CommonSurfaceShader Revisited: Improvements and Experiences

Karsten Schwenk  Yvonne Jung  Gerrit Voß  Timo Sturm  Johannes Behr
Fraunhofer IGD  Fraunhofer IGD  Fraunhofer IDM@NTU  Fraunhofer IGD  Fraunhofer IGD
NTU Singapore

Abstract
We present an improved version of the CommonSurfaceShader node, a modern declarative surface shader for X3D. The new version is better suited for physically-based rendering, has support for anisotropic surfaces, and overcomes some of the limitations the original design had for layered materials. We also present a general discussion of how the node performed in practice.

CR Categories: I.3.7 [COMPUTER GRAPHICS]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: X3D, material description, declarative shader

1 Introduction
Two years ago, at Web3D 2010, the CommonSurfaceShader node [Schwenk et al. 2010] was proposed as an extension to X3D [Web3D Consortium 2008]. Inspired by the “default” material found in many DCC packages and exchange formats, CommonSurfaceShader provides a slim, declarative interface that can capture a wide range of common materials and allows authors to specify surface appearance in a compact and portable way. With regard to X3D’s architecture, the node can be seen as an attempt to close the gap between the portable but outdated Material node and expressive but implementation-dependent shader programs. For an in-depth discussion of related work we refer to the original paper [Schwenk et al. 2010].

Since then, the CommonSurfaceShader node has been in use in InstantReality [Fraunhofer IGD 2012], an X3D-based VR/AR system, where it serves as an up-to-date supplement for the Material node. The system actually contains three implementations for three different rendering back-ends: OpenGL-based rasterization, Whitted-style ray tracing, and photo-realistic path tracing. In addition, CommonSurfaceShader is used in the WebGL-based X3DOM framework [Behr et al. 2009] and a third implementation for the generic scene graph library OpenSG [Reiners et al. 2002] is currently in the works.

In this short paper, we present several improvements to the original proposal and discuss our experiences with the node. Compared to the version presented in [Schwenk et al. 2010], our new design has departed even further from the old X3D Material node and better support for physically-based rendering, HDR lighting, and global illumination. Our paper aims at reviving the discussion about a new, modern surface shading component for the X3D standard, and we hope that our extension will be regarded as a valuable contribution to such a discussion.

2 Improvements
The original proposal consists of a core component that captures diffuse and glossy reflection, a bump mapping component, and a component that models perfect specular reflection and refraction. In this section we describe improvements to the core and perfect specular components.

2.1 Core Component
The diffuse and glossy parts of the original core component (we call them the core-BRDF) are very similar to X3D’s and OpenGL’s default material: a Lambertian diffuse term added to a simple glossy Blinn-Phong lobe. This core-BRDF is familiar and easy to specify, which is why the original design was reluctant to abandon it, but it has some serious limitations.

The biggest problem is that it is not properly normalized (not energy conserving [Lewis 1994]), which makes it ill-suited for HDR rendering and global illumination (Fig. 1). (Of course diffuseFactor and specularFactor can be manually scaled to simulate a normalization, but this is not a practical solution.) To address this issue we dropped the simple Blinn-Phong model. The CommonSurfaceShader now requires a glossy part according to the Blinn-Torrance-Sparrow model (or a close approximation) and a diffuse part modeling a Lambertian surface (or a close approximation). More precisely, we mean the Torrance-Sparrow microfacet model [Torrance and Sparrow 1967; Cook and Torrance 1982] with a Blinn microfacet distribution ($D_1$ in [Blinn 1977]) as described in [Pharr and Humphreys 2004]. Collada uses a similar definition [Khronos Group 2010].

We intentionally left the exact BRDF model unspecified, in order to allow each implementation to find a reasonable balance between complexity and speed (see Section 3 for a discussion). For example, the omission of the geometric attenuation and Fresnel terms is still within the boundaries of “close approximation,” as is the Ashikhmin-Shirley model [Ashikhmin and Shirley 2000] or a properly normalized Blinn-Phong model [Akene-Möller et al. 2008]. We do not demand strict energy conservation, but the core-BRDF should be approximately normalized to prevent excessive energy loss and remain physically plausible. Note that this applies to the whole core-BRDF, i.e. the BRDF for a purely diffuse surface is (approximately) $\text{diffuseFactor}/\pi$, not just $\text{diffuseFactor}$ as it was in the original design.

We also did not specify whether the diffuse and glossy parts should be combined into a layered BRDF or simply be added, as long as the core-BRDF as a whole obeys the constraints given above. The layered model makes sense primarily for plastic, which can be described as a mostly diffuse substrate under a glossy coating. See Section 3 for a discussion of this decision.

As an additional improvement, we now allow an anisotropic glossy part (Fig. 2). For this purpose, $\text{shininessFactor}$ has now type $\text{SFVec2f}$ and holds two values, one for each direction of the uv-parameterization of the surface. If no explicit tangents are provided, they should be calculated from the first textures coordinate set of the geometry, or, if a normal/bump map is present, from the texture coordinates used to sample this texture map. Per default,
The new core component requires a normalized BRDF and a Material instance fit for a specific renderer. CommonSurfaceShader linearly to an exponent $s$ and shininessFactor node. (with a small CommonSurfaceShader underneath fresnelBlend parameter to achieve this effect. $e \in \varepsilon_{\infty}$ now blends (and applied to both in LDR should be modulated should define a surface and how much freedom a perfect shininessTexture shininessFactor $\in [0, 1]$ linearly to an exponent $e \in [0, 512]$, we now use the formula $e = 1 / \max(e, (1 - s)) \in [1, \infty)$ (with a small $e$ to prevent division by zero). This formula is better suited for microfacet models and eases interoperability with DCC tools, because $(1 - s)$ corresponds to the “roughness” parameter used in many implementations [Pharr and Humphreys 2004]. The new default value of shininessFactor is $(0, 0)$, corresponding to an exponent of 1.

### 2.2 Perfect Specular Component

For the perfect specular component, we have changed the way the perfect specular component is combined with the core component. In the original design, the two components are completely independent and just added together. While this is easy to implement, it is not physically plausible. Usually, when combining a diffuse (or glossy) component with a perfect specular one, the intention is to model a layered material with a diffuse layer underneath a perfect specular coating (e.g. finished wood or car paint). Light reflected at the perfect specular layer does not reach the diffuse layer, so the diffuse component has to be attenuated according to the Fresnel equations.

In the current version, we have extended the meaning of the fresnelBlend parameter to achieve this effect. fresnelBlend now blends between a simple addition (fresnelBlend=0) and a physically plausible layered model (fresnelBlend=1). Originally, fresnelBlend was only used to control whether or not the relative strengths of reflection and refraction in the perfect specular component should depend on the angle of incidence (see [Schwenk et al. 2010]). The new fresnelBlend parameter generalizes the old meaning to the whole material and allows authors to choose between a physically plausible behavior according to the Fresnel equations and an physically implausible (but sometimes still useful) behavior. Figure 3 shows the difference between the old and the new blending approach. Currently, we do not demand that all implementations support this behavior, but we have found that such an option is necessary for high-quality renderings.

Two interesting questions are how the material should behave if the core component is partially transparent, and how the backside should be shaded. The backside should be shaded by flipping the normal, just like the core component if no perfect specular component is present. If the core component is fully transparent, the perfect specular component should behave as defined in the original approach (i.e. as far as appearance is concerned there is no core component and the surface is a simple Fresnel interface). Partial transparency in the core component is more complicated, as light can suddenly (partially) pass through the core component of the surface and exit on the other side. Although this opens up some interesting possibilities (like embedding opaque diffuse patches in a glass surface), there are some technical issues that are not easy to resolve and implement (e.g. how to model the Fresnel interface of the backside to ensure reciprocity). We also felt this was beyond the scope of what the CommonSurfaceShader node should handle and therefore decided to leave the behavior for this case undefined.

As a small cosmetic correction, we have removed the environmentTexture and environmentFactor fields from the standard specification and declared them extensions. Environment maps make sense for most renderers and they will certainly be widely used, but in the end they are still an implementation-dependent extension and not an inherent property of the surface.

### 3 Discussion

Moving towards a more physically plausible behavior has many advantages, as detailed in the previous section, but also bears some dangers. Most notably, one runs the risk of losing touch with real-time graphics which is still X3D’s primary domain. However, even in real-time rendering HDR lighting and physically-based shading are standard today [Akenine-Möller et al. 2008]. There can be issues when using the new CommonSurfaceShader in LDR-pipelines without proper tone mapping (clamping of highlights) and with older assets that assume an unnormalized Blinn-Phong model, which was the standard in real-time rendering for many years (overly bright highlights), but usually these are not big problems. In addition, the unnormalized Blinn-Phong lobe is still available in the Material node.

Portal a certain appearance between potentially very different renderers was a key feature of the original approach, and at first sight the new definition of the core-BRDF (Sec. 2.1) may seem a bit too lax to ensure this. There was a long debate about how clearly CommonSurfaceShader should define a surface and how much freedom implementations should have in interpreting this definition, but we think we have found a reasonable compromise. Experience shows that differences between BRDFs compliant with the requirements given in Section 2.1 are usually not significant compared to differences that result from properties of the underlying renderers; and often a certain amount of fine-tuning cannot be avoided to make a CommonSurfaceShader instance fit for a specific renderer. The largest differences usually arise from different shadowing algorithms, the perfect specular component, and indirect/ambient illumination. Figure 4 shows a case where the portability of appearance
Figure 3: Combination of core component and perfect specular component for car paint. Left: The new blending approach treats the core component as a blue diffuse layer underneath the perfect specular component that represents the clear coat. Right: The old approach simply adds the two components, which is not correct and results in a "dull" appearance. All materials in this image use the CommonSurfaceShader. (Viper model courtesy of Chrysler Group / thegizome.com.)

Figure 4: Three purely glossy spheres (no diffuse component) demonstrate the limits of portability of appearance. The path tracer can produce the characteristic metallic look (a). When rendered with rasterization without support for glossy interreflections the appearance is completely different, although the same CommonSurfaceShader is used (b). Even tweaking the shader and adding a diffuse and ambient component does not create the metallic look of the path traced materials (c).

fails spectacularly because indirect illumination is not taken into account, although the same core-BRDF is used. Figure 5, on the other hand, shows a case where the overall look remains the same, although different core-BRDFs are used. All in all, we think insisting on the lowest common denominator, as the original approach did, is too restrictive and does not allow rendering back-ends to play out their individual strengths. What is more, even demanding a single formula does not guarantee portability of the exact appearance – renderers are just too different. On the other hand, we have to acknowledge that for some use cases the differences between compliant core-BRDFs are significant and that in the future, e.g. for an official specification, a more strict definition may be required.

It may be instructive to see how we use this new room for interpretation in our implementations. Currently, we use the Ashikhmin-Shirley model [Ashikhmin and Shirley 2000; Pharr and Humphreys 2004] in our ray tracers, with a single modification: if specularFactor is zero (no glossy part) a Lambertian diffuse term is used, not the diffuse term proposed by Ashikhmin and Shirley. The rasterizers will soon switch to a simpler but faster normalized version of the Blinn-Phong model [Akenine-Möller et al. 2008, Ch. 7]. Compared to Ashikhmin-Shirley, this model lacks the Fresnel term as well as the masking and shadowing terms, but is significantly faster to compute. It also does not have the non-Lambertian diffuse term that is used in Ashikhmin-Shirley to produce the effect of a layered material. Figure 5 shows the same CommonSurfaceShader rendered with different core-BRDFs. Differences are visible, especially at grazing angles where the Fresnel-weighted glossy term and the non-Lambertian diffuse term come into play, but the overall appearance is well preserved.

Then, there is always the danger of losing the balance between generality and simplicity when developing a default material model. Compactness and ease of implementation were two strong points of the original design, and we think this is still true for the improved version. As far as the interface is concerned, we only added a second value for the shininessFactor parameter, in order to account for anisotropic surfaces. The issues with energy conservation and layered materials, which were the most painful limitations we experienced while using the original approach in practice, were resolved without altering the interface. So the new CommonSurfaceShader has almost the same interface as the old one, but is even more broadly applicable. Appendix A contains an overview of the fields of the current iteration of the CommonSurfaceShader node as presented in this paper (with extensions).

Of course, there will always be materials that cannot be captured adequately by a single default shader, no matter how many knobs and dials it has. The most prominent examples are procedural or effect materials. These kinds of materials are the domain of imperative shaders and can be specified in X3D by using the programmable shaders component. Usually this will lead to a loss in portability, although generic shading languages may alleviate this in the future (e.g. MetaSL [Mental Images 2011]). Another option is to provide implementation-specific extensions for the CommonSurfaceShader (e.g. support for light maps). In the long run, a component-based BSDF system, similar to the one used in PBRT [Pharr and Humphreys 2004], may allow greater flexibility while maintaining the high abstraction and portability that come with declarative approaches. However, so far we shied away from such a solution and clung to the pragmatic, easy-to-implement solution CommonSurfaceShader provides.

4 Conclusion

We presented an improved version of the CommonSurfaceShader node, a declarative surface shader designed to overcome X3D’s...
 outdated material model. Compared to the original approach, the current iteration is physically more plausible, supports anisotropic surfaces, and can capture layered materials better. At the same time we tried to keep the interface slim and the node easy to implement. The new version also allows implementations more freedom, as we dropped some overly restrictive parts of the specification.

The current limitations are, in short, what one would expect from such an uber-shader approach: For some surfaces it can only deliver an approximation of the exact appearance, and some materials and effects cannot be captured at all (see [Schwenk et al. 2010] for details). Especially procedural materials pose a problem. All in all, however, we think we have found a reasonable modern default material for X3D.

### A The New CommonSurfaceShader Node

CommonSurfaceShader : SurfaceShader {
    ...
    # core component
    S芬Node [in,out] emissionTexture NULL
    S芬Node [in,out] ambientTexture NULL
    S芬Node [in,out] diffuseTexture NULL
    S芬Node [in,out] specularTexture NULL
    S芬Node [in,out] shininessTexture NULL
    S芬Node [in,out] alphaTexture NULL
    SFVec3f [in,out] emissionFactor 0.0 0.0 0.0
    SFVec3f [in,out] ambientFactor 0.2 0.2 0.2
    SFVec3f [in,out] diffuseFactor 0.8 0.8 0.8
    SFVec3f [in,out] specularFactor 0.0 0.0 0.0
    SFVec3f [in,out] shininessFactor 0.0 0.0
    SF芬at [in,out] alphaFactor 1
    SF芬ool [in,out] invertAlphaTexture FALSE
    # bump mapping component
    S芬Node [in,out] normalTexture NULL
    S芬String [in,out] normalFormat "UNORM"
    SFVec3f [in,out] normalSpace "TANGENT"
    SFVec3f [in,out] normalBias -1 -1 -1
    SF芬loat [in,out] heightTexture NULL
    SF芬loat [in,out] heightScale 1
    # perfect specular component
    S芬Node [in,out] reflectionTexture NULL
    S芬Node [in,out] transmissionTexture NULL
    SFVec3f [in,out] reflectionFactor 0 0 0
    SFVec3f [in,out] transmissionFactor 0 0 0
    SFVec3f [in,out] relativeIndexOfRefraction 1 1 1
    SF芬loat [in,out] fresnelBlend 1.0
    # extensions
    SF芬int32 [in,out] tangentTextureCoordinatesId -1
    SF芬int32 [in,out] binormalTextureCoordinatesId -1

### References


