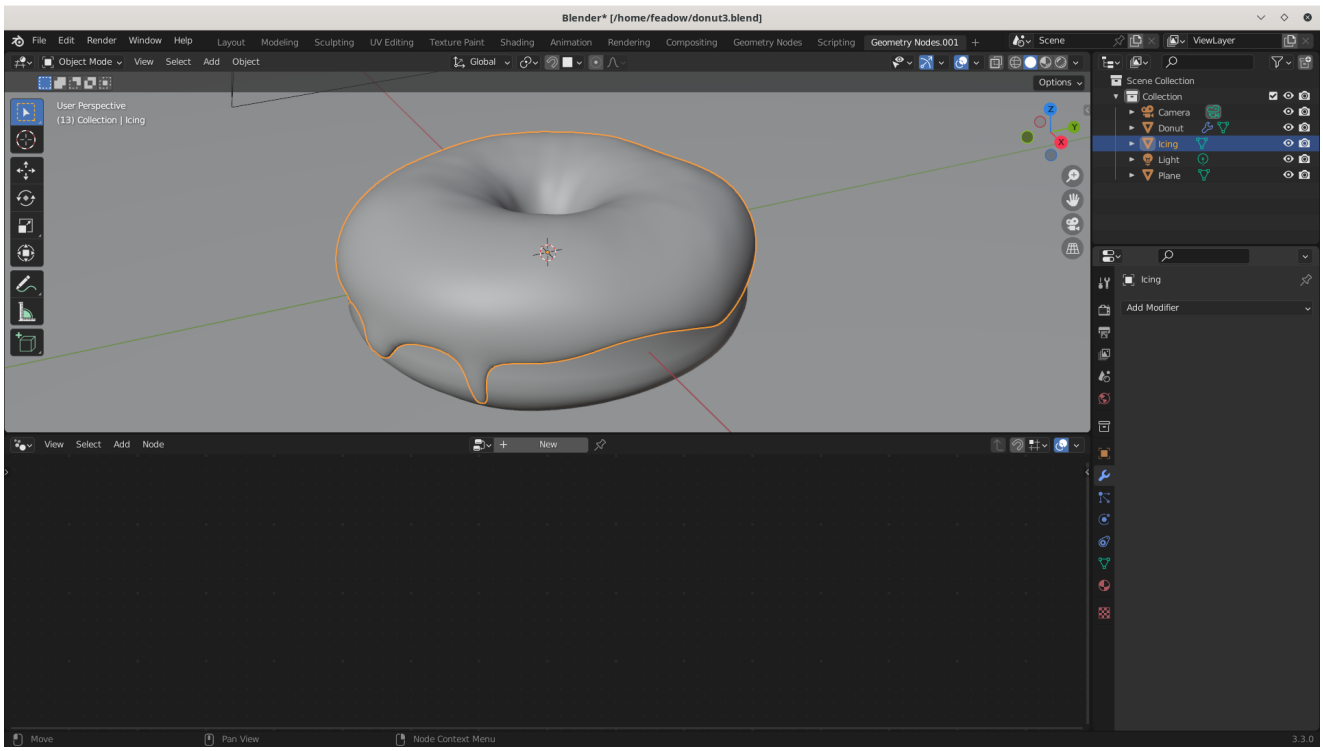
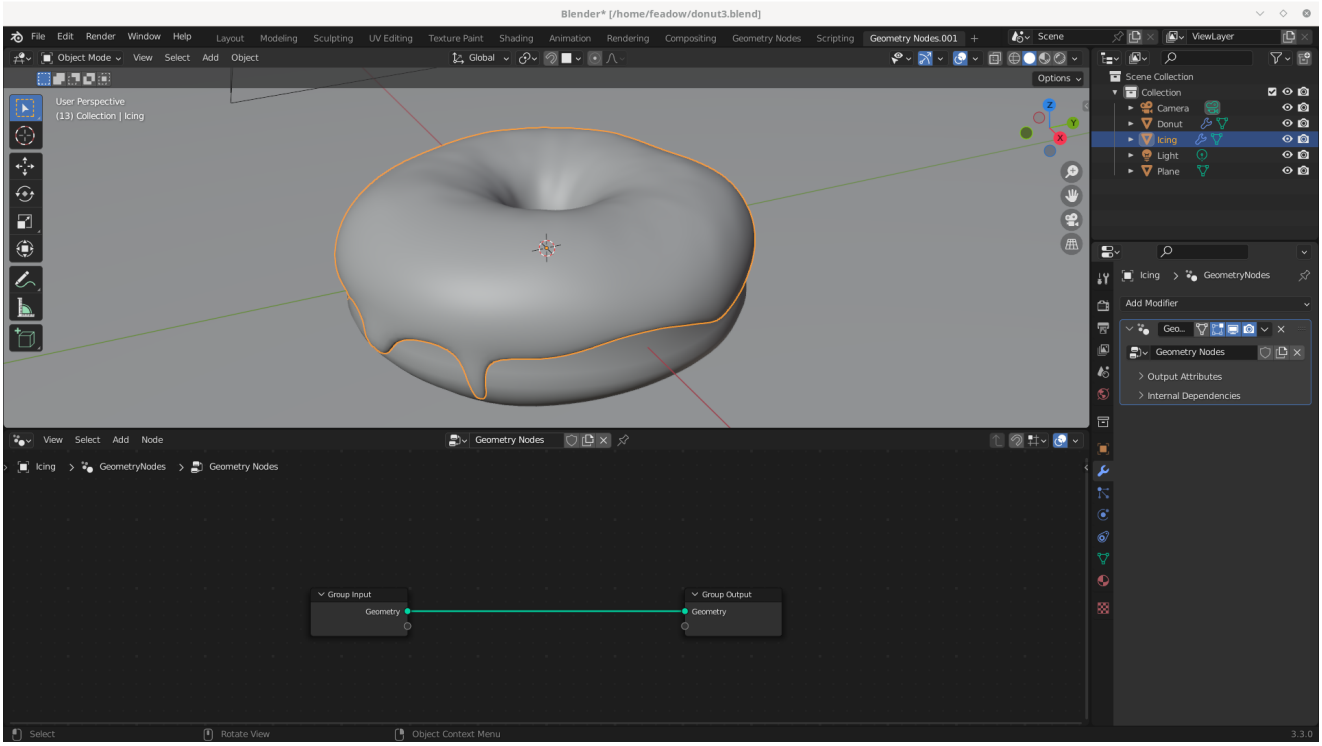
Geometry Nodes in Blender

Geometry nodes allow sophisticated manipulation of a mesh's geometry in Blender, the same way the node editor (see D#yb497j) allows manipulating surfaces, colours and textures.

With geometry nodes, the possibilities are endless.

In Blender, geometry nodes are implemented as a **modifier** (read about modifiers: D#d73iqq). By setting up a geometry nodes workspace and selecting the desired object and hitting 'New', a new modifier is applied to the selected object that "implements" the changes performed using the geometry nodes.





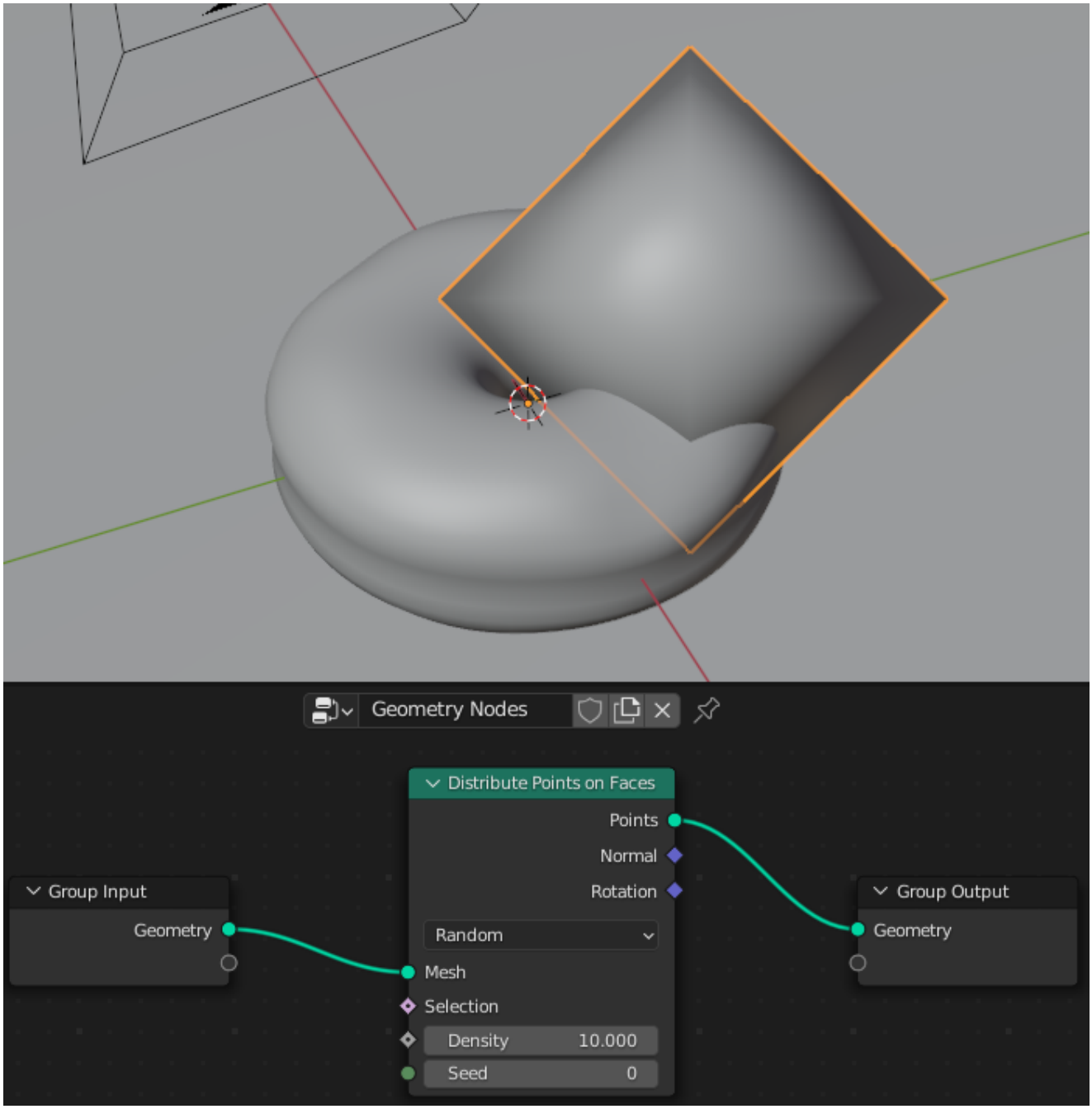
As one can see, 2 "nodes", connected with a green line, appear in the view immediately after creating the modifier for an object. These 2 nodes are the "input" and "output" of the geometry nodes editor. Any changes to the mesh will be done through various nodes that will live between the input and output nodes. At the moment, the input is directly connected to the output, which simply means that the shape "after modification" should be the same as shape "before modification" and that the modifier in this case is doing basically nothing.

Now let's see what some of the nodes can do..

Generating Points on a Surface

The **Distribute Points on Faces** node essentially takes the object and transforms it into a bunch of points (that live on top of the original object)..

> The *Distribute Points on Faces* node places points on the surface of the input geometry object. [1]



Here we have one "particle", but could add more by increasing the 'Density' pointer..

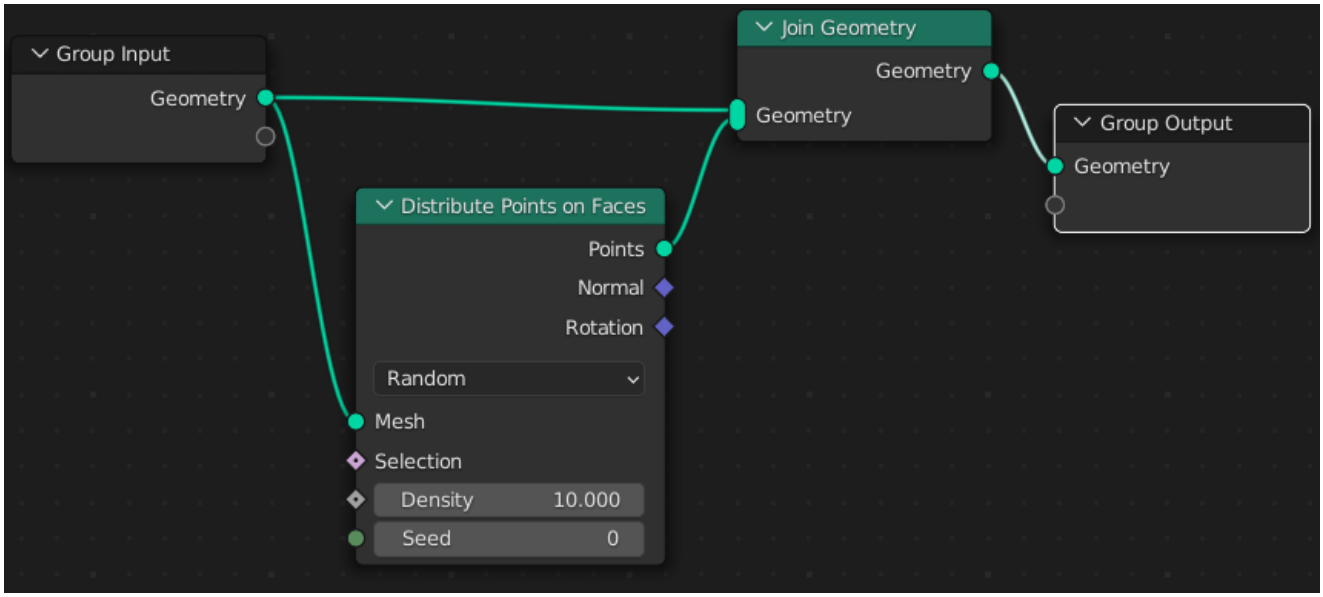
Combining Geometries

Notice how, in the previous section, the "icing" has completely disappeared. i.e. the icing *became* the points that now live "on top of the surface" of the original icing object. Could we do something that makes the icing remain in place.

In other words, we want the points *AND* the original geometry. [2]

The **Join Geometry** node does just that.

No need to elaborate further, here's how to use 'Join Geometry' to implement "the fix":



The "icing" object will come back to the scene.

Cloning "Particles" on a Generated Set of Points

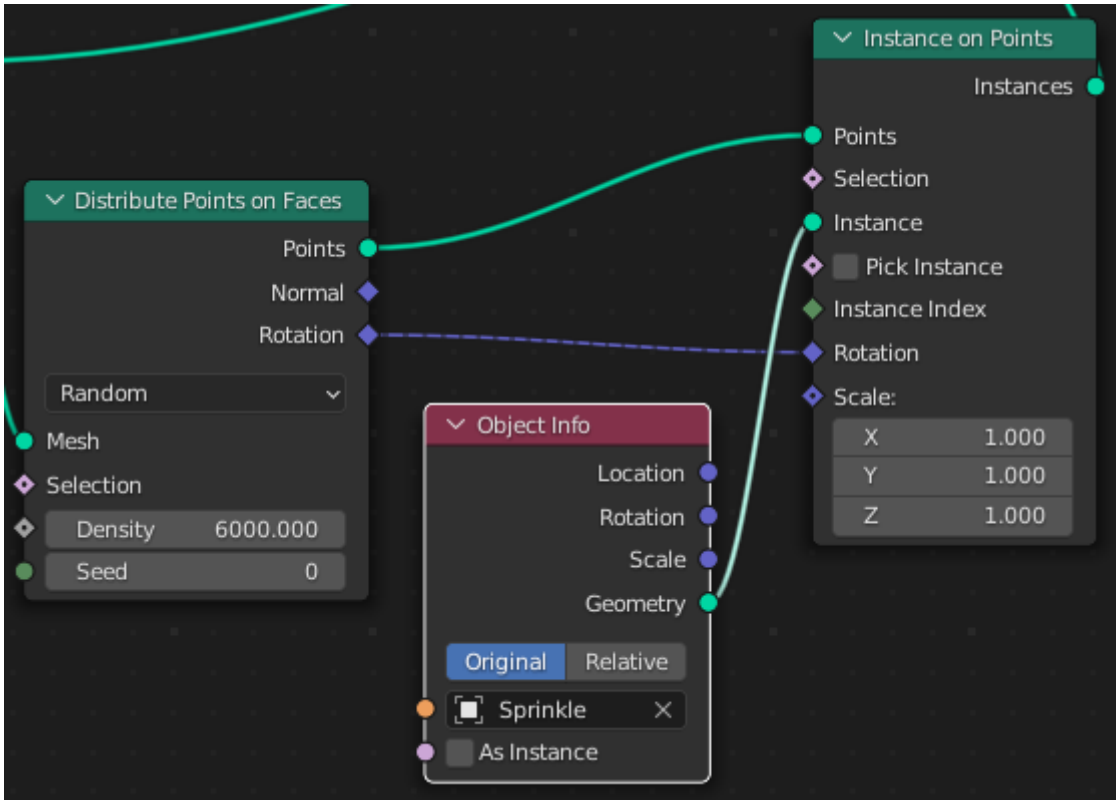
Now we need to make the "points" that were generated on top of the icing in the previous step look how we want them to. To do that, we can create a reference object and use **Instances on Points** node to make that reference object "clone" to wherever one of these points exist.

For example, we can create a reference cylinder object with miniscule dimensions that models a "sprinkle". We then drag the listing of our "sprinkle cylinder" from the 'Scene Collection' menu on the top right to the geometry nodes editor to get an 'Object Info' node. This 'Object Info' node can then be connected to the 'Instance' field of the 'Instance on Points' node to use it as a "particle" for the object to be placed at the generated points.

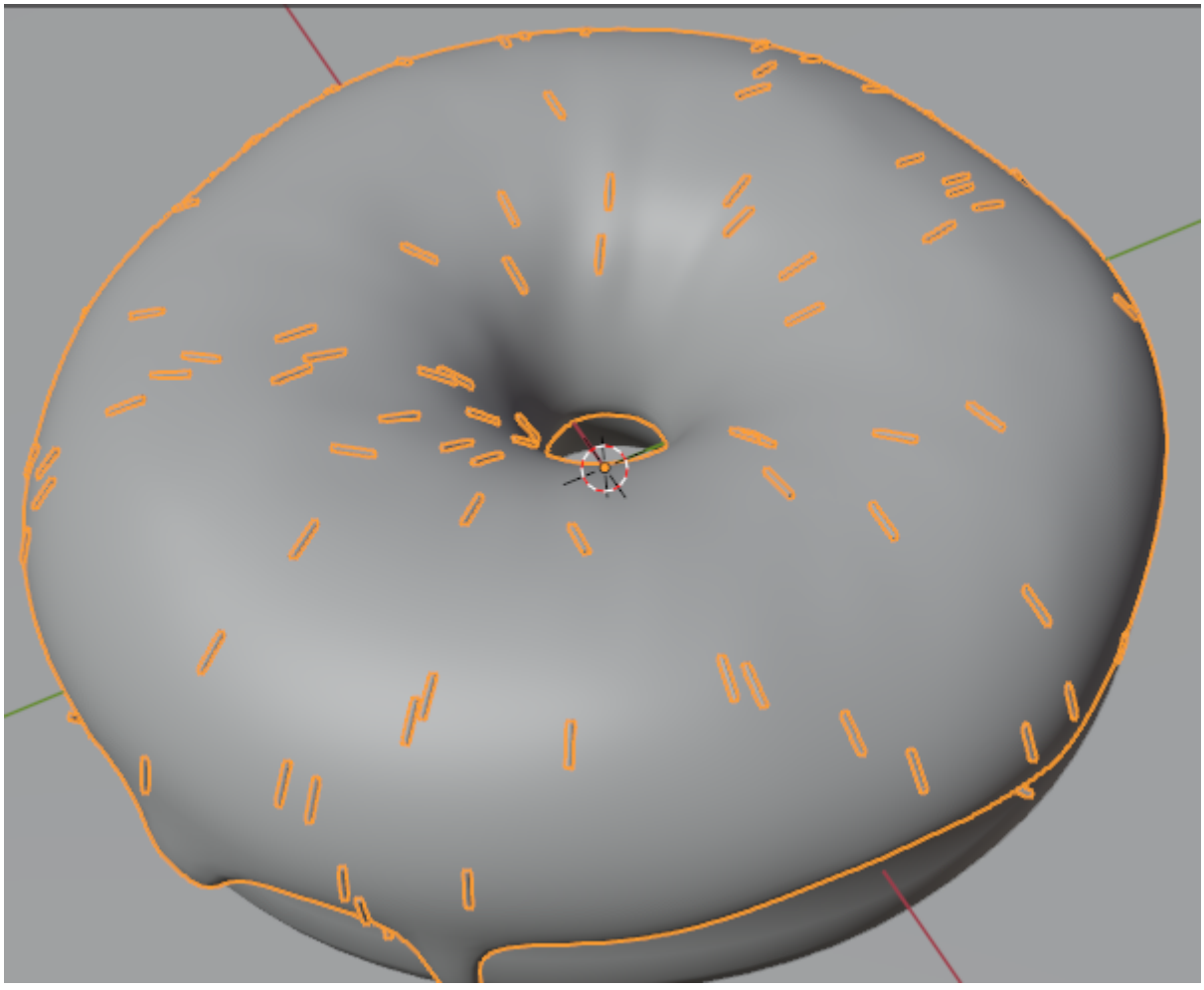
> Remember to "apply the scale" if you rescale your "particle object"!

Additionally, we could connect the 'Rotation' field from the 'Distribute Points on Faces' node to the 'Rotation' field from 'Instance on Points' node to make the "sprinkle particles" follow the rotation of the surface!

Setting everything up:



We get a promising result:



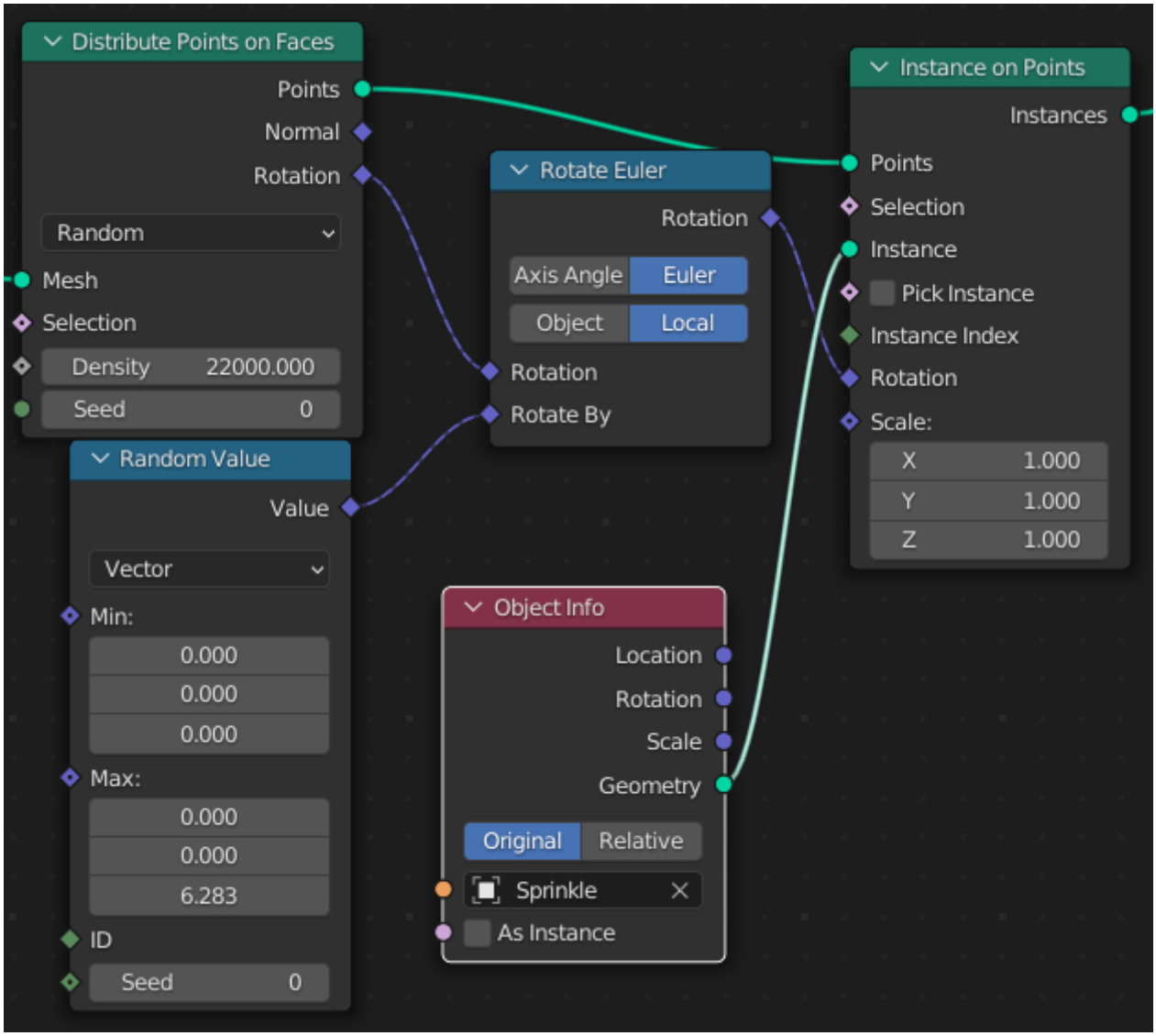
Rotating

In the previous section, we managed to make the "sprinkles" become "flat on the surface" by linking the rotation of the generated points on the surface with the rotation of the "sprinkle".

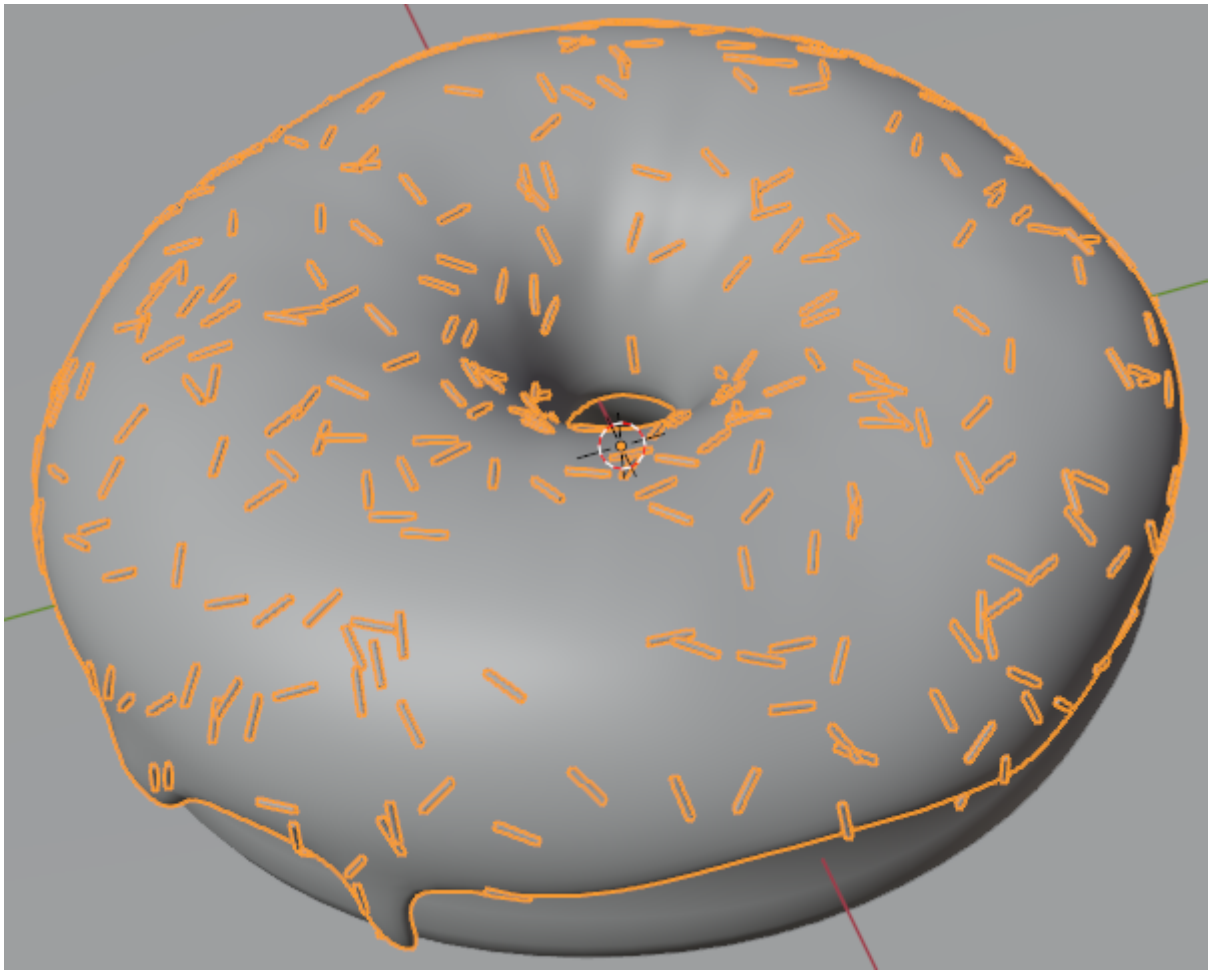
But notice how all the sprinkles are pointing inwards, that's not what we would normally want. Ideally, each of the sprinkles would have a random rotation, while still laying flat on top of the icing.

This is where the **Rotate Euler** node comes into play!

By applying the 'Rotate Euler' node in the fashion shown in the image below, we can effectively induce the desired randomness in the alignment of the sprinkles.



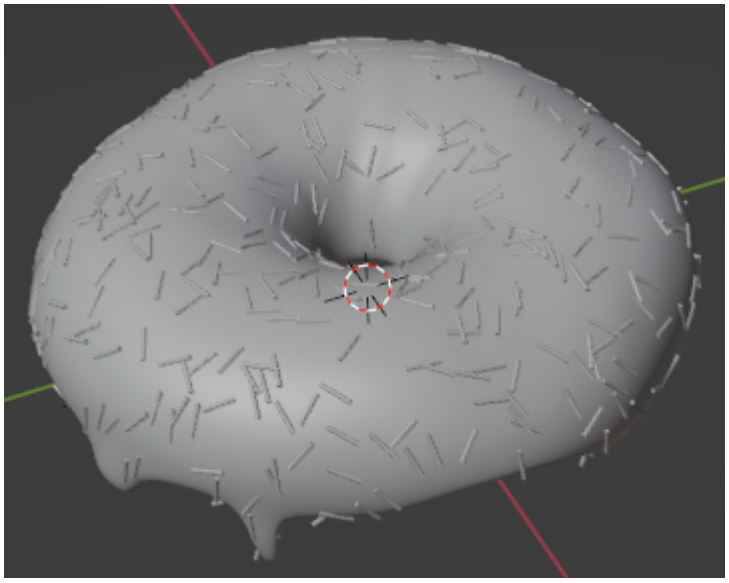
We get:



> Note how we used the **Random Value** node to generate a rotation vector with a random z-component!

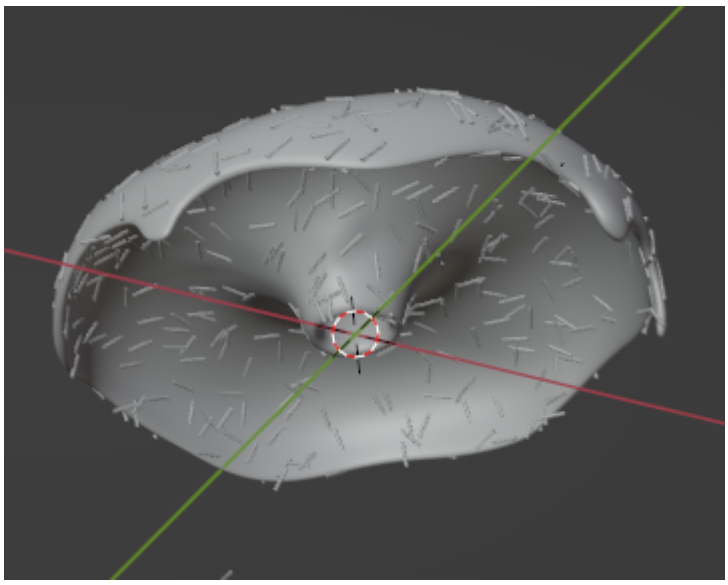
"Exposing" Fields and doing Math

Now we have the following "donut icing" object:



Notice how the "sprinkles" - which are implemented using geometry nodes (are distributed in a weird way: there are sprinkles at the "droopy bits" of the icing, where one doesn't expect them to be there.

There are even sprinkles attached to the "lower side" of the icing:

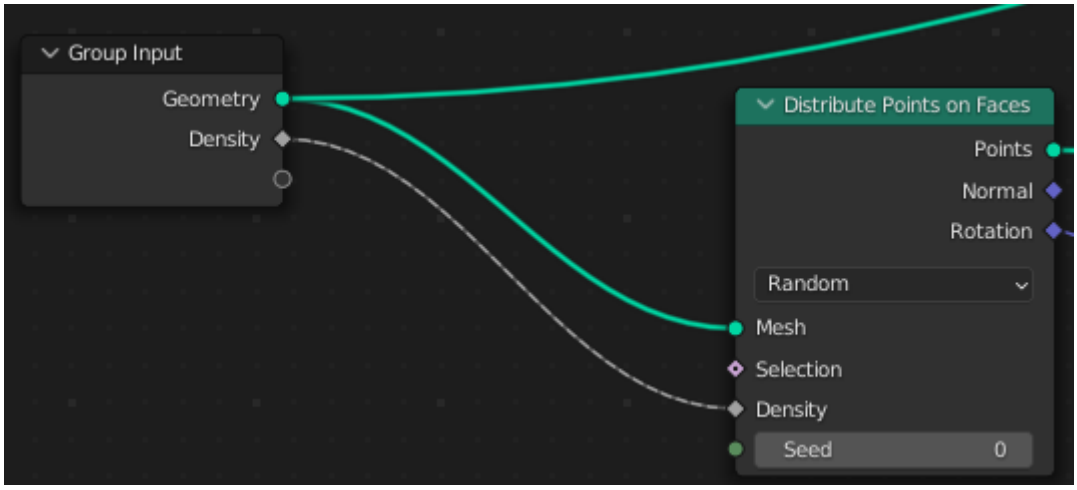


..that doesn't make much sense.

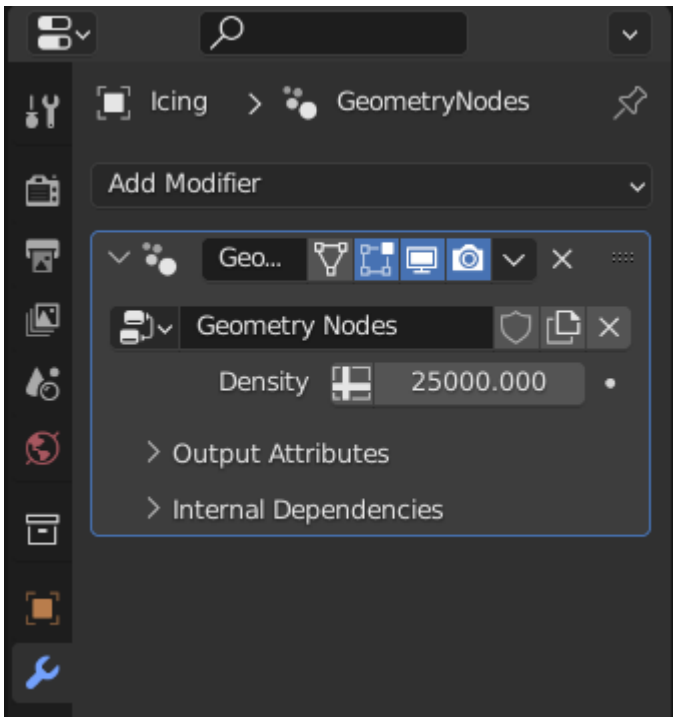
One way to fix this is using weight painting. By treating the icing mesh as a vertex group, one can create a "weight map" that controls the distribution of sprinkles on the icing's surface (see D#5muz63).

Once we have the **vertex group generated by the weight painting**, we can assign the vertex group as a "density function" for our sprinkles.

We do so by "exposing the density field" by linking it to the input node, as such:



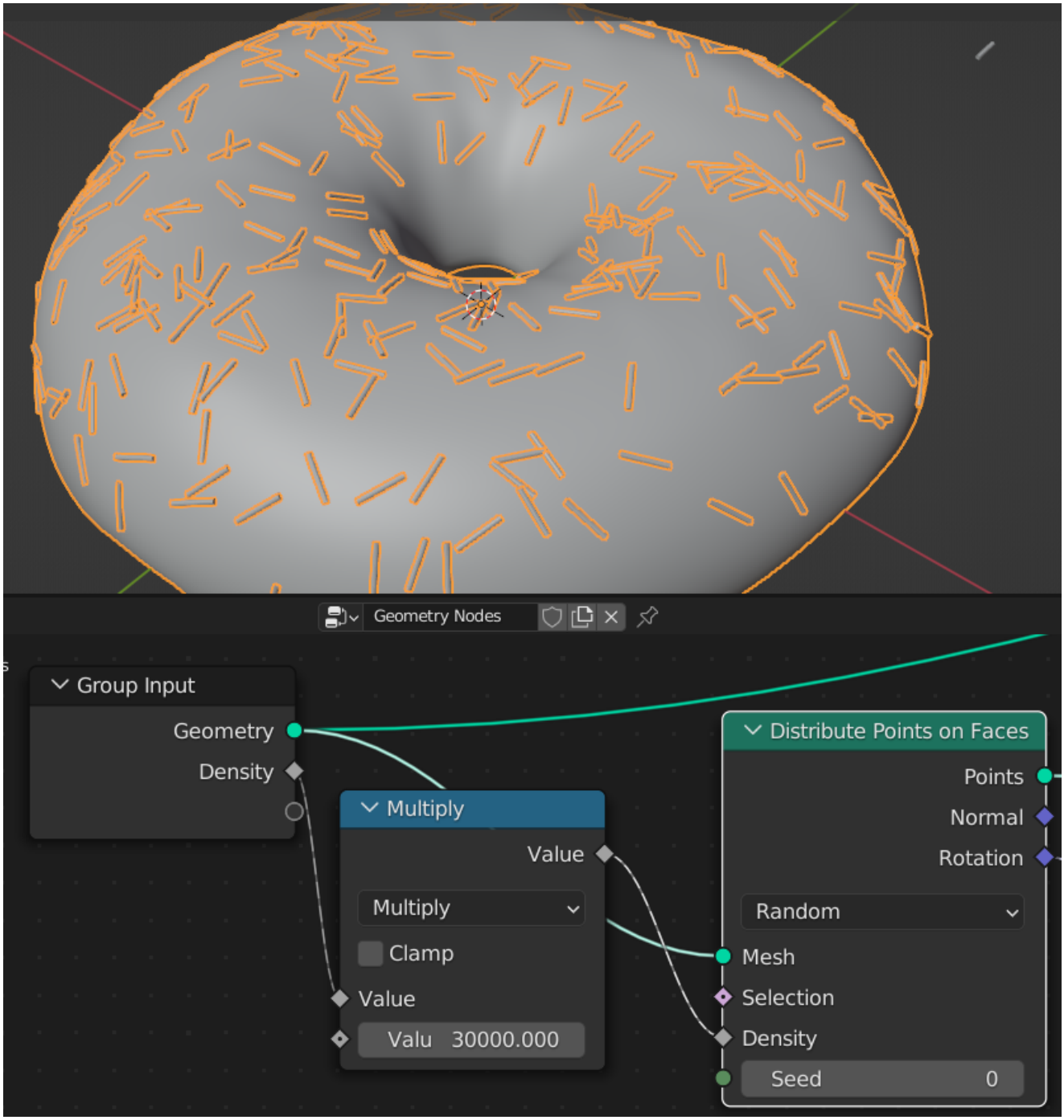
Now the 'Density' field can be controlled from the modifier stack:



Now, by clicking the "+"-shaped sign in the 'Density' field, we can assign the value to be our vertex group's name (the vertex group created by weight painting).

The donut will now appear without any sprinkles, the reason for this is that the density function has a range of 0 to 1. While previously, our salient values for the 'Density' were in the tens of thousand!

To solve this problem, we can simply use a **Math** node to multiply the "density link" by the suitable factor, as such:



Aha!.. now the donut's got some proper sprinklin' !

> We can also "expose" the "factor" of the "multiplication node" in order to control the "density amplitude" from the modifiers menu.

Note that this method introduces "clipping" between the sprinkles, to (partially) fix that, one could select the 'Poisson Disk' option in the 'Distribute Points on Faces' node, as explained in [prs].

Bibliography

[prs]: Beginner Donut Blender Tutorials, Blender Guru 2021 - part 9
[s1]: Blender 3.3 Manual

---

1. from [s1]↩
2. phrasing from [prs]↩