To: OpenRails Development Team

From: Carl "DR_AERONAUTICS"

Date: March 20, 2022

Subject: Recoding of the Sky Shader file, *SkyShader.fx*

Memo

Greetings! I wanted to let the developers know that I chose to undertake revising the shader file *SkyShader.fx* because of some noticeable problems with how that file presents the setting and rising sun.

Situation

I'm on the water often, an amateur astronomer, and a private pilot, so I witness many sunsets and sunrises in any given year. My experiences have led me to determine the following six criteria that should be taking place in a sunrise or sunset:

#	Criteria	Description	OpenRails Sim Correct?
1	Blue sky from dawn to dusk	The sky overhead remains blue to some degree during or brighter than Nautical Twilight (Nautical Twilight being the time the geometric center of the sun is less than 12 degrees below the horizon (see Figure 1).	No. The sky turns gray as the setting sun is passing below the horizon and gray as the sun is rising in the morning.
2	No stars visible with sun up	There are no stars visible in the sky when the sun is visible. Typically, the brightest stars will appear in the minutes following sunset and nearly all stars will be visible at the end of nautical twilight.	No. Nearly all stars are visible with the sun a good deal above the horizon.
3	Realistic scattering at sunset and sunrise	The horizon surrounding the sun tends to turn orange when it nears the horizon. On very hazy days, only the horizon turns fully red. The sky opposite the sunset will remain blue on a clear weather day.	No. In the evening, the sun remains white for a good period of time, then turns pure yellow and the surrounding sky pure red. In the morning, color scattering is totally absent.

4	No unrealistic rapid color changes	Simply put, the process of sunset is a slow color evolution that takes place gradually over the course of several minutes to hours. There is no rapid change other than the sun disappearing below the horizon, removing direct sunlight from the world.	No. A few minutes before sunset the entire sky flashes from red to yellow, back to red, and on to gray quickly.
5	Dark ground at night	After astronomical twilight, when outside, and away from the dense city, the ground is dark to the point that people need flashlights to see any ground detail. Everything otherwise is silhouette.	Yes. The ground is sufficiently dark.
6	Timing of the ground becoming dark	As the sun sinks lower, it gradually reduces the amount of light on the ground. But it starts to become noticeable when the sun starts to yellow. By the end of civil twilight, the ground is nearly totally dark.	Yes. The ground is coded to begin darkening at y=0.1, which corresponds to +6 degrees, and be dark at y=-0.1or -6 degrees, the end of civil twilight.

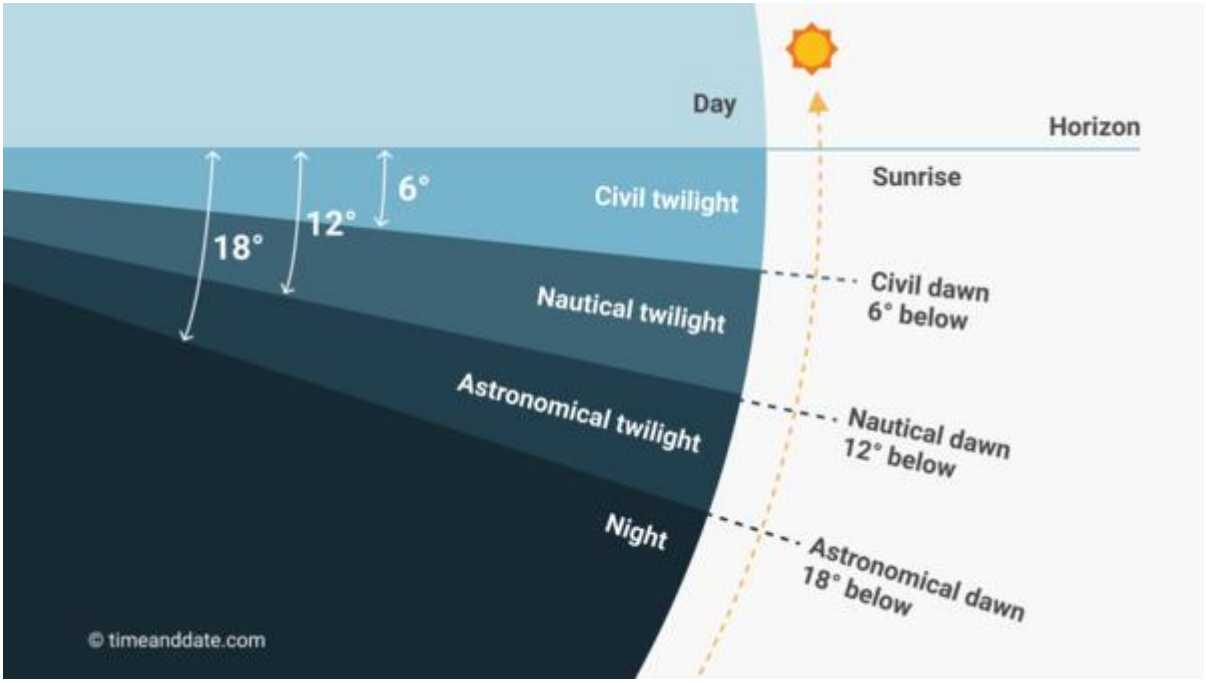


Figure 1- The Twilights

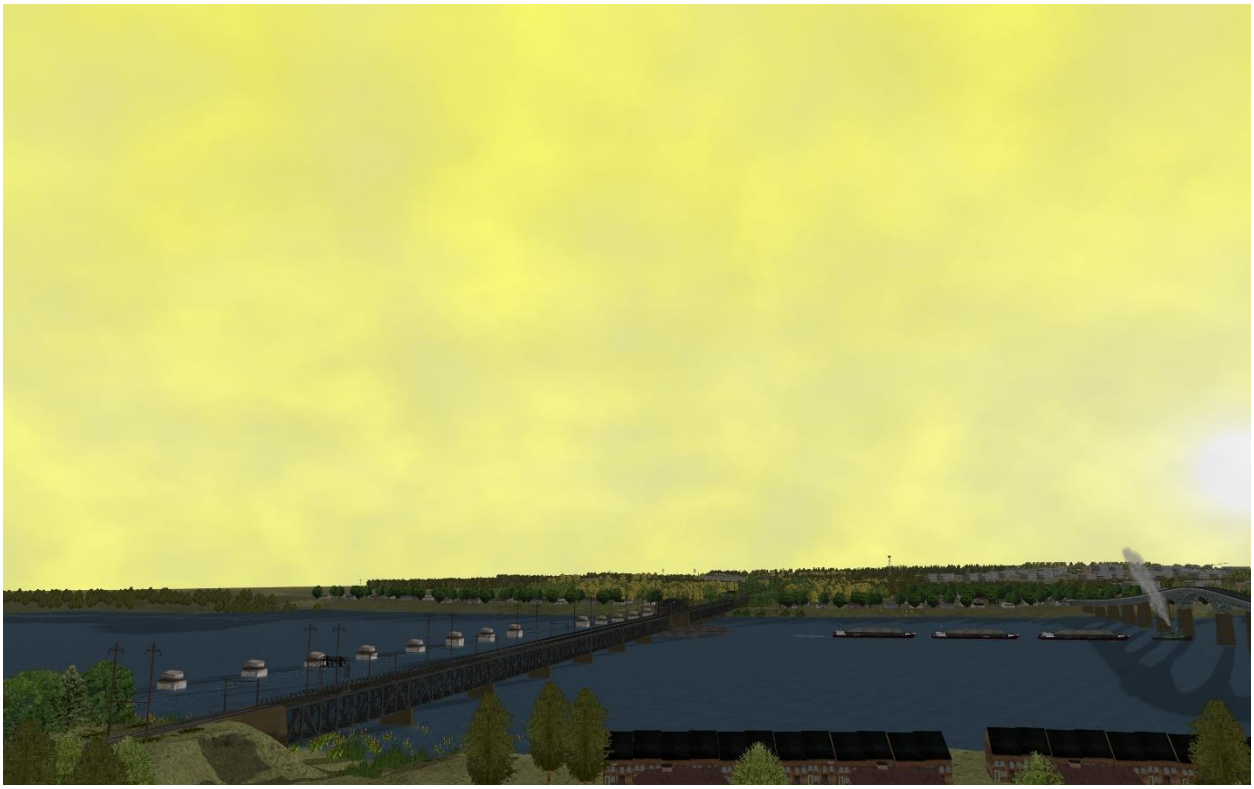
Current State of OpenRails 1.4

Below are some captions from the currently released OpenRails to shore up my claims about the situation.

Sky is not blue in the minutes after sunset.

Stars are visible with the sun.

Oversaturated scattering.



The yellow flash.

The ground is sufficiently dark.

Solution

I have changed the code in SkyShader.fx and the minimum darkness coefficient from ParticleEmitterShader.fx to render a darker night (just changed the line 144 from 0.15 to 0.03).

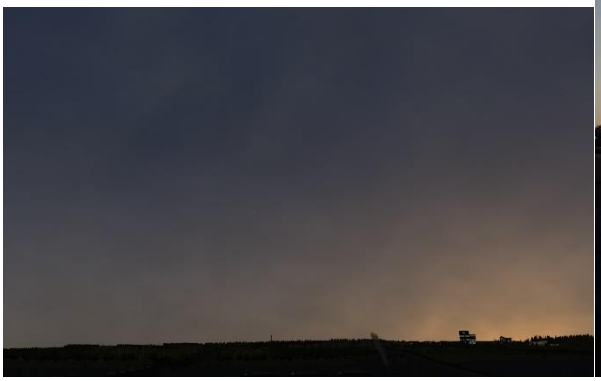
The update has been tested according to the below matrix. This includes testing in all seasons, weather, and three hemispheres. Locations were Pennsylvania USA, Norway, and Australia.

	PRR Spring	PRR Summer	PRR Autumn	PRR Winter	PRR Winter Snow	PRR Summer Rain	Winter VBSR	Summer Blue Mts.
Analog Date	21-Mar	23-Jun	20-Sep	27-Dec	27-Dec	23-Jun	28-Dec	20-Dec
Event								
+15S	16:53	18:05	16:41	14:57	14:57	18:05	DNE	17:44
Geometric Sunset	18:12	19:30	18:00	16:38	16:38	19:30	15:07	19:17
EECT	18:45	20:09	18:31	17:15	17:15	20:09	16:14	19:37
EENT	19:17	20:51	19:03	17:50	17:50	20:51	17:10	20:12
EEAT	19:48	21:40	19:35	18:22	18:22	21:40	18:02	20:51
BMAT	4:31	2:32	4:18	5:46	5:46	2:32	6:34	3:00
BMNT	5:02	3:21	4:51	6:19	6:19	3:21	7:25	3:39
BMCT	5:34	4:02	5:23	6:53	6:53	4:02	8:22	4:15
Geometric Sunrise	6:05	4:41	5:55	7:28	7:28	4:41	9:29	4:49
+15R	7:24	6:07	7:14	9:11	9:11	6:07	DNE	6:07
Sky blue till EENT?	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
No stars with sun?	PASS	PASS	PASS	PASS	DNA	DNA	PASS	PASS
Sky color citrus?	PASS	PASS	PASS	PASS	DNA	DNA	PASS	PASS
No flash?	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
Ground fully dark?	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
Gnd dark at EECT?	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS

DNA = Does Not Apply (typically used when clouds should cover the sun)

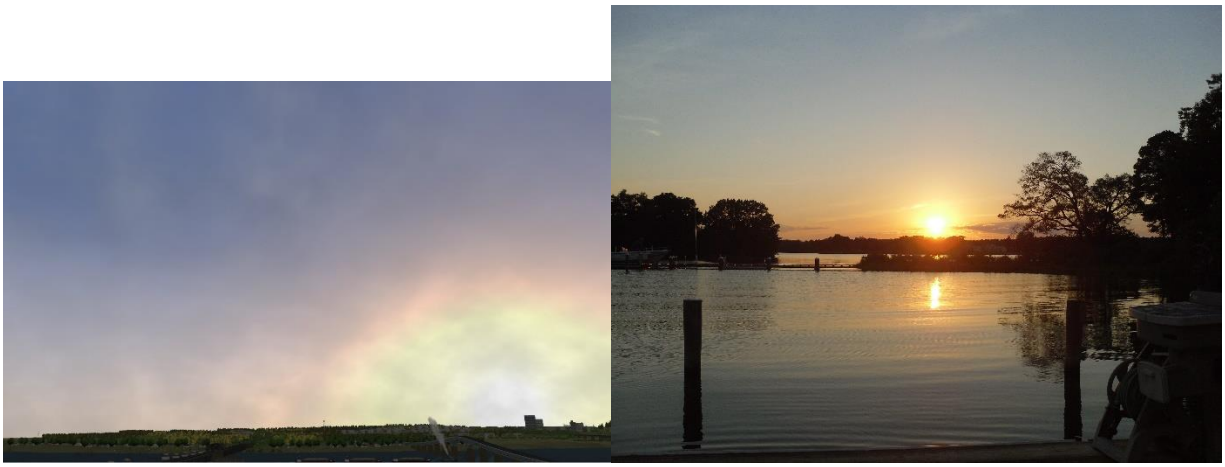
Results

I have the following images below which are taken with the updated *Skyshader.fx* code active. I also compare them to real life images alongside.




Sky is some amount of blue until nautical twilight.

Stars do not appear until about 10 minutes after sunset.



Scattering looks more true to real life.

Sunrise scattering now supported! It is a tad bit more yellow.

Ground is still truly dark.

Videos

For additional information and insight into what this looks like in-game, you can visit my YouTube channel.

- 2 Minute video comparing a real life timelapse of sunset to the original OpenRails *SkyShader.fx* and the new version of *SkyShader.fx* with these improvements applied: https://www.youtube.com/watch?v=6NHYx8R1vWo
- 2 Hour video showing the *SkyShader.fx* fixes in the sunset timeframe as I drive a train in realtime in Maryland: https://youtu.be/OvW9w8UGZaI

Code and Conclusion

The code is pasted below starting on the next page. I have also pasted a diff file report to save you time for what I have changed. The diff file starts on page 22. I strongly urge you to consider committing these changes to the next build of OpenRails. I believe they make the simulator so much more immersive and enjoyable. To get in contact with me, or give me your comments, questions, or concerns, please email me and I will be happy to personally respond.

Carl Hansen

capsfancah@verizon.net

```
// COPYRIGHT 2010, 2011, 2012, 2013 by the Open Rails project.
//
// This file is part of Open Rails.
//
// Open Rails is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// Open Rails is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with Open Rails.  If not, see <http://www.gnu.org/licenses/>.

// This file is the responsibility of the 3D & Environment Team.

////////////////////////////////////////////////////////////////////////////
//                  S H A D O W   M A P   S H A D E R                     //
////////////////////////////////////////////////////////////////////////////

////////////////////    G L O B A L   V A L U E S    ////////////////////////

float4x4 WorldViewProjection;  // model -> world -> view -> projection
float4   LightVector;  // Direction vector to sun, w = 1/length of vector
float    Time;  // Used for moving textures across the sky
```

```
float4   Overcast;  // x = alpha, y = contrast, z = brightness, w = !Overcast.y && !Overcast.z
float2   WindDisplacement;
float3   SkyColor;
float3   FogColor;
float4   Fog;
float2   MoonColor;
float2   MoonTexCoord;
float    CloudColor;
float3   RightVector;
float3   UpVector;
texture  SkyMapTexture;
texture  StarMapTexture;
texture  MoonMapTexture;
texture  MoonMaskTexture;
texture  CloudMapTexture;

sampler SkyMapSampler = sampler_state
{
        Texture = (SkyMapTexture);
        MAGFILTER = LINEAR;
        MINFILTER = LINEAR;
        MIPFILTER = LINEAR;
        MIPLODBIAS = 0.000000;
        AddressU = Wrap;
        AddressV = Wrap;
};

sampler StarMapSampler = sampler_state
```

```
{
	Texture = (StarMapTexture);
	MAGFILTER = LINEAR;
	MINFILTER = LINEAR;
	MIPFILTER = LINEAR;
	MIPLODBIAS = 0.000000;
	AddressU = wrap;
	AddressV = wrap;
};

sampler MoonMapSampler = sampler_state
{
	Texture = (MoonMapTexture);
	MAGFILTER = LINEAR;
	MINFILTER = LINEAR;
	MIPFILTER = LINEAR;
	MIPLODBIAS = 0.000000;
	AddressU = wrap;
	AddressV = wrap;
};

sampler MoonMaskSampler = sampler_state
{
	Texture = (MoonMaskTexture);
	MAGFILTER = LINEAR;
	MINFILTER = LINEAR;
	MIPFILTER = LINEAR;
	MIPLODBIAS = 0.000000;
```

```
	AddressU = wrap;
	AddressV = wrap;
};

sampler CloudMapSampler = sampler_state
{
	Texture = (CloudMapTexture);
	MAGFILTER = LINEAR;
	MINFILTER = LINEAR;
	MIPFILTER = LINEAR;
	MIPLODBIAS = 0.000000;
	AddressU = wrap;
	AddressV = wrap;
};

////////////////////  V E R T E X  I N P U T S  ////////////////////////

struct VERTEX_INPUT
{
	float4 Pos      : POSITION;
	float3 Normal   : NORMAL;
	float2 TexCoord : TEXCOORD0;
};

////////////////////  V E R T E X  O U T P U T S  ////////////////////////

struct VERTEX_OUTPUT
{
```

```
	float4 Pos      : POSITION;
	float3 Normal   : TEXCOORD0;
	float2 TexCoord : TEXCOORD1;
};

////////////////////  V E R T E X   S H A D E R S  ////////////////////////////

VERTEX_OUTPUT VSSky(VERTEX_INPUT In)
{
	VERTEX_OUTPUT Out = (VERTEX_OUTPUT)0;

	Out.Pos      = mul(WorldViewProjection, In.Pos);
	Out.Normal   = In.Normal;
	Out.TexCoord = In.TexCoord;

	return Out;
}

VERTEX_OUTPUT VSMoon(VERTEX_INPUT In)
{
	VERTEX_OUTPUT Out = (VERTEX_OUTPUT)0;

	float3 position = In.Pos.xyz;
	position += (In.TexCoord.x - 0.5) * RightVector;
	position += (In.TexCoord.y - 0.5) * UpVector;

	Out.Pos      = mul(WorldViewProjection, float4(position, 1));
	Out.Normal   = In.Normal;
```

```
	Out.TexCoord = In.TexCoord;

	return Out;
}

////////////////////  P I X E L   S H A D E R S  ////////////////////////////

// This function adjusts brightness, saturation and contrast
// By Romain Dura aka Romz
// Colors edit by DR_Aeronautics
float3 ContrastSaturationBrightness(float3 color, float brt, float sat, float con)
{
	// Increase or decrease theese values to adjust r, g and b color channels separately
	const float AvgLumR = 0.5;
	const float AvgLumG = 0.5;
	const float AvgLumB = 0.5;

	const float3 LumCoeff = float3(0.2125, 0.7154, 0.0721);

	float3 AvgLumin = float3(AvgLumR, AvgLumG, AvgLumB);
	float3 brtColor = color * brt;
	float intensityf = dot(brtColor, LumCoeff);
	float3 intensity = float3(intensityf, intensityf, intensityf);
	float3 satColor = lerp(intensity, brtColor, sat);
	float3 conColor = lerp(AvgLumin, satColor, con);
	return conColor;
}
```

```
float4 PSSky(VERTEX_OUTPUT In) : COLOR
{
	// Get the color information for the current pixel
	float4 skyColor = tex2D(SkyMapSampler, In.TexCoord);
	float2 TexCoord = float2((1.0 - In.TexCoord.x) + Time, In.TexCoord.y);
	float4 starColor = tex2D(StarMapSampler, TexCoord);

	// Adjust sky color brightness for time of day
	skyColor *= SkyColor.y;

	// Stars (power function keeps stars hidden until after sunset)
	skyColor = lerp(starColor, skyColor, pow(SkyColor.y,0.125));

	// Fogging
	skyColor.rgb = lerp(skyColor.rgb, FogColor.rgb, saturate((1 - In.Normal.y) * Fog.x));

	// Calculate angular difference between LightVector and vertex normal, radians
	float dotproduct = dot(LightVector.xyz, In.Normal);
	float angleRcp = 1 / acos(dotproduct * LightVector.w / length(In.Normal));

	// Sun glow
	// Coefficients selected by the author to achieve the desired appearance - fot limits the
effect
	skyColor += angleRcp * Fog.y;

	// increase orange at sunset and yellow at sunrise - fog limits the effect
	if (LightVector.x < 0)
	{
		// These if-statements prevent the yellow-flash effect
```

```
        if (LightVector.y > 0.13)
        {
            skyColor.rg += SkyColor.z*2 * angleRcp * Fog.z;
            skyColor.r += SkyColor.z*2 * angleRcp * Fog.z;
        }

        else
        {
            skyColor.rg += angleRcp * 0.075 * SkyColor.y;
            skyColor.r += angleRcp * 0.075 * SkyColor.y;
        }
    }
    else
    {
        if (LightVector.y > 0.15)
        {
            skyColor.rg += SkyColor.z*3 * angleRcp * Fog.z;
            skyColor.r += SkyColor.z * angleRcp * Fog.z;
        }

        else
        {
            skyColor.rg += angleRcp * 0.075 * SkyColor.y;
            skyColor.r += pow(angleRcp * 0.075 * SkyColor.y,2);
        }
    }

    // Keep alpha opague
```

```
	skyColor.a = 1.0;
	return skyColor;
}

float4 PSMoon(VERTEX_OUTPUT In) : COLOR
{
	// Get the color information for the current pixel
	float2 TexCoord = float2(MoonTexCoord.x + In.TexCoord.x * 0.5, MoonTexCoord.y + In.TexCoord.y * 0.25);
	float4 moonColor = tex2D(MoonMapSampler, TexCoord);
	float4 moonMask = tex2D(MoonMaskSampler, In.TexCoord);

	// Fade moon during daylight
	moonColor.a *= MoonColor.x;

	// Fogging
	moonColor.rgb = lerp(moonColor.rgb, FogColor.rgb, saturate((1 - In.Normal.y) * Fog.x));

	// Mask stars behind dark side (mask fades in)
	moonColor.a += moonMask.r * MoonColor.y;

	return moonColor;
}

float4 PSClouds(VERTEX_OUTPUT In) : COLOR
{
	// Get the color information for the current pixel
	// Cloud map is tiled. Tiling factor: 4
```

```
	// Move cloud map to suit wind conditions
	float2 TexCoord = float2(In.TexCoord.x * 4 + WindDisplacement.x, In.TexCoord.y * 4 + WindDisplacement.y);
	float4 cloudColor = tex2D(CloudMapSampler, TexCoord);
	float alpha = cloudColor.a;

   // Fogging
   cloudColor.rgb = lerp(cloudColor.rgb, FogColor.rgb, saturate((1 - In.Normal.y) * Fog.x));

   // Adjust amount of overcast by adjusting alpha
	if (Overcast.w)
	{
		alpha += Overcast.x;
		// Reduce contrast and brightness
		float3 color = ContrastSaturationBrightness(cloudColor.xyz, 1.0, Overcast.z, Overcast.y); // Brightness and saturation are really need to be exchanged?
		cloudColor = float4(color, alpha);
	}
	else
	{
		alpha *= Overcast.x;
	}

	// Adjust cloud color brightness for time of day
	cloudColor *= CloudColor;
	cloudColor.a = alpha;
	return cloudColor;
}
```

```
//////////////////////////    T E C H N I Q U E S    ////////////////////////////

// These techniques are all the same, but we'll keep them separate for now.

technique Sky {
   pass Pass_0 {
            VertexShader = compile vs_4_0_level_9_1 VSSky();
            PixelShader = compile ps_4_0_level_9_1 PSSky();
   }
}

technique Moon {
   pass Pass_0 {
            VertexShader = compile vs_4_0_level_9_1 VSMoon();
            PixelShader = compile ps_4_0_level_9_1 PSMoon();
   }
}

technique Clouds {
   pass Pass_0 {
            VertexShader = compile vs_4_0_level_9_1 VSSky();
            PixelShader = compile ps_4_0_level_9_1 PSClouds();
   }
}
```

-DIFF FILE BEGINS ON NEXT PAGE-

```
// COPYRIGHT 2010, 2011, 2012, 2013 by the Open Rails project.
//
// This file is part of Open Rails.
//
// Open Rails is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// Open Rails is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with Open Rails.  If not, see <http://www.gnu.org/licenses/>.

// This file is the responsibility of the 3D & Environment Team.

////////////////////////////////////////////////////////////////////////////
//                S H A D O W   M A P   S H A D E R                       //
////////////////////////////////////////////////////////////////////////////

////////////////////   G L O B A L   V A L U E S   ////////////////////////////

float4x4 WorldViewProjection;  // model -> world -> view -> projection
float4   LightVector;  // Direction vector to sun, w = 1/length of vector
float    Time;  // Used for moving textures across the sky
float4   Overcast;  // x = alpha, y = contrast, z = brightness, w = !Overcast.y && !Overcast.z
```

```
float2   WindDisplacement;
float3   SkyColor;
float3   FogColor;
float4   Fog;
float2   MoonColor;
float2   MoonTexCoord;
float    CloudColor;
float3   RightVector;
float3   UpVector;
texture  SkyMapTexture;
texture  StarMapTexture;
texture  MoonMapTexture;
texture  MoonMaskTexture;
texture  CloudMapTexture;

sampler SkyMapSampler = sampler_state
{
        Texture = (SkyMapTexture);
        MAGFILTER = LINEAR;
        MINFILTER = LINEAR;
        MIPFILTER = LINEAR;
        MIPLODBIAS = 0.000000;
        AddressU = Wrap;
        AddressV = Wrap;
};

sampler StarMapSampler = sampler_state
{
        Texture = (StarMapTexture);
```

```
	MAGFILTER = LINEAR;
	MINFILTER = LINEAR;
	MIPFILTER = LINEAR;
	MIPLODBIAS = 0.000000;
	AddressU = wrap;
	AddressV = wrap;
};

sampler MoonMapSampler = sampler_state
{
	Texture = (MoonMapTexture);
	MAGFILTER = LINEAR;
	MINFILTER = LINEAR;
	MIPFILTER = LINEAR;
	MIPLODBIAS = 0.000000;
	AddressU = wrap;
	AddressV = wrap;
};

sampler MoonMaskSampler = sampler_state
{
	Texture = (MoonMaskTexture);
	MAGFILTER = LINEAR;
	MINFILTER = LINEAR;
	MIPFILTER = LINEAR;
	MIPLODBIAS = 0.000000;
	AddressU = wrap;
	AddressV = wrap;
};
```

```
sampler CloudMapSampler = sampler_state
{
	Texture = (CloudMapTexture);
	MAGFILTER = LINEAR;
	MINFILTER = LINEAR;
	MIPFILTER = LINEAR;
	MIPLODBIAS = 0.000000;
	AddressU = wrap;
	AddressV = wrap;
};

////////////////////  V E R T E X  I N P U T S  ////////////////////////

struct VERTEX_INPUT
{
	float4 Pos     : POSITION;
	float3 Normal   : NORMAL;
	float2 TexCoord : TEXCOORD0;
};

////////////////////  V E R T E X  O U T P U T S  ////////////////////////

struct VERTEX_OUTPUT
{
	float4 Pos     : POSITION;
	float3 Normal   : TEXCOORD0;
	float2 TexCoord : TEXCOORD1;
};
```

```
////////////////////  V E R T E X  S H A D E R S  ////////////////////////////

VERTEX_OUTPUT VSSky(VERTEX_INPUT In)
{
	VERTEX_OUTPUT Out = (VERTEX_OUTPUT)0;

	Out.Pos      = mul(WorldViewProjection, In.Pos);
	Out.Normal   = In.Normal;
	Out.TexCoord = In.TexCoord;

	return Out;
}

VERTEX_OUTPUT VSMoon(VERTEX_INPUT In)
{
	VERTEX_OUTPUT Out = (VERTEX_OUTPUT)0;

	float3 position = In.Pos.xyz;
	position += (In.TexCoord.x - 0.5) * RightVector;
	position += (In.TexCoord.y - 0.5) * UpVector;

	Out.Pos      = mul(WorldViewProjection, float4(position, 1));
	Out.Normal   = In.Normal;
	Out.TexCoord = In.TexCoord;

	return Out;
}
```

```
//////////////////// P I X E L  S H A D E R S ////////////////////////////

// This function adjusts brightness, saturation and contrast
// By Romain Dura aka Romz
// Colors edit by DR_Aeronautics
float3 ContrastSaturationBrightness(float3 color, float brt, float sat, float con)
{
	// Increase or decrease theese values to adjust r, g and b color channels separately
	const float AvgLumR = 0.5;
	const float AvgLumG = 0.5;
	const float AvgLumB = 0.5;

	const float3 LumCoeff = float3(0.2125, 0.7154, 0.0721);

	float3 AvgLumin = float3(AvgLumR, AvgLumG, AvgLumB);
	float3 brtColor = color * brt;
	float intensityf = dot(brtColor, LumCoeff);
	float3 intensity = float3(intensityf, intensityf, intensityf);
	float3 satColor = lerp(intensity, brtColor, sat);
	float3 conColor = lerp(AvgLumin, satColor, con);
	return conColor;
}

float4 PSSky(VERTEX_OUTPUT In) : COLOR
{
	// Get the color information for the current pixel
	float4 skyColor = tex2D(SkyMapSampler, In.TexCoord);
	float2 TexCoord = float2((1.0 - In.TexCoord.x) + Time, In.TexCoord.y);
	float4 starColor = tex2D(StarMapSampler, TexCoord);
```

```
// Adjust sky color brightness for time of day
skyColor *= SkyColor.xy;

// Stars (power function keeps stars hidden until after sunset)
skyColor = lerp(starColor, skyColor, pow(SkyColor.y,0.125));

// Fogging
skyColor.rgb = lerp(skyColor.rgb, FogColor.rgb, saturate((1 - In.Normal.y) * Fog.x));

// Calculate angular difference between LightVector and vertex normal, radians
float dotproduct = dot(LightVector.xyz, In.Normal);
float angleRcp = 1 / acos(dotproduct * LightVector.w / length(In.Normal));

// Sun glow
// Coefficients selected by the author to achieve the desired appearance - fot limits the effect
skyColor += angleRcp * Fog.y;

// increase orange at sunset and yellow at sunrise - fog limits the effect
if (LightVector.x < 0)
{
        // These if-statements prevent the yellow-flash effect
        if (LightVector.y > 0.13)
        {
                skyColor.rg += SkyColor.z*2 * angleRcp * Fog.z;
                skyColor.r += SkyColor.z*2 * angleRcp * Fog.z;
        }

        else
```

```
            {
                skyColor.g += rg += angleRcp * 0.075 * SkyColor.y;
                skyColor.r * Fog.w+= angleRcp * 0.075 * SkyColor.y;
            }
        }
        else
        {
            if (LightVector.y > 0.15)
            {
                skyColor.rg += SkyColor.z*3 * angleRcp * Fog.z;
                skyColor.r += SkyColor.z * angleRcp * Fog.z;
            }

            else
            {
                skyColor.rg += angleRcp * 0.075 * SkyColor.y;
                skyColor.r += pow(angleRcp * 0.075 * SkyColor.y,2);
            }
        }

        // Keep alpha opague
        skyColor.a = 1.0;
        return skyColor;
}

float4 PSMoon(VERTEX_OUTPUT In) : COLOR
{
        // Get the color information for the current pixel
```

```
	float2 TexCoord = float2(MoonTexCoord.x + In.TexCoord.x * 0.5, MoonTexCoord.y + In.TexCoord.y * 0.25);
	float4 moonColor = tex2D(MoonMapSampler, TexCoord);
	float4 moonMask = tex2D(MoonMaskSampler, In.TexCoord);

	// Fade moon during daylight
	moonColor.a *= MoonColor.x;

	// Fogging
	moonColor.rgb = lerp(moonColor.rgb, FogColor.rgb, saturate((1 - In.Normal.y) * Fog.x));

	// Mask stars behind dark side (mask fades in)
	moonColor.a += moonMask.r * MoonColor.y;

	return moonColor;
}

float4 PSClouds(VERTEX_OUTPUT In) : COLOR
{
	// Get the color information for the current pixel
	// Cloud map is tiled. Tiling factor: 4
	// Move cloud map to suit wind conditions
	float2 TexCoord = float2(In.TexCoord.x * 4 + WindDisplacement.x, In.TexCoord.y * 4 + WindDisplacement.y);
	float4 cloudColor = tex2D(CloudMapSampler, TexCoord);
	float alpha = cloudColor.a;

   // Fogging
   cloudColor.rgb = lerp(cloudColor.rgb, FogColor.rgb, saturate((1 - In.Normal.y) * Fog.x));
```

```
   // Adjust amount of overcast by adjusting alpha
        if (Overcast.w)
        {
            alpha += Overcast.x;
            // Reduce contrast and brightness
            float3 color = ContrastSaturationBrightness(cloudColor.xyz, 1.0, Overcast.z, Overcast.y);
// Brightness and saturation are really need to be exchanged?
            cloudColor = float4(color, alpha);
        }
        else
        {
            alpha *= Overcast.x;
        }

        // Adjust cloud color brightness for time of day
        cloudColor *= CloudColor;
        cloudColor.a = alpha;
        return cloudColor;
}

//////////////////////////  T E C H N I Q U E S  ////////////////////////////////

// These techniques are all the same, but we'll keep them separate for now.

technique Sky {
  pass Pass_0 {
          VertexShader = compile vs_4_0_level_9_1 VSSky();
          PixelShader = compile ps_4_0_level_9_1 PSSky();
```

```
  }
}

technique Moon {
  pass Pass_0 {
        VertexShader = compile vs_4_0_level_9_1 VSMoon();
        PixelShader = compile ps_4_0_level_9_1 PSMoon();
  }
}

technique Clouds {
  pass Pass_0 {
        VertexShader = compile vs_4_0_level_9_1 VSSky();
        PixelShader = compile ps_4_0_level_9_1 PSClouds();
  }
}
```