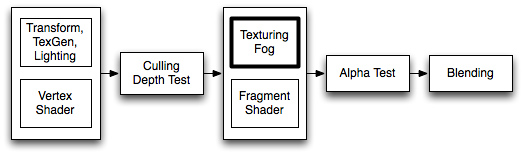
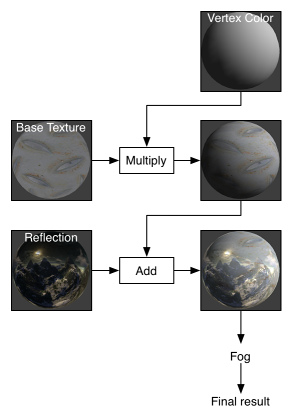
ShaderLab syntax: Texturing  
  
  
After the basic vertex lighting has been calculated, textures are applied. In ShaderLab this is done usingSetTexture command.  
  
在基本的顶点照明计算完成以后，纹理将被设置。在ShaderLab中用SetTexture命令来设置纹理。  
  
SetTexture commands have no effect when fragment programs are used; as in that case pixel operations are completely described in the shader.  
  
当使用片段编程时候SetTexture 命令无效。在这种情况下shader里面完全描述像素级操作。  
  
  
Texturing is the place to do old-style combiner effects. You can have multiple SetTexture commands inside a pass- all textures are applied in sequence, like layers in a painting program. SetTexture commands must be placed at the end of a Pass.  
  
纹理处理是在做旧式的合并效果。你可以在1个pass中使用多个SetTexture命令--所有的纹理都在一个队列当中，就像多个图层在1个绘图程序中一样。SetTexture 命令必须处在Pass的末尾。  
Syntax  
SetTexture [TexturePropertyName] { Texture Block }  
Assigns a texture. TextureName must be defined as a texture property. How to apply the texture is defined inside the TextureBlock.  
指派一个纹理，TextureName 必须是被当做纹理的属性来定义。在TextureBlock中设置如何定义这个纹理。  
  
The texture block controls how the texture is applied. Inside the texture block can be up to three commands:combine, matrix and constantColor.  
  
TextureBlock控制了如何设置这些纹理。在这个块中可以使用3个命令：combine, matrix , constantColor.  
Texture block combine command  
  
combine 命令  
combine src1 \* src2  
Multiplies src1 and src2 together. The result will be darker than either input.  
src1和src2相乘，结果纹理将比输入纹理更暗。  
  
combine src1 + src2  
Adds src1 and src2 together. The result will be lighter than either input.  
src1和src2相加，结果纹理将比输入纹理更明亮。  
  
combine src1 - src2  
Subtracts src2 from src1.  
src1和src2相减  
  
combine src1 +- src2  
Adds src1 to src2, then subtracts 0.5 (a signed add).   
src1和src2相加，然后减去0.5(有符号相加)  
combine src1 lerp (src2) src3  
Interpolates between src3 and src1, using the alpha of src2. Note that the interpolation is opposite direction:src1 is used when alpha is one, and src3 is used when alpha is zero.   
在src3和src1之间做插值，使用src2的alpha值。请注意插值是相反的方向，当alpha为1时候src1使用，当alpha为0时候src3使用。  
combine src1 \* src2 + src3  
Multiplies src1 with the alpha component of src2, then adds src3.   
src1与src2的alpha值混合，然后添加src3.  
combine src1 \* src2 +- src3  
Multiplies src1 with the alpha component of src2, then does a signed add with src3.   
src1与src2的alpha值混合，然后和有符号的src3相加.  
  
combine src1 \* src2 - src3  
Multiplies src1 with the alpha component of src2, then subtracts src3.  
src1和src2的alpha值相乘，然后和src3相减。  
  
All the src properties can be either one of previous, constant, primary or texture.  
  
所有的src属性可以是previous, constant, primary 或texture  
Previous is the the result of the previous SetTexture. 在使用SetTexture命令之前的纹理  
Primary is the color from the lighting calculation or the vertex color if it is bound. 从光照计算或者受限的顶点颜色中得到的颜色  
Texture is the color of the texture specified by [\_TextureName] in the SetTexture (see above). 从SetTexture [\_TextureName]中指定的纹理颜色  
Constant is the color specified in ConstantColor. 常量颜色从ConstantColor设置中得到  
Modifiers: 修正：  
The formulas specified above can optionally be followed by the keywords Double or Quad to make the resulting color 2x or 4x as bright.  
上述所有的公式可以选择设置关键词Double或者Quad使结果的颜色2倍或者4倍明亮。  
All the src properties, except lerp argument, can optionally be preceded by one - to make the resulting color negated.  
所有的src属性，除lerp外，可以使用符号-来使最终颜色反相。  
All the src properties can be followed by alpha to take only the alpha channel.  
所有的src属性后面可以有且仅有1个alpha.  
Texture block constantColor command 常量颜色命令  
ConstantColor color  
Defines a constant color that can be used in the combine command.  
定义1个常量颜色，在combina命令中使用  
Texture block matrix command  
matrix [MatrixPropertyName]  
Transforms texture coordinates used in this command with the given matrix.  
利用此命令使用指定的矩阵转换纹理坐标  
Details 详情  
  
Older graphics cards use a layered approach to textures. The textures are applied one after each other, modifying the color that will be written to the screen. For each texture, the texture is typically combined with the result of the previous operation.  
  
老的显卡使用分层的办法来处理纹理。1个纹理在另外1个纹理之后，修改颜色将被显示在屏幕上面。对于每个纹理，纹理通常和前1个操作的结果合并。  
  
  
Note that on "true fixed function" devices (OpenGL, OpenGL ES 1.1, Wii) the value of each SetTexture stage isclamped to 0..1 range. Everywhere else (Direct3D, OpenGL ES 2.0) the range may or may not be higher. This might affect SetTexture stages that can produce values higher than 1.0.  
  
请注意，“真正的固定功能"设备(OpenGL ES1.1,wii,OpenGL)中，每个SetTexture阶段的值限制在0 ..1范围内。在其他地方（Direct3D,OpenGL ES 2.0）的范围可能会或可能不会更高。这可能会影响SetTexture阶段，因为可以产生值高于1.0的值。  
Separate Alpha & Color computation 单独的Alpha计算和颜色计算  
  
By default, the combiner formula is used for calculating both the RGB and alpha component of the color. Optionally, you can specify a separate formula for the alpha calculation. This looks like this:  
  
在默认情况下，合并公式是计算颜色的RGB和Alpha分量。你可以选择指定一个独立的公式去进行alpha计算，就像这样：  
  
SetTexture [\_MainTex] { combine previous \* texture, previous + texture }  
  
Here, we multiply the RGB colors and add the alpha.  
  
这里我们将RGB颜色相乘然后添加alpha.  
Specular highlights  
  
By default the primary color is the sum of the diffuse, ambient and specular colors (as defined in the Lighting calculation). If you specify SeparateSpecular On in the pass options, the specular color will be added inafter the combiner calculation, rather than before. This is the default behavior of the built-in VertexLit shader.  
  
镜面高光  
默认情况下，主色是漫反射光，环境和镜面的颜色（如在光照计算中定义的）的总和。如果你在pass中指定SeparateSpecular on ，镜面反色光的颜色将在合并计算结束后加入，而不是之前。这是内置的VertexLit(顶点光照)着色器的默认行为。  
Graphics hardware support 显卡硬件支持  
  
Some old graphics cards might not support some texture combine modes, and different cards have different number of SetTexture stages available. The shader author should write separate SubShaders for the cards he or she wants to support.  
  
一些旧的显卡可能不支持一些纹理合并的模式，不同的显卡有不同数量的SetTexture阶段可用。编写shader时候应该要写他或她想要支持的显卡的独立SubShaders。  
  
Graphics cards with pixel shader 1.1 support (NVIDIA GeForce 3 and up, ATI Radeon 8500 and up, Intel 9xx) support all combiner modes and have at least 4 texture stages available. The following table summarizes the hardwaresupport:  
  
支持像素着色器1.1的显卡（NVIDIA公司的GeForce3时，ATI RADEON 8500及以上，英特尔9XX），支持所有合模式，并有至少有4个纹理阶段。下表总结了硬件支持：Card Stage count Combiner modes not supported  
NVIDIA GeForce 3/4Ti and up 4 In OpenGL on Windows, src1\*src2-src3 is notsupported  
NVIDIA TNT2, GeForce 256, GeForce 2, GeForce 4MX 2 In OpenGL on Windows, src1\*src2-src3 is notsupported  
ATI Radeon 9500 and up 4-8 8 in OpenGL, 4 in D3D9  
ATI Radeon 8500-9250 4-6 6 in OpenGL, 4 in D3D9  
ATI Radeon 7500 3   
ATI Rage 2 src1\*src2+src3   
src1\*src2+-src3   
src1\*src2-src3  
  
Examples 例子  
Alpha Blending Two Textures 2个纹理的Alpha混合  
  
This small examples takes two textures. First it sets the first combiner to just take the \_MainTex, then isuses the alpha channel of \_BlendTex to fade in the RGB colors of \_BlendTex  
  
这个小例子，有两个纹理。首先，它设置首次合成只采用\_MainTex，然后是使用\_BlendTex的alpha通道进行\_BlendTex的RGB颜色褪色  
Shader "Examples/2 Alpha Blended Textures" {  
Properties {  
\_MainTex ("Base (RGB)", 2D) = "white" {}  
\_BlendTex ("Alpha Blended (RGBA) ", 2D) = "white" {}  
}  
SubShader {  
Pass {  
// Apply base texture 设置基础纹理  
SetTexture [\_MainTex] {  
combine texture  
}  
// Blend in the alpha texture using the lerp operator 使用插值操作混合alpha纹理  
SetTexture [\_BlendTex] {  
combine texture lerp (texture) previous  
}  
}  
}  
}   
Alpha Controlled Self-illumination alpha控制的自发光  
  
This shader uses the alpha component of the \_MainTex to decide where to apply lighting. It does this by applying the texture to two stages; In the first stage, the alpha value of the texture is used to blend between the vertex color and solid white. In the second stage, the RGB values of the texture are multiplied in.  
  
  
这个shader使用的的\_MainTex的alpha分量去决定在哪应用照明。它分两个阶段来实现这个效果。在第一阶段，纹理的alpha值用于混合顶点颜色和白色固体。在第二阶段中，再乘以纹理的RGB值。  
Shader "Examples/Self-Illumination" {  
Properties {  
\_MainTex ("Base (RGB) Self-Illumination (A)", 2D) = "white" {}  
}  
SubShader {  
Pass {  
// Set up basic white vertex lighting  
Material {  
Diffuse (1,1,1,1)  
Ambient (1,1,1,1)  
}  
Lighting On  
// Use texture alpha to blend up to white (= full illumination)使用纹理alpha和白色混合  
SetTexture [\_MainTex] {  
constantColor (1,1,1,1)  
combine constant lerp(texture) previous  
}  
// Multiply in texture  
SetTexture [\_MainTex] {  
combine previous \* texture  
}  
}  
}  
}   
  
We can do something else for free here, though; instead of blending to solid white, we can add a self-illumination color and blend to that. Note the use of ConstantColor to get a \_SolidColor from the propertiesinto the texture blending.  
  
这里我们就可以发挥想象了。我们可以在属性中设置自发光的颜色,代码如下。  
Shader "Examples/Self-Illumination 2" {  
Properties {  
\_IlluminCol ("Self-Illumination color (RGB)", Color) = (1,1,1,1)  
\_MainTex ("Base (RGB) Self-Illumination (A)", 2D) = "white" {}  
}  
SubShader {  
Pass {  
// Set up basic white vertex lighting  
Material {  
Diffuse (1,1,1,1)  
Ambient (1,1,1,1)  
}  
Lighting On  
// Use texture alpha to blend up to white (= full illumination)  
SetTexture [\_MainTex] {  
// Pull the color property into this blender  
constantColor [\_IlluminCol]  
// And use the texture's alpha to blend between it and  
// vertex color  
combine constant lerp(texture) previous  
}  
// Multiply in texture  
SetTexture [\_MainTex] {  
combine previous \* texture  
}  
}  
}  
}   
  
And finally, we take all the lighting properties of the vertexlit shader and pull that in:  
  
最后我们可以设置顶点光照着色器的光照属性，修改成下面这样：  
Shader "Examples/Self-Illumination 3" {  
Properties {  
\_IlluminCol ("Self-Illumination color (RGB)", Color) = (1,1,1,1)  
\_Color ("Main Color", Color) = (1,1,1,0)  
\_SpecColor ("Spec Color", Color) = (1,1,1,1)  
\_Emission ("Emmisive Color", Color) = (0,0,0,0)  
\_Shininess ("Shininess", Range (0.01, 1)) = 0.7  
\_MainTex ("Base (RGB)", 2D) = "white" {}  
}  
SubShader {  
Pass {  
// Set up basic vertex lighting  
Material {  
Diffuse [\_Color]  
Ambient [\_Color]  
Shininess [\_Shininess]  
Specular [\_SpecColor]  
Emission [\_Emission]  
}  
Lighting On  
// Use texture alpha to blend up to white (= full illumination)  
SetTexture [\_MainTex] {  
constantColor [\_IlluminCol]  
combine constant lerp(texture) previous  
}  
// Multiply in texture  
SetTexture [\_MainTex] {  
combine previous \* texture  
}  
}  
}  
}