Sessions 14 & 15

Dynamo for Structural Design

Julien Benoit, Groupe Legendre, Julien.BENOIT@groupe-legendre.com
Håvard Vasshaug, Dark Architects, hvasshaug@gmail.com

Class Description

The visual programming interface of Dynamo is enabling structural engineers with the tools to build complex structures with minimal energy and make their own structural design tools. Based on the Revit Platform, tomorrow's structural designers can use their creativity to develop optimized structural systems using computational logic in an advanced building information modeling environment. This lab will discuss how to run analytical simulations in the Dynamo+Revit environment, by collecting data from analysis software, virtually manipulate and modify the analytical model in Dynamo, run iterations to optimize the design based on results. Participants will also learn how to create organic and computational structural forms and systems, create interactions between elements and collect structural data in order to increase productivity and avoid boring and repeating labor. We are passionate about empowering young engineers and architects with exceptional digital design tools, and firmly believe this to be the next generation for many of them.

About the Speakers

Julien is BIM manager for Legendre Group, a French general contractor based in Rennes. With an experience of 15 years in construction management on and off site, he started with Revit in late 2008. Formerly working for Bouygues Construction, Julien is now responsible for BIM implementation in Legendre's construction company, in charge of Revit integration, from Design team, technical department and cost estimation to construction sites and concrete precast plant. He is part of the new Rennes subway line team, in charge of structural coordination in Legendre's scope of work. Julien is also a moderator of RevitForum.org, pretty active on Twitter @Jbenoit44 and share knowledge and ideas on his blog.

Håvard Vasshaug is a structural Engineer (M.Sc.), BIM Manager, and Revit+Dynamo power user at Dark Architects; one of Norway's fastest growing architectural studios. He has vast experience providing Revit training, solutions, and seminars for architects and engineers over the past 8 years, and now uses this background to share knowledge of digital building design solutions.

He regularly speaks about technical workflows, digital innovation and human development at various national and international conferences and seminars, and receive wide acclaim for his talks and classes. He writes about BIM and visual scripting solutions on vasshaug.net, and administers the national Norwegian Revit forum.

Håvard has a passion for making technology work in human minds and on computers and have three means to do so: Building project development, technology research, and knowledge sharing. When he can do all that, he is a very happy camper.

Introduction

Dynamo is a visual programming interface that connects computational design to building information modeling (BIM). With Dynamo, users can create scripts that build, changes and moves building information in whatever way the user wants. It is free and open source.

Computational design with BIM through Dynamo creates some interesting opportunities for the building design industry.

First, Dynamo allows us to design organic and optimized buildings and structures faster than with traditional modeling tools, using computational methods. This is because we can create, associate and analyze multiple building parameters, and have them revise our designs automatically. We can iterate and evaluate multiple building design options with ease, and build structures based on natural and mathematical principles.

Second, visual programming in BIM offers us a way of expanding the boundaries of what actually can be accomplished in a BIM tool. We can access and edit building parameters more effectively than traditional hard coded tools allow. We can establish relationships between building element parameters, and modify these using almost any external data. We can move any information about a building or its surroundings through our BIM effortlessly, something that is normally reserved for those who are software savvy.

This opens the first door to a vision of building designers taking ownership of, and designing, their own design tools. Ever since the Personal Computer became mainstream, almost all building designers have been subject to what software developers have created for them. This is an opportunity for the building design industry to start getting actively involved in how its software works. We can create, and obtain a deep understanding of, our own design tools.

When we were introduced to the building industry as young engineers more than a decade ago, our design tasks included drawing, copying and offsetting lines, while trying to make sense of complex 2D blueprints. It was not only mind numbing, but also time consuming and inefficient. We

now focus all our energy on teaching young architects and engineers about the exceptional digital building design tools they can use. We try to help them to avoid the same experience, and show them how to create their own software.

Computation is going to be a big part of the future building design workflow for architects and engineers. Dynamo, right now, manifests that vision.

Acknowledgements

We wish to thank Zach Kron of Autodesk for lots (too much really) of valuable tips and knowledge on the exercice and math in this class. Without his ideas and feedback we'd be having a much less wonderful experience.

Also, hat tips to the Bad Monkey Group.

Note

All information in this class handout is based on the following software versions: Revit 2015 Build: 20140905_0730(x64) Update Release 4 and Dynamo 0.7.2.2114.

If any of our examples deviate from your experience, please run a check on the versions you are using.

Table of Contents

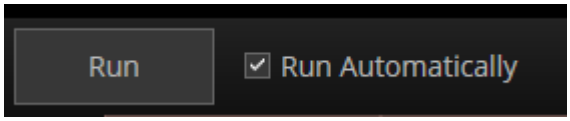
Table of Contents 5
Part A 6
Basics 6
Mathematics 12
Mathematics illustrated 14
Surface 17
Integration with Revit Elements 19
Part B 31
Get Analytical Model from Adaptive Components 31
Create Line loads on members 38
Create Boundary Conditions 43
Extra: Create Point Load custom node 48

Part A

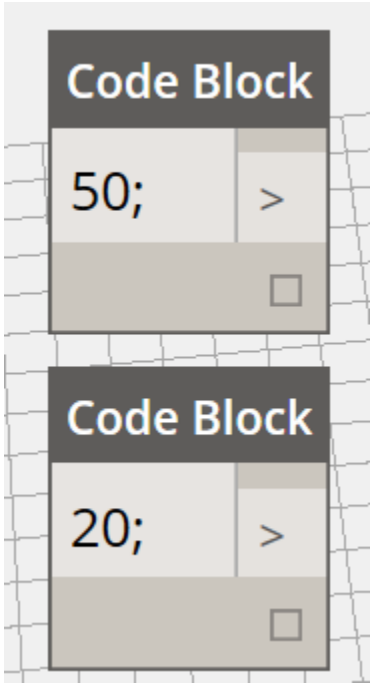
In this part of the lab we will model a double-curved, mathematically defined roof structure base on Dynamo and Adaptive Components in Revit.

Basics

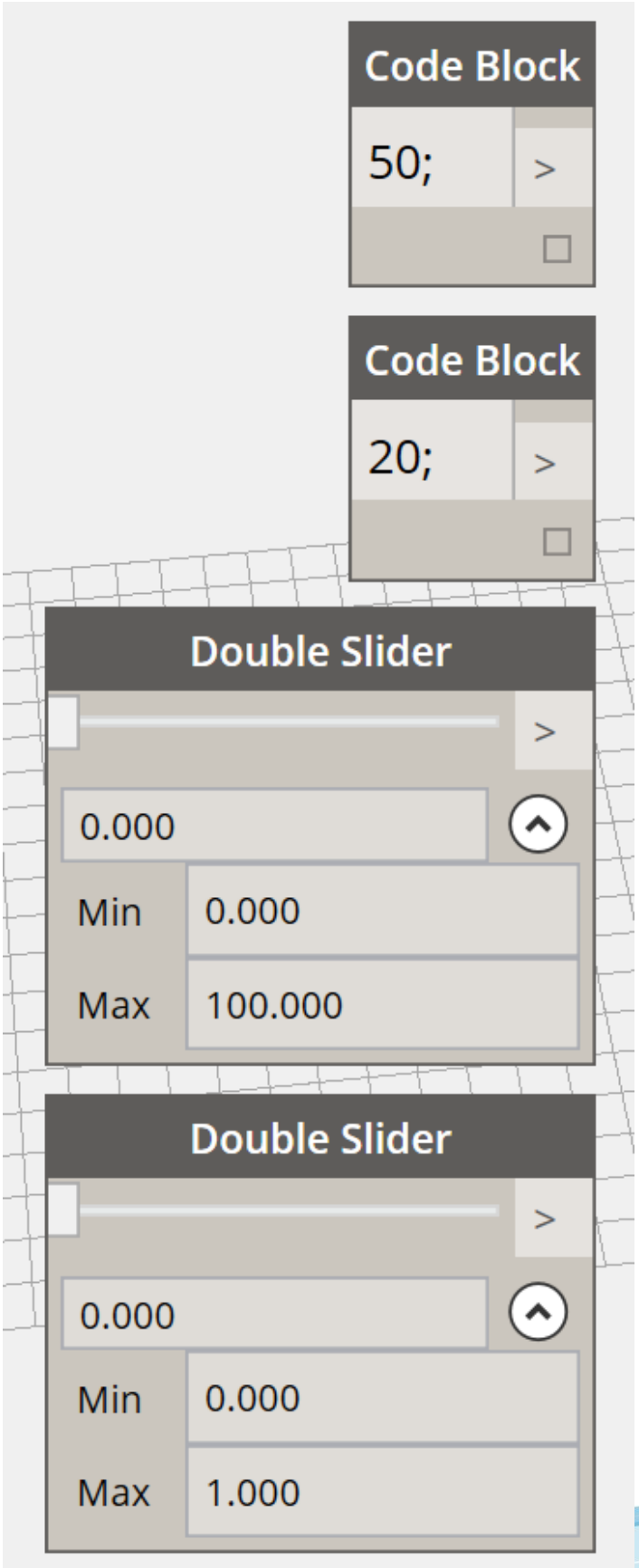
1. Start Revit 2015 and Dynamo 0.7.1.
2. Turn on Run Automatically in Dynamo.



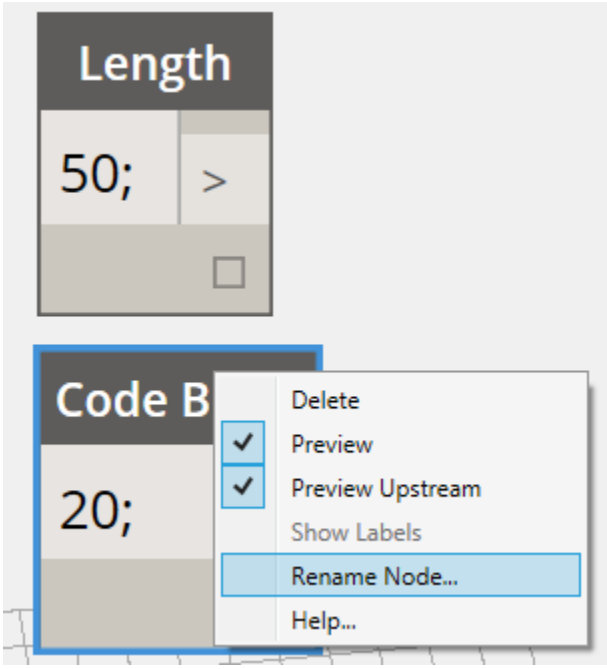
3. Double click in canvas to produce two Code Blocks. Enter values
   50;
   20;



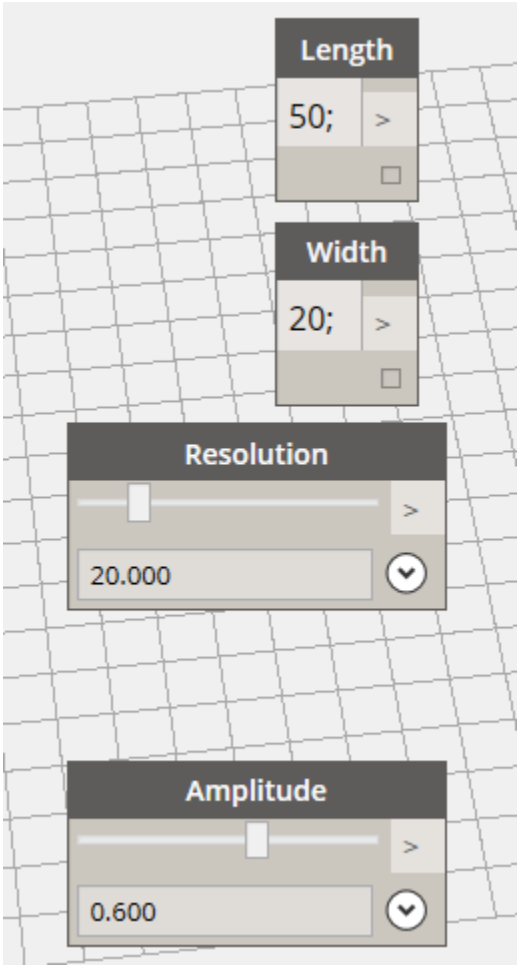
4. Search for Double Sliders in Library Search, and produce two nodes with Min and Max values 0 and 100/1 respectively.


Code Block
50;
>
Code Block
20;
>
Double Slider
>
0.000
Min
0.000
Max
100.000
Double Slider
>
0.000
Min
0.000
Max
1.000

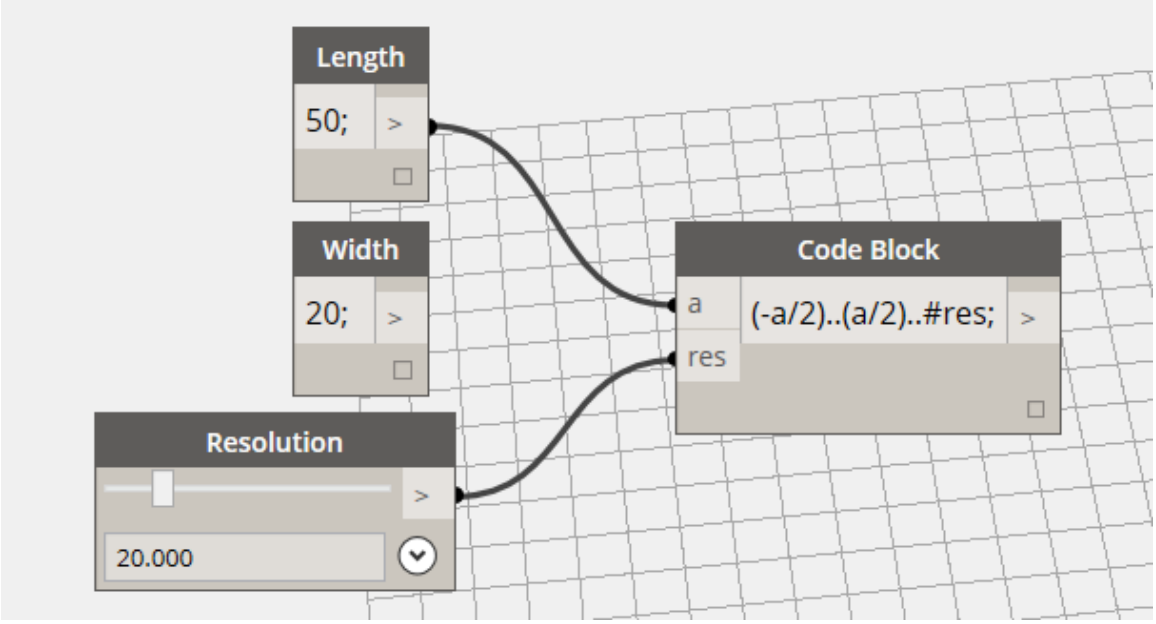
5. Right Click on all 4 nodes and change their names to
   Length
   Width
   Resolution
   Amplitude



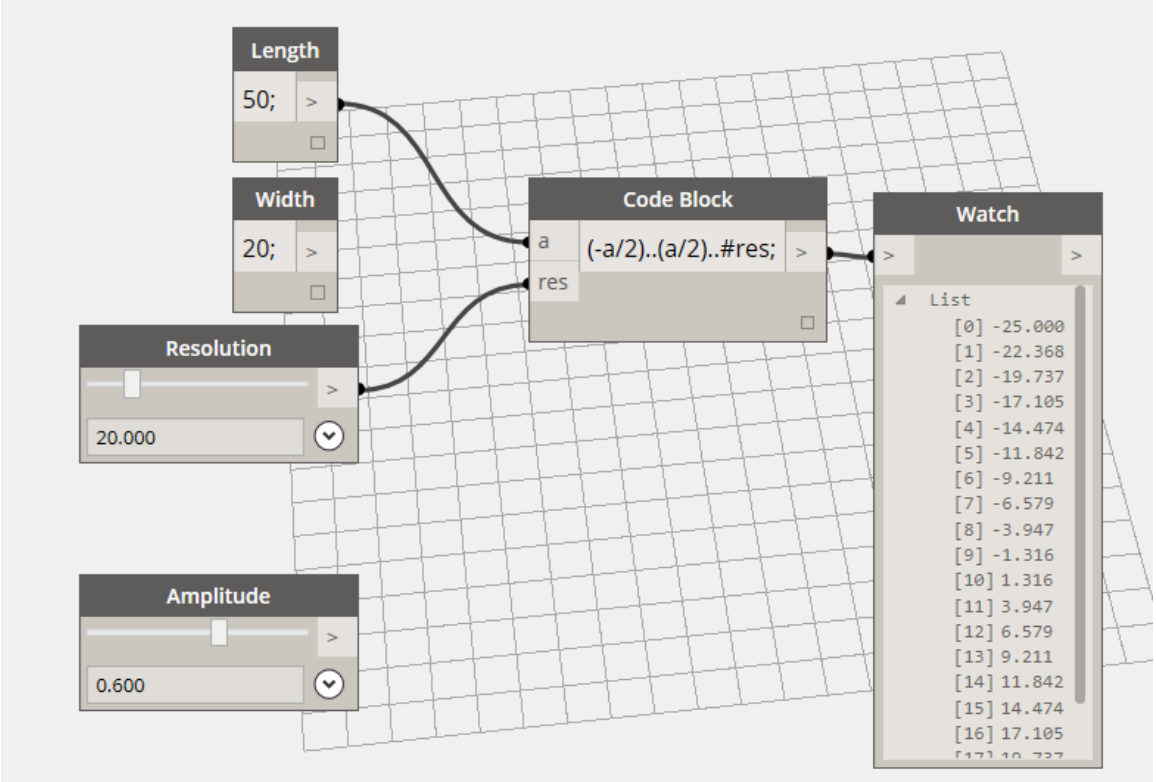
6. Change values of Resolution and Amplitude to
   20
   0.6



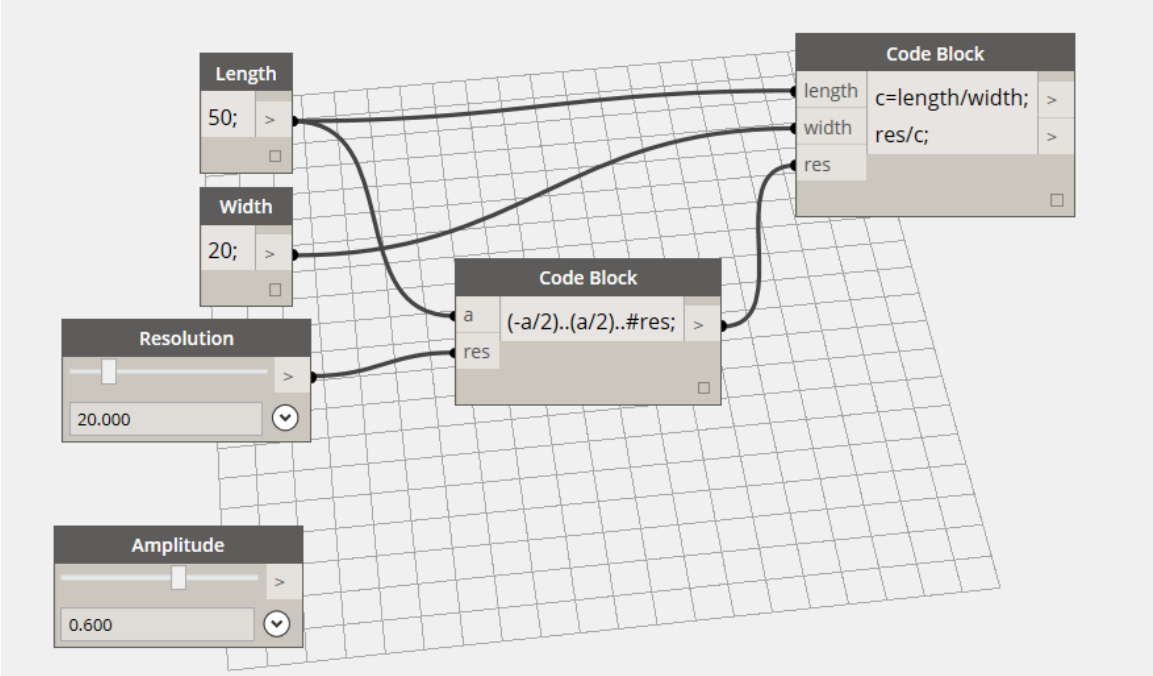
7. Make a Code Block with the syntax
   (-a/2)..(a/2)..#res;
   and wire it to Length and Resolution.



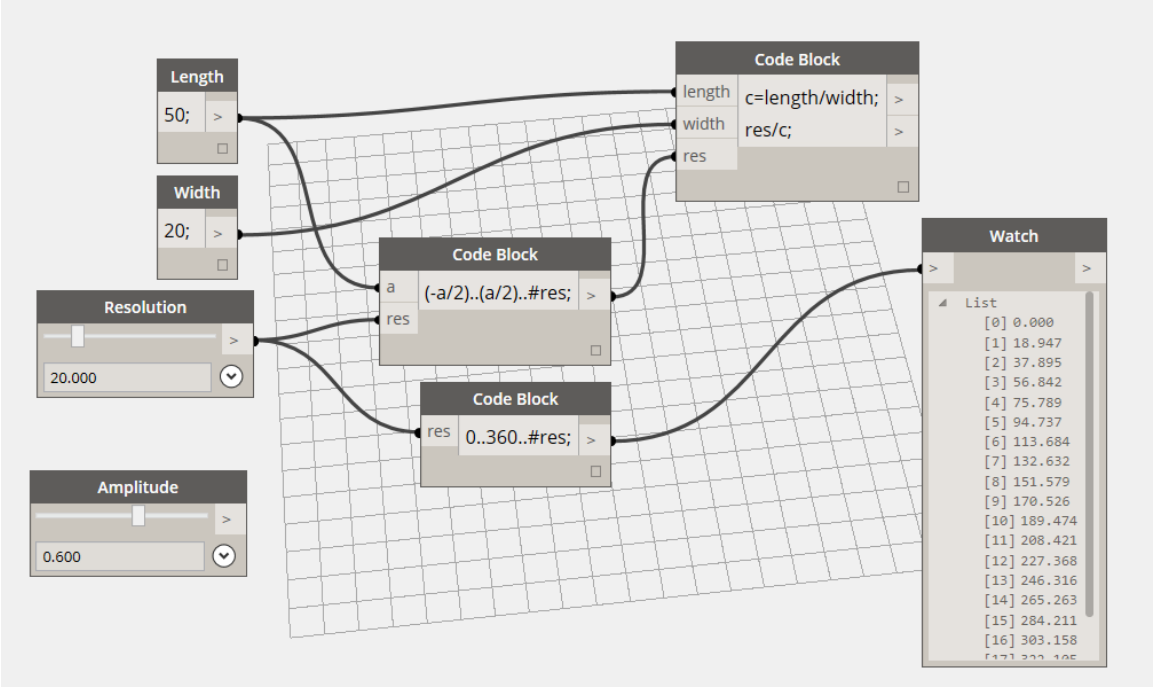
8. Make a Watch node and wire it to the Code Block output.



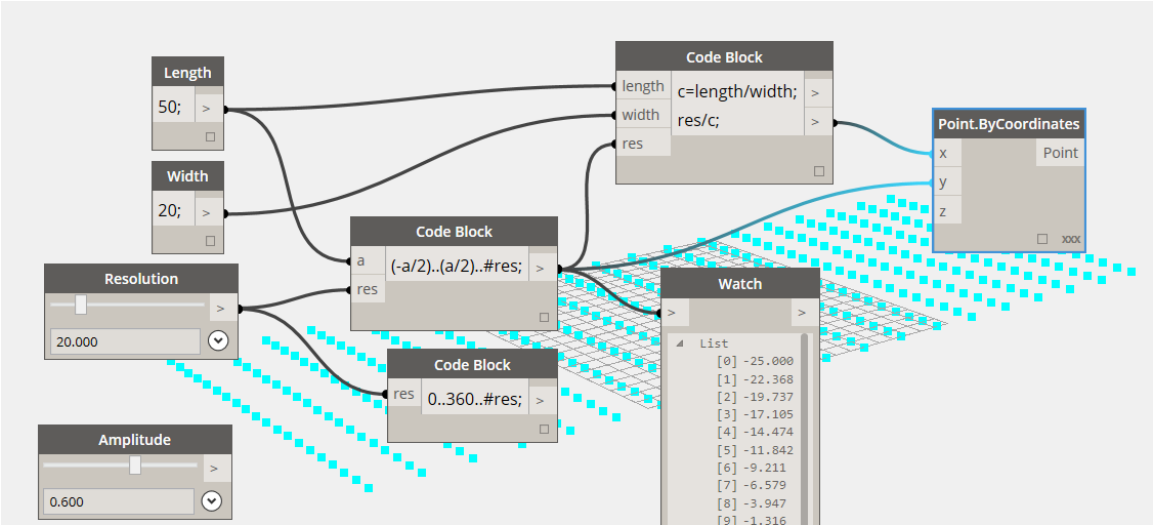
9. Make another Code Block with the syntax
   c=length/width;
   res/c;
10. Wire it to Length, Width and the output of the previous Code Block.



11. Make a Code Block with syntax
    0..360..#res;
    and wire it to Resolution



12. Produce a Point.ByCoordinates node, and wire it to the two Code Blocks like below. (Press Geom or Ctrl+G in the bottom right corner to navigate the background.)

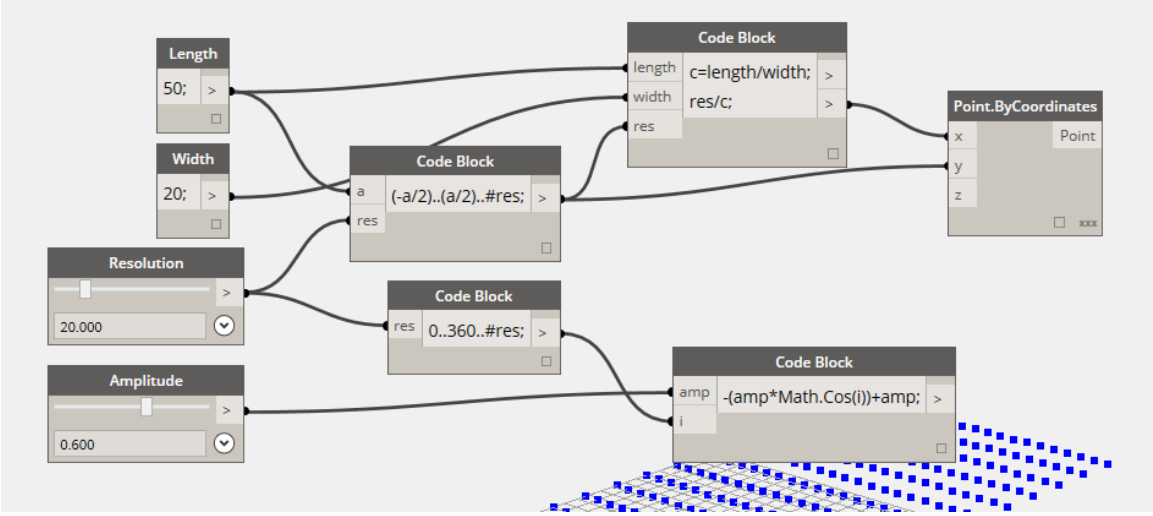


Mathematics

1. Make a new Code Block with the syntax -(amp*Math.Cos(i))+amp;

This math produces a trigonometric function that we will use as basis for our geometry.

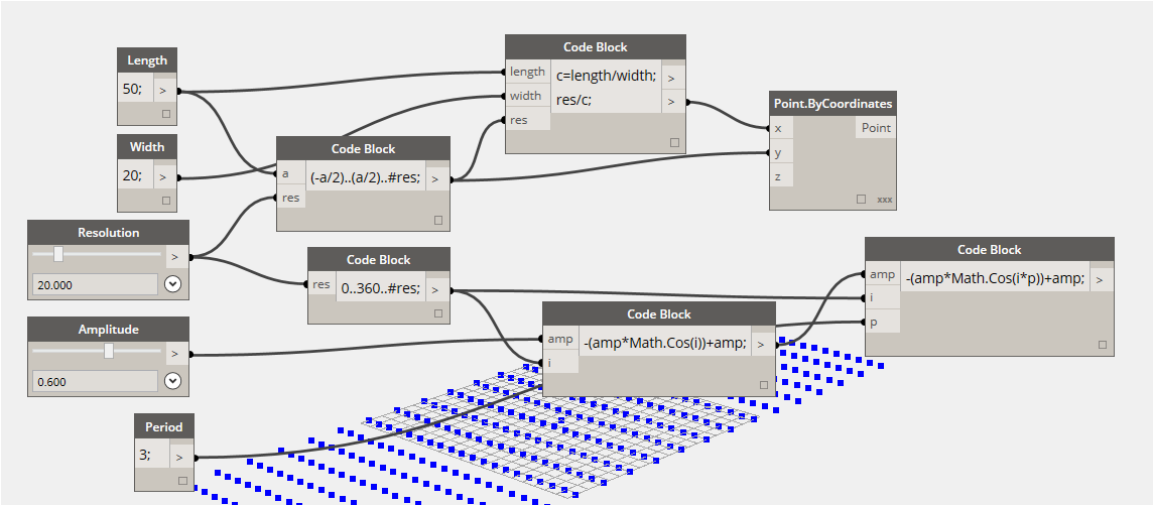
2. Wire the new Code Block like below.



3. Make another Code Block with the same math as the previous (ctrl+c & v), only change i to i*p:
-(amp*Math.Cos(i*p))+amp;

4. Make a new Code Block; change its name to Period and value to 3.

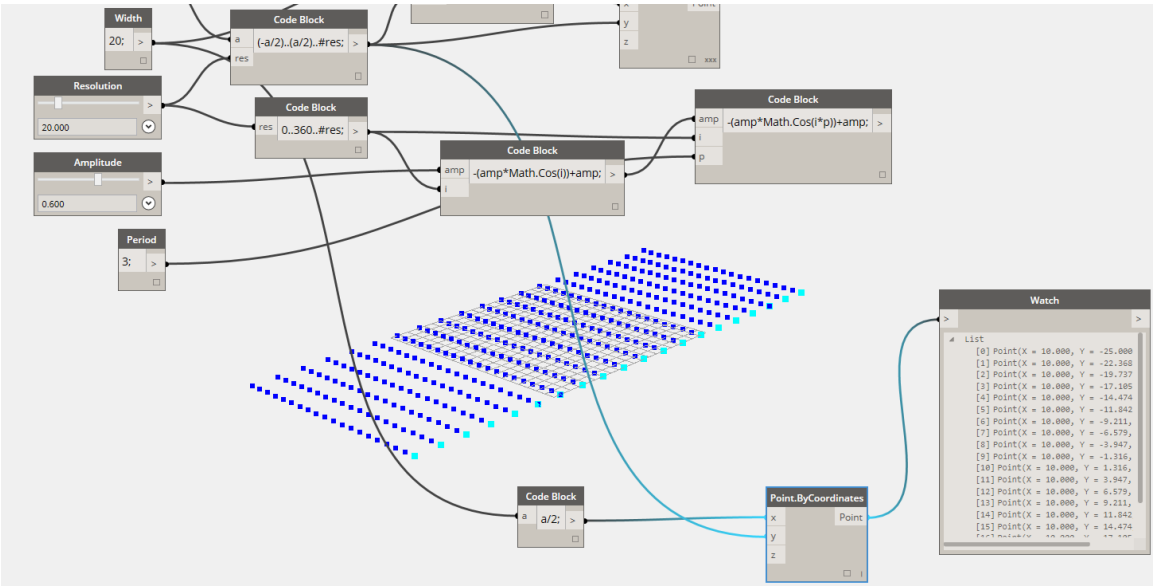
5. Wire like below.



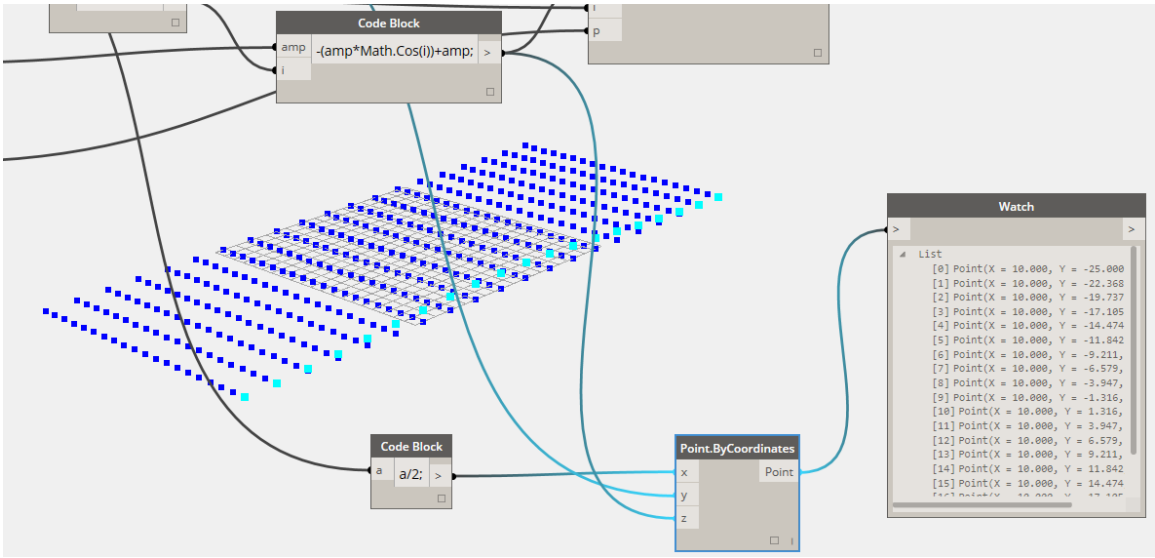
Mathematics illustrated

Let's illustrate what's going on here before we proceed.

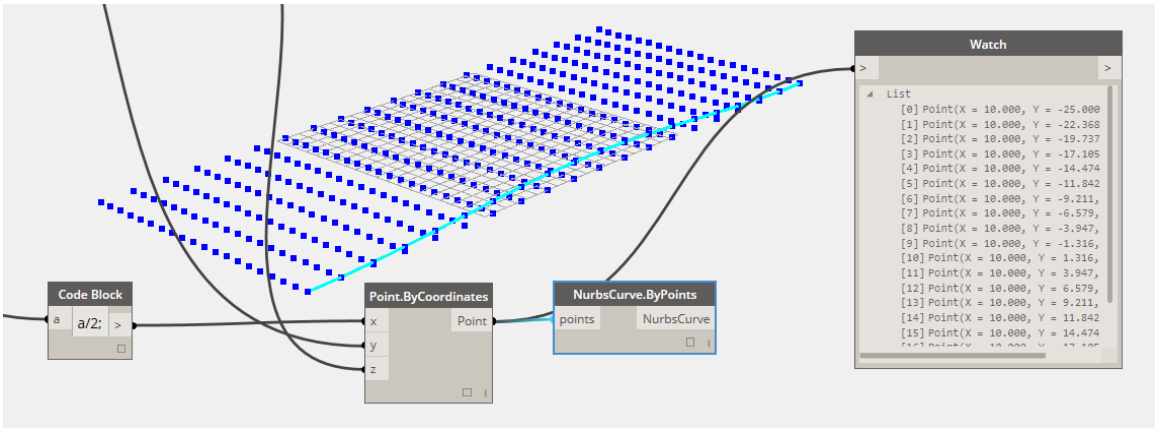
1. Add a Point.ByCoordinates node.
2. Add a Code Block with syntax
   a/2;
3. Wire Width into the new Code Block, and the resolution output to y, like so:



4. Wire the *first* cosine function to z.

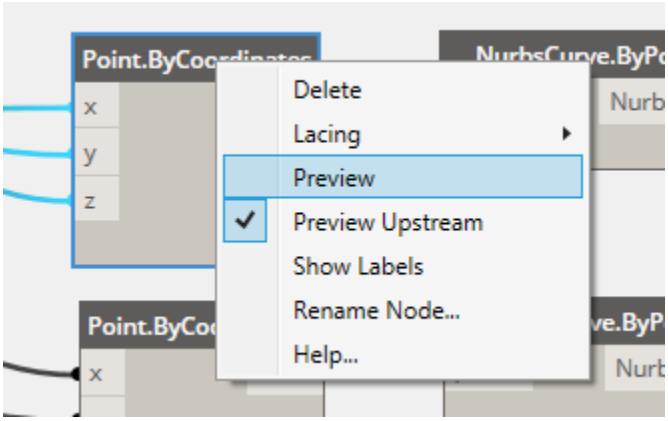


5. Add a NurbsCurve.ByPoints node, and wire it to the point output.

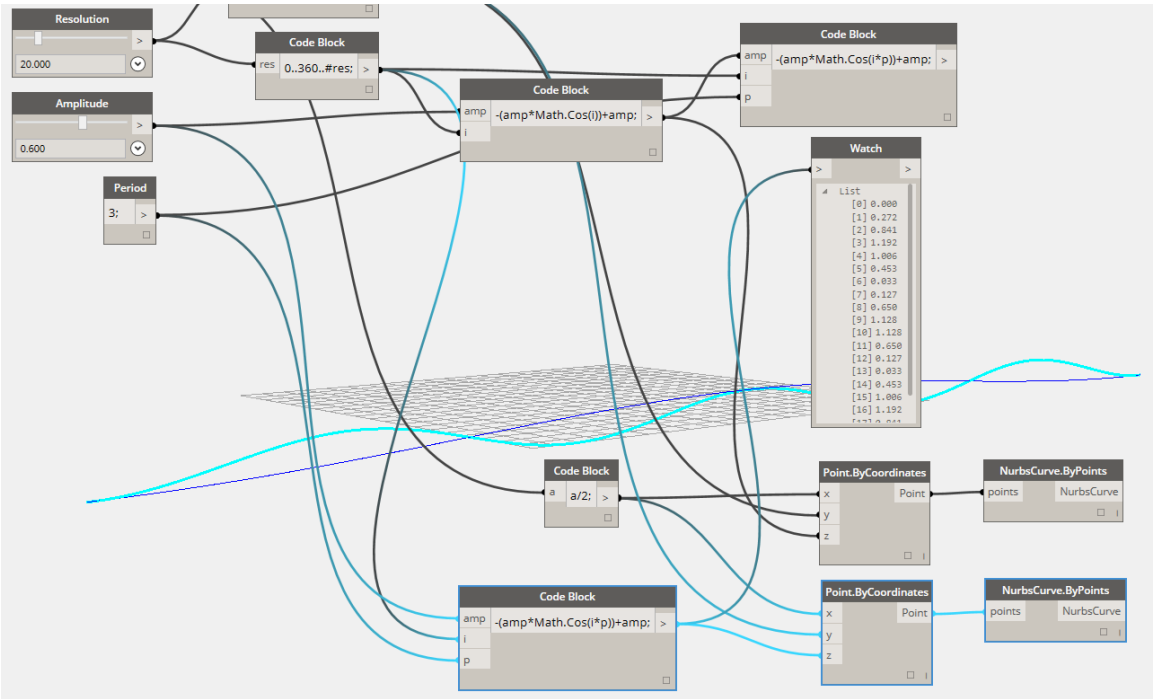


What we see now is a beautiful cosine graph between 0 and 360, divided at a resolution of 20. This is the outcome of the first trigonometry function.

6. Copy the point and nurb nodes.
7. Right Click on *all* point nodes and deselect Preview.

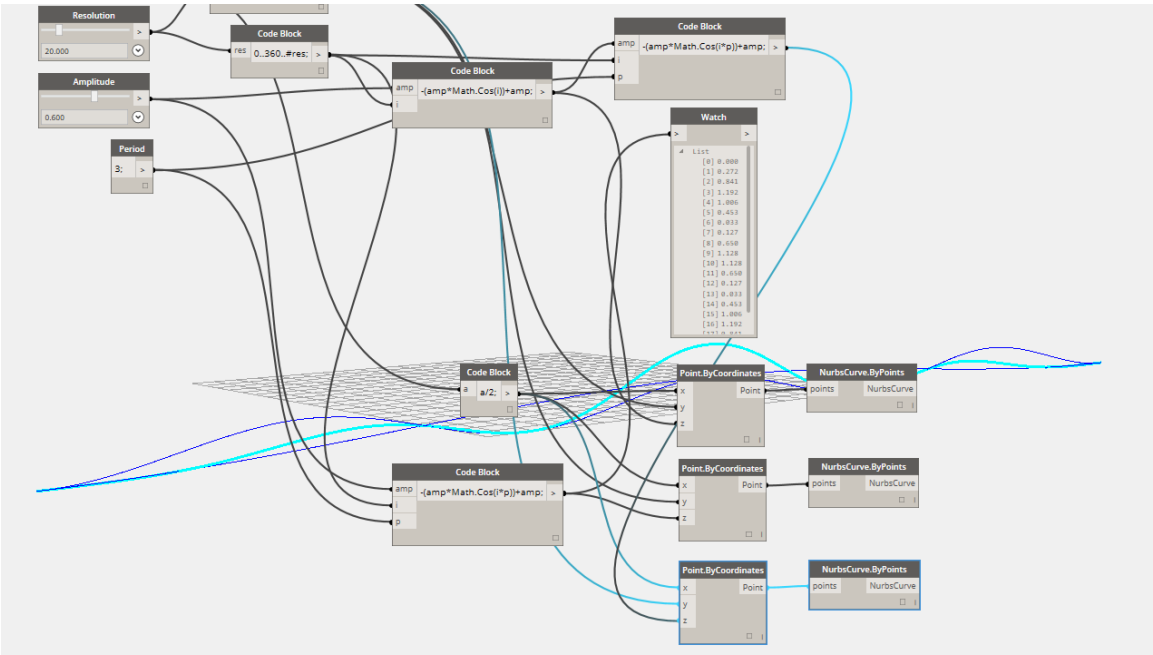


8. Copy the *last* trig function node from above, and wire it like below.



This is the same function as above, only with 3 periods inside the same resolution (20 between 0 and 360).

9. Copy a third point and nurbs nodes, only this we wire to the *last* trigonometry function.

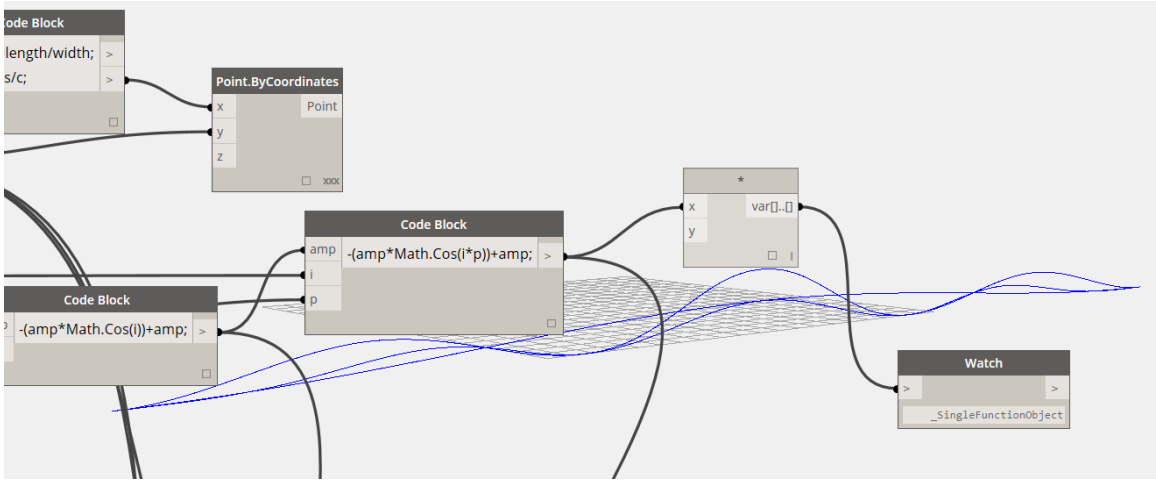


Here we see the two cosine functions (with one and three periods) combined.

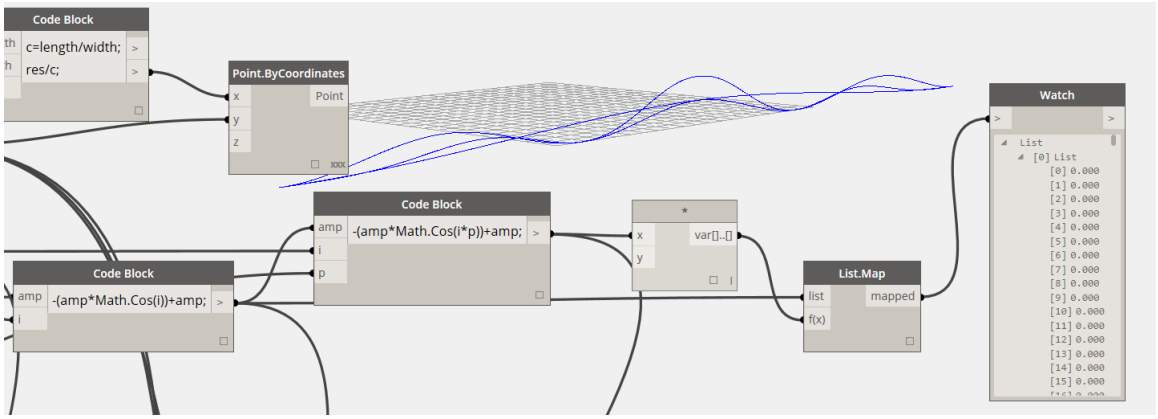
Surface

Let's turn this beautiful math into a surface.

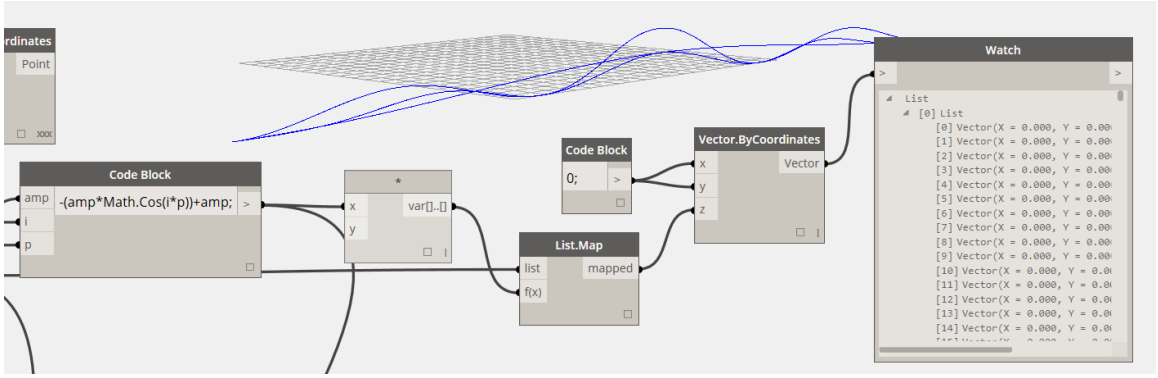
1. First, produce a * node and wire its x input to the last trigonometry function.



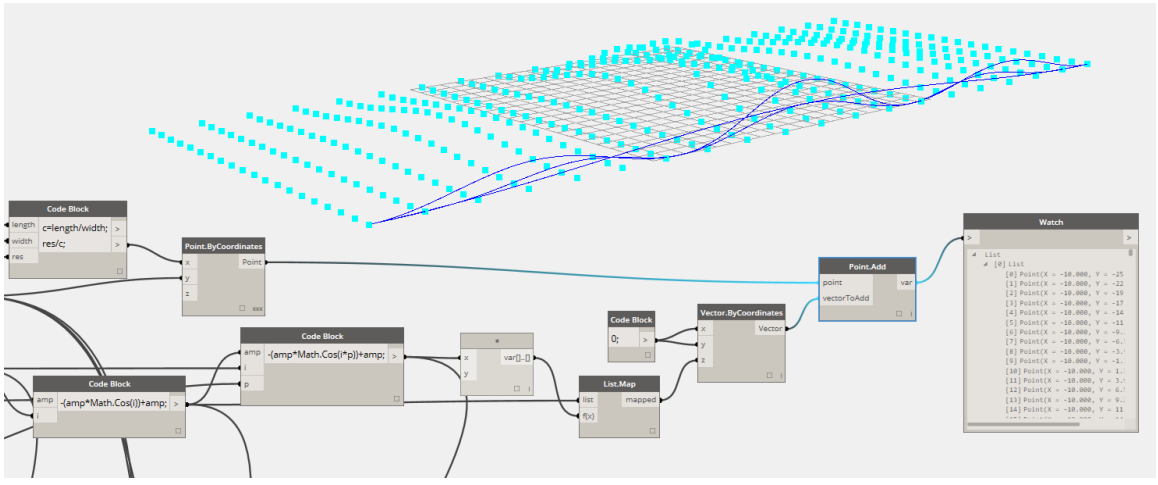
2. Use a List.Map to connect the first and second (last) trig functions in a new list of numbers.



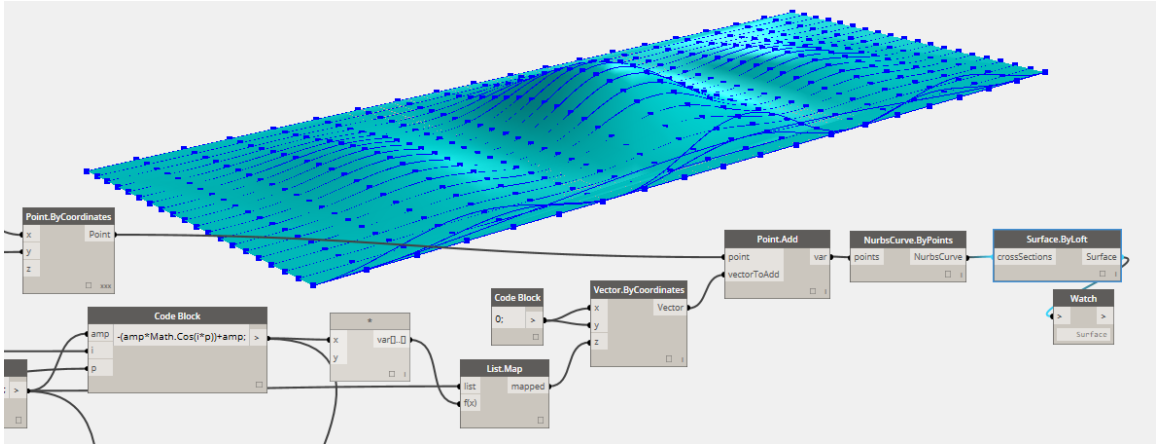
3. Turn this list into a list of vectors by producing a Vector.ByCoordinates node. Add a 0 input to its x and y input, and the List.Map to its z input.



4. Produce a Point.Add node, and wire it to the points and vectors like below.



5. Last, add a NurbsCurve.ByPoints and Surface.ByLoft.



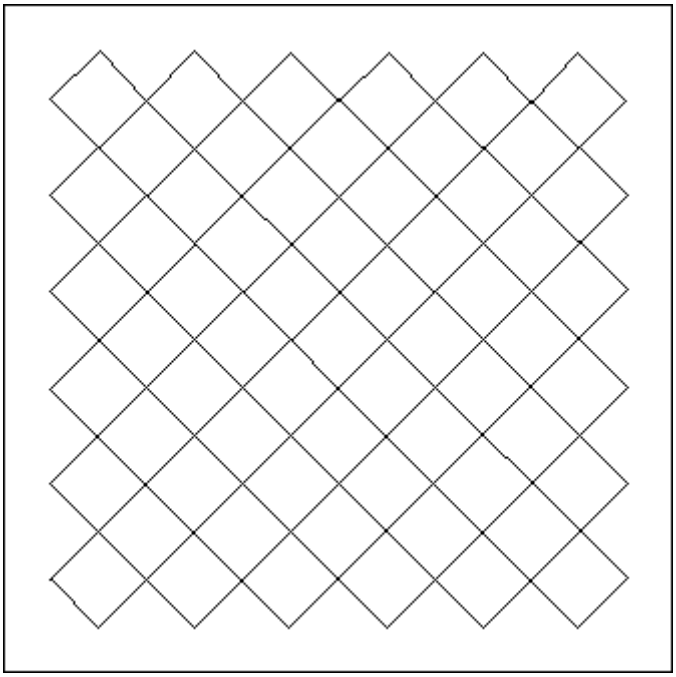
Integration with Revit Elements

One major experienced difference between dealing with Dynamo geometry and Revit Elements is that viewing, changing and interacting with Dynamo geometry is superfast. The same cannot always be said about Revit geometry. Still, one of the great advantages with Dynamo is that it actually *can* interact with Revit Elements. Let's have a look at how that works.

Adaptive Components

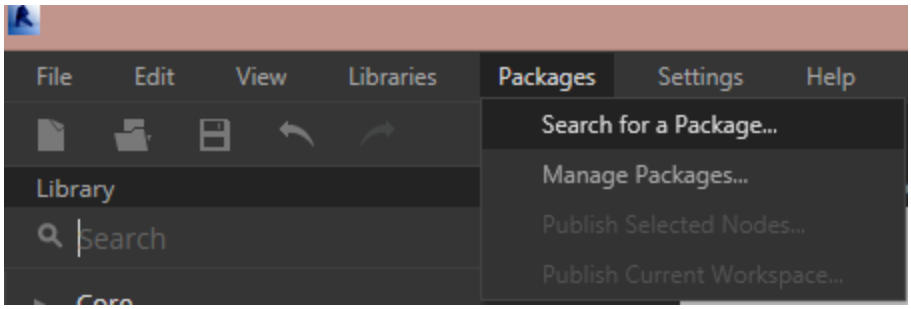
Integration with Revit Adaptive Components requires a set of placement points that correspond to an adaptive component family's adaptive points. In our example, the number of adaptive components, and their respective number of points, depends on the grid we choose to work with.

Let's make a diamond grid.

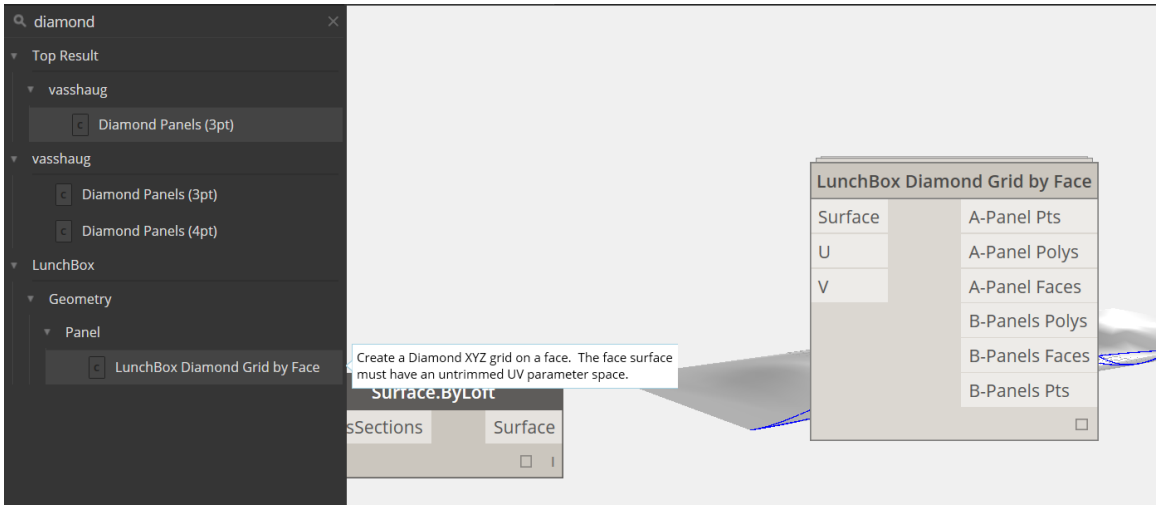


A diamond grid has 3-point panels along the edges and 4-point panels on the inside. To create these panels we need a custom node called LunchBox Diamond Grid by Face by Nathan Miller.

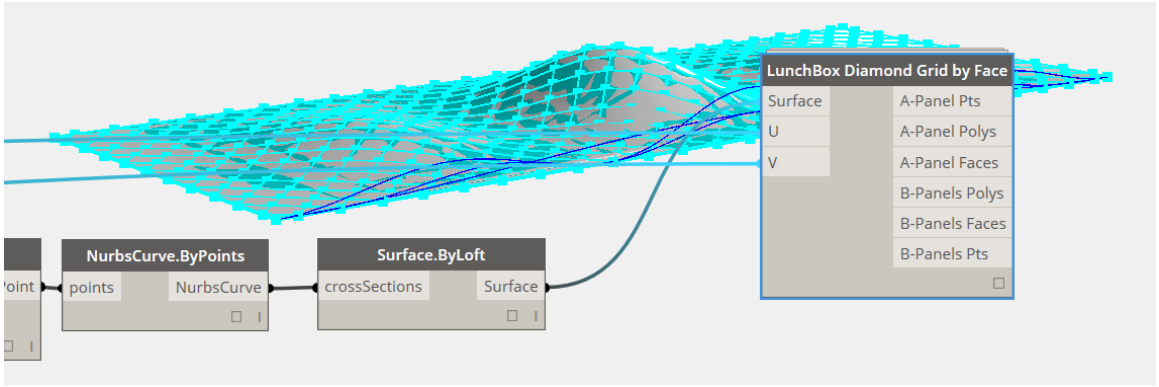
6. Open the drop-down menu Packages – Search for a Package.

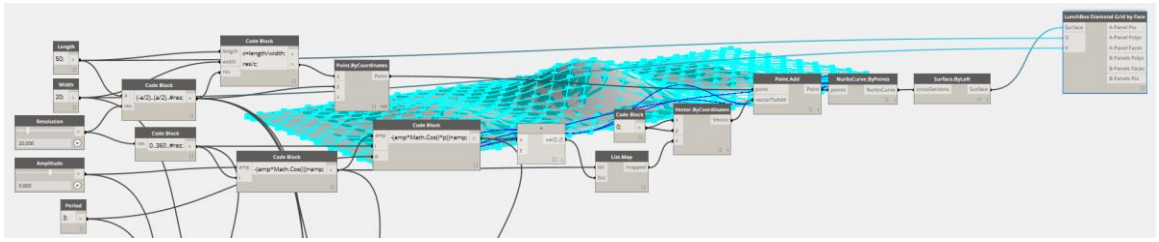


7. Search for "LunchBox", and click Install.
8. In our Dynamo search field, use the same keyword and add the custom node Diamond Grid by Face to our canvas.



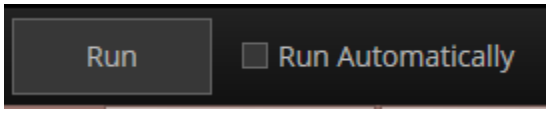
9. Wire the diamond node to the surface, Length (U) and Width (V) outputs.



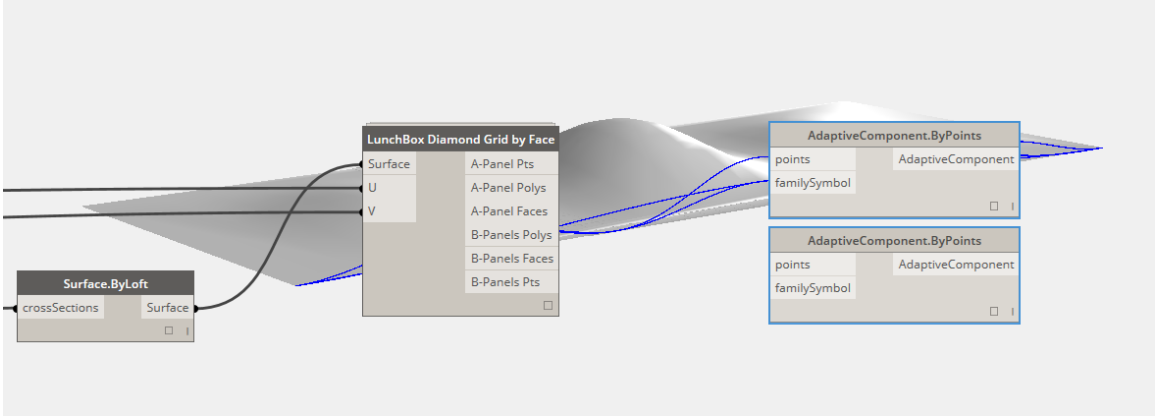


10. Now right click all nodes that generate point or line geometry in the preview background, including the panel nodes, and deselect Preview.

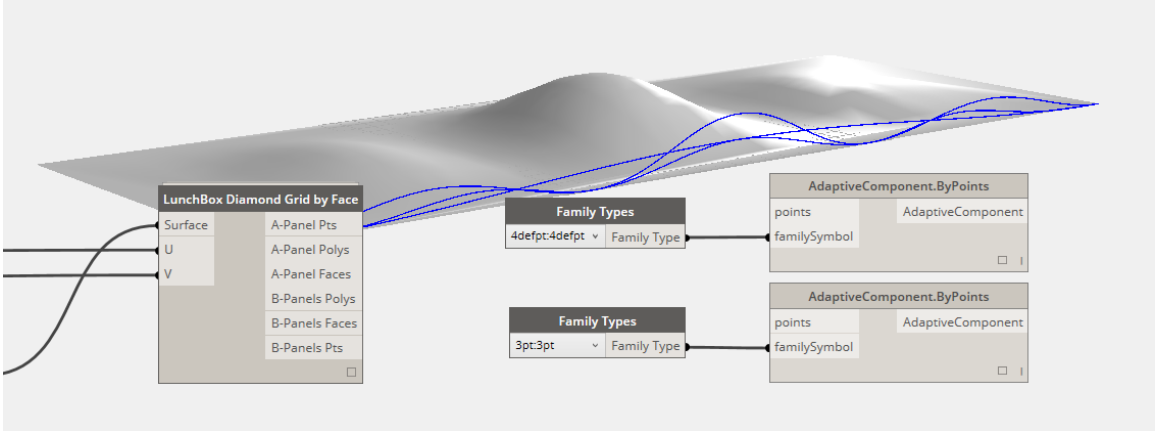
11. De-select Run Automatically.



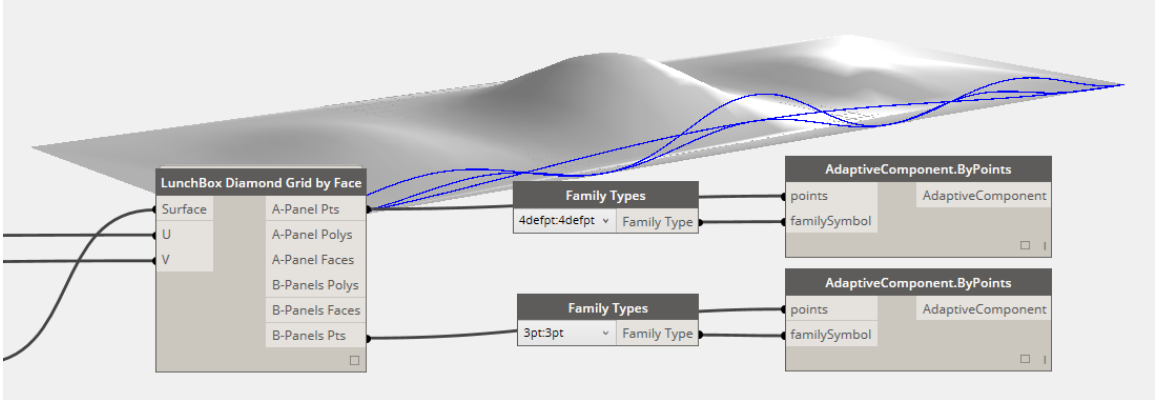
12. Add two AdaptiveComponent.ByPoints nodes.



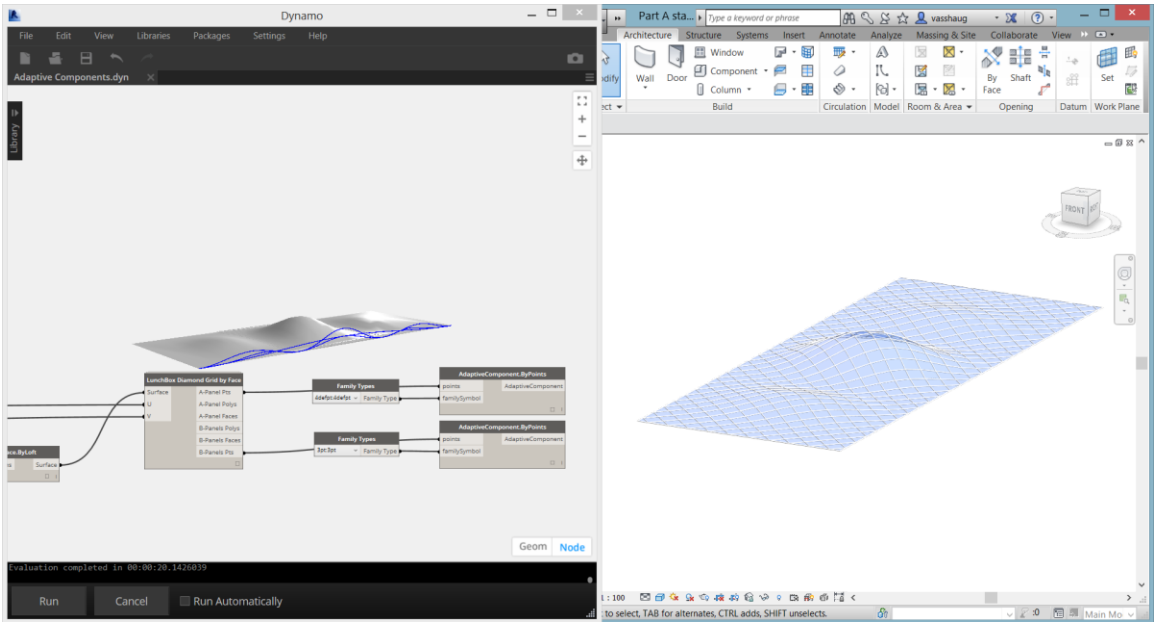
13. Add two Family Types nodes. Select the families 3pt and 4ptdef, and wire each to its own Adaptive Component node.



14. Wire the A-Panel Pts output to the points input of the Adaptive Component node that places the family 4ptdef, and B-Panel Pts to the other.



15. Now run the definition. Your screen should look like this:



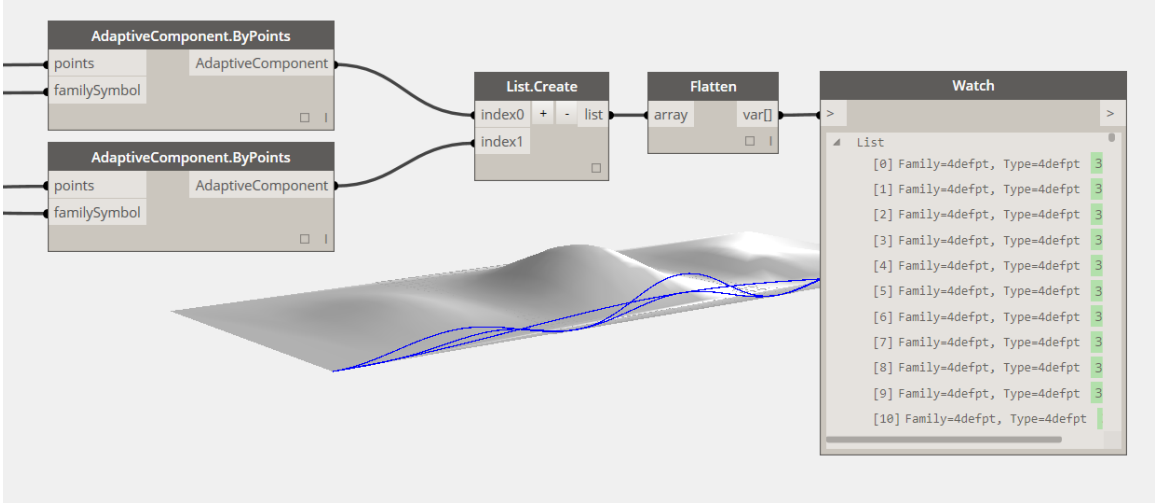
Congratulations! You've made a perfectly mathematically defined double curved cosine surface of native Revit elements using Dynamo. There are a couple of simple operations we can utilize on this model to get some ideas on shapes, sizes and constructability (cost). One of them is colorizing ranges of parameters in a view.

Simple Colorized Design Analytics

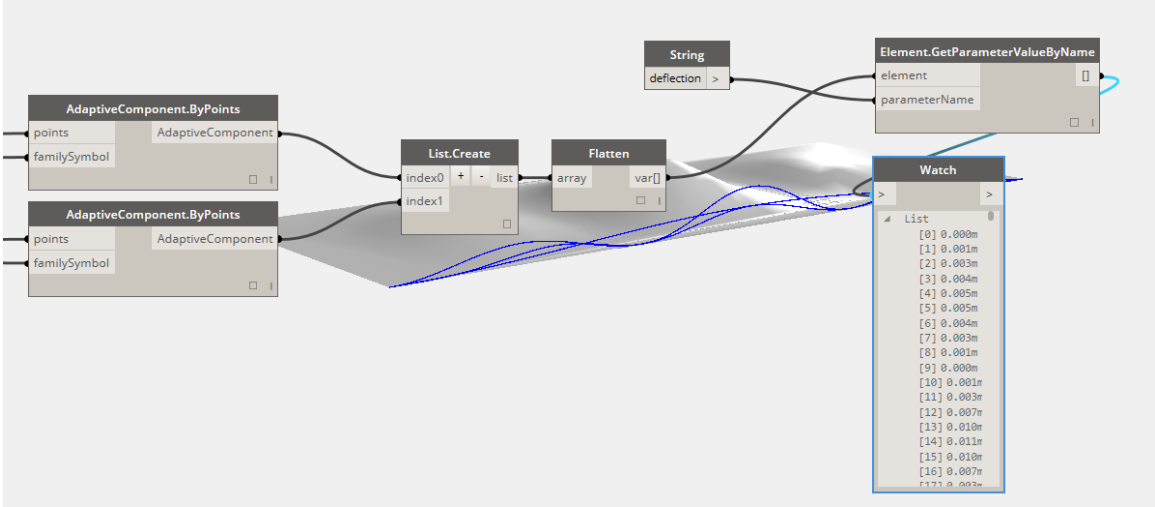
The adaptive family 4ptdef in this example is equipped with a parameter that reports the family's deflection. For details on how to build such a family I encourage you to check out Zach Kron's video lecture "Adaptive Components: From Data to taDa!" from September 21, 2012 (http://youtu.be/sZWSQJWVhbY), and the family in this Revit file.

We will use this parameter to create an intuitive and visual representation of the panel deflections in our model.

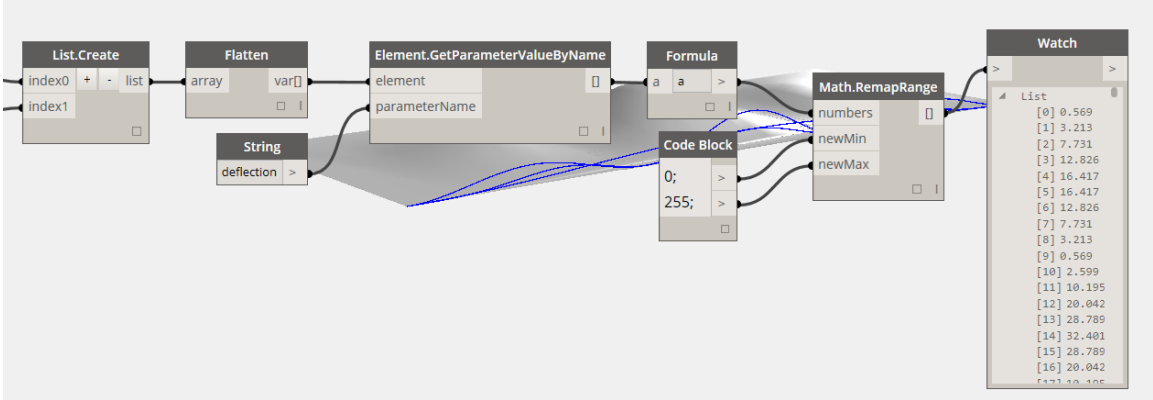
1. Add a List.Create with two inputs and a Flatten node to pull together all Adaptive Components into one list.



2. Produce an Element.GetParameterValueByName node, and a String node with the syntax
   deflection

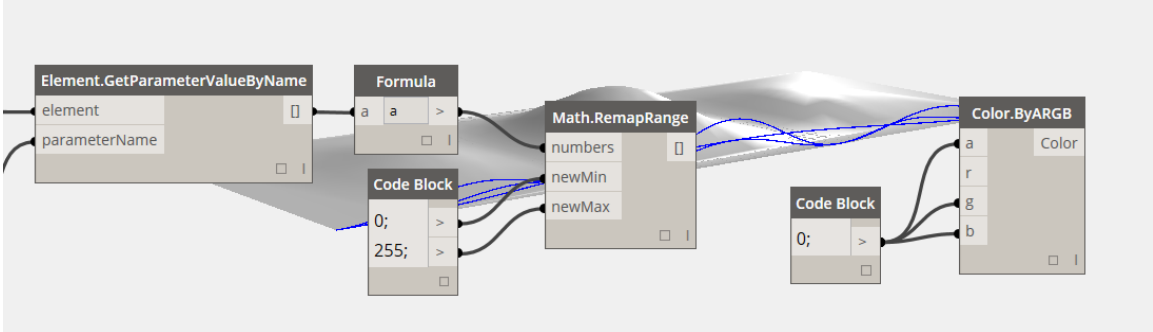


3. Pull out a Math.RemapRange node, and a Code Block with syntax
   0;
   255;
4. Add a formula node with syntax
   a

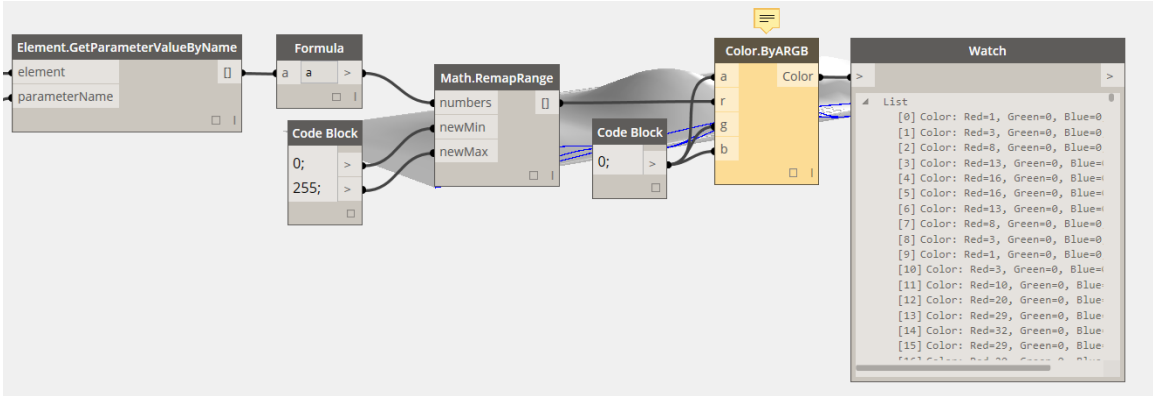


The reason behind the Formula node is we need to convert the deflection lengths (meters) to numbers. The formula node will do that. 0 and 255 represents color ranges.

5. Add a Color.ByARGB node, and wire it's a, g and b inputs to a Code Block with 0 syntax.

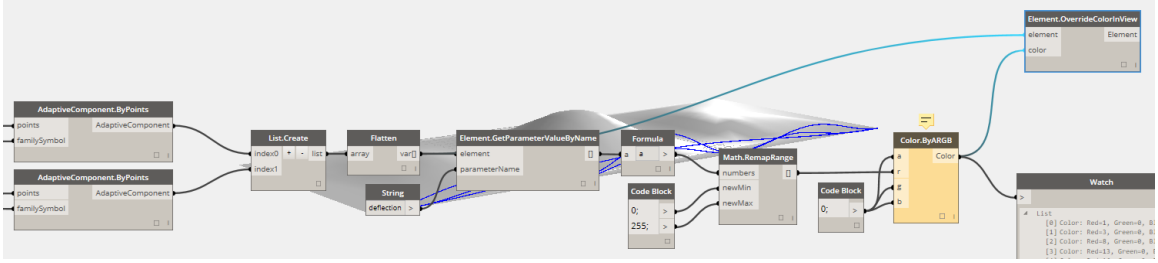


6. Wire the remapped numbers to the r input.

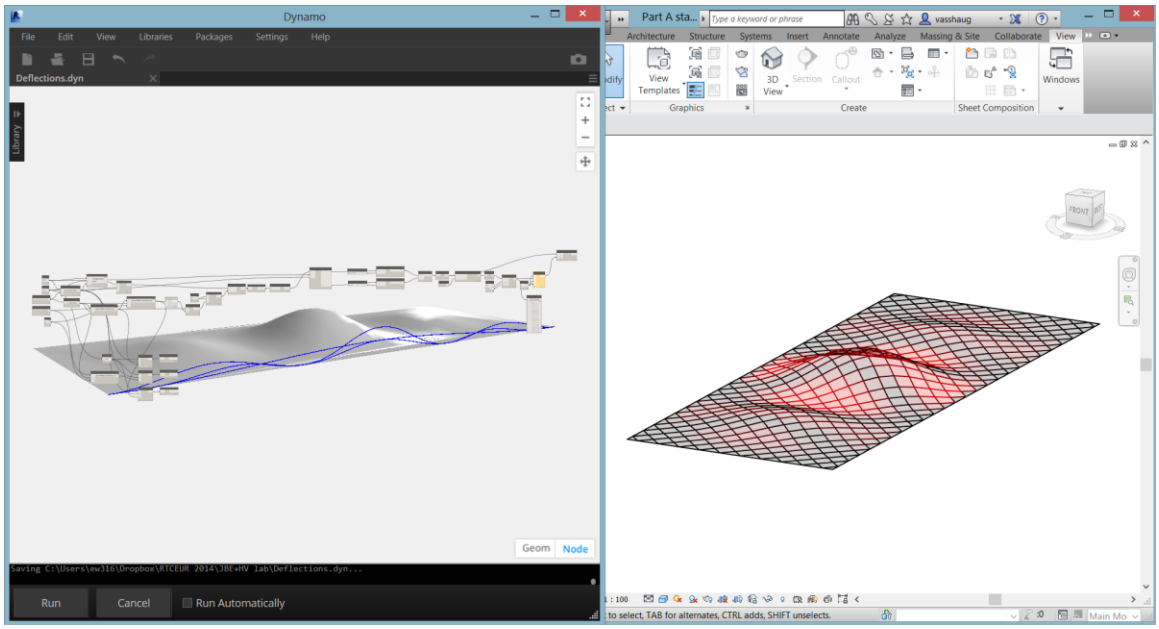


The yellow indication on the Color node is just a warning that we can ignore here.

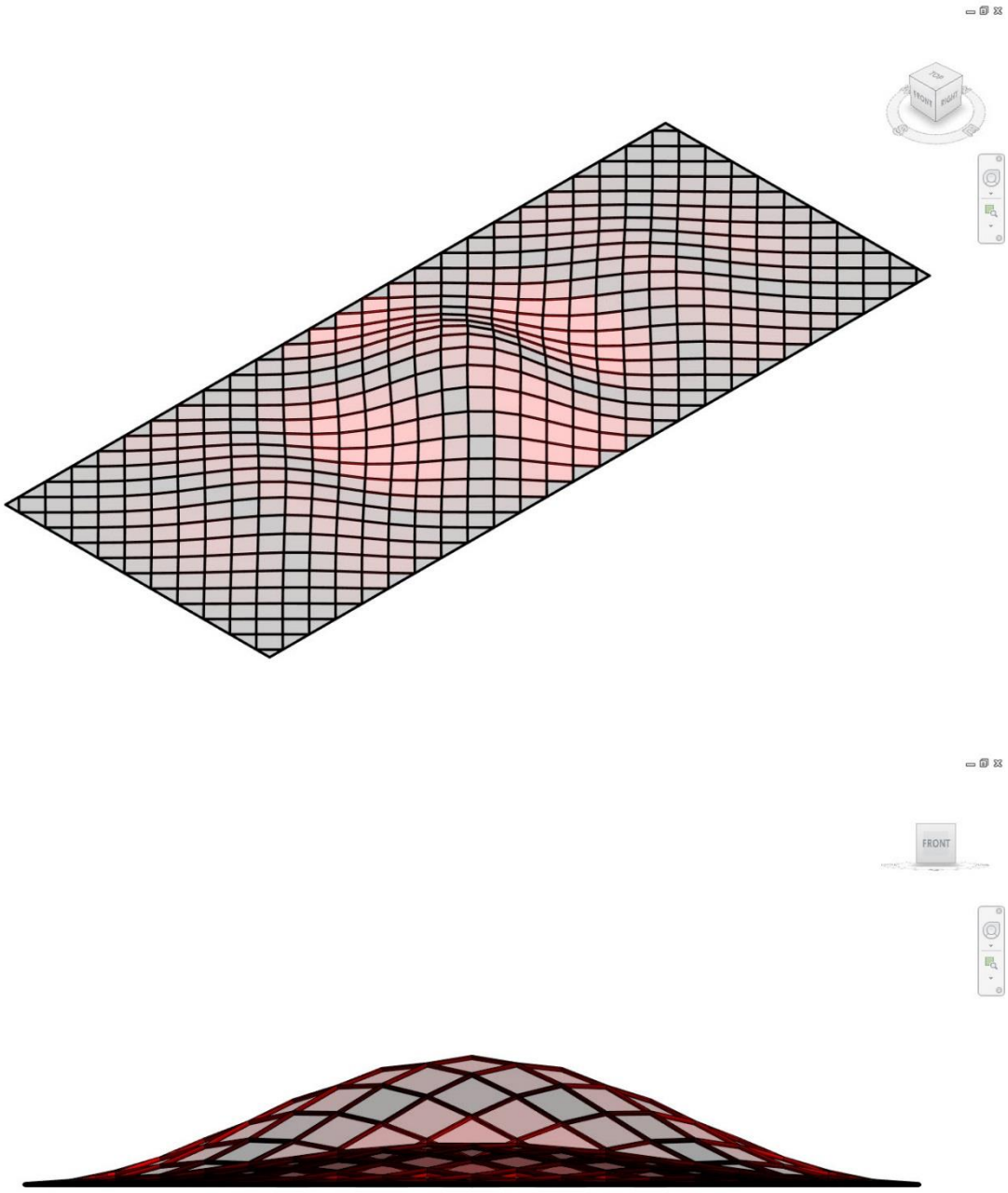
7. Add an Element.OverrideColorInView node and wire it to the collected Adaptive Components list and the color output.

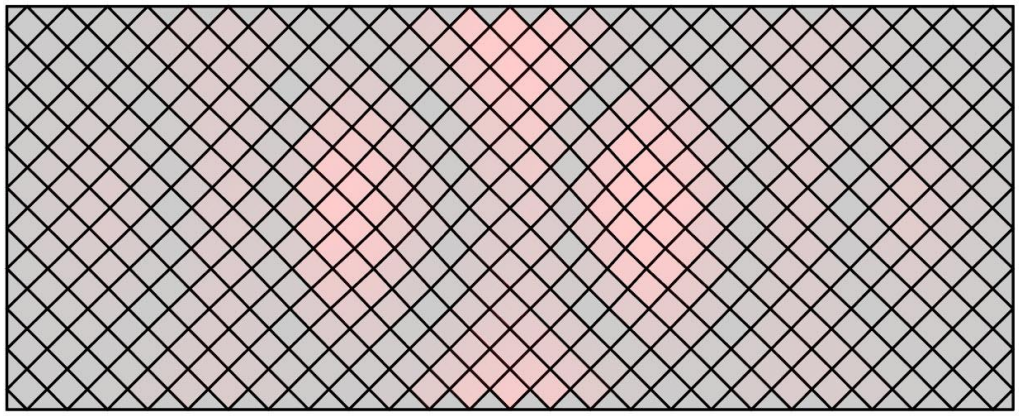


8. Press Run, and check out the open Revit view.



With this information we can produce, analyze and communicate constructability and cost at a visual and informative level that can help clients and contractors to understand what's easy and what's difficult.


FRONT

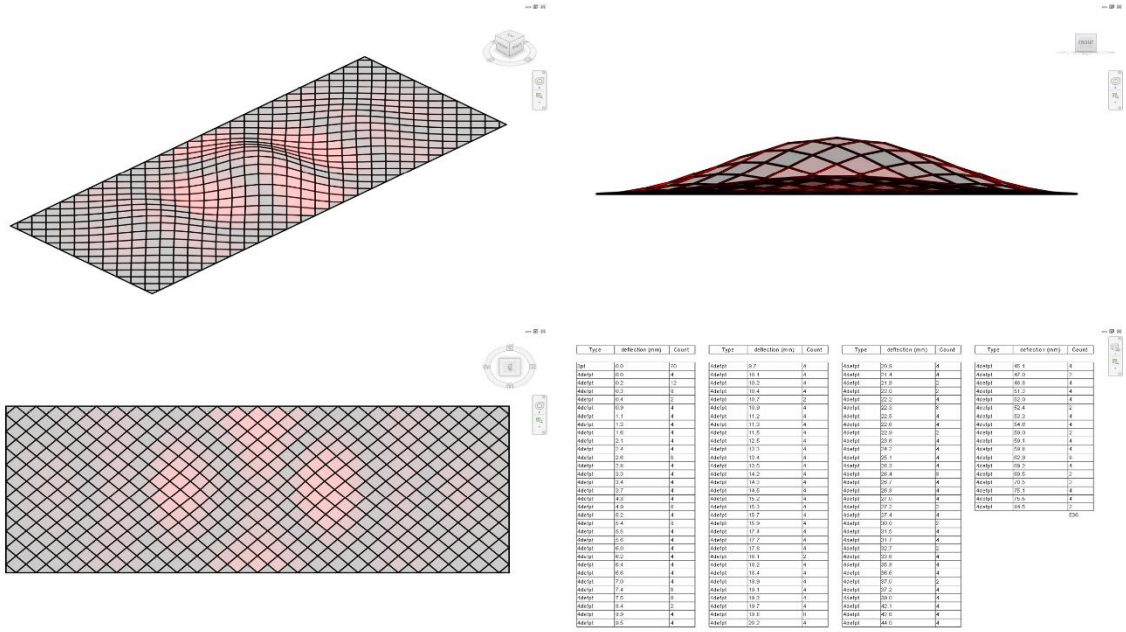


Type	deflection (mm)	Count
3pt	0.0	70
4defpt	0.0	4
4defpt	0.2	12
4defpt	0.3	8
4defpt	0.4	2
4defpt	0.9	4
4defpt	1.1	4
4defpt	1.3	4
4defpt	1.6	4
4defpt	2.1	4
4defpt	2.4	4
4defpt	2.6	8
4defpt	2.8	4
4defpt	3.3	4
4defpt	3.4	4
4defpt	3.7	4
4defpt	4.3	4
4defpt	4.9	8
4defpt	5.2	4
4defpt	5.4	8
4defpt	5.5	4
4defpt	5.6	4
4defpt	6.0	4
4defpt	6.2	4
4defpt	6.4	4
4defpt	6.6	4
4defpt	7.0	4
4defpt	7.4	8
4defpt	7.5	8
4defpt	8.4	2
4defpt	8.9	4
4defpt	9.5	4

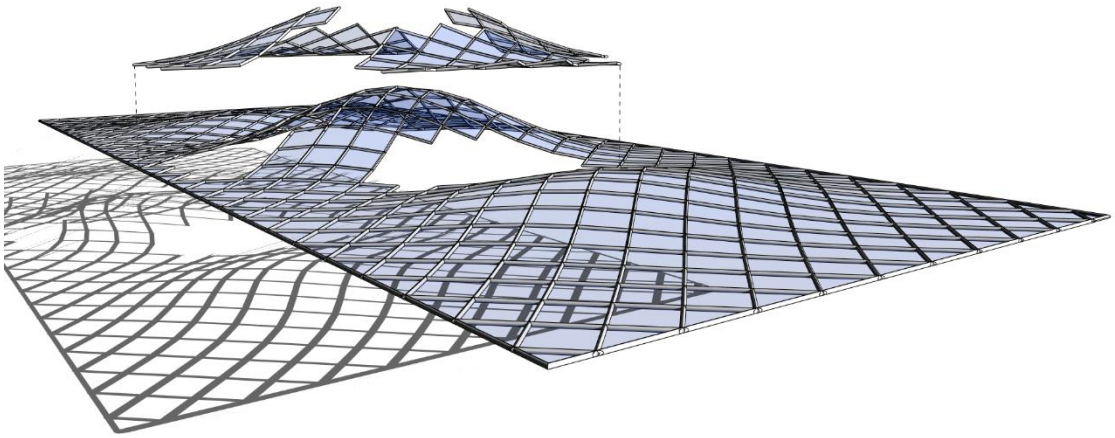
Type	deflection (mm)	Count
4defpt	9.7	4
4defpt	10.1	4
4defpt	10.2	4
4defpt	10.4	4
4defpt	10.7	2
4defpt	10.9	4
4defpt	11.2	4
4defpt	11.3	4
4defpt	11.5	4
4defpt	12.5	4
4defpt	13.3	4
4defpt	13.4	4
4defpt	13.5	4
4defpt	14.2	4
4defpt	14.3	4
4defpt	14.5	4
4defpt	15.2	4
4defpt	15.3	4
4defpt	15.7	4
4defpt	15.9	4
4defpt	17.4	4
4defpt	17.7	4
4defpt	17.8	4
4defpt	18.1	2
4defpt	18.2	4
4defpt	18.4	4
4defpt	18.9	4
4defpt	19.1	4
4defpt	19.3	4
4defpt	19.7	4
4defpt	19.8	8
4defpt	20.2	4

Type	deflection (mm)	Count
4defpt	20.9	4
4defpt	21.4	4
4defpt	21.9	2
4defpt	22.0	2
4defpt	22.2	4
4defpt	22.3	8
4defpt	22.5	4
4defpt	22.6	4
4defpt	22.9	2
4defpt	23.6	4
4defpt	24.2	4
4defpt	25.1	4
4defpt	26.3	4
4defpt	26.4	8
4defpt	26.7	4
4defpt	26.8	4
4defpt	27.0	4
4defpt	27.2	2
4defpt	27.4	4
4defpt	30.0	2
4defpt	31.5	4
4defpt	31.7	4
4defpt	32.7	2
4defpt	33.6	4
4defpt	35.8	4
4defpt	36.6	4
4defpt	37.0	2
4defpt	37.2	4
4defpt	39.0	4
4defpt	42.1	4
4defpt	42.6	4
4defpt	44.0	4

Type	deflection (mm)	Count
4defpt	45.1	4
4defpt	47.0	2
4defpt	48.8	4
4defpt	51.3	4
4defpt	52.0	4
4defpt	52.4	2
4defpt	53.3	4
4defpt	54.6	4
4defpt	59.0	2
4defpt	59.1	4
4defpt	59.8	4
4defpt	62.9	8
4defpt	69.2	4
4defpt	69.5	2
4defpt	70.5	2
4defpt	75.1	4
4defpt	75.5	4
4defpt	84.5	2
		536



Panel Deflection > 40mm



The combination of the Revit information database and scripting with Dynamo opens a world of possibilities to working fast with complex and optimized structures.

Next, we'll look at how we can integrate structural analysis in this workflow, with adaptive components, analytical models, loads and

boundary conditions generation. You will also discover some automation process to document or explore the model.

Part B

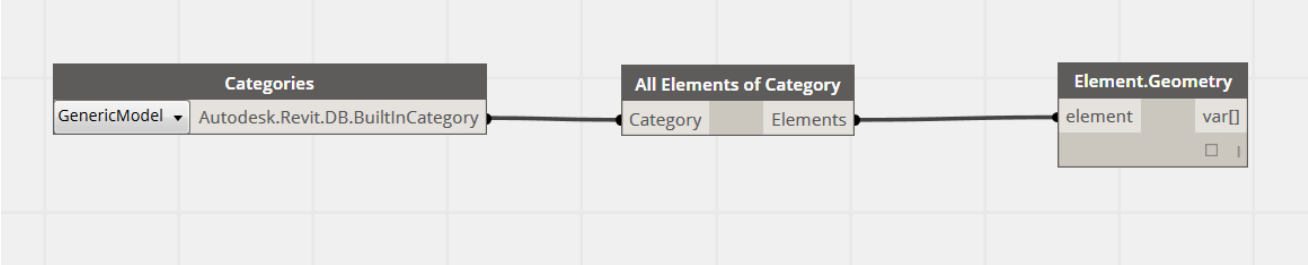
Get Analytical Model from Adaptive Components

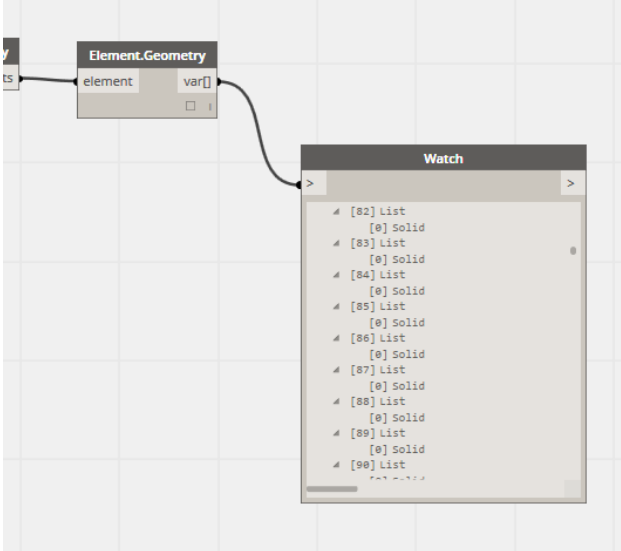
Use the model from Part A, or open the project Part A-finished.rvt

Open a new definition in Dynamo.

(As you are now familiar, I will avoid explaining all the steps to get the nodes)

Just a short try: collect all the AC from the model and look for the inner geometry.



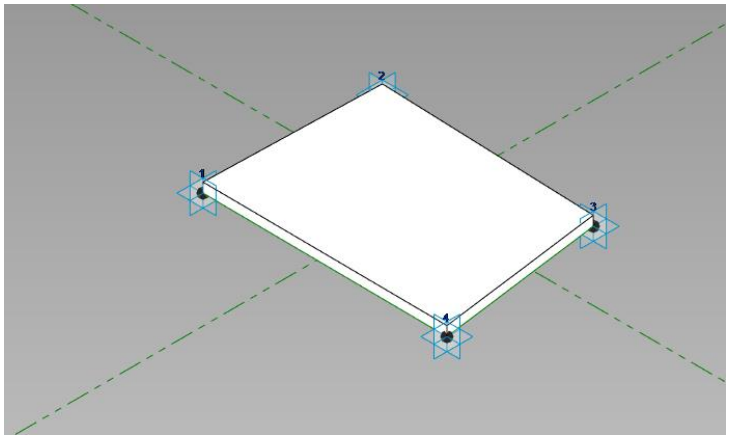


Nothing for us here. We need to do something.

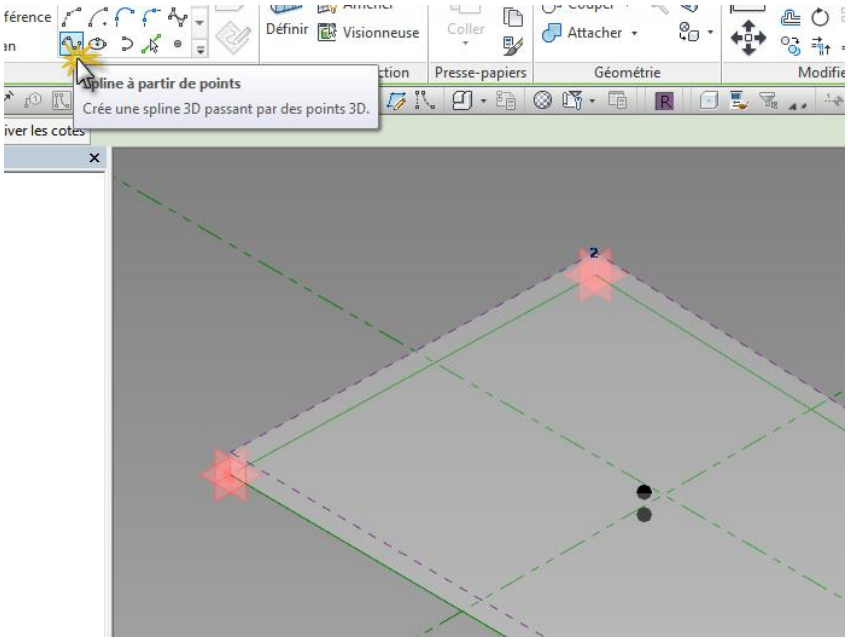
The first action is to determine what is going to be a Structural member in the AC.

The key point is: adding a few lines in the AC will give you control on what will be translated as Structural Element.

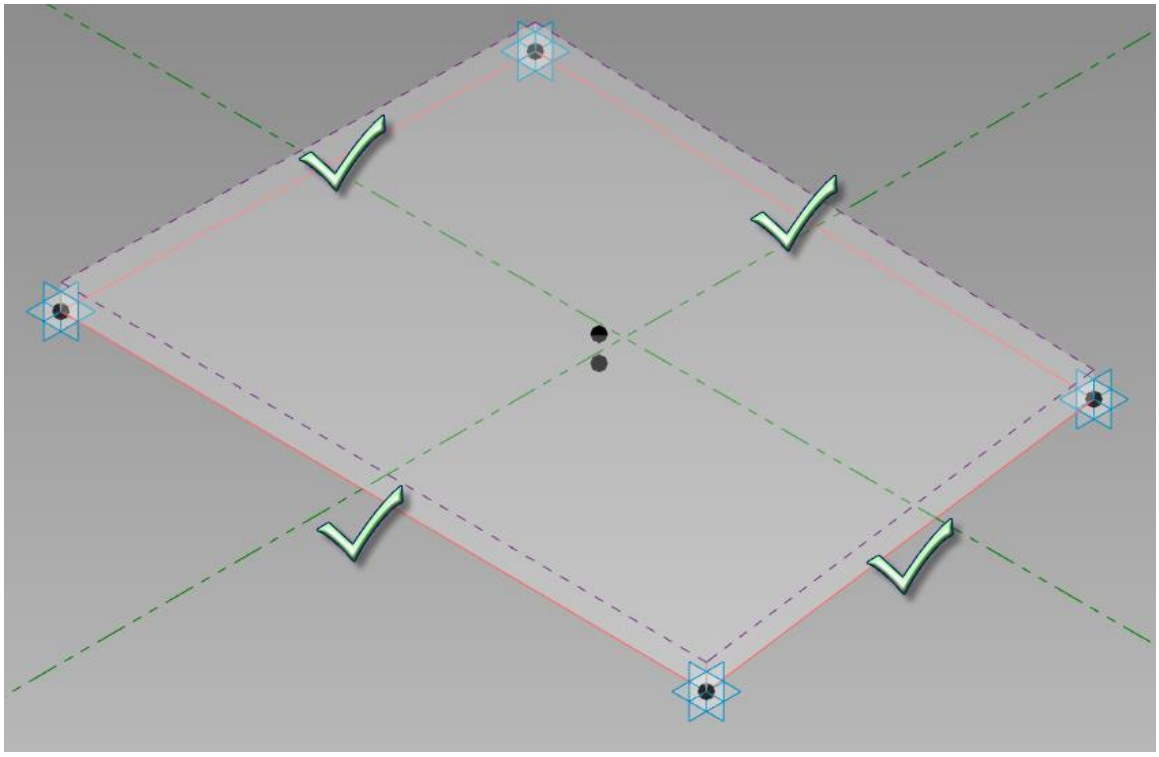
Select one AC and open the Family.



Select Adaptive Points by pair, and create a line between.



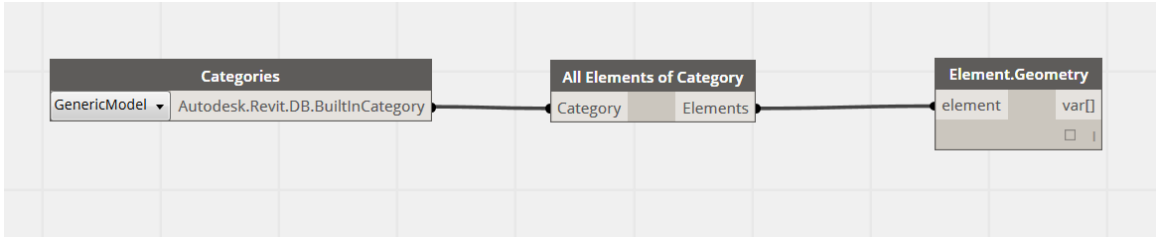
Do this for all the points to get in the end 4 lines.



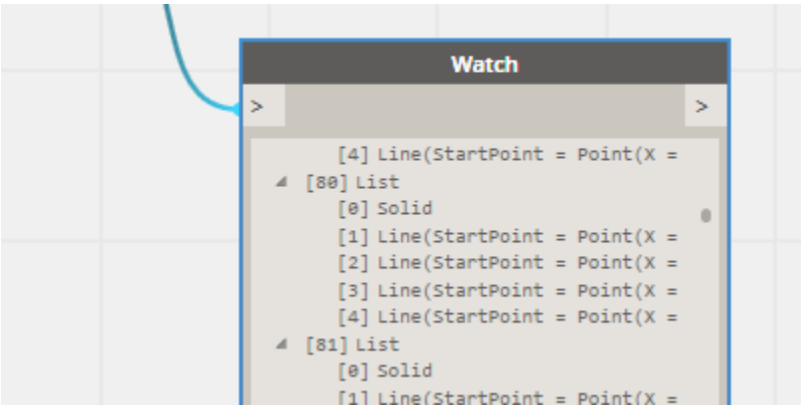
Reload family into the project.

Do this for both families of AC.

Open the xxx.dyn file.

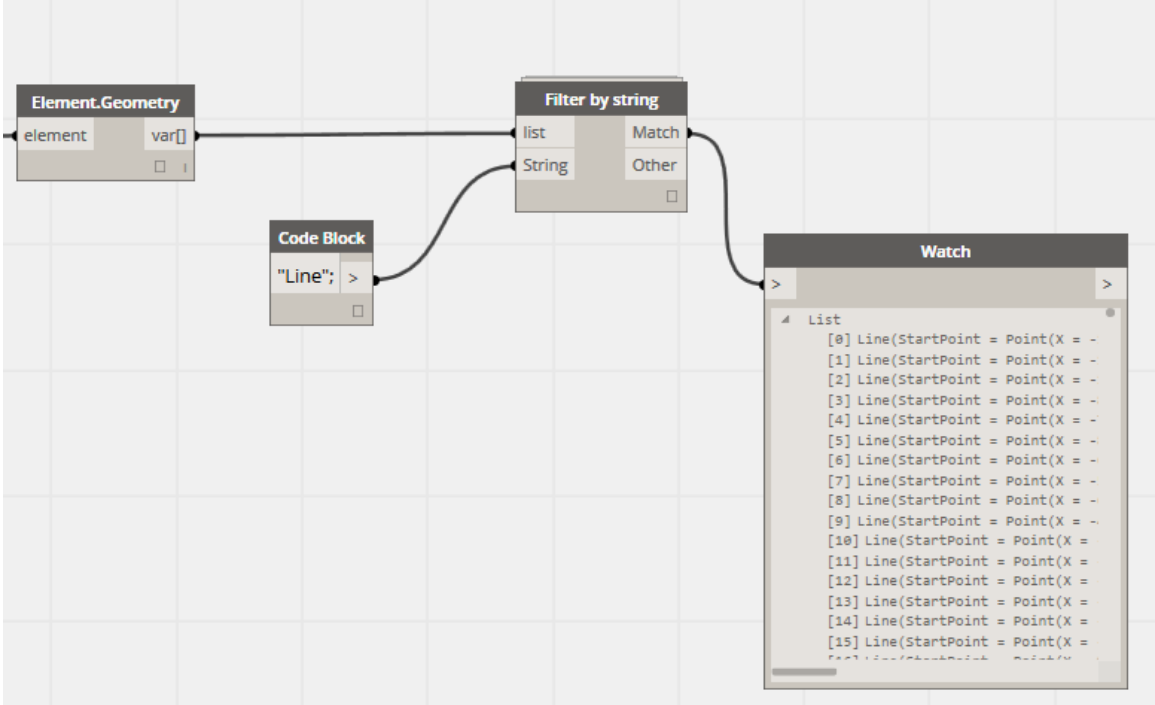


Run this again. You will see some lines, the ones we just added.



Now we have to sort this, to retrieve the lines from the list.

Look for a Filter by String custom node.



Add a Code Block, and type in the string “Line”. Result: List of line.

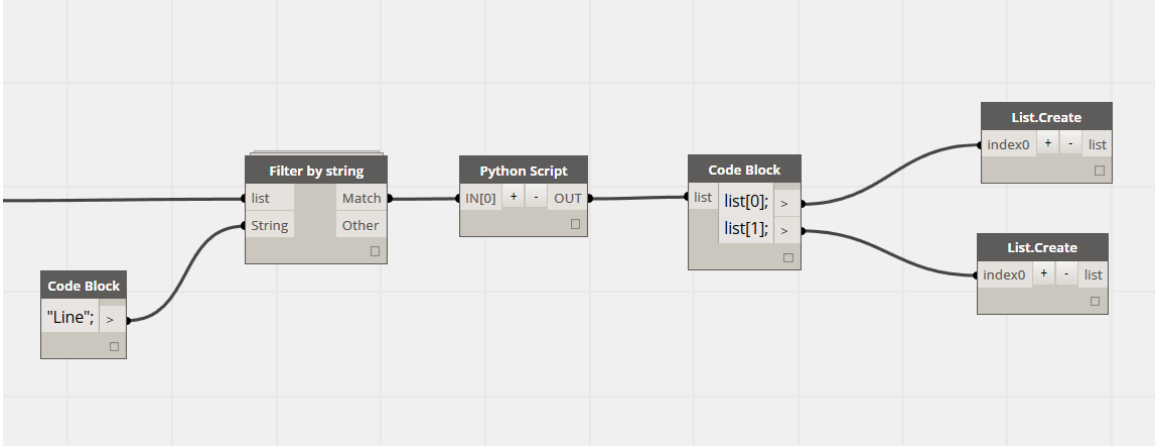
Then we will pass those guys into a specific node: obviously we have duplicated lines, we need to filter out duplicates and keep unique items.

Python is involved here.

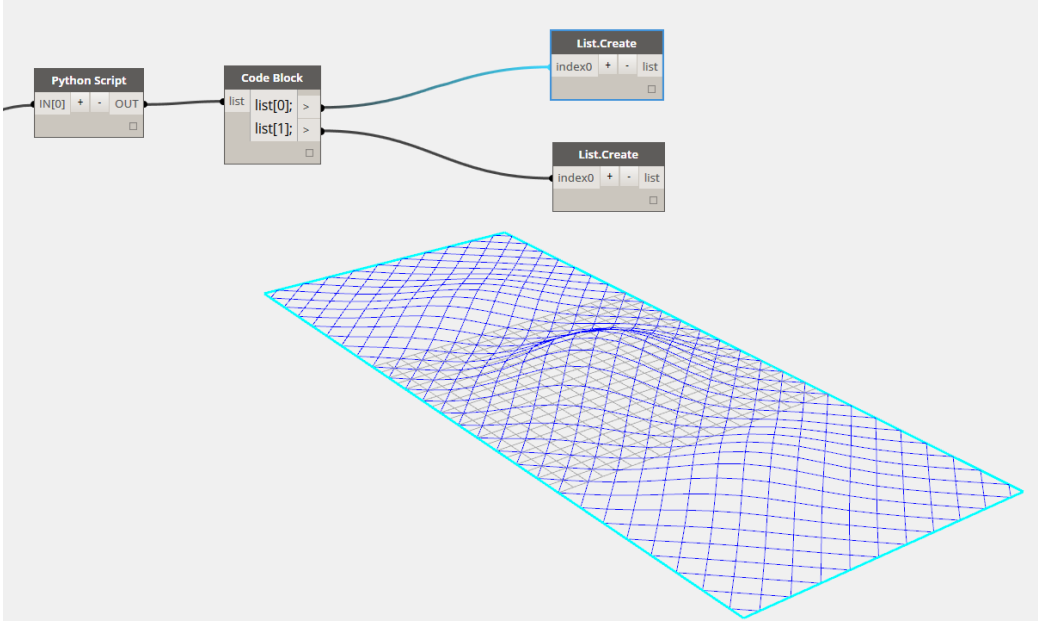
Add a Python Node, edit by double click.

Copy paste text from *Code Remove Duplicate.txt* file.

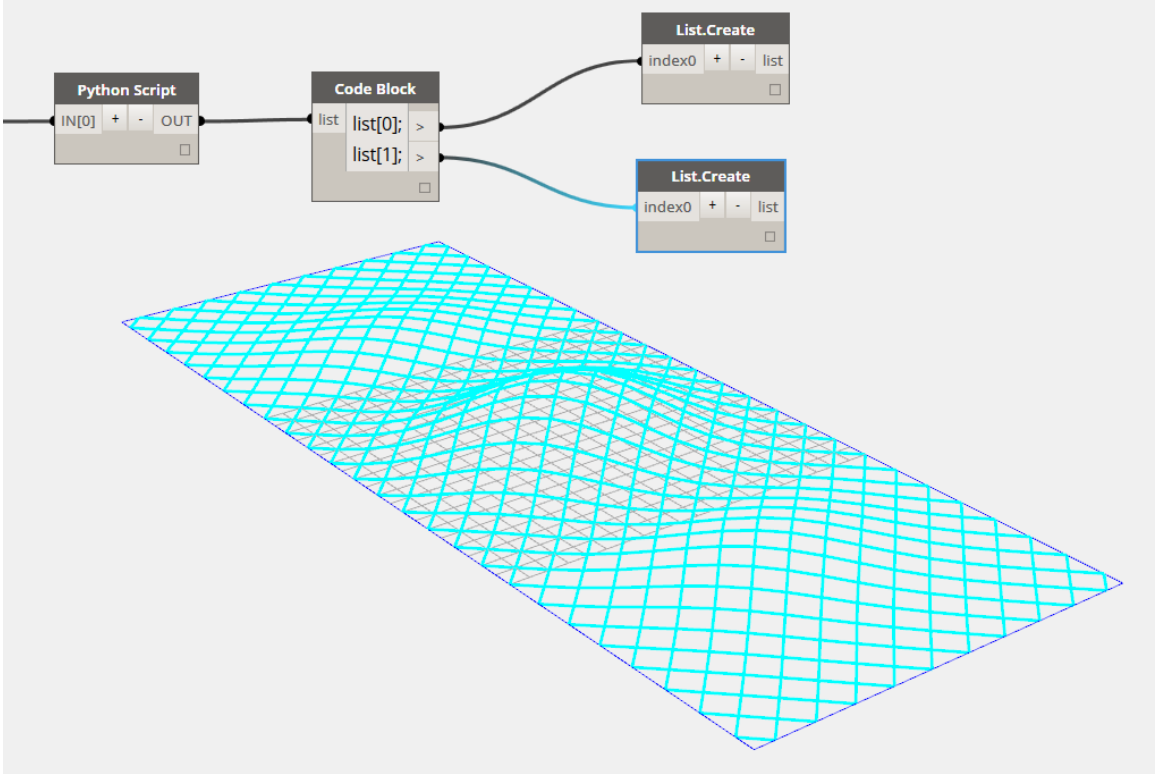
Then build this:



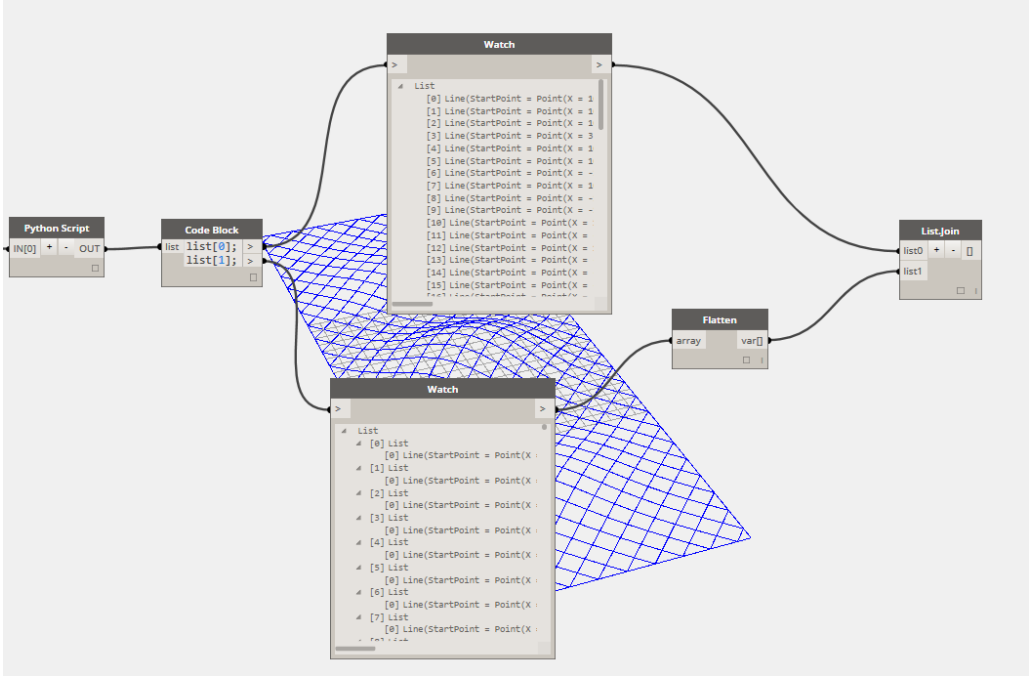
First output list: contains the edges



Second output contains the unique lines from all the duplicated.

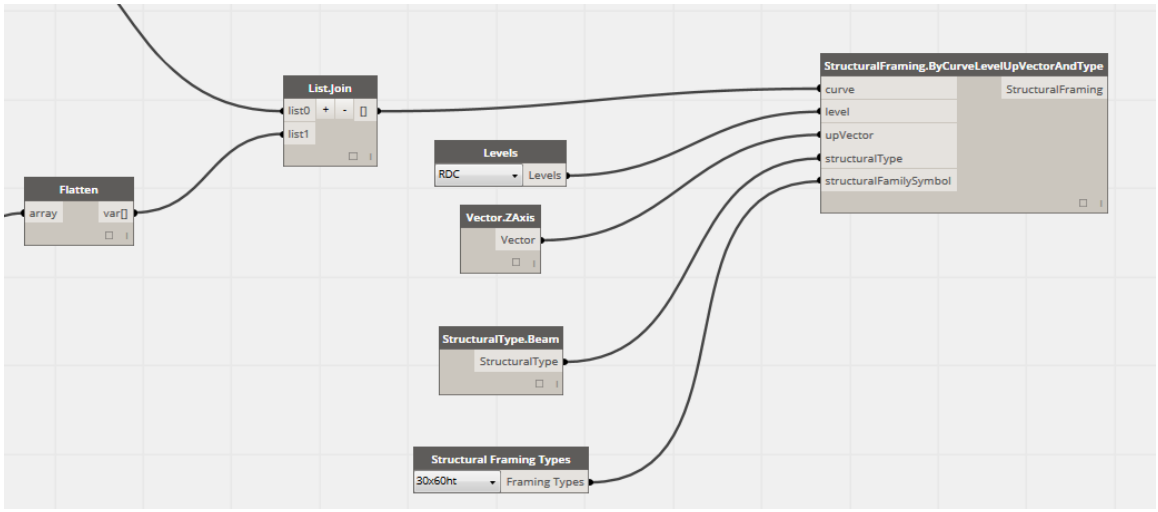


End of the exercise is piece of cake.

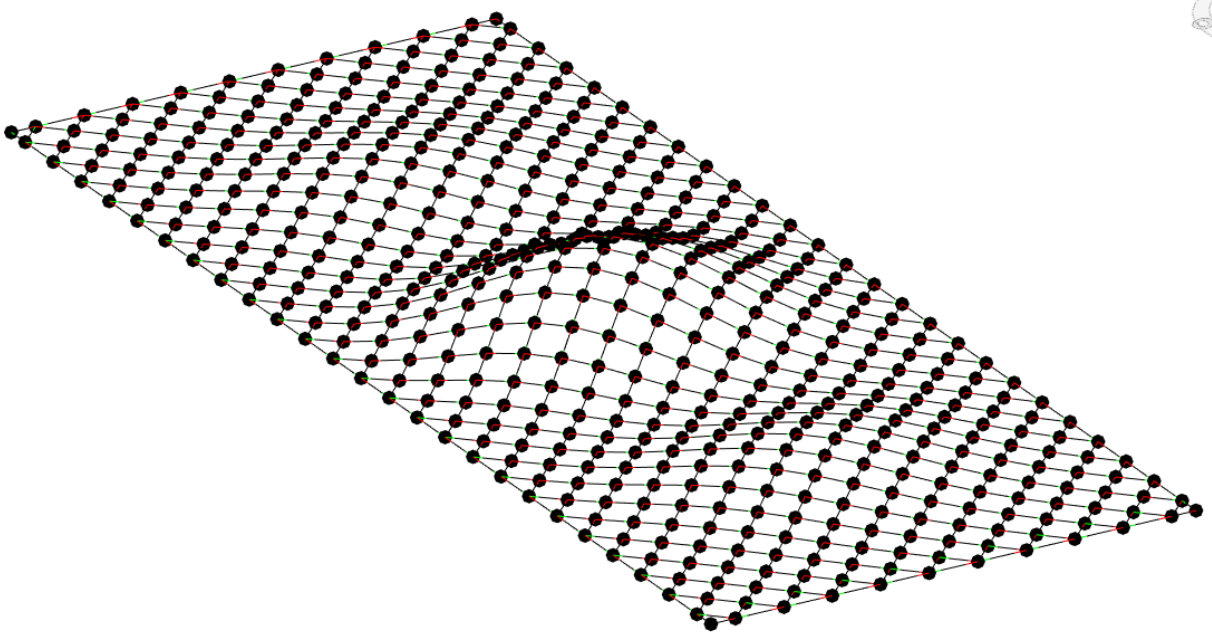


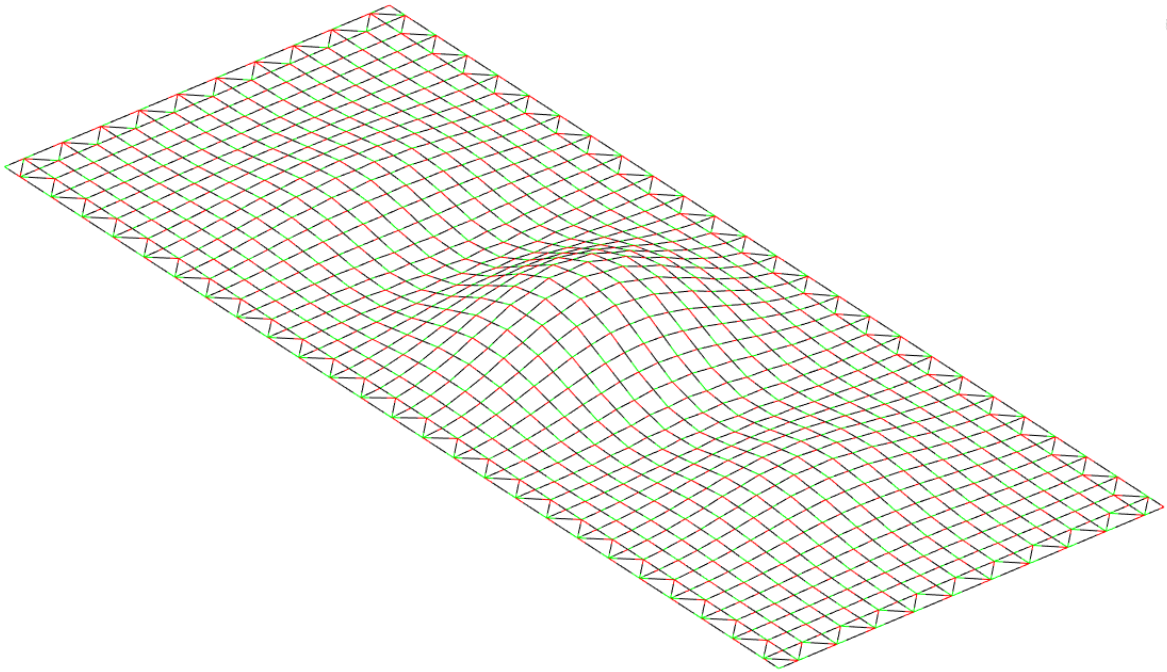
Flatten list of duplicates then use List.Join to get a single list.

Those lines are going to be curves for Structural Framing node.

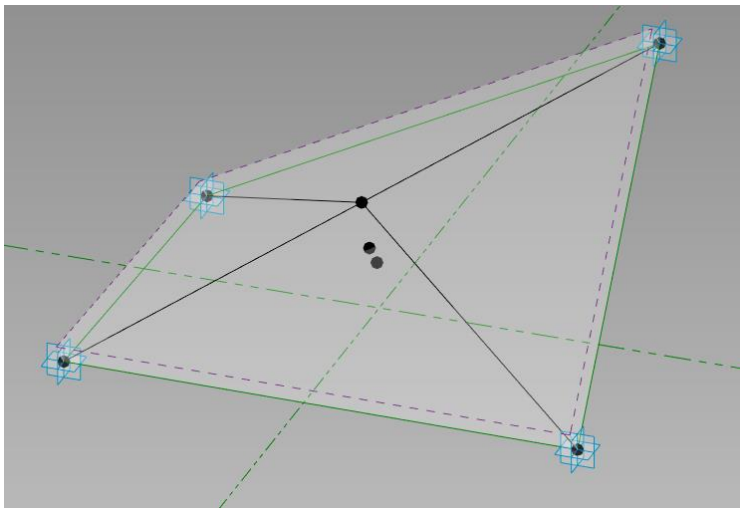


Result: analytical model from ACs.





How? Just modify the way you've edited the AC family.



Create Line loads on members

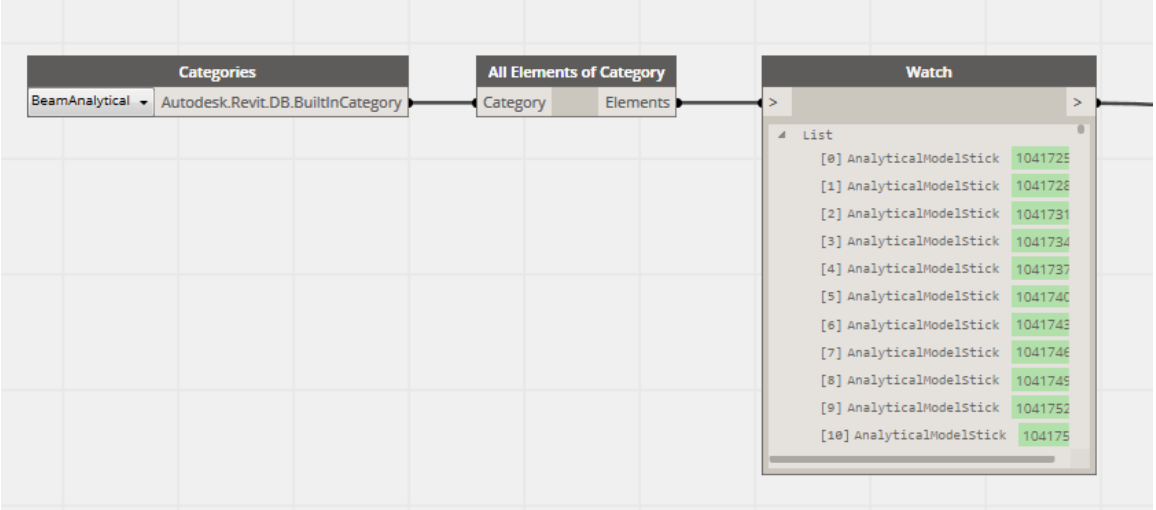
For analysis purpose we add loads to our model, here line loads.

Dynamo is a great way of combining Visual programming and Revit API.

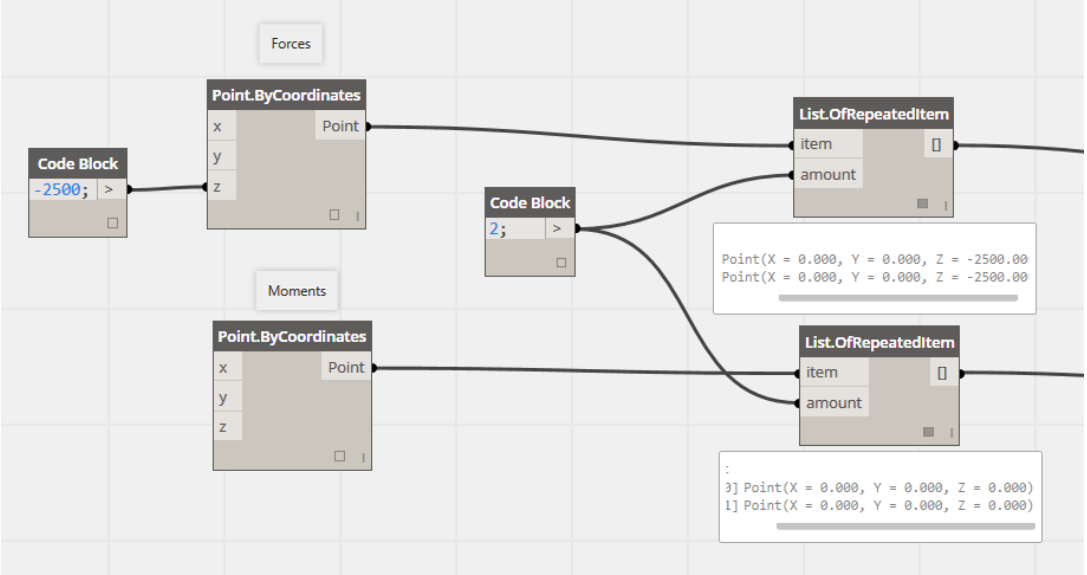
Here's how.

Use the same project file, or open Part B-AM is done.rvt

Collect Analytical beams from the project.

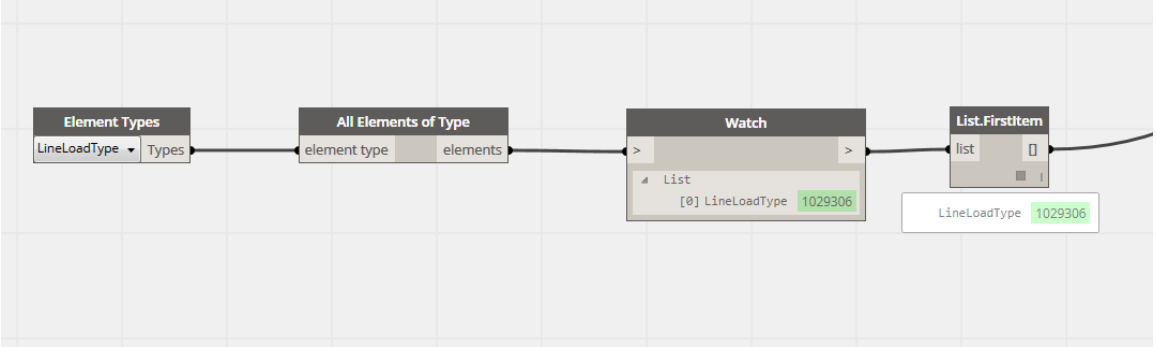


Forces and Moments are defined in Revit by XYZ, AKA Points.



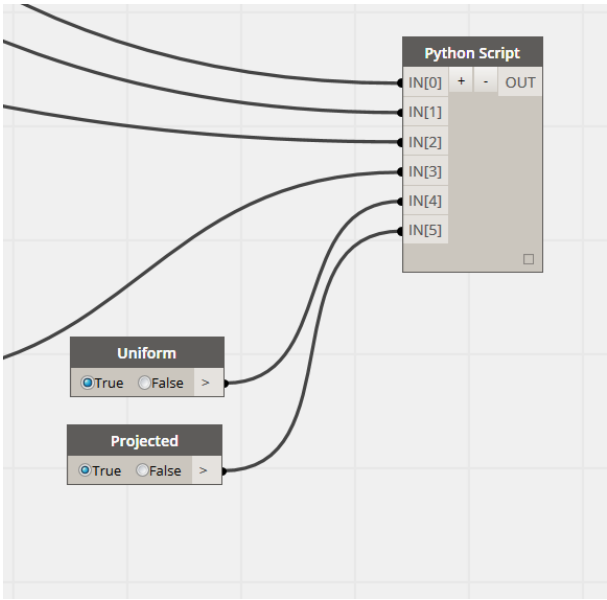
Value is negative to be applied from Top to Bottom, Z axis.

It has to be duplicated because user can choose different values at start and end of the beam. Duplicate node creates similar values.



Collect Line Load type. Tip: if the project has never use a Line Load, it will return an empty list. Just create manually a Line load first, and delete.

Now it is all set, ready to be connected in a python node we will explore more in details. Double click on Python node to edit.



Some description of what's involved.

Python script:

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
# Import ToDSType(bool) extension method
clr.AddReference("RevitNodes")
import Revit
clr.ImportExtensions(Revit.Elements)
# Import geometry conversion extension methods
clr.ImportExtensions(Revit.GeometryConversion)
# Import DocumentManager and TransactionManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager
from System.Collections.Generic import *
# Import RevitAPI
clr.AddReference("RevitAPI")
import Autodesk
from Autodesk.Revit.DB import *
from System.Collections.Generic import *

doc = DocumentManager.Instance.CurrentDBDocument
```

Settings: import all the references so that the code runs fine. Collected from various sources on the Net.

```python
forces=[]
for i in IN[1]:
    forces.append(UnwrapElement(i).ToXyz(False))

moments=[]
for i in IN[2]:
    moments.append(UnwrapElement(i).ToXyz(False))

loads=[]
loadtype=UnwrapElement(IN[3])
```

Since Dynamo 0.7, elements in dynamo are not Revit elements. Unwrapping is necessary to apply API classes.

Forces are coming from a list input, as Moments.

Loadtype comes as a single value.

ILIST creation in Python

```
37 F=List[XYZ](forces)
38 M=List[XYZ](moments)
39
40 p1=XYZ(1,0,0)
41 p2=XYZ(0,0,1)
42 p3=XYZ(0,0,0)
43 p=Plane(p1,p2,p3)
44
45 # Start Transaction
46 TransactionManager.Instance.EnsureInTransaction(doc)
47
48 skp=SketchPlane.Create(doc,p)
49 for e in elt:
50     a=doc.Create.NewLineLoad(e,F,M,IN[4],IN[5],False,loadtype,skp)
51     loads.append(a)
52
53 # End Transaction
54 TransactionManager.Instance.TransactionTaskDone()
55
56 OUT = loads
```

Line in rectangle is the one calling the API method, Python style.

NewLineLoad is asking for various inputs, defined as:

C#

```
public LineLoad NewLineLoad(
          Element host,
          IList<XYZ> forces,
          IList<XYZ> moments,
          bool uniform,
          bool projected,
          bool isReaction,
          LineLoadType symbol,
          SketchPlane plane
)
```

In details:

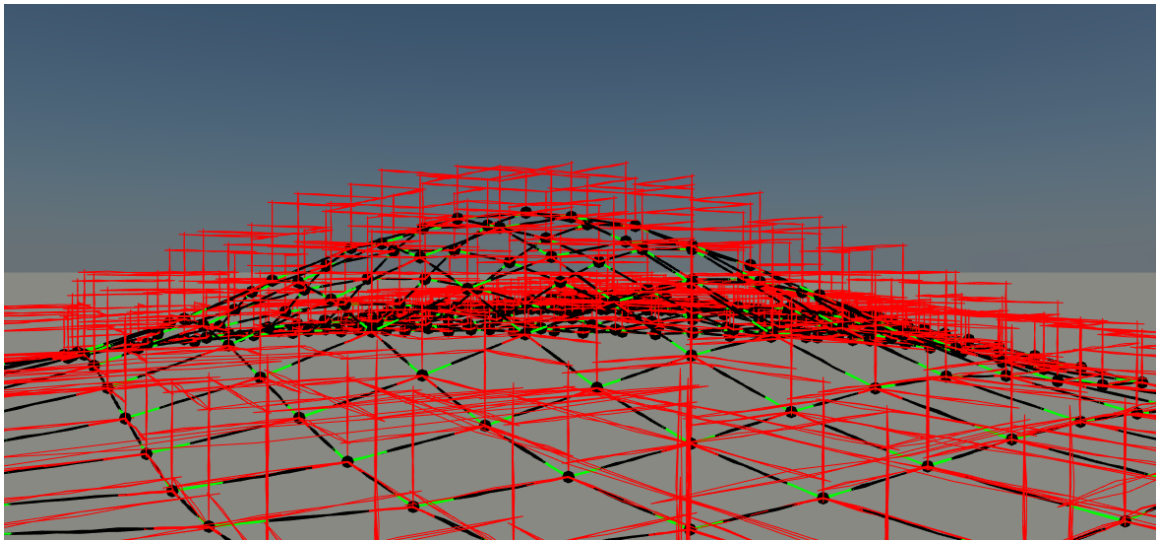
- a=doc.Create.**NewLineLoad**(e,F,M,IN[4],IN[5],False,loadtype,skp)
- doc = Revit document
- Create.NewLineLoad = method
- e = input from the list *(for e in elt)*
- F, M = ILIST of XYZ
- IN[4], IN[5]: boolean
- False = boolean for IsReaction (if true, it as frozen in the project and can't be reversed)
- Loadtype = selected LineLoadType

- Skp = Sketchplane as an XZ plane

All information are coming from the Revit API help file provided in the Revit SDK.

So feed the method as indicated, and it will run.

Result: bunch of line loads in a single click.



Let go further.

Create Boundary Conditions

For analysis the model has to be connected to supports, by boundary conditions.

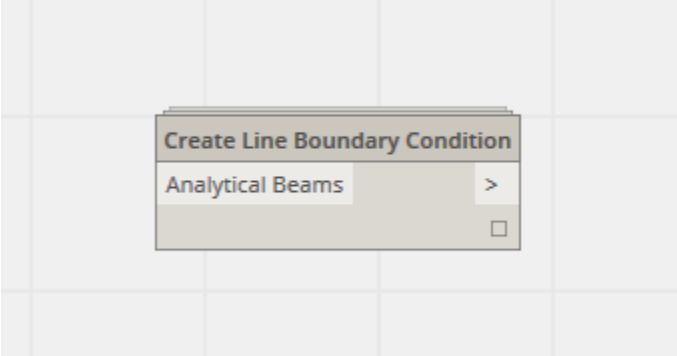
In this case, making it manually one by one will be painful and risky.

Use a custom node provided named Create Line Boundary Condition.

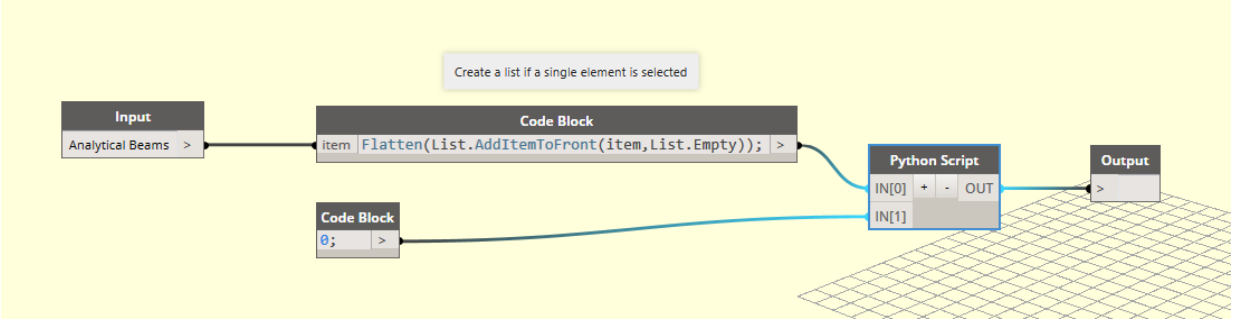
Same project, or open Part B-Loads finished.rvt.

New definition in Dynamo.

Goal:



Looking inside the node:



And Python code:

Edit Python Script...

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
# Import ToDSType(bool) extension method
clr.AddReference("RevitNodes")
import Revit
clr.ImportExtensions(Revit.Elements)
# Import geometry conversion extension methods
clr.ImportExtensions(Revit.GeometryConversion)
# Import DocumentManager and TransactionManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager
from System.Collections.Generic import *
# Import RevitAPI
clr.AddReference("RevitAPI")
import Autodesk
from Autodesk.Revit.DB import *
from System.Collections.Generic import *

doc = DocumentManager.Instance.CurrentDBDocument

elt=UnwrapElement(IN[0])

R=float(IN[1])

bc=[]
# Start Transaction
TransactionManager.Instance.EnsureInTransaction(doc)

for e in elt:
    s=Autodesk.Revit.DB.Structure.TranslationRotationValue.Fixed
    a=doc.Create.NewLineBoundaryConditions(e,s,R,s,R,s,R,s,R)
    bc.append(a)

# End Transaction
TransactionManager.Instance.TransactionTaskDone()

OUT =bc
```

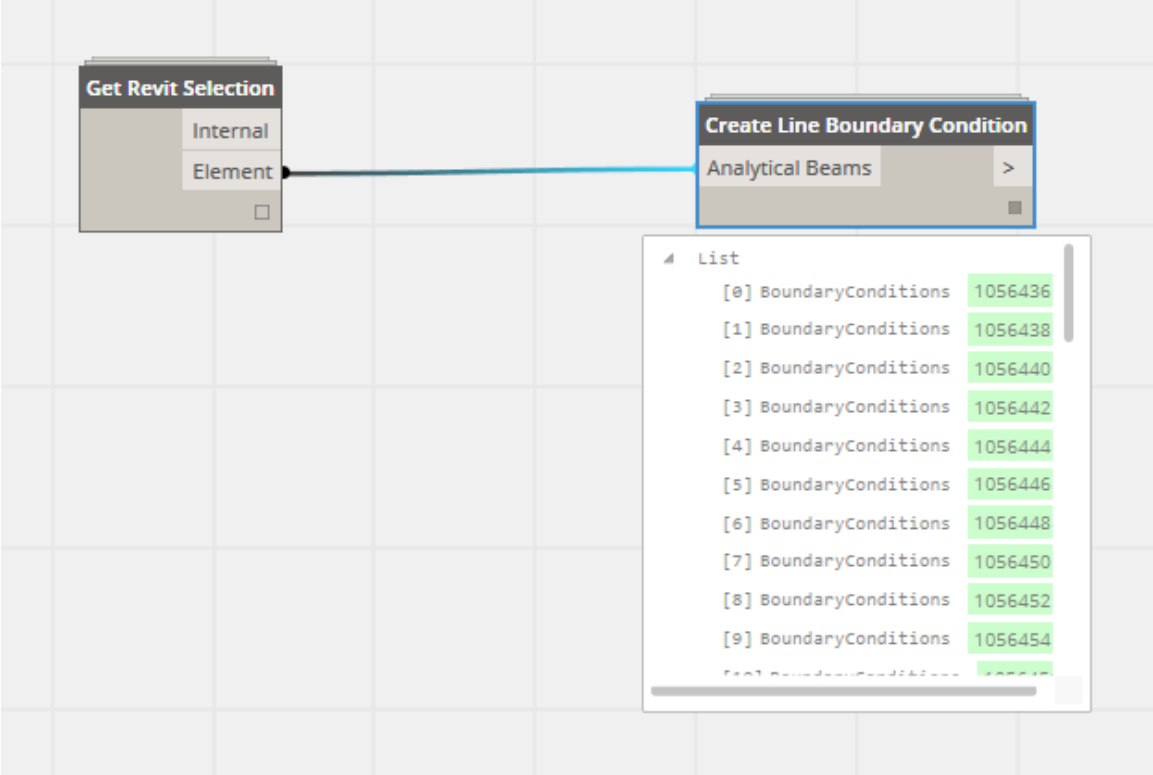
Accept Changes | Cancel

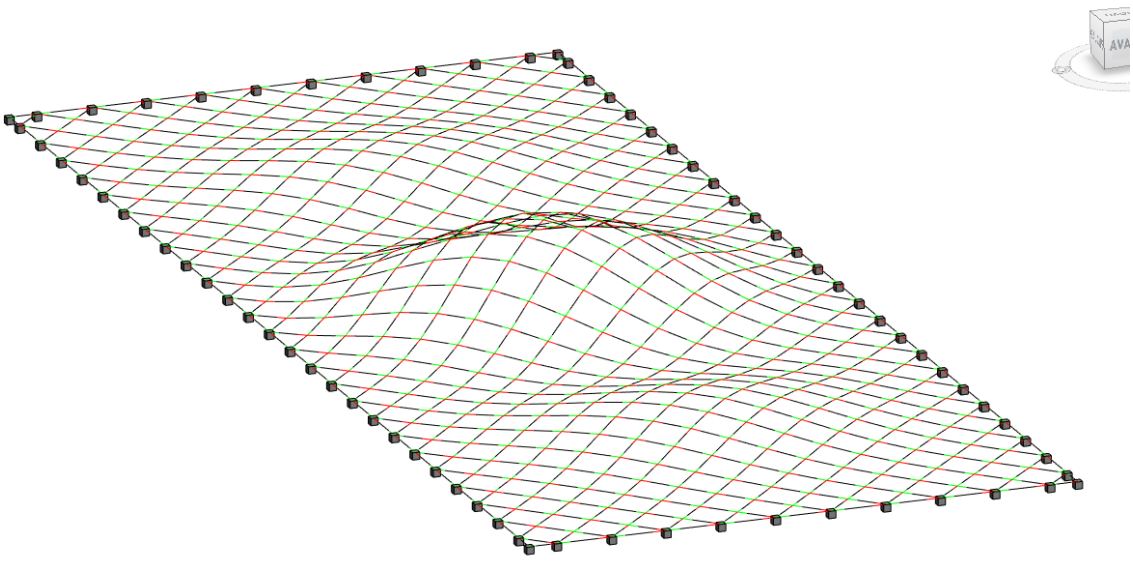
Purpose here is not to redo this but get live explanation of what's involved.

Filtering Analytical beams from the edges from dynamo is tricky, needs extra effort and is waste of time here.

Just carefully selected edges with rectangular selection.

Add a Get Revit Selection node in the graph. Done.



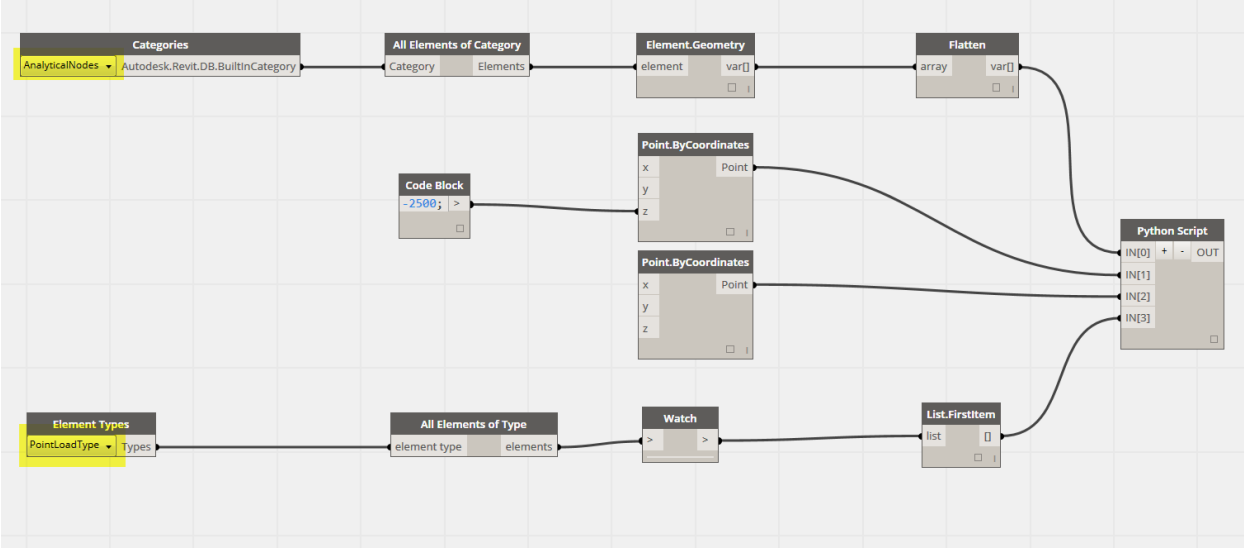


The model is ready for analysis.

Extra: Create Point Load custom node

As an extra, I suggest to make a new custom node to insert Point Loads in the model.

Same Revit project, open 0.7 RTC04-create line loads.dyn



We just have to make some modification in definition and Python node.

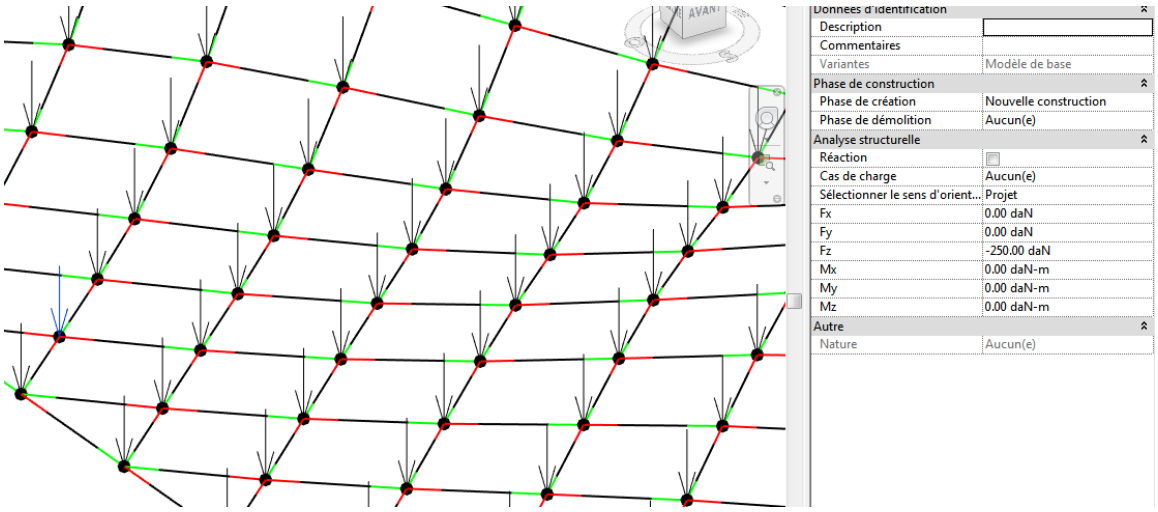
Python will now use the class NewPointLoad as defined in the Revit API help file.

```
public PointLoad NewPointLoad(
          XYZ point,
          XYZ force,
          XYZ moment,
          bool isReaction,
          PointLoadType symbol,
          SketchPlane plane
)
```

So edit the custom node to get this result, explanation are given live.

```
23
24 points=[]
25 for i in IN[0]:
26     points.append(UnwrapElement(i).ToXyz(True))
27
28 F=UnwrapElement(IN[1]).ToXyz(True)
29 M=UnwrapElement(IN[2]).ToXyz(True)
30
31 loads=[]
32 loadtype=UnwrapElement(IN[3])
33
34 p1=XYZ(1,0,0)
35 p2=XYZ(0,0,1)
36 p3=XYZ(0,0,0)
37 p=Plane(p1,p2,p3)
38
39 # Start Transaction
40 TransactionManager.Instance.EnsureInTransaction(doc)
41
42 skp=SketchPlane.Create(doc,p)
43 for x in points:
44     a=doc.Create.NewPointLoad(x,F,M,False,loadtype,skp)
45     loads.append(a)
46
47 # End Transaction
48 TransactionManager.Instance.TransactionTaskDone()
49
50 OUT = loads
```

If needed, open the definition 0.7 RTC06-create point loads.dyn.



Thanks for listening!