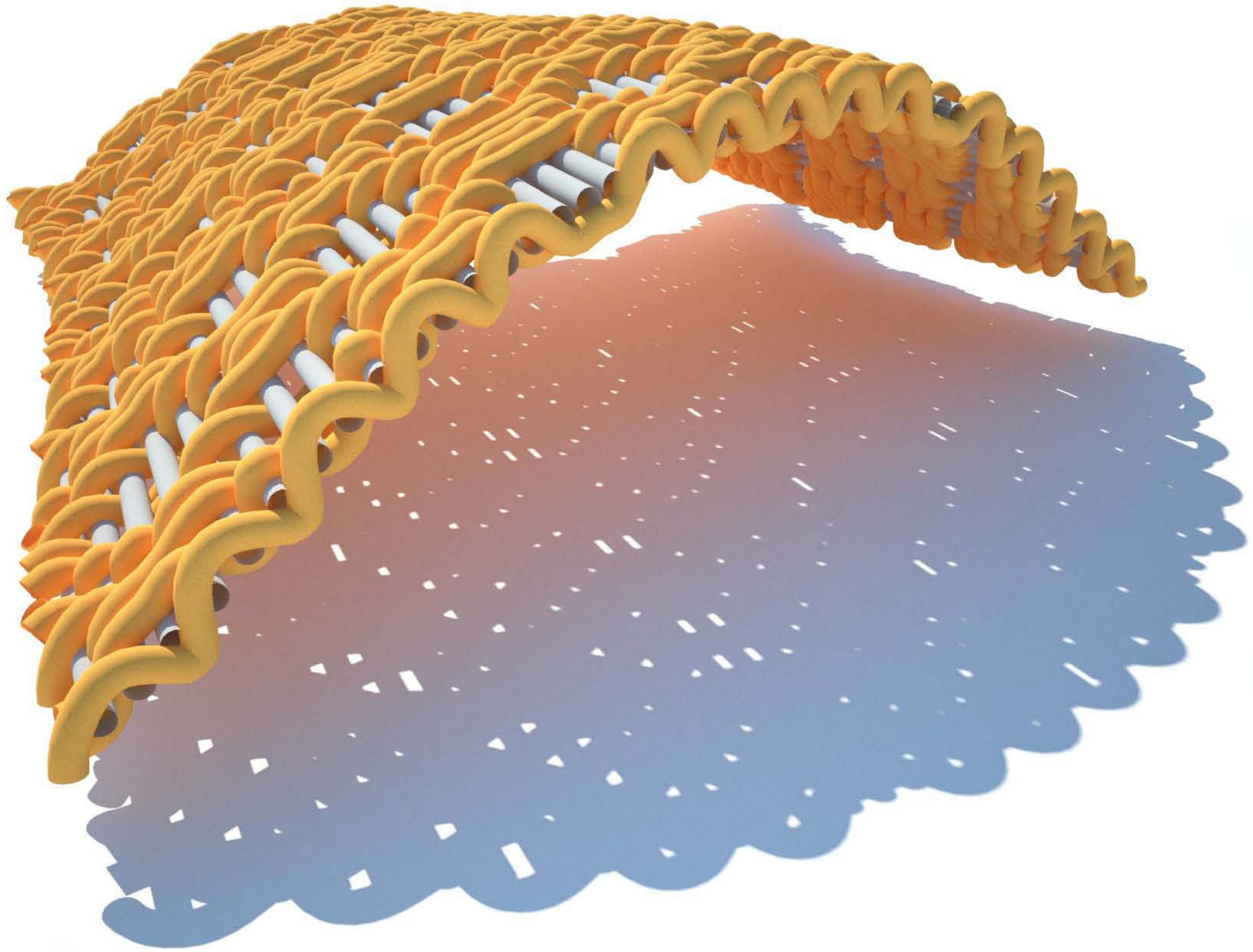


GENERATIVE ALGORITHMS

CONCEPTS and EXPERIMENTS: WEAVING

ZUBIN KHABAZI

Generative Algorithms

Concepts and Experiments

1_Weaving

Zubin Khabazi

To see the latest updates visit my website or for enquiries contact me:

www.MORPHOGENESISM.com

zubin.khabazi@googlemail.com

Generative Algorithms, Concepts and Experiments

Introduction to the Series

‘Generative Algorithms’ which published on-line, was aimed to bring forward subjects and concepts on geometrical issues related to architectural design and some basic experiments using parametric modeling and algorithmic approach with Grasshopper. Since then, readers of the book while got involved with different projects, have asked for help on topics, methods, solutions and definitions and proved that Grasshopper is fast growing and more detailed resources to discuss contemporary subjects of design are needed. That’s why I started these series of ‘Generative Algorithms, Concepts and Experiments’ through which I want to share some specific areas of knowledge and my researches and achievements in design with algorithmic approaches.

The subject and method comes from a series of global communication with readers, while I realized a gap between Themes, approaches, and methods in geometry with final products. We can simply find lots of images from these algorithmic experiments in weblogs and websites because of the availability of tools in digital realm (CAD and Algorithmic/Parametric 3D softwares) but the easiness of this production, or their availability, sometime causes confusion, sometime shows lack of sufficient knowledge about the inherent properties of these systems or possibilities of their material outcome / Fabrication and also performances which can be achieved from them. In this situation, studies on the subject, physical experiments and observations, and searching for different algorithmic solutions could be the minimum requirements to understand these systems and to be able to implement them into our design fields.

In this series, ‘**Generative Algorithms, Concepts and Experiments**’, I am trying to search for concepts and methods, combine physical and digital experiments on topics that seem ‘Generative’ as prototypes for possible applications in architecture, to extend the catalogue of available systems and methods in design. I hope these experiments benefit your design issues or line of research, as mine.

Good luck!

Zubin M Khabazi (Master of Emergent Technologies and Design / AA)

_Notice: Going all the way through to the details of projects, I assumed you already have the basic knowledge of how to work with Grasshopper which I believe by now, you have studied ‘Generative Algorithms’ and you know what does ‘<Component>’, ‘Domain’ or ‘Vector Amplitude’ mean, otherwise check the ‘notes’ section of this book.

Contents

1_Generatice Algorithms, Concepts and Experiments_1_Waeving

1_1_Introduction

Parametric Geometry is a powerful tool and even more, a methodology in design, in such a way that claimed as a style: Parametricism. “*We pursue the parametric design paradigm all the way, penetrating into all corners of the discipline. Systematic, adaptive variation, continuous differentiation (rather than mere variety), and dynamic, parametric figuration concerns all design tasks from urbanism to the level of tectonic detail, interior furnishings and the world of products…Contemporary avant-garde architecture is addressing the demand for an increased level of articulated complexity by means of retooling its methods on the basis of parametric design systems. The contemporary architectural style that has achieved pervasive hegemony within the contemporary architectural avant-garde can be best understood as a research program based upon the parametric paradigm. We propose to call this style:* **Parametricism**.”[1]

The notion of parametricism started with the advances in computational geometry and digital tools (both software and hardware) and developed through lots of design projects in CAD spaces and resulted in loads of images and concepts in digital space which was, let’s say, first phase of the parametricism. But soon after that, pursuing these concepts and ideas in practice, potentials of this new method leads architects to transfer properties of simple material systems into basic computational constructs in order to use their potentials as generative logics and small scale components in reality. Under the influence of methods and techniques of parametricism, these systems proliferated on larger spans and set up a new system of material outcomes based on real physical, small scale constructs. The shift that we encounter in parametricism now, is from Top-Down, result-based design to a Bottom-up, system-development approach which embed properties of material system as parameters of larger scale products which demonstrate qualities of underlying material system [2] and sometimes more. [3]

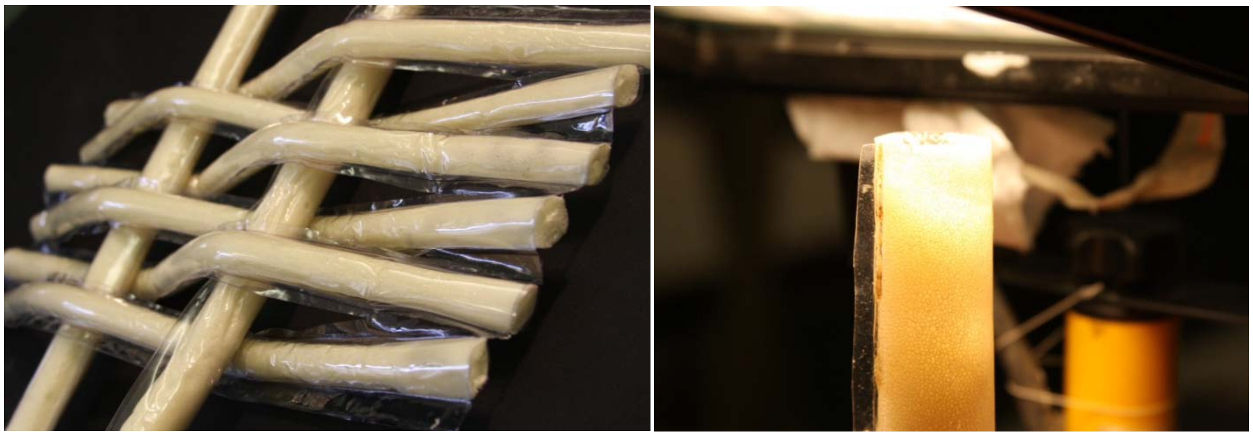


Fig.1.1. Experiments with ‘Hardened Pneumatics’ leaded to explore potentials of woven products and weaving algorithms in order to transfer material performances into parametric computations.

In order to transfer a material system into digital parametric realm, one needs to study and observe the system in different levels to understand its behavior, its inherent properties which is affected by material, geometry, shape, … and if needed, its structural behavior, physical and chemical specifications and so on and so forth. So this is not just transformation of geometry, but it encounters the implementation of material behavior into a digital product which is parameterized by its own properties. That's why careful studies and analysis is tied up with this level of parametricism and 'Generative Algorithms, Concepts and Experiments' also follows this method by a research and design approach.

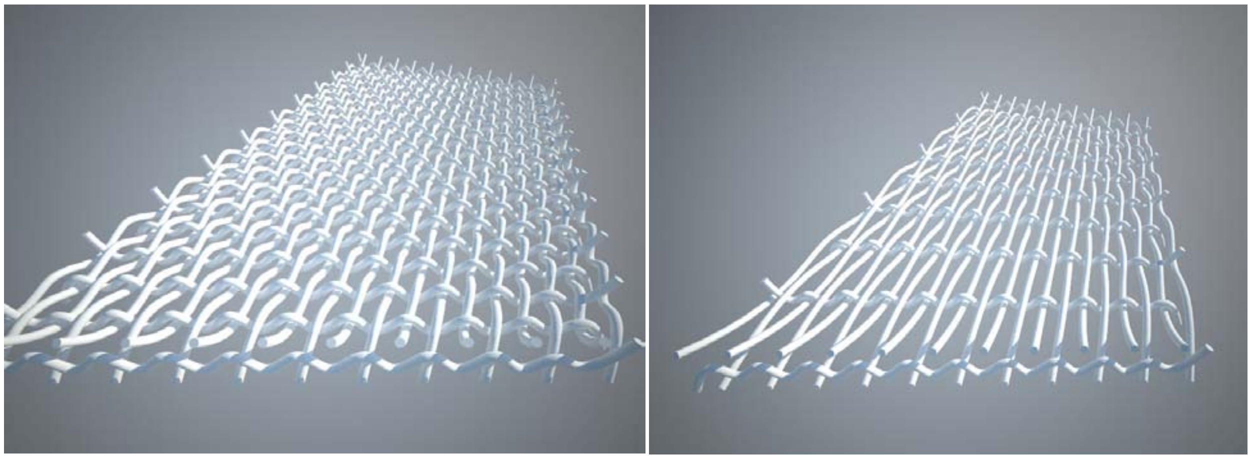


Fig.1.2. Development of digital parametric models based on researches on 'Hardened Pneumatics', experiments at EmTech Studio, AA school of architecture, London.

1_2_Weaving

Among the broad catalogue of material systems, handmade old-to-new products, biological and natural systems and so on, weaving is one the concepts that sounds easy enough to be explored as a basic construct that can be used in parametric notion, and complicated enough, to be the generator of complex hierarchical systems. Because of the intrinsic properties of the fabric which makes it free-form object, yet comprised of small continuous element, its transformation into digital algorithmic space would be helpful for development of similar free-form surfaces with the same properties and a generative logic behind.

Fig.1.3. Fabrics are close to our body in everyday life. They are flexible, soft and delicate with different patterns, colors and they accept various methods of design. While they cover our bodies, they have potential to cover spaces as well; without any prediction about functional or structural ability, their system and concept worth looking closer, in detail.

Weaving described as a systematic interlacing of two or more elements to form a structure [4]. More specific, in fabric or cloth, there are sets of parallel yarns called Warps that are being interlaced by a filling yarn called Weft in a process of weaving. This process has been done by a device called **'Loom'**. According to Webster, a Loom is 'a frame or machine for interlacing at right angles, two or more sets of threads or yarns to form a cloth. [5]

The Process of weaving, using a loom machine can be described in a repetitive three stage operation: 1. First stage is Shedding. Shedding is the rising of a group of warps (simplest is every other yarn) to make a Shed (an empty space between raised and unraised yarns) through which the filling yarn (weft) passes with a device called Shuttle. The yarns are hung by Heddles which are connected to Harnesses that move up and down and make sheds repeatedly. 2. Second stage is Picking which refers to the single crossing of the shuttle (that contains the weft yarn) from one side of the loom to the other side. This part lays down one layer of weft between raised and unraised warps. 3. Third stage is Battening. In this stage another frame called Reed, presses or battens the filling yarn (weft) to make the fabric. It tightens the weft into the edge of fabric (woven part) and not-yet-woven warps. This line between the woven fabric and the rest of warps called Fell. The rest of the process is the same. The harnesses should move (at least one up, one down) and a new shed would be created and picking and battening and so on. [6]

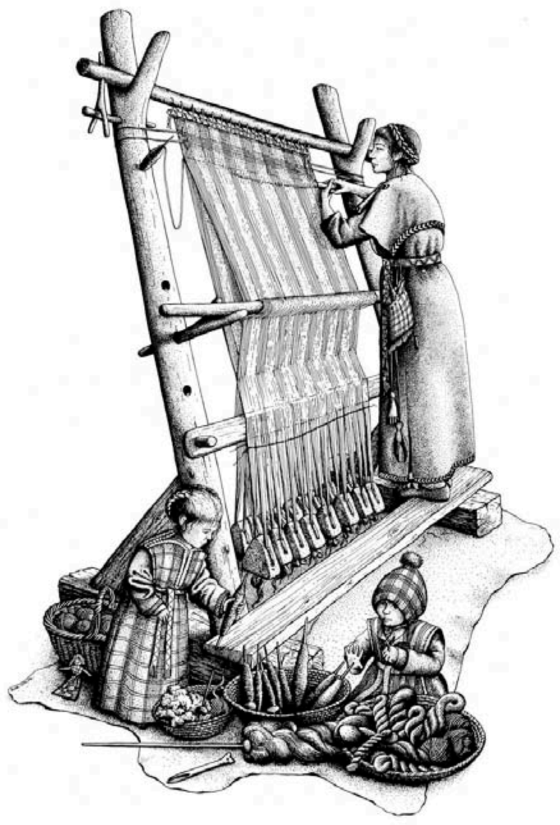


Fig.1.4. Loom machine, Warps and Wefts and a woman who weaves fabric !

Based on techniques and machineries, different types of textile can be woven. Three basic types of textile weaving are Plain weave, Satin Weave and Twill weave. For example it depends on the number of harnesses that can raise the warps at each time or the offset pattern of wefts when take turn to go under or above warps.

Weaving is not limited to fabrics. Basketry is another branch which is weaving of natural fibers like leaves to create baskets and bags and other types of holding and containing tools. Weaving could be range from smooth fabrics for clothes up to tough ones for making tents and harder ones in baskets and even more in walls and fences with metals. Today, more than just type or material, there are other functions that can be achieved by fabrics like E-Textiles, which are aimed to perform distributed sensing or computing actions in small scales inside the structure of a textile. [7]

2_Loom Project

The aim of this chapter is to set up an algorithm (using Grasshopper as platform) that works as a loom machine and weaves virtual textiles. In order to set up the algorithm, to simulate textile, concentration on operation of the loom is needed because properties of the final product (Textile/Fabric/Cloth) are dependent on this operation. In fact various types of fabrics are associated with looms which work differently and result in multiple products. Investigations of weaving and loom machines started by studies and experiments with a simple plain-weave loom machine which would result in a simple Criss-Cross pattern.

Fig.2.1. 'Weaver at the loom' by Vincent Van Gogh.

2_1_Where should start?

The algorithm is aimed to be versatile, and not limited to weave flat-tensioned fabrics. This concept wanted to be customized, used in different situations and as a generative construct for various purposes. So what is the final product and how final geometry can fits into that?

Textiles are usually flexible, free-form objects. They can get any shape based on their positioning. It is noticeable that as non-elastic solid objects, creases and folds appear when fabrics get complex forms, different than their basic initial flat shape, but even considering these folds, let's define them as free-form surfaces. In geometrical term, let's define the general form of textile (final state of product) as a NURBS surface in the algorithm.

It should be noticed that any form of surfaces might not be achievable by a single piece of fabric because of the creases and folds that mentioned before and to get special forms, Cutting-Pattern needed. This is what tailors do when they want to make dresses from fabric. But in order to set up an algorithm here, since there is not any fabric and the algorithm aimed to weave it, weaving any free-form but one piece NURBS surface assumed possible.

2_2_How should weave?

Textile as mentioned, in its simplest construct, has two sets of parallel yarns, which are orthogonal and interlaced into each other. Looking closer into its structure, small wavy threads can be seen going up and down of other perpendicular sets of threads endlessly. Based on the precision and smoothness of the textile, wavy pattern of threads should be visible in warps, and wefts should be straight, but in rough fabrics or for example in baskets, elements are both wavy, interweaving each other.

Yarns are basic ingredients of weaving textiles. Yarns 'consists of fibrous matter as twisted together during the process of spinning' [8]. Fiber materials can be natural or synthetic. They include wool, linen, cotton, silk, acrylic, polyester and so on. Slenderness, smoothness, and other properties of yarns affect the behavior of the textile.

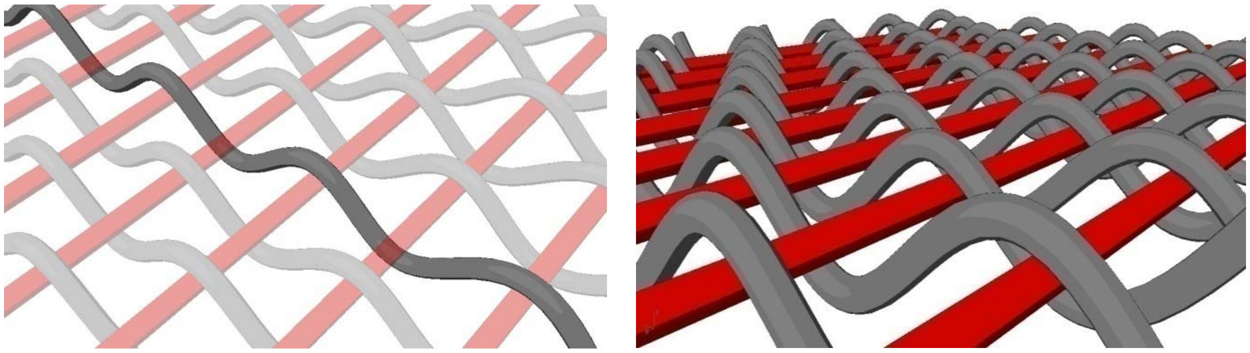


Fig.2.2. Any single warp (if wefts imagined as straight geometries) has a wavy form while through its corrugation goes above and under the wefts.

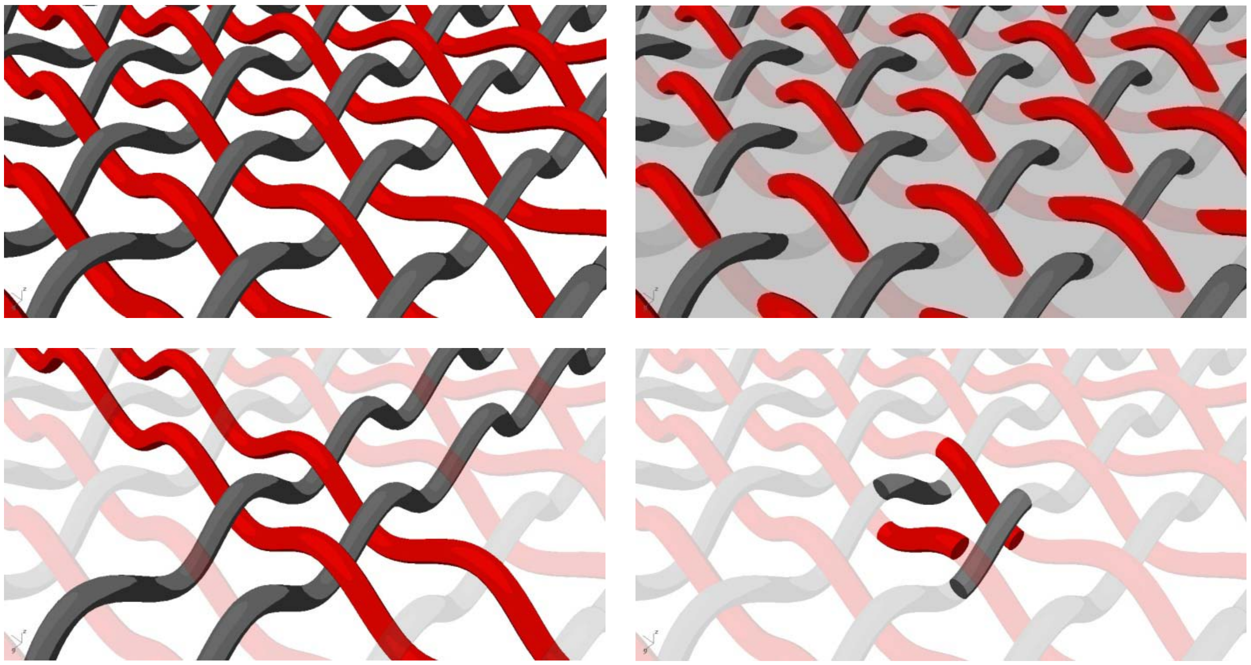


Fig.2.3. Geometrical analysis of waved warps and wefts in a digital textile construct. Both warps and wefts are corrugated forms which go above and under each other repeatedly.

It seems that if the algorithm generates wavy objects (like pipes or cylinders) upon the final-state NURBS surface of the algorithm, the product meets the basic qualities that one might ask for designing a digital textile.

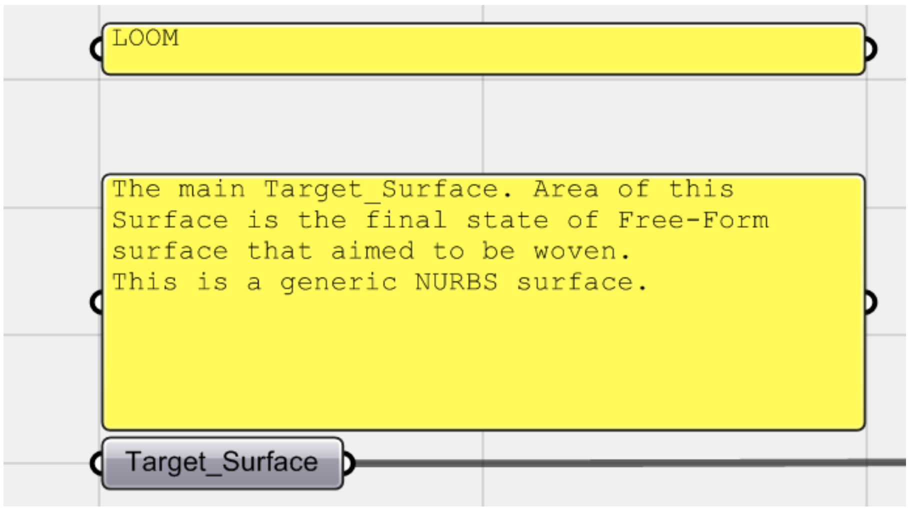
2_3_The Loom Algorithm

Let's start the algorithm by an overall view to its process and flow of information. The input geometry which is a NURBS surface should be introduced to the algorithm. Warps should be generated in its U direction and wefts in V direction. Each warp or weft is look like a sine graph, a corrugated curve or equivalent solid geometry. In order to draw these curves, a list of points needs to be provided for each one. These points are distributed across the surface in its certain U and V iso-curve positions, and goes up and down the surface in its normal direction. Points would move repeatedly in an order like one point up, next down, then up, down, up, down… and goes on to the edge of the surface. There are two important points in their structure:

1. Every warp (or weft) if started with a point UP, the next row would start with a point DOWN. This means half a module shift of corrugation in each row is happening.

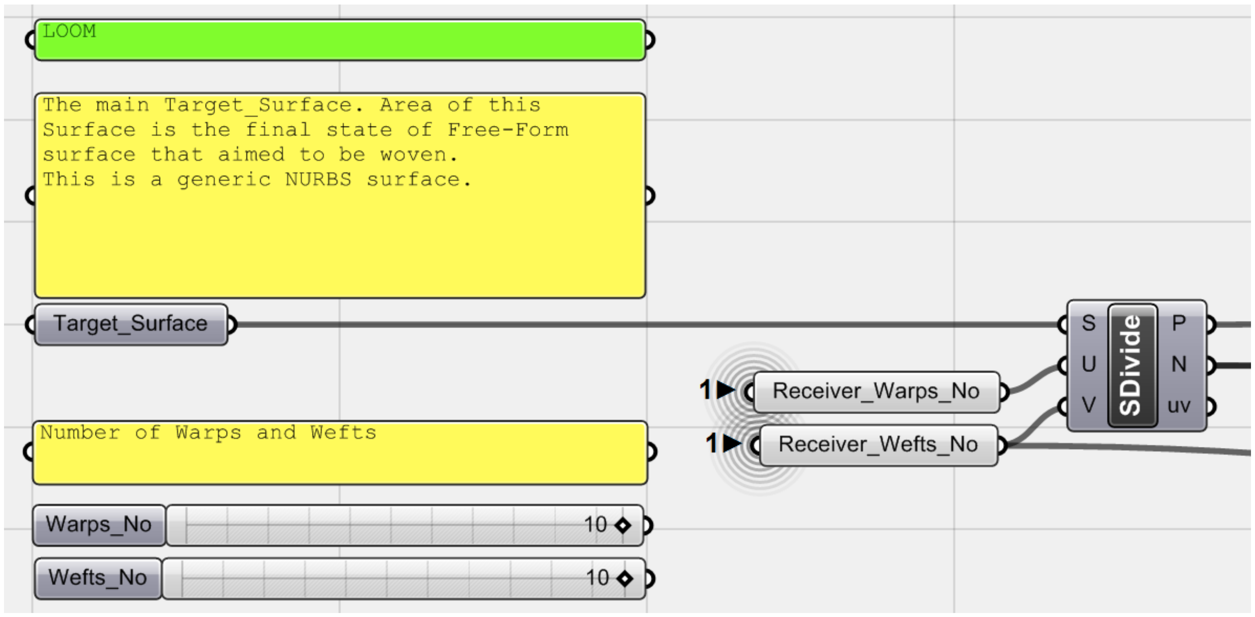
2. At any point on the surface, if the point moves **+A** in the direction of the surface normal to generate a point for a warp, it should move **–A** to generate a point for associate weft on that point (and vice-versa).

Based on the above conception, the algorithm in Grasshopper can be developed as follows:


LOOM

The main Target_Surface. Area of this Surface is the final state of Free-Form surface that aimed to be woven.
This is a generic NURBS surface.

Fig.2.4. To start the algorithm, as mentioned before, a <Surface> geometry component is needed to introduce the general form of final digital textile to canvas as a NURBS surface. It renamed to <Target_Surface>.

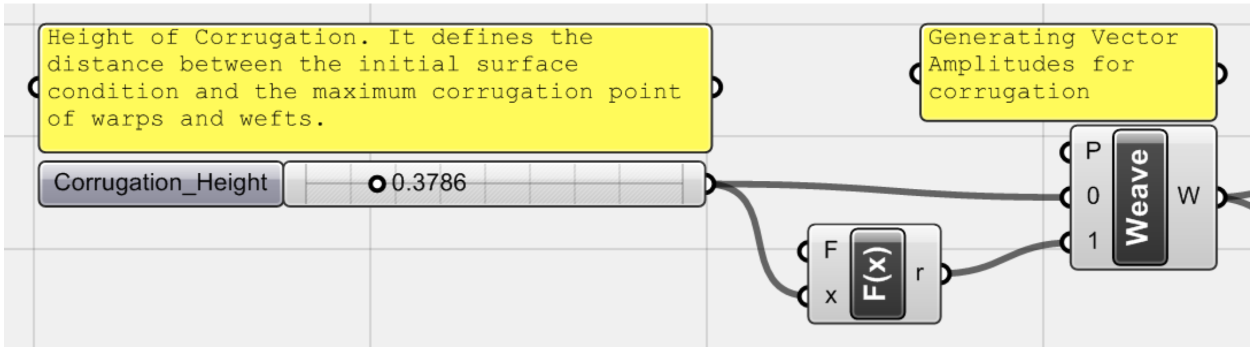


LOOM

The main Target_Surface. Area of this Surface is the final state of Free-Form surface that aimed to be woven.
This is a generic NURBS surface.

Number of Warps and Wefts

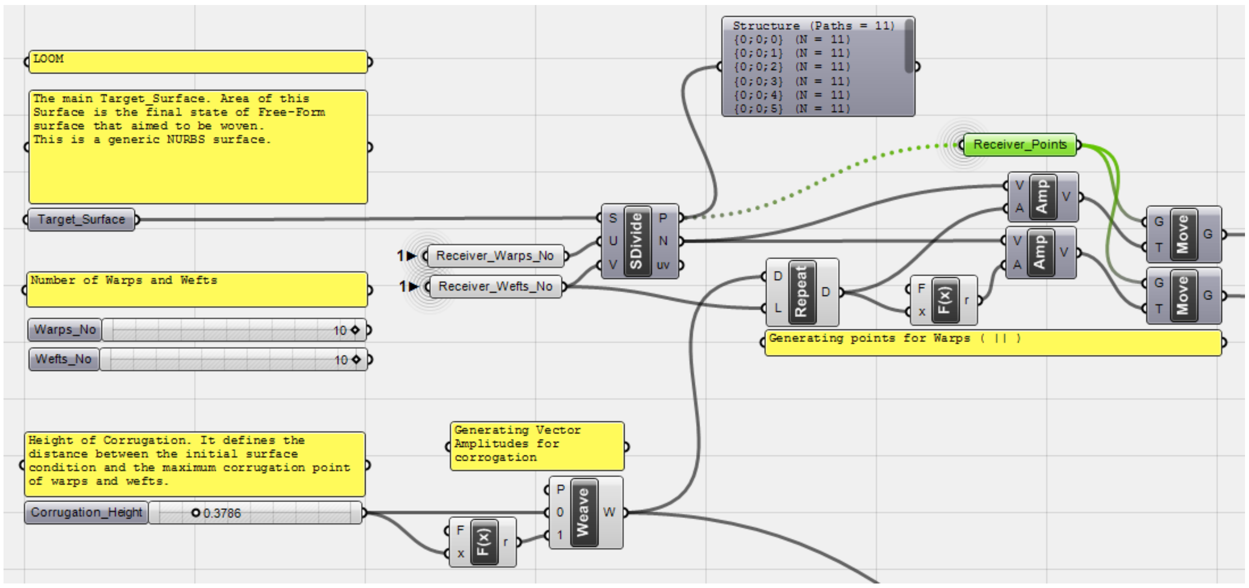
Fig.2.5. The <Target_Surface> is <Subdivide>d based on the number of warps and wefts, defined by two <number sliders> and connected with two <Receivers>. With this component, ‘Division Points’ and ‘Normals’ of the surface at those division points are accessible.



Height of Corrugation. It defines the distance between the initial surface condition and the maximum corrugation point of warps and wefts.

Generating Vector Amplitudes for corrugation

Fig.2.6. As mentioned earlier, division points have to move above and under the surface to generate points for warps and wefts. To do that, group of vectors with a value that is <Corrugation_Height> should be generated (this value defines the amount of displacement of points from the surface). The <function> component here is F(X) = -X (which sets the displacement value in other direction of the surface). Here by a <Weave> component a group of numbers comprised of displacement values or corrugation height in positive (+) and negative (-) values are generated.


LOOM

The main Target_Surface. Area of this
Surface is the final state of Free-Form
surface that aimed to be woven.
This is a generic NURBS surface.

Number of Warps and Wefts

Height of Corrugation. It defines the
distance between the initial surface
condition and the maximum corrugation point
of warps and wefts.

Generating Vector
Amplitudes for
corrogation

Generating points for Warps (||)

```
Structure (Paths = 11)
{0;0;0} (N = 11)
{0;0;1} (N = 11)
{0;0;2} (N = 11)
{0;0;3} (N = 11)
{0;0;4} (N = 11)
{0;0;5} (N = 11)
```

Fig.2.7. ***Warp Generation***. *With this part of the algorithm all points needed for curves as axis of threads are generated. A <Repeat> component, as it sounds, repeats the pair of displacement positive and negative values. It repeats data with the number of wefts (the numbers of points in each warp is equal to number of wefts). These values used to set the <amplitude> of normal vectors of target_surface once directly and another time by their negative values (F(x) = - x). That's because to generate another set of curve's points whose start point is opposite. Although it generates two sets of points, for each warp, since for each row, the position of first point should be different than previous and next one (if previous and next one are above the surface, it should be under the surface) so by this technique, point sets are generated for both situations but then would be <Cull>ed every other one to get the desired curves with opposite start points. Finally all subdivision points of the surface (Via <Receiver>) are moved with both vector groups.*

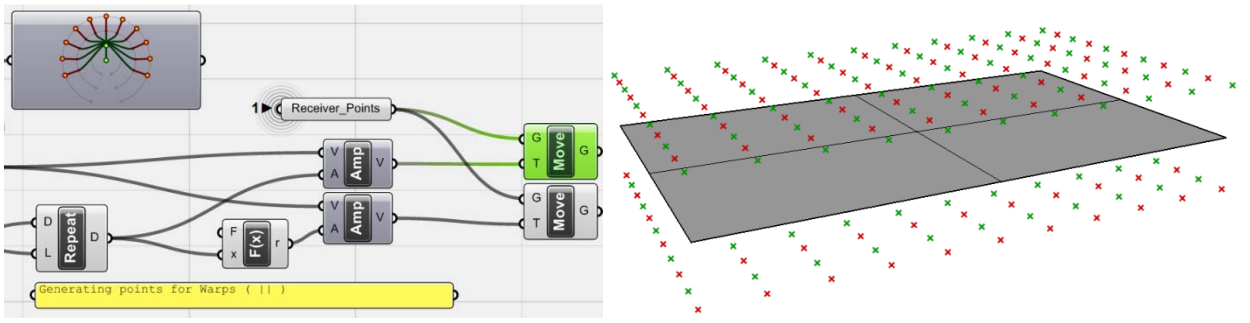

Generating points for Warps (||)

Fig.2.8. All subdivision points (<Receiver> connected to P port of <Subdivide>) are moved with both sets of vectors and one group of displaced points is selected. The most important point is when using a repeated list of both positive/negative heights as vector values, the order of points is one UP, one Down, … .

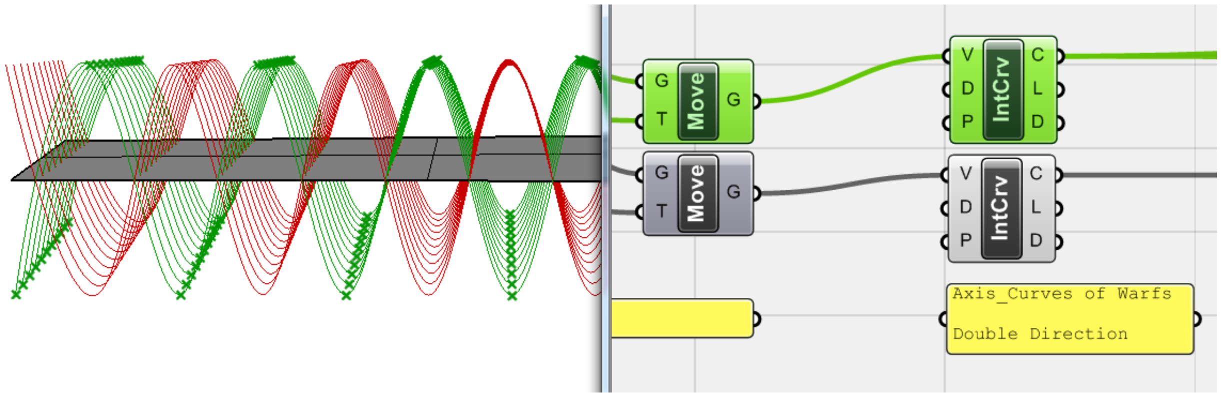


Fig.2.9. As a result, if <move>d points connected to <Interpolate> components, two sets of mirrored curves as axis of threads in the position of warps would generate. Now these lists should be <Cull>ed to omit additional ones.

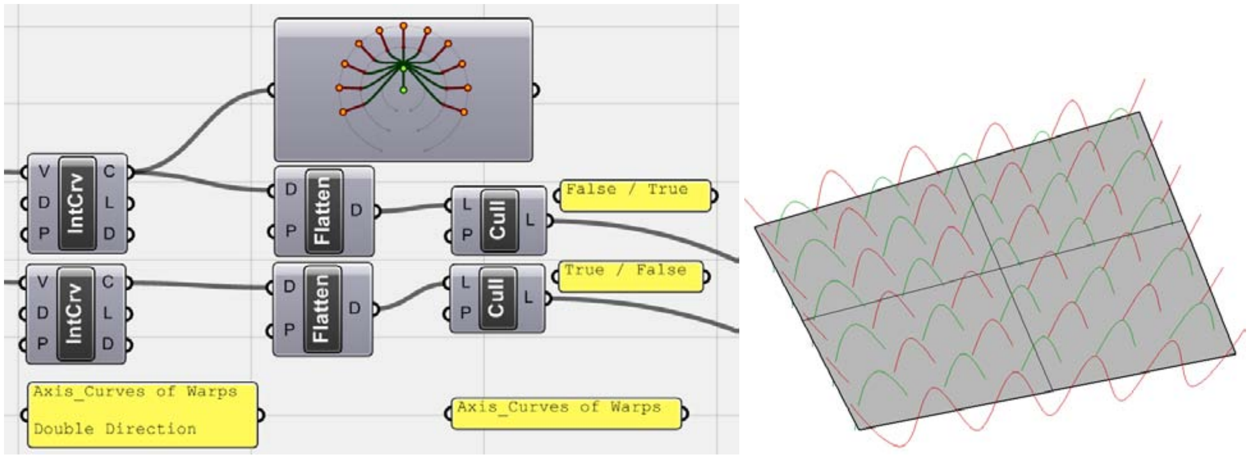
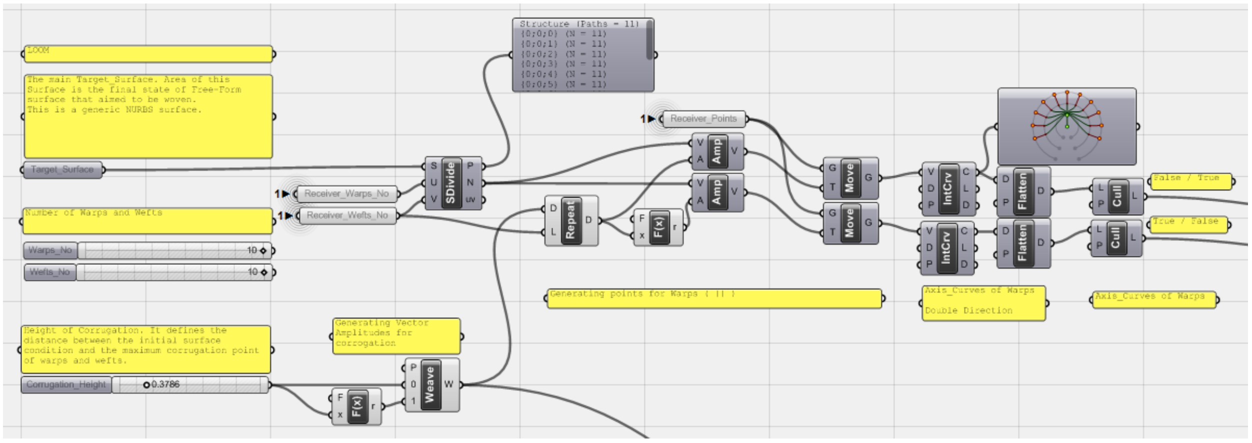


Fig.2.10. As reflected in <Param viewer>, curves are in different data branches. In order to <Cull> them, first <Flatten> components has been used and then by two <Cull Pattern> with inverted Boolean values, all Axis_Curves for warps which are corrugated and shifted half a module in each row are achieved.


LOOM

The main Target_Surface. Area of this Surface is the final state of Free-Form surface that aimed to be woven.
This is a generic NURBS surface.

Number of Warps and Wefts

Height of Corrugation. It defines the distance between the initial surface condition and the maximum corrugation point of warps and wefts.

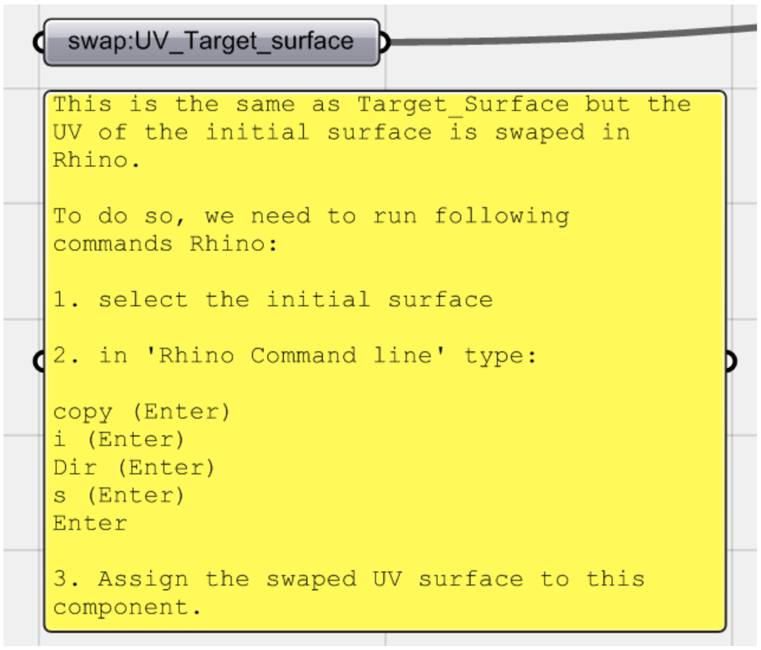
Generating Vector Amplitudes for corrogation

Generating points for Warps (||)

```
Structure (Paths = 11)
{0;0;0} (N = 11)
{0;0;1} (N = 11)
{0;0;2} (N = 11)
{0;0;3} (N = 11)
{0;0;4} (N = 11)
{0;0;5} (N = 11)
```

Fig.2.11. The overall view of the Loom algorithm for the warp generation part.

_Weft Generation



```
This is the same as Target_Surface but the
UV of the initial surface is swaped in
Rhino.

To do so, we need to run following
commands Rhino:

1. select the initial surface

2. in 'Rhino Command line' type:

copy (Enter)
i (Enter)
Dir (Enter)
s (Enter)
Enter

3. Assign the swaped UV surface to this
component.
```

Fig 2.12. For the next part of the algorithm different methods and techniques has been examined to generate wefts. As <Subdivide> component, arranges data in branches based on U division, data needed to rearrange in order to branch by V direction. Here, to solve the problem in an easy way, the <Target_Surface> copied in its place and its U and V direction swapped in Rhino. So by using the same steps as for warps, wefts can be generated.

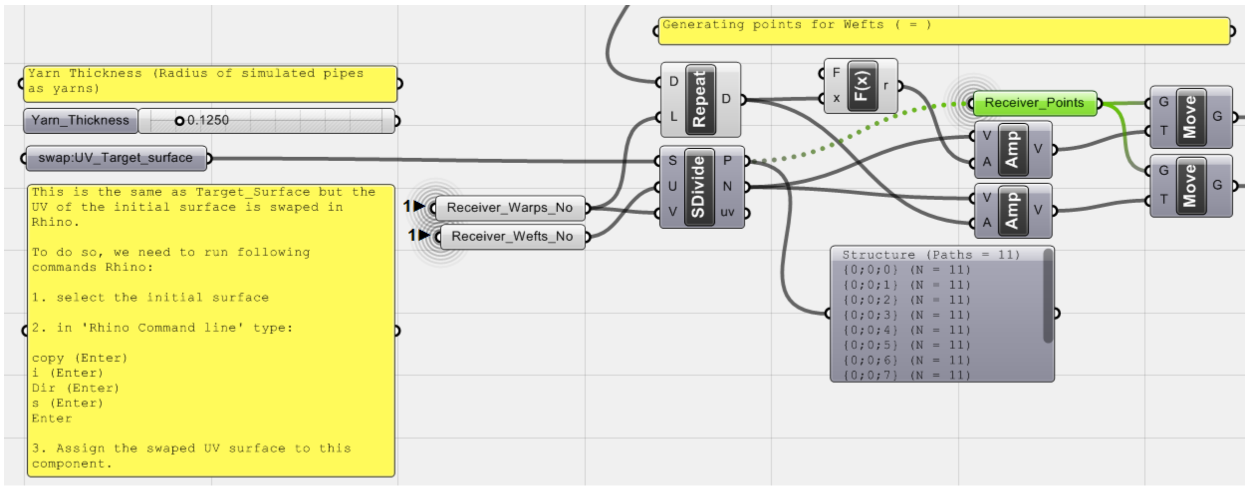
In order to do this, following steps in Rhino should accomplish:

1. *Select the Target Surface*
2. *Type following commands in Rhino command line:*

   Copy (Enter)
   i (Enter)
   Dir (Enter)
   S (Enter)
   (Enter)

   (These commands copy the surface into its original place (i), then by swapping UV direction (s) of the surface a surface with replaced U and V directions but with same size and position would generate)

3. *Assign this new surface to the second <Surface> geometry component which is renamed to <Swap:UV_Target_surface>.*



Generating points for Wefts (=)

Yarn Thickness (Radius of simulated pipes as yarns)

This is the same as Target_Surface but the UV of the initial surface is swaped in Rhino.

To do so, we need to run following commands Rhino:

1. select the initial surface

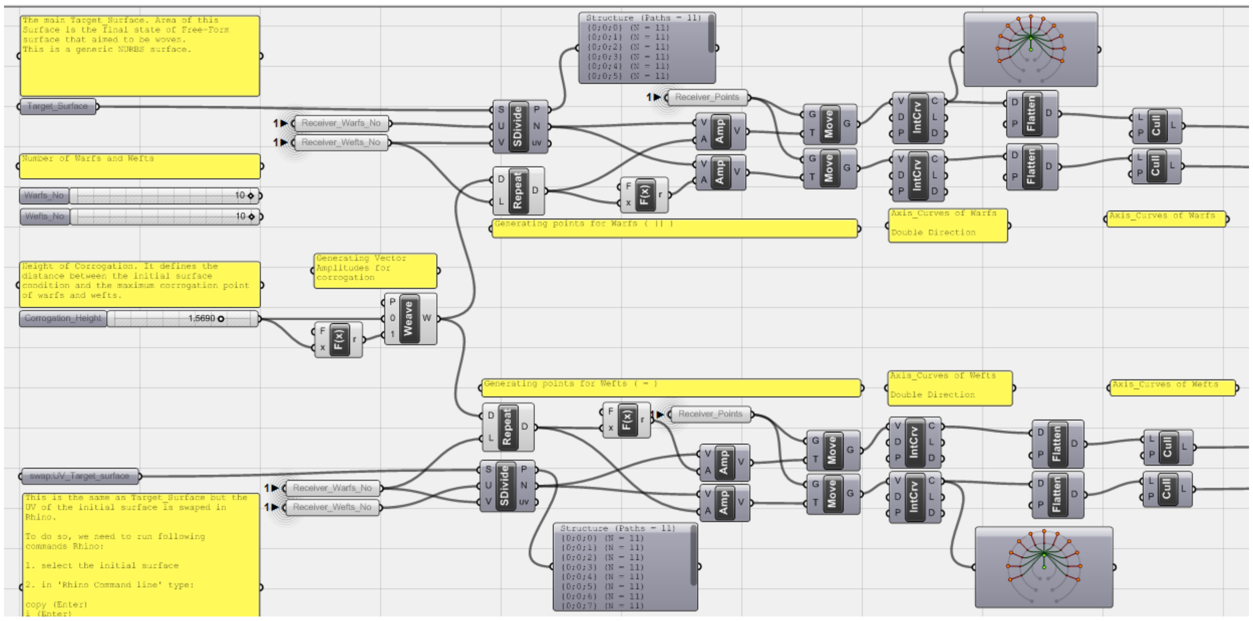
2. in 'Rhino Command line' type:

```
copy (Enter)
i (Enter)
Dir (Enter)
s (Enter)
Enter
```

3. Assign the swaped UV surface to this component.

```
Structure (Paths = 11)
{0;0;0} (N = 11)
{0;0;1} (N = 11)
{0;0;2} (N = 11)
{0;0;3} (N = 11)
{0;0;4} (N = 11)
{0;0;5} (N = 11)
{0;0;6} (N = 11)
{0;0;7} (N = 11)
```

Fig.2.13. Here again this surface subdivided. Notice that the U port, of the <Subdivide> is connected to the number of wefts and the V port, to the number of warps and the data for vector values is <Repeat>ed by the number of warps this time. The rest of the algorithm is the same.



The main Target_Surface. Area of this Surface is the final state of Free-Form surface that aimed to be woven.
This is a generic NURBS surface.

Number of Warfs and Wefts

Height of Corrogation. It defines the distance between the initial surface condition and the maximum corrngation point of warfs and wefts.

Generating Vector Amplitudes for corrogation

Generating points for Warfs (||)

Axis_Curves of Warfs

Double Direction

Axis_Curves of Warfs

Generating points for Wefts (=)

Axis_Curves of Wefts

Double Direction

Axis_Curves of Wefts

This is the same as Target_Surface but the UV of the initial surface is swaped in Rhino.

To do so, we need to run following commands Rhino:

1. select the initial surface

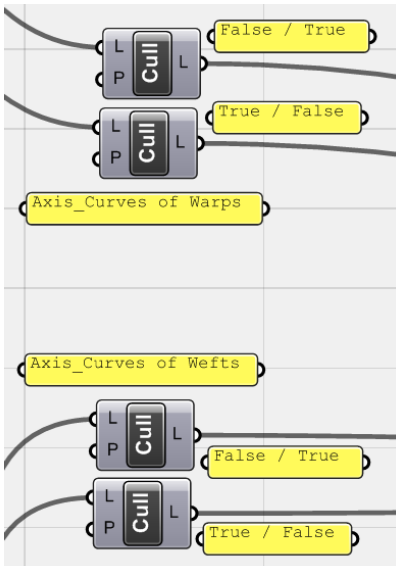
2. in 'Rhino Command line' type:

```
copy (Enter)
i (Enter)
```

```
Structure (Paths = 11)
{0;0;0} (N = 11)
{0;0;1} (N = 11)
{0;0;2} (N = 11)
{0;0;3} (N = 11)
{0;0;4} (N = 11)
{0;0;5} (N = 11)
```

```
Structure (Paths = 11)
{0;0;0} (N = 11)
{0;0;1} (N = 11)
{0;0;2} (N = 11)
{0;0;3} (N = 11)
{0;0;4} (N = 11)
{0;0;5} (N = 11)
{0;0;6} (N = 11)
{0;0;7} (N = 11)
```

Fig.2.14. Both sides of the algorithm look the same for warps and wefts.



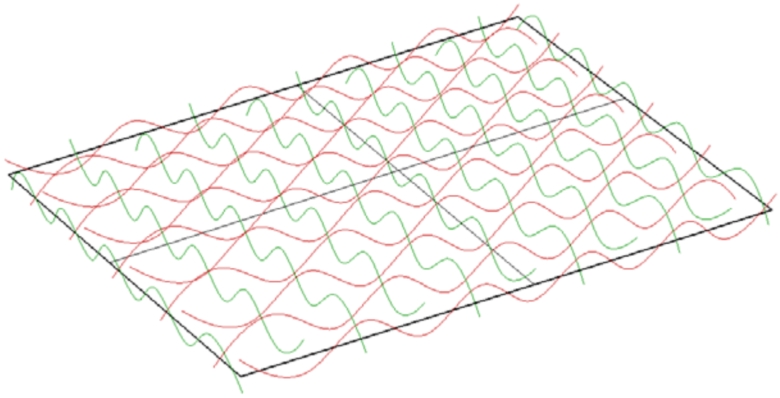


Fig.2.15. Be sure that for <Cull> pattern the above False/True and True/False order should be set. If everything set correctly, then the final axis-curves to generate threads of digital textile are available in a corrugated format.

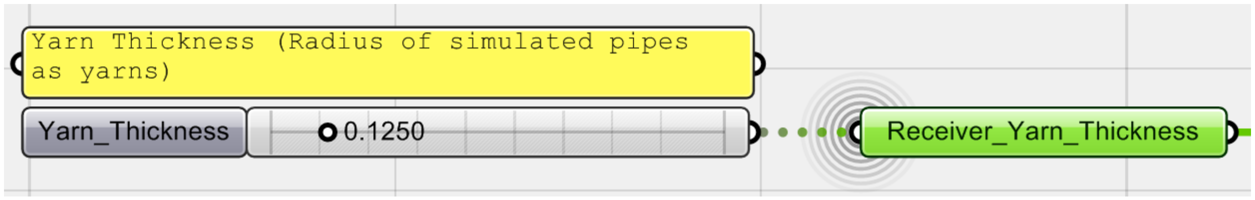
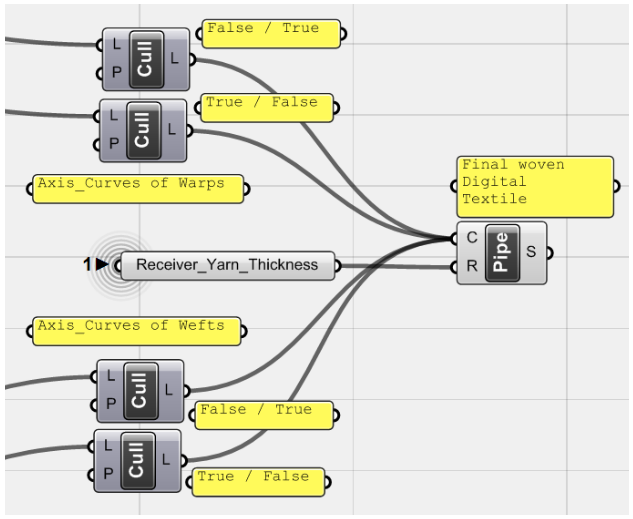


Fig.2.16. The last bit. Threads are aimed to simulate as pipes so their radius or the thickness of threads should be set, here manually controlled by a slider.



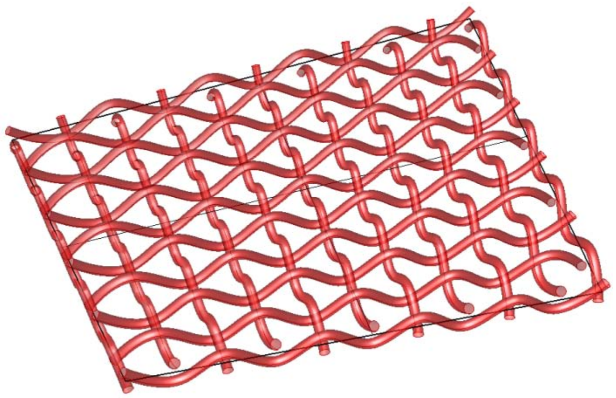
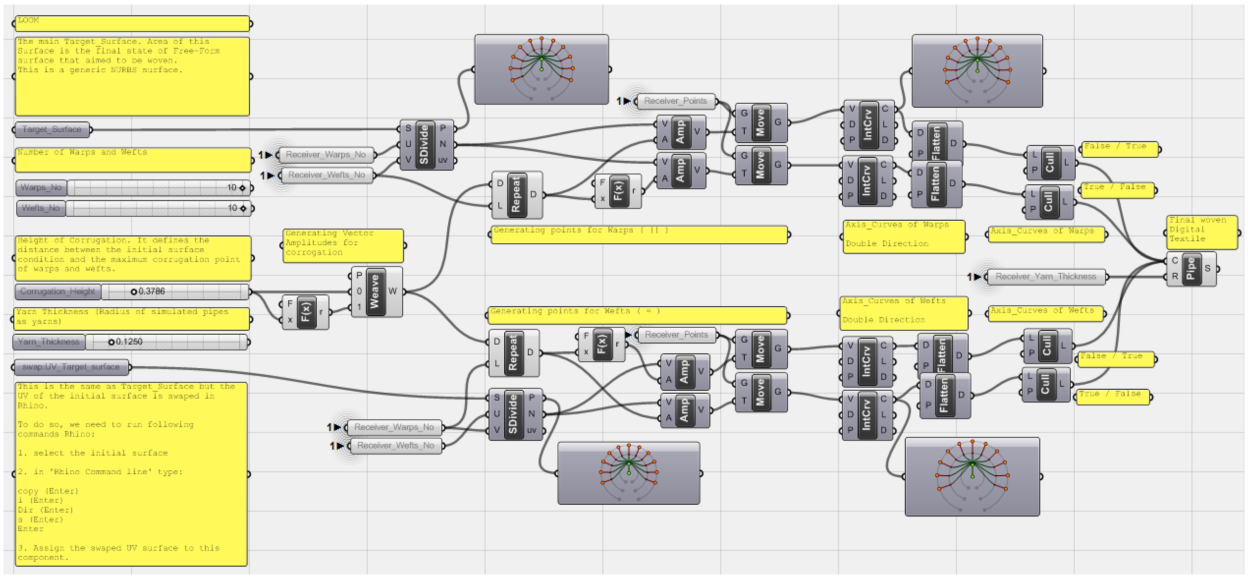


Fig.2.17. connecting all components containing curves to a <Pipe> with a manually controlled radius which is <Receiver_Yarn_Thickness>, now all threads of the digital textile are generated. It is possible to use two <pipe> components and set different thickness values for warps and wefts as well.



The main Target_Surface. Area of this Surface is the final state of Free-Form surface that aimed to be woven.
This is a generic NURBS surface.

Height of Corrugation. It defines the distance between the initial surface condition and the maximum corrugation point of warps and wefts.

Yarn Thickness (Radius of simulated pipes as yarns)

This is the same as Target_Surface but the UV of the initial surface is swaped in Rhino.

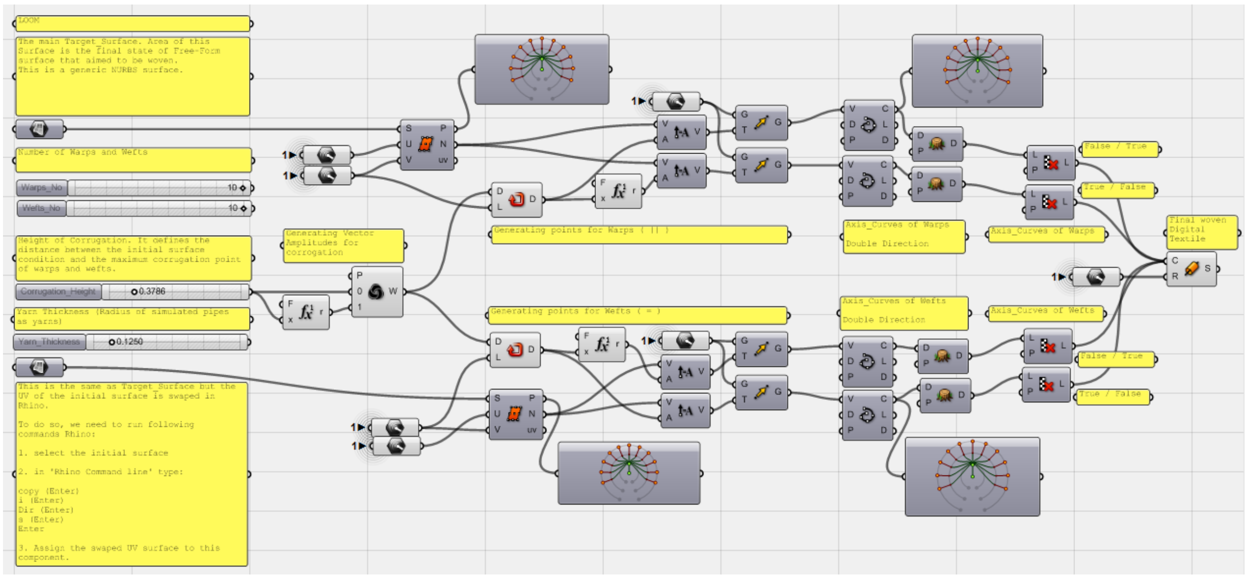
To do so, we need to run following commands Rhino:

1. select the initial surface

2. in 'Rhino Command line' type:

```
copy (Enter)
i (Enter)
Dir (Enter)
s (Enter)
Enter
```

3. Assign the swaped UV surface to this component.


LOOM
Number of Warps and Wefts
Warps_No
10
Wefts_No
10
Corrugation_Height
0.3786
Yarn_Thickness
0.1250
Generating Vector Amplitudes for corrugation
Generating points for Warps (||)
Generating points for Wefts (=)
Axis_Curves of Warps Double Direction
Axis_Curves of Wefts Double Direction
Axis_Curves of Warps
Axis_Curves of Wefts
False / True
True / False
Final woven Digital Textile

The main Target_Surface. Area of this Surface is the final state of Free-Form surface that aimed to be woven.
This is a generic NURBS surface.

Height of Corrugation. It defines the distance between the initial surface condition and the maximum corrugation point of warps and wefts.

Yarn Thickness (Radius of simulated pipes as yarns)

This is the same as Target_Surface but the UV of the initial surface is swaped in Rhino.

To do so, we need to run following commands Rhino:

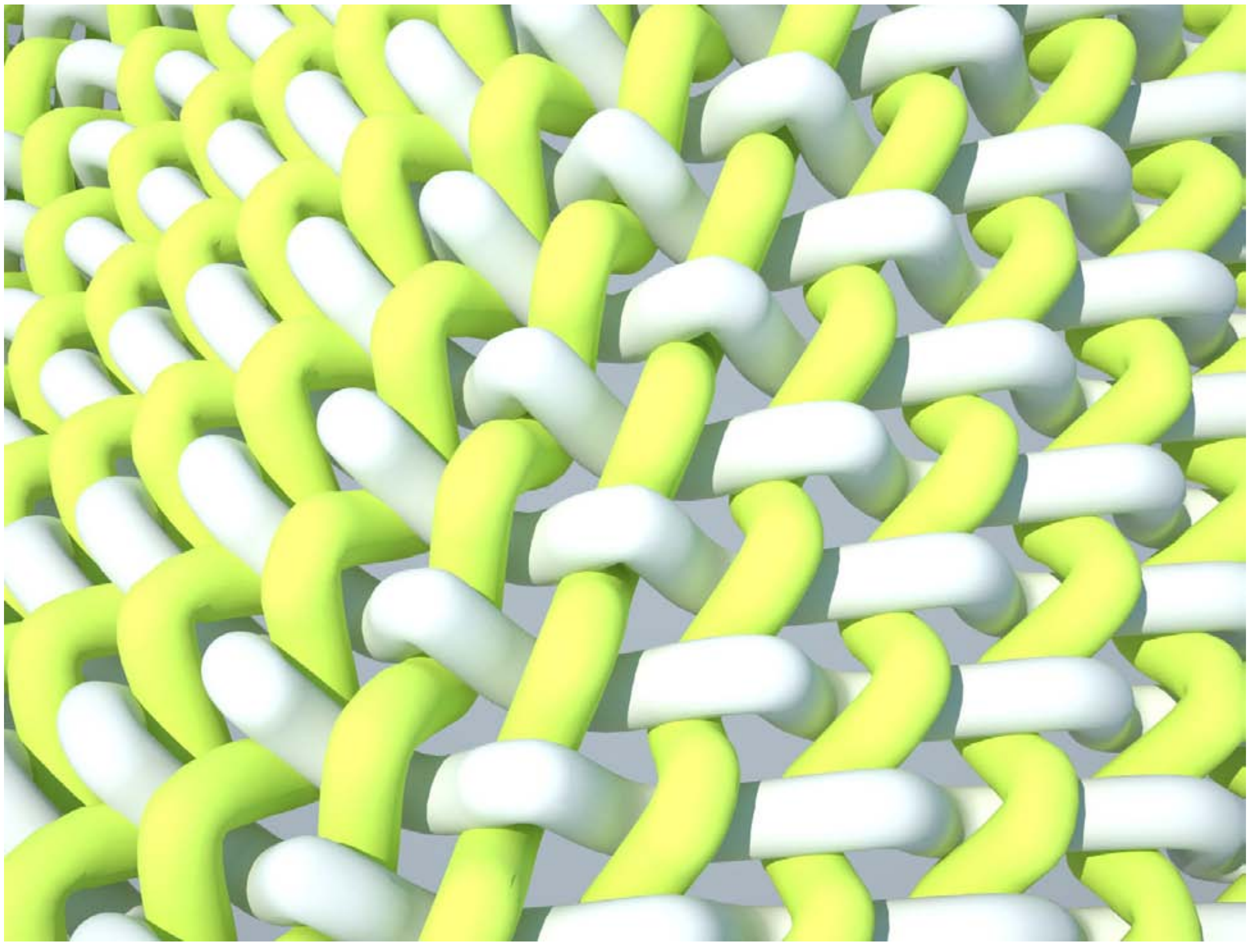
1. select the initial surface

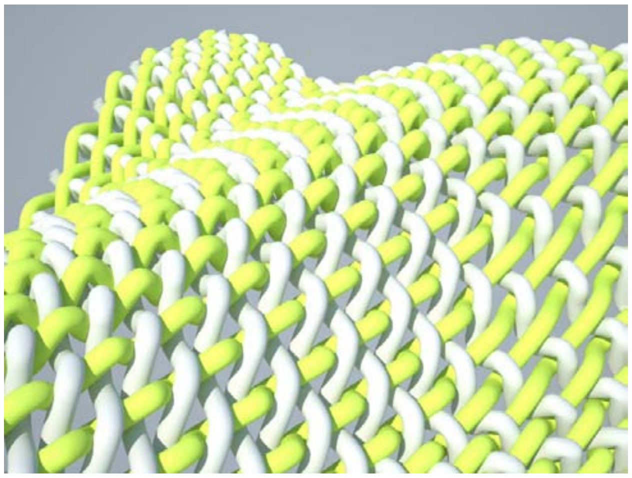
2. in 'Rhino Command line' type:

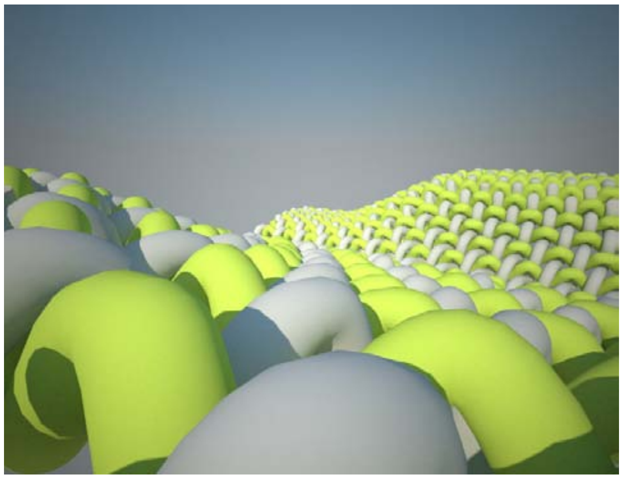
copy (Enter)
i (Enter)
Dir (Enter)
s (Enter)
Enter

3. Assign the swaped UV surface to this component.

Fig.2.18. The Overall look of the Loom Algorithm which weaves a digital textile with a plain weave format.







3_Jacquard Weave

Although the loom algorithm accomplished simple plain-weaving, it was the simplest technique to weave textile. In the history of weaving, people's desire to achieve more colorful, more interesting textiles with different patterns, pushed the industry and inventors to work on more advanced machines to be able to control the pattern of textile, get various products for different purposes.

Development of machineries and tools, several looms created in order to differentiate the pattern and ease the production. First it was the invention of the iron-cast, powered loom that revolutionized the industry[9]. But among different inventions Dobby loom was one of the influential ones. '*Dobby loom is a loom in which each harness can be manipulated individually. This is in contrast to a treadle loom, where the harnesses are attached to a number of different treadles depending on the weave structure. [... Dobby loom is] A type of loom on which small, geometric figures can be woven as in a regular pattern. Originally this type of loom needed a "dobby boy" who sat on the top of the loom and drew up warp threads to term a pattern. Now the weaving is done entirely by machine. This loom differs from a plain loom in that it may have up to thirty-two harnesses and a pattern chain and its expensive weaving.'[10]*

Having control over the warps in different groups (attached to multiple harnesses rather than one), in a dobby loom it is possible to make patterns in textile by weaving wefts through certain groups of warps instead of arbitrary every other one. Observing this increased degree of control over groups of warps to weave textiles, Joseph-Marie Jacquard invented a system to facilitate the ability of the loom for generating patterns even more; he invented **Jacquard Loom**.

3_1_Jacquard Loom

In weaving, device incorporated in special looms to control individual warp yarns. It enabled looms to produce fabrics having intricate woven patterns such as tapestry, brocade, and damask, and it has also been adapted to the production of patterned knitted fabrics. The Jacquard system was developed in 1804–05 by Joseph-Marie Jacquard of France, but it soon spread elsewhere. His system improved on the punched-card technology of Jacques de Vaucanson's loom (1745). Jacquard's loom utilized interchangeable punched cards that controlled the weaving of the cloth so that any desired pattern could be obtained automatically. These punched cards were adopted by the noted English inventor Charles Babbage as an input-output medium for his proposed analytical engine and were used by the American statistician Herman Hollerith to feed data to his census machine. [11]

This advanced system, has control over every single yarn instead of groups of them. So as a result, with the machine that has **single-end warp control**, it became possible to weave numerous patterns in textile. In order to get these patterns in product, they should be designed in a way to transfer to the loom by a medium. The method developed for that was punching the pattern in cards. Based on those yarns that should be raised, they punched some cards as a means of harness controller whose punched data became relevant to the actions that machine should perform (which yarns should be raised and which should remain unraised).

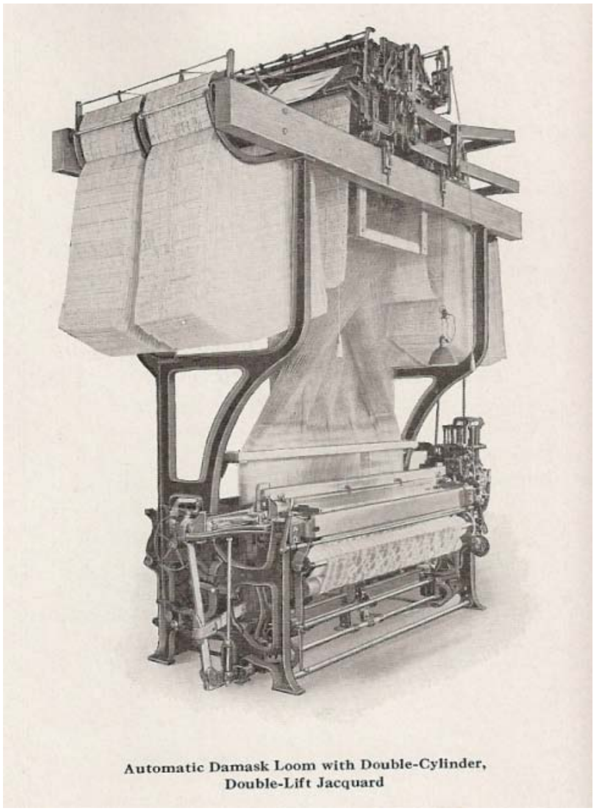

Automatic Damask Loom with Double-Cylinder, Double-Lift Jacquard

Fig.3.1. Jacquard loom

3_2_ Principles of Operation in a Jacquard Loom

Each hole in the card corresponds to a "Bolus" hook, which can either be up or down. The hook raises or lowers the harness, which carries and guides the warp thread so that the weft will either lie above or below it. The sequence of raised and lowered threads is what creates the pattern. Each hook can be connected via the harness to a number of threads, allowing more than one repeat of a pattern. A loom with a 400 hook head might have four threads connected to each hook, resulting in a fabric that is 1600 warp ends wide with four repeats of the weave going across. [12]

3_3_Jacquard Loom Project

While simple looms operate on an arbitrary fashion and weave a simple pattern, as experienced earlier, Jacquard looms operate on a single-end warp control method which needs a designed pattern to be transferred into machine. Referring to the basic understanding of simple algorithmic loom machine and its geometrical necessities, an algorithm for the Jacquard system needs following steps to operate on digital space and weave desired digital textile:

3_3_1_First of all a Pattern needed for weaving. Pattern should be transferred into numerical data, readable by the algorithm, and this numeric data should be retrieved to be used in geometry.

3_3_2_A generic form (like a NURBS surface as previous simple loom) should be considered as the final state of the resultant geometry. This could be Target_Surface.

3_3_3_A system should be developed in order to plot the pattern onto the Target_Surface (considering scale, repetition, ...)

3_3_4_Basic geometries for yarns of the textile should be generated based on the plotted pattern.
3_3_5_Finally the geometry of digital textile should be created.

3_3_1_Pattern of Weaving

As discussed, in earlier Jacquard looms, pattern provided by punch cards, associated with harnesses that controlled yarns to raise or remain unraised. There was a special process to make cards based on desired patterns.

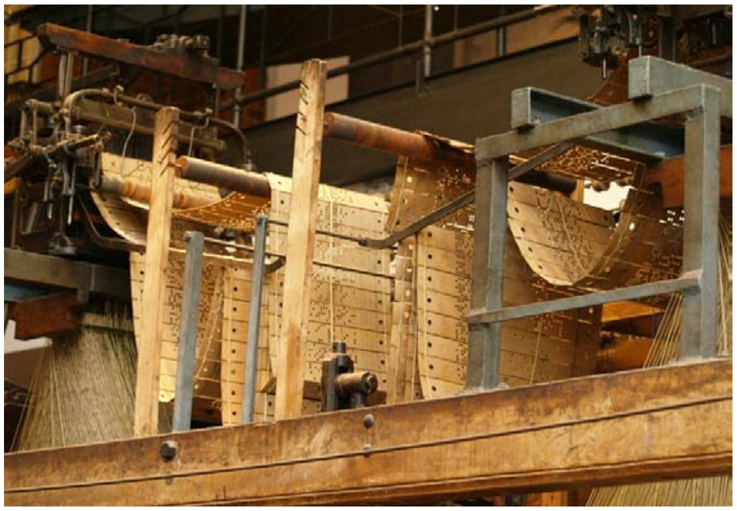
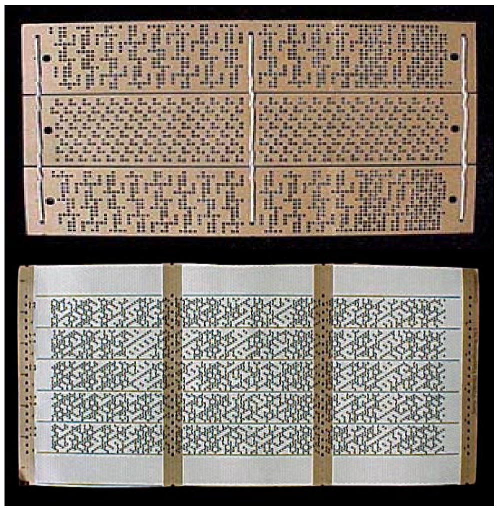



Fig.3.2. Punch cards on a jacquard loom and examples of pattern punch cards. [13]

The age of punch cards is over now. There are multiple ways to design a pattern for weaving textile in digital formats. It could be designed manually on paper as a drawing and then transferred into digital data or it could be generated digitally, with software packages from scratch. Moving into Grasshopper algorithm, data, to represent pattern, can be provided in multiple formats as well.

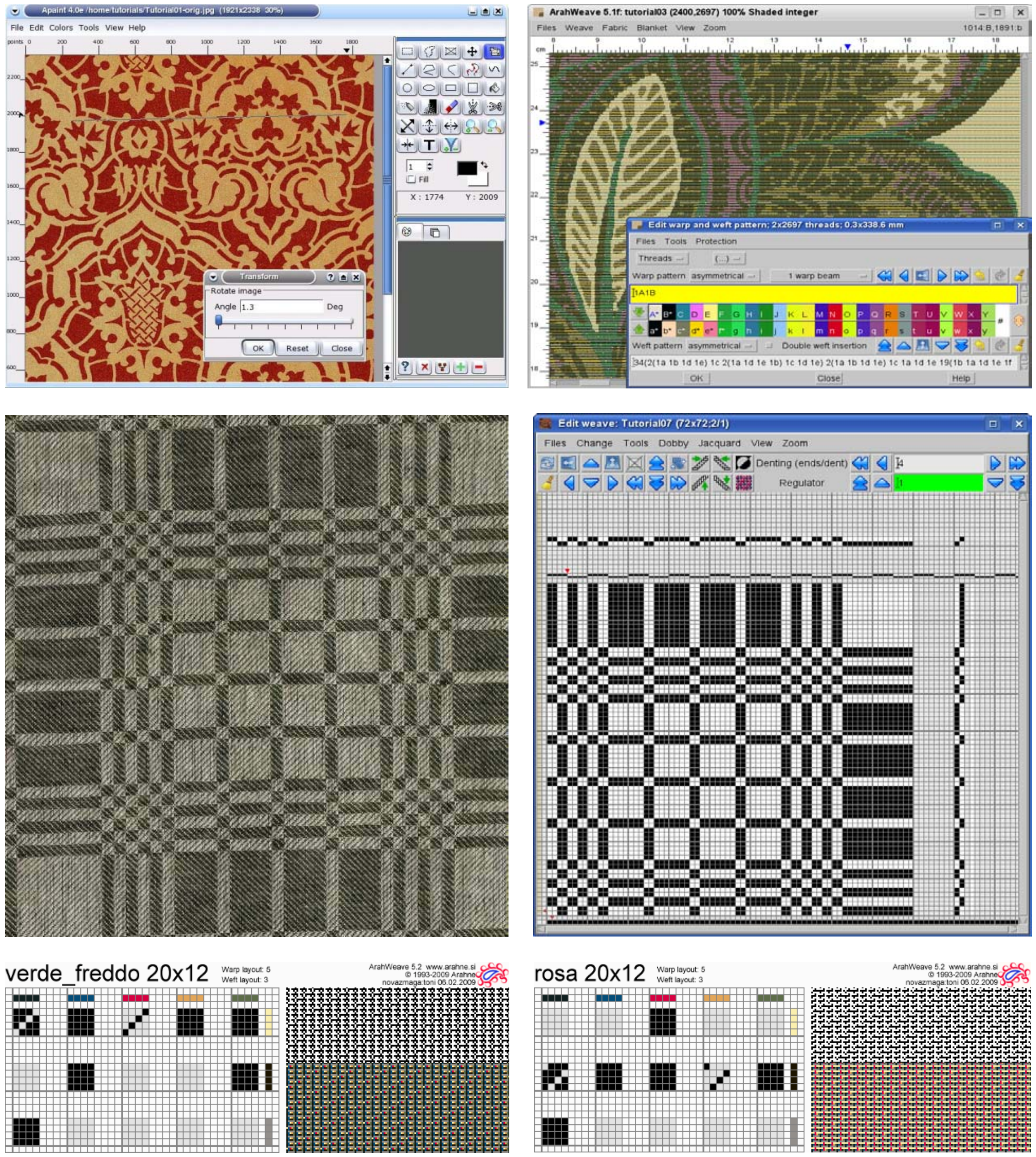

Apaint 4.0e /home/tutorials/Tutorial01-orig.jpg (1921x2338 30%)
File Edit Colors Tools View Help
Transform
Rotate image
Angle 1.3 Deg
OK Reset Close
ArahWeave 5.1f: tutorial03 (2400,2697) 100% Shaded integer
Files Weave Fabric Blanket View Zoom
Edit warp and weft pattern; 2x2697 threads; 0.3x338.6 mm
Files Tools Protection
Warp pattern asymmetrical
Weft pattern asymmetrical
Double weft insertion
OK Close Help
Edit weave: Tutorial07 (72x72;2/1)
Files Change Tools Dobby Jacquard View Zoom
Denting (ends/dent)
Regulator
verde_freddo 20x12
Warp layout: 5
Weft layout: 3
ArahWeave 5.2 www.arahne.si
© 1993-2009 Arahne
rosa 20x12
Warp layout: 5
Weft layout: 3
ArahWeave 5.2 www.arahne.si
© 1993-2009 Arahne

Fig.3.3. Software Packages for designing patterns for fabrics.

Although using softwares to design a pattern is functional and interesting, these softwares are designed for special loom machines and their output data managed to fit them. In a Grasshopper algorithm, possible methods to generate a pattern can be as follows:

1- Pattern Data as a stream of Booleans (True/False). Boolean values could associate with the warps, tell them which one should be raised, which one not. (i.e. **T, F, T, F, F, T, T, T, F, T, ...**)

2- Pattern Data as Tables (i.e. Excel / Spread sheets). Values on the table (like 0/1) can be associated with warps to go up and down.

0	**0**	**1**	**0**
1	**0**	**0**	**0**
1	**1**	**0**	**1**
0	**1**	**1**	**0**

3- Pattern Data as Graphic reference (i.e. Bitmap images). These images can be used from external resources like visual output of software packages or huge amount of graphical references of patterns available in books or on-line.

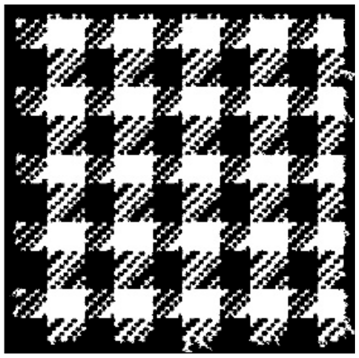


Fig 3.4. An Example of graphical patterns for weaving purposes.

4- Pattern Data can be set internally via available Grasshopper components which is more abstract and probably numerical (includes series of numbers, random values, domains, ...).

_Discussion and Selection of Method

Although it is possible to provide Pattern Data by codes from Spreadsheets or Boolean streams or Numerical Data, but these sources of data are abstract and need data-preparation process which is tough and sometimes non-sense. Using software packages is also not viable option for all users. These softwares are not free and easy to use and their output is suitable for machines, not for any geometrical algorithmic use. Looking at possible applications in Grasshopper and available resources in industry, it seems that using graphical references is a more viable method for algorithmic purposes.

Graphical resources are visual representations of desired patterns which make them more tangible. It is easy to understand the pattern from graphics rather than code. It is even easier than code to provide such data, using graphical softwares. Although it might not considered, it is really important that there are lots of drawn patterns available in resources from printed ones up to on-line that enrich the catalogue of possible options quite a lot.

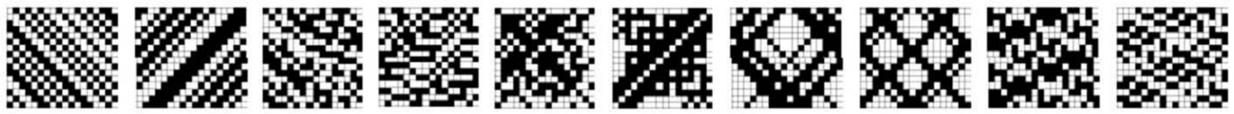


Fig.3.5. Various patterns provided digitally as graphics (Bitmap images).

_Implementation of the selected method into Grasshopper.

Having <Image Sampler> as a component in Grasshopper makes it easy to import images into canvas and start to extract data from them. This useful component is the base material for pattern-data transformation into algorithm in this experiment but certain considerations needed.

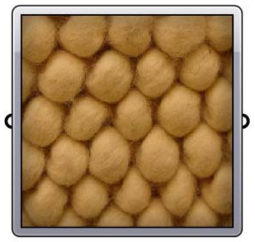


fig.3.6. The <Image Sampler> component could extract color information from an image as numerical data.

The <image sampler> evaluates color data, restored in pixels of image file, based on point coordinates. This means a list of points should be provided to virtually plot onto the image and extract data from those specific positions of the image.

Most of available patterns as reference are grayscale images which represent pattern as a graphical table with black and white cells. Bitmap images, in this sense, are grid based images in which each cell is organized to represent a bit of data (each cell is comprised of multiple pixels of the image). So the basic task in this first part of the algorithm is to extract desired data by providing exact points that associate with each cell positions on the image. The best way to extract precise information from these cells is to plot reference points onto the middle point of cells, otherwise errors would happen when reading data.

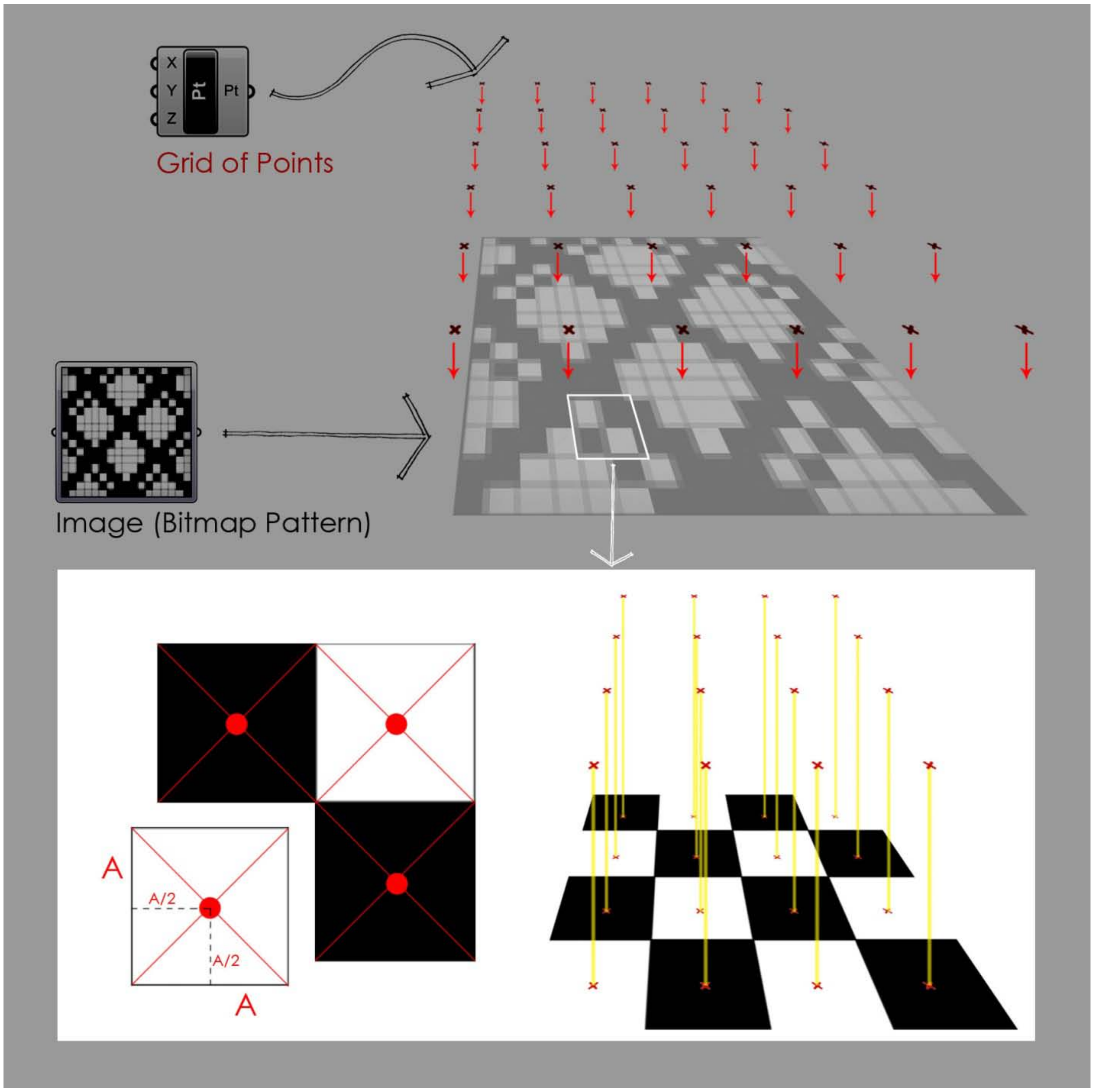


Fig.3.7. When using <Image Sampling> it is important to plot points onto the center points of cells otherwise they would gather wrong data.

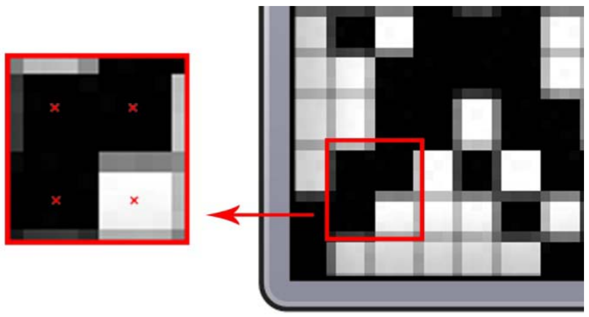


Fig.3.8. When looking closer to the source image, there are pixels with mid-tones which might provide wrong data when the point of evaluation positions out of the center point of the cell.

_Generating Points for <Image Sampling> process.

In order to generate sampling points, a grid of points needed. The number of points in rows and columns should be matched the number of cells inside the image. The easiest way to extract these information from an image file is to find image size and each cell size and then divide image size by cell size to get the number of cells in rows and columns. The image size is easy to read from the file and to find the cell size, it is easy to 'Crop' a cell in a graphic software and get the cell size from that image (both sizes in pixel).

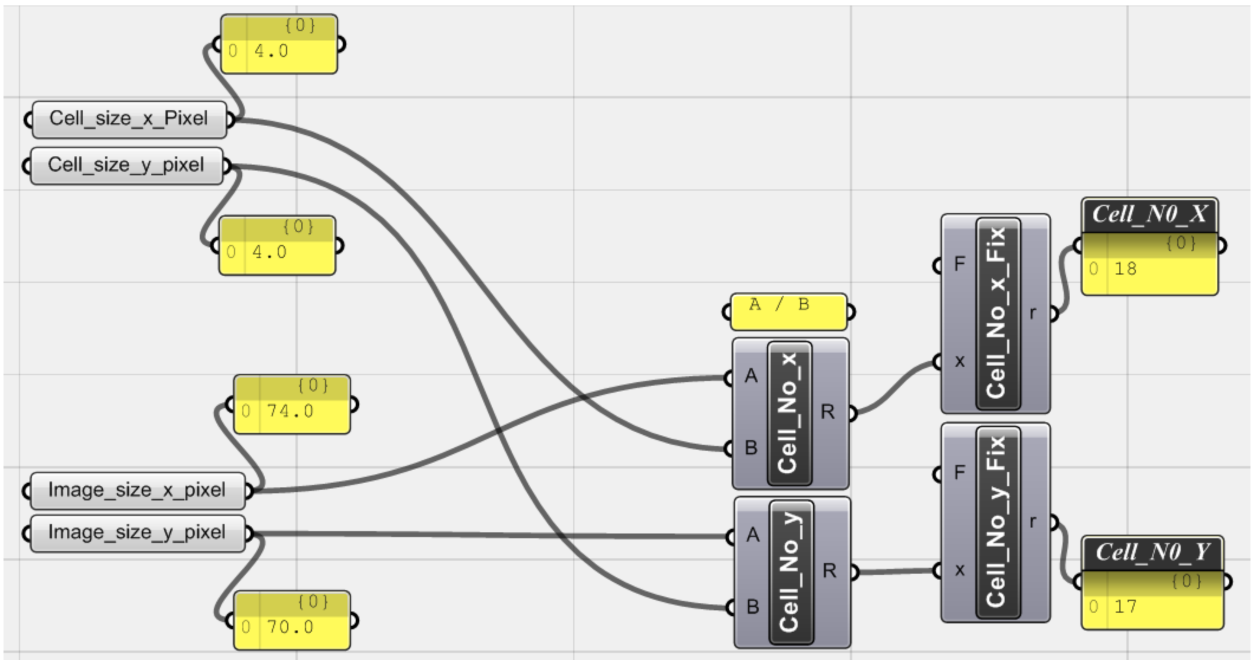


Fig.3.9. Image size and Cell size are set manually (X represents width of the image and Y represents its height). Dividing image size by cell size in x and y direction gives the number of cells in each direction. Using a <Funcion>, F(X)=Fix(x) would result in integers because the number of cells might be a real number with decimals and that happens when the source image has some cells which are not the same as others and this situation mighe accure because source images are generated by different techniques like scanning and these small faults are probable. It is needed to change the cell size to real numbers sometimes, to solve this problem.

Having basic ingredients, points should be generated now. As mentioned before, points should be in the center of cells. This means that if the first point shifts half-size of the cell in X and Y direction it will posit on the center of the first cell. Now if the distance between this first point and other points in the grid set by cell size in both X and Y direction, all points would be in the center points of cells. The number of points in each direction is equal to the number of cells in that direction, calculated in the previous step.

Although it would be easier to use a <Series> component based on the above description, it is not precise enough. Here the best way is to set a domain and divide this domain to get points with more precision.

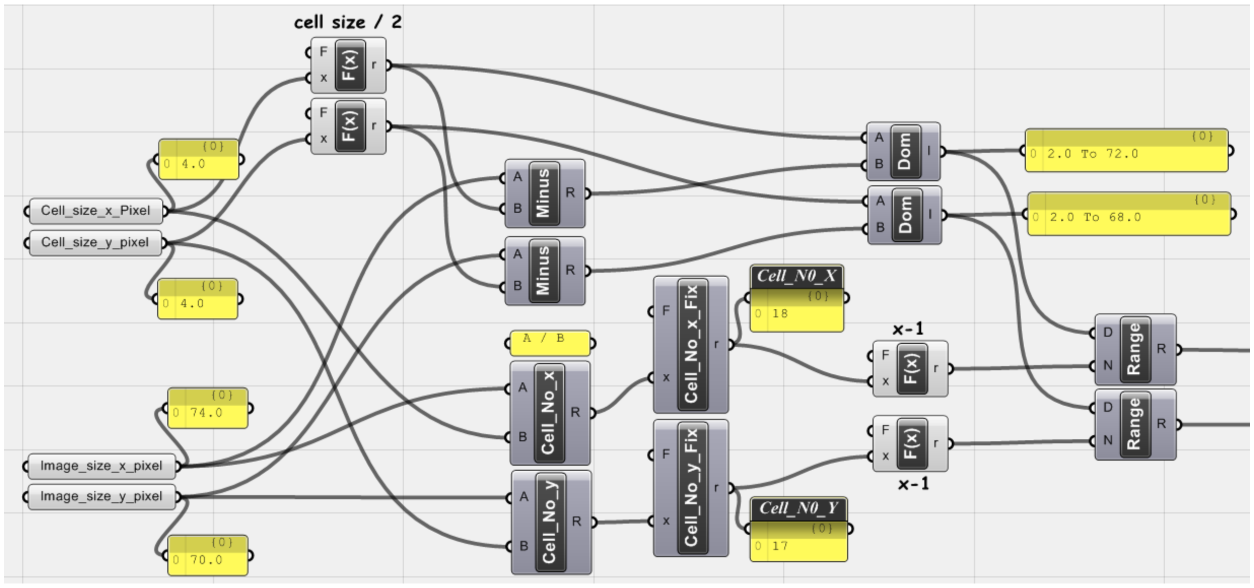
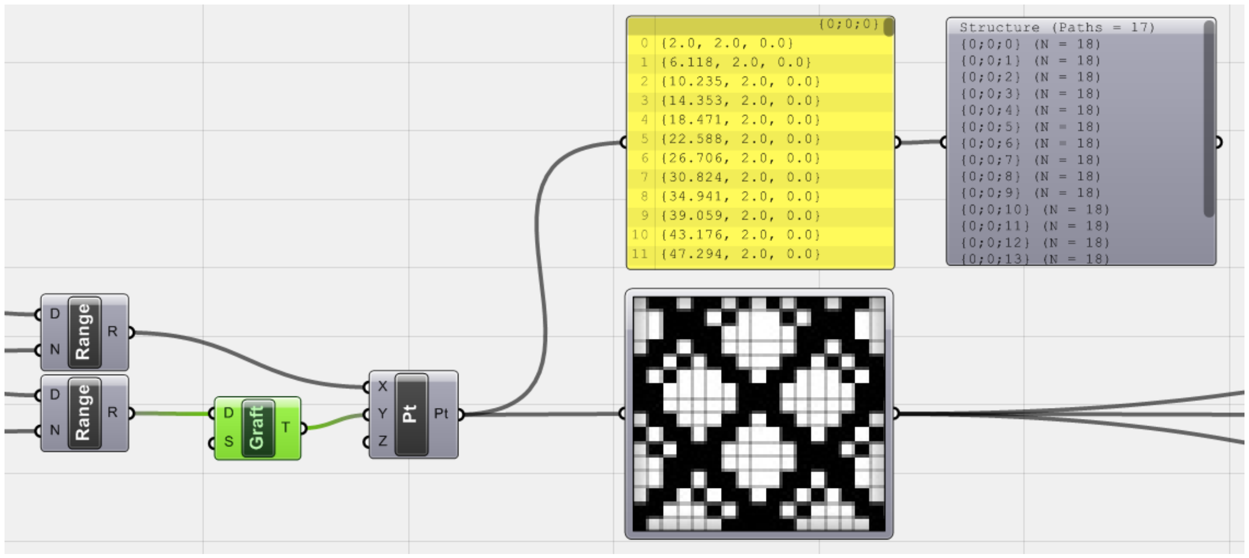


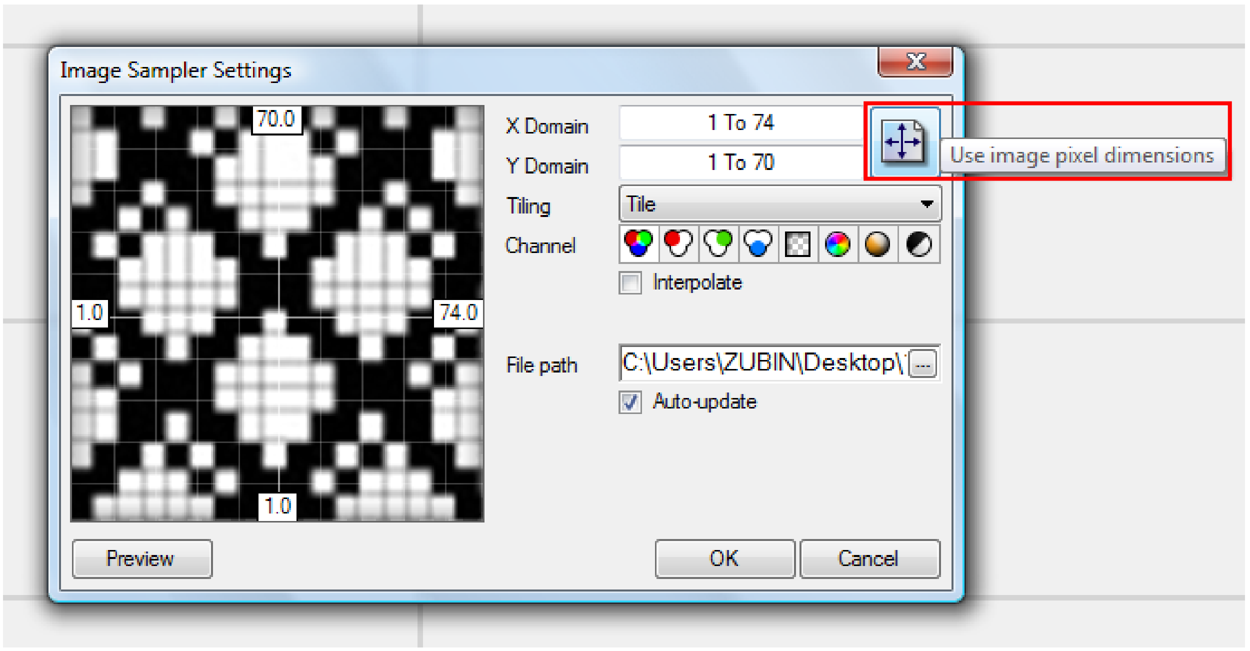
Fig.3.10. Half of the cell size in X and Y direction used as the start value of the numeric domain (which means shifting the first point into the first cell's midpoint). On the other hand, Image size minus cell's half size, both in X and Y direction, is the upper limit of the numeric domain. Dividing these numeric domain by <Range> component will provide numerical values to generate points. The <range> components divided by number of cells (in each direction) minus one (1) since range produces N+1 values when divides a domain into N parts.



```
{0;0;0}
0  {2.0, 2.0, 0.0}
1  {6.118, 2.0, 0.0}
2  {10.235, 2.0, 0.0}
3  {14.353, 2.0, 0.0}
4  {18.471, 2.0, 0.0}
5  {22.588, 2.0, 0.0}
6  {26.706, 2.0, 0.0}
7  {30.824, 2.0, 0.0}
8  {34.941, 2.0, 0.0}
9  {39.059, 2.0, 0.0}
10 {43.176, 2.0, 0.0}
11 {47.294, 2.0, 0.0}
```

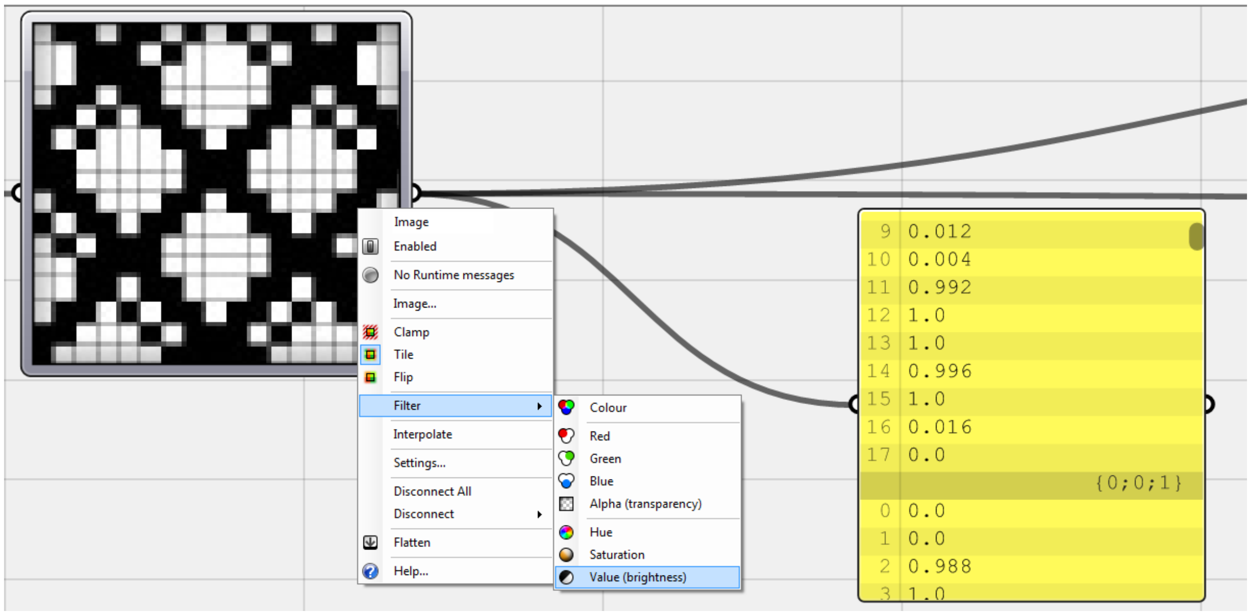
```
Structure (Paths = 17)
{0;0;0} (N = 18)
{0;0;1} (N = 18)
{0;0;2} (N = 18)
{0;0;3} (N = 18)
{0;0;4} (N = 18)
{0;0;5} (N = 18)
{0;0;6} (N = 18)
{0;0;7} (N = 18)
{0;0;8} (N = 18)
{0;0;9} (N = 18)
{0;0;10} (N = 18)
{0;0;11} (N = 18)
{0;0;12} (N = 18)
{0;0;13} (N = 18)
```

Fig.3.11. Both <Range> components are used to generate coordinates for sampling points to feed <Image Sampler> component. Using a <Graft> component for Y values of points, each row of data (points with the same Y value) will sort in one data branch. So in essence, the resultant point grid would have data branches as much as cells in Y direction and in each data branch there are points with the number of cells in X direction. Analyticaly this means that if any row of cells assumed as a representative of a weft, each weft is sorted into one branch of data tree.



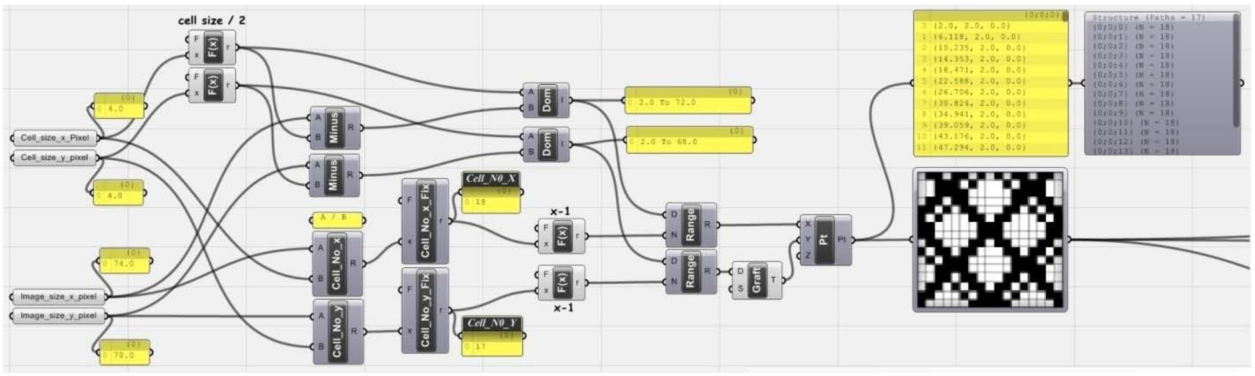
X Domain: 1 To 74
Y Domain: 1 To 70
Tiling: Tile
Channel
Interpolate
File path: C:\Users\ZUBIN\Desktop\
Auto-update

Fig.3.12. Another important bit of the image sampling. <Image Sampler> has a setting window in its context pop-up menu. The X and Y domain of the image should be set to the image size for this experiment, using the button provided here (in most recent versions of Grasshopper). If this domain does not set, points might be out of domain with Null output at the end.



```
 9 0.012
10 0.004
11 0.992
12 1.0
13 1.0
14 0.996
15 1.0
16 0.016
17 0.0
          {0;0;1}
 0 0.0
 1 0.0
 2 0.988
 3 1.0
```

Fig.3.13. Using Grayscale images as the pattern, here the Filter of the <Image Sampler> set to ***'Value'*** *because only Brightness value needed for the rest of the algorithm (in this experiment color is not important).*



```
{0;0;0}
0  {2.0, 2.0, 0.0}
1  {6.118, 2.0, 0.0}
2  {10.235, 2.0, 0.0}
3  {14.353, 2.0, 0.0}
4  {18.471, 2.0, 0.0}
5  {22.588, 2.0, 0.0}
6  {26.706, 2.0, 0.0}
7  {30.824, 2.0, 0.0}
8  {34.941, 2.0, 0.0}
9  {39.059, 2.0, 0.0}
10 {43.176, 2.0, 0.0}
11 {47.294, 2.0, 0.0}
```

```
Structure (Paths = 17)
{0;0;0}  (N = 18)
{0;0;1}  (N = 18)
{0;0;2}  (N = 18)
{0;0;3}  (N = 18)
{0;0;4}  (N = 18)
{0;0;5}  (N = 18)
{0;0;6}  (N = 18)
{0;0;7}  (N = 18)
{0;0;8}  (N = 18)
{0;0;9}  (N = 18)
{0;0;10}  (N = 18)
{0;0;11}  (N = 18)
{0;0;12}  (N = 18)
{0;0;13}  (N = 18)
```

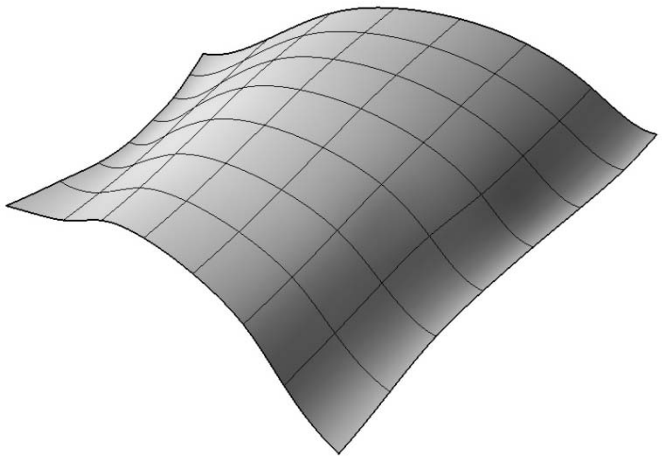
Fig.3.14. First part of the algorithm which extracts data from source image as pattern of weaving.

3_3_2_Target Space/Form of Weaving

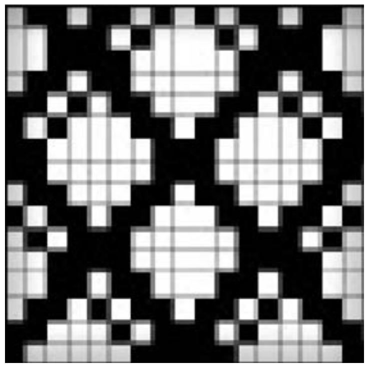
After transformation of pattern of weaving into numerical data, it is possible to set a generic surface as the final state of the weaving algorithm - target surface - and plot the pattern onto it. To have an analytical insight into algorithm for this part, it should be mentioned here that like the first simple loom algorithm, the process has three main steps: first, series of points should be generated in association with target surface as basic generative geometries. The next step is to displace these points above and under the target surface based on the pattern. Finally for each set of points a curve should interpolate. These curves are axis curves for further yarn productions.

To be able to do all these, some considerations needed:

1- The first and most important point is to know or assume the scale of target surface in relation to the scale of pattern. Projection of the pattern onto the target surface is possible with different scales:



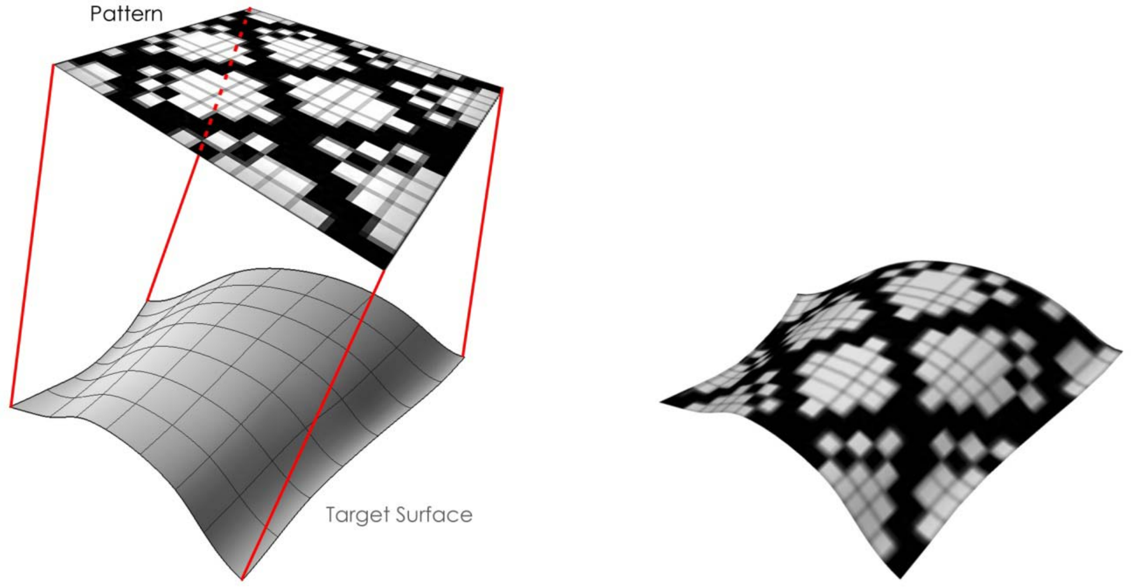
A. Target Surface



B. Pattern

Fig.3.15. A. Target surface as the input of the Algorithm. *B. Pattern for weaving.*

A:



B:

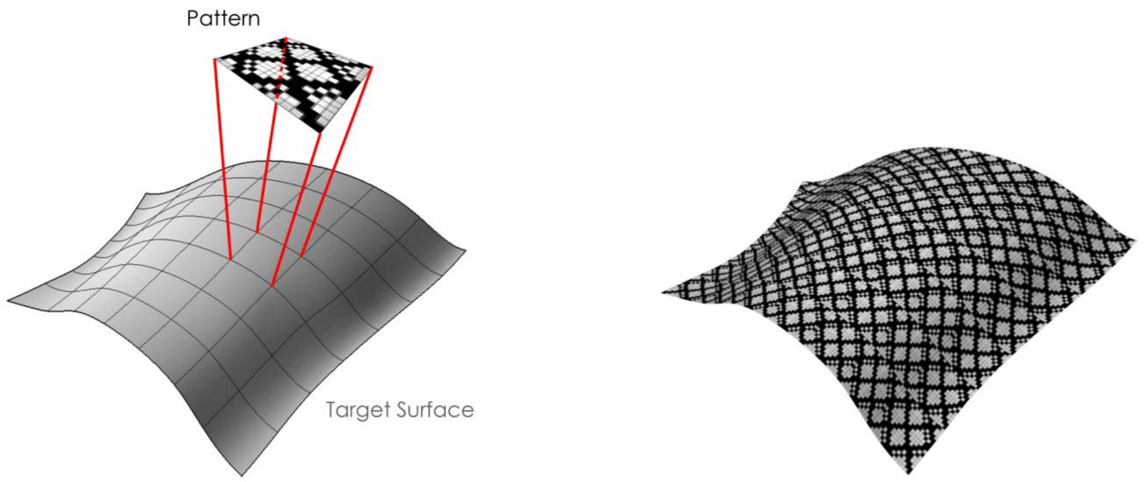


Fig.3.16. The first point to weave target surface based on a predefined pattern is to decide about its size. It should be calculated somehow or the better way is to set some parameters that make it possible to control the scale which would enable the designer to check and change it visually.

As it is clear in above diagrams, it is not always the case to project the pattern onto an entire surface and pre-requisites for repetition of the pattern should be implemented in algorithm.

2- Another important point is to consider that the pattern in this algorithm is not like a mosaic which could be repeated as separate modules, like using Morphing technique. Yarns of the textile should be one-piece geometries and since the method of yarn production in this algorithm is based on the curve extrusion, these curves should be one-piece for each yarn as well. That's why data (points) for each weft or warp should be isolated in one data branch.

3- If the repetition of the pattern includes mirroring the pattern as well, it should be implemented into the algorithm (but ignored in the current experiment).

3_3_3_Plotting Points onto Target Surface

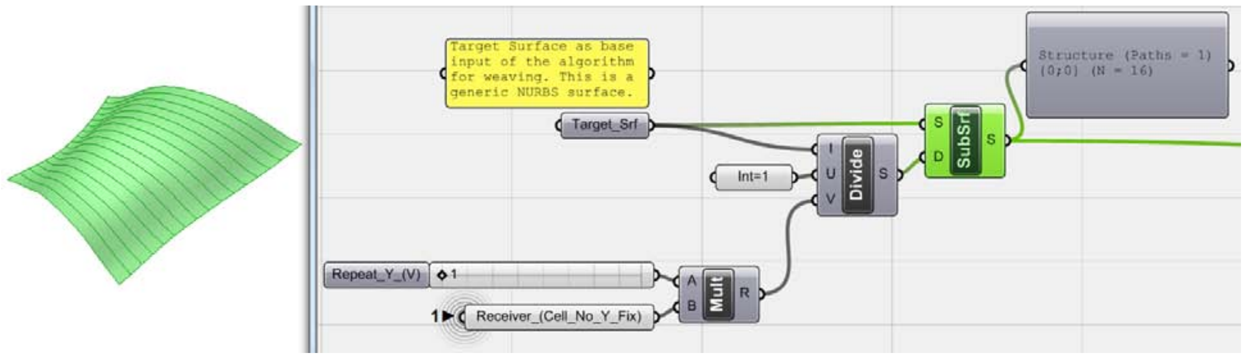

Target Surface as base input of the algorithm for weaving. This is a generic NURBS surface.
Structure (Paths = 1) {0;0} (N = 16)
Target_Srf
Int=1
I
U
V
Divide
S
D
SubSrf
Repeat_Y_(V)
1
A
B
Mult
R
Receiver_(Cell_No_Y_Fix)

Fig.3.17. For the second part of the algorithm, a generic NURBS surface introduced to the canvas as the basic geometry that should be woven as <Target_Srf>. Having tested different techniques for subdividing this <target_srf>, Isotrim -<subsrf>- has been selected instead of divide surface (it will discuss why).

In order to get sub-surfaces, the domain of the surface divided first in its V direction. That's why the U input of the <Divide Domain2> is set to <1>. To divide the V direction here, two factors are important. The <Receiver_(Cell_No_Y_Fix)> component receives data from the < Cell_No_Y_Fix> from previous part. This item is arbitrary for subdividing surface and means if the pattern wanted to repeat only once, the surface should be divided at least with the amount of pattern cells in its Y direction. And if the designer wanted to control the amount of repetition (Scale of pattern on the <Target_Srf>) then it should be multiplied by a given integer from <Slider>.

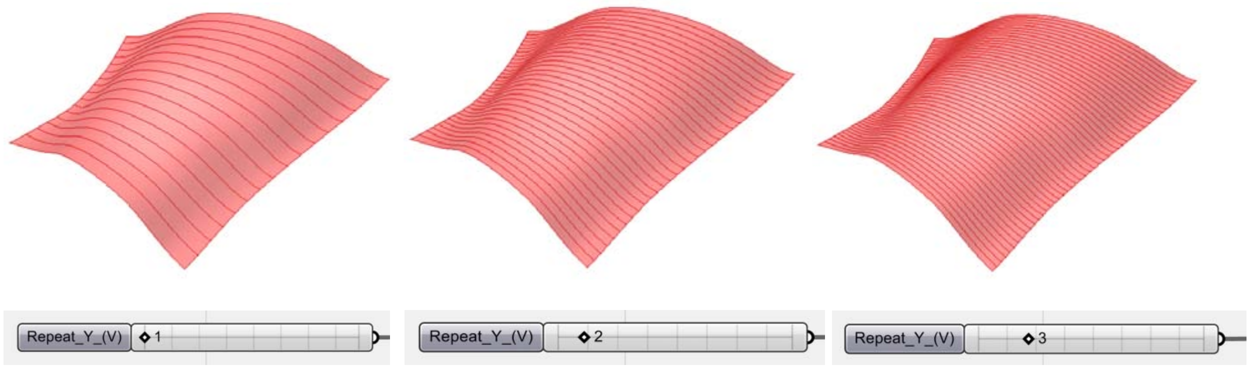


Fig.3.18. Subdividing the <Target_Srf> into sub-surfaces 1.pattern will repeat once, 2. Pattern will repeat twice and 3. Pattern will repeat three times.

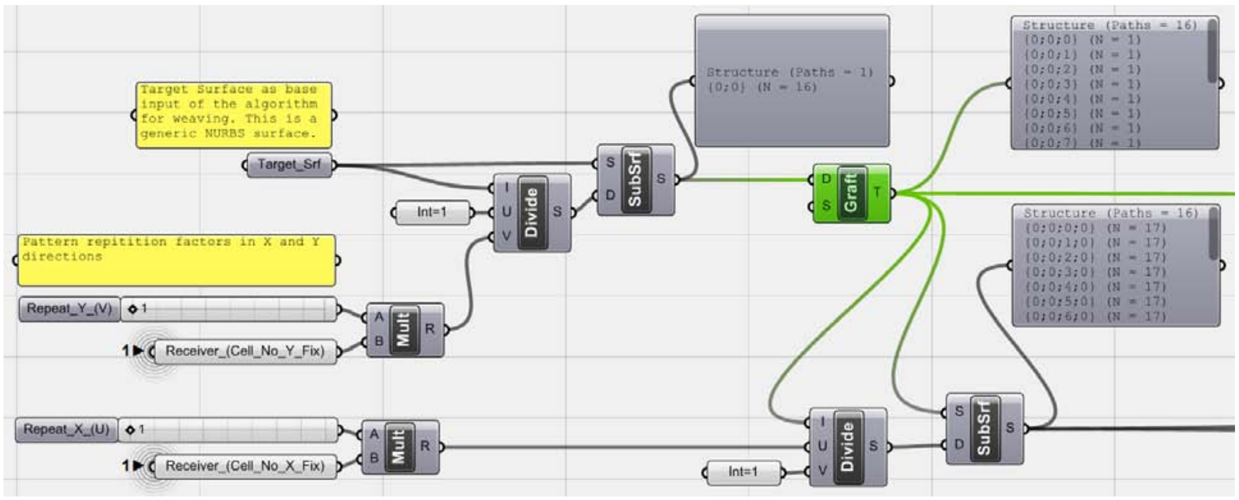


Fig.3.19. In the second step, previously subdivided surfaces are again divided but this time in their U direction. So the same concept has been used but this time their domain divided -using <Divide Domain2>- by V=1 and U=<cell_No_X_Fix> * <Slider> which defines how many time pattern should repeat in U direction. Looking at <Graft> component here shows why this process of sub-dividing happened in two steps. The second <SubSrf> component and its resultant <Param Viewer> proves that data has been branched as much as division factor for V direction, and each data branch, contains as much values as division factor in U direction.

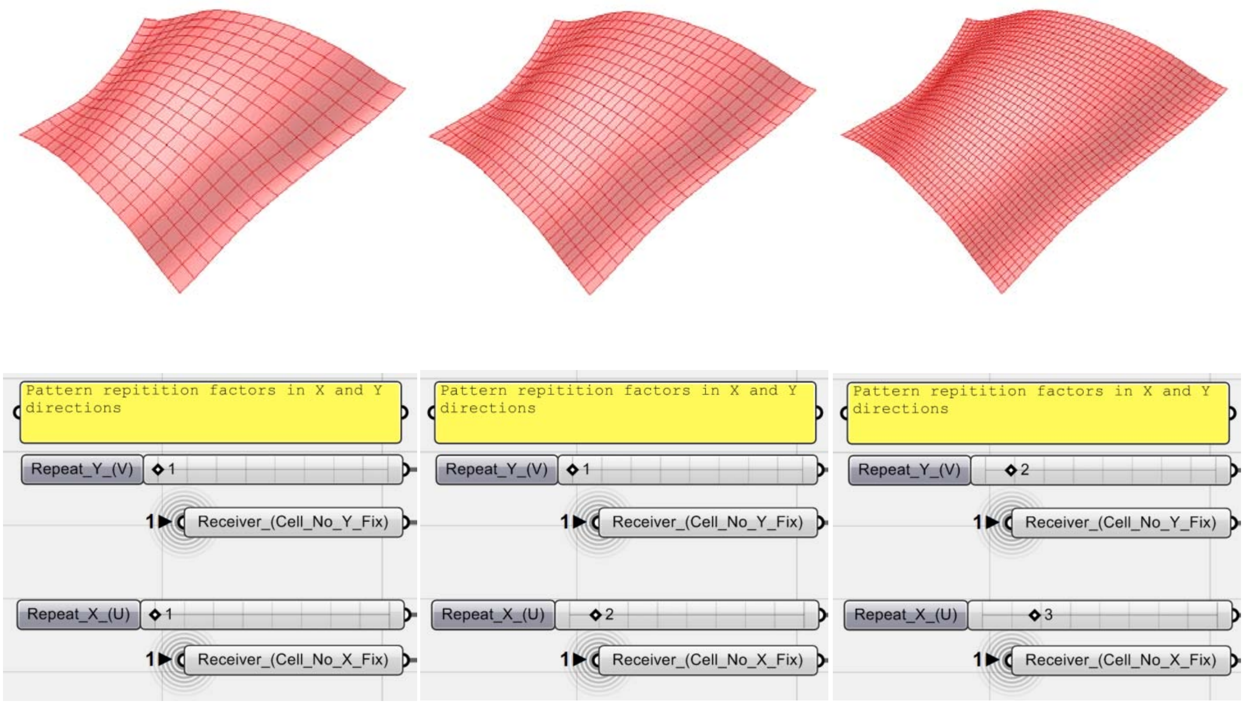
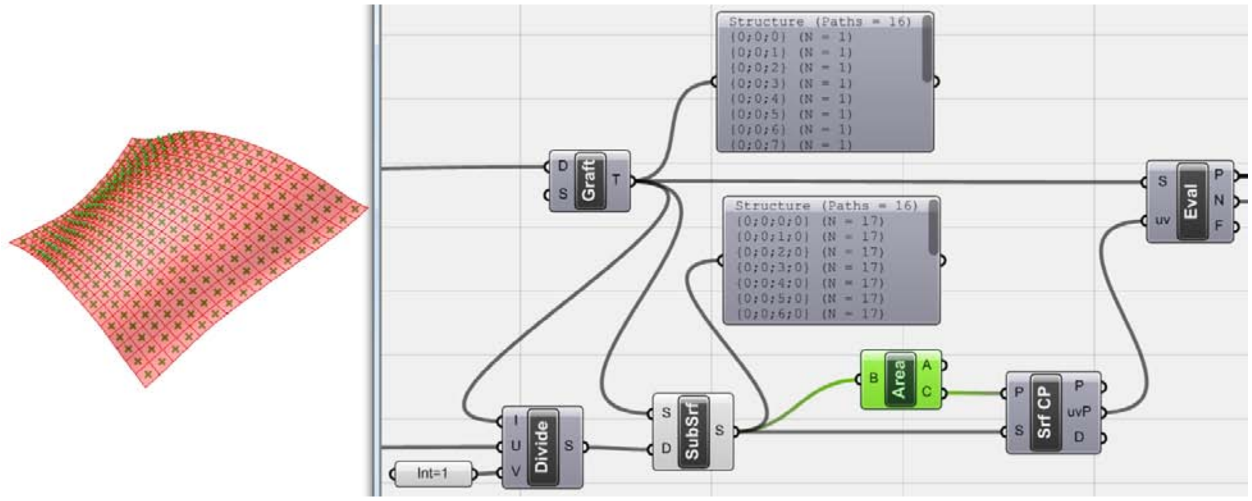


Fig.3.20. Surface subdivision in both X and Y (U and V) directions.

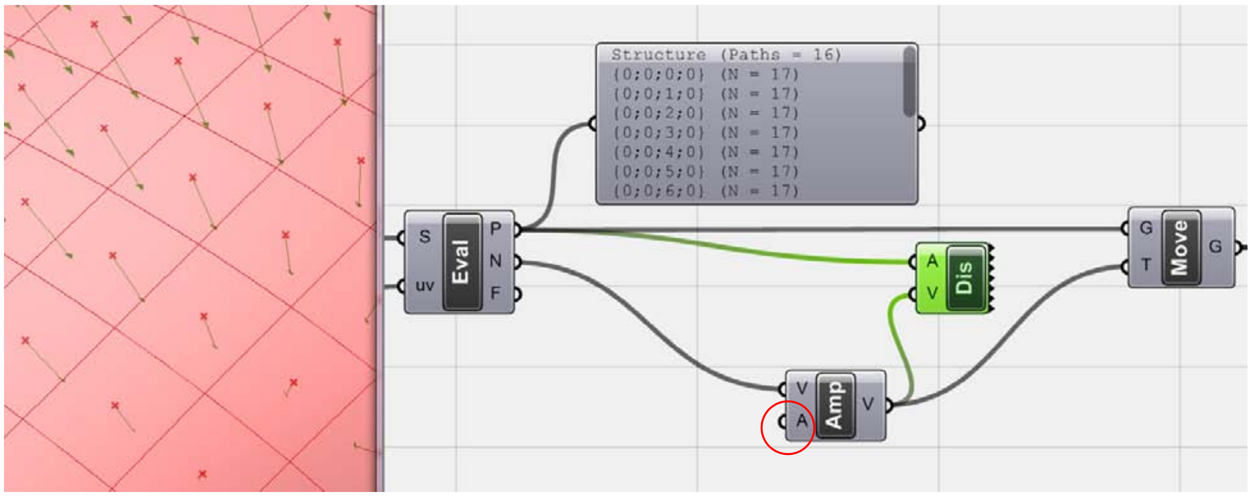
Now all subdivided surfaces are available in both U and V (X and Y) directions, it is time to extract points from them. Considering each subdivided surface as a cell which can associate with the pattern cells, the best point to extract from these surfaces is their mid-point.



```
Structure (Paths = 16)
{0;0;0} (N = 1)
{0;0;1} (N = 1)
{0;0;2} (N = 1)
{0;0;3} (N = 1)
{0;0;4} (N = 1)
{0;0;5} (N = 1)
{0;0;6} (N = 1)
{0;0;7} (N = 1)
```

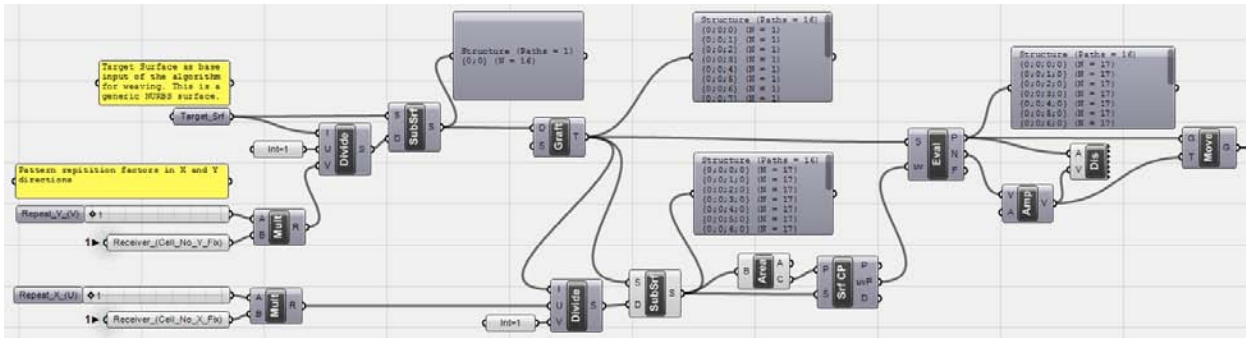
```
Structure (Paths = 16)
{0;0;0;0} (N = 17)
{0;0;1;0} (N = 17)
{0;0;2;0} (N = 17)
{0;0;3;0} (N = 17)
{0;0;4;0} (N = 17)
{0;0;5;0} (N = 17)
{0;0;6;0} (N = 17)
```

Fig.3.21. The midpoint of sub-surfaces are calculated by an <Area> component which gives the area-centroid of surfaces as well. These points then <Evaluate>d again on sub-surfaces using <Surface CP> and <Evaluate Surface> in order to have access to both 'Points' and also 'Normals' of surfaces at the same time.



```
Structure (Paths = 16)
{0;0;0;0} (N = 17)
{0;0;1;0} (N = 17)
{0;0;2;0} (N = 17)
{0;0;3;0} (N = 17)
{0;0;4;0} (N = 17)
{0;0;5;0} (N = 17)
{0;0;6;0} (N = 17)
```

Fig.3.22. At the end of this phase, all points should be <Move>d, using 'Normal' vectors of each point. But the direction of movement and the amount of movement has to be set, using data collected in previous step from pattern image. As it is shown by a vector <Display> component, all 'Normal' vectors now have same direction which does not create any corrugation. Here using a vector <Amplitude> would define vector length and its direction (positive or negative) for this displacement. Values for amplitude are what pattern should pass to the rest of the algorithm and waiting to be set now.


Target Surface as base input of the algorithm for weaving. This is a generic NURBS surface.

Pattern repitition factors in X and Y directions

```
Structure (Paths = 1)
{0;0} (N = 14)
```

```
Structure (Paths = 16)
{0;0;0} (N = 1)
{0;0;1} (N = 1)
{0;0;2} (N = 1)
{0;0;3} (N = 1)
{0;0;4} (N = 1)
{0;0;5} (N = 1)
{0;0;6} (N = 1)
{0;0;7} (N = 1)
```

```
Structure (Paths = 16)
{0;0;0;0} (N = 17)
{0;0;1;0} (N = 17)
{0;0;2;0} (N = 17)
{0;0;3;0} (N = 17)
{0;0;4;0} (N = 17)
{0;0;5;0} (N = 17)
{0;0;6;0} (N = 17)
```

```
Structure (Paths = 16)
{0;0;0;0} (N = 17)
{0;0;1;0} (N = 17)
{0;0;2;0} (N = 17)
{0;0;3;0} (N = 17)
{0;0;4;0} (N = 17)
{0;0;5;0} (N = 17)
{0;0;6;0} (N = 17)
```

Fig.3.23. Surface Subdivision phase of the algorithm so far.

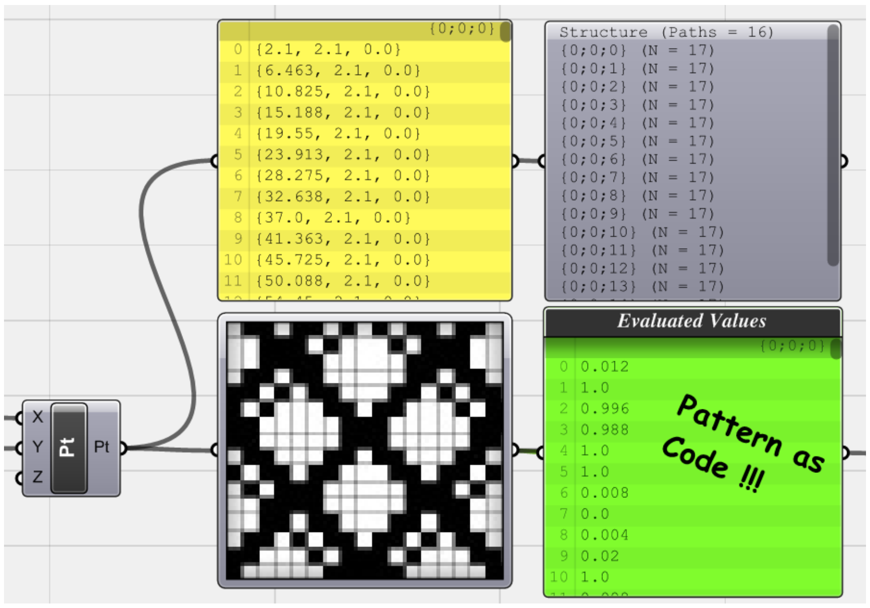
_Why using sub-surfaces instead of divide surface

Using divide surface in this algorithm would result in points on surfaces that start from one edge and end at the opposite one. This means that when pattern repeats, at each border from one complete pattern to the next one, last point of the first pattern and first point of the second pattern would overlap (because their surface edges are seamless and connected). But each cell of the pattern should have its own associate point on the surface and this overlapping would damage the pattern. So it is better to use the concept of 'surface subdivision' and 'area-centroid' instead of dividing the surface.

3_3_4_Weaving Target surface by Pattern

The <Image Sampler> in the first phase of algorithm extracted data from pattern image and converted it into numerical values based on the brightness of the image cells. These values which are associated with black and white cells of the pattern are aimed to tell the algorithm that in which part, yarns should go above and where they should go under the filling yarns (or geometrically above and under the target_srf). Since in the second part of the algorithm, series of points and associate vectors generated to represent basic geometries of yarns, now the data of the first phase of the algorithm should tell the vectors of the second phase that which one should go above and which one under the filling yarns (which is believed that lied on the target surface).

All together this means data coming from the <Image Sampler> should become the vector amplitude for the movement of points on target surface. To handle this, some data management and sorting needed which is better to discuss inside the experiment.

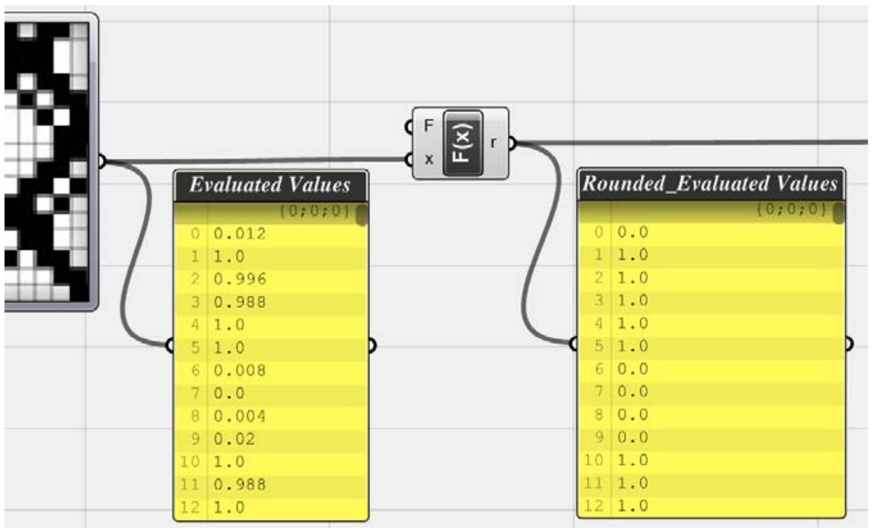


```
{0;0;0}
0  {2.1, 2.1, 0.0}
1  {6.463, 2.1, 0.0}
2  {10.825, 2.1, 0.0}
3  {15.188, 2.1, 0.0}
4  {19.55, 2.1, 0.0}
5  {23.913, 2.1, 0.0}
6  {28.275, 2.1, 0.0}
7  {32.638, 2.1, 0.0}
8  {37.0, 2.1, 0.0}
9  {41.363, 2.1, 0.0}
10 {45.725, 2.1, 0.0}
11 {50.088, 2.1, 0.0}
```

```
Structure (Paths = 16)
{0;0;0} (N = 17)
{0;0;1} (N = 17)
{0;0;2} (N = 17)
{0;0;3} (N = 17)
{0;0;4} (N = 17)
{0;0;5} (N = 17)
{0;0;6} (N = 17)
{0;0;7} (N = 17)
{0;0;8} (N = 17)
{0;0;9} (N = 17)
{0;0;10} (N = 17)
{0;0;11} (N = 17)
{0;0;12} (N = 17)
{0;0;13} (N = 17)
```

```
Evaluated Values
{0;0;0}
0  0.012
1  1.0
2  0.996
3  0.988
4  1.0
5  1.0
6  0.008
7  0.0
8  0.004
9  0.02
10 1.0
```

Fig.3.24. Let's have a look back at the output of the <Image Sampler>. The selected green <Panel> shows that the output is comprised of single real numbers. But looking again, it is noticeable that the list mainly comprised of 0 and 1 values and others are very close to them (i.e. 0.02, 0.004 or 0.996, 0.988 and so on). This happened because of the brightness and contrast of the image which in some pixels is not exactly black (RGB: 0,0,0) or white (RGB:255,255,255). Since for the purpose of this algorithm it does not change the pattern and the algorithm only wanted black or white cells, at first step it should be fixed and values should be converted into 0 and 1 only.

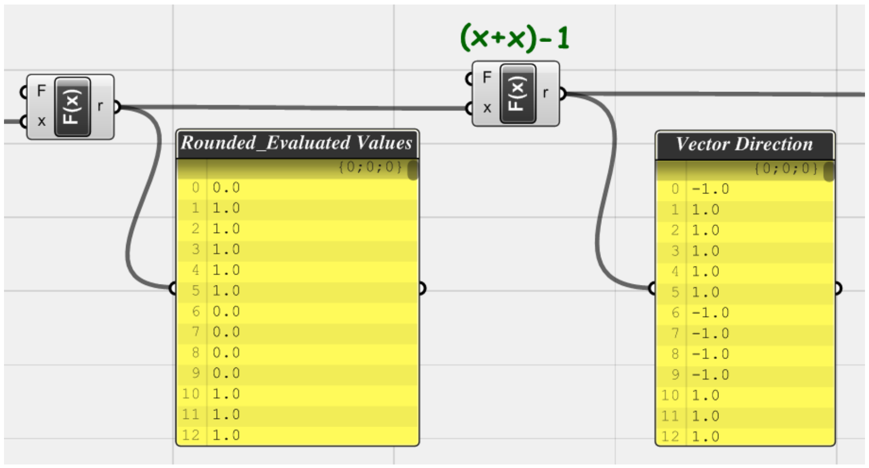


```
Evaluated Values
{0;0;0}
0  0.012
1  1.0
2  0.996
3  0.988
4  1.0
5  1.0
6  0.008
7  0.0
8  0.004
9  0.02
10 1.0
11 0.988
12 1.0
```

```
Rounded_Evaluated Values
{0;0;0}
0  0.0
1  1.0
2  1.0
3  1.0
4  1.0
5  1.0
6  0.0
7  0.0
8  0.0
9  0.0
10 1.0
11 1.0
12 1.0
```

*Fig.3.25. Using a <Function> component by **F(X)=Round(X)**, all values are Rounded into their nearest decimal values which are 0 and 1. Now the list comprised of only 0 and 1 values.*

All Evaluated values from <Image Sampler> are going to be used as vector amplitudes. These vectors should push yarns above or under the filling yarns which means they should be vectors with positive (+) or negative (-) directions to tell yarns to move to which side. Each vector in this definition should have positive or negative magnitude, but all with the same size. This means that if the magnitude of vectors is decided to be 2.345 then for each point, based on the pattern, if wanted to go above filling yarn, it should be +2.345 and if wanted to go under the filling yarn, it should be -2.345. The Absolute magnitude of vectors can be set manually and controllable, but the direction of vector (either positive or negative) should be extracted from the numerical values of source image.

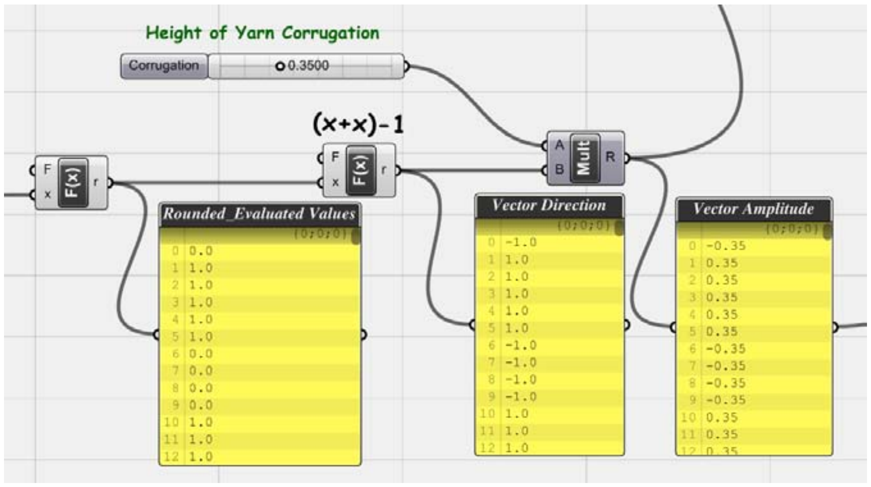


```
Rounded_Evaluated Values      Vector Direction
{0;0;0}                       {0;0;0}
0  0.0                        0  -1.0
1  1.0                        1  1.0
2  1.0                        2  1.0
3  1.0                        3  1.0
4  1.0                        4  1.0
5  1.0                        5  1.0
6  0.0                        6  -1.0
7  0.0                        7  -1.0
8  0.0                        8  -1.0
9  0.0                        9  -1.0
10 1.0                        10 1.0
11 1.0                        11 1.0
12 1.0                        12 1.0
```

Fig.3.26. In essence, numerical values from <Image Sampler> are going to define which vector should have positive magnitude, and which one negative. The list of numerical values comprised of 0 and 1, but the algorithm needs -1 and +1 to define vectors' direction. This means in the source list, all +1 values can remain unchanged and all 0 values should convert into -1. To do this, a simple mathematical operation, using a function defined as: ***F(x)=(X+X)-1*** *applied to the data list. Using this function, the resulted list of data is what that defines the direction of each vector based on the pattern.*

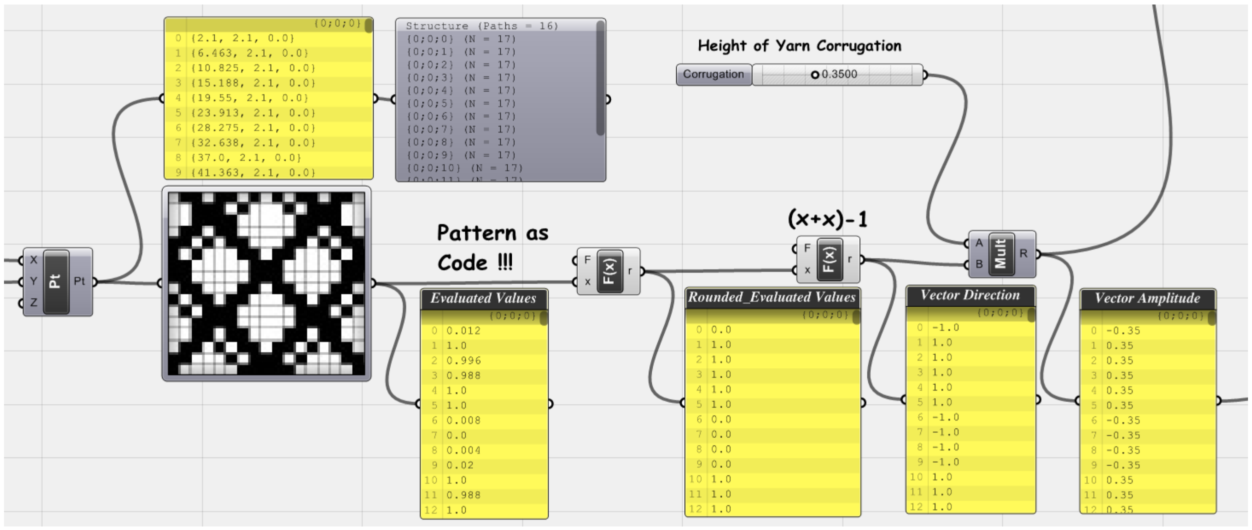
X=1 > F(X)=(1+1)-1= +1

X=0 > F(X)=(0+0)-1= -1



```
Rounded_Evaluated Values   Vector Direction   Vector Amplitude
{0;0;0}                    {0;0;0}            {0;0;0}
0  0.0                     0  -1.0            0  -0.35
1  1.0                     1  1.0             1  0.35
2  1.0                     2  1.0             2  0.35
3  1.0                     3  1.0             3  0.35
4  1.0                     4  1.0             4  0.35
5  1.0                     5  1.0             5  0.35
6  0.0                     6  -1.0            6  -0.35
7  0.0                     7  -1.0            7  -0.35
8  0.0                     8  -1.0            8  -0.35
9  0.0                     9  -1.0            9  -0.35
10 1.0                     10 1.0             10 0.35
11 1.0                     11 1.0             11 0.35
12 1.0                     12 1.0             12 0.35
```

Fig.3.27. Finally, using a <Multiply> component and a <Slider>, the list of vector amplitudes is now ready to apply on points.



```
{0;0;0}
0 {2.1, 2.1, 0.0}
1 {6.463, 2.1, 0.0}
2 {10.825, 2.1, 0.0}
3 {15.188, 2.1, 0.0}
4 {19.55, 2.1, 0.0}
5 {23.913, 2.1, 0.0}
6 {28.275, 2.1, 0.0}
7 {32.638, 2.1, 0.0}
8 {37.0, 2.1, 0.0}
9 {41.363, 2.1, 0.0}
```

```
Structure (Paths = 16)
{0;0;0} (N = 17)
{0;0;1} (N = 17)
{0;0;2} (N = 17)
{0;0;3} (N = 17)
{0;0;4} (N = 17)
{0;0;5} (N = 17)
{0;0;6} (N = 17)
{0;0;7} (N = 17)
{0;0;8} (N = 17)
{0;0;9} (N = 17)
{0;0;10} (N = 17)
{0;0;11} (N = 17)
```

```
Evaluated Values
{0;0;0}
0  0.012
1  1.0
2  0.996
3  0.988
4  1.0
5  1.0
6  0.008
7  0.0
8  0.004
9  0.02
10 1.0
11 0.988
12 1.0
```

```
Rounded_Evaluated Values
{0;0;0}
0  0.0
1  1.0
2  1.0
3  1.0
4  1.0
5  1.0
6  0.0
7  0.0
8  0.0
9  0.0
10 1.0
11 1.0
12 1.0
```

```
Vector Direction
{0;0;0}
0  -1.0
1  1.0
2  1.0
3  1.0
4  1.0
5  1.0
6  -1.0
7  -1.0
8  -1.0
9  -1.0
10 1.0
11 1.0
12 1.0
```

```
Vector Amplitude
{0;0;0}
0  -0.35
1  0.35
2  0.35
3  0.35
4  0.35
5  0.35
6  -0.35
7  -0.35
8  -0.35
9  -0.35
10 0.35
11 0.35
12 0.35
```

Fig.3.28. Generation of vector amplitudes as the result of image sampling process.

_Matching Data Lists

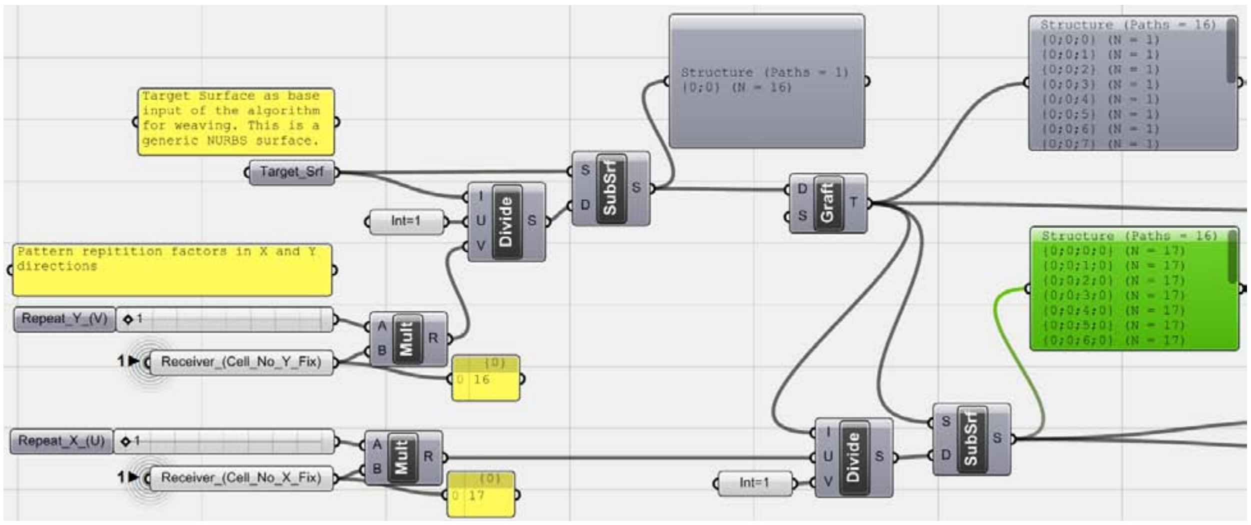
The number of values that the <Image Sampler> generated so far is as much as cells that pattern has. But as implemented inside the algorithm, plotting the pattern onto Target_Surface has a pattern repetition factor. So while the pattern repeated and resulted in more points and vectors than original pattern cells, the list of vector amplitudes also needed to repeat as well, to match the vectors list.

Let's preview the situation in Figure.3.29 again to have an insight into the rest of the algorithm analytically. Selected <Param Viewer>s in all three figures are our target to compare.

1- In the first figure, while both slider values are set to 1, the <Param Viewer> shows 16 paths of data (equal to the value from <Recieiver_(**Cell_No_Y**_Fix)>) and 17 values in each branch of data (equal to the value from <Recieiver_(**Cell_No_X**_Fix)>).

2- In the second figure, the <Repeat_**Y**_(V)> slider set to 2 and doubled the pattern in Y direction. As a result, the number of **data paths doubled**, but the amount of values in each path remained the same.

3- In the third image, opposite the previous step, the <Repeat_**X**_(U)> slider set to 2 and doubled the pattern in X direction. As a result, in this time the number of data paths remained unchanged, but the amount of **values in each path doubled**.



Target Surface as base input of the algorithm for weaving. This is a generic NURBS surface.

Pattern repitition factors in X and Y directions

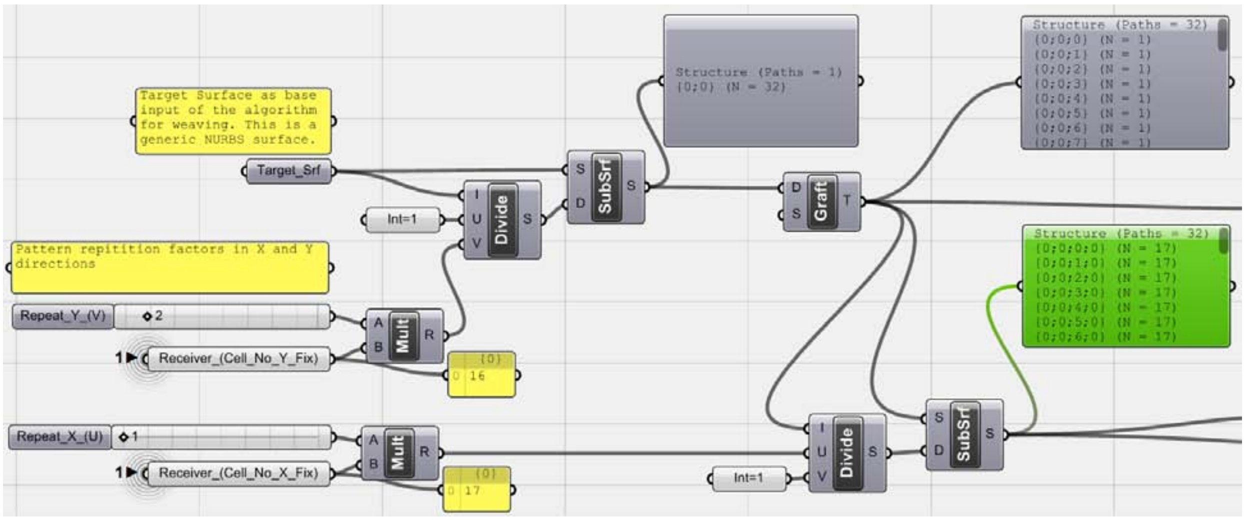
```
{0}
0 16
```

```
{0}
0 17
```

```
Structure (Paths = 1)
{0;0} (N = 16)
```

```
Structure (Paths = 16)
{0;0;0} (N = 1)
{0;0;1} (N = 1)
{0;0;2} (N = 1)
{0;0;3} (N = 1)
{0;0;4} (N = 1)
{0;0;5} (N = 1)
{0;0;6} (N = 1)
{0;0;7} (N = 1)
```

```
Structure (Paths = 16)
{0;0;0;0} (N = 17)
{0;0;1;0} (N = 17)
{0;0;2;0} (N = 17)
{0;0;3;0} (N = 17)
{0;0;4;0} (N = 17)
{0;0;5;0} (N = 17)
{0;0;6;0} (N = 17)
```



Target Surface as base input of the algorithm for weaving. This is a generic NURBS surface.

Pattern repitition factors in X and Y directions

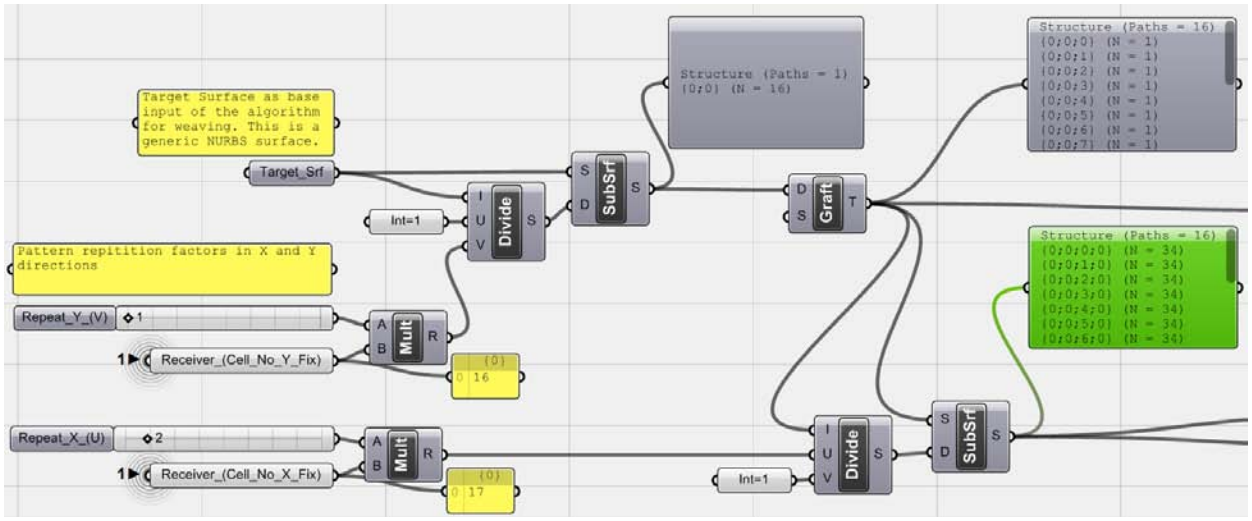
```
{0}
0 16
```

```
{0}
0 17
```

```
Structure (Paths = 1)
{0;0} (N = 32)
```

```
Structure (Paths = 32)
{0;0;0} (N = 1)
{0;0;1} (N = 1)
{0;0;2} (N = 1)
{0;0;3} (N = 1)
{0;0;4} (N = 1)
{0;0;5} (N = 1)
{0;0;6} (N = 1)
{0;0;7} (N = 1)
```

```
Structure (Paths = 32)
{0;0;0;0} (N = 17)
{0;0;1;0} (N = 17)
{0;0;2;0} (N = 17)
{0;0;3;0} (N = 17)
{0;0;4;0} (N = 17)
{0;0;5;0} (N = 17)
{0;0;6;0} (N = 17)
```



Target Surface as base input of the algorithm for weaving. This is a generic NURBS surface.

Pattern repitition factors in X and Y directions

```
{0}
0 16
```

```
{0}
0 17
```

```
Structure (Paths = 1)
{0;0} (N = 16)
```

```
Structure (Paths = 16)
{0;0;0} (N = 1)
{0;0;1} (N = 1)
{0;0;2} (N = 1)
{0;0;3} (N = 1)
{0;0;4} (N = 1)
{0;0;5} (N = 1)
{0;0;6} (N = 1)
{0;0;7} (N = 1)
```

```
Structure (Paths = 16)
{0;0;0;0} (N = 34)
{0;0;1;0} (N = 34)
{0;0;2;0} (N = 34)
{0;0;3;0} (N = 34)
{0;0;4;0} (N = 34)
{0;0;5;0} (N = 34)
{0;0;6;0} (N = 34)
```

Fig.3.29_1.2.3. comparison of data structures in different repetition factors.

Comparing all these data structures in order to match vectors' amplitude list, lets summarize everything: by repeating pattern in X direction, the values for vectors' amplitude should be duplicated inside each branch while repeating pattern in Y direction needs data to be repeated entirely by its branches or let's say data paths should be copied this time and values inside each branch will remain unchanged. To achieve this concept, some data management components should implement into the algorithm.

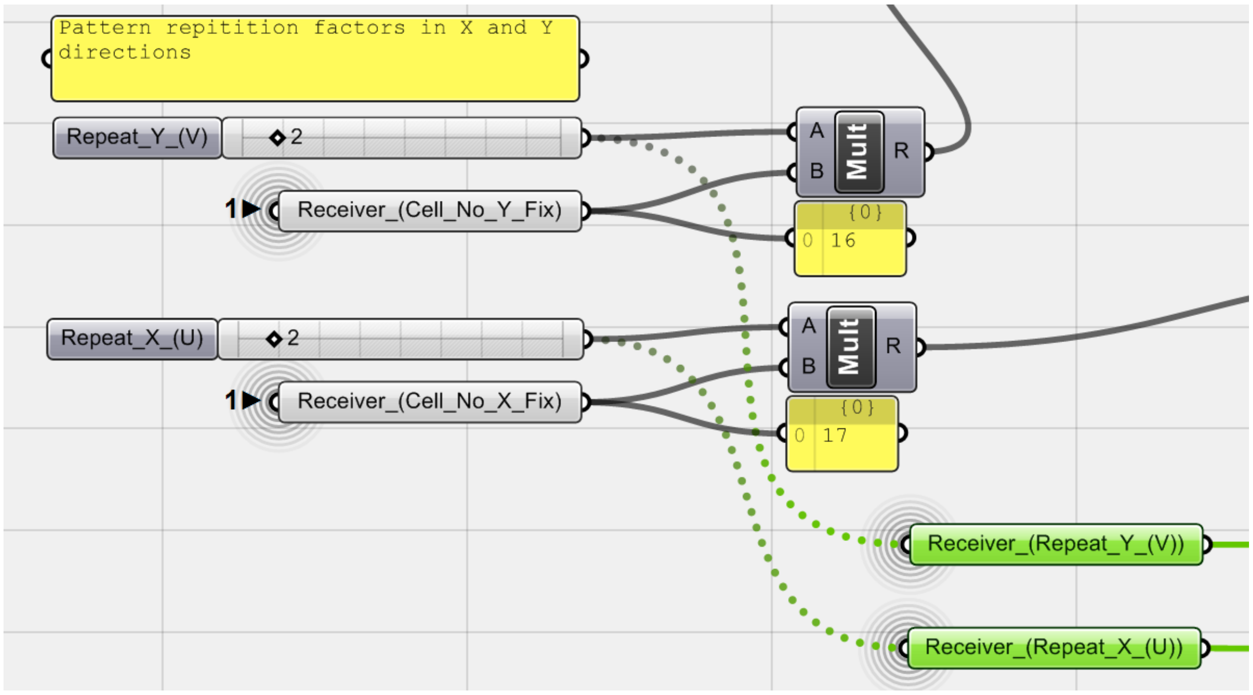
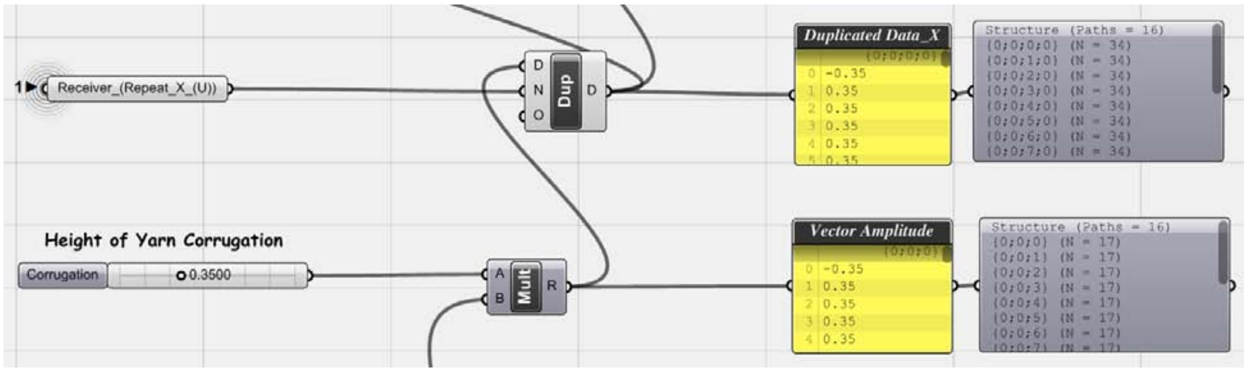


Fig.3.30. Two sliders that define the repetition factors of the pattern are important values in this part of the algorithm. Both are connected to <Receiver> components to affect the number of data duplication.



```
{0;0;0;0}
0 -0.35
1 0.35
2 0.35
3 0.35
4 0.35
5 0.35
```

```
Structure (Paths = 16)
{0;0;0;0} (N = 34)
{0;0;1;0} (N = 34)
{0;0;2;0} (N = 34)
{0;0;3;0} (N = 34)
{0;0;4;0} (N = 34)
{0;0;5;0} (N = 34)
{0;0;6;0} (N = 34)
{0;0;7;0} (N = 34)
```

```
{0;0;0}
0 -0.35
1 0.35
2 0.35
3 0.35
4 0.35
```

```
Structure (Paths = 16)
{0;0;0} (N = 17)
{0;0;1} (N = 17)
{0;0;2} (N = 17)
{0;0;3} (N = 17)
{0;0;4} (N = 17)
{0;0;5} (N = 17)
{0;0;6} (N = 17)
{0;0;7} (N = 17)
```

Fig.3.31. In the first step, data from the last phase (amplitude of vectors) is duplicated by the amount of <Reciever_(Repeat_X_(U))>. As it is shown in <Param Viewer>, the number of data paths remains the same but the number of values in each branch changed. This means every time, when designer changes the number of pattern repetition factor in X direction, the number of values as vectors' amplitude would also change inside each data path.

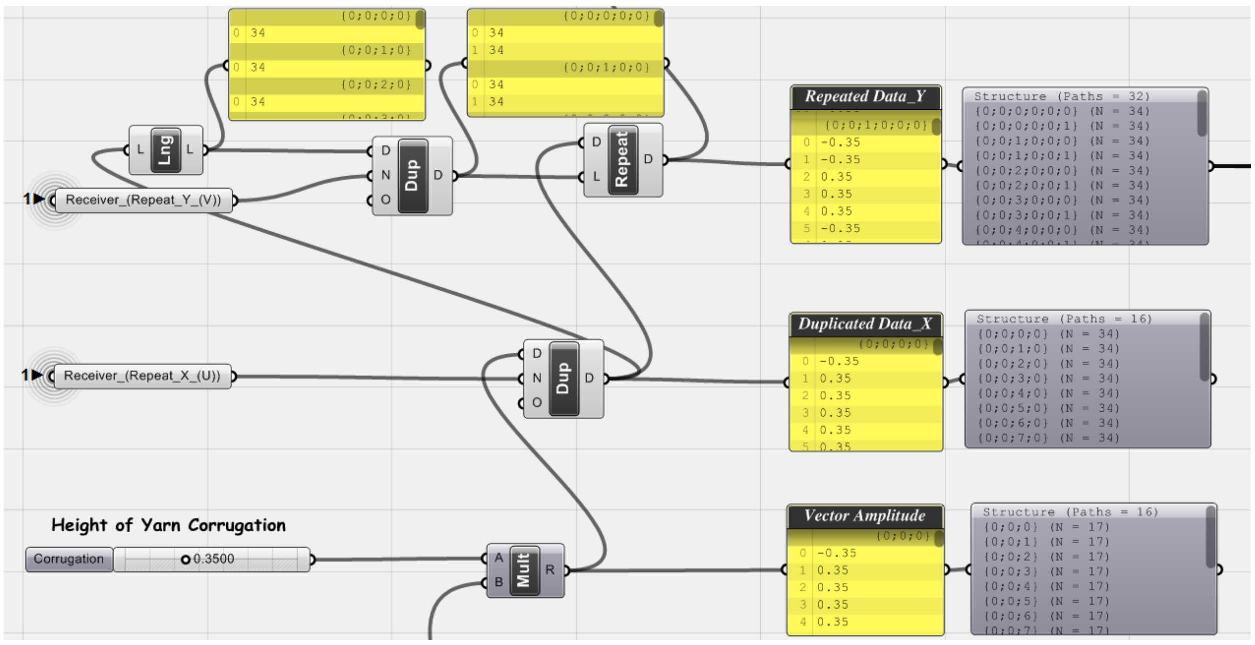


Fig.3.32. In this second part, data list should <Repeat> instead of <Duplicate> because data branches wanted to be copied. A bit complex part of the algorithm is preparation of the (L) port of the <Repeat>. L is the length of final pattern so it should be the amount of values in each branch. That's why the <Length> of the previous <Duplicate> component used. But this value should be duplicated by itself as much as branches we need in the final data list in order to copy data paths. So as a result <Repeat> component repeats all branches of data with all of their values (twice in this example). As it is shown in the third <Param Viewer> now data paths are multiplied by the value comes from <Receiver_(Repeat_Y_(V))>.

Basically with the first part, data duplicated inside each branch, but in second part, data paths repeated with entire values in each path and branches of data tree increased.

Although if one looks at the final <Param Viewer> would say that data duplicated as much as points and vectors and 'ok! Finished', there is still one big problem. To realize the situation let's move all points with prepared vectors' amplitude list and check the result by 'interpolating' displaced points.

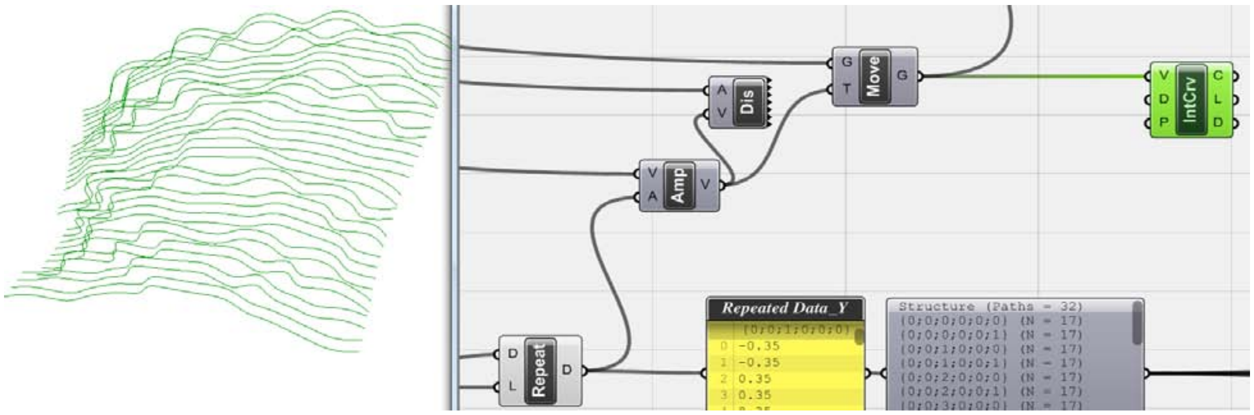


Fig.3.33. As shown in the above figure, although repetition happened (factor=2), but it happened for each yarn right after itself (in Y direction), not for the whole pattern, so each yarn repeated (in this example once) and the result has a pattern of double yarns. It is clear that in the resultant geometry, as what expected, the whole pattern should be repeated not each yarn separately after itself.

Talking with a simplified language of code, if the original data list has a structure for its rows like A,B,C,D, while the desired duplication of this pattern should be A,B,C,D,A,B,C,D but the outcome has a structure like A,A,B,B,C,C,D,D.

This problem should be solved in order to get the complete repeated pattern otherwise the algorithm performs single line repetition instead of pattern repetition. Conceptually there are different methods to solve this problem, but the way that has been chosen works by rearranging the data tree's structure. Let's have an analytical look at the situation:

Isolating data tree by only four branches as an example, the <Param Viewer> would show data paths like this:

{0;0;0;0;0;0}

{0;0;0;0;0;1}

{0;0;1;0;0;0}

{0;0;1;0;0;1}

But as described before, the desired data paths should be:

{0;0;0;0;0;0}

{0;0;1;0;0;0}

{0;0;0;0;0;1}

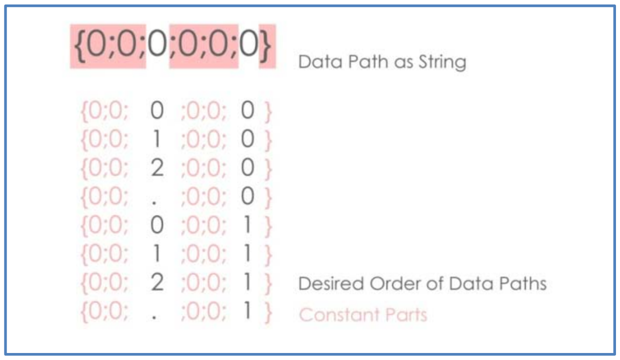
{0;0;1;0;0;1}

So if the branch paths sorted like the above fashion, then textile would be woven as repetition of entire pattern instead of line by line. This means all data paths whose address ends up at same numbers represent one complete pattern.

Here the desired list of data paths should be generated and then the whole data list recalled by these new paths so the result would be the list of values with the planed order. To do this, data paths should be generated as follows:

Fig.3.34. As shown in the above diagram, beside constant bits of data paths, two different lists of numbers should be generated:

1- *The first list is an iterative list of main branch paths. The number of values is equal to the pattern's cell number in Y direction and the number of iteration (Duplication) is equal to the number of pattern repetition in Y direction.*
2- *The second list is a repetitive list of small branches (last items). Here in contrast with the first list, the values (0,1,...) are coming from the number of repetition of pattern, and their duplication number is equal to the number of pattern's cells in Y direction.*



Although the algorithm is not something special but the concept needs a little bit of thinking and concentration. Following next steps would clarify the whole situation.

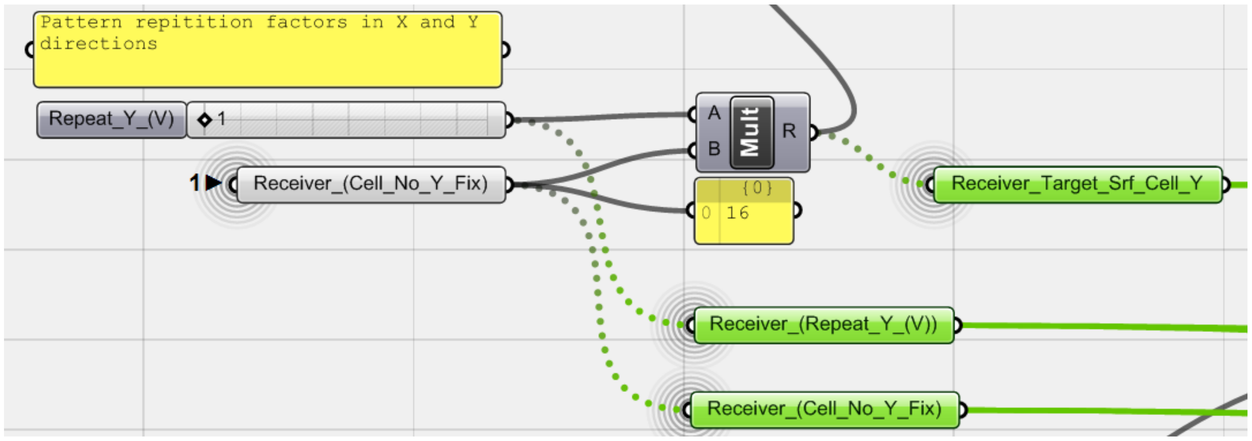


Fig.3.35. In the first step, three <Receiver> components defined to avoid distraction in canvas. They all attach to the values coming from the second phase of the algorithm, or Target_Surface Subdivision. Pay attention to the names. They are repeated in the rest of the algorithm so many times.

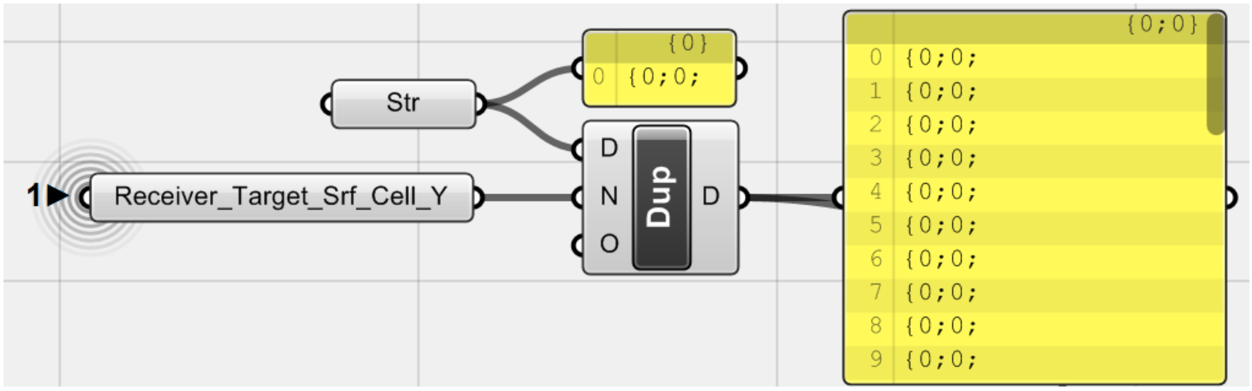


Fig.3.36. Five different parts of Data paths should be generated. The first one is a string which is **' {0;0; '**. This item <duplicate>d and the number of duplication is set by target surface cells number in Y direction (the whole list length).

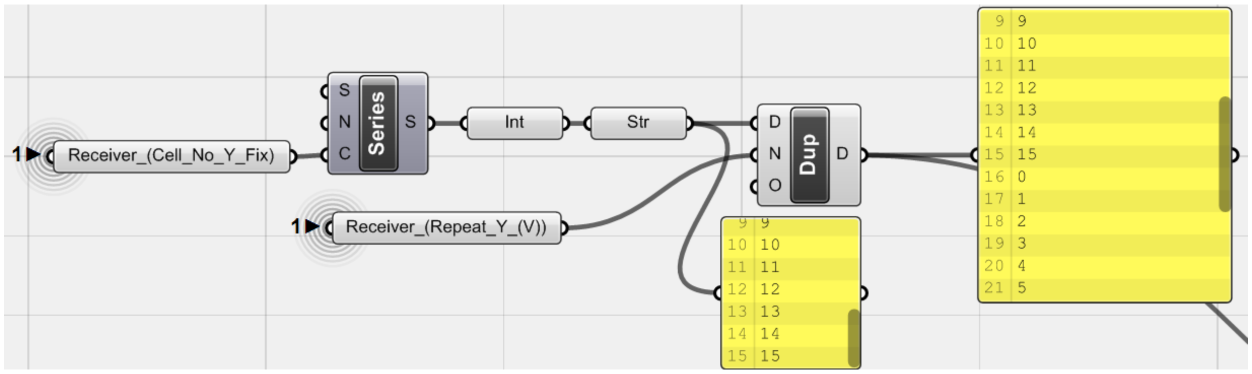
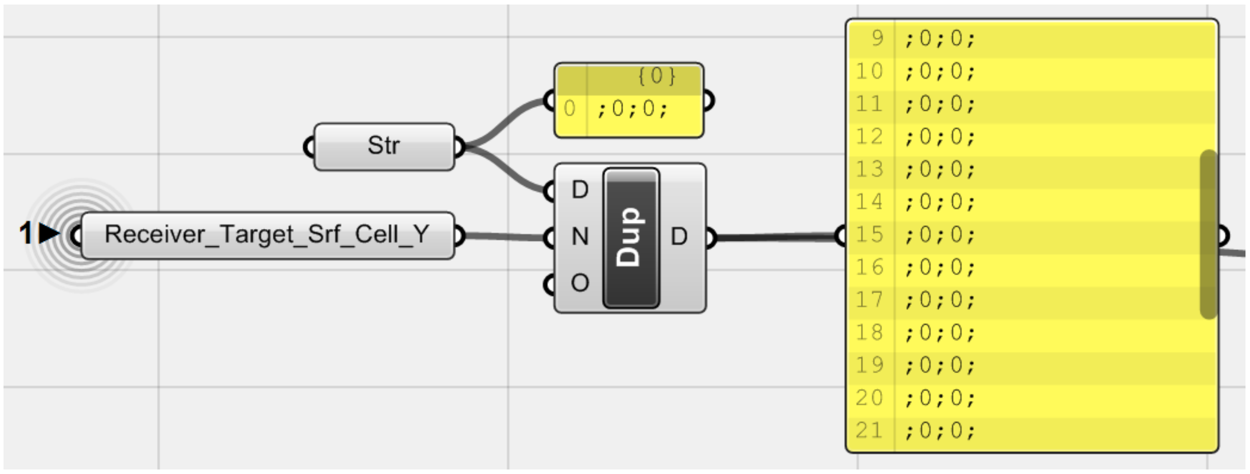


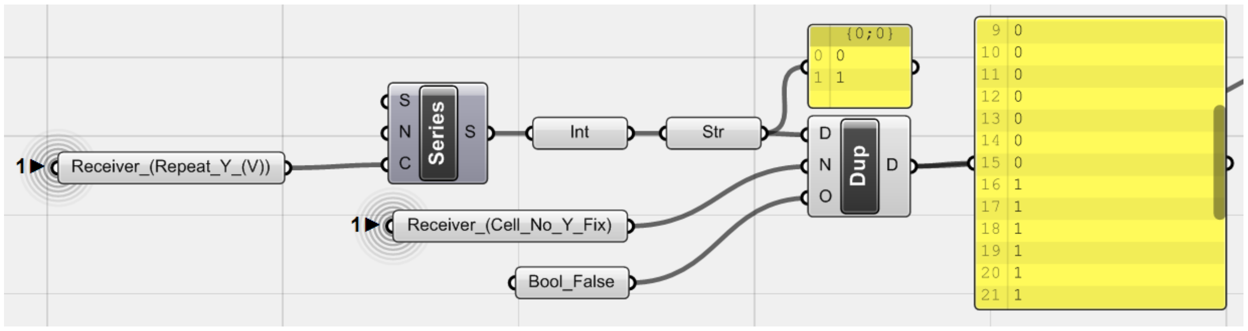
Fig.3.37. The second part of the string: a <Series> of numbers generated with the length of pattern's cells in Y direction. This list of numbers converted to <Integer>s (0.0, 1.0, ... > 0, 1, ...) and then into <String>s and then duplicated. The number of duplication is equal to the pattern repetition in Y direction.



```
{0}
0 ;0;0;
```

```
9  ;0;0;
10 ;0;0;
11 ;0;0;
12 ;0;0;
13 ;0;0;
14 ;0;0;
15 ;0;0;
16 ;0;0;
17 ;0;0;
18 ;0;0;
19 ;0;0;
20 ;0;0;
21 ;0;0;
```

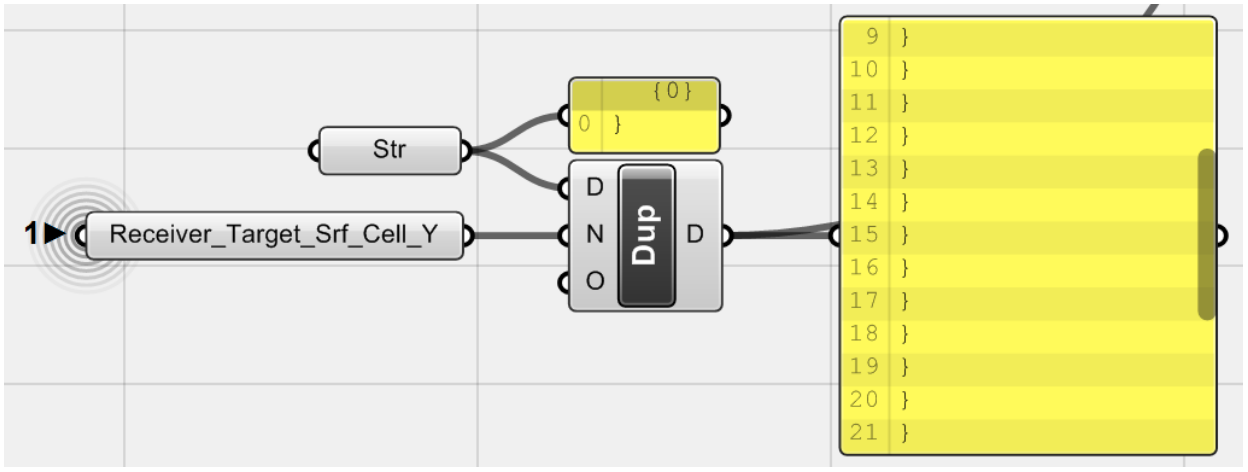
*Fig.3.38. The third bit: a <string> of ‘ **;0;0;** ’ is duplicated like the first bit. The number of duplication is equal to the number of target surface cells in Y direction (the whole list length).*



```
{0;0}
0 0
1 1
```

```
9  0
10 0
11 0
12 0
13 0
14 0
15 0
16 1
17 1
18 1
19 1
20 1
21 1
```

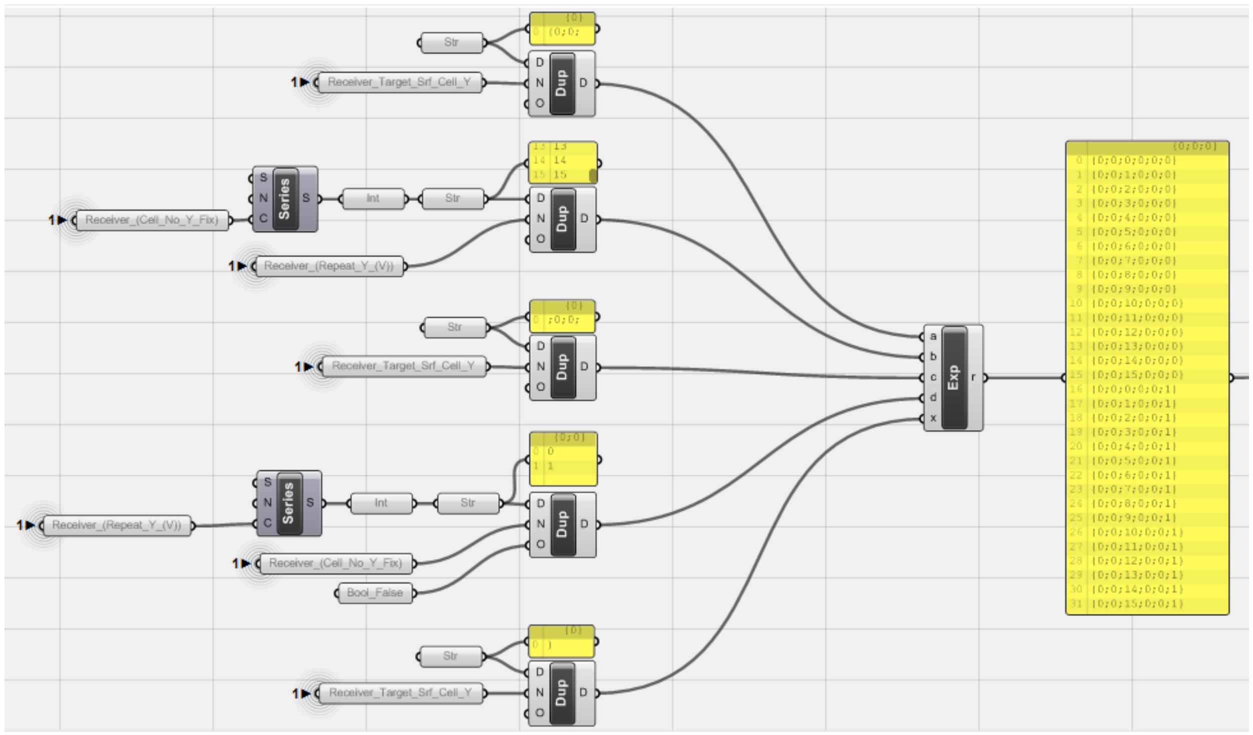
*Fig.3.39. The fourth bit of string: here another <Series> of numbers generated with the length of pattern repetition in Y direction. This list converted into <Integer>s and then <String>s and then <Duplicate>d. the number of duplication is equal to the number of pattern’s cells in Y direction. Notice that the Boolean switch for retaining the order is set to ‘**False**’.*



```
{0}
0 }
```

```
9  }
10 }
11 }
12 }
13 }
14 }
15 }
16 }
17 }
18 }
19 }
20 }
21 }
```

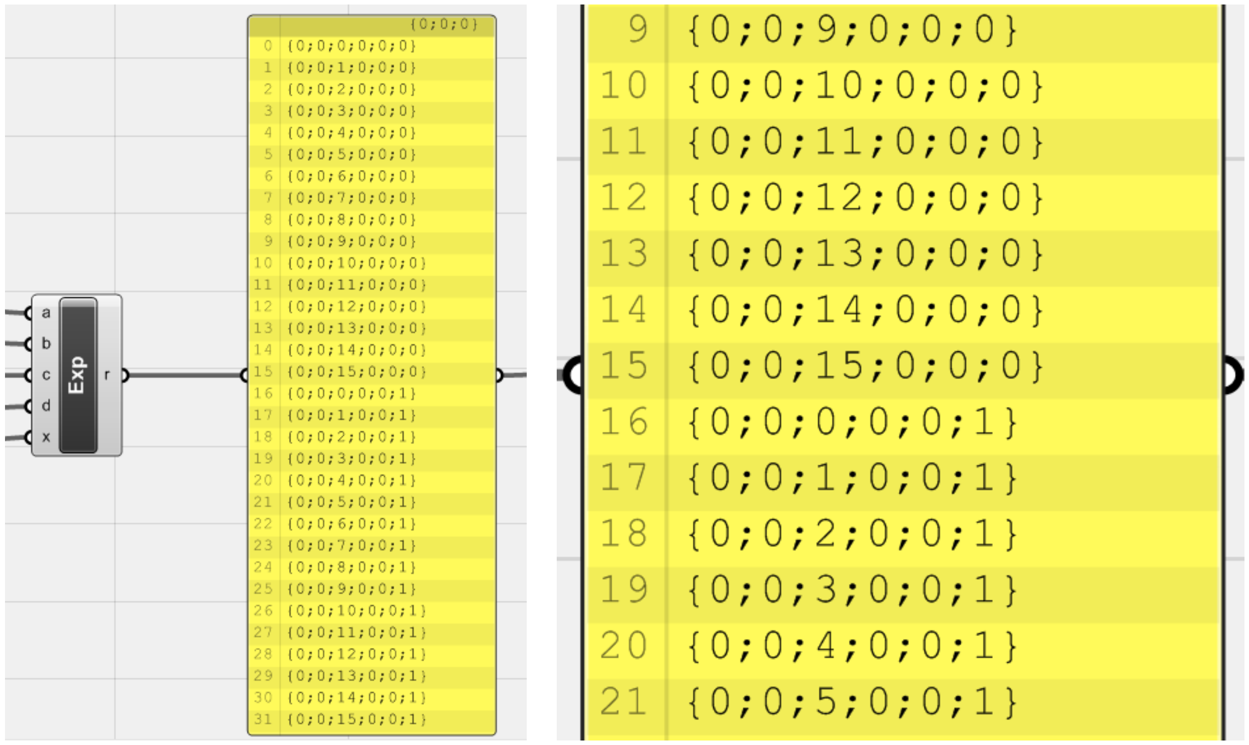
*Fig.3.40. Final bit, a <string> of only ‘ **}** ’ again duplicated by the number of target surface cells in Y direction (the whole list length).*



```
{0;0;0}
 0 {0;0;0;0;0;0}
 1 {0;0;1;0;0;0}
 2 {0;0;2;0;0;0}
 3 {0;0;3;0;0;0}
 4 {0;0;4;0;0;0}
 5 {0;0;5;0;0;0}
 6 {0;0;6;0;0;0}
 7 {0;0;7;0;0;0}
 8 {0;0;8;0;0;0}
 9 {0;0;9;0;0;0}
10 {0;0;10;0;0;0}
11 {0;0;11;0;0;0}
12 {0;0;12;0;0;0}
13 {0;0;13;0;0;0}
14 {0;0;14;0;0;0}
15 {0;0;15;0;0;0}
16 {0;0;0;0;0;1}
17 {0;0;1;0;0;1}
18 {0;0;2;0;0;1}
19 {0;0;3;0;0;1}
20 {0;0;4;0;0;1}
21 {0;0;5;0;0;1}
22 {0;0;6;0;0;1}
23 {0;0;7;0;0;1}
24 {0;0;8;0;0;1}
25 {0;0;9;0;0;1}
26 {0;0;10;0;0;1}
27 {0;0;11;0;0;1}
28 {0;0;12;0;0;1}
29 {0;0;13;0;0;1}
30 {0;0;14;0;0;1}
31 {0;0;15;0;0;1}
```

Fig.3.41. Looking back at what has been done so far, all fifth parts of the string merged together, using a function with Variable Expression. The function set to ***F(a,b,c,d,x) = a & b & c & d & x***

This function with ***&*** *between string variables adds different parts of the string together.*



```
{0;0;0}
 0 {0;0;0;0;0;0}
 1 {0;0;1;0;0;0}
 2 {0;0;2;0;0;0}
 3 {0;0;3;0;0;0}
 4 {0;0;4;0;0;0}
 5 {0;0;5;0;0;0}
 6 {0;0;6;0;0;0}
 7 {0;0;7;0;0;0}
 8 {0;0;8;0;0;0}
 9 {0;0;9;0;0;0}
10 {0;0;10;0;0;0}
11 {0;0;11;0;0;0}
12 {0;0;12;0;0;0}
13 {0;0;13;0;0;0}
14 {0;0;14;0;0;0}
15 {0;0;15;0;0;0}
16 {0;0;0;0;0;1}
17 {0;0;1;0;0;1}
18 {0;0;2;0;0;1}
19 {0;0;3;0;0;1}
20 {0;0;4;0;0;1}
21 {0;0;5;0;0;1}
22 {0;0;6;0;0;1}
23 {0;0;7;0;0;1}
24 {0;0;8;0;0;1}
25 {0;0;9;0;0;1}
26 {0;0;10;0;0;1}
27 {0;0;11;0;0;1}
28 {0;0;12;0;0;1}
29 {0;0;13;0;0;1}
30 {0;0;14;0;0;1}
31 {0;0;15;0;0;1}
```

```
 9 {0;0;9;0;0;0}
10 {0;0;10;0;0;0}
11 {0;0;11;0;0;0}
12 {0;0;12;0;0;0}
13 {0;0;13;0;0;0}
14 {0;0;14;0;0;0}
15 {0;0;15;0;0;0}
16 {0;0;0;0;0;1}
17 {0;0;1;0;0;1}
18 {0;0;2;0;0;1}
19 {0;0;3;0;0;1}
20 {0;0;4;0;0;1}
21 {0;0;5;0;0;1}
```

Fig.3.42. Looking at result, series of data paths has been generated with planed order as discussed before.

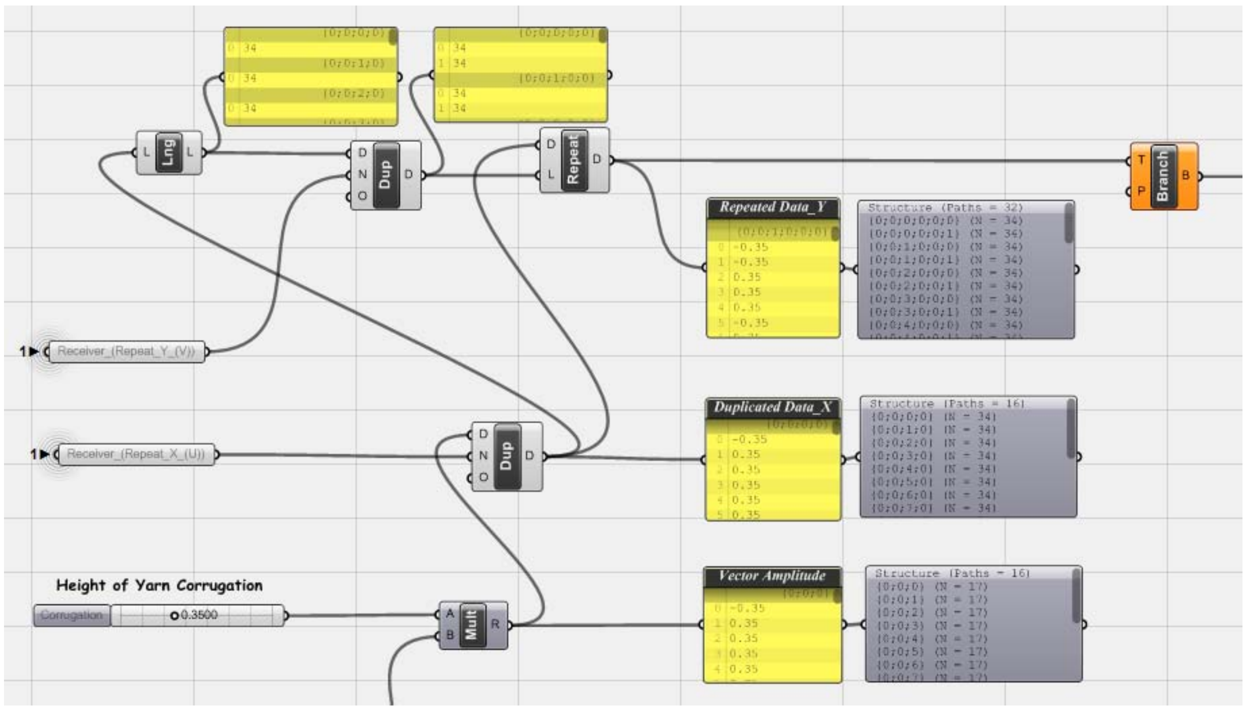


Fig.3.43. Now it is time to check these data paths. First of all, a <Tree Branch> component used to extract data by predefined data paths. This component fed by data that has been provided in the third phase of the algorithm by <Repeat>ing all amplitude values.

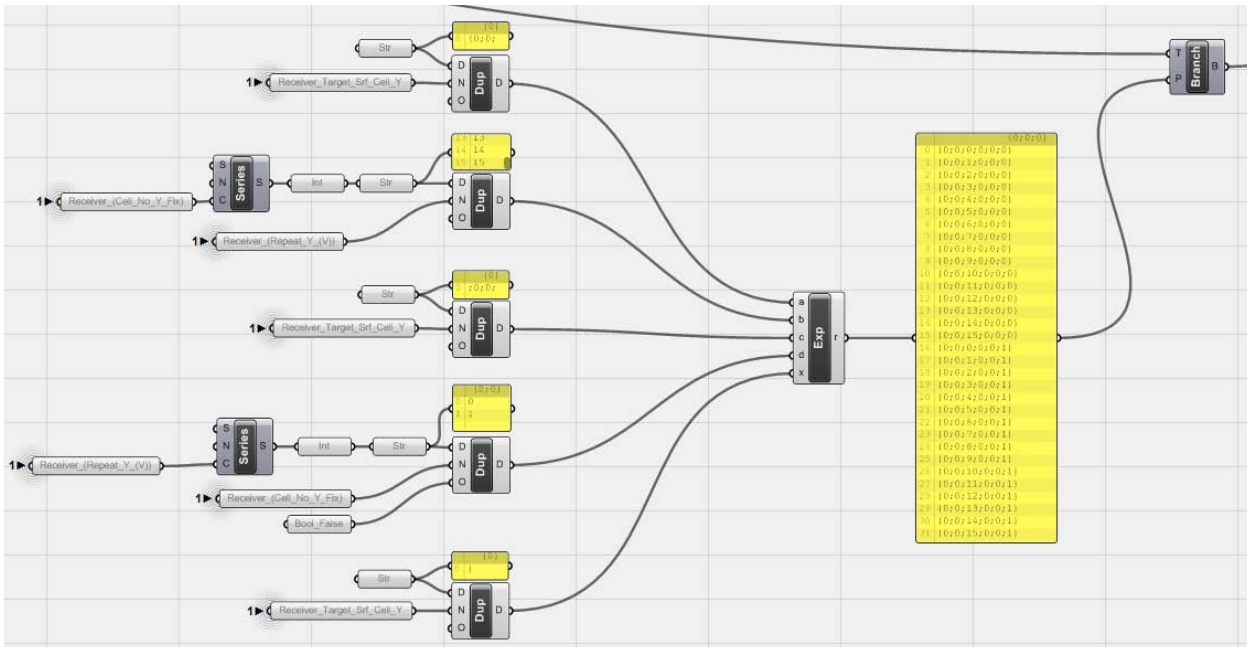
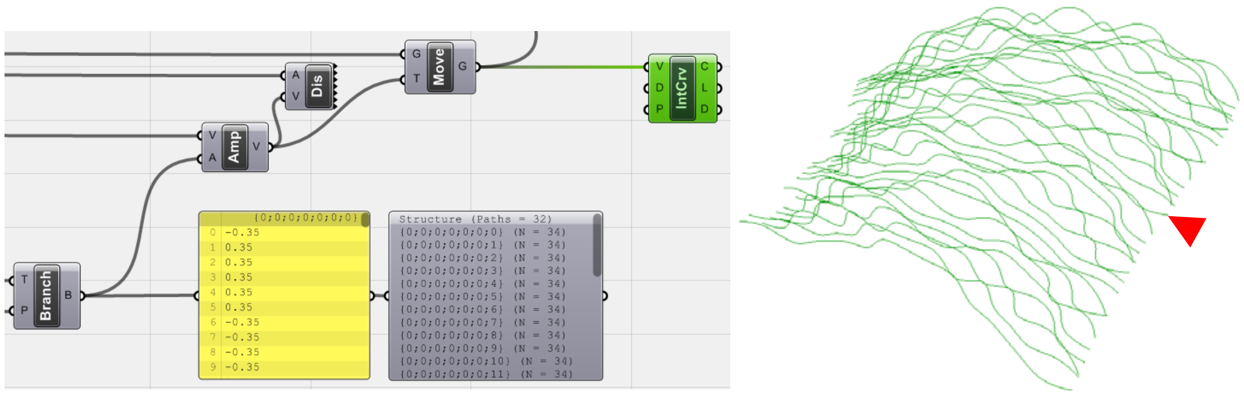


Fig.3.44. Finally the <Tree Branch>'s P port, fed by generated data paths. Let's check the result in geometry.



```
{0;0;0;0;0;0;0}
0 -0.35
1 0.35
2 0.35
3 0.35
4 0.35
5 0.35
6 -0.35
7 -0.35
8 -0.35
9 -0.35
```

```
Structure (Paths = 32)
{0;0;0;0;0;0;0} (N = 34)
{0;0;0;0;0;0;1} (N = 34)
{0;0;0;0;0;0;2} (N = 34)
{0;0;0;0;0;0;3} (N = 34)
{0;0;0;0;0;0;4} (N = 34)
{0;0;0;0;0;0;5} (N = 34)
{0;0;0;0;0;0;6} (N = 34)
{0;0;0;0;0;0;7} (N = 34)
{0;0;0;0;0;0;8} (N = 34)
{0;0;0;0;0;0;9} (N = 34)
{0;0;0;0;0;0;10} (N = 34)
{0;0;0;0;0;0;11} (N = 34)
```

Fig.3.45. Redistributed/Resorted data has been used as vector amplitudes and then points <move>d and again <interpolate>d curves are generated to check the result. Although the pattern is not recognizable in the above figure but it is pretty much clear that single lines did not repeat this time and the whole pattern duplicated from the red point.

3_3_5_Yarn Geometry Generation

The last phase of the algorithm is to generate yarn geometries. In this experiment, again <pipe> geometries has been chosen but other type of extrusion along curves are possible methods. Another point is that to make the rest of the algorithm simple, filling yarns designed simple, without corrugation.

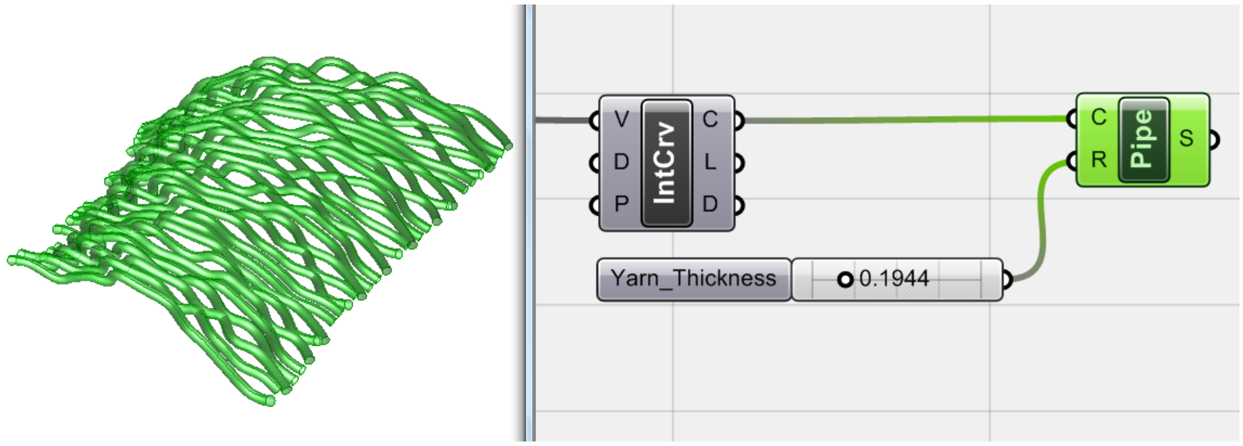
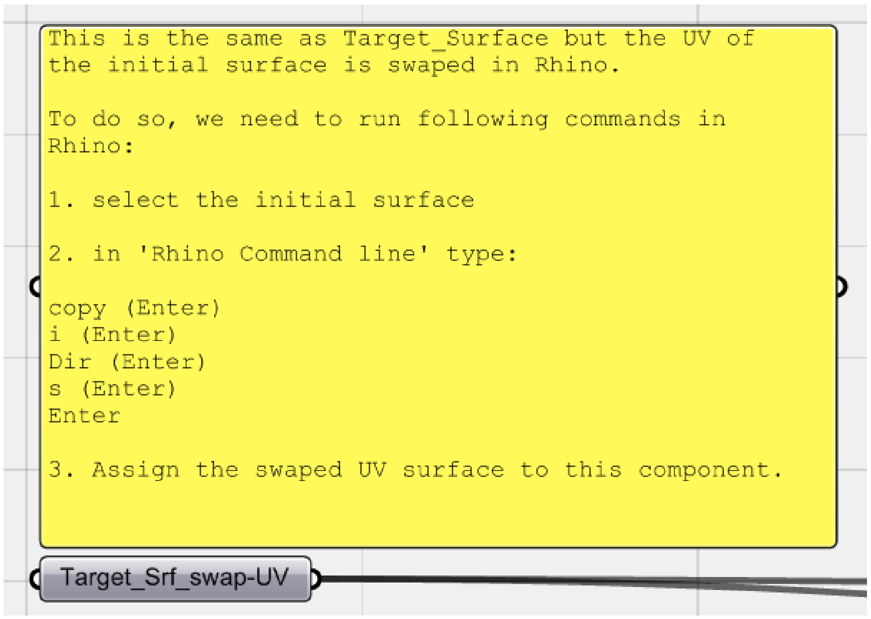


Fig.3.46. Interpolated curves used to generate <Pipe> geometries by controllable radius as <Yarn_Thickness>.



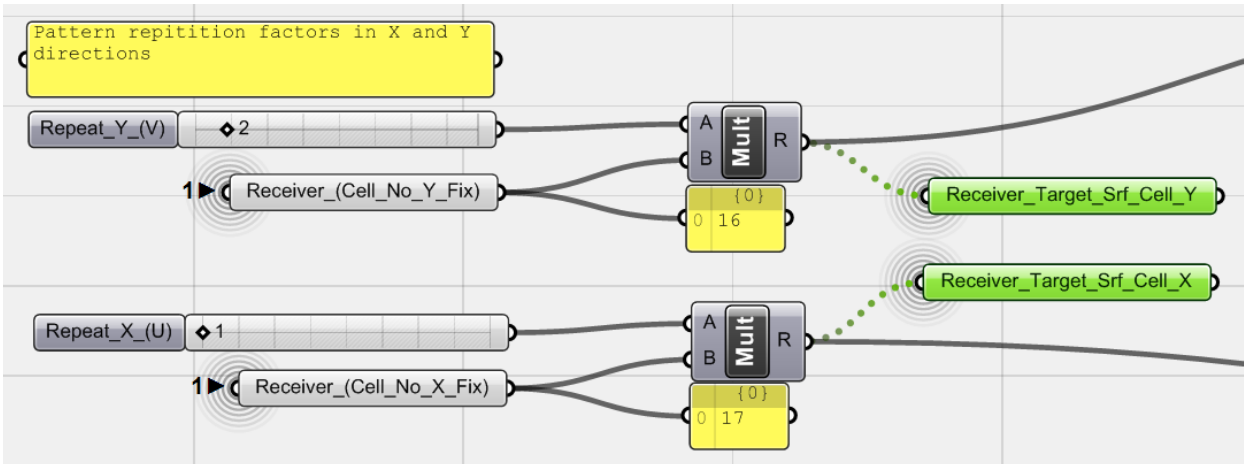
```
This is the same as Target_Surface but the UV of
the initial surface is swaped in Rhino.

To do so, we need to run following commands in
Rhino:

1. select the initial surface

2. in 'Rhino Command line' type:

copy (Enter)
i (Enter)
Dir (Enter)
s (Enter)
Enter

3. Assign the swaped UV surface to this component.
```

Fig.3.47. As explained in the first Loom algorithm, here to generate filling yarns a UV_Swaped surface again introduced to canvas. To do this, in Rhino these commands should be applied to the target surface:

Select the surface and then:

Copy (Enter) / i (Enter) / Dir (Enter) / s (Enter) / (Enter)

Assign the second surface to the <Target_Srf_swap-UV>



Pattern repitition factors in X and Y directions

Fig.3.48. Values of main surface subdivision again assigned to two <Receiver> components to subdivide the second surface as well.

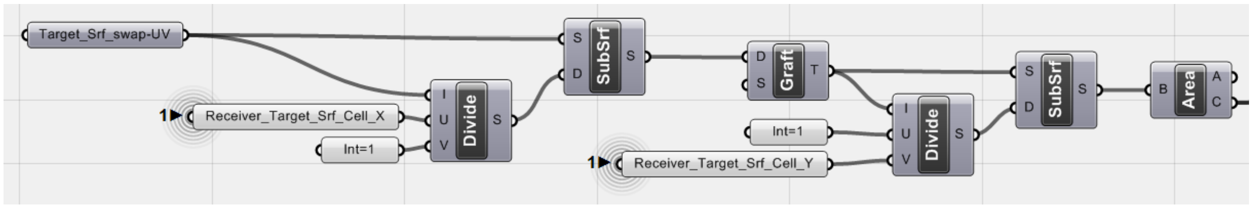


Fig.3.49. Second surface subdivision (this part of the algorithm is almost the same as second phase when the <Target_Srf> subdivided, instead here for filling yarns in a reverse fashion first U direction subdivided and then V direction): <Target_Srf_swap-UV> domain is divided by a <Divide domain2> using <Receiver_Target_Srf_Cell_X> as ***U*** *port first, and the surface subdivided by these domains. Using <Graft> component, all of them sorted into separate data paths. These <Graft>ed surface domains again divided but this time by <Receiver_Target_Srf_Cell_Y> as* ***V*** *port of <divide domain2> and again surfaces are subdivided. The Area-Centroid of all these surfaces calculated using <Area> component.*

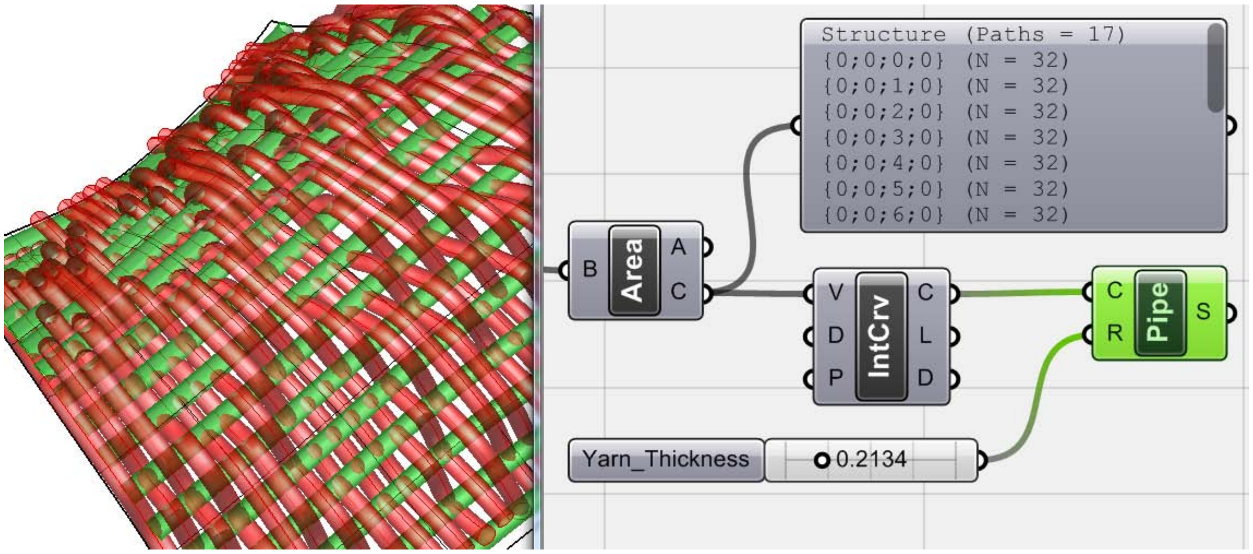

```
Structure  (Paths = 17)
{0;0;0;0}  (N = 32)
{0;0;1;0}  (N = 32)
{0;0;2;0}  (N = 32)
{0;0;3;0}  (N = 32)
{0;0;4;0}  (N = 32)
{0;0;5;0}  (N = 32)
{0;0;6;0}  (N = 32)
```

Fig.3.50. The area-centroids are <Interpolate>ed and resultant curves used to generate <Pipe> geometries as filling yarns of the digital textile, again with controllable radius as <Yarn_Thickness> (Yarn thickness could be set the same as other yarns).

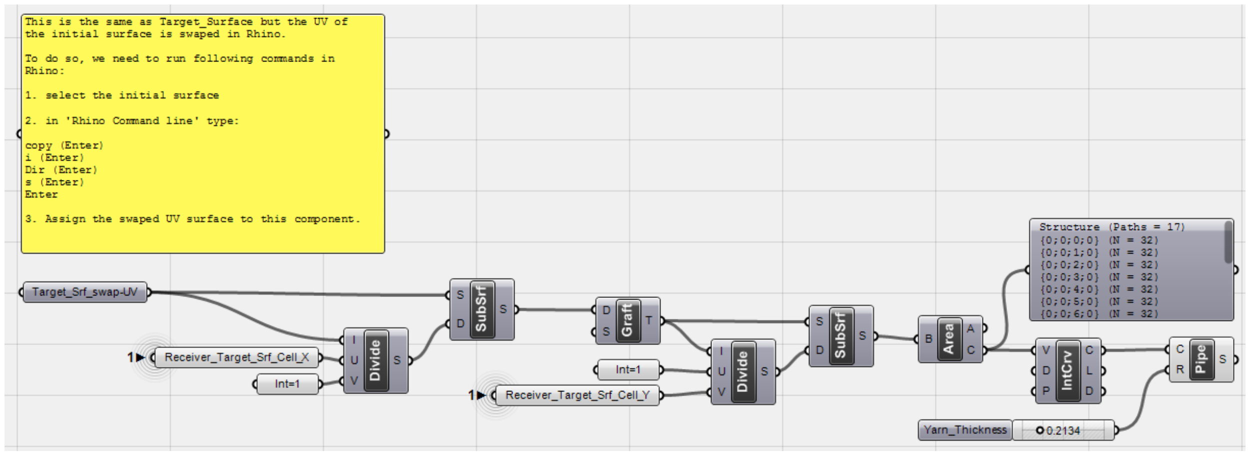

```
This is the same as Target_Surface but the UV of
the initial surface is swaped in Rhino.

To do so, we need to run following commands in
Rhino:

1. select the initial surface

2. in 'Rhino Command line' type:

copy (Enter)
i (Enter)
Dir (Enter)
s (Enter)
Enter

3. Assign the swaped UV surface to this component.
```
```
Structure  (Paths = 17)
{0;0;0;0}  (N = 32)
{0;0;1;0}  (N = 32)
{0;0;2;0}  (N = 32)
{0;0;3;0}  (N = 32)
{0;0;4;0}  (N = 32)
{0;0;5;0}  (N = 32)
{0;0;6;0}  (N = 32)
```

Fig.3.51. Filling-yarn algorithm part.

_Overall View

Now looking back again at different phases of the algorithm, there were five different steps:

1. **Pattern-Data Extraction**
2. **Target Surface Subdivision**
3. **Applying Pattern onto the Target Surface**
4. **Axis-Curve Generation**
5. **Yarn Geometry Generation**

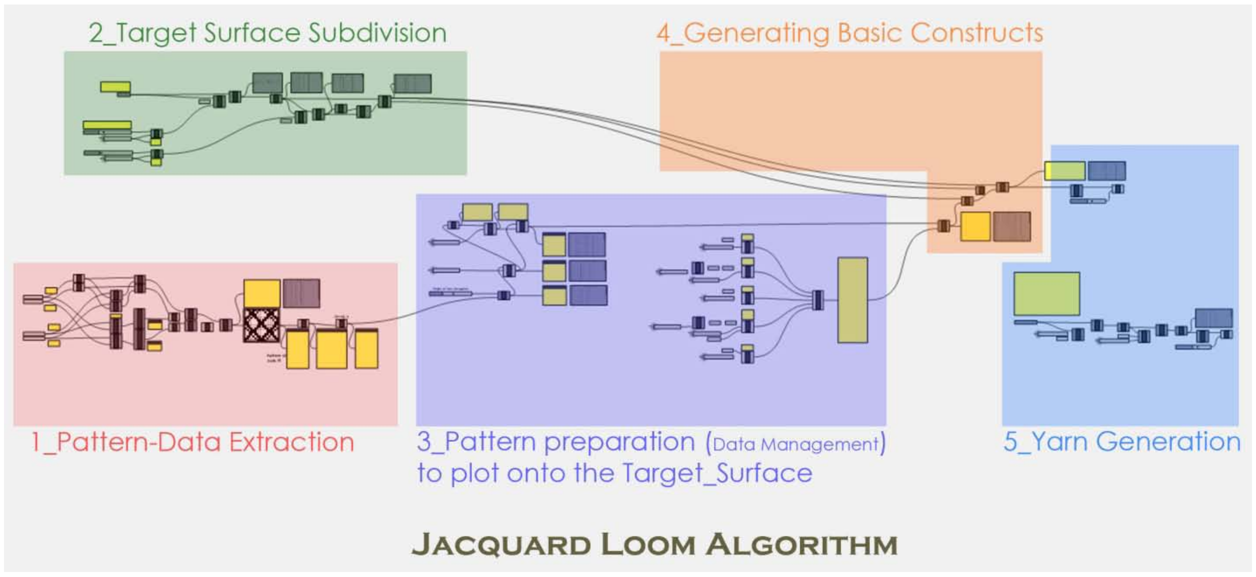
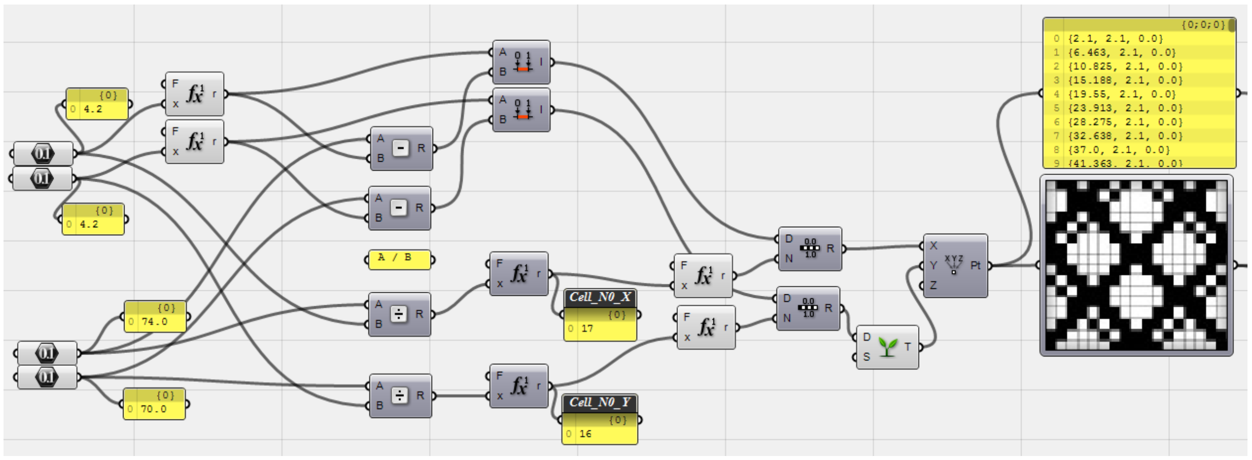
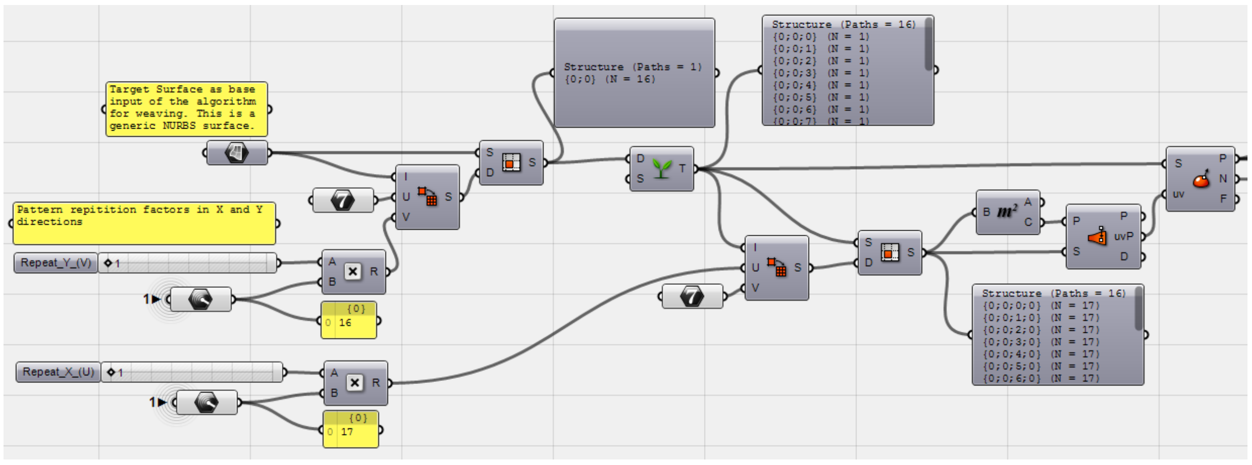

2_Target Surface Subdivision
4_Generating Basic Constructs
1_Pattern-Data Extraction
3_Pattern preparation (Data Management) to plot onto the Target_Surface
5_Yarn Generation
JACQUARD LOOM ALGORITHM

Fig.3.52. Overall view of Jacquard Loom Algorithm.



```
{0;0;0}
0 {2.1, 2.1, 0.0}
1 {6.463, 2.1, 0.0}
2 {10.825, 2.1, 0.0}
3 {15.188, 2.1, 0.0}
4 {19.55, 2.1, 0.0}
5 {23.913, 2.1, 0.0}
6 {28.275, 2.1, 0.0}
7 {32.638, 2.1, 0.0}
8 {37.0, 2.1, 0.0}
9 {41.363, 2.1, 0.0}
```

Fig.3.53. Phase one: Pattern-Data Extraction



```
Target Surface as base
input of the algorithm
for weaving. This is a
generic NURBS surface.
```

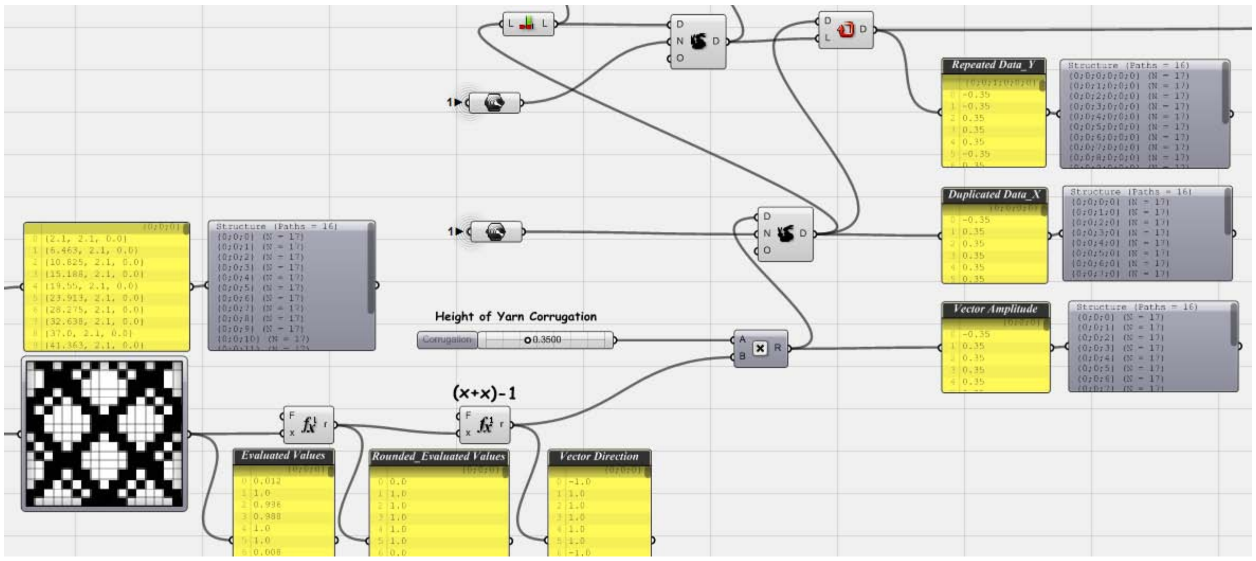
```
Pattern repitition factors in X and Y
directions
```

```
Structure (Paths = 1)
{0;0} (N = 16)
```

```
Structure (Paths = 16)
{0;0;0} (N = 1)
{0;0;1} (N = 1)
{0;0;2} (N = 1)
{0;0;3} (N = 1)
{0;0;4} (N = 1)
{0;0;5} (N = 1)
{0;0;6} (N = 1)
{0;0;7} (N = 1)
```

```
Structure (Paths = 16)
{0;0;0;0} (N = 17)
{0;0;1;0} (N = 17)
{0;0;2;0} (N = 17)
{0;0;3;0} (N = 17)
{0;0;4;0} (N = 17)
{0;0;5;0} (N = 17)
{0;0;6;0} (N = 17)
```

Fig.3.54. Phase two: Target Surface Subdivision



```
{0;0;0}
0 {2.1, 2.1, 0.0}
1 {6.463, 2.1, 0.0}
2 {10.825, 2.1, 0.0}
3 {15.188, 2.1, 0.0}
4 {19.55, 2.1, 0.0}
5 {23.913, 2.1, 0.0}
6 {28.275, 2.1, 0.0}
7 {32.638, 2.1, 0.0}
8 {37.0, 2.1, 0.0}
9 {41.363, 2.1, 0.0}
```

```
Structure (Paths = 16)
{0;0;0} (N = 17)
{0;0;1} (N = 17)
{0;0;2} (N = 17)
{0;0;3} (N = 17)
{0;0;4} (N = 17)
{0;0;5} (N = 17)
{0;0;6} (N = 17)
{0;0;7} (N = 17)
{0;0;8} (N = 17)
{0;0;9} (N = 17)
{0;0;10} (N = 17)
```

```
Repeated Data_Y
{0;0;1;0;0;0}
0 -0.35
1 -0.35
2 0.35
3 0.35
4 0.35
5 -0.35
6 0.35
```

```
Structure (Paths = 16)
{0;0;0;0;0;0} (N = 17)
{0;0;1;0;0;0} (N = 17)
{0;0;2;0;0;0} (N = 17)
{0;0;3;0;0;0} (N = 17)
{0;0;4;0;0;0} (N = 17)
{0;0;5;0;0;0} (N = 17)
{0;0;6;0;0;0} (N = 17)
{0;0;7;0;0;0} (N = 17)
{0;0;8;0;0;0} (N = 17)
```

```
Duplicated Data_X
{0;0;0;0}
0 -0.35
1 0.35
2 0.35
3 0.35
4 0.35
5 0.35
```

```
Structure (Paths = 16)
{0;0;0;0} (N = 17)
{0;0;1;0} (N = 17)
{0;0;2;0} (N = 17)
{0;0;3;0} (N = 17)
{0;0;4;0} (N = 17)
{0;0;5;0} (N = 17)
{0;0;6;0} (N = 17)
{0;0;7;0} (N = 17)
```

```
Vector Amplitude
{0;0;0}
0 -0.35
1 0.35
2 0.35
3 0.35
4 0.35
```

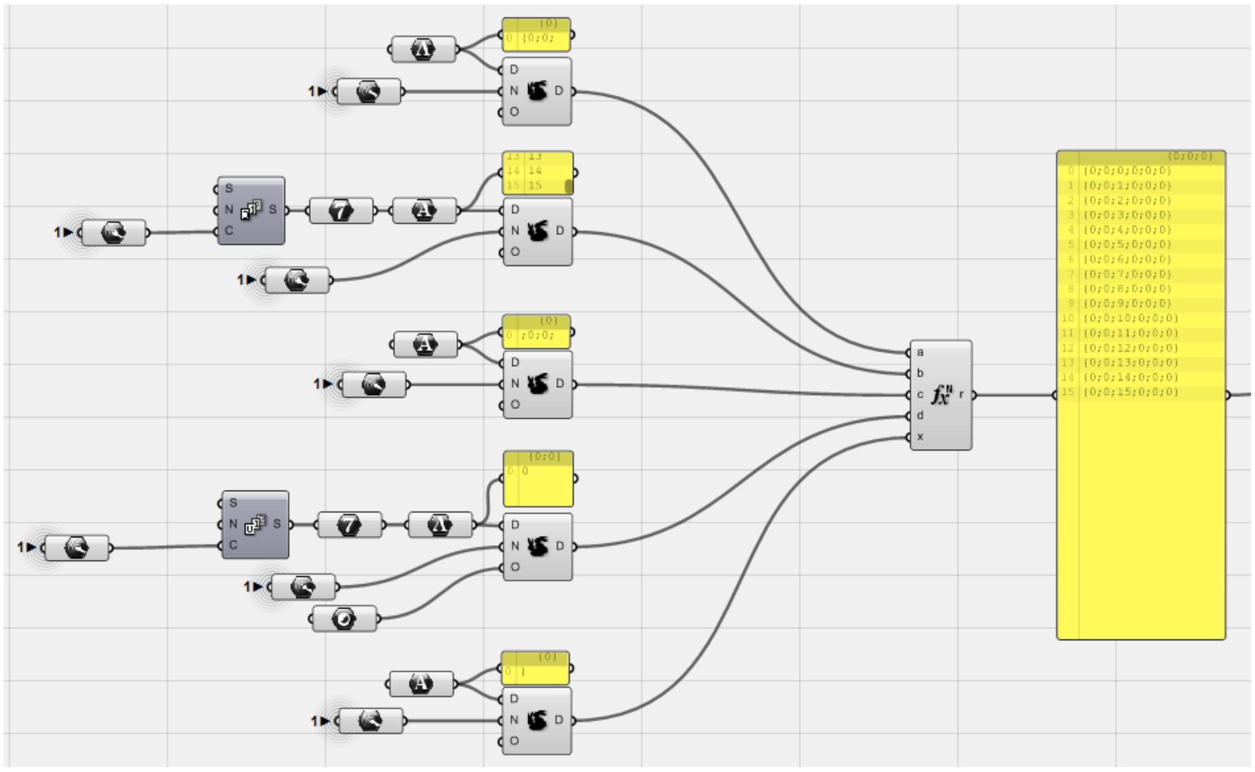
```
Structure (Paths = 16)
{0;0;0} (N = 17)
{0;0;1} (N = 17)
{0;0;2} (N = 17)
{0;0;3} (N = 17)
{0;0;4} (N = 17)
{0;0;5} (N = 17)
{0;0;6} (N = 17)
{0;0;7} (N = 17)
```

```
Evaluated Values
{0;0;0}
0 0.012
1 1.0
2 0.936
3 0.988
4 1.0
5 1.0
6 0.008
```

```
Rounded_Evaluated Values
{0;0;0}
0 0.0
1 1.0
2 1.0
3 1.0
4 1.0
5 1.0
6 0.0
```

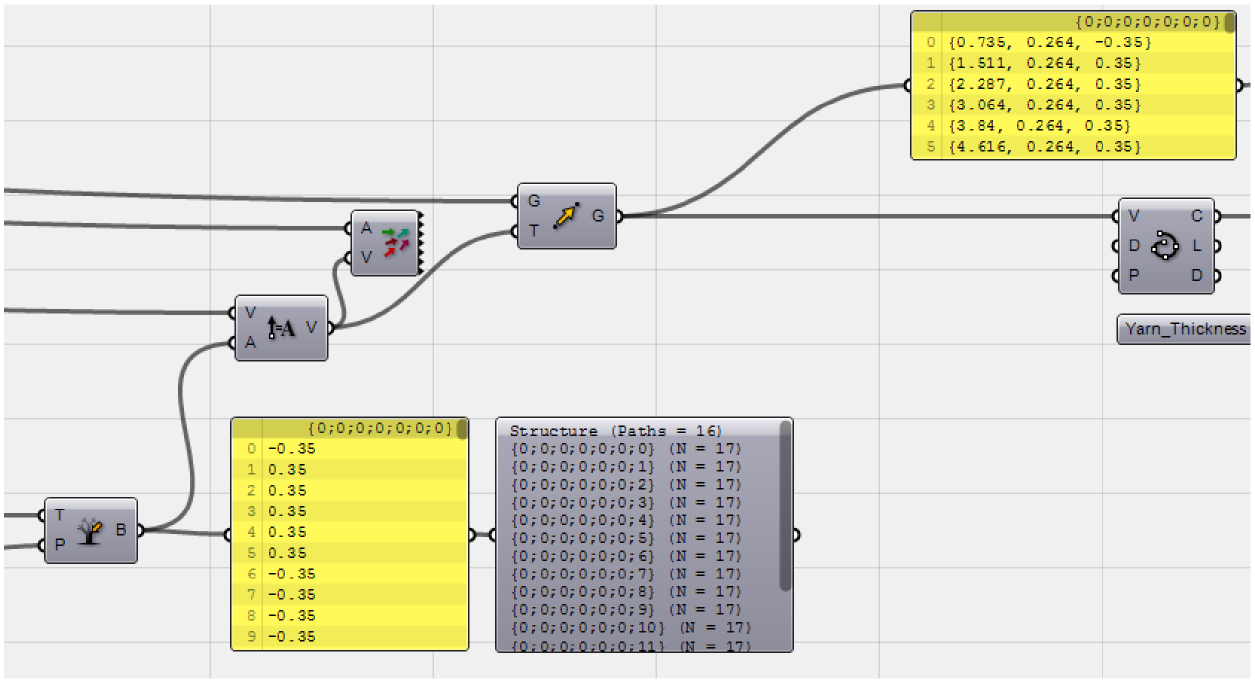
```
Vector Direction
{0;0;0}
0 -1.0
1 1.0
2 1.0
3 1.0
4 1.0
5 1.0
6 -1.0
```

Fig.3.55. Phase three/1: Applying Pattern to Target_Surface / Data preparation



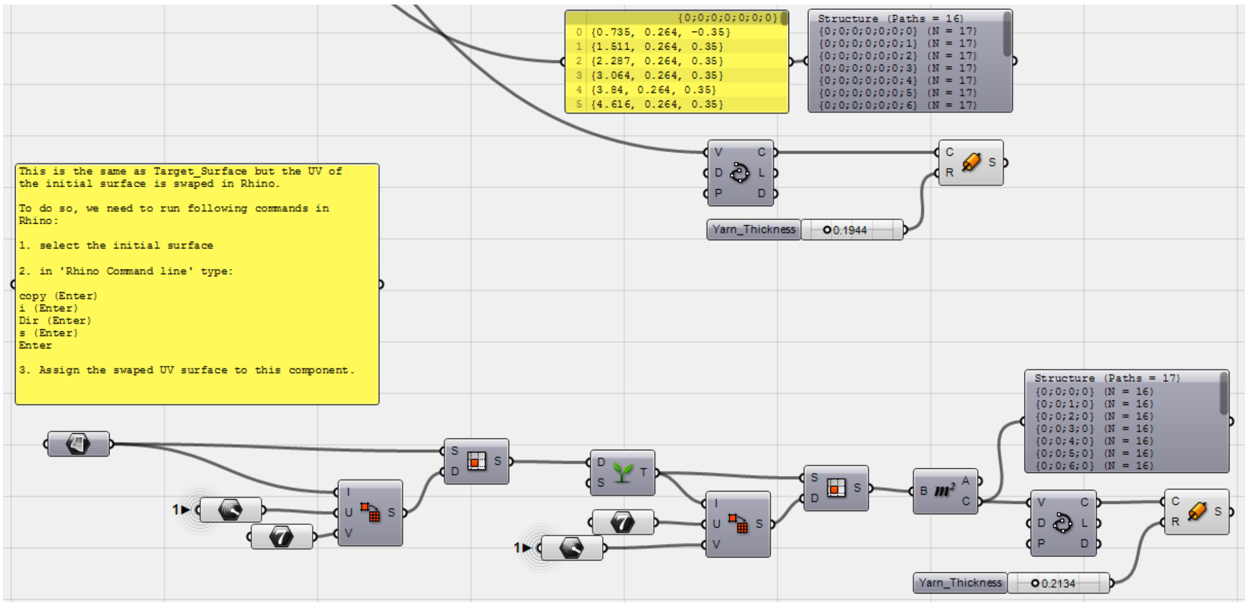
```
{0}
0 {0;0;

13 13
14 14
15 15

{0}
0 ;0;0;

{0;0}
0 0

{0}
0 1

{0;0;0}
0 {0;0;0;0;0;0}
1 {0;0;1;0;0;0}
2 {0;0;2;0;0;0}
3 {0;0;3;0;0;0}
4 {0;0;4;0;0;0}
5 {0;0;5;0;0;0}
6 {0;0;6;0;0;0}
7 {0;0;7;0;0;0}
8 {0;0;8;0;0;0}
9 {0;0;9;0;0;0}
10 {0;0;10;0;0;0}
11 {0;0;11;0;0;0}
12 {0;0;12;0;0;0}
13 {0;0;13;0;0;0}
14 {0;0;14;0;0;0}
15 {0;0;15;0;0;0}
```

Fig.3.56. Phase three/2: Applying Pattern to Target_Surface / Data preparation



```
{0;0;0;0;0;0;0}
0 {0.735, 0.264, -0.35}
1 {1.511, 0.264, 0.35}
2 {2.287, 0.264, 0.35}
3 {3.064, 0.264, 0.35}
4 {3.84, 0.264, 0.35}
5 {4.616, 0.264, 0.35}

{0;0;0;0;0;0;0}
0 -0.35
1 0.35
2 0.35
3 0.35
4 0.35
5 0.35
6 -0.35
7 -0.35
8 -0.35
9 -0.35

Structure (Paths = 16)
{0;0;0;0;0;0;0} (N = 17)
{0;0;0;0;0;0;1} (N = 17)
{0;0;0;0;0;0;2} (N = 17)
{0;0;0;0;0;0;3} (N = 17)
{0;0;0;0;0;0;4} (N = 17)
{0;0;0;0;0;0;5} (N = 17)
{0;0;0;0;0;0;6} (N = 17)
{0;0;0;0;0;0;7} (N = 17)
{0;0;0;0;0;0;8} (N = 17)
{0;0;0;0;0;0;9} (N = 17)
{0;0;0;0;0;0;10} (N = 17)
{0;0;0;0;0;0;11} (N = 17)
```

Fig.3.57. Phase four: Basic Constructs _ Axis Curve Generation



```
{0;0;0;0;0;0;0}
0 {0.735, 0.264, -0.35}
1 {1.511, 0.264, 0.35}
2 {2.287, 0.264, 0.35}
3 {3.064, 0.264, 0.35}
4 {3.84, 0.264, 0.35}
5 {4.616, 0.264, 0.35}
```

```
Structure (Paths = 16)
{0;0;0;0;0;0;0} (N = 17)
{0;0;0;0;0;0;1} (N = 17)
{0;0;0;0;0;0;2} (N = 17)
{0;0;0;0;0;0;3} (N = 17)
{0;0;0;0;0;0;4} (N = 17)
{0;0;0;0;0;0;5} (N = 17)
{0;0;0;0;0;0;6} (N = 17)
```

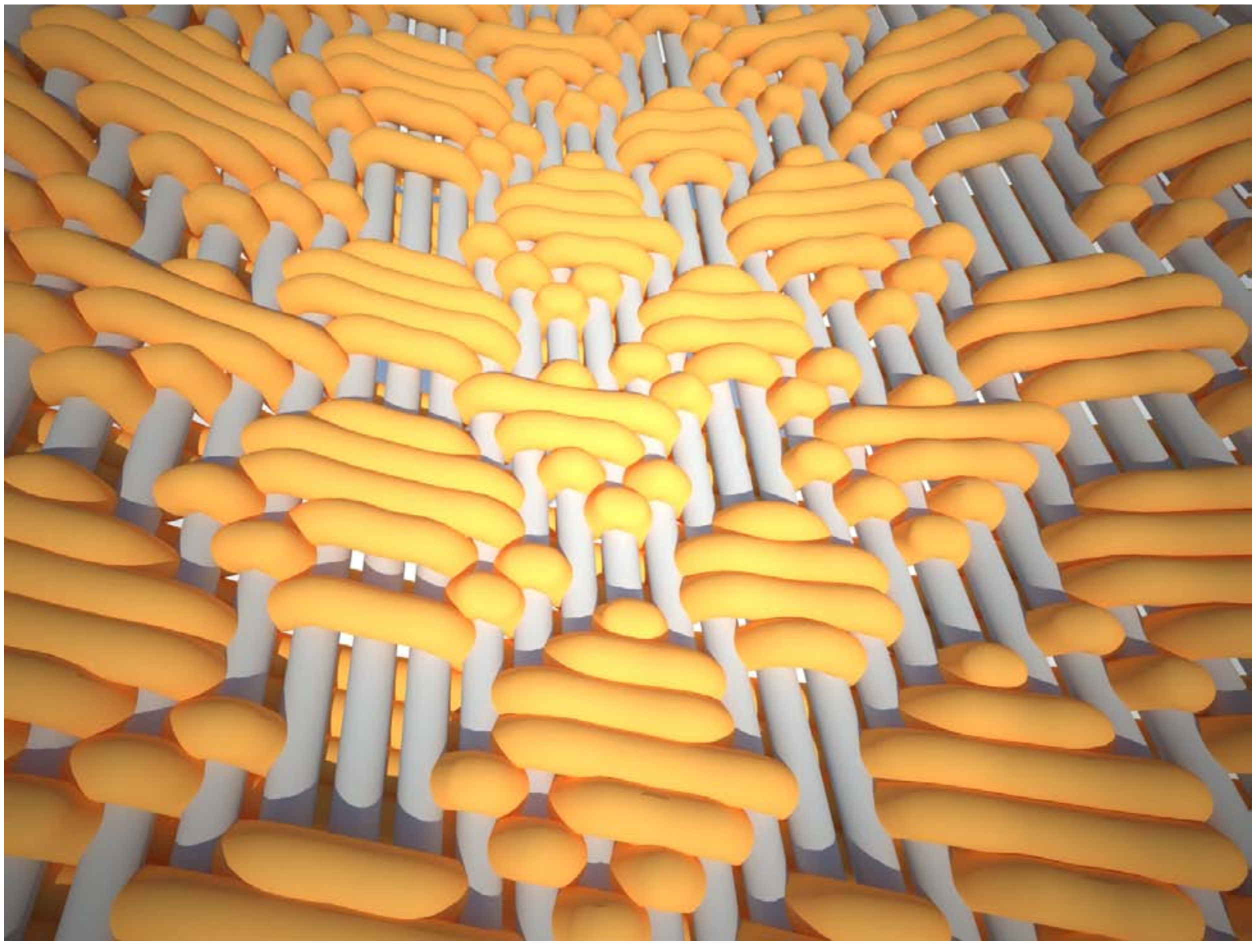
```
This is the same as Target_Surface but the UV of
the initial surface is swaped in Rhino.

To do so, we need to run following commands in
Rhino:

1. select the initial surface

2. in 'Rhino Command line' type:

copy (Enter)
i (Enter)
Dir (Enter)
s (Enter)
Enter

3. Assign the swaped UV surface to this component.
```

Yarn_Thickness 0.1944

```
Structure (Paths = 17)
{0;0;0;0} (N = 16)
{0;0;1;0} (N = 16)
{0;0;2;0} (N = 16)
{0;0;3;0} (N = 16)
{0;0;4;0} (N = 16)
{0;0;5;0} (N = 16)
{0;0;6;0} (N = 16)
```

Yarn_Thickness 0.2134

Fig.3.58. Phase five: Yarn Generation + Filling yarns generation



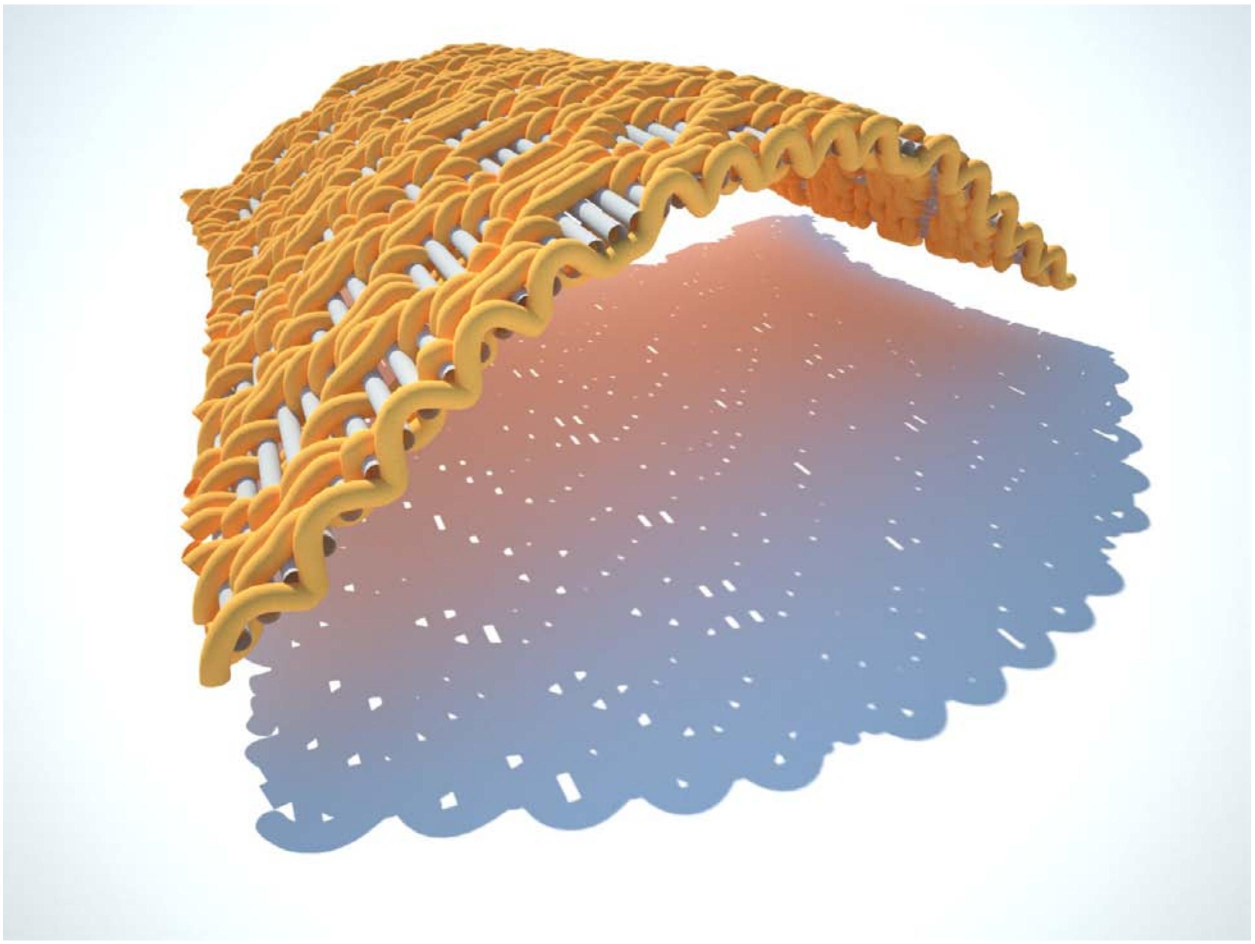


Fig.3.59. Final production of one possible pattern in Jacquard Loom Algorithm

<u>Notice:</u>

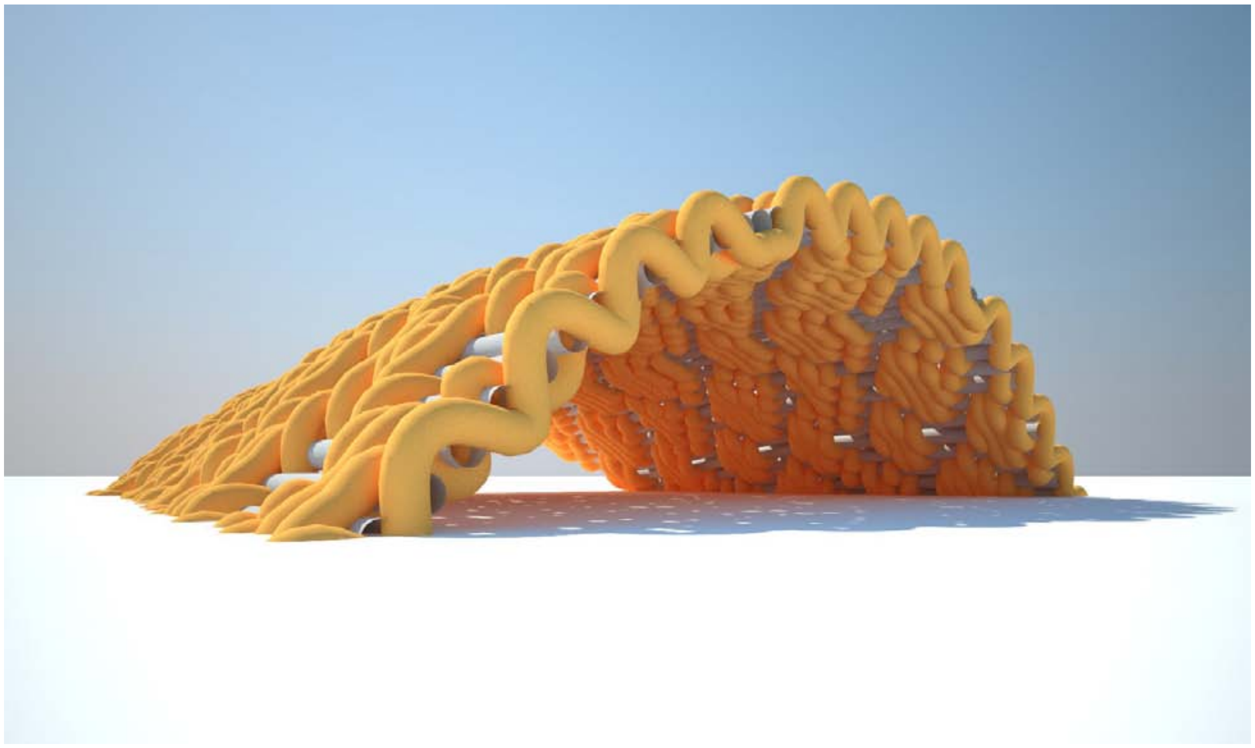
Yarn intersection problem: looking closer at final product of the algorithm, there are some warps and wefts which intersect each other. To avoid this intersection two solutions are available: first of all it is possible to change the <height of corrugation> and <yarn thickness> to control the situation. It should result in more slender yarns and increased space between them. It is also possible to control the situation by changing the repetition factor in relation to the target_surface scale to increase the distance between yarns. Second solution is to design filling yarns as corrugated yarns like the first loom algorithm.

4_Conclusion

The approach of 'Generative Algorithms, Concepts and Experiments' is not to search for any architectural or even generally design application. The term 'Generative' here emphasizes on the target of these studies; algorithms and concepts are aimed to be generative in a way to implement into larger design projects, to be used as logic for further development of idea, method or tool in order to pursue a real, more specific project. So the final product of such approach would be, and actually was, a prototype, a system which is capable of further development in order to promote for different performances; A system that can be adaptable for future design applications. An Algorithmic Solution.

The benefit of such system development is to bring new concepts into design field and broaden the catalogue of available logics and tools. Contemporary architecture enriches itself by focus on small scale systems which have the potential to proliferate in larger spans. Looking at biological systems, finding the mathematical logic of cellular solids, discovering the potentials of natural patterns, or other data extraction from natural or artificial systems are becoming fundamental processes for architects in their design practice.

'Generative Algorithms, Concepts and Experiments' is aimed to look for such concepts and search for design systems and tools. Techniques and methods of this search which is undergoing by all current designers are believed to make systemic change in future architecture.



_Notes

1. Patrick Shoumacker: '**Paraemtricism as a style, Parametricist Manifesto',** http://www.patrikschumacher.com/, 2008

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

3. As an emergent property: the result is more than the sum of individuals.

4. Susan Wylly: 'The Art and History of Weaving', Georgia College & State University, 2001

5. http://www.merriam-webster.com/dictionary/loom

6. Info from Britannica, the digital encyclopedia

7. Virginia Tech University; The Digital Textile Research Group

8. http://www.1911encyclopedia.org/Yarn

9. http://www.britannica.com/

10. http://textileglossary.com/

11. http://www.britannica.com/EBchecked/topic/299155/Jacquard-loom

12. http://en.wikipedia.org/wiki/Jacquard

13. Image source: http://en.wikipedia.org/

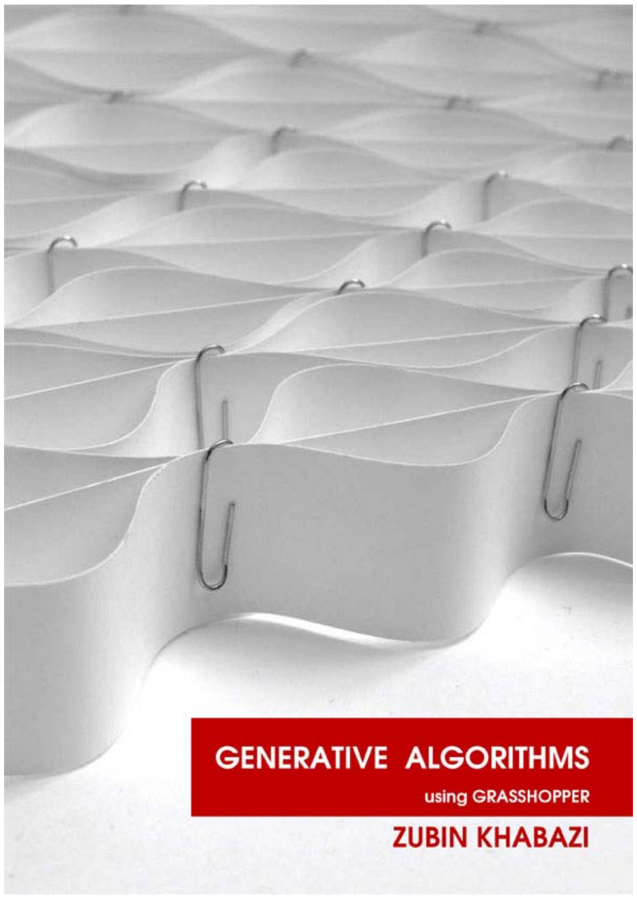
_General References:

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

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

Zubin M Khabazi: 'Generative Algorithms (Using Grasshopper)', On-line publication by www.Grasshopper3d.com, 2010.

_Notes



Generative Algorithms (Using Grasshopper) / Zubin Khabazi

Previously published on-line by www.Grasshopper3d.com, 'Generative Algorithms' as a mixed tutorial on geometry and grasshopper was the first of these design experiments series which focused on basic topics. These topics gathered together various fields of Generative Algorithms and Parametric Geometries include:

1. Generative Algorithms
2. The Very Beginning
3. Data sets and Math
4. Transformations
5. Parametric Space
6. Deformations and Morphing
7. NURBS Surfaces and Meshes
8. Fabrication
9. Design Strategy

The book comprised of design experiments in order to examine subjects and concepts in a practical way and would be useful for beginners.

Generative Algorithms

Concepts and Experiments

1 _ Weaving

By

Zubin Khabazi

www.MORPHOGENESISM.com

zubin.khabazi@googlemail.com