Simple Procedural Skybox

A Procedural Shader designed for creating Skyboxes in Unity

V2.0

Introduction to This Package

Simple Procedural Skybox is a Package that procedurally creates a skybox for Unity. It allows you to create physically based skyboxes that can change on the fly using mathematical formulas from *A Practical Analytic Model for Daylight* Paper reference (Link in the header of `Preetham` Shader).

About the Scene Example

You can access a scene example by navigating to `Assets/ProceduralSkybox/Example/Sky.unity`. This example scene contains a preview about the sky shader used in this scene.

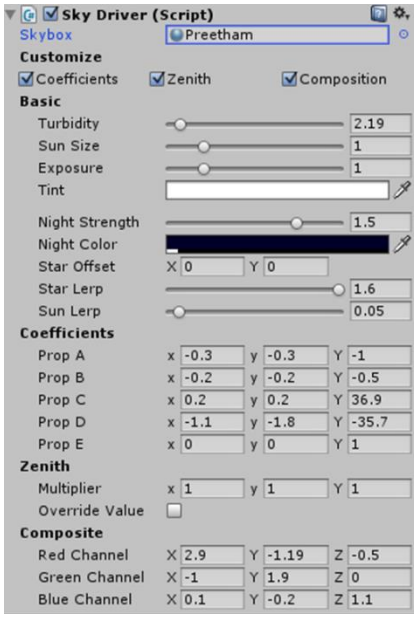
How to Apply the Component

You can create a new material then select shader located in `Skybox/Preetham` then attach it your scene skybox, then attach `SkyDriver` component to an object that has directional light in your scene. `SkyDriver` will automatically pick up the material from scene skybox.

Please note that you **don't** have to create a new material for each of your scene since their values will always be overwritten by `SkyDriver` (and assuming you only opens one scene in the editor).

The Component Properties

About The Shader



The shader properties itself are quite complex to be understood, so attaching `SkyDriver` to your scene is important, especially if you don't know how this shader work in detail. Most of this documentation covers how to edit by `SkyDriver`, not directly by shader itself. So here are the sections of `SkyDriver`.

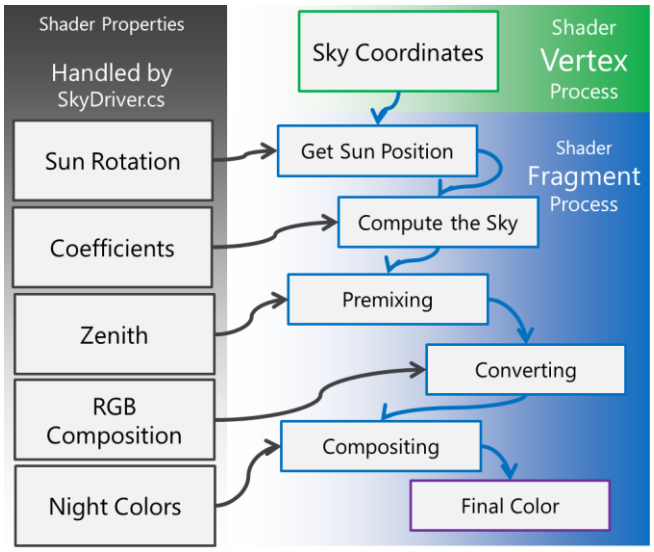
At the very top of the Component, **Skybox** represents the material Target. The **Customize** section offers if you want extend the customizations of the sky. And in the **Basic** sections covers the main part of the sky. **Coefficients**, **Zenith**, and **Composite** sections are visible when you enabling one of the Customize Checkbox in the top.

These Properties will be explained in detail in the next section of this documentation.

In Depth Explanation about How it Works and Customize It

This section covers how the Shader can make colors can fit into our eyes.

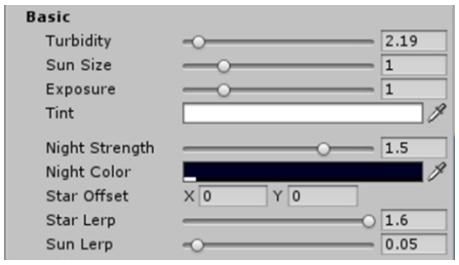
Pre-Depth: The Short Summary



There are many processes inside of this single `Preetham` shader. You can see the whole processes in image at the left side of this paragraph are all computed in Shader Fragment Program.

Note at shader properties in inspector there are lot of properties that need to be filled correctly. So for that reason we need to attach a script component called `SkyDriver` to our Directional Light so the sky can be synchronized with Directional Light Rotation.

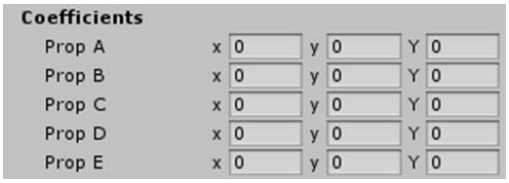
Pre-Depth: The Basic Properties



The Basic Section covers the basic part of this skybox. **Turbidity** is a property that will always be a reference if you doesn't do any Zenith and Coefficients Customization. **Sun Size** determining how big the Sun it is, or make it zero to make it not visible. **Exposure** determine how bright the sky, and **Tint** make sky are more tinted.

Pre-Depth: Color Space between CIE vs. RGB

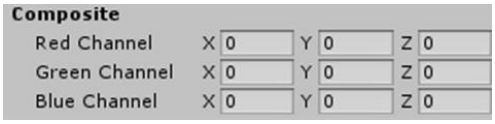
The main computation requires a special color values named CIE 1931 color space. And because of that we need to understand how this CIE color space work.



Coefficients are all in xyY values, while editing these values, make sure adjust the Y (luminance) first, then you can adjust the rest

The CIE xyY color space is work differently from RGB, it works based on our perception on light wavelengths and:

- the x and y value gives you a chromaticity (colorfulness) value
  - higher x meaning redder it become (or more blue if lower)
  - higher y meaning greener it become (or more pink if lower)
    - Beware to not make this value negative, or the color will broke.
  - Change these values only if you need change their color.
- Y value gives you a luminosity (brightness) value.
  - Higher Y means brighter the color (or darker if lower)
  - Change this Value if you need to change the intensity



All values in Composite are for determining how much XYZ values affected with each RGB Channel, the values can be positive (for additive conversion) or negative.

CIE xyY will be converted to XYZ Color value before ending up in RGB Value. The XYZ Color is much like RGB Value, but **we can't** describe which is red, green, blue (just say it's an alien color), so we take the following equation to converting from XYZ into RGB Color:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{pmatrix} R_X & R_Y & R_Z \\ G_X & G_Y & G_Z \\ B_X & B_Y & B_Z \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

The R_x, R_y … (etc.) are coefficients that are needed for changing into RGB Color. The default value is for sRGB Color Space Model which is used in most Screen Displays. You can see more values for different models here.

Pre-Depth: The Fragment Program

The Skybox Shader in Unity works by determining colors in every directions of the sky. This was done in Fragment program, while in the vertex program, we get the coordinate data from "sky vertices" (yes, skybox is just like kind of special mesh that bound infinitely in our scene).

So, in a simple word, the shader will **repeat** the Fragment Program **over and over again** for each coordinate that visible in scene Camera until there is no hole to draw anymore. And Amazingly in Unity this process can be done almost **instantly**.

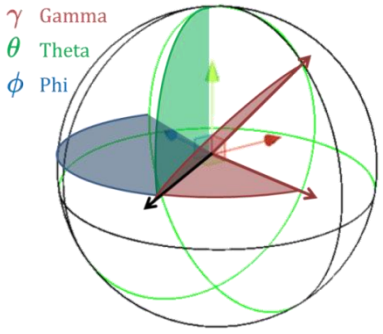
So, below here, are steps that run on the Fragment shader, and how they can return a nice color for each direction of the sky.

1. Determining Coordinates and Sun Spots

At the starting computation in fragment shader, three variables are created and used, that is:

- **Phi** For defining X Coordinate of sun
- **Theta** For defining Y Coordinate of sun
- **Gamma** For defining the angle between sun and coordinate that we want to draw

These variables are all in radians. **Phi** and **Theta** are declared in shader properties, while **Gamma** is pre-computed since this variable is unique for every direction of scene skybox.



The Skybox Dome. Here, the Black Vector is the direction of the sun, while the Red Vector is a Vector that we want to draw. Three variables are created for each direction of the sky

2. Pre-computing Coefficients

If you were enabling customization for Coefficients and Zenith, you can see there are more properties that you can edit using your own value. By default, if you disable these customization, they will filled by default value generated by turbidity value.

The **Prop A, B, C, D** and **E** are coefficient values that are needed for drawing skies with Perez Algorithm (See the next step). By default these properties are filled depending on Turbidity value (and you can see how it work by see `SkyDriverHelper.cs`)

Now here's the difficult topic, as you see when customizing zenith value, you can see two properties, **Multiplier** and **Override Value**. See step number four for more information.

3. Draw with Perez Algorithm

The shader will start to paint in xyY Color space. The skybox painting was done in mathematical formula. *Perez, Yu et al.* give us an idea that this painting process can be done with this function,

$$Y_P = Y_Z \frac{f(\phi,\gamma)}{f(0,\phi)} \quad \text{Where} \quad f(\phi,\gamma) = (1 + A\, e^{\frac{B}{\cos(\phi)}})(1 + Ce^{D\gamma} + E\cos^2(\gamma))$$

The Y_P variable represents each color channel, while ϕ and γ symbol represent `Theta` and `Gamma` variable that we'd calculate it earlier. The Y_Z variable represents `Zenith` that will act as Master Channel (or, global multiplier) in each color channel, and the denominator $f(0,\phi)$ since only contains zero and ϕ we can calculate it outside of the fragment program (ie. it will calculated along with zenith calculation).

Notice A, B, C, D, E variable (not small e, which is Euler Number), these variables are represent of **Prop A, B, C, D** and **E** with each representing color Channel (ie. If Y_P is Y channel, then A would be Prop A at Y too). These Props have special meanings which are:

- **Prop A** represents darkening or brightening of the horizon
- **Prop B** represents luminance gradient near the horizon
- **Prop C** represents relative intensity of the circumsolar region
- **Prop D** represents width of the circumsolar region
- **Prop E** represents relative backscattered light.

You can make yourself understand how they works by adjusting the luminance (Y Channel) for each Props (Adjust value by dragging the channel label is the easiest way to do), before going further with adjusting their chromaticity.

4. Premixing with Zenith

Zenith is a coefficient that simple yet very useful. It controls the exposure of each color channel. By default, Zenith will controlled by **Turbidity** and **Theta** (Y Coordinate of sunspot). Zenith is responsible for changing sky colors based on sun direction (ie. giving a redder color at sun near the horizon). Zenith also returns zero exposure when the sun is beneath the Horizon (giving dark, black color at return).

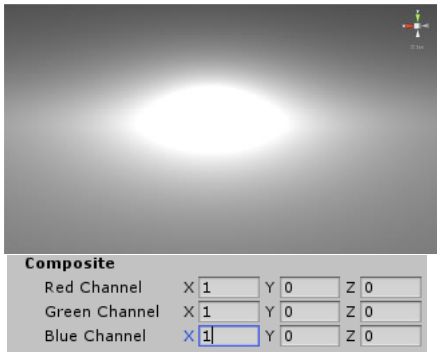
Unlike Coefficient, Zenith can be adjusted by changing their **multiplier** while letting default calculation still being used, and the result will multiplied together. But if you want to omit the default calculation you can check **Override Value**, but it's **not** recommended since the sky will stick with same ambient color regardless of the sun direction.

Another cool thing is **Zenith Clamp**. It's determining how much the **minimum** value of Allowed **Theta.** For example if you set it to maximum value (half of Pi), then the sky ambient will stick at constant value until the sun is beneath the horizon.

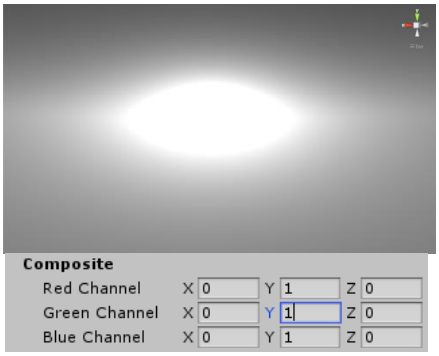
5. Converting CIE xyY to RGB

After these Algorithms have been calculated, the resulting CIE xyY Model has to be converted into RGB Color Space before we take the last step. Remember what we said earlier that xyY model will be converted XYZ Color Space. Take a look of brief data in XYZ Color Model (with turbidity 2.5 no Customization):

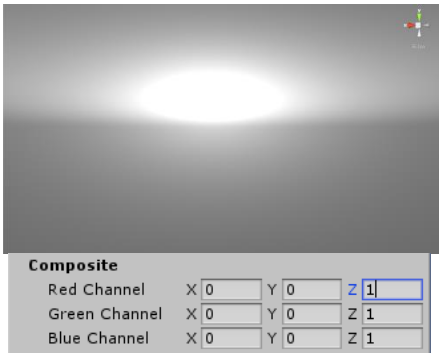
X Channel



Y Channel



Z Channel



X and Y are look similar, but be careful it's not exactly similar image! Call pixel detective to proof it :)

By default, these properties will be filled with default value (as we said, sRGB Constant, see `SkyDriverHelper.cs`), but if you want a different mood, you can always customize it together with coefficients to get a pleasant look.

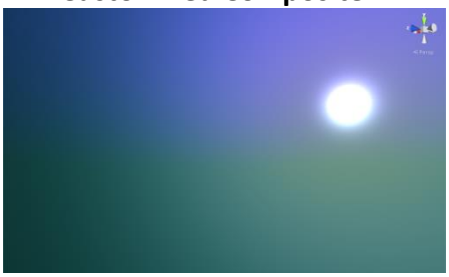
Original Composite

Original Props

Customized Composite

Original Props

Customized Composite

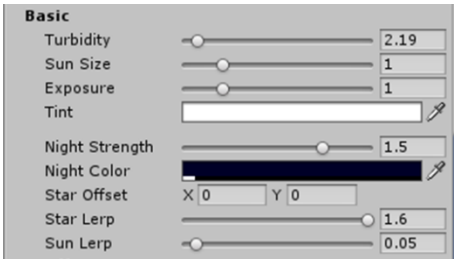


Customized Props

6. Compositing with Night Backgrounds

At correct values it will result in nice looking skies over night (And a Meteor!)

Now here's the last touch. When the sun goes beneath the horizon, the sky will be dimmer. But instead of full of blackness, the sky can be added with stars in the background.

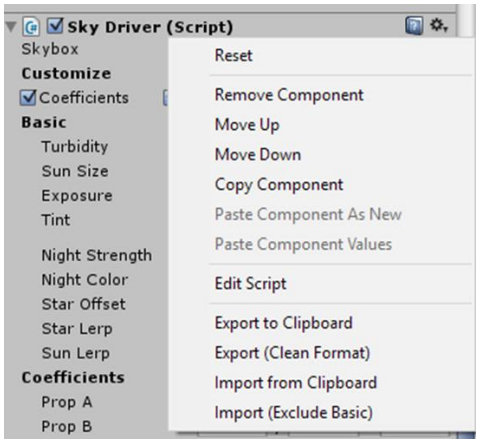


Here, 5 properties created for describing what we will draw when night has come. The **Night Color** is the **minimum** color for ground and horizon glows. The **Night Strength** is property that determine how the glows interpolation in the horizon sky, Zero mean plain (ie. Sky color will same like ground color), and the higher value means shorter the glows, this also make the stars look brighter.

Star Offset is not meant to be used in editor, but it can be interpolated in runtime. This property makes stars appear move when you change it additively. **Star Lerp** determines when the star visible on the sky, higher the value meaning the stars will appear more soon even the sun is on the horizon. **Sun Lerp** works similarly, if it set to zero, the sun will **instantly** disappear as soon as it beneath the horizon.

Import & Export Customization and Some Cool Templates

Import & Export Mechanism



If you click at the gear box, you'll see some functions in the bottom. SkyDriver has an Export and Import feature which allow us to create, share, backup, and revise all data properties in the component.

When you click **Export**, the data will be copied to your clipboard as a **Plain Text**, so you can paste it everywhere outside of Unity Engine. There is export with **Clean Format** which is meaning the data will be rounded until one behind decimal for readability across many people.

If you want to enter the data to the component, simply copy the **whole** data (you can't copy partial text that are generated before!). Then click **Import**. If you want to keep the Basic Settings (including check box customizes), simply select Import with **Exclude Basic.**

Here Some Cool Templates available to try.

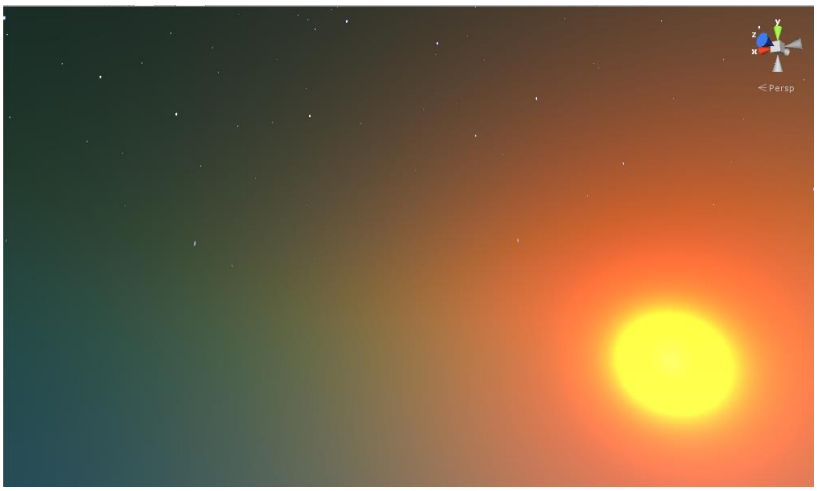
Copy these numbers and paste it in the SkyDriver to see it in action.

"The Darkling Sky"

```
      7,1.7,1.0,
1.0,1.0,1.0,1.0,
2.0,0.0,0.0,0.3,
0.0,0.0,1.6,0.1,
          0,0.9,
 -0.3,-0.2,-1.0,
 -0.2,-0.2,-0.5,
   0.2,0.2,36.9,
-1.1,-1.8,-35.7,
    0.0,0.0,1.8,
    0.4,0.4,0.3,
    1.0,1.0,1.0,
  2.9,-1.5,-0.5,
   -1.0,1.9,0.0,
    0.1,-0.2,1.1
```

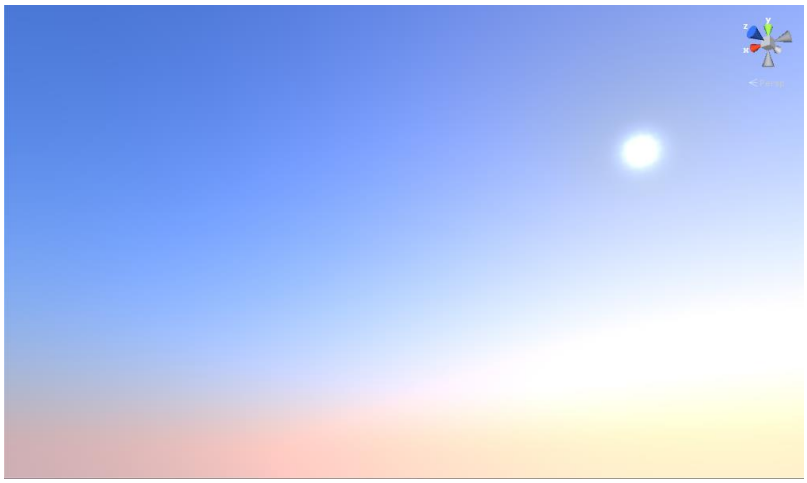
“Red-Burning Sun”

```
      7,1.7,1.0,
1.0,1.0,1.0,1.0,
2.0,0.0,0.0,0.3,
0.0,0.0,1.6,0.1,
           0,0.9,
 -0.3,-0.1,-0.9,
  0.0,-0.2,-0.6,
   1.0,0.2,37.6,
-1.1,-1.8,-27.1,
    0.0,0.0,1.5,
    0.3,0.3,0.2,
    1.0,1.0,1.0,
  2.9,-1.5,-0.5,
   -1.0,1.9,0.0,
    0.1,-0.3,0.8
```



“Small But Mighty”

```
      6,3.4,1.0,
1.0,1.0,1.0,1.0,
2.0,0.0,0.0,0.3,
0.0,0.0,0.6,0.1,
           0,0.53,
 -0.3,-0.3,-1.0,
 -0.2,-0.3,-0.6,
    0.2,0.2,4.6,
-1.1,-1.8,-75.1,
    0.0,0.0,0.7,
    0.3,0.3,0.7,
    1.0,1.0,1.0,
  3.2,-1.5,-0.5,
   -1.0,1.9,0.0,
    0.1,-0.2,1.1
```



About This Package

This package is provided in Asset Store published by Wello Soft.

Website | Asset Store | Email