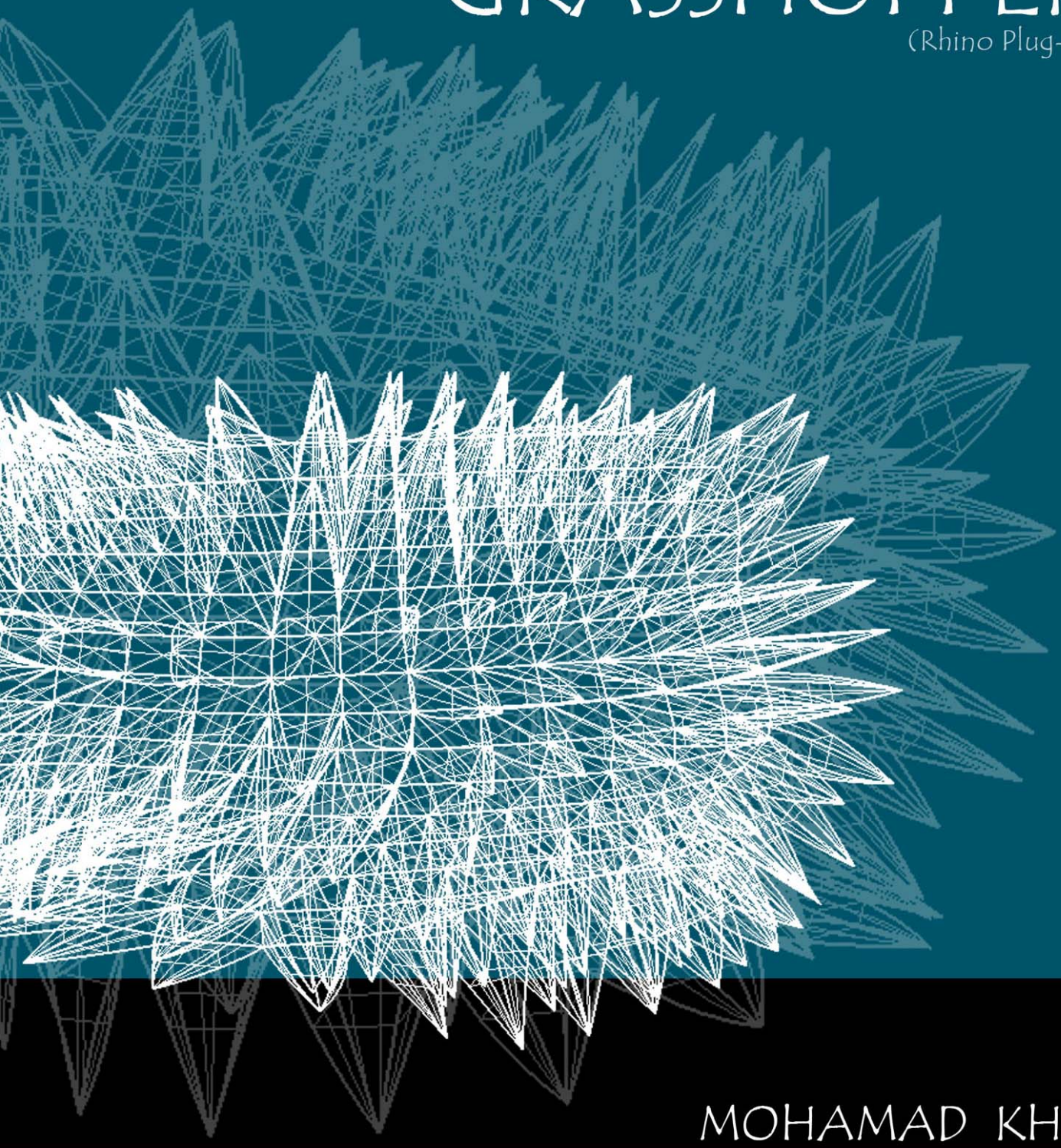
ALGORITHMIC MODELLING

With

GRASSHOPPER

(Rhino Plug-in)



MOHAMAD KHABAZI

Introduction

Have you ever played with LEGO Mindstorms NXT robotic set? Associative modelling is something like that! While it seems that everything tends to be Algorithmic and Parametric why not architecture?

During my Emergent Technologies and Design (EmTech) master course in the Architectural Association (AA), I decided to share my experience in realm of Algorithmic design and Associative Modelling with Grasshopper as I found it a powerful platform for design in this way. I did this because it seems that the written, combined resources in this field are limited (although on-line resources are quiet exciting). This is my first draft and I hope to improve it and I also hope that it would be helpful for you.

Mohamad Khabazi

m.khabazi@gmail.com

www.khabazi.com/flux

Contents

Chapter_1_Algorithmic Modelling 1

Chapter_2_The very Beginning 5

2_1_Method 6

2_2_The very basics of Grasshopper 7

2_2_1_Interface, workplace 7

2_2_2_Components 8

2_2_3_Data matching 14

2_2_4_Component's Help (Context pop-up menu) 16

2_2_5_Type-In component searching / adding 17

2_2_6_Geometry Preview Method 17

2_3_Other Resources 18

Chapter_3_Data sets and Math 19

3_1_Numerical Data sets 20

3_2_On Points and Point Grids 22

3_3_Other Numerical Sets 23

3_4_Functions 25

3_5_Boolean Data types 28

3_6_Cull Patterns 30

3_7_2D Geometrical Patterns 35

Chapter_4_Transformation 46

4_1_Vectors and planes 48

4_2_On curves and linear geometries 49

4_3_Combined Experiment: Swiss Re 57

4_4_On Attractors 68

Chapter_ 5_Parametric Space 80
5_1_One Dimensional (1D) Parametric Space 81
5_2_Two Dimensional (2D) Parametric Space 83
5_3_Transition between spaces 84
5_4_Basic Parametric Components 85
5_4_1_Curve Evaluation 85
5_4_2_Surface Evaluation 86
5_5_On Object Proliferation in Parametric Space 88

Chapter_6_ Deformation and Morphing 96
6_1_Deformation and Morphing 97
6_2_On Panelization 99
6_3_Micro Level Manipulations 102
6_4_On Responsive Modulation 106

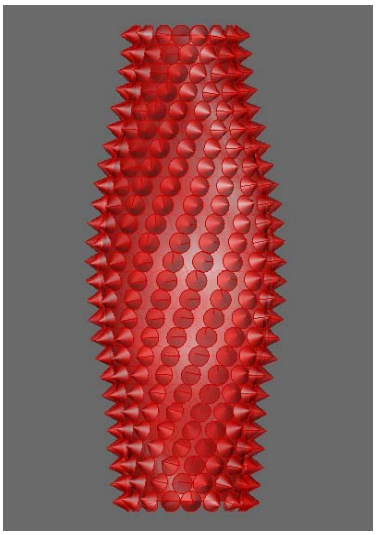
Chapter 7_NURBS Surface and Meshes 112
7_1_Parametric NURBS Surfaces 113
7_2_Mesh vs. NURBS 124
7_2_1_Geometry and Topology 124
7_3_On Particle Systems 126
7_4_On Colour Analysis 135
7_5_Manipulating Mesh objects as a way of Design 139

Chapter_8_Fabrication 141
8_1_Datasheets 143
8_2_Laser Cutting and Cutting based Fabrication 155

Chapter_9_Design Strategy 170

Bibliography 174

Chapter_1_Algorithmic Modelling



Chapter_1_Algorithmic Modelling

If we look at architecture as an object represented in the space, we always deal with geometry and a bit of math to understand and design this object. In the History of architecture, different architectural styles have presented multiple types of geometry and logic of articulation and each period have found a way to deal with its geometrical problems and questions. Since computers started to help architects, simulate the space and geometrical articulations, it became an integral tool in the design process. Computational Geometry became an interesting subject to study and combination of programming algorithms with geometry yielded algorithmic geometries known as Generative Algorithm. Although 3D softwares helped to simulate almost any space visualized, it is the Generative Algorithm notion that brings the current possibilities of design, like 'parametric design' in the realm of architecture.

Architects started to use free form curves and surfaces to design and investigate spaces beyond the limitations of the conventional geometries of the "Euclidian space". It was the combination of Architecture and Digital that brought 'Blobs' on the table and then push it further. Although the progress of the computation is extremely fast, architecture has been tried to keep track with this digital fast pace progress.

Contemporary architecture after the age of "Blob" seems to be even more complex. Architectural design is being affected by the potentials of algorithmic computational geometries with multiple hierarchies and high level of complexity. Designing and modelling free-form surfaces and curves as building elements which are associated with different components and have multiple patterns is not an easy job to do with traditional methods. This is the time of algorithms and scripts which are forward pushing the limits. It is obvious that even to think about a complex geometry, we need appropriate tools, especially softwares, which are capable of simulating these geometries and controlling their properties. As the result architects feel interested to use Swarms or Cellular Automata or Genetic Algorithms to generate algorithmic designs and go beyond the current pallet of available forms and spaces. The horizon is a full catalogue of complexity and multiplicity that combines creativity and ambition together.

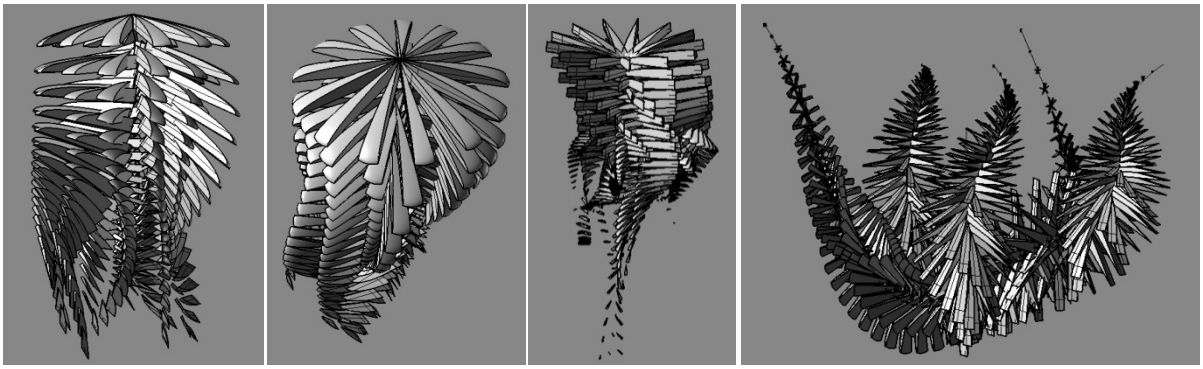


Fig.1.1. Parametric Modelling for Evolutionary Computation and Genetic Algorithm, Mohamad khabazi, Emergence Seminar, AA, conducted by Michael Weinstock, fall 2008.

A step even forward, now embedding the properties of material systems in design algorithms seems to be more possible in this parametric notion. Looking forward material effects and their responses to the hosting environment in the design phase, now the inherent potentials of the components and systems should be applied to the parametric models of the design. So not only these generative algorithms does not dealing only with form generation, but also there is a great potential to embed the logic of material systems in them.

"The underlying logic of the parametric design can be instrumentalised here as an alternative design method, one in which the geometric rigour of parametric modelling can be deployed first to integrate manufacturing constraints, assembly logics and material characteristics in the definition of simple components, and then to proliferate the components into larger systems and assemblies. This approach employs the exploration of parametric variables to understand the behaviour of such a system and then uses this understanding to strategise the system's response to environmental conditions and external forces" (Hensel, Menges, 2008).

To work with the complex objects, usually a design process starts from a very simple first level and then other layers being added to it; complex forms are comprised of different hierarchies, each associated with its logics and details. These levels are also interconnected and their members affect each other and in that sense this method called 'Associative'.

Generally speaking, Associative modelling relates to a method in which elements of design being built gradually in multiple hierarchies and at each level, some parameters of these elements being extracted to be the generator for other elements in the next level and this goes on, step by step to produce the whole geometry. So basically the end point of one curve could be the center point of another circle and any change in the curve would change the circle accordingly. Basically this method of design deals with the huge amount of data and calculations and runs through the flow of algorithms.

Instead of drawing objects, Generative Algorithmic modelling usually starts with numbers, mathematics and calculations as the base data to generate objects. Even starting with objects, it extracts parametric data of that object to move on. Any object of design has infinite positions inside,

and these positions could be used as the base data for the next step and provide more possibilities to grow the design. The process called 'Algorithmic' because of this possibility that each object in the algorithm generated by previously prepared data as input and has output for other steps of the algorithm as well.

The point is that all these geometries are easily adjustable after the process. The designer always has access to the elements of the design product from the start point up to details. Actually, since the design product is the result of an algorithm, the inputs of the algorithm could be changed and the result would also be updated accordingly. In conventional methods we used to modify models and designs on paper and model the final product digitally, to avoid changes which was so time-consuming. Any change in the design affected the other geometries and it was dreadful to fix the problems occurred to the other elements connected with the changed element and all those items should be re-adjusted, re-scaled, and re-orientated if not happened to re-draw.

It is now possible to digitally sketch the model and generate hundreds of variations of the project by adjusting some very basic geometrical parameters. It is now possible to embed the properties of material systems, Fabrication constraints and assembly logics in parameters. It is now even possible to respond to the environment and be associative in larger sense. *"… Parametric design enables the recognition of patterns of geometric behaviour and related performative capacities and tendencies of the system. In continued feedback with the external environment, these behavioural tendencies can then inform the ontogenetic development of one specific system through the parametric differentiation of its sub-locations" (Hensel, Menges, 2008).*

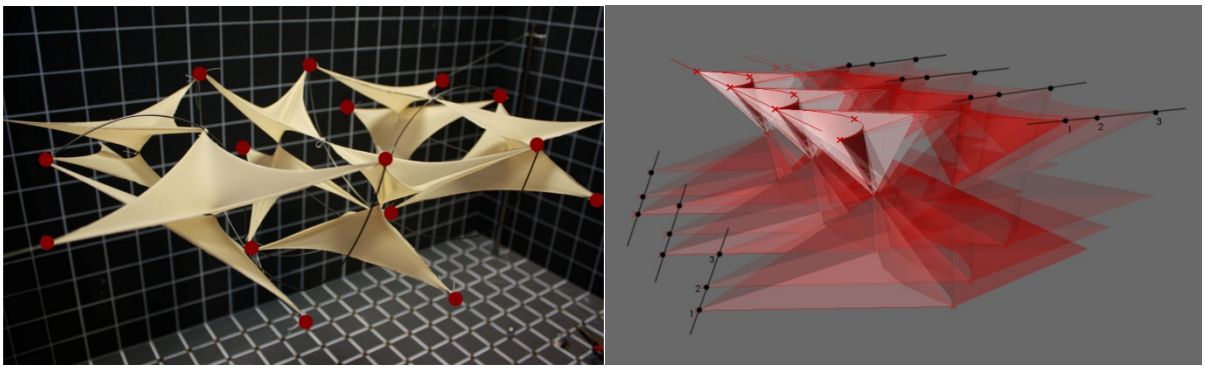
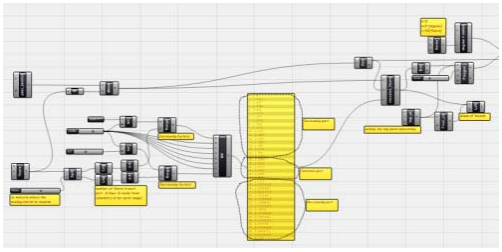


Fig.1.2. A. form-finding in membranes and minimal surfaces, physical model, B. membrane's movement modelled with Grasshopper, Mohamad Khabazi, EmTech Core-Studio, AA, Conducted by Michael Hensel and Achim Menges, fall 2008.

Grasshopper is a platform in Rhino to deal with this Generative Algorithms and Associative modelling. The following chapters are designed in order to combine geometrical subjects with algorithms and to address some design issues in architecture in an 'Algorithmic' method.

Chapter_2_The very Beginning



Chapter_2_The very Beginning

2_1_Method

By the time that you downloaded the Grasshopper, I am sure that you went through the website and up to now you have some idea that what is it all about and how it works generally. You might have seen some of the on line video tutorials as well. If you have a look at the ”Grasshopper Primer” by Andy Payne of Lift Architects (which is addressed in the Grasshopper website), you will find almost all basic understanding of working with components and some related subjects like vectors, different types of curves, surfaces and so on.

I would try not to repeat this great information and I recommend you to go through them, if you have not yet! So in the following chapters I would try to focus on different concepts and examples of Associative Modelling mostly related to architecture. In most cases I assumed that you already know the basic understanding of the ingredients of the discussion and I would not go through the definition of the ‘degree of a curve’ although I will touch some.

To start the Grasshopper and have a general idea about it, the best to do is to go to the following link and check the Grasshopper web page. There is some useful information that gives you the basic understanding to start with. You can keep yourself updated by the discussions in the forum as well. By the way here in this chapter I just briefly discussed about general issues of workplace and basics of what we should know in advance.

http://grasshopper.rhino3d.com/

2_2_The very basics of Grasshopper

2_2_1_Interface, workplace

Beside the other usual Windows menus, there are two important parts in the Grasshopper interface: Component panels and Canvas. Component panels provide all elements we need for our design and canvas is the work place. You can click on any object and click again on canvas to bring it to work place or you can drag it on to the work place. Other parts of the interface are easy to explore and we will be familiar with them throw using them later on. (If you like to know more, just go to http://grasshopper.rhino3d.com/2008/05/interface-explained_26.html for more details)

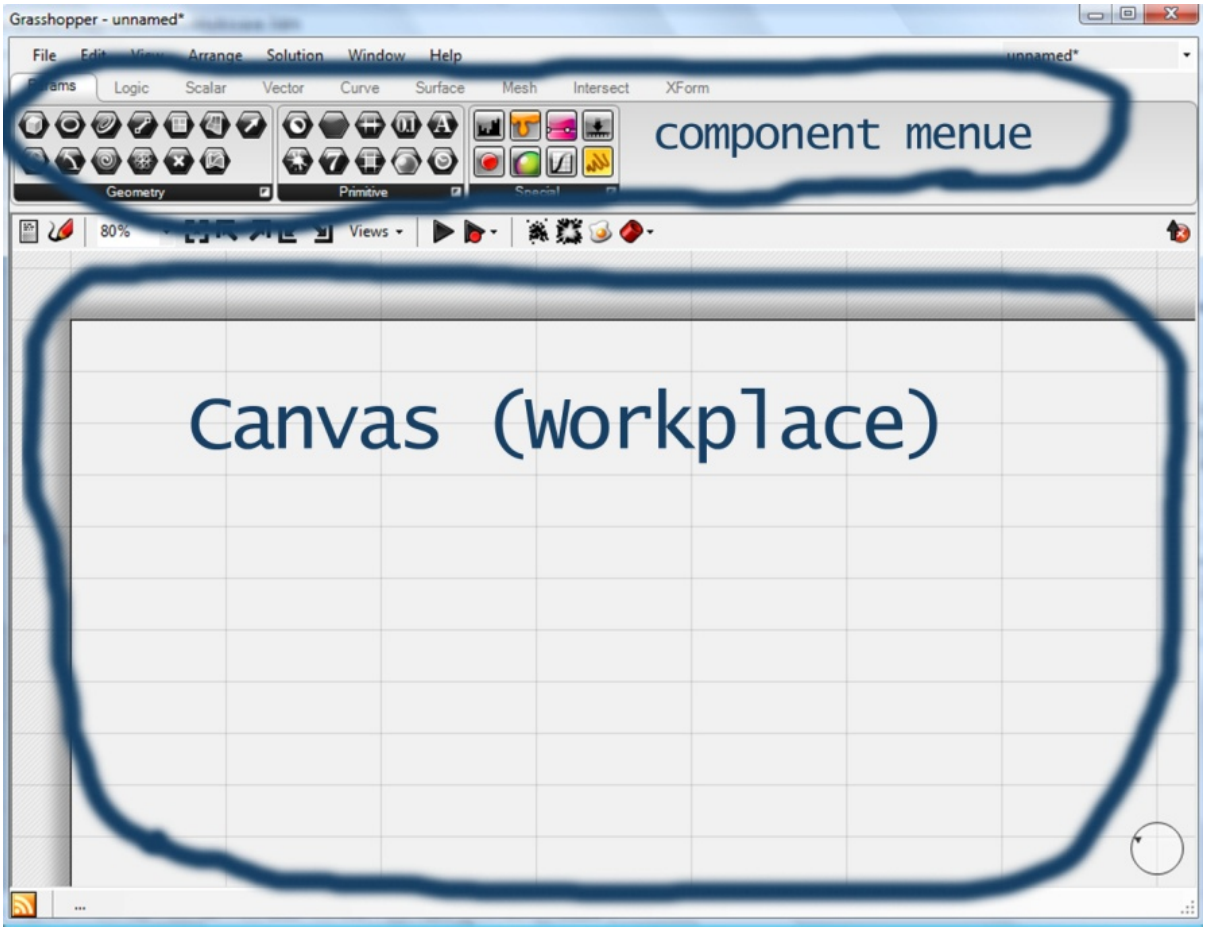
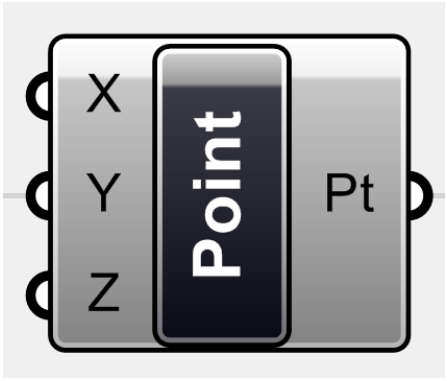


Fig.2.1. Grasshopper Component menu and Canvas.

2_2_2_Components

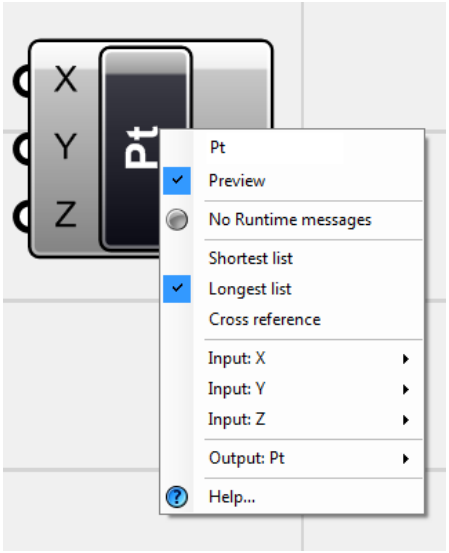
There are different types of objects in Grasshopper component menu which we use to design stuff. You can find them under nine different tabs called: Params, Logic, Scalar, Vector, Curve, Surface, Mesh, Intersect and XForm.

Parameters are objects that represent data, like a point or line. We can define them manually from Rhino objects as well. Components are objects that do actions with them like move, orientate, and decompose. We usually need to provide relevant data for them to work. In this manual I used the term component to talk about any objects from the component panel to make life easier! and I always use <> to address them clearly in the text, like <Point>.



<Point> component

If you right-click on a component a menu will pop-up that contains some basic aspects of the component. This menu called **"context pop-up menu"**.



Context pop-up menu of <Pt> component

Defining external geometries

Most of the time we start our design by introducing some objects from Rhino workplace to the Grasshopper; A point, a curve, a surface up to multiple complex objects to work on them. Since any object in Grasshopper needs a component in canvas to work with, we can define our external geometries in canvas by components in the Params tab under Geometry. There is a list of different types of geometries that you can use to define your 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 to the component. By introducing an object/multiple objects to a component it becomes a Grasshopper object which we can use it for any purpose. It means we can use our manually created objects or even script generated objects from Rhino in Grasshopper.

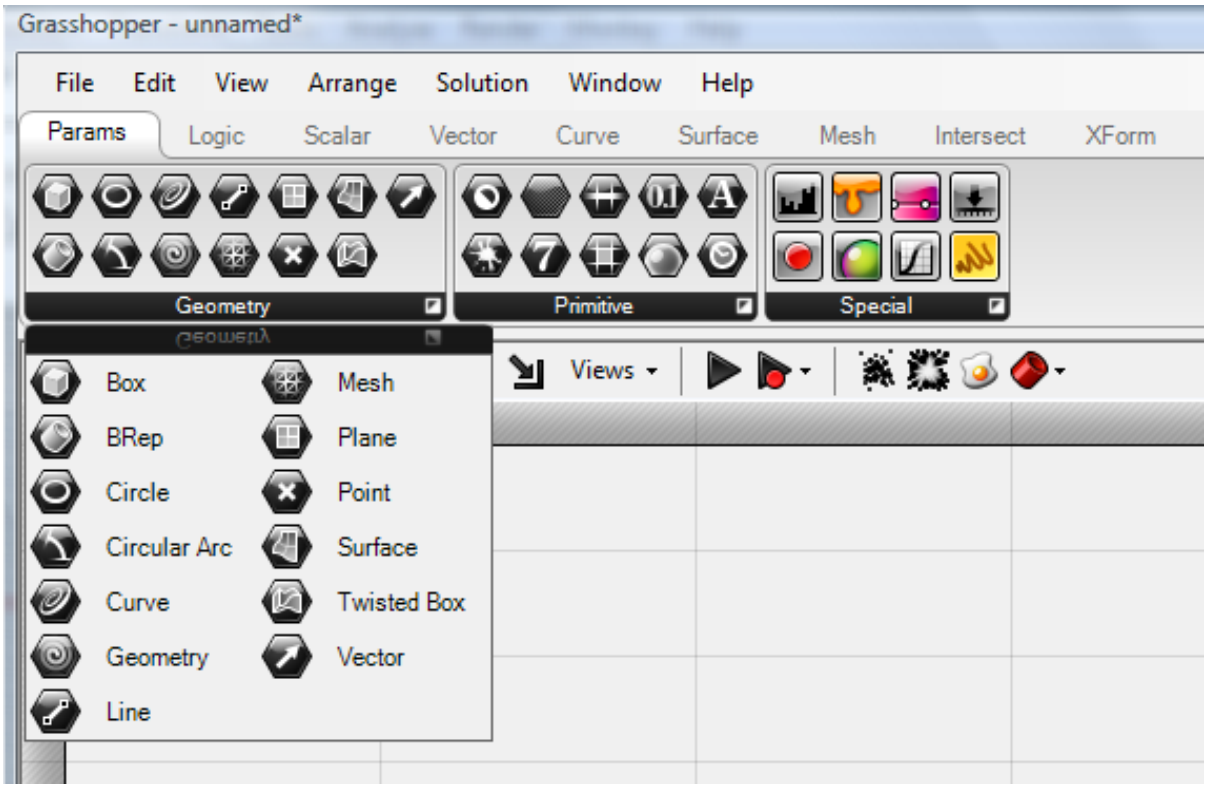


Fig.2.2. Different geometry types in the Params > Geometry menu

Let’s have a simple example.

We have three points in the Rhino viewport and we want to draw a triangle by these points. First we need to 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.2.6).

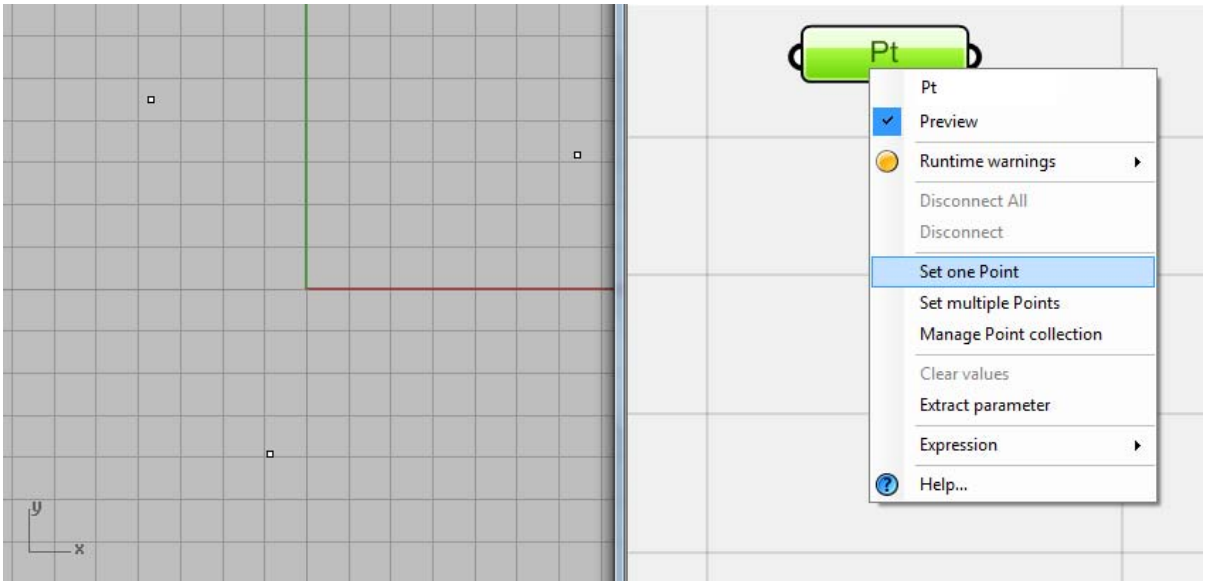


Fig.2.3. Set point from Rhino in Grasshopper component

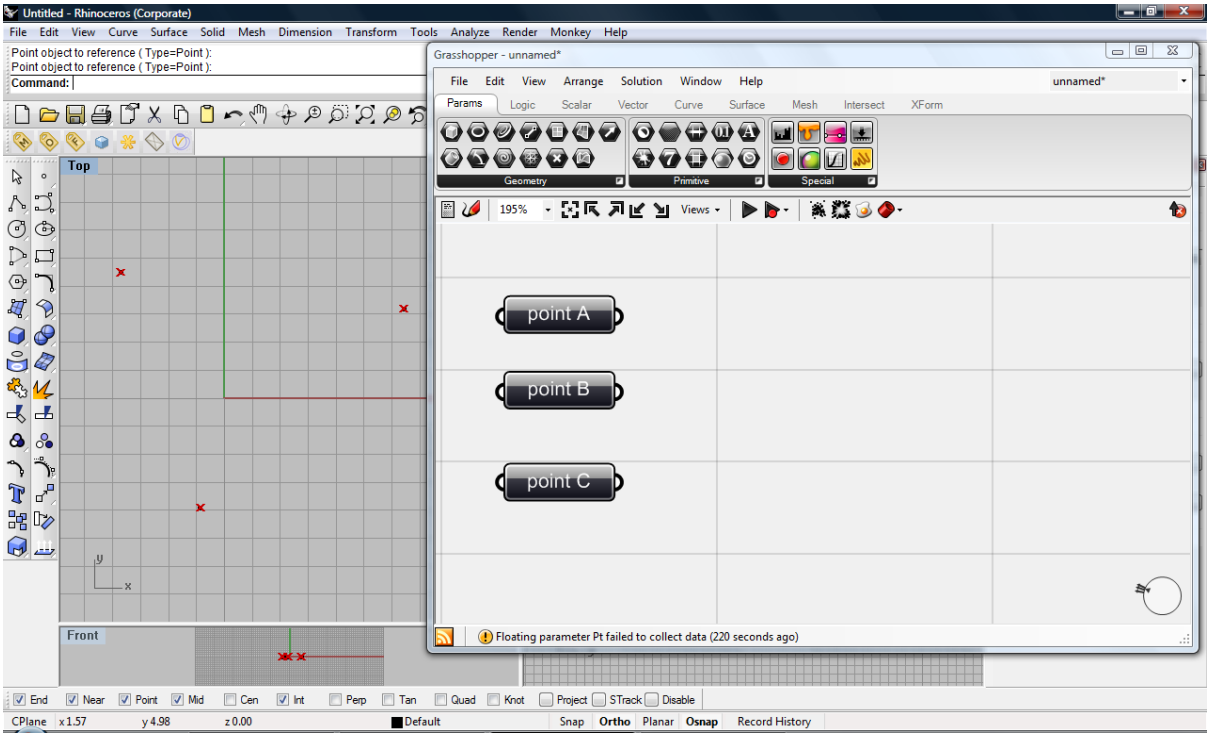


Fig.2.4. The Grasshopper canvas and three points defined in the canvas which turned to (x) in the Rhino workplace. I renamed the components to point A/B/C by the first option of their menu to recognize them easier in Grasshopper canvas.

Components and connections

There are so many different actions that we can perform by components. Generally a component takes some data from another source (like parameters) and gives the result back. We need to connect the component which includes the input data to the processing component and connect the result to the other component that needs this result and so on.

Going back to the example, now if you go to the Curve tab of components, 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 (just click on the semi-circle and drag it up to the other semi-circle on the target. You can see that Rhino draws a line between these points).

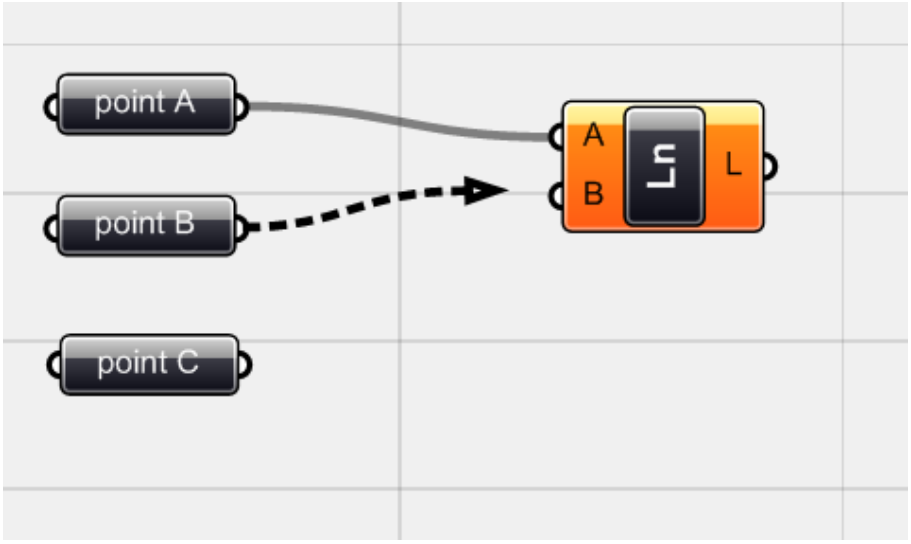


Fig.2.5. Connecting the <point> components to the <line> component by dragging from output of the <point B> to the input of the <line>.

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.

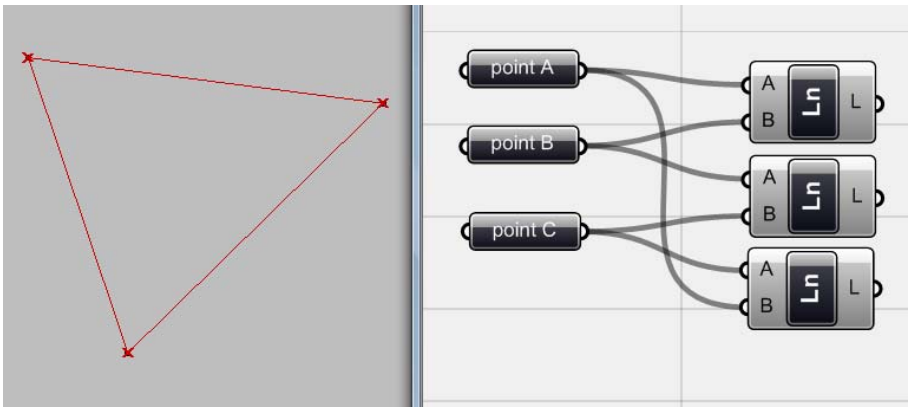


Fig.2.6. The <line> components draw lines between <point> components. As you see any component could be used more than once as the source of information for other actions.

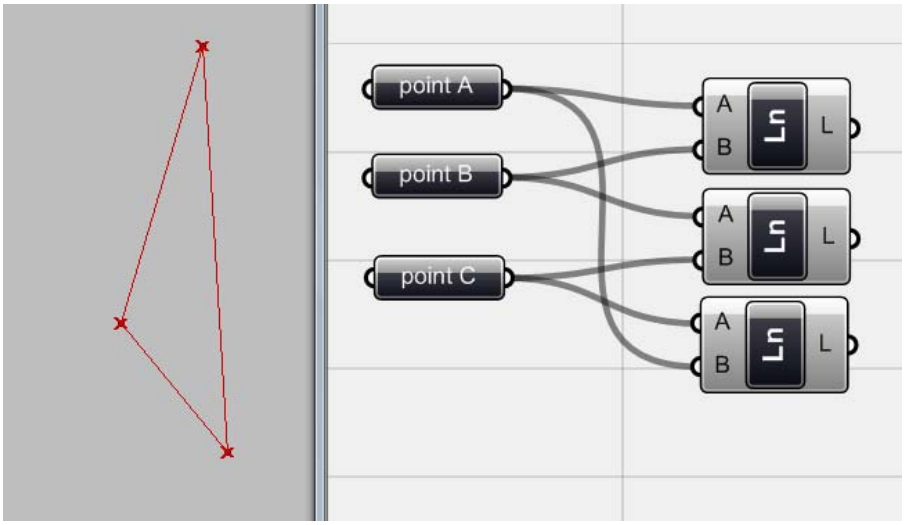


Fig.2.7. Now if you change the position of the points manually in Rhino viewport, the position of points in Grasshopper (X ones) and the triangle will change accordingly and you do not need to redraw your lines any more.

As you can see in this very first example, the associative modelling technique made it possible to manipulate the points and still have the triangle between these points without further need to adjustment. We will do more by this concept.

Input / Output

As mentioned before, any component in the grasshopper has input and output which means it processes the given data and gives the processed data back. Inputs are at left part of the component and outputs at right. The data comes from any source attached to the input section of the component and the output of the component is the result of that specific function.

You have to know that what sort of input you need for any specific function and what you get after that. We will talk more about the different sort of data we need to provide for each component later on. Here I propose you to hold your mouse or "hover" your mouse over any input/output of the components. A tooltip will pop up and you will see the name, sort of data you need to provide for the component, is any predefined data there or not, and even what it for is.

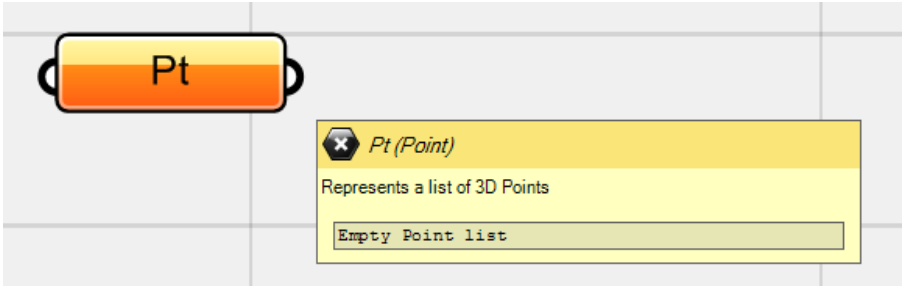


Fig.2.8. Pop-up tooltip

Multiple connections

Sometimes you need to feed a component 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 C. you can use two different <line> components or you can use one <line> component and attach both point B and C as the second point of the <line> component. 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 (Fig.2.12). When holding shift, the arrow of the line appear in a green circle with a tiny (+) icon while normally it is gray. You can also use Ctrl key to disconnect a component from another (or use menu) to disconnect an existing unwanted connection. In this case the circle around the arrow appears in red with a tiny (-) icon.

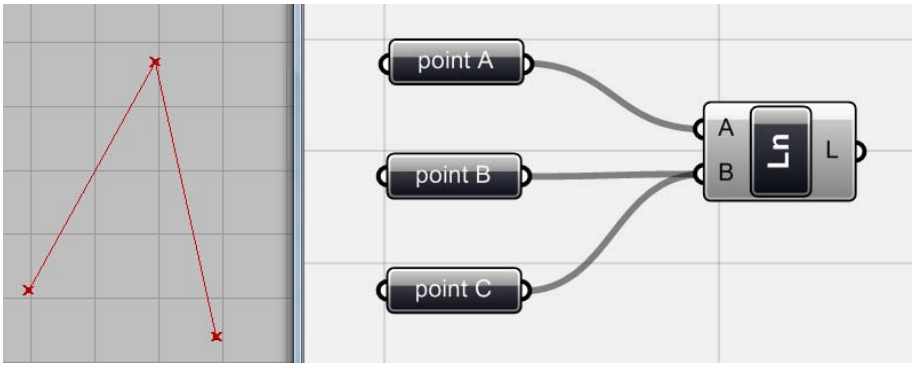


Fig.2.9. Multiple connections for one component by holding shift key

Colour coding

There is a colour coding system inside the Grasshopper which shows the components working status.

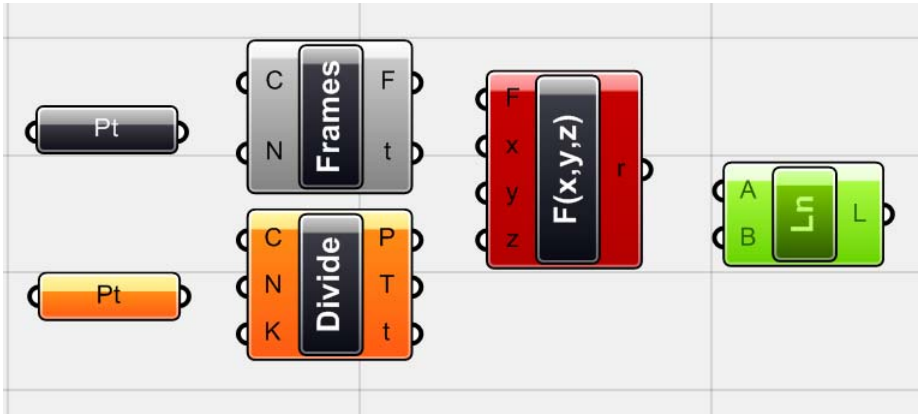


Fig.2.10. The colour coding.

Any gray component means there is no problem and the data defined correctly/the component works correctly. The orange shows warning and it means there is at least one 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 the error should be found and solved in order to make

component works properly. You can find the first help about the source of error in the component's context menu (context menu > Runtime warning/error) and then search the input data to find the reason of the error. The green colour means this component selected. The geometry which is associated with this component also turns into green in Rhino viewport (otherwise all Grasshopper geometries are red).

Preview

The components that produce objects in Rhino have the 'Preview' option in their menu. We can use it to hide or unhide it in the scene. Any unchecked preview make the component black part become hatched. We usually use preview option to hide the undesired geometries like base points and lines in complex models to avoid distraction.

2_2_3_Data matching

For many Grasshopper components it is always possible to provide a list of data instead of just one input. So in essence you can provide a list of points and feed a <line> component by this list and draw more lines instead of one. It is possible to draw hundreds of objects just by one component if we provide information needed.

Look at this example:

I have two different point sets each with seven points. I used two <point> components and I used 'set multiple points' to introduce all upper points 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> component, seven lines being generated between them. So we can generate more than one object with each component (Fig.2.14)

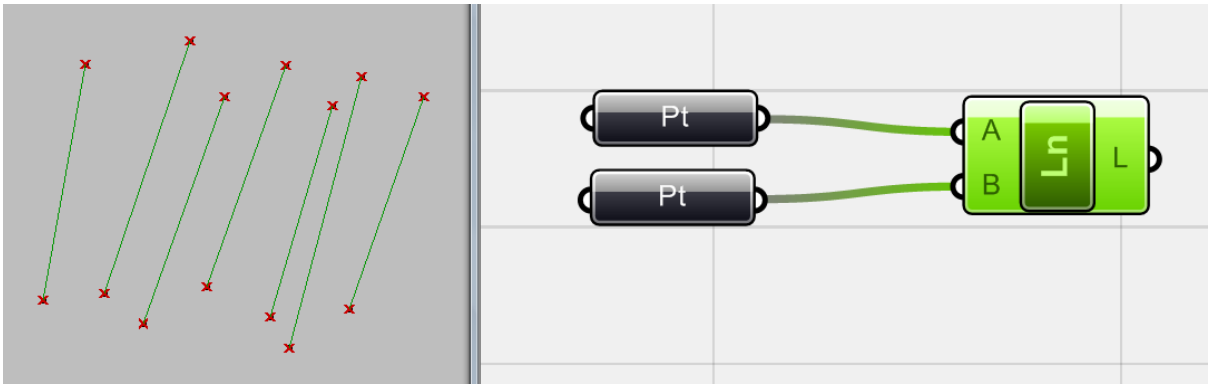


Fig.2.11. Multiple point sets and generating lines by them.

But what would happen if the number of points would not be the same in two point (data) sets?

In the example below I have 7 points in top row and 10 points in the bottom. Here we need a concept in data management in Grasshopper called 'Data matching'. 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

Look at the difference in the Figure 0.15

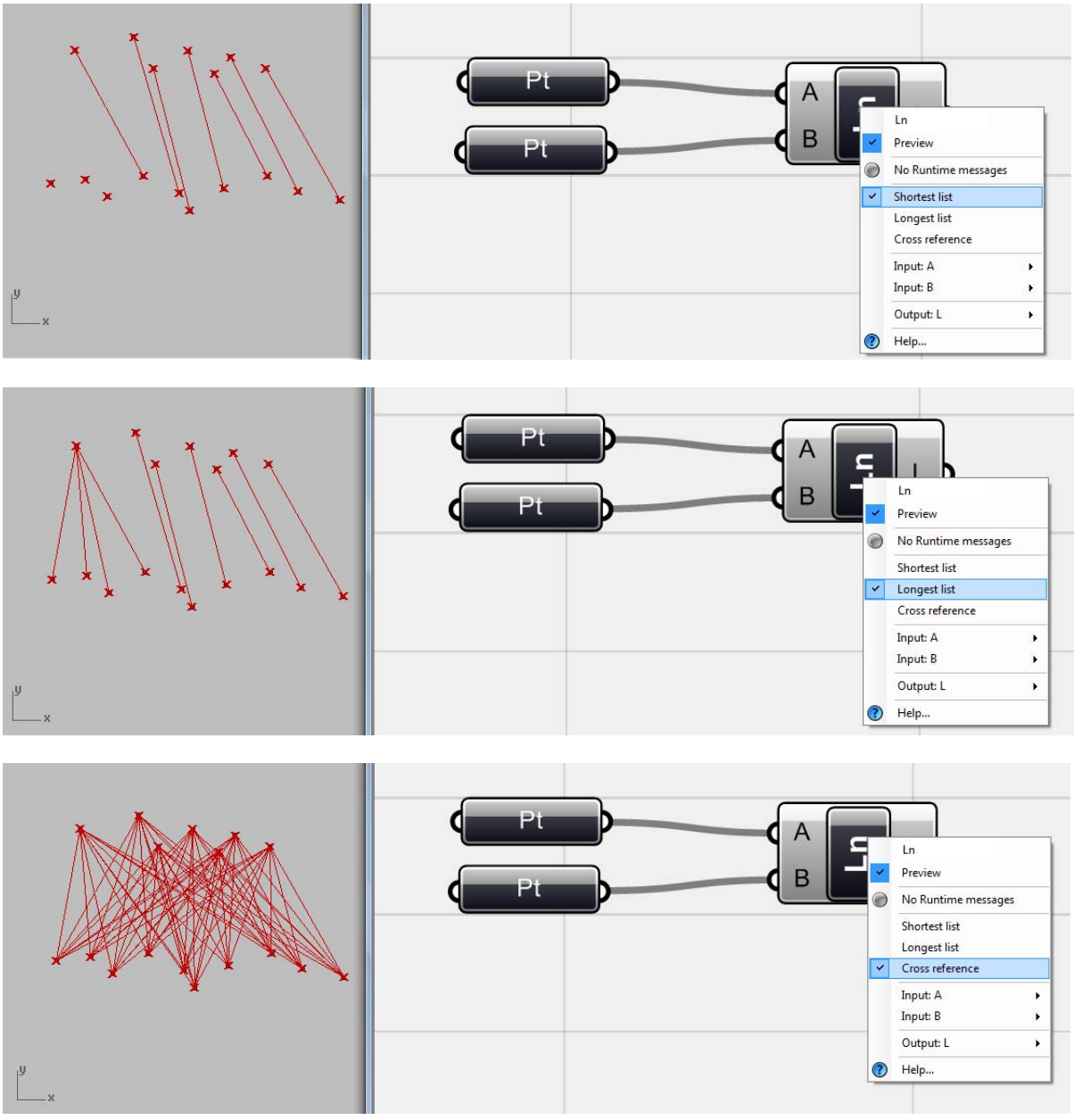


Fig.2.12. Data matching A: shortest list, B: longest list and C: cross reference

It is clear that the shortest list uses the shortest data set to make the lines with, and the longest list uses the longest data set while uses an item of the shortest list more than once. The cross reference option connects any possible two points from the lists together. It is very memory consuming option and sometimes it takes a while for the scene to upgrade the changes.

Since the figures are clear, I am not going to describe more. For more information go to the following link: http://grasshopper.rhino3d.com/2008/06/description-of-data-stream-matching.html

2_2_4_Component's Help (Context pop-up menu)

As it is not useful to introduce all components and you will better find them and learn how to use them gradually in experiments, I recommend you to play around, pick some components, go to the components context menu (right-click) and read their Help which is always useful to see how this component works and what sort of data it needs and what sort of output it provides. There are other useful features in this context menu that we will discuss about them later.

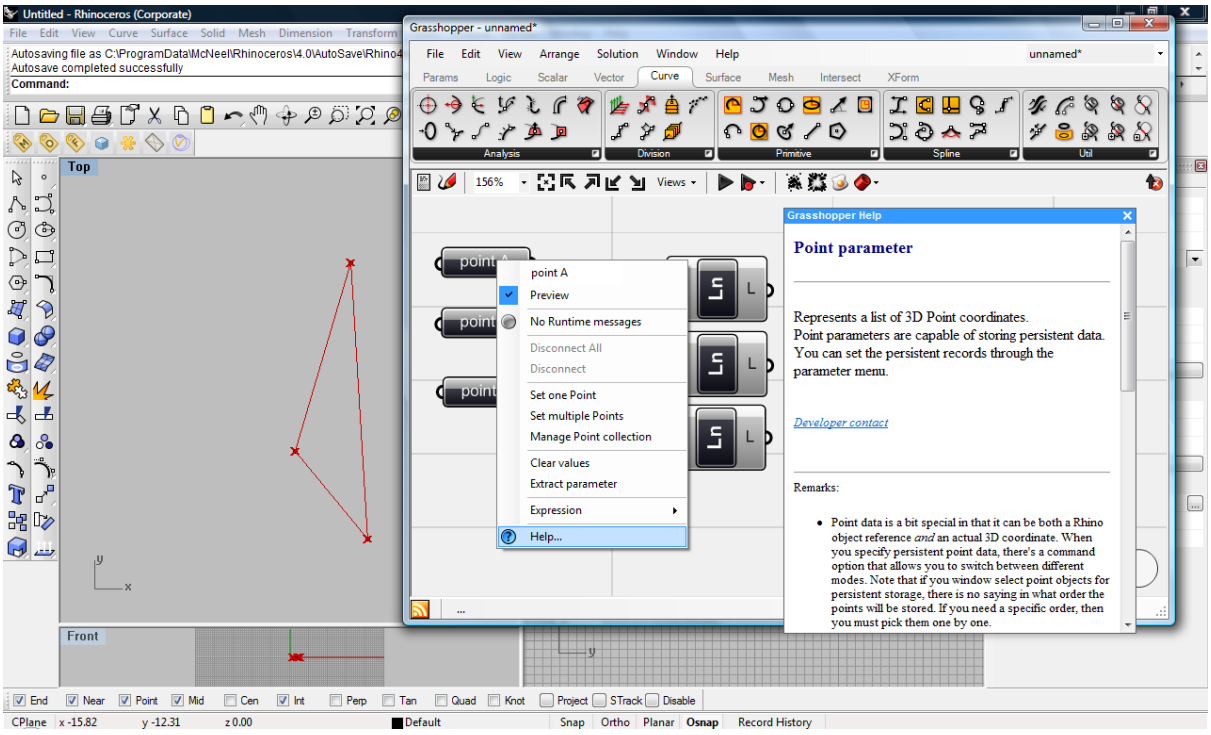


Fig.2.13. Context pop-up menu and Help part of the component

2_2_5_Type-In component searching / adding

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

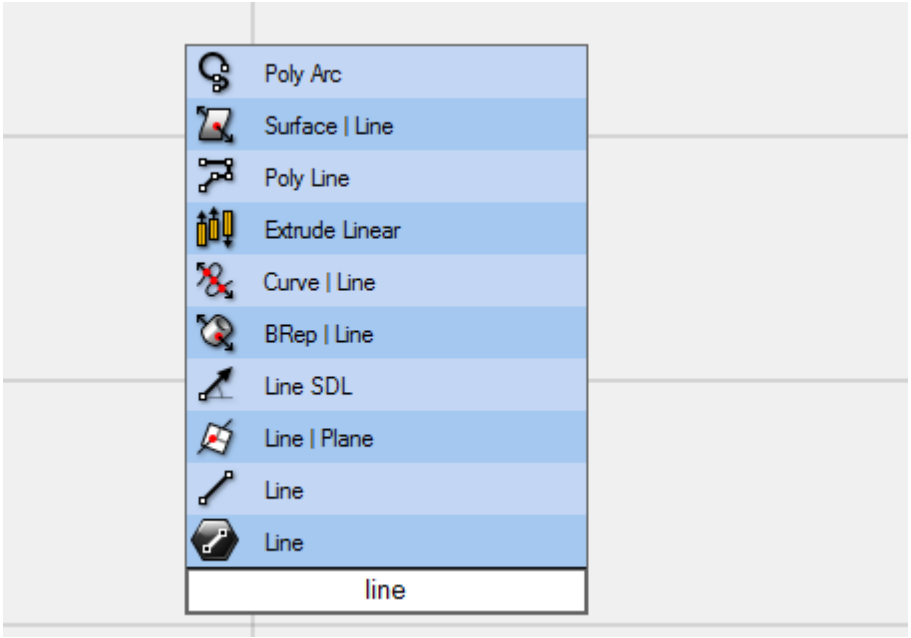


Fig.2.14. Searching for line component in the component-pop-up menu by double clicking on the canvas and typing the name of it. The component will be brought to the canvas.

2_2_6_Geometry Preview Method

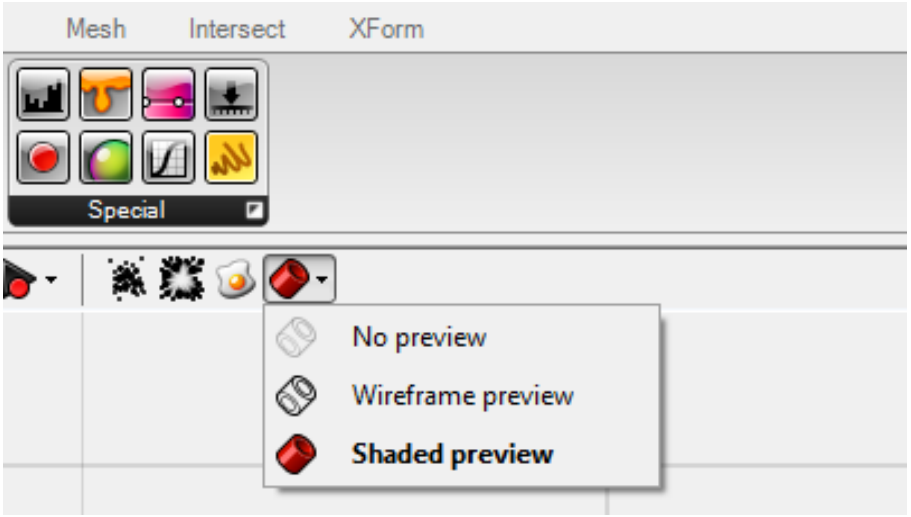


Fig.2.15. In order to enhance the working speed and get faster updates, whenever your project becomes heavy to calculate, use the Wireframe Preview option. It is always faster.

2_3_Other Resources

There are so many great on-line resources and creative ideas that you can check and learn from them. Here are some of them:

Main Grasshopper web page:

http://grasshopper.rhino3d.com/2008/06/some-examples-of-grasshopper.html

Some resources on McNeel Wiki WebPages:

http://en.wiki.mcneel.com/default.aspx/McNeel/ArchitectureCommunity.html

http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryExamples.html

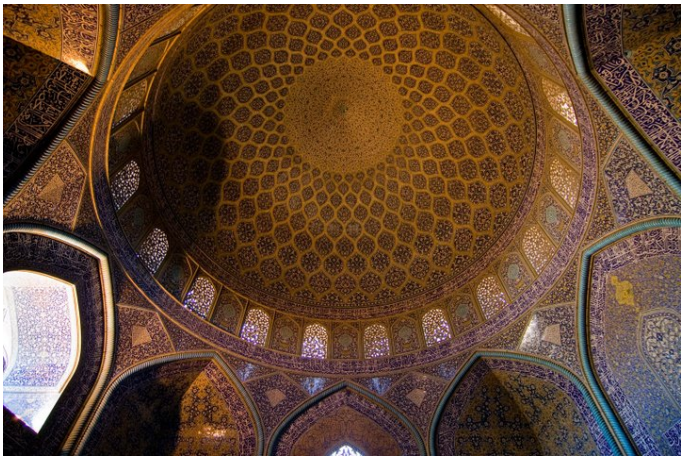
(Links to other resources)

As mentioned before, the Grasshopper Primer from Lift Architects:

http://www.liftarchitects.com/journal/2009/1/22/the-grasshopper-primer.html

And hundreds of on-line video tutorials which you can search easily.

Chapter_3_Data sets and Math



Chapter_3_Data sets and Math

Although in 3D softwares we used to select our geometry from menus and draw them explicitly by clicking without thinking of the mathematical aspects of what we design, in order to work with Generative Algorithms, as the name sounds, we need to think a bit about data and math to make inputs of algorithm and generate multiple objects. Since we do not want to draw everything manually, we need some sources of data as the basic ingredients to make this generation possible.

The way algorithm works is simple and straightforward. As I said, instead of copying by clicking 100 times in the screen, we can tell the algorithm, copy an item for 100 times in X positive direction. To do that you need to define the 100 as number of copying and X Positive direction for the algorithm, and it performs the job automatically. All we are doing in geometry has some peace of math behind. We can use these simple math functions in our algorithms, in combination of numbers and objects, generate infinite geometrical combinations.

Let's have a look; it is easier than what it sounds!

3_1_Numerical Data sets

First of all we should have a quick look at numerical components to see how we can generate different numerical data sets and then the way we can use them.

One numerical value

The most useful number generator is <Number 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 them all by 'Edit' part of the context menu.

For setting one fixed numeric value you can go to the Params > Primitive > Integer / Number to set one value.

Series of numbers

We can produce a list of discrete numbers by <series> component (Logic > Sets > Series). This component produces a list of numbers which we can adjust the start point, step size of the numbers, and the number of values.

0, 1, 2, 3, … , 100

0, 2, 4, 6, … , 100

10, 20, 30, 40, … , 1000000

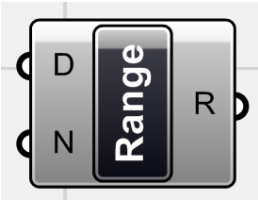
Rang of numbers

We can divide a numerical range between a low and high value by evenly spaced numbers and produce a range of numbers. We need to define an interval to set the lower and upper limit and also the number of steps between them (Logic > Sets > Range).

1, 2, 3, … , 10

1, 2.5, 5, … , 10

1, 5, 10



Intervals

Intervals provide a range of all real numbers between a lower and upper limit. There are one dimensional and two dimensional intervals that we talk about them later. We can define a fixed interval by using Params > Primitive > Interval/interval^2 component or we can go to the Scalar > Interval which provides a set of components to work with them in more flexible ways.

Intervals by themselves do not provide numbers, they are just extremes, upper and lower limits. As you now there are infinite real numbers between any two numbers. We use different functions to divide them and use division factors as the numerical values.

3_2_On Points and Point Grids

Points are among the basic elements for geometries and Generative Algorithms. As points mark a specific position in the space they can be a start point of a curve or multiple curves, centre of a circle, origin of a plane and so many other roles. In Grasshopper we can make points in several ways.

- We can simply pick a point/bunch of points from the scene and introduce them to our workplace by <point> component (Params > Geometry > point) and use them for any purposes (These points could be adjusted and moved manually later on in Rhino scene and affect the whole project. Examples on chapter_2).

- We can introduce points by <point xyz> component (vector > point > point xyz) and feed the coordinates of the points by different datasets, based on our needs.

- We can make point grids by <grid hexagonal> and <grid rectangular> components.

- We can extract points from other geometries in many different ways like endpoints, midpoints, etc.

- Sometimes we can use planes (origins) and vectors (tips) as points to start other geometries and vice versa.

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 sets by <series>, <range> and <number slider> components and other numerical data providers.

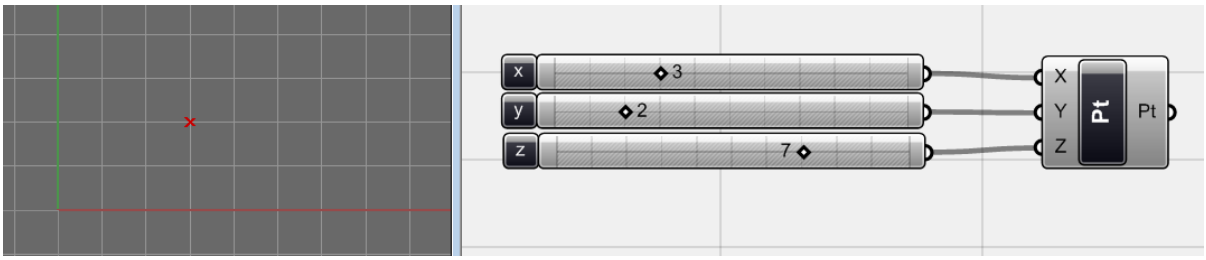


Fig.3.1. feeding a <point xyz> component by three <number slider> to make a point by manually feeding the X,Y and Z coordinates.

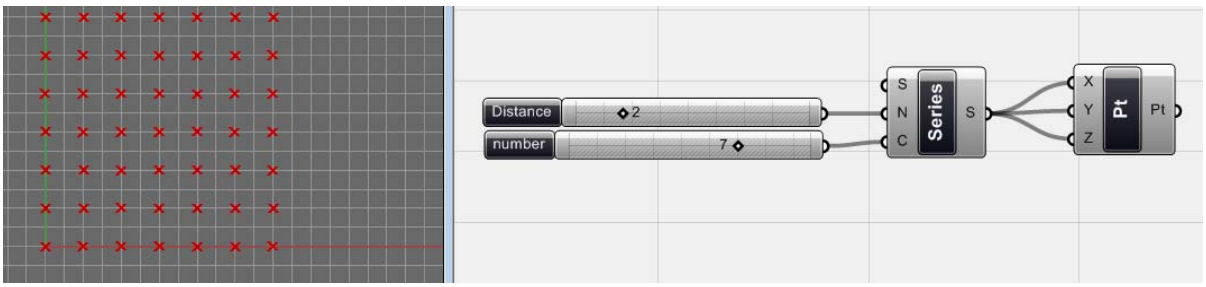


Fig.3.2. Making a grid of points by <series> and <point xyz> components while the first <number sliders> controls the distance between points and the second one controls the number of points by controlling the number of values in <series> component (The data match of the <pt> set into cross reference to make a grid of points but you can try all data matching options).

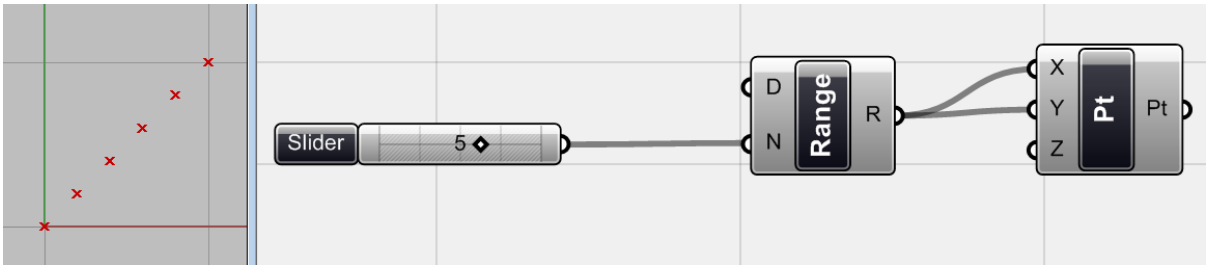


Fig.3.3. Dividing a numerical range from 0 to 1 by 5 and feeding a <pt> component with 'Longest list' data match. You can see we have 6 points which divided the range by 5 and all points drawn between the origin point and (1, 1) on the Rhino workplace (you can change the lower and upper limit of the <range> to change the coordinates of the point).

Since the first experiments look easy, let's go further, but you can have your own investigations around these components and provide different point grids with different positions and distances.

3_3_Other Numerical Sets

Random data sets

I was thinking of making a randomly distributed set of points for further productions. All I need is a set of random numbers instead of a <series> to feed my <pt> component (I use <pt> instead of <point xyz> because it is shown on the component). So I pick a <random> component from Logic > sets. To avoid the same values for X,Y and Z, I need different random numbers for each.

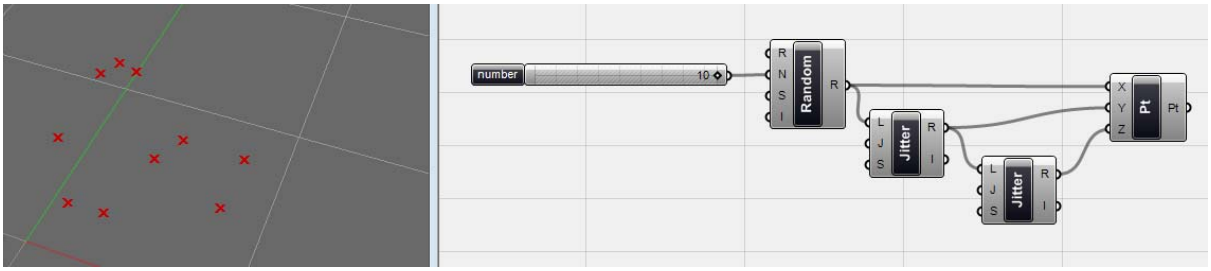


Fig.3.4. Making a random point set.

The <random> component produces 10 random numbers which is controlled by <number slider> and then this list is shuffled by <jitter> component (Logic > Sets > Jitter) for Y coordinate of the points once, and again for Z coordinates, otherwise you could see some sort of pattern inside your grid (check it!). The data match set to longest list again to avoid these sorts of patterns in the grid.

In the figure 3.4 all points are distributed in the space between 0 and 1 for each direction. To change the distribution area of the points we should change the numerical domain in which random component produces the numbers. This is possible by manually setting the "domain of random numeric range" on Rhino command line or by defining the domain intervals adjustable by sliders. (Fig.3.5)

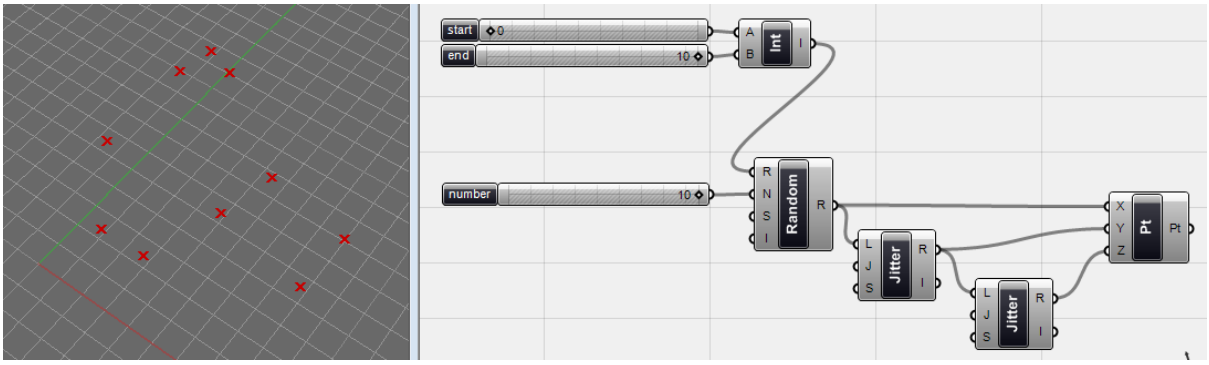


Fig.3.5. Setting up a domain by an <interval> component (Scalar > Interval > Interval) to increase the distribution area of the points (look at the density of the scene's grid in comparison with the Fig.3.4). If you connect only one <number slider> to the domain of the <random> component it just adjusts the upper interval of the domain (with lower as 0).

Fibonacci series

What about making a point grid with non-evenly spaced increasing values? Let's have a look at available components. We need series of numbers which grow rapidly and under Logic tab and Sets section we can see a <Fibonacci> component.

A Fibonacci is a series of numbers with two first defined numbers (like 0 and 1) and the next number is the sum of two previous numbers.

N(0)=0, N(1)=1, N(2)=1, N(3)=2, N(4)=3, N(5)=5, … , N(i)=N(i-2)+N(i-1)

Here are some of the numbers of the series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

As you see the numbers grow rapidly.

Here I use <Fibonacci> series (Logic > Sets > Fibonacci) to produce incremental numbers and feed the <pt> component with them.

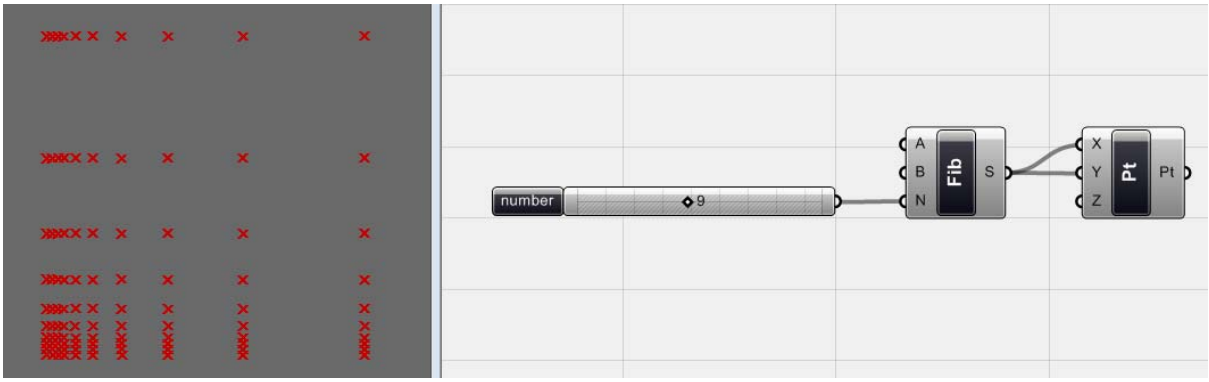


Fig.3.6. Using <Fibonacci> series to produce increasing distances (none-evenly spaced series of numbers) to make points. The number of points could be controlled with a <number slider>.

3_4_Functions

Functions are components that are capable of performing math functions. There are functions from one to eight variables (Scalar > Expressions). You need to feed a function with different data (not always numeric but also Boolean, coordinate, etc) and it performs the user defined function on the input data. To define the function you can right-click on the (F) part of the component and type it or go to the **Expression Editor**. Expression editor has so many predefined functions and a library of math functions for help.

Math functions

Using the predefined components is not always what we aimed for, but in order to get the desired result we can use mathematical functions to change the data sets and feed them for making geometries.

A simple example is the mathematical function of a circle that is X=Sin(t) and Y=Cos(t) while (t) is a range of numbers from 0 to 2 Pi. I am producing it by a <range> of numbers which is starts from 0 to 1 by N number in between, times 2Pi by <function> that means a range of numbers from 0 to 2pi that makes a complete circle in radian.

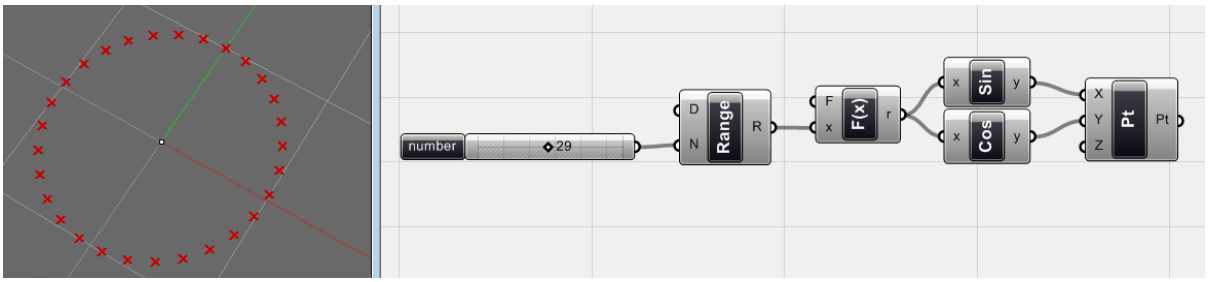


*Fig.3.7. Parametric circle by mathematical functions. You have <Sin> and <Cos> functions in the Scalar > Trig. (F(x)=x * 2Pi).*

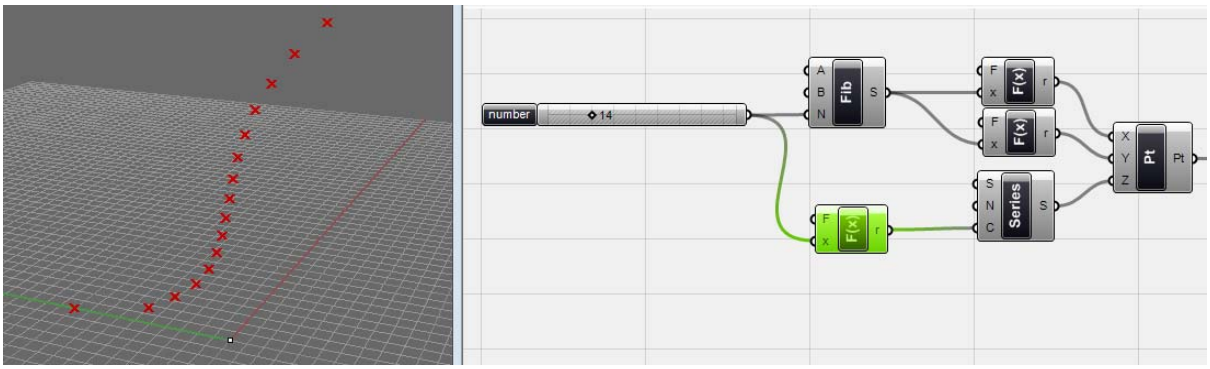


Fig.3.8. Series of points which is defined by <Fibonacci> series and simple mathematical functions (x->F(x)=x/100, y->F(x)=x/10). The selected green F(x) is a simple function to add 1 to the <number slider> (x+1) in order to make the values of <series> numbers equal to the Fibonacci numbers. The aim is to show you that we can simply manipulate these data sets and generate different geometries accordingly.

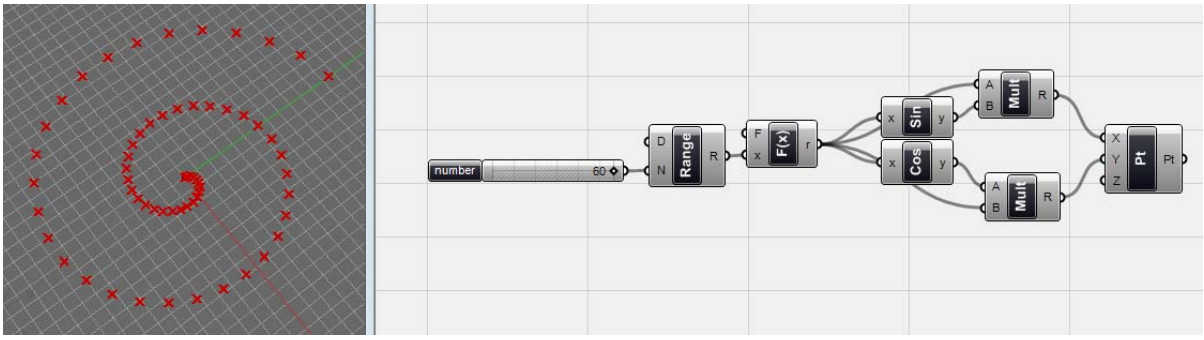


*Fig.3.9. A <range> of numbers from 0 to 2 times by 2Pi with <Function> that make it a numerical range from 0 to 4Pi that feeds the <pt> component by the following math function (X=t * Sin(t), Y=t * Cos(t)).*

You can reduce all components between <range> and <pt> by two functions to feed the <pt> by defining the whole process in Expression Editor.

X of pt > F(x)=X * Sin (x*2*Pi)

Y of pt > F(x)=X * Cos(x*2*Pi)

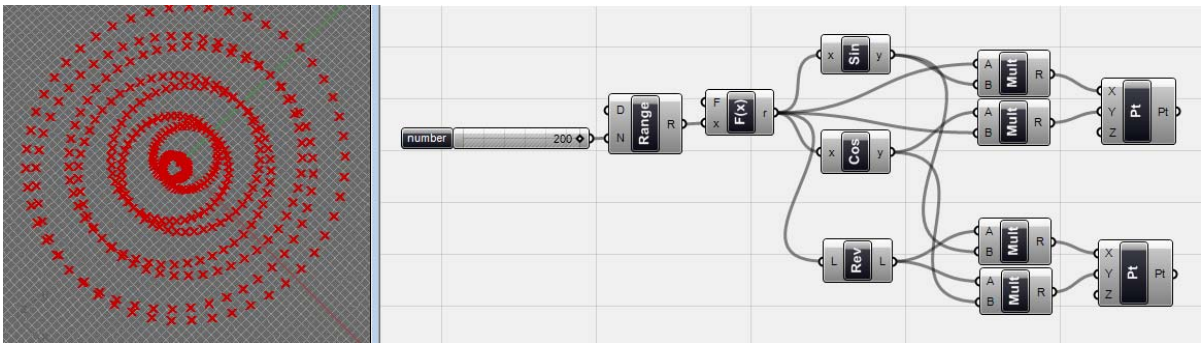


Fig.3.10. Inter tangent spirals from two inverted spiral point sets (<range> interval from 0 to 4 multiplied by 2Pi, makes the data set from 0 to 8Pi which is inverted for the second spiral by <Reverse list> component from Logic > Lists as 8pi to 0).

*First <pt>: X=t * Sin(t), Y=t * Cos(t) in which t=0 to 8Pi*

*Second <pt>: X=t' * sin(t), Y=t' * Cos(t) in which t'=8Pi to 0 (<reverse list>)*

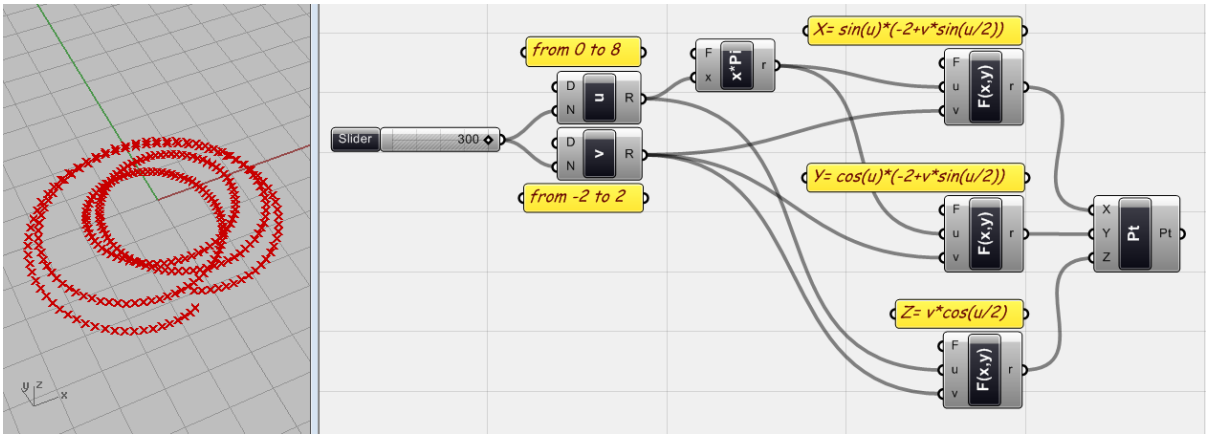


Fig.3.11. Moebius by points

X= Sin(u)*(-2+v*sin(u/2))

Y= Cos(u)*(-2+v*sin(u/2))

Z= v*Cos(u/2)

While u=0 to 8Pi and v=-2 to 2

Playing around the math functions could be endless. You can find so many mathematical resources to match your data sets with them. The important point is that you can manipulate the original data sets and generate different numerical values and feed other components by them.

So as you see by some simple set of numerical data we can start to generate different geometries. Since we need to work with these data sets as a source of our geometries lets go further with them.

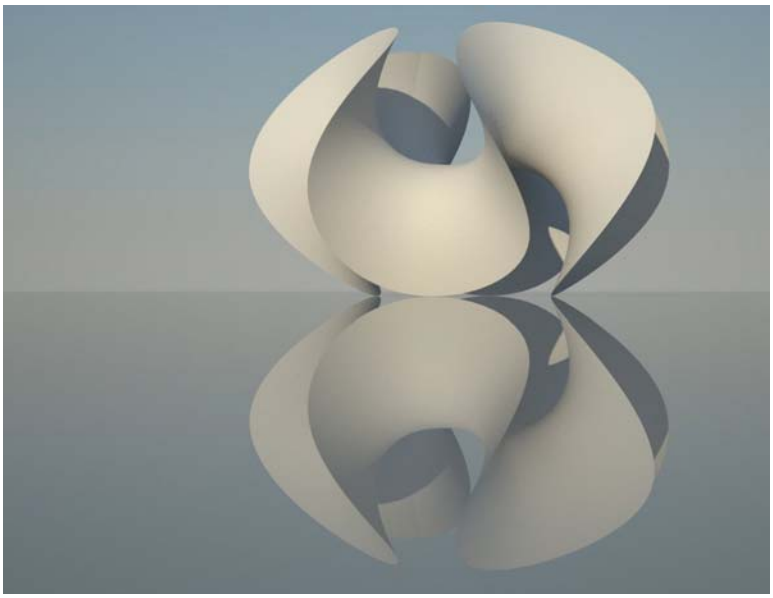


Fig.3.12.Enneper surface, by Rhino's Math function plug-in. Designing surfaces with mathematical equations.

3_5_Boolean Data types

Data is not limited to Numbers. There are other data types that are useful for other purposes in programming and algorithms. Since we are dealing with algorithms, we should know that the progress of an algorithm is not always linear. Sometimes we want to decide whether to do something or not. Programmers call it conditional statements. And we want to see whether a statement meets certain criteria or not to decide what to do next. The response of the conditional 'question' is a simple yes or no. in algorithms we use Boolean data to represent these responses. Boolean values are data types which represent **True (yes)** or **False (no)** values only. If the statement meets the criteria, the response is True, otherwise False. As you will see later, this data type is very useful in different cases when you want to decide about something, select some objects by certain criteria, sort objects, etc.

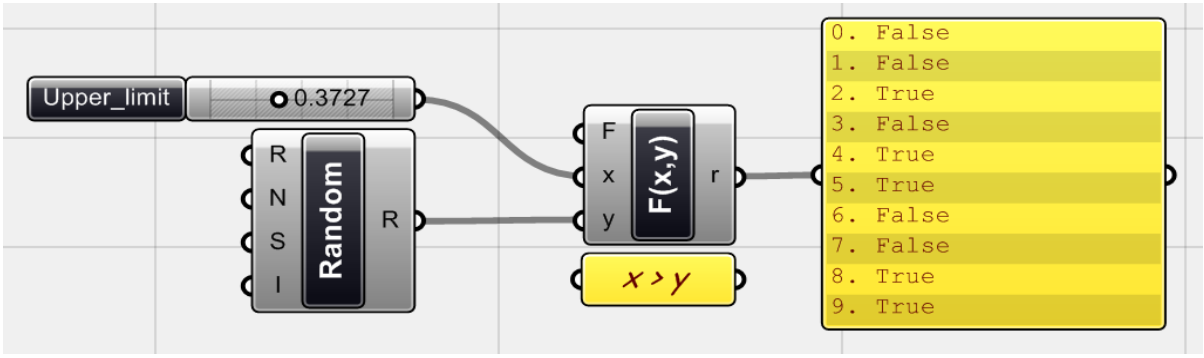


Fig.3.13. Here I generated ten <random> values and by a <function> component I want to see if these numbers are less than a certain <Upper_limit> or not. As you see the <function> is simply X>Y and whenever the numbers meet the criteria, the function passes True to the <panel>.

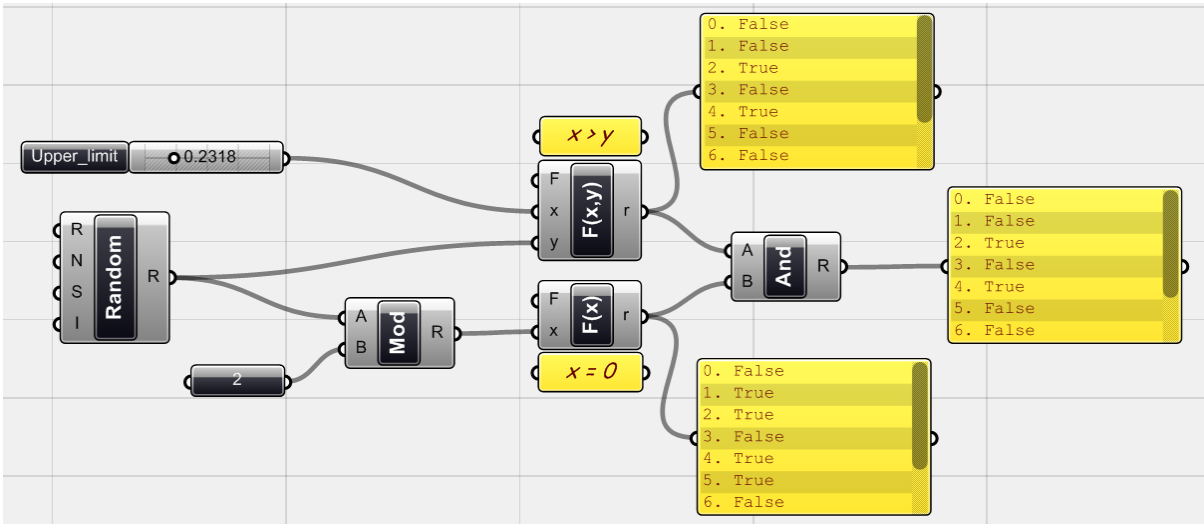


Fig.3.14.a. For the next step, I used a <Modulus> component (Scalar > Operators > Modulus) to find the remainder of the division of the Random values by <2> and I pass the result to a <function> to see if this remainder =0 or not (f(x)=x=0), simply means whether the number is even or not. As you see the result is another <panel> of True/False values.

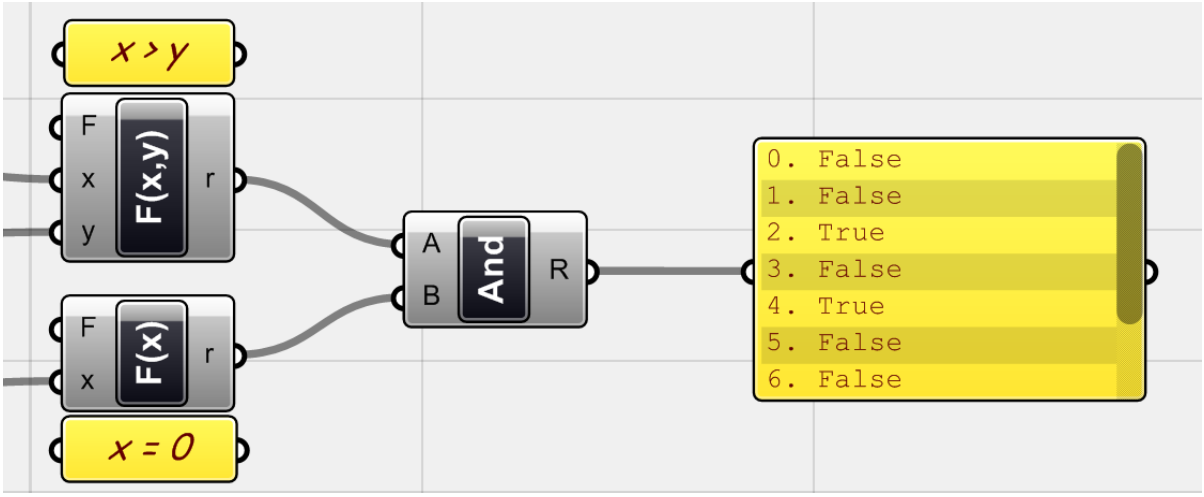


Fig.3.14.b.Here I used a <Gate And> component (Logic > Boolean > Gate And) and I attached both <function>s to perform Boolean conjunction on them. The result is True when both input Boolean values are True, otherwise it would be False. As you see, those numerical values which are both even and bigger than the <Upper_limit> are meeting the criteria and pass True at the end. We will discuss how to use these Boolean values later.

There are multiple Boolean operators on Boolean section of the Logic tab that you can use to create your criteria and combine many of them.

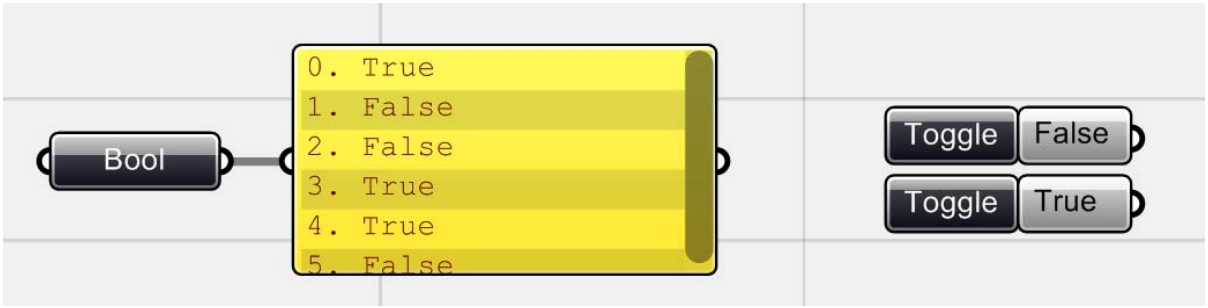


Fig.3.15. we can manually define a set of Boolean data by <Boolean> component from Params tab under Primitive section or use <Boolean Toggle> in Special section to use one manually changeable Boolean value.

3_6_Cull Patterns

There are many reasons that we might want to select some of the items from a given data set and do not apply a function to all elements. 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.

Up to now there are two <cull> components to cull a list of data in Grasshopper. While <cull Nth> omit every N item of the 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 will pass and those that associate with False will 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 logic

I am thinking of selecting some points from a point set based on their distance to another point (reference point). Both point set and the reference point are defined each by a <point> component. First of all what we need is a <distance> component (Vector > Point > Distance) that measures the distance between points and the reference. I compared these distances by a user defined number (<number slider>) with a <F2> component. This comparison generates Boolean values as output (True/False) to show whether the value is smaller (True) or bigger (False) than the upper limit. I am going to use these Boolean values to feed the <Cull pattern> component (the function of <F2> component defined as f=x>y).

As mentioned before, <Cull pattern> component takes a list of generic data and a list of Boolean data and omits the members of the generic list of data who associate with the false value of the Boolean list. So 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, because the X>Y function always pass True for the smaller values. To show them better I just connected them to the reference point by a simple line.

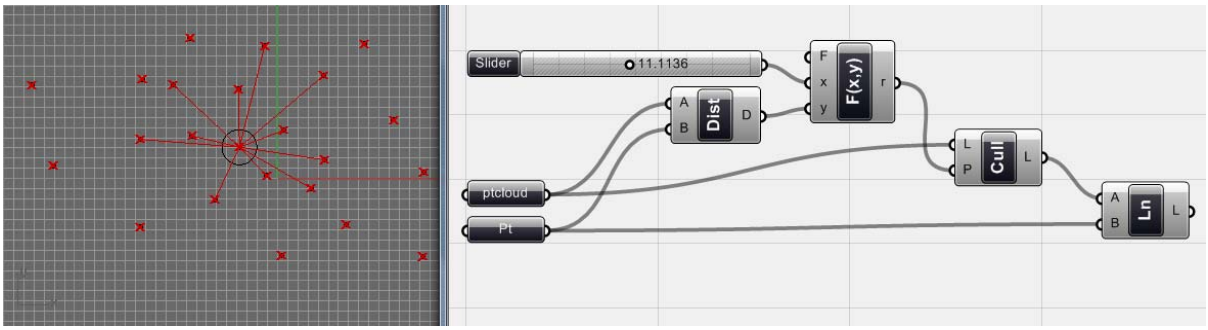


Fig.3.16. Selection of points from a point set based on their distance from a reference point with <Cull pattern> component.

Topography

Having tested the first distance logic, I am thinking of selecting some points which are associated with contour lines on a topography model, based on their height.

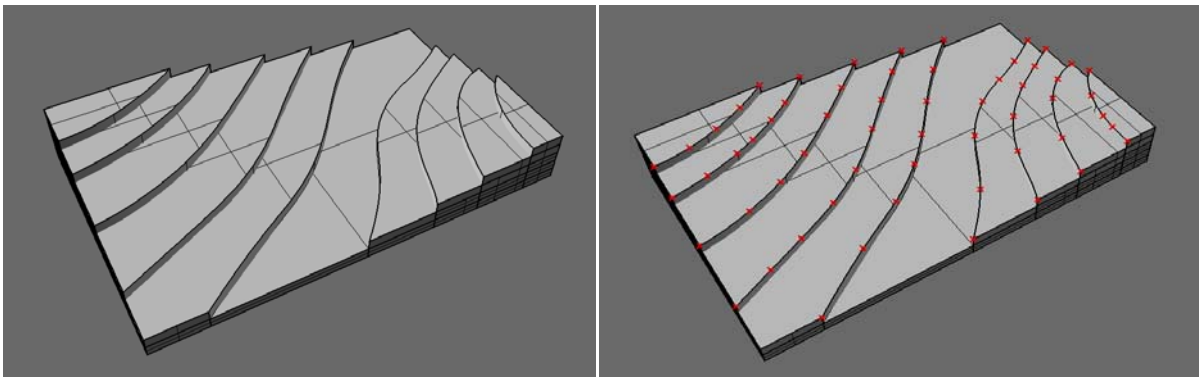


Fig.3.17. Topography with points associated with contour lines.

I want to select these points based on their height. What I have is a point set which is defined by a <point> component (named topography). I need the height of the point with the same logic as the above example to select the specific points. Here I used a <Decompose> component (Vector > Point > Decompose) to get the Z values of these points. I compared these values with a given number (<number slider>) with a <F2> component to produce a list of associative Boolean values. The <Cull pattern> component passes those who associated with the True values which means selected points are higher than the user defined height value (the function of <F2> component defined as f=x>y).

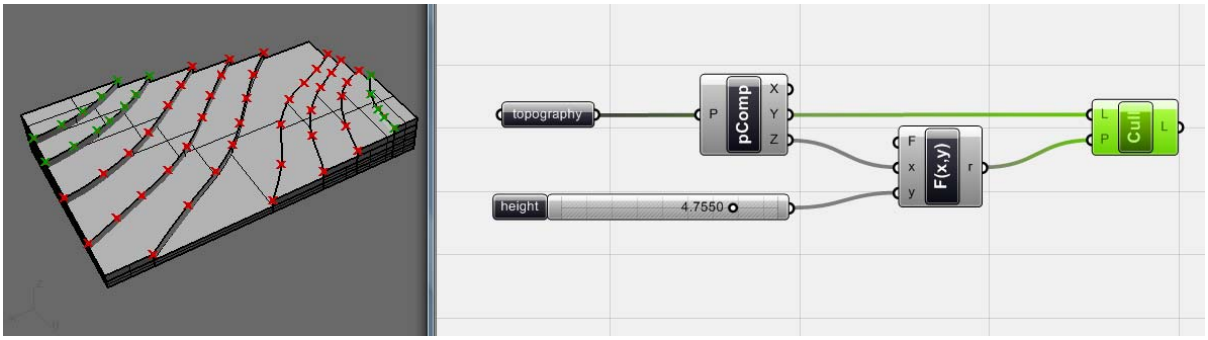


Fig.3.18. Selected points which are higher than 4.7550 units! (A user defined value). These points are now ready to plant your Pine trees!!!!

Connectivity logic: Triangles

Let's have another example of culling this time with <Cull Nth>. Imagine we have a network of points and we want to draw lines to make triangles with a pattern like Figure.3.19.

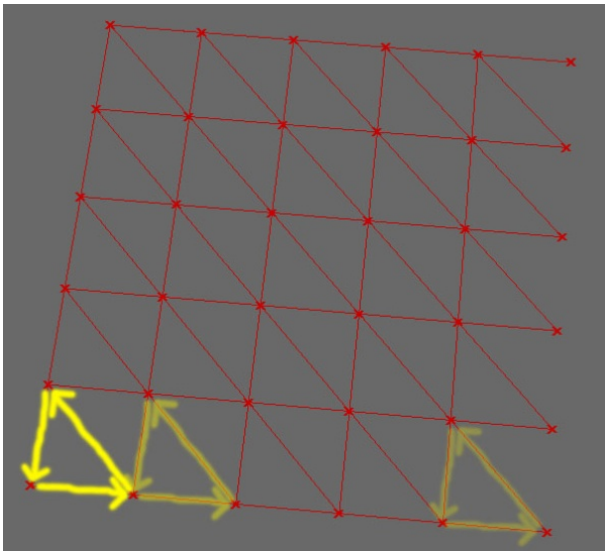


Fig.3.19. Making triangles by a network of points.

The first step is to simply create a grid of points by <series> and <pt> components. The next step is to find the proper points to draw lines in between. Each time we need a line starts from a point and ends at the next point on the next column, then another line goes from there to the back column but at the next row and final line goes back to the start point. To do this, it seems better to make three different lists of points, one for all first points, one for all second points and another for all third points and then draw line between them.

I can use the original points as the list for all start pints.

The first second point is the second point on the point set and then the list goes on one by one. So to select the second points I just shifted the original list by <Shift list> component (Logic > List > Shift list) by shift offset=1 to shift the data set by one value and make the second points list. (Go to the <shift list> help to learn more about component). Since the original data is a set of points which make the start points in our example, the shifted data would be the second point of the triangles (all second points in the network).

The third point of triangles is in the same column as the start point but in next row, so I shifted the original list of points again by shift offset=the number of columns (the value comes from the <number slider>) to find these points for all point set and make a list of all third points.

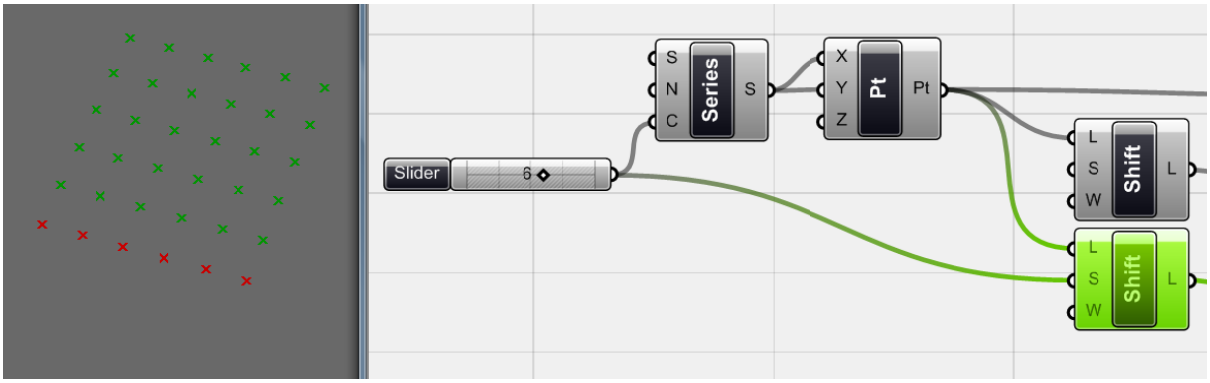


Fig.3.20. Selected item is the shifted points by the shift offset value of equal to the number of columns which produces all third points of the triangles.

To complete the task I need to omit some points in each set. First of all the points in the last column never could be first points of triangles so I need to omit them from the list of start points. The points on the first column also never could be the second points, so I need to omit them from the list of second points and the same for last column again as third points. So basically I attached all points' lists each to one <Cull Nth> component which omits specific members of a data set by a number which is cull frequency (Fig.3.17). In this case all data sets culled by the number of columns which is clear why. So I just connected the <number slider> to each <Cull Nth> component as frequency.

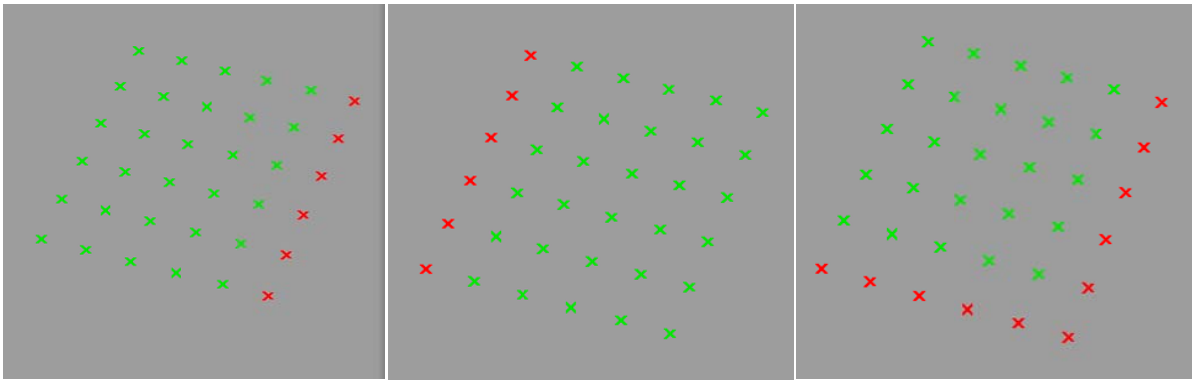


Fig.3.21. Using <Cull Nth> to omit A. last column, B. first column and c. last column of the first, second and third points' lists.

The next step is just to feed three <line> components to connect first points to the second, then second points to the third and finally third points to the first again.

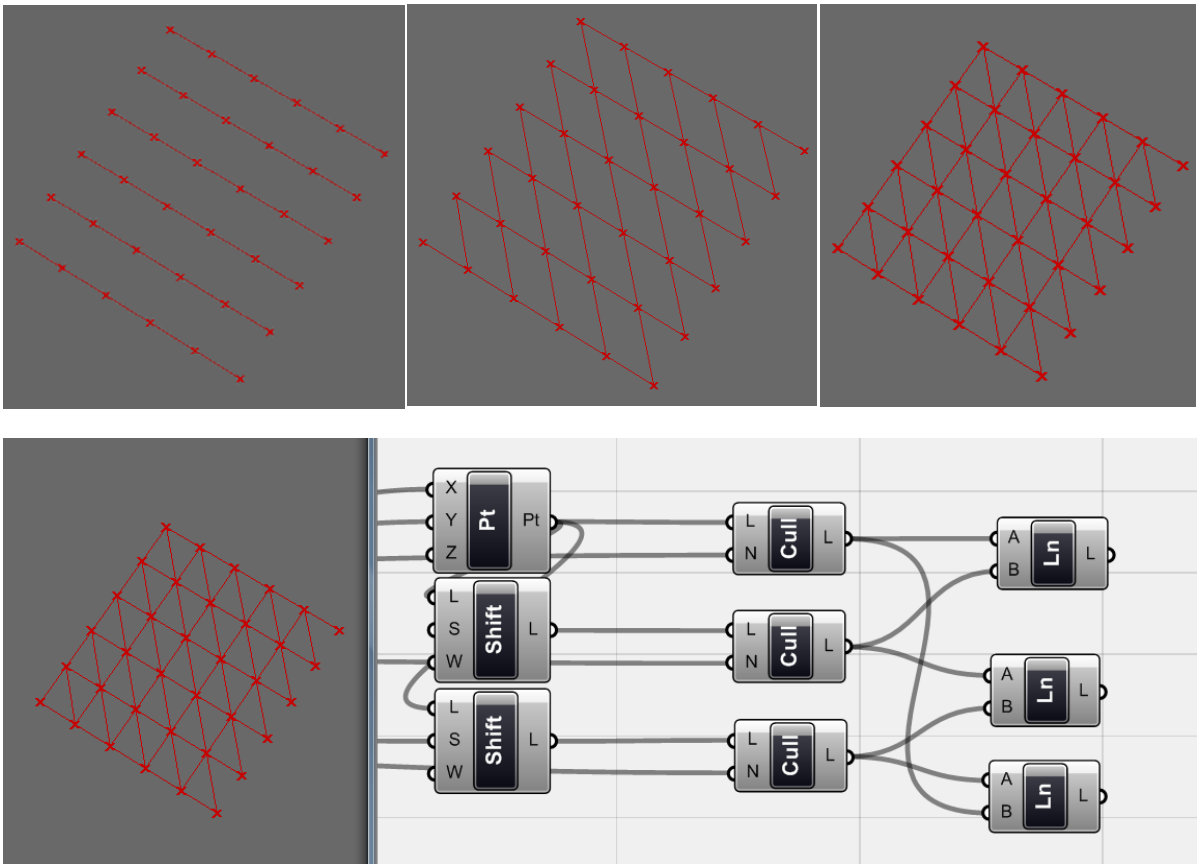


Fig.3.22. Making lines by connecting culled lists of points to the <Line> component.

Now by changing the <number slider> you can have different grids of points which produces these triangles accordingly.

Although there are still some problems with our design and we now that we should not start any triangle from the points of the last row, but the concept is clear…… so let's go further. We will come back to this idea while talking about mesh geometries and then I will try to refine it.

3_7_2D Geometrical Patterns

Geometrical Patterns are among the exciting experiments with the Generative Algorithms and in Grasshopper. We have the potential to design a motif and then proliferate it as a pattern which could be used as a base of other design products and decorations. In case of designing patterns we should have a conceptual look at our design/model and extract the simple geometry that produces the whole shape while being repeated. So by producing the basic geometry we can copy it to produce the pattern as large as we need (Fig.3.23).

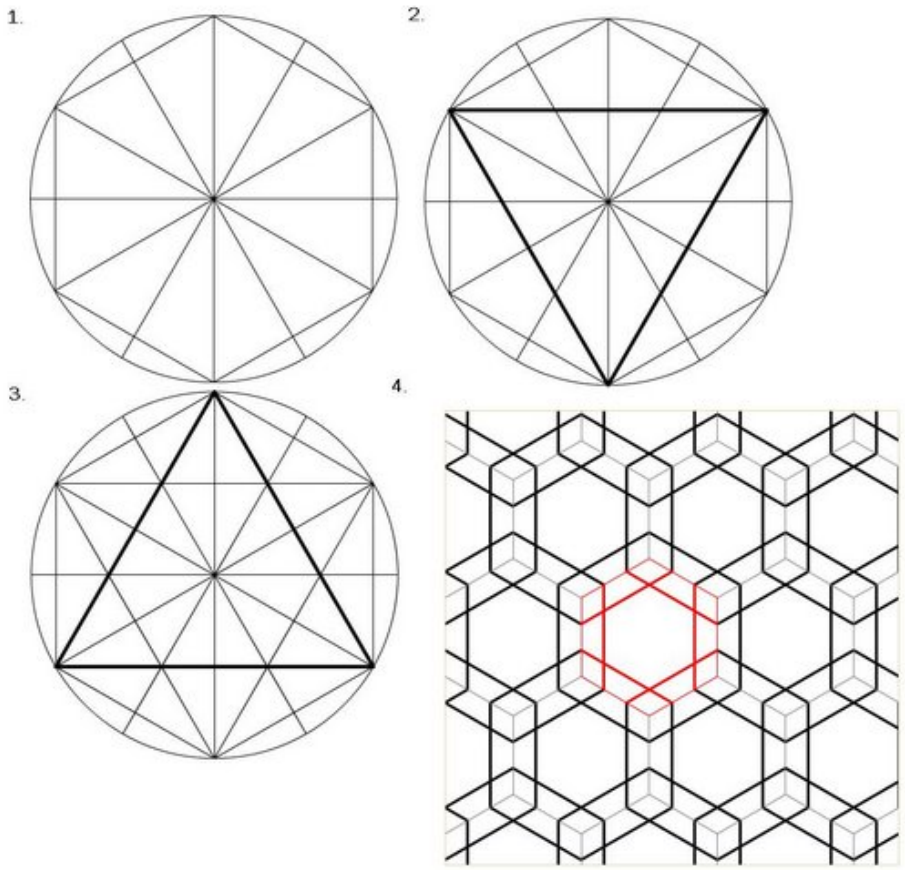


Fig.3.23. Extracting the concept of a pattern by simple geometries.

I still insist on working on this models by data sets and simple mathematical functions instead of other useful components just to see how these simple operations and numerical data have the great potential to generate shapes, even classical geometries.

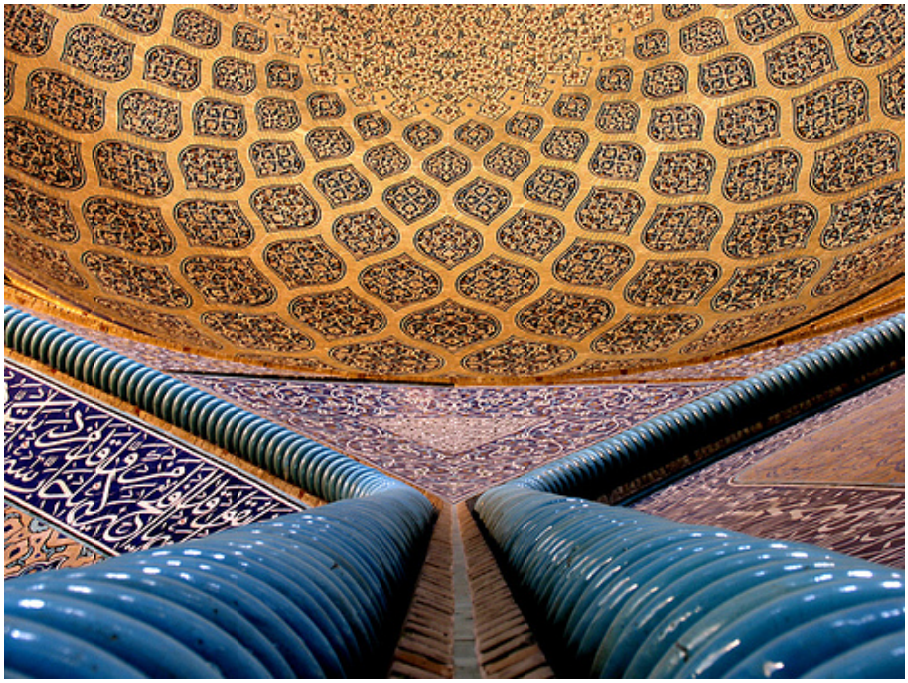


Fig.3.24. Complex geometries of Sheikh Lotfolah Mosque's tile work comprises of simple patterns which created by mathematical-geometrical calculations. Sheikh Lotfolah Mosque, Isfahan, Iran.

Simple linear pattern

Here I decided to make a simple pattern by some intersecting lines and my aim is to draw some patterns similar to Figure.3.25.

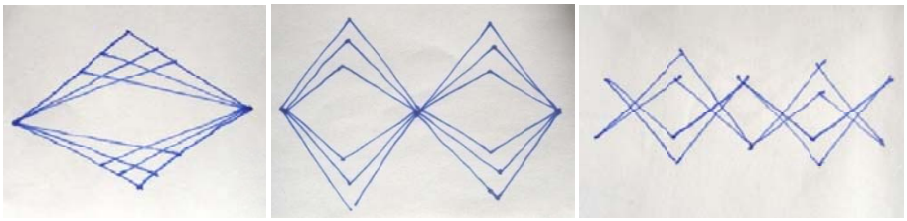


Fig.3.25. Examples of simple concepts to make patterns.

I started my definition by a <series> which I am able to control the number of values (here points) and the step size (here distance between points). By this <series> I generated a set of points and I also generated another three different sets of points with different Y values which I am adjusting them relatively by a <number slider> and <F1> components (y=-x/3, y=x, y=x/3, y=x+(x/3) if x is the number coming from the <number slider>).

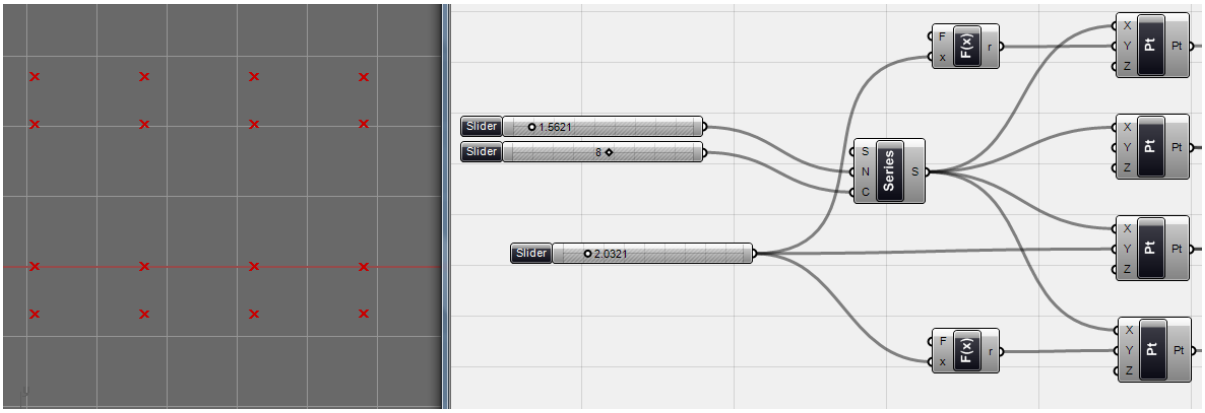


Fig.3.26. Generating four set of points, all associated with a one <series> component which controls the number and distance of points and another <number slider> which controls the Y value of the point sets (I am using longest list for points data matching).

To get the "zig-zag" form of the connections I need to cull the point sets with <cull pattern> one with True/False and another one with False/True pattern and then connect them together (Fig.3.20).

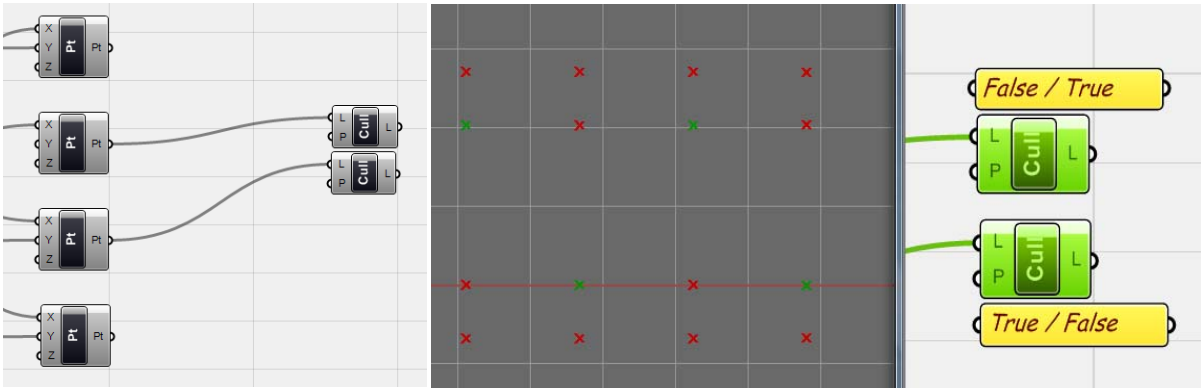


Fig.3.27. Cull pattern and selected points as a base for "zig-zag" pattern.

Since I want to draw poly line on this culled point sets, I cull all point sets with the same logic and merge these points to make a one merged stream by <merge 02> component (Logic > Streams > Merge 02).

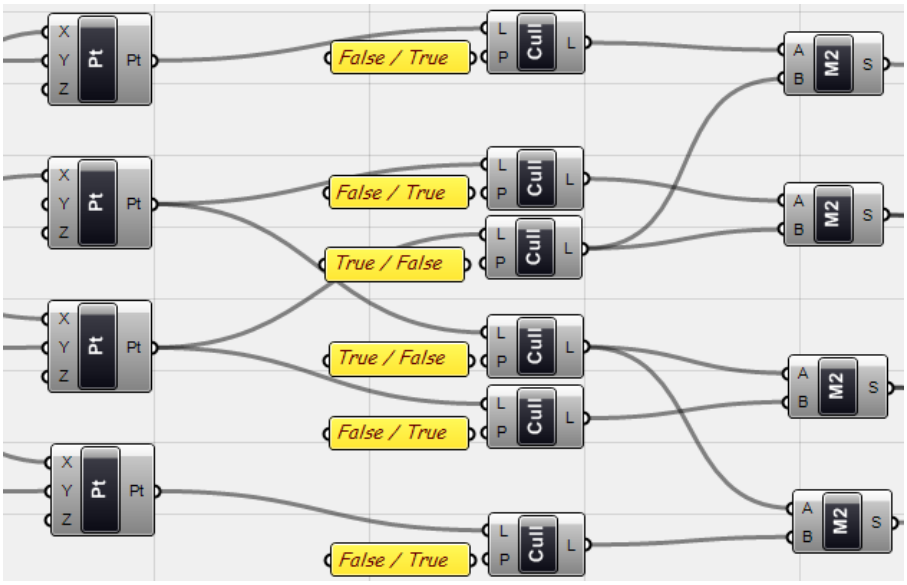


Fig.3.28. Cull pattern for the middle points inverted to make the mirrored pattern of the first set.

To make the second rows of points, I used the same points of the first row and merge them again with the second row to have four merged data streams at the end. If you connect these merged data sets to a <poly line> component you will get a z-shaped poly line and that is because the points are not in a desired order and the <merge> component just add the second list of points at the end of first one. So I need to sort the points in the desired way. A <Sort> component sorts some generic data based on a sortable key. What do we have as a sortable key?

If we look at the order of the points we can see that the X dimension of the points increases incrementally, which seems suitable as an item to sort our points with. To do this, we need to extract the X coordinate of the points. The <decompose> component make it possible. So we connect the X coordinate of the points as a sortable key to the <sort> component and then sort the points with that. Finally we can use these sorted points to feed <polyline> components, make our pattern with them (Fig.3.29).

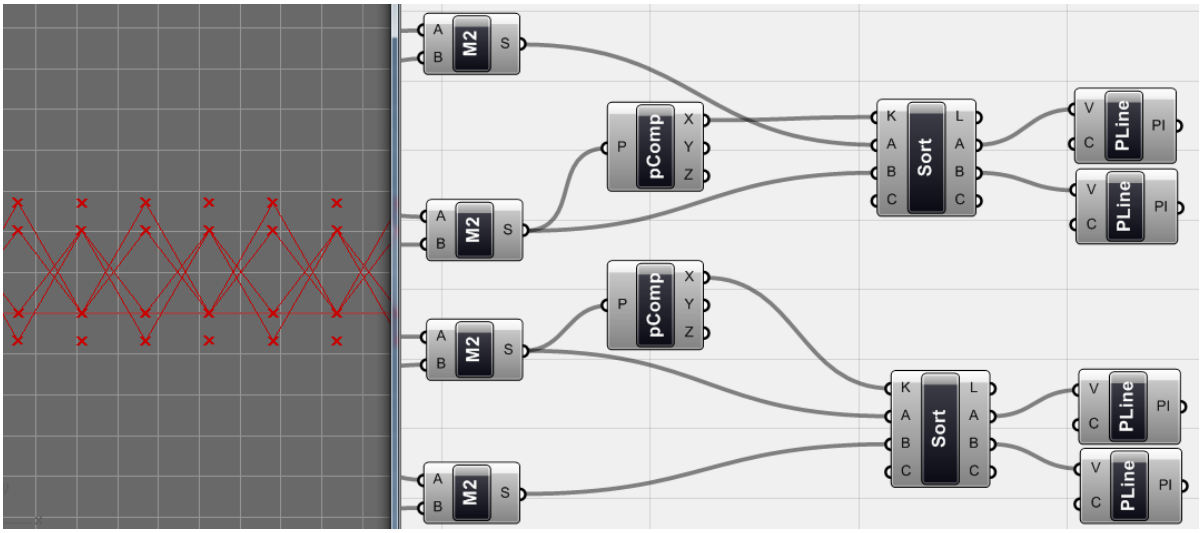


Fig.3.29. Sorting points by their X component as key and then making polyline with them. I sorted mirrored geometry by another <sort> component because they are from different cull pattern.

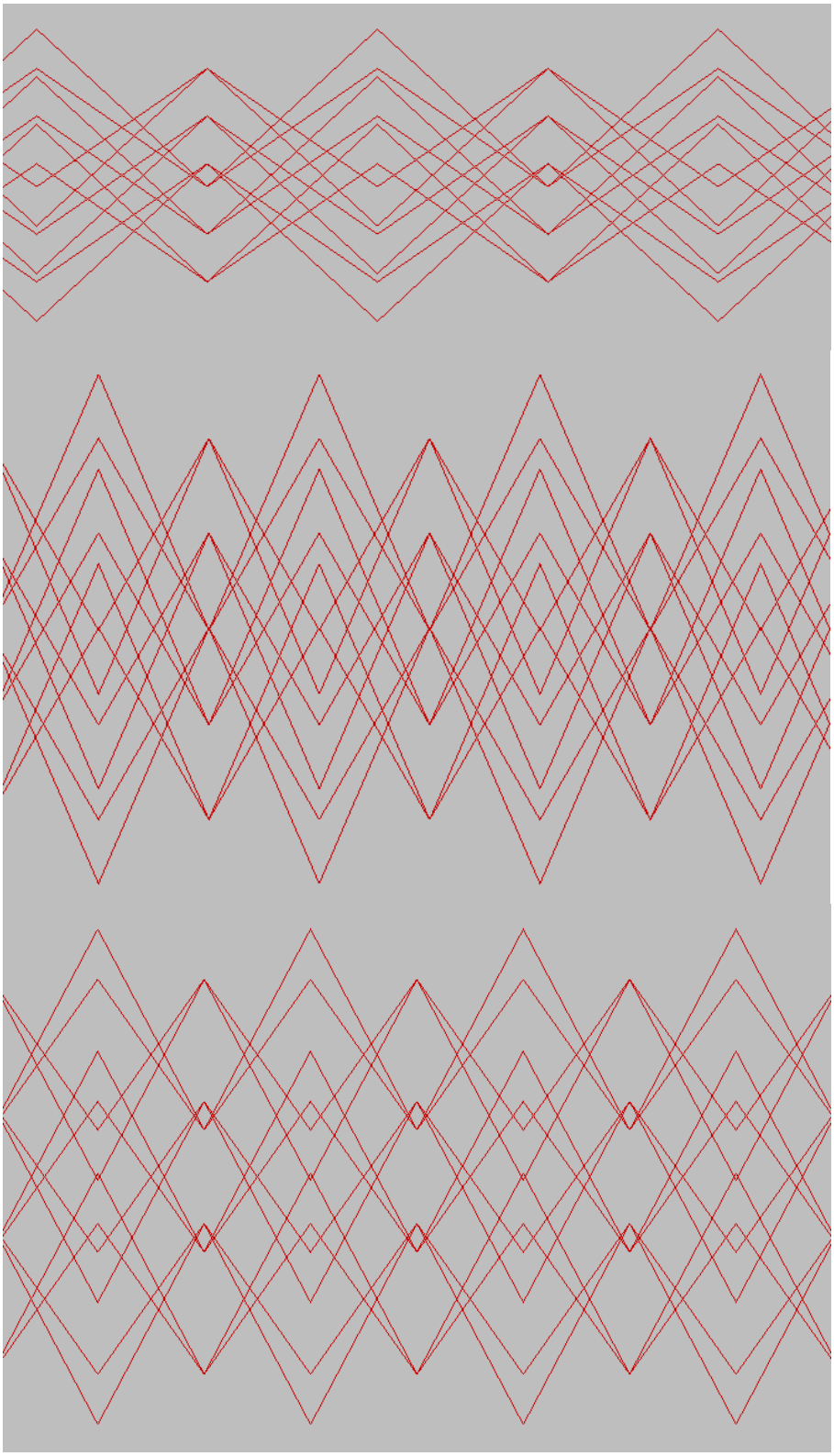


Fig.3.30. Later on we will discuss how we could create repetitive patterns by simple components and the way we can array the simple motif to create complex geometries.

Circular patterns

There are endless possibilities to create motifs and patterns in this associative modelling method. Figure.3.31 shows another motif which is based on circular patterns rather than the linear one. Since there are multiple curves which all have the same logic I will just describe one part of the algorithm and keep the rest for you.

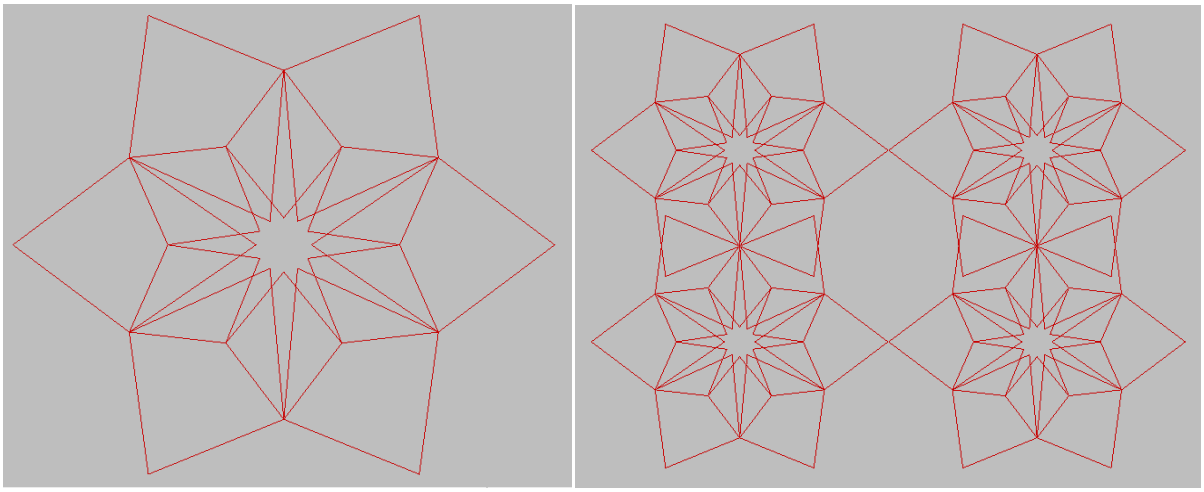


Fig.3.31. Circular geometrical patterns which is repeated in the second picture.

The start point of this pattern is a data set which produces a bunch of points along a circle, like the example we have done before. This data set could be rescaled from the centre to provide more and more circles around the same centre. I will cull these sets of points with the same way as the last example. Then I will generate a repetitive 'zig-zag' pattern out of these rescaled-circular points to connect them to each other, make a star shape curve. Overlap of these stars could make motifs and using different cull patterns make it more interesting.

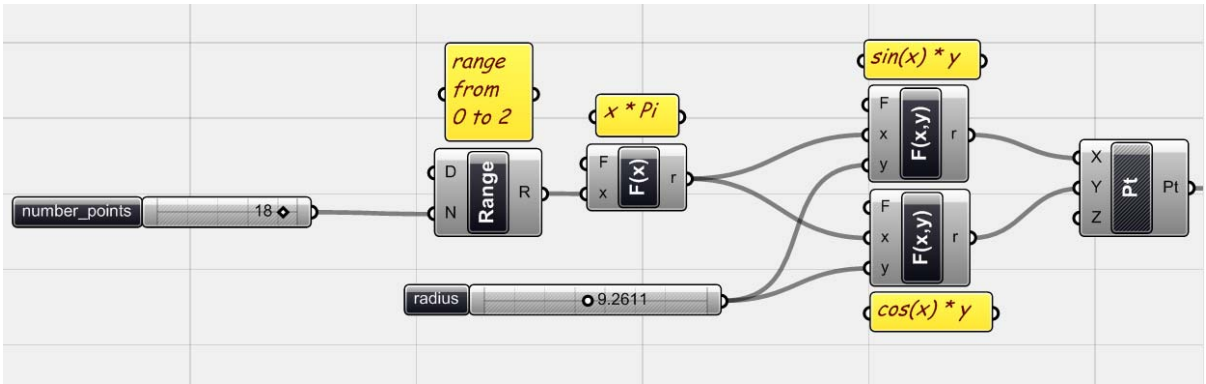


Fig.3.32. Providing a range of 0 to 2Pi and by using Sin/Cos functions, making a first set of points. The second <number slider> changes the radius of the circle (power of the X and Y).

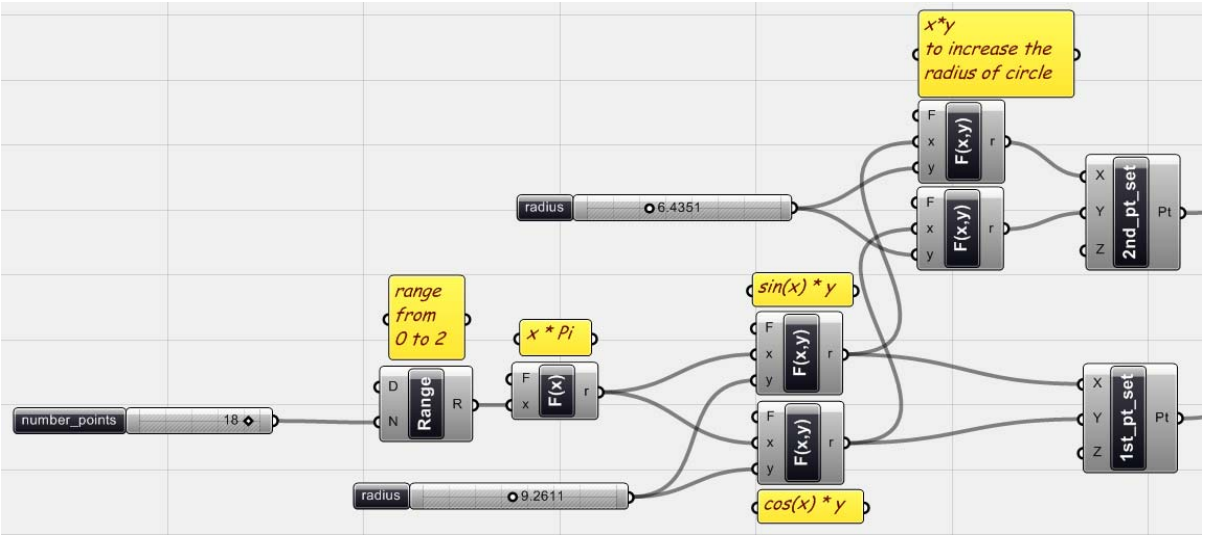


Fig.3.33. Increasing the numbers of Sin/Cos functions by a <number slider> making the second set of points with bigger radius.

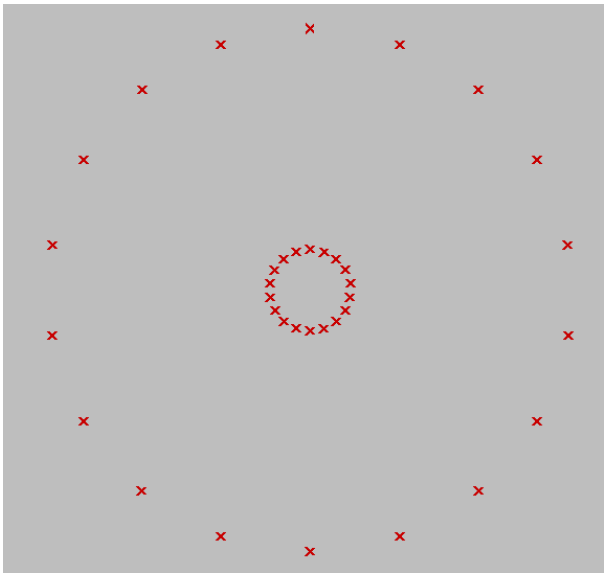


Fig.3.34. First and second circles made by points.

In order to cull the points, we can simply use the <Cull pattern> for the points and use True/False like the last example. But how we can sort the list of points after all? If you connect the culled points to a <poly line> component you will not get a star shape poly line but two offset polygon connected to each other. Here I think it is better to sort the points based on their index number in the set. Because I produced the points by a <range> component, here we need a <series> component to provide the indices of the points in the list. The N parameter of the <range> factor defines the number of steps of the range so the <range> produces N+1 number. I need a <series> with N+ 1 value to be the index of the points (Fig.3.34) and cull and sort these points based on their indices.

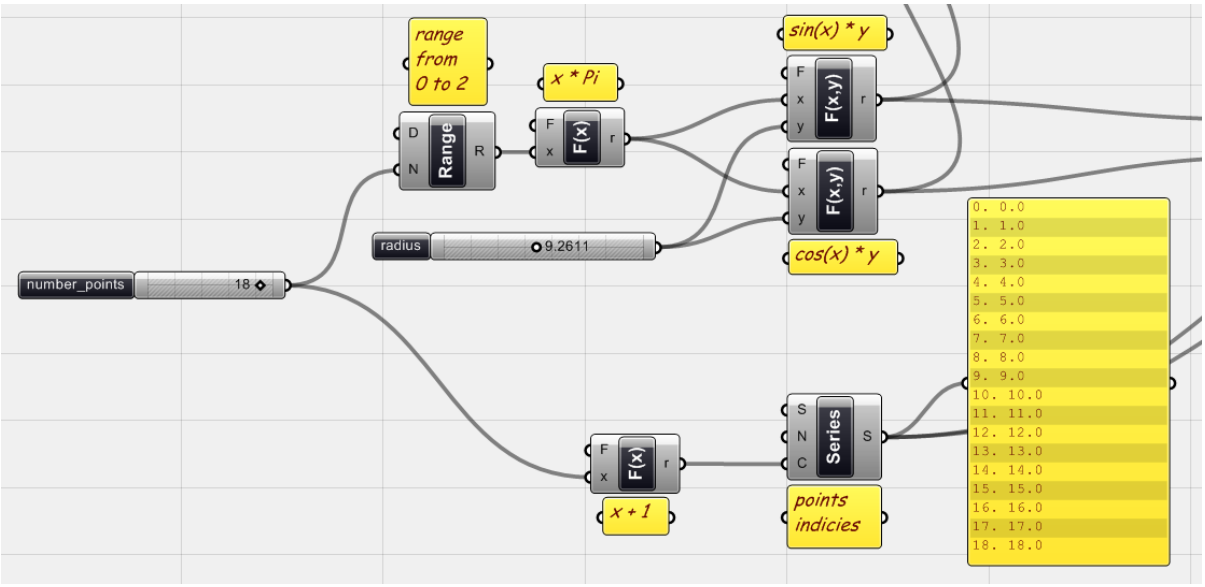


Fig.3.35. Generating index number of the points.

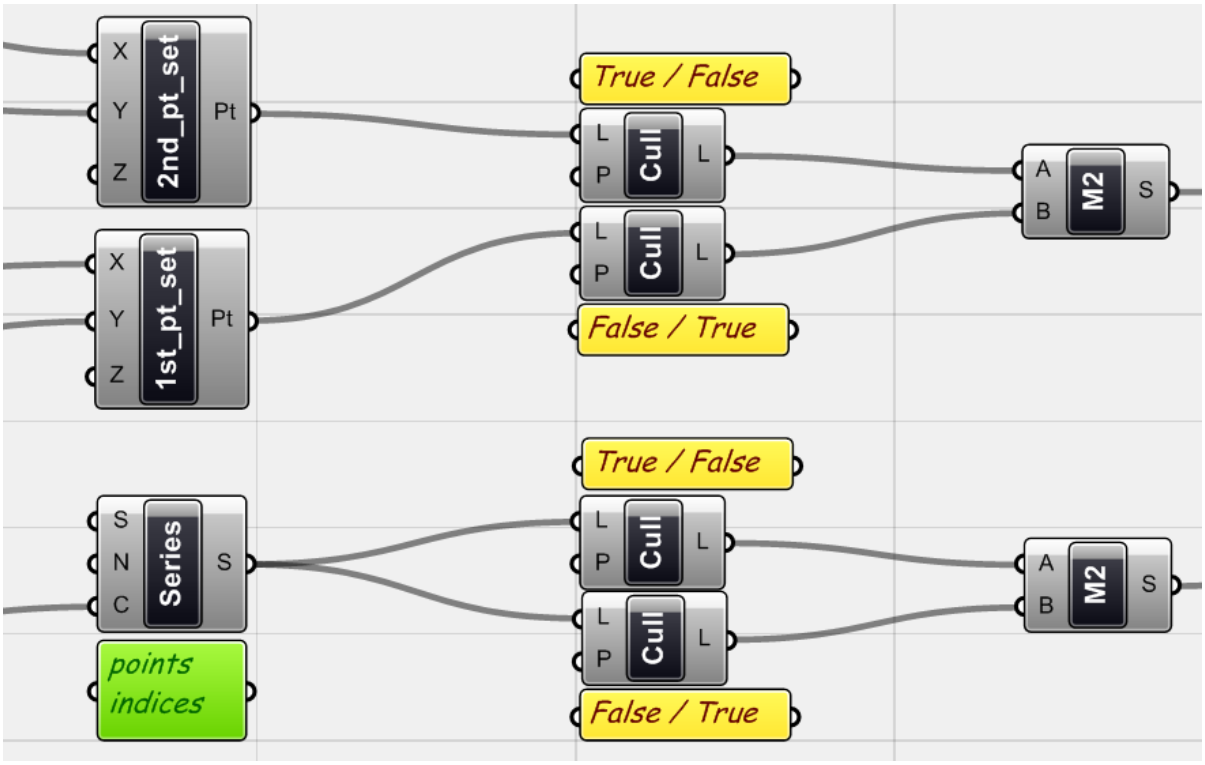


Fig.3.36. We need to cull indices and points the same and merge them together. Although the result of the merging for <series> could be again the numbers of the whole data set, the order of them is like the points, and so by sorting the indices as sortable keys we can sort the points as well. The only thing remain is to feed a <polyline> component by sorted points.

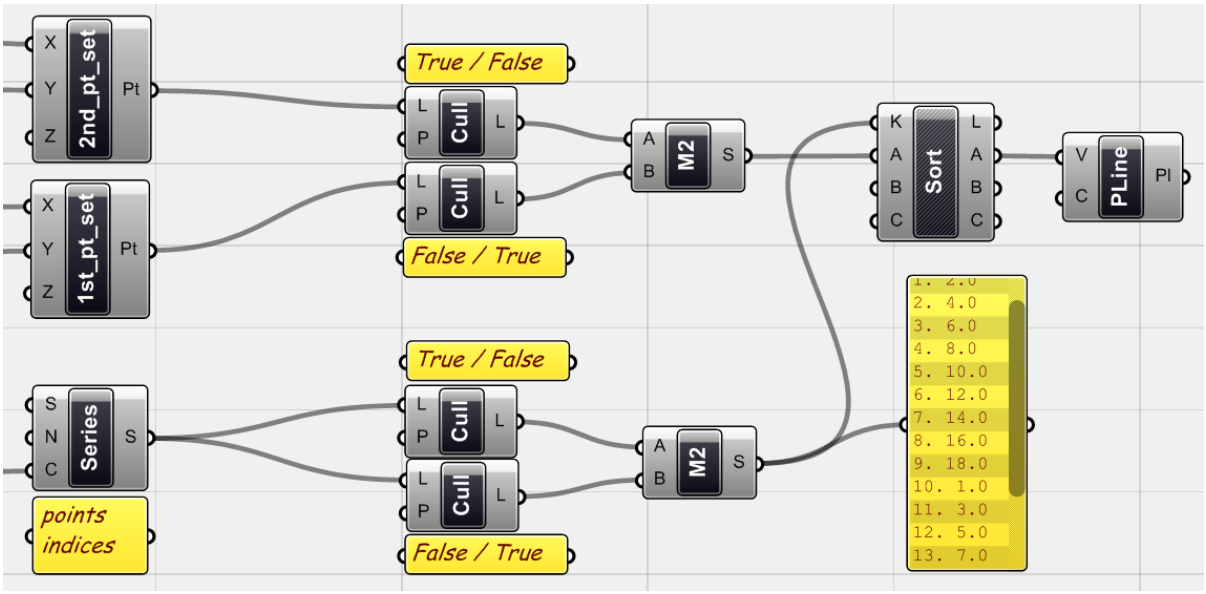


Fig.3.37. Generating Polyline by sorted points.

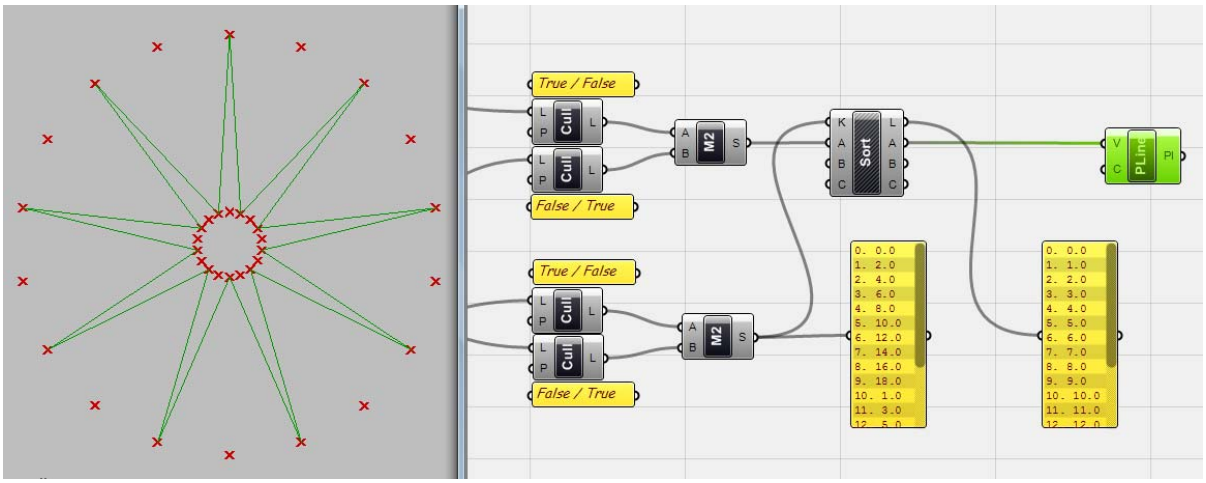


Fig.3.38. Star-shaped polyline.

The same logic could be used to create a more complex geometry by simply generating other point sets, culling them and connecting them together to finally produce patterns. We can use these patterns as inputs for other processes and design other decorative shapes.

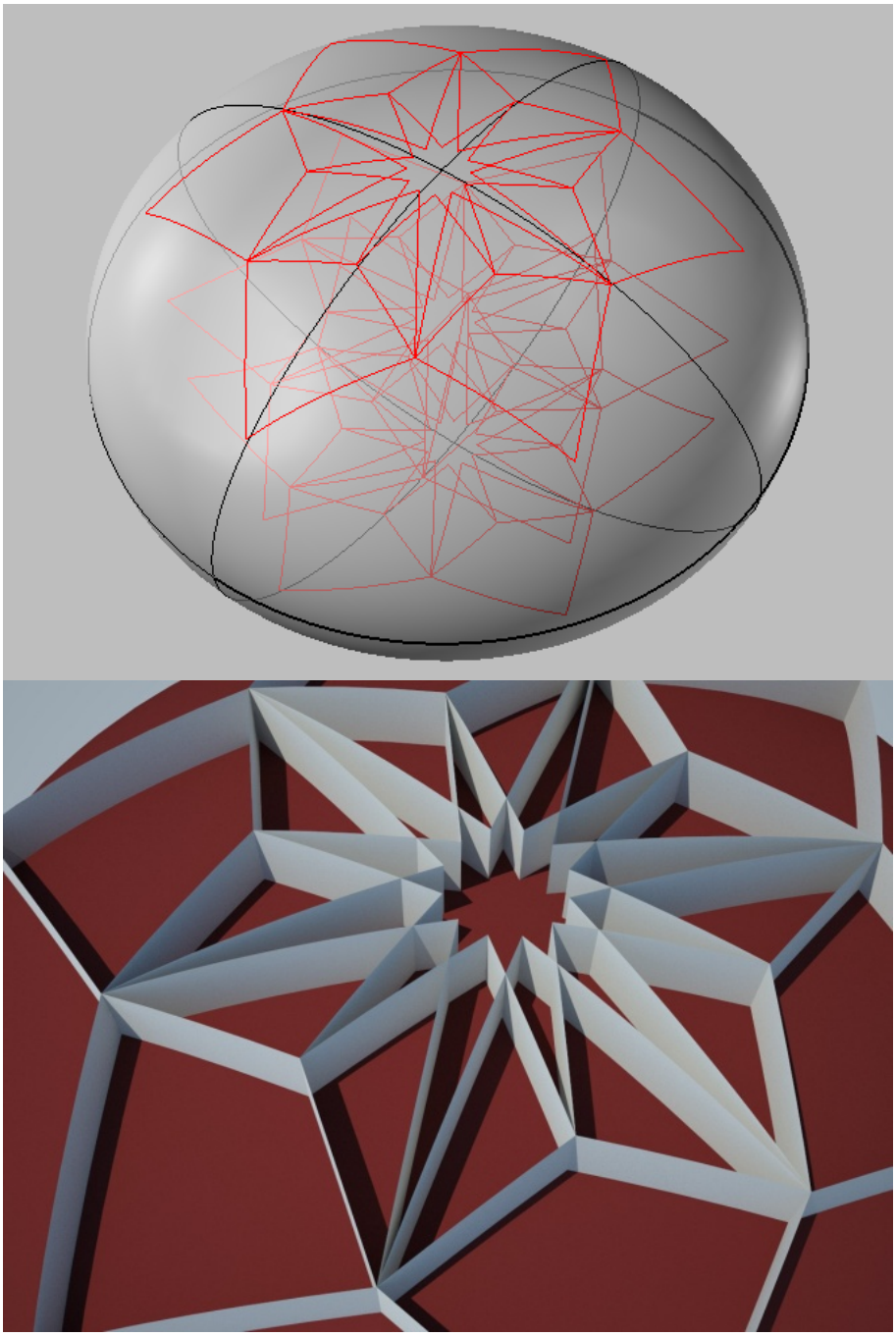


Fig.3.39. You can think about different possibilities of the patterns and linear geometries in applications.

Although I insisted to generate all previous models by data sets and simple mathematical functions, we will see other simple components that made it possible to decrease the whole process or change the way we need to provide data.

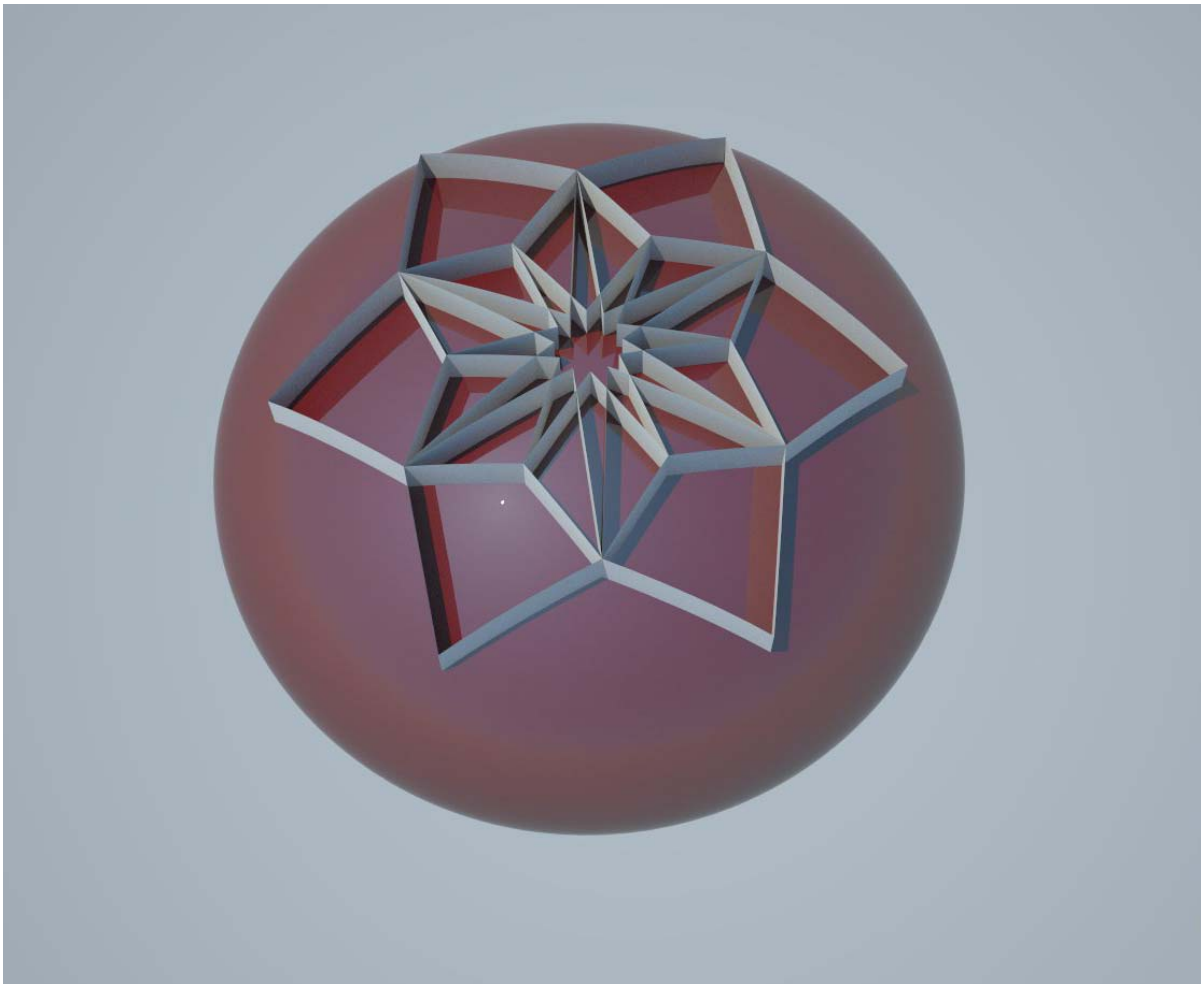
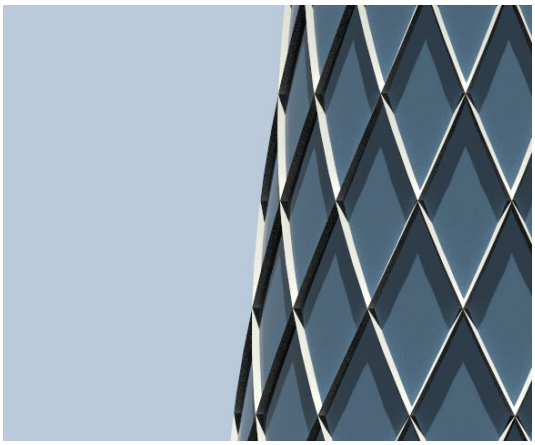


Fig.3.40. Final model.

Chapter_4_Transformation



Chapter_4_Transformation

Transformations are essential operations in modelling and generating geometries. They can enable us to get variations from the initial simple geometries. Transformations help us to scale and orientate our objects, move, copy and mirror them, or may result in accumulation of objects, that could be the desired model we. There are different types of transformations but to classify it, we can divide it to main branches, the first division is linear and the second is spatial transformations. Linear transformations perform on 2D space while spatial transformations deal with the 3D space and all possible object positioning.

In other sense we can classify transformations by status of the initial object; transformations like translation, rotation, and reflection keep the original shape but scale and shear change the original state of the object. There are also non-linear transformations. In addition to translation, rotation and reflection we have 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 space.

In order to transform objects, conceptually we need to move and orientate objects (or part of objects like vertices or cage corners) in the space and to do this we need to use vectors and planes as basic constructs of these mathematical/geometrical operations. We are not going to discuss about basics of geometry and their mathematical logic here but first let's have a look at vectors and planes because we need them to work with.

Fig.4.1. Transformations provide great potential to generate forms from individuals. Nature has some great examples of transformations in its creatures.

4_1_Vectors and planes

Vector is a mathematical/geometrical object that has magnitude (or length) and direction and sense. It starts from a point, go toward another points with certain length and specific direction. Vectors have wide usage in different fields of science and in geometry and transformations as well.

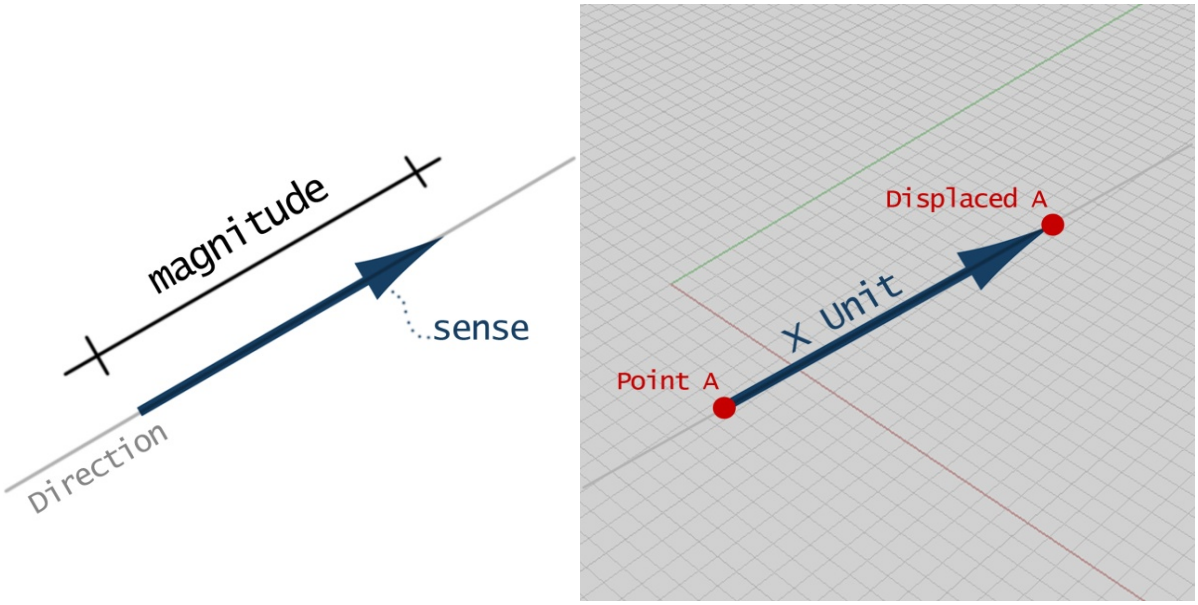


Fig.4.2. A: Basic elements of a Vector, B: point displacement with a vector.

Simply if we have a point and a vector, this vector could displace the point with the distance of vector's magnitude and toward its direction to create a new position for the point. We use this simple concept to generate, move, scale and orientate geometries in our associative modelling method.

Planes are another useful set of geometries that we can describe them as infinite flat surfaces which has an origin point. Construction planes in Rhino are these types of 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 the 3D space we cannot orientate an abject on a vector! but we need two vector to make a plane to be able to put geometry on it.

Vectors have direction and magnitude while planes have orientation and origin. So they are two different types of constructs that can help us to create, modify, transform and articulate our models.

Grasshopper has some of the basic vectors and planes as predefined components. These are including X, Y and Z vectors and XY, XZ, and YZ planes. There are couple of other components which we can produce and modify them which we talk about them in our experiments. So let's jump into design experiments and start with some of the simple usage of vectors and go step by step forward.

4_2_On curves and linear geometries

As we have experimented with points that are 0-Dimension geometries now we can start to think about curves as 1-Dimensional objects. Like points, curves could be the base for constructing so many different objects. We can extrude a simple curve along another one and make a surface, we can connect different curves together and make surfaces and solids, we can distribute any object along a curve with specific intervals and so many other ways to use a curve as a base geometry to generate other objects.

Displacements

We generated so many point grids in chapter 3. There is a component called <Grid rectangular> (Vector > Point > Grid rectangular) which produces a grid of points which are connected together make some cells also. We can control the number of points in X and Y direction and the distance between points.

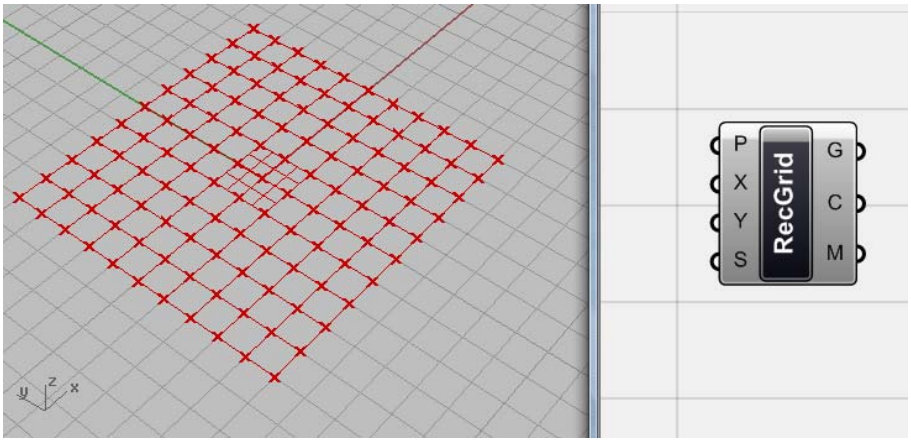


Fig.4.3. a simple <Grid Rectangular> component with its predefined values.

You can change the size of grid by a <number slider>. I want to change the Z coordinates of the points as well. So I need to change the base plane of the grid. To do this, I introduced a <XY plane> component (Vector > Constants > XY plane) which is a predefined plane in the orientation of the X and Y axis and I displaced it in Z direction by a <Z unit> component (Vector > Constants > Z unit) which is a vector along Z axis with the length (magnitude) of one. I can change the height of this displacement by the size of the vector through a <number slider> that I connected to the input of the <Z unit> component. So by changing the position of the <XY plane> along the Z axis the height of the grid also changes.

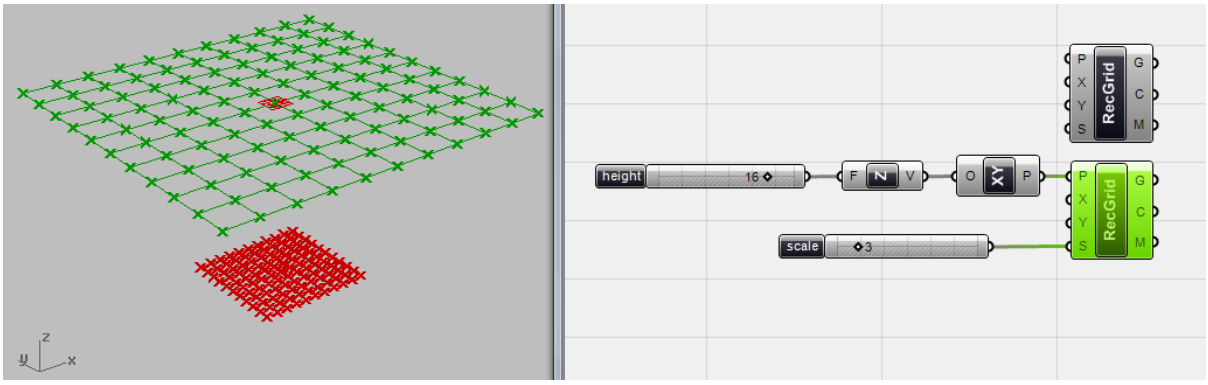


Fig.4.4. Manipulated Grid (selected in green) with one <number slider> for scale of the grid and another with a <Z unit> and <XY plane> to change the Z coordinate of the grid's points. (Further you can just connect the <Z> vector component to the <Grid rectangular> component and get the same result).

Now if you look at the output of the <grid rectangular> you can see that we have access to the whole points as well as grid cells and cell centres. I am looking for a bunch of lines that start from the grid cells' centre points and spread out of it to the space. Although we can simply connect these points from the two <grid> component M part to a <line> component, the length of lines in this case would be different. But as I want to draw lines with the same length, I need another strategy. Here, I am going to use a <line SDL> component. This component draws a line by Start point(S), Direction (D), and Length (L). So exactly what I need; I have the start points (cell's midpoint), and length of my lines. What about the direction? Since the direction of my lines are in the direction of the lines that connect the mid points of the cells of the two grids, I am going to make a set of vectors by these to point sets.

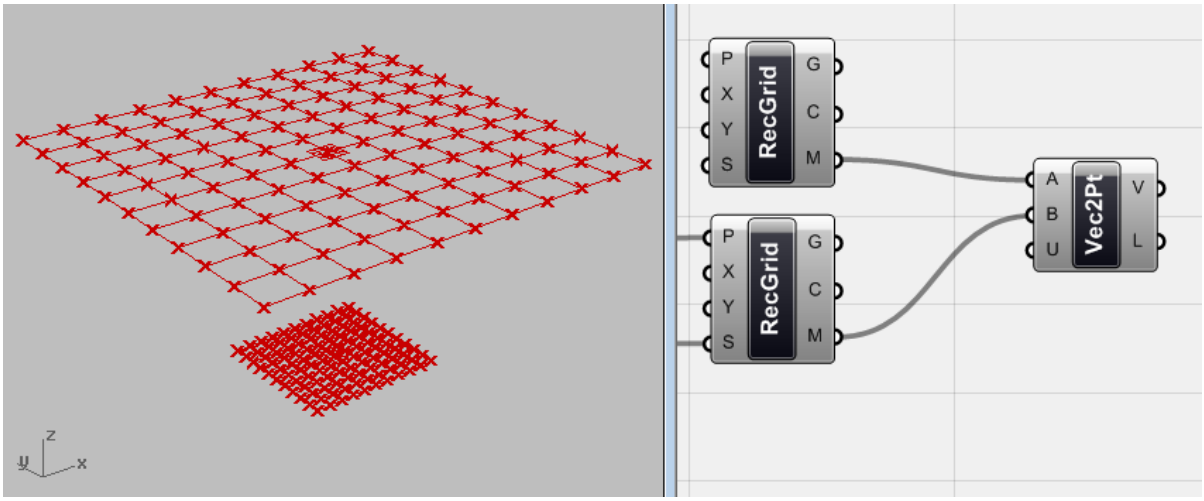


Fig.4.5. Making vectors from the cells midpoints of the first grid toward the cells midpoints of the second grid by <vector 2pt> component (Vector > Vector > vector 2pt). This component makes vectors by the start and end point of vectors.

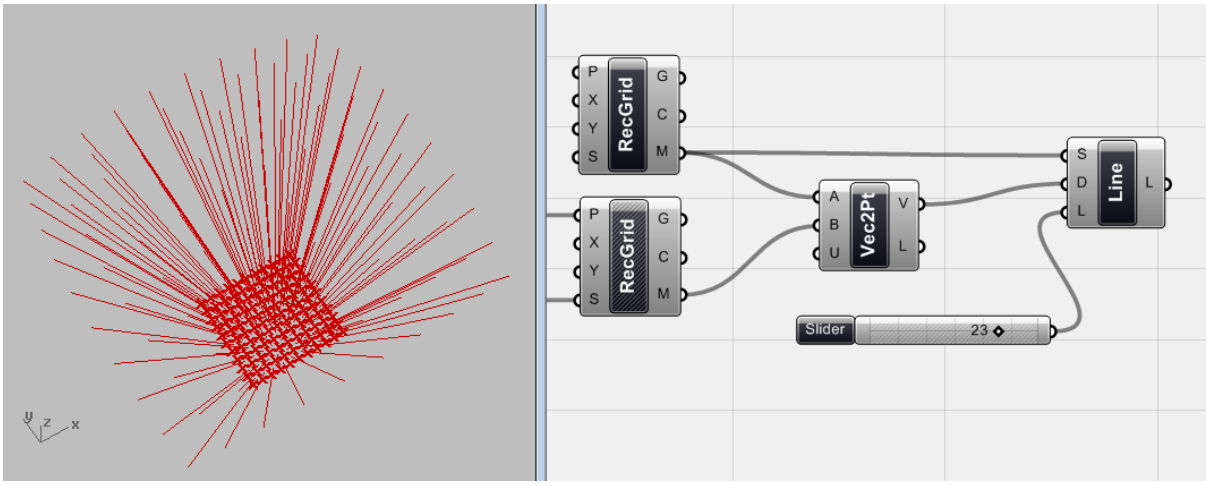


Fig.4.6. The <line SDL> component generates bunch of lines from the grid cell midpoints that spread out into space. I can change the length of lines by <number slider>.

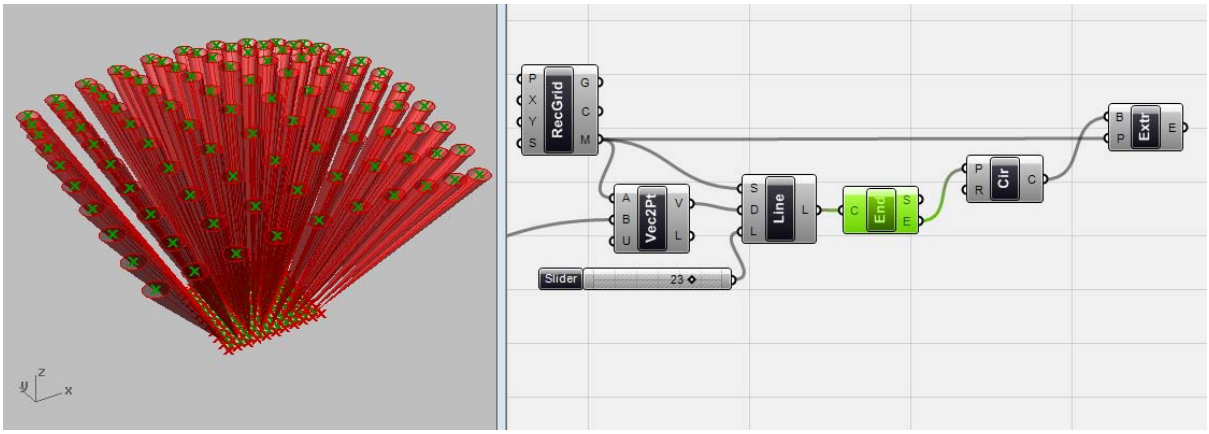


Fig.4.7. By simply using an <end points> component (Curve > Analysis > End points) and using these 'end points' as the 'base points' for a set of <circle> components (Curve > Primitive > Circle) and extruding these circles by <extrude point> component we can generate a set of cones, pointing toward same direction and we can finish our first experiment.

Random displacements

I decided to make a set of randomly distributed pipes (lines) with random length, woven to each other, make a funny element. I sketched it like Figure.4.8.

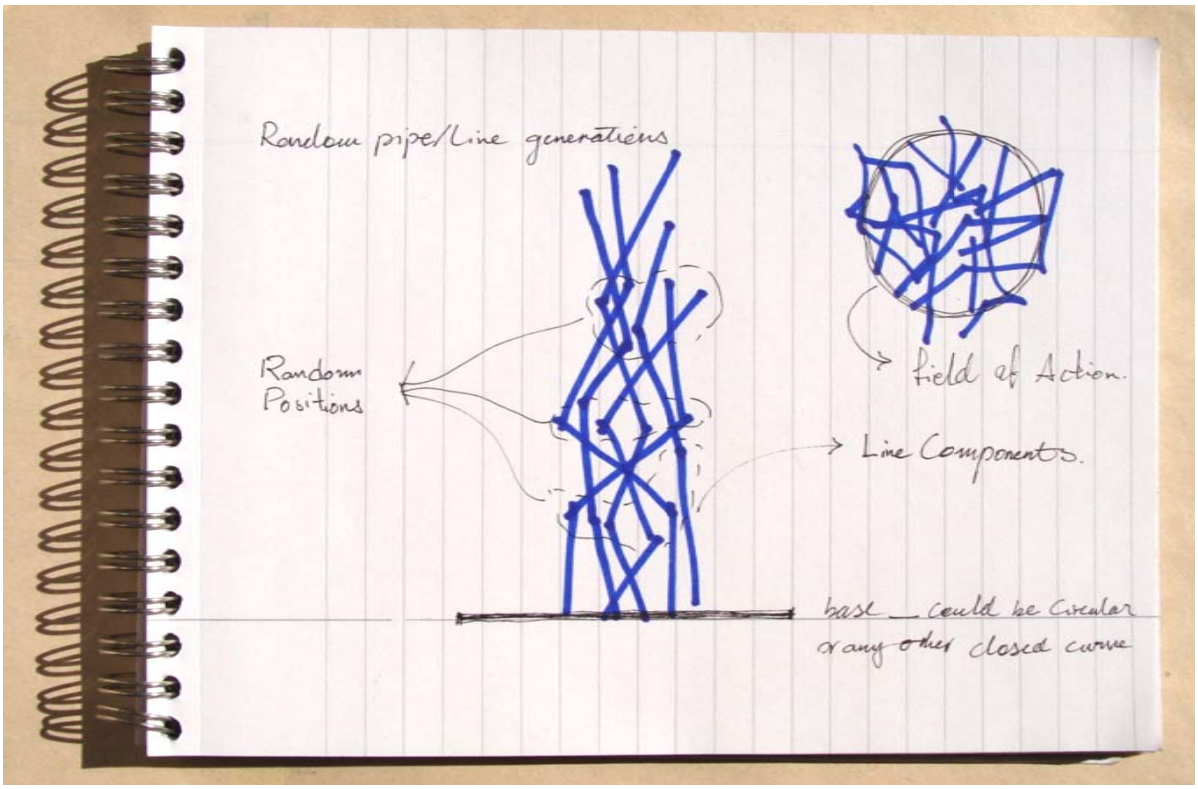


Fig.4.8. First sketches of model

There are different ways to design this model. I am thinking of making a circle as the base curve, divide it into desire parts and then generate some random curves from these points and then make pipes by these curves. But let's see how we can do it in Grasshopper.

As I said, first I want to do it with a circle to divide it and create the base points. I used a <circle> component (Curve > Primitive > Circle) and I attached a <number slider> for further changes of its radius. Then we attach our circle to a <divide curve> component (Curve > Division > Divide curve) to divide the circle. Again we can control the number of divisions by an integer <number slider>. The <Divide curve> component gives me the points on the curve (as the division points). These are the first set of points for generating our base lines.

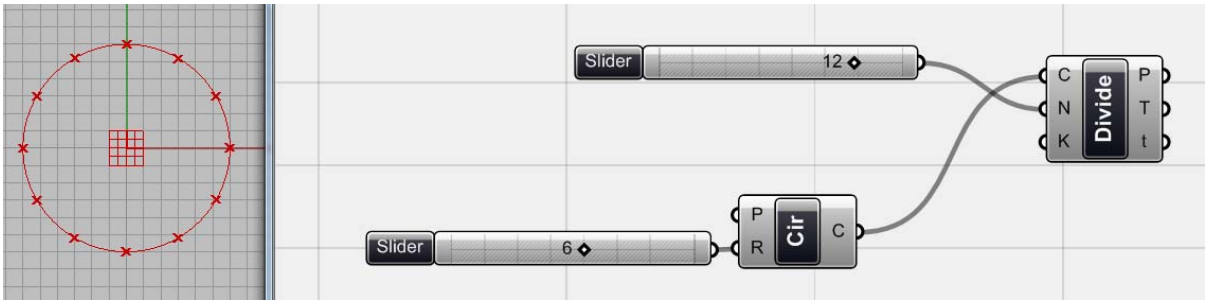


Fig.4.9. Dividing the base circle.

In this step, for some reasons! I want to generate these base lines in different segments. In order to draw our first step, I need the second set of points, above the first set (to make the lines semi-horizontal !!!!) and in a randomly distributed field. To do this, I want to make a set of random vectors in Z direction and displace the first set of points with these random vectors. By connecting the first and second sets of points we would have our first part of our lines.

Because I need the same amount of vectors as the base points, I used the value of the <number slider> that controls the amount of base points (circle divisions), to generate the same amount of random vectors. So I connect the <number slider> of <divide curve> component to a <random> component N part to generate N random values. Then I used a <unit Z> component but I feed this <Unit Z> vector component with the <random> component so it produces N random length vectors in Z direction accordingly.

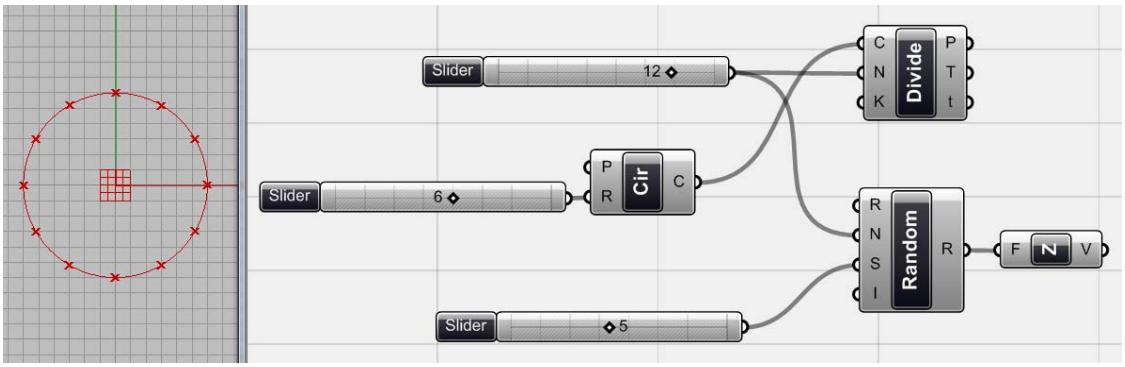


Fig.4.10. Generating random length vectors in Z direction. The <number slider> which is connected to the S part of the <random> component differs the random numbers as 'seed' of random engine.

So now we are ready to displace the points by these vectors. Just bring a <move> component (XForm > Euclidian > Move) to the canvas. Basically a <move> component moves geometries by given vectors. You can move one object/a group of objects with one vector/a group of vectors. Since you still have the component of the source geometry, the <move> component works like a 'copy' command in Rhino. To see how these vectors displaced the points, we can use a <line> component to see the result.

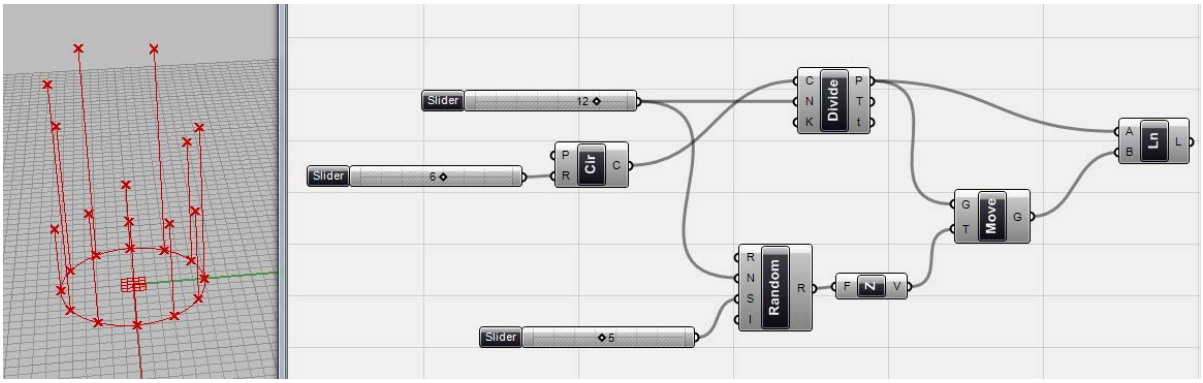


Fig.4.11. The first set of lines that made by the randomly displaced points.

As you see in the Figure.4.11 the lines are vertical and I am looking for more random distribution of points to make the whole lines penetrating each other. I will do it by a <jitter> component to simply shuffle the end points.

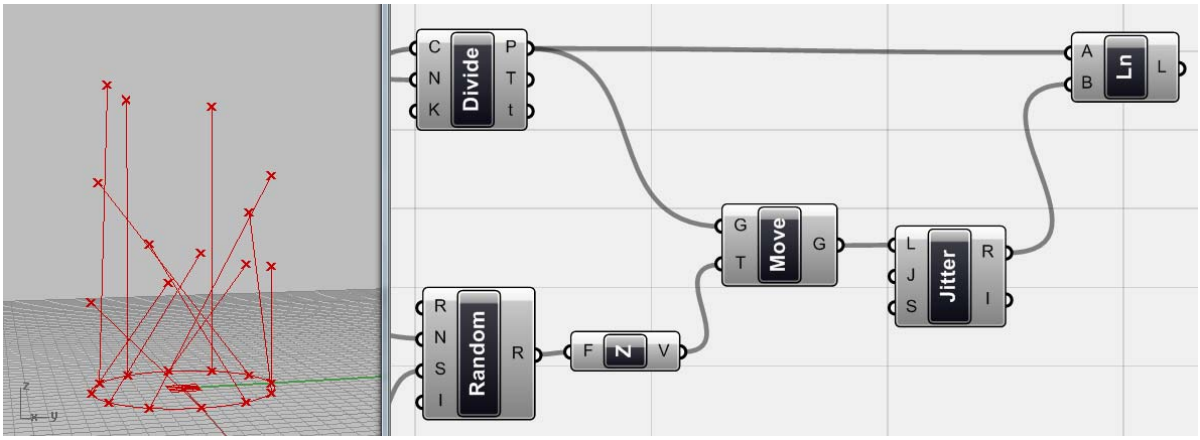


Fig.4.12. Shuffling the end points and making totally random lines.

The rest of the process is simple. I think three segments of lines would be fine. So I just repeat the move and shuffling concept to make second and third set of displaced points.

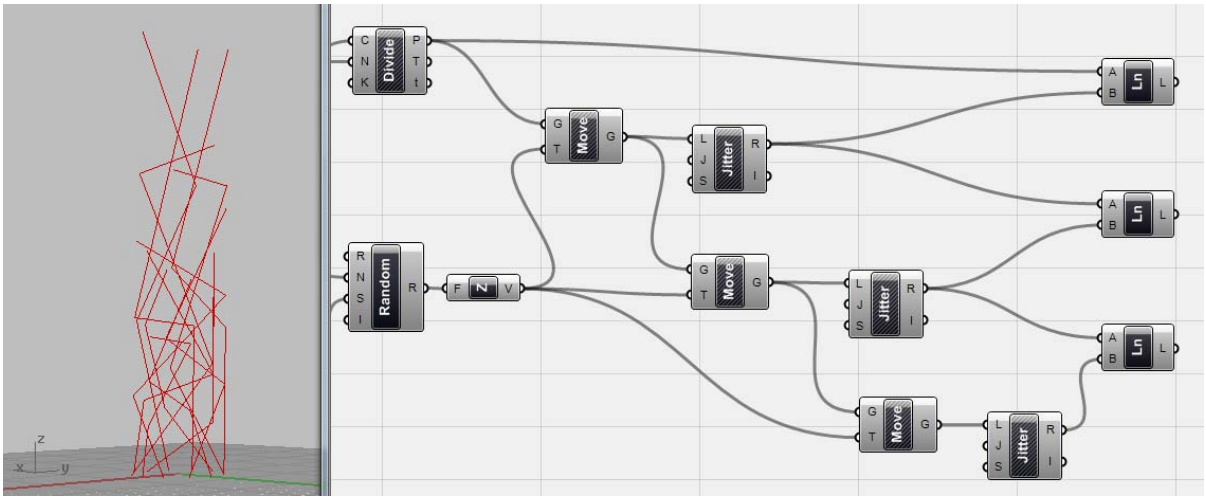


Fig.4.13. Generating 3 line segments for each path to be converted to pipes. To clean up the scene you can uncheck the other components and just preview the lines.

Ok. We have the base geometry. Now just add a <pipe> component (Surface > Freeform > pipe) and attach all <line> components (by holding shift key!) to the 'base curve' part of the component and use a <number slider> to control the radius of pipes.

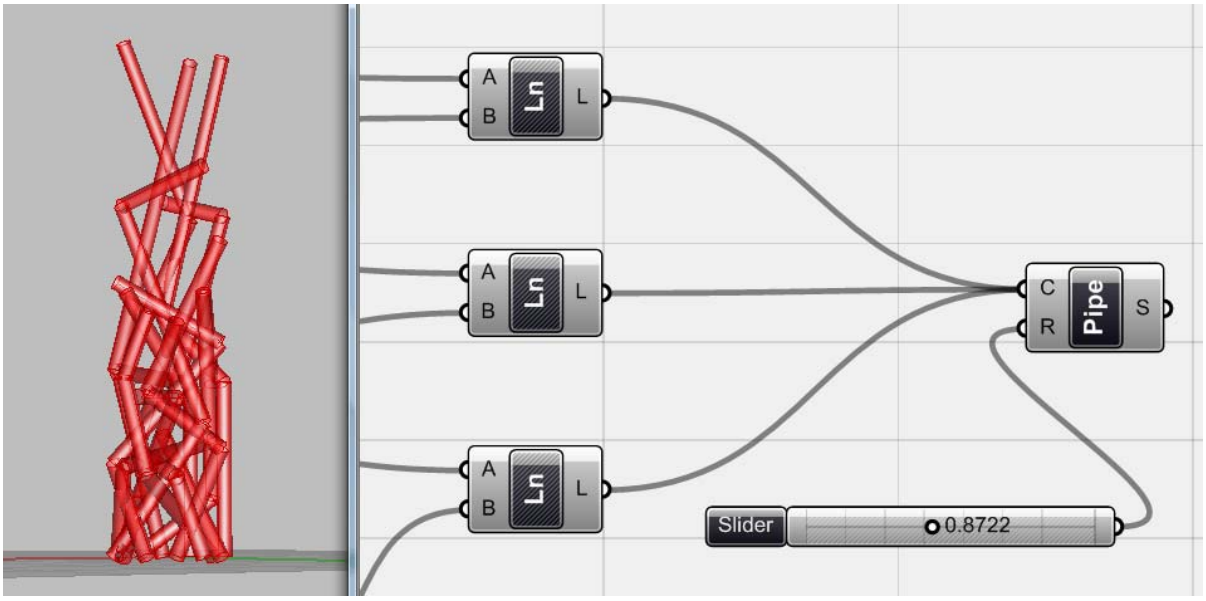


Fig.4.14. Final pipes. Uncheck the preview of lines. It speeds up your processing time.

That's it. Now you can change the radius of the base circle to distribute the pipes in larger/smaller areas, you can change the number of pipes (curves), you can change the random seed and pipe's radius. To do all and check the result you can go to the 'View' menu of the Grasshopper and select '**Remote Control Panel**' to have the control panel for your adjustments which is much more easier than the sliders inside the canvas when you want to observe the changes in Rhino scene. To hide/Unhide the canvas, just double-click on its window title bar.

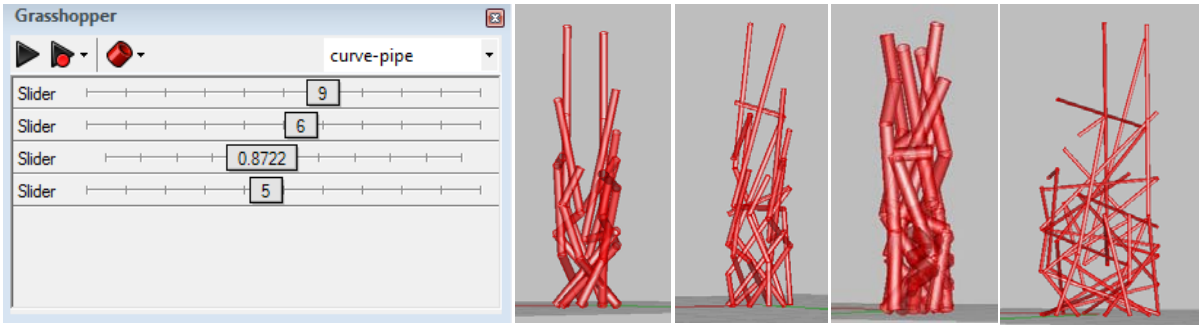


Fig.4.15. Remote Control Panel (from view menu) and observing the changes of the model.

Fig.4.16. Although the connections of pipes need a bit more elaboration, for an experiment it is enough.

4_3_Combined Experiment: Swiss Re

Today it is very common to design the concept of towers with this associative modelling method. It allows the designer to generate differentiated models simple and fast. There are so many potentials to vary the design product and find the best concepts quiet quickly. Here I decided to model a tower and I think the “Swiss Re” tower from ‘Foster and partners’ seems sophisticated enough to start, for some modelling experiments.

Let me tell you the concept. I am going to draw a simple plan of the tower and copy it to make the floors. Then I will rescale these floors to match the shape, and then I will make the skin of the surface and finally the façade’s structural elements. I will do the process with very simple geometries for this step and also to save time.

Let’s start with floors. I know that the Swiss Re’s floors are circles that have some V-shaped cuts around it, but I just use a simple circle to make the section of the tower. Since I know that it has 41 floors I will copy this section for 41 times In Z direction with 4 meters space in between and I will play around the proportions because I don’t know the real dimensions, but I will try to manage it visually.

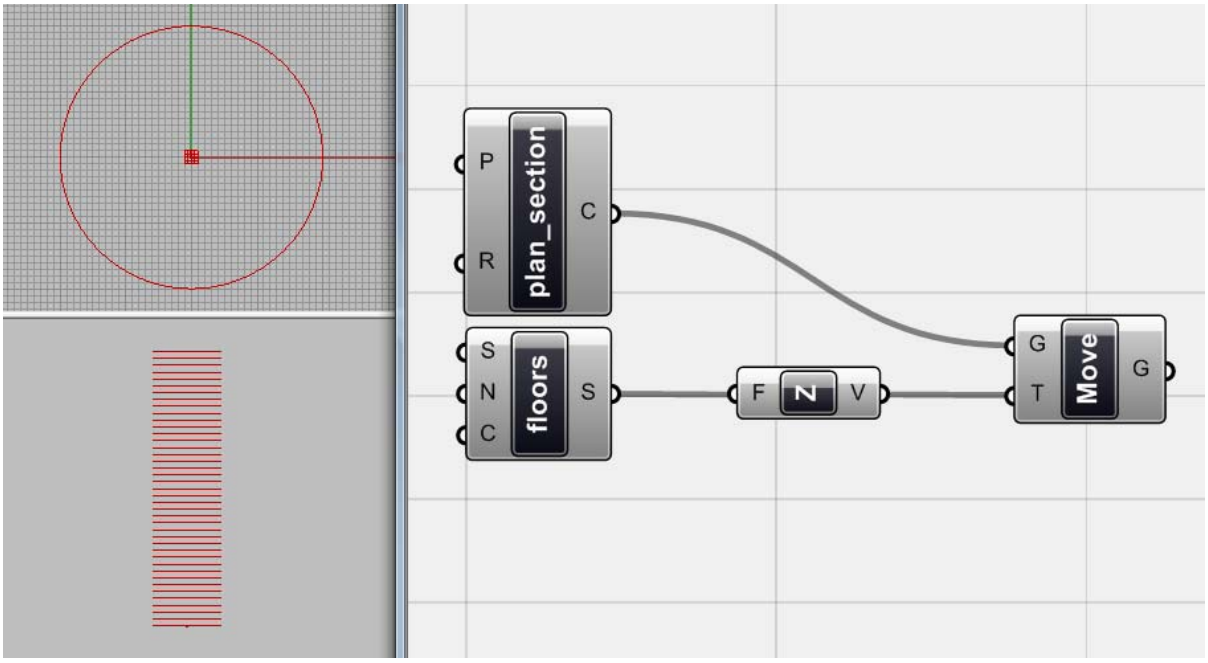


Fig.4.17. The <circle> component (plan_section) with radius of 20 which is copied by <move> component along Z direction by a <Z unit> vector component for 41 times above. To get this I used a <series> component (floors) starts from 0 and has the step size=4 with 41 values. (I renamed the components to recognize them easily).

Now the first thing we need is to rescale these circles to make the proper floors. I need a <scale> component (XForm > Affine > Scale) to rescale them. The <scale> component needs the geometry to scale, centre for scaling and the factor of scaling. So I need to feed the geometry part of it by our floors or circles which is <move> component.

The predefined centre of scaling is the origin point, but if we scale all our floors by the origin as centre, the height of the tower would rescale and the plane of each floor would change. So we need the centre of rescaling at the same level at the floor level and exactly at the centre of it. So I used a <Centre> component (Curve > Analysis > Centre) which gives me the centre of the circles. By connecting it to the <scale> you can see that all circles would rescale in their level without movement.

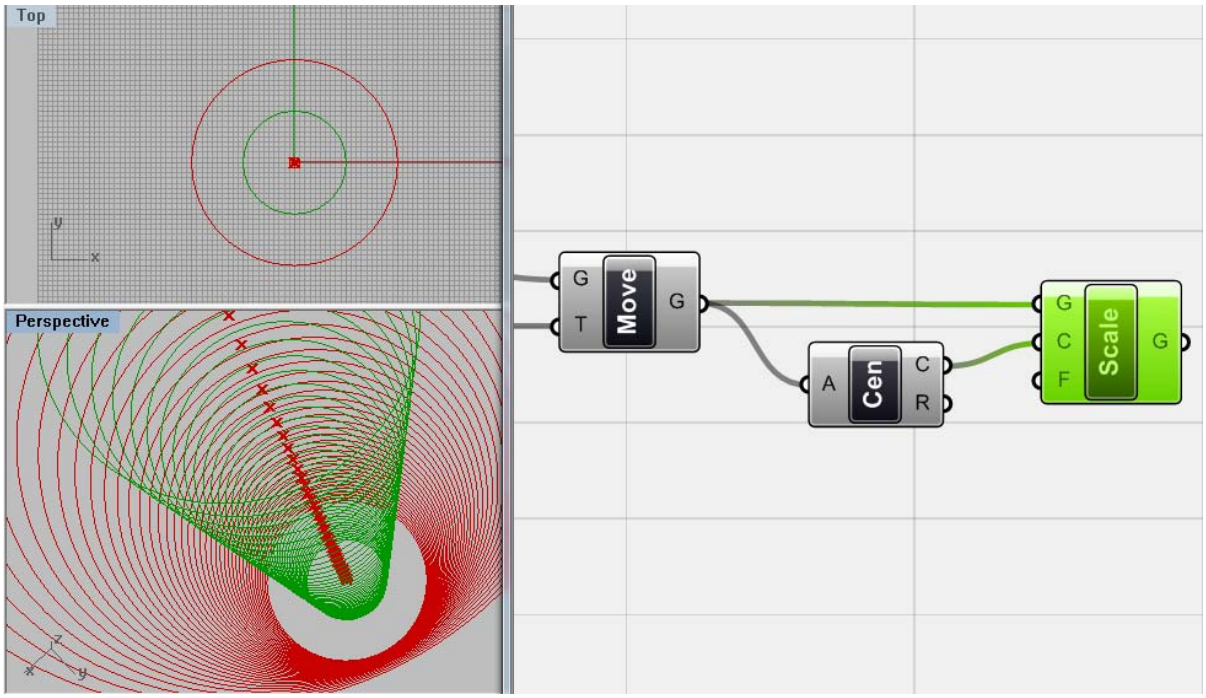


Fig.4.18. Rescaling floors from their centre point as the centre of scaling, using a <centre> component. The scaled circles are selected in green.

Although I rescaled the whole circles the same, but we know that all floors are not in the same size, so we need to rescale our floors different from each other; and we know that from the circle which is grounded on earth they first become bigger up to certain height, look constant on the middle parts and then become smaller and smaller up to the top point of the tower. So I need to provide a list of scale factors for all floors which are 41 and again I now that this list has three different parts that if we say starts from 1 then increases up to certain factor, then remain constant in some floors and then decreases. If you look at Figure 4.19 you cans see the pictures that give you a sense of these scaling factors. So basically I need to provide a list of data (41 values) which is not strait forward this time.

Fig.4.19. Swiss Re HQ, 30 St Mary Axe, London, UK, 1997-2004, (Photos from Foster and Partners website, http://www.fosterandpartners.com).

Scale Intervals

As I mentioned before, intervals are numeric ranges. They are real numbers from lower limit to upper limit. Since I said real numbers, it means we have infinite numbers in between. They are different types of usage for these mathematical domains. As we experimented before, we can divide a numerical interval by a certain number and get divisions as evenly distributed numbers between two numbers.

As I also mentioned that we have three different parts for the scaling factors of the tower, we need three different set of numbers, the first and the last ones are intervals and the middle part is just a real number which is constant. Let's have a look at Figure 4.20. Here I used two <interval> component (Scalar > Interval > Interval) to define two numerical range, one increasing and one decreasing. The increasing one starts from 1 which I assumed that the ground floor is constant and then increases up the number of the <number slider>. The second <interval> starts from <number slider> and ends at another <number slider> which is the lower limit. By using the same <number slider> for the middle part, I am sure that middle part of the data set is the same from both sides.

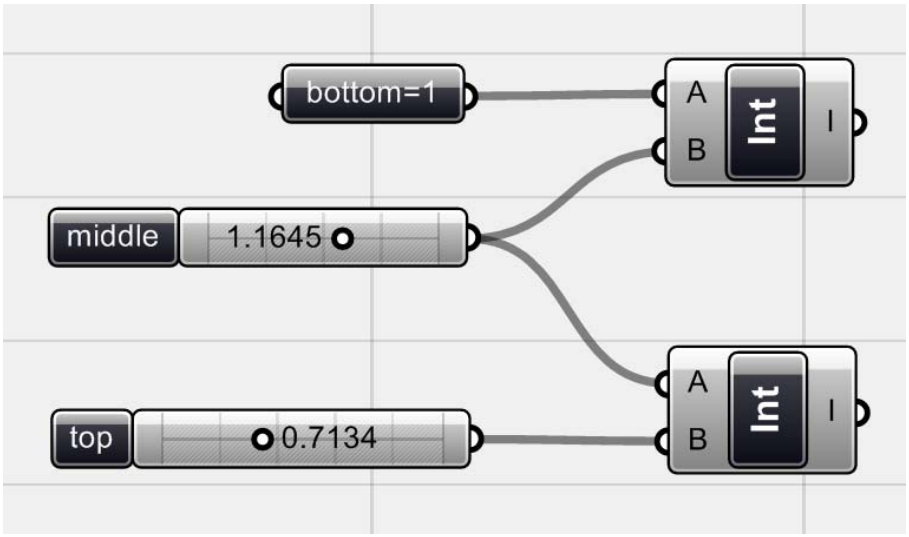


Fig.4.20. Two <interval> components, top one increasing and the bottom one decreasing. I do this because I don't know the exact proportions, but this help us to discuss about more interesting stuff!

Know I have the increasing and decreasing numeric intervals, but to produce the scaling factors I need numbers not ranges. So I am going to use <range> components to divide these numeric intervals up to certain numbers.

Because I don't know the projects data, still I do not know in which floor it starts to remain constant and where it decreases. So in order to assign these factors correctly, I am going to split the floors in two parts. I am using a <Split list> component (Logic > List > Split list) and I attach the <series> component which I named it floors to it to split the floors in two different parts. Then I need to know how many floors are there in each part that later on I can change it manually and correct it visually. <List length> component (Logic > List > List length) do this for me and passes the number of items in the list, so I know that how many floors are in the increasing scale part and how many in the decreasing scale part.

The only remaining part is the constant floors. I just assumed that there are 8 floors which their scale does not change and I need to omit them from the floors in the increasing and decreasing scaling part. Since two of these floors are included in increasing and decreasing lists as the maximum scale I need to omit another 6 floors. So I just need to get the number of the floors from <list length> and apply (-3) function to each, to omit these 6 floors and distribute them between both two lists.

And the final trick! Since the <range> component divides the domain to N parts, it produces N+1 number at the end which means 1 more number than we need. So all together I need to add a <function> component by (X-4) expression to reduce the number of steps that each <range> component wants to divide its numeric range.

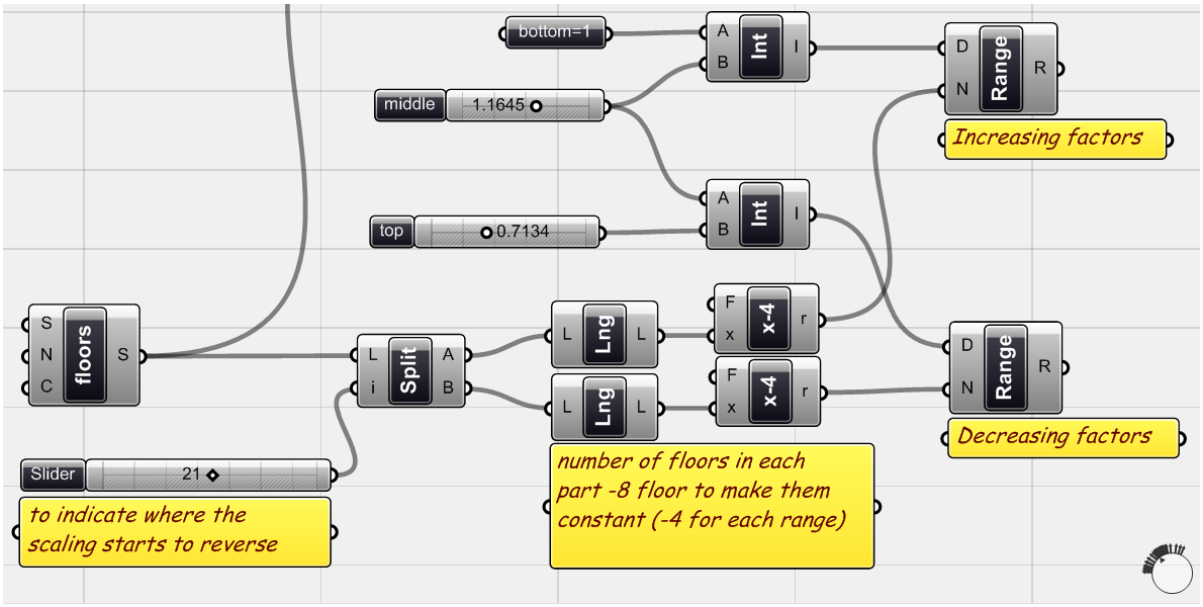


Fig.4.21. Generating the scale factors.

And finally as you can see in the Figure.4.22, I merged all my data by a <merge 8> component to make a unified list of data which is the numeric factors for the increasing parts, 6 constant number just coming from the < number slider> (the constant factor between the two range) and the decreasing factor. The <merge 8> component includes 41 scaling factor that now I can attach to the <scale> component and rescale all floors.

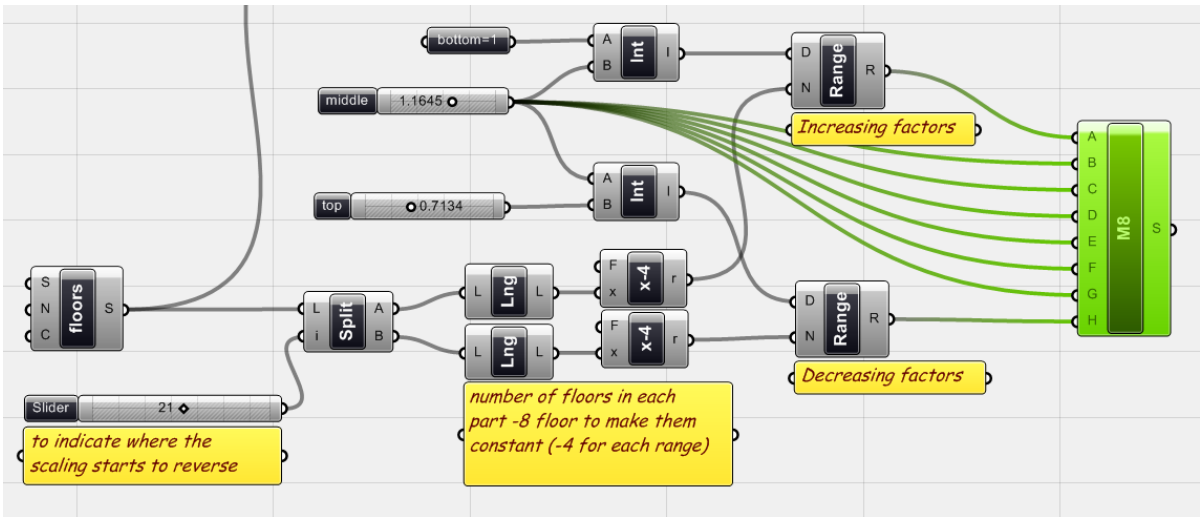


Fig.4.22. The <merge 8> component includes <range> of increasing numbers, 6 constant numbers (which is the maximum number of the increasing and decreasing parts), and the <range> of decreasing numbers as one unified list of data.

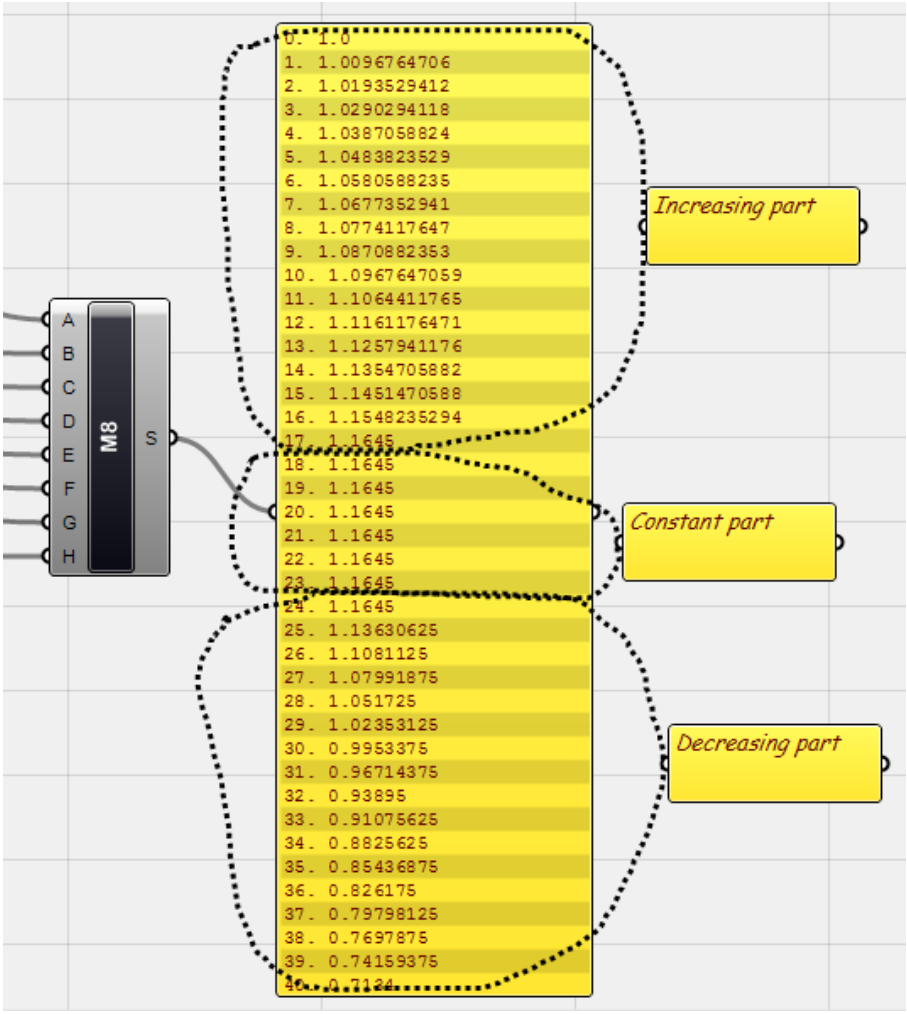


Fig.4.23. Scaling factors

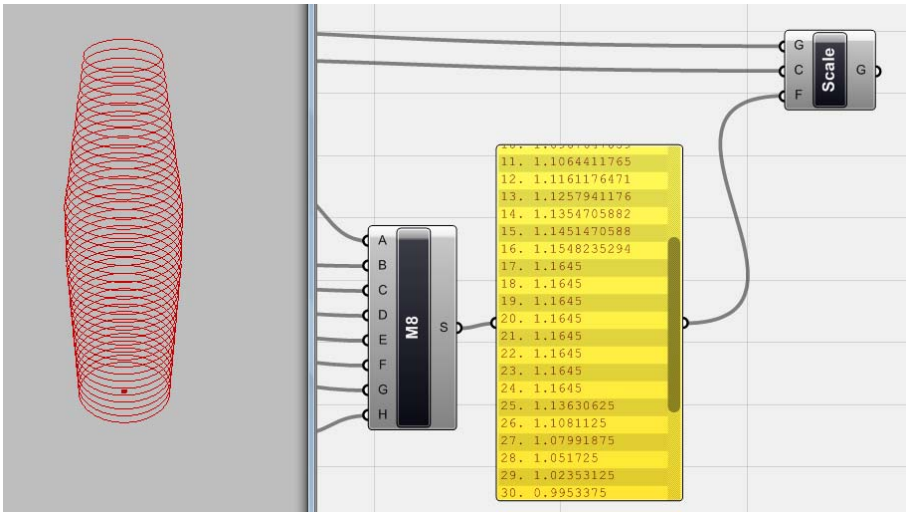


Fig.4.24. Rescaled floors. Now I can change the position of the constant floors and the scaling factors to visually match the model by the original building.

Ok! Let's go for façade elements.

The steel elements around the façade are helical shapes that have the cross section like two connected triangles but again to make it simple, I just make the visible part of it which is almost like a triangle (in section). I want to generate these sections and then 'loft' them to make a surface.

I started with a <polygon> component (Curve > Primitive > Polygon). I used an <end points> component to get the start/end points of my floors. By attaching these points as the base for <polygon> I can generate couple of polygons on the start points of my floors. I attached a <number slider> to the <polygon> to control its radius and I set the number of segments to 3 manually. I renamed the <scale> component to the <rescaled_floors>.

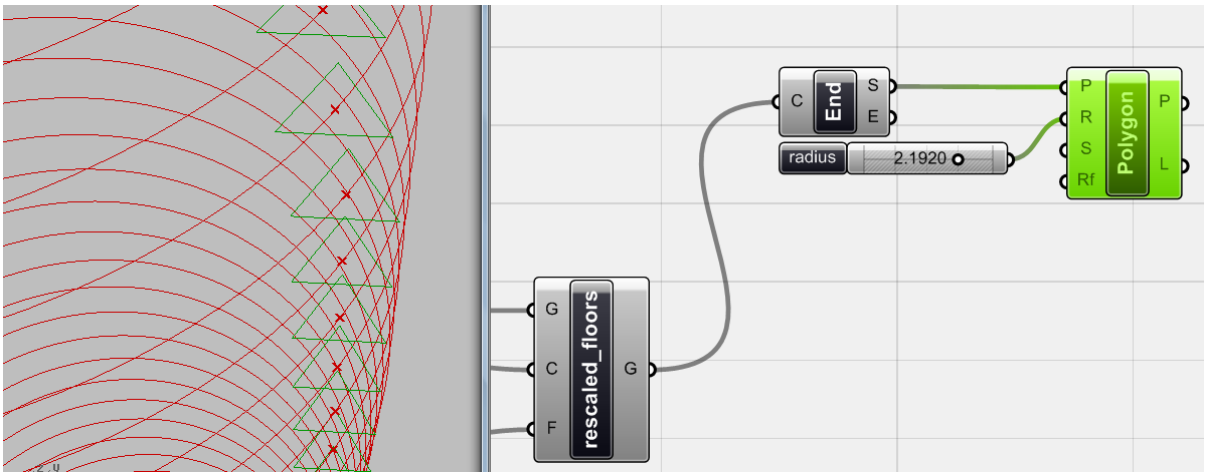


Fig.4.25. Polygons positioned on the facade.

Now I want to make the helical transformation for the polygons. For some reasons I think that every floors rotate for 5 degree! So I need to rotate all polygons for 5 degree around the centre of the floors. So I brought a <rotate> component to the canvas (XForm > Euclidian > Rotate). Geometry is my <polygon> and the base of rotation is the centre of the floors / <centre> of circles. To produce the rotation angle I need a list of incremental factors that each time adds 5 degree. So I used a <series> starts from 0 with the step size of 5 and with 41 values which come from the floors and number of <polygon> also. The only thing remain is because <rotate> component works with Radian I need to convert Degree to Radian by a <function> which is Radian=Degree * Pi / 180 (There is a predefined function called RAD(x) that converts degree to radian also. Check the functions library).

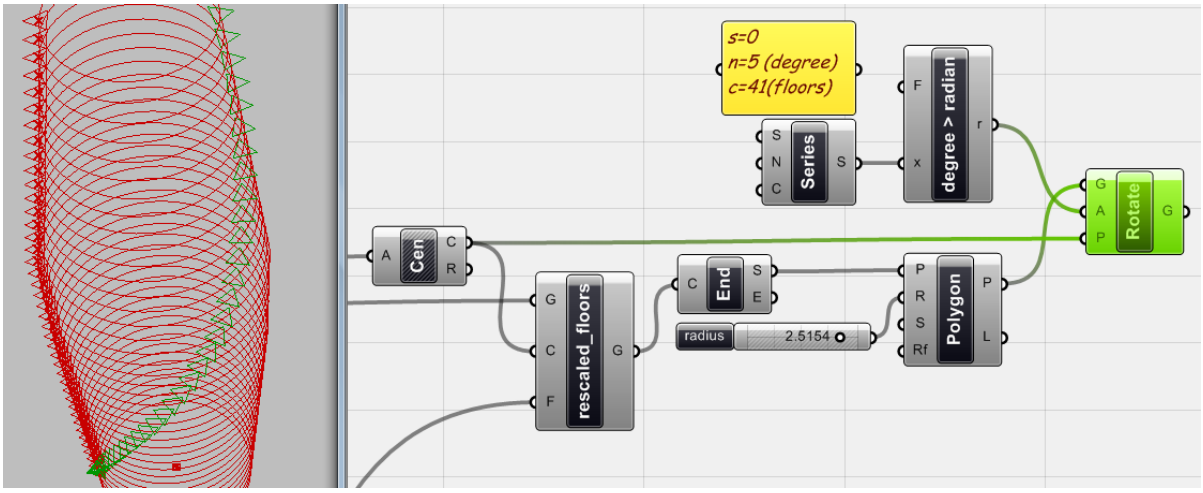


Fig.4.26. Rotating the polygons around the centre of floors, each for 5 degree from the previous one.

Now just use a <loft> component to make a surface by connecting all these sections together. To make the scene clean, uncheck the preview of any unnecessary geometry.

As you can see, we don't have the top point of the tower because we don't have any floor there. We know that the height of tower is 180m. So I added a simple <point> component (0, 0, 180) to the canvas and I attached it to the <polygon> component (by holding shift). So the <polygon> component produces another polygon at the top point of the tower and the number of polygons becomes 42. So I changed the <series> component to produce 42 numbers as well. I know that the top part of the "Swiss Re" is more elegant than this model but for our purpose this is fine.

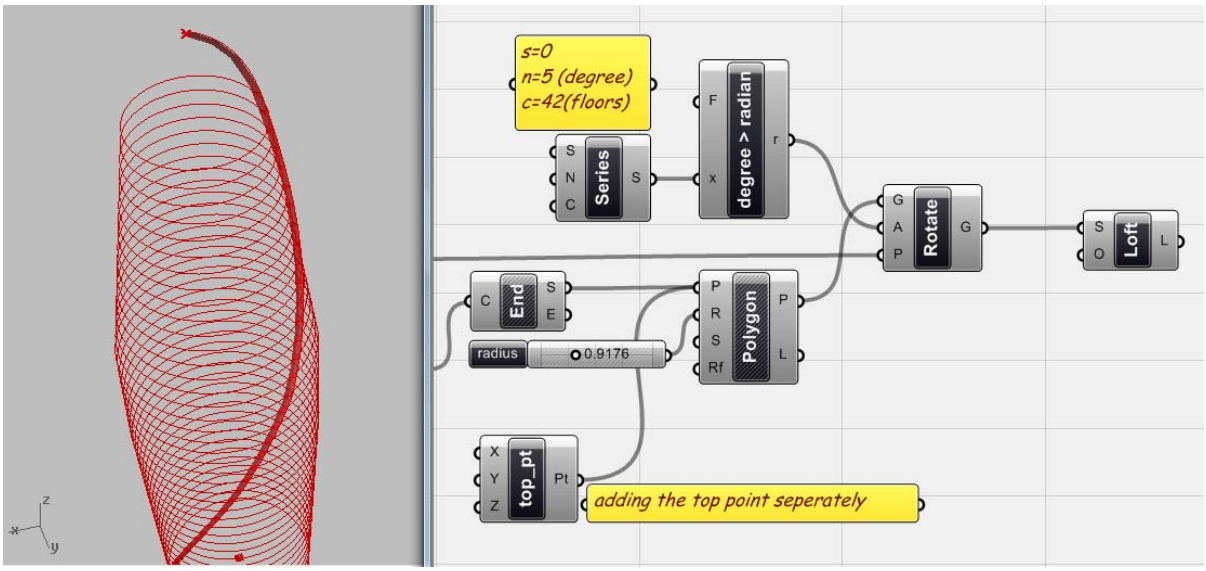


Fig.4.27. one of the façade elements made by the <loft> component.

To generate these elements all around the building, I am using a <rotate> component which I attached the <loft> object as source geometry and I used a <range> from 0 to 2Pi divided by a <number slider> as the number of elements, to rotate it all around the circle. Since the centre of rotation is the Z axis at the centre of the tower, and it is pre-defined on the component, I do not need to change it or introduce any plane for rotation.

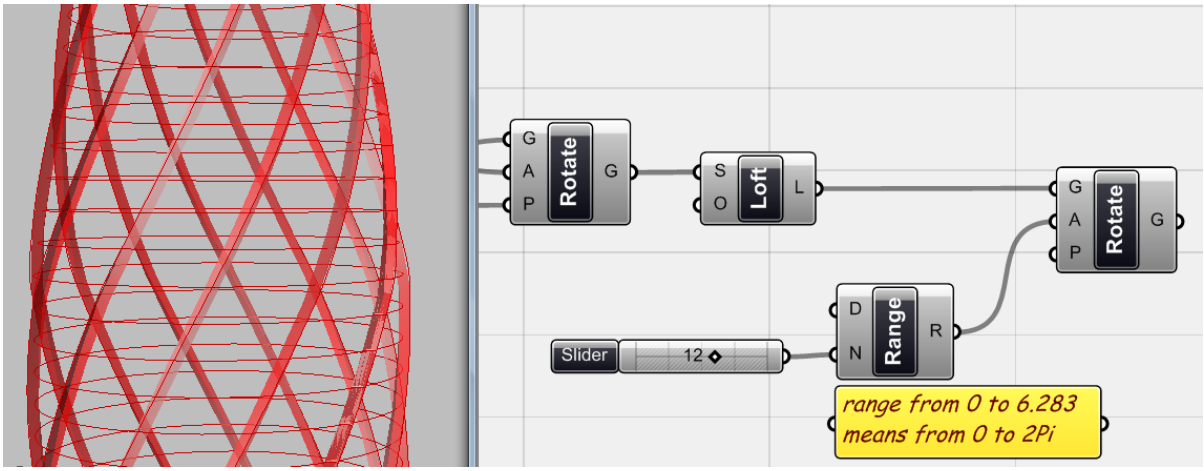


Fig.4.28. first set of spiral elements around the tower.

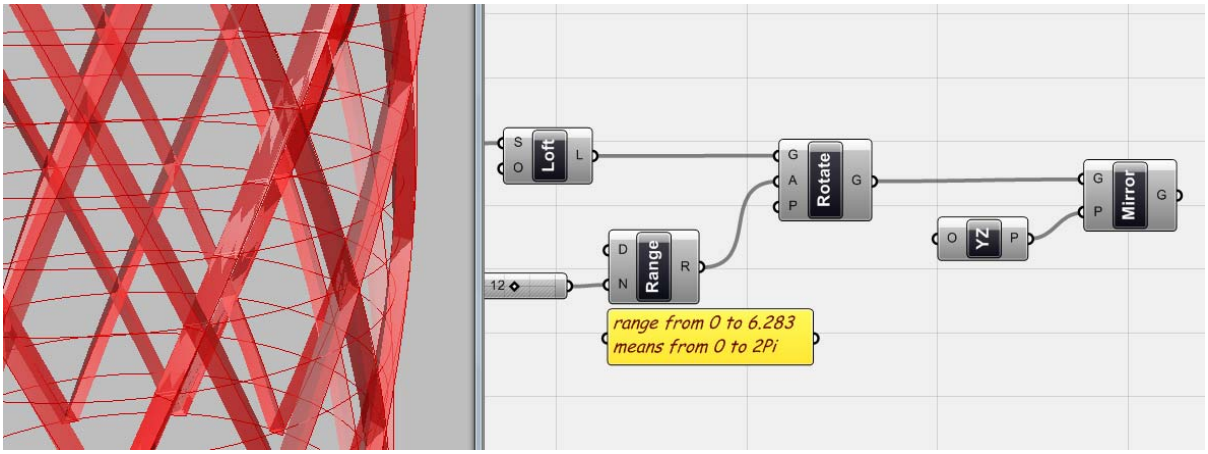


Fig.4.29. I need to <mirror> (XForm > Euclidian > Mirror) the rotated geometry by <YZ plane> (Vector > Constants > YZ plane) to have the lofted elements in a mirrored helical shape. So at the end I have a lattice shape geometry around the tower.

Glass cover

To cover the whole tower simply with glass, I should go back to the floors component and <loft> them. Again since we don't have any floor at the top point of the tower, I used another <circle> component and I feed it by the top point as the position and I attached it to the <loft> object to make the loft surface complete up to top point.

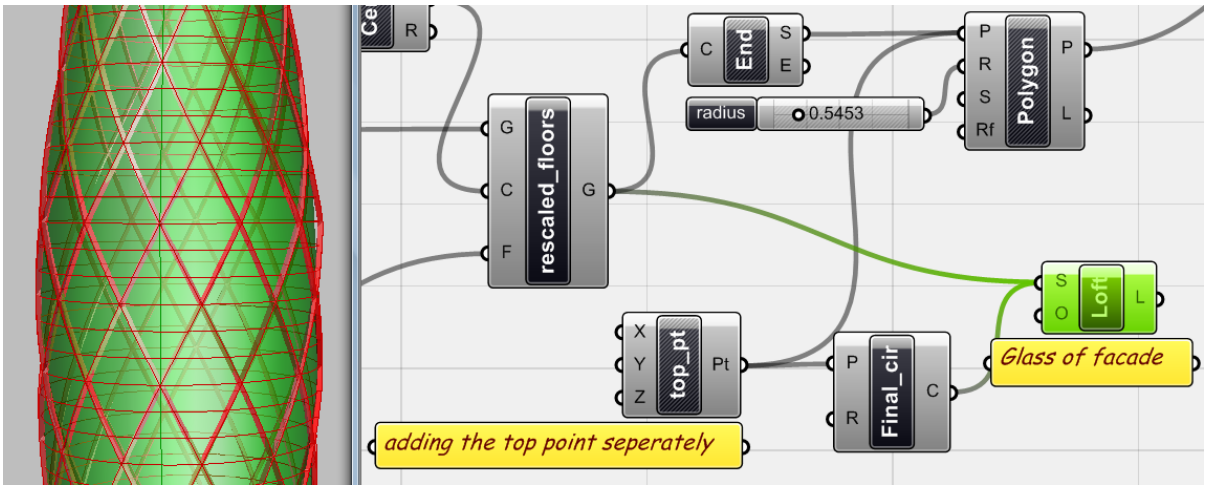


Fig.4.30.a. lofting floor curves to make the façade's glass cover.

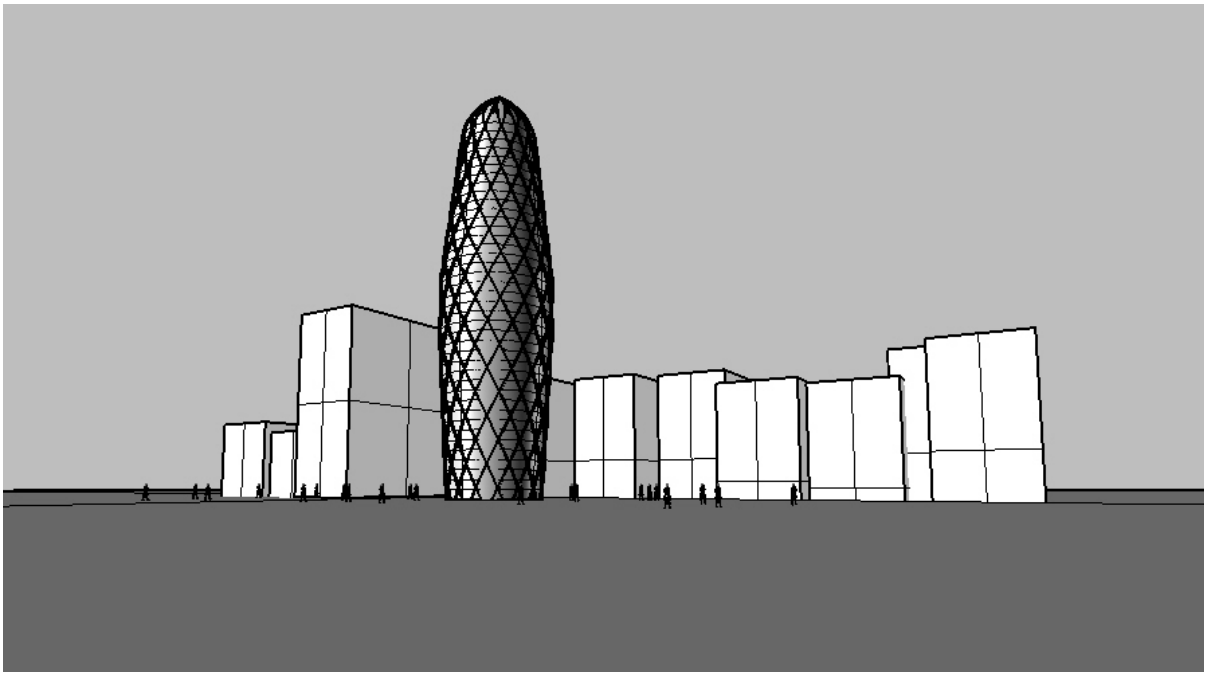
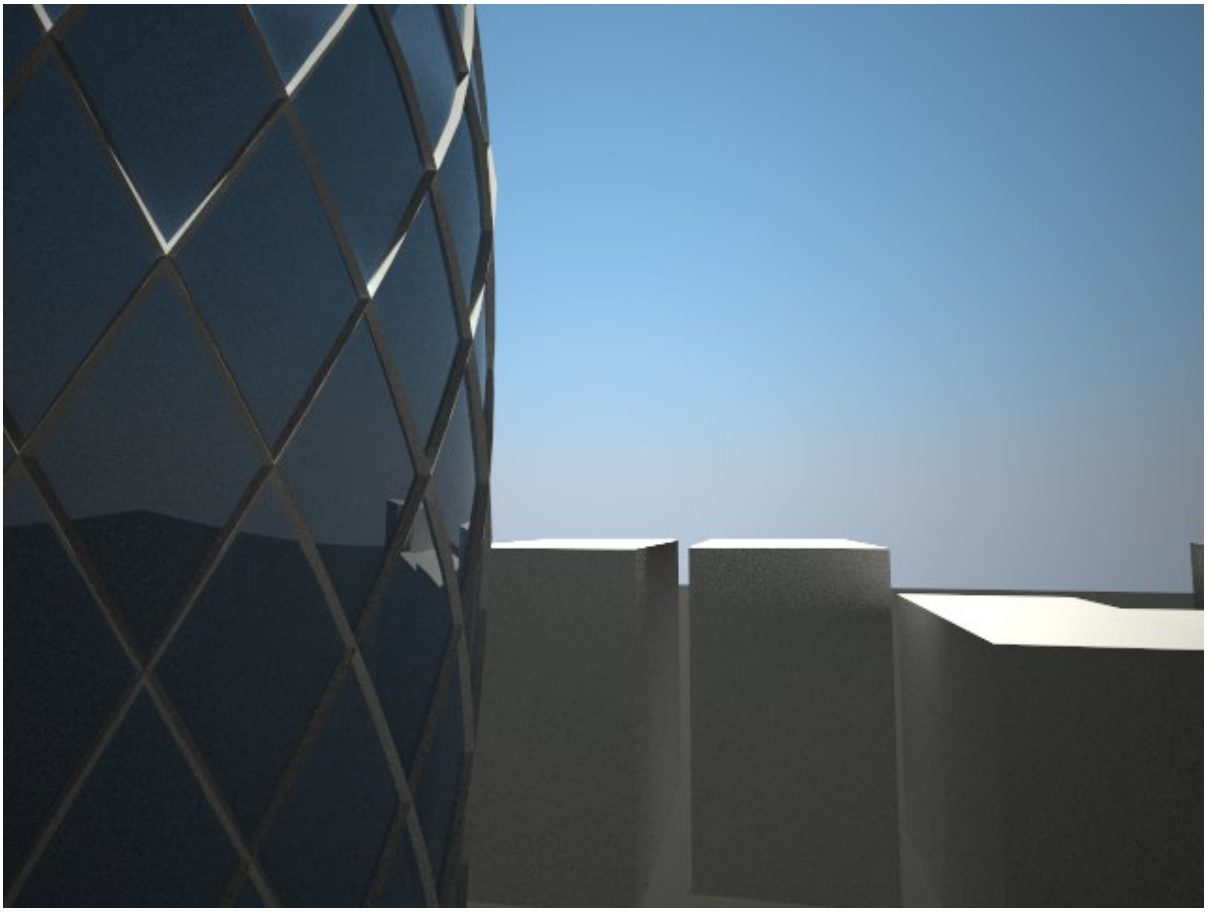


Fig.4.30.b. The lofted surface covers the whole façade. In the Swees Re project, there is two colours of glass. If once we decided to make this effect, we can use façade structure to produce different surfaces and render them differently.



4.31. Final model. Although it is not exactly the same, but for a sketch model in a short time, it would work.

4_4_On Attractors

"Attractor is a set of states of a dynamic physical system toward 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 center of the bowl; the final state of position and motionlessness is a point attractor."

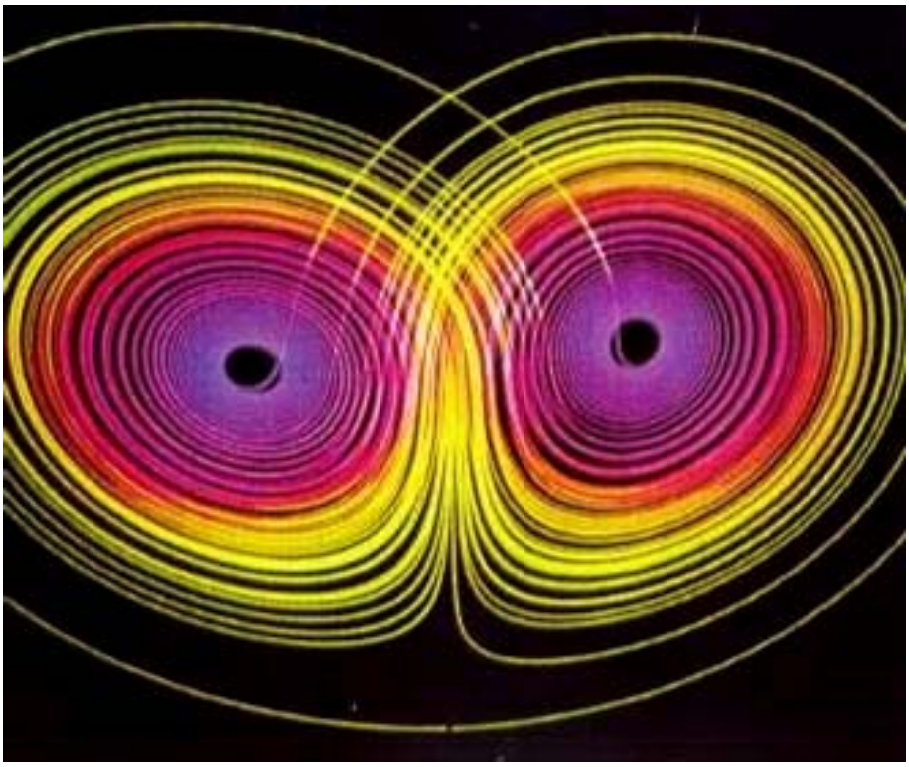


Fig.4.32. Strange Attractor.

In the case of design and geometry, attractors are elements (usually points but could be curves or any other geometry) that affect the other geometries in the space, change their behaviour and make them 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 since they have the potential to change the whole objects of design constantly. Defining a field, attractors could also affect the multiple agent systems in multiple actions. The way they could affect the product and the power of attractors are all adjustable. We go through the concept of attractors in different occasions so let's have some very simple experiments first.

Point Attractors

I have a grid of points that I want to generate a set of polygons on them. I also have a point that I named it <attractor_1> and I draw a <circle> around it just to realize it better. I want this <attractor_1> affects all my <polygon>s on its field of action. It means that based on the distance between each <polygon> and the <atractor_1>, and in the domain of the <attractor_1>, each <polygon> respond to the attractor by change in its size.

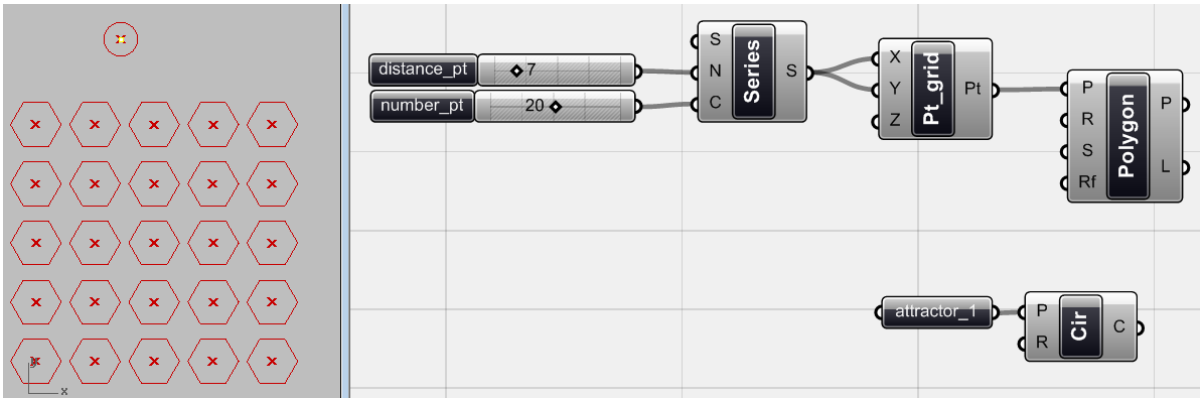


Fig.4.33. Base <point_grid> and the <polygon>s and the <attractor_1>.

The algorithm is so simple. Based on the <distance> between <attractor_1> and the <Pt-grid>, I want to affect the radius of the <polygon>, so the 'relation' between attractor and the polygons define by their distance. I need a <distance> component to measure the distance between <attractor_1> and the polygon's center or <pt_grid>. Because this number might become too big, I need to <divide> (Scalar > Operators > Division) this distance by a given number from <number slider> to reduce the power of the <attractor_1> as much as I want.

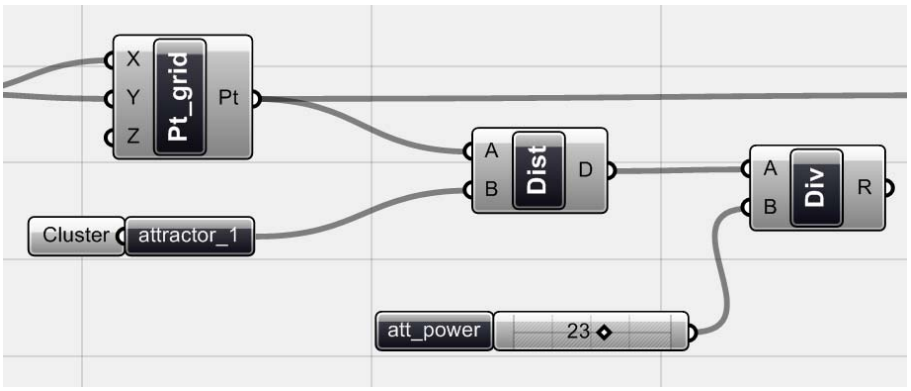


Fig.4.34. <Distance> divided by a number to control the 'power' of the <attractor_1>. I also made a Cluster by <attractor_1> and its <circle> to have one component as attractor in the canvas. You can convert any group of related geometries to clusters by selecting them and using 'make cluster from selection' from the canvas toolbar (or Arrange menu or Ctrl+G).

Now if you connect this component to the Radius (R) part of the <polygon> you can see that the radius of polygons increases when they go farther from the <attractor_1>. Although this could be good for the first time, we need to control the maximum radius of the polygons, otherwise if they go farther and farther, they become too big, intersecting each other densely (it also happens if the power of the attractor is too high). So I control the maximum radius value of the polygons manually.

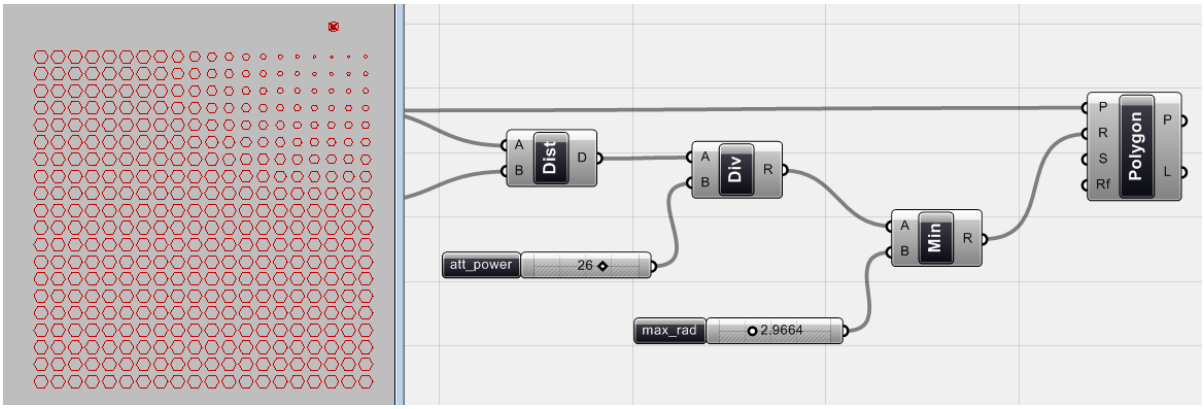


Fig.4.35. By using a <minimum> component (Scalar > Util > Minimum) and a user defined number, I am telling the algorithm to choose the value from the component, if it is smaller than the number that I defined as a maximum radius by <number slider>. As you can see in the pictures, those polygons that are in the power field of attractor being affected and others are constant.

Now if you change the position of the <attractor_1> in the Rhino workplace manually, you can see that all polygons get their radius according to the <attractor_1> position.

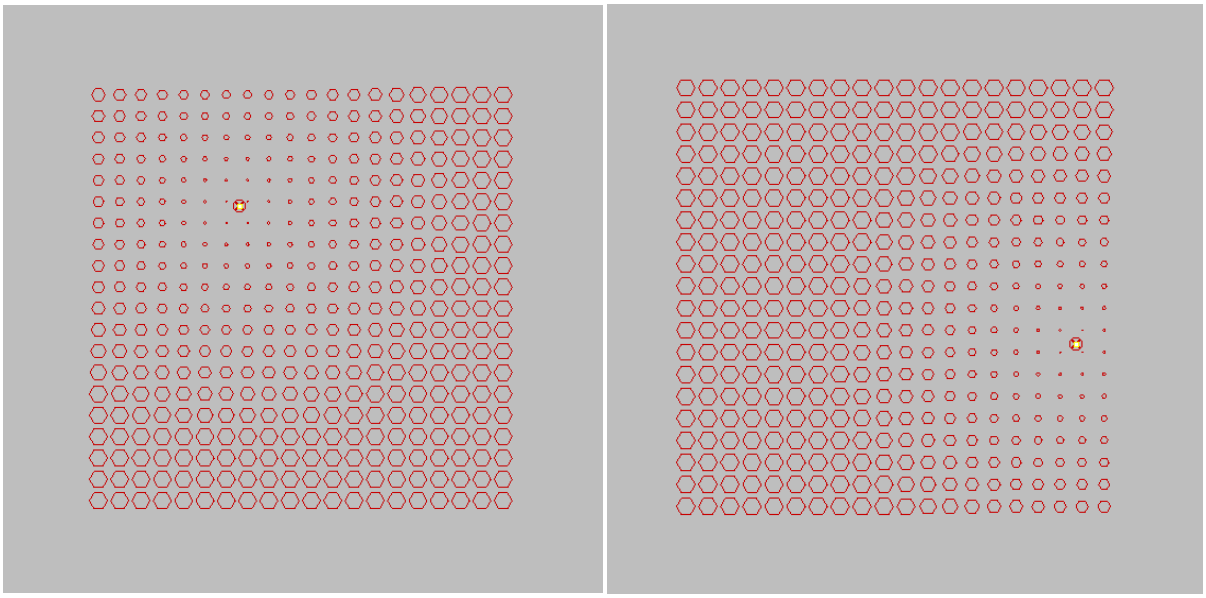


Fig.4.36. The effect of the <attractor_1> on all polygons. Displacement of the attractor, affects all polygons accordingly.

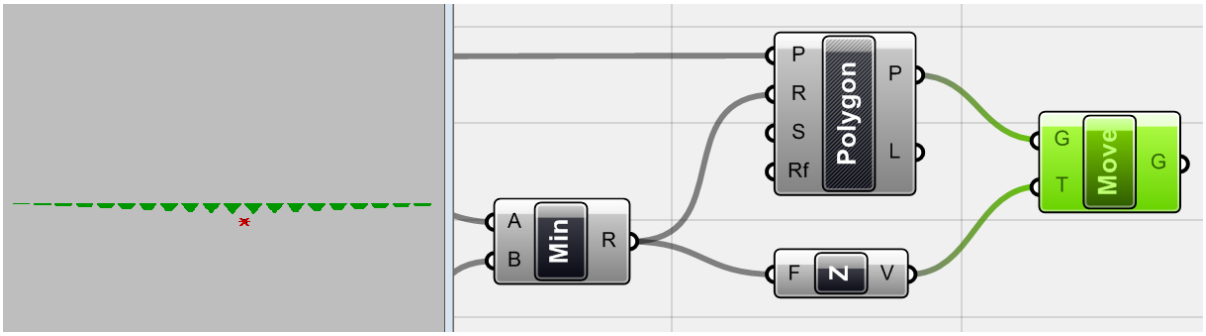


Fig.4.37. Wiyh the same concept, I can displace polygons in Z direction based on the numbers coming from the <Min> component or changing it by mathematical functions, if necessary.

Simple. I can do any other function on these polygons like rotate, change colour, etc. But let's think what would happen if I would have two attractors in the field. I made another cluster which means another point in Rhino associated with a <point> and <circle> in Grasshopper.

It seems that the first part of the algorithm is the same. Again I need to measure the distance between this <attractor_2> and the polygons' center or <pt_grid> and then find the <min> of these distances and the previously defined maximum number for the radius.

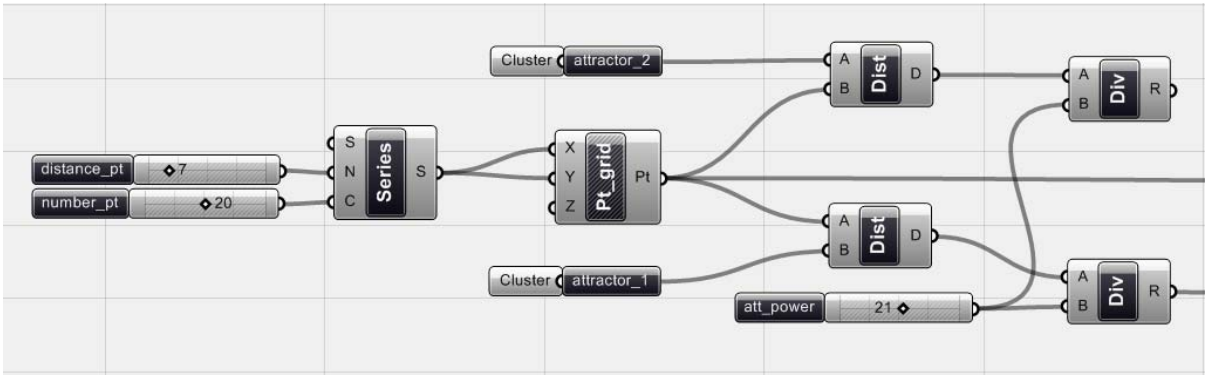
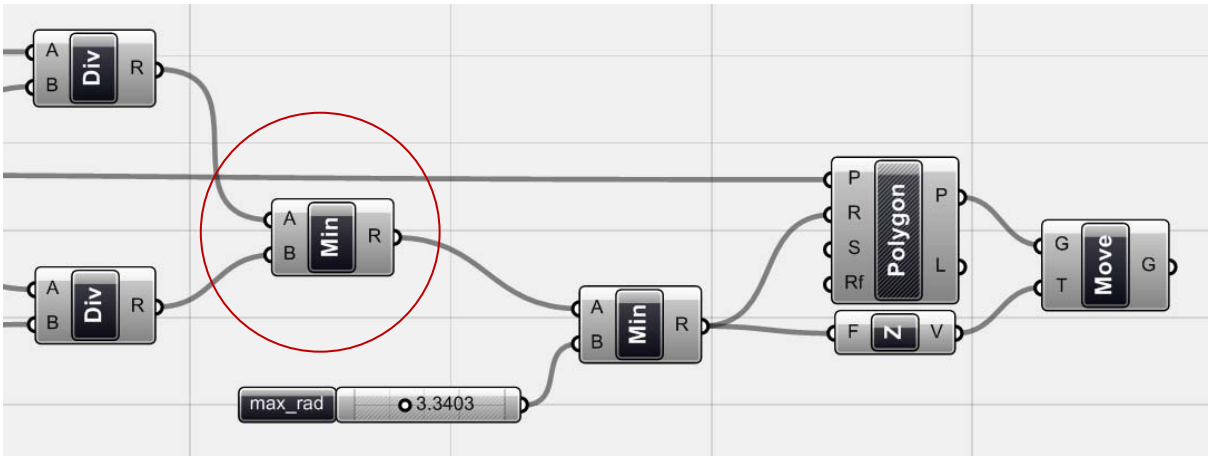


Fig.4.38. Introducing second <attractor_2> to and applying the same algorithm to it.

Now we have two different data lists that include the distance from the polygon to each attractor. Since the closer attractor would affect the polygon more, I should find one which is closer, and use that one as the source of action. So I will use a <min> component to find which distance is minimum or which point is closer.



4.39. Finding the closer attractor. After finding the closer one by <min> component, the rest of the process would be the same. Now all <polygon>s are being affected by to attractors.

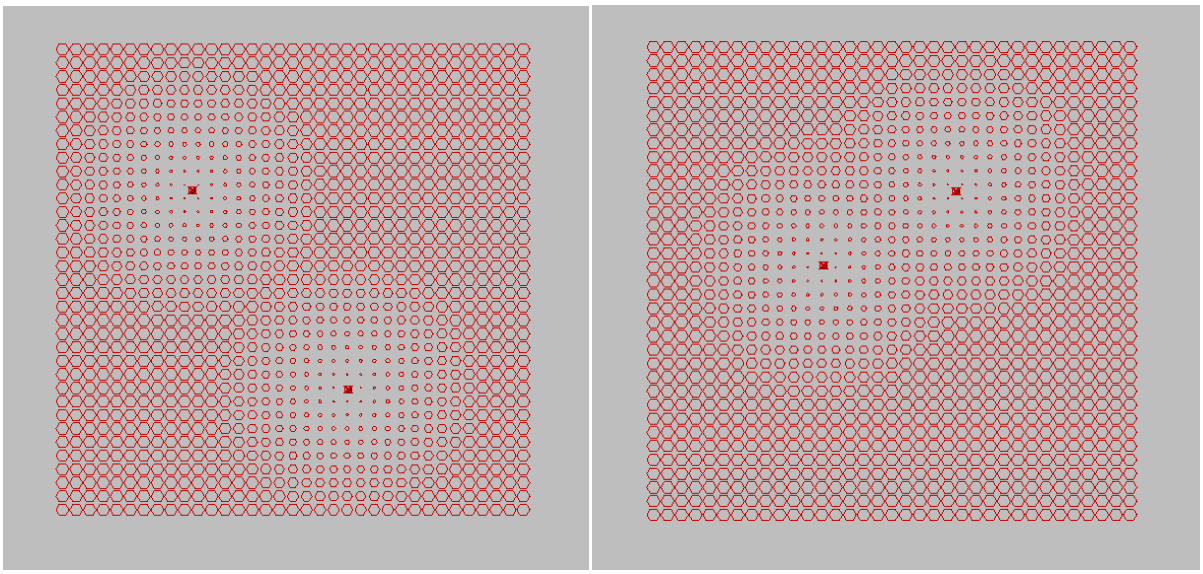


Fig.4.40. Again you can change the position of the attractors and see how all polygons reacting accordingly.

We can have more and more attractors. The concept is to find the attractor which is closer for each polygon and apply the effect by selecting that one. Selection in terms of distance happens with <min> functions, but we will talk about other types of selection later.

There are other ways of dealing with attractors like using <cull> component. In this method you need to provide different lists of data from the distance between points and attractors and then culls those far, select the closer one by simple Boolean function of a>b. since there are multiple examples on this topic on-line, I hope you will do them by yourself.

Curve Attractors: Wall project

Let's complete this discussion with another example but this time by Curve attractors because in so many cases you need to articulate your field of objects with linear attractors instead of points.

My aim here is to design a porous wall for an interior space to let me have a multiple framed view to the other side. This piece of work could be cut from sheet material. In my design space, I have a plane sheet (wall), two curves and bunch of randomly distributed points as base points of cutting shapes. I decided to generate some rectangles by these points, cutting them out of the sheet, to make this porous wall. I also want to organize my rectangles by this two given curve so at the end, my rectangles are not just some scattered rectangles, but randomly distributed in accordance to these curves which have a level of organisation in macro scale and controlled randomness in micro scale.

What I need is to generate this bunch of random points and displace them toward the curves based on the amount of power that they receive from these lines. I also decided to displace the points toward both curves so I do not need to select closer one, but I displace the points based on their distance to the curve. Then I want to generate my rectangles over these points and finally I will define the size of these rectangles in relation to their distance to the attractors.

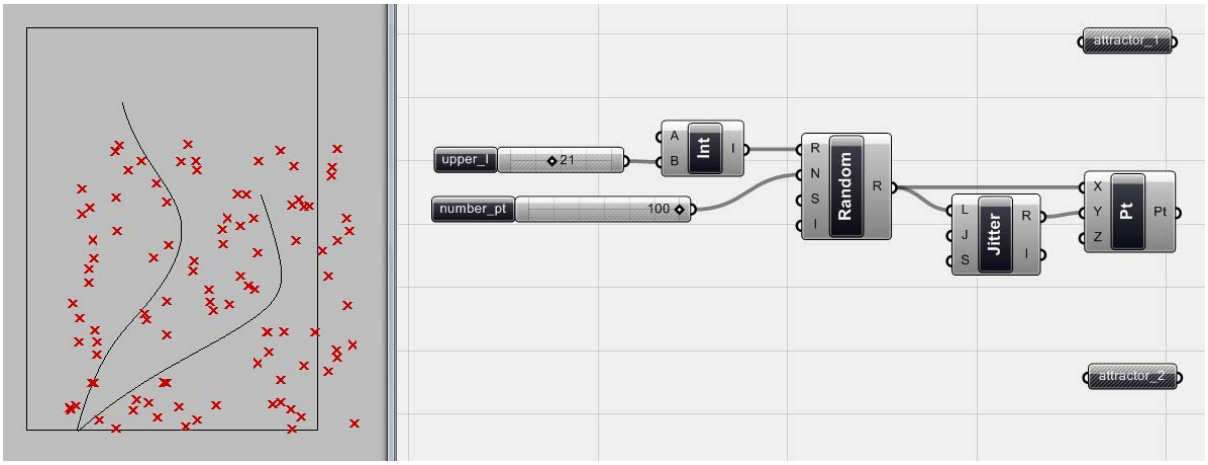


Fig.4.41. Generating a list of randomly distributed <point>s and introducing the attractors by two <curve> component (Params > Geometry > Curve) over a sheet. I used an <interval> component to define the numeric interval between 0 and <number slider> for the range of random points. I will make a cluster by <interval>, <random>, <jitter> and <point> to make the canvas more manageable.

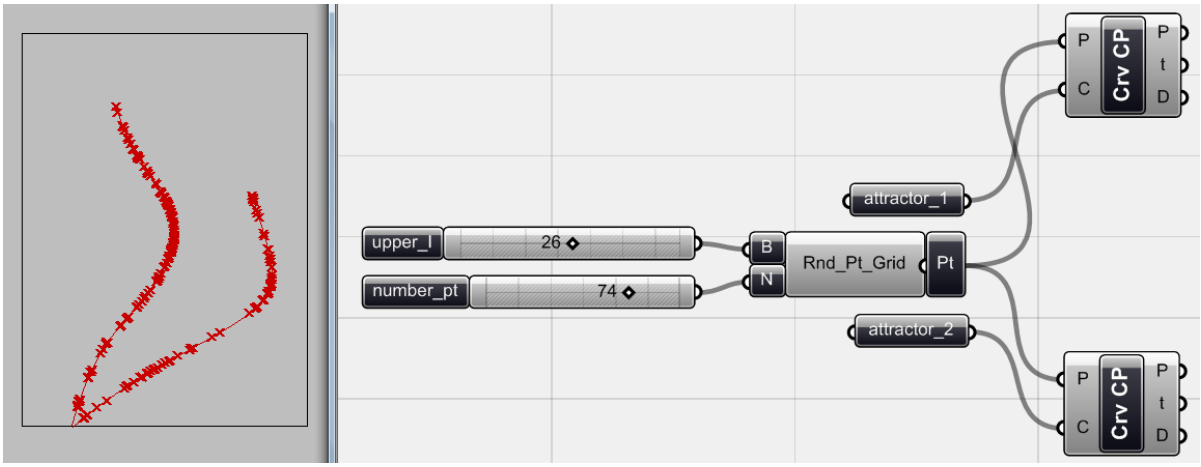


Fig.4.42. When the attractor is a point, you can simply displace your geometry towards it. But when the attractor is a curve, you need to find a relative point on curve and displace your geometry towards that specific point. And this point must be unique for each geometry, because there should be a one to one relation between attractor and any geometry in the field. If we imagine an attractor like a magnet, it should pull the geometry from its closest point to the object. So basically what I first need is to find the closest point of <Rnd_pt_grid> on both attractors. These points are the closest points on the attractors for each member of the <Rnd_Pt_Grid> separately. I used <Curve CP> component (Curve > Analysis > Curve CP) which gives me the closest point of the curve to my <Rnd_Pt_Grid>.

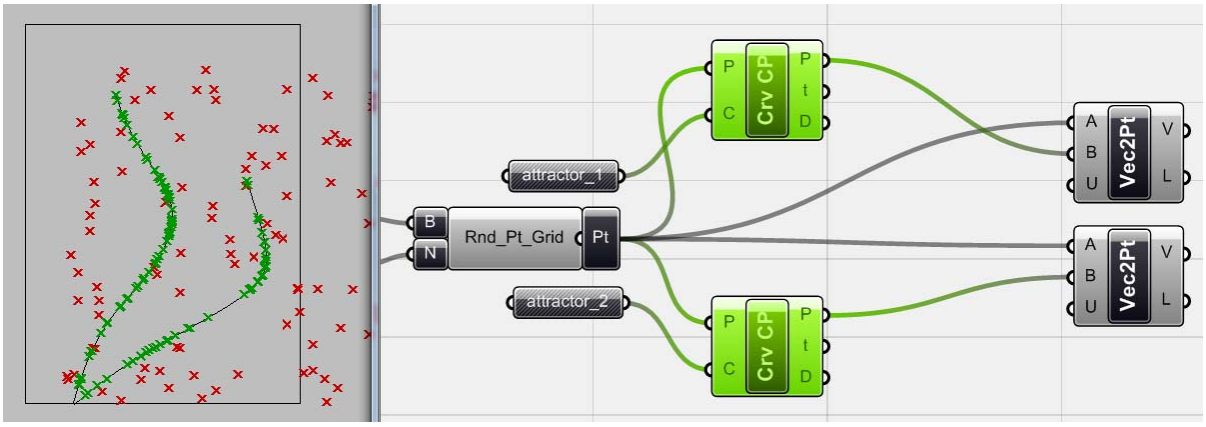


Fig.4.43. In order to displace the points towards the attractors, I need to define a vector for each point in <Rnd_Pt_Grid>, from the point to its closest point on the attractors. Since I have the start and end point of the vector I am using a <vector 2Pt> component to do that. The second point of the vector (B port of the component) is the closest point on the curve.

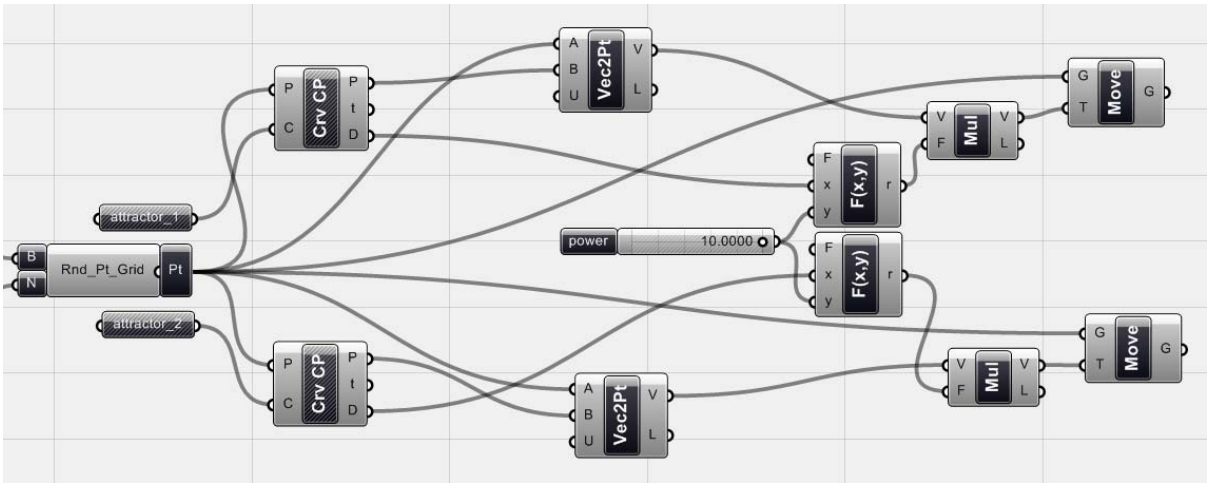


Fig.4.44. Now I connected all my <Rnd_Pt_Grid> to two <move> components to displace them towards the attractors. But if I use the vector which I created in the last step, it displaces all points onto the curve and that's not what I want. I want to displace the points in relation to their distance to the attractor curves. If you look at the <Curve CP> component it has an output which gives us the distance between the point and the relevant closest point on the curve. Good. We do not need to measure the distance by another component. I just used a <Function 2> component and I attached the distance as X and a <number slider> to Y to divide the X/Log(Y) to control the factor of displacement (Log function change the linear relation between distance and the resulting factor for displacement).

I just used a <multiply> component (Vector > Vector > Multiply), I attached the <vector 2P>as base vector and I changed its size by the factor I created by distance, and I attached the resulting vector to the <move> components which displaces the <Rnd_Pt_Grid> in relation to their distance to the attractors, and towards them.

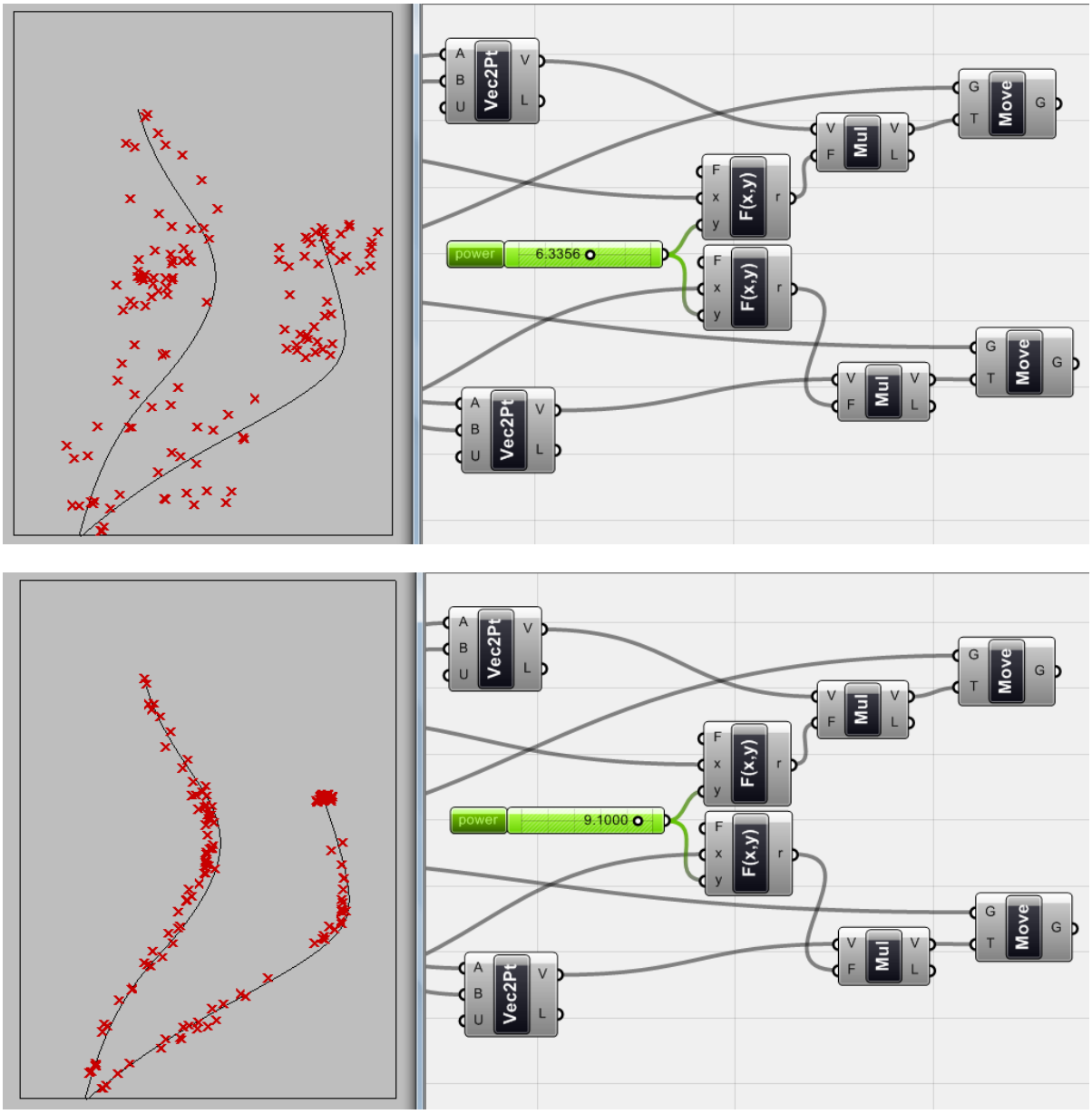


Fig.4.45. The <number slider> changes the power with which attractors displace objects towards themselves.

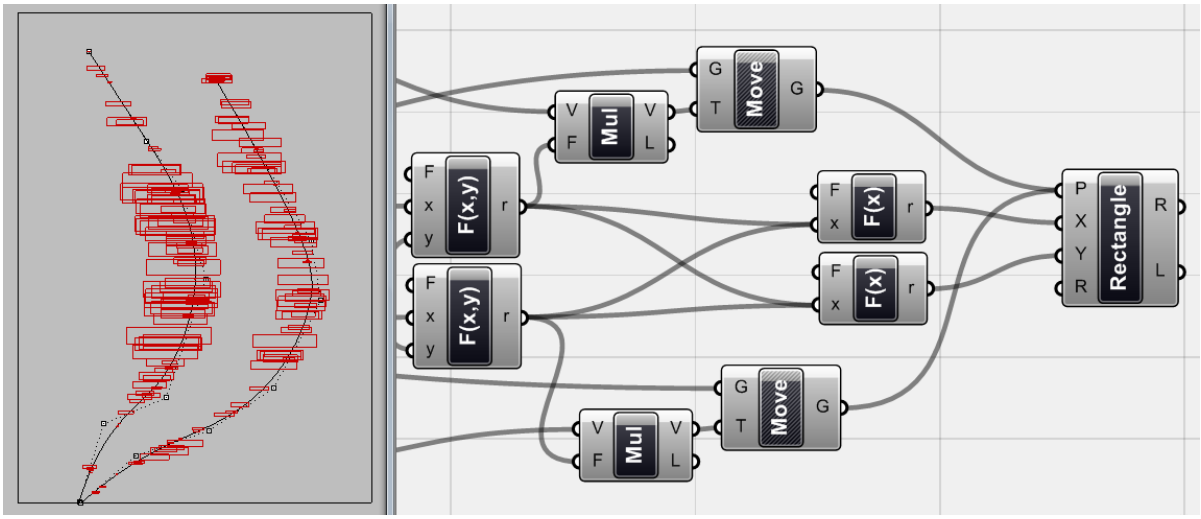


Fig.4.46. I used a <rectangle> component and I attached the <move>d or displaced points to it as the base point (planes) for my rectangle components. But as I told you, I want to change the size of the <rectangle>s based on their distances to each <attractor>. So I used the same numerical values which I used for vector magnitude and I changed them by two functions. I divided this value by 5 for the X value of the rectangles and I divided by 25 for the Y value. As you can see, rectangles have different dimensions based on their original distance from the attractor.

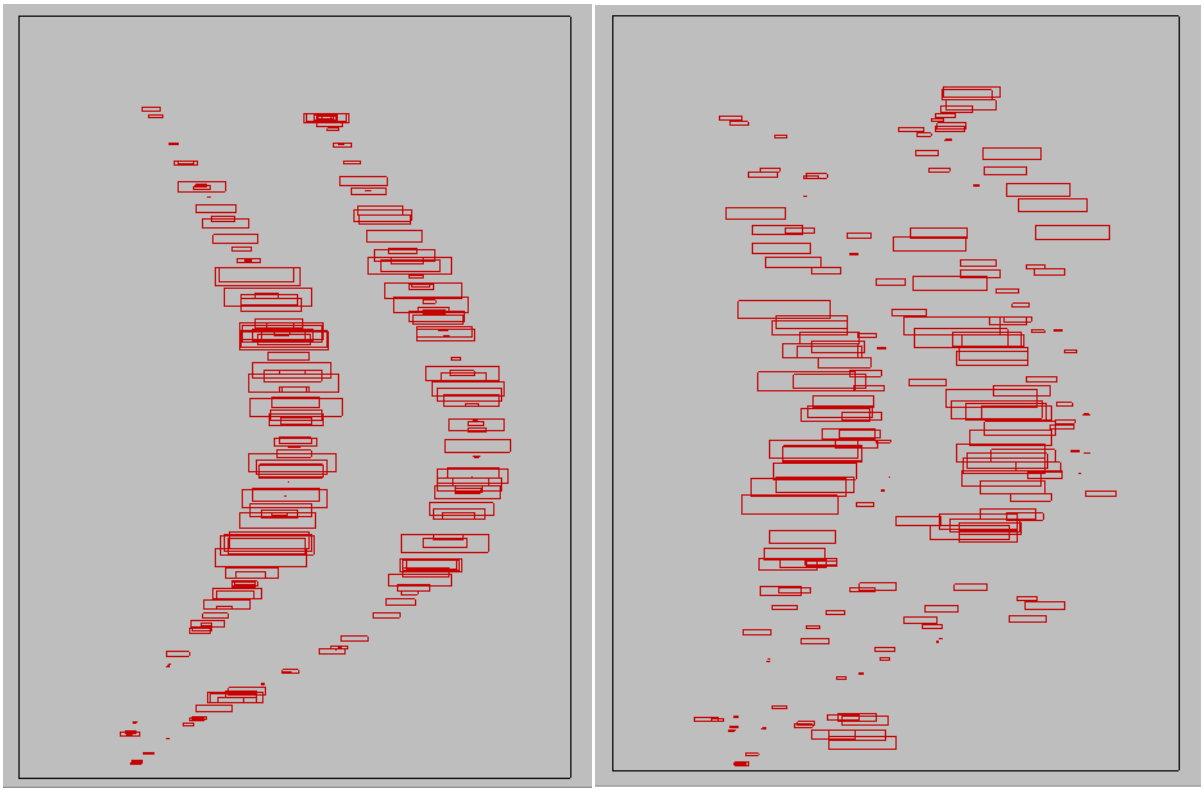
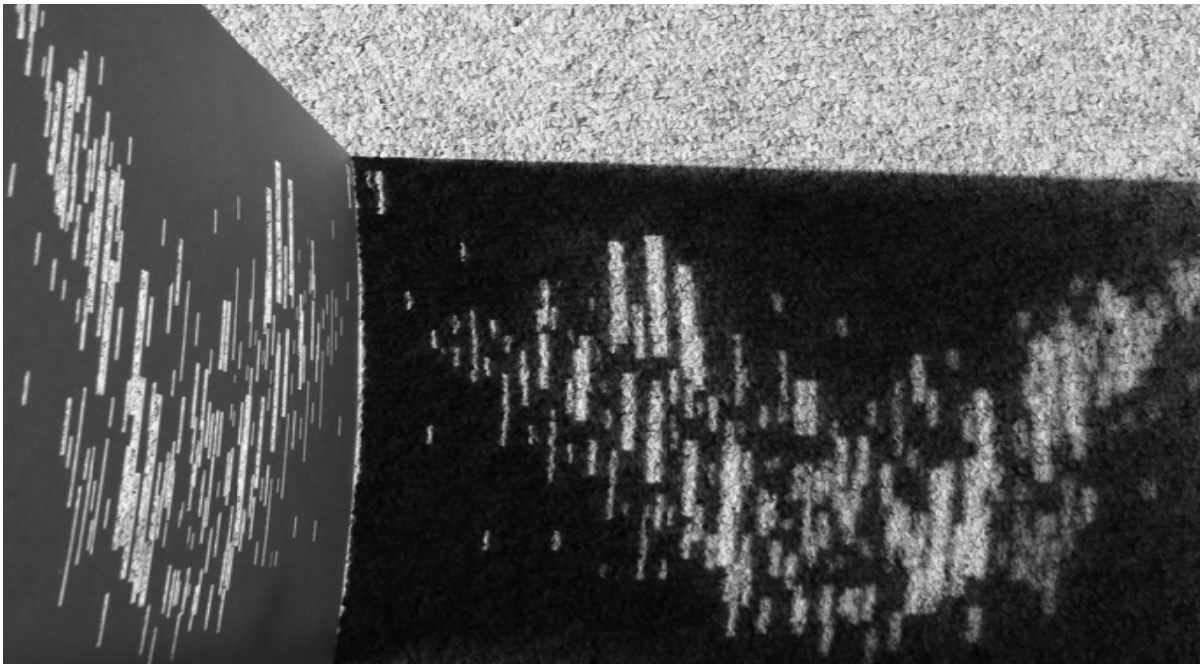


Fig.4.47. Manipulating the variables would result in differentiated models that I can choose the best one for my design purpose.



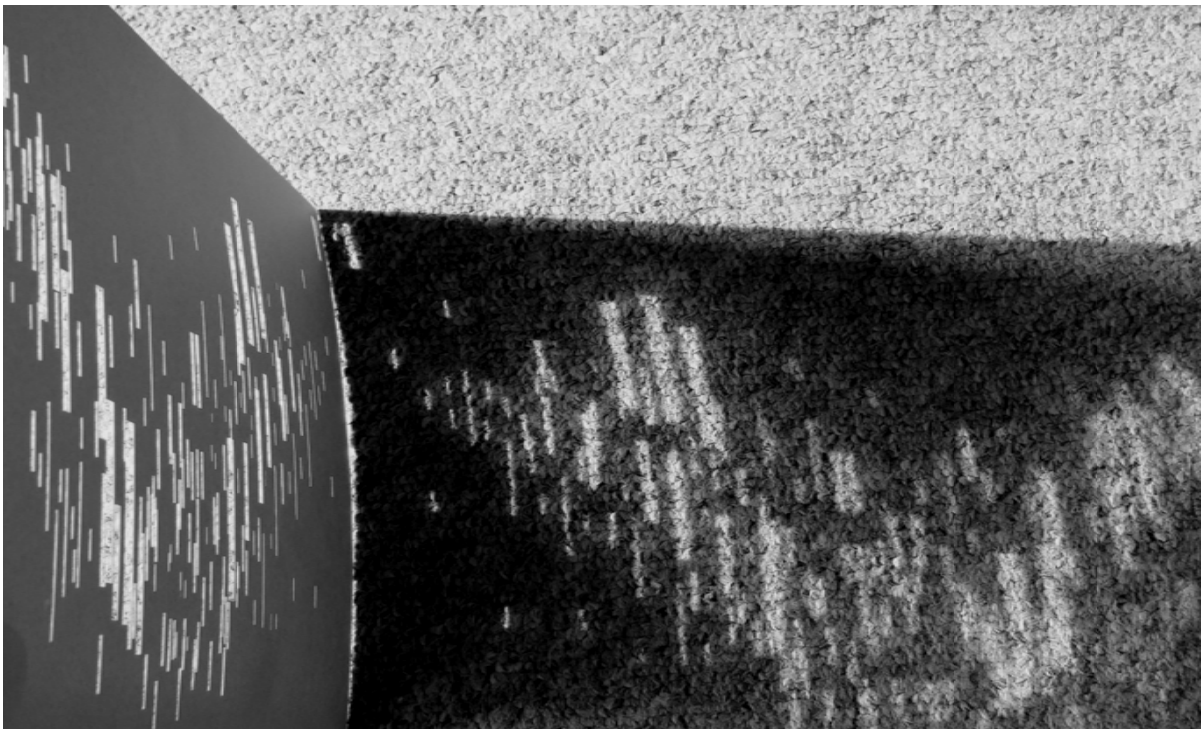


Fig.4.48. Different shadow effects of the final design product as a porous wall system.

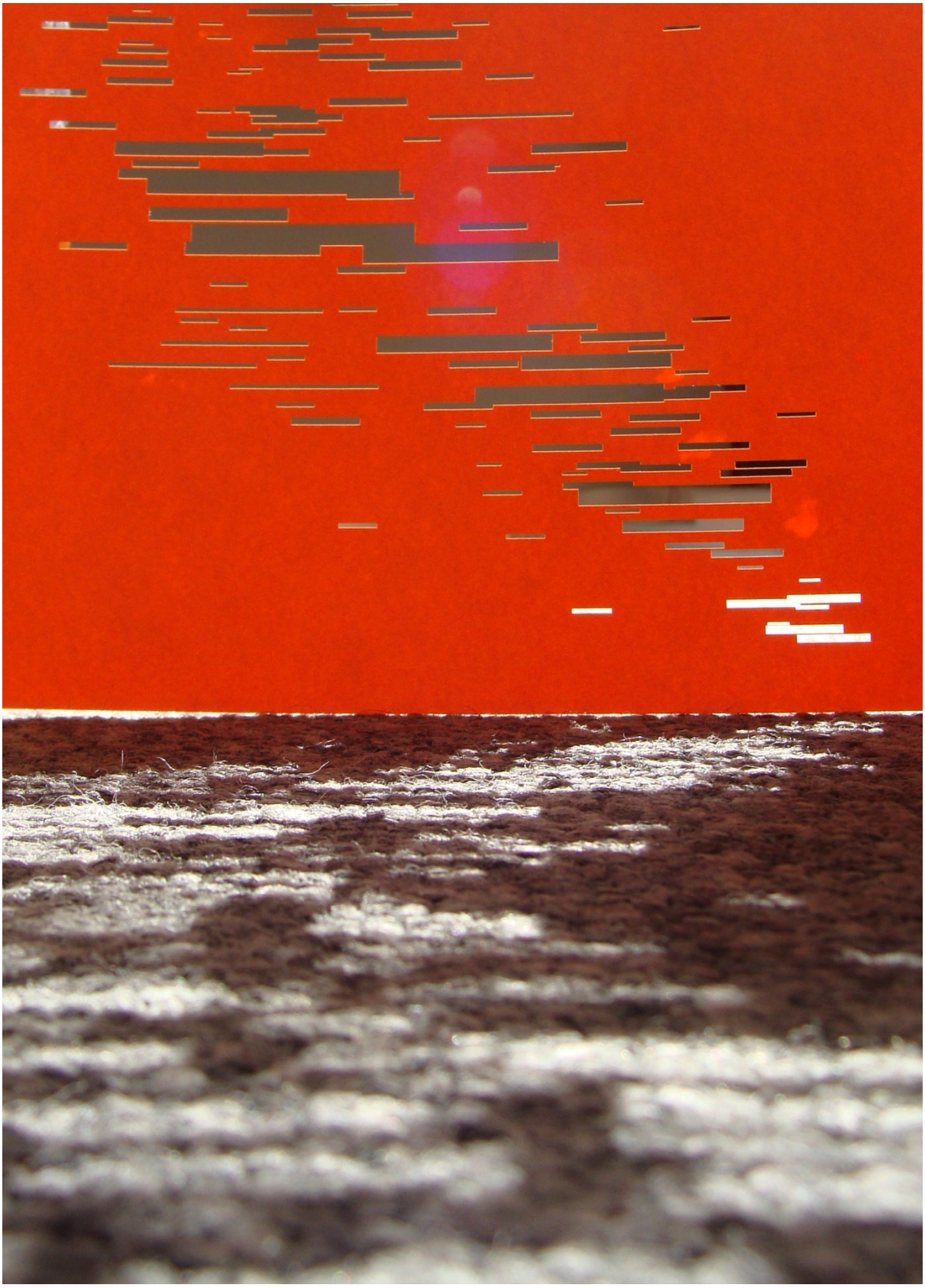
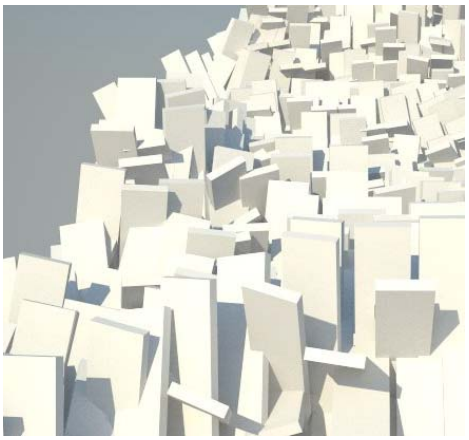


Fig.4.49. Final design product.

Chapter_ 5_Parametric Space



Chapter_ 5_Parametric Space

Our survey in Geometry looks for objects in the space; Digital representation of forms and tectonics; different articulation of elements and multiple processes of generations; from classical ideas of symmetry and pattern up to NURBS and curvature continuity.

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 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 system - flat space (x, y) - and one dimension-linear space (x) - as well. While parametric design shifts between these spaces, we need to understand them as parametric space a bit.

5_1_One Dimensional (1D) Parametric Space

The X axis 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 could be parameterised in 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 our 1D parameter space when we talk about a point, it could be described by a real number which is associated with a specific point on the curve we are dealing with.

It is important to know that since we are not working on the world X axis any more, any curve has its own parameter space and these parameters does not exactly match the universal measurement systems. Any curve in the Grasshopper has a parameter space starts from zero and ends in a positive real number (Fig.5.1).

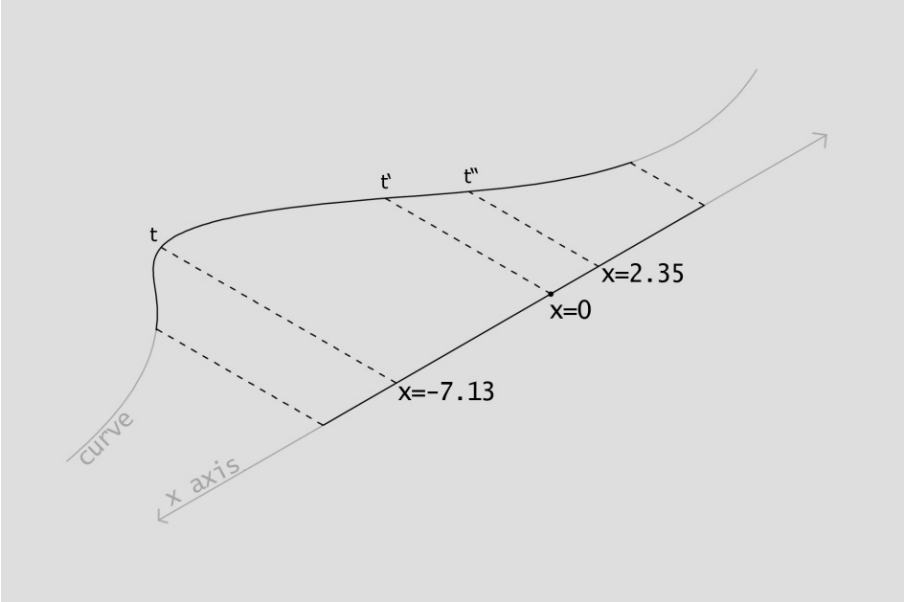


Fig.5.1. 1D-parameter space of a curve. Any 't' value is a real number associated with a position on the curve.

So talking about a curve and working and referencing some specific points on it, we do not need to always deal with points in 3D space with p=(X,Y,Z) but we can recall a point on a curve by p=t as a specific parameter on it. And it is obvious that we can always convert this parameter space to a point in the world coordinate system. (Fig.5.2)

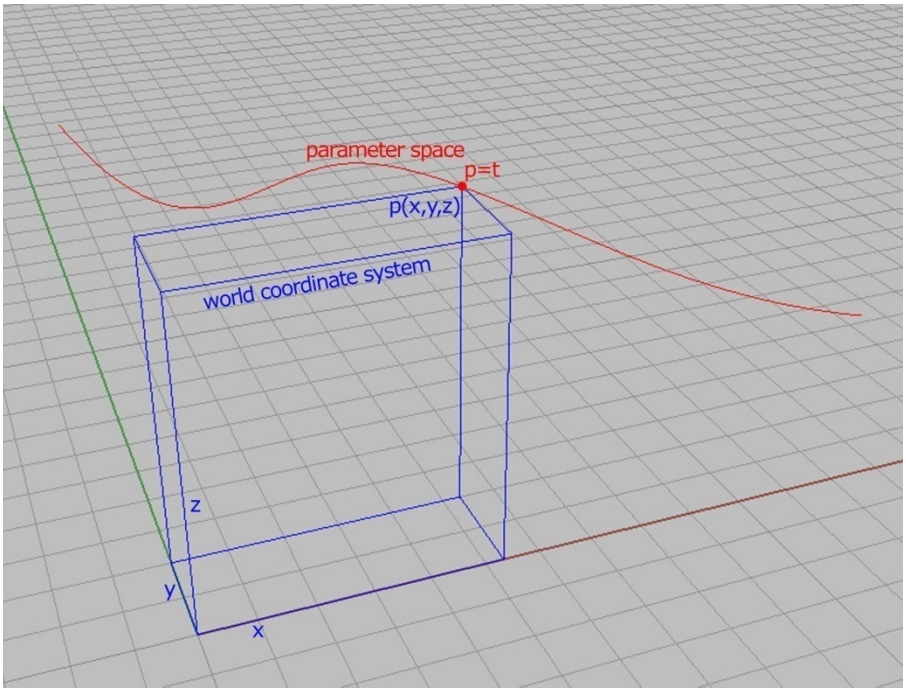


Fig.5.2. 1D-parmeter space and conversion in 3D coordinate system.

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 that each point on this space 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 on any surface in the space. So basically we can parameterize a coordinate system on a curved surface in the space, and call different points of it by a pair of numbers here known as UV space, in which P=(U,V) on the surface. Again we do not need to work with 3 values of (X,Y,Z) as 3D space to find the point and instead of that we can work with the UV "parameters" of the surface. (Fig.5.3)

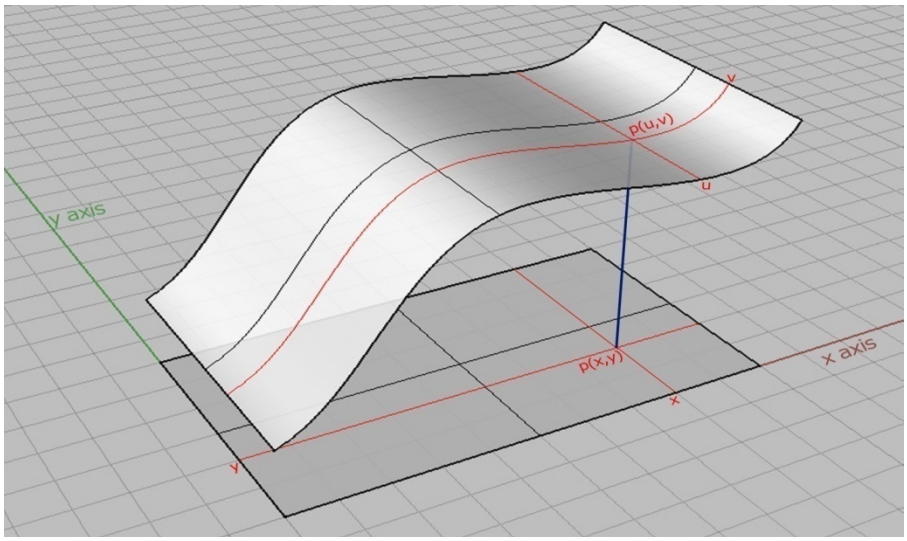


Fig.5.3. UV (2D) parameter space of surface.

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 it parametric! Again we have access to the 3D equivalent coordinate of any point on the surface (Fig.5.4).

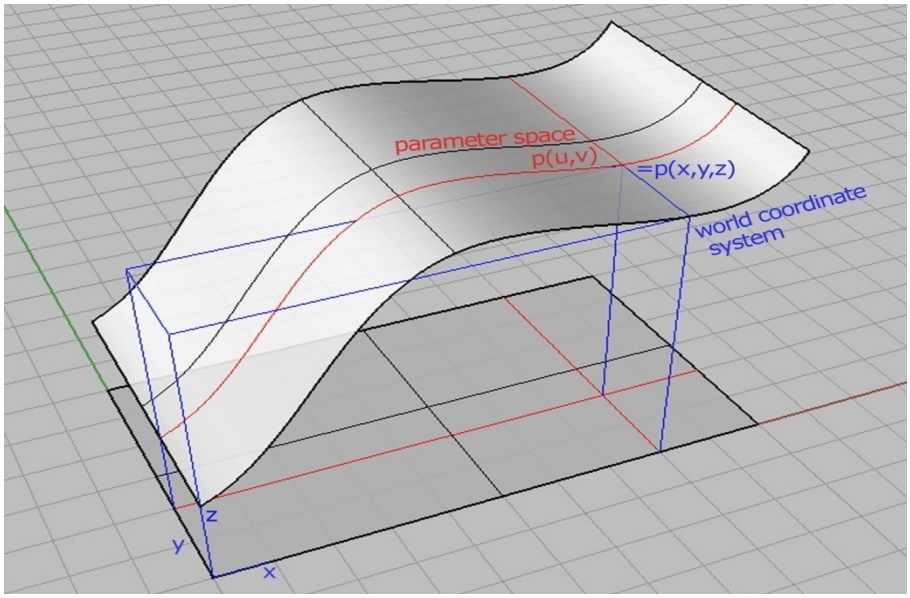


Fig.5.4. Equivalent of the point P=(U,V) on the world coordinate system p=(X,Y,Z).

5_3_Transition between spaces

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

Consider that it is not always a parameter or a value in a coordinate system that we need in order to call geometries in Generative Algorithms and Grasshopper, sometimes we need just an index number to do it. If we are working with a bunch of points, lines or whatever, and they have been generated as a group of objects, like point clouds, since each object associated with a natural number that shows its position in a list of all objects, we just need to call the number of the object as index instead of any coordinate system. The index numbering like array variables in programming is a 0-based counting system which starts from 0 (Fig.5.5).

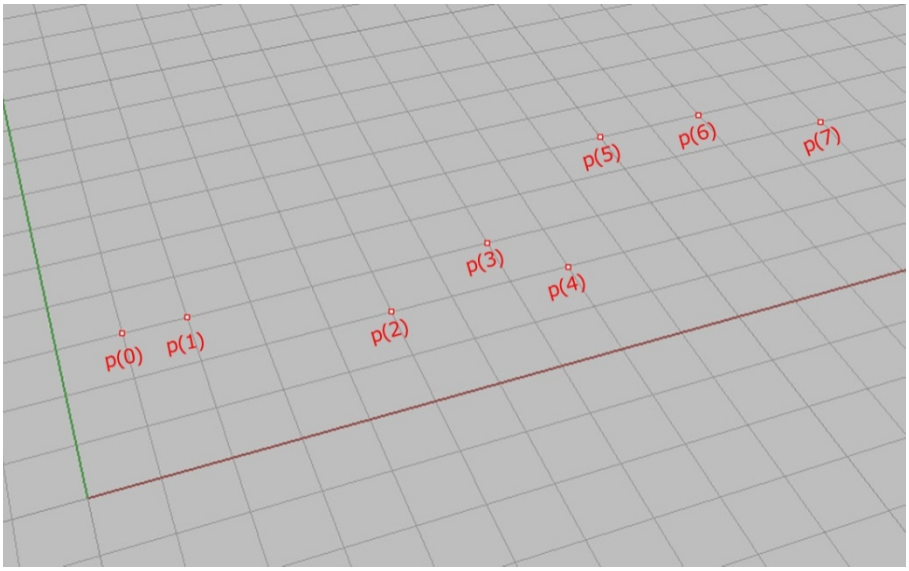


*Fig.5.5. Index number of a group of object is a simple way to call an on object. This is **0-based counting system** which means numbers start from 0.*

So as mentioned before, in Associative modelling we generate our geometries step by step as some related objects and for this reason we go into the parameter space of each object and extract specific information of it and use it as the base data for the next steps. This could be started from a simple field of points as basic generators and ends up at the tiny details of the model, in different hierarchies.

5_4_Basic Parametric Components

5_4_1_Curve Evaluation

The <evaluate> component is the function that can find the point on a curve or surface, based on the parameter you feed. The <evaluate curve> component (Curve > Analysis > Evaluate curve) takes a curve and a parameter (a number) and gives back a point on curve on that parameter.

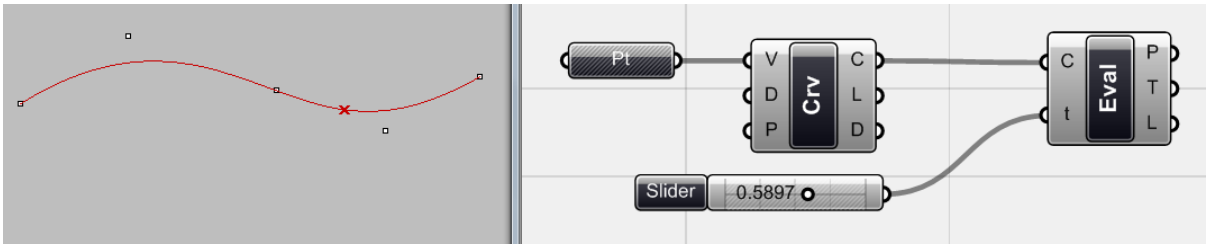


Fig.5.6. The evaluated point on <curve> on the specific parameter which comes from the <number slider>.

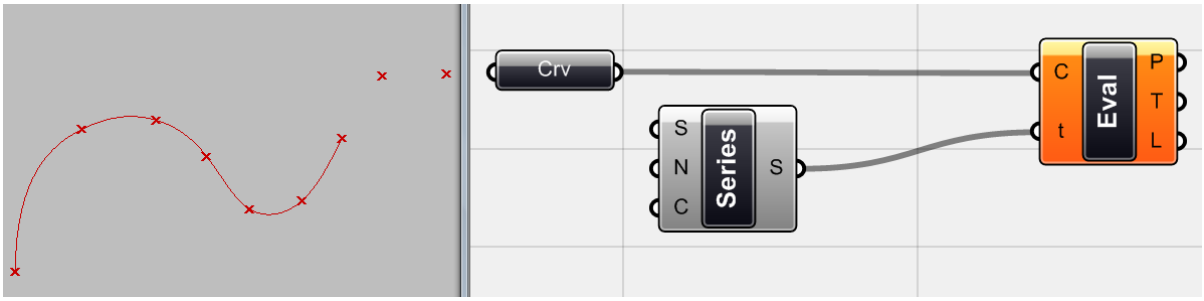


Fig.5.7. We can use <series> of numbers as parameters to <evaluate> instead of one parameter. In the above example, because some numbers of the <series> component are bigger than the domain of the curve, you see that <Evaluate> component gives us warning (becomes orange) and that points are located on the imaginary continuation of the curve.

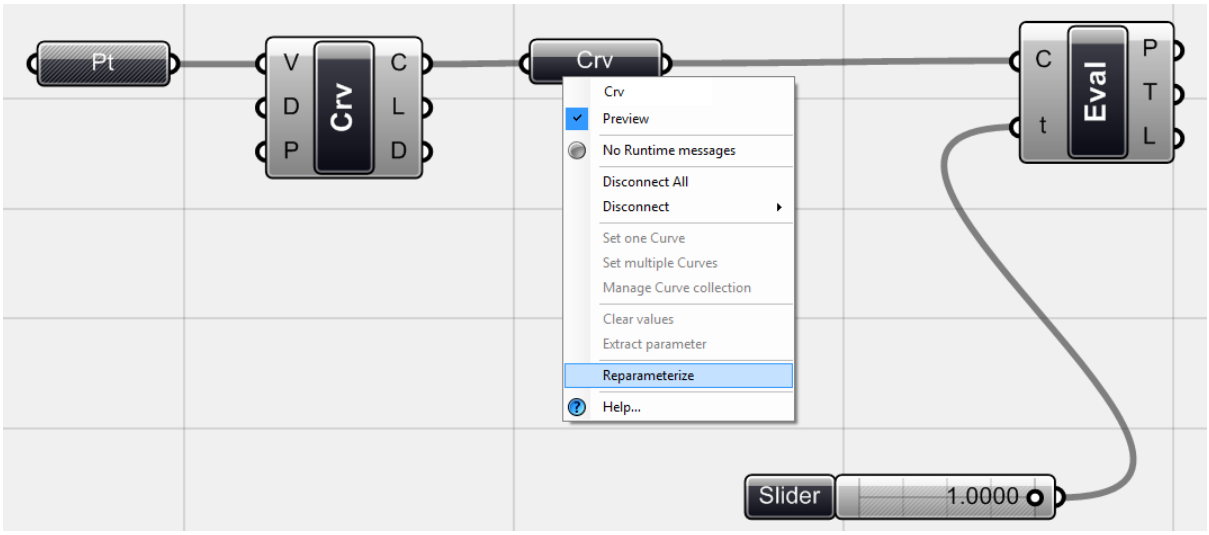


*Fig.5.8. Although the 'D' output of the <curve> component gives us the domain of the curve (minimum and maximum parameters of the curve), alternatively we can feed an external <curve> component from Param > Geometry and in its context menu, check the **Reparameterize** section. It changes the domain of the curve to 0 to 1. So basically I can track all <curve> long by a <number slider> or any numerical set between 0 and 1 and not be worry that parameter might go beyond the numerical domain of the curve.*

There are other useful components for parameter space on curves on Curves > Analysis and Division that we talk about them later.

5_4_2_Surface Evaluation

While for evaluating a curve we need a number as parameter (because curve is a 1D-space) for surfaces we need a pair of numbers as parameters (U, V), with them, we can evaluate a specific point on a surface. We use <evaluate surface> component (Surface > Analysis > Analysis) to evaluate a point on a surface on specific parameters.

We can simply use <point> components to evaluate a surface, by using it as UV input of the <Evaluate surface> (it ignores Z dimension) and you can track your points on the surface just by X and Y parts of the <point> as U and V parameters.

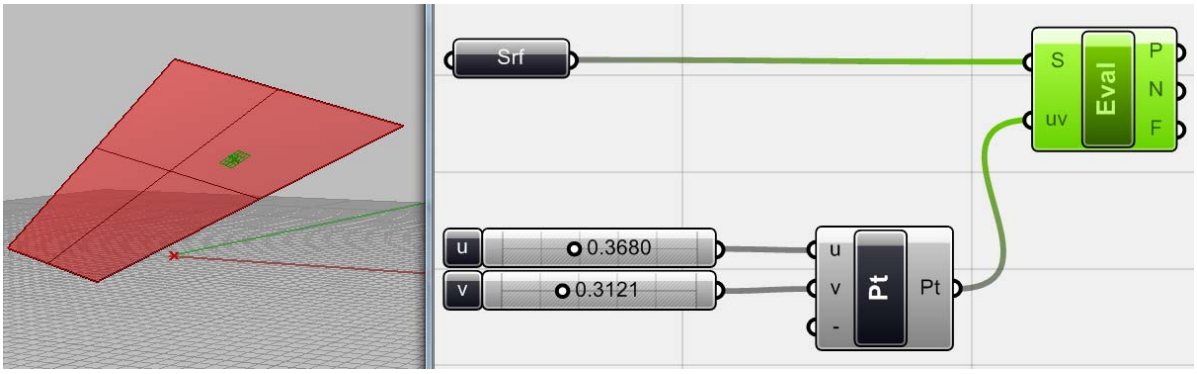


Fig.5.9. A point <Evaluate>d on the <surface> base on the U,V parameters coming from the <number slider> with a <point> component that make them a pair of Numbers. Again like curves you can check the 'Reparameterize' on the context menu of the <surface> and set the domain of the surface 0 to 1 in both U and V direction. Change the U and V by <number slider> and see how this <evaluated> point moves on the surface (I renamed the X,Y,Z inputs of the component to U,V,- manually).

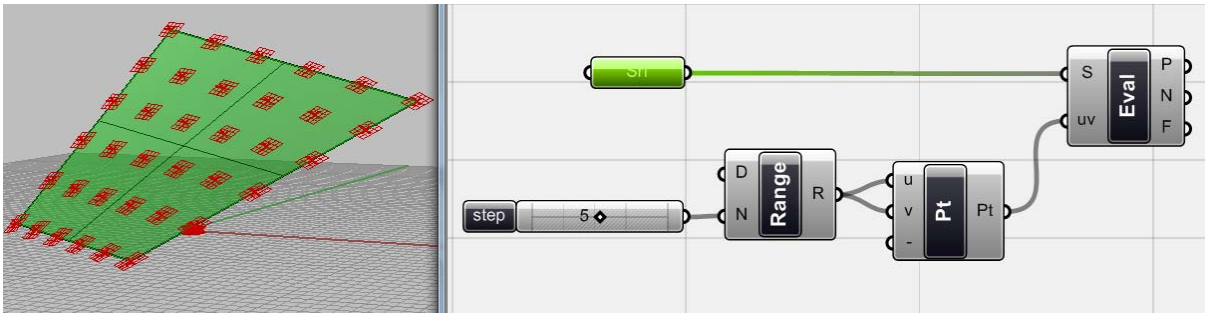


Fig.5.10. Since we can use <point> to <evaluate> a <surface> as you see we can use any method that we used to generate points to evaluate on the <surface> and our options are not limited just to a pair of parameters coming from <number slider>, and we can track a surface with so many different ways.

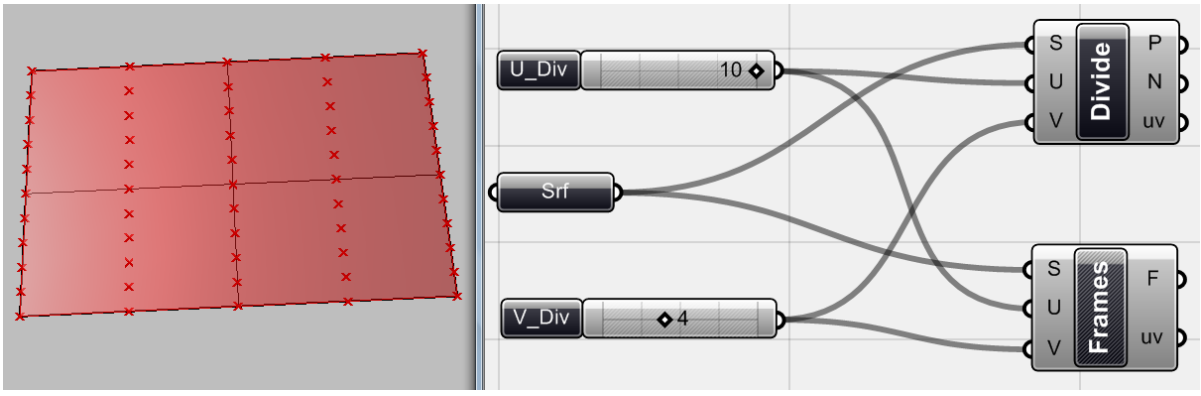


Fig.5.11. To divide a surface (like the above example) in certain rows and columns we can use <Divide surface> or if we need some planes across certain rows and columns of a surface we can use <surface frame> both from Surface tab under Util section.

5_5_On Object Proliferation in Parametric Space

For so many design reasons, designers now use surfaces to proliferate some other geometries on them. Surfaces are flexible, continues two dimensional objects that prepare a good base for this purpose. There are multiple methods to deal with surfaces like Penalisation, but here I am going to start with one of the simplest one and we will discuss about some other methods later.

We have a free-form surface and a simple geometry like a box. The question is, how we can proliferate this box over the surface, in order to have a differentiated surface i.e. as an envelope, in that we have control of the macro scale (surface) and micro scale (box) of the design separately, but in an associative way.

In order to do this, we should deal with this surface issue by dividing it to desired parts and generate our boxes on these specific locations on the surface and readjust them if we want to have local manipulation of these objects.

Generating the desired locations on the surface is easy. We can divide surface or we can generate some points based on any numerical data set that we want.

About the local manipulation of proliferated geometries, again we need some numerical data sets which could be used for transformations like rotation, local displacement, resize, adjustment, etc.

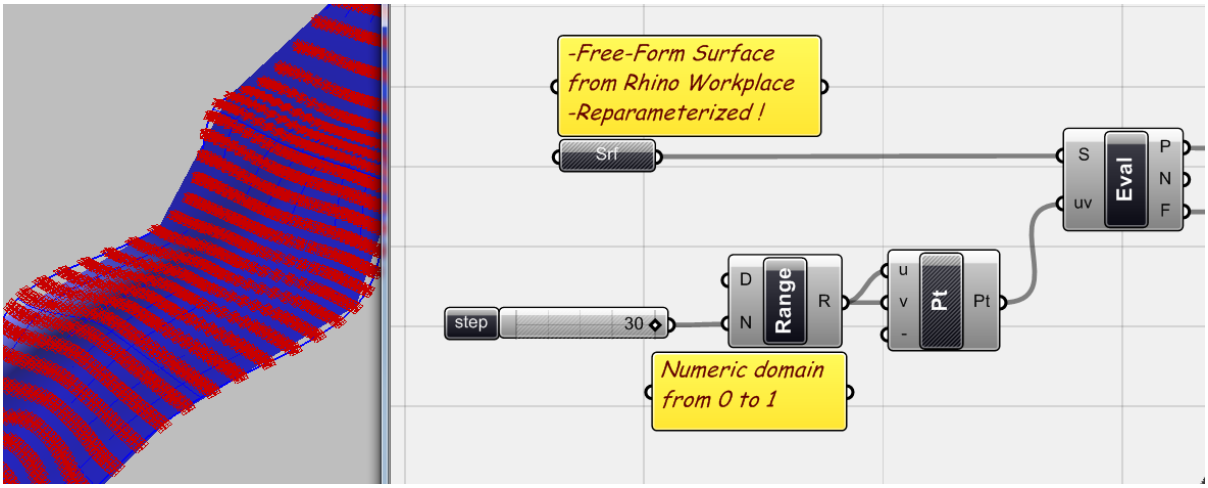


Fig.5.12. A free-form, reparameterized, <surface> being <evaluate>d by a numeric <range> from 0 to 1, divided by 30 steps by <number slider> in both U and V direction. (Here you can use <divide surface> but I still used the <point> component to show you the possibilities of using points in any desired way).

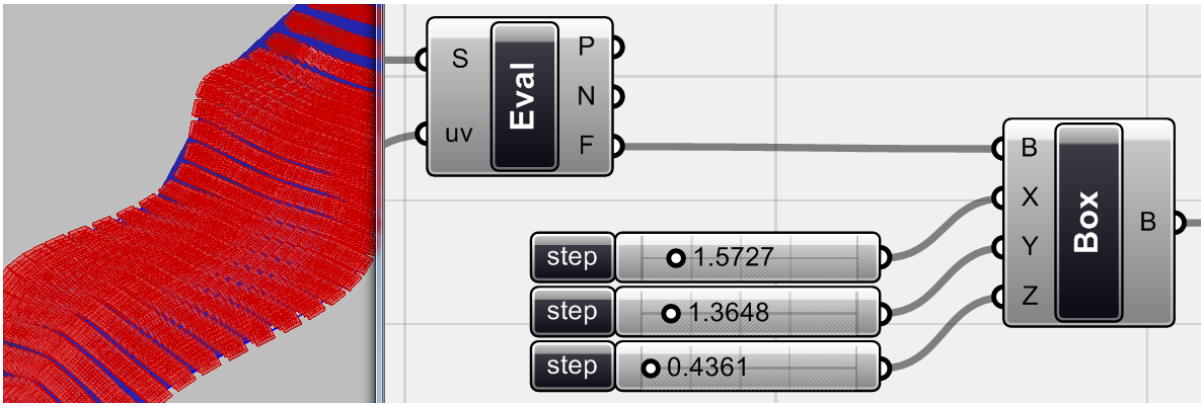


Fig.5.13. As you see the <evaluate> component gives 'Normal' and 'plane' of any evaluated points on the surface. I used these frames to generate series of <box>es on them while their sizes are being controlled by <number slider>s.

In order to manipulate the boxes locally, I just decided to rotate them, and I want to set the rotation axis the Y direction of the coordinate system so I should use the XZ plane as the base plane for their rotation (Fig.5.13).

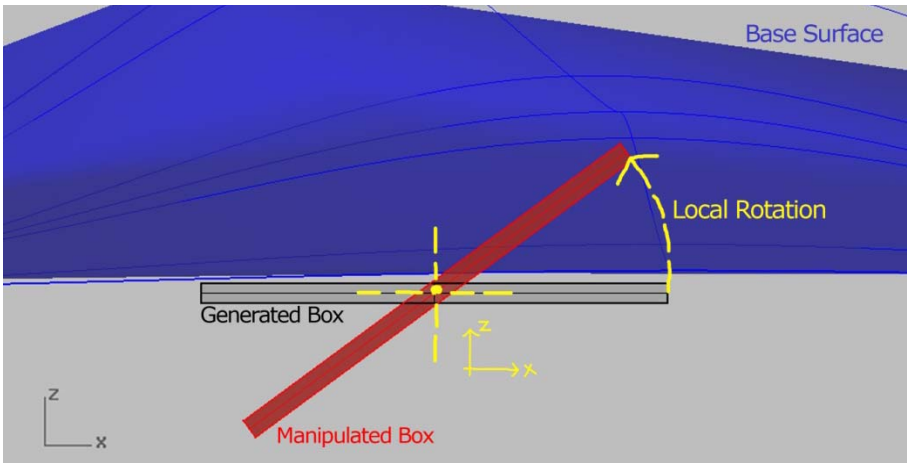


Fig.5.14. Local rotation of the box.

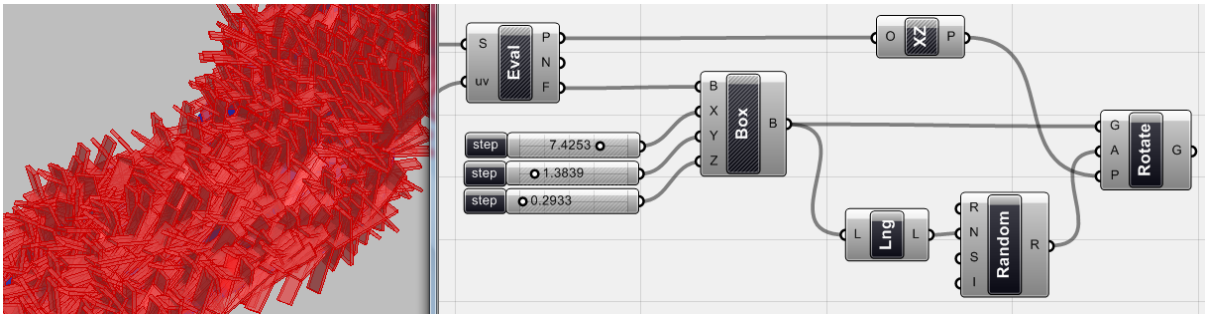


Fig.5.15. The <rotate> component needs 'geometry' which I attached <box>es and 'rotation angle' that I used random values (you can rotate them gradually or any other way) and I set the Number of random values as much as boxes. Finally to define the plane of axis, I generated <XZ plane>s on any point that I <evaluate>d on the <surface> and I attached it to the <rotate> component.

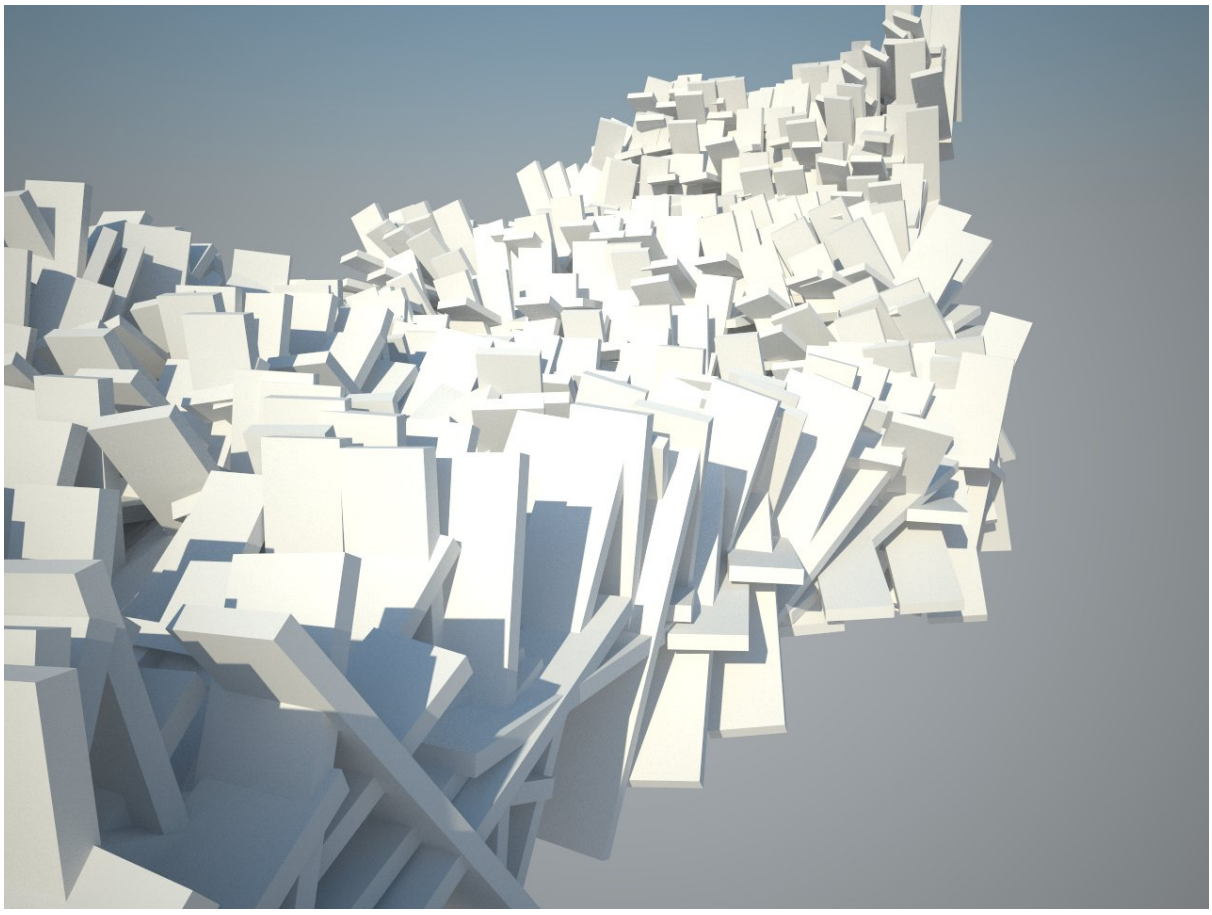


Fig.5.16. Final geometry.

Non-uniform use of evaluation

During a project this idea came to my mind that why should I always use the uniform distribution of the points over a surface and add components to it? Can I set some criteria and evaluate my surface based on that and select specific positions on the surface? Or since we use the U,V parameter space and incremental data sets (or incremental loops in scripting) are we always limited to a rectangular division on surfaces?

There are couple of questions regarding the parametric tracking a surface but here I am going to deal with a simple example to show how in specific situations we can use some of the U,V parameters of a surface and not a uniform rectangular grid over it.

Social Space

I have two Free-form surfaces as covers for a space and I think to make a social open space in between. I want to add some columns between these surfaces but because they are free-form surfaces and I don't want to make a grid of columns, I decided to limit the column's length and add as many places as possible. I want to add two inverted and intersected cone as columns in this space just to make the shape of them simple.

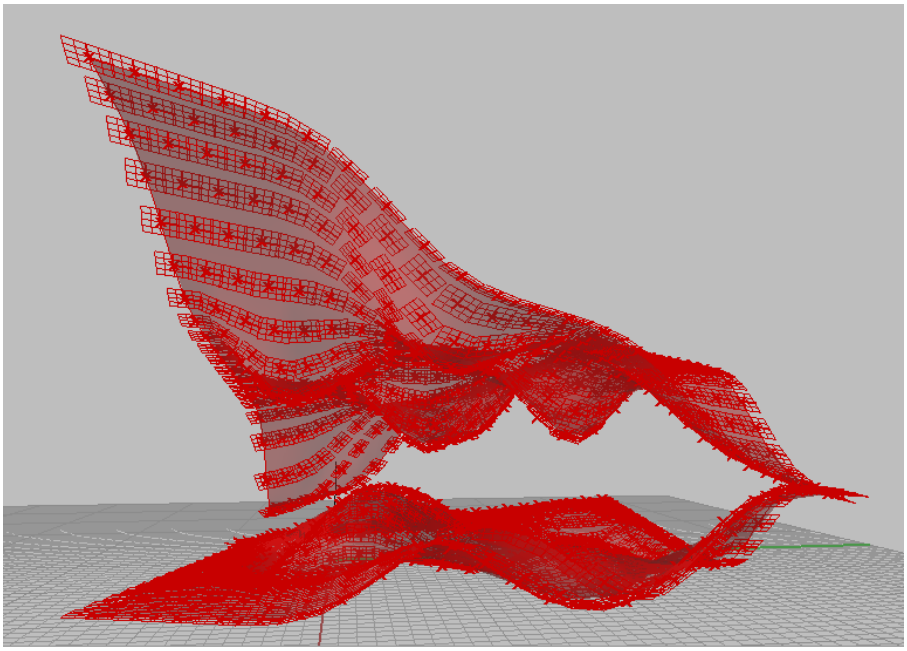


Fig.5.17. Primary surfaces as covers of the space.

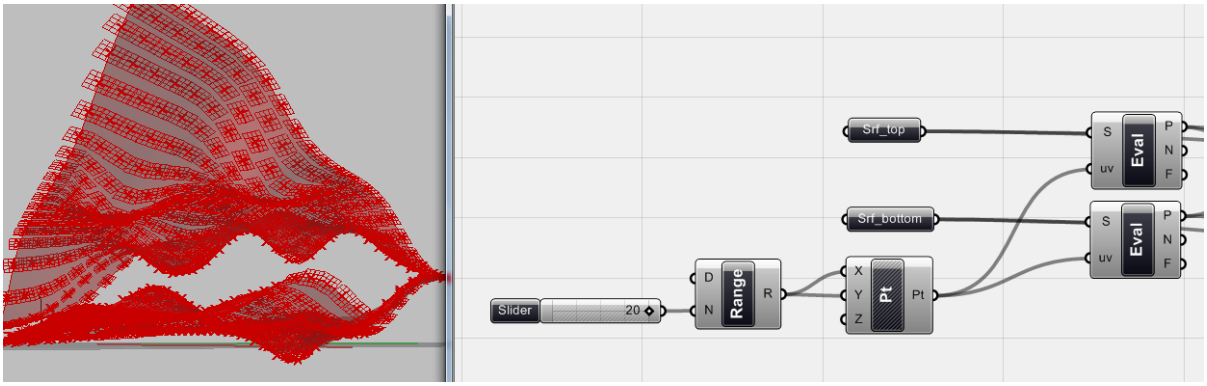


Fig.5.18. I introduced surfaces to Grasshopper by <srf_top> and <srf_bottom> and I Reparameterized them. I also generated a numerical <range> between 0 and 1, divided by <number slider>, and by using a <point> component I <evaluate> these surfaces at that <points>. Again just to say that still it is the same as surface division.

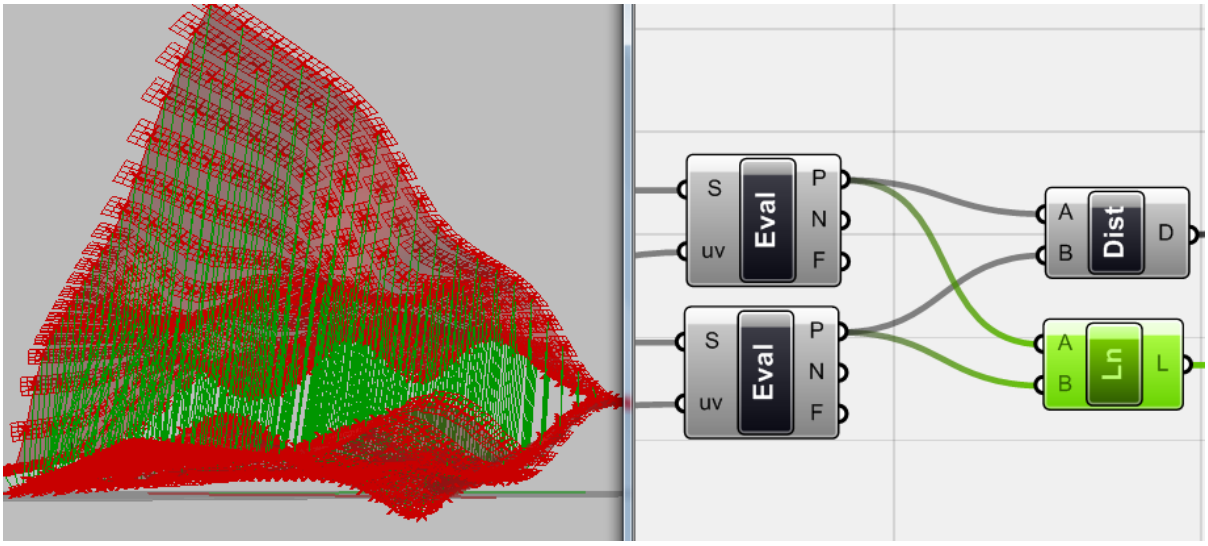


Fig.5.19. I generated bunch of <line>s between all these points, but I also measured the distance between any pair of points (we can use line length also), as I said I want to limit these lines by their length.

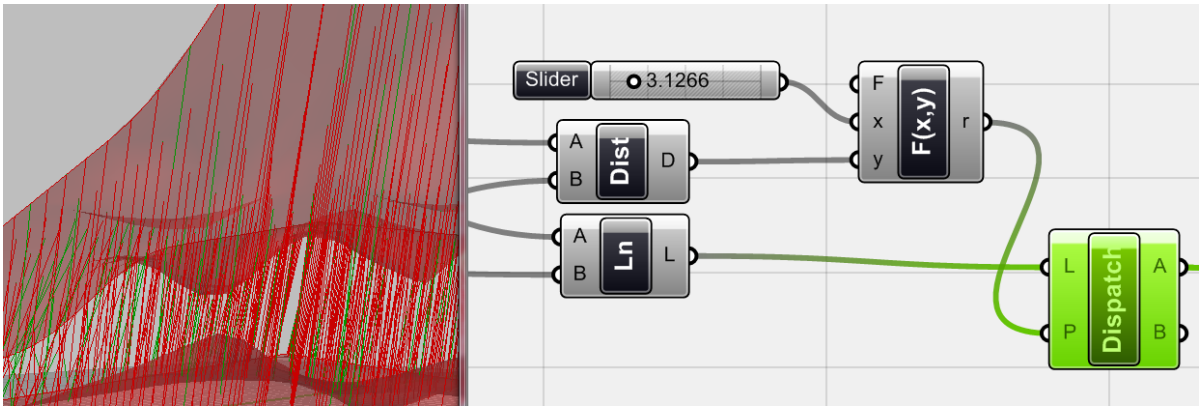


Fig.5.20. Here I used a <dispatch> component (Logic > Streams > Dispatch) to select my lines from the list. A <dispatch> component needs Boolean data which is associated with the data from the list to sent those who associated with True to the A output and False one to the B output. The Boolean data comes from a simple comparison function. In this <function> I compared the line length with a given number as maximum length of the line (x>y, x=<number slider>, y=<distance>). Any line length less than the <number slider> creates a True value by the function and passes it through the <dispatch> component to the A output. So if I use the lines coming out the output of the <dispatch> I am sure that they are all less than the certain length, so they are my columns.

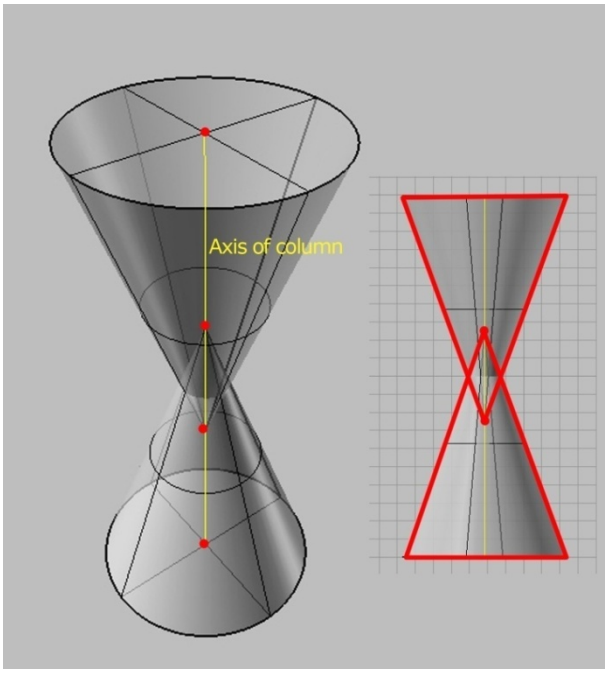


Fig.5.21. The geometry of columns is just two inverted cones which are intersecting at their tips. Here because I have the axis of the column, I want to draw to circles at the end points of the axis and then extrude them to the points on the curve which make this intersection possible.

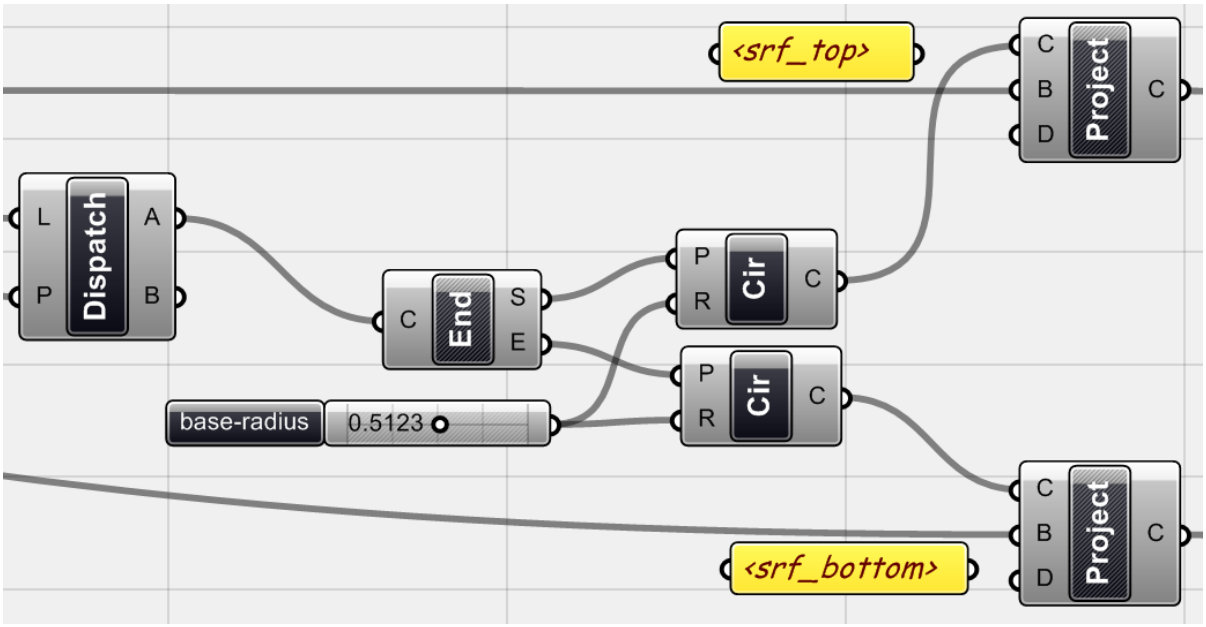


Fig.5.22. By using an <end points> component I can get the both ends of the column. So I attached these points as base points to make <circle>s with given radius. But you already know that these circles are flat but our surfaces are not flat. So I need to <project> my circles on surfaces to find their adjusted shape. So I used a <project> component (Curve > Util > Project) for this reason.

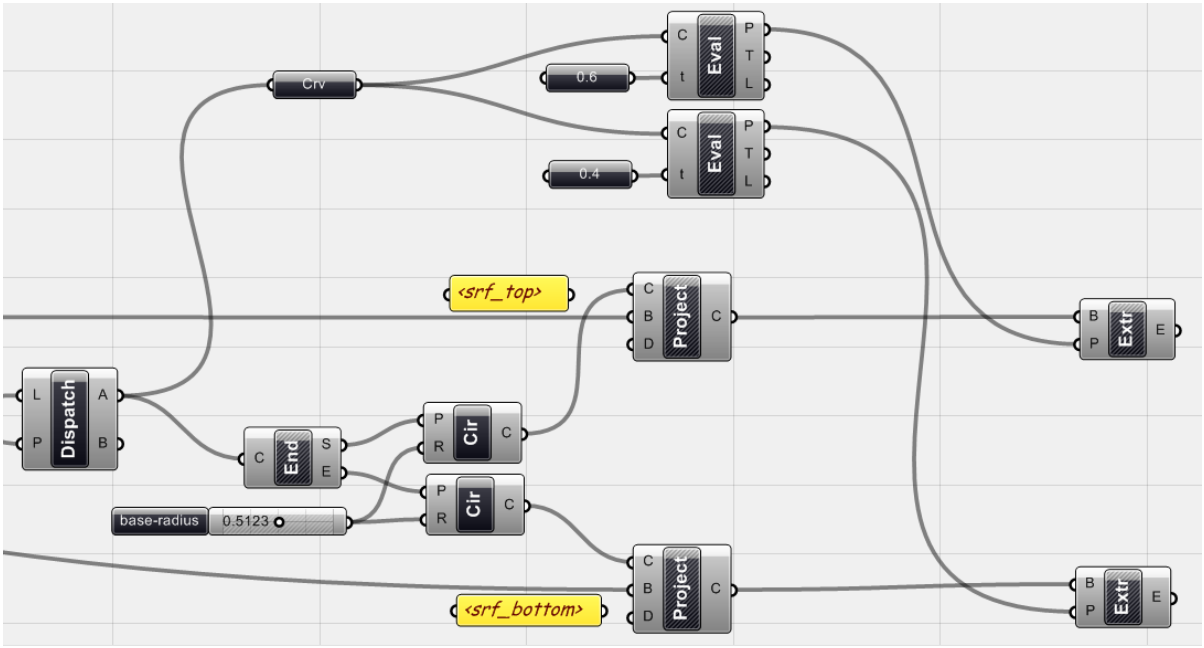


Fig.5.23. The final step is to extrude these projected circles towards the specified points on column's axis (Fig.5.20). So I used <extrude point> component (Surface > Freeform > Extrude point) and I connected the <project>ed circles as base curves. For the extrusion point, I attached all columns' axis to a simple <curve> component and I 'Reparameterized' them, then I <evaluate>d them in two specific parameter of 0.6 for top cones and 0.4 for bottom cones.

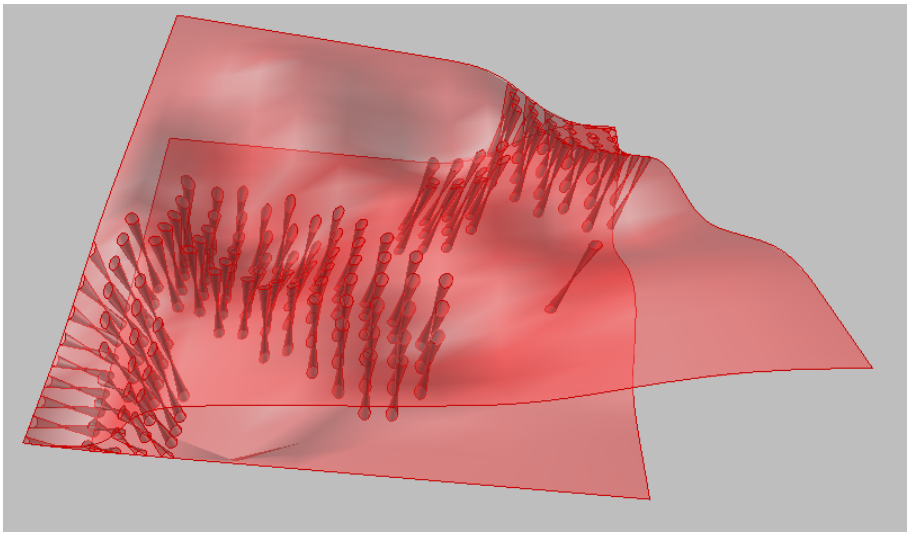


Fig.5.24. Although in this example, again I used the grid based tracking of the surface, I used additional criteria to choose some of the points and not all of them uniformly.

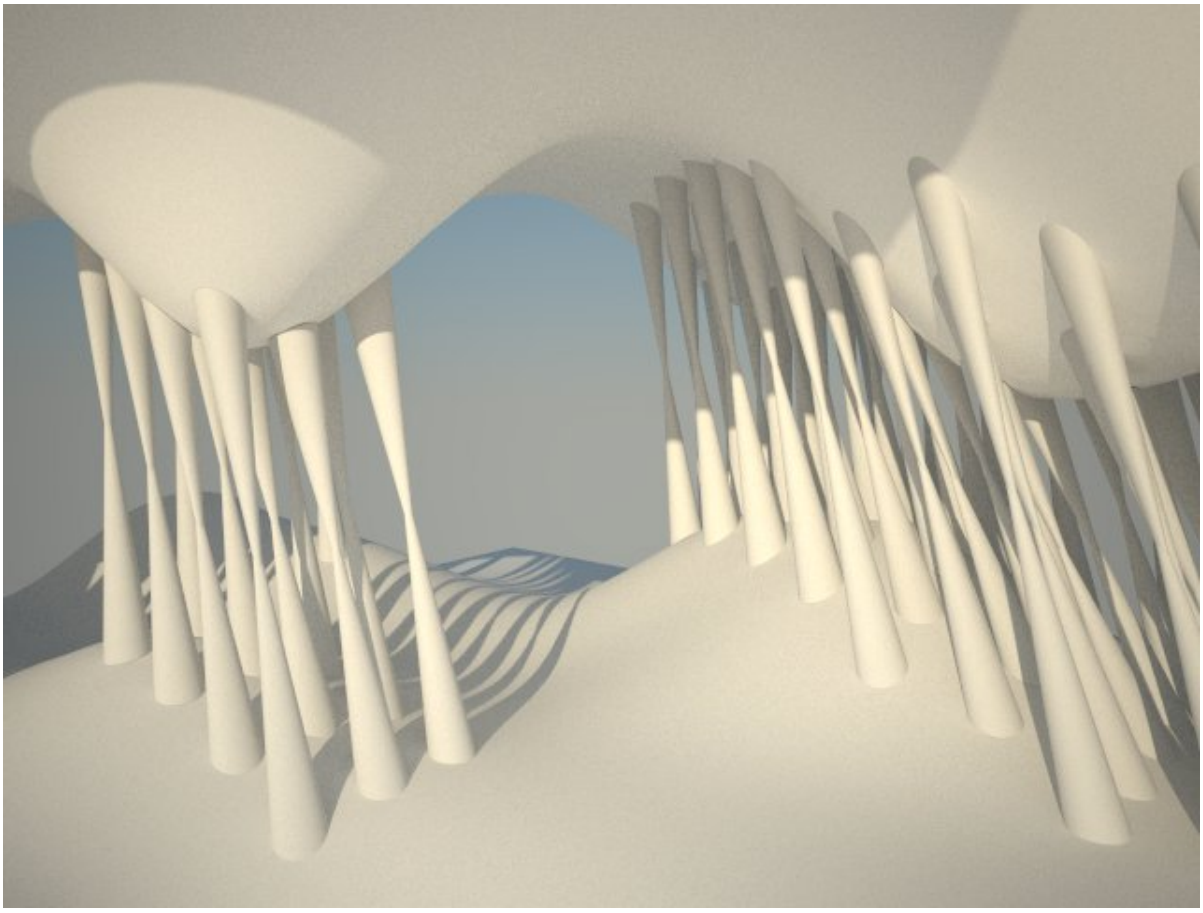
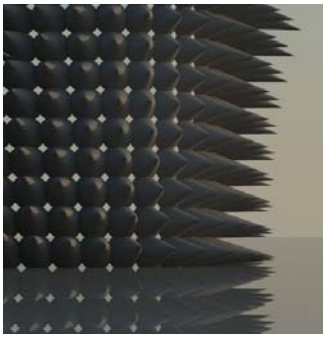


Fig.5.25. Final model.

Chapter_6_ Deformation and Morphing



Chapter_6_ Deformation and Morphing

6_1_Deformation and Morphing

Deformation and Morphing are among the powerful functions in the realm of free-form design. By deformations we can twist, shear, blend, … geometries and by Morphing we can deform geometries from one boundary condition to another.

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.

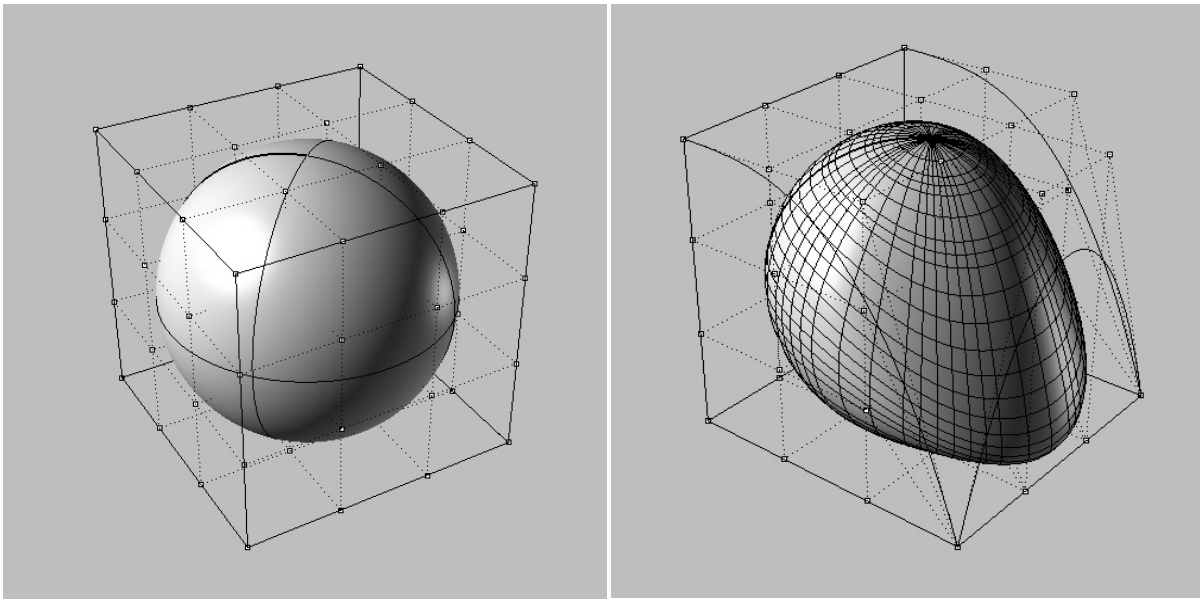


Fig.6.1. Deformation by Bounding-box (cage).

Based on different manipulations, we might call it shear or blend or free deformation. For any deformation function, we might need the whole bounding-box, or just one of its sides as a plane or even one of the points to deform. If you check different deformation components in Grasshopper you can easily find the base geometrical constructs to perform the deformations.

Morphing in animation means transition from one picture to another smoothly or seamlessly. Here in 3D space it means deformation from one state or boundary condition to another. The Morphing components in Grasshopper work in the same fashion. There are two <morph> components, one deform an object from a reference box (Bounding Box) to a target box, the other component works with a surface as a base on that you can deform your geometry, on the specified domains of the surface and height of the object.

The first one is <Box Morph> and the next one is <Surface Morph> both from XForm tab under the Morph section. Since we have couple of commands that deform a box, if we use these deformed boxes as target boxes then we can deform any geometry in Grasshopper by combination with Box Morph component.

As you see in Figure.6.2 we have an object which is introduced to Grasshopper by a <Geometry> component. This object has a bounding-box around it which I draw here just to visualize the situation. I also draw another box by manually feeding values.

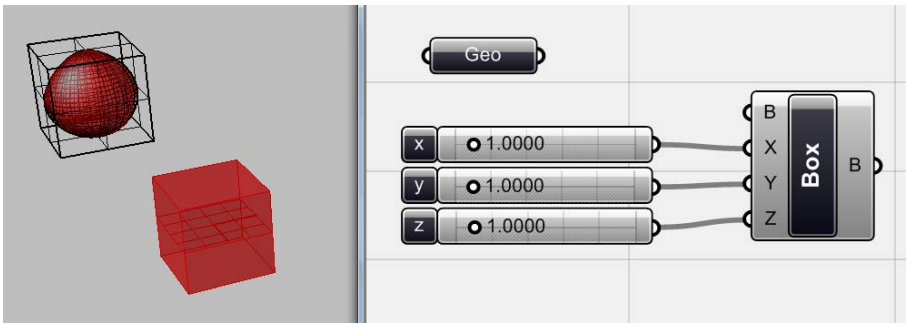


Fig.6.2. Object and manually drawn box.

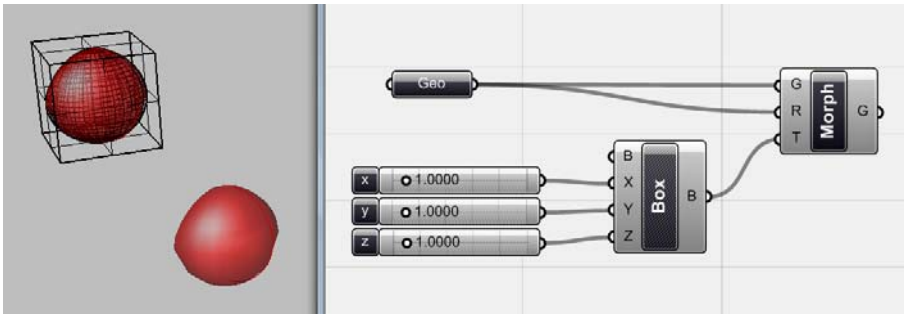


Fig.6.3. The <Box morph> component (XForm > Morph > Box morph) deforms an object from a reference box to a target box. Because I have only one geometry I attached it as a bounding box or reference box to the component but in other cases, you can use <Bounding box> component (Surface > Primitive > Bounding box) to use as the source box. I unchecked the preview of the <Box> component to see the morphed geometry better.

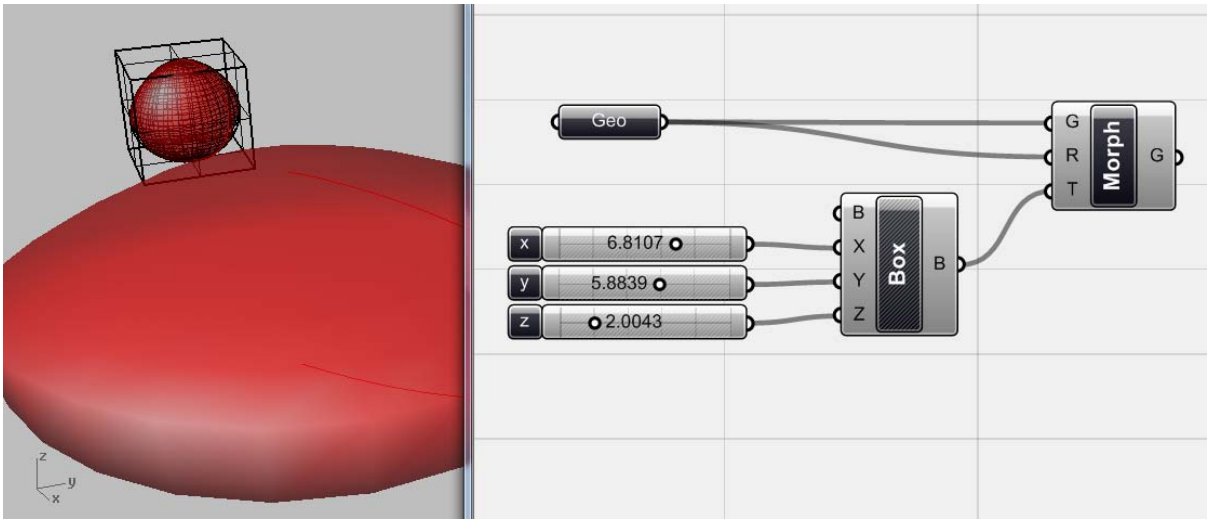


Fig.6.4. Now if you simply change the size of the target box you can see that the morphed geometry would change accordingly.

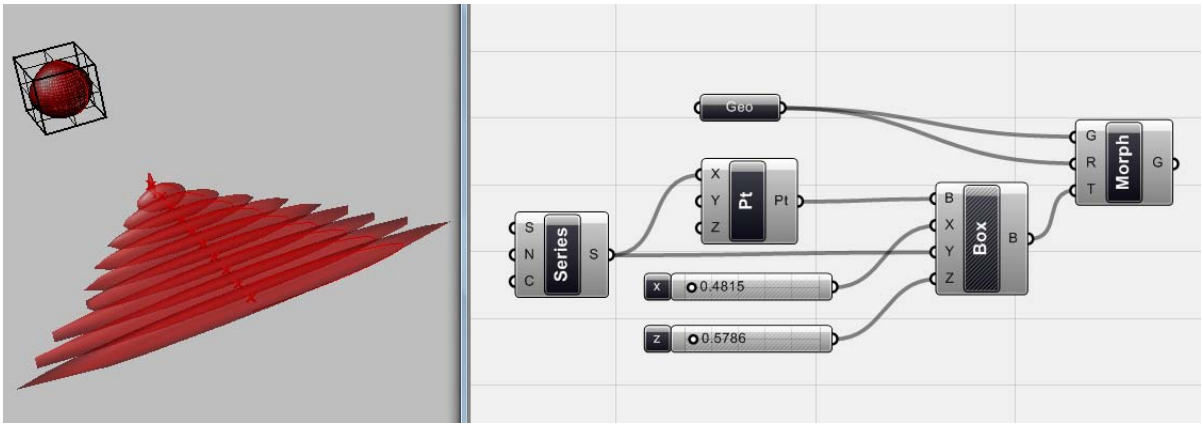


Fig.6.5. here you see that instead of one box, if I produce bunch of boxes, we can start to morph our object more and more. As you see the differentiated boxes by the <series> component in their Y dimension, show the differentiation in the morphed object as well.

6_2_On Panelization

One of the most common applications of the morphing functions is Panelization. The idea of panelization comes from the division of a free-form surface geometry into small parts especially for fabrication issues. 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. The idea of panelization is to divide a surface into small parts which are easier to fabricate and transport and also more controllable in the final product. Sometimes the reason is to divide a curve surface into small flat parts and then get the curvature by the accumulation of the flat geometries which could be then fabricated from sheet materials. There are multiple issues regarding the size, curvature, adjustment, etc. that we try to discuss some of them.

Let's start with a simple surface and a component as the module to make this surface.

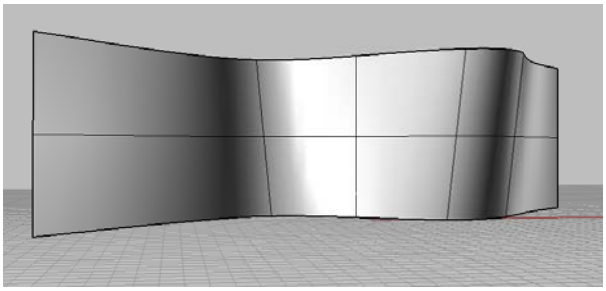


Fig.6.6. A simple double-curve surface for panelization.

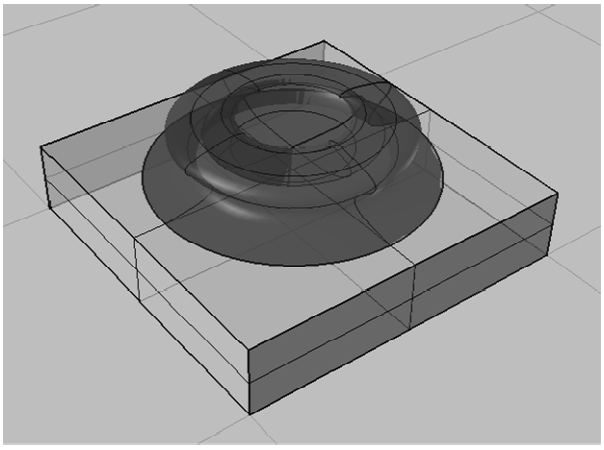


Fig.6.7. The component that I want to proliferate on the surface……. Not special, just for example !!!

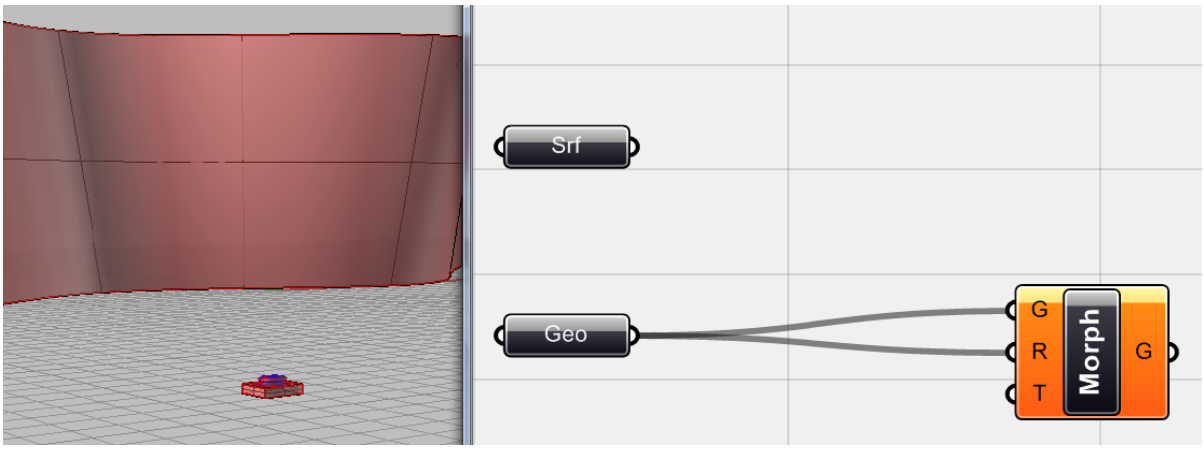


Fig.6.8. First of all, we need to introduce our surface and module as Grasshopper components. Based on the possible components in the Grasshopper, the idea is to generate couple of boxes on the surface and use these boxes as target boxes and morph or module into them. So I used the <box morph> and I used the geometry itself as bounding-box. Now we need to generate our target boxes to morph the component into them.

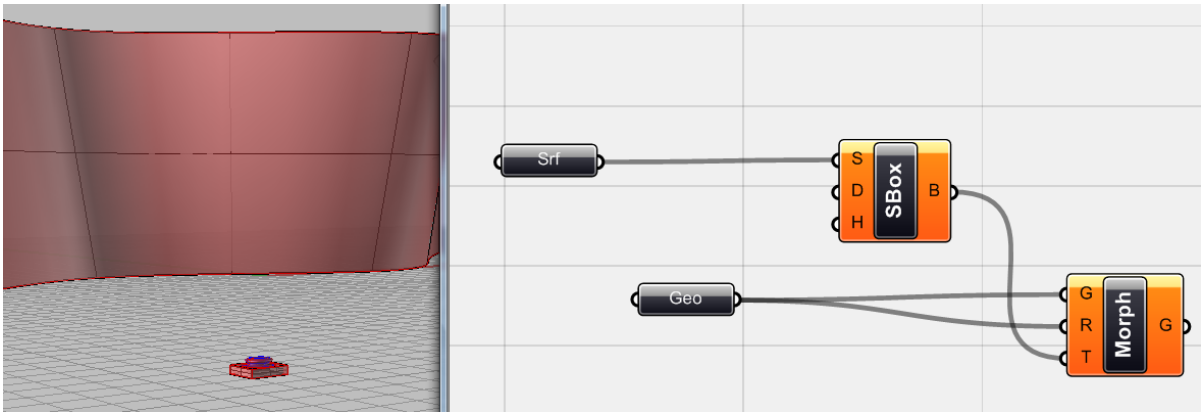


Fig.6.9. The component that we need to make our target boxes is <surface box> (XForm > Morph > Surface box). This component generates multiple boxes over a surface based on the intervals on the surface domain and height of the box. So I just attached the surface to it and the result would be the target boxes for the <box morph> component. Here I need to define the domain interval of the boxes, or actually number of boxes in each U and V direction of the surface.

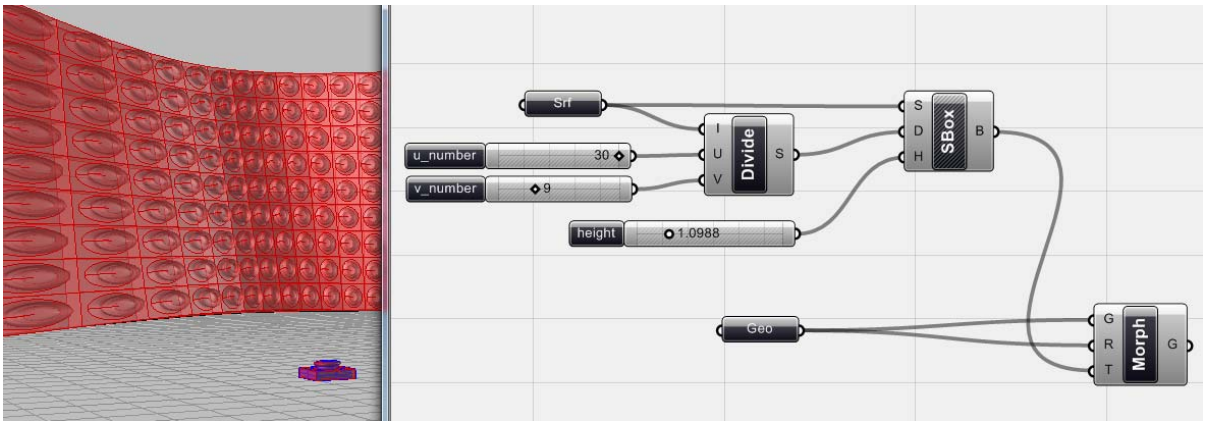


Fig.6.10. Now I connected <divide interval2> which tells the <surface box> that how many divisions in U and V directions we need. Another <number slider> defines the height of the target boxes which means height of the morphed components.

So basically the whole idea is simple. We produce a module (a component) and we design our surface. Then we make certain amount of boxes over this surface (as target boxes) and then we morph the module into these boxes. After all we can change the number of elements in both U and V direction and also change the module which updates automatically on the surface.

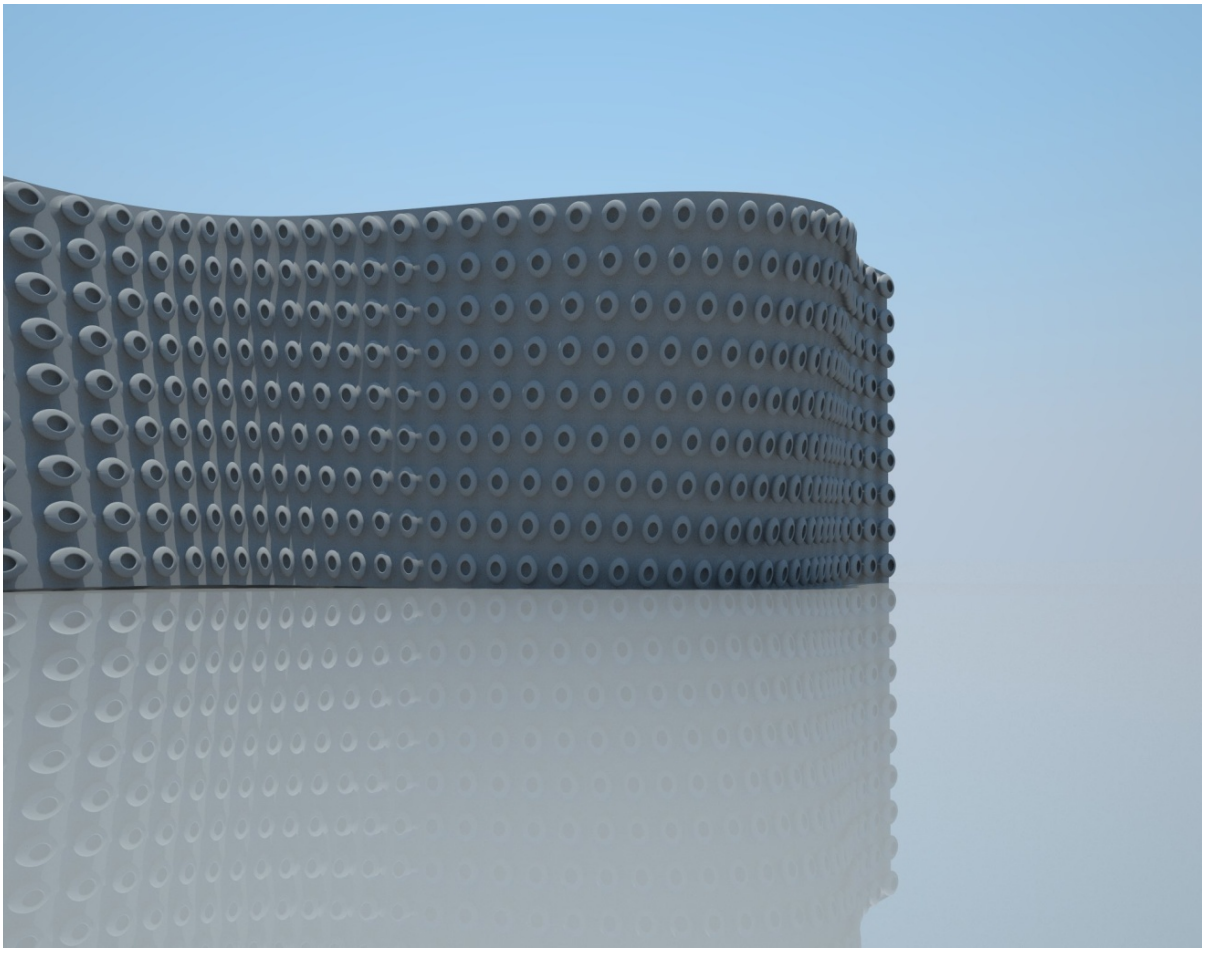


Fig.6.11. Final surface made up of our base module

6_3_Micro Level Manipulations

Although it is great to proliferate a module over a surface, it still seems a very generic way of design. It is just a 'bumpy' surface. We know that we can change the number of modules, or change the module by itself, but still the result is a generic surface and we don't have local control of our system.

Now I am thinking of making a component based system that we could apply more local control over the system and avoid designing generic surfaces which are not responding to any local, micro scale criteria.

In order to introduce the concept, let's start with a simple example and proceed towards a more practical one. We used the idea of attractors to apply local manipulations to a group of objects. Now I am thinking to apply the same method to design a component based system with local manipulations. The idea is to change the components size (in this case, their height) based on the effect of a (point) attractor.

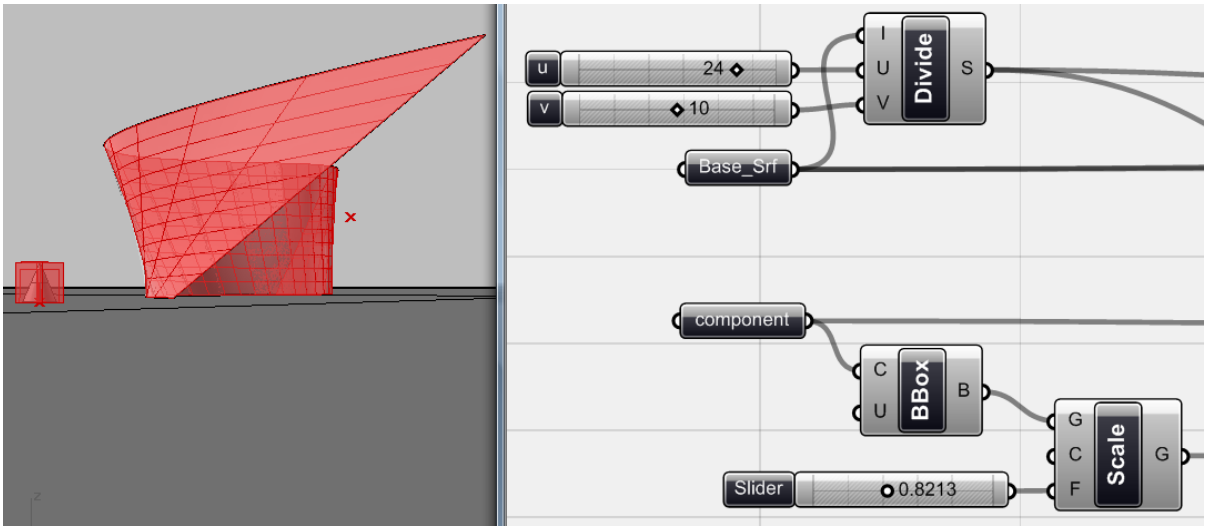


Fig.6.12. A double-curve surface introduced as <Base_Srf> and a cone which is introduced as <component> to the Grasshopper, a <divide interval^2> for surface divisions, and a <bounding box> as the reference box of the <component>. Here I used a <scale> component for my bounding box. Now if I change the size of the bounding box, I can change the size of all components on the <base_srf> because the reference box has changed.

As you have seen, the <surface box> component has the height input which asks for the height of the boxes in the given intervals. The idea is to use relative heights instead of constant one. So instead of a constant number as height, we can make a relation between the position of each box in relation to the attractor's position.

What I need is to measure the distance between each box and the attractor. Since there is no box yet, I need a point on surface at the center of each box to measure the distance.

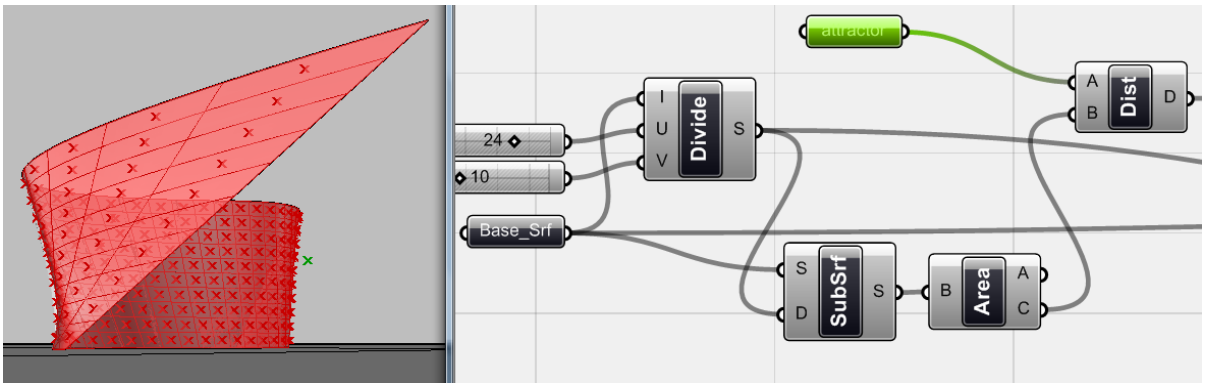


Fig.6.13. Here I used the same <divide interval^2> for an <Isotrim> component (Surface > Util > Isotrim). This component divides the surface into sub-surfaces. By these sub-surfaces I can use another component which is <BRep Area> (Surface > Analysis > BRep area) to actually use the by-product of this component that is 'Area Centroid' for each sub-surface. I measured the distance of these points (area centroids) from the <attractor> to use it as the reference for the height of the target boxes in <surface box> component.

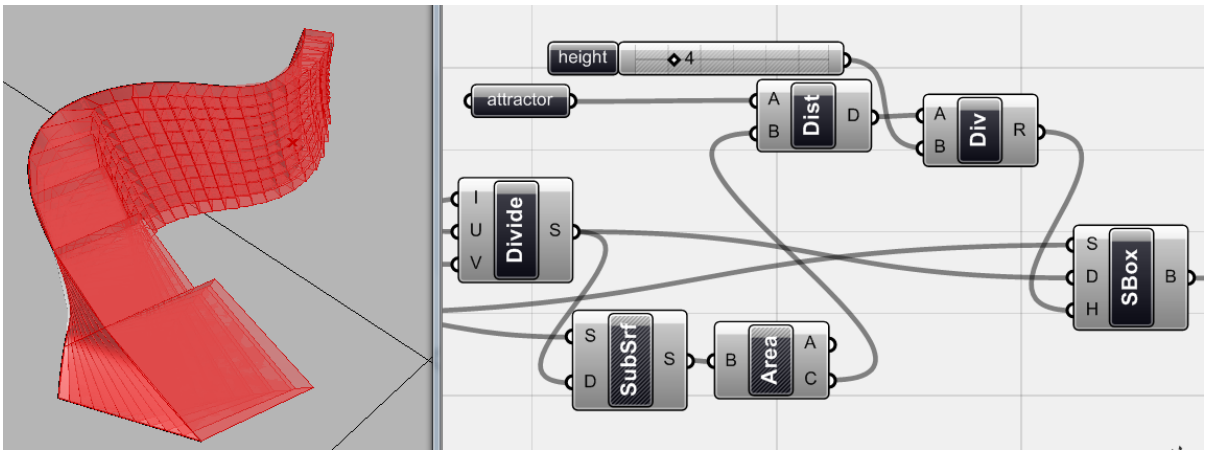


Fig.6.14. I just divided the distances by a given number to control the effect of the attractor and I used the result as 'height' to generate target boxes with <surface box> component. The surface comes from the <base_srf>, the <divide interval^2> used as surface domain and heights coming from the relation of box position and the attractor. As you see, the height of boxes now differ, based on the position of the point attractor.

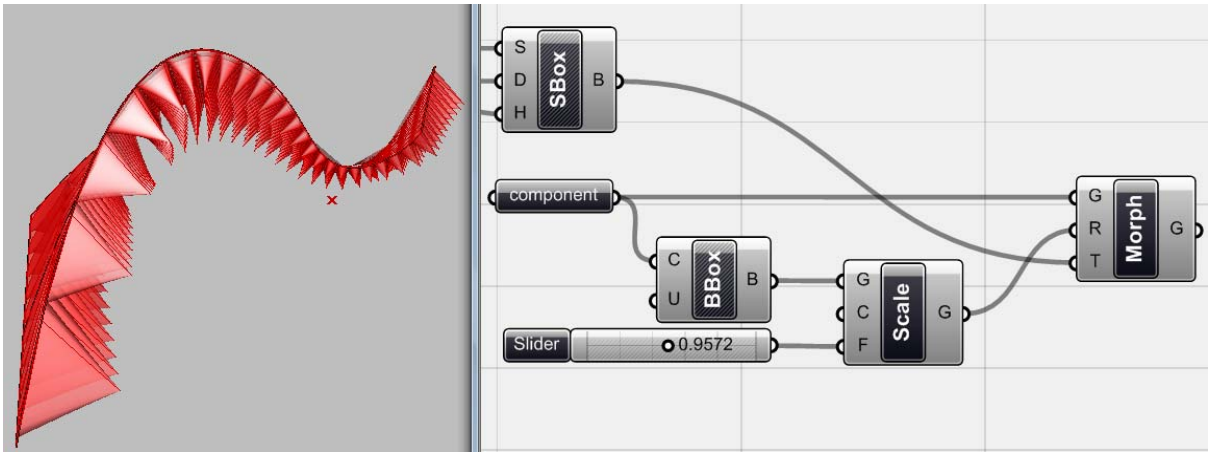


Fig.6.15. The only remaining part, connecting the <component>, <scale>d bounding box and <surface box> to a <morph box> component and generate the components over the surface. By changing the scale factor, you can change the size of the all components and like always, the position of the attractor is also manually controllable.

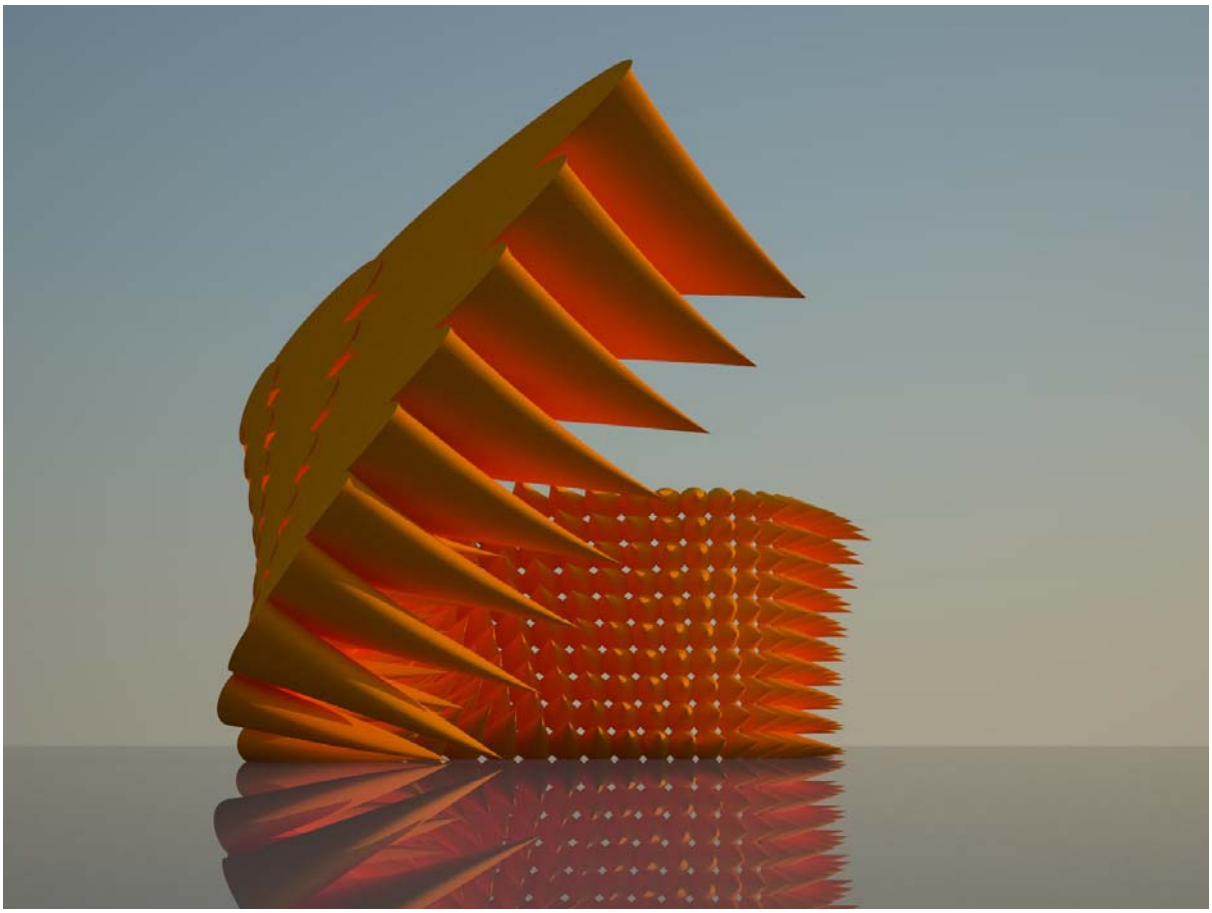


Fig.6.16. Final model.

As you see, the size of components started to accept local manipulations, based on an external property which is here a point attractor. Although the idea is a simple attractor, the result could be interesting and you could differentiate your reference boxes in so many other ways as well. Now we know that the morphing concept and panelization is not always generic. Having tested the concept, let's go for another practical experiment.

6_4_On Responsive Modulation

The idea for the next step is to modulate a given surface with control over each module, means any module of this system, has to be responsible for some certain criteria. So even more than regional differentiation of the modules, here I want to have a more specific control over my system by given criteria. These could be environmental, functional, visual or any other associative behaviour that we want our module be responsible for.

In the next example, in order to make a building's envelope more responsive to the host environment, I just wanted the system be responsive to the sun light. In your experiments it could be wind, rain or internal functions or any other criteria that you are looking for.

Here I have a surface, simply as the envelope of a building which I want to cover with two different types of components. One which is closed and does not allow the penetration of the sun light and the other has opening. These components should be proliferated over my envelope based on the main direction of the sun light at the site. I set a user defined angle to say the algorithm that for the certain degrees of sun light we should have closed components and for the others, open ones.

The Grasshopper definition does not have anything new, but it is the concept that allows us to make variation over the envelope instead of making a generic surface. Basically when the surface is free-form and it turns around and has different orientations, it also has different angles with the main sun light at each part. So based on the angle differentiation between the surface and the sun light, this variation in the components happens to the system.

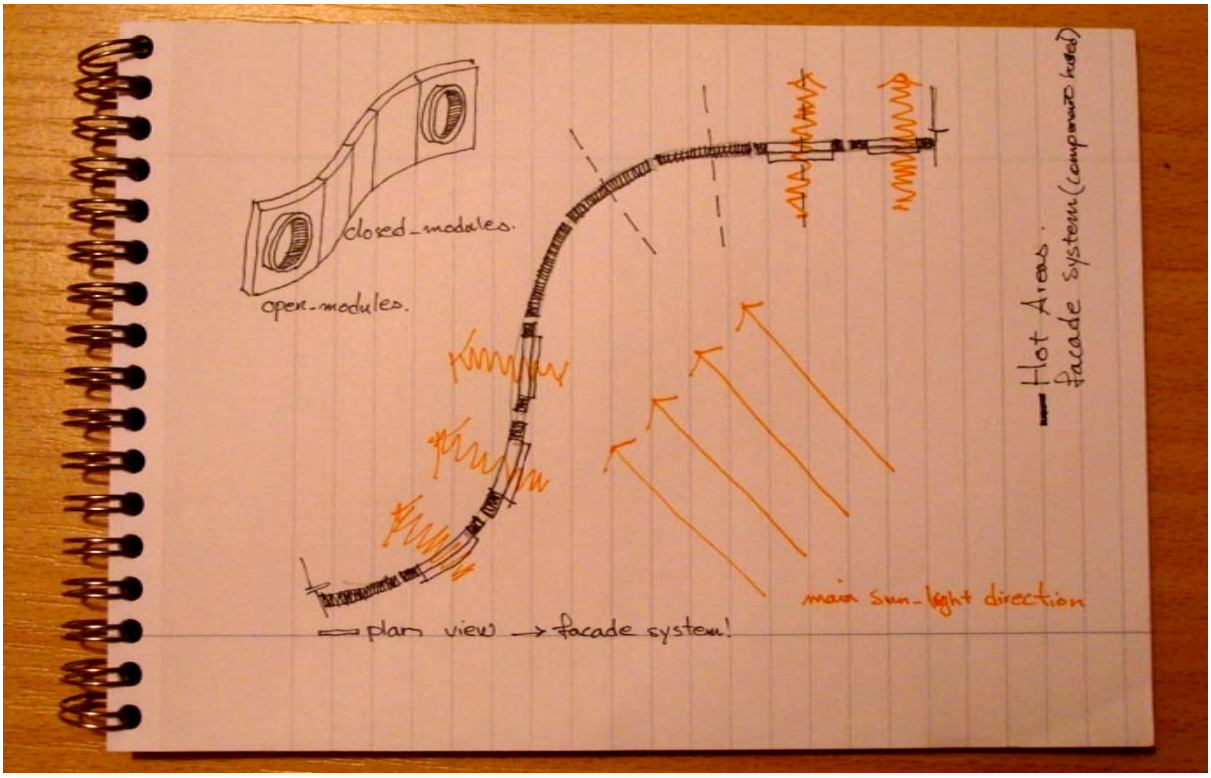


Fig.6.17.First sketches of responsive modules on façade system.

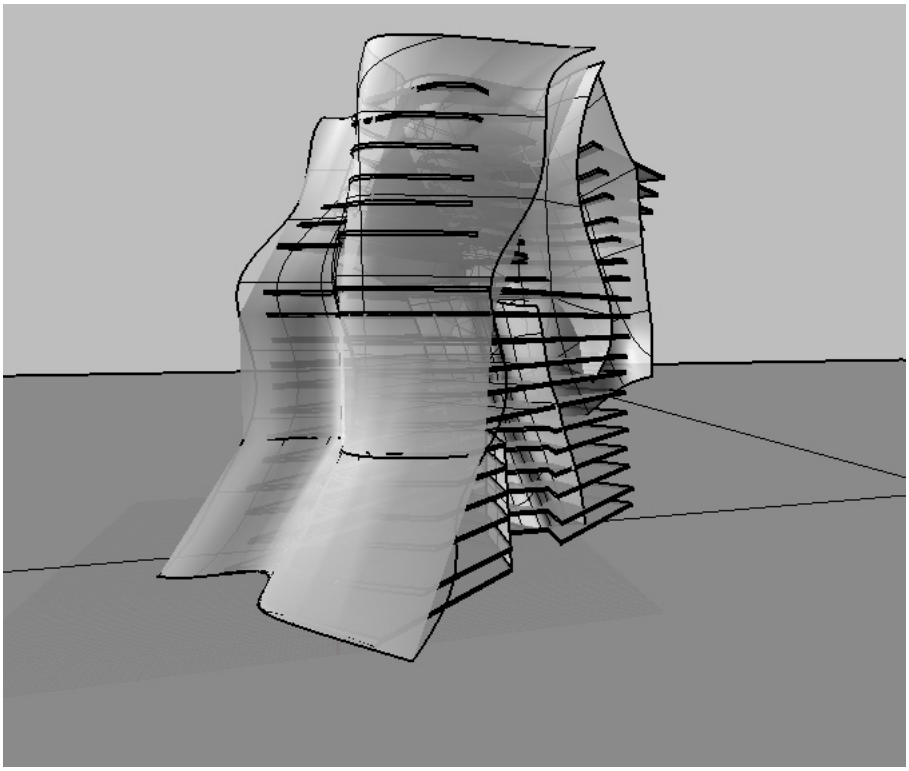


Fig.6.18. Surface of the building as envelope.

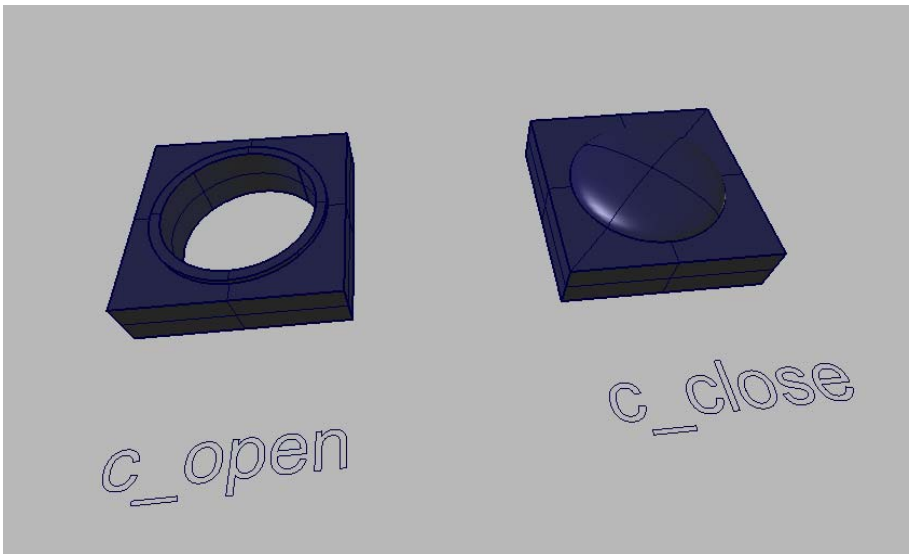


Fig.6.19. Two different types of components for building envelope.

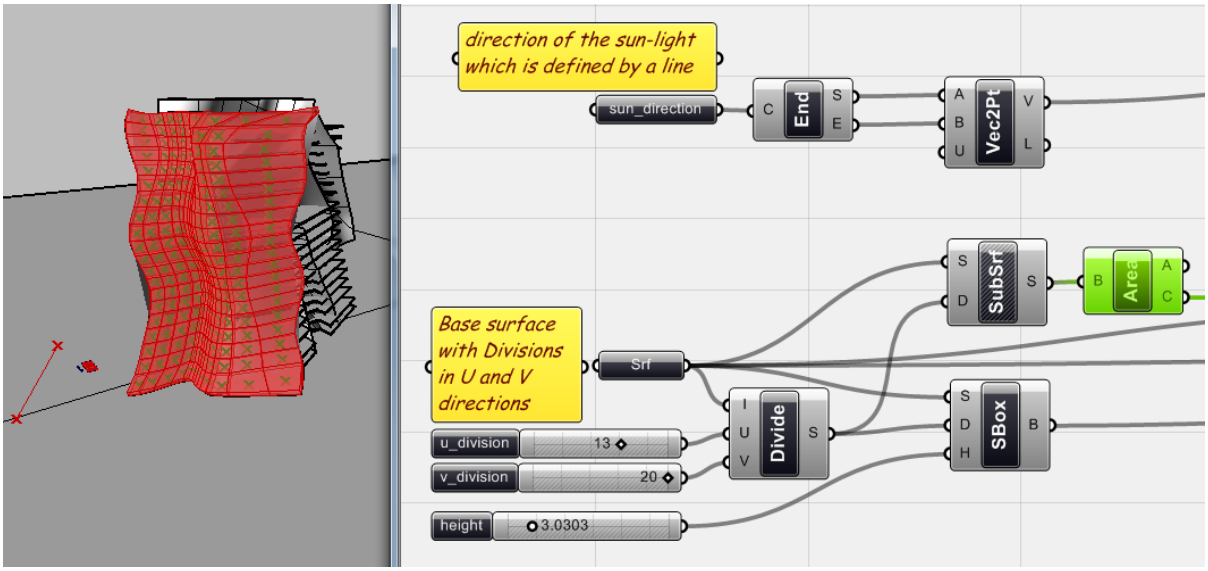


Fig.6.20. The first step is similar to the previous experiments. I introduces <surface> and I used <divide interval^2> to divide it in U and V directions and I generated target boxes by <surface box>. I also used the <isotrim> with the same intervals and I used <BRep area> to find the centroid of each area (which is selected in green). At the same time I used a <curve> component to introduce the main sun-light angle of the site and whit its <end points> I made a <vector 2pt> which specify the direction of the sun light. You can manipulate and change this curve to see the effect of sun light in different directions on components later on.

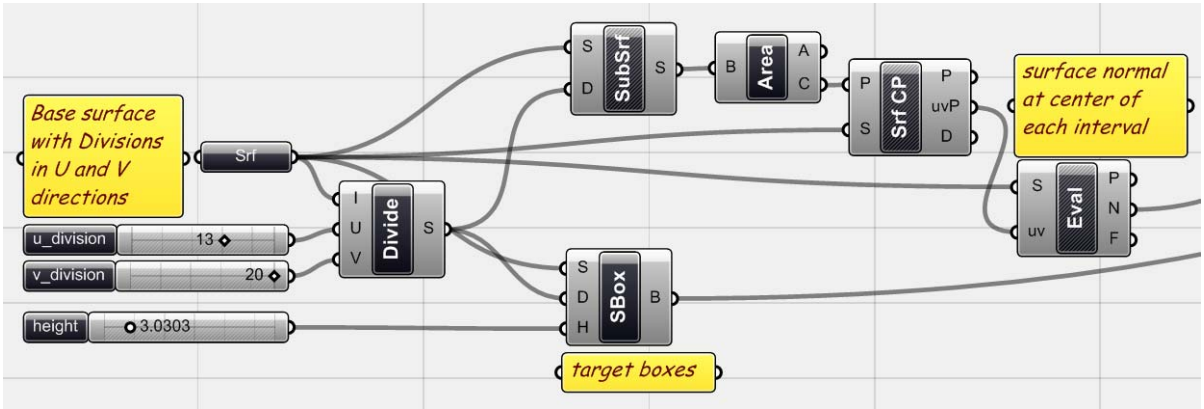


Fig.6.21. in order to evaluate the angle between the sun-light and the surface, I want to measure this angle between sun light and normals of the surface at the position of each component. So I can decide for each range of angles what sort of component would be useful. So after generating the center points, I need normals of the surface at those points. that's why I used a <surface CP> to find the closest point of each center point on the surface to get its UV parameters and use these parameters to <evaluate> the surface at that points to actually get the normals of surface at that points.

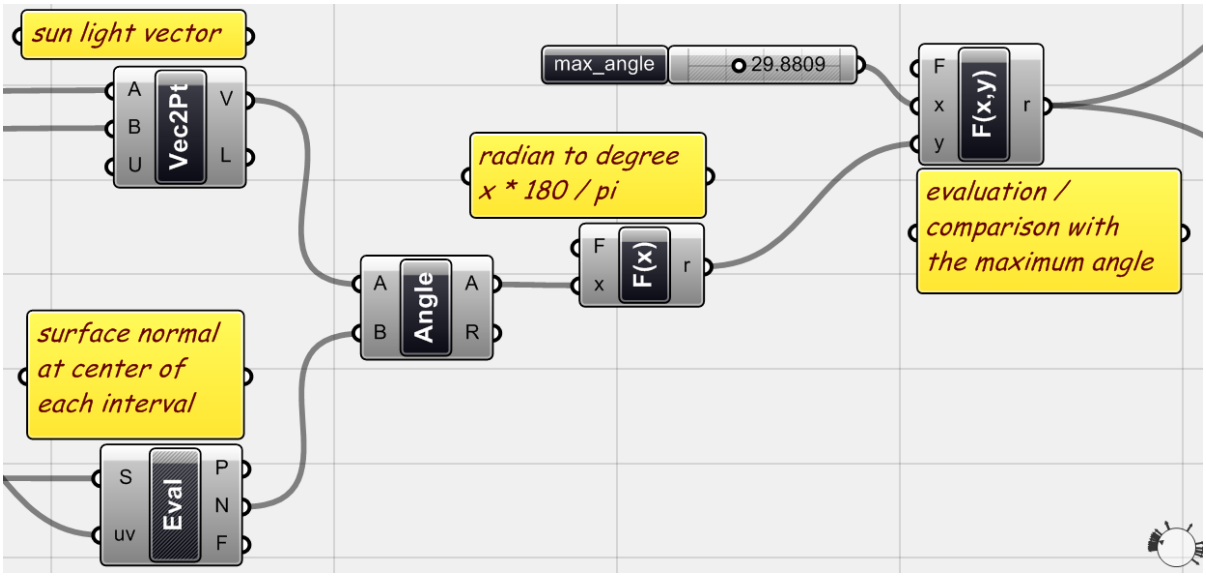


Fig.6.22. I used an <angle> component (Vector > Vector > Angle) to evaluate the angle between the sun direction and the façade. Then I converted this angle to degree and I used a <function> to see whether the angle is bigger than the <max_angle> or not. This function (x>y) gives me Boolean data, True for smaller angles and False for bigger angles.

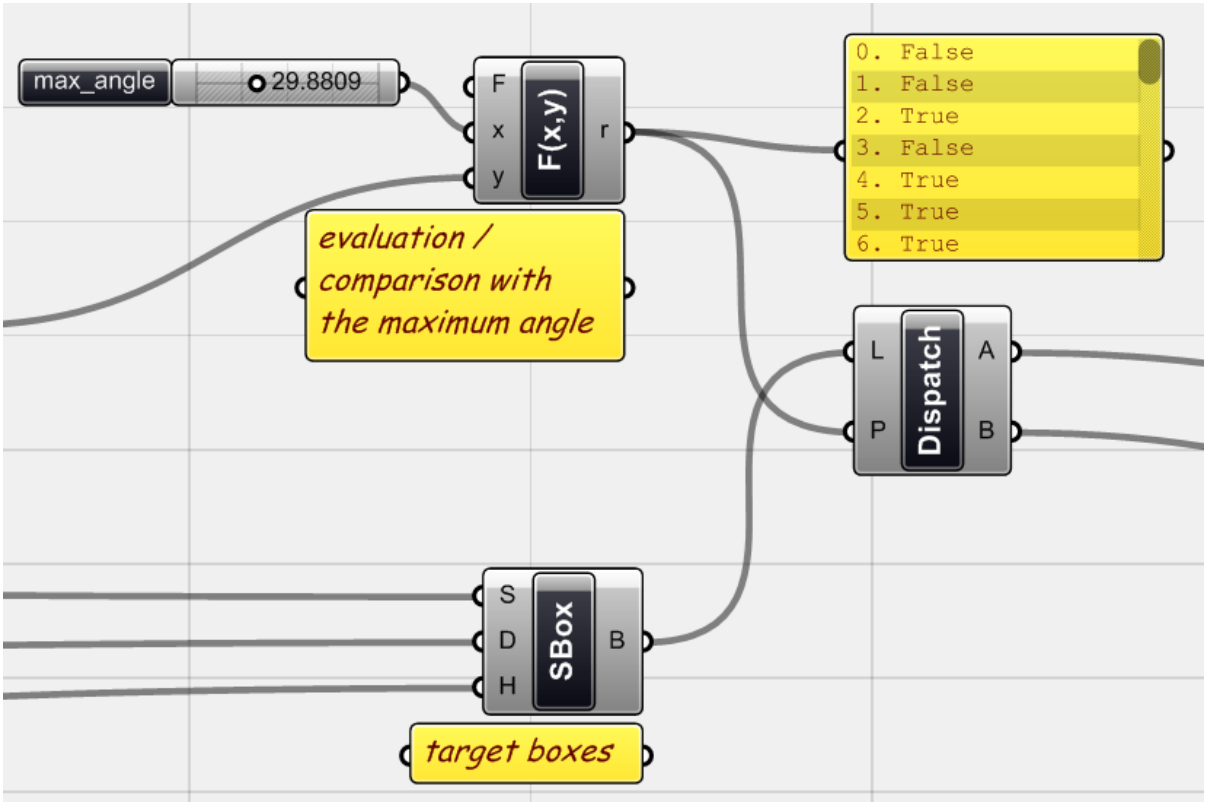


Fig.6.23. Based on the Boolean data comes from the angle comparison, I <dispatch> the data which are target boxes (I have the same amount of target box as the center points and normals). So basically I divided my target boxes in two different groups whose difference is the angle they receive the sun light.

The rest of the algorithm is simple and like what we have done before. I just need to morph my components into the target boxes, here for two different ones.

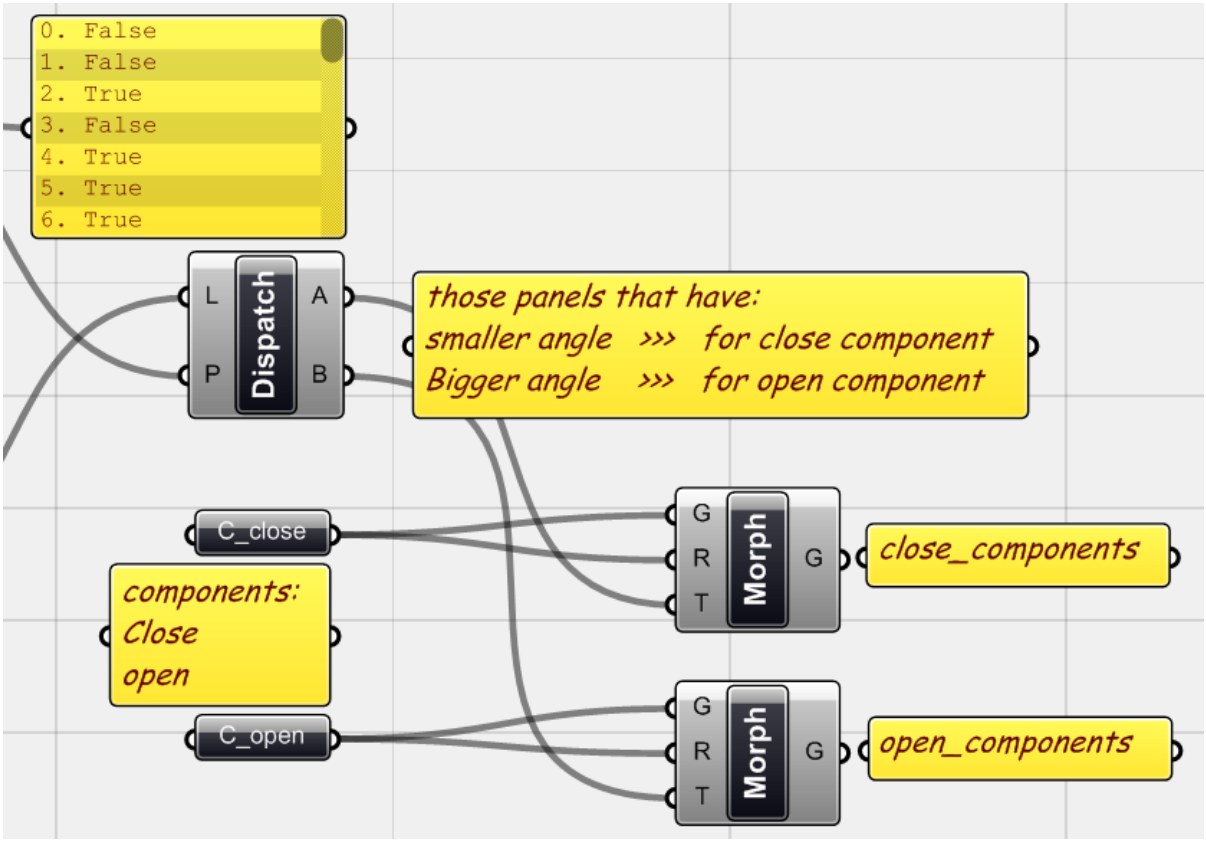
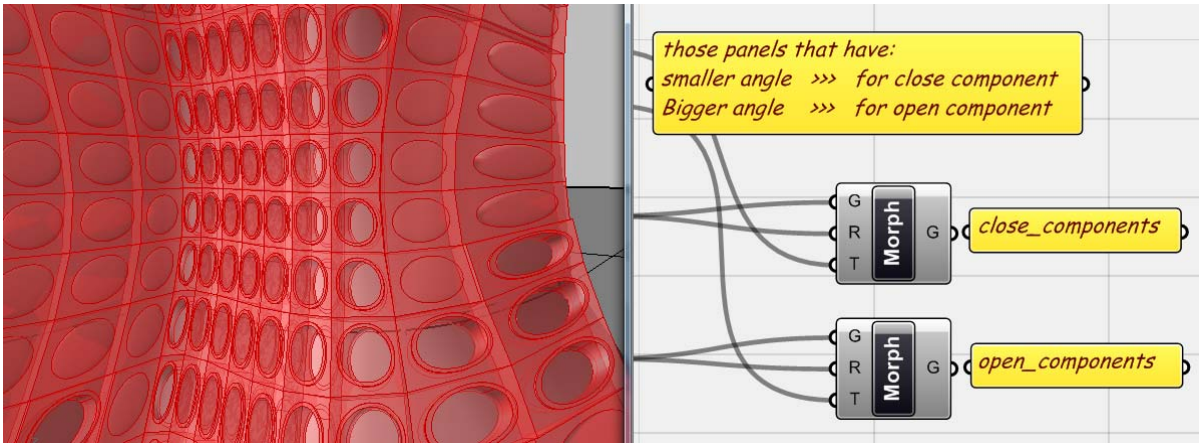


Fig.6.24. Here I introduced two different components as single geometries and I used two <morph box> components each one associated with one part of the <dispatched> data to generate <C_close> or <C_open> components over the façade.



6.25. Now if you look closer, you can see that in different parts of the façade, based on its curvature and direction, different types of components are generated.

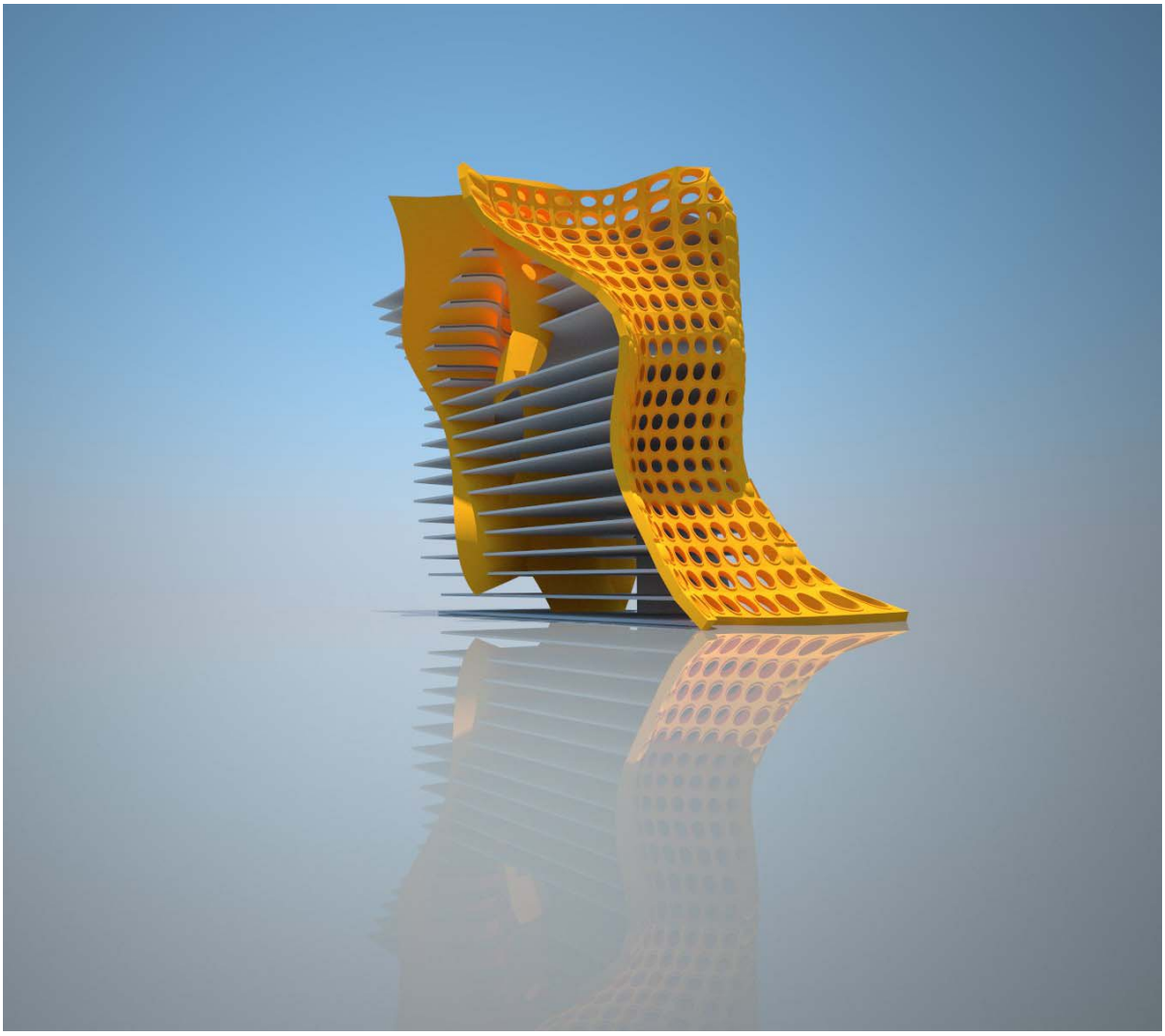
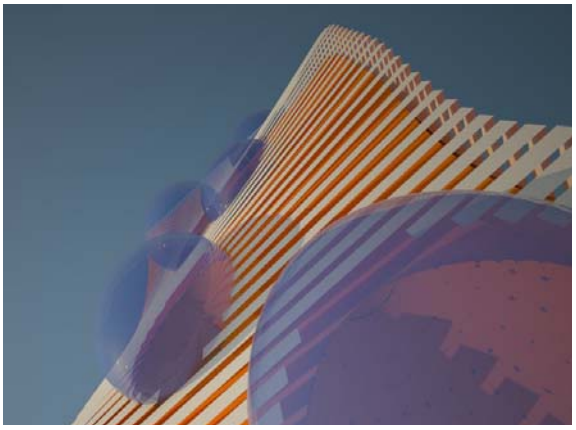


Fig.6.26. Final model. The bifurcation of the target boxes (and the components) could be more than two in the algorithm. It depends on the design and the criteria that we use.

We can think about a component based façade, in which some components are closed, and some of them are open, in which open ones have adjustable parts that orientate towards external forces, and have apertures that adjust themselves to the internal functions of the building and so on and so forth. You see that the idea is to have module based control over the system. We still have the global control (form) and regional control (affecting components height or scale regionally) over the system as well.

Chapter 7_NURBS Surface and Meshes



Chapter 7_NURBS Surface and Meshes

We have had some experiments with surfaces in the previous chapters. We used Loft and Pipe to generate some surfaces. We also used free form surface and some surface analysis accordingly. Usually by surfaces, we mostly mean Free-Form surfaces which we generate them by curves or sometimes bunch of points. So usually generating surfaces depends on the curves that we provide for our surface geometries. There are multiple surface components in the Grasshopper and if you have a little bit of experience in working with Rhino you should know how to generate your surface geometries by them.

Surface geometries seems to be the final products in our design, like facades, walls etc. and that's why we need lots of effort to generate the data like points and curves that we need as the base geometries and finally surfaces. Here I decided to design a very simple schematic building just to indicate that the multiple surface components in the Grasshopper have the potential to generate different design products by very simple basic constructs.

7_1_Parametric NURBS Surfaces

In the areas of Docklands of Thames in London, close to the Canary Wharf, where the London's high rises have the chance to live, there are some potential to build some towers. I assumed that we can propose one together, and this design could be very simple and schematic, here just to test some of the basic ideas of working with free-form surfaces. Let's have a look at the area.

Fig.7.1. Arial view, Canary Wharf, London (image: www.maps.live.com, Microsoft Virtual Earth).

The site that I have chosen to design my project is in the bank of Thames, with a very good view to the river and close to the entrance square to the centeral part of Canary Wharf (Westferry Road).

I don't want to go through the site specifics so let's just have a look at where I am going to propose my tower and continue with the geometrical issues.

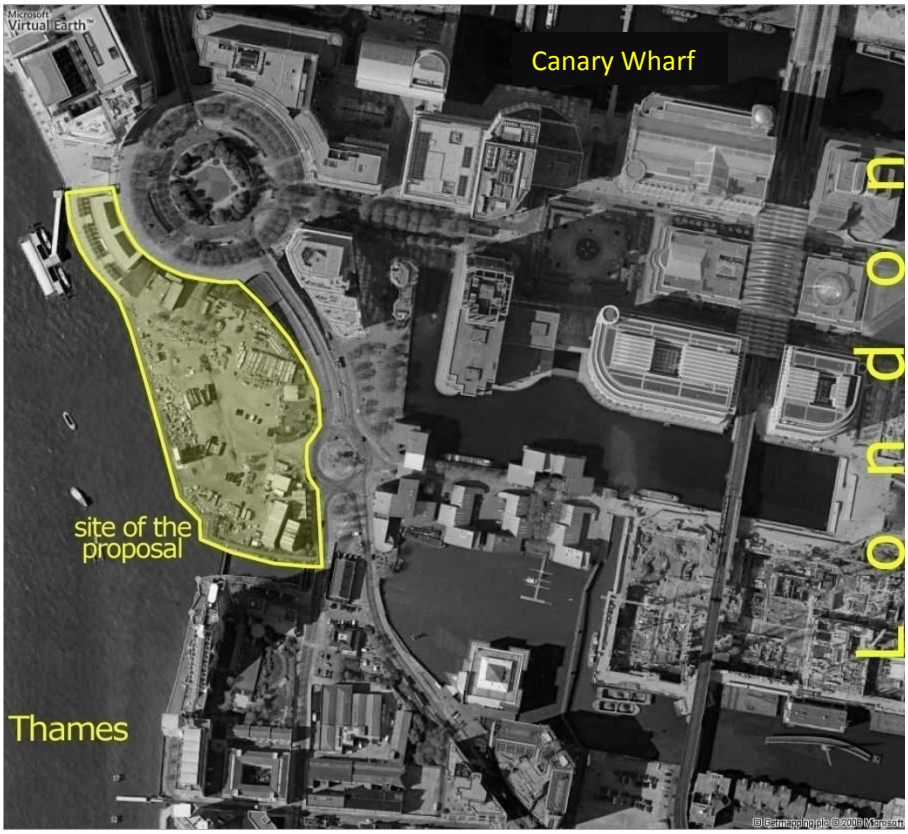


Fig.7.2. Site of the proposed tower.

Manual drawings

There are multiple ways to start this sketch. I can draw the ground floor and copy it and start to add detail like what we have done for Swiss Re. Another way is to draw some boundary curves for the tower and use this curves as base geometry for the building volume and then add more details to complete the sketch. I would prefer the second way since we already know about the first one.

I am going to draw the boundary curves in Rhino. I can draw these lines in Grasshopper but because I want them to match the geometry of the site so I need some data from the site which I can get in this way.

I have a vague idea in mind. My tower has some linear elements in façade which are generic, but I also want to have some hollow spaces between the outside and inside, positioned on the façade.

I also want to design a public space close to the river and connected to the tower with the same elements as façade, continuous from tower towards the river bank.

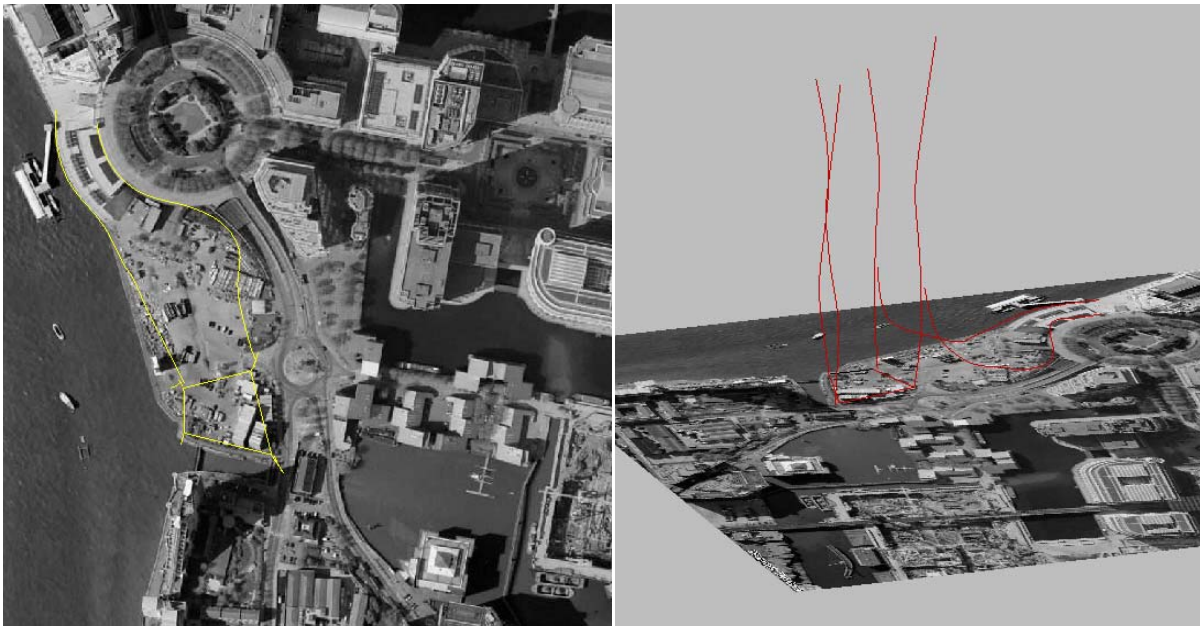


Fig.7.3. The basic lines of the tower associated with the site.

As you see in the Figure.7.3 I have drawn my base curves manually in Rhino. These curves correspond to the site specifics, height limitations, site's shape and borders, etc, etc. four curves drawn for the main building and another two curves as the borders of the public space, which started from the earth level and then went up to be parallel to the tower edges. These curves are very general. You could draw whatever you like and go for the rest of the process.

Basic façade elements

Fig.7.4. For the first step I imported these four corner curves into Grasshopper by <curve> components and then I used <divide curve> to divide these curves into <N_div> parts.

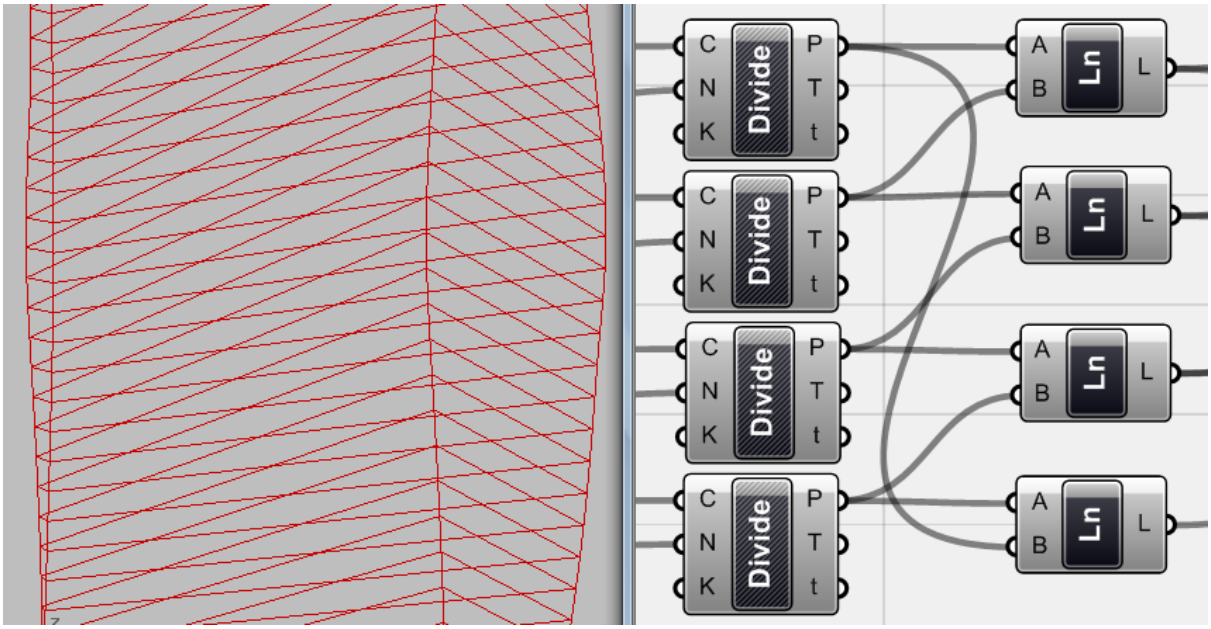


Fig.7.5. Now I have the same amount of division points on each curve and I draw <line> between each curve's points and the next one, so basically I generated <N_div> number of quads like floor edges.

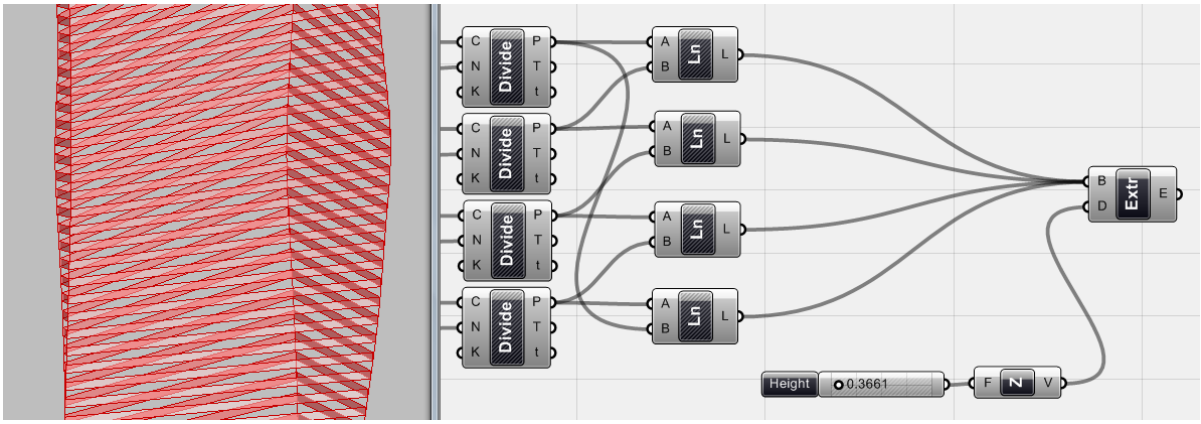


Fig.7.6. By connecting these <line>s to <extrude> component (Surface > Freeform > Extrude) and extrusion in Z direction, I generated surfaces across façade which reminds me Mies high-rises but in a differentiated way!

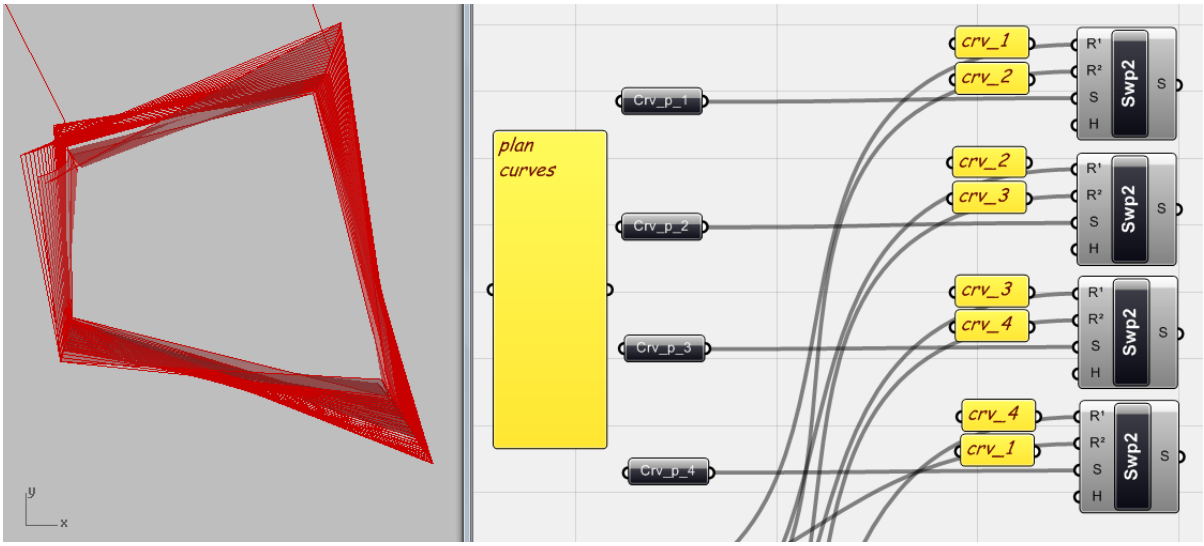


Fig.7.7. There are four straight lines that connect the four corners of the tower in plan. I introduced these lines by another four <curve> components and I generated four surfaces as general façade surfaces that could be glass. Here because I have the planner section curve and two curves that define the boundary of the façade on each face, I used a <Sweep 2> component to use Rails for making a surface by a section curve. So basically after I attached each <Crv_p_n> to the <Sweep2> component as the section curve, I attached the corner curves as the rails to make each face's surface. For each section curve in the plan, I used two edge curves that start from its end points. So the order is <crv_1> and <crv_2> for the <crv_p_1> and so on.

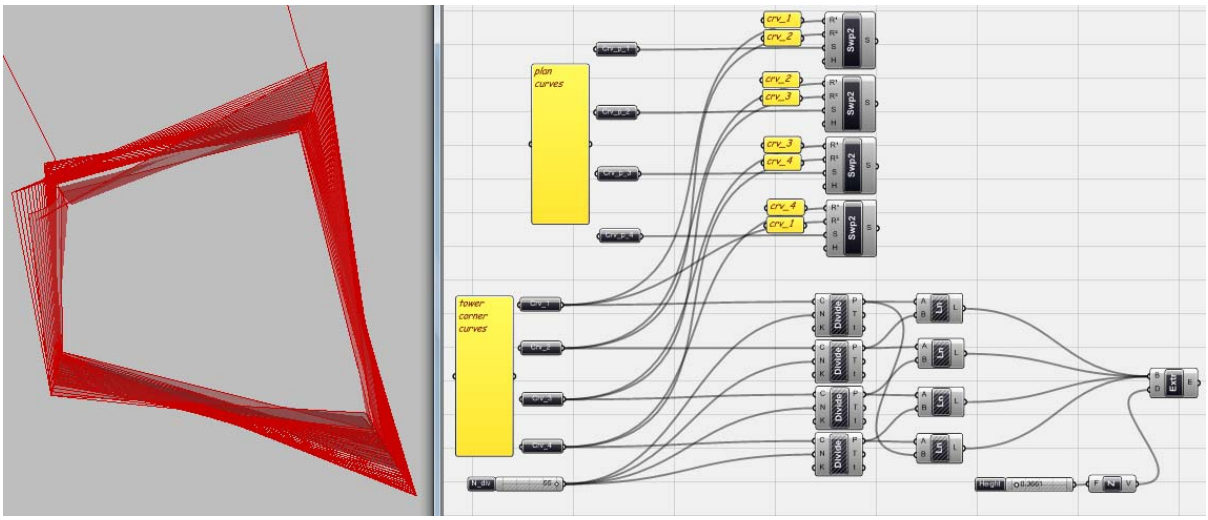


Fig.7.8. Using tower's edge curves as Rails to make façade surfaces behind the ribs shaped façade elements that we made before.

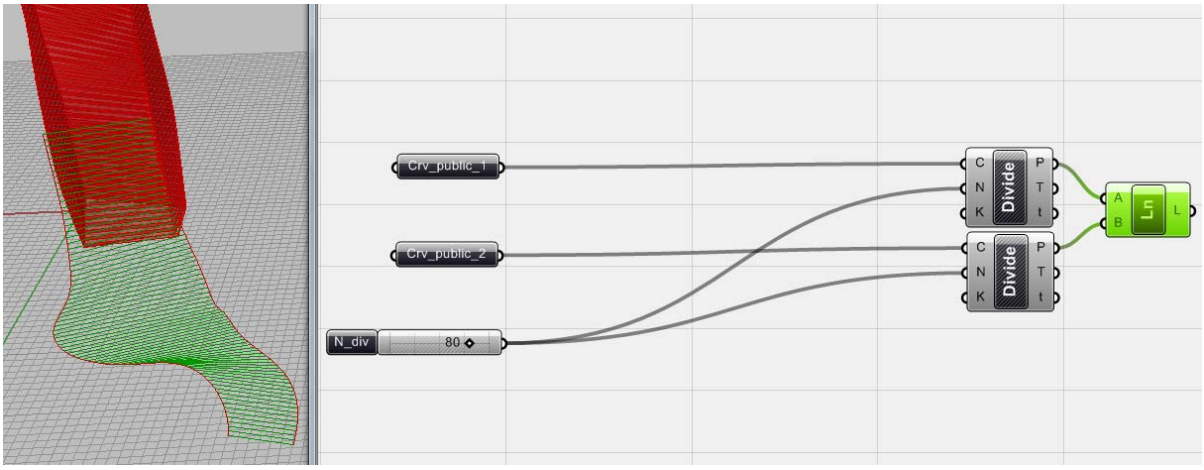


Fig.7.9. With the same method, I just divided another two curves that I have drawn for the public space and I attached them to a line component to generate some lines as the base geometry for extrusion. I will attach it to the same <extrude> component as the façade (don't forget to hold the Shift key when you want to add a new component to the input of an already existing component).

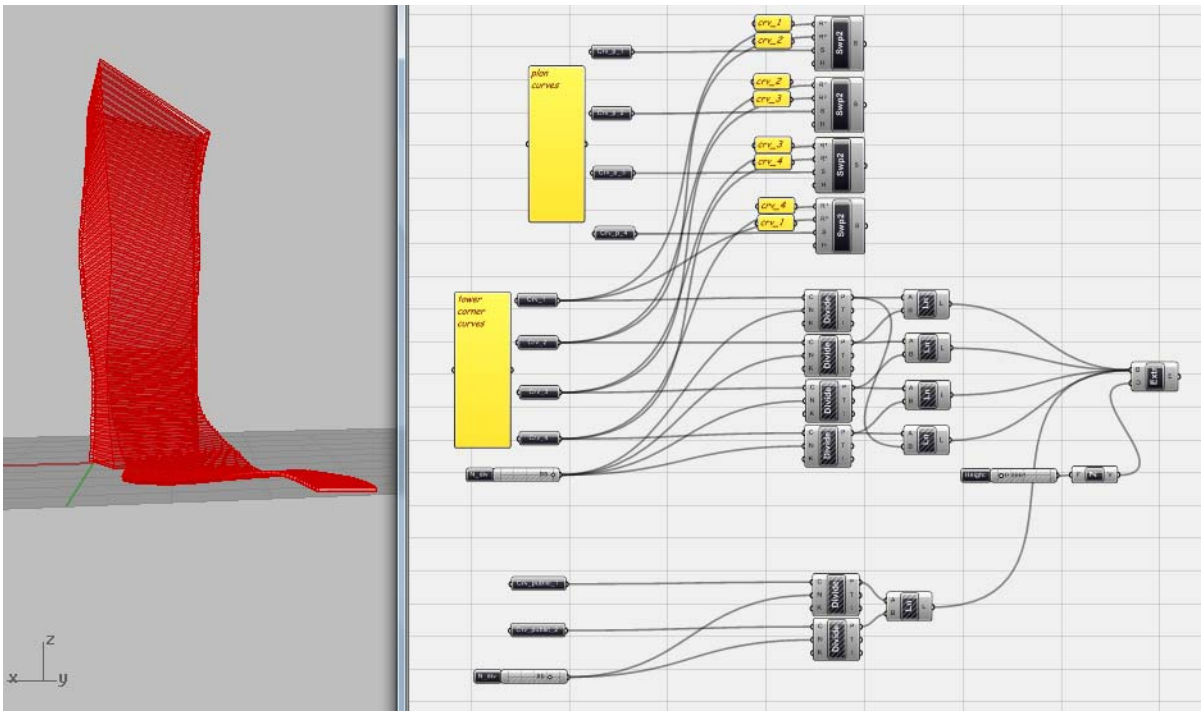


Fig.7.10. Now you can alter the corner curves if you like, the number of divisions and the height of extrusion to get the best schematic design.

Local changes

Up to this point you already know that how to design this tower with differentiation and gradual changes across the façade elements. Here I designed very generic surface elements and the only parameters that make the global differentiations are edge curves of the tower. Now I want to add some elements to make some local changes in these generic elements to show how you can change the behaviour of these elements you designed.

As you see, the main element in our case is some lines that I generated between division points of the edge curves and then I extruded them to make the façade elements. There are many ways that you can change and manipulate these lines and then extrude them, so you can get different forms outside, and different spaces inside.

Here I decided to add some hollow spaces between these elements and make the façade with these general linear elements with locally omitted parts as the combination of general and local design effects.

To add these hollow spaces, I want to add them manually as solids to the tower and then subtract them from the already existing forms. As an example I want to add some very simple ellipsoids to the façade, and randomly distribute them alongside the its face. These ellipsoids could be added by a certain spatial criteria but here the aim is just to explore modelling properties and techniques and I don't want to go further in design issues.

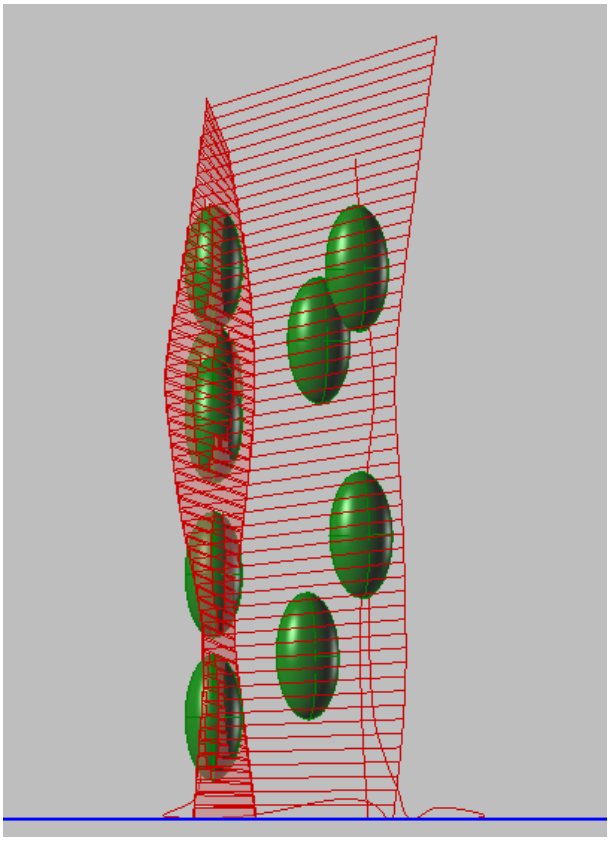


Fig.7.11. As you can see, I randomly added some ellipsoids o the tower that could be special hollow spaces in the building. I will do it for all surfaces of the tower.

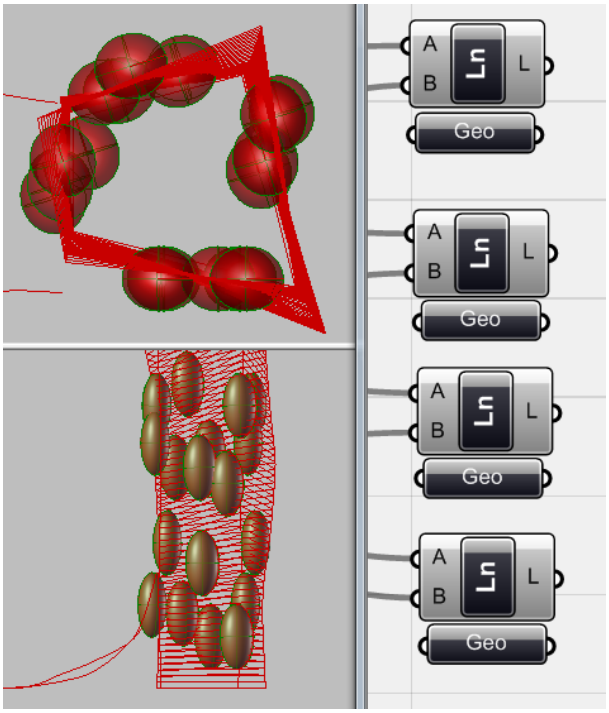


Fig.7.12. after distributing all ellipsoids alongside the façade surfaces now I import them to the Grasshopper by <Geometry> components and for each side of the façade I use one <geometry> component and introduce them as 'Set multiple Geometries'. So each side of the façade has one group of ellipsoid geometries.

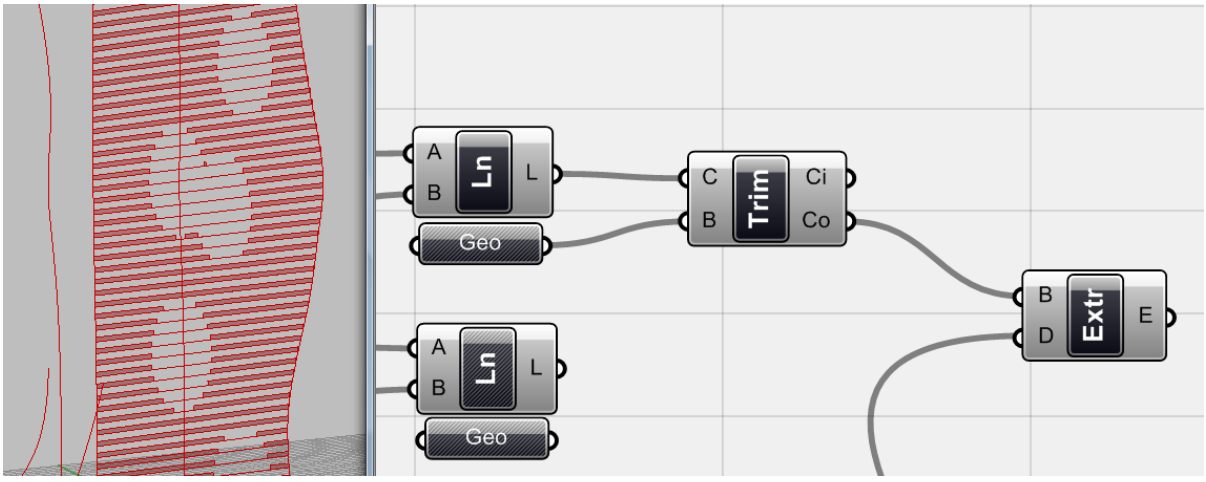


Fig.7.13. By using a <Trim with BReps> component (Curve > Util > Trim with BReps) I can trim these lines across the façade by these randomly distributed ellipsoids. The output of the component are curves inside the BRep or outside it and I use the outside curves to extrude with the same <extrude> component that I generated before. I hide all geometries in Rhino and uncheck the preview in Grasshopper to see the result better. I will do it for all faces and I will connect the output curves to the extrude component.

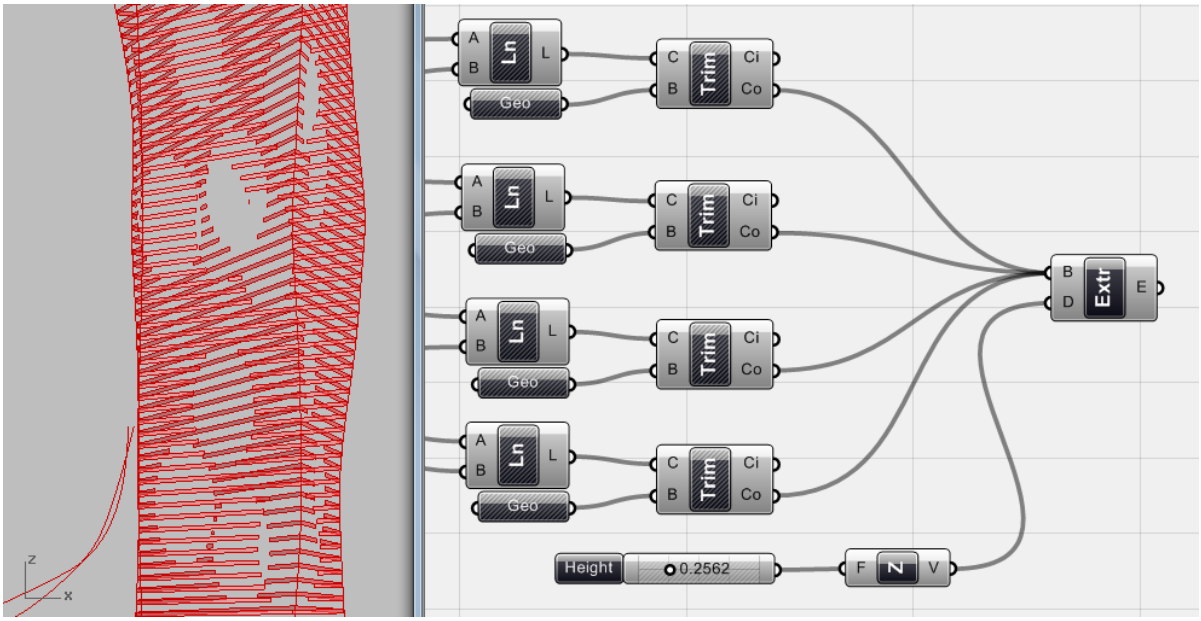


Fig.7.14. Now the facade elements are all have their trimmed ellipsoids inside.

The next step is to add these hollow spaces to the façade surface as well. The process is simple. I just need to find the difference between these ellipsoid geometries and façade surfaces. The only point is that the direction of the normal of the surface affects the 'difference' command so you should be aware of that and if needed, change this direction.

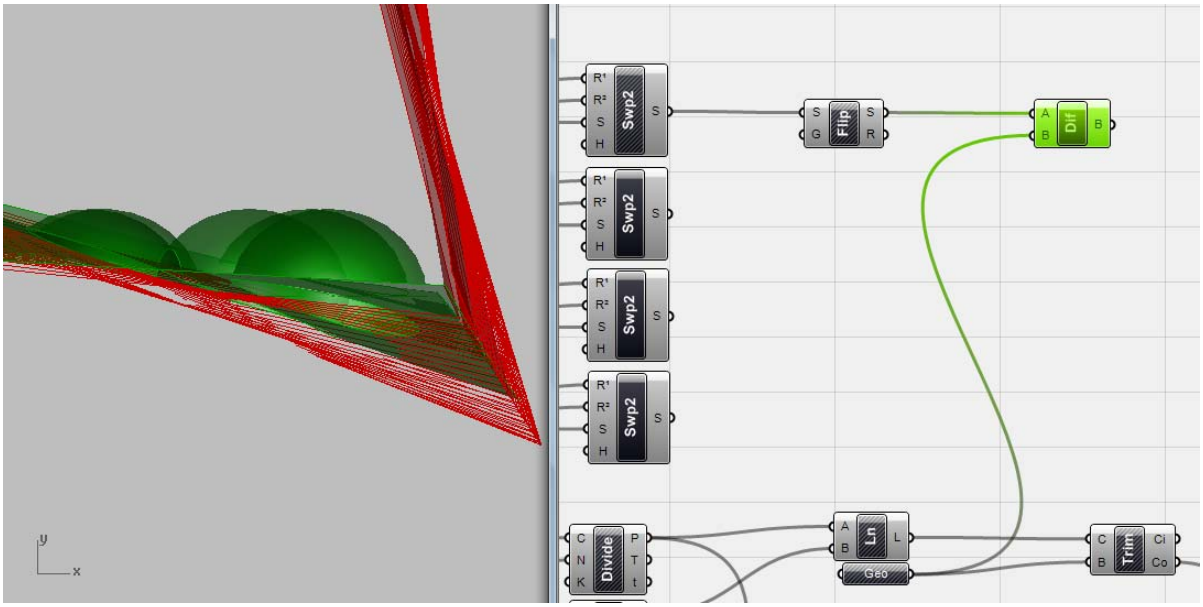


Fig.7.15. The <Solid difference> component (Intersect > Boolean > Solid difference) takes two different solids and performs solid difference on them. So I attached the façade surface and hollow geometries to find the difference of them. As I told you here I also used <Flip> component (Surface > Util > Flip) to flip the Normal of the surface and then attach it to the <Solid difference> component otherwise the hollow space would face the inside of the tower. I will do it for all surfaces of the tower.

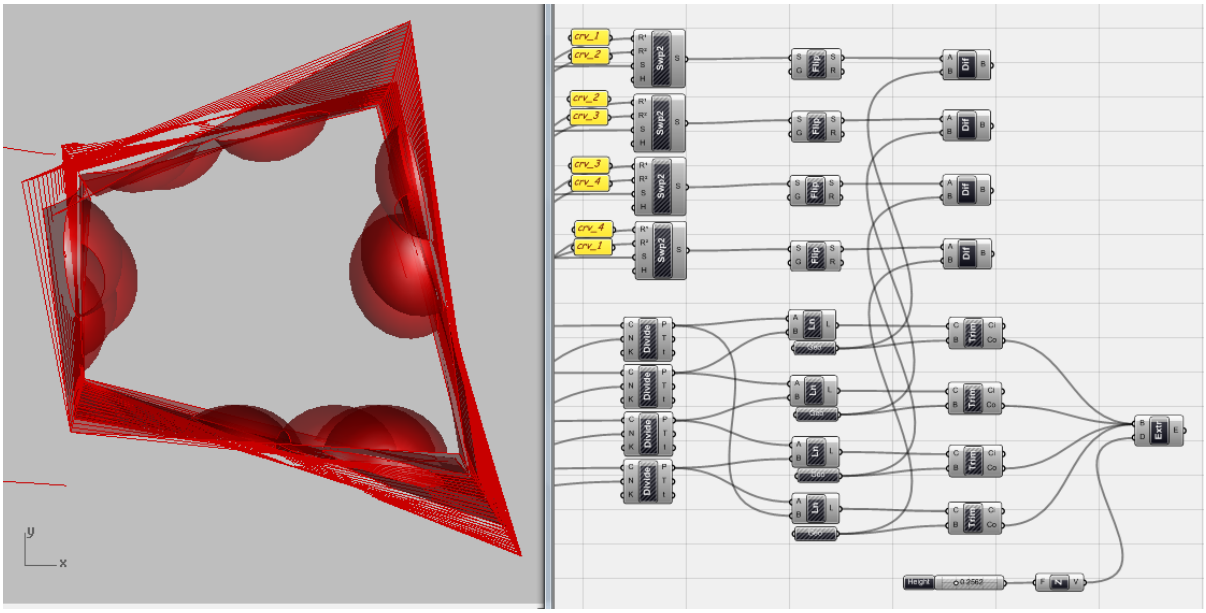
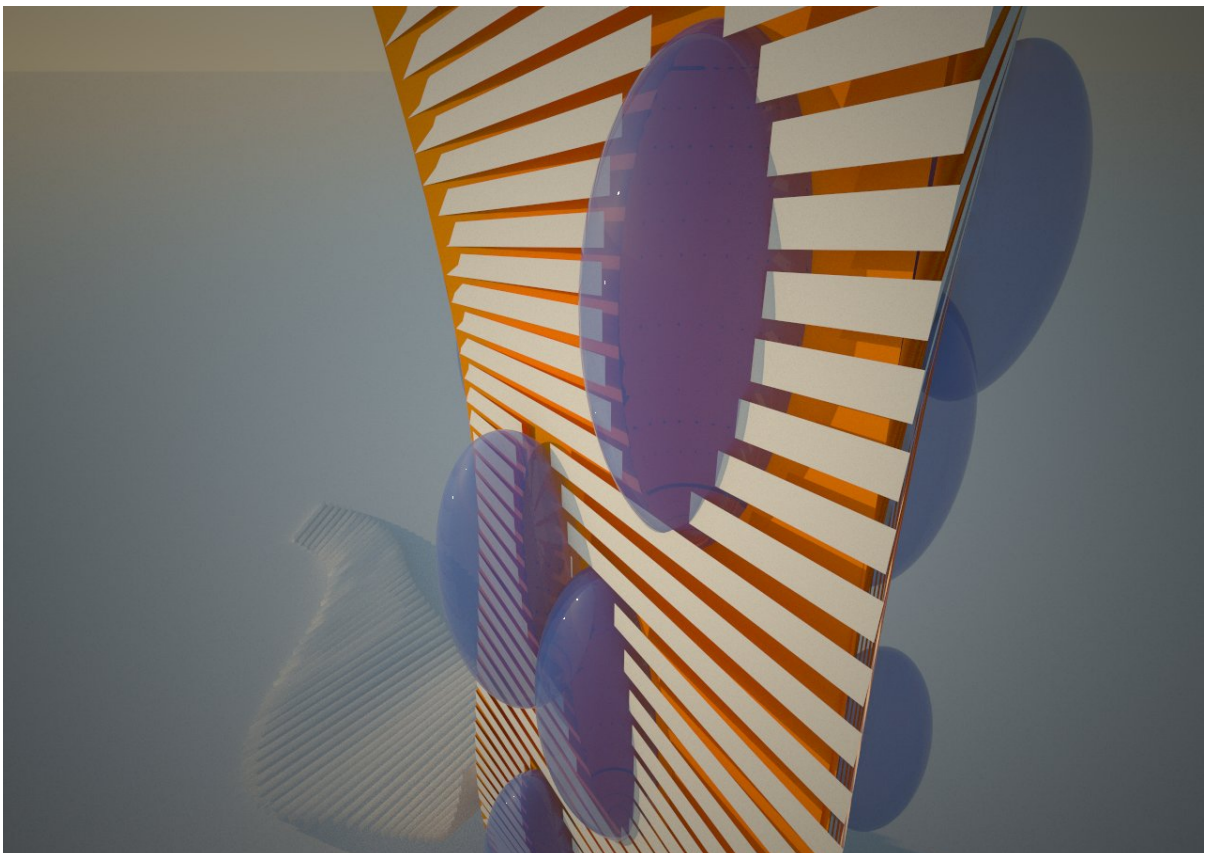


Fig.7.16. All trimmed façade surfaces.

Now you only need to burn your geometries, cap the tower and render it to complete your sketch. Again I should mention here that this was just an example to show that how you can generate series of elements and differentiate them in global and local scale with surface components.



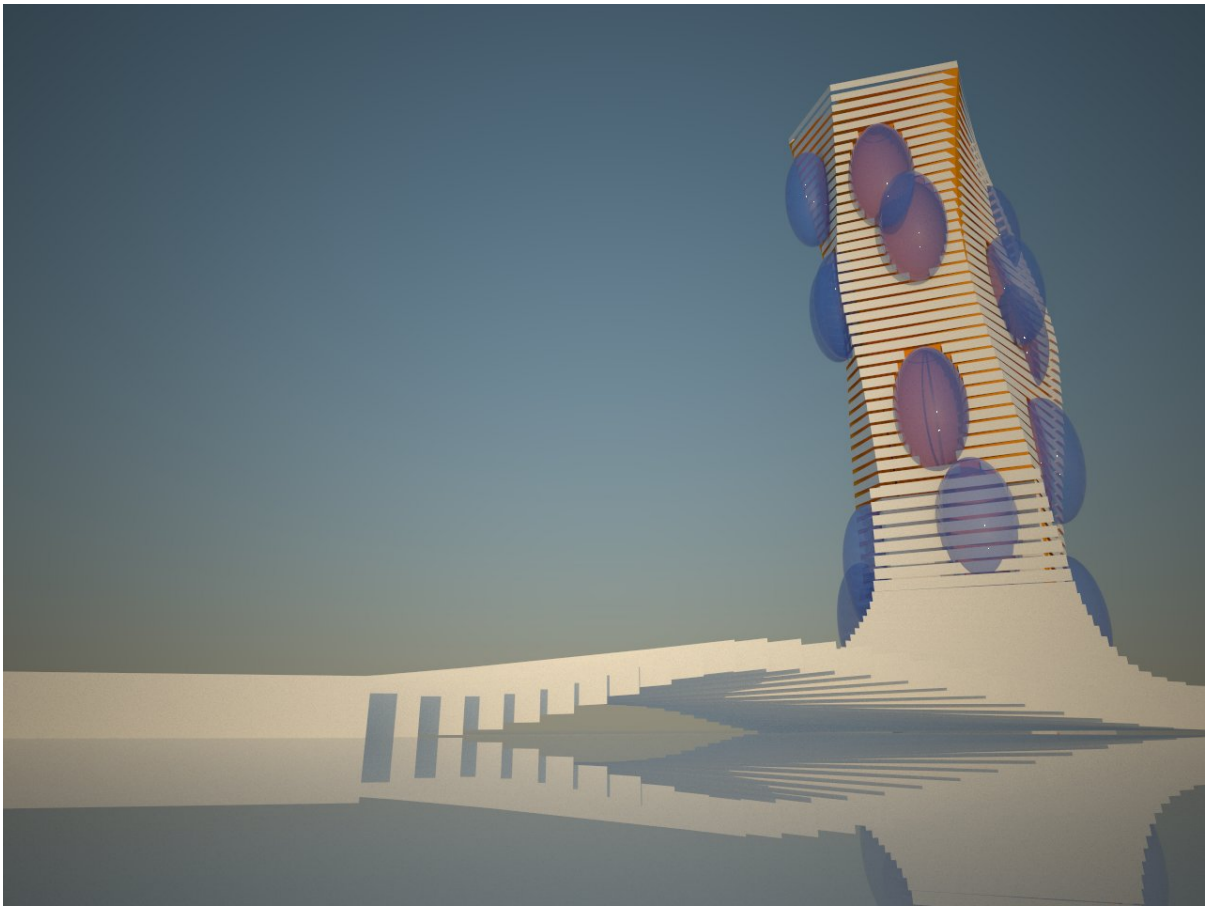


Fig.7.17.a/b. Final model.

7_2_Mesh vs. NURBS

Up to now we have used different components that worked with the NURBS surfaces. But as mentioned before there are other types of surfaces which are useful in other contexts. It is not always the smooth beauty of the NURBS that we aimed for, but we might need more precise control, easier processing or simpler equations. Beside the classical surface types of revolution, ruled or pipes, we have different free form surfaces like Besier or B-Splines. But here I am going to talk a little bit about meshes which are quiet different types of surfaces.

Meshes are another type of free-form surfaces but made up of small parts (faces) and accumulation of these small parts makes the whole surface. So there is no internal, hidden mathematical function that generates the shape of the surface, but these faces define the shape of the surface all together.

If we look at a mesh, first we see its faces. Faces could be triangle, quadrant or hexagon. By looking closer we can see a grid of points which make these faces. These points are basic elements of the mesh surface. Any tiny group of these points (for example any three in triangular mesh) make a face with which the whole geometry become surface. These points are connected together by straight lines.

There are two important issues about meshes: position of these points and the connectivity of these points. The position of the points related to the geometry of the mesh and the connectivity of the points related to the topology.

7_2_1_Geometry and Topology

While the geometry deals with the position of the stuff !! in the space, topology deals with their relations. Mathematically speaking, topology is a property of the object that transformation and deformation cannot change it. So for instance circle and ellipse are topologically the same and they have only geometrical difference. Have a look at Figure 7.20. As you 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 the displacement of one point. But in the second image, the geometry of points is the same but the connectivity is different and they are not topologically equivalent.

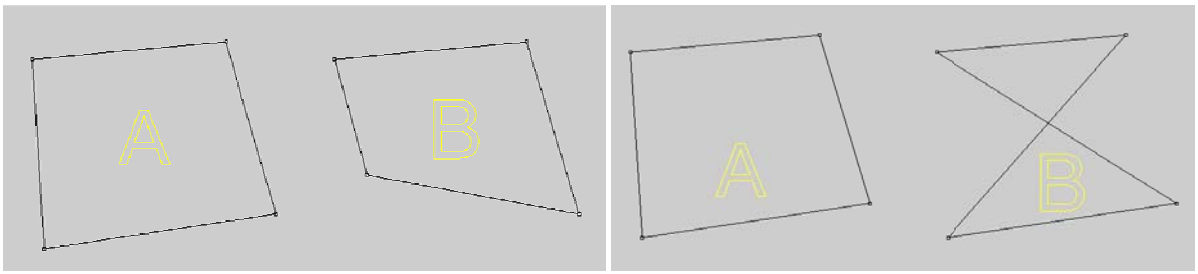


Fig.7.20. Topology and Geometry.

The idea of topology is very important in meshes. Any face in the mesh object has some corner points and these corner points are connected to each other with an order in a same way for all faces of the mesh object. So we can apply any transformation to a mesh object and displace the vertices of the mesh in the space even non-uniformly, but the connectivity of the mesh vertices should be preserved to remain the faces otherwise it collapses.

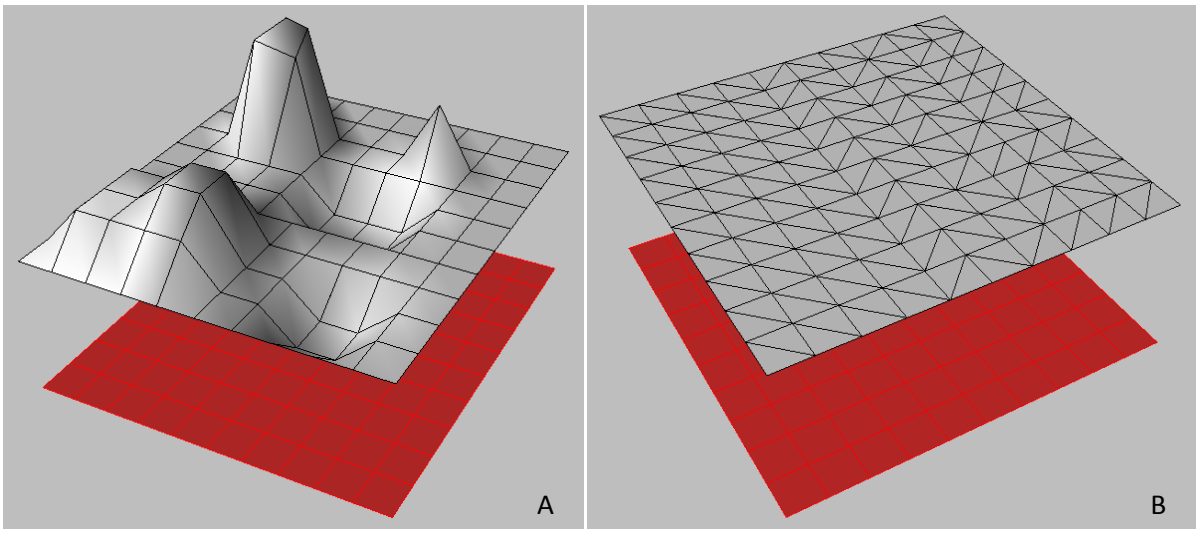


Fig.7.21. A: Both red and grey surfaces are meshes with the same amount of faces and vertices, but in the gray one, the vertices are just displaced, make another geometrical configuration of the mesh, but the connectivity of the mesh object is not changed and both surfaces are topologically the same.

B: Two Mesh surfaces are geometrically the same and have the same amount of vertices but connectivity of vertices are different, faces are triangular in the gray one but quad in the red, make different topology for them.

Knowing the importance of topological aspects of mesh objects, they are powerful geometries while we have bunch of points and we need a surface type to represent them as a continuous space. Different types of algorithms that working with points could be applied to the mesh geometry since we save the topology of the mesh. For instance, using finite element analysis or specific applications like dynamic relaxation, and particle systems, it is easier to work with meshes than other types of surfaces since the function can work with mesh vertices.

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

There are multiple components that dealing with the mesh objects in 'mesh' tab in Grasshopper. Let's start a mesh from scratch and then try to grow our knowledge.

7_3_On Particle Systems

I have a group of points and I want to make a surface by these points. In this example the group of points is simplified in a grid structure. To discuss the basic concept of the 'particle systems' here I will try to project this idea on a mesh geometry in a simple example.

I am thinking of a vertical grid of points that represent the basic parameters of a surface which is being affected by an imaginary wind pressure. I want to displace the points by this wind factor (or any force that has a vector) and represent the resultant deformed surface. Basically by changing the wind factor, we can see how the surface changes.

(Actually I want to deal with a surface geometry and an extra force that applies to this surface, with each point as a particle separately, and observe the result with changes of the main force)

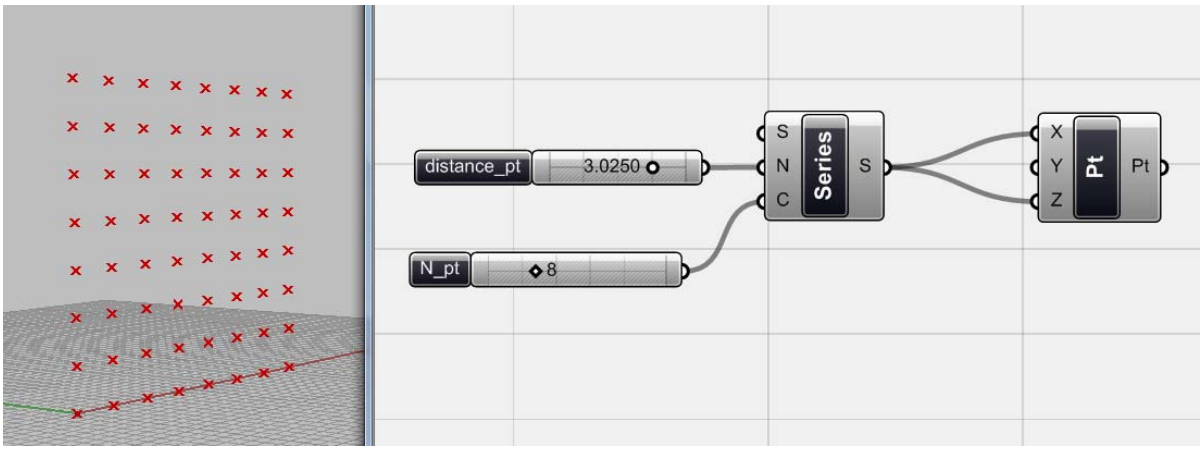


Fig.7.22. The first step is simple. By using a <series> component with controlled number of points <N_pt>, and distance between them <distance_pt> I generated a grid of cross referenced <point>s.

The pressure of the imaginary wind force, affects all points in the grid but I assumed that the force of the wind increases when goes up, so the wind pressure become higher in the higher Z values of the surfaces. And at the same time, the force affects the inner points more than the points close to the edges. The points on the edges in the plan section do not move at all (fix points). In particle system there is a relation between particles as well, and we should define another set of rules for them, but here I just set the external forces just to show how we can work with points as particles.

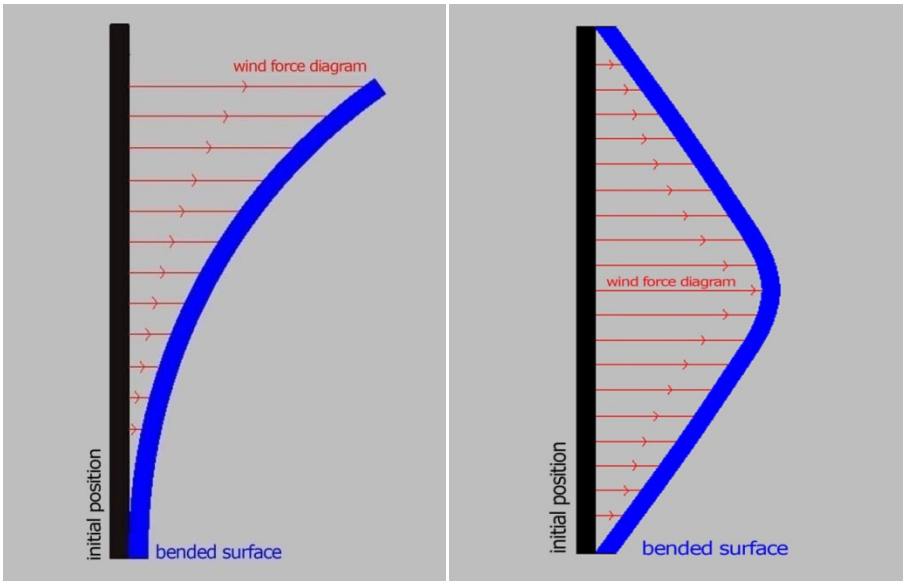


Fig.7.23. Diagram of the wind force affected the surface. A: section; the vertical effect of the force, B: plan; the horizontal effect.

Basically I need two different mechanisms to model these effects, one for the section diagram and another for the plan. I simplified the mechanism to an equation just to show the way we want the force affects points. For the first mechanism, to produce points displacements by using (X^2) while X is the Z value of the point being affected by the force. So for each point I need to extract the Z coordinate of the point.

To make everything simple, I just assumed that the force direction is in the Y direction of the world coordinate system. So for each point on the grid, I need to generate a vector in Y direction and I set its force by the number that I receive from the first equation. For the second diagram we need a bit more of an equation to do. Let's have a look at part one first.

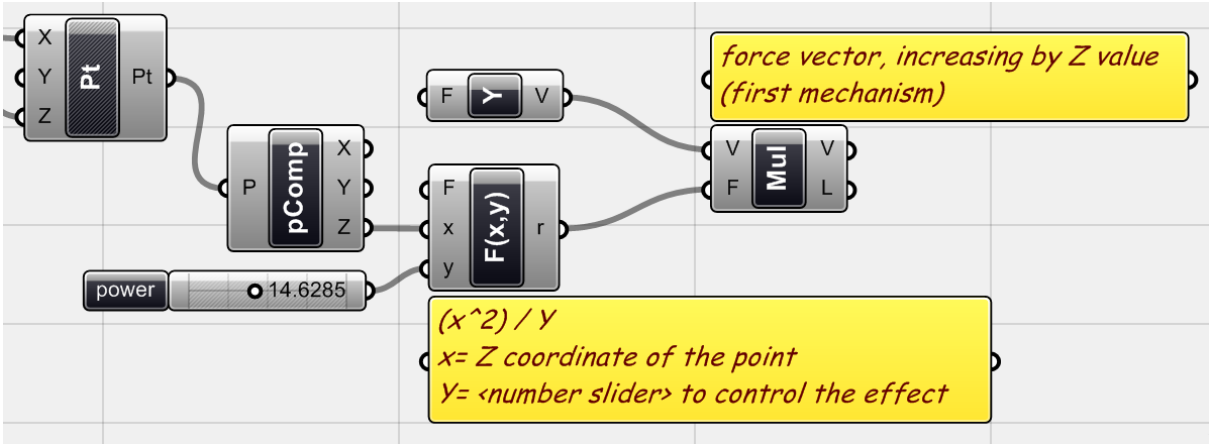


Fig.7.24. The Z coordinate of the point extracted by a <decompose> component and then powered by (x^2) and then divided by a given <number slider> just to control the movement generaly. The result is a factor to <multiply> the vector (Vector > Vector > Multiply) which is simply a world <unit Y> vector as force. So basically I generated force vectors for each point in the grid.

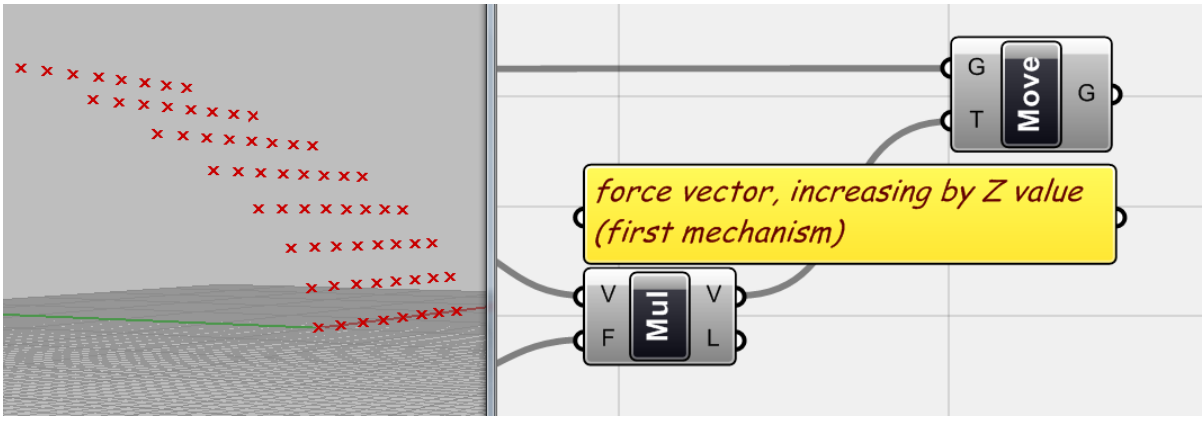


Fig.7.25. If I displace the points by these vectors you can see the resultant grid of points that satisfy the first step of this modelling.

Now if we look at the second part of the force modelling, as I said, I assumed that in the planner section, the points on the edge are fixed and the points on the middle displaced the most. Figure 7.26 show this displacement for each row of the point grid.

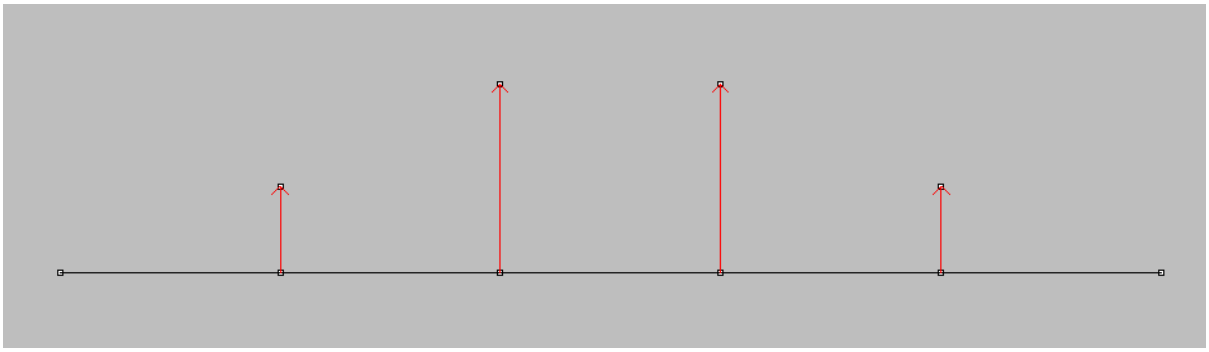


Fig.7.26. Displacement of points in rows (planner view).

Since I have the force vectors for each point, I need to control them and set a value again, to make sure that their displacement in the planner section is also met the second criteria. So for each row of the points in the grid, I want to generate a factor to control the force vector's magnitude. Here I assumed that for the points in the middle, the force vector's power are maximum that means what they are, and for the points on the edges, it become zero means no displacement and for the other points a range in between.

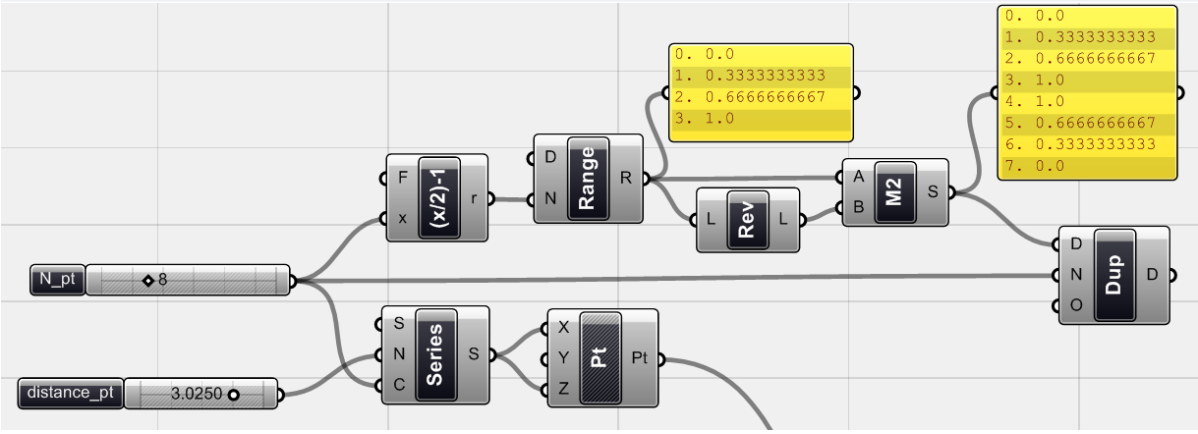


Fig.7.27. For the second mechanism, I need a <range> of numbers between 0 and 1 to apply to each point; 0 for the edge, 1 for the middle. I need a range from 0 to 1 from one edge to the middle and then from 1 to 0 to go from the middle to the other edge. I need this <range> component generates values as half of the numbers of points in each row. I set the <N_pt> to the even numbers, and I divided it by 2, then minus 1 (because the <range> component takes the number of divisions and not number of values). You see the first <panel> shows four numbers from 0 to 1 for the first half of the points. then I <reverse>d the list and I merge these two list together and as you see in the second <panel> I generated a list from 0 to 1 to 0 and the number of values in the list is the same as number of points in each row.

The final step is to generate these factors for all points in the grid. So I <duplicate>d the points as much as <N_pt> (number of rows and columns are the same). Now I have a factor for all points in the grid based on their positions in their rows.

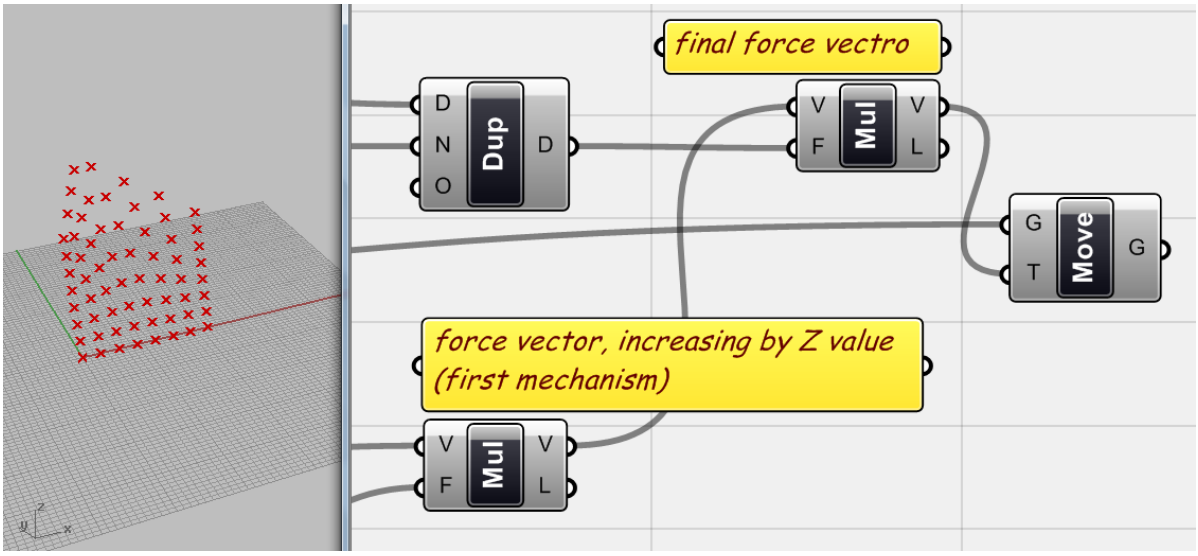


Fig.7.28. Now I need to <multiply> the force vector again by the new factors. If I displace the points by these new vectors, we can see how two different mechanisms affected the whole point grid.

Actually this part of the example needed a little bit of analytical thinking. In reality, methods like Particle spring systems or Finite Element Analysis use this concept that multiple vectors affecting the whole points in the set and points affecting each other also. So when you apply a force, it affects the whole points and points affecting each other simultaneously. These processes should be calculated in iterative loops to find the resting position of the whole system. Here I just make a simple example without these effects of particle systems and I just want to show a very simple representation of such a system dealing with multiple forces which in real subjects are a bit more complicated!

Mesh

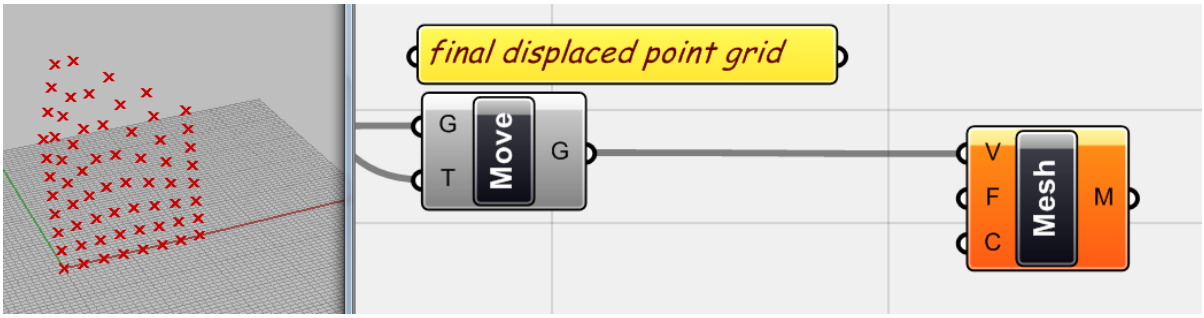


Fig.7.29. Mesh generation. Now if you simply add a <mesh> component (Mesh > Primitive > Mesh) to the canvas and connect the displaced points to it as vertices, you will see that nothing happening in the scene. We need to define the faces of the mesh geometry to generate it. Faces of the mesh are actually a series of numbers who just defines the way these points are connected together to make the faces of the surface. So here vertices are the geometrical part of the mesh but we need the topological definition of the mesh to generate it.

Every four corner point of the grid, define a quadrant face for the mesh object. If we look at the point grid, we see that there is an index number for each point in the grid. We know each point by its index number instead of coordinates in order to deal with its topology.

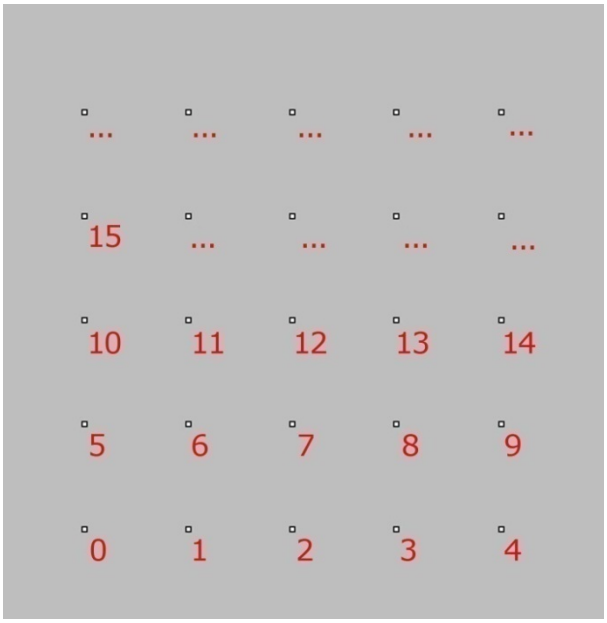


Fig.7.30. Index number of points in the grid.

To define the mesh faces, we need to call every four corners that we assumed to be a face and put them together and give them to the <mesh> component to be able to make the mesh surface. The order of this points in each face is the same for all faces.

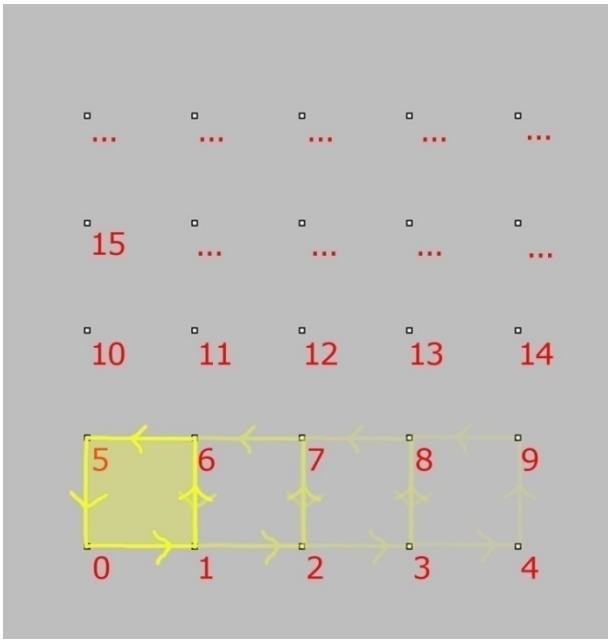


Fig.7.31. In a given point grid, a simple quadrant face defined by the order of points that if you connect them by a polyline, you can make the face. This polyline starts from a point in the grid, goes to the next point, then goes to the same point of the next row and then goes to the back column point of that row, and by closing this polyline, you will see the first face of the mesh finds its shape. Here the first face has points [0,1,6,5] in its face definition. The second face has [1,2,7,6] and so on.

To define all mesh faces, we should find the relation between these points and then make an algorithm that generates these face matrices for us.

If we look at the face matrix, we see that for any first point, the second point is the next in the grid. So basically for each point (n) in the grid, the next point of the face is (n+1). Simple!

For the next point of the grid, we know that it is always shifts one row, so if we add the number of columns (c) to the point (n) we should get the point at the next row (n+c). So for instance in the above example we have 5 columns so c=5 and for the point (1) the next point of the mesh face is point (n+c) means point (6). So for each point (n) as the first point, while the second points is (n+1), the third point would be (n+1+c). That's it.

For the last point, it is always stated in one column back of the third point. So basically for each point (n+1+c) as the third point, the next point is (n+1+c-1) which means (n+c). So for instance for the point (6) as the third point, the next point becomes point (5).

All together for any point (n) in the grid, the face that starts from that single point has this points as the ordered list of vertices: [n, n+1, n+1+c, n+c] while (c) is the number of columns in the grid.

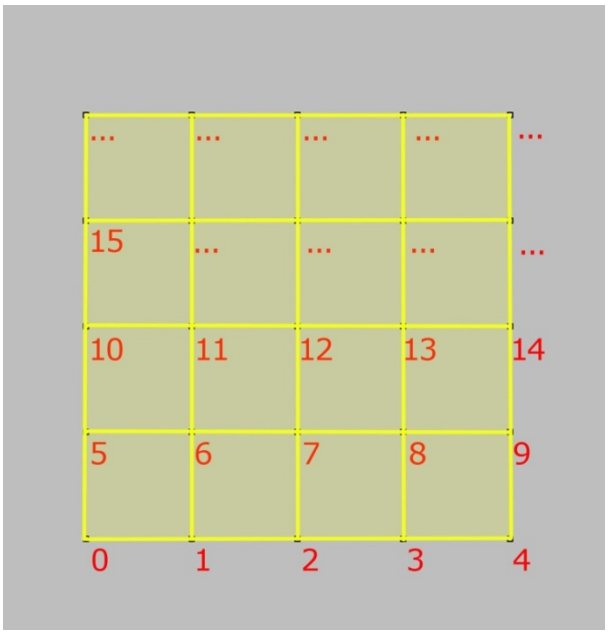


Fig.7.32. After defining all mesh faces, the mesh can be generated.

Looking at the mesh vertices, there is a bit more to deal with. If you remember the 'Triangle' example of chapter_3, there was an issue to select the points that could be the first points in the grid. If you look at the grid of points in the above example, you see that the points on the last column and last row could not be the start points of any face. So beside the fact that we need an algorithm to generate the faces of the mesh object, we need a bit of data management to generate the first points of the whole grid and pass these first points to the algorithm and generate mesh faces.

So basically in the list of points, we need to omit the points of the last row and last column and then start to generate face matrices. To generate the list of faces, we need to generate a list of numbers as the index of points.

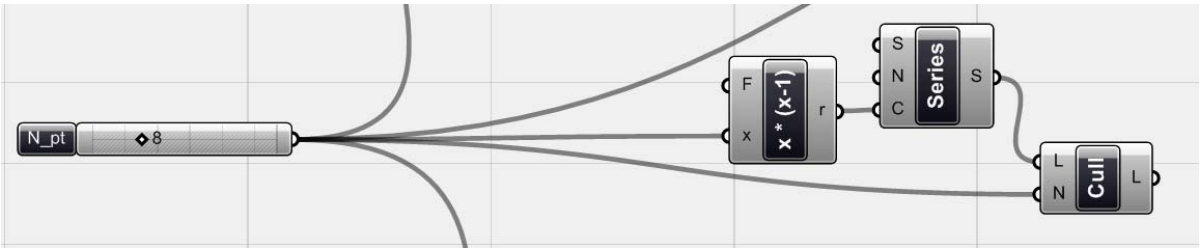


*Fig.7.33. Generating the index number of the first points in the grid with a <series> component. The number of values in the series comes from the <N_pt> as the number of columns (same as rows) and by using a function of < x * (x-1)> I want to generate a series of numbers as <columns*(rows-1)> to generate the index for all points in the grid and omit the last row. The next step is to <cull> the index list by the number of columns (<N_pt>) to omit the index of the last columns as well.*

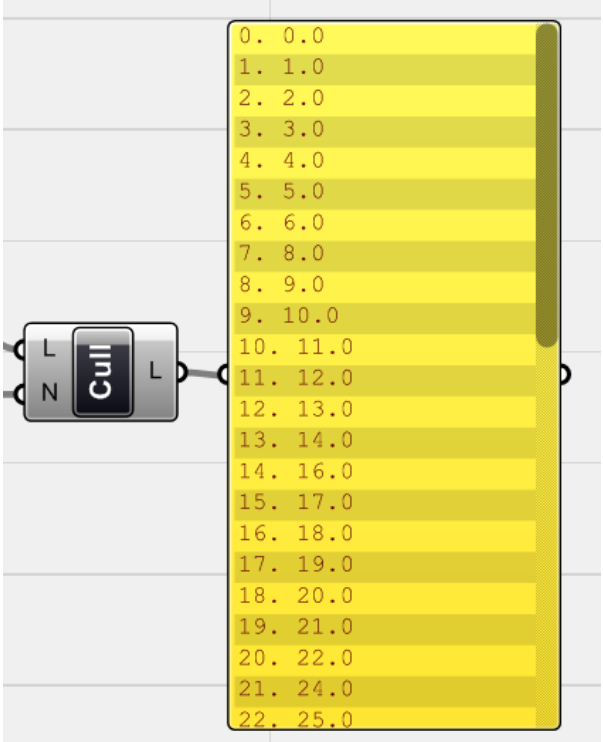


Fig.7.34. Final index number of the possible first points of the mesh faces in the grid (with 8 points in each column).

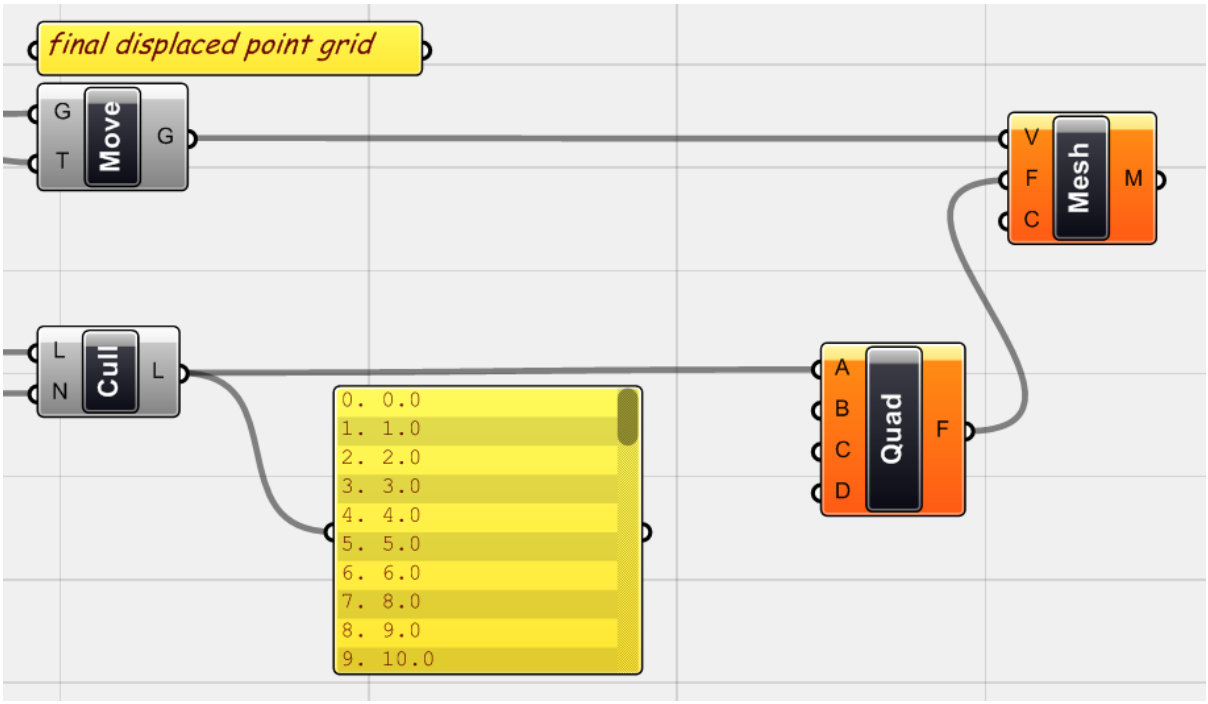


Fig.7.35. A <Mesh quad> component (Mesh > Primitive > Mesh quad) is in charge of generating quad faces in Grasshopper. I just attached the list of first numbers to the first point of the <quad>.

Now this is the time to generate the list of indices for the faces:

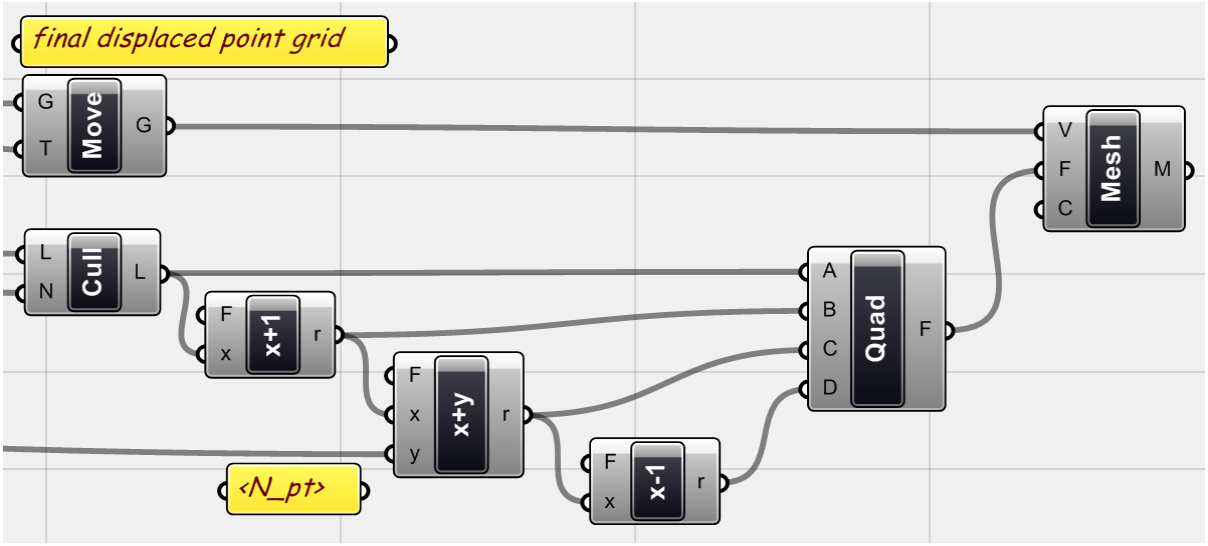


Fig.7.36. While (x) is the index of the first point (<cull> list) and (y) is the number of the columns(from <N_pt> number slider), the second point is (x+1), the third point is ((x+1)+y) (the index of second point + number of columns), and the last point is ((x+1+y)-1) (the index of the third point -1).

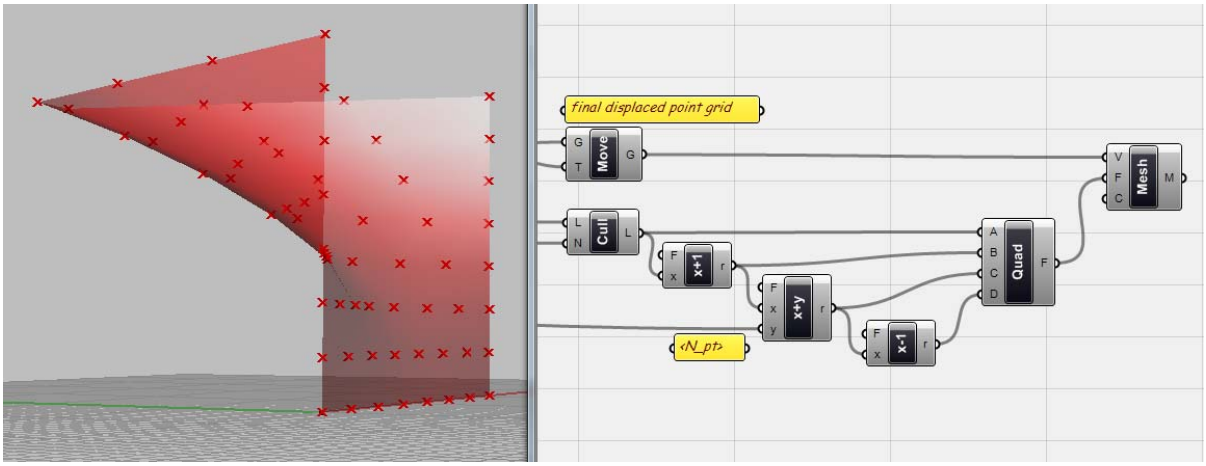


Fig.7.37. The resultant mesh.

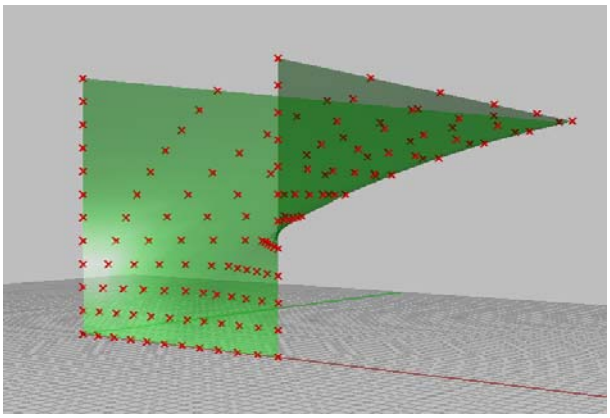


Fig.7.38. Changing the parameters, related to the force and manipulating the mesh.

7_4_On Colour Analysis

To finish this example, let's have a look at how we can represent our final mesh with colours as a medium for analysis purposes. There are different components in the Grasshopper that provide us colour representations and these colours are suitable for our analysis purpose.

Here in this example, again to bring a concept, I simply assumed that at the end, we want to see the amount of deviation of our final surface from the initial position (vertical surface). I want to apply a gradient of colours start from the points which remained fix with bottom colour up to the points which has the maximum amount of deviation from the vertical position with the higher colour of the gradient.

Simply, to find the amount of deviation, I need to measure the final state of each point to its original state. Then I can use these values to assign colour to the mesh faces base on these distances.

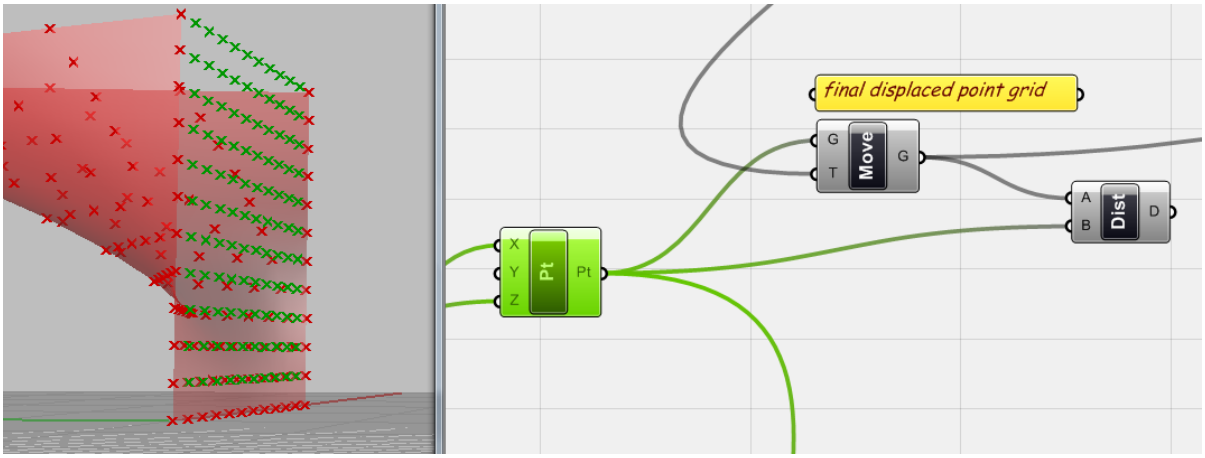


Fig.7.39. If I simply go back, I have the initial point grid that we generated in the first step and I have also the final displaced point grid that I used to generate the mesh vertices. I can use a <distance> component to measure the distance between the initial position of the point and its final position to see the deviation of the points.

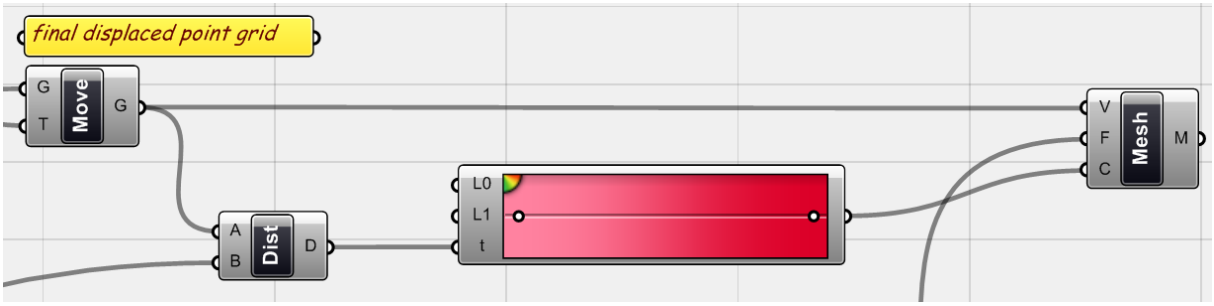


Fig.7.40. For our analysis purpose I want to use a <Gradient> component (Params > Special > Gradient) to assign gradient of colours to the mesh. I attached my <distance> values to the parameter part (t) of the <Gradient> and I attached it to the Colour input of the <mesh> component.

But to complete the process I need to define the lower limit and upper limit of gradient range (L0 and L1). Lower limit is the minimum value in the list and upper limit is maximum value in the list and other values are being divided in the gradient in between.

To get the lower and upper limit of the list of deviations I need to simply sort the data and get the first and last values in that numerical range.

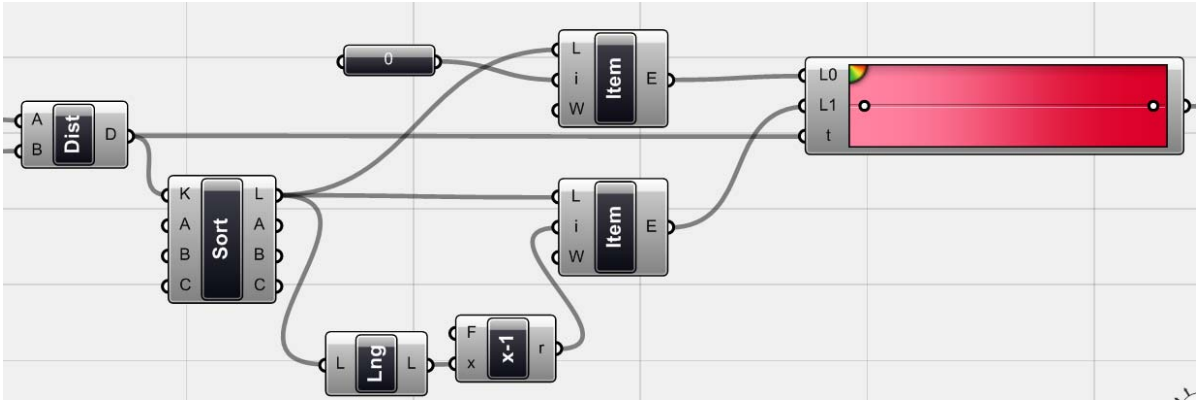


Fig.7.41. By using a <sort> component to sort the distances, I get the first item of the data list (index= 0) as lower limit and the last one (index= <list length> - 1) as the upper limit of the data set (deviation values) to connect to the <gradient> component to assign colours based on this range.

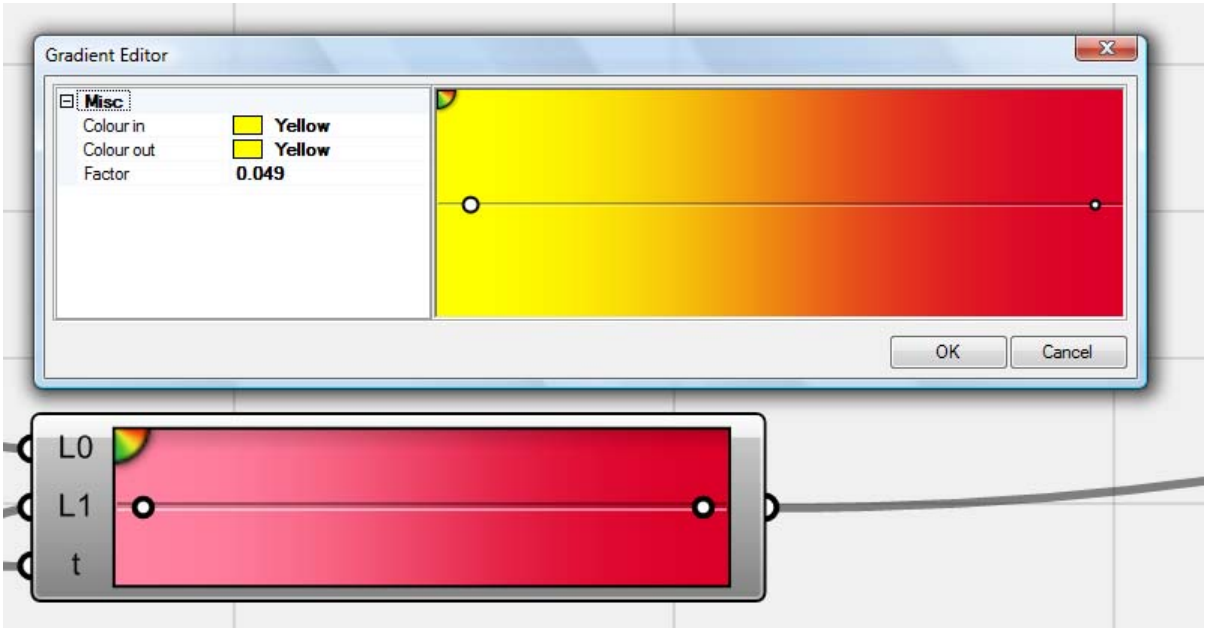


Fig.7.42. By clicking on the small colour icon on the corner of the <gradient> component we can change the colours of the gradient.

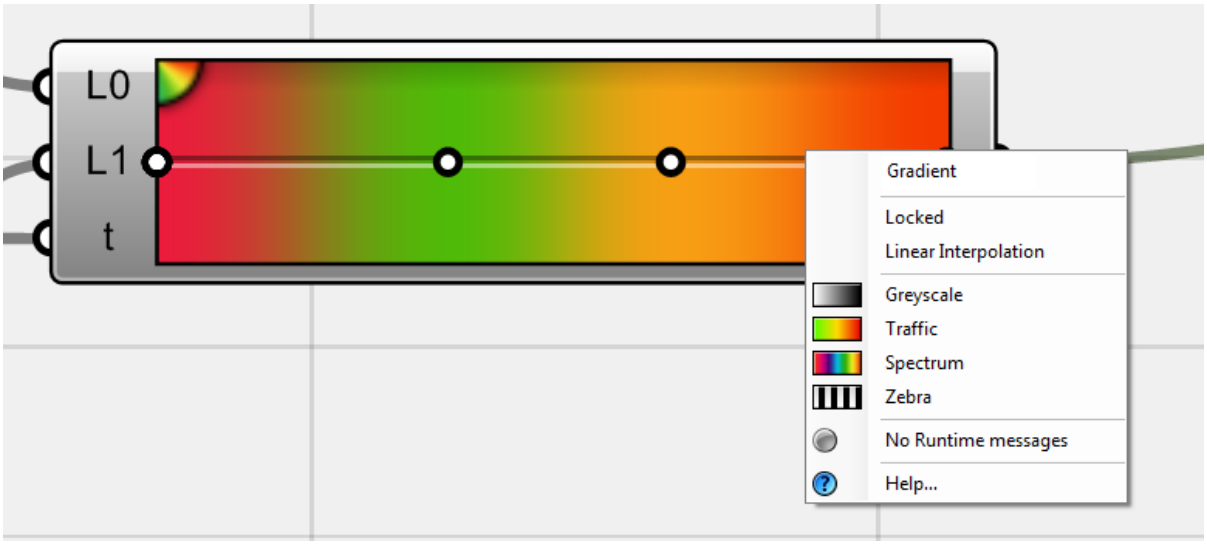


Fig.7.43. Right-click on the component and on the context pop-up menu you have more options to manipulate your resultant colours. Different types of colour gradient to suit the graphical representation of your analysis purpose.

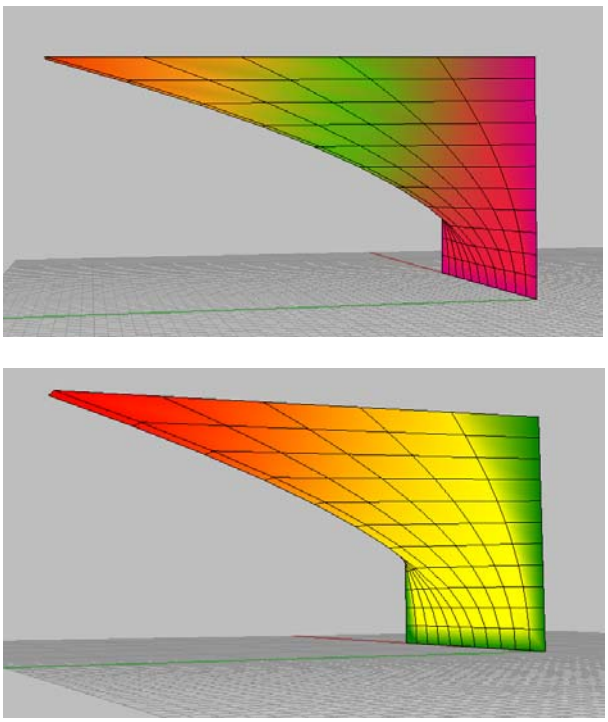


Fig.7.44.a/b. Different gradient thresholds.

7_5_Manipulating Mesh objects as a way of Design

Depends on the object and purpose of the modelling, I personally prefer to get my mesh object by manipulating a simple mesh geometry instead of generating a mesh from scratch since defining the point set and face matrices are not always simple and easy to construct. By manipulating, I mean that we can use a simple mesh object, extract its components and change them and then again make a mesh with varied vertices and faces. So I do not need to generate points as vertices and matrices of faces. Let's have a look at a simple example.

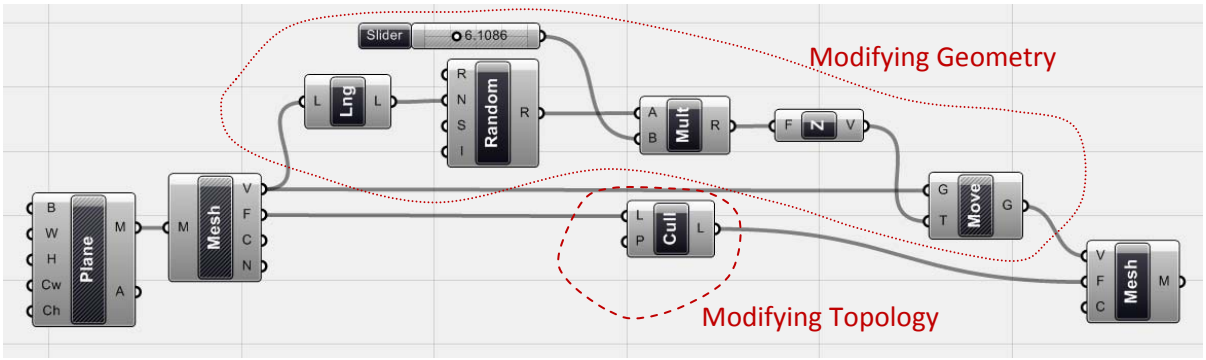


Fig.7.45. In this example, I simply used a <mesh plane> component and I extracted its data by using a <mesh components> to have access to its vertices and faces. Then I displaced vertices along Z direction by random values powered by a <number slider> and again attached them to a <mesh> component to generate another mesh. Here I also used a <cull pattern> component and I omitted some of the faces of original mesh and then I used them as new faces for making another mesh. The resultant mesh has both geometrical and topological difference wih its initial mesh and can be used for any design purpose.

This idea of geometrically manipulating the vertices and topologically changing the faces has so many different possibilities that you can use in your design experiments. Since mesh objects have the potential to omit some of their faces and still remain as surface, different design ideas like porous surfaces could be pursued by them.

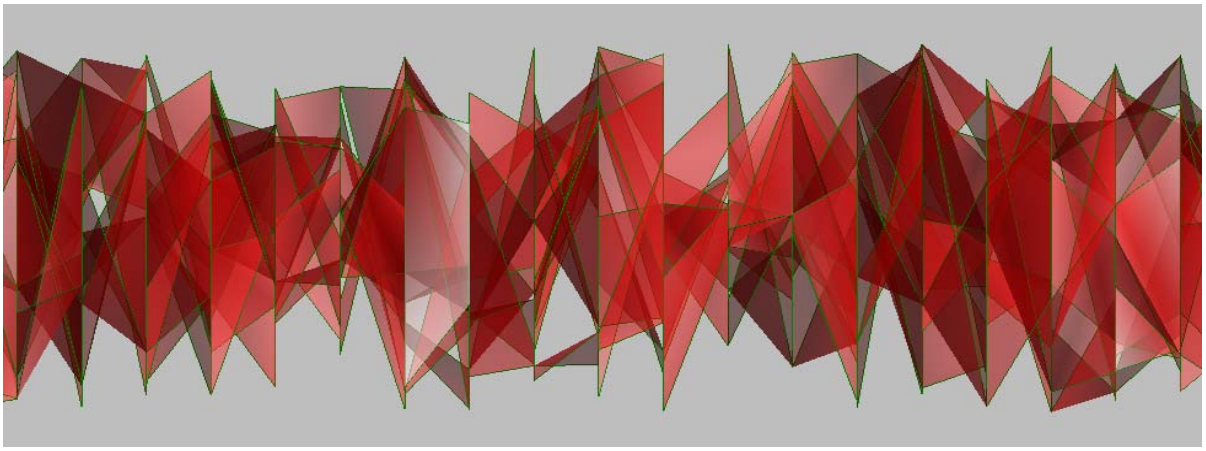
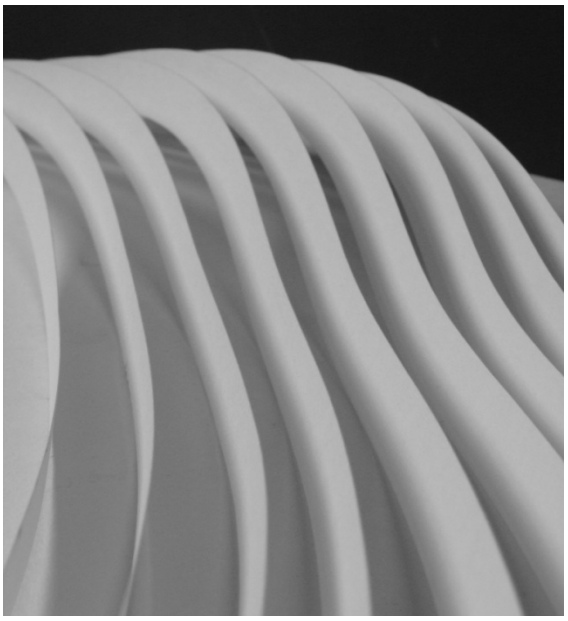


Fig.7.46. Resultant manipulated mesh (just a random case).

Fig.7.47.a/b. Final model.

Chapter_8_Fabrication



Chapter_8_Fabrication

Today there is a vast growing interest on material practice and fabrication in combination with Computer Aided Manufacturing. Due to the changes have happened in design processes, it seems a crucial move and one of the 'Musts' in the field of design. Any design decision in digital area, should be tested in different scales to show the ability of fabrication and assembly. Since it is obvious that the new design processes and algorithms do not fit into the traditional building processes, designers now try to use the modern technologies in fabrication to match their design products. From the moment that CNC machines started to serve the building industry up to now, a great relation between digital design and physical fabrication have been made and many different technologies and machineries being invented or adjusted to do these types of tasks.

In order to design building elements and fabricate them, we need to have a brief understanding of the fabrication processes for different types of materials and know how to prepare our design outputs for them. This is the main purpose of the fabrication issues in our design process. Based on the object we designed and material we used, assembly logic, transportation, scale, etc. we need to provide the suitable data from our design and get the desired output of that to feed machineries.

If traditional way in realization of a project made by Plans, Sections, Details, etc. today, we need more details or data to transfer them to CNC machines, to use them as source codes and datasheets for industries and so on.

The point here is that the designer should provide some of the required data, because it is highly interconnected with design object. Designer sometimes should use the feedback of the fabrication-data-preparation for the design readjustment. Sometimes the design object should be changed in order to fit the limitations of the machinery or assembly.

Up to this point we already know different potentials of the Grasshopper to alter the design, and these design variations could be in the favour of fabrication as well as other criteria. I just want to open the subject and touch some of the points related to the data-preparation phase, to have a look at different possibilities that we can extract data from design project in order to fabricate it or sometime readjust it to fit the fabrication limitations.

8_1_Datasheets

In order to make objects, sometimes we simply need a series of measurements, angels, coordinates and generally numerical data. There are multiple components in Grasshopper to compute the measurements, distances, angels, etc. the important point is the correct and precise selection of the objects that we need to address for any specific purpose. We should be aware of any geometrical complexity that exits in the design and choose the desired points for measurement purposes. The next point is to find the positions that give us the proper data for our fabrication purpose and avoid to generate lots of tables of numerical data which could be time consuming in big projects but useless at the end. Finally we need to export the data from 3D software to the spreadsheets and datasheets for further use.

Paper_strip_project

The idea of using paper strips attracted me for some investigations, although it had been tested before (like in Morpho-Ecologies by Hensel and Menges, 2008). To understand the simple assemblies I started with very simple combinations for first level and I tried to add these simple combinations together as the second level of assembly. It was interesting in the first tries but soon it became out of order and the result object was not what I assumed. So I tried to be more precise to get the more delicate geometries at the end.

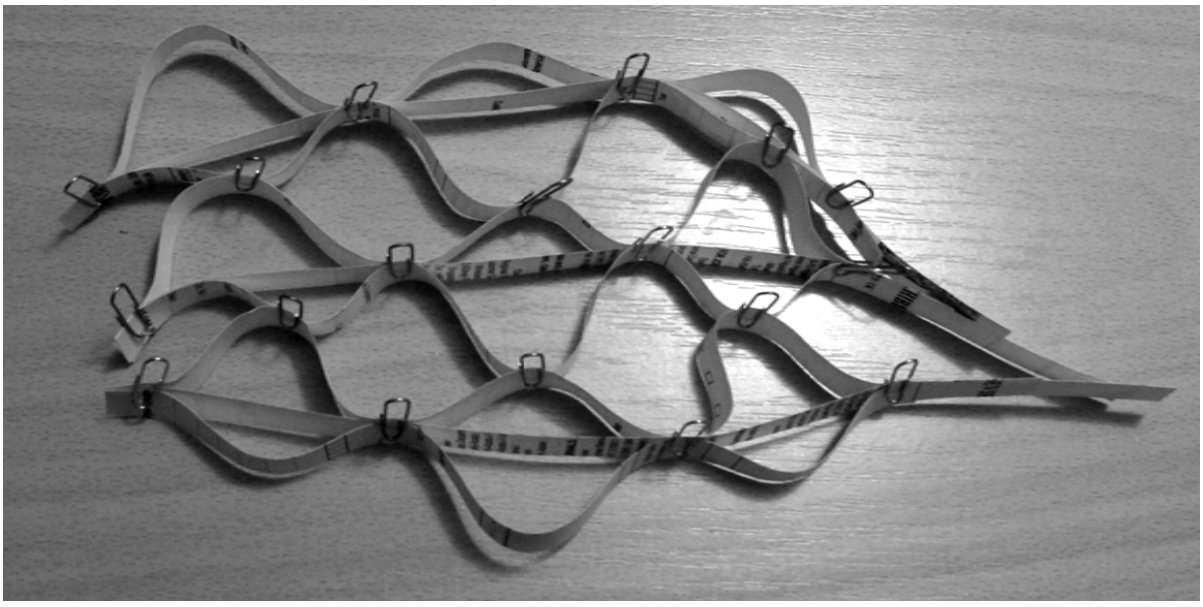


Fig.8.1. Paper strips, first try.

In the next step I tried to make a very simple set up and understand the geometrical logic and use it as the base for digital modelling. I assumed that by jumping into digital modelling I would not be able to make physical model and I was sure that I need to test the early steps with paper.

My aim was to use three paper strips and connect them, one in the middle and another two in two sides with longer length, restricted at their ends to the middle strip. This could be the basic module.

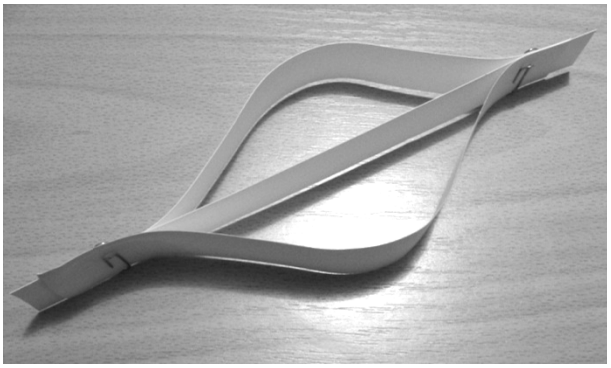


Fig.8.2. simple paper strip combination to understand the connections and move towards digital modelling.

Digital modelling

Here I wanted to model the paper strip digitally after my basic understanding of the physical one. From the start point I need a very simple curve in the middle as the base of my design and I can divide it and by culling these division points (true, false) and moving (false ones) perpendicular to the middle curve and using all these points (true ones and moved ones) as the vertices for two interpolated curves I can model this paper strips almost the same as what I described.

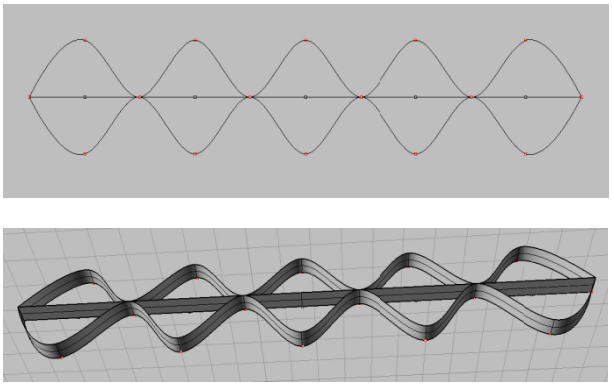


Fig.8.3.a/b. First modelling method with interpolated curves as side strips.

But it seemed so simple and straightforward. So I wanted to add a gradual size-differentiation in connection points so it would result in a bit more complex geometry. Now let's jump into Grasshopper and continue the discussion with modelling there. I will try to describe the definition briefly and go to the data parts.

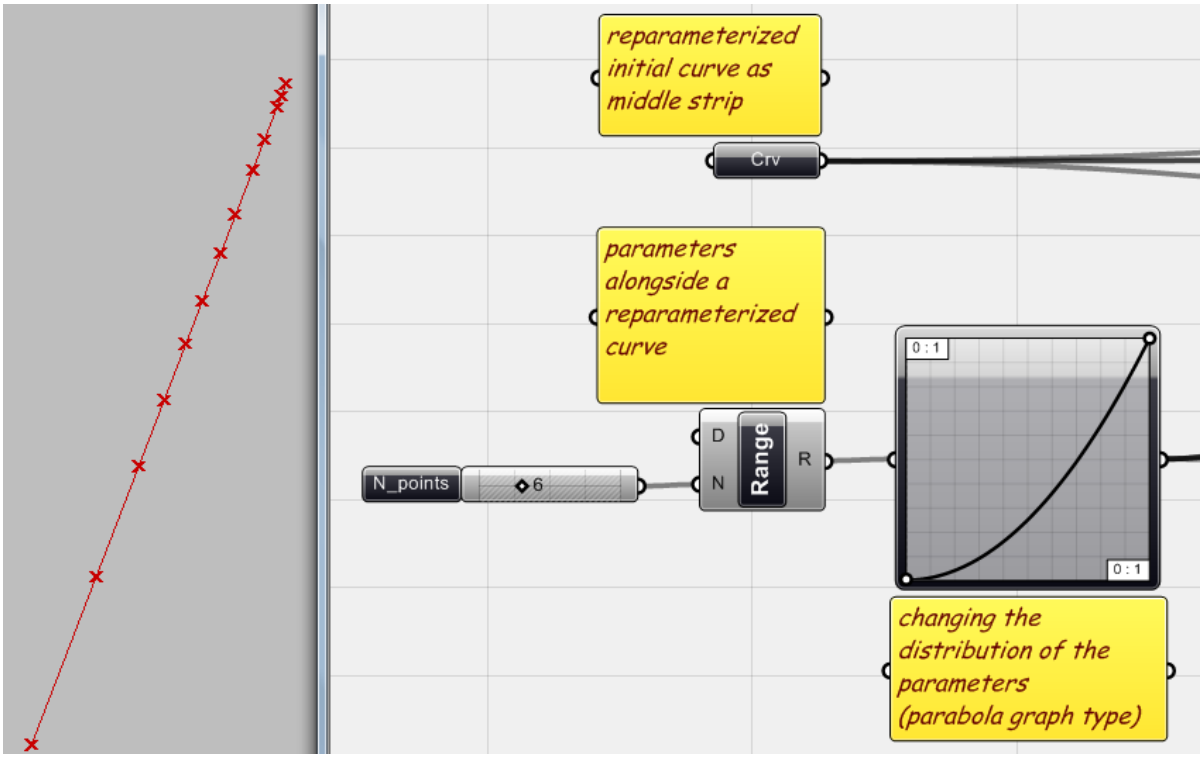


Fig.8.4. The <curve> component is the middle strip which is a simple curve in Rhino. I reparameterized it and I want to evaluate it in the decreasing intervals. I used a <range> component and I attached it to a <Graph Mapper> component (Params > Special > Graph Mapper). A <Graph mapper> remaps a set of numbers in many different ways and domains by choosing a particular graph type. As you see, I evaluated the curve with this <Graph mapper> with parabola graph type and the resultant points on the curve are clear. You can change the type of graph to change the mapping of numeric range (for further information go to the component help menu).

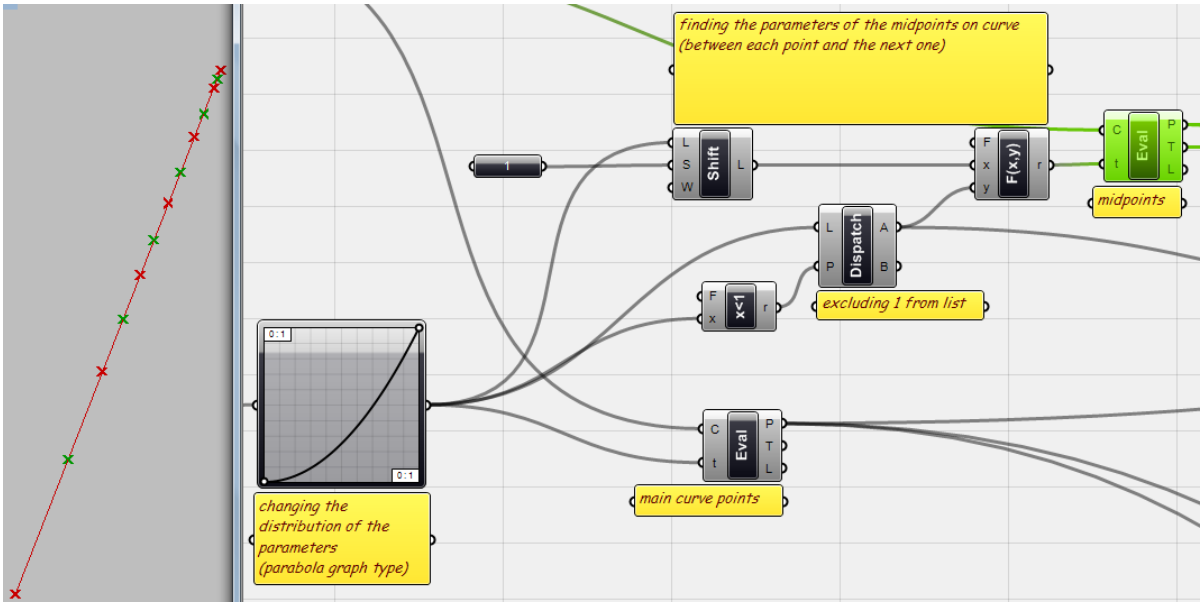


Fig.8.5. After remapping the numerical data I evaluated the middle curve with two different <evaluate> components. First by simply attach it to the data from <graph mapper> as basic points. Then I need to find the midpoints. Here I find the parameters of the curve between each basic point and the next one. I <shift>ed the data to find the next point and I used <dispatch > to exclude the last item of the list (exclude 1) otherwise I would have one extra point in relation to the <shift>ed points. The <function> component simply find the parameter in between (f(x)=(x+y)/2) and you see the resultant parameters being evaluated.

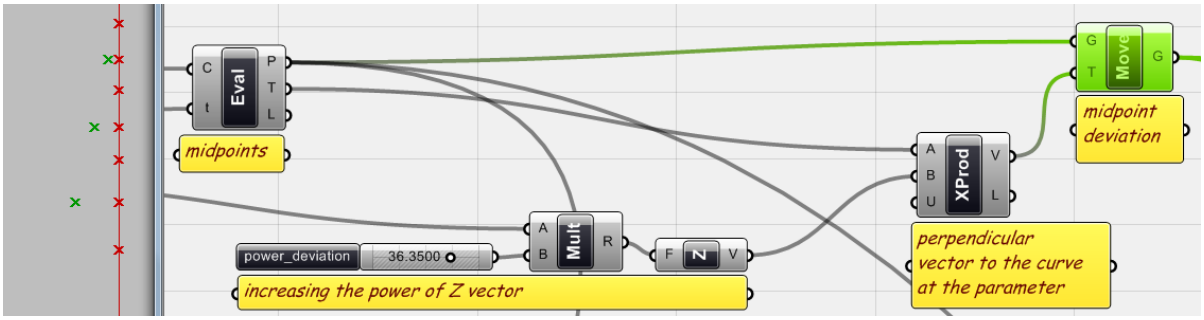


Fig.8.6. Now I want to move the midpoints and make the other vertices of the side strips. Displacement of these points must be always perpendicular to the middle curve. So in order to move the points I need vectors, perpendicular to the middle curve at each point. I already have the Tangent vector at each point, by <evaluate> component. But I need the perpendicular vector. We now that the ***Cross product*** *of two vectors is always a vector perpendicular to both of them (Fig.8.7). For example unit Z vector could be the cross product of the unit X and Y vectors. Our middle curve is a planer curve so we now that the Z vector at each point of the curve would be always perpendicular to the curve plane. So if I find the cross product of the Tangent of the vector and Z vector at each point, the result is a vector perpendicular to the middle curve which is always lay down in the curve's plane. So I used Tangent of the point from <evaluate> Component and a <unit Z> vector to find the <XProd> of them which I know that is perpendicular to the curve always. Another trick! I used the numbers of the <Graph Mapper> as the power of these Z vectors to have the increasing factors for the movements of points, in their displacements as well, so the longer the distance between points, the bigger their displacements.*

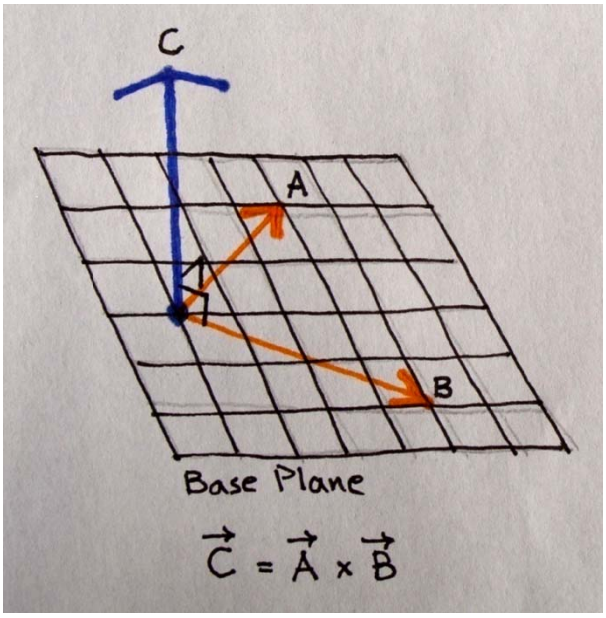


Fig.8.7. ***Vector cross product****. Vector A and B are in base plane. Vector C is the cross product of the A and B and it is perpendicular to the base plane so it is also perpendicular to both vectors A and B.*

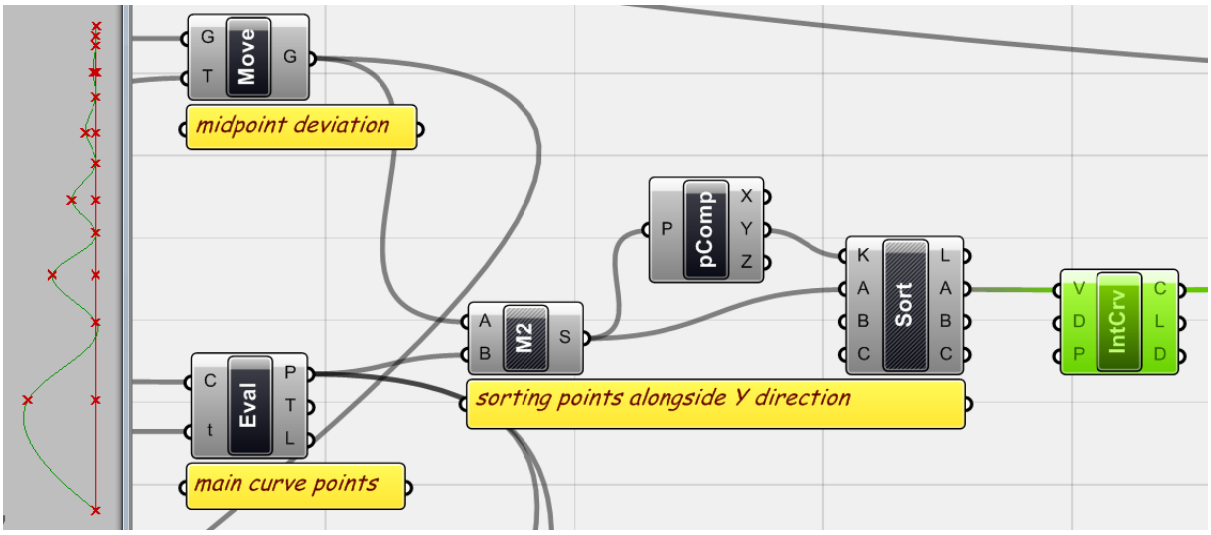


Fig.8.8. Now I have both basic points and moved points. I <merge>d them together and I sorted them based on their (Y) values to generate an <interpolate>d curve which is one of my side paper strip. (If you manipulate your main curve extremely or rotate it, you should sort your points by the proper factor).

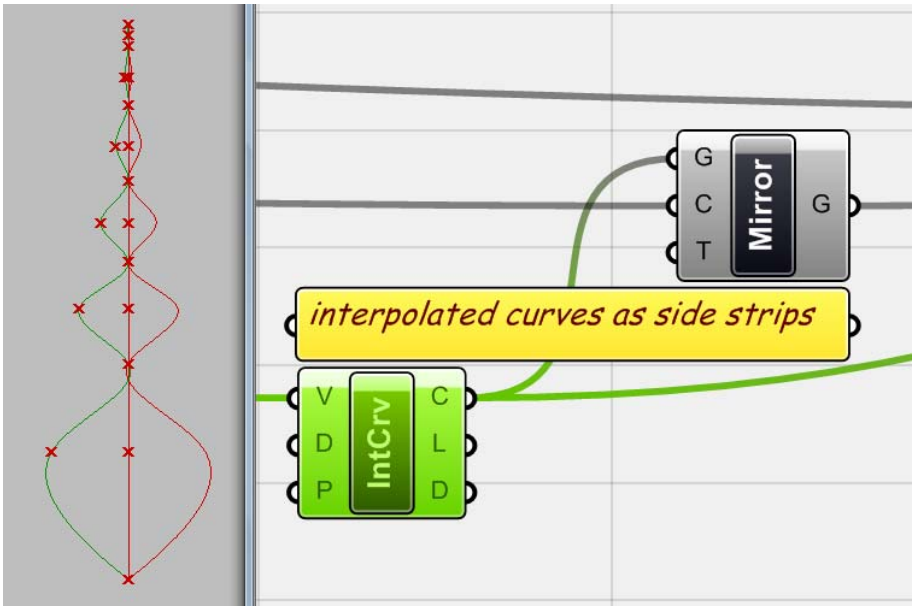


Fig.8.9. Using a <Mirror Curve> component (XForm > Morph > Mirror Curve) I can mirror the <interpolate>d curve by middle <curve> so I have both side paper strips.

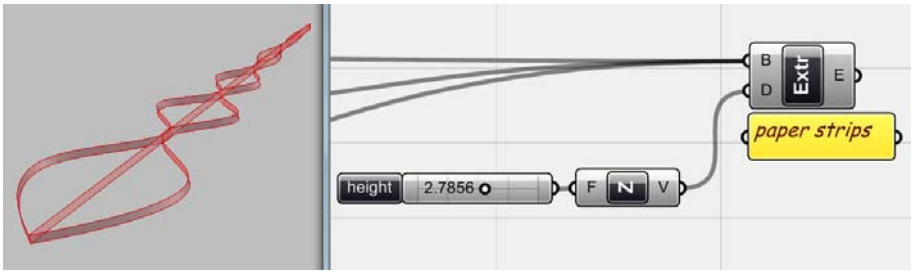


Fig.8.10. Now if I connect middle curve and side curves to an <extrude> component I can see my first paper strip combination with decreasing spaces between connection points.

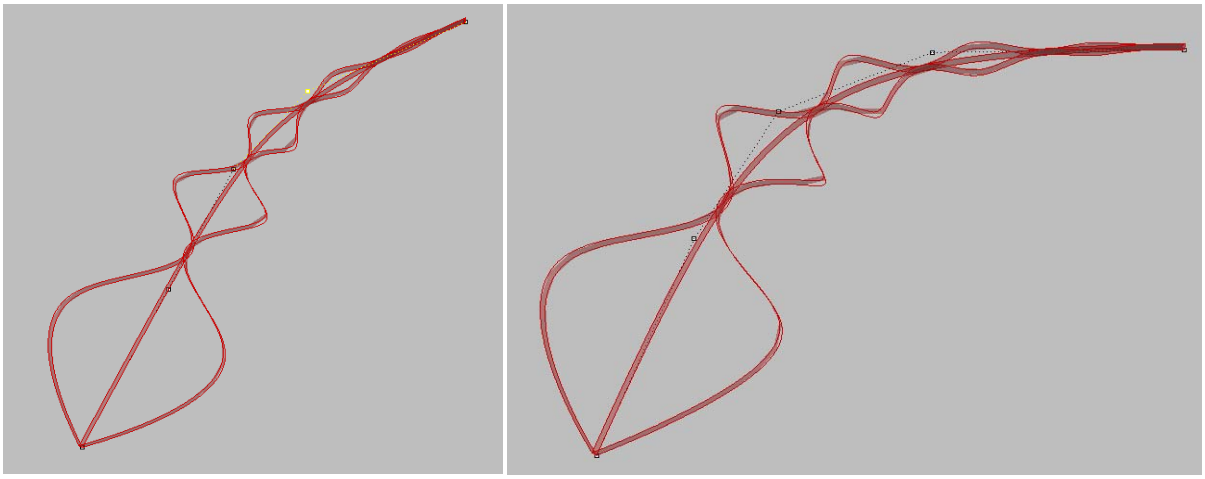


Fig.8.11. I can simply start to manipulate the middle strip and see how Grasshopper updates the three paper strips which are connecting to each other in six points.

After I found the configuration that I wanted to make a paper strip model, I needed to extract the dimensions and measurements to build my model with that data. Although it is quiet easy to model all these strips on paper sheets and cut them with laser cutter but here I like to make the process more general and get the initial data needed to build the model, so I am not limited myself to one specific machine and one specific method of manufacturing. You can use this data for any way of doing the model even by hand !!!! as I want to do in this case to make sure that I am not overwhelmed by digital!

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

Since strips are curve, the <distance > component does not help me to find the measurements. I need the length of curve between any two points on each strip. When I evaluate a parameter on a curve, it gives me its distance from the start point as well. So I need to find the parameter of the connection points of the strips (curves) and evaluate the position of them for each curve and the <evaluate> component would give me the distance of the points from the start point of curve means positions of connection points.

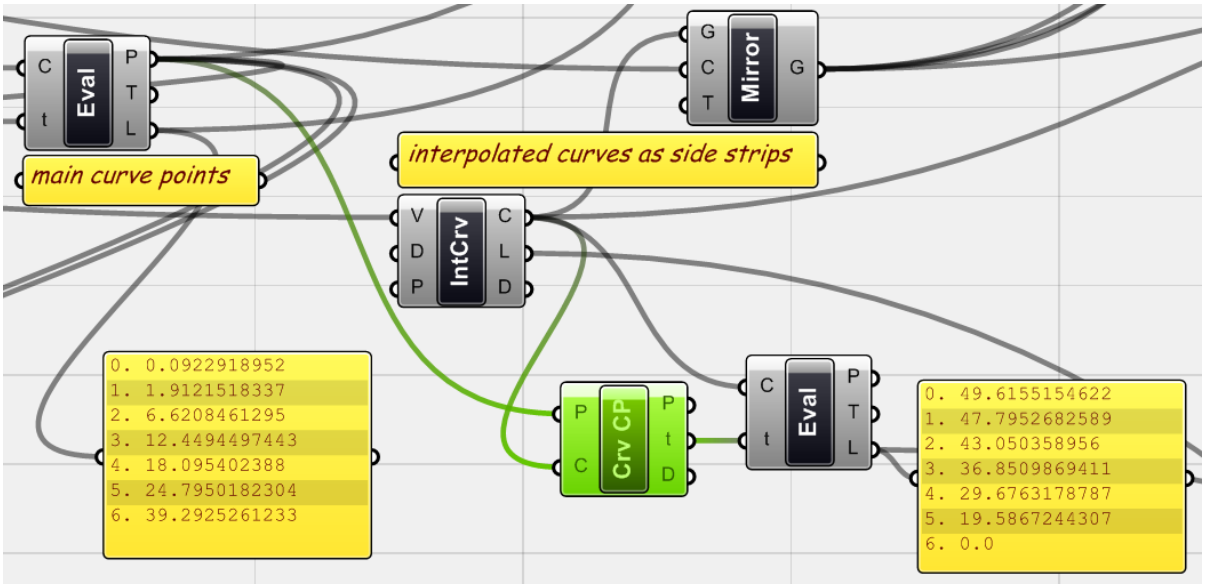


Fig.8.12. Although my file became a bit messy I replaced some components position on canvas to bring them together. As you see I used the first set of points that I called them 'main curve points' on the middle strip (initial curve). These are actually connection points of strips. The (L) output of the component gives me the distances of connection points from the start points of the middle strip. I used these points to find their parameter on the side curves as well (<curve cp> component). So I used these parameters to evaluate the side curve on those specific parameters (connection points) and find their distances from the start point. I can do the same to find the distance of the connection points on the other side strip (<mirror>ed one) also. At the end, I have the position of all connection points in each paper strip.

Make sure that the direction of all curves should be the same otherwise you need to change the direction of the curve or if it affects your project, you can simply add a minus component to minus this distances from the curve length which mathematically inverses the distance and gives you the distances of points from the start point instead of end point (or vice versa).

Exporting Data

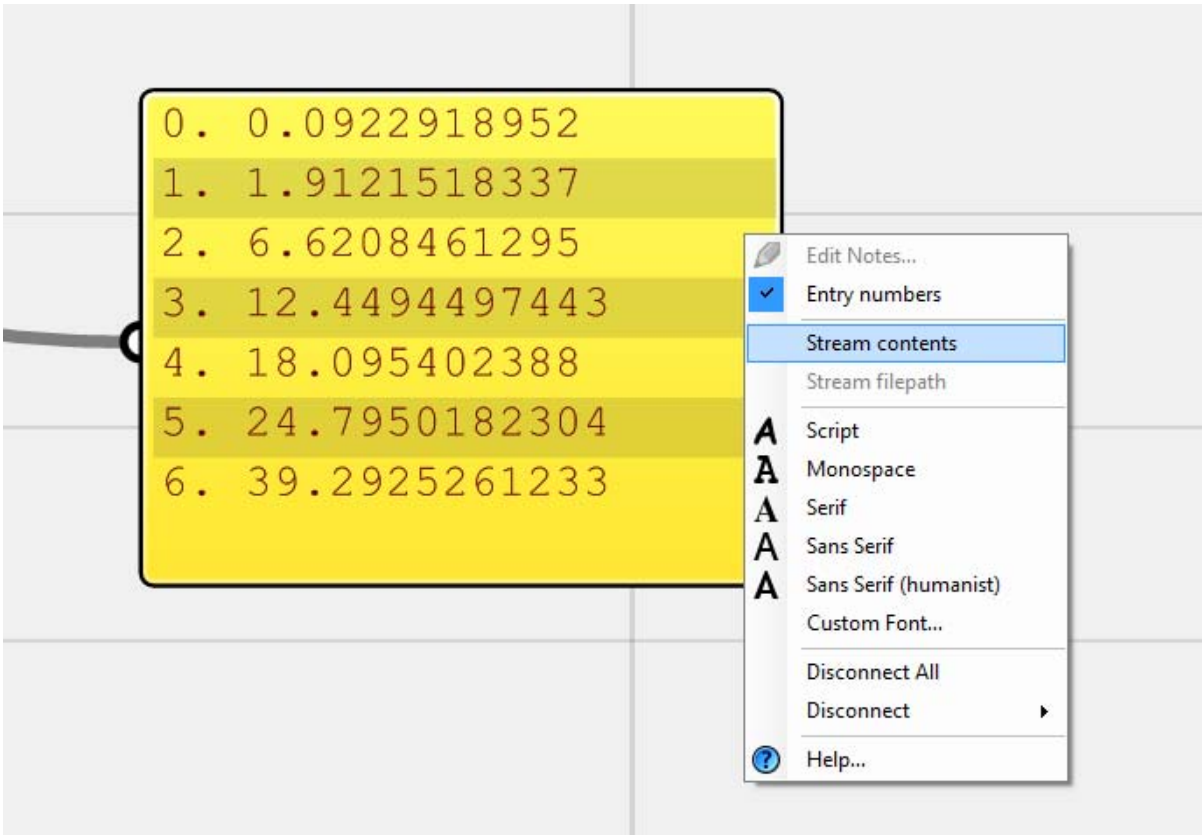


*Fig.8.13. Right-click on the <panel> component and click on the 'stream contents'. By this command you would be able to save your data in different formats and use it as a general numeric data. Here I would save it with simple .**txt** format and I want to use it in Microsoft Excel.*

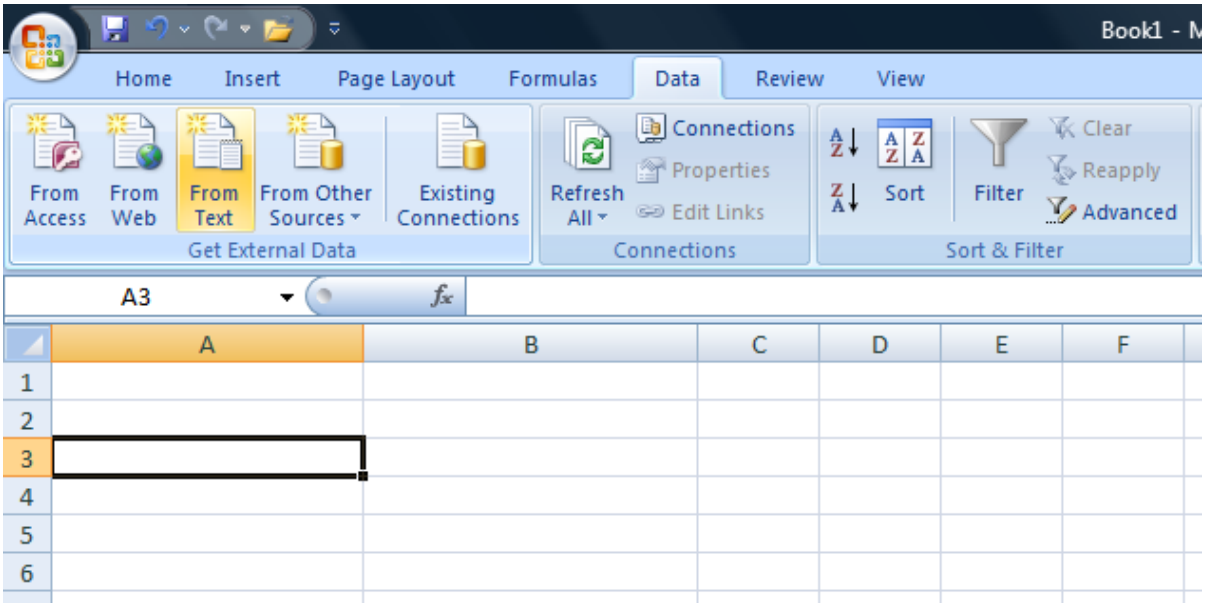


*Fig.8.14. On the Excel sheet, simply click on an empty cell and go to the **'Data'** tab and under the **'Get External Data'** select **'From Text'**. Then select the saved txt file from the address you saved your stream content and follow the simple instructions of excel. These steps allow you to manage your different types of data, how to divide your data in different cells and columns etc.*

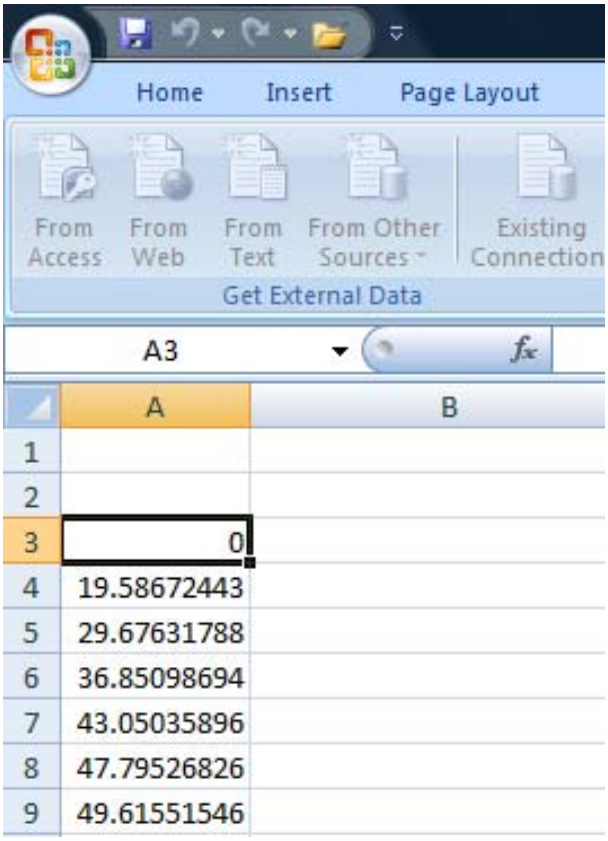


Fig.8.15. Now you see that your data placed on the Excel data sheet. You can do the same for the rest of your strips.

	Connection point distances from start point		
Connection points	Right_strip	Middle_strip	Left_strip
start point	0	0	0
1st connection point	19.58672443	14.49750789	18.71121037
2nd connection point	29.67631788	21.19712374	28.22812292
3rd connection point	36.85098694	26.84307638	34.95597765
4th connection point	43.05035896	32.67167999	41.0266449
5th connection point	47.79526826	37.38037429	45.75826257
end point_(strip length)	49.61551546	39.20023423	47.57834643

Fig.8.16. Table of the connection points alongside the strip.

If you have a list of 3D coordinates of points and you want to export them to the Excel, there are different options than the above example. If you export 3D coordinates with the above method you will see there are lots of unnecessary brackets and commas that you should delete. You can also add columns by clicking in the excel import text dialogue box and separate these brackets and commas from the text in different columns and delete them but again because the size of the numbers are not the same, you will find the characters in different columns that you could not align separation lines for columns easily.

In such case I simply recommend you to decompose your points to its components and export them separately. It is not a big deal to export three lists of data instead of one.

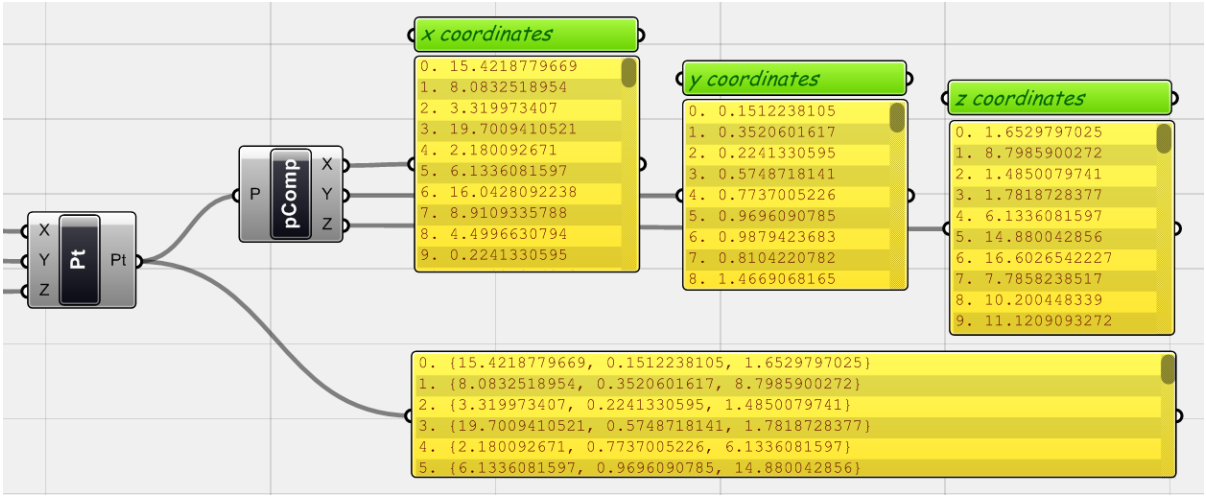
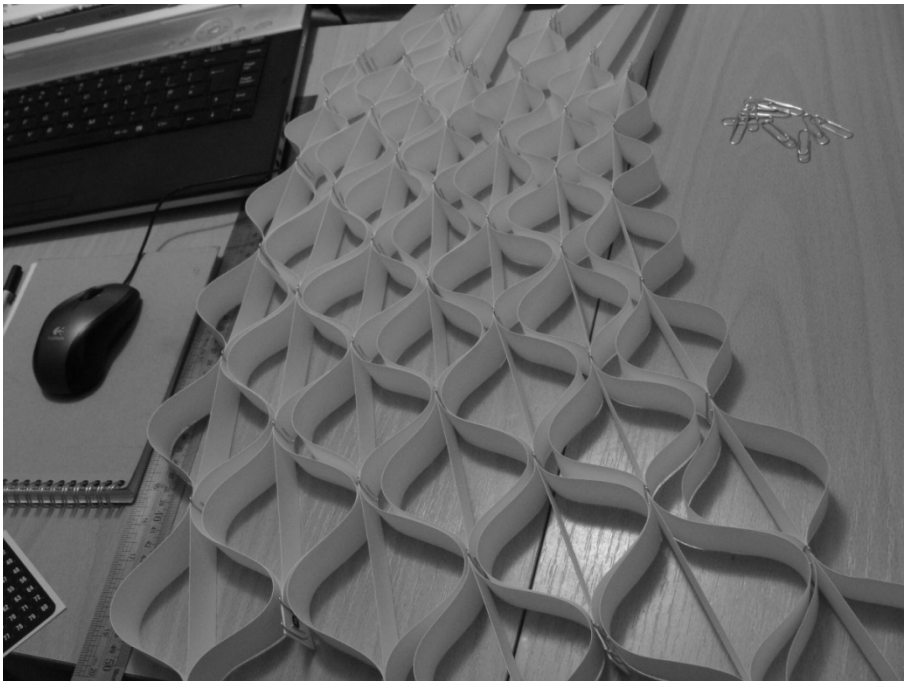


Fig.8.17. Using <decompose> component to get the X, Y and Z coordinates of the points separately to export to a data sheet.

You can also use the 'Format()' function to format the output text, directly from a point list in desired string format. You need to define your text in way that you would be able to separate different parts of the text by commas in separate columns in datasheet.

Enough for modelling! I used the data to mark my paper strips and connect them together. To prove it even to myself, I did all the process with hand !!!! to show that fabrication does not necessarily mean laser cutting (**HAM**, as Achim Menges once used for Hand Aided Manufacturing!!!!). I just spent an hour to cut and mark all strips but the assembly process took a bit longer which should be by hand anyway.



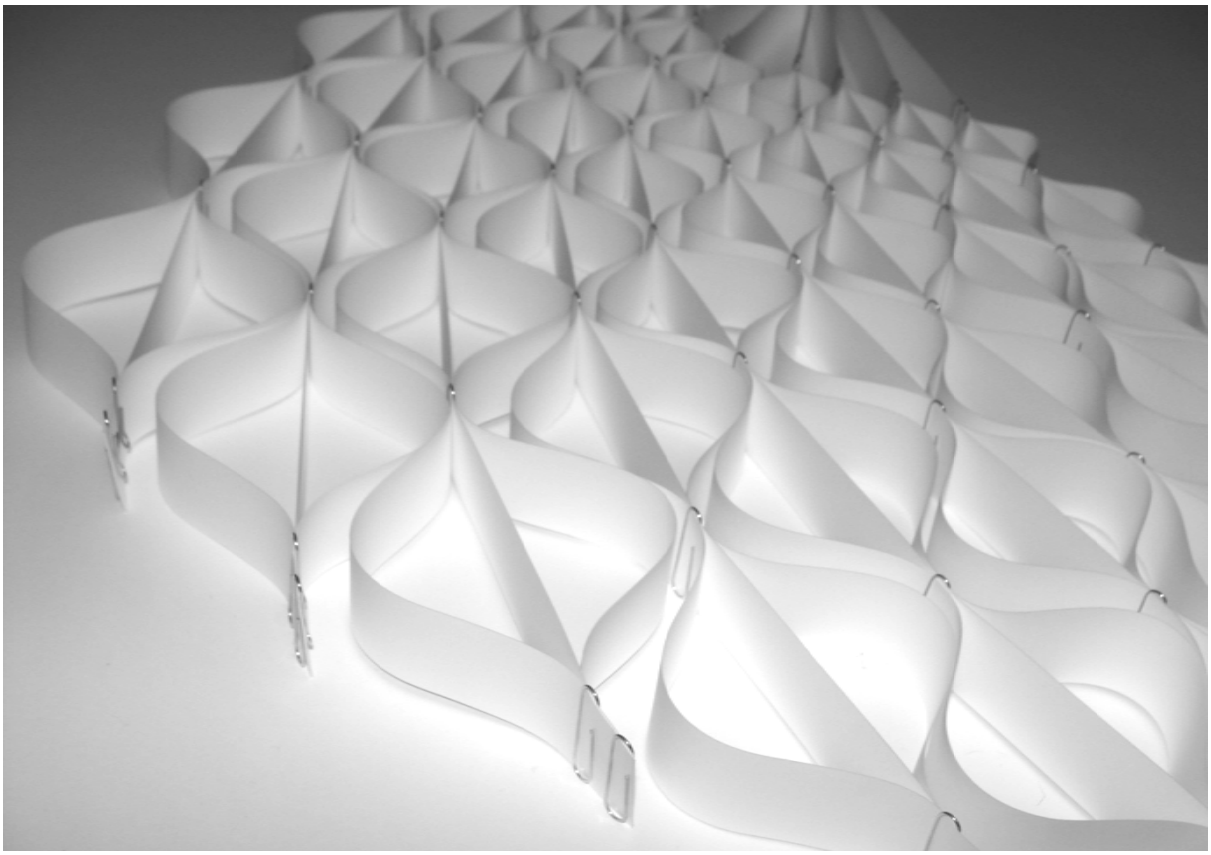


Fig.8.18.a/b/c. Final paper-strip project.

8_2_Laser Cutting and Cutting based Fabrication

The idea of laser cutting on sheet materials is very common these days to fabricate complex geometries. There are different ways that we can use this possibility to fabricate objects. Laser cutter method suits the objects that built with developable surfaces or folded ones. One can unfold the digital geometry on a plane and simply cut it out of a sheet and fold the material to build it. It is also suitable to make complex geometries that could be reduced to separate pieces of flat surfaces and one can disassemble the whole model digitally in separate parts, nest it on flat sheets, add the overlapping parts for connection purposes (like gluing) and cut it and assemble it physically. It is also possible to fabricate double-curve objects by this method. It is well being experimented to find different sections of any 'Blob' shaped object, cut it at least in two directions and assemble these sections together usually with Bridle joints and make rib-cage shaped models.

Since the laser cutter is a generic tool, there are other methods also, but all together the important point is to find a way, to reduce the geometry to flat pieces to cut them from a sheet material, no matter paper or metal, cardboard or wood and finally assemble them together (if you have Robotic arm and you can cut 3D geometries it is something different!).

Among the different ways discussed here I want to test one of them in Grasshopper and I am sure that you can do the other methods based on this experiment easily.

Free-form surface fabrication

I decided to fabricate a free-form surface to have some experiments with preparing the nested parts of a free-form object to cut and all other issues we need to deal with.

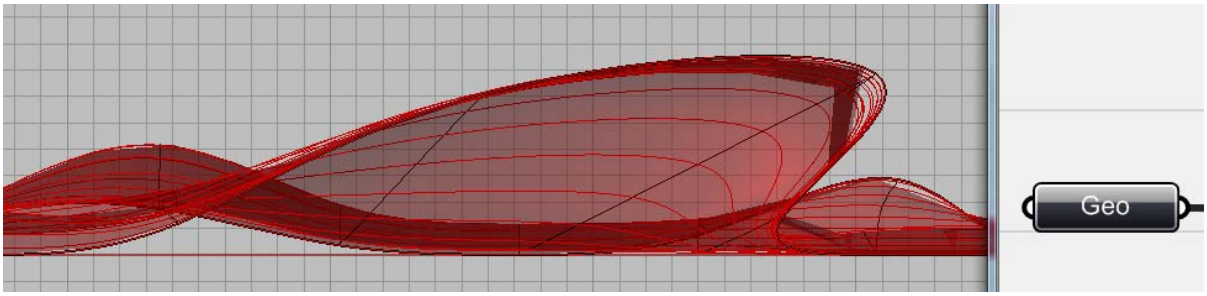


Fig.8.19. Here I have a surface and I introduced this surface to Grasshopper as a <Geometry> component, so you can introduce any geometry that you have designed or use any Grasshopper object that you have generated.

Sections as ribs

In order to fabricate this generic free-form surface I want to create sections of this surface, nest them on sheets and prepare the files to be cut by laser cutter. If the object that you are working on has a certain thickness then you can cut it but if like this surface you do not have any thickness you need to add a thickness to the cutting parts.

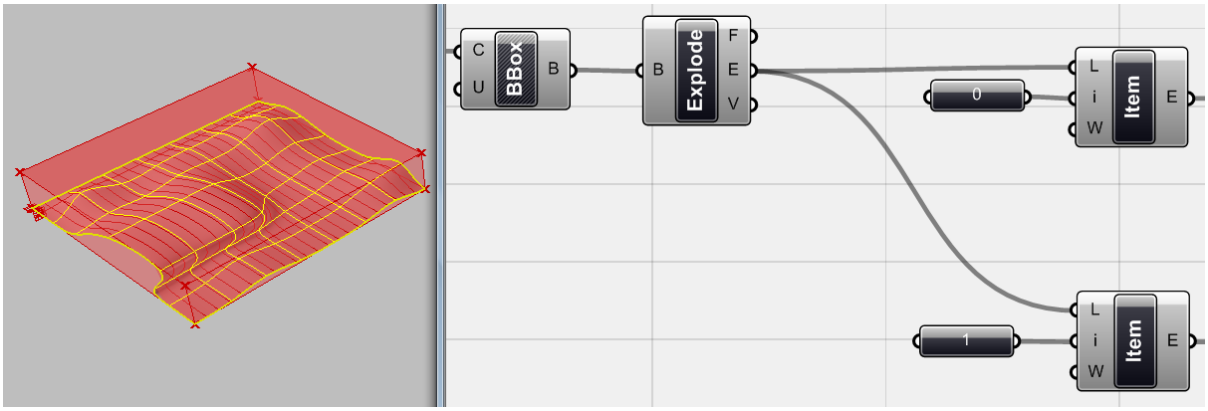


Fig.8.20. In the first step I used a <Bounding Box> component to find the area that I want to work on. Then by using an <Explode> component (Surface > Analysis > BRep components) I have access to its edges. I selected the first and second one (index 0 and 1) which are perpendicular to each other.

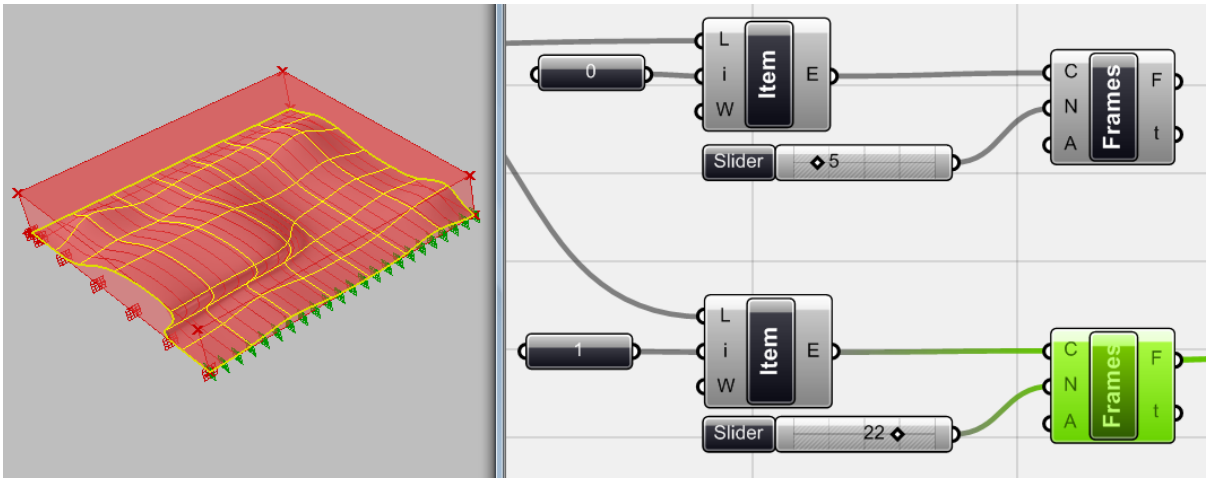


Fig.8.21. In this step I generated multiple perpendicular frames alongside each of selected edges. The number of frames is actually the number of ribs that I want to cut.

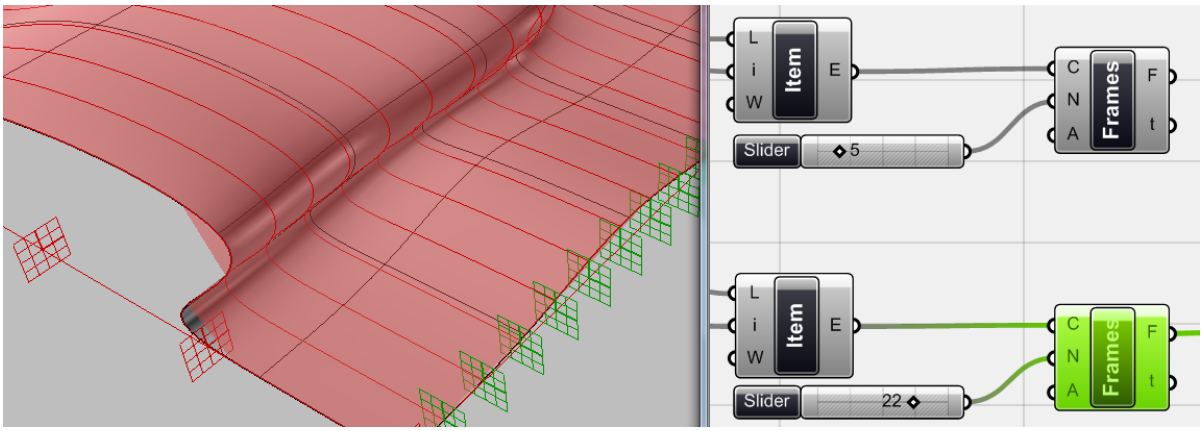


Fig.8.22. Closer view of frames generated alongside the length and width of the object's bounding box. As you see I can start to cut my surface with this frames.

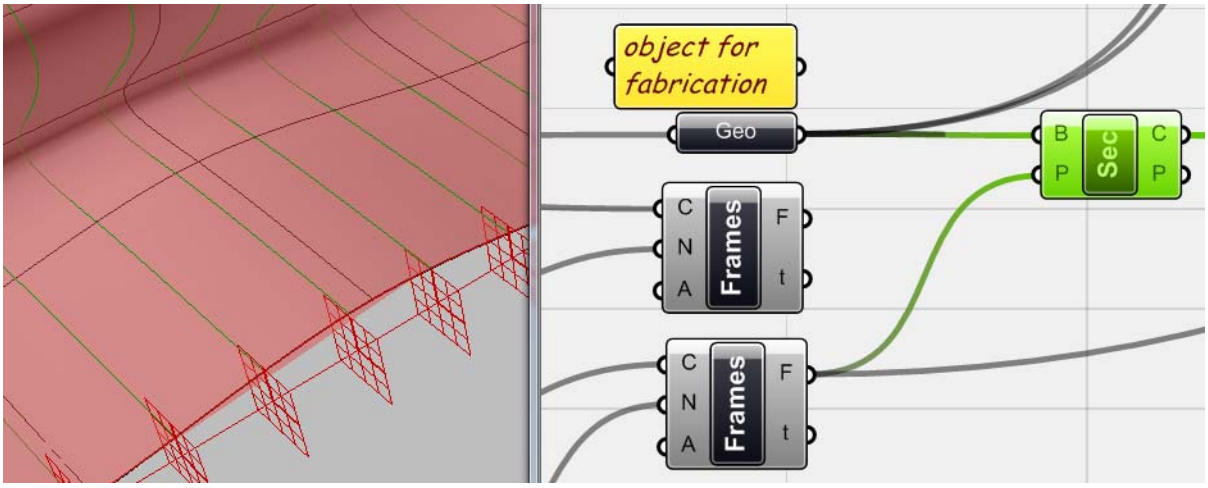


Fig.8.23. Now if I find the intersections of these frames and the surface (main geometry), I actually generated the ribs base structure. Here I used a <BRep | Plane> section component (Intersect > Mathematical > BRep | Plane) to solve this problem. I used the <Geometry> (my initial surface) as BRep and generated frames, as planes to feed the section component.

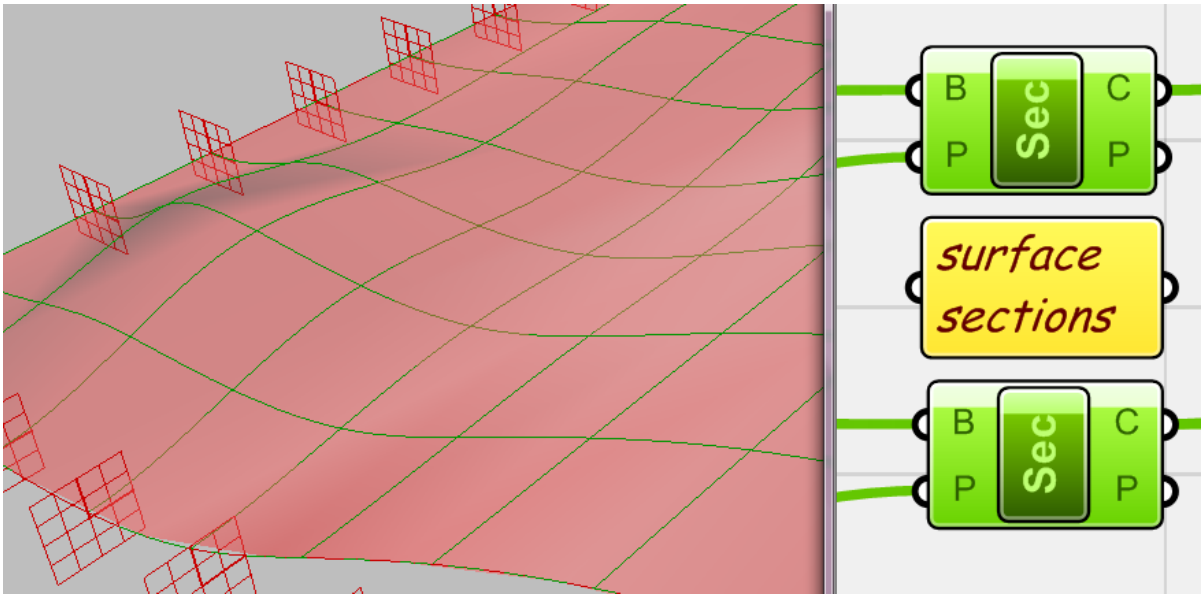


Fig.8.24. Intersections of frames and surface, resulted in series of curves on the surface.

Nesting

The next step is to nest these curve sections on a flat sheet to prepare them for the cutting process. Here I drew a rectangle in Rhino with my sheet size. I copied this rectangle to generate multiple sheets overlapping each other and I drew one surface that covers all these rectangles to represent them into Grasshopper.

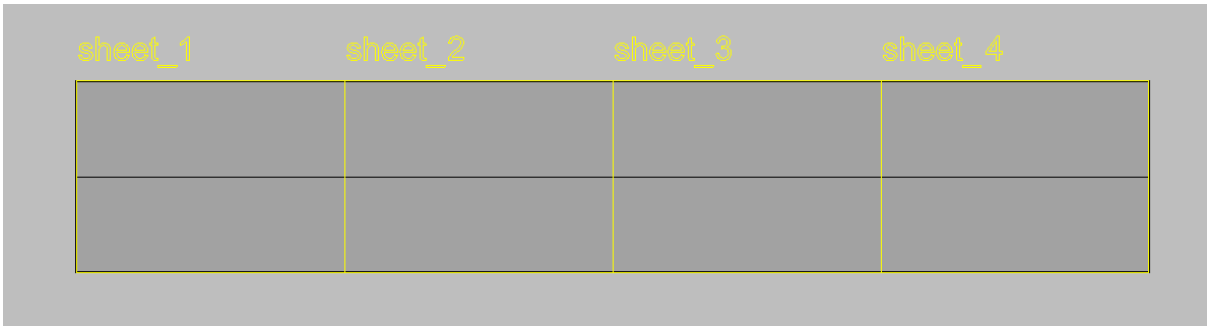


Fig.8.25. Paper sheets and an underlying surface to represent them in Grasshopper.

I am going to use <Orient> component (XForm > Euclidian > Orient) to nest my curves into the surface which represents the sheets for cutting purpose. If you look at the <orient> component you see that we need the objects plane as reference plane and target plane which should be on the surface. Since I used the planes to intersect the initial surface and generate the section curves, I can use them again as reference planes, so I need to generate target planes.

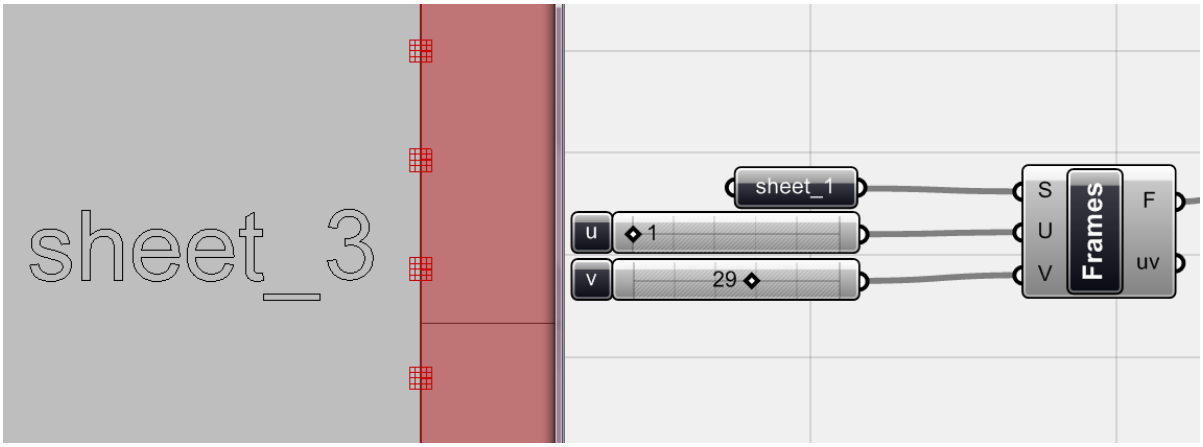


Fig.8.26. I introduced the cutting surface to Grasshopper and I used a <surface Frame> component (Surface > Util > Surface frames) to generate series of frames across the surface. It actually works like <divide surface> but it generates planes as the output, so exactly what I need.

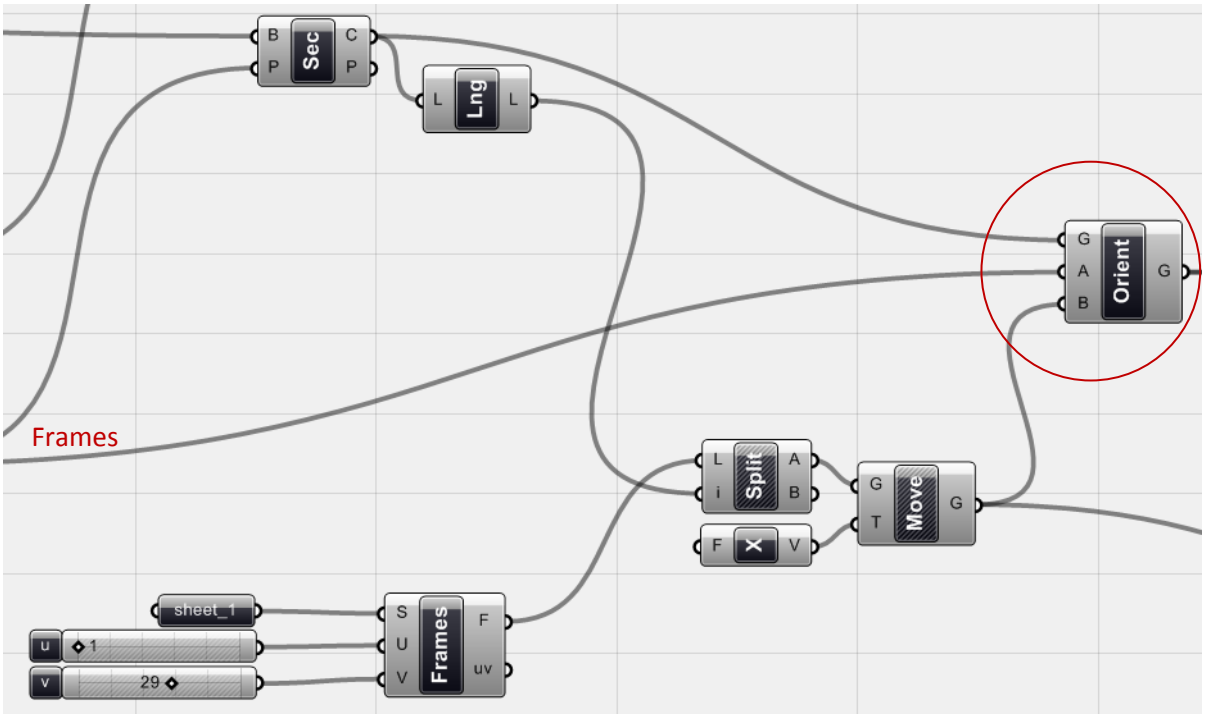


Fig.8.27. Orientation. I connected the section curves as base geometries, and the planes that I used to generate these sections as reference geometry to the <orient> component. But still a bit of manipulations is needed for the target planes. If you look at the <surface frame> component results you see that if you divide U direction even by 1 you will see it would generate 2 columns to divide the surface. So I have more planes than I need. So I <split> the list of target planes by the number that comes from the number of reference curves. So I only use planes as much as curve that I have. Then I moved these planes 1 unit in X direction to avoid overlapping with the sheet's edge. Now I can connect these planes to the <orient> component and you can see that all curves now nested on the cutting sheet.

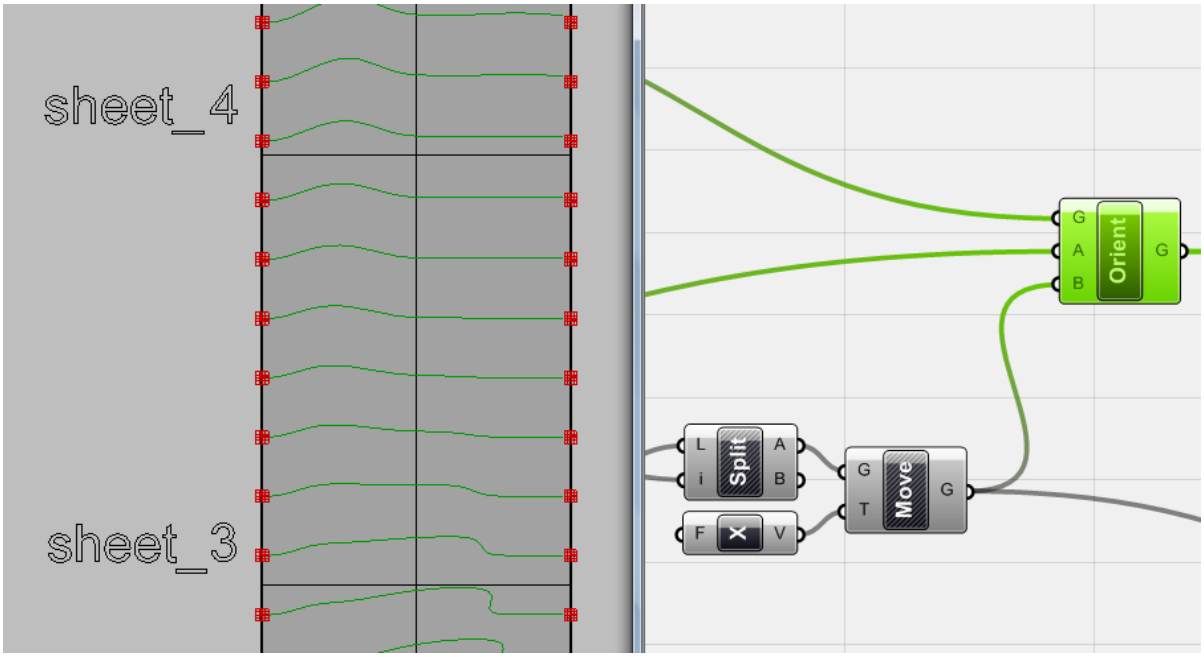


Fig.8.28. nested curves on the cutting sheet.

Making ribs

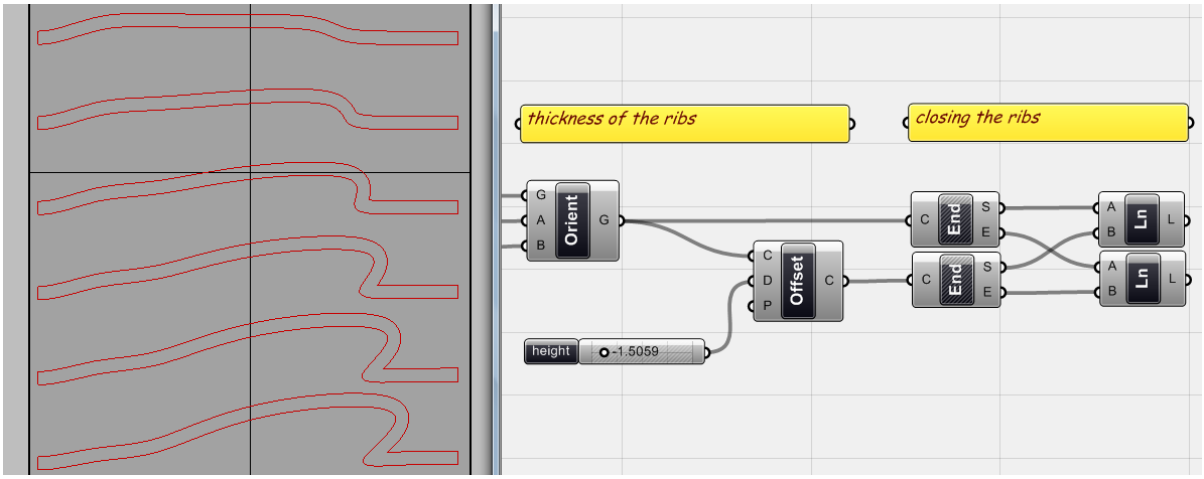


Fig.8.29. After nesting the curves on the cutting sheet, as I told you, because my object does not have any thickness, in order to cut it, we need to add thickness to it. That's why I <offset> oriented curves with desired height and I also add <line>s to both ends of these curves and their offset ones to close the whole drawing so I would have complete ribs to cut.

Joints (Bridle joints)

The next issue is to generate ribs in other direction and make joints to assemble them after being cut. Although I used the same method of division of the bounding box length to generate planes and then sections, but I can generate planes manually in any desired position as well. So in essence if you do not want to divide both directions and generate sections evenly, you can use other methods of generating planes and even make them manually.

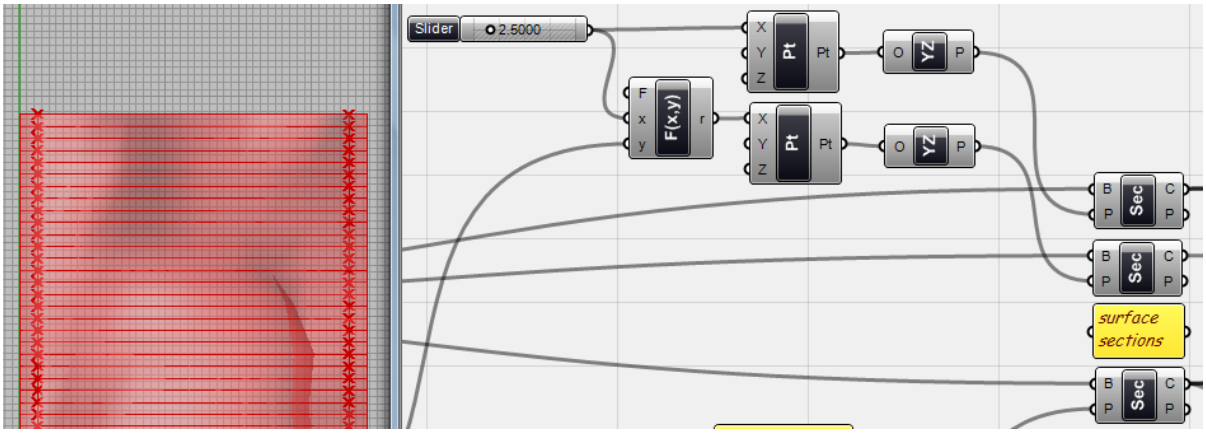


Fig.8.30. As you see here, instead of previously generated planes, I used manually defined planes for the sections in the other direction of the surface. One plane generated by X value directly from <number slider> and another plane comes from the mirrored plane on the other side of the surface (surface length – number slider). The section of these two planes and surface is being calculated for the next steps.

Now I can orient these new curves on another sheet to cut which is the same as the other one. So let's generate joints for the assembly which is the important point of this part.

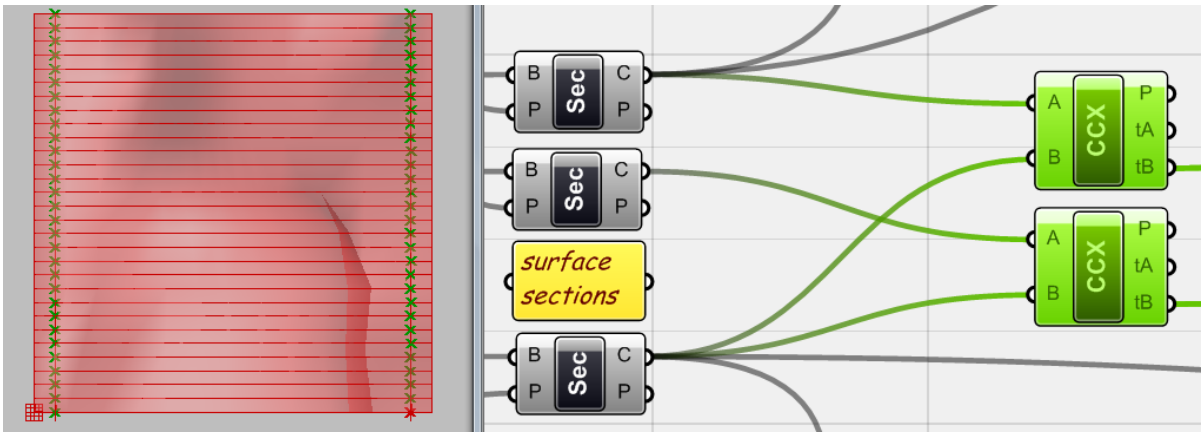


Fig.8.31. since we have the curves in two directions we can find the points of intersections. That's why I used <CCX> components (Intersect > Physical > Curve | Curve) to find the intersect position of these curves which means the joint positions (The <CCX> component is in cross reference mode).

After finding joint's positions, I need a bit of drawing to prepare these joints to be cut. I am thinking of preparing bridle joints so I need to cut half of each rib on the joint position to be able to join them at the end. First I need to find these intersect position on the nested ribs and then draw the lines for cutting.

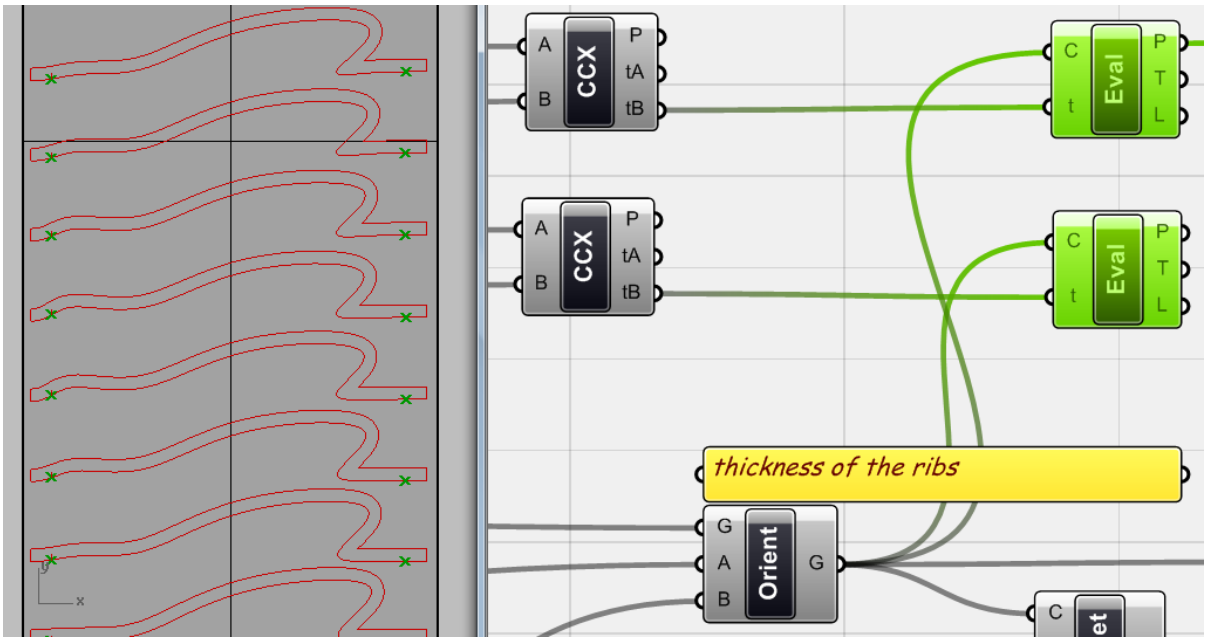


Fig.8.32. If you look at the outputs of the <CCX> component you can see that it gives us the parameter in wish each curve intersect with the other one. So I can <evaluate> the nested or <orient>ed curves with these parameters to find the joint positions on the cutting sheet as well.

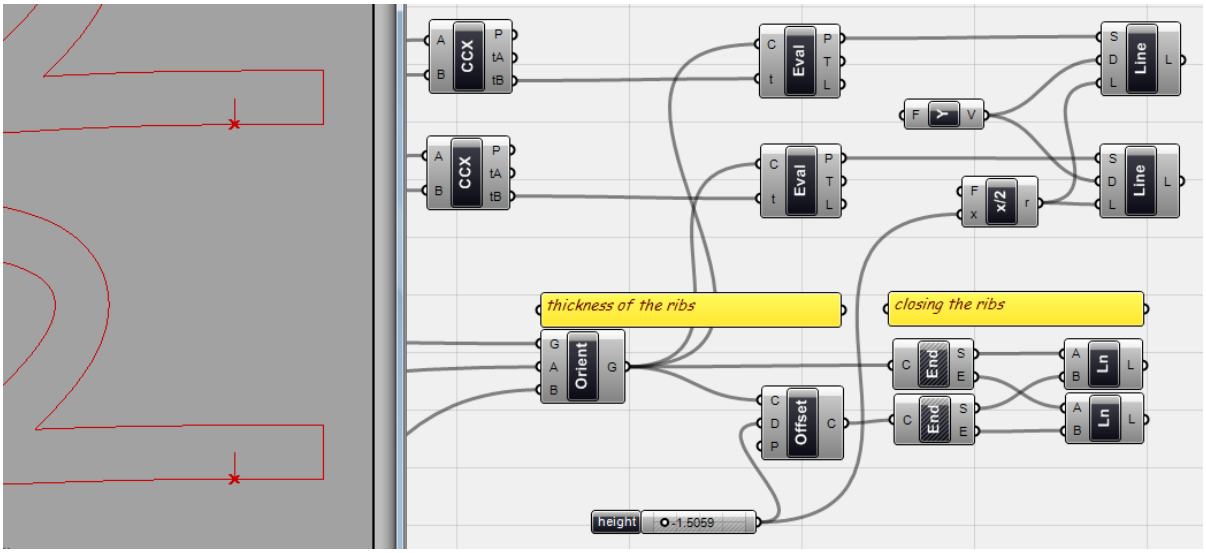


Fig.8.33. Now we have the joint positions, we need to draw them. First I drew lines with <line SDL> component with the joint positions as start points, <unit Y> as direction and I used half of the rib's height as the height of the line. So as you see each point on the nested curves now has a tiny line associated with it.

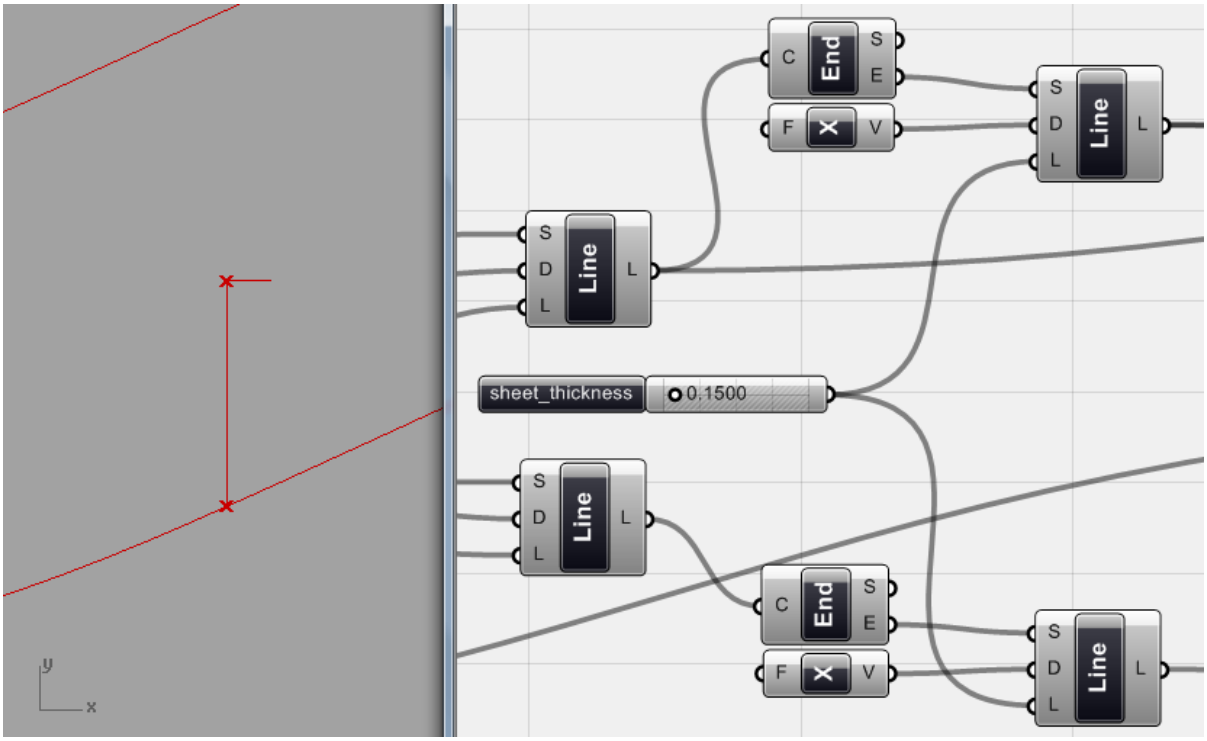


Fig.8.34. Next step, draw a line in X direction from the previous line's end point with the length of the <sheet_thickness> (depends on the material).

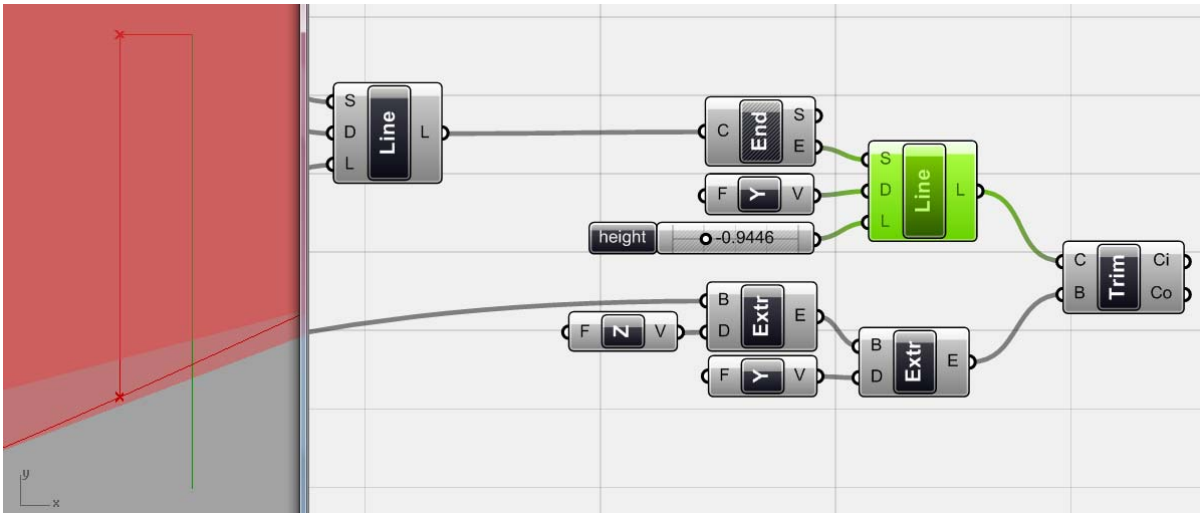


Fig.8.35. This part of the definition is a bit tricky but I don't have any better solution yet. Actually if you offset the first joint line you will get the third line but as the base curve line is not straight it would cross the curve (or not meet it) so the end point of the third line does not positioned on the curve. Here I drew a line from the end point of the second line, but longer than what it should be, and I am going to trim it with the curve. But because the <trim with BRep> component needs BRep objects not curves, I extruded the base curve to make a surface and again I extruded this surface to make a closed BRep. So if I trim the third line of the join with this BRep, I would get the exact joint shape that I want.

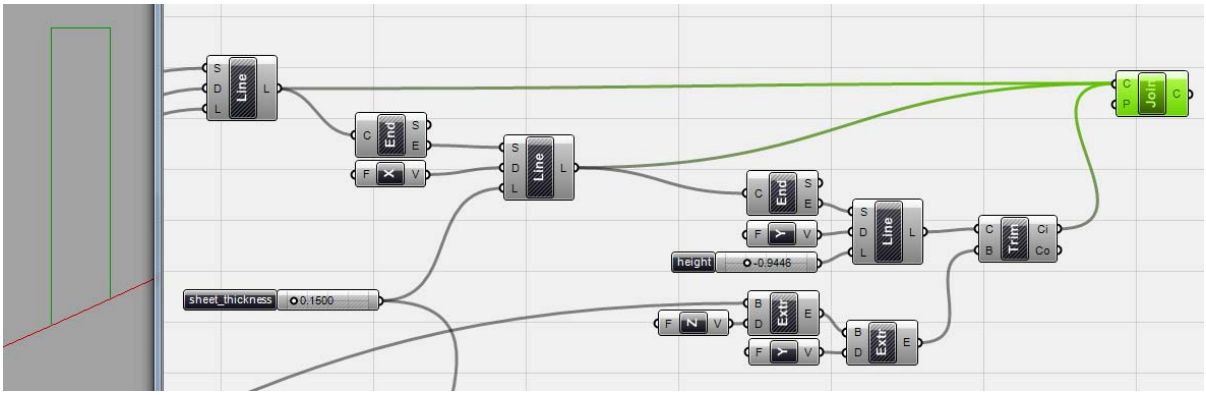


Fig.8.36. Using a <join curves> component (Curve > Util > Join curves) now as you can see I have a slot shaped <join curve> that I can use for cutting as bridle join in the ribs.

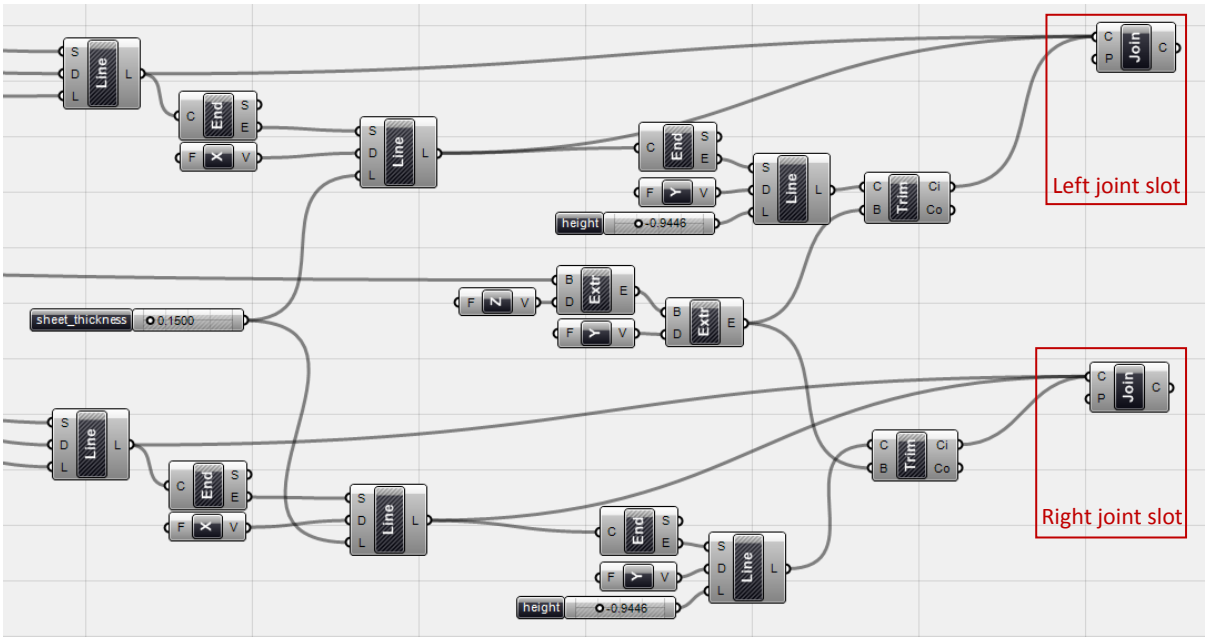


Fig.8.37. I am applying the same method for the other end of the curve (second joints on the other side of the curve).

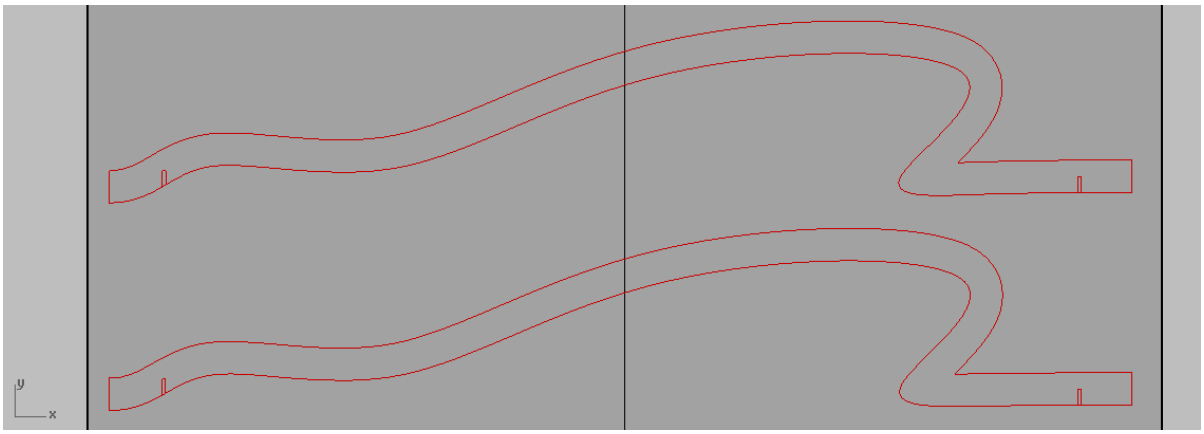


Fig.8.38. Ribs with the joints drawn on their both ends. With the same trick I can trim the tiny part of the base curve inside joint but because it does not affect the result I can leave it.

Labelling

While working in fabrication phase, it might be a great disaster to cut hundreds of small parts without any clue or address that how we are going to assemble them together, what is the order, and which one goes first. It could be simply a number or a combination of text and number to address the part. If the object comprises of different parts we can name them, so we can use the names or initials with numbers to address the parts also. We can use different hierarchies of project assembly logic in order to name the parts as well.

Here I am going to number the parts because my assembly is not so complicated.

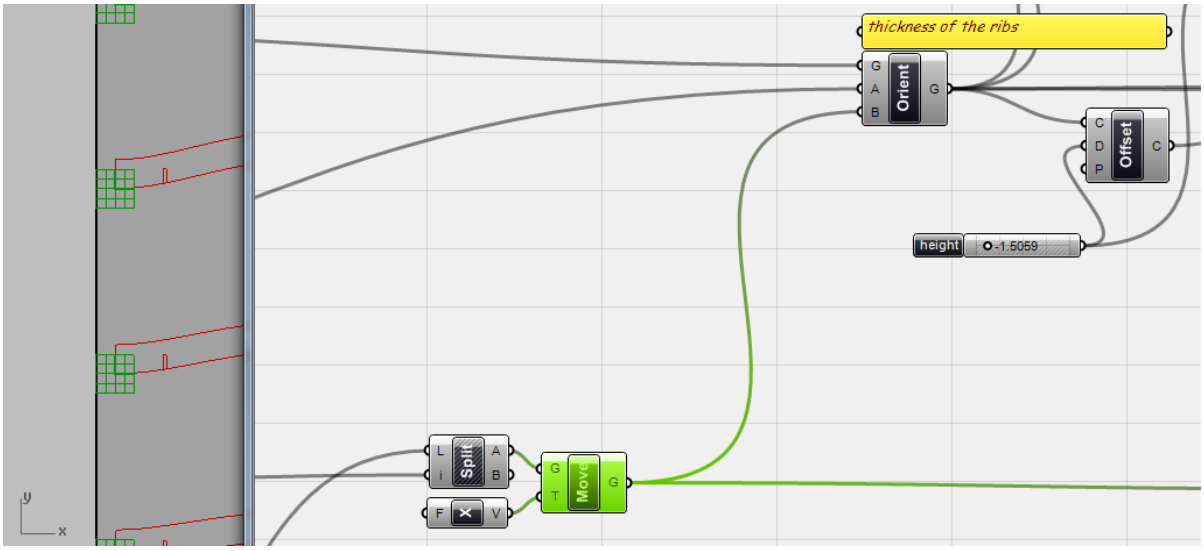


Fig.8.39. As you remember I had a series of planes which I used as the target planes for orientating my section curves on the sheet. I am going to use the same plane to make the position of the text. Since this plane is exactly on the corner of the rib I want to displace it first.

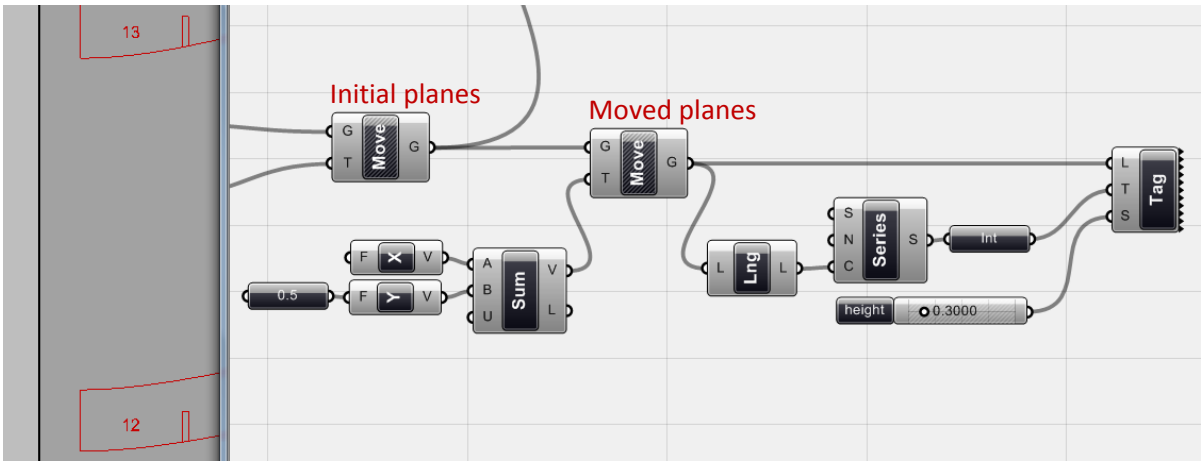


Fig.8.40. I moved the corner planes 1 unit in X direction and 0.5 unit in Y direction (as <sum> of the vectors) and I used these planes as the position of the text tags. Here I used <text tag 3D> and I generated a series of numbers as much as ribs I have to use them as texts. The <integer> component that I used here simply converts 12.0 to 12. As the result, you can see all parts have a unique number in their left corner.

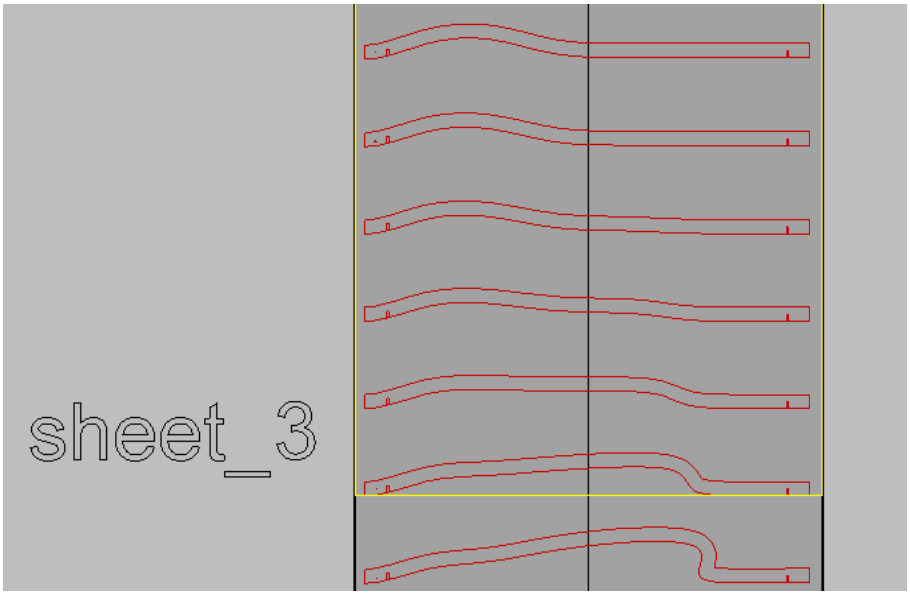


Fig.8.41. I can change the division factors of the cutting surface to compress ribs as much as possible to avoid wasting material. As you see in the above example, from the start point of the sheet_3 ribs started to be more flat and I have more space in between. Here I can split ribs in two different cutting surface and change the division points of each to compress them based on their shape. But because I am not dealing with lots of parts I can do this type of stuff manually in Rhino, all parts does not necessarily to be Associative! Now I have the ribs in one direction, and I am going to do the same for the other direction of ribs as well. The only thing that you should consider here is that the direction of the joints flip around here, so basically while I was working with the <orient> geometry in the previous part here I should work with the <offset> one.

Cutting

When all geometries become ready to cut, I need to burn them and manage them a bit more on my sheets. As you see in Figure 8.42 they all nested in three sheets. I generated three different shapes for the ribs in the width direction of the object to check them out. The file is now ready to be cut.

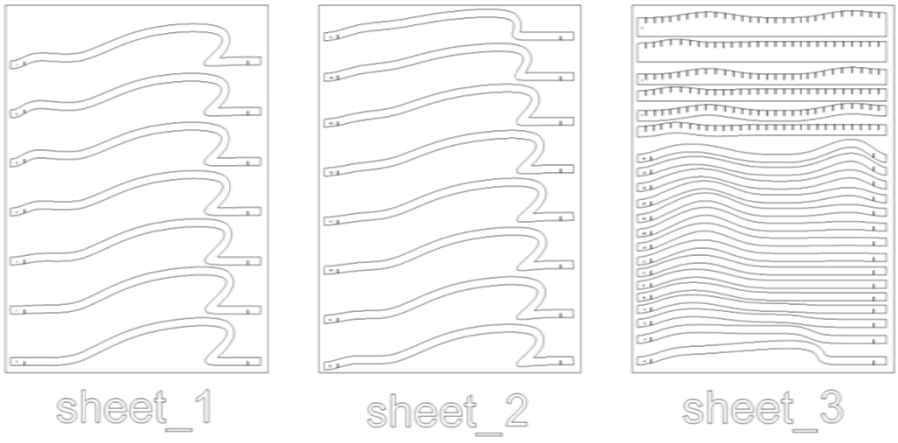


Fig.8.42. Nested ribs, ready to be cut.

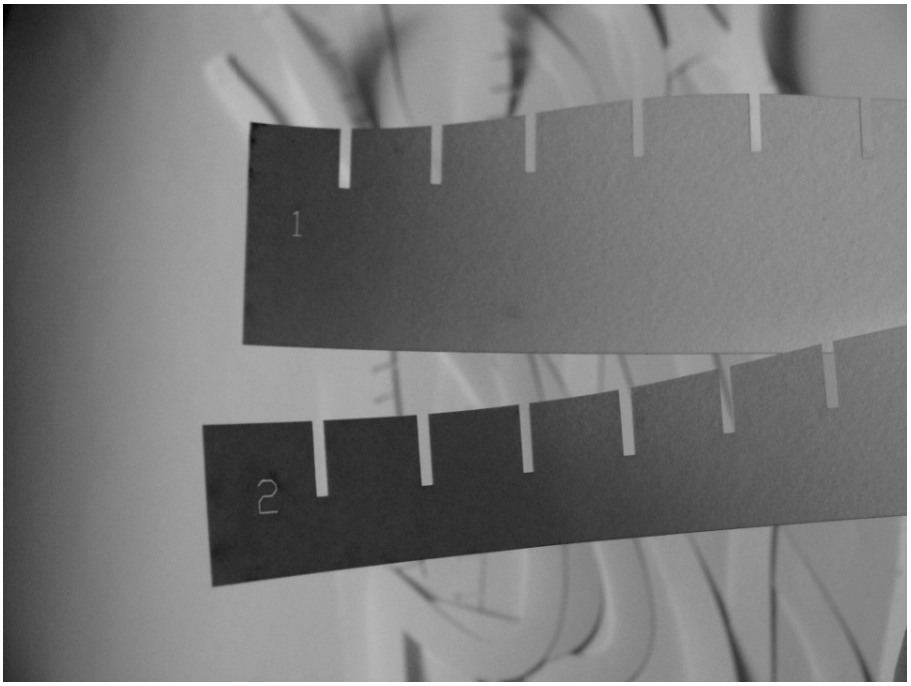
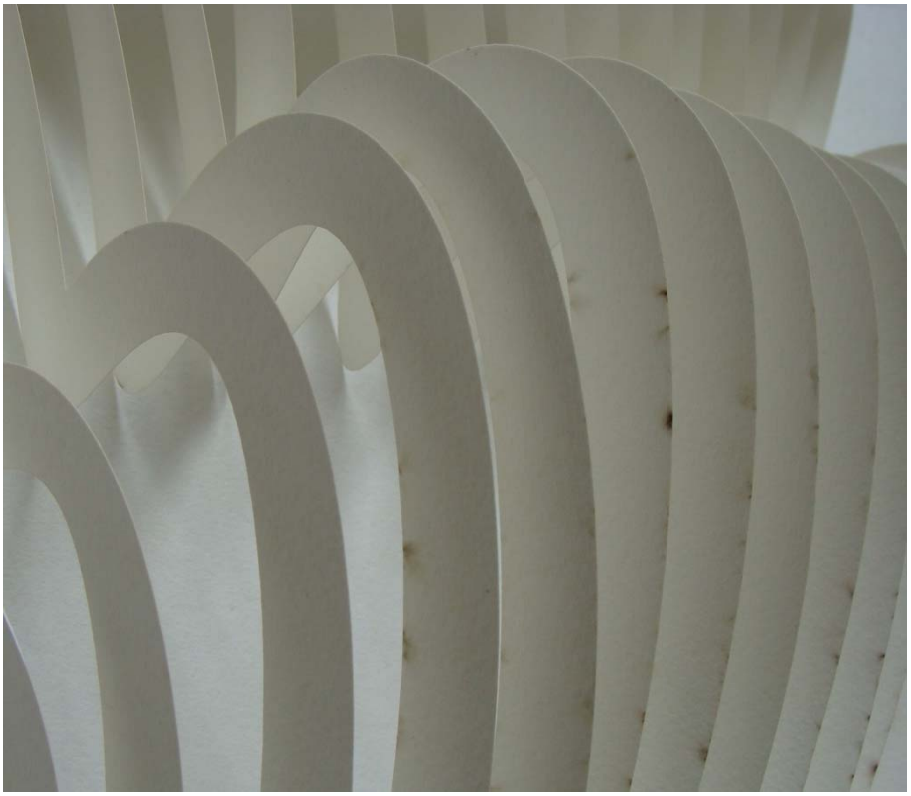
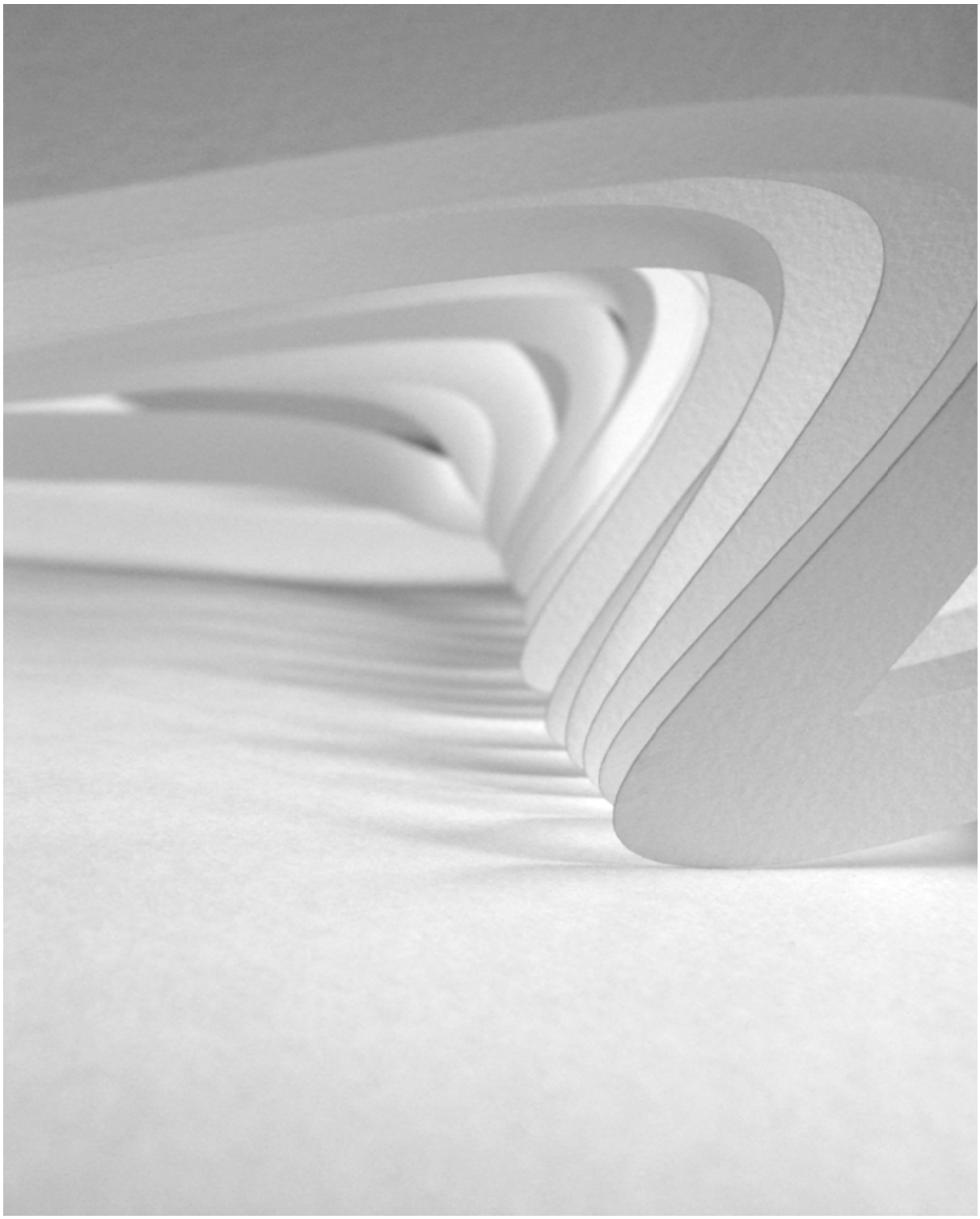


Fig.8.43. Cut ribs, ready to assemble.

Assembly

In our case assembly is quiet simple. Sometimes you need to check your file again or even provide some help files or excel sheets in order to assemble your parts in different fabrication methods. All together, here is the surface that I made.





8.44.a/b. Final model.

Fabrication is a wide topic to discuss. It highly depends on what you want to fabricate, what is the material, what is the machine and how fabricated parts going to be assemble and so on. As I told you before, depend on the project you are working on, you need to provide your data for the next stages. Sometimes it is more important to get the assembly logic, for example when you are working with simple components but complex geometry as the result of assembly.

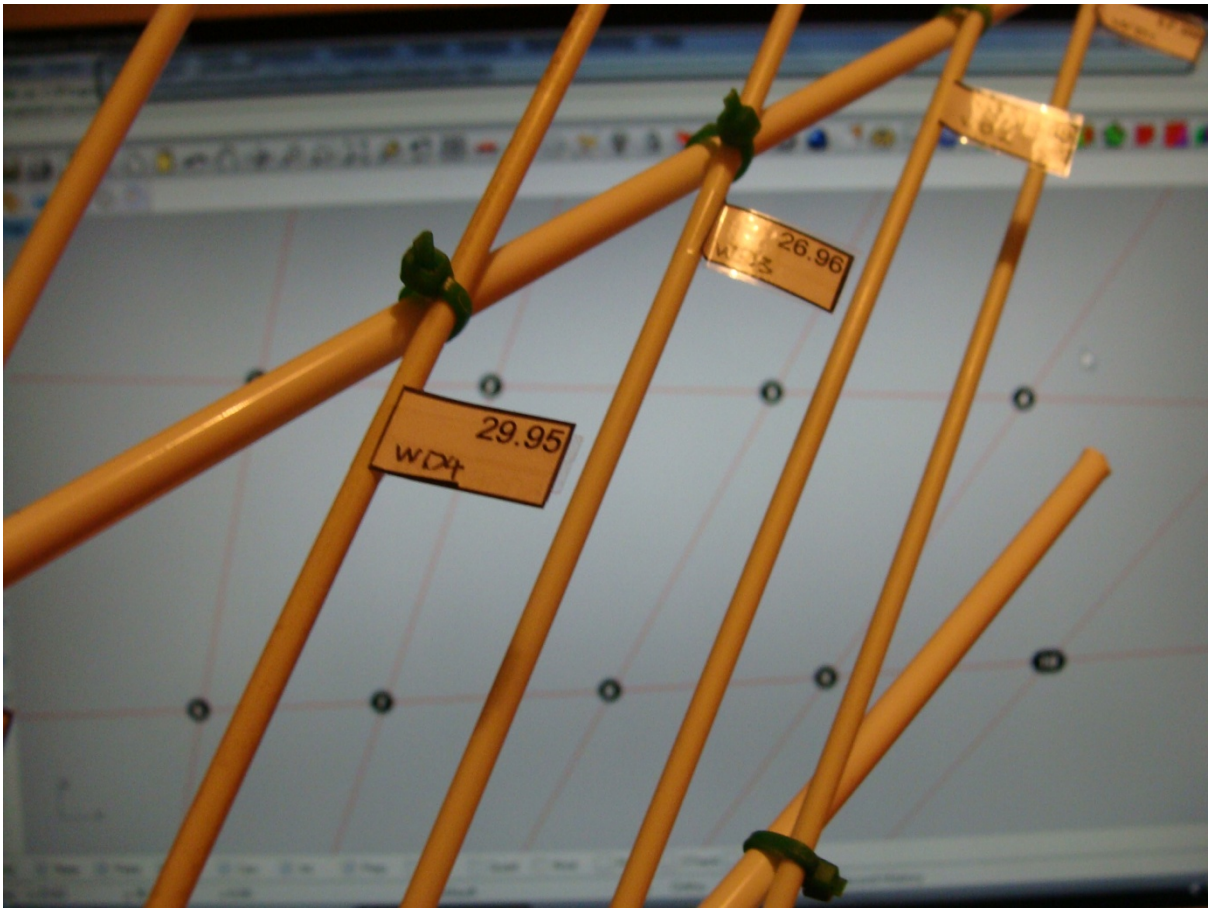
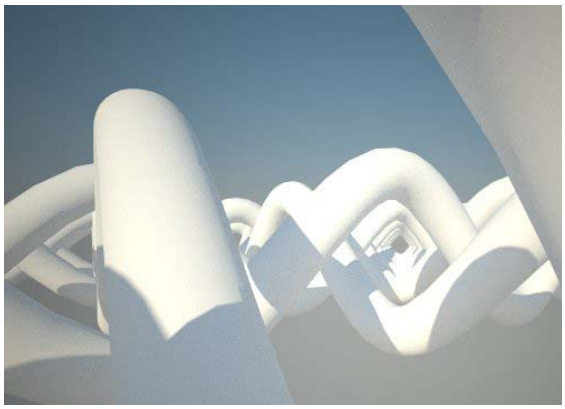


Fig.8.45. Assembly logic; material and joints are simple; I can work on the assembly logic and use the data to make my model.

Chapter_9_Design Strategy



Chapter_9_Design Strategy

Associative modelling is an algorithmic way of dealing with geometry. More than the conventional geometrical objects, with this algorithmic method, now we have all possibilities of computational geometries as well as dealing with the huge amount of data, numbers and calculations. It seems that there is a great potential in this realm. Here the argument is to not limit the design in any predefined experiment, and explore these infinite potentials; there are always alternative ways to do algorithmic modelling. Although it seems that the in-built commands of these parametric modelling softwares could limit some actions or dictate some methods, but alternative solutions could always be brought to the table, let our creativity fly away of limitations.

In order to design something, having a Design Strategy always helps to set up a good algorithm and to find the design solution. Thinking about the general properties of the design object, drawing some parts, even making some physical models, would help to a better understanding of the algorithm so better choice of <components> in digital modelling. Thinking about fix parameters, parameters that might change during the design, numerical data and geometrical objects needed, always helps to improve the algorithm. It would be helpful to analytically understand the design problem, sketch it and then start an algorithm that can solve the problem.

We should think in an Algorithmic way to design Algorithmic.

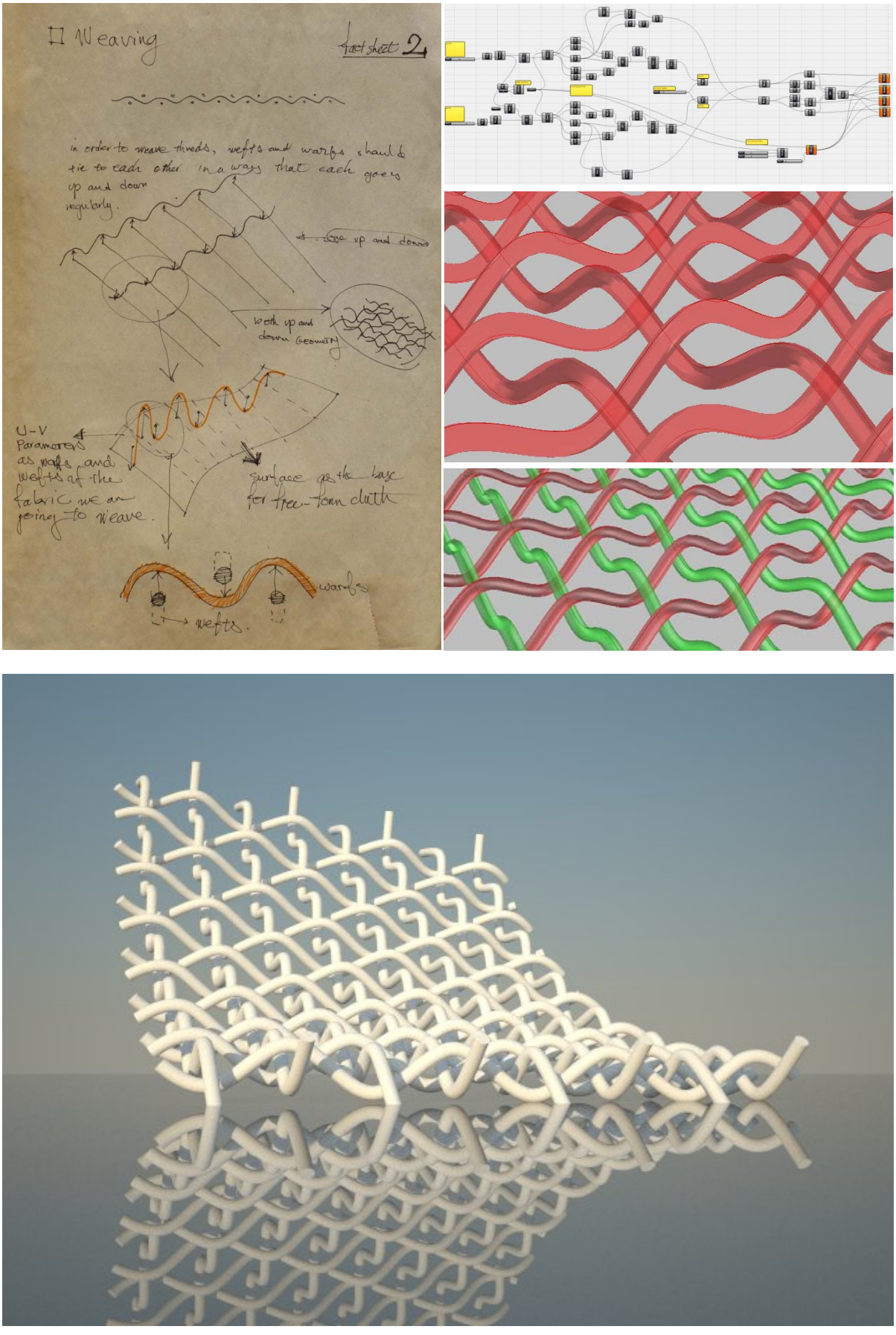


Fig.9.1. Weaving project; From Analytical understanding to Associative modelling.

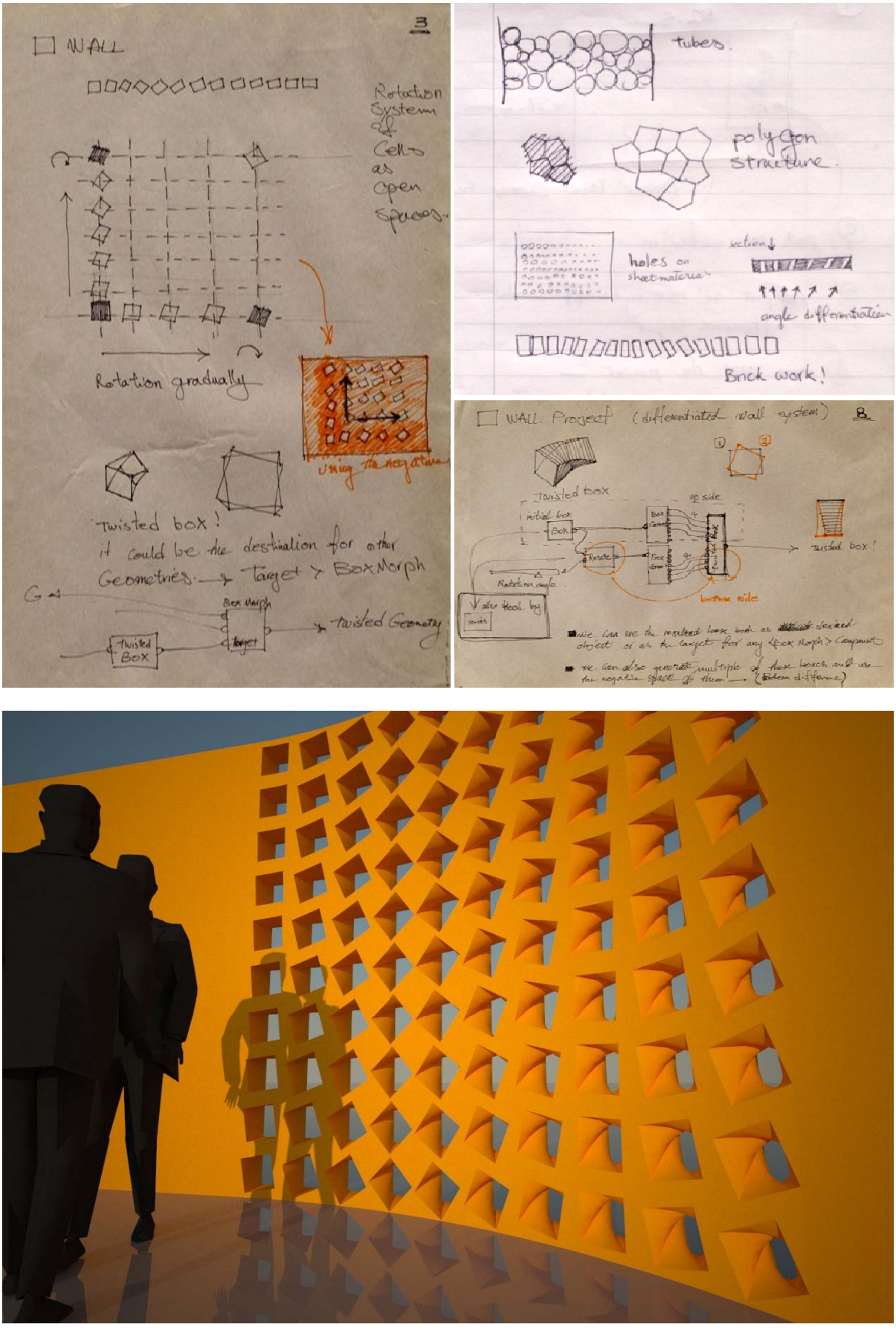


Fig.9.2. Porous wall project; From Analytical understanding to Associative modelling.

Bibliography

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

Hensel, Michael and Menges, Achim, 2008: **'Morpho-Ecologies'**, Architectural Association.

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

Flake, Gary William, 1998: **'The computational beauty of nature, computer explorations of fractals, chaos, complex systems, and adaptation'**, The MIT Press.

Main Grasshopper web page: http://grasshopper.rhino3d.com/

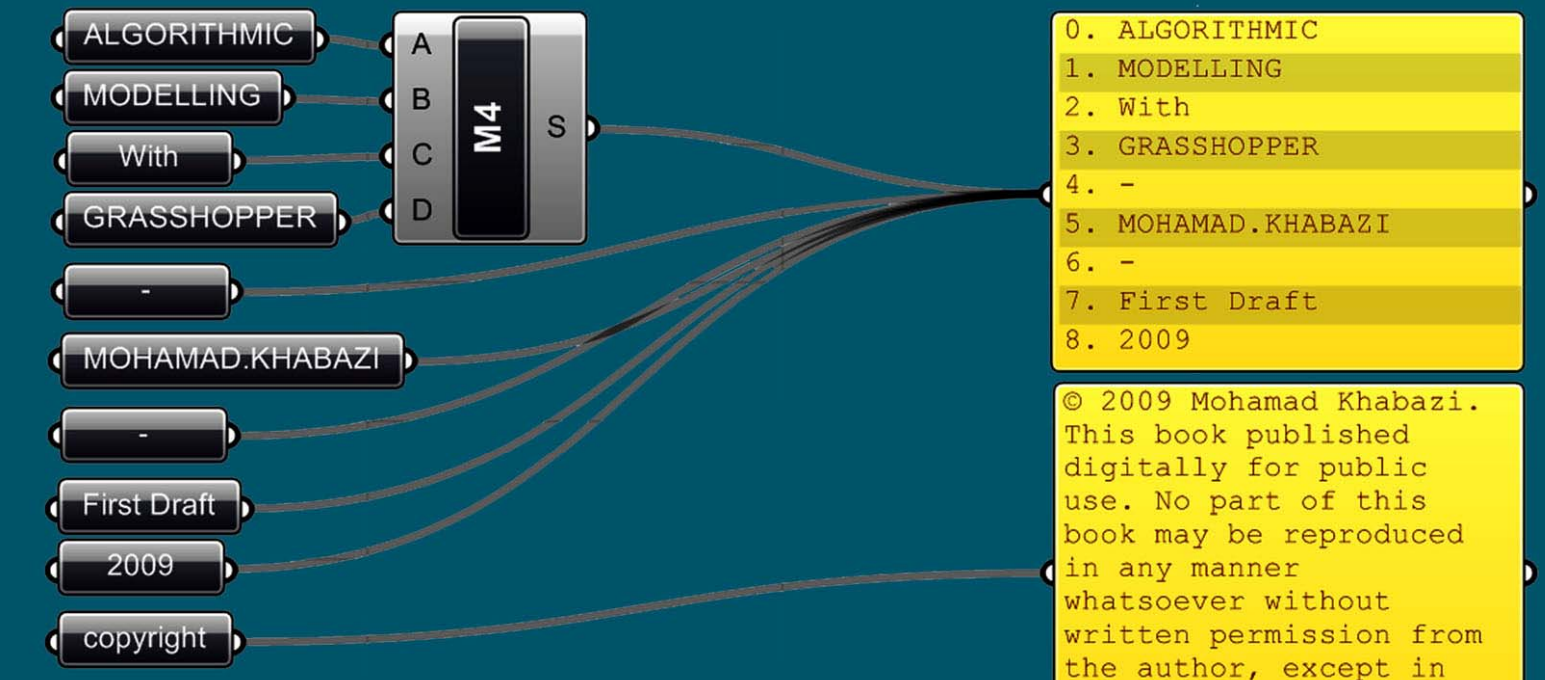
Grasshopper tutorials on Robert McNeel and Associates wiki:
http://en.wiki.mcneel.com/default.aspx/McNeel/ExplicitHistoryExamples.html

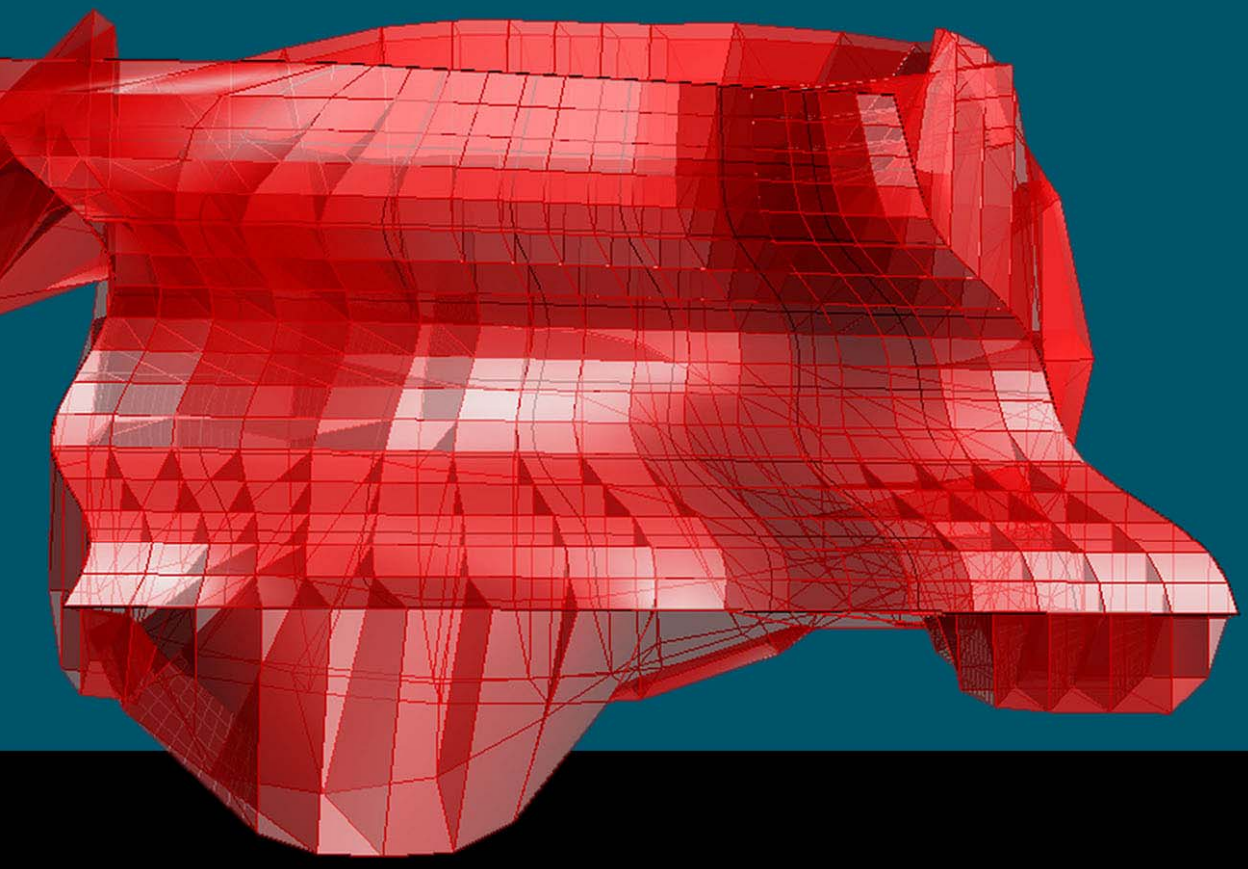
Kilian, Axel and Dritsas, Stylianos: **'Design Tooling - Sketching by Computation'**,
http://www.designexplorer.net/designtooling/inetpub/wwwroot/components/sketching/index.html

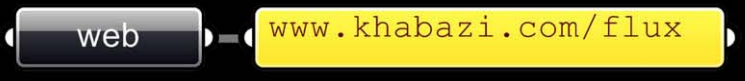
Wolfram Mathworld: http://mathworld.wolfram.com/

Stylianos Dritsas, http://jeneratiff.com/

Notes


ALGORITHMIC
MODELLING
With
GRASSHOPPER
-
MOHAMAD.KHABAZI
-
First Draft
2009
copyright
A
B
C
D
M4
S
0. ALGORITHMIC
1. MODELLING
2. With
3. GRASSHOPPER
4. -
5. MOHAMAD.KHABAZI
6. -
7. First Draft
8. 2009
© 2009 Mohamad Khabazi. This book published digitally for public use. No part of this book may be reproduced in any manner whatsoever without written permission from the author, except in the context of reviews. m.khabazi@gmail.com




web
www.khabazi.com/flux