# Snowflaker Plugin for 3dsMax

Mariana Ávalos Arce
Universidad Panamericana
Advanced Plugin Development
Autumn 2020

Guadalajara, Mexico

# 1 Introduction

The following document expalins the logic process during the development of the Snowflaker Plugin v1.0. This plugin's main aim is to create a snowflake mesh geometry inside a 3dsMax scene, using Autodesk's programming language MAXScript.

# 2 Creating The Set of Vertices

## 2.1 Inner Ring

The first thing is to create a ring of points, which will be the *inner ring* of the snowflake, a ring from which all the spikes will come out of, as extensions of *each* even side. For that I used the concept of polar coordinates:

$$\begin{aligned} x &= rcos(t) \\ y &= rsin(t) \end{aligned} \tag{1}$$

Where $r$ symbolizes the Radius parameter of the plugin, given by the user's dragging of the mouse over the X axis of the grid inside 3dsMax. Then, the parameter of Spikes multiplied by 2 will provide the **total sides** of the ring, and this will be used to define the `step_angle` that will indicate when to place a vertex for the ring (t variable). The default is 6 spikes, and therefore,

$$step_{angle} = \frac{360}{sides} = \frac{360}{12} = 30^{\circ} \tag{2}$$

The task is then to create an inner ring with the previous parameters plus Thickness parameter defining the width of the ring, just as shown below with a 5-spiked example.

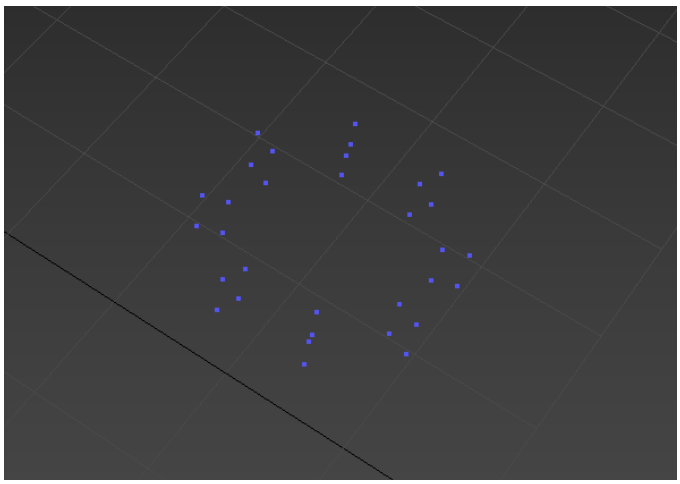
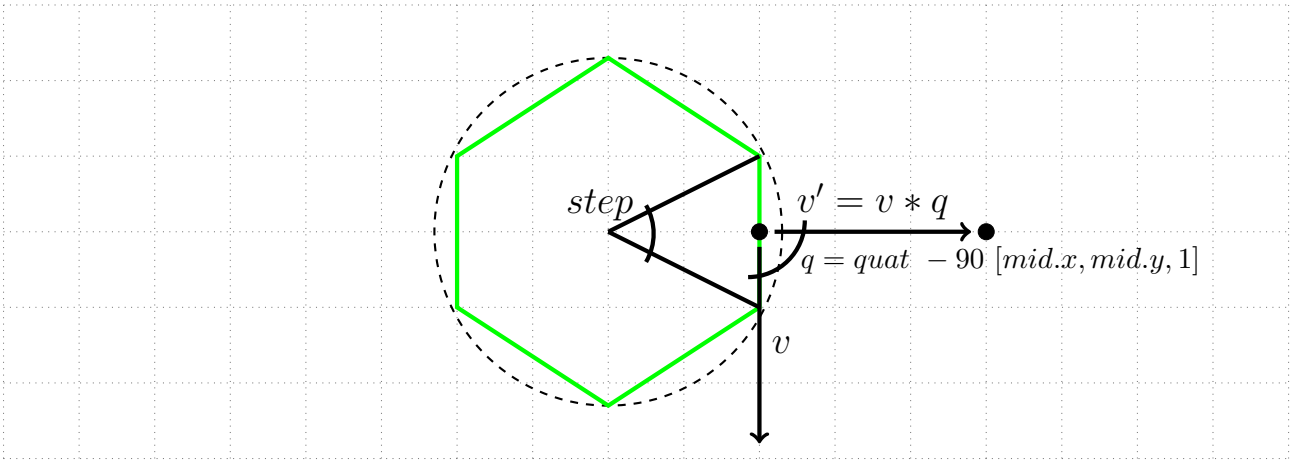


Figure 1: Inner ring vertices

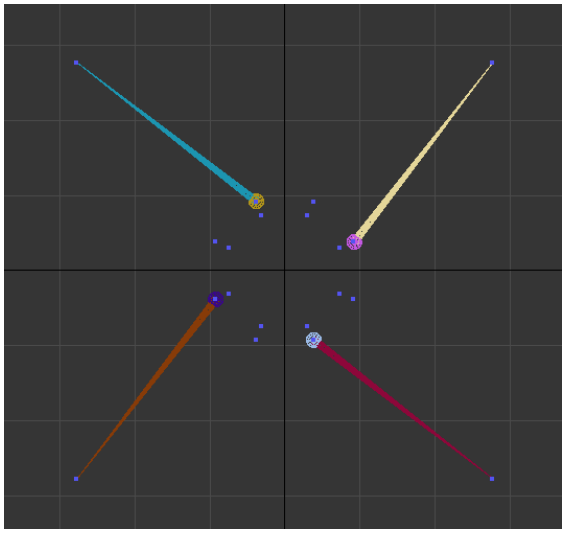
## 2.2 Main Spike End Point

The next step is to define the main spike points, where these points are the finish point of each spike of the snowflake. For this, I started by simply adding `step_angle / 2` to the same inner ring loop, just with the $r$ parameter using the **Spike Size** this time as length. It created the desired result, but I wanted to reuse the same function to create the sub-spikes, so I had to think of another way.

The way I came up with was to define the spike end point first by placing an imaginary vector coming from the middle point between each odd pair of the inner ring's vertices, then rotating this vector by 90° and an axis using **quaternions**, and if this vector's length is the **Spike Size** parameter, then after rotating it, the end of the vector would be the desired spike end point. This idea is best explained by the following graphic of a hypothetical 3-spiked snowflake (6 sides total).




The angle in the quaternion is -90 because in 3dsMax, the quaternion rotation works by default in clockwise sense, so intead of 90 just changed it to -90 [1].



This algorithm was coded into a function called `createSpike()`, which mainly receives two points, `p_0` and `p_1`, the angle and the size of the vector (spike size), apart from the amount of spike segments. The end point of each spike is put by the function and this can be used for the future sub-spikes that will be needed. The output is shown in the previous image.

## 2.3 Main Spike Segment Points

Once having the end point of a spike, we calculate the distance and pointing vector between its `p_0` and the end point. Having this vector, we normalize it and using a factor like the one below,

$$factor = \frac{distance}{segments}, \tag{3}$$

We multiply this factor by the normalized pointing vector, and then add it to either `p_0` or `p_1`, we place the segment points of a spike. The output of this calculation can be seen in the image below.

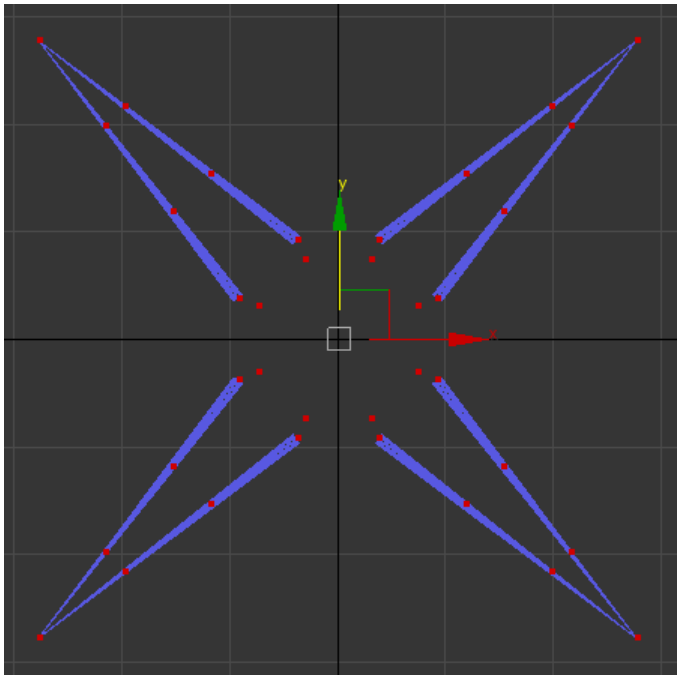



Figure 2: Segment vertices

## 2.4 Creating Sub-spikes

By having the previous function, the next thing in order to create the subs-pikes becomes simply to use it again for each sub-spike of the snowflake, and therefore sending the two involved vertices, which this time will be spike vertices, an angle of 120/-120, and the sub-spike

length determined in the user parameters of the UI, apart from the segments. First, a constant sub-spike was used, and it produced the following output after calling `createSpike()` for each **odd** pair of vertices.

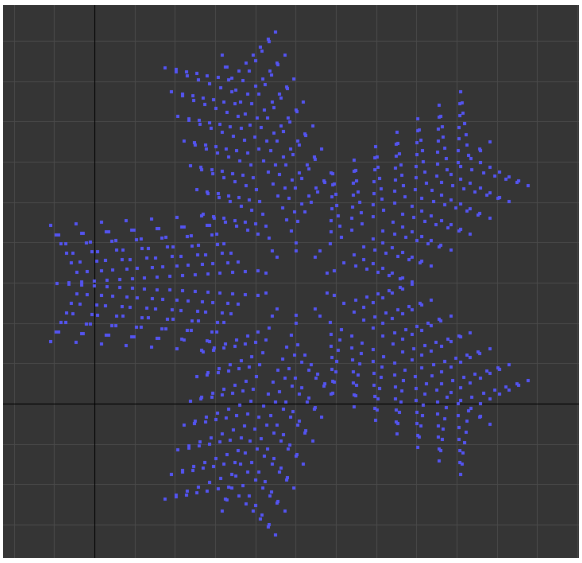


Figure 3: Sub-spike vertices

## 2.5 Sub-spike Shapes

I created 4 types of snowflake shapes: Decay (default), Growth, Sinusoidal and Constant. For this, I defined **frequency** and **amplitude** parameters in the UI, used for the Sinusoidal shape. For the Decay and Growth, I calculated a factor that would represent the size differential,

$$\Delta s = \frac{\left(\frac{distance}{2}\right)}{segments}, \tag{4}$$

and this would either be added (Growth) or substracted (Decay) from the current size that is sent to the `createSpike()` function. The Sinusoidal shape was calculated with the size differential being given by

$$\Delta s = \mathrm{A} \times sin(F \times i), \tag{5}$$

Where i is the current sub-spike. This Sinusoidal differential is then added to the size of the sub-spike and sent to `createSpike()` to create current sub-spikes. As a side note, I added the **Only In Tip** checkbox and the **Skip Degree** parameter, which where handled by changing the fact **odd** pairs of vertices where sent to create a sub-spike, by using **modulo** operator with the degree the user gave.

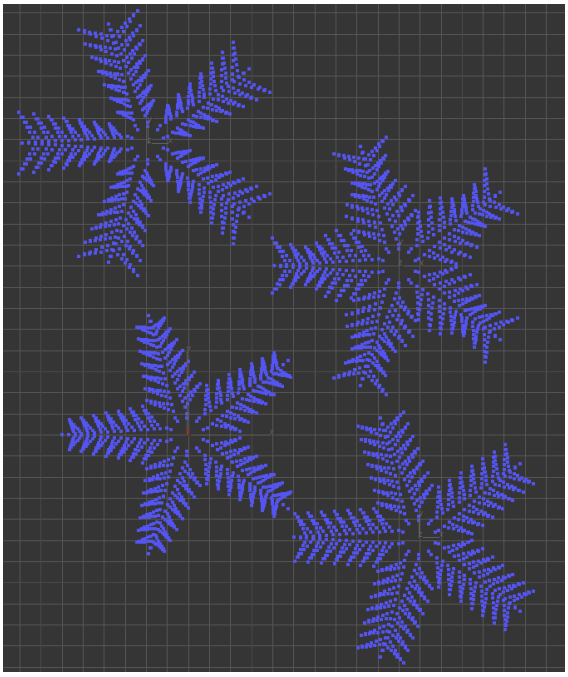


Figure 4: Four snowflake shapes

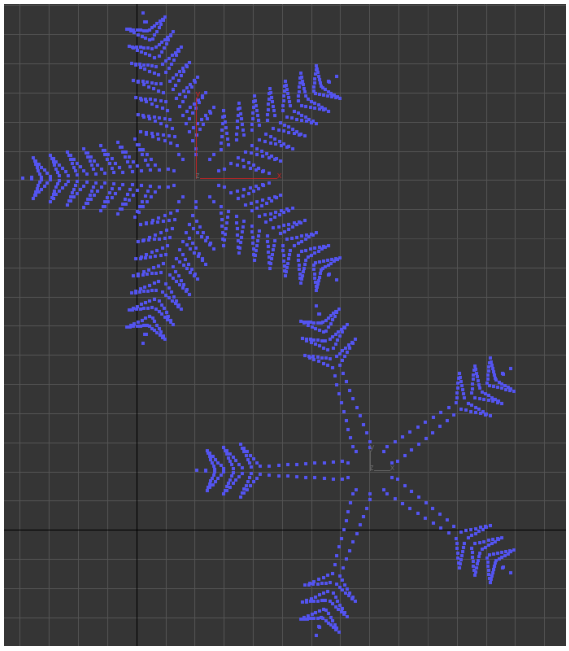


Figure 5: Only In Tip parameter

# 3 Creating The Faces

## 3.1 Joining Vertices as Faces

After setting all types of vertex arrangements, the next thing was creating the snowflake faces. I started by the inner ring, then the main spikes and the sub-spikes in loops of even 5 levels, because of the complexity of the vertex-face pattern.

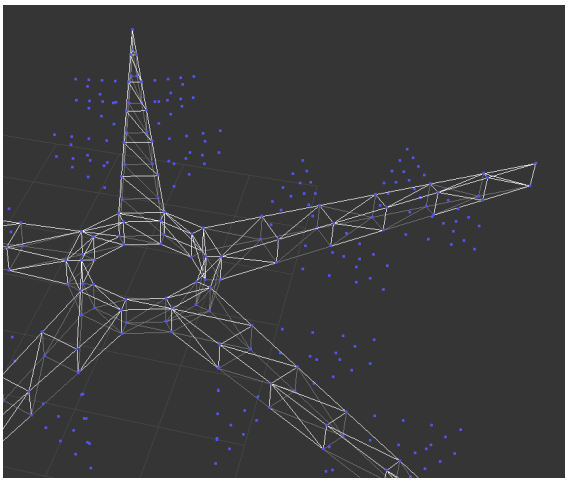


Figure 6: Main spike faces

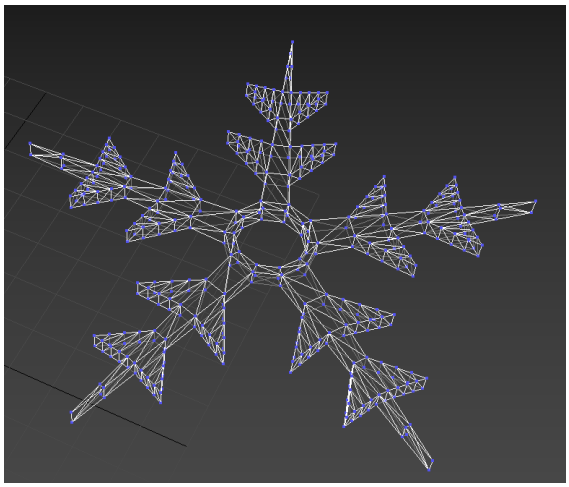


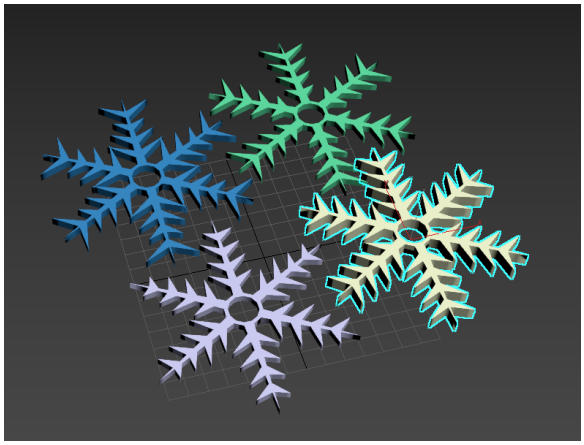
Figure 7: Faces finished

## 3.2 Arranging The Smoothing Groups

After having the bare faces formed, the next step was to figure out how to place the faces in a desired smoothing group. In 3dsMax, the smoothing groups are not as simple as assigning an integer with the group. Instead, for example, if you want a face to be in group 4, you must input an 8. The relationship this follows is the one below.

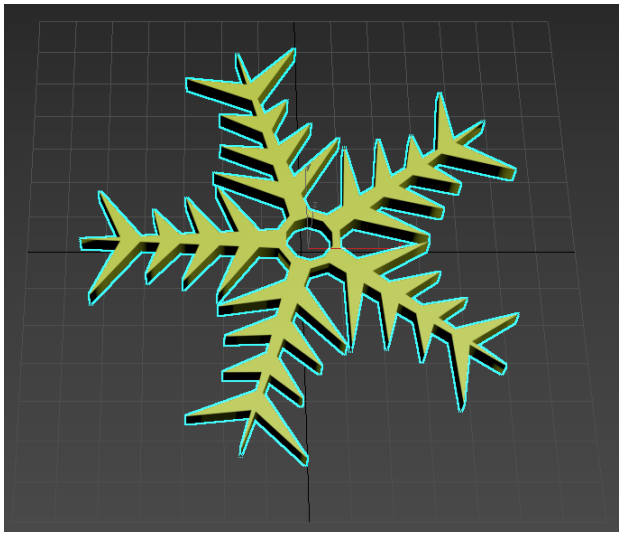
$$sg = 2^{N-1}, \tag{6}$$

Where N is the desired smoothing group, and sg the resulting number you must send in MAXScript. The result of the smoothing grouped faces is presented in the following image.



## 3.3 Variation Parameters

I added two variation slides in UI: angle and size of the sub-spikes. These sliders set the **limit of randomness** for these two variables, making each of the sub-spikes not perfectly aligned. The default is zero (no randomness), but when changed the snowflake differs slightly.



# 4 Acknowledgements

This plugin was my final project for the subject Advanced Plugin Development, and therefore I thank my Professor for the freedom of the plugin's objective and documentation, as well as the knowledge provided during the lectures. It was a lot of fun to do this during a pandemic semester.

# References

[1] 3D Modeling for Programmers. *Quaternions*. 2020. URL: https://cathyatseneca.gitbooks.io/3d-modelling-for-programmers/content/mathematical_background/quaternions.html.