# Documentation

## Procedural Planets and Stars

v1.3

Check online for a possible newer version

Tadej Slivnik
tadej.slivnik@outlook.com

October 5, 2018

# Contents

# 1 Introduction

## 1.1 Substance Archives

What are *Substance Archives* (.sbsar files)? A *Substance Archive* is an archive created by *Allegorithmic Substance Designer*, a program used to create materials for use in 3D content, such as animations, special effects, and video games. These files can be used to create 3D materials, such as concrete, carpet, glass, marble, metal, wood or just different pattern and noises. They can be opened by a variety of programs, including Substance Designer, Substance Painter, Unity, Unreal Engine 4, Maya, 3ds Max, Cinema 4D, CATIA, TouchDesigner...

In Unity, these archives contain *Procedural Materials* which generate textures based on seeds and other parameters. In this asset *Procedural Materials* are used to generate different normal maps, height maps, city lights textures or simply different noise textures.

**Unity 2018 does not support *Procedural Materials* anymore**. In Unity 2018 *Procedural Materials* are replaced by *Substance Graphs*. *Allegorithmic* are in the process of releasing their own plugin for Unity 2018 to allow more rapid implementation of features compared to when it was integrated in Unity 2017 or earlier versions. The *Substance in Unity* plugin for Unity 2018 is currently in open beta and available on the Unity Asset Store. For release notes and beta information, please check the Beta Information page. Currently it is not possible to instantiate *Materials/Graphs* and conveniently generate textures from scripts. It's also good to be aware that beta builds are bound to break references in your scene every so often. As soon as they implement all the functionality, I will update this asset with easy to use generation scripts for Unity 2018. For more information regarding Substances, visit Allegorithmic's official website.

## 1.2 Generation Workflow (Unity 2017)

**Quick Start Tutorial Video**

I have provided a few types of scripts for run-time generation. *Generators* are scripts which generate objects based on the *Data* provided. Before you can use the provided *Generator* scripts, you must first include the respective *Manager* script. For example, the *Generator PPS_Body* script needs *PPS_BodyManager* script in the scene.

*Managers* are scripts that hold all the data necessary for generation. You need **only one** *Manager* per scene. **It is preferred that you do not destroy *Managers* when changing scenes.** All the *Managers* can be on the same *GameObject* and it is actually preferred that way. At the start you will be prompted to assign all the references via a button in the inspector and alerted if there are any errors. After the necessary references are assigned, you will be able to customize various settings which are **global**! That means that you can not have two different scenes with the same *Manager* but different settings. Once you build your solution, these settings can not be changed when changing scenes. That also means that you can not have a scene in 2D and another scene in 3D (at least not with the provided scripts and shader settings; contact me if you absolutely require said feature).

*Data* scripts are scriptable objects that contain only the necessary data for generation and a few functions that take care of injecting new data into *Materials* and *ProceduralMaterials*. *Data* object are created from the *create* menu in the *inspector*, under *PPS*. You can use these objects as a data container for a single planet or a whole planet type, e.g. water planets, lava planets, rocky planets. The *Data* object contain a custom structs which generate a random value in the user specified range. The checkbox toggles randomness. Instead of colors the scriptable object uses *Gradients*, which work the same way: the object chooses a random color in the provided *Gradient*. Of course the random value depends on the seed that is chosen on the *Generator* component.

*Generators* have the same name as their *Managers* with the word "Manager" omitted. Once the required *Manager* script is set-up and once a data object of the correct type is provided, you can start generating. In edit-mode all the generator of the same type use the same material. Material instantiating is only available in play-mode and of course build. Thanks to scriptable objects, data


Figure 1: Rim Radius will always have a value of 0.03. Rim Power will be a (based on a seed) random value $x \in [1.4, 3.5]$.

changes are also saved in play-mode.
Provided *Generator* scripts (*Managers* are named respectively):

- PPS_Body
- PPS_PlanetaryRing
- PPS_Starfield2D
- PPS_Starfield3D
- PPS_Nebula2D
- PPS_Skybox

If you want to make your own scripts for generation, you can delete the *Generation* folder under *PPS/Scripts*. That should remove all the generation features I have provided without errors.

# 2 Meshes

## 2.1 Spheres

For planets and suns I have provided several spheres, named PPS_SphereX, where the last part stands for the number of triangles the sphere is made of. I have found that 1120 triangles is the minimum triangles required for a satisfactory rendering of the rim. The provided spheres are parameterized by the standard spherical coordinates and have additional vertices near the poles for less distortions.

Shaders do not render correctly (only the rim part) on the Unity's built-in sphere. This is mainly because the build-in sphere has a diameter of 1 whereas the provided spheres have a radius of 1 (diamater of 2). If you desperately need the shaders to work on the built-in Unity sphere, please contact me.

For 3D starfields, you should use the provided PPS_StarSphereX, where again the last part stands for the number of triangles the sphere is made of. If you need more stars, you can have multiple *PPS_Starfield3D* objects in the scene. **It is very important not to use spheres that are used for other stuff**, because the provided mesh gets modified on Start (mesh culling). If a mesh was modified that was not meant to be modified, just restart Unity. The PPS_StarSphereX spheres do not have the correct uv mapping and are not designed to be used for other things. This may change in the future. With a different approach to spherical parametrization, they provide the best distribution for stars. Use only with *PPS_Starfield3D.shader*.

Sphere meshes may change in future builds. I also intend to provide a better workflow for generating spheres with noise. This can be used for generating asteroids, for example.

**PPS_StarSphereX and PPS_SkyCube meshes produce an error (figure 2) in the inspector. This happens when you click on the mesh itself, to look at its properties and its preview. Nothing is actually wrong, the inspector just doesn't know how to render a mesh preview because the mesh extents are set to infinity (to prevent culling).**

## 2.2 Cube - Skybox

The PPS_SkyCube is a standard cube mesh with a modified uv mapping. This mesh is used similarly to PPS_StarSphereX, which means it also gets modified (mesh culling) if used with the provided

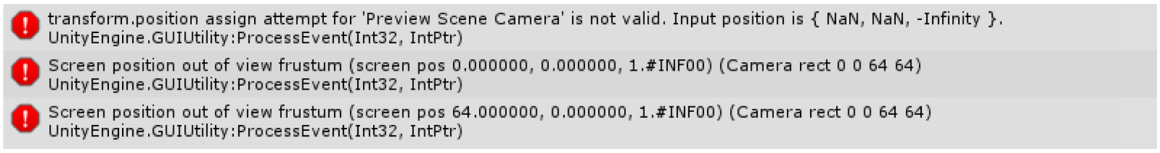


Figure 2: PPS_StarSphereX and PPS_SkyCube mesh error.

generation scripts. Unity's implemented skyboxes don't mix well with planetary rings because they are transparent and have shadows. In general there doesn't exist a 'best' solution for transparent objects with shadow support. I do believe my skybox is less efficient than the built in version, but not much less.

If you absolutely need Unity's implemented skybox, you should play with the render queue of objects, that don't display correctly. Objects with a transparent render queue don't support shadows though and I believe they are less efficient because of their reversed draw order.

### 2.3 Double Sided Plane

Another simple mesh which slightly differs from Unity's plane and quad. This mesh correctly renders from both sides and is particularly made for use with *Planetary Rings*. This mesh consists of 9 vertices and 8 triangles on each side. The main reason it can not be just a standard quad is that the *Planetary Ring shader* requires a vertex on the center.

## 3 Optimization

### 3.1 Substance Archives

There are a couple of things you can do to optimize texture generation. These settings are located at the end of every procedural material, which are created in a Substance Archive: texture width/height, texture format and Substance load behavior. If you are using the provided generation scripts, these settings can be found on the *Managers*. If they are not, then they are optimized for performance, which means they are imported at a very low resolution and compressed.

Texture generation time largely depends on texture size. I recommend using small textures if you do not need a lot of detail on the planets. By default the texture format is set to *Compressed*, but I highly recommend using the *RAW* format for less artifacts. Experiment with the import settings... Sometimes the difference is barely noticeable in which case, it is more performance friendly if *Compressed* textures are used. Performance-wise, it is better to use *RAW* $1024px$ textures than *Compressed* $2048px$ textures. They are a lot faster to generate and shaders preform faster with a lower resolution texture, despite it not being compressed.

Unity 2018 will also have the option to generate $4096px$ textures, though I think the ideal resolutions are $512px$, $1024px$ and sometimes $2048px$, depending on your use.

### 3.2 Shaders

You can modify all the shader (material) properties at run-time from scripts with a few exceptions. Some effects that you toggle on your planets can not be changed at run-time. These restrictions are mainly there for compatibility and performance reasons. More about this can be found in *CheatSheet.pdf*.

#### 3.2.1 Shadows

Shadows only work correctly with Unity's lighting system (*Light component*). If you do not want to have shadows in your game, make sure you turn off shadows in the *Mesh Renderer* component or simply set the shadow type to None under the Light component.

All options regarding shadow distance and quality can be found under the project's *Quality* settings:

$$Edit > ProjectSettings > Quality > Shadows$$

and *Light* component settings. The *PPS_PlanetaryRing.shader* also contains a slider that controls the shadow opacity it receives from objects that cast shadows.

# 4 Other Provided Scripts

- **PPS_GradientGenerator.cs**
  Used for generating gradients for planets, stars and rings. Generated gradients can be injected directly into the material or saved as assets.
- **PPS_Noise.cs**
  A collection of noise generator functions. May be removed in future versions.
- **PPS_NormalBumpMerge.cs**
  For making textures that work with the asset's shaders. Merges normal (.rg) and bump (.b) textures into a "normal-bump" texture (.rgb). Users also have the option to **auto level** the bump texture so it ranges from zero to one.
- **PPS_SphereGenerator.cs**
  Generates spheres with noise. Can be used for asteroids. I intend to make asteroid generation friendlier in the future.
- **PPS_SunRimCullingFix.cs**
  Fixes mesh culling for suns.

# 5 Contact

Please do not hesitate to contact me at tadej.slivnik@outlook.com if you have any questions, requests, suggestions or bugs to report. Every reported bug is appreciated!