SPACE NEBULA AND STARFIELD V3

PLUGIN FOR UNREAL ENGINE
THAUROS-DEVELOPMENT

✧

→ DRAFT ← → CONSTRUCTIONZONE ← → WIP ←

Based on Version 3.24 (As part of Space-Nebula and Starfield Plugin V2.201)

Space Nebula and Starfield V2 Documentation

This tutorial gives you a quick introduction into the Plugin and how to use it.

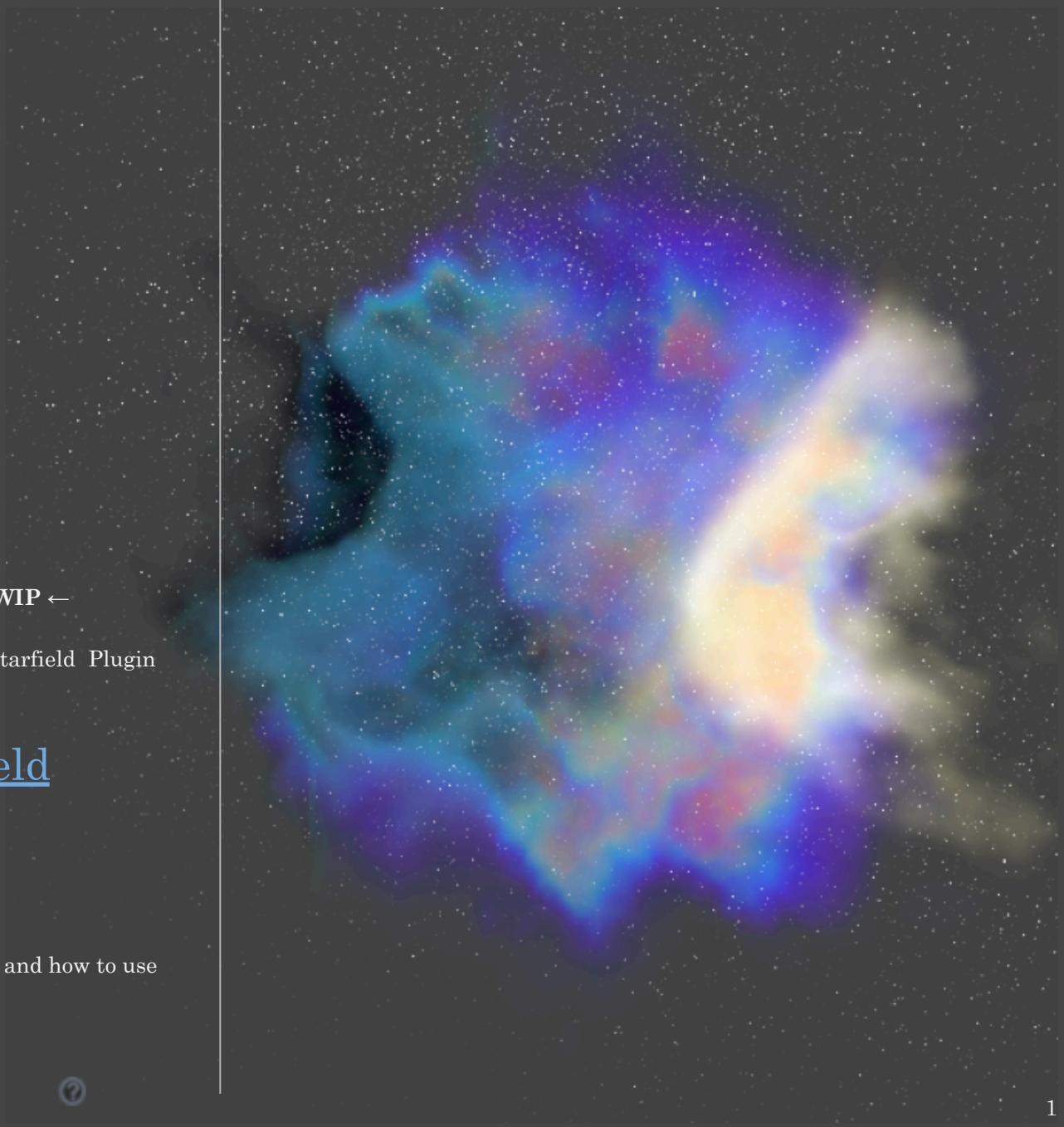
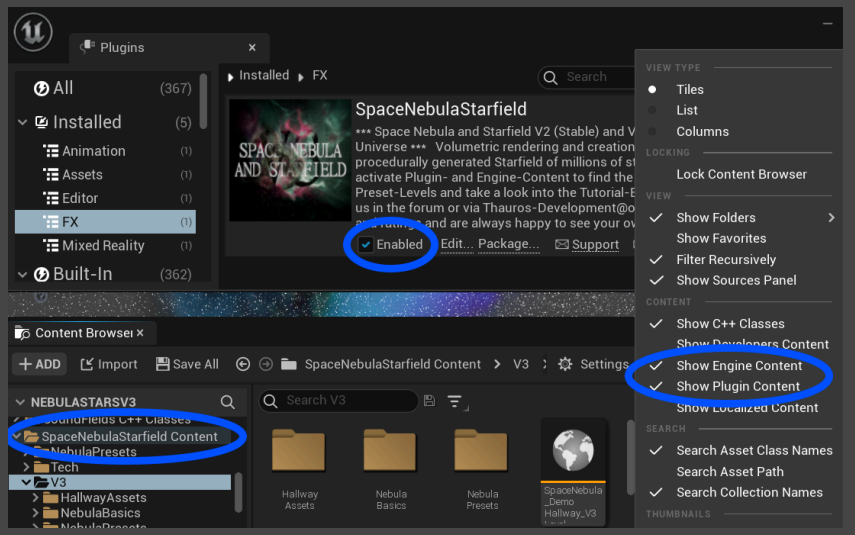


Table of contents:

1. Installation

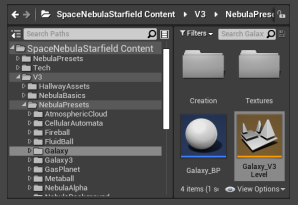
When the Plugin is installed though the Epic-Games Launcher, it is placed inside your Engine-folder. Ensure in your Project that the Plugin is enabled in the Plugins-Browser. To find all the Example-Content enable “Show Engine Content” and “Show Plugins Content” in the View-Settings of the Asset-Browser.



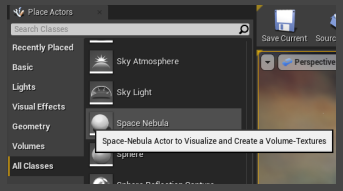
You can also paste the Plugin into your Project’s “/Plugins/”-folder. It is then Active by default and just “Show Plugin Content” has to be enabled.

2. Space-Nebula

A Space-Nebula in this Plugin is an **Actor** which you can freely place your Level as an Object. The `SpaceNebula`-Actor has the general abilities to **create and visualize Volume-Textures**.



Many **Example-Presets** are available as Levels and Blueprints to jump directly into Space and beyond.



You can also find the base-class of the `SpaceNebula`-Actor in the "Place Actors" tab and derive a Blueprint child-class from this like the Example-Presets do.

2.1. Overview

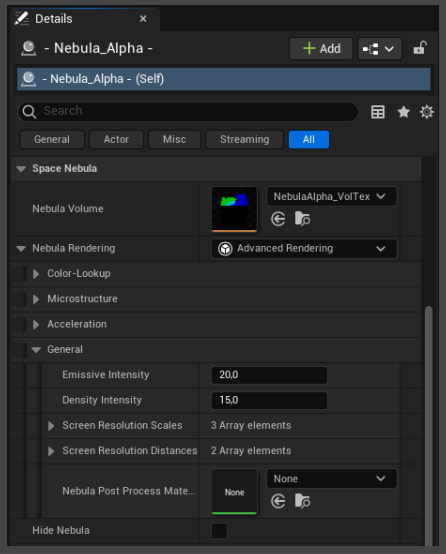
Notes on V2 vs V3
while V2 is the Blueprint Edition of the Plugin, V3 is the C++ implementation gathering more speed, usability and possibilities. Introducing a modular Rendering-System and a Creation-Pipeline with customizable and randomizable Stages (introducing Material-Compute-Shaders), the SpaceNebulaV3-Plugin is more like a general Tool for Creating and Visualizing real Volumetric-Textures.

2.2. Visualize Nebulas

Visualize Nebulas
Every Space-Nebula has a Volume-Texture to specify the Color- and Density- distribution in 3D-Space. Inside the Level, the Nebula is a Bounding-Cube but can freely be placed, rotated and scaled/stretched. In the `SpaceNebula` Actor, a `NebulaRendering` is set-up to specify the visualization. Different kinds of `NebulaRenderings` are available for simple to advanced purposes.

> Info: Volume-Raymarching is the common approach to visualize Volume-Textures to display semi-transparent, surfaceless shapes in 3D-Space.
>
> In respect to a certain View, a 3D-Texture is projected onto the 2D-Screen. To achieve this, a straight line from the View is shot through each Pixel of the Screen and further through Space. The final Screen-Color gets accumulated by sampling the Color and Density properties, while marching along this line in well-defined step-sizes.

`Nebula-Volume` is the actual Volume-Texture asset defining the content of the 3D-space. It is either stored on disk (Asset-Browser) or procedurally created using the Space-Nebula Creation utility.



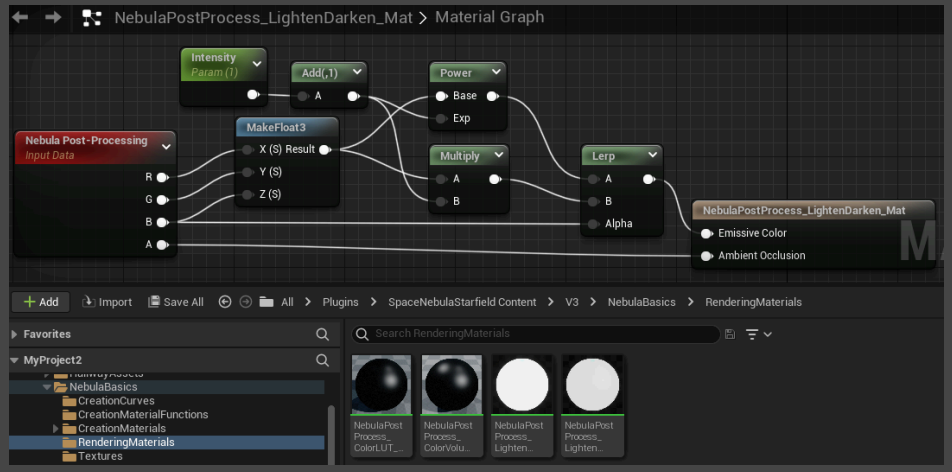
`Density-Intensity` scales the Absorption properties of the entire 3D-structure. Increasing this makes the Nebula darker and more dense.

`Emissive-Intensity` scales the overall light emission. Increasing this makes the Nebula more bright.

`Screen-Resolution` settings are performance related controls. The `Screen-Resolution` is scaled down

when the camera is close to the Nebula (due to limited volume-resolution) and scaled up when further away (to still show all details in the distance). The `Distances` parameters will define when to switch between `Near-`, `Mid-` and `Far-Scales`.

`Nebula-Post-Process-Material` can be set optionally to modify the appearance of the Nebulas in a separate, post-raymarching pass. This can be useful to e.g. increase contrast or apply color-grading (LuT) at full screen-resolution. When setting a Material for a single Nebula, then the effect gets applied to all Nebulas in the Level.



Note: Translucent Materials get drawn after the Nebula-Rendering, so close objects (e.g. cockpit glass) cover Nebulas correctly. Otherwise consider `Opaque` and `Dither Opacity Mask` to cover a Translucent Object by the Nebulas.

2.2.1. Simple Rendering

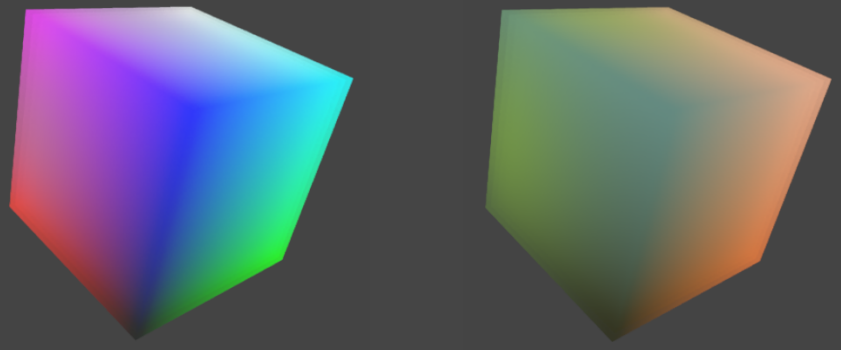
This Rendering method is a simple Volume-Raymarcher, displaying the Volume-Texture as its RGB-Colors and using the Alpha-Channel for Density. The Sampling Stepsize is…

2.2.2. Advanced Rendering

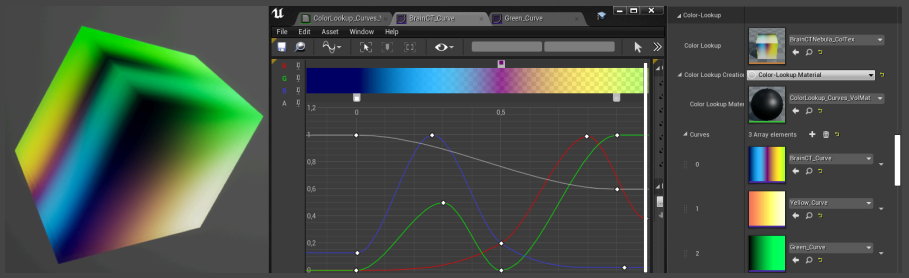
The Advanced Rendering method is similar to the simple Raymarching, though it uses additional methods to specify the visual appearance and improve performance.

2.2.2.1. Color-Lookup

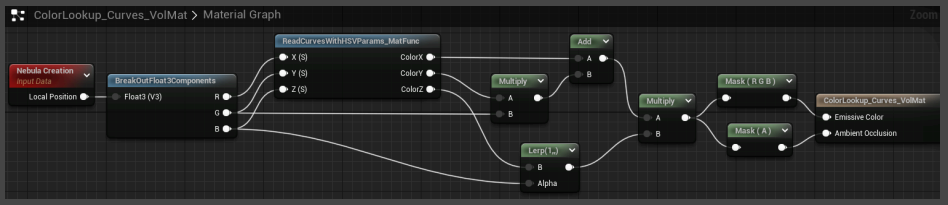
At Rendering the RGB channels of the Nebula-Texture are used to look-up the actual color from an additional Color-Lookup texture. This concept may be familiar as Transfer-Function (mapping a grayscale onto a Color-Curve) or Color-Grading using LUTs.



Generally the Color-Lookup specifies different particles as Emissive (RGB) and Absorbing (Alpha) properties, while the Nebula-Texture specifies for each Voxel what type of particle (RGB) and how much (Alpha) it represents. A Color-Lookup Volume-Texture can be created with Materials (see 2.4.2.1 - Nebula-Creation-Material), e.g. allowing the utilization of Color-Curve assets for composing.



Almost any example Nebula is using the ColorLookup_Curves_VolMat which combines three Color-Curves into a Volume-Texture. But it is totally valid to modify/exchange this Material to specify the emissive and absorptive properties of the 3D structure.



2.2.2.2. Microstructure

Microstructure

2.2.2.3. Acceleration

Acceleration

> Important: The Acceleration considers the Density- and Emissive-Intensity parameters as well as the Microstructure. So when changing these settings, re-creation of the Nebula/Acceleration is recommended.

2.3. Block Rendering

Block Rendering

2.3.1. Signed-Distancefield Rendering

Signed Distancefield Rendering

The Signed-Distancefield-Rendering works a bit different than the classical approach of Volume-Raymarching. Thereby the Density-Channel of the Volume-Texture describes an approximation of a Surface which is not constrained by the Volume-Resolution. The Nebula-Rendering will induce Density near and inside the Surface, which gives more details than the usual Rendering-Method. Also, since the most change in structure is expected at its surface, the Signed-Distancefield deals directly as sampling Acceleration to systematically skip empty areas for performance reasons. So, no need of an Acceleration-Stage and no Micro-Structure required.

Though the designing of structure is restricted to output Distance not Density. In the Nebula-Creation-Material, a Signed-Distancefield can be designed e.g. by describing various Shapes (see "DistanceField" Material-Functions) or a Voronoi-Noise. As Result, each point in space holds the distance to its nearest surface, with a negative sign when the pixel is inside the structure.

2.3.2. Within Rendering

Within Rendering

2.4. Create Nebulas

Create Nebulas

Serialization:

By default, the Volume-Creation does process one Creation-Stage each Frame and will block the Game-Thread. This may cause hitches or longer durating drops in Frame-Rate, especially when excecuting complex scenarios like high-quality Flow-Simulation.

With the `Serialization` parameter set to a larger value, GPU-work gets split into multiple parts and the Game-Thread will not stall for the Creation-Stage to finish, maintaining a constant framerate during Nebula-Creation. Though, the overall Creation-Process will take longer and results may be visible at later frames.

bHDR: By default the Nebula-Volume-Texture uses 8-bit values to store Pixel-values in a range of 0 to 1. bHDR decides to use 16-bit values instead to allow more different values, also <0 and >1.

2.4.1. Creation-Pipeline

Creation Pipeline

2.4.2. Creation-Stages

Creation Stages

2.4.2.1. Nebula-Creation-Material

The most used and useful Creation-Stage is using a Material-Asset to define the Color and Density at each 3D-point (Voxel) of the Volume (Grid). The procedural composition of 3D structures comes with the mathematical definition of Shapes, Noises and Gradients, with respect to the Local- or World-Position of each Voxel and laid out in four channels of RGBA.

> Note: While Materials in Unreal are Vertex-/Pixel-Shaders, the Nebula-Creation-Materials are Compute-Shaders and so do straight-forward GPU utilization for Volume-Creation.

> Important: In the Material-Editor Texture-based Noise-Types are currently not supported for Nebula-Creation-Materials! Use Computational Noises instead.

2.4.2.2. Lightsource

Lightsource
(SceneOcclusion struggles with Nebula-Scale)

2.4.2.3. Nebula-Brush

Nebula Brush

2.4.2.4. Flow-Simulation

Flow Simulation

2.4.2.5. Mean-/Median-Blur

3D Blur

2.4.2.6. Cellular-Automatas

Cellular Automatas
Cyclic Cellular Automata
Cellular Nonlinear Network

2.4.3. Bake Volume-Textures

Volume Texture Bake

2.4.4. Randomizer

Use the Randomizer utility to design a procedural set of Nebulas to be created at runtime. Most Nebula Creation-Stages do have a List of Parameters for Randomization. The list can be automatically filled by using the `Recollect-Randomization` button/function and will then mirror all design-related Parameters of the Creation-Stage with Min-, Max- and Default-Values. For Nebula-Creation-Materials, all the Scalar-Material-Parameters are gathered and their Slider Min and Slider Max values are for the Randomize Min and Max. This allows active designing of the procedural Nebula-Creation within Material and the Randomize-Parameters Lists. Note that changing a Stage's actual parameter may have no effect since it may get overwritten by the randomized parameter from the List.

Every time the `Randomize` button/function is used, a Random-Seed is applied to derive all randomized Parameters from a Random-Stream internally. A positive Seed-Number will cause the Randomizer to apply this specific Seed on the Parameters. When the Seed is a negative number, the Seed will be randomized on Randomization so you can use that Seed without the negative sign to restore a certain randomization.

2.4.5. Particles and Volumetrics

2.4.5.1. Niagara-Nebula

Niagara-Nebula: This example shows how a Nebula can be used to feed a Niagara-System for Particle-Distribution.

Inside the Niagara-System, the Particles itself are made of 2D-Textures while the Volume-Texture from the Nebula-Creation is read to define Particle-Color
and -Opacity.

2.4.5.2. Niagara-Vectorfield

Niagara-Vectorfield: In this Example a Nebula-Actor is used to dynamically create Vectorfield-Textures for the
Niagara-Particle-System. Inside Niagara, a Force is applied to each Particle which is directly read from a Volume-Texture.

This Vectorfield Volume-Texture gets created from a single Nebula-Material-Creation-Stage. In this Nebula-Material, a Gravity and a Vector-Noise with spherical direction are added together. Also an Attractor towards a Blueprint-specified position is added to the Vectorfield.

The Nebula-Rendering is set to Simple to visualize the direction of the Vectorfield-content in Color-values for debugging.

2.4.5.3. Volumetric-Fog and -Clouds

You can create Volume-Textures with a Space-Nebula to apply as Texture-Parameter in any Material. This way, a baked or runtime changing Volume-Texture can be induced into Unreal's Volumetric-Fog and -Cloud systems.

2.5. Pick Nebulas

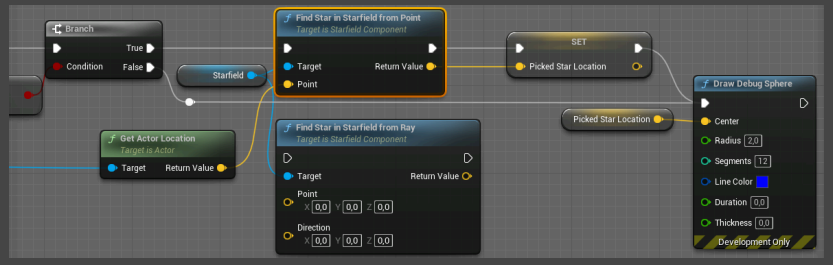
Pick Color, Density and Gradient

3. Starfield

Add a `Starfield-Component` to your Level for a 3D-cluster with a gazillion of stars. From the "unlimited" amount of potential stars, only the

actual visible stars are considered for performance.

The visualization of stars does apply a simple photon-model, spreading photons accordingly to a gaussian-distribution in screen-space. This represents kind of a physical approach and is useful to make the appearance persistent with Temporal-Anti-Aliasing and other Post-Process effects like Bloom (try Convolutional-Kernels) and Motion-Blur.
Internally stars get placed pseudo-randomly and locations can be gathered, e.g. to spawn space-objects when close, to be a consistent universe. By adding elements to the List of Closest-Star-Locations, these get automatically filled with the stars near the camera. In Blueprint stars near a point or ray within the Starfield can also be gathered by using FindStarInStarfieldFromPoint or FindStarInStarfieldFromRay.



4. More Tips

Tips and Tricks

5. Example Presets

Example Presets

5.1. Nebula-Alpha, -Zed, -Omega, -Kappa

5.2. Galaxy, Sun, Atmospheric-Cloud, Polar-Lights, Rainbow

5.3. Basic example for painting Nebula and Flow

5.4. Basic examples of applying Nebula- and Vectorfield-Volumes into Niagara

5.5. Procedural Starfield

5.6. Import examples of Brain-CT and Mitch-Meyer's Cloud

5.7. Signed-Distance Field Nebula-Sigma, Gas-Planet, Metaballs, Water-Waves

5.8. Cellular-Automata, Cyclic-Cellular-Automata and Cellular-Nonlinear-Network