AUTODESK UNIVERSITY 2014

Computational Logic in Structural Design

Håvard Vasshaug – Dark Architects

SE6925-L

The visual programming interface of Dynamo is enabling structural engineers with the tools to build optimized structures with minimal energy, and subsequently make their own design tools. Based on the Revit Platform, we can use our creativity to develop optimized structural systems using computational logic in an advanced building information modeling environment. This lab will teach participants how to create and iterate computational space frames with native Revit Framing elements and Adaptive Components, and how these can be used in structural analysis.

Learning Objectives

1. Learn what Dynamo is, and why we want to make our own Structural Design tools.
2. Learn how to use Dynamo to create Computational Space Frame Structures.
3. Learn how to integrate the structural analysis tools in Revit software with the Dynamo extension and conceptual modeling
4. Learn how to integrate Structural Analysis tools in Revit with Dynamo and Conceptual Modeling.

About the Speaker

Håvard Vasshaug is a Structural Engineer (M.Sc.), BIM Manager and Design Technologist 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.

hvasshaug@gmail.com | vasshaug.net | @vasshaug | facebook.com/hvasshaug

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 exercise 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 0.7.4.3245.

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

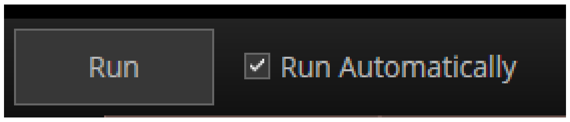
Table of Contents

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

Basics

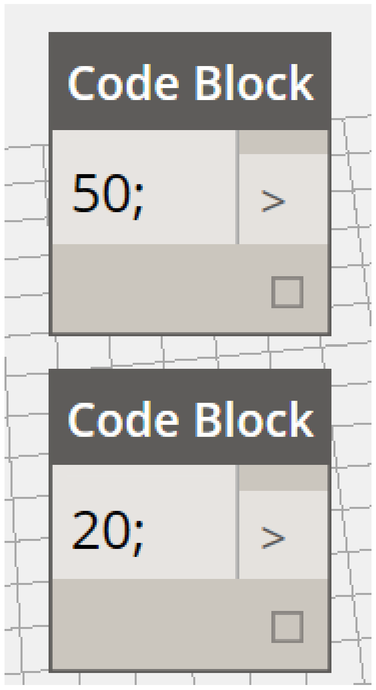
5. Start Revit 2015 and Dynamo 0.7.3.

6. Turn on Run Automatically in Dynamo.

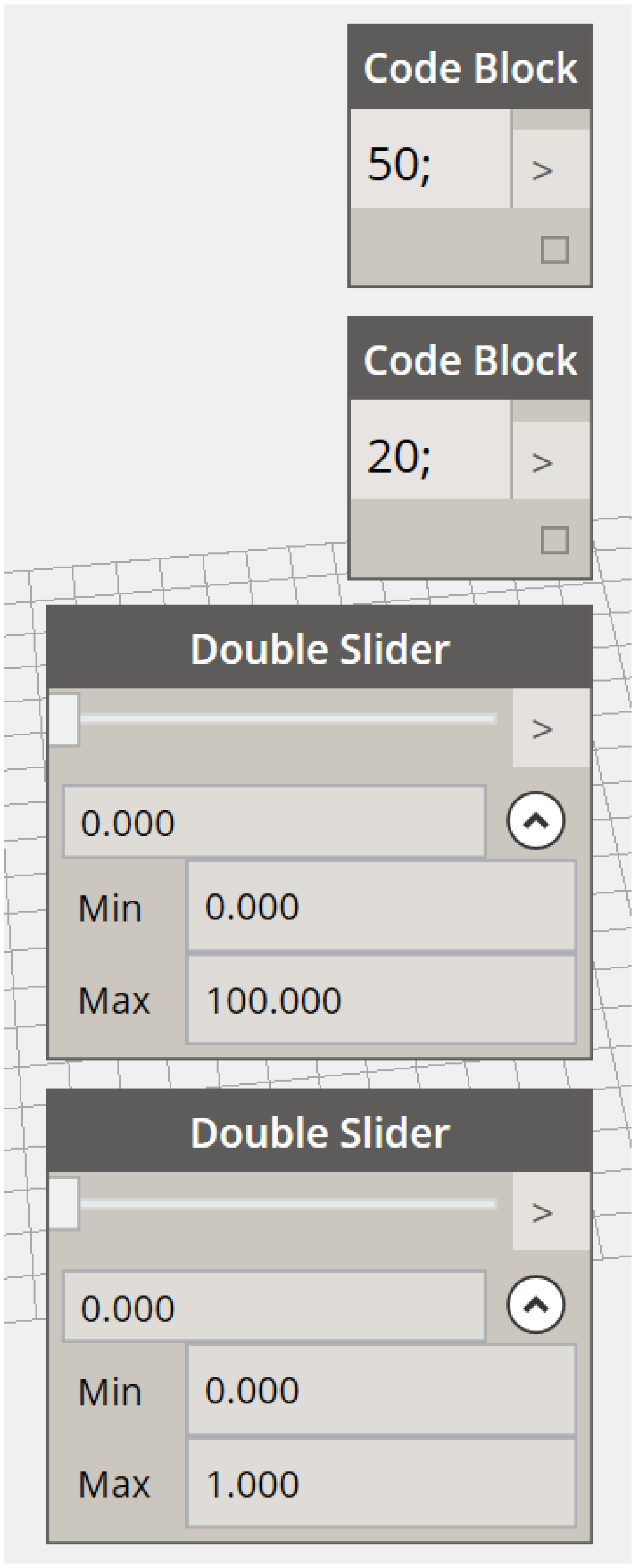


7. Double click in canvas to produce two Code Blocks. Enter values

   50;
   20;

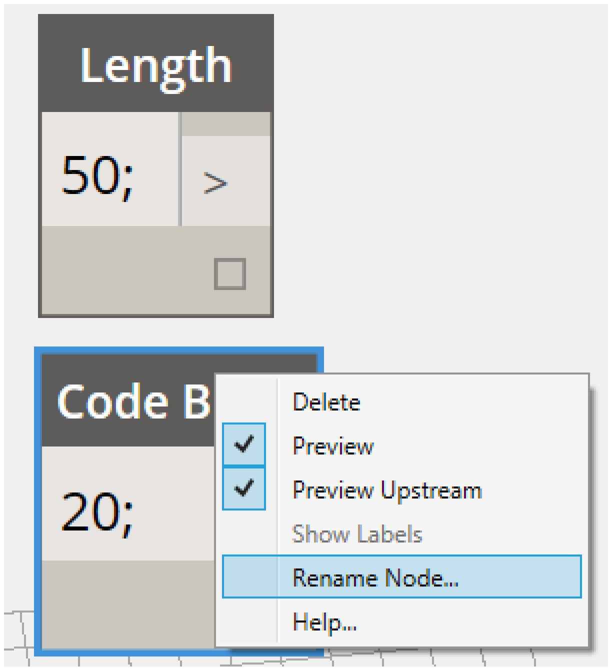


8. Search for Double Sliders in Library Search, and produce two nodes with Min and Max values 0 and 100/1 respectively.


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

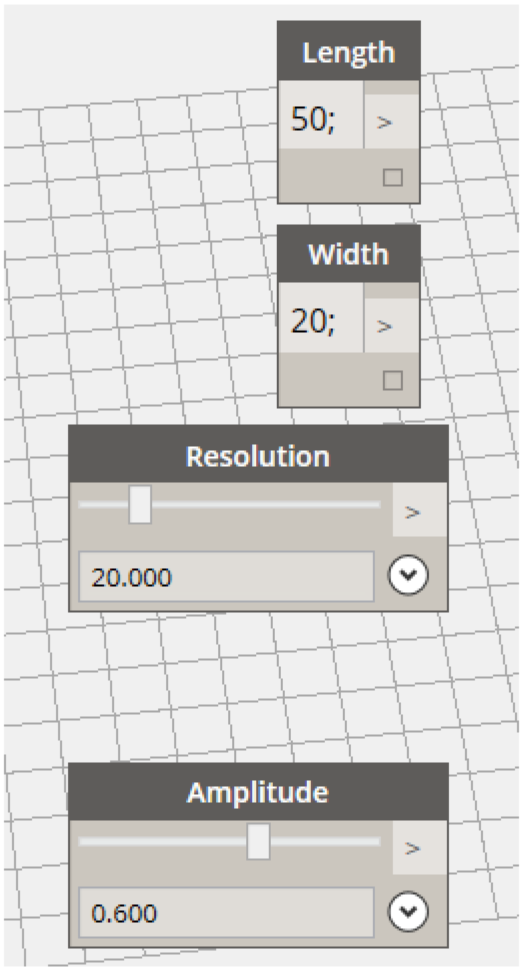
9. Right Click on all 4 nodes and change their names to

   Length
   Width
   Resolution
   Amplitude



10. Change values of Resolution and Amplitude to

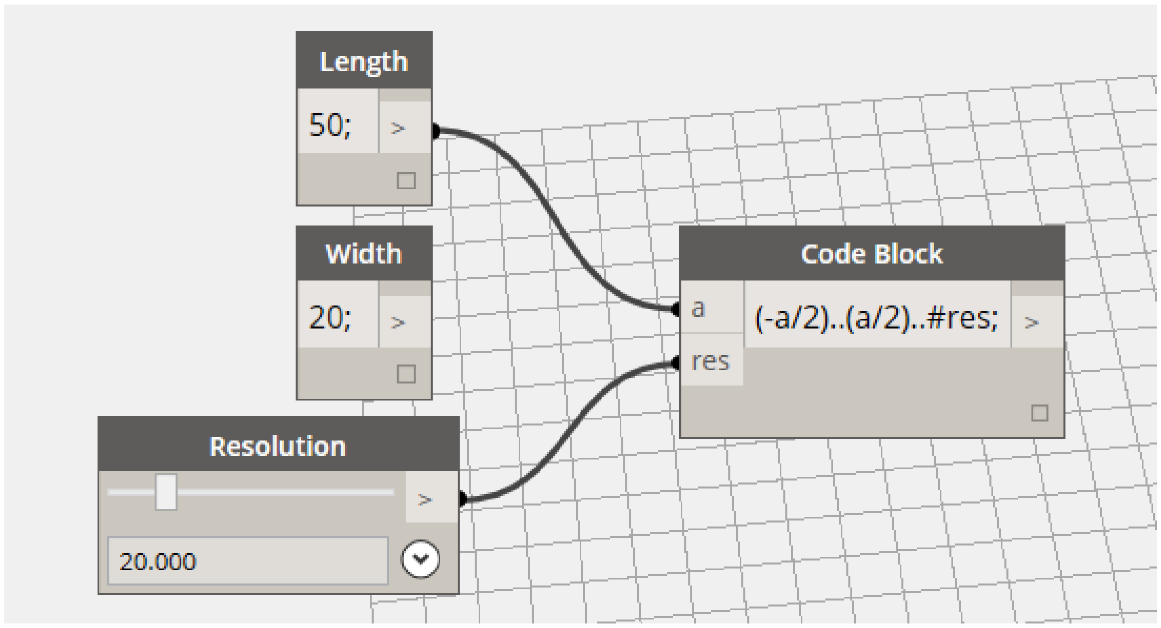
    20
    0.6



11. Make a Code Block with the syntax

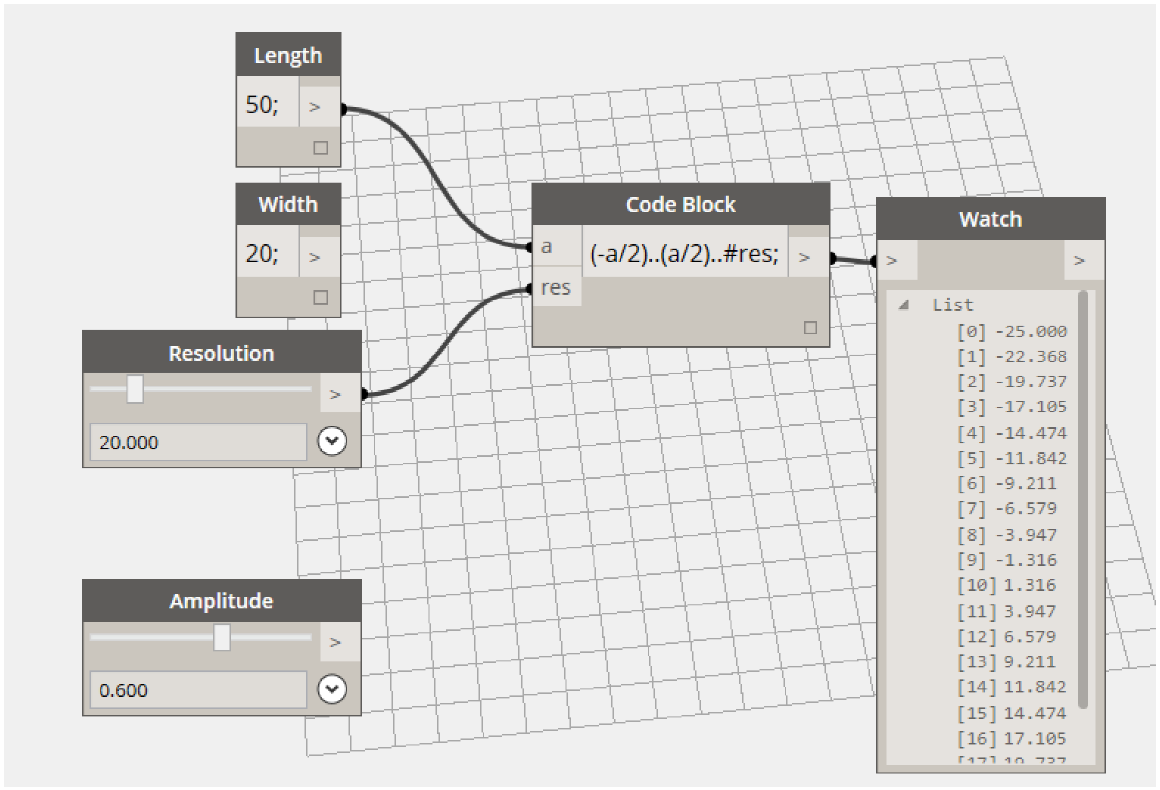
    (-a/2)..(a/2)..#res;

    and wire it to Length and Resolution.



```
(-a/2)..(a/2)..#res;
```

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



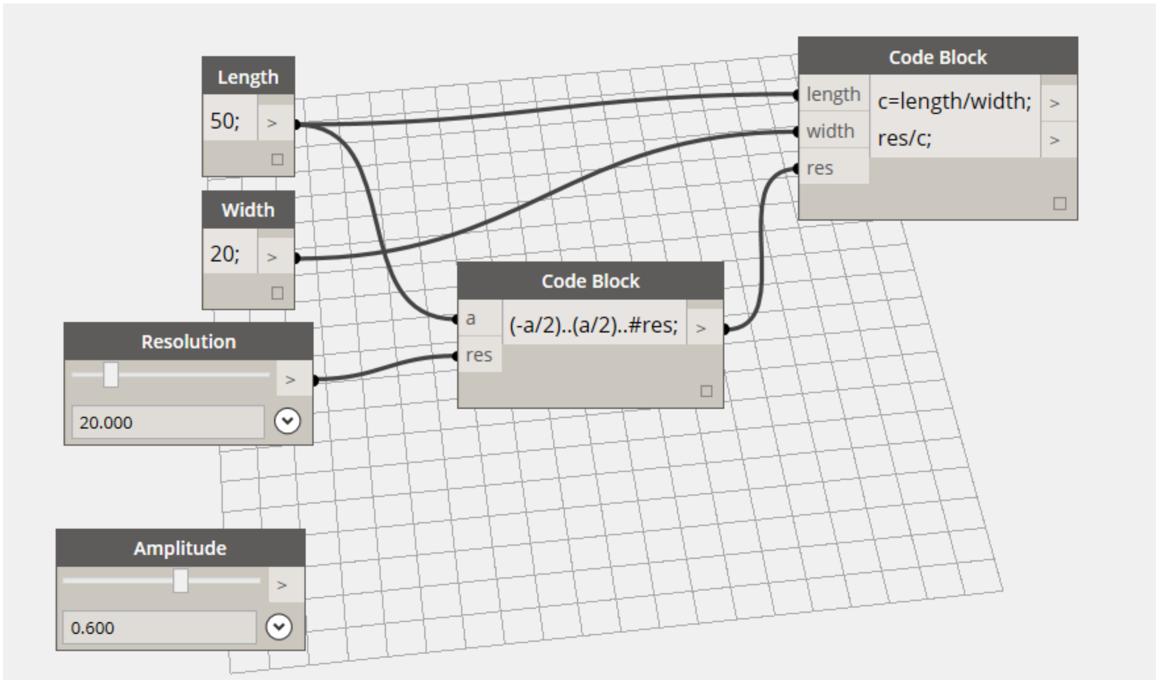
```
(-a/2)..(a/2)..#res;
```

```
List
  [0] -25.000
  [1] -22.368
  [2] -19.737
  [3] -17.105
  [4] -14.474
  [5] -11.842
  [6] -9.211
  [7] -6.579
  [8] -3.947
  [9] -1.316
  [10] 1.316
  [11] 3.947
  [12] 6.579
  [13] 9.211
  [14] 11.842
  [15] 14.474
  [16] 17.105
```

13. Make another Code Block with the syntax

c=length/width;
res/c;

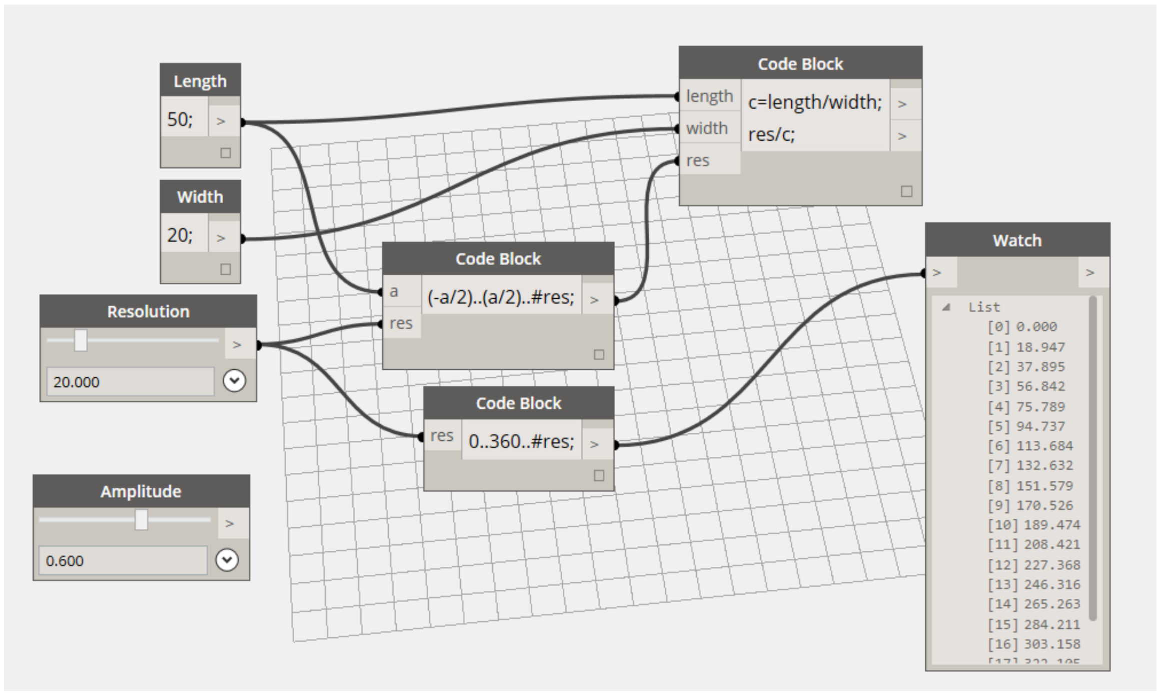
14. Wire it to Length, Width and the output of the previous Code Block.



15. Make a Code Block with syntax

0..360..#res;

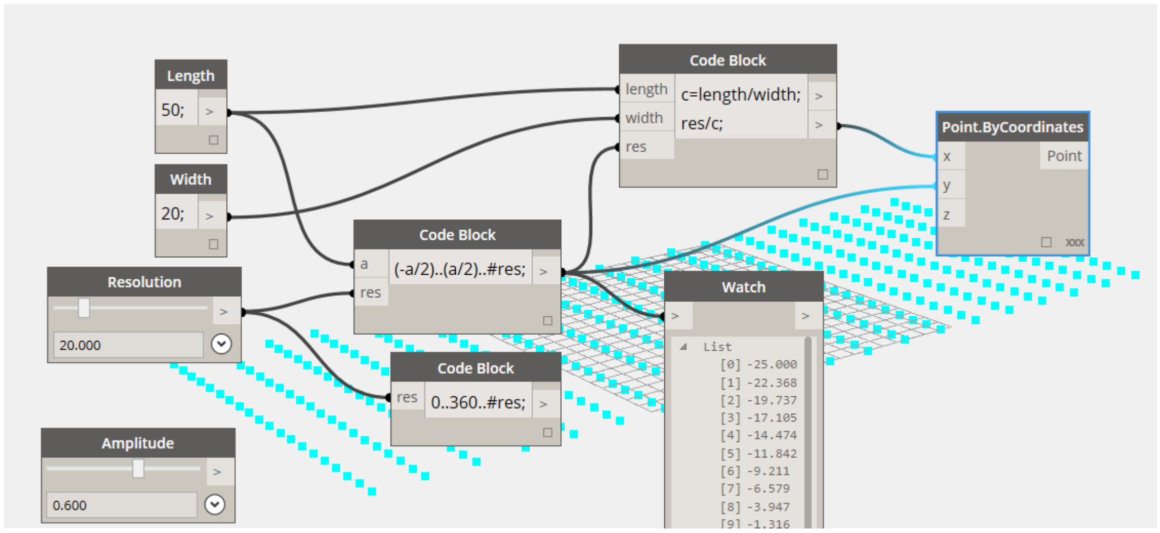
and wire it to Resolution



```
Length: 50;
Width: 20;
Resolution: 20.000
Amplitude: 0.600
Code Block: c=length/width;
            res/c;
Code Block: (-a/2)..(a/2)..#res;
Code Block: 0..360..#res;
```

```
List
  [0] 0.000
  [1] 18.947
  [2] 37.895
  [3] 56.842
  [4] 75.789
  [5] 94.737
  [6] 113.684
  [7] 132.632
  [8] 151.579
  [9] 170.526
  [10] 189.474
  [11] 208.421
  [12] 227.368
  [13] 246.316
  [14] 265.263
  [15] 284.211
  [16] 303.158
```

16. 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.)



```
Length: 50;
Width: 20;
Resolution: 20.000
Amplitude: 0.600
Code Block: c=length/width;
            res/c;
Code Block: (-a/2)..(a/2)..#res;
Code Block: 0..360..#res;
```

```
List
  [0] -25.000
  [1] -22.368
  [2] -19.737
  [3] -17.105
  [4] -14.474
  [5] -11.842
  [6] -9.211
  [7] -6.579
  [8] -3.947
  [9] -1.316
```

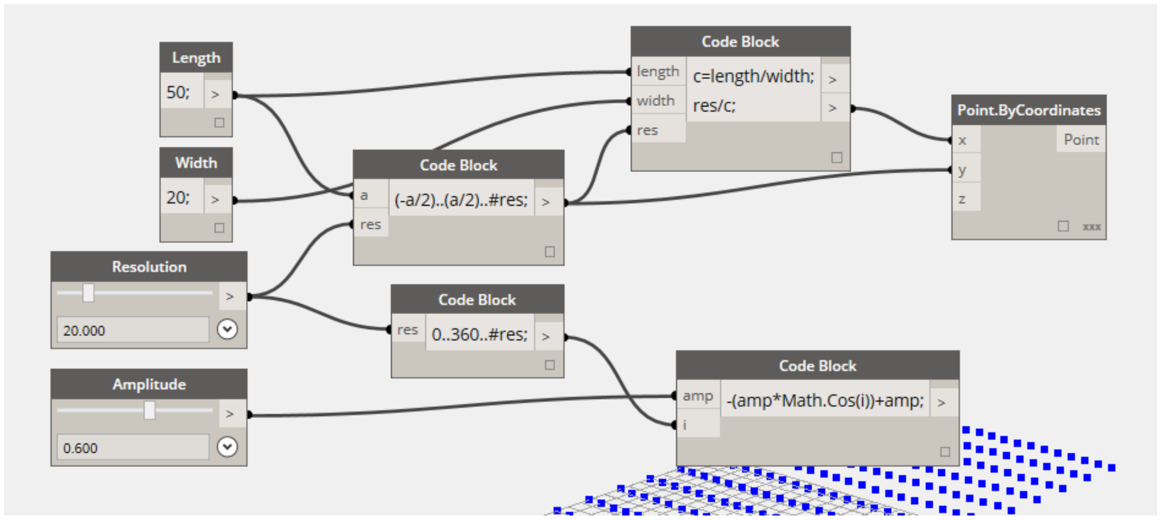
Mathematics

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.

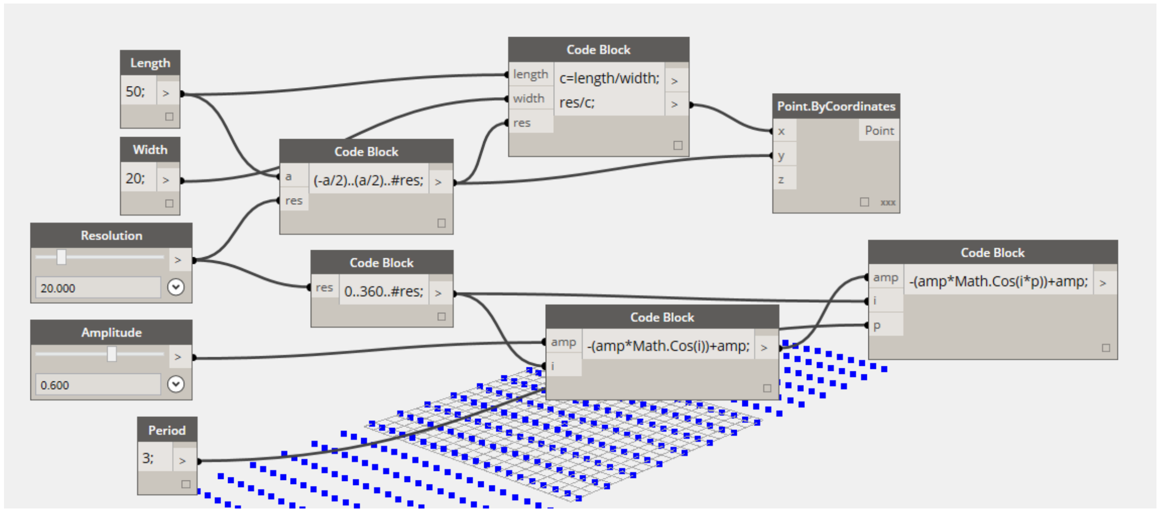


3. Make another Code Block with the same math as the previous (ctrl+c & v), only change i to i*p:

-(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.



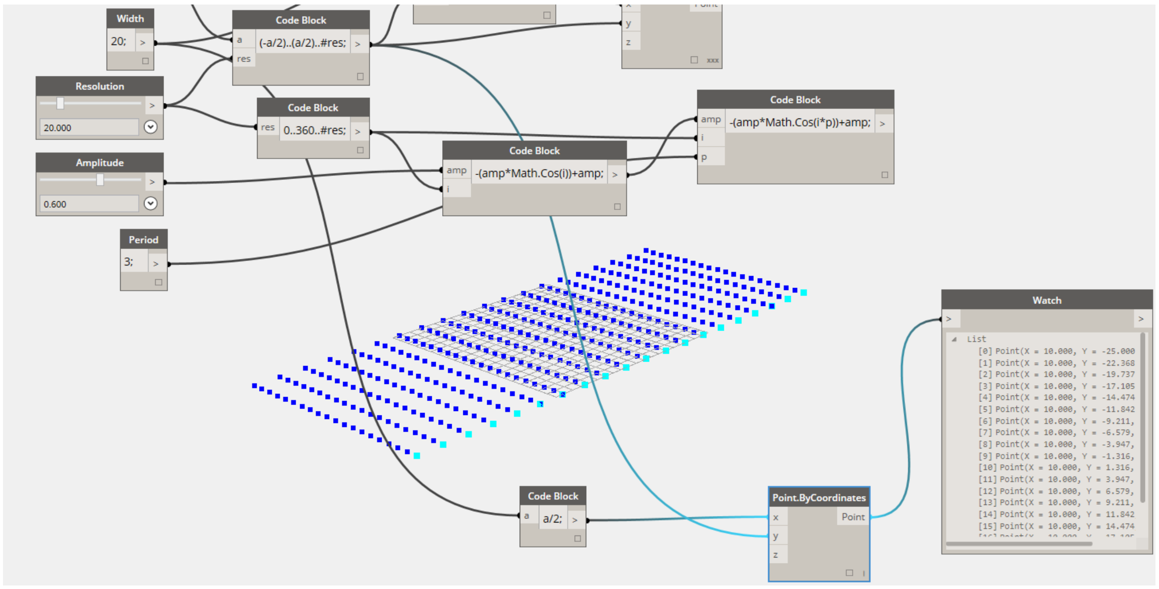
Mathematics illustrated

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

1. Add a Point.ByCoordinates node.
2. Add a Code Block with syntax

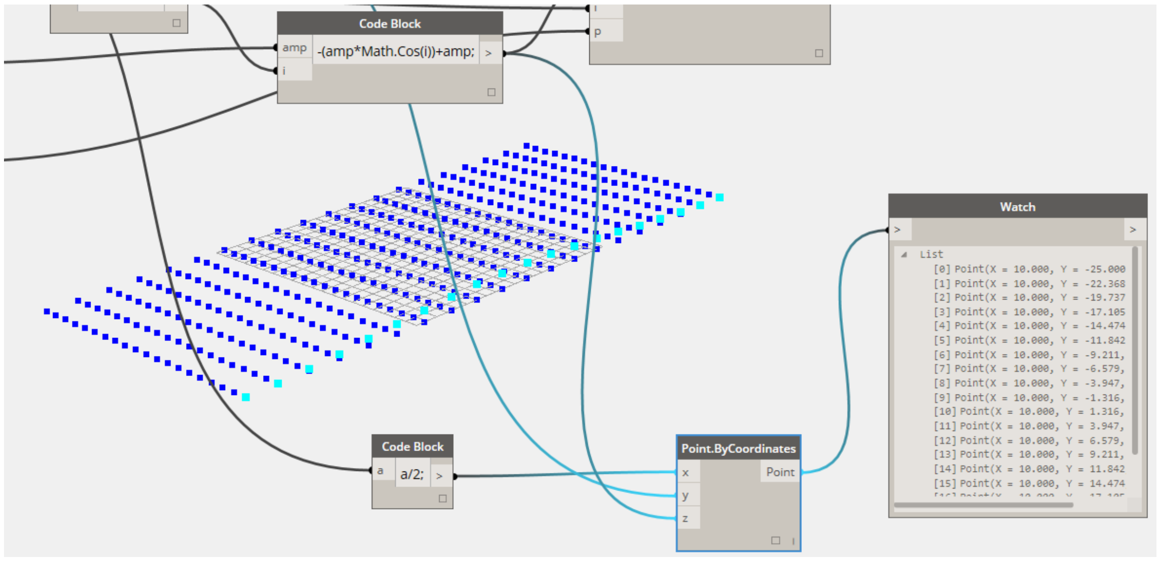
   a/2;

3. Wire Width into the new Code Block, and the resolution output to y, like so:



```
List
  [0] Point(X = 10.000, Y = -25.000
  [1] Point(X = 10.000, Y = -22.368
  [2] Point(X = 10.000, Y = -19.737
  [3] Point(X = 10.000, Y = -17.105
  [4] Point(X = 10.000, Y = -14.474
  [5] Point(X = 10.000, Y = -11.842
  [6] Point(X = 10.000, Y = -9.211,
  [7] Point(X = 10.000, Y = -6.579,
  [8] Point(X = 10.000, Y = -3.947,
  [9] Point(X = 10.000, Y = -1.316,
  [10] Point(X = 10.000, Y = 1.316,
  [11] Point(X = 10.000, Y = 3.947,
  [12] Point(X = 10.000, Y = 6.579,
  [13] Point(X = 10.000, Y = 9.211,
  [14] Point(X = 10.000, Y = 11.842
  [15] Point(X = 10.000, Y = 14.474
```

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

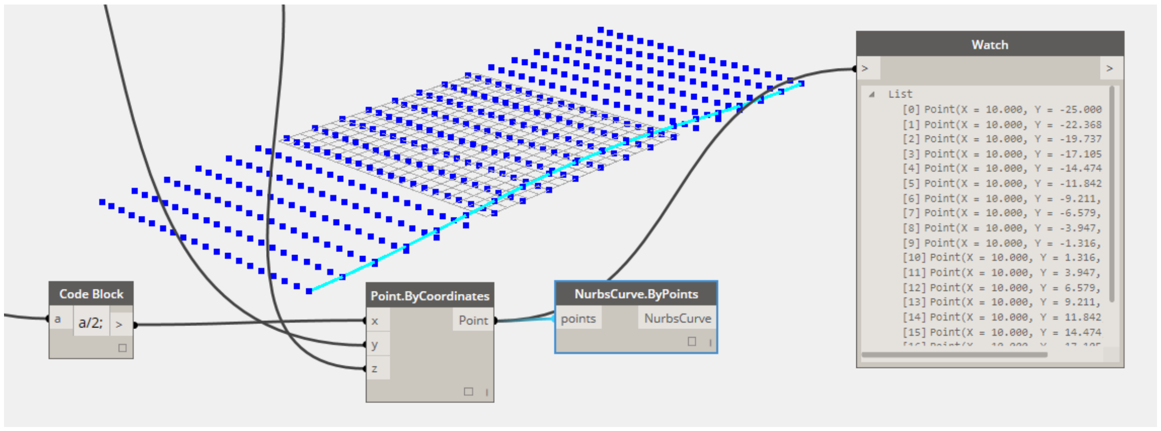


```
-(amp*Math.Cos(i))+amp;
```

```
a/2;
```

```
List
  [0] Point(X = 10.000, Y = -25.000
  [1] Point(X = 10.000, Y = -22.368
  [2] Point(X = 10.000, Y = -19.737
  [3] Point(X = 10.000, Y = -17.105
  [4] Point(X = 10.000, Y = -14.474
  [5] Point(X = 10.000, Y = -11.842
  [6] Point(X = 10.000, Y = -9.211,
  [7] Point(X = 10.000, Y = -6.579,
  [8] Point(X = 10.000, Y = -3.947,
  [9] Point(X = 10.000, Y = -1.316,
  [10] Point(X = 10.000, Y = 1.316,
  [11] Point(X = 10.000, Y = 3.947,
  [12] Point(X = 10.000, Y = 6.579,
  [13] Point(X = 10.000, Y = 9.211,
  [14] Point(X = 10.000, Y = 11.842
  [15] Point(X = 10.000, Y = 14.474
```

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



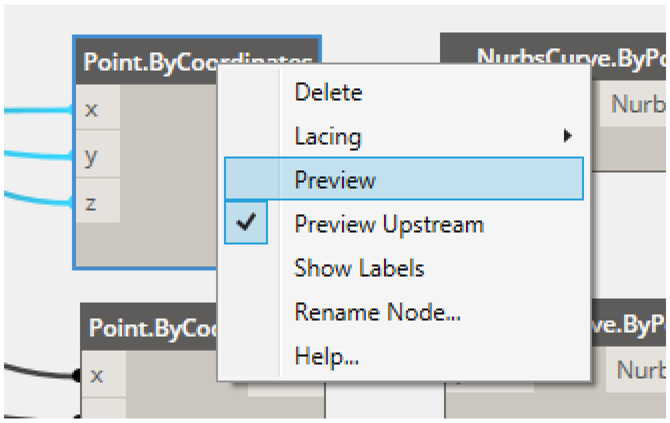
```
a/2;
```

```
List
  [0] Point(X = 10.000, Y = -25.000
  [1] Point(X = 10.000, Y = -22.368
  [2] Point(X = 10.000, Y = -19.737
  [3] Point(X = 10.000, Y = -17.105
  [4] Point(X = 10.000, Y = -14.474
  [5] Point(X = 10.000, Y = -11.842
  [6] Point(X = 10.000, Y = -9.211,
  [7] Point(X = 10.000, Y = -6.579,
  [8] Point(X = 10.000, Y = -3.947,
  [9] Point(X = 10.000, Y = -1.316,
  [10] Point(X = 10.000, Y = 1.316,
  [11] Point(X = 10.000, Y = 3.947,
  [12] Point(X = 10.000, Y = 6.579,
  [13] Point(X = 10.000, Y = 9.211,
  [14] Point(X = 10.000, Y = 11.842
  [15] Point(X = 10.000, Y = 14.474
```

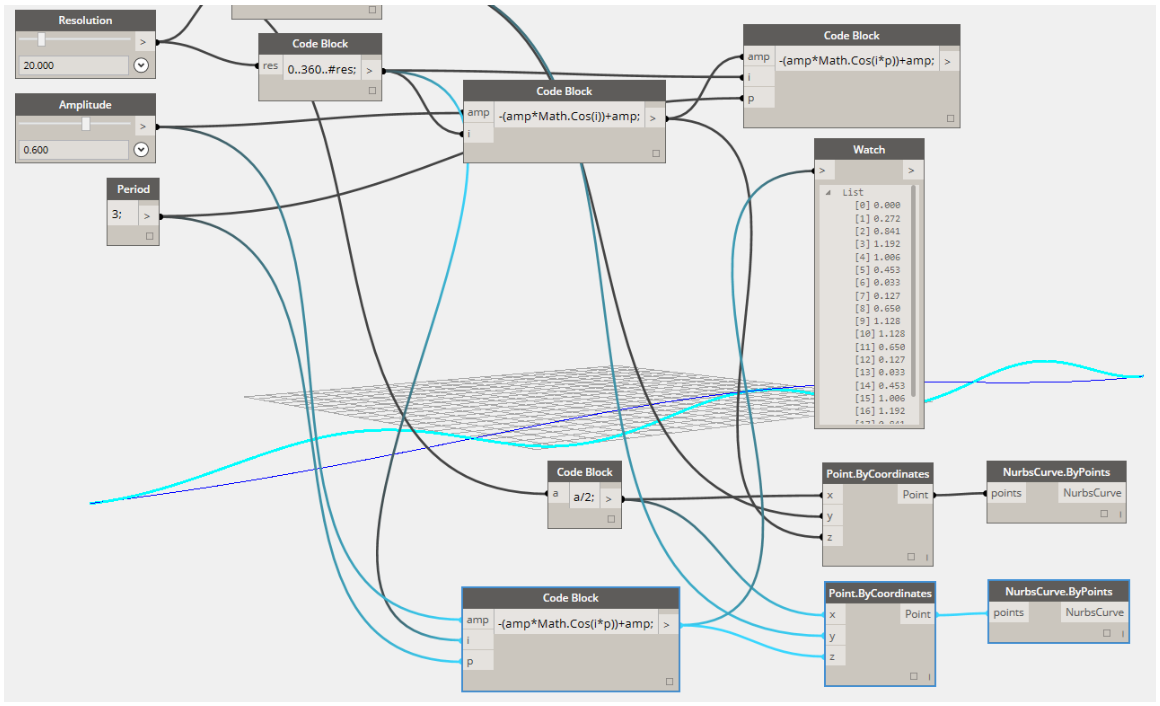
What we see now is a beautiful cosine graph between 0 and 360, divided at a resolution of 20. This is the outcome of the first trigonometry function.

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.

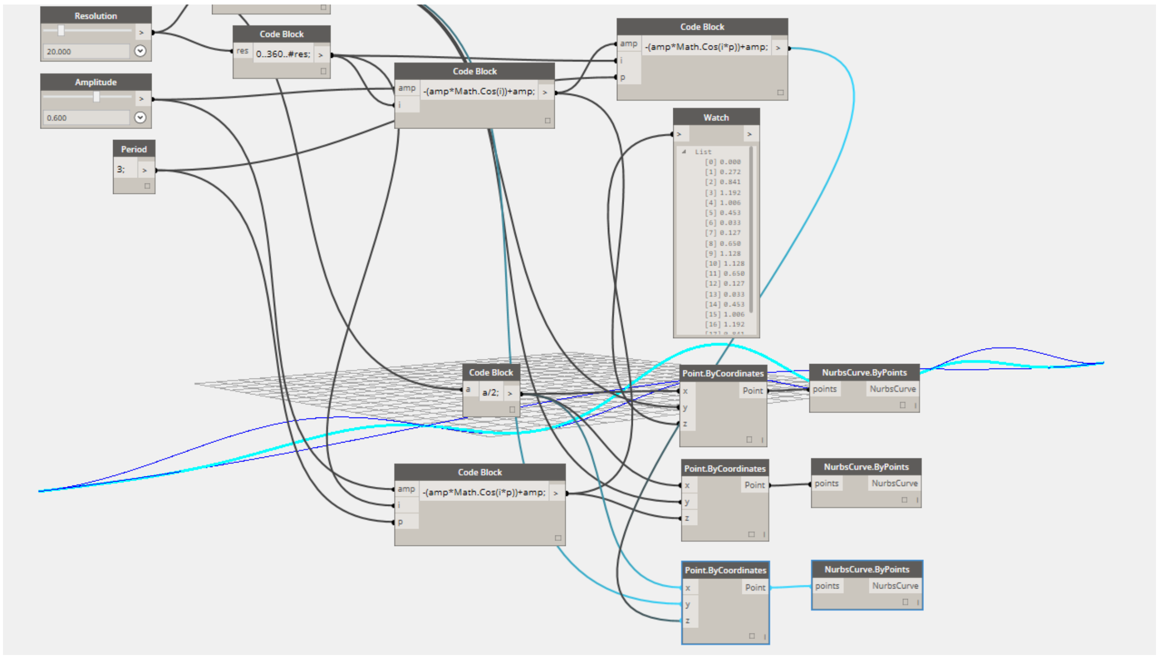


```
3;
0..360..#res;
-(amp*Math.Cos(i))+amp;
-(amp*Math.Cos(i*p))+amp;
a/2;
```

```
List
  [0] 0.000
  [1] 0.272
  [2] 0.841
  [3] 1.192
  [4] 1.006
  [5] 0.453
  [6] 0.033
  [7] 0.127
  [8] 0.650
  [9] 1.128
  [10] 1.128
  [11] 0.650
  [12] 0.127
  [13] 0.033
  [14] 0.453
  [15] 1.006
  [16] 1.192
```

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.



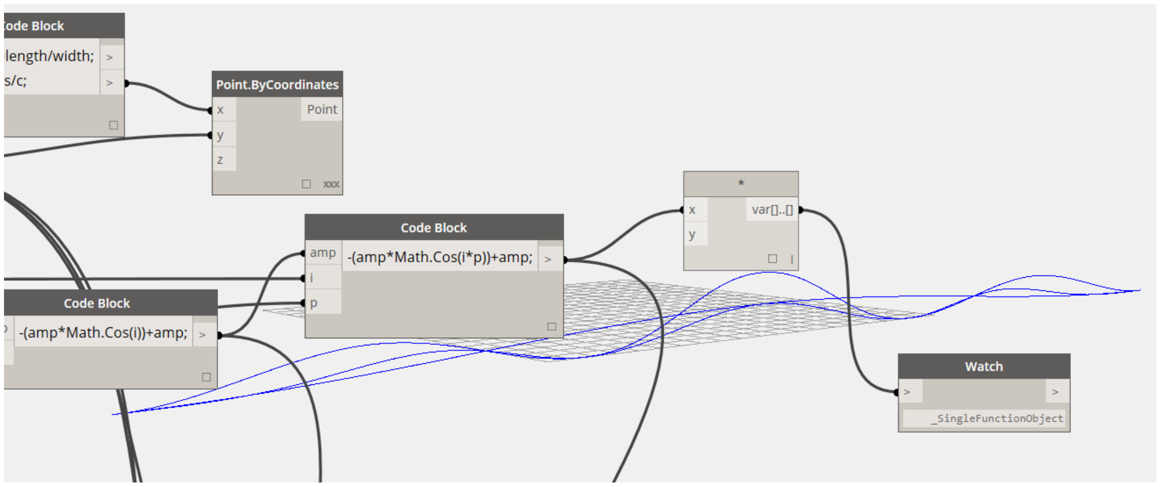
```
List
  [0] 0.000
  [1] 0.272
  [2] 0.841
  [3] 1.192
  [4] 1.006
  [5] 0.453
  [6] 0.033
  [7] 0.127
  [8] 0.650
  [9] 1.128
  [10] 1.128
  [11] 0.650
  [12] 0.127
  [13] 0.033
  [14] 0.453
  [15] 1.006
  [16] 1.192
```

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

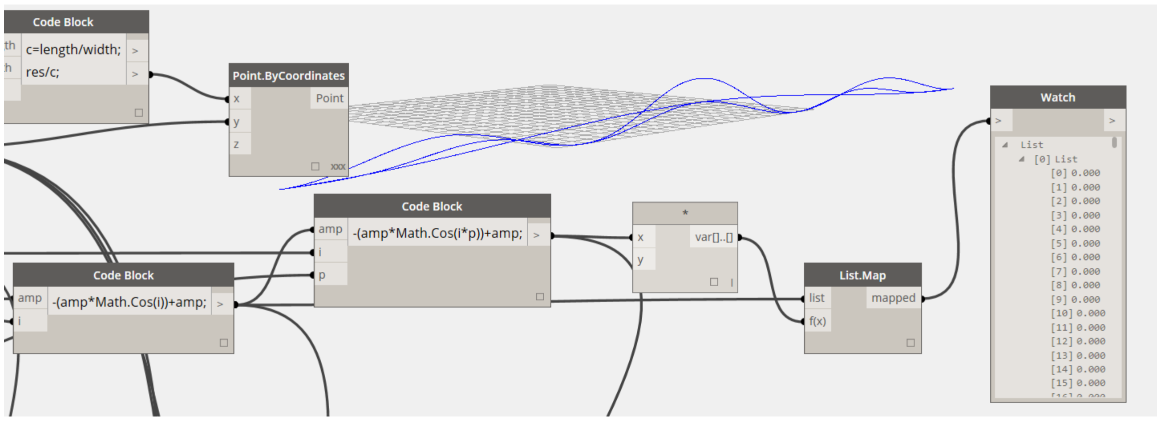
Surface

Let's turn this beautiful math into a surface.

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

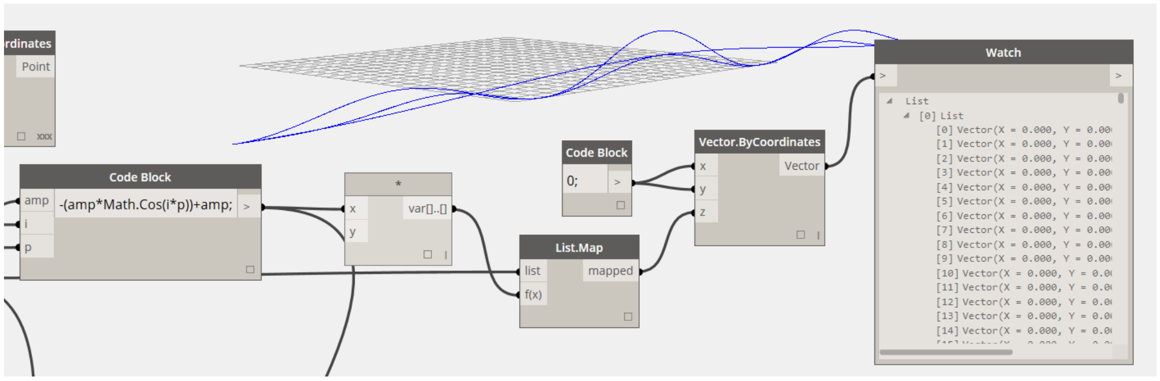


2. Use a List.Map to connect the first and second (last) trig functions in a new list of numbers.



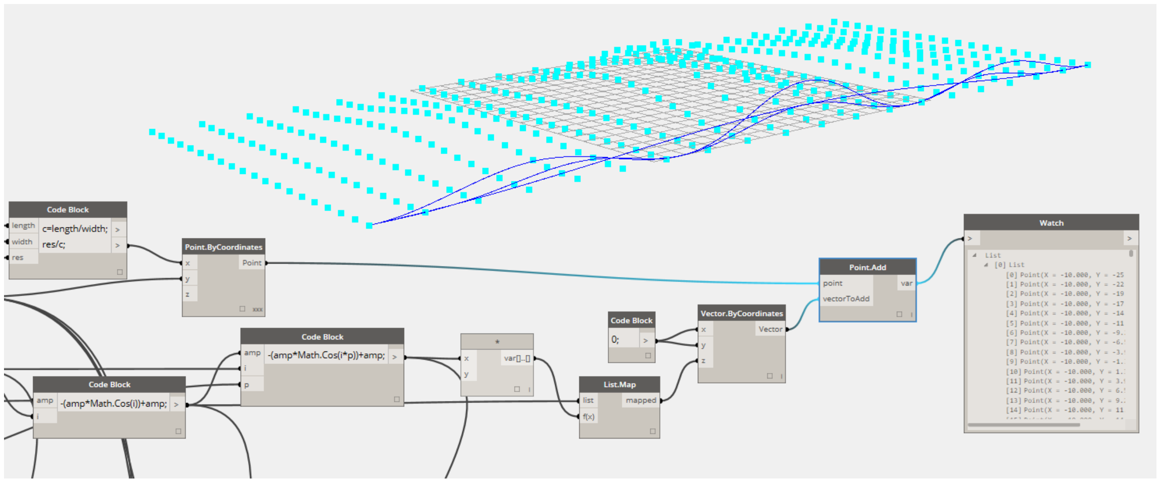
```
List
  [0] List
    [0] 0.000
    [1] 0.000
    [2] 0.000
    [3] 0.000
    [4] 0.000
    [5] 0.000
    [6] 0.000
    [7] 0.000
    [8] 0.000
    [9] 0.000
    [10] 0.000
    [11] 0.000
    [12] 0.000
    [13] 0.000
    [14] 0.000
    [15] 0.000
```

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.



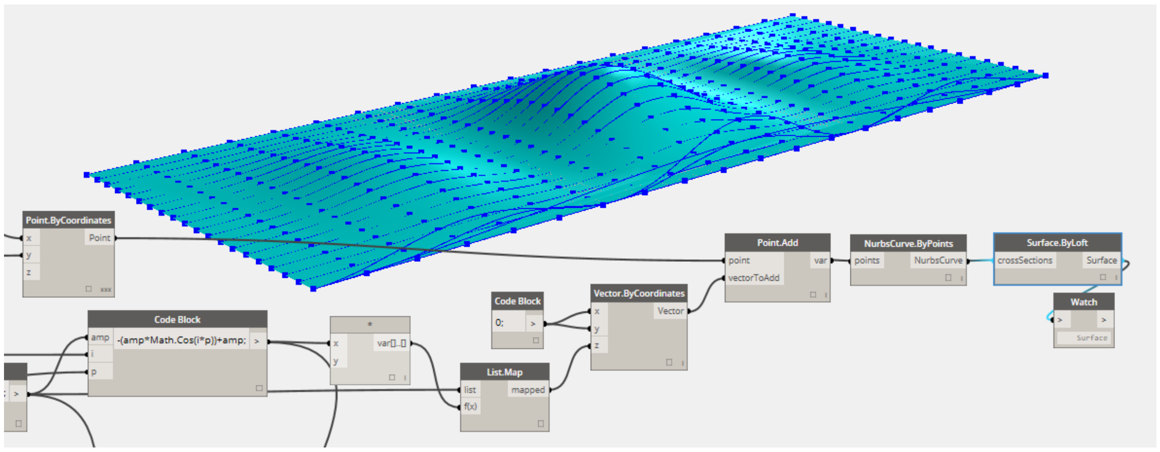
```
List
  [0] List
    [0] Vector(X = 0.000, Y = 0.00
    [1] Vector(X = 0.000, Y = 0.00
    [2] Vector(X = 0.000, Y = 0.00
    [3] Vector(X = 0.000, Y = 0.00
    [4] Vector(X = 0.000, Y = 0.00
    [5] Vector(X = 0.000, Y = 0.00
    [6] Vector(X = 0.000, Y = 0.00
    [7] Vector(X = 0.000, Y = 0.00
    [8] Vector(X = 0.000, Y = 0.00
    [9] Vector(X = 0.000, Y = 0.00
    [10] Vector(X = 0.000, Y = 0.0
    [11] Vector(X = 0.000, Y = 0.0
    [12] Vector(X = 0.000, Y = 0.0
    [13] Vector(X = 0.000, Y = 0.0
    [14] Vector(X = 0.000, Y = 0.0
```

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



```
List
  [0] List
    [0] Point(X = -10.000, Y = -25
    [1] Point(X = -10.000, Y = -22
    [2] Point(X = -10.000, Y = -19
    [3] Point(X = -10.000, Y = -17
    [4] Point(X = -10.000, Y = -14
    [5] Point(X = -10.000, Y = -11
    [6] Point(X = -10.000, Y = -9.
    [7] Point(X = -10.000, Y = -6.
    [8] Point(X = -10.000, Y = -3.
    [9] Point(X = -10.000, Y = -1.
    [10] Point(X = -10.000, Y = 1.
    [11] Point(X = -10.000, Y = 3.
    [12] Point(X = -10.000, Y = 6.
    [13] Point(X = -10.000, Y = 9.
    [14] Point(X = -10.000, Y = 11
```

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.

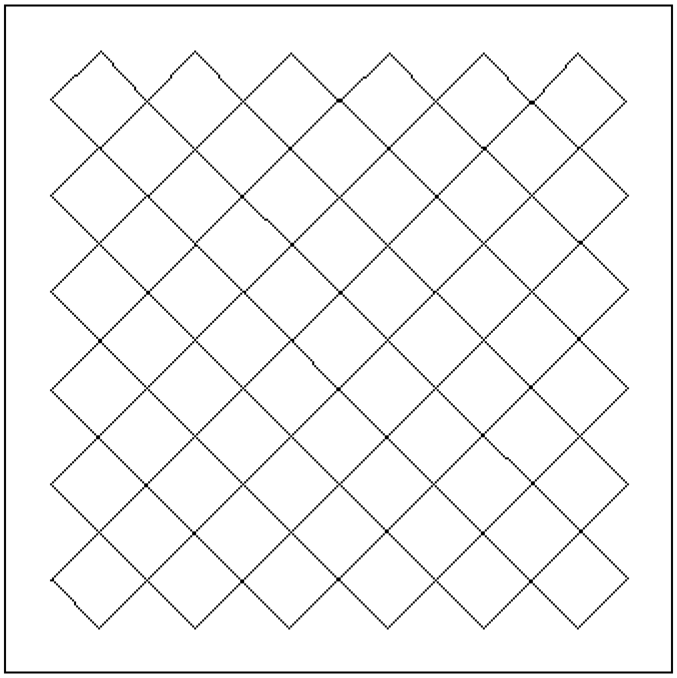
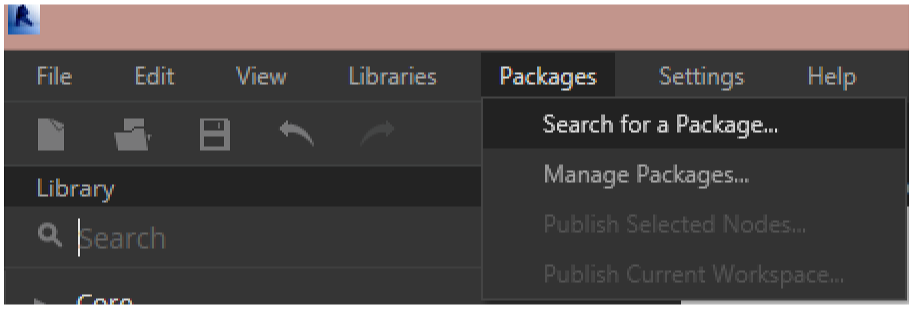


Figure 1: Diamond Grid according to Google Images

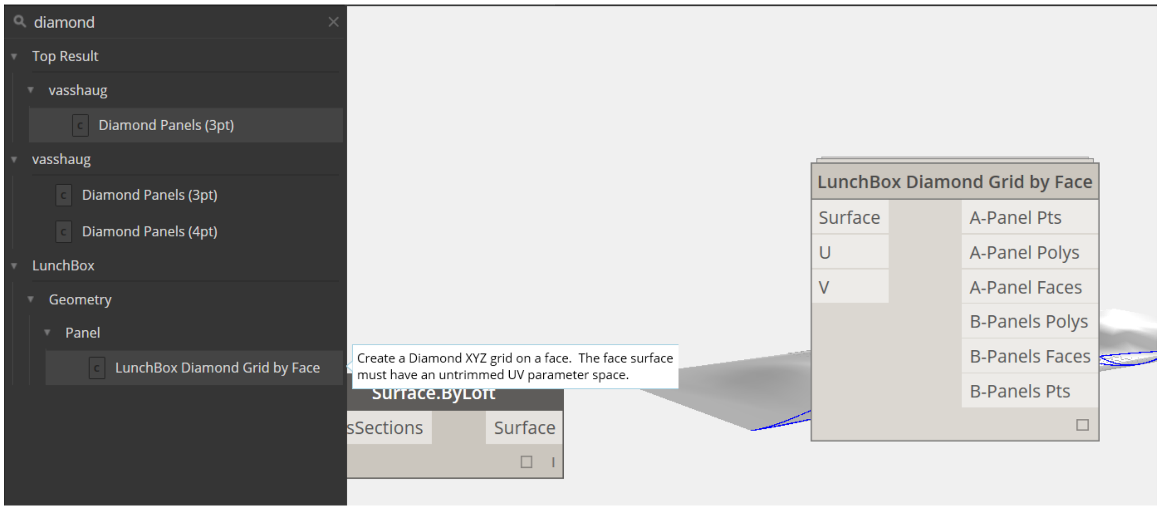
A diamond grid has 3-point panels along the edges and 4-point panels on the inside. To create these panels we need a custom node called LunchBox Diamond Grid by Face by Nathan Miller.

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



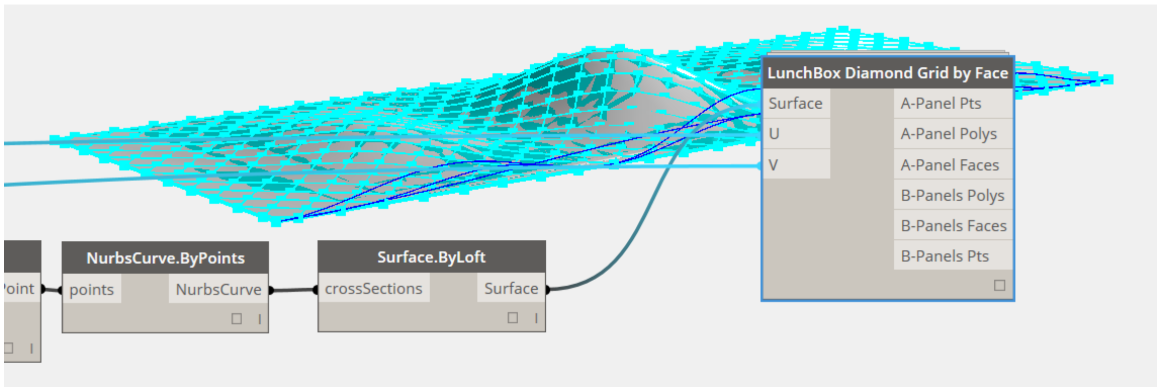
7. Search for “LunchBox”, and click Install.

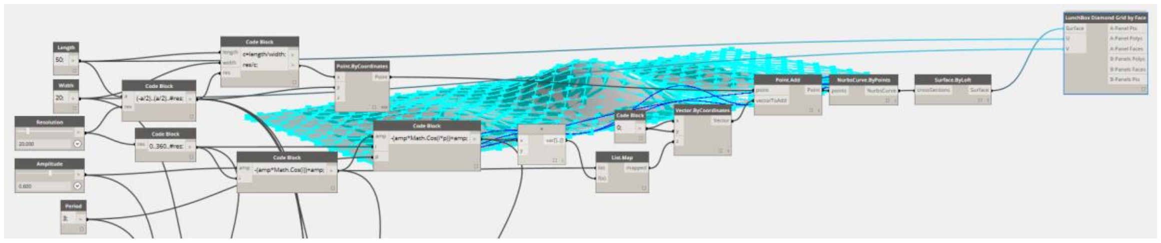
8. In our Dynamo search field, use the same keyword and add the custom node Diamond Grid by Face to our canvas.



Create a Diamond XYZ grid on a face. The face surface must have an untrimmed UV parameter space.

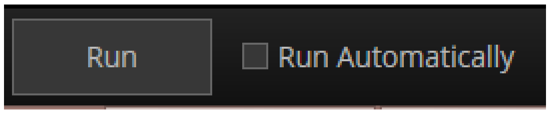
9. Wire the diamond node to the surface, Length (U) and Width (V) outputs.



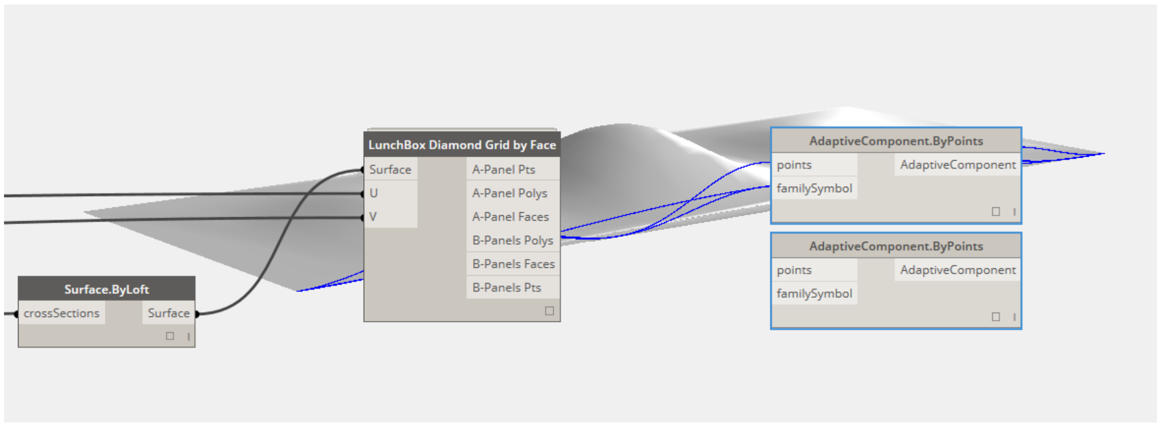


10. Now right click all nodes that generate point or line geometry in the preview background, including the panel nodes, and deselect Preview.

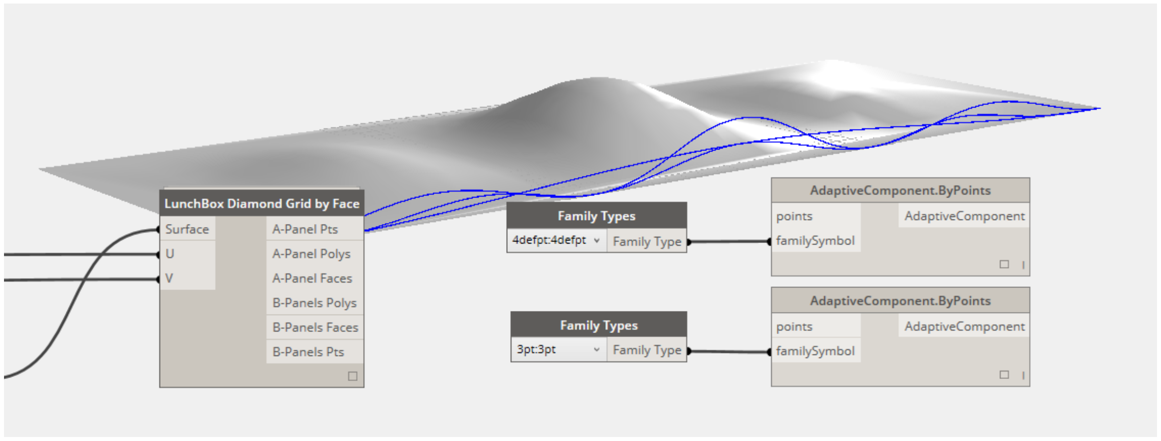
11. De-select Run Automatically.



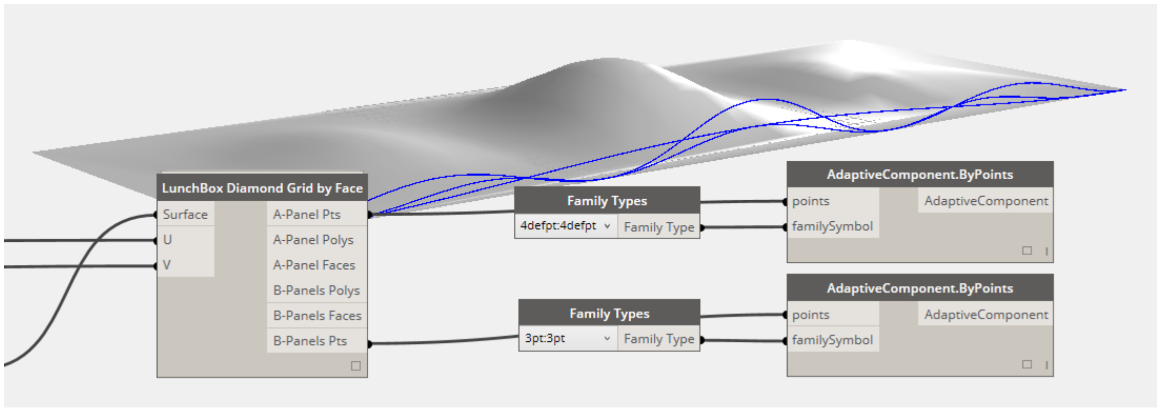
12. Add two AdaptiveComponent.ByPoints nodes.



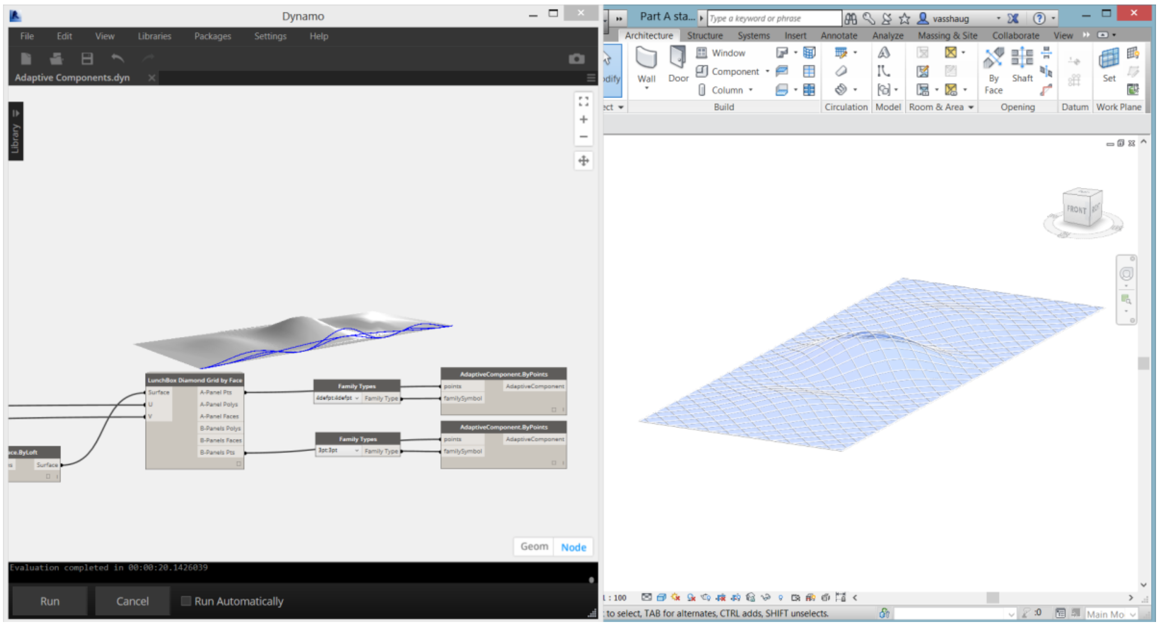
13. Add two Family Types nodes. Select the families 3pt and 4ptdef, and wire each to its own Adaptive Component node.



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:


Dynamo
File Edit View Libraries Packages Settings Help
Adaptive Components.dyn
LunchBox Diamond Grid by Face
Family Types
AdaptiveComponent.ByPoints
Geom Node
Run
Cancel
Run Automatically
Architecture Structure Systems Insert Annotate Analyze Massing & Site Collaborate View

```
Evaluation completed in 00:00:20.1426039
```

Congratulations! You've made a perfectly mathematically defined double curved cosine surface of native Revit elements using Dynamo.

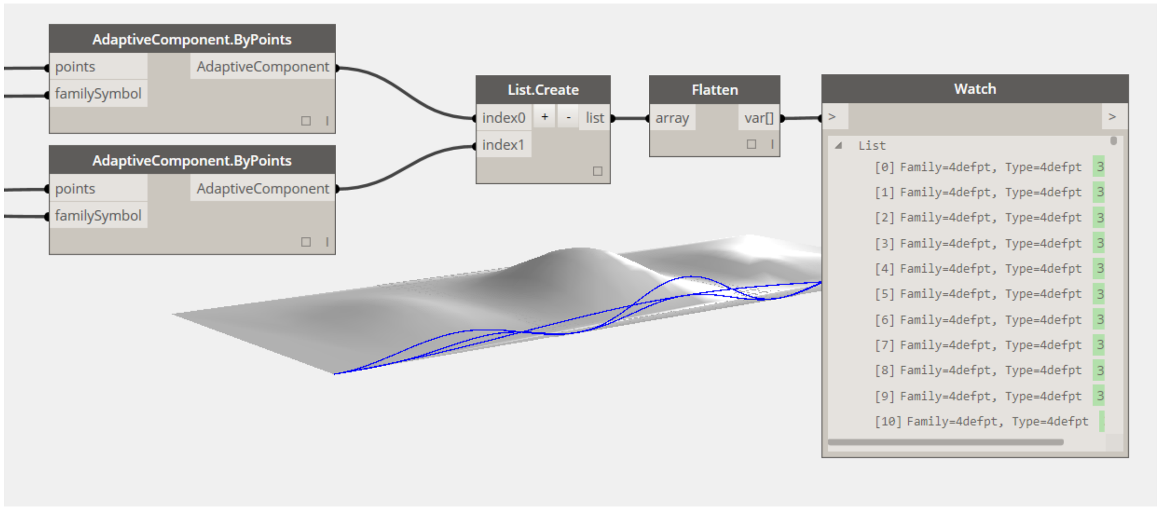
There are a couple of simple operations we can utilize on this model to get some ideas on shapes, sizes and constructability (cost). One of them is colorizing ranges of parameters in a view.

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).

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

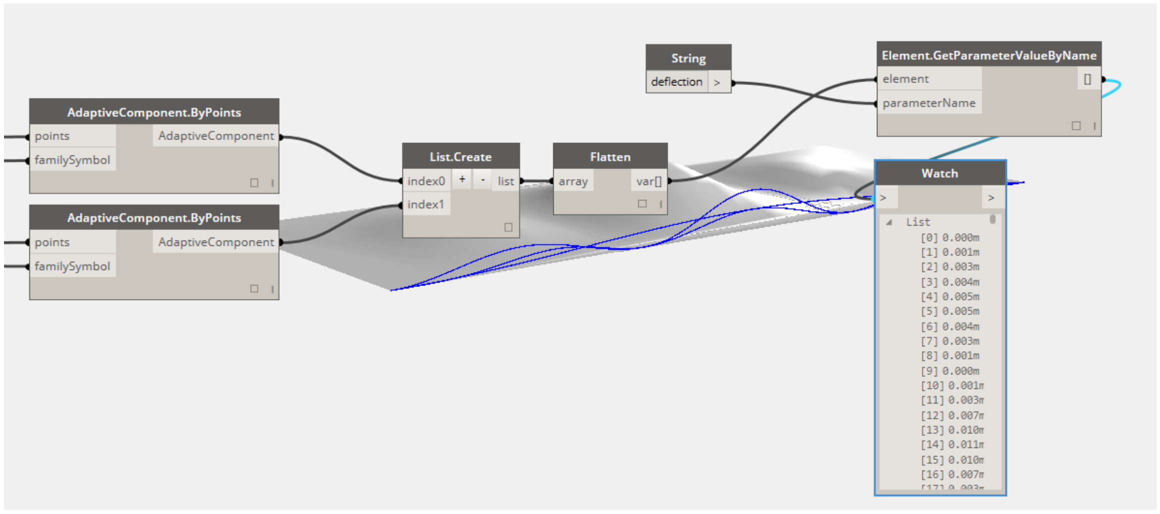
1. Add a List.Create with two inputs and a Flatten node to pull together all Adaptive Components into one list.



```
List
  [0] Family=4defpt, Type=4defpt  3
  [1] Family=4defpt, Type=4defpt  3
  [2] Family=4defpt, Type=4defpt  3
  [3] Family=4defpt, Type=4defpt  3
  [4] Family=4defpt, Type=4defpt  3
  [5] Family=4defpt, Type=4defpt  3
  [6] Family=4defpt, Type=4defpt  3
  [7] Family=4defpt, Type=4defpt  3
  [8] Family=4defpt, Type=4defpt  3
  [9] Family=4defpt, Type=4defpt  3
  [10] Family=4defpt, Type=4defpt
```

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

   syntax
   deflection



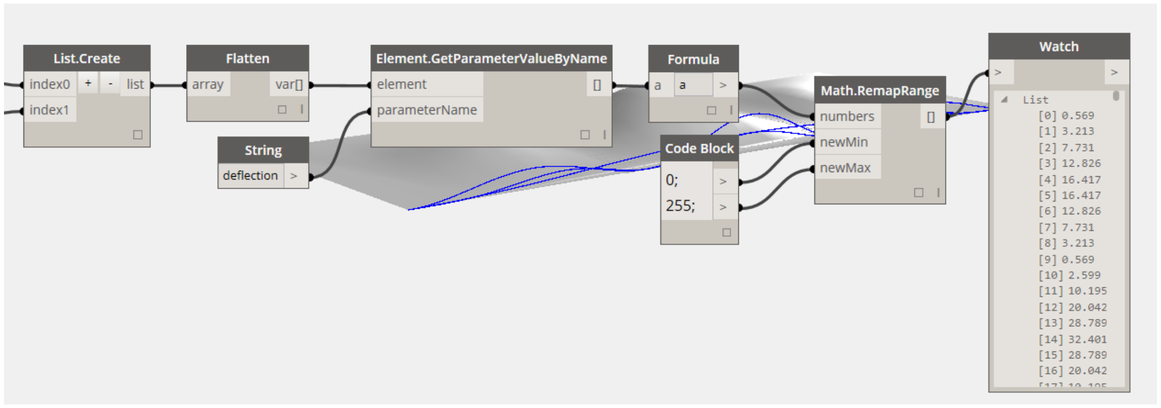
```
List
  [0] 0.000m
  [1] 0.001m
  [2] 0.003m
  [3] 0.004m
  [4] 0.005m
  [5] 0.005m
  [6] 0.004m
  [7] 0.003m
  [8] 0.001m
  [9] 0.000m
  [10] 0.001m
  [11] 0.003m
  [12] 0.007m
  [13] 0.010m
  [14] 0.011m
  [15] 0.010m
  [16] 0.007m
  [17] 0.003m
```

3. Pull out a Math.RemapRange node, and a Code Block with syntax

   0;
   255;

4. Add a formula node with syntax

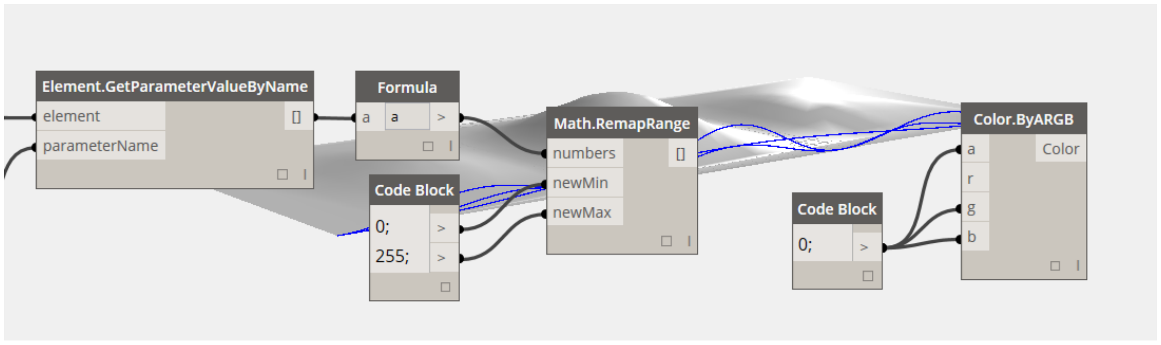
   a



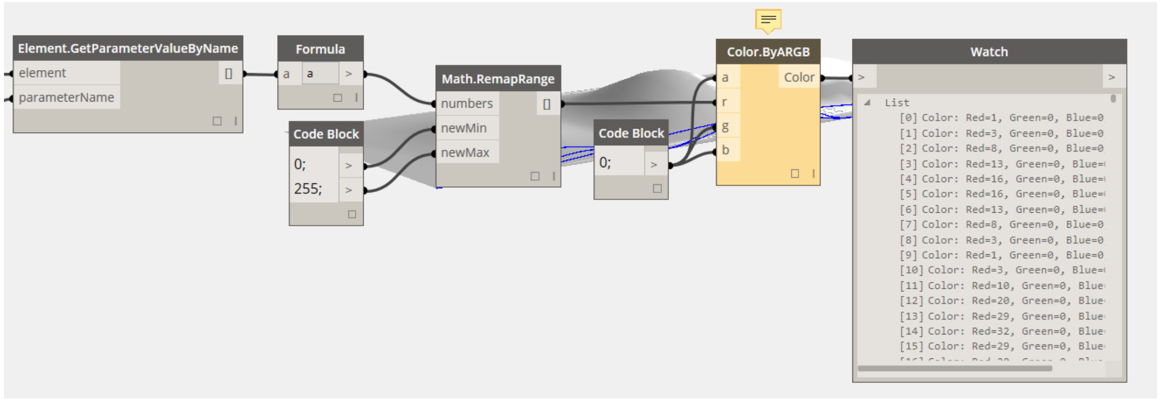
```
List
  [0] 0.569
  [1] 3.213
  [2] 7.731
  [3] 12.826
  [4] 16.417
  [5] 16.417
  [6] 12.826
  [7] 7.731
  [8] 3.213
  [9] 0.569
  [10] 2.599
  [11] 10.195
  [12] 20.042
  [13] 28.789
  [14] 32.401
  [15] 28.789
  [16] 20.042
```

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.



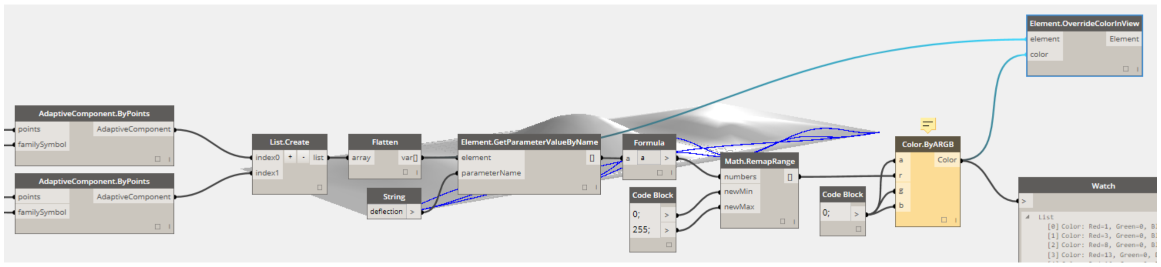
6. Wire the remapped numbers to the r input.



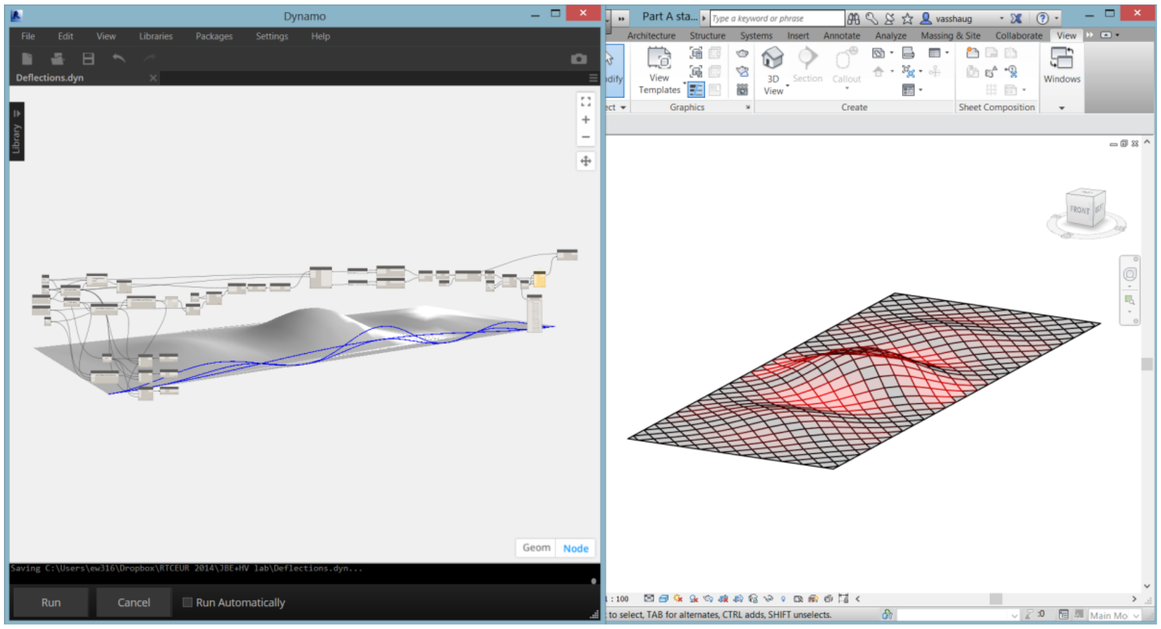
```
List
  [0] Color: Red=1, Green=0, Blue=0
  [1] Color: Red=3, Green=0, Blue=0
  [2] Color: Red=8, Green=0, Blue=0
  [3] Color: Red=13, Green=0, Blue=
  [4] Color: Red=16, Green=0, Blue=
  [5] Color: Red=16, Green=0, Blue=
  [6] Color: Red=13, Green=0, Blue=
  [7] Color: Red=8, Green=0, Blue=0
  [8] Color: Red=3, Green=0, Blue=0
  [9] Color: Red=1, Green=0, Blue=0
  [10] Color: Red=3, Green=0, Blue=
  [11] Color: Red=10, Green=0, Blue
  [12] Color: Red=20, Green=0, Blue
  [13] Color: Red=29, Green=0, Blue
  [14] Color: Red=32, Green=0, Blue
  [15] Color: Red=29, Green=0, Blue
```

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

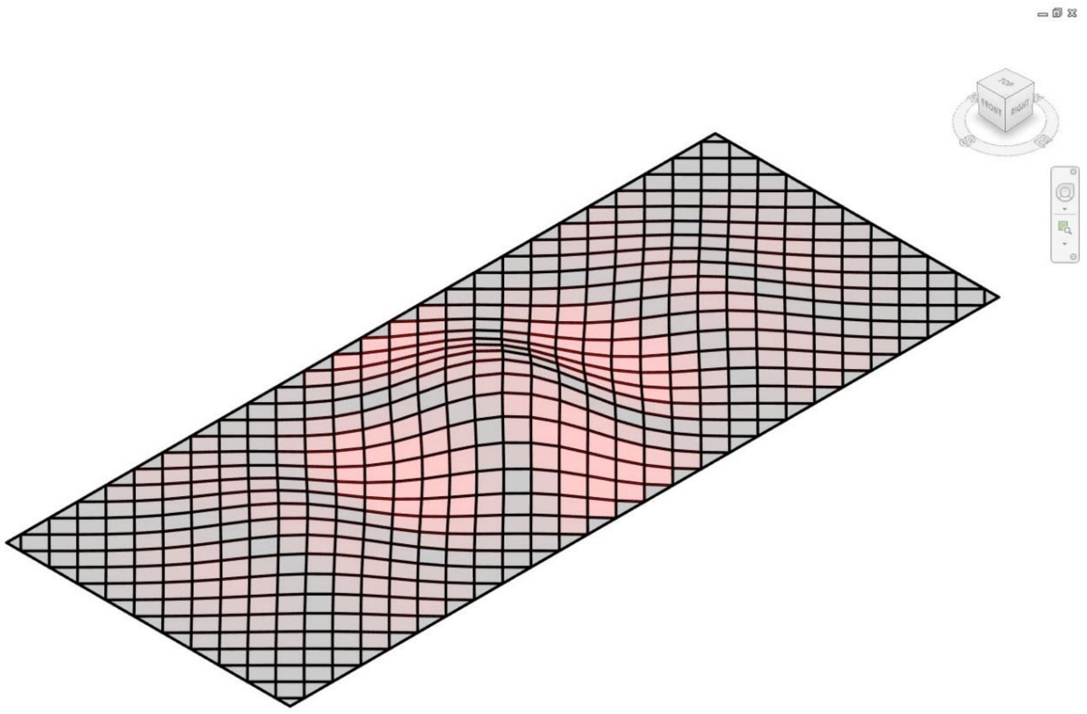
7. Add an Element.OverrideColorInView node and wire it to the collected Adaptive Components list and the color output.

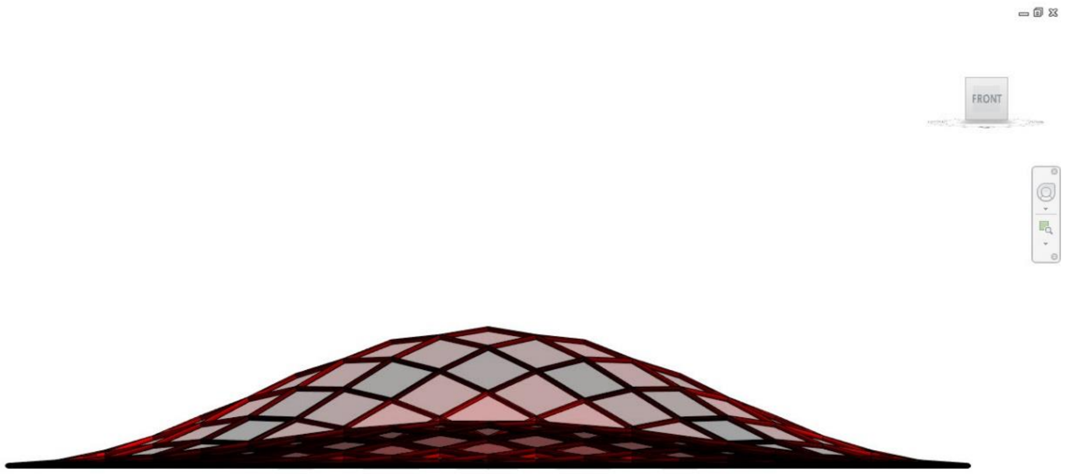


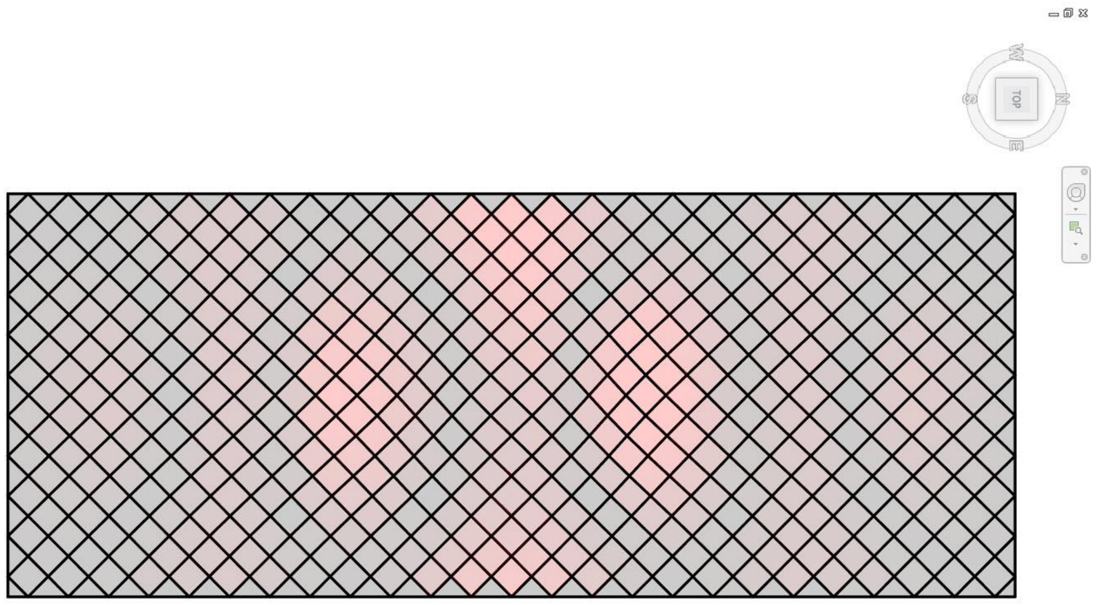
8. Press Run, and check out the open Revit view.

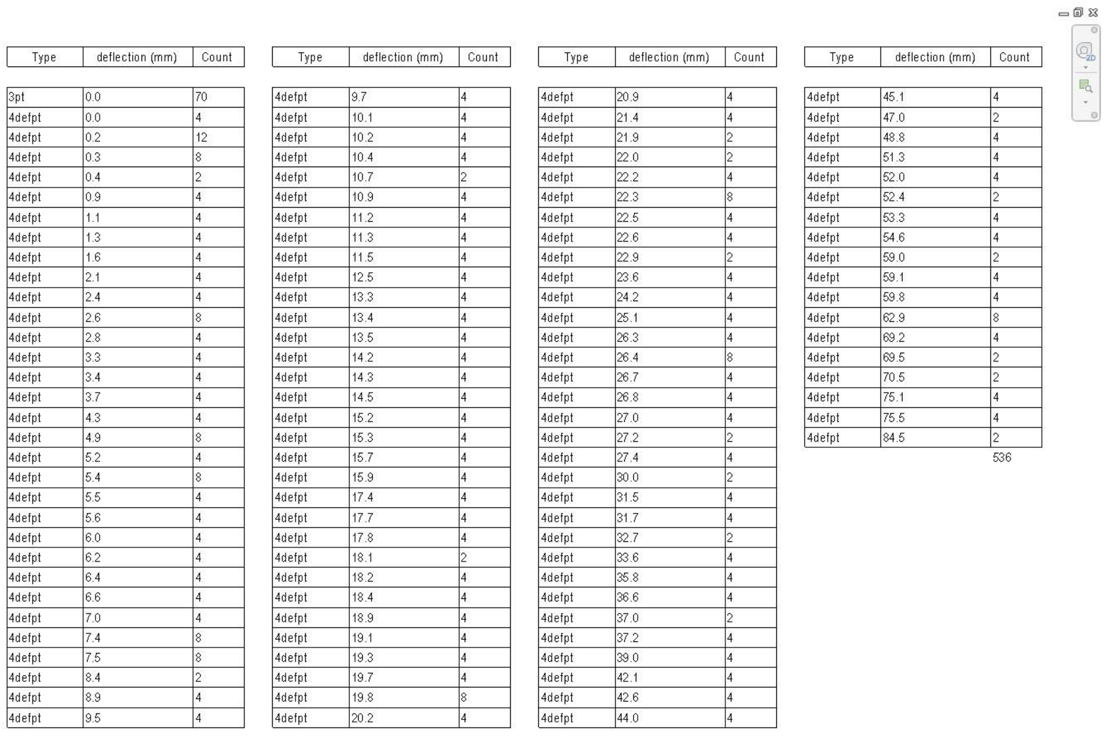


With this information we can produce, analyze and communicate constructability and cost at a visual and informative level that can help clients and contractors to understand what's easy and what's difficult.

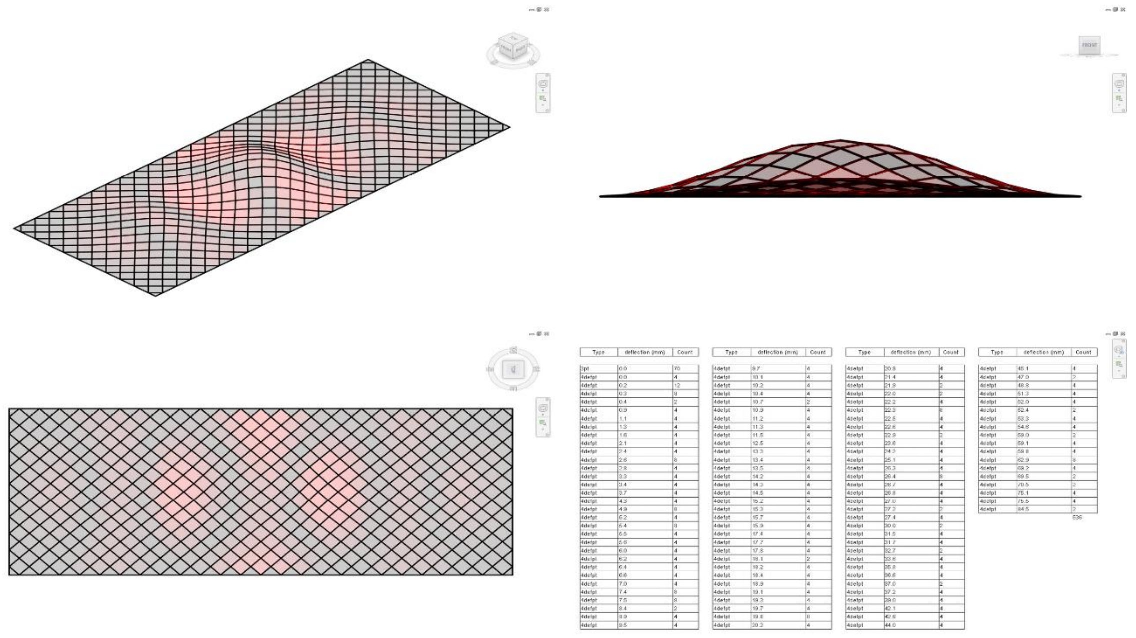

TOP
FRONT
RIGHT


FRONT



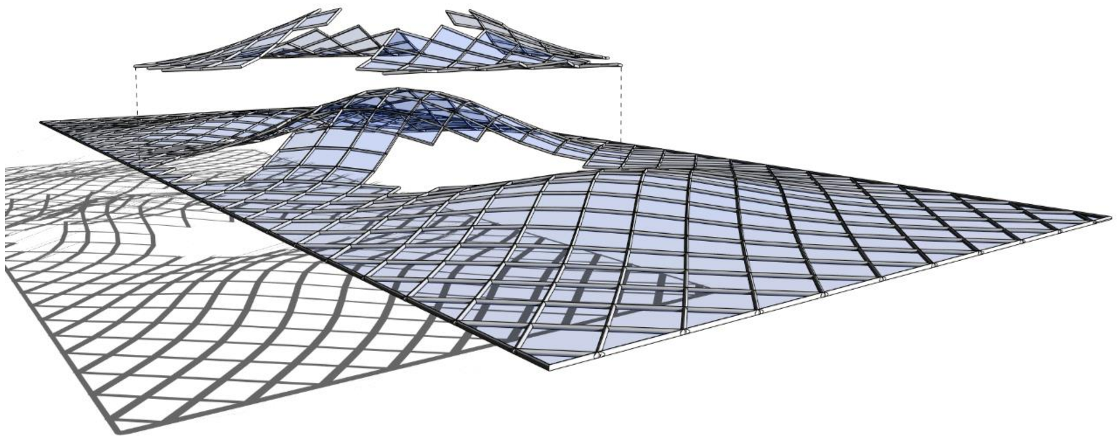


Type	deflection (mm)	Count
3pt	0.0	70
4defpt	0.0	4
4defpt	0.2	12
4defpt	0.3	8
4defpt	0.4	2
4defpt	0.9	4
4defpt	1.1	4
4defpt	1.3	4
4defpt	1.6	4
4defpt	2.1	4
4defpt	2.4	4
4defpt	2.6	8
4defpt	2.8	4
4defpt	3.3	4
4defpt	3.4	4
4defpt	3.7	4
4defpt	4.3	4
4defpt	4.9	8
4defpt	5.2	4
4defpt	5.4	8
4defpt	5.5	4
4defpt	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
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
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
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



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
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
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
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	54.8	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
		636

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. This is one example. Now, make your own and share it with the world!