Check online for a newer version

Unity Asset - Documentation

Starfield Shaders

v1.1.9

by Tadej Slivnik
tadej.slivnik@outlook.com

March 15, 2020

Contents

1 Introduction

Starfield Shaders is a Unity asset that contains starfield assets and other transparent/space effects for 2D and 3D projects. Shaders are custom made vertex/fragment or geometry shaders, designed for performance. I also provide powerful yet simple to use scripts. Minimal to no coding is required on the user's part.

My goal when creating scripts is to make them smart and very easy to use. I also try to provide all the functions you will ever need for controlling my assets through your own scripts if you wish to do so. I have also made sure users can generate meshes and use scripts outside of play-mode.

1.1 Importing the asset

Since users usually won't be using both 2D and 3D assets for the same project, I have made it easy to import only 2D or 3D folders/assets. The *Shared* folder must always be imported.

1.2 How To Start

All of my scripts for generating starfields/noise are designed to be used in a simple and consisted way. To start using my asset, follow the steps below:

1. Create an empty GameObject in your scene
2. Add **only one** of my scripts to the created GameObject
   (Refer to the next two sections for more information on scripts)
3. Click on the "Generate Mesh" and "Apply Data" buttons

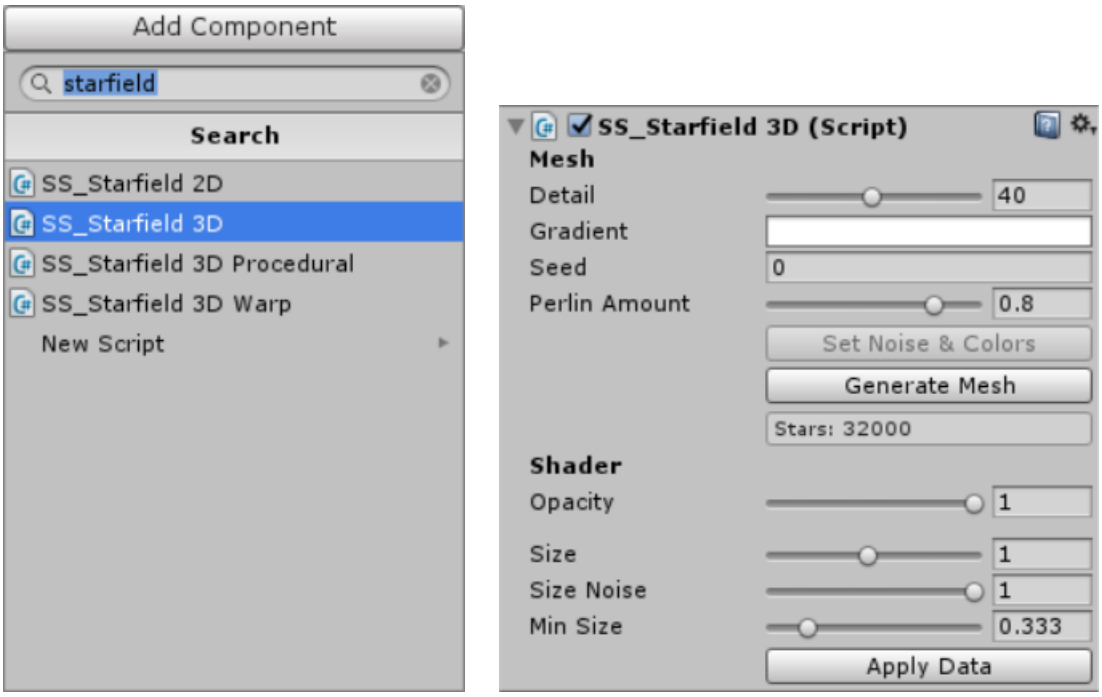

Figure 1: Adding a 3D starfield script.

Instancing out of play-mode is not supported. You can get different starfields/nebulae in edit-mode by deleting a material and **Applying Data** again. This forces my scripts to create a new material.

1.3 2D Scripts

Scripts/shaders also contain warp effects. Included scripts for 2D projects:

- SS_Noise2D
  Generates 2D noise. Also provides the parallax effect. This is not meant to be used as a background.
- SS_Noise2DMesh
  Static class that contains functions for generating plane meshes, designed to be used with the 2D noise shader.
- SS_Starfield2D
  Generates 2D starfields. Also provides the parallax effect.
- SS_Starfield2DMesh
  Static class that contains functions for generating plane meshes, designed to be used with the 2D starfield shader.

1.4 3D Scripts

Included scripts for 3D projects:

- SS_Noise3D
  Generates noise backgrounds.
- SS_Noise3DWarp
  Generates noise warp effects. This is not mean to be used as a background (the shader is transparent).
- SS_Noise3DWarpMesh
  Static class that contains functions for generating cylindrical meshes, designed to be used with the 3D noise warp shader.
- SS_SphereMesh
  Static class that contains functions for generating sphere meshes, designed to be used with the 3D starfield and noise shader.
- SS_Starfield3D
  Generates 3D starfields.
- SS_Starfield3DProcedural
  Generates 3D procedural starfields.
- SS_Starfield3DProceduralMesh
  Static class that contains functions for generating triangles, designed to be used with the 3D procedural starfield shader.
- SS_Starfield3DWarp
  Generates warp effects for stars.
- SS_Starfield3DWarpMesh
  Static class that contains functions for generating cylindrical meshes, designed to be used with the 3D starfield warp shader.

2 Meshes and Shaders

This asset is optimized for both orthographic and perspective camera view. While shaders and scripts for both perspectives are built in the same way, they each have their differences in features they provide and meshes they use with the help of which starfields are rendered.

Planes (2D) and spheres (3D) are generated through scripts and can be generated in-game. Instead of providing a noise texture to shaders, the noise is stored in the mesh's UV channel. This means that the provided meshes do not contain the correct UV information. The same goes for colors: the color is not sampled from a gradient texture but is instead saved into the mesh itself. These decisions were made mainly for performance reasons and simplicity. Meshes can provide up to 128000 stars per starfield.

All shaders are transparent and rendered in the background or transparent render queue. All calculations are done without textures (no texture sampling) and mainly in the vertex or geometry part. Properties can be changed in real-time, meaning that the shaders can also be animated if needed.

When you are resizing stars, you are actually resizing their triangles. This makes sure the overdraw for transparent objects in Unity is as low as possible. One other thing to be aware of is that with increasing the number of triangles on meshes, you are decreasing the default star size.

Warning 1: the performance will decrease significantly when rendering huge stars (triangles) - the whole point of these shaders is to give you realistically sized stars/starfields.

Warning 2: Unity's skybox draws over these shaders. Visit my cloud storage for skybox shaders that do not draw over starfields and noise.

Warning 3: Noise shaders are vertex/fragment shaders but contain **a lot of calculations**, some devices may have difficulties rendering these shaders. Noise shaders are not meant to replace your main space background!

Warning 4: *SS_Starfield3DWarp.shader* and *SS_Starfield3DProcedural.shader* are still **geometry** shaders. These two shaders may not work on all devices (e.g.: Apple devices).

2.1 2D

Planes are used in 2D because we don't need the additional dimension. The most important setting in this mode is the "starfield bounds" setting. These bounds make sure that triangles aren't rendered outside of the given X (width) and Y (height) values. They are still rendered though, just on the opposite side. This gives a parallax effect.

The mesh can have as many triangles as you want (up to 128000). By increasing the number of triangles, the default star size will be smaller. This is due to the decision of making the mesh height constant. The width, on the other hand, can vary in its length. The 2D starfield version also has the **parallax effect**. The parallax scrolling effect is calculated in the shader.

2.2 3D

2.2.1 Skybox

3D skyboxes are rendered with the help of spheres. Spheres are icosahedron-based because I believe this parametrization provides the best and most consistent distribution of stars. The spheres' bounds are set to a huge float number to remove culling. Starfields may still get culled when moving **very** far away from the origin - if that is the case, append the starfield object to the camera's GameObject. Spheres are transformed into view space, removing the need for positioning the starfield.

These sphere meshes also produce an error in the inspector when using *SS_Starfield3D*. This happens when you click on the mesh to look at its preview. Nothing is actually wrong, the inspector just doesn't know how to render a mesh preview, because the mesh bounds are set to infinity (to prevent culling).

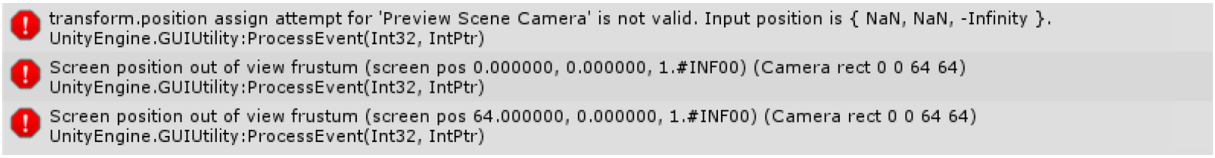

Figure 2: Screen position out of view frustum - mesh error.

2.2.2 Procedural

Procedural starfields are still in testing. I am not entirely satisfied with the speed of the generation. These scripts may drastically change in the future (3D perlin noise in shaders).

Procedural starfields are rendered with the help of triangles, positioned at integer coordinates in three dimensional space. The shader for procedural starfields uses the billboard effect, so all the triangles are always rotated toward the camera. Procedural generation is achieved with the help of 3D perlin noise.

Perlin noise is also used for coloring stars in this case. It is good to know that perlin noise values (in practice) are somewhat normally distibuted. This means that colors on the center of the gradient are

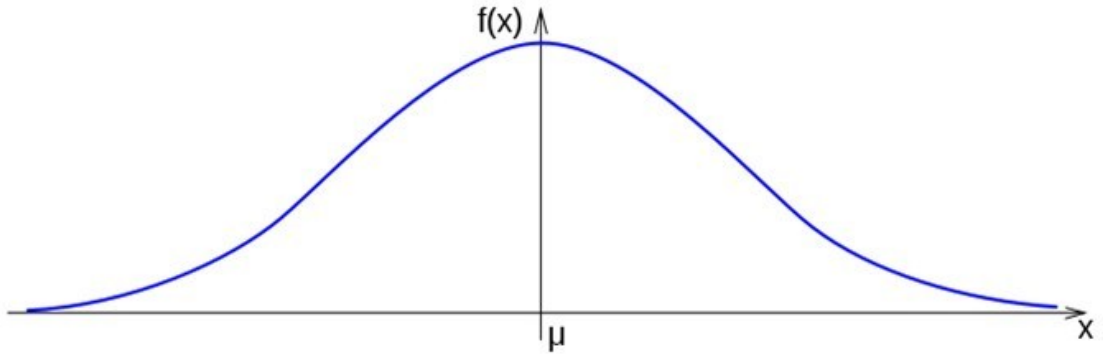

Figure 3: Normal distribution.

more likely and colors on the edges are less likely.

2.3 Augmented Reality

Procedural stars were created with AR in mind. Thought procedural generation may not work by default, because it is based on the starfields center position which never changes in AR projects (by default).

Augmented reality doesn't render objects that are in the background rendering queue (skyboxes). You can also make skybox shaders work in AR by changing the rendering queue ("Queue") in the shader from "background" to "transparent".

```
SubShader
{
    Tags
    {
        "RenderType"="Transparent"
        "Queue"="Background+2"
        "DisableBatching"="True"
        "ForceNoShadowCasting"="True"
        "IgnoreProjector"="True"
        "PreviewType"="Skybox"
    }
```

Figure 4: Shader tags section.

3 Performance/Quality

3.1 Overdraw

You can duplicate stars and noise to get a denser look, but this comes with a noticeable performance cost - overdraw.

Overdraw is the term for when the same pixel is drawn multiple times. This happens when objects are drawn on top of other objects and contributes greatly to fill rate issues. To understand overdraw, we must understand the order in which Unity draws objects in the scene. An object's shader determines its draw order, usually by specifying which render queue the object is in. Unity uses this information to draw objects in a strict order.

With multiple starfields, if the stars are small, overdraw is not that large of an issue because of the way the mesh and shader work together (small triangles). Noises, on the other hand, are drawn over the whole screen. If you use two noises, you will be drawing each pixel at least two times (also depending on your own transparent effects). The Unity profiler (GPU) is a great way to see how much of an impact these things have.

3.2 Noise Detail

Perlin noise is calculated per vertex. The more vertices a mesh has, the more detailed the noise will look. With that in mind, you do not necessarily need a lot of vertices. Sometimes, if the noise scale is high enough, you could get away with low detailed meshes. In 2D, it also depends on how fast the parallax/animation effect is.

4 Contact

Please do not hesitate to contact me, if you have any other questions regarding my assets. Want me to add features? Tell me what you would like to see in future releases. Does something need improving? Suggest how I can make it better! If you find any bugs, please report them via email!

Send me an email at **tadej.slivnik@outlook.com**

Check my other Unity assets on **My Publisher Page**

Find high-quality images of my assets on **My Website**

You can also find a few assets on **My Cloud Storage**