Planet Saturn and its Moons Asset
V0.2

Documentation

Charles Pérois - 2015

Table des matières

1. Introduction

This asset aims to provide a simple and fast way to add realistic looking planet Saturn and its moons in your project.

The asset contains:

- A demo scene to show how to use it.
- Custom shaders for the Saturn rings, its shadows and Titan.
- A high polygon sphere.
- A set of textures.
- A sunlight example.
- A Milky Way skybox.
- Scripts, mainly for the demo scene.

This is still an early version of it, there's plenty of rooms for improvements:

- The planet shadow on the rings doesn't blend well with the rings transparent shader, It looks fine on black/dark backgrounds though, but on brighter ones, not good: transparent areas are filled with black instead of nothing..
- Planet part of Saturn uses simple diffuse shader, so edges of it are sharp. I plan to find a way of adding some alpha gradient to make it look more "gas giant" like.

2. Release Notes

V0.2
Following moons of Saturn added:

- Titan
- Enceladus
- Iapetus
- Rhea
- Dione
- Mimas
- Tethys

V0.1.3
Unity 5 version

V0.1.2
bug with the rings shadow projected on the planet fixed.

V0.1.1
bug with Saturn shadow projected on rings fixed.

V0.1
Initial release

3. How to Use

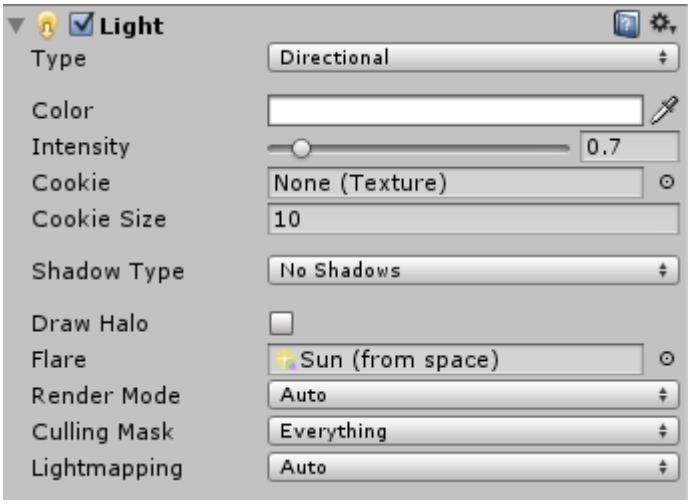
1. Set the scene

1. Set a scene with prefabs

First, copy all assets into your project.

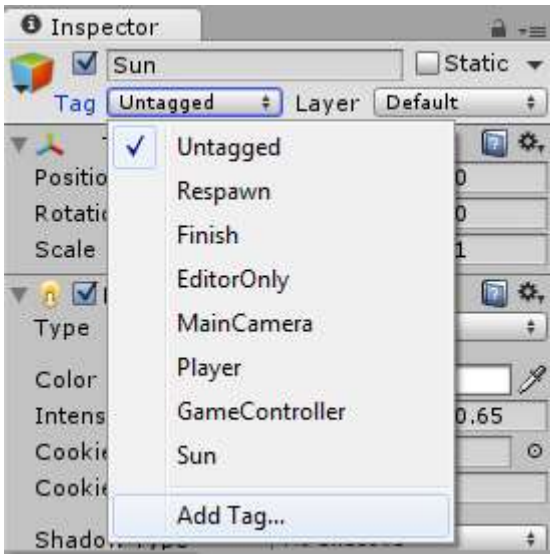
Then, if your scene doesn't contain any sun object yet (or any object that emit a light), create it.

Add a light to the Sun object (Add component > Rendering > Light), make it directional, then add a lens flare to it (one is included in "Lens Flares/" folder).

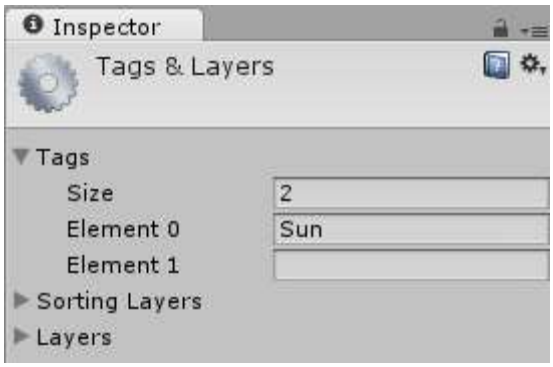


Finally, add a tag "Sun" to the sun object (or the one that emit the light). You can do it this way:

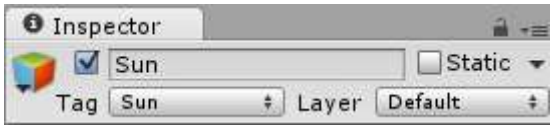
Select « Add Tag… »



Add a tag named “Sun”



And select it in the drop down list



The tag have to be “Sun”, this allows the dynamic projector shadow of the prefab to works without any setting, so that way you can dynamically instantiate it by code.

Now you just have to drag the planet prefab you want to your scene. Prefab can be found in these folders:

Planets/<Planet Name>/<Planet Name>.prefab

Newly added prefab should looks like this in the hierarchy view (Saturn example):

Prefabs can be dragged anywhere in the hierarchy, whether on the root or inside another object.

Run the scene, and that should be it!

2. Instantiate prefabs by code

All planets (or moons) present in this package can be dynamically instantiated using the provided prefabs.

The Script named GUI.cs demonstrate how to do it. You can find the script here:

Scripts/MainGUI.cs

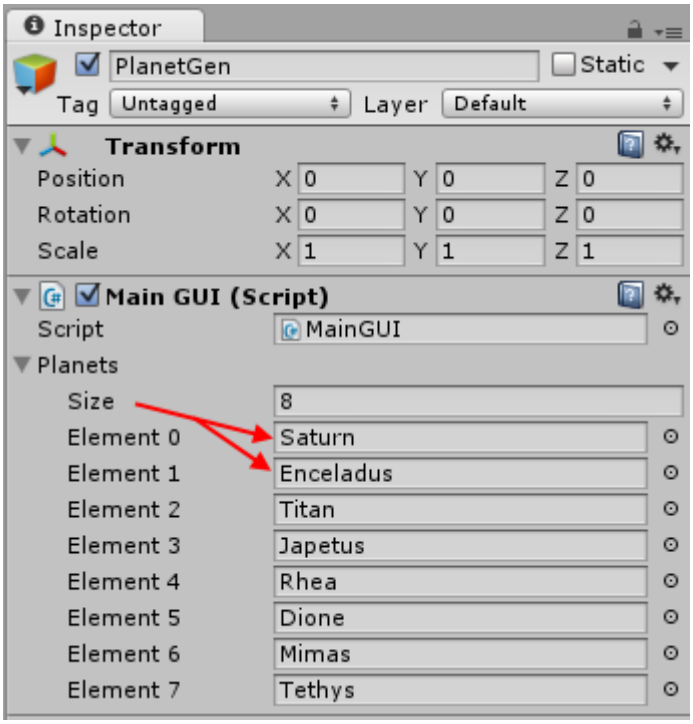
It presents one way to do it, but there are many others.

Still, if you want to make your own script here is how you can do it (using the provided script MainGUI.cs as reference):

- Create an object
- Create a new C# script (or any other language, but honestly…)
- Then you can copy paste this code in it :

```
public class GUI : MonoBehaviour {

        public GameObject[] planets;
        private GameObject displayedPlanet;
        // Use this for initialization
        void Start () {
                displayedPlanet = Instantiate(planets[0],transform.position,planets[0].transform.rotation)
as GameObject;
                displayedPlanet.transform.parent=transform;
        }

        void OnGUI(){

                GUILayout.BeginArea(new Rect(Screen.width-180, 20,150,500));
                GUILayout.BeginVertical();

                for(int i = 0; i<planets.Length; i++){
                        if(GUILayout.Button(planets[i].name)){
                                Destroy(displayedPlanet);
                                displayedPlanet =
Instantiate(planets[i],transform.position,planets[i].transform.rotation) as GameObject;
                                displayedPlanet.transform.parent=transform;

                        }
                }
                GUILayout.EndVertical();
                GUILayout.EndArea();
        }
}
```

- Attach this script to the object you created just before, or any active object in the scene.
- On the inspector view of the object, set the size of the Planets array (8 here) and drag and drop planets prefabs here :



Script explanation:

Start() method:

```
displayedPlanet = Instantiate(planets[0],transform.position,planets[0].transform.rotation) as GameObject;
displayedPlanet.transform.parent=transform;
```

Here we instantiate the first prefab contained in the `planets` GameObject array, once the object is loaded (here when the scene is loaded), at the same position and rotation as the object you attached the script to. In the second line we ask the newly created planet to be a child of that object.

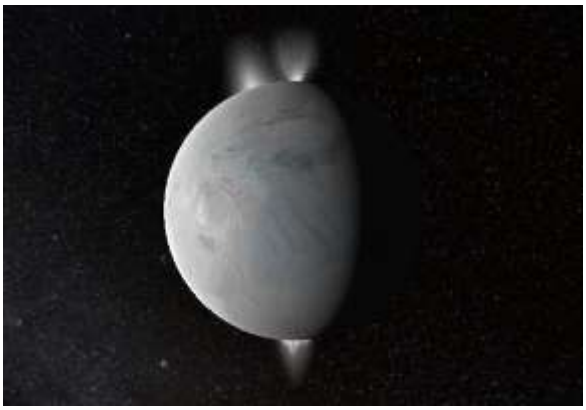
OnGUI() method :

```
GUILayout.BeginArea(new Rect(Screen.width-180, 20,150,500));
GUILayout.BeginVertical();
for(int i = 0; i<planets.Length; i++){
        if(GUILayout.Button(planets[i].name)){
                Destroy(displayedPlanet);
                displayedPlanet = Instantiate(planets[i],transform.position,planets[i].transform.rotation)
        as GameObject;
                displayedPlanet.transform.parent=transform;

        }
}
GUILayout.EndVertical();
GUILayout.EndArea();
```

This code will loop through all planets you have attached in the `GameObject[] planets` GameObject array in the inspector (once again we got only one here), then for each planet it will create a GUI button and bind some code on the click event of this button.

When a user click on a button, it will destroy the currently displayed planet and instantiate a new one.

2. Enceladus Geysers



By default, Enceladus prefab loads 3 geysers on predefined locations and those geysers will uses the billboard technique (geysers are actually luminous textures on a plane, the billboard technique allows them to "face" you wherever you/they are).

But! If these design choices doesn't make you happy, you can change them that way:

1. Geyser Selection

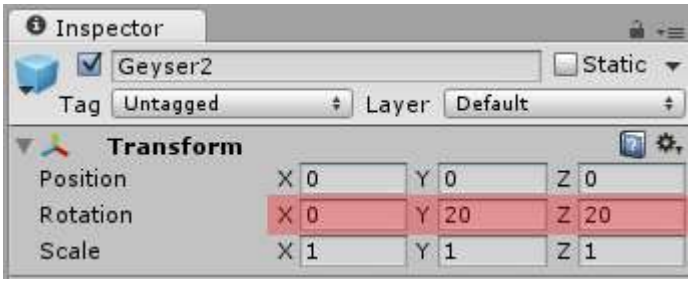
Geyser are actually prefabs which are instantiated by the Enceladus prefab on load (Inception stuff). Geysers prefabs are located in the same folder as the Enceladus prefabs (named Geyser1, Geyser2 and Geyser 3) (Planets/Enceladus):

You can set how many (or none at all), and/or which ones will be loaded on the Enceladus prefabs. To do so, select the Enceladus/Planet object (see picture above) (You can drag the prefab in the hierarchy view, and select the "Planet" object there, or select it directly within the prefabs in the project explorer view, both work, and as long as the object name is blue in the hierarchy view, both will be "connected")
Once you have select it, go to the Inspector and look for the "Instantiate Geysers" Script section:

This is the script that Instantiate the selected geysers once Enceladus prefab is loaded.
So, to choose what geyser you want to be loaded, just set the size (= number of geyser that will be loaded) of the Geyser Prefabs array, and drag the geyser prefabs you want in the Element X fields. You can add as much prefabs as you want, but don't add the same prefab 2 times, because they'll load in the same location. Which takes us to the next chapter..

2. Geyser Location

To change the location of a geyser, select its prefab, and change its Transform > Rotation values:



So, if you want to load 2 identic geysers in different locations on the planet, do this this way:

- Drag a geyser prefab into the hierarchy view, this will create a GeyserX object.
- Duplicate the just created GeyserX object: right click > duplicate.
- Change the Transform > Rotation values of this newly created object.
- Rename it if you want, then drag this newly created object to the Project Explorer.
- A new prefab has been created, so now you can add it to the "Instantiate Geyser" Script.

If you want to create a totally new geyser (with a new texture), considering you've created a new texture (the texture should have an alpha channel), just follow these previous steps, and once the new geyser is in the scene view, drag your texture on it (this way it'll create a new material automatically, and avoid that very annoying situation when you modify a material in scene view, and by doing so you also modify another object you forget about that uses the same material ☺)

3. Billboard settings

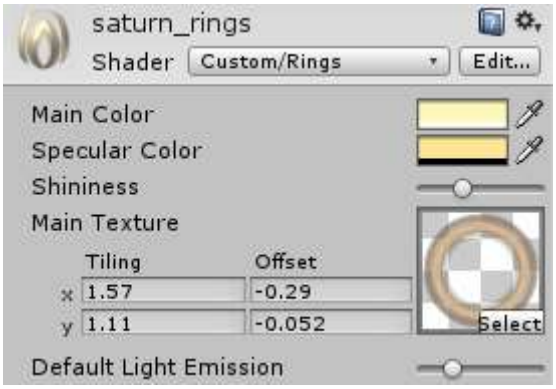
Going back 2 chapters before, on "Instantiate Geysers" section of the Enceladus>Planet object inspector:

You have notice this "Disable Billboard" checkbox.
By default, geyser textures will always "face" you (on Y axis only though). This simulate a "3d" rendering of the geysers and prevent them to look super thin on some view angles. The problem with

it though, is when the texture approaches the center of the planet, you will see it “flipping” around that center.
To avoid this, you can check this checkbox, they won’t turn at all and will stay fixed.

3. Specific Shaders

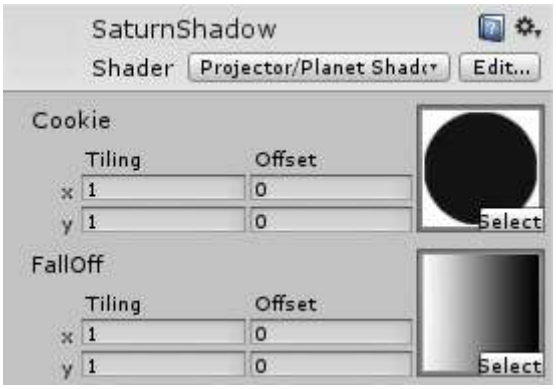
1. Saturn Rings



This is the shader used to render the Saturn rings.

Main Color: The tint color of the ring (this color will be multiplied with the texture color)
Specular Color: The color of the Sun reflection on the rings
Shininess: Move this cursor to play with the strength and size of the Sun reflection.
Main Texture: Texture of the ring (As you can see in tilling and offset settings, it was a bit tricky to make it centered ☺)
Default Light Emission: This parameter sets the light emission of the rings when they’re not directly enlightened by the Sun light. It may sounds weird, but without this forced light emission, the rings looks way too dark when the light direction tend to be parallel with the ring angle.

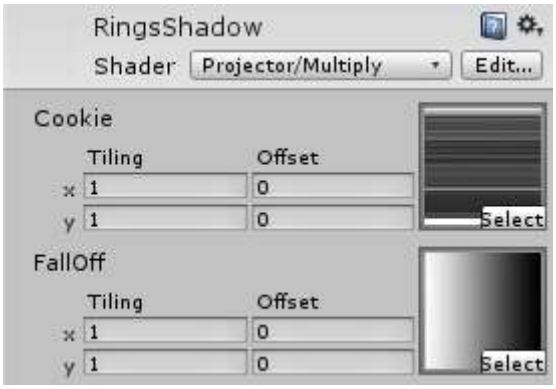
2. SaturnShadow (Projector)



This is the projector shader used to project the Saturn shadow to its rings.

Cookie: The texture that represent the projected pattern
FallOf: The gradient fall off of the projector. Here, it does nothing because of the blending problem projectors have projecting on transparent surface shaders..

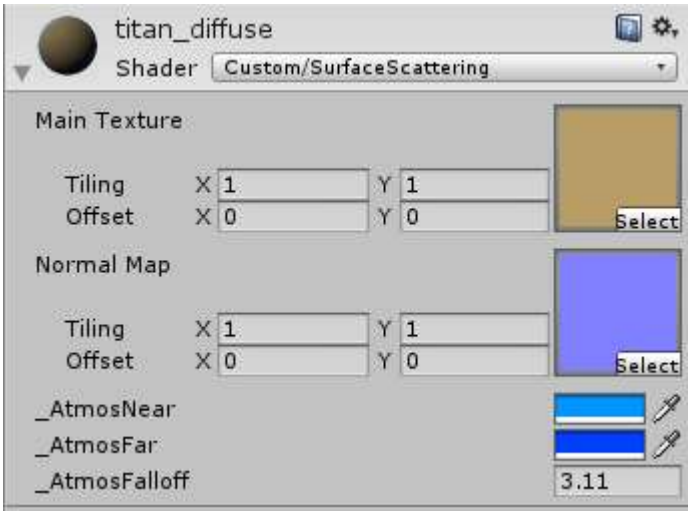
3. RingsShadow (Projector)



This is the projector shader used to project the ring shadows on Saturn sphere.

Cookie: The texture that represent the projected pattern
FallOf: The gradient fall off of the projector.

4. SurfaceScattering (used for Titan)



Main Texture: Main diffuse texture of the planet
Normal Map: Normal map texture of the planet to add some relief effect

The next 3 parameters affect the atmospheric scattering gradient effect on the surface of the planet:
_AtmosNear: inner gradient color tint.
_AtmosFar: outer gradient color tint.
_AtmosFalloff: set here the thickness of the gradient. Lower value means a thinner atmosphering gradient ring.

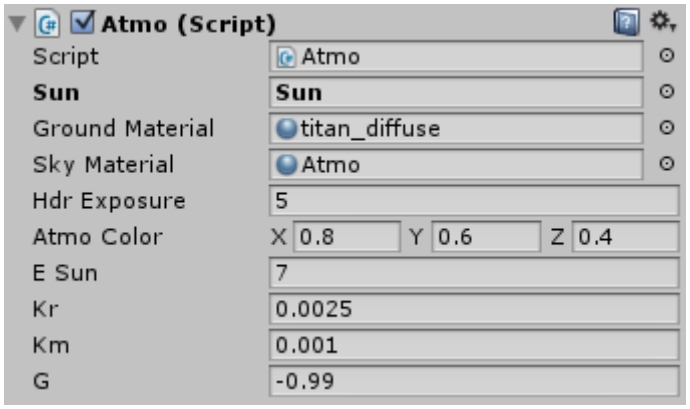
5. SkyFromSpace + Atmo.cs Script (used for Titan)

To render the Atmosphere of titan, the shader SkyFromSpace is used in conjuction with the Atmo.cs Script.

This shader should be attached to the object that contains the atmosphere sphere (the biggest one).

The Atmo.cs Script should be attached to the parent object which contains the atmosphere object.

It looks like this in inspector:



Sun: the sun object where the light is attached (not necessary if your sun has the “Sun” Tag)

Ground Material: the material that is used for the Planet object.
Sky Material: the material used for the Atmo object.
Hdr Exposure: set the brightness of the atmosphere
The “Atmo Color” parameters plays with the wave length of the scattering algorithm, thus, play with will affect the color of the atmosphere.
The other parameters shouldn’t be touched, if so it’ll shift various effects in a not realistic way.

Be warned, the atmospheric effect doesn’t appear in the editor view (or barely appears but looks like crap), you have to run the scene to see it. So you should play with the value while the scene is running to see effects. Examples:

4. Other Scripts

There are some scripts I haven't talk about in this documentation, here's theirs purposes:

- SaturnRingShadow.cs: Handles how the rinds shadows projector projects its texture on the planet.
- SaturnShadows.cs: Handles how the planet shadow projector projects its texture on the rings.
- CameraZoom.cs: Allow to zoom-in and zoom-out usin the scroll wheel. Attach it to the camera.
- EarthMovement.cs: Allow any object to automatically rotate.
- RotateCamera.cs: Controls most of the camera "flyby" commands.
- RotateEarth.cs: Allow users to rotate a planet with the right mouse button or the arrow keys. Attach it to any planet object.
- RotateSun.cs: Allow users to rotate the sun light with the left mouse button or the Q-D keys. Attach it to the "Sun" object.
- GeyserLight.cs: Handles the luminosity of Enceladus geysers based on sun light direction and angles.

3. Credits

Saturn Texture:

Celstia Motherlode - Runar Thorvaldsen

http://www.celestiamotherlode.net/catalog/saturn.php

Saturn Rings Texture:

By Alpha-Element - Deviantart

http://alpha-element.deviantart.com/art/Stock-Image-Saturn-Rings-393767006

Dione Texture:

http://photojournal.jpl.nasa.gov

Enceladus, Iapetus, Tethys, Rhea and Mimas Textures:

By Steve Albers

http://laps.noaa.gov/albers/sos/sos.html

Titan: I actually made Titan all by hand.

Skymap textures uses a photograph from the European Southern Observatory (ESO) / S. Brunier:

http://commons.wikimedia.org/wiki/File:ESO_-_The_Milky_Way_panorama_(by).jpg