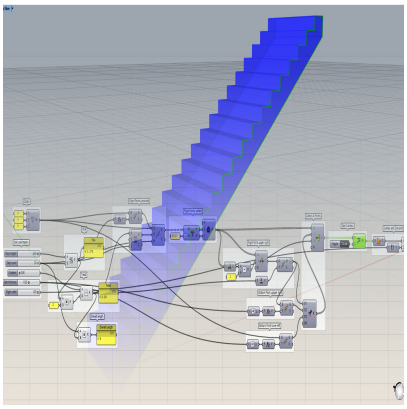


Rhino Grasshopper: Simple Stairs

Description

While diving further into Rhino Grasshopper I set myself this task: Extrude a simple flight of stairs from its elevation. The Parameters: Floor height, step count, gradient, slab thickness, flight width.

New to Grasshopper? I suggest you read this article in the first place.

Need more learning resources? Check this out.

Content

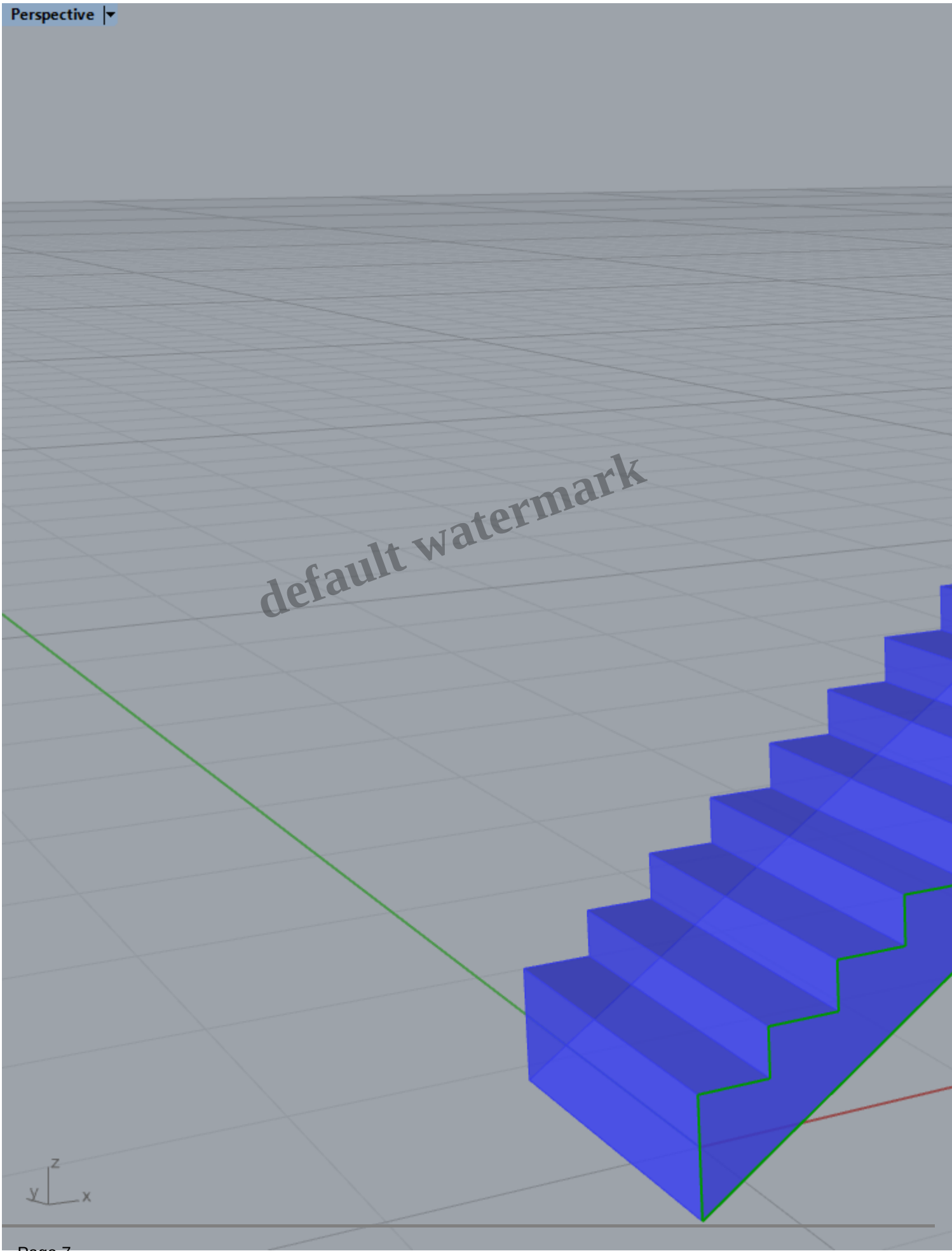
At the end of this article you'll find a download link to my stair file.

Outset: Stairs Production Strategy <

Disclaimer: This is my own solution. I developed it when I tried to figure out how to handle lists in Grasshopper. I am well aware there are other and maybe better solutions. To get aquainted with Grasshopper see this article.

My basic idea was to:

- Define *floor height, step count, gradient, slab thickness* and *flight width* (input parameters)
- Produce the according side elevation via *points* and *polyline*
- Produce the stairs as *extrusion* of this elevation


Perspective
default watermark
z
y
x

Whereas my list looks straightforward, the middle part – constructing the points establishing the stair contour – is by far the biggest. Let's go ahead.

See my article on Starting with Grasshopper. You'll find more learning resources here.

Input Parameters <

A flight of stairs can be defined by 5 input parameters:

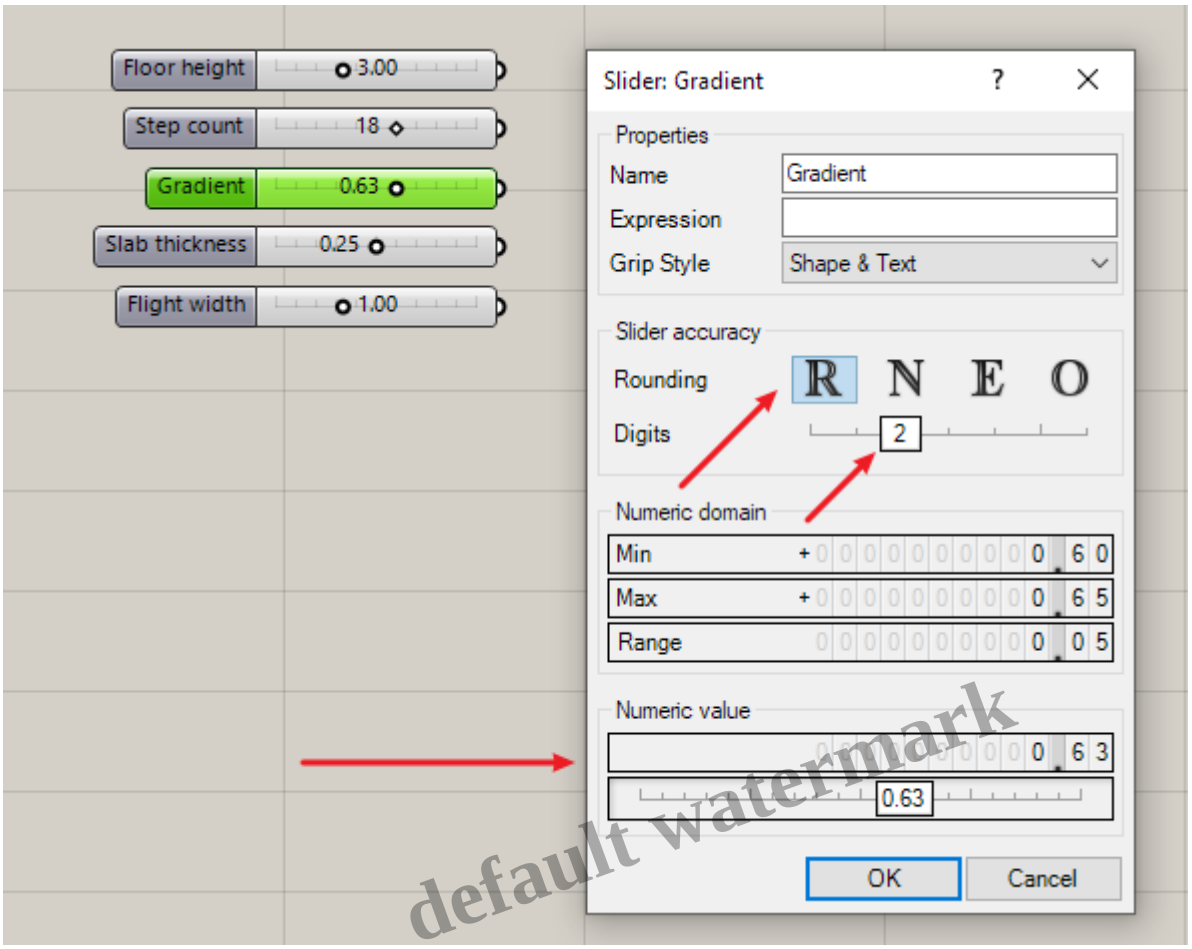
- *Floor height* (overall height)
- *Step count* (how many steps?)
- *Gradient* (steep – shallow)
- *Slab thickness*
- *Flight width*

This relates to a massive concrete stair without landing, turns and railing.

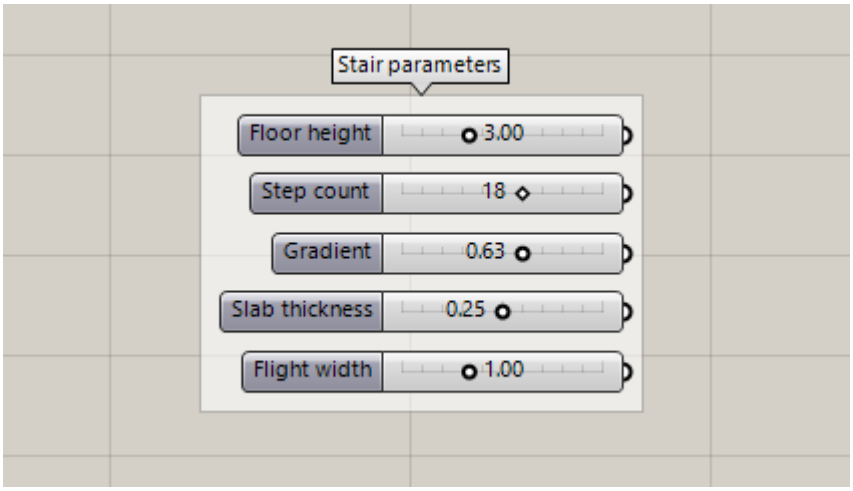
So let's produce 5 *Number Sliders* via *typing* numbers (metric) on the GH *canvas* as shown. Don't forget giving proper names:

- *Floor height:* 2.75<3<3.5
- *Step count:* 15<18<20
- *Gradient:* 0.60<0.63<0.65 (I'll explain)
- *Slab thickness:* 0.2<0.25<0.3
- *Flight width:* 0.75<1<1.5

Observe: Grasshopper automatically produces *Integer* or *Real* sliders with according decimals based on your input:



Let's *group* these sliders and call it *Stair Parameters:*

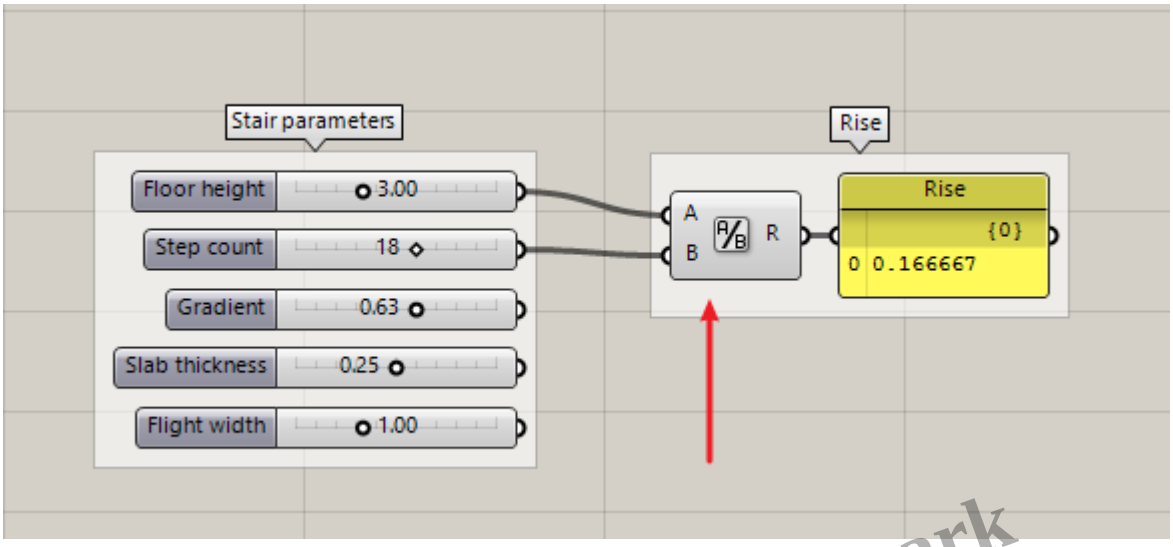


See my article on Starting with Grasshopper. You'll find more learning resources here.

Rise, Tread, Overall Length <

Based on our inputs we'll now calculate the *Rise* and *Tread* values plus the overall length of the stair.

First, *Rise* – that's pretty straight forward: Use a *Division* component with *Number Slider Floor height* and *Number Slider Step count* as inputs:

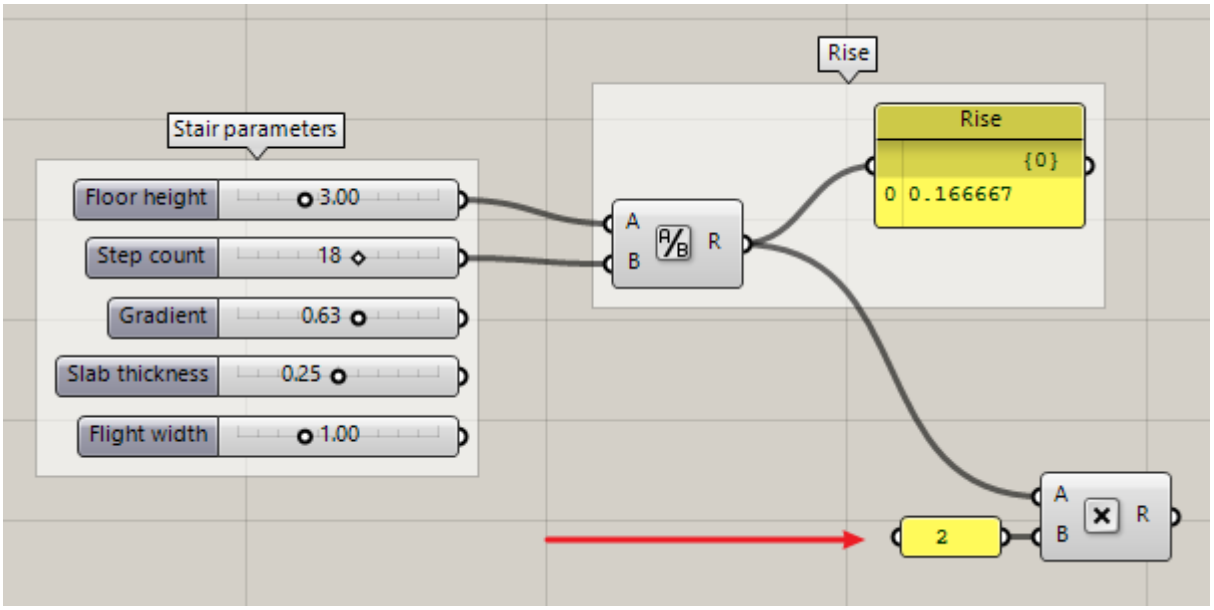


As you see I added a *Panel* to check the result and again grouped the new components.

Now for the *Tread*: This one has to be calculated based on *Rise* and our *Gradient* input:

- The real-life formula is: 2 Risers + 1 Tread = 60-65 cm (Comfort Rule)
- Which in our example translates to: 2 x *Rise* + *Tread* = *Gradient*
- *Rise* and *Gradient* being given, we solve the equation for *Tread*: *Tread* = *Gradient* – 2 x *Rise*

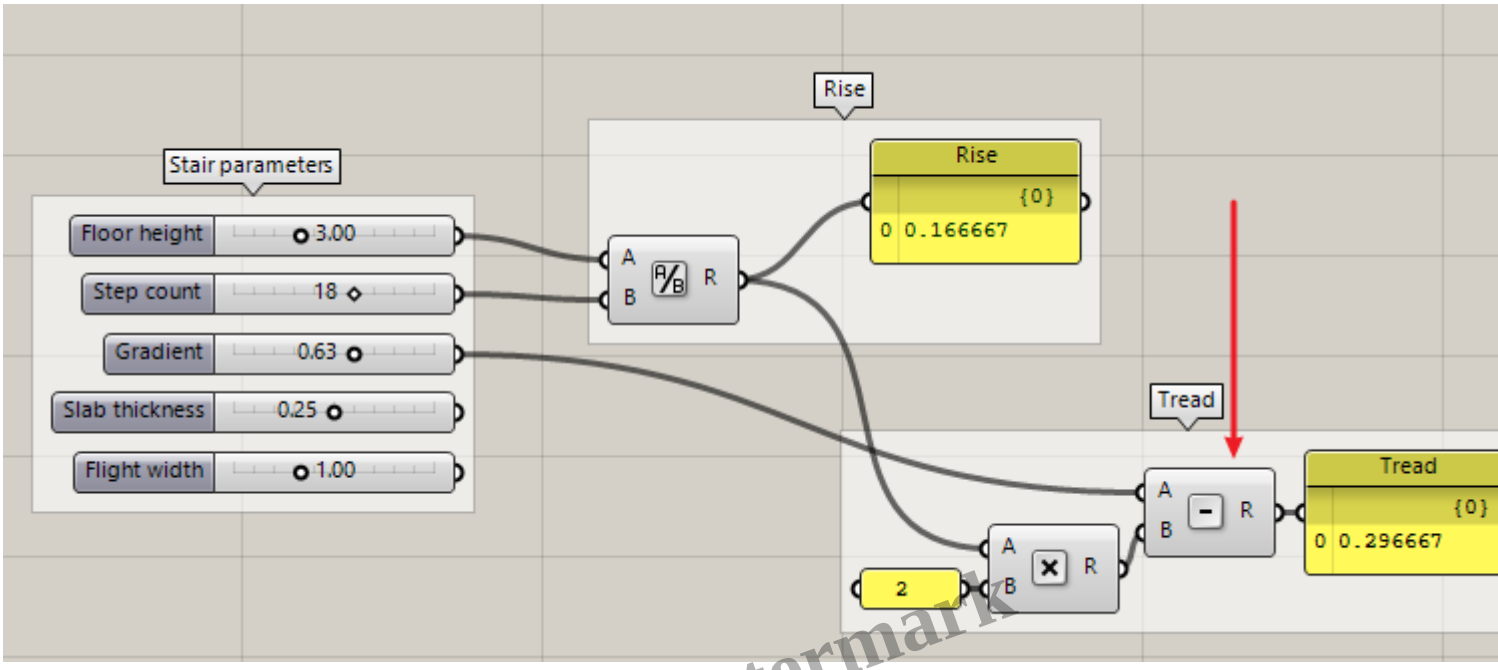
So first of all use a *Multiplication* component and input the result of our little *Rise* calculation:



For *Rise x 2*, we could right-click input *B* and *Set Data Item = 2*. I prefer to use an external panel with

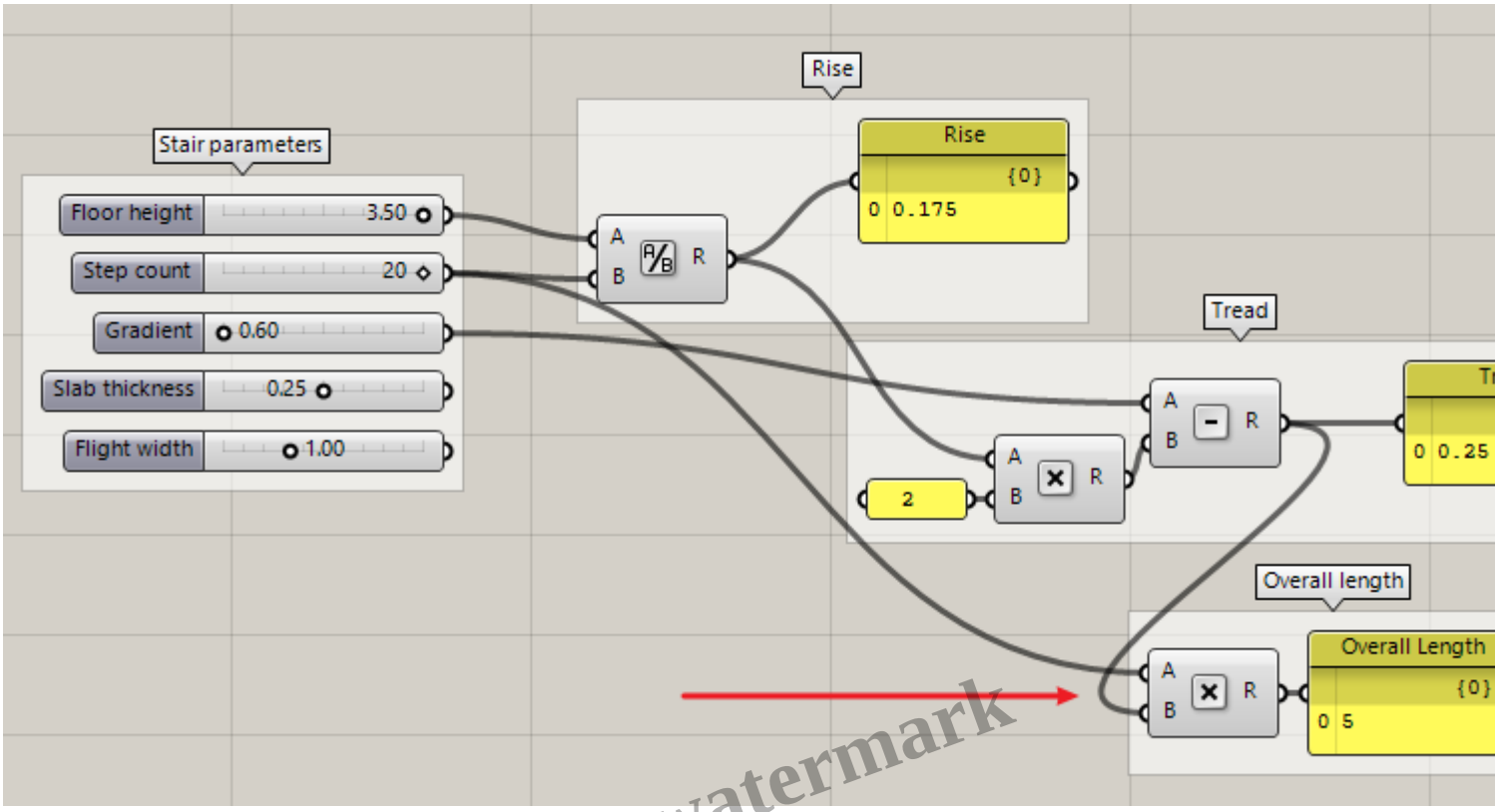
the same data input *(= 2)* because it makes my definition more comprehendable.

To subtract this *Rise x 2* from our *Gradient* input use a *Subtraction* component:



Input *A* is our *Gradient* value, input *B* the result of *Rise x 2*. Again, I added a panel showing the resulting tread length for better readability.

Last item here: the overall length of the stairs. We don't need this to produce our stair, it's just for information. As you anticipate you'll use a *Multiplication* again, *Step count* x *Tread:*

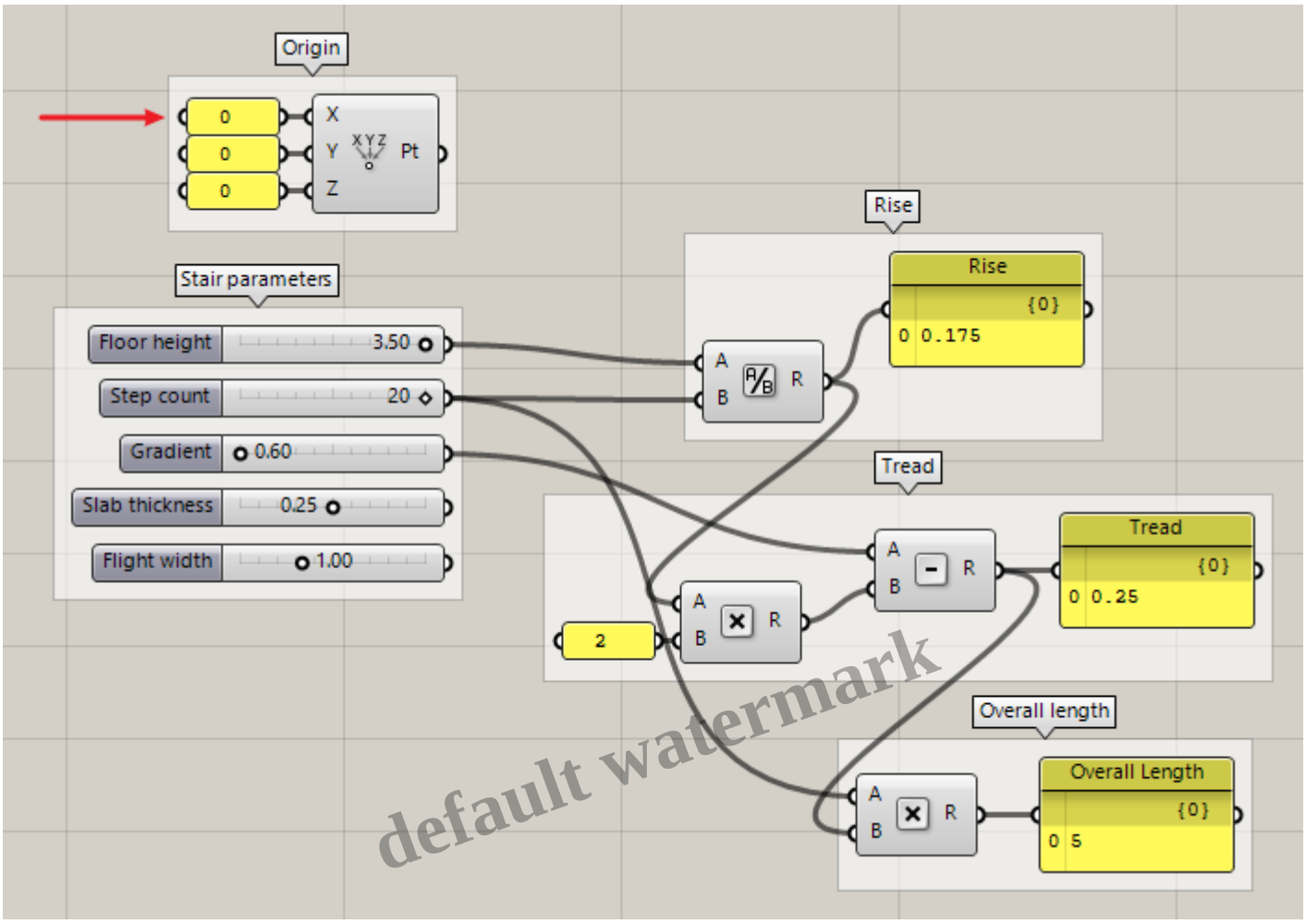


That is all nice and fine, we have set up all geometric information we need for the moment. (*Flight width* is an issue we'll deal with later.) Let's start with real construction work.

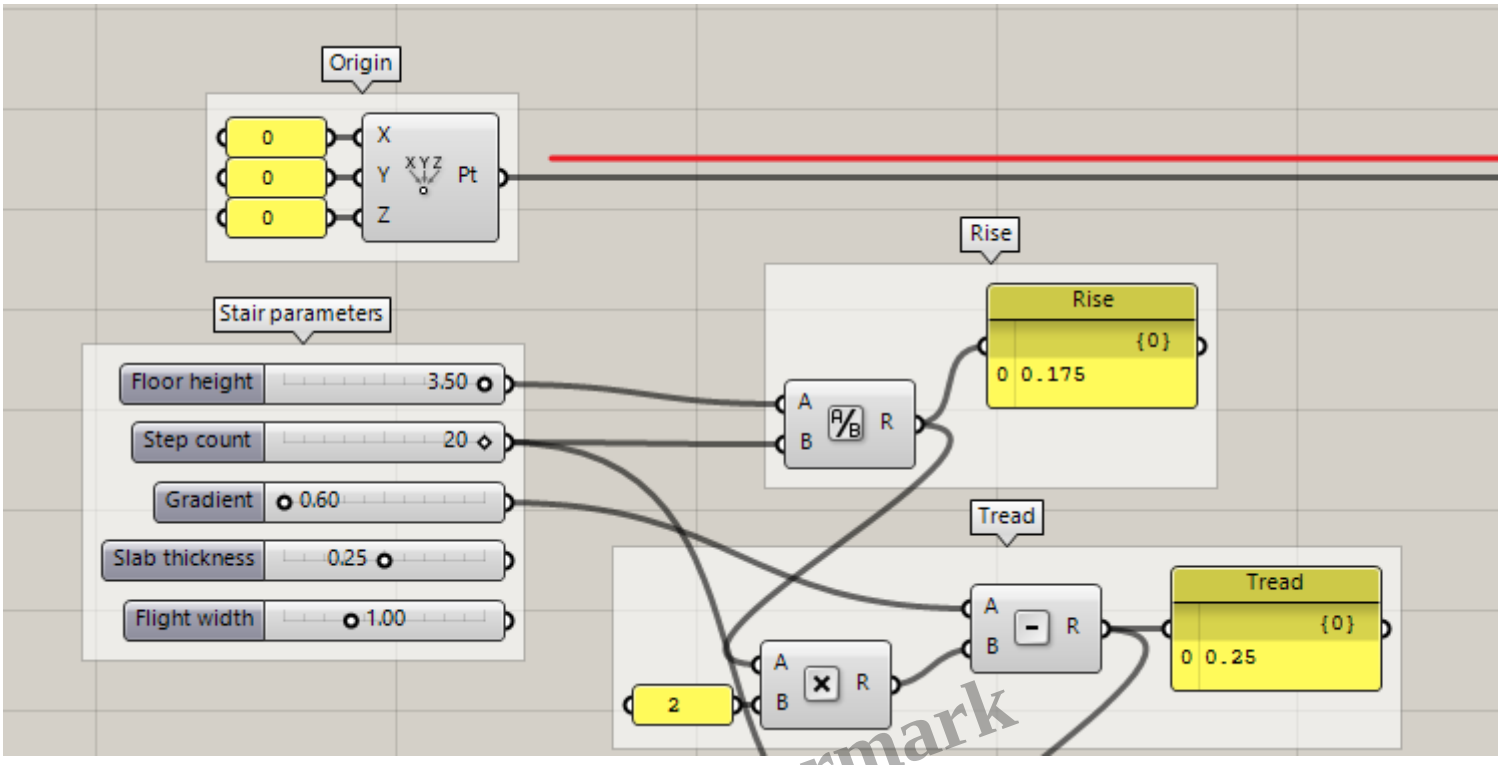
See my article on Starting with Grasshopper. You'll find more learning resources here.

Constructing Points: Risers <

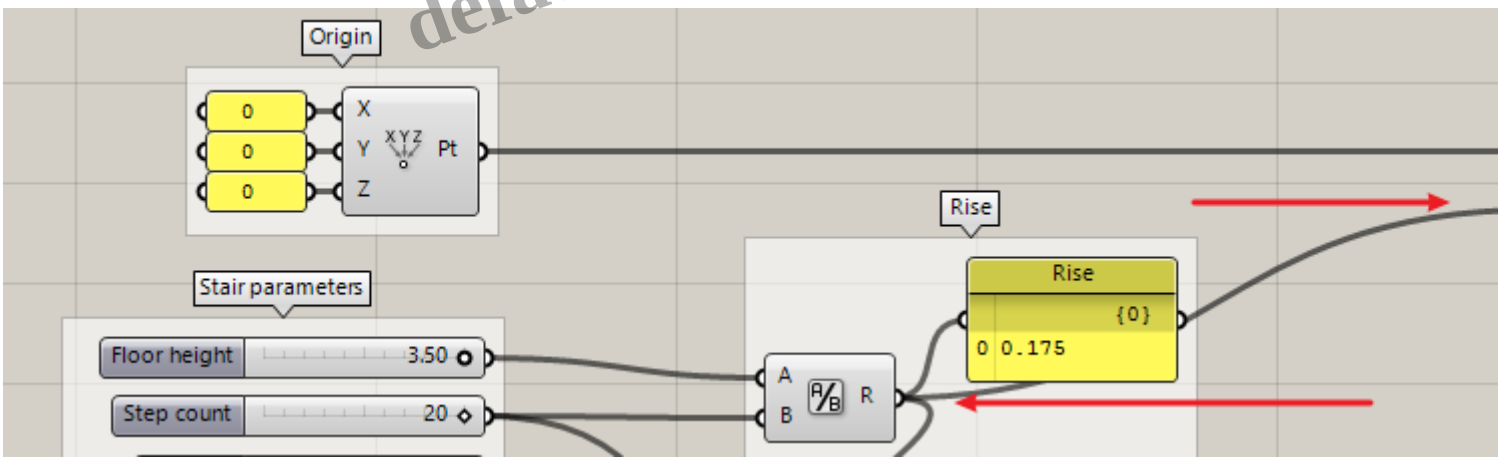
The first set of points we will draw are the ones on top of each riser. To start with, we need a point which serves as origin for our stair object. Use a *Construct Point* component and feed it with *Zeros*. Again, I would suggest using external panels to make this little thing more readable:



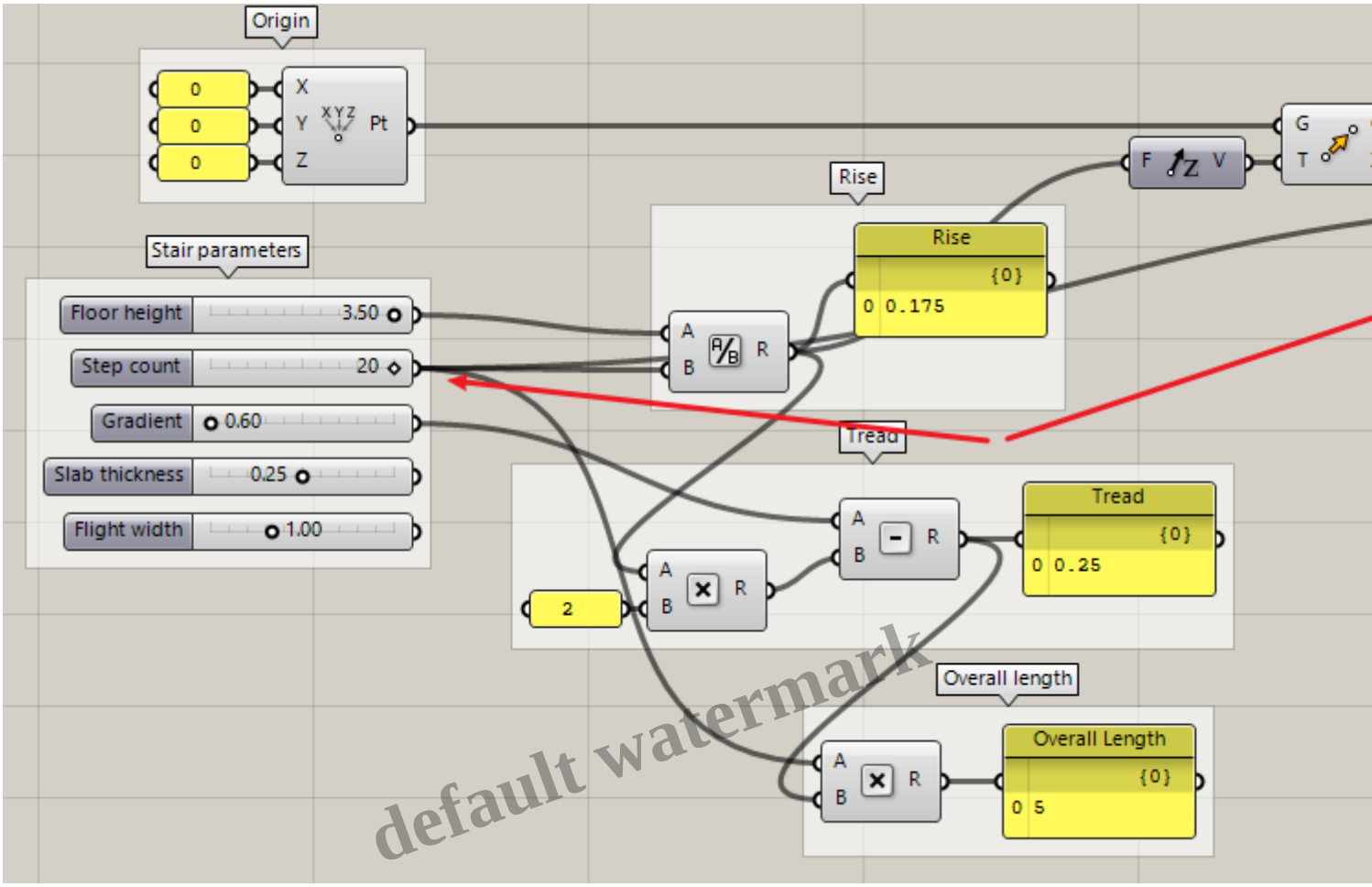
Now let's move this point up. *Moving* in *Grasshopper* always means *copying*. Use a *Move* component and connect it with your *Origin* point. Feed its *T* input with a *Unit Z* vector component:



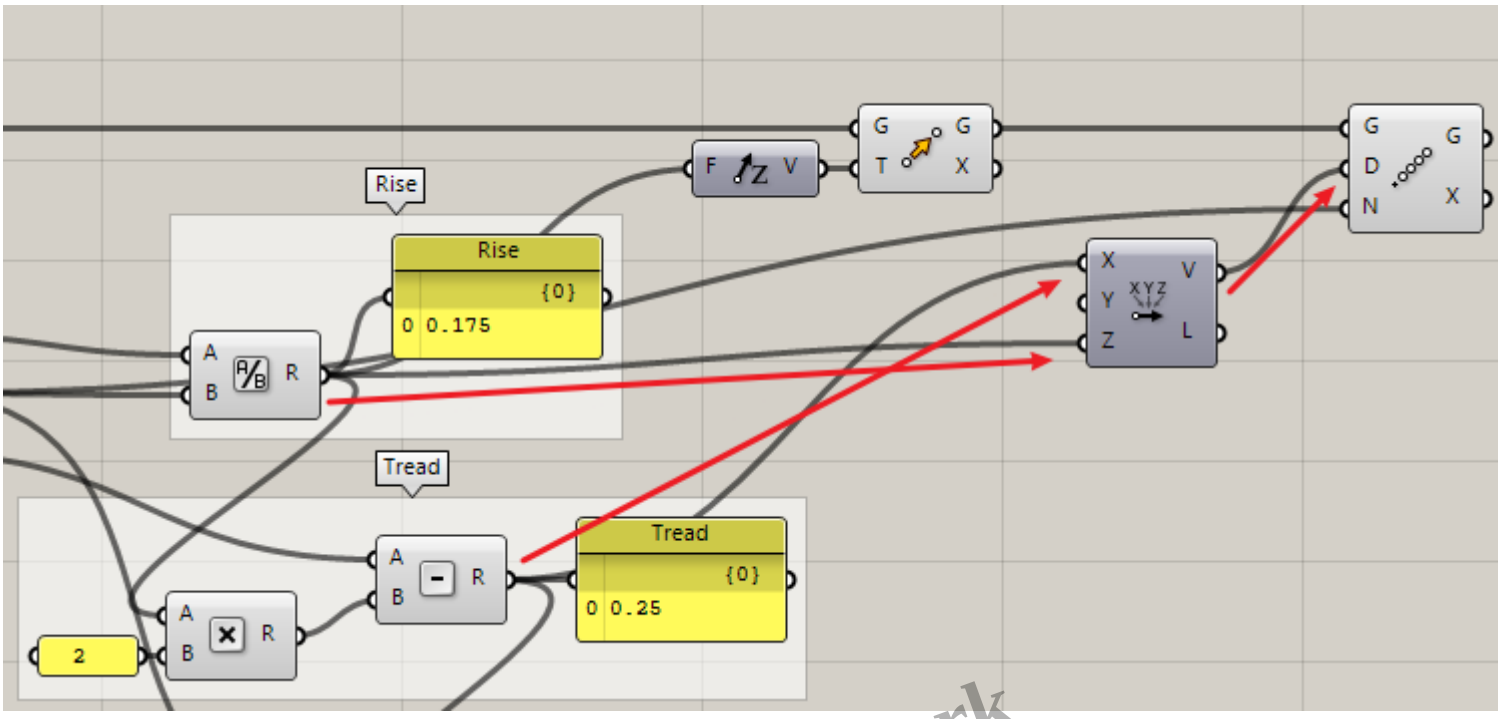
In *Rhino's Front* viewport you already see two points. Now *Unit Z* does not mean the point's height is correct. Feed its *F* input with our *Rise* value:



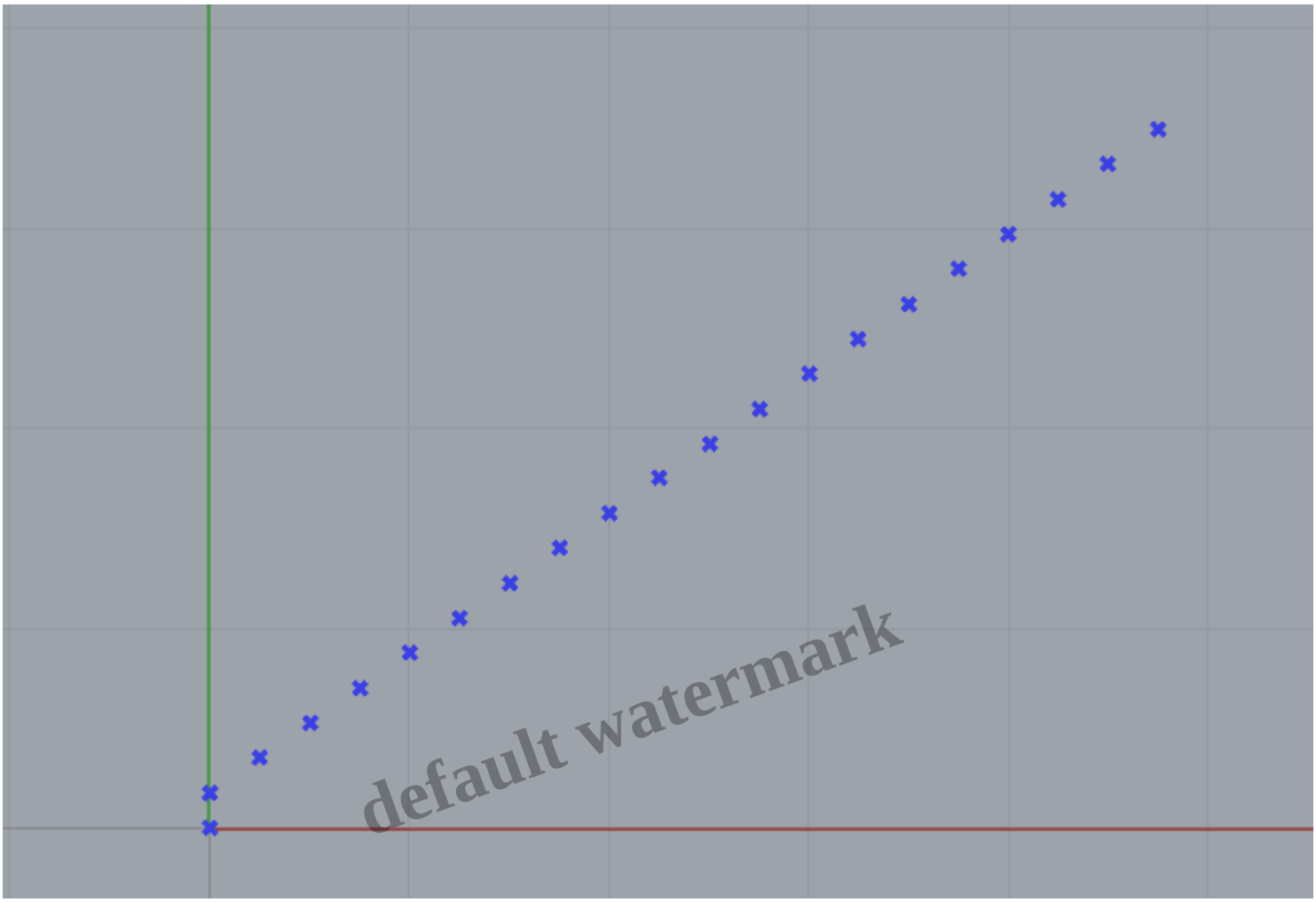
Congratulations, you have constructed your first 2 points. Now instead of constructing the following tread I want you to multiply this last point. Use a *Linear Array* component for this. Connect *G* output of your moved point to *G* input of your *Array* component. Connect its *N* input to the *Step count Number Slider* because you want as many point instances as you have steps:



Linear Array's Direction input needs explicit cordinates. Use a *Vector XYZ* for this task – it allows you to define a vector with custom direction. First of all connect its *V* output to the *Linear Array's D* input. Now in the plane we are working in we have two axes we can utilize: *X* and *Z*. As you might imagine, for our vector's *X* value you'll use the *Tread* value while for *Z* you will use the *Rise* value. So connect everything accordingly:



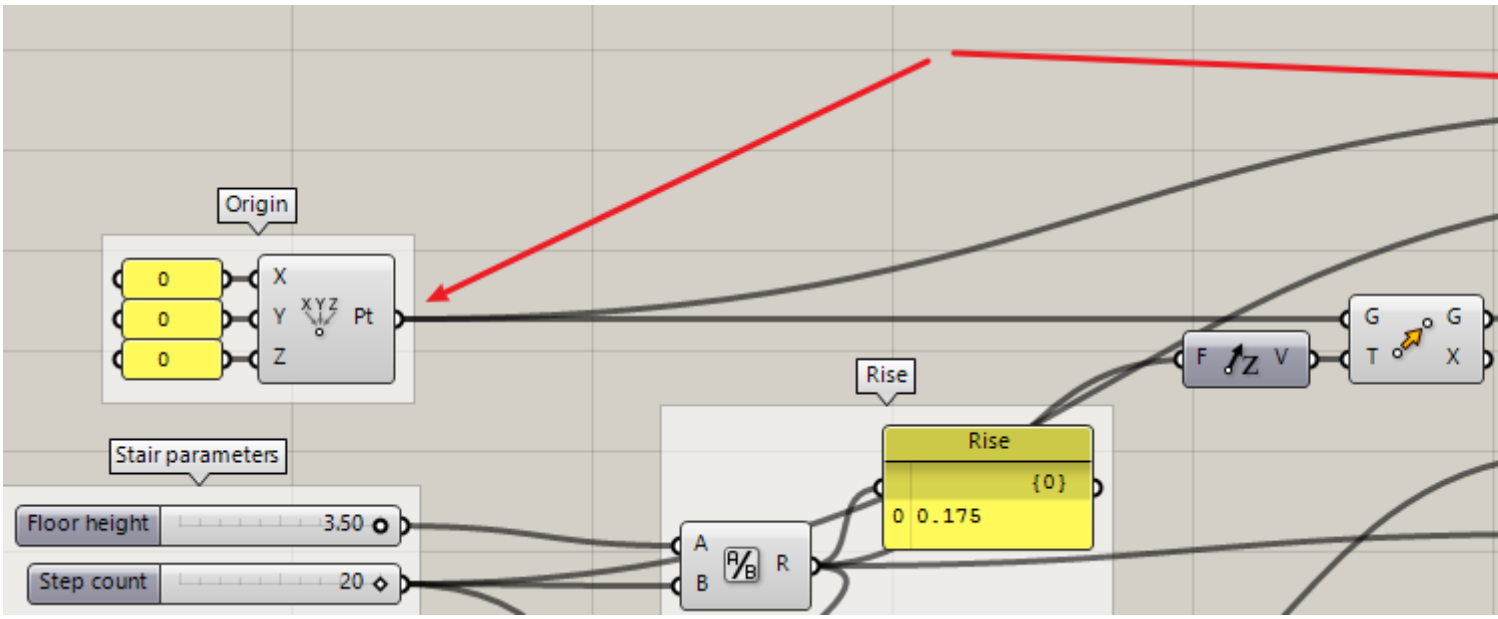
And this is the result in *Rhino's* viewport:



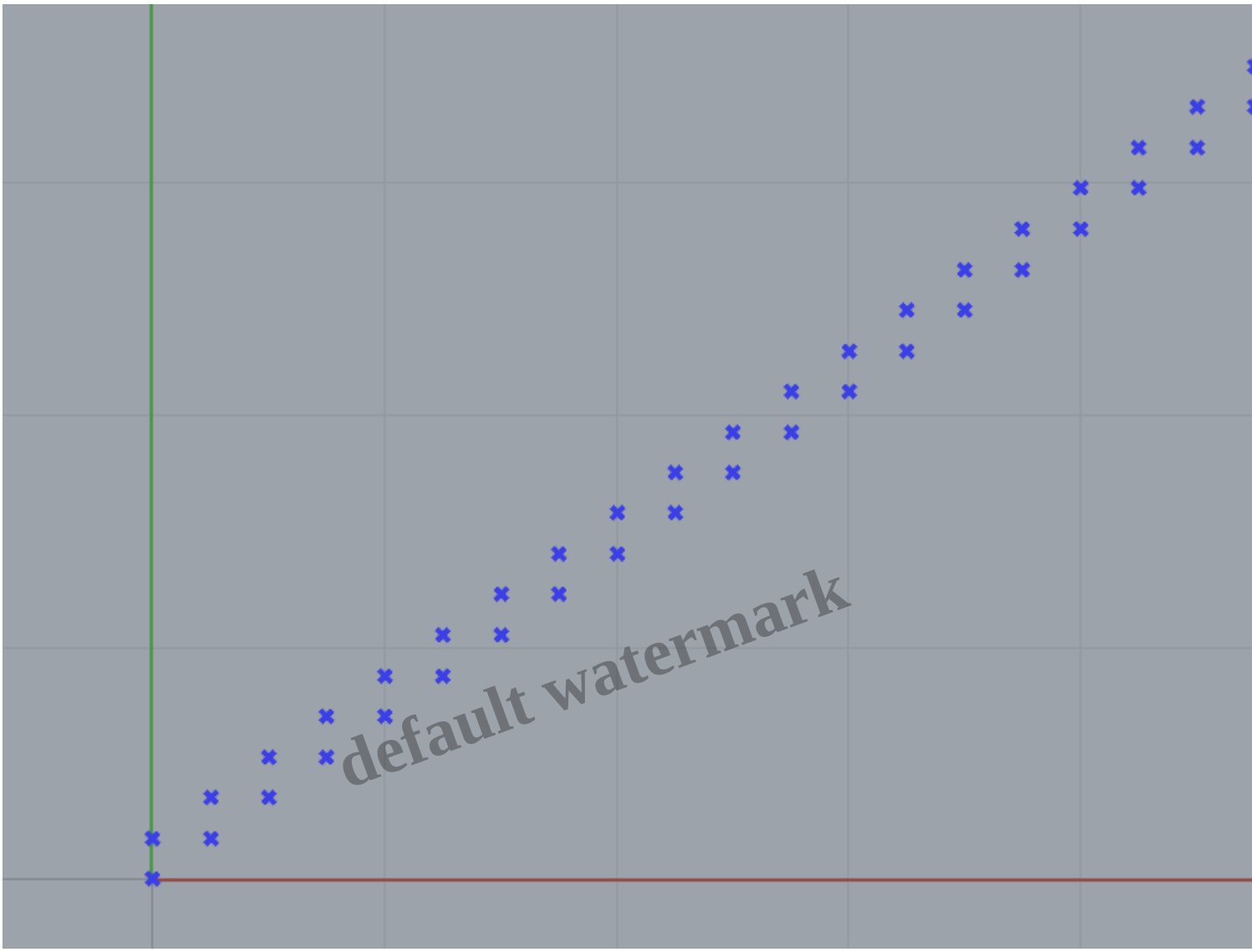
See my article on Starting with Grasshopper. You'll find more learning resources here.

Constructing Points: Treads <

Your next point array will produce the missing points in your stair's zigzag contour. Easy enough, you can utilize components you already have: *Linear Array* together with its *XYZ Vector* – only this time we'll be using the *Origin* point instead. So just connect your *Origin* point output to the *Linear Array* input holding down the *Shift* key:



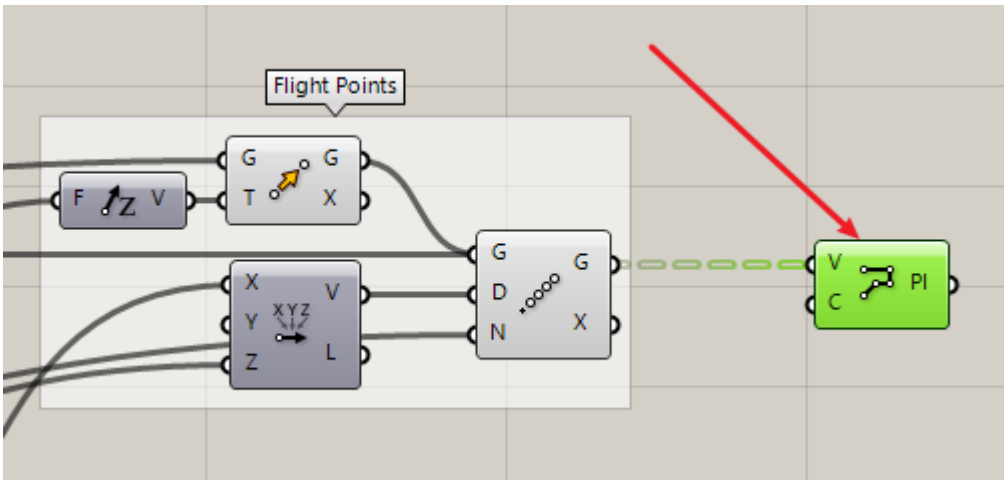
(Holding down *Shift* you may plug more than one feed into one input.) This is the result:



See my article on Starting with Grasshopper. You'll find more learning resources here.

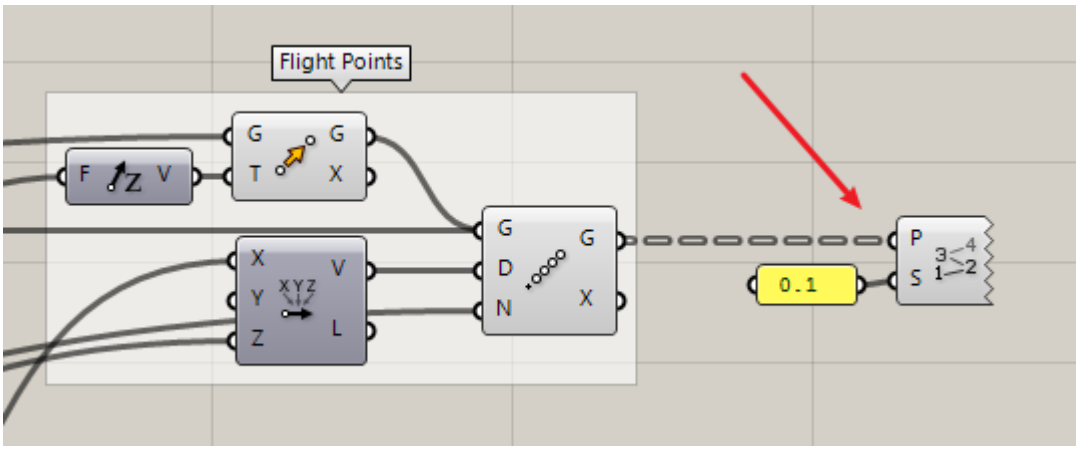
Sorting Points <

Our aim is a closed stair contour that we can fill and extrude into 3D. The contour will be a polyline connecting the points we are just constructing. For a correct polyline point order is crucial. As we already have a bunch of points we'll want to check how they are ordered. To test it use a *Polyline* component and connect it to your *Array G* output:



As you see your point order is not correct:

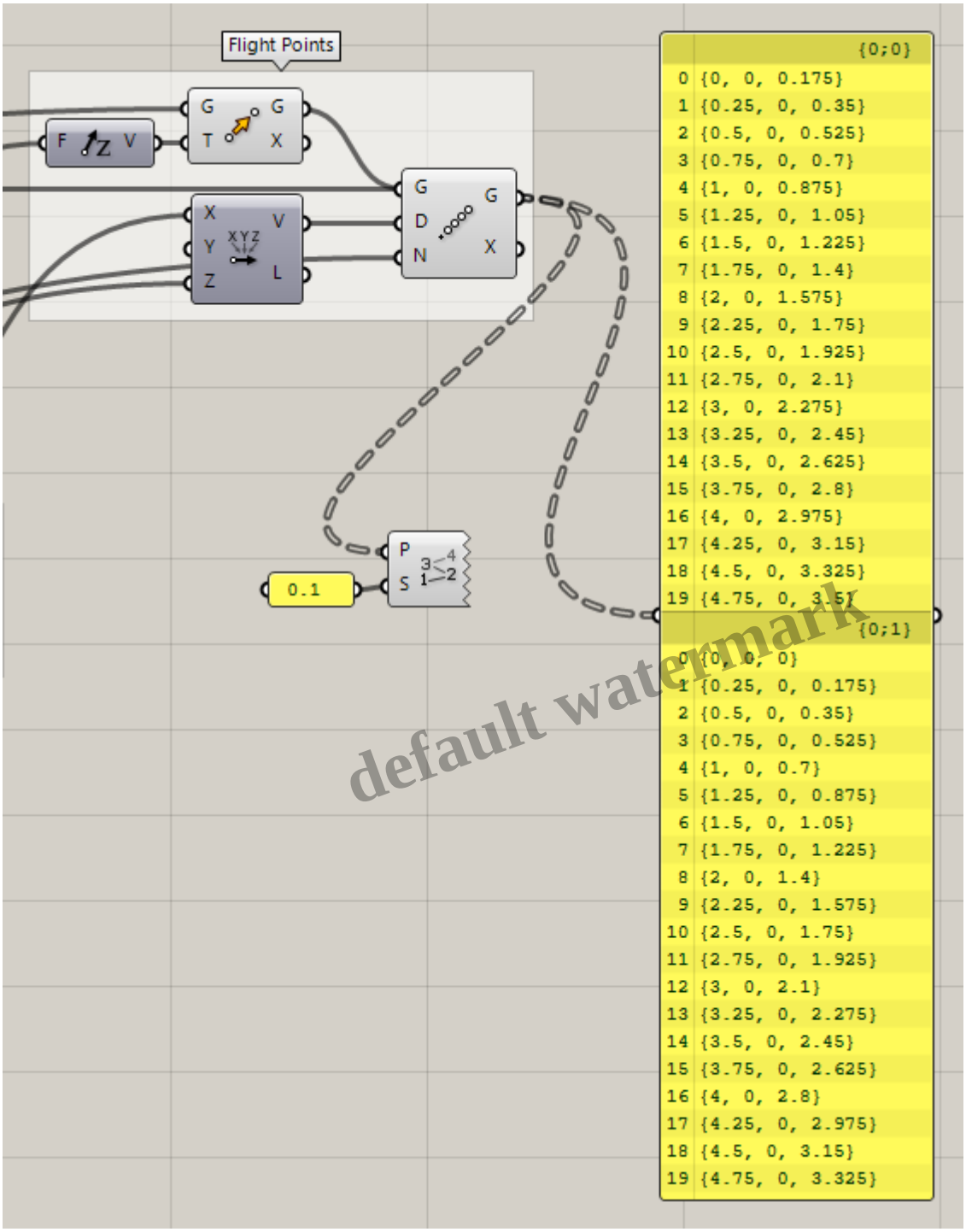
Disconnect the *Polyline* component and use a *Point List* component to display point numbers in *Rhino's* viewport:



As you can see the numbering relates to the way the polyline was drawn:

We have two lists of points that are sorted in itself but independant from one another. This is no surprise because we constructed them that way. So obviously we will have to combine the two lists in a way that the list items are paired.

First of all let’s have a look at the *Linear Array’s* output. Just feed it into a text panel:



What you see are two lists showing our points. The first column shows the indices (always starting with *0)*, the second column shows the point ccordinates.

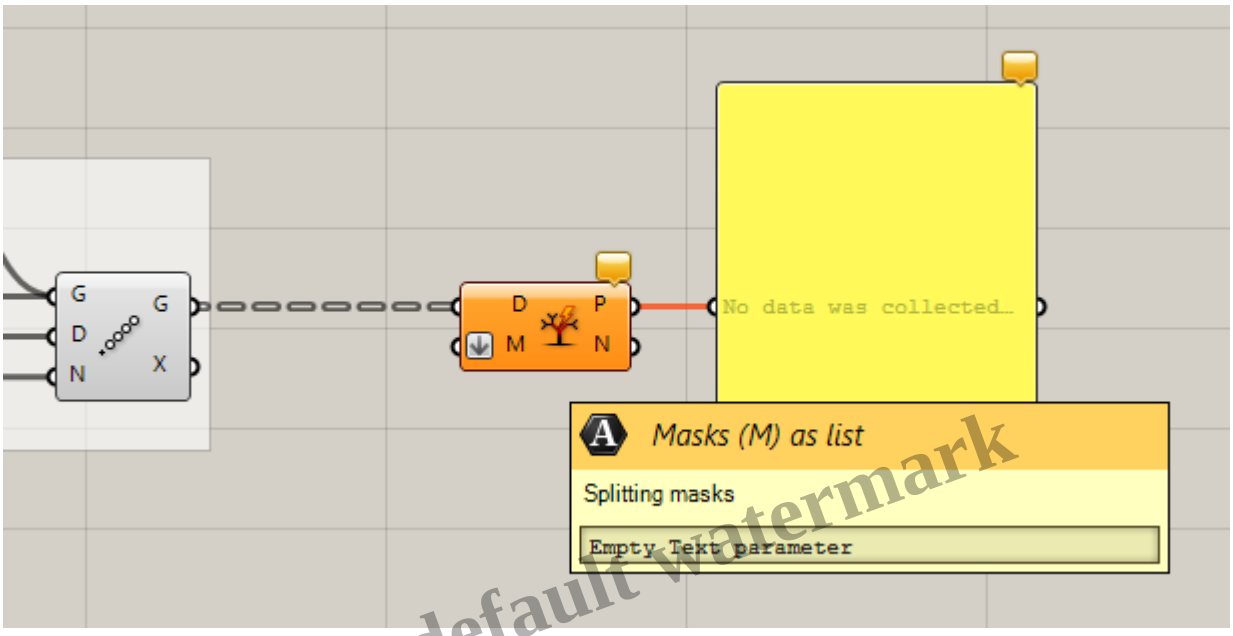
In *Grasshopper* these sub-lists are called *paths* – one is *path {0;0},* the other one is *path {0;1}.*

What we need, however, is one list with all the points sorted like this:

- Index *0:* point *0* from *path {0;0}*
- Index *1:* point *0* from *path {0;1}*

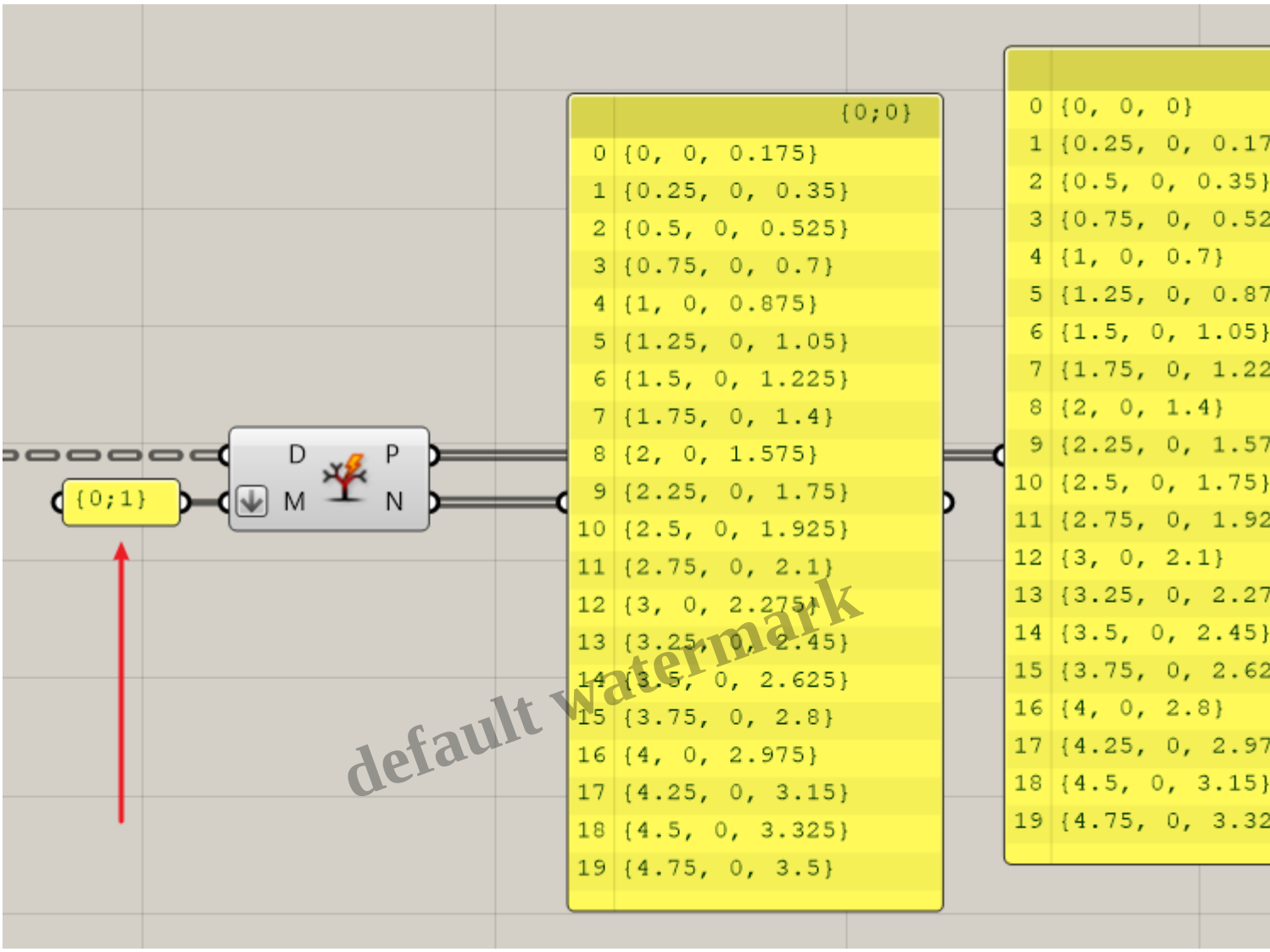
- Index *2:* point *1* from *path {0;0}*
- Index *3:* point *1* from *path {0;1}*
- … and so on

O.K. First of all we have to split the *Array* output. Use a *Split Tree* component and feed the *Array G* output into its *Data* input. Up to now nothing happens because the component needs to be fed with a *mask:*

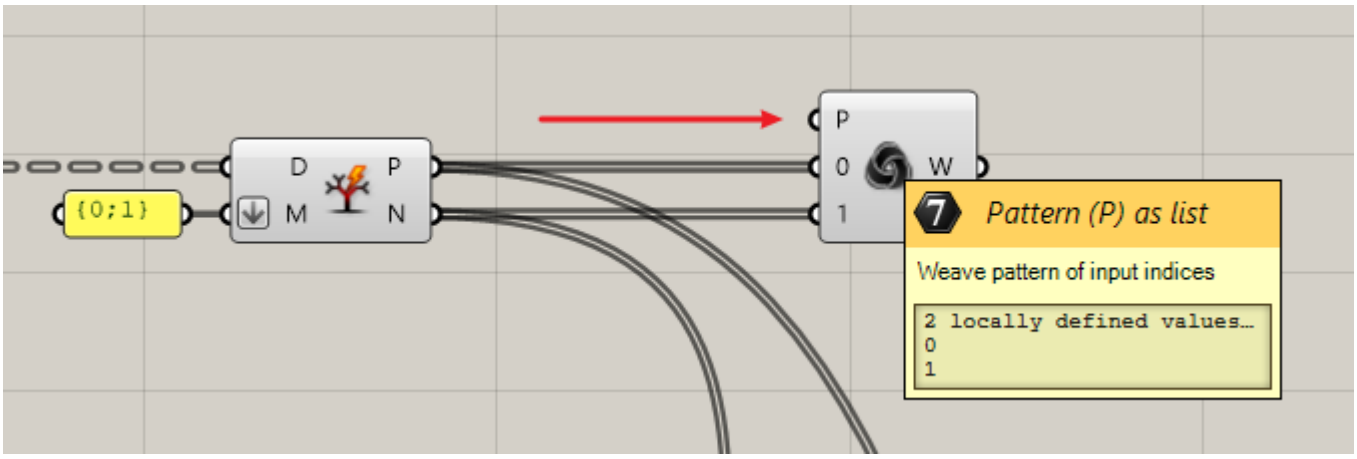


What this basically means is that *Grasshopper* wants to know which paths you want to move to *P* or *N* list. This may seem unnecessary because we have only two paths to deal with. But of course your list may well consist of more paths so this is what the mask is for.

And how is this mask supposed to look? Use a panel and write *{0;1}.* Connect this panel to *Split Tree's M* input. As you see you get two separate lists as output *P* and *N:*



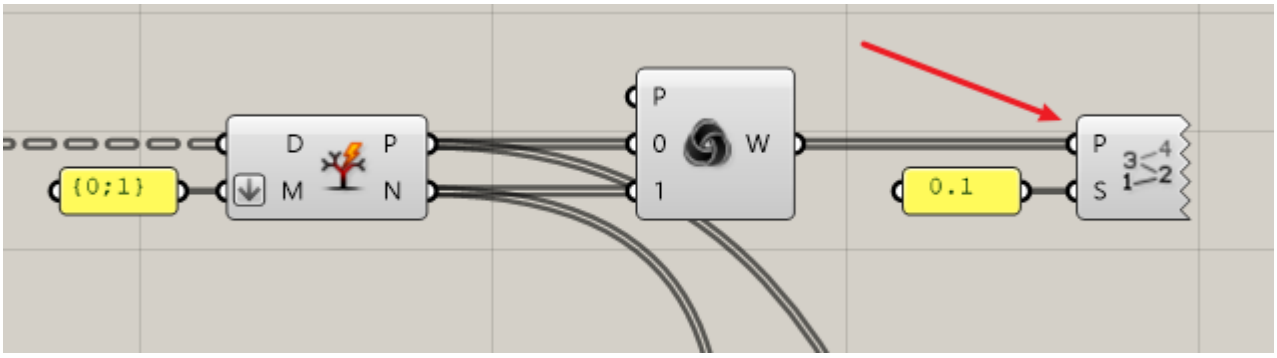
As said above the two list have to be combined again pairing the points with the same index. Use the *Weave* component for this:



Connect *P* and *N* outputs of *Split Tree* with *0* and *1* inputs of your *Weave* component. This way you tell *Grasshopper*

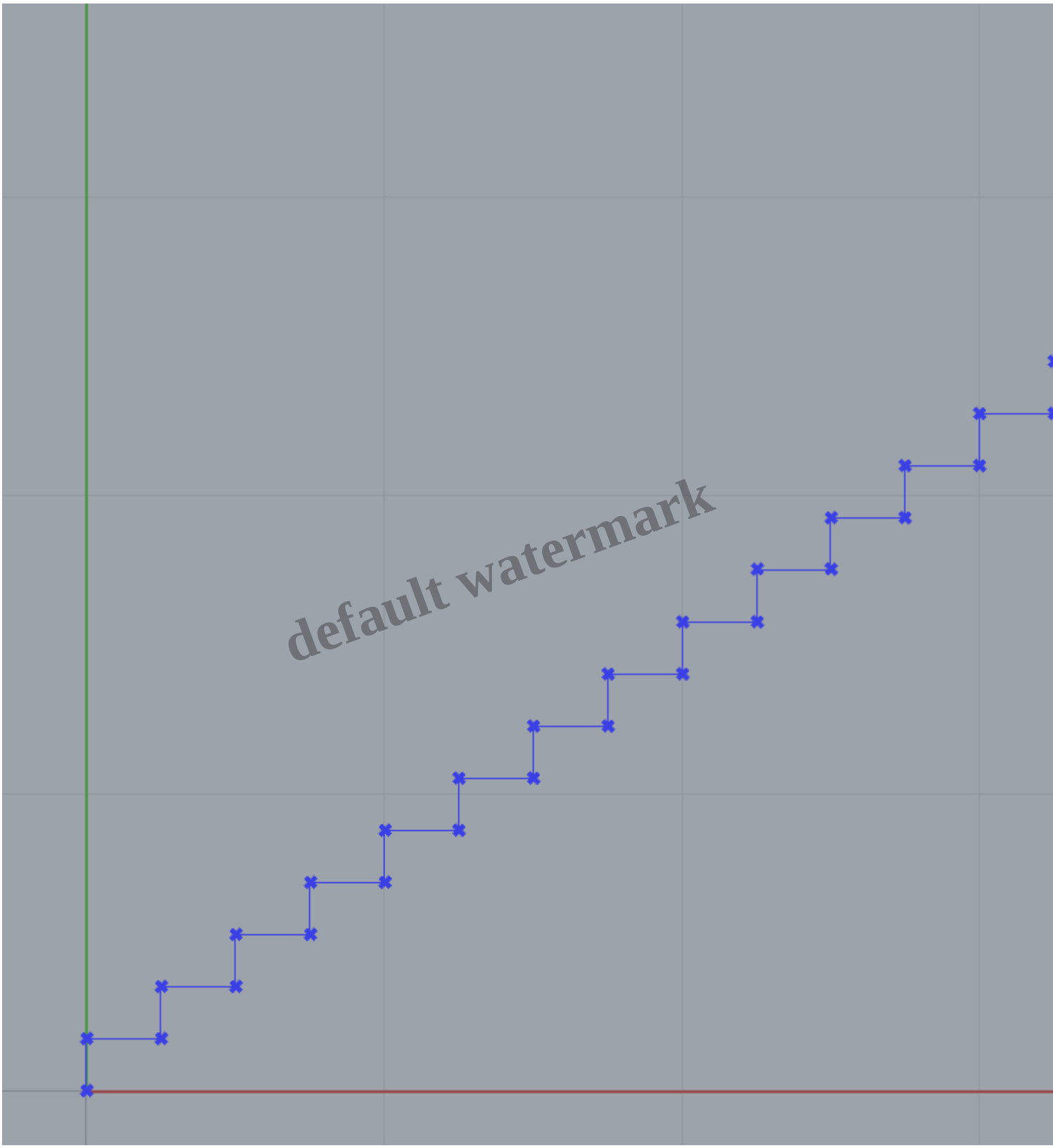
that you want to merge these two list into one. What's crucial about this component ist the *Pattern* in which this merging is done. As you see per default the *Pattern* is set to *0-1.* This means that *Weave* takes an item of list *0,* then of list *1* … and so on until all list items are dealt with.

Looking at my list panels above it seems that this default pattern might actually work. To check reconnect the *Point List* component to the *Weave* output:



Alas! *Rhino's* viewport shows a correct sorting of our points:

This was by far the most complex subject in this tutorial. For the moment let’s reconnect the *Polyline* component to our result:



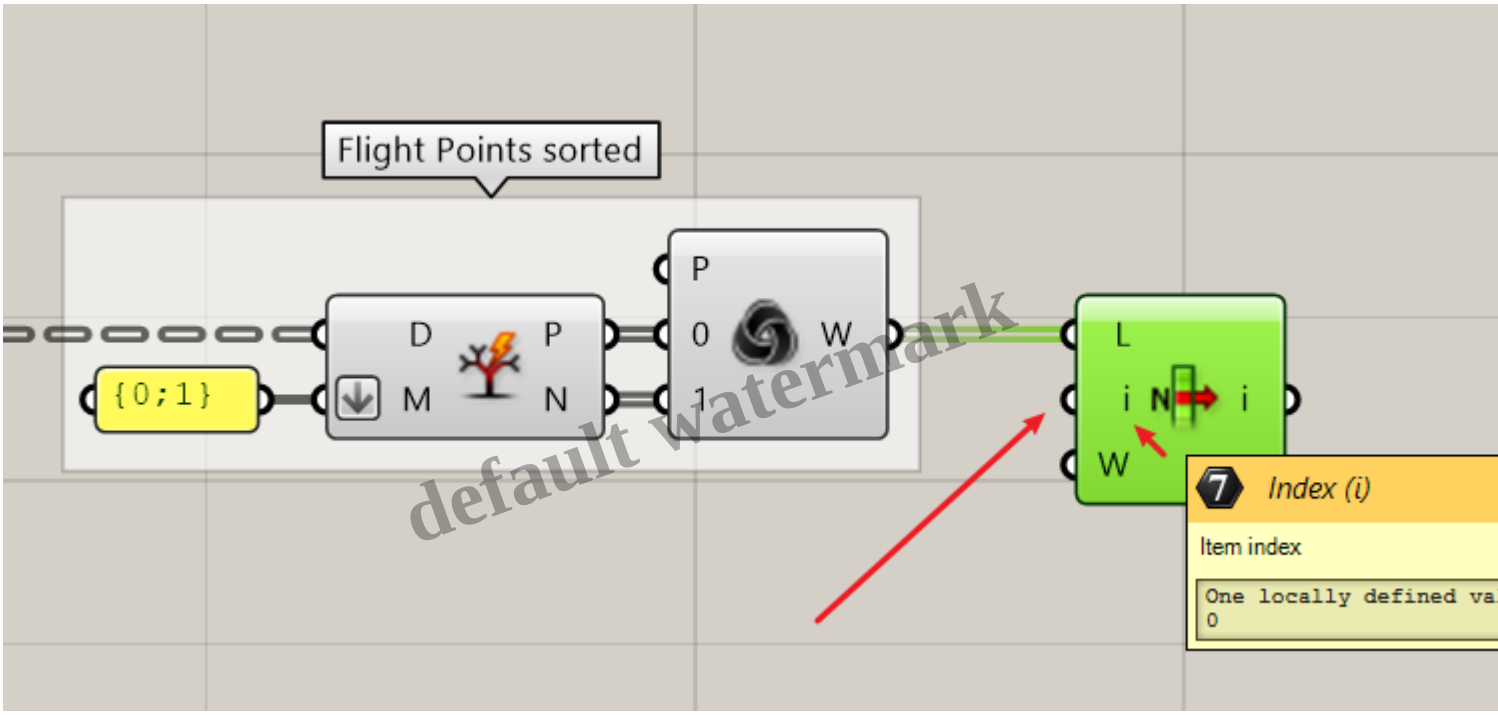
See my article on Starting with Grasshopper. You'll find more learning resources here.

Constructing Missing Points <

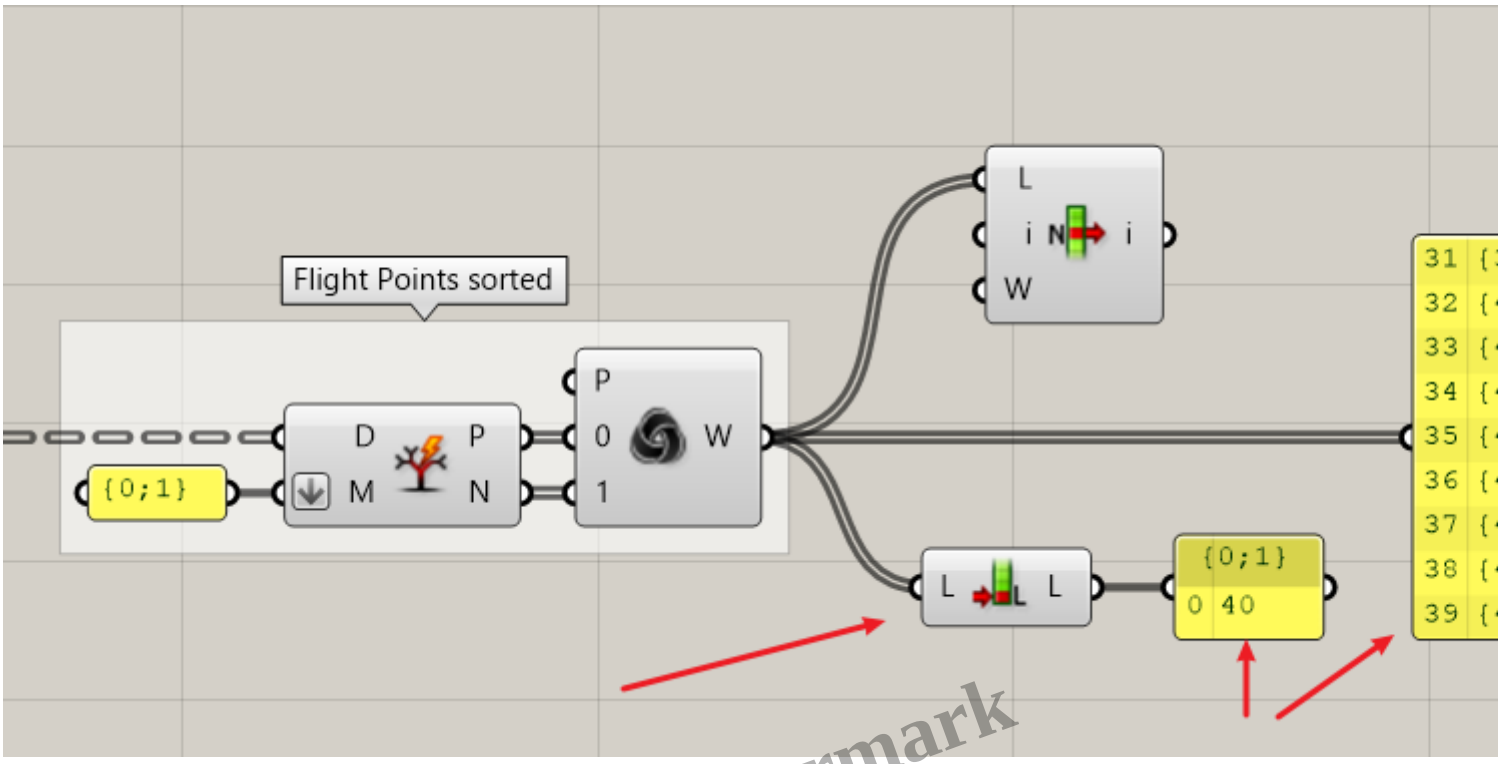
We are not finished though. We still need some additional points for our stair contour to be complete.

Let's start with the last tread on upper floor level – here we need a copy of our last point in *X*-direction, the offset being *1 Tread* value. But how do we adress the last point in our list?

The suitable component is *List Item*. It has to be fed with our point list (obviously) and with an index indicating the very list item that has to be retrieved:

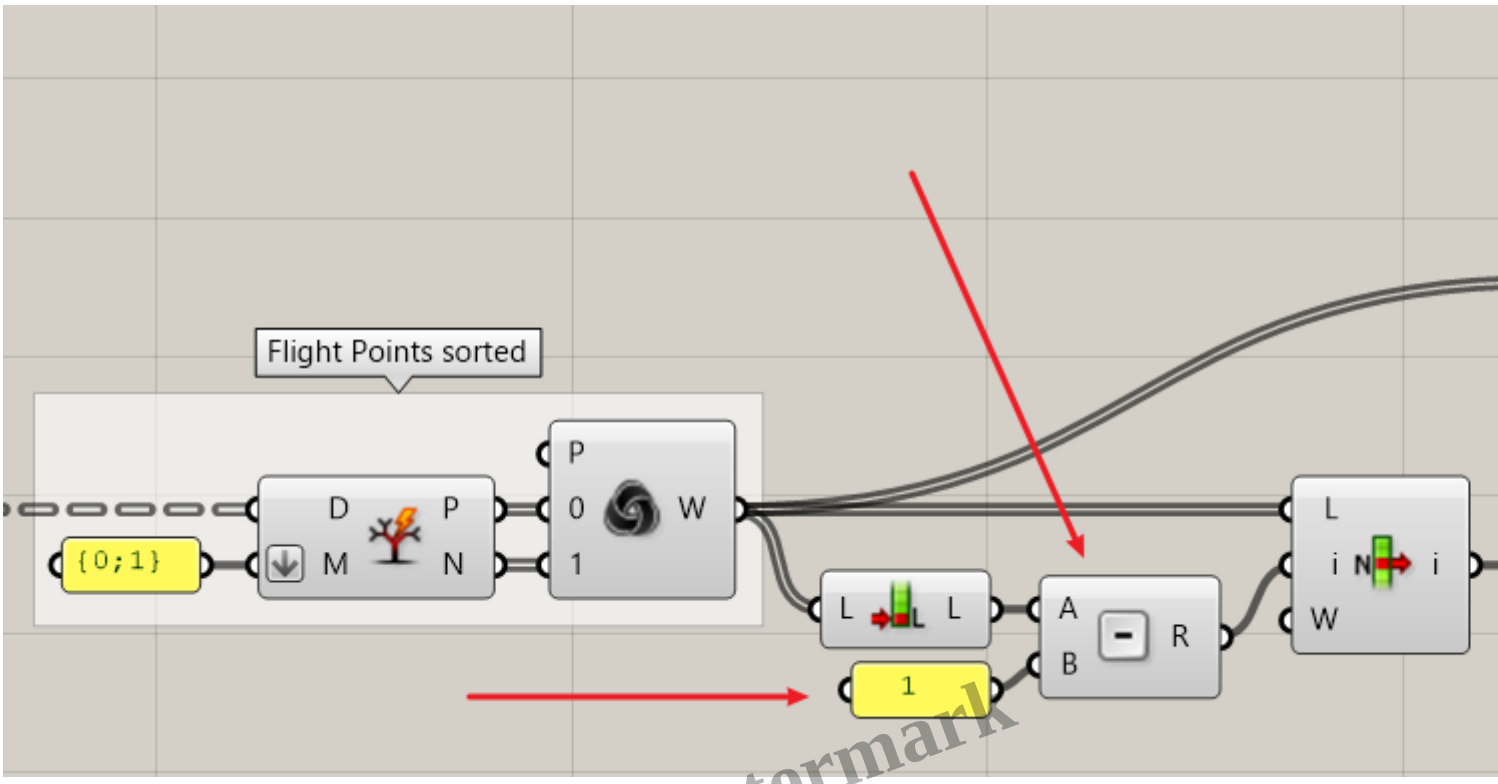


But here's the thing: Our index can't be a fixed value because our point list varies according to the *Step count*. We'll solve this dilemma by utilizing the *List Length* component for a little calculation:



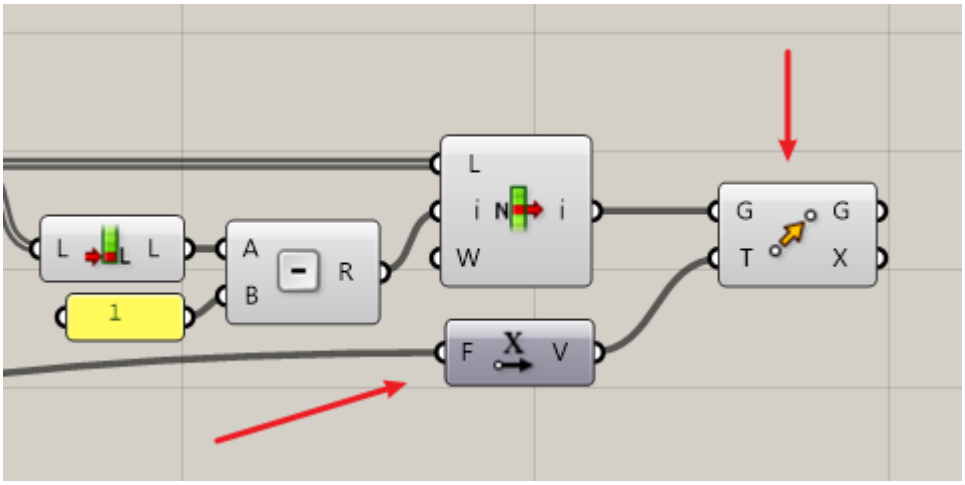
When you connect your *Weave* list output to the *List Length* component you see that it gives you the number of list entries (*40* – this number equals *Step count x 2).* The last index of our point list is *39.* This is because the first list index is always *0.*

Now if we want to adress the last point in our list using the *List Item* component we need *List length -1* as input for the *i* input of the *List Item* component. Use a *Subtraction* component:

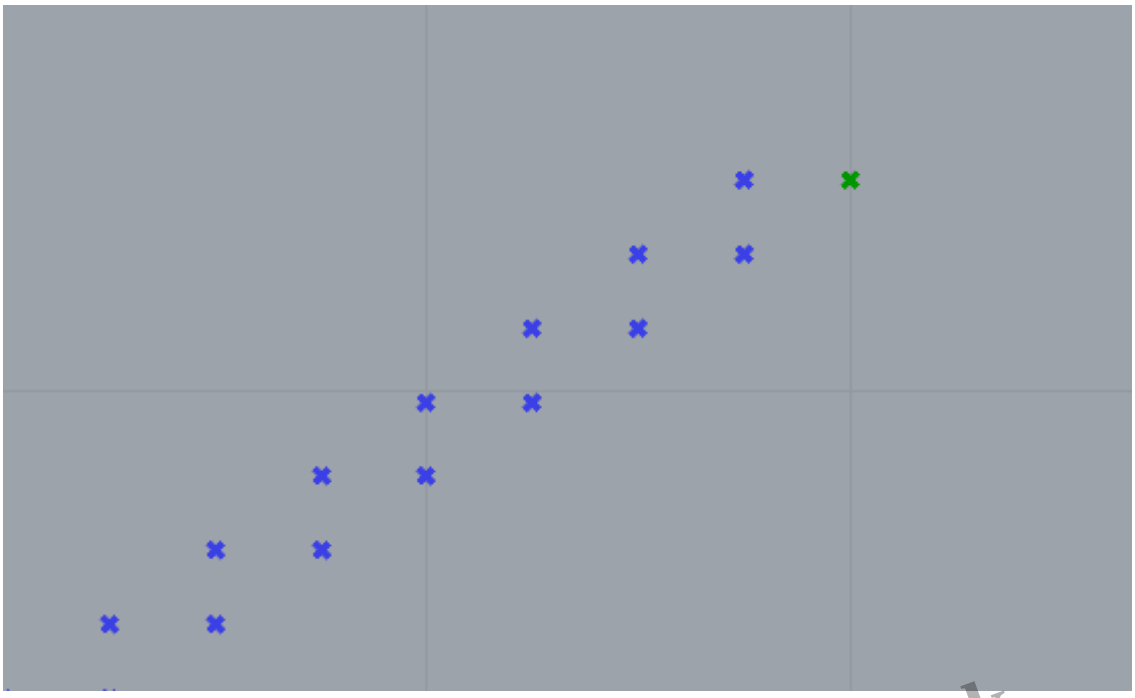


List Item has successfully retrieved the last point out of our list.

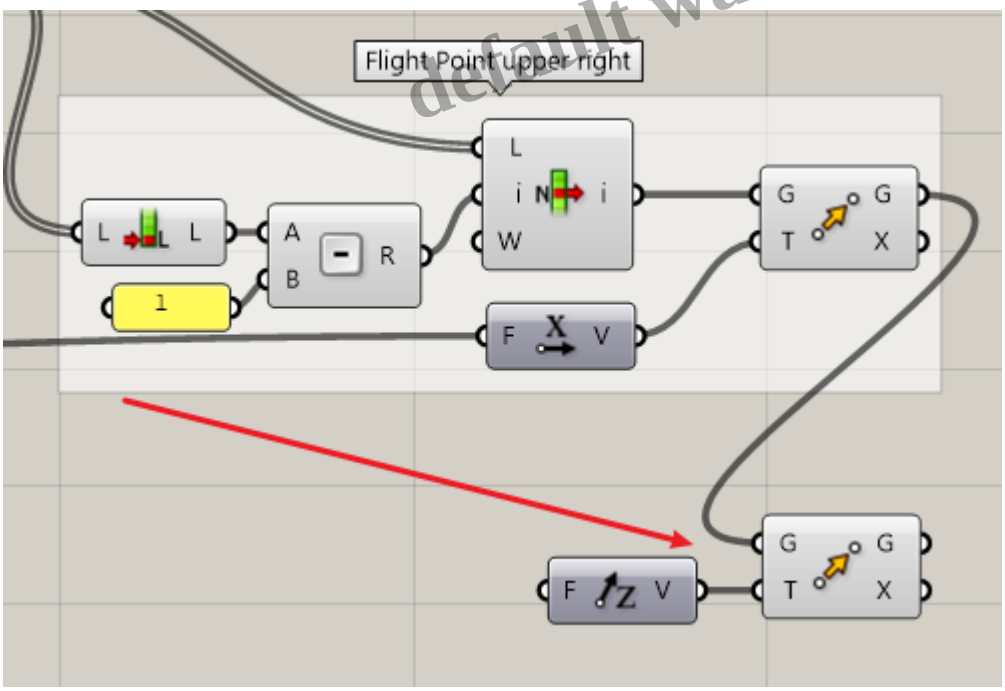
Now we want to move (and copy) this point in *X*-direction with an offset of *1 Tread.* Use a *Move* component together with a *Unit X* vector component. Feed this vector *(F)* with the *Tread* measure output:



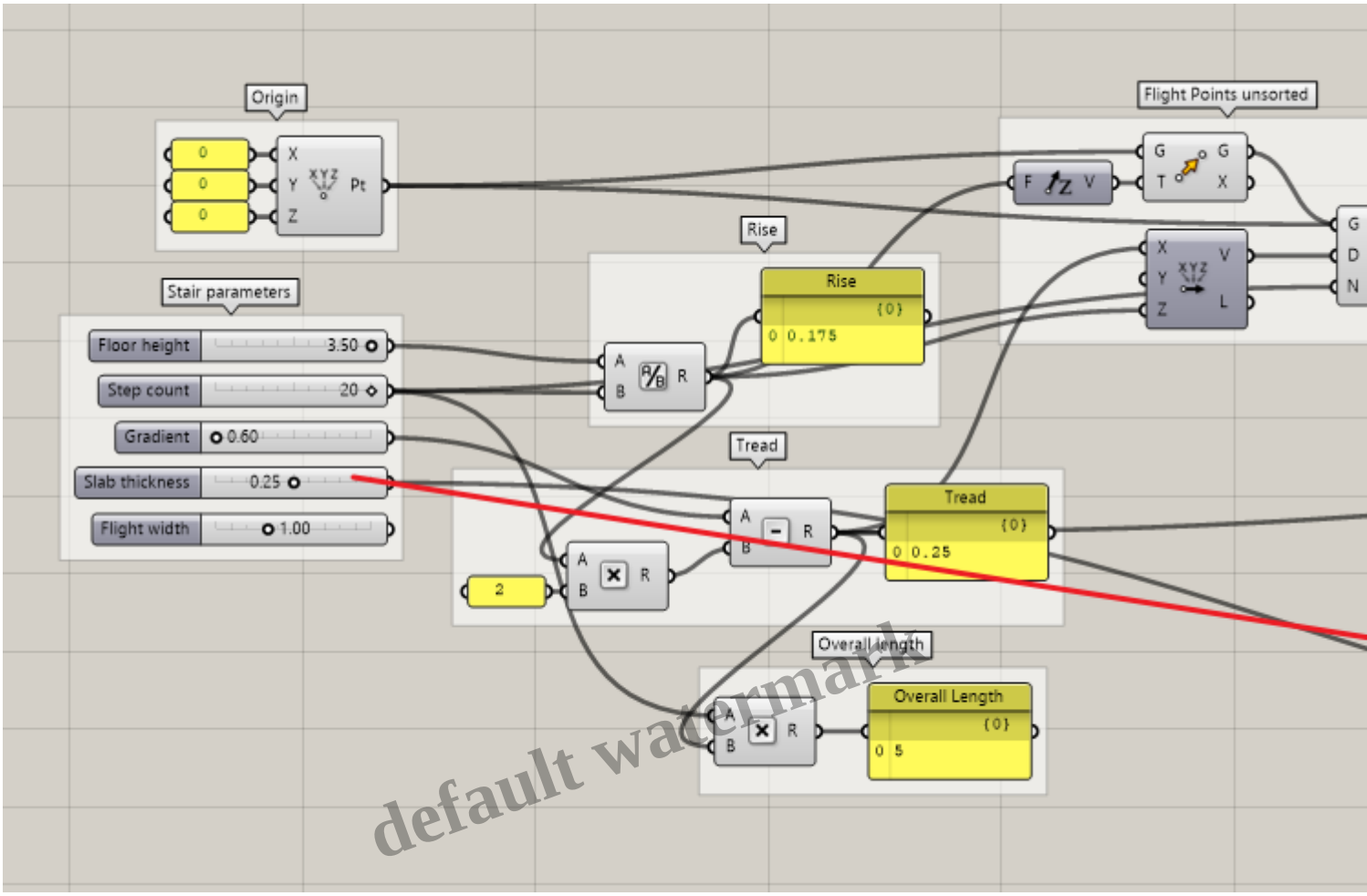
Here we are:



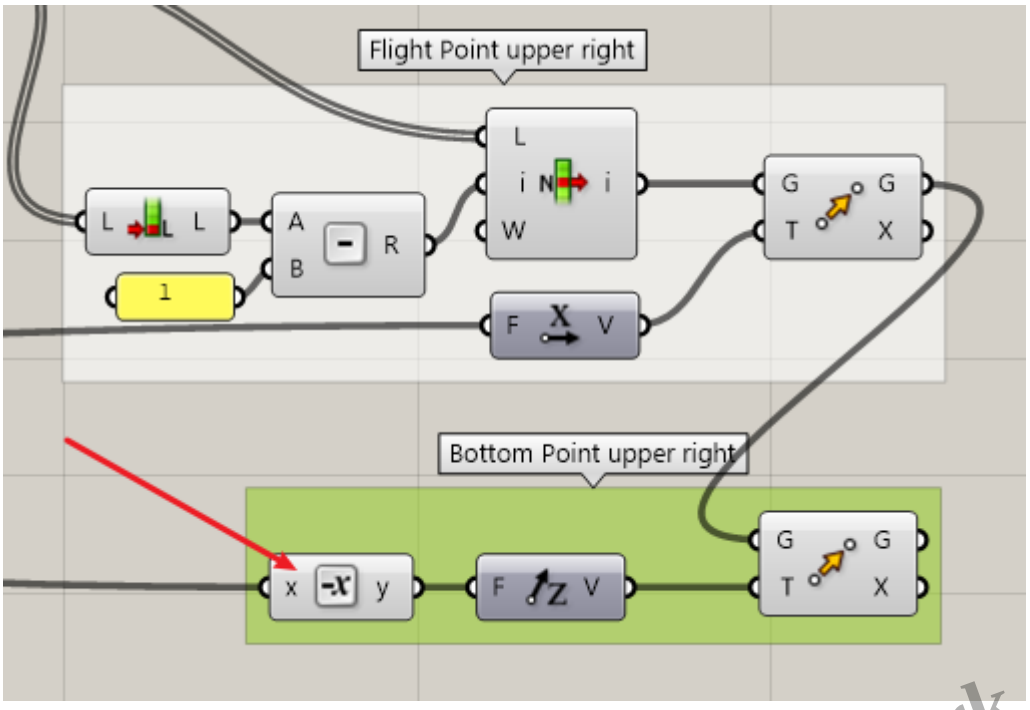
Now for the next point we move (and copy) this last point down in *Z*-direction, the offset being *Slab thickness*. Use a *Move* component again together with *Unit Z:*



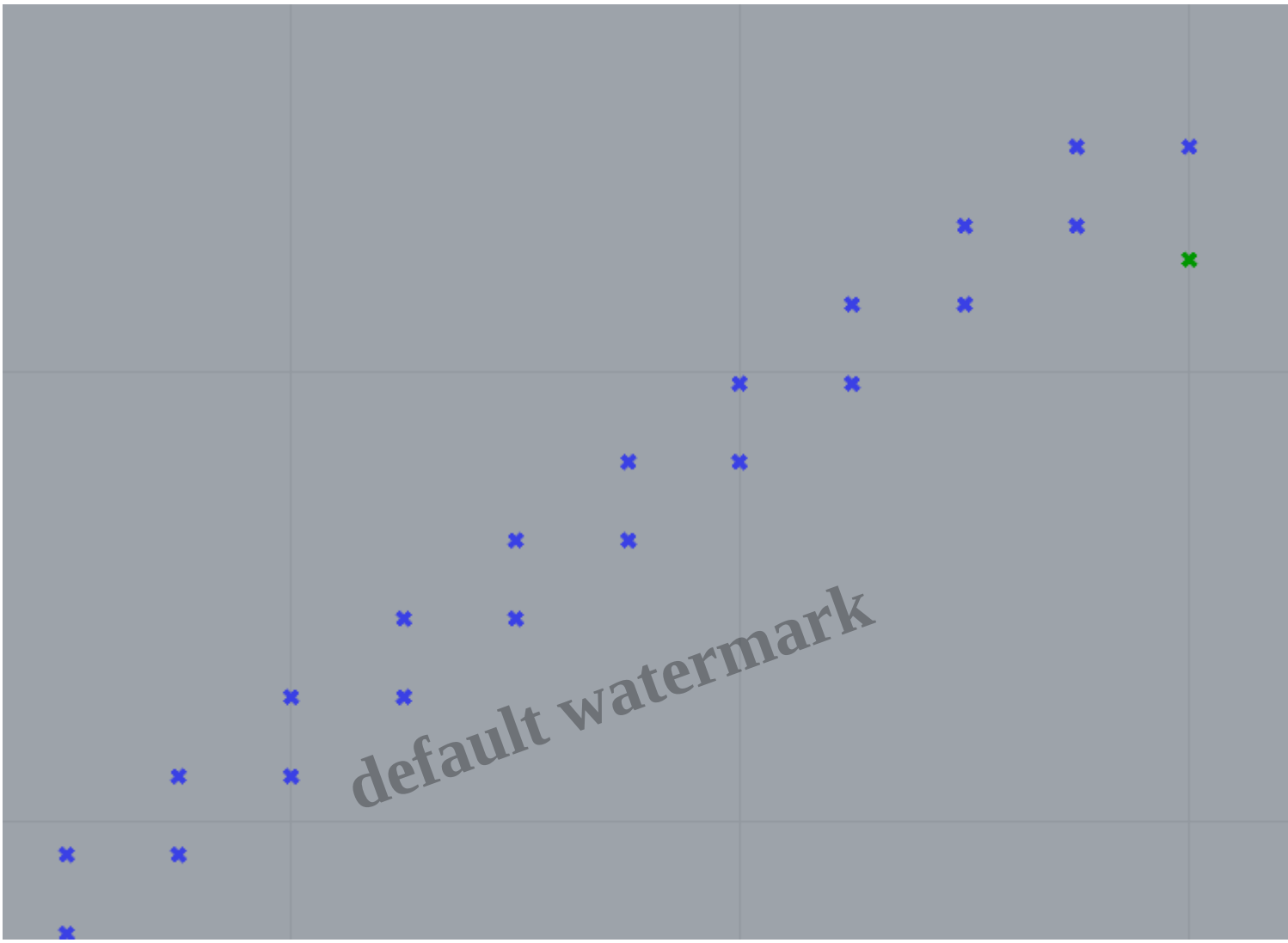
Unit Z's F input is *Slab thickness.* Connnect it to the according *Number Slider:*



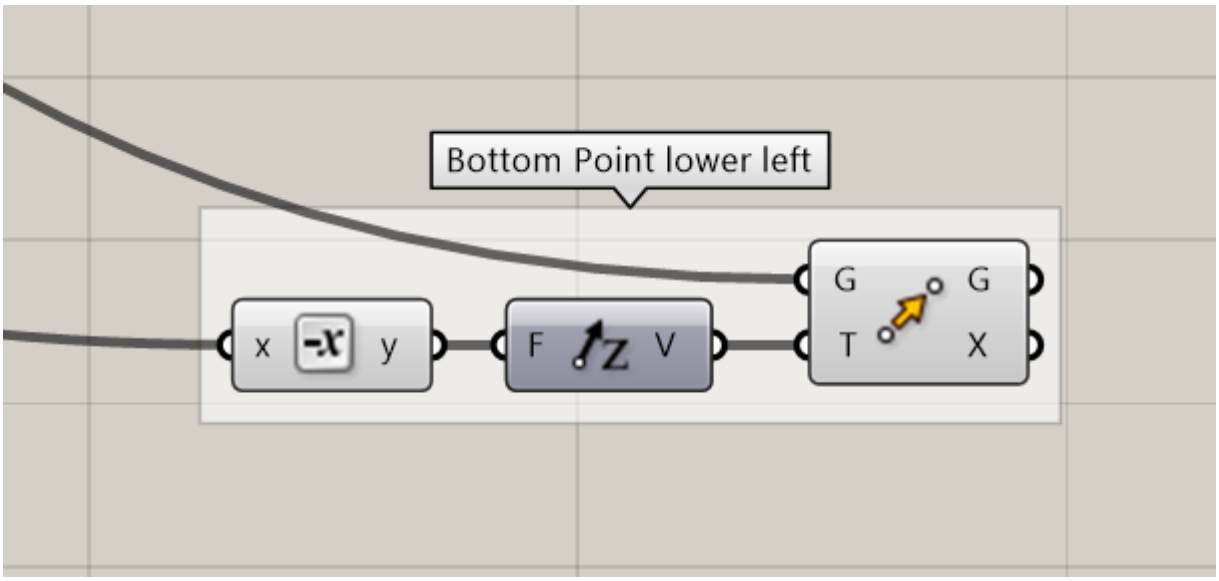
As our point has to move down however we need a negative of our *Slab thickness* output. Grab a *Negative* component and put it between *Slab thickness Number Slider* and *Unit Z:*



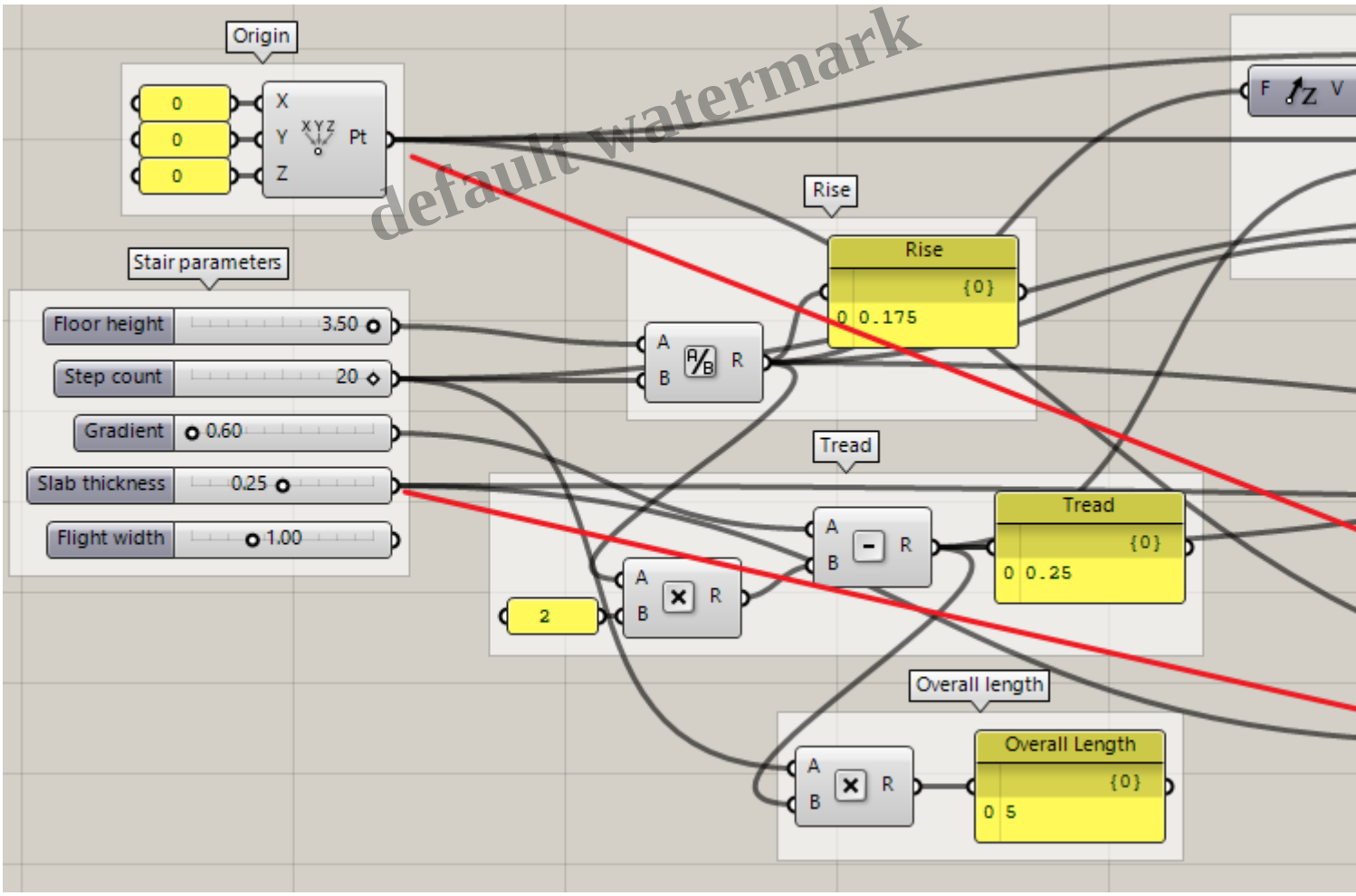
Here we are:



We need exactly this kind of point at the lower left of the stairs too. All we need to do ist to take our *Origin* point and move (and copy) it downwards in *Z*-direction, its offset is *Slab thickness again:*



Move component's *G* input is our *Origin* point, *Negative's X* input is *Slab thickness:*



Here we are, again:

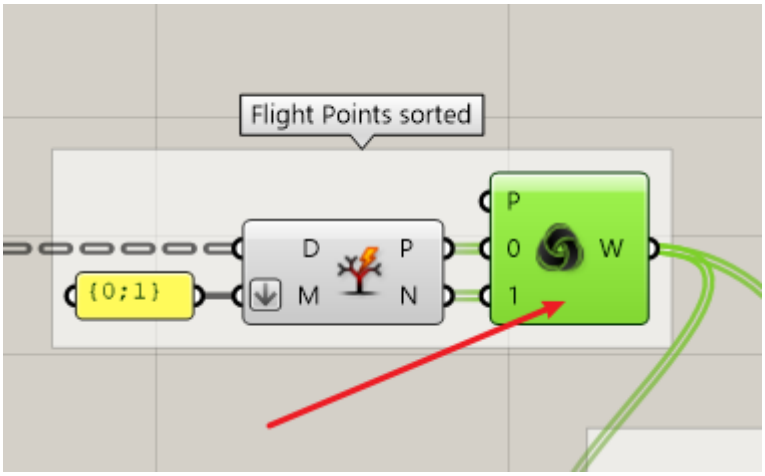
See my article on Starting with Grasshopper. You'll find more learning resources here.

Connecting the points <

We have all the points necessary to construct our stair. Before drawing the 2d-contour however we should combine our points first and check if the sequence is correct.

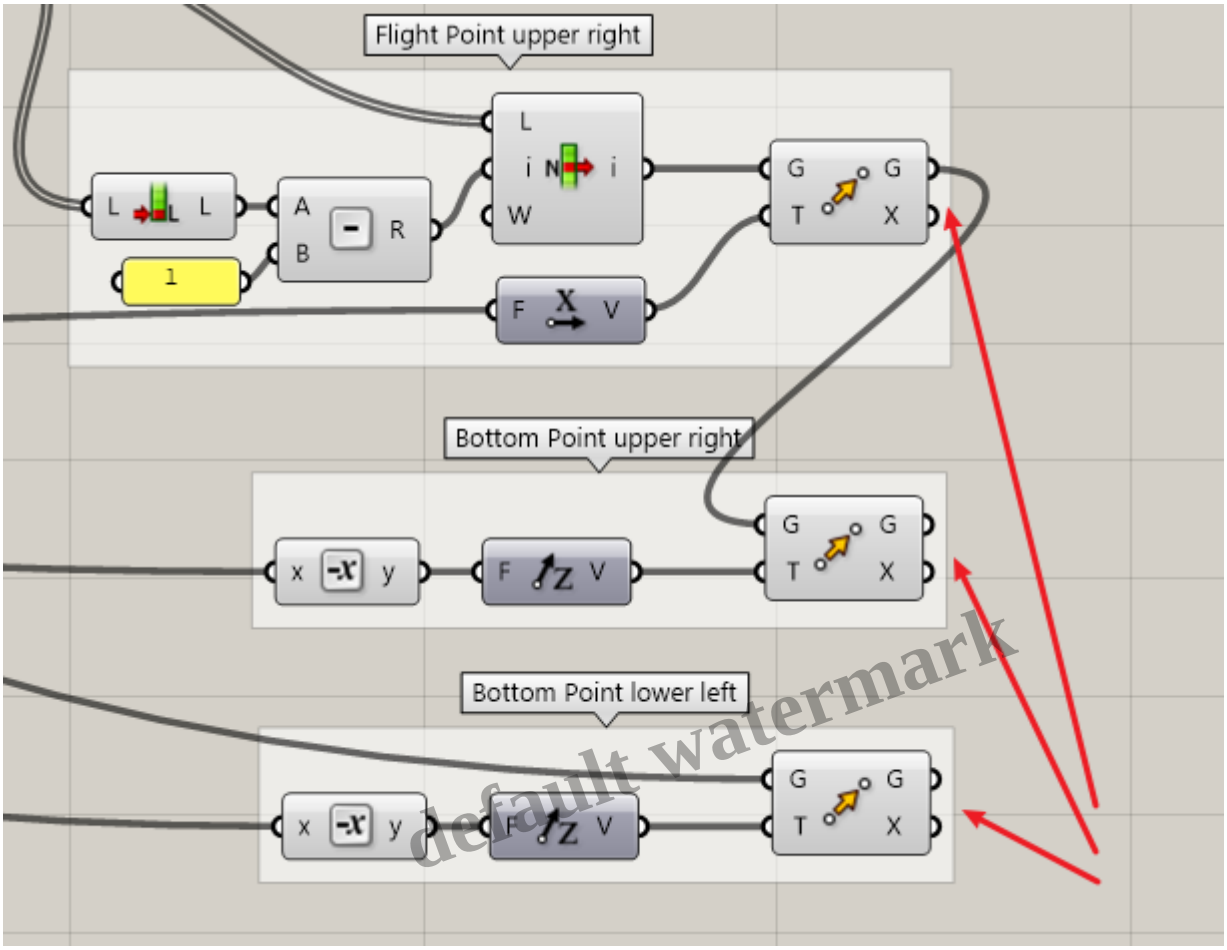
We have a group of points representing the flight upperside zigzag contour:

The according component is our *Weave:*

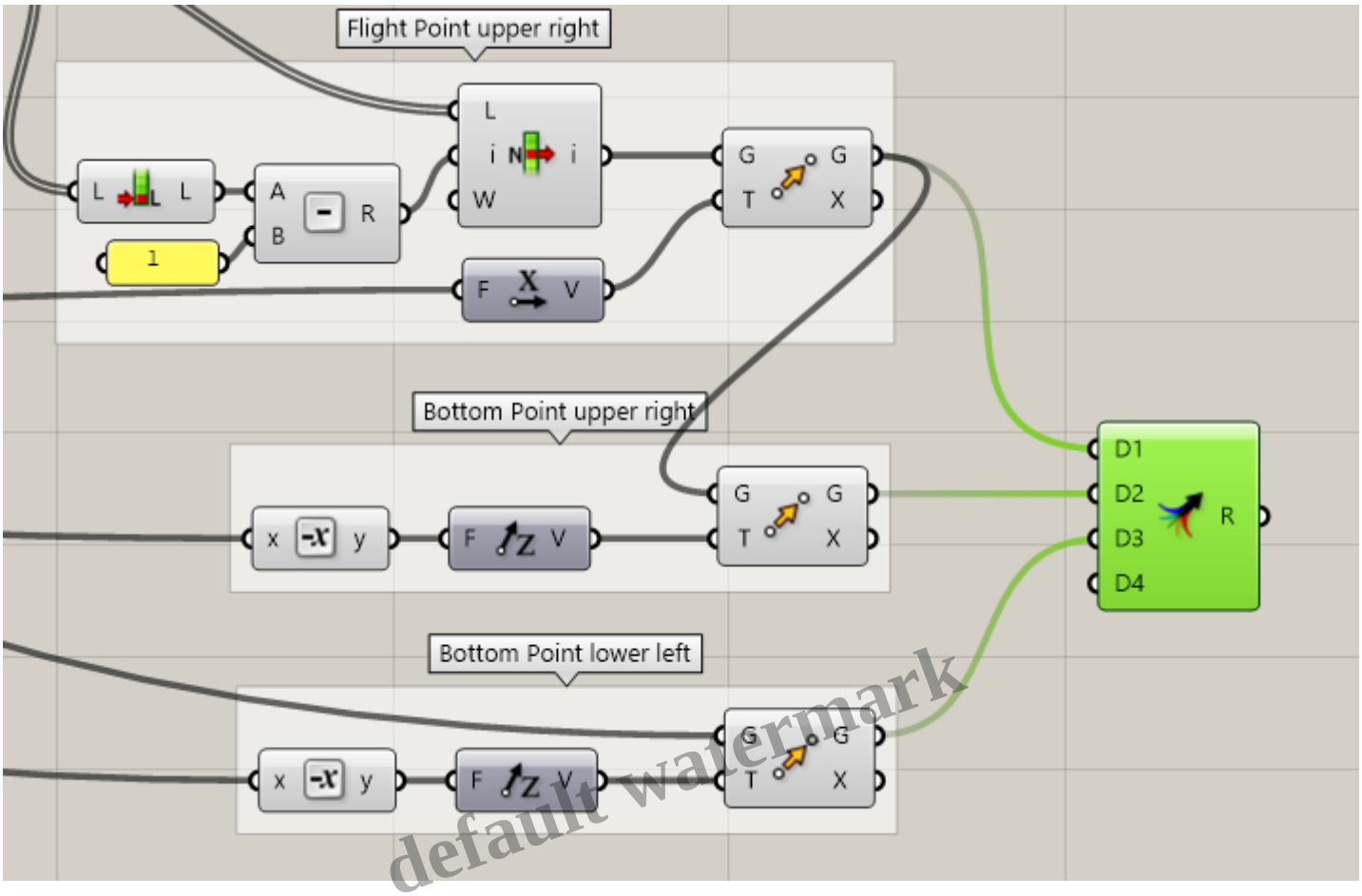


Then we have 3 single points:

They are represented by these *Move* components:



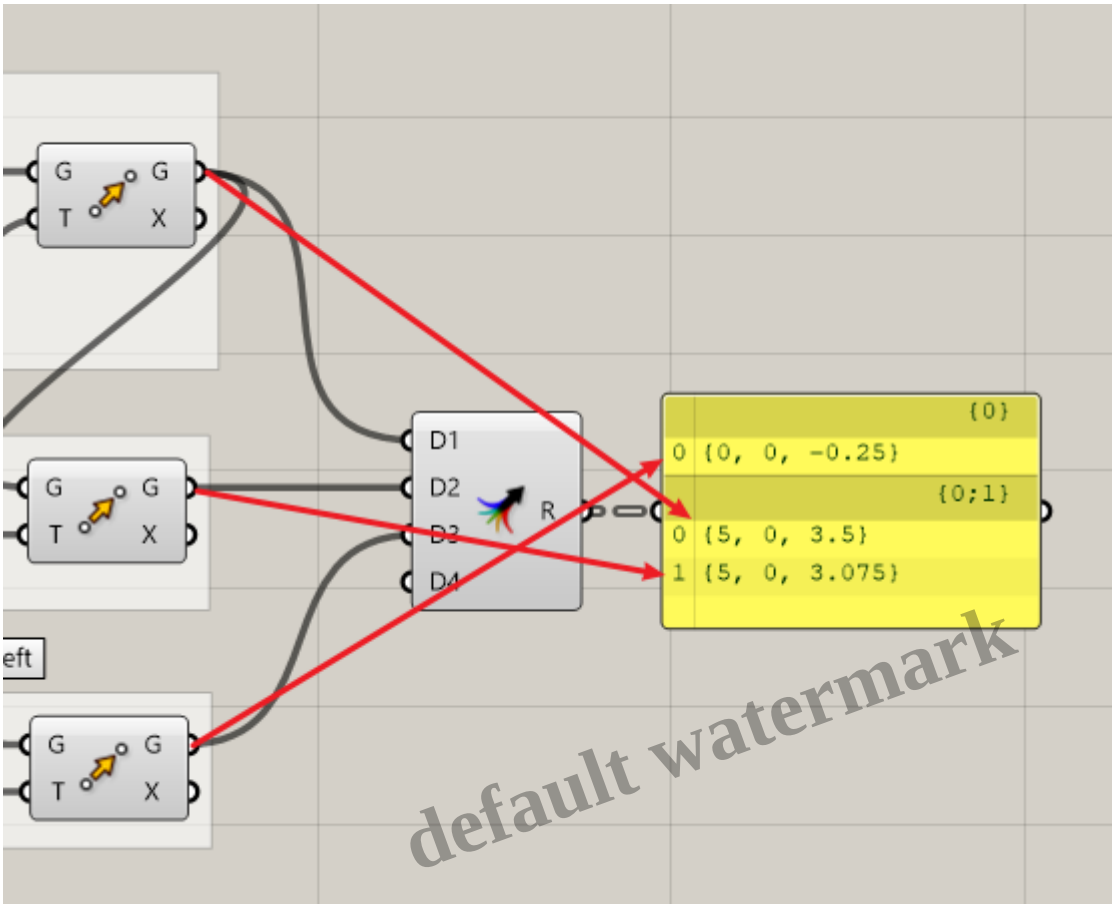
To be safe for our contour construction we should merge these 4 point outputs into 1 list. As the *Weave* output already is a list we first combine the 3 single points into another list using the *Merge* component. Just plug the point *Move* outputs into *Merge's* inputs – the input list grows as soon as the first *2* inputs are filled:



Before we utilize this new list let's check if it's got the right order. What we want is a point order that goes from the upper right point to the point beneath it to the point in the bottom left corner – the one below our Origin point:

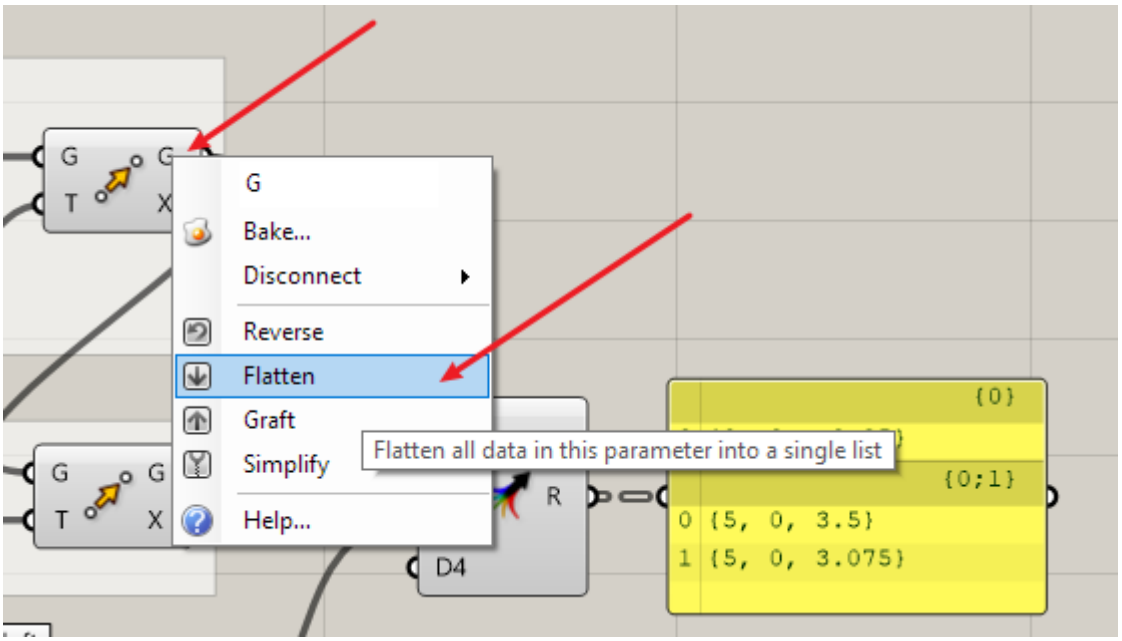

default watermark

In an attached panel we see that this is not the case:

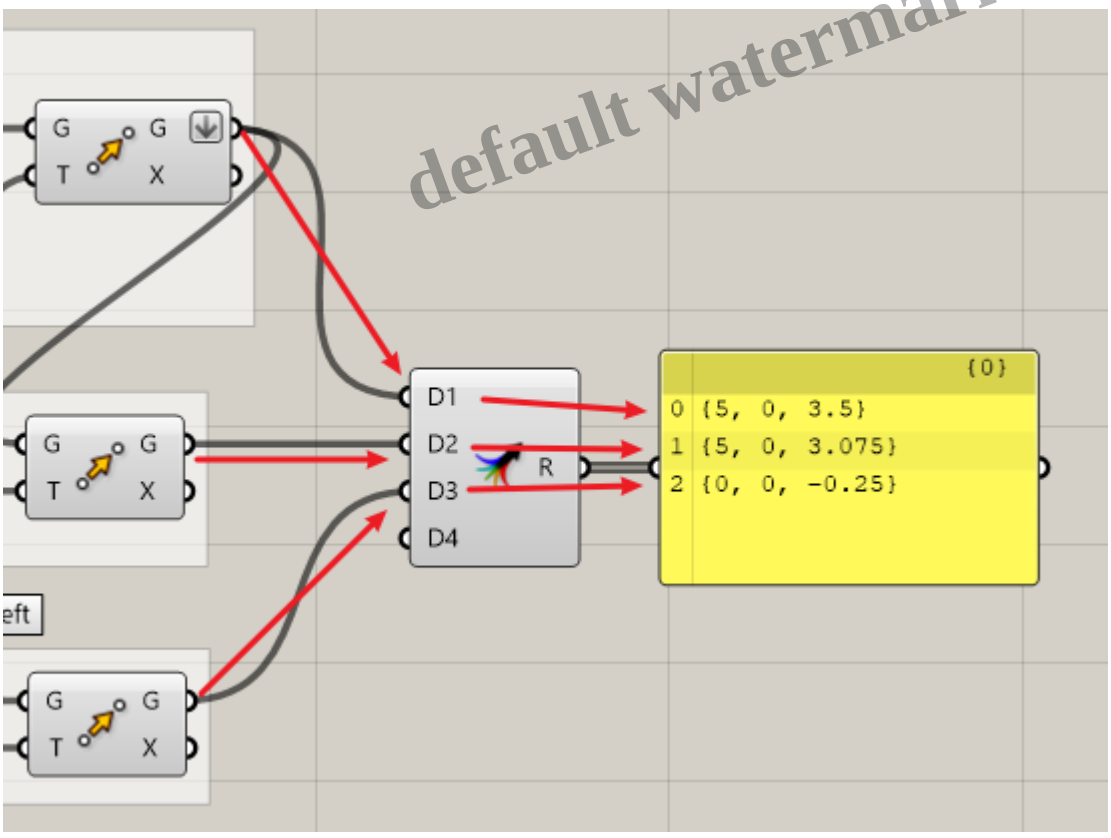


Why? Based on the way we constructed them *Grasshopper* has combined the *2* upper points in *1* path which results in an unordered point list. The solution is to dissolve the upper point *path {0;1}.* How?

Just right-click the according *G*-output of one of the upper *Move* components and choose *Flatten:*



And voilá – the list is ordered. Mind you: Now the list follows the order of the *Merge* inputs:

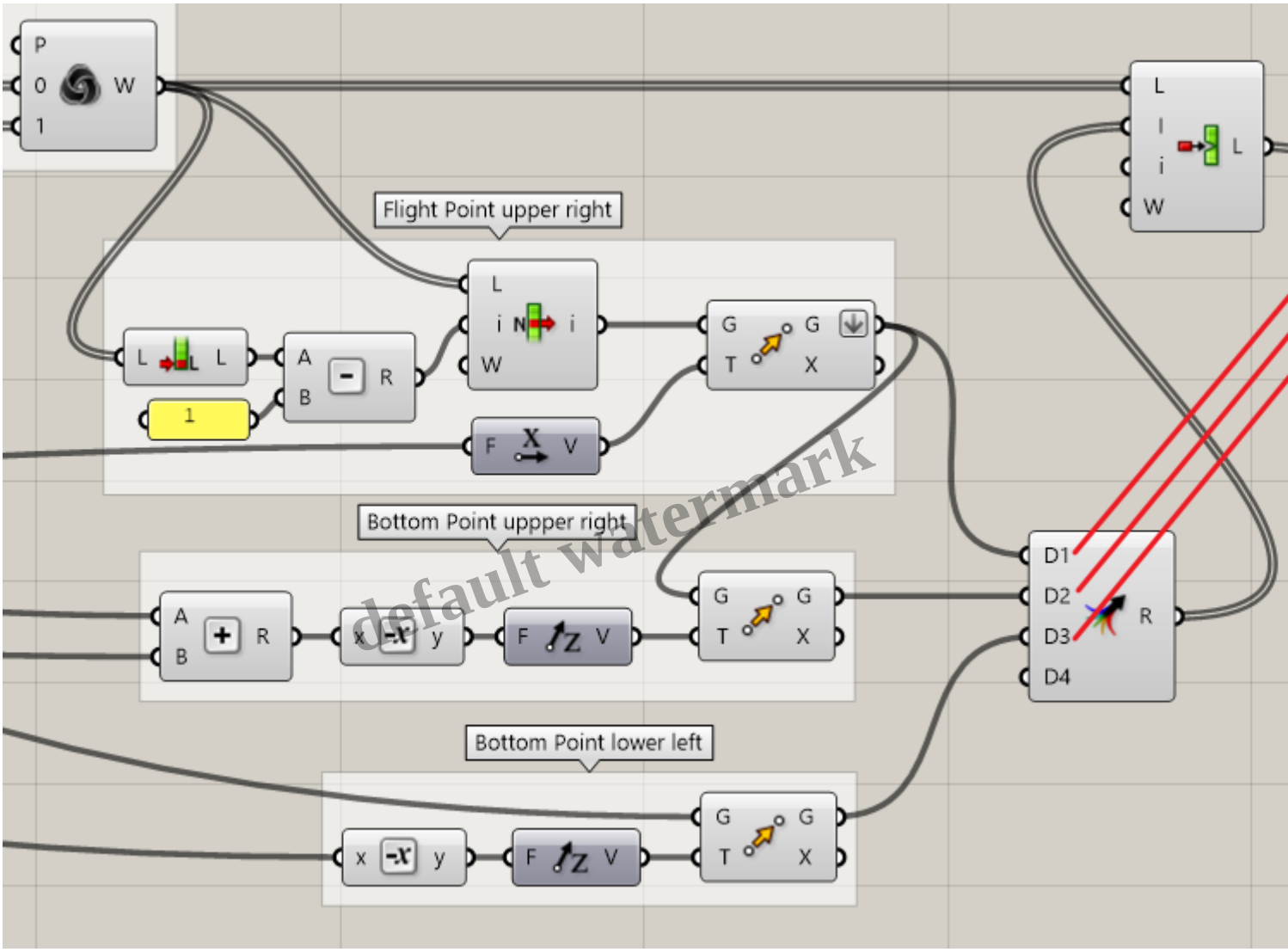


Now we have only to deal with *2* point lists – the shorter one (that we just produced) is supposed to be appended to the long one produced by *Weave.*

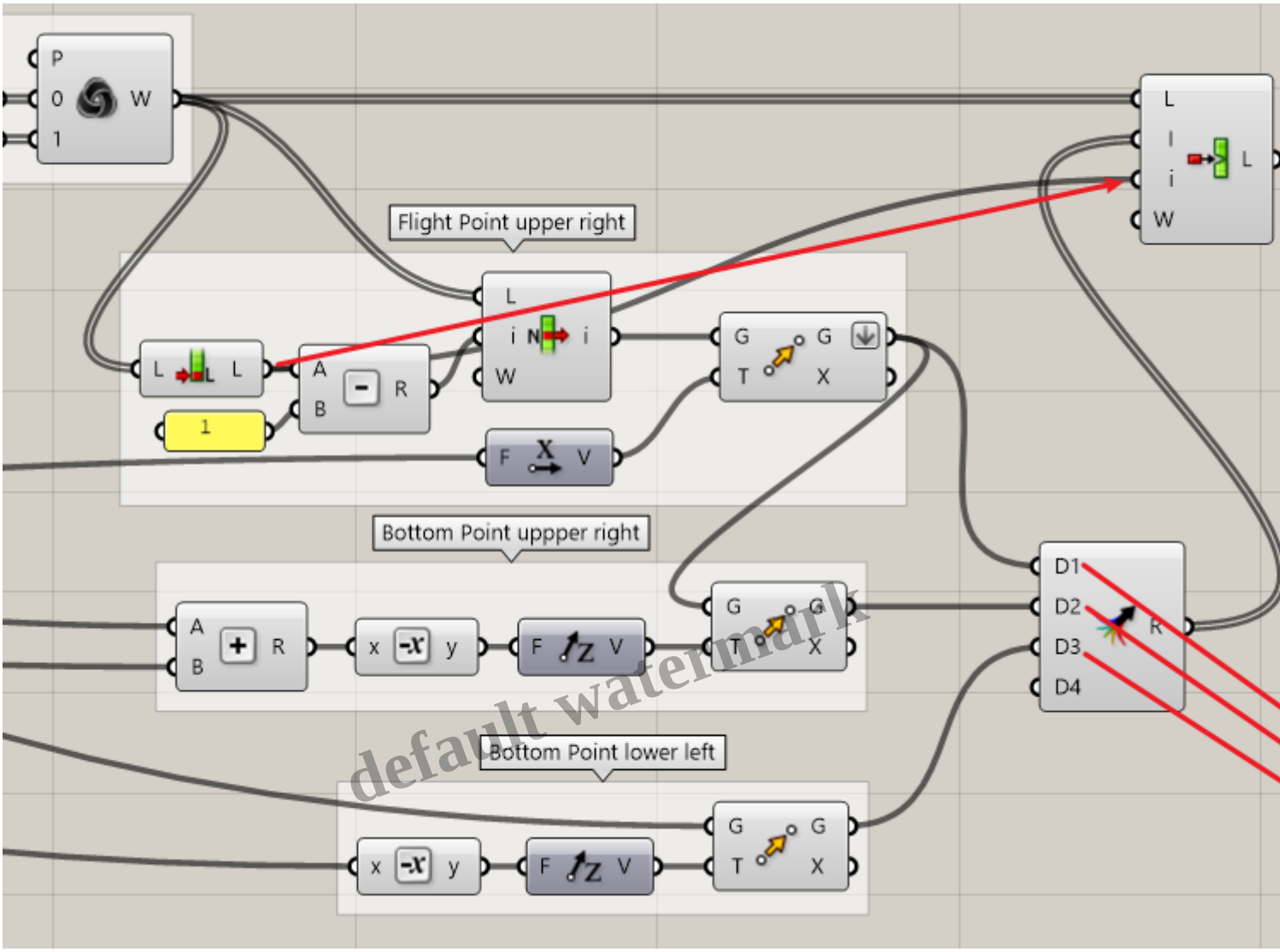
Use an *Insert Items* component:

- Plug the *Weave* output into the *List* input of the *Insert Items* component
- Plug the *Merge* output into the *Item* input of the *Insert Items* component

The result is correct – the group of 3 points is added at the beginning producing a correct point order:



However I would prefer them to be appended at the end to give my rise/tread-chain of points priority. To achieve this all you have to do is change the *Index* input of the *Insert Items* component. This has to be *List Length* value of the *Weave* component:

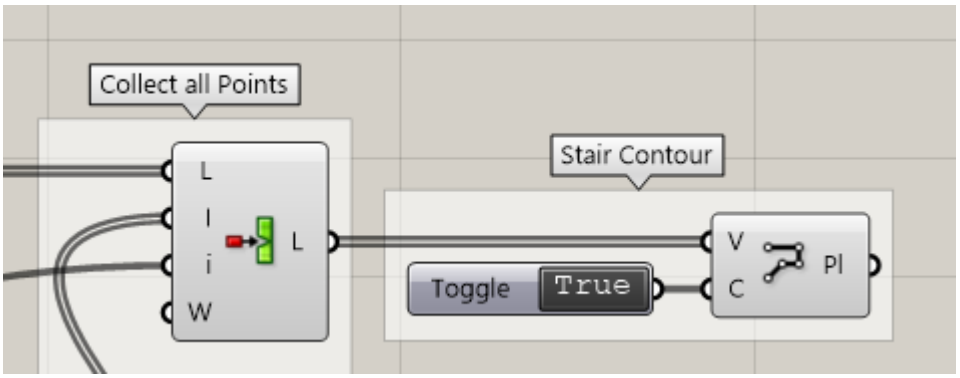


The *Index* input marks the main list's position where the second list begins – so there's that.

See my article on Starting with Grasshopper. You'll find more learning resources here.

Drawing the Stair Contour <

That's easy: Use a *Polyline* component and plug the last component that collected points (in this case: our *Insert Items* component) into *Polyline's Vertex* input. To close the contour (because there is a gap we did not fill) feed a *Boolean Toggle* into *Polyline's C* input. Double-click to set it to *True:*



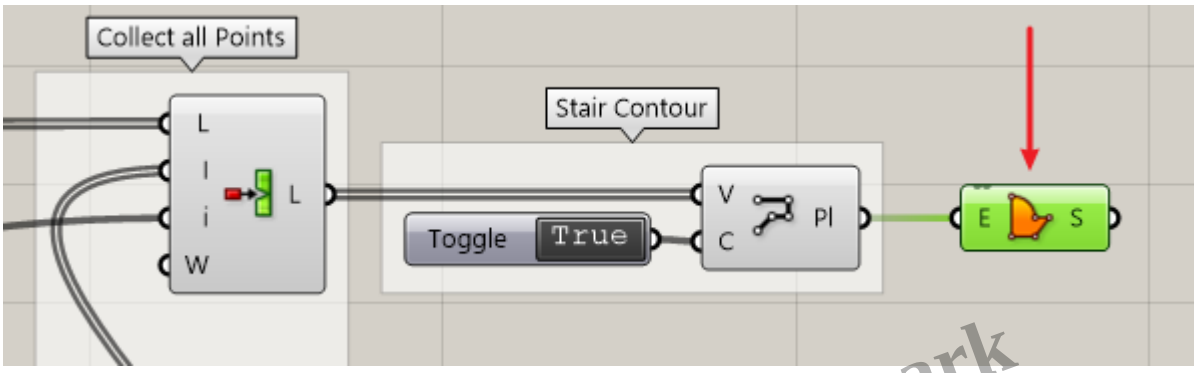
Now our stair contour looks fine:

See my article on Starting with Grasshopper. You'll find more learning resources here.

Filling the Stair Contour <

Now that our stair contour is complete and closed we can fill it with a surface. This is a prerequisite to produce a solid extrusion finally.

Use the *Boundary Surfaces* component and connect it to our *Polyline* output:



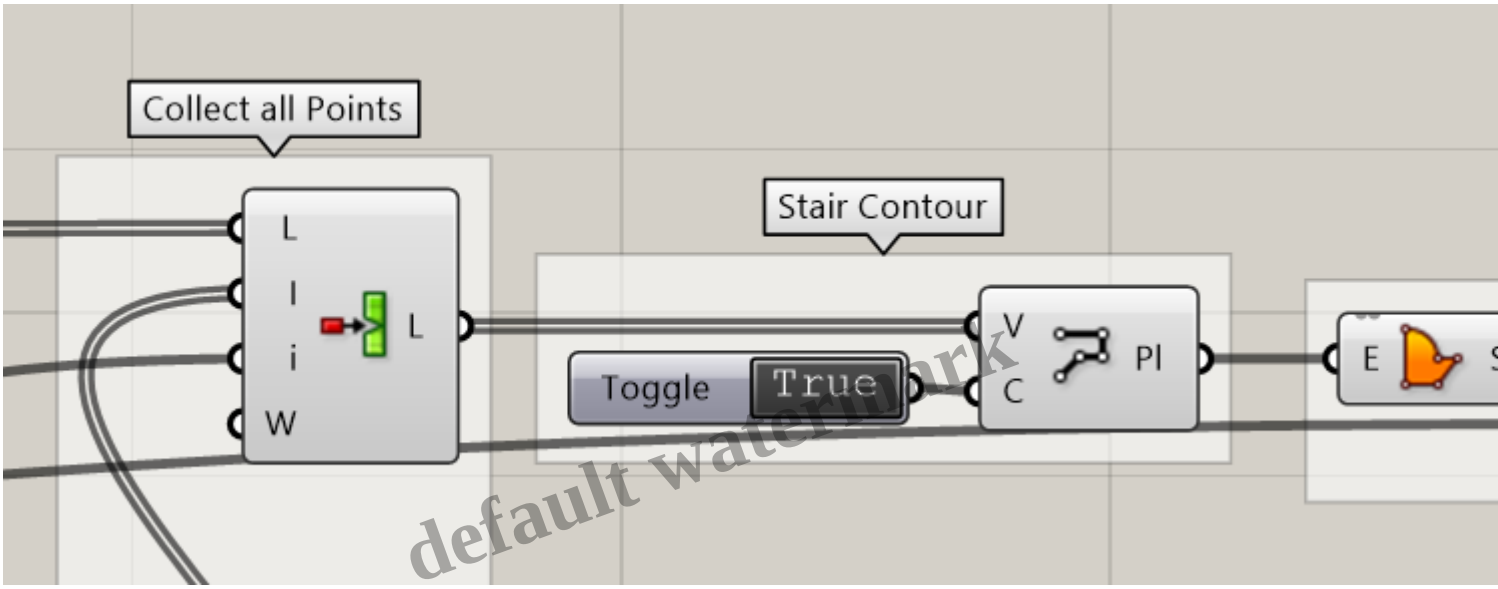
This is the result:

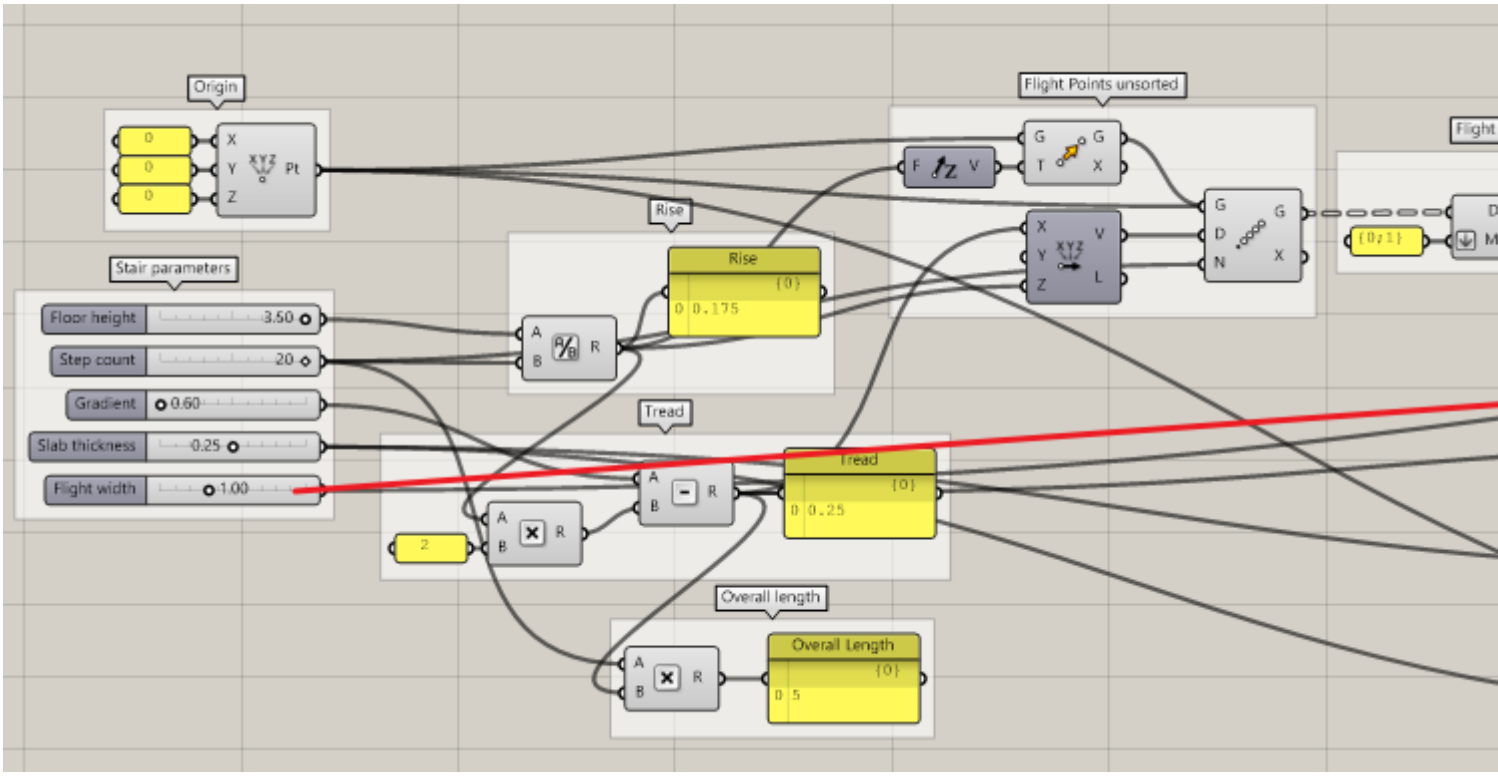
This will be the base for our final extrusion.

See my article on Starting with Grasshopper. You'll find more learning resources here.

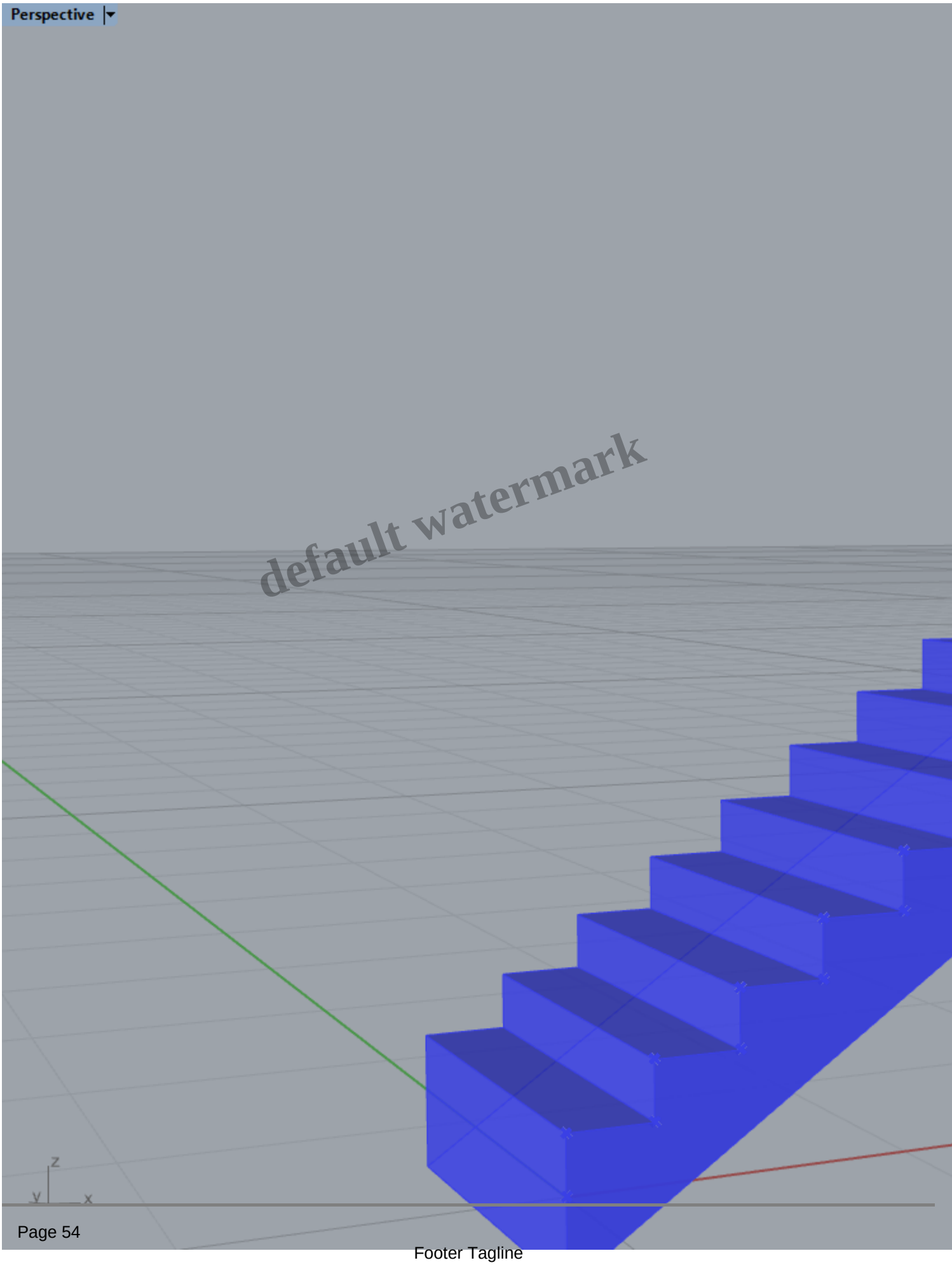
Finale: The 3D Stairs <

Also, very simple – use an *Extrude* component and connect its *B* input to *Surfaces'* output. Plug a *Unit Y* vector component into *Extrude's D* input and feed it with – at last! – our *Flight width Number Slider:*





And this is the result. Feel free to play around with your input sliders ?


Perspective
default watermark
z
y
x

Roundup <

That's it. I think the main takeaway from this tutorial is to break down a task into parts. In this case, they are:

- Setting parameters that constitute the stair geometry
- Constructing contour points
- Sorting points and combine point lists
- Drawing 2D
- Drawing 3D

And, there's also a form-developing concept I chose: To start stair development from its elevation or section. A said in the beginning, this is certainly only one of many possible strategies.

See my article on Starting with Grasshopper. You'll find more learning resources here.

Find my Grasshopper Stair Definition here.

Category

1. Rhino/Grasshoppper

Tags

1. BIM Model
2. Parametric Modeling

Date Created
October 2018
Author
hsondermanncom