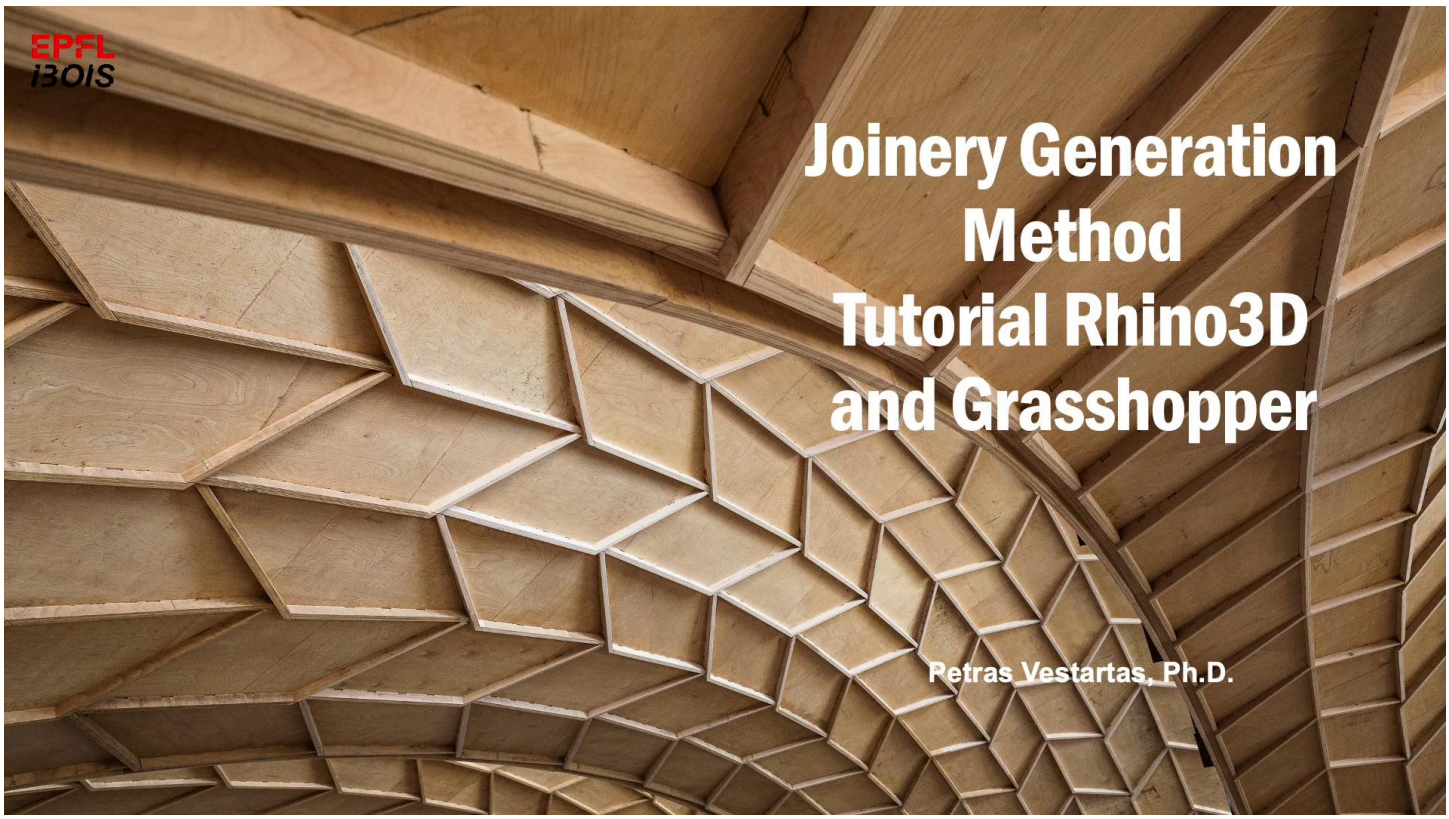
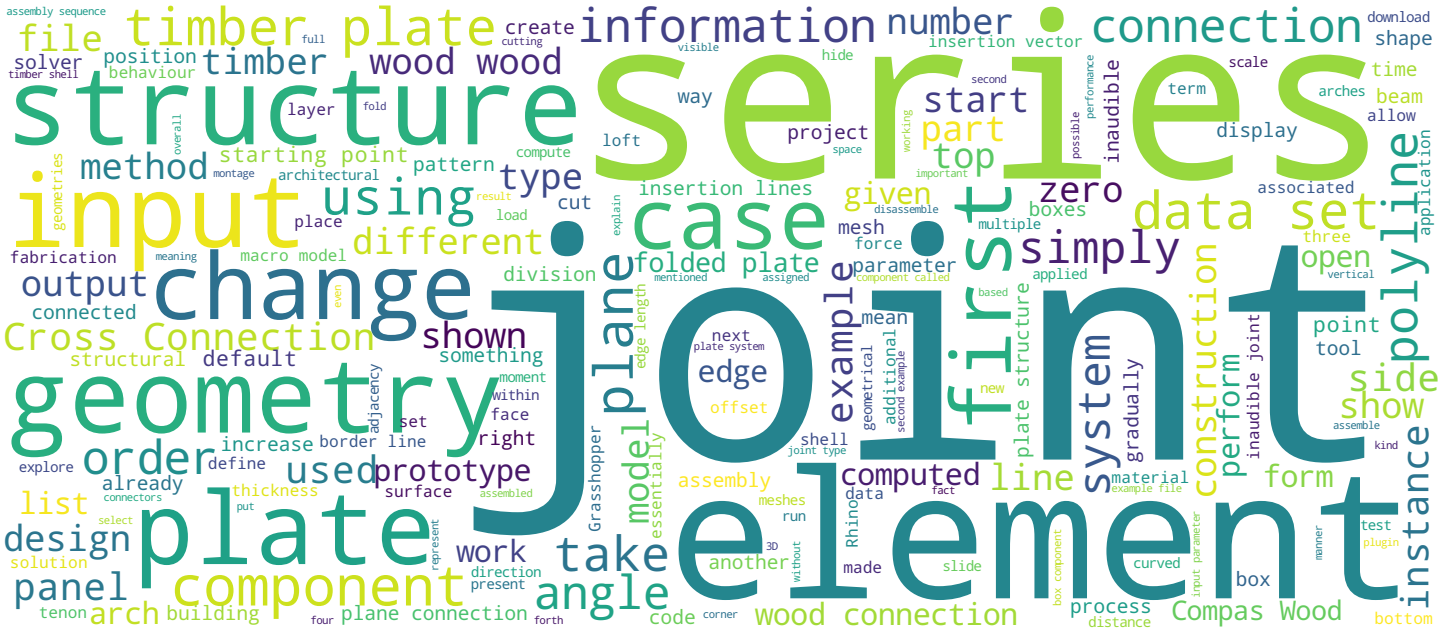
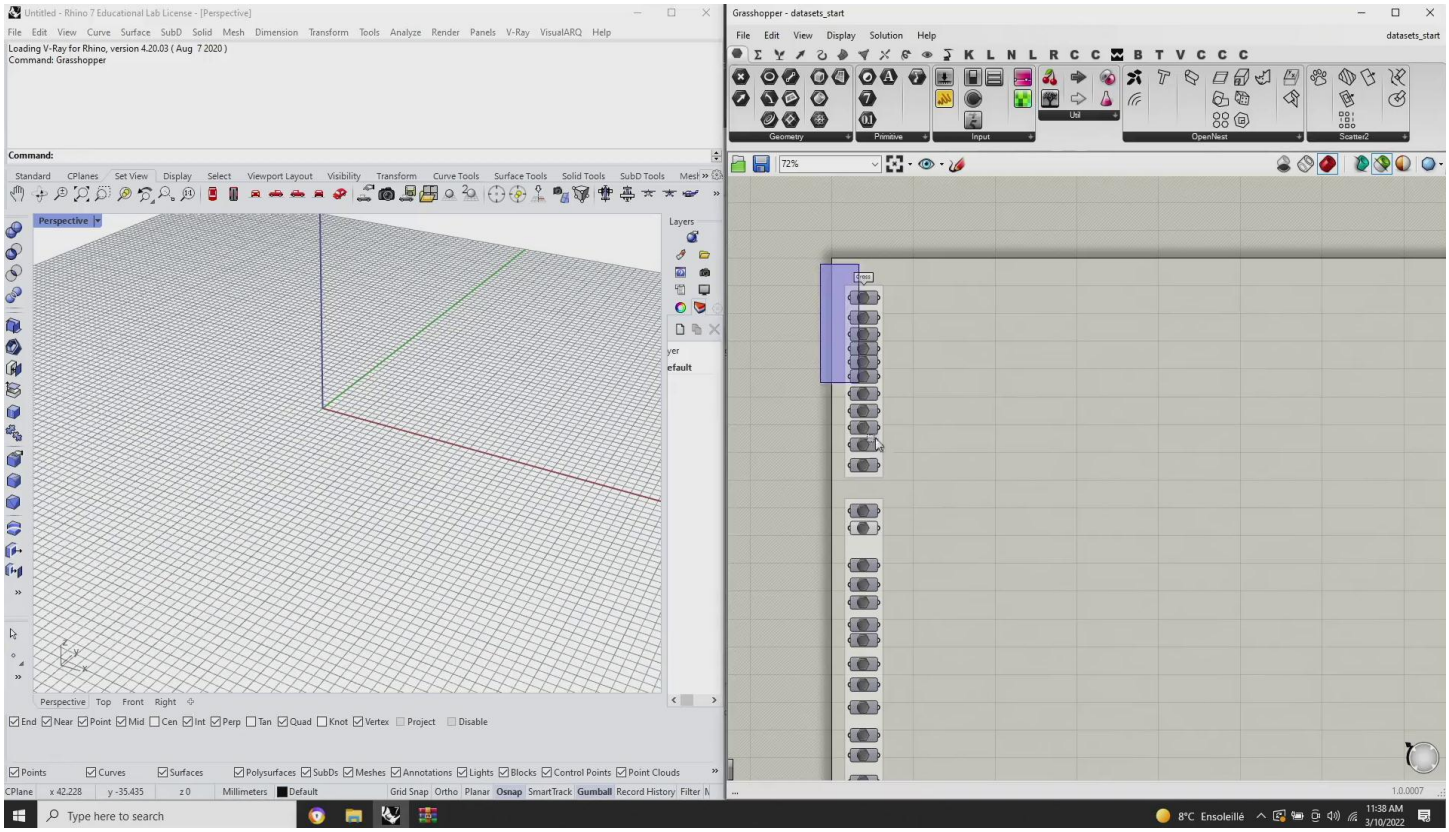

EPFL
IBOIS
Joinery Generation Method Tutorial Rhino3D and Grasshopper
Petras Vestartas, Ph.D.


joint
series
structure
element
geometry
input
change
case
first
plane
plate
component
polyline
system
construction
instance
connection
information
timber
wood
method
using
different
folded plate
simply
data set
open
side
show
form
line
computed
take
angle
model
Grasshopper
Rhinoplate
Cross Connection
output
order
used
prototype
design
panel
arch
building
wood connection
Compas Wood
process
curved
box
Search MOOC
Video
EPFL

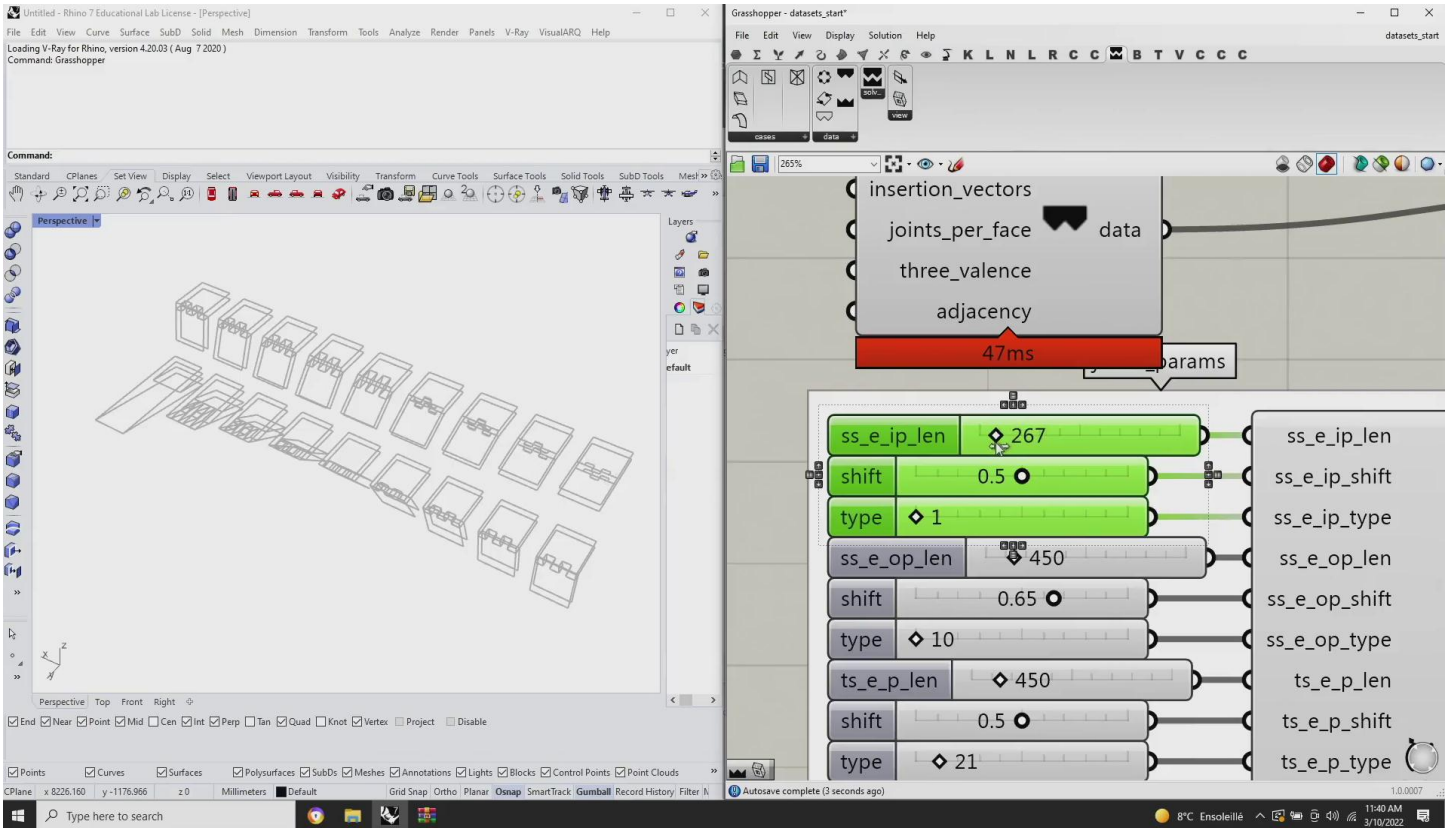


First, let's go to the release repository of the Compas Wood in order to download the Compas with example files you're going to use today as a starting points for the Rhino and Grasshopper. If you download these are files, we can see that there are series of files in the example folder, and we are going to use only the starting points. These are the starting points that we're going to use and the end parts and the actual solutions to the examples. For instance, I can take these samples, I can go to My Computer, Compas Wood course, and we can simply place those starting points into the folder, and we actually will go through them. The chevron or the onion or the segment timber shell, we did already last time. Now we're going to look at the data sets, then the Cross-Connection and the Folded Plate System as an example for this lecture, Let's start from the data sets. If you have Rhino open, you can simply drag and drop the files. Take this data set starting point and drag and drop to Rhino. You will open the file that already has a lot of geometries internalised. So there are a bunch of geometries that can be used for testing different example files that we actually use to develop this application.

Notes

Summary

0m 04s

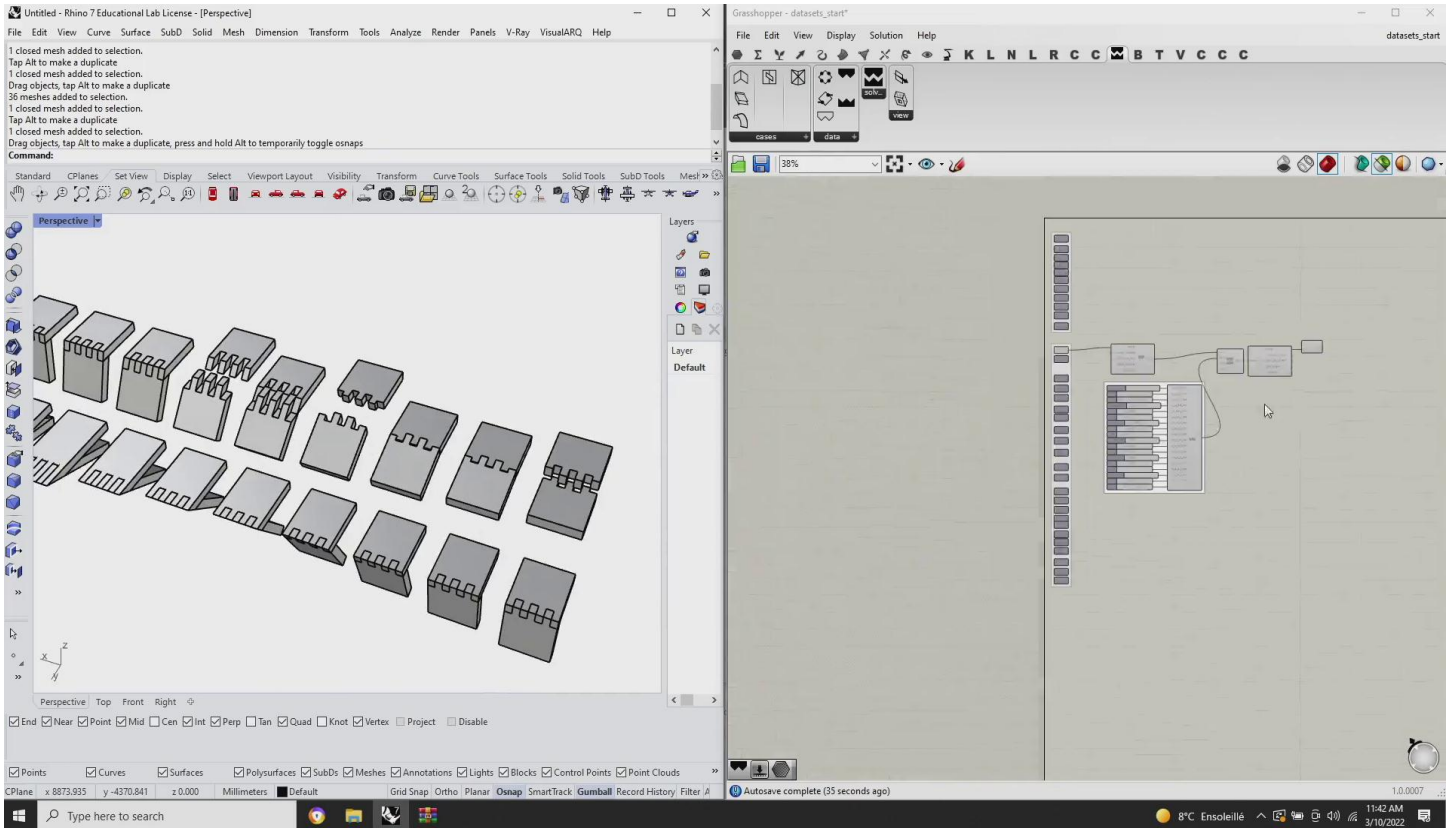


And the first joint that you can see is simply the Cross-Connection. And now what we're going to do, we start not from the top part, but from the bottom, which is Side-Side Connections. As I mentioned before, there was one sample that shows different angles. So this was a data set of that. And we would like to generate a series of timber plates. And those timber plates must adapt to the different fabrication angle. So when we go to the Compas Wood, we simply can start by taking the solver that's responsible for computing the geometry. Just Preview on. Then we need to convert this geometry into solver readable geometry. That's for we need to have the input set component that is visible here. Then we take the list of polylines we join them to data set. And you see that this something computed. Let's take the proper order, take list of parameters, put them to the joint input. We can change for the implant joints this type. And since we can take number two, which is a joint type number for two-point planes and this group is controlling, controlling the joint number depending on the edge length. We see that those joints are changing depending on the edge length.

Notes

Summary

2m 00s

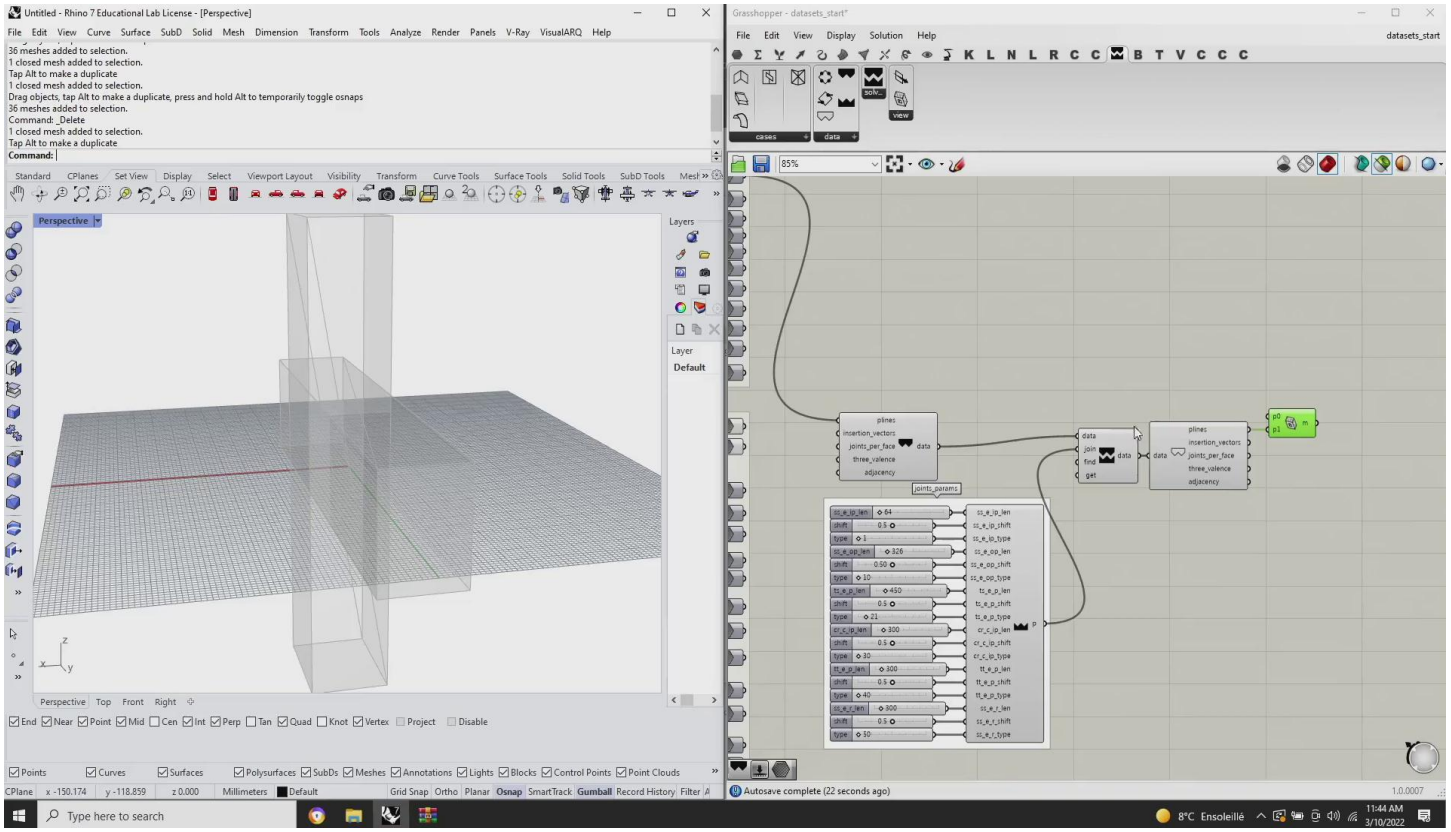


And then another group is actually dependent, or controlling those ones that are out of plane connections meaning that the angle is changing between the planes, you actually have not the same joint, but so called Out-of-Plane Connection. With this SSEOP, which is out of plane, and then SSEIP, this in plane. For instance, we can change parameters, if you'd like to have a dot-feel connection, you can change that. If you want to change an angle, you can increase the number of joints depending on the division value. And then you see that the joint is not working anymore, but then angle is really sharp, and you cannot really use it for any fabrication method. Therefore we can now go and take a loft component. We also need to output the information. So we use this component to convert data to the polylines. And you can simply connect polylines to p one. In this case, you can now display the geometry. I will select the component and click insert, in order to have the Rhino geometry. And you can see here that the joints were created as a series of meshes. And these are the top tails and so on and so forth. We can explore also different data sets as we like but I'm going to present the basic principle that there are in-plane and out-of-plane connections in the set.

Notes

Summary

3m 55s

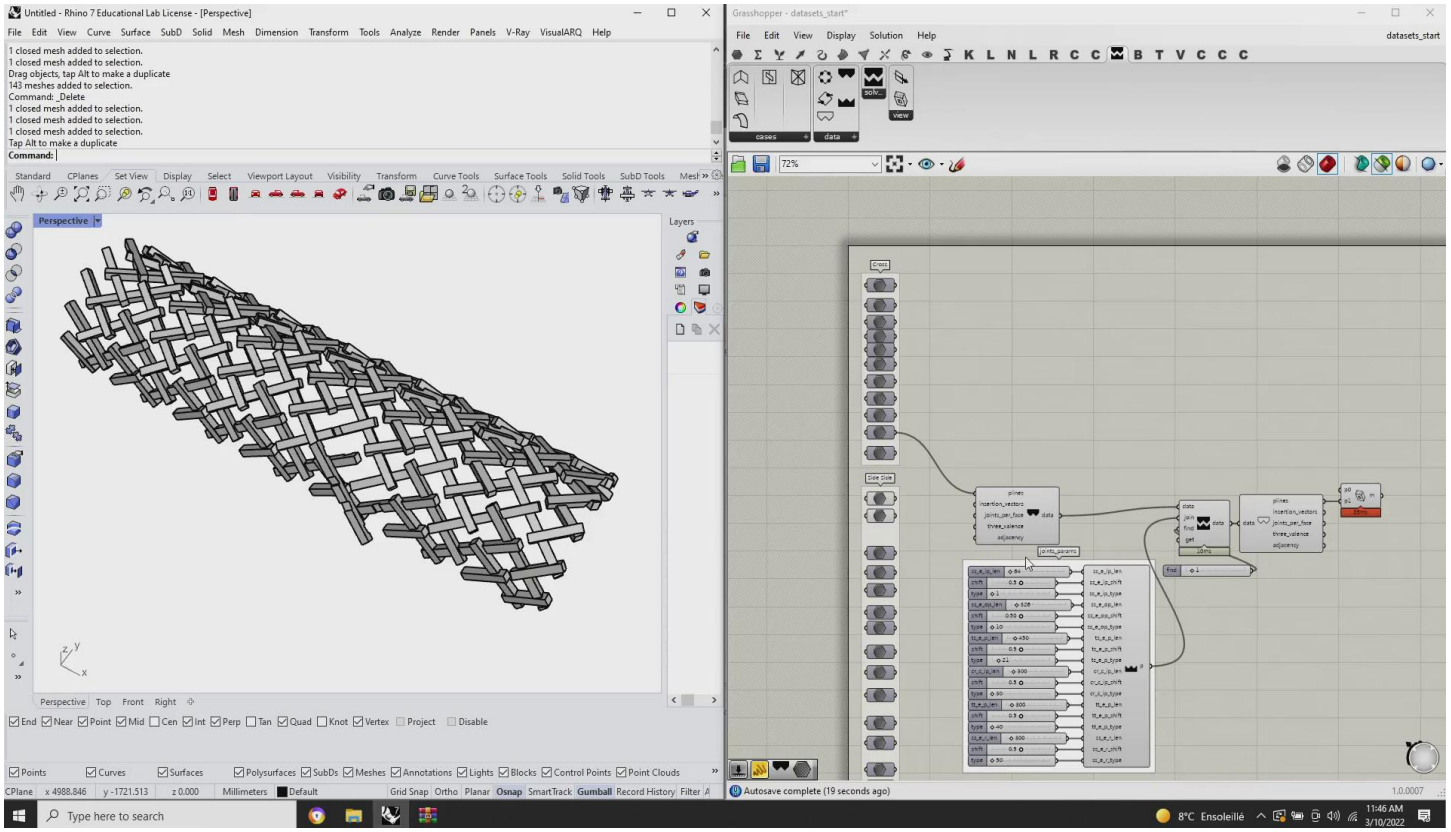


For instance, I'll take another set, which is a planarised shell, they have a this semi-hexagonal structure that you'd like to input a list of polylines, then we can see something is getting computed. And this is actually the [inaudible 00:06:06] joints. And you can change the the number of joints. You can loft the geometry to display it in 3D, you can change the angle between the joints. You can also change the type and so on. So if we bake, this is geometry, We can see this is a valid geometry to work with. So this is the second example I showed you during the presentation. And now we're going to show a series of Cross-Connections that we've shown as well, in terms of understanding how we can intersect joints, not in the plane-to-plane intersection but plane-to-face intersection. And we can simply take a data set that is coming from here, and we can also input it to the planes. And for this, we need to change the defined input from zero to one. There are actually three types of searches. The first one is side-to-side. The other one is plane-to-face, which is used for cross. And then the third one tries to perform both of them.

Notes

Summary

5m 46s

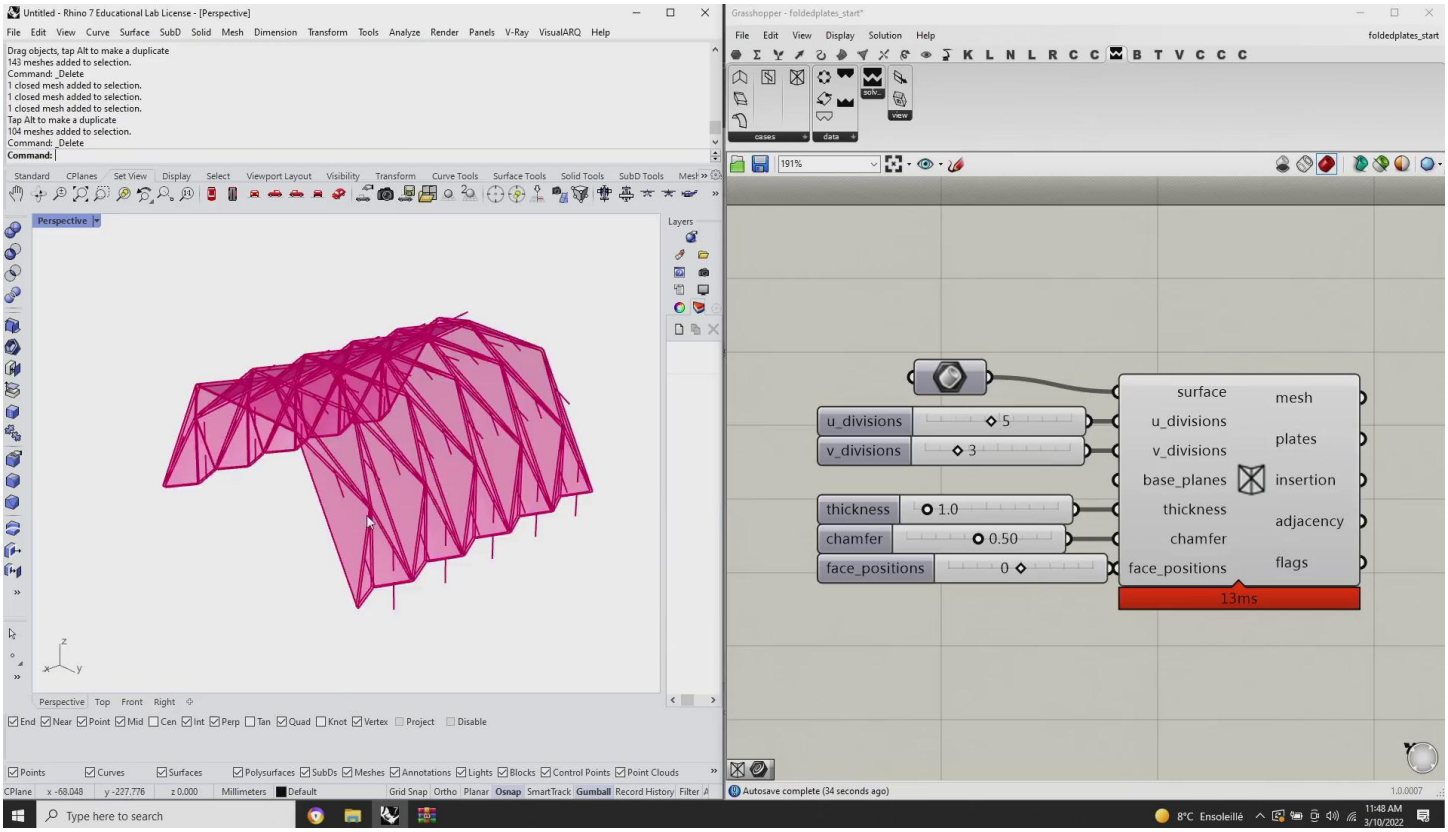


So let's take number two, place it to the number two, and you see that the joint is computed like this and have this simple connection case. And then we can explore other solutions. See here, sometimes it's possible to compute side-to-side and cross-connection. So that's why you need to be explicit at some moments, if you like to find both or just one. In this case, I would like to find just the plane-to-face connection, which is a cross-connection. That's for you can change this input. And then you can see that you can have these connections cut out. There were other samples as well. We can find something more interesting to look at. This actually is a series of plates in different intersection scenarios, also these beam cases. And then there is also, let's see a bit different structure we had this hexagonal shell in one manner that was using the in-plane connections. And now we have the cross-connections. Then there are some other data sets as well. For instance, take a look at the source of protocol system. And you have a series of rectangular beams, and you can simply input the top and bottom outline, and you can get the full geometry afterwards. So this is a basic principle of the code.

Notes

Summary

7m 34s

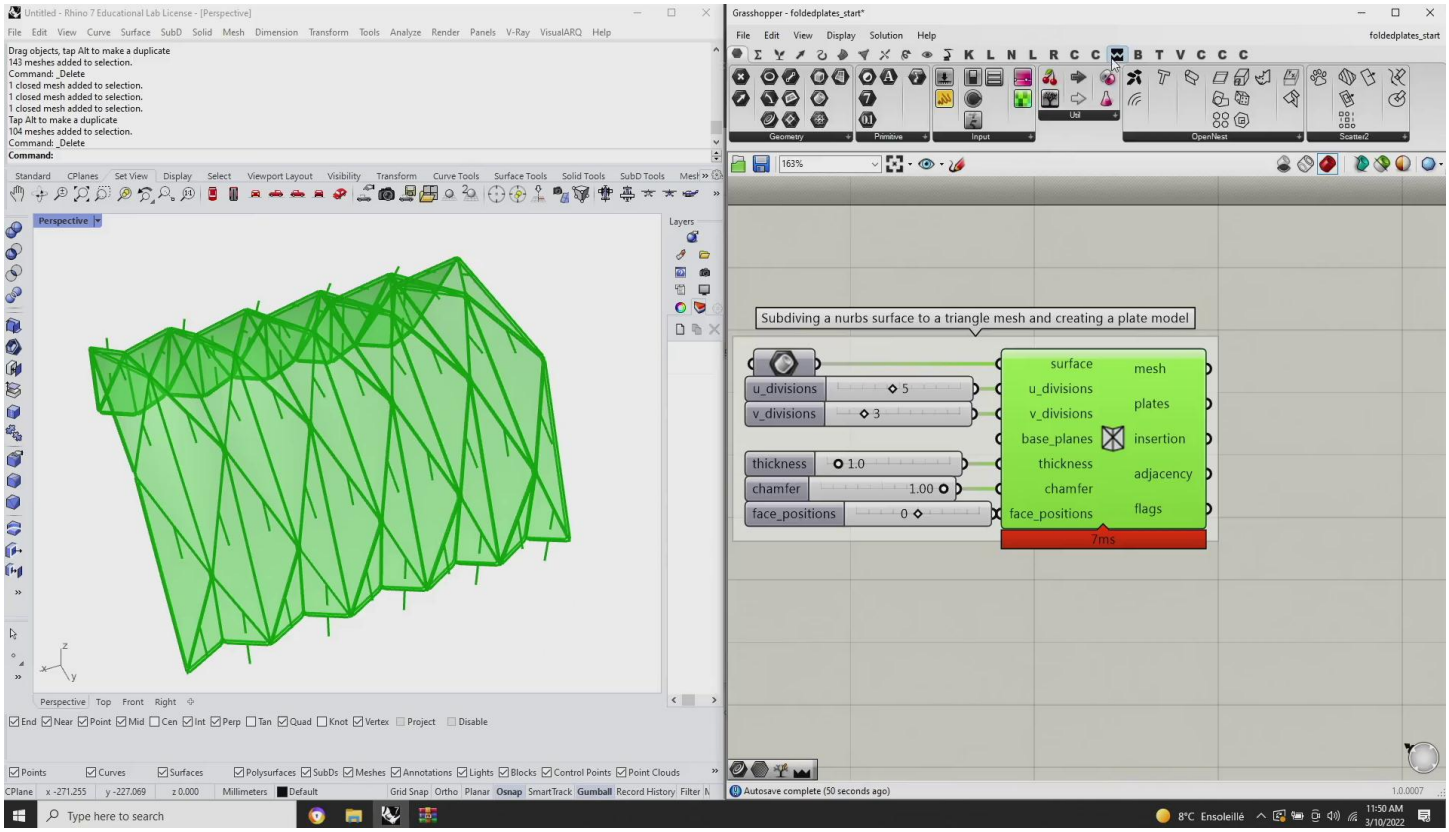


We can change the input type of the tile, which is written in the documentation what these types mean. There's a shift value, and then there is an edge division. So we have one group, second group, third group, fourth group, fifth and sixth group, depending on different data set you're using. This is it for the short data set example. And we are going to start the second example called folded-plates start. I will delete the rest of geometry to not interfere with anything. So, we have a cylinder that we'd like to convert to a triangular mesh. And we can use a component called case-free-folded-plates component. And you can see, after inputting the information, we can get the geometry. I'll explain a little bit the inputs. We have the U division, we have the V division, the folded plates work within certain limits, meaning if the angle is too open, the intersections might fail, but then the angle is sufficient enough, the intersections can perform very well. All the plate systems are restricted to the maximum angle they do work, meaning that angle has to be relatively big. Then there is the offset of the shell.

Notes

Summary

9m 38s

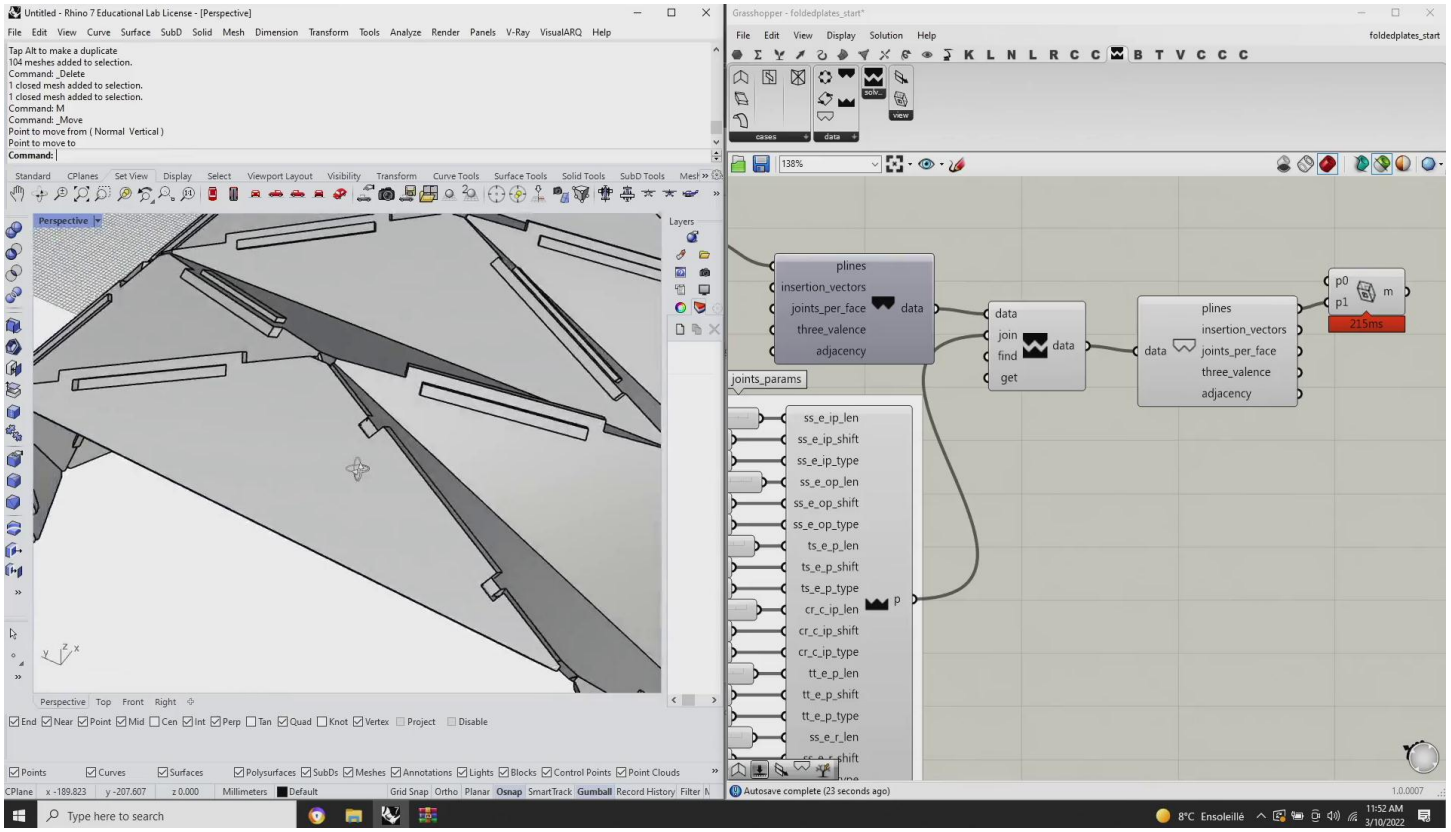


And then there is a chamfer, which is controlling the chamfer case of each corner, because we have this multi [inaudible 00:11:43] joints that cannot be really offset properly. And we have the face positions, which means how, what is the distance between the offset? By default, let's leave it to zero. Also will show the outputs we'll hide for the moment, this is a simple mesh, but we can, again, subdivide it more, subdivide it less and in one or in the other direction. And then we can have a series of plates, which are essentially series of border lines. Is this visible right now? So there's a thickness of those border lines and the chamfer value. Then there are insertion lines. So we will use the insertion sequence. We have series of insertion lines that define the one insertion back to the plate. Then there is adjacency showing which element is connected to which one. And for the various reasons, we have additional flags, which essentially gives the checkerboard pattern. This part is essentially subdividing the orb surface to a triangle mesh and creating a plate model. Now we need to somehow make the joints. Again, we, we have the component in the Compas Woods.

Notes

Summary

11m 33s

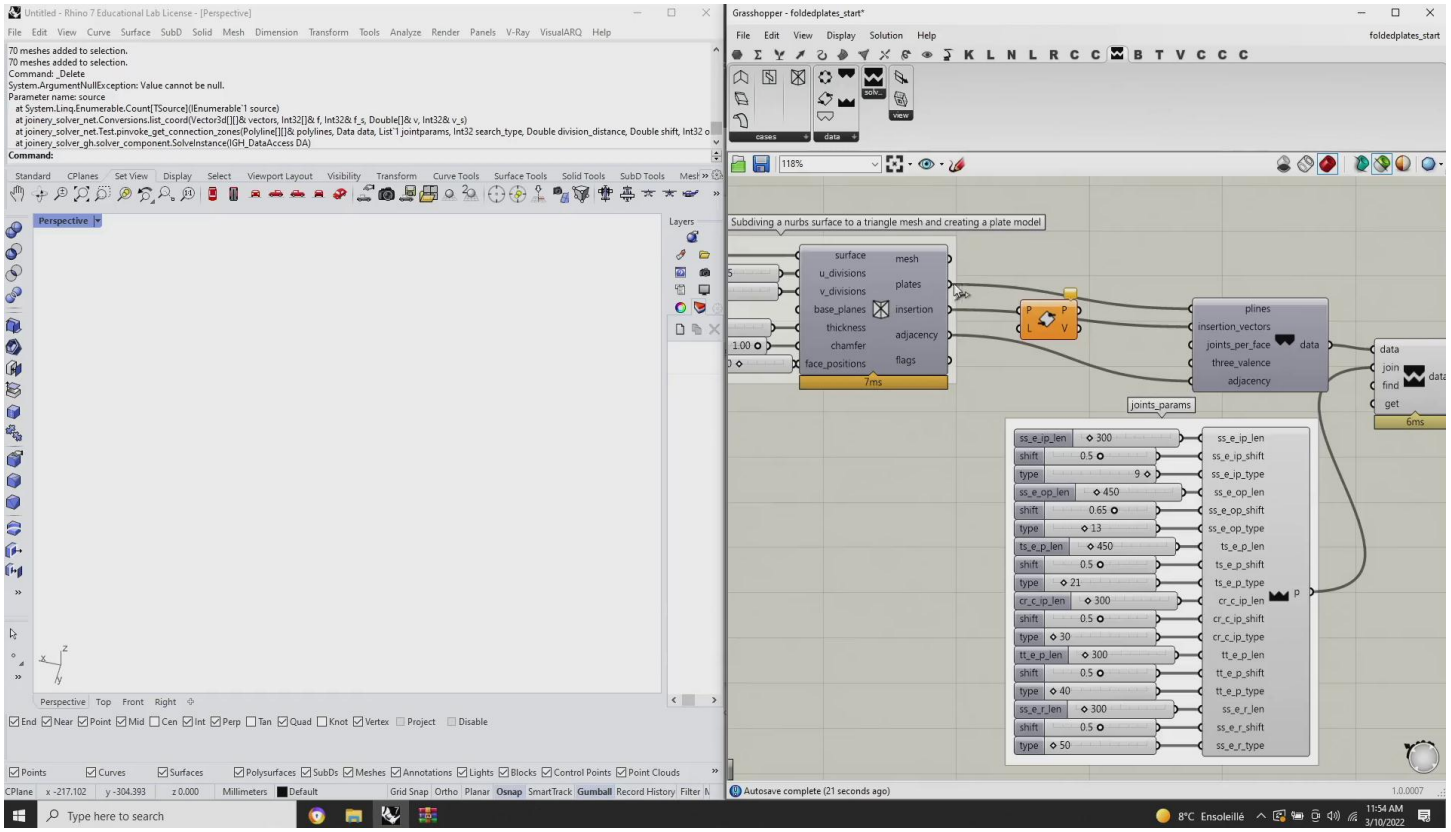


We use the solver to have the solving information. Then we need somehow to compute the joints. First we need the input set. We use plates as an input to the data set. And you you see, by default, something gets computed, might not be the proper geometry. And this is something you need to change, because in this case, we are going to to find out that we need a mitre joint that has a [inaudible 00:13:58]connection. To make this work, we need to have an input parameter for joints, so joint sets, and then we place parameters into a joint input, and we need to change the out of clean section from ten to, I believe, 13, yes, 13. And you see that this is a [inaudible 00:14:29] it's not super clear, because it's only wire-free mode, but we can output the data to our polylines and loft the elements. So if I would add it to the canvas, you can see there is a series of geometry. If you try to really disassemble this piece, you can see that it actually intersecting with the other joint. There is no wearily disassemble the structure or assemble it, after fabrication, that's why you need a set of insertional vectors to change one joint and you need to change another joint.

Notes

Summary

13m 25s

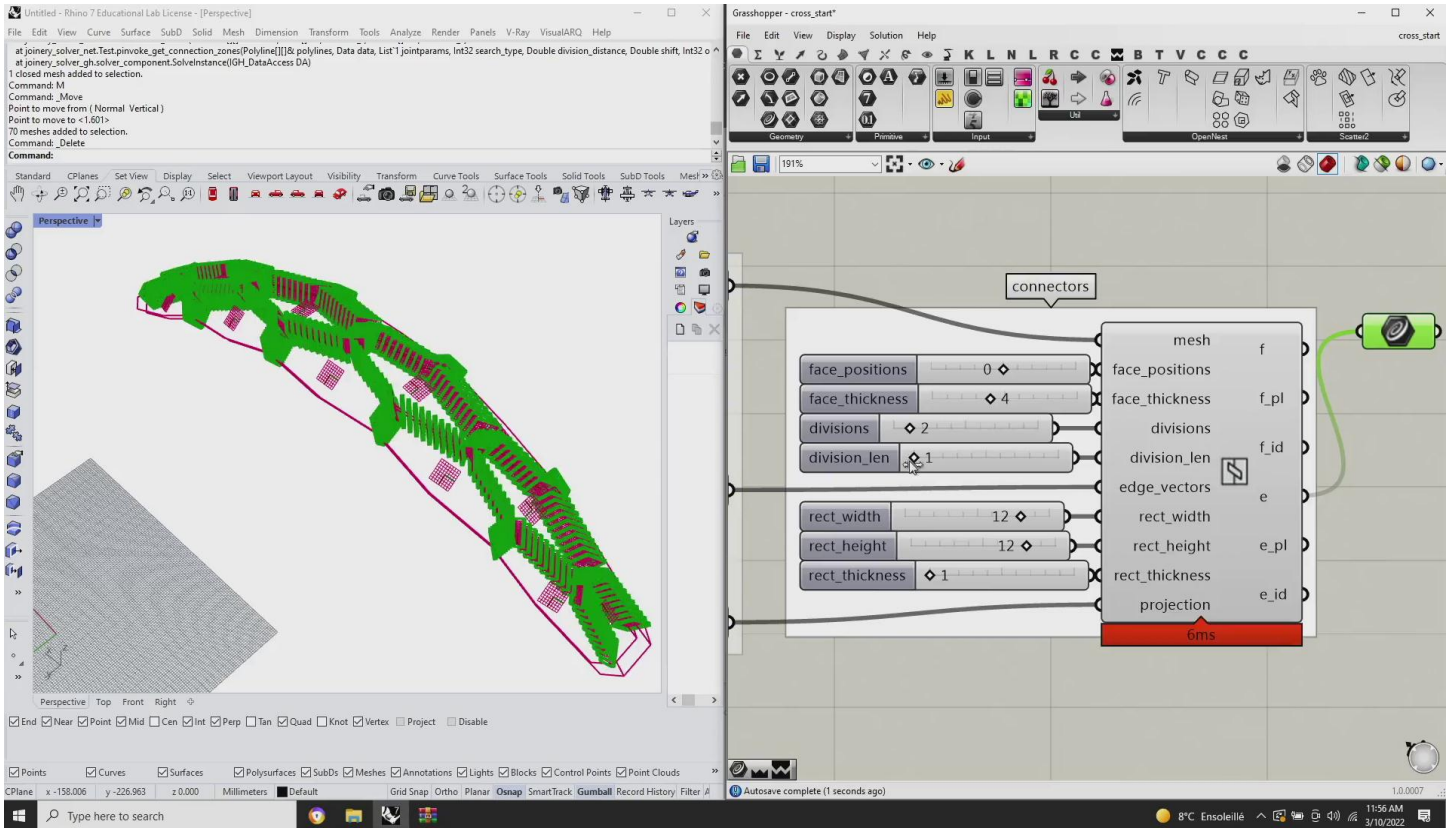


Luckily, in this component, we have the input parameter called insertion, and also the adjacency that says which element is controlled by which. For the first case, I will input adjacency to adjacency. And these [inaudible 00:15:38] joints, there is a very important point I would like to show, it's adjacency does do depends on the way we order the elements. And you don't know which element is female and which one is male. If you simply reverse the geometry, you can have the reverse part. And then in the opposite case, you can have the right part. So, this means that when you have the adjacency flags, these numbers tells you which one is a positive and which one is a negative part. In case we have non-symmetrical joint. And then the last part, we need insertion vectors, these are series of lines that we put to insertion lines, but these lines by default are seen as not working because they are not assigned to individual edge. And if you remember from previous tutorial, we need to do a special search procedure, which is called Closest Lines. We take a series of polylines, we take a series of lines, gradually we find the polylines, and gradually we find a series of lines.

Notes

Summary

15m 18s

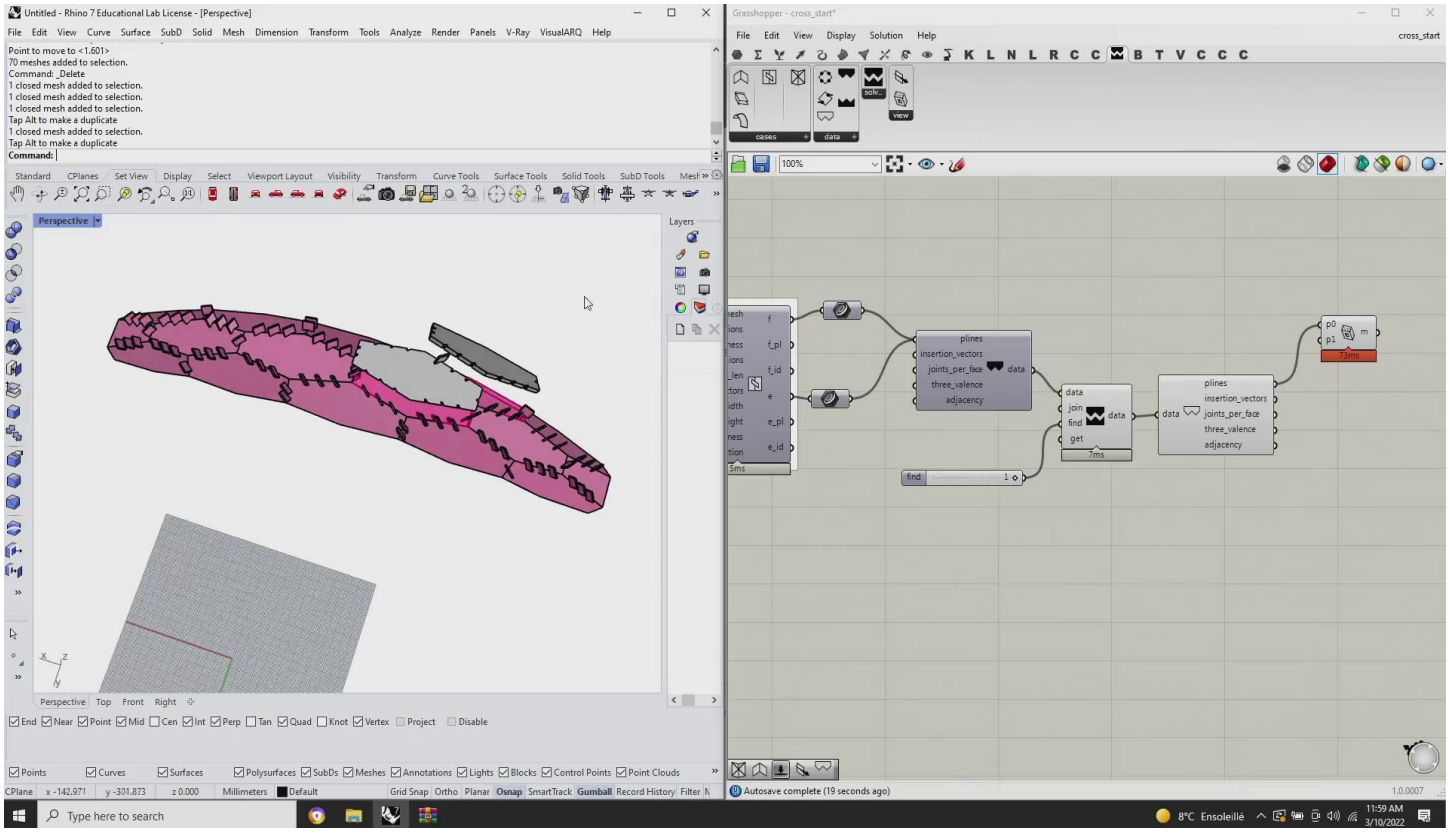


What you see right now is that those lines have been assigned to a series of polygons. Each edge has insertion direction. And afterwards you can have those joints that can be disassembled using one insertion vector. So this works clearly without any, any additional collision. And gradually we can disassemble the full structure plate, plate by plate. And now the last example that I'm going to show today is the Cross-Connection. If I would open the example Cross-Start, you would see that there are a series of inputs to use, again, a special component that helps you to define the structure. We have a series of insertion lines, we have a series of meshes, and then we have a boundary condition. So we can use a component called Connectors, just an example of this Cross-Connection. So we have the case where we input the mesh, where we input edge vectors and we input the projection part. I will hide not needed information, but you what you can see right now, you have a series of plates that were computed. Then you have also a series of connectors that were computed, then you can divide based on edge length. And if this is not given, or if it's given by zero, then you can divide by divisions.

Notes

Summary

16m 53s

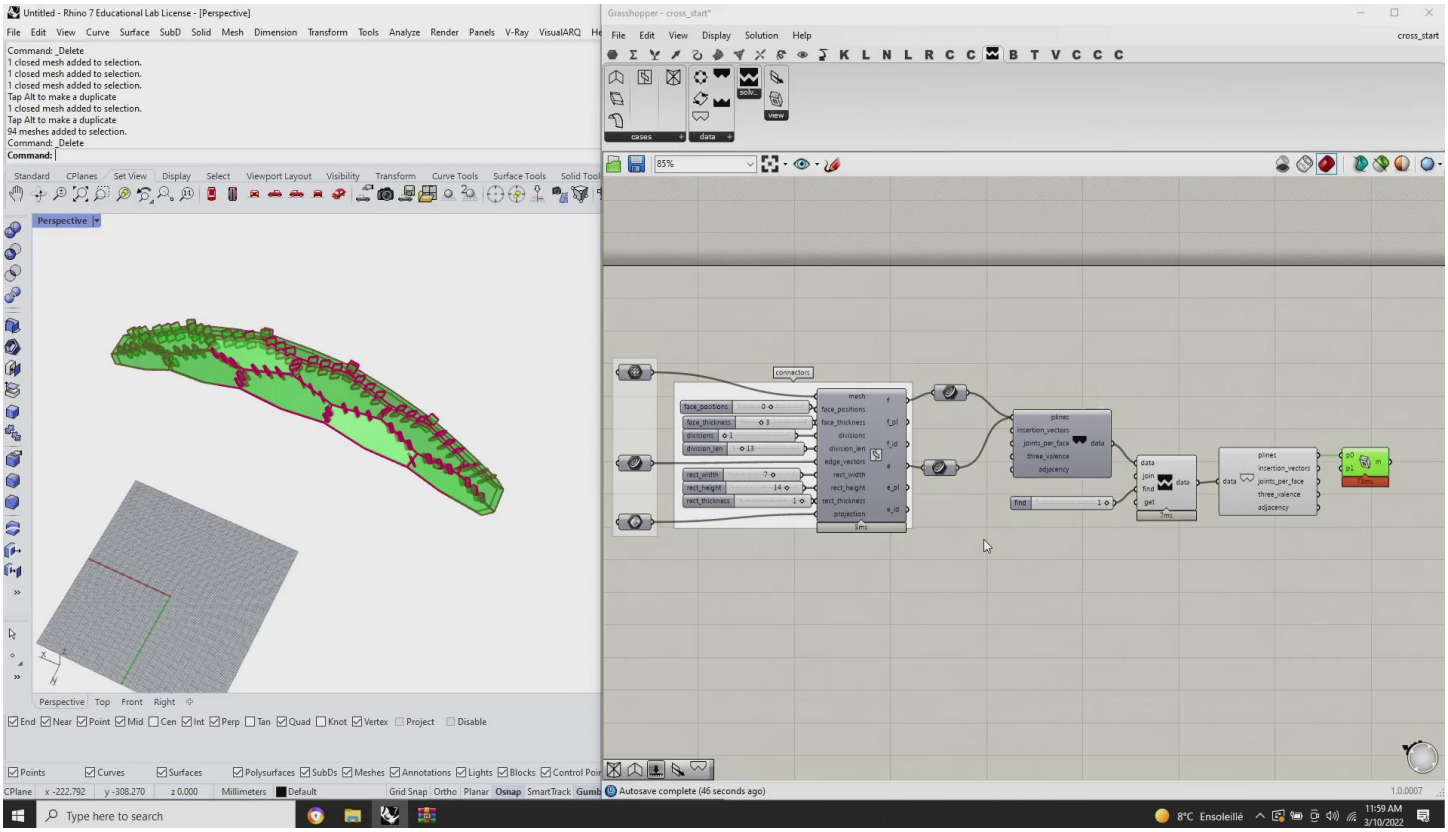


And afterwards you can also increase the edge offset. So if I would again go to faces only, you can change the face offset, then you can change the position of it because you might have multiple face positions. If you would like to construct a two layer shell, you can have both of them. And you see that connectors are also offset afterwards, depending on this edge distances. By default, I will use zero. So we have only one layer structure. This is the offset value that we can use to offset those elements and so on and so forth. When we have this information, we can also perform the intersection process, which we can use the the Compas Wood component, which is called the solver. And then we can also input the series of border lines, if you converted to data, and you see that intersection has been computed. For this we need to change the input variable from zero to two, or one. Let's take it to one. So we see that the joints were actually computed. And then we can output the information in order to loft the data. You see that the cuts between elements were done. And we can have this example finished. This summarises these very short, small, little examples.

Notes

Summary

19m 00s



You can also find the solutions in the example folder, similar or the same. These are several cases you can perform and this compost would plugin. We can also do something more as these are very basic, simple examples you can work with.

Notes

Summary

21m 38s