MaterialX Physically-Based Shading Nodes

Niklas Harrysson - niklas.harrysson@autodesk.com
Doug Smythe - smythe@ilm.com
Jonathan Stone - jstone@lucasfilm.com
July 16, 2019
(Revised October 17, 2019)

Introduction

The MaterialX Specification describes a number of standard nodes that may be used to construct node graphs for the processing of images, procedurally-generated values, coordinates and other data. With the addition of user-defined custom nodes, it is possible to describe complete rendering shaders using node graphs. Up to this point, there has been no standardization of the specific shader-semantic nodes used in these node graph shaders, although with the widespread shift toward physically-based shading, it appears that the industry is settling upon a number of specific BxDF and other functions with standardized parameters and functionality.

This document describes a number of shader-semantic nodes implementing widely-used surface, scattering, emission and volume distribution functions and utility nodes useful is constructing complex layered rendering shaders using node graphs. These nodes in combination with other nodes may be used with the MaterialX shader generation (ShaderGen[1]) system.

[1] https://github.com/materialx/MaterialX/blob/master/documents/DeveloperGuide/ShaderGeneration.md

Table of Contents

Physical Material Model

This section describes the material model used in the MaterialX ShaderGen PBR library and the rules we must follow to be physically plausible.

Scope

A material describes the properties of a surface or medium that involves how it reacts to light. To be efficient, a material model is split into different parts, where each part handles a specific type of light interaction: light being scattered at the surface, light being emitted from a surface, light being scattered inside a medium, etc. The goal of our material model definition is to describe light-material interactions typical for physically plausible rendering systems, including those in feature film production, real-time preview, and game engines.

Our model has support for surface materials, which includes scattering and emission of light from the surface of objects, and volume materials, which includes scattering and emission of light within a participating medium. For lighting, we support local lights and distant light from environments. Geometric modification is supported in the form of bump and normal mapping as well as displacement mapping.

Physically-Plausible Materials

The initial requirements for a physically-plausible material are that it 1) should be energy conserving and 2) support reciprocity. The energy conserving says that the sum of reflected and transmitted light leaving a surface must be less than or equal to the amount of light reaching it. The reciprocity requirement says that if the direction of the traveling light is reversed, the response from the material remains unchanged. That is, the response is identical if the incoming and outgoing directions are swapped. All materials implemented for ShaderGen should respect these requirements and only in rare cases deviate from it when it makes sense for the purpose of artistic freedom.

Quantities and Units

Radiometric quantities are used by the material model for interactions with the renderer. The fundamental radiometric quantity is **radiance** (measured in $Wm^{-2}sr^{-1}$) and gives the intensity of light arriving at, or leaving from, a given point in a given direction. If radiance is integrated over all directions we get **irradiance** (measured in Wm^{-2}), and if we integrate this over surface area we get **power** (measured in W). Input parameters for materials and lights specified in photometric units can be suitably converted to their radiometric counterparts before being submitted to the renderer.

The interpretation of the data types returned by surface and volume shaders are unspecified, and left to the renderer and the shader generator for that renderer to decide. For an OpenGL-type renderer they will be tuples of floats containing radiance calculated directly by the shader node, but for an OSL-type renderer they may be closure primitives that are used by the renderer in the light transport simulation.

In general, a color given as input to the renderer is considered to represent a linear RGB color space. However, there is nothing stopping a renderer from interpreting the color type differently, for instance to hold spectral values. In that case, the shader generator for that renderer needs to handle this in the implementation of the nodes involving the color type.

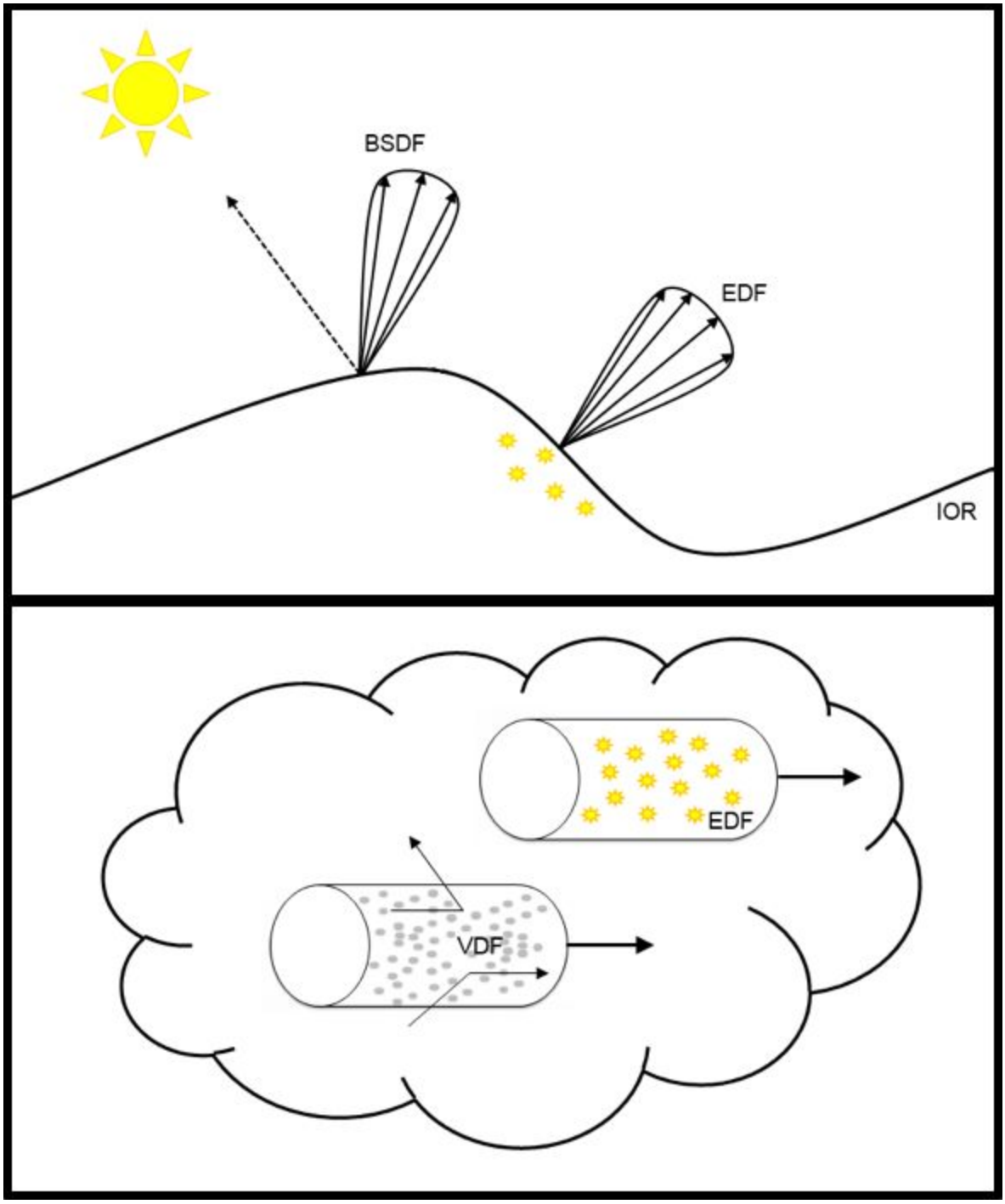
Color Management

MaterialX supports the use of color management systems to associate colors with specific color spaces. A MaterialX document typically specifies the working color space that is to be used for the document as well as the color space in which input values and textures are given. If these color spaces are different from the working color space, it is the application's and shader generator's responsibility to transform them.

The ShaderGen module has an interface that can be used to integrate support for different color management systems. A simplified implementation with some popular and commonly used color transformations are supplied and enabled by default. A full integration of OpenColorIO (http://opencolorio.org) is planned for the future.

Surfaces

In our surface shading model the scattering and emission of light is controlled by distribution functions. Incident light can be reflected off, transmitted through, or absorbed by a surface. This is represented by a Bidirectional Scattering Distribution Function (BSDF). Light can also be emitted from a surface, for instance from a light source or glowing material. This is represented by an Emission Distribution Function (EDF). The PBR library introduce the data types `BSDF` and `EDF` to represent the distribution functions, and there are nodes for constructing, combining and manipulating them.



Another important property is the **index of refraction** (IOR), which describes how light is propagated through a medium. It controls how much a light ray is bent when crossing the interface between two media of different refractive indices. It also determines the amount of light that is reflected and transmitted when reaching the interface, as described by the Fresnel equations.

In reality IOR varies with the wavelength of light. But to simplify our model we use a single

scalar value for dielectric surfaces, and for conductor surfaces we use an artistic parameterization given as reflectance at facing and grazing angles[2]. This parametrization is easier to work with and understand than the physical complex refraction index and extinction coefficients. However, if a physical parameterization is needed this can still be done using a node that convert to the artistic parameterization (see the **Utility Nodes** section below).

A surface shader is represented with the data type `surfaceshader`. In the PBR library there are nodes that construct a `surfaceshader` from a `BSDF` and an `EDF`. Since there are nodes to combine and modify them, you can easily build surface shaders from different combinations of distribution functions. Inputs on the distribution function nodes can be connected, and nodes from the standard library can be combined into complex calculations, giving flexibility for the artist to author material variations over the surfaces.

Layering

In order to simplify authoring of complex materials, our model supports the notion of layering. Typical examples include: adding a layer of clear coat over a car paint material, or putting a layer of dirt or rust over a metal surface. Layering can be done in a couple of different ways:

- Horizontal Layering: A simple way of layering is using per-shading-point linear mixing of different BSDFs where a mix factor is given per BSDF controlling its contribution. Since the weight is calculated per shading point it can be used as a mask to hide contributions on different parts of a surface. The weight can also be calculated dependent on view angle to simulate approximate Fresnel behavior. This type of layering can be done both on a BSDF level and on a surface shader level. The latter is useful for mixing complete shaders which internally contains many BSDFs, e.g. to put dirt over a car paint, grease over a rusty metal or adding decals to a plastic surface. We refer to this type of layering as **horizontal layering** and the various <mix> nodes in the PBR library can be used to achieve this (see below).
- Vertical Layering: A more physically correct form of layering is also supported where a top BSDF layer is placed over another base BSDF layer, and the light not reflected by the top layer is assumed to be transmitted to the base layer; for example, adding a dielectric coating layer over a substrate. The refraction index and roughness of the coating will then affect the attenuation of light reaching the substrate. The substrate can be a transmissive BSDF to transmit the light further, or a reflective BSDF to reflect the light back up through the coating. The substrate can in turn be a reflective BSDF to simulate multiple specular lobes. We refer to this type of layering as **vertical layering** and it is modeled using a base BSDF input on nodes that support this type of layering, where a BSDF representing the base layer can be connected. See <dielectric_brdf>, <sheen_brdf> and <thin_film_brdf> below.
- Shader Input Blending: Calculating and blending many BSDF's or separate surface shaders can be expensive. In some situations good results can be achieved by blending

[2] Ole Gulbrandsen, Artist Friendly Metallic Fresnel. http://jcgt.org/published/0003/04/03/paper.pdf, 2014

the texture/value inputs instead, before any illumination calculations. Typically one would use this with an über-shader that can simulate many different materials, and by masking or blending its inputs over the surface you get the appearance of having multiple layers, but with less expensive texture or value blending. Examples of this are given in the main MaterialX Specification "pre-shader compositing" example.

Bump/Normal Mapping

The surface normal used for shading calculations is supplied as input to each BSDF that requires it. The normal can be perturbed by bump or normal mapping, before it is given to the BSDF. As a result, one can supply different normals for different BSDFs for the same shading point. When layering BSDFs, each layer can use different bump and normal maps.

Surface Thickness

It is common for shading models to differentiate between thick and thin surfaces. We define a **thick surface** as an object where the surface represents a closed watertight interface with a solid interior made of the same material. A typical example is a solid glass object. A **thin surface** on the other hand is defined as an object which doesn't have any thickness or volume, such as a tree leaf or a sheet of paper.

For a thick surface there is no backside visible if the material is opaque. If a backside is hit by accident in this case, the shader should return black to avoid unnecessary computations and possible light leakage. In the case of a transparent thick surface, a backside hit should be treated as light exiting the closed interface. For a thin surface both the front and back side is visible and it can have different materials on each side. If the material is transparent in this case the thin wall makes the light transmit without refracting, like a glass window or bubble.

Two nodes in our material model are used to construct surface shaders for these cases. The <surface> node constructs thick surfaces and is the main node to use since most objects around us have thick surfaces. The <thin_surface> node may be used to construct thin surfaces, and here a different BSDF and EDF may be set on each side. See the **Shader Nodes** section below for more information.

Volumes

In our volume shader model the scattering of light in a participating medium is controlled by a volume distribution function (VDF), with coefficients controlling the rate of absorption and scattering. The VDF represents what physicists call a *phase function,* describing how the light is distributed from its current direction when it is scattered in the medium. This is analogous to how a BSDF describes scattering at a surface, but with one important difference: a VDF is normalized, summing to 1.0 if all directions are considered. Additionally, the amount of

absorption and scattering is controlled by coefficients that gives the rate (probability) per distance traveled in world space. The **absorption coefficient** sets the rate of absorption for light traveling through the medium, and the scattering coefficient sets the rate of which the light is scattered from its current direction. The unit for these are m^{-1}.

Light can also be emitted from a volume. This is represented by an EDF analog to emission from surfaces, but in this context the emission is given as radiance per distance traveled through the medium. The unit for this is $Wm^{-3}sr^{-1}$. The emission distribution is oriented along the current direction.

The <volume> node in the PBR library constructs a volume shader from individual VDF and EDF components. There are also nodes to construct various VDFs, as well as nodes to combine them to build more complex ones.

VDFs can also be used to describe the interior of a surface. A typical example would be to model how light is absorbed or scattered when transmitted through colored glass or turbid water. This is done by connecting a VDF as input to the BSDF node describing the surface transmission; see the **BSDF Nodes** section below.

Lights

Light sources can be divided into environment lights and local lights. Environment lights represent contributions coming from infinitely far away. All other lights are local lights and have a position and extent in space.

Local lights are specified as light shaders assigned to a locator, modeling an explicit light source, or in the form of emissive geometry using an emissive surface shader. The <light> node in the PBR library constructs a light shader from an EDF. There are also nodes to construct various EDFs as well as nodes to combine them to build more complex ones. Emissive properties of surface shaders are also modelled using EDFs; see the **EDF Nodes** section below for more information.

Light contributions coming from far away are handled by environment lights. These are typically photographically-captured or procedurally-generated images that surround the whole scene. This category of lights also includes sources like the sun, where the long distance traveled makes the light essentially directional and without falloff. For all shading points, an environment is seen as being infinitely far away. Environments are work in progress and not yet defined in the PBR library.

MaterialX PBR ShaderGen Library

MaterialX's ShaderGen module includes a library of types and nodes for creating physically plausible materials and lights as described above. This section outlines the content of that library.

Data Types

- `BSDF`: Data type representing a Bidirectional Scattering Distribution Function.
- `EDF`: Data type representing an Emission Distribution Function.
- `VDF`: Data type representing a Volume Distribution Function.

The shadergen PBR nodes also make use of the following standard MaterialX types:

- `surfaceshader`: Data type representing a surface shader.
- `lightshader`: Data type representing a light shader.
- `volumeshader`: Data type representing a volume shader.
- `displacementshader`: Data type representing a displacement shader.

BSDF Nodes

A naming convention is used to differentiate BSDF nodes handling reflection and transmission. Nodes that handle reflection have names ending with "brdf", while nodes that handle transmission have names ending with "btdf"; nodes applicable to both reflection and transmission have names ending with "bsdf".

- **`diffuse_brdf`**: Constructs a diffuse reflection BSDF based on the Oren-Nayar reflectance model[3]. A roughness of 0.0 gives Lambertian reflectance. Parameters and inputs:
  - `weight` (input, float): Weight for this BSDF's contribution, range [0.0, 1.0]. Defaults to 1.0.
  - `color` (input, color3): Diffuse reflectivity (albedo). Defaults to (0.18, 0.18, 0.18).
  - `roughness` (input, float): Surface roughness, range [0.0, 1.0]. Defaults to 0.0.
  - `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.

- **`diffuse_btdf`**: Constructs a diffuse transmission BSDF based on the Lambert reflectance model. Parameters and inputs:
  - `weight` (input, float): Weight for this BSDF's contribution, range [0.0, 1.0]. Defaults to 1.0.
  - `color` (input, color3): Diffuse reflectivity (albedo). Defaults to (1.0, 1.0, 1.0).

[3] M. Oren, S.K. Nayar, Diffuse reflectance from rough surfaces, https://ieeexplore.ieee.org/abstract/document/341163, 1993

- `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.

- **`burley_diffuse_brdf`**: Constructs a diffuse reflection BSDF based on the corresponding component of the Disney Principled[4] model. Parameters and inputs:
  - `weight` (input, float): Weight for this BSDF's contribution, range [0.0, 1.0]. Defaults to 1.0.
  - `color` (input, color3): Diffuse reflectivity (albedo). Defaults to (0.18, 0.18, 0.18).
  - `roughness` (input, float): Surface roughness, range [0.0, 1.0]. Defaults to 0.0.
  - `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.

- **`dielectric_brdf`**: Constructs a reflection BSDF based on a microfacet reflectance model and a Fresnel curve for dielectrics. A BSDF for the surface beneath can be connected to simulate a layered material with vertical layering. By chaining multiple <dielectric_brdf> nodes you can describe a surface with multiple specular lobes. Parameters and inputs:
  - `weight` (input, float): Weight for this BSDF's contribution, range [0.0, 1.0]. Defaults to 1.0.
  - `tint` (input, color3): Color weight to tint the reflected light. Defaults to (1.0, 1.0, 1.0). Note that changing the tint gives non-physical results and should only be done when needed for artistic purposes.
  - `ior` (input, float): Index of refraction of the surface. Defaults to 1.5. If set to 0.0 the Fresnel curve is disabled and reflectivity is controlled only by weight and tint.
  - `roughness` (input, vector2): Surface roughness. Defaults to (0.0, 0.0).
  - `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.
  - `tangent` (input, vector3): Tangent vector of the surface. Defaults to world space tangent.
  - `distribution` (parameter, string): Microfacet distribution type. Defaults to "ggx".
  - `base` (input, BSDF): BSDF for the base surface below the coating. Defaults to "".

- **`dielectric_btdf`**: Constructs a transmission BSDF based on a microfacet reflectance model and a Fresnel curve for dielectrics[5]. A VDF describing the surface interior can be connected to handle absorption and scattering inside the medium, useful for colored glass, turbid water, etc. Parameters and inputs:
  - `weight` (input, float): Weight for this BSDF's contribution, range [0.0, 1.0]. Defaults to 1.0.
  - `tint` (input, color3): Color weight to tint the transmitted light. Defaults to (1.0, 1.0, 1.0). Note that changing the tint gives non-physical results and should only be done when needed for artistic purposes. To simulate colored glass an absorption VDF can

[4] Brent Burley, Physically-Based Shading at Disney, https://disney-animation.s3.amazonaws.com/library/s2012_pbs_disney_brdf_notes_v2.pdf, 2012

[5] Bruce Walter et al., Microfacet Models for Refraction through Rough Surfaces, https://www.cs.cornell.edu/~srm/publications/EGSR07-btdf.pdf, 2007

be used as interior for better physical results.
    - `ior` (input, float): Index of refraction of the surface. Defaults to 1.5.
    - `roughness` (input, vector2): Surface roughness. Defaults to (0.0, 0.0).
    - `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.
    - `tangent` (input, vector3): Tangent vector of the surface. Defaults to world space tangent.
    - `distribution` (parameter, string): Microfacet distribution type. Defaults to "ggx".
    - `interior` (input, VDF): VDF for the surface interior. Defaults to "".

- **`conductor_brdf`**: Constructs a reflection BSDF based on a microfacet reflectance model[6]. Uses a Fresnel curve with complex refraction index for conductors/metals, but with an artistic parametrization[7]: reflectivity and edgecolor. If a scientific complex refraction index is needed the <complex_ior> utility node can be connected to handle this. Parameters and inputs:
    - `weight` (input, float): Weight for this BSDF's contribution, range [0.0, 1.0]. Defaults to 1.0.
    - `reflectivity` (input, color3): Reflectivity per color component at facing angles. Defaults to (0.8, 0.8, 0.8).
    - `edge_color` (input, color3): Diffuse reflectivity (albedo). Defaults to (0.18, 0.18, 0.18).
    - `roughness` (input, vector2): Surface roughness. Defaults to (0.0, 0.0).
    - `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.
    - `tangent` (input, vector3): Tangent vector of the surface. Defaults to world space tangent.
    - `distribution` (parameter, string): Microfacet distribution type. Defaults to "ggx".

- **`generalized_schlick_brdf`**: Constructs a reflection BSDF based on a microfacet model and a generalized Schlick Fresnel curve[8]. A BSDF for the surface beneath can be connected to simulate a layered material with vertical layering. By chaining multiple nodes you can describe a surface with multiple specular lobes. Parameters and inputs:
    - `weight` (input, float): Weight for this BSDF's contribution, range [0.0, 1.0]. Defaults to 1.0.
    - `color0` (input, color3): Reflectivity per color component at facing angles. Defaults to (1.0, 1.0, 1.0).
    - `color90` (input, color3): Reflectivity per color component at grazing angles. Defaults to (1.0, 1.0, 1.0).
    - `exponent` (input, float): Exponent for the Schlick blending between color0 and color90. Defaults to 5.0.

[6] Brent Burley, Physically-Based Shading at Disney, https://disney-animation.s3.amazonaws.com/library/s2012_pbs_disney_brdf_notes_v2.pdf, 2012
[7] Ole Gulbrandsen, Artist Friendly Metallic Fresnel. http://jcgt.org/published/0003/04/03/paper.pdf, 2014
[8] Sony Pictures Imageworks, Revisiting Physically Based Shading at Imageworks, https://blog.selfshadow.com/publications/s2017-shading-course/imageworks/s2017_pbs_imageworks_slides.pdf.

- `roughness` (input, vector2): Surface roughness. Defaults to (0.0, 0.0).
- `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.
- `tangent` (input, vector3): Tangent vector of the surface. Defaults to world space tangent.
- `distribution` (parameter, string): Microfacet distribution type. Defaults to "ggx".
- `base` (input, BSDF): BSDF for the base surface below the coating. Defaults to "".

- **`subsurface_brdf`**: Constructs a subsurface scattering BSDF for subsurface scattering within a homogeneous medium. The parameterization is chosen to match random walk Monte Carlo methods as well as approximate empirical methods[9]. Parameters and inputs:
  - `weight` (input, float): Weight for this BSDF's contribution, range [0.0, 1.0]. Defaults to 1.0.
  - `color` (input, color3): Diffuse reflectivity (albedo). Defaults to (0.18, 0.18, 0.18).
  - `radius` (input, vector3): Sets the average distance that light might propagate below the surface before scattering back out. This is also known as the mean free path of the material. The radius can be set for each color component separately. Default is (1, 1, 1).
  - `anisotropy` (input, float): Anisotropy factor, controlling the scattering direction, range [-1.0, 1.0]. Negative values give backwards scattering, positive values give forward scattering, and a value of zero gives uniform scattering. Defaults to 0.0.
  - `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.

- **`sheen_brdf`**: Constructs a microfacet BSDF for the back-scattering properties of cloth-like materials. This can be layered on top of another BSDF by connecting to the base layer input. All energy that is not reflected will be transmitted to the base layer[10]. Parameters and inputs:
  - `weight` (input, float): Weight for this BSDF's contribution, range [0.0, 1.0]. Defaults to 1.0.
  - `color` (input, color3): Sheen reflectivity. Defaults to (1.0, 1.0, 1.0).
  - `roughness` (input, float): Surface roughness, range [0.0, 1.0]. Defaults to 0.2.
  - `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.
  - `base` (input, BSDF): BSDF for the base surface below the sheen layer. Defaults to "".

- **`thin_film_brdf`**: Adds an iridescent thin film layer over a microfacet base BSDF[11]. A connection to the base input is required, as the node is a modifier and cannot be used as a standalone BSDF. Parameters and inputs:

[9] Pixar, Approximate Reflectance Profiles for Efficient Subsurface Scattering, http://graphics.pixar.com/library/ApproxBSSRDF/. 2015
[10] Sony Pictures Imageworks, Production Friendly Microfacet Sheen BRDF, https://blog.selfshadow.com/publications/s2017-shading-course/imageworks/s2017_pbs_imageworks_sheen.pdf
[11] Laurent Belcour, Pascal Barla, A Practical Extension to Microfacet Theory for the Modeling of Varying Iridescence, https://belcour.github.io/blog/research/2017/05/01/brdf-thin-film.html, 2017

- `thickness` (input, float): Thickness of the thin film layer. Default is 0.001.
- `ior` (input, float): Index of refraction of the thin film layer. Default is 1.5.
- `base` (input, BSDF): BSDF for the base surface below the thin film.

EDF Nodes

- **`uniform_edf`**: Constructs an EDF emitting light uniformly in all directions. Parameters and inputs:
  - `color` (input, color3): Radiant emittance of light leaving the surface. Default is (1, 1, 1).

- **`conical_edf`**: Constructs an EDF emitting light inside a cone around the normal direction. Parameters and inputs:
  - `color` (input, color3): Radiant emittance of light leaving the surface. Default is (1, 1, 1).
  - `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.
  - `inner_angle` (parameter, float): Angle of inner cone where intensity falloff starts (given in degrees). Defaults to 60.
  - `outer_angle` (parameter, float): Angle of outer cone where intensity goes to zero (given in degrees). If set to a smaller value than inner angle no falloff will occur within the cone. Defaults to 0.

- **`measured_edf`**: Constructs an EDF emitting light according to a measured IES light profile[12]. Parameters and inputs:
  - `color` (input, color3): Radiant emittance of light leaving the surface. Default is (1, 1, 1).
  - `normal` (input, vector3): Normal vector of the surface. Defaults to world space normal.
  - `file` (parameter, filename): Path to a file containing IES light profile data. Default is "".

VDF Nodes

- **`absorption_vdf`**: Constructs a VDF for pure light absorption. Parameters and inputs:
  - `absorption` (input, vector3): Absorption rate for the medium (rate per distance traveled in the medium, given in m^{-1}). Default is (0, 0, 0).

- **`anisotropic_vdf`**: Constructs a VDF scattering light for a participating medium,

[12]Standard File Format for Electronic Transfer of Photometric Data, https://www.ies.org/product/standard-file-format-for-electronic-transfer-of-photometric-data/

based on the Henyey-Greenstein phase function[13]. Forward, backward and uniform scattering is supported and controlled by the anisotropy parameter. Parameters and inputs:

- `absorption` (input, vector3): Absorption rate for the medium (rate per distance traveled in the medium, given in m^{-1}). Default is (0, 0, 0).
- `scattering` (input, vector3): Scattering rate for the medium (rate per distance traveled in the medium, given in m^{-1}). Default is (0, 0, 0).
- `anisotropy` (input, float): Anisotropy factor, controlling the scattering direction, range [-1.0, 1.0]. Negative values give backwards scattering, positive values give forward scattering, and a value of 0.0 (the default) gives uniform scattering.

Shader Nodes

- **`surface`**: Constructs a surface shader describing light scattering and emission for closed ”thick” objects. If the BSDF is opaque only the front side will reflect light and back sides will be black if visible. If the BSDF is transmissive the surface will act as an interface to a solid volume made from this material. Output type "surfaceshader". Parameters and inputs:
  - `bsdf` (input, BSDF): Bidirectional scattering distribution function for the surface. Default is "".
  - `edf` (input, EDF): Emission distribution function for the surface. If unconnected, then no emission will occur.
  - `opacity` (input, float): Cutout opacity for the surface. Default to 1.0.

- **`thin_surface`**: Constructs a surface shader describing light scattering and emission for non-closed ”thin” objects. The surface is two sided and can describe a different BSDF and EDF for each side. If only one BSDF is connected the same BSDF will be used on both sides. If an EDF is not connected no emission will occur from that side. Output type "surfaceshader". Parameters and inputs:
  - `front_bsdf` (input, BSDF): Bidirectional scattering distribution function for the front side. Default is "".
  - `front_edf` (input, EDF): Emission distribution function for the front side. Default is no emission.
  - `back_bsdf` (input, BSDF): Bidirectional scattering distribution function for the back side. If not provided, the `front_bsdf` BSDF will be used for both sides.
  - `back_edf` (input, EDF): Emission distribution function for the back side. Default is no emission.
  - `opacity` (input, float): Cutout opacity for the surface. Default to 1.0.

- **`volume`**: Constructs a volume shader describing a participating medium. Output type "volumeshader". Parameters and inputs:

[13]Matt Pharr, Wenzel Jakob, Greg Humphreys, Physically Based Rendering: From Theory To Implementation, Chapter 11.2, http://www.pbr-book.org/3ed-2018/Volume_Scattering/Phase_Functions.html

- `vdf` (input, VDF): Volume distribution function for the medium. Default is "".
- `edf` (input, EDF): Emission distribution function for the medium. If unconnected, then no emission will occur.

- **`light`**: Constructs a light shader describing an explicit light source. Output type "lightshader". Parameters and inputs:
  - `edf` (input, EDF): Emission distribution function for the light source. Default is no emission.
  - `intensity` (input, color3): Intensity multiplier for the EDF's emittance. Default to (1.0, 1.0, 1.0).
  - `exposure` (input, float): Exposure control for the EDF's emittance. Default to 0.0.

- **`displacement`**: Constructs a displacement shader describing geometric modification to surfaces. Output type "displacementshader". Parameters and inputs:
  - `displacement` (input, float or vector3): Scalar (along the surface normal direction) or vector displacement (in (dPdu, dPdv, N) tangent/normal space) for each position. Default is 0.
  - `scale` (input, float): Scale factor for the displacement vector. Default is 1.0.

Utility Nodes

- **`mix`**: Mix two same-type distribution functions according to a weight. Performs horizontal layering by linear interpolation between the two inputs: bg∗(1−mix) + fg∗mix. Parameters and inputs:
  - `bg` (input, BSDF or EDF or VDF): The first distribution function. Defaults to "".
  - `fg` (input, same type as `bg`): The second distribution function. Defaults to "".
  - `mix` (input, float): The mixing weight, range [0.0, 1.0]. Default is 0.

- **`add`**: Additively blend two distribution functions of the same type. Parameters and inputs:
  - `in1` (input, BSDF or EDF or VDF): The first distribution function. Defaults to "".
  - `in2` (input, same type as `in1`): The second distribution function. Defaults to "".

- **`multiply`**: Multiply the contribution of a distribution function by a scaling weight. The weight is either a float to attenuate the channels uniformly, or a color which can attenuate the channels separately. To be energy conserving the scaling weight should be no more than 1.0 in any channel. Parameters and inputs:
  - `in1` (input, BSDF or EDF or VDF): The distribution function to scale. Defaults to "".
  - `in2` (input, float or color3): The scaling weight. Default is 1.0.

- **`backfacing`**: Returns true (or 1) if the surface being shaded is seen from the back side (or the inside of a closed object). Returns false (or 0) if seen from the front or the outside of a closed object. Output type "boolean", "integer" or "float"; no parameters or inputs.

- **`roughness_anisotropy`**: Calculates anisotropic surface roughness from a scalar roughness and anisotropy parameterization. An anisotropy value above 0.0 stretches the roughness in the x-direction of the surface tangent. An anisotropy value of 0.0 gives isotropic roughness. The roughness value is squared to achieve a more linear roughness look over the input parameter range [0,1]. Output type "vector2"; parameters and inputs:
  - `roughness` (input, float): Roughness value, range [0.0, 1.0]. Defaults to 0.0.
  - `anisotropy` (input, float): Amount of anisotropy, range [0.0, 1.0]. Defaults to 0.0.

- **`roughness_dual`**: Calculates anisotropic surface roughness from a dual surface roughness parameterization. The roughness is squared to achieve a more linear roughness look over the input parameter range [0,1]. Output type "vector2"; parameters and inputs:
  - `roughness` (input, vector2): Roughness in x and y directions, range [0.0, 1.0]. Defaults to (0.0, 0.0).

- **`glossiness_anisotropy`**: Calculates anisotropic surface roughness from a scalar glossiness and anisotropy parameterization. This node gives the same result as roughness anisotropy except that the glossiness value is an inverted roughness value. To be used as a convenience for shading models using the glossiness parameterization. Output type "vector2"; parameters and inputs:
  - `glossiness` (input, float): Roughness value, range [0.0, 1.0]. Defaults to 0.0.
  - `anisotropy` (input, float): Amount of anisotropy, range [0.0, 1.0]. Defaults to 0.0.

- **`blackbody`**: Returns the radiant emittance of a blackbody radiator with the given temperature. Output type "color3"; parameters and inputs:
  - `temperature` (input, float): Temperature in Kelvin. Default is 5000.

- **`complex_ior`**: Converts complex IOR values to the artistic parameterization reflectivity and edgecolor. Can be used with the conductor_brdf node to input true complex IOR values instead of artistic reflectivity. Output type "multioutput"; parameters, inputs and outputs:
  - `ior` (input, vector3): Index of refraction. Default is (0.18, 0.42, 1.37) (approximate IOR for "gold").
  - `extinction` (input, vector3): Extinction coefficient. Default is (3.42, 2.35, 1.77) (approximate extinction coefficients for "gold").
  - `reflectivity` (**output**, color3): Computed reflectivity per color component at facing angles.
  - `edge_color` (**output**, color3): Computed reflectivity per color component at grazing angles.

- **`artistic_ior`**: Converts the artistic parameterization reflectivity and edgecolor to complex IOR values; this is the inverse of the `complex_ior` node. Output type "multioutput"; parameters, inputs and outputs:
  - `reflectivity` (input, color3): Reflectivity per color component at facing angles. Default is (0.947, 0.776, 0.371).

- `edge_color` (input, color3): Reflectivity per color component at grazing angles. Default is (1.0, 0.982, 0.753).
- `ior` (**output**, vector3): Computed index of refraction.
- `extinction` (**output**, vector3): Computed extinction coefficient.

- **`fresnel`**: Calculate the float-type Fresnel amount from an index of refraction for a dielectric surface. Parameters, inputs and outputs:
  - `ior` (input, float): Index of refraction. Default is 1.5.
  - `normal` (input, vector3): Surface normal vector; defaults to world-space normal.
  - `viewdirection` (input, vector3): Camera view direction; defaults to world space viewdirection.

Shading Model Examples

This section contains examples of shading model implementations using the MaterialX ShaderGen PBR library. For all examples, the shading model is defined via a <nodedef> interface plus a nodegraph implementation. The resulting nodes can be used by a <shaderref> in a MaterialX material definition.

Autodesk Standard Surface

This is a surface shading model used in Autodesk products created by the Solid Angle team for the Arnold renderer. It is an über shader built from ten different BSDF layers[14].

A MaterialX definition and nodegraph implementation of Autodesk Standard Surface can be found here:
https://github.com/Autodesk/standard-surface/blob/master/reference/standard_surface.mtlx

UsdPreviewSurface

This is a shading model proposed by Pixar for USD[15]. It is meant to model a physically based surface that strikes a balance between expressiveness and reliable interchange between current day DCC's and game engines and other real-time rendering clients.

A MaterialX definition and nodegraph implementation of UsdPreviewSurface can be found here:
https://github.com/materialx/MaterialX/blob/master/libraries/bxdf/usd_preview_surface.mtlx

[14] Autodesk, A Surface Standard, https://github.com/Autodesk/standard-surface, 2019.
[15] Pixar, UsdPreviewSurface Proposal. https://graphics.pixar.com/usd/docs/UsdPreviewSurface-Proposal.html, 2018.

