Annabel Sun
Dec 2, 2022

Procedural Floating Islands - CG398 Final Project

The goal of this project was to make a tool that procedurally generate a series of floating islands, using input data of a curve for the islands to follow along, with the ability to control many parameters of the generation.

The final project places islands along a curve according to a set frequency, and automatically scales these islands according to their distance apart from each other — more distant islands are bigger, while islands that are densely packed are smaller. Doing this scaling helps balance the island sizes proportionally, and makes sure the islands don't overlap each other. The island scale is also given a user-controlled multiplier.

Bridges are created connecting the islands to each other, following the curve. The bridges are procedurally modeled

All in all, the island has these controls:
Seed - Controls seed for all randomization
Island Frequency - Controls frequency of islands placed along curve
Island Jitter Percent - Controls amount of variation in island frequency and side
Island Size Multiplier - Multiplier for overall island scale
Height Scale - Scales height variation for islands

The bridges have these controls:
Bridge Slack - Controls amount of slack / bend in the bridges
Bridge Plank Width - Controls width of bridge planks
Bridge Jitter Percent - Controls amount of variation in bridge plank width
Bridge Width - Controls width of overall bridge
Bridge Height - Controls height of overall bridge

In addition, the models have been given simple vertex colors, which can be modified.
Island Top Color - Color for the tops of islands.
Island Base Color 1 - 1st Color for the island base gradient, controlling colors at the top
Island Base Color 2 - 2nd color for island base gradient, controlling colors at the bottom
Bridge Wood Color - Controls color for bridge planks and stakes
Bridge Rope Color - Controls color of bridge ropes.

Overall, I ended up approaching a lot of the problems in this project with VEX snippets and scripts. While there may have been simpler solutions to my problems, I often found that using VEX code to modify attributes gave me a lot of power when it came to controlling the generation and placement of geometries in my project.

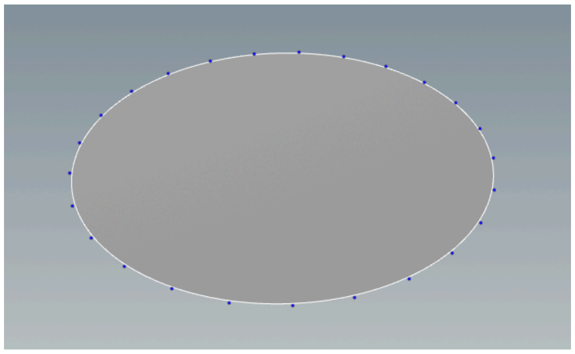
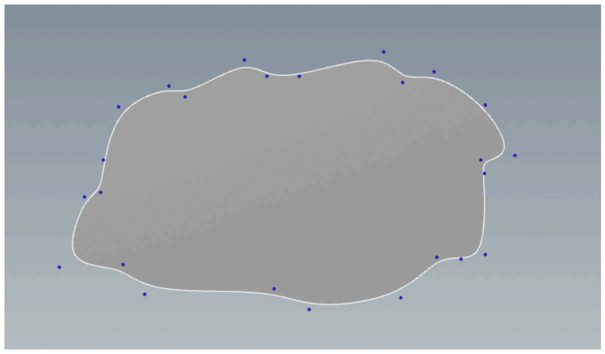
The main steps towards creating this project were:

1. Creating island geometry
2. Placing islands along the curve
3. Finding and placing bridge to connect islands
4. Creating procedural bridge geometry

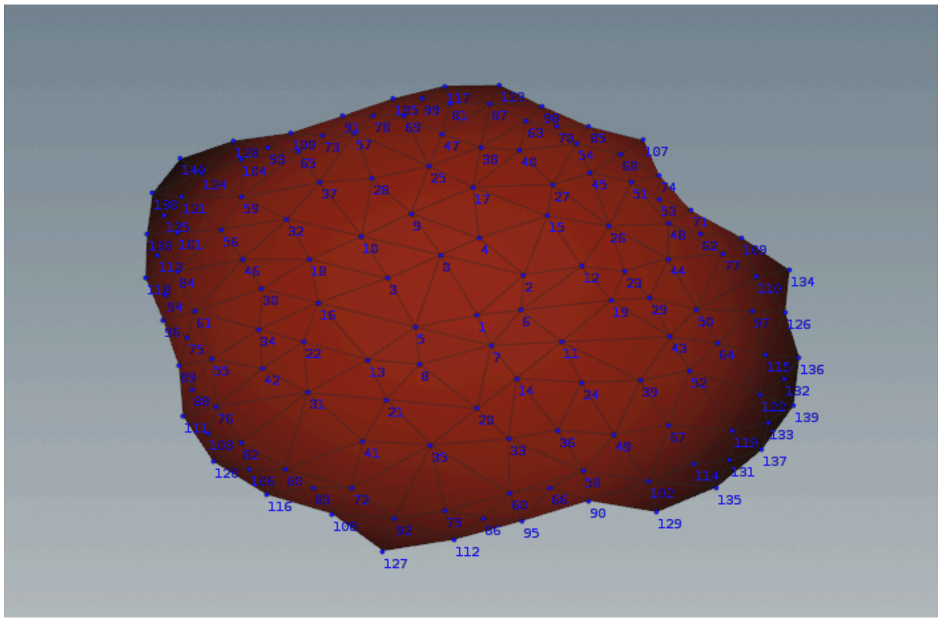
Part 1: Creating Islands
My island creation was based upon a simplification of this tutorial:
https://www.youtube.com/watch?v=udIjbBjGPtA

Because my project was focusing mostly upon the island placement, I chose to simplify my island creation instead of trying to learn about terrain tools or other complex generation methods.

I used a circle as the base form of my islands, then used a Point Jitter node to move the circle's points randomly in x / z. This gave me a random shape for my island.

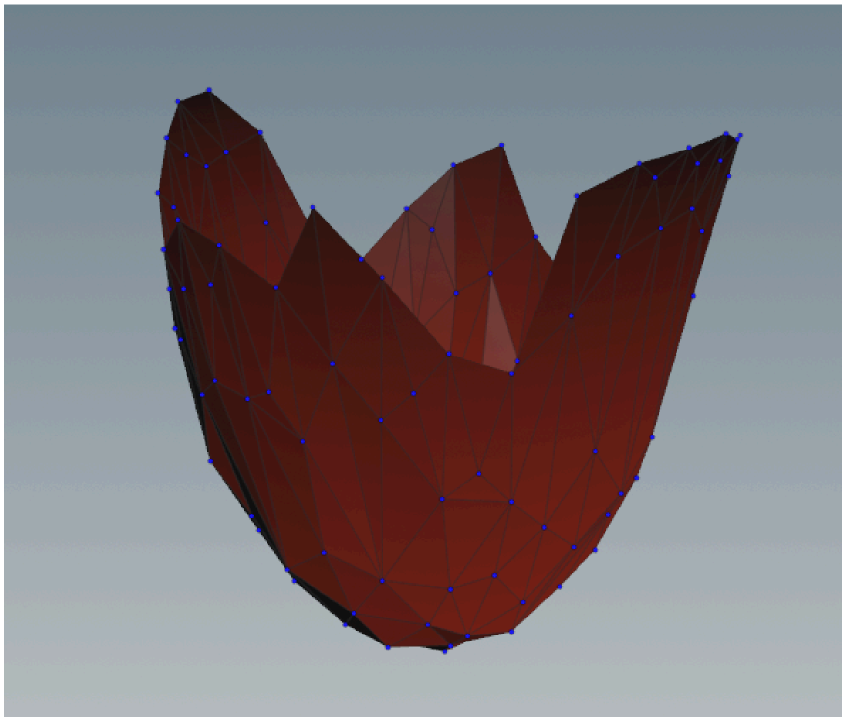



As a starting point to create the island's depth, I triangulated the base into many faces, and then sorted the points by their proximity to the center.

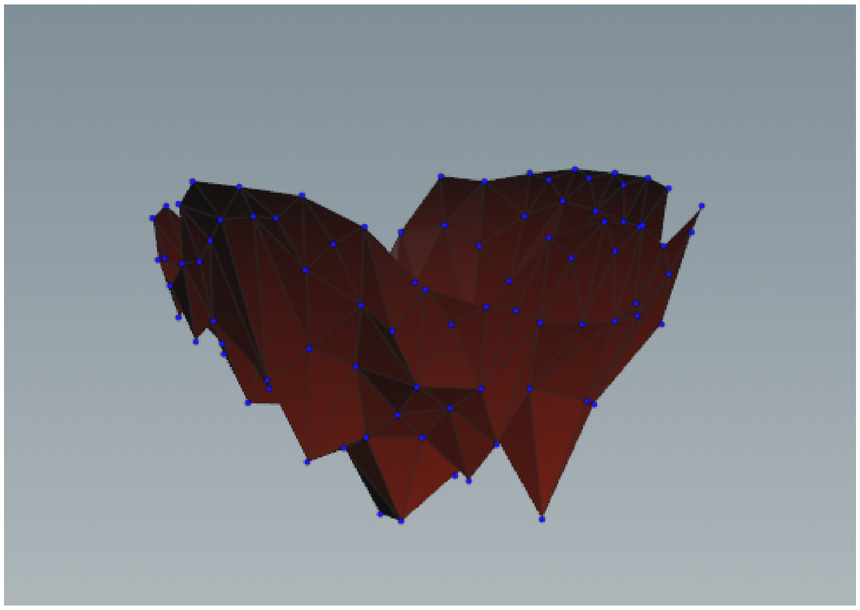


(In this photo, the red color channel is a visual representation of point number.)

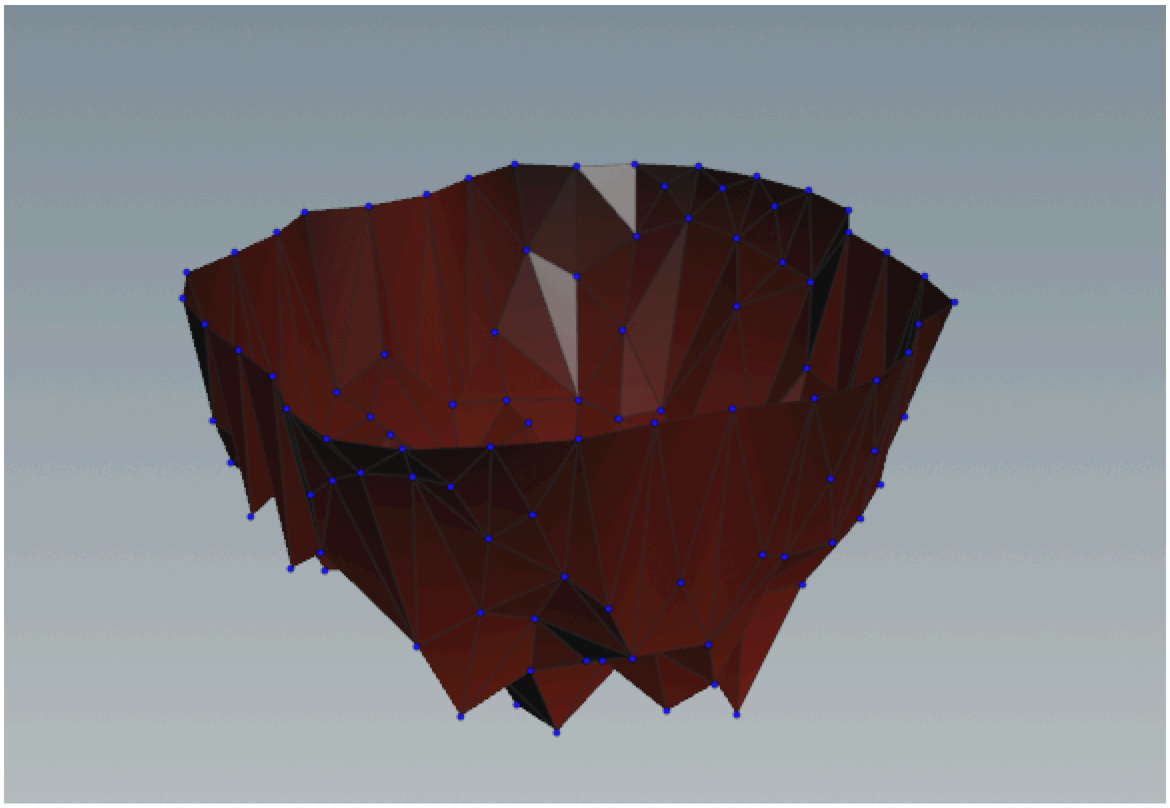
Sorting the points in this way allows us to get a value for each point’s proximity to the center, which we can then subtract from each point’s Y position, creating a curved “bowl” for the island shape.



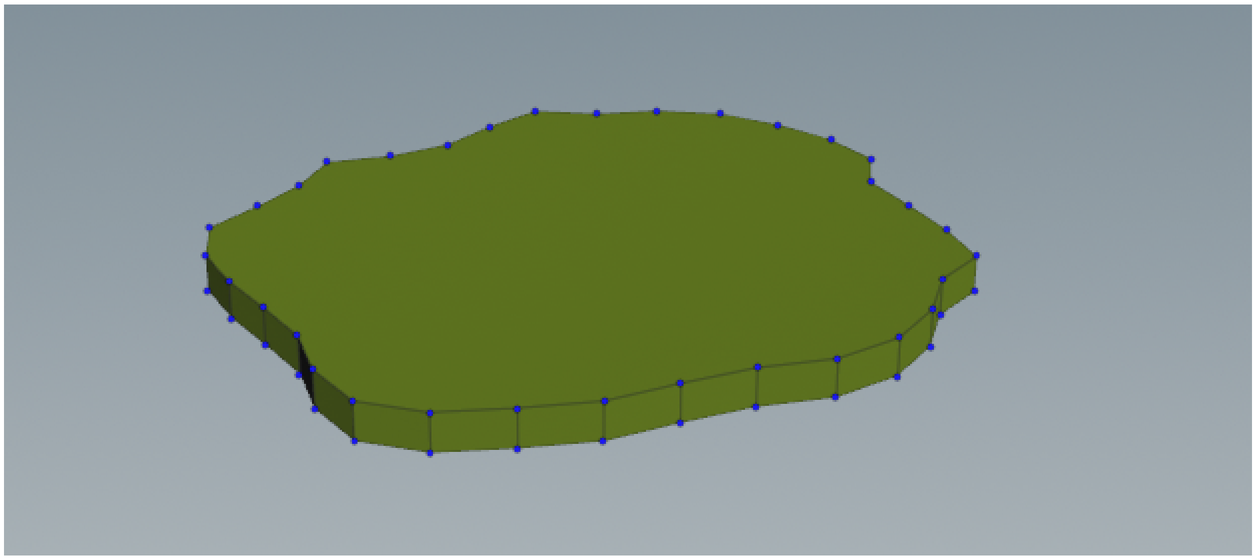
On top of this, we can apply some noise to the height to get some variation in the island’s base shape.



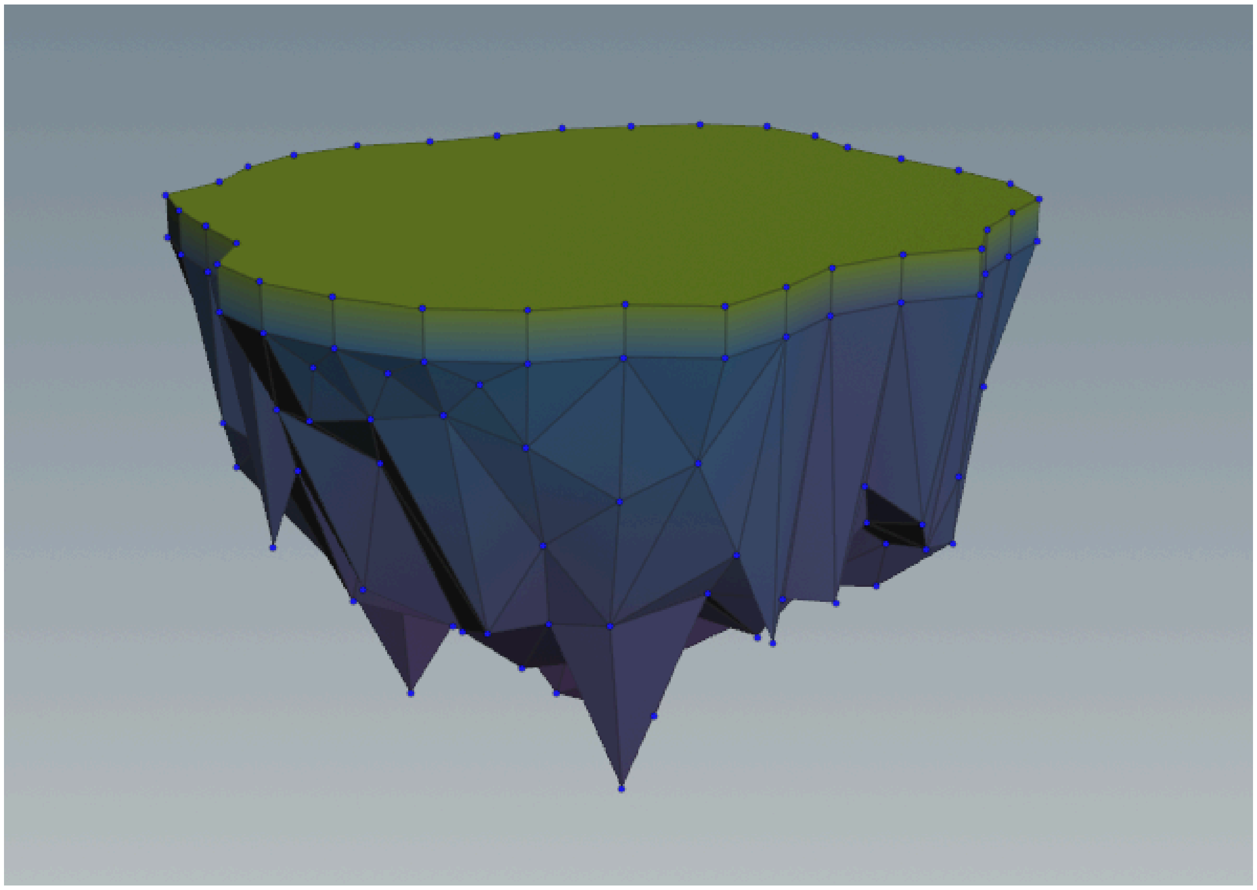
Then, I took the points along the edge of the island and raised them up to y=0, to flatten out the top of the island base.



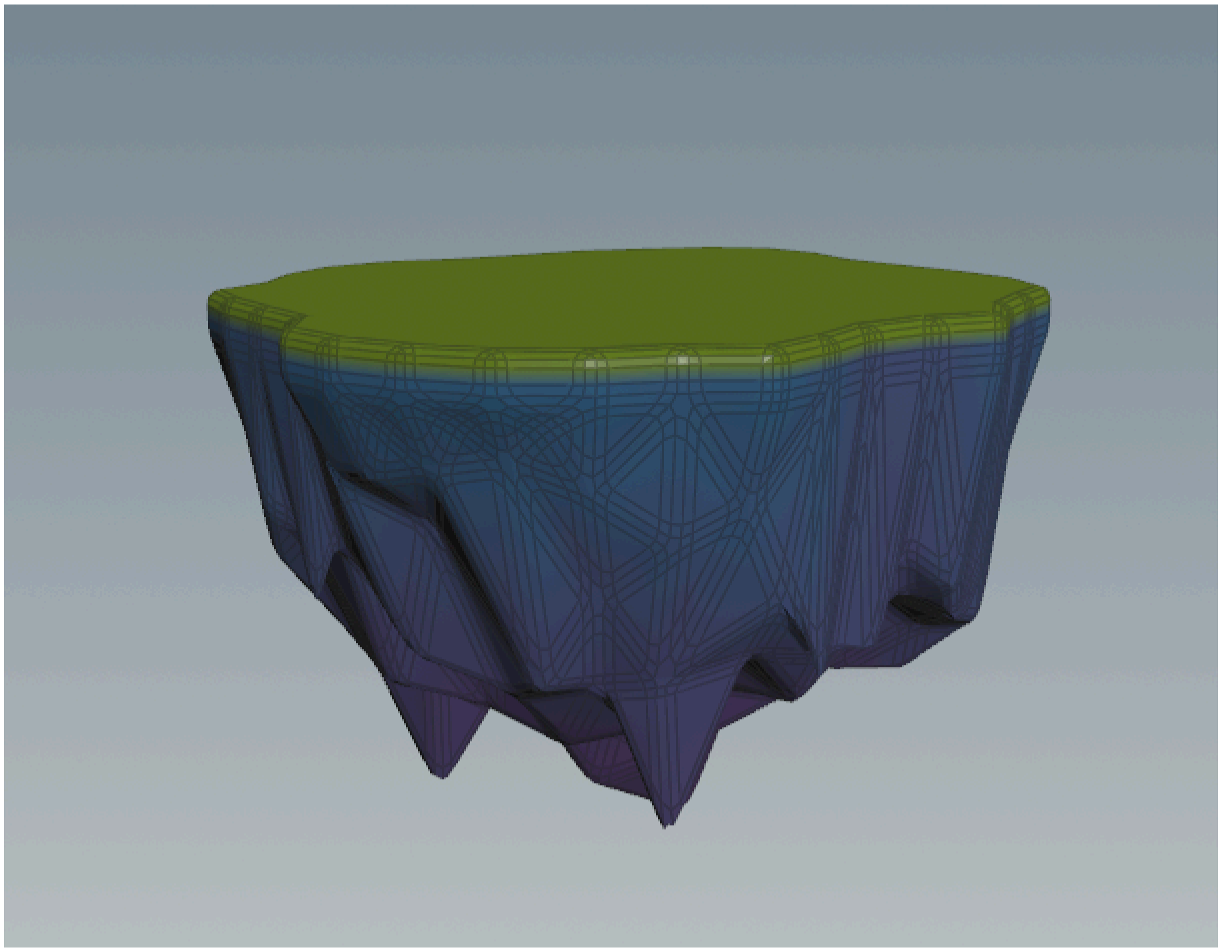
To create the top of the island, I did a simple extrusion of the original base shape:



Then, merged and fused the cap to the base.



I wanted my islands to look a little softer while still preserving a simple / low-poly feel, so I decided to use a bevel node to smooth out the islands without losing the island’s planes.

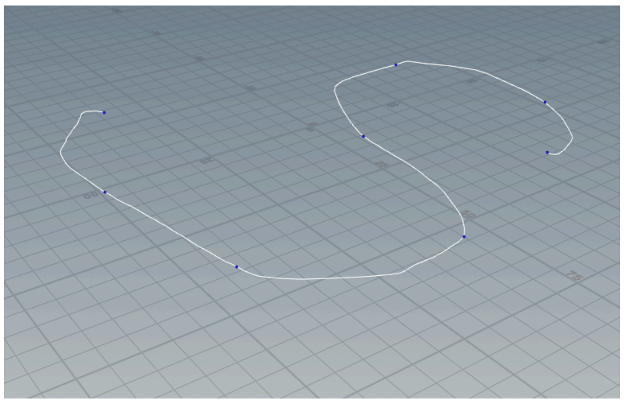
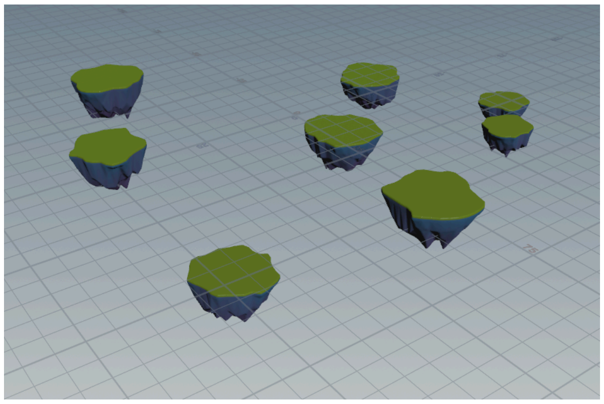


2. Placing islands along the curve

Island generation took me a surprisingly long time to figure out. My initial approach was to scatter points along the curve using the Scatter node, but I found that this didn't give me control like I wanted - the scatter node sometimes created islands that were extremely far apart from others, and it was difficult to intuitively control the density of the islands. In addition, the points that were created by the Scatter node were not numbered in any particular order, and the nature of user-inputted curves meant that the points would be hard to automatically number.

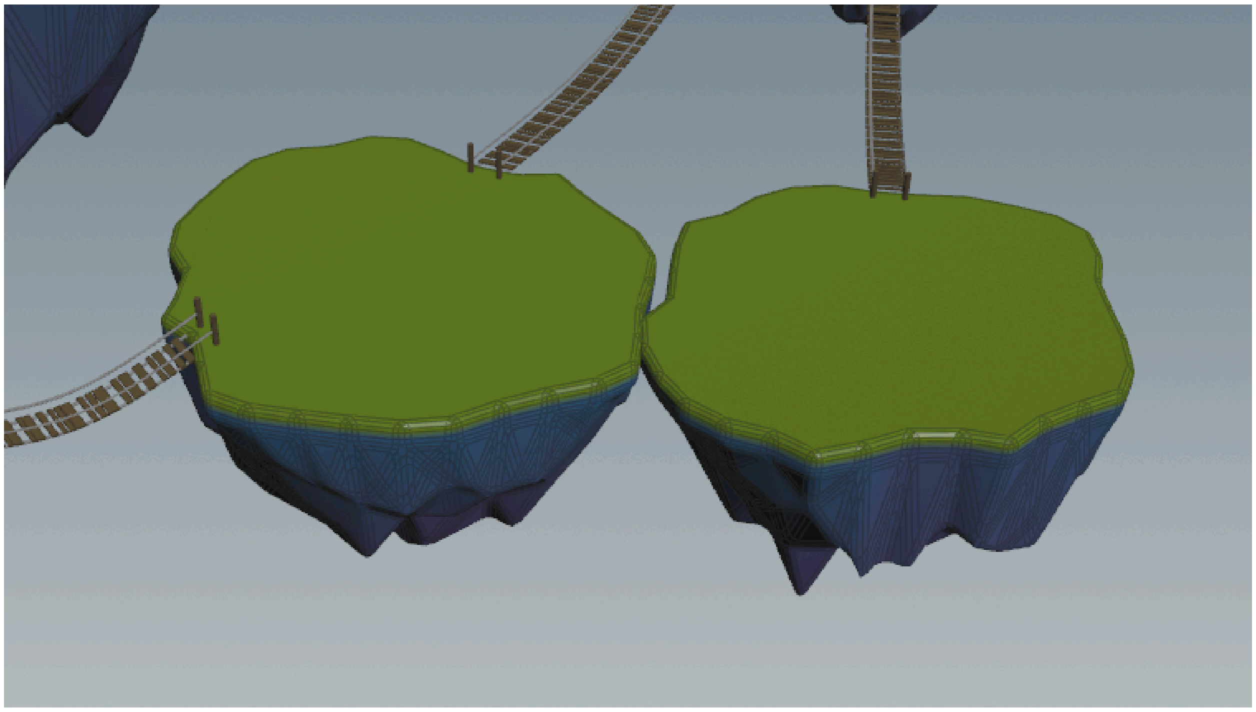
Instead, I decided to create a VEX script that would travel along the curve, and place islands at random intervals apart from each other.

I found that the best way to trace the curve was by resampling the curve to define it with fairly dense points, and then stepping through the curve point-by-point. This was one of the ways that I had to get used Houdini's unique way of thinking through problems - my initial approach was to simply use x/y/z distance, but the code was much simpler if I took advantage of Houdini's ability to iterate through points, and tracking position using the @P attribute.

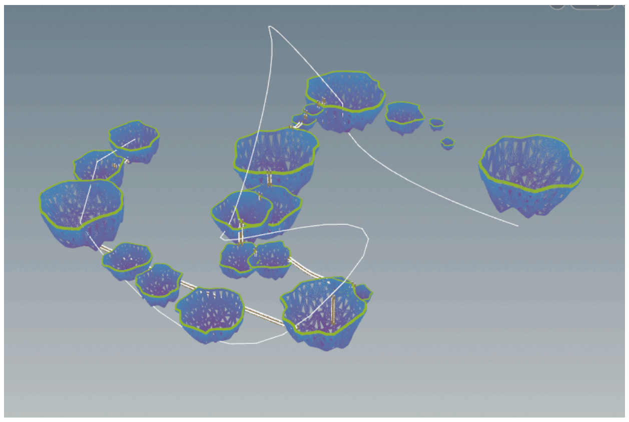



To scale the islands, I took advantage of the Copy To Points node's ability to transform copied geometry using the target point's orientations.
In order to avoid islands overlapping each other, I took proximity to other islands into account when scaling.. I scaled the island to span half the distance to its nearest neighbor — because of this, when at maximum size, two neighboring islands will just barely touch each other.

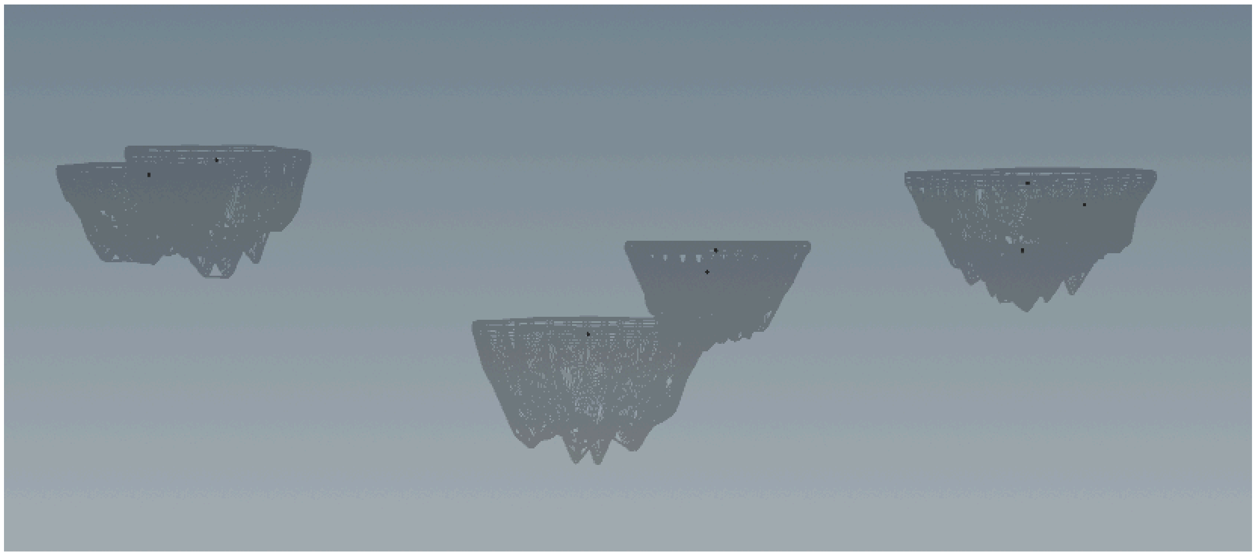
I used the nearpoints() function to find each island's nearest neighbor, so even islands that are not connected to each other will scale properly.



Initially, I allowed the user-defined curves to be three-dimensional, but this caused severe issues with overlapping islands and bridge placement if the curve had too much height variation or looped back on itself.

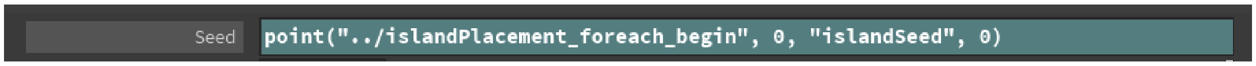


Because of this, I decided to flatten the input curve to be 2-dimensional. I added in simple height randomization, scaled by user input, to allow height variation for islands.



To give each island random generation, I had make an attribute for a random seed while placing points, then reference that seed when actually generating the island. For some reason, I found this process extremely unintuitive!

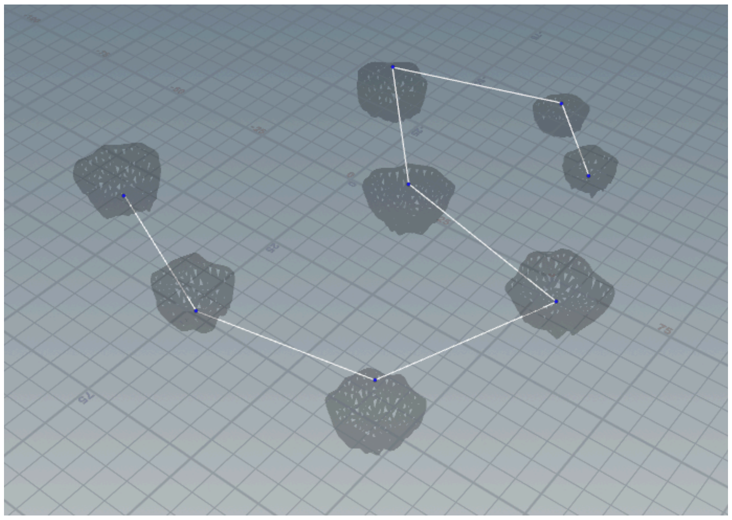
The solution was to use a point() function within the Seed channel (which I created on an attribute wrangle node), to be able to retrieve an attribute from a point. The point function can reference another piece of geometry, so I was able to pull from the beginning of the for-loop used for placing islands on points.



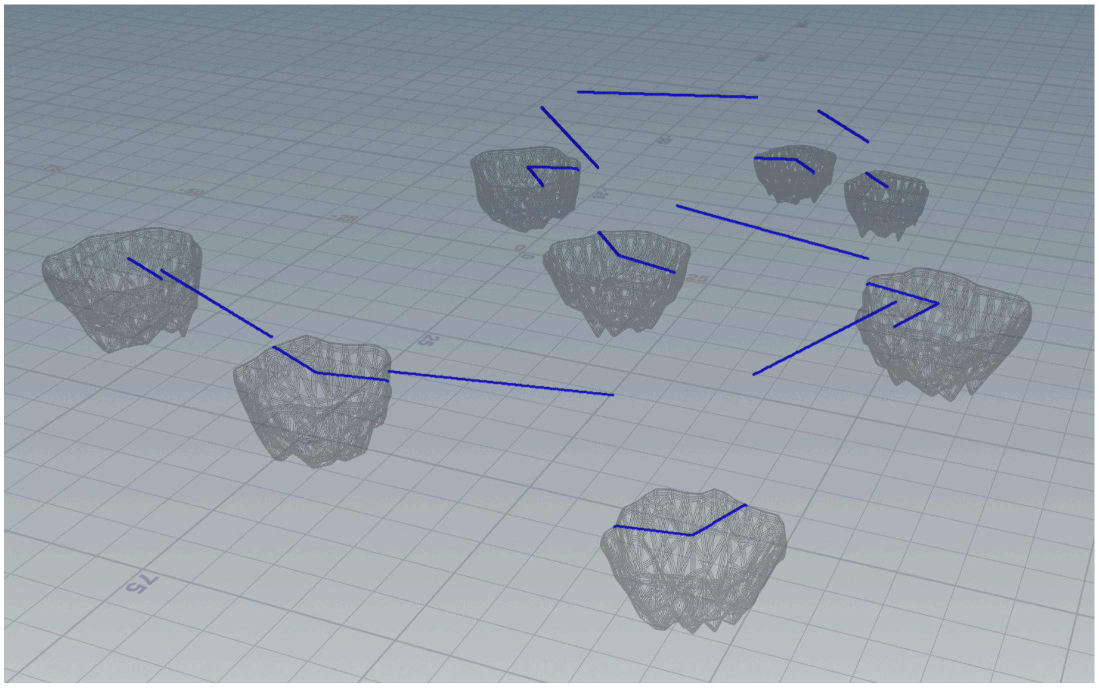
```
point("../islandPlacement_foreach_begin", 0, "islandSeed", 0)
```

3. Placing bridges between islands

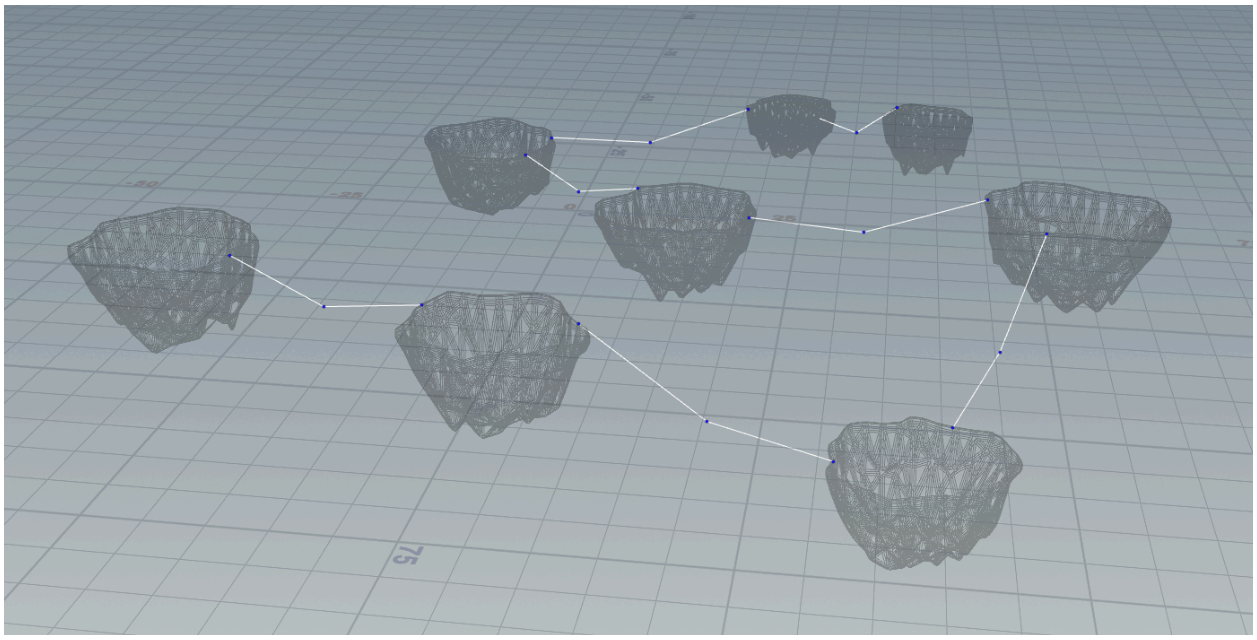
For bridges, I started by creating a new curve that connected the center points of each island. The challenge here was to find the point along this curve, right at the edge of an island — connecting up these island edge points would create bridge connections.



I used the Ray node with a downward projection to determine which parts of this line were above islands, which allowed me find and keep the end points of each "island" segment to serve as end points for bridge placement.



To give slack / a curve to the bridge shape, I found the midpoint of each bridge, and transformed it downwards.

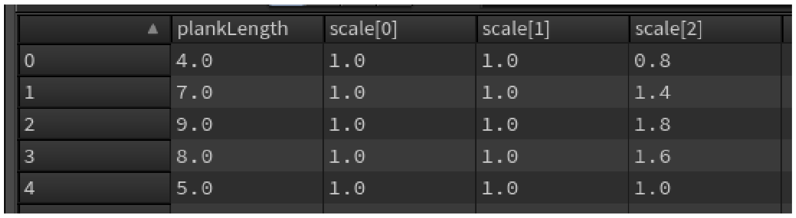


4. Creating Procedural Bridges

Bridge creation can be further broken down into three parts: planks, stakes, and ropes.

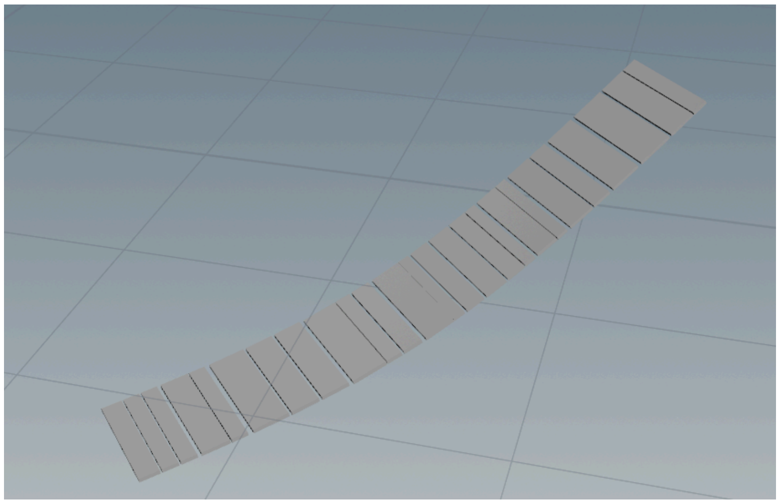
The planks were placed with a very similar manner to islands: I wanted to add some complexity to the bridge by adding variation to the width of each plank on the bridge, and I could use the same method as the island placement for this.

To scale the bridge planks, I used a different approach from the islands. I wanted the planks to be flush with each other, while the approach I used with the islands sometimes created large gaps. To solve this, I kept track of all my generated bridge lengths by using attributes, and used the attribute to scale my planks instead of scaling them seperately.

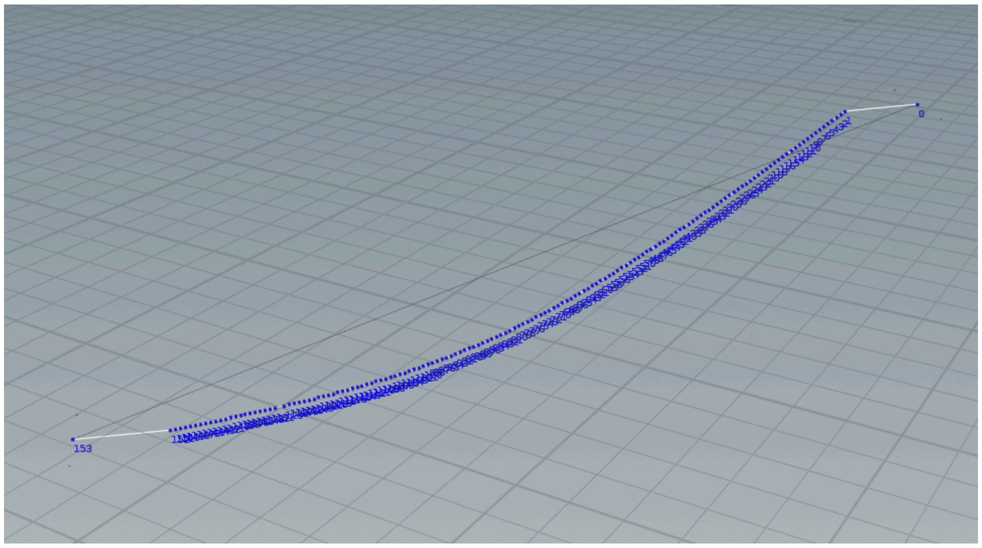


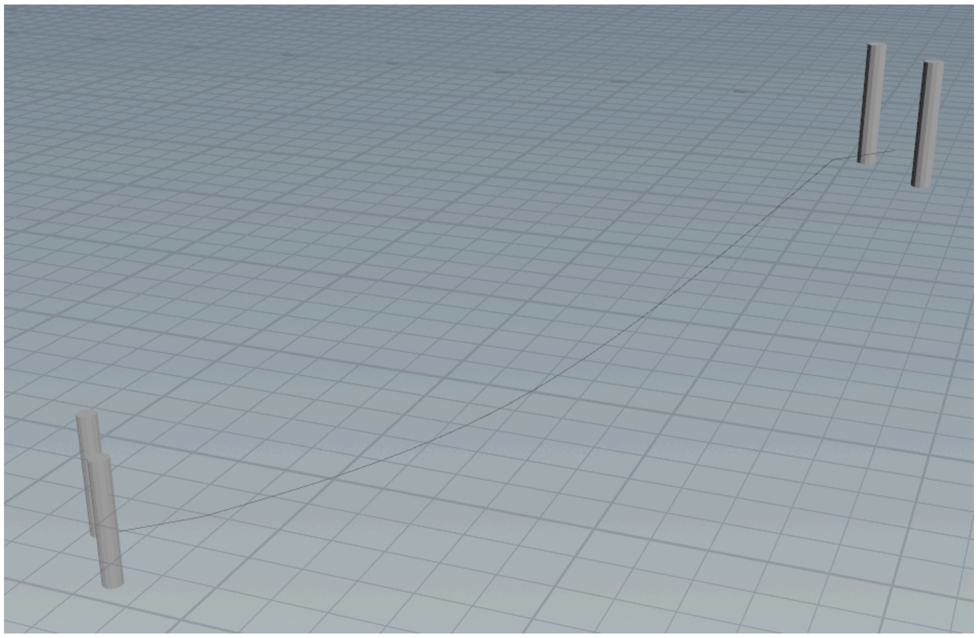
▲	plankLength	scale[0]	scale[1]	scale[2]
0	4.0	1.0	1.0	0.8
1	7.0	1.0	1.0	1.4
2	9.0	1.0	1.0	1.8
3	8.0	1.0	1.0	1.6
4	5.0	1.0	1.0	1.0

The only added complexity the bridges created was the need to rotate planks to follow the direction of the bridge. Luckily, this ended up being a fairly easy fix - by using an Orientation Along Curve node, I was able to create normals, which the Copy To Points node was able to reference.
I referenced this video for orienting my planks:
https://www.youtube.com/watch?v=x3aeEutYulw&t=537s

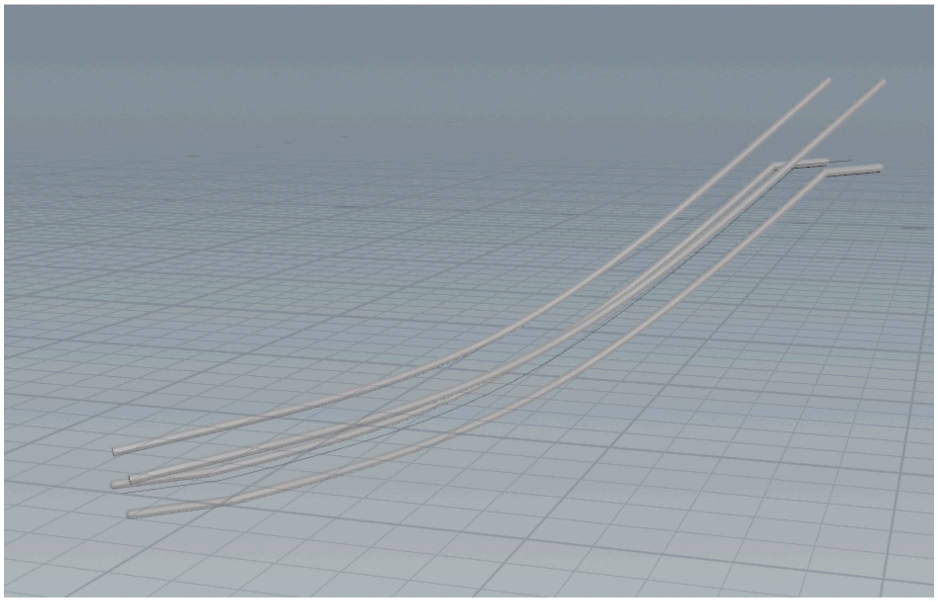


When placing stakes for the ends of the bridges, I took advantage of the normals generated by the Orientation Along Curve node to find the direction necessary for placing stakes, using vector math to find the perpendicular direction to the curve's normals.
To create new points at the start and ends of the bridges, and ensure the stakes would not be floating off the islands, I also made sure to push the start and end points of the curve backwards. The stake points were placed referencing these points, at a distance determined by the bridge's width.

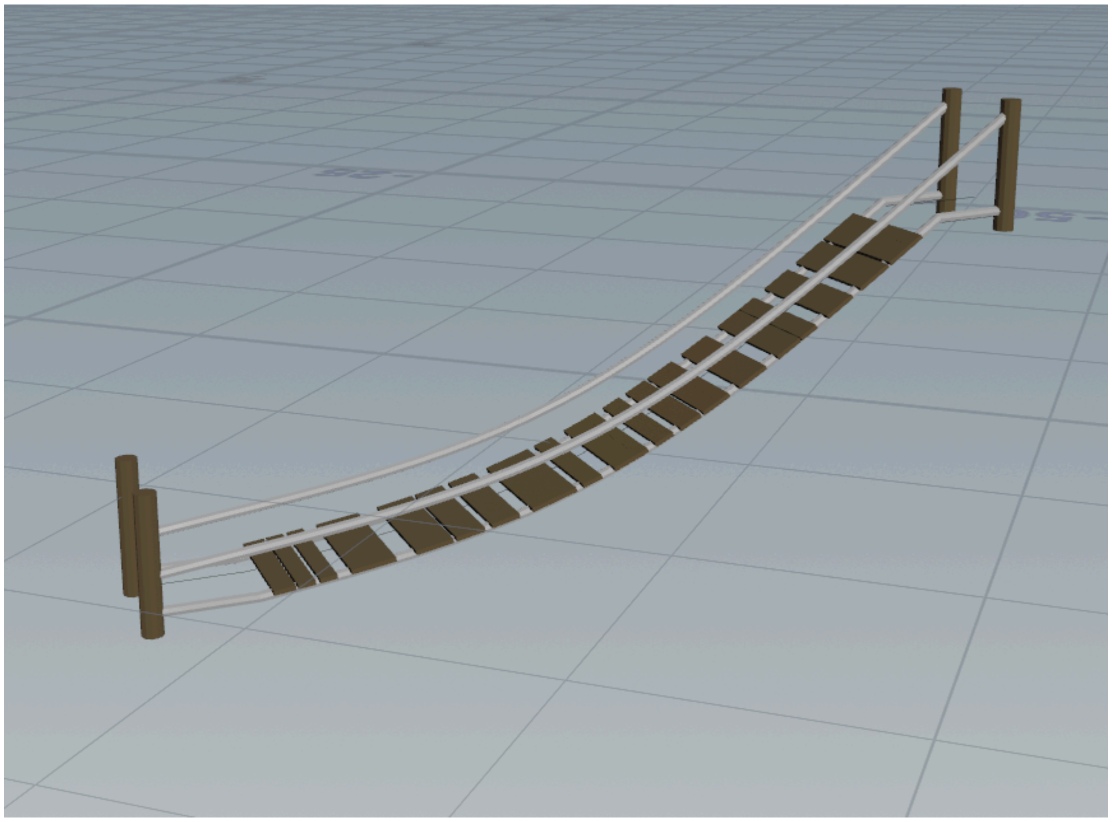




The ropes used a similar technique to the stakes for placement. However, instead of placing new points and copying geometry to them, I simply took the curve and used a Sweep node to create geometry. I created two sets of ropes - one for the bottom of the bridge, and one to act as handrails / support.



Completed bridges:



Final generation: