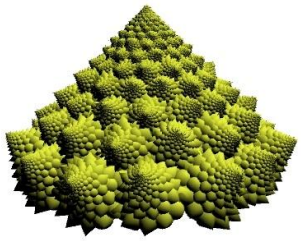


Procedural Flower Generation

Suryaveer Singh Lodha and Jeremy Williams
University of California - Berkeley

As our final project for the course - Solid Modeling, we have developed a procedural generator for interesting and complex shapes and attempted to produce the generated geometry on the Fused Deposition Modeling (FDM) machine. We started this project by generating a Fibonacci Pinecone model and progressed toward the final goal – to generate the Romanesco broccoli model, which has complex 3D fractal geometry based on a logarithmic spiral. The fact that no one had yet developed a 3D model for Romanesco broccoli was a motivation for us to try and develop this aesthetically pleasing geometry. Also the simplicity of the flower and the mathematical beauty behind it always amazed us and hence we decided to build this model as our final project. We did encounter numerous problems on the way related to generating the actual model in 3D, making it look realistic by adding organized jitter and then actually generating such complex geometry with highly intricate features. We discuss our approach to the various difficulties involved, our final model and the lessons learned in the following sections.

Initial Approach: Prototype development

As our final project goal was to develop a procedural generator for the Romanesco broccoli which is a complex object, with multiple levels of recursion and oriented conical geometry at each level, we decided to break down the task into two sub tasks before we try to generate a procedural generator for the Romanesco broccoli.

The first sub task was to determine a platform on which we could build procedural models. We experimented with SideFX Houdini and Autodesk Maya. As we had no prior experience with developing procedural generators in either of these platforms, we decided to generate a model with less complexity first and take it through all the steps - model generation in 3D, perform Boolean operations to get a good boundary representation (b-rep), export the final poly mesh to a STL format and simulate the model in QuickSlice to determine if we should do anything differently in our procedural generator.
The second sub task was to pick a model with lower complexity to test our proposed pipeline which would help us on deciding on a platform. We picked the “Fibonacci Pinecone” as an example as the Pinecone is the evolutionary precursor to the flower, and its spines spiral in a perfect Fibonacci sequence[1]. We picked the

[1] Accessed on 12/15/2011,<http://ournewearth.tv/news_details.php?id=88>

Pinecone model primarily because it had only one level of recursion, with a well defined mathematical equation and the primitives used were spheres, hence we didn't have to worry about orientation. However, the challenges were to decide on an approach to write procedural generators and decide on attributes which could be interactively modified by the user to generate different shapes (Figure 1). To make the model look aesthetically pleasing, we also had to fine tune the degree of separation between primitives and learnt about the use of golden angle in geometry construction for organic shapes.

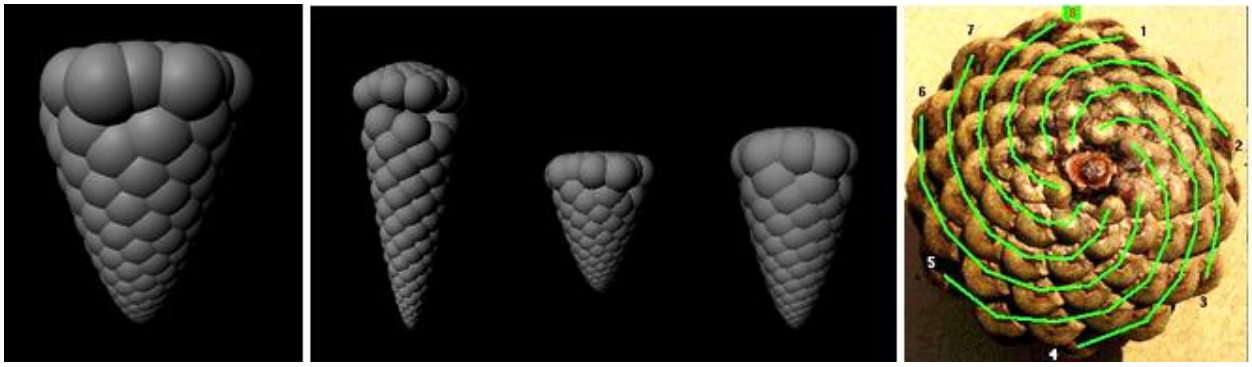


Figure 1: Pinecone generated in Maya with different parameters to attain different organic shapes and a real life pinecone (extreme right) with explanation of the spiral.

Based on our trials with the Fibonacci Pinecone, we decided to use Autodesk Maya. We could generate procedural model by using *python* and Maya also gave us some nice Boolean and rendering options. We could also install a STL export plug-in for Maya which allowed for easy export of geometry form Maya to STL format, which was compatible with the input required by QuickSlice. We also did get a good b-rep for this model from Maya, and that gave us confidence to pick Maya as our platform for developing the Romanesco Broccoli.

Romanesco Broccoli: Introduction

Romanesco Broccoli has great complexity and depends upon mathematical concepts like the logarithmic spiral, so modeling it was challenging and enjoyable .The flower is composed of spirally arranged segments which are identical copies of the whole flower. The copying process continues infinitely as a 3D fractal form.

We wrote a python script in the Maya to produce a Romanesco broccoli model given certain initial parameters. The buds of our Romanesco broccoli form a cone-like shape, so some of the input parameters for the broccoli relate to this underlying structure (Figure 2).

The input parameters for the broccoli are as follows:

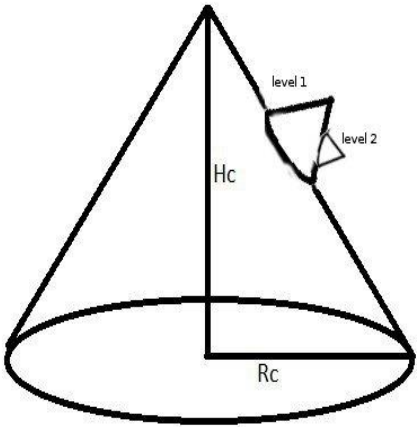


Figure 2

R_c: the base radius of the overall cone shape.
H_c: the height of the cone shape.
N_s: the number of spirals you wish to have on the broccoli.
thetaLimit: the positive theta limit of the log spiral function that defines a spiral.
dTheta: the change in theta for the placing of each cone bud.
coneBudScale: an arbitrary variable that scales all the cones produced for the broccoli.
LODCutoff: level of detail cutoff - if an object's can be contained within a LODCutoff sided cube, then the object is not produced for the broccoli.
recDepth: the recursive depth of the fractal geometry. One means that we put one set of spirals on the largest cone. Two means that we put cones on those smaller cones. Etc.
jitterAngle: the random amount of error allowed in the x,y, and z rotational coordinates for each cone.
jitterScale: the random amount of error allowed in the x,y, and z scalars for each cone.

Romanesco Broccoli: Approach and Final Model

First we worked with spheres to correctly create the log spiral and wrap it around a cone-like shape. From this we progressed to using cones and providing more functionality for the size and placing of the cones (Figure 3). Then, we used less spacing and placed cones upon cones for 3D fractal geometry. Finally, we added some angular and scalar jitter to the cones to give the broccoli a more organic feel.

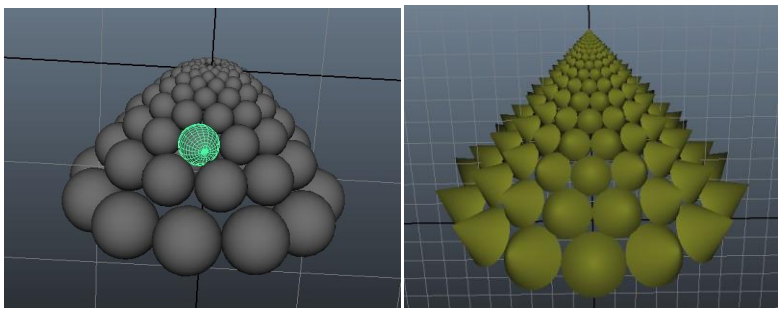
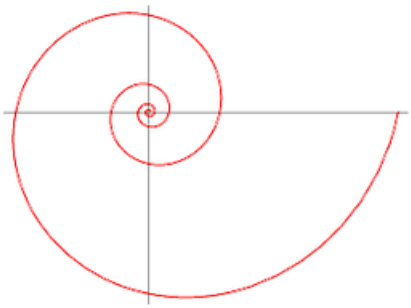


Figure 3: Spheres and cones to make a spiral

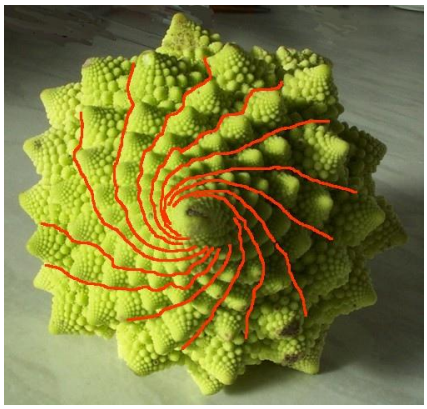
Designing the Romanesco broccoli class proved difficult. The broccoli depended on the log spiral, but its immediate application, the coefficients of the log spiral, and the relationships between parameters were not clear at first.



For clarification the log spiral is defined in polar coordinates by

$$r = Ke^{a\theta}$$

where “K” and “a” are constants defined for the spiral. An image is to the right. First, we created our desired 2D spiral before moving to 3D. This 2D spiral was based off the following image.



For this image, most of the sweep of the log spiral comes happens in the range theta belongs to [0, π]. From this, we realized it would be helpful to have parameters that gave the user greater control over the spirals of the broccoli. Thus, the user can specify “thetaLimit.” For the log spiral, theta spans [-pi, thetaLimit]. Also, the user specifies R_c, the base radius of the cone the broccoli forms around. Using thetaLimit and R_c, we calculated K for the above log spiral equation such that $R_c = Ke^{a*thetaLimit}$

Before moving to 3D, we worked on spacing between spirals. For N_s spirals around the broccoli, at each level, essentially had to perfectly space N_s circles around a circle with radius $Ke^{a*theta}$. Once we used cones, this simply involved letting the diameter of the cones be the length of each side of a regular N_s-gon perfectly enclosing the circle with radius $Ke^{a*theta}$. Some trigonometry gives the answer. Dividing this by two, we get the radius of each cone bud.

Moving to 3D, we defined the height of the current cone bud proportionally to the radial distance from the center broccoli axis. Given H_c, the height of the broccoli cone, we defined the height as $(H_c / R_c) * Ke^{a*theta}$. We oriented the cone buds to the angle of the cone face.

For putting cones on cones, we ran the same operation but with the transformation of the parent cone applied to the child cone. Finally, we added some random jitter to the cones’ orientation and size to add more realism. Final Broccoli models generated in Maya (Figure 4) and comparison between real life broccoli (Figure 5)

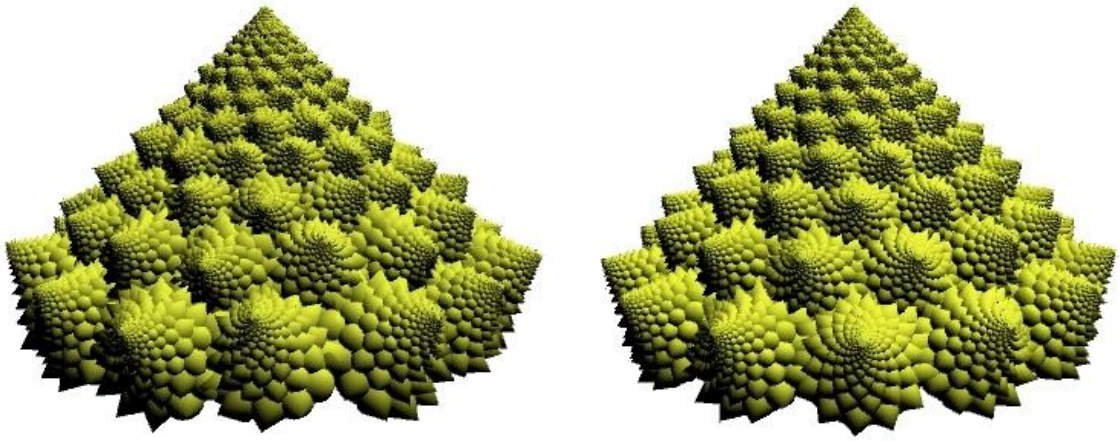


Figure 4: Broccoli Models generated in Maya.

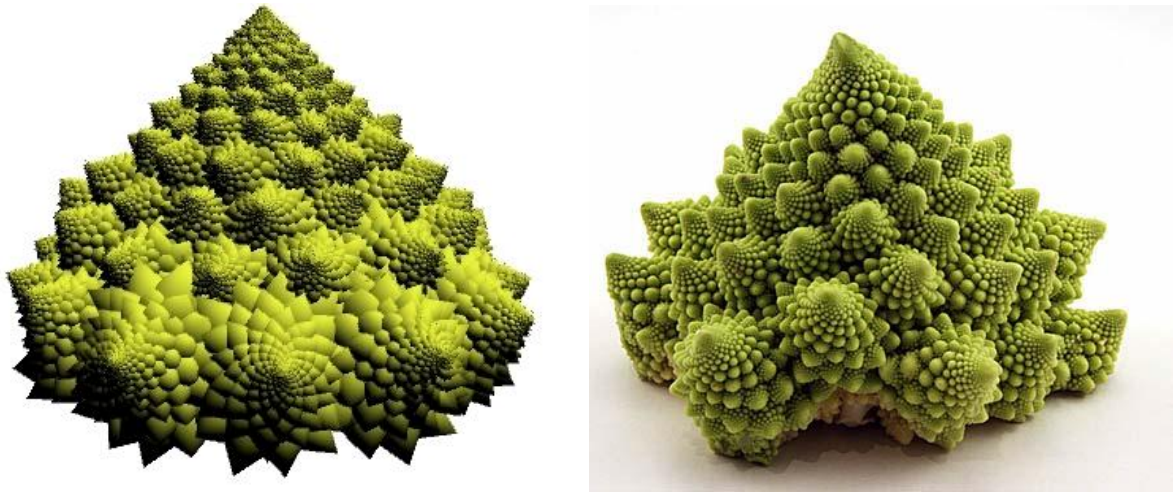
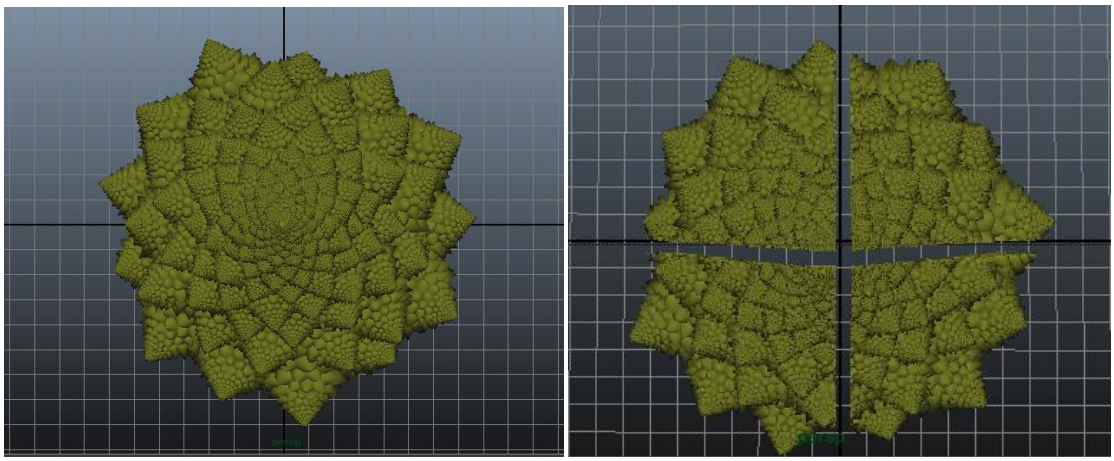


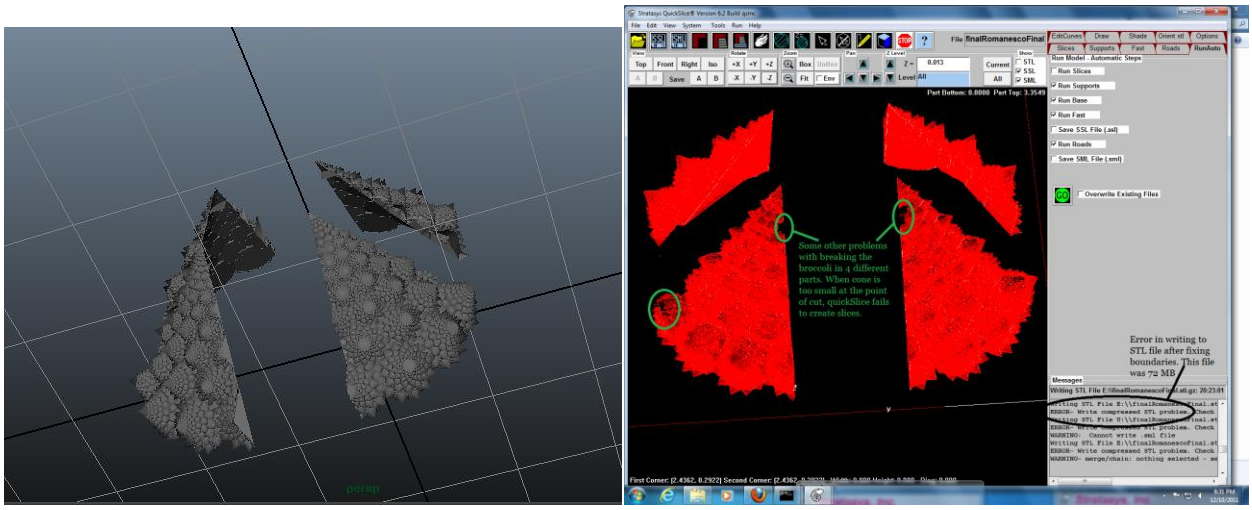
Figure 5: Computer generated broccoli v/s real life broccoli.

Romanesco Broccoli: Model Printing

The final geometry generated by even a two level cone-generation for Romanesco broccoli was quite heavy and complex. We had difficulty in combining 725,000+ triangles into one poly mesh in Maya. In retrospect, it might have been better to develop our own voxel based Boolean operation, but we didn't have enough time for that. The second major difficulty was that for even 4X4X3 inch model, the printing machine was estimated to take almost 2 days. As that was not affordable, we decided to scale the model down to half its size, but then the cones towards the top of each cone in the spiral became too small for the machine to build. It was helpful that we had control for stopping recursion based on scale factors, so we could easily generate a less complex geometry which could be wholly built by the machine. Our final geometry has 185,000 triangles.
We initially decided to break the model into 4 pieces to optimize the build time and material usage, as shown below:



This worked well in Maya, but Quickslice had difficulty in generating the geometry from STL files of the 4 piece model. We believe the reasons to be precision issues between Maya and QuickSlice, and the fact that the model is not exactly symmetrical when we draw a cutting plane along an axis, some of the smaller cones towards the top of parent cone eventually get cut very close to their peak, or very close to their base (such that they are 95% on the other side of cut). This causes problems such as these:



(In Maya) (In QuickSlice)

Also one more thing we noticed was that, even if we try to circumvent this issue with different orientations and cuts, as there are so many features which are very small in size, it would be very tedious and difficult to remove the support from them. On the other hand, it would have been better if we build it as one piece, oriented in such a way that there would be no support on any of the sharp features. Hence we decided to build the model in one piece. There were some problems with initial build with some slices overlapping in the STL files. We used Autodesk 3dsMax's STL checker

to fix our issues with the STL files, which identified 4 overlapping edges. Our final model submission is shown below (Figure 6) and takes 7 hours to build.

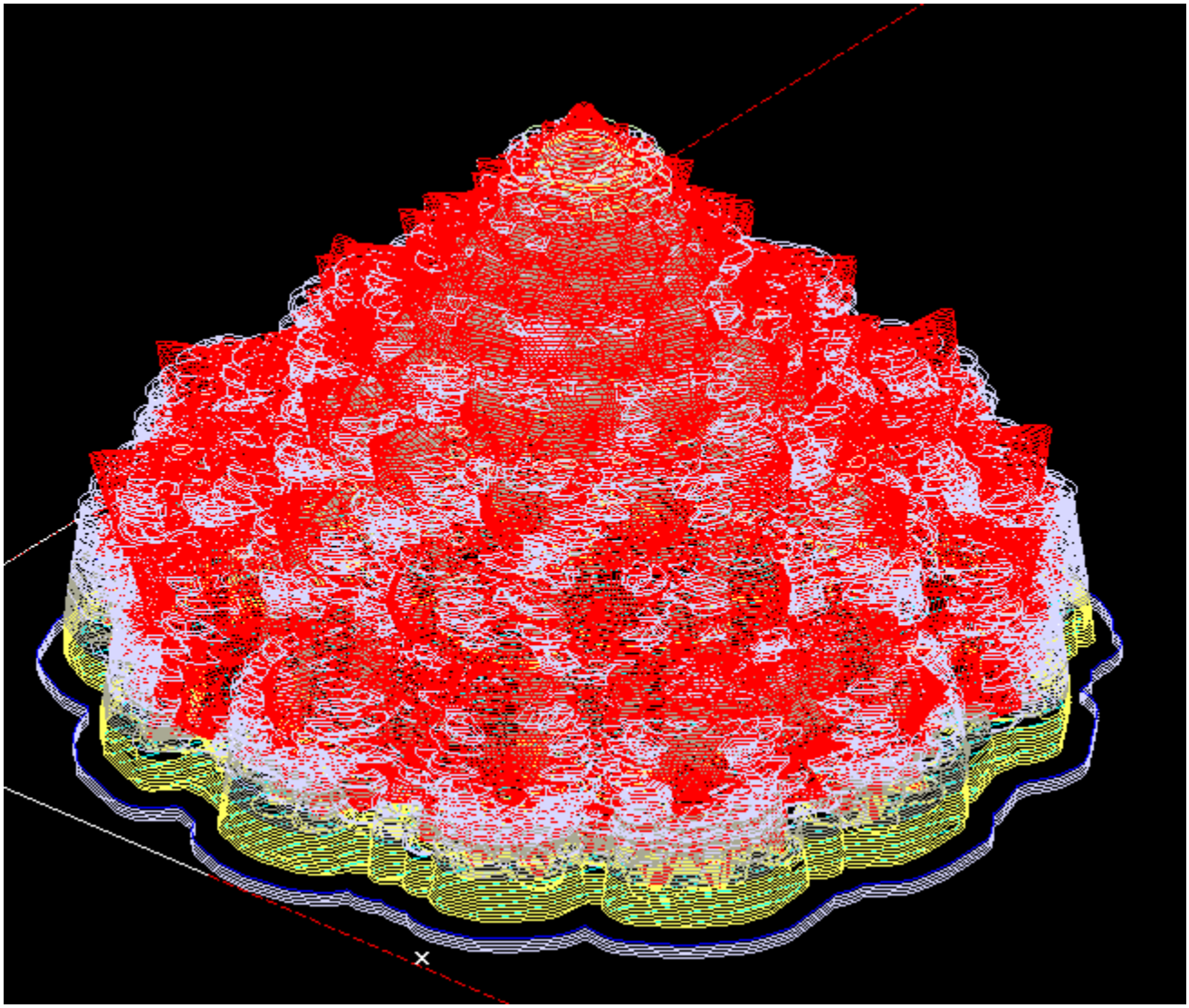


Figure 6: Final Model simulation in QuickSlice with support structures

Acknowledgement

We are greatly thankful to the Professor Carlo H. Sequin for his advice and insights during all phases of development and model building, and to Toby Mitchell for his explanation of the logarithmic spiral and the use of a spiral transform.