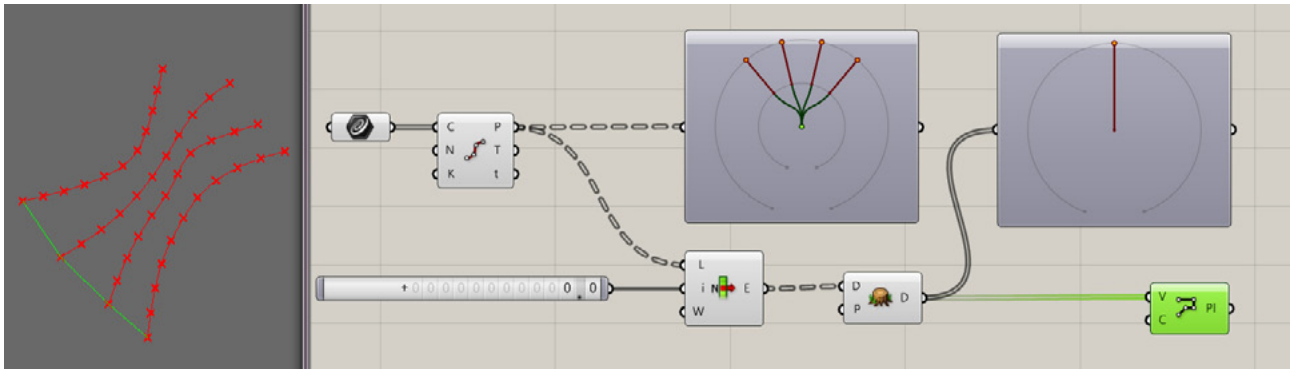
If we use a <divide curve> and then check the data tree status of the result by a <Param Viewer>, we will see that data is divided into branches now, 4 branches actually, the same number as the amount of curves. Since we might need to have access to the points of divided curves separately, data has been branched in this component. Now if we select the item index 0 of these points, we will see that all first points of the curves are selected, not just the first point of the first curve.



Why this is important? There are various situations in which data tree management is needed. In the above example, if we want to make a <Polyline> by connecting all item index 0 of divided curves together, we will see that it would not work. We have 4 points in 4 different data branches and <Polyline> recognize them as 4 separate points, not 4 points in a data list to connect them together.



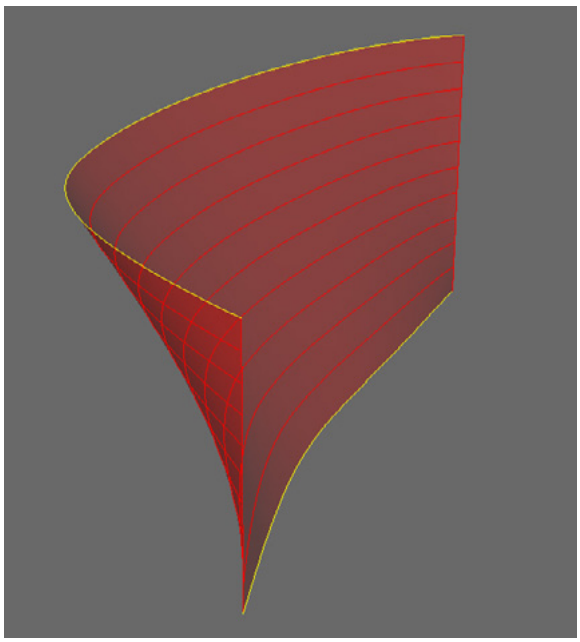
There are a group of components in Sets > Tree which are available to control and manage data in this level. Here for example, using a <Flatten Tree> will help us to remove branches and prepare all data in one data branch (data list) and <Polyline> would have the chance to connect those points in the list together. We will use these components for data manipulation in our design experiments.



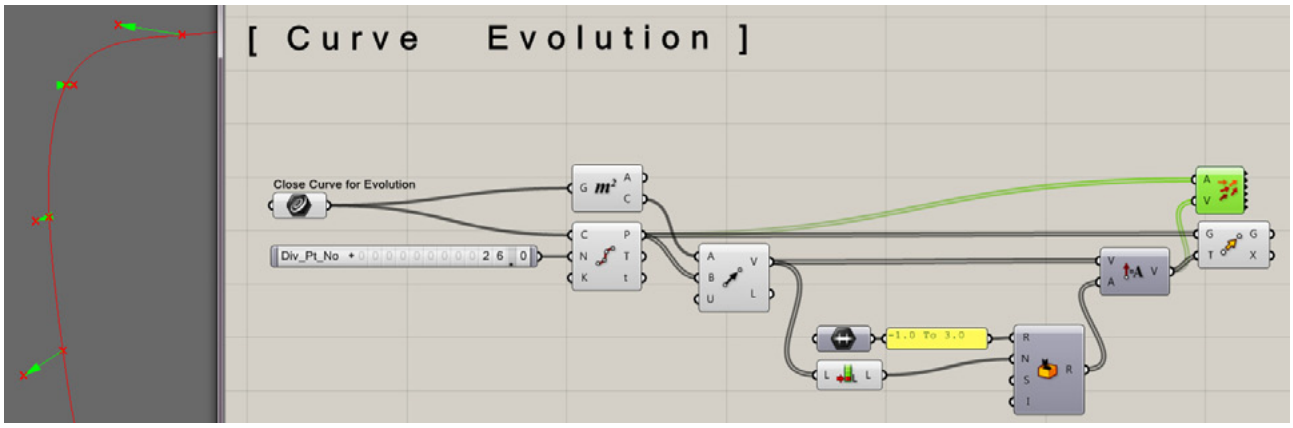
4_5_Curve Evolution

Curve Evolution is the smooth process of shape changing in a curve. For example when we loft various curves to generate a surface, curve evolution happens between them, to change the profile smoothly and generate a smooth surface as well. There are various techniques and also applications for curve evolution or Curve Flow but here we want to use the concept in our way to benefit in design applications.

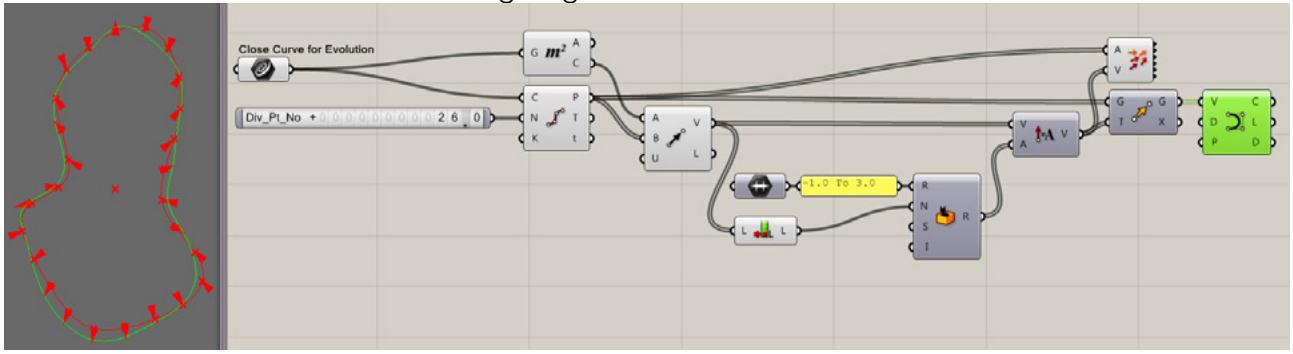
Since contemporary design employs curves for various design situations it is interesting to manipulate curves and extract various profiles out of one primary defined curve. This will help to design smooth surfaces while their fluidity is desirable for designers' tendencies.



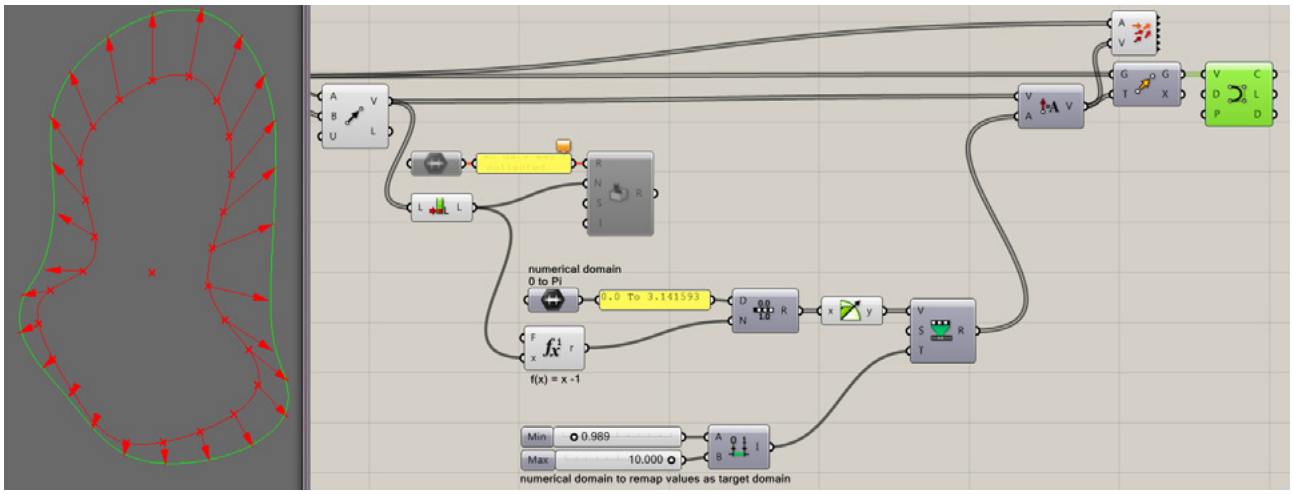
In our design space we have a curve and we want to smoothly evolve this curve to generate variations of this primary curve for further design issues. The idea is to enlarge a closed <Curve> from its central point, using sample points across the curve and moving them away from the centre by <Random> values in this first example. <Vector Display> shows couple of vectors which are displacing sample points across the curve. Here an <Area> component has been used to find the centre of the closed curve.



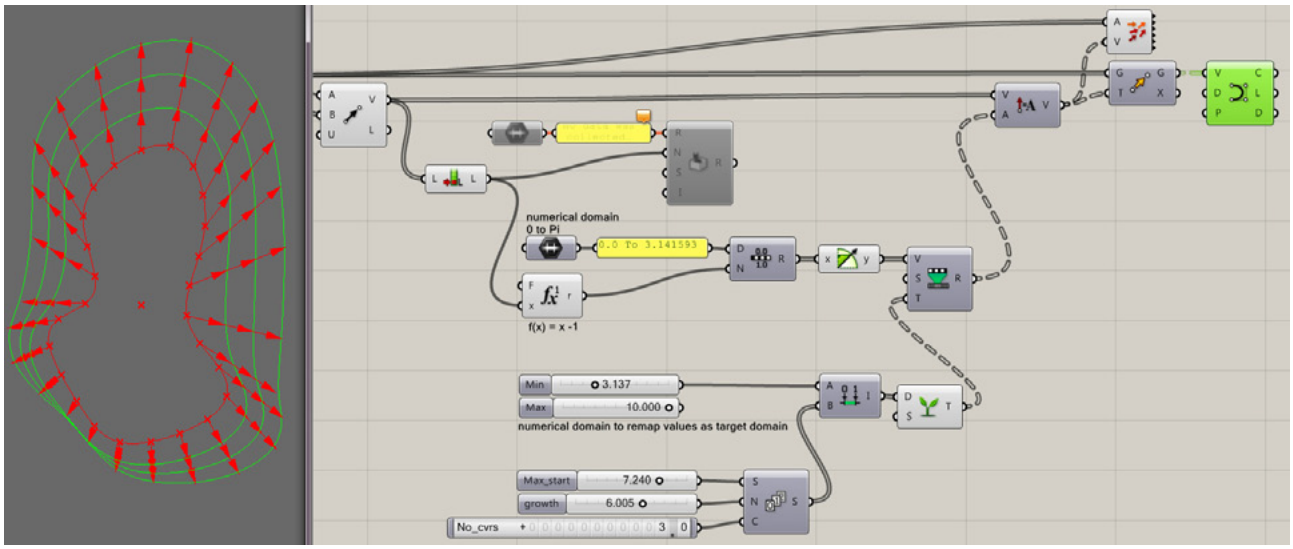
Using an <interpolate>, we can draw the randomly evolved curve. But random evolution might not be desirable and we might look for other options. Let's assume that we want to evolve curve in a way that it starts to deviate from the original curve from a minimum value and reaches a maximum value at the middle of the curve and again goes back to the minimum value to close the curve.

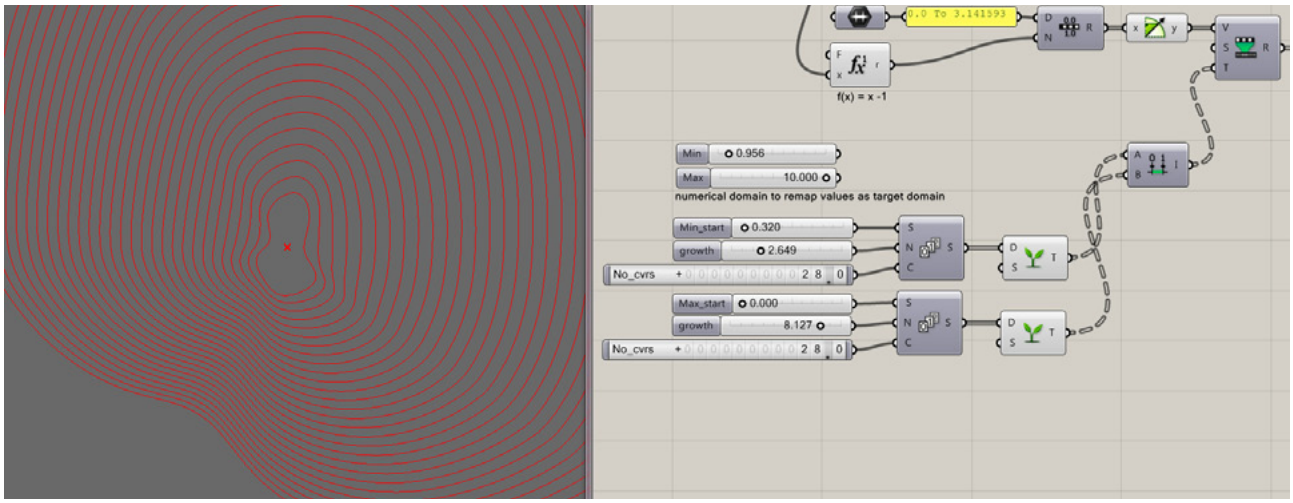


Instead of <Random> values, a range of numbers from 0 to Pi has been used and divided. The <Sine> function generates a repeated set of numbers from 0 to 1 to 0. This set of numbers re-mapped by a <Remap numbers> from Math > Domain in order to raise the amount of deviation by a new numerical domain. These numerical values have been used to empower vectors and displace points. Now you get the concept and you can provide any numerical set to control the curve evolution.



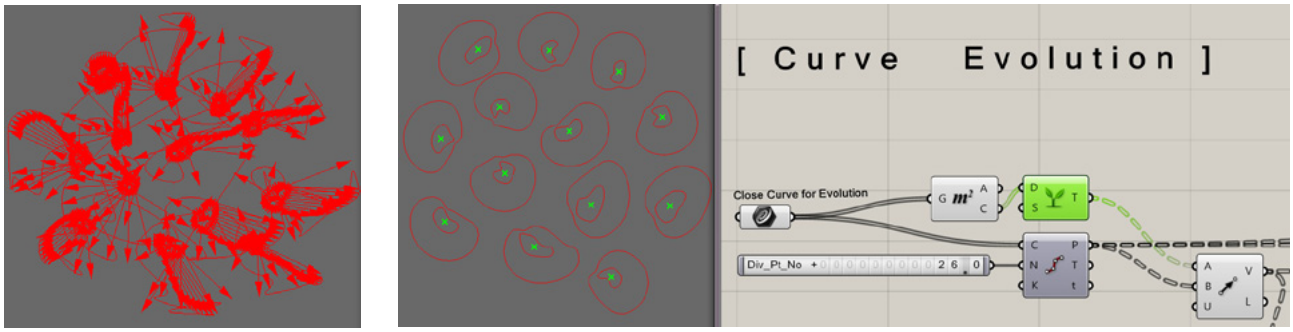
A <Graft> component (Sets > Tree) would generate one separate data branch for each item in its data list. Combing this possibility with a <series>, there are various numerical ranges available for the curve now, and the evolution of curve happens in multiple steps. The Minimum and Maximum of the domain can be controlled by changing numerical values.



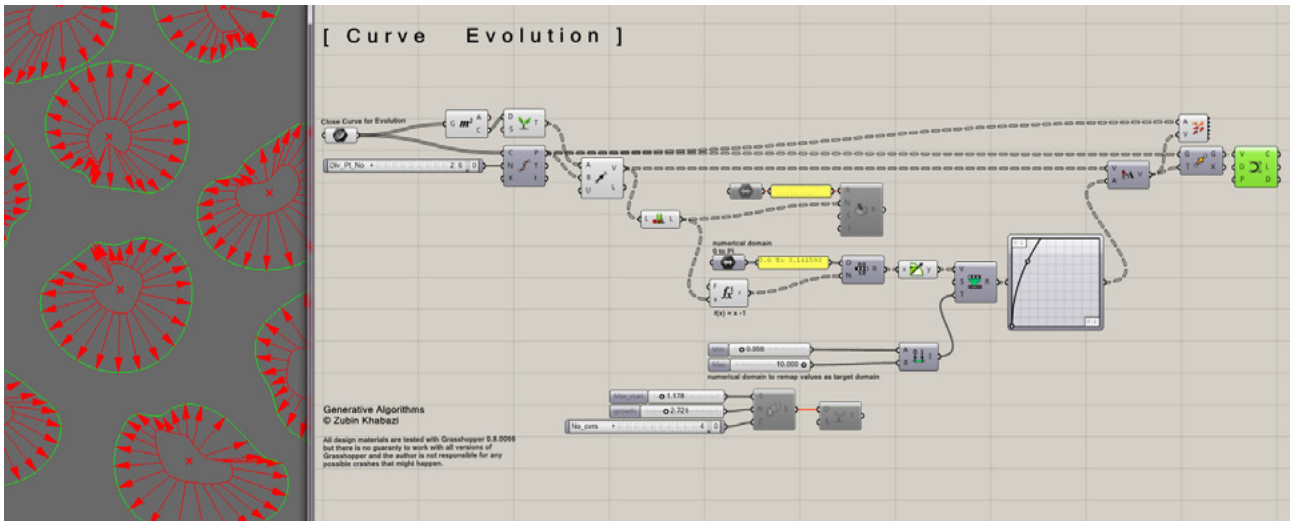


What about using bunch of curves instead of one? Here there are other data tree management needed to get the result. Since all curves are in one data branch but their divided points are in various data branches, we need to <Graft> all centre points to generate vectors in separate data branches as well.

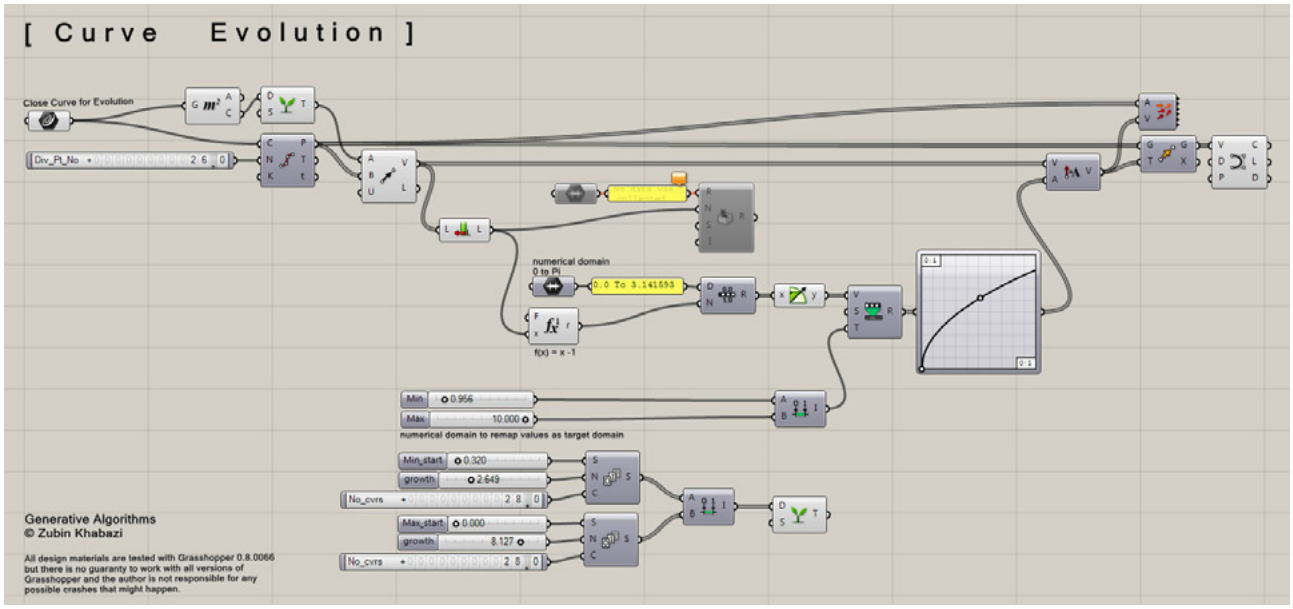
The result would be fine if you have one numerical domain to evolve each curve. Now you can think about how you can evolve all curves in the input data list, more than once. This needs complicated data list and data tree management.

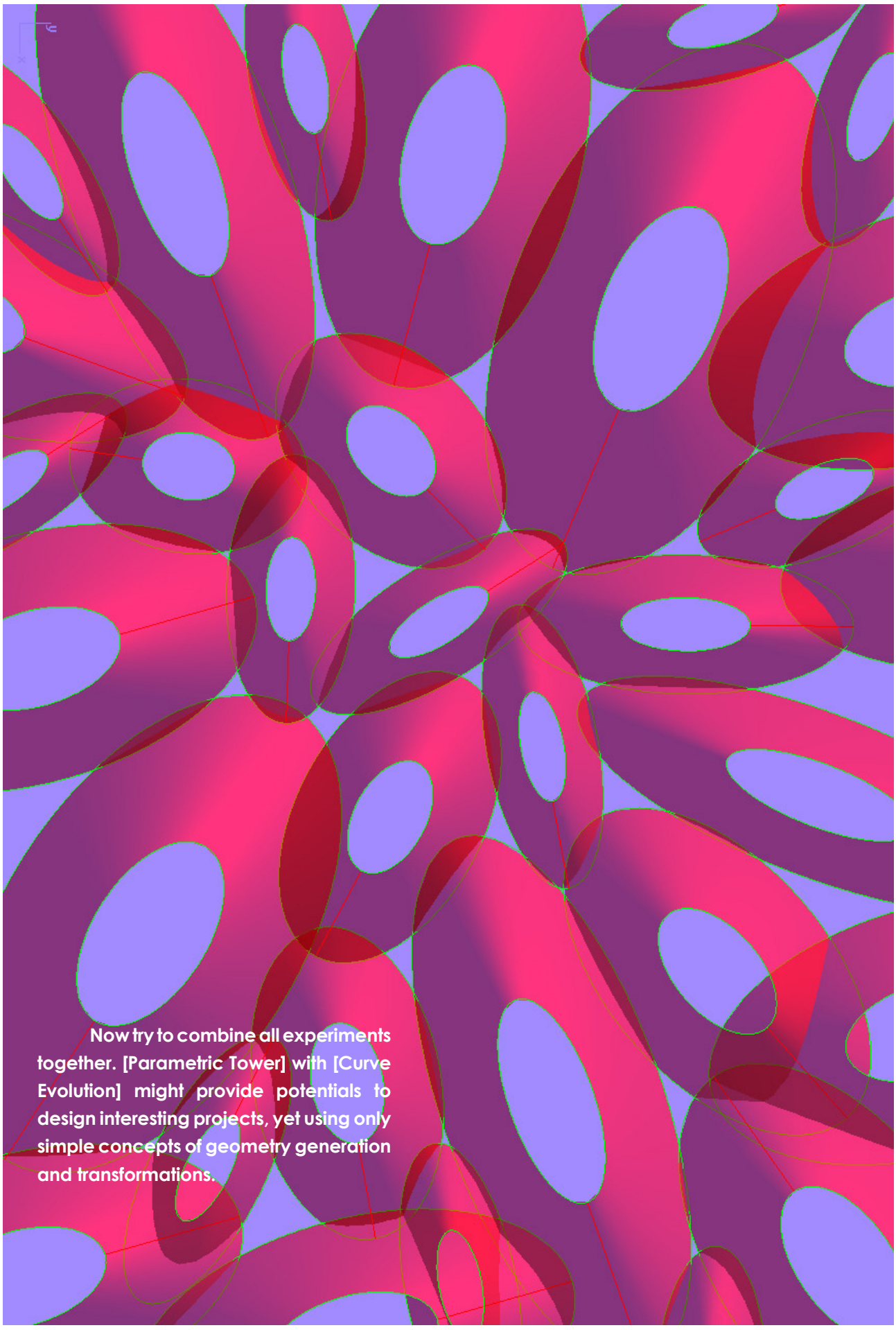


Try changing numerical values by <Graph Mapper> to again redistribute the numbers.



All Experiment files are available with the book for further investigations, so I will keep explanations as minimum as possible.





Now try to combine all experiments together. [Parametric Tower] with [Curve Evolution] might provide potentials to design interesting projects, yet using only simple concepts of geometry generation and transformations.

CHAPTER FIVE

PARAMETRIC SPACE

Chapter Five
Parametric Space

Our survey in Generative Algorithms and Geometry Generation includes observations of objects in space; Digital representation of forms and tectonics; articulation of solids and surfaces and multiple processes of formations; from classical ideas of symmetry and pattern up to complexity and maybe chaos.

We are dealing with objects. These objects could be boxes, spheres, cones, curves, surfaces or any articulation of them. In terms of their presence in the space they generally divided into points as 0-dimensional, curves as 1-dimensional, surfaces as 2-dimensional and solids as 3-dimensional objects.

We formulate the space by coordinate systems to identify some basic properties like position, direction and measurement. The Cartesian coordinate system is a 3 dimensional space which has an Origin point O=(0,0,0) and three orthogonal axis intersecting at this point which make the X, Y and Z directions. But we should consider that this 3D coordinate system also includes two-dimensional (flat space (x, y)) and one-dimension (linear space (x)) systems as well. In parametric design, we are dealing with these systems. We need to enter the interior space of curves or surfaces. We parameterize objects and then deal with their numerical parameters in order to design.

5_1_One Dimensional (1D) Parametric Space

The X axis in Cartesian coordinate system is an infinite line which has some numbers associated with different positions on it. Simply x=0 means the origin and x=2.35 a point on the positive direction of the X axis which is 2.35 unit away from the origin. This simple, one dimensional coordinate system can be parameterised on any curve in the space. So basically not only the World X axis has some real numbers associated with different positions on it, but also any curve in the space has the potential to be parameterized by a series of real numbers that show different positions on the curve. So in 1D parameter space, when we talk about a parameter, it could be described by a real number which is associated with a specific point on a curve.

Since we are not working on the world X axis any more, it is important to know that any curve has its own parameter space and these parameters does not exactly match the universal measurement systems. Any curve in Grasshopper has a parameter space starts from zero and ends in a positive real number.

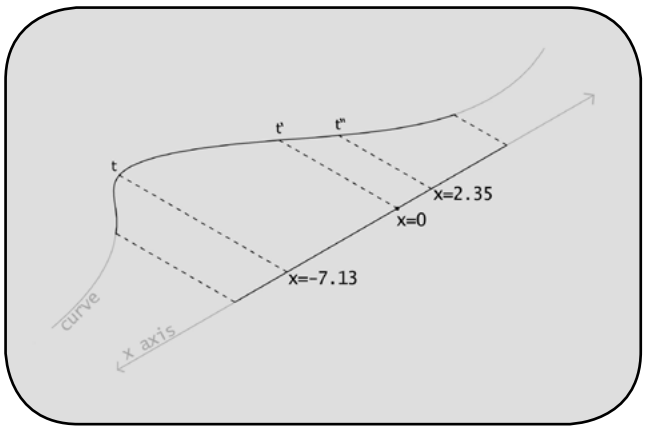


Fig.5.1. 1D-parameter space of a curve. Any 't' value is a real number associated with a position on the curve.

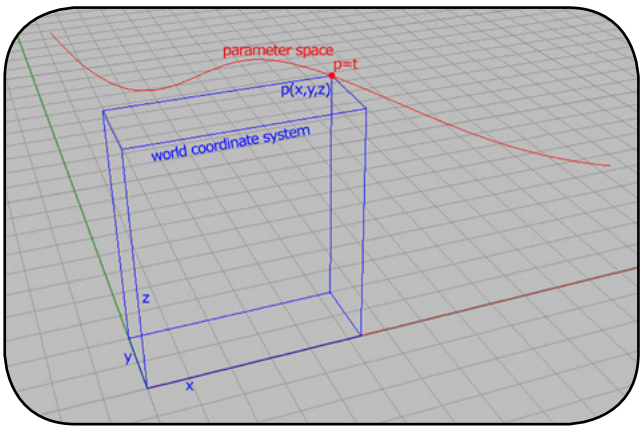


Fig.5.2. 1D-parmeter space and conversion to 3D coordinate system. Talking about a curve and working and referencing some specific positions on it, we do not need to deal with points in 3D space (p=(X,Y,Z)) but we can call a point on a curve by P=t as a specific parameter on it.

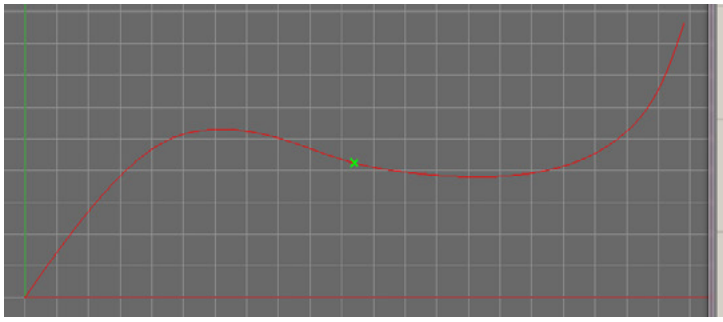
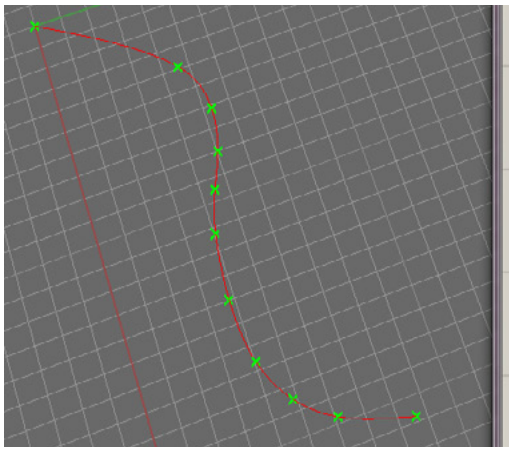


Fig.5.3. <Point on Curve> passes a point an a curve while it divides the domain of a curve from 0 to 1 and generates a point on curve by a number between 0 and 1.



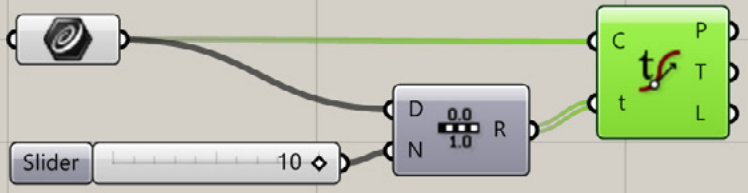


Fig.5.4. An <Evaluate Curve> evaluates points on curve based on numeric values in the domain of the curve.

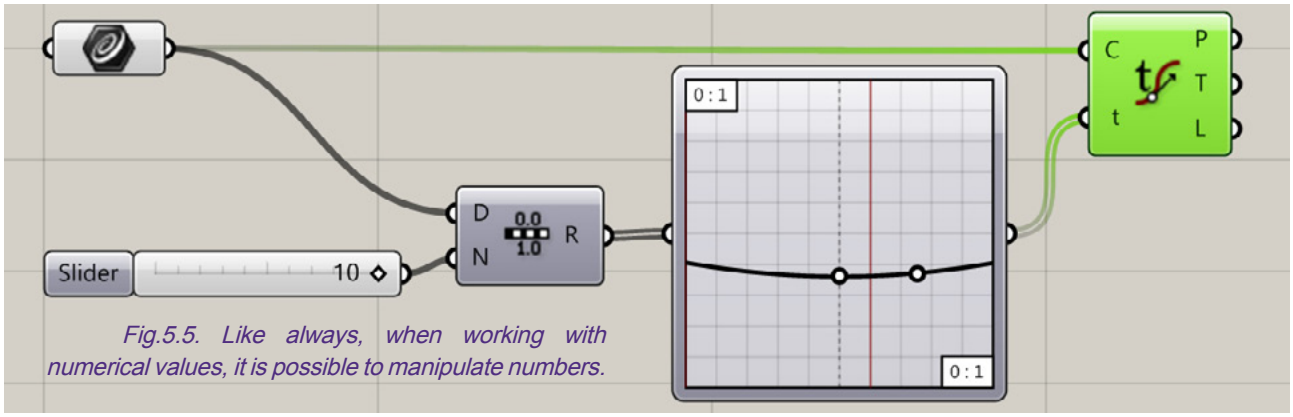


Fig.5.5. Like always, when working with numerical values, it is possible to manipulate numbers.

5_2_Two Dimensional (2D) Parametric Space

Two axis, X and Y of the World coordinate system deals with the points on an infinite flat surface in which, each point is associated with a pair of numbers p=(X,Y). Quite the same as 1D space, here we can imagine that all values of 2D space could be traced not only on World's coordinate flat surface, but also on any generic non-flat surface in space. So basically we can parameterize a coordinate system on a curved surface in space, and call different points of it by a pair of numbers here known as UV space, in which any point P on the surface is P=(U,V). These "Parameters" are specific for each surface by itself and they are not generic data like the World coordinate system, and that's why we call them parametric!

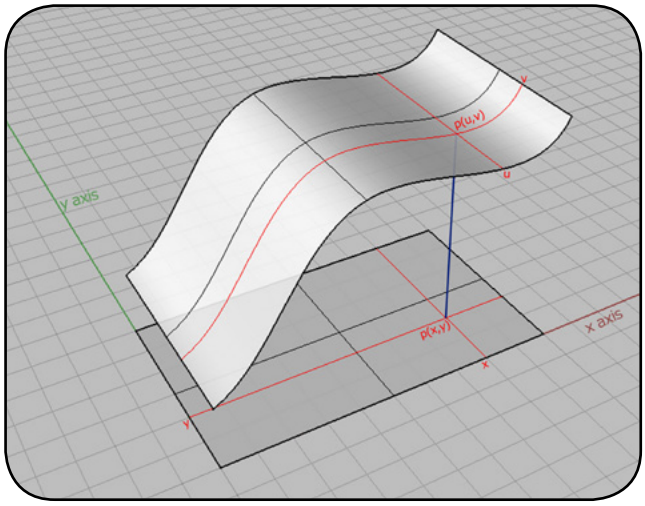


Fig.5.6. UV and 2D Parameter space.

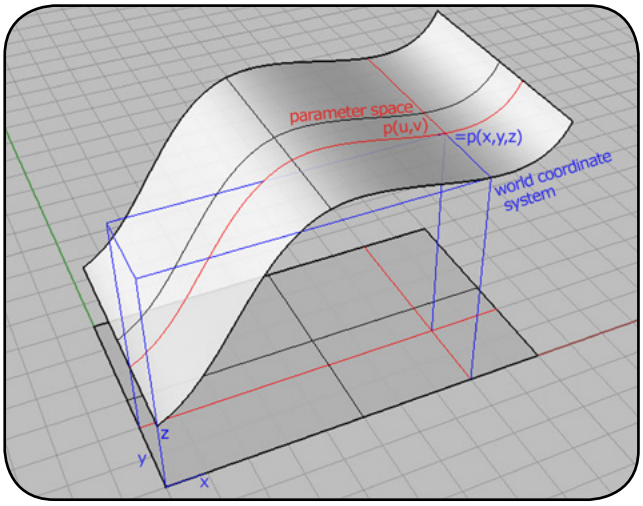


Fig.5.7. Equivalent of the point P=(U,V) on the world coordinate system p=(X,Y,Z).

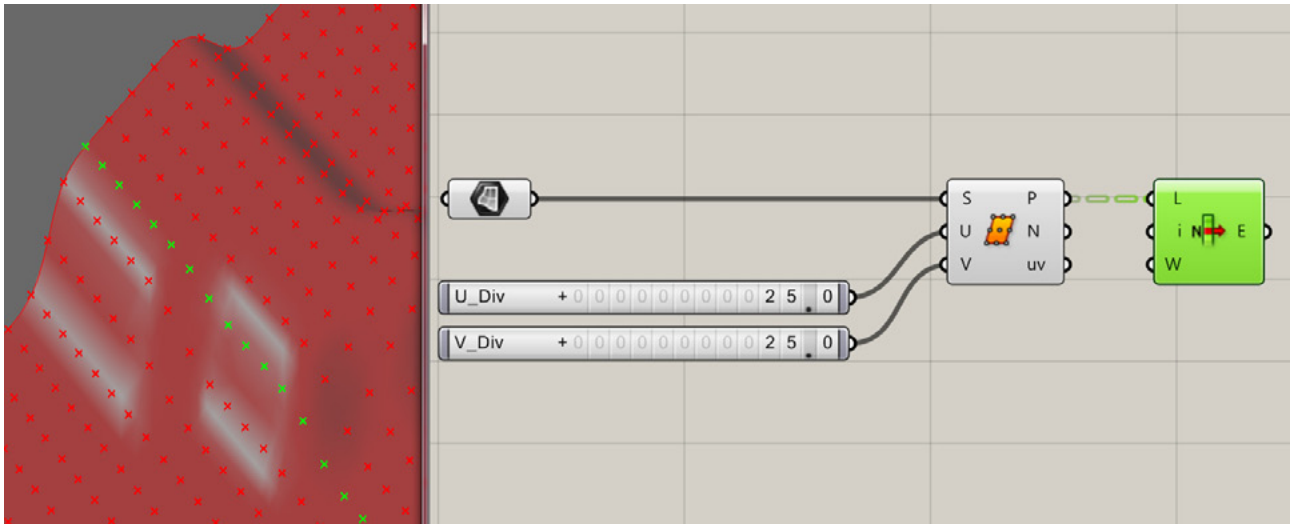


Fig.5.8. A <Divide Surface> asks for U and V division numbers and divides a surface in its U/V directions and generate points. Notice that Points are now sorted into various data branches which are coming from number of divisions in U direction and the number of values in each data branch come from the V division value.

5_3_Transition between spaces

It is a crucial part in parametric design to know exactly which coordinate system or parameter space we need. Working with free form curves and surfaces, we need to provide data for parameter space but we also need to go back and forth for the world coordinate system to provide data for other geometry creations or transformations. It is more complicated in scripting, but since Grasshopper has a visual interface rather than code, you would simply identify which sort of data you need to provide for your design purpose.

5_4_On Object Proliferation in Parametric Space

For many design purposes, designers use free-form surfaces as base geometries to proliferate some other objects on them. Surfaces are flexible, continues two dimensional objects that represent suitable bases for this design objective. There are multiple methods to deal with surfaces like Penalisation, but here I am going to start with one of the simples and we will discuss about some other methods later.

We shall think of various ways that objects could be generated over a free-form surface but after these techniques we need to think about geometrical considerations of such population. Objects might intersect; they might deform or might face some changes which are not desirable for our design. All these issues should be considered while designing with objects in parametric space especially for fabrication purposes, where dealing with real physical objects, not renders.

Object Proliferation

We have a free-form surface and a simple geometry which is a box. The question is how we can proliferate this box over the surface, in which we have control over the macro scale (surface) and micro scale (box) separately.

The design process encompasses dividing a surface and generating boxes over these divisions. Then by adding transformations to the boxes, we will have additional controlling options.

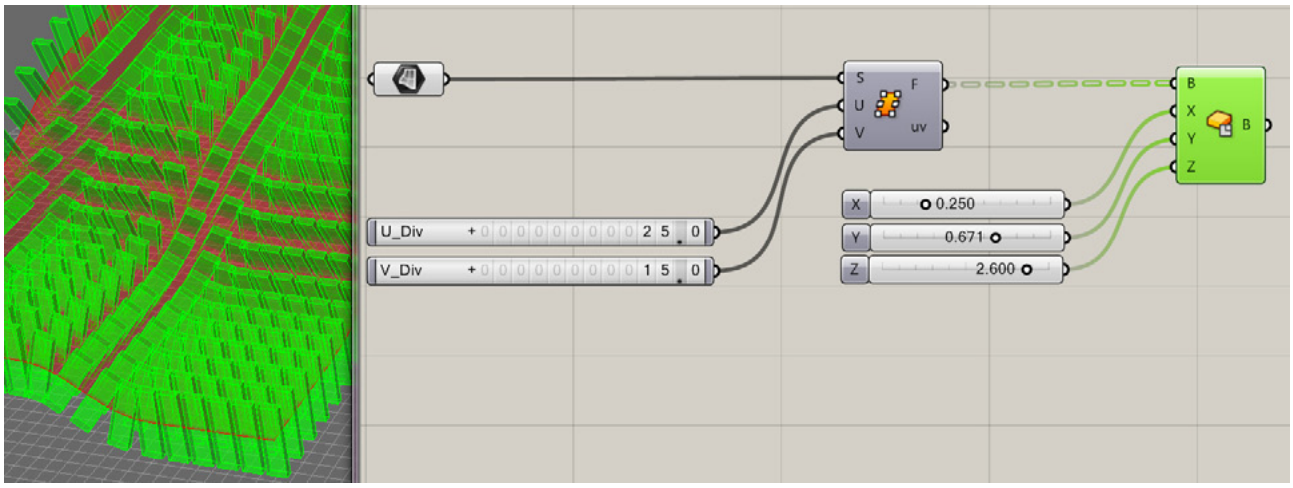


Fig.5.9. A free-form <surface> is being divided by a <surface frames> which is generated some planes over the surface. These planes are used as base planes for some <Box Plane> objects with manually set dimensions.

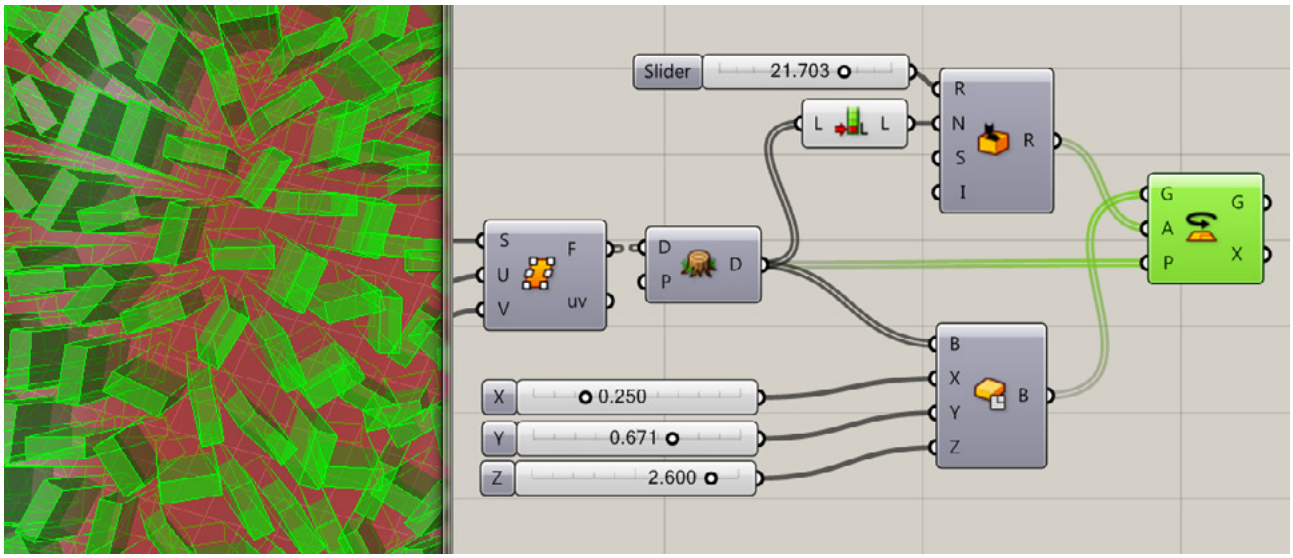


Fig.5.10. In order to manipulate boxes locally, the rotation has been tried. Don't forget to uncheck the Preview of the previously generated objects. Take care of the data tree management.

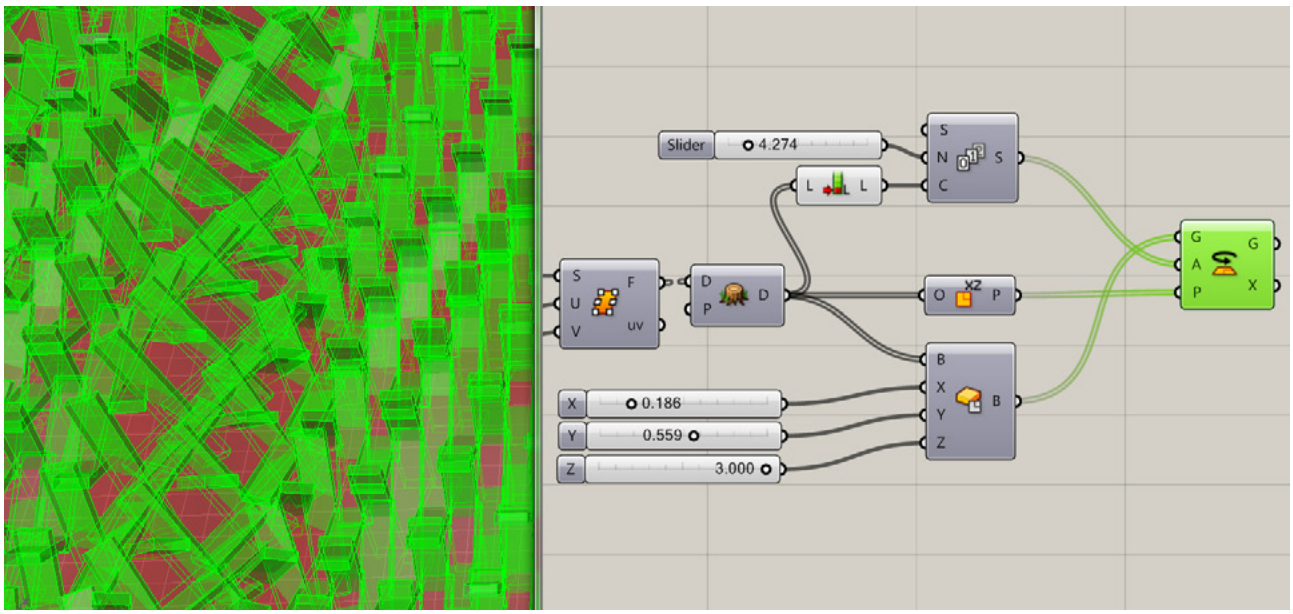
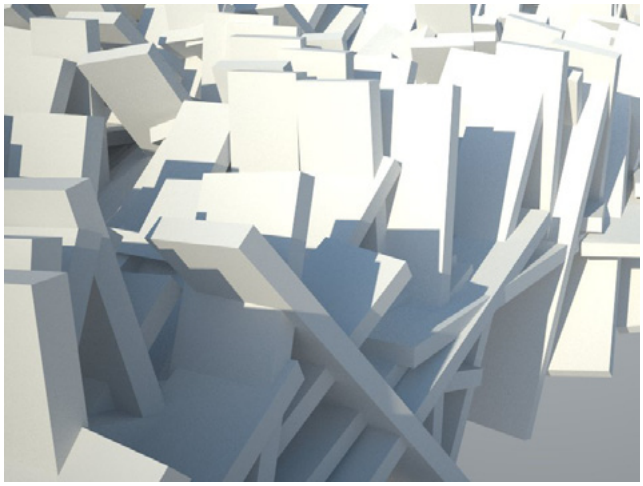
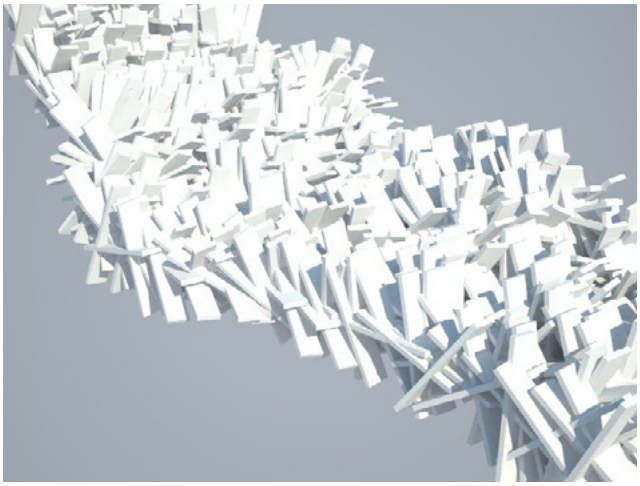


Fig.5.11. To have a hidden order in the arrangement of boxes, a series of gradually changing numbers have been used as angle of rotation. There are various other options to manipulate objects locally. We will discuss that how we should approach this manipulation in association with other elements of design.





Surface Subdivision

One step forward in object population on 3D curve surfaces is to draw or generate objects based on the specific design which could be more complex than basic Euclidean geometries. The process encompasses subdivision of the surface and then generating geometries, using those subdivided elements (Sub-Surfaces or Division Points or Planes). Here is an example.

In design space we have a surface that should be subdivided and then some 4-corner surfaces are aimed to be generated at each sub-surface in which 3 corners are remain on the base surface and one of the corners get a height which should be generated by a perpendicular translation from the base surface at each point.

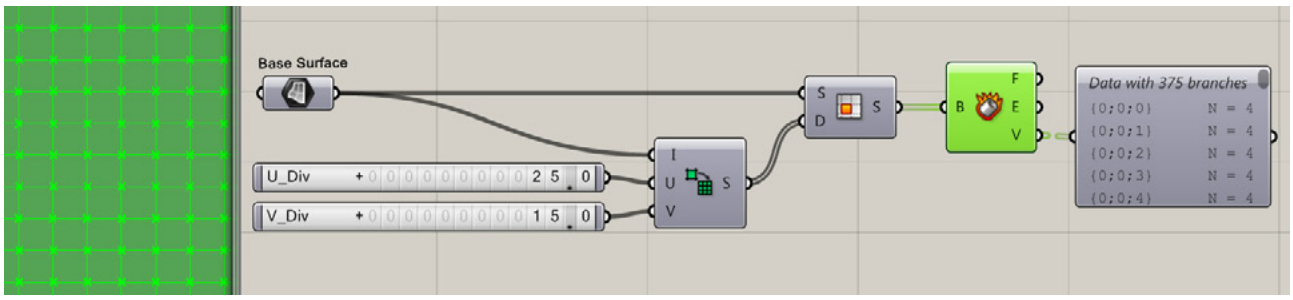


Fig.5.12. Base Surface is subdivided by an <Isotrim> using <Divide Domain2>. Sub-Surfaces are passed to a <Brep Components> to have access to their vertices.

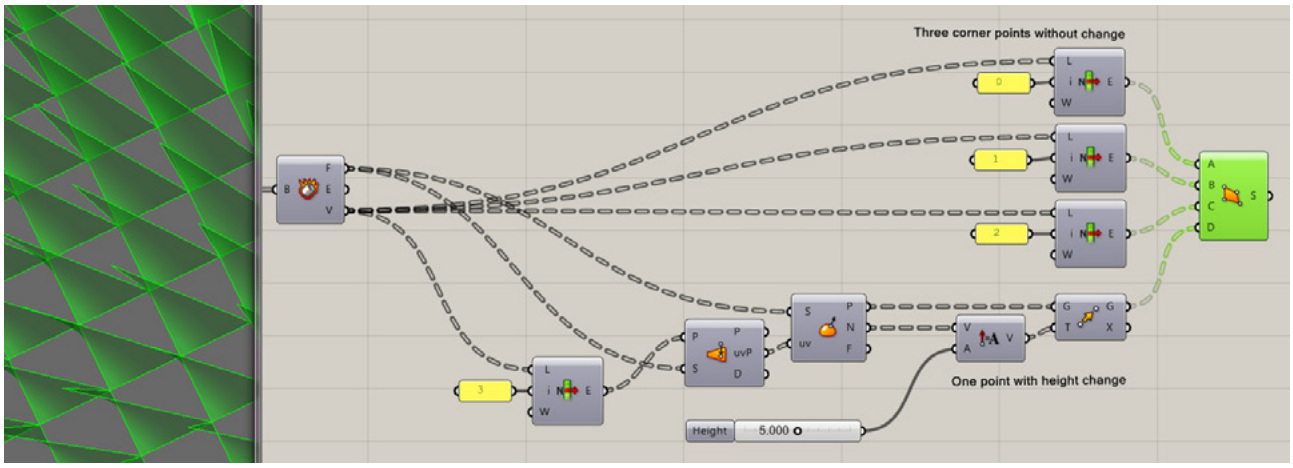
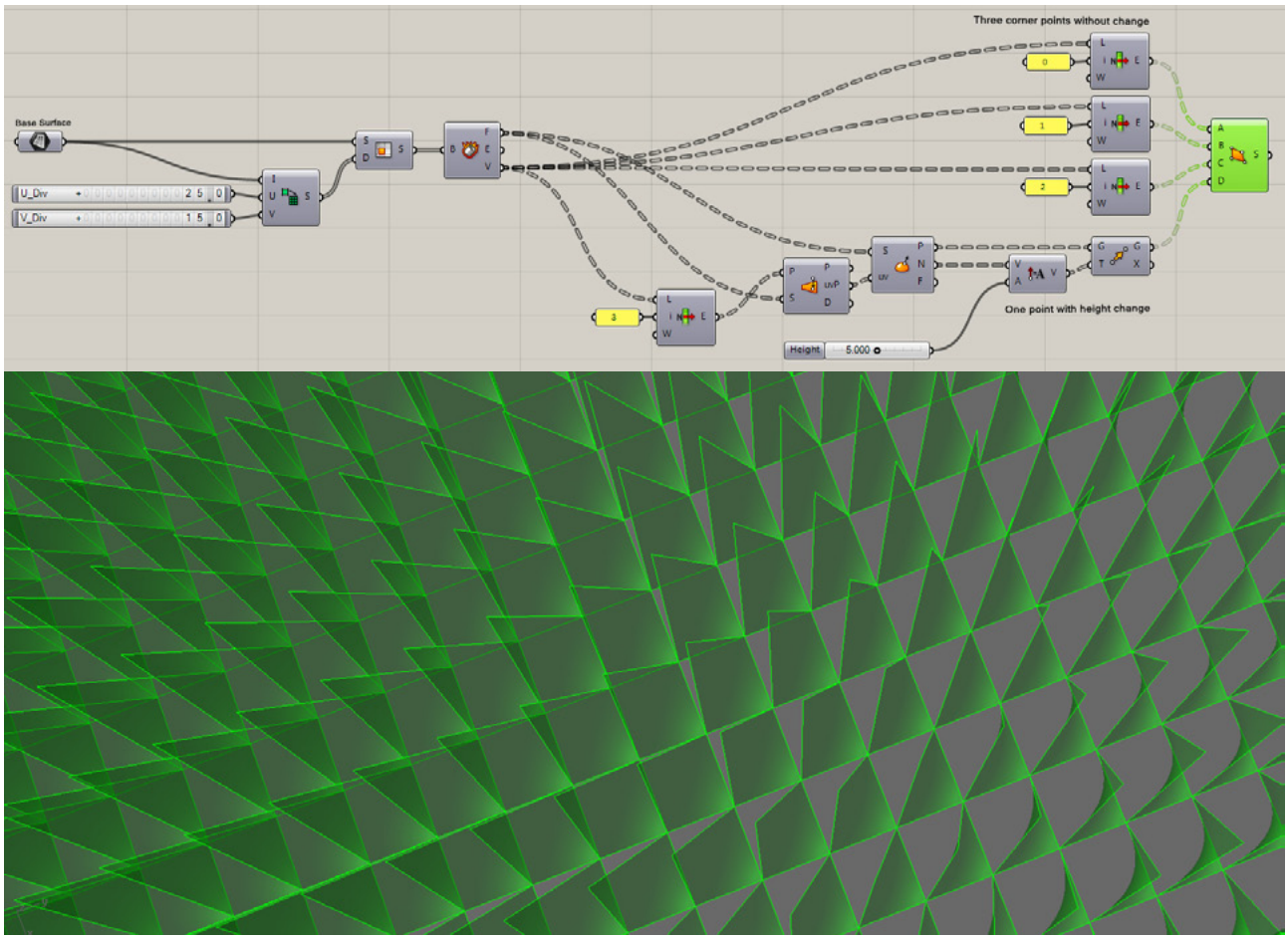
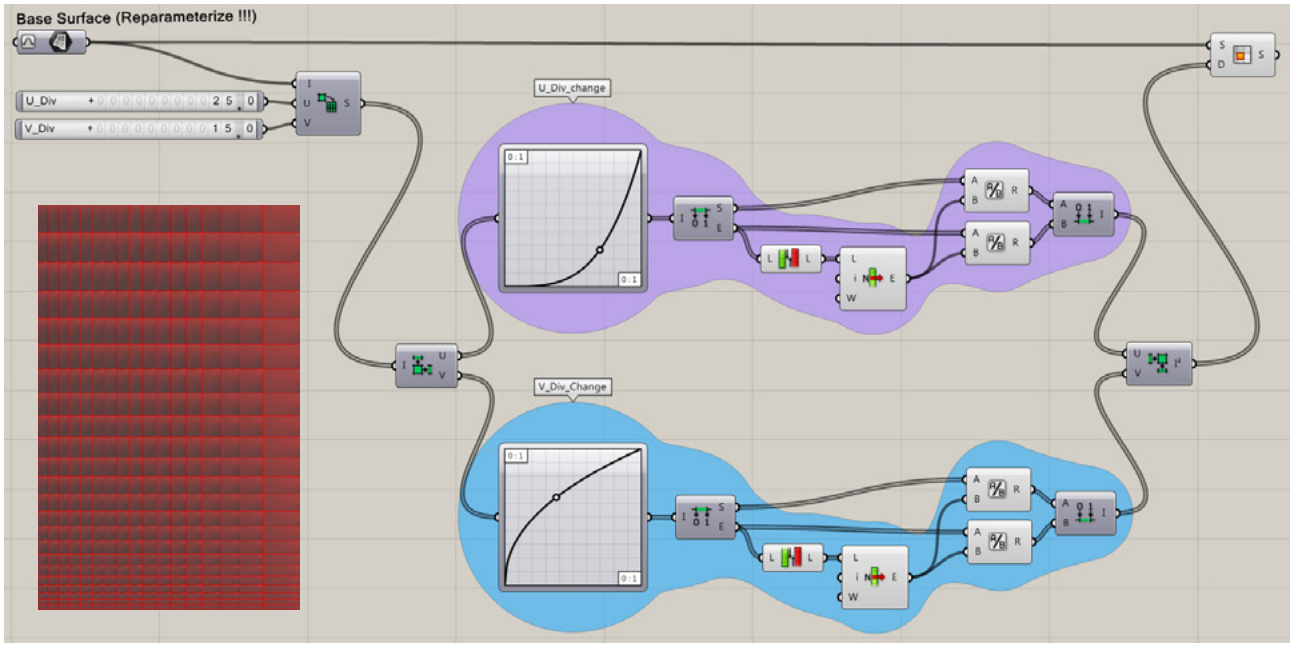


Fig.5.13. Four different corners are selected separately. The last one has been moved in the direction of the surface Normal which is available through <Evaluate Surface> Component. A Normal is a vector, perpendicular to the surface at any specific point of it. <Surface CP> (Surface Closest Point) has been used to find the UV coordinate of selected points. <4Point Surfaces> are generated using the extracted points and also moved ones.





[Surface Population]

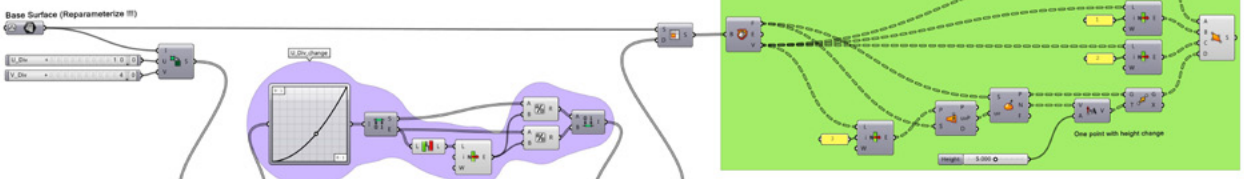
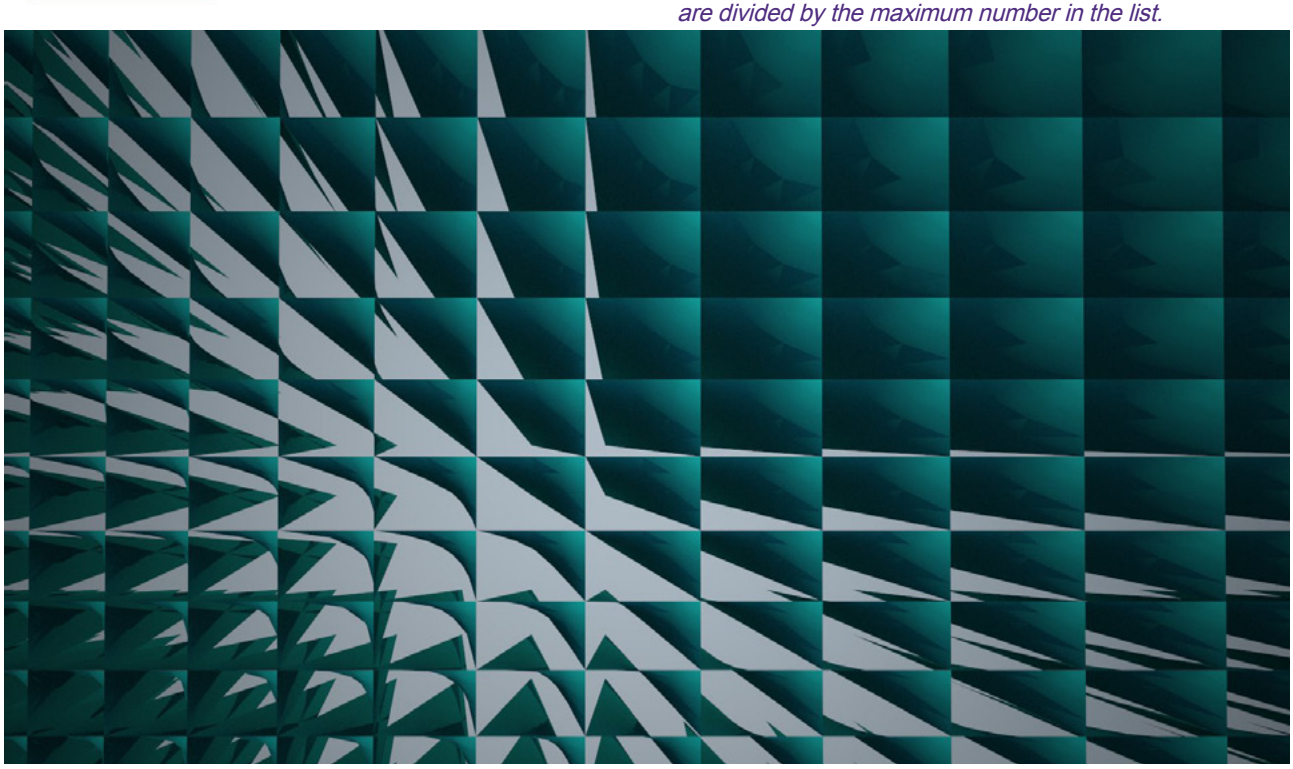


Fig.5.14. Once again working with numbers in domain divisions has the potential for numerical manipulations. The base surface has been 'Reparameterized' through its context menu, to set its domain from 0 to 1 in both U and V directions. Here components of domains are extracted and manipulated and again converted into a numerical domain. To have manipulated values from 0 to 1 again, all of them are divided by the maximum number in the list.



5_5_On Differentiation

Contemporary design techniques include keywords like Fluidity, Smoothness, Differentiation and Versioning, Adaptability, Responsiveness and so on, among them Differentiation is interesting for this part of the Generative Algorithms. Differentiation could be described as small changes that happen to the repetitive elements of design that cause a gradient in their formation. It might happen in all design elements or might occur only on some part of the design space.

Since there is a vast usage of free-form, fluid surfaces in algorithmic design which are sub-divided into small parts, differentiation happens a lot in designing such component based objects. Differentiation might start with visual effects but could entangle with responsiveness of building components and adaptation of the building to the host environment as well.

Differentiation could be applied on various properties of objects. Size and scale of objects is one target. Size differentiation could gradually enlarge objects in design space. Differentiation could also happen on the objects orientation or position. This type of differentiation should be achieved through transformation. Differentiation might also happen through shape changing. We will discuss it later via Deformation. Differentiation could be applied on all design elements in the field or could be limited to some part of the design with the defined positions or under the influence of external stimuli like Attractors. Let's discuss them through experiments as always.

Differentiations Achieved by gradual Transformations :

_ Size Differentiation

_ Orientation Differentiation

_ ...

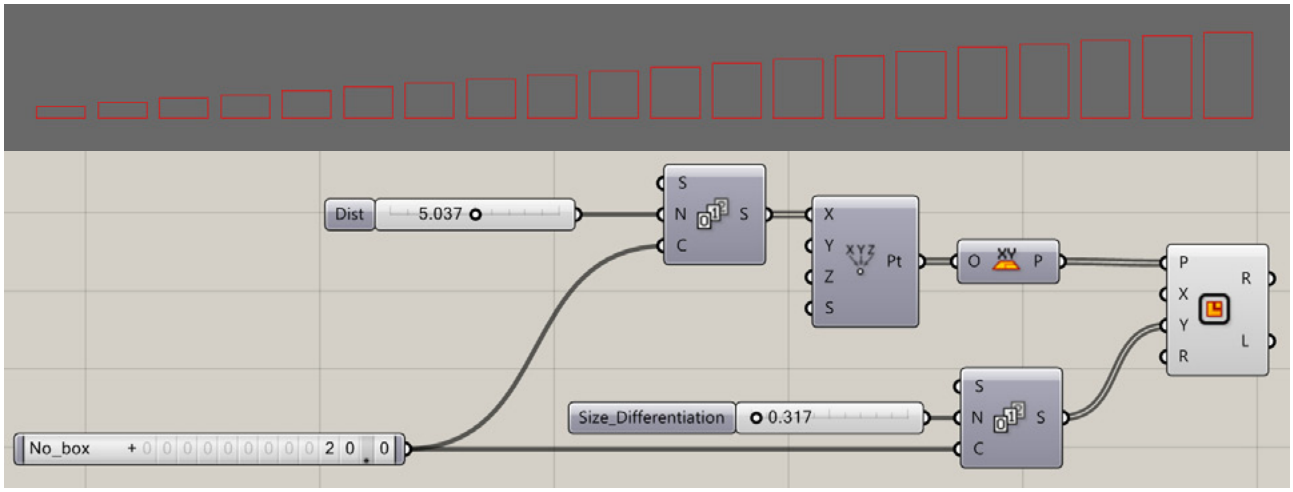


Fig.5.15. Size Differentiation in rectangles.

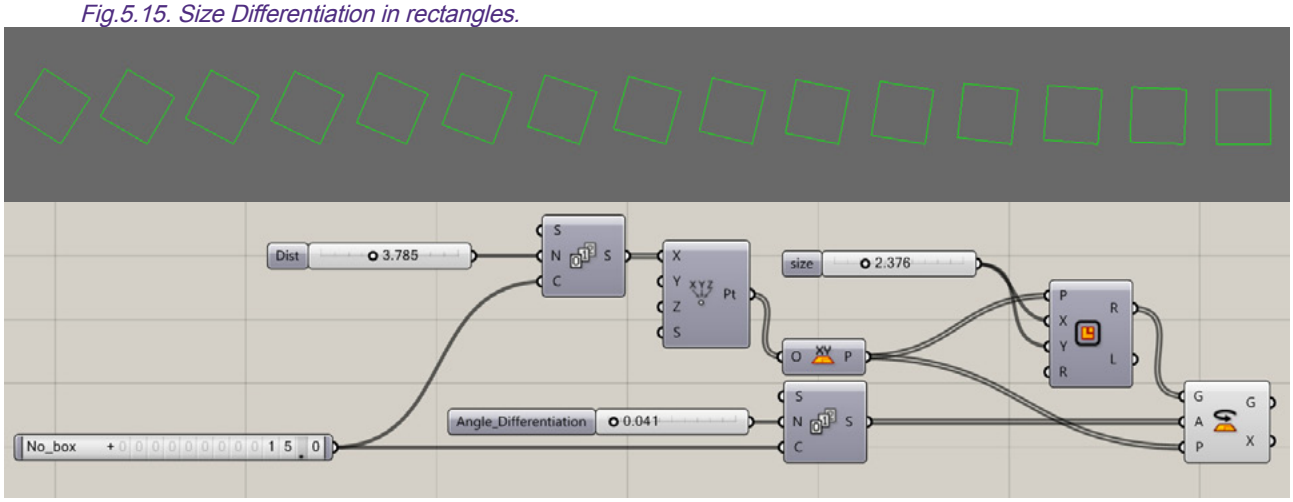


Fig.5.16. Angle (Orientation) Differentiation in rectangles.

5_5_1_ On Attractors

As mentioned, differentiation could be applied on all design elements, but also could be restricted on some specific parts of the design. This restriction could be defined by numbers of elements or coordinates or could be defined in relation to other design elements like Attractors.

"Attractor is a set of states of a dynamic physical system towards which that system tends to evolve, regardless of the starting conditions of the system. A point attractor is an attractor consisting of a single state. For example, a marble rolling in a smooth, rounded bowl will always come to rest at the lowest point, in the bottom centre of the bowl; the final state of position and motionlessness is a point attractor." (Dictionary.com/Science Dictionary)

In case of design and geometry, attractors are elements (usually points) that affect other geometries in the space in a certain filed, and could change their behaviour and force them to displace, re-orientate, rescale, etc. They can articulate the space around themselves and introduce fields of actions with specific radius of power. Attractors have different applications in parametric design and could be introduced into design algorithm as various external stimuli which can affect the design elements. As a very simple example, Sun in designing elements of a façade system could be defined as an attractor for them.

Point Attractor

A very simple example of an attractor could be a point attractor that has the potential to affect design elements in a field and change their size. To make everything simple, elements are circles in this example and attractor could change their radius. It is important that attractor should have a radius of action so it does not affect all elements in design, but those which are in its radius of action.

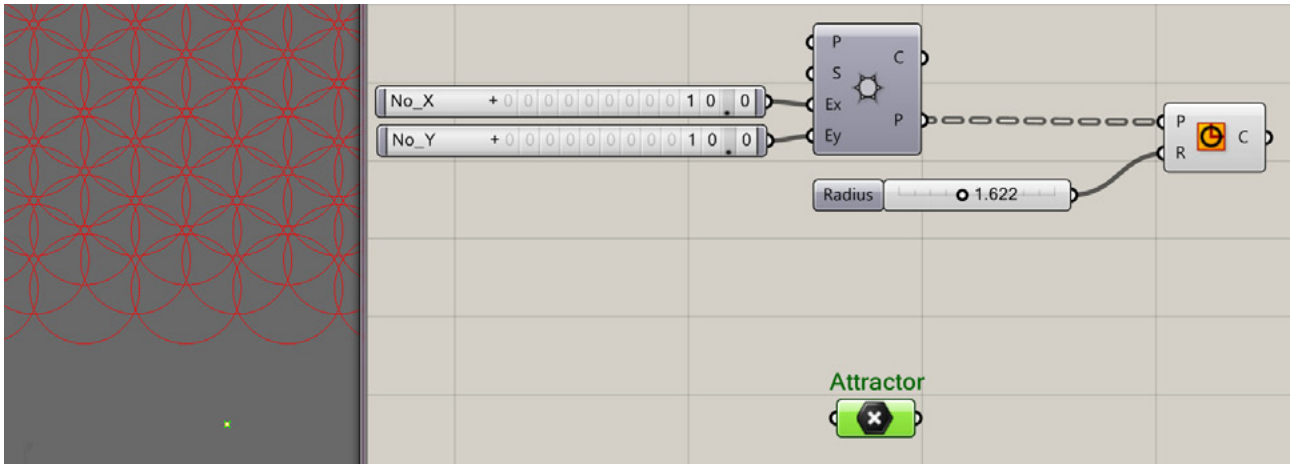


Fig.5.17. A point Attractor and a grid of circles. The aim is to affect circle sizes in relation to the attractor.

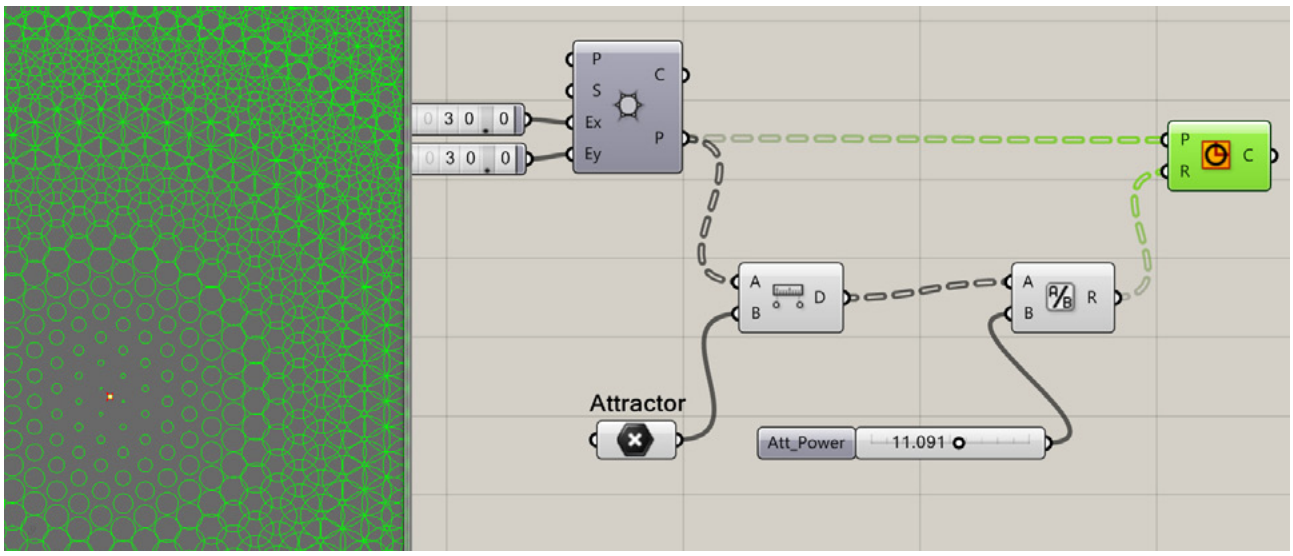


Fig.5.18. The <Distance> of attractor to the points (Circles' Centers) is the most important value in this definition which is going to be used for setting the circles' radius.

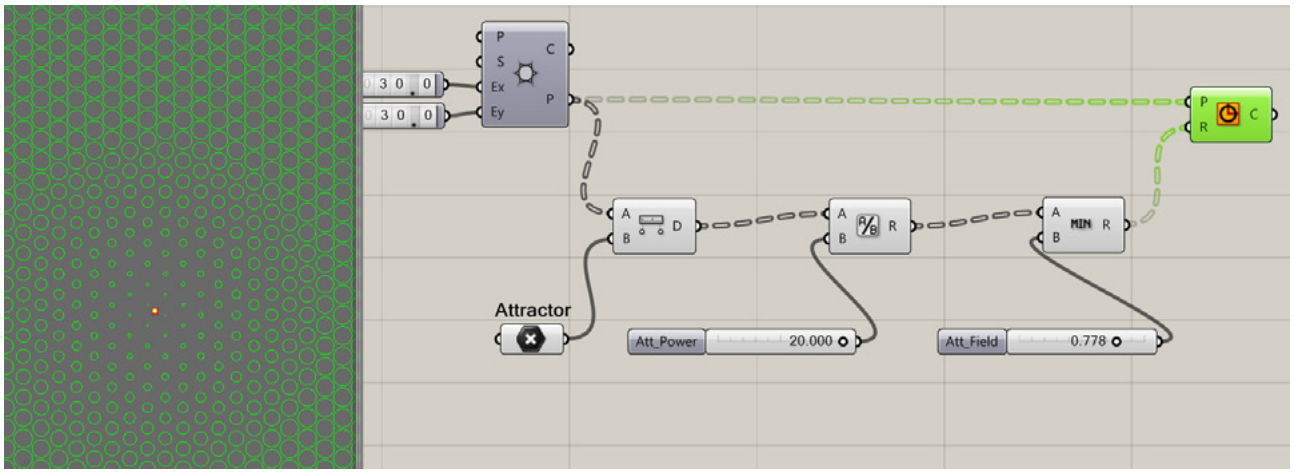


Fig.5.19. The radius of circles would enlarge as much as they get far away from the attractor but there should be a limit to this enlargement. Here a maximum value is set to use as radius of circles if getting bigger than predefined value.

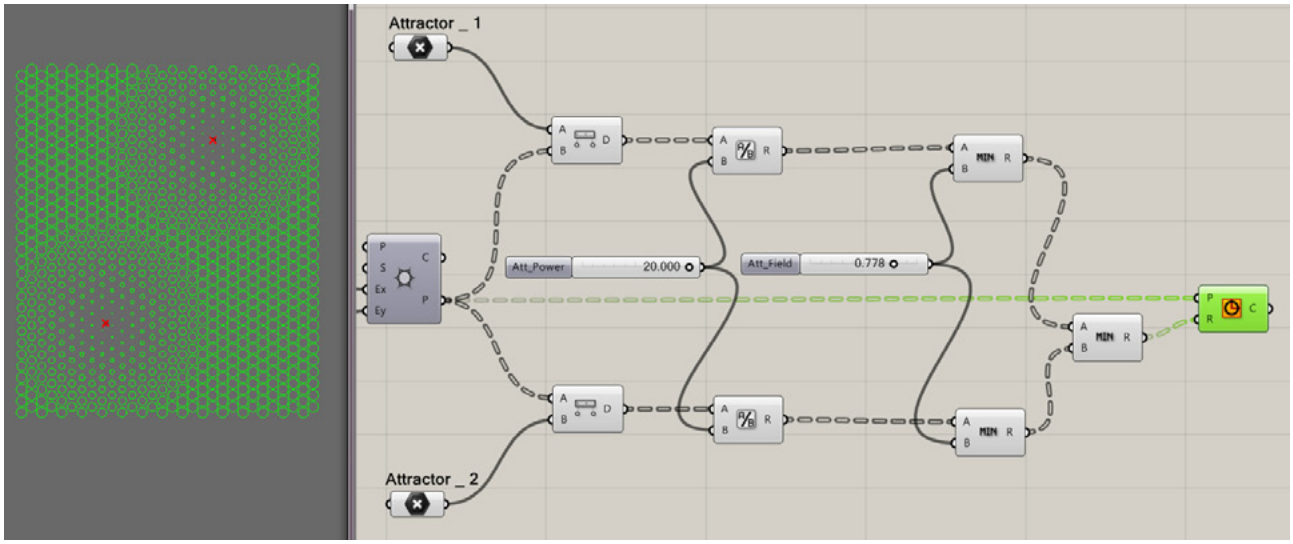


Fig.5.20. Quite the same as the first attractor, there could be another one with the same calculations. At the end, radius of circles from both attractors should be compared and the smaller one should be passed to the circle component.

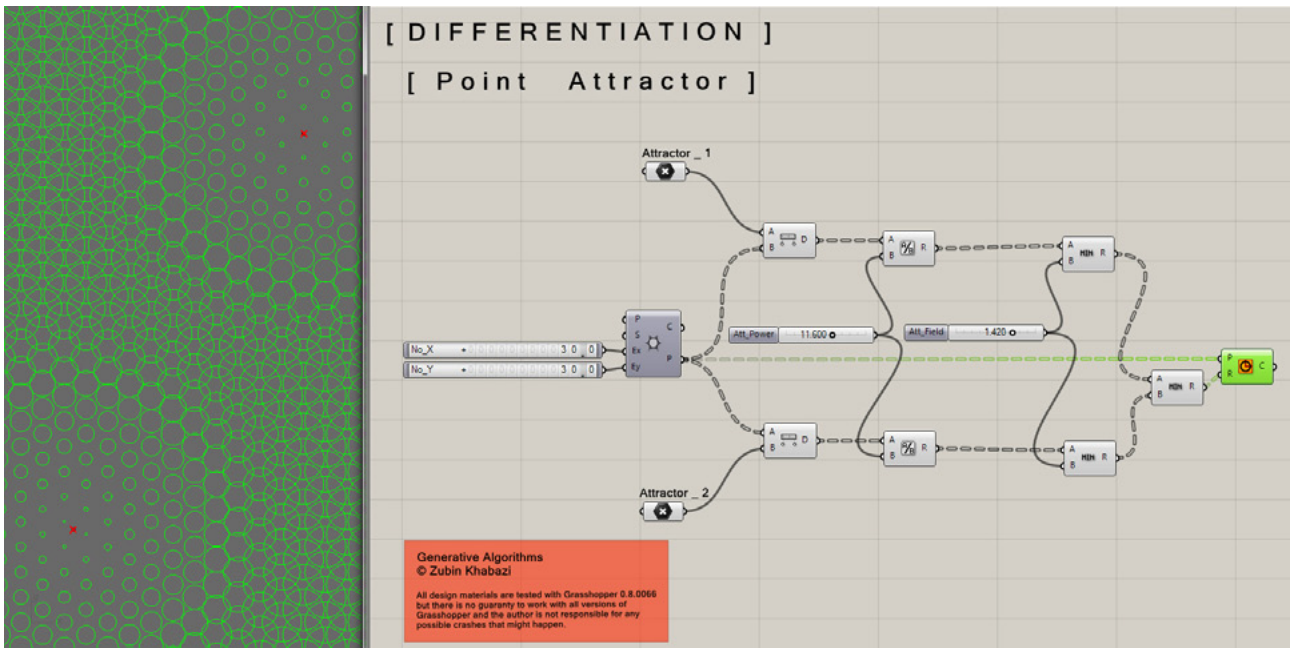
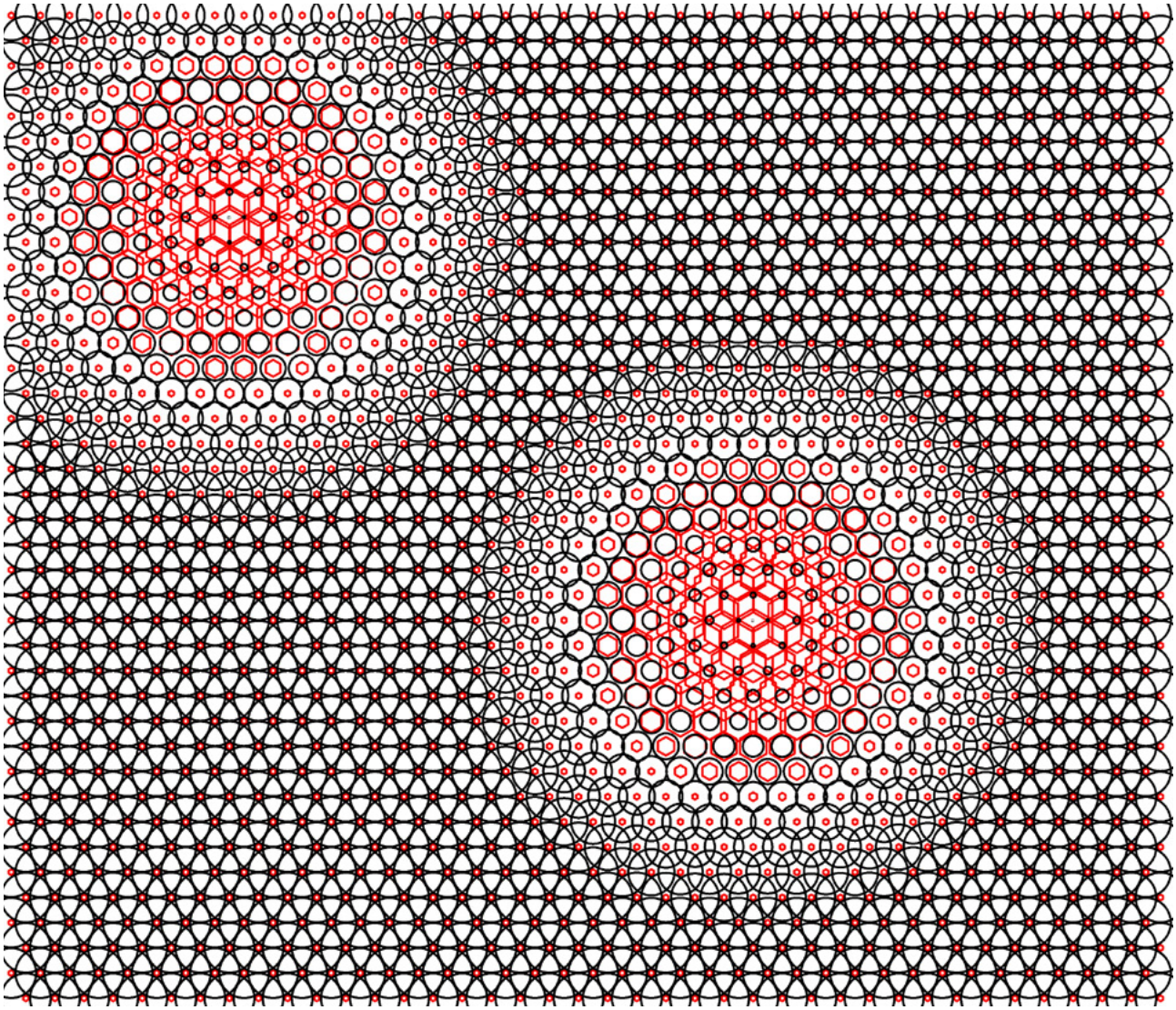


Fig.5.21. A Point Attractor definition to apply Differentiation on design elements (in this case Size Differentiation).



PART TWO
FABRICATION

CHAPTER SIX

DATA OUTPUT FOR FABRICATION

Chapter Six
Data Output for Fabrication

As much as Digital Design and CAD helped designers to utilize computers in design industry, CAM (Computer-Aided Manufacturing) facilitated the utilization of digital manufacturing technologies in construction of architecture, known as Digital Fabrication. Complex design products which come out of algorithmic design processes need relevant fabrication techniques, which should be fast, precise and also computer based means that instead of working with drawings, 'code' would be used as the medium of data transfer for fabricating architecture.

There are various technologies for digital fabrication based on machineries that can be used which include CNC machines, Laser Cutter, Water jet, Robotic Arms or rapid prototyping technologies like 3D print machines or smaller CNC machines with various software technologies. Based on these various machines which have different potentials, availabilities, costs, sizes, powers, and limitations, fabrication techniques have developed in design as well.

Fabrication techniques are generally categorized under three main techniques which are Cutting based, Subtractive and Additive Technologies (Pottman et al, 2007). Since these technologies are available through certain machineries with defined properties, designers should be aware of what is the technique they want to use for their designed system, what is the material, what is the size limitation and what are the post-production necessities in order to be able to design. When designing with such awareness, the design product would be fabrication-informed and the fabrication process would become easier to pursue.

In order to be able to use these fabrication technologies with certain properties that they impose on design products, designers developed various design techniques to be able to transfer projects into those machineries and fabricate their products. These techniques are developing fast and growing based on projects but as Lisa Iwamoto for example mentions in 'Digital Fabrication' some of them could be Sectioning, Tessellating, Folding, Contouring and Forming (Iwamoto, 2009).

It is really important to understand which technique is relevant to the project that we are working on and what sort of data we should provide for our fabrication. We might need datasheets to have coordinates or measurements or simple drawings or codes for machineries. Some of the techniques are discussed in this part of the book.

8_1_Datasheets

Talking about digital design, we need data as output of design algorithm for fabrication. This data could be code that is readable by certain machines (Like G-Code) or sometimes just numerical values like coordinates, angles and measurements. The important point is the correct and precise selection of elements that we need to address for any specific purpose. We should be aware of geometrical complexities that exist in design to choose desired positions for measurement purposes. The next point is to find positions that give us proper data for our fabrication purpose and avoid to generate lots of tables of numbers which could be time consuming in big projects but useless at the end. Finally we need to export data from 3D software to spreadsheets and datasheets and sometimes we need to manipulate this data in a way needed.

Paper-Strip Project

The idea and technique of paper strips is simple but interesting. To understand the logic of assemblies I started with very simple combinations and how these strips could be marked and measured in order to interconnect them, retain their shape in specific positions.

The aim was to use three paper strips and connect them, one in the middle and another two in both sides, but with longer length where restricted and interconnected at certain points to the middle strip. This could be the basic module to repeat and generate bigger assemblies.

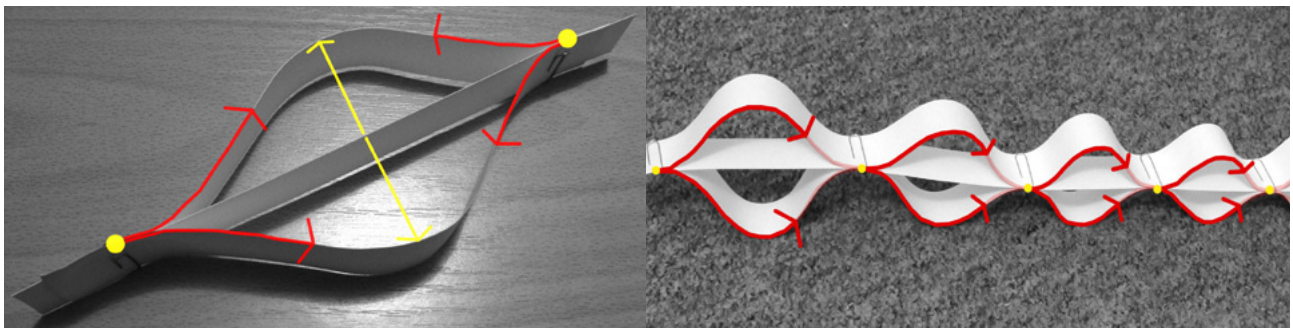


Fig.6.1. Logic of interconnectivity in paper strips

Digital modelling

After basic understanding of the physical model, the digital model developed for design. In this example, curves are used as base geometries for further extrusion to convert them to strips. I have added a gradual size-differentiation in connection points as well.

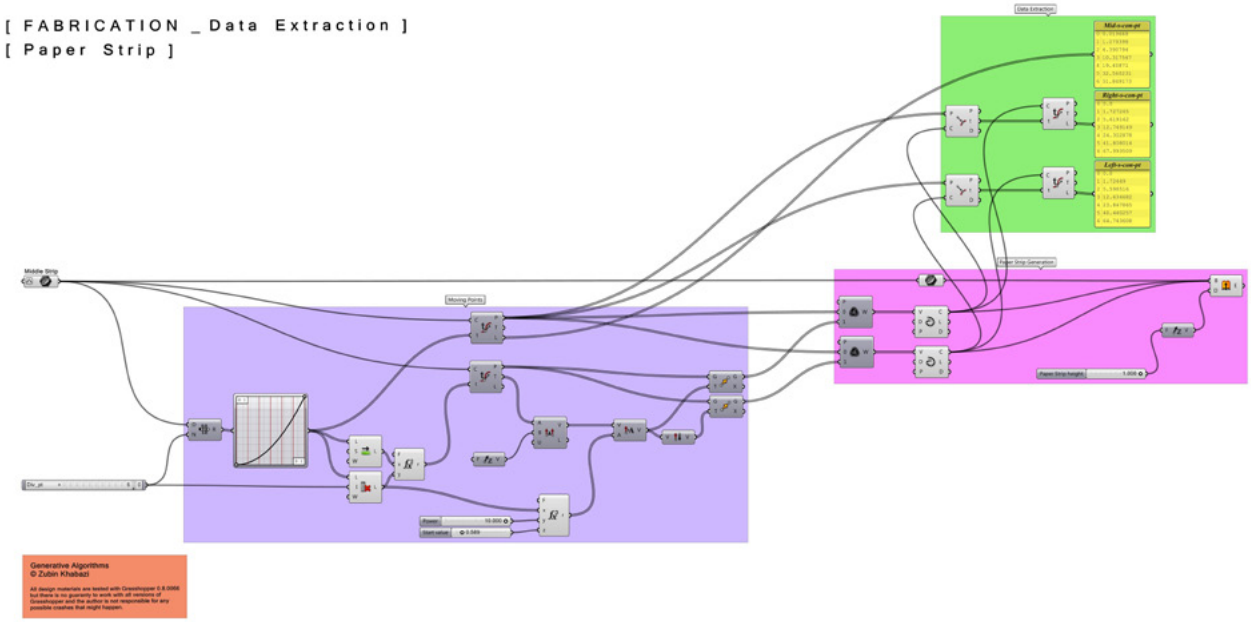


Fig.6.2. The [Paper Strip] definition has corrugated curves which are interconnected and created profile of strips to be extruded (GH file is available for explorations). The green part is where we want to focus for data extraction.

Data Extraction

By doing a simple paper model test, I know that I need the position of connection points on strips and it is obvious that these connection points are in different length in left_strip, right_strip and middle_strip. So if I get the division lengths from Grasshopper I can mark them.

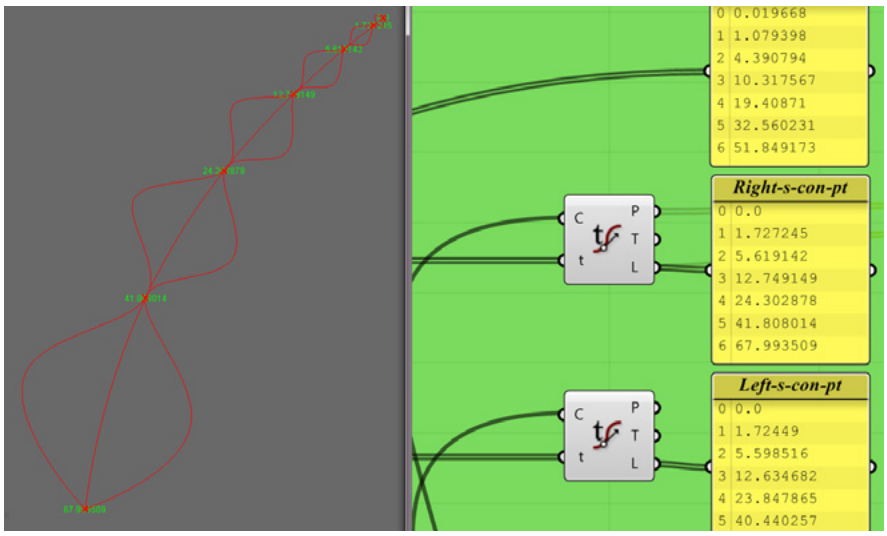
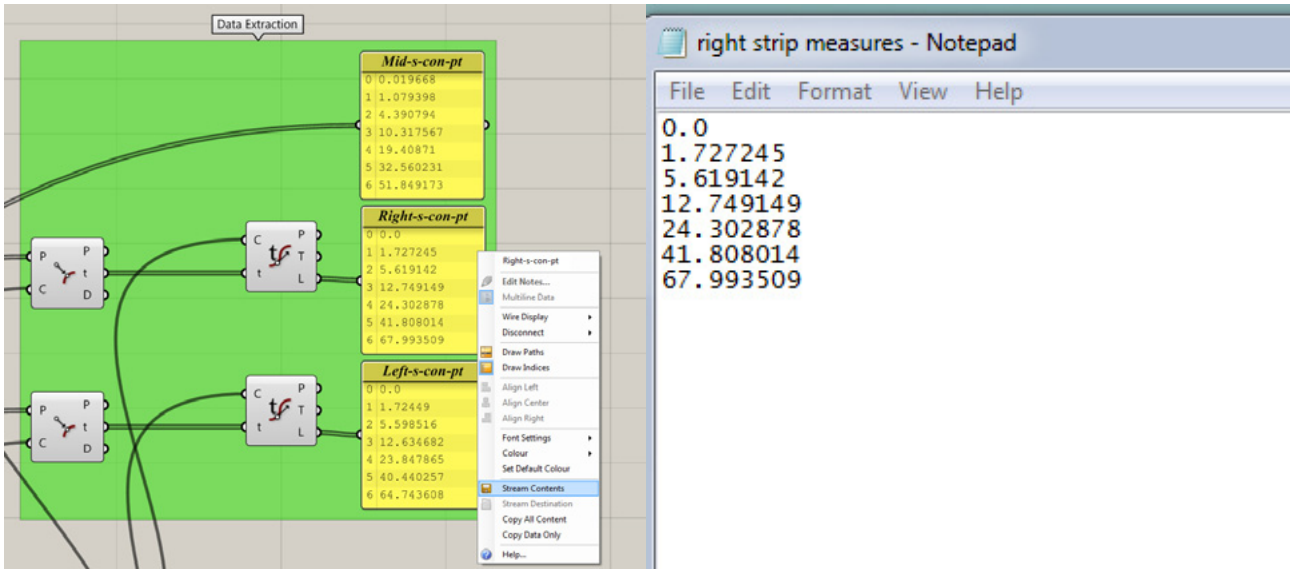


Fig.6.3. All measurements are extracted by <Evaluate Curve> and then passed to a <Panel> component. Right-click on the <Panel> component and click on the 'Stream Contents'; By this command you would be able to save your data in different formats and use it out of Rhino/Grasshopper. Here I will save it with .txt format to be able to use it in Microsoft Excel, Google Documents, Open Office or other spread sheets.



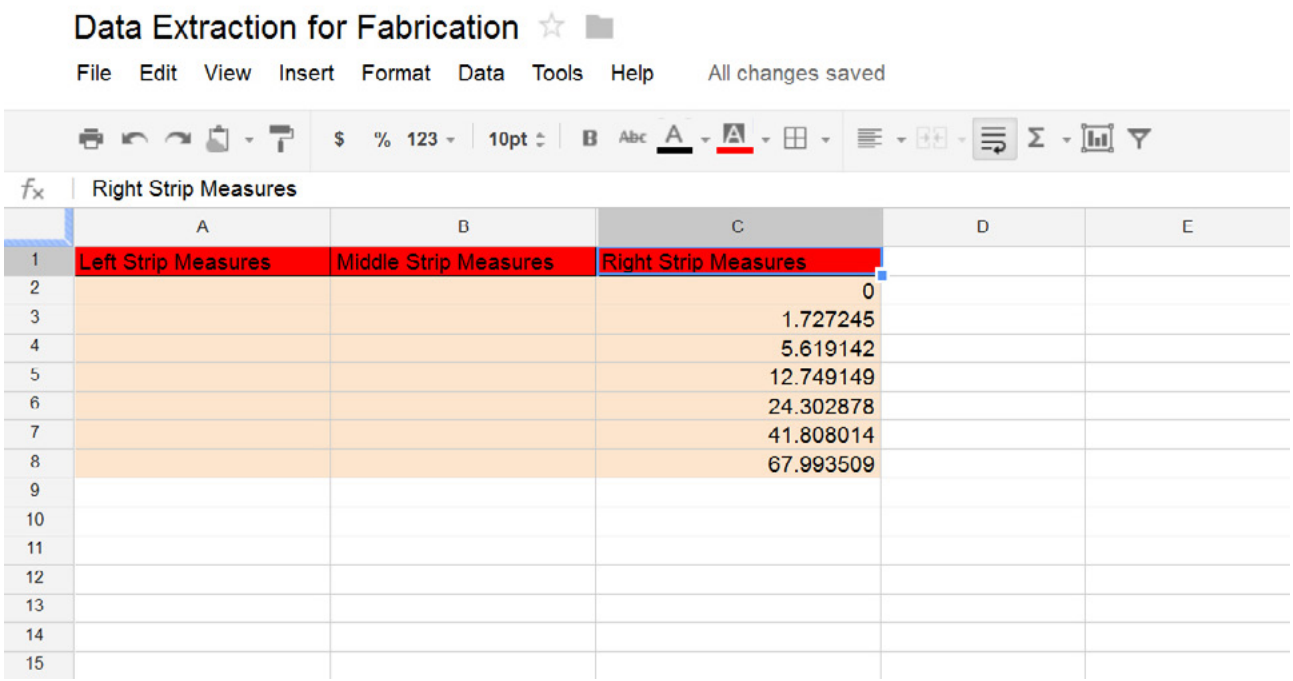
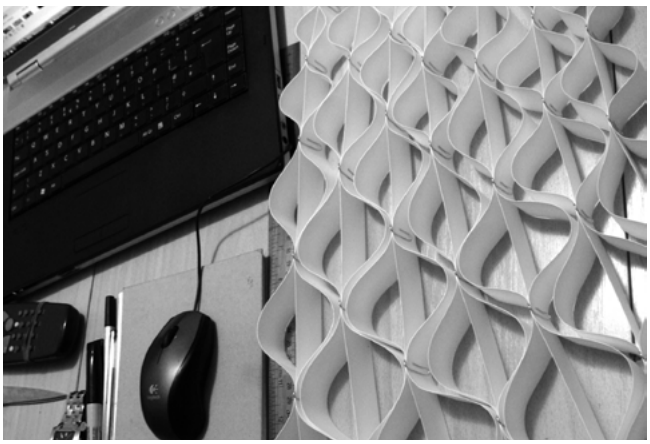
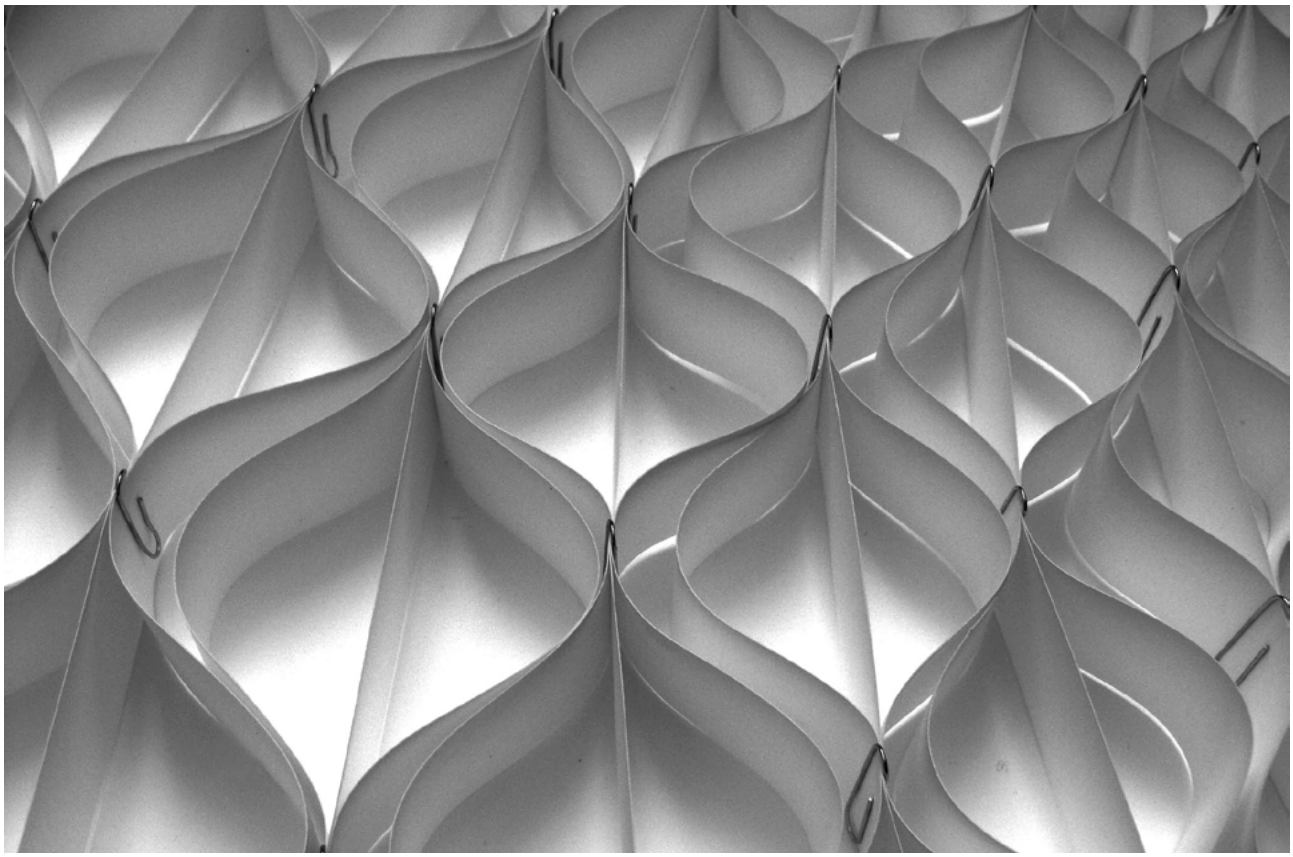
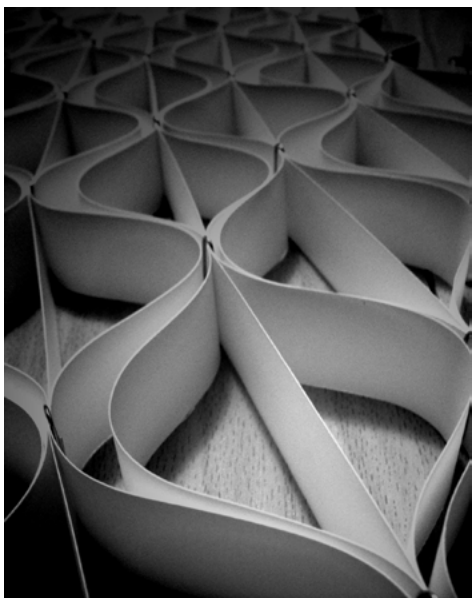


Fig.6.4. Imported data in Google Docs.








26

CHAPTER SEVEN

INTERSECTION

Chapter Seven
Intersection

One of the possible techniques in fabricating complex, free-form objects is to divide them by sections (contours) and fabricate these sections by cutting based fabrication technique like using laser cutter and flat sheet materials. In this case, various sections of the object should be cut and then assembled together. Assembly might include mounting these section layers on top of each other or connecting them by other elements. For example there could be section layers on different directions as well to interlock with the first layer and generate a ribbed structure.

The crucial point with intersection is that it approximates the free form geometry. As much as layers of sections become dense, the approximation would be more close to the original object but includes lots of materials and weight in fabricating. An optimum point should be found in order to set the number of intersection layers for each object.

Contour Lines of a Free-Form Object

Here a free-form object wanted to be fabricated using contour sections. Having various contour lines, it is possible to think about a cutting based fabrication. One can extract sections, cut them and attach them to fabricate the object.

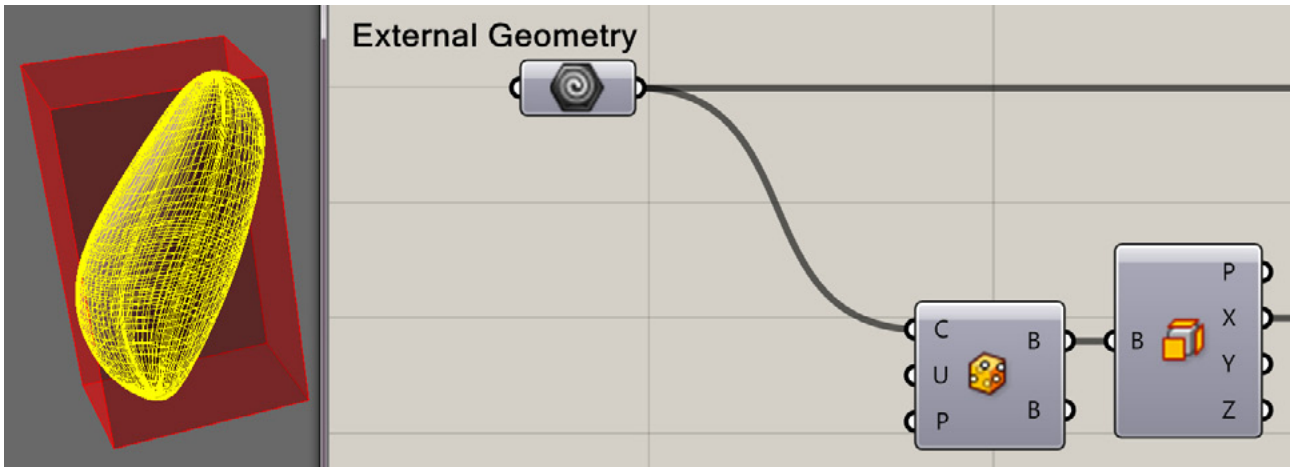


Fig.7.1. A generic geometry is added to the definition and its <Bounding Box> is used to set contour sections based on its domains.

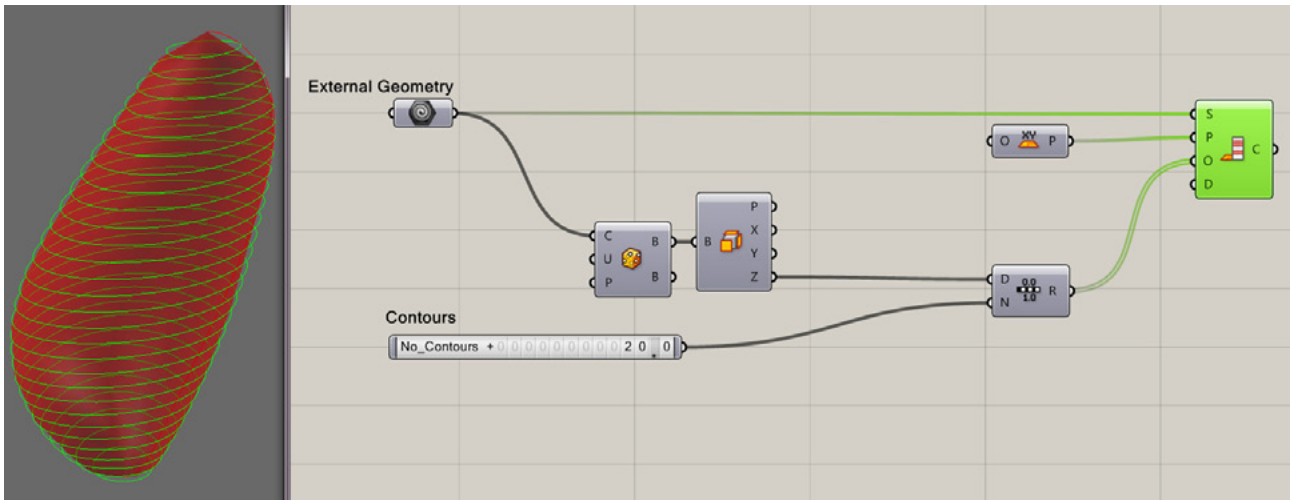


Fig.7.2. Contours are generated by a <Contour(Ex)>, using the divided domain of the bounding box.

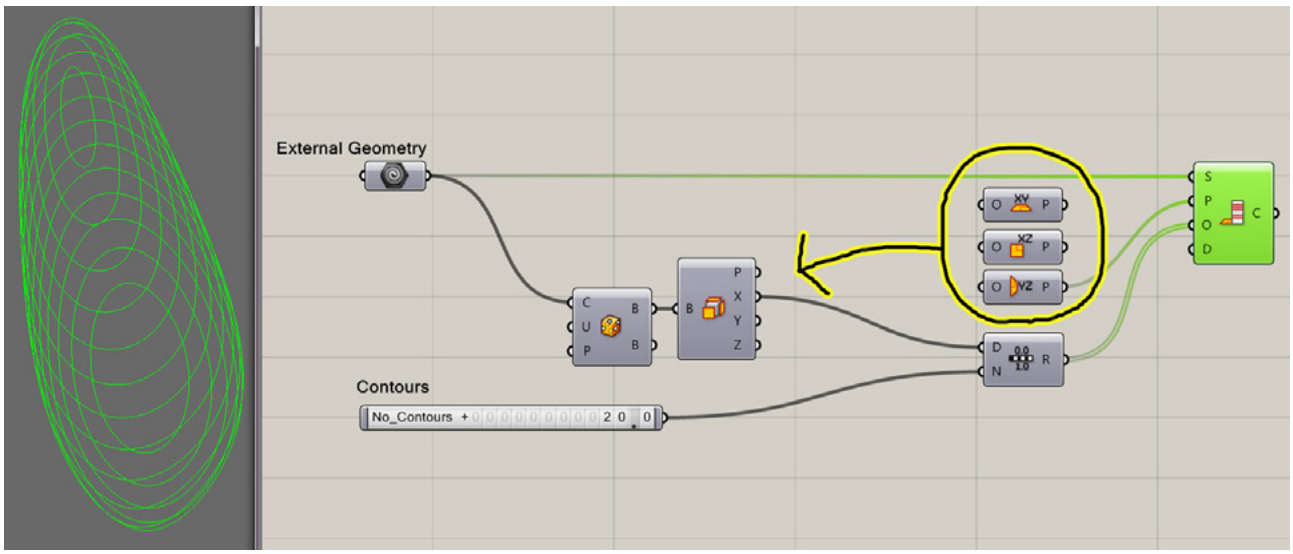


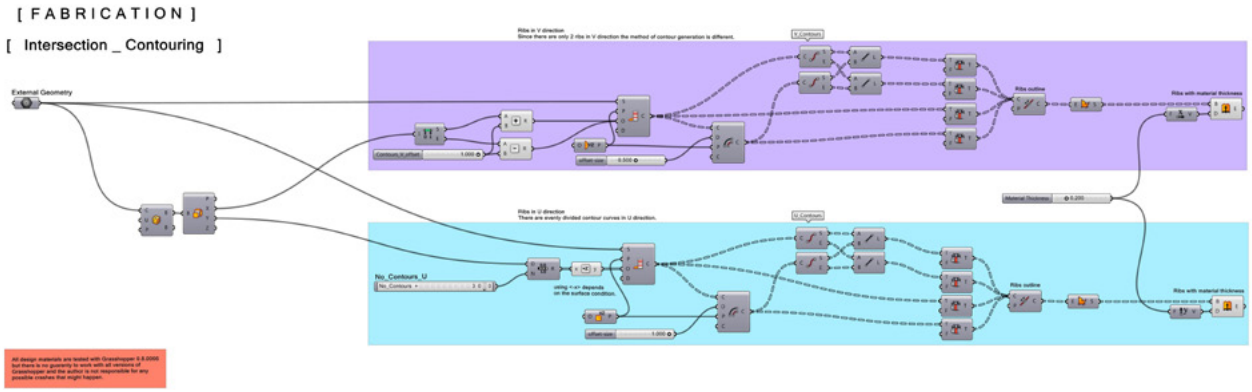
Fig.7.3. It is possible to generate contours in various directions based on our needs and object's condition. While using other planes to extract contours, it is important to set the appropriate domain of the bounding box.

7_1_Cutting based Fabrication

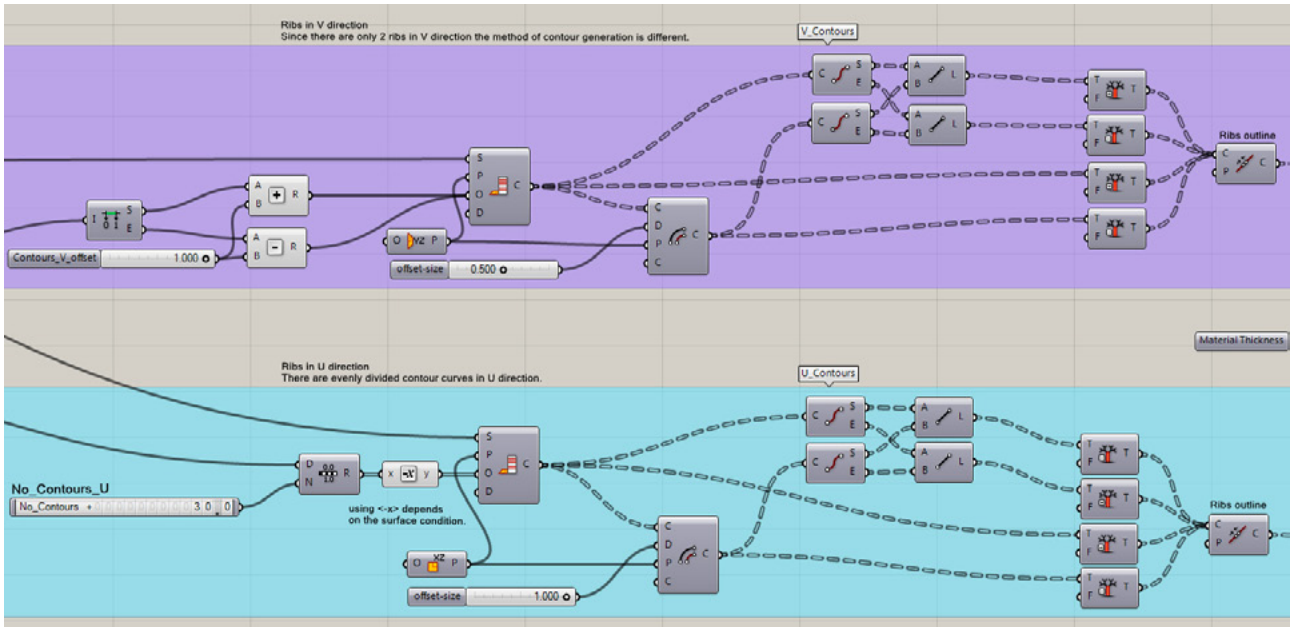
Ribbed Structure

The idea of cutting based fabrication using sheet materials is very common to fabricate complex geometries with simple techniques. Cutting based methods suit objects that built with developable surfaces or folded ones which could be unrolled or flattened to be able to cut out of a sheet material. It is also suitable for other complex geometries that could be approximate by section contours like ribs in the following examples.

I decided to build a physical model of a free-form surface to show some experiments with intersection technique to approximate the shape of surface.



The Algorithm comprised of two main, but parallel parts: both parts are generating ribs for the fabrication of the surface but in perpendicular directions. Although generating contour sections is easy, finding the intersection points to generate joints of the structure is the most important part in the design-fabrication algorithm. The idea is to provide bridle joints for the intersecting ribs.



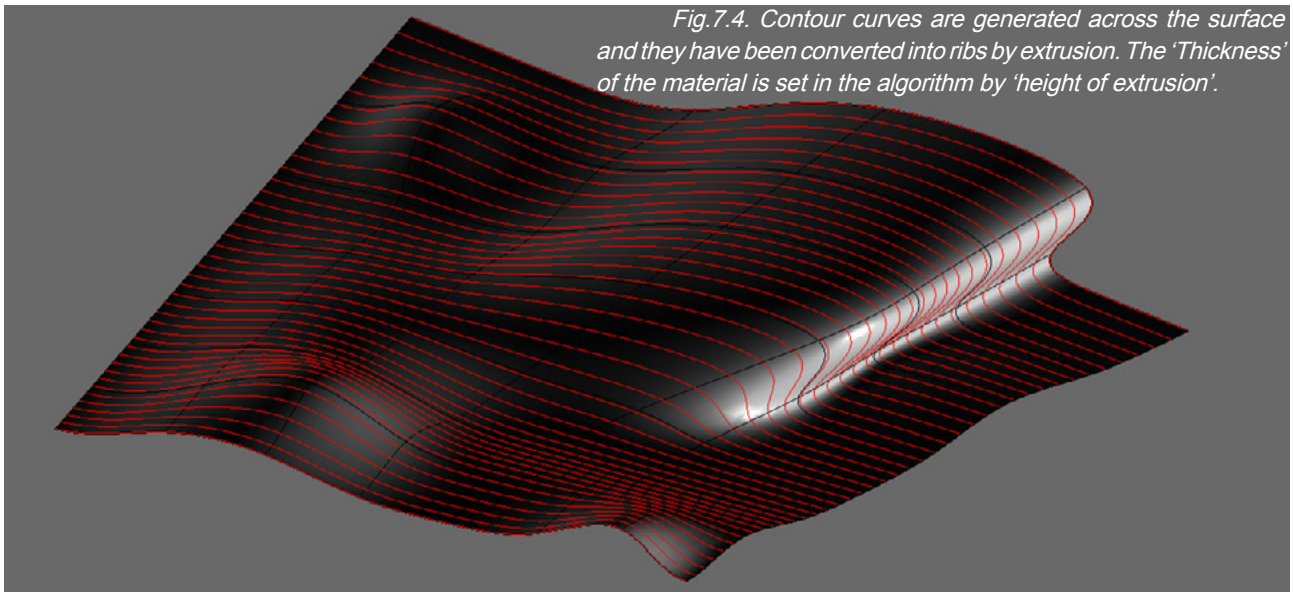
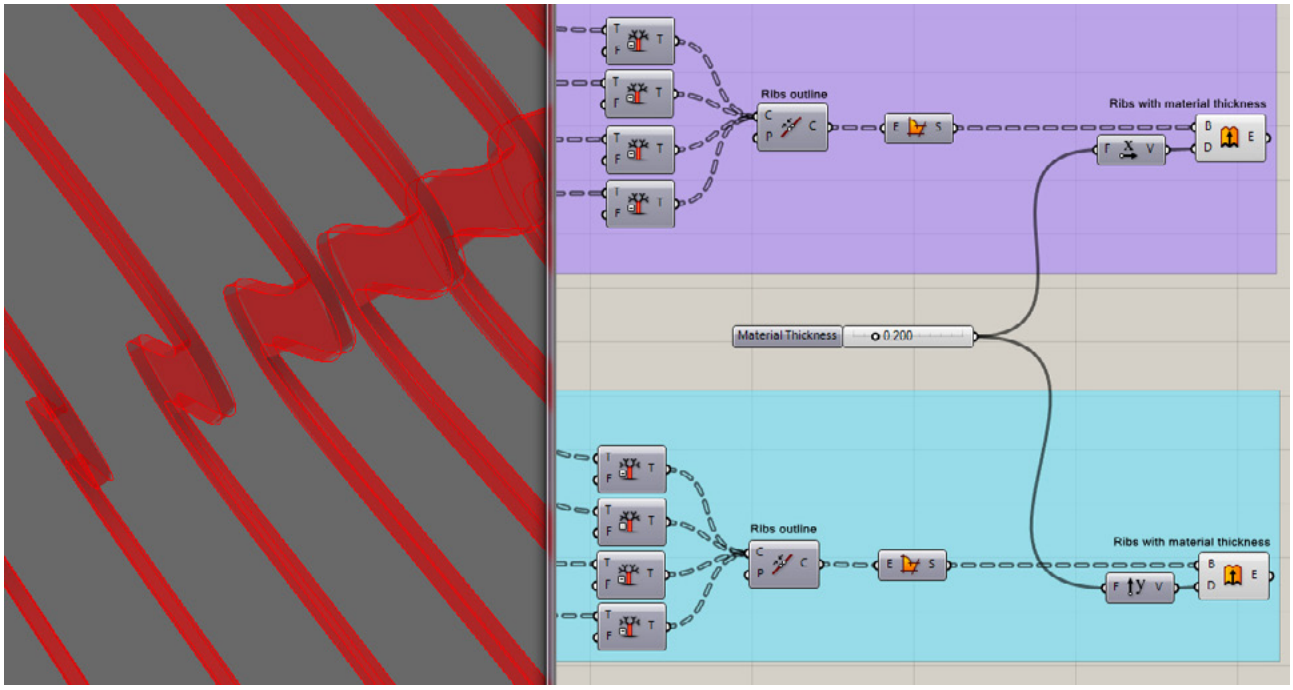


Fig.7.4. Contour curves are generated across the surface and they have been converted into ribs by extrusion. The 'Thickness' of the material is set in the algorithm by 'height of extrusion'.



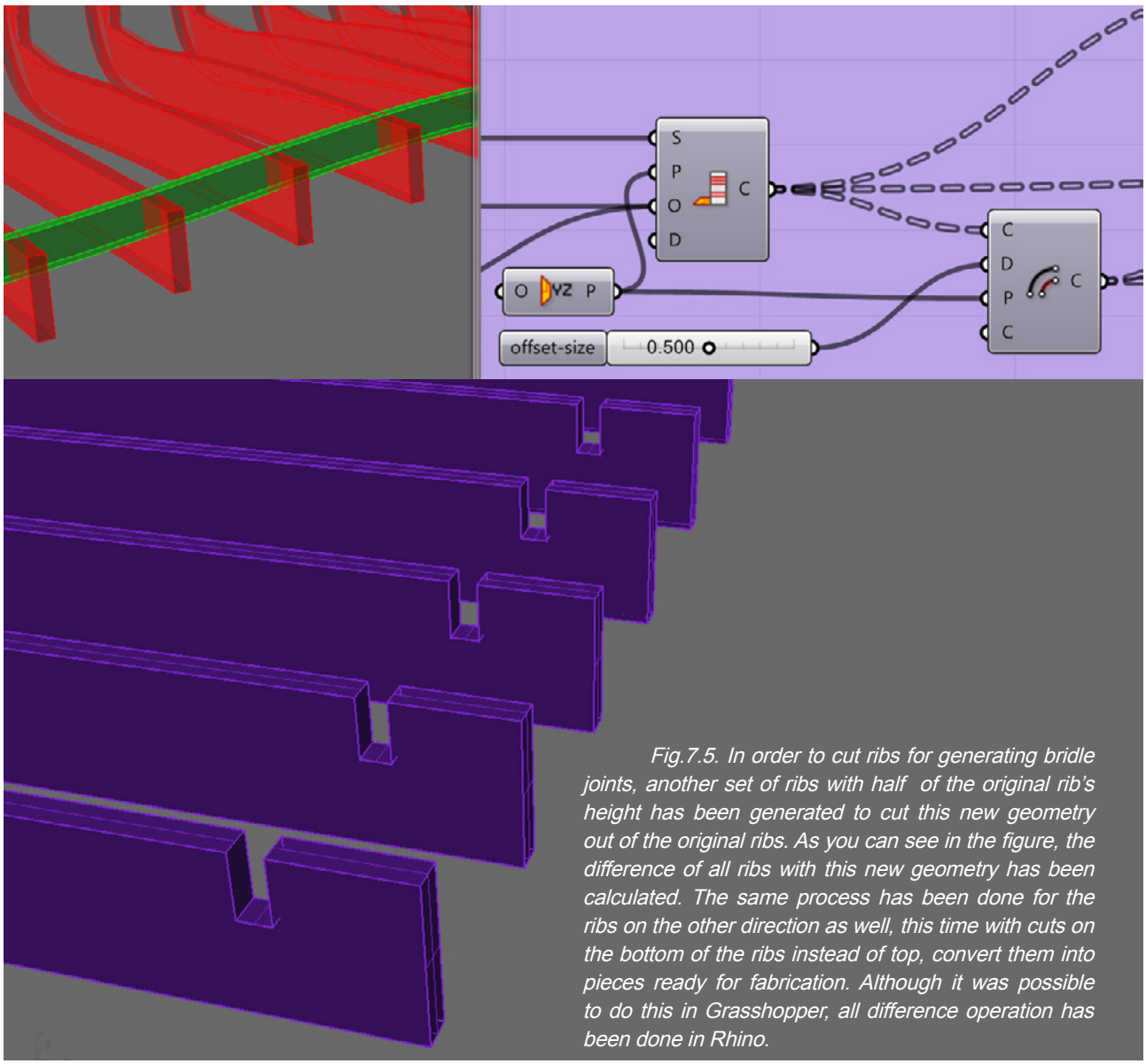


Fig.7.5. In order to cut ribs for generating bridle joints, another set of ribs with half of the original rib's height has been generated to cut this new geometry out of the original ribs. As you can see in the figure, the difference of all ribs with this new geometry has been calculated. The same process has been done for the ribs on the other direction as well, this time with cuts on the bottom of the ribs instead of top, convert them into pieces ready for fabrication. Although it was possible to do this in Grasshopper, all difference operation has been done in Rhino.

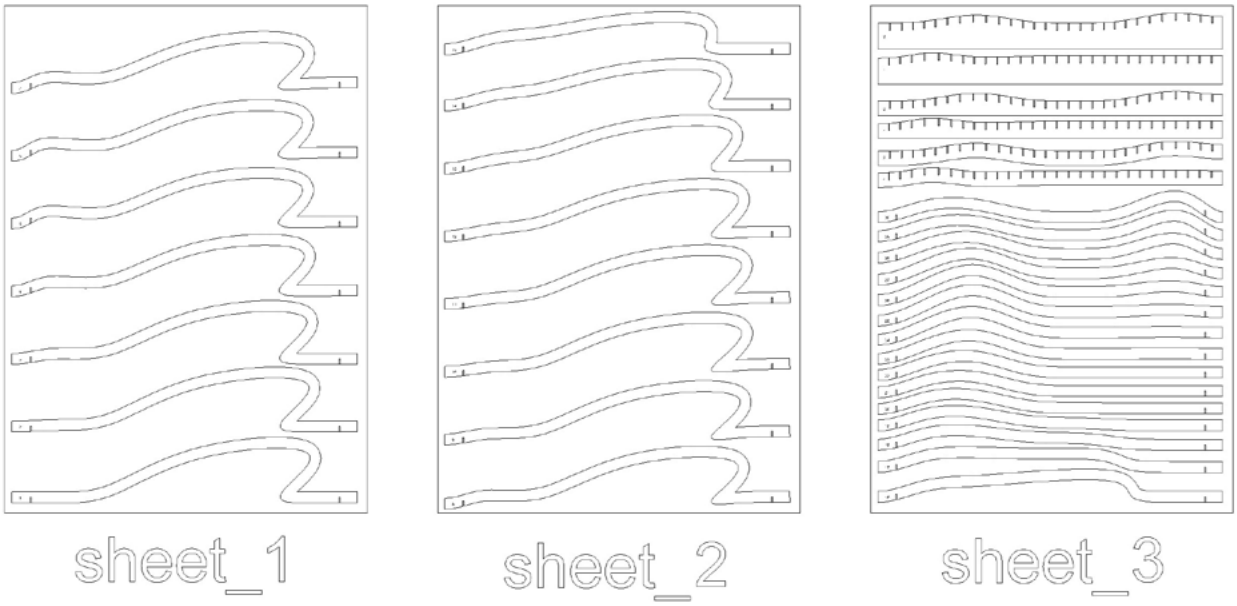
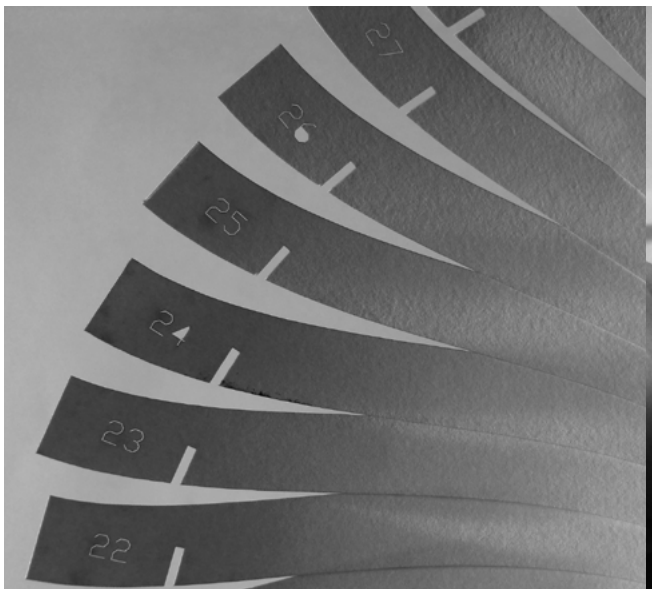
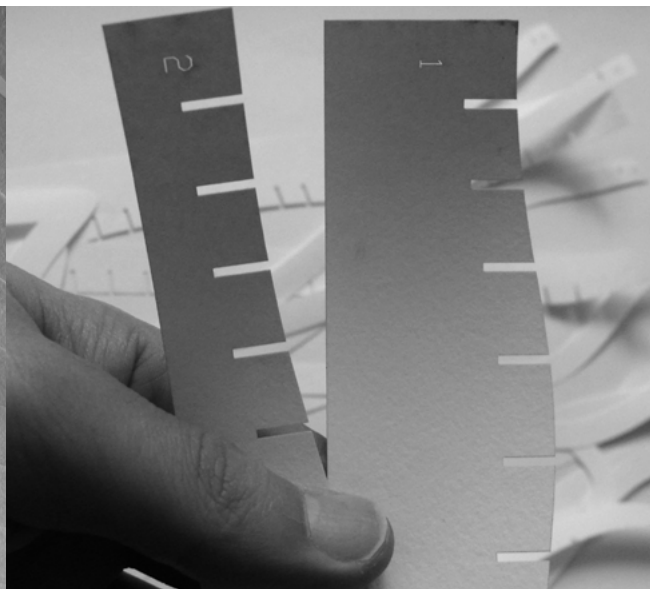
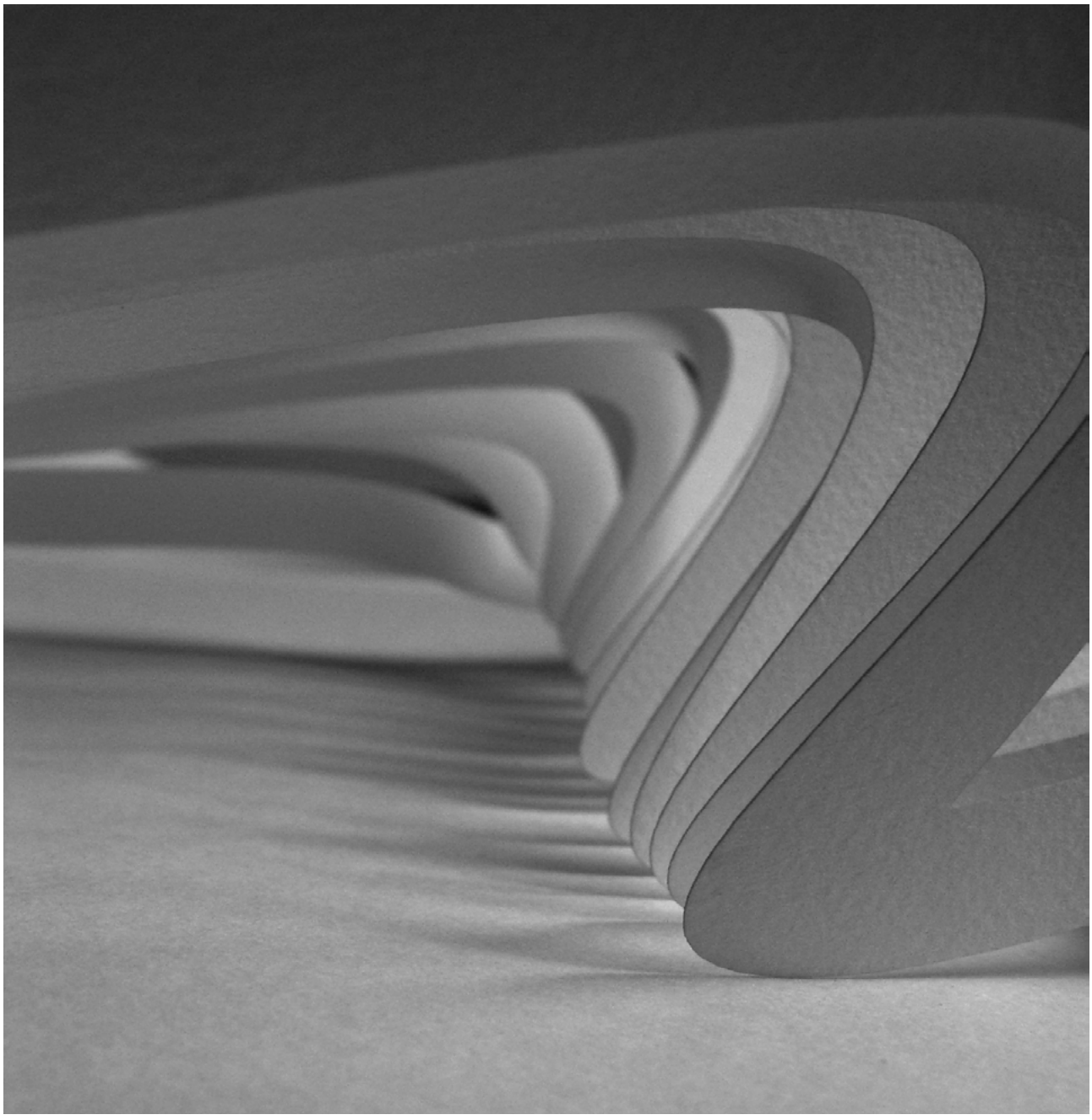


Fig.7.6. Nested Pieces on sheets, ready for cutting. We will discuss about nesting later on.


27
26
25
24
23
22


2
1



CHAPTER EIGHT

PROJECTION

Chapter Eight
Projection

One of the classical aspects of constructing architecture and now fabrication is to provide proper drawings of 3D objects. Using Unfolding, Unrolling, Unflattening or Tessellation, Intersection or any other fabrication technique, there might be some pieces (building blocks) that should be converted into drawings for further manufacturing/making processes like cutting out of sheet materials. This is somehow relevant to one of the key aspects and branches of geometry which is called Descriptive Geometry that studies the representation of 3D objects on 2D drawings. The process of this representation called Projection. While standard architectural drawings like plans and sections are Parallel Projections, there are also Perspective Projections for visualization purposes, Orthogonal and Axonometric Projections and Non-linear Projections as well (Pottman, et al, 2007). CAD software like Rhino provide real-time projection in their viewports.

What we might need in this chapter is to deal with some methods to provide 2D drawings of various elements of a design product (a 3D object). Since there are so many techniques that tessellate a free-form object into smaller pieces, or there are components which are populated along a surface over a space, one might need to provide drawings of these pieces to fabricate them. In this case, these objects should be projected onto a flat surface to provide drawings for fabricating with Laser Cutter or CNC.

There are not simple universal techniques to accomplish these projection or nesting tasks for fabrication. One might pursue different approaches based on the properties of the object which is working on. Here in this chapter a possible example would be presented as an idea and other methods and techniques should not be so much different.

8_1_Projection

Grasshopper provides some components for projection purposes. While <Project> from Transform>Affine can simply project any 3D object onto a plane, a <Project> from Curve>Util will project a curve onto any surface. There is also <Map to Surface> which might be helpful in this sense.

But here in our design experiments we cannot always use these available components. Since we are dealing with complex objects, it is important to have clear drawings and proper projections from various sides of them which needs composite projections of poly-surfaces together. We are also facing the problem of projecting so many objects not one, and providing a plan or section drawing for each one is an issue. We might approach projections with our own techniques.

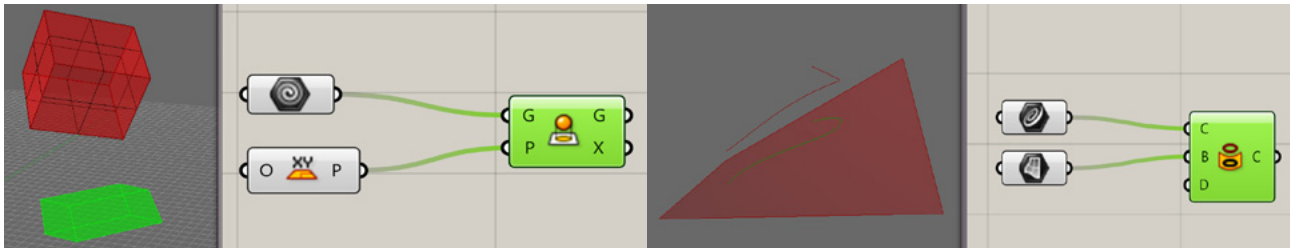


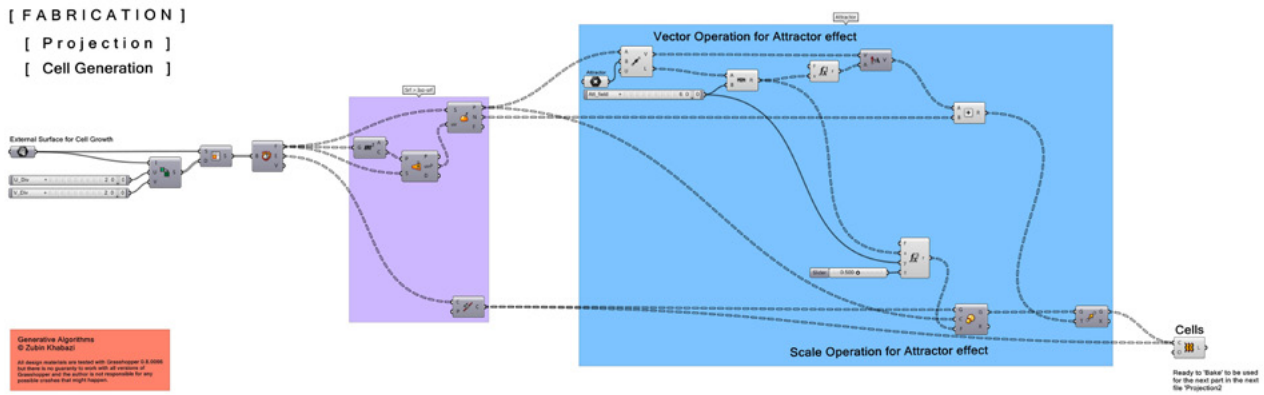
Fig.8.1. Projections in Grasshopper

The Red Porifera Project

In design space, there is a surface which is aimed to be subdivided and some truncated pyramids are going to be generated on top of each sub-surface. These truncated pyramids are pointed towards an external point and for those which are closer to the point (Attractor), they want to get higher. The truncated pyramids would be generated by a rectangle at the bottom and this rectangle would be moved to the top side as well to finally generate a loft surface by both sides. The size of top rectangle would also be affected by the attractor, become smaller when closer. At the end, the idea is to fabricate it with flat sheet materials and cutting technique, and all of the process should be accomplished using Grasshopper.

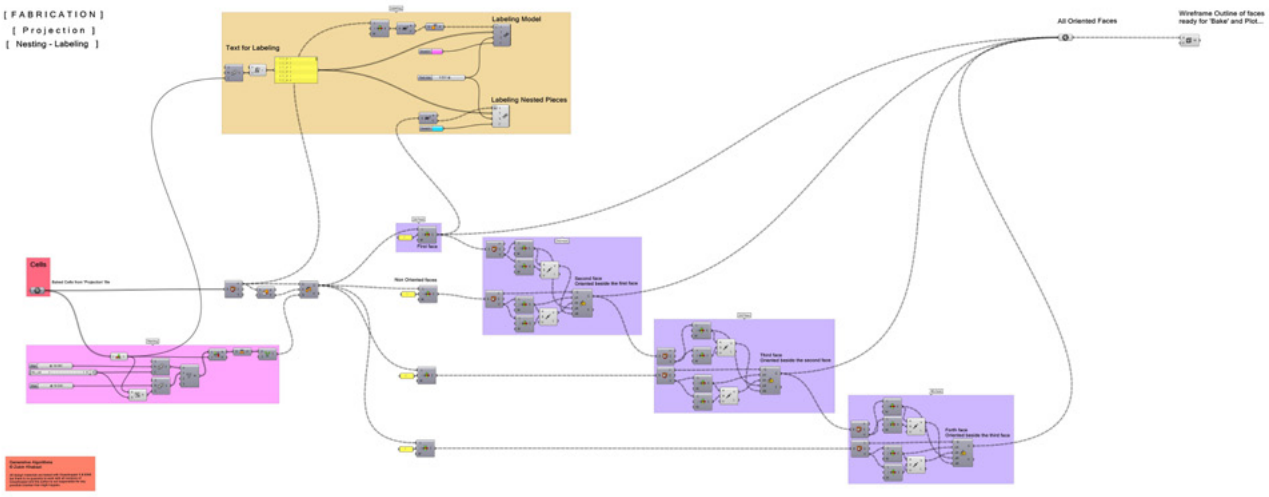
Design Algorithm

There are two main parts in the project. In the first algorithm all truncated pyramids which I call them Cells are generated. The technique has nothing new and it is possible to go through the file to see the definition. The only point is that all cells should be 'Bake'd for the next step. It is also possible to make it in one Grasshopper definition but I personally prefer to do it separately, reducing the chances of crashes or heavy files with low performances.



Fabrication Algorithm

The second file is where it is aimed to fabricate all those cells. Cells are unfolded and projected onto a flat surface. Since cells are poly-surfaces with 4 faces, the most important part is to unfold them and then prepare them to cut, means all adjacent cell faces should be attached together. The middle part of the algorithms is responsible for that purpose.



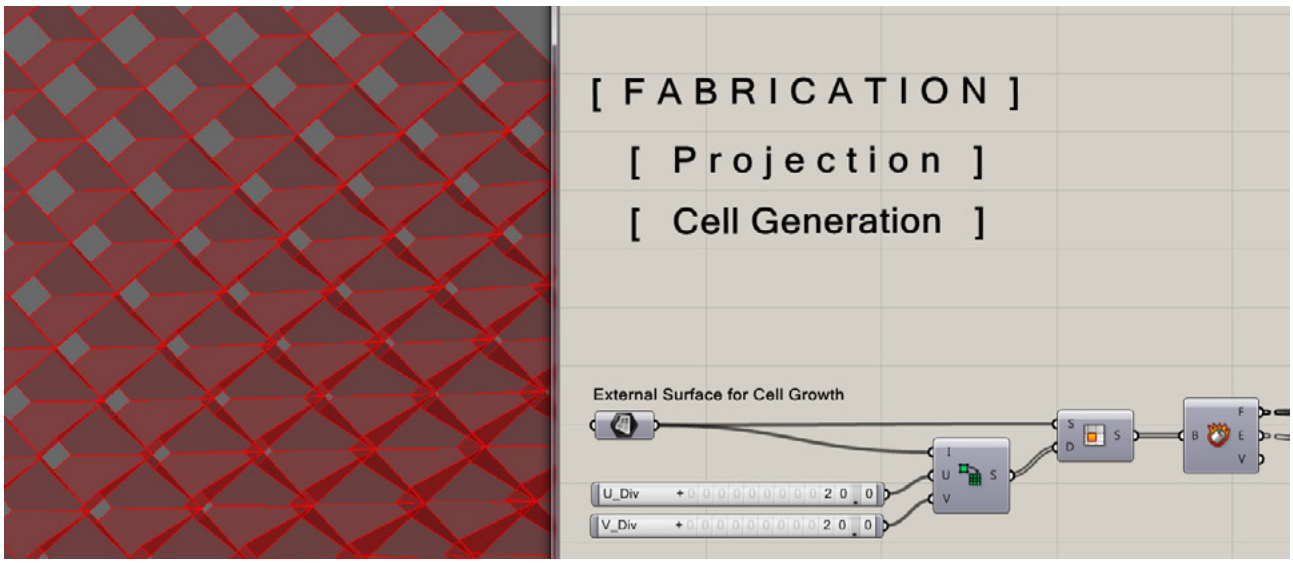


Fig.8.2. A generic NURBS surface is subdivided and the outline of these sub-surfaces has been used for pyramid generation.

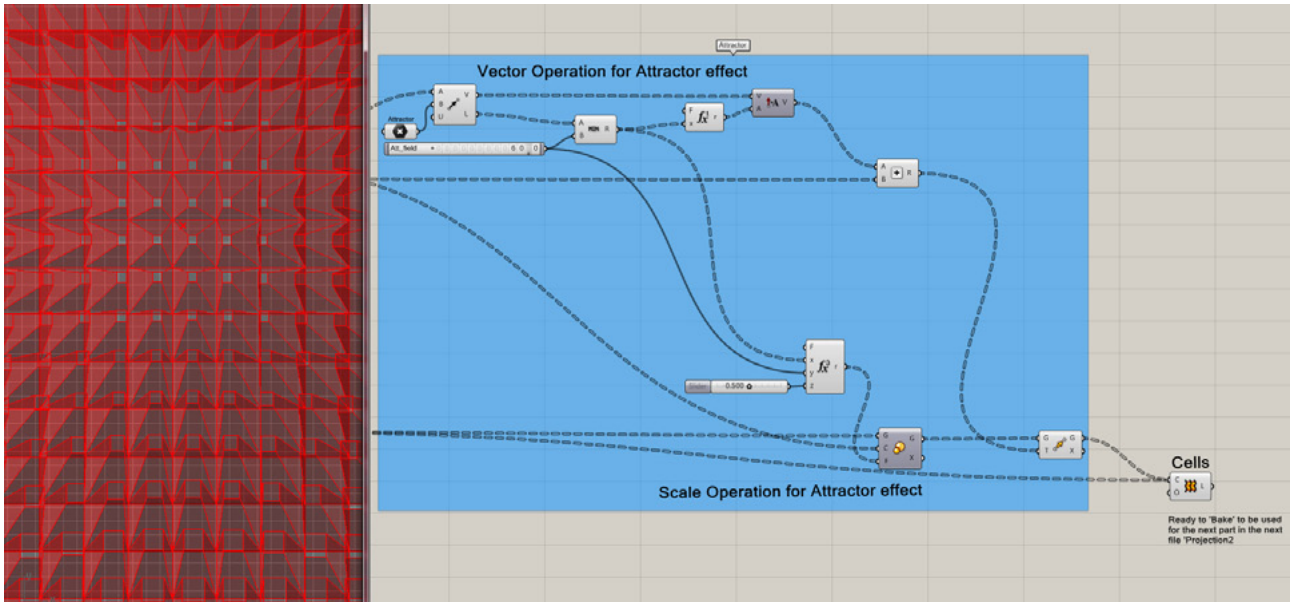


Fig.8.3. Attractor point affects the height and size of openings of cells.

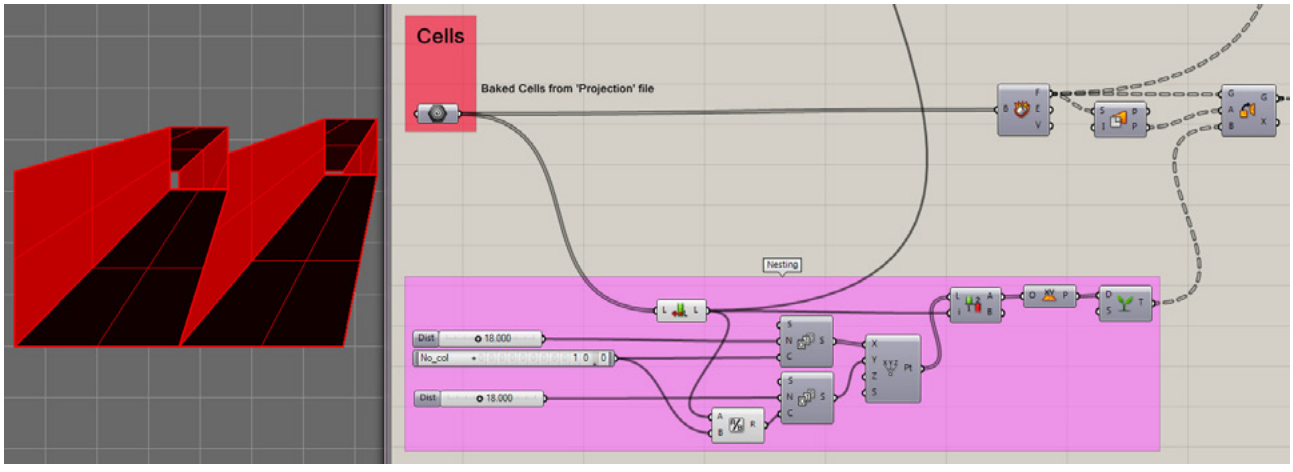


Fig.8.4. Cells should be Baked for further processes of the Fabrication Algorithm. The start point of the Fabrication algorithm is to introduce all cells as input data.

8_2_Nesting

While fabrication of the project encompasses cutting of several small objects, all these objects should be unrolling and then organized in small flat sheets in order to cut by machines. The process of organizing cutting pieces on sheets and arriving at an optimum point of using the surface area of the sheet is called nesting. This part needs both algorithmic and manual drawing techniques.

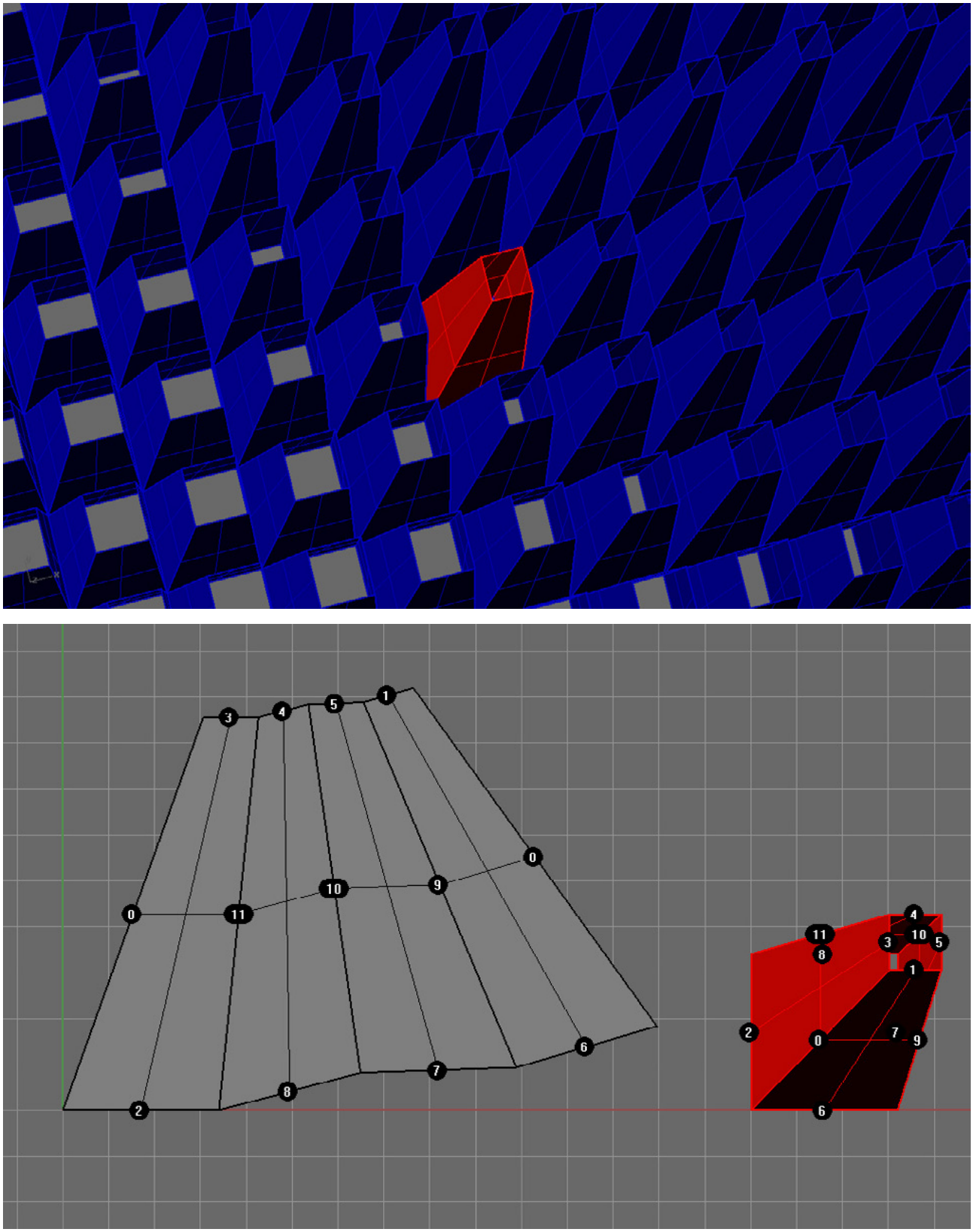


Fig.8.5. The idea of Fabrication algorithm is to unfold each cell separately onto a flat surface so it would be possible to cut it by laser cutter and fold it back to make the cell. Fabrication algorithm should follow the Rhino 'Unrollsrf' command.

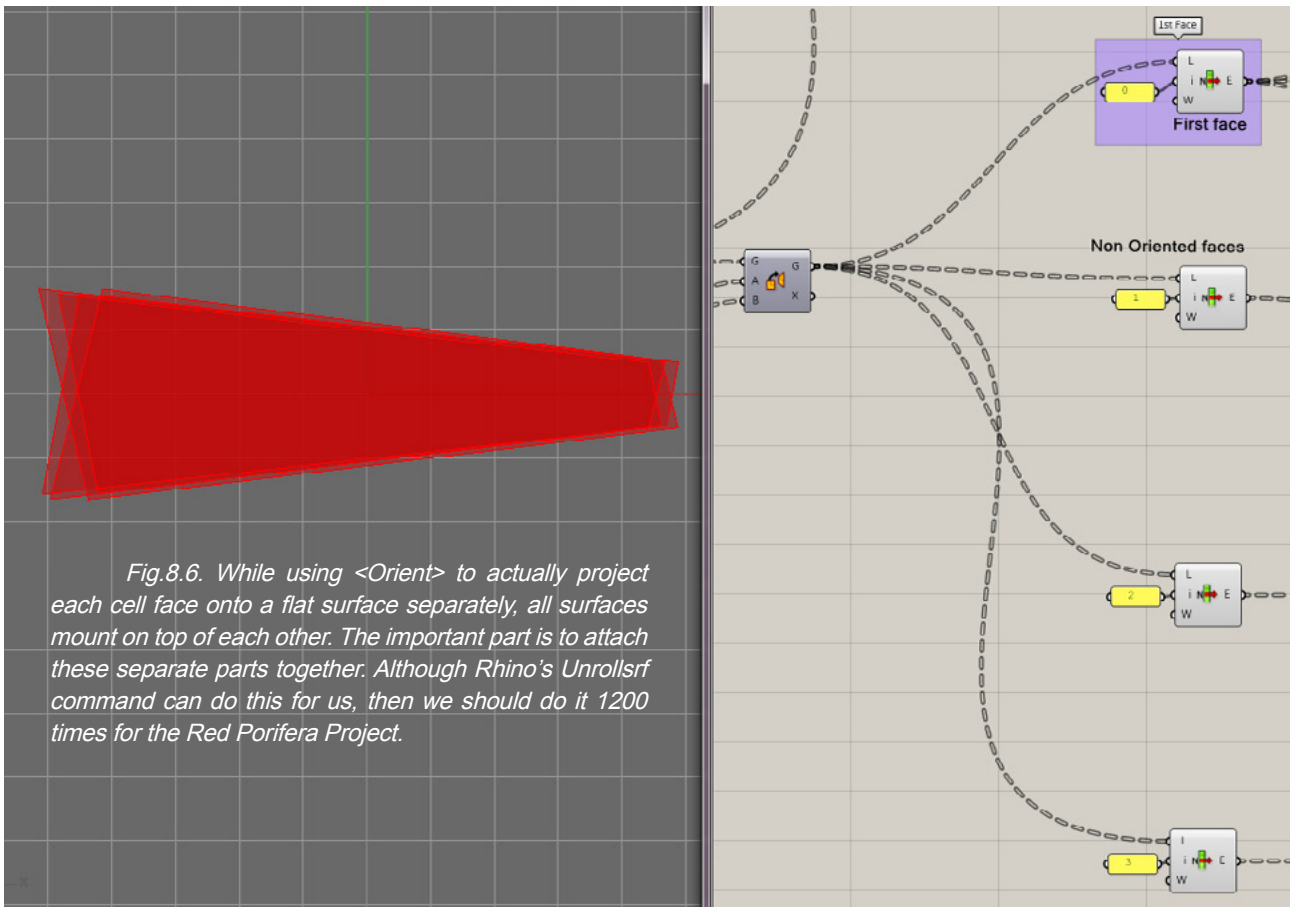


Fig.8.6. While using <Orient> to actually project each cell face onto a flat surface separately, all surfaces mount on top of each other. The important part is to attach these separate parts together. Although Rhino's Unrollsrf command can do this for us, then we should do it 1200 times for the Red Porifera Project.

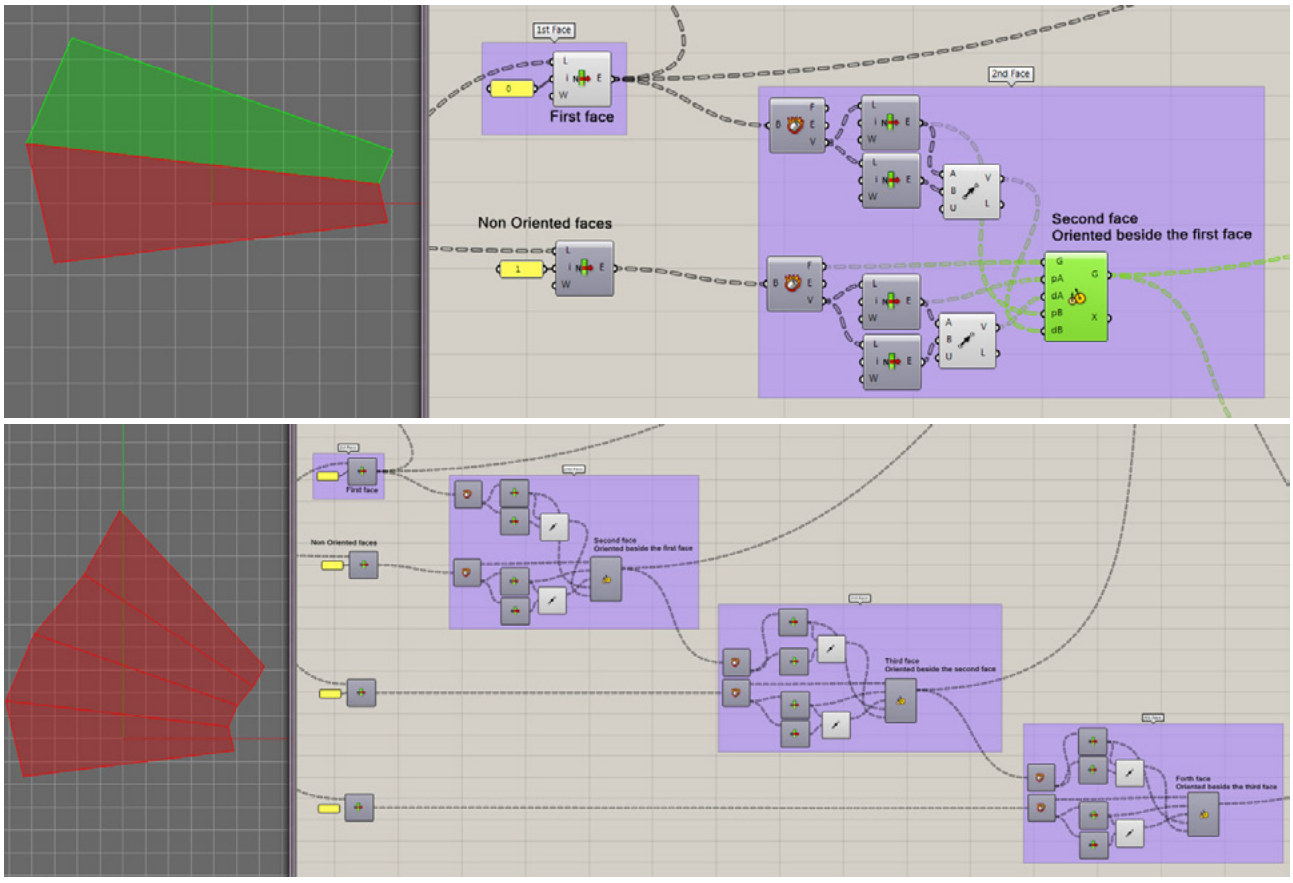


Fig.8.7. So far the best way is to select each face and then re-orientate it to attach to the previous face. The selection of points on surfaces and finding reference and target points might be a tricky business here.

8_3_Labelling

Like other fabrication techniques with multiple pieces, it is crucial to label all pieces for further addressing and ease of access. Both nested pieces and cells in the model are addressed with the same labels. Assembly plans with the labelled pieces would be needed during assembly phase.

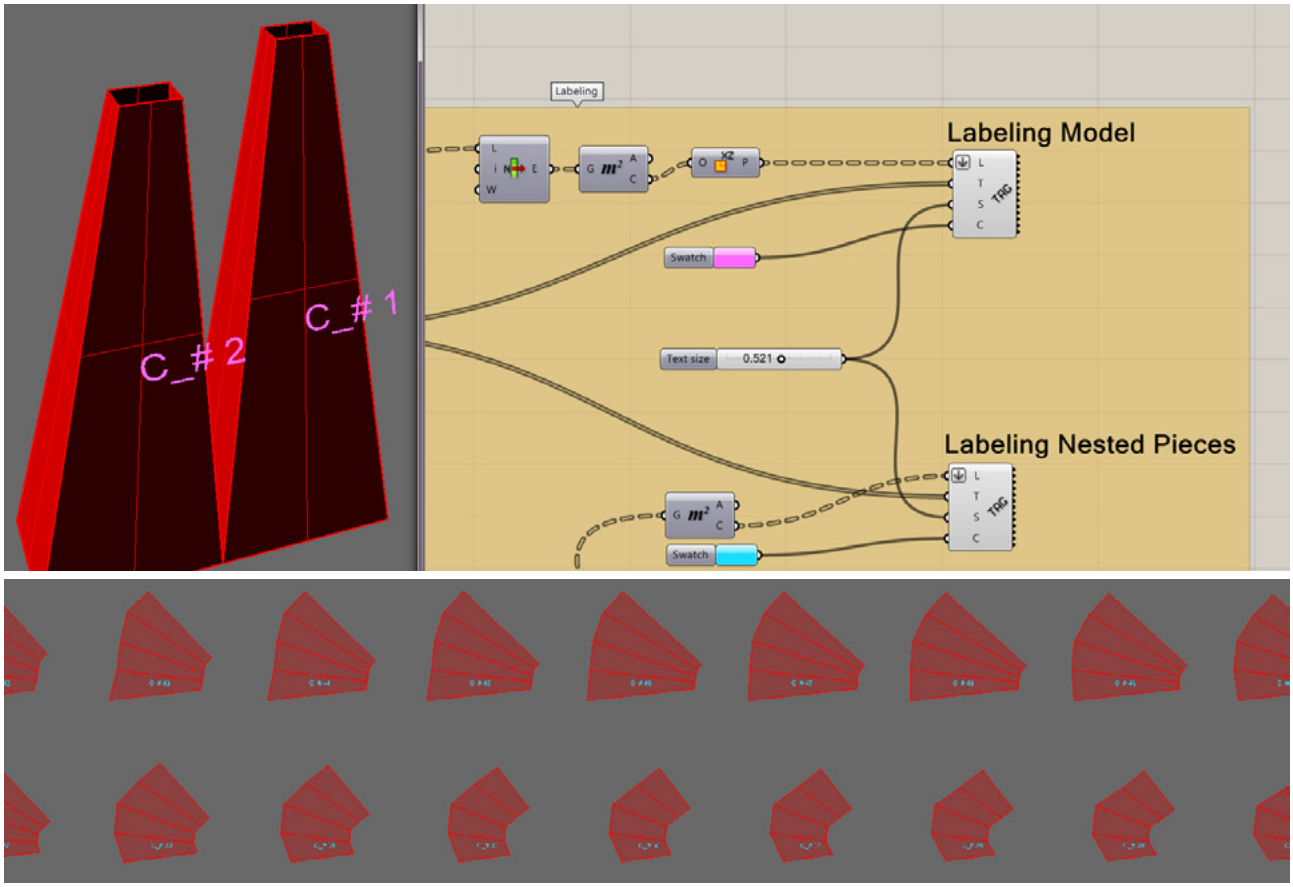


Fig.8.8. Labelling

Fig.8.9. Cutting and Folding process

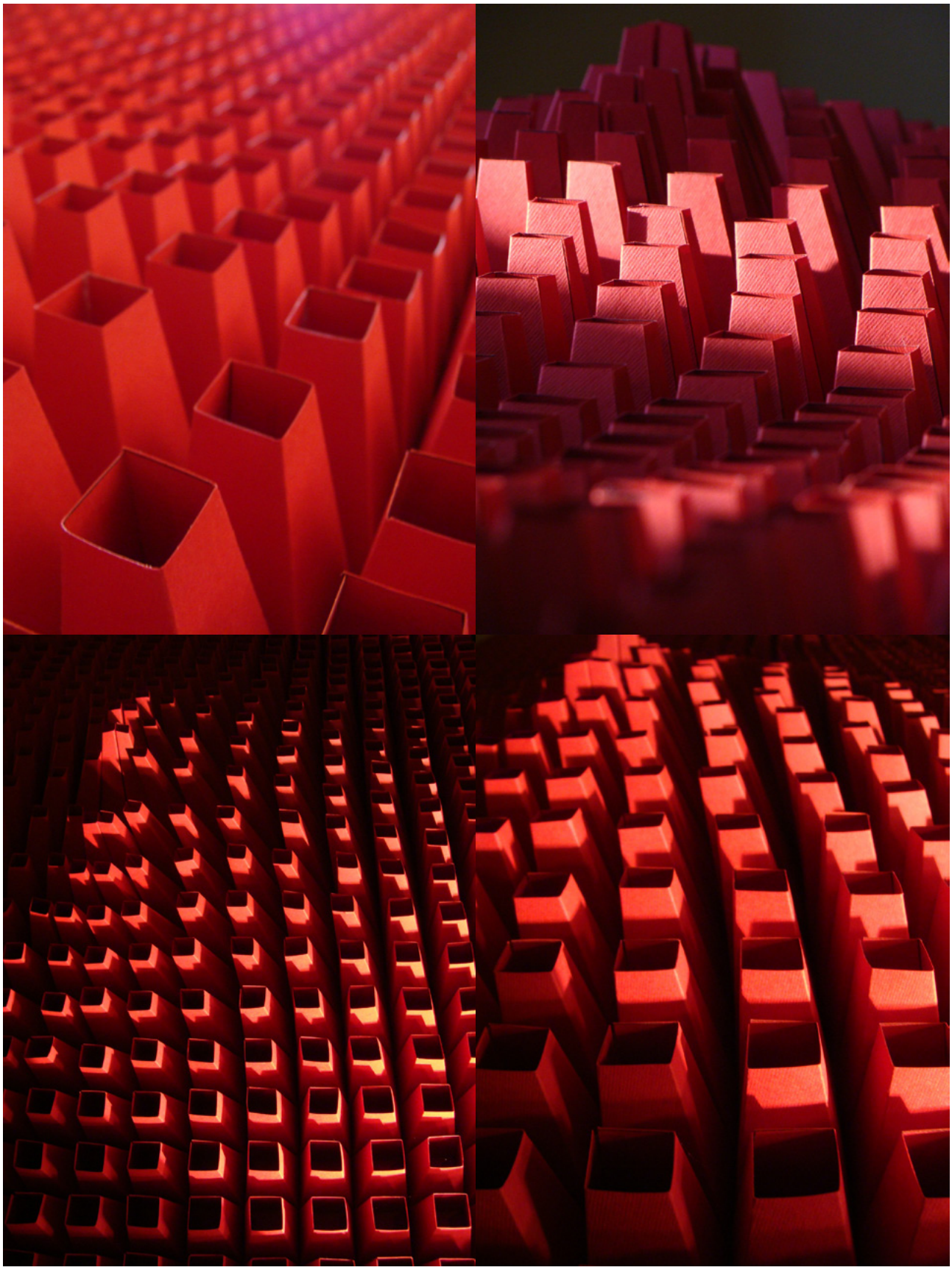
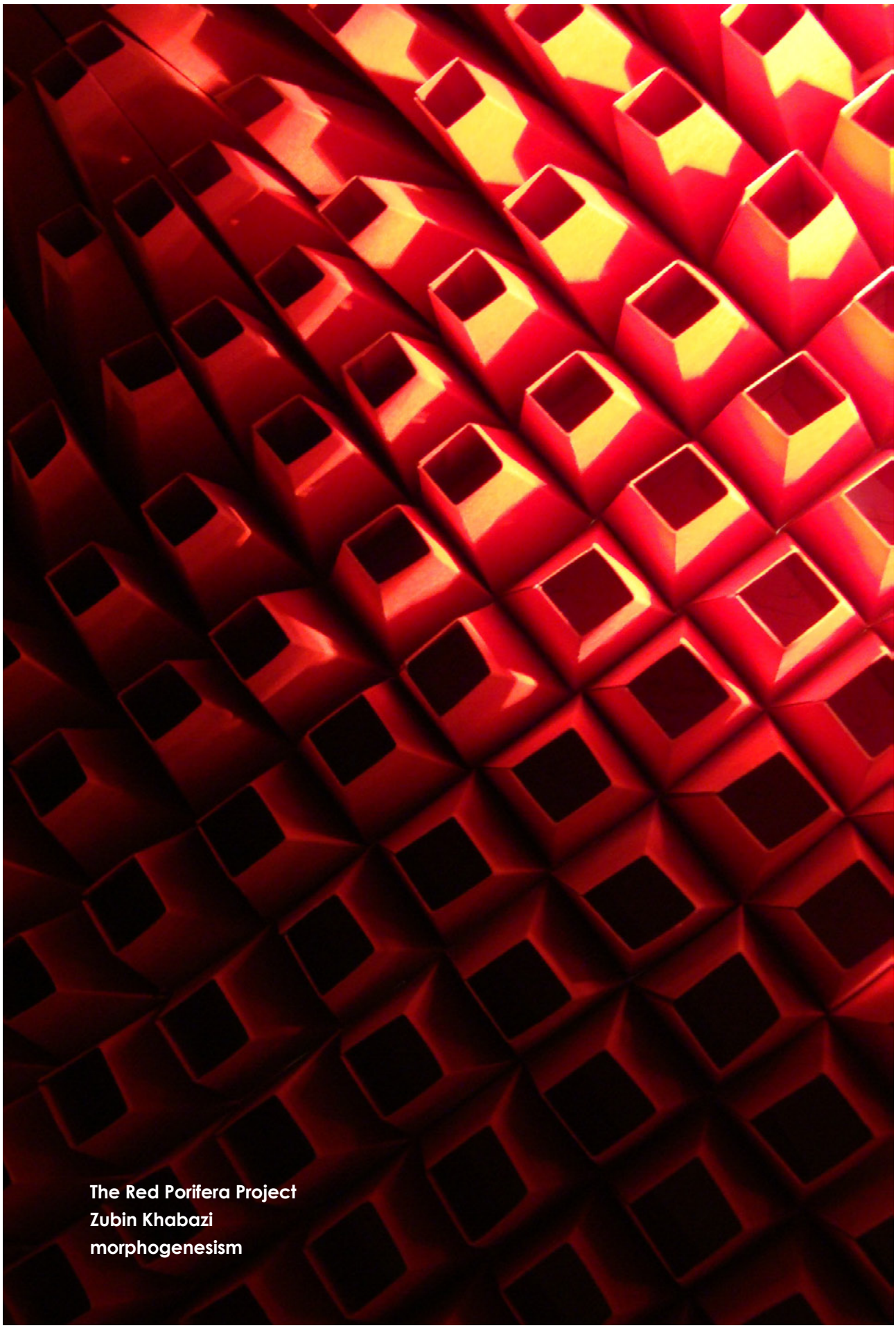


Fig.8.10. Assembly of cells


The Red Porifera Project
Zubin Khabazi
morphogenesism

PART THREE
ANALYSIS

CHAPTER NINE

FREE-FORM SURFACES

Chapter Nine
Free-Form Surfaces

9_1_Digital Analysis

It is now quite a while that contemporary architects deal with 'Digital' as a prominent paradigm in design practice. Introduction of CAD started to affect the design industry, first by appearing as tool but then as technique and now as methodology or even as Patrik Schumacher described, a 'style' (Parametricism). While CAD software affected the design process, CAM made it possible to realize those digitally designed objects. So digital technology and machinery started to solve the problem of construction and convert the 'Architecture for Journals' to the 'Architecture for Physical World'. Contemporary architecture progresses rapidly through development of both digital design and digital fabrication technologies but in its conversion to physical products, progresses with digital analysis tools as well.

Various software development in the field of building analysis and its performance helped designers to evaluate their products from different aspects. Structural Analysis, Building Performance Analysis, Material Behaviour, Environmental Analysis, Interior Climate Analysis and all other software packages made it possible to improve the design product, optimize its performance and rationalize its construction. By using contemporary algorithmic and parametric software packages, 'Digital Analysis' evolved into 'Feedback' loops, effectively present in the process of form generation. The design output of such processes which are informed by performance of buildings has different criteria to be criticised which are more in favour of behaviour rather than beauty. These new criteria for architectural design need their own tools and techniques. With analogy to design industry, software development evolves as a paradigm in critical thinking.

Digital Tools and Techniques for Analysis

Analysis software packages are growing rapidly. Images which show analysis by colour ranges are becoming inseparable parts of design projects. These multi-coloured images are substituting with sketches of super star architects which were trying to show the idea behind projects. These new analytical images try to draw attention towards building's behavioural capacities. Solvers and form-finding software are also becoming common. They deviate architectural design from 'form' to 'formation'. Interactive algorithms are also developing and computational interfaces, made it possible to interact with computer and digital tools more efficiently and effectively. Human needs and necessities which were hard to compute and convert to data are captured by those interfaces and techniques such as Interactive Evolutionary Algorithms try to implement human evaluation in computation to generate fitness functions which encompass artistic or behaviour preferences.

In this part of the book, the journey of Generative Algorithms would continue with utilization of tools and techniques to implement some analysis methodologies and feedbacks in design process. It is important here to convert analysis tools, from post-design processes into real-time factors which could affect the design process. This will increase the buildings behavioural capacities in order to push the design product towards a more sustainable one as well.

9_2_NURBS Surfaces

As mentioned before, designers who utilize algorithmic architecture and CAD systems are tend to use free-form surfaces in their design products. Since these technological tools made it possible and somehow easy to control the geometry of complex free-form curves and surfaces and extract relevant data for drawing and fabricating such surfaces, this tendency becomes logical. Some believed that modernist architecture were restricted to the geometry of Euclidean objects but now we have advanced tools and techniques so we have to get out of the box and use these free-form objects in our design.

We already have some design experiments with NURBS surfaces. The fluidity of these surfaces is really interesting for designing contemporary objects. NURBS are among the various types of free-form surfaces like B-Spline or Bezier Surfaces or Meshes. They have internal curves that control the shape and curvature of the surface and also control points across these curves which have weight and this weight can cause the power with which the point pulls the surface towards itself.

NURBS surfaces have a parametric space with UV coordinates, so a positive U and V direction could be imagined in their edges starting from one corner with UV=(0,0). NURBS surfaces also have direction, means they have front and back. Normal vectors of a surface are always directed towards the front side. One can 'Flip' the surfaces' U and V directions and also the Surface's Normal direction.

It is possible to manipulate and edit the shape of a NURBS surface by editing and transforming its control points. A NURBS surfaces has a degree (3, 5,...). Based on the degree of a surface, its manipulated control points pull surface towards themselves and generate the overall shape of it.

While there are various components in Grasshopper to design and analyse a NURBS surface, it is important to have analytical ideas about designing with such surfaces. Since these surfaces could represent envelops, interior walls, floors, shades or various other building elements in design, there must be some analytical understanding of their performance and behaviour in the building and their responsibility in relation to the effective factors of design. In this section, we will design an imaginary project in which the generation of patterns and resultant surfaces are related to the analysis of the data which comes from the site. This data could be geographical, geometrical, ecological or any other relevant type of data that we can assess and implement in our design algorithm.

Coastline Land Subdivision Project

In this section, as an imaginary experiment, a coastline which is restricted by a canal has been selected to design a very simple pattern which subdivides the land with wavy shape curves. These curves are aimed to convert into surfaces like dunes as temporary structures.

Here the idea is to use the shape of the site as one of the important elements of subdividing the land and also apply some data from the site (here Heat analysis) to see how the formation of our design product could be affected by such analysis. The design product (which is really simple and basic) is aimed to be affected by these external data of the site.

Fig.9.1. Site aerial view

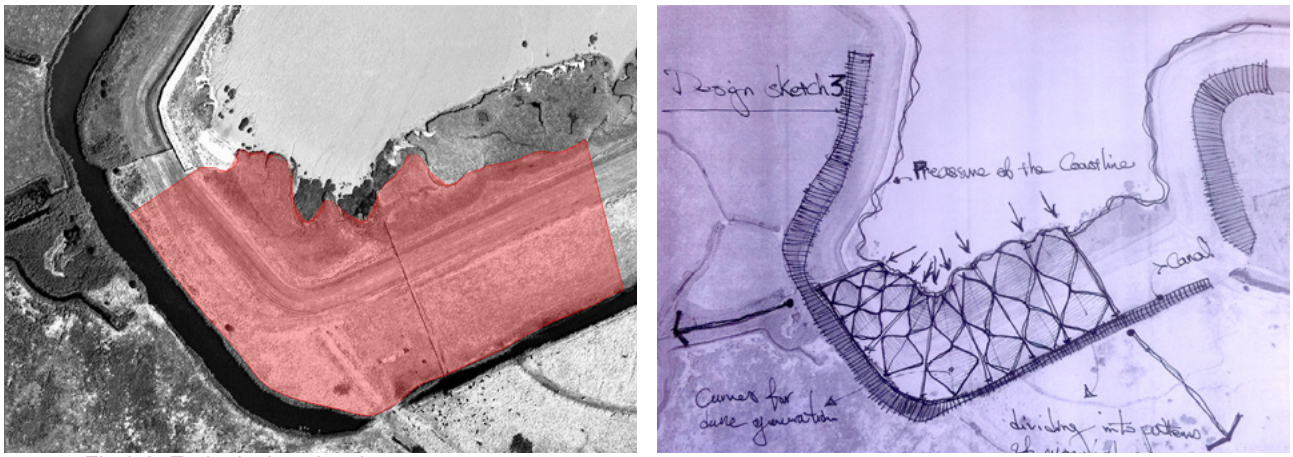


Fig.9.2. Early design sketches

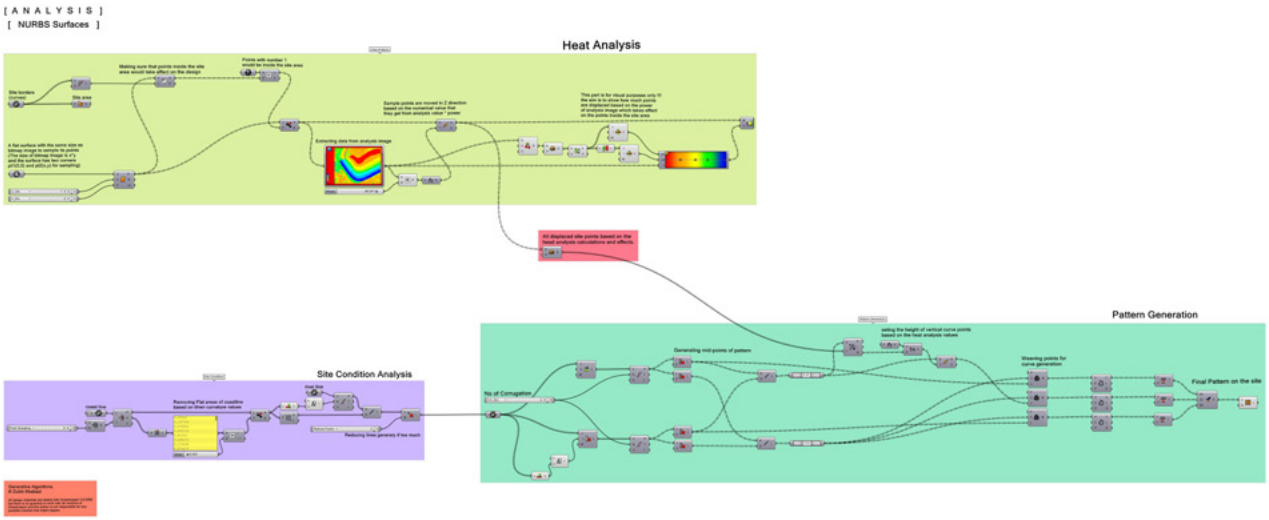


Fig.9.3. Design Algorithm in three parts: Site Analysis / Heat Analysis of the site / Pattern Generation

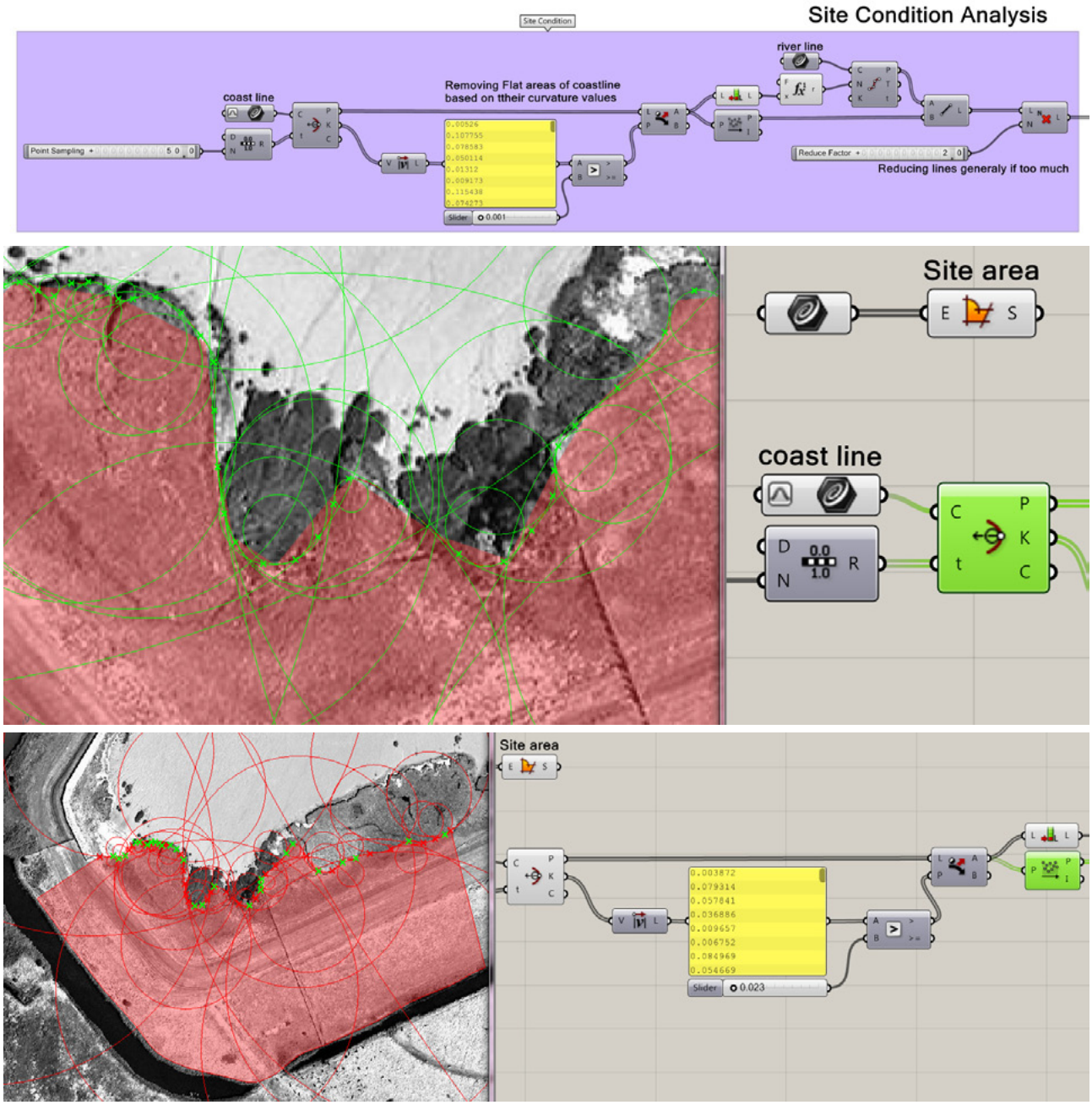


Fig.9.4. While the aim is to subdivide the site area with some straight lines to generate the wavy pattern between them, it is important to extract points across site borders for this subdivision. These points are generated across the coastline. Then based on the curvature of the coastline and the density of points, these points have been selected/omitted.

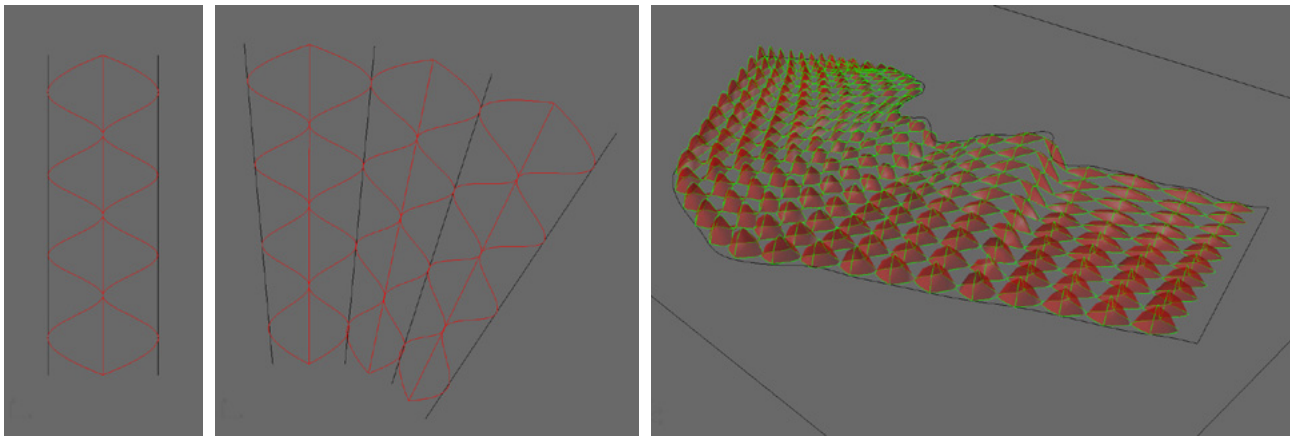


Fig.9.5. Simple, Multiple, and finall pattern on the site

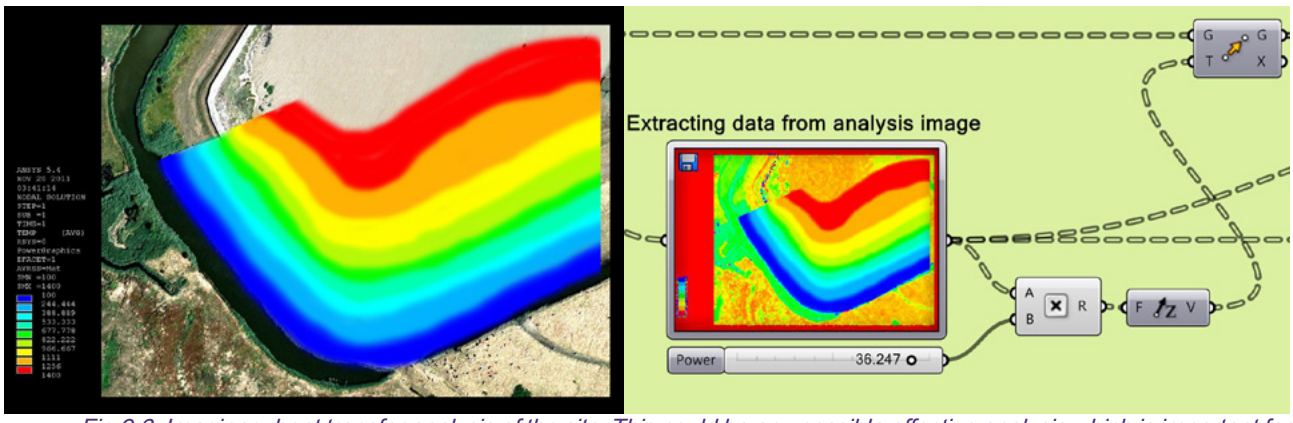
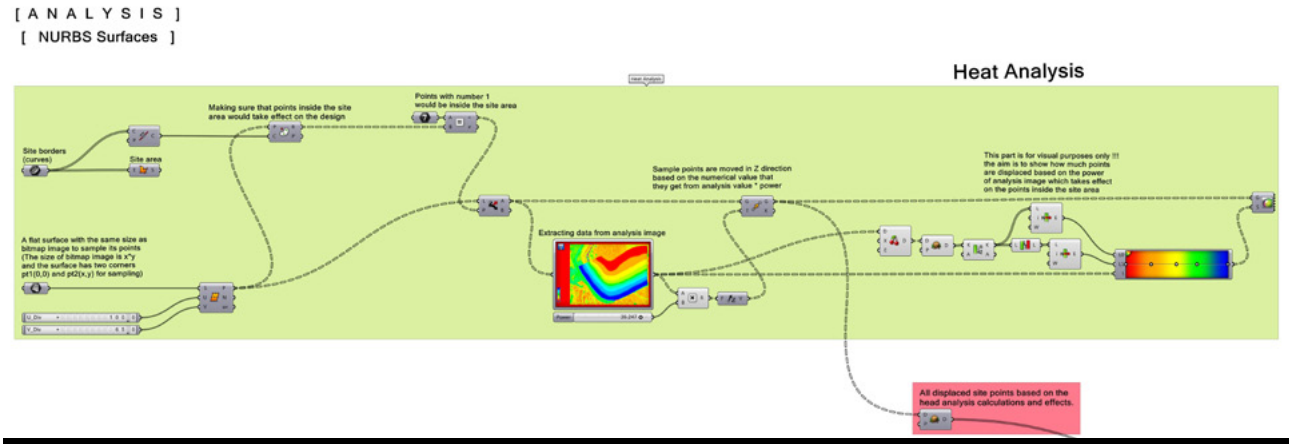


Fig.9.6. Imaginary heat transfer analysis of the site. This could be any possible effective analysis which is important for the design and here the idea is to extract its data through <Image Sampler> via colour coding system and apply it to design.



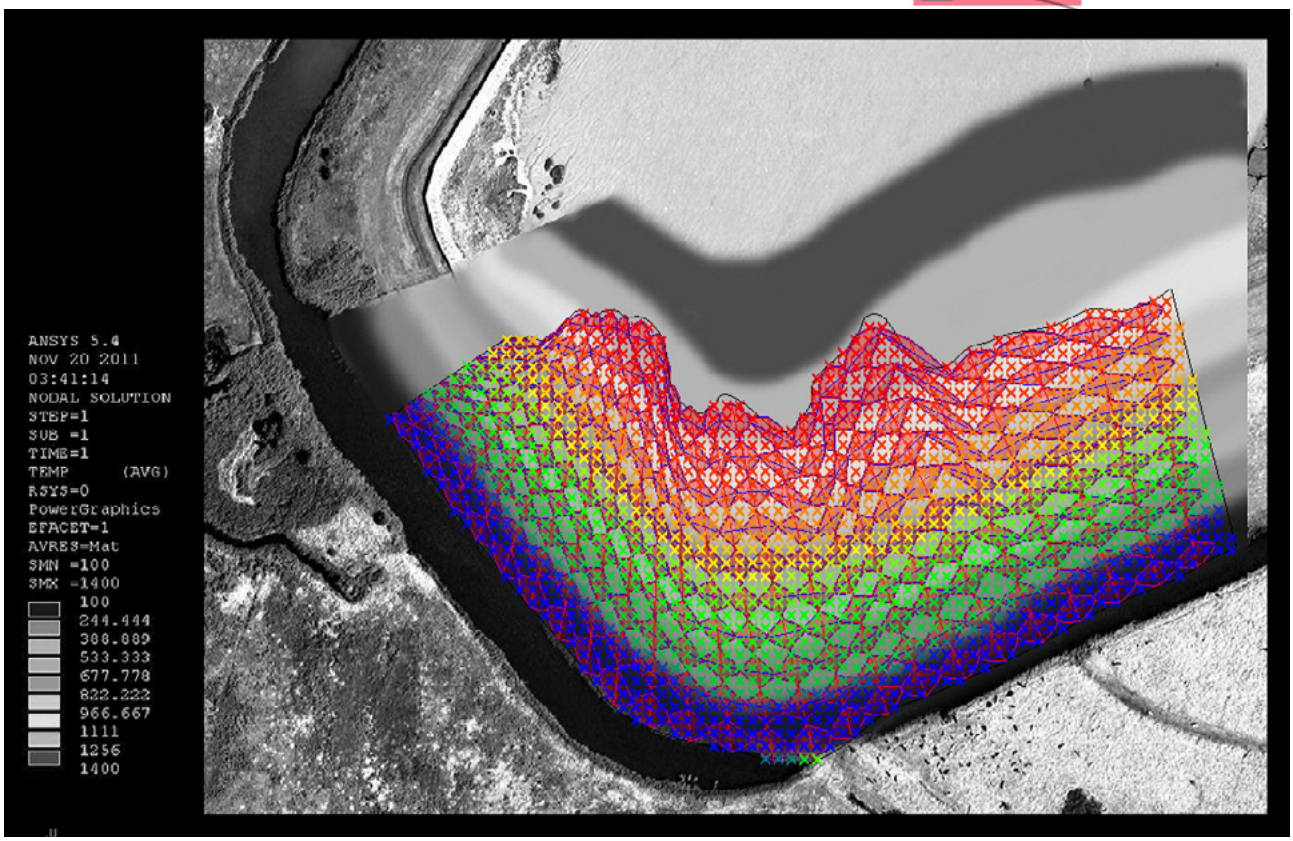


Fig.9.7. Data has been extracted from colour analysis codes and all sample points across the site are relocated based on the power that they get from associate data. Now it is possible to set the height of dune shape geometries based on these relocated points which is a relation between geometry and analysis data.

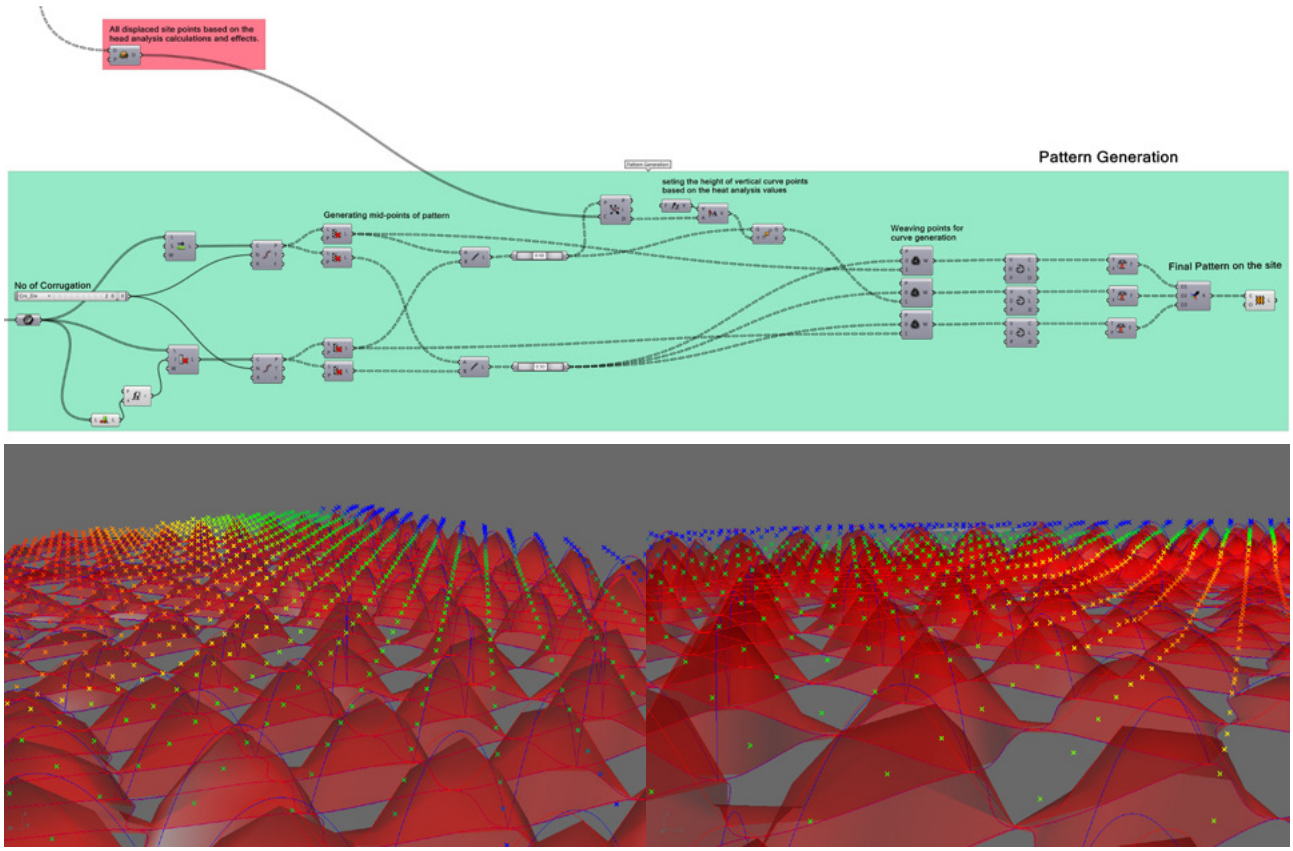


Fig.9.8. Generating Pattern and Dune shpae geometries

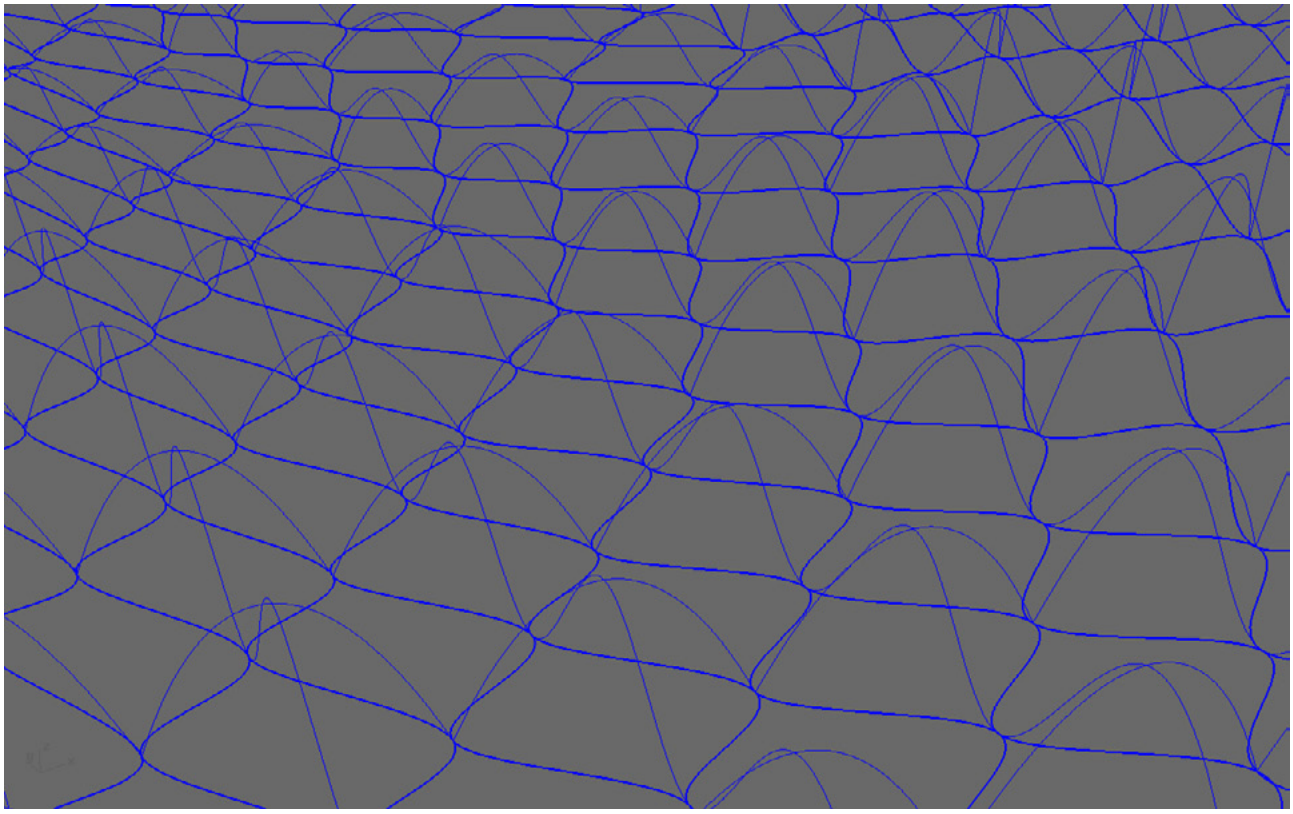


Fig.9.9. Final subdivided land area

9_3_Meshes

Up to now we have used different NURBS surfaces. But as mentioned before there are other types of surfaces which are useful in other contexts. It is not always the fanciness, smooth beauty of NURBS that we admire, but we might need more precise control, easier processing or simpler equations. In this section we will discuss a bit about Meshes.

Meshes are another type of free-form surfaces which are made up of small parts (faces) like Mosaics. So there is no internal, hidden mathematical curves that generate the shape of the surface, but these faces define the shape together.

If we look at a mesh, first we see its faces. Each Face has vertices which are connected together by a polygon like a triangle, quadrant or hexagon. These vertex points are connected together by straight lines. There are two important issues about meshes: position of these points and connection between these points. Position of points related to the geometry of mesh and connectivity of points related to its topology.

9_3_1_Geometry vs. Topology

While Geometry deals with the position of objects in space, Topology deals with their relations. Mathematically speaking, topology is a property of object that transformation and deformation cannot change it. So for instance circle and ellipse are topologically equivalent and they have only geometrical difference. Have a look at the following figure, you will see there are four points which are connected to each other. In the first image, both A and B have the same topology because they have the same relation between their points (same connection). But they are geometrically different, because of displacement of one point. But in the second image, the geometry of points is the same but their connectivity is different and they are not topologically equivalent.

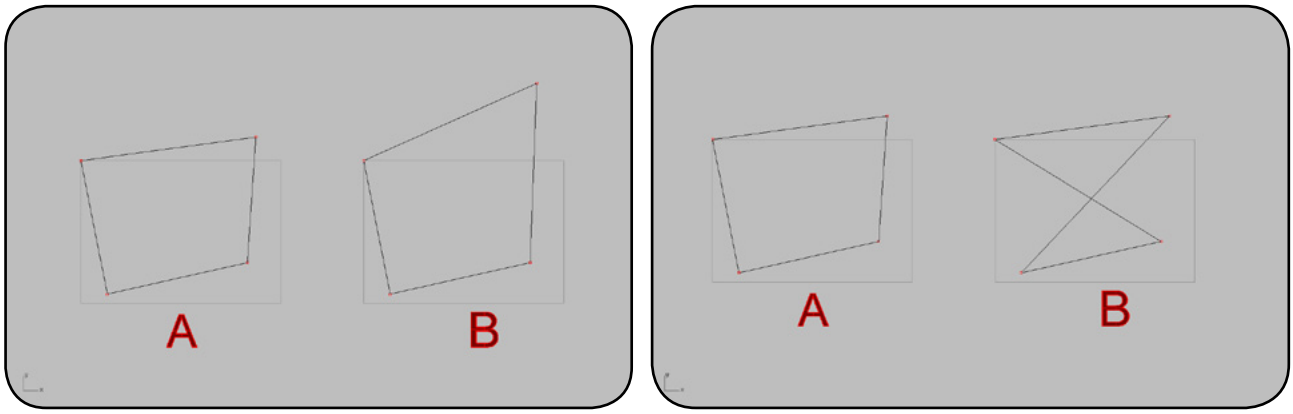


Fig.9.10. Topology and Geometry. *1.* *2.*

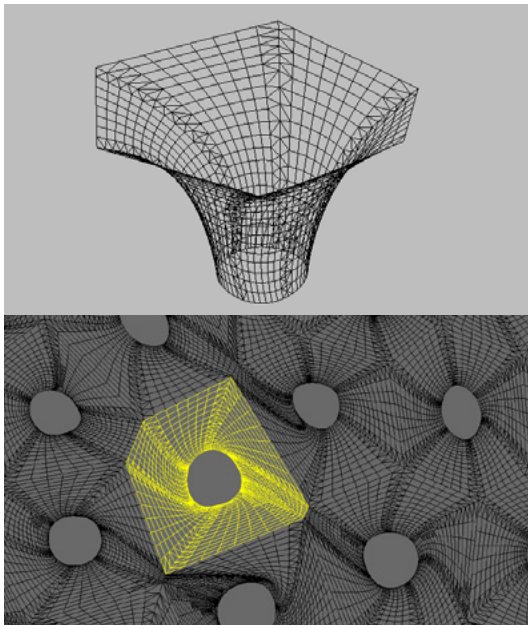
Topology is very important in mesh objects. Any face in a mesh object has some vertex points and these vertex points are connected to each other with an order. So we can apply any transformation to a mesh object and displace vertices of the mesh in space even non-uniformly, but the connectivity of mesh vertices should be preserved to retain faces otherwise the mesh object will collapses.

Knowing the importance of topological aspects of mesh objects, they are powerful geometries to work with. One can assume a mesh object as bunch of points where it is possible to apply different types of algorithms that transform and manipulate points. For instance, using finite element analysis or specific applications like dynamic relaxation and spring systems, it is possible to apply such forces and calculations to those points (Vertices) and get the mesh surface at the end of the process.

Mesh objects are simple to progress and fast to process; they are capable of having holes inside and discontinuity in the geometry. There are also multiple algorithms to refine meshes to make smoother surfaces. Since different faces could have different colours initially, mesh objects are suitable representations for analysis purposes (by colour) as well.

Roof Design with Sun-Shade Analysis

In the design space there is a closed curve (A rectangle) which represents the boundary of a roof space. The idea is to design a porous roof. The area of the roof is aimed to be subdivided by a subdivision algorithm (here Voronoi) and then each cell boundary should be a place where a funnel shape geometry would be generated. These funnels are responsible to let the light penetrate into the space underneath. But here, while dealing with geometrical complexities of the design problem, the idea is to control this amount of light penetration, and get the best possible solution based on the effective factors. So the design process has coupled with a feedback process with which it is possible to control the size of openings and the height of funnel shape geometries. Again the analysis of the external data of the site is important for the design process.



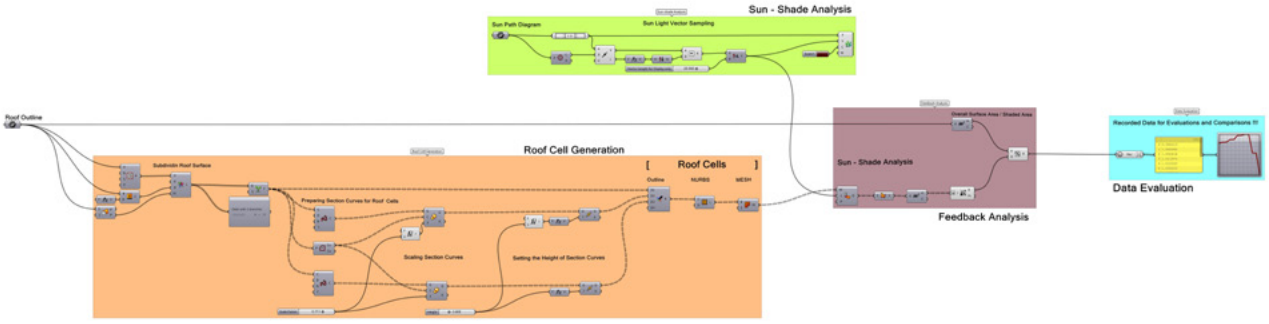
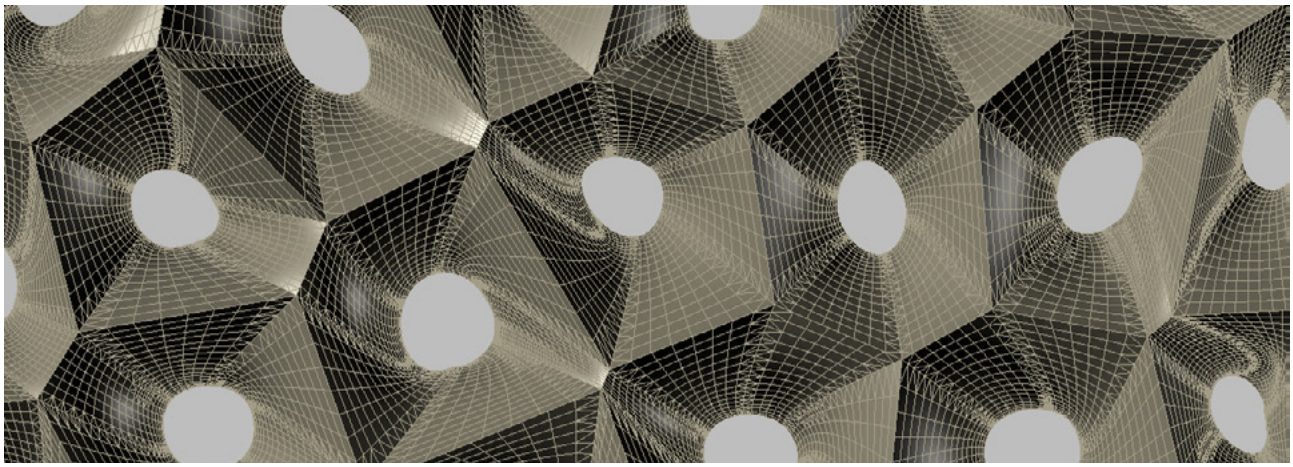
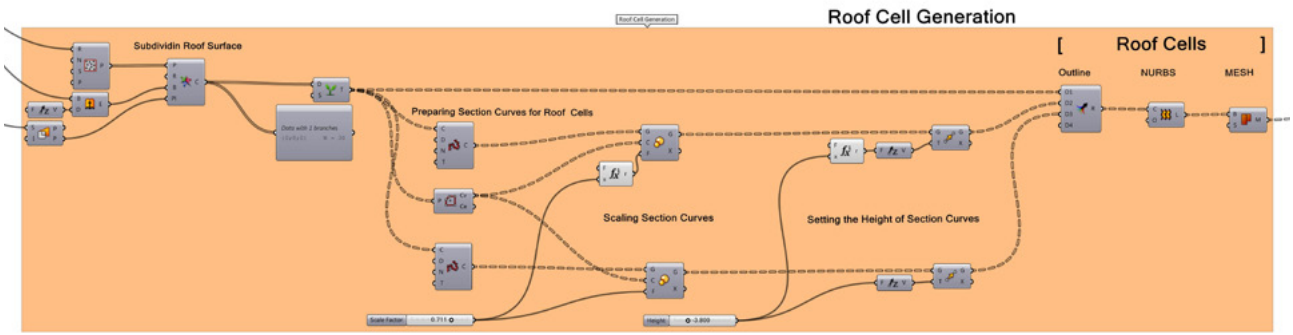


Fig.9.11. The very first part of the algorithm is to subdivide the area of the roof and generate cells. Roof cells are generated by lofting section curves which are scaled and moved. To make everything more interesting, the base polygons of cells are sampled with <rebuild> to convert polygons to smooth curves. Height and size of openings are adjustable.





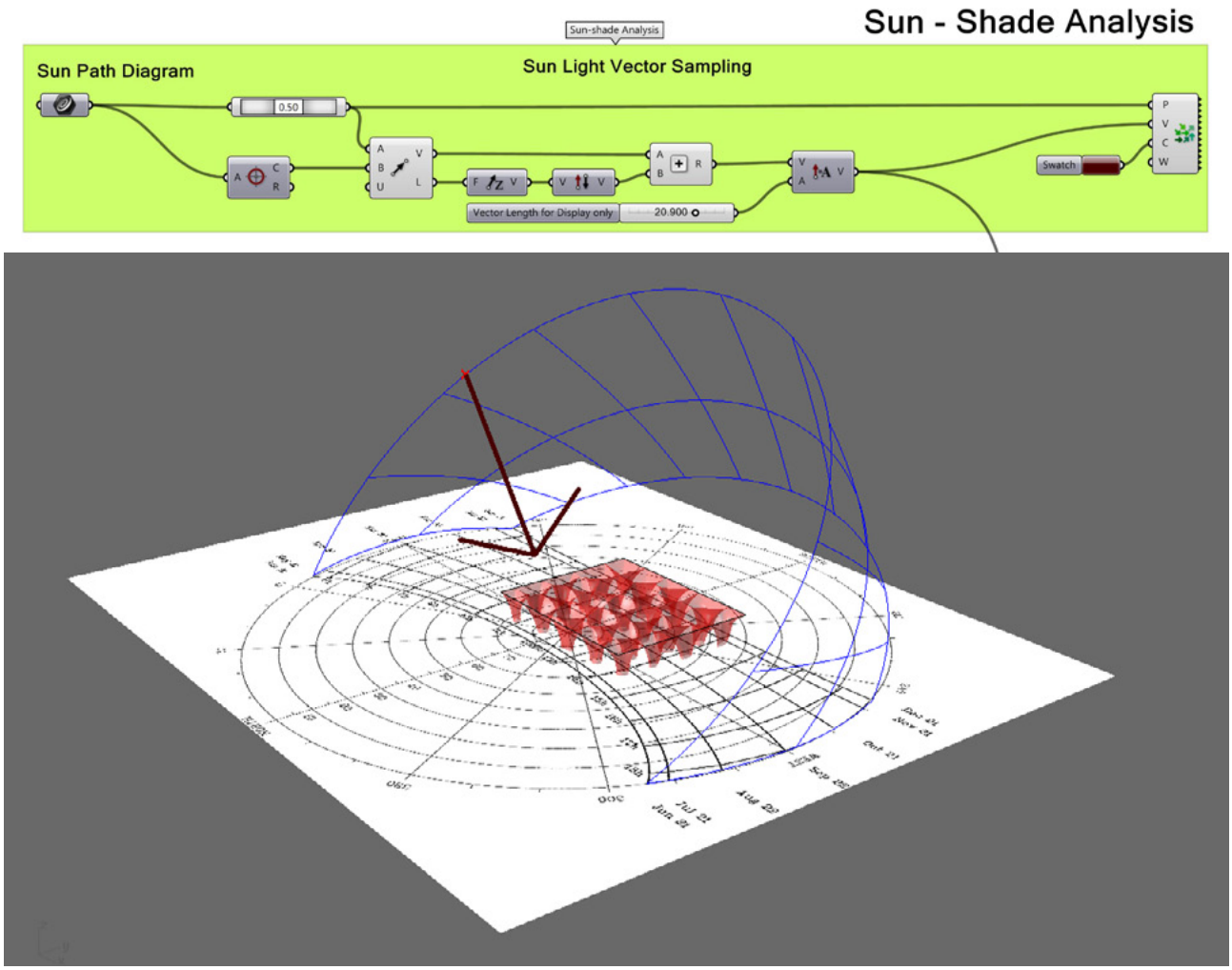
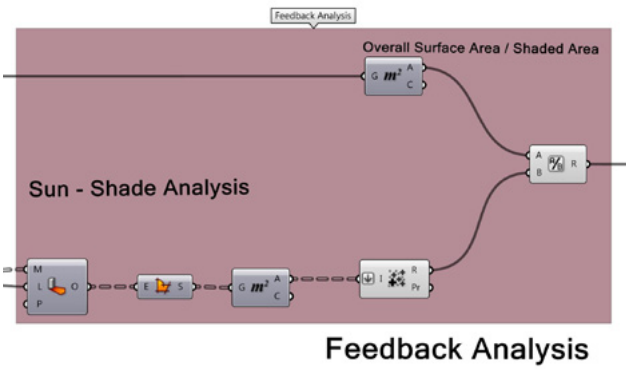


Fig.9.12. Although there are better and more elaborated solutions available on-line, here a very simple representation of a sun path diagram has been modelled to set a sun ray vector. This vector has been used to calculate the shade of the mesh object underneath itself. The roof area has been compared with the shade area to have a key value for comparisons and evaluations. Since some decisions are going to be made based on this analysis, a <Data Recorder> and a <Quick Graph> has been used to record the change of values of area/shade to be able to compare it for various parameters of the design. Here the sun ray was constant and the height of cells was constant as well and the size of openings has changed. The selection of the best possible option depends on the site situation which should be made by more detailed information of the project.



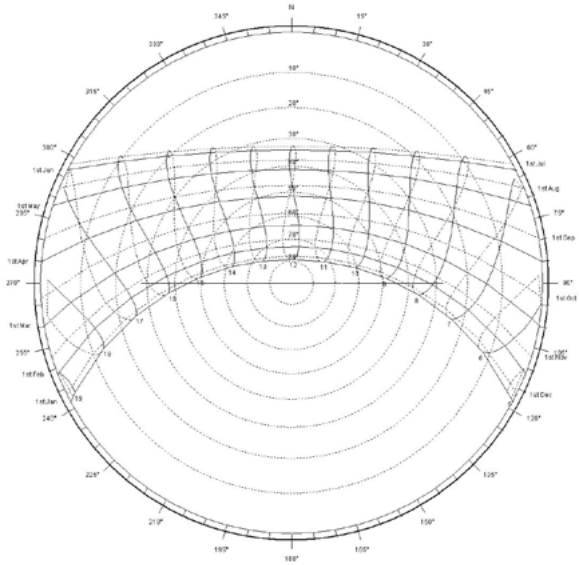


Fig.9.13. Although obvious, make sure that there are different sun path diagrams and other bioclimatic data available for each area of the little earth so find the proper one for your site!!!

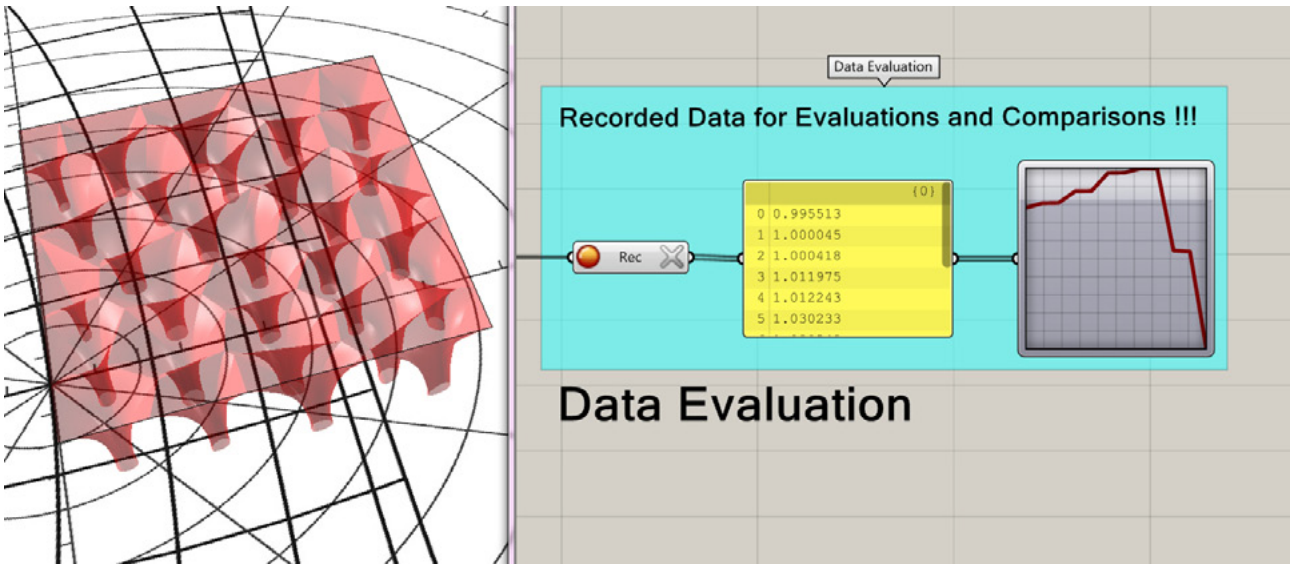


Fig.9.14. Record data for various options of height, size and even number of roof cells and compare these recorded data to find the best possible option of the design (based on its degree of light penetration).

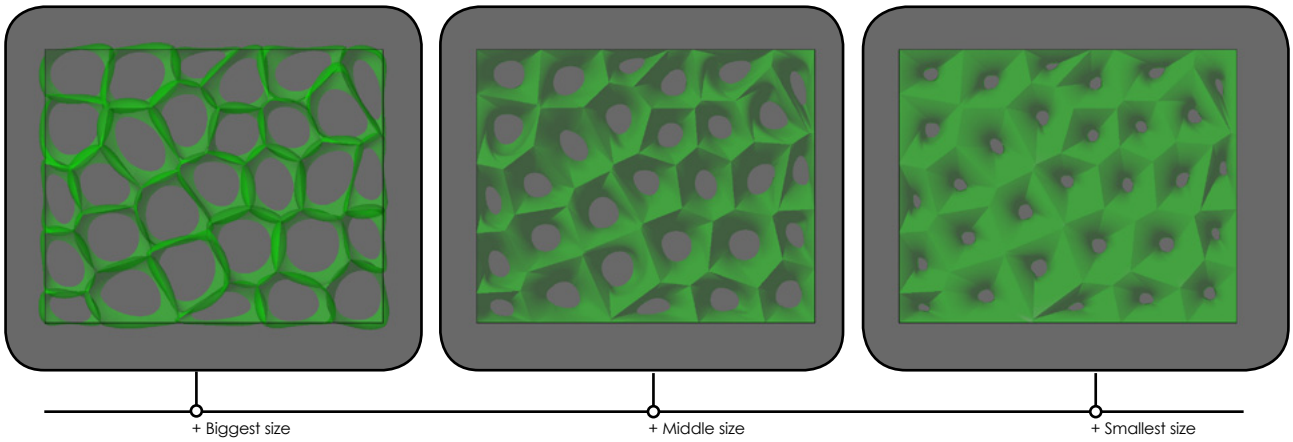


Fig.9.15. Testing the size of openings.

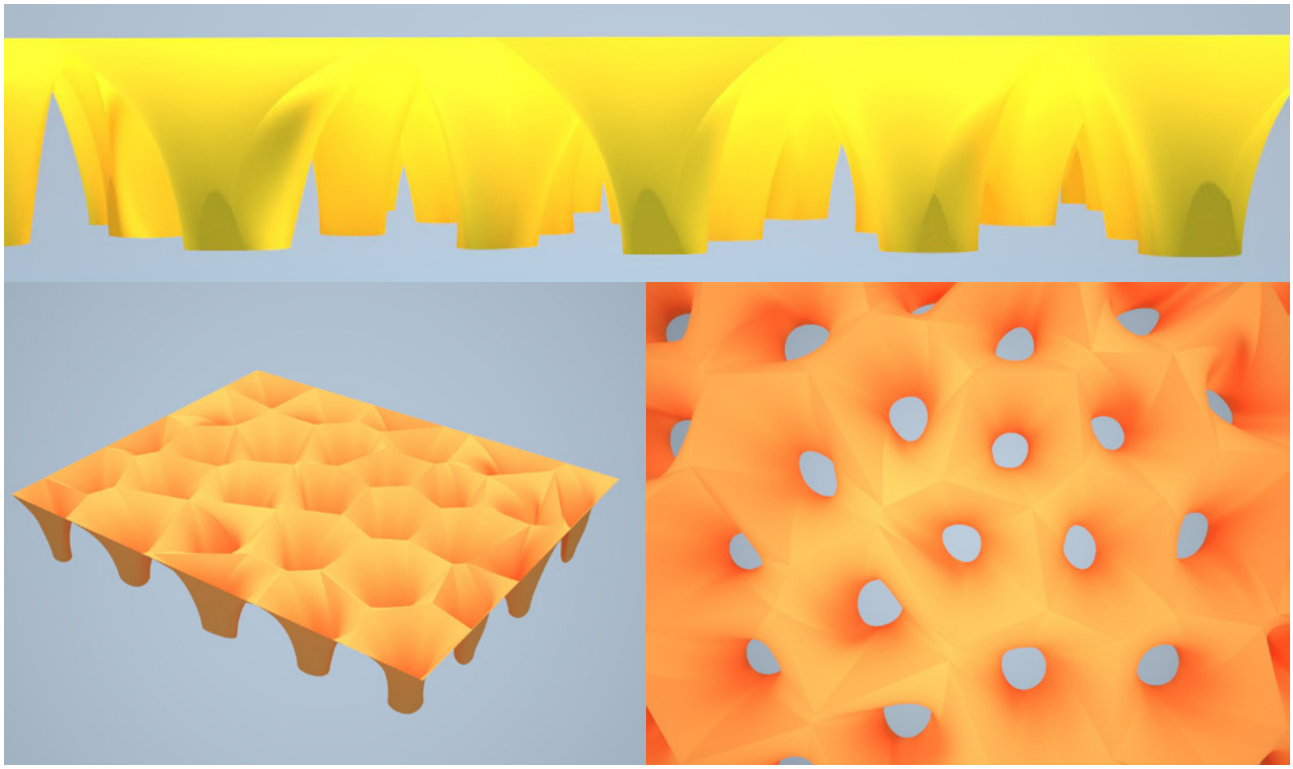
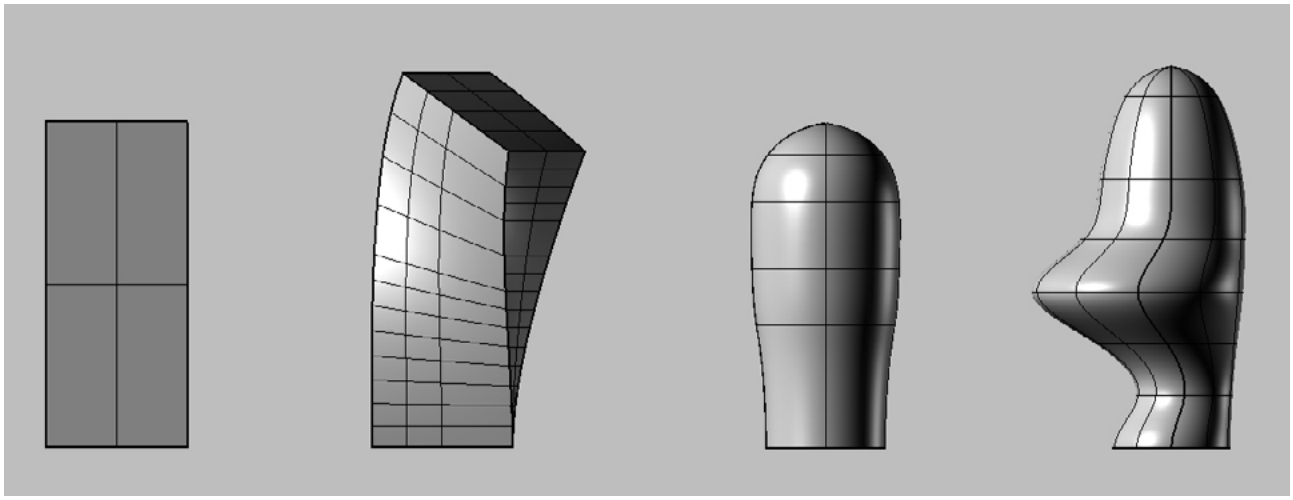


Fig.9.16. Final mesh output of the algorithm as Roof Cells

9_4_Macroscopic Design

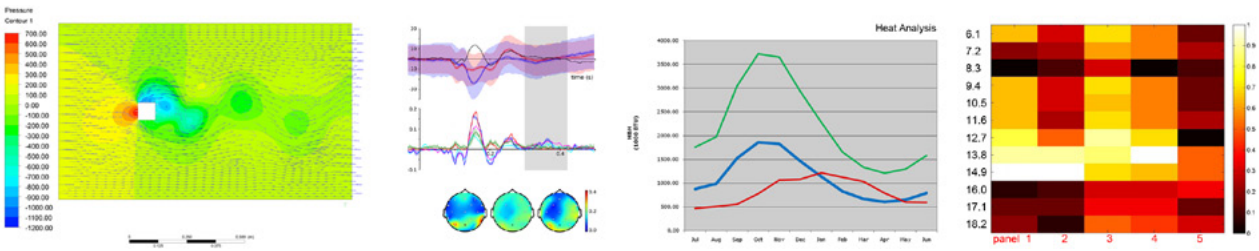
Design of any single building is restricted to various data which are coming from different sources. Project's site specifications and location, Planning Permission and Urban Authorities regulations, Building and Technical Regulations, Program of the project, Bioclimatic and Ecosystem data and all other large and small pieces of information that might be effective. After all, there would be architects and clients tendency, desire or logic to come up with ideas for overall form of a building. Whatever this form is, one or multiple geometries, small or tall, cube or curve, it could be a poly-surfaces that cover this building. These surfaces, as they are mostly curve and smooth in algorithmic design, are called envelop where they cover a space with smooth transition from one side to another. Beside all limitations and regulations, there are degrees of freedom in designing such forms that should be addressed in design algorithms as we did analytically and simply in two design experiments.



Macroscopic design here means understanding and utilizing the data which is associated with the project and seems to be important for its formation. Sun-Shade, Wind, Humidity and Weather Analysis are among those data that could affect the Macroscopic design of a building; Also structural behaviour, seismic forces or project's internal pressure, program or organization. These factors and forces could be transferred into the design algorithm through vectors or numerical values, bitmap images, etc.

A design algorithm should include vector operations and evaluations of such forces to apply them onto the design space (i.e. Envelop). A project which is 'Optimized' by such forces/data has more chances to respond to the external stimuli and forces in better ways, help to reach higher performance capacities for the entire project, hopefully push it towards a more green, 'Sustainable' design.

It is important to notice that Microscopic design is also important and could affect the Macroscopic design. Details, Material Systems, Fabrication necessities of such systems, Material Performances and all other data that come from Microscopic design level would also be part of the design in Macroscopic scale and project's overall form needs to be optimized again by its Microscopic necessities. This will be discussed in the following chapter.



CHAPTER TEN

DEFORMATION AND MORPHING

Chapter Ten
Deformation and Morphing

10_1_Deformation

While 'Generative Algorithms' was about generating geometry so far, it is also possible to deform those geometries, alter their original state. Deformation techniques are among the powerful functions in the realm of free-form design. By deformations we can Twist, Taper, Shear, Bend, … objects. Deformation can be described and sometimes can be calculated by using a bounding box which is modified and caused the inside object to be modified as well. Free-Form Deformations happen when the deformation is not one of the above named techniques and corner points of the bounding box in 3D space are displaced without relation to each other.

Let's have a look at a simple deformation. If we have an object like a sphere, we know that there is a bounding box (cage) around it and manipulation of this bounding box could deform the whole geometry.

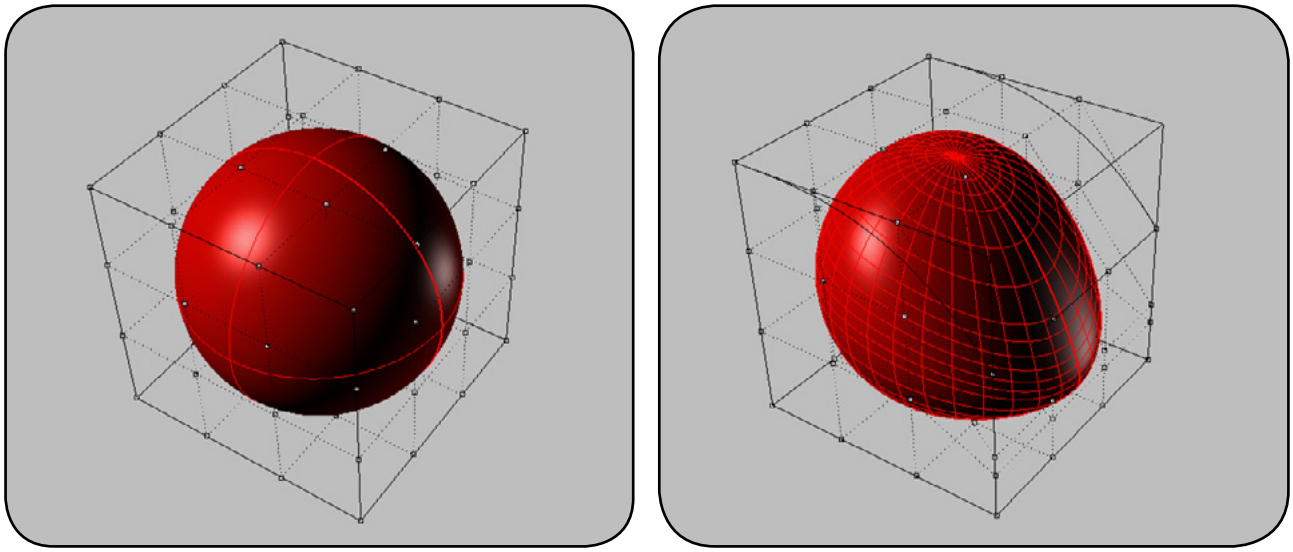


Fig.10.1. Deformation of an object by its Bounding-box (cage).

Based on angel, displacement, rotation, etc. we might call deformation shear or bend or free-form. This means that each type of deformation needs its own relevant data as well. If you check different deformation components in Grasshopper you can easily find the base geometrical constructs to perform deformations.

10_2_Morphing

Morphing in animation means transition from one picture to another smoothly and seamlessly. In 3D space it means smooth deformation from one state or boundary condition to another. Morphing components in Grasshopper work in the same fashion. Here for example <Box morph> component (Transform>Morph) deforms an object from a reference box (Bounding Box) to a target box, or <Surface morph> component works with a surface as a base, on that you can deform your geometry, on the specific domains of the surface.

Any object in 3D space has a bounding box and it is possible to deform this bounding box (Shear, Twist, Taper,...) and then morph the object into this new condition of the bounding box to have a deformed object.

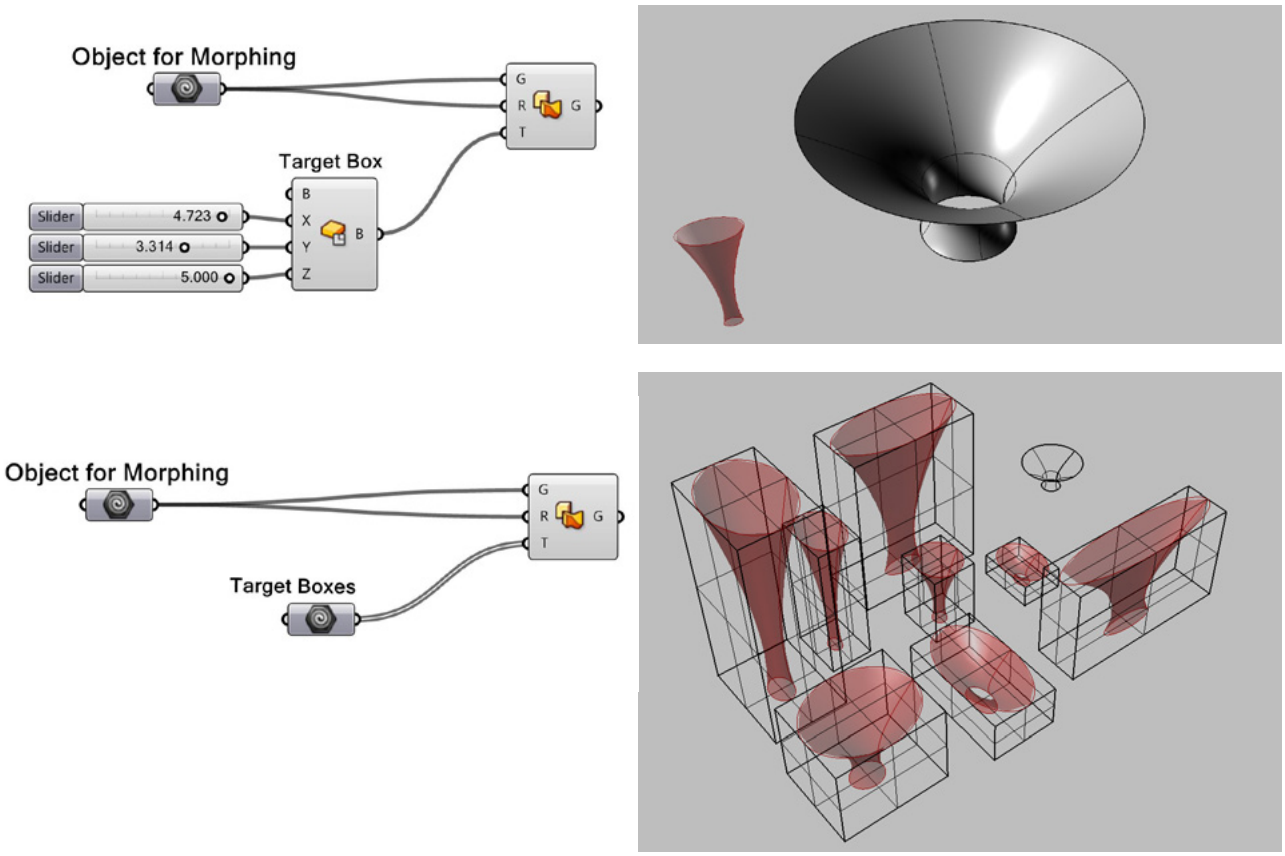


Fig.10.2. A <Box morph> component deforms an object from a reference box to a target box. It is possible to morph an object into one or more target boxes.

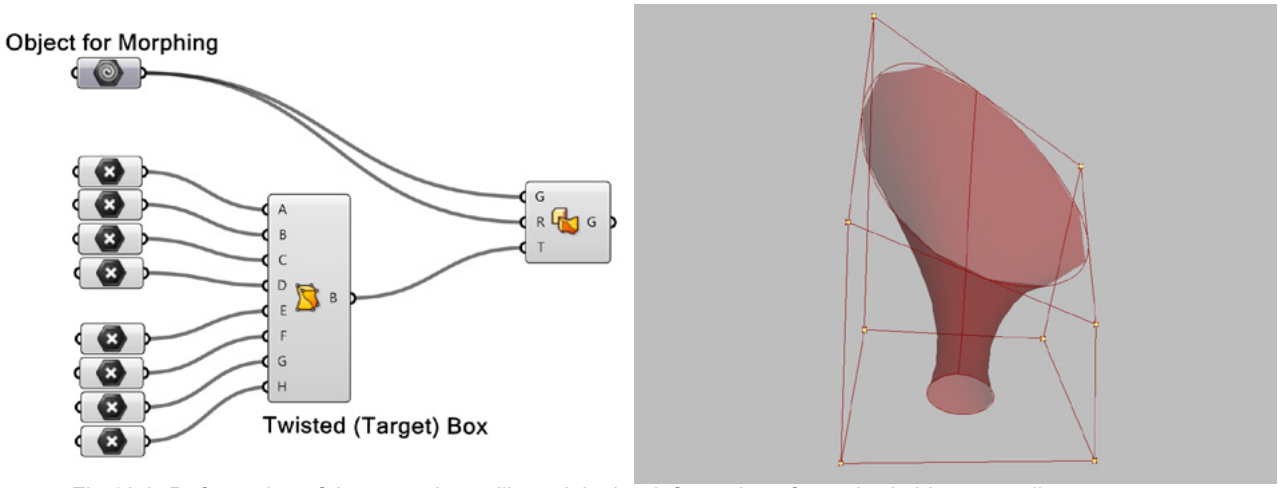


Fig.10.3. Deformation of the target box will result in the deformation of morphed object as well.

10_3_On Panelization

One of the most common applications of morphing is Panelization. The idea of panelization comes from the fabrication necessities of a free-form surface geometry in which it should be divided into small parts (tessellation) which wanted to be covered by a Panel (A component). Although free-form surfaces are widely being used in car industry, it is not an easy job for architecture to deal with them in large scales. Panelization helps to sub-divide a surface into small pieces, called components which are easier to fabricate and transport and more controllable in terms of precision in final product and also assembly.

It is also possible sometimes to divide a curve surface into small flat parts and then get the overall curvature by accumulation of the flat geometries which could be then fabricated from flat-sheet materials. When doing Panelization, There are multiple issues regarding the size, curvature, adjustment, deformation, etc. that we try to discuss some of them, but there are huge amount of reading materials and experiments to search and find the logic and method.

Panelization

Basically the idea is to design a component and a global surface to be covered by a panel. The technique comprised of two main steps. Generating boxes (as target boxes) over the global surface and then morphing components into these boxes. The number of panels in each direction should be set through surface subdivision and there are various options to control the height of components accordingly.

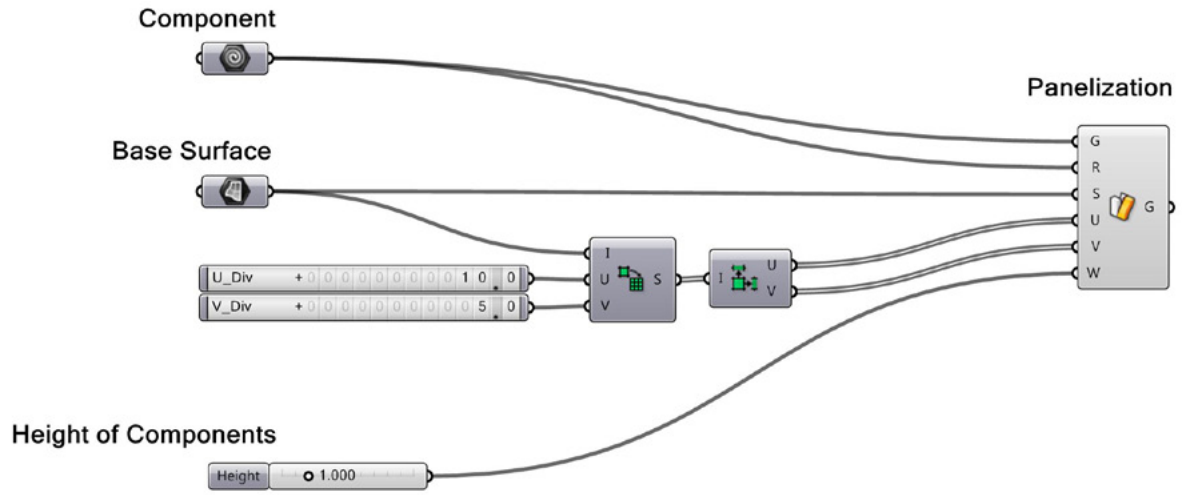


Fig.10.4. A Base Surface is subdivided in U,V directions and a <Surface Morph> component has been used to morph the designed component over the divided domains of the surface with controlled height.

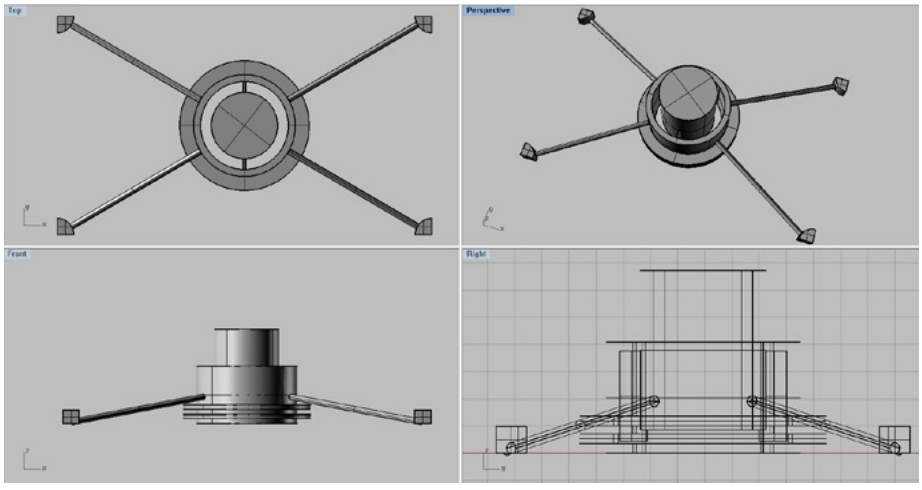
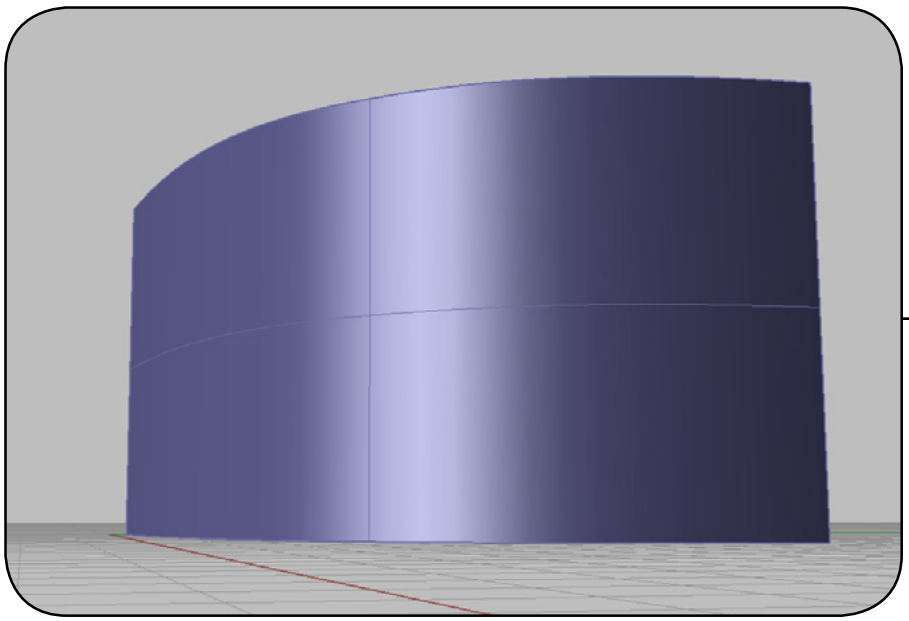
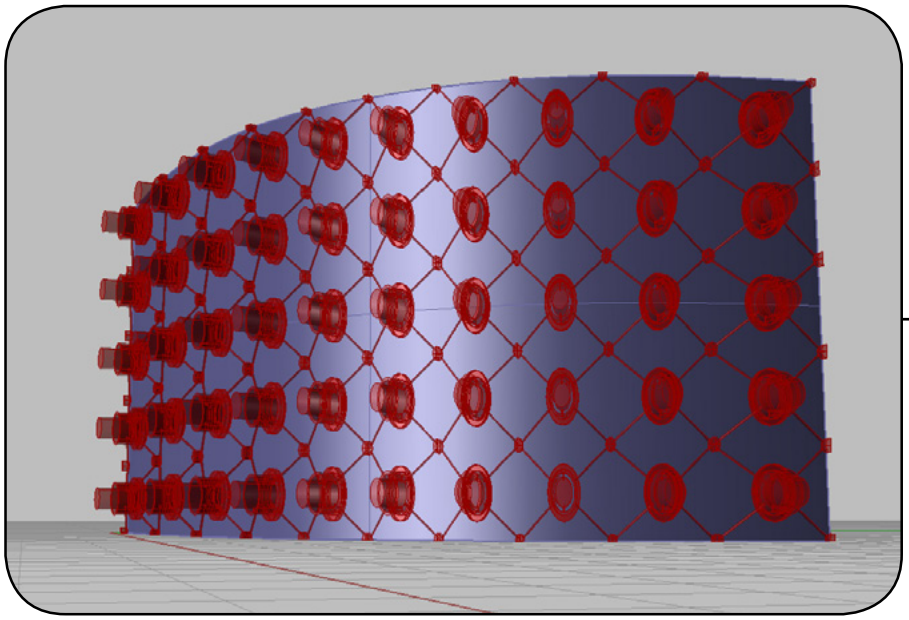


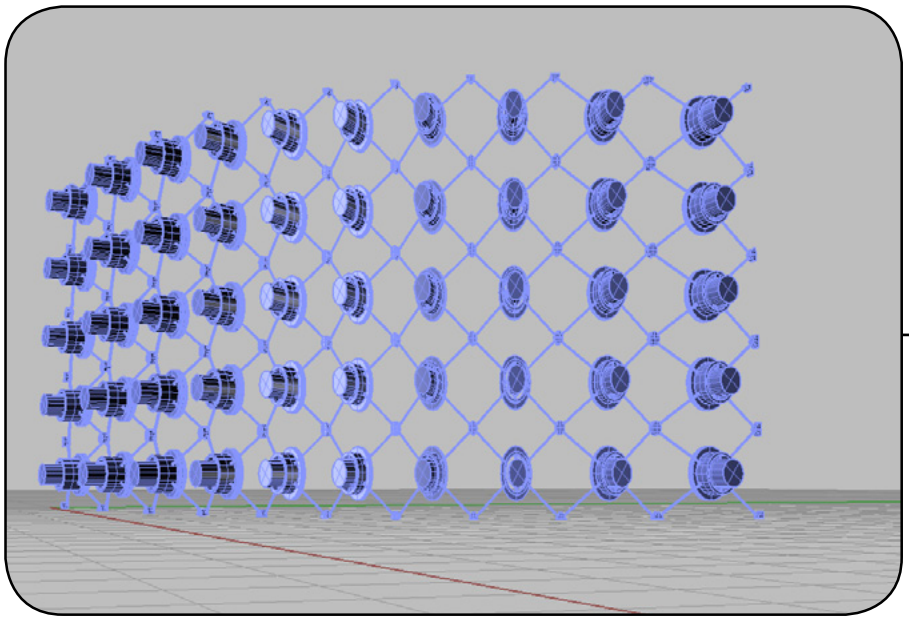
Fig.10.5. Component



+ Panelization 1:
_ Base Surface

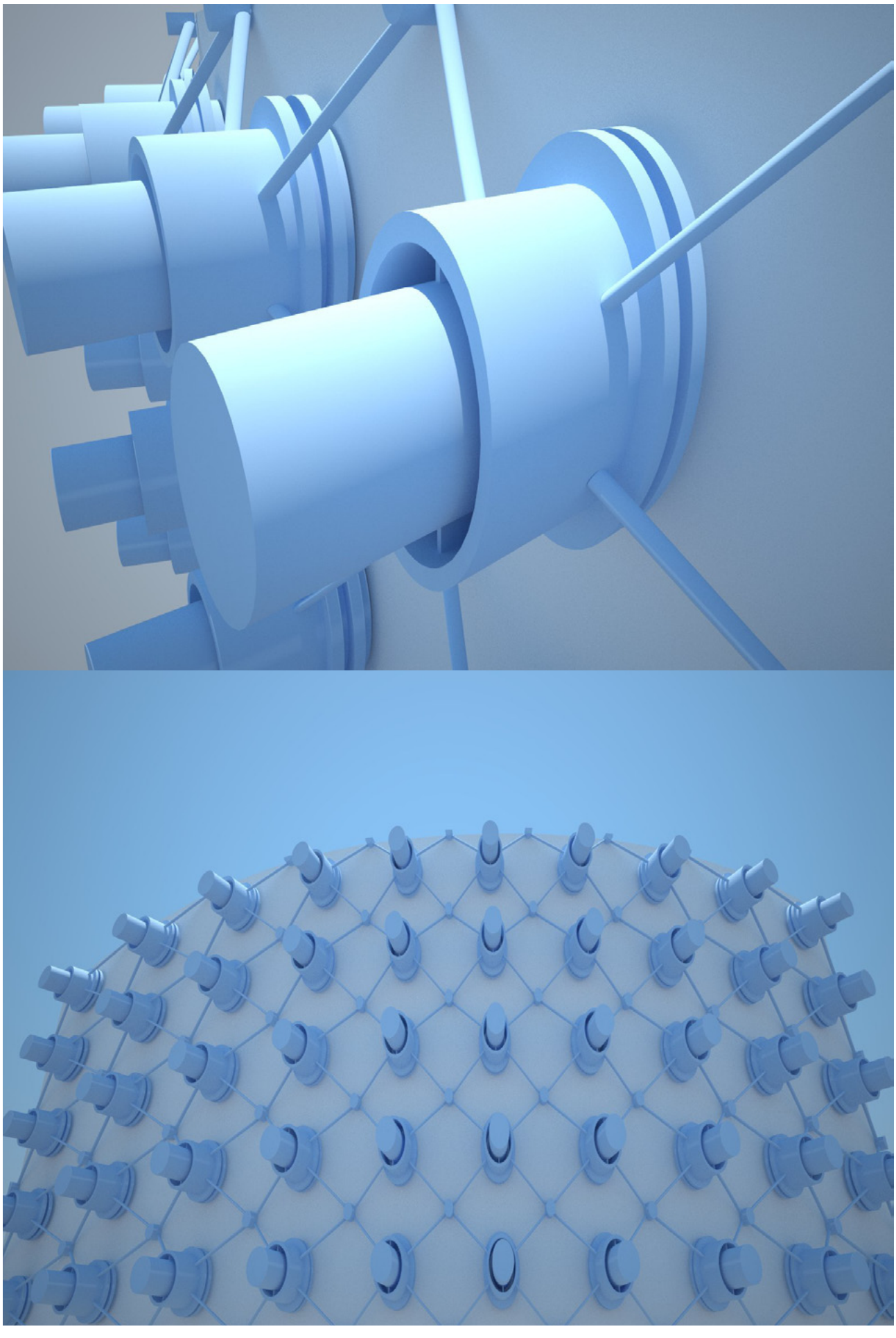


+ Panelization 2:
_ Morphing



+ Panelization 3:
_ Baked Component

Fig.10.6. Panelization



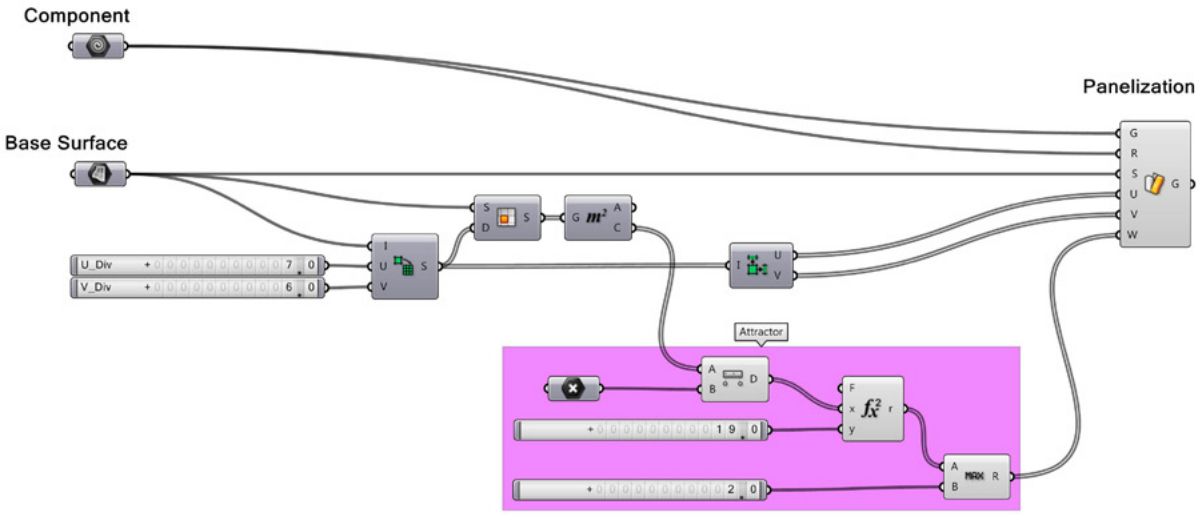
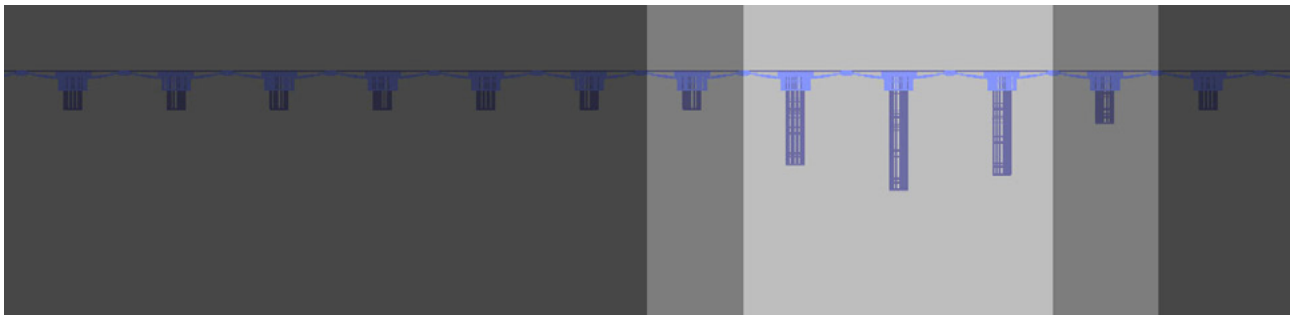
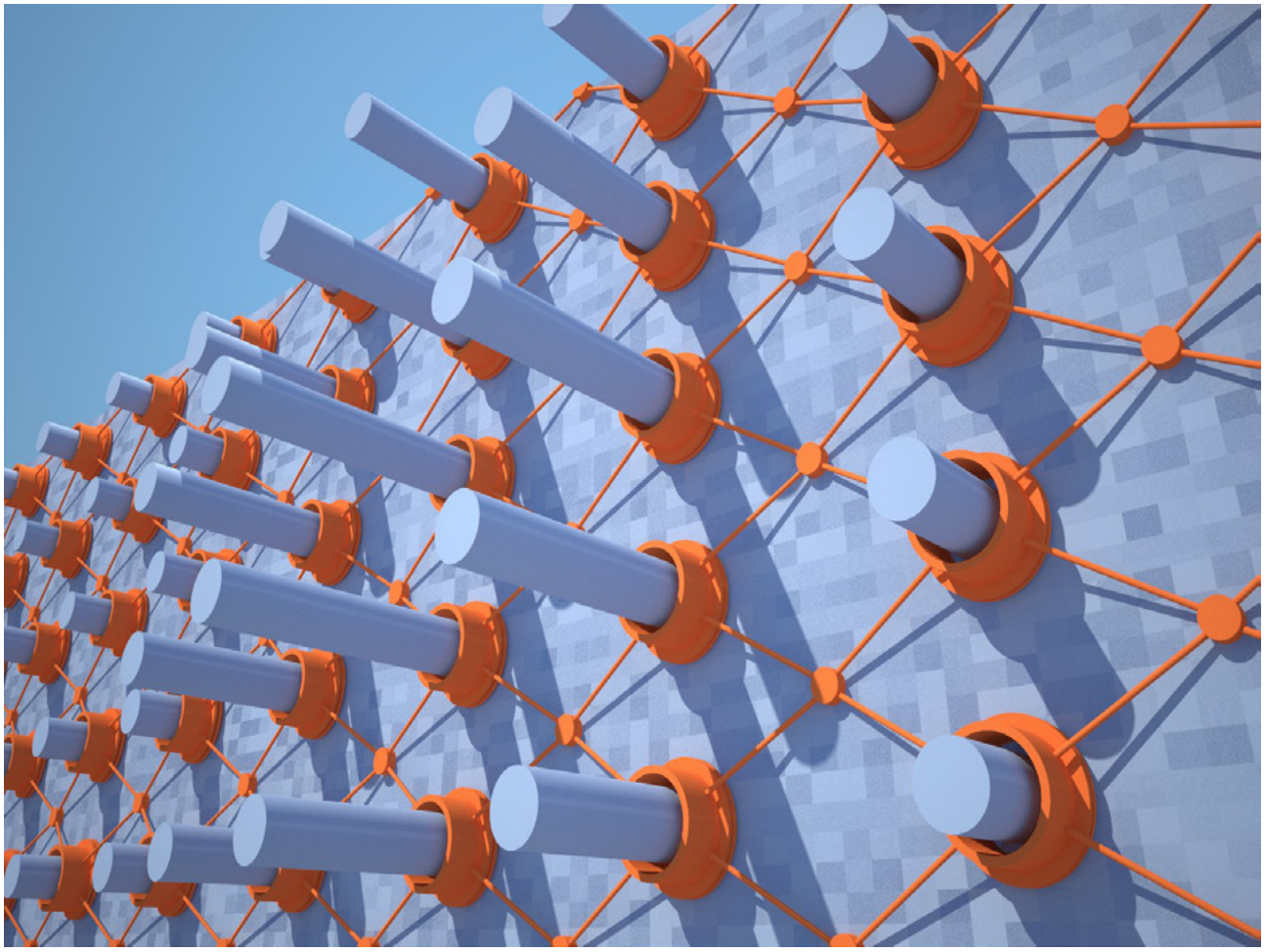


Fig.10.7. The height of target boxes in Box Morph or Surface Morph can be set via various criteria, here in relation with an external point that can affect the panelization depth locally. In this experiment only cylinders have been affected by this definition and the rest is normal panelization.





10_4_Microscopic Design

In the Macroscopic design section we discussed how we should approach the design of a project in its large scale, and it was described that there are various elements which are effective in the formation of the project's global form. Imagine that a surface from such process formed and aimed to be panelized. Here there is another important level of design awareness which is Microscopic design.

In the Microscopic scale, designers try to implement ideas in small scale which might come from properties of the material system that they are using, properties of interconnection and assembly of panels, possibilities of applying various changes, transformations and deformations to panels and so on and so forth. So in this case, where the global shape of the envelope is constant, there are various manipulations and deformations that could be applied into small scale. It worth mentioning that microscopic design is not just limited to the panelization and morphing and all other design strategies could be informed by this level of design awareness.

Here for example look at this example. There is a material system consists of two interconnected layers of bended wood which make a cross section. These pieces should be fabricated by wooden sheets to be bended in order to get the shape. Since the material has limitations in its bending properties based on its thickness, material and fibre directionality, the degree of this bending should be controlled, means that too much bending would create cracks on the piece and bending less than a certain amount would cause it to become loose and flat without any structural capacity.

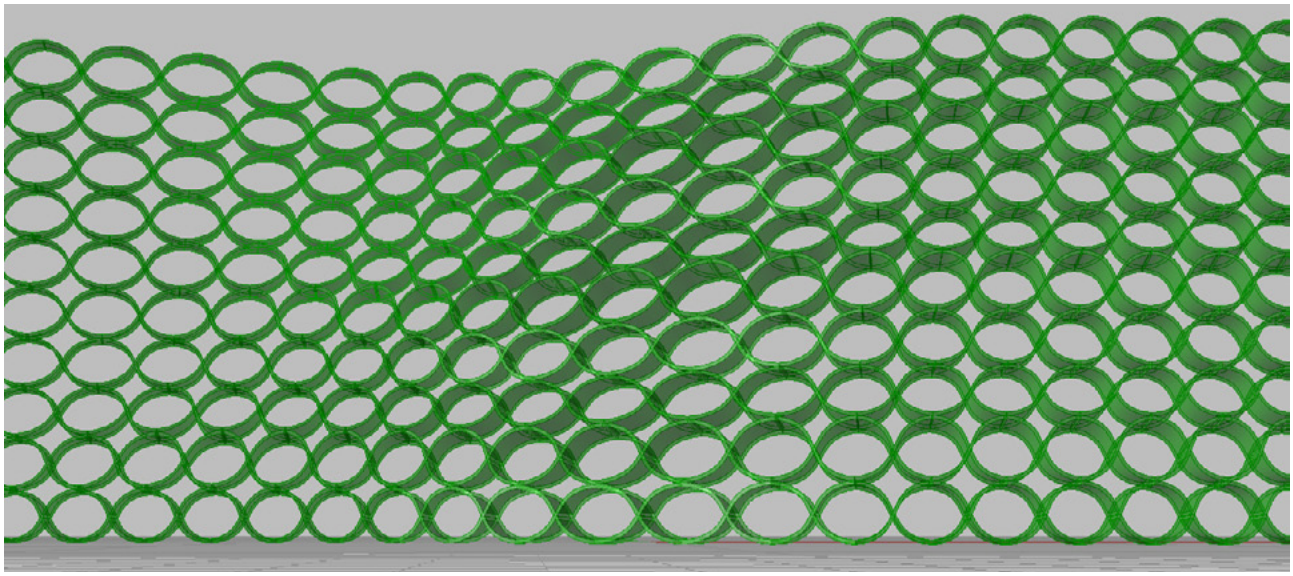


Fig.10.8. Panelizing a wall with bended wooden sheets

While in digital geometry it is easy to model the initial state of the cross section as the basic component and also overall surface as the final state of the project, it is also easy to panelize the whole surface by copying the component over it. The morphing engine would deform all target boxes in order to fit the component into them and here is where problems would start to appear. The size of twisted boxes might change based on the surface condition and this would result in the change of the size of initial component. In order to prevent damages to the pieces that should be fabricated afterwards, all parts should be controlled and those smaller/Larger than acceptable size should be highlighted in order to solve the problem. If the designer does not pay enough attention to these very simple observations and controls, the resultant manufacturing and fabrication might encounter problems, waste of material and loss of time and money.

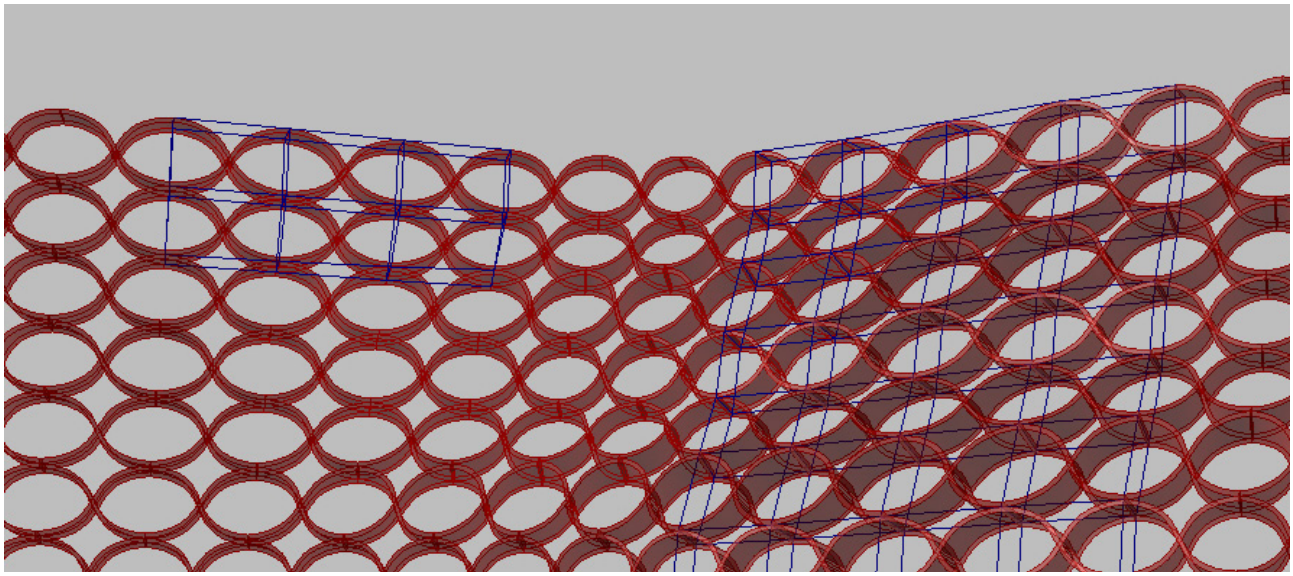
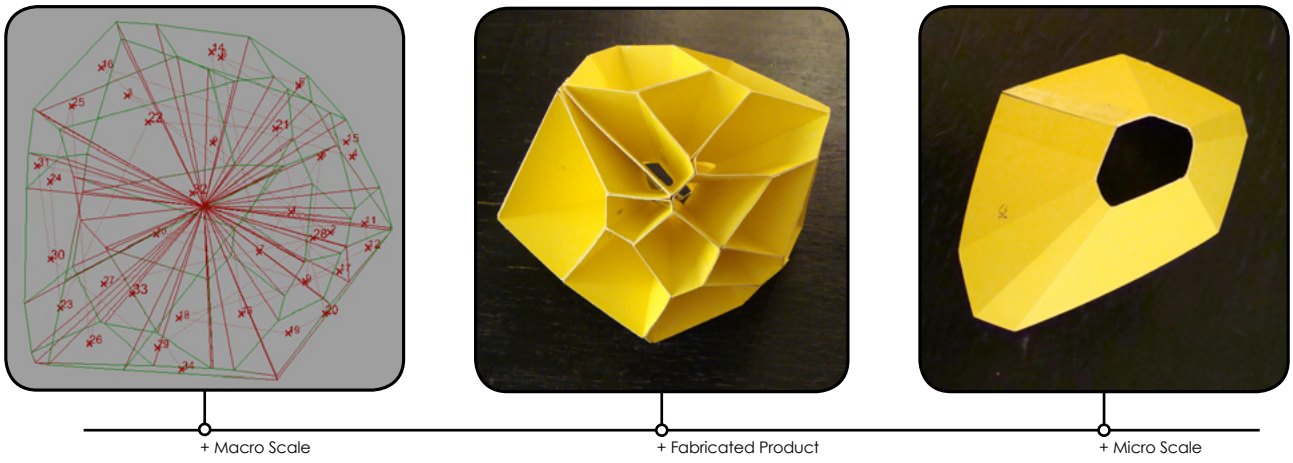


Fig.10.9. Controlling the size of components for fabrication and material purposes. Those which are in the range of danger are highlighted to control and change their size in order to prevent pieces to crack.

These Microscopic controls might include the amount of bending for a piece, the direction of material where working with anisotropic materials, material's thickness in different parts of the project, joints, interconnection and overlapping of pieces, deformation which is not supported by flat-sheet materials, spaces out of the reach of CNC head, material pieces and components bigger/Smaller than standards for joining, fabricating or transport, geometry of components in relation to their material properties especially for cutting (i.e. cutting wood parallel or perpendicular to its fibre directions where engraving bridle joints) etc, etc.

Microscopic design might also affect the component's behaviour or even type. There are various reasons that the design needs differentiation in components or even change in its type. While talking about responsive components, they might tend to accept changes in order to adapt themselves to various situations of the project in relation to external stimuli. This might open up the subject of adaptation of building elements and their responsiveness to environmental stimuli which is really interesting to implement in design algorithms.



Looking at Macro / Micro scale. Xanathous Sponge project. morphogenesis

10_5_On Responsive Modulation

In contemporary design practice, Responsive Modulation is a design strategy which helps the enhancement of architectural products for their environmental performances. For instance in a building envelope, it might designed like human skin where it modulates the interior climate by controlling the amount of energy and material that transfers between inside and outside.

We are not going to discuss design solutions here. The aim in this section is to do some experiments to deal with the necessities of algorithms of such design processes that fits this section of the book. Here the aim is to design an envelope for a building which has more than one type of component to cover it. The technique is still panelization but using different panels, it is possible to assign various types of them into the building envelop based on its local conditions. These conditions can be evaluated from various external/internal data.

The aim is to design various components which are responding to the sun light and change their porosity (it could be wind, rain or internal functions or any other criteria). There are three different types of components with various degrees of porosity. The first one has more than 75% openings in its area. The second one is about 50% porous and the third one less than 25%. This combination of components will generate a gradient of components across the façade surface in order to cover the building and create the envelope. It is also possible to change the components in terms of their behaviour, or any other properties which help for the better performance of the façade system.

Although responsive components and responsive building skins are not that simple, here the technique of bifurcating data, based on external stimuli is important. The design solutions to generate complex and intelligent, yet passive-responsive elements and envelopes are up to you as the designer.

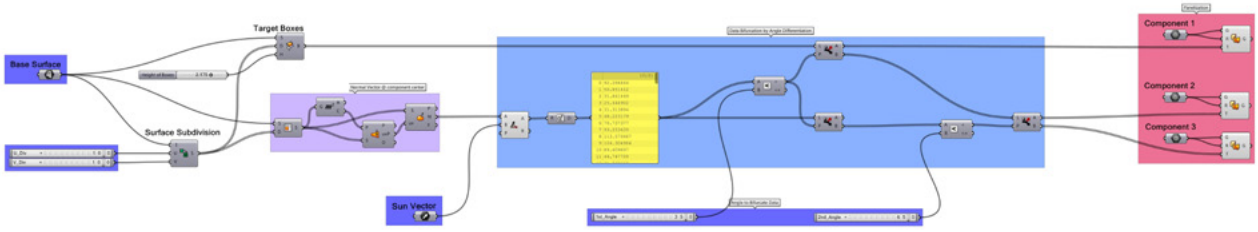


Fig.10.10. A Panelization definition with different components to distribute over the base surface.

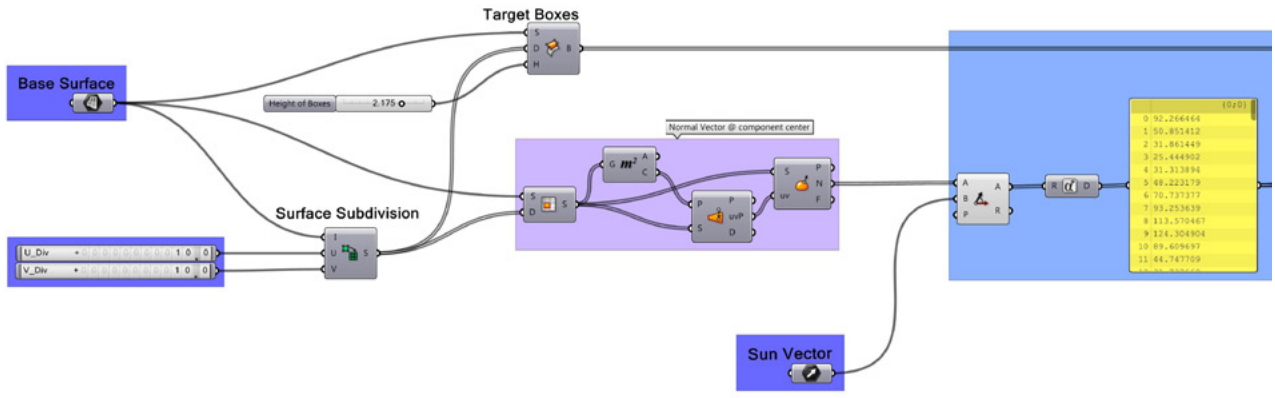


Fig.10.11. While the base surface is subdivided to generate target boxes, the angle between a pre-defined <sun vector> and the surface Normal at each subdivided surface center has been calculated.

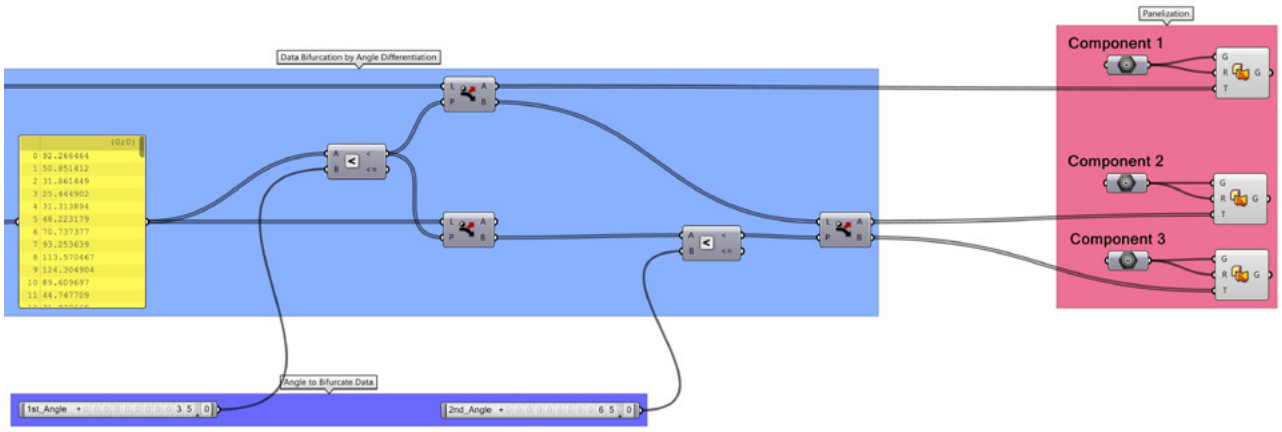
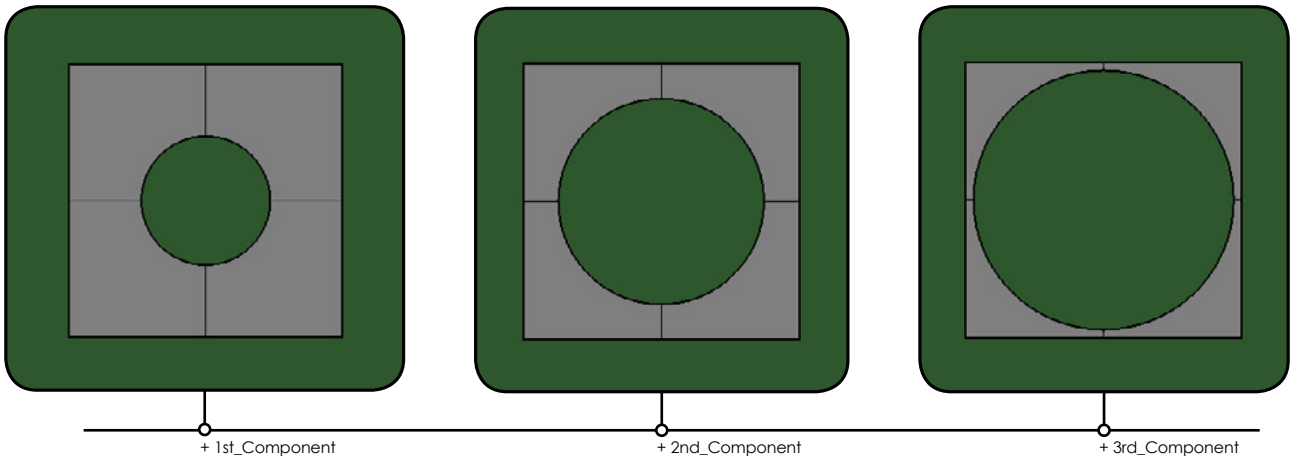
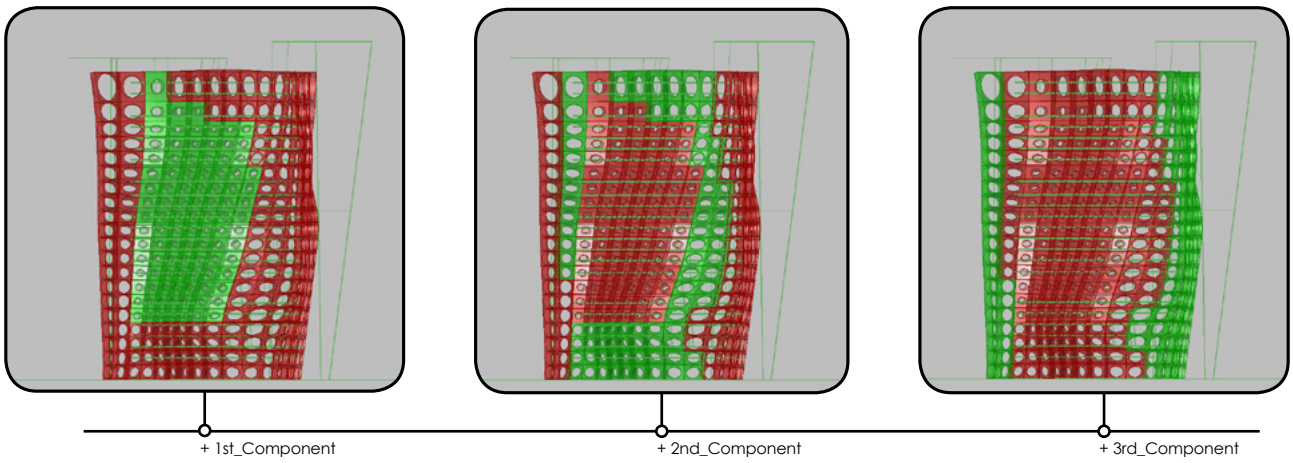


Fig.10.12. The angle differences has been compared with different predefined values and based on the Boolean data of these comparisons, target boxes dispatched into three different data lists, each for a different morphing.



Micro Scale with 3 different component types



Macro Scale with 3 different component type distributed based on the angle differentiation between the predefined sunlight vector and component's position.

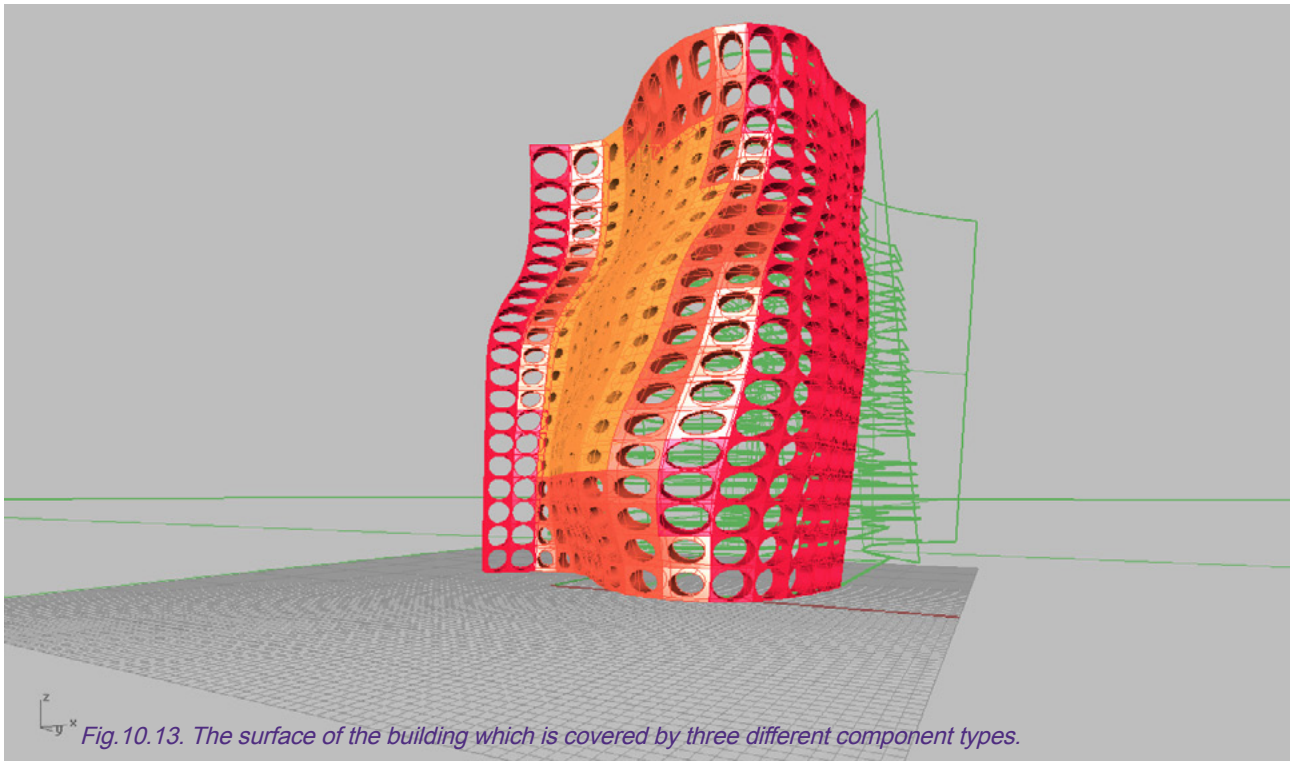
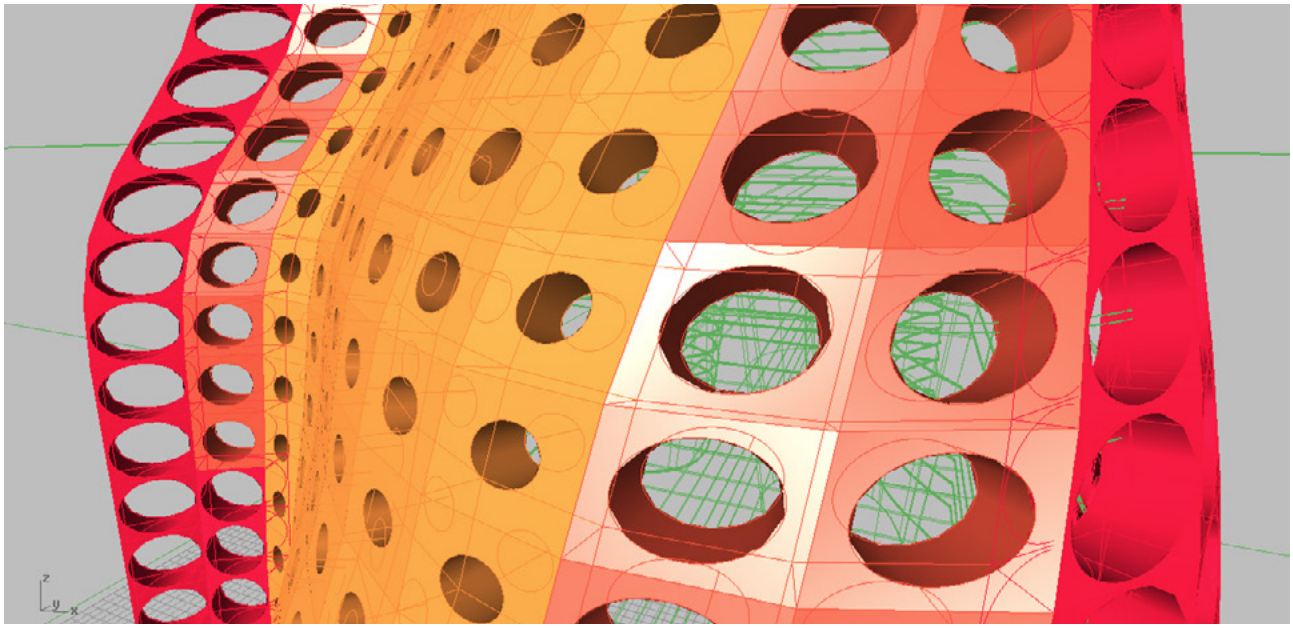
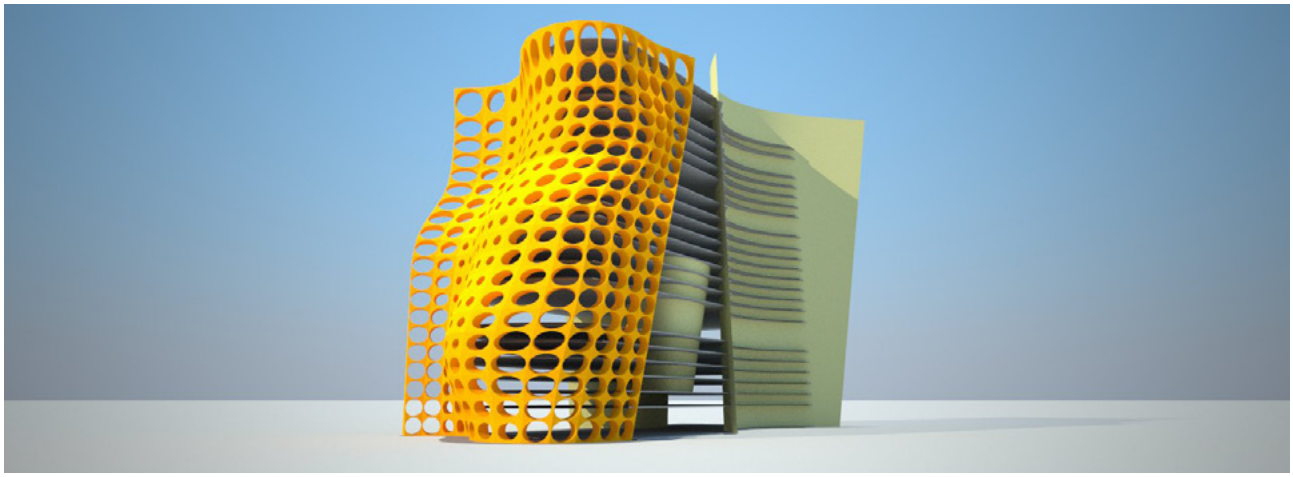


Fig.10.13. The surface of the building which is covered by three different component types.





CHAPTER ELEVEN

OPTIMISATION

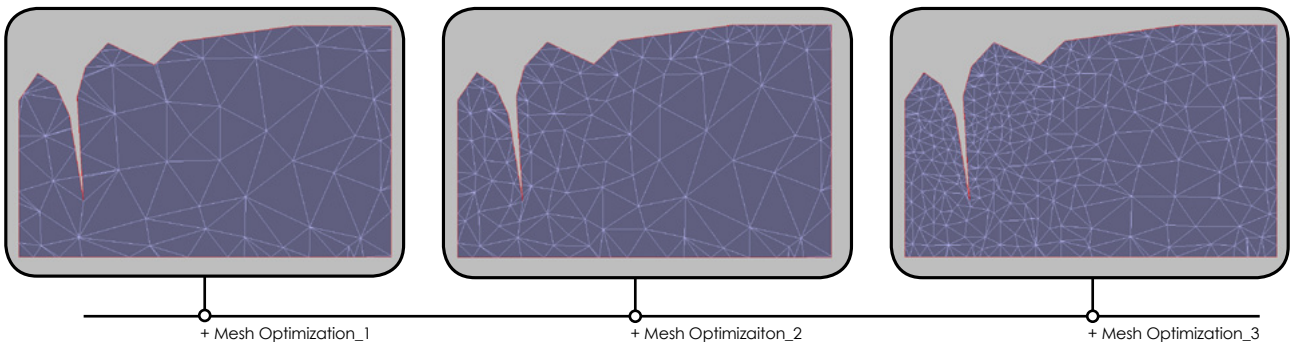
Chapter Eleven
Optimization

11_1_Optimizing Architecture

What is the aim of architectural criticism? If one tries to make it simple can say criticising architecture is for improving it; to push it towards better future; to strength positive achievements and highlight negative points to be omitted; to visualize 'dreamland' and lead the practice towards that desired imaginations. How it is possible? Critics are looking at built structures and try to find out their high and low points. They figure out qualities as much as short comes. But here is the point. Digital technology made it possible to virtually 'realize' a building, before its real construction. Now architects model, visualize and rapid-prototype their design products which enable them to criticise it from various aspects, far behind the point of construction in time, far beyond the elaboration of naked eye in detail.

CAD, 3D Modelling Tools, Rapid Prototyping Technologies, Augmented Reality, BIM, Virtual Media Environments are among the technologies that facilitate to digitally realize architecture before its physical realization. While critic was desperate to move around a built structure, digital technology made it possible to criticise architecture before construction, a self-informative, self-improving, self-criticising process.

Crisis of Post Modern styles and a shift towards digital era is changing the focus point of architecture from theory to new values. This has been discussed and addressed in various papers and journals recently. Neil Leach insisted on a "significant shift in emphasis from an architecture based on purely visual concerns towards an architecture justified by its performance" (Leach, 2009). This shift of emphasis coupled with the development of digital technologies which helped to set up a new language for architectural criticism: Criticising Behaviour and Performance. Criticising 'Behaviour' is different from criticising 'Beauty' and it needs its own language and syntax. Emphasis on criticising here means a step more than just engineering analysis for physical improvements of buildings. It means defining new design criteria, new approaches and even new values.

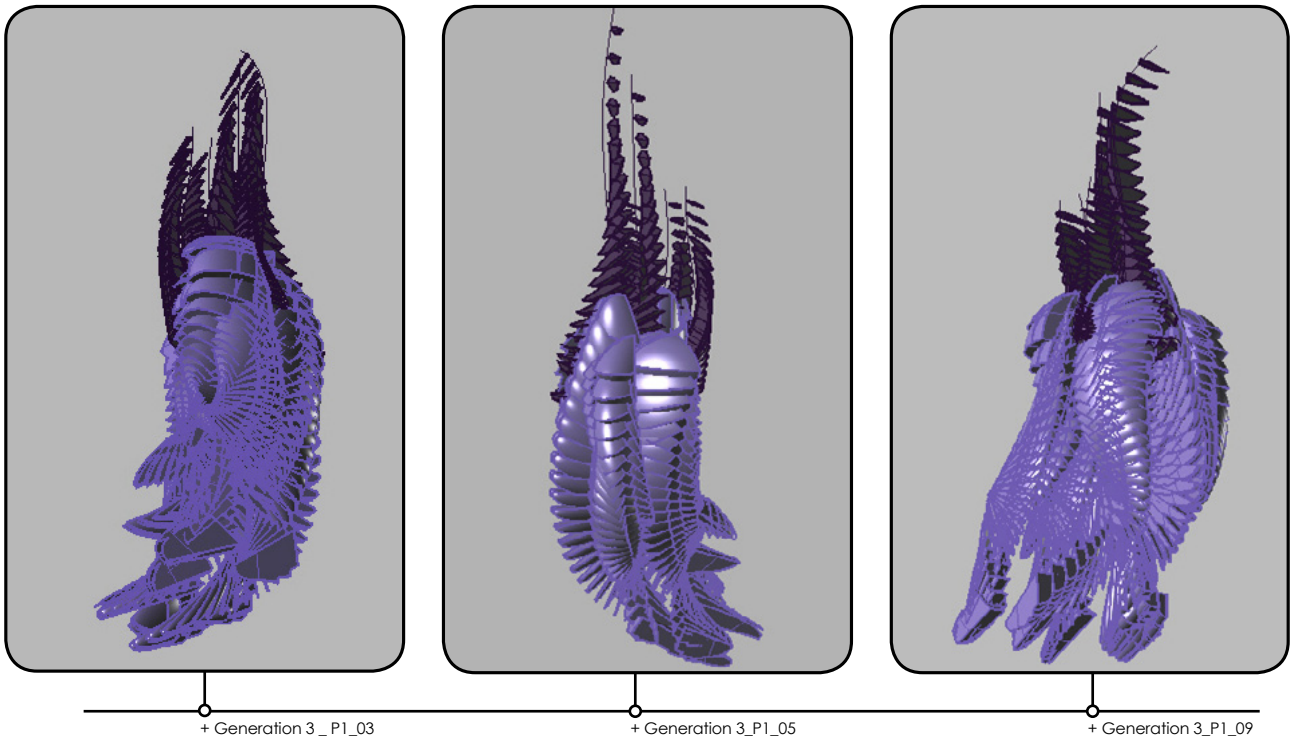


Mesh Optimization to increase/decrease faces to refine mesh for different uses.

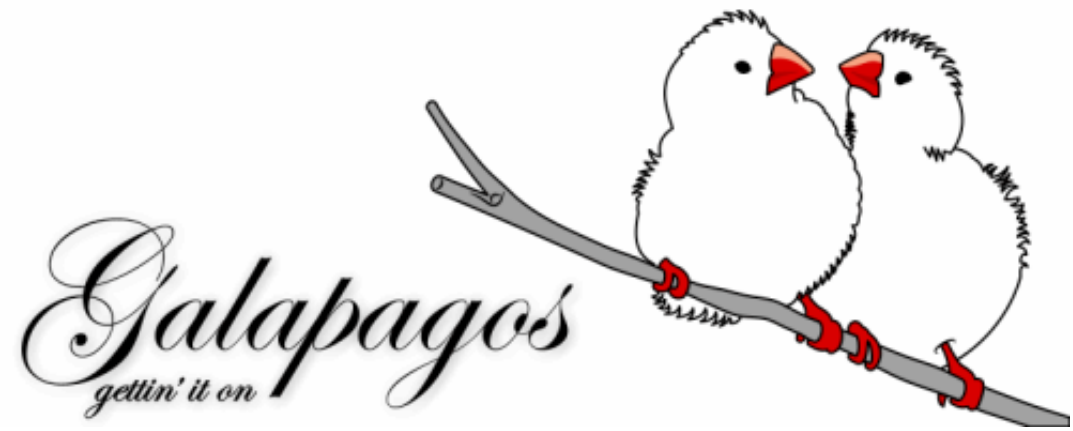
The architects' growing interests on dealing with material systems, material behaviour, bottom-up approaches and building performances caused their deviation from concept/theory based architecture. While the last one was more top-down and in favour of visual qualities, it entangled with the theory based criticism a lot. During this period, criticism was more about theory rather than technique, large scale rather than detail. Time has changed and architects are becoming more interested in small scale material systems rather than large scale theory based formal accumulations of 'boxes and cones'. This change in interest which is happening in the design methodology and fabrication technology, consequently affects the criticism.

Values behind the design of a unique building which wanted to last for ever or aimed to be monumental is now becoming blurred and there is a quest for more sustainable design, a building with higher performances in its material and energy consumption and optimized from various aspects. In this sense, talking about 'Optimization' in architecture shows how architects are getting away from the idea of unique building design and how they tend to move towards family of forms in order to select the 'optimized' one. Evolutionary computation and other optimization techniques in design technology approves that architecture is no more a unique product, but it could be generated in populations, improved in generations and selected based on its performance criteria.

An architecture product of evolutionary algorithms and feedback loops has design process and criticism process in a coupled platform. Criticism becomes an active paradigm in the formation of architecture. 'Better Future' which should be addressed by critic is now described through 'Fitness Functions', 'Selection Criteria' or 'Feedback Loops' and actively improves the quality of product. The responsibility of the critic is to set up all these criteria as comprehensive as possible, relevant to the current issues of sustainable development and other human concerns.



Form Generation in Evolutionary Computation using rules of floating lifestyle as fitness criteria



11_2_Galapagos

Grasshopper is equipped with its first solver, Galapagos, so far which works based on Evolutionary Computation. As David Rutten, the developer, says "It is my hope that Galapagos will provide a generic platform for the application of Evolutionary Algorithms to be used on a wide variety of problems by non-programmers", we can assume a solver which acts upon generic problems in Grasshopper which is possible to find optimized solutions in various design processes.

Talking about a solver, we know that we have to clearly define a problem and we should also address the answer to that problem as well. To find a solution for a problem with various variables, a defined fitness functions is needed. This means that optimizing a design product is possible while a fitness function can tell us we are going towards better/more optimized solutions. We need something, a criteria, to evaluate that our design product is responding well to that criteria. We are optimizing design based on such criteria.

Galapagos can find a solution to a defined problem with defined fitness function. The criteria with which we want to optimize our product should be numeric in order to pass it to Galapagos and ask it to solve the problem. These criteria should be numeric, to be able to increase/decrease it in loops, to see we are going towards better/worth conditions in comparisons. It is possible to set the solver to find solutions for higher/lower values to be closer to the fittest options. So a design product will be optimized or a problem could be solved by using a numeric fitness function in its highest/lowest possible value.

Let's see how this solver works in a simple example in order to be able to go further. In the design filed there are two curves as tangent curves, between them a circle wanted to be fitted with the maximum possible area. The <Circle Tan Tan> can draw such circle between two tangent curves by using a guide point close to the centre of the circle. Although there are various ways to find this point, here a very simple strategy has been used to keep everything simple. A curve has been drawn manually between the first two curves and presented in Grasshopper by a 'Reparameterized' <curve> component. Evaluating this curve by a slider from 0 to 1 will generate a point across this curve and this point could be a suitable guide for the centre of the CTT circle.

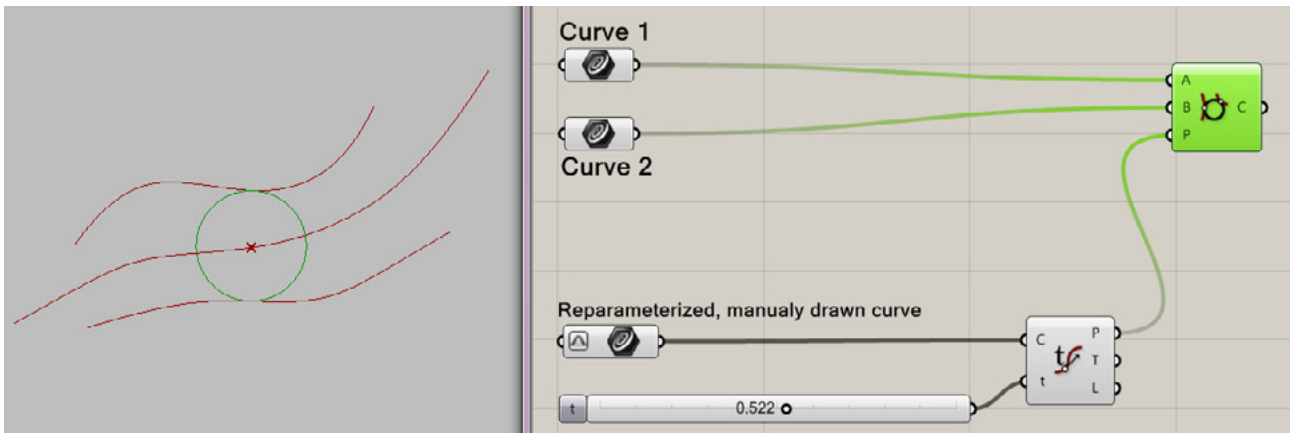


Fig.11.1. Looking for the biggest possible CTT circle

Now it is possible to navigate across the middle curve and check the area of the circle which is generated between two tangent curves. But which parameter of 't' would generate the biggest possible circle? How one can get that parameter? Here is where we have a design problem which is finding a circle between two tangent curves and a numerical criterion which is the biggest possible area. Galapagos can help to find this circle. It will ask for two different sets of input data to operate; A Fitness value to push it towards highest/lowest and a slider variable, a Genome, where its change can result in the change of fitness. While both connected, it is possible to run the solver by double clicking on the Galapagos component and opening its window.

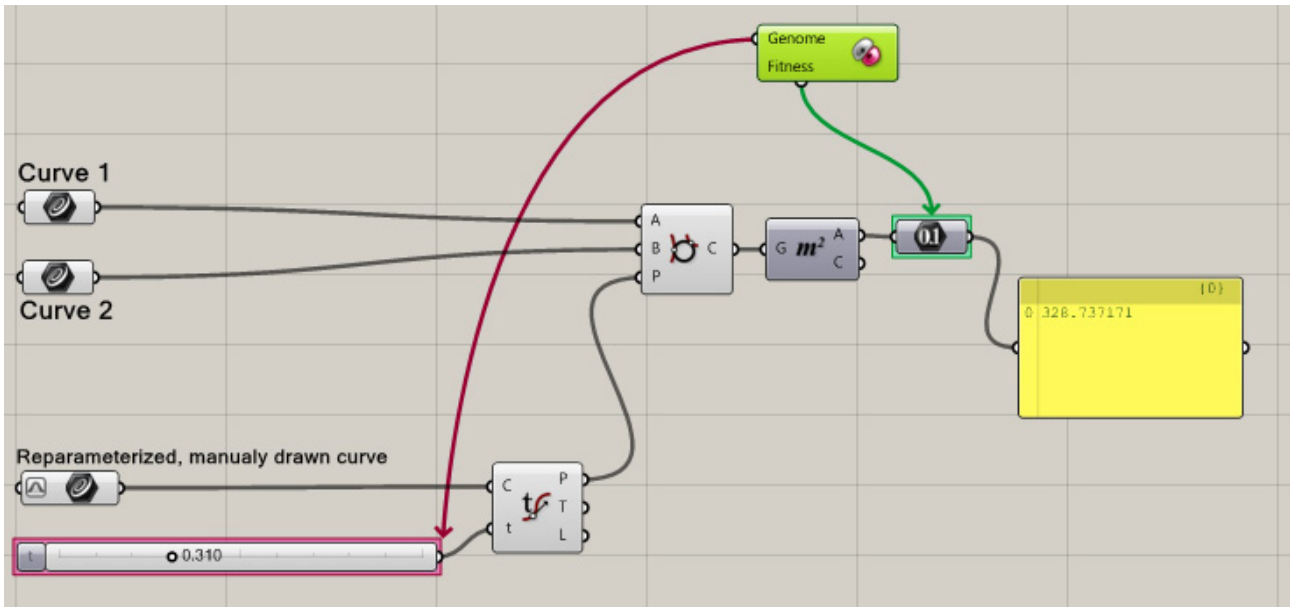
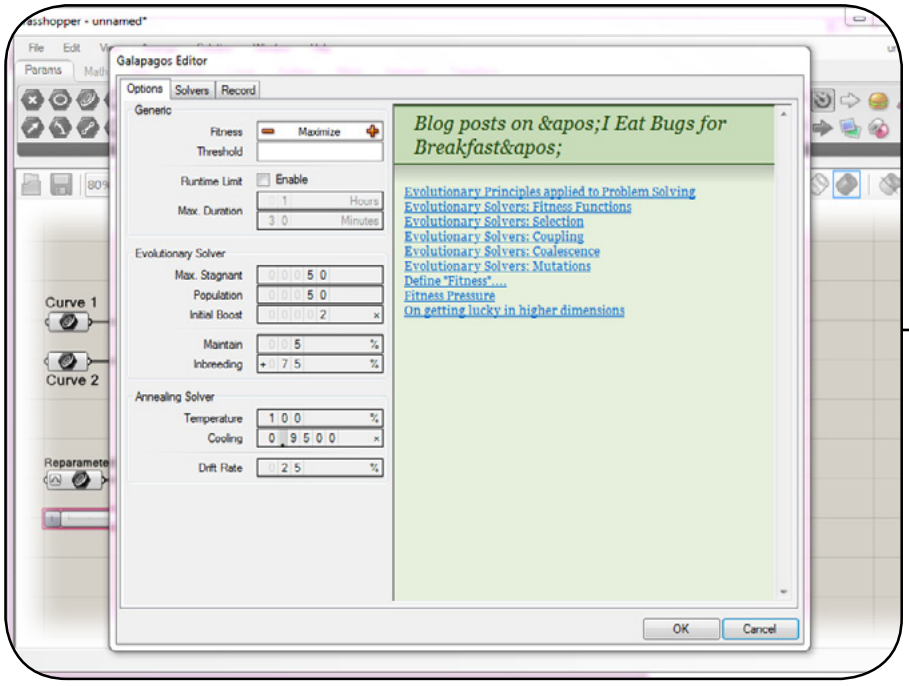
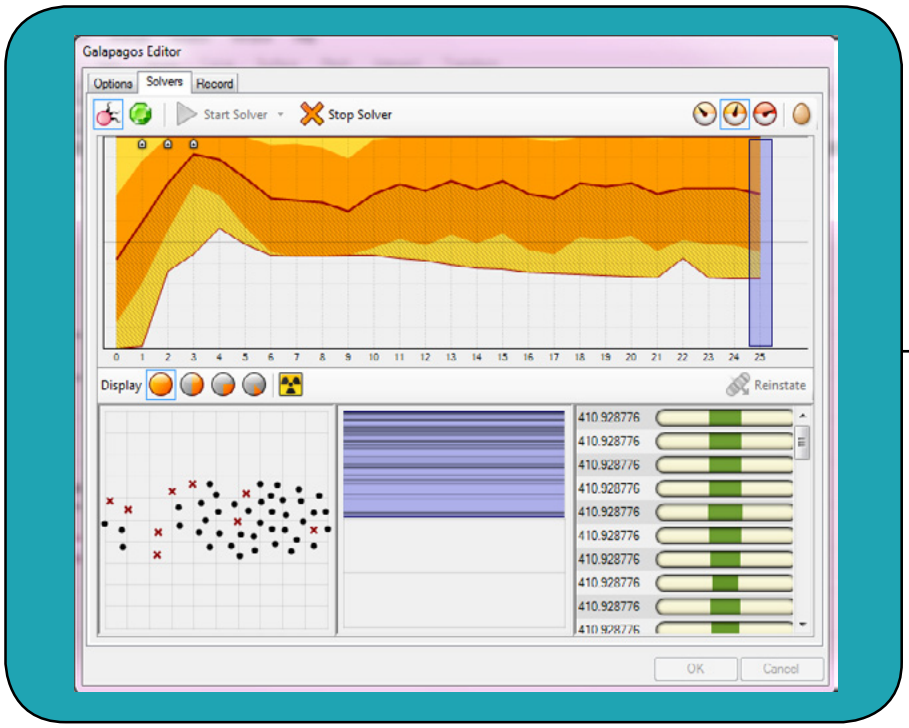


Fig.11.2. Galapagos with two input ports: A. a Genome to mutate the product (Slider) and B. a Fitness value which is a number that Galapagos tries to increase/decrease to get the fittest.

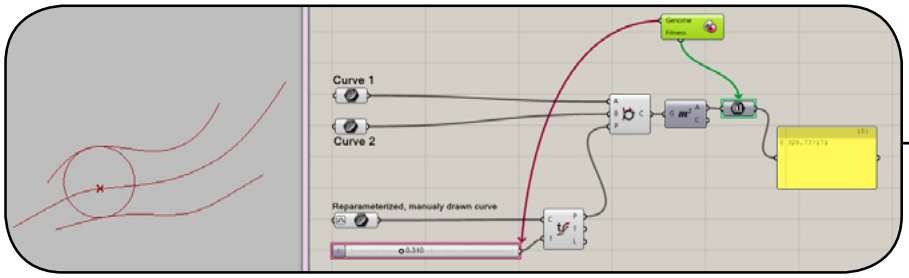
In Galapagos window we can define various elements of the solver which can globally set the fitness function. It is possible to set 'Maximize' or 'Minimize' that generally push the fitness value towards a highest or lowest value. Other options will also control the fitness function and the way it operates on the progress of the solver. Now if we hit the 'Start Solver' in Solver section, it will start to find the solution. There are links to valuable reading materials by David Rutten in the right side of the Options section of the solver which well describe how this solver works.



+ Galapagos Solver
Options of the Solver
(Fitness)



+ Solver interface



+ Finding biggest circle
(the slider is set now)

Now it is possible to imagine that the tangent curves provide more possible places in order to generate a CTT circle as a solution. Here for instance I used a <Genes> component to provide various mutable numerical values which has been used as parameters to evaluate the middle curve for guide points generation. The solver stops at more than one location in the middle curve and there are more circles in the range which can provide big areas in comparison with other circles around them. This will help to find more available solutions for one problem in various locations of the design field rather than sticking to the first available solution.

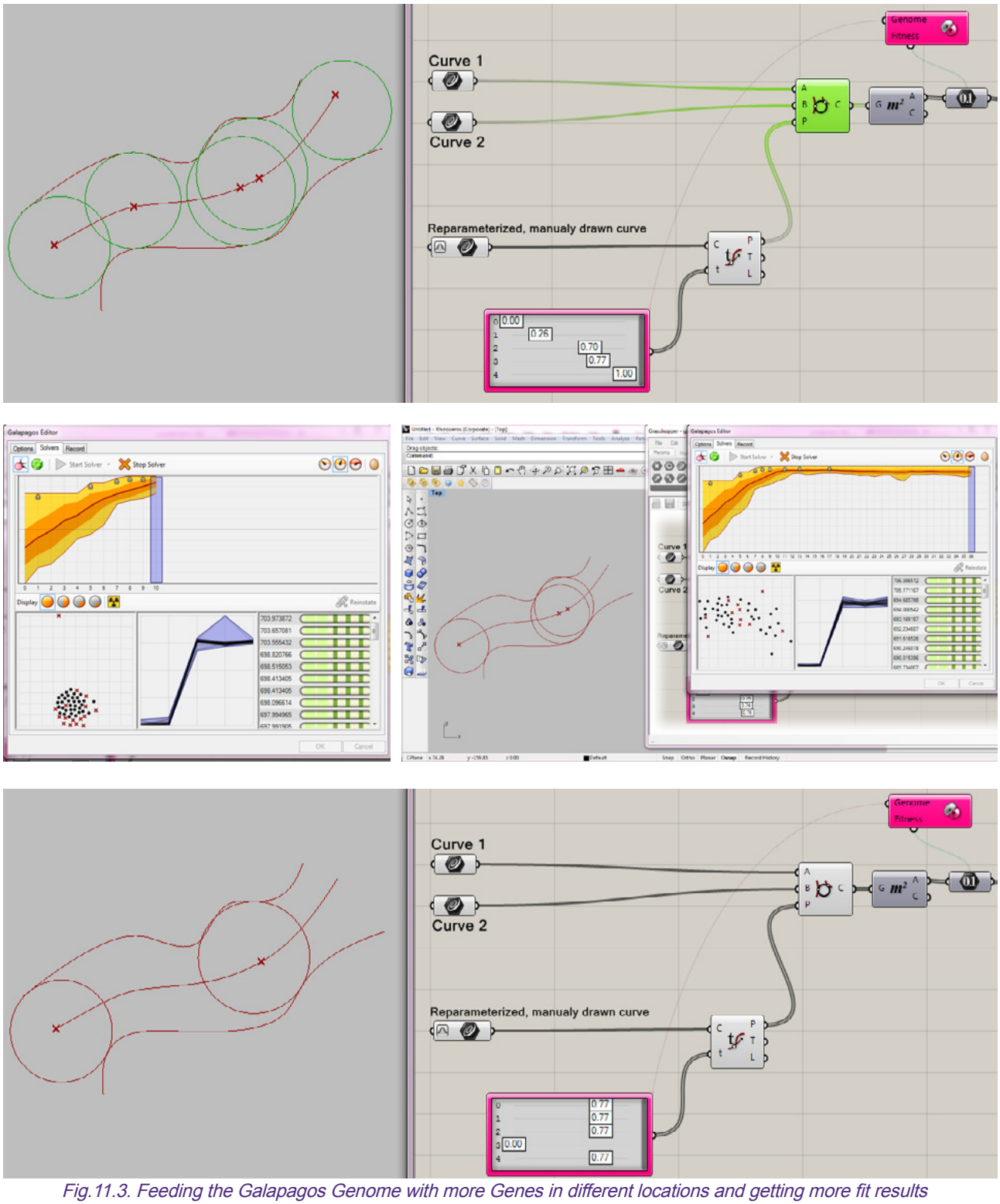


Fig.11.3. Feeding the Galapagos Genome with more Genes in different locations and getting more fit results

11_3_Fitness Function

The process of finding an optimized solution for a problem encounter a fast-forward part of generating populations of answers and then slow-down part of evaluating them against the fitness criteria to select 'fit' ones for the next generation. Galapagos does it internally. As mentioned before, dealing with this production and evaluation in Galapagos is numeric so far. Geneses should be mutated through numbers and the result should be evaluated by numbers as well. The solver would mutate Genes in order to achieve higher ratios of fitness criteria.

In most cases, setting up the part which encounters geometry generation is easier than defining the fitness criteria. Fitness criteria are the combination of interests towards them we like to push the project. Although they define 'what the project should be', it is not always easy to convert these definitions into numbers and algorithms. We are not always dealing with direct, accessible numbers to optimize our project. Numbers are hidden in some cases. The important point is to find out which values of the solution are measurable and also viable to evaluate as fitness. There are various Analysis and Measurement components in Grasshopper that help us to measure and extract numerical values from objects. For sure development of the software will increase these possibilities.

There are methods and techniques that can help to convert values of beauty and visual concerns into algorithms. Techniques like Interactive Evolutionary Computation can facilitate the utilization of human evaluation in selection processes of fitness functions. But as mentioned before, here we are most concerned about performance capacities of the design product. We intend to use these solvers to find solutions for problems regarding environment, energy, material and so on. The designer's intention in this situation would be conversion of such criteria into evaluation processes which are measurable in algorithms and possible to convert into math functions. That's the way factors like light, rain, humidity, wind, … can be measured and their effect can be applied to the project. These functions will define how much pressure can be set for one factor, should it be amplified or not, how it should be combined with other factors and so on. These math functions will result in the formation of the project accordingly. An example would show it more tangible.

Fitness Function Example

In the design space there is a point cloud and around each, an ellipse wanted to be generated. The aim is to have these ellipses as large as possible (for example to have the maximum canopy effect underneath). But as much as they become bigger, they overlap each other. The question or the problem is 'What is the biggest area for each ellipse with the minimum overlapping?'. Once again we are dealing with a design problem with certain fitness criteria: maximum size, minimum overlapping. It is easy to set up the design algorithm up to the point that we measure the surface area of all ellipses. But how we should measure or analyse the overlapping? Here, as one possible solution, the number of intersection points of each circle with rest of circles has been calculated and it has been tried to keep it minimum.

There are three values that Galapagos can operate on as Genes: Radius 1, Radius 2 and the angle of the ellipse (base plane). As it is defined in the fitness function section, the global area of all ellipses has been calculated but then the square root of this value has been used in the function. The number of intersection points also calculated, but then multiplies by (-x) in order to amplify its negative effect in the fitness function. The sum of these two values is the fitness which is aimed to be maximized by Galapagos. As far as ellipses become bigger, they overlap and generate intersection points which have a negative effect on the fitness value. So the algorithm tries to enlarge and rotate each ellipse as far as it does not overlap the others. Changing the negative power of the overlap factor would result in the change of solution; bigger ellipses with more intersections or smaller ones with less intersection. Based on the canopy area and number of intersection for various outcomes, it is possible to decide which option is suitable.

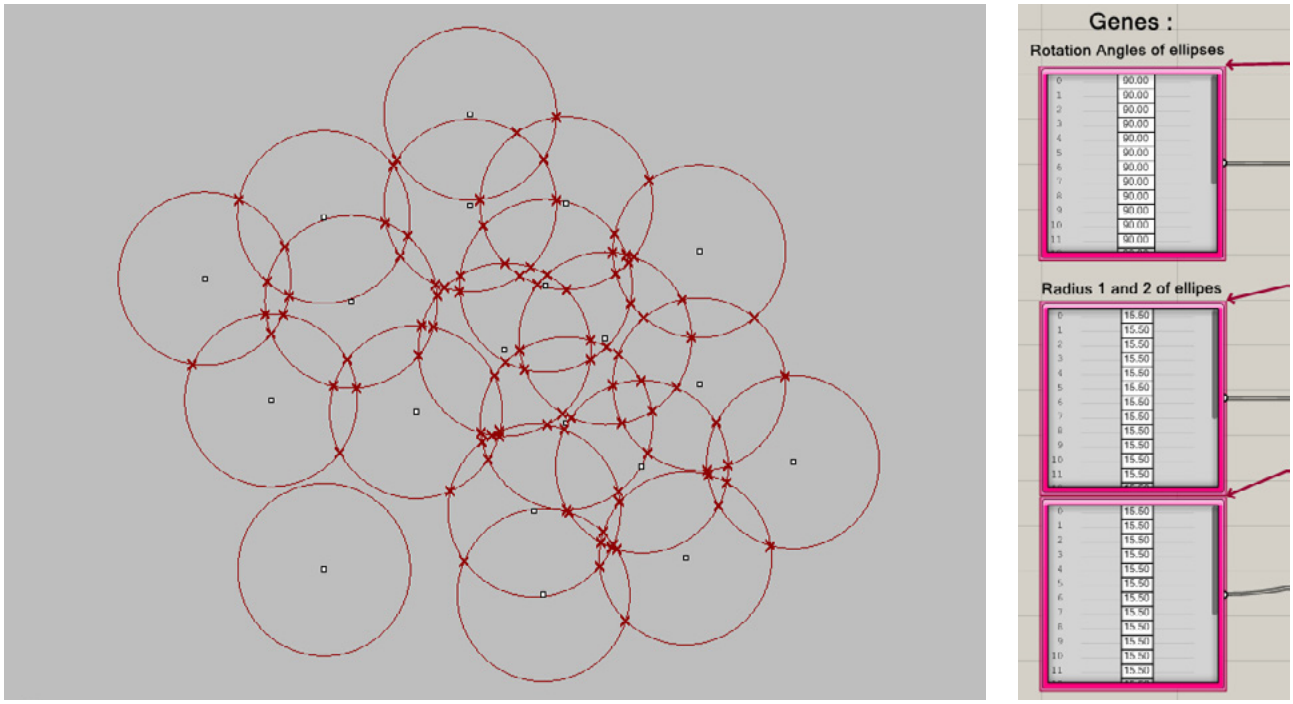


Fig.11.4. The initial state of ellipses

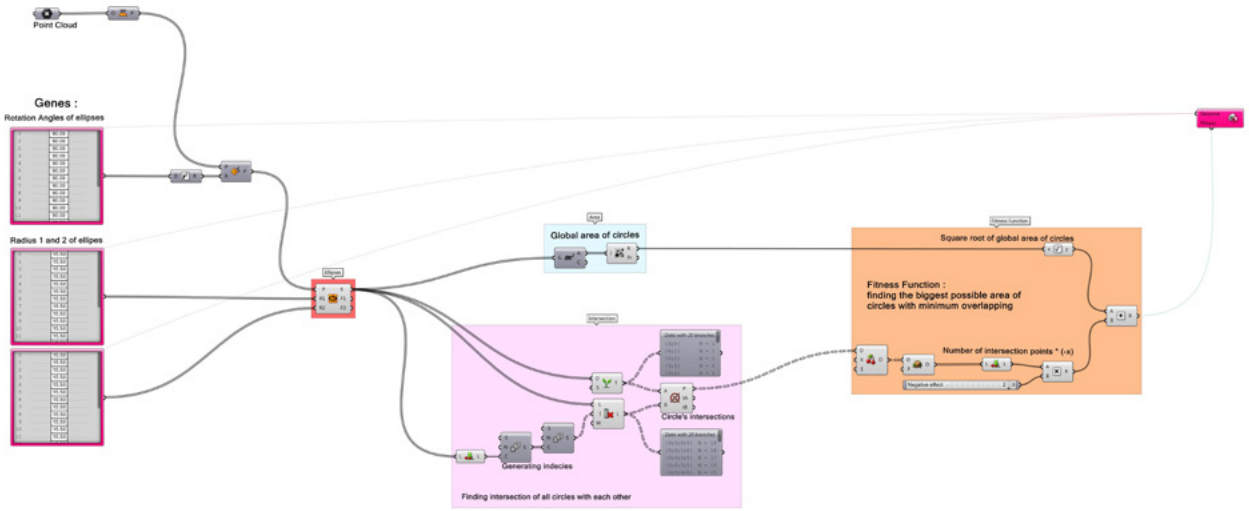


Fig.11.5. Optimization Algorithm

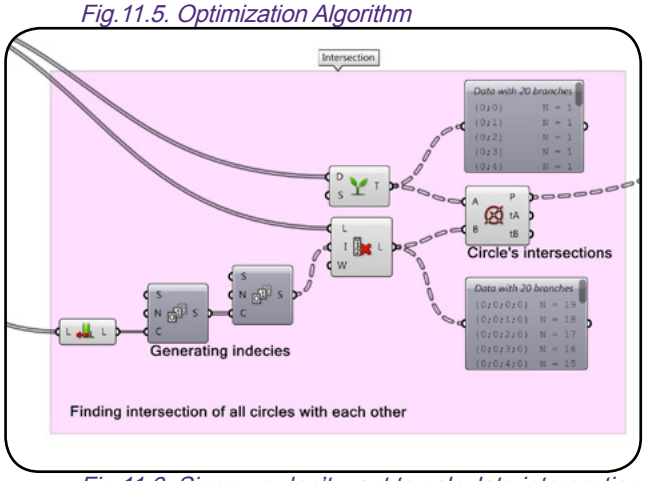


Fig.11.6. Since we don't want to calculate intersection points more than once, we need to cull ellipses little by little, calculating intersections for the first ellipse with the rest, the second one with the rest without the first one, third with the rest without first and second one and so on.

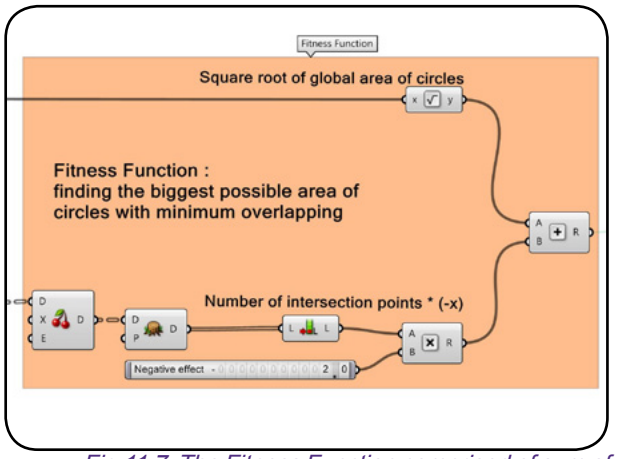
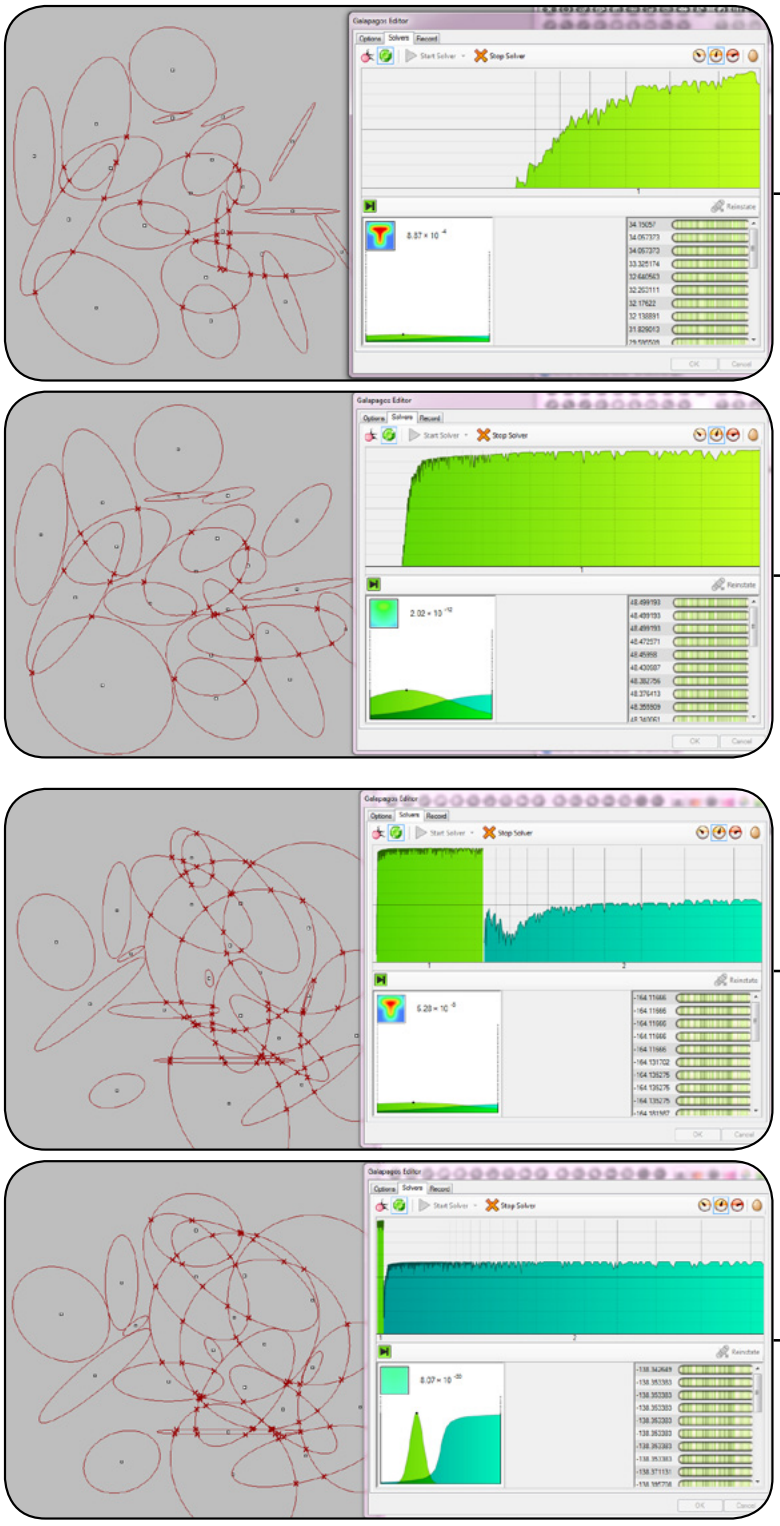


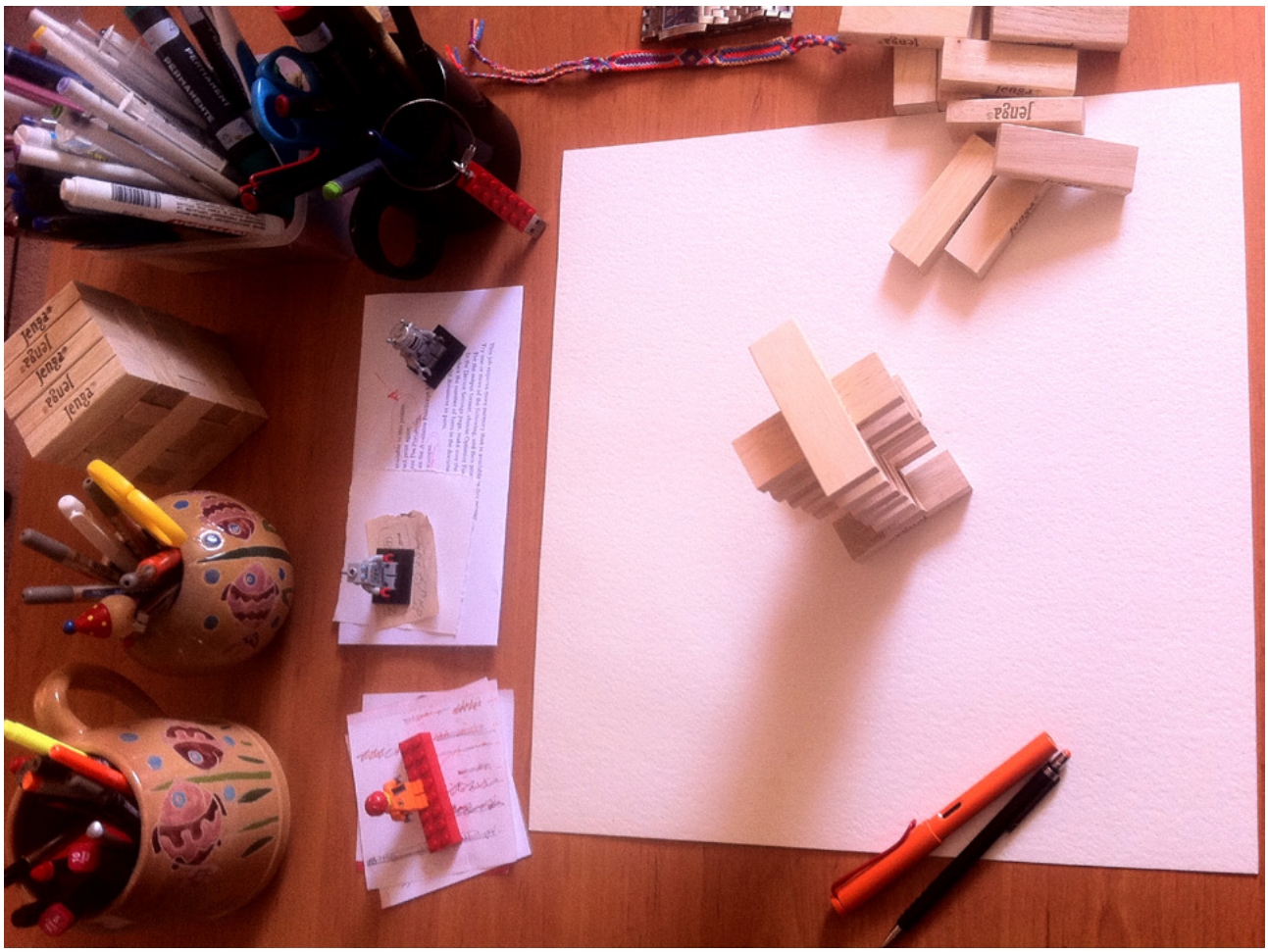
Fig.11.7. The Fitness Function comprised of sum of two main parts: 1_Global area of ellipses which is delivered to a square root to reduce its power and 2_Number of intersections multiplied by a negative number to generate a negative effect on the fitness value.



Any form that we design has properties like size, area, volume, coordinate, curvature, normal direction, angle difference from a base vector, distance from a base position, height …. Among the list, there are certain values that should be used for evaluation of objects against different criteria. Once it has been realized that each property of the object is responsible for/in relation with what performance/behaviour of the object, it is possible to define the fitness as a mathematical function of those values. An example could be like:

$$f = 1.33\ (A_S) + sqr\ (S_w) + Ln\ (w_o / v_o)$$

In which (A_S) represents the angle difference with sun direction, (S_w) is for the speed of the wind and (w_o/v_o) represents object's Weight/Volume.



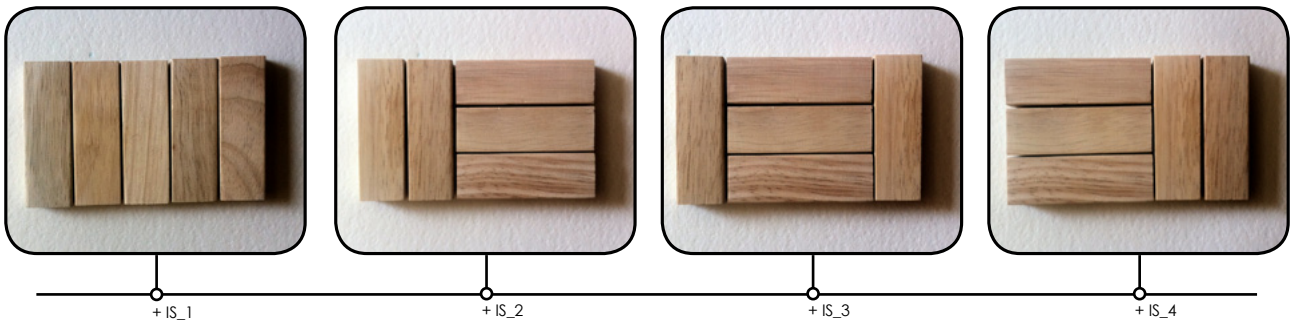
11_4_Optimization: Design with Feedback Loops

The idea of finding solutions or optimizing products based on the defined problem of the design is interesting for various reasons. It pushes designers to think about the logic of any design product, to formalize this logic with the language of math and geometry and to a constant movement towards better solutions by adding more and more criteria to evaluate their product.

Using Galapagos might help to find the solution based on a fitness which is possible to define mathematically in Grasshopper but there are also situations that further evaluations might be necessary after the process; For example evaluating the product again with other software for additional criteria and using the outcome to feed the algorithm again. Getting feedback from various analysis that we could test in our design products can set the feedback loop in which the product could be redesigned based on the data that comes from the interaction of the product with various external/internal stimuli. As mentioned before, while the design might encompass some basic elements, arriving at a point that an optimization process can work with certain fitness function might need fighting with the problem a lot.

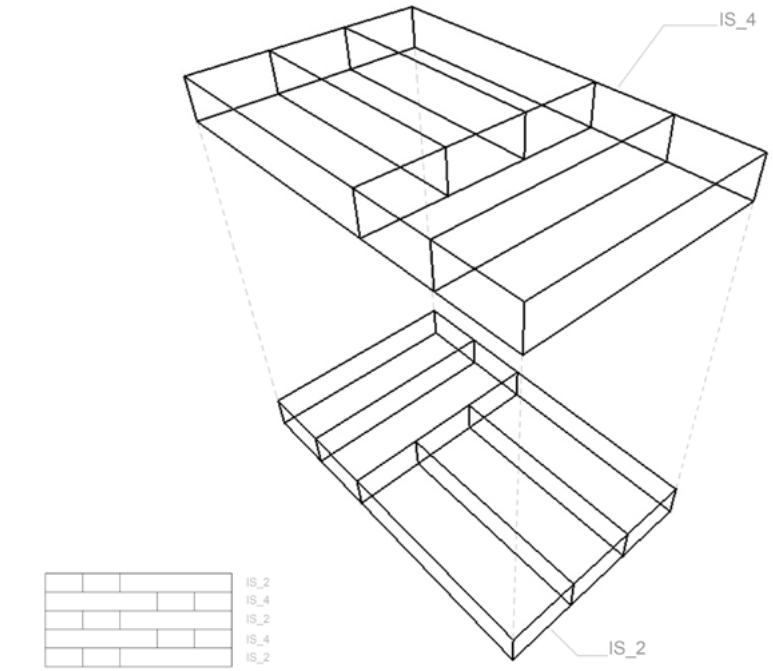
Jenga® Project

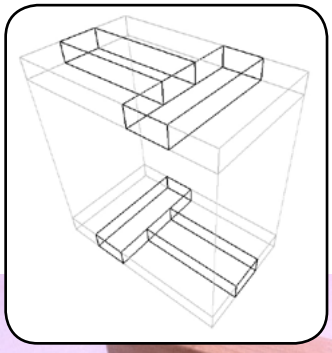
The project that I am working on it, is a game driven from Jenga! The initial setup of Jenga comprised of a tower with three blocks in each layer which their directions (longer length), perpendicular to the next/previous layer. Here in my game, instead of three blocks, I used a setup of five blocks in each layer. There are four different situations that blocks could be packed in each layer in order to set a 5-block assembly product.

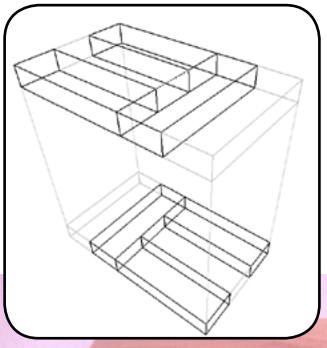


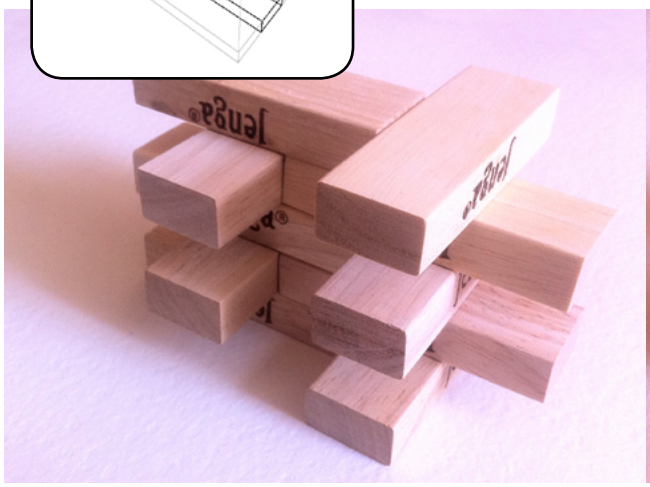
4 different initial states of the blocks in a 5-block assembly / layer

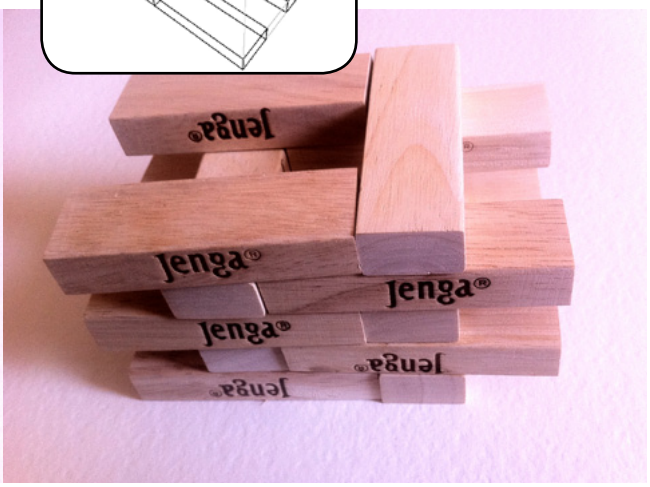
Now layering these initial states on top of each other to set up a mass and starting to remove some of the blocks (without any desire to put them back on top of the mass yet) would result in various types of accumulations of interconnected Jenga blocks in a porous or Habitat-type of forms. Let's start the game with a setup using only two initial states, 2 and 4. The layers would be 2, 4, 2, 4, 2 in order to simplify our fist experiment. Removing blocks could happen in various orders, two of them are represented here, both following a repetitive pattern.











The most important rule of designing patterns of positioning or removing blocks for this experiment is related to their stability. In terms of stability of blocks, to remain in their positions in the structure, not to fall down, there are some very simple, obvious rules. There are five main conditions that would result in a block stay stable in its position (the position of each block remains constant during the process in this experiment). There are three situations in which a block rests on the other one and remain constant, because the (G) vector (a vector from block's volume centroid towards earth) goes through another block. There are two other situations that would result in a stable block, one when the block has two other supports at its two sides (the Bridge condition), and another one when it has been cantilevered and has enough weight on top of itself to keep it constant as well (the Cantilever condition). Although in both cases the G vector goes through an empty space, the block remains stable in its position.

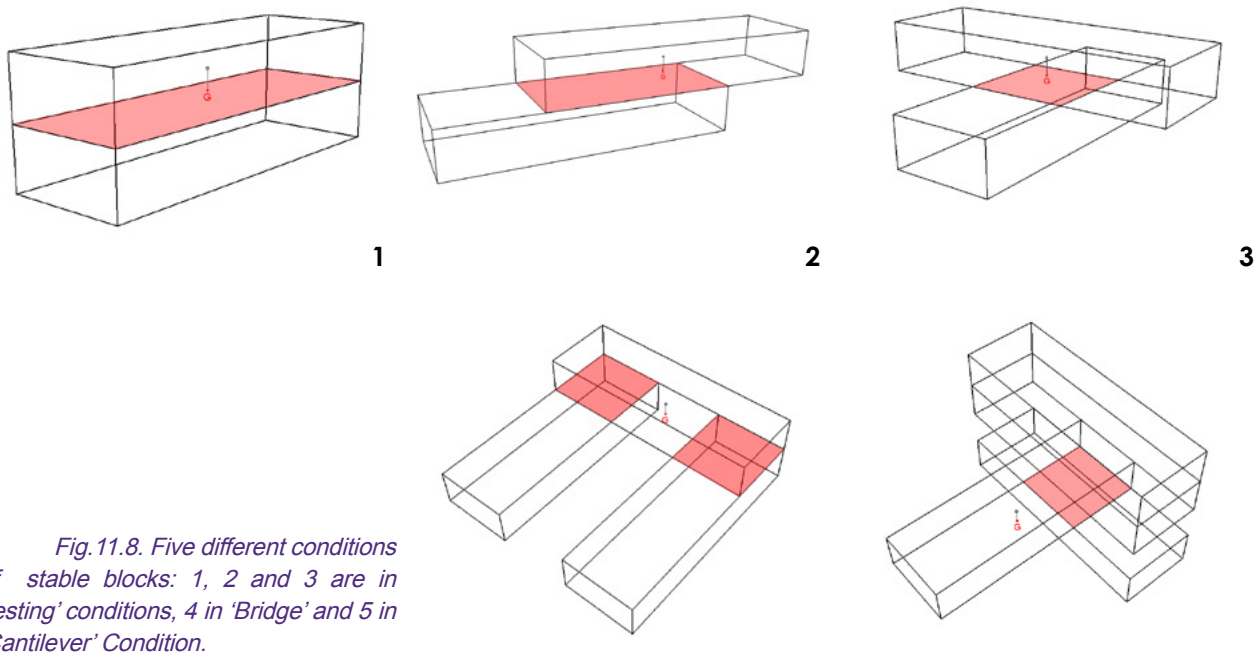
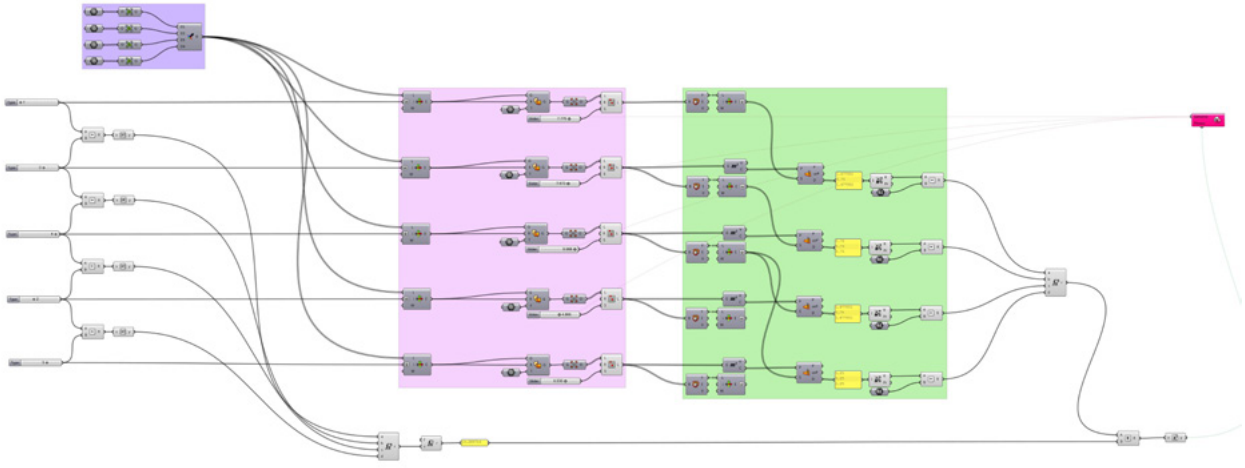
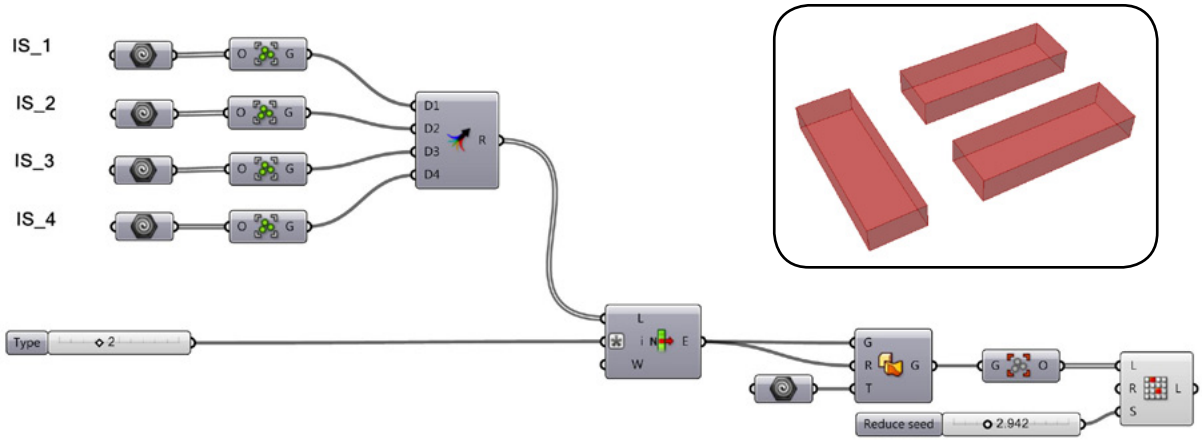


Fig.11.8. Five different conditions of stable blocks: 1, 2 and 3 are in 'resting' conditions, 4 in 'Bridge' and 5 in 'Cantilever' Condition.

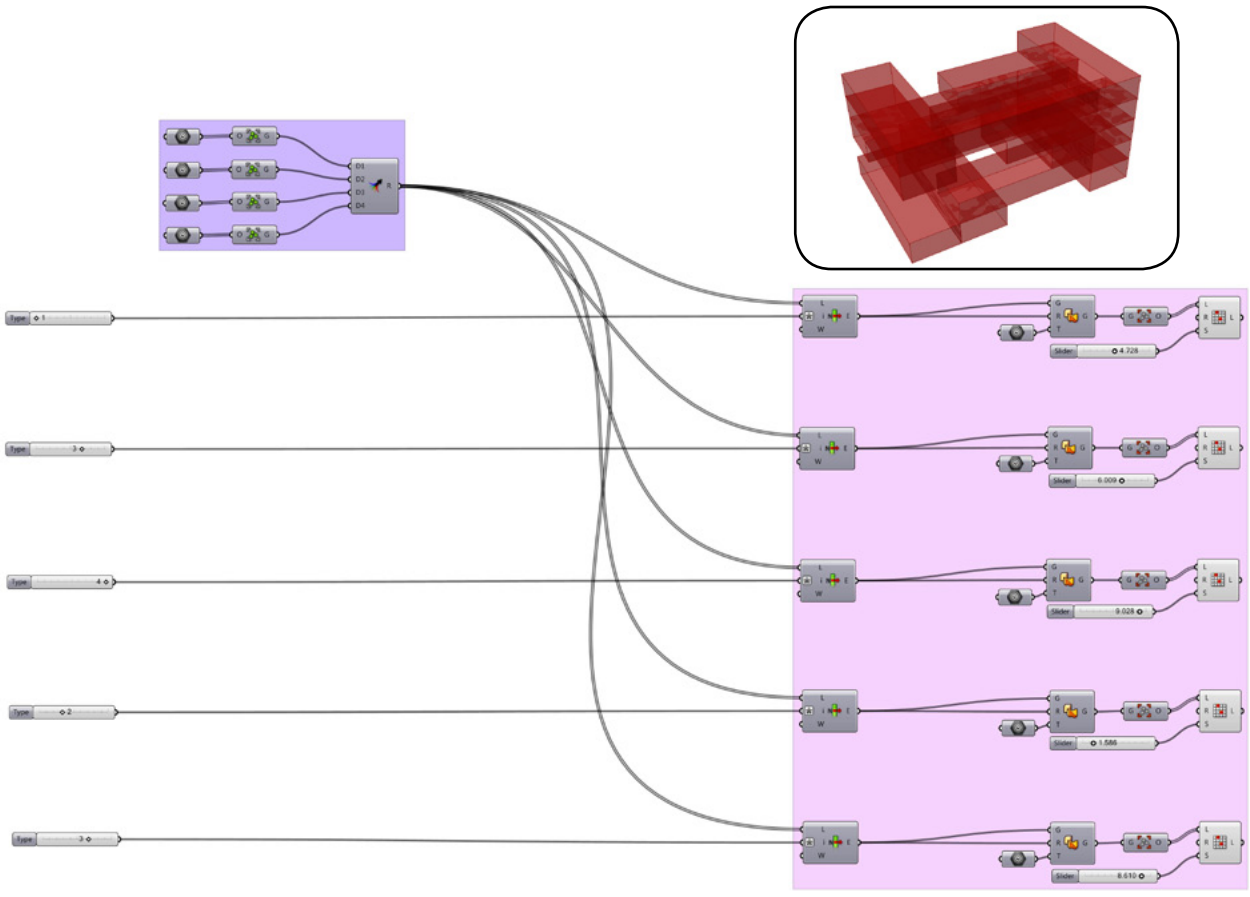
The idea is to convert this game into an algorithm. I am designing an algorithm in which I want to have five layers of Jenga blocks, each comprised of five blocks with four different initial states. The aim is to randomly remove blocks from layers and get a stable structure at the end. In order to design the algorithm there are two processes needed; one to distribute blocks with various initial states in different layers and remove some of them, two is to control if the structure is stable or not (a design problem and a criterion to evaluate the product).



For the design part, four different initial states of blocks are created by boxes in Rhino and imported as geometries in Grasshopper as <Group> of objects. These groups are merged and then connected to a <list item> component with a <slider> that generates numbers 1, 2, 3, 4 with a little function of (i-1) to generate indices for 'initial state' selection. Changing the slider would result in four different outputs as initial states of the blocks. I have generated five bounding-boxes on top of each other and introduced them into canvas by five different <Geometry> component, morphed selected initial state into first target box. Using a <random reduce>, I removed two blocks from the layer. The seed <slider> would result in the different patterns of removing.

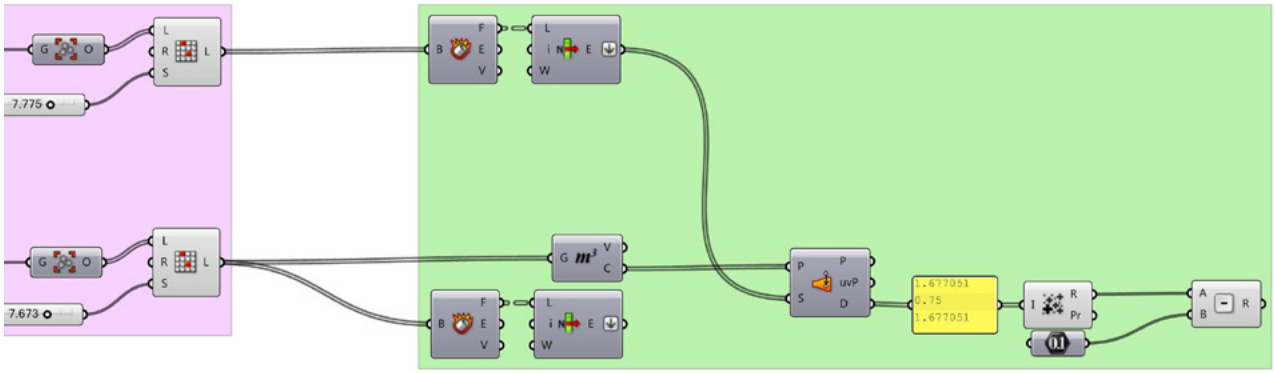


Now I copied the above process five times, with five different target boxes and so far I have generated all five layers of blocks on top of each other while 'type' sliders would change the initial state type in each layer and 'reduce-seed' slider would change the pattern of removing blocks. There is no guaranty that blocks would stay stable yet.



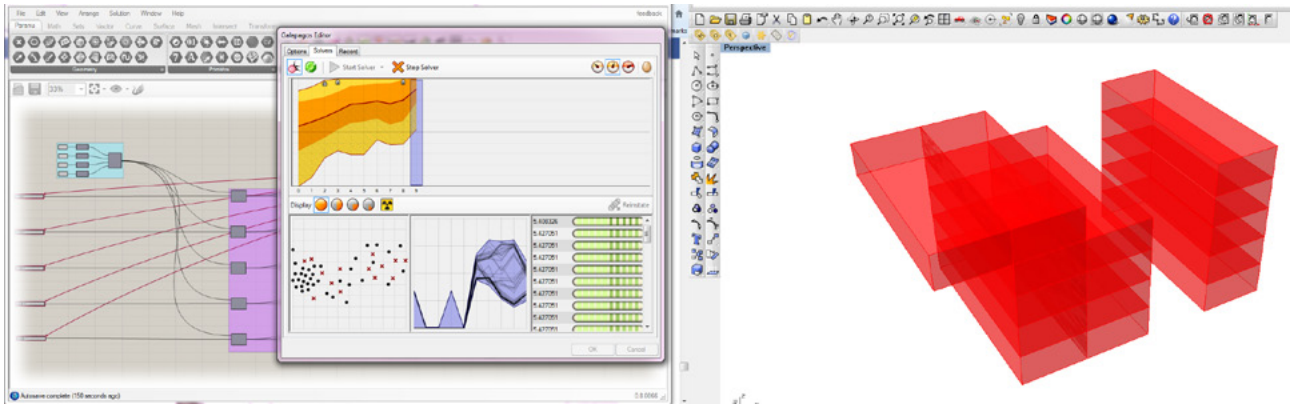
As mentioned before, the criterion which is aimed to evaluate this product against is its stability. There are five different positions for each block to make it stay, three of them introduced as resting, another two as bridge and cantilever conditions. The cantilever condition is not desirable in this experiment since it is not suitable for assembly. The bridge is omitted from the algorithm as well but the aim is to add this option for later stages. So there are only resting positions that are aimed to be evaluated and accepted as stable positions of the blocks.

But how we should analyse these resting positions? As mentioned earlier, in all resting conditions, the 'G' vector of each block, a vector that starts from block's 'volume centroid' towards earth is going through another block in underneath layer. This means that if one divides the block into three parts, the middle part always rests on top of another block. But how we should analyse this in math? There are multiple ways, here I used the distance between the block's volume centroid with the closest point on top surface of the underneath block. If this distance equals half of the block's thickness, it means that the block rests on top of another block. The <mass addition> of these distances for three blocks should be equal to 2.25 (half of block's thickness, 0.75 * 3 = 2.25) to make sure that all three blocks are in resting position. So any number more than 2.25 means that there should be block(s) which are unstable and we should look for better options. So the value of deviation from the predefined 2.25 (for each layer) will be used as the fitness value of the algorithm.



Asking Galapagos to solve this problem, there are two sets of genes that Galapagos could operate on, in order to change the result: Type sliders and Reduce-seed sliders. Changing the initial state of blocks and seeds for those two that should be removed from each layer in a combination for five layers would be all possible options that Galapagos would search for the most stable option. Galapagos would try to 'Minimize' the fitness which means it would try to find those situations in which all blocks (except layer one) are rested on top of another one (the best value for all stable blocks should be 0).

The problem in running this function is that Galapagos would push all layers to get the same type, all removing blocks, the same pattern as well. This would be a totally stable product but not a porous and interlock one that I am looking for. There should be mechanisms in which pushing all layers 'to have the same initial type' should be assigned with a penalty that affects the fitness.

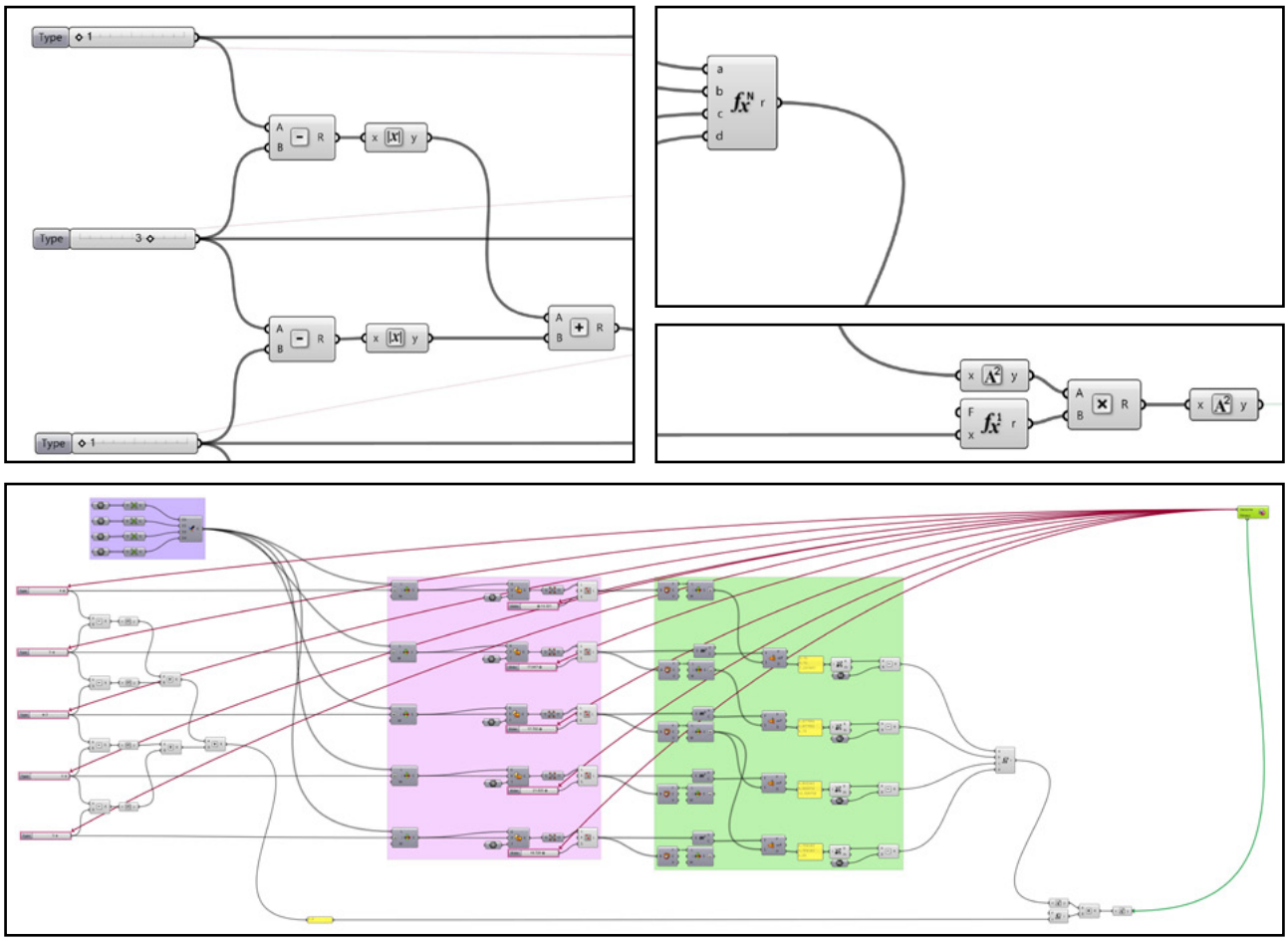


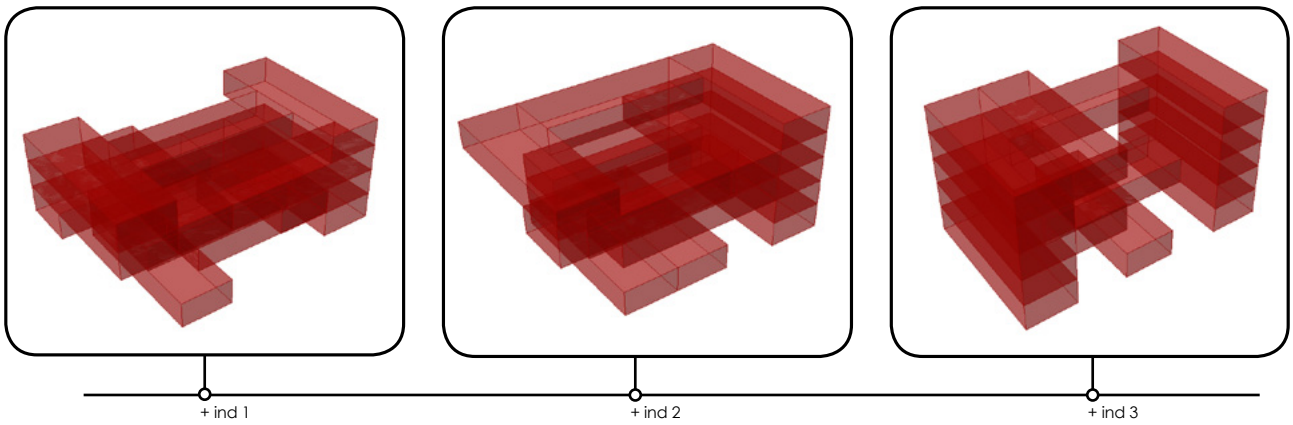
To do so, the difference between the types of any two adjacent layers (their slider values) has been calculated and all these values are summed up and affected the fitness by this function:

$$f = [\ Dev_b * 1/(Dif_t^2+1)\ ]^2$$

Dev_b = Sum of Deviation of each block from 0.75
Dif_t = Sum of difference of initial state type between any two adjacent layer

If type sliders differ from each other, **Dif_t** would be a large value which cause **$1/(Dif_t^2+1)$** become a very small value (less than 1). This would cause the fitness value become smaller and smaller. But if type sliders become equal, then **Dif_t** would be 0 and **$1/(Dif_t^2+1)$** would be 1 which means global fitness would be a large value which is only affected by **Dev_b** .





Running Galapagos solver again, since there is a great penalty to push type sliders to be the same, the solver would try not to have same layer types in two adjacent layers. Parallel effect of these two factors would push the solver to find non-similar, yet stable combination of blocks. Based on the seeds that we can provide for the start of the algorithm, Galapagos can find results which are most likely to be randomly distributed, porous, but stable in structure. There might be some cases that a block would not be stable in the outcome but running function with different start values could generate different combinations with more stable results. One can add more pressure to the **Dev_b** part of the fitness function in order to make sure that all blocks are stable, although there might be the lack of differentiation at the output product. **That would be designer's call !**

CHAPTER TWELVE

design research

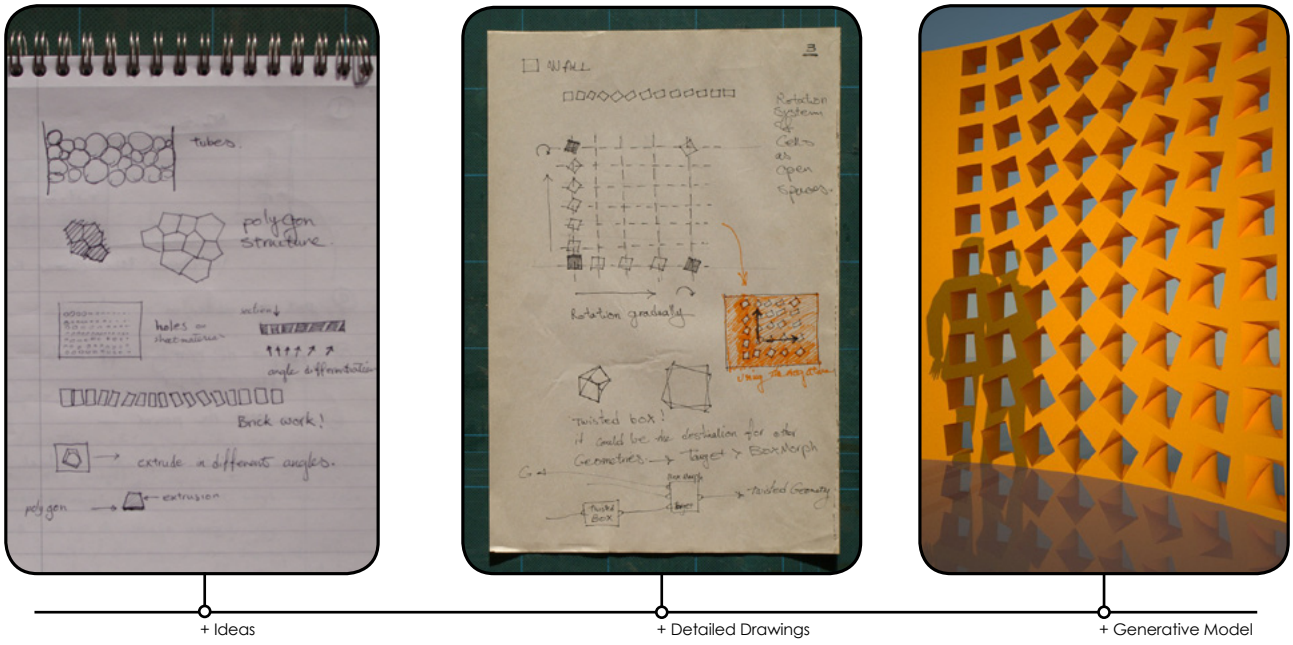
Chapter Twelve
Design Research

12_1_Desgin Strategy

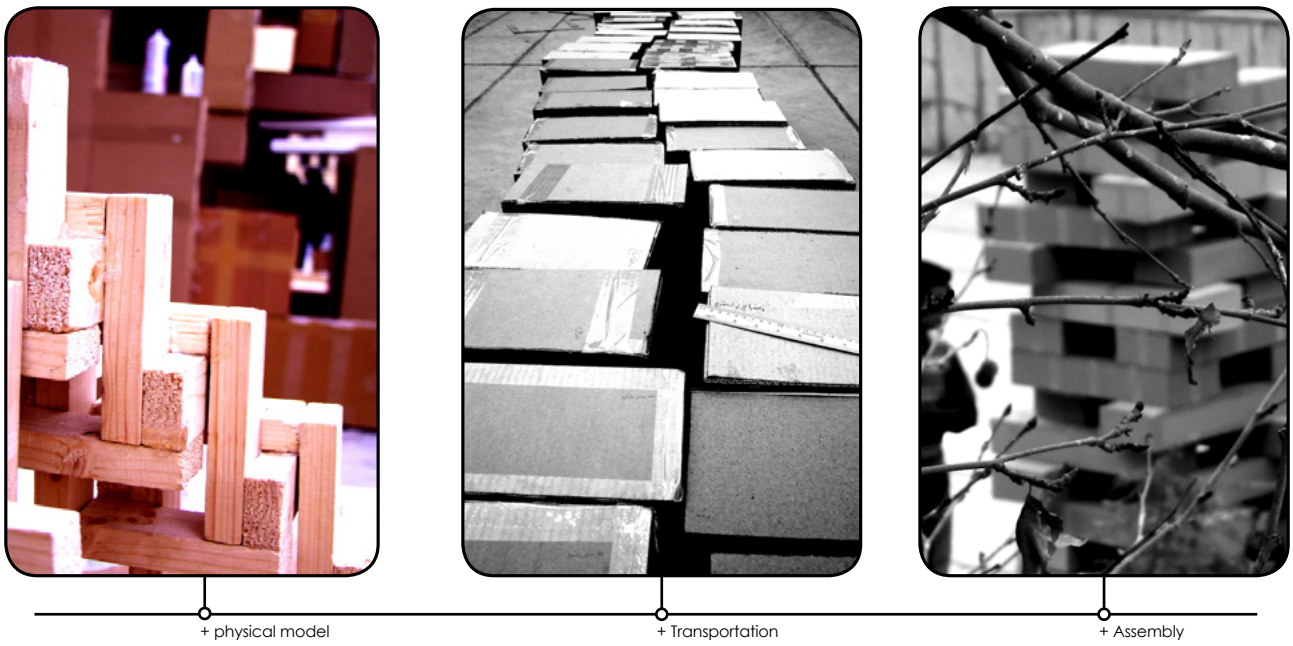
Any design project encompasses an Agenda; a Brief; a Program that shows what is needed; what should be designed; what are the aim and objectives. The design project would be a possible answer to the brief. Designer would think about the agenda; Is it about making a shelter? an enclosure? an envelope? Space filling or something else? Designer would set up strategies to respond to the project's needs and necessities. While dealing with algorithmic design, any design process should be equipped with strategies for fabrication and assembly of the project as well. The design/fabrication/assembly are inseparable and based on the relations of these steps, all design decisions should be made.

Design/Fabrication/Assembly strategies might consider different issues that affect the project in various ways:

- **Modelling.** Algorithmic and computational complexities in front of project
- **Structure.** The structural system. The structural behaviour of components
- **Material.** Properties of proposed materials. Positive and negative feedbacks
- **Performance.** The performance of the designed system. What should be implemented?
- **Duration.** The timespan that is aimed for the project to last
- **Scale.** What is the scale of the project and how does it affect other factors
- **Machineries.** Technicality, limitations and potentials of machineries engaged in the project
- **Transportation.** Limitations, considerations and costs of transportation
- **Energy.** The energy consumption of the project and ways to reduce it
- **Interaction/Adaptation.** The interaction of elements with each other or with external stimuli. The way project/elements adapt to such stimuli and interactions
- **Ecology.** What are the long term considerations of the project's life in its host environment
- **Economy.** What is the global economic situations of the project in short/log term

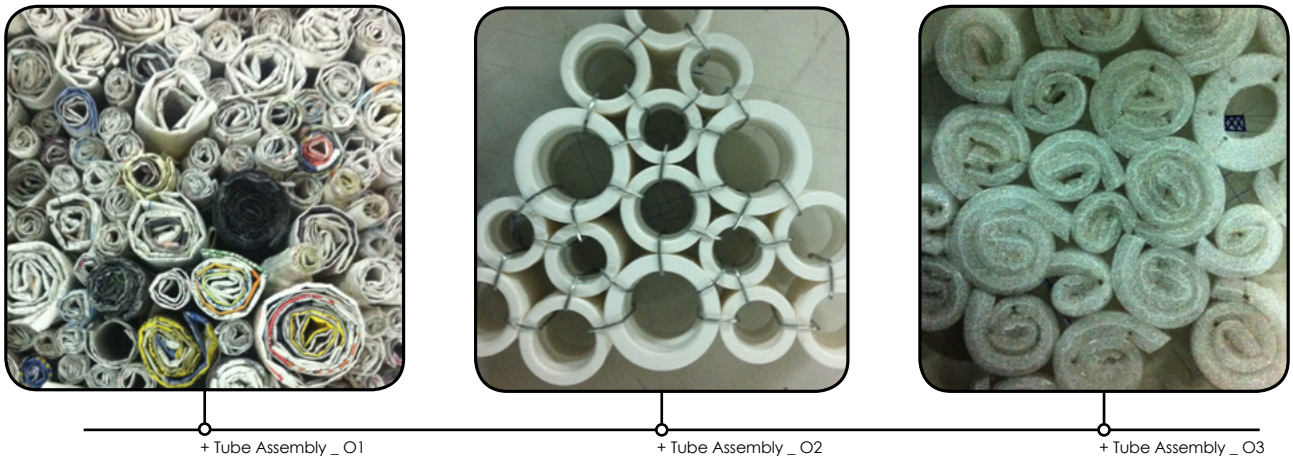


Porous wall drawings and model. [morphogenesism]



Modular wall Project. [morphogenesism]

The designer's strategies might respond to all of them but with different priorities. There might be direct design responses or just considerations. These strategies then should be converted into algorithms. What we have experimented so far in this book were not the best possible methods for sure. This was an educational text for step by step experimentation with algorithmic design by Grasshopper. But the way designer can set up design algorithms would not be that different. It might encounter some parts of the strategies, methodologies and techniques which have been used so far. The point is to see what is 'important' for the project, what sort of information is available as 'input' to operate on, and what should be provided as 'output' in order to set up the design algorithm. It would be always helpful to analytically understand the design problem, sketch it, make physical models and look at problems from various aspects before diving into any algorithm. It is always better to start from simple but strong logics and then make it complex step by step instead of creating complexity from early steps of design. Strong projects and intelligent algorithms lay their foundation on a very comprehensive understanding of the logic of any design action and an approach to solve it from very simple stages towards higher degrees of complexity.



Looking at various material system properties: tube assembly. [morphogenesism]

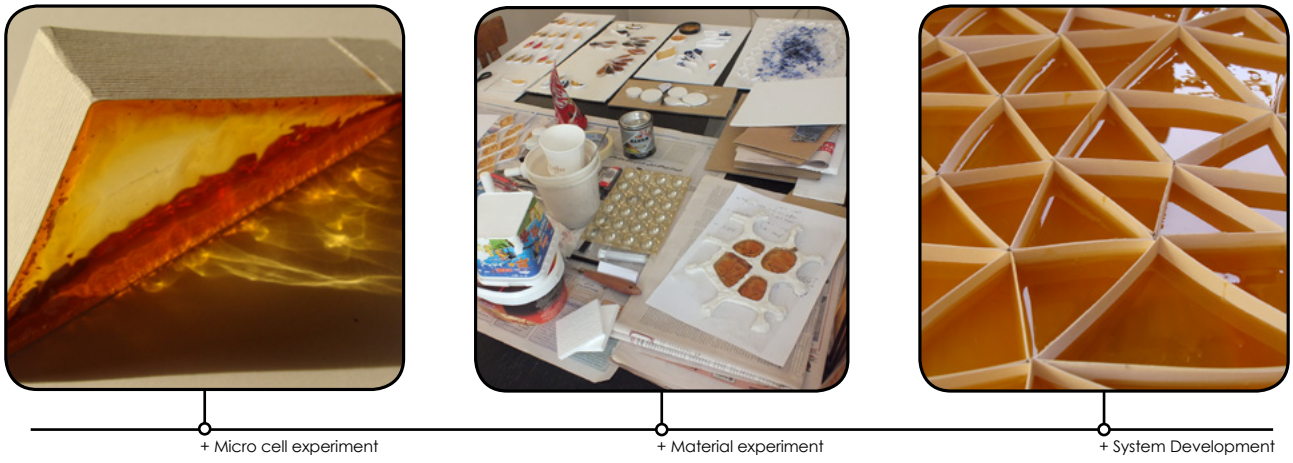
12_2_Design-Research, the Methodology for Innovation

What is the current state of innovation in architecture?

Looking at the above list of issues that should be considered in the process of design/fabrication/assembly, there are hundreds of subjects in each, for research. While it mentioned that architects try to deviate from a theory based practice, research in, and experimentation with material systems, fabrication technologies, new materials, new design techniques,… become a great source of creativity and innovation for contemporary architecture. Any design practice with emphasis on experimentation in such issues seems to have new methodologies, values and techniques, in each part, there are research areas that push the boundaries of the knowledge, creating new spaces for innovation. As Patrik Schumacher in his manifesto for 'Parametricism' says: "Avant-garde styles might be interpreted and evaluated in analogy to new scientific paradigms, affording a new conceptual framework, and formulating new aims, methods and values. Thus a new direction for concerted research work is established. My thesis is therefore: Styles are design research programmes".

Thus any practice in contemporary 'design' which include algorithmic thinking and designing have the potential to be engaged with 'research' activities, their convergence as 'Design-Research' would not only support the innovation and creativity for architectural design, but also pushes the boundary of design practice towards new areas for further investigations. This new areas could come from other disciplines of science via the same language of algorithms. While any branch of science tries to use algorithms as a basic language for communicating with computers to fast forward the progression of the knowledge, these algorithms could migrate to other branches of science and enrich them with new methods and techniques. Architecture in this sense has a great chance to use algorithms, techniques and solutions from other disciplines like mathematics, biology or genetics. This represents the current state of interdisciplinary and multidisciplinary knowledge that would change the practice a lot.

The future of the design practice would be shaped by Design-Research. It uses design algorithms and computation as a platform to apply and use innovations from architecture or other areas of science to create a new type of built environment for the future. The aim is to use all these new tools and techniques like Grasshopper to support these movements. This would have lots of potentials for innovation.



Material system development with experimentation on material properties. Translucent Cell Project. [morphogenesism]

Translucent Cell Project
Zubin Khabazi
morphogenesism

APPENDIX

Bibliography

Ball, P. (2009). Shapes, Nature's Patterns, a tapestry in three parts', New york: Oxford university press.

Camazine, S. and Deneubourg, J.L. and Franks, N.L. and Sneyd, J. and Theraulaz, G. and Bonabeau, E. (2003): 'Self-Organization in Biological Systems', New Jersey: Princeton University Press.

De Berg, Mark and Van Kreveld, Marc and Overmars, Mark (2000): 'Computational Geometry', Springer.

Dempsy, A and Obuchi, Y (2010): 'Nine Problems in the form of a pavilion', AA Agendas No.8, London, Architectural Association.

Flake, G.W. (1998): 'The computational beauty of nature, computer explorations of fractals, chaos, complex systems, and adaptation', Cambridge: The MIT Press.

Hensel, M. and Menges, A. and Weinstock, M. (2010): 'Emergent Technologies and Design, Towards a biological paradigm for architecture', Oxon: Routledge.

Hensel, M. and Menges, A (2008): 'Morpho-Ecologies', London: Architectural Association.

Iwamoto, L. (2009): 'Digital Fabrications: Architectural and Material Techniques', New York: Princeton Architectural Press.

Khabazi, Z. (2011): Generative Algorithms, Concepts and Experiments: Porous Shell. Online Publication: www.Grasshopper3D.com.

Khabazi, Z. (2011): Generative Algorithms, Concepts and Experiments: Strip Morphologies. Online Publication: www.Grasshopper3D.com.

Khabazi, Z. (2012): 'Algorithmic Architecture Paradigm', Kasra Publication, Farsi Edition.

Leach, N. (2009): 'Digital Morphogenesis', Architectural Design. Special Issue: Theoretical Meltdown, Volume 79, January/February 2009, London, Wiley-Academy.

Menges, A. (2012): 'Material Computation', AD special issue, March/April 2012, London, Wiley-Academy.

Otto, F. and Rasch, B. (1996): 'Finding Form: Towards an Architecture of Minimal'. Edition Axel Menges.

Pottman, Helmut and Asperl, Andreas and Hofer, Michael and Kilian, Axel, (2007): 'Architectural Geometry', Bently Institute Press.

Rutten, David, (2007): 'Rhino Script 101', digital version by David Rutten and Robert McNeel and Association.

Schumacher, P. (2008): 'Parametricism, A New Global Style for Architecture and Urban Design', AD Architectural Design - Digital Cities, Vol 79, No 4, July/August 2009, London, Wiley-Academy.

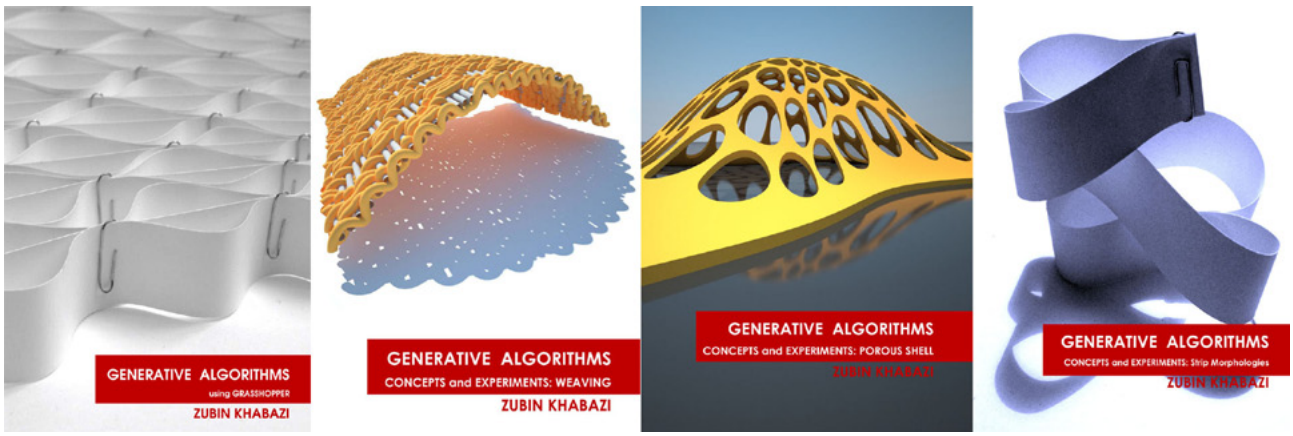
Terzidis, K. (2006): 'Algorithmic Architecture', Architectural press.

About the Author

ZUBIN KHABAZI, Architect, Researcher and Writer, is the founder of morphogenesism. He studied Master of Emergent Technologies and Design (EmTech) at Architectural Association, School of Architecture (AA) London, after previous studies at IAUM, where he granted Master of Architecture (MArch) with Honors. He has taught and lectured in different universities and schools of architecture and published papers and researches but recently broaden his research and design investigations through publications of 'Generative Algorithms' worldwide. 'Generative Algorithms' as a design research medium in the field of parametric architecture and algorithmic solutions, gained global attention by students and architects. In parallel he was involved in several architectural projects and worked for architecture companies mostly as a technology advisor and consultant and developer of algorithmic design processes. Investigations for system development in architecture via Computational Geometry, Algorithmic Solutions for complex material systems, and Digital and Parametric Form-Finding techniques are his current field of research and interest which he pursues in the experimental studio of architecture [morphogenesism].

Generative Algorithms

'Generative Algorithms' are series of design experiments which are aimed to develop concepts, theories, tools and techniques for algorithmic design. The aim is to share the knowledge of algorithmic design with all designers and practitioners who are active in this field. While the first publication of Generative Algorithms' focused on the basic principles of designing with algorithms (using Grasshopper as the platform), others were about specific subjects, each concentrated on design development and fabrication of a project or prototype (Waving, Porous Shell and Strip Morphologies, all available in Grasshopper webpage). This is the aim of [morphogenesism] to publish the knowledge of Generative Algorithms with new media for public use.



Generative Algorithms

(using Grasshopper)

Zubin Khabazi

morphogenesism

www.morphogenesism.com