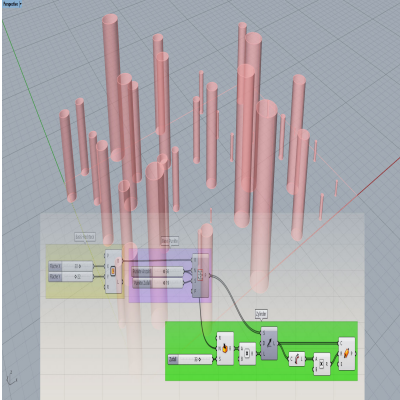


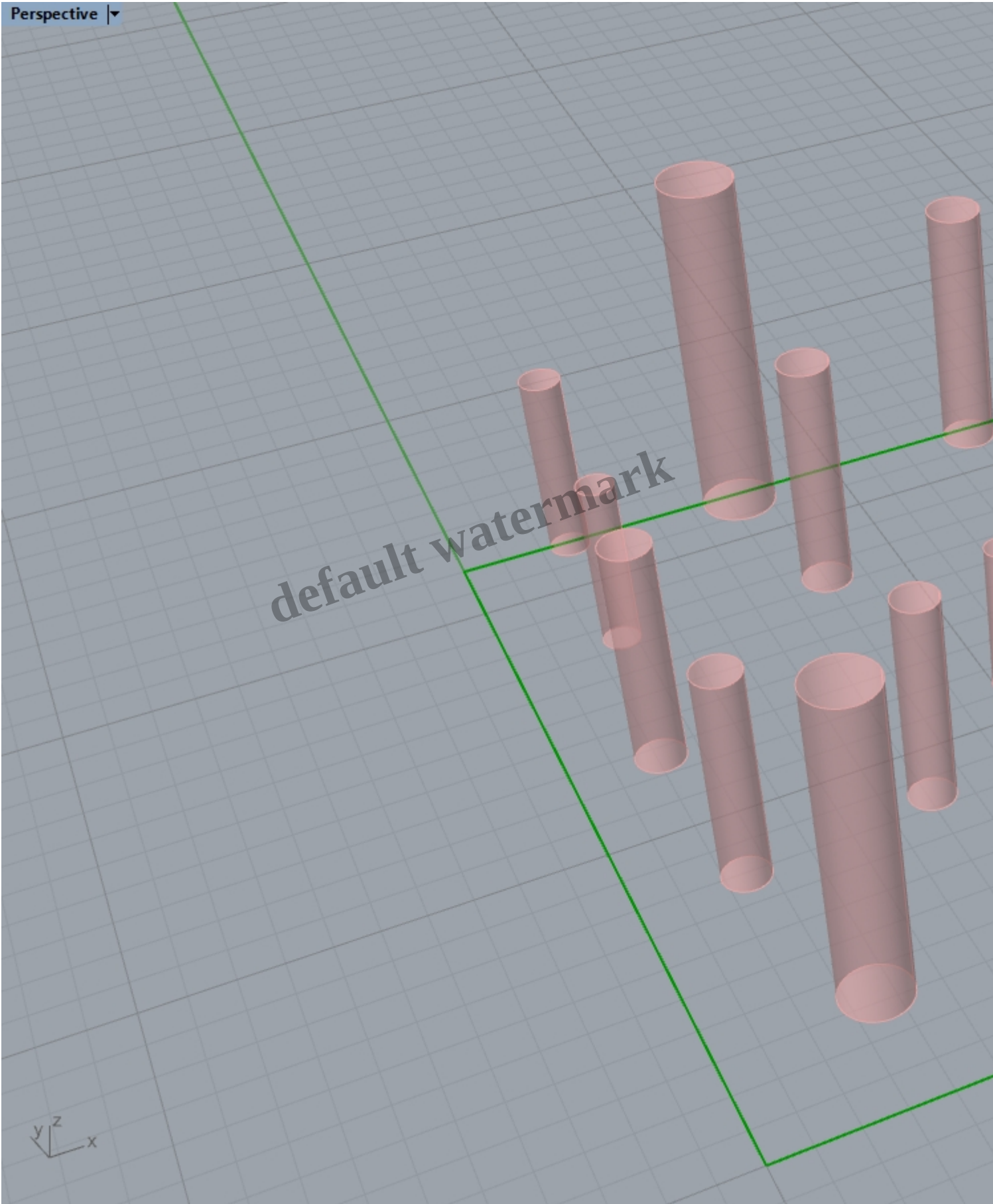


Rhino Grasshopper: Random Geometry Distribution

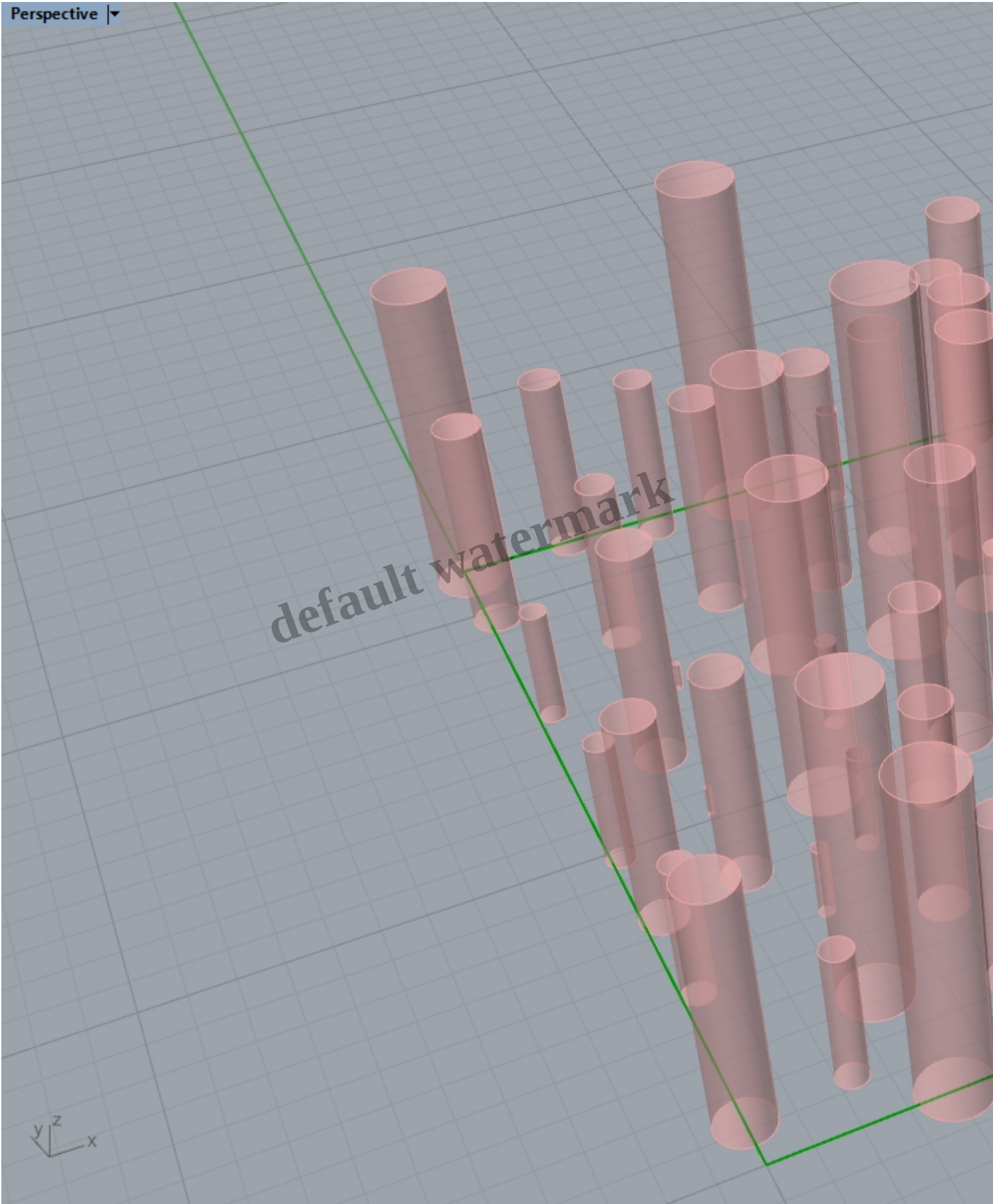
Description

I wondered if I could produce the kind of random geometry in *Rhino Grasshopper* that I can generate with *Cinema4D*'s cloner tool. As shown in the images above, it's about distributing objects over a defined surface allowing for random positioning, scaling and rotating. Let me show you how it works in Rhino Grasshopper.

default watermark

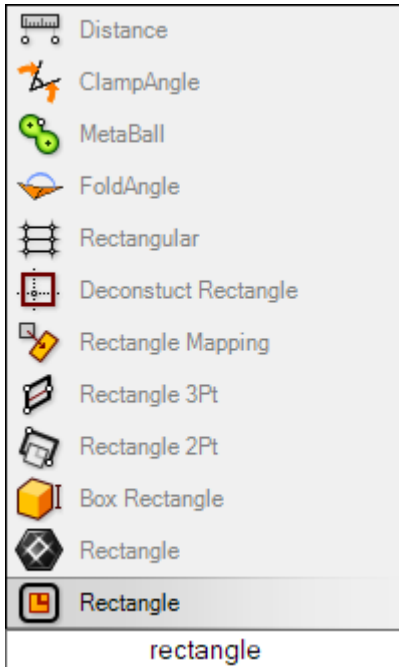


default watermark



Create a rectangle

First of all we need a base surface, to keep things simple in this case it will be a regular rectangle. Double-click on the Grasshopper canvas and write *rectangle*:



Now that you have your *Rectangle* component produce 2 *Number Sliders*. Do this by double clicking on the canvas and write: *1<20<50*:

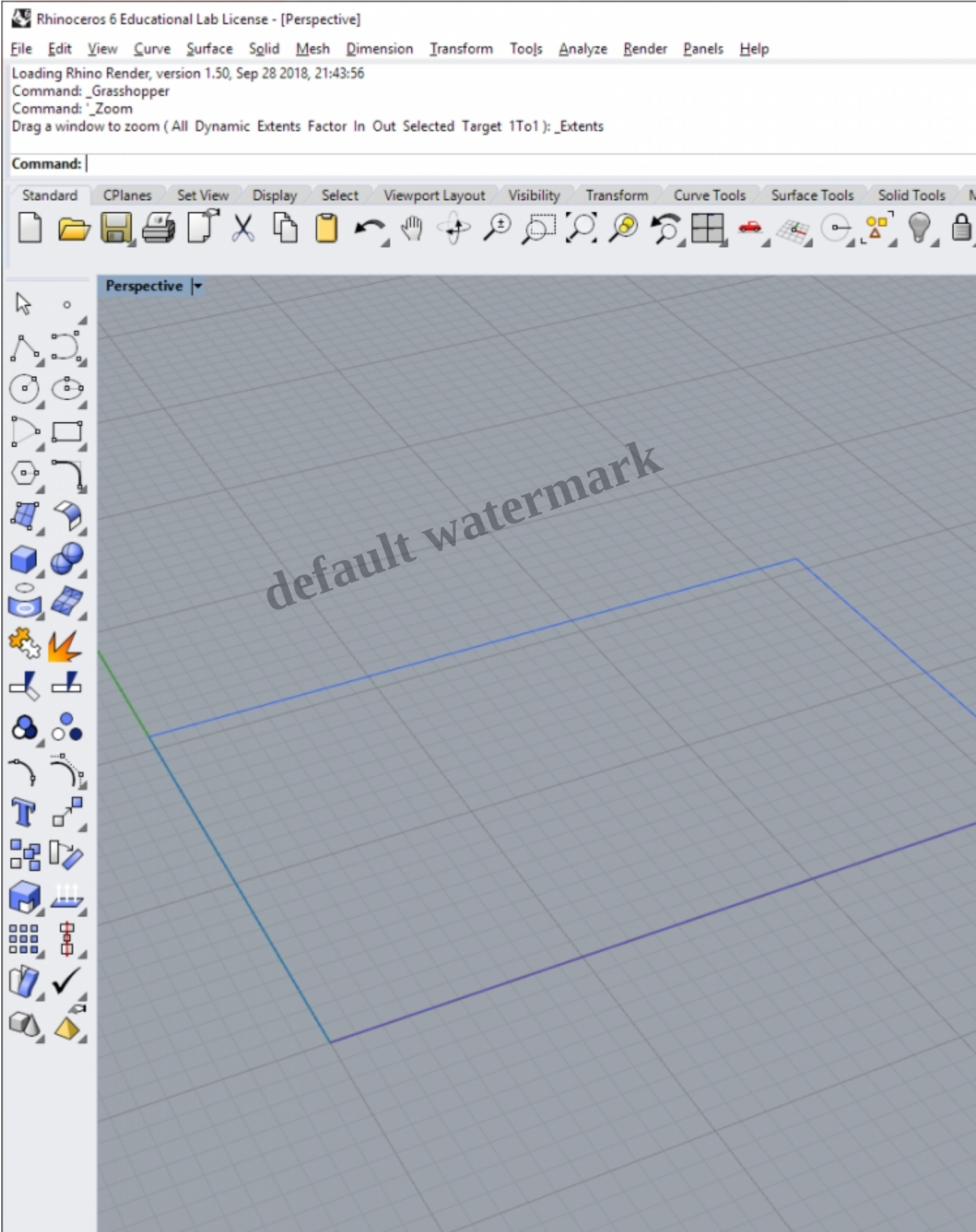


When you hit *Return* you'll see that GH has produced a number slider showing integers ranging from 1 to 50 and showing 25 as default. As I said above, do this twice and connect the two sliders to the X and Y-Inputs of the *Rectangle*.

Note: You can also produce one slider only and *Alt-Drag* it to get an exact copy of it. (Start dragging

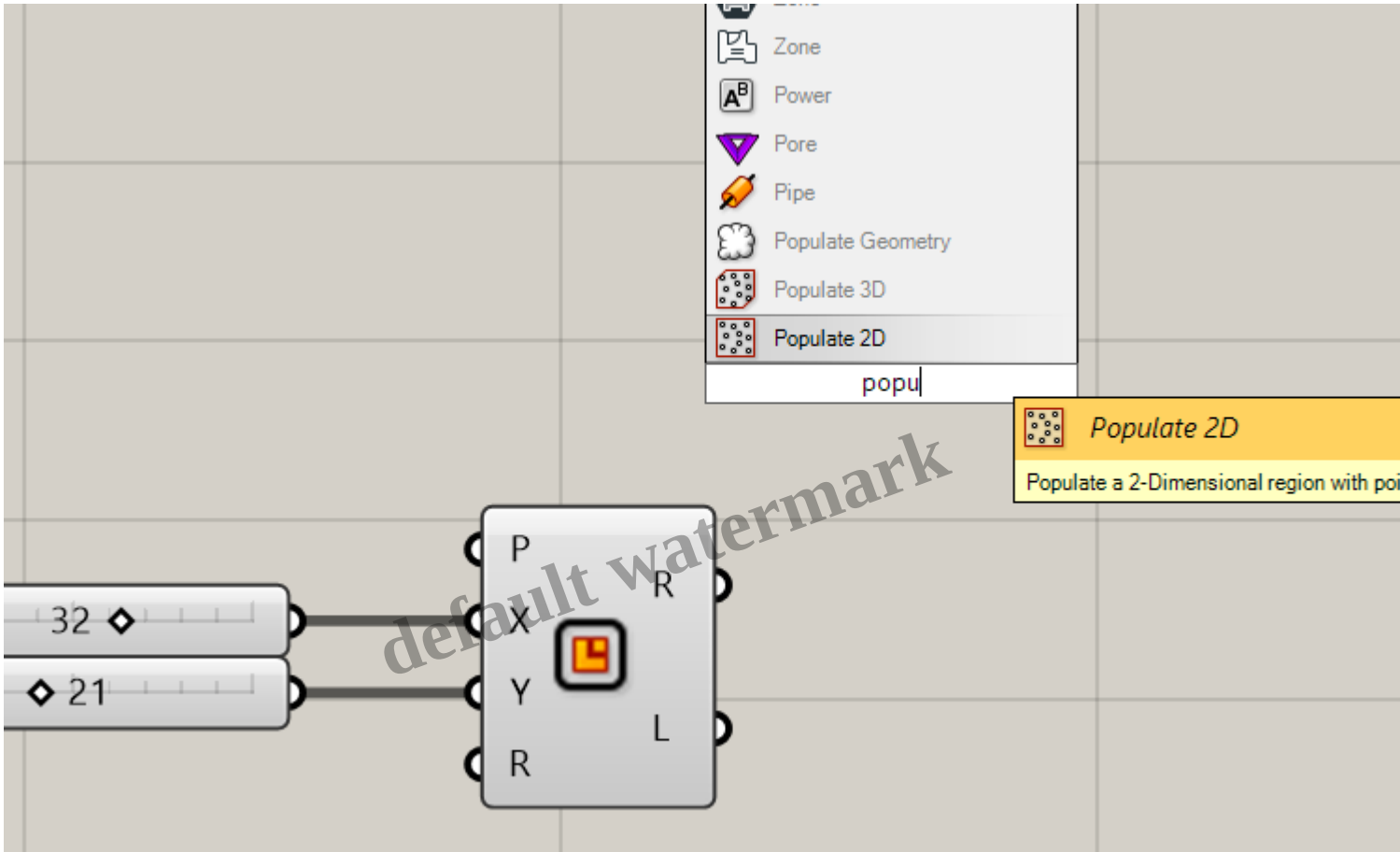
and *then* press *Alt* – if you press *Alt* first you'll get the Moses effect.) Now you can adjust the base rectangle's size via your new *Number Sliders*:

default watermark

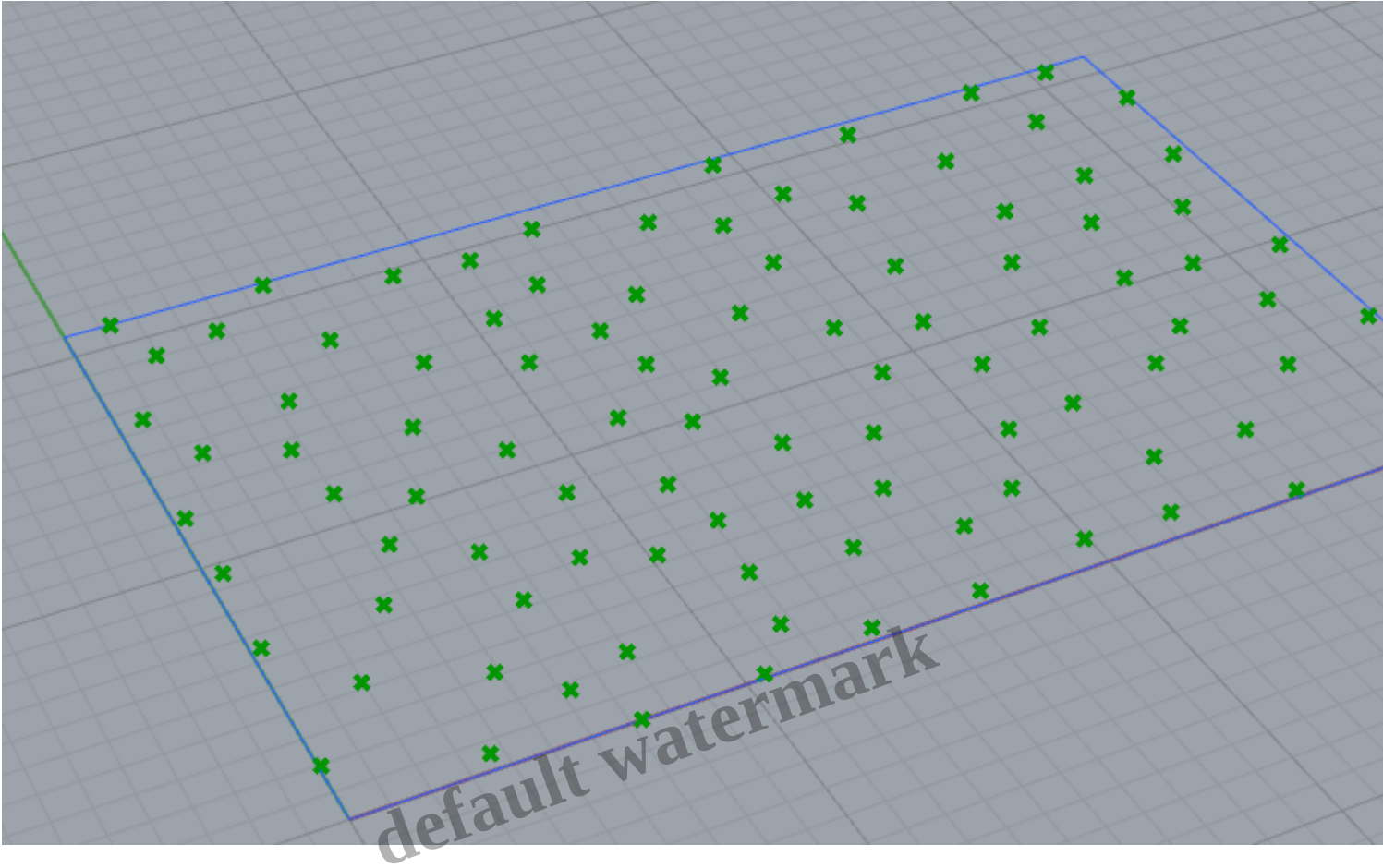


The Points

To distribute objects on this rectangle we start with points. They will serve as origins for our cylinders.
Produce a *Populate 2D*-Component:

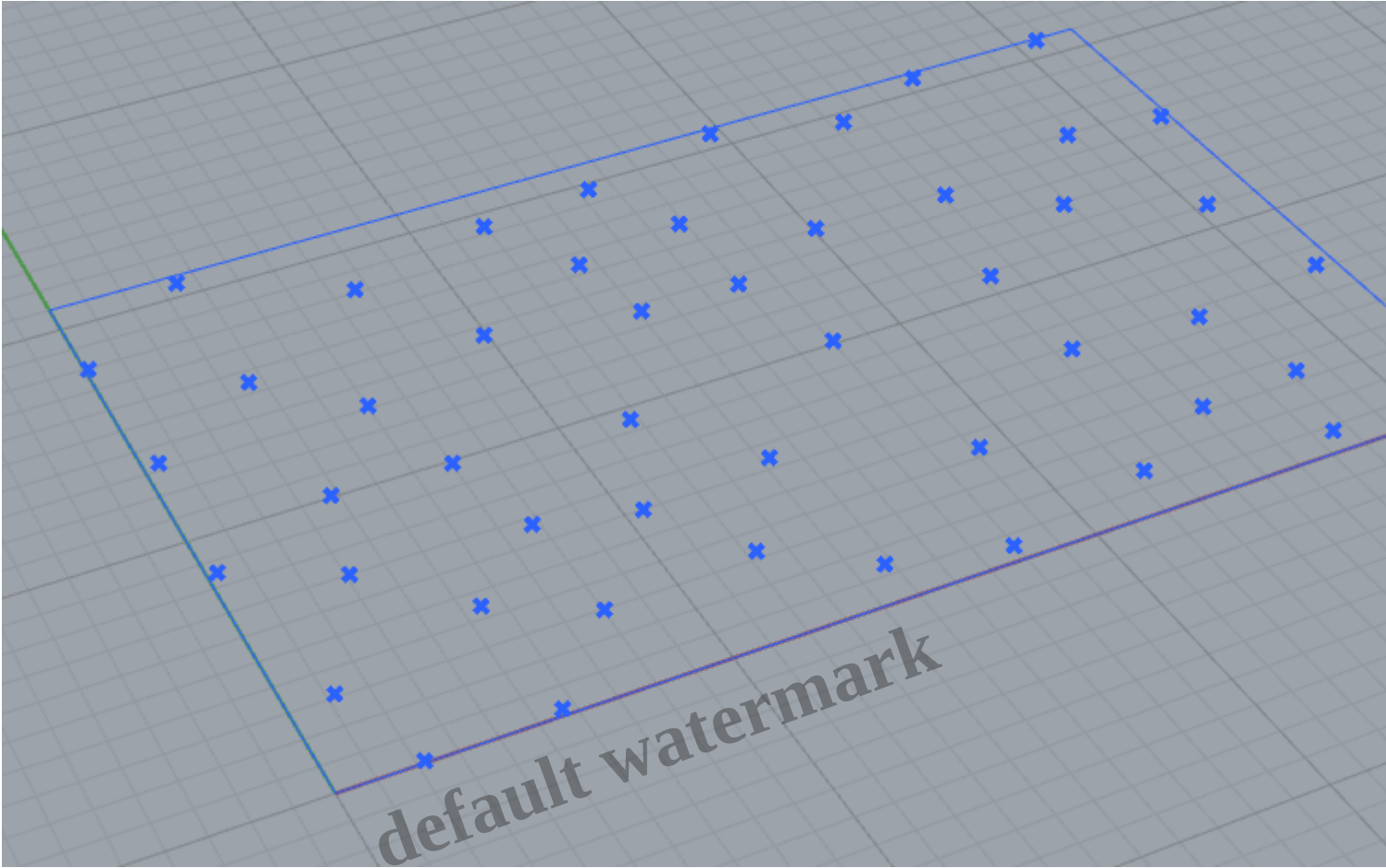


As you see this component produces random points on our *XY-Plane*. And it has an *R-Input* asking for a *Region* to use for its point distribution. Intuition tells you that you'll want to connect your rectangle's *R-Output* to this input. As a result, GH fills your rectangle accordingly:



A tool that distributes objects this way should offer variables for the objects' quantity and distribution pattern. In this case, the according Inputs of our *Populate 2D* component are *N* (for *Count*) and *S* (for *Seed*).

Again, a case for integer *Number Sliders*: Produce 2 more of them and connect them to these *N* and *S* inputs. As a suggestion, for *Count* I chose $1 < 50 < 100$ and for *Seed* $1 < 25 < 50$:

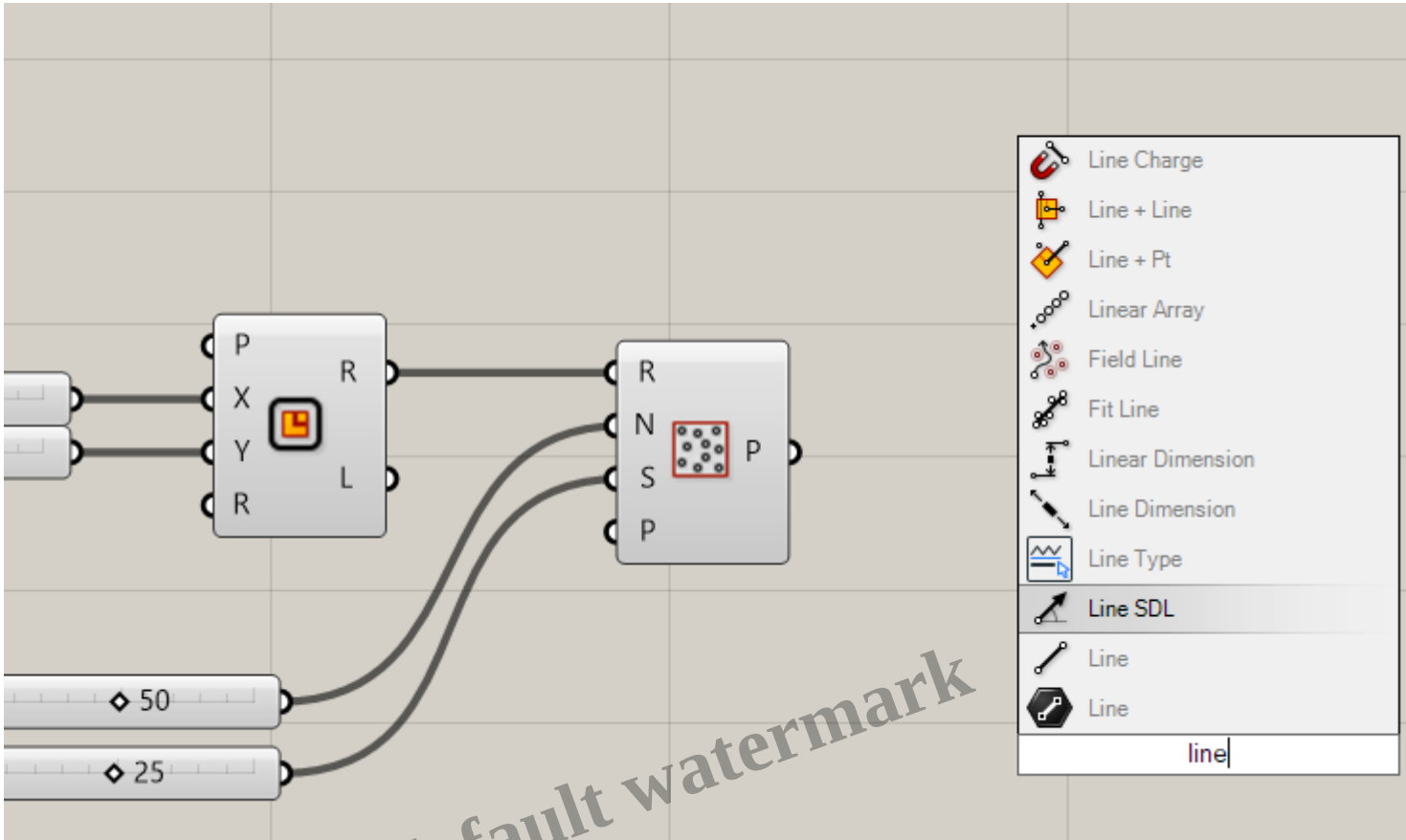


By the way, in case you didn't know: The *Number Sliders* on their left show the names of the inputs they are connected to – automatically.

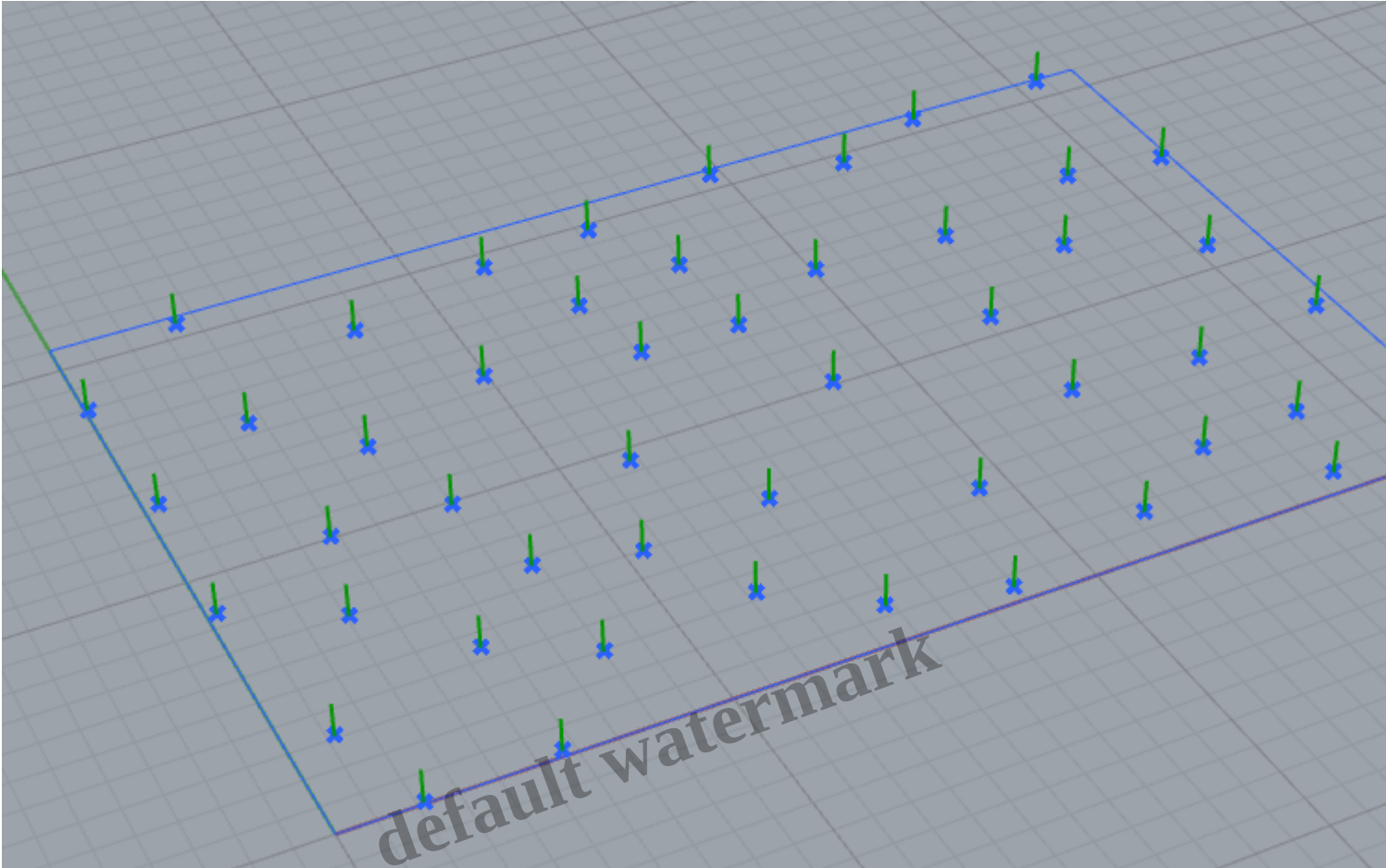
Anyway: Now you may play around with those sliders, producing random point topology on a randomized base rectangle.

The Cylinder Axes

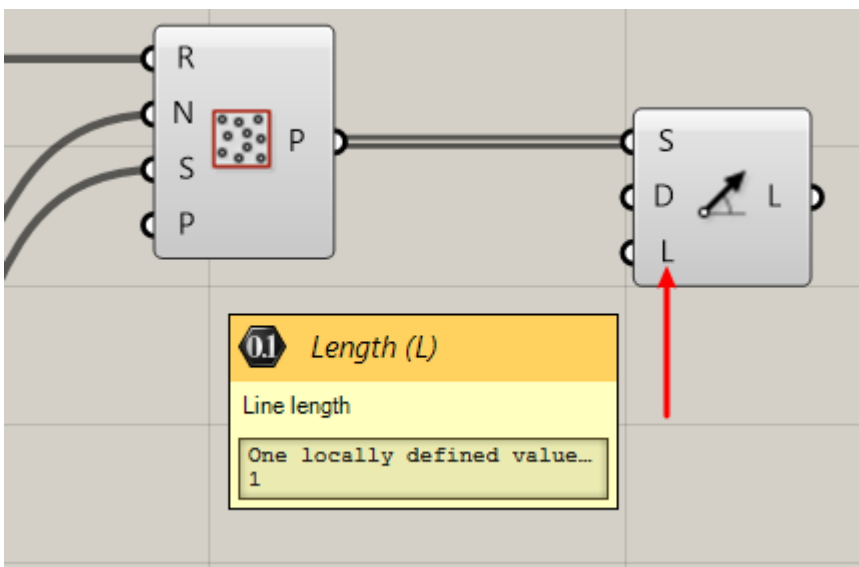
Starting from the points we want to see vertical lines serving as axes for cylinders. So produce a *Line SDL Component*:



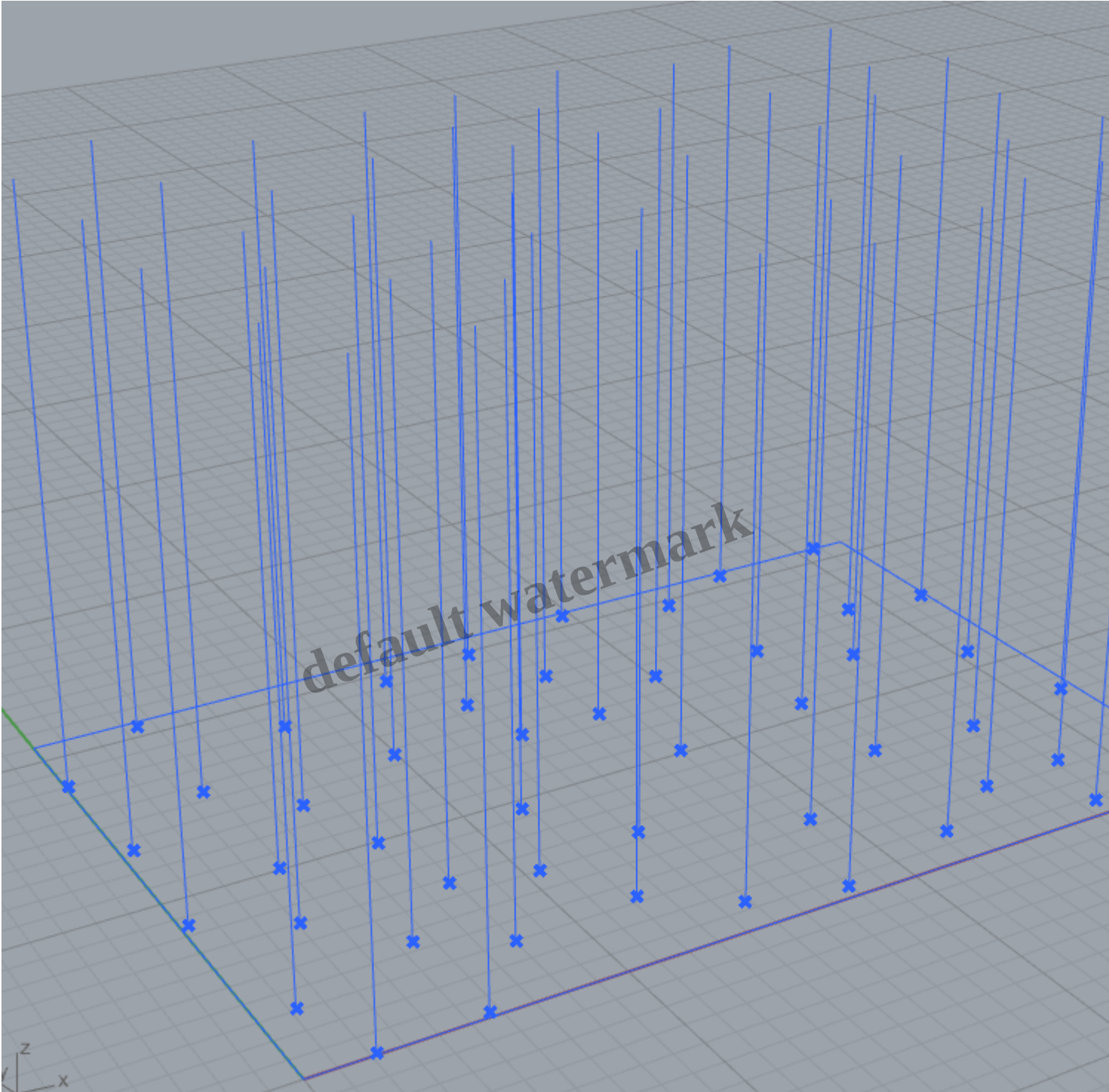
As you can see there are several *Line* components – but *Line SDL* is most suitable for our needs because it requires a *Start Point (S)*, a *Direction (D)* and a *Length (L)* as input. And, as you might imagine, our random points serve as those start points, so connect the *P-Output* of the *Populate* component to the *S-Input* of *Line SDL*. The result is a bit disappointing, our cylinder axes appear very short:



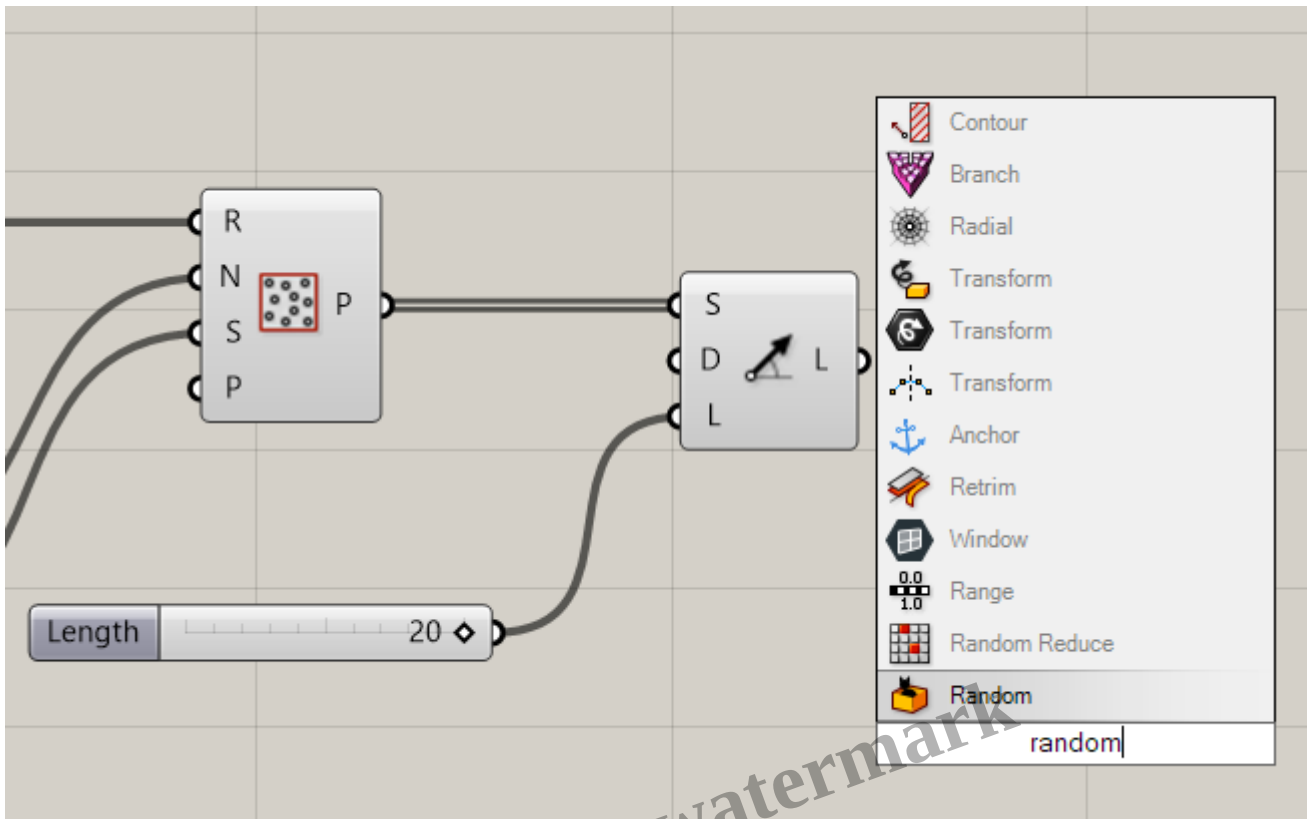
Well, let's do something about that using the *Length* input. When you mouse-hover over the letter *L* Grasshopper tells you the input uses a default value 1:



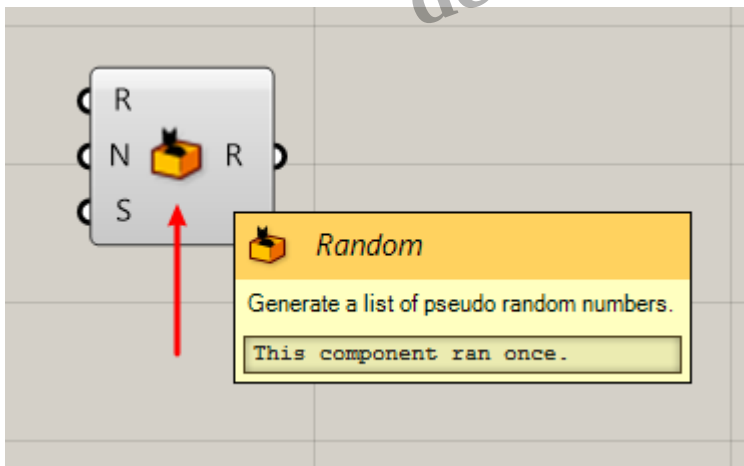
Now obviously *L* could use some more input than that. Place another *Number Slider* ($1 < 10 < 20$) and connect it to *L*. Now you have some decent axes showing up:



You feel the length of the axes could also vary in a random fashion? No problem: Place a *Random* component:

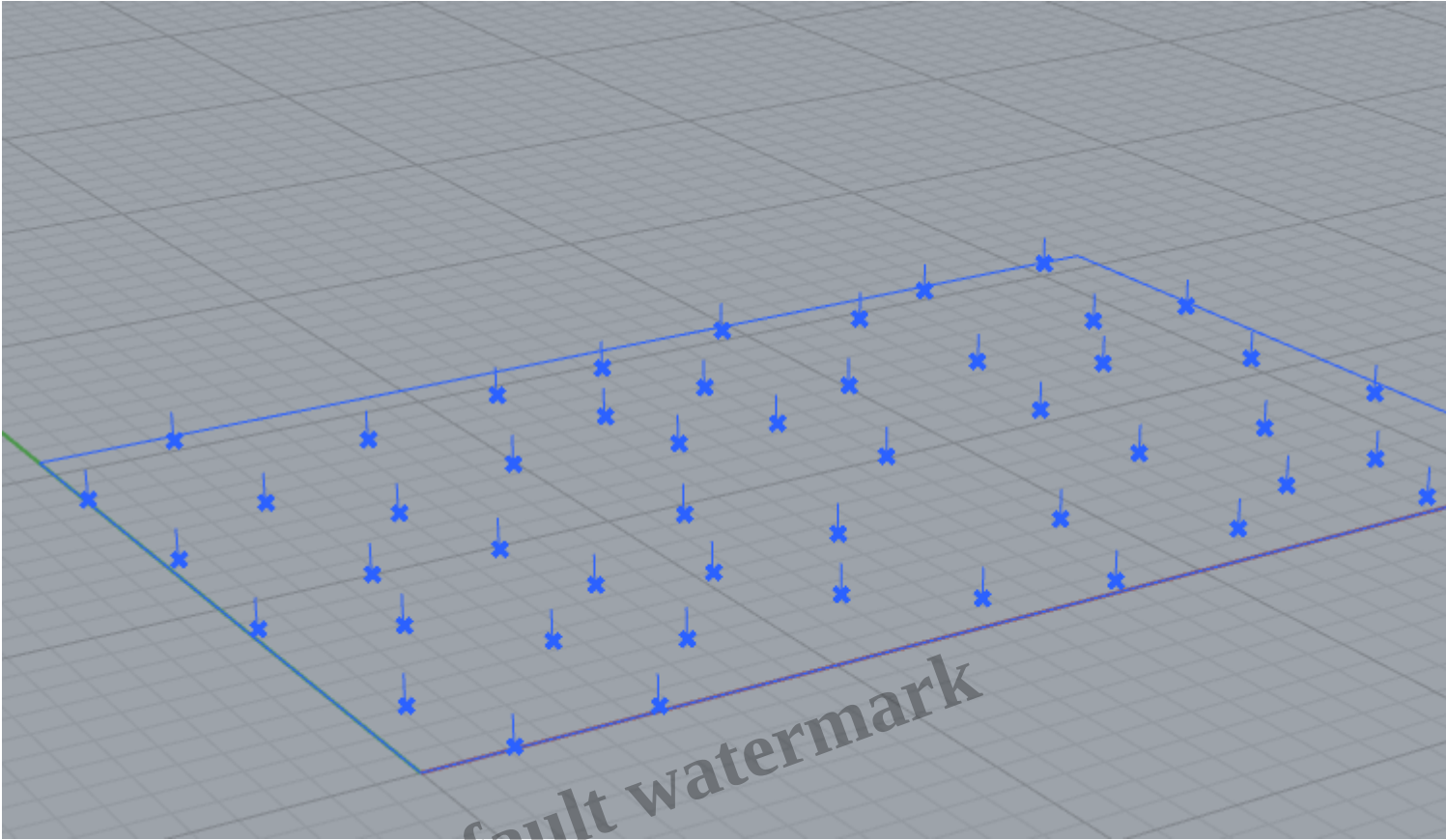


What does it do? It produces a series of random numbers:

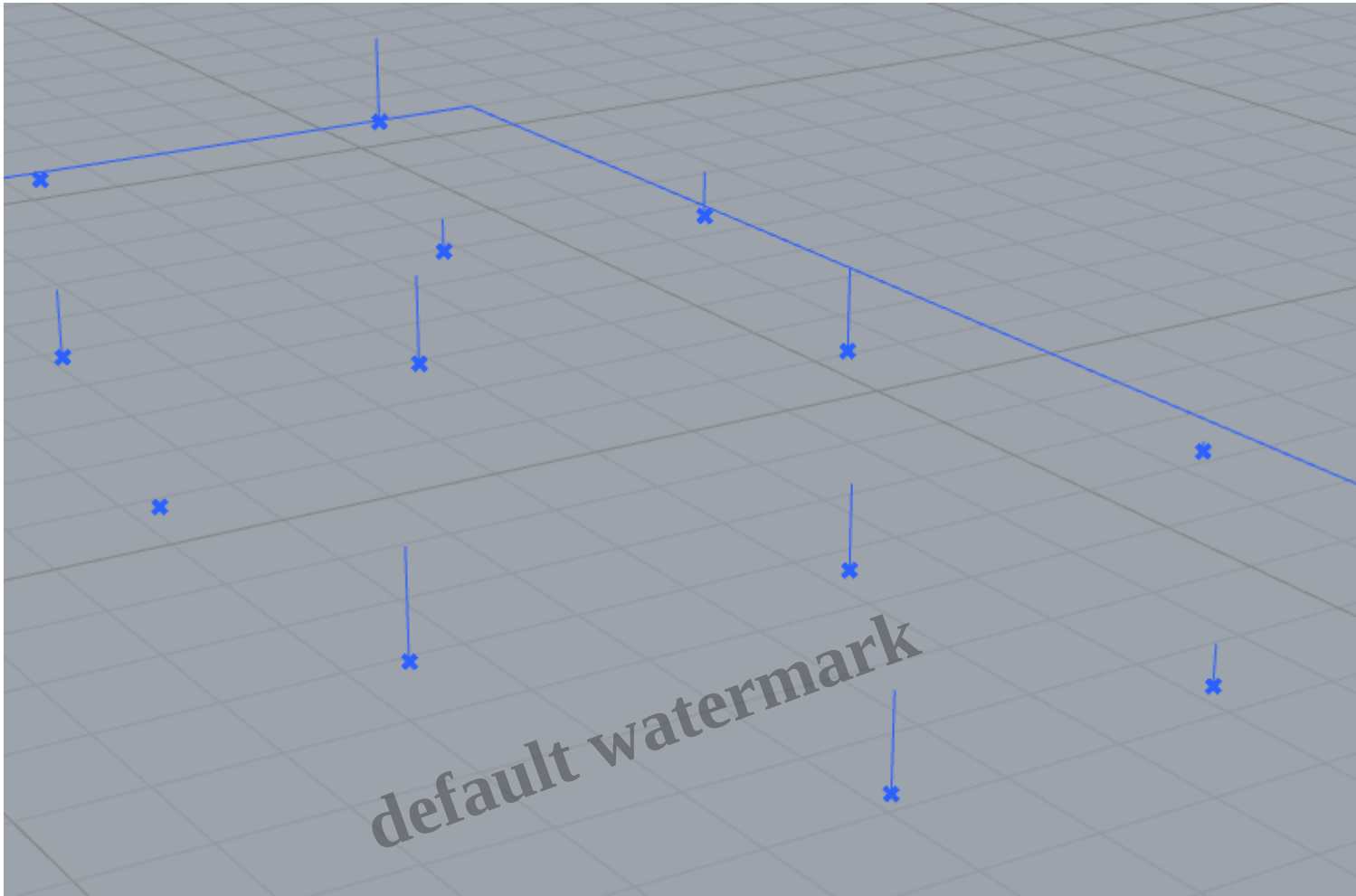


It has 3 inputs: *R* (*Range of Values*), *N* (*Number of Values*) and *S* (*Seed*, i.e. the random pattern index).

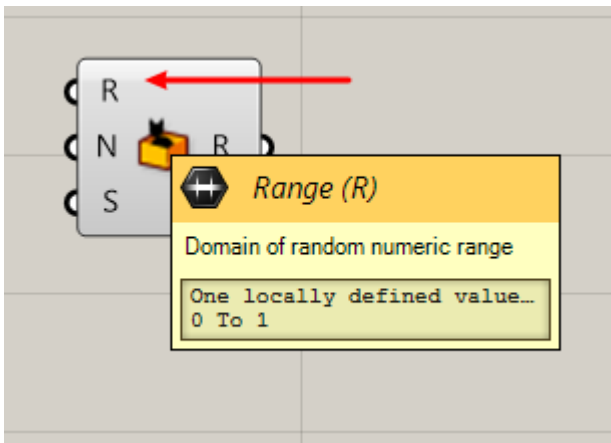
First of all let's connect this *Random* component to our *Line SDL*, via *R* output (*Random*) to *L* input (*Line SDL*). When you do this the *Number Slider* gets disconnected automatically:



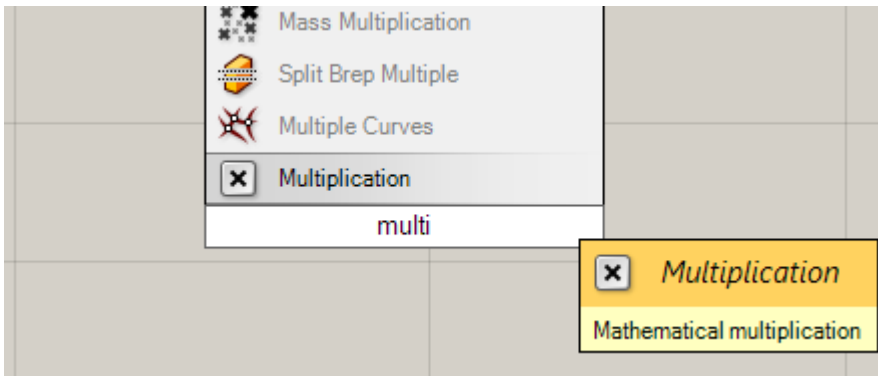
Up to now there is no randomness to be observed, because the *Number* input of our *Random* component is set to *1* (=no Variation). To set higher values you might come up with the idea to have as many variations as you have lines. The solution is to connect the *Populate Count Number Slider* to the *N* input of the *Random* component. Now the line lengths differ:



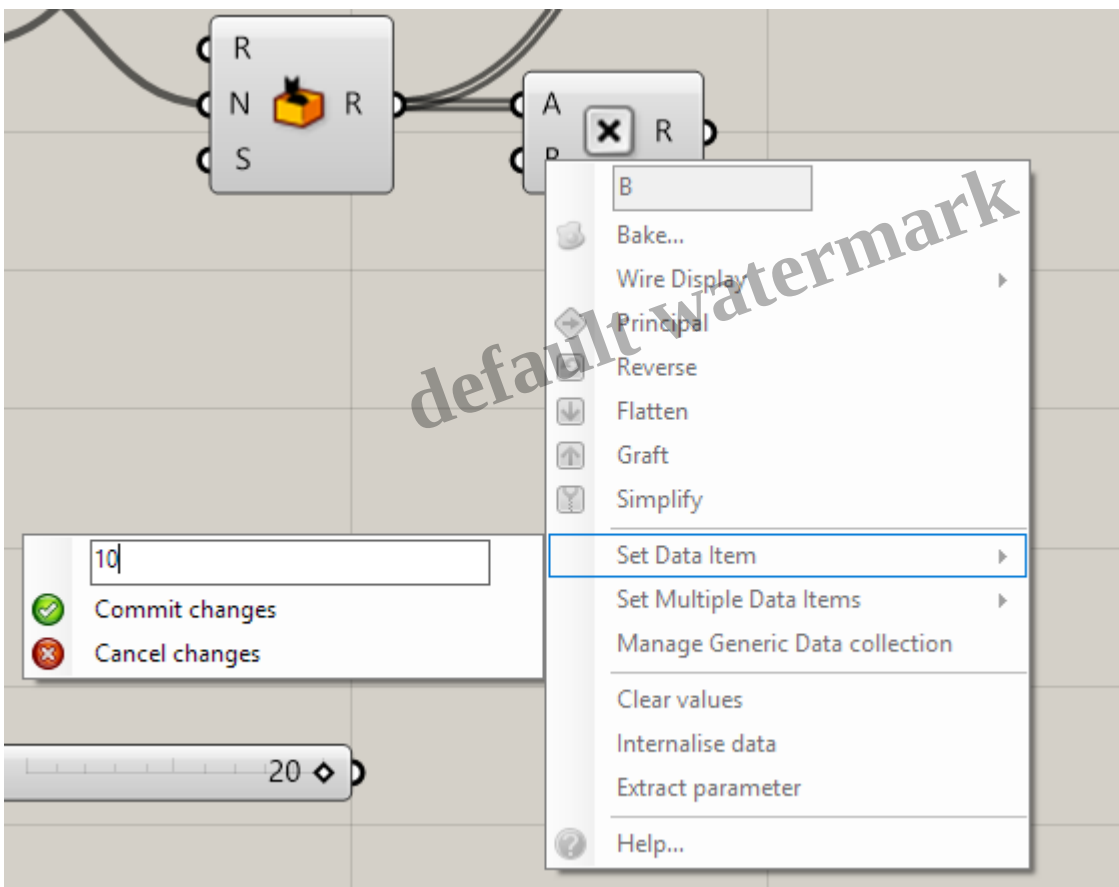
Now for the overall length of the lines. The *Random* component's *Range* default is 0 to 1:



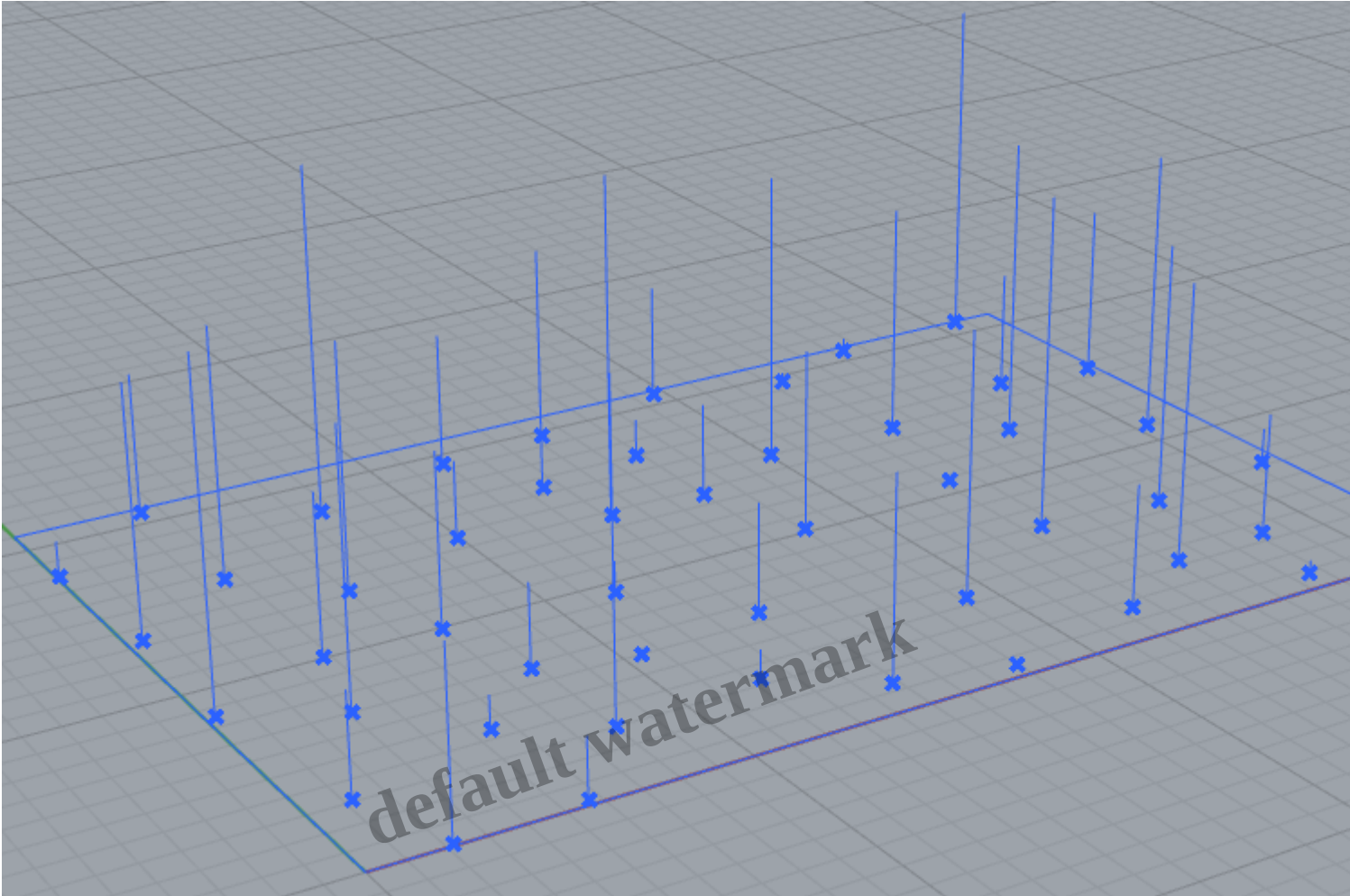
As we see, that produces very short lines. Now instead of changing the *Range* within the *Random* component I suggest using the random values as a factor for a multiplication, i.e. with 10. In order to do that, produce a *Multiplication* operator:



Plug the *Random* output into the *A* input of the *Multiplication* operator. Set *B* input to *10*, via right-click on *B* – *Set Data Item*:



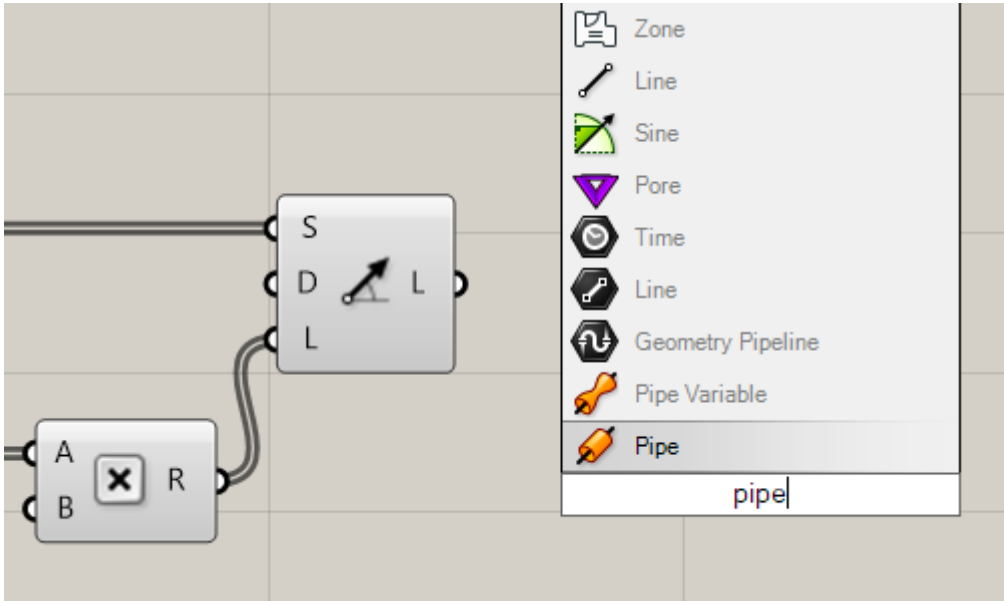
Now to utilize the multiplication result plug the *R* output into the *Line SDL*'s *Length* input:



As you see the lines have grown. Staying with the default *Random Range* from 0 to 1 was actually a good thing – by adding a *Multiplication* operator with a suitable *B* value we can scale up our lines as we like.

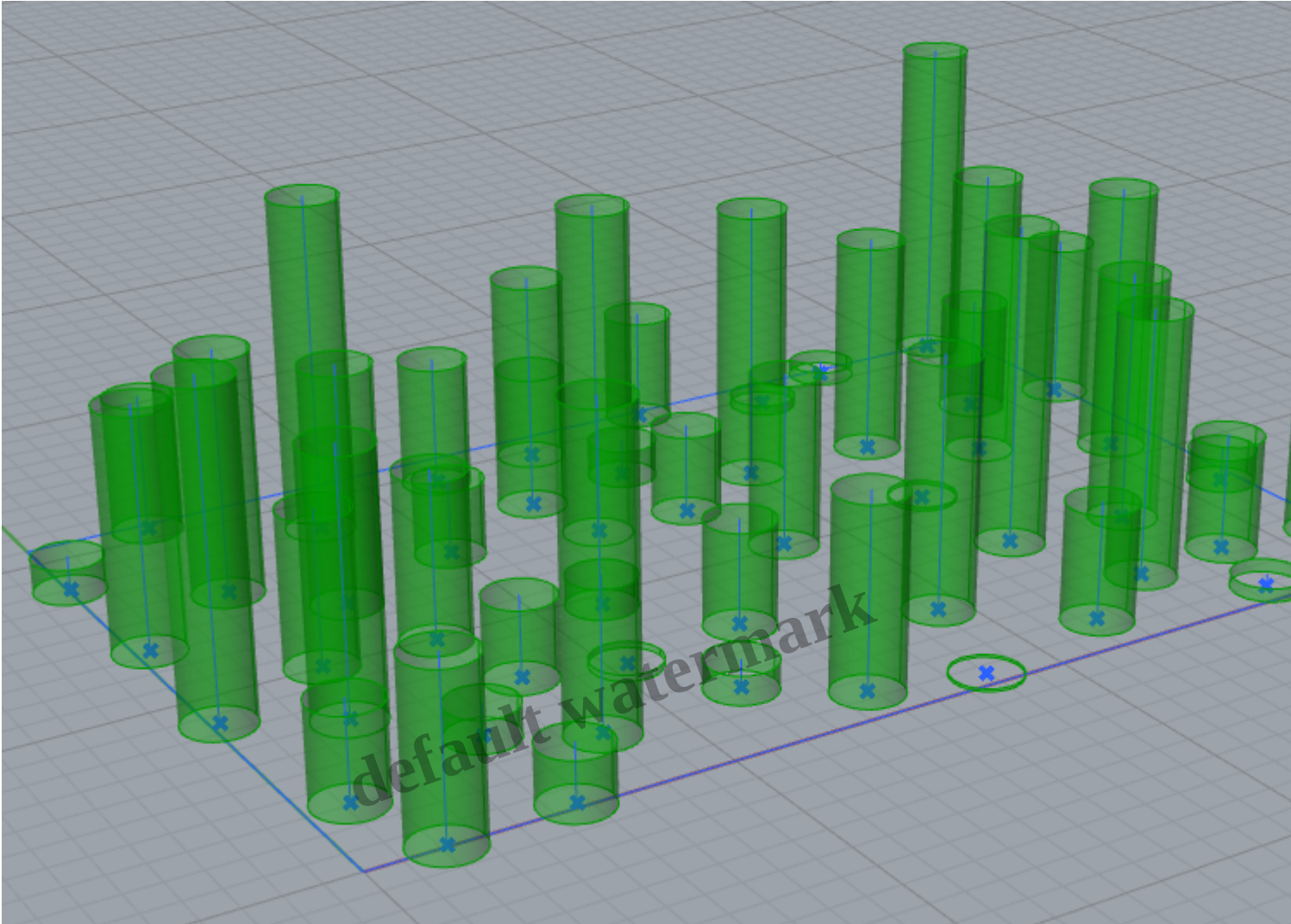
Finally, the Cylinders

To get some 3D around our lines we use the *Pipe* component:



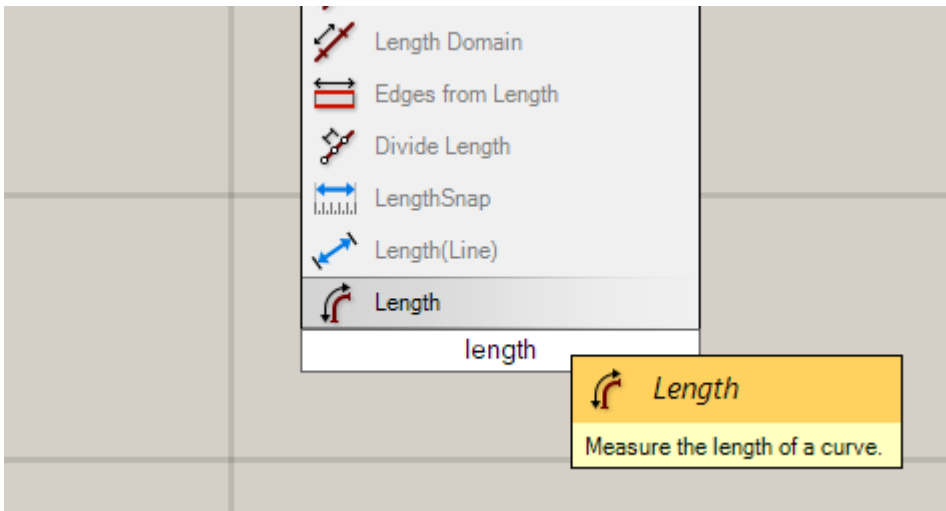
Pipe produces a cylindrical extrusion along a curve. So first of all you'll want to connect the *Pipe's C (Curve)* input to our *Line SDL* output. In Rhino's preview our lines have changed into a bunch of cylinders:

default watermark

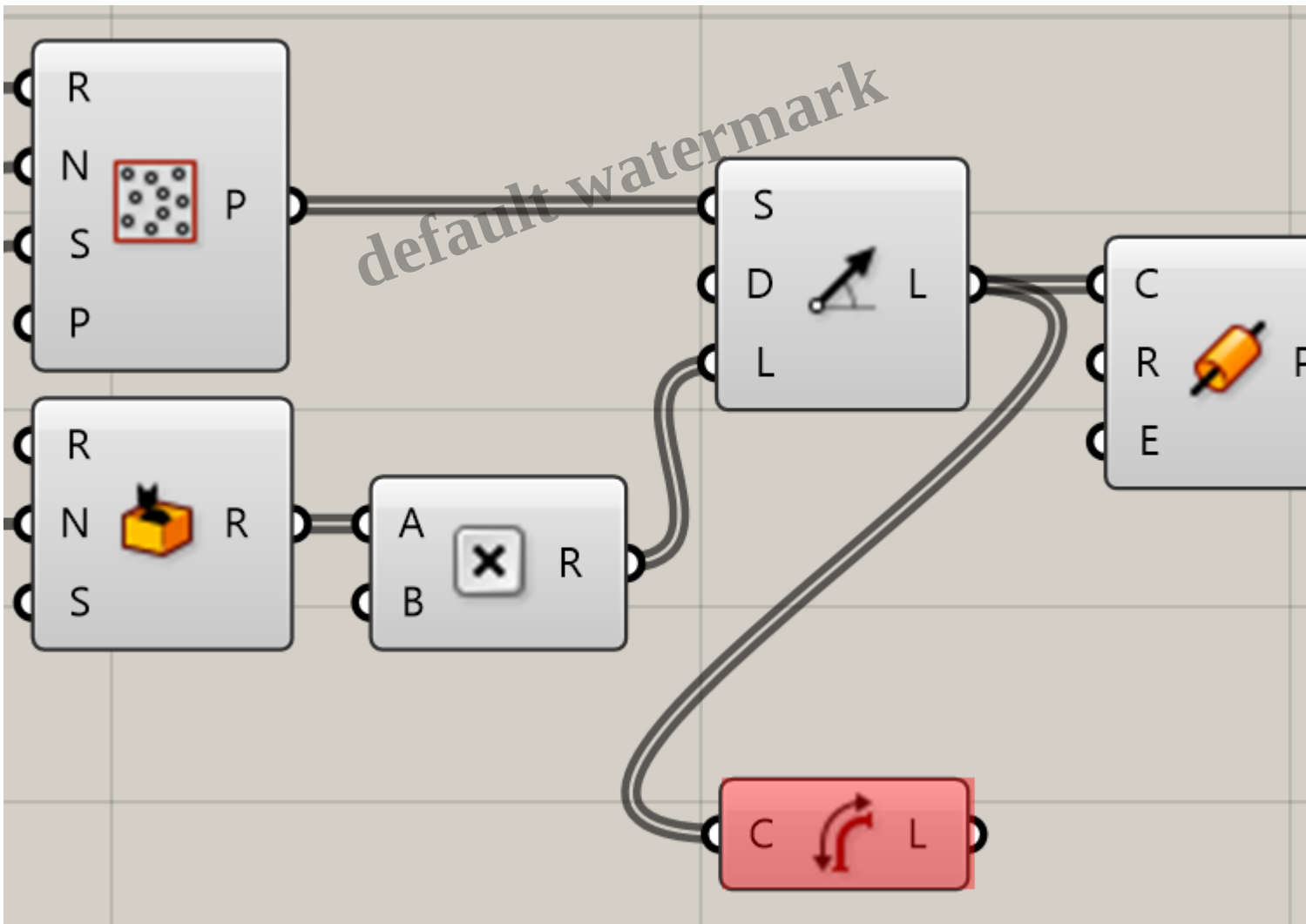


The concept of randomness still demands some work: I want to change the cylinders' radius according to their height. To be more precise, I would like to have each cylinder's radius to be $1/10$ of it's height. R is the according input of the *Pipe* component. Now what to plug into it?

First of all I need each cylinder's height. The height is represented by our *Line SDL* component. To retrieve it's value I need a *Length* component:

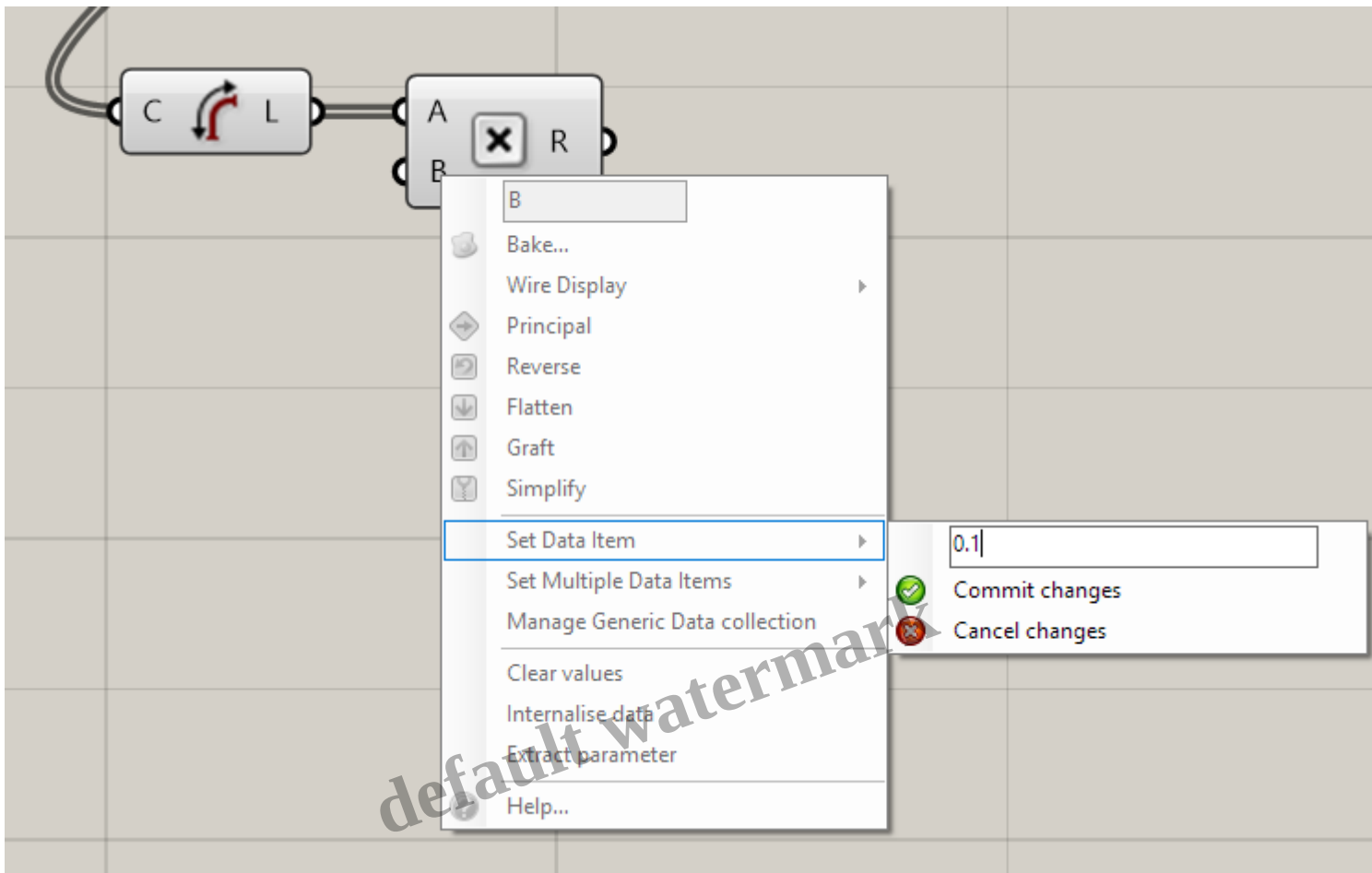


Now plug the *Length* input to the *Line SDL* output:



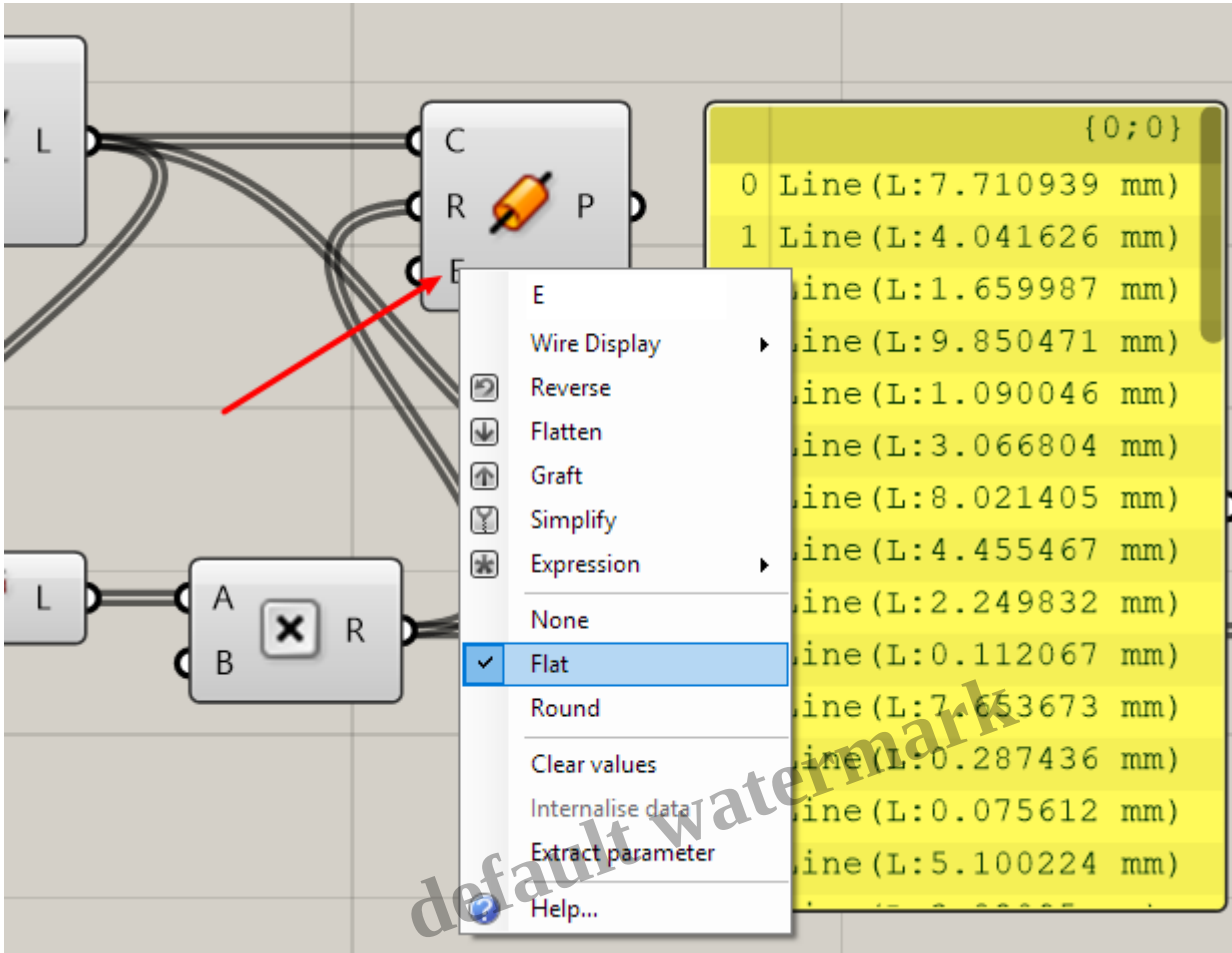
When each cylinder's radius is supposed to be 1/10 of it's height, then we have to divide *Length* by 10. Or better (at least in my opinion): We use a *Multiplication* again. So produce another *Multiplication*

operator and connect the *Length* output to it's *A* input. Set *B* to *0.1* (right-click – *Set Data Item*):



Now when you connect the *Multiplication* output to *Pipe*'s *R* input you see the cylinders' thickness change. For a test, play around with your sliders to see how everything changes accordingly.

To show you the relation between *Line SDL*'s output and *Pipe*'s *Radius* input (10:1) I added two *Panels* to compare the resulting values:



We could do some more random stuff like slanting the cylinders. For this we would have to deal with the *Line* SDL's *D* input. To keep things simple and leave something for you to find out yourself however we'll stop here.

Roundup

There's more to come. Just for now I want you to keep practising. Check also my other article on good [Grasshopper learning resources](#). Feel free to comment!

© 2018 / Horst Sonderrmann / All Rights reserved

Category

1. Rhino/Grasshopper

Tags

1. BIM Model
2. Parametric Modeling

Date Created

October 2018

Author

hsondermanncom

default watermark