Digital Built Week Europe 2019
BILT Europe 2019

EICC, Edinburgh
10 – 12 October 2019

Session 2.2

Visual Programming in Infrastructure: Gung-ho Tactics for Getting Stuff Done

Jostein Berger Olsen, Bad Monkeys

Class Description

The introduction of BIM in Infrastructure has been, and still is, a bumpy ride. Adding to it, Infrastructure projects are complex and faceted in its very nature. With visual programming tools you as an engineer, modeller or drafter get a new set of tools that can be applied to a range of different production challenges for consultants, owners and contractors alike. This lab will look into how we can parametrically drive Infrastructure BIM-models using visual programming tools. We'll cover some basics, but also explore aspects of geometry, math and data handling that in sum can more or less directly be applied to live projects.

So, if you're an Infrastructure BIM'mer come attend this lab, and learn how things can be done better and faster than ever before!

Jostein Berger Olsen, Bad Monkeys Bad Monkeys

About the Speaker:

Jostein Olsen is a structural engineer, turned visual programming specialist, now working for the Bad Monkeys. Before joining the monkeys, he has gathered 10 years of experience in the AEC industry working on everything from small scale industrial projects to large infrastructure projects. He has a wide experience with practical BIM workflows and has dug deep into the wonderful world of Dynamo, on which he has his own blog jbdynamo.blogspot.com. He has a passion for finding the practical middle ground between the BIM-idealists and the more conservative forces in the industry and thinks parametric modelling software and computational thinking can help close that gap.

Table of Contents

1.0 Introduction

Welcome to this handout. If you are reading this it will presumably be because you work with Infrastructure projects and you want to do stuff with Visual programming. Good! Then you're in the right place, hopefully..

I did an entire session last year on why the need for visual programming in Infrastructure and if you want to read it you can find it here:
http://jbdynamo.blogspot.com/2018/10/bilt-eur-2018-computational-thinking-in.html
Basically, what I'm stating is that Infrastructure projects are especially prone for using Visual Programming software because of the projects' complexities, the lack of proper modelling software (at least for some disciplines) and that the numbers show us that we're not becoming all that much more efficient, on the contrary. So, Infrastructure is very ripe for Visual Programming, &&/|| programming in general. And fantastic examples have been put forth by many key players in the industry. My goal though is NOT to show you fancy examples and scare you into using visual programming, no, my goal is to actually teach you something. And hopefully, this handout and session at BILT EUR 2019 will give you some direct and concrete examples that will inspire and you and provide you with something that you can apply straight away in your projects.

As a little something extra, tidbits of fun facts or tips & tricks are strewn about this handout. If that is all you are after, look for text in green! 😊

1.1 A Little Note on Software

The handout and lab-session will be centred around Revit and Dynamo. I know for sure that isn't necessarily the best combination, and certainly not the only one, for Infrastructure projects.
Grasshopper/Rhino for instance are very good competitors with their connections to Tekla etc.
Common for both of them is that they are not originally infrastructure tools, and by that, I mean, they are not built for linear elements. Revit and Tekla for instance were created to be used with grids and levels. Not with superelevations, corridors and alignments. Moreover, these tools have led the way in using BIM for construction and now we see that more and more infrastructure projects are also turning their eyes on BIM. Therefore, it is only natural that many BIM power users with experience from BIM-software are moving into infrastructure projects only to find that their software are far from complete

and "infrastructure ready". I think this is one of the main reasons people have been so keen on picking up Visual Programming as well as the sheer scale of modelling efforts needed on these big projects.
Most of the examples can with little translation be used in either so hopefully you won't leave empty handed.

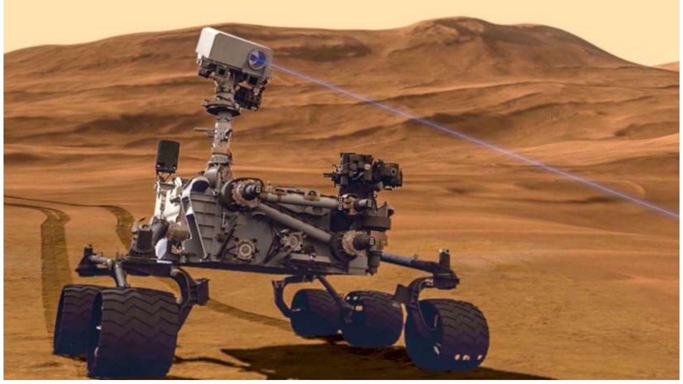
But first, a little something on how we learn.

1.2 How Do We Learn?

I've given a lot of thought to this subject. I've been at numerous workshops, classes, sessions, talks etc without actually learning anything. Especially I find this true on some, I won't say names, branded conferences where it seems people are just in it for showing off. That all fine and dandy, but what is the purpose if we're not learning anything.
So, how do we learn?

Well, I can't speak for anyone else than myself but there are a couple of main drivers that spark my own learning.

1. This fella

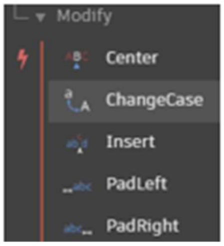


Or not the robot in itself, but its name: Curiosity.
Curiosity is one of the main drivers every time I want to learn something. Why is that so? How does this node work? Why is that node so shiny? (*probably because of IRIS, it's a package, go check it out:* [*https://forum.dynamobim.com/t/i-is-extension-dynamo-2-graph-colorization-customization/35227*](https://forum.dynamobim.com/t/i-is-extension-dynamo-2-graph-colorization-customization/35227))

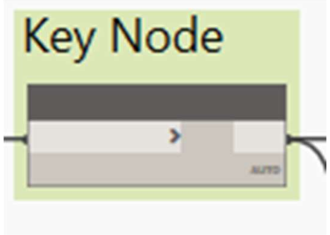
2. Directness
   What you learn must directly apply to what you want to achieve. Want to learn Dynamo or grasshopper? Then figure out what problems you want to solve first and then begin learning with that as a basis.
   (If you want to hear a very good podcast on learning go here: https://gettingsimple.com/scott-young)

3. Getting ideas
   I'm uncertain of the scientific backing on this one, but for me it is almost always the case. I sit in a class at AU, at a course, at BILT and I'm almost falling asleep, then all of a sudden, a lightning bolt of inspiration. It can be a picture, it can be a formula, it can be whatever, but often it's those small things that spark an idea that you can ponder on and work on once you get back home to your office. It's the set off point for your curiosity!

All the three elements above have led me into a "educational path" for this lab. And the concept of key nodes. With key nodes I mean those nodes or small combinations of nodes that together form an idea or that have sparked my curiosity somehow and that are directly correlated to Infrastructure. It is direct. This is what I want to pass on to you hopefully. What I don't believe you learn much from is showing you huge, overly complex graphs. Neither does it create any interest or spark much learning looking for the nodes in the library. That, you can do on your own with this handout in the quiet comfort of your cubicle...

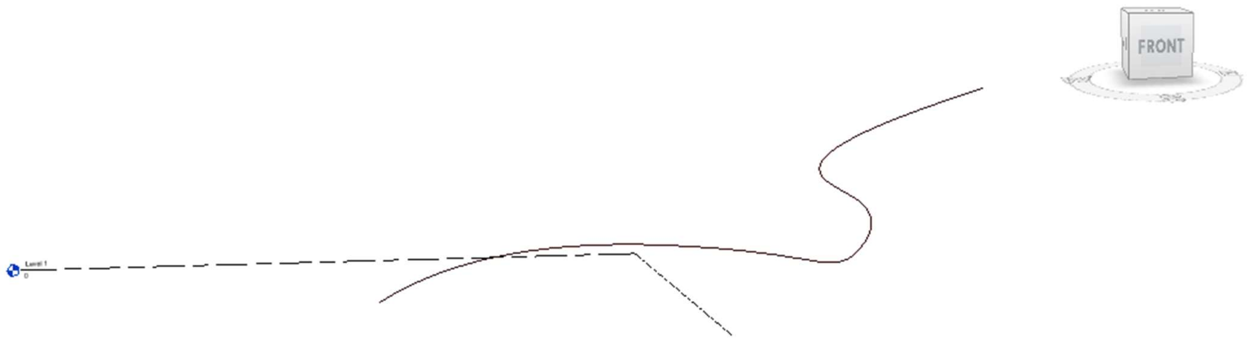


The key nodes will be marked as a group in Dynamo and look like this:



1.3 What's the Deal with These Curves

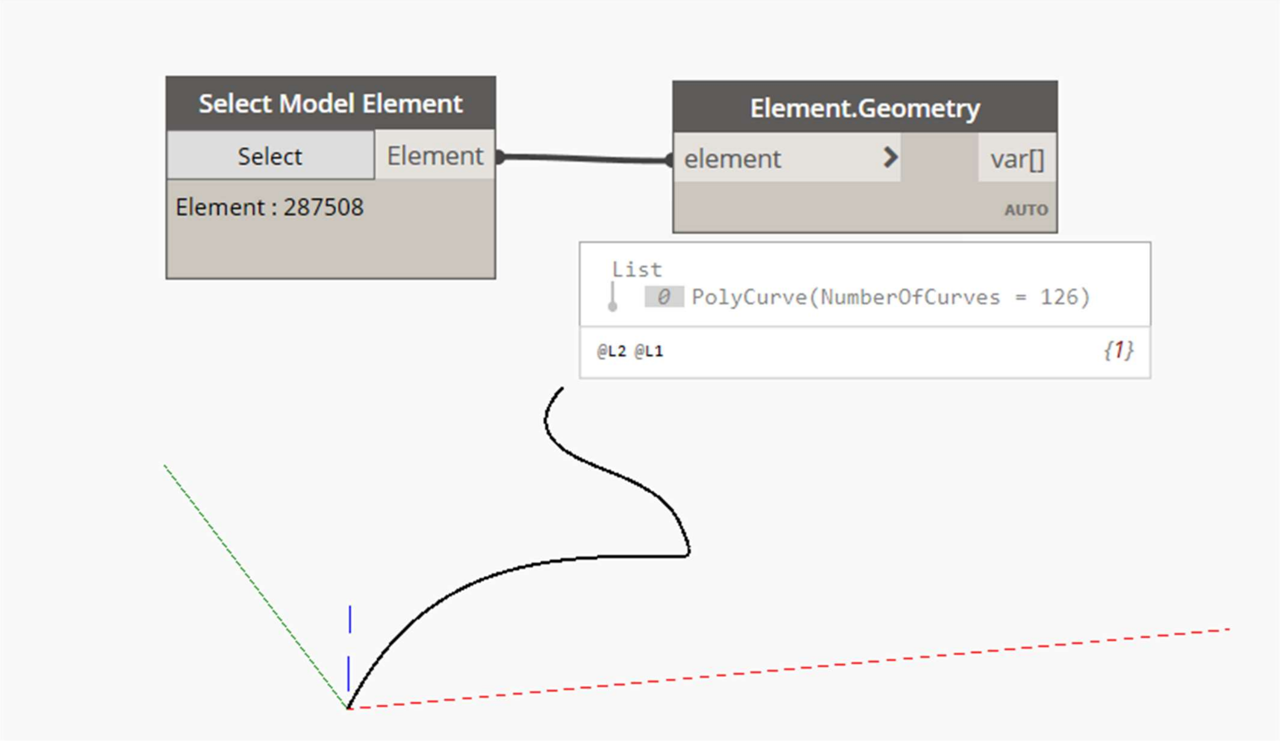
So, as previously stated, Infrastructure is a vast subject. Hell, even just Visual Programming in Infrastructure projects is a vast subject. In order to have any real value gained from this session we must churn it down to a manageable piece. First off, let's exclude stuff like Power generation, waste management and focus on linear projects, like rail, roads, water supply etc etc. One of the most idiosyncratic traits of designing in these kind of infrastructure projects is using curves. Corridors, alignments, feature lines, all of them stems from the geometric concept of curves. So that is what will serve as a basis for most of the examples in this handout. The vastness and complexity of these projects rears their ugly heads also in just how we use curves. The curve is often created in a linearly based software like Civil 3d, Novapoint, Bentley OpenRoads or something and then it is derived to any number of different formats like LandXML, dwg, VIPS. To simplify, in this session I have an imported dwg that will serve as an outset for all our scripts and algorithms.
I present:
The Curve:



It is a Polyline curve with the following traits

- Length: 102 meters
- Fluctuates in horizontal direction (XY)
- Fluctuates in vertical direction (Z)

The way we are going to operate on this curve is by extracting the imported elements geometry.



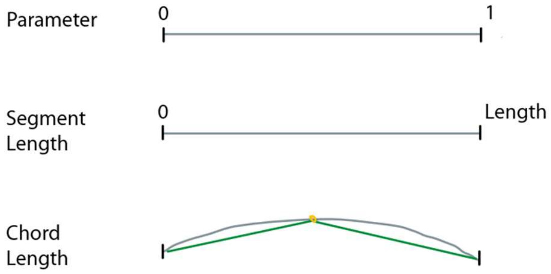
```
List
  0 PolyCurve(NumberOfCurves = 126)
```

There is a number of ways of doing this, but for simplicity's sake we're going to keep it at that.

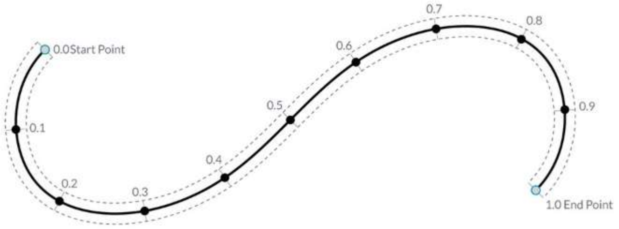
Curves ain't curves..

There are lines, arcs, polylines, circles, ellipses and whatnot. Moreover, we have NURBS-curves (Non-Uniform Rational B-Spline) and a whole more. In a Visual Programming environment, we have two ways of placing points, calculate vectors, position elements and do other operations when dealing with curves.

1. We can use the parametrization of the curve
2. We can use actual segment lengths or chord lengths



This matter profoundly.

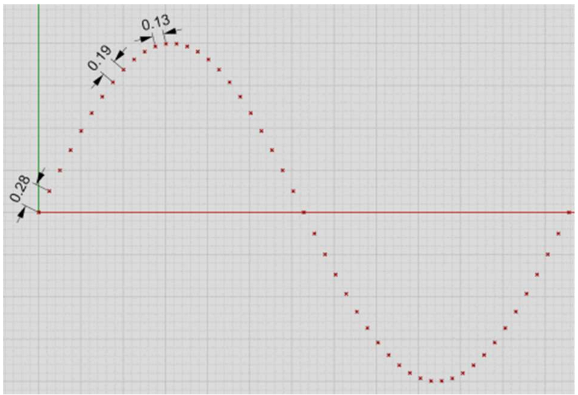


So, what is a curve?

Well, a curve in mathematical terms is really just a collection of one or more functions that that describe all possible points between a start and endpoint. If you feed the curve collection the parameter **t** into these functions, you'll get the geometric shape. Like a licorice lace right. You could bend, twist it, do whatever.

Inputting the parameter t is the same as saying "No matter how long this curve really is, I'm gonna say it starts at 0 and end at 1. If I want a point in the middle of the curve, I'll input t=0.5" But this is only partially true. It is only true for rational curves. See, for more complex curves, like Non-Rational ones, like NURBS the parameter t and the actual segment length doesn't necessarily correspond. This is crucial knowledge. And you shal not look further than to the author of this handout for someone who have gone straight into that pitfall…

NURBS mathematics use control points and weights and they can work like magnets on the parametrization resulting in stuff like this:



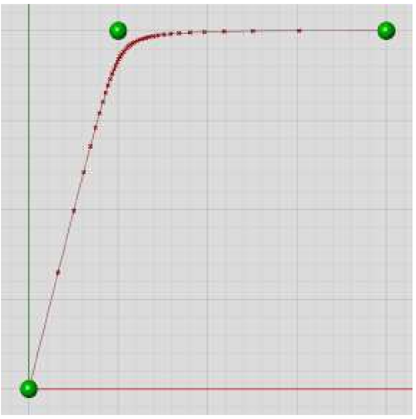
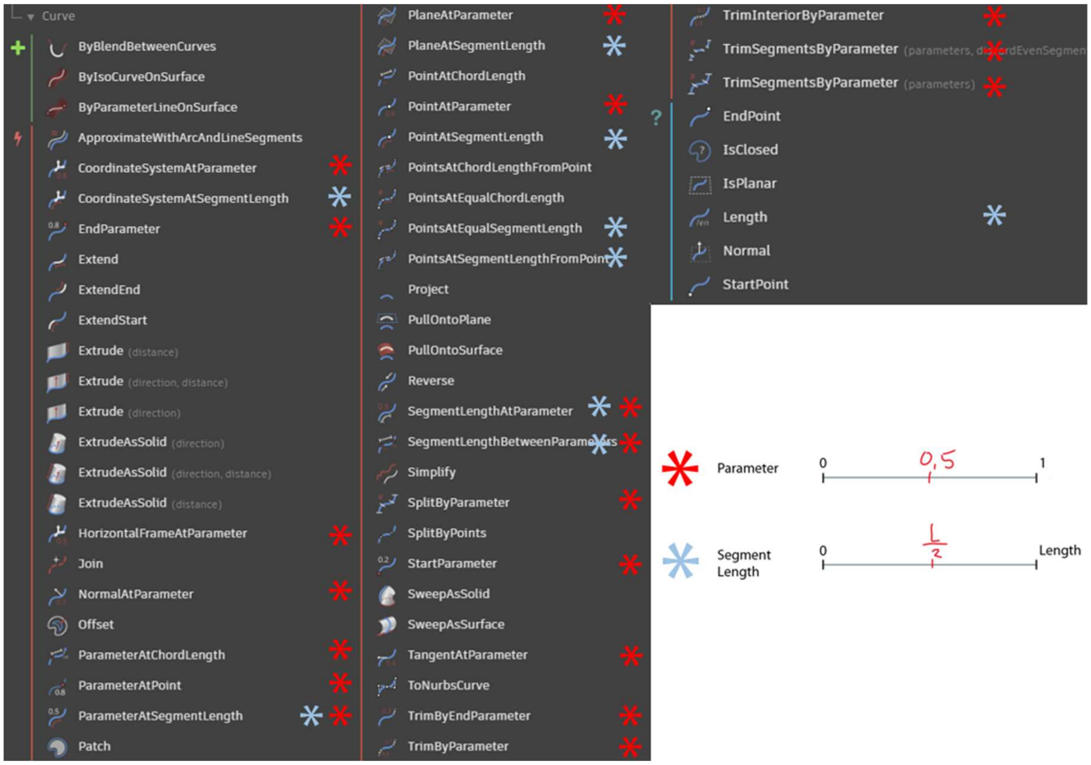


Figure 1 Sine Wave parameters (Mcneel)Weighted parameter (Mcneel)

Figure 2 Sine Wave parameters (Mcneel)

As you can see in the below picture, there are more functions dealing with parameters(0 to 1)than actual curve length (0 to Length), so keep this in mind if your constructions or objects are prone for tolerance error for instance! Equal distance between parameters doesn't necessarily mean that the segment lengths are also equal!



I'm gonna leave it at that, but if you want to read more on this, check out this awesome article by Grasshoppers father David Rutten: https://ieatbugsforbreakfast.wordpress.com/2013/09/27/curve-parameter-space/

2 Points (Along curves)

A lot of things we need to do when BIM'ing Infrastructure elements require us to deal with geometry. (In fact, that probably is the single biggest selling point for using visual programming interfaces all together since our tradtional level and grid tools are quite bad of dealing with stuff derived from linear elements.)

Visual programming sort of opens up the geometric toolbox that our tools incorporate themselves in their source code. At the base level, geometry is an abstract thing no different from numbers to the computer. All the geometry that you create on your screen is basically numbers and a structure of relationships between those numbers in a given coordinatsystem.
That means: A point consist three numbers right, given a cartessian coordinatsystem.
A line is defined by two of those points
A plane, or any 2D construct consist of multiple lines with multiple points etc etc.
We have abstracted all geometry to be dependent on each other.

In short computational geometry is a hierarchy.

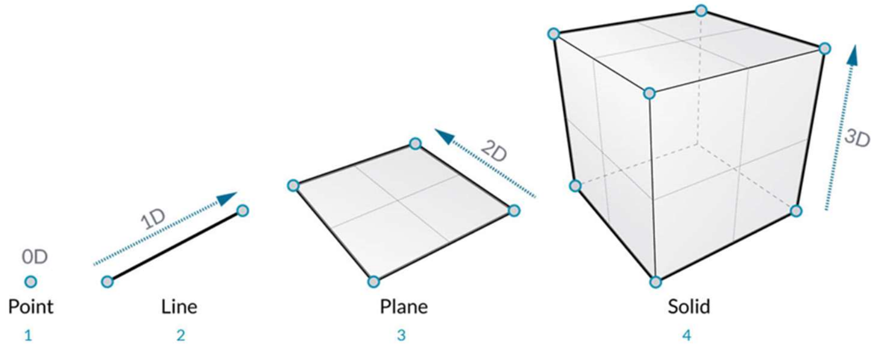


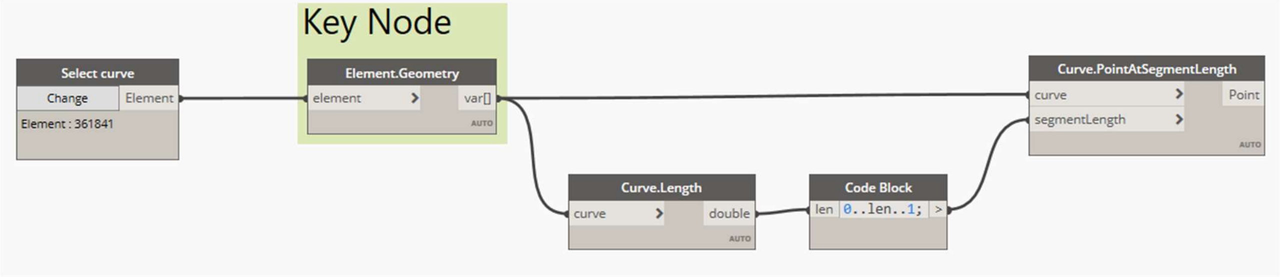
Figure 3 Geometry=Hierarchy: https://primer.dynamobim.org/05_Geometry-for-Computational-Design/5-1_geometry-overview.html

So, if geometry is the language, points would be the alphabet. They are the compound which all other computational geometry relies on.

This is also true in Infrastructure, so first we're going to have a look at how we can create points along curves in Dynamo.

2.1 Points along curves

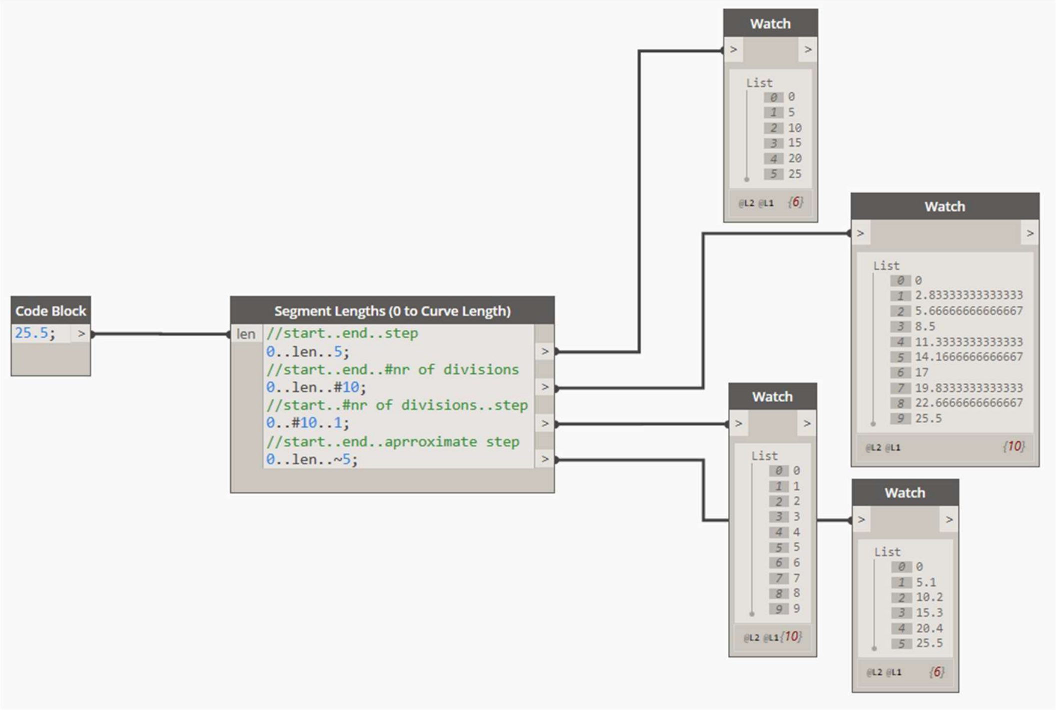
Below you can see one of the standard ways of creating points along a curve.



```
0..len..1;
```

This particular graph uses **Curve.PointAtSegmentLength**, but as stated you can also replace it with **Curve.PointAtParameter**. Just, again, remember that the points will not be equidistant on more complex curves.

Another thing to notice is the way the range is set. You could use the **Range** or **Sequence** nodes, but I really recommend using ranges in **DesignScript**. Below you'll find some of the most used explained.



```
25.5;
```

```
//start..end..step
0..len..5;
//start..end..#nr of divisions
0..len..#10;
//start..#nr of divisions..step
0..#10..1;
//start..end..aproximate step
0..len..~5;
```

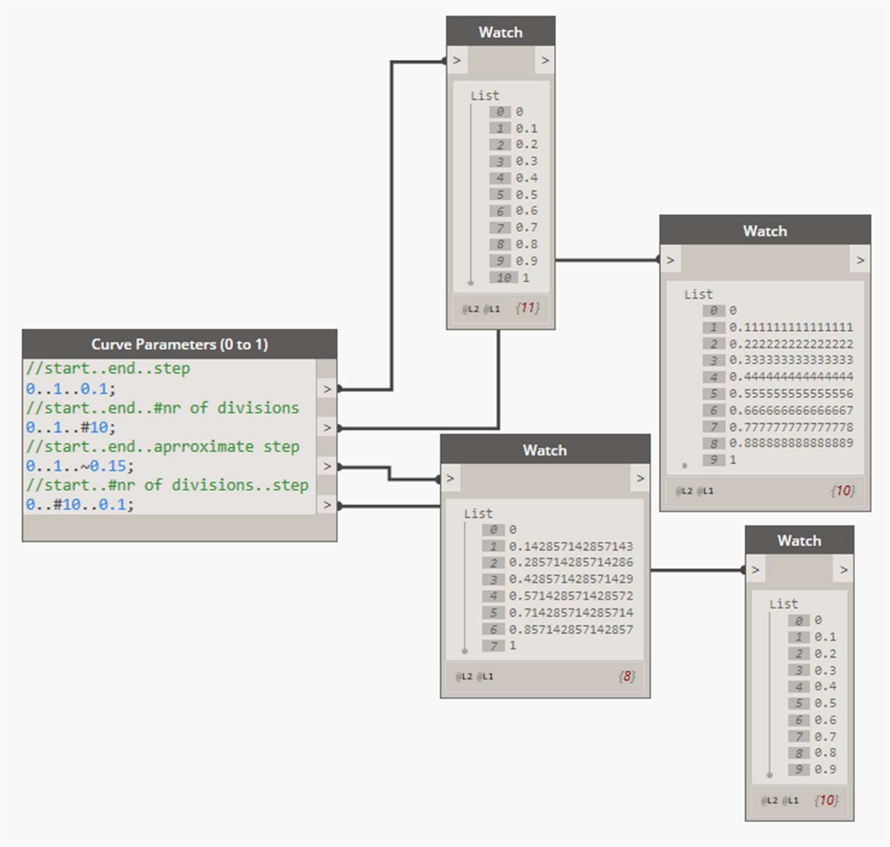
```
List
  0 0
  1 5
  2 10
  3 15
  4 20
  5 25
@L2 @L1 {6}
```

```
List
  0 0
  1 2.83333333333333
  2 5.66666666666667
  3 8.5
  4 11.3333333333333
  5 14.1666666666667
  6 17
  7 19.8333333333333
  8 22.6666666666667
  9 25.5
@L2 @L1 {10}
```

```
List
  0 0
  1 1
  2 2
  3 3
  4 4
  5 5
  6 6
  7 7
  8 8
  9 9
@L2 @L1{10}
```

```
List
  0 0
  1 5.1
  2 10.2
  3 15.3
  4 20.4
  5 25.5
@L2 @L1 {6}
```

Figure 4 Using Curve Lengths



```
//start..end..step
0..1..0.1;
//start..end..#nr of divisions
0..1..#10;
//start..end..aprroximate step
0..1..~0.15;
//start..#nr of divisions..step
0..#10..0.1;
```

```
List
  0  0
  1  0.1
  2  0.2
  3  0.3
  4  0.4
  5  0.5
  6  0.6
  7  0.7
  8  0.8
  9  0.9
  10 1
{11}
```

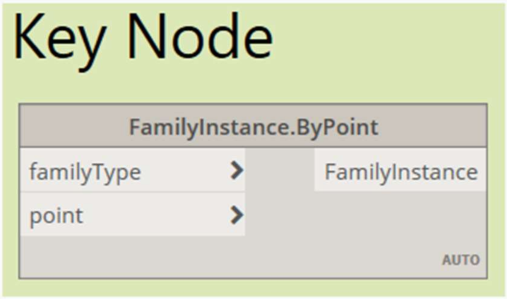
```
List
  0  0
  1  0.111111111111111
  2  0.222222222222222
  3  0.333333333333333
  4  0.444444444444444
  5  0.555555555555556
  6  0.666666666666667
  7  0.777777777777778
  8  0.888888888888889
  9  1
{10}
```

```
List
  0  0
  1  0.142857142857143
  2  0.285714285714286
  3  0.428571428571429
  4  0.571428571428572
  5  0.714285714285714
  6  0.857142857142857
  7  1
{8}
```

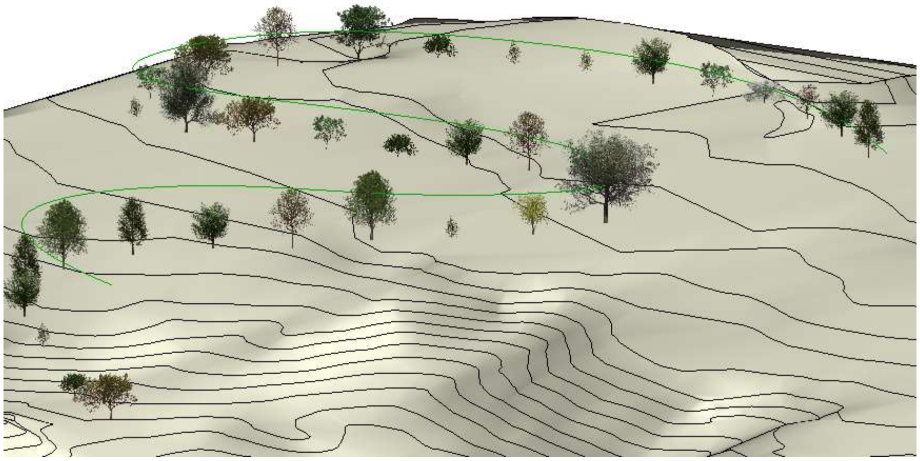
```
List
  0  0
  1  0.1
  2  0.2
  3  0.3
  4  0.4
  5  0.5
  6  0.6
  7  0.7
  8  0.8
  9  0.9
{10}
```

Figure 5 Using Curve Parametrization

Placing stuff with points are quite easy with Dynamo. They key node you can use for placing 1-point families is this one:

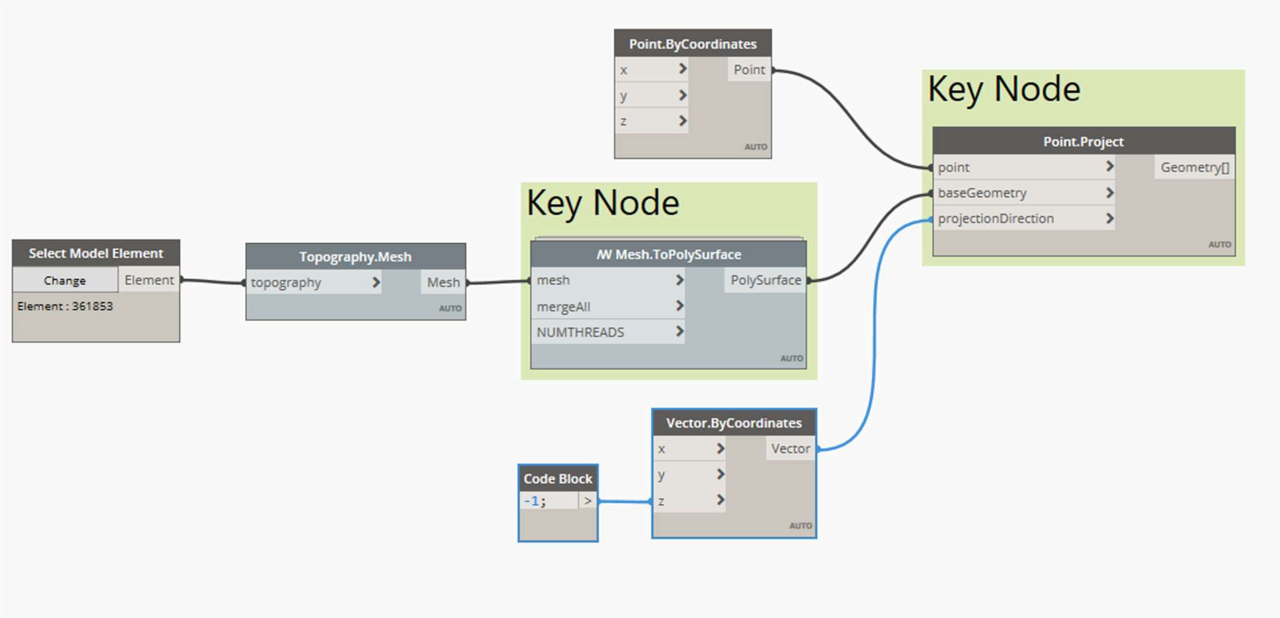


Which lets you place families in the Revit document. This can be used for placing infrastructure elements that doesn’t rely on tilting or rotating around its axis. Like trees! 😊



2.2 Points Projected

One of the most versatile features with regards to Infrastructure in Visual programming is the ability to shoot points at stuff… I can’t begin to explain how many hours’ worth of unnecessary labour this has saved during my time with it. In some of my previous sessions I have explained this in detail so go jbdynamo.blogspot.com and go to my BILT posts for an in-depth.
But Point.Project is useful, so useful. And this is how you set it up:

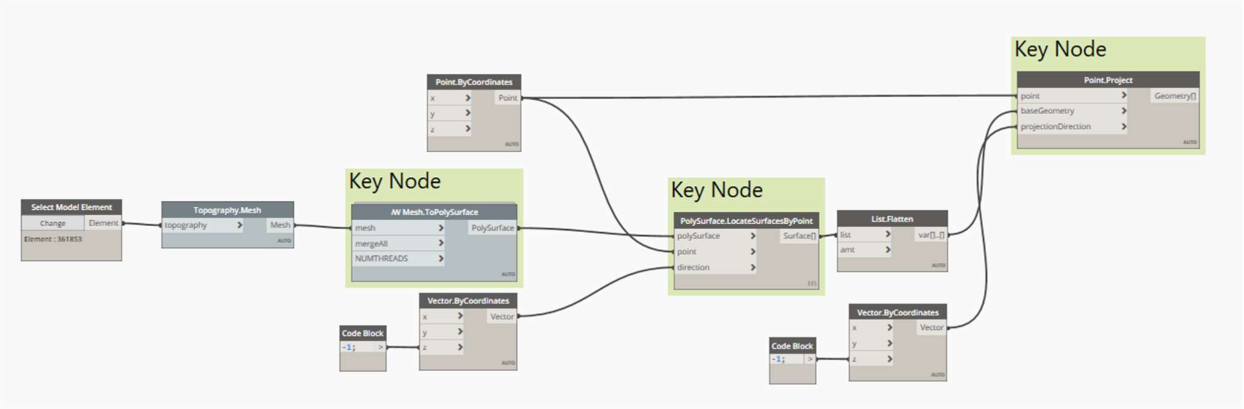


As you can see, there are two key nodes in this graph.

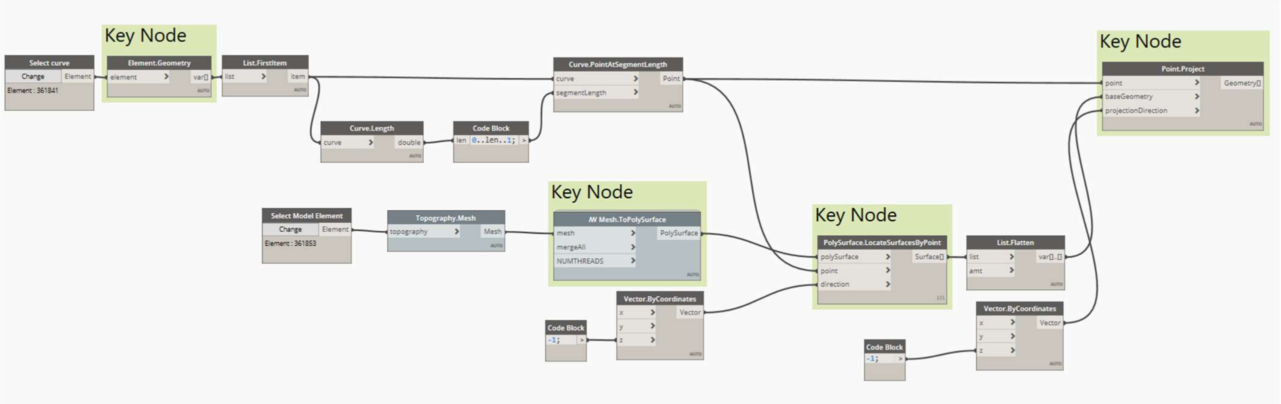
1. **Mesh.ToPolysurface:** The first is from a package called spring nodes and converts a mesh derived from a topography in Revit to a polysurface in Dynamo.
2. **Point.Project:** This is the actual "shooting node" where you provide a direction and a starting point.

Mind you that the conversion of a mesh to a polysurface is a very computationally heavy operation, so be aware when you have a large toposurface.

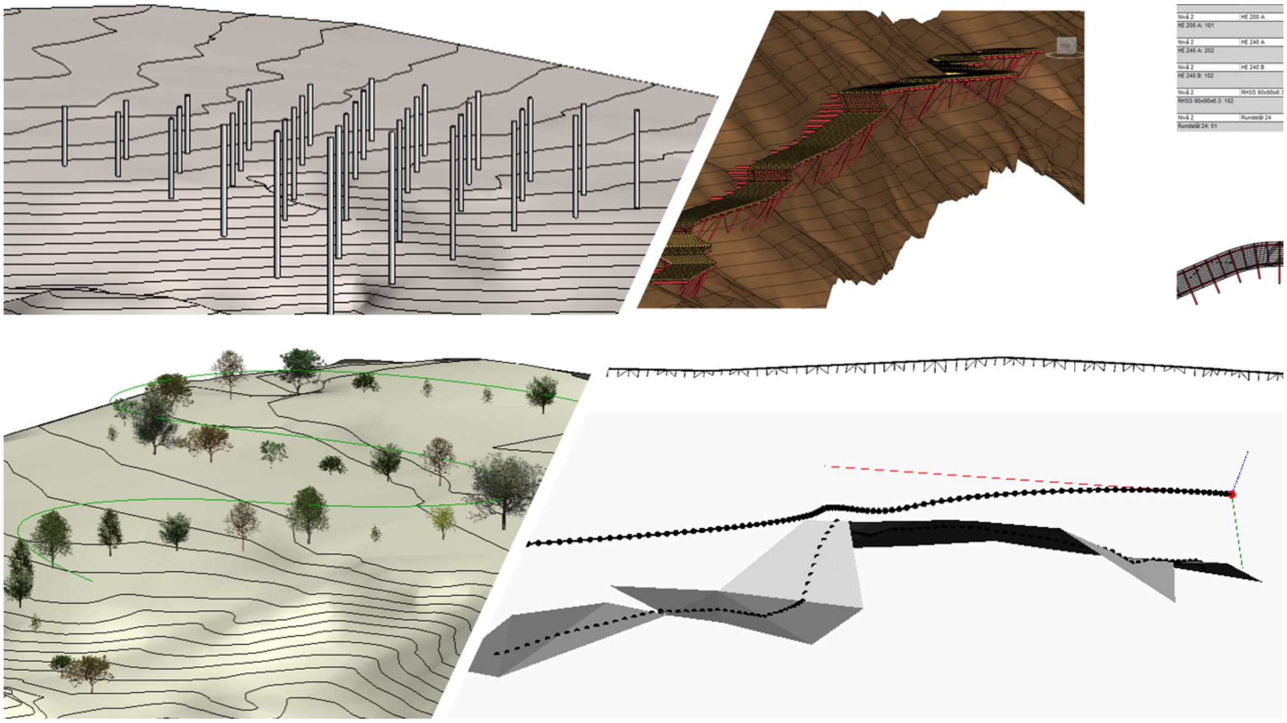
A little efficiency tips here is to add a node called **Polysurface.LocateSurfacesByPoint**. This will get the single surfaces of the polysurface that is actually valuable to us. This increases performance by quite a bit.



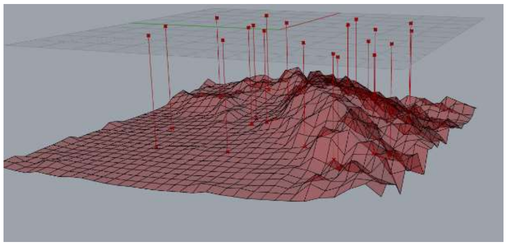
Put this together with our created curve points:



And you have yourself a rig for creating a whole range of terrain adapted stuff, like sheet piling, piles, rock anchors and so on.

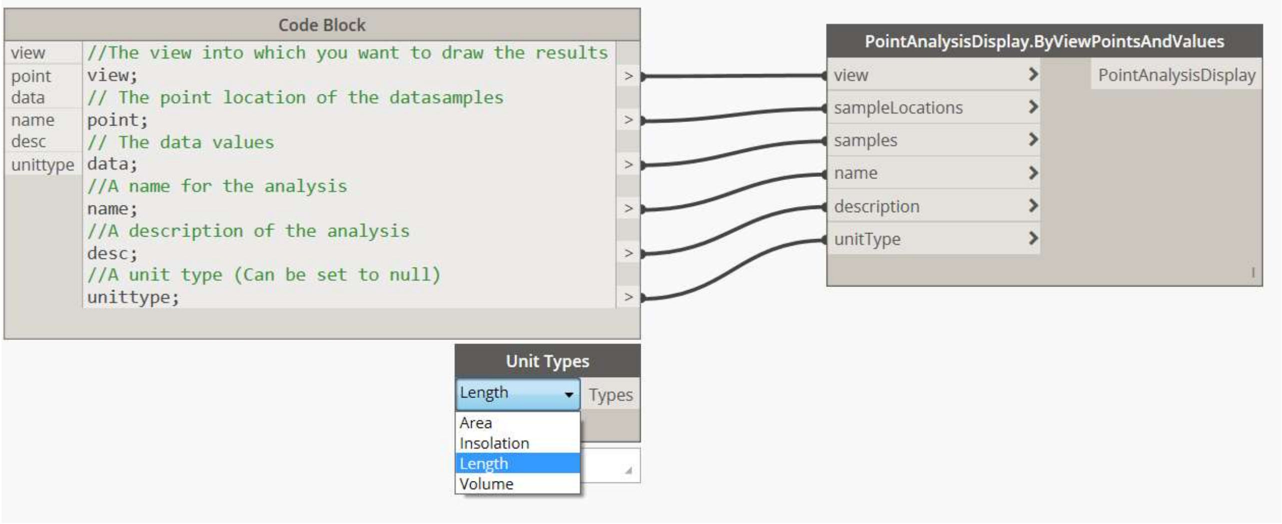


By the way, if you do have Grasshopper/Rhino available, these calculations are WAY faster to do in Grasshopper. Just sayin'. 😊



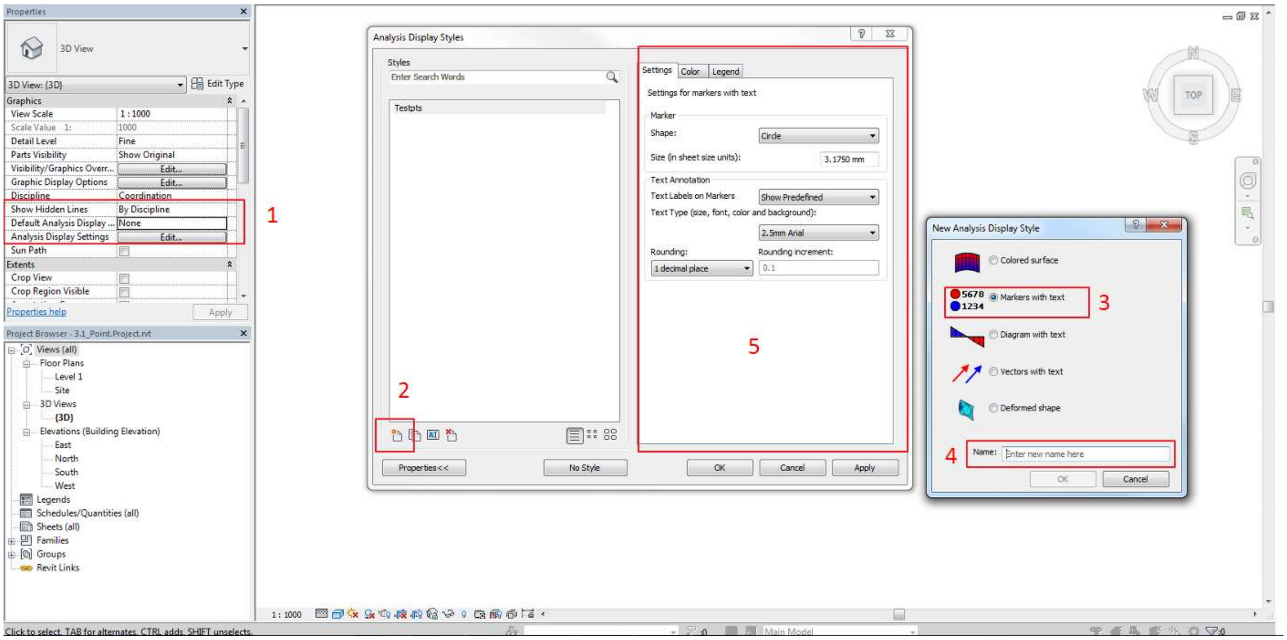
2.3 Points Analysed

Another set of Key Nodes I find interesting is the **PointsAnalysisDisplay** framework in Dynamo for Revit. This lets you graphically show any numerical data connected to a point in Revit with colours and labels in a flexible way. The key node is this one:



```
//The view into which you want to draw the results
view;
// The point location of the datasamples
point;
// The data values
data;
//A name for the analysis
name;
//A description of the analysis
desc;
//A unit type (Can be set to null)
unittype;
```

In order for it to work though, you need to set up your Revit view in a certain manner like this:



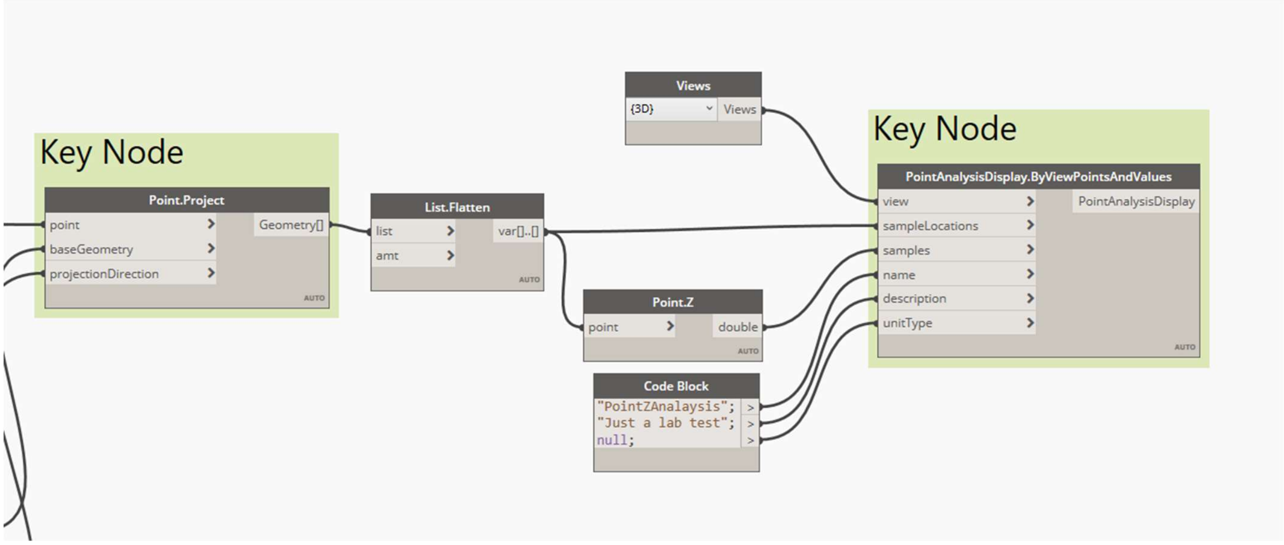
Properties: 3D View: (3D); View Scale 1 : 1000; Scale Value 1: 1000; Detail Level Fine; Parts Visibility Show Original; Discipline Coordination; Show Hidden Lines By Discipline; Default Analysis Display ... None; Analysis Display Settings Edit...

Analysis Display Styles – Settings for markers with text: Marker Shape: Circle; Size (in sheet size units): 3.1750 mm; Text Labels on Markers: Show Predefined; Text Type (size, font, color and background): 2.5mm Arial; Rounding: 1 decimal place; Rounding increment: 0.1

New Analysis Display Style: ○ Colored surface; ● Markers with text; ○ Diagram with text; ○ Vectors with text; ○ Deformed shape; Name: Enter new name here

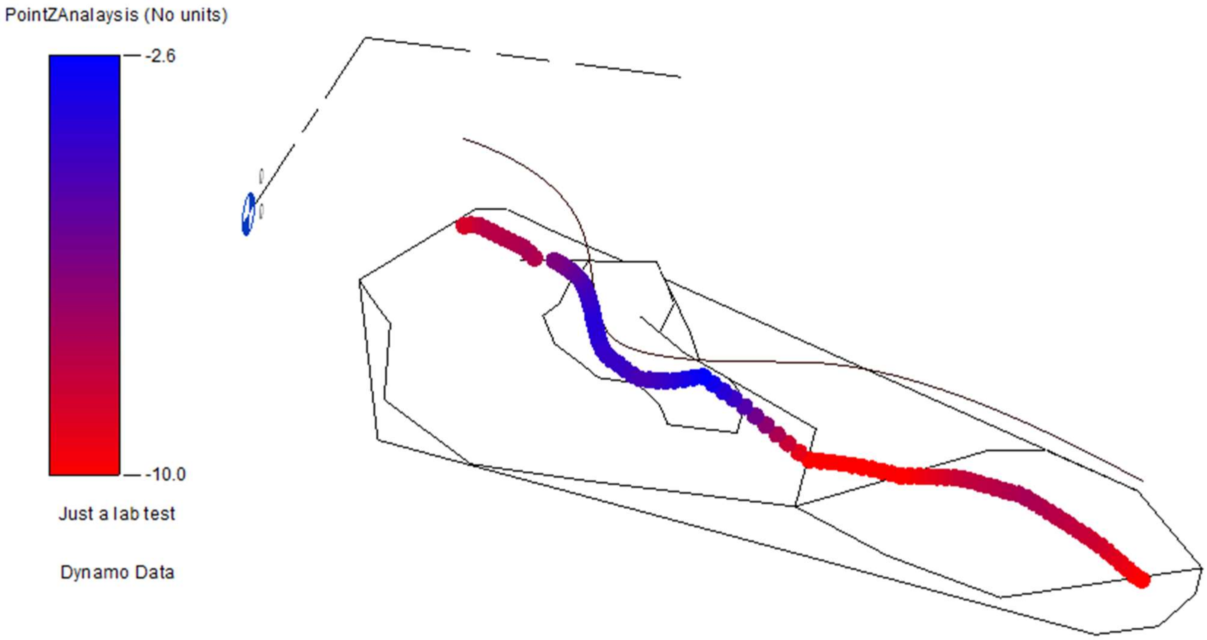
1. Select the Default Analysis Display
2. Select "New"
3. Select the "Markers with text" option
4. Give the Analysis Display Style a name
5. Set the settings

Let’s say we were to use it on our previous example with projection of points and wanted to visually identify the different heights of the points.

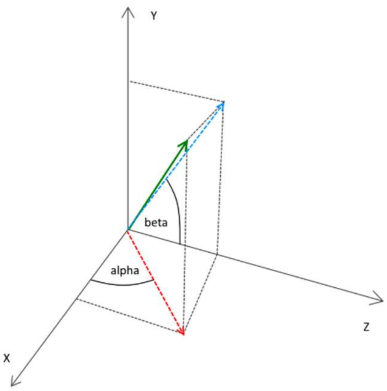


```
"PointZAnalaysis";
"Just a lab test";
null;
```

And the result:



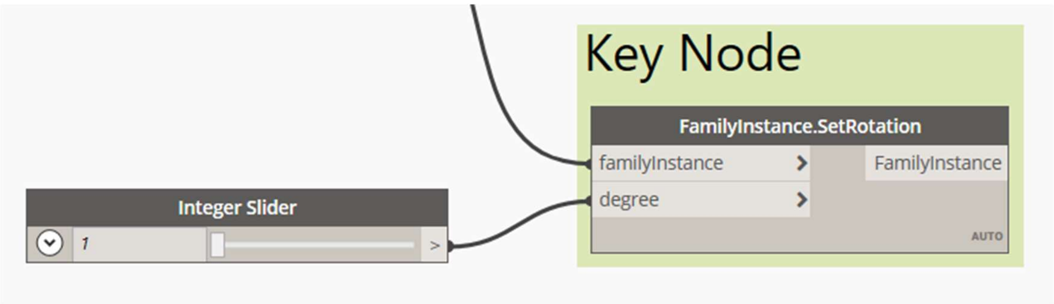
3 Vectors and Angles



More often than not, stuff that we want to place along our alignments and curves are actually following the curve as it is twisting and turning up, down and sideways. So, though trash bins may very well be placed without any rotation, lamp post are not for instance. The same can be said for placing profiles along curves, creating bridges, etc etc. In that case we need to start operating with angles and vectors. So, buckle up, we're going back to school now for some maths! 😊

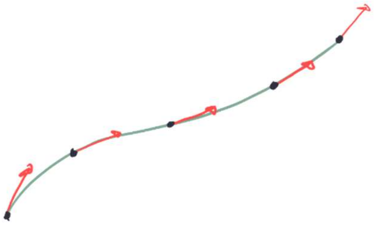
3.1 Setting an Elements Rotation

The key node for setting rotations on Revit elements with Dynamo is the **FamilyInstance.SetRotation** node.

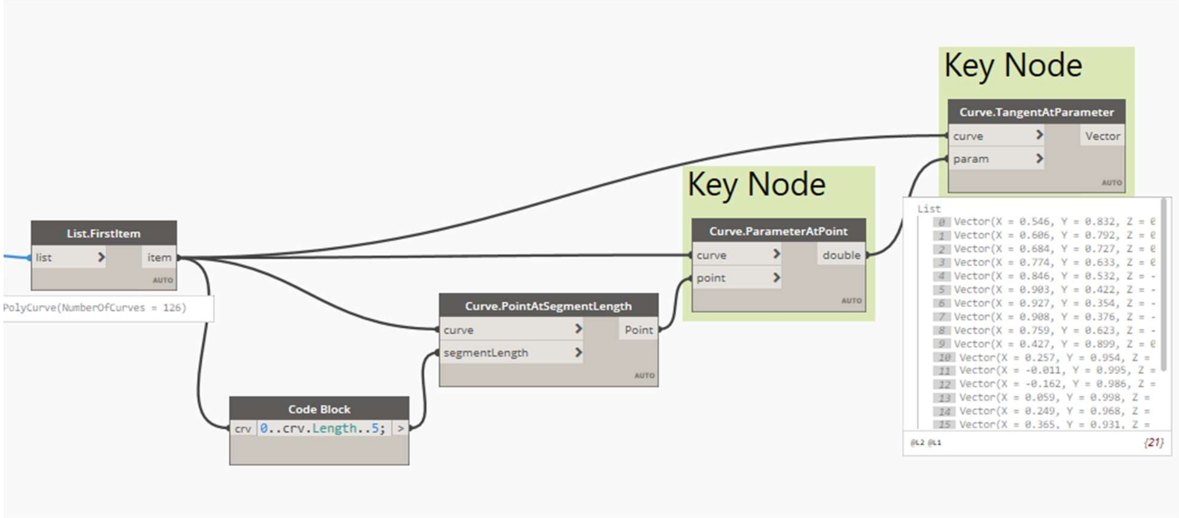


This node accepts family instance(s) and angle(s) between 0 to 360, ie the Euler angle. (Check it out here: https://en.wikipedia.org/wiki/Euler_angles)

To find angles along curves given that we already have placed location points along it is quite easy, we can extract the tangent vector at the points we have placed along the curve and measure the angle between the tangent and a fixed vector (like the Y-axis.)



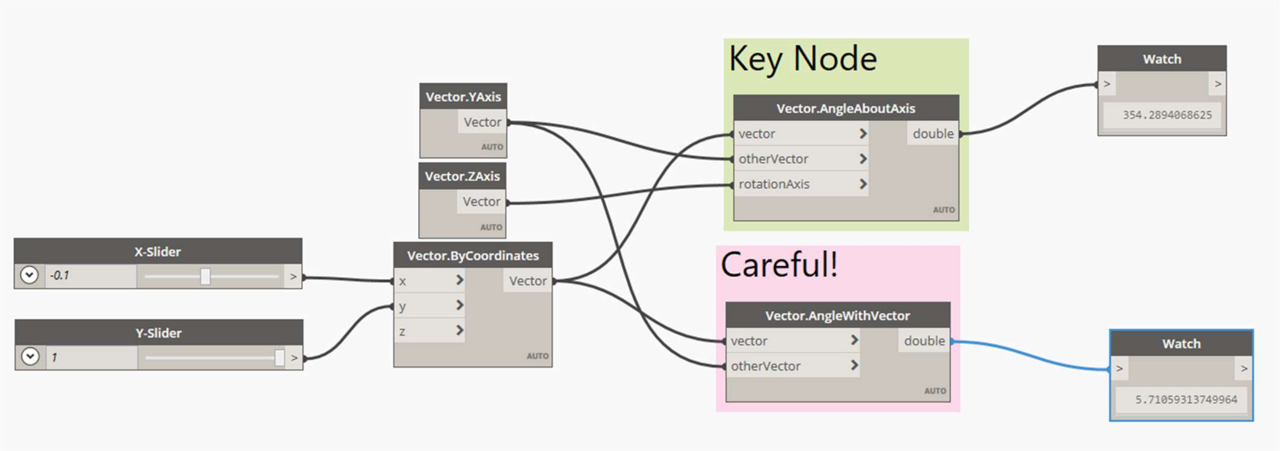
The only thing to keep in mind is that Dynamo only way of finding tangents along a curve is by using the curve parametrization, ie parameter between 0 to 1, so we need to compute the parameter values at those points we have placed if segment lengths is used to place the points. Hard to read sentence, but the pic below sums it up good.



```
0..crv.Length..5;
```

```
List
  0 Vector(X = 0.546, Y = 0.832, Z = 0
  1 Vector(X = 0.606, Y = 0.792, Z = 0
  2 Vector(X = 0.684, Y = 0.727, Z = 0
  3 Vector(X = 0.774, Y = 0.633, Z = 0
  4 Vector(X = 0.846, Y = 0.532, Z = -
  5 Vector(X = 0.903, Y = 0.422, Z = -
  6 Vector(X = 0.927, Y = 0.354, Z = -
  7 Vector(X = 0.908, Y = 0.376, Z = -
  8 Vector(X = 0.759, Y = 0.623, Z = -
  9 Vector(X = 0.427, Y = 0.899, Z = 0
  10 Vector(X = 0.257, Y = 0.954, Z =
  11 Vector(X = -0.011, Y = 0.995, Z =
  12 Vector(X = -0.162, Y = 0.986, Z =
  13 Vector(X = 0.059, Y = 0.998, Z =
  14 Vector(X = 0.249, Y = 0.968, Z =
  15 Vector(X = 0.365, Y = 0.931, Z =
@L2 @L1                          {21}
```

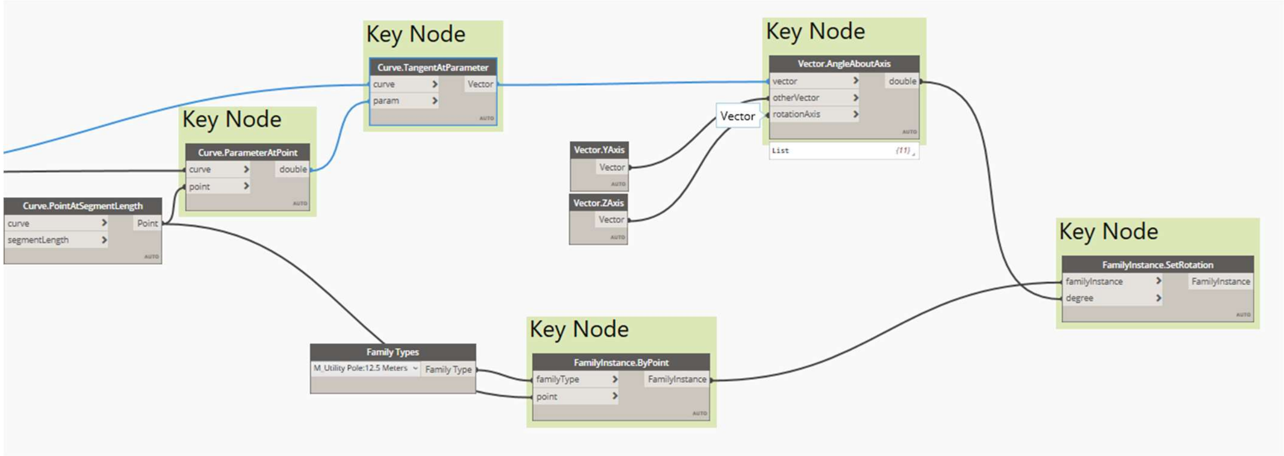
So now, we have a list of vectors and we can measure the angle with a fixed vector to set the rotations. There are two nodes in Dynamo for doing this and you must use one over the other.



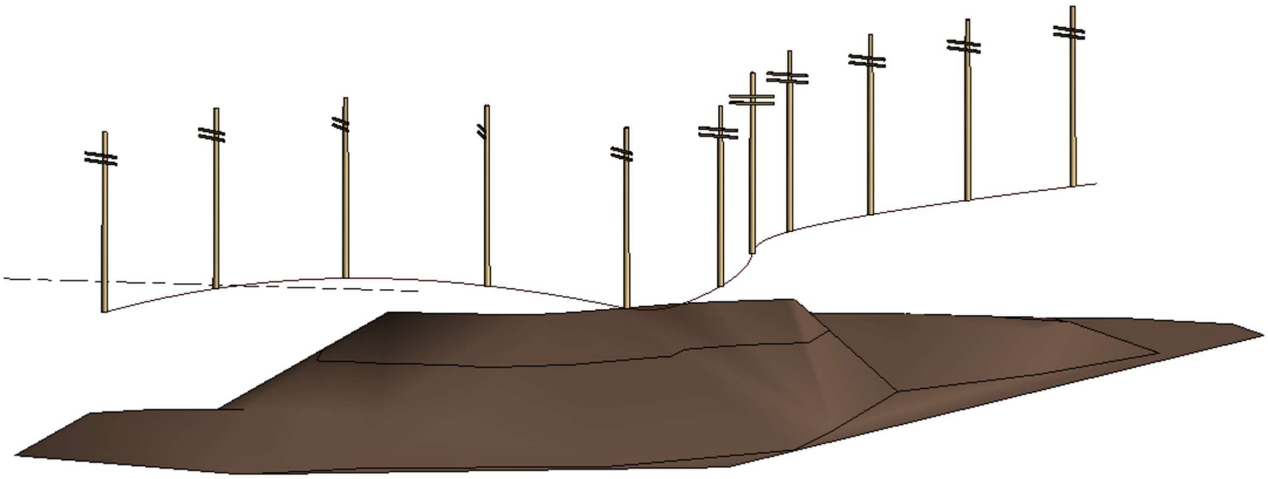
```
X-Slider: -0.1
Y-Slider: 1
Watch: 354.2894068625
Watch: 5.71059313749964
```

The reason for choosing **Vector.AngleAboutAxis** is that it returns a number between 0 to 360 whereas the **Vector.AngleWithVector** returns values between 0 and 180. Revit expects 0 to 360 so be careful of using the latter!

Say we want to create utility poles along our little line, we can continue the graph above like this:



And we'll get the result in Revit like this:



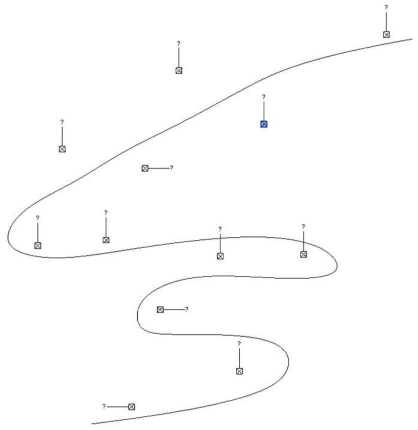
4 Data

One of the really strong things about Dynamo is the ability to acquire, modify and set data within Revit. Compared to other combinations of modelling software and Visual Programming this is where Revit and Dynamo really shines.
This is especially handy in Infrastructure projects where scalability is paramount since we're usually looking at huge stacks of elements, data and geometry but where the logic is often quite fathomable and easily "programmable". (like these utility poles will be placed once every 5 meters, perpendicular to the curve they follow, 3 meters from the centreline, but there is like 10000 of them!!!) Often data in Infrastructure projects are bespoke in the sense that the naming system, the data schema you have to fill in in BIM-projects etc. vary heavily from project to project. This makes it hard for Software developers to adapt and care for all the needs for their clients since it is so fragmented. So, enter Visual programming tools again! 😊

4.1 Renumber Elements

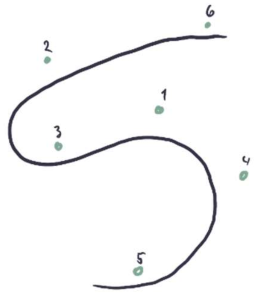
I've seen many examples for dealing with data with visual programming tools, but I time and time again come back to this one. Renaming elements after a curve. It is a good example because it shows us how we can extract geometry from the modelling software, like Revit, use the geometry engine and sorting abilities in Dynamo and push it all back into Revit again.

So, for simplicity, let's say we have a number of bollards in Revit randomly placed and then we want to rename after a curve so that the closer the bollard is to the start of the curve the lower its mark number is supposed to be.

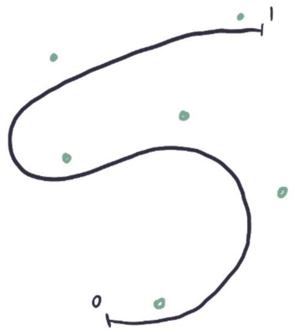


So, this is the gist of the logic in the script:

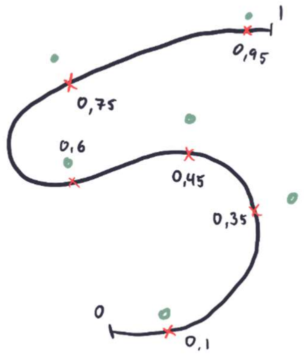
Extract the location points with their existing numbering and the curve that we want to use.



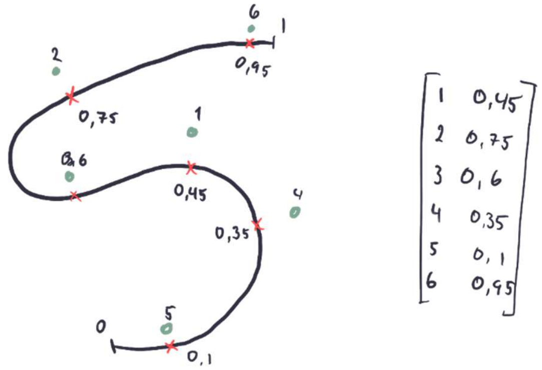
Now, if we normalize the curve length we can say that no matter how the curve may look it runs from 0 to 1 in length



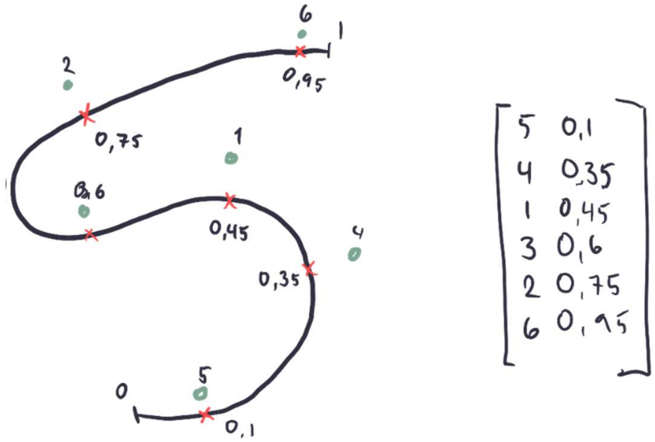
If we take our points find the closest points on the curve, we can extract the normalized curve length (or parameter) at those points.



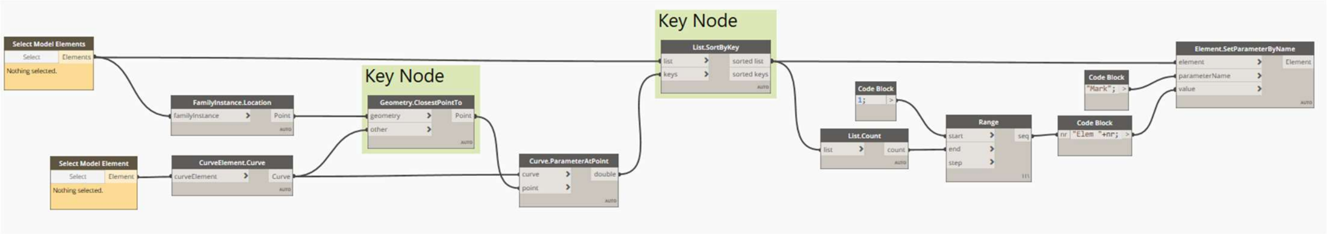
This means that we now have the original list of point numbering with its belonging parameter along the curve.



Knowing from earlier experience that there exists functionality that can sort list based on another list we can now sort the original list of points with regards to the curve parameter belonging to them.



Now we have a list that have sorted the points and we can renumber them.

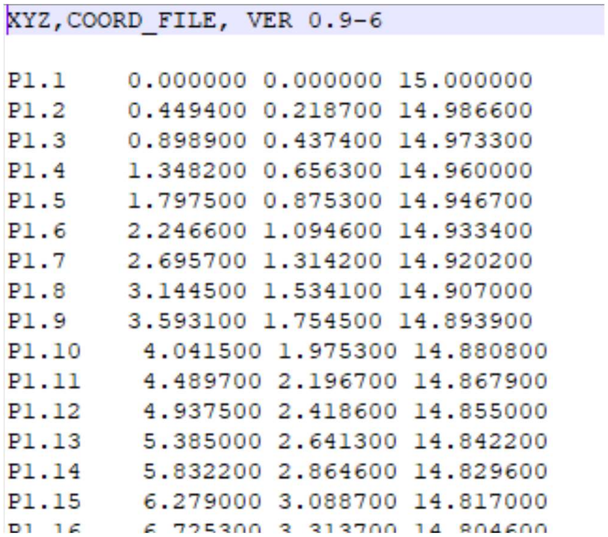


Have a look at the “Data” Dynamo file in the examples.

4.2 Parsing Text Data

Often when the lack of standards and compatibility gets too frustrating, you'll find yourself diving into text files or CSV or whatnot to get the data you need. Visual programming tools again shows off their versatility in that you can use them to parse this information into actual useful geometry or data in your BIM software of choice.

This is a typical file that represents some export from an unidentified road creation software.



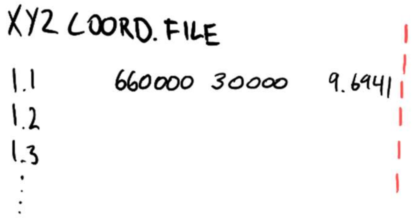
```
XYZ,COORD_FILE, VER 0.9-6

P1.1    0.000000 0.000000 15.000000
P1.2    0.449400 0.218700 14.986600
P1.3    0.898900 0.437400 14.973300
P1.4    1.348200 0.656300 14.960000
P1.5    1.797500 0.875300 14.946700
P1.6    2.246600 1.094600 14.933400
P1.7    2.695700 1.314200 14.920200
P1.8    3.144500 1.534100 14.907000
P1.9    3.593100 1.754500 14.893900
P1.10    4.041500 1.975300 14.880800
P1.11    4.489700 2.196700 14.867900
P1.12    4.937500 2.418600 14.855000
P1.13    5.385000 2.641300 14.842200
P1.14    5.832200 2.864600 14.829600
P1.15    6.279000 3.088700 14.817000
P1.16    6.725300 3.313700 14.804600
```

In Dynamo we have the power of string parsing which lets us translate this text file into something useful. Abstracting the process would look like this:

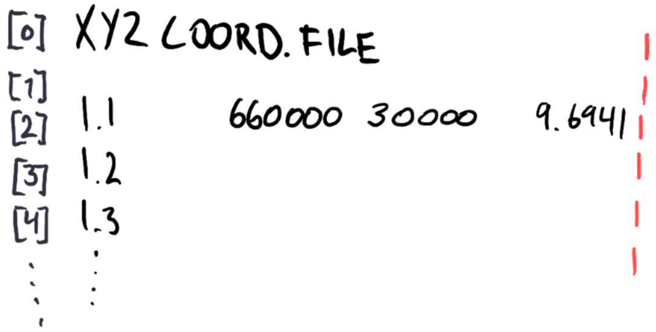
You have the text in one string looking like this

```
XYZ COORD.FILE
1.1      660000 30000    9.6941
1.2
1.3
:
```

A good way of starting would be to split the string at each new line, right?



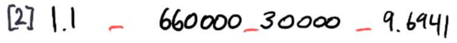
This would lead to a list of items like this (this part involves using the concept of storing data in lists, like Dynamo does..)



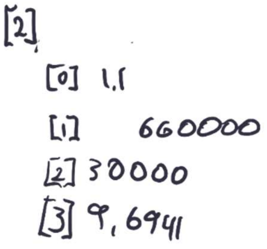
Now that we have separate items how can we split it up further? Let's take one line:

[2] 1.1 660000 30000 9.6941

Now, we see that there is at least one space between the numbers we want, so we can split it at each "space".



This returns a list of lists.

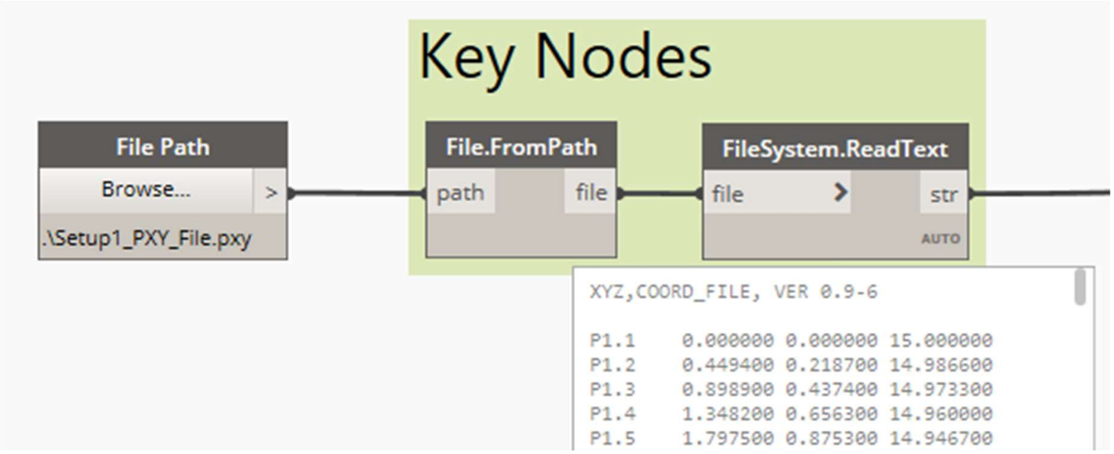


As we can see though, splitting by just one “space” character will leave some whitespace in front or behind our data, so we have to get rid of all whitespace on all items as well.

[2]
  [0] 1.1
  [1] 660000
  [2] 30000
  [3] 9,6941

This way we have the items we need separately and now the final thing is to convert the string values of XYZ to actual number values.

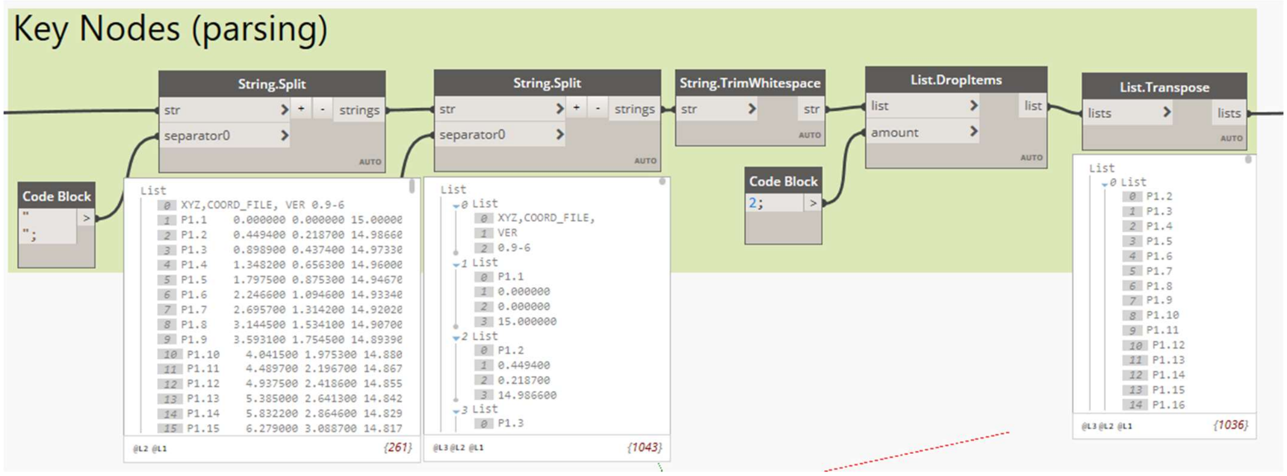
In Dynamo the implementation will start something like this:



```
XYZ,COORD_FILE, VER 0.9-6

P1.1    0.000000 0.000000 15.000000
P1.2    0.449400 0.218700 14.986600
P1.3    0.898900 0.437400 14.973300
P1.4    1.348200 0.656300 14.960000
P1.5    1.797500 0.875300 14.946700
```

(The **File.FromPath** is necessary because if the text file updates, Dynamo will be able to tell that it is updated and reruns the script.)

For this particular case this is how we can continue like this:



```
"
";
```

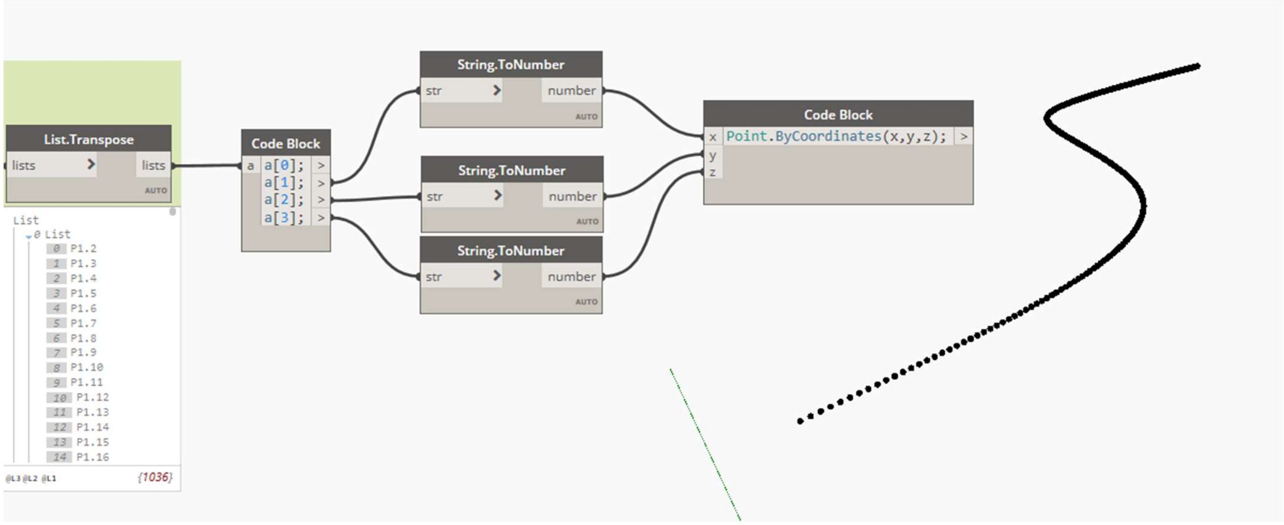
```
2;
```

```
List
  0  XYZ,COORD_FILE, VER 0.9-6
  1  P1.1    0.000000 0.000000 15.00000
  2  P1.2    0.449400 0.218700 14.98660
  3  P1.3    0.898900 0.437400 14.97330
  4  P1.4    1.348200 0.656300 14.96000
  5  P1.5    1.797500 0.875300 14.94670
  6  P1.6    2.246600 1.094600 14.93340
  7  P1.7    2.695700 1.314200 14.92020
  8  P1.8    3.144500 1.534100 14.90700
  9  P1.9    3.593100 1.754500 14.89390
  10  P1.10    4.041500 1.975300 14.880
  11  P1.11    4.489700 2.196700 14.867
  12  P1.12    4.937500 2.418600 14.855
  13  P1.13    5.385000 2.641300 14.842
  14  P1.14    5.832200 2.864600 14.829
  15  P1.15    6.279000 3.088700 14.817
@L2 @L1                                {261}
```

```
List
  0 List
    0  XYZ,COORD_FILE,
    1  VER
    2  0.9-6
  1 List
    0  P1.1
    1  0.000000
    2  0.000000
    3  15.000000
  2 List
    0  P1.2
    1  0.449400
    2  0.218700
    3  14.986600
  3 List
    0  P1.3
@L3 @L2 @L1                    {1043}
```

```
List
  0 List
    0  P1.2
    1  P1.3
    2  P1.4
    3  P1.5
    4  P1.6
    5  P1.7
    6  P1.8
    7  P1.9
    8  P1.10
    9  P1.11
    10  P1.12
    11  P1.13
    12  P1.14
    13  P1.15
    14  P1.16
@L3 @L2 @L1          {1036}
```

This splits our data into usable list structures in Dynamo and we're able to get actual Dynamo geometry from otherwise unreadable file formats. As long as the data is structured, we can extract at least something useful from it.



```
a[0];
a[1];
a[2];
a[3];
```

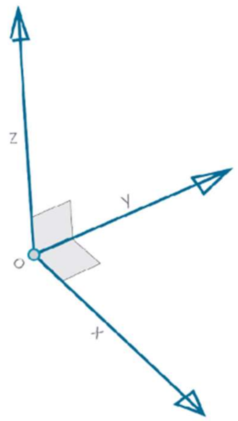
```
Point.ByCoordinates(x,y,z);
```

```
List
  0 List
    0  P1.2
    1  P1.3
    2  P1.4
    3  P1.5
    4  P1.6
    5  P1.7
    6  P1.8
    7  P1.9
    8  P1.10
    9  P1.11
    10  P1.12
    11  P1.13
    12  P1.14
    13  P1.15
    14  P1.16
@L3 @L2 @L1          {1036}
```

From here we can use the points to create lines, polycurves etc etc. Whatever makes the most sense to you! 😊

This is where we're using Dynamo as a middleware. If you're interested in reading more about "The Rise of Middleware" Nathan Miller has an awesome blogpost right here: https://provingground.io/2016/06/21/the-wicked-problem-of-interoperability/

5 Coordinate Systems



Once we get have points, vectors and angles covered, a natural progression is to have a go at coordinate systems (CS). What is a CS you might say?
According to Wikipedia: "a CS is a system which uses one or more numbers, or coordinates, to uniquely determine the position of the points or other geometric elements on a manifold such as Euclidean space." Easy..

In Infrastructure, especially linearly based, projects some of the disciplines designing the projects are confronted with two diametrically different domains. Continuous elements and discrete elements.

Placing XYZ Coordinate Systems in visual programming tools (Frames in Grasshopper) are a good construct for alleviating some of the pain in transforming between the two. Especially going from continuous elements like curves and place discrete elements along them.

The reason for why it is good to use CS is because we're able to place them along curves and then use them for placing stuff, ie with an offset in XYZ direction locally instead of having to use real world coordinates.

There are many ways of constructing them in Dynamo, below is two ways I've found to be the most reliable when dealing with curves. One for creating CS where the Z-axis "follows" the curvature and one where the XY-plane is level.

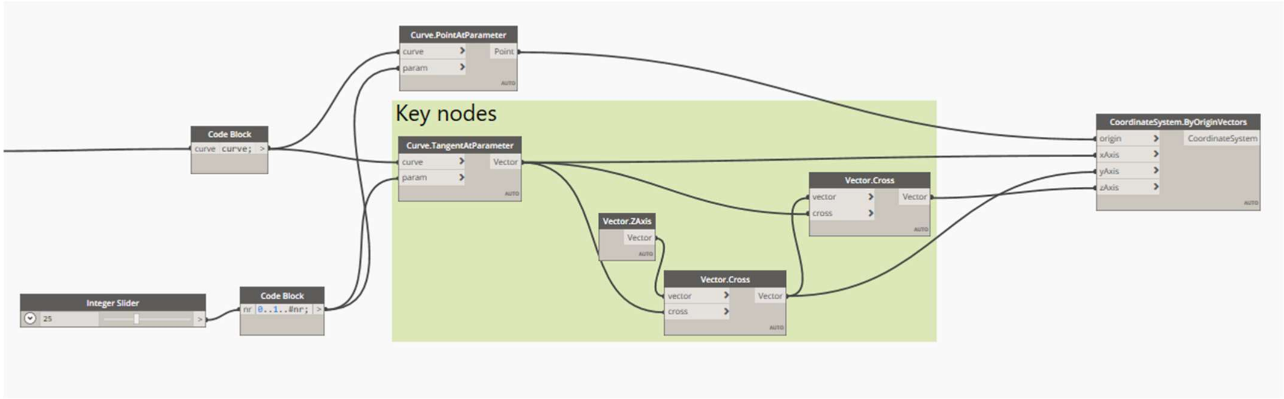


Figure 6 Z-axis follows curvature

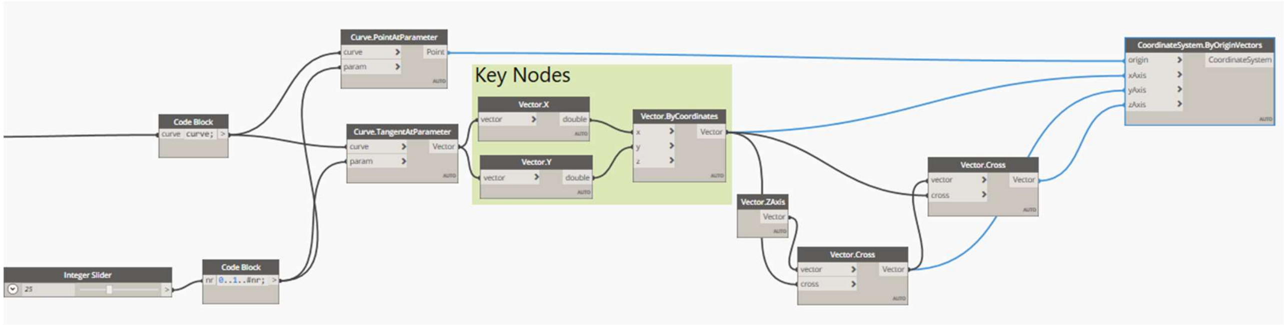
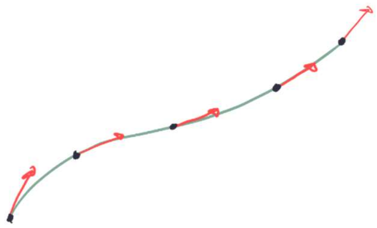


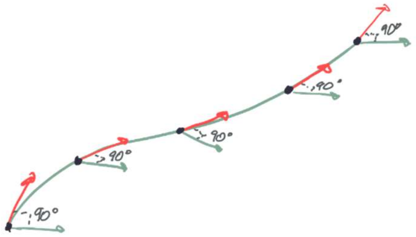
Figure 7 Z-axis is "up" (ie XY-plane is horizontal)

Explained in steps the abstraction of it is something like this. And, yes, you will find your college vector math useful here! 😊

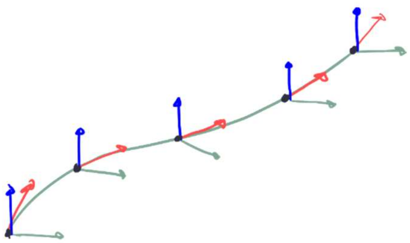
If we can find the tangent-vector to the curve at the points given we will get one of the axis, the X-axis. If we want to "flatten" out the vector we can simply extract the tangents X and Y values and create a new vector where the Z-value is 0.



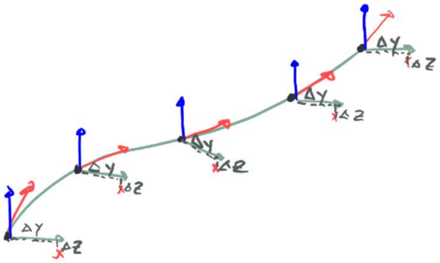
Then, using our college math, we use Cross-product between the tangent and the Z axis to get the Y-axis of our CS. No matter what angle our tangent has with the ground plane (XY-plane) the cross product between the tangent and Z-axis will always stand orthogonal to our tangent vector.



Finally, we can use math again and take a cross product between our X-axis and Y-axis and we'll find the Z-axis of our system! 😊

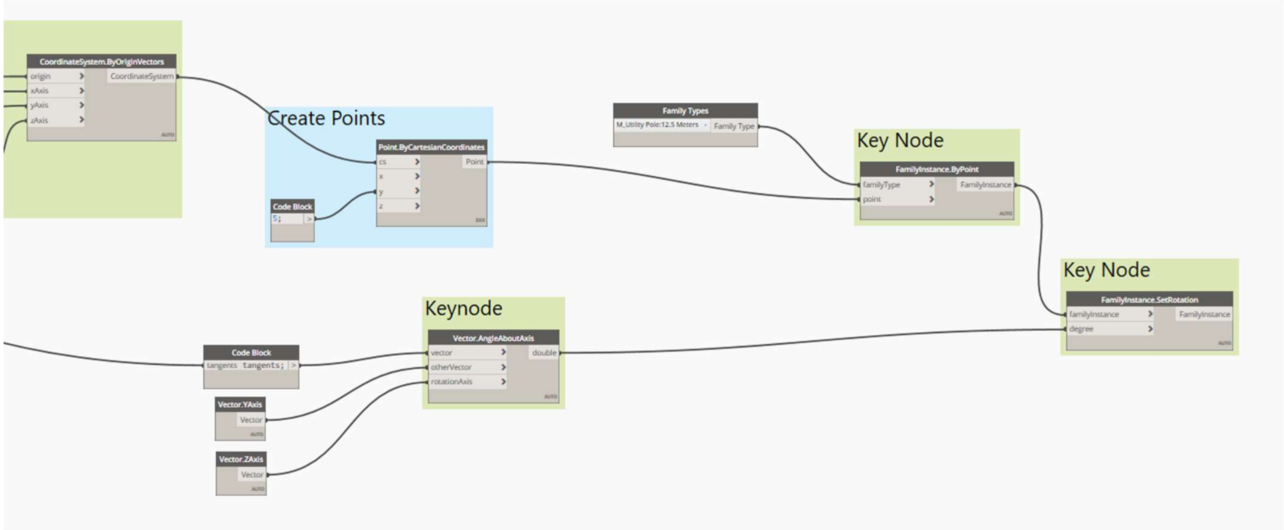


Now, we can begin placing points in each local CS. So now, instead of real world coordinates we use XYZ-values within each respective system.

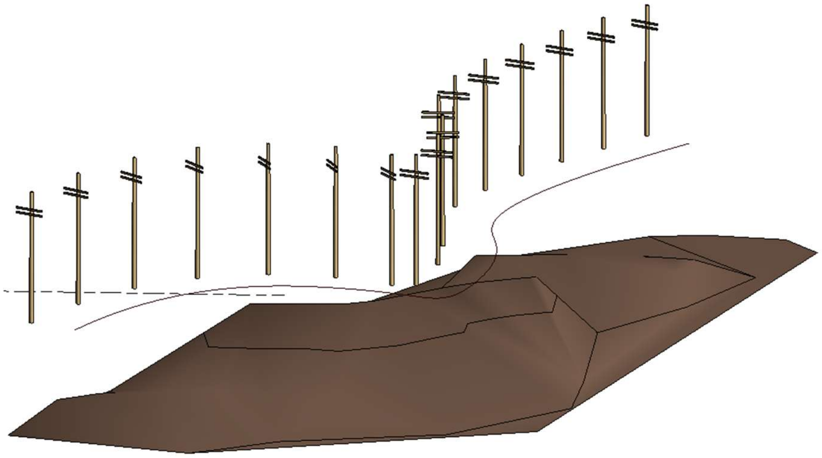


5.1 Placing Utility poles again...

Going back to our utility pole example building on what we now know about CS we can set them out with an offset.



Resulting in this:



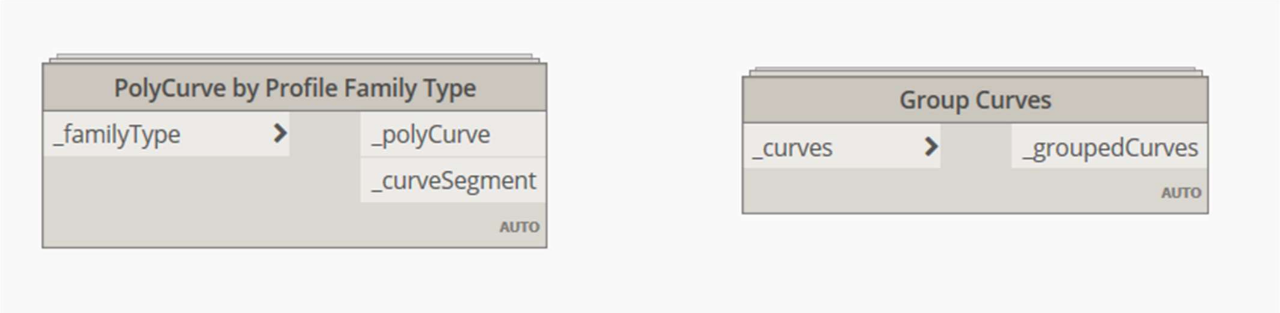
5.2 Building Solids using CS

There are many bespoke solutions out there for creating, updating, maintaining Solid infrastructure geometry. The following example is just a simple example that can be used in early stages of a project and doesn't necessarily represent what should be done in the closing stages of a project with rebars, shop drawings and what not. Still, being able to quickly generate models based on other disciplines input in early stages of the project is maybe where automation and Visual Programming shines the most. So, if you're in dire need of visualising a bridge and don't really have the hours, or don't know if the road lines are fixed yet, and the Visualisation team delivery is due Monday next week? Then my all means, read on! 😉

As stated, apart from taking a profile a lofting it along a curve ((Which doesn't really give you control over the profiles rotation along the path! Like if you have done sweeps in Revit..), this is one of the simplest ways of creating a solid along a path that fluctuates in XYZ.

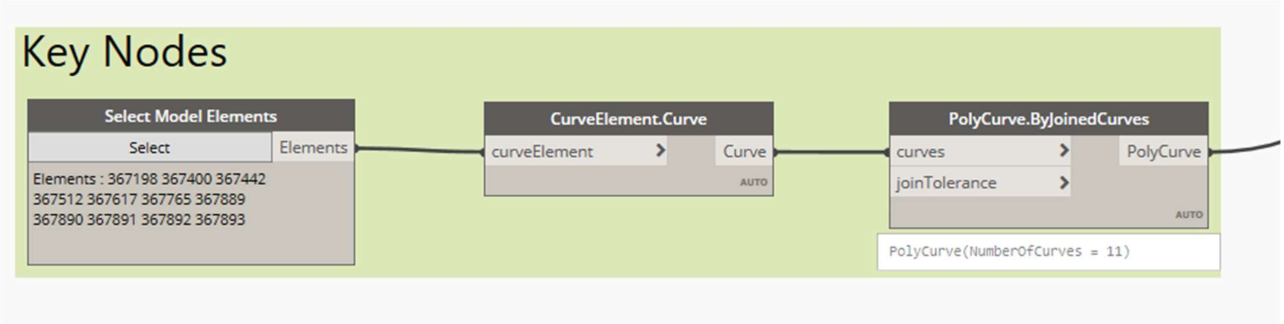
What we're aiming for is to copy out a profile to all our created CS and then loft between them. A simplification I've done for the matter of clarity is that I've drawn the profile with model lines at the start of the curve perpendicular to the curve's path so that it fits snuggly with the first CS we create.

This can be done in different ways depending on how you have the profile available
TIPS! Check out the Archi-lab.net package and these two nodes:

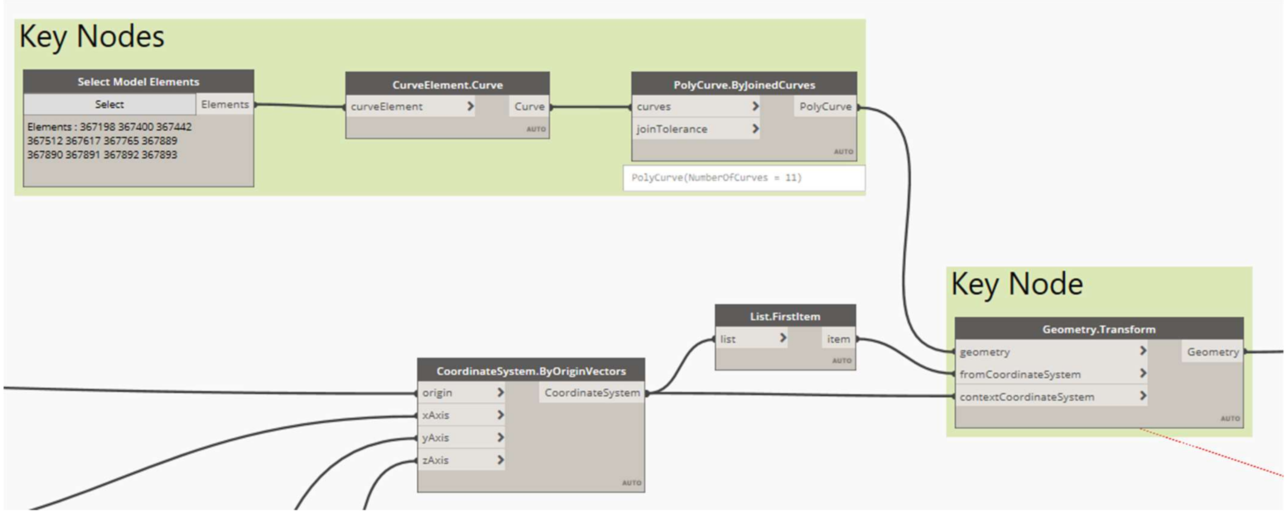


One creates a polycurve for you from a Revit Profile family and the other sorts and groups an unordered list of curves into groups of joinable curves for a polycurve. Quite nifty, If you want to support Konrad Sobon who created these go to his Patron page here: https://www.patreon.com/archilab

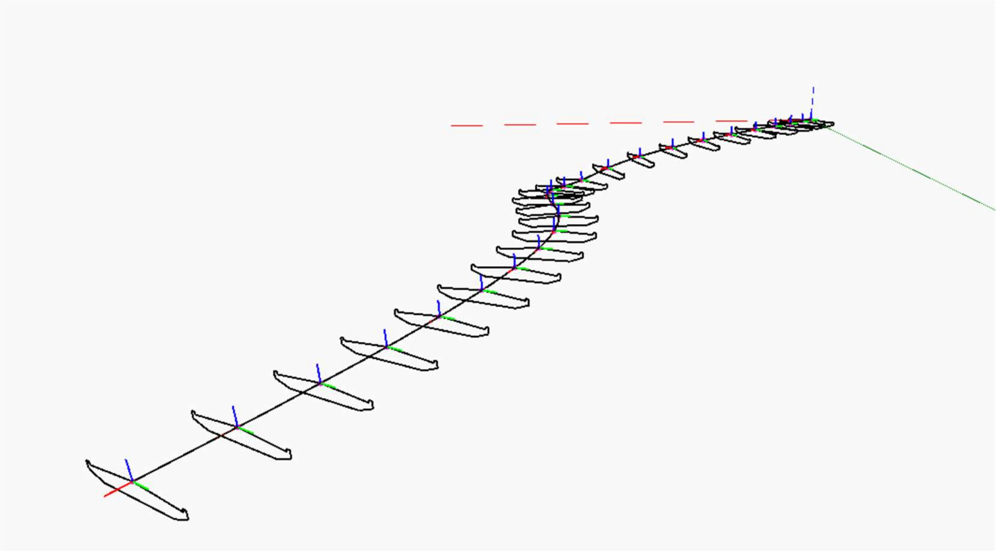
So, to get to it the key nodes we need are first these for extracting a polycurve from model lines in Revit:



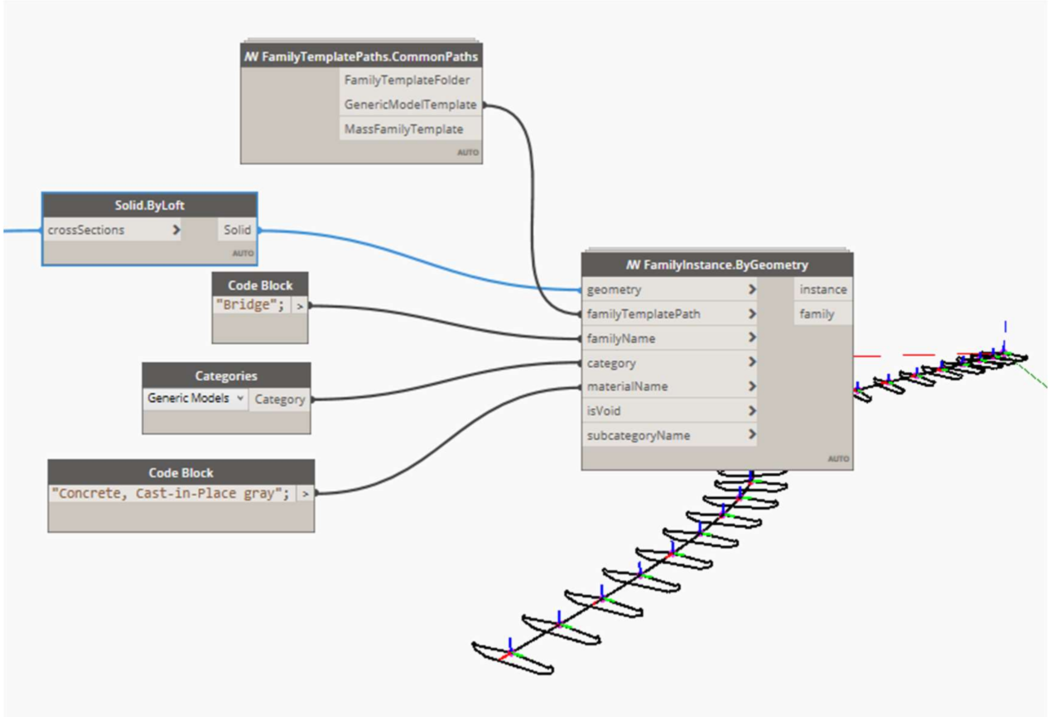
And then the **Geometry.Transform** for copying the polycurve from the first CS to the rest:



Resulting in this:



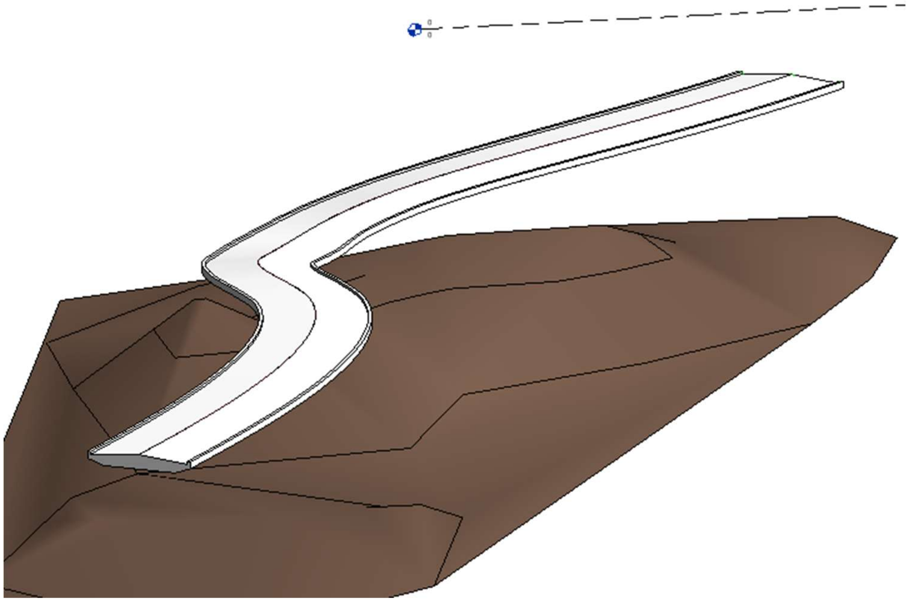
Now, from here there are a lot of options. You can loft all the profiles together, you can loft between two and two, you can do surfaces, solids etc etc. For this example, we'll do with creating a continuous Solid and use spring nodes **FamilyInstance.ByGeometry** package.



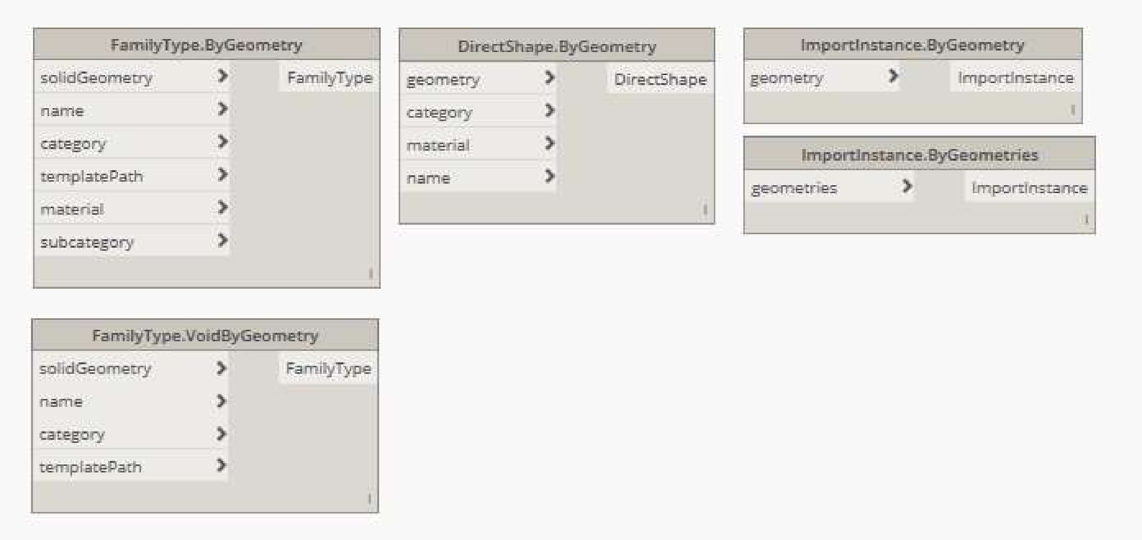
```
"Bridge";
```

```
"Concrete, Cast-in-Place gray";
```

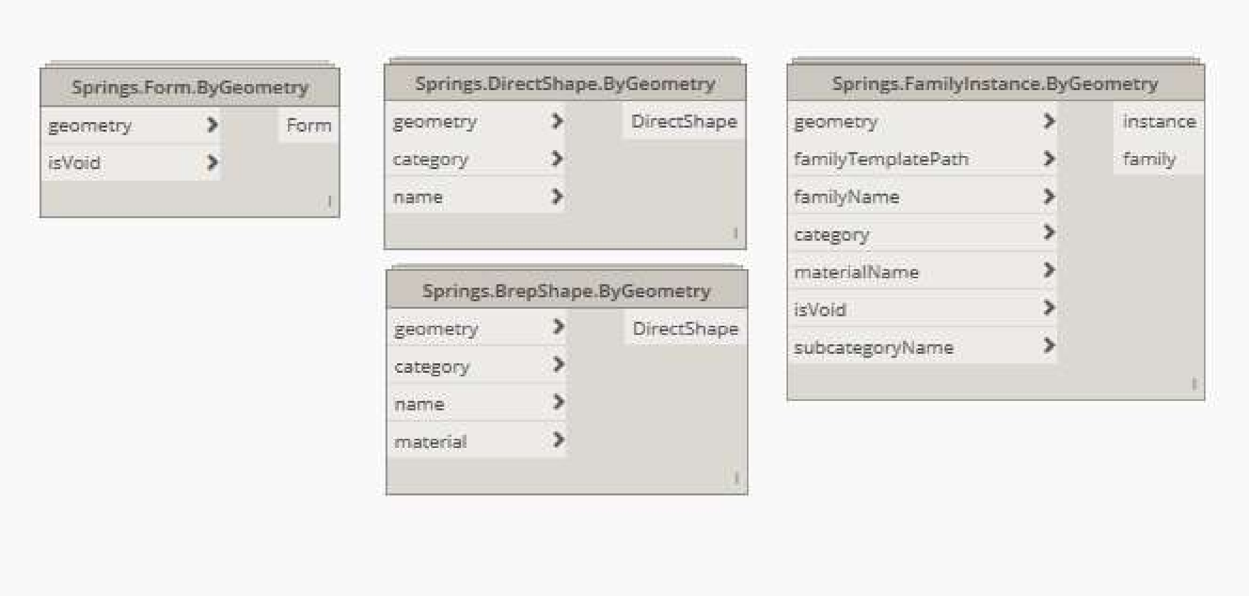
Resulting in this:



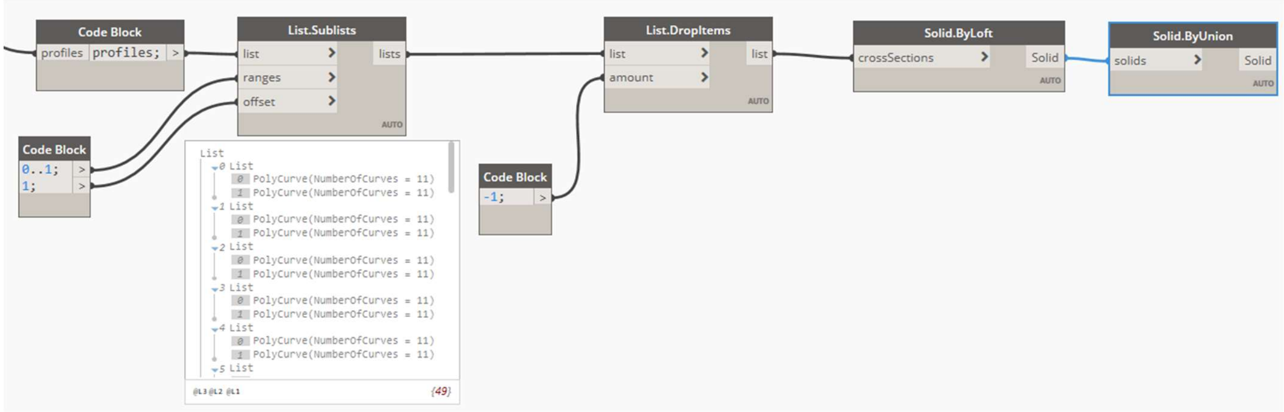
Other import possibilities for Solids:



But I really recommend spring nodes. A lot of options there that are more stable on a general level.



NOTE! Dynamo doesn't have settings for the lofting, like Grasshopper. You may find yourself in a situation were lofting between two and two profiles and then join into one solid in the end will provide better tolerances with the profiles. It that case you can do it like this:



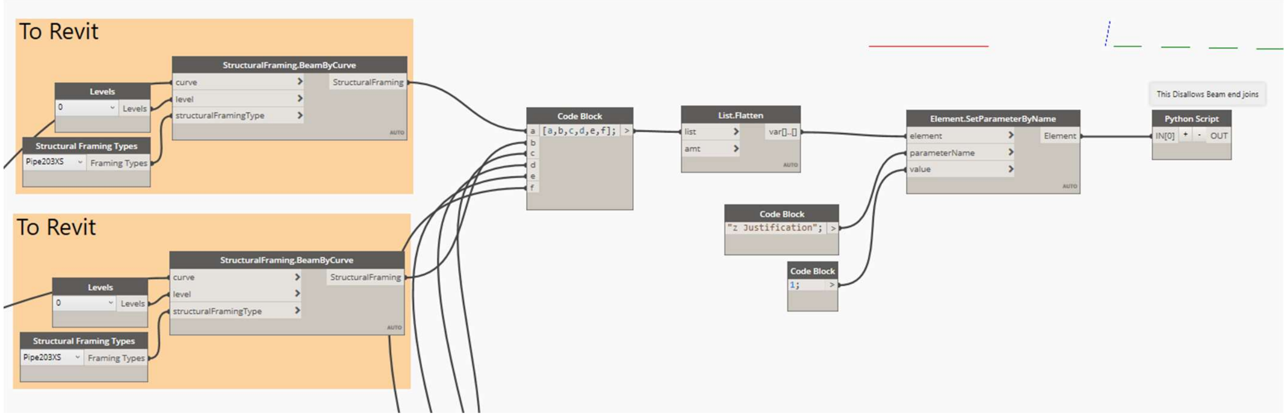
```
List
 0 List
   0 PolyCurve(NumberOfCurves = 11)
   1 PolyCurve(NumberOfCurves = 11)
 1 List
   0 PolyCurve(NumberOfCurves = 11)
   1 PolyCurve(NumberOfCurves = 11)
 2 List
   0 PolyCurve(NumberOfCurves = 11)
   1 PolyCurve(NumberOfCurves = 11)
 3 List
   0 PolyCurve(NumberOfCurves = 11)
   1 PolyCurve(NumberOfCurves = 11)
 4 List
   0 PolyCurve(NumberOfCurves = 11)
   1 PolyCurve(NumberOfCurves = 11)
 5 List
@L3 @L2 @L1                        {49}
```

Or maybe using Rhino.Inside.. 😉

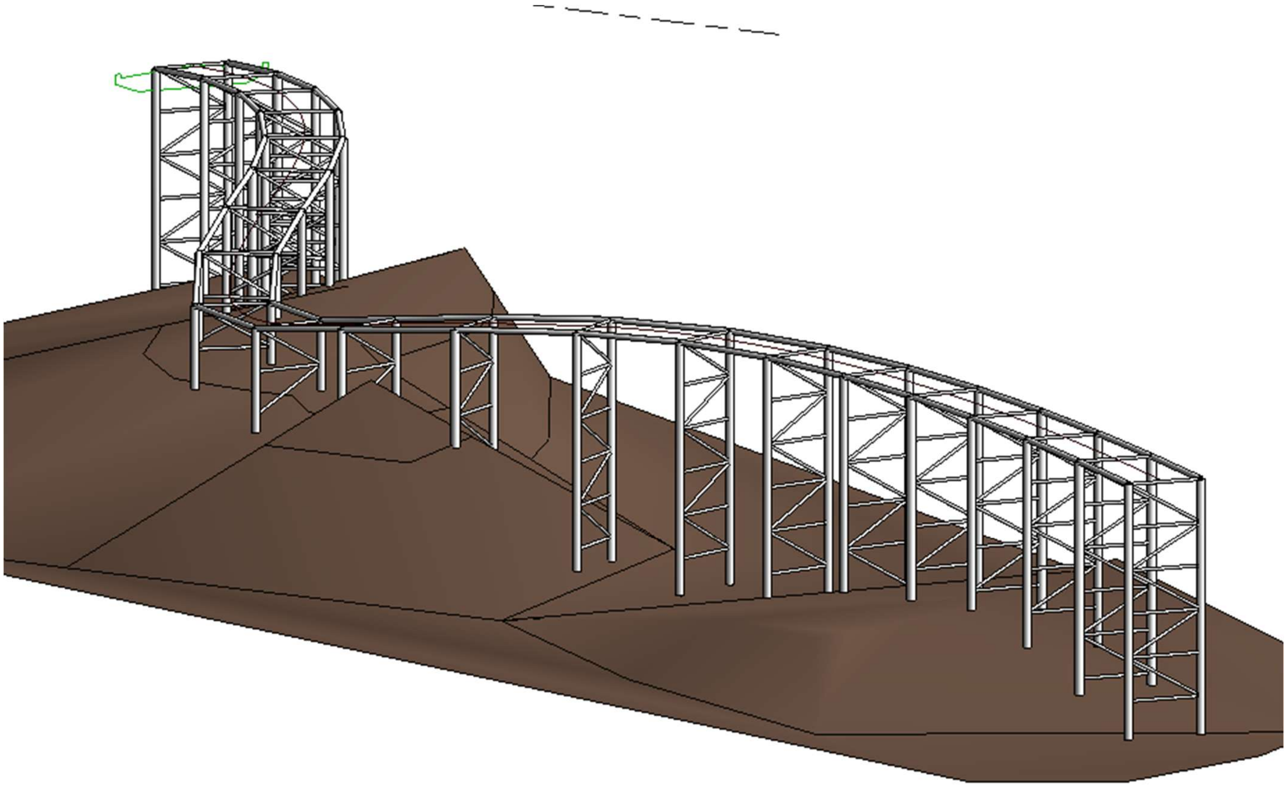
5.3 CS and Discrete Geometry

As a last example I just wanted to do something fun to show you the power of parametric modelling and the power of using coordinate systems in Dynamo/Revit/Infrastructure environment. I'm not going to go through this in detail, try to just follow the data stream and you'll get it. It is all about building step by step carefully planning which elements should depend on each other so you'll get a full parametric model that can handle changes in inputs.

In short, there is nothing new in this script, just basically a lot of what we have gone through put together in a large script. The only new introduction worth mentioning are the creation of beams in Revit. What I do below is that I create the beams using my Dynamo curves and then set the beam parameters so that the Z Justification is at centre and then disallow joins on the beam ends.

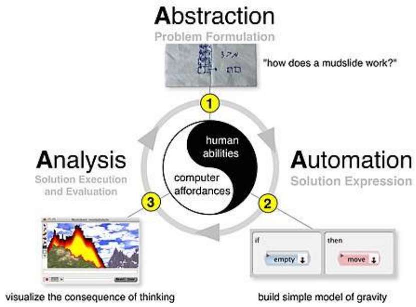


So, if you got it down you should see something like this:



6 Closing Remarks

Hopefully, you have learned something from reading this handout. If even just the Tips and Tricks in green gave you something, I'm happy.
If you're just starting to apply visual programming or have done it for a while it is worth mentioning that they aren't just a new set of tools like picking up Revit or Microstation. They also represent learning yourself a different frame of mind, **Computational Thinking**. (Again, go to my blog at jbdynamo.blogspot.com and read last year's BILT EUR handout and see why I think the mere computational thinking part is worth your penny.)


Abstraction
Problem Formulation
"how does a mudslide work?"
1
human abilities
computer affordances
Automation
Solution Expression
2
if
empty
then
move
build simple model of gravity
Analysis
Solution Execution and Evaluation
3
visualize the consequence of thinking

Now, in my opinion, the only way to get to thinking computationally, is by learning programming. Since it can prove hard for us engineers/architects/designers to learn programming on such a level that it is actually useful, Visual Programming is the way to go to increase your computational thinking.

However, it is also necessary to keep in mind that if you have a hammer, you'll see everything as a nail. Even though, sometimes, it is a screw. Dynamo isn't perfect, hell, even Grasshopper isn't perfect, even though I keep hearing fanboys screaming on either side.
The important part though is that your mind will be structured in such a way that after a while you'll easily identify what kind of work that's perfect candidates for some automation so you can work smarter and not harder. Be it through 3 party software vendors or internal visual programming champions.

I believe that picking up and learning some visual programming aids in identifying those tasks in our fragmented and complex Infrastructure projects. And therefore, I hope you have enjoyed the lab and enjoyed the handout. Please let me know if there are any questions! 😊

Now, have a good one!