Civil Systems Engineering

Modeling Civil Engineered Systems

Civil Systems Engineering – Modeling Engineering Products

Individual Assignment II – Parametric Modeling

studentsupport@civilsystems.tu-berlin.de

20 Portfolio Points

Released: 26.09.2025 Due: 12.01.2026 (23:59)

The goal of this second modeling assignment is to understand the relation systematically and logically between a number of important parameters of your engineering product, its geometrical configuration and embodiment and a set of high-performance criteria. As discussed in the lectures, the definition of well thought out parametric models allows for the very quick generation of possible design alternatives. Quick generation, in turn, has the potential to allow engineers to quickly explore design spaces, find optimal solutions within these design spaces, and to ensure that pre-given design constraints are accounted for. On the downside, the danger while creating parametric models is that they can quickly become a very complex and entangled mesh of parameters that is hard to explain, understand, and work with.

Specific Tasks:
To complete this assignment, you will need to complete the following tasks:

1. For your civil engineering product choose a design challenge that requires balancing two or more design parameters and that influence your products physical embodiment. To this end, select at least two high performance criteria which are used to assess the performance of your model.

2. Model the system with Dynamo BIM. To get an introduction you can follow the Dynamo BIM tutorials online and follow our bridge tutorial. Make sure that you keep the geometric embodiment as simple as possible. Please do not underestimate the complexity of the software solution. Getting acquainted with Dynamo BIM will require some intensive effort.

3. Experiment with the model and evaluate your design space. Find a number of minimum three good alternatives and present these in your report underlying the engineering aspects.

Assessment:

When grading we are expecting at identifying the following points: (1) complexity of the model, (2) knowledge encapsulated, and (3) engineering rationale. On top of the above, clarity of the written report, expected for a master's level student, represents grading criteria.

Written report (20 Portfolio points)
Please submit a written report of maximum four pages that summarizes your identified design challenge, selected high performance criteria, and the related parameters you identified. Describe the logic of your parametric model. Then discuss the design space with its extremes and limits. Finally, discuss your identified good alternatives and explain why they represent well embodied solutions.

Expected deliverables:
Please submit your report in pdf format. Also, submit your Dynamo file. Upload these two files through the ISIS website. Make sure you submit everything by the deadline shown on ISIS. All files should allow us to easily identify your student number and your name. They also should be professionally formatted.

IMPORTANT NOTES:

PLEASE DO NOT SUBMIT .ZIP FILES. PLEASE DO NOT SUBMIT WORD DOCUMENTS. PLEASE MAKE SURE TO PROPERLY SUBMIT YOUR WORK SO THAT IT IS NOT A "DRAFT." WE CANNOT GRADE DRAFTS AND WE WILL NOT GRADE LATE SUBMISSIONS WITHOUT PRIOR PERMISSION BASED ON THE OFFICIAL REGULATIONS.

Please do not hesitate to contact us at any time (studentsupport@civilsystems.tu-berlin.de). Often, we can solve problems and questions you have rather quickly, in particular, if they are related to software. Please also make use of our „face-to-face" work sessions in the computer cluster 180 (Mon 14:15 – 15:45 and Wednesday – 10:15 – 11:45).

We wish you fun and luck with the assignment!

Civil Systems Engineering

Modeling Civil Engineered Systems

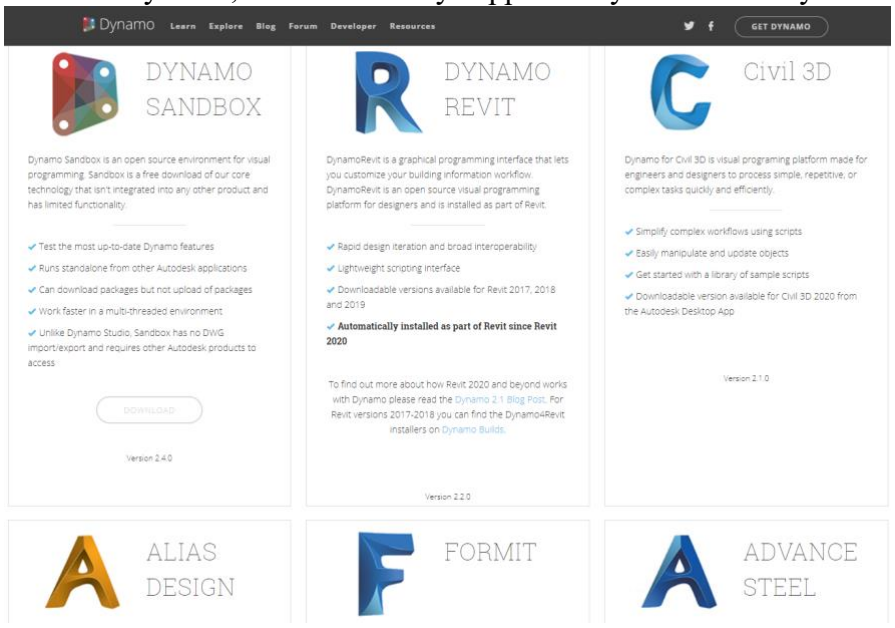
Bridge Tutorial

Parametric model generation with Dynamo

studentsupport@civilsystems.tu-berlin.de

Getting started with Dynamo BIM

Installing Dynamo BIM

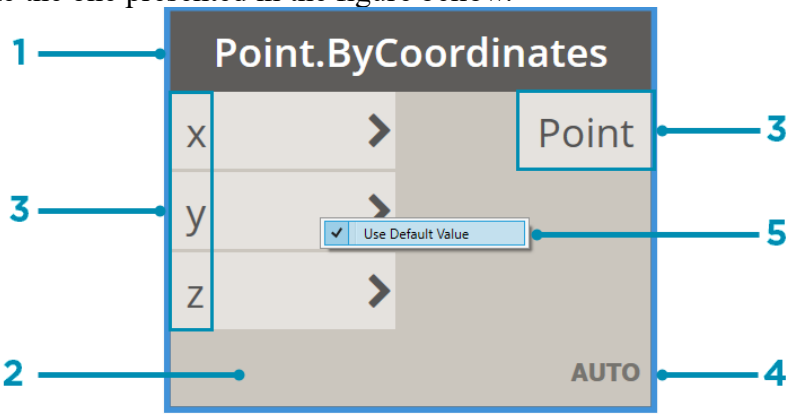
Step	Instruction
1.	Go to https://dynamobim.org/
2.	Go to Download section (https://dynamobim.org/download/) by using the Get **Dynamo Button.** There are various paths you can follow to install Dynamo, as it is currently supported by a wide variety of design tools developed by Autodesk. 

3	Our recommendation is to work with Dynamo that is built in Revit. Latest version of Revit is already installed on the computers in our PC-Pools, which you can access with your student IDs. You can of course install it on your own PC but we cannot guarantee support for your personal laptops.

Working with Dynamo

Dynamo is a tool for parametric and computational design. Dynamo Interface is not complicated, and it is quickly explained in the following page:
https://primer.dynamobim.org/02_Hello-Dynamo/2-2_the_dynamo_ui.html

A typical node in Dynamo usually looks like the one presented in the figure bellow.



The top part of the node usually represents the name of it. The left side is dedicated to the input ports while the right side is reserved to the output ports. Double-clicking on the name of the node allows you to change its name with something more representative for the definition you are creating. Right-clicking on the node will enable you to access some additional functionality related to the node. The lacing represents one important aspect. We will be making use of this functionality during this tutorial so that it can be introduced to you together with some examples.
More details are provided in the following page from Dynamo Primer where the above image was taken from.
https://primer.dynamobim.org/03_Anatomy-of-a-Dynamo-Definition/3-1_dynamo_nodes.html

Parametric definition of a bridge

Defining the design theme

Bridge parametric models – argumentation on the selection of the parameters

The domain we are focusing the effort of creating a parametric model is Arch-tied Bridges.
The main idea which represents the foundation of developing a parametric model for an arch-tied bridge is to support the structural analysis.

The following parameters were selected as the ones to control the parametric model:
- length of the bridge (L);
- width of the bridge (W)
- ratio of height of the arch (H) to the bridge's length(L) - H/L between 1/5 and 1/65
- ratio of transversal span(T) to the bridge's length (L) – T/L
- deck elevation (h)
- inclination angle for the arches
- inclination angle for the hangers (cables)

What's the purpose of these parameters?

The length of the bridge is the main parameter considered for structural analysis. This parameter influences most of the decisions taken during bridge design. Having length as a controlling parameter will allow us to use the parametric model for various site-related conditions. Of course, once the location of the future bridge is selected, the length hardly changes.

A first relation to be highlighted is the one with the height of the arch. (H/L). An optimum ratio between height(H) and length(L) will reduce the dead loads, axial deformations and leads to an efficient use of the materials.

Bridge's width, together with the length and the structural systems of the deck will impact the sizing of the primary elements (main girders and arches) by having an impact on the size of the dead load. The width of the bridge is usually dependent on the regime of the road on which we want to design the bridge. Once the road's regime is defined, this parameter hardly changes.
Spacing of cables (hangers) → T/L ratio
- from a structural point of view, it is desirable to space the cables as close as possible.
- from an economic point of view, it is desirable to space the cables as far away as possible.
- Besides the two factors mentioned above, aesthetics factor plays an important role since bridges are pieces of art that should integrate seamlessly

with their environment.
- another parameter related to hangers is their inclination in relation to main girders. Structural engineers will need to vary the inclination so that they find the optimal orientation of the cables to reduce the dynamic effects given by cars when crossing the bridge.

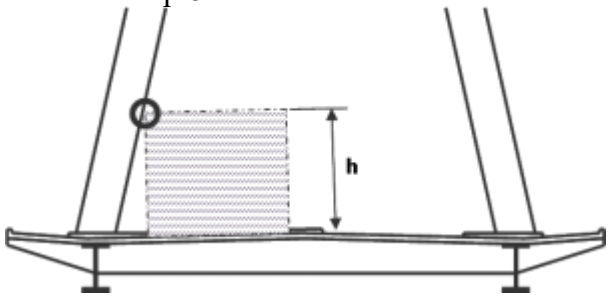
The spacing of cables is an important factor on selecting the deck system.

The inclination angle for the arches will allow us to explore various configurations such as the ones in A, V or H.

Multiple solutions generated using the parametric model will serve as input for the structural analysis.

To keep a low level of details, the main elements we are going to model are: longitudinal girders, arches, ties (cables or hangers), elements between arches, and the deck system.

Before starting, it is essential to establish some high-performance criteria, so that we can measure the performance of our design. In the case of inclined arches, one of the main factors we need to consider is the **headroom clearance** and the used width of the bridge, as shown in the figure below. Please see step 8 for this.



Another aspect we need to consider is the **quantity of materials** used. This requires running the structural analysis, but for this tutorial, we will implement only a basic one. Please see step 9 for this.

Defining the input parameters

For each input parameter, the following data is predefined/preset:

Input Parameter	**Min. value**	**Max. Value**
Bridge Length (L) - meters	100	250
T/L	1/20	1/10

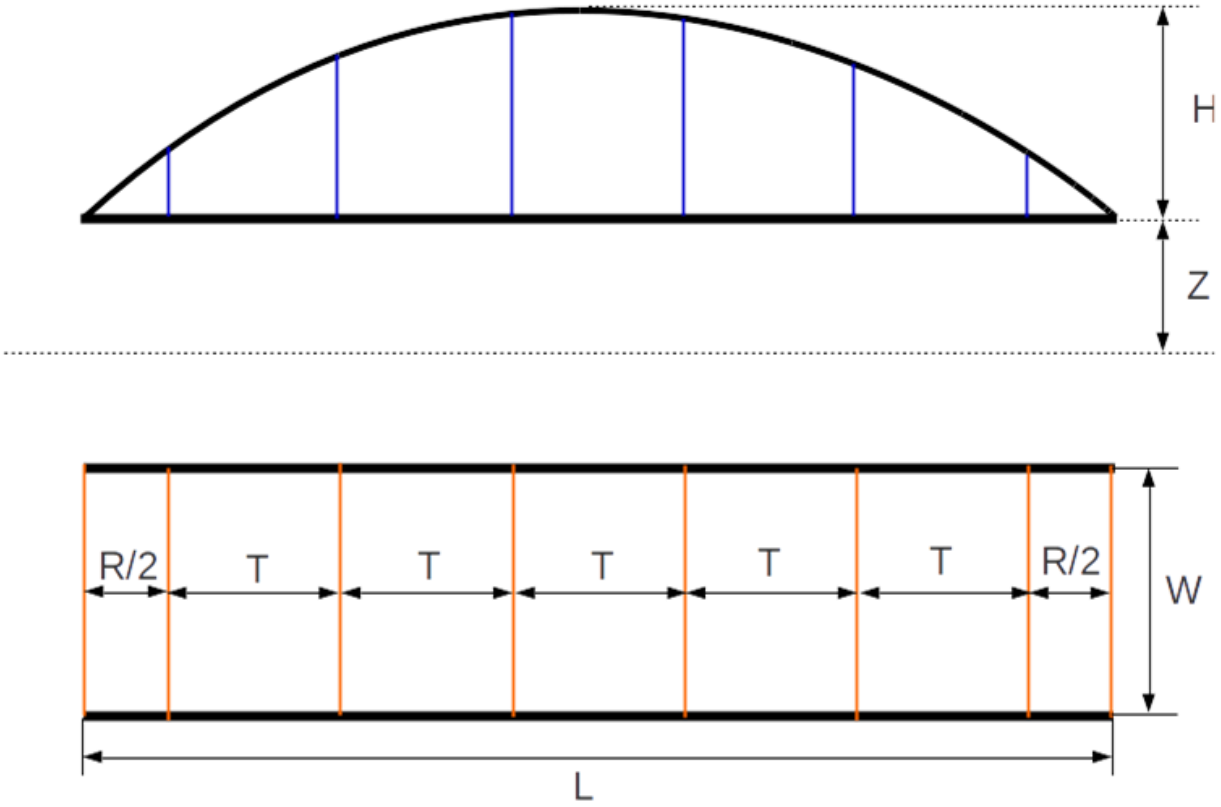
Bridge Width (W) - meters	7.5	25
Bridge Deck Elevation (Z) - meters	0.00	20
H/L	1/6.5	1/5

These values will be used to constrain the input values of the input nodes.

Creation of the parametric model

The parametric model will be created based on following mathematical guidelines.

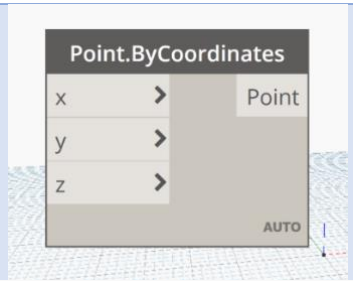
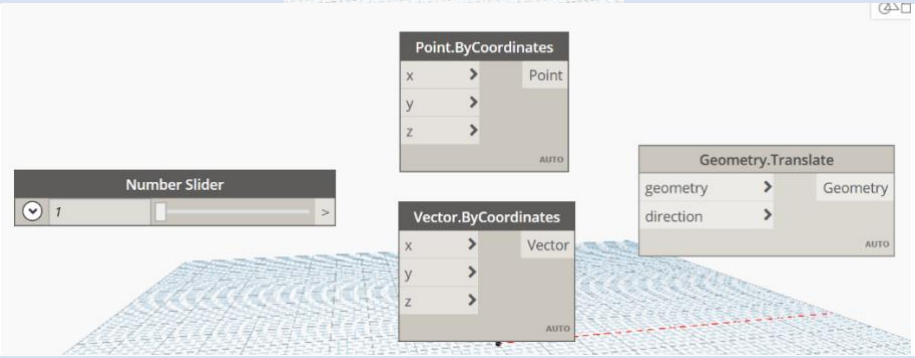
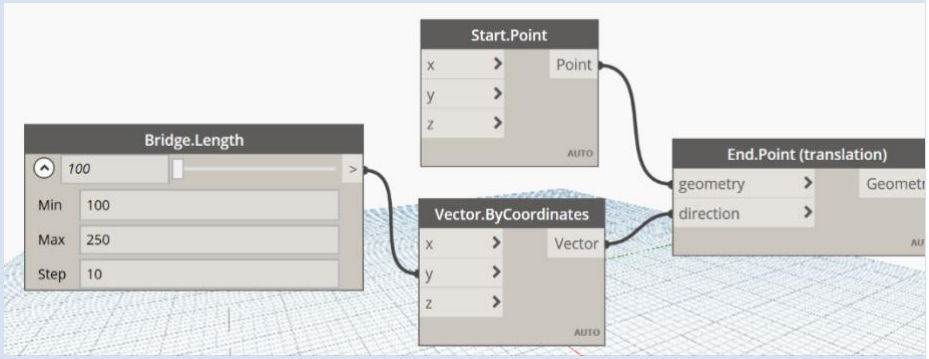
a. We are starting with 2 parameters: Bridge length (**L**) and transversal span (**T**) as shown in the figure below.

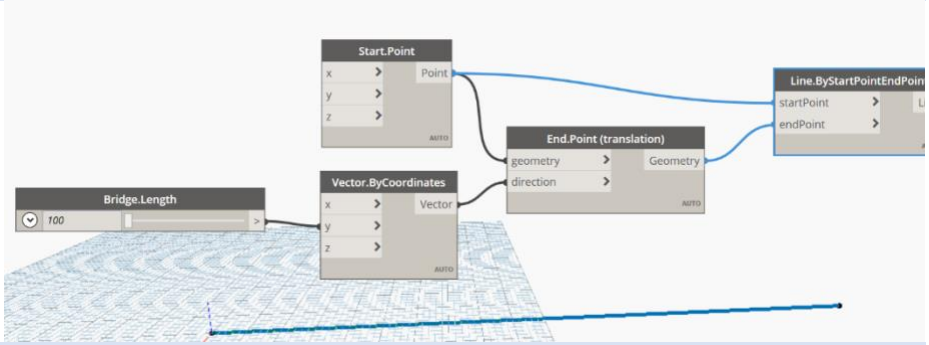
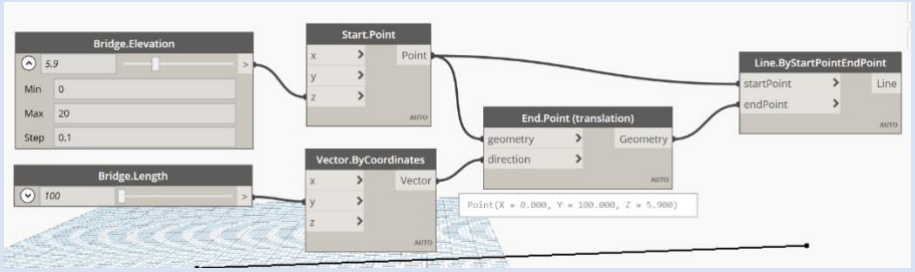
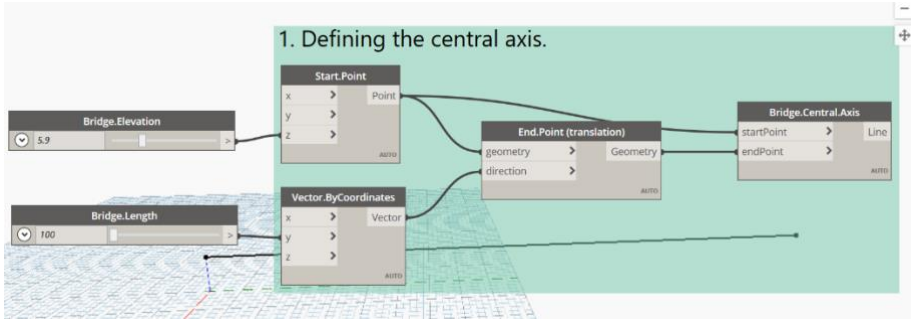


First equation we will use is: L / T = Q *T + R , where Q is the *quotient* and R is the *remainder*.
The quotient will give us the number of full spans having the spacing T. The reminder will be distributed evenly on both sides of the bridge as shown in the figure above R/2.

1. Defining the central axis of the bridge.

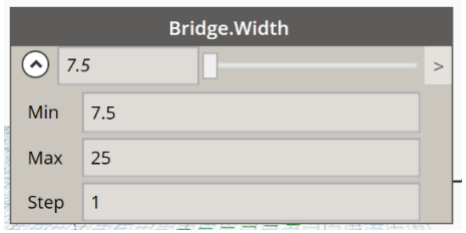
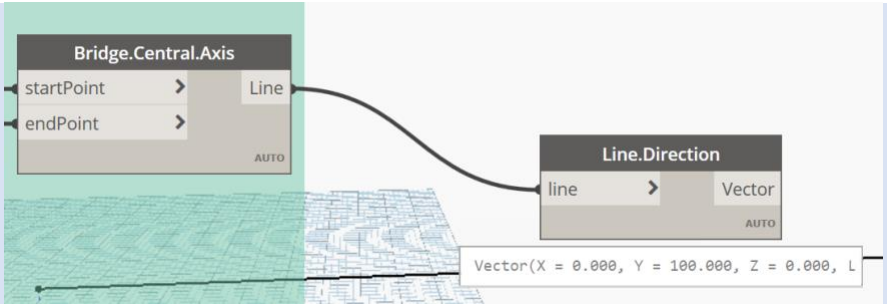
This step is also available as video tutorial: https://autode.sk/2Q8TPeG

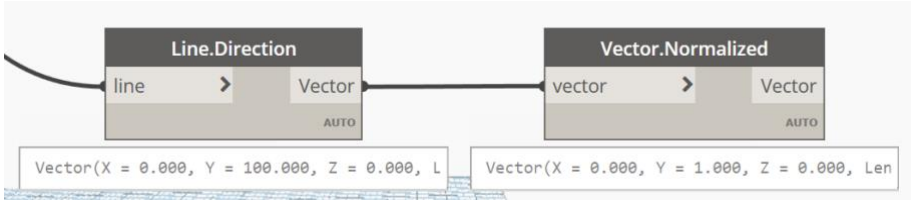
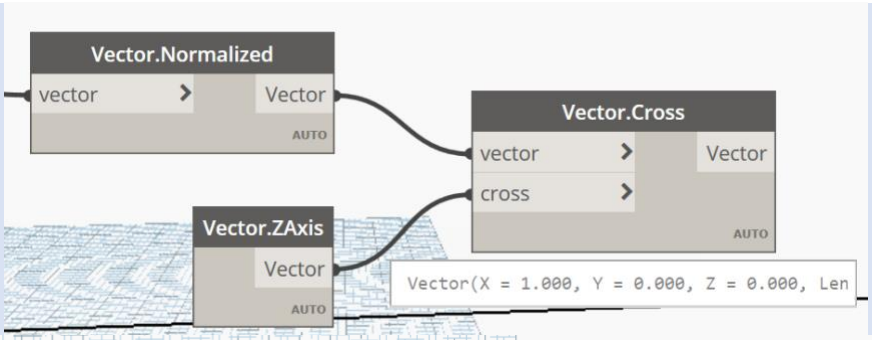
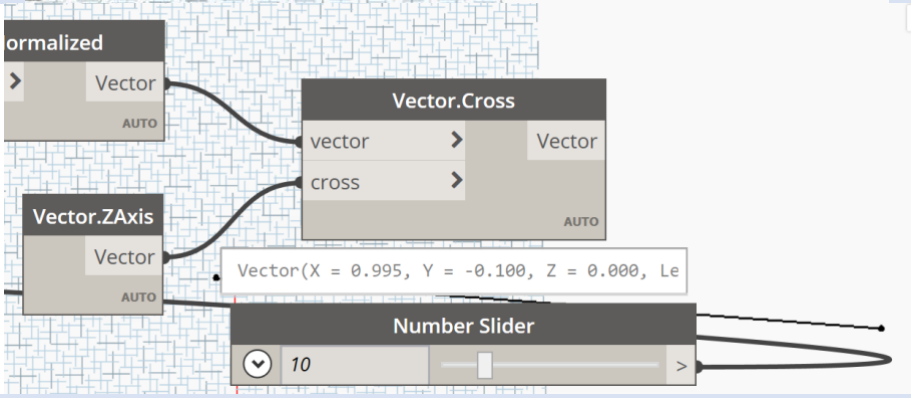
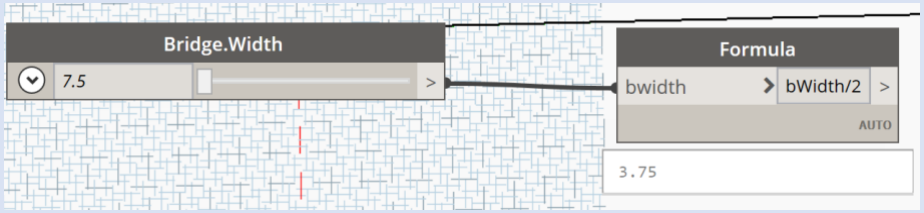
Step	Instruction	Result
1.1	Add a *Point.ByCoordinates* node. It can be found in the left-side menu under: *Geometry → Points → Point → ByCoordinates* *(x, y, z)*. By default, the point will have the coordinates (x, y, z) = (0, 0, 0).	
1.2	Instead of defining a new point, use the point created at step 1.1 and use the translate operation. For this, add the following nodes: *Geometry → Abstract → Vector → ByCoordinates* *(x, y, z)*. *Geometry → Modifiers → Geometry → Translate* *(direction)*. *Input → Basic → Number Slider.* After adding these nodes, your screen will look as the one shown under results column.	
1.3	At this step, we will perform some changes, so that you will be able to identify the purpose of the nodes, later when coming back to it. Double click on the *Number Slider* node and change its name to *Bridge.Length.* For the same node, click on the arrow from the left side and add the following values: *Min: 100; Max: 250; Step: 10*. Change the name of the *Point.ByCoordinates* to *Start.Point* and the name of the *Geometry.Translate* node to *End.Point (translation).* Connect the nodes as indicated in the image in the results column. At this point notice the additional point which appeared in the	

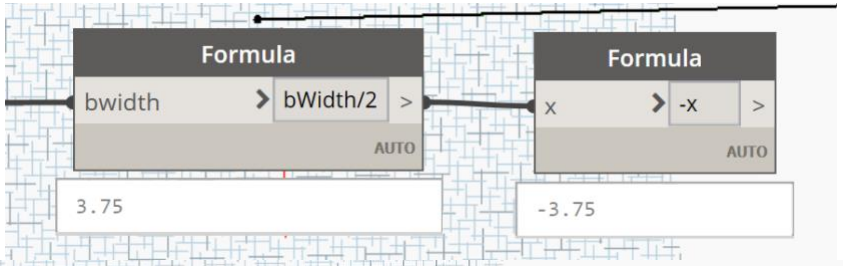
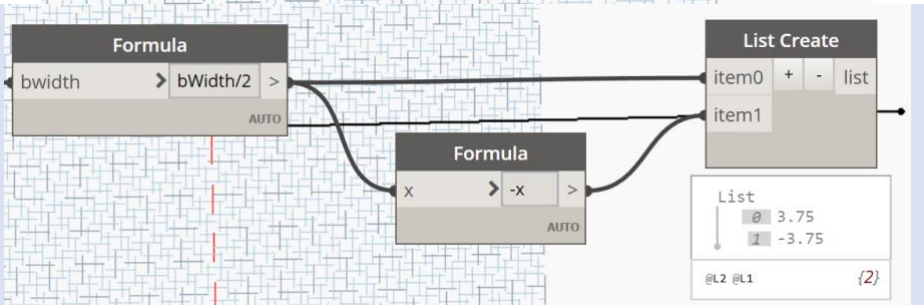
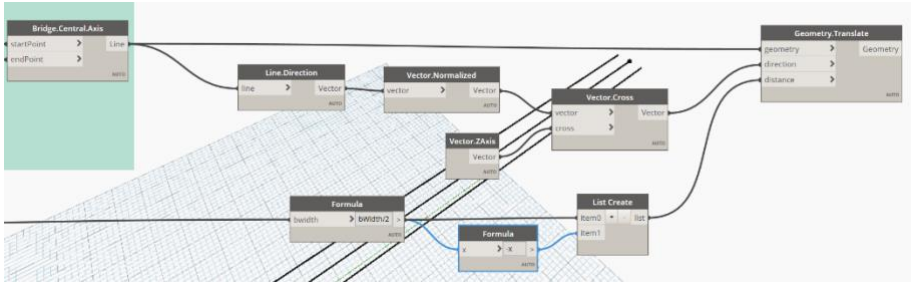
Step	Instruction	Result
	Background preview.	
1.4	To complete the creation of the central axis of the bridge, add a node to connect the two points with a line. *Geometry → Curves → Line → ByStartPointEndPoint.* Make the connections as show in the figure. You noticed the line showing up in the background.	
1.5	Add one more input slider for numerical values. *Input → Basic → Number Slider.* We will name it Bridge.Elevation and we will provide the following setup for it: Min: 0; Max: 20; Step: 0.1. Connect its output port to the z input port of the Start.Point node. Once connected, as you start changing its value notice that the End.Point, which we obtained as a translation of the starting point inherits the elevation. This is the behavior we want to achieve.	
1.6	To keep the definition organized, select all the nodes, except the input sliders and use the key combinations *CTRL+G*. This will create a colored transparent frame around the nodes called group. *Double click* on the title and change it to *1. Defining the central axis*. *Right clicking* on it, you can access more settings such as changing its color or adjusting the text size. Another cool functionality is the possibility to automatically arrange the nodes of the group. *Right click* on the group and select *Cleanup Node Layout*. Also, notice that if you *click and drag* the frame, everything included within its boundaries will move as desired.	

2. Create the side lines of the bridge

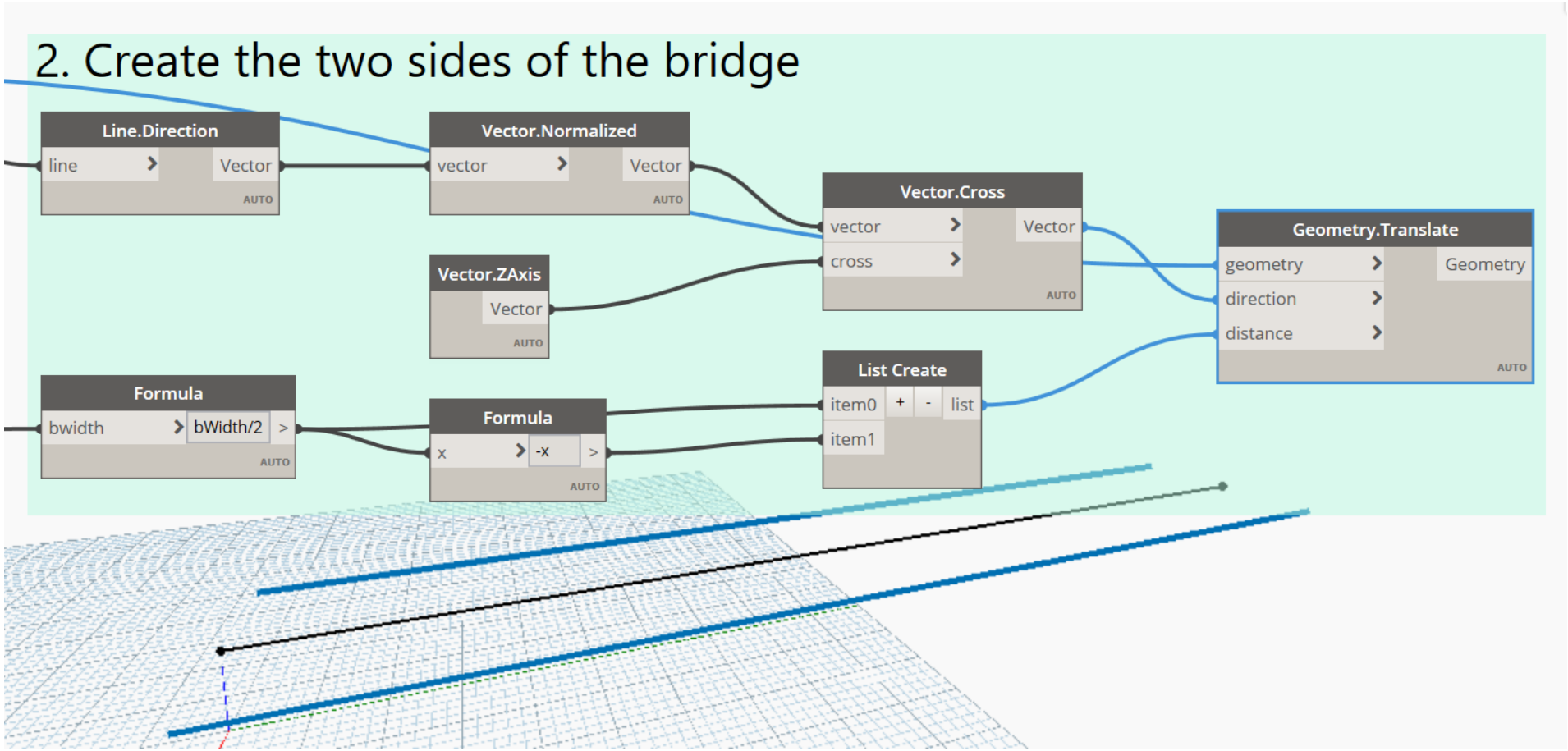
This step is available also as video tutorial as following: https://autode.sk/2NBRmYz

Step	Instruction	Result
2.0	Now that the central axis is defined, we can offset it to create the side lines of the bridge based on the *Bridge.Width* parameter. The behavior we want to achieve is: - Using the central axis, offset it on both sides of it with *Bridge.Width*/2; - We want to offset to be always parallel on the offset axis; - And of course, to inherit the details of the already defined geometry;	
2.1	First, we add a new input slider for the *Bridge.Width* parameter. *Input* → *Basic* → *Number Slider.* The parameters of this node are defined as following: Min: 7.5; Max: 25; Step 1; Name: Bridge.Width. Usually, this is a parameter calculated based on the road regime and the traffic load. For the purpose of the tutorial, it is defined numerically, and in the last assignment this information is provided by the road engineers.	
2.2	To make sure that the offset stays parallel to the bridge axis, extract some information from the already defined line. We will add the node *Line.Direction* from: *Geometry* → *Curves* → *Line* → *Direction* and connect the output port of the *Bridge.Central.Axis* node to its input port as shown in the figure. This returns a vector which represent the normal of the line.	

Step	Instruction	Result
2.3	The vector cannot be used as it is, as it has the magnitude of the Y coordinate equal to the length of the bridge. We need to normalize it, and for this we add the node *Vector.Normalized* from: *Geometry → Abstract → Vector → Normalized* Once connected with the vector extracted from the line, the magnitude of the Y coordinate is normalized.	
2.4	Next step is to use a math trick to make sure that the offset direction will be always perpendicular on the normal of the line, hence the line will be parallel. For this, add two more nodes: *Vector.Cross* and *Vector.ZAxis*. *Geometry → Abstract → Vector → Cross* *Geometry → Abstract → Vector → ZAxis*	
2.5	Adding a dummy slider to and changing to 10 the X input of the *Vector.ByCoordinates* node we added at step 1.2, you can see that now the values of the vector, perpendicular on the normal vector of the central axis changes accordingly.	
2.6	Make use of the *Bridge.Width* parameter defined at step 2.1 to define the magnitude of the offset as defined at point 2.0. For this, add *a Formula* node from: *Math → Functions → Formula* and enter the formula **bWidth/2**. Then connect the output of the *Bridge.Width* node the input port of the formula node called	

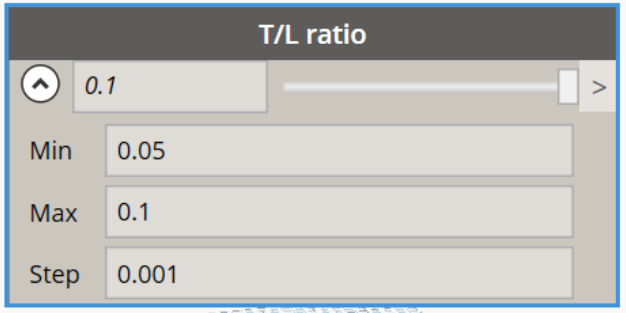
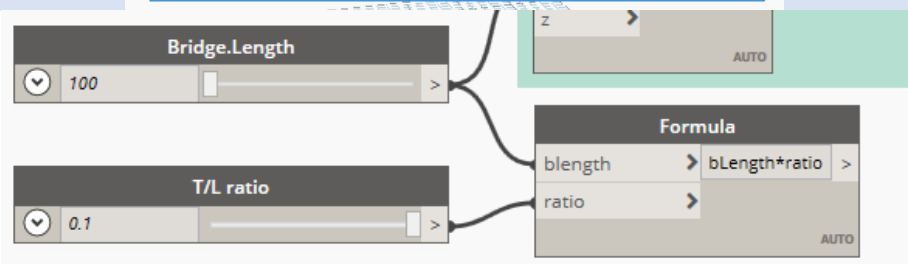
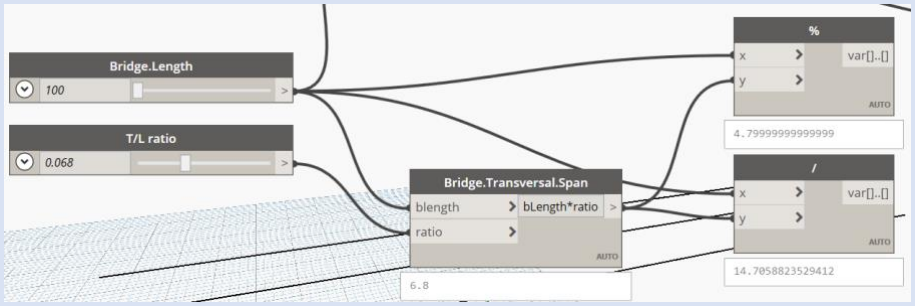
Step	Instruction	Result
	bWidth, as the variable we defined.	
2.7	To make the offset on both sides of the central axis, Add two values, such as: ***[bWidth/2, - bWidth/2]*** For this, add another *Formula* node and add the formula: *-x* and make the connection as shown in the picture.	
2.8	Use both values to perform the offset at once, and to do this, compile both values in a list. For this, add the node *List Create* from: *List* → *Generate* → *List Create* Click on its plus button to add one more input ports. Make the connections as show in the figure.	
2.9	To perform the offset, use a *Geometry.Translate*. This will be a bit different than the node added at step 1.2 because will allow you to provide as input the direction and the distance. *Geometry* → *Modifiers* → *Geometry* → *Translate* *(direction, distance)*. The geometry port will take as input the line of the central axis The direction port will take as input the vector we got from the cross product The distance port will take as input the list of distances we defined at step 2.8. You can already notice the two new lines showing up on the background on both sides of the central line.	

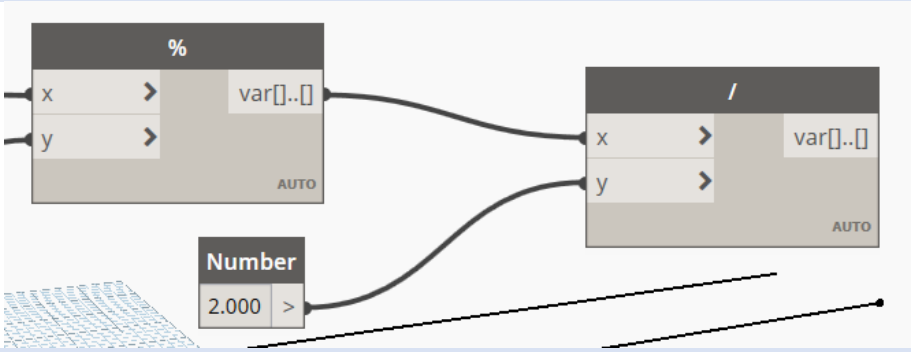
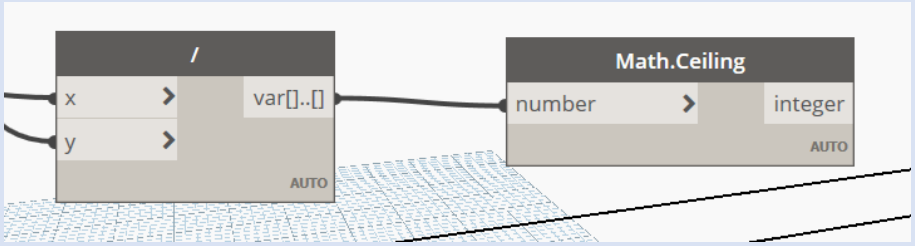
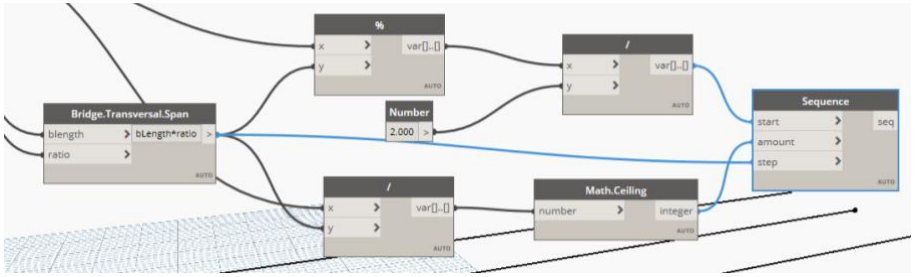
Step	Instruction	Result
2.10	To finalize this, group the nodes added at this step and name the group: *2. Create the two sides of the bridge*. The result is shown below.	

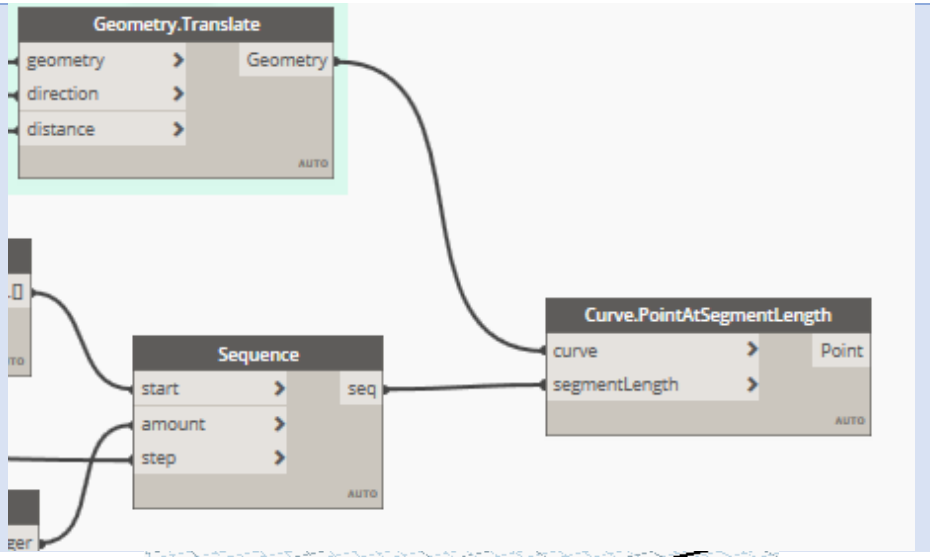
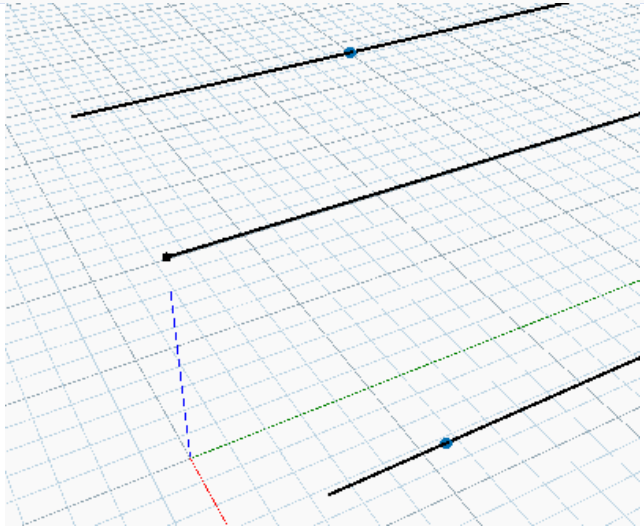


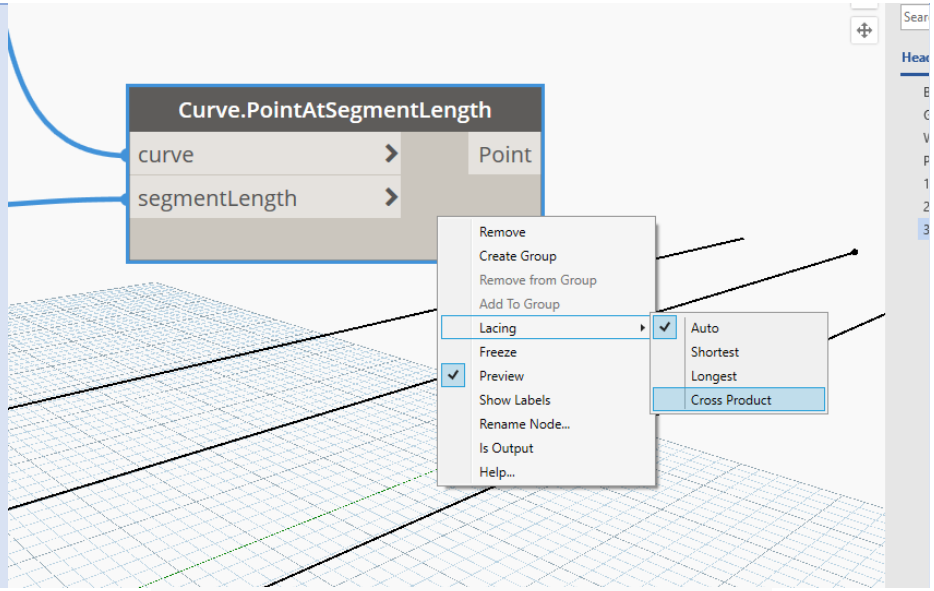
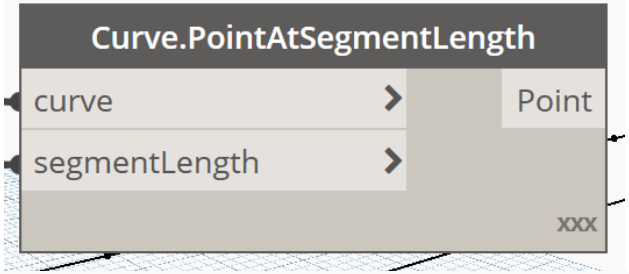
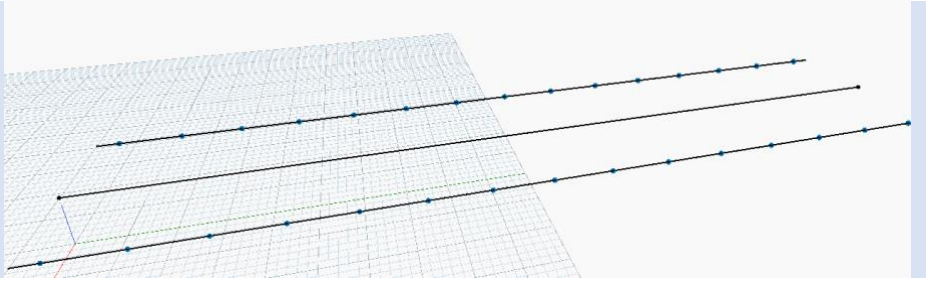
3. Defining the distribution of the transversal girders

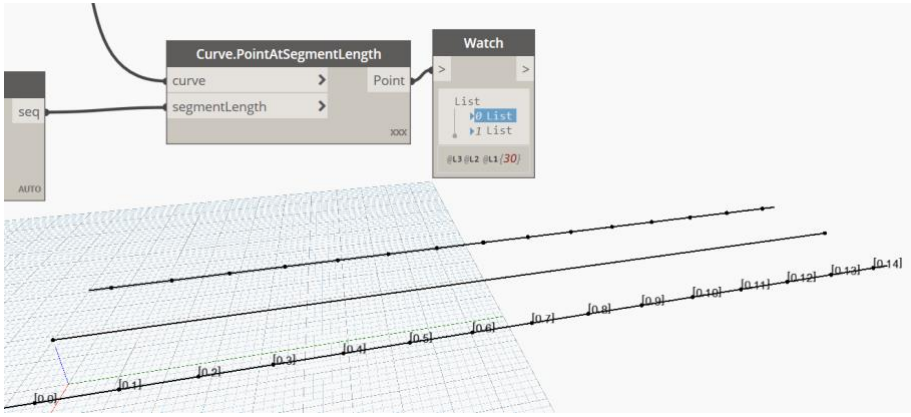
This step is also available as video tutorial as following: https://autode.sk/2NBdsul

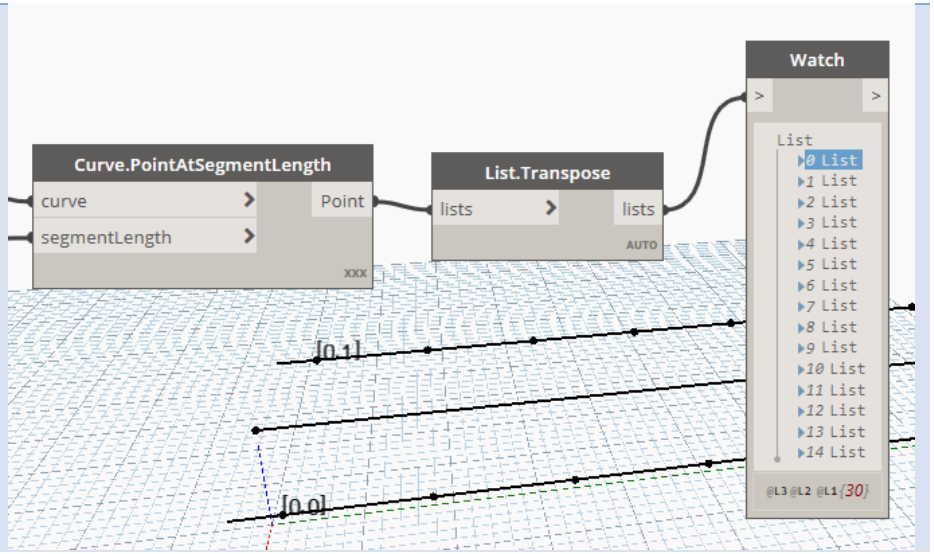
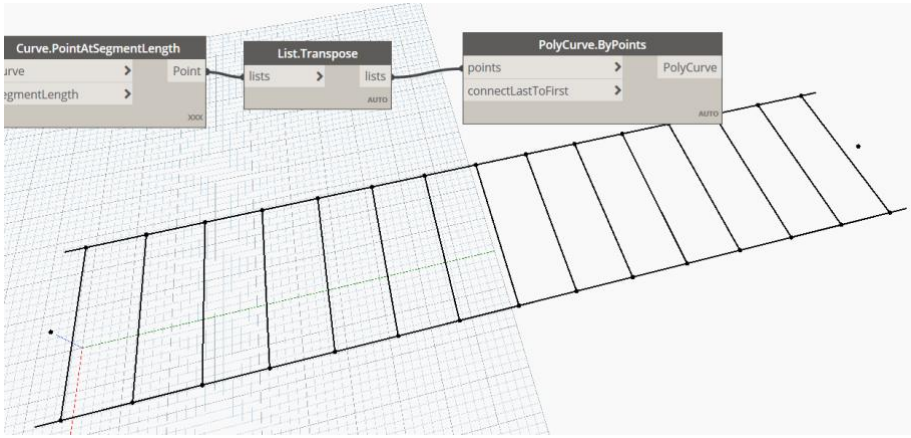
Step	Instruction	Result
3.1	The distance between the transversal girders(T) of the deck is defined by the ratio: T/L = [1/10, 1/20] where L is the length of the bridge. First, add an input slider. *Input → Basic → Number Slider.* Setup this slider as following: Name: T/L ratio; Min = 0.05; Max = 0.1; Step = 0.001	
3.2	Next, add a *Formula* node to calculate the transversal span (T) using the ratio defined at step 3.1 and the length of the bridge with the following formula: *bLenght * ratio* Connect to it the output of the input sliders *Bridge.Lenght* and *T/L ratio* as shown in the figure.	
3.3	Next step is to embed a little engineering in our definition. The total length of the bridge will not always be perfectly divided by the transversal span. The reminder needs to be equally distribute on both ends of the bridge, while the full spans will be distributed in the central part of the bridge. In this step, we will add two nodes as math operators: "/" (division) and "%" (reminder). *Math → Operators → /* *Math → Operators → %* Connect the output of the *Bridge.Lenght* node to the *x* input port of both nodes and the output of the *Bridge.Transversal.Span* to the *y* input ports of the nodes.	

Step	Instruction	Result
	For the case when T/L ratio is 0.068 the reminder is 4.79 while the division is 14.70.	
3.4	To distribute the reminder on both ends of the bridge, we will add another division node (/). *Math* → *Operators* → */* We also add a simple input number node and at the value 2 to it. *Input* → *Basic* → *Number*	
3.5	From the division node added at step 3.3 you can extract various information. Using floor rounding you get the total number of spans needed. Using the ceiling rounding, you get the number of points we need to create the number of spans. The latter serves our purpose, so add a *Math.Ceiling* node. *Math* → *Functions* → *Ceiling.* Connect the output port of the division node added at step 3.3 to the input port of the *Math.Ceiling* node.	
3.6	Next step is to create a sequence of numbers. For this, use a Sequence node. *List* → *Generate* → *Sequence* Connect to its input nodes the following: *start:* the results from the *division* node from step 3.3; *amount:* the results of the *Math.Ceiling* node; *step:* the output of the *Bridge.TransversalSpan*.	

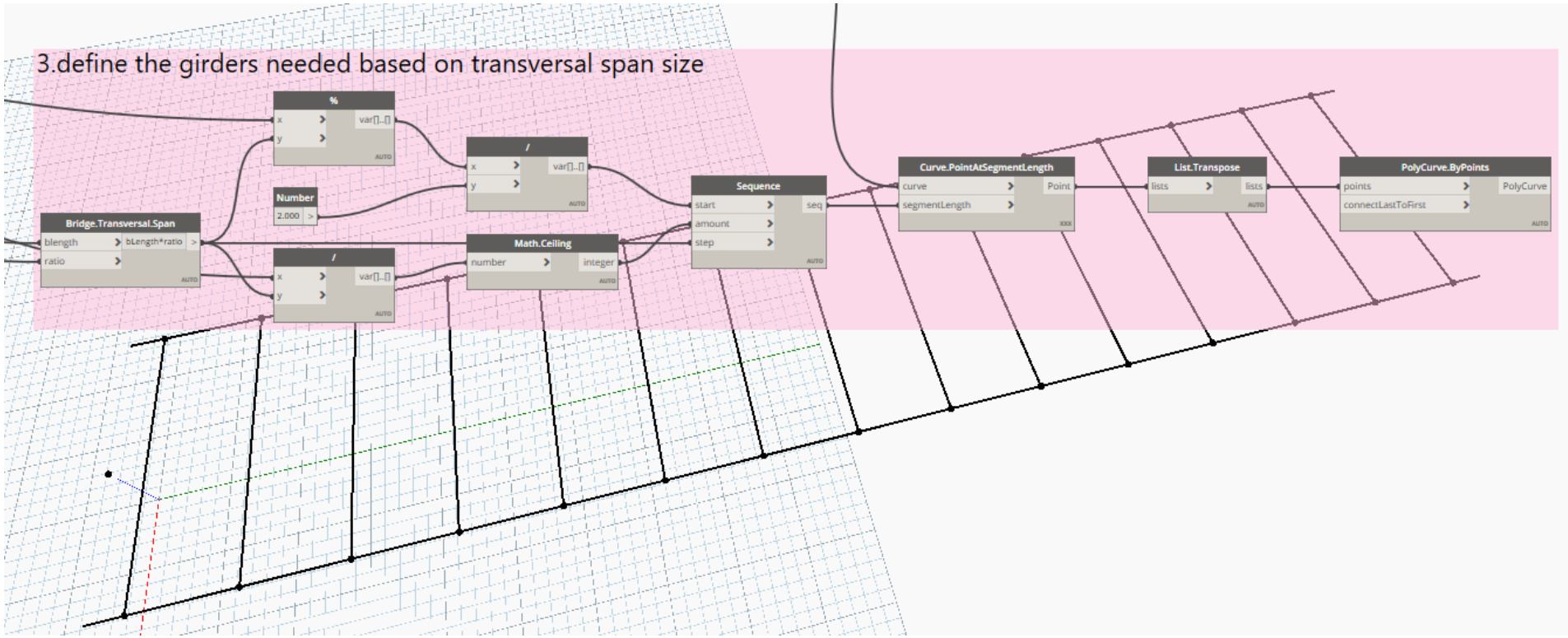
Step	Instruction	Result
3.7	Using this sequence, generate points on both lines defining the sides of the bridge (the ones generated at step 2.9). For this, add a node called *Curve.PointAtSegmentLength*. Geometry → Curves → Curve → *PointAtSegmentLength* Connect to its curve input port the Geometry output port of the *Geometry.Translate* node from step 2.9. To its *segmentLength* port connect the output port of the *Sequence* node created at step 3.8	
3.8	At this point, you might have noticed that only two points appeared in the preview. The intention is to create a sequence of points distribute over the whole length of both lines. For this make use of a cool feature called *Lacing*. For more detailed explanations of the concept visit the following web page. https://primer.dynamobim.org/06_Designing-with-Lists/6-1_whats-a-list.html	

Step	Instruction	Result
3.9	To access the Lacing functionality, right-click on the node and then navigate to Lacing and select Cross Product.	
3.10	First thing to notice is that on the bottom right corner no longer appear *AUTO* but xxx.	
3.11	Of course, one thing to notice is that the sequence of points distributed over the length of the lines provided as input.	

Step	Instruction	Result
3.12	One important aspect to get familiar with is the underlying data structure Dynamo uses. For more details, have a look here: https://primer.dynamobim.org/06_Designing-with-Lists/6_designing-with-lists.html You can easily visualize how data is structures in the output. Use a *Watch* node for this. *Display* → *Watch* → *Watch* This will allow you to visualize the numerical output of a node. In this case, you have a List Containing two other lists which each list contains 15 elements. As shown in the figure, you can select on a specific level to find out the elements which are part of the list. Once you selected the list at index 0, the position of each point within the list is indicated in the background preview. You can see that point [0, 0] is the first one followed by the point [0, 1] and so on. When you click on the other list at index 1, the notation changes to the other lines and it the numbering such as [1,0] [1,1] and so on.	

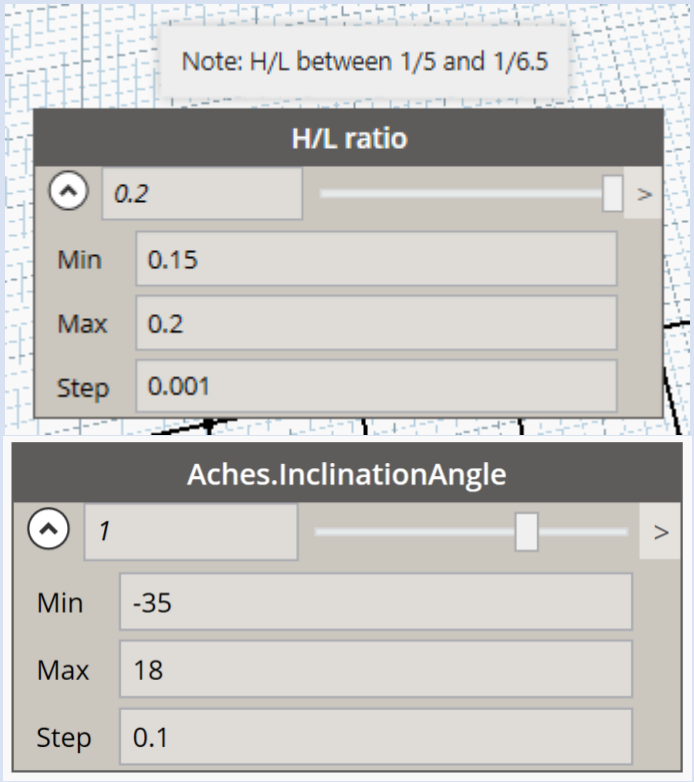
Step	Instruction	Result
3.13	For this exercise, we want to connect the points from the first line to the points of the second one, following their order such as connect [0, 0] to [1,0] to create a line. For this, manipulate the way the data is arrange in our lists. Use a Transpose node. *List → Organize → Transpose* You might be familiar with this operation from your math classes where you used it as operation for matrices. In this case, the transformation is from 2 lists of 15 elements to 15 lists of 2 elements. After transpose operation, when clicking on the first list, you can see that it contains the first point on the first line and the first point on the second line.	
3.14	To generate the lines connecting the points, use *PolyCurce.ByPoints* node. *Geometry → Curves → PolyCurve → ByPoints*	

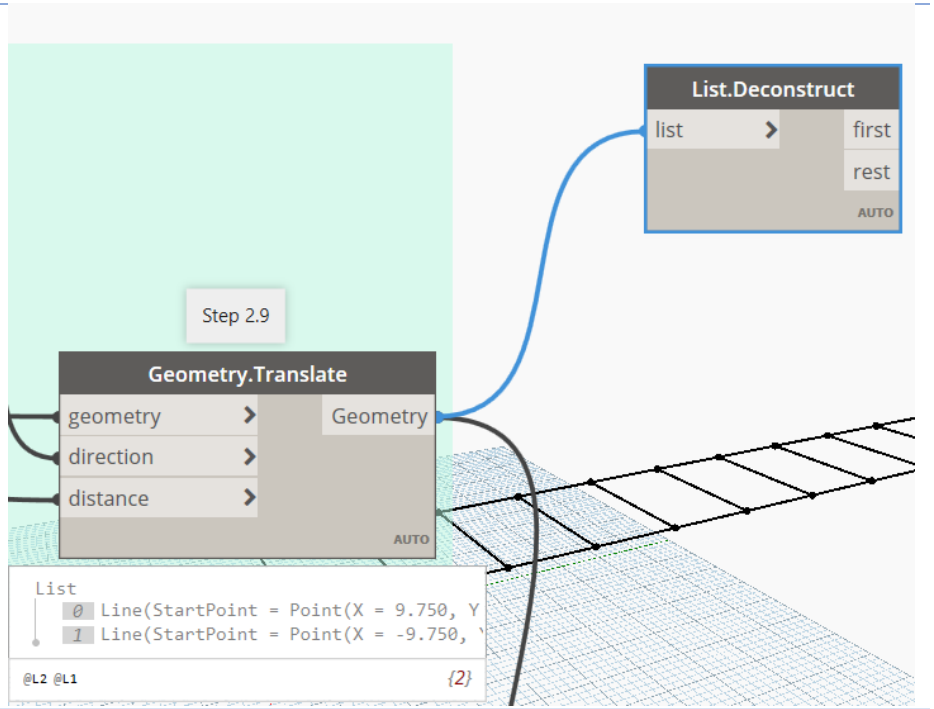
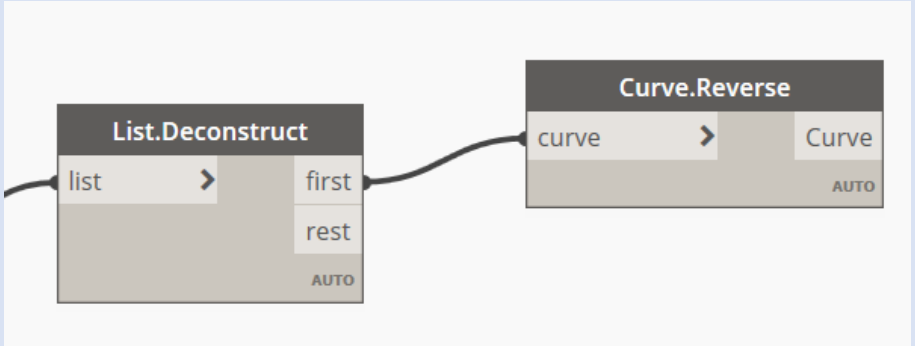
Step	Instruction	Result
3.15	Of course, to keep the definition organized, group the nodes added during this step 3. Hereunder, you can see all the nodes added to generate the lines of the transversal girders.	

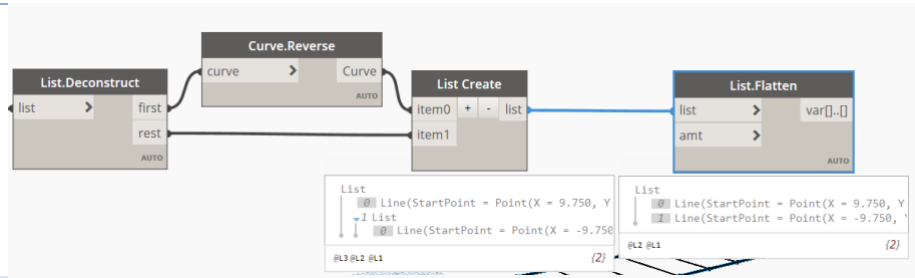
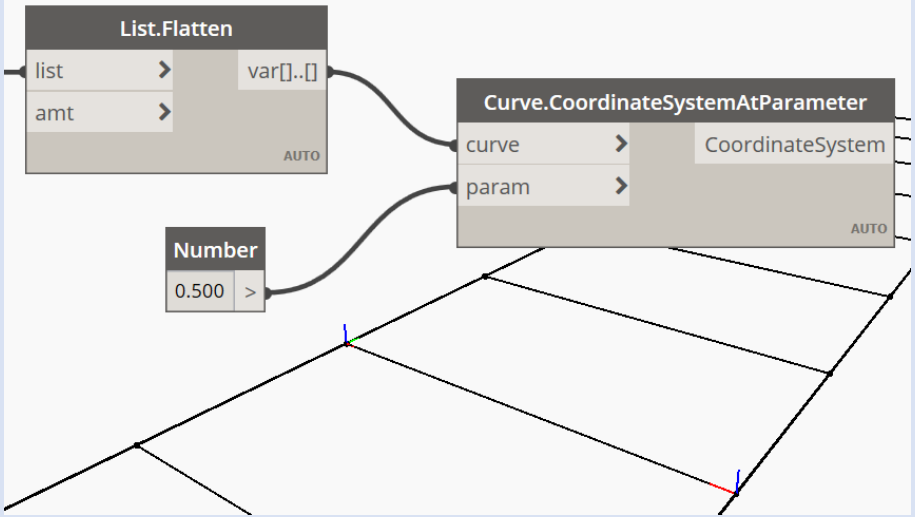


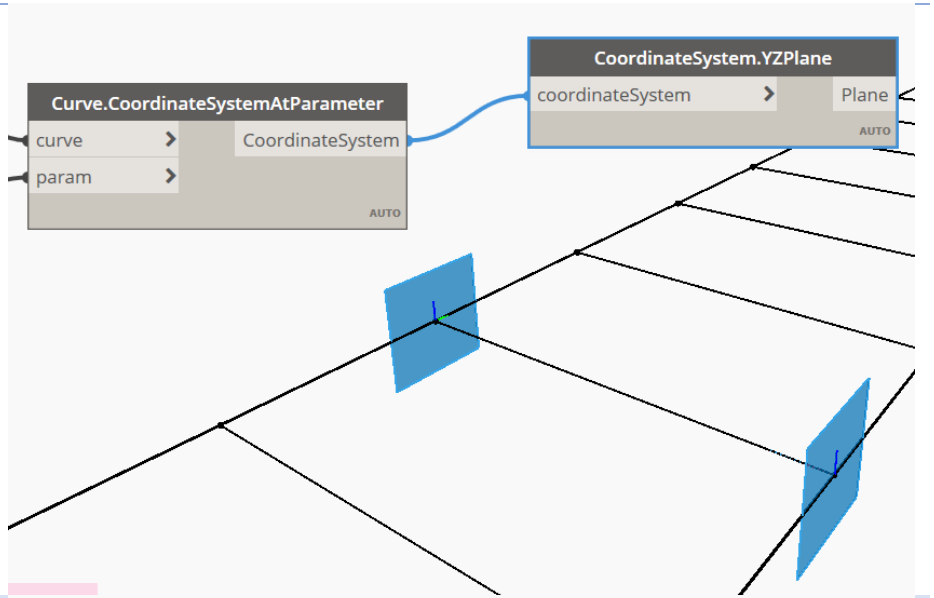
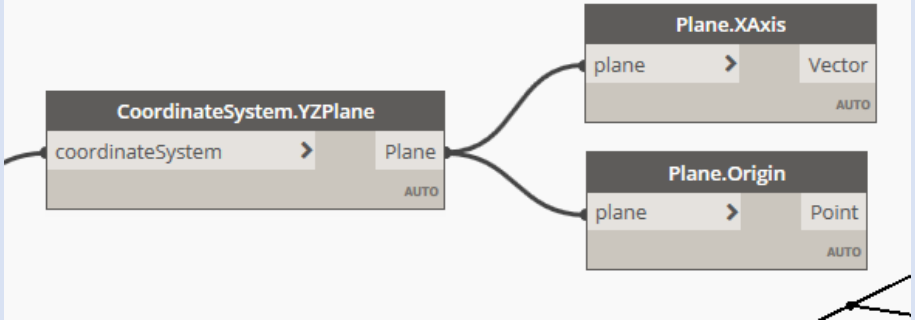
4. Create the arches

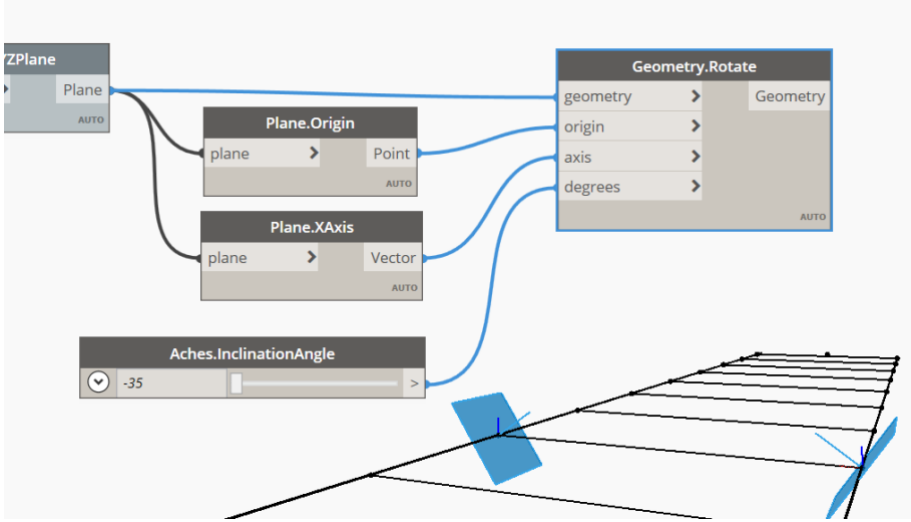
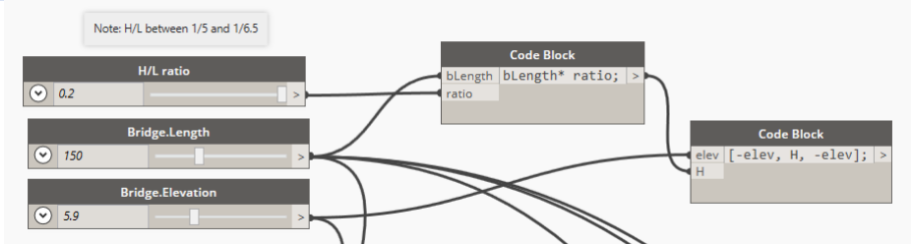
This step is also available as video tutorial as following: https://autode.sk/2X28Ikx

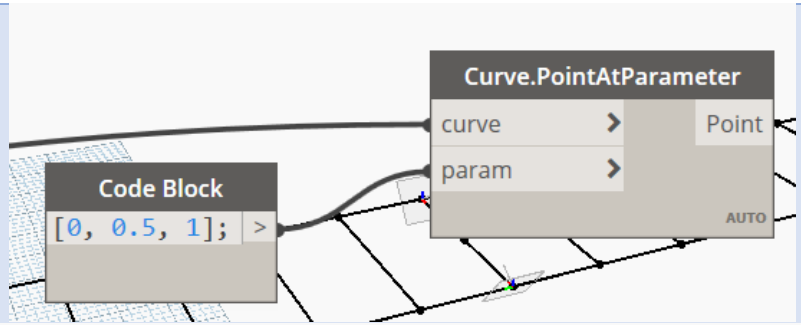
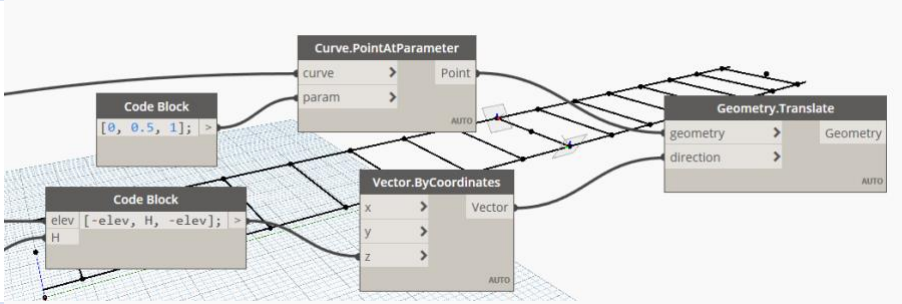
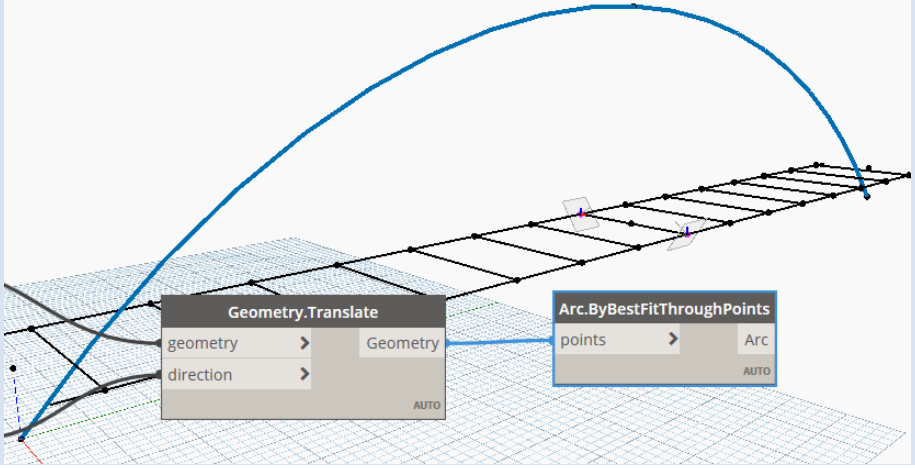
Step	Instruction	Result
4.1	To get started with the creation of the arches, add two Number Slider nodes for input. *Input → Basic → Number Slider.* First input slider is to provide input about arch height(H) to bridge length(L) ratio. H/L = [1/5, 1/6.5]. Using this, the setting we do for the first input slider are as following: *Name: H/L ratio* *Min: 0.15* *Max: 0.2* *Step: 0.001* **! To add a note, use the shortcut CTRL+W, double click on it and add the text you want.** The purpose of the second input slider is to control the inclination of the Arches. The settings for this node are as following: *Name: Aches.InclinationAngle* *Min: -35* *Max: 18* *Step 0.1.* The logic behind this slider is as following: - when this angle is 0 the arches will pe parallel: | | - when the angle is positive, the arches will for an A: / \ - when the angle is negative, the arches will form a V: \ /	

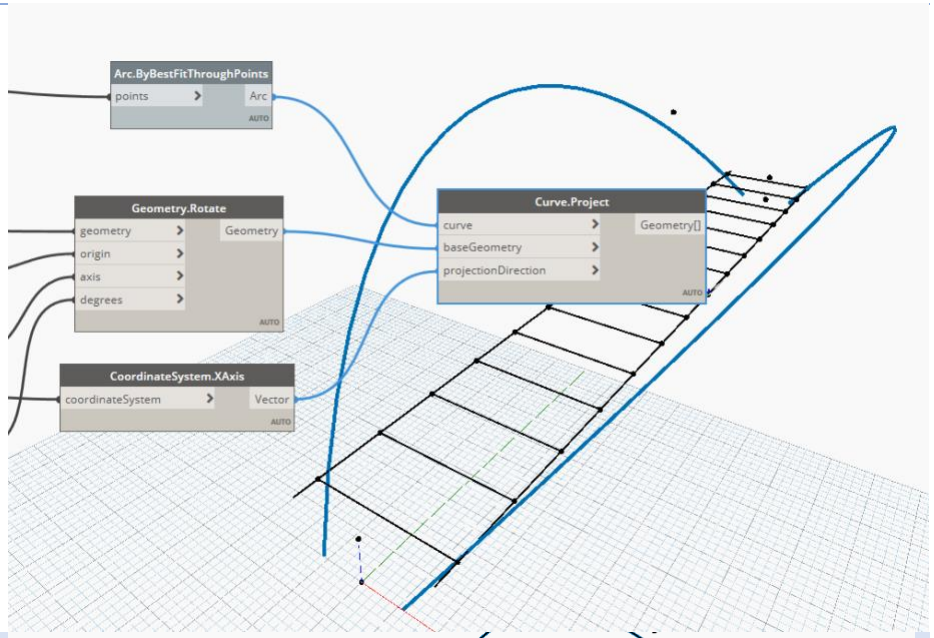
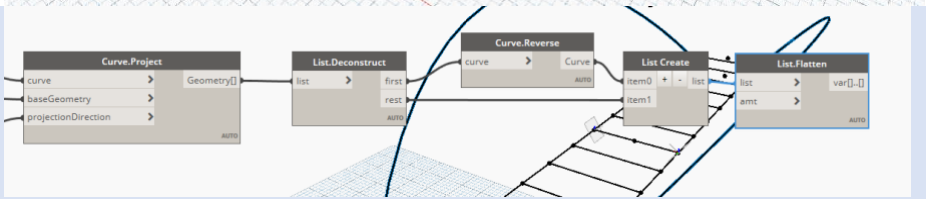
Step	Instruction	Result
4.2	Make use of the geometry created so far and start the creation of the arches using the lines created at step 2.9. A first node to add is *List.Deconstruct*. *List* → *Modify* → *Deconstruct* This allows you to access the two side lines independently.	
4.3	In this step, get one of the curves and reverse its direction using the node *Curve.Reverse*. *Geometry* → *Curves* → *Curve* → *Reverse* This node allows you to reverse the vector direction of the line. This means, that if the start point of the line was near by the origin, using this node the start of the line will be the previous end point. For our case, we reverse the direction of the first line.	

Step	Instruction	Result
4.4	Use a *List Create* node to bring back together the two lines in one list and a *List.Flatten* node to remove the unwanted level created during previous operations. *List* → *Generate* → *List Create* *List* → *modify* → *Flatten*	
4.5	Next, create a coordinates system on both lines using the node *Curve.CoordinateSystemAtParameter*. *Geometry* → *Curves* → *Curve* → *CoordinateSystemAtParameter* As input for parameter, use 0.5 so that the coordinate system will be always in the middle of the line. For this add a number input node. *Input* → *Basic* → *Number* In the figure, you can also see the two coordinates systems created.	

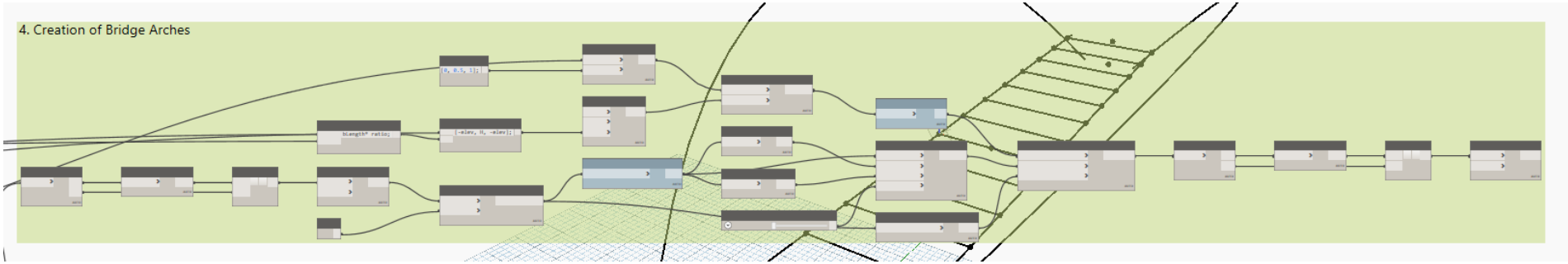
Step	Instruction	Result
4.6	Using these coordinates systems, extract their YZPlane using the node *CoordinateSystem.YZPlane*. *Geometry → Abstract → CoordinateSystem → YZPlane.*	
4.7	From the planes generated at step 4.6, extract their origin using *Plane.Origin* node as well as their X axis using the node *Plane.XAxis*. *Geometry → Abstract → Plane → Origin* *Geometry → Abstract → Plane → XAxis*	

Step	Instruction	Result
4.8	As indicated at step 4.1, we want to have the arches rotated according to specific angles. To do this, define a plane on which to project our arches. This plane is nothing else than the rotation of the plane extracted at step 4.6. To rotate it, use the node Geometry.Rotate. *Geometry → Modifiers → Geometry → Rotate* *(origin, axis, degree)* The geometry input port gets the planes from step 4.6. The origin and the axis ports get the origin and the axis extracted at step 4.7. the degree input port gets connected to the input slider defined at step 4.1 for Arches.InclinationAngle. In the background, you can see the planes rotated in opposite directions. This is because at step 4.2 you reversed the direction of one of the lines, so that the associated coordinates system will be have the XAxis opposite compare to the other one.	
4.9	Now that you defined the plane, you need to define the geometry, i.e. the arch to project on the two planes defined at step 4.8. The height of the arch is defined relative to bridge length using the ration defined at step 4.1 Add a *Code Block* node (double click on the definition canvas) on which add the formula: *blenght * ratio* this will create two input ports for our node and connect Bridge.Length slider to blenght input port and H/L ration node to the ration input.	
4.10	Next, create a vector using again a Code block node containing the following values. [-elev, H, -elev], where H is the arch height, defined at step 4.9 and elev is the bridge elevation we defined at step 1.5 Aiming to create a three points arch, this will allow us to define the Z coordinates of the three points.	

Step	Instruction	Result
4.11	Before using the Z coordinates defined at step 4.10, create three points on the central axis. The points are created at parameter on the central axis [0, 0.5, 1]. This means, [start, middle, end]. For this, use the node *Curve.PointAtParameter*. *Geometry → Curves → Curve → PointAtParameter* The curve input port gets as input the central axis of the bridge defined at step 1.4.	
4.12	Add two more nodes: one to perform the translation operation (*Geometry.Translate*) and another one to get the elevations transformed into a vector (*Vector.ByCoordinates*). *Geometry → Modifies → Translate (direction)* *Geometry → Abstract → vector → ByCoordinates(x,y,z)*	
4.13	Now that you have the points of the arch, use the *Arc.BybestFitThroughPoints* to generate the arc. *Geometry → Curves → Arc → ByBestFitThroughPoints*	
4.14	Next step is to project this arc on the planes we created at step 4.8.	

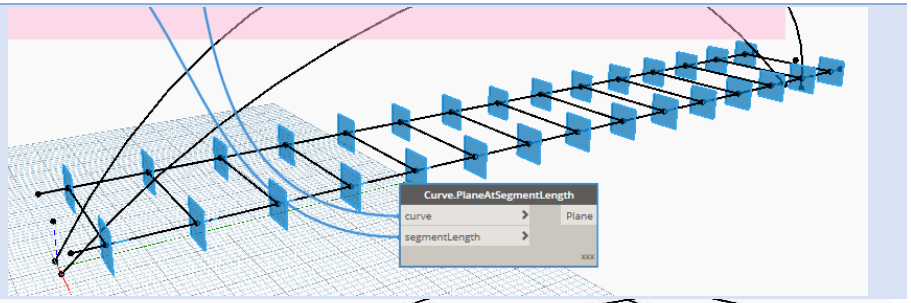
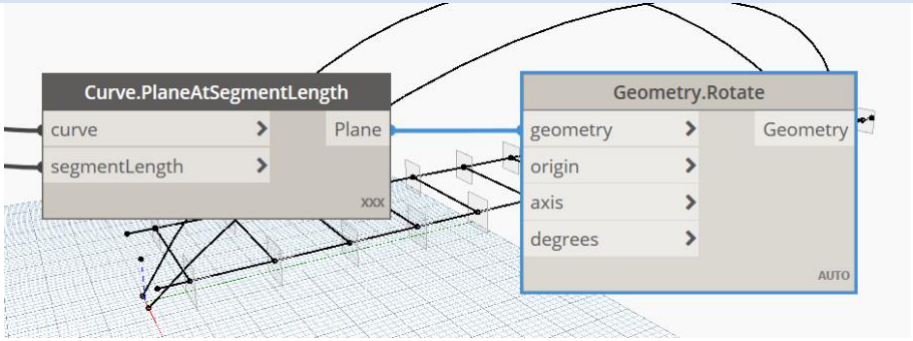
Step	Instruction	Result
	For this, use the node *Curve.Project*. *Geometry* → *Curves* → *Curve* → *Project* This node also requires a direction for the projects and for this use the CoordinateSystem.XAxis to extract the x axis of the coordinates systems defined at step 4.5. *Geometry* → *Abstract* → *CoordinateSystem* → *XAxis*	
4.15	Now that you finished with the definition of the arches, perform some cleanup operations – mainly to reverse the effect of the reverse direction we applied at the beginning of this step. Use the same operations: *List.Deconstruct*, *Curve.Reverse, List.Create, List.Flatten.*	

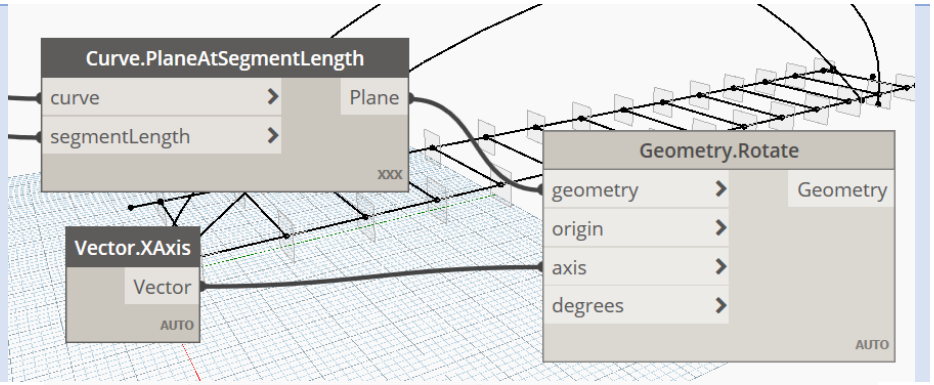
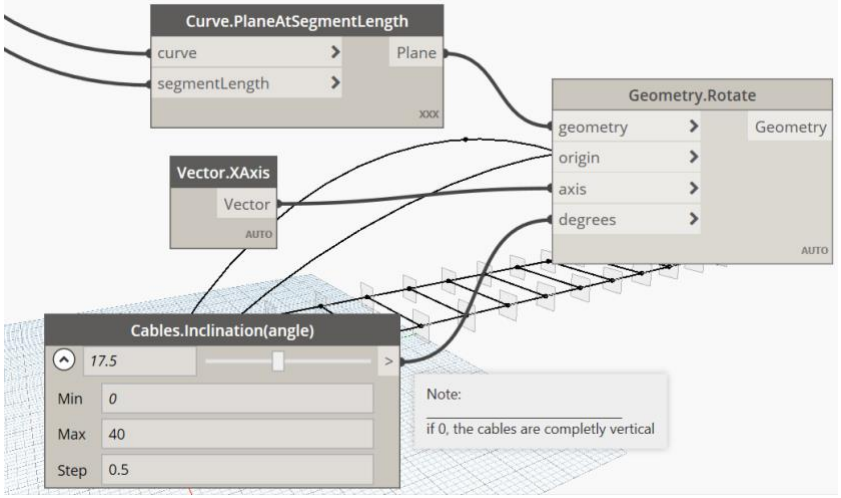
This conclude the creation of the bridge arches. Again, to keep the definition organized, create a group for the nodes we added at step 4.

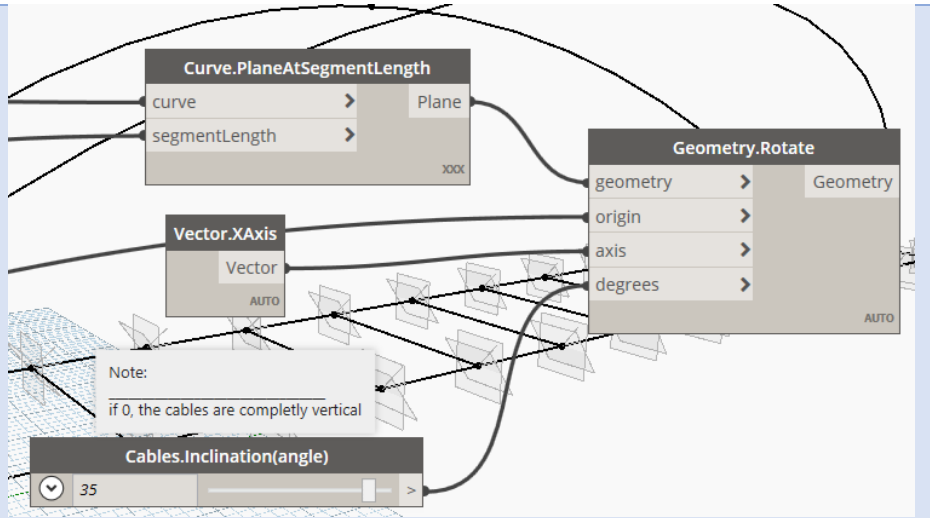
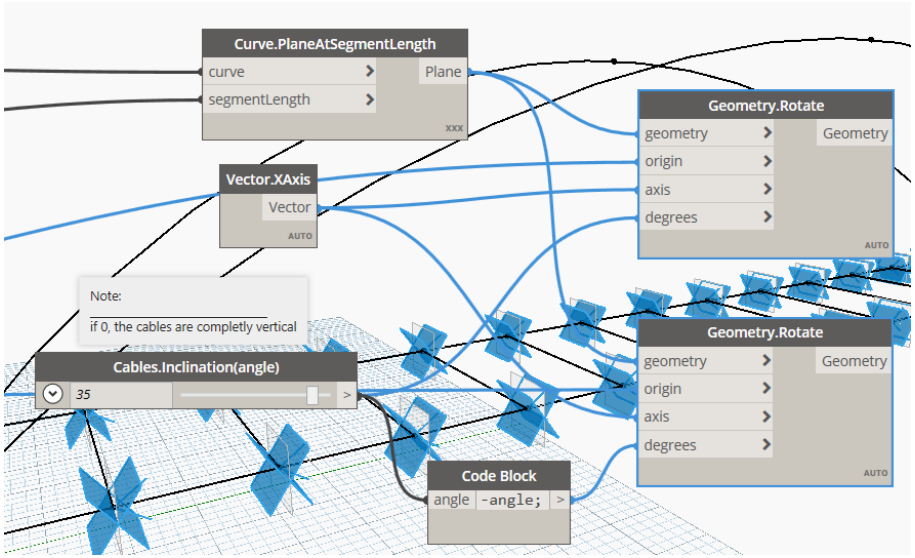

4. Creation of Bridge Arches

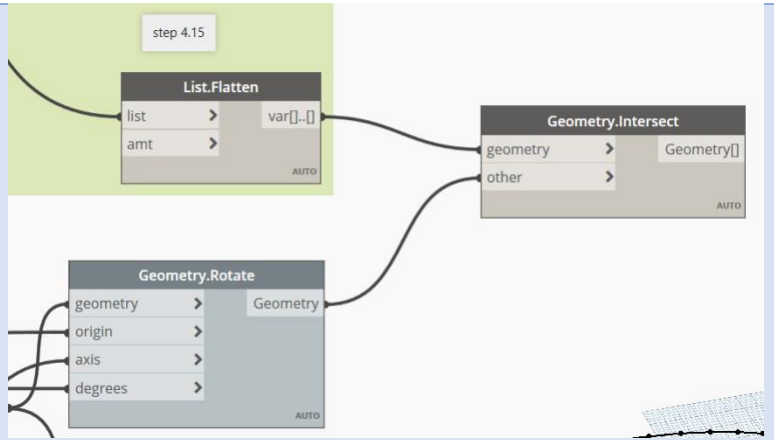
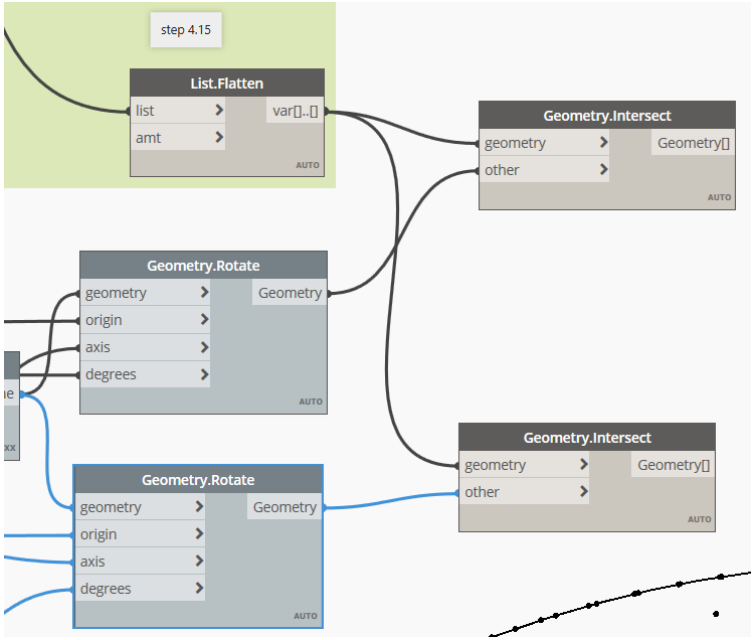
5. Cables geometry

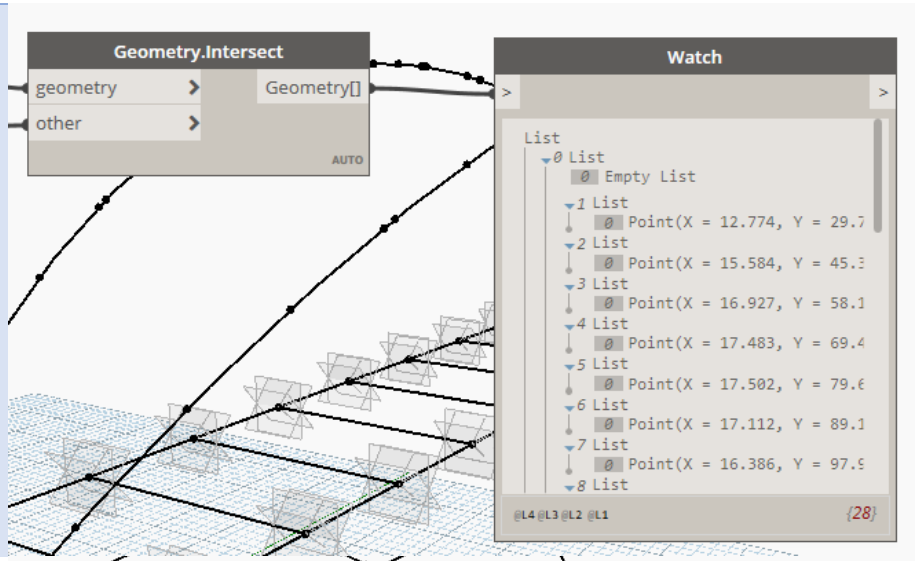
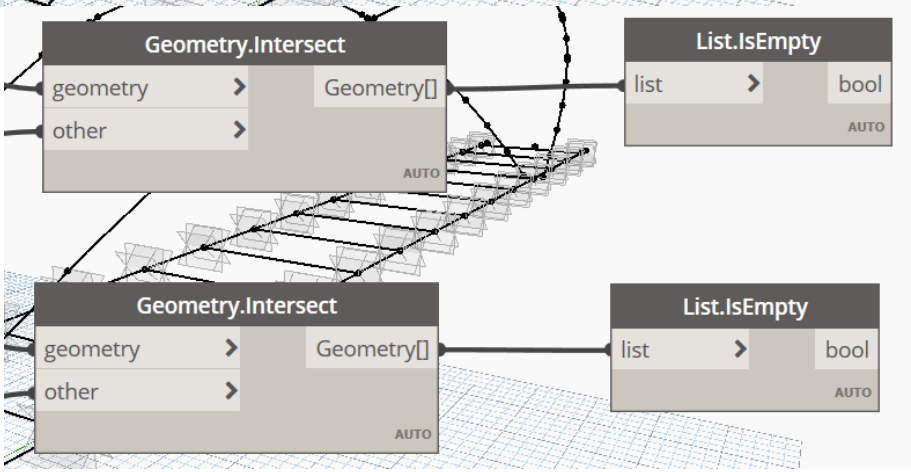
This step is also available as video tutorial as following: https://autode.sk/2Cur18z

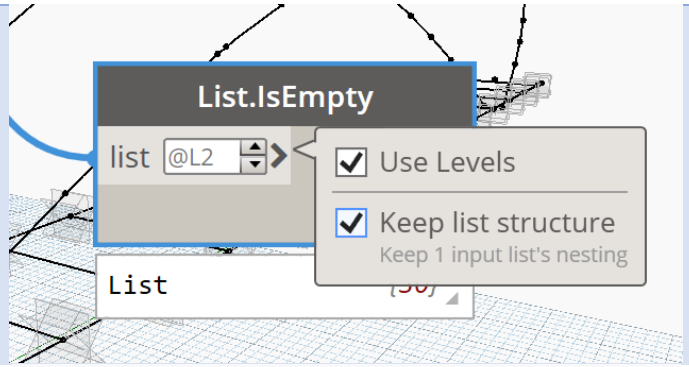
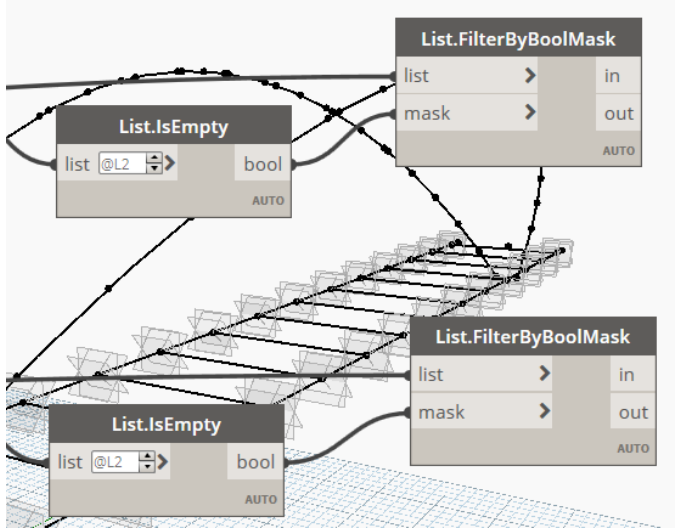
Step	Instruction	Result
5.1	To get started, add a node called *Curve.PlaneAtSegmentLength*. *Geometry* → *Curves* → *Curve* → *PlaneAtSegmentLength* As input for the *curve* port use the two lines created at step 2.9. As input for the *segmentLength* port use the sequence created at step 3.6. Set the *Lacing* of the node to *Cross Product*. In the background a set of plans was added on the same points as the points used to define the lines of the transversal girders.	
5.2	Because we want to place the cables at different angles and generate various configurations, use the rotate operation on the plans created at step 5.1 *Geometry* → *Modifiers* → *Geometry* → *Rotate* *(origin, axis degree)* For this, provide as input for the geometry node the plans created at step 5.1.	

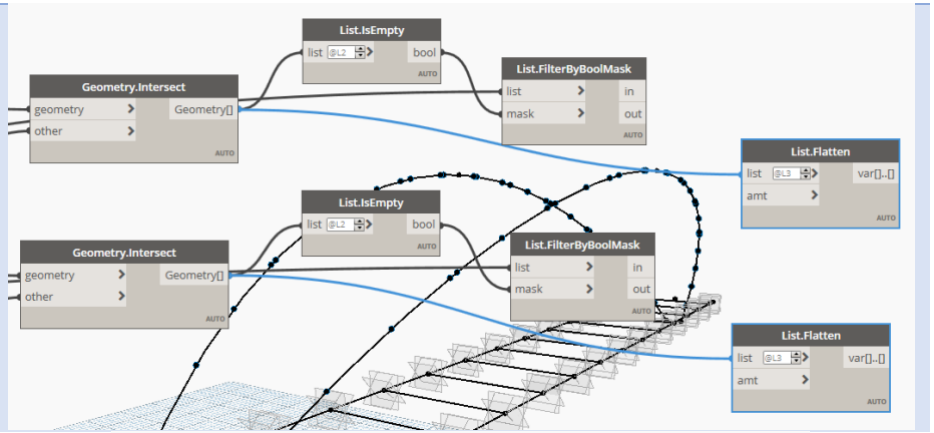
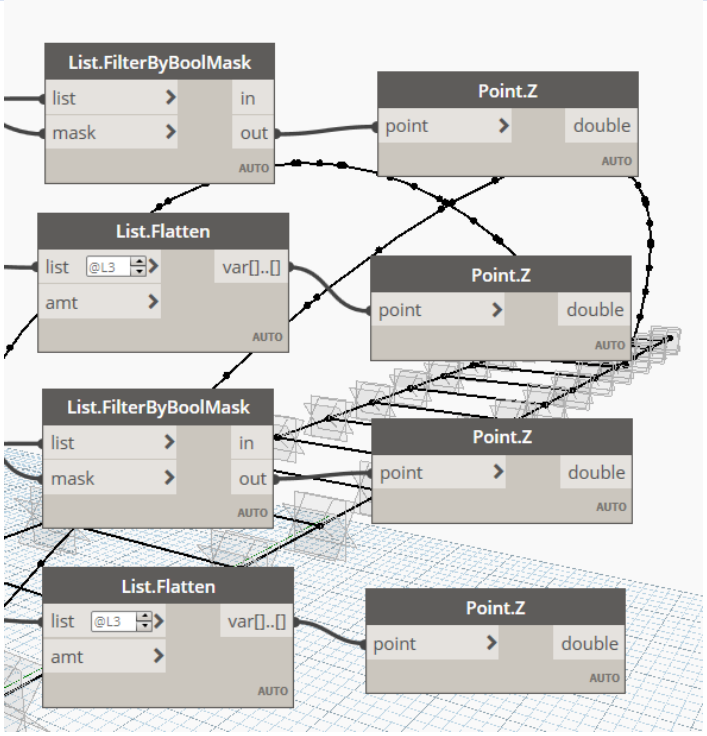
Step	Instruction	Result
5.3	Use X as the rotation axis. For this add a *Vector.XAxis* node. *Geometry → Abstract → Vector → XAxis*	
5.4	For the degrees input port use a *Number Slider* to define the variation of the angle. *Input → Basic → Number Slider* For this, define the following parameters: *Name: Cables.Inclination(angle)* *Min: 0* *Max: 40* *Step: 0.5* When the degree is 0, then the cables are completely vertical as indicated in the note we added.	

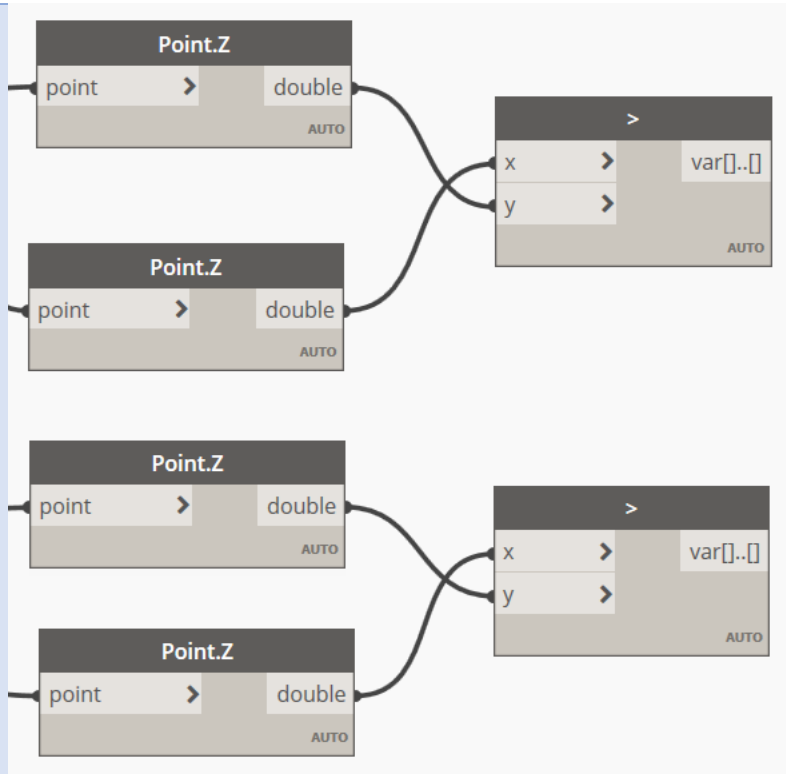
Step	Instruction	Result
5.5	To complete the rotation, provide the origin of the rotation. The origin is used as the rotation point. For this use the points defined at step 3.7. Once connected the points to the port origin, you can already see on the background a new set of planes create, which are rotated according to the specifications.	
5.6	The planes defined until step 5.5 allow you to create the cables only in one direction. Use the same logic to create the planes rotated in the opposite direction. A fast way to do this, Copy and paste the *Geometry.Rotate* node defined at step 5.2 using the keyboard shortcuts (CTRL+C – copy and CTRL+V – paste). This will keep the connections as defined previously. Additionally, add a *Code Block* node to negate the angle and re-wire the input we provide to the degrees port.	

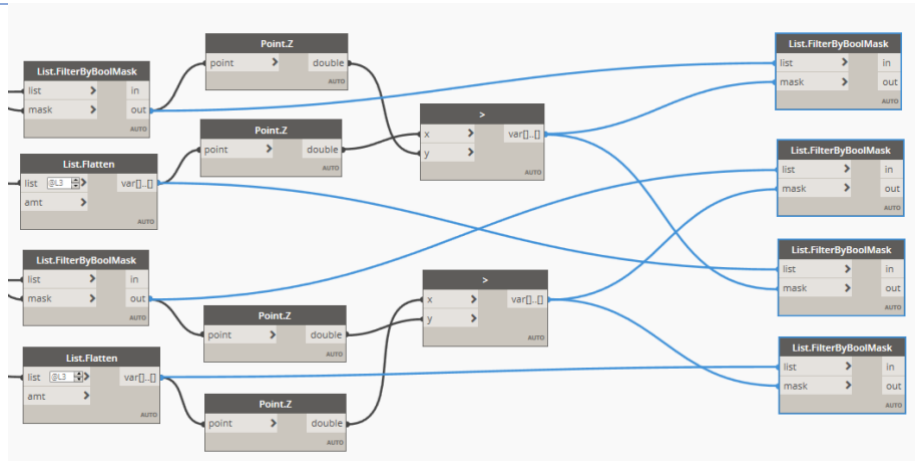
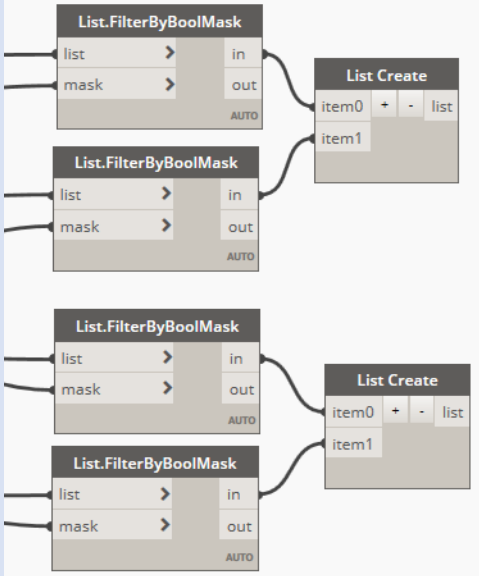
Step	Instruction	Result
5.7	Use the planes defined to create the hanging points of the cables on the arches. For this use the *Geometry.Intersect* node. *Geometry* → *Modifiers* → *Geometry* → *Intersect* The input for the geometry port will be the arches defined at step 4.15 after applying the List.Flatten operation. The input for the other port will be the planes defined at step 5.5 using the rotation operation.	
5.8	Repeat the logic from step 5.7 to create the intersection points of the second set of planes with the arches.	

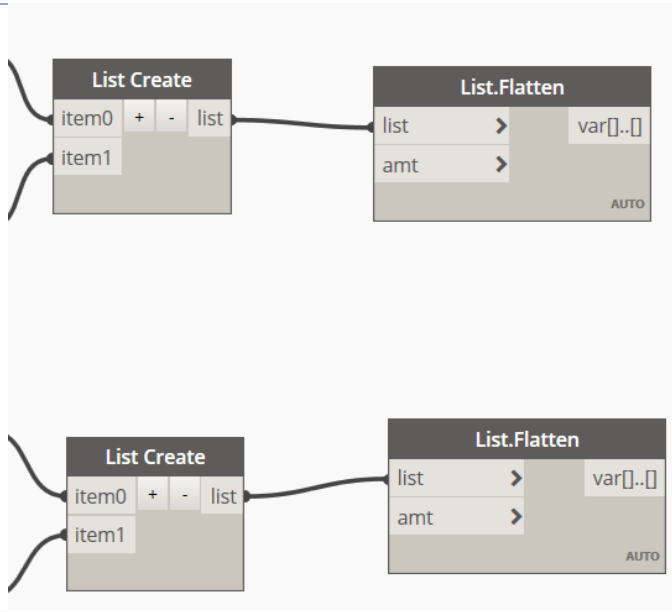
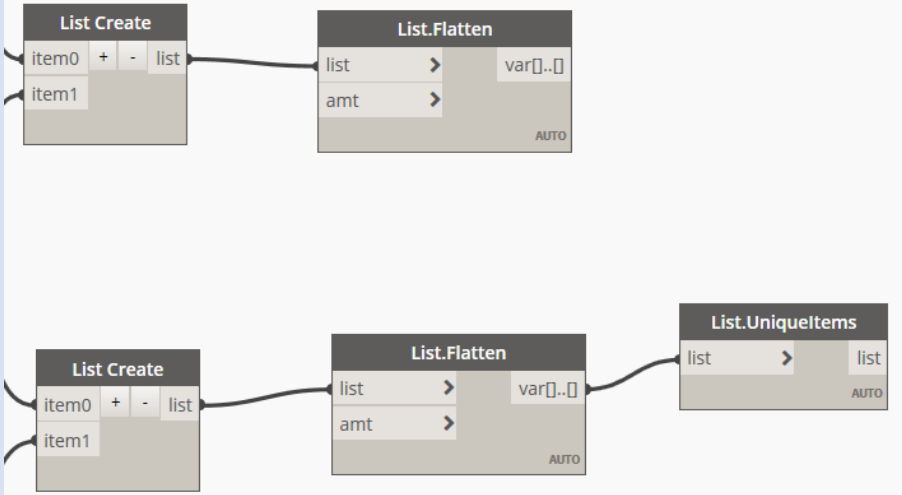
Step	Instruction	Result
5.9	Before connecting the set of points with a line, perform some operations related to lists manipulation. Notice that when you set the angle to 40 degrees, one of the lists is empty, which means that no intersection point is create by one for the planes. Add a Watch node to see this. *Display → Watch → Watch*	
5.10	From this point, perform the lists manipulation operations on both lists of points in parallel. Add two *List.isEmpty* nodes. *List → Inspect → isEmpty*	

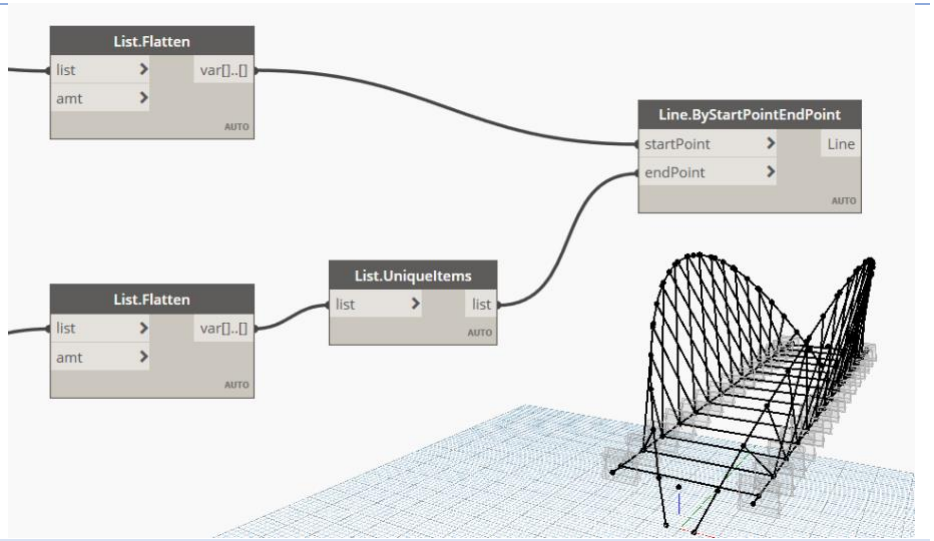
Step	Instruction	Result
5.12	Next, the checks need to be done, not on the first level of the list, but on the second level. To do this, click on the arrow of the list input port and select Use Levels. Do this for both *List.isEmpty* nodes. Also, thick the *Keep list structure*.	
5.13	Because the main reason of doing all these operations was to remove the null lists, use a *List.FilterByBoolMask* to identify items to keep and to drop. Keep the ones having the Value false in the *List.IsEmpty* and drop the ones having the value True. *List* → *Modify* → *FilterByBoolMask* The mask is defined by the output of the node *List.IsEmpty*, while the list to filter is list of points at step 3.7 (*Curve.PointAtSegmentLength*)	

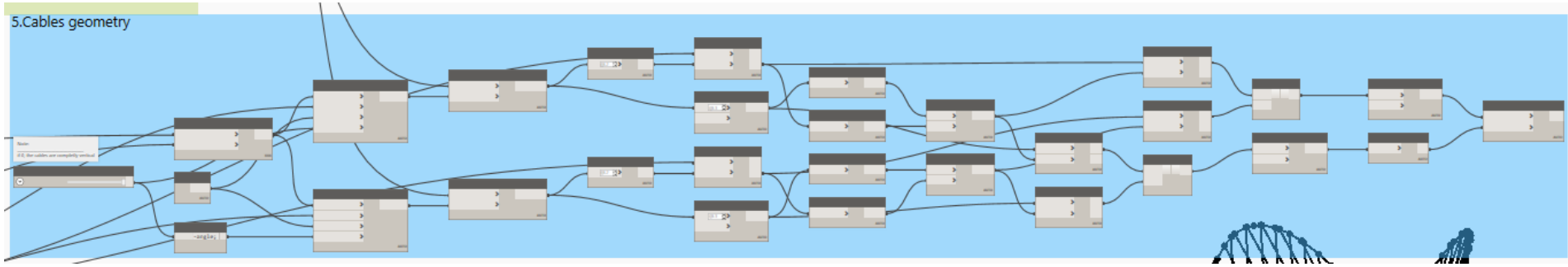
Step	Instruction	Result
5.14	Because the bridge deck is elevated in reference to the arches base, we will have points which, after intersection, will be located on the arches bellow the deck. We remove those points, as we only want to have tension forces in our cables. A first step is to change the structure of the lists we got after intersecting the planes with the arches (step 5.8). *List* → *Modify* → *Flatten* Make sure you do the flattening of the list at level L3.	
5.15	Next, extract the Z coordinates of each point using the node *Point.Z*.	

Step	Instruction	Result
5.16	Remember, the points you got after *List.FilterByBoolMask* are the ones located on the arches, while the ones from *List.Flatten* are the ones at the bridge deck level. Eliminate those points on arches located under the bridge deck. For this, use the math operator ”>” *Math* → *Operators* → >	

Step	Instruction	Result
5.17	Use again the node *List.FilterByBoolMask* to keep those points which are above the bridge deck. Notice in the figure the way the nodes are connected. Also to organize the nodes, such that the two nodes at the bottom containing the points on the arches, while the two top ones contain the points on the bridge deck line.	
5.18	Use a *List Create* node to combine the lists of points two by two. *List → Generate → List Create*	

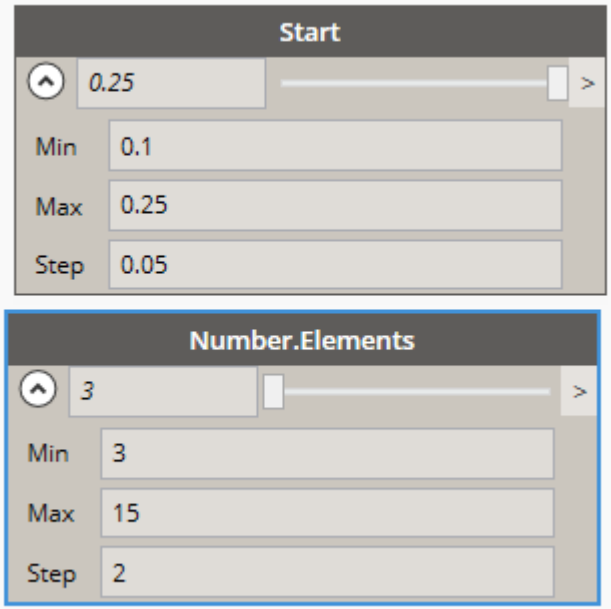
Step	Instruction	Result
5.19	In the next step, flatten both lists created. *List → Modify → Flatten*	
5.20	A final operation before creating the lines of the cables is to use a *List.UniqueItem*. This is mainly required to avoid duplication when the angle of the cables is 0 (the cables are vertical). *List → Inspect → UniqueItems*	

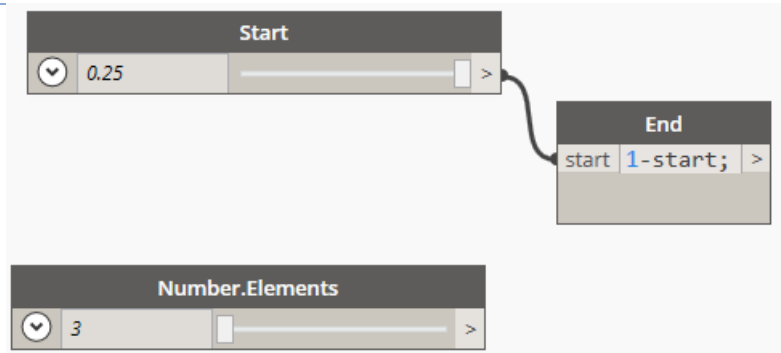
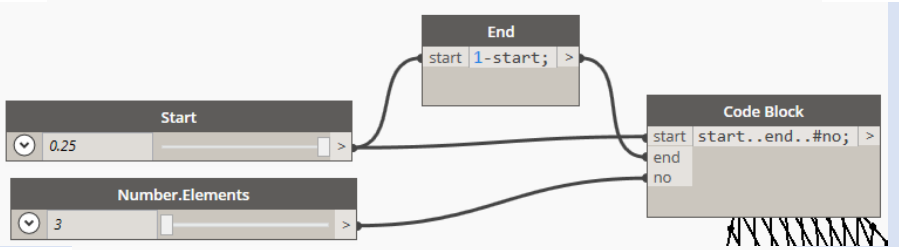
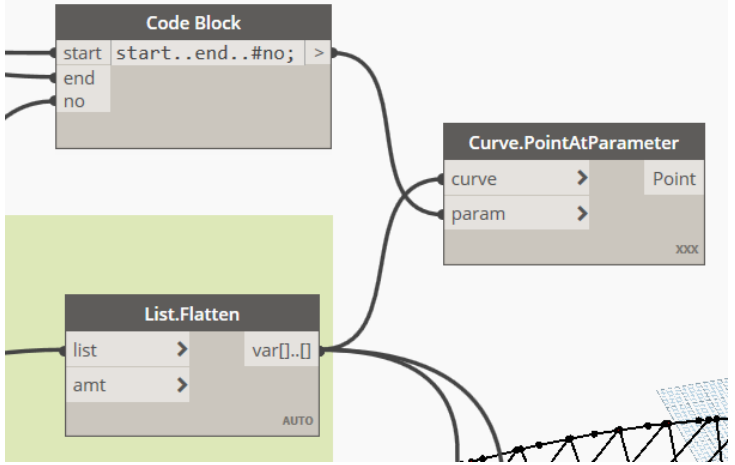
Step	Instruction	Result
5.21	Create the lines of the cables using the node Line.ByStartPointEndPoint. *Geometry → Curves → Line → ByStartPointEndPoint*	
5.22	Finally, create a group around the nodes added for the creation the cables' geometry.	

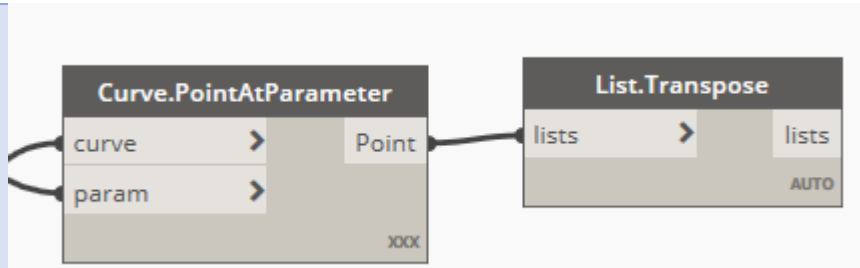
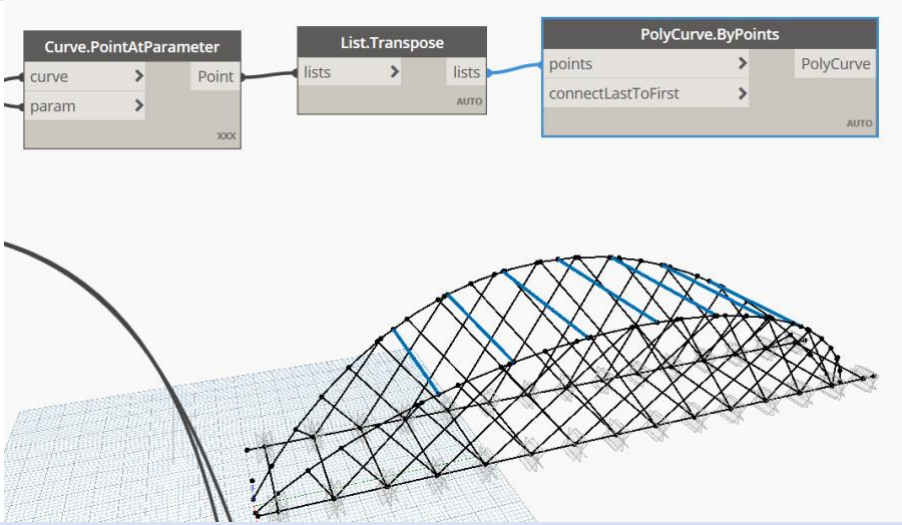


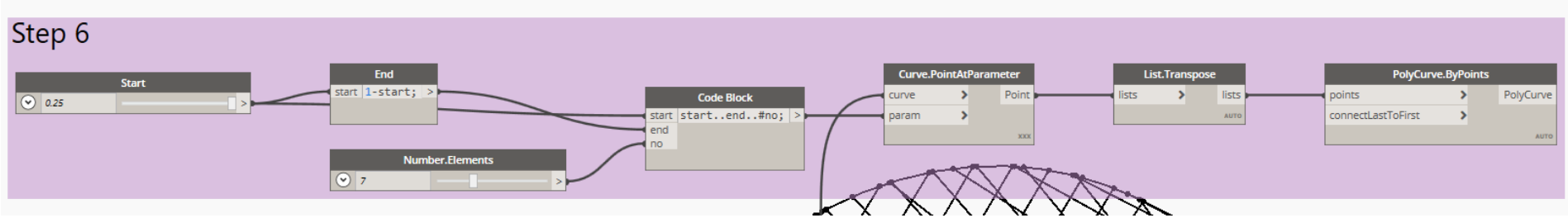
6. Create transversal elements between arches

This step is also available as video tutorial as following: https://autode.sk/2p9dznx

Step	Instruction	Result
6.1	Add two Number Sliders, which will allow you to control the position of the elements on the arches. First slider is used to define the start of the distribution, as a parameter of the arch. Remember, when defining a parameter on a curve, 0 means starting point and 1 means the end point. For the first slider, we define the following setup: • *Name: Start* • *Min: 0.1* • *Max: 0.25* • *Step: 0.05* For the second slider, we define the following setup: • *Name: Number.Elements* • *Min: 3* • *Max: 15* • *Step: 2*	

Step	Instruction	Result
6.2	To keep a symmetry between the start and the end of the elements' distribution, use the start to calculate the end using a *Code Block* and the formula *1-start*	
6.3	Use a *Code Block* to define a sequence of n=Number.Elements distributed from Start to End, using the following notation: *start..end..#no;*	
6.4	Use a *Curve.PointAtParameter* node to create the points based on the sequence created at step 6.3. *Geometry → Curves → Curve → PointAtParameter* The curves on which to distribute the points are the two arches of the bridge (created at step 4.15 after flattening). Make sure that the *Lacing* of the node is set to *Cross Product*.	

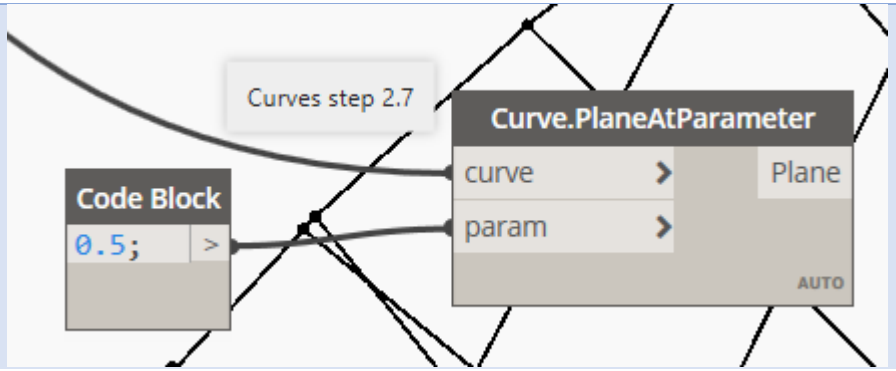
Step	Instruction	Result
6.5	Next, use *List.Transpose* node to group the points two by two, which means the first group will be composed by the first point on the first arch and the first point on the second arch. *List → Organize → Transpose*	
6.6	Create the lines of the elements which will be between the two arches using *PolyCurve.ByPoints* node. *Geometry → Curves → PolyCurve → ByPoints*	
6.7	Finally, create a group around the nodes added for the creation of these elements.	


Step 6
Start
0.25
End
start
1-start;
Number.Elements
7
Code Block
start
end
no
start..end..#no;
Curve.PointAtParameter
curve
param
Point
List.Transpose
lists
lists
AUTO
PolyCurve.ByPoints
points
connectLastToFirst
PolyCurve
AUTO

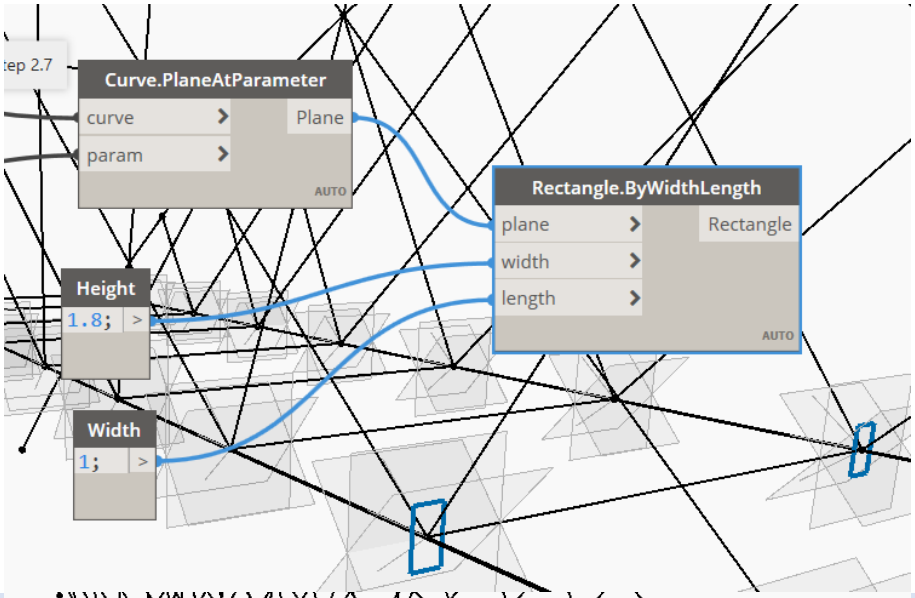
7. Create the 3D geometry of the bridge elements

This step is also available as video tutorials as following: https://autode.sk/33CiI6D & https://autode.sk/2rE2ewt

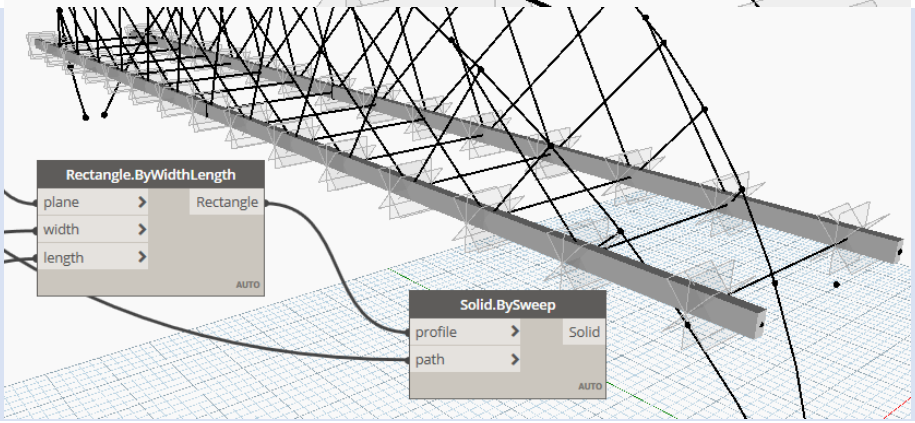
So far, the elements of our bridge are represented as curves. In this final step, we aim at adding the geometry of their cross section. Of course, at this point, their geometry will be purely indicative, as the exact sizes require to run a detailed structural analysis.

Step	Instruction	Result
7.1	Start with the geometry of the longitudinal girders, for which we create a rectangular section of 1m width and 1.8 meters height. First, create a plane, perpendicular to the line of them. This can be easily created using the node *Curve.PlaneAtParameter* and placed in the middle of the line. As input to the curve slot, provide the lines from step 2.9.	

7.2 Use the plane to create on it the cross section of the longitudinal girders. For this use the *Rectangle.byWidthLength* node.
Geometry → Curves → Rectangle → ByWidthLength *(plane, width, length)*
As you can see, this node allows you to define the plane on which to create the rectangle.
This will create the rectangle, centered on the origin point of the reference plane → see the figure, and the blue rectangles.



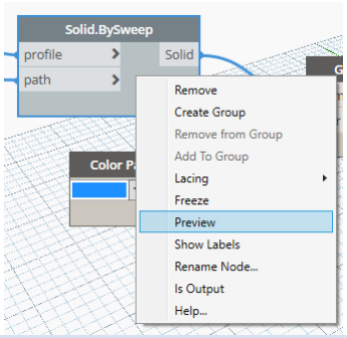
7.3 Once you have the Cross Section, you can use the node *Solid.bySweep* to create the 3D geometry of the girders.
Geometry → Solids → Solid → BySweep.
For this node, the profile input slot receives the rectangles defined for the cross section and the path will be the lines we defined at step 2.9.

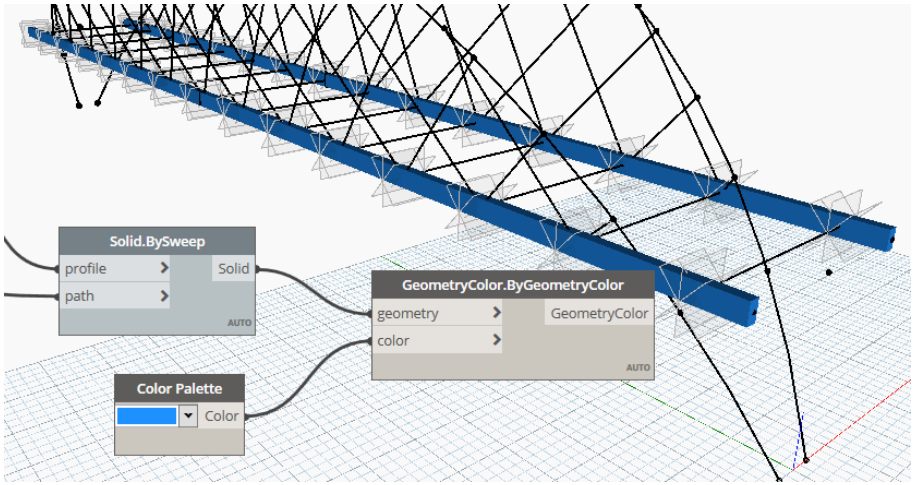


7.4 For a better visibility (because by default all the geometry is gray) we will apply a color to the solids we created at step 7.3 using the node *GeometryColor.BygeometryColor*.
Geometry → Modifiers → Geometry Color → ByColorGeometry

For the color input, pick a color using the node *Color Palette*.
Display → Color → Color Palette.

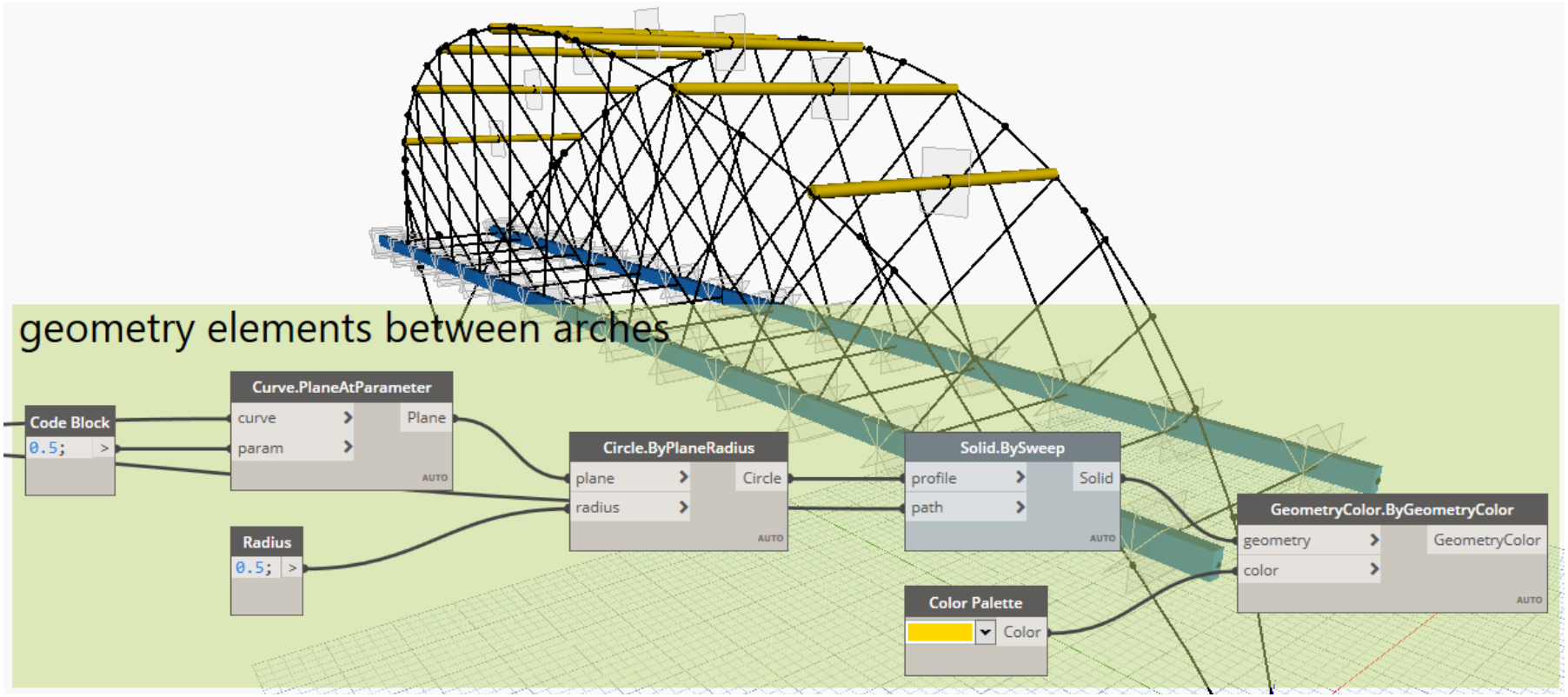
Make sure you turn off the preview of the node *Solid.bySweep* by right clicking on the node and uncheck Preview.



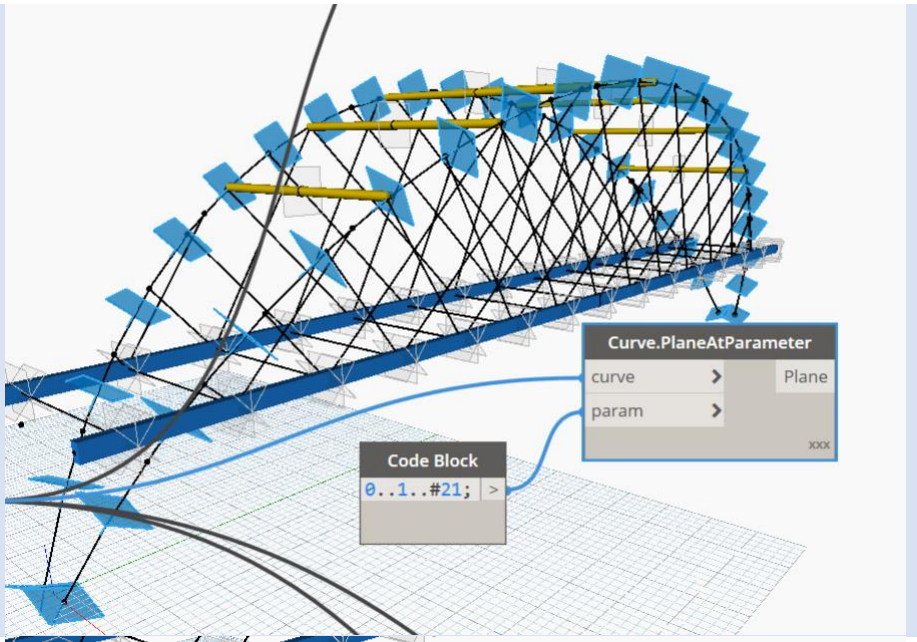


7.5 The same logic is applied to create the geometry of the elements between the arches, except that we are going to use a circular section instead of a rectangular one.

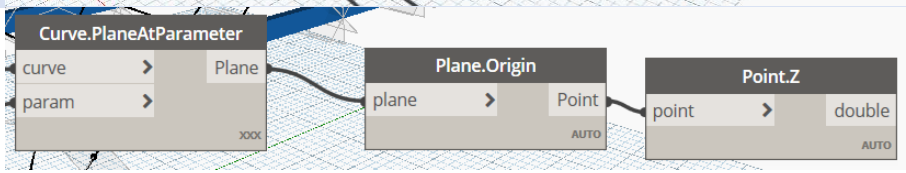
Geometry → Curves → Circle → ByPlaneRadius


geometry elements between arches
Code Block
0.5;
Curve.PlaneAtParameter
curve
param
Plane
AUTO
Radius
0.5;
Circle.ByPlaneRadius
plane
radius
Circle
AUTO
Solid.BySweep
profile
path
Solid
AUTO
Color Palette
Color
GeometryColor.ByGeometryColor
geometry
color
GeometryColor
AUTO

7.6 The geometry of the arches we want to be a little bit more complex in the way that we want to have a smaller cross section on top and a bigger gross section at the start and end of the arches. The cross section variation is as following:
Start: 2m; middle 1m; end: 2m;
First, we want to create several cross sections to make sure that the disctibution follows the shape of the arch.



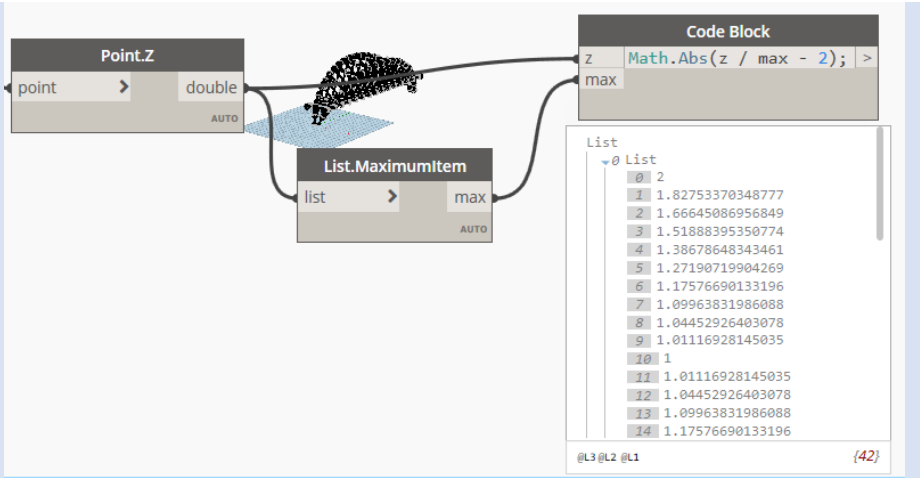
7.7 Because we want the cross section of the arches to vary according to the elevation, we get the plane origin using *Plane.Origin* node and get the Z coordintaes of the origin points using the node *Point.Z*.



7.8 Next, get the maximum value for the elevation using the node *List.MaximumItem* and use the formula:

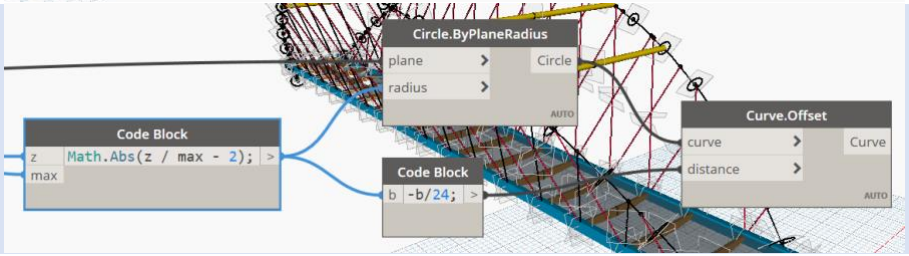
Math.Abs(z / max - 2)

to calculate the radius of the cross section.

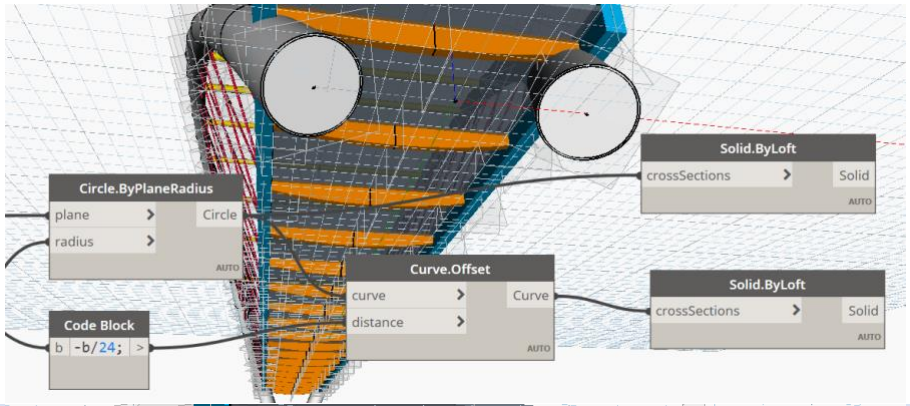


7.9 Now generate the circles using the node *Circle.ByPlaneRadius*.

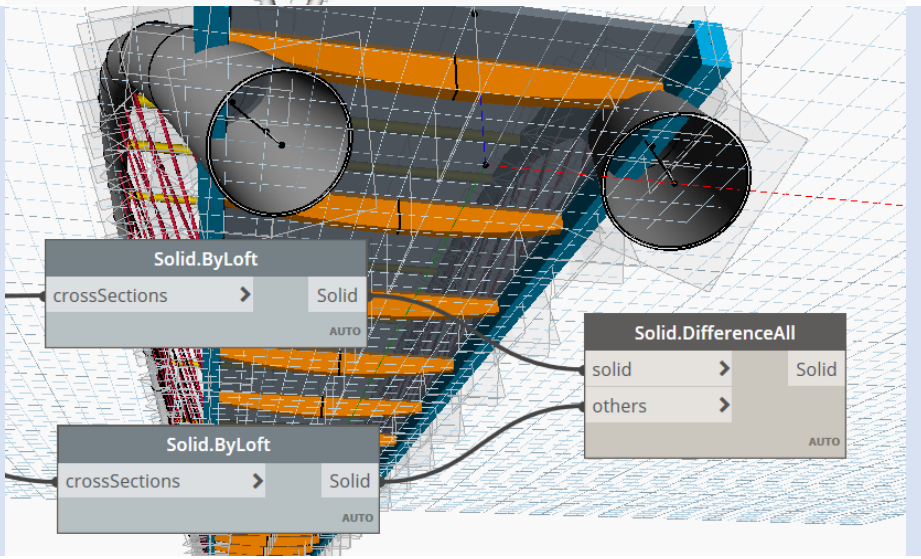
7.10 Because we want to create a circular hollow section for the cross section of the arches, we will create another circle which will give us the thickess of the arch pipe of radius/24. For this, add a *Code Block* and write the formula -b/24. Then add a a *Curve.Offset* node to offset the circle create at step 7.9



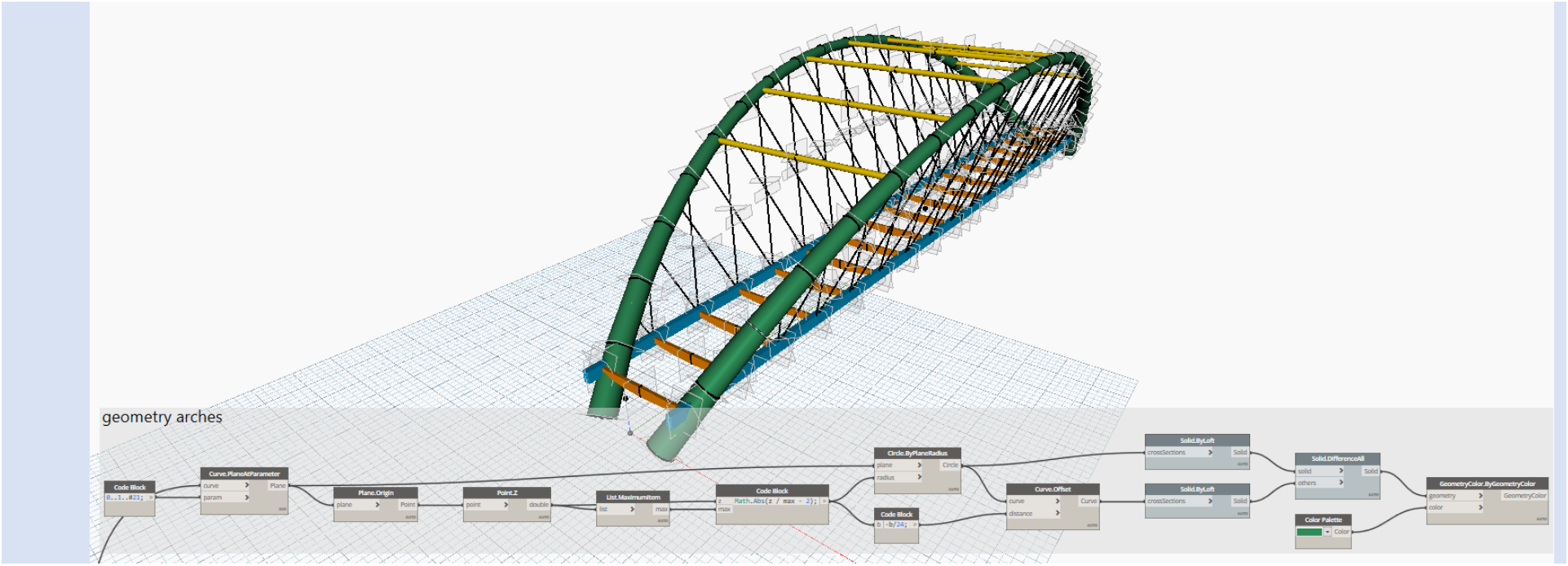
7.11 Next, create two solids using the node *Solid.byLoft*.



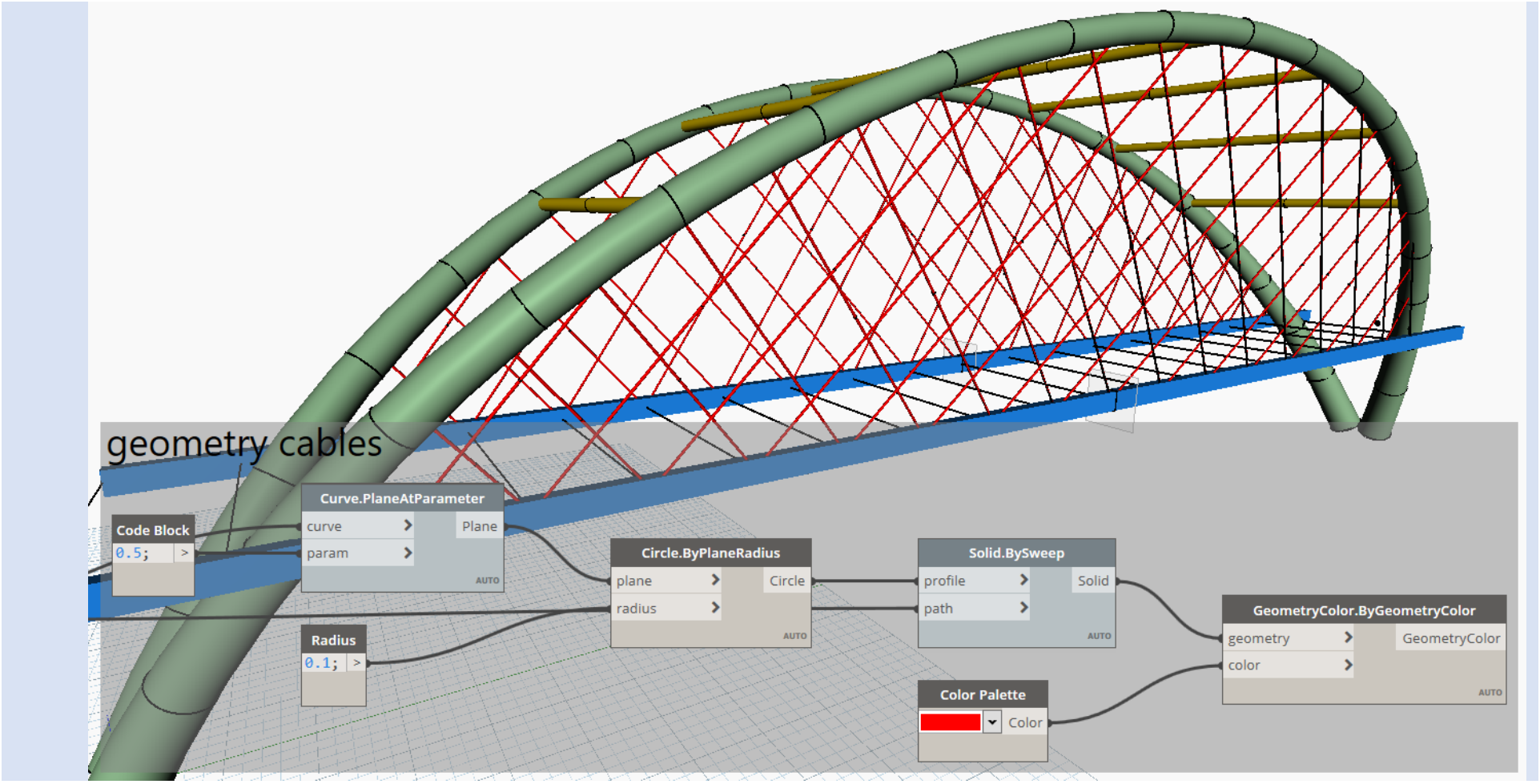
7.12 Use a boolean operation with solids to subtract the smaller solid from the bigger one. To do this, use the node *Solid.DifferenceAll*.



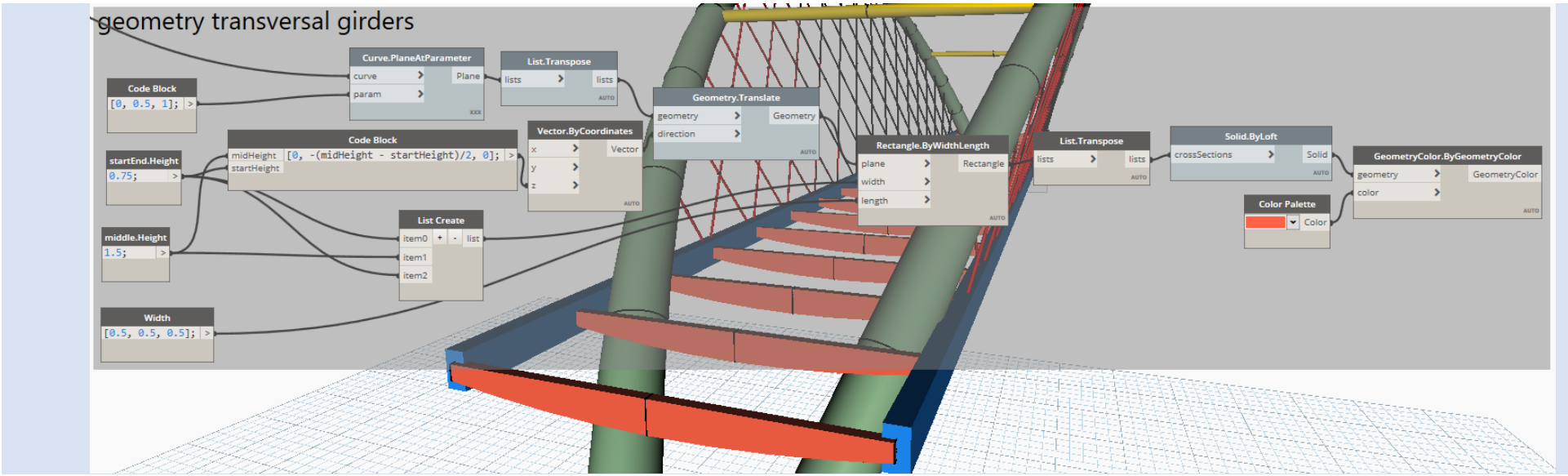
7.13 As a final step, use the node GeometryColor.ByGeometryColor to change the color of the geometry.



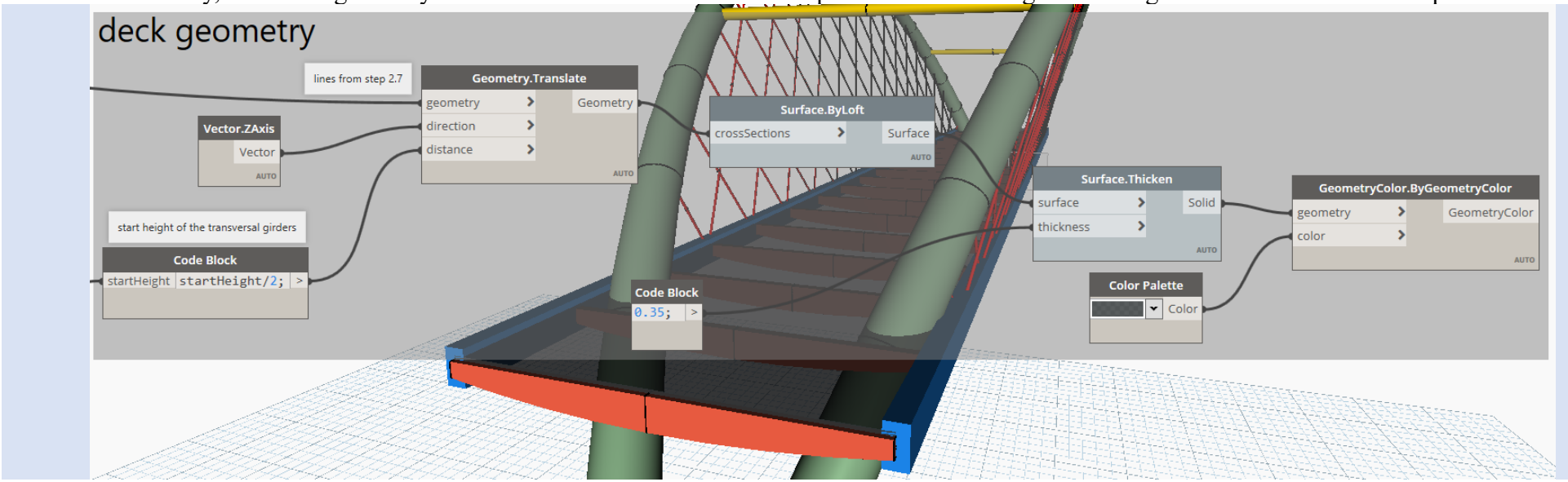
7.14 The geometry of the cables is created in the same way we created the geometry of the main girders and the elements between arches.



7.15 Applying the logic you learn previously from creating the geometry of the arches, you create the geometry of the transversal girders.

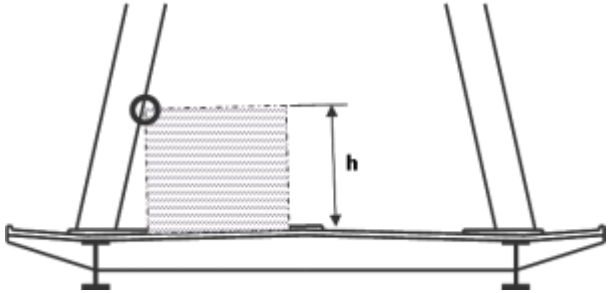


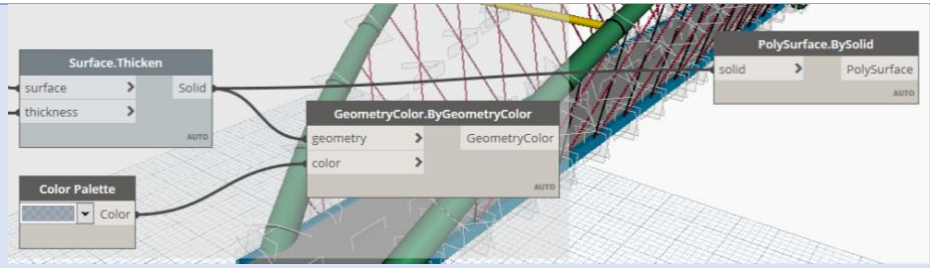
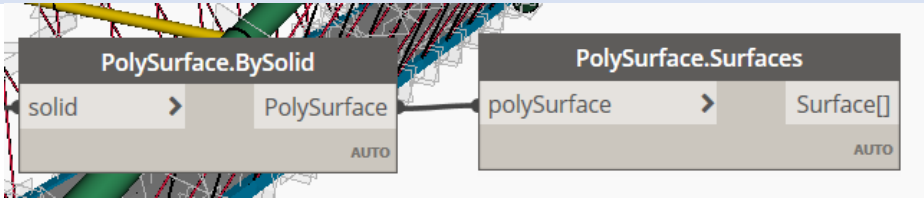
7.16 And Finally, create the geometry of the deck – the concrete slab on top of the transversal girders using the lines we created at step 2.9.



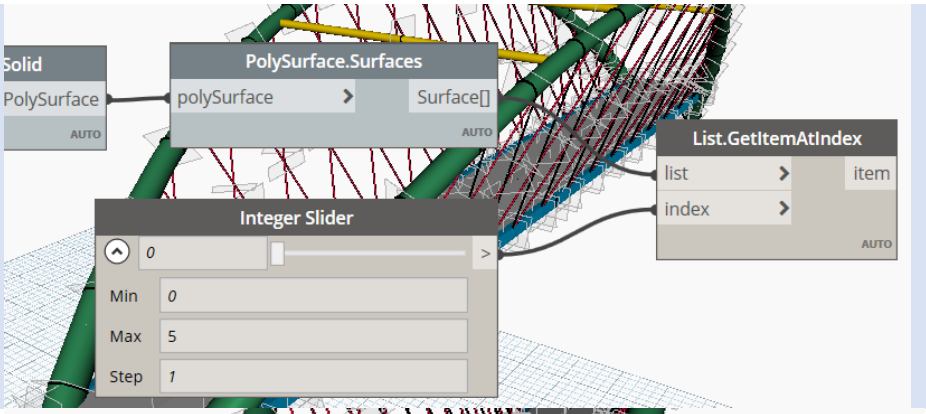
8. Calculate the lost width based on headroom conditions

One of the performance criteria we set in the motivation part of this tutorial is the headroom clearance. Based on this, we will calculate the lost width which is the difference between the total width and the width left based headroom clearance conditions.

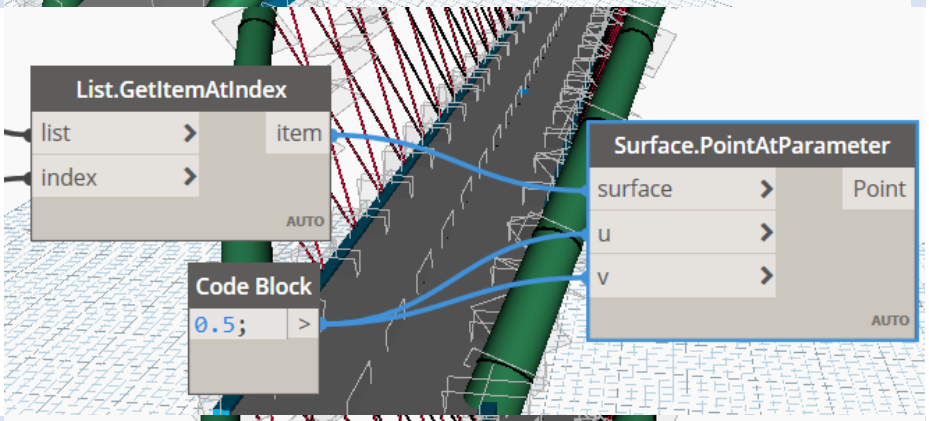


Step	Instruction	Result
8.1	To base the calculation of the lost width on the geometry already created, add a *PolySurface.BySolid* node to extract the surfaces of the bridge deck solid. As input, provide the solid obtained using the *Surface.Thicken* node. (see step 7, creation of the geometry of the bridge deck).	
8.2	Step 8.1 returns all the poly surfaces of the bridge deck solid. Add a *PolySurface.Surfaces* node to convert these on simple surfaces.	

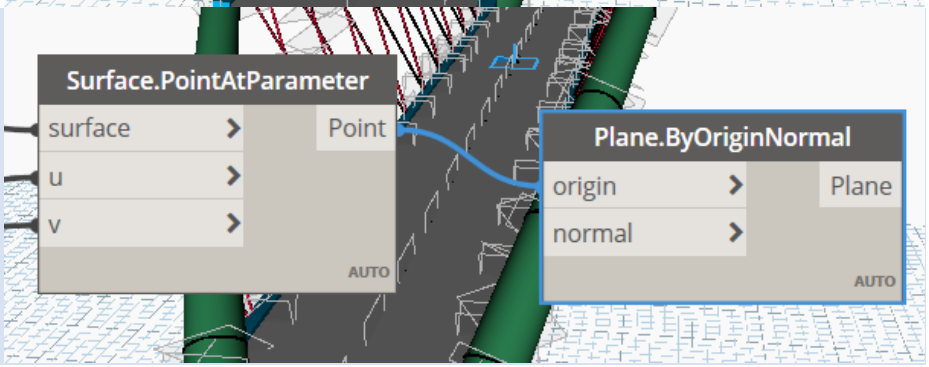
8.3 Add a *List.GetItemAtIndex* node and an *Integer Slider* node. These nodes are used to select only the top surface. on the settings of the slider, set the maximum value to 5. The slider will allow you to vary the index number of the surfaces from the list, allowing you to search for the top surface of the deck. In this case, the index is 0.



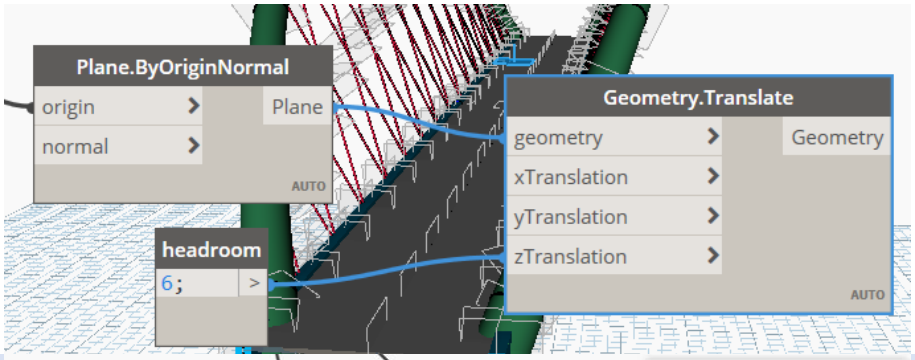
8.4 Create a point on the surface extracted at step 8.3 using the node *Surface.PointAtParameter*. For the u and v parameters, add a *Code Block* node and type 0.5, which will add a point in the middle of the deck surface.



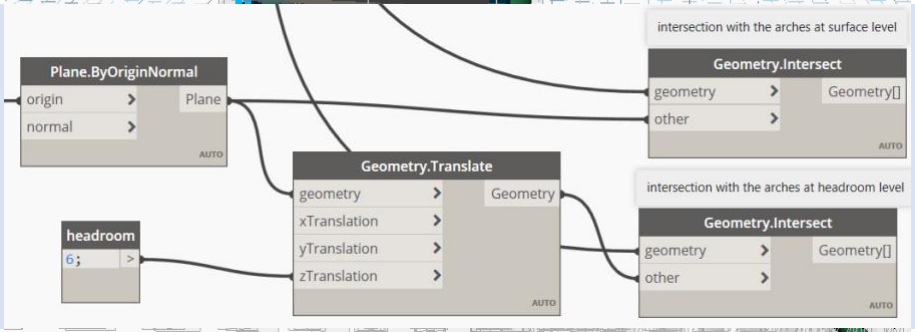
8.5 Add a *Plane.ByOriginNormal* node and use the point created at step 8.4 as origin. The normal by default is represented by Z axis,



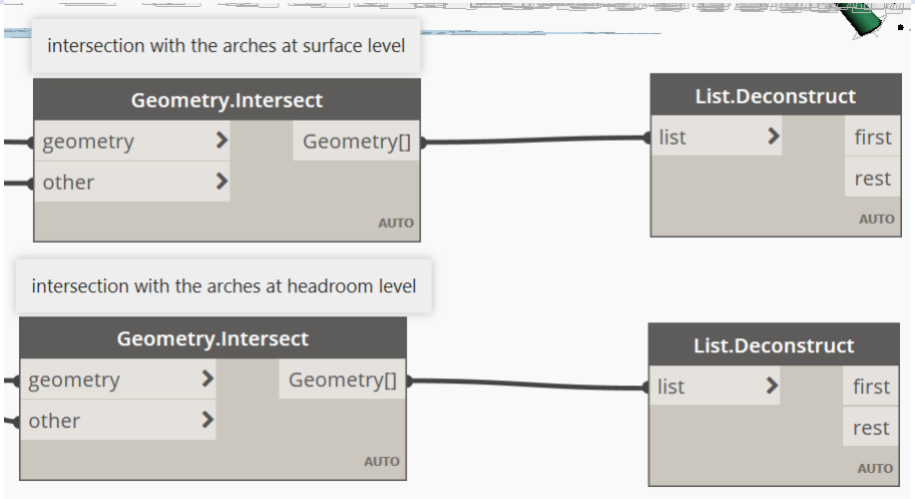
8.6 Add a *Geometry.Translate* node to translate the plane created at step 8.5 using the headroom height of 6 meters.



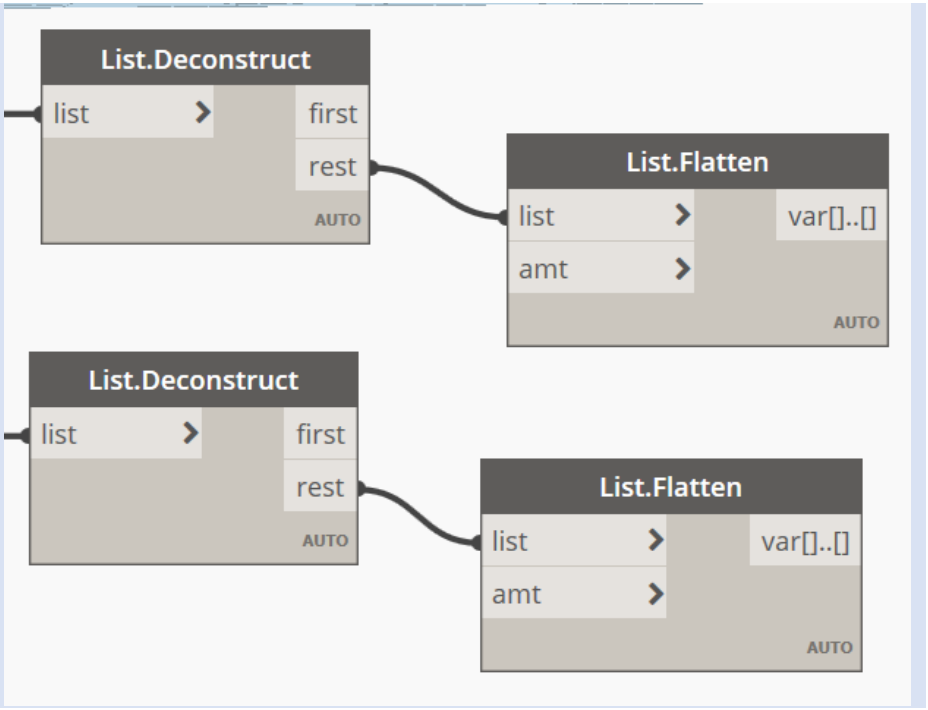
8.7 Add two *Geometry.Intersect* nodes. Use one to intersect the arches at the level of the deck surface and the other one at the level of the headroom. As input for the geometry port use the geometry obtained at step 7.12 using the node *Solid.DifferenceAll.* The other port gets as input the planes created at steps 8.5 and 8.6



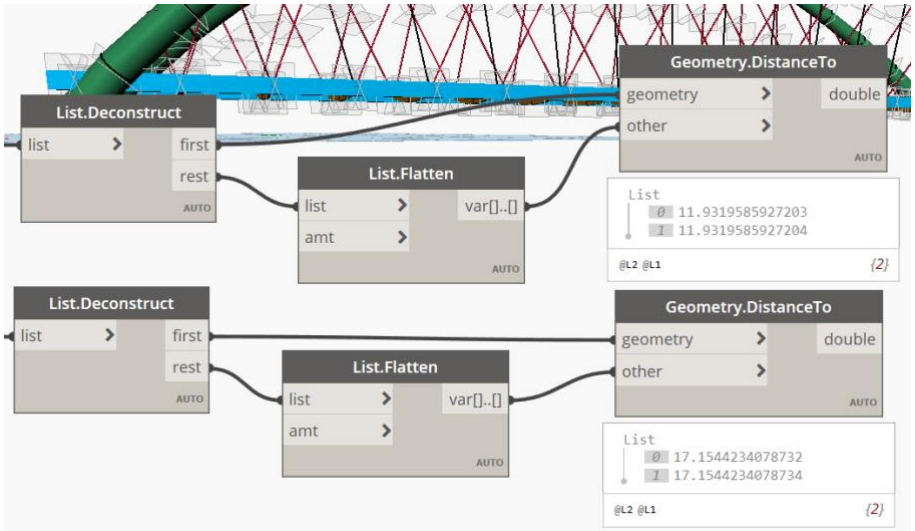
8.8 Add two *List.Deconstruct* nodes and connect to them the nodes from the step 8.7.



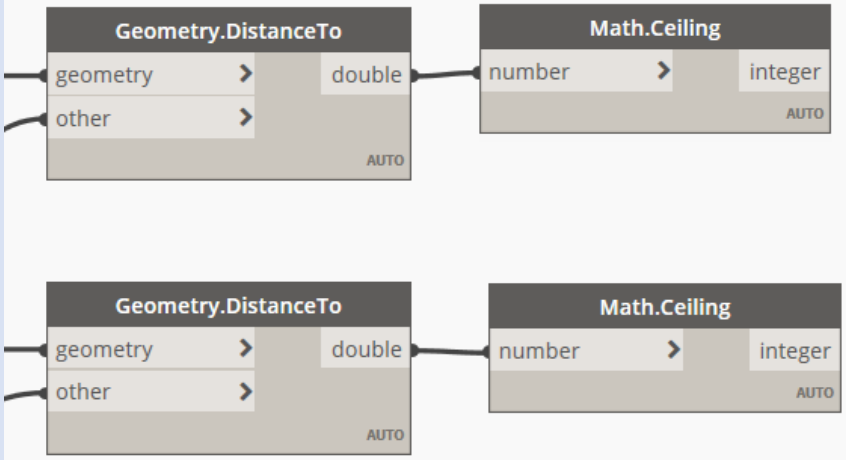
8.9 Add two *List.Flatten* nodes and provide as input the rest ports of the nodes from step 8.8.



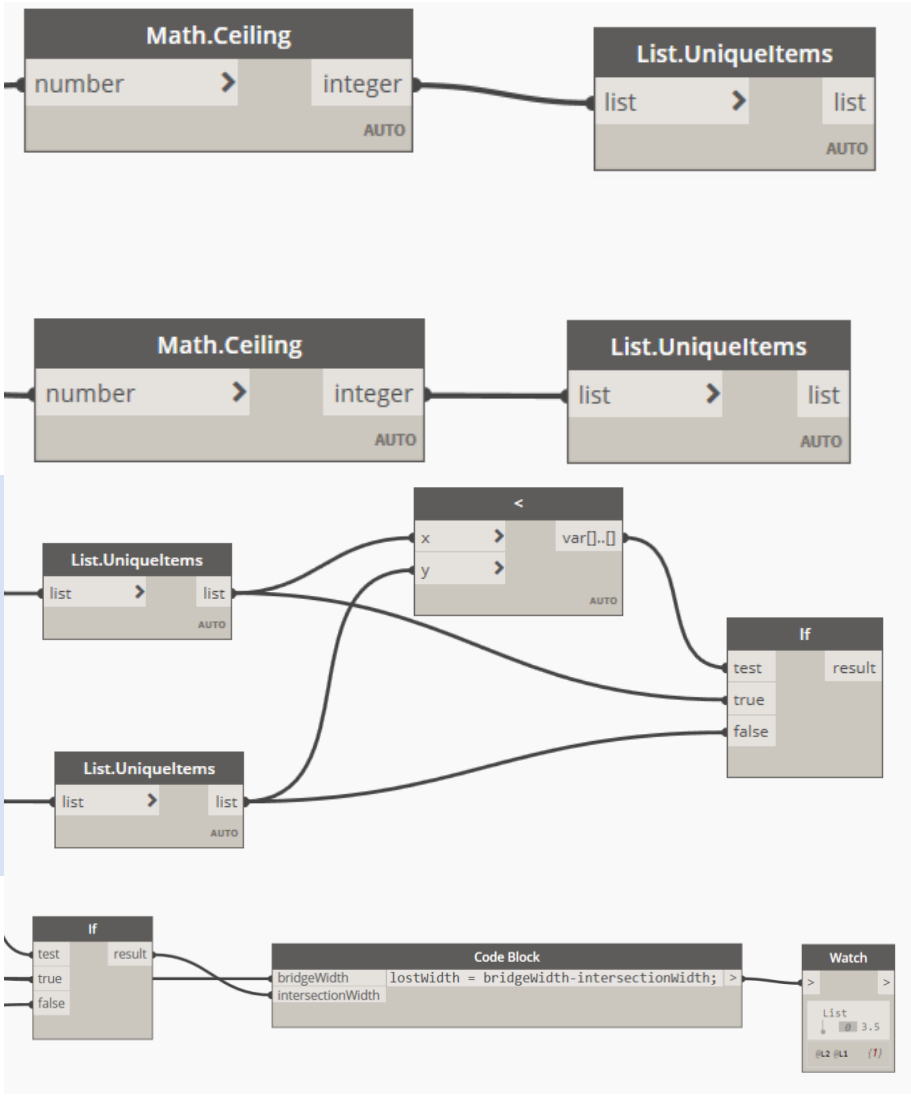
8.10 To get the distance between the intersections, add two nodes *Geometry.DistanceTo* and make the connections as shown in the figure. These give the distance between the two arches at surface level as well at headroom level.



8.11 Add two *Math.Ceiling* nodes to round up the distance values.



8.12 Because the distance between the intersections is expected to be the same at the start and at the end of the bridge, add two nodes *List.UniqueItems* to keep only one value.

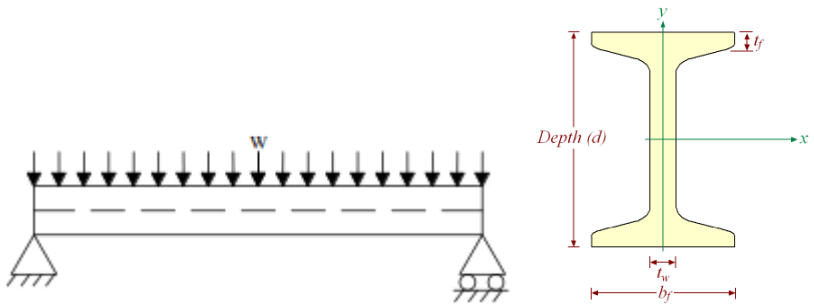


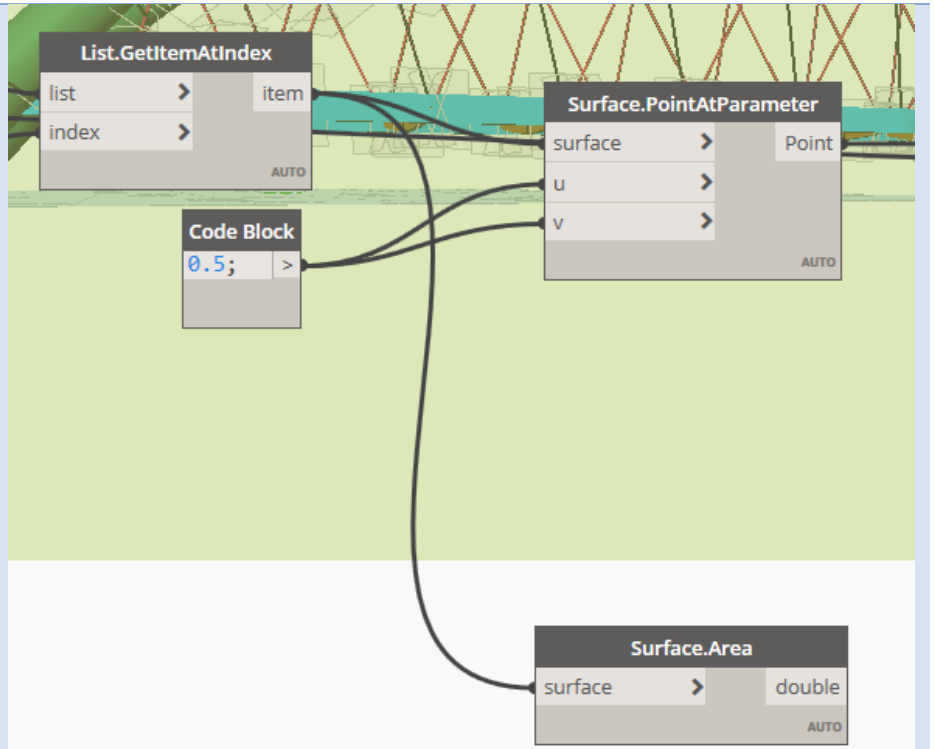
8.13 Add the node for the math operator < to check which value is smaller and an *If* node to select the minimum value.

8.14 To finalize, add a *Code Block* node to calculate to total lost width. The bridgeWidth is the value from the slider defined at step 2.1. Add also a *Watch* node to visualize the result. Of course, do not forget to wrap up the nodes added into a group to keep the definition organized.

9. Calculate the total steel weight for the cross girders

Another performance criterion we want to add to our parametric model is the total steel weight. Because usually the structural analysis is done using finite elements analysis software, we will only cover here the case of calculating the cross section of the cross girder. We will consider this as a simply supported beam, on which we distribute a linear load. We will calculate the section as an I-type beam.



Step	Instruction	Result
9.1	First, get the surface area of the deck using the node Surface.Area. As input, provide the surface you selected at step 8.3.	

9.2 Add a Code Block node and add to it the equations as shown in the figure bellow. All the steps are explained with the Code Block

StructuralAnalysisCrossGirders

surfaceArea
transversalSpan
bridgeWidth
elements

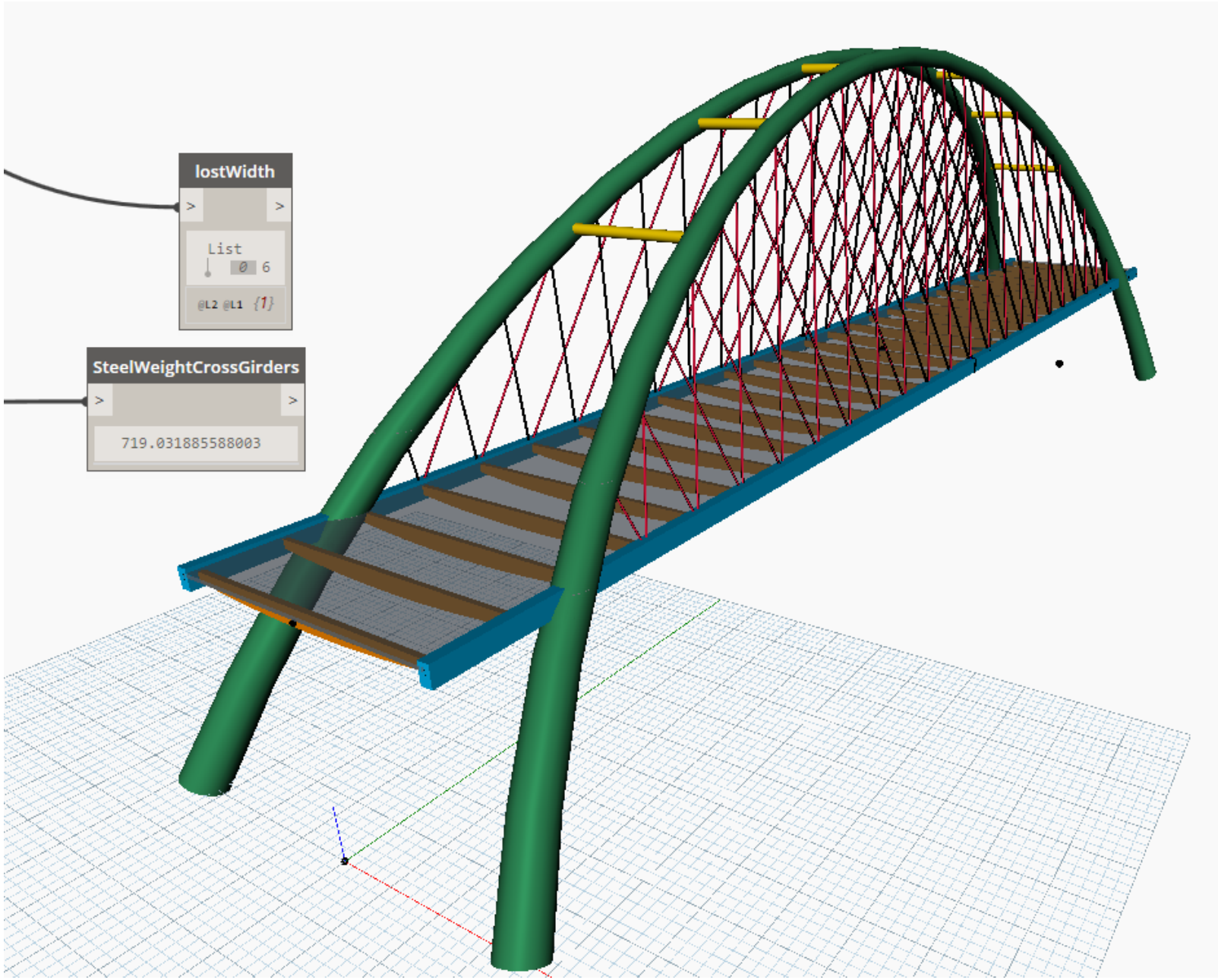
```
//load to consider for structural analysis unit: kN;
load = 79000;
//extracted from the deck solid (step...)
totalAreaDeck = surfaceArea;
//distribute the load on the deck surface kN /square meters
distLoad = load / totalAreaDeck;
//distance between cross girders(calculateted at step 3.2)
transversalSpan;
//liniarly distributed load on transversal girders kN/meter
linearLoad = transversalSpan * distLoad;
//lenght of the transversal gitders is equal to bridge width(step 2.1)
length = bridgeWidth;
//calculate maximum bending moment
Mmax = (linearLoad * Math.Pow(length, 2)) / 8;
//steel yield strength fy - steel grade: S355 kN/m2
fy = 355000;
Fb = 0.55 * fy;
//height of the web
hw = length / 20;
//imposed thickness for the web (m)
tw = 0.015;
//area web
Aw = hw * tw;
//flange area
Af = Mmax / (Fb * 0.78) - Aw/6;
//calculate total area of the cross section
At = 2 * Af + Aw;
//volume for a single girder
volume = At * length;
//weight one girder
weightGirder = volume * 7850;
//extract the number of girders based on the no. of curves created at step 3.14
noGirders = List.Count(elements);
//calculate tones of steed needed for all the crooss girders - tonnes
totalWeight = (weightGirder * noGirders) / 1000;
//calculate the reaction load at the bearing point kN
reaction = linearLoad*length/2;
```

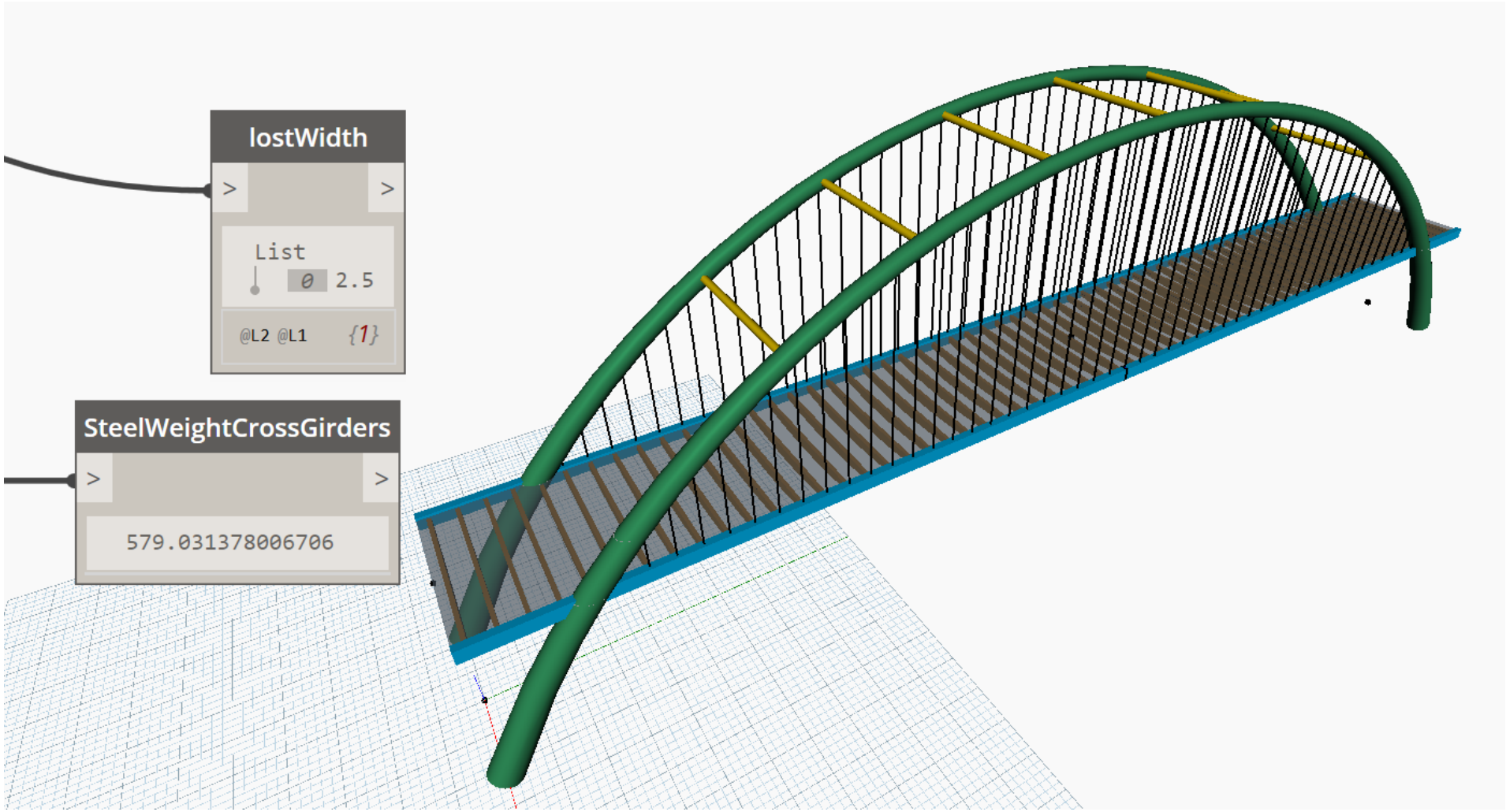
9.3 Add a watch node to the output of the totalWeight line to see the total weight of steel required only for the cross girders.

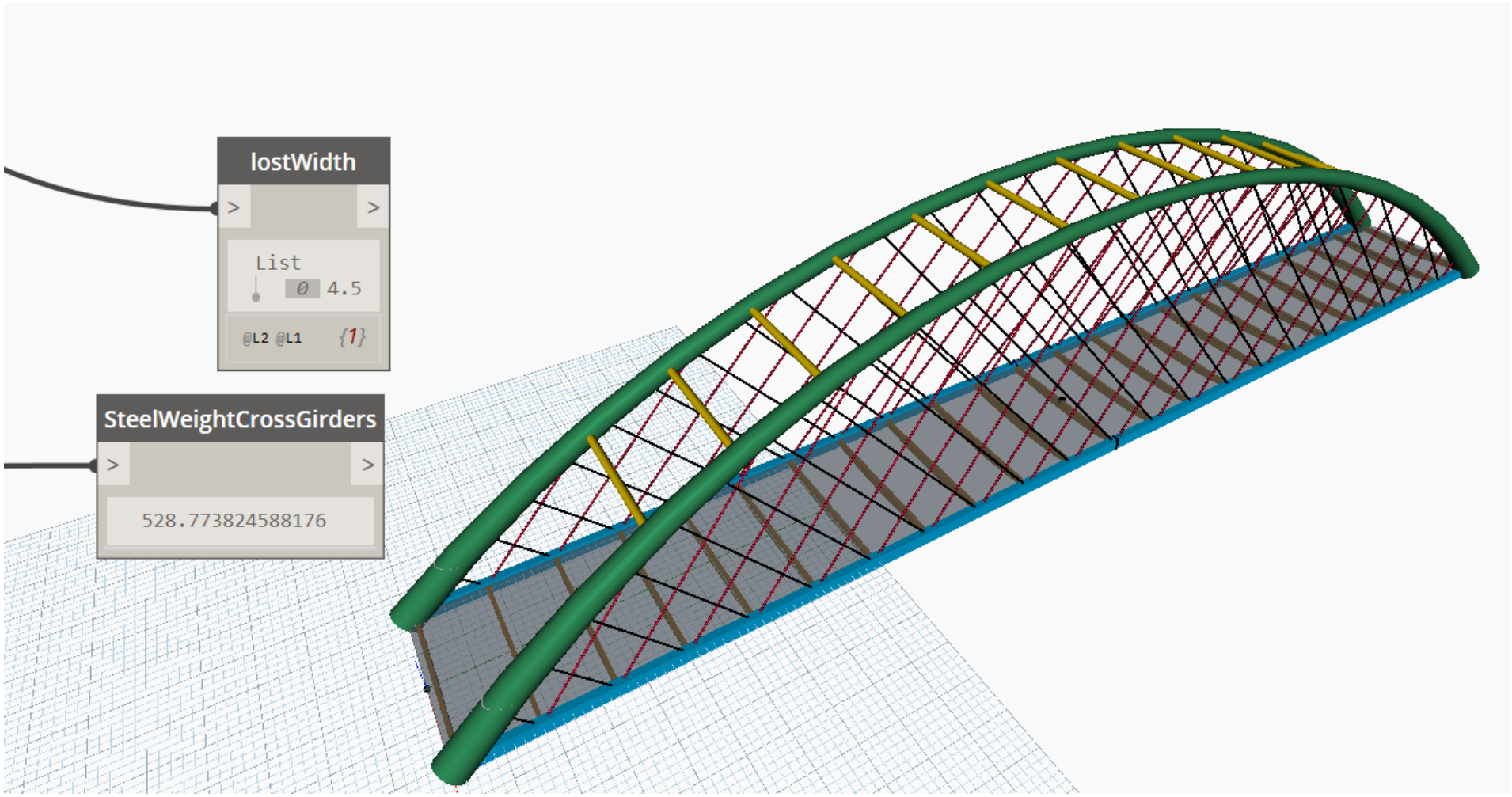
The model is not perfect, and of course, with each model we create, we need to be aware about its limitations. At this point, the structural engineers can create various configurations which can be used for the finite element's analysis. Some of the variations possible are included below.

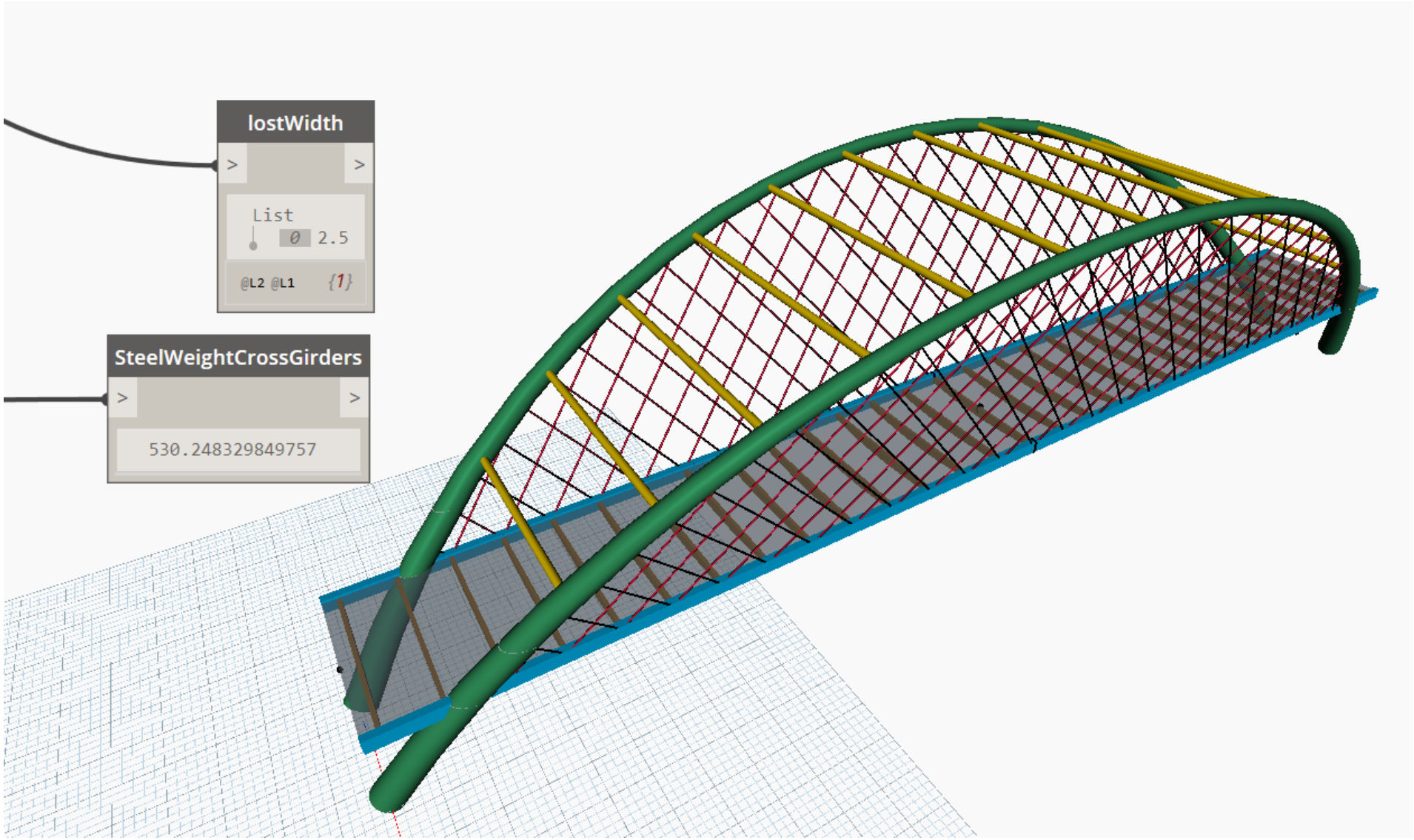
Civil Systems Engineering

Modeling Civil Engineered Systems




lostWidth
>
>
List
0
2.5
@L2 @L1
{1}
SteelWeightCrossGirders
>
>
579.031378006706


lostWidth
>
>
List
0 4.5
@L2 @L1 {1}
SteelWeightCrossGirders
>
>
528.773824588176


lostWidth
>
>
List
0 2.5
@L2 @L1 {1}
SteelWeightCrossGirders
>
>
530.248329849757