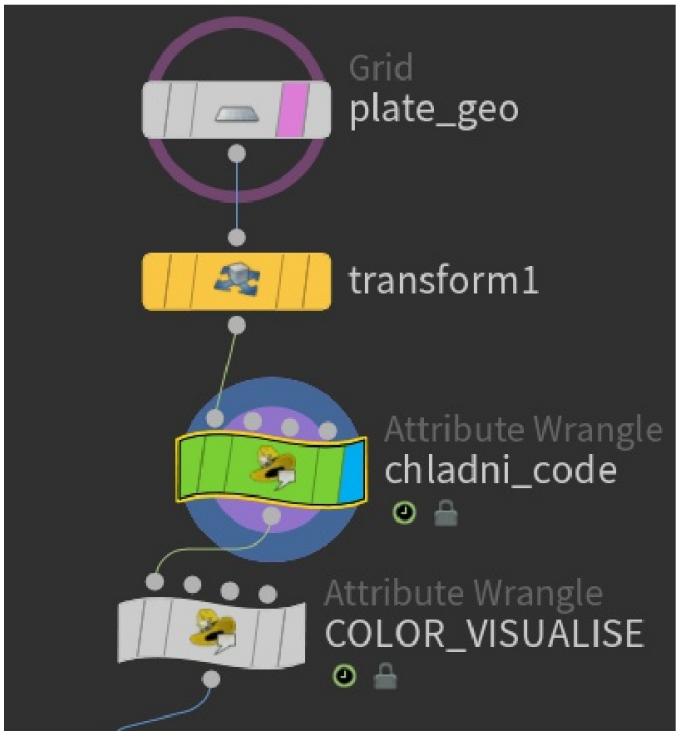
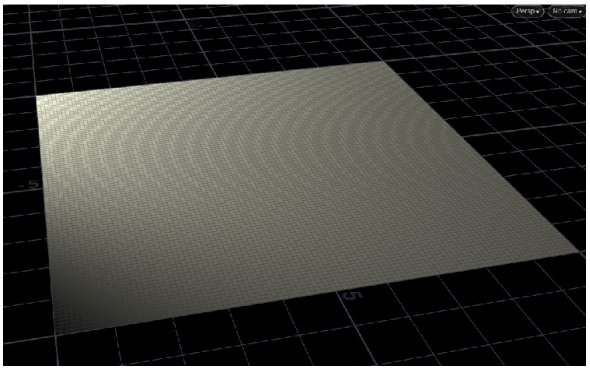
Documentation, Ivan Iliev VK, VFX Chladni Plate Animatio, Aufbaukurs 3D II



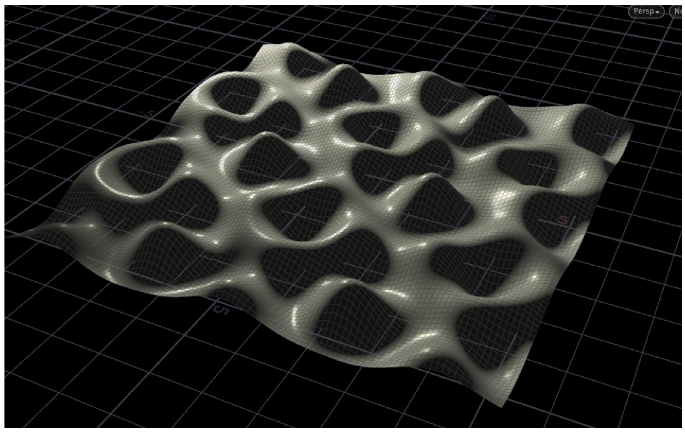
We begin our project by creating a simple grid object. The resolution (rows and columns) of the grid will be very important for further animation.



Than we implement the code for vector displacement of **@P** attribute. At row 16 and 17 is the equation for Chladni waves.

VEXpression

```
1  float L = ch("Dimensions");  // Dimensions of the rectangular space
2  float n = ch("nodallinesinxdirection");  // Number of nodal lines in x direction
3  float m = ch("nodallinesiny3");  // Number of nodal lines in y direction
4  float A = ch("Amplitude");;  // Amplitude of the wave
5  float f = ch("Frequency");;  // Frequency of the wave
6  float t = @Time;  // Current time
7
8  float x = @P.x - 0.5 * L;
9  float y = @P.z - 0.5 * L;
10 float density = noise(@P * ch("densityScale") + ch("densityOffset"));
11 density = clamp(density, 0, 1);
12 @density = density;
13
14 float phase = 2 * M_PI * f * t;
15
16 float displacement = @density * A * (cos(n * M_PI * x / L + phase) *
17 cos(m * M_PI * y / L) - cos(m * M_PI * x / L + phase) * cos(n * M_PI * y / L));
18
19 @P += displacement * {0, 1.4 , 0};
20 // Apply displacement to the z-coordinate of the point position
21
```

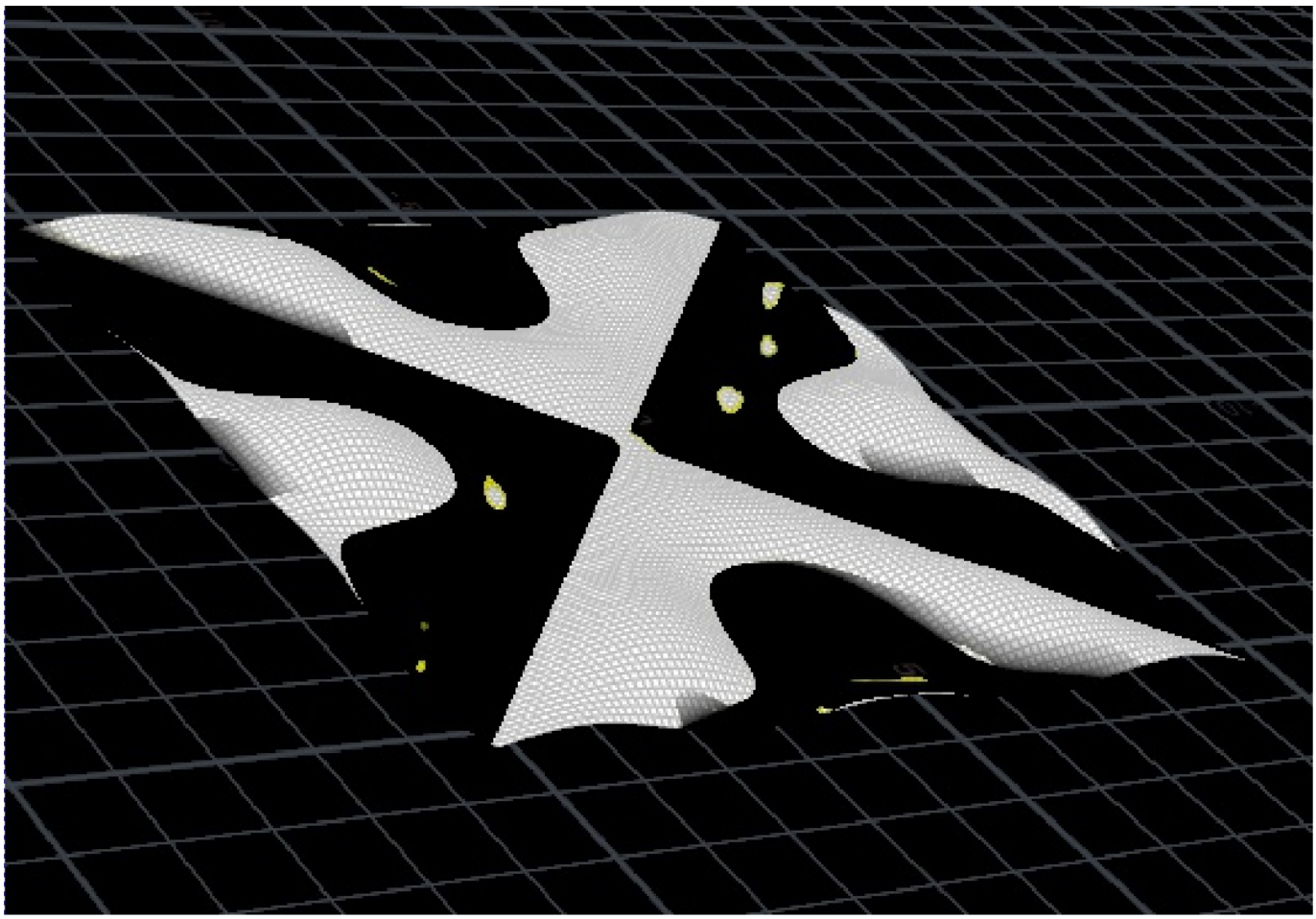


The code is deforming the geometry. However, if we just try to drop packed rigid bodies like sand or particles with adjusted low or high or even low frequencies, it is very computationally intensive, thus not practical for VFX and visualization purposes. Test simulations with low particle count show that it works, but my Ryzen 7 5800 shows to be incapable of doing this in a reasonable time because high-frequency movement requires a very high simulation substep value. So I decide to extract the shape in a different way.

Expression

```
1 v@Cd = v@P.y;
2 v@Cd=v@Cd*10000;
```

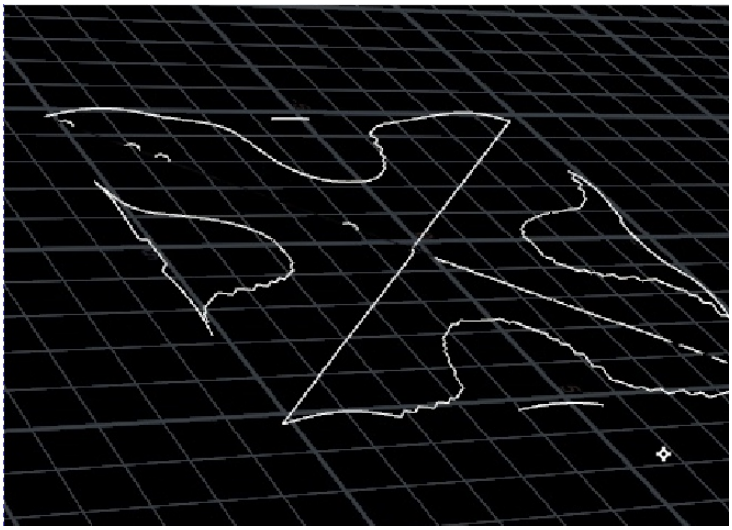
I adjusted the @Cd color attribute to have black and white values based on position with a very simple equation.

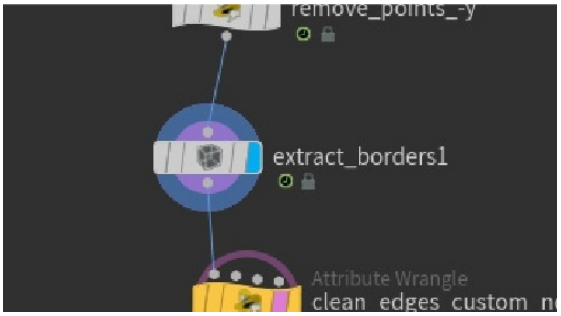


Then we have high contrast visualization of our Chladni waves. It could be done with any attribute; however, I decided to use the color attribute because I can use it as a visual reference.

```
Expression
1 if(@P.y < 0)
2 {
3 removepoint(0, @ptnum );
4 };
```

Please correct the spelling if necessary: We remove, with a simple conditional statement, the points that have a position less than 0. In order to work, the grid should be placed accordingly at y = 0.

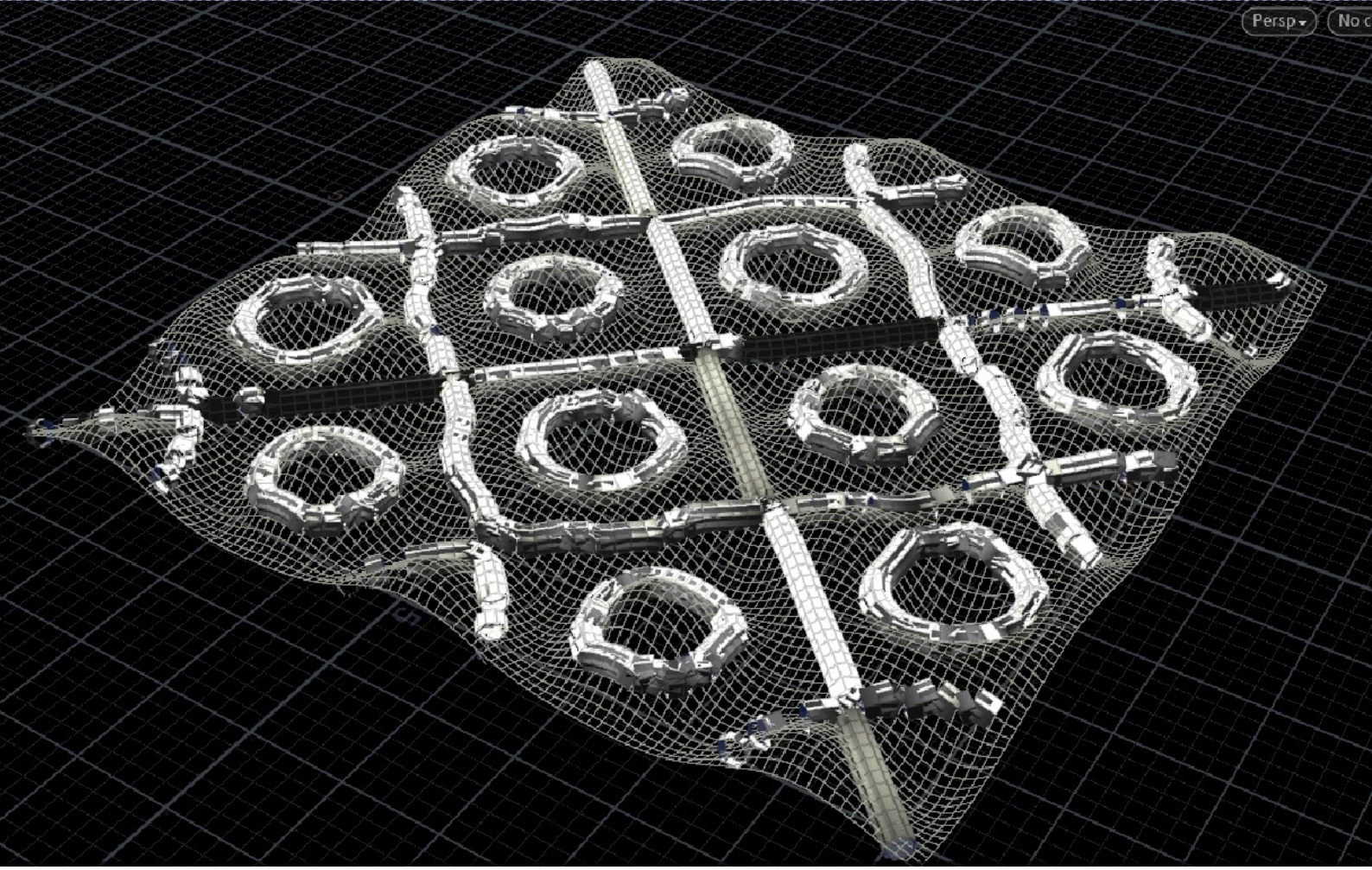


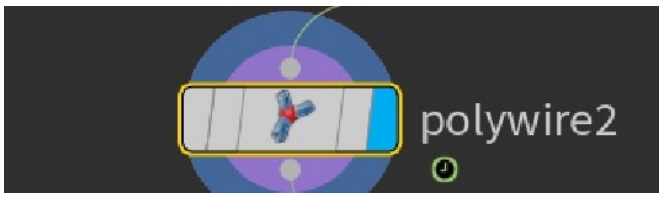


Please correct the spelling if necessary: Then we extract the borders with the Labs Extract Borders node. We now have lines that

```
Snippet
1  float threshold_x = ch("threshold_x");
2  if (@P.x < threshold_x) {
3      removeprim(0, @primnum, 1);
4  }
5
6  float threshold_z = ch("threshold_z");
7  if (@P.z < threshold_z) {
8      removeprim(0, @primnum, 1);
9  }
10
11 float threshold_x_low = ch("threshold_x_low");
12 if (@P.x > threshold_x_low) {
13     removeprim(0, @primnum, 1);
14
15 }
16
17 float threshold_z_low = ch("threshold_z_low");
18 if (@P.z > threshold_z_low) {
19     removeprim(0, @primnum, 1);
20 }
```

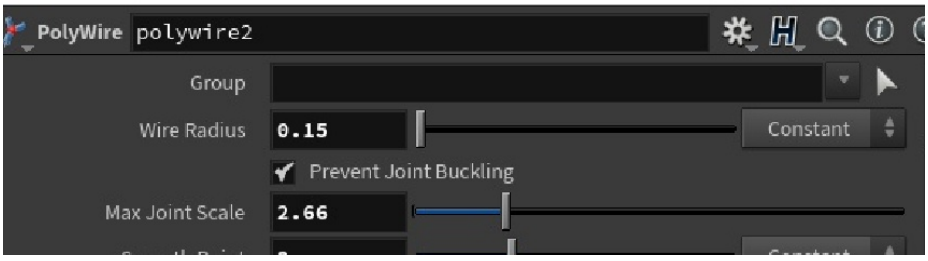
We trim the edge with simple code because extract borders is catching the outer edges of the grid.

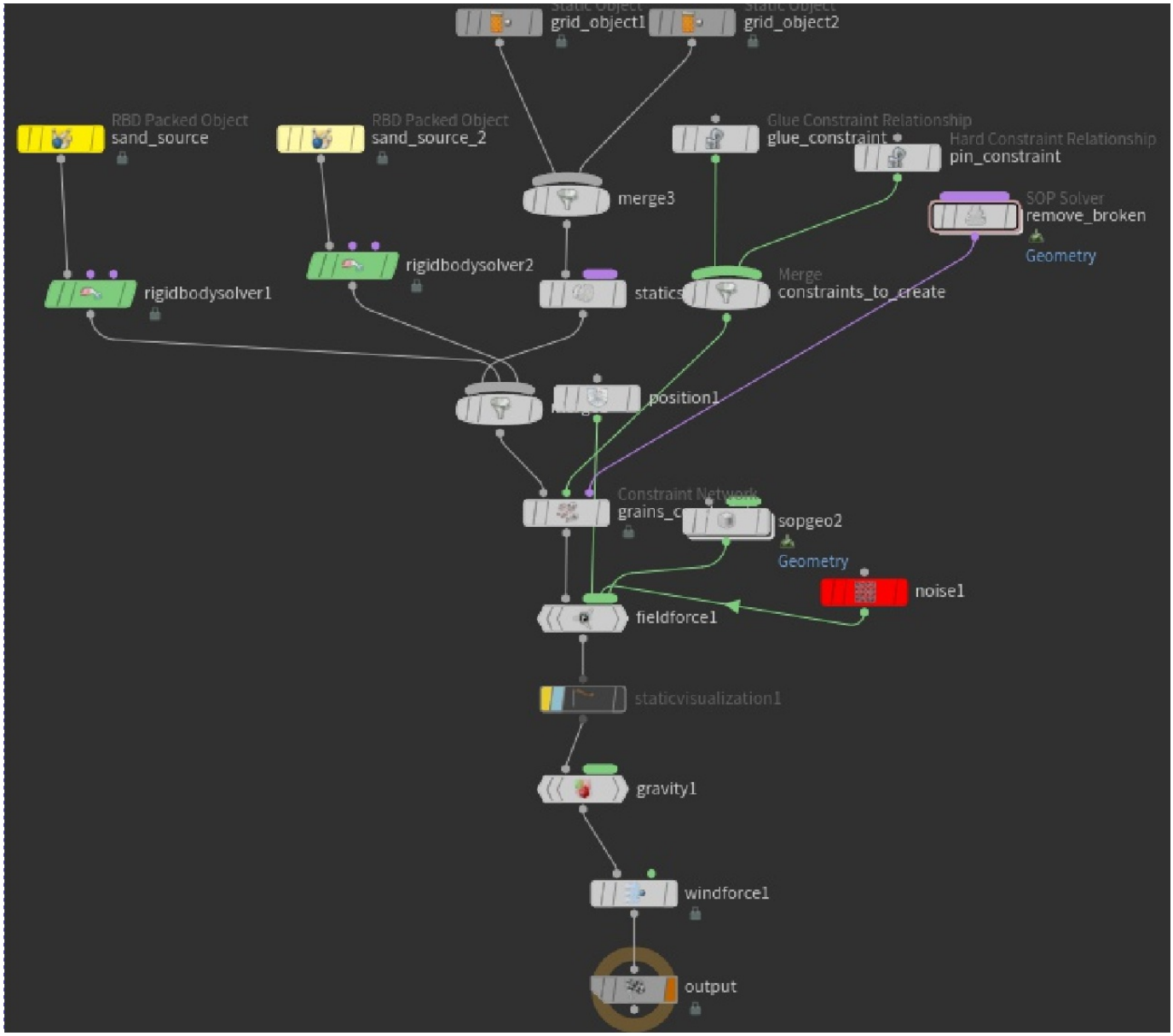




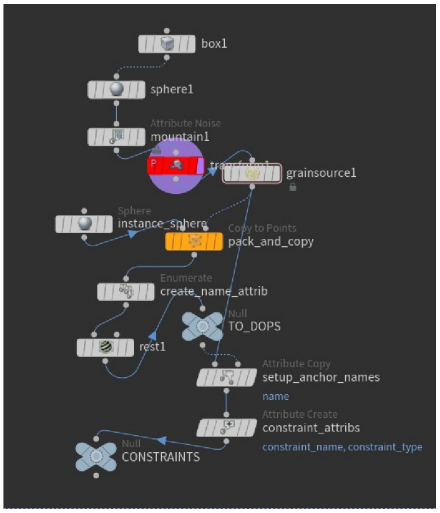
Then we use the Polywire node to build a tubular shape around the extracted shape. The radius of the polywire has a significant effect on how forces will be formed in the DOP network.

A smaller value will create sharper forces, while a bigger radius will create a softer shape.





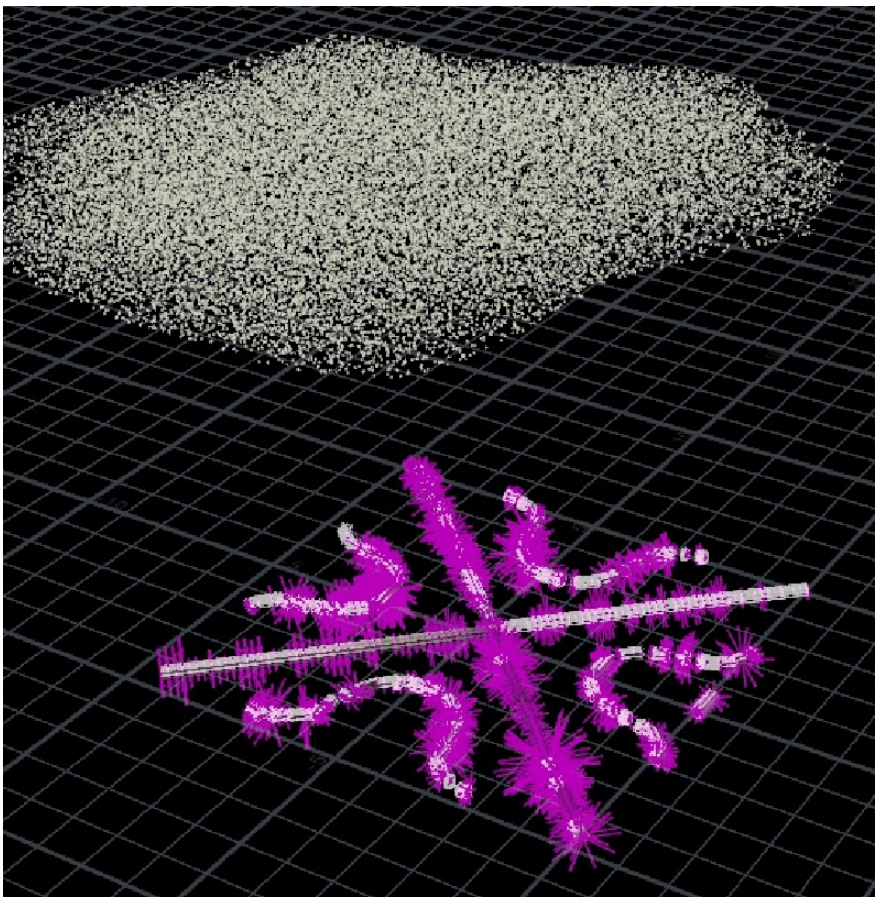
Here is a simple example of my Dynamic network. I removed some microsolvers and nodes to have a better view. The most important node here is the FieldForce node. For the FieldForce shape, I used the tubular geometry that I created in previous pages. For the force vectors, I used the **@N attribute** with reversed values. It is very suitable to create an attracting force due to its native perpendicularity to the geometry. When feeding this geometry as SOP geometry, it needs very few adjustments and settings to have a working magnet-like force.



We are using **packed RBD objects** for our network. It's a very trivial network, but we don't need anything more. It's worth mentioning that some of the attributes that are created automatically from the shelf tool are better to be disabled because this could accelerate the conversion from .bgeo to .usd cache, which is required in my case for Karma rendering. Without .usd cache files, I will have less accurate motion blur, and I will be forced to make it on a 2D plane.

```python
import hou

# Get the DOP_IMPORT_PYRO geo node
dop_geo_node = hou.node('/obj/DOP_IMPORT_PYRO')

# Get the filecache2 node inside the DOP_IMPORT_PYR
filecache_node = dop_geo_node.node('filecache2')

# Set the file path for the filecache2 node
filecache_path = 'path/to/save/cache.bgeo'

# Save the filecache2 node to disk
filecache_node.parm('execute').pressButton()
```

Because cache files require a lot of time for calculation. After my preview render resolution, I will batch render multiple cache nodes with the **shelf Python tool**. It is important only to follow the proper sequence.

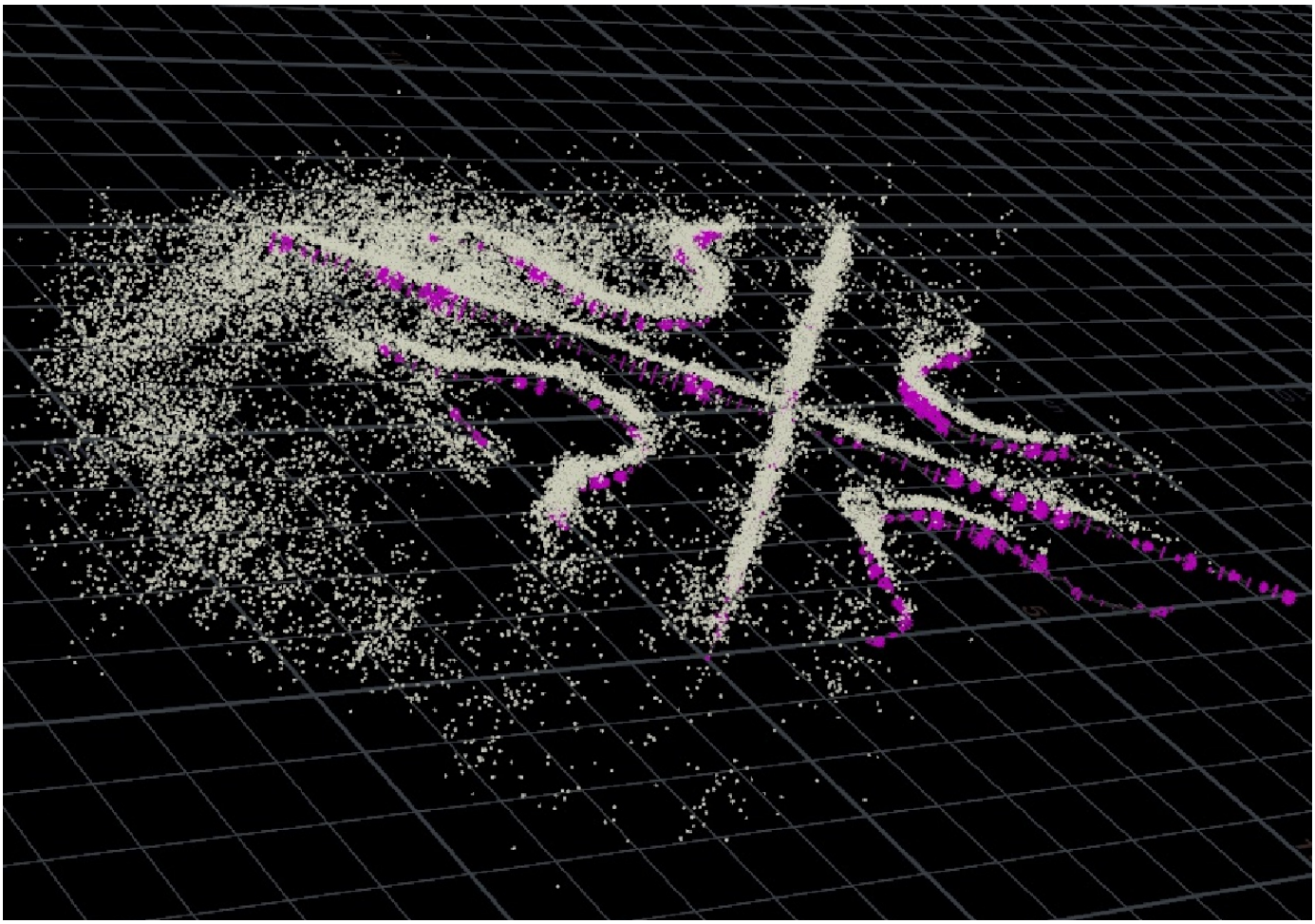


After a lot of tweaks in solvers, here is a simple visualization of our Chladni waves. It's worth mentioning that this dynamic network can be fed with any shape. On a higher level, it could convert any high-contrast image or sequence into a

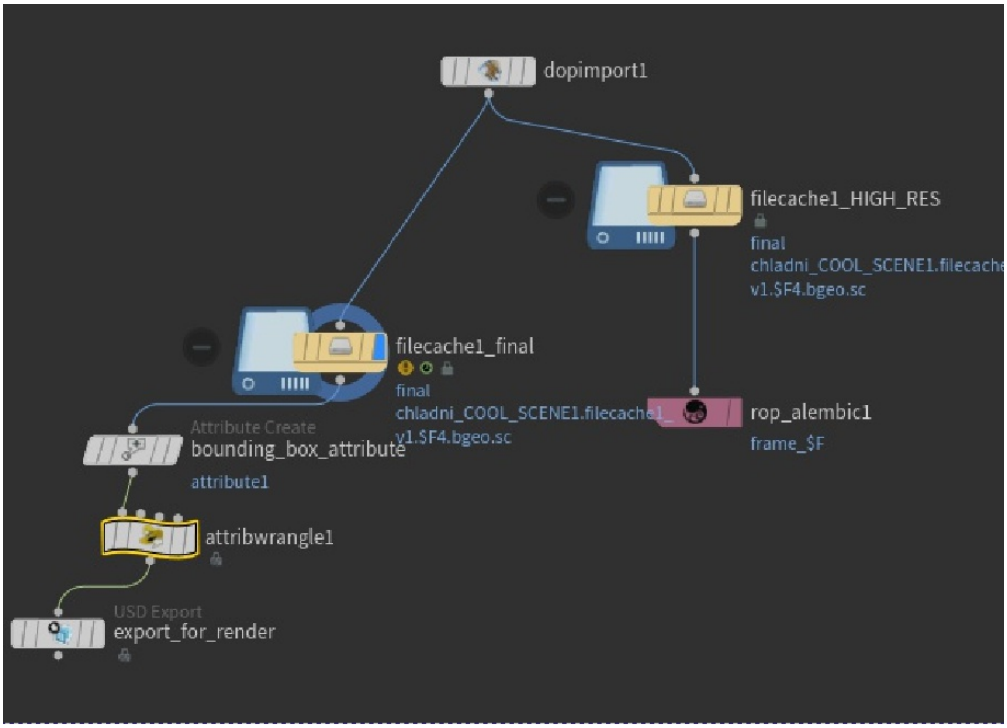
VEXpression

```
1 if (@P.x < -@attribute1 || @P.x > @attribute1 ||
2     @P.y < -@attribute1 || @P.y > @attribute1 ||
3     @P.z < -@attribute1 || @P.z > @attribute1)
4 {
5     removepoint(0, @ptnum);
6 }
```

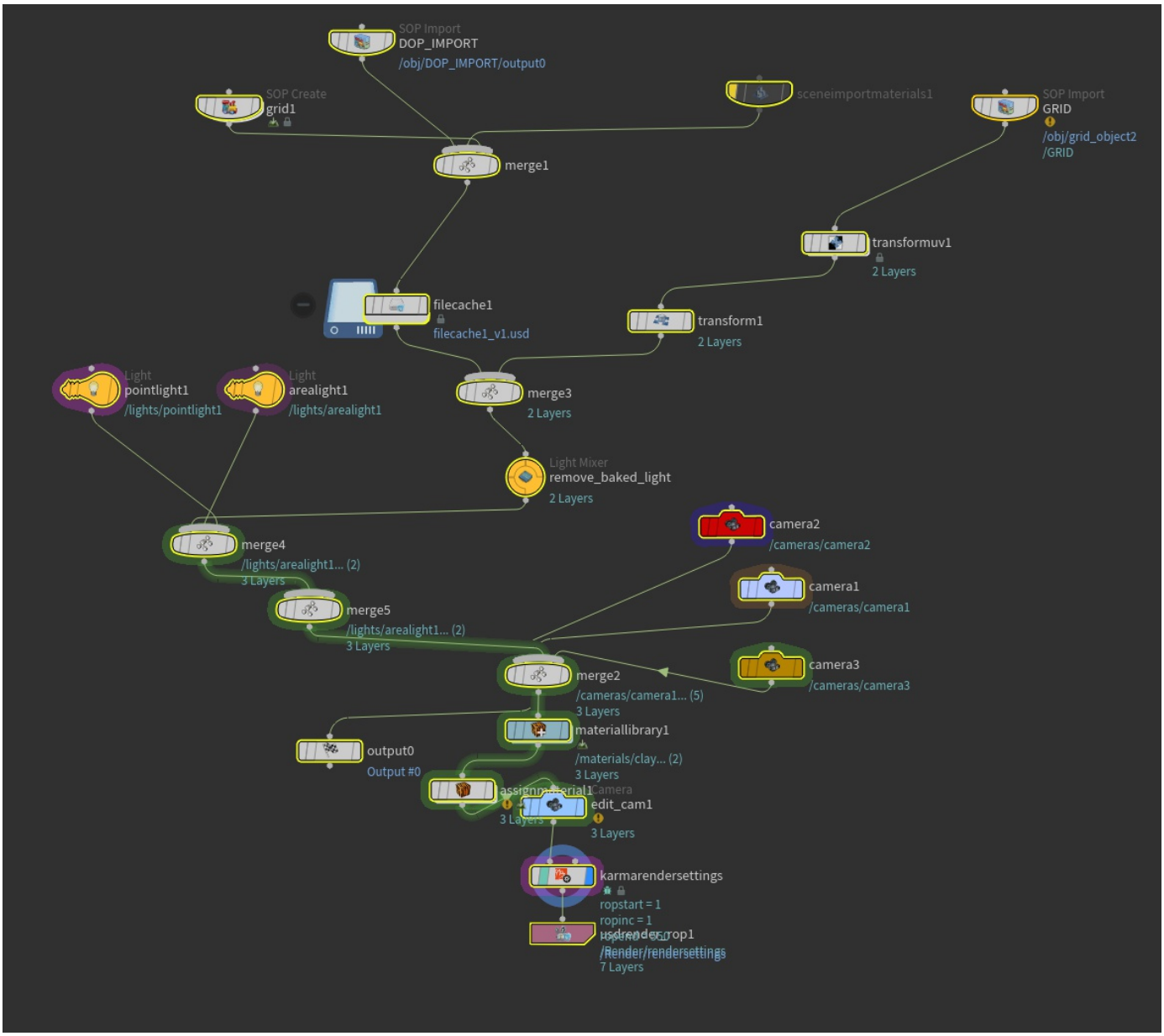
The upper visualization image is trimmed with this code for performance purposes. Due to the high RBD count, it is highly recommended to do so.



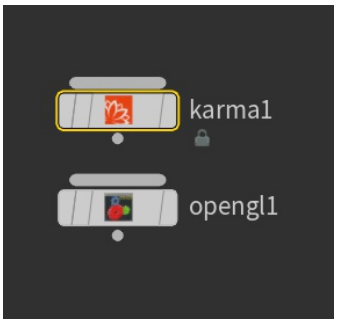
Here is example of DOP network in action.



Here is an example of a GEO network that contains a USD cache with stripped-down geometry for export.



For rendering was used Solaris and Karma render engine. Karma render engine enables to be used GPU rendering, which is faster. Here is also set cameras and lighting. For importing the objects is used SOP imoprt nodes. There aren't any special materials used for rendering. Basically principaled shader with textures. Different Cameras represent different perspectives.



One last network is ***"out".*** There is OpenGL which is used for viwing the scene before rendering it in Karma.