DREAMSTAR GENERATOR

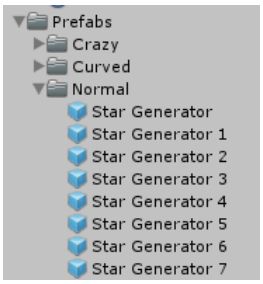
INTRODUCTION:

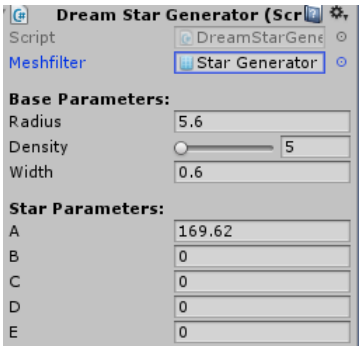
Simple algorithms can create fancy visuall effects. In this case star like structures.

The Dreamstar Generator is capable of creating procedural generated 2D star like meshes by using such algorithms. In combination with special materials, you can create fancy and nice visuall effects. Unlike other generator, this generator directly creates the mesh instead of merging predefined shapes together. This allows much larger density and has a far greater performance which allows animations.

HOW TO USE:

For fast testing, just drag in one of those prefabs in your scene and play around with the values as you see the results immediately. The most important parameters are the radius, density and width parameters. The radius parameter is an overall multiplier of the final result. The density describes the amount of points generated. The lower the value, the denser it is with a minimum value of 0.1 as smaller values would create an infinite amount of points. The width describes the line width of the result.





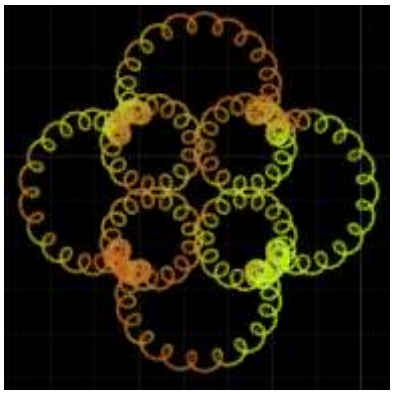
The Parameter A, B, C, D, E are simple float values which are used by the generation algorithm. They can have any value including negative one. Depending on the algorithm, not every value here must be used. Often one value is enough.

Other algorithms can have other values but it is recommended to use these values as these can be animated which is explained below.

Currently there are 3 different algorithms which can create an unlimited variation of interesting results.

DREAMSTAR MIXER:

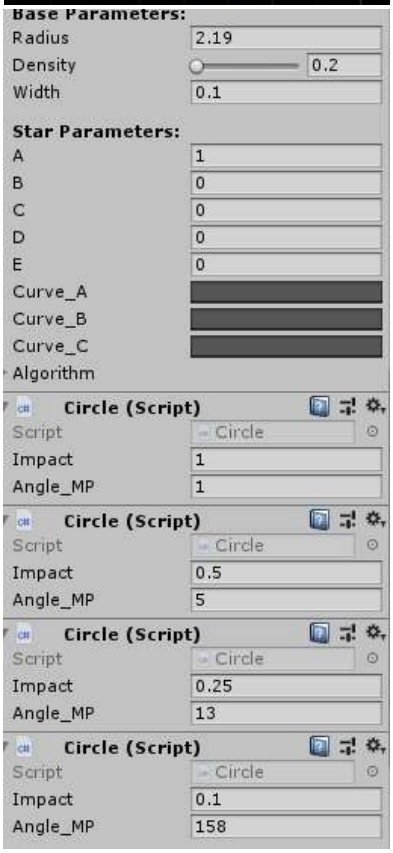
The mixer is a new feature which allows the combination of multiple algorithms together. This allows more possibilities than before which is shown in the MagicStars_Mixer sample scene. To use this feature, add the mixer component to a game object and then add any number of algorithm components to it.



These algorithms are similar to the 3 provided already but exist as extra component so multiple instances can be added. During calculation, the position value is combined together in an additive way so order does not matter.

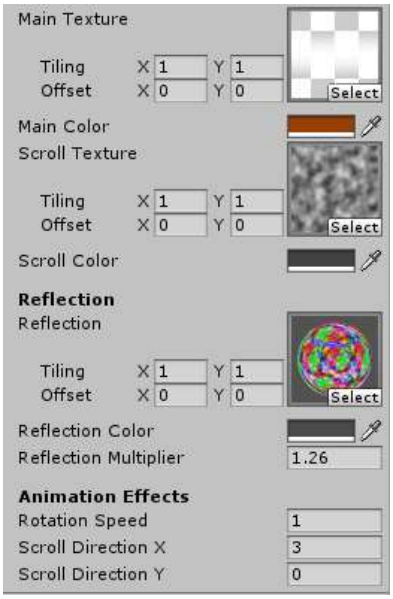
The most important parameters are Impact and Angle_MP. The impact value is a multiplier of the result of the concrete calculation. For example 0 means no influence at all. Angle_MP multiplies the input angle of the concrete calculation.

Due to the insane complexity, it is recommended to just play around with the values because the impact of parameters rise exponentially with each algorithm attached. Procedural content generation is always unpredictable (for humans at least ☺) . In the right sample, all 4 circle components share the same star parameters where the value A is kept at 1.



MAGICSTAR SHADER:

This asset also contains a fancy shader which combines reflection and scrolling to create nice materials.

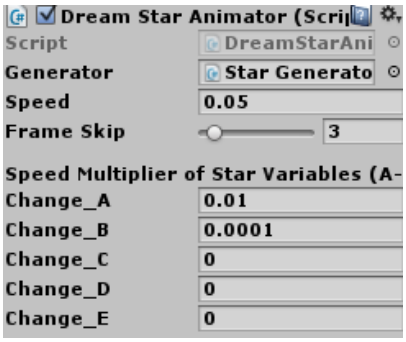


Parameters:

- Main Texture: The main Texture. Alpha value is used by all other textures.
- Main Color: Color multiplier
- Scroll Texture: Additive scroll texture
- Scroll Color: According color
- Reflection: Any cubemap for reflection
- Reflection Color: Color multiplier
- Reflection Multiplier: Reflection strength
- Rotation Speed: Cubemap rotation
- Scroll Direction X/Y: Scroll speed

ANIMATION:

The results can also be animated as the update cycle is fast enough. To let them animate, attach the Star Animator component to the game object. It automatically assigns the generator entry when it is empty so you don't have to do it yourself.



The speed value is the main increment value. It is used as base value every fixed update. Every one of those five values is altered by the speed value multiplied with the according multiplier (Change_X).

It is recommended to use very low speed parameters because small changes can have a huge impact.

EXPORTING RESULTS:

There is also a mesh exporter included. It is the located in Scripts/MeshGenerator. Just attach it to the component and export it as .asset or .obj file.

HOW TO CREATE YOUR OWN ALGORITHMS: (REQUIRES BASIC C# SKILLS):

It is very easy to create your own algorithm by simply derive from the star generator class and override the function "Initialize" and "StarAlgorithm". The "Initialize" function is called before the generation is executed. Use this function to set certain values or do heavy operations.

The "star algorithm" uses Angle as parameter and returns Vector3 data type. This is the star algorithm which describes how the result will look like. With simple math functions, crazy results can be generated.

IMPORTANT THINGS:

When you create something you can also doing it wrong. Here is a bunch of advices for creating procedural generated stuff:

- Divisions: Although the generator handles divide by zero, divide by zero can occur especially if the divisor is a changeable parameter.
- Using Patented Formulars: Yes it is true! Formulas can be patented (if you have the money). One example is the Superformular which can create many different objects. Whenever you commercially sell your product, you have to make sure that you don't inflict some cancerous patents. If it would not be patented, it would be included to this product but it is so I am not allowed to include it unless I pay some "WTF" licence fee. Patents can be really ridiculous in slowing down science. Imagine someone patenting Pythagoras.
- Randomness: Use a changeable parameter as seed or you will completely lose control over your generator.

LAST NOTES:

If you have any questions, suggestions, bug reporting don't hesitate to contact me. If you are going to sell a game which uses this asset, inform me because I may buy your game and play it ☺

Contact Information:

E-Mail: mhartl.mmt-b2013@fh-salzburg.ac.at

Homepage: http://fraktalia.org/