FALCON handbook

Elizabeta Šamec and Krešimir Fresl

1 Introduction

An open source toolkit FALCON has been developed as a Grasshopper (GH) plugin that allows users to create, transform, and evaluate the geometry of spatial truss structures in real time. You can use the tool to find a form of spatial truss structures in tension or compression. This is done through a computational process that runs in the background and implements the algorithms developed in author's PhD research on enhancing the iterative application of force density method.

As seen in Fig. 2 the design tool is a collection of User Object clusters divided into *Geometry*, *Network*, and *Visualisation* groups. These clusters are available for the user to combine and add to GhPython components from the *Solver* group to create a form-finding workflow that matches the specifics of each example. To help you get started, examples of commonly used components and features are provided below. You can begin using the tool by following the example described hereafter. Once you are familiar with the tool, you will find several examples towards the end of the manual to try out by yourself. Examples as gh. files are provided inside FALCON.zip folder on FALCON page. There you can also find video tutorials and additional documentation. If you are new to GH, we recommend that you first browse this manual to understand the main principles.

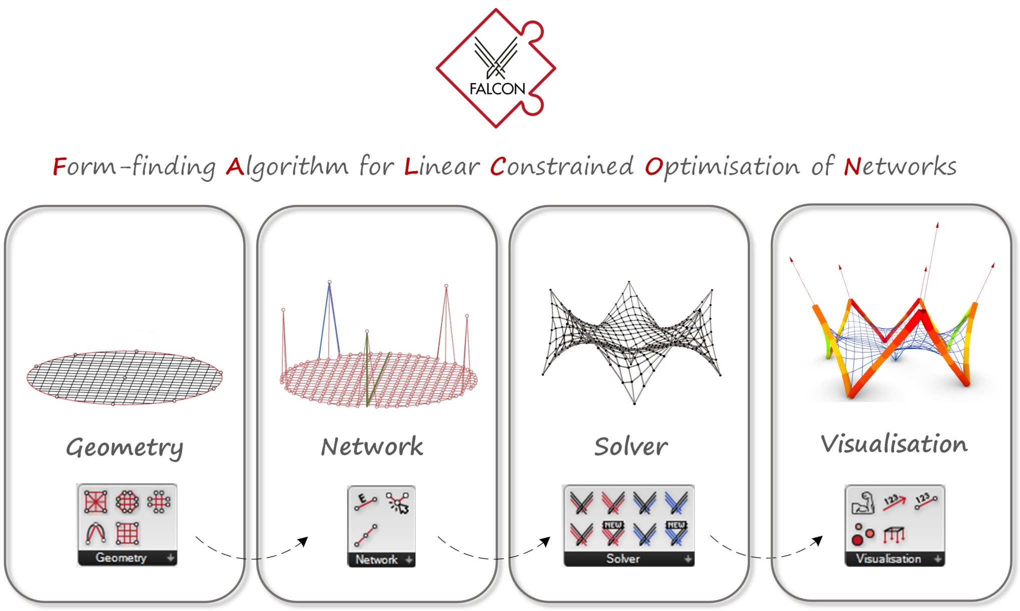
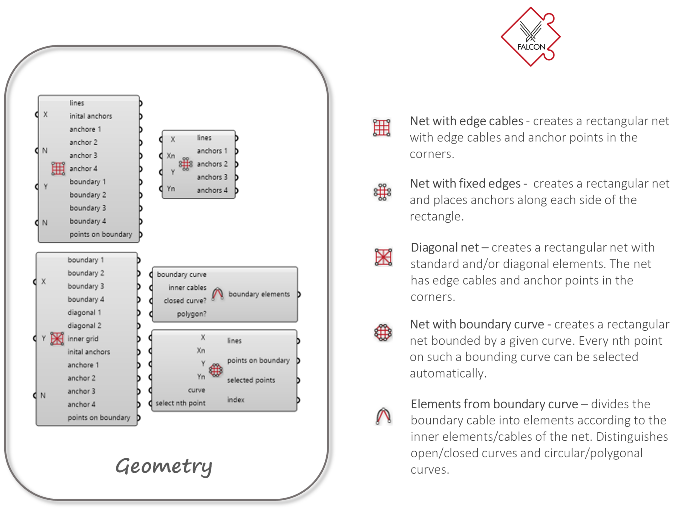


Figure 1: FALCON tool – group of components developed for constrained interactive form-finding.

2 Example to get started

2.1 How to create geometry

In order to assign geometry to the form-finding function, a pin-jointed model must be created. The model is represented by a series of lines in the ground plane (xy) and the supports are represented by points (see Fig. 4). These points are also initially defined in the ground plane and are referred to as the initial position of the anchors (initial anchors). To represent the topology, the geometry can either be modelled with a static Rhino-Geometry or directly and more flexibly scripted with the GH environment. The most common topologies are defined by components in the *Geometry* group, as can be seen in Fig. 2.



Net with edge cables - creates a rectangular net with edge cables and anchor points in the corners.

Net with fixed edges - creates a rectangular net and places anchors along each side of the rectangle.

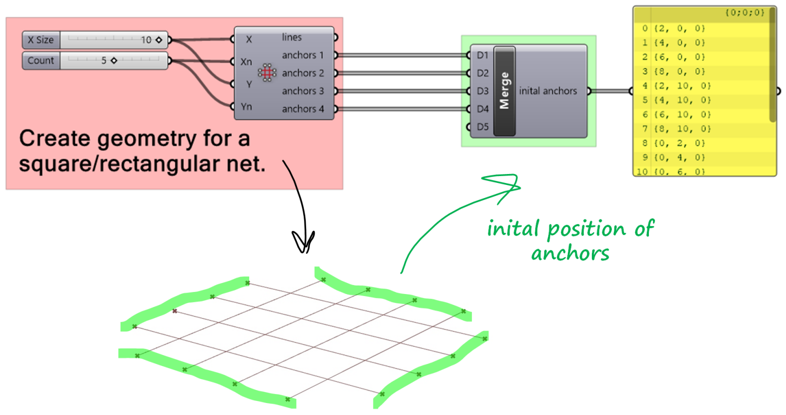
Diagonal net – creates a rectangular net with standard and/or diagonal elements. The net has edge cables and anchor points in the corners.

Net with boundary curve - creates a rectangular net bounded by a given curve. Every nth point on such a bounding curve can be selected automatically.

Elements from boundary curve – divides the boundary cable into elements according to the inner elements/cables of the net. Distinguishes open/closed curves and circular/polygonal curves.

Figure 2: *Geometry* group of components.

For starters, choose the component *Net with fixed edges*, which creates a topology for a square or rectangular model with fixed edge anchors (no edge elements). As seen in Fig. 3, you can change the overall size of the model using the slider that goes into the inputs X and Y, and the density of the subdivision by changing the values Xn and Yn. As output, the component returns the lines of the model and the list of anchors on each side of the net. As mentioned before, the initial position of the anchors is needed for further work, so the lists *anchors 1-4* can be combined with the component *Merge* into one.



```
                {0;0;0}
 0 {2, 0, 0}
 1 {4, 0, 0}
 2 {6, 0, 0}
 3 {8, 0, 0}
 4 {2, 10, 0}
 5 {4, 10, 0}
 6 {6, 10, 0}
 7 {8, 10, 0}
 8 {0, 2, 0}
 9 {0, 4, 0}
10 {0, 6, 0}
```

Figure 3: Defining initial geometry using *Net with fixed edges* component from *Geometry* group.

It is important to emphasise that the drawn or generated geometry can be defined as a set of lines without the need to draw each element between the intersection points separately (splitting the lines). This will be done automatically when we convert our geometry into an interactive network in the next step.

If one of the topologies is to be used, but the boundary must be defined by a curve or polygon, component *Elements from boundary curve* from the *Network* group can be used to approximate this boundary by a series of lines corresponding to the intersections with the network geometry (latter shown in examples on Fig. 23 and 26). If such an example is based on a regular network, the component *Net with boundary curve* can also be used, which additionally allows interactive selection of points on the resulting boundary cable (Fig. 26). Examples of structures whose geometry was created using the described components can be found in Fig. 4. But, for now, let us return to our "simple" example.

2.2 How to turn geometry to interactive network

To perform the form-finding, at least one of the supports must be moved from the original xy plane. The supports are usually moved from their initial position using a combination of components such

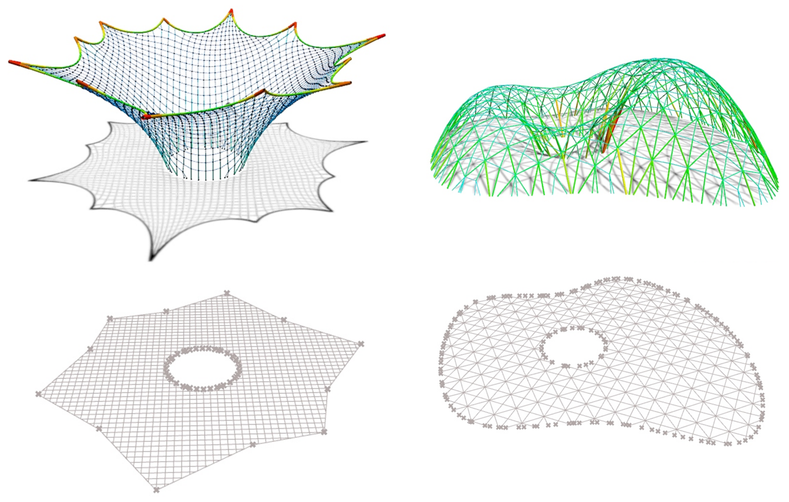
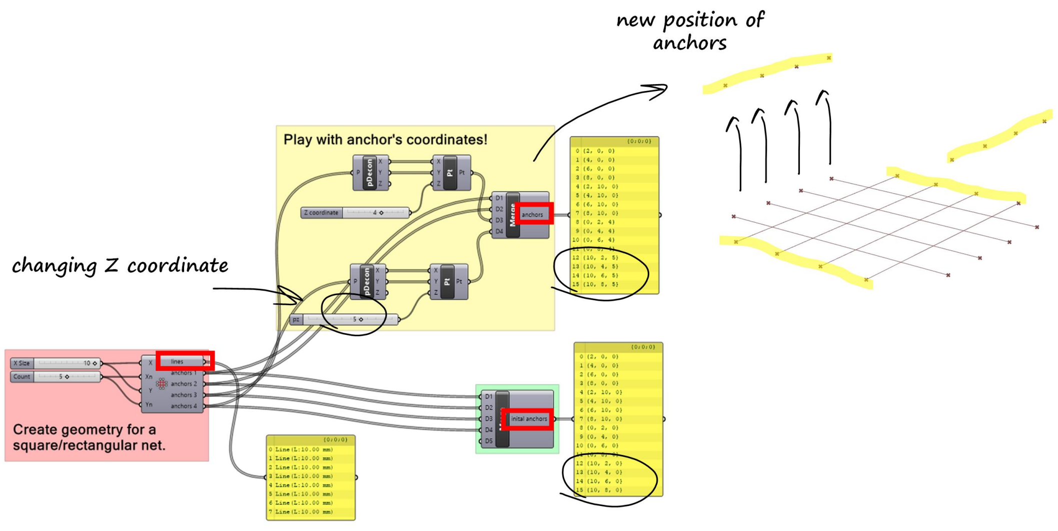


Figure 4: Examples of initial geometries that can be easily created with FALCON *Geometry* components and corresponding final structures.

as *Point Deconstruct* - *Point Construct* (see Figure 5) or by a component *Move* manipulated by a vector. Advanced users can define other types of anchor manipulations or project the anchor points onto the geometry of the supporting substructure. The order of anchors in the *initial anchors* and *anchors* lists must be the same, and new anchors can always be defined by adding the coordinates in the two lists.



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

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

```
{0;0;0}
0 Line(L:10.00 mm)
1 Line(L:10.00 mm)
2 Line(L:10.00 mm)
3 Line(L:10.00 mm)
4 Line(L:10.00 mm)
5 Line(L:10.00 mm)
6 Line(L:10.00 mm)
7 Line(L:10.00 mm)
```

Figure 5: Defining new position of anchors.

The interactivity of such a network must be ensured so that changes in the positions of the anchor points are automatically detected and the geometry is restored according to the topology. Therefore, the next step of the workflow is to create the network consisting of nodes and elements defined by intersections of lines. This is done with the help of the component *Interactive net*, which generates the network according to the changes in real time, as shown in Fig. 6. In this way a connection between the user and the tool is established. This component is located under the *Network* group, as shown in Fig. 7. The inputs for this component are the outputs of the geometry generation: *lines* to specify the topology, *initial anchors* as the list of initial coordinates of all anchors, and *anchors* as the list of coordinates of their new position.

The cluster *Interactive net* contains two components from the Heteroptera plugin, *Network form lines* and *Rebuild network*. Therefore, Heteroptera must be installed along with FALCON (see installation details at FALCON page). The network is generated using the *Network from lines* component, which returns network elements defined by points on the intersection of lines, i.e. nodes, along with the connectivity and topology of the nodes. Then, the coordinates of the initial anchors are replaced with new ones and the network is updated using the component *Rebuild*

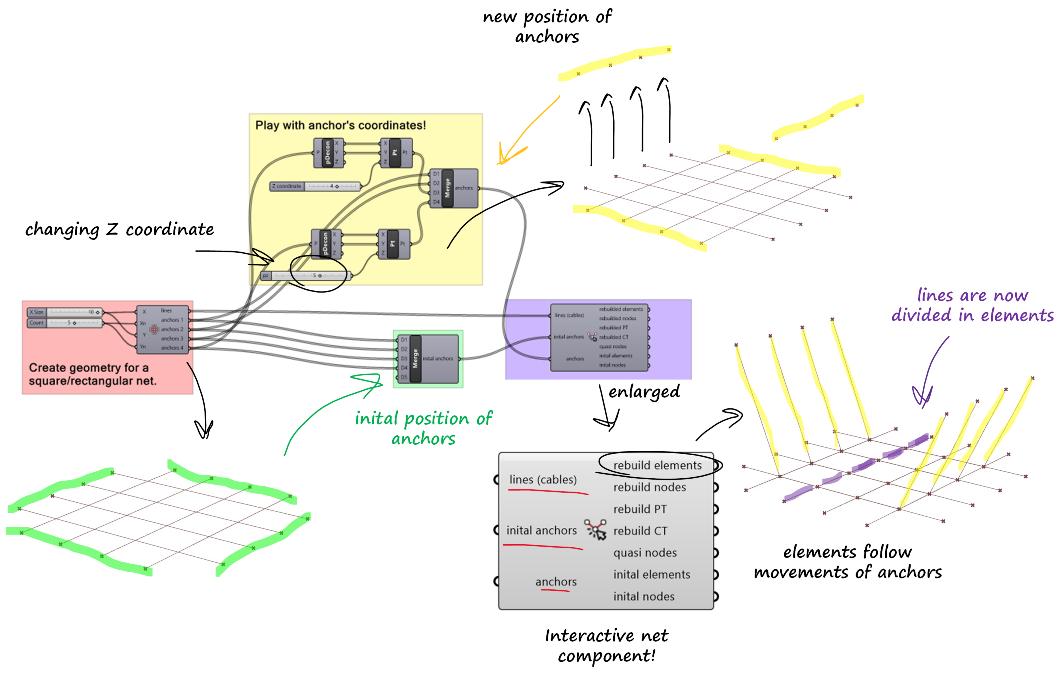
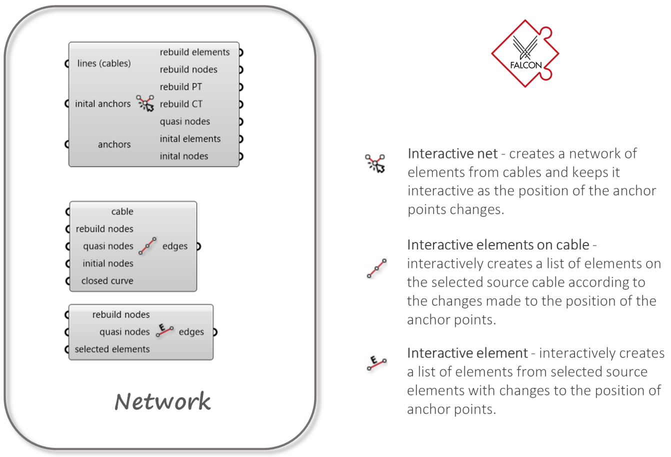

new position of anchors
Play with anchor's coordinates!
changing Z coordinate
Create geometry for a square/rectangular net.
lines are now divided in elements
inital position of anchors
enlarged
lines (cables)
inital anchors
anchors
rebuild elements
rebuild nodes
rebuild PT
rebuild CT
quasi nodes
inital elements
inital nodes
elements follow movements of anchors
Interactive net component!

Figure 6: Creating interactive network from geometry.

network. *Interactive net* returns to the user the newly created elements, nodes, connectivity and topology of the nodes, but also the originally generated network (elements and nodes) in the xy plane. Another output is so-called quasi-points, which is relevant to the part of the workflow where elements are selected for constraint assignment. These points are generated so that the initial nodes are not confused with anchors when they are moved and their position may overlap.



Interactive net - creates a network of elements from cables and keeps it interactive as the position of the anchor points changes.

Interactive elements on cable - interactively creates a list of elements on the selected source cable according to the changes made to the position of the anchor points.

Interactive element - interactively creates a list of elements from selected source elements with changes to the position of anchor points.

Figure 7: *Network* group of components.

The functionality and use of the other two components from this group will be explained a little later, but for now to the form-finding!

2.3 How to find a form

For basic form-finding, we use the component FALCON (one step of FDM), the first of the four components of the subgroup Solver TENSION (see Fig. 8). The other three components are for constrained form-finding, which we will explain later.

For the FALCON component to work and provide a solution in equilibrium, the following inputs are minimally required:

- *rebuild elements* - the list of lines forming the network, output of the *Interactive net* component,

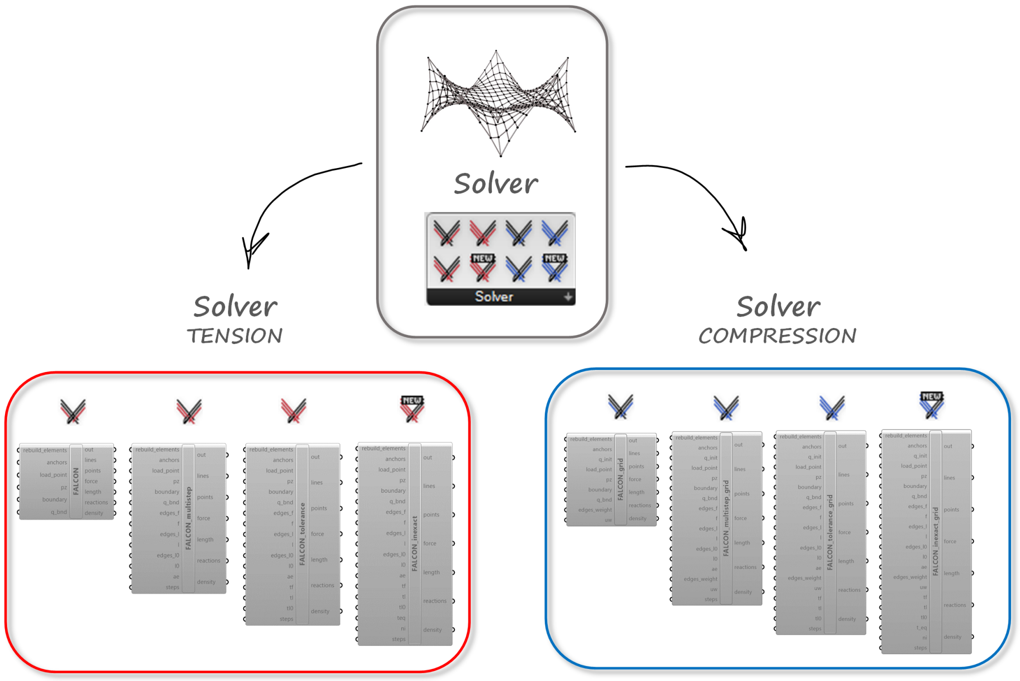


Figure 8: *Solver* group of components divides to subgroups TENSION and COMPRESSION.

- *anchors* - the list of support points at the desired positions.

So let us draw ″spaghetti″ from those outputs and enter the data for the FALCON component. To activate the FALCON component, right-click on the name and select Enable. If all is well connected, the component will become dark grey, but you will not see the nice result as in Fig. 9 before using the visualisation components from the next chapter. However, you can see the solution when you preview the result by right-clicking FALCON again and choosing Preview.

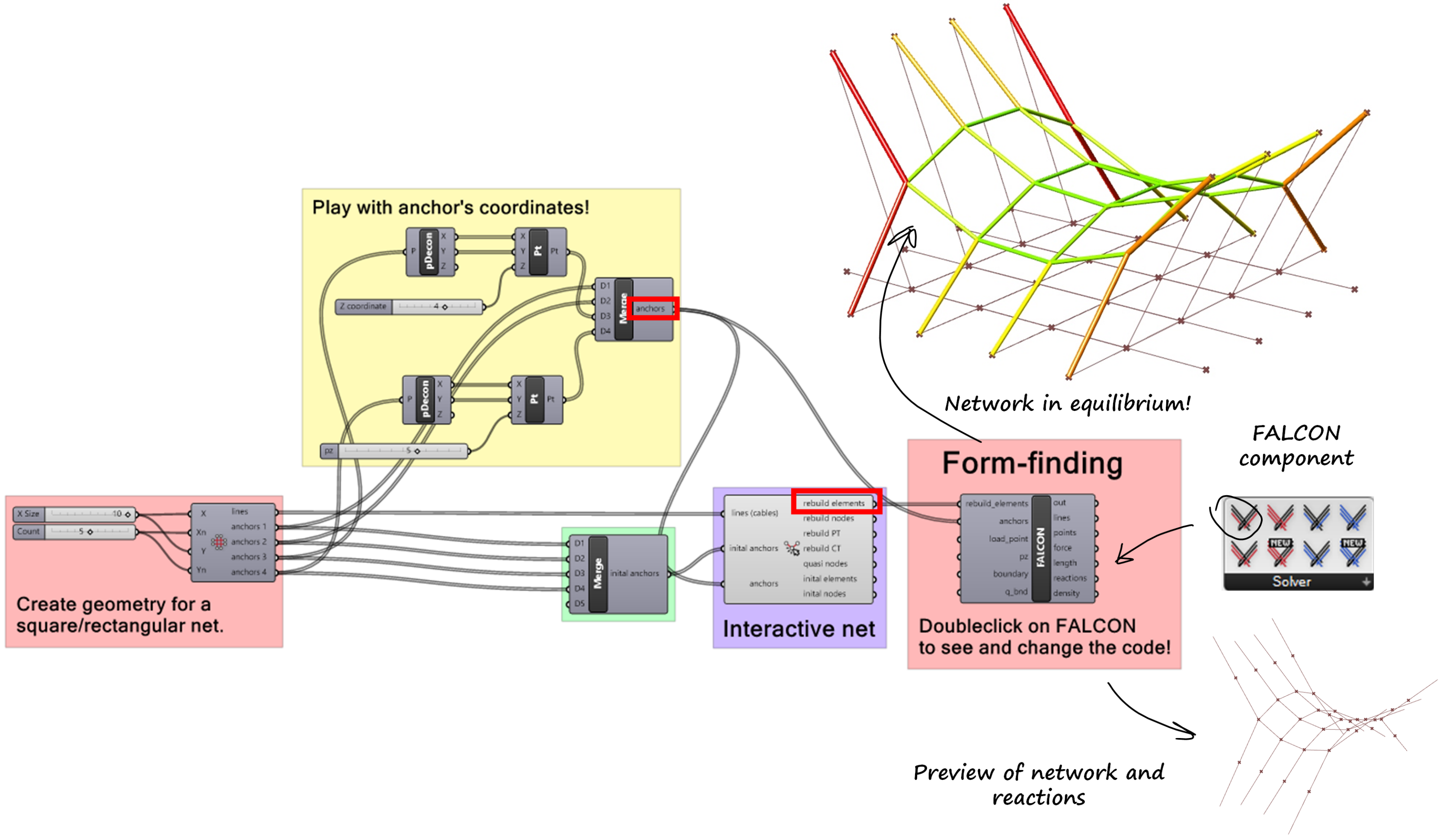
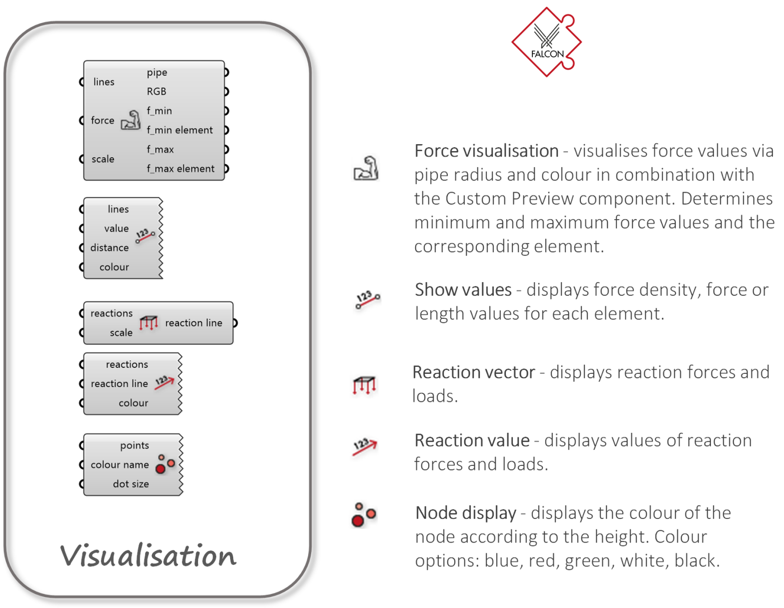


Figure 9: Form-finding workflow for net with fixed edges.

2.4 How to visualise results

After the calculation is performed by calling the form-finding function FDM, the net is drawn and the values of the forces, lengths and force densities in the elements are given as output. The *Visualisation* group of components enables to visualise those outputs. The *Force visualisation* component giving the coloured 3D representation of the force values (and the possibility to highlight the elements with the minimum and maximum force values by a different colour) helps to visually get information about the behaviour of the structure. Reactions can also be previewed by vector using *Reaction vector* or together with their values using *Reaction value* component. Usage of all three components is shown in Fig. 11. The values of forces, lengths or force densities can be displayed using *Show values* component as shown in Fig. 12. In addition, the colour of the nodes can be graded by height, which can be very useful when you check the arrangement of the elements in the plan view after finding the shape (Fig. 13).



Force visualisation - visualises force values via pipe radius and colour in combination with the Custom Preview component. Determines minimum and maximum force values and the corresponding element.

Show values - displays force density, force or length values for each element.

Reaction vector - displays reaction forces and loads.

Reaction value - displays values of reaction forces and loads.

Node display - displays the colour of the node according to the height. Colour options: blue, red, green, white, black.

Figure 10: *Visualisation* group of components.

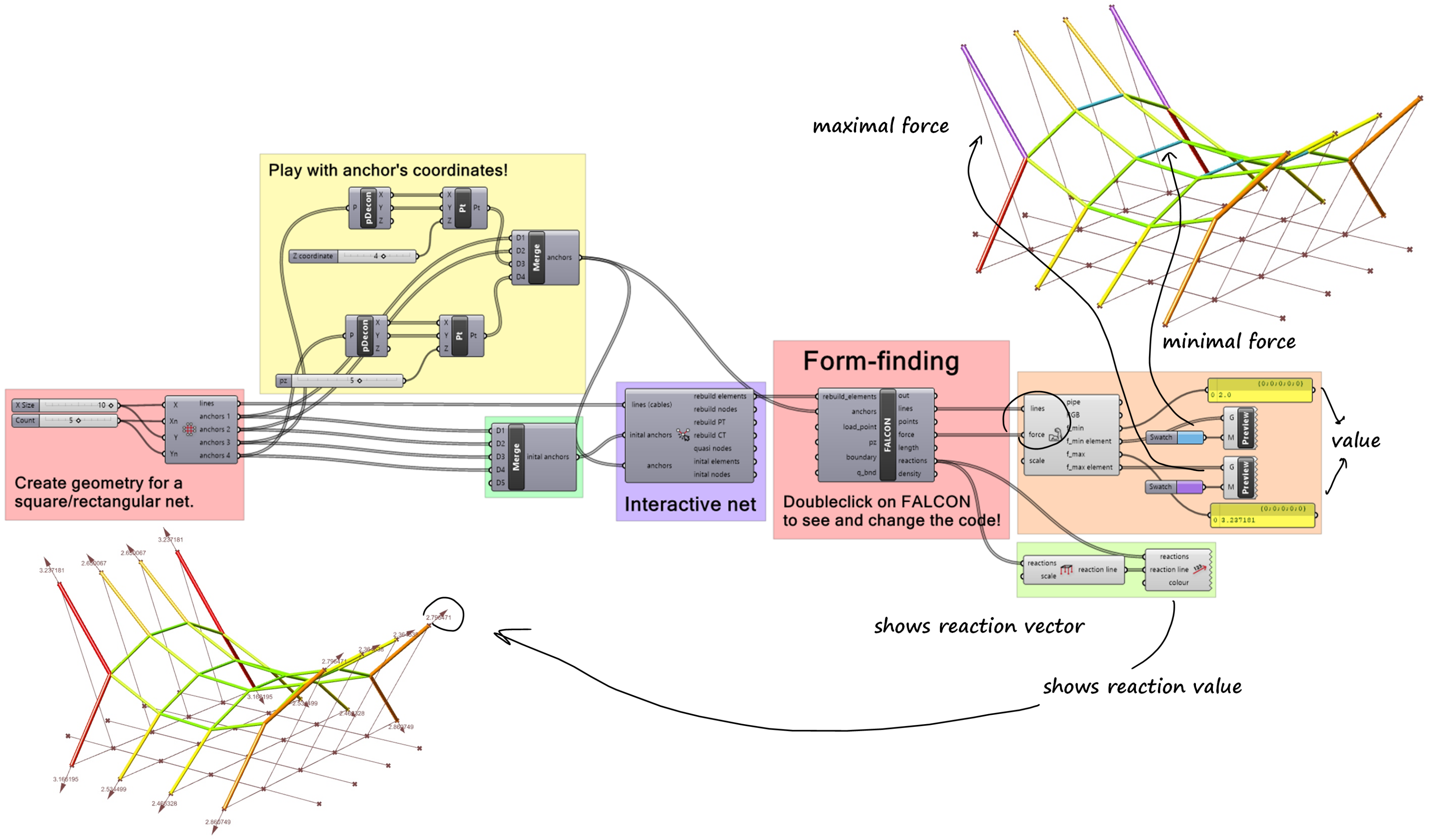


Figure 11: Visualisation of forces and reactions.

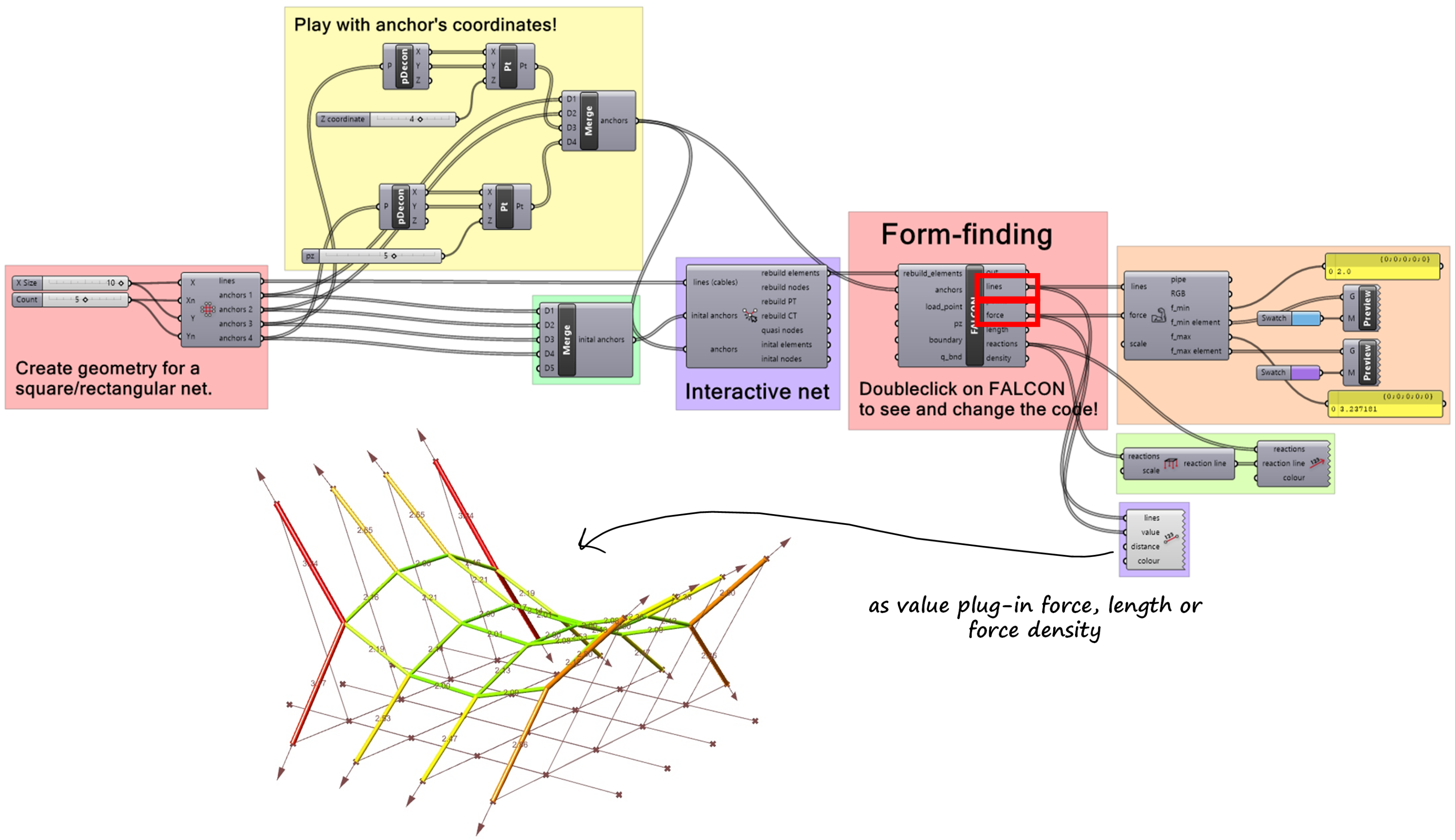

Play with anchor's coordinates!
Create geometry for a square/rectangular net.
Interactive net
Form-finding
Doubleclick on FALCON to see and change the code!
as value plug-in force, length or force density

Figure 12: Displaying value of force in elements.

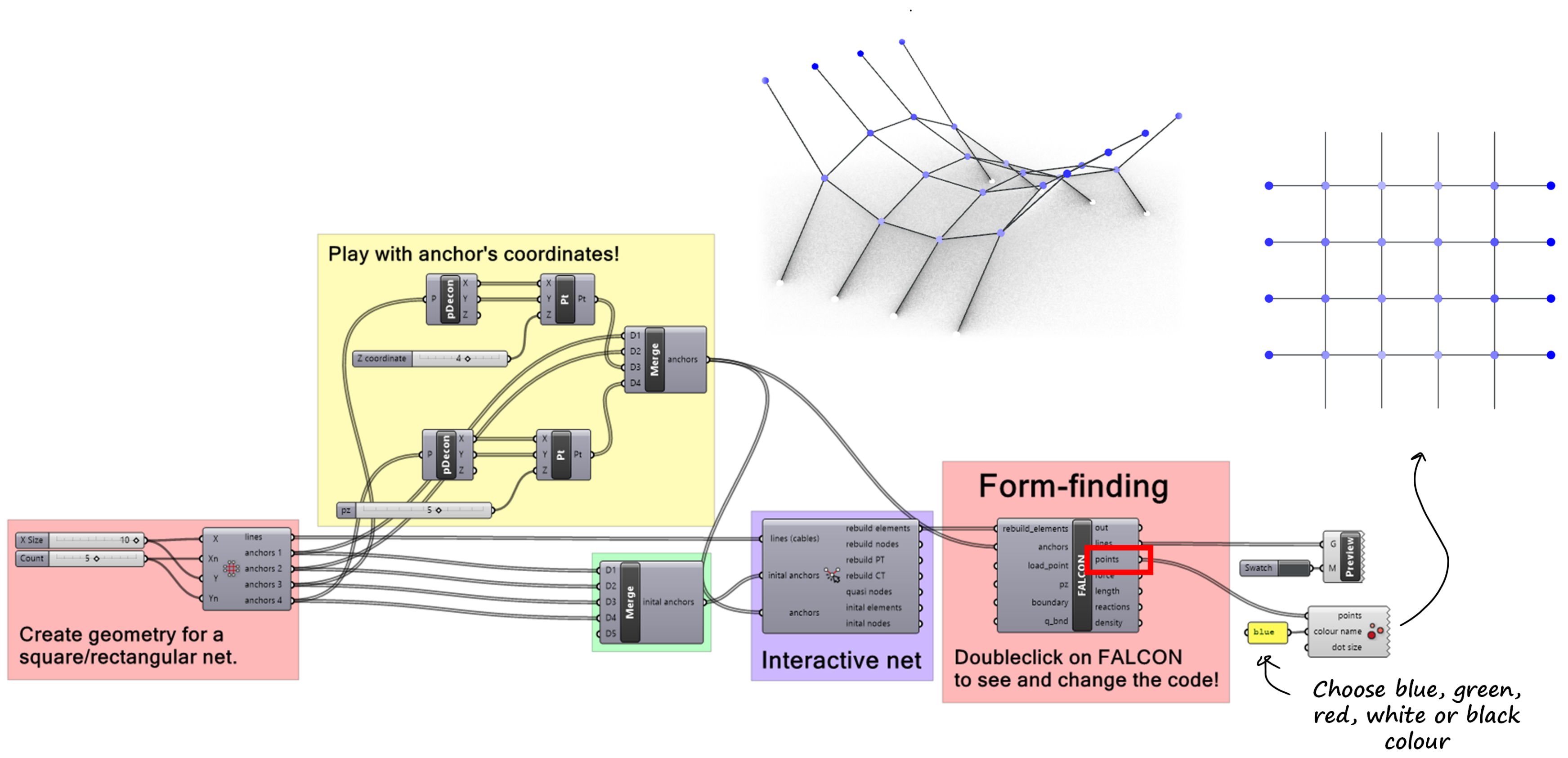


Figure 13: Displaying node colour according to height.

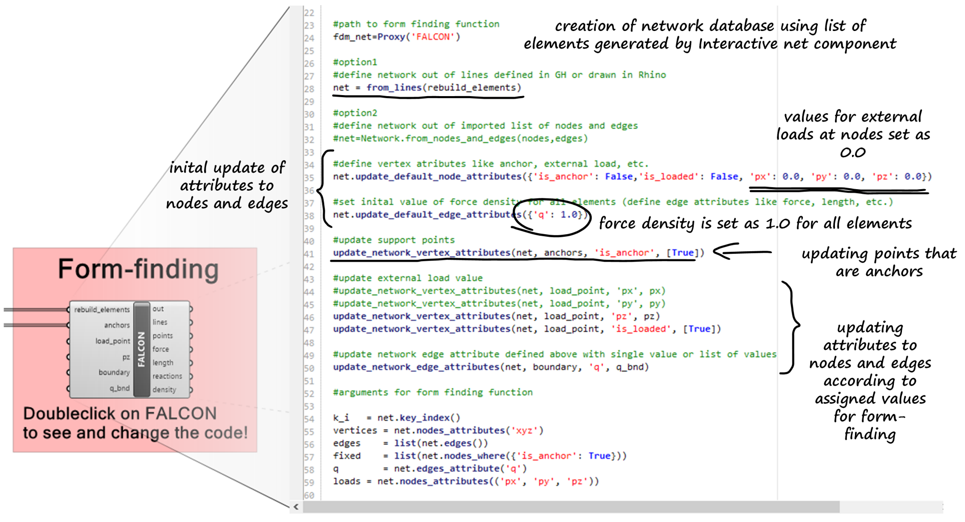
2.5 How to change force density

FALCON is developed due to the lack of interactive form-finding tools based on the force density method, especially ones enabling the assignment of additional constraints. The force density method is based on the ratio of force and length in the element - force density. Due to its easy-to-understand fundamental principles, it is an extremely versatile method that, even though it is developed almost half a century ago, remains unique due to its remarkable ability to start from poorly defined geometry and still provide satisfactory result since only the topology matters in equilibrium shape. That is why we believe that this method is still worth revising in the new era of form-finding.

Since FALCON is not the "black box" component, you can open the component by clicking on its name. You will find the code whose parts are explained in Figs. 14 and 15. As you can see in Fig. 14 in line 38, all elements are automatically assigned a unit force density. This can be changed. If we want to assign higher force density values to a particular cable from our example, we can use the *Interactive elements on cable* component - one of the two from the *Network* group we neglected earlier.

Interactive elements on cable and *Interactive element* components are intended for the selection of elements to be constrained. Depending on the used component list of elements is interactively created based on the selected source cable or from selected source elements according to the changes made to the position of the anchor points. Therefore, their inputs, except from the cable/elements we want to select, are outputs of the *Interactive net* component and that is why they are all-together put under *Network* group. The difference between the two components is the possibility to select all elements on a given cable drawn or generated at the beginning of the workflow, or to select the individual elements created by the component *Interactive net* (*initial elements*) or selected from Rhino.

To select a cable to put in the *Interactive elements on cable* component, just take a *List item* component, insert *lines* output from the *Net with fixed edges* component and select a line with a slider. If the remaining inputs are connected from the *Interactive net* component (leave *closed curve* untouched for now), you can get the elements on cable that can interactively change if the anchors move. These elements need to be hand over to the *boundary* input in FALCON (the name is explained a bit latter, feel free to change it, just do not forget to do the same in the code (line 50)). A slider should be connected to *q_bnd* so that in line 50 (Fig. 14) these elements get assigned the new values of the force densities. If one of the concave cables is selected, as in picture, by increasing the value of *q_bnd* the structure will be elevated from increased tension in that cable - higher force density higher force.

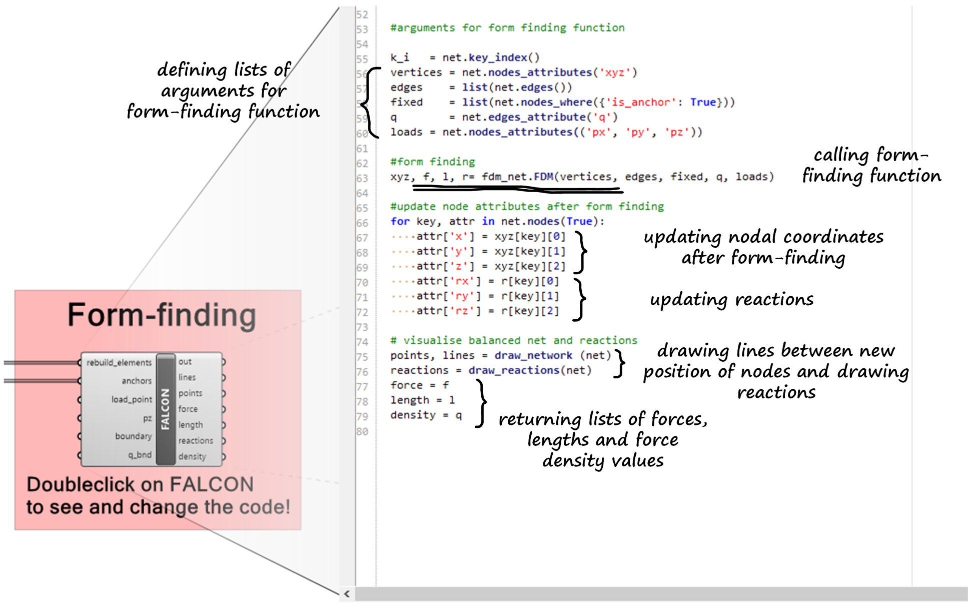


```
22
23  #path to form finding function
24  fdm_net=Proxy('FALCON')
25
26  #option1
27  #define network out of lines defined in GH or drawn in Rhino
28  net = from_lines(rebuild_elements)
29
30  #option2
31  #define network out of imported list of nodes and edges
32  #net=Network.from_nodes_and_edges(nodes,edges)
33
34  #define vertex atributes like anchor, external load, etc.
35  net.update_default_node_attributes({'is_anchor': False,'is_loaded': False, 'px': 0.0, 'py': 0.0, 'pz': 0.0})
36
37  #set inital value of force density for all elements (define edge attributes like force, length, etc.)
38  net.update_default_edge_attributes({'q': 1.0})
39
40  #update support points
41  update_network_vertex_attributes(net, anchors, 'is_anchor', [True])
42
43  #update external load value
44  #update_network_vertex_attributes(net, load_point, 'px', px)
45  #update_network_vertex_attributes(net, load_point, 'py', py)
46  update_network_vertex_attributes(net, load_point, 'pz', pz)
47  update_network_vertex_attributes(net, load_point, 'is_loaded', [True])
48
49  #update network edge attribute defined above with single value or list of values
50  update_network_edge_attributes(net, boundary, 'q', q_bnd)
51
52  #arguments for form finding function
53
54  k_i    = net.key_index()
55  vertices = net.nodes_attributes('xyz')
56  edges    = list(net.edges())
57  fixed    = list(net.nodes_where({'is_anchor': True}))
58  q        = net.edges_attribute('q')
59  loads = net.nodes_attributes(('px', 'py', 'pz'))
60
```

creation of network database using list of elements generated by Interactive net component
values for external loads at nodes set as 0.0
inital update of attributes to nodes and edges
force density is set as 1.0 for all elements
updating points that are anchors
updating attributes to nodes and edges according to assigned values for form-finding
Form-finding
rebuild_elements
anchors
load_point
pz
boundary
q_bnd
FALCON
out
lines
points
force
length
reactions
density
Doubleclick on FALCON to see and change the code!

Figure 14: Code inside FALCON component - part 1.

If the network has edge cables instead of fixed support points, boundary elements are often defined with a higher value of force density. That is why a specific input named *boundary* was created for those elements, and the corresponding input on force density value *q_bnd*. Take a look at the additional example in Fig. 24. The workflow is very similar to one in our example, except that the component *Net with edge cables* is used to create the geometry. This component has as



defining lists of arguments for form-finding function
calling form-finding function
updating nodal coordinates after form-finding
updating reactions
drawing lines between new position of nodes and drawing reactions
returning lists of forces, lengths and force density values
Form-finding
rebuild_elements
anchors
load_point
pz
boundary
q_bnd
FALCON
out
lines
points
force
length
reactions
density
Doubleclick on FALCON to see and change the code!

```
#arguments for form finding function

k_i    = net.key_index()
vertices = net.nodes_attributes('xyz')
edges    = list(net.edges())
fixed    = list(net.nodes_where({'is_anchor': True}))
q        = net.edges_attribute('q')
loads = net.nodes_attributes(('px', 'py', 'pz'))

#form finding
xyz, f, l, r= fdm_net.FDM(vertices, edges, fixed, q, loads)

#update node attributes after form finding
for key, attr in net.nodes(True):
    attr['x'] = xyz[key][0]
    attr['y'] = xyz[key][1]
    attr['z'] = xyz[key][2]
    attr['rx'] = r[key][0]
    attr['ry'] = r[key][1]
    attr['rz'] = r[key][2]

# visualise balanced net and reactions
points, lines = draw_network (net)
reactions = draw_reactions(net)
force = f
length = l
density = q
```

Figure 15: Code inside FALCON component - part 2.

output all four boundary cables. When we want to tens them, we can select the elements on these cables by plugging each of the lines (*boundary 1-4*) into the *cable* input of the *Interactive elements on cable* component. If we want to do it all at once, we can set them all as input (use Shift to plug in more "spaghetti"). One more example with tightening of boundary cables is given in Fig. 25.

In addition to the mandatory inputs, the component also has inputs for the points to be loaded named *load_point* and the value of the load in the vertical direction *pz*. Other load directions can be added as inputs by zooming in on the input part of the component and adding input using plus sign that appears (on right click set type as list). There is already prepared part of the code to take in to account those inputs in lines 44 and 45 in Fig. 14, it just needs to be enabled. As seen in Fig. 17, from the list *rebuild nodes* (output of *Interactive net* component) using *List item* component one can select any node to put load one. To make assigning easier, index can be previewed using combination of components *List length* and *Series*. Plugging length of the *rebuild nodes* list in to C input of *Series* component will create indexes that can be seen as text if they are placed on the location of the nodes coordinates using *Text Tag* component. The dot can be created to visually follow the node selection (all shown on Fig. 17).

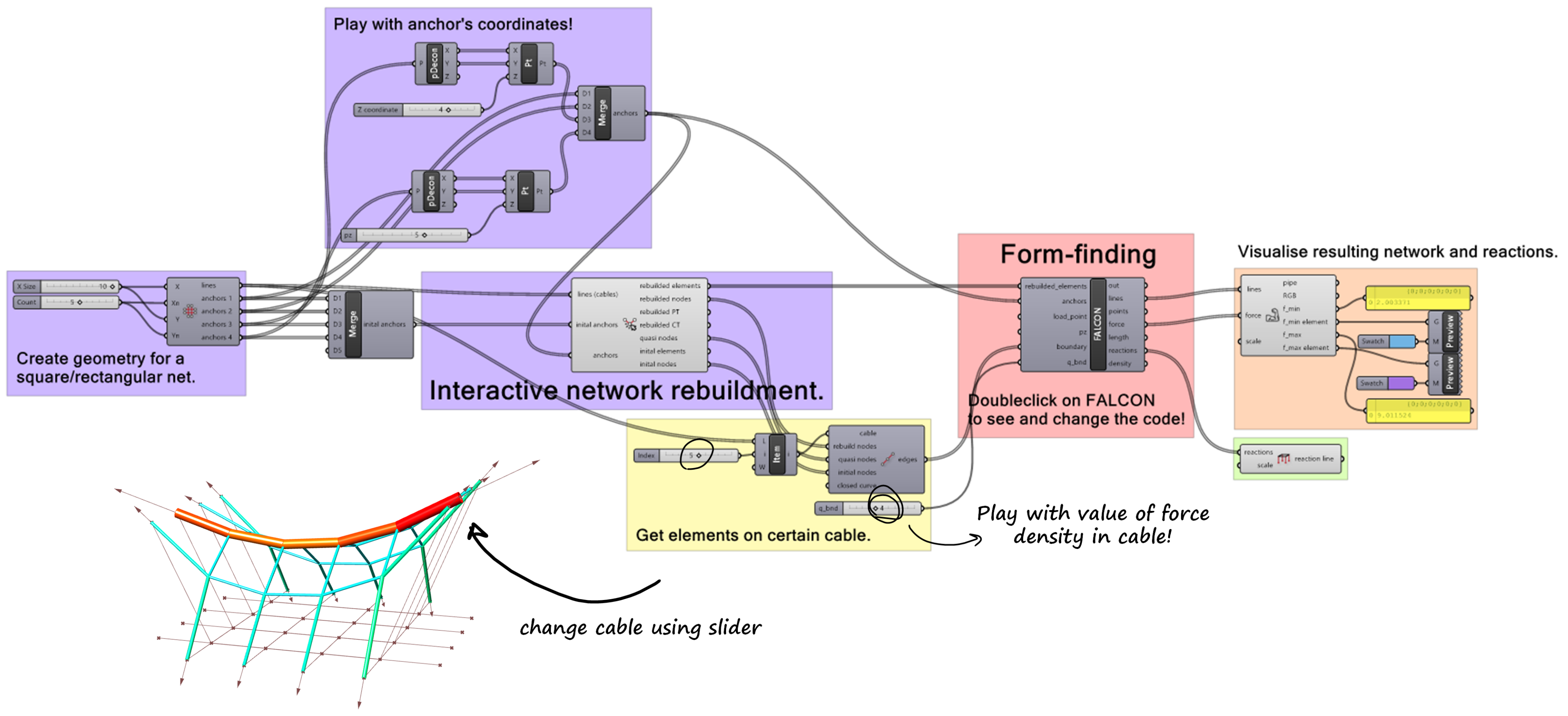
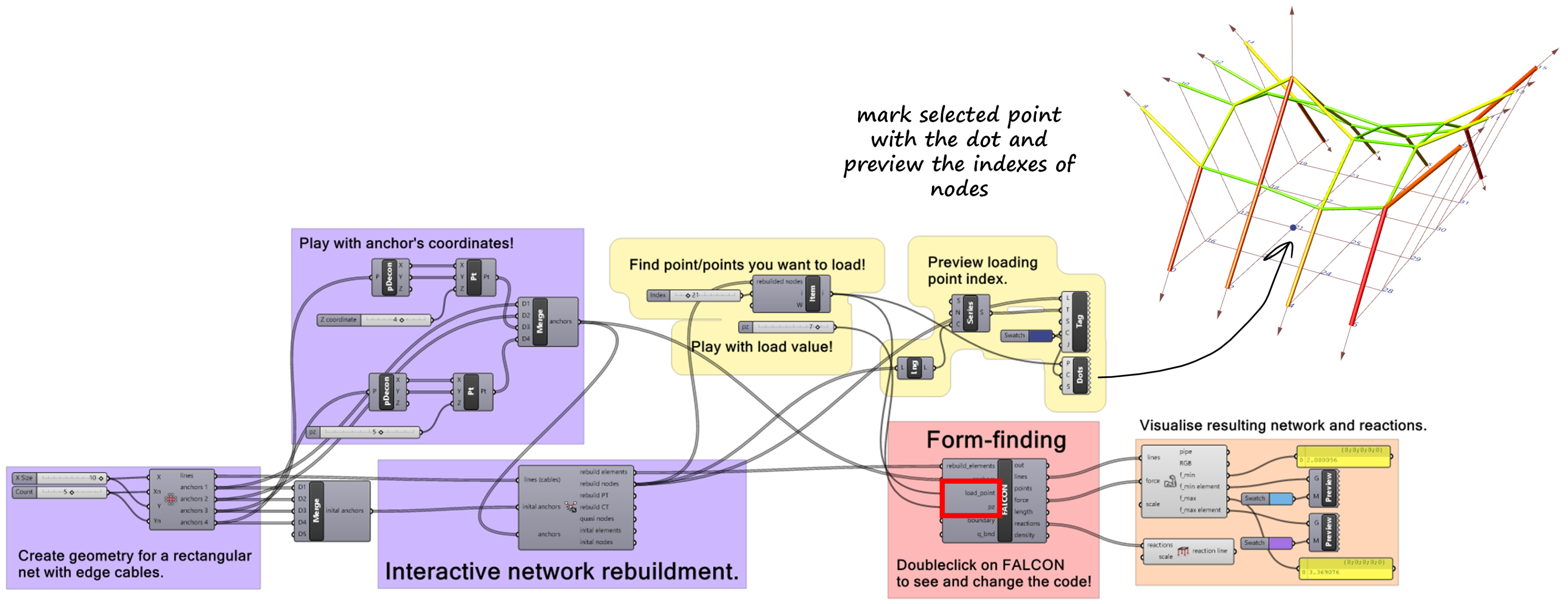

Play with anchor's coordinates!
Create geometry for a square/rectangular net.
Interactive network rebuildment.
Get elements on certain cable.
Form-finding
Doubleclick on FALCON to see and change the code!
Visualise resulting network and reactions.
Play with value of force density in cable!
change cable using slider
FALCON
Merge
Item
Preview
Swatch

Figure 16: Increasing force density value in selected cable.



mark selected point with the dot and preview the indexes of nodes
Play with anchor's coordinates!
Find point/points you want to load!
Preview loading point index.
Play with load value!
Create geometry for a rectangular net with edge cables.
Interactive network rebuildment.
Form-finding
Doubleclick on FALCON to see and change the code!
Visualise resulting network and reactions.

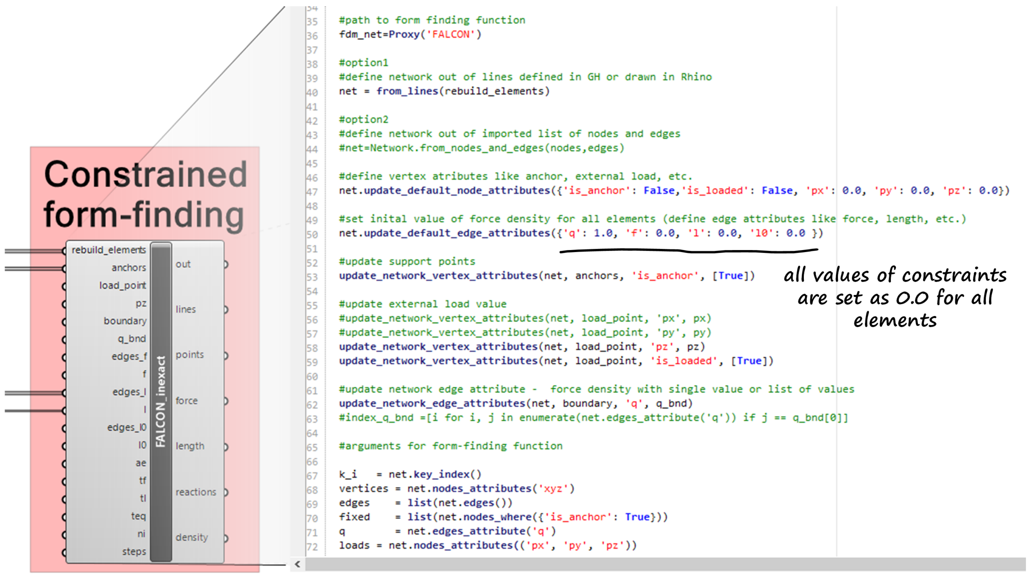
Figure 17: External load can be assigned in the nodes of the structure.

2.6 How to add constraints

The procedure for assigning constraints is similar to that for force density. For the selection of elements to be constrained, we use the same two components from the group *Network - Interactive elements on cable* and *Interactive element*. While the selection of the elements on the cable, which we will now use again, has already been explained in the previous section, you can see the use of the other component in Fig. 23.

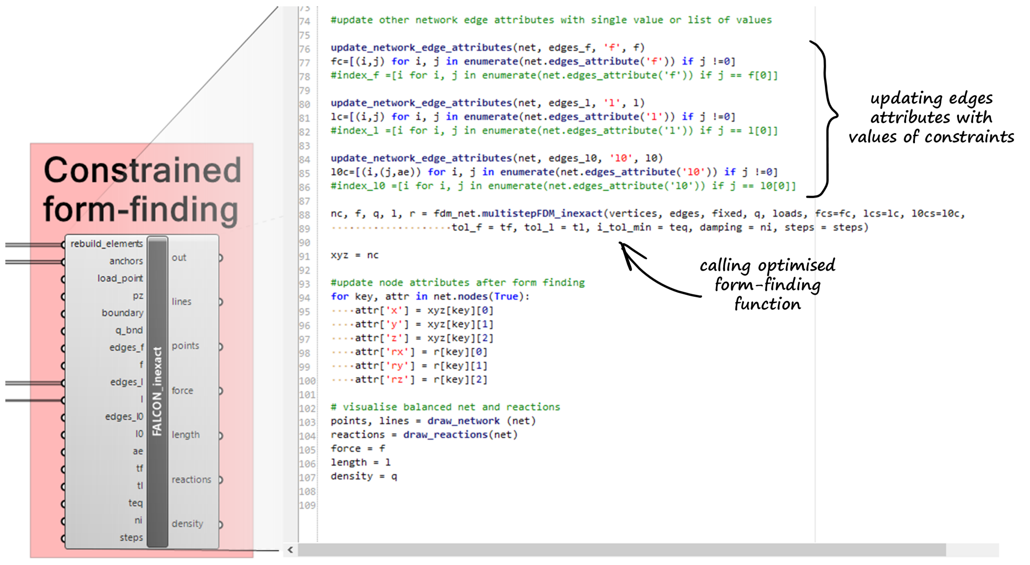
After selecting the elements, one of the three remaining components from the TENSION group in Fig. 8 can be used for form-finding, depending on the required speed and accuracy. The component FALCON _inexact contains the optimised algorithm developed as part of a doctoral thesis by the author. The other two components, FALCON _multistep and FALCON _tolernace, also apply FDM iteratively until either a specified number of steps or force/length tolerances are satisfied. These components are slower than _inexact, but for smaller networks or networks with fewer constraints, they are fast enough to perform interactive form-finding and fewer parameters need to be defined.

The parts of the code that allow constraint assignment are highlighted in Figs. 18 and 19, the rest of the code is the same as in the component FALCON except for the called form-finding function.



```python
#path to form finding function
fdm_net=Proxy('FALCON')

#option1
#define network out of lines defined in GH or drawn in Rhino
net = from_lines(rebuild_elements)

#option2
#define network out of imported list of nodes and edges
#net=Network.from_nodes_and_edges(nodes,edges)

#define vertex atributes like anchor, external load, etc.
net.update_default_node_attributes({'is_anchor': False,'is_loaded': False, 'px': 0.0, 'py': 0.0, 'pz': 0.0})

#set inital value of force density for all elements (define edge attributes like force, length, etc.)
net.update_default_edge_attributes({'q': 1.0, 'f': 0.0, 'l': 0.0, 'l0': 0.0 })

#update support points
update_network_vertex_attributes(net, anchors, 'is_anchor', [True])

#update external load value
#update_network_vertex_attributes(net, load_point, 'px', px)
#update_network_vertex_attributes(net, load_point, 'py', py)
update_network_vertex_attributes(net, load_point, 'pz', pz)
update_network_vertex_attributes(net, load_point, 'is_loaded', [True])

#update network edge attribute -  force density with single value or list of values
update_network_edge_attributes(net, boundary, 'q', q_bnd)
#index_q_bnd =[i for i, j in enumerate(net.edges_attribute('q')) if j == q_bnd[0]]

#arguments for form-finding function

k_i    = net.key_index()
vertices = net.nodes_attributes('xyz')
edges    = list(net.edges())
fixed    = list(net.nodes_where({'is_anchor': True}))
q          = net.edges_attribute('q')
loads = net.nodes_attributes(('px', 'py', 'pz'))
```

Figure 18: Code inside FALCON_inexact component - part 1.



```python
#update other network edge attributes with single value or list of values

update_network_edge_attributes(net, edges_f, 'f', f)
fc=[(i,j) for i, j in enumerate(net.edges_attribute('f')) if j !=0]
#index_f =[i for i, j in enumerate(net.edges_attribute('f')) if j == f[0]]

update_network_edge_attributes(net, edges_l, 'l', l)
lc=[(i,j) for i, j in enumerate(net.edges_attribute('l')) if j !=0]
#index_l =[i for i, j in enumerate(net.edges_attribute('l')) if j == l[0]]

update_network_edge_attributes(net, edges_l0, 'l0', l0)
l0c=[(i,(j,ae)) for i, j in enumerate(net.edges_attribute('l0')) if j !=0]
#index_l0 =[i for i, j in enumerate(net.edges_attribute('l0')) if j == l0[0]]

nc, f, q, l, r = fdm_net.multistepFDM_inexact(vertices, edges, fixed, q, loads, fcs=fc, lcs=lc, l0cs=l0c,
                    tol_f = tf, tol_l = tl, i_tol_min = teq, damping = ni, steps = steps)

xyz = nc

#update node attributes after form finding
for key, attr in net.nodes(True):
    attr['x'] = xyz[key][0]
    attr['y'] = xyz[key][1]
    attr['z'] = xyz[key][2]
    attr['rx'] = r[key][0]
    attr['ry'] = r[key][1]
    attr['rz'] = r[key][2]

# visualise balanced net and reactions
points, lines = draw_network (net)
reactions = draw_reactions(net)
force = f
length = l
density = q
```

Figure 19: Code inside FALCON_inexact component - part 2.

The inputs for the components that enable the search for the constrained form are the same as in the unconstrained case with additional inputs for lists and values of forces, lengths, and unstrained lengths that we want to assign - *edges_f* and *f*, *edges_l* and *l*, and *edges_l0* and *l0*. The input is also a product of cross-sectional area and modulus of elasticity- *ae*, which is predefined as 100 but can be changed. The component *_multistep* as input has the number of steps (set to 100). In addition to all the above inputs, the *_tolerance* component has values for tolerances for all three types of assigned values (*tf*, *tl* and *tl0* set to 0.001). The *_inexact* component requires the tolerance for the system solving *t_eq* and the "damping" constant *ni*, since it also performs the optimization of the form-finding process. All preset values can be changed on a right click, or by using a slider.

As explained earlier, the components *Interactive elements on cable* and *Interactive element* are used to select elements for which constraint values should be set. Constraint values can be defined as a single number or as a list of numbers. For a single number, a numeric slider can be used if you want to interactively see how changing the constraint affects the geometry. Defining constraints as numbers is typical for force constraints, where a uniform force through the cable must often be achieved. On the other hand, element lengths are more often defined using lists as they vary throughout the network.

Let us take our simple example and constrain the force along one of the convex cables. To select the cable and the elements on the cable, use the procedure already described in the previous chapter with the components *List item* and *Interactive elements on cable*. As output we get a list of elements on the cable, which we can now pass to the *edges_f* input in the FALCON _inexact component. The value of *f* can be assigned with a slider so that we can interactively perceive the influence of the constraint. The other inputs can be left as default values.

In addition to the forces, the lengths and unstrained lengths of the elements can also be set as constraints, as you can see in Figs. 23 and 27 .

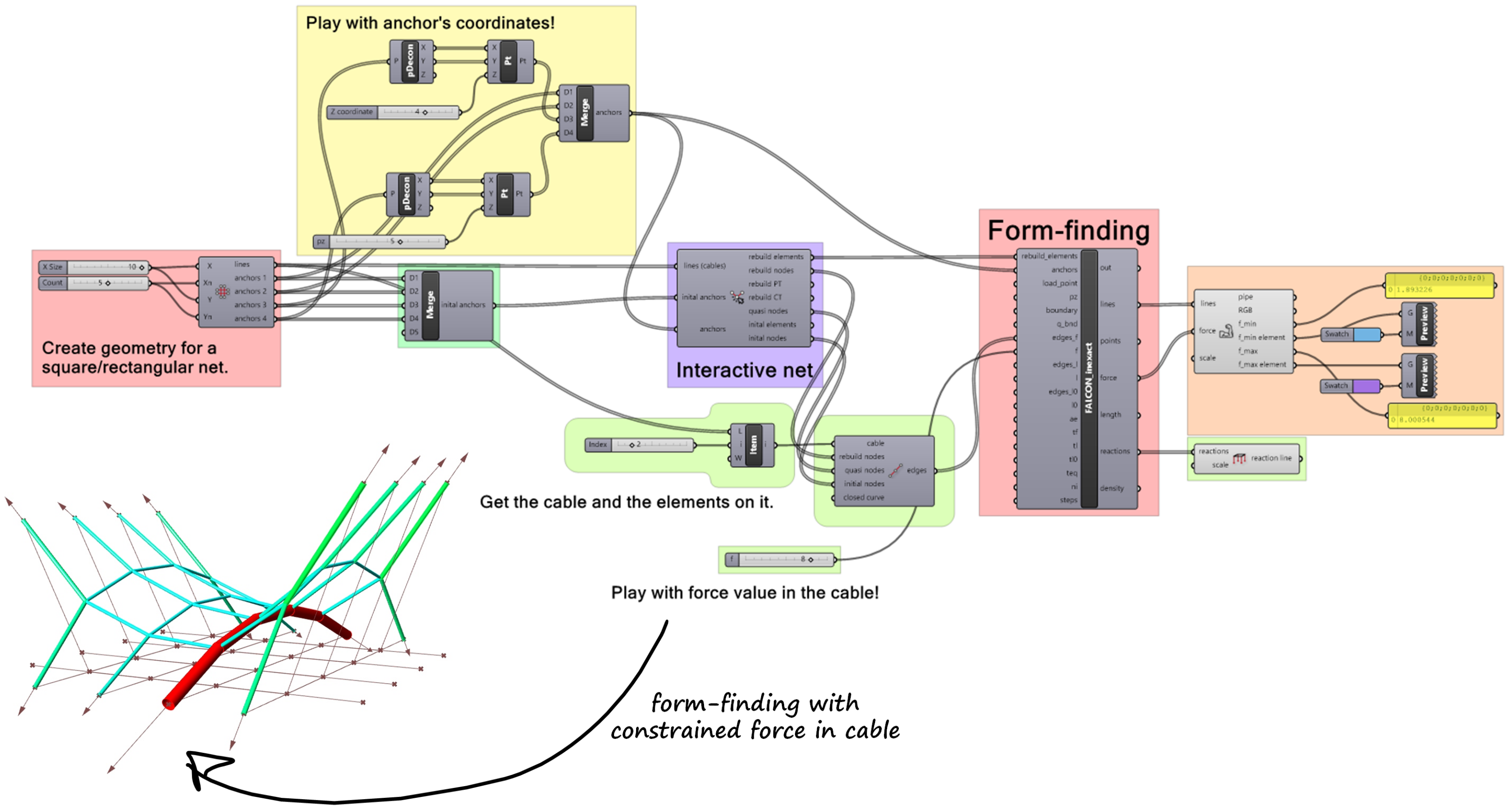
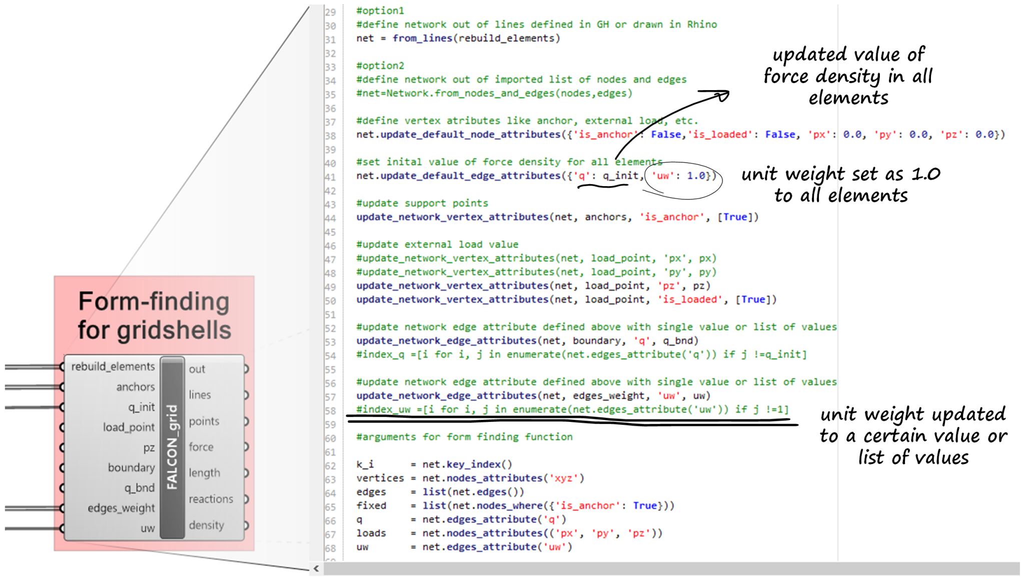

Play with anchor's coordinates!
Create geometry for a square/rectangular net.
Interactive net
Form-finding
FALCON_inexact
Get the cable and the elements on it.
Play with force value in the cable!
form-finding with constrained force in cable

Figure 20: Constraining force in cable.

3 Tension - compression analogy

All of the above also applies to structures in compression, where the equilibrium form of a gridshell is obtained by inverting the result of the hanging chain method. For this purpose the weight of the elements must be included in the calculation. To control the assignment of elements and unit weight values, two new inputs are made, *edges_weight* as the list of elements to be assigned a specific unit weight value, and *uw* for it's value, or list of values. By default, all elements are assigned a unit weight of 1.0, as you can see in Fig. 21. If we take the same example as we had in tension, and lower the lifted anchors back to the ground level, we can obtain a solution in compression by using the component FALCON_grid, which returns the mirrored shape in equilibrium (Fig. 22). As can be seen in the first image, due to the small number of divisions, the number of cables is small and therefore the shape is not appealing. Let us increase the number of divisions. By doing so, as seen in the middle picture, the weight also increases, making the shape elongated. To control the height of the shape, the value of the force density for all elements can be increased with the new input *q_init*. The input *q_init* can be found in all components from COMPRESSION group to easily change the value of the force density in all elements and increase it beyond the default value of 1.0.


Form-finding for gridshells
rebuild_elements
anchors
q_init
load_point
pz
boundary
q_bnd
edges_weight
uw
FALCON_grid
out
lines
points
force
length
reactions
density
updated value of force density in all elements
unit weight set as 1.0 to all elements
unit weight updated to a certain value or list of values

```
#option1
#define network out of lines defined in GH or drawn in Rhino
net = from_lines(rebuild_elements)

#option2
#define network out of imported list of nodes and edges
#net=Network.from_nodes_and_edges(nodes,edges)

#define vertex atributes like anchor, external load, etc.
net.update_default_node_attributes({'is_anchor': False,'is_loaded': False, 'px': 0.0, 'py': 0.0, 'pz': 0.0})

#set inital value of force density for all elements
net.update_default_edge_attributes({'q': q_init, 'uw': 1.0})

#update support points
update_network_vertex_attributes(net, anchors, 'is_anchor', [True])

#update external load value
#update_network_vertex_attributes(net, load_point, 'px', px)
#update_network_vertex_attributes(net, load_point, 'py', py)
update_network_vertex_attributes(net, load_point, 'pz', pz)
update_network_vertex_attributes(net, load_point, 'is_loaded', [True])

#update network edge attribute defined above with single value or list of values
update_network_edge_attributes(net, boundary, 'q', q_bnd)
#index_q =[i for i, j in enumerate(net.edges_attribute('q')) if j !=q_init]

#update network edge attribute defined above with single value or list of values
update_network_edge_attributes(net, edges_weight, 'uw', uw)
#index_uw =[i for i, j in enumerate(net.edges_attribute('uw')) if j !=1]

#arguments for form finding function

k_i      = net.key_index()
vertices = net.nodes_attributes('xyz')
edges    = list(net.edges())
fixed    = list(net.nodes_where({'is_anchor': True}))
q        = net.edges_attribute('q')
loads    = net.nodes_attributes(('px', 'py', 'pz'))
uw       = net.edges_attribute('uw')
```

Figure 21: Code inside FALCON_grid component.

Since the weight added to the nodes as an additional load depends on the length of the elements, in constrained form-finding - the iterative application of FDM - the values of the weights are updated at each step according to the changes in the element length. Again, the three components are created for constrained form-finding, and the constraint assignment remains the same as for structures in tension, only the form-finding function changes.

Let us adjust our example a bit. To do this, we add a circular opening in the center of the net. To place the opening in the center, we should find the center of the grid by finding the intersection of the middle cables (applies only if the number of divisions is even), as shown in Fig. 23. For this purpose, the list of lines is split to cables in each direction by setting the splitting index in the component *Split list* as number of divisions minus one (since the list starts with 0). Then the middle cables are selected by setting the index in the *List item* component to a value by one less than half the split number. The *Curve | Curve* component provides the intersection point, which is then used as the central point for creating the circle, along with the slider for the radius, both of which are plugged into the *Circle* component. Once the circle is created, we will use it to create elements based on its intersections with inner cables. To do this, we can use the *Interactive elements on boundary curve* component from the *Geometry* group. By inserting the circle as *boundary curve*, *lines* as *inner cables* and setting *Boolean Toggle* component as True for the *closed curve?*, we get the list of elements at the location of the circle. To remove part of the cables that are inside the circle, you can use the *Trim with Region* component, where the circle is the region, *lines* are curve, and xy is set as the plane. The output *Outside (Co)*, together with the created elements from the circle, will form the lines that will be plugged into the *Interactive net* component. We can use the _inexact component for form-finding, keeping all the basic inputs the same as in the previous example. To get the elements of the opening from the newly created interactive net we use the *Interactive element* component, where *boundary elements* are *selected elements* and the rest of the inputs come from the *Interactive net* component. It is to these elements that the lengths will be constrained, so we pass them to the input *edges_l* and select the value with the slider for the input *l* (see Fig. 23).

If you want the values of forces or lengths to be displayed only for constrained elements, you can enable lines of code 87, 91 or 95 to get the indices of these elements and set them as the output of the component. Then these indexes can be used to filter out the lines and values that correspond to the elements using List item component and preview them using the *Show values* component.

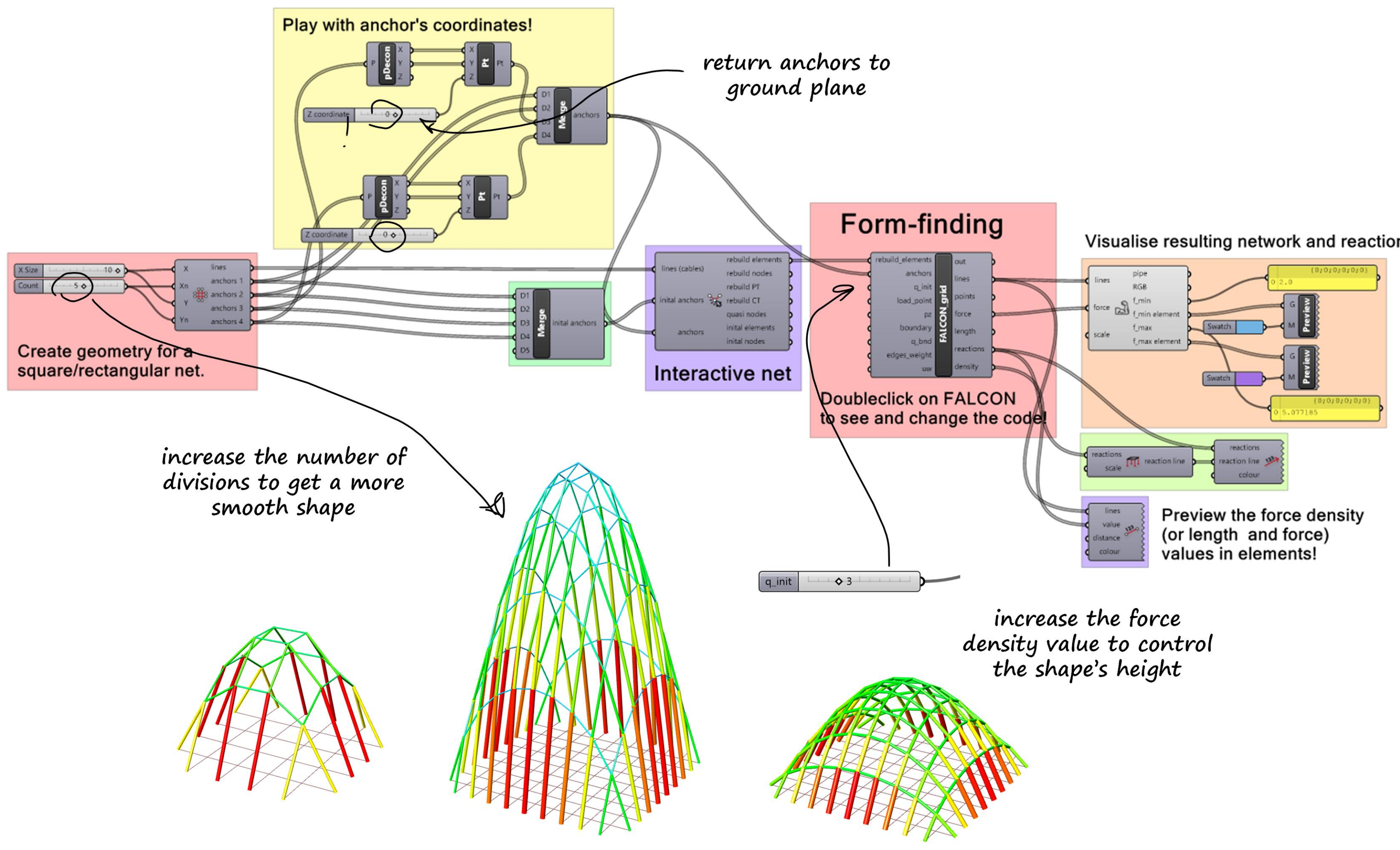
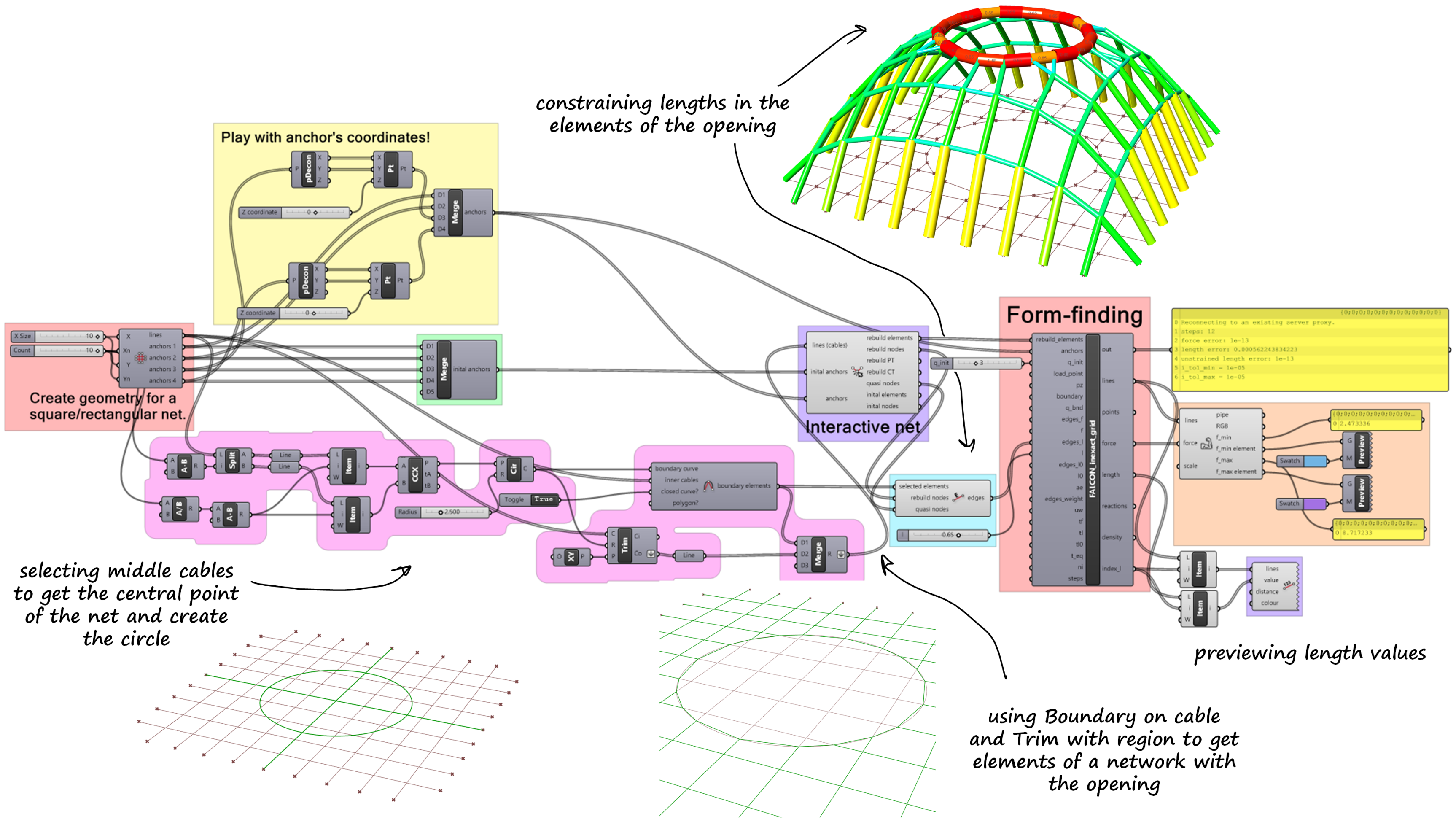

Play with anchor's coordinates!
return anchors to ground plane
Create geometry for a square/rectangular net.
Interactive net
Form-finding
Doubleclick on FALCON to see and change the code!
Visualise resulting network and reactions.
Preview the force density (or length and force) values in elements!
increase the number of divisions to get a more smooth shape
increase the force density value to control the shape's height
q_init
FALCON grid
Merge
pDecon
Pt
Z coordinate
X Size
Count
Preview
Swatch

Figure 22: Form-finding of gridshell based on tension-compression analogy.


constraining lengths in the elements of the opening
Play with anchor's coordinates!
Create geometry for a square/rectangular net.
Interactive net
Form-finding
FALCON_inexact_grid
selecting middle cables to get the central point of the net and create the circle
using Boundary on cable and Trim with region to get elements of a network with the opening
previewing length values

```
{0;0;0;0;0;0;0;0;0;0;0;0;0}
0 Reconnecting to an existing server proxy.
1 steps: 12
2 force error: 1e-13
3 length error: 0.000562243834223
4 unstrained length error: 1e-13
5 i_tol_min = 1e-05
6 i_tol_max = 1e-05
```

Figure 23: Constraining lengths of elements in ring cable.

4 Some examples to try by yourself

In this chapter you will find some examples that you can reproduce yourself. The first two examples were created with the component *Net with edge cables*. For structures with edge cables, it is essential to increase the value of the force densities in the edge elements, as explained earlier. Apart from selecting these elements and increasing q_bnd, the second example shows how you can add additional anchor points on cables in a few steps and easily modify the original predefined mesh defined by the FALCON component.

The interactivity of the tool is highlighted in the third example, the parametric model. Here we show how the model created with the *Net with boundary curve* component can completely change its shape by changing a single parameter. Like the components before, this component has inputs for the dimensions and the number of divisions in the x and y directions, but also has an input for the curve that defines the boundary cable. Another input is the number n, which defines the output *selected points*. By defining n, we select every nth point where the boundary cable and the net intersect. In this way, we can change the number of anchors very easily. For example, if we use the *Cull Nth* component, we can remove every second point from the selected points to raise half of the anchors. If we use the *Find similar* component in combination with *Replace*, we can replace the anchors at the ground with raised ones and get two lists that we need for form-finding - *initial anchors* and *anchors*. In this example, we have used the *Pick single* component from the drop-down menu of the *Network* group. Since we have not mentioned this component before, let us explained it a bit. As input it takes all the elements of the network as well as the elements we do not want to select, and returns the elements that are not on both lists. This way, for example, you can easily select all inner elements since in most cases we the edge elements are already selected with one of the components mentioned before. In this example, force constraints are applied to the inner elements to achieve an even distribution of forces in the network. The elements are selected by plugging in *rebuild elements* from *Interactive net* component and *edges* from *Interactive elements* component (see Fig. 26).

With the last example, a gridshell based on a Voronoi diagram, we wanted to show that you can be creative and do not have to stick to the geometry generation provided by FALCON. Here, a curve drawn in Rhino was used as the boundary for the Voronoi pattern. The curve can be modified in Rhino and the net will change interactively. The pattern can also be manipulated with sliders, as shown in Fig. 27. Here the unconstrained lengths are set for all elements. The tolerance for the unconstrained lengths can be changed with the slider depending on the desired accuracy. Feel free to draw different curves and explore the structure you get by changing the parameters for Voronoi!

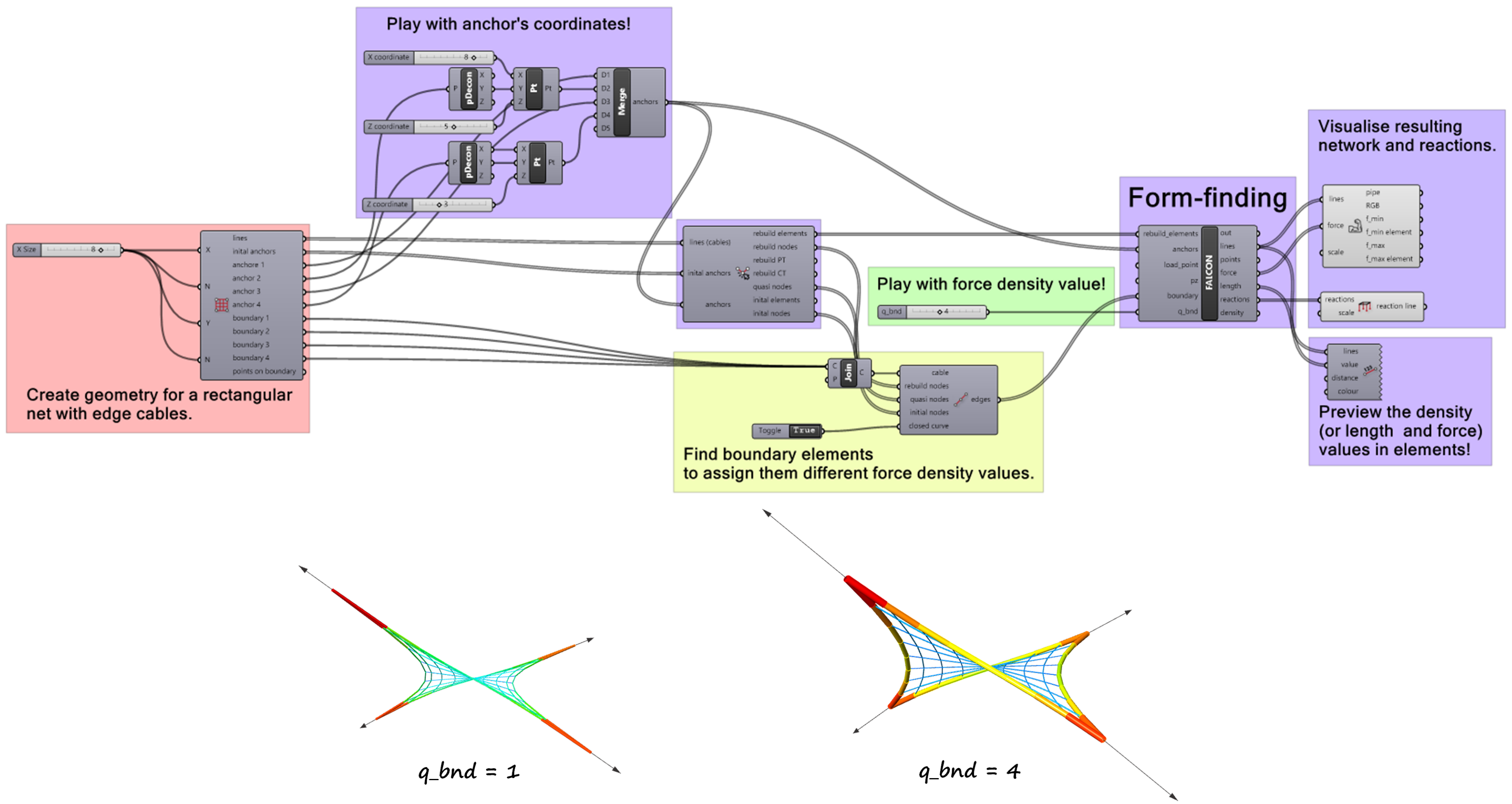


Figure 24: Increasing force density value in boundary cables.

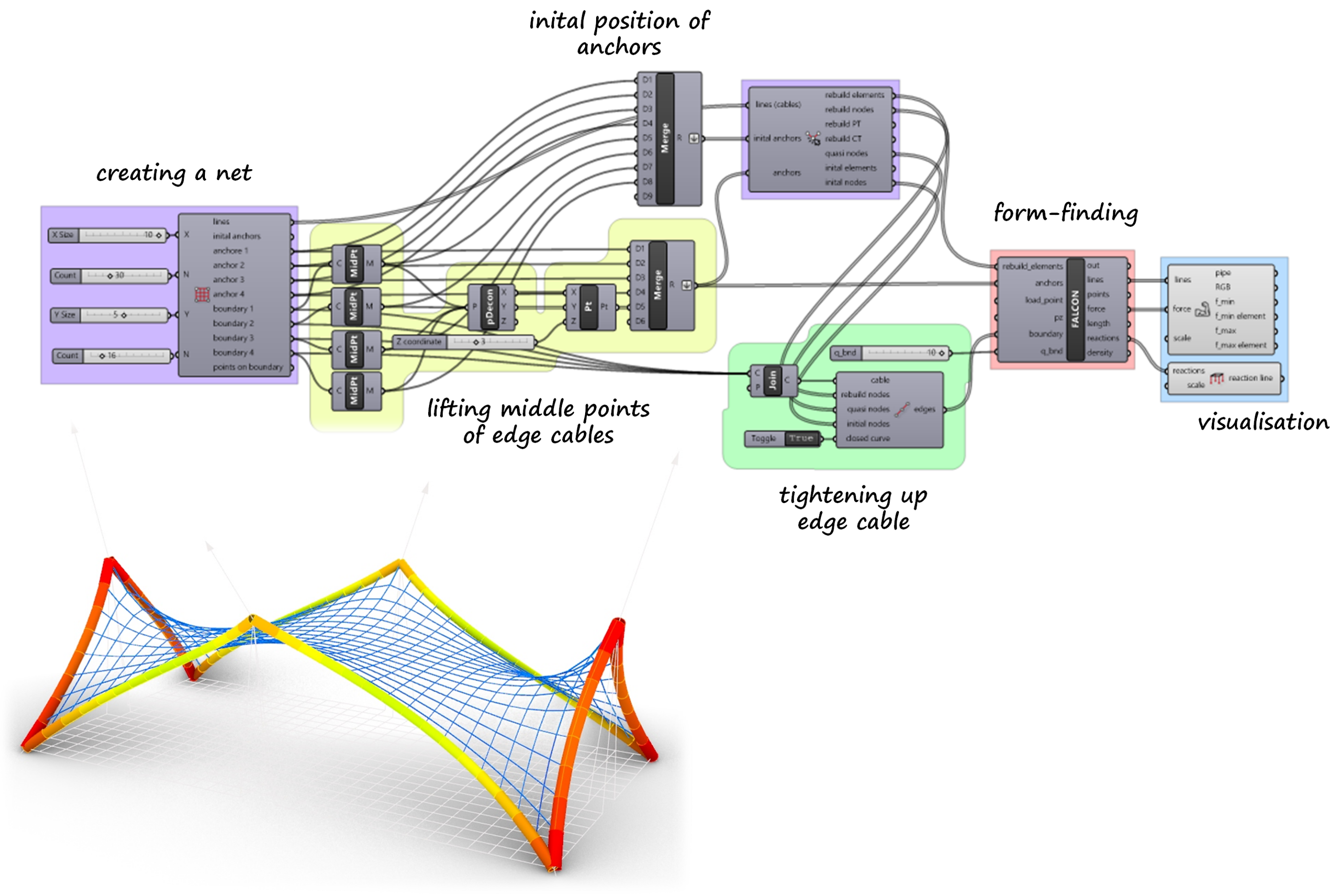


Figure 25: Tightening up boundary cables.

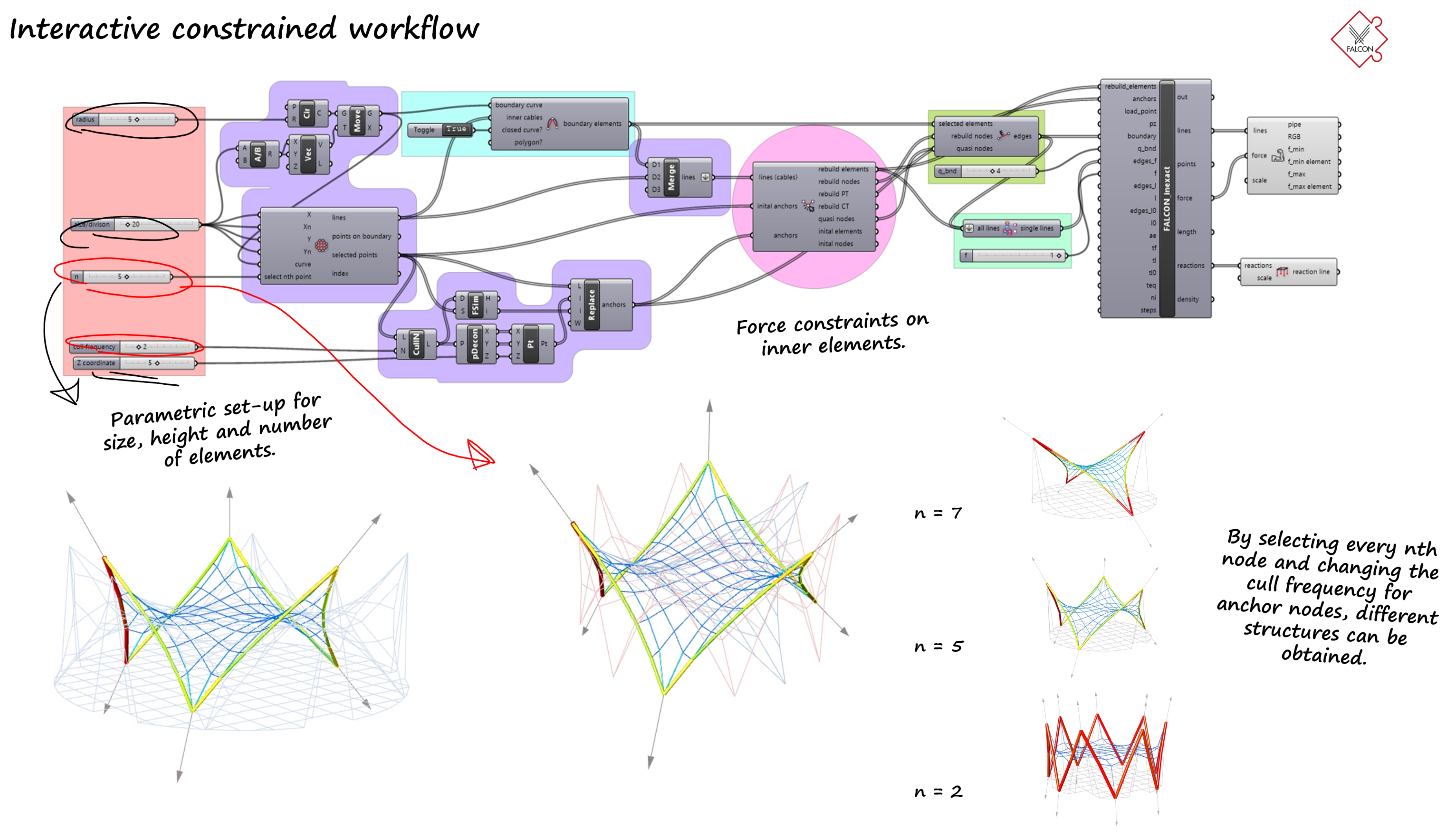


Figure 26: Parametric model.

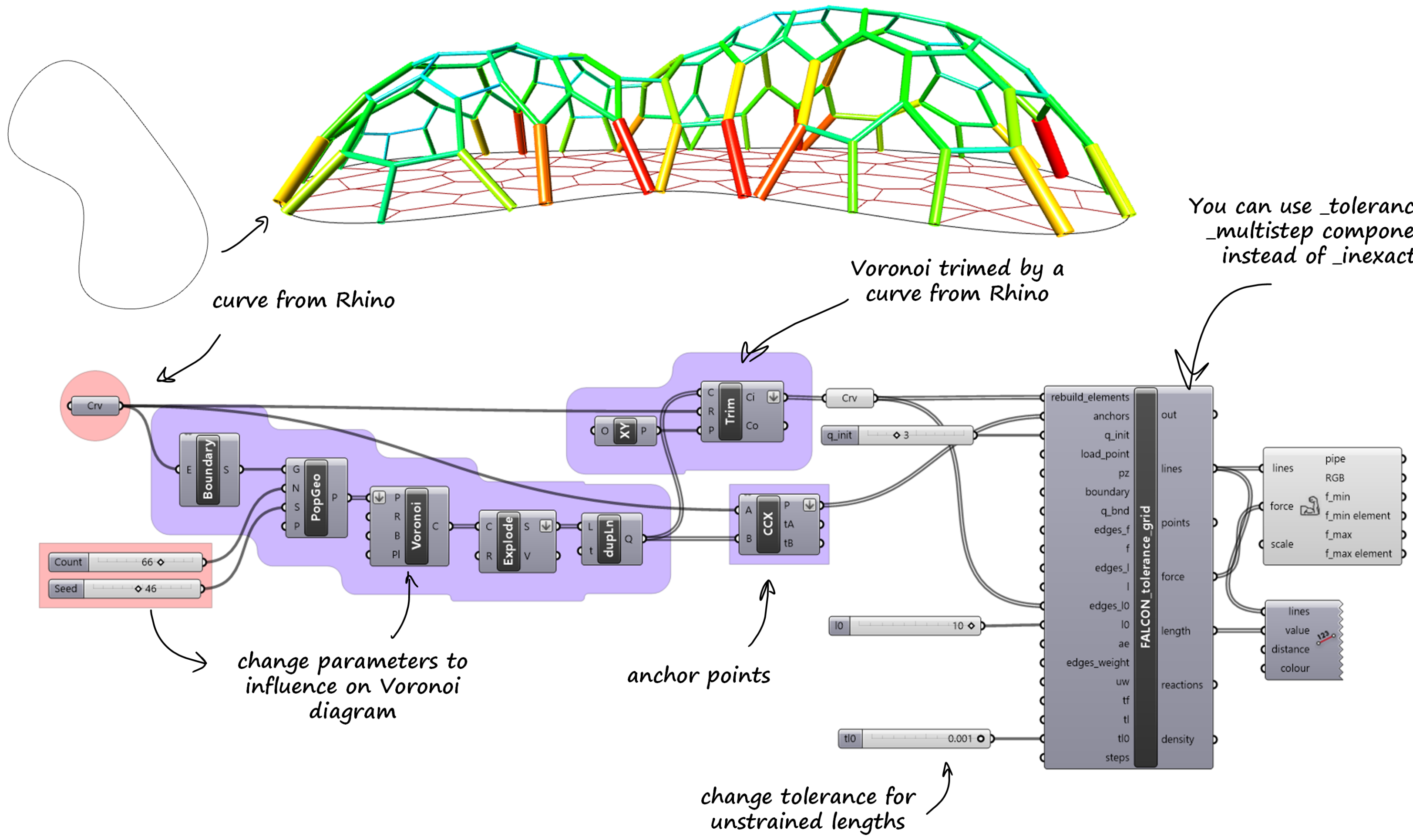


Figure 27: Constraining unstrained lengths for elements of Voronoi model.

5 Hidden gems

In all of the GHPhyton components shown, the functions called to determine the equilibrium form are executed in Python. Since the tool is primarily intended for educational use, the components presented are fast enough to allow interactive, feedback-based form-finding, and they are robust enough to handle any change in input parameters or even incorrectly assigned inputs. For research or practical use of FALCON, the components calling form-finding functions written in C++ are available in the Solver's drop-down window. These components can be used to find the shape of structures in tension as well as in compression. The components can speed up form-finding, but can also be unstable due to the conversion of data between Python and C++, especially for constrained form-finding due to a high number of arguments. Since these components are still under development, advanced users are recommended to find the shape with the default components to better understand the behaviour of the structure and the influence of the assigned parameters, and then use the faster components to explore different possible solutions by changing the supporting conditions or the assigned constraints.

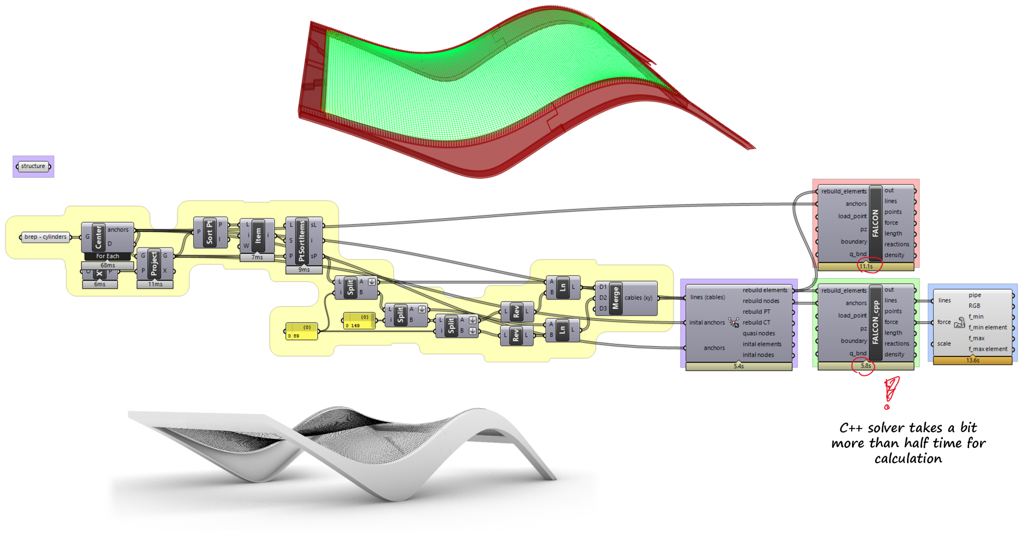


Figure 28: For examples with lot of elements, like Poljud stadium (26760 elements), one can use components with C++ solver for faster calculation.

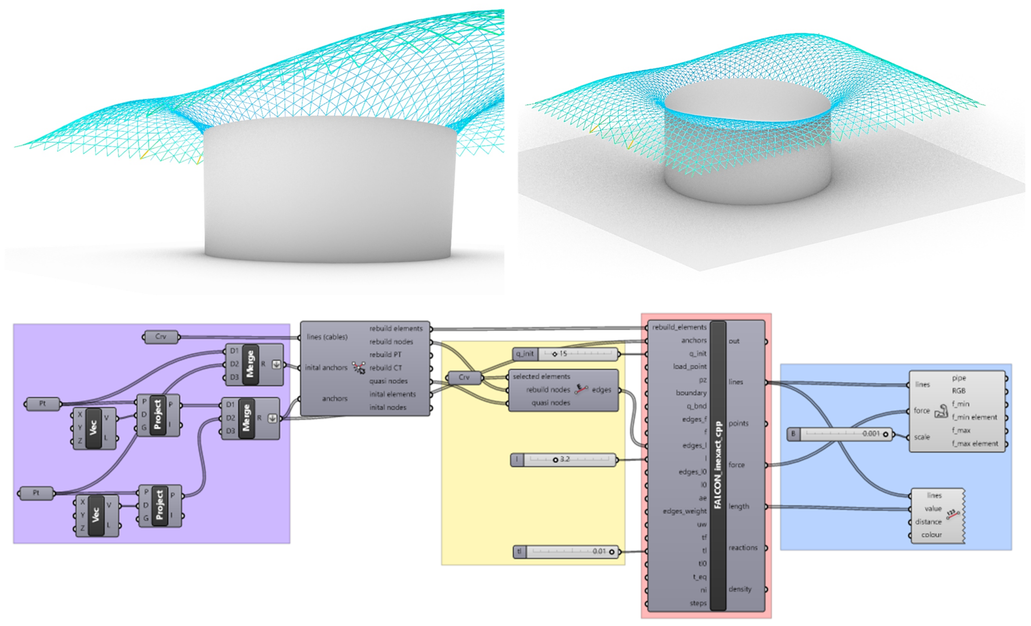


Figure 29: Using C++ components for constrained form-finding of "heavy" examples like model of Great Court gridshell at the British Museum.

6 Good luck!

We hope you find this handbook useful, and that the FALCON tool will support you well in creatively and actively designing equilibrium networks. Do take a look at our FALCON page for video tutorials! Please also let us know if you have any feedback or if you find a bug (e-mail: esamec@grad.hr).