Chan Hiu Yan & Tsang Tsz Wai
com bin ati on, 2019

3D-printed PLA sculpture, coded with HE_Mesh library on Processing
100mm x 110mm x 170mm

The Artwork

The tree-like sculpture is generated with an L-System that models the mathematics of calculating the number of possible combinations. This specific piece represents the number pattern of the possible combinations of 8 out of 10. The algorithm utilizes a binary-based addition, rather than the conventional factorial formula, a path that seems senseless from an efficiency point of view. However, as one gazes at the aesthetics of the sculpture, the intriguing fractal patterns of the number series, one discovers the endless possibilities of the unexplored realm of inefficient mathematics.

Video Documentation:
https://vimeo.com/334659564

The Concept

How many possible combinations of 3 people can a group of 10 people form? I'm sure you've solved similar problems in secondary math class. First, you would multiply 10x9x8 to solve for the possible arrangements. Then you would divide it by 3! because there would be 3! number of different order of the same 3 people. But this has always fascinated me that in order to solve a combination problem, we have to go through factorials. Factorials are convenient for calculating permutations where order matters, but for combinations order doesn't matter. Can there be another algorithm for solving combinations besides factorials?

As a nerd in middle school, I drew out what I thought was the more intuitive way of solving this problem. Addition.

Take a group of 5 people: A, B, C, D, and E. The combination of 1 person is obviously 5. The combinations of 2 people are A[AB, AC, AD, AE] (4) plus B[BC, BD, BE] (3) plus C[CD, CE] (2) plus D[DE] (1). Thus 4+3+2+1 = 10. Terrifically simple.

Or so I thought as I move on to combinations of 3. A{ B[ABC, ABD, ABE] (3) C[ACD, ACE] (2) D[ADE] (1) } plus B{ C[BCD, BCE] (2) D[BDE] (1) } plus C{ D[CDE] (1) }. Thus 3+2+1+2+1+1 = 10.

As you might imagine, the string of addition gets more confusing when calculating combinations of large numbers. In fact, it recurs itself like a fractal. You know what, I should name this as the Chan Fractal.

Combinations in a group of 8 people (n = 8)

# of people in a combination k	0	1	2	3	4	5	6	7	8
# of possible combinations C(n,k)	1	8	28	56	70	56	28	8	1
Algorithm	1	8	7654321	654321 54321 4321 321 21 1	54321 4321 321 21 1 4321 321 21 1 321 21 1 21 1 1	4321 321 21 1 321 21 1 21 1 1 321 21 1 21 1 1 21 1 1 1	321 21 1 21 1 1 21 1 1 1 21 1 1 1 1	21 1 1 1 1 1	1

Okay they were right. This might not be the most efficient way to calculate combinations. But certainly a fascinating pattern to explore, right? Let’s make some interesting observations. Take a positive integer *n* as the total number of people. Let *k* be the number of people in a combination. Let *C(n,k)* be the number of possible combinations.

$$C(n,k) = \frac{n!}{(n-k)!k!}$$

In the second row, we can see that from *k* = 0 to *k* = *n*, the value of *C* increases and decreases symmetrically. Also we can note that across the algorithm row from *k* = 1 to *k* = *n*, the first number decreases by 1 from *n*. One way to represent this pattern from *k* = 2 to *k* = *n*, is that you have a starting axiom of 7654321. The next generation, you start from 1 less than the first number, in this case 6. You search for the last time you see 6, and copy and paste from there to the last digit. Then you search for the last time you see 5, and copy and paste from there to the last digit. So forth until 1.

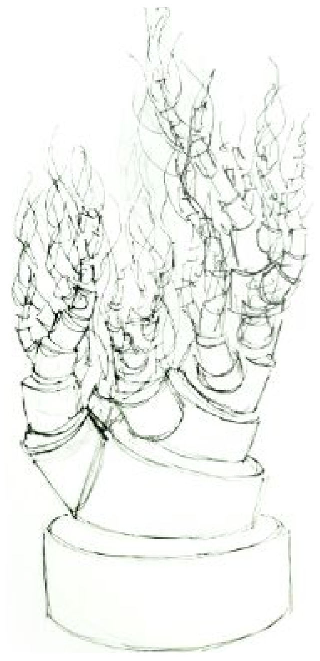
Another way to model this is with a recursive function. The function takes two parameters (*n,k*) and returns a list of digits. But in order to produce this list, it calls itself with the parameters (*n, k*-1) and so forth. This function, unlike ones found readily on the web, doesn't return the answer *C*, but rather, returns the list of digits that can be used to create interesting fractals, I hope.

Since this algorithm is very inefficient and perhaps counterintuitive, I was not able to find existing ones with this approach. This opens up an unventured region in mathematics where we can explore the beauty of taking the longer route. Counter-optimization. The possibilities from possibilities. The aesthetic embodiment of a mathematical concept taken over-board by a middle-schooler. In a sense, meaningless in our world today if we only value efficiency and problem-solving. But are there not many routes to the same destination? The length of the route does not determine its validity. In fact, to think that life must be completely efficient is unrealistic. Randomness is breathing in and out of our lives every minute. A noise, a curve, a bump - life is living in the a pool of possibilities that runs free from human will. And in these aberrations one finds adventure and beauty.

The Process

1. *Making a plan*

We planned to make a 3D-printed fractal sculpture created from a recursive algorithm for calculating combinations. The form will resemble a twisted tower, with each section of the tower extending another tower recursively. Optionally, we may add noise or other aesthetic elements like blossoms to it.

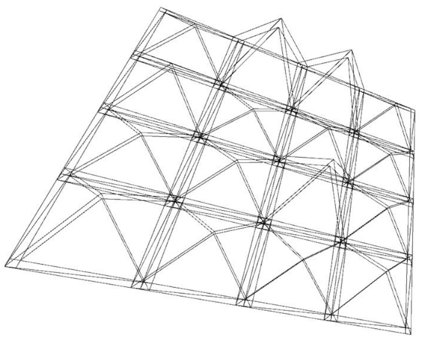


The original idea was to use cylinders as the basic building blocks. The size of the cylinders will be controlled by the value of the numbers in the pattern, while the location of the cylinders will be based on the "generation" of the number.

2. Finding a Processing library for 3D manipulation

In order to build the 3D mesh, we need a coding-based program that provides a library of already-made meshes for ease of use. For familiarity's sake, we chose Processing, and searched for an external library.

Our first find was an Instructables project using the Modelbuilder library.

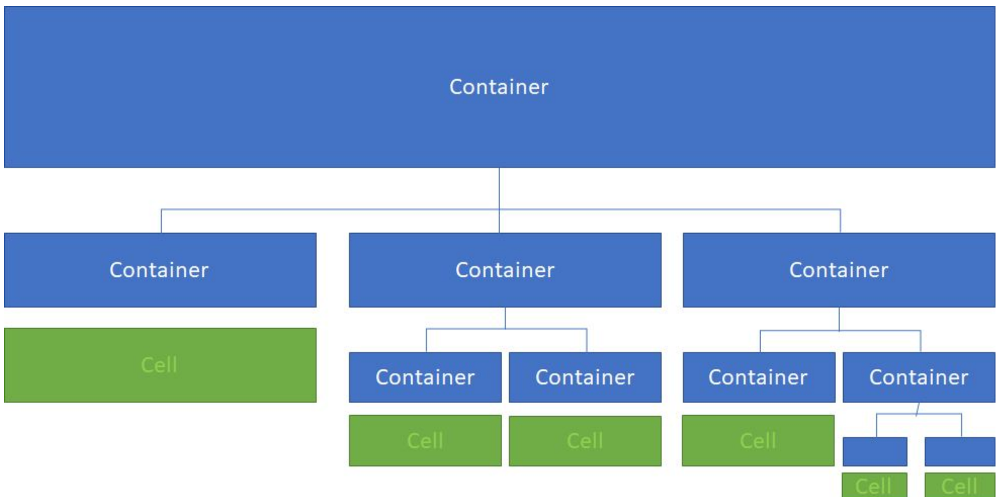


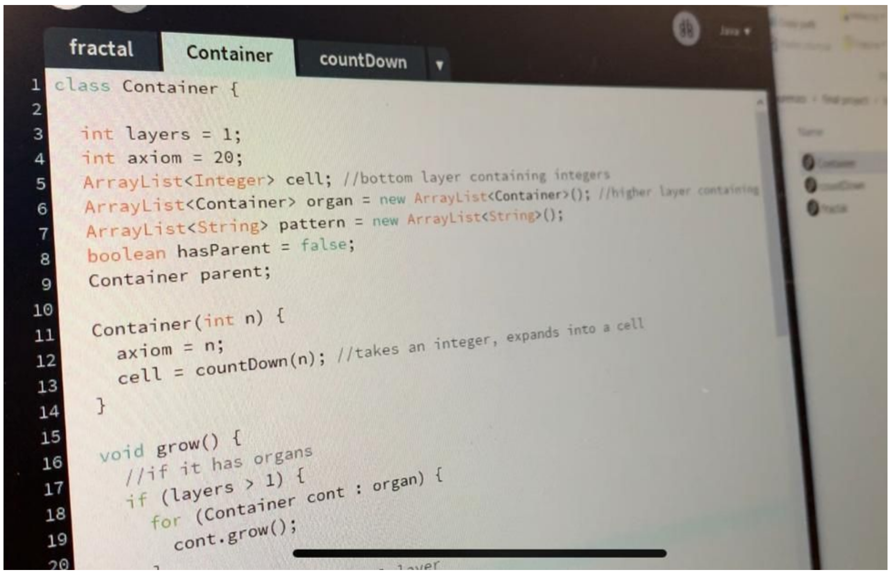
But it was not very helpful, as the mesh was created by iterating the coordinates of each point.

After a lot of googling, we found the HE_Mesh library on Github. It includes numerous pre-defined 3D forms and modifying functions. Perfect. A complete resource of examples to start the journey of self-learning.

3. Structuring the system (2D testing)

Now that we know we have a good library of the 3D stuff, it is time to structure our algorithm system. At first, I created a class named Container (very descriptive name, I know), which contains an arraylist of integers and an arraylist of other Containers. In this way, the system forms a tree structure.





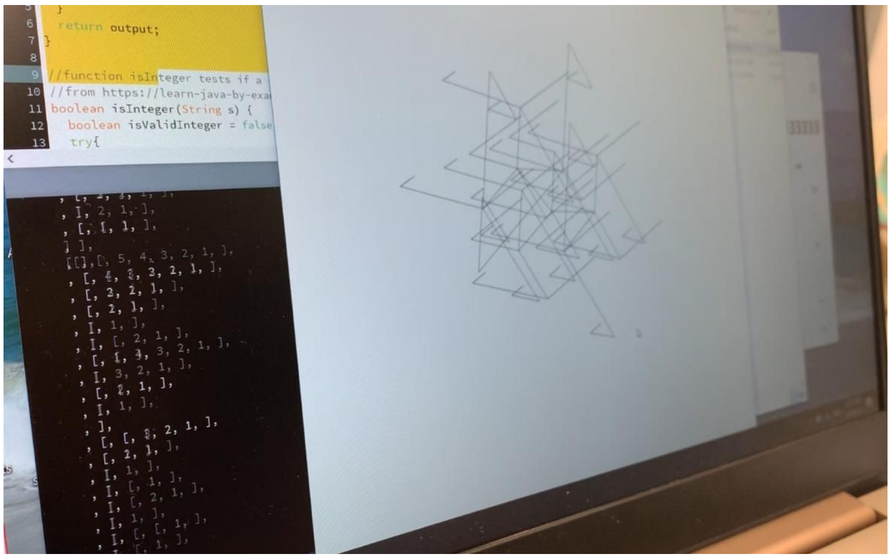
```
class Container {

  int layers = 1;
  int axiom = 20;
  ArrayList<Integer> cell; //bottom layer containing integers
  ArrayList<Container> organ = new ArrayList<Container>(); //higher layer containing
  ArrayList<String> pattern = new ArrayList<String>();
  boolean hasParent = false;
  Container parent;

  Container(int n) {
    axiom = n;
    cell = countDown(n); //takes an integer, expands into a cell
  }

  void grow() {
    //if it has organs
    if (layers > 1) {
      for (Container cont : organ) {
        cont.grow();
```

Each time the Container grows, a new generation of Containers is born. The basic function that governs over this process is the countDown(). It takes an input of an integer and returns an arraylist of integers.
5 → [5,4,3,2,1]



```
  }
  return output;
}

//function isInteger tests if a
//from https://learn-java-by-exa
boolean isInteger(String s) {
  boolean isValidInteger = false
  try{
```

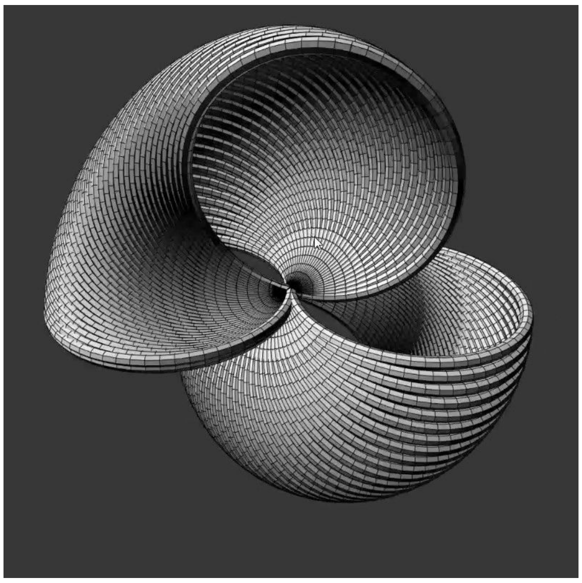
```
 , I, 2, 1, ],
 , [, I, 1, ],
 ] ],
 [[],[, 5, 4, 3, 2, 1, ],
  , [, 4, 3, 3, 2, 1, ],
  , [, 3, 2, 1, ],
  , [, 2, 1, ],
  , I, 1, ],
  , I, [, 2, 1, ],
  , [, I, 4, 3, 2, 1, ],
  , I, 3, 2, 1, ],
  , [, 2, 1, ],
  , I, 1, ],
  , ],
  , [, [, 3, 2, 1, ],
  , [, 2, 1, ],
  , I, 1, ],
  , I, [, 1, ],
  , I, [, 2, 1, ],
  , I, 1, ],
  , I, [, [, 1, ],
  , I, [, 1, ],
```

We tested the system with some 2D line drawings.

4. Messing with the meshes

Now that we have a somewhat working system, it’s time to master that new HE_Mesh library we found. It took a whole lot of looking through the examples folder to see what it is capable of.

To figure out how to draw meshes at different positions, I tried using translate() and rotate(), and rendering the mesh at (0,0,0).


Here’s one with many tori rotated. Isn’t it beautiful? Too bad it’s not a fractal.

5. Combining the system with the meshes

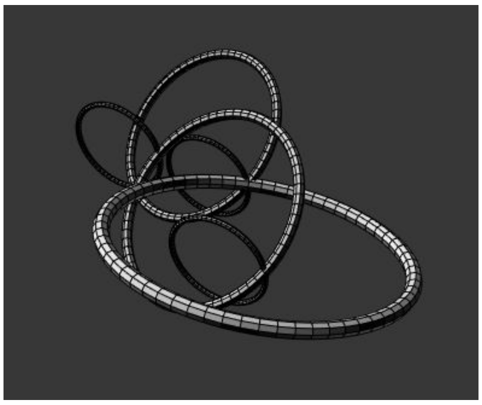
With both system and HE_Mesh library ready, let’s combine them. The system generates a Container with a pattern like this:

```
[
[
[54321]
[4321]
[321]
[21]
[1]
]
[
[4321]
[321]
[21]
[1]
]
[
[321]
[21]
[1]
]
[

[21]
[1]
]
[
[1]
]
]
```

Another class takes in this pattern and analyzes it. Every time it sees a number it draws a torus with the appropriate size. Everytime it sees an open bracket, it pushMatrix() and applies a transformation. The close bracket will popMatrix().

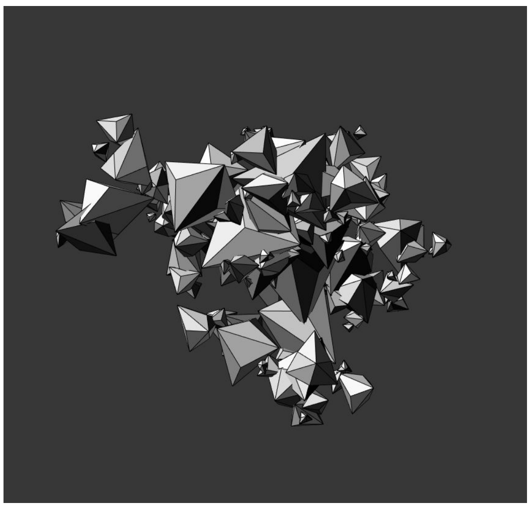
6. Fail? Change the system



The problem is that the meshes are, well, not very appealing. And it is awfully slow on my computer because each time the program runs draw() it transforms, creates, and render multiple times.

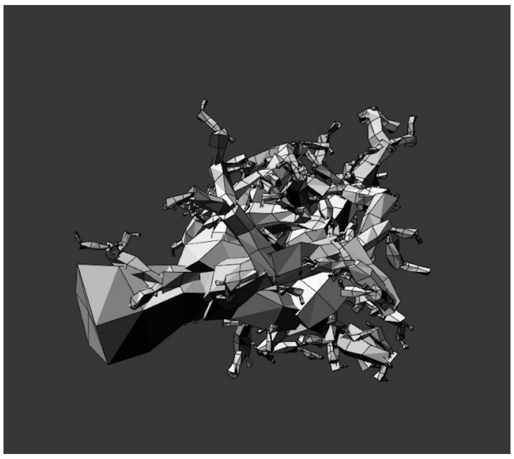
With many tries and changing the variables, the outcome is still unappealing. It seemed the problem lies in the structure and the location of the brackets. To reform my system, the Container-within-Container system was rejected and a new one was implemented, working much like an L-system. The function countDown() also changed to this:
5 → 5 [4, 3, 2, 1]

The grow() function also changed so that only the integers within the innermost brackets will perform countDown. In the above case, 5 will not expand anymore. Strictly speaking, the pattern is no longer exactly the number sequence in the combination in the Chan Fractal, but for the sake of beauty, something must be sacrificed.

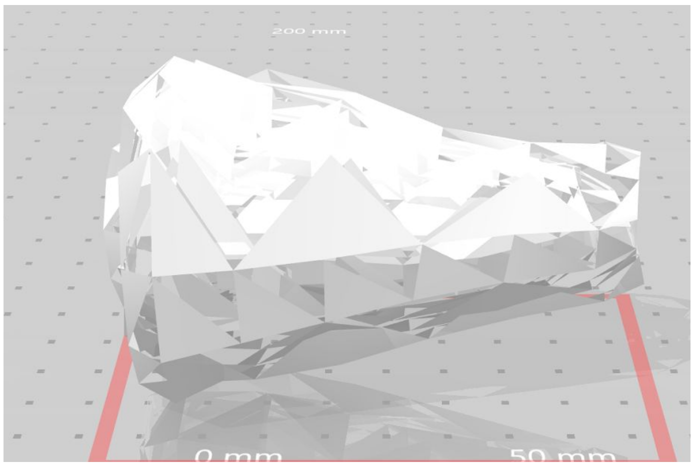


Different basic 3D forms were explored, as well as modifiers like noise and smooth.

7. *Export!*



Great! Let's save to STD file.



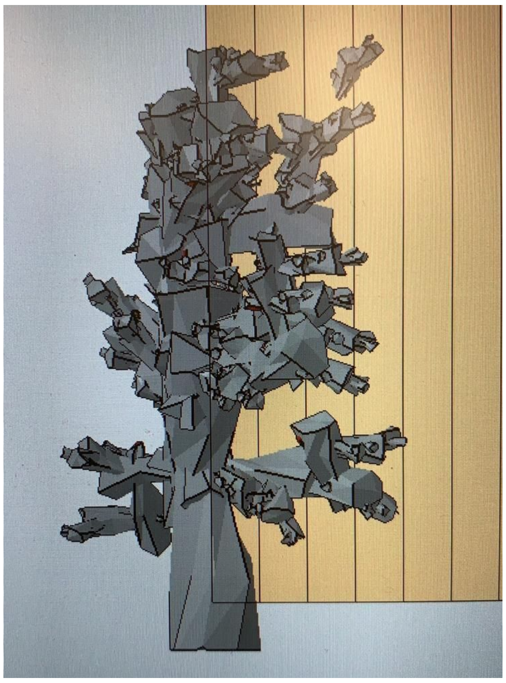
What in the world? The problem lies in the transformation. Everything is located at (0,0,0) when exported.

8. *Fail? Change the system again*

So it's time to restructure the system again. Now we are not using transformations but vectors to control the position of the meshes.

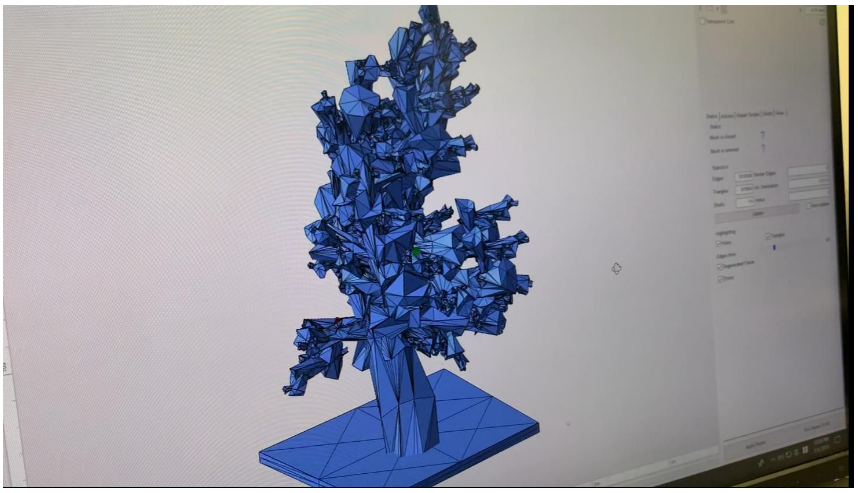
9. *Export!*

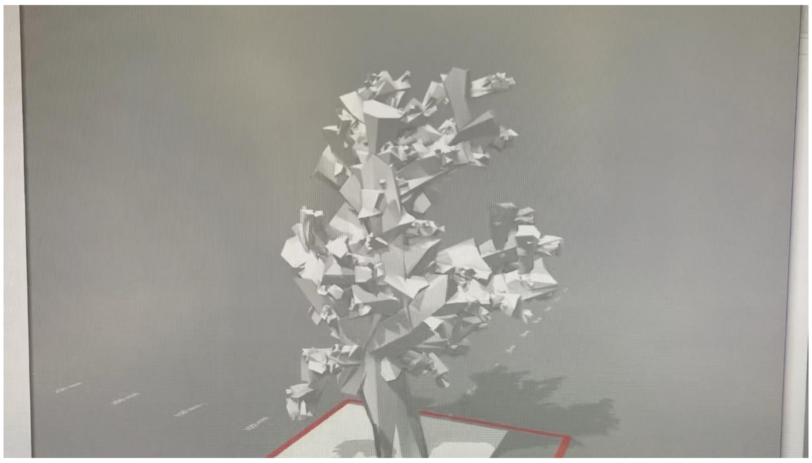
Export to STL file



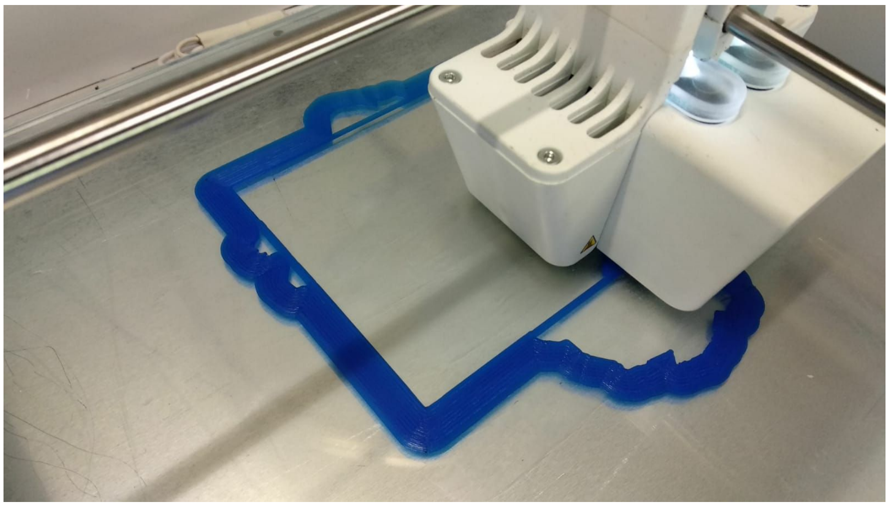
10. Print it!

To materialize the mesh, a 3d printer is needed to achieve the goal. Therefore, we went for the 3d-printing service that provided by the university, which is the GE Lab (P4801). The area provides several equipment for students to enjoy the experience related to 3d modeling, like 3D printers and 3D scanners.



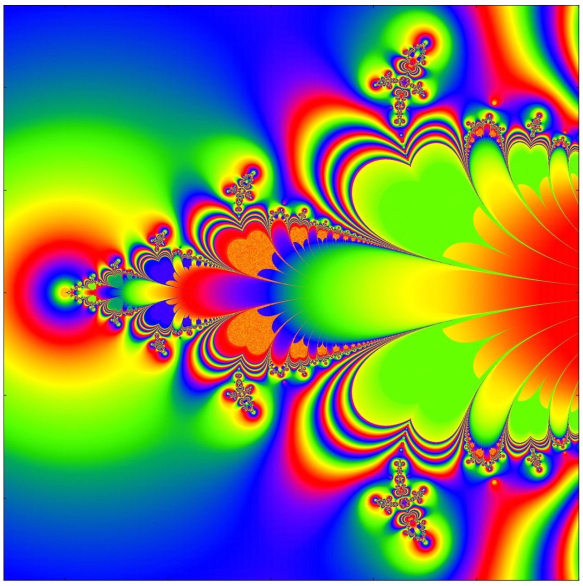


Before we started printing the mesh, slicing the 3d model in Ultimaker Cura, a software for 3d model preview and printing, was necessary.



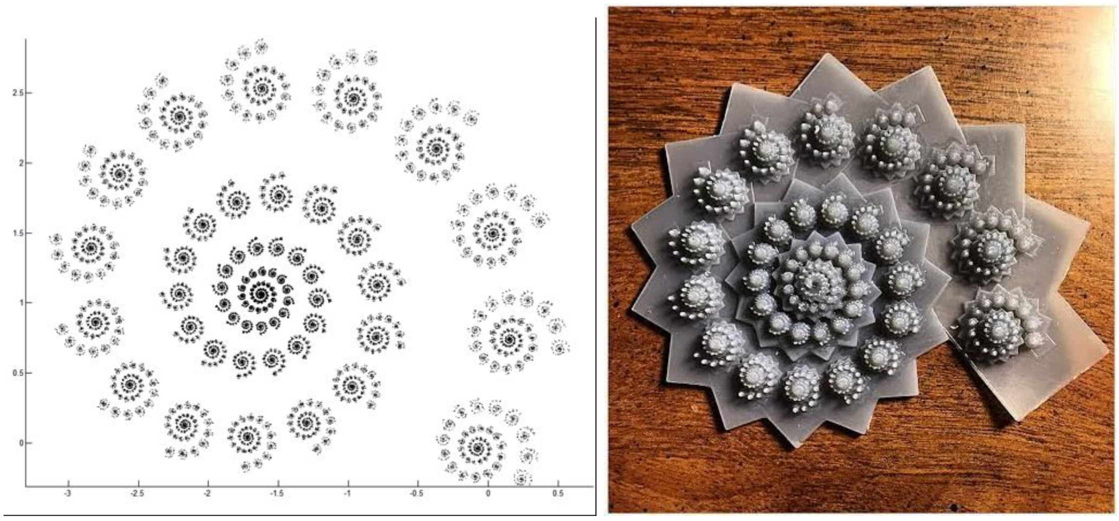
Finally, we successfully started the printing. However, it takes around 4 days to finish the printing process, which means we have to wait for several days to get the finish product.

Previous works

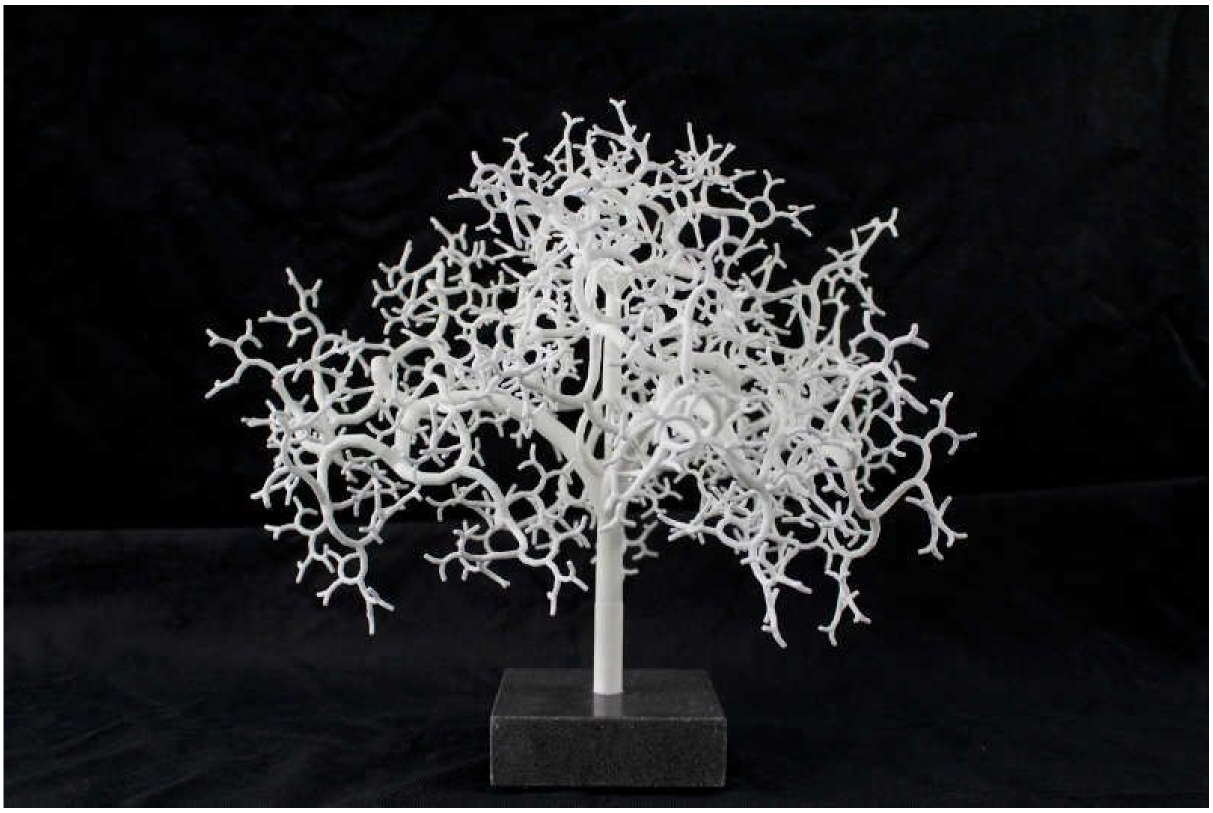


Created by Thomas Oléron Evans (2015)

This fractal was created by graphing the factorials of complex numbers in the complex plane. The colors were determined by whether taking increasing factorials of the number will converge (to one) or diverge (to infinity). This is somewhat related to our concept, as it deals with factorials. However, it is quite dissimilar in nature, because we are approaching combinations not with factorials, but with the Chan Fractal. We are going to stay in the realm of real numbers.



3D printed fractal created by Molly Holder


The Shadow Tree "Ombrier I" created by Henk Mulder

These two pieces is based on the concept of fractal and export it to 3D printing sculpture. The one created by Molly Holder is based on the 2-dimension graph with the concept of fractal, then she export it to 3D model building software and make the graph taller in the third dimension. It looks like an artwork that uses the sculptural technique of relief. For the Shadow Tree created by Henk Mulder, which is also based on the concept of fractal. Different from the previous one, it's layout tends to be a 3D sculpture model. It might requires more skills of building the z-axis on the software when comparing with the first one.