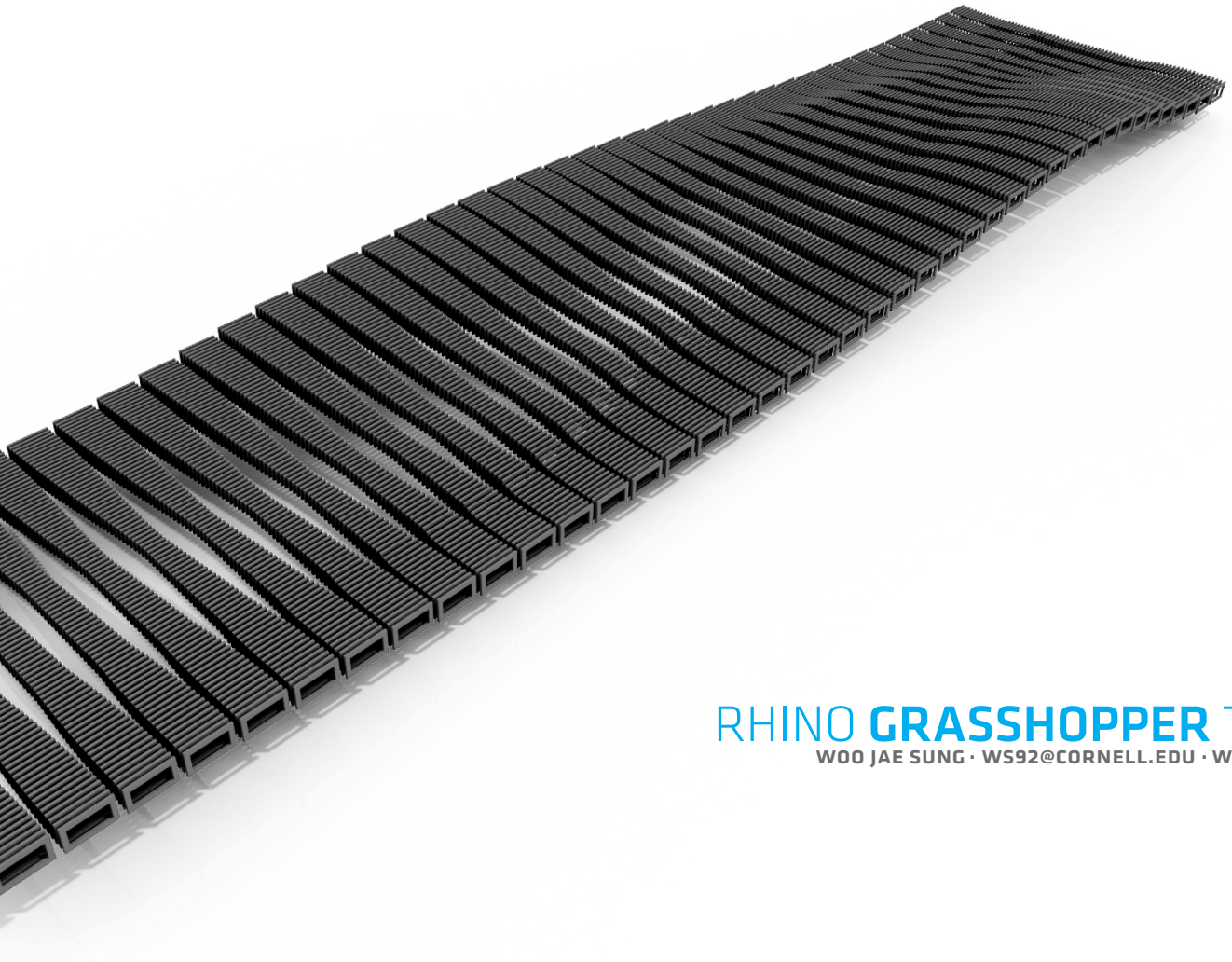


RHINO **GRASSHOPPER** TUTORIAL

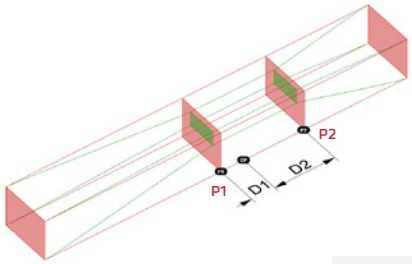
WOO JAE SUNG · WS92@CORNELL.EDU · WWW.WOOJSUNG.COM

OVERVIEW

- Main goal of this project is to make successive bridge-like structures based on parametric idea.

+ **The first step is to make a single component that reacts to the user inputs.**

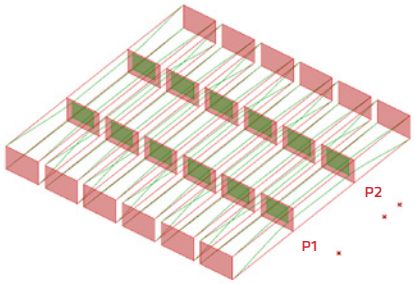
  The component is a box shape [red]. Single user input [p1 & p2] deforms its shape [green].



+ **step 1**

+ **The second step is to replicate the single component to get multiple components which behave under the same logic.**

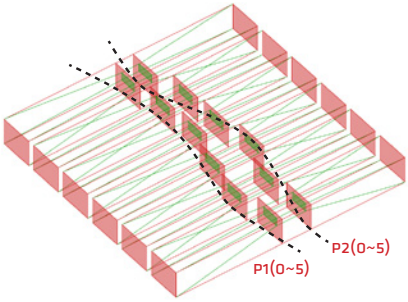
  Single user input [p1 & p2] controls multiple components.



+ **step 2**

+ **The third step is to make a graph variable which controls multiple point inputs.**

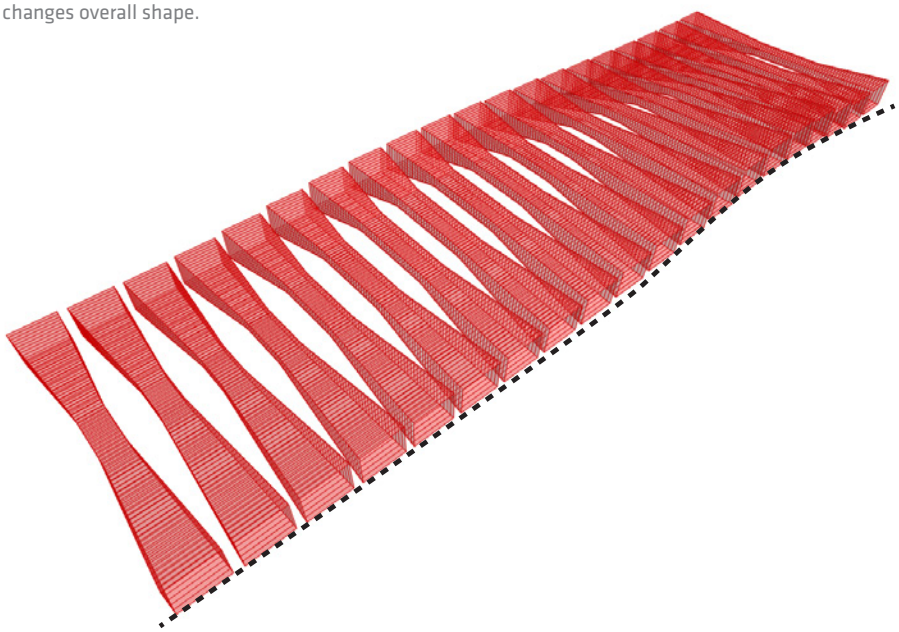
  Single user input [graph/curve] generate multiple input points [p1(0)~p1(5), p2(0)~p2(5)]. Each input controls corresponding component.



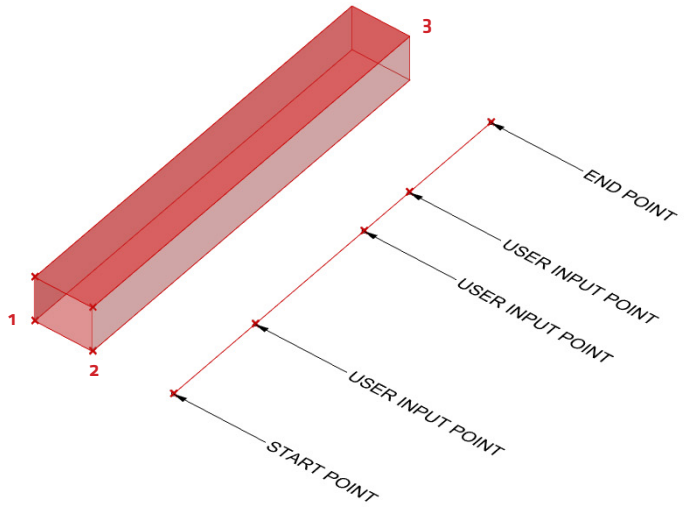
+ **step 3**

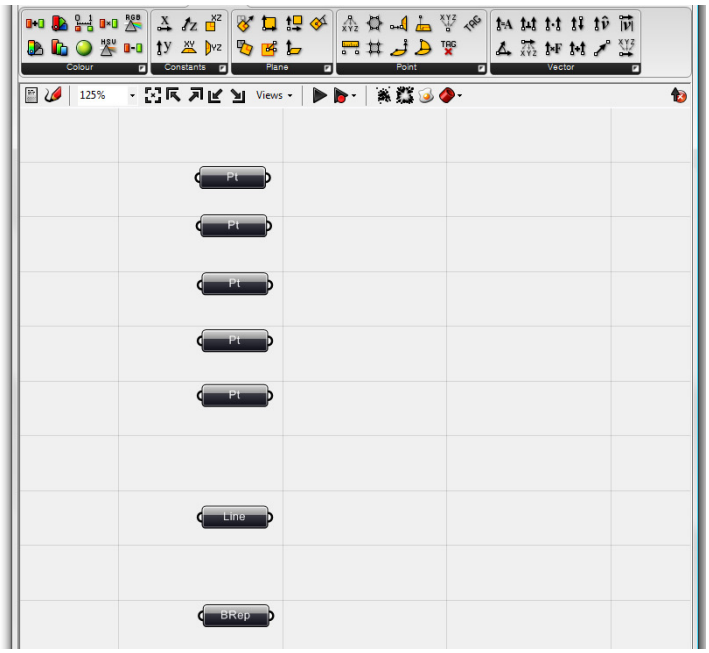
+ **The last step is the application of the component system to the specific site condition.**

  Specific site condition changes overall shape.

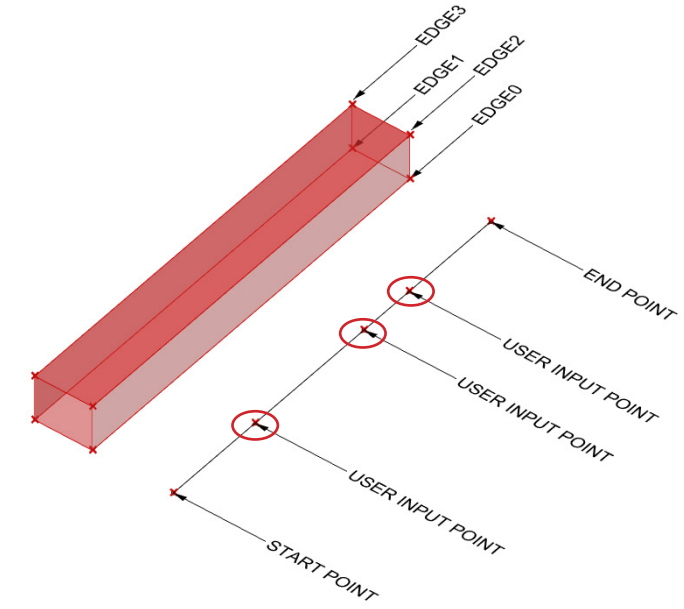


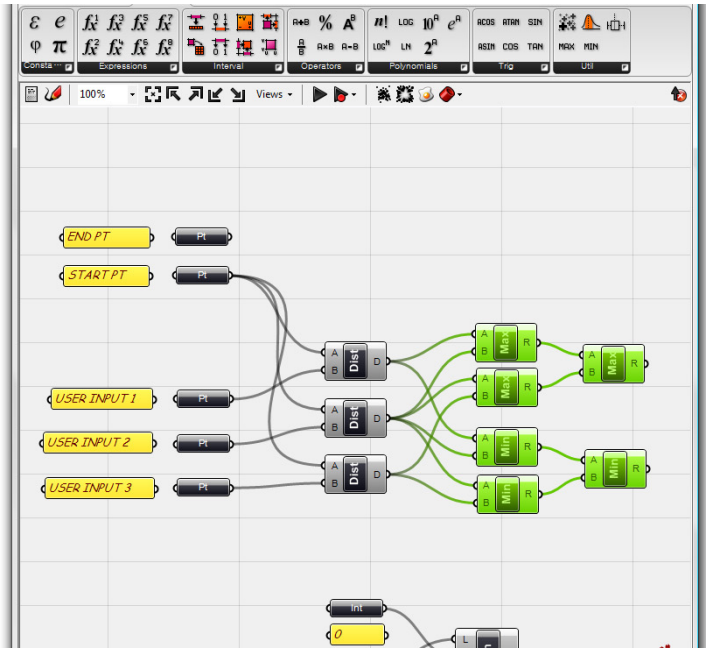
+ **step 4**





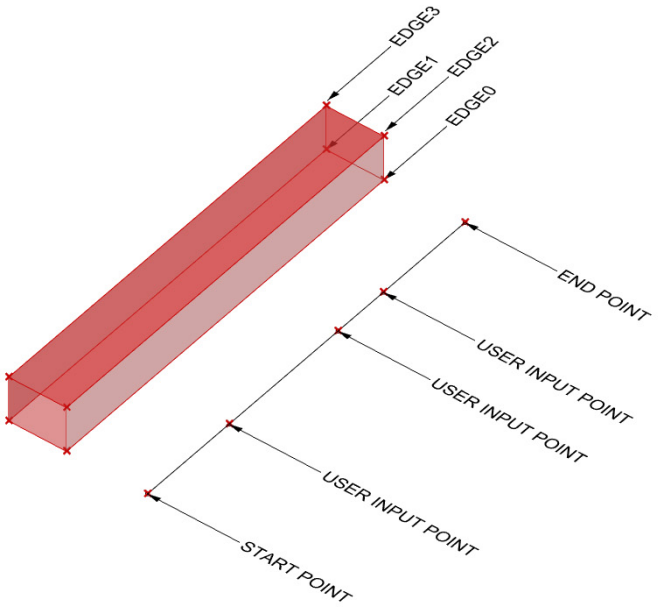
+ **Draw a 3 pt-box, a line and five points on rhino.**
  - Pickup three points with the same order like above. The order is important in grasshopper, because of element id's. Reverse point picking will make a flipped box even if they look same.
+ **In grasshopper canvas, put five box objects, one line object, and one brep object.**
+ **Connect grasshopper and rhino objects.**

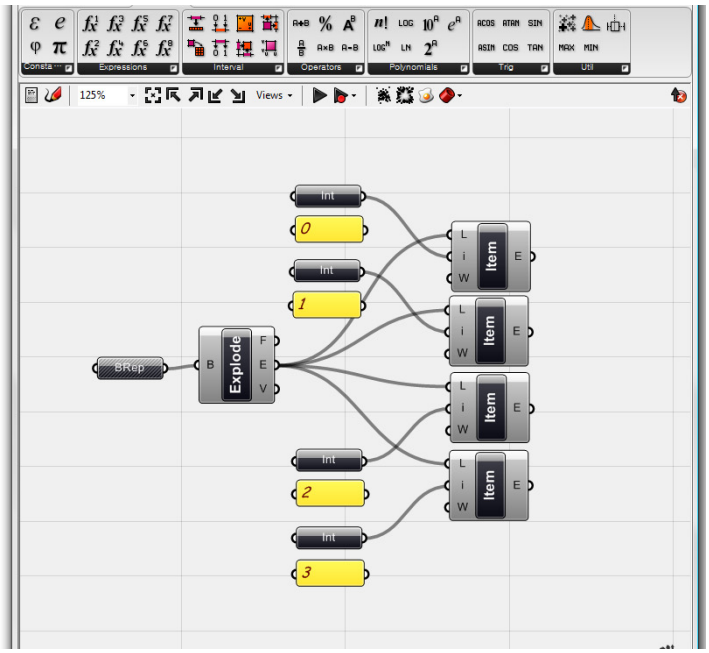




+ **Get the distances from start point to 3 user input points.**
+ **Get the Max. and min. value out of 3 distance values.**

  We need only Max. and min. values. Remove mid one by comparing each other.

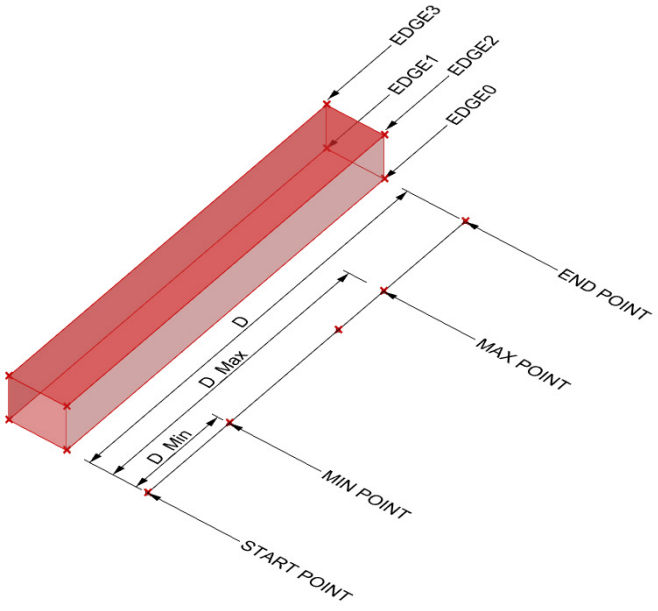


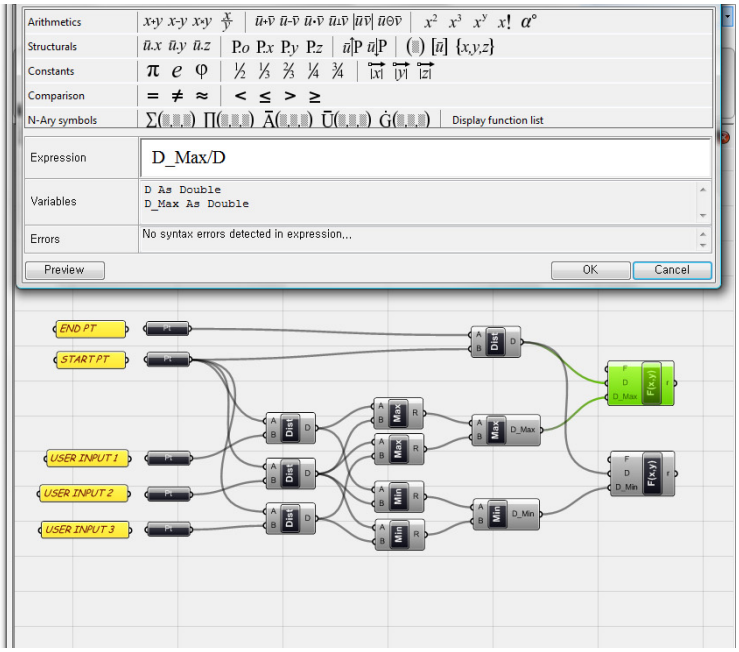


+ **Explode box.**
+ **Retrieve 1st, 3rd, 5th, and 7th edges using list item object.**

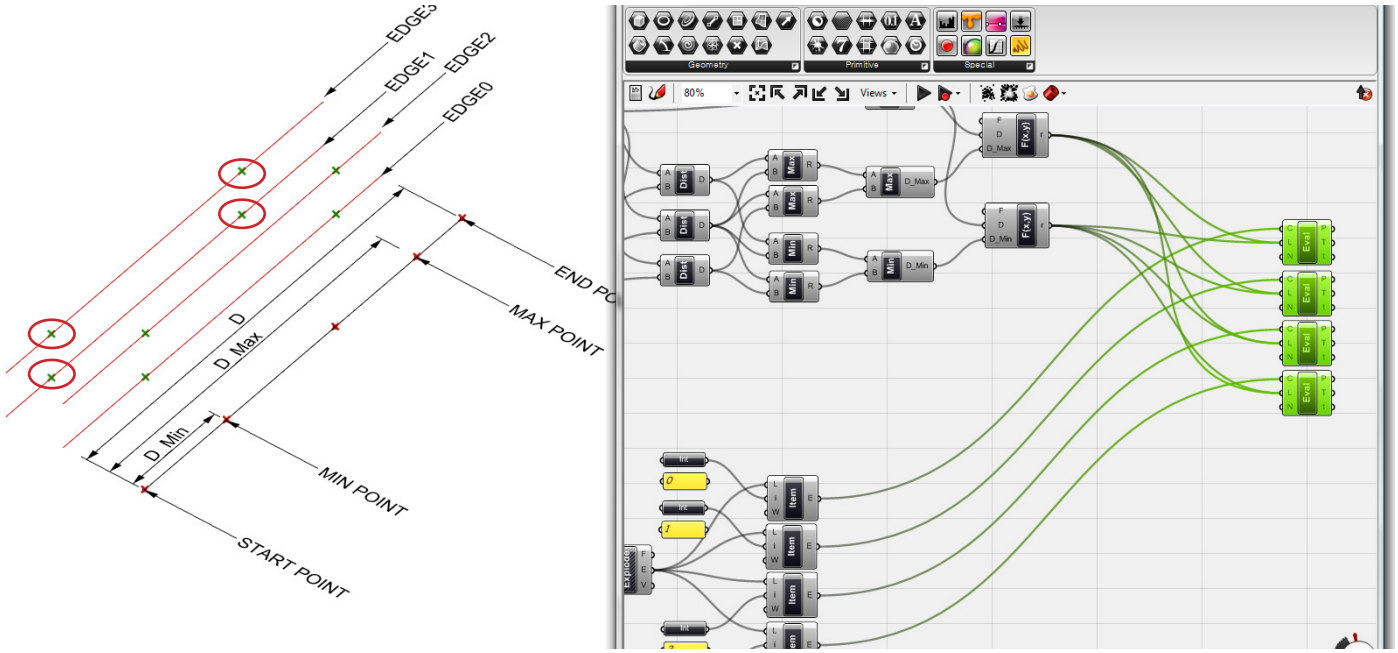
  List item object enables us to pickup an item(items) of specific index.

  Note that in the item tab of list item object, 0 means the 1st item.





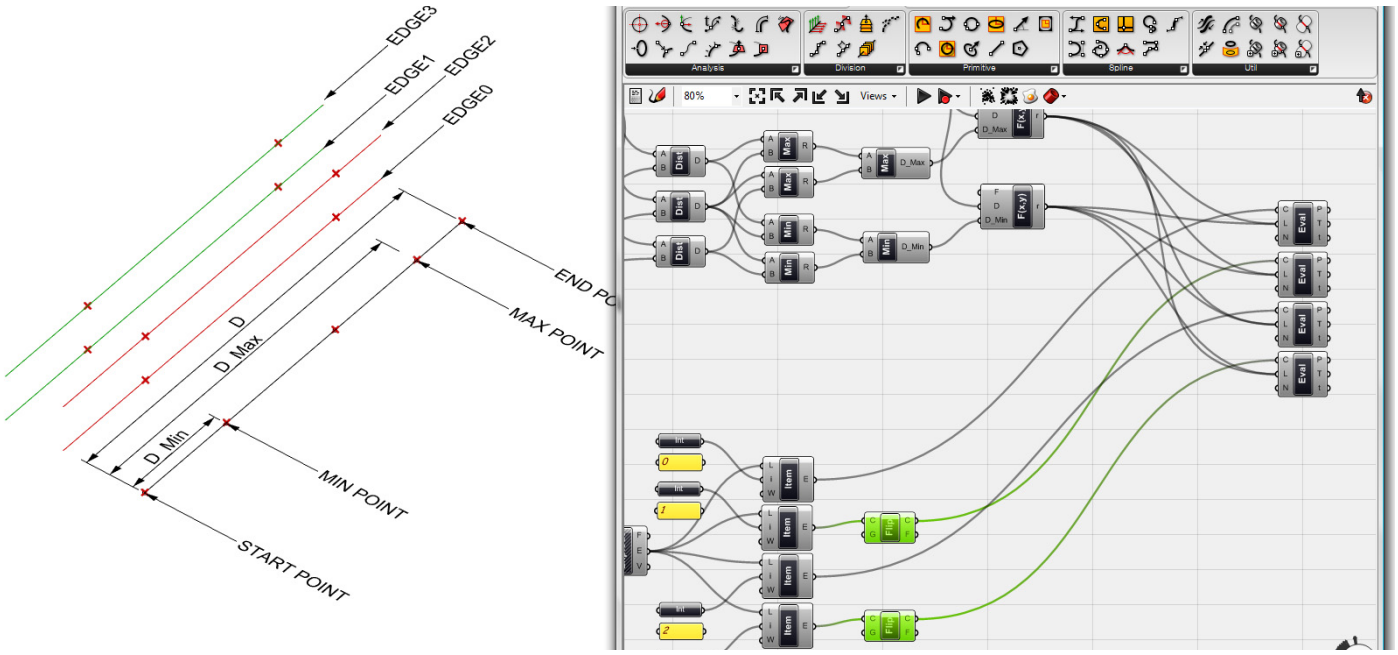
+ **Calculate the ratio of D_max and D as well as D_min and D, using 2-var-function object.**



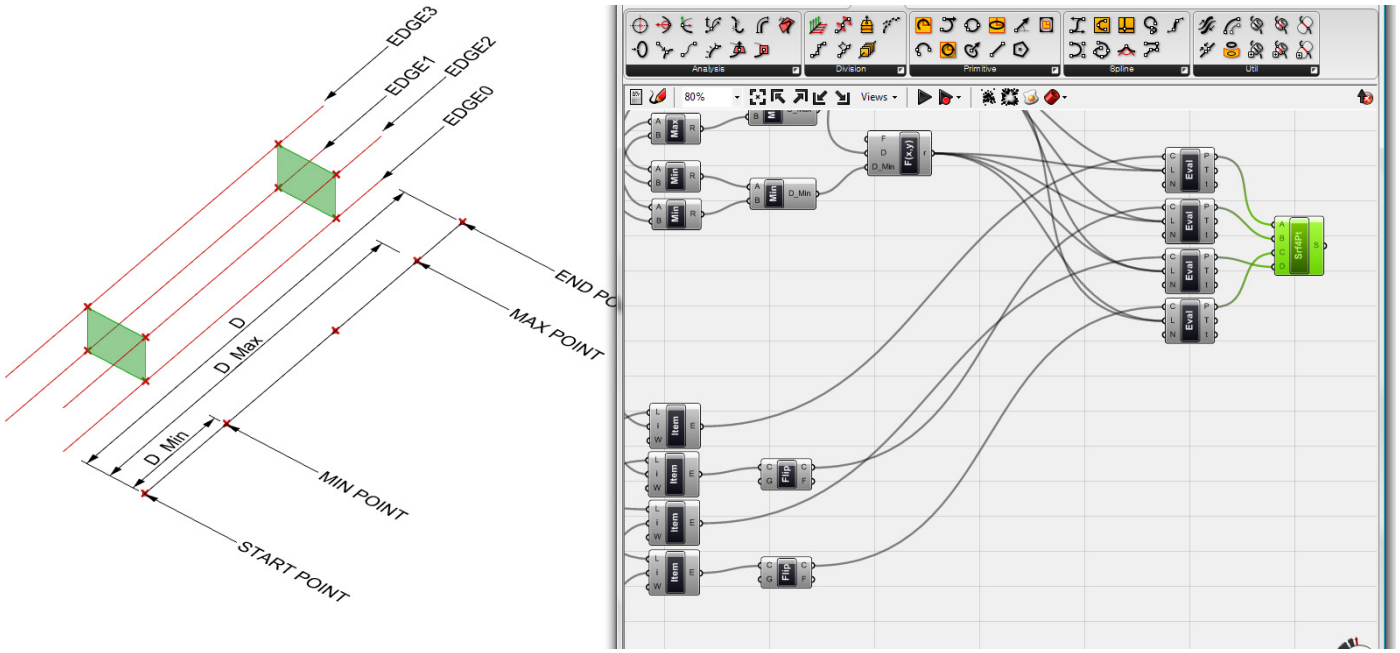
+ **Evaluate 4 edges with the distance ratio.**

  Find two corresponding max and min points on each edge. Edge1 and Edge3 should be flipped.

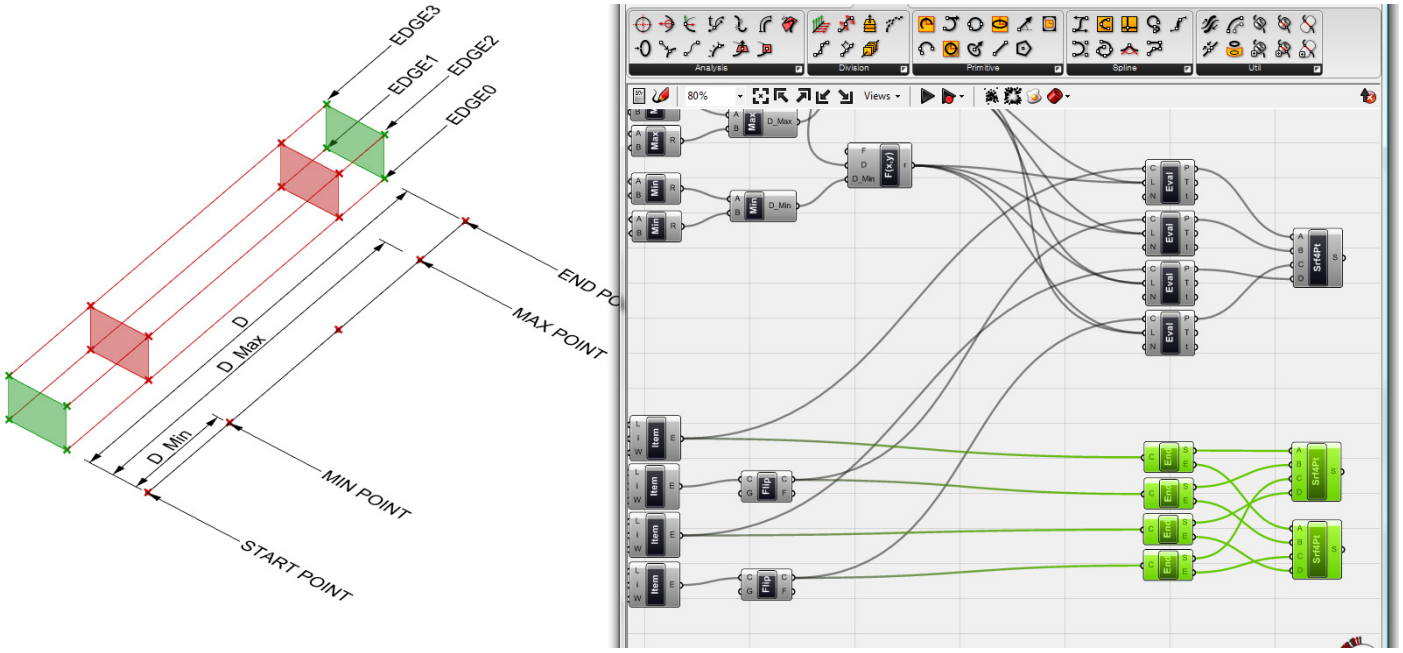
  - Note that D_Max and D_min ratio values are merged into the length factor of Eval objects. This is okay for now since we have only one component. However, it will be a problem as the number of components increase. To avoid that problem, using two sets of independent Eval objects group for each ratio value(D_Max, D_min) would be a better choice. For now, just leave it as it is.



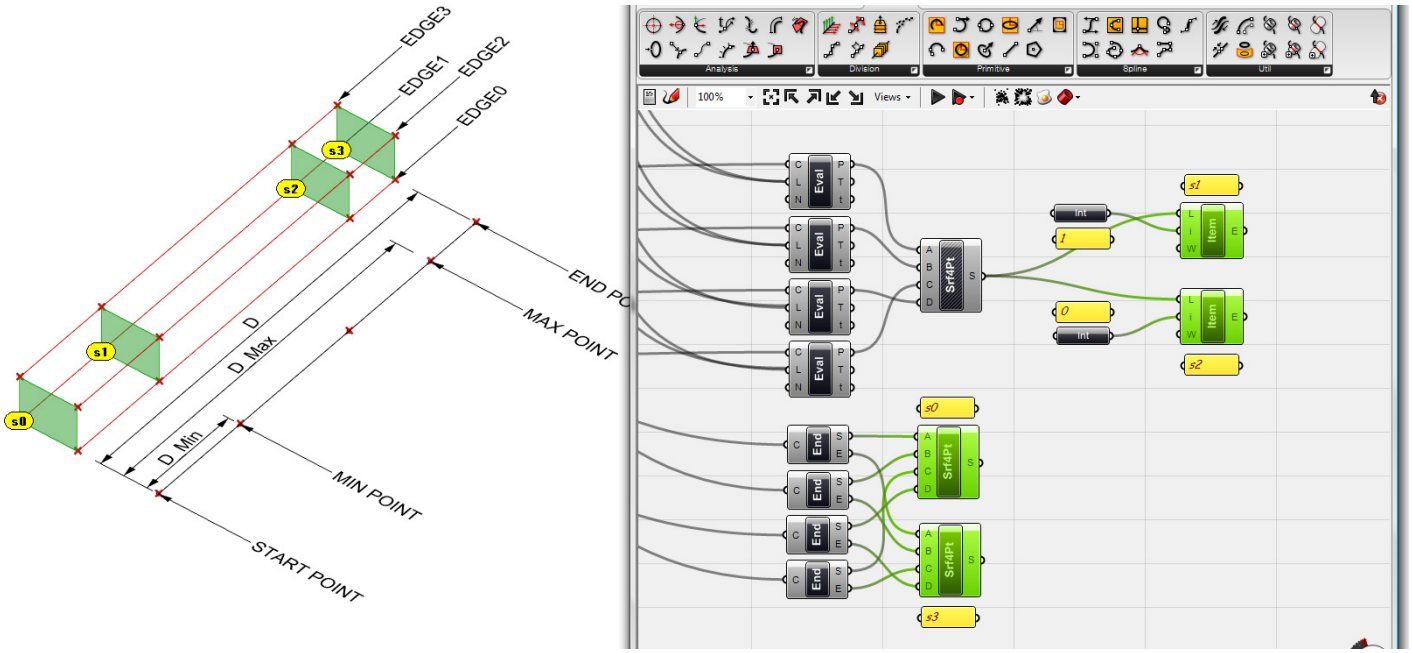
+ **Flip edge1 and 3.**



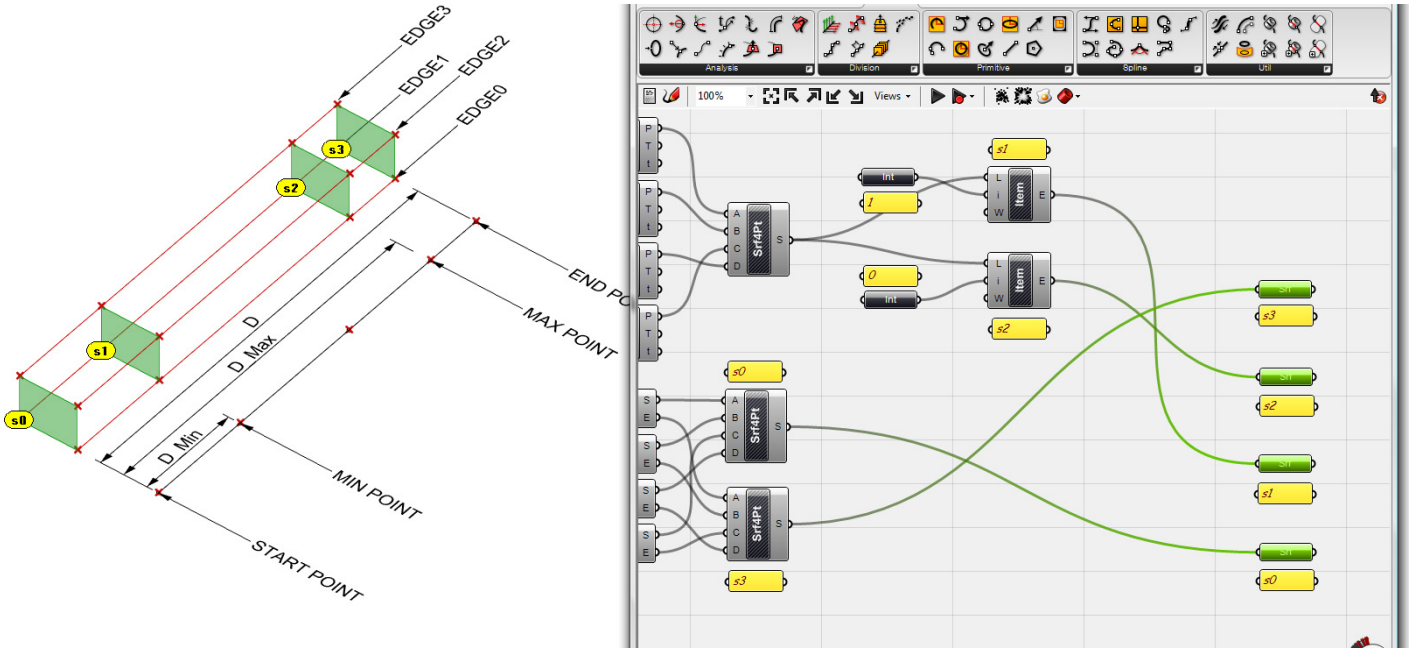
+ **Make two surfaces at the max and min positions.**



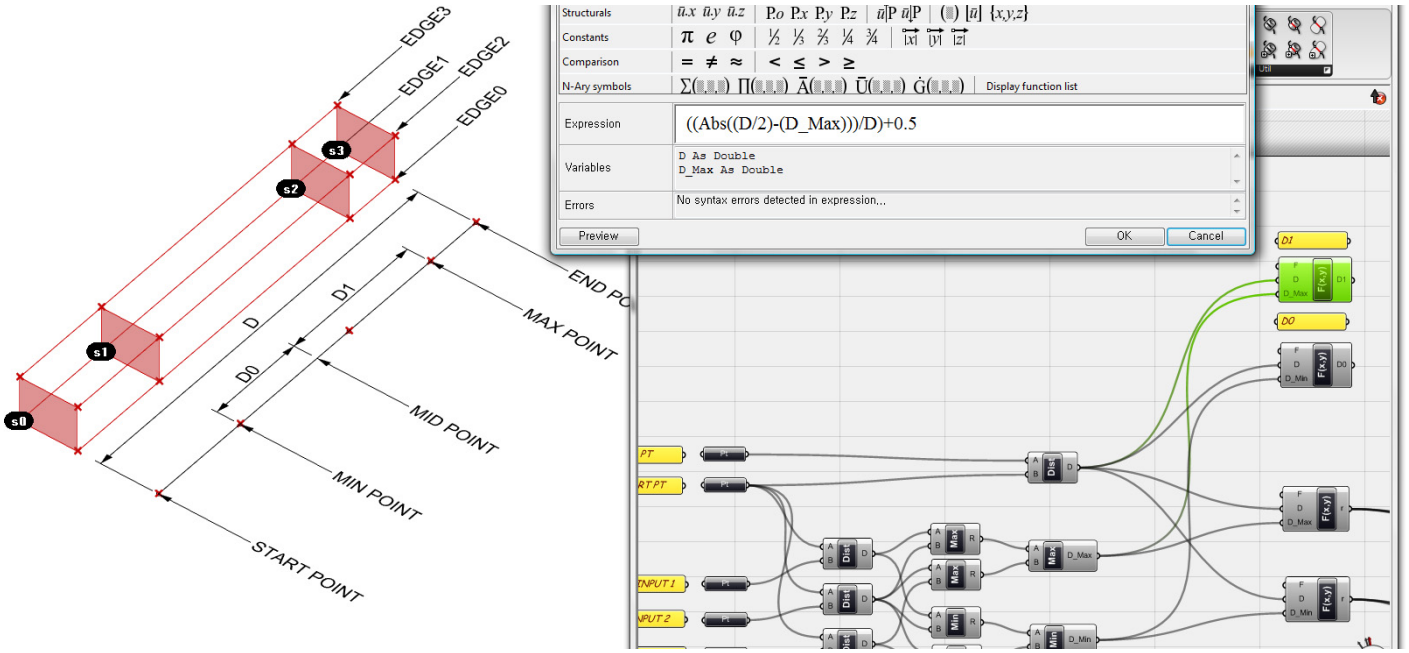
+ **Make two surfaces at the start and end positions.**



+ **Make srf1 and srf2 independent to each other.**
  - If you made two separate Eval object groups for both D_max and D_min ratio value at page step01_03, you don't need this.

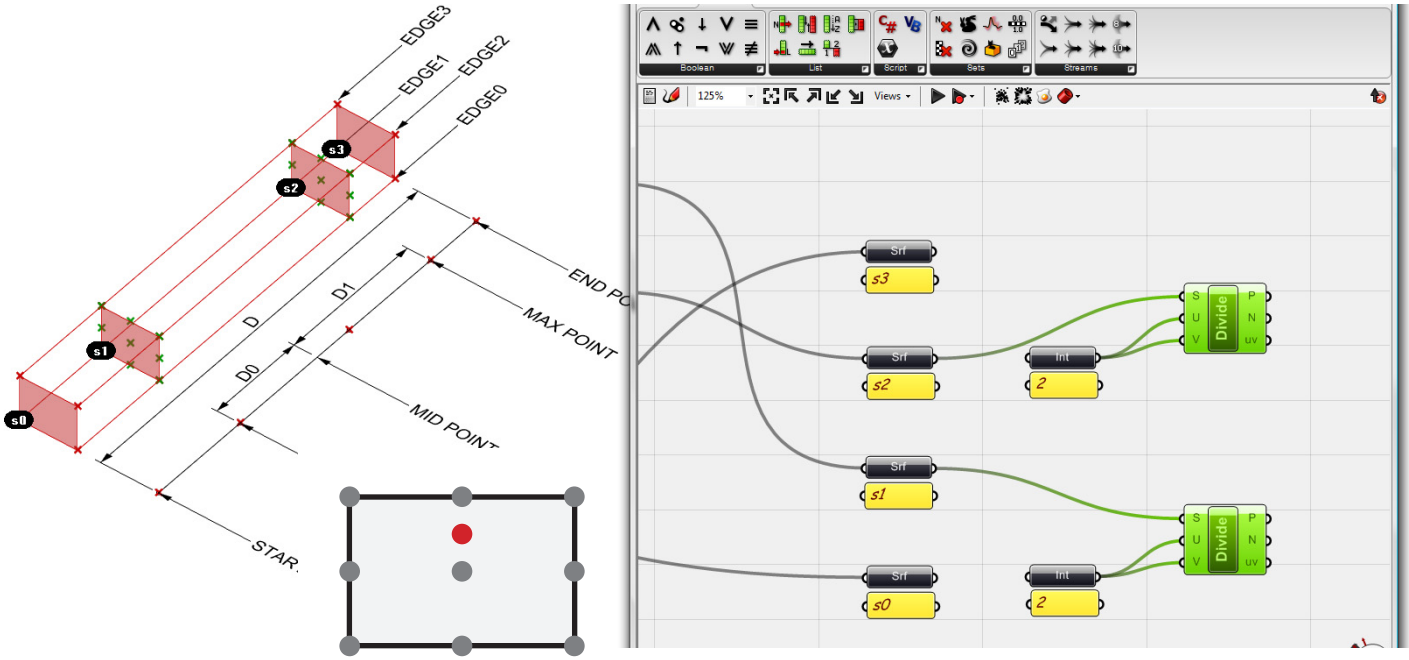


+ **Extract surfaces using Srf parameter for better presentation.**



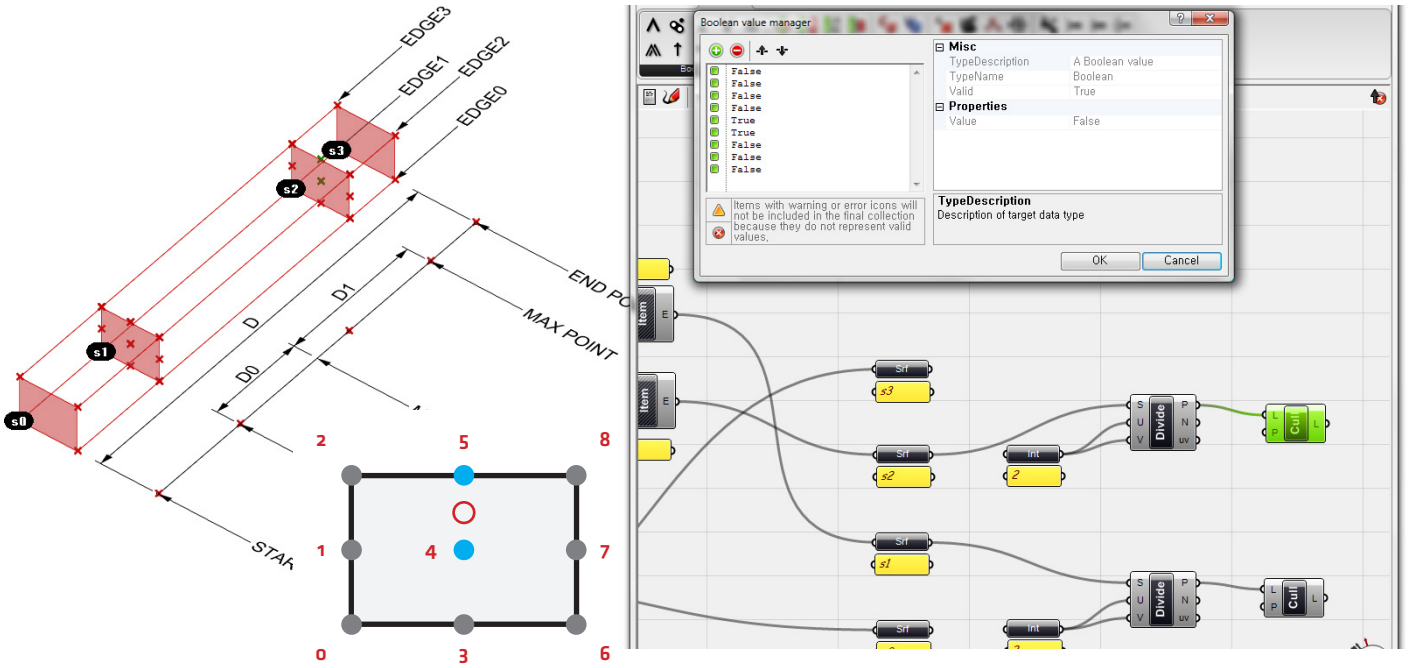
+ **Define scaling factor for S1 and S2.**

  We want to scale down S1 and S2 based on their distances from the mid point. Make two scaling factors with 2-var-function objects. S1 and S2 will be half size at the mid point position, and will be original size at start and end point position.



+ **To define base point for surface scaling, divide S1 and S2 using Divide Surface objects. Set U and V value as 2 and 2.**

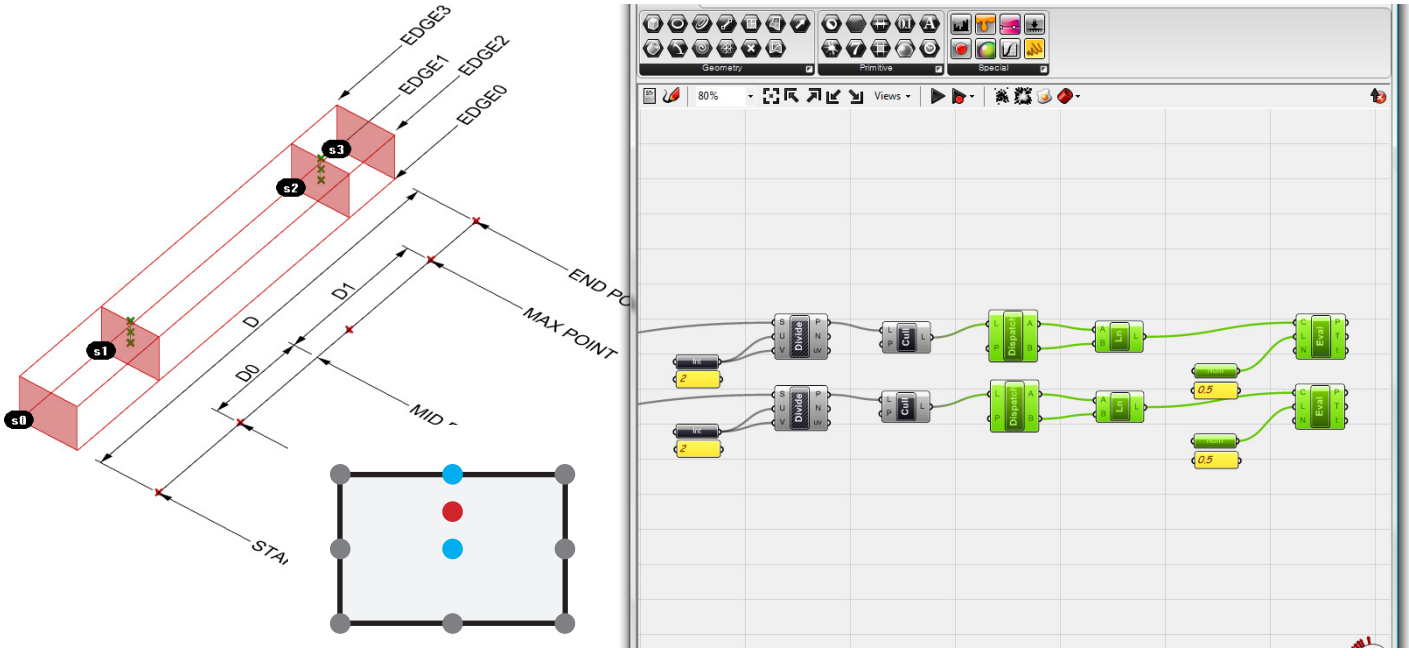
  Red point will be scaling base point.



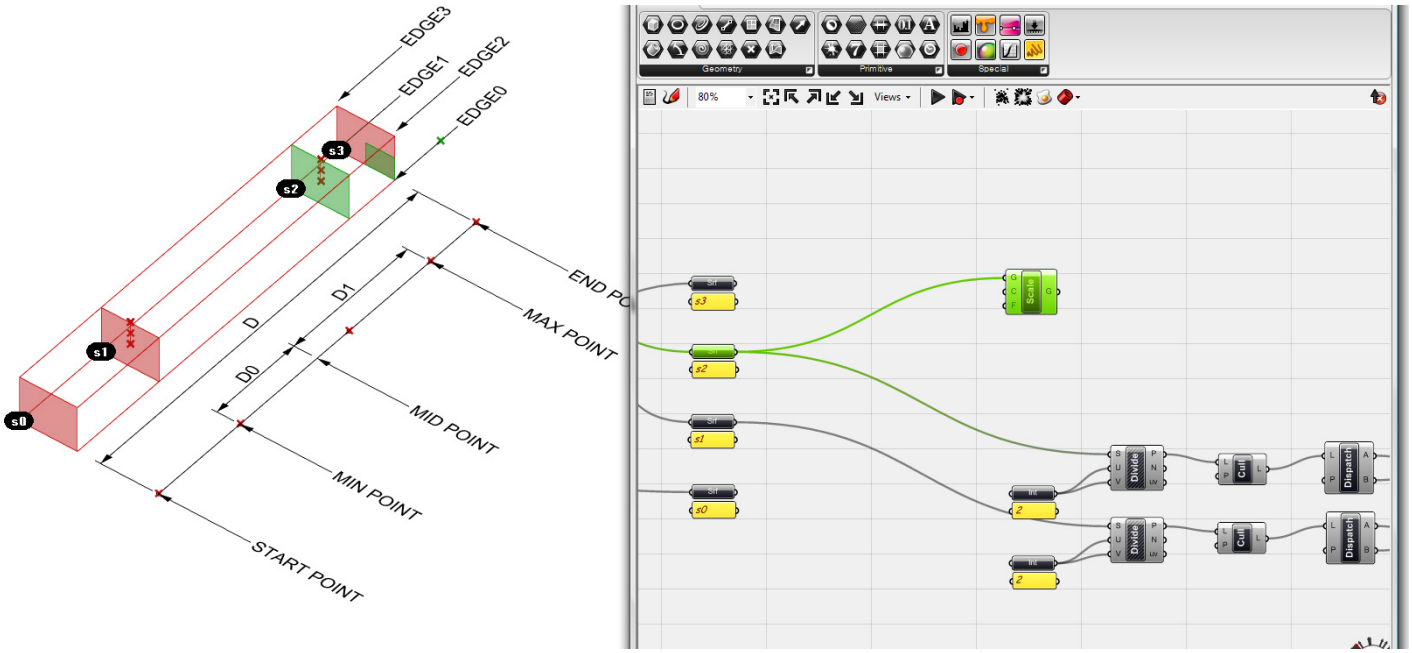
+ **Extract top mid and center points using Cull Pattern object. Set cull pattern "F-F-F-F-T-T-F-F-F"**

  You can also use List Item object. However, considering that eventually we will have more than one components, Cull Pattern will be a better choice.

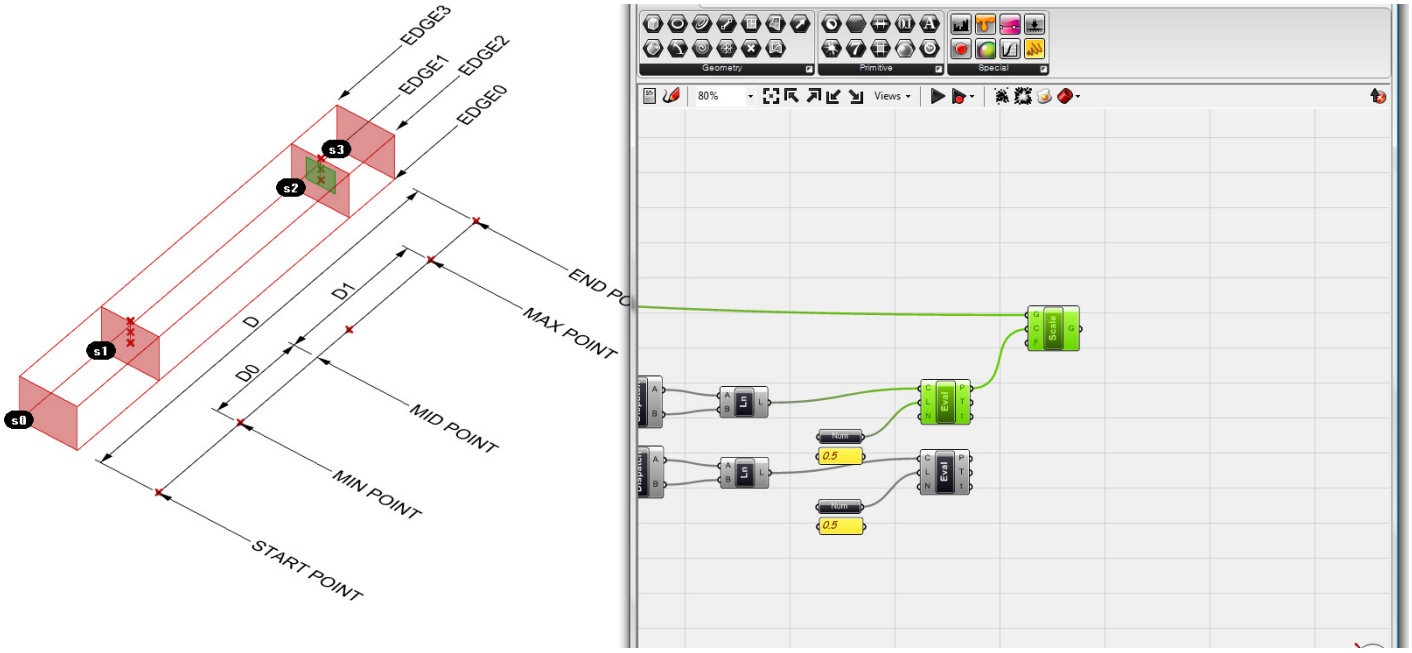
  - Cull Pattern removes specific item(s) from input list based on pattern mask. True parameter prevents an item from being removed from the input list by masking it. On the contrary, List Item object picks up specific item(s) from input list. List Item object can play exactly the same role as long as we know the length of input item. Because of pattern masking, Cull Pattern object is very useful when handling input data which have unlimited or flexible upper bound.



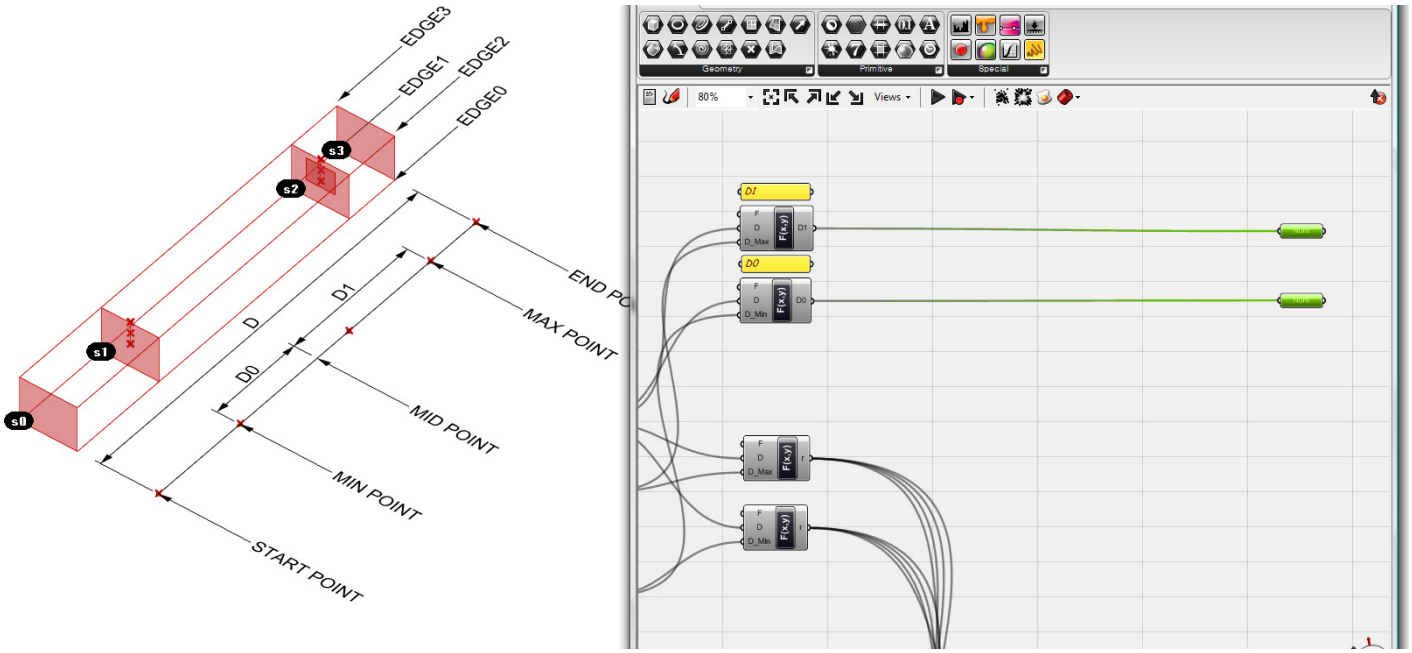
+ **Get mid point of two points. It will work as a scaling base point.**



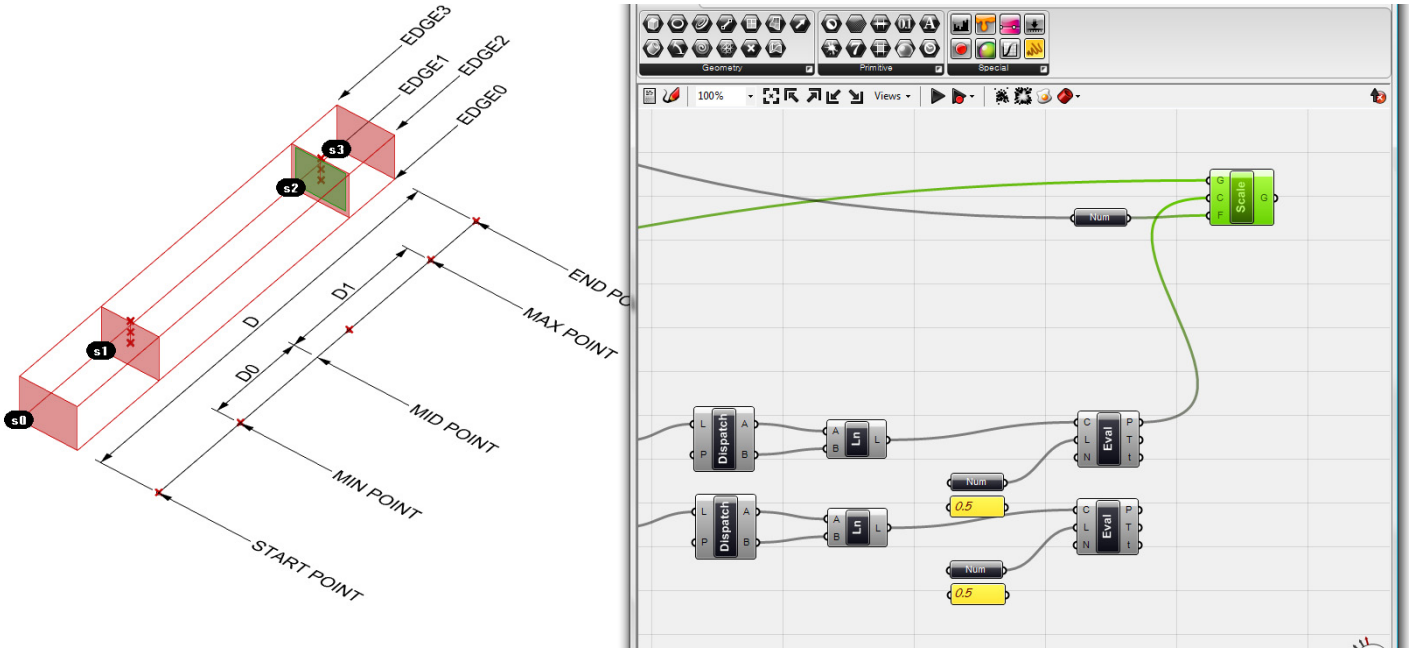
+ **Put a Scale object. Connect it to surface S1(or S2).**
  - You see the small green box on top of S3. That is because we did not connect scaling base point to Scale object.



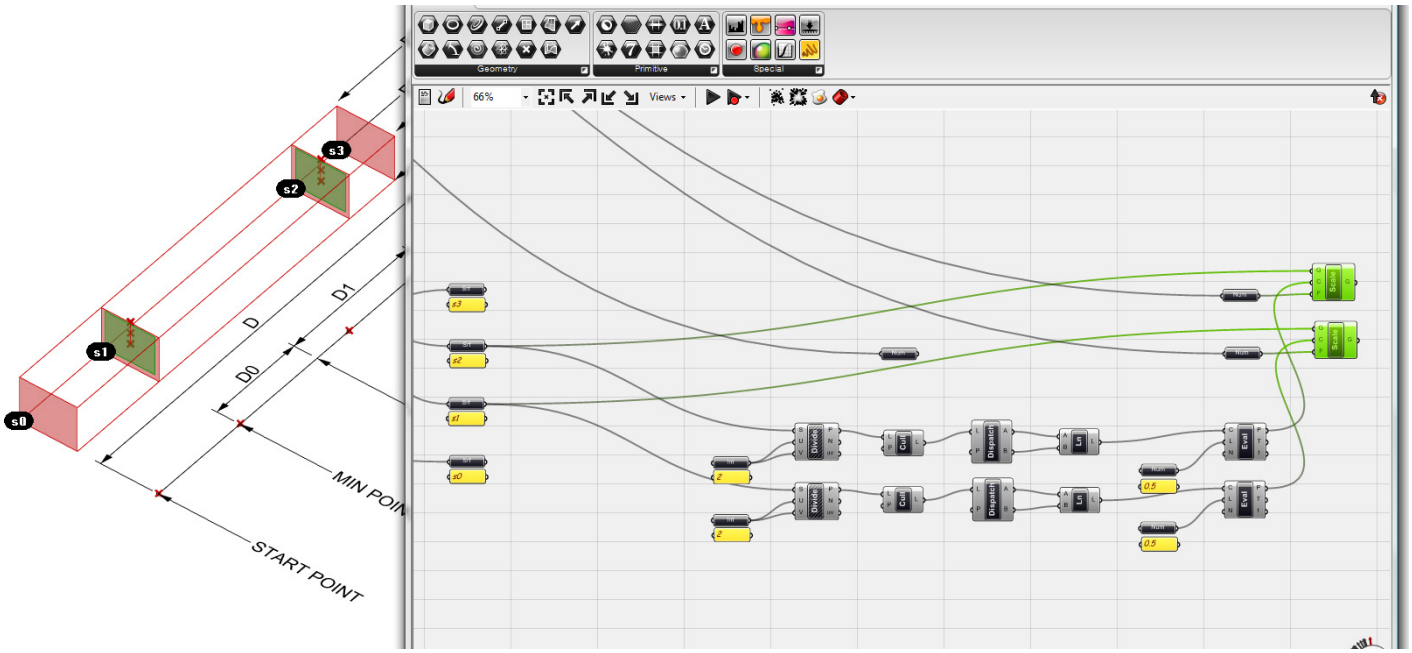
+ **Set scaling base point.**



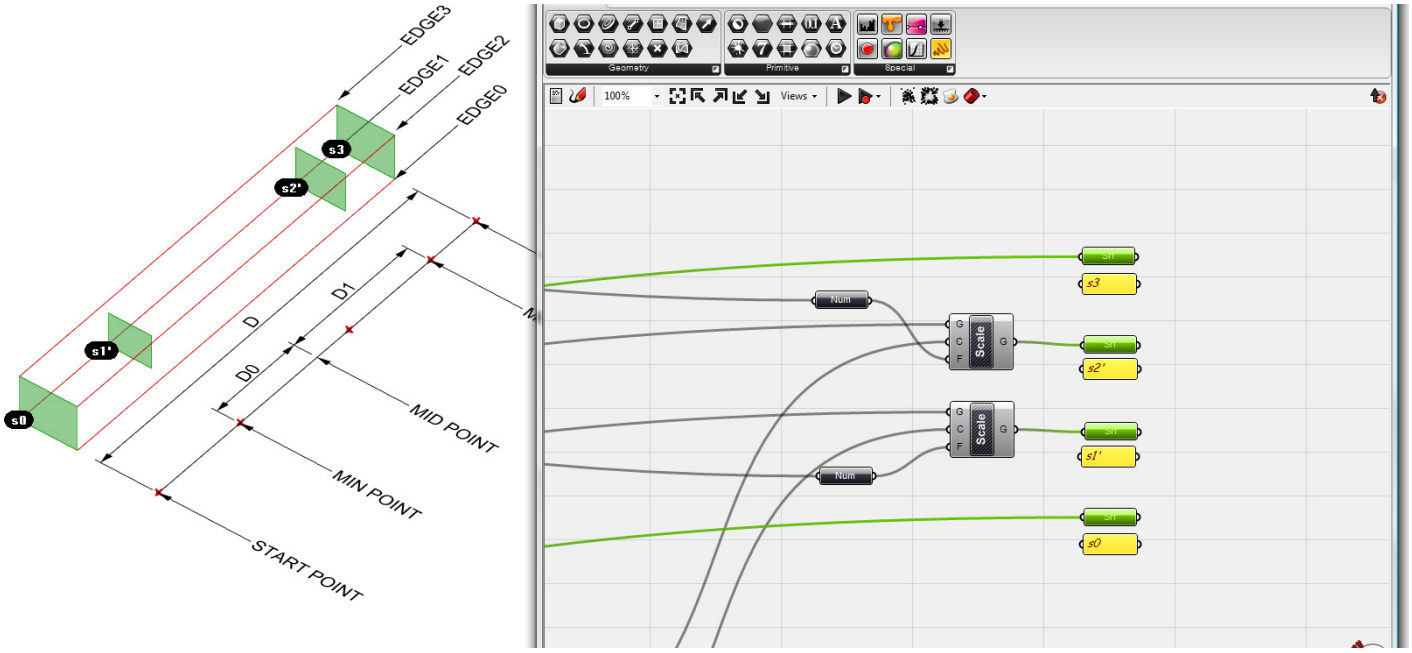
+ **Extract scaling factors that we defined at page step01_06.**



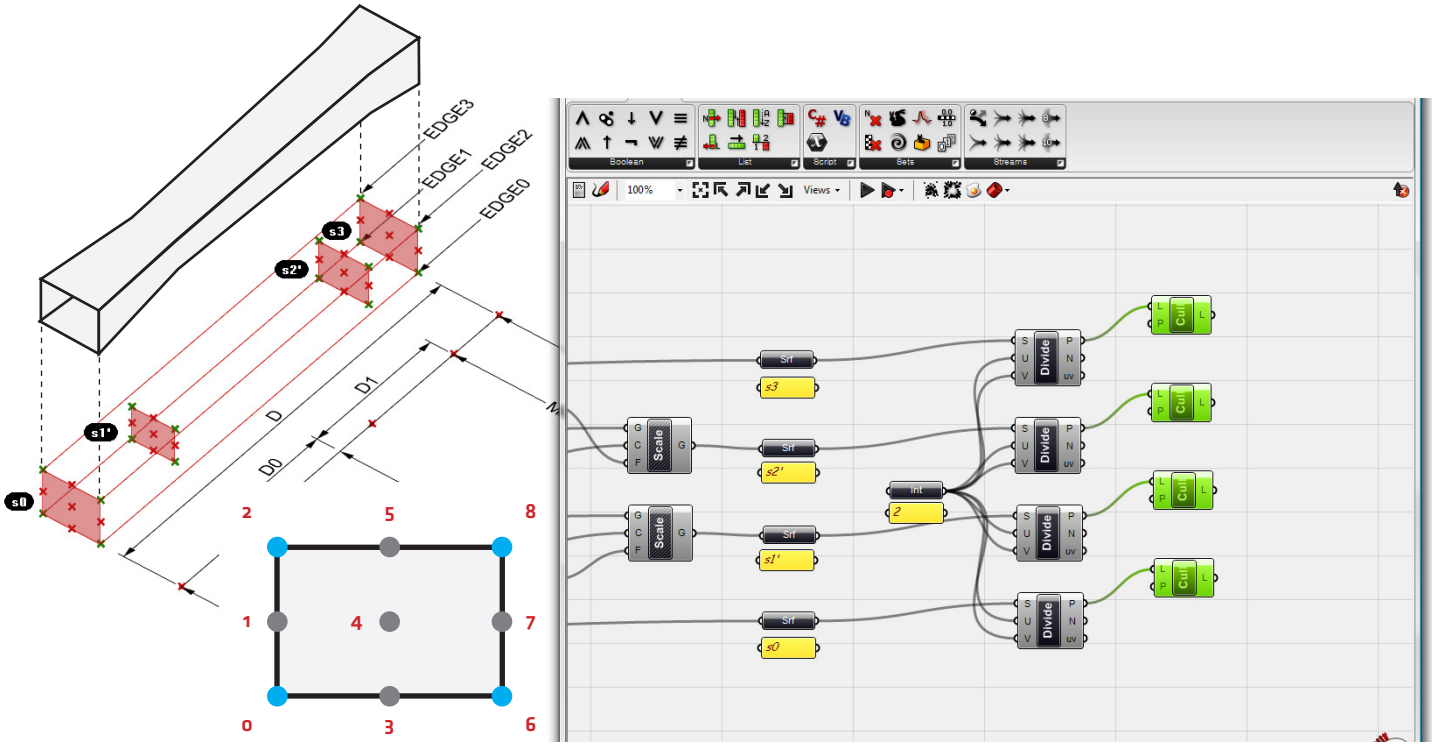
+ **Connect scaling factor to Scale object.**



+ **Do the same thing for the other surface.**

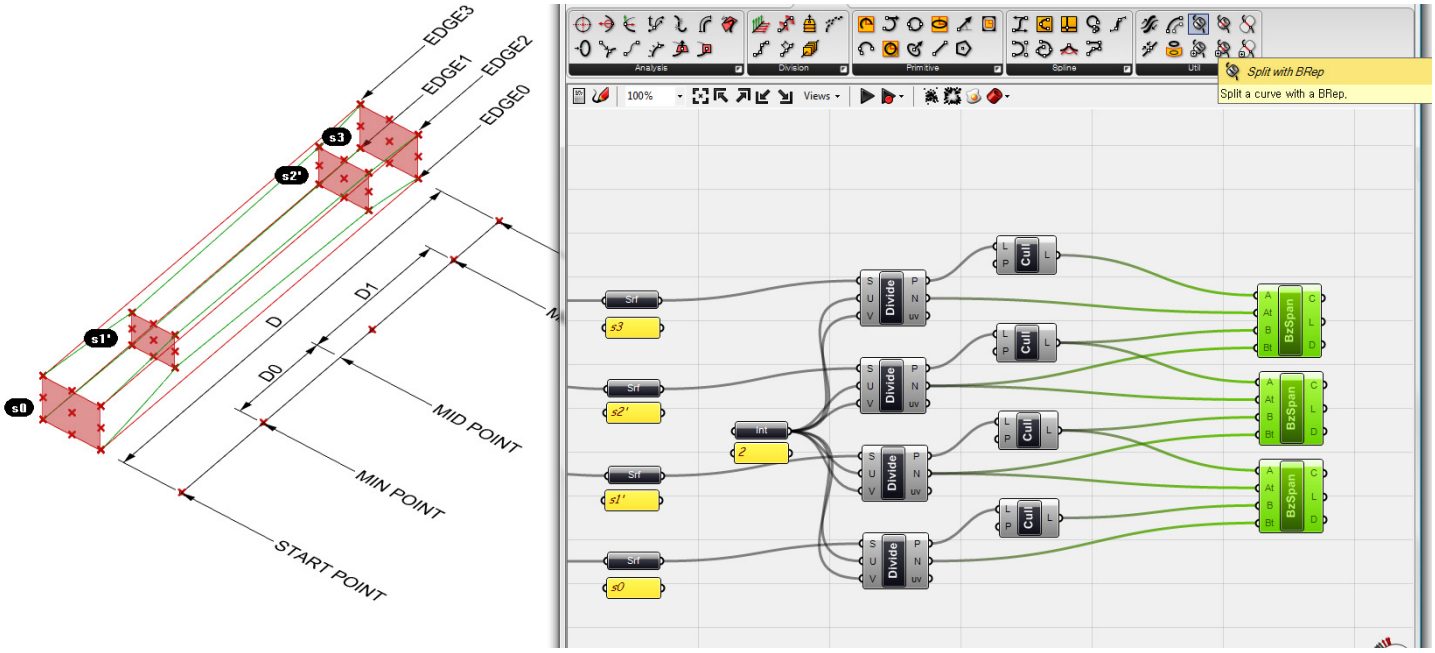


We have two surfaces at each end of box, and two middle surfaces which response to the scaling factor.



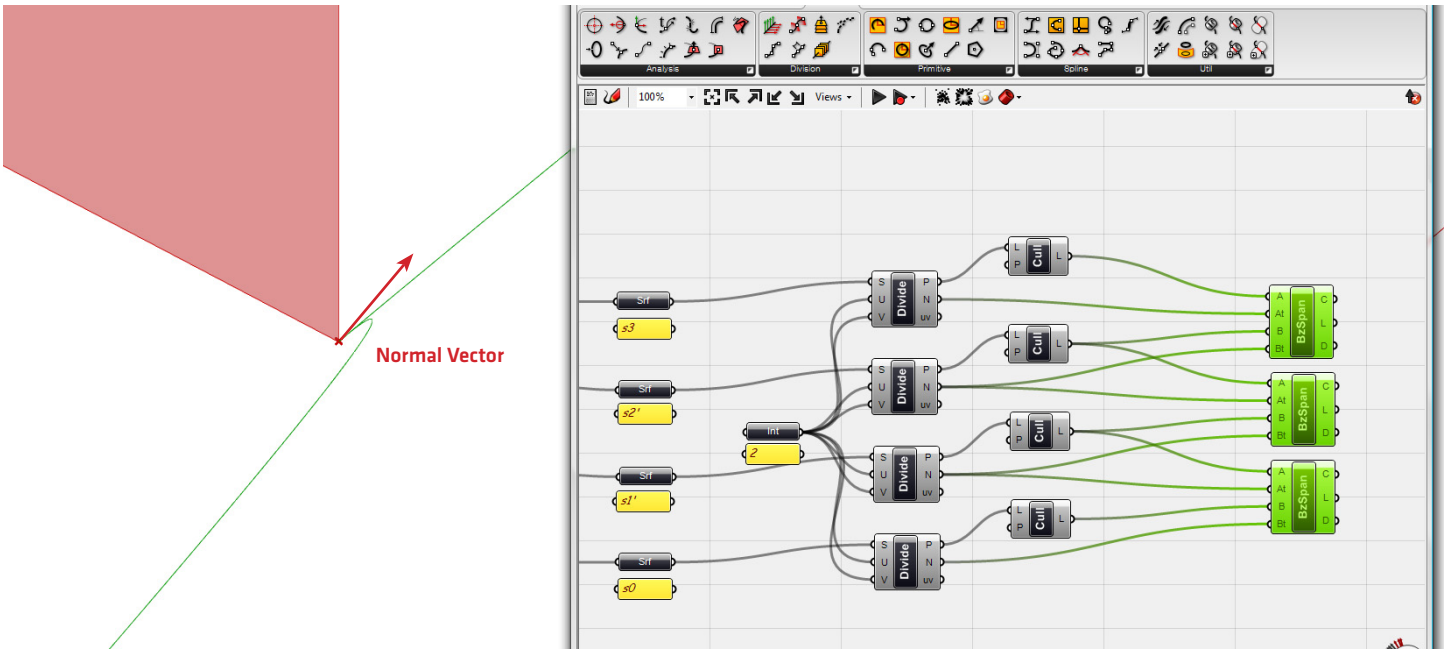
Next, we will make a deformed box geometry through four surfaces.

+ **Divide surfaces.**
+ **Extract corner points of each surface with "T-F-T-F-F-F-T-F-T" cull pattern.**

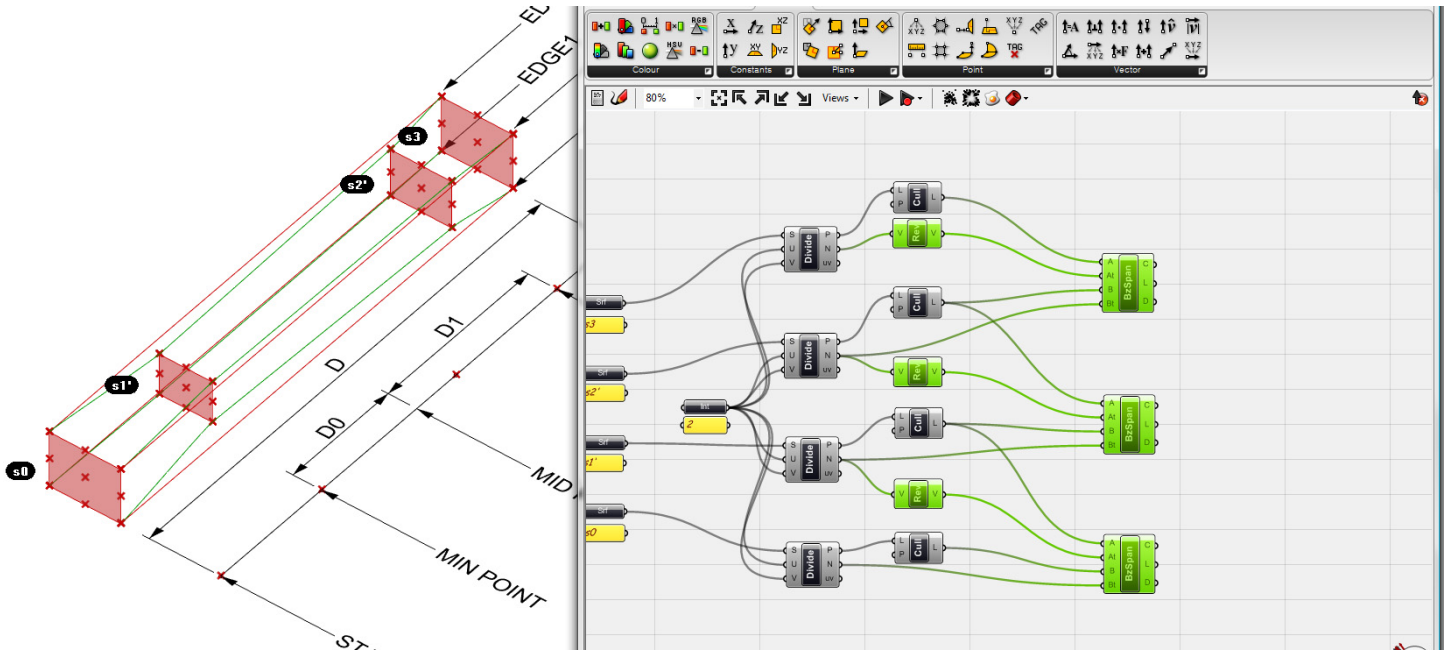


+ **Make bezier curves using BzSpan object.**

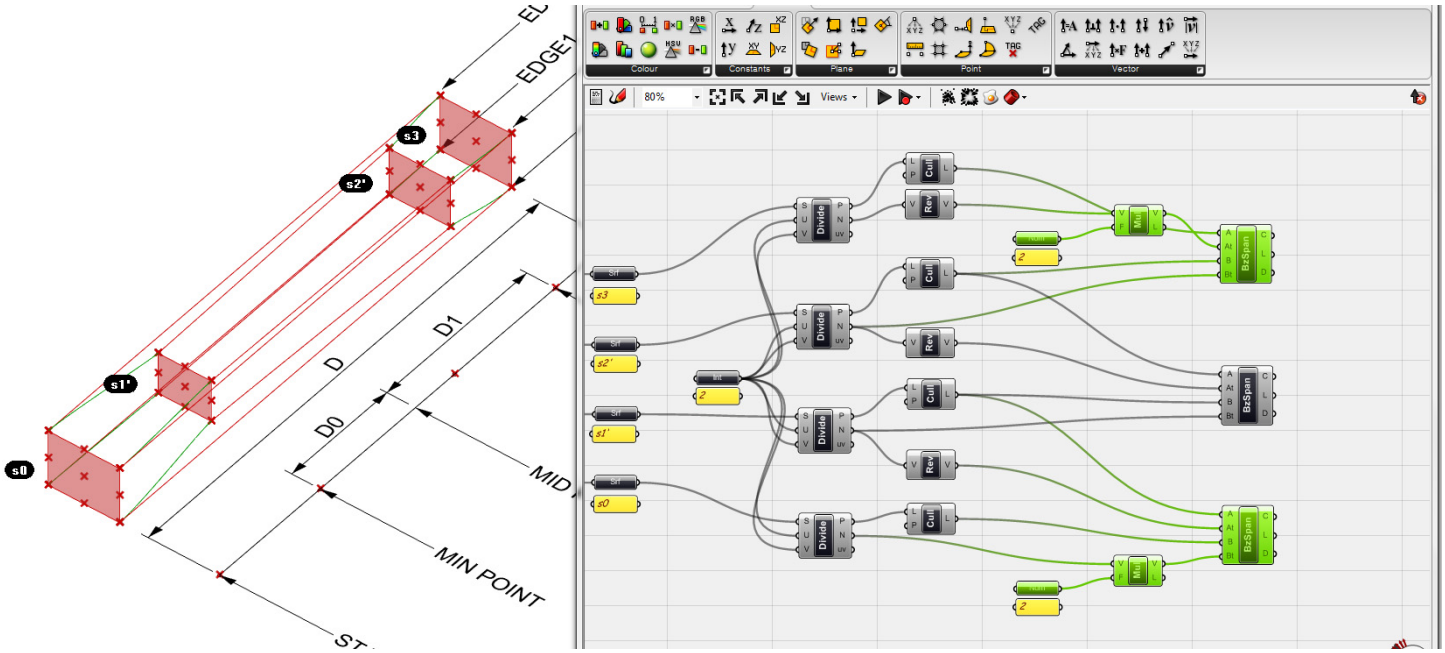
Two sets of points and normal vector for one bezier curve.



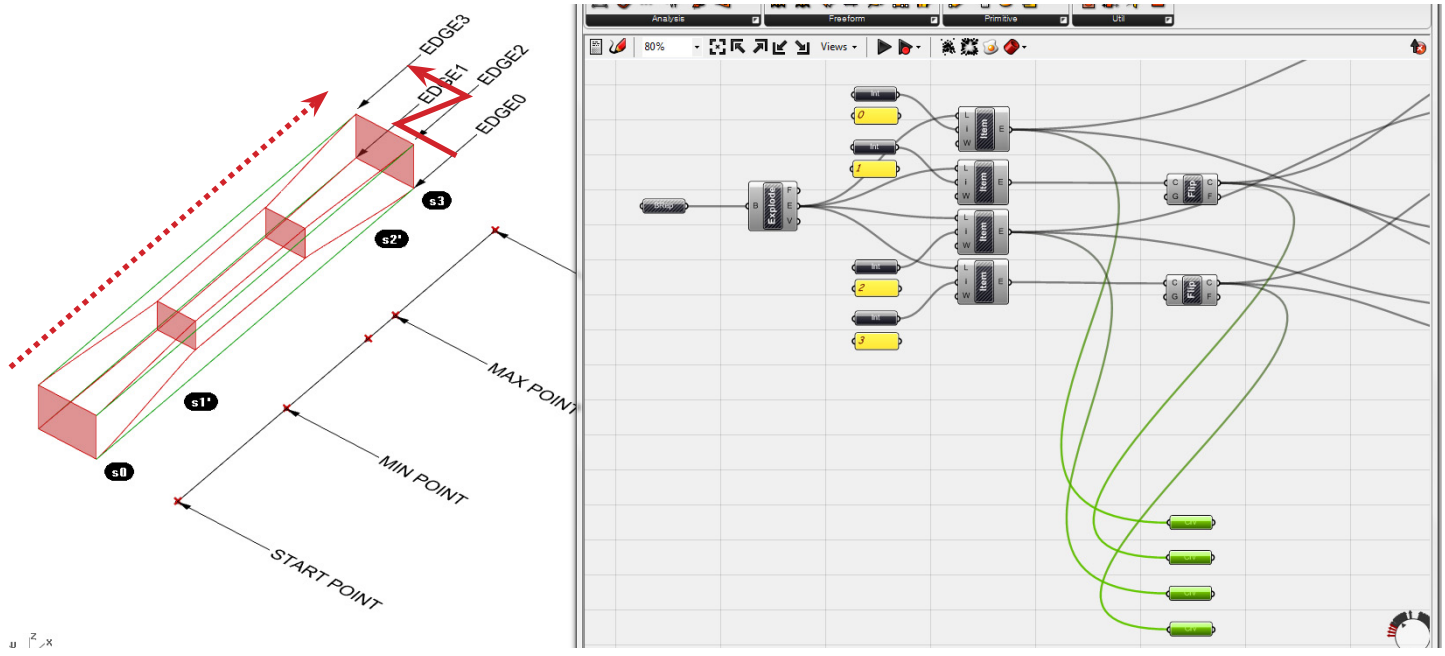
- Some curve ends need to be fixed by reversing normal vector direction.



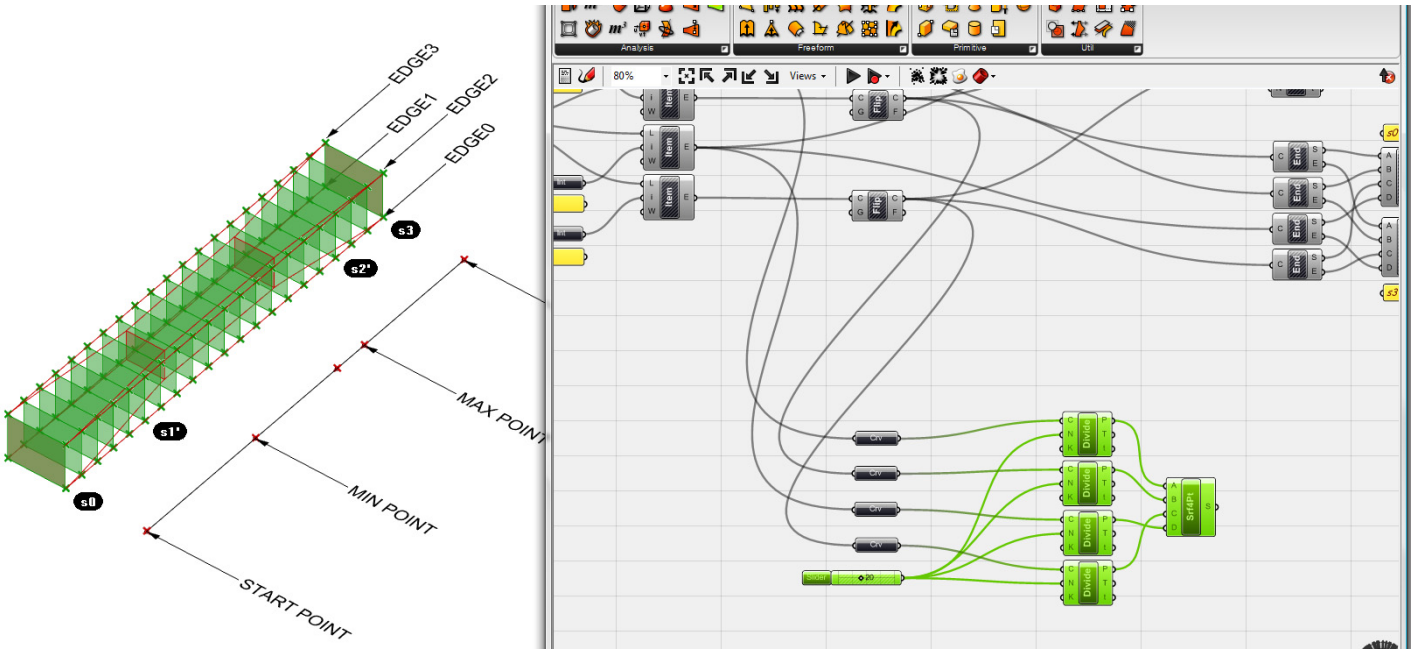
+ **Use Reverse Vector object to fix the end problem.**



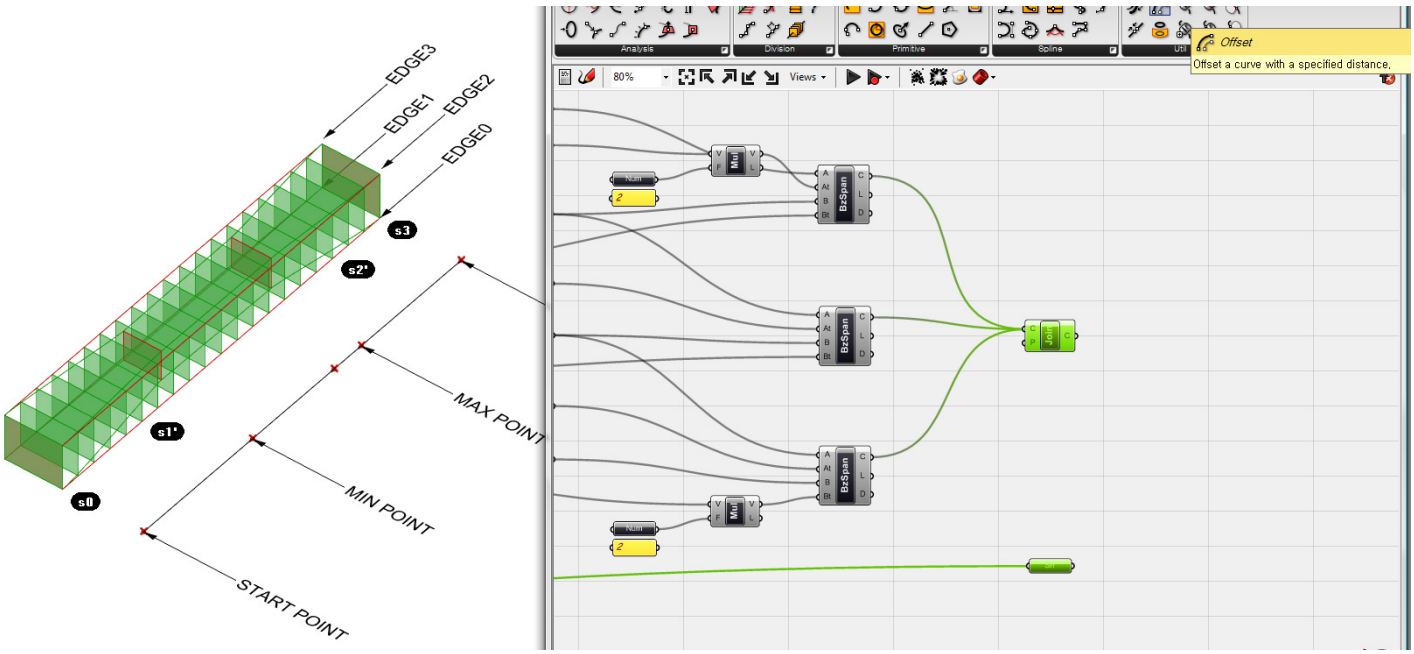
You can also have smoother curve by multiplying normal vector. I would set the multiply factor as 2 for now.



- Initially, I wanted to make surfaces out of those four deformed edge curves. However, it looks tough because of the edges' zigzag order. So, I changed plan to generate series of sections along with its longitude direction and loft them together.

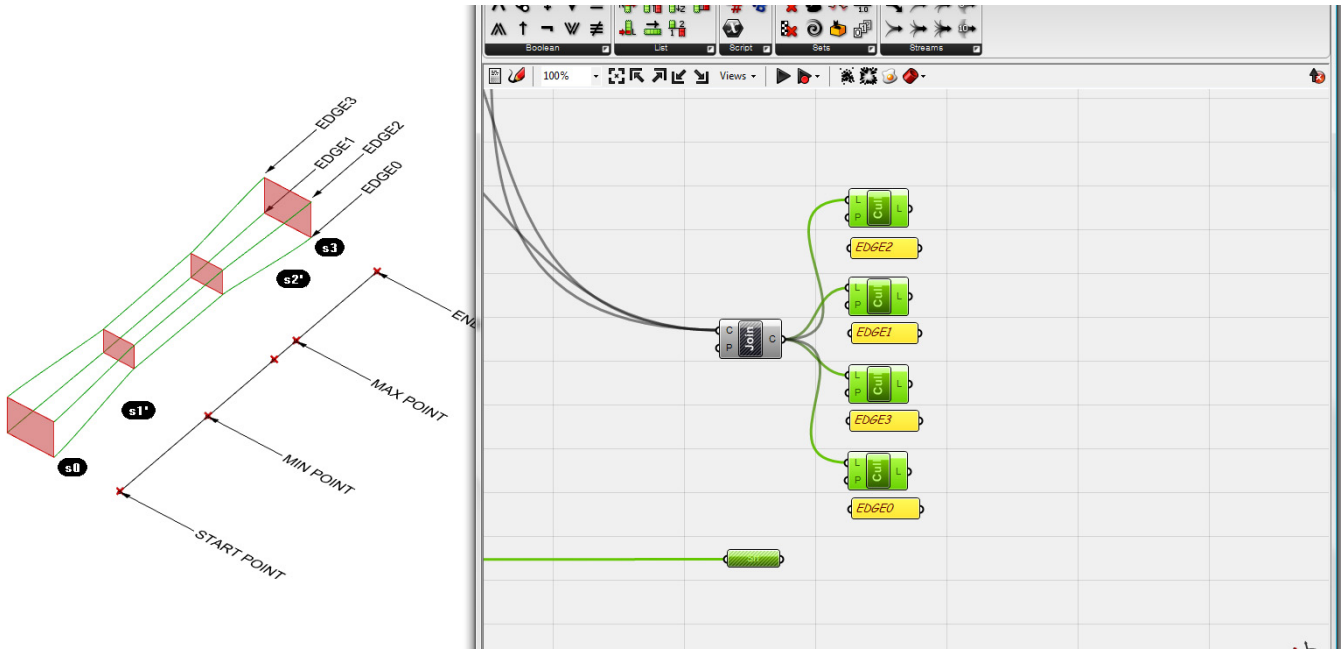


+ **Divided the edges using Div Crv objects.**
+ **Set up a Numbers Slide object for number of division.**
+ **Generated series of surface using Srf4Pt object.**

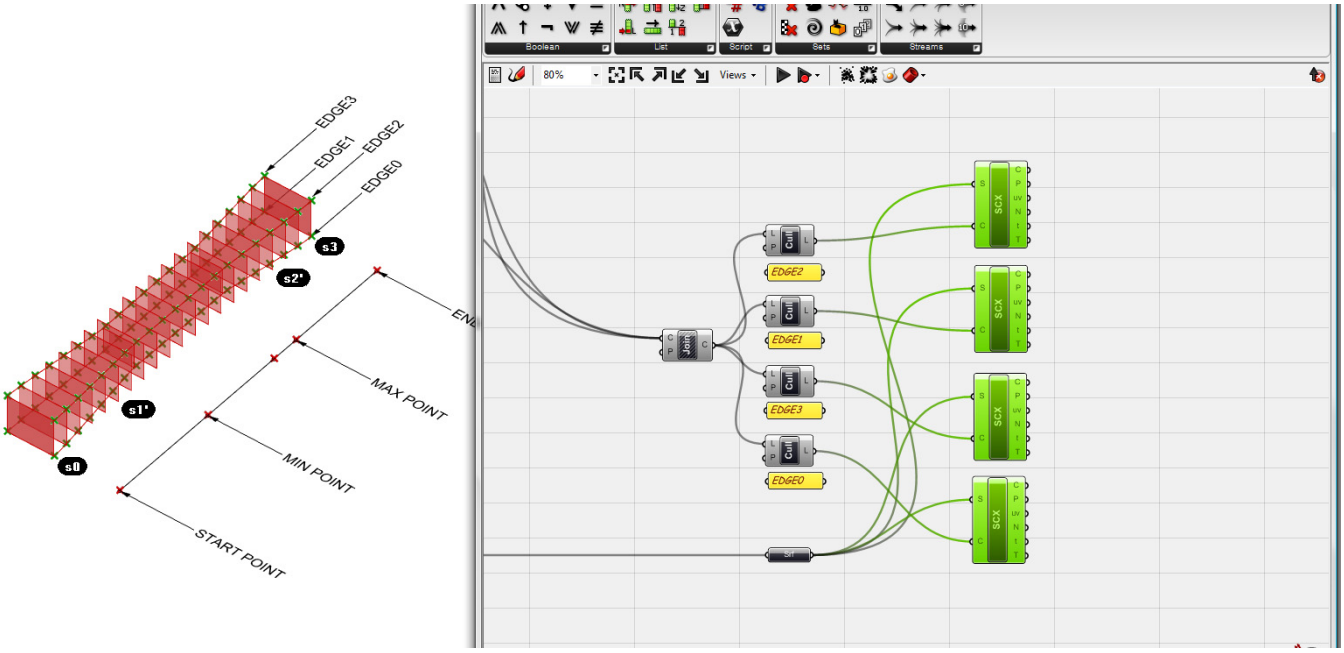


+ **Join three sets of bezier curves using Join Crv object.**

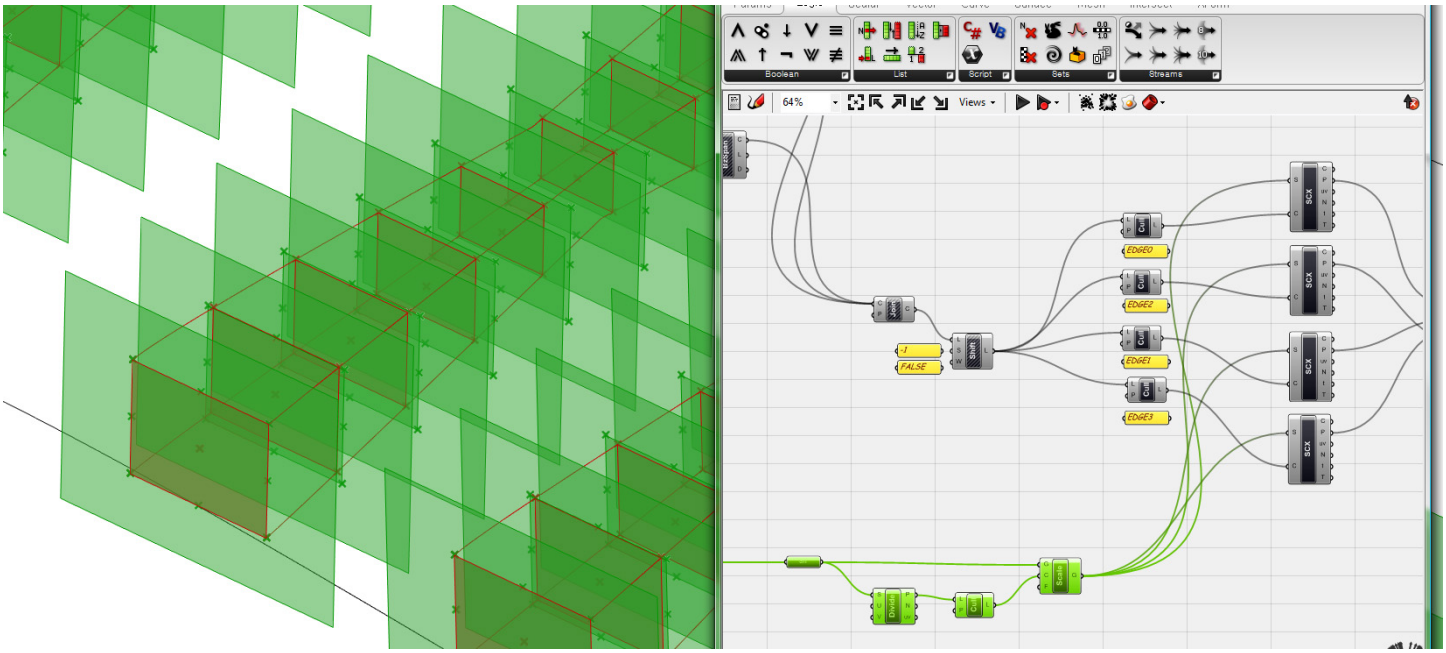
  These bezier curves will intersect with surfaces to generate section profiles for lofting.



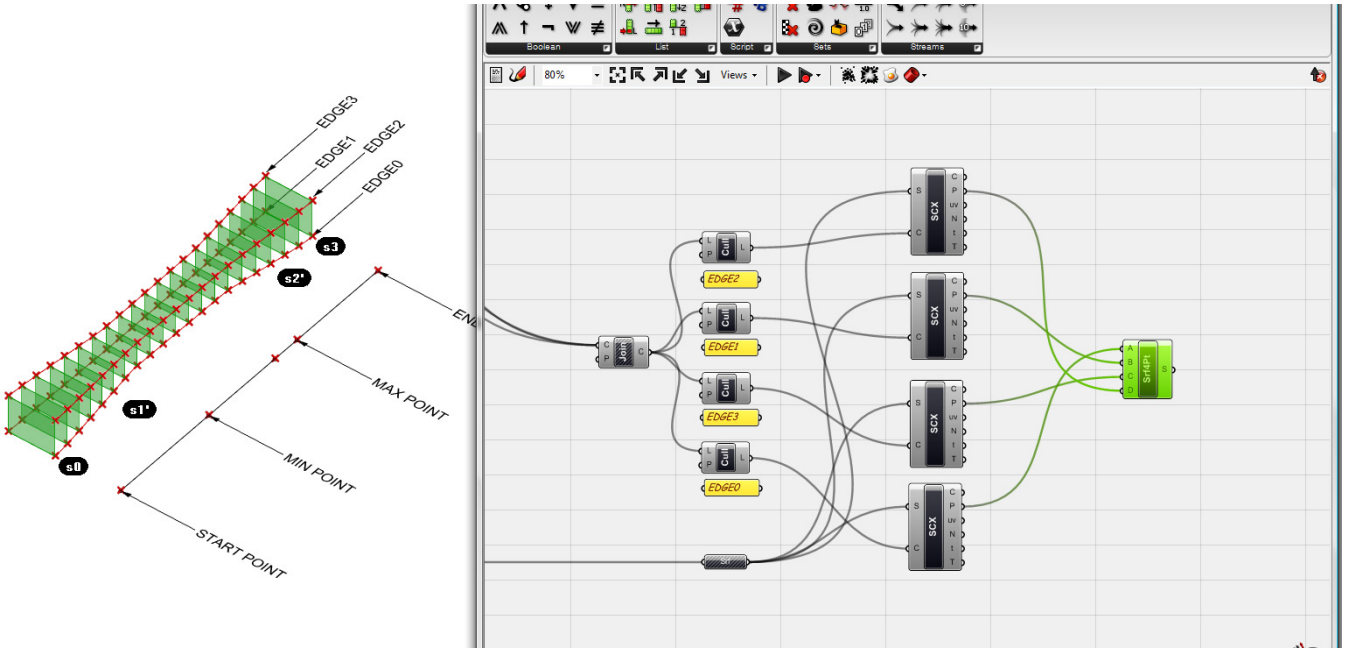
+ **Split the bezier curve set into four individual bezier curves.**



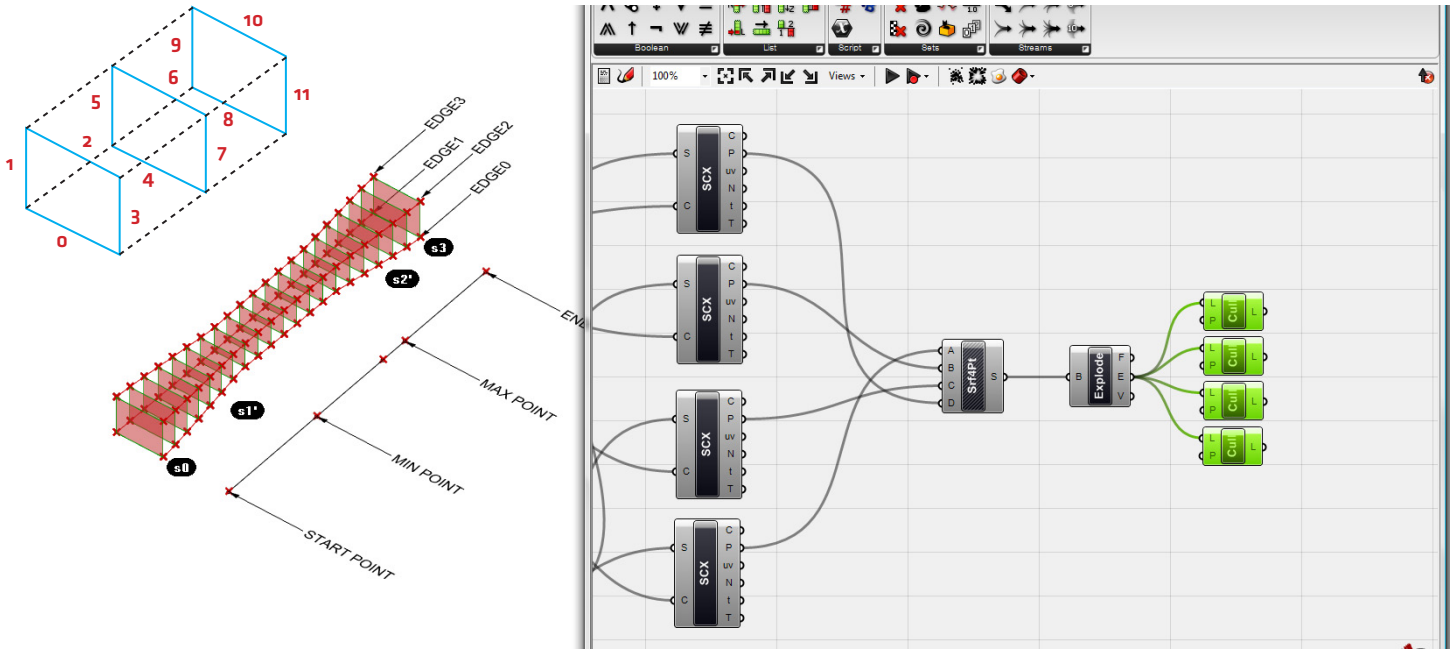
+ **Get intersect points using Surface and Curve Intersection object.**



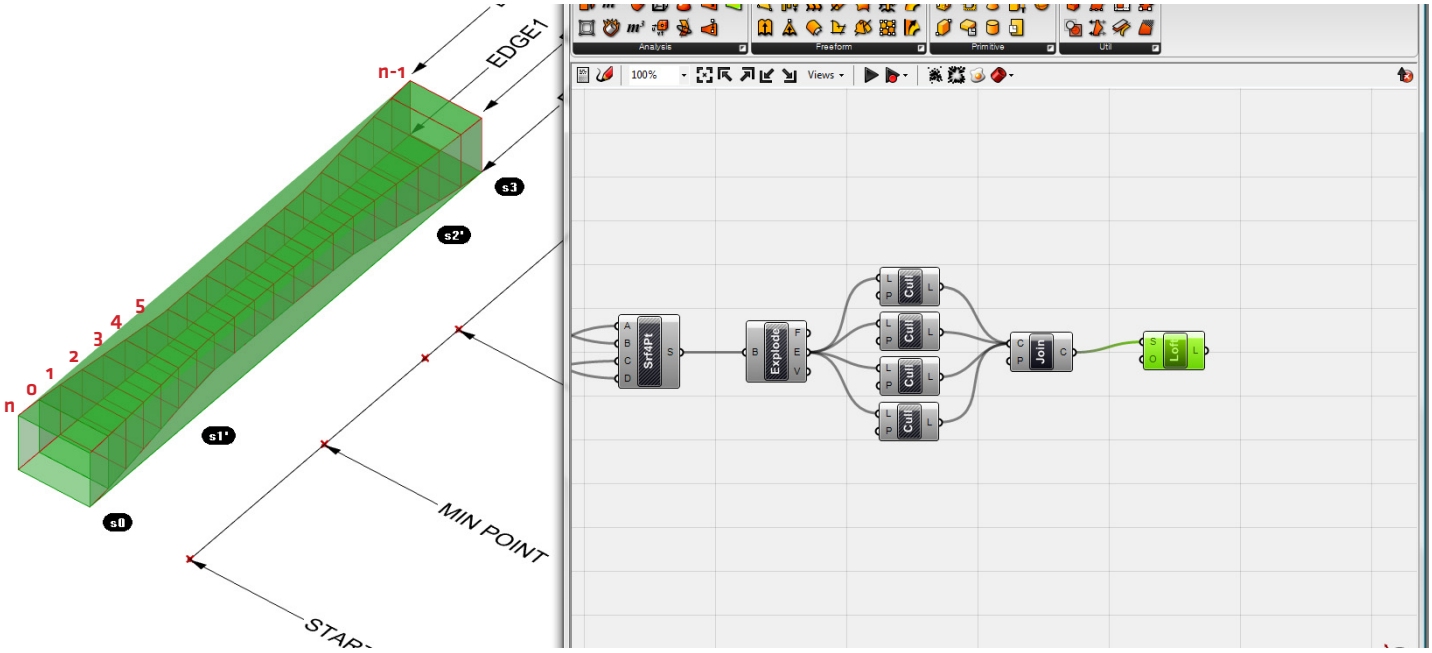
+ **To ensure that every one of them intersects, scaled up surface little bit using Scale object.**
  - Scaling up too much will cause a problem with neighboring curves. I put 1.1 as scale factor.



+ **Make surfaces that fit into bezier curves using four point sets.**

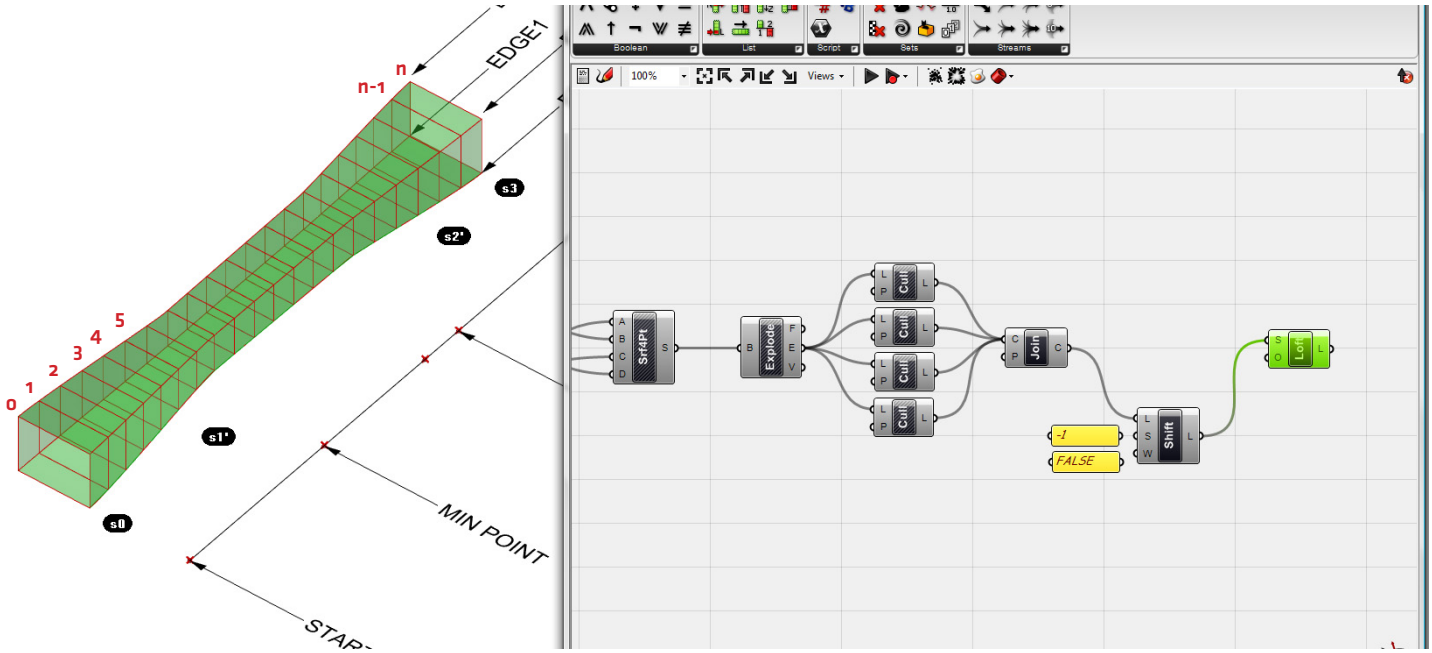


+ **Explode surfaces to get surface edges.**
+ **Sort edges using Cull Pattern object.**
  - Before sorting, exploded edges are in single spiral order. (0,1,2,3/4,5,6,7/8,9,10,11). After sorting, there are four different lists. (0,4,8)/(1,5,9)/(2,6,10)/(3,7,11)
+ **Join edges to make loft profile curves.**



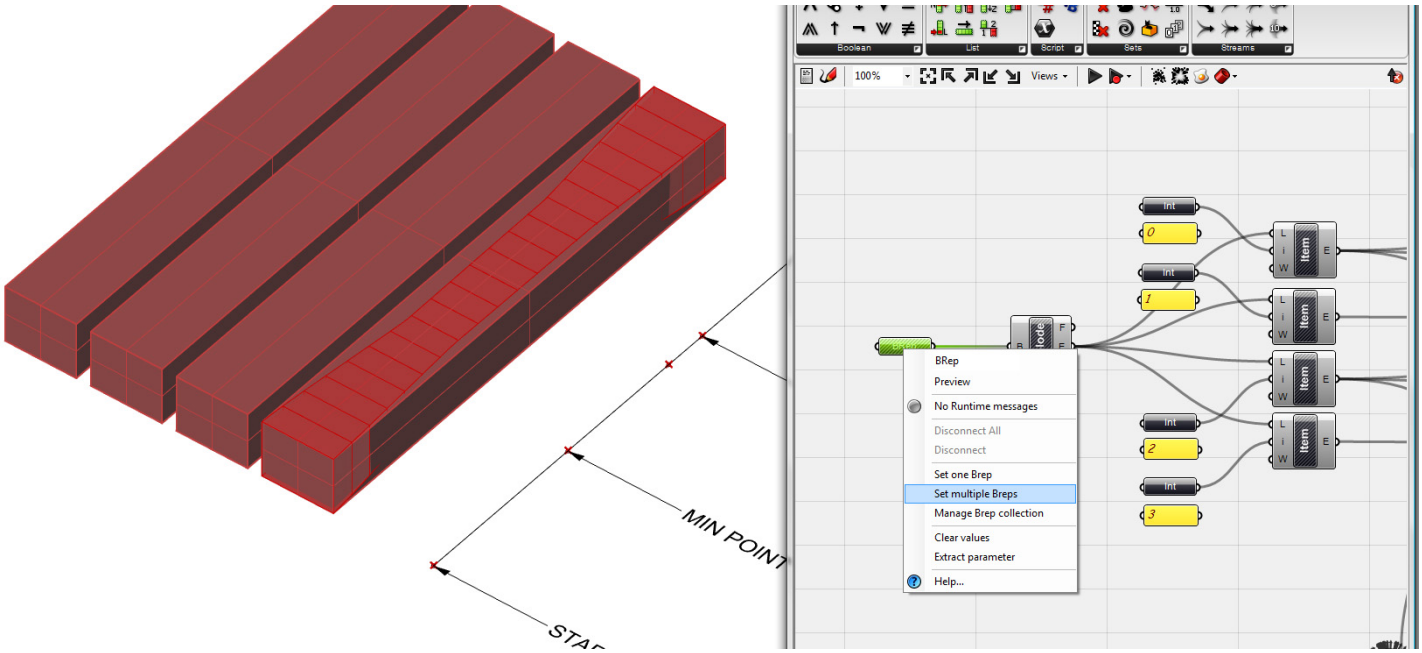
+ **Loft joined curves.**

  Note that we should check straight loft in the loft option.
  - We have a problem here. This is because of the order of curves. 1st curve of loft should be the curve on S0 surface. But it seems that loft start from the second curve.

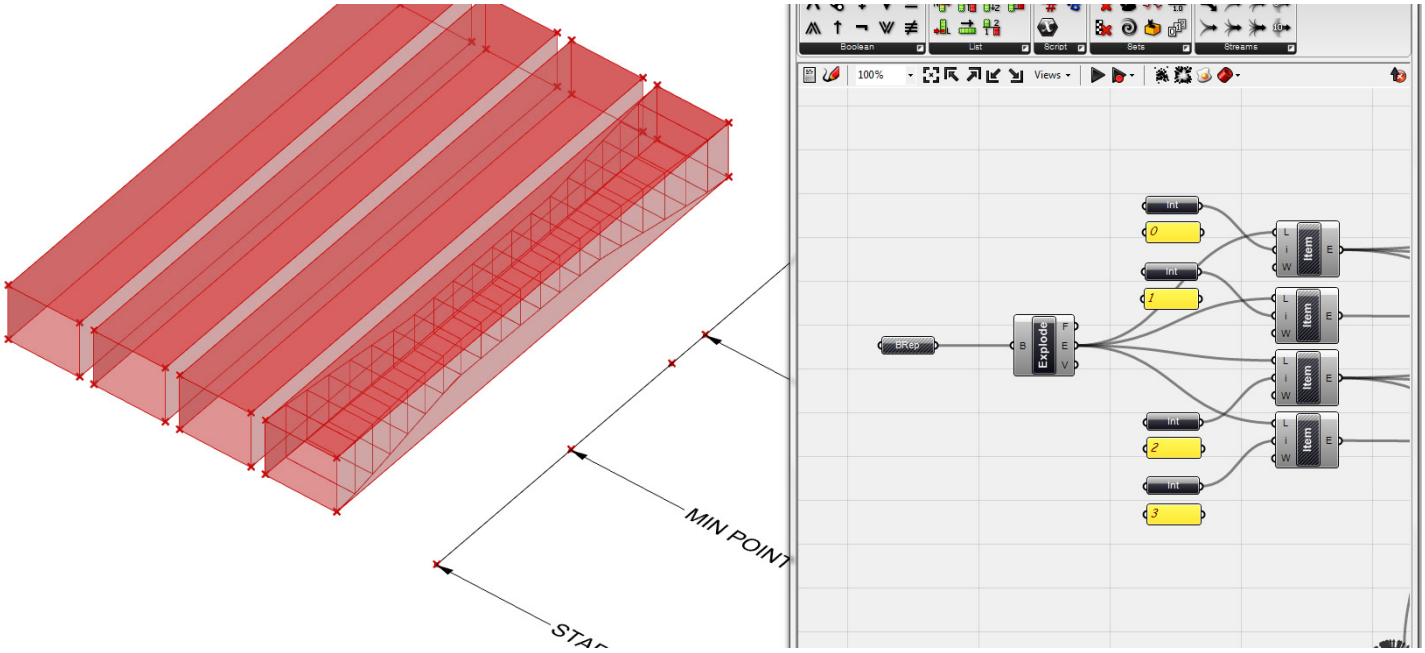


+ **Shift curve list using Shift Item object.**

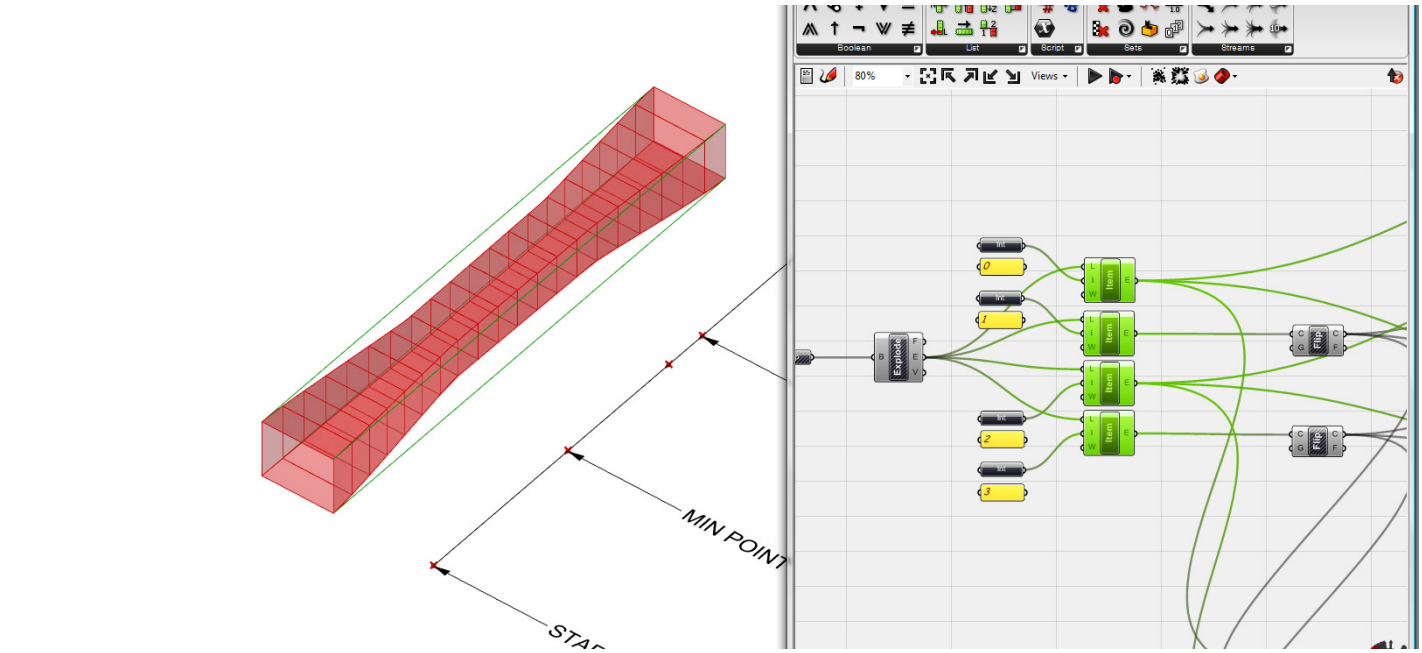
  Put -1 and False as parameters.



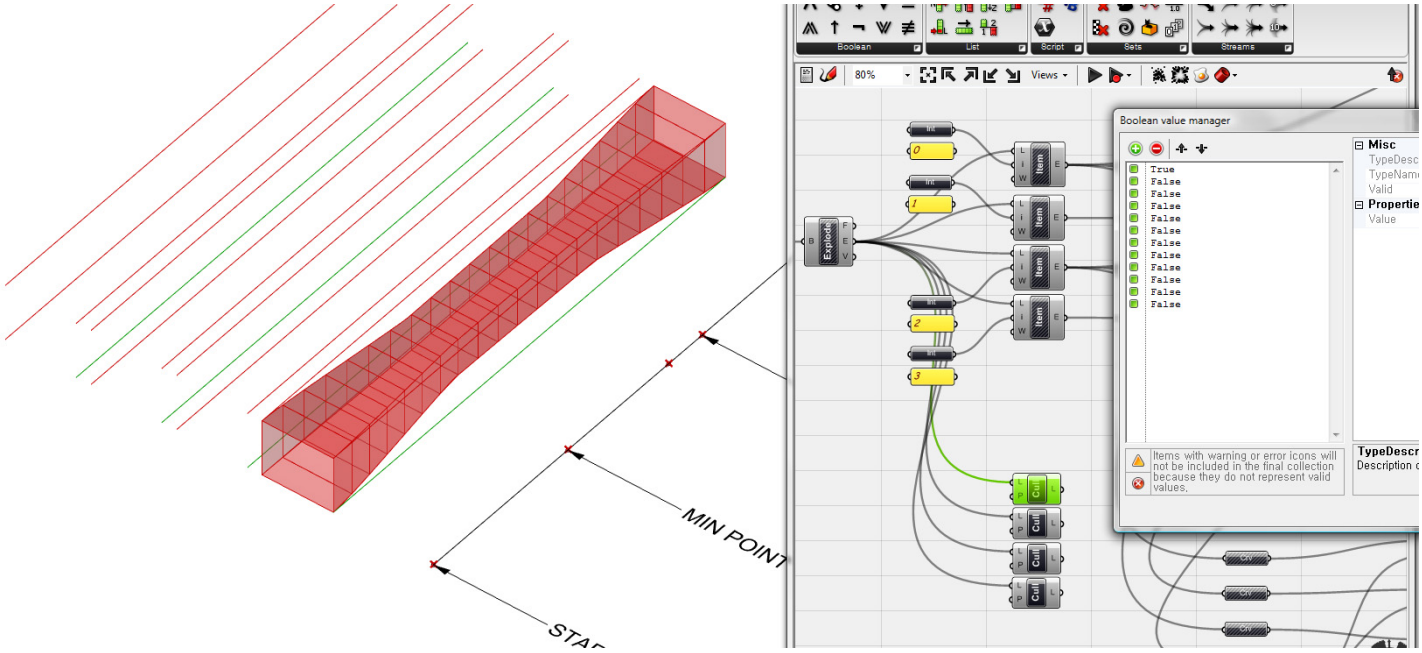
+ **Set multiple brep to replicate a component.**



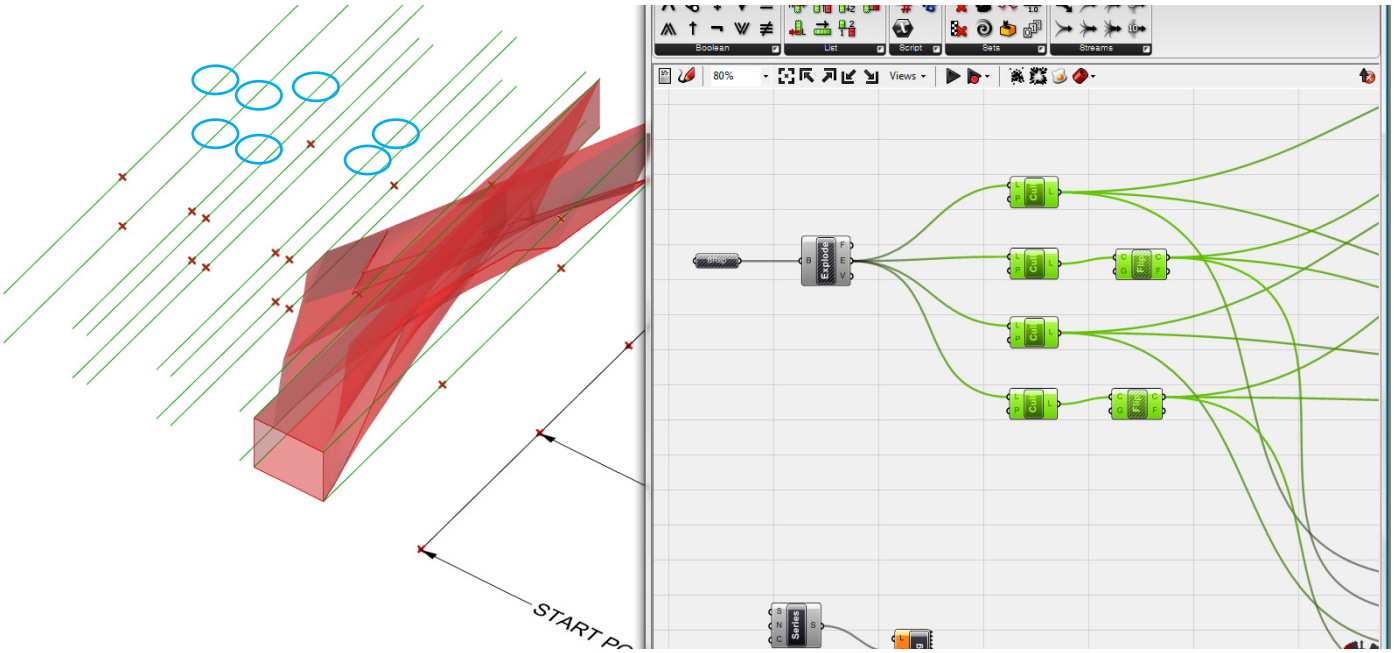
• We have a problem here. It just works for the first box though we selected four boxes.



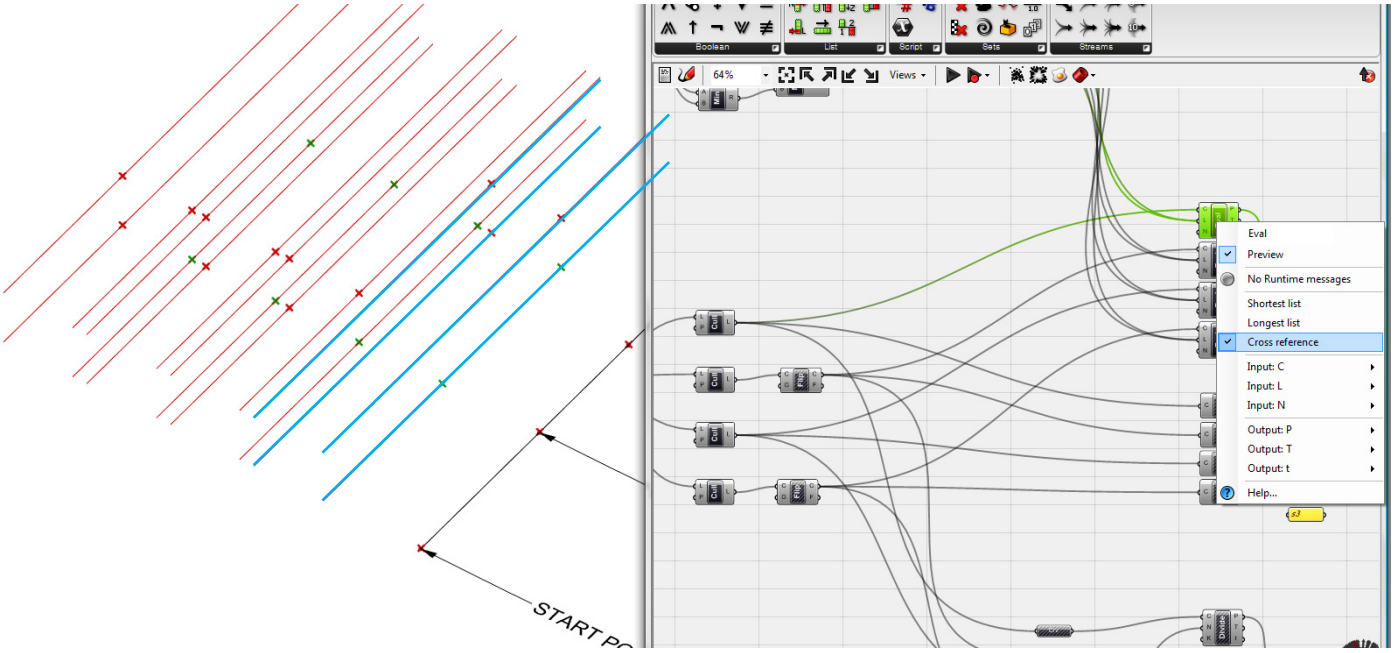
• Remember that we used list item objects to extract four edges out of the initial box. But, here we need every 1st, 2nd, 3rd and 4th edges for each box.



+ **Instead of list item object, use cull pattern object. Set the cull pattern like above.**

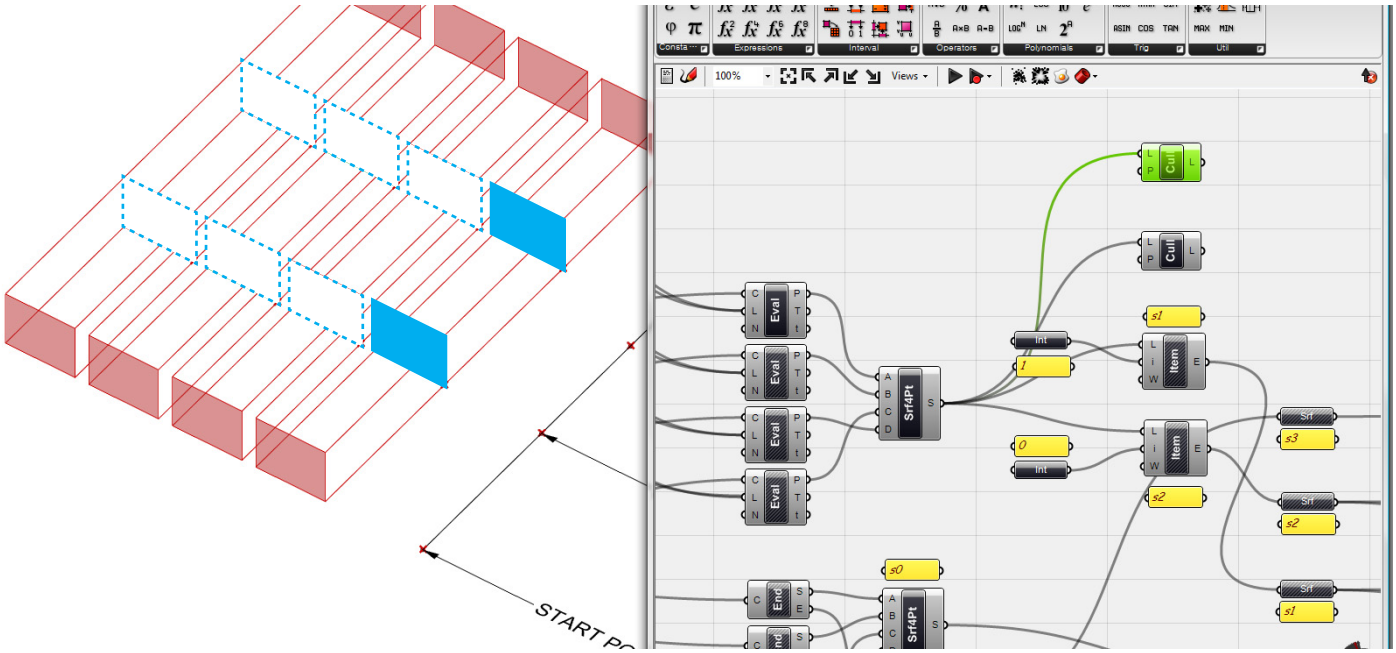


- We have a problem here again. You see that there are some missing points. This is because of data matching process.

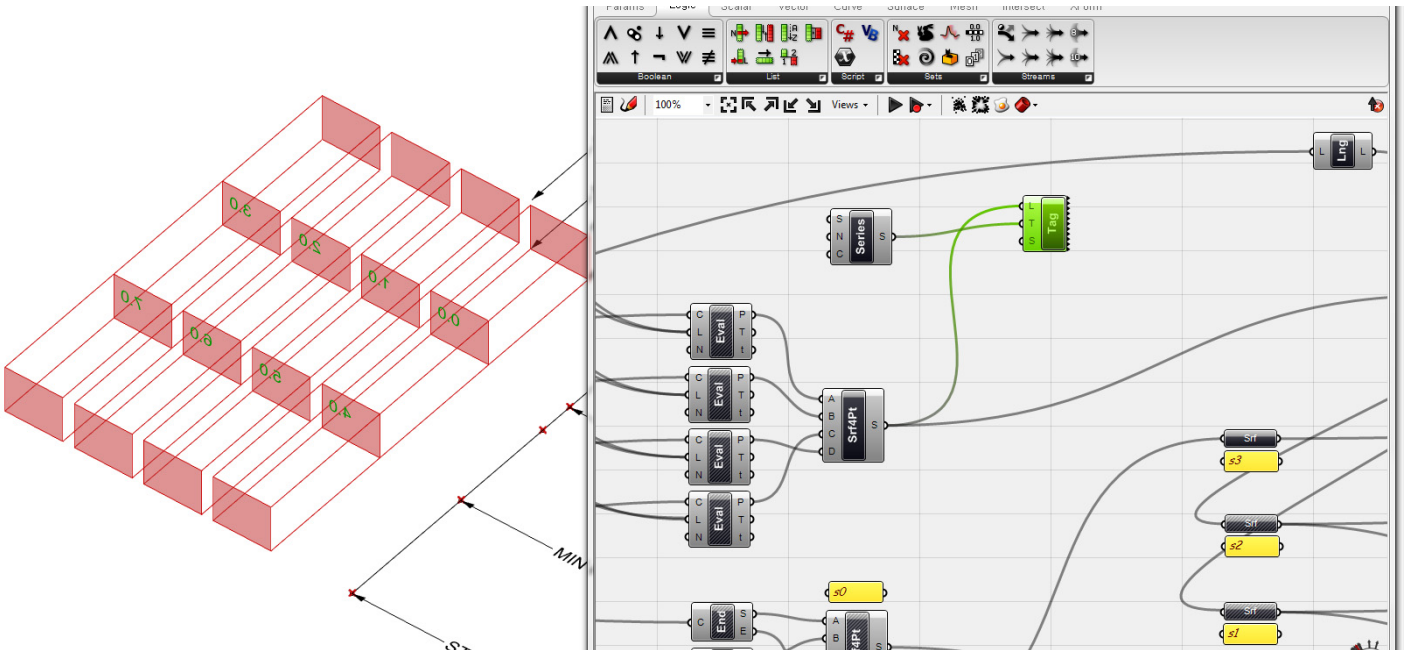


- When we had one initial box(brep), we had only one curve and two evaluate lengths for each evaluate curve object. However, now we have more than one box(breps), and curve input is not a single curve but a list of curves.

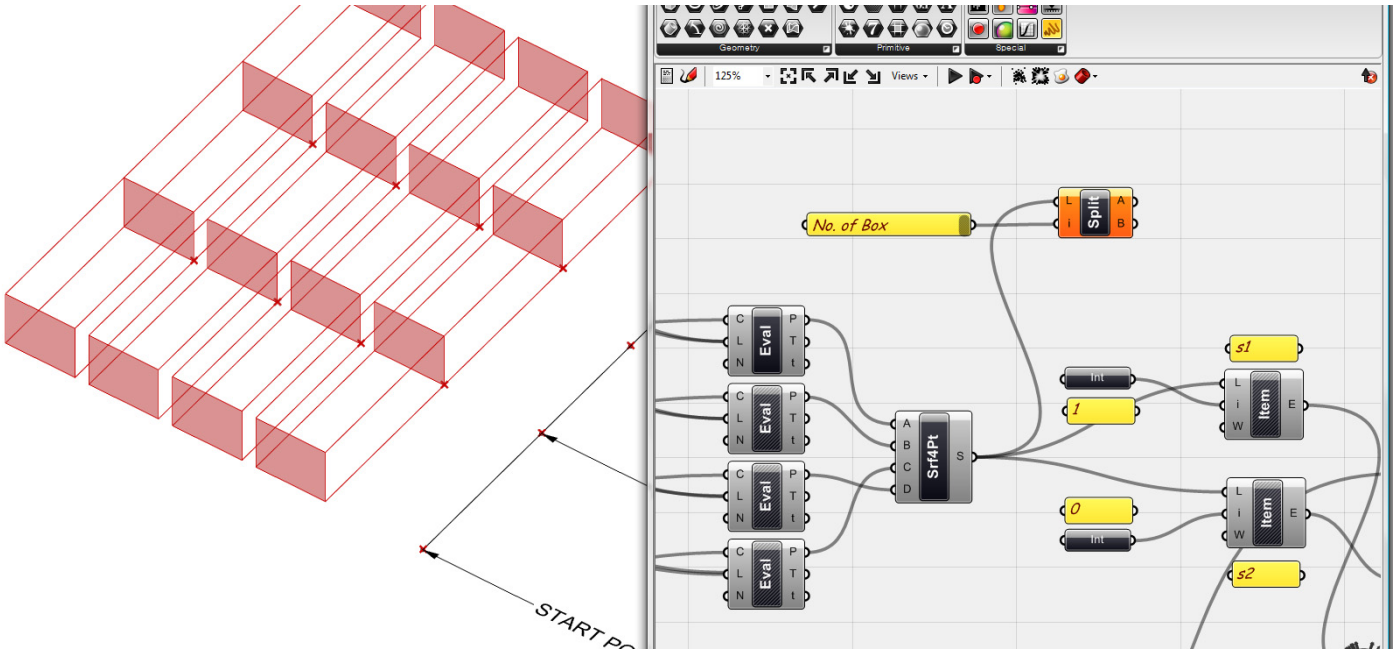
+ **To find all matches, set data matching method as Cross. Otherwise, we will just miss some points.**



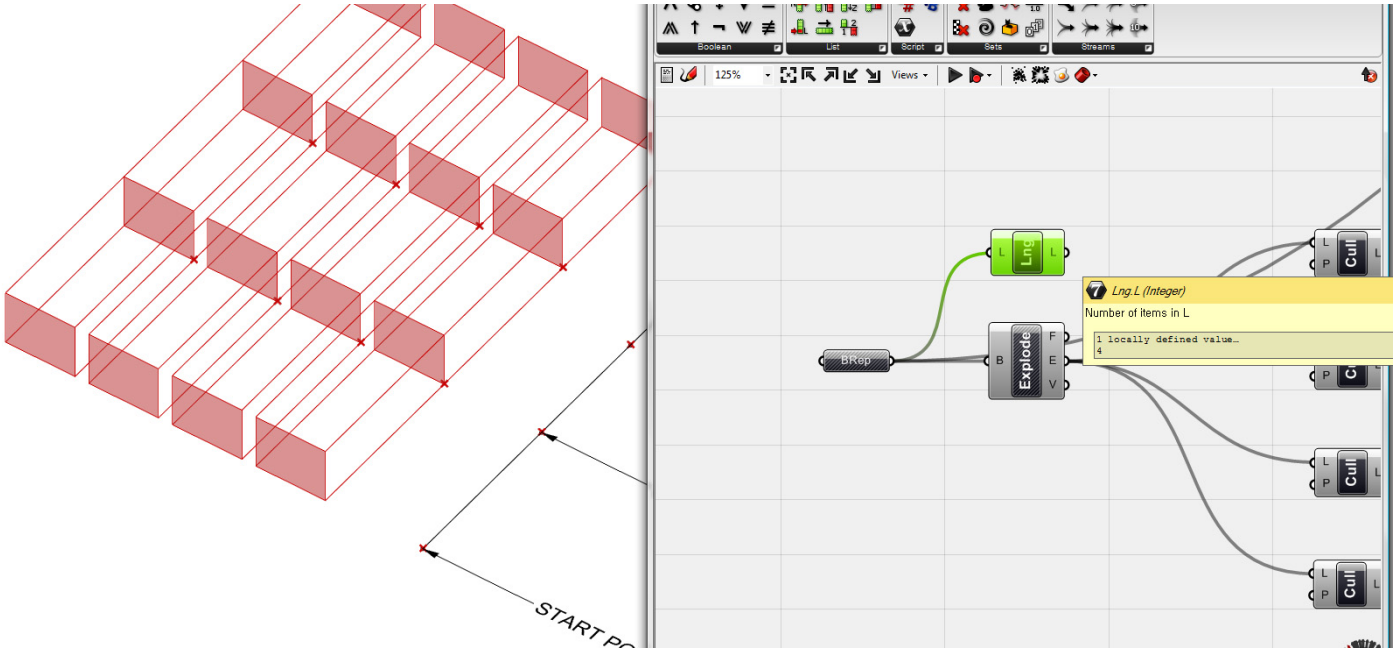
- By now, you must have found out that the geometry still looks weird. When we had only one box(brep), we had two middle surfaces. Since we now have four boxes(breps), with the same List Item objects, we will end up missing rest of surfaces. Refer to the page step01_03, 05.



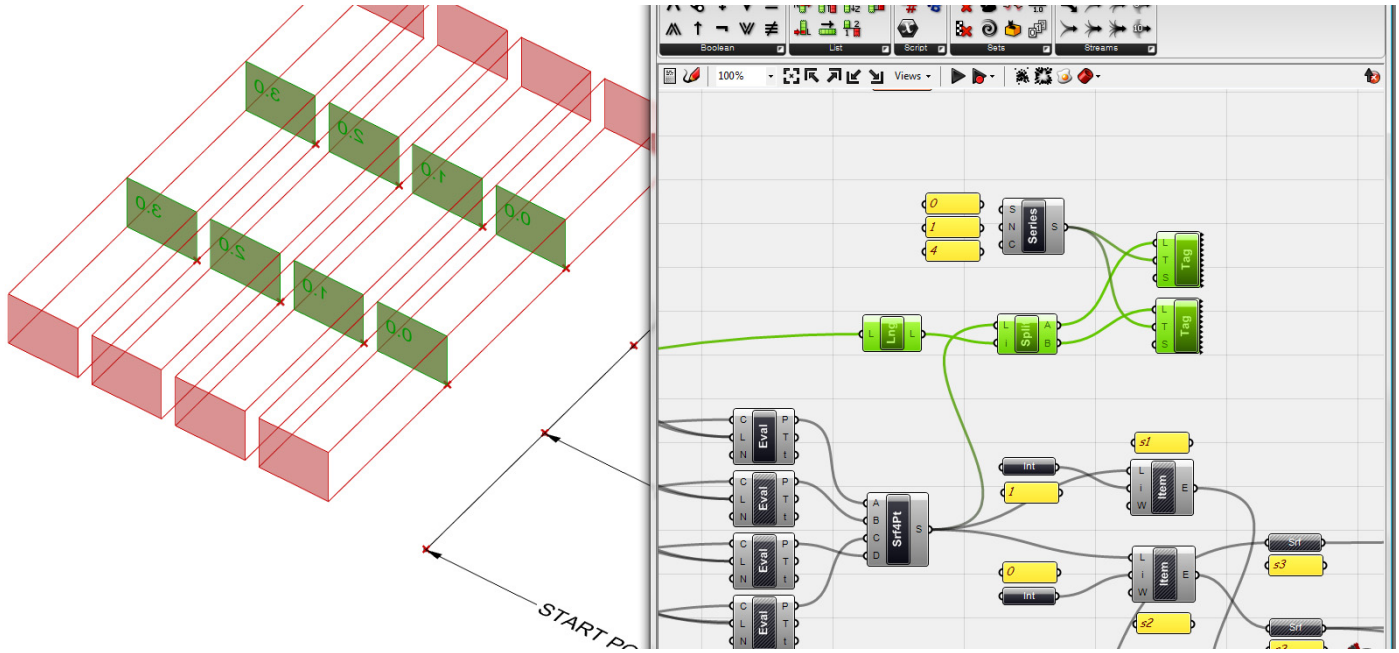
+ **Put 3d tag object to the middle surfaces to see their order.**



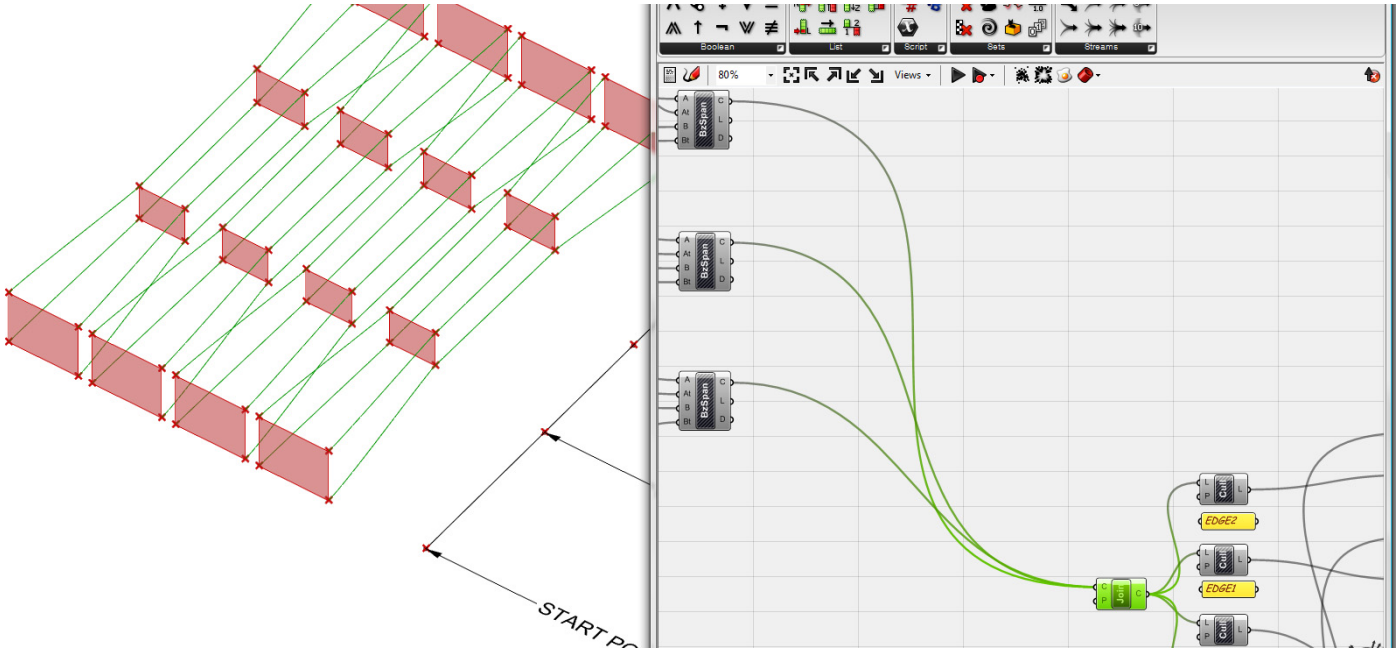
+ **Split surfaces into two groups.**



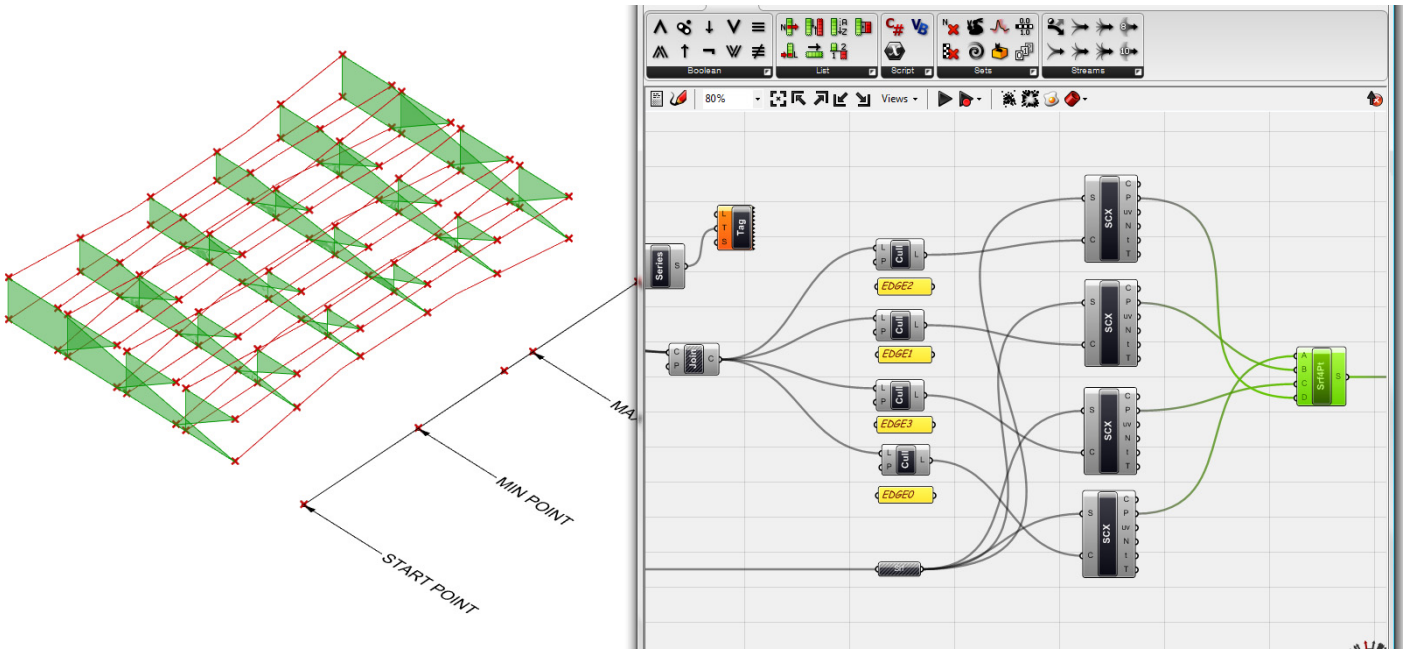
+ **To get the split point, get the item length of boxes(breps).**



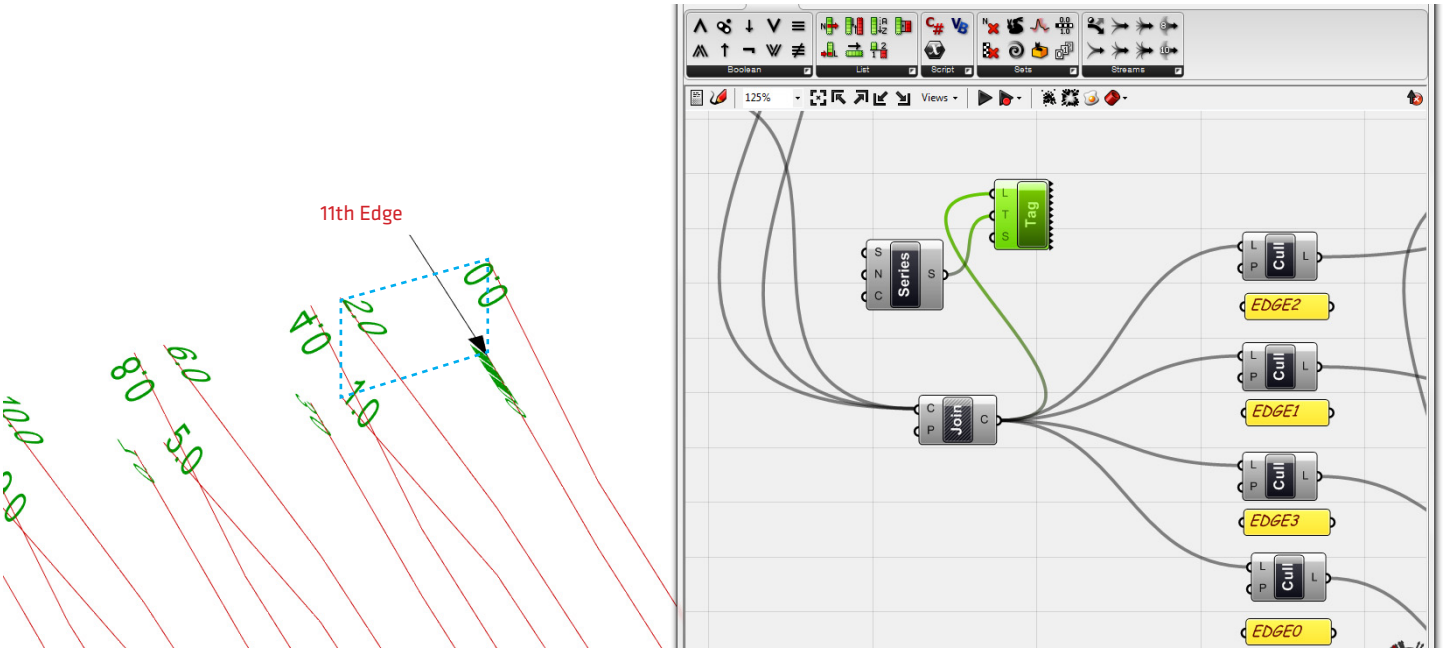
+ **Attach 3d tag objects to check surface order.**



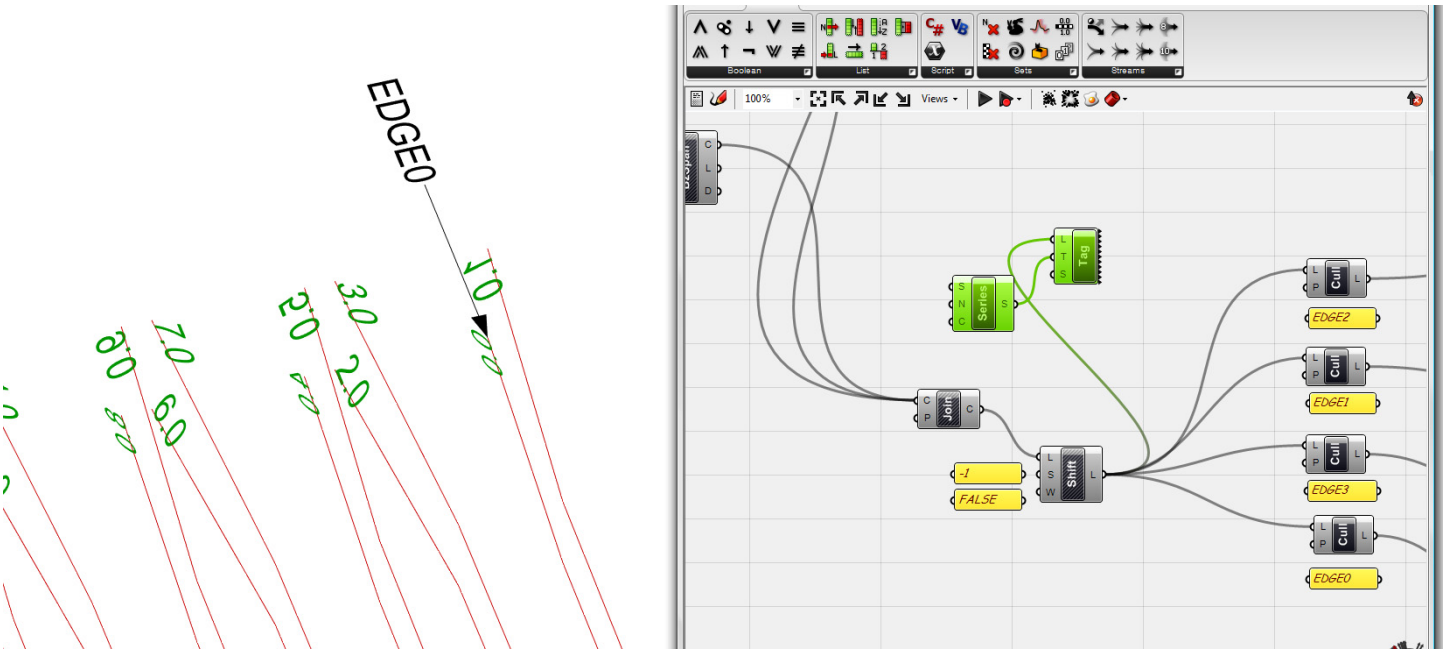
Bezier curves are fine too.



- Another problem here. Looks like point or bezier curve order problem.

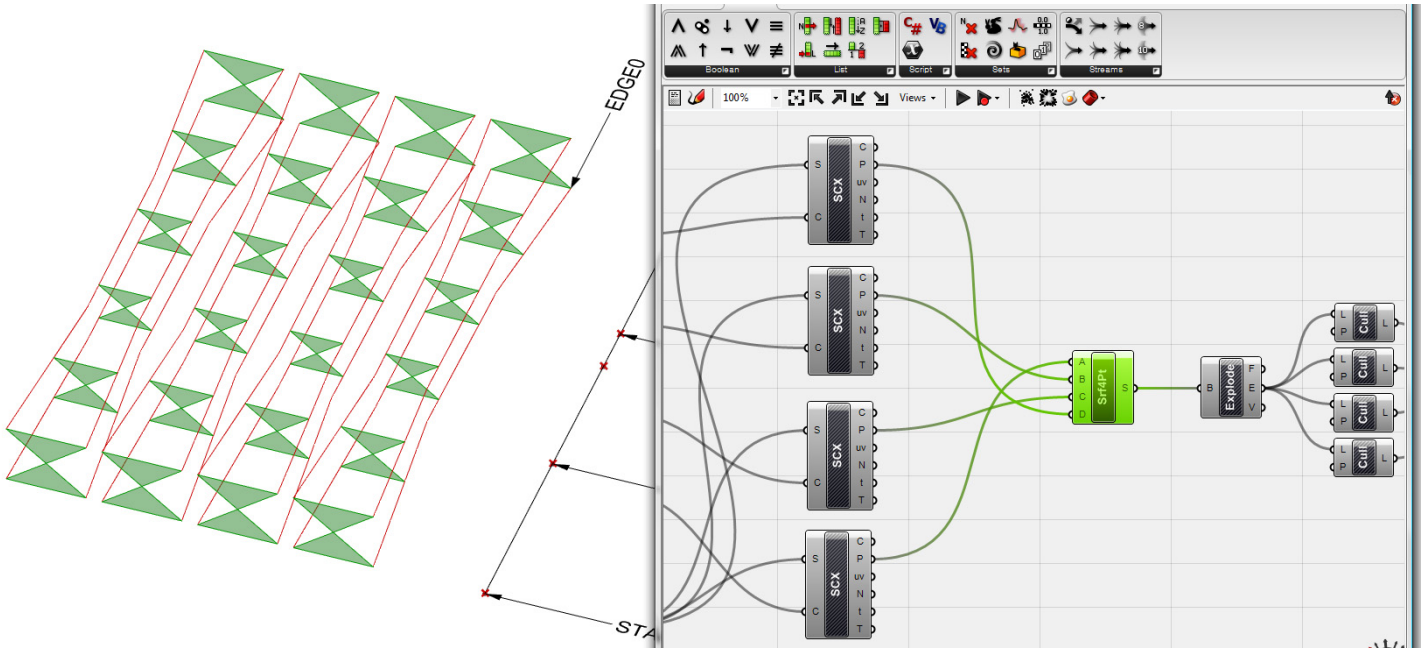


- Bezier curves are in an unexpected order. With just one box(brep), the order of bezier curves was not a problem at all. However, as we have more than one boxes now, it makes problem. So the first set of bezier curves is composed of three bezier curves which are in correct order(0,1,2) and one from the last set of bezier curves(11).

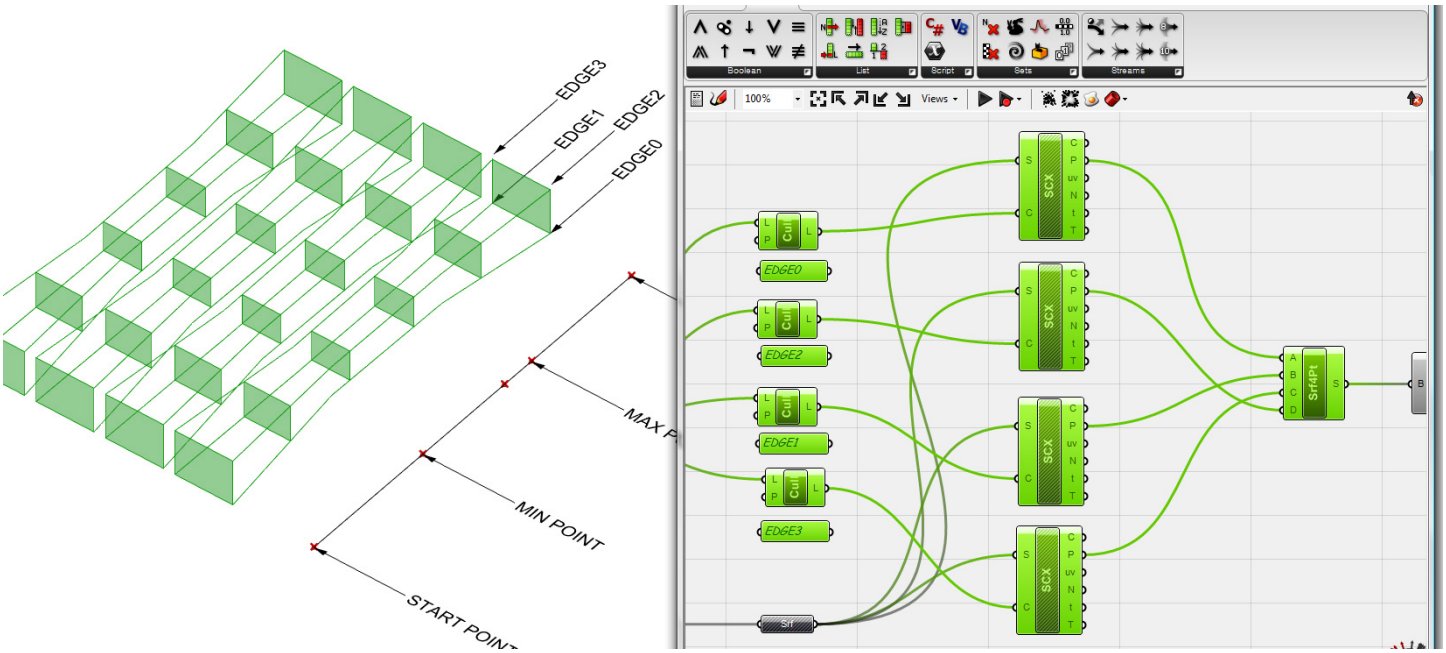


+ **Shift the order of bezier curves.**

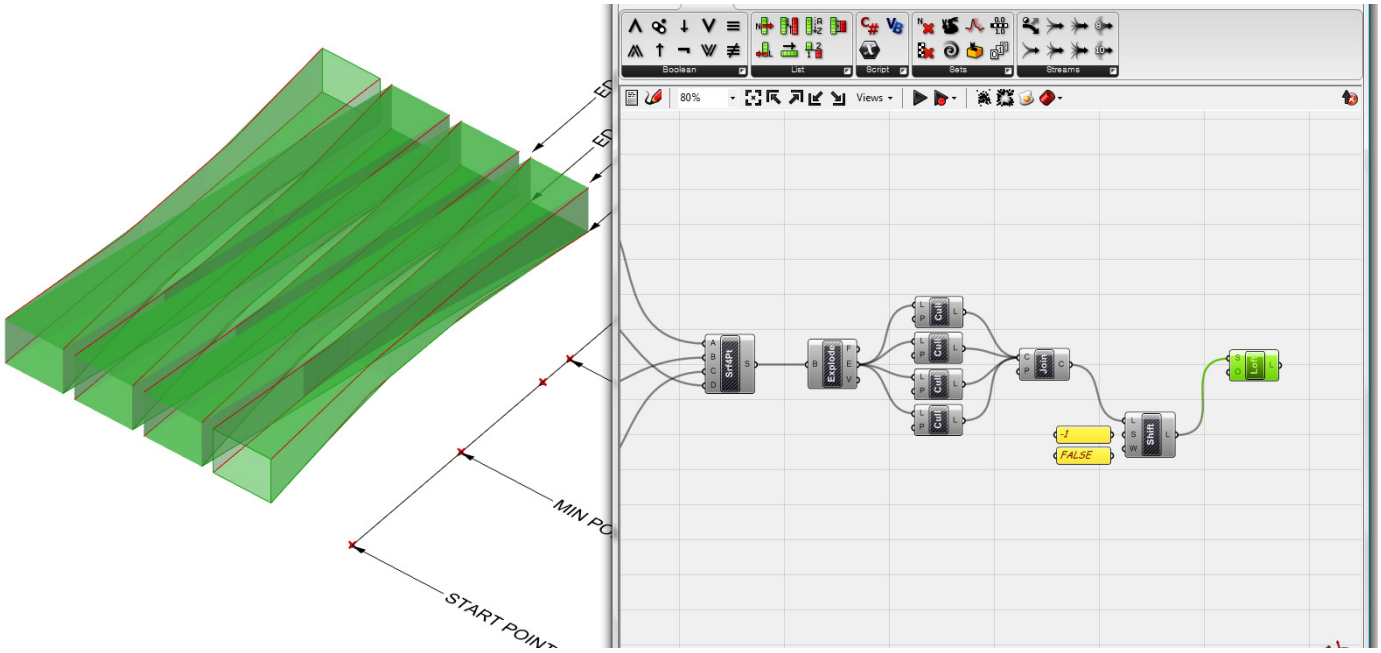
  Note that shift offset should be -1, and wrap value true.



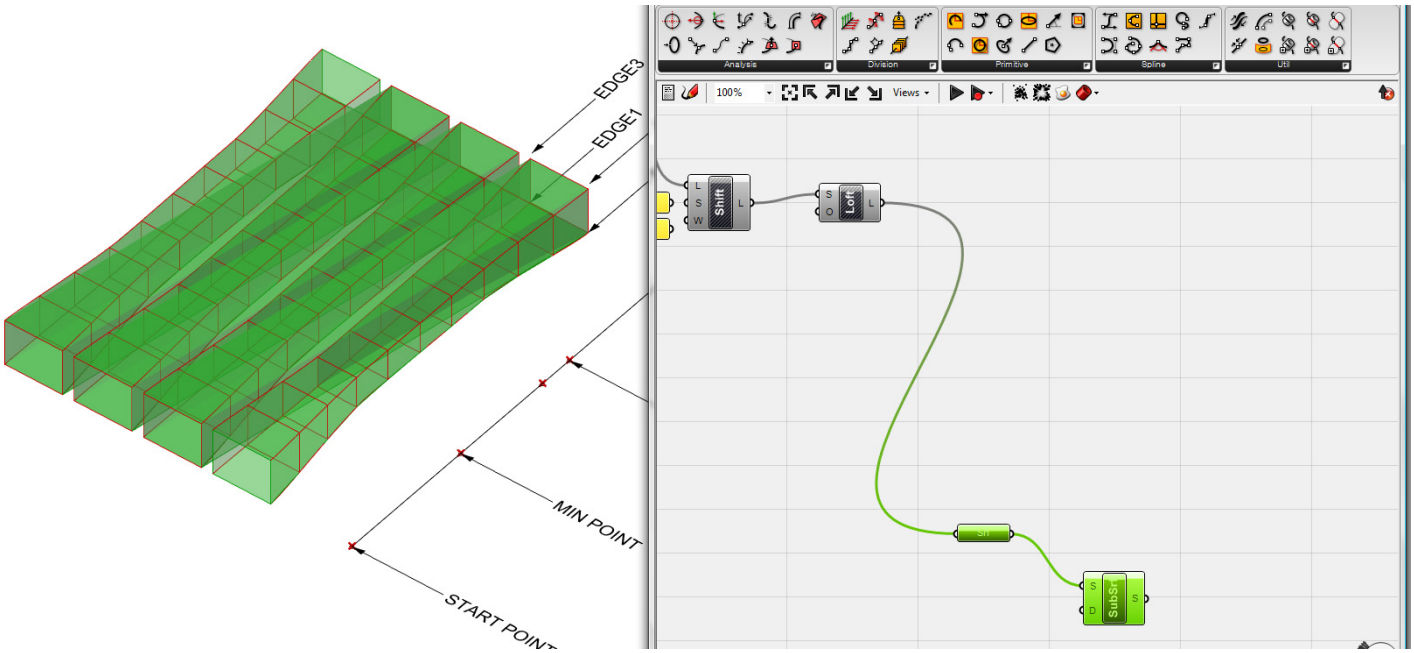
- Another problem here. It is also order problem.



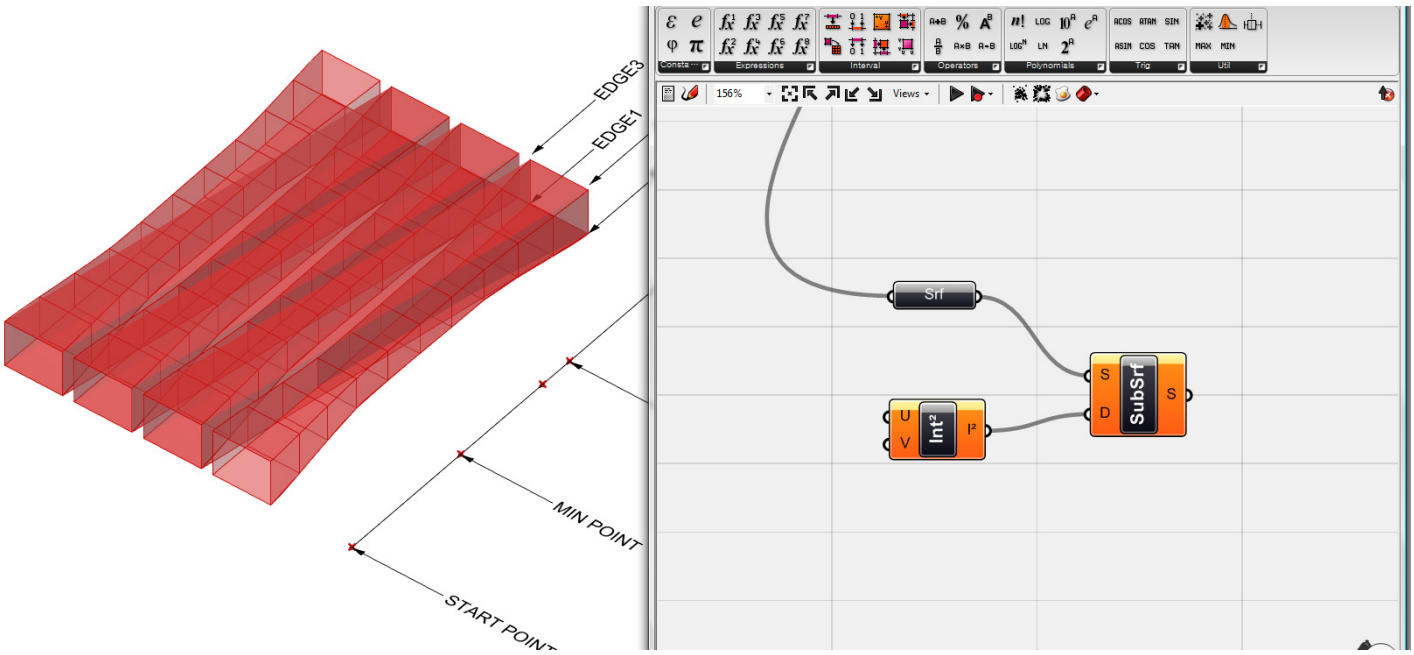
+ **Fix this by switching input points orders.**



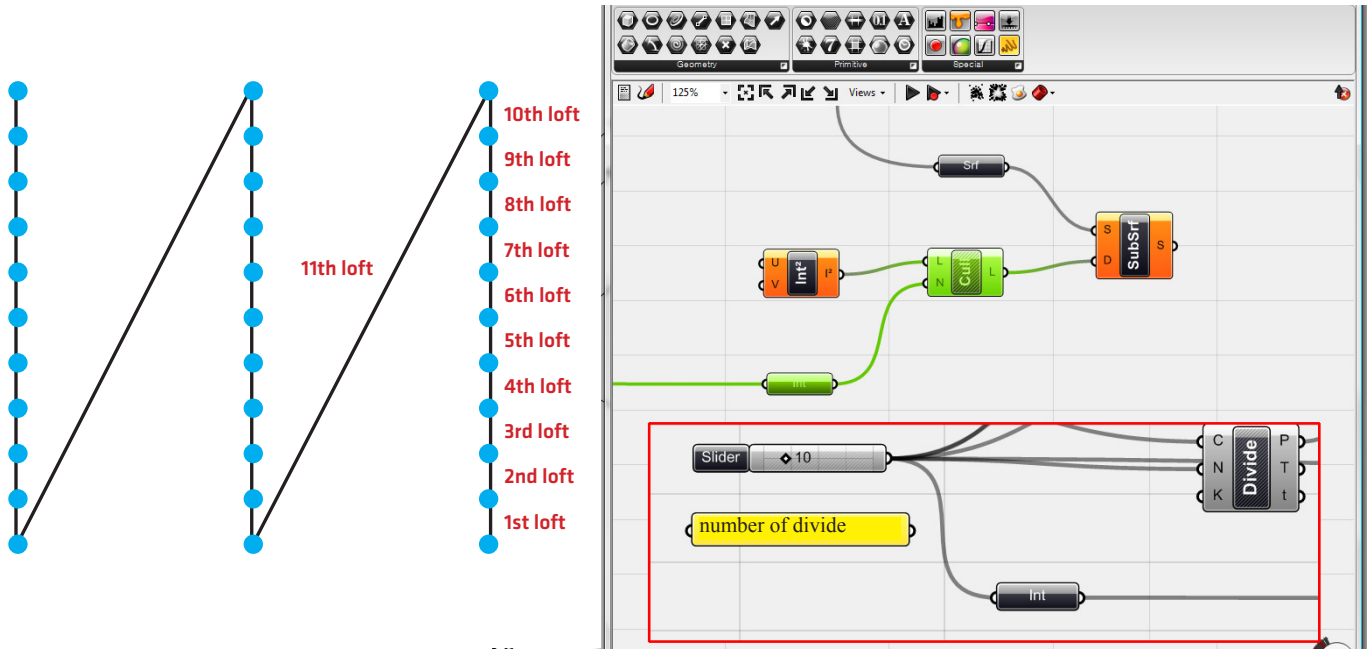
- Turn on the loft object. Another problem again. The problems is loft itself. The last section rectangle of the first box and the first section rectangle of the second box should be lofted.



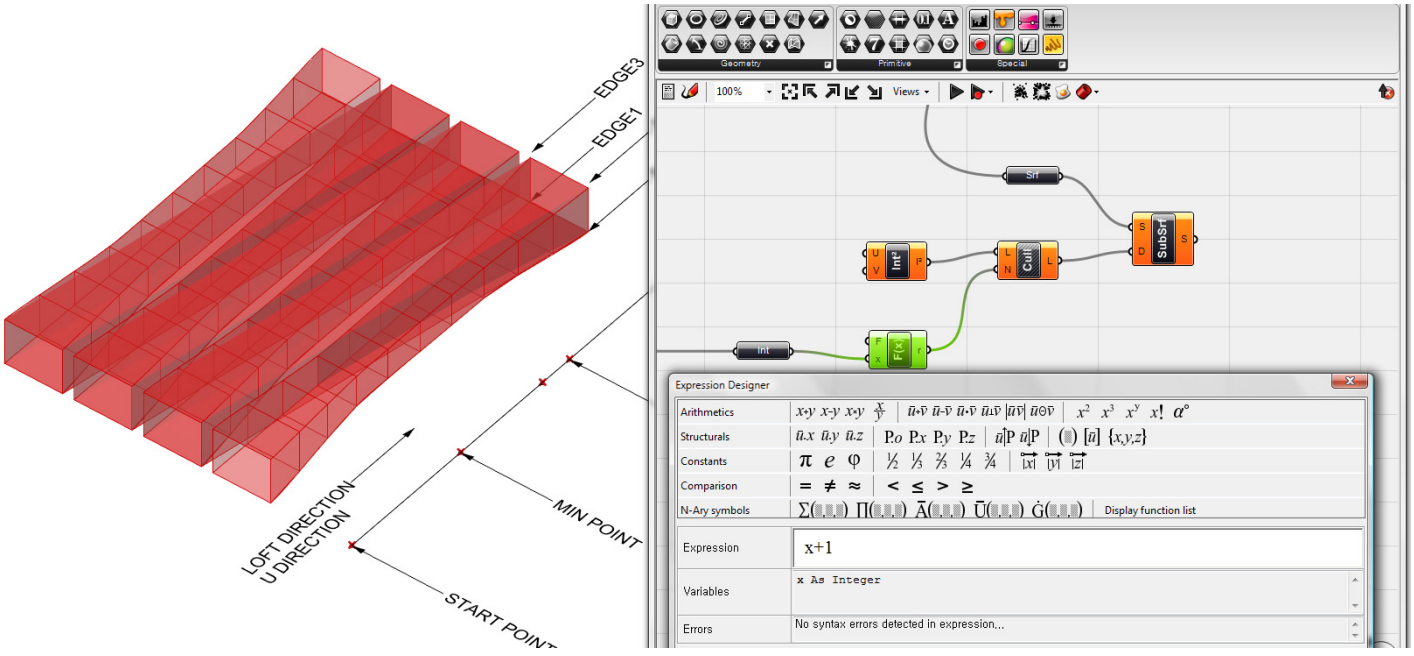
+ **To get rid of the unexpected diagonal loft shape, put Isotrim SubSrf object.**
  - What this object do is to make a subset surface based on given domain. We have no idea about domain right now but we do know that we should use this to delete diagonal surfaces.



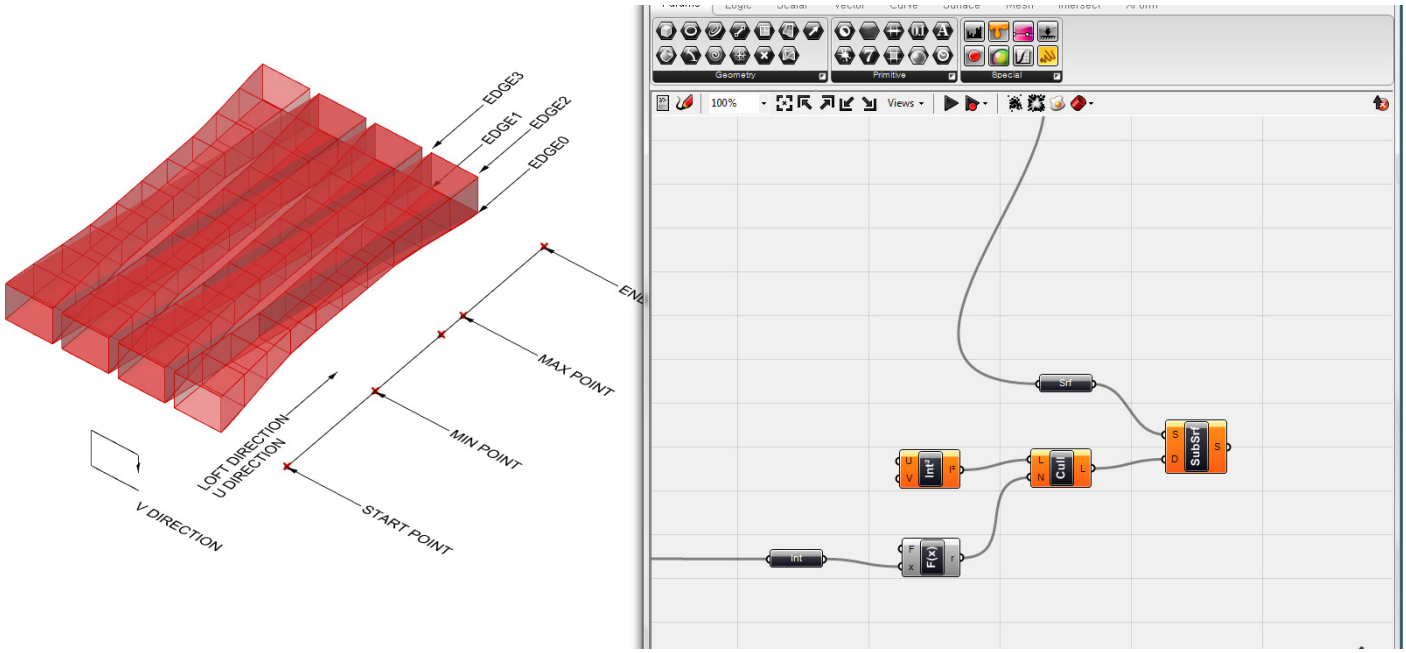
+ **Connect two dimensional domain to the subsrf object.**



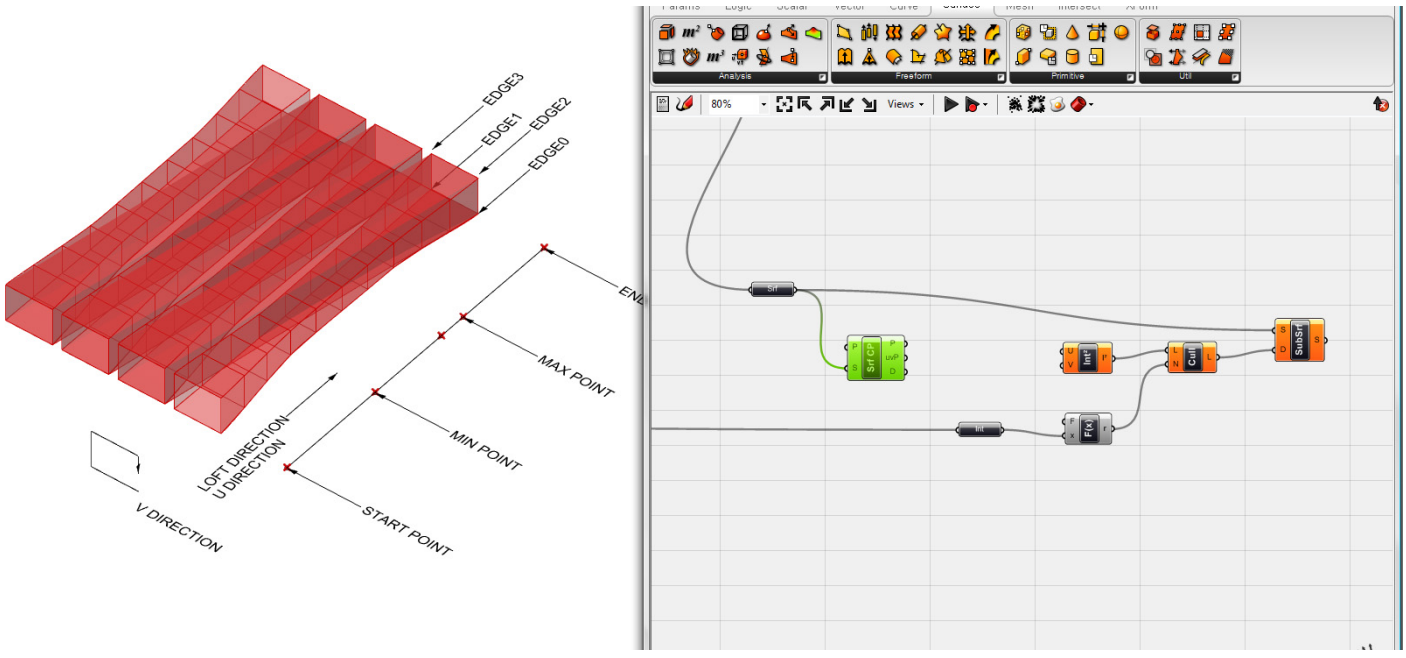
- We should remove every 11th loft(diagonal loft). Since we have number slide that decide the number of divide(in this case 10), we can simply remove 11th section using Cul Nth object.



+ **To remove (N+1)th loft, set 1-var-function object like this.**

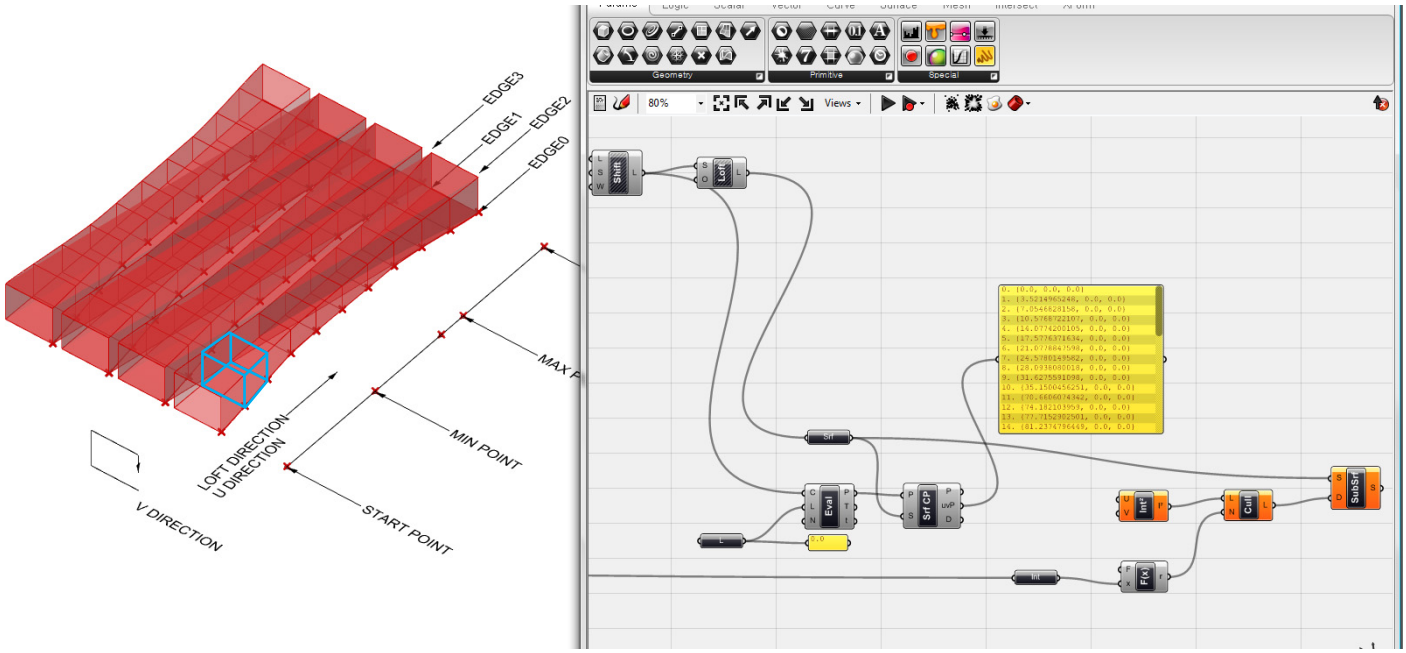


To determine domain, lets just take a look at the loft surface. When we loft, there exists three different direction, U, V and W. (I am not sure if W is correct term. Anyway...). U direction is lofting direction. V is perpendicular to U and in this kind of closed shape loft, V starts and ends at the same position. W is the distance from the resulting surface, which means when we dealing a certain point on a surface, we can just put W value as 0.



+ **Get uvw point on surface by SufCp object.**

  SufCp object finds the closest point on a surface by an external point. It will provide us the UV value of that closest point.

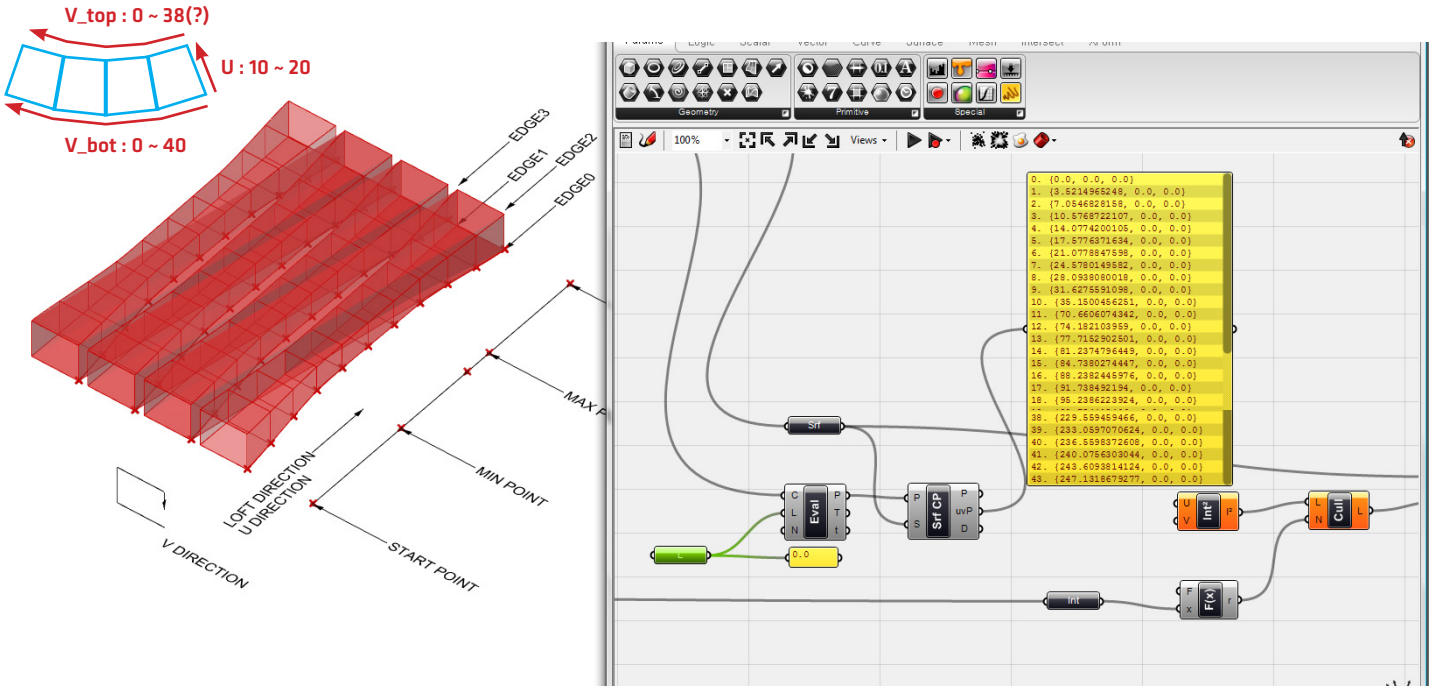


+ **To define an external point(we can use any point on section curves), evaluate every section profile curves.**

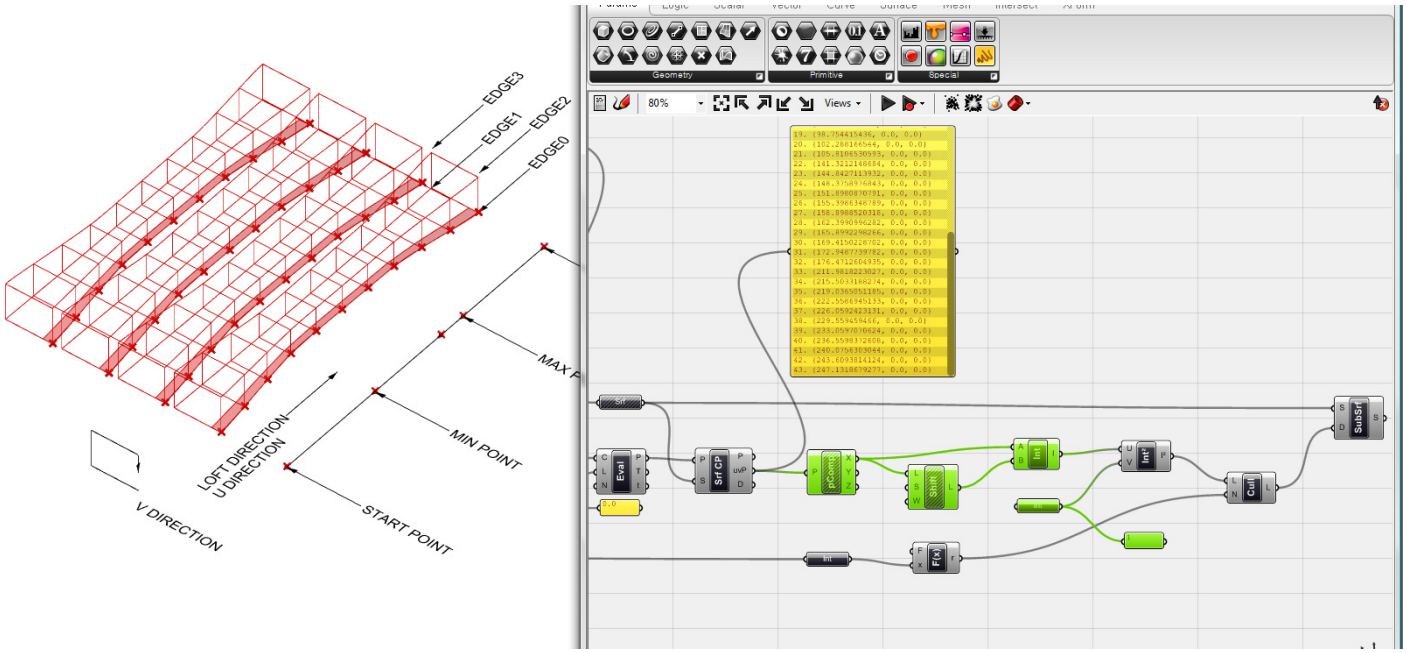
  In this case, we will take start point of profile section curve as our external point.

+ **Set Eval length factor as 0.**

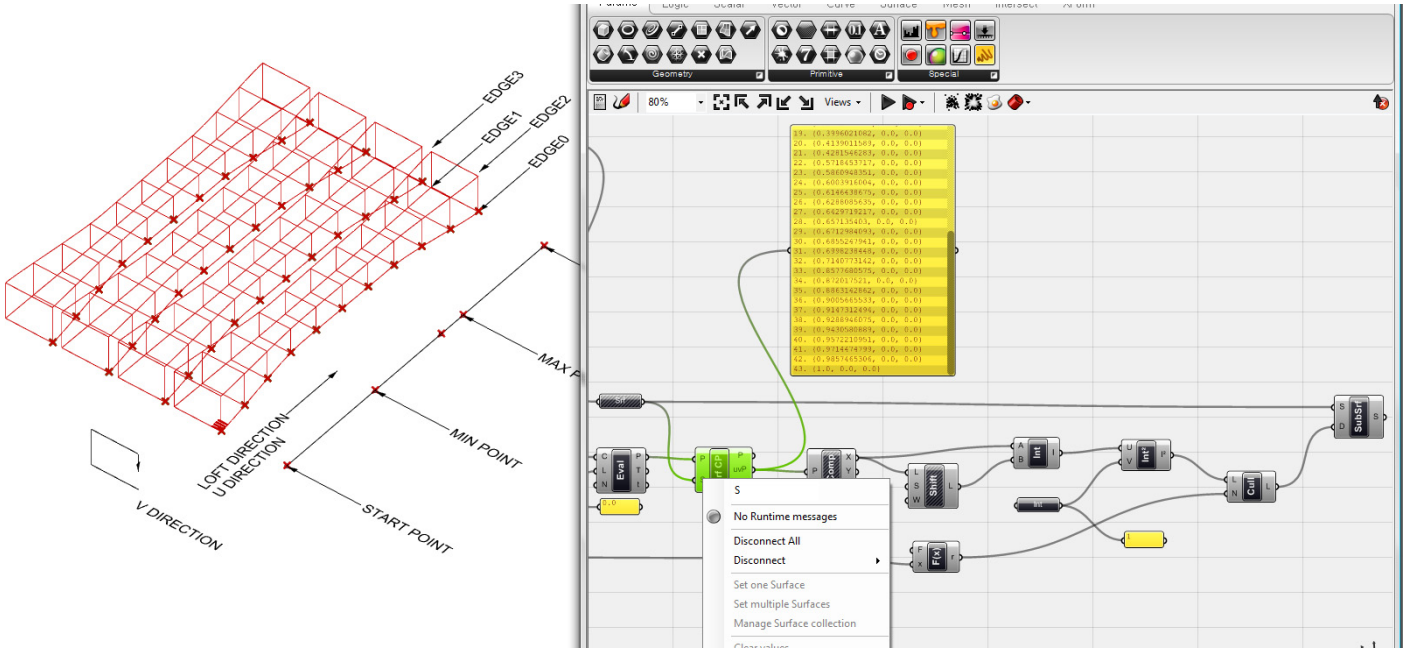
  By doing this, we defined start point of the domain. Also, we have UV coordinates for those points.



- Note that uv point coordinates are in real scale. This real scale value will not be a problem with U direction, because we can set series of U domains that start from the last point of (N-1)th loft segment to the first point of (N)th loft segment, and we have those values in real scale.
- However, when it comes to the case of V direction, it is a different story, since every section has different V value in real scale. That is because unfolded shape is trapezoidal instead of rectangle.

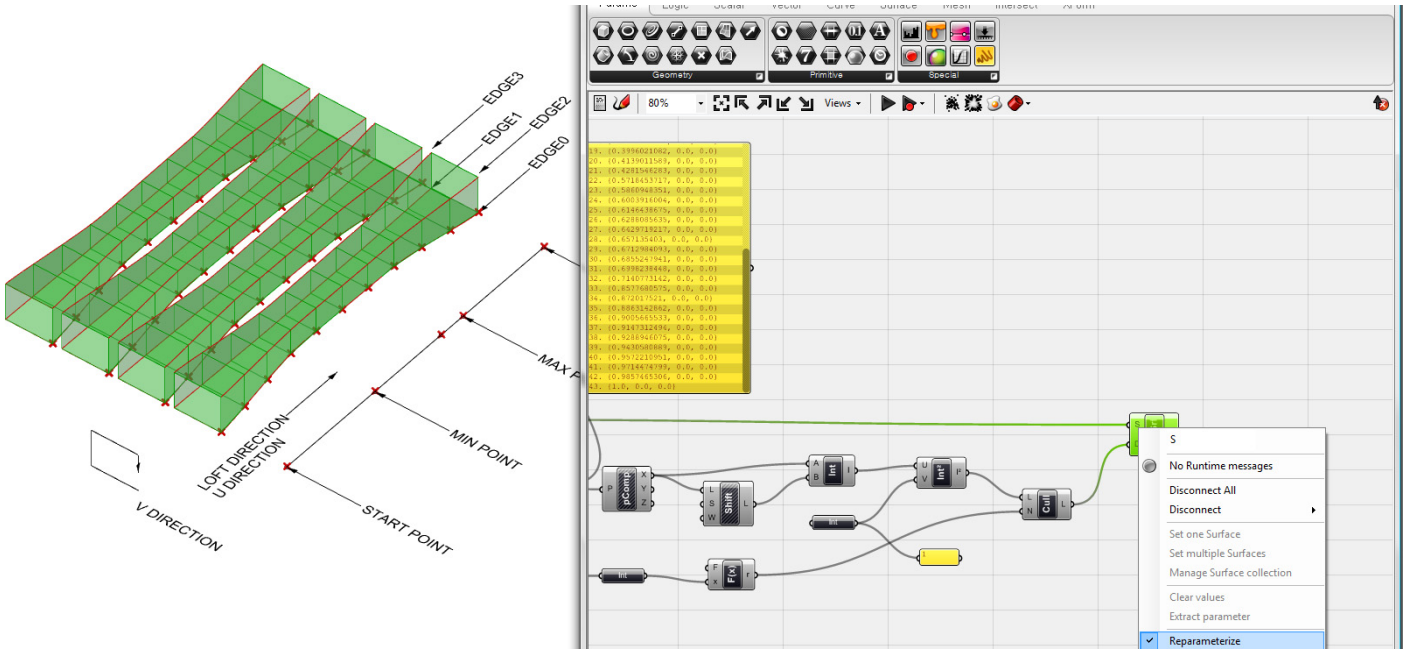


+ **To define the U domain, put Decompose XYZ object to extract U values only.**
+ **Generate U domain using Shift Item object and 1-d-interval object.**
+ **Connect it to U value of 2-d-interval object.**
+ **Put 1 as V value for now.**
  - What happened? U domain works perfect. However, V domain works partial, I mean, from zero to one in real scale.



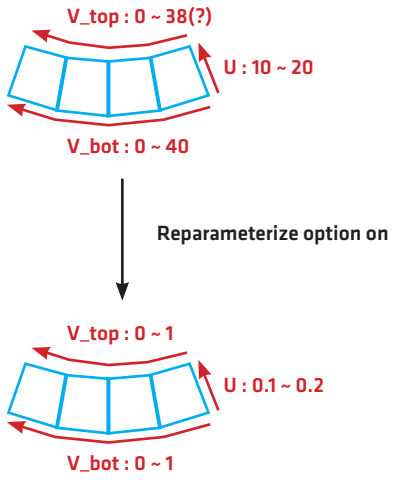
+ **Select SrfCp object and then right click on S item, then check reparameterize option.**

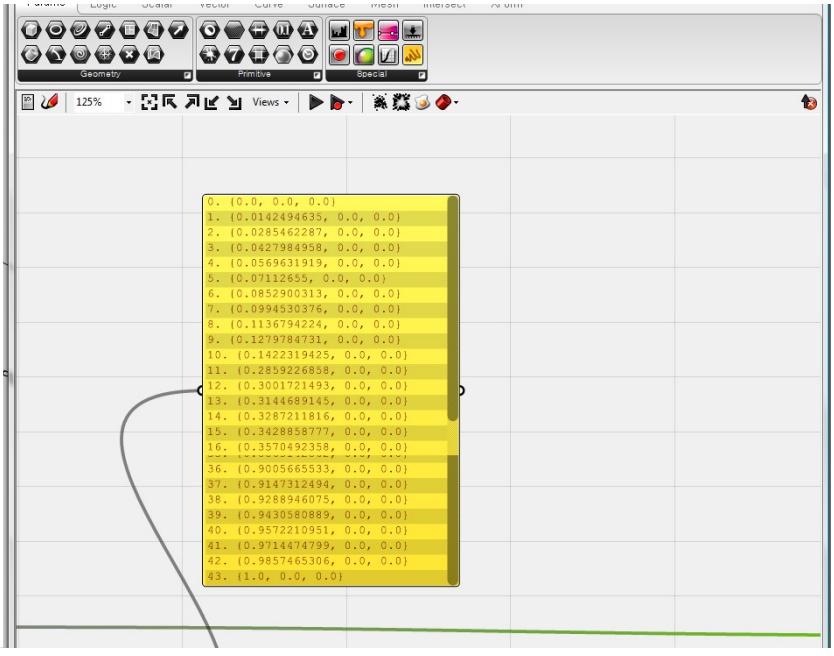
  This will unitize input value. For example, let's say that we have input value which start from 100 to 150. Then this reparameterize option will convert this value from 0 to 1.
  - After reparameterize option on, it still looks weird. That is because we just reparameterized only domain value.



- We also have to reparameterize surface input.

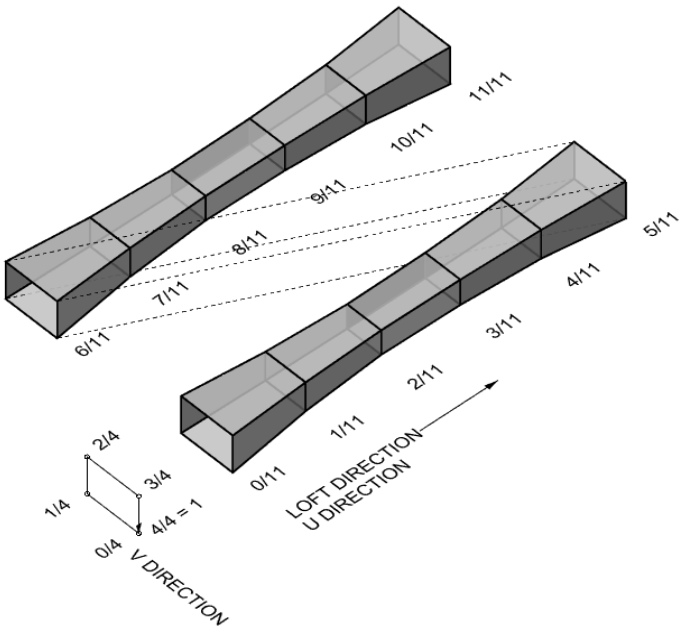
+ **Select Isotrim SubSrf object and then right click on S item, then check reparameterize option.**





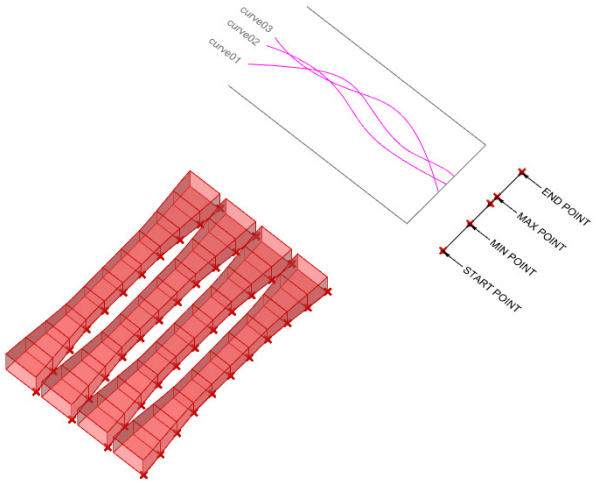
Reparameteriz option unitized UV input value like above.

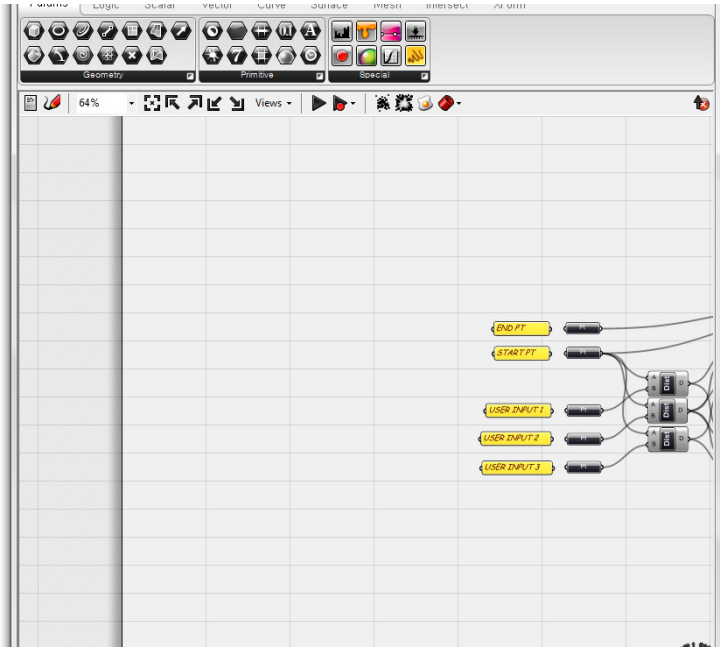
- Note that we have U value from 0.1 to 0.2 since this strip is just a part of several strips. In the big picture, U value also ranges from 0 to 1.



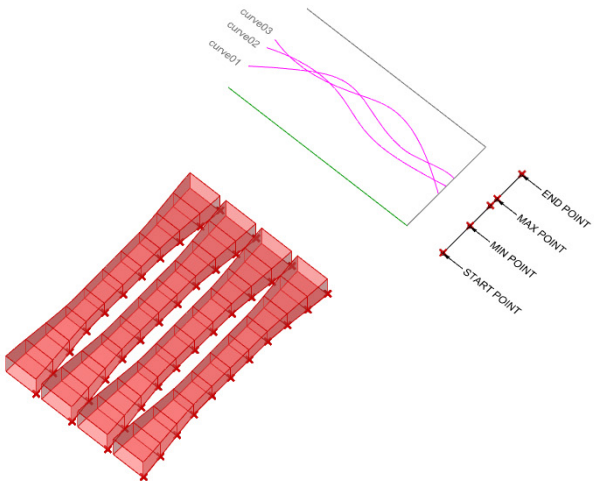
For better understanding, see the diagram.

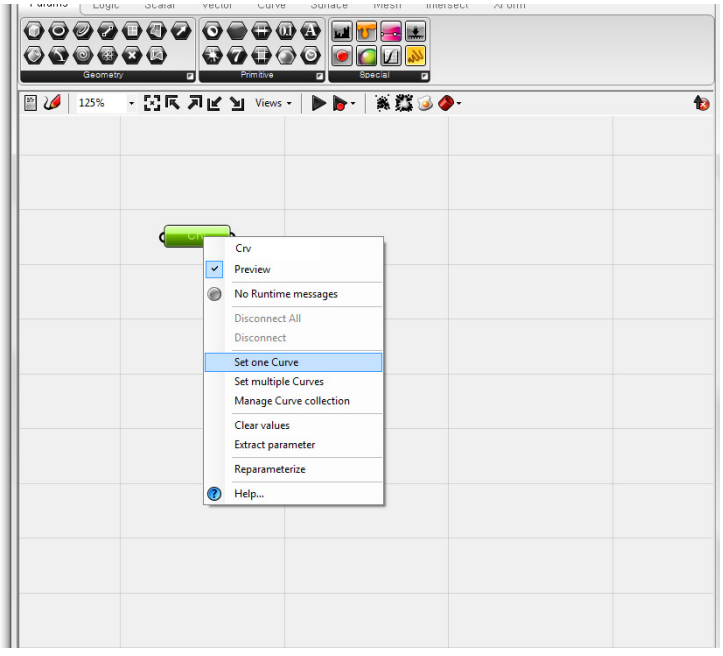
- Let's say that we have one loft surface which is composed of 11 loft segments. As shown in the diagram, overall u and v value range from 0 to 1 since we parameterized surface input. For segment 6 between 5th and 6th section profile, u value starts with 5/11 and ends with 6/11 while v value ranges from 0 to 1. That is why we remove 6th u value with cull pattern object. And since all of the segment have 0 to 1 v value, we just input 1 as v value.



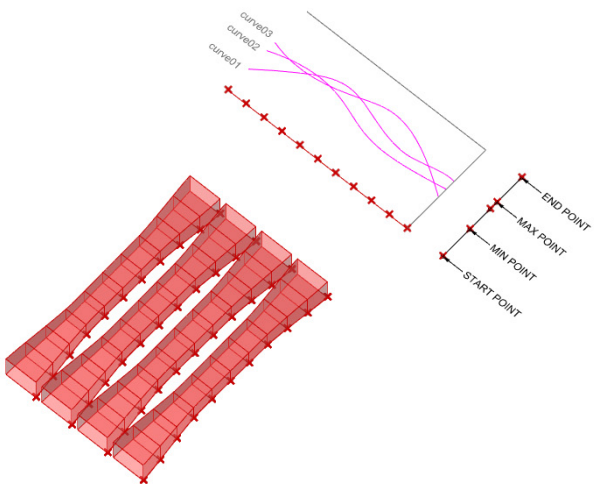


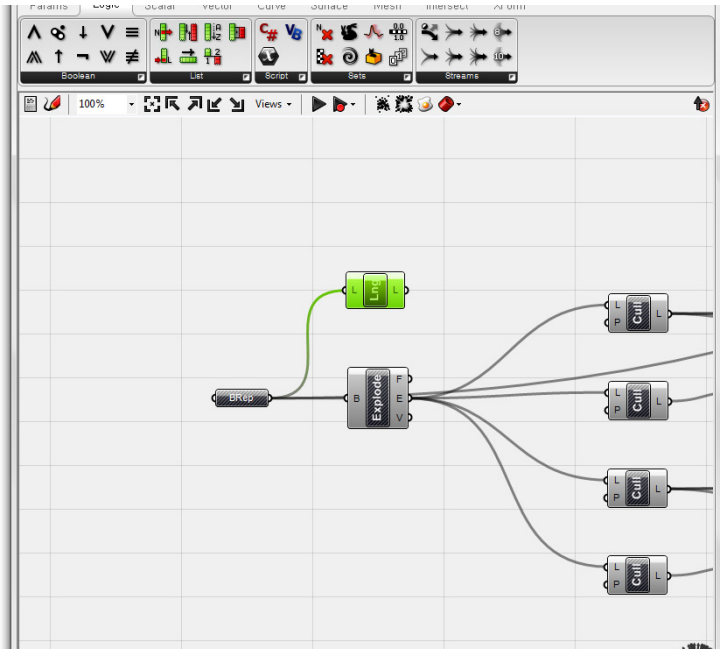
- In the stpe 3, we will learn how to control user input using external graph.
- In the previous step, we had only one set of user input variable; Max, mid and min point. Note that start and end points are not variables. They are kind of fixed value that just define overall distance.
- What we are going to do is to use three magenta curves as our user input instead of three points.





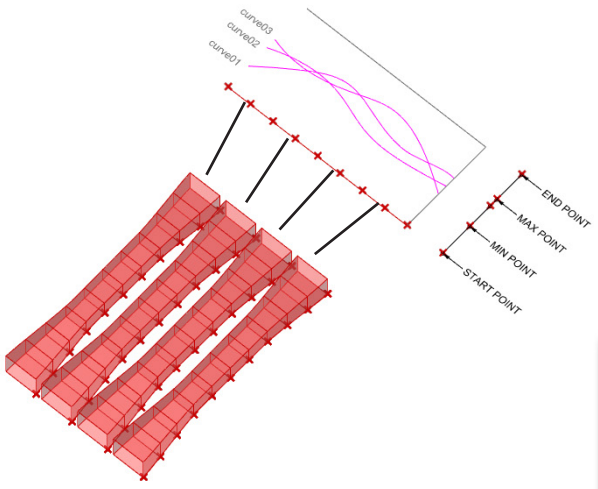
+ **Connect start line (green) with curve parameter.**

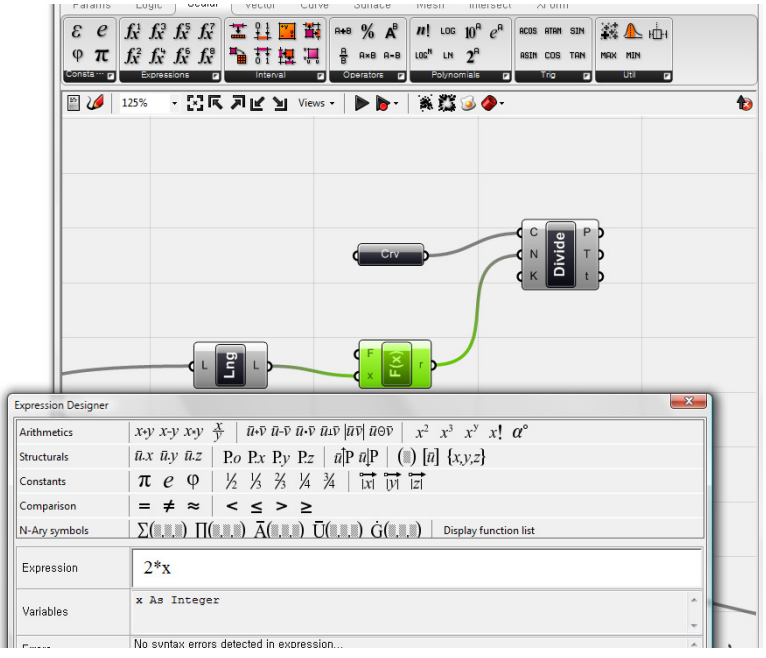




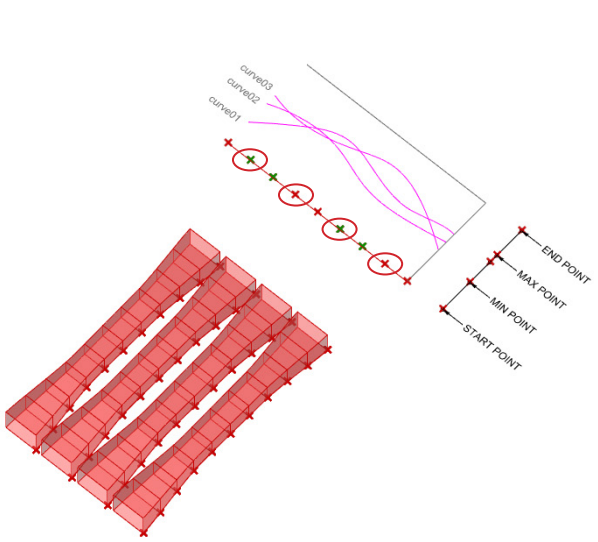
To divide the line, we need to know the number of box geometry.

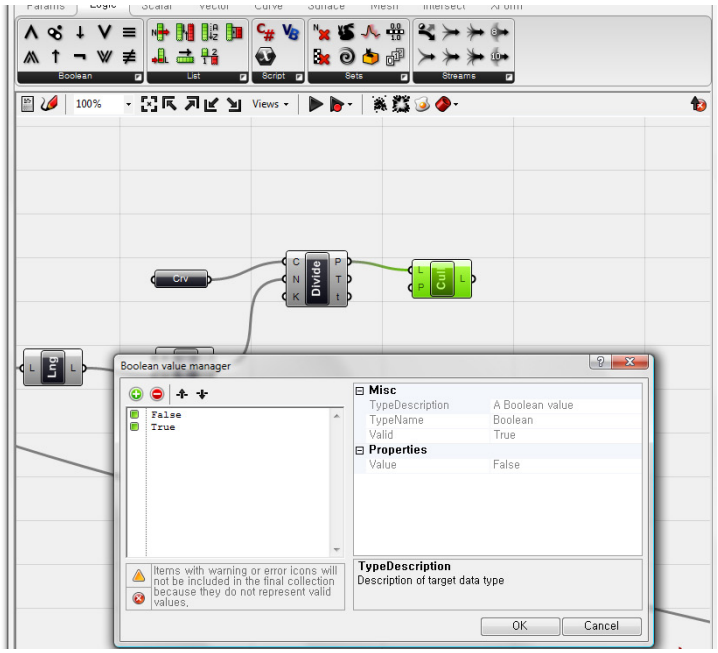
+ **Attach List Length object to brep object.**



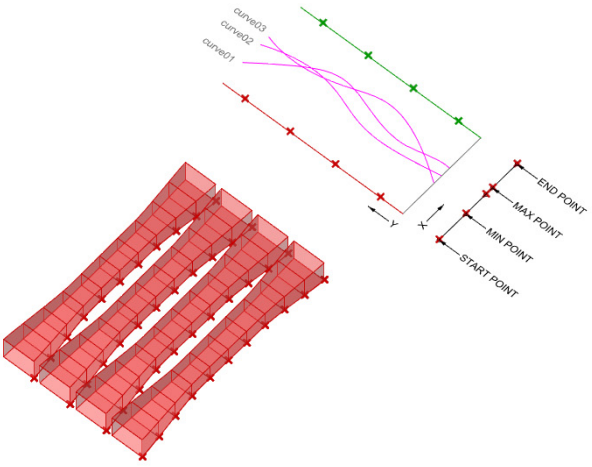


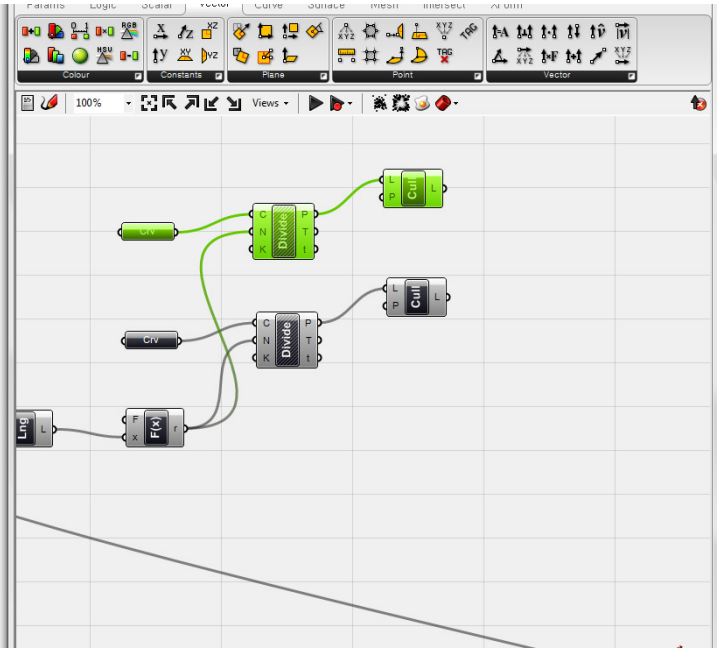
+ **To have one to one matching, use 1-var-function object like above.**



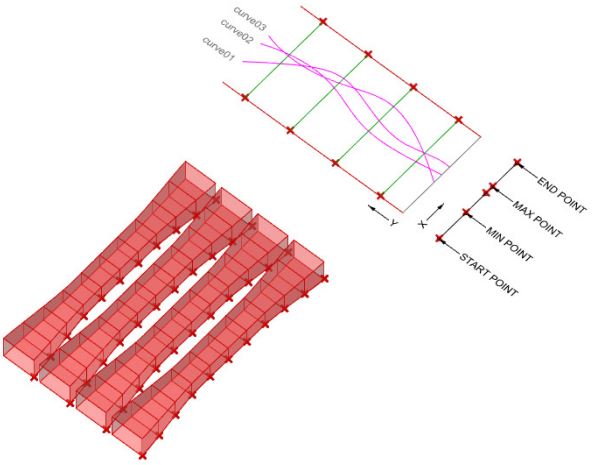


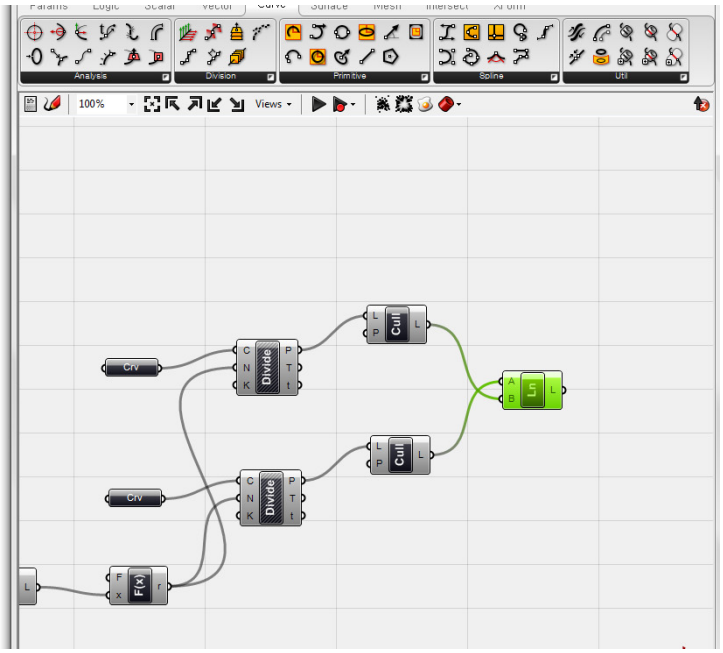
+ **Cull pattern object with F-T cull pattern.**



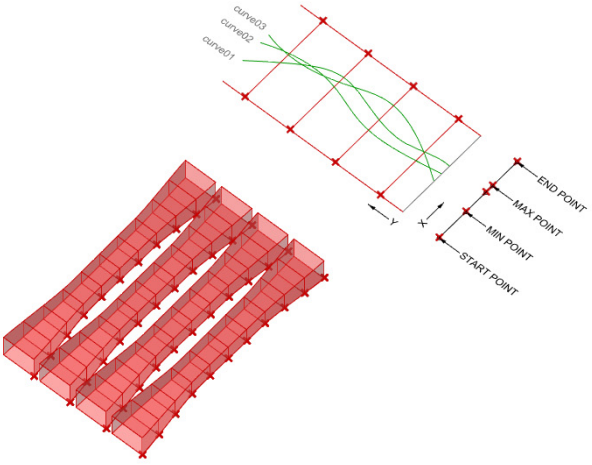


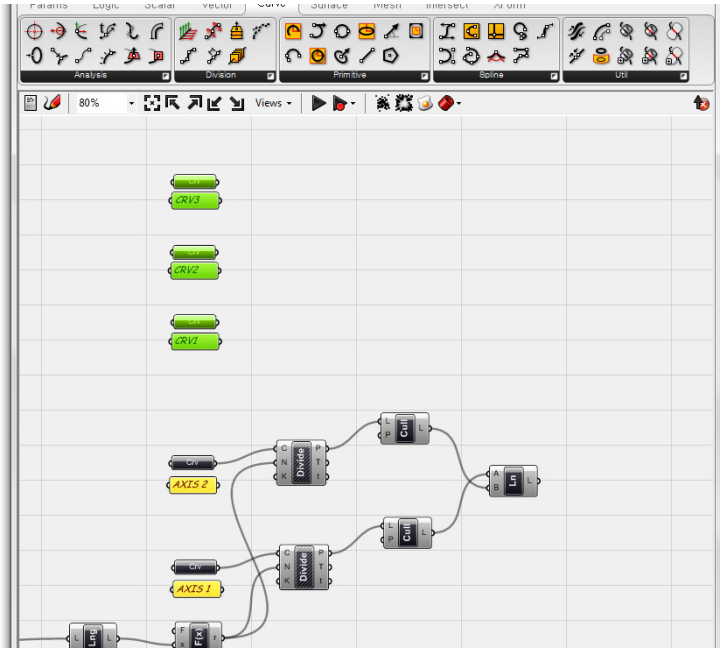
+ **Do the same thing for the end line.**



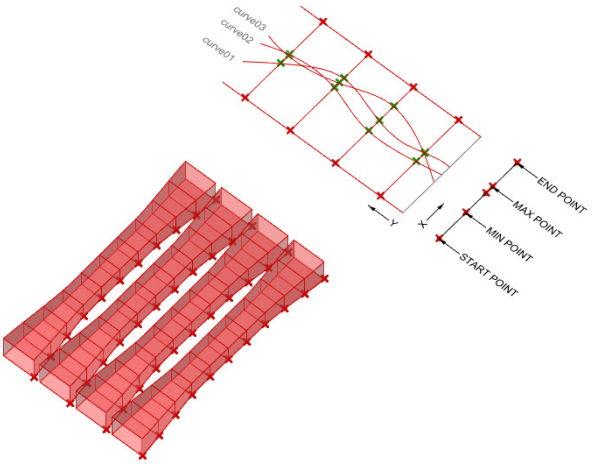


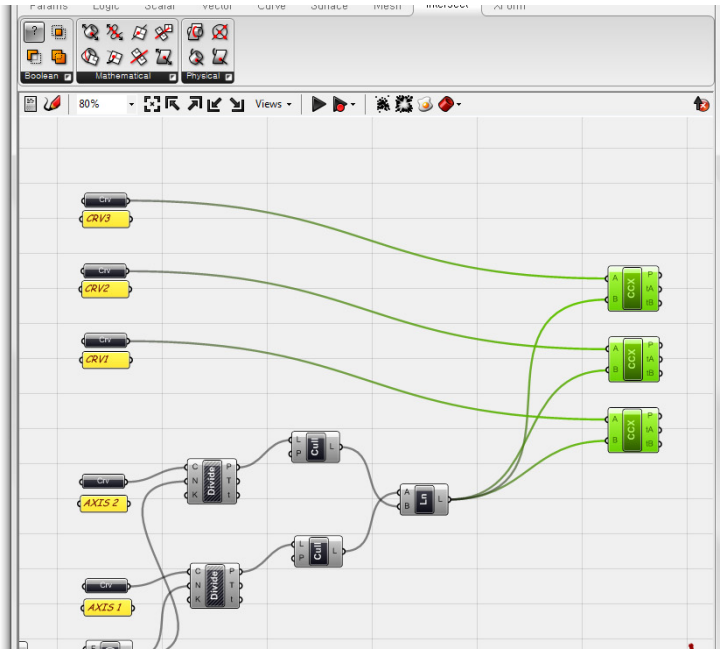
+ **Get lines using line object between two sets of points.**



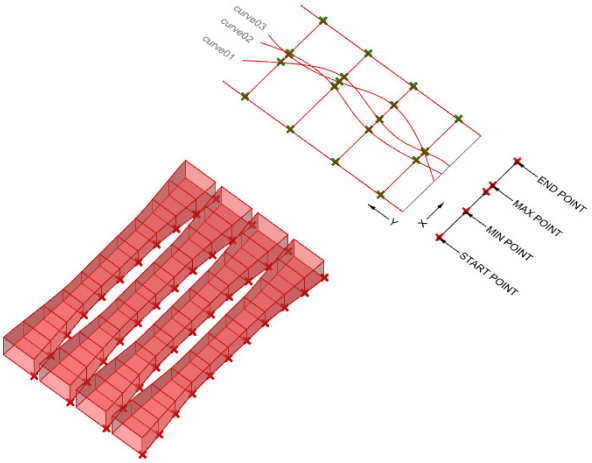


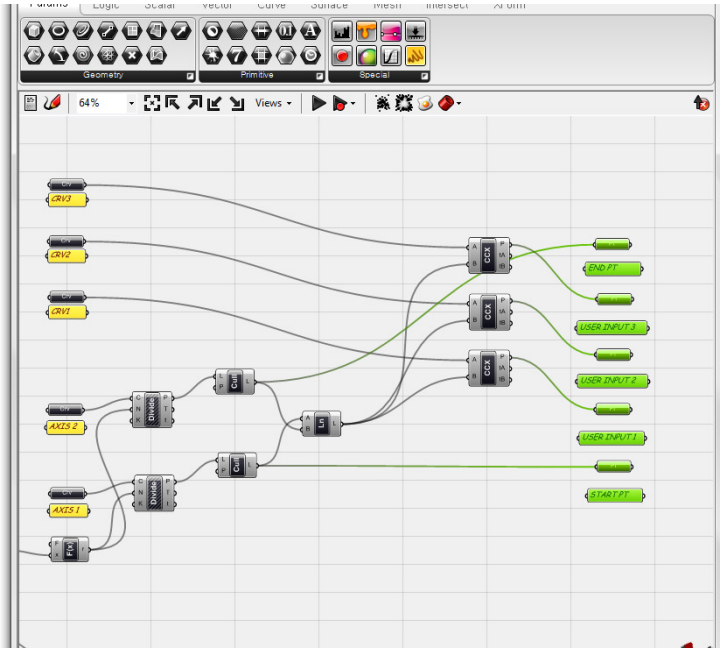
+ **Connect 3 curves with crv objects.**





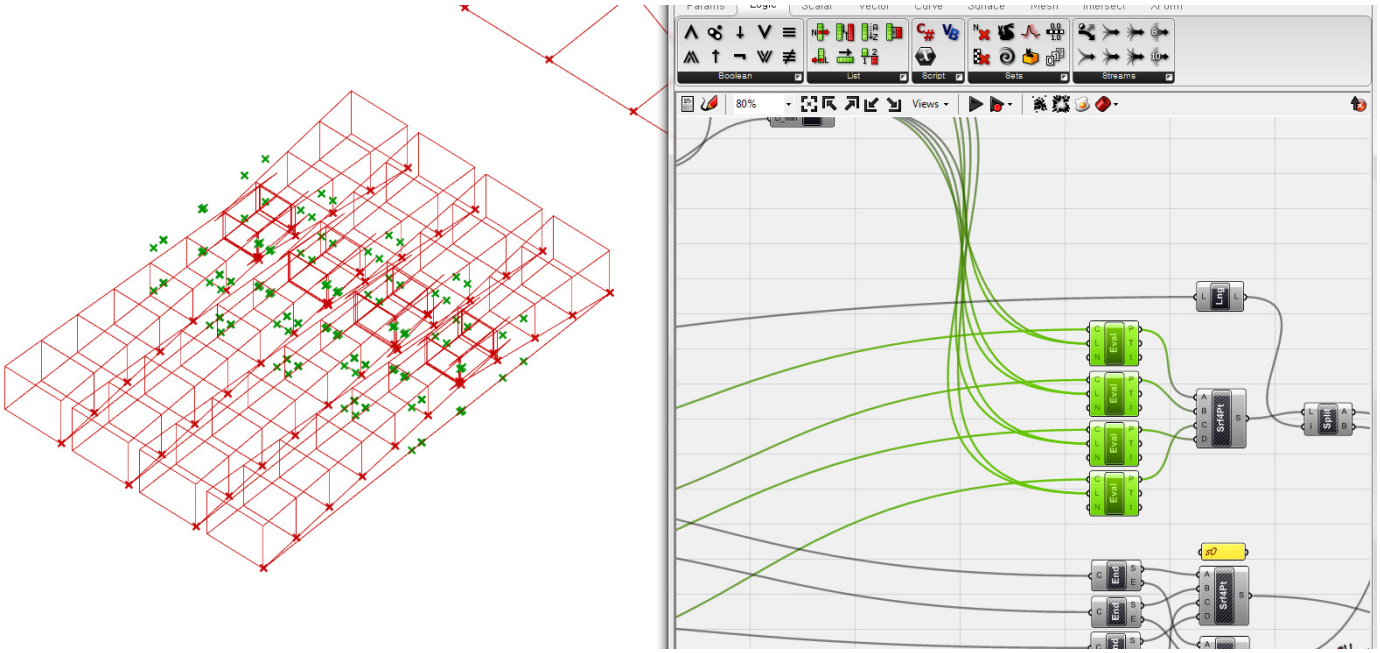
+ **Get intersection points.**



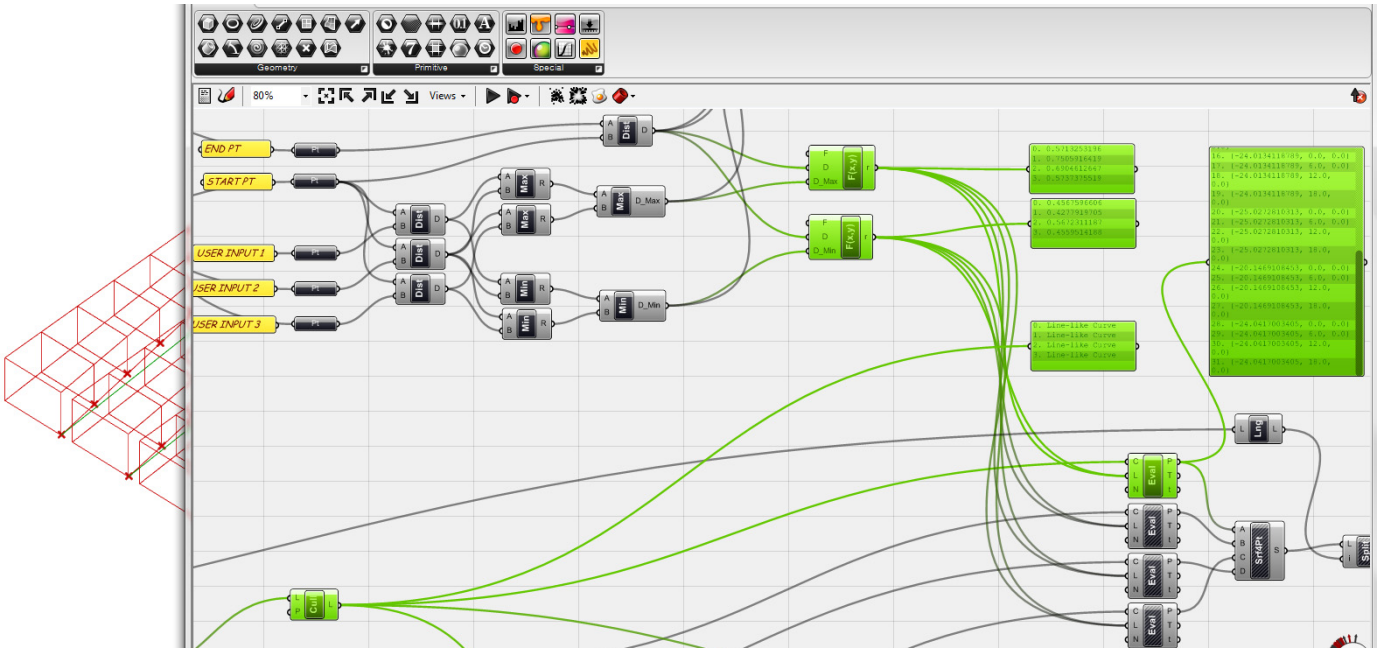


Now we have 5 points per line.

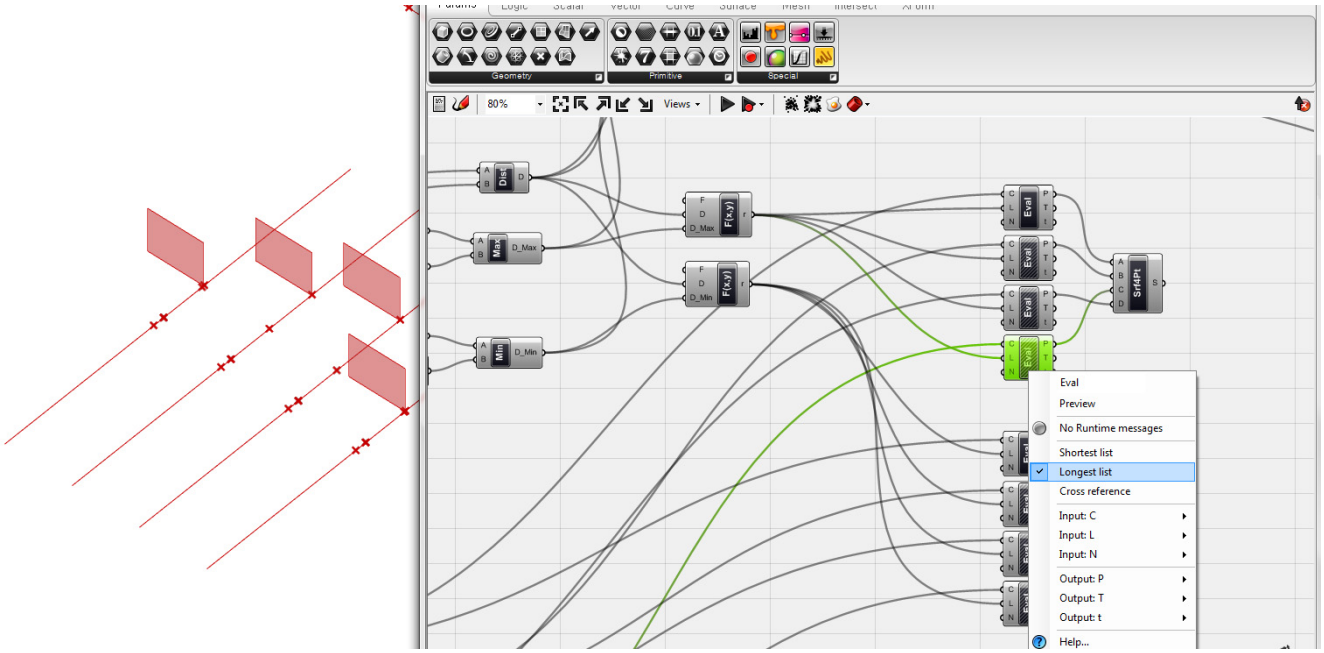
+ **Use those 5 points as our initial input as we did in the step 01. (see page step01_01)**



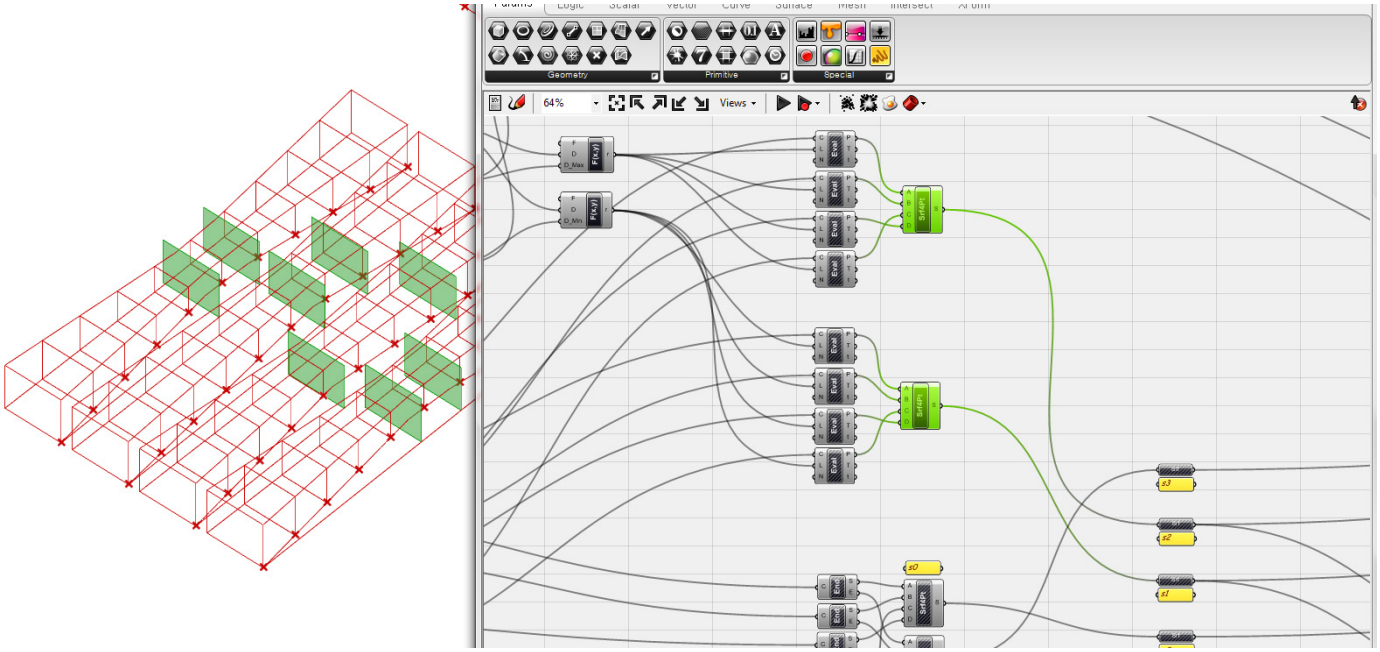
+ **Connected 5 points to the initial input points parameter in the step01.**
  - We have a problem.



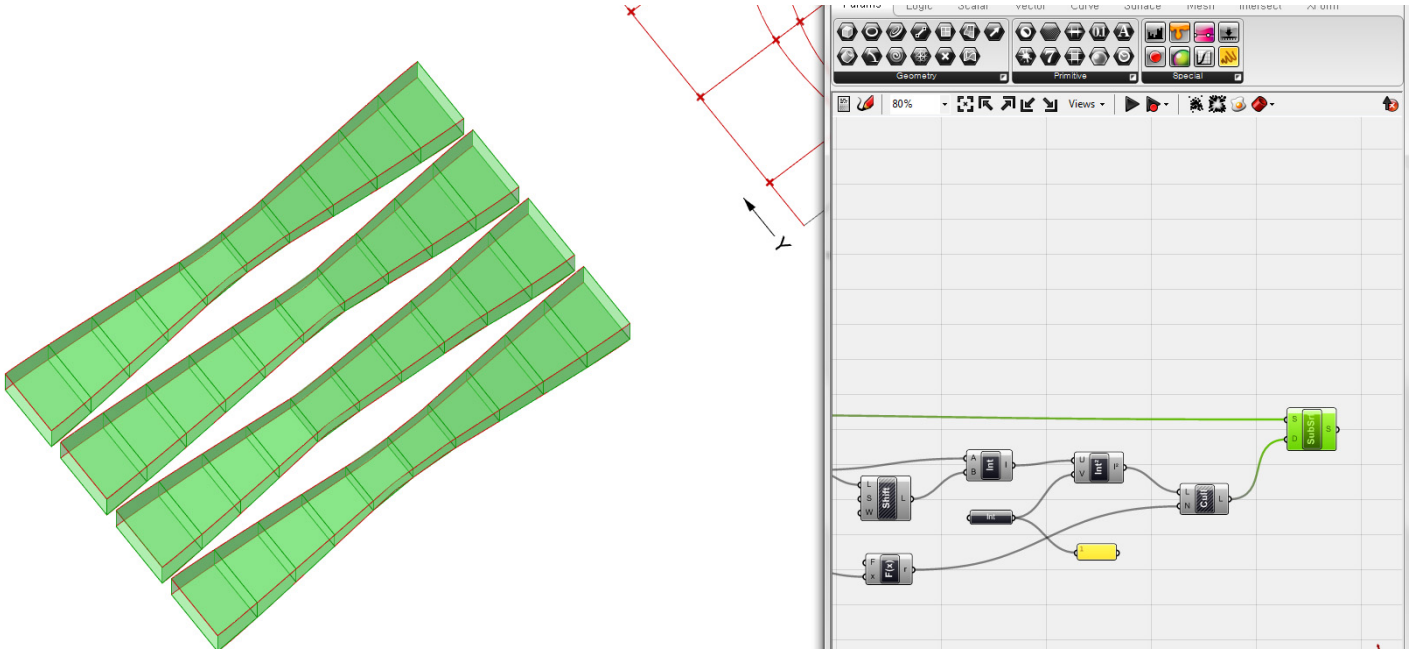
- Remember that we have had this data matching issues in both step 1 and 2.
- In step one, we didn't have to care about data matching, since at that time we just had to divide one edge with two distance parameters.
- In step two, because we had to divide multiple edges with same distance parameter condition, we set data matching as crossing. See page step02_03.
- But in this step, we have multiple input both for edges and distance parameters. And because of crossing data matching, we now have more points than we need.



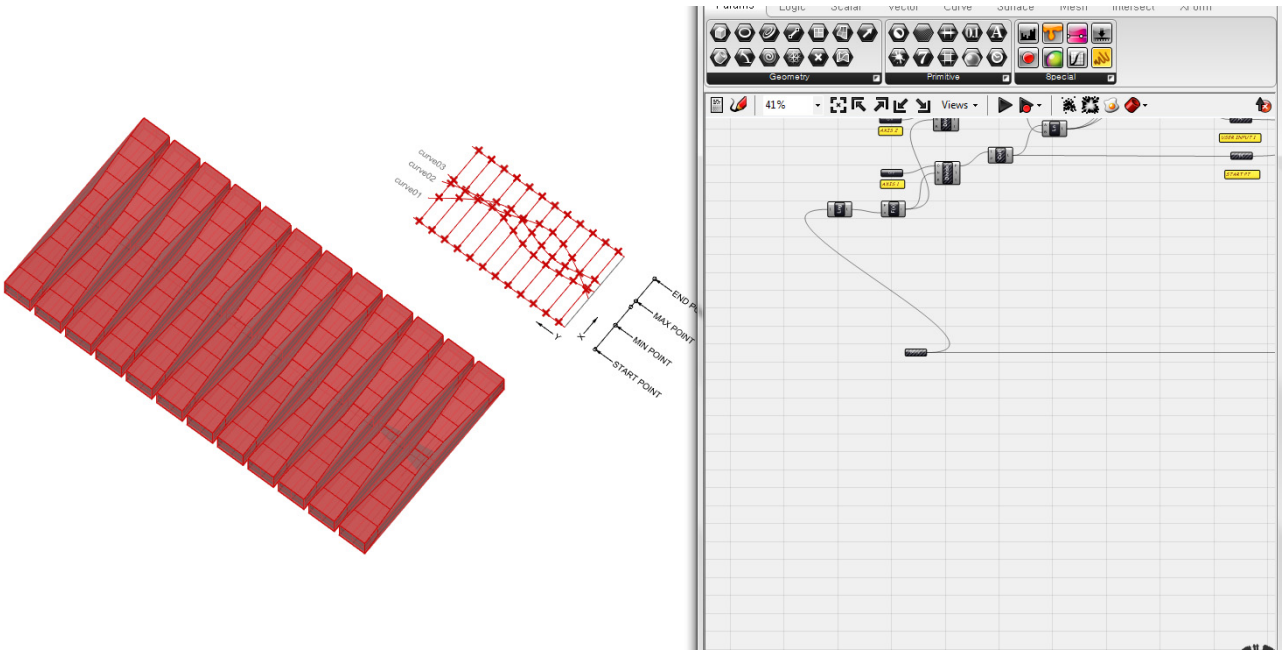
+ **Divide middle surfaces physically into two groups. One for Max position and the other for min position.**
+ **Now, every Eval Crv object has the same input number both for edges and distance factors. So we can set data matching back to longest.**



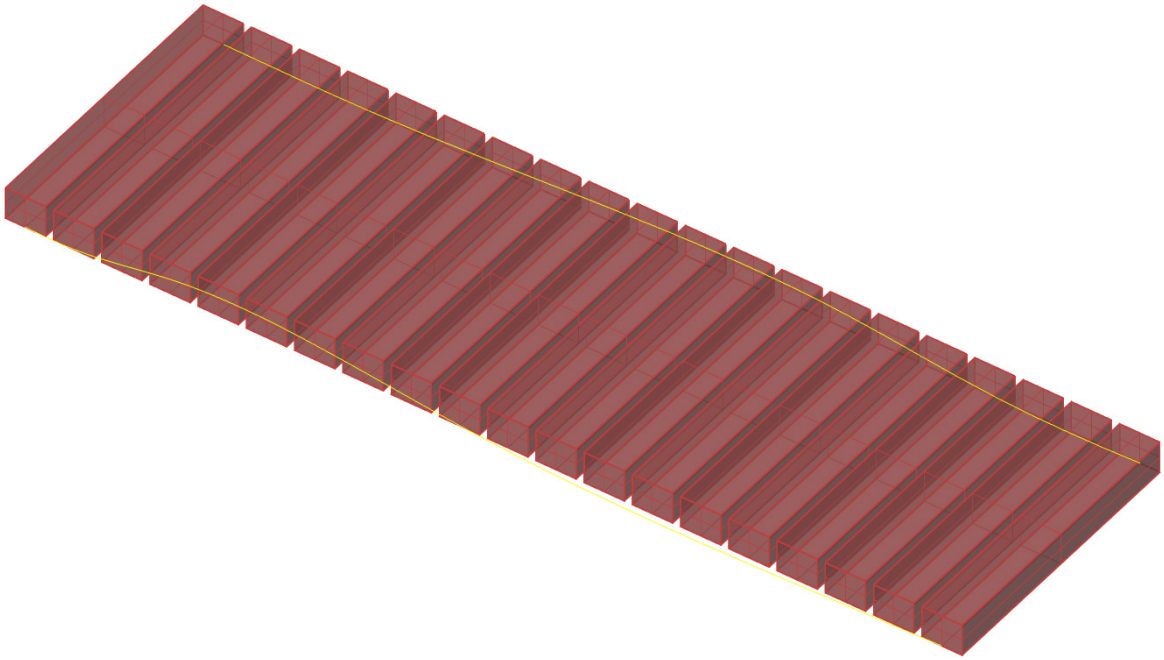
Works fine.



Turn on the loft surface and it looks good.

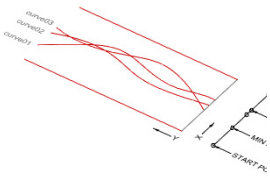


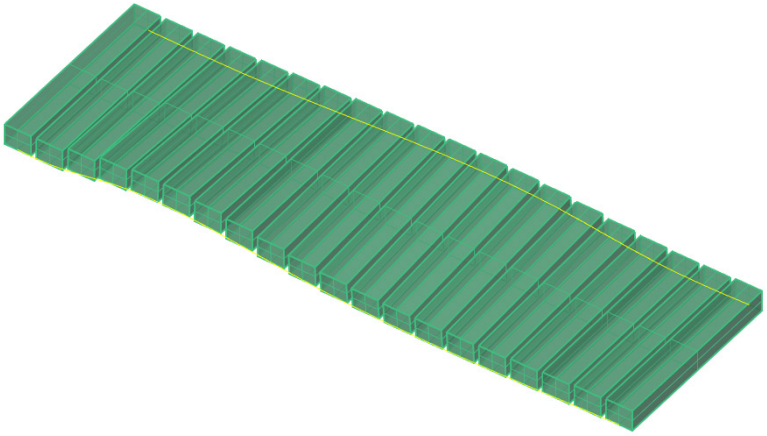
You can increase the number of input boxes like this. Then grasshopper automatically increase the number of input points.
You also can modify input curves as you want.



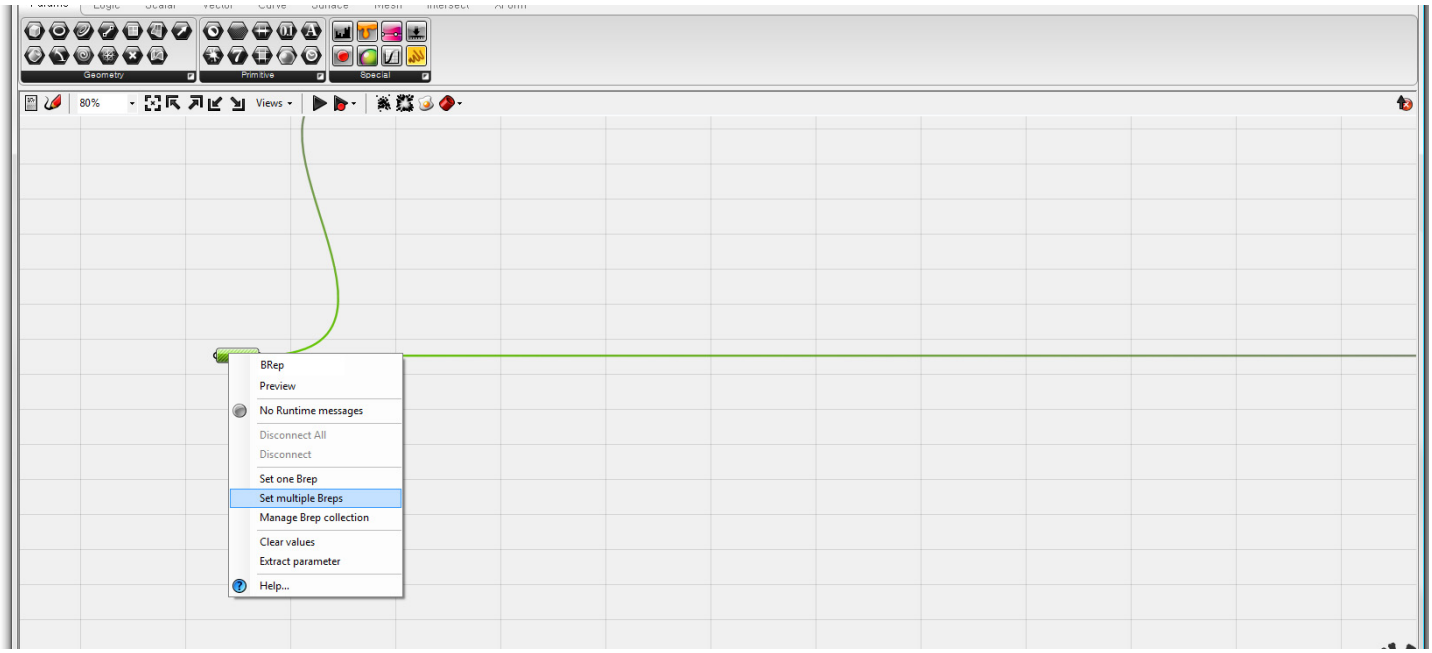
- This step is actually not about parametric idea. Actually, I also did this step using grasshopper, and if you successfully followed all the steps above, I am sure that you can do something parametric for this step by yourself.

  Anyway, I will modify initial box arrangement along with two 3 dimensional curves(yellow).

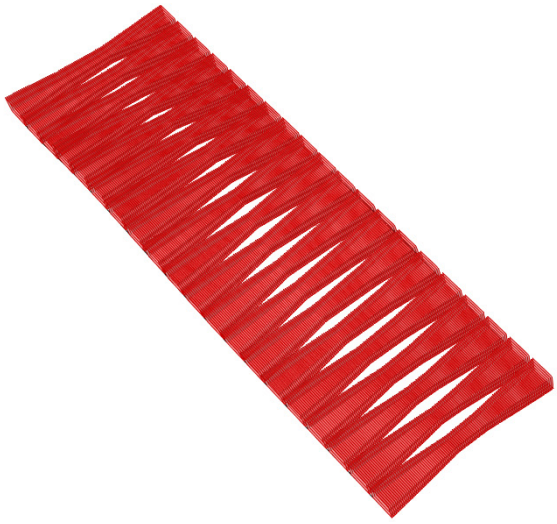


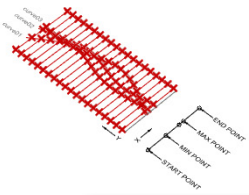


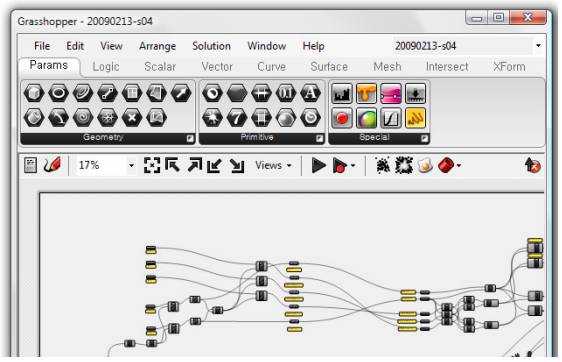
You can do this in manual fashion.



+ **Connect brep to those boxes.**







Works perfect!