DEEPSKY HAZE USER GUIDE

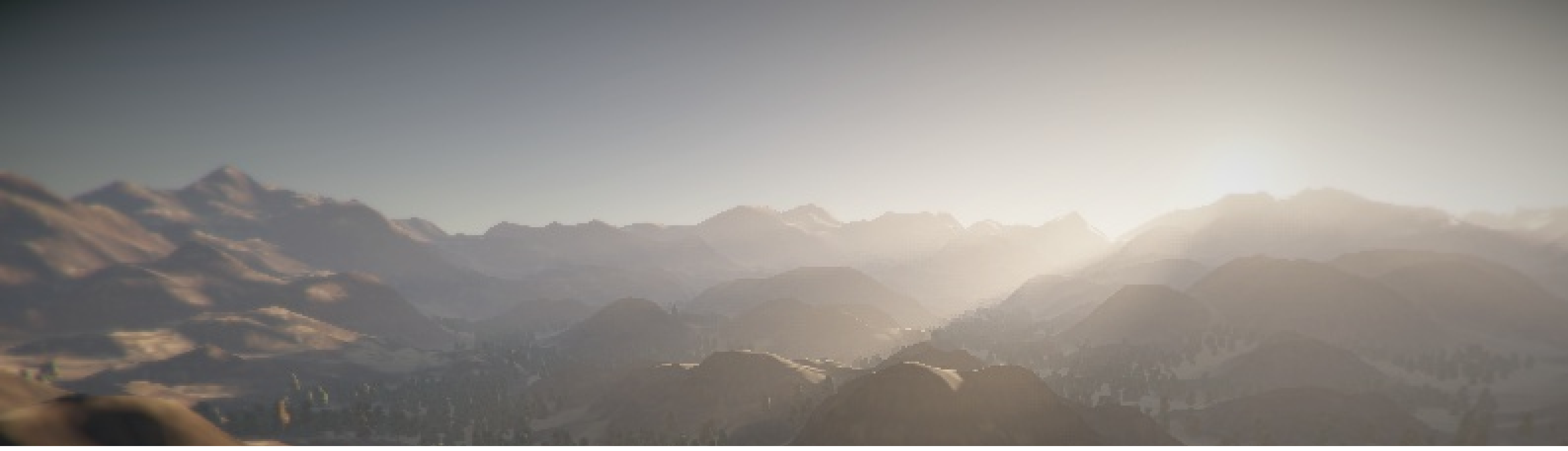
V1.3.3

CONTACT

For help/support and general questions please do get in touch: piranlbb@gmail.com

Will endeavour to reply within 48 hours, please be aware of time zone differences if you're not in GMT. You can also find me on Twitter for updates: @PiranLBB

CONTENTS



INSTALLATION

DeepSky Haze can be installed anywhere within your project Assets folder, but please be aware of the following:

The default location for saving Contexts is **<path under Assets>/DeepSky Haze/Contexts** and will be created if it doesn't exist. Note that renaming the main DeepSky Haze folder will cause an error if you then try to save a Context.

Unfortunately due to the way the shader compiler searches for shader include files, there is one hard-coded path in the file **DeepSky Haze/Resources/DS_TransparentLib.cginc** - if you install DeepSky Haze to a custom folder (eg. Assets/Packages/DeepSky Haze) then you will need to change the path to **DS_Lib.cginc**. For example:

From: **#include "Assets/DeepSky Haze/Resources/DS_Lib.cginc"**
To: **#include "Assets/Packages/DeepSky Haze/Resources/DS_Lib.cginc"**

This is so the shader compiler can always find the required library files, while allowing you to place custom transparent shaders whereever you like within the project.

ATMOSPHERIC SCATTERING OVERVIEW

If you are already familiar with how atmospheric scattering works you can skip this section. Otherwise, here’s some background on the types of light scattering simulated by DeepSky Haze.

As light travels through the atmosphere, it can hit and be scattered by the various particles in the air; some of this light can then end up travelling towards our view. When the particles are very small, the amount of scattering changes for different wavelengths of light. This causes the classic blue colour of the sky: blue wavelengths scatter more than green and red so more of the blue light is bounced into our view. As the light travels through more of the atmosphere at sunrise/sunset the blue light is almost completely scattered away, resulting in classic red/orange skies. This type of scattering is known as Rayleigh scattering, and accounts for the blue colour of distant objects as well. In DeepSky Haze this is controlled by the Air parameters.

When the particles are large (such as grains of dust or water vapour) then the scattering is not dependent on wavelength and the light is scattered as a whole. This is the white/yellow ‘haze’ seen around the sun on a dusty day and is typically found closer to the ground (below ~1500 metres). This is known as Mie scattering, and is controlled by the Haze parameters.

The amount of scattering can also depend on the angle between the direction the light is travelling in and the direction of the view. Mie scattering is strongly anisotropic, meaning it is more pronounced in one direction or another. Light scattered more towards the direction it was original travelling is known as forward scattering. Back scattering is when the light is scattered in the opposite direction back towards the light source. Rayleigh scattering isn’t very anisotropic so for performance it is considered to scatter equally in all directions (isotropic scattering).

In addition to the large scale atmospheric light scattering, DeepSky Haze can also simulate the local scattering from spot and point light sources, such as in a smoky room. This is calculated in exactly the same physically-based manner as the atmospheric haze, using the same anisotropic Mie scattering method.

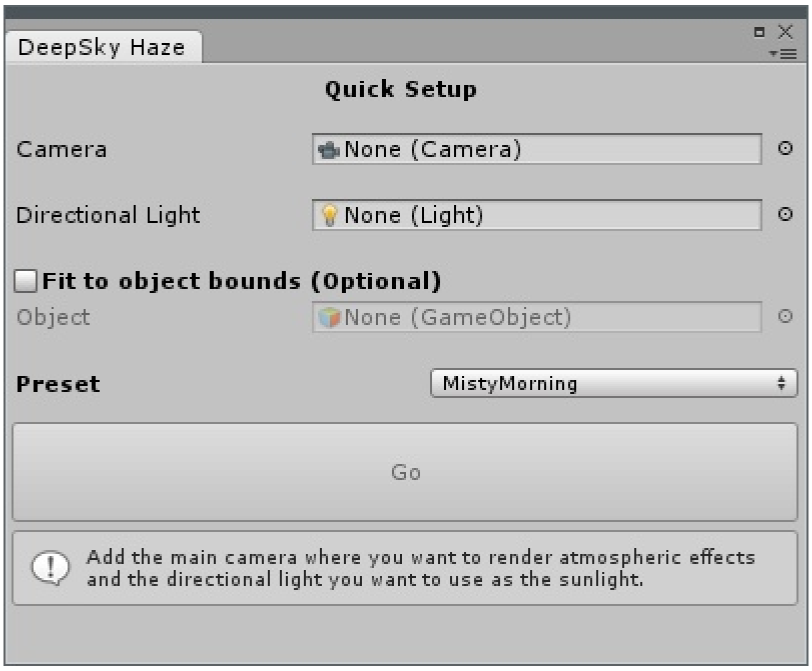
GETTING STARTED

Setting up DeepSky Haze in your scene is easy. The next two sections will show how; firstly using the Quick Setup option and secondly by creating the required GameObjects and Components by hand. By the end of these sections you should be able to apply DeepSky Haze atmospherics to any scene in your project, and have a general overview of how the system works.

QUICK SETUP

Applying atmospherics to an existing scene is easy. Open the Quick Setup window using the GameObject -> DeepSky Haze -> Create Quick Setup menu option.

The Quick Setup window will popup. The options are:



Add the main camera where you want to render atmospheric effects and the directional light you want to use as the sunlight.

Camera - drag the camera from your hierarchy (or select from the scene by clicking the small circle icon) which will be rendering the atmospherics. This should be your main camera (you can setup additional cameras later).
Directional Light - drag in the main directional light you are using as the sun/moon/etc. This should be light being used to cast realtime shadows. All lighting values for the air and haze scattering will be physically-based and driven by this light source.

Fit to object bounds (Optional) - if you would like the initial atmospherics Zone to be scaled to enclose your terrain or a certain object, tick this box.
Object - the initial atmospherics Zone will be fitted to completely enclose this object, plus a little extra height at the top. You can drag in any Terrain or GameObject with a MeshRenderer component.

Preset - choose from the existing atmospheric settings to apply as a starting point for your own effects. Note: you can add your own saved presets to the DeepSky Haze -> Contexts folder to have them appear in this dropdown.

Once you've added a camera, a directional light and optionally a terrain or mesh object, simply press the big Go button to add the setup to your scene.

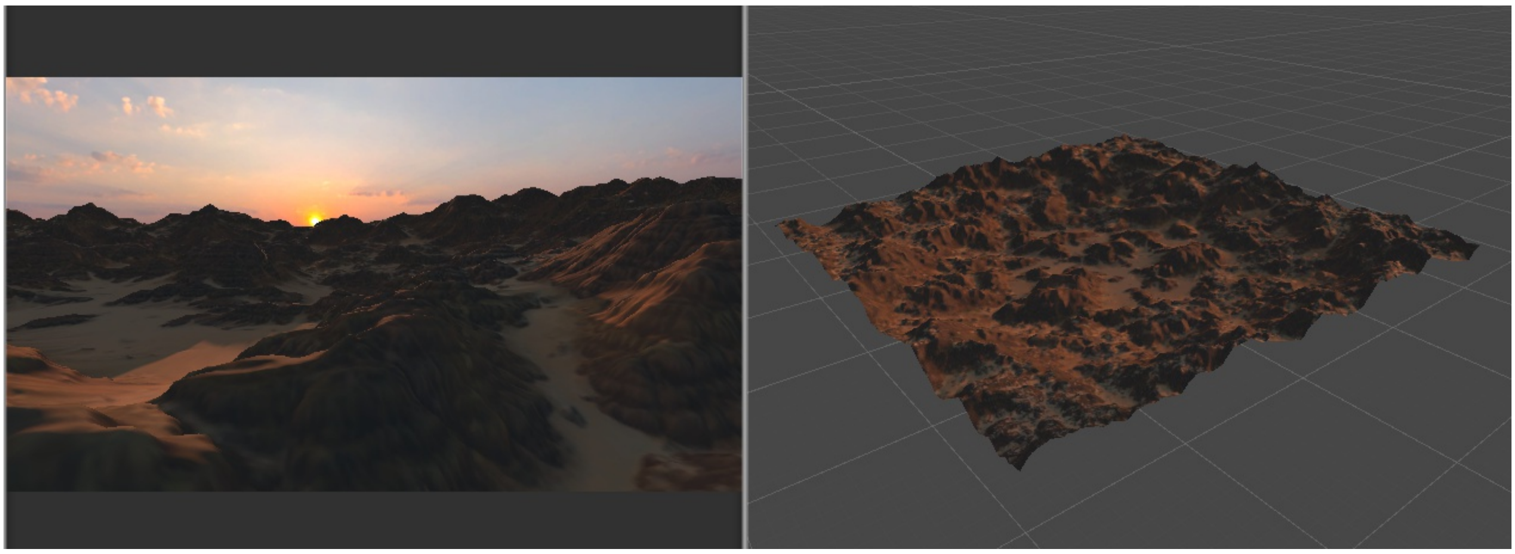
That's it! You should now be seeing the chosen preset applied in your game view (make sure you're looking through the same camera you used in the Camera slot).

You can now start customising the atmospheric settings on the Zone, or add more Zones for blending different effects in different parts of your scene. Take a look at the section DeepSky Haze Components to see more detail about all the available settings.

BASIC SETUP

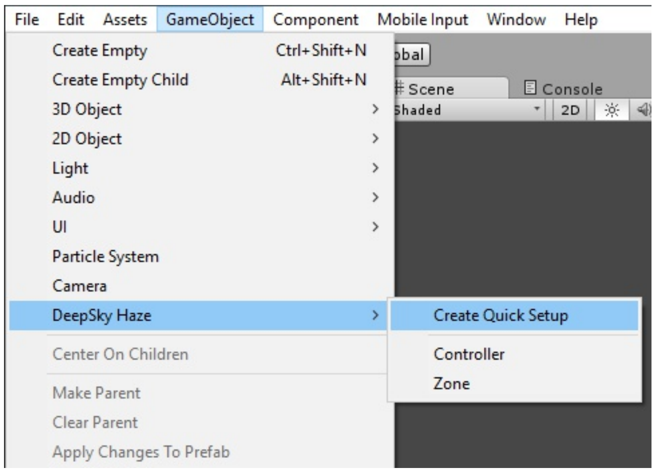
The Quick Setup window is the easiest way to add DeepSky Haze atmospherics to your scene, but it's useful to know how to create the setup manually for when you want to customise it further, or apply it to more complicated scenes. This section will take you through the process and show some simple adjustments to the atmospheric settings, in a basic scene with a landscape using Unity terrain (the same as the BasicSetup scene included in the Examples folder, with all the DeepSky Haze components removed).

This simple scene has a terrain, a directional light and the main camera. The sky is a HDR skybox and the directional light has been aligned with the sun in the image and given an orange tint. By default it looks like this:



Let's make this a more atmospheric sunset. To setup DeepSky Haze we need three things: one or more Zones that hold the atmospheric settings (called a Context), a Controller to manage the Zones and finally a View to actually do some rendering. An important idea in DeepSky Haze is the separation of Context and View – the Context defines what to render and the View defines how. This allows one collection of settings to be used by multiple Views.

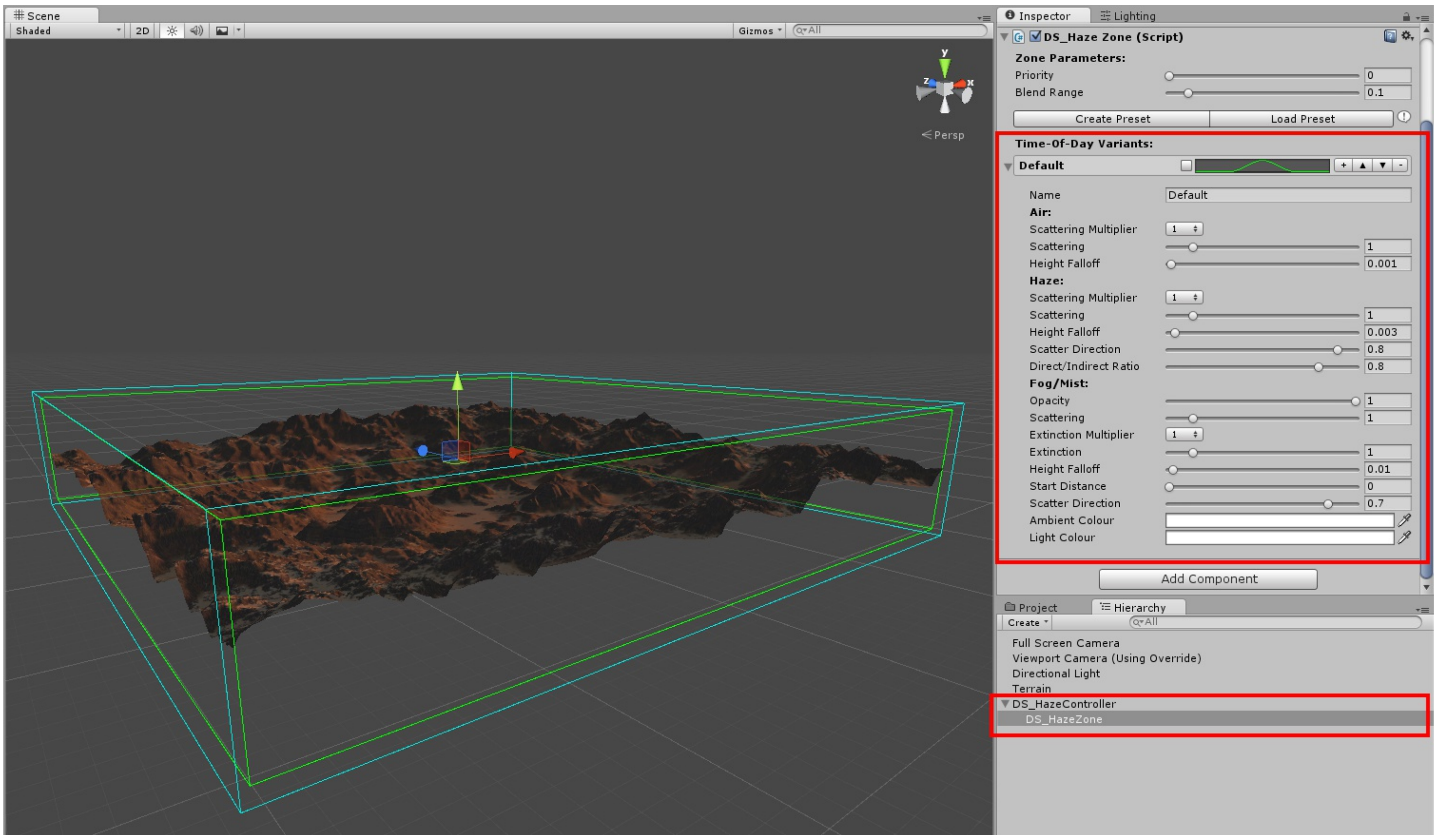
Add a Haze Controller using the **GameObject → DeepSky Haze → Controller** menu item. This is the component that manages the atmospheric zones.



Add a Haze Zone using the **GameObject → DeepSky Haze → Zone** menu item. Scale and move it's transform so the blue bounding box surrounds the terrain. By default the Zone object will be added as a child of the Controller, but if it isn't then simply drag it onto the Controller in the Hierarchy tab (only zones that are children of the controller will have an effect).

The Zone will have a Default time-of-day variant with some initial settings in the Inspector – we'll come back to these in a bit.

The scene should now look something like this:



DS_Haze Zone (Script)

Zone Parameters:

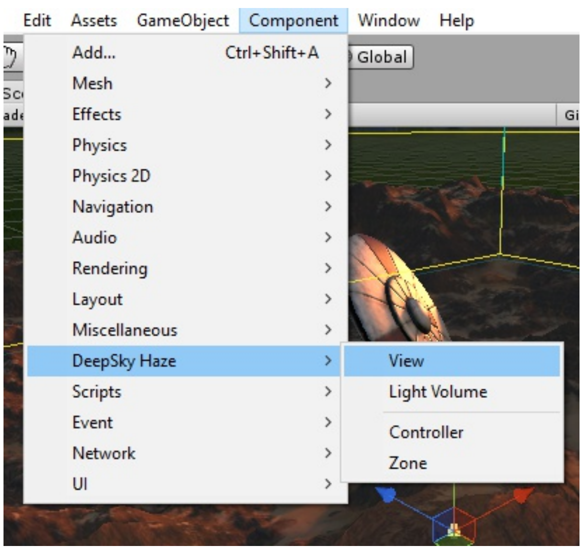
Priority	0
Blend Range	0.1

Time-Of-Day Variants:

Default

Name	Default
Air:	
Scattering Multiplier	1
Scattering	1
Height Falloff	0.001
Haze:	
Scattering Multiplier	1
Scattering	1
Height Falloff	0.003
Scatter Direction	0.8
Direct/Indirect Ratio	0.8
Fog/Mist:	
Opacity	1
Scattering	1
Extinction Multiplier	1
Extinction	1
Height Falloff	0.01
Start Distance	0
Scatter Direction	0.7
Ambient Colour	⬜
Light Colour	⬜

Full Screen Camera
Viewport Camera (Using Override)
Directional Light
Terrain
DS_HazeController
DS_HazeZone

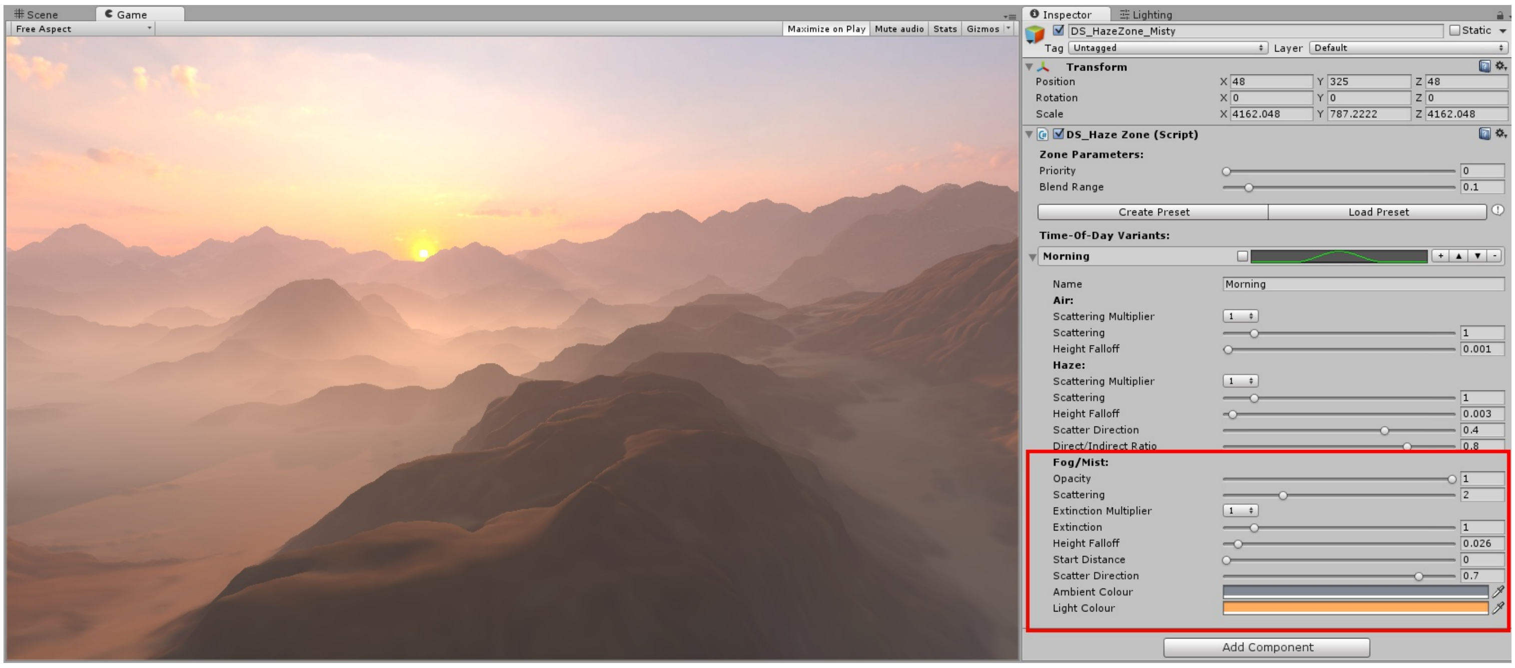


To actually see the effect, we need to add a component to the main camera. With the camera selected, add a View component using the **Component → DeepSky Haze → View** menu option.

Drag the directional light from the Hierarchy tab onto the Direct Light field of the View, so the light direction and shadows will come from the scene. The scene should now be filled with white fog in the Game View, so let's make it prettier.

Select the Zone again and expand the settings for the Default time-of-day variant using the little arrow to the left, if they're not visible. We want to tweak the fog values in the Fog/Mist section, particularly the Extinction, Scattering, Height Falloff and the Ambient Colour and Light Colour. See the Components section later for descriptions of all the settings available.

After a few tweaks you should end up with something like this:



DS_HazeZone_Misty · Static
Tag: Untagged · Layer: Default

Transform

	X	Y	Z
Position	48	325	48
Rotation	0	0	0
Scale	4162.048	787.2222	4162.048

DS_Haze Zone (Script)

Zone Parameters:

Priority	0
Blend Range	0.1

Time-Of-Day Variants:

Morning

Name	Morning
Air:	
Scattering Multiplier	1
Scattering	1
Height Falloff	0.001
Haze:	
Scattering Multiplier	1
Scattering	1
Height Falloff	0.003
Scatter Direction	0.4
Direct/Indirect Ratio	0.8
Fog/Mist:	
Opacity	1
Scattering	2
Extinction Multiplier	1
Extinction	1
Height Falloff	0.026
Start Distance	0
Scatter Direction	0.7
Ambient Colour	⬜
Light Colour	🟧

As you rotate the directional light, you should be able to see the light shafts cast by the distant mountains.

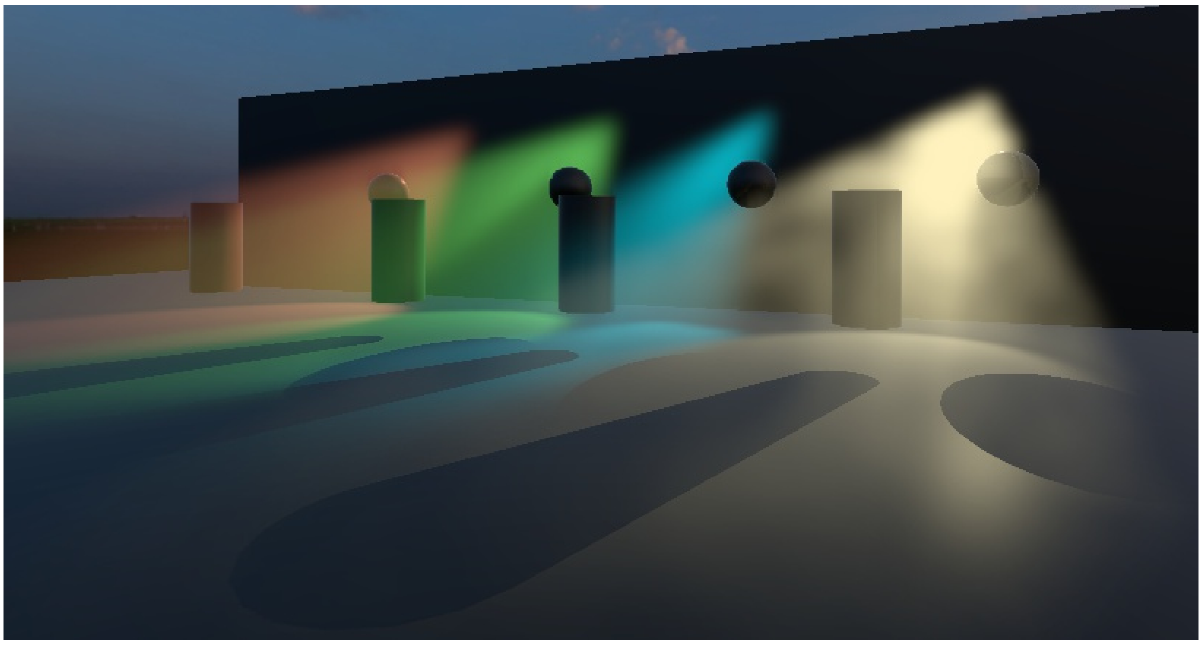
LOCAL LIGHT VOLUMES

OVERVIEW

As well as simulating large-scale atmospheric scattering from the sun, DeepSky Haze can also create scattering effects on local spot and point lights. These are handled separately from the atmospheric scattering and are implemented as an additional component, the **DS_HazeLightVolume**, added to light objects. This allows maximum control over quality and performance on a per-light basis.

Note that only **Spot** and **Point** lights are supported (as the Area light type is used for baking only). Adding a DS_HazeLightVolume component to an Area light will have no effect.

DeepSky Haze will use all the existing light settings in order to integrate cleanly into your scene - colour, intensity, shadows and cookies will all be used automatically.



The examples above show several spot lights with different settings:

- Orange - colour and intensity only.
- Green - light casts shadows.
- Blue - light casts shadows and has a rectangular cookie texture assigned.
- Yellow - light casts shadows, has a cookie texture and uses a 3D texture to control the volume density.

USING IN A SCENE

Local light volumes require a Core object, a Zone object and a View on the main camera in order to be visible just like the standard atmospheric scattering. See the section Getting Started to find out how to create a basic setup. Note that although light volumes do not use the atmospheric settings from the Zone system, they will still only be rendered when a camera is inside at least one Zone.
To make a light create volumetric scattering effects, simply add a DS_HazeLightVolume component to it using the **Components -> DeepSky Haze -> Light Volume** menu option. You can then adjust the scattering parameters for the intensity, the range and sampling parameters for optimisation and add a 3D density texture for extra detail and animation if desired. For a detailed description of the settings, see the section DeepSky Haze Components.

PERFORMANCE

While DeepSky Haze is fast, there are naturally limits to how much you can render in a single frame. The main factor on performance is the **number of samples** each light uses to render scattering effects. You should set this as low as possible - the minimum of 4 samples will often be plenty for small lights. For larger lights, or those casting detailed shadows or cookies, you will likely need to increase the number of samples. However you should always try the smaller settings first, due to the way DeepSky Haze interpolates samples you might be surprised by what you can get away with!

Also to help with optimising your game, light volumes will only be rendered when their parent light is **active** and when they are less than the distance from the camera determined by the per-light **End Fade** setting. You should aim to fade light volumes out as soon as possible - the smooth fade between the **Start Fade** and **End Fade** parameters helps make this less noticeable.

Note that DeepSky Haze make no assumptions about how you are managing and culling lights in your game. If you already have a system to enable/disable lights then you will already be culling light volumes as well (they are only active if the parent light and GameObject are active). You may need to adjust the Start Fade and End Fade distances to match any existing culling system to avoid the light volume 'popping' in and out.

EXAMPLE SCENES

There are two example scenes included demonstrating several features of DeepSky Haze and can be found in the **DeepSky Haze\Examples** folder.
In all scenes use the mouse to look around and press Escape to exit play-mode (in the editor) or quit the player.

BASIC SETUP

This scene contains the basic setup with a landscape as shown in the section Getting Started. It also contains a second Zone with thicker fog settings - drag the camera into this zone to see how the atmospheric effects are smoothly blended.

There is also a second camera (Viewport Camera) rendering another view of the same scene using a Context Override. This allows cameras to get their atmospheric settings from a saved Context, rather than using the Context in the scene. For more details, see the section Using Multiple Cameras And Overrides.

LIGHT VOLUMES

This scene shows 4 spotlights with light volume components as in the example image in the section Light Volumes.

Each spot light (from left to right) uses a new feature - experiment with toggling shadows, changing the light intensity/colour and the animation speed of the 3D density texture to see what kind of effects are possible.

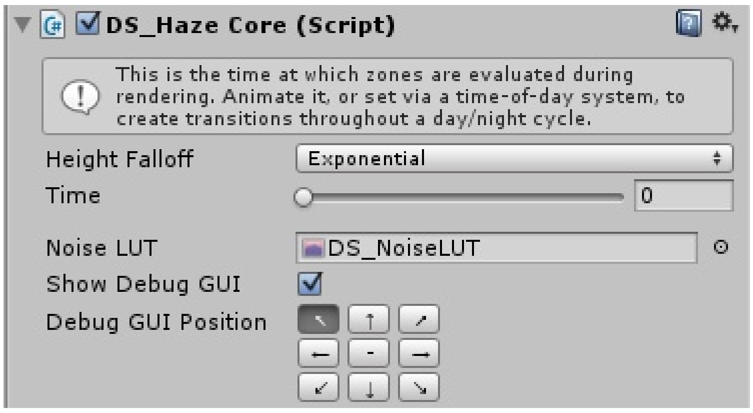
DEEPSKY HAZE COMPONENTS

- DS_HAZECORE
- DS_HAZEZONE
- DS_HAZEVIEW
- DS_HAZELIGHTVOLUME

This section details the components that make up DeepSky Haze and lists the properties available in the Inspector window. You can click on the ⓘ icons in a component to expand the help text for a section. Click again to collapse the text. With the exception of the DS_HazeView component, you will most likely not be creating the components directly. Instead, it is easier to create the complete GameObjects from the **GameObject → DeepSky Haze** menu which will already have the components attached. Of course if you want to combine these components with existing systems then they can be added to any GameObject as normal from the **Components → DeepSky Haze** menu.

DS_HAZECORE

Attached to a GameObject. Created via **Component → DeepSky Haze → Core** or **Component → DeepSky Haze → Core**



DS_Haze Core (Script)

This is the time at which zones are evaluated during rendering. Animate it, or set via a time-of-day system, to create transitions throughout a day/night cycle.

Height Falloff: Exponential
Time: 0
Noise LUT: DS_NoiseLUT
Show Debug GUI: ✓
Debug GUI Position

This is the main controller component for the DeepSky Haze system. There should only be one in the scene (any extras will be disabled during Awake). You can get a reference to it in scripts via the static DS_HazeCore.Instance property.

- **Time** – the time at which the zones are evaluated during rendering, in the range 0 to 1. It can be set via script using the public Time property, the passed value will be clamped between 0 and 1. Note that DeepSky Haze doesn't make any assumptions about what specific times map to 0 and 1. In the included example 0 and 1 are considered midnight, with 0.5 being midday but this is entirely dependent on the convention used by your game.

- **Height Falloff Type** - the type of height falloff to use globally. The default value of Exponential means the density of air, haze and fog will decrease exponentially with height, much like in the real world, starting from 0 on the world Y-axis. This is great for landscapes but for other types of scenes (eg. in space) you can set the height falloff type to None. This means the density of the atmospherics is only dependant on viewing distance, regardless of the camera's height. Note when the falloff type is None the Height Falloff parameters in Zones will be greyed out, as they are ignored when rendering.

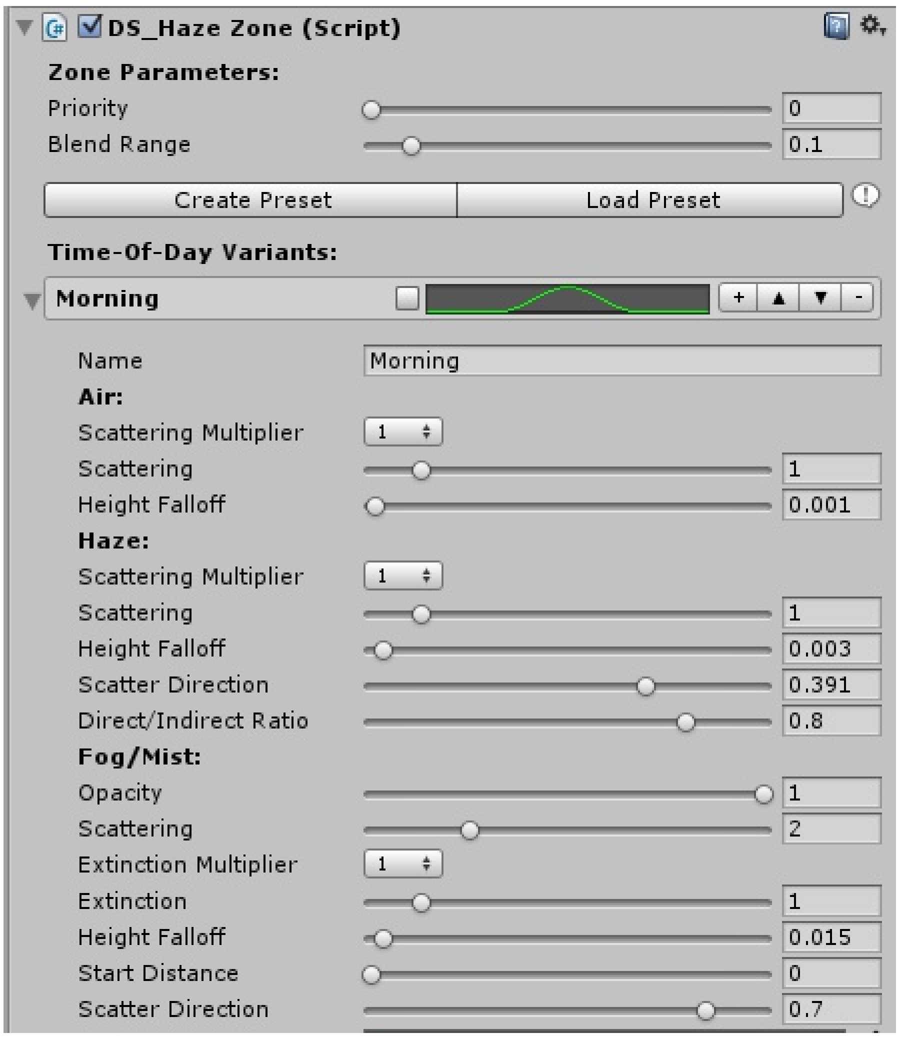
- **Noise LUT** - this is a small 3D look-up texture used internally to randomize the volume samples. The default is a 16x16x16 Blue Noise texture which gives great results, but the option is there if you want to experiment. Note however that the shaders assume the texture is 16x16x16.

- **Show Debug GUI** - display a small debug overlay in the Editor and Development builds, from which you can choose a DS_HazeView component in the scene to view information about its current settings. Use the arrow buttons below the toggle to choose the position on-screen for the overlay.

DS_HAZEZONE

Attached to a GameObject. Created via **GameObject → DeepSky Haze → Zone** or **Component → DeepSky Haze → Zone**.

The Zone component defines an area in which a collection of settings take effect. The position and size of the Zone are taken from the Transform's position and scale properties. Zones can be rotated to better fit an area, but properties relating to vertical height are always based on the world Y axis, starting from global 0.



DS_Haze Zone (Script)

Zone Parameters:

Priority	0
Blend Range	0.1

Create Preset | Load Preset

Time-Of-Day Variants:

Morning

Name	Morning
Air:	
Scattering Multiplier	1
Scattering	1
Height Falloff	0.001
Haze:	
Scattering Multiplier	1
Scattering	1
Height Falloff	0.003
Scatter Direction	0.391
Direct/Indirect Ratio	0.8
Fog/Mist:	
Opacity	1
Scattering	2
Extinction Multiplier	1
Extinction	1
Height Falloff	0.015
Start Distance	0
Scatter Direction	0.7

Zone Parameters

- **Priority** – determines which order zones are blended when they overlap. Higher numbers take precedent over lower. If two overlapping zones have the same priority then the smallest zone is considered higher priority.
- **Blend Range** – distance over which this zone will blend with any overlapping zones. The inner yellow box shows where this zone will have total effect; between the blue and yellow boxes the settings will be linearly interpolated.

Buttons

- **Create Preset** – save these settings as an asset in the Contexts folder.
- **Load Preset** – get the settings from a previously saved asset.

Time-Of-Day Variants Stack

The stack shows all the time-of-day variants available in this zone. They are blended from top to bottom when calculating the Context to render with. Click on the drop-down arrow next to an item's header to expand or collapse its settings.

The header shows the following properties (left-to-right):

- **Name** – shows the name assigned to this variant.
- **Solo** – when checked this variant will be used directly and no blending will take place. Use this to preview how a variant looks while working on it (Note: this setting is only applied in the Editor).
- **Blend Weight** – this curve defines how much influence these settings have at any particular time, 0 being no effect to 1 being fully on. The blend weight is ignored for the top item in the stack, which is always fully on. Click on the curve to bring up the curve editor.
- **Duplicate (+)** – creates a new variant with the same settings as this one. It will be added to the bottom of the stack with '_Copy' appended to the name.
- **Move Up (▲)** – move this item up the stack.
- **Move Down (▼)** – move this item down the stack.
- **Remove (-)** – remove this item completely from the stack.

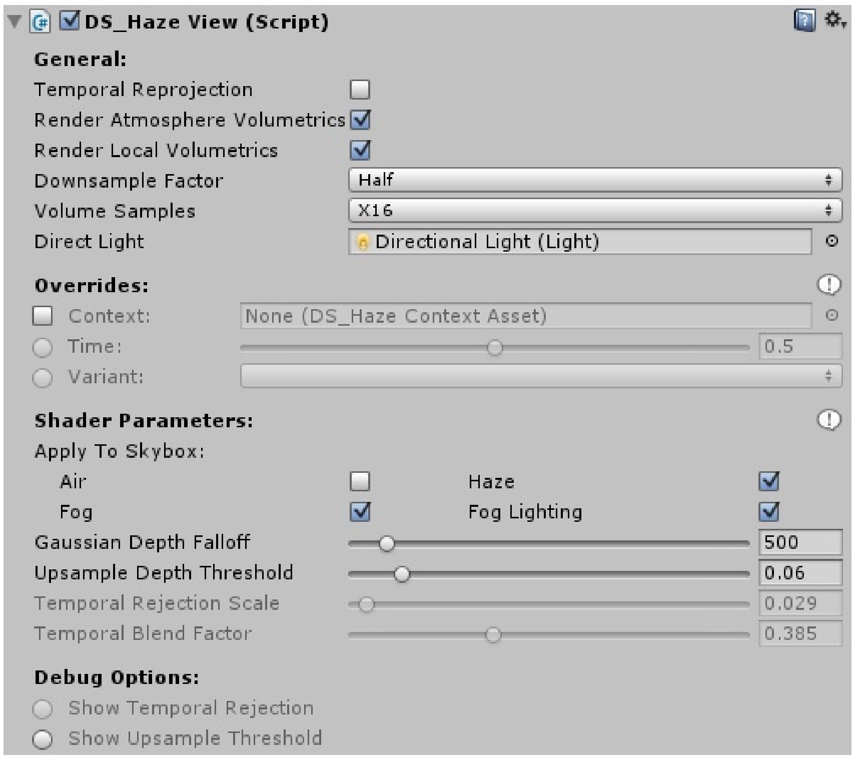
When expanded, a time-of-day variant will display the following settings for its Context Item:

- **Name** – change the name of the item here for easy identification when collapsed.
- **Air** – the following properties relate to scattering caused by the air molecules (known as Rayleigh scattering).
    - **Scattering Multiplier** - scale the scattering value. Allows the range of the scattering amount to be more easily tuned to the size of the view distance (eg. use high multipliers for more visible scattering over shorter view distances).
    - **Scattering** – how much scattering occurs.
    - **Height Falloff** – how quickly the air density decreases with height.
- **Haze** – the following properties relate to the scattering caused by dust/pollution in the air.
    - **Scattering Multiplier** - scale the scattering value (see Air scattering multiplier above).
    - **Scattering** – how much scattering occurs.
    - **Height Falloff** – how quickly the dust density decreases with height. The default of 0.003 gives a typical real-world falloff.
    - **Scatter Direction** – how much the light is scattered forwards (positive values) or backwards (negative values) along the original light direction. Dust is typically strongly forward scattering, creating the bright glow around the sun.
    - **Direct/Indirect Ratio** – control the blend between direct and indirect lighting. The direct light uses the shadow map so to prevent it becoming too unrealistically dark in the shadowed areas, some secondary scattering is added in.
- **Fog** – the following properties relate to the height-based fog.
    - **Opacity** - maximum amount the fog can obscure the scene, regardless of the amount of actual extinction. Anything other than 1 is not physically-based but can help tune the fog appearance in certain types of scenes.
    - **Scattering** – how much lighting is applied to the fog.
    - **Extinction Multiplier** – scale the fog extinction value. Similar to the scattering multipliers above, this allows the extinction to be more easily tuned to the size of the view distance (eg. use a low multiplier for more control over larger view distances).
    - **Extinction** – how quickly the background scene is obscured by the fog.
    - **Height Falloff** – how quickly the fog density decreases with height.
    - **Start Distance** – how far from the camera the fog starts. This value is in the range 0 to 1 between the camera near and far clip planes.
    - **Scatter Direction** – how much the light is scattered forwards (positive values) or backwards (negative values) along the original light direction. Higher values will create a bright glow around the sun.
    - **Ambient Colour** – the base colour of the fog.
    - **Light Colour** – the colour used for lighting in the fog.

Note that both the fog Ambient and Light colours are scaled automatically by a View when rendering to match the intensity of the referenced directional light. This ensures the fog lighting is always relative to the intensity of the lighting in the other components of the atmospheric effect, particularly when rendering in high-dynamic range.

DS_HAZEVIEW

Attached to a Camera. Created via **Component → DeepSky Haze → View.**



DS_Haze View (Script)

General:
Temporal Reprojection ☐
Render Atmosphere Volumetrics ☑
Render Local Volumetrics ☑
Downsample Factor: Half
Volume Samples: X16
Direct Light: Directional Light (Light)

Overrides:
☐ Context: None (DS_Haze Context Asset)
○ Time: 0.5
○ Variant:

Shader Parameters:
Apply To Skybox:
Air ☐ Haze ☑
Fog ☑ Fog Lighting ☑
Gaussian Depth Falloff: 500
Upsample Depth Threshold: 0.06
Temporal Rejection Scale: 0.029
Temporal Blend Factor: 0.385

Debug Options:
○ Show Temporal Rejection
○ Show Upsample Threshold

The View component is the actual renderer for the atmospheric effects. During rendering it will query the Core controller in the scene to get the settings with which to render (or use the settings from the override context, if specified).

General

- **Temporal Reprojection** - if a later image effect also uses temporal reprojection (eg. Anti-Aliasing) you can disable it here to gain some performance and reduce memory usage.
- **Render Atmosphere Volumetrics** - should this view render volumetric haze from the directional light?
- **Render Local Volumetrics** - should this view also render local light volumes?
- **Downsample Factor** - what resolution the volumetric effects render at, relative to the screen size. The default is half-resolution but quarter resolution will be significantly faster (at the expense of quality).
- **Volume Samples** - the number of volumetric samples used per-fragment. When rendering at half-resolution this should be left at the default (x16) but if using quarter resolution you can increase the samples to x24 or x32 to reduce artefacts.
- **Direct Light** – the directional light object to use for lighting direction, colour and intensity.

Overrides

- **Context** – a saved Context asset to render with instead of getting settings from the Controller in the scene.
- **Time** – the time to render the Context at, regardless of the time defined by the Controller.
- **Variant** – the specific time-of-day variant to render, without blending all the variants at the defined time.

Shader Parameters

- **Apply To Skybox** - use the check boxes to control which components of the atmospherics will be rendered on top of the skybox as well as the normal scene geometry. By default, only haze and fog are applied as atmospheric scattering is assumed to be already present in the skybox (such as when using Unity's default procedural sky shader or a cubemap using a HDR image of a real sky).
    - **Air** - apply the air (Rayleigh) scattering. Usually this should be left off except for special cases, such as a custom skybox texture where atmospheric scattering has not been painted in.
    - **Haze** - apply the haze (Mie) scattering. Usually this should be left on as the haze is typically strongly view dependent and therefore can change dynamically as the directional light rotates. This is also required to correctly apply the volumetric scattering to the sky.
    - **Fog** - apply the fog extinction to the sky. Usually on, unless the skybox texture already has fog painted in.
    - **Fog Lighting** - apply the lighting component of the fog to the sky. Similar to haze, the fog lighting is strongly view dependent and can change dynamically with the directional light. Even if your skybox texture has fog painted in already, it's likely you will still want to apply the lighting. This will help your pre-painted fog to correctly integrate with the foreground elements of the scene.

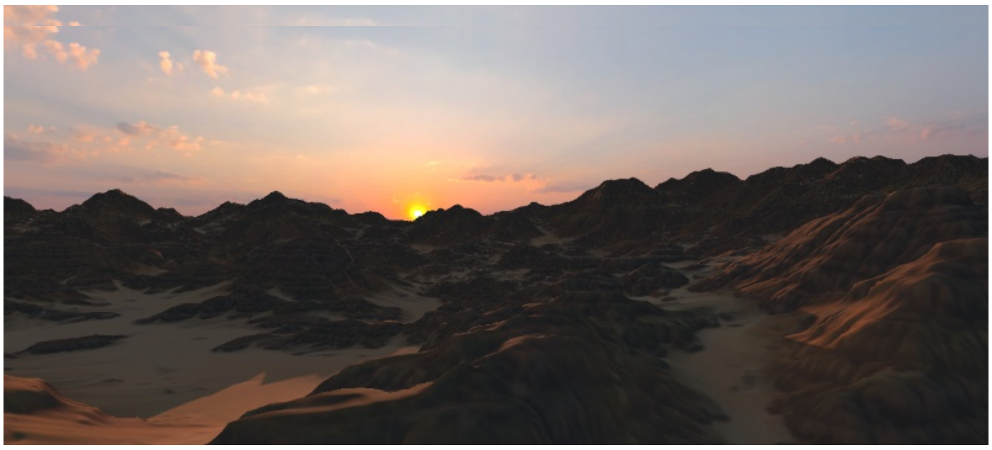
Debug Options

- **Show Temporal Rejection** – display rejected pixels in red. These pixels do not re-use data from the previous frame.
- **Show Upsample Threshold** – display pixels that are considered edges in the depth buffer in green. These are the pixels that will use point sampling on DX11/OpenGL Core to help preserve geometry edges.

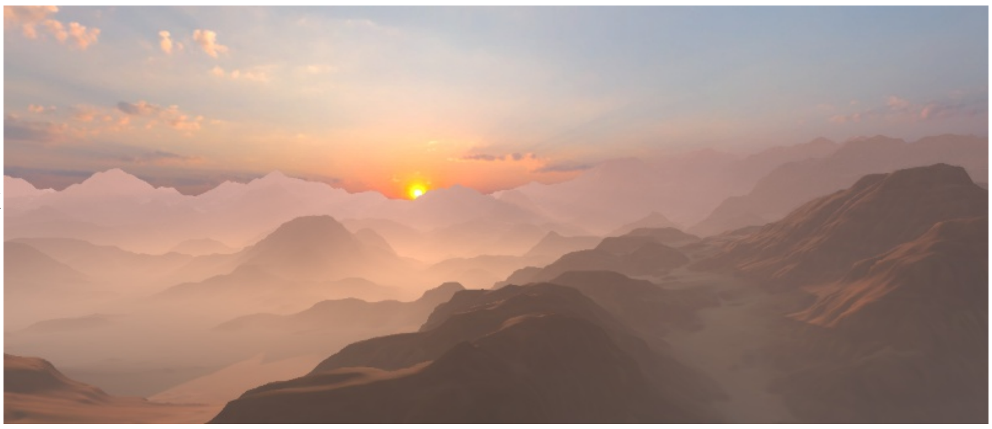
Note: when either debug option is enabled, the frame will be darkened to make the debug colours easier to see.

APPLY TO SKYBOX EXAMPLES

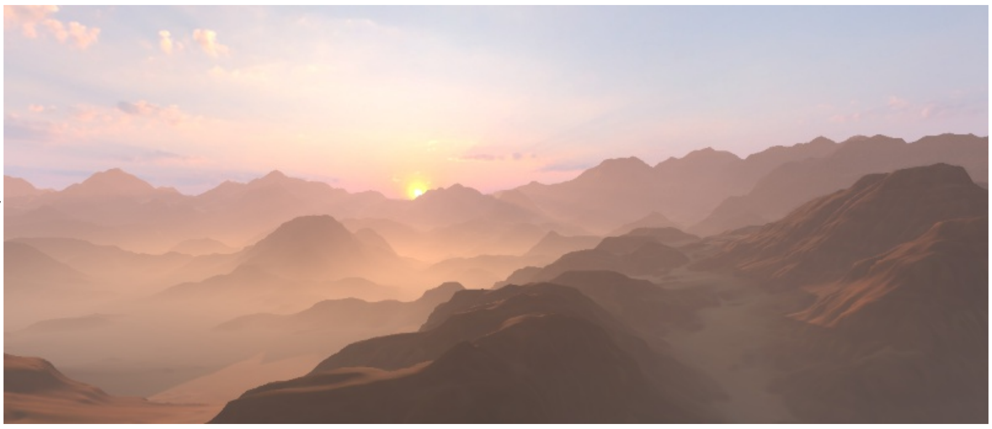
Base scene - DeepSky Haze disabled



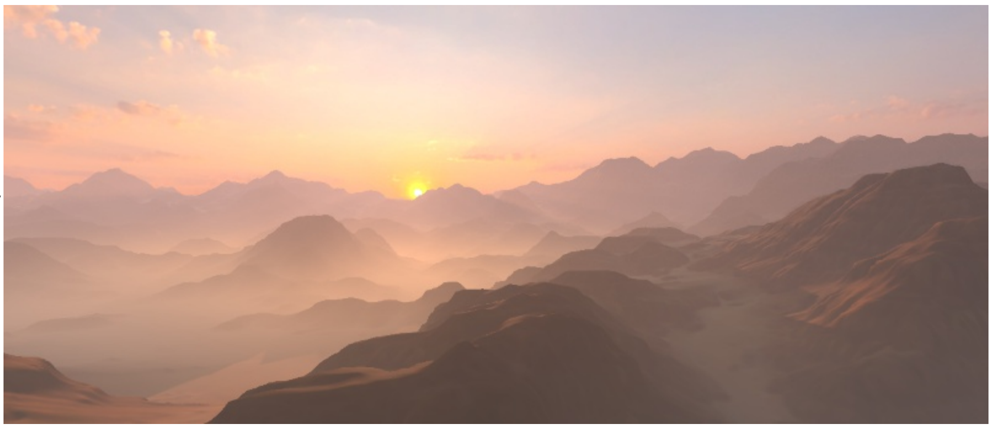
Nothing applied to the skybox



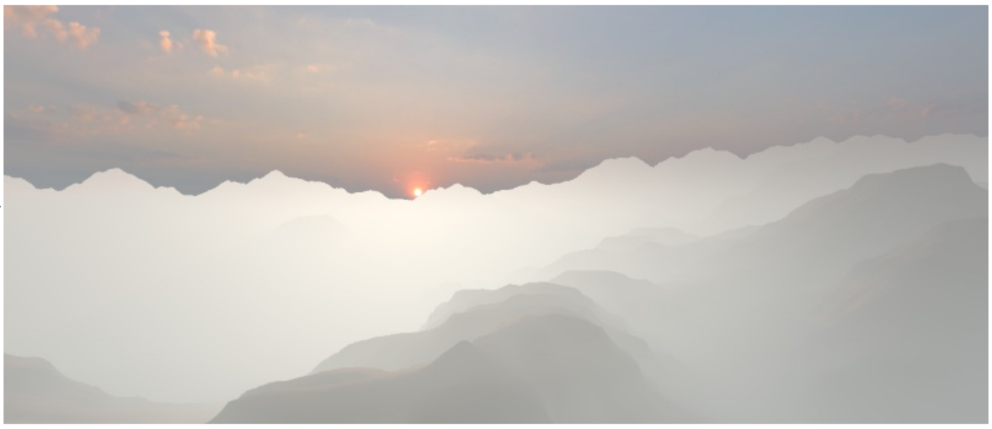
Air scattering only



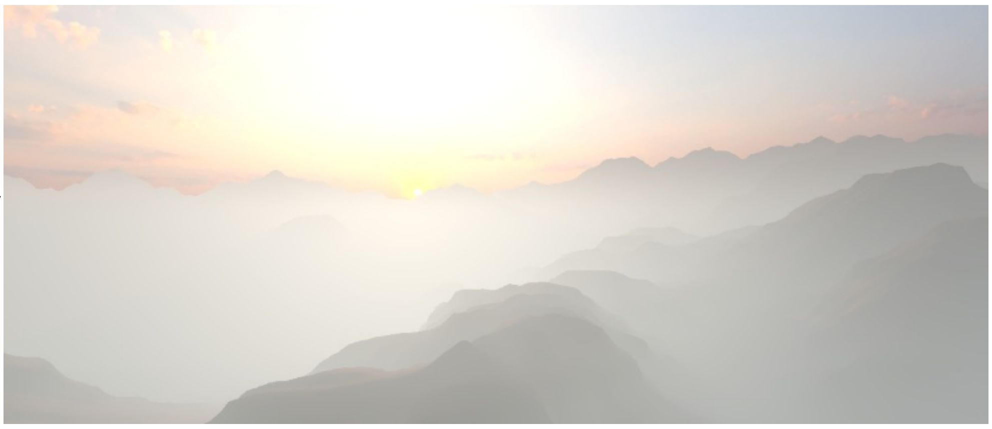
Haze scattering only



Increased fog - fog only



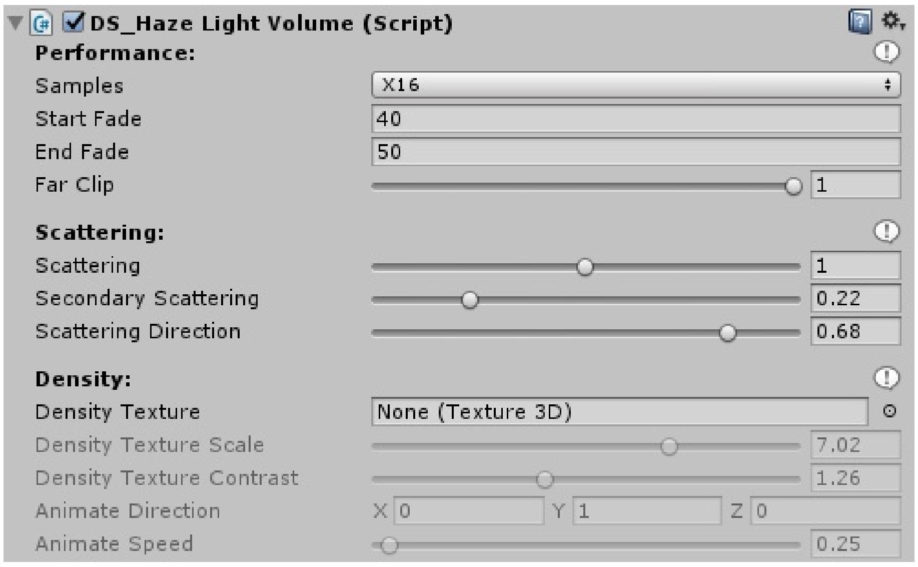
Fog lighting only



DS_HAZELIGHTVOLUME

Attached to a spot or point light. **Created via Component -> DeepSky Haze -> Light Volume**.

The Light Volume component enables local volumetric effects for spot and point lights. These are independent of the atmospheric scattering, and are configured on a per-light basis. Light Volumes will use the same falloff, shadow maps and cookie textures as the light component. During rendering, the Core controller is queried to find any active light volumes within range.



DS_Haze Light Volume (Script)

Performance:

Samples	X16
Start Fade	40
End Fade	50
Far Clip	1

Scattering:

Scattering	1
Secondary Scattering	0.22
Scattering Direction	0.68

Density:

Density Texture	None (Texture 3D)
Density Texture Scale	7.02
Density Texture Contrast	1.26
Animate Direction	X 0 Y 1 Z 0
Animate Speed	0.25

Performance

• **Samples** - the number of samples to take while rendering the volumetrics.
Higher numbers give better quality, at the cost of performance. You should set this as low as possible and only raise it when you see artifacts appear, you might be surprised how good just 4 samples actually looks!
• **Start Fade** - the distance at which the effect starts to fade out.
• **End Fade** - the distance at which the effect is completely invisible, beyond this distance the effect is culled and will not be rendered.
• **Far Clip (spotlights only)** - The distance from the light at which the light volume cone ends, as a multiplier of the light's Range setting. So a light with a Range of 10 and a Far Clip of 0.5 will stop drawing volumetrics at 5 units from the light source. As the performance cost depends on the screen-space size of the light, this can help reduce the number of pixels required to render the effect.

Scattering

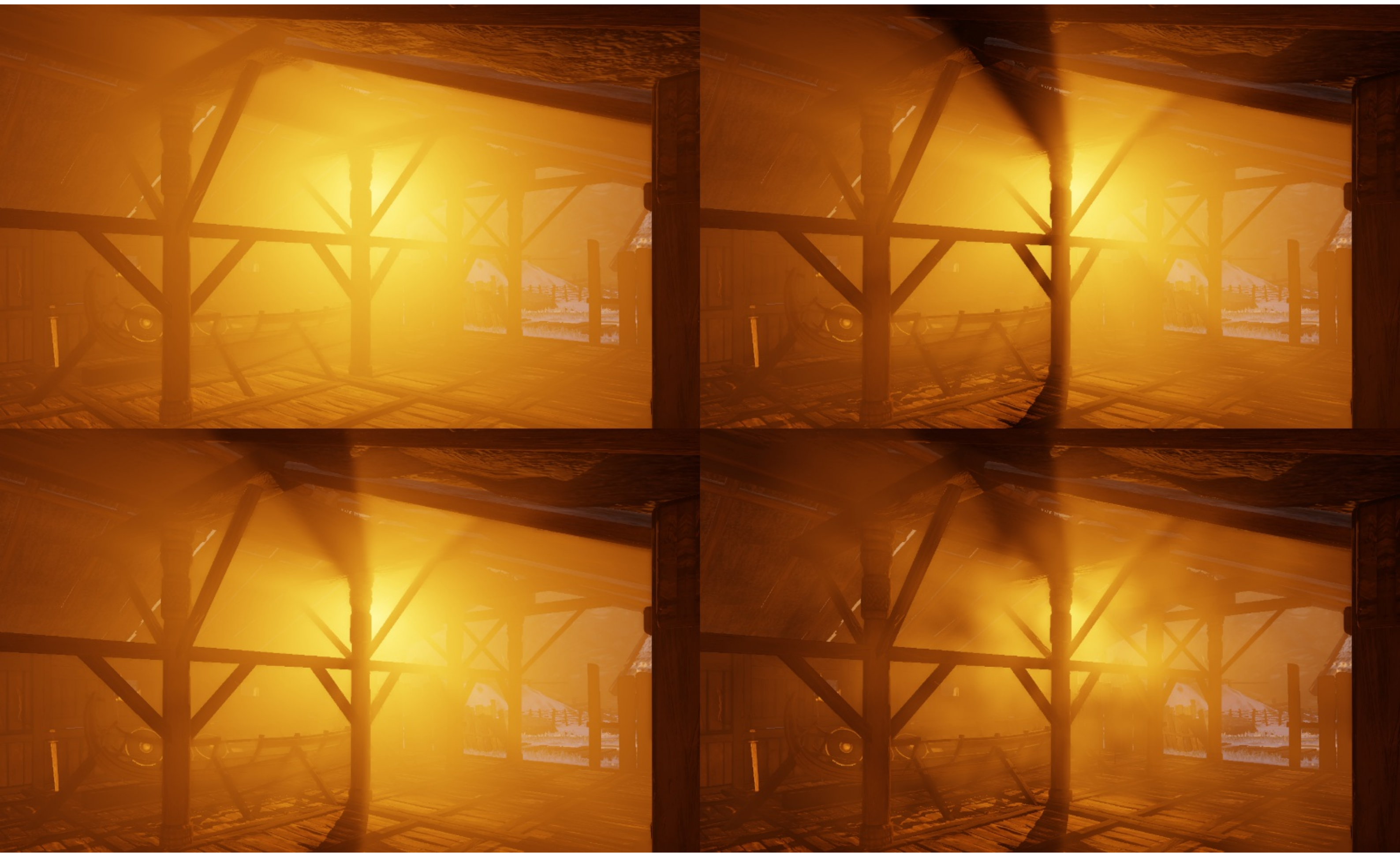
- **Scattering** - how much of the light is scattered by dust/water vapour etc. This acts as a multiplier for the light's Intensity value. You can set this value above 1 to exaggerate the effect, but be aware that is not physically accurate as you are then adding extra light!
- **Secondary Scattering** - how much light is scattered in shadowed areas. As light bounces around, it will sometimes scatter several times resulting in shadows in the volume being 'filled in'. Use this value to approximate secondary scattering and limit how dark shadowed parts of the light volume can become.
- **Scattering Direction** - how much the light is scattered forwards (positive values) or backwards (negative values) along the original light direction. This is most visible on spotlights.

Density

- **Density Texture** - a 3D texture that acts as a multiplier for the dust/water vapour density that the light is passing through. By assigning a texture you can simulate a smoky or foggy atmosphere.
- **Density Texture Scale** - how big the density texture appears.
- **Density Texture Contrast** - how much influence the density texture has, high contrast values will create empty patches in the volume.
- **Animate Direction** - the direction of movement (world-space vector) in which the density texture scrolls. By animating the texture you can simulate rising smoke, or wind-blown mist.
- **Animate Speed** - how fast the density texture moves.

EXAMPLES

- (top-left) - light does not cast shadows, scattering only.
- (top-right) - light casts shadows.
- (lower-left) - light casts shadows, plus Secondary Scattering at 0.2, note how shadows appear lighter.
- (lower-right) - 3D density texture added to break up the volume and create a more smoky atmosphere.



USING MULTIPLE CAMERAS AND OVERRIDES

Multiple cameras in a scene can all get their settings from the single Controller, allowing settings to be shared. Simply add a View component to any camera requiring the atmospheric effects.

If a camera needs to render different settings to those in the scene, you can override the Context on a per-camera basis. This is particularly useful for example during cutscenes, where you may wish to show a different time of day or create a specific atmosphere without intefering with the settings in the scene.

To use an override, setup the atmospherics as required and save the Context using the Save Preset button on the Zone component. This will save the Context in the DeepSky Haze/Context folder (you can move it where ever you like in the project). Now in the Overrides section on the camera's View component, check the box next to Context. You can now drag the saved Context onto the slot. That's it, the camera will now render using the saved settings rather than those in the scene.

When using an override Context, you can also choose to render a specific time-of-day variant, rather than the blended result using the time value from the controller. Simply check the option next to Variant and choose the specific one you want from the dropdown list.

Lastly, you can set the specific time to render at by checking the option next to Time and adjusting the slider. This also works without a Context override, so you can use the same settings as the scene, just rendered at a specific time if required. Note that you can either set the Variant or the Time, not both.

TRANSPARENT SHADERS

DeepSky Haze includes several shaders that allow transparent objects to integrate with the atmospheric effects. As with all effects that require depth information, it is difficult to combine transparent objects with the scene without artifacts - but in most cases the included shaders will provide seamless results.

By default the transparent shaders calculate the atmospherics in the vertex shader. This gives great performance but requires the model to have enough vertices to provide a smooth blend. For very large objects with vertices spaced far apart (such as cloud planes), you may notice blending artifacts so all shaders have the option to calculate the atmospherics per-fragment instead. For most objects, the difference between vertex and fragment modes will be unnoticeable, so for best performance only enable Use Per Fragment when necessary.

They are broadly divided into two categories - the **Standard** shaders and the **Skybox** shaders.

STANDARD SHADERS

The standard shaders use Unity's built-in Standard and Standard Specular lighting models and have variants to apply all the atmospheric effects, or just the lighting components (haze scattering and fog lighting). Just like the Standard shaders, you can drop in a Metallic/Smoothness (or Specular/Smoothness) map and Normal map to enable those features. The shader will not have the extra expense of normal mapping if no normal map is present.

- **Standard Transparent - Full** - Use the Standard lighting model and calculate all atmospheric components.
- **Standard Transparent - Lighting Only** - Use the Standard lighting model, only calculate Haze and Fog lighting.
- **Standard (Specular) Transparent - Full** - Use the Standard (Specular) lighting model and calculate all Air, Haze and Fog effects.
- **Standard (Specular) Transparent - Lighting Only** - Use the Standard (Specular) lighting model, only calculate Haze and Fog lighting.

SKYBOX SHADERS

The skybox shaders are designed for use on background elements such as cloud planes or painted backdrops. They ignore all lighting in the scene and instead provide an intensity slider to allow balancing the brightness with the rest of your environment. They also ignore the actual depth of the fragment being rendered, instead calculating the atmospherics as if the fragment was on the far-clip plane. This helps unify your skybox elements by making sure the atmospherics are applied consistently across all of them. Similar to the Standard shaders, there are two versions:

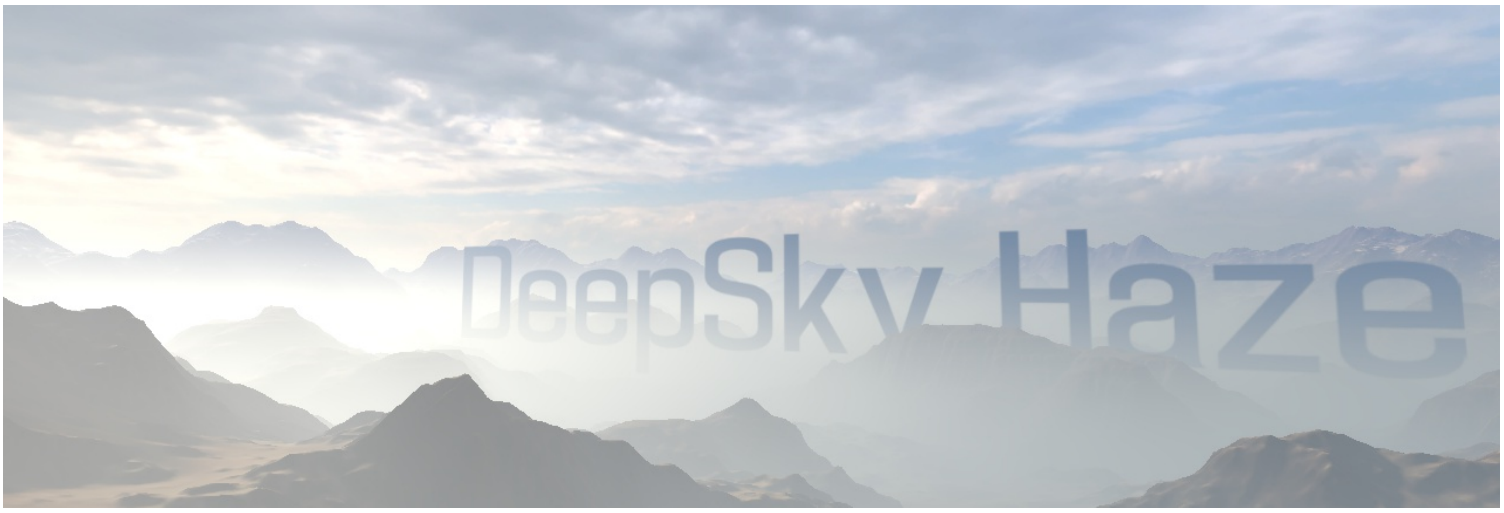
- **Skybox Transparent - Full** - Unlit, full depth and calculate all atmospheric components.
- **Skybox Transparent - Lighting Only** - Unlit, full depth, only calculate Haze and Fog lighting.

It's likely you will already have come atmospheric effects in your skybox, if using a HDR probe for example it will already contain scattering from the air and possible some fog or mist. You can use the Lighting Only variant to apply the dynamic lighting effects (Haze and Fog lighting) from your directional light without 'doubling up' the other effects already present.

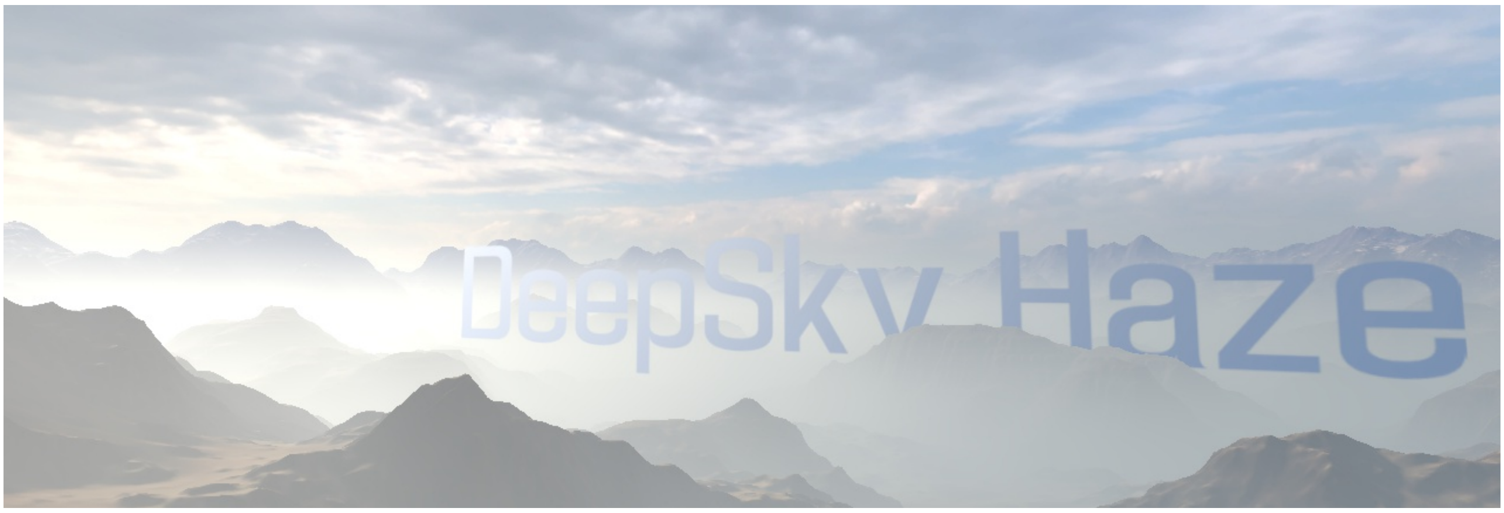
As always, creating a beautiful skybox takes time and careful adjustments but these shaders should help you integrate the various elements.

TRANSPARENT SHADER EXAMPLES

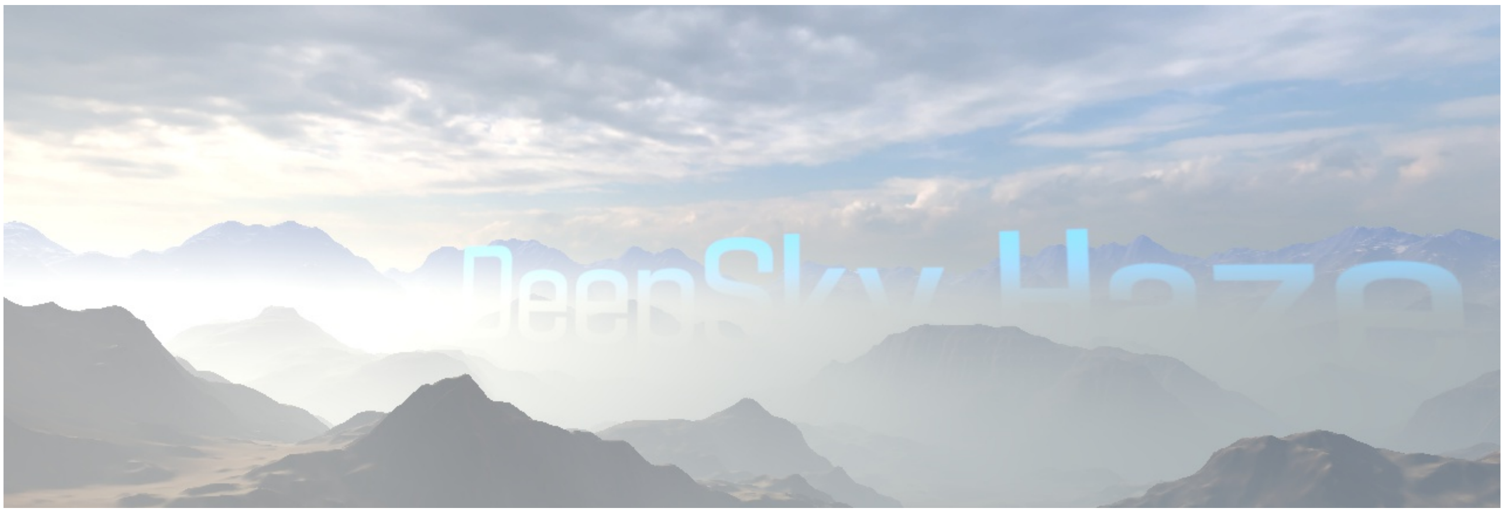
Standard Transparent - Full - all atmospheric components are applied.



Standard Transparent - Lighting Only - only haze and fog lighting are applied. Note the lighter appearance from the lack of fog extinction.



Skybox Transparent - Full - all components are applied at full depth (the actual fragment's depth is ignored). Note the consistent look of the fog despite the angle of the text. The air scattering turns the blue text even brighter.

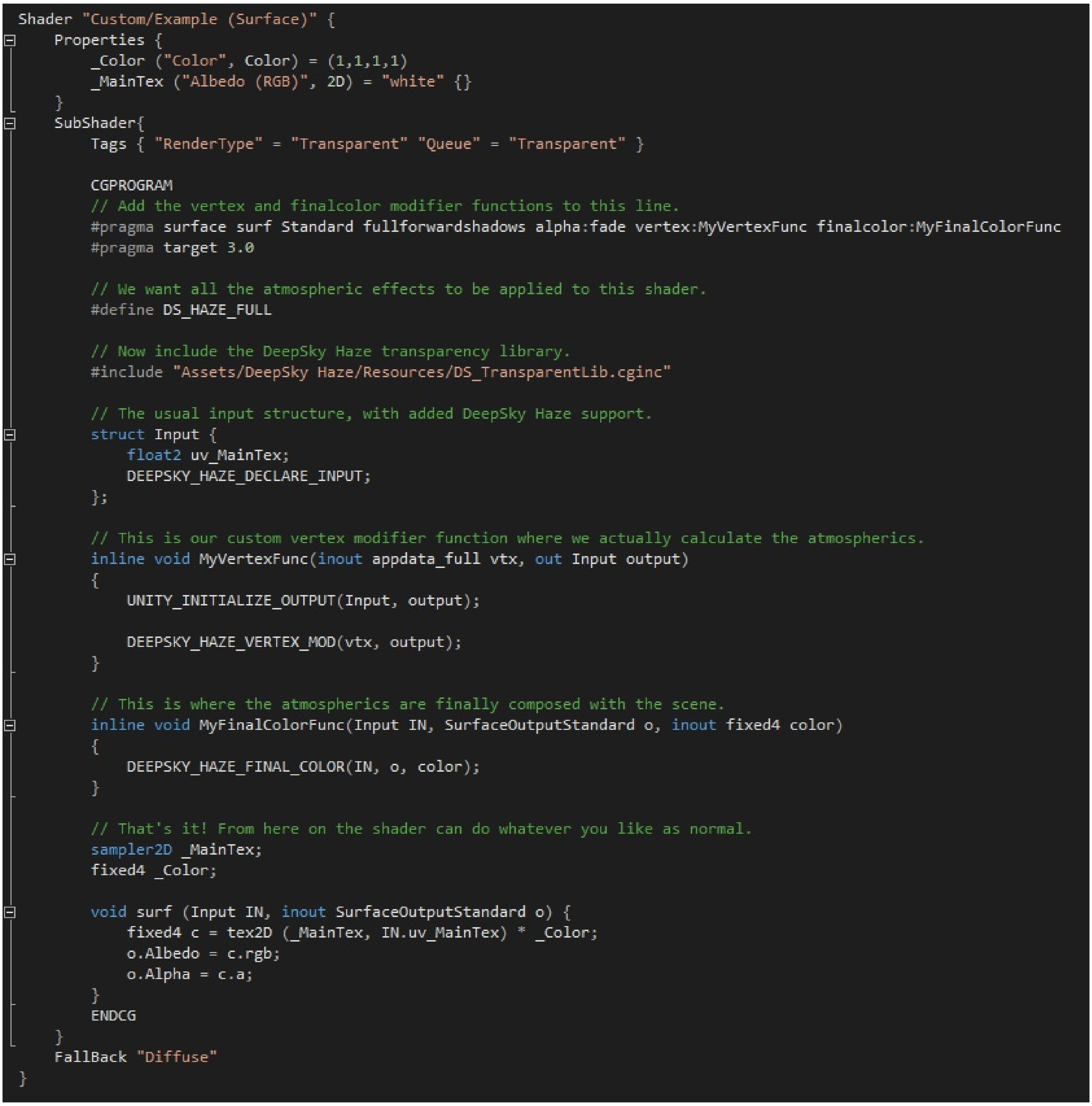


APPLYING ATMOSPHERICS TO CUSTOM TRANSPARENT SHADERS

Adding support for DeepSky Haze to your own shaders is easy; whether writing surface or vertex/fragment shaders, all the atmospheric effects are wrapped up in just a few functions.

Surface shaders make use of the vertex and finalcolor modifiers to apply atmospheric effects without interfering with the rest of your shader. If you are already using vertex or finalcolor modifiers for your own effects, adding support for DeepSky Haze requires just a couple of extra lines.

Here is a simple, but complete, transparent surface shader; it uses Unity's Standard physically-based lighting model and supports a texture and a colour input:



```
Shader "Custom/Example (Surface)" {
    Properties {
        _Color ("Color", Color) = (1,1,1,1)
        _MainTex ("Albedo (RGB)", 2D) = "white" {}
    }
    SubShader{
        Tags { "RenderType" = "Transparent" "Queue" = "Transparent" }

        CGPROGRAM
        // Add the vertex and finalcolor modifier functions to this line.
        #pragma surface surf Standard fullforwardshadows alpha:fade vertex:MyVertexFunc finalcolor:MyFinalColorFunc
        #pragma target 3.0

        // We want all the atmospheric effects to be applied to this shader.
        #define DS_HAZE_FULL

        // Now include the DeepSky Haze transparency library.
        #include "Assets/DeepSky Haze/Resources/DS_TransparentLib.cginc"

        // The usual input structure, with added DeepSky Haze support.
        struct Input {
            float2 uv_MainTex;
            DEEPSKY_HAZE_DECLARE_INPUT;
        };

        // This is our custom vertex modifier function where we actually calculate the atmospherics.
        inline void MyVertexFunc(inout appdata_full vtx, out Input output)
        {
            UNITY_INITIALIZE_OUTPUT(Input, output);

            DEEPSKY_HAZE_VERTEX_MOD(vtx, output);
        }

        // This is where the atmospherics are finally composed with the scene.
        inline void MyFinalColorFunc(Input IN, SurfaceOutputStandard o, inout fixed4 color)
        {
            DEEPSKY_HAZE_FINAL_COLOR(IN, o, color);
        }

        // That's it! From here on the shader can do whatever you like as normal.
        sampler2D _MainTex;
        fixed4 _Color;

        void surf (Input IN, inout SurfaceOutputStandard o) {
            fixed4 c = tex2D (_MainTex, IN.uv_MainTex) * _Color;
            o.Albedo = c.rgb;
            o.Alpha = c.a;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```

Regardless of what else your shader is doing, setting up DeepSky Haze always follows the same format as above:

- use #defines to configure DeepSky Haze
- include DS_TransparentLib.cginc (note this must be included after the #defines!)
- add **DEEPSKY_HAZE_DECLARE_INPUT** to your Input structure
- add **DEEPSKY_HAZE_VERTEX_MOD()** to your vertex function
- add **DEEPSKY_HAZE_FINAL_COLOR()** to your finalcolor function

You can choose how the atmospheric effects are applied using several #defines. The available options are:

- **#define DS_HAZE_FULL** - calculate and apply everything!
- **#define DS_HAZE_LIGHTING_ONLY** - only calculate and apply the dynamic lighting components (haze and fog lighting). Useful for background elements which already have fog and atmospheric scattering in their textures.
- **#define DS_HAZE_APPLY_PER_FRAGMENT** - calculate the atmospherics in the fragment shader. By default, DeepSky Haze will calculate the effects per-vertex for better performance. In some cases this can cause artifacts (if the vertices are very far apart for example) - use this option to trade performance for quality. Note that if you enable per-fragment calculation you should not add DEEPSKY_HAZE_VERTEX_MOD() to your vertex modifier function!
- **#define DS_HAZE_USE_FULL_DEPTH** - calculate the atmospherics as if this fragment were situated on the camera's far-plane (ie. ignore the actual depth of the fragment). This is useful for skybox elements which should all receive the same atmospheric effects for consistency, regardless of distance.

THE INPUT STRUCTURE

DeepSky Haze requires several additional fields in the Input structure and these are defined by the DEEPSKY_HAZE_DECLARE_INPUT macro. Two of these are standard Unity inputs so if you are already using either you will need to remove their declaration (you can still use them as normal, you just don't need to declare them):

- **float4 screenPos** (only when DS_HAZE_USE_FULL_DEPTH is defined)
- **float3 worldPos** (only when DS_HAZE_APPLY_PER_FRAGMENT is defined)

So if, for example, your input structure looks like this:

```
struct Input {
	float2 uv_MainTex;
	float3 worldPos;
}
```

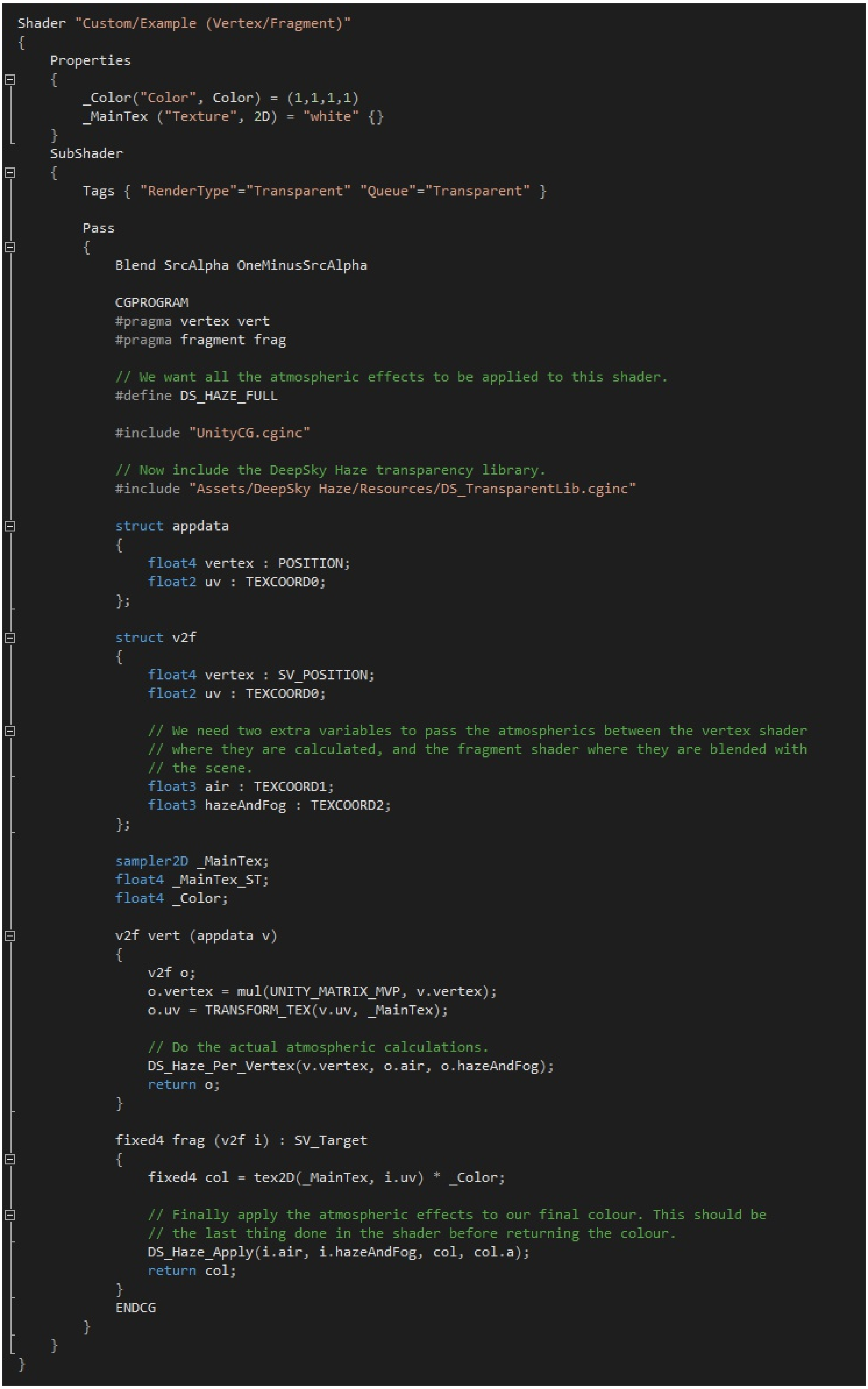
When adding support for DeepSky Haze it will need to become this:

```
struct Input{
	float2 uv_MainTex;
	DEEPSKY_HAZE_DECLARE_INPUT;
}
```

VERTEX / FRAGMENT SHADERS

Adding DeepSky Haze to vertex and fragment shaders is slightly more complicated, but several helper functions are provided to make it easier.
Note that currently only per-vertex atmospherics are supported via the helpers. Using #define DS_HAZE_APPLY_PER_FRAGMENT in a vertex/fragment shader will cause errors!

Here is an example (equivalent to the Surface shader earlier):



```
Shader "Custom/Example (Vertex/Fragment)"
{
    Properties
    {
        _Color("Color", Color) = (1,1,1,1)
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        Tags { "RenderType"="Transparent" "Queue"="Transparent" }

        Pass
        {
            Blend SrcAlpha OneMinusSrcAlpha

            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            // We want all the atmospheric effects to be applied to this shader.
            #define DS_HAZE_FULL

            #include "UnityCG.cginc"

            // Now include the DeepSky Haze transparency library.
            #include "Assets/DeepSky Haze/Resources/DS_TransparentLib.cginc"

            struct appdata
            {
                float4 vertex : POSITION;
                float2 uv : TEXCOORD0;
            };

            struct v2f
            {
                float4 vertex : SV_POSITION;
                float2 uv : TEXCOORD0;

                // We need two extra variables to pass the atmospherics between the vertex shader
                // where they are calculated, and the fragment shader where they are blended with
                // the scene.
                float3 air : TEXCOORD1;
                float3 hazeAndFog : TEXCOORD2;
            };

            sampler2D _MainTex;
            float4 _MainTex_ST;
            float4 _Color;

            v2f vert (appdata v)
            {
                v2f o;
                o.vertex = mul(UNITY_MATRIX_MVP, v.vertex);
                o.uv = TRANSFORM_TEX(v.uv, _MainTex);

                // Do the actual atmospheric calculations.
                DS_Haze_Per_Vertex(v.vertex, o.air, o.hazeAndFog);
                return o;
            }

            fixed4 frag (v2f i) : SV_Target
            {
                fixed4 col = tex2D(_MainTex, i.uv) * _Color;

                // Finally apply the atmospheric effects to our final colour. This should be
                // the last thing done in the shader before returning the colour.
                DS_Haze_Apply(i.air, i.hazeAndFog, col, col.a);
                return col;
            }
            ENDCG
        }
    }
}
```

FAQ / TROUBLESHOOTING

Which platforms does DeepSky Haze support?

DeepSky Haze has been tested on Windows with DX9, DX11 and OpenGL Core and Max OSX with OpenGL Core. Mobile platforms are **not** supported. Several projects are also using it successfully on PS4 and XBox One, however please note support for these platforms is limited as testing requires access to development kits!

How can I add atmospherics to my custom shaders?

Any opaque shader rendered as part of the standard geometry pass (before opaque image effects) will have the atmospherics applied correctly along with the rest of the scene.

You can easily add support for DeepSky Haze to your own transparent shaders, see the guide here.

Can’t use MSAA in Forward rendering (Unity versions before 5.5)?

Unfortunately when MSAA is enabled Unity versions **before 5.5** won't allow certain render texture formats to be used, which prevents DeepSky Haze from working. Please disable MSAA to use DeepSky Haze when using Forward rendering in these versions of Unity.

Can’t see any effects?

Make sure the camera has a View component and it is **active**. The View components will disable themselves if there are any errors, such as render texture formats not being supported by the system. If the camera is not in any zones then it will also have nothing to render. Also make sure all Zones are children of the Controller GameObject. Lastly, check the scattering properties and fog extinction are not set to 0 on the zone – that will effectively disable the effects!

Can’t see any volumetric light shafts?

Check the **Direct / Indirect Ratio** parameters in Zones are not zero – this will make all the haze come from the non-volumetric secondary scattering. If you can't see light shafts from the directional light, check the draw distance for shadow cascades in the project’s Quality settings; if it’s too low and distant objects aren’t being rendered into the shadow map then they will also not create light shafts. Also make sure the DS_HazeView component on the camera has a reference to the directional light.

When the camera moves there appears to be ‘ghosting’ or ‘trails’?

This occurs when the temporal blend factor is too high and the temporal rejection scale is too low. Parts of the previous frame are being re-used but the camera has moved far enough for there to be a noticeable difference. Raise the rejection scale and lower the blend factor to remove these artifacts. Use the Show Temporal Rejection debug

Thin geometry/alpha cutouts have fuzzy outlines?

Due to the volumetric haze being rendered at a lower resolution for performance, occasionally the effect can ‘bleed’ over the edges of thin geometry or cause blurriness where it shouldn’t. Increase the Gaussian Depth Threshold on the View to help remove these artifacts.

Moving the Time slider on the Controller (or setting via script) has no effect?

Make sure the Zone the camera is currently in has more than one time-of-day variant and that it has a non-zero weight curve. Also check to see if the Zone has the Solo option checked for any variants, as this will override the blending (note this only applies in the Editor, so if the blending works correctly in a build, this is most likely the issue). Lastly, check the View component on the camera does not have any of the override options set, as these will replace the settings from the Controller.

Zones blend in the wrong direction or ‘pop’?

Make sure the priorities on the Zones are set correctly – a higher number means a higher priority and zones blend from lowest to highest. If the priorities are the same then the smaller zone will take precedent over the larger. Also make sure the **inner yellow** bounding box of a higher priority zone overlaps the **blue outer** bounding box of one with a lower priority. This ensures the higher priority zone is fully ‘on’ when the camera leaves the zone with the lower priority (which could cause a visual ‘pop’).

Moving spot / point shadow-casting light volumes darken or 'pop'?

Unity will not render a shadow map for a light if there are no objects within it's range to cast shadows. This happens automatically during rendering and unfortunately it is not possible to detect when Unity has decided not to create the shadow map. The light volume then gets passed an empty shadow map when it comes to render, causing the entire volume to act as if it's in shadow. This means only the secondary scattering will be visible. The only way to avoid this is to make sure any shadow-casting lights with light volumes always have **at least one** shadow-casting object in range.

Really thick fog/scattering when using Post-Processing Stack and Forward Rendering?

If you are using the open-source post-processing stack effect from Unity Technologies (available from the Asset Store or via Github) please be aware that it can break other post effects that require the depth buffer in Forward rendering. To fix this: find the file **PostProcessingBehaviour.cs** in the post-processing stack asset folder and change **line 153** from this:

```
context.camera.depthTextureMode = flags;
```

to this:

```
context.camera.depthTextureMode |= flags;
```