Atmospheric scattering

This asset has been created by SYNERGY IT company, we develop plurality of amazing asset's created in Uniy3d - with guarantee of perfect quality and technical support for all of our products.
Atmospheric scattering can be used in many various ways starting from simple visualisation to end up with a complex space games.

Introduction

Atmospheric scattering asset is implementation of Sean O'Neil's method (described in GPU Gems 2, Chapter 16. Accurate Atmospheric Scattering) for Unity3D. It allows to realistically simulate light scattering in planet's atmosphere, which causes glow around the planet and transition between sky colors at sunrises and sunsets.

Method is based on a concept of rendering two spheres - one for a planet and one for an atmosphere. Each is rendered with specific material (shader). Additionally, different materials are used, depending on whether camera position is inside or outside the planet's atmosphere. Thus, there are four materials in total.

Calculations of scattering are quite complicated. Optical depths and light scattering in points visible through the atmosphere are computed by numerical integration. Most calculations are performed per vertex (in vertex shader) and the rest (much less) per pixel (in pixel shader).

Designed algorithm assumes thickness of the atmosphere is equal to 2.5% of planet radius and this parameter is constant.

Controlling the effect

Main game object that encompasses both planet and atmosphere game objects is named PlanetWithAtmosphere. From here, you can control all parameters that influence working of the effect. You shouldn't make any changes in subobjects (Planet and Atmosphere)!

Transformations. Translations, rotations and scaling should be done only on PlanetWithAtmosphere object. Changing transformations independently in Planet and Atmosphere objects can and will lead to incorrect rendering of the effect.

Camera. Algorithm is heavily based on information about camera position. Thus it's important to set camera that you will be rendering your scene with to **Camera** property.

Light. In order for the effect to work correctly in your 3D environment, you need to set proper direction of light in **Light Direction** property.. If you use type of light for which position matters (e.g. point light), you should calculate light direction by subtracting light position from PlanetWithAtmosphere object position. However, effect is designed to work with directional light

- use it, if you want to obtain the best result.

Textures. You will get the most of the asset if you set both day and night planet textures (**Planet Day Texture** and **Planet Night Texture** properties). Along with the asset we provide high quality textures of Planet Earth. Use your own textures to adjust the effect to needs of your game.

Light wavelengths. The most important reason why atmospheric scattering looks so impressive is that light of different wavelenghts is scattered differently. That's why sky is blue and sunset is reddish. But this is not necessarily the case in your atmosphere! To control how which colors behave in your atmosphere, modify following properties: **Wavelength Red, Wavelength Green, Wavelength Blue**.

Light Intensity. To control how strongly Sun (or other star that lights your 3D world) affects planet and atmosphere, use **Light Intensity** property.

Light Scattering. Light scattering process is quite complicated and depends on properties of the atmosphere. Rayleigh and Mie scattering are two kinds of scattering that affect final look. To control their behaviour, modify **Rayleigh Scattering Constant** and **Mie Scattering Constant** properties. There is also mysterious **GValue** property that is supposed to control symmetry of scattering. Effects of modyfing it are mostly visible when staying inside the atmosphere and looking towards the sun.

Accuracy of calculations. In introduction, we mentioned that numerical integration is used in computation process. In our case it boils down to using multiple sample points along eye direction - which means FOR loop in vertex shader. The more samples you use, the more exact calculation is. We hardcoded number of two (2) samples to our shaders. We couldn't make it a material property, because Shader Model 2.0 GPU hardware generation doesn't support dynamic flow control - all loops must be unrolled, so number of loop iterations must be known at compile time. If you wish to use less or more samples, you'd have to change value of variable named **_NumSamples** in each of the shaders manually:

```
const float _NumSamples = 2.0f;
```

can be changed to, for example:

```
const float _NumSamples = 4.0f;
```

Two samples, however, look well and are cheap enough to run on most hardware.
NOTE: This it the only time when you will need to make change outside PlanetWithAtmosphere object properties. Unless you're an advanced user, don't try to make any other changes in shaders by yourself - it may cause unexpected results.