

NUAJ' v1.1

The Weather Simulator



User Documentation v1.1

By Patapom

1. Contents

1. Contents	1
2. Installation	4
3. Presentation	4
4. Quick Setup	5
5. Main Module	6
5.1. Sun Parameters	6
5.2. World Parameters	8
5.3. Shadow Map Parameters	11
5.4. Local Variations Parameters.....	13
5.5. Lightning Parameters	15
5.6. Luminance Adaptation Parameters.....	15
5.7. Tone Mapping Parameters	17
5.8. Glow Support.....	19
5.9. Render To Cube Map.....	20
6. Sky Module	20
6.1. Sky Model	20
6.2. Standard Parameters.....	21
6.3. Rendering Parameters.....	24
6.4. Low Altitude Fog Layer	26
6.4.1. Layer Parameters.....	26
6.4.2. Appearance Parameters	27
6.4.3. Noise Parameters	30
6.4.4. Animation Parameters.....	30
7. 2D Clouds Module	31
7.1. Appearance Parameters.....	32
7.2. Noise Parameters	34
7.3. Animation Parameters	36
7.4. Advanced Parameters	36
8. 3D Clouds Module	38
8.1. Shadow Map parameters	39
8.2. Appearance Parameters.....	40
8.3. Noise Parameters	44

8.4.	Animation Parameters	45
8.5.	Advanced Parameters	45
8.5.1.	Directional Parameters.....	45
8.5.1.	Isotropic Parameters	46
8.5.1.	Isotropic Contributions.....	47
8.5.1.	Phase Parameters.....	49
9.	Satellites Module.....	49
9.1.	Generic Parameters.....	50
9.1.1.	Simple Parameters	50
9.1.2.	Revolution Plane.....	50
9.2.	Nearby Star Satellite.....	52
9.3.	Planetary Body Satellite	53
9.3.1.	Standard parameters.....	53
9.3.1.	Night Light	53
9.3.2.	Lighting Simulation	54
9.4.	Stellar Background Satellite	55
9.5.	Custom Rendering	56
10.	NuajTime	56
11.	Prefabs.....	57
11.1.	MapLocator	57
11.2.	LightningBolt.....	58
12.	Orchestrator	61
12.1.	Time, Date & Location	61
12.2.	Weather Control.....	62
13.	Water Reflection	63
14.	Rendering Pipeline	65
14.1.	Main Pipeline.....	65
14.2.	Post-processing Pipeline.....	66
15.	Performance	66
15.1.	CPU Read Back.....	66
15.2.	Software Environment Rendering	67
15.3.	Manual Luminance Input.....	67
15.4.	Upsampling Techniques	68
15.5.	Increasing Sky Rendering Performance.....	68

15.6.	Layer Clouds Performance.....	68
15.7.	Volume Clouds Performance.....	68
15.8.	Satellites Performance	69
16.	Troubleshooting	69
17.	Acknowledgments	70

2. Installation

Nuaj' comes as a ZIP archive containing the Unity Package to import and this PDF file.

- 1) Open Unity and create a new project or open an existing project.
- 2) Right-click the « Project » window and select « *Import Package > Custom Package...* ».
- 3) Select the *Nuaj'* unity package file that was in the ZIP archive.
- 4) Unity should import the assets and recompile both script libraries and the new *Nuaj'* shaders.

NOTE: Ensure there was no error in the console during import or compilation, otherwise *Nuaj'* may not work properly on your system. If you locate errors, please refer to the troubleshooting section at the end of this document.

3. Presentation

Nuaj' provides both realistic rendering and the flexibility of customization that is necessary in the video games industry. Indeed, designers and artistic directors usually don't pay much attention about a tool that is "physically accurate" and much rather prefer something that fits their own needs and is highly customizable.

Nuaj' is a powerful and complete weather simulator that can handle up to 4 different atmospheric layers like 2D clouds, 3D volumetric clouds or low-altitude fog, each layer taking into account a directional light source representing sunlight, lightning bolt linear light sources for realistic storm simulation as well as an optional artificial lighting from below (city lights).

All rendering in *Nuaj'* is performed in HDR (High Dynamic Range) and many –more or less clever– tricks are used to adapt HDR rendering to Unity, which is not yet capable of performing HDR rendering (by the way, *Nuaj'* is compatible with both LDR and HDR rendering from Unity, if the HDR rendering option comes in someday).

Due to the existence of many parameters, *Nuaj'* can be very complex to put into use but a fair amount of helpers (**Orchestrator**) and tutorials exist to teach you how to use it at its maximum efficiency. *Nuaj'* also comes with many solutions and algorithms you can choose from to make it very scalable and adaptable to most platforms.

Nuaj' is centered about the **NuajManager** script which is attached to the *Nuaj'* prefab Game Object. The manager controls everything and is composed of "modules":

- **Sky Module**, responsible for sky and low-altitude fog display.
- **Layer Clouds Module**, responsible for the display of 2D layered clouds.
- **Volume Clouds Module**, responsible for the display of 3D volumetric clouds (the real deal !).
- **Satellites Module**, responsible for the display of background satellites like the Sun, the Moon or even the stellar background.

External to the manager, you will find the following prefab objects in the “Prefabs” directory (cf. the Prefabs section) :

- **MapLocator**, is used to setup a planar map that helps driving specific parameters in **Nuaj'** like local density, artificial lighting and local fog density
- **LightningBolt**, is used to place lightning bolts in the scene that can be assigned to the manager for lightning simulation during a storm.

Finally, since **Nuaj'** has many parameters (!!) that can sometimes be hard to tweak correctly, I have included a script called the “Orchestrator” (also found in the “Prefabs” directory) that can help you to setup a complete full-blown storm using a single slider (cf. the Orchestrator section).

4. Quick Setup

The fastest way to have **Nuaj'** up and running is to load the example scene provided with the package.

Otherwise, if you want to start from scratch, here are the 8 quick steps to make **Nuaj'** operational in your project:

- 1) Load or Create some scene
- 2) Select the main camera and clear the background to black !
- 3) Drop the **Nuaj'** prefab into the scene (the prefab can be found at the root of the “Nuaj” folder after importing the Unity package)
- 4) Select the **Nuaj'** prefab in the scene
- 5) Using drag’n drop, assign the main camera as the “Drive Camera” property
- 6) Create a directional light (“Main Menu > Game Object > Create Other > Directional Light”)
- 7) Using drag’n drop again, assign the directional light as the “Drive Sun” property

From this point on, you should already have a nice blue sky and some clouds.

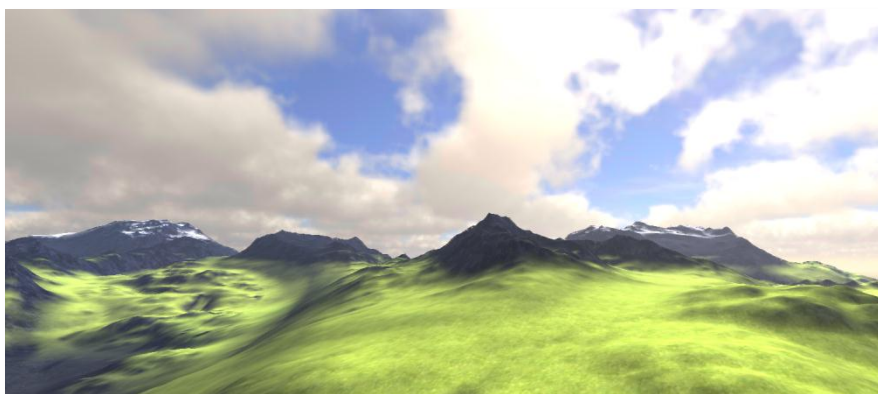


Figure 4-1– Your typical daylight in Unity

- 8) Go to the “Prefabs” folder and drop the “Orchestrator” prefab into the scene. You can then use the orchestrator to drive the Sun & Weather, all using a simple slider.

And you're done ! I know I can't prevent you from playing around with the nice parameters now... ☺



*Figure 4-2 – Later this afternoon...
(I think you can forget about that BBQ)*

But when you're done playing, let's see a description of the individual modules anyway, starting with the Main module controlling the Manager itself.

5. Main Module

In the Main module, you will find all the parameters that are global to the atmospheric rendering.

The most important parameter is the “**Drive Camera**” that tells which camera *Nuaj'* should use for rendering.

Once you assign a Drive Camera, *Nuaj'* will automatically attach the camera with an Image Effect called “**Compose Atmosphere**”. It's this effect that is responsible for triggering the rendering of clouds and the composition with the scene.

NOTE : Changing the Drive Camera can be expensive, especially if the new camera has a different resolution than the last one ; this will indeed trigger a rebuild of all the internal buffers used in *Nuaj'* !

5.1. Sun Parameters

They control the Sun, which in turn controls the entire lighting of the scene and atmosphere.

Elevation and Azimuth

They control the angles of the Sun in the Sky. Elevation is the vertical angle from a noon Sun (top) to a totally antipodal position. Azimuth is the horizontal angle that moves the Sun about the horizon.

Nuaj' is driving

You can specify a Game Object that represents the Sun. A special attention is paid when the Game Object is a directional light since **Nuaj'** can also drive the light's color and intensity.

In addition to the Sun parameters, you also have 3 options in the form of check boxes :

- **Drive Direction**, if checked then the Game Object's direction is updated by **Nuaj'**. If unchecked, your Game Object drives the Sun's direction in **Nuaj'**.
- **Drive Color**, if checked then the Game Object's light component is updated with the Sun color used by **Nuaj'**. If unchecked, your Game Object is unaffected (but it does NOT control the Sun color or intensity in **Nuaj'**)
- **Drive Ambient**, if checked then the Unity ambient light ("*Main Menu > Edit > Render Settings*") is updated with the sky's ambient color computed by **Nuaj'**. If unchecked then the ambient color is unaffected (and it does NOT control the sky color in **Nuaj'** either).

Sun Color / Intensity

This is simply the color of the light source and its intensity. Although you may somehow believe the Sun is yellow, this is entirely due to atmospheric effects. This color here is the color of the Sun in space which radiates across the entire spectrum, so the correct natural value for the Sun color is white.

About the intensity though, it does not really matter how much intensity you put for the Sun since all rendering is performed in HDR and is later adapted to be displayed in LDR for Unity, it's really up to you if you want to change it but that will not have the dramatic "burn" effect you would expect, you should rather play with the maximum luminance values we will see later...

Software Environment

This is a very important parameter. Basically, to get a nice coherent appearance, **Nuaj'** renders a tiny environment map that is used to light the clouds and setup the ambient term. This option only specifies how you would like to compute the environment.

Software computation will emulate hemispherical sky light but won't take cloud coverage into account, yielding a blue-ish ambient lighting even in very overcast conditions so it won't be as realistic as hardware computation.

Hardware computation, on the other hand, takes the same algorithms used to render the sky and the clouds and packs all this into a single value that is much more accurate and correct. The downside is that we need to read back a 1x1 texture with the CPU and, although all precautions have been taken to prevent stalling the GPU, this solution still eats a lot of time in Unity (cf. the CPU Read Back section). The best way to decide is yet to profile both renderings and see which suits you the most...

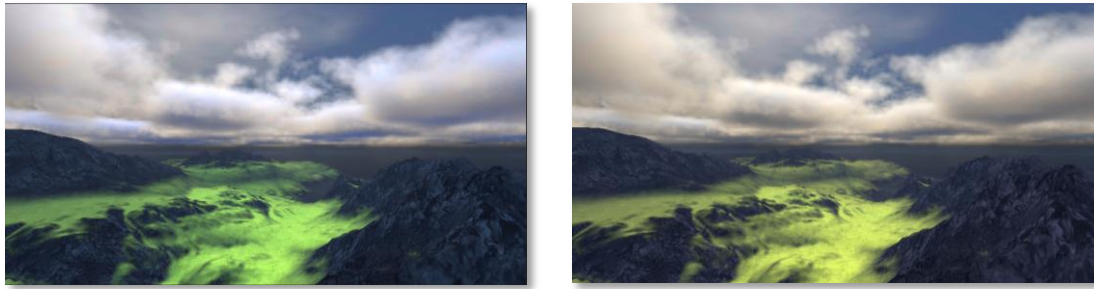


Figure 5-1 – The typical blue-ish tint of software environment rendering (left) versus the warmer and more accurate tone of hardware rendering (right)

Sunset Angles

These 2 angles (*Sunset Start & End*) help you specify when the sunset should occur. If the Sun's elevation is greater than the start angle, then the “night mode” is starting to appear until the elevation is greater than the end angle, in which case we are fully in night mode and the Moon may become the primary source of light if chosen so (cf. chapter 9.3 about Planetary Body Satellites).

5.2. World Parameters

An important notion to keep in mind with *Nuaj'* is that everything is rendered inside a system of spherical layers, as shown in Figure 5-2.

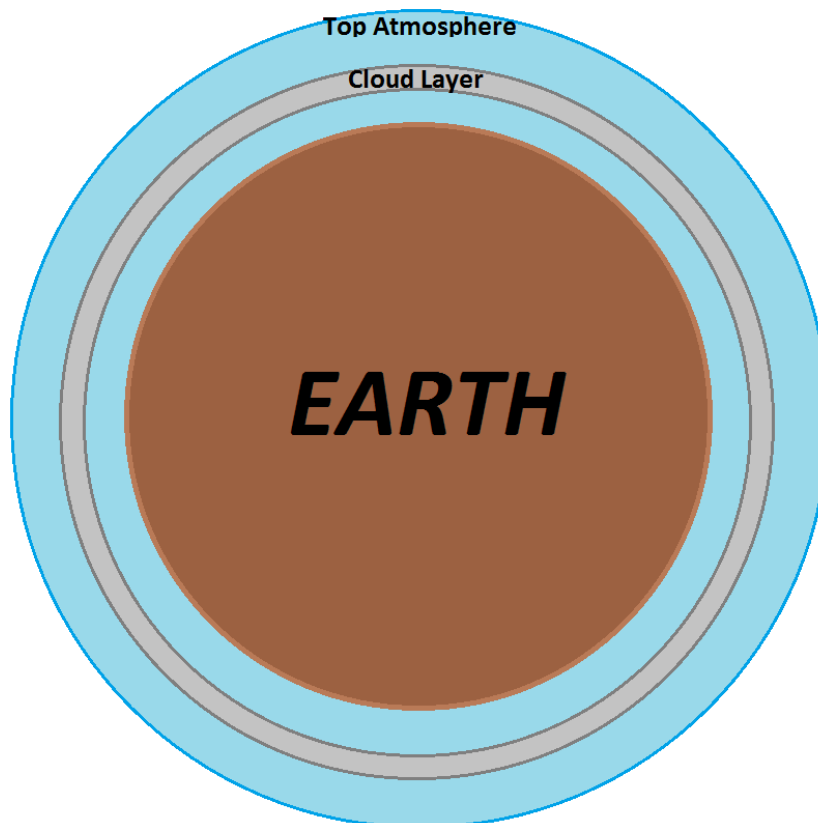


Figure 5-2 – The rendering system in *Nuaj'* uses spherical layers.

The « world » we stand in, in **Nuaj'**, is a real planet with a center and a radius.

Any subsequent layer we are going to talk about, whether it's a slice of sky or a fog layer or a cloud layer is mainly constrained by 2 parameters: its altitude from ground level and its thickness.

The first "layer" we need to define is the planet itself of course. It obviously has no altitude parameter but has a thickness described by its radius.

World Scale

This is a very important parameter as it tells **Nuaj'** how many kilometer a Unity world unit represents. After some measurements I found that a value of 0.01 is quite correct for a scene like the Bootcamp demo (this means that 1 Unity unit = 0.01km or 10 meter) while for large terrains, you would decrease that to 0.005.

The look of your scene can change dramatically because of this parameter so it is a very important to get right from the very beginning, it's not the kind of parameter you change right before shipping a game !

NOTE: Unless specified otherwise, all units in **Nuaj'** are in kilometer.

Planet Radius

This is the radius of the planet.

Be aware that if you change this, you also need to change the center of the planet by script (not displayed in the editor) to make your position match (cf. [API Manual](#)).

Indeed, I initialized the center of the planet exactly 6400km below the camera's feet to match the 6400km radius used by default. This makes us stand on the ground but if you set the radius to, say, 6500km then we will stand 100km below the surface, which is not good (cf. see what happens in [The Core](#) !).

Horizon Offset

This represents the negative altitude offset where you want to place the horizon line.

Indeed, although the horizon should stand at altitude 0 (i.e. sea level), sometimes it's better to make it go below the planet's surface, especially if you have a body of water exactly at altitude 0: the horizon may show and flicker a little, which can be annoying so make sure you make it disappear with that parameter.

Atmosphere Altitude

This is the altitude to the top of the atmosphere. Go above this and you're in outer space.

The default value is 60km.

Upsampling & Refinement

Many rendering processes in **Nuaj'** take place in downsampled buffers as you will see. We need a way to upsample them back properly to the actual screen resolution.

There are currently 4 ways to do so:

- **ACCURATE**, this is the most accurate way (hence the name) but also the most expensive performance-wise. Indeed, this method checks the discrepancy between the computed distance and the actual scene distance, and if the error is too large then a full recomputation is triggered for that pixel.
This yields nice sharp edges on the scene and is obviously the preferred solution but be warned that if too many pixels need to be recomputed then this can rapidly become very expensive! This is especially the case when the scene features many jagged edges like with vegetation for example (cf. the *ZBuffer Threshold* and *Show ZBuffer Discrepancies* parameters below).
- **SMART**, this method tries to do much with very little.
It attempts to guess which pixel is the best for a pixel whose Z discrepancy is too large, in which case it will sharply default to that best guess pixel. Otherwise, if all pixels are equal, it will use a standard bilinear interpolation. It has the advantage of both keeping sharp edges and not triggering any further computation so it's not an expensive method.
It usually performs quite well as long as you don't have tiny flickering details
- **CUTOUT**, the idea here is to simply cutout the *Nuaj'* rendering with the scene as long as the scene stands in front.
It is kind of simple but it does the trick, at the expense of a slight difference in rendering since the *Nuaj'* rendering no longer takes the scene's depth into account.
This is not an expensive method and is quite acceptable as long as clouds stand in the distance and behind the scene.
- **BILINEAR**, this is the simplest and cheapest method.
It simply upsamples the rendering by interpolating the values in a bilinear fashion. This, of course, is quite fast but has the downside of a poor rendering precision.

When selecting the **Accurate** method, you can specify the threshold above which pixels are recomputed using the **ZBuffer Threshold** parameter. This indicates a difference in depth values in Unity's world units, not kilometers this time.

As a visual helper, you can enable the **Show ZBuffer Discrepancies** option that will show you the pixels that get recomputed. Red pixels are refined sky pixels while green pixels are refined cloud pixels. If too many red or green pixels appear, this is usually a good sign that you should definitely opt for a refinement technique other than Accurate !

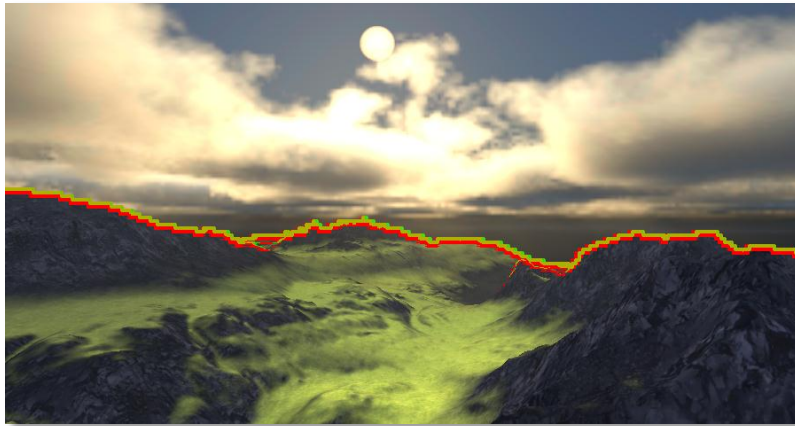


Figure 5-3 – Your typical refinement horizon line.

When selecting the **Smart** method, you can specify the importance the difference in Z (DeltaZ) plays in the computation of the pixel weight using the **DeltaZ Factor** parameter. You can also tweak the weight of the smart upsampling technique using the **Smart Weight Factor** parameter that will interpolate between no weight at all (standard bilinear) to full weight (technique enabled).

NOTE : Although *Nuaj'* controls refinement schemes as a single parameter accessible from the Manager, you can actually use different schemes for different modules (like Accurate for the sky and Smart for the clouds for example) but you will have to code this manually.

5.3. Shadow Map Parameters

This is where we specify the shadow map dimensions but also the optional cookie we assign to the light to cast cloud shadows onto the scene.

Shadow Map Size / Shadow Far Clip

This parameter specifies the size of the shadow map texture.

The shadow map supports up to 4 cloud layers, each top layer casting its shadow onto the lower one. The larger the shadow map, the more precise the shadow of course but also the more time consuming to render.

A typical value is 512 but keep in mind that clouds are usually quite fuzzy so you can use quite low values here.

The *far clip* will specify the distance from the camera at which the shadow frustum will clip.

It has a strong incidence on how much area the shadow will cover, this should be a large number otherwise you can see the shadow map wrap at some point (especially when viewing from high altitude).

NOTE : 3D clouds in *Nuaj'* use their own volumetric shadow with its own size. It is important to know that having two different sizes for the shadow maps can introduce a little flickering so you should keep the 2 size parameters tied together as much as possible.

Light Cookie

Nuaj' can optionally offer to render the shadow map into a light cookie that will then be attached to the light, but only if the light is a directional light!

You can also specify the size of the cookie texture (same remarks as for the shadow map above) and the size of the projector in world units. The larger the cookie size, the more ground will be covered by the cookie shadow map obviously.

NOTE: Unfortunately, Unity doesn't provide a "Cookie Size" property other than through the inspector so I had to duplicate that property in *Nuaj'* (the "Light Cookie Size" property). You **MUST** make that size match the cookie size in your directional light or some nasty texture wrap bugs may appear !

Finally, you can specify either a custom altitude at which the shadow map should be sampled, or use the current altitude of the camera.

Procedure to correctly configure your directional light to receive the cookie

Firstly, you need to understand that the cookie shadow map is computed where the directional light is located. It means that if the light is simply dropped into the world, the cookie will show shadowing for a square around the light's position, even if your camera stands far away.

If you want to maximize the impression of clouds rolling and billowing over your head and casting shadows to the ground then you need to make the light somehow stick to the camera.

That is exactly the purpose of the little script you can find in

"Nuaj/Scripts/Helpers/LightScripts/ProjectLightToGround".

You want to take that script and attach it to your directional light.

Of course, this is not perfect as if you start climbing up into the sky then you will see more and more of the landscape below, so the cookie should increase in size to cover that area. Unfortunately, as said earlier, there is no way (yet, but the bug has been filed!) to change the cookie size manually via scripting in Unity.

UPDATE: Still no fix, it's been more than a year now! C'mon Unity guys, it's a simple parameter to expose!

5.4. Local Variations Parameters

These parameters control local variations that are taken into account by fog & cloud layers.

Local Coverage

Clouds can have a local coverage guided by a local density map that will modulate the presence or absence of clouds. That map is provided by a MapLocator game object prefab.

The local coverage texture controls the density of each of the 4 cloud layers via its R,G,B,A components, each component corresponding to a different layer. The components are ordered from lowest layer (Red) to highest (Alpha). For the default offset and factor values in the locator, a value of 0 (black) in the texture means no cloud while 1 (white) means full cloud density.

This is ideal to simulate cloud fronts, or zones of calm weather among a storm far away, or even to paint your own exotic cloud shapes.



Figure 5-4 – Funny custom coverage

HINT: If you know your way around shaders, you can easily create a material that will render a dynamic custom coverage into a render texture that you can then feed to the locator each frame. This will have the effect of dynamically controlling the cloud density to achieve very cool effects, like making the clouds appear over a specific position as a result of a magic spell or something...

Emissive Terrain

You can specify an emissive ground color that will be used to paint the clouds from below. This can be useful to simulate artificial lighting from a city for example. The light is directly projected vertically onto the clouds' bottom so there is no fancy lighting formula applied here. Don't expect mighty diffusion within the clouds or anything.

NOTE: The Alpha component of the emissive terrain texture is used to modulate the terrain albedo.

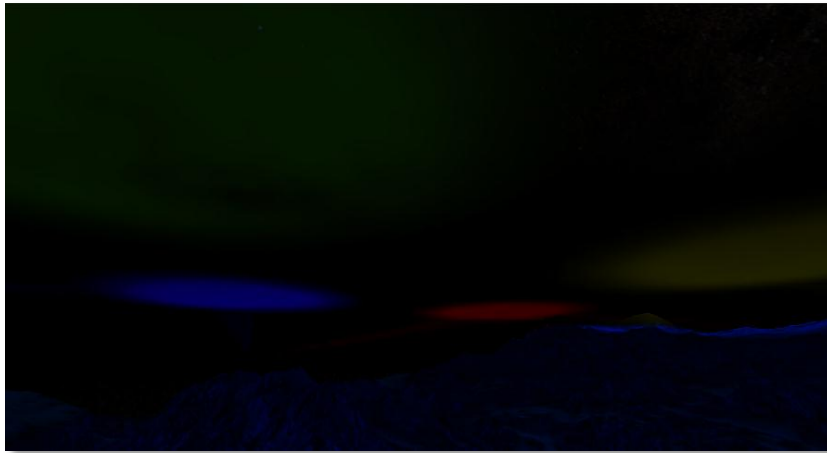


Figure 5-5 – Funky cloud coloring at night as a result of using an emissive map.

Terrain Albedo

This parameter is used to specify the albedo of the terrain so **Nuaj'** knows how much sun light the terrain will reflect.

NOTE : The alpha component of the albedo color is used as a factor to the color. For example, if we use a color of (255,128,64) then an alpha of 255 will indeed yield an albedo of (255,128,64), but if you use an alpha of 128, the actual albedo will then be (128,64,32) (divided by 2). This helps you to have a higher precision.

The environment map computed by **Nuaj'** is also used to guess the average amount of sunlight that reaches the ground unaffected by the clouds.

I use that ambient light to reflect sunlight to the bottom of the clouds, providing some realistic backlighting effect but it's important to get right and not too bright for you might get unexpected and unrealistic results.

Typical albedo values for terrains can be found here: <http://en.wikipedia.org/wiki/Albedo>

As you can see, albedos are quite low except for fresh snow where they can reach up to 85%.

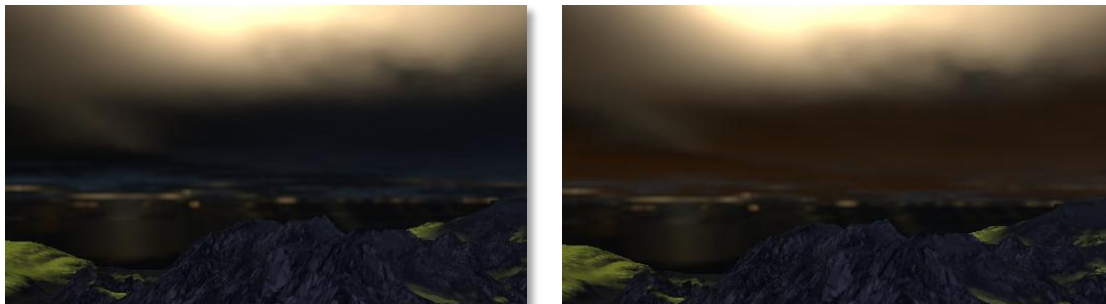


Figure 5-6– The effect of terrain albedo on the bottom of clouds.
Left shows no albedo. Right shows a moderate albedo.

5.5. Lightning Parameters

Yes, you can simulate lightning storms in **Nuaj'**!

You can specify up to 2 simultaneous lightning bolts as linear light sources (a segment of light) within the clouds. Of course, the lightning bolts could also be used to simulate other sources like the lights of a plane flying through the clouds, some kind of explosion or anything really...

Lightning is provided by the LightningBolt game object prefab.

You only need to drop a prefab into your scene hierarchy then assign this instance to one of the 2 lightning bolt slots existing in the manager.



*Figure 5-7 – The effect of lightning in the distance.
(Note here that only lighting is shown, no lightning mesh is used to locate the effect)*

5.6. Luminance Adaptation Parameters

As was stated earlier, all rendering in **Nuaj'** takes place in HDR (High Dynamic Range) to achieve the most realistic look. This means the light levels that are computed can be very high during daytime or very low during night time.

As we saw earlier, the Sun's intensity is set to 10 by default whereas Unity only allows light levels up to 1 and clamps anything above that.

On the other hand, at night, intensity levels go down to 0.01 so we have a factor of 1,000 (!!)

between daytime and nighttime, which can only be handled in HDR.

In order for Unity to display such large ranges of values, we absolutely need to adapt the HDR luminance into one that a standard display device can handle.

Computation Type

This indicates which algorithm we want to use to compute the image's overall HDR luminance.

- **Downsample**, this setting tells **Nuaj'** to downsample the image down to a single 1x1 pixel that we can then read back as being the image's luminance.

- **DownsampleLog**, this is the same as above except we use the logarithm of the luminance. This is especially useful when dealing with very large ranges but is mostly useless in a majority of cases.

NOTE : In both of the above algorithms, we need to make a choice between the kind of luminance we want to use for computation : maximum or average ?

That is the purpose of the “**Average or Max ?**” slider.

If you decide to use the maximum luminance then the brightest pixel will be elected “miss image luminance of this frame”, yielding a (perhaps too) perfectly adapted luminance.

On the other hand, if you decide to use the average luminance then bright pixels will be washed out against dark ones and the image may burn because of under-adaptation. This is really up to you to select the correct setting.

- **One**, this is a basic setting that assumes a constant image luminance of 1. You might experience really bright images at day and very dark ones at night, obviously.
- **Custom**, this lets you specify your own custom luminance. Along with the **One** algorithm, this is the cheapest method and it can also really become the method of choice when you have to script-control the environment rendering and need to feed the luminance manually depending on the area the player is standing (cf. the Manual Luminance Input section).

NOTE : Both downsampling algorithms are quite expensive in terms of computation time but are essential to a correct luminance adaptation.

What is expensive is not really the downsampling part that reduces the screen buffer down to a 1x1 value but rather the fact that I lock the 1x1 texture to read it back with the CPU !

This is never a good idea in general but it cannot be prevented here, unless you change your entire lighting pipeline (you should contact me so I tell you how to do that if you're interested).

Input Scene Type

Using the “**Scene in HDR**” checkbox, you can tell if the input Unity scene was already rendered in HDR, in which case the next parameter “Scene Luma Correction” is of no use at all.

Otherwise, well, we need to compensate for the fact the input scene was computed in LDR (Low Dynamic Range).

The “clever” trick I’m using to create a HDR scene from a LDR one is simply to multiply the scene’s color by the inverse of the factor I used to convert it from HDR to LDR the last frame.

This is quite important that you understand that because this trick induces some sort of feedback loop that can go really awry when setting the Luma Correction to idiotic values.

Faking Unity's LDR Rendering into a HDR rendering

So basically you have that LDR scene color rendered by Unity that I multiply by some “factor to make it HDR” and then multiplied again by the Luma Correction value, yielding a somewhat fake HDR color value for the input scene.

It's very important at this stage not to overshoot the luma factor so the scene doesn't give too much luminance, which could darken the *Nuaj'* HDR rendering. And it must not be too low either so the input scene doesn't burn.

This fake HDR color from the Unity scene is then composited with *Nuaj'* rendering (which was rendered in true HDR), yielding a somewhat less fake HDR color.

Finally, this resulting color is converted back into LDR via Tone Mapping Parameters.

Day-/Night-Time Luminance Levels

These parameters describe the adaptation capacity of the eye during daytime and nighttime, as well as its response speed to changes in luminance.

Did you know you had 2 pieces of hardware in your eyes ?

Cones, working during the day (*a.k.a. photopic vision*), and rods working during the night (*a.k.a. scotopic vision*).

Well, *Nuaj'* performs a very basic simulation of that behavior (although you can override it using custom luminance adaptation, cf. [API Manual](#)) and lets you set the minimum and maximum luminance levels the eye can see during day/nighttime but also the speed at which the eye can adapt to a quick change in luminance (like when exiting a tunnel during daytime, or blinded by a powerful flashlight during nighttime).

Indeed, adaptation speed is much faster during daytime whereas it can take several minutes in some cases to accommodate in low luminance conditions.

So, in general when you move the camera around, don't expect the intensity levels to accommodate perfectly and immediately: depending on the luminance levels and adaptation speed, it can take several frames (what I do usually in edition mode is frenetically middle-clicking the viewport to trigger a refresh and wait for the luminance adaptation to come to a stop).

5.7. Tone Mapping Parameters

Once the image luminance has been determined, it is important to apply a tone mapping operator on the HDR image so that it is converted back into a LDR image that can be displayed by a standard display device. The Tone Mapping parameters let you do just that!

There are 5 different algorithms implemented in *Nuaj'*, each having its particular look and feel. You can also disable tone mapping altogether so the HDR luminance is written as-is, at your own risk of course.

Here are the results for each algorithm (using a gamma of 2.2):

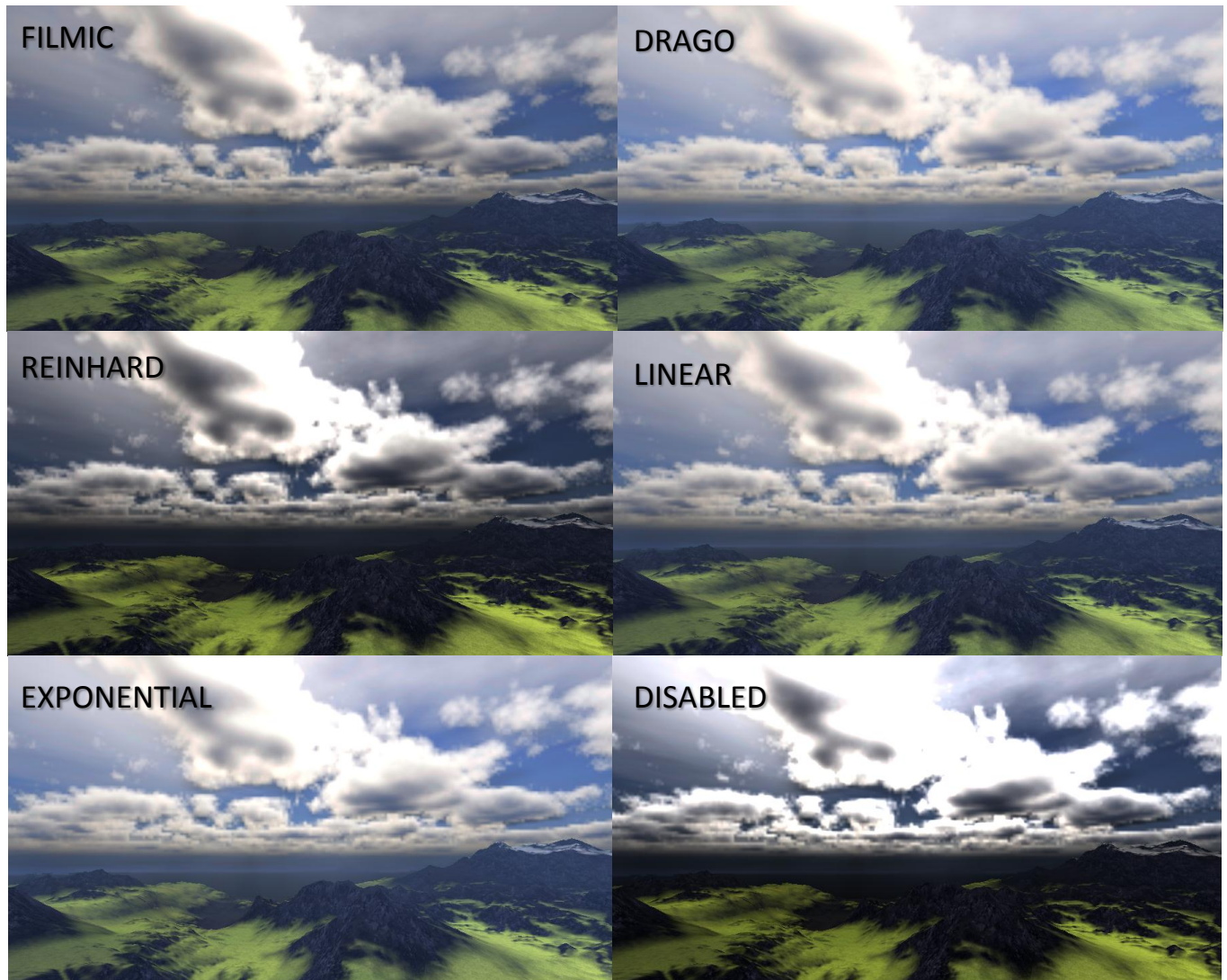


Figure 5-8 – The various tone mapping operators used in *Nuaj'*

You can find some technical documentation and discussions about the algorithms there :

- **Filmic** <http://filmicgames.com/archives/75>
- **Drago** <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.7814>
- **Reinhard** <http://www.cs.ucf.edu/~reinhard/cdrom/>
- **Exponential** http://megapov.inetart.net/manual-1.2/global_settings.html

Gamma Correction

This is the classical gamma correction exponent that allows compensating for the non-linearity of

display devices. I'll let you google that if you need more information on the subject. The default value is 2.2

Sun & Ambient Color Factors

As the image is converted from HDR back to LDR, so do the Sun and Sky's ambient colors that get tone mapped as well, using the same operator as the image.

These factors are applied to the Sun light you used for **Nuaj'**, and to the scene's Ambient light that is found in "Main Menu > Edit > Render Settings".

These factors are extremely important to get right as they are the ones that condition the lighting of your scene and the lighting has to be coherent with the rendering of sky and clouds otherwise the resulting image will feel "wrong".

5.8. Glow Support

Nuaj' is a team player. **Nuaj'**s rendering can be used to write alpha values that are compatible with the Glow Image Effect (UnityPro only).

Indeed, the strength of the glow effect is guided by the value stored in the alpha channel of the screen buffer. **Nuaj'** can specifically write an alpha value that is consistent with its HDR rendering, hence creating a more realistic glow effect.

Enable Glow Support

If you added the Glow image effect to the drive camera **Nuaj'** is using, then, providing the Glow component stands after **Nuaj'** "ComposeAtmosphere" component, you can enable that option.

If unchecked, then the alpha value from your scene will be used for the Glow, as if **Nuaj'** wasn't even there (pass-through mode).

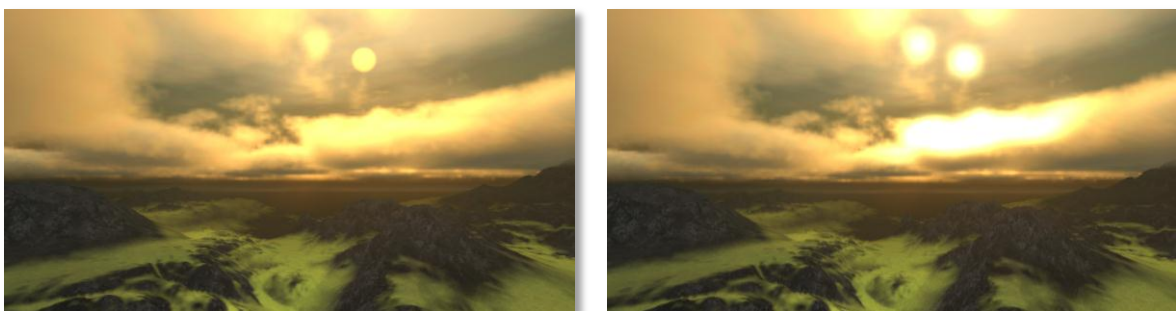


Figure 5-9- Glow support in action.

Combine Glow Alphas

If enabled, the written alpha value is the maximum of your scene's alpha value and **Nuaj'**s luminance value.

If disabled, *Nuaj'* only writes its luminance value into the alpha, discarding your scene's alpha.

Min/Max Intensities

These 2 values control the minimum threshold from which the glow starts to appear, and the value at which it reaches its maximum intensity.

5.9. Render To Cube Map

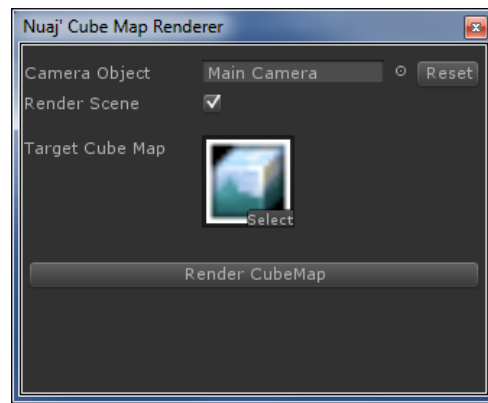


Figure 5-10 – The infamous *Nuaj'* cube map renderer !

Typical `Camera.RenderToCubemap()` function doesn't work with *Nuaj'*, so I had to rewrite my own. This helps creating nice cube maps to use as environment maps for rendering. The tool is pretty self-explanatory.

We're now exiting the global controls and entering the specific modules of *Nuaj'*.

6. Sky Module

The sky module performs many things, not only the rendering of the sky but also the essential composition of the sky and the clouds.

Indeed, nothing is entirely separated in *Nuaj'* and all modules are inter-related. This helps maintaining an overall coherent rendering.

6.1. Sky Model

There are 3 sky models available in *Nuaj'* :

- **COMPLEX**, this is the most realistic model as it is a truly volumetric model that takes into account local air/aerosols quantities as well as local shadowing (*a.k.a.* God rays).

With this model, you can also climb up to outer space and view the Earth from orbit. To achieve all this, it obviously takes a bit more time than the other models.

- **SIMPLE**, this is a much simpler model that assumes a constant air/aerosols density wherever you are. It's cheap to evaluate but is also somewhat less realistic, except in heavy weather conditions like storms where air is very dense anyway. With this model, you have no God rays and you can't escape to outer space.
- **NONE**, this is the cheapest model of all since it doesn't compute a sky at all. It can be useful when you need to custom render a cloud alone on a dark background that you can later composite as you wish.

NOTE: This parameter has a strong influence on rendering speed! (haha no kidding?)

Custom Sky Rendering (advanced user !)

It is possible to custom render the sky yourself by disabling the sky module and hooking the **CustomSkyComposition** event in NuajManager (cf. [API Manual](#)).

You are then given the different cloud layers as distinct RenderTextures, as well as the background map (i.e. satellites) that you can tweak together to yield a final composited map.

6.2. Standard Parameters

For both sky models (except the NONE model of course), you have access to these parameters:

Rayleigh Density

This parameter controls the density of air molecules at sea level (*i.e.* 0 km high). This is the parameter that gives the sky its familiar blue color. The more air sunlight traverses, the more it gets tinted: blue first, then red and even green if you're too generous with the parameter! ☺

This parameter should be increased at sunrise or sunset to give the sky a warm red color. In nature the "parameter" increases automatically at dusk because air gets colder and packs down to the ground.

It also decreases automatically at dawn as air gets warmer and flies upward (the day cycle is also the reason why sunrises and sunsets have a slightly different look: at sunset, warm air descends while at sunrise cold air rises, yielding different configurations).

NOTE: The Orchestrator script automatically increases the density when the Sun is low.

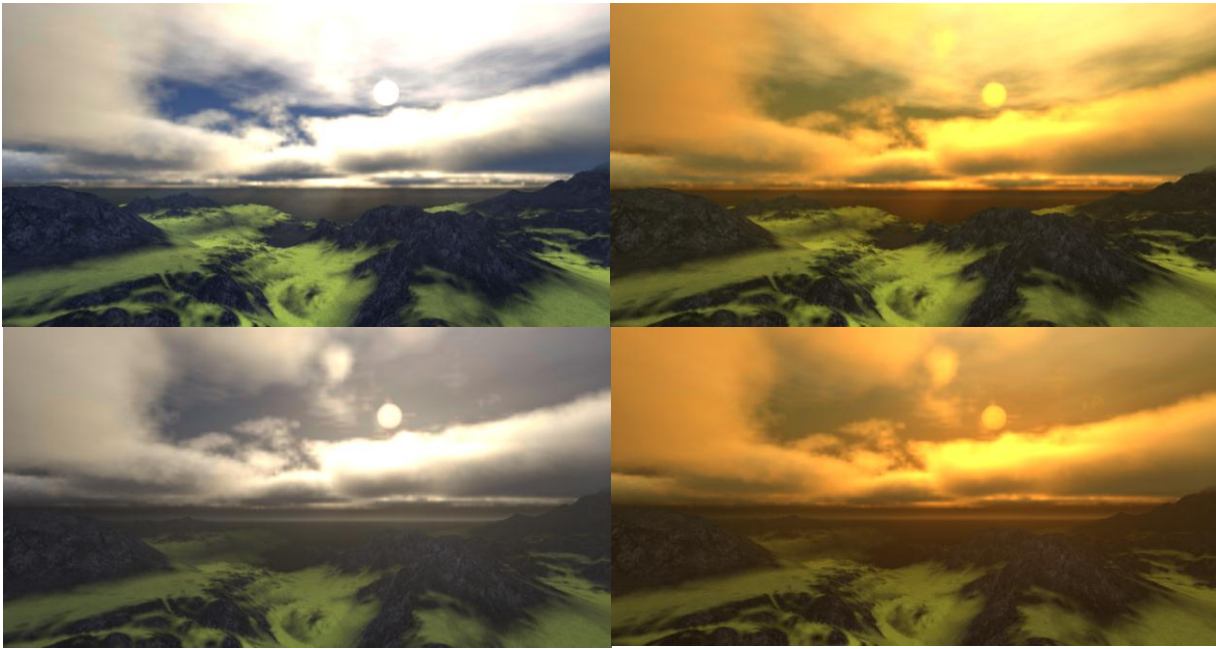
Mie Density

This parameter controls the density of aerosol particles at sea level. It can be thought of as fog if you like. But atmospheric fog, not the low-altitude layered fog we will discuss later.

As for Rayleigh density, the parameter should be increased at sunset and decreased at sunrise but also depending on atmospheric pressure.

Low pressures are also accompanied by an increase in humidity, and small water droplets are typically the kind of particles represented by the Mie density parameter.

NOTE: The Orchestrator script automatically increases the density when the Sun is low.



*Figure 6-1 – The effect of Rayleigh and Mie densities on rendering.
Top-left : low Rayleigh & Mie. Top-Right : high Rayleigh and low Mie.
Bottom-left : low Rayleigh and high Mie. Bottom-Right : high Rayleigh & Mie.*

Scattering Anisotropy

This parameter is not that fun to play with. It operates on Mie light-scattering and basically tells how the light hitting a particle is going to be scattered. It can take values from -1 to +1, where +1 means the light is fully reflected (as if particles were little mirrors) and -1 means the light is fully refracted (goes on through the particle as if they were perfectly transparent).

Its default value of 0.5 is quite correct and makes the particles somewhat transparent and dispersive.

Scattering & Extinction Boost

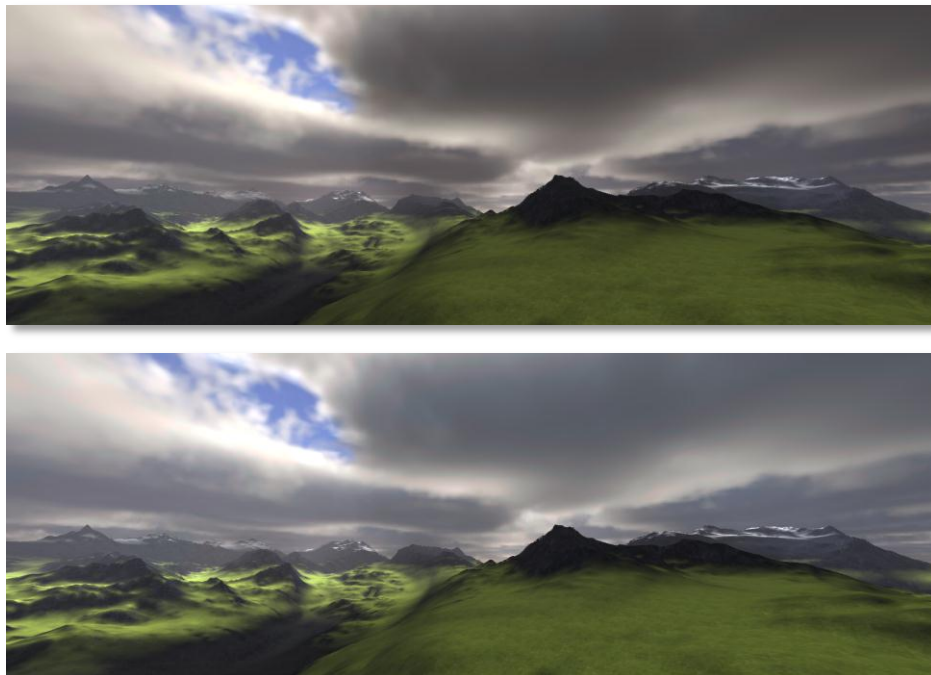
New since version 1.1 is the ability to boost the scattering & extinction effects of the sky so you can fully override the default values. This is very useful to increase the impression of greatness of some sceneries.



*Figure 6-2 – Default values (top) vs. boosted values (bottom).
Notice how the haze is slightly increased at the horizon and the sky is a much brighter blue.*

Custom Ambient Sky

If enabled, this lets you override the ambient sky color used for isotropic cloud lighting. This color is usually computed automatically (cf. Sun Parameters and the Software/Hardware sky computation) but you can totally change it and use your own, this can be very useful sometimes if you think clouds deserve to be a lot more red at sunset and so on...



*Figure 6-3 – Automatically computed ambient sky light (top)
vs. custom blue sky light (bottom).*

6.3. Rendering Parameters

Downsample Factor

As stated earlier in the World Parameters section, many modules render in downsampled buffers that are later upsampled to the screen resolution.

This parameter guides the downsampling factor applied to the screen resolution to determine the size of the buffer the sky will be rendered into. For example, the default factor of 0.25 means the sky will be rendered in a buffer $\frac{1}{4}$ the size of the actual screen.

NOTE: This parameter has a strong influence on rendering speed! The lowest the factor, the faster the rendering of course, but also the less precise.

Sky Steps Count (complex sky model only)

As mentioned at the beginning of the chapter, the complex sky model is a fully volumetric model and, as such, is traced by following rays from the camera up to the atmosphere, taking small steps at a time.

NOTE: This parameter has a strong influence on rendering speed! The more steps you use, the more precise the rendering will get, but also the slower.

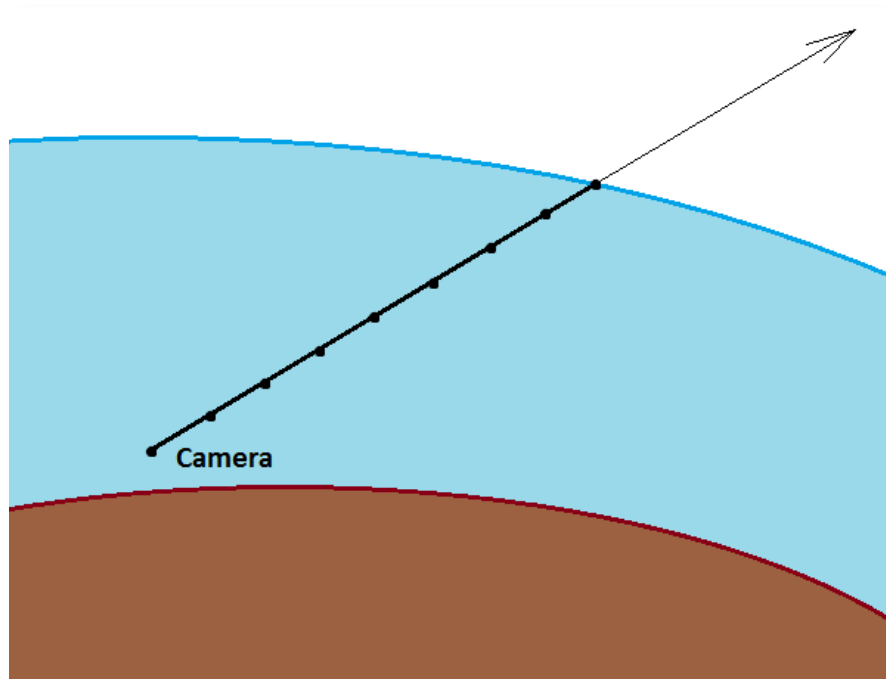


Figure 6-4 – Volumetric tracing through the sky using small steps.

Enable God Rays (complex sky model only)

The slice of sky below the lowest cloud layer is a special case : we take a particular care to trace it with more steps than the rest of the sky and we also sample the sky shadow map to know if a particular point in the volume is lit or not : this has the effect of producing what is usually known as

“God Rays”.

God Rays are better seen when dark clouds are present, or when Mie/Rayleigh densities are set to high values, like at Sunset for example.

NOTE: This parameter has a strong influence on rendering speed!



Figure 6-5 – Your friendly neighborhood God paying a visit.

God Rays Min Steps Count (complex sky model only)

The amount of trace steps used for the sky is overridden in the case of God Rays (i.e. lowest altitude sky layer) using this value.

NOTE: This parameter has a strong influence on rendering speed! The higher the value, the better quality of the God Rays, but also the slower the rendering.

Enable Earth Shadow

This parameter tells if you want to take the shadow of the Earth into account. This is most important at sunset and at night for the Earth to correctly shadow the sky otherwise you can end up with a sky lit at night!

The effect of the Earth shadow is more obvious as seen from high altitude, where the [terminator](#) is clearly visible.



Figure 6-6 – The terminator, seen from orbit.

6.4. Low Altitude Fog Layer

You can amplify the fog effect given by the Mie density parameter we saw earlier if you enable the Low Altitude Fog Layer.


NOTE : enabling the fog eats up one of the 4 available cloud layer slots (*Nuaj'* only supports up to 4 simultaneous cloud layers).

The low altitude fog can be used to simulate many effects from simple mist to sand storms and rain. It's made to have a low volumetric resolution only (only a few steps are traced within the fog, the default value is 8 steps) and each step samples a density texture and a 3D noise texture used to perturb the density.



Figure 6-7 – Some funky volumetric effect showing in the fog.

6.4.1. Layer Parameters

As this is your first layer, let's take some time to explain the basic parameters available for all layers. First, notice that all layers (and satellites) have a small  icon anchored to their right: that is used to reset the layer's parameters to their default values.

Second, the basic shape of a layer is a slice of dense particles sandwiched between 2 spheres, as seen previously in the World Parameters section.

Altitude & Thickness

These 2 parameters describe the altitude of the bottom of the layer (i.e. where the layer starts, basically) and its thickness. Both parameters are in kilometers.

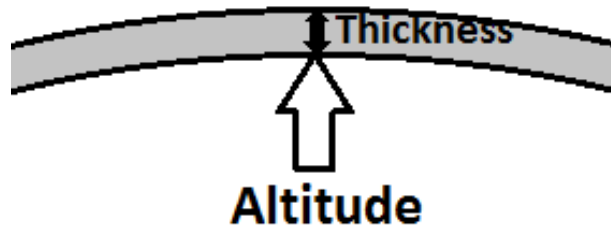


Figure 6-8 – “I don’t like your altitude!”

Cast Shadows

Enables shadow casting by the layer. This is very important for high altitude cloud layers as they cast shadows to the layers below. A little less important for the fog layer as it’s usually the lowest altitude layer and, as such, only casts shadow to the ground below.

6.4.2. Appearance Parameters

Locator

The locator helps to place the fog density texture within the scene. The density texture allows you to specify the fog density at different heights using the RGBA components: Red is density at the bottom of the fog layer while Alpha is density at the top.

This way, you can paint all sorts of volumetric textures with 4 layers stacked one on top of the other. This density value is also composited with the local density texture we discussed in the Local Variations Parameters section.

Finally, the density is perturbed by a 3D noise value (a single octave), adding low or high frequency variations to an otherwise smooth fog.

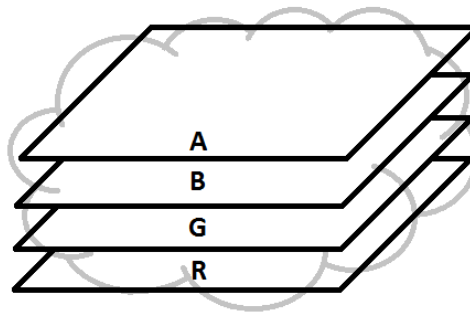


Figure 6-9 – Vertical density layers encoded by the density texture.

Mie Density Factor

This is the factor to apply to the Mie Density parameter from the sky we saw in the previous section. This allows increasing or decreasing the fog density for the layer, but always in relation to the global sky's Mie density : changing the fog's factor whereas the sky has Mie density of 0 won't change anything.

Density Bottom Ratio

This is somewhat another disguised factor : it indicates how much of the Mie density we want at the bottom of the layer. A value of 1 gives the same density at the bottom and the top. A value of 0 gives a full density at the top and no density at all at the bottom while a value > 1 gives a stronger density at the bottom than at the top (such values make more sense as more fog packs to the bottom, usually).

Fog Max Distance

This parameter indicates the maximum distance at which the fog spreads. You can have very large or very localized areas of fog.

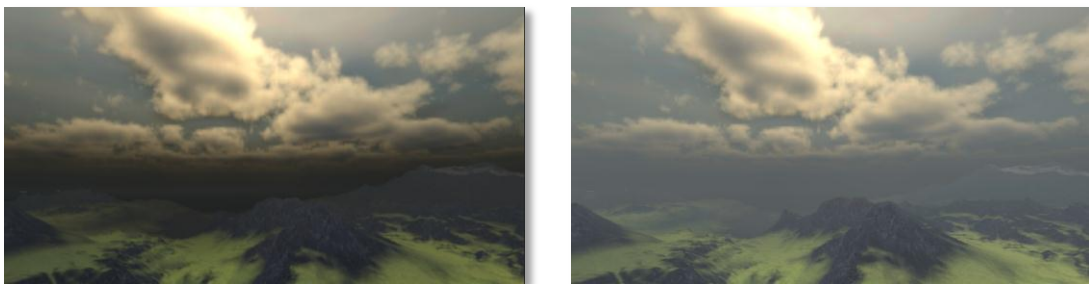


Figure 6-10 – Short versus long distance fog.

Fog Color

Well... This is the fog color.

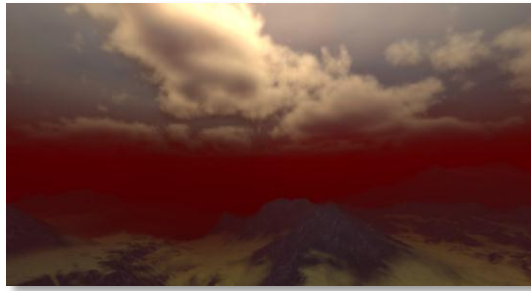


Figure 6-11 – The red fog menace !

I really have no idea why you would need a funky colored fog for but I suppose if you want to create some sort of radioactive-zombies game or a nasty flesh-eating mutant fog virus, it might come in handy... Who am I to judge?

Ambient Sky Factor

Such a pedantic name ! Well, lighting in **Nuaj'** comes in two flavors: directional light from the Sun and ambient light from the sky (a hemispherical light source).

This parameter allows you to control how much of the ambient sky light diffuses within the fog.

In **Nuaj'**, ambient sky light can also sometimes be called “isotropic” light, from “iso” meaning equal and “tropos” meaning direction: the same from all directions (as opposed to anisotropic, which is biased towards a specific direction).

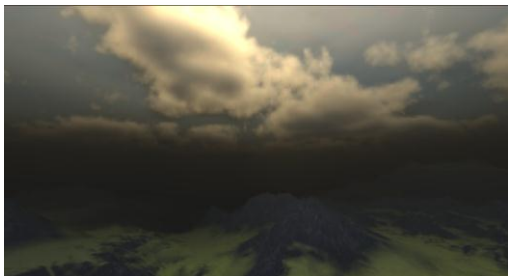


Figure 6-12 – The effect of low versus strong ambient sky diffusion factor.

Steps Count & Size

These two parameters indicate the amount of volumetric steps we will take within the fog and the size of such steps.

NOTE: This parameter has a strong influence on rendering speed! The more steps you use, the more precise the rendering will be but also the slower.

The step size, on the other hand, has no direct influence on speed but rather on quality and appearance.

- Large step sizes (above a few hundred meters) allow you to create long-range effects and are more suited to a fog that varies slowly, like a mist.
- Short step size allow you to create more localized and high-frequency effects like rain, or puffs of wind that are common to sand storms.

6.4.3. Noise Parameters

As stated earlier, the fog density is perturbed with the use of a simple 3D noise. By varying its frequency and amplitude, we can achieve some pretty cool effects.

To simplify your life, I created 3 main presets: Rain, Mist and Sandstorm.

Each preset updates the Noise and Animation parameters, but also the “Step Size” parameter seen just before. Indeed, rain for example is a short range effect that needs the most resolution near the camera so the step size is set to a low value. On the other hand, mist is a long range, low frequency phenomenon so the step size is set to a large value.

NOTE: To achieve a convincing rain effect, it's usually not enough to click the “Rain” preset.

Indeed, this preset works much better when you accompany it with an overcast sky by bringing in some clouds that darken the overall atmosphere. I mean, you can't usually have rain by a clear sky, can you?

Also, the rain effect only alters the density of air, it does not display rain particles. This should be the job of a totally different process (cf. the Tutorials section on www.nuaj.net).

Frequency Parameters

You can alter the noise's frequency through the horizontal and vertical tiling factors. The horizontal noise tiling indicates how much the noise will tile in the horizontal plane (front and right) while the vertical noise tiling indicates how much the noise will tile vertically.

It's quite important to separate the 2 since interesting elongated noise effects can be achieved by setting a high horizontal tiling frequency and a low vertical one: this way you can simulate something that looks a bit like heavy rain.

Amplitude & Offset

With these parameters you can define the amplitude of the noise and its bias.

A default noise has an amplitude of 1 and is unbiased: its values will vary homogeneously between -1 (subtracting density) and +1 (increasing density).

The Offset parameter helps changing whether the noise will rather decrease intensity (offset < 0) or increase it (offset > 0).

6.4.4. Animation Parameters

The noise would be nothing without animation. You can define wind that animates the noise in the horizontal plane. Wind is defined by its force and its direction.

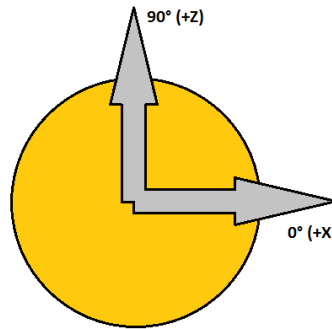


Figure 6-13 – Wind direction is aligned with the Unity axes.

Downpour Strength

The noise can also be animated vertically to simulate rain downpour. Increasing downpour strength will make the noise scroll down more rapidly.

7. 2D Clouds Module

This module can display several layers of thin clouds (up to several hundred meters) that are usually well suited for high altitude clouds like the cirrus and stratus.

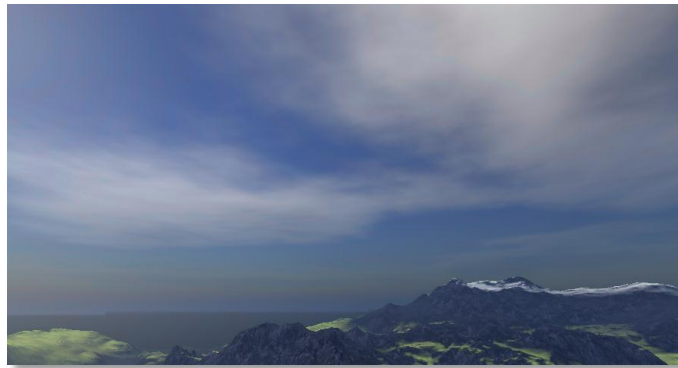



Figure 7-1 – Some high altitude clouds spanning a huge area.

You can add a new 2D cloud layer by pressing the “Add Layer +” button, and remove an existing layer by pressing the small  button at the top right of each cloud layer.

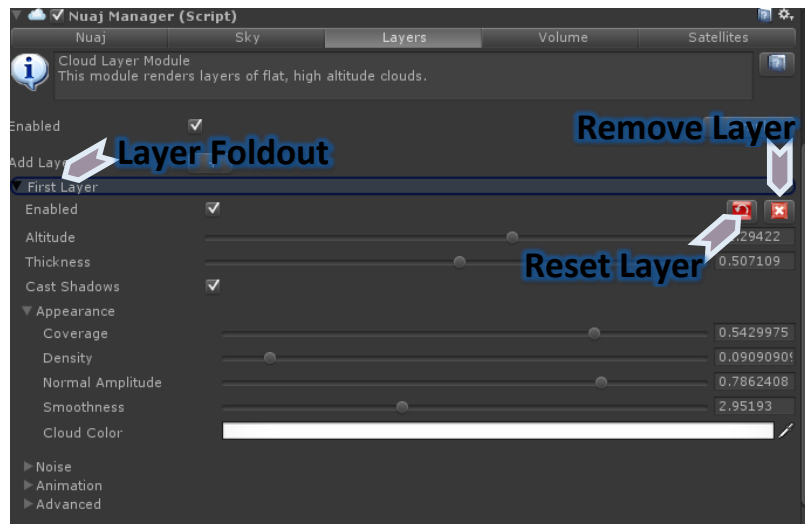


Figure 7-2 – The 2D cloud layer editor.

As for all layers, you find the Altitude & Thickness parameters described earlier in the Fog's Layer Parameters section.

7.1. Appearance Parameters

Coverage

Coverage represents the percentage of area covered by the clouds. It's not a totally accurate measure as it also depends on the noise textures that you are using but it's guaranteed the sky is totally empty at 0 coverage, and totally full at 1 coverage (unless the texture is empty, in that case, as the good Dr. House would say: you're a moron).

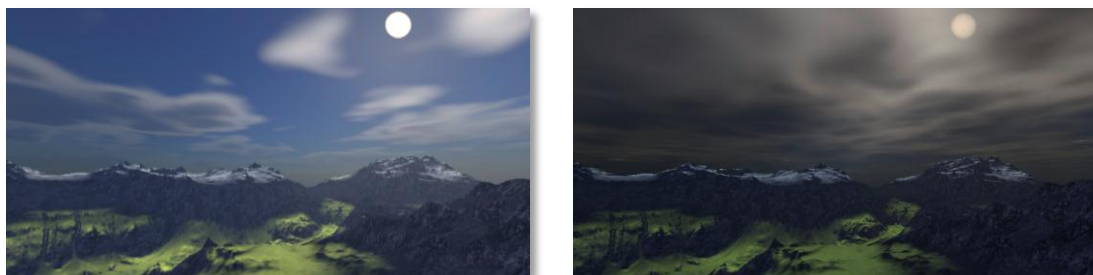


Figure 7-3 – Low- vs. High-Coverage.

Density

This parameter indicates the density of “cloud material” (water droplets, ice crystals, dust, whatever) per unit volume and strongly affects the appearance of the cloud. You can have a very thick cloud with a very low density or a thin cloud with a very high density, yet it will not give the same result.



Figure 7-4 – A thick, low-density cloud (left) vs. a thin, high-density cloud (right).

Normal Amplitude

This allows you to amplify the cloud's normal to increase the change in lighting. This is especially true at grazing light angles at Sunset. Try not to boost the amplitude too much though, as it can sometimes lead to unrealistic “water-like” clouds.

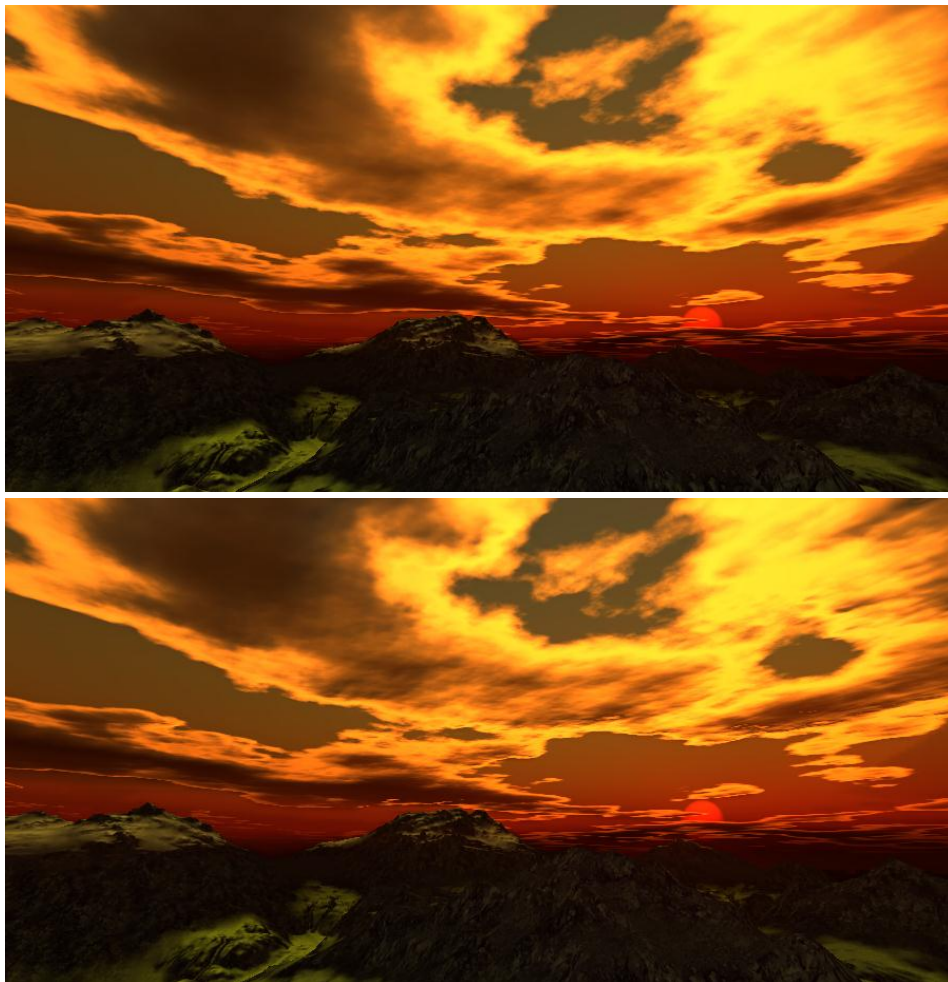


Figure 7-5 – Low normal amplitude (top) vs. High normal amplitude (bottom).
Note that it also impacts the cloud's lighting itself.

Smoothness

This parameter changes the smoothness of the clouds. In fact, it's a direct control over the mip-level we are using for sampling the cloud texture. This mip level automatically changes with distance but you can decrease or increase it manually to obtain sharper or smoother clouds. If you only need to sample at lower frequency to create slow-varying clouds, then you should better use the noise tiling factor that we will review in a bit.

Cloud Color

Well, if you like green clouds, here is your chance!

7.2. Noise Parameters

Noise Tiling

This is the distance the cloud texture will cover before it wraps around. Usually set to a very low value as we don't want to notice the tiling. The default value of 1 is already pre-multiplied by a low value to create a more manageable parameter.

Octaves Count

Well, usually, interesting fractal patterns are obtained when adding several layers of noise (in our case, a noise texture) on top of each other, each with a different importance.

- The first layer of noise has a strong importance, say 1, but has a low frequency (the noise tiling parameter above).
- The second layer of noise has a lesser importance, say 0.5, but has a higher frequency.
- You get the idea.

Here's what it's like in pictures (from <http://libnoise.sourceforge.net/glossary/index.html#octave>) :

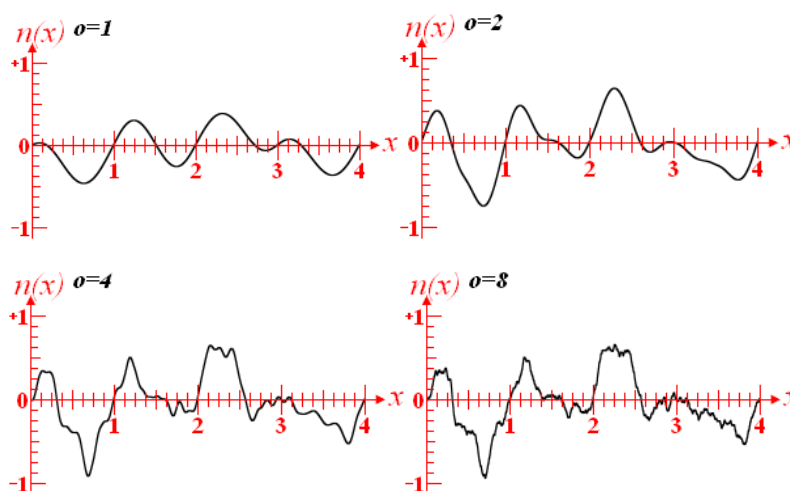
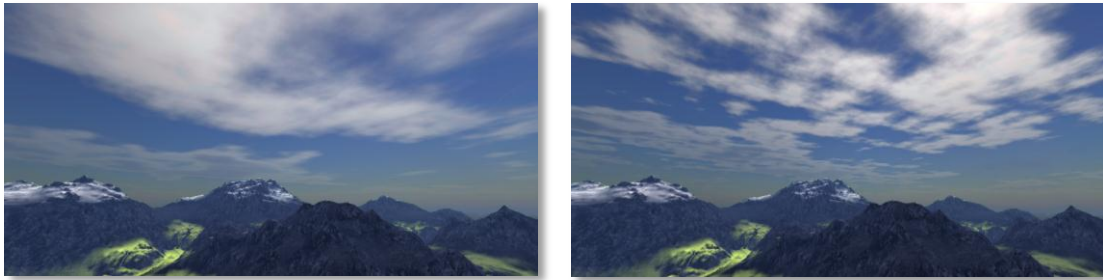


Figure 7-6 – The effect of adding more noise octaves.
This example shows a simple 1D noise but, although we are using
2D noise textures in our case, the idea is the same though.

Of course, the more octaves you have, the more detail you get but it takes also more time to compute. A value of 3 octaves is usually a good choice.

Frequency Factor

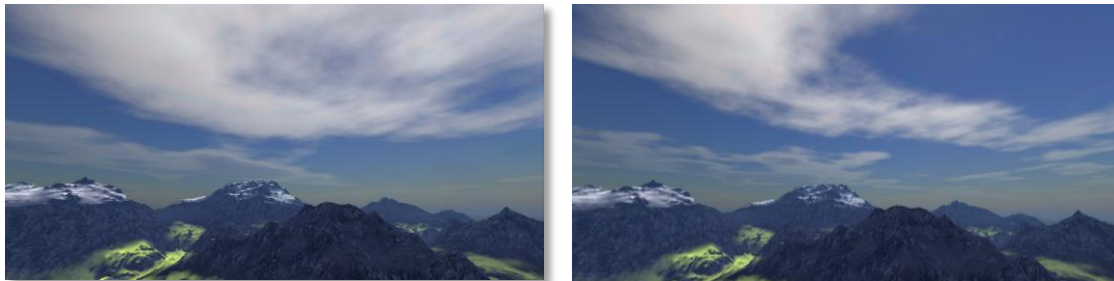
With the octaves story in mind, we understand that we have the choice in the frequency factor we choose for each new octave. A factor of 2 means the frequency will double each new octave, adding more details to an otherwise slowly varying texture.



*Figure 7-7 – Frequency factor of 2 (left) vs. a factor of 4 (right).
Notice how the cloud patterns change more rapidly.*

Amplitude Factor

Also, we have the choice in amplitude factor for each new octave. A factor of 0.5 means the importance of each new octave will be halved. Choosing a factor of 0 is obviously useless as it would mean octaves others than the first one will have no influence at all on the appearance (in which case it would be faster to set the octaves count to 1).



*Figure 7-8 – Amplitude factor of 0.5 (left) vs. a factor of 1 (right).
This means all octaves have the same weight.*

Frequency Anisotropy

This parameter is useful to squash the clouds in a specific direction. This can be used to give the impression clouds are more elongated along an axis, as can be sometimes observed in nature.

Noise Textures

Here you will find the 4 textures slots used for each of the 4 possible noise octaves, as well as some buttons to assign these slot with preset textures (these textures can be found in the “Nuaj/Resources/Textures/CloudLayers” directory).

NOTE: all these textures have been generated using the Noise Generator application, which is a very simple program (Windows only I'm afraid) that is available on www.nuaj.net.

Another way of generating noise textures for *Nuaj'* is to use Photoshop :

1. Use the difference clouds to create the height map first
2. Use the nVidia tool to create a normal map from the height map and tick the "Keep Height in Alpha" option.

7.3. Animation Parameters

Wind

As for the fog layer, you can define the wind force as well as its direction.

Evolution Speed

This parameter helps to specify the speed of each octave relative to the other. An evolution speed of 2 means the second octave will move twice as fast as the first one, the 3rd octave will move twice as fast as the 2nd one and 4 times as fast as the first one and so on. This parameter is important to give the illusion that clouds are changing and evolving.

7.4. Advanced Parameters

Cloud lighting is a complex subject not for the faint of heart. *Nuaj'* is only scratching the surface of the complex physics and chemical reactions going on inside the atmosphere. I'm using many simplifications to achieve rendering in real time since dealing with the actual computation would be utterly infeasible, even with the most powerful computer to date.

Not to go too far into the details here, I want to give a brief insight of what is going on inside a cloud.

First of all, clouds are composed of tiny particles like water droplets several μm in diameter but also tiny ice crystals and water vapor, whose density depends on many –mostly localized– atmospheric parameters like temperature and pressure. Also, water condensates around dust particles: if there is no dust, there is no cloud.

All these particles have an effect on light traversing them. Indeed, the water droplets act as tiny mirrors or crystal balls that locally reflect or transmit light. Each reflection or transmission –or to sum up, each time the light changes direction because of an interaction– is called a *scattering event*.

On a global scale, clouds almost never absorb light (their albedo is very high, almost 100%) and leave a large part of the light unaltered: this means the light goes in a straight path, often for hundreds of meters, without encountering a single particle. But for large clouds of course, the probabilities of encountering such particle are more and more frequent and light can often bounce hundreds of times (a hundred scattering events!) before exiting the cloud.

Simulating as many scattering events would be totally out of question, moreover that would be quite pointless as the most significant effects come from early scattering events. Like single-scattering (light bounces off once) and double-scattering (light bounces off twice), the rest is often summarized as “Multiple Scattering” (the guy who came up with that term really didn’t take any risk, did he? 😊).

Enough with the physics and technicalities, let’s get back to **Nuaj’**! Note that the following parameters must be set very cautiously as you can easily end up with more output energy than what came in, resulting in ultra-bright clouds and a washed out sky due to luminance adaptation.

Zero-Scattering

This parameter specifies the weight to give to light that has not been scattered, *i.e.* the light that went straight through the cloud.

Single-Scattering

This parameter specifies the weight to give to light that has been scattered once (one bounce) before exiting the cloud.

Double-Scattering

This parameter specifies the weight to give to light that has been scattered twice (two bounces) before exiting the cloud.

Multiple-Scattering

This parameter specifies the weight to give to light that has been scattered multiple times (three or more bounces) before exiting the cloud. This is also the term that drives the “ambient” aspect of the cloud.

Sky Scattering

This parameter specifies the weight to give to the scattering of sky light that lights clouds from above.

Terrain Scattering

This parameter specifies the weight to give to the scattering of light reflected by the terrain that lights clouds from below (cf. Local Variations Parameters).

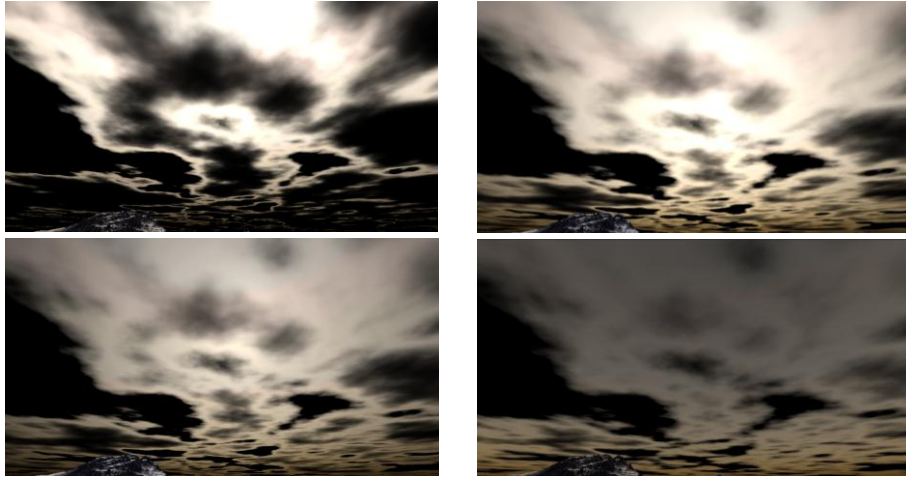


Figure 7-9 – The effects of 0-scattering (top-left), single-scattering (top-right) double-scattering (bottom-left) and multiple scattering(bottom-right).

8. 3D Clouds Module

This module can display several layers of thick volumetric clouds (up to several kilometers) that are usually well suited for low altitude rain clouds like the cumulus and nimbus.

The huge cumulo-nimbus clouds still remaining a big challenge for real time display, but certainly available in **Nuaj'** v2.0. 😊

These clouds represent the meat of the **Nuaj'** beast as they have not really been available in any other package up until now (seriously guys, cut the marketing bullsh.t and have the balls to admit your clouds are not really volumetric but are just a bunch of particles!).

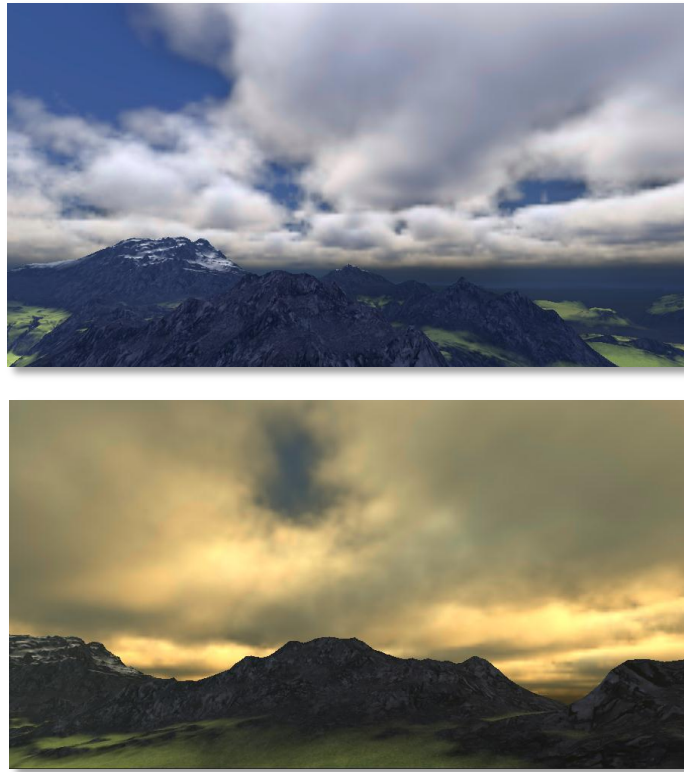



Figure 8-1 – Examples of low altitude clouds and various lighting conditions.

You can add a new 3D cloud layer by pressing the “Add Layer +” button, and remove an existing layer by pressing the small  button at the top right of each cloud layer.

As for all layers, you find the Altitude & Thickness parameters described earlier in the Fog’s Layer Parameters section.

8.1. Shadow Map parameters

Volume clouds need their personal shadow map, on top of the existing shadow map system. These are not ordinary shadow maps : these are volumetric shadow maps !

Shadow Map Opacity

This changes the opacity of the shadow map and is very useful to make the clouds whiter or darker depending on the atmosphere you’re trying to setup.

Shadow Map Size

As for common shadow maps, you need to specify the size of the shadow map texture.

You need at least a value of 256 to get a significant resolution, unless you enable the Smooth Shadow Map option below, that can smooth out the otherwise jaggy aspect of low resolution shadows.

NOTE : 3D clouds in *Nuaj'* use their own volumetric shadow with its own size. It is important to know that having two different sizes for the shadow maps –the 3D clouds shadow map and the main shadow map mentioned in §5.3– can introduce a little flickering so you should keep the 2 size parameters tied together as much as possible.

Shadow Quality

This sets the quality of the volumetric aspect of the shadow map. You can either choose 1, 2 or 3 layers of volume shadow.

Each layer contains 4 sub-layers of shadow map, so choosing 3 layers actually creates a 12-layers deep volumetric shadow.

The more layers the better of course, but it also uses more memory and more time to compute.

Shadow Steps Count

As for the cloud rendering itself, the cloud shadow is evaluated by volumetric ray-marching. This parameter helps to setup the proper amount of steps to take for correct rendering.

Shadow Map Smoothing

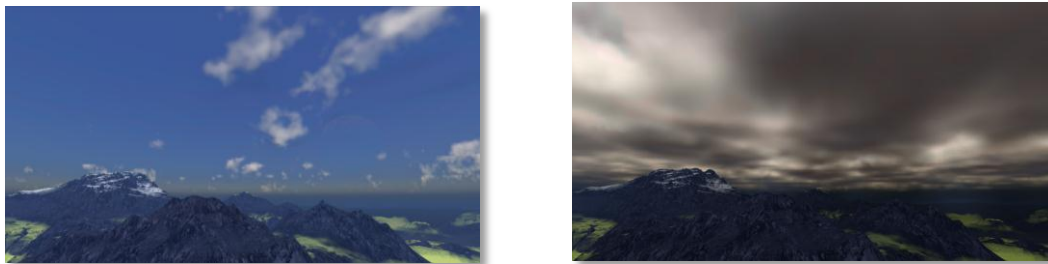
You can decide to smooth out the shadow map a little to get better, smoother results.

For the God Rays at sunset where sun rays are very slant and elongated, this option can be a life saver (and it's not very expensive to use!) but it can also lead to shadows that are a bit "dull" (clouds lacking a clear border).

8.2. Appearance Parameters

Coverage Offset

Coverage offset represents the percentage of area covered by the clouds.



*Figure 8-2 – Low- vs. High-CoverageOffset.
Contrary to the 2D clouds, the effect of coverage is quite dramatic here.*

Coverage Contrast

Coverage contrast represents the sharpness of the cloud density function.

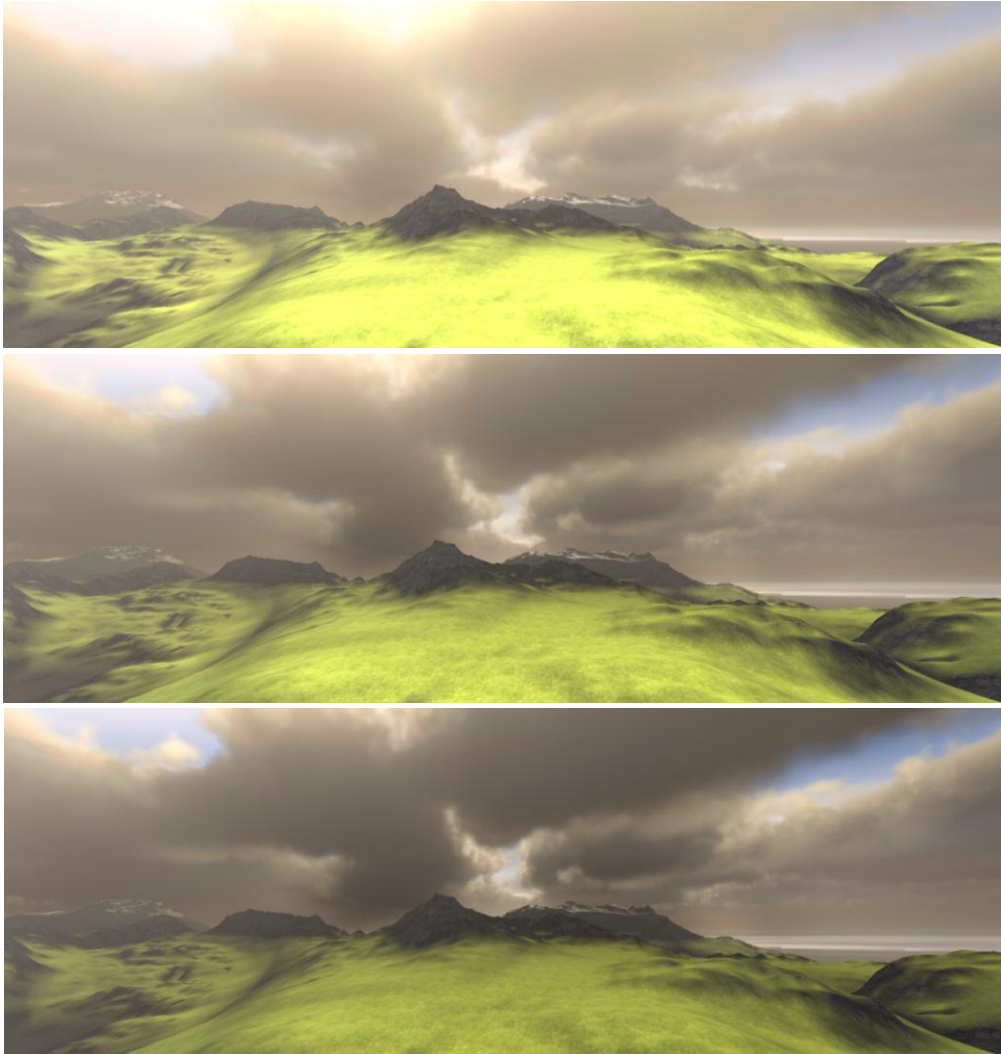


Figure 8-3 – From top to bottom: coverage contrast of 1, 2.5 and 4

Density

This parameter indicates the density of “cloud material” (water droplets, ice crystals, dust, whatever) per unit volume and strongly affects the appearance of the cloud. You can have a very thick cloud with a very low density or a thin cloud with a very high density, yet it will not give the same result.

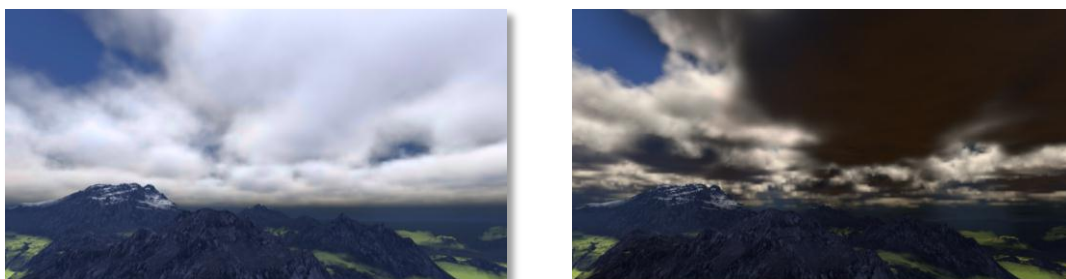


Figure 8-4 – The effect of low-density (left) vs. high-density clouds (right).

Bevel Softness

This parameter is very useful to soften the top and bottom of the cloud layer.

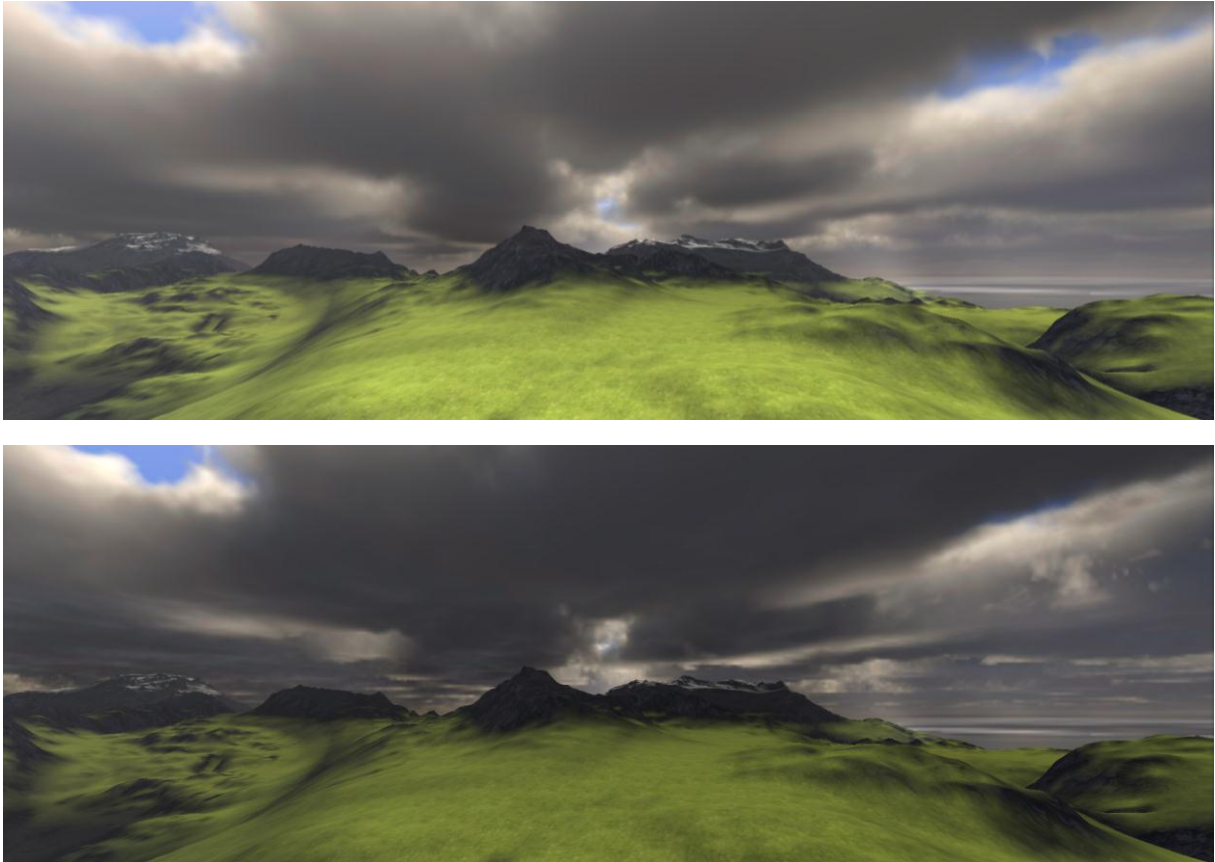


Figure 8-5 – Top: Softness=1 Bottom: Softness=0.3

Cloud Color

As always, if you need green clouds, this option is made for you!

Trace Limiter

This option is essentially used when navigating through clouds to limit the distance at which we can see. It has no clearly defined units, you can think of this parameter as the percentage of the distance we're tracing compared to the actual distance we need to trace.

Indeed, tracing for the actual distance within the clouds when viewing at the horizon would require us to perform many many steps along scores of kilometers to be accurate, but we can't do that so we trace only up to a close range.

This parameter is automatically calibrated based on the density, coverage and view angle when approaching the clouds. You should increase it to your liking if the transition between in and out of clouds is too obvious, or if you feel the viewing range is wrong.



Figure 8-6 – Appropriate Trace Limiter value.



Figure 8-7 – Inappropriate values. Too low (left) and we don't see enough of the clouds. Too much (right) and an ugly horizon line starts to appear.

Horizon Blend

This is very useful to control how the clouds blend in the distance.

You have 2 settings for **Start** and **End** distance that help you set the boundaries of the blending.

This helps a lot to avoid the flickering of high-frequency clouds at the horizon.

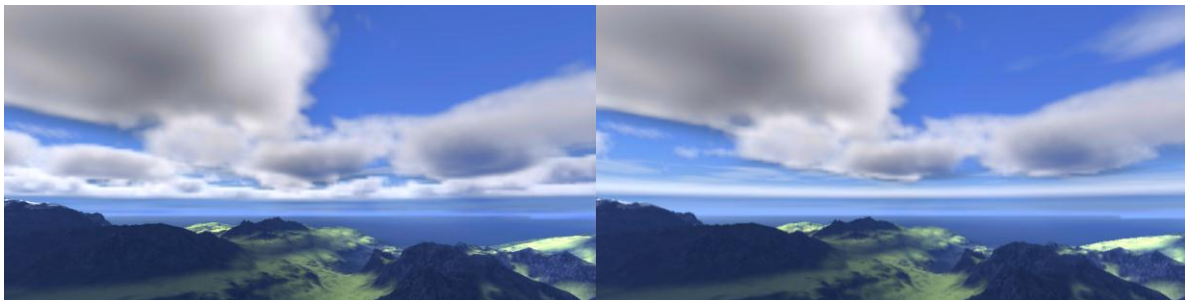


Figure 8-8 – The effect of horizon blending.
Far blending (top) vs. Near blending (bottom).

Steps Count

As stated at the beginning of the chapter, these clouds are fully volumetric and, as such, are traced by following rays through the clouds, taking small steps at a time. Exactly like the complex sky model.

NOTE: This parameter has a strong influence on quality but also on rendering speed! The more steps you use, the more precise the rendering will be but also the slower.

8.3. Noise Parameters

Unlike the 2D clouds, you have no choice in the noise texture used here, it's a single 3D texture combined at multiple octaves (cf. Noise Parameters of the 2D clouds for an explanation about octaves).

Noise Tiling

This is the distance the cloud texture will cover before it wraps around. Usually set to a very low value as we don't want to notice the tiling. The default value of 1 is already pre-multiplied by a low value to create a more manageable parameter.

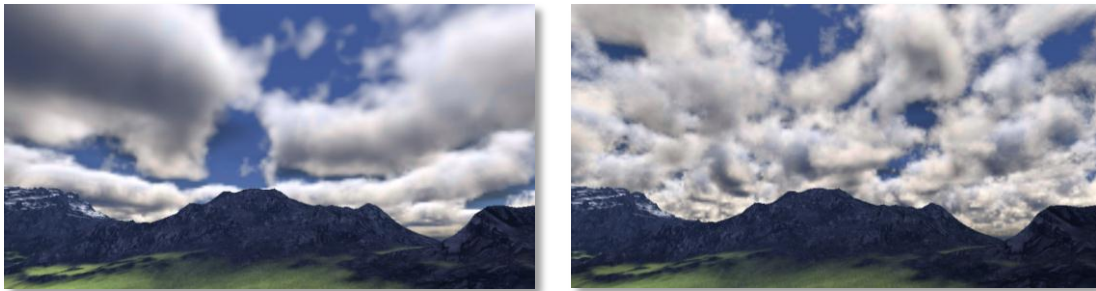


Figure 8-9 – Default 1x tiling (left) vs. 4x tiling (right).

Frequency Factor

We have the choice in the frequency factor we choose for each new octave.

A factor of 2 means the frequency doubles each new octave, adding more details to an otherwise slowly varying texture.

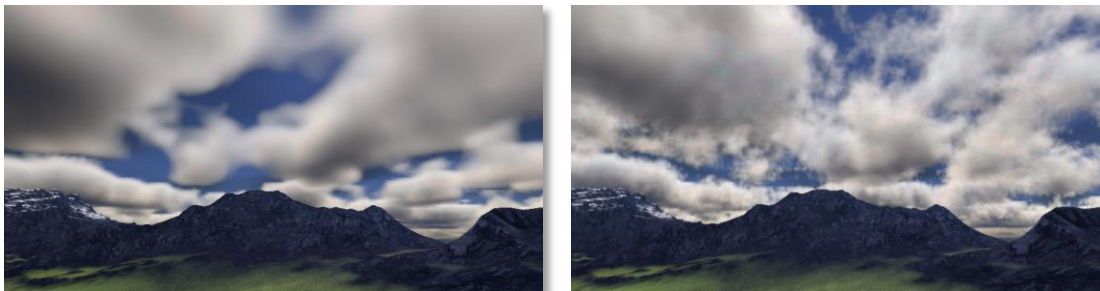


Figure 8-10 – Frequency factor of 2 (left) vs. a factor of 4 (right).
Notice how the cloud patterns change more rapidly and adds more details.

Amplitude Factor

Also, we have the choice in amplitude factor for each new octave.

A factor of 0.5 means the importance of each new octave will be halved.

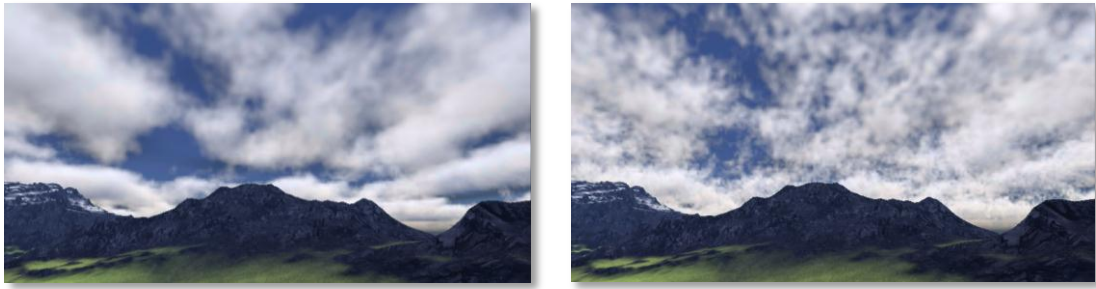


Figure 8-11 – Amplitude factor of 0.5 (left) vs. a factor of 1 (right).
This means all octaves have the same weight.

8.4. Animation Parameters

Wind

As for the fog layer and the 2D cloud layers we saw earlier, you can define the wind force as well as its direction.

Evolution Speed

This parameter helps to specify the speed of each octave relative to the other.

An evolution speed of 2 means the second octave will move twice as fast as the first one, the 3rd octave will move twice as fast as the 2nd one and 4 times as fast as the first one and so on.

This parameter is important to give the illusion that clouds are changing and evolving.

8.5. Advanced Parameters

As explained in the previous Advanced Parameters section of the 2D clouds, cloud lighting is quite complex.

I used simple algorithms for the 2D clouds lighting since we could make the assumption they were thin enough to allow a lot of simplifications but this is no longer the case with 3D clouds, so we're looking at a totally new model here.

Let's try and describe the parameters without entering too much in the details...

8.5.1. Directional Parameters

The cloud lighting is split into 2 parts: directional and isotropic (or ambient).

Directional lighting represents the scattering of Sun light within the cloud in a directional manner, while isotropic lighting represents the scattering of Sun + Sky within the cloud in an isotropic manner (i.e. more uniform scattering).

Directional Albedo

This parameter specifies the percentage of light that is reflected by the clouds. As we saw earlier, clouds are highly reflective so this factor should almost be 100%. The default value is 85%.

Directional Factor

This factor indicates the amount of contribution of direct Sun light.



Figure 8-12 – Turning off the isotropic lighting shows the directional light part.



Figure 8-13 – Turning off the directional lighting shows the isotropic sky light part.

8.5.1. Isotropic Parameters

On the other hand of the lighting equation, we have isotropic lighting by the Sun and the sky. This is a very important aspect of clouds rendering as it accounts for ambient scattering of light within the clouds and it's very difficult to get right but I managed to get a pretty good approximation for it since version 1.1.

Isotropic Albedo

As for the directional part, the isotropic lighting needs an albedo to tell how much isotropic light will

be able to scatter within the cloud. It's a very important parameter as it will guide the appearance of the ambient cloud lighting.

Be careful when setting this parameter not to get unrealistic values: always check with a fully covered sky to ensure your ambient sky/reflected terrain values don't blow up!

Isotropic Factor

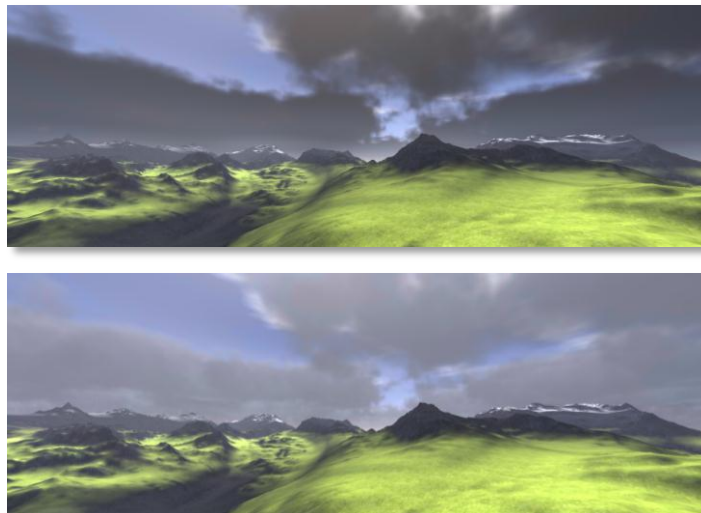
This factor indicates the amount of contribution of isotropic light.

8.5.1. Isotropic Contributions

Now, isotropic lighting is further split into 3 parts for which there are individual parameters too. These 3 parts are: ambient sky light, ambient Sun light and Sun light reflected by the terrain.

Sky Contribution

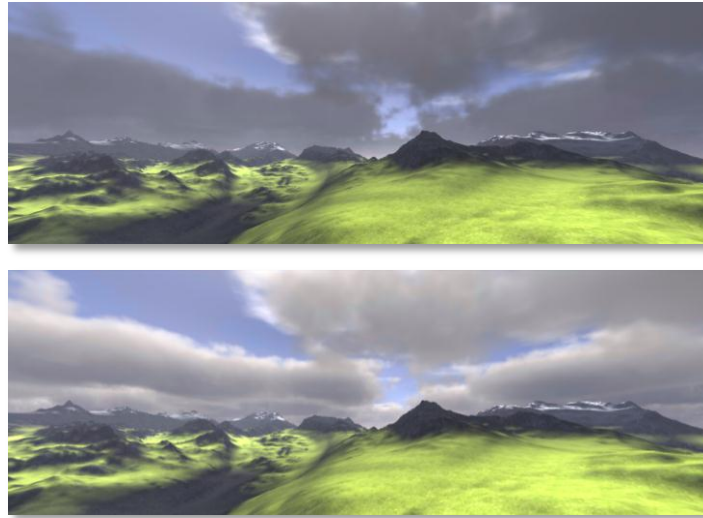
This factor controls the contribution of the sky reflection and diffusion through the clouds. It adds a great visual value to the rendering which would otherwise look too dark.



*Figure 8-14 – The importance of isotropic sky lighting.
No sky light (top) vs. sky light enabled (bottom).*

Sun Contribution

This factor controls the contribution of the isotropic Sun reflection and diffusion through the clouds.

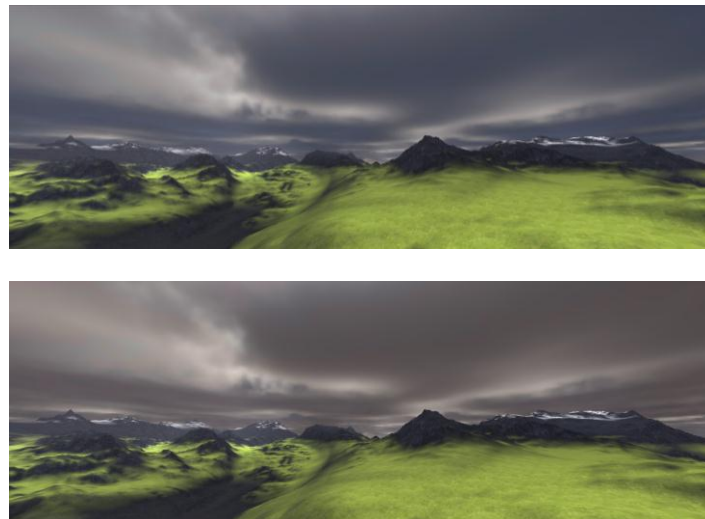


*Figure 8-15 – The importance of isotropic sun lighting.
Sky light only, no Sun light (top) vs. Sun light enabled (bottom).*

Terrain Contribution

This factor controls the contribution of the Sun light's reflection on the terrain that bounces up against the bottom of the clouds.

It uses the terrain albedo seen earlier in the Local Variations Parameters to determine the average color of the reflected light.



*Figure 8-16 – The terrain reflection is especially visible with important cloud coverage,
when almost no direct light source passes through the clouds
No terrain contribution (top) vs. slight contribution (bottom).*

8.5.1. Phase Parameters

Next and final are the “phase parameters”.

Lighting reflection in the 3D clouds is guided by a combination of several phase functions. A phase function indicates how light is being reflected given the angle at which it's being viewed.

We already saw a phase function in the sky module when we spoke about Mie Anisotropy: this is the same for the cloud, except we use not only one but a combination of several phases.

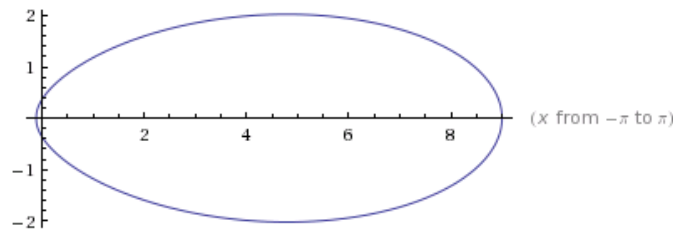


Figure 8-17 – An example of a forward phase function.
Much light is reflected when the camera views straight in the direction of the light source.

There are 4 phase functions in the 3D cloud, each having its own weight and its preferred direction (i.e. anisotropy) :

- **Strong Forward**, this is a tiny peak of direct light that goes forward through the cloud. Its anisotropy is > 0 (i.e. forward) and almost equal to 1. Its weight is very low.
- **Forward**, this is a larger forward peak of light whose isotropy is also > 0 but much less than 1. Its weight is quite low.
- **Backward**, this is a large peak of reflected light whose isotropy is < 0 (i.e. backward). Its weight is a bit higher

NOTE : It's not advised to change these values too much, especially the weights as you can end up with more energy coming out than what entered the cloud in the first place which would result in very bright clouds against a washed out dark sky.

It's very easy to mess everything up, these parameters must be used carefully!

You have been warned. 😊

9. Satellites Module

There are 3 types of satellites supported by *Nuaj'* :

- **Nearby Stars**, this satellite is represented by the 🌟 icon and is typically used to render the Sun.
- **Planetary Body**, this satellite is represented by the 🌕 icon and is typically used to render the Moon.


- **Stellar Background**, this satellite is represented by the  icon and is typically used to render the stellar background with galaxies and distant stuff.



Figure 9-1 – On the importance of satellites...

9.1. Generic Parameters

Before describing the parameters of each type of satellite, we can focus on the parameters that are common to all satellites.

9.1.1. Simple Parameters

Name

You can give a name to your satellites that will help you to identify them. Nothing fancy here.

Distance

This is the distance of the satellite from the planet, in millions of kilometers. This doesn't change the aspect of the satellite but is rather used to sort satellites for rendering, helping to determine which satellite should be in front of the other.

Typical values are 0.3 (i.e. 300,000 km) for the Moon, 185 (i.e. 185,000,000 km) for the Sun and some kind of large value for the stellar background that makes it render first.

Tilt Angle

This angle helps to rotate the satellite's image about its axis.

9.1.2. Revolution Plane

These are used to describe the revolution plane of the satellite.

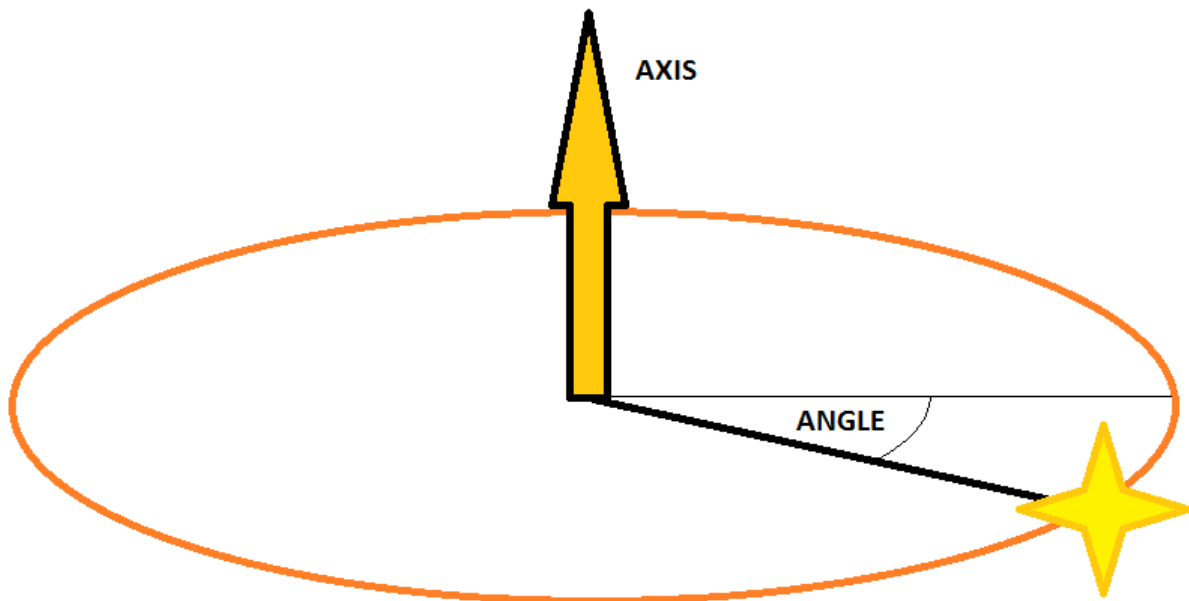


Figure 9-2 – The revolution plane.

The revolution plane is described by an **axis** in Unity's world space. The position on the elliptical orbit is given by an **angle**. There is no specification of radius since the satellite is assumed to be infinitely far from the viewer.

Revolution Axis & Angle

These parameters are quite difficult to enter correctly by hand and are better driven by script, although the initial values when creating a new satellite should allow you to see the Moon, the Sun and the background correctly.

Revolution Period & Time

Another way to describe the angle on the elliptical orbit is to use a **revolution period** : this is the time (in seconds) it takes the satellite to achieve a full 360° revolution on its orbit.

Once the revolution period is set to a non-0 value, then you can specify the time (in seconds again) of the satellite. This is automatically transformed into an angle for you.

For example, if you specify a revolution period of 100 seconds, it will then take the satellite 1 minute and 40 seconds to perform a full rotation, and you specify a time of 25 seconds then it will mean the satellite will be at a quarter of its full rotation, or a 90° angle.

The time specification is quite useful if you want to animate the satellite in real time... But **Nuaj'** also does that for you.

Simulate Cycle

If enabled, this will make the satellite automatically revolve about its axis using NuajTime.

This means that on every frame update, the satellite's time is incremented and the satellite follows its orbit automatically.

Luminance

Each satellite has its own luminance. Satellites are usually textures displayed as a billboard or via a cube map, but the white color needs to be mapped to a HDR luminance somehow. This is where this parameter comes in handy.

- For nearby stars, the luminance can be pretty high. As high as the Sun's luminance described in the Sun Parameters section actually, so we're talking about a magnitude of 10 here.
- For the moon, the luminance is much lower. About 1 for a full moon since the albedo of the Moon is about 0.12 on average, so a full moon reflects $1/10^{\text{th}}$ of the Sun's luminance.
- For the stellar background, this time the luminance is really really faint. By default I use a luminance of 0.1, which is $1/100^{\text{th}}$ that of the Sun ! It also really depends on the cube map you are using actually.

Now let's see the parameters specific to each type of satellite.

9.2. Nearby Star Satellite

As said earlier, this type of satellite is perfect to simulate suns. It renders as an **emissive alpha-blended billboard** facing the camera. Its luminance should be about the same as the one used in the Sun Parameters section.

Emissive

This is the texture slot for the billboard.

Display Size

This is the display size of the billboard, in pixels.

Aspect Ratio

This is the aspect ratio of the billboard that allows you to stretch the quad along its X axis.

Texture Crop

This is a feature I thought might be useful: you can specify a rectangle in UV-space (normalized [0,1] space) where the texture will be sampled.

This is very interesting if you want to animate satellites: you simply need to create a texture page with the different animations and scroll inside that T-page to display the anim frame you need.

Auto-Drive

If you enable the **Sun Drives Satellite** option (default) then the satellite's direction will follow the Sun's direction.

If you enable the **Satellite Drives Sun** option then the Sun will follow the satellite's direction. This can be useful if the satellite is in "Simulate Cycle" mode and you want the Sun to rise and set automatically with time.

Use Sun Luminance

If enabled (default), this simply uses the Sun's luminance specified in *Nuaj'* and overrides any luminance you set for the satellite. This prevents you to have to make the satellite's luminance match the one set in *Nuaj'* every time you change it.

9.3. Planetary Body Satellite

As said earlier, this type of satellite is perfect to simulate moons. It renders as a **diffuse-lit alpha-blended billboard** facing the camera. Its luminance should be about $1/10^{\text{th}}$ of the one used in the Sun Parameters section. Due to the fact the moon has quite a low luminance, it is mostly invisible by day unless on certain conditions (i.e. very low orbit or clear sky that does not reflect much light).

9.3.1. Standard parameters

Albedo

This defines the albedo used for the satellite. The typical albedo for the Moon is 0.12 so you should use a color like (30,30,30) (dark grey).

Diffuse

This is the texture slot for the billboard. If the "simulate lighting" option is disabled, then this texture is used as an emissive texture. Otherwise, it's used as a diffuse texture.

Display Size

This is the display size of the billboard, in pixels.

Aspect Ratio

This is the aspect ratio of the billboard that allows you to stretch the quad along its X axis.

Texture Crop

With this option, you can specify a rectangle in UV-space (normalized [0,1] space) where the texture will be sampled. See the description in Nearby Star Satellite for more information.

9.3.1. Night Light

You have the option to use the satellite as the primary light source at night.

Luminance Factor

If enabled, the satellite will replace the Sun at night and its luminance will be the satellite's

$\text{luminance} * \text{the phase of the Moon} * \text{Luminance Factor}$.

9.3.2. Lighting Simulation

Nuaj' can perform an automatic lighting of the moon for you, provided the diffuse texture you specify is a disc. This option allows you to have a real moon with its phases realistically lit depending on the Sun's position in the sky.

The lighting model used for the Moon is the Oren-Nayar model, which description you can find here : http://en.wikipedia.org/wiki/Oren%E2%80%93Nayar_reflectance_model.



Figure 9-3 – A nice Moon crescent showing up above the horizon.

Enable Lighting

Check to enable lighting simulation (default).

Use Sun Luminance

If enabled (default), then the Sun's luminance is used as light source intensity to perform the lighting otherwise, the satellite's luminance is used.

In any case, the Sun's direction is used to compute the lighting.

Surface Roughness

This indicates the roughness of the satellite's surface.

A value of 0 makes the moon quite smooth, which yields a non-realistic Lambert lighting.

A value of 1 on the other hand, makes the Moon quite rough, which corresponds more to reality as the surface of the Moon is covered with dust that tends to reflect light in all directions but with a factor more complex than the usual cosine-weight of the classical Lambertian model.

Normal

This is an optional normal map to help refine the lighting, you can add some craters!

NOTE: you may have difficulties finding the Moon at night, despite the sky being clear of all clouds.

That may be because, when you enable the lighting simulation, the Moon really simulates waxing and waning so maybe your Sun is behind the Moon?

Try and play with the Sun's azimuth angle in the Sun Parameters section to reveal the satellite, or disable the lighting simulation completely.

9.4. Stellar Background Satellite

As said earlier, this type of satellite is perfect to simulate a stellar background like the Milky Way with plenty of stars. It renders as an **emissive alpha-blended cube map**. Its luminance should be about 1/100th as the one used in the Sun Parameters section. This makes the stellar background very faint and visible only at night where ambient luminance is also very low, otherwise objects of higher luminance (Sun and sun-lit clouds) take precedence.

Emissive

This is the texture slot for the cube map.

Brightness / Contrast / Gamma

These well-known modifiers are applied to the cube map's pixel to tune-up the details of the cube map in real time.

Ambient

The ambient color is very important since it's the main source of lighting at night. This value must be set to a very low value (it's also internally multiplied by the satellite's Luminance) otherwise the night sky will be very bright. It's the perfect value to use to simulate light pollution in a city for example.

NOTE : The alpha component of the ambient color is used as a factor to the color. For example, if we use a color of (255,128,64) then an alpha of 255 will indeed yield an ambient of (255,128,64), but if you use an alpha of 128 then the actual ambient will be (128,64,32) (divided by 2). This helps to have a higher precision.



Figure 9-4 – The effect of adding an ambient night sky color.
(the top image is NOT a joke ! ☺)

9.5. Custom Rendering

If you like, you can render your own custom satellites by hooking to the “*CustomBackgroundClear*” and “*CustomBackgroundRender*” events in NuajManager (cf. [API Manual](#)).

The CustomBackgroundClear event occurs BEFORE satellites are rendered and allows you to override the default clear to black of the background texture so you can perform custom rendering of stars or whatever.

The second event CustomBackgroundRender occurs AFTER satellites have been rendered so you can add other custom types of satellites.

10. NuajTime

The time in **Nuaj'** is not directly the time in Unity but passes through a class called “NuajTime” (cf. [API Manual](#)).

This class is very useful to have a unique entry point to control and fine tune the time used in **Nuaj'**. It stores its own time and, by default, makes time evolve at the same pace as Unity but you can specify a “Time Multiplier” that will allow you to slow down or accelerate the passing of time, or even go back through time.

For example, using a Time Multiplier of 0 will freeze time altogether but using a multiplier of 10 will make clouds and day-cycle evolve 10 times faster than Unity time.

Using a negative multiplier will make **Nuaj'** go back through time.

11. Prefabs

11.1. MapLocator

The **MapLocator** prefab is very useful to precisely locate and scale some maps required by *Nuaj'*.

These maps are typically density maps to locally control the presence or absence of cloud/fog at a given location, but also the terrain emissive map that helps simulating artificial lights from the ground.

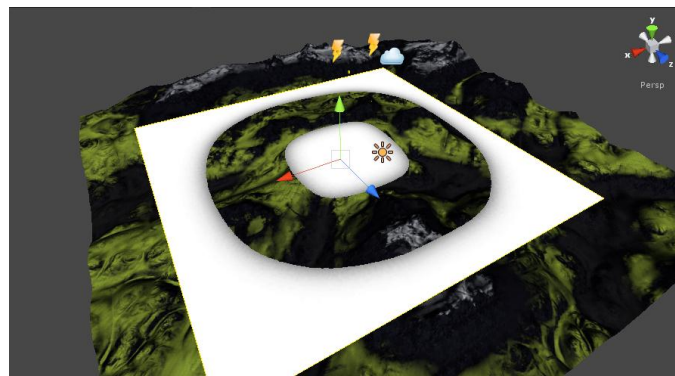


Figure 11-1 – A Map Locator in action, that helps to setup the local cloud coverage map.

You can use the standard Transform (Position, Rotation and Scale) to easily setup the mapped quad.

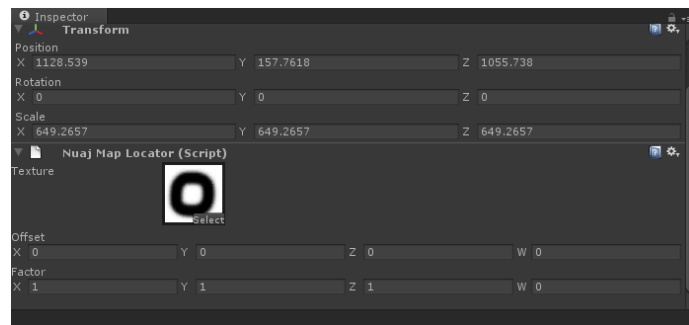


Figure 11-2 – Typical setup.

Texture

This is the texture slot for the local map.

Offset & Factor

These 2 vectors of 4 components (for R,G,B and Alpha) allow you to specify the offset added to the colors of the texture and the factor applied to them.

The exact formula used by the shaders is **TargetColor = Offset + Factor * SourceColorFromTexture**.

So for example, you can invert a texture's colors by specifying an Offset of 1 and a Factor of -1.

Dynamic Control

The advantage of passing through a MapLocator is that you can also perform custom rendering of the local density map or emissive terrain into a RenderTexture of your own and feed it to the MapLocator.

Nuaj' will then use that RenderTexture every frame as a source of dynamic cloud density/terrain lighting.

You can thus achieve pretty interesting effects like clouds gathering all of a sudden due to some magic spell for example, or even paint the clouds dynamically due to some explosion.

11.2. LightningBolt

These prefabs allow you to locate lightning bolts in the scene. Lightning bolts in **Nuaj'** are considered as emissive segments of lights defined by 2 points: Start and End.

To be totally honest, only the Start point is used as a light source at the time though. 😊

NOTE : Special care is taken if you attach a Point Light as the first child of the lightning bolt. This light will also be driven in position and intensity to match the Start position and intensity of the lightning bolt.

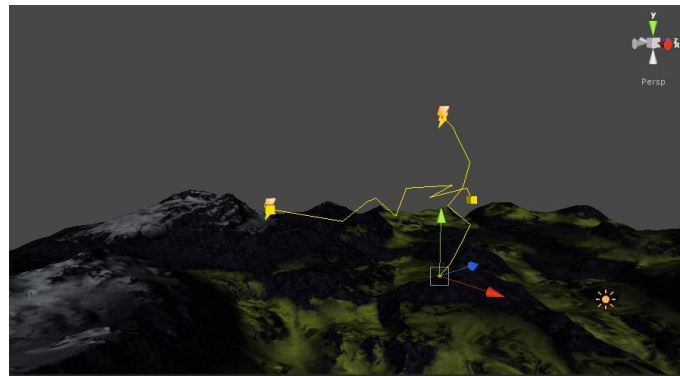


Figure 11-3 – Cute lightning bolts in action.
Each yellow cube can be manipulated relative to the prefab's transform.

You can use the standard Transform (Position, Rotation and Scale) to quickly move and position the lightning bolt's segment wherever you want, but you also have 2 custom manipulators for each of the Start and End points in the form of 2 yellow cubes that help you setup the bolt segment's points relative to the transform. The Start point is visualized by the little ⚡ icon that helps differentiating it from the End point.

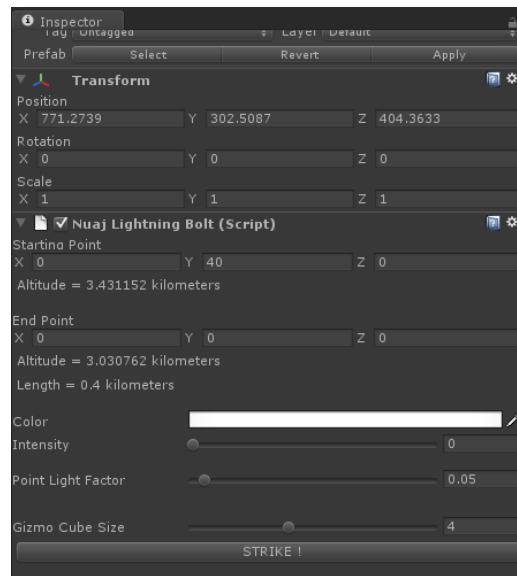


Figure 11-4 – Lightning bolt's parameters.

Start & End Points

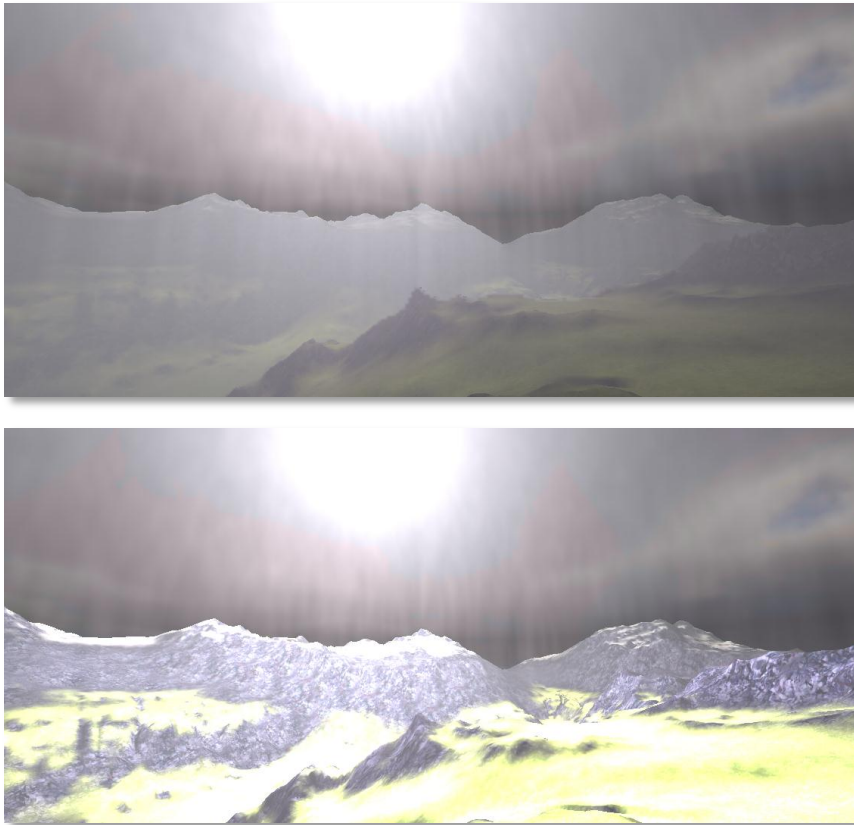
These are 2 vectors defining the position of the Start and End points, relative to the prefab's transform. As a helper, the altitudes in kilometers from the planet's surface are indicated, as well as the length of the lightning bolt segment. This is helpful if you want your bolt to start within a cloud layer at a known altitude.

Color & Intensity

These specify the color and intensity of the lightning bolt. Note that the intensities are defined "per meter" and are accumulated all along the lightning bolt's segment.

Point Light Factor

This allows you to specify the factor to apply to the lightning intensity to obtain the intensity to use in the child point light (if any is attached).



*Figure 11-5 – The effect of Point Light Factor.
A factor of 0 (no point light) (Top).
The default factor of 0.05 (Bottom).*

Gizmo Cube Size

This is simply a GUI parameter that setups the size of the yellow manipulator cubes.

Strike Button

The “Strike !” button can be used to trigger a lightning strike. It actually calls the “StartStrike()” method on the NuajLightningBolt script (cf. [API Manual](#)) that triggers a strike simulation.

A strike is simply a fast variation in intensity starting with a fast discharge and a relatively slow decay of the intensity.

You can follow how the StartStrike() and UpdateStrike() methods are written to create your own lightning strike if you like.

12. Orchestrator

The **Orchestrator** is a tiny but powerful piece of script that can make you become a Storm God with the use of a single slider!

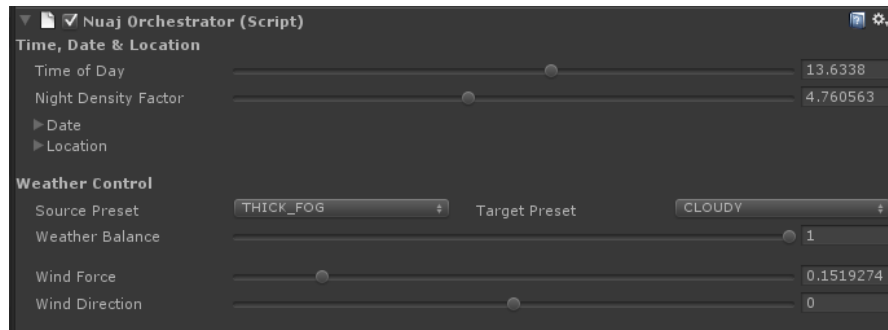


Figure 12-1 – The Orchestrator in action.

Nuaj' with its multiple parameters can be very complex to setup, and the setting of a particular weather condition may require tweaking many different sliders at the same time.

Same can be said for the time of day : creating a convincing sunset not only requires you to lower the Sun toward the horizon but also to increase the Rayleigh and Mie density in the Sky module to make the sky become more red and also decrease the maximum adaptable luminance depending on the time of day.

The Orchestrator lets you do all of these manipulations with a single click !

NOTE : The orchestrator is meant to drive the sky, the fog layer, a single 2D cloud layer and a single 3D cloud layer. It will NOT control additional layers nor will it create the single 2D or 3D cloud layers it needs if they are not already present : you need to create them yourself.

It is divided into 2 very simple sections.

12.1. Time, Date & Location

This section lets you set the time of day as well as the date and location of the observer.

Time of Day

This slider represents the time of day from 0 to 24 hours. By default, the date has been set to June the 21st (Summer Solstice) so the Sun should rise at approximately 5AM and set at 21PM (daylight saving time is NOT taken into account here).

Night Density Factor

You can think of this slider as the “Dramatic Sunset” effect. It tells the increase in air & fog density at sunrise or sunset.

Usually a factor of 2 gives a convincing sunset.

Date Parameters

Well, these are your usual Day & Month settings that may alter the time at which the Sun rises and sets.

Location Parameters

These represent the latitude (between North Pole at 90° and South Pole at -90°) and longitude (between -180° and +180°). By default, the latitude & longitude are set to a typical North European 0°W, 45°N.

NOTE : The time of day setting is relative to a longitude of 0°!

If you increase the longitude (meaning you're travelling west) and don't change the time of day then it will have the effect of time going backward.

Simply imagine that the Sun hasn't risen yet in that western country you're moving to.

12.2. Weather Control

This section is by far the most interesting because it allows you to tweak the entire weather system using a single slider!

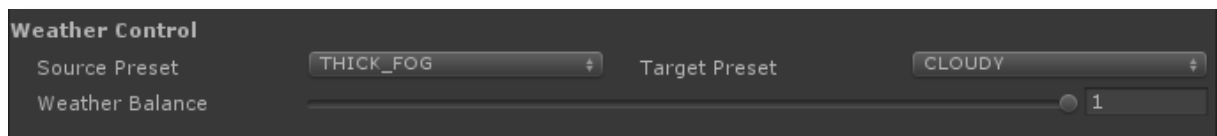


Figure 12-2 – The magic slider !

On the left you have a drop down list that allows you to choose the **source weather** among a list of several presets. On the right, you have another list that allows you to choose the **target weather**.

The slider below helps you to interpolate between the source and target weathers. Simple as that! And it does e-v'ry-thing !

You can easily add more presets using the same model as shown in the script... With a little bit of code, you can even imagine interpolating between 4 different weathers using 3 sliders instead of only one. It's really up to you at this point!

Wind Parameters

This slider allows you to drive the force & direction of the wind for all layers at the same time.

13. Water Reflection

For clouds reflecting in water, I have included a helpful script called “EffectReflectSky” that can be found in the *Nuaj/Scripts/CameraEffects* folder.

This is a camera effect specially created to the “Water4” advanced water effect in the “Water (Pro)” package.



Figure 13-1 – Clouds reflecting in the water

This camera effect is quite cheap as it *does not recompute* the clouds but rather takes advantage of the Fresnel reflection that only reflects the sky at grazing angles to sample the already computed cloud texture from the main camera.

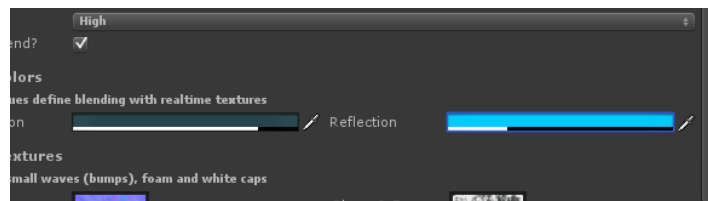
Of course, there are limitations; mainly this effect is designed for reflections of clouds when the camera stands close to the ground and below the clouds.

It doesn't work in altitude, especially when you are inside or above the clouds.

In a general way, the more the rays are perpendicular to the water surface, the more the effect will be wrong as it will need to sample clouds that have not been computed since they are not visible by the main camera used by *Nuaj'*.

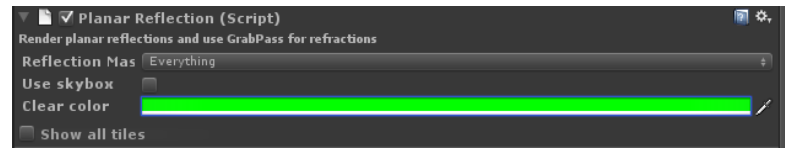
Configuration

- 1) You need to create a water plane using the Water4 (advanced) prefab that uses a Planar Reflection component.
- 2) Adjust the “Reflection” alpha to something less than 255 so the clouds can show



An alpha of 0.25 shows a 75% cloud reflection.

- 3) In the “Planar Reflection” component, clear the “Skybox” checkbox and set the “Clear Color” to a very distinct color that will be used by the reflection effect as a color key (here, a distinct green)



- 4) Now, select the “Reflection Camera” the Water4 prefab created and drop the “EffectReflectSky” script onto the game object



- 5) Assign the Water4 plane object to the “Water Object” field of the “ReffectReflectSky”, the main camera to the “Camera Above” field and it should be working...

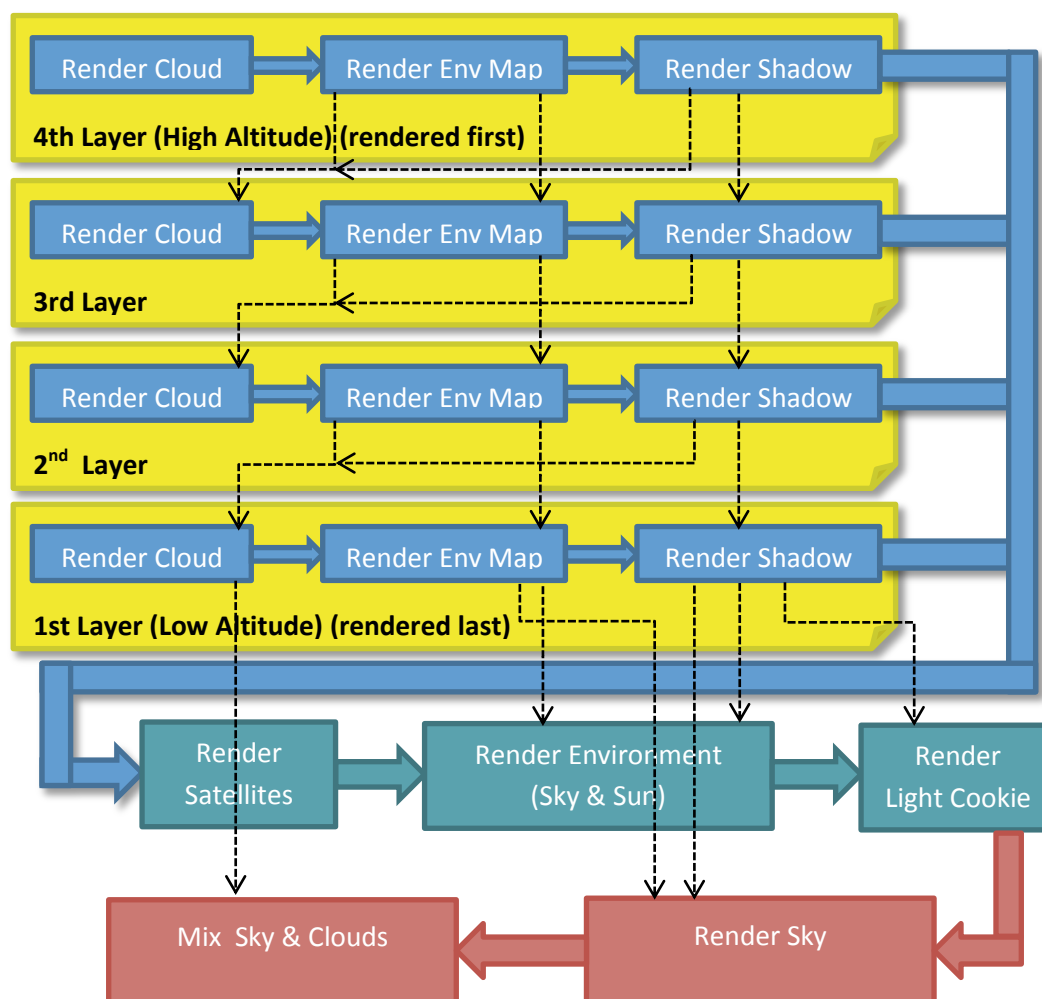
NOTE: You can adjust the altitude of the “Dummy Cloud Layer” used in the effect for the reflection. Ideally, it should be set to the altitude of your lowest cloud layer but it should never be less than the altitude of the camera.

14. Rendering Pipeline

This is the rendering pipeline used in *Nuaj'*.

14.1. Main Pipeline

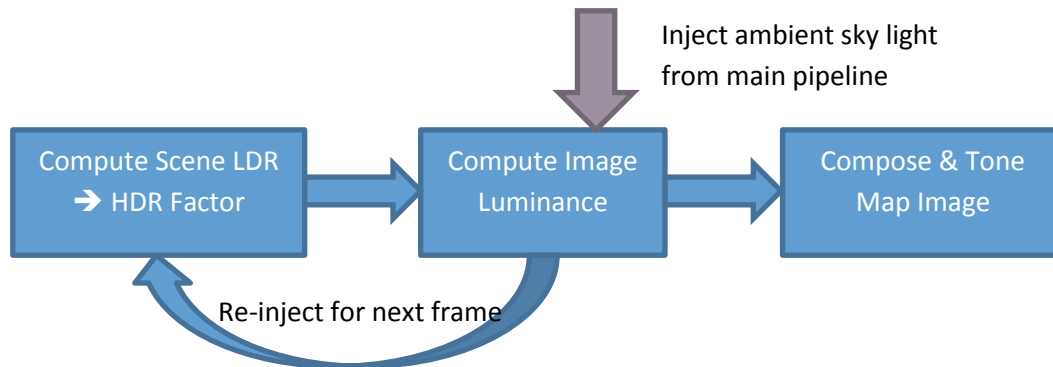
First, the actual clouds & sky rendering:



Notice here how the environment map is also used to light the clouds but also used as ambient light for the Unity scene. This means that if below the clouds you have a scene that reflects light a lot (like snow or high albedo terrain) then this reflection will show at the bottom of the clouds, which will in turn contribute to the next frame's environment map and so on.

You have an actual feedback loop here that can be assimilated to standard radiosity rendering where lighting converges frame after frame to the final rendering, assuming the scene doesn't reflect too much energy, in which case you don't obtain a convergent solution but a divergent one : make sure the scene's Luma Correction and Ambient Color Factor seen in the Tone Mapping Parameters section are not set to too high values.

14.2. Post-processing Pipeline



This is where the scene and sky/clouds are mixed and tone mapped together to yield the final image.

15. Performance

The following performances have been measured with complex sky, fog, a single 2D and a single 3D cloud layer enabled:

- 3ms on a GeForce 285 GTX
- 6.58ms on a GeForce 320M (mobile laptop)

Nuaj' posts the equivalent of 30 draw calls per frame in average.

Due to volumetric effects in sky and volume clouds computation, performance can suffer a lot if you render at incorrect resolutions or if you use too many marching steps.

On the positive side though, *Nuaj'* doesn't take more time to render if the sky is more or less covered with clouds as is usually the case with solutions using particles.

Nuaj' is very scalable in that it lets you choose which method to use for rendering but also the resolution and quality to use.

15.1. CPU Read Back

The major performance drawback in *Nuaj'* certainly comes from the CPU read back. The CPU read back of GPU values eats a lot (!!) of time.

In my experience, it's possible to quickly read tiny 1x1 textures without damaging the framerate at all using special tricks like [http://msdn.microsoft.com/en-us/library/windows/desktop/bb205132\(v=vs.85\).aspx#Accessing](http://msdn.microsoft.com/en-us/library/windows/desktop/bb205132(v=vs.85).aspx#Accessing) for example, but it seems Unity does its own thing internally and ruins the triple-buffer optimization.

I centralized CPU read back at a single place so it's performed only once, although 3 values need to be read back:

- The image luminance, for luminance adaptation and tone mapping. This value is read back if the Luminance Computation Type is set to DOWNSCALE or DOWNSCALE LOG.
- The Sun Color & Sky Ambient Color, which are read back if Software Environment rendering is disabled.

If any of these values is needed, then CPU read back occurs at every frame.

I also tried to perform read back every few frames but that really feels like the camera has annoying hiccups so it's read back every frame or no read back at all.

In order to prevent CPU read back altogether, you need to disable both luminance downsampling and hardware environment rendering.

NOTE : Downsampling algorithms are quite expensive in terms of computation time but are essential to a correct luminance adaptation.

What is expensive is not really the downsampling part that reduces the screen buffer down to a 1x1 value but rather the fact that I lock the 1x1 texture to read it back with the CPU !

This is never a good idea in general but it cannot be prevented here, unless you change your entire lighting pipeline (you should contact me so I tell you how to do that if you're interested).

15.2. Software Environment Rendering

By enabling the software environment rendering, you have a less accurate environment lighting but this can increase performance a lot since the GPU is no more stalled to read back environment color values with the CPU (but you have to disable luminance downscaling too!).

You can also perform your own custom environment rendering using the "CustomEnvironmentRender" event (cf. [API Manual](#)).

15.3. Manual Luminance Input

You can gain some more performance by disabling the luminance adaptation and choosing the CUSTOM setting. By doing so, you will need to provide your own luminance level by script, setting the "NuajManager.ImmediateImageLuminance" property (cf. [API Manual](#)).

It is quite easy to have custom luminance values in an array that you will fetch depending on the time of day or weather conditions.

This little optimization will save you the cost of downscaling the image AND reading back the luminance with the CPU, preventing any GPU stall (but you have to disable hardware environment rendering too!).

Downsampling only once every few frames may also seem a convenient solution but the main problem here is not the downsampling itself but the CPU read back (again!).

15.4. Upsampling Techniques

Another optimization to use is to choose the “BILINEAR”, “CUTOUT” or “SMART” upsampling technique whenever possible.

Of course they are cheaper... but also uglier. ☹

15.5. Increasing Sky Rendering Performance

You should use the simple sky model whenever possible. This prevents you from flying too high and you can't escape to outer space with that model but it is good enough if you intend to stay on the ground.

Also, it lacks god rays and is really uglier but these are eye candy and are not essential, although they also convey a more natural look to the sky as clouds cast their shadow not only on the ground but also the atmosphere below them.

If you choose to use the complex sky model anyway, you can save some time at the expense of god rays resolution by setting the “God Rays Min Steps Count” to 0 so the god rays are traced with whatever steps are left, and not forced to a high steps count.

Then, the rendering speed is mainly controlled by the amount of Sky Steps Count. You can set these to a very low value but you will certainly notice a change in color and quality.

Finally, you can decrease the downsample factor to something like 0.125 that will have the effect to render the sky at $1/8^{\text{th}}$ the screen resolution. This is quite alright since the sky changes very slowly but if you are not using the ACCURATE upsample technique then you might notice artifacts at pixels where the sky and scene meet.

15.6. Layer Clouds Performance

These clouds are as simple as they can be. There is no clear way of improving any performance loss except a little trick that consists in disabling the “Cast Shadow” option.

Indeed, these layer clouds are usually high altitude clouds and are also often very thin so the amount of shadows they are casting to the layers beneath is negligible. Thus, disabling the shadow casting can save a small amount of time and should have little or no impact on the final rendering.

15.7. Volume Clouds Performance

These clouds can be quite time consuming due to their volumetric nature.

Once again, the main parameter to tweak to improve performance is the amount of marching steps. The more the better of course, but you can go as low as 16 and might still have a nice look for clouds that are not too thick.

Finally, you can use only one deep shadow map layer instead of the default 2-layers setting (that saves quite some time) and disable the “Smooth Shadow Map” option, at the expense of shadow and god rays quality of course.

15.8. Satellites Performance

Satellites render very fast, there is no clear way of optimizing their performance except disabling them altogether. You could disable the stellar background during day time and the sun during night time for example.

NOTE : Although there should be a small or no impact on framerate, I noticed slowdowns when enabling the star background cubemap on Mac so perhaps you should disable it during daytime.

16. Troubleshooting


First, here are the minimum requirements for *Nuaj'* to function properly :

- You need Unity Pro v3.0 or more recent
- You need a graphic card that supports Shader Models 3.0 (SM3)

If importing the *Nuaj'* package caused some errors or you have rendering problems, maybe you can find the solution listed here :

The Nuaj' component on the Nuaj prefab is disabled

This usually comes from the fact that one or several modules in *Nuaj'* didn't turn on correctly, probably because of faulty materials.

You can check the modules by clicking on each tab in the component, if a tab shows an information panel with a red  icon then the corresponding module failed to initialize, probably because of a wrong material or shader.

Unity is never very eloquent with shader errors so you won't have much detail about the problem except that “the shader doesn't work on your machine”. And that's probably the case anyway, unfortunately there is not much to do at this point except sending a complaint e-mail to support@nuaj.net, indicating the problem and the nature of your hardware as well as the Unity version you are using. Hopefully a solution will be found very soon.

My particle FX don't show !

Nuaj' is a post-process and, as such, overrides whatever is written to the screen before it.

There is no clear way of correctly mixing FX and volumetric rendering from **Nuaj'**, the only true way is to render FXs in another target then add this target to the **Nuaj'** rendering, using an ImageEffect on top of **Nuaj'** own effect.

To do this, you need to create a second camera with a Target Texture and render your FX with it, then create an image effect that will mix the FX's texture on top of **Nuaj'** rendering.

17. Acknowledgments

My best thanks to Guillaume for his great terrain, the access to his assets server and for making me drive 30km to debug on his machine by a dreadful fog (I had firsthand experience of what a real volumetric fog was that night ☺).

Sonia for the cute website of course.

Laurent Le Brun, for writing a special version of his Shader Minifier just for me ♥.

Piotr for his squad routine.

Valérian, David, Géraldine for helping me debugging on ATI and Mac.

Fulbert, for lending me his Mac for my last chance debug.

And, last but not least, Gaël for eating my food, drinking my wine, helping in the debug process and making the key images for the asset store subscription.

Vague hellos to all my others "friends" who didn't even notice I had disappeared for 3 months writing **Nuaj'**. ☺