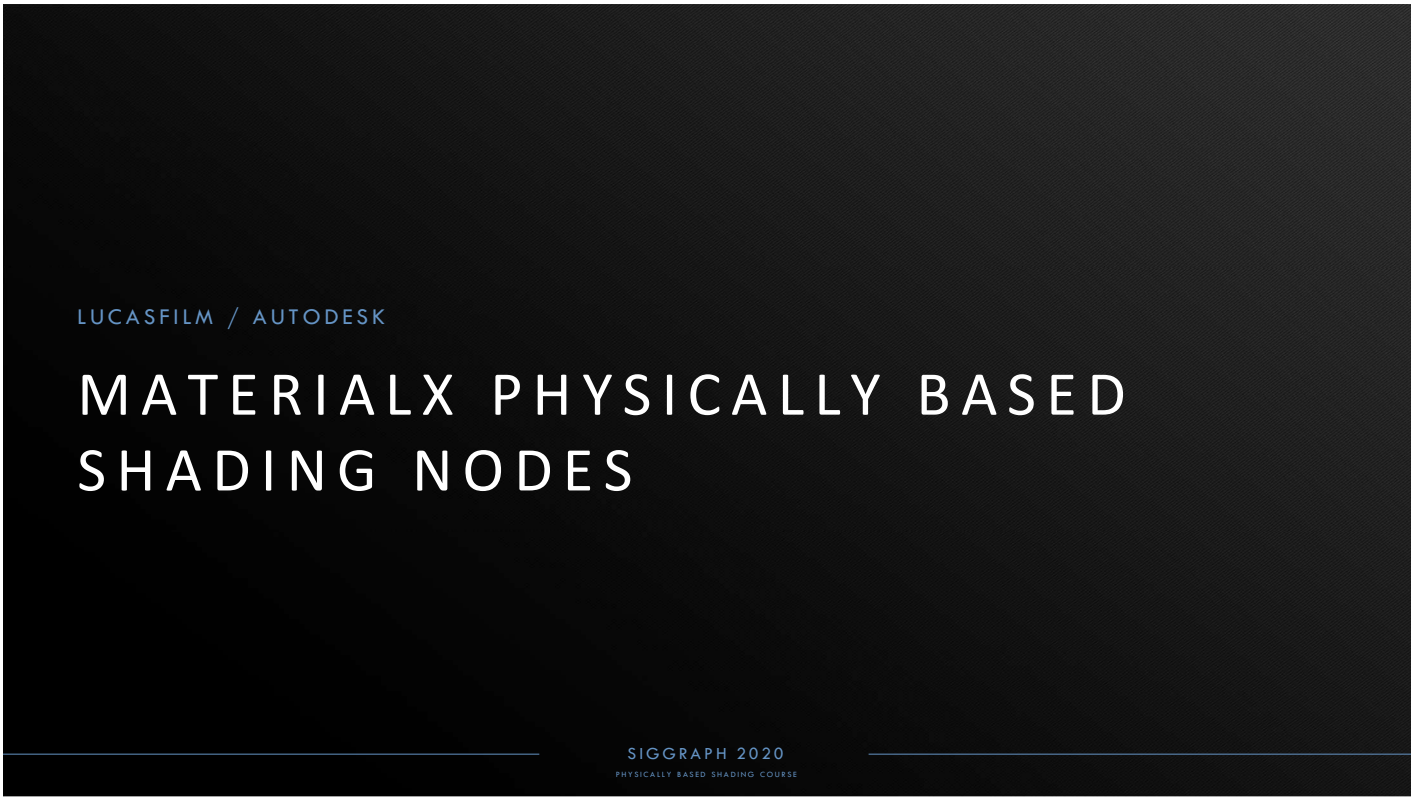

LUCASFILM
Ltd
AUTODESK.



Today we're going to talk about the Physically Based Shading nodes in MaterialX, which are closely related to the theme of this course, and still a relatively new feature of the MaterialX project. I'm Jonathan Stone from the Lucasfilm Advanced Development Group, and my co-presenter today will be Niklas Harrysson from Autodesk.

In this introduction, we'll give a bit of background on what MaterialX is and set the stage for what motivated the development of the Physically Based Shading nodes.

MaterialX was launched at Lucasfilm back in 2012, as part of a broad initiative to build assets that would be independent of the tools in which they were authored, and the media for which they were first created. We knew that there would be new shows and experiences in the Star Wars universe for many years to come, and we wanted to begin building a digital backlot for Star Wars content that would remain useful far into the future.

In the spirit of existing open-source initiatives such as Alembic, OpenEXR, and OpenColorIO, which captured geometry, textures, and color spaces in an application-agnostic fashion, MaterialX was created to describe *materials*, in the expressive, networked form that they were authored by artists in tools such as Mari, Substance, and Maya.

The first film to use MaterialX as its main material format was *Star Wars: The Force Awakens* in 2015, and that same year it was used in its first real-time experience, a virtual reality project called *Trials on Tatooine* from a newly formed Lucasfilm unit named ILMxLAB.

MOTIVATIONS

MATERIALX ORIGINS

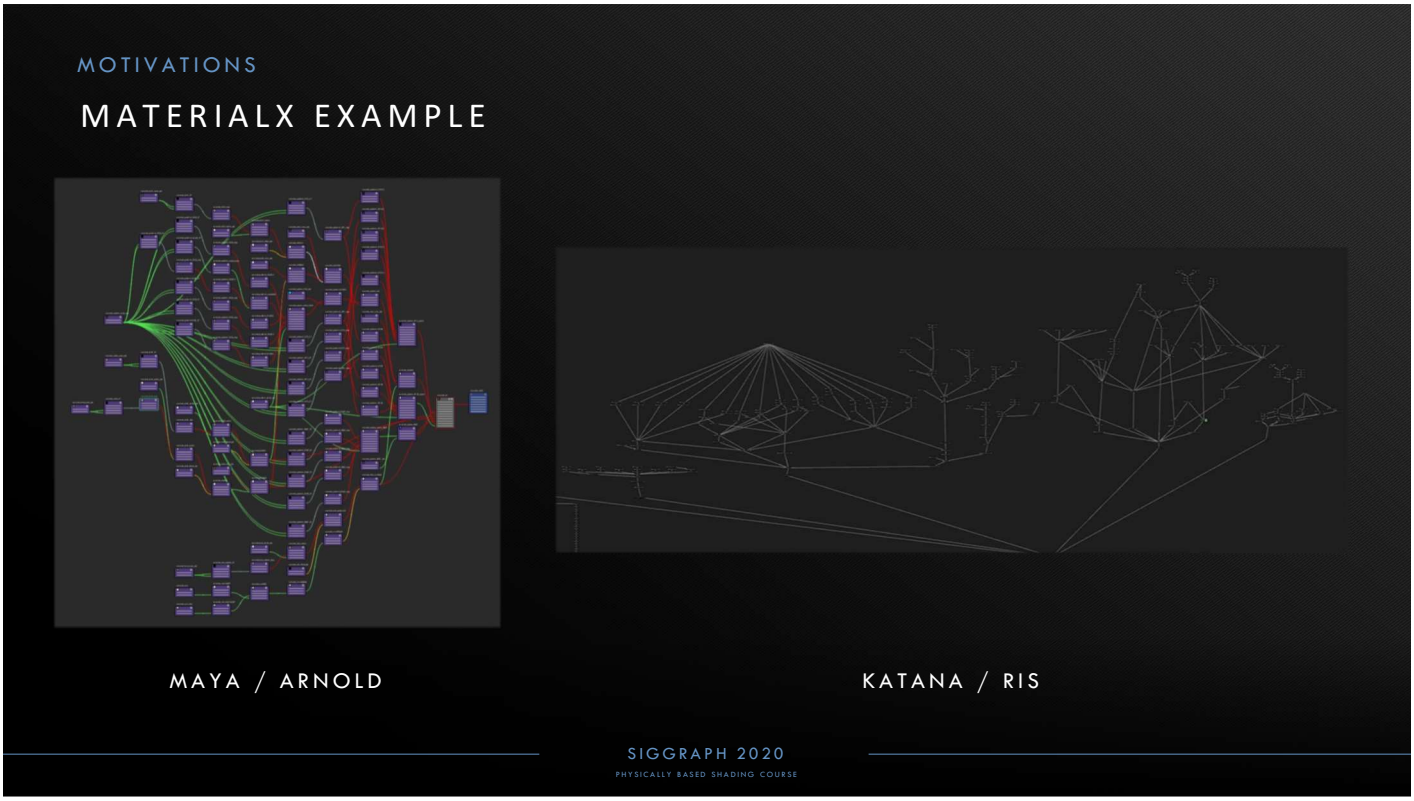
- Builds in its usage on Lucasfilm shows
- Launched collaboration with Autodesk in 2016
- Released as open-source project in 2017

SIGGRAPH 2020
PHYSICALLY BASED SHADING COURSE

As MaterialX was building in its usage at Lucasfilm and becoming our canonical format for materials, we began discussing it as a potential open standard with a widening group of studios and companies across the industry.

One notable discussion from that era was with Autodesk, who had been developing a similar internal standard named Abstract Material Graphs. After learning more about the synergy between the two standards, both of our companies decided to rally development efforts around MaterialX, with Autodesk working to include their own innovations as part of this single standard.

The first public specification for MaterialX was released in 2016, and it became an open-source project on GitHub in 2017.



So, to get a sense of what a MaterialX graph looks like, here's an example from the film *Ready Player One*, where it expressed the surface material for a building in the New York City sequence.

In this image, we're seeing the pattern graphs for a complex surface material asset in two very different environments, Maya using Solid Angle's Arnold renderer, and Katana using Pixar's RIS renderer.

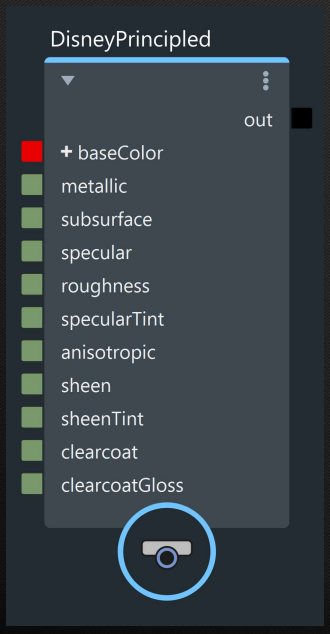
MaterialX was used to describe this shading graph in a unified way across these environments…

… with comparable rendered results, as seen here. This enabled workflows where artists could block out a surface material in one of the two environments, and then finalize and render it in the second, without rebuilding any of the subtle details that they had originally authored.

MOTIVATIONS

SHADING MODELS

- This works well for pattern graphs, but what about the shading model itself?
- It's just as critical that the BSDF remains independent of its authoring environment
- Lucasfilm and Autodesk propose a standard set of BSDF nodes



This classic usage of MaterialX is a step in the right direction, and it works well for the *pattern graphs* that artists author in tools such as Maya and Katana, defining the complex signals that are fed into the inputs of a shading model. In this slide, we're looking at an example of such a shading model, the original *Disney Principled BRDF* that Brent Burley presented in 2012.

But if we're truly aiming to express materials independently of their authoring environment, don't we need to somehow describe the shading model just as rigorously as the patterns that feed into it? Although popular, open models such as Disney Principled have whitepapers that guide the author of a new implementation, there's still significant variation in those implementations across applications and renderers. And in addition, there are the numerous *custom* shading models that studios have designed for their own productions, often relying upon blocks of renderer-specific code rather than portable definitions of the underlying BSDF.

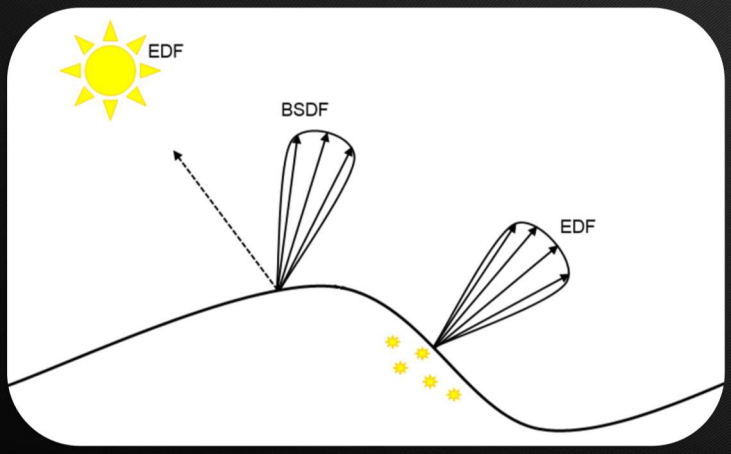
In order to make material assets truly independent of the tools and media for which they were first developed, allowing them to be transferred accurately between studios or stored in digital backlots for reuse far into the future, we'll also need to rigorously express the *shading models* on which those materials are based.

This was the motivation in our minds, and what led Lucasfilm and Autodesk to collaborate on a project to extend the MaterialX node set from pure patterns to include shading-model components, creating a new library of *Physically Based Shading Nodes*.

PHYSICALLY BASED SHADING NODES

MATERIAL MODEL DESCRIPTION

- A set of distribution functions
  - BSDF: light scattering at surfaces
  - EDF: light emitted from surfaces or volumes
  - VDF: light scattering in volumes
- Ways to combine and layer them



At its core, the material model in this library is a set of distribution functions, capturing the scattering and emission of light at surfaces and in volumes.

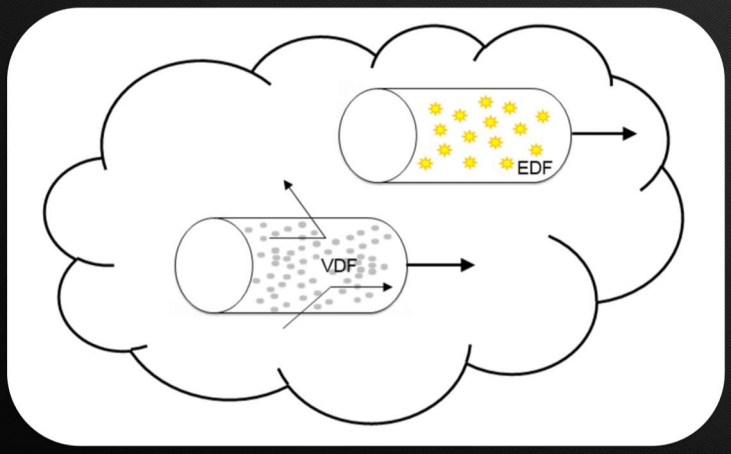
This abstraction exists already in high-level shading languages like OSL and MDL. In OSL there is a collective “closure” type for all the various light interactions, whereas in MDL this is more strictly typed and differentiates between three distribution function types:

Bidirectional Scattering Distribution Functions – light scattering at surfaces
Emission Distribution Functions – light emitted from surfaces or volumes
Volume Distribution Functions – light scattering in volumes or participating media

PHYSICALLY BASED SHADING NODES

MATERIAL MODEL DESCRIPTION

- A set of distribution functions
  - BSDF: light scattering at surfaces
  - EDF: light emitted from surfaces or volumes
  - VDF: light scattering in volumes
- Ways to combine and layer them



We choose to follow this stricter typing since with a separation between the various light interactions we direct the user and prevent creation of nonsensical graphs. Having a strict separation also makes it easier on the implementation side, especially for rasterization where we don't have these concepts natively in the shading language.

In our library, nodes represent different variations of these function types and by combining them you build up your shading model.

BSDF NODES

- Standard diffuse BSDF nodes
- Standard specular BSDF nodes
- Additional BSDF nodes

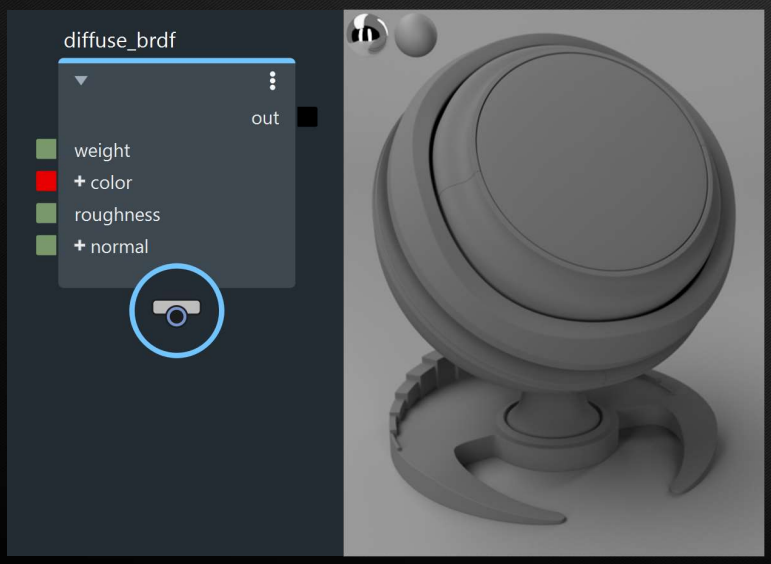
To get an overview of the node library, here's a quick rundown of the different variations of these nodes that are currently available.

For BSDF nodes, we have assembled a set of popular and widespread BRDF models that are commonly used in our industry today.

This is a proposal for a standardization of BRDF models that cover most day to day use cases. This can be extended as future models become commonly used.

BSDF NODES

- Standard diffuse BSDF nodes
- Standard specular BSDF nodes
- Additional BSDF nodes



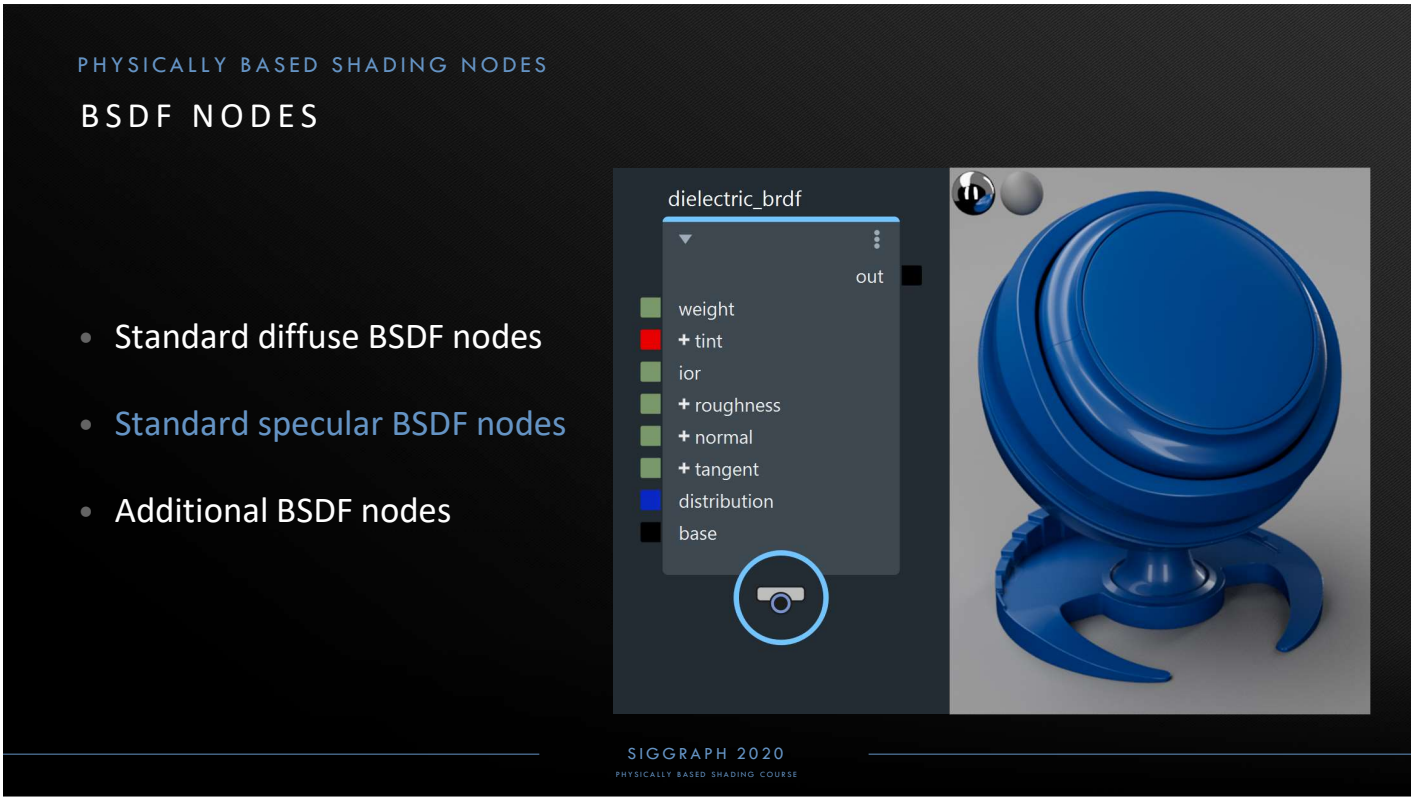
Our standard node for diffuse reflection, based on the Oren-Nayar BRDF.

PHYSICALLY BASED SHADING NODES

BSDF NODES

- Standard diffuse BSDF nodes
- Standard specular BSDF nodes
- Additional BSDF nodes

A second node for diffuse reflections, based on the diffuse component from the Disney Principled BRDF by Brent Burley, thus named burley_diffuse_brdf.



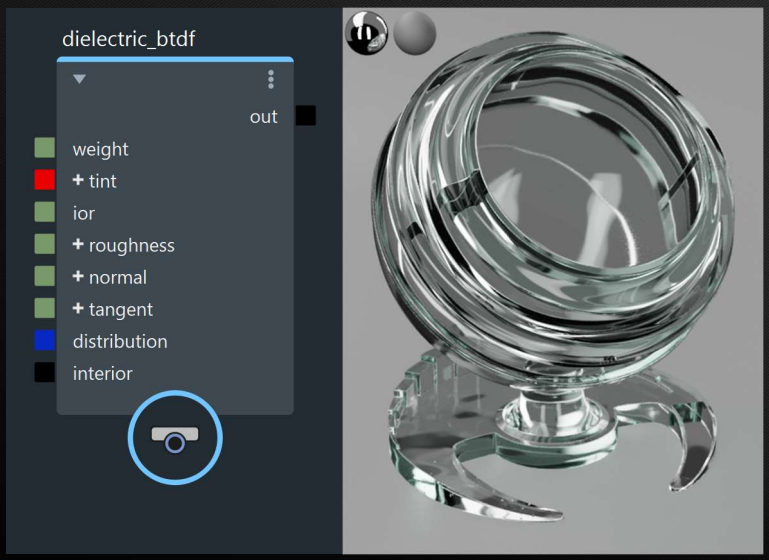
For specular nodes we have split them based on the Fresnel model used:

For dielectric materials, one node for reflection…

PHYSICALLY BASED SHADING NODES

BSDF NODES

- Standard diffuse BSDF nodes
- Standard specular BSDF nodes
- Additional BSDF nodes

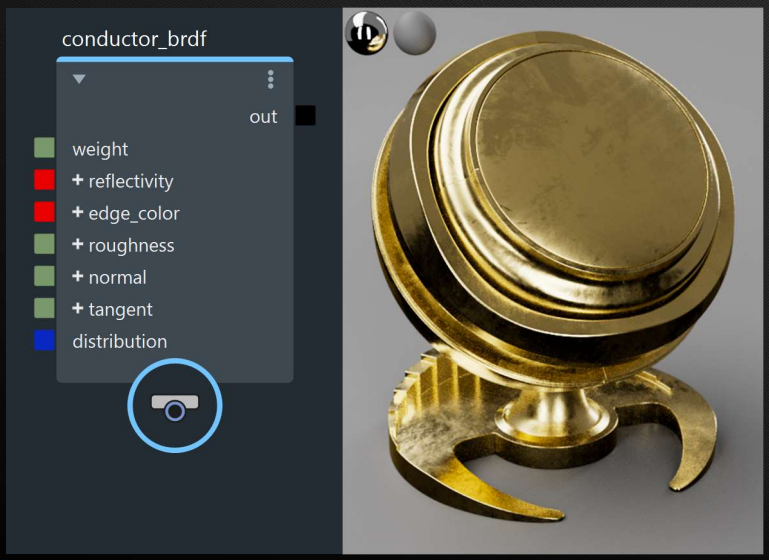


…and another node for transmission.

PHYSICALLY BASED SHADING NODES

BSDF NODES

- Standard diffuse BSDF nodes
- Standard specular BSDF nodes
- Additional BSDF nodes

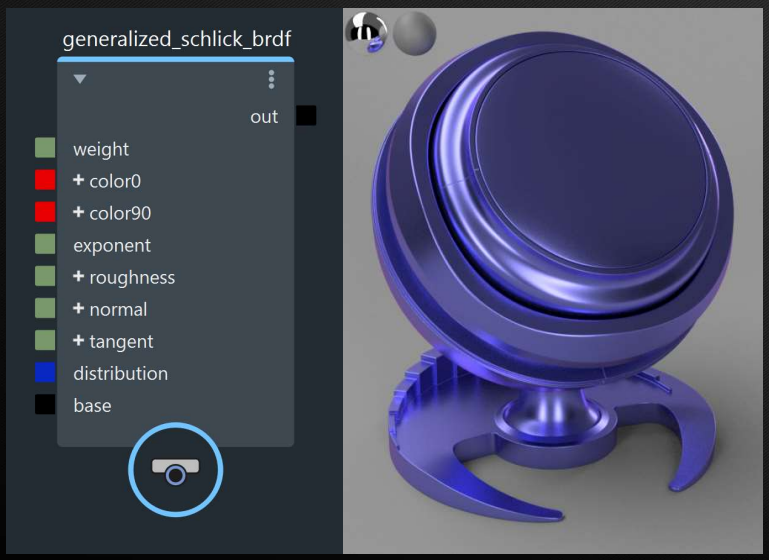


For metals a node with physical conductor Fresnel

PHYSICALLY BASED SHADING NODES

BSDF NODES

- Standard diffuse BSDF nodes
- Standard specular BSDF nodes
- Additional BSDF nodes

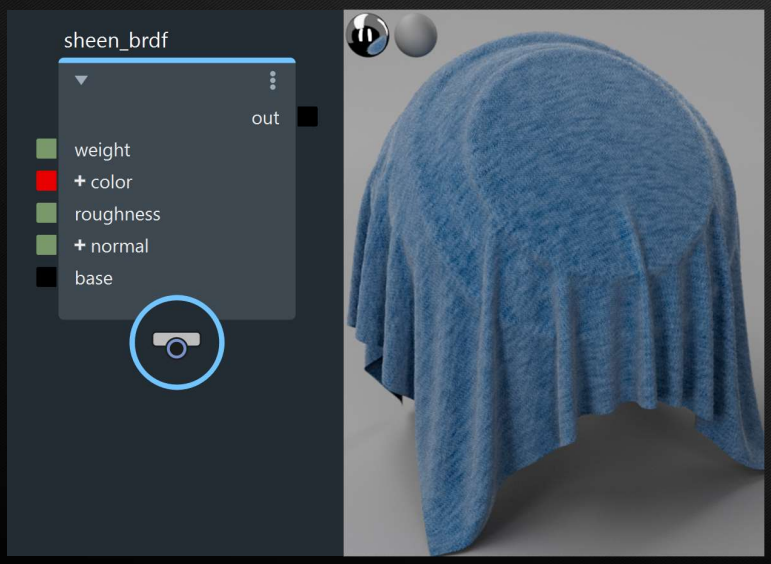


And in addition we have a node using Schlick Fresnel. A general model with more artistic freedom, that can be used for both dielectrics and conductors.

PHYSICALLY BASED SHADING NODES

BSDF NODES

- Standard diffuse BSDF nodes
- Standard specular BSDF nodes
- Additional BSDF nodes

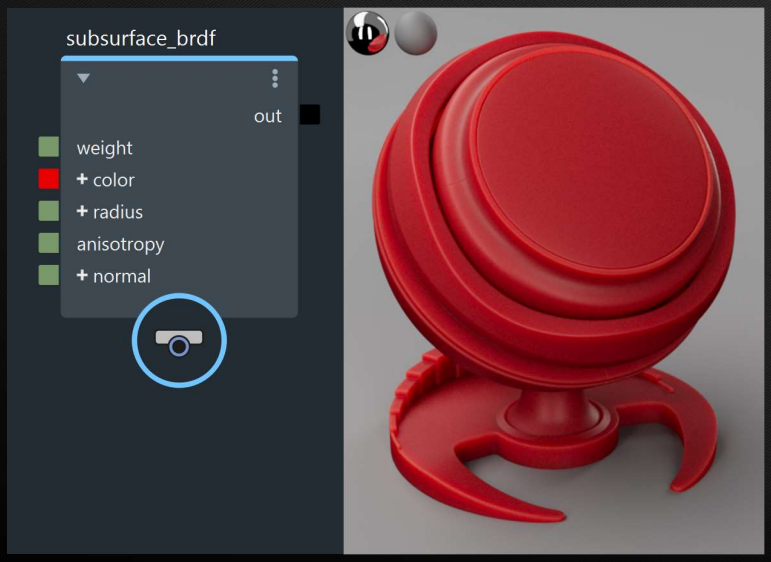


A sheen BRDF - for the back-scattering effects of cloth-like materials

PHYSICALLY BASED SHADING NODES

BSDF NODES

- Standard diffuse BSDF nodes
- Standard specular BSDF nodes
- Additional BSDF nodes

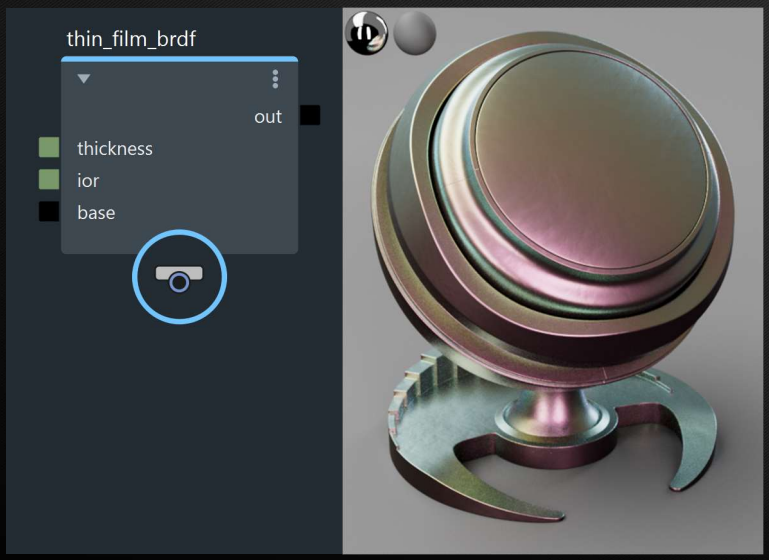


A subsurface BSDF - for the effect of light scattering under the surface.

PHYSICALLY BASED SHADING NODES

BSDF NODES

- Standard diffuse BSDF nodes
- Standard specular BSDF nodes
- Additional BSDF nodes

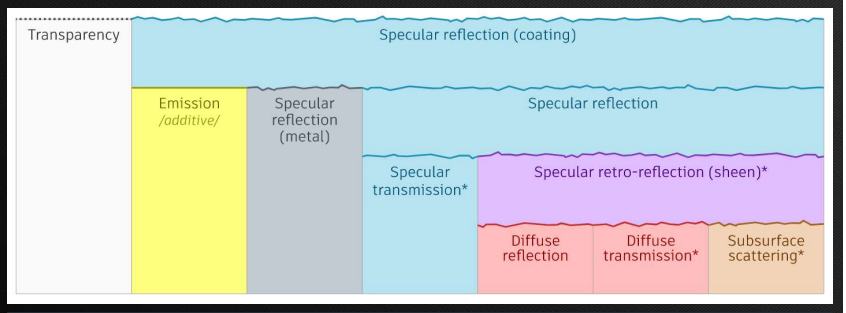


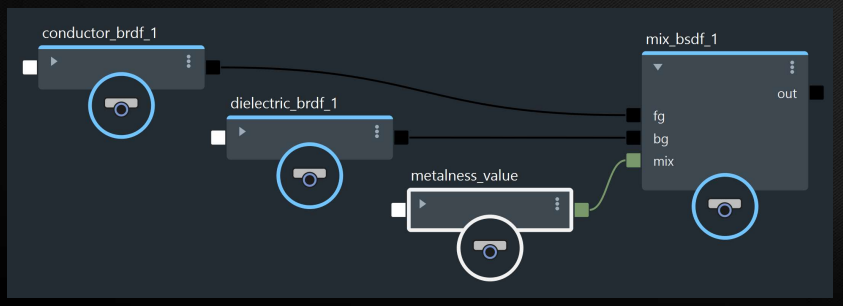
And a thin-film BRDF - for rainbow-like iridescence effects.

PHYSICALLY BASED SHADING NODES

SURFACE LAYERING MODELS

- Horizontal layering (mixing)
- Vertical layering





SIGGRAPH 2020
PHYSICALLY BASED SHADING COURSE

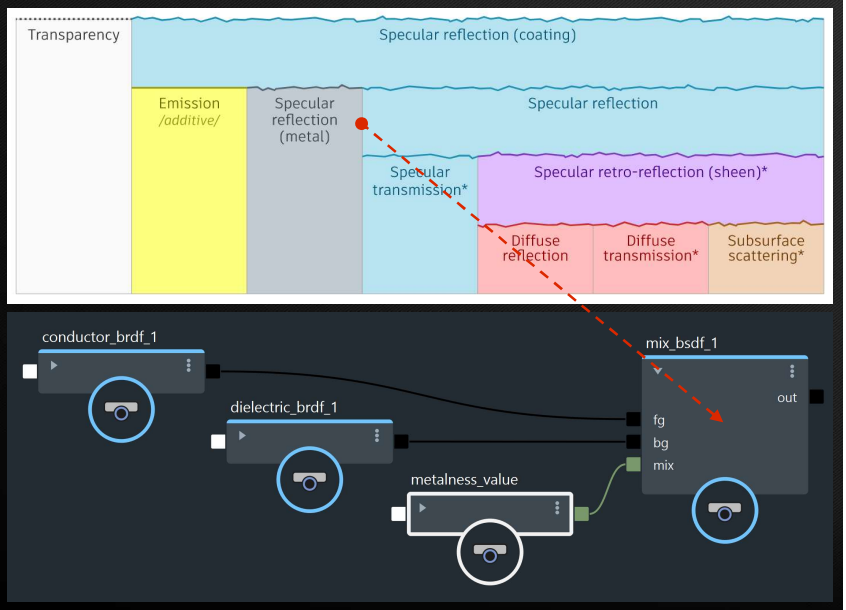
Now in order to construct more complex shading models from these BSDF nodes we need layering, so atomic BSDFs can be combined to form multi-layer materials.

To describe layering in our model, we differentiate between *horizontal layering and vertical layering*.

PHYSICALLY BASED SHADING NODES

SURFACE LAYERING MODELS

- Horizontal layering (mixing)
- Vertical layering



Horizontal layering is a statistical mix of two BSDFs. An example of this is shown here, mixing between a conductor layer and a dielectric-over-diffuse layer, where the mixing weight controls “metalness”, as it’s called in many shading models.

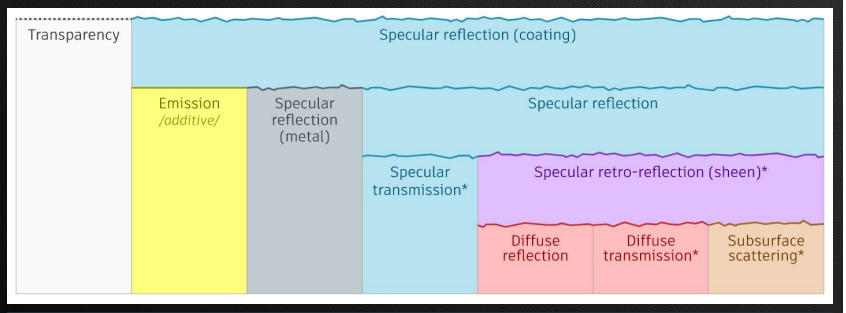
The mixing weights can be textured to produce variation horizontally over the surface, hence the name horizontal layering.

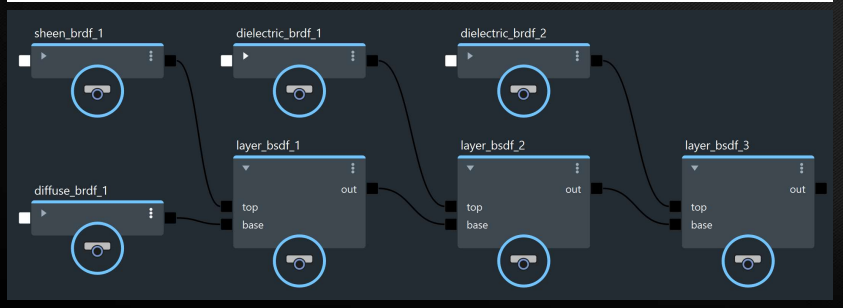
To describe this we have extended the standard MaterialX <mix> node to support the BSDF type, to mix these nodes.

PHYSICALLY BASED SHADING NODES

SURFACE LAYERING MODELS

- Horizontal layering (mixing)
- Vertical layering



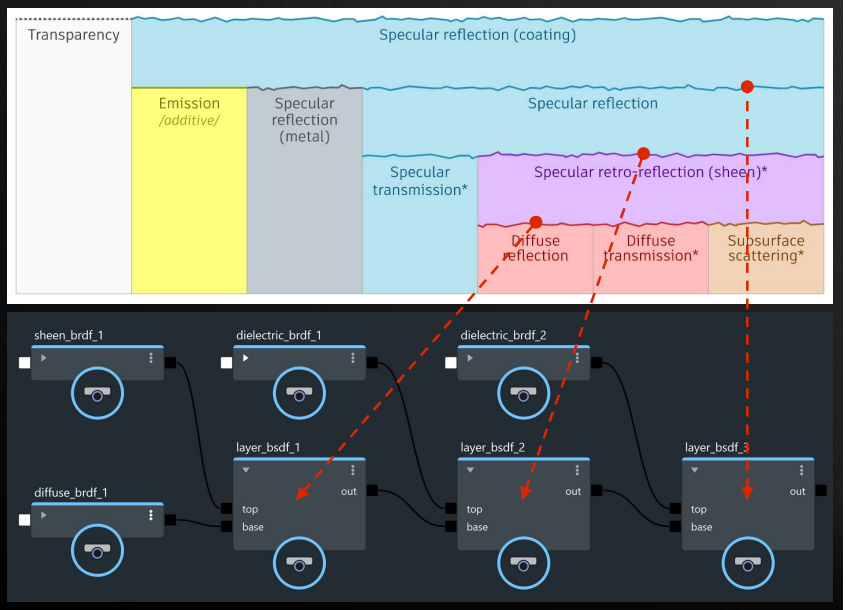


In vertical layering the BSDFs are placed on top of each other, describing a coating over a substrate. The light that is not reflected by the top layer is transmitted down to the layer below. The substrate can be a transmissive BSDF and transmit the light further through the surface, or a reflective BSDF to reflect some of the light back up through the coating.

PHYSICALLY BASED SHADING NODES

SURFACE LAYERING MODELS

- Horizontal layering (mixing)
- Vertical layering



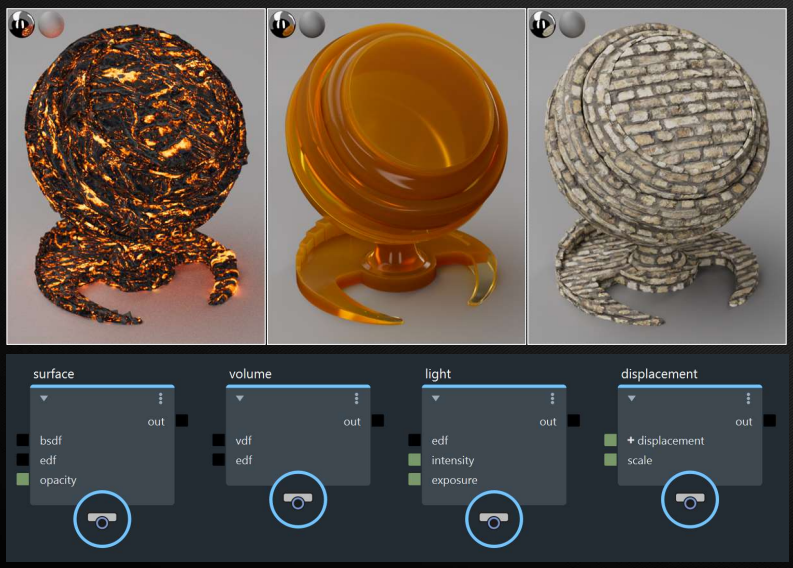
To describe vertical layering we introduce a <layer> node that connects the top and base BSDFs and outputs the new layered BSDF. These nodes can be nested to describe many vertical layers, as in the example here where a diffuse lobe is placed under a sheen lobe, which in turn is placed under a primary and a secondary dielectric coating.

For the case where a transmission BSDF is placed at the bottom of the stack, absorption and scattering inside the media can be controlled by a volume distribution function (VDF) node, that is optionally connected to the transmission BSDF node's "interior" input.

PHYSICALLY BASED SHADING NODES

ADDITIONAL NODE TYPES

- EDF nodes
- VDF nodes
- Shader nodes
- Material nodes



SIGGRAPH 2020
PHYSICALLY BASED SHADING COURSE

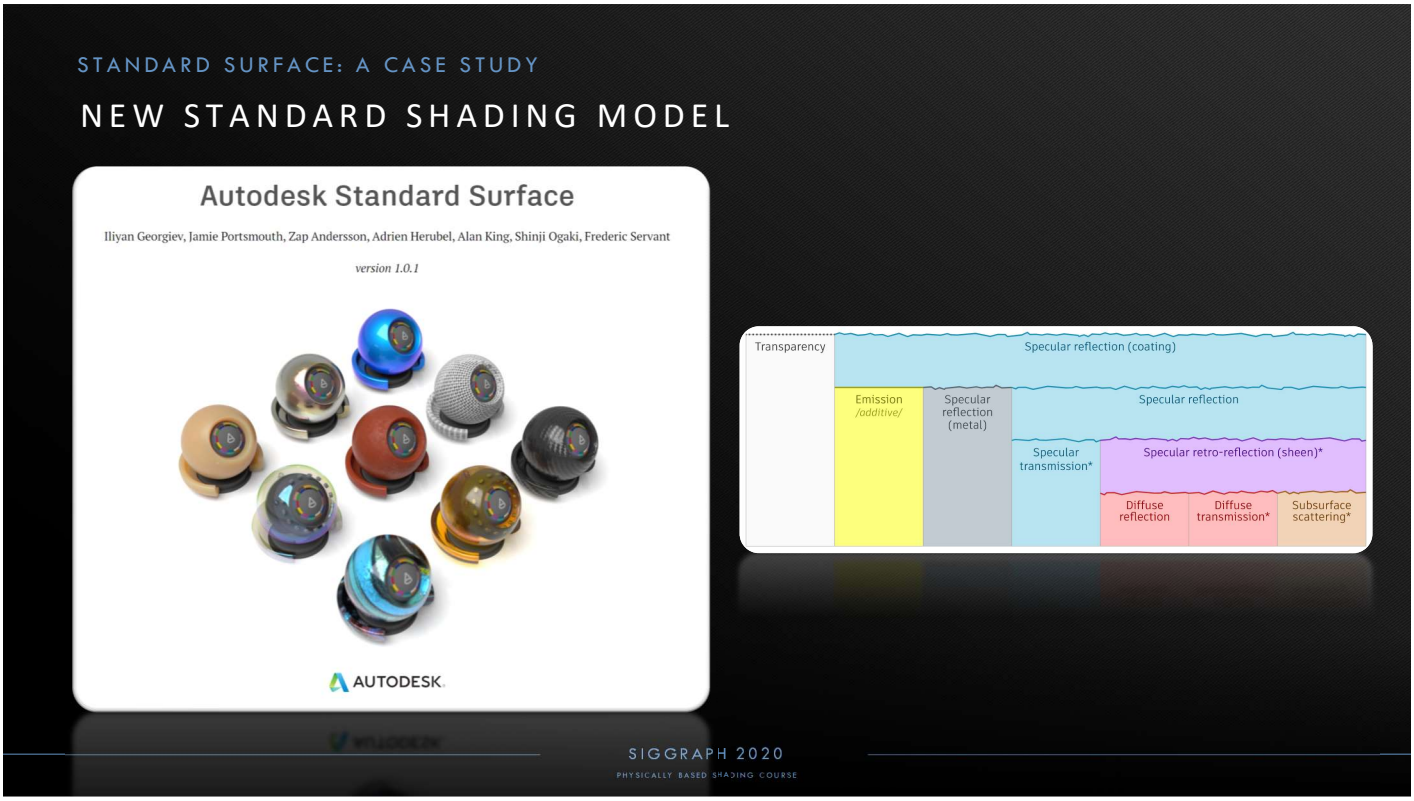
In addition to BSDF nodes we have nodes to define the various EDFs and VDFs. These are out of scope for this talk, but an example of emission as well as volumetric scattering can be seen in the left and middle images here.

Using BSDF, EDF and VDF nodes, together with layering operators, we can now construct shaders by connecting these networks to shader constructor nodes. There are constructor nodes for surface shaders, volume shaders, light shaders and displacement shaders. The bottom image show these nodes and which data they need to be constructed.

Finally, shader nodes are connected to root material nodes to define a material.

The top-left and top-right image show materials with both surface shaders and displacement shaders being used.

The next section is a case study of how these nodes can be used to construct an industry production ubershader.



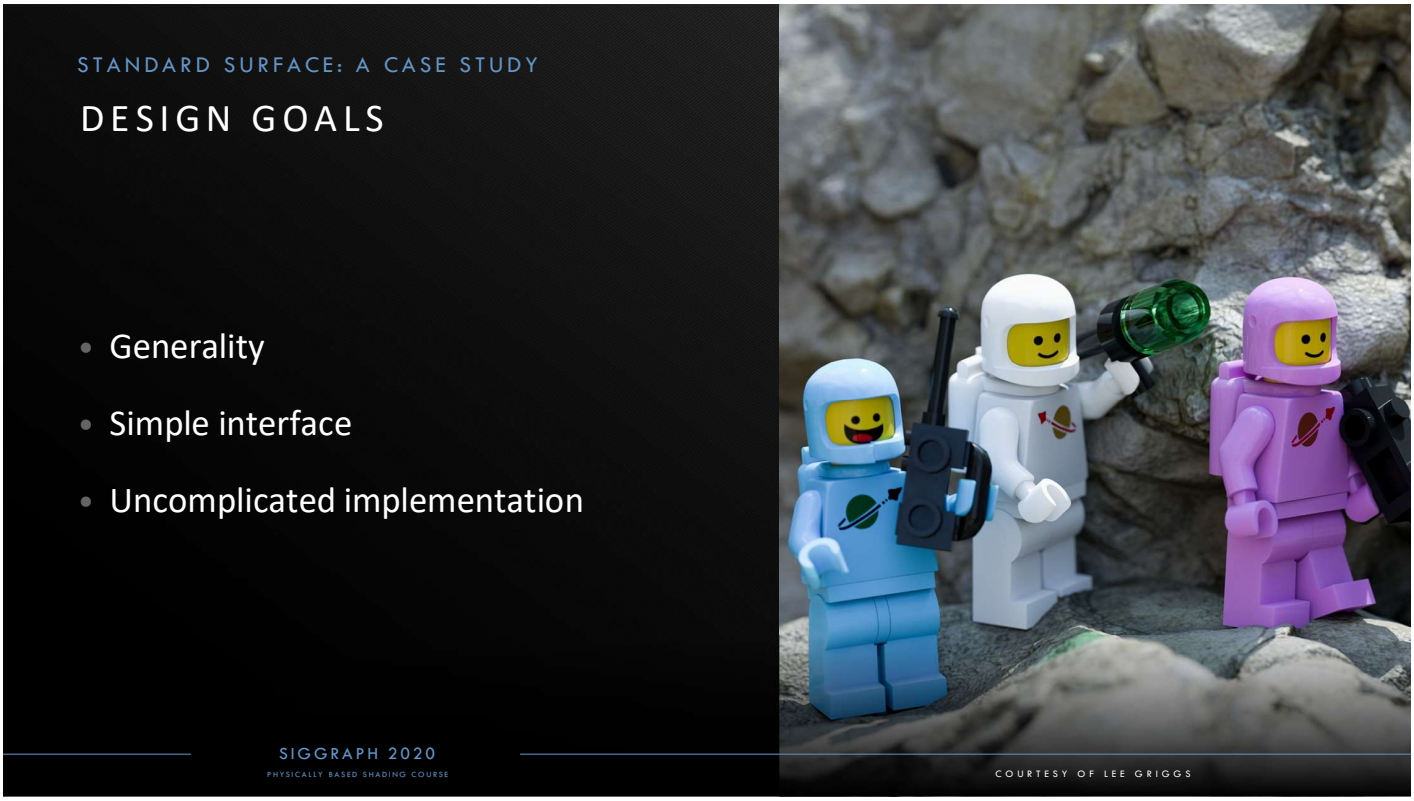
In parallel to the development of the Physically Based Shading Nodes, at Autodesk we were in the process of developing an open standard “uber” shading model for offline and real-time rendering.

This provided us with a valuable proving ground for the new MaterialX shading node set.

The layering example we showed before is from Standard Surface, showing the set of BSDFs used and their layering configuration.

Whitepaper: https://autodesk.github.io/standard-surface/

Reference: https://github.com/Autodesk/standard-surface/tree/master/reference



The design goals set for this shading model were:

- To keep it general enough to be able to model the vast majority of materials used in visual effects and feature animation productions.

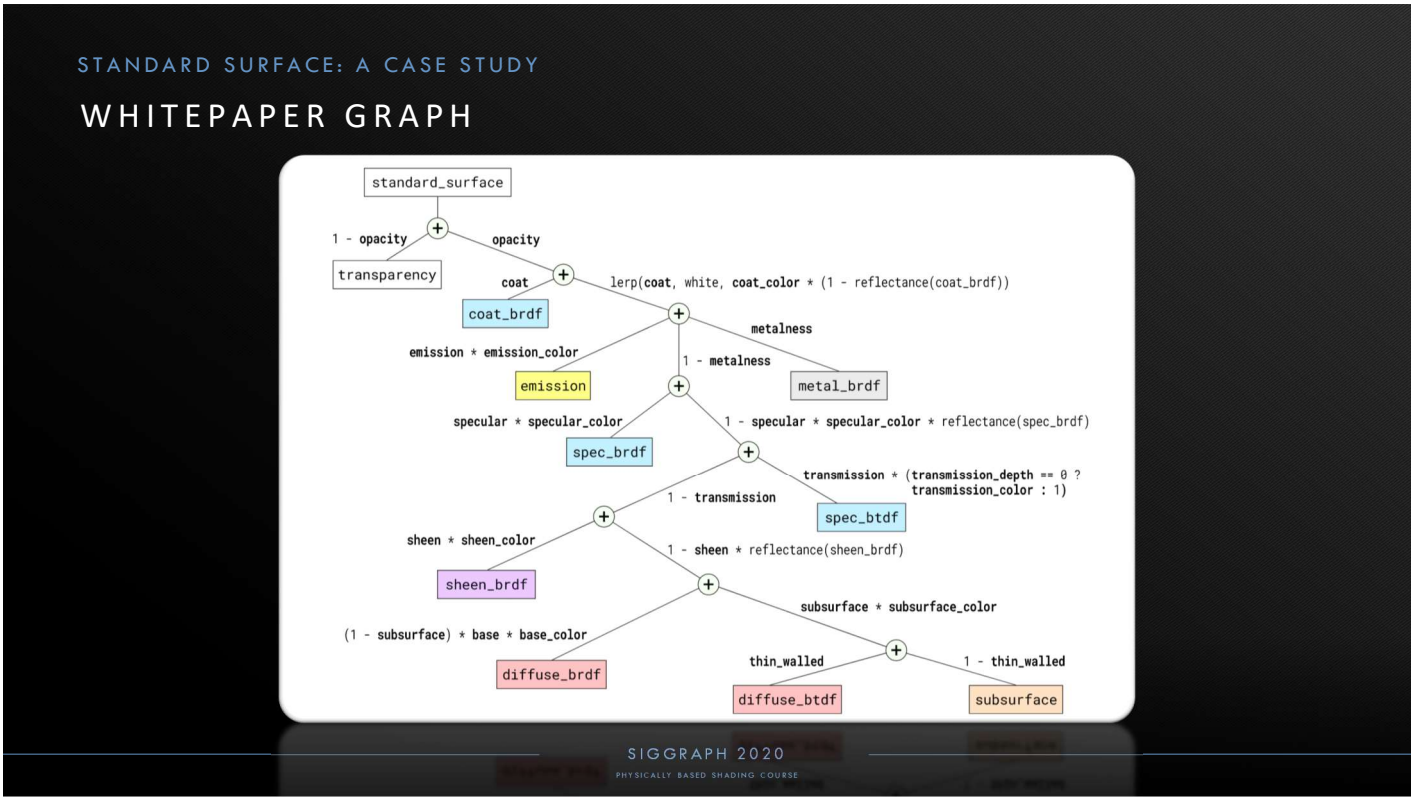
- To reduce the set of parameters to only those that are most useful in practice. The resulting parameters should also have intuitive meanings and ranges.

- To have a relatively uncomplicated implementation, that scales to many different architectures.

The aim was for the behavior to be simple, intuitive, and understandable, so the model can cover most day-to-day use cases. For the few it does not cover, a specific shading network or renderer-specific shader can be used instead.

STANDARD SURFACE: A CASE STUDY

WHITEPAPER GRAPH



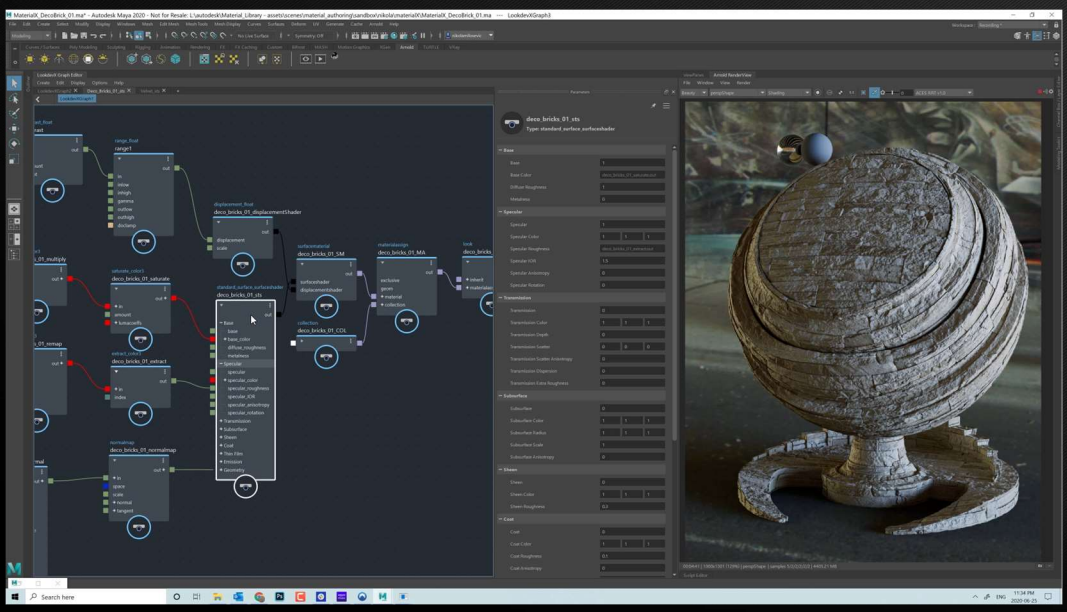
Standard Surface was defined as a graph from its inception, so this fits very well with our model.

Here we show the layering configuration in graph form, where each edge in the graph gives the weighting to use for layers below it.

So this is what we want to describe with our MaterialX nodes.

STANDARD SURFACE: A CASE STUDY

MATERIALX GRAPH



SIGGRAPH 2020

PHYSICALLY BASED SHADING COURSE

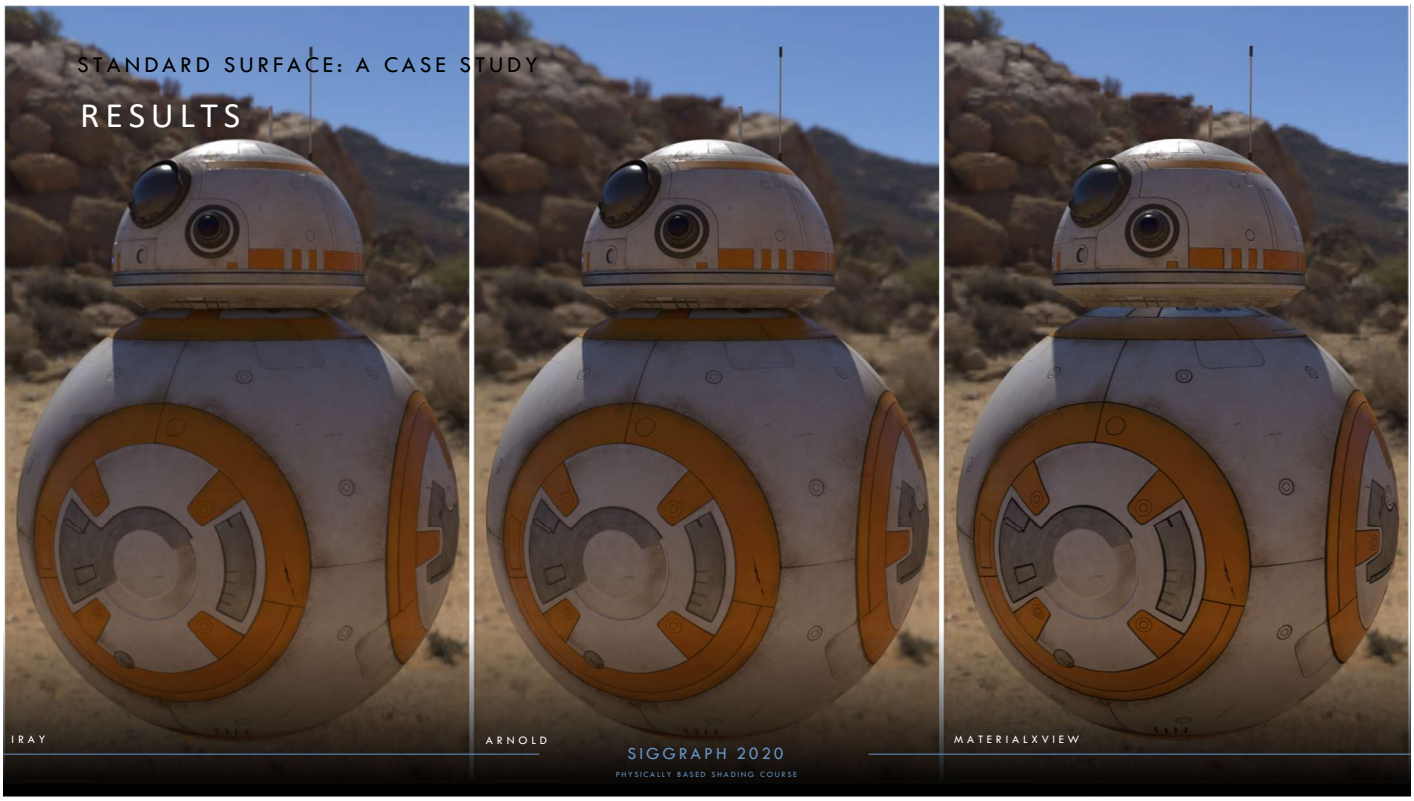
And here we will look at the resulting graph built from the MaterialX Physically Based Shading nodes.

- This is an instance of standard surface, and we can look at the implementation graph.
- Here we see how the various layers are formed and connected in a layer stack.
- Diffuse & subsurface scattering, layered with sheen, then mixed with transmission, and over this a primary specular + thin film, then mix with a conductor for metalness, and finally a secondary specular coating.
- This we feed into the shader constructor.
- Which also takes and EDF input for emission, and an input for cutout opacity.

This graph is visualized in an upcoming Autodesk project titled LookdevX, which is a native graph editor for MaterialX content. It's expected to be one of the first public tools that will support the full set of BSDF nodes in MaterialX, allowing the user to navigate, edit, and create BSDF graphs, with the resulting shaders being generated on demand in GLSL and OSL.

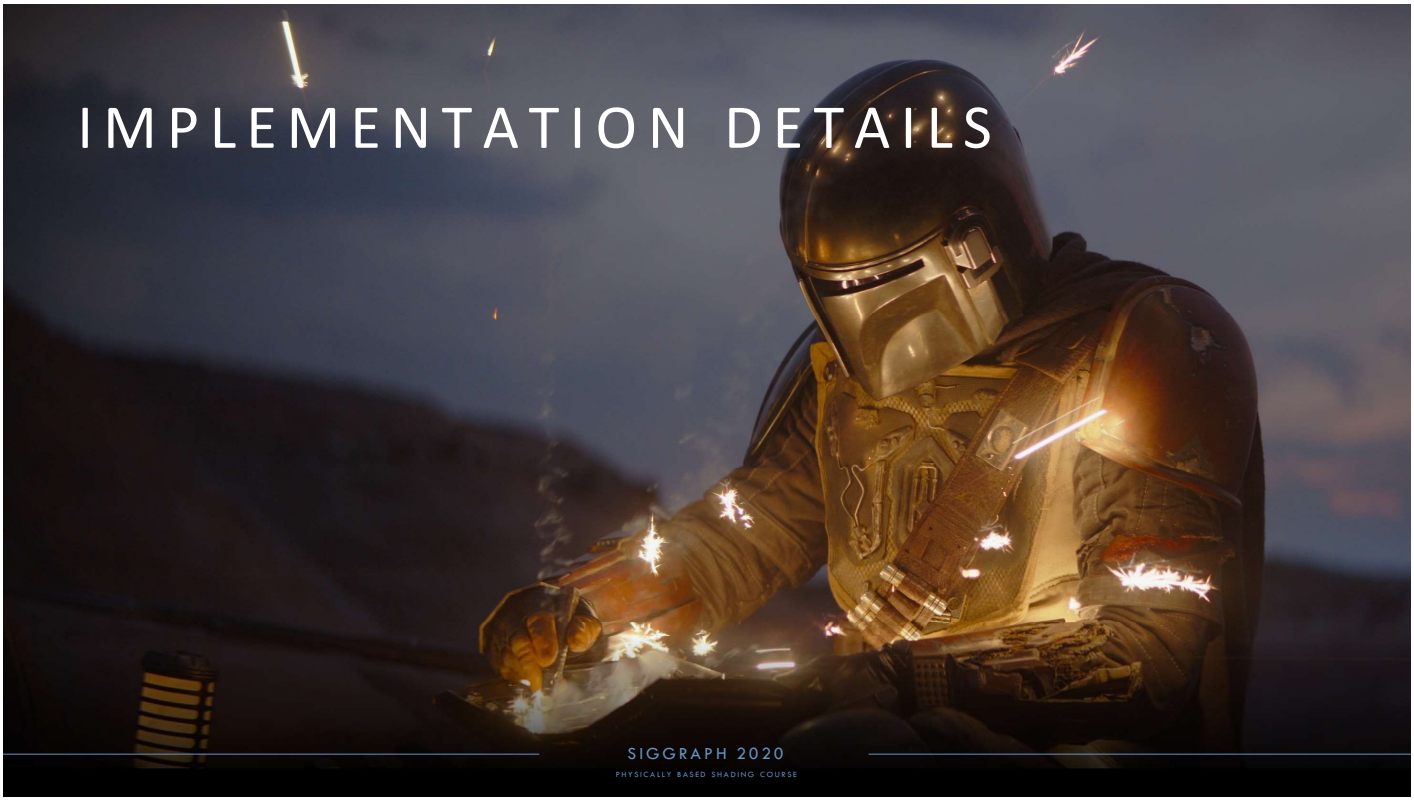
Once the shading model is described in MaterialX form, we can render it in different renderers (we will show how in the Implementation Details section).

Here showing examples rendered in Arnold (top), and in Iray (bottom).



And here’s another example of an asset built with Standard Surface, rendered consistently across three different systems, including OpenGL rasterization.

From left to right: Iray, Arnold and MaterialXView.

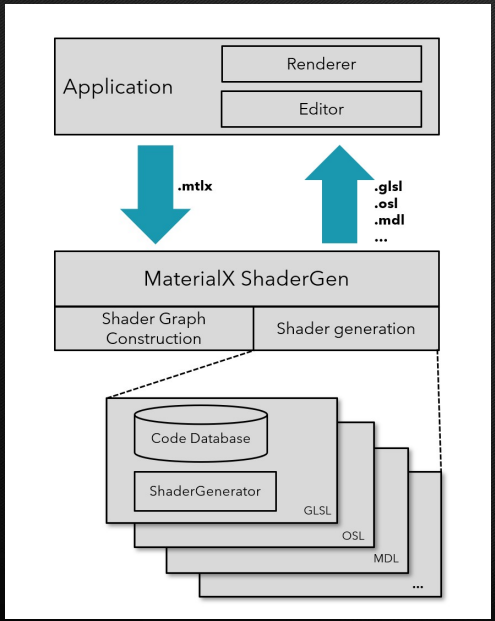


In this next section, we will look at more details how this is implemented.

IMPLEMENTATION DETAILS

MATERIALX CODE GENERATION

- Translation to multiple shading languages
- Extensible to new languages
- Extensible to new target renderers
- See MaterialX developer guide for details

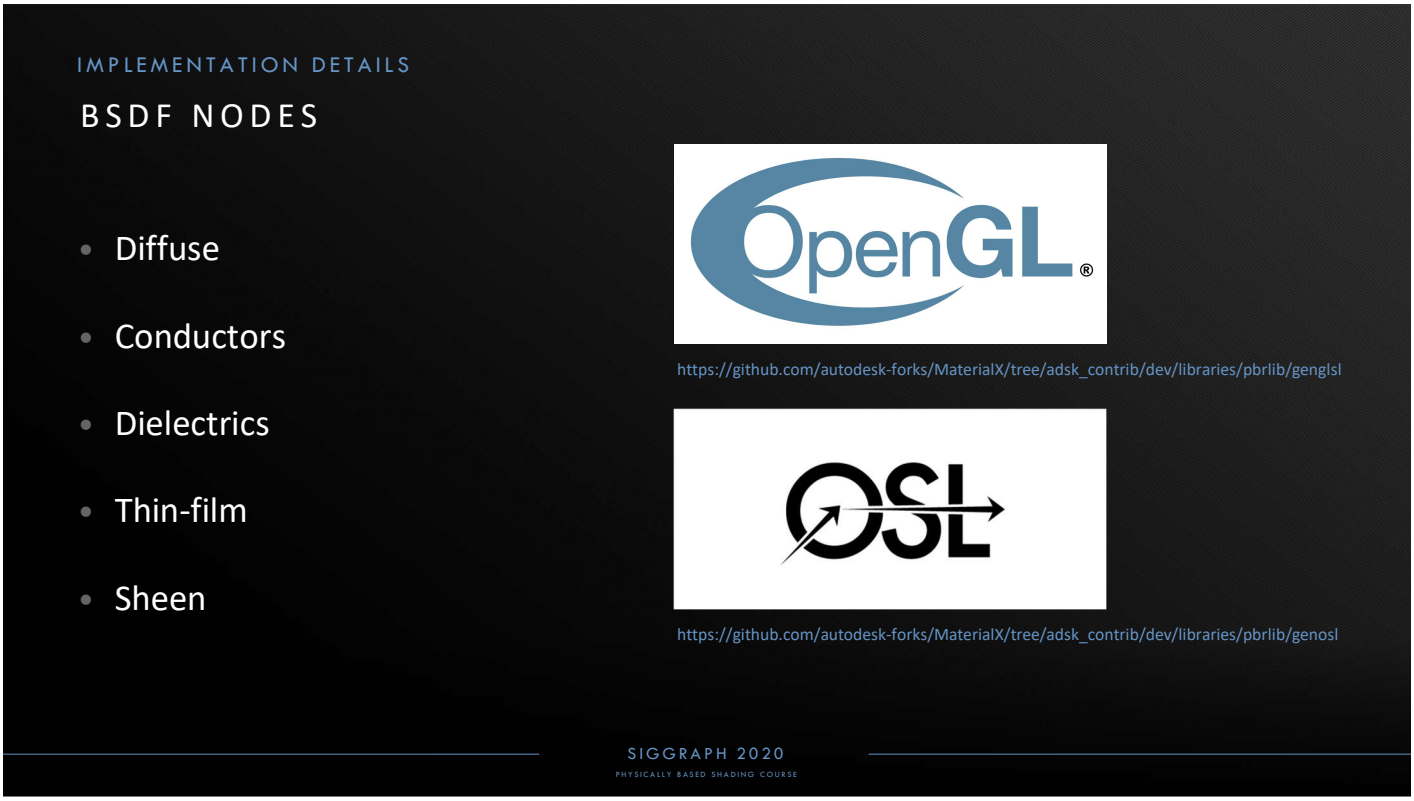


In order to render these graphs on different architectures, we use MaterialX code generation. So the graph is translated to shader code for different target renderers.

By supporting open standard shading languages like OSL and MDL, we hope to ease the adoption of MaterialX shader graphs. If you have a renderer using such languages, you can take advantage of this right away. To support the Physically Based Shading nodes and all materials constructed this way, you just need to have closures that can represent these BSDFs. Out of the box you also get full support for the MaterialX standard library, with all of its math and texturing nodes.

The system is extendable to new shading languages, or new targets for already supported languages, like different OpenGL viewports or different OSL renderers.

More details on the code generation can be found in the MaterialX developer guide docs and full source code is available:
https://github.com/materialx/MaterialX/blob/master/documents/DeveloperGuide/ShaderGeneration.md
https://github.com/materialx/MaterialX/tree/master/source



Now a look at some of the BSDF nodes and how they are implemented. Both in the context of real-time rendering and offline rendering.

Source code for our reference implementations are available at these links:
https://github.com/autodesk-forks/MaterialX/tree/adsk_contrib/dev/libraries/pbrlib/genglsl

https://github.com/autodesk-forks/MaterialX/tree/adsk_contrib/dev/libraries/pbrlib/genosl

https://github.com/autodesk-forks/OpenShadingLanguage/tree/adsk-contrib/dev

IMPLEMENTATION DETAILS

DIFFUSE

`diffuse_brdf`

- Oren-Nayar BRDF

`burley_diffuse_brdf`

- Disney Principled BRDF

SIGGRAPH 2020
PHYSICALLY BASED SHADING COURSE

Our standard diffuse reflection node is a standard implementation of Oren-Nayar, and the Burley diffuse reflection node closely follows the Disney paper from 2012.

All wedge renderings, unless explicitly stated, show the visual difference with a varying roughness value 0.0 - 1.0.

There is a clear visual difference between the Oren-Nayar and Burley BRDFs at higher roughness values, and we believe it's important to provide both diffuse nodes in MaterialX, in order to accurately preserve this visual distinction in the shading models that are based on them. Materials based on the Disney Principled or PxrSurface shading model, for example, would noticeably change their character if forced to use Oren-Nayar or Lambert diffuse instead of the Burley diffuse for which they were originally authored.

Oren-Nayar:
https://www1.cs.columbia.edu/CAVE/publications/pdfs/Oren_SIGGRAPH94.pdf
https://mimosa-pudica.net/improved-oren-nayar.html

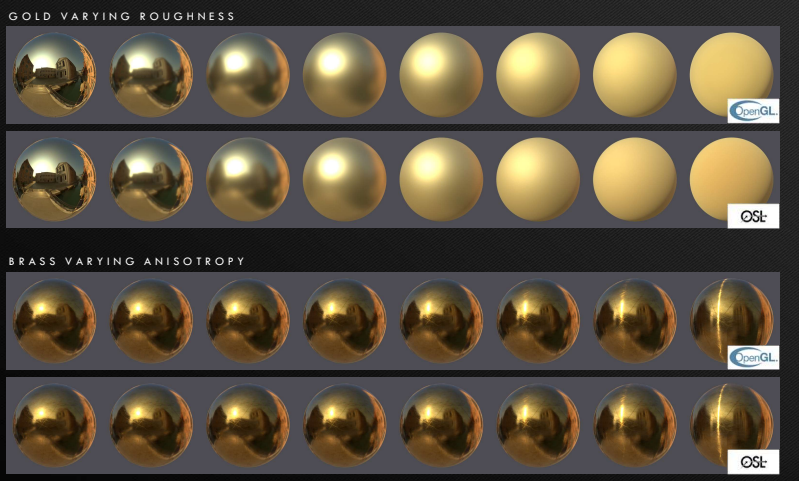
Burley 2012:
https://blog.selfshadow.com/publications/s2012-shading-course/burley/s2012_pbs_disney_brdf_notes_v3.pdf

IMPLEMENTATION DETAILS

CONDUCTORS

`conductor_brdf`

- Microfacet BRDF
  - D = GGX | Beckmann
  - G = Height-correlated Smith
  - F = Conductor
- Artist Friendly Metallic Fresnel



Our conductor BRDF is based on a standard microfacet model, with a selection of GGX or Beckmann as the microfacet distribution.

We use height-correlated Smith for the shadowing term, and physical Fresnel for conductors with complex refraction index. However we use the artistic reparameterization of this, from Gulbrandsen, to make it easier to control. If there is a need to enter a complex refraction index, we have a conversion node that can be connected upstream.

Our GLSL implementation supports roughness and anisotropic reflections both for punctual and environment lights, as can be seen here. We will share more details on this later.

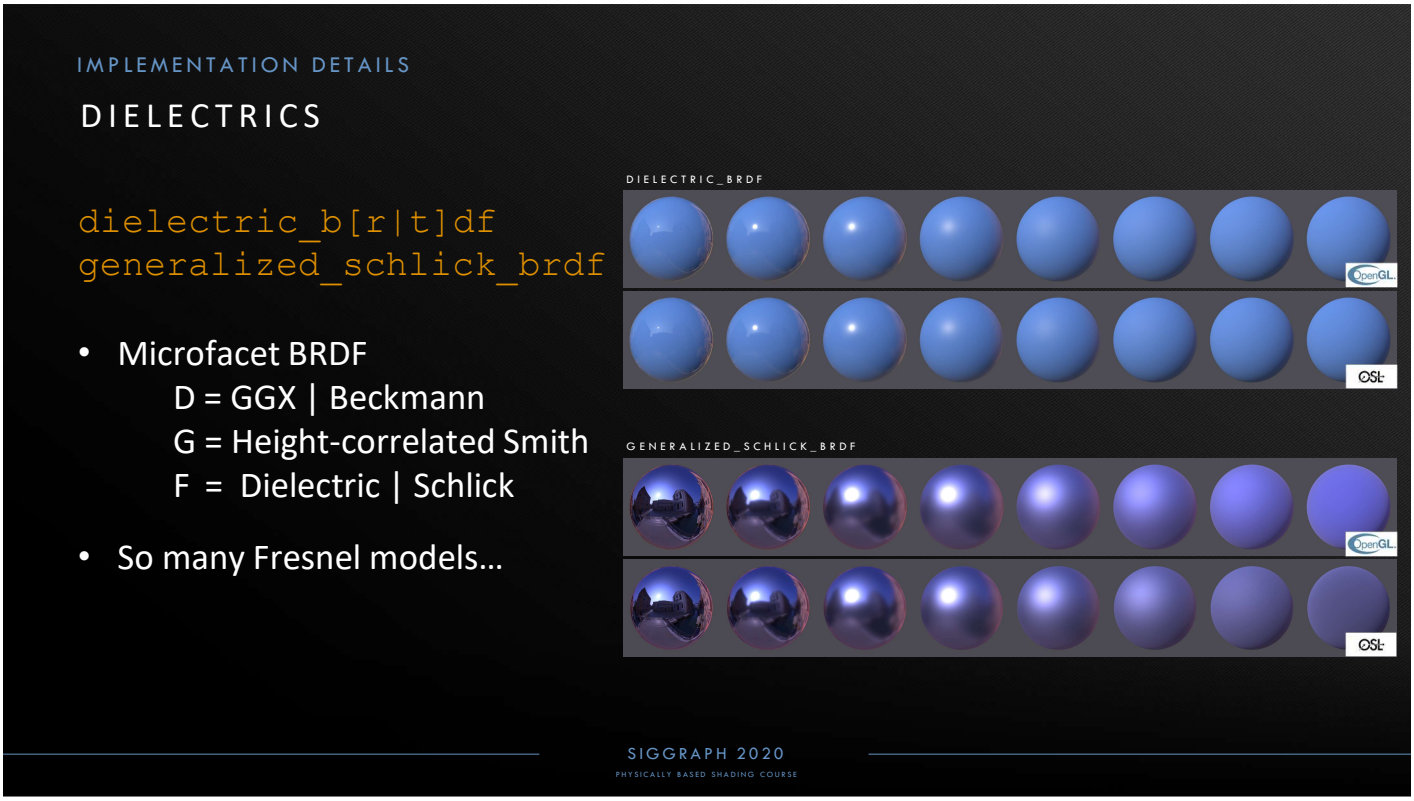
Gulbrandsen 2014:
http://jcgt.org/published/0003/04/03/paper.pdf

To compensate for energy missing from multiple scattering on rough surfaces, we use the method from Turquin. It's simple enough to be used both for offline and real-time rendering. For OSL, if the target renderer doesn't have this built into the closure already, we can use this method in shader code. For adding this inside a closure, one can use Turquin's method or the method from Conty and Kulla where an extra lobe is added to compensate for energy loss.

Wedges here show gold rendered with energy compensation using Turquin's method (top), with energy compensation using Conty and Kulla's method (middle), and with compensation disabled (bottom).

Turquin 2019:
https://blog.selfshadow.com/publications/turquin/ms_comp_final.pdf

Conty and Kulla 2017:
https://blog.selfshadow.com/publications/s2017-shading-course/imageworks/s2017_pbs_imageworks_slides.pdf



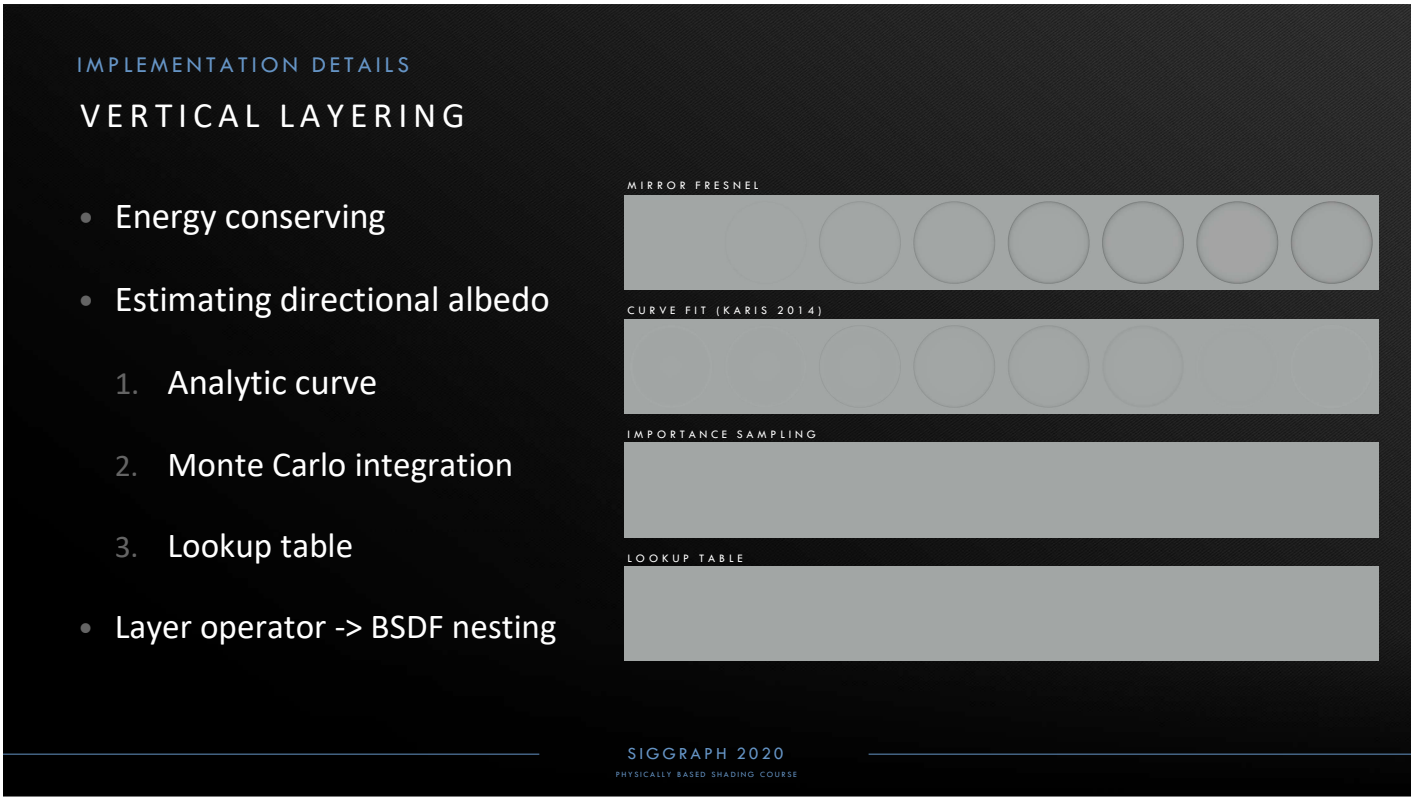
For our dielectric nodes, we use the same microfacet model, only with different Fresnel terms.

Either a physical dielectric Fresnel with an explicit index of refraction, or Schlick Fresnel which takes colors for incident and grazing reflectance, along with an optional exponent to influence the shape of the Fresnel curve.

Including the conductor Fresnel from before, we now have three different Fresnel models. It's not ideal to have three different nodes to handle specular reflections, but it's an intentional choice in MaterialX, since there are popular shading models that rely upon each of these approaches for accurate rendering. But we are closely following the research in this area, for example, as shown in the talk just before this. Having a single Fresnel model that can cover all needs would simplify our library.

For dielectrics we use the same method for multiple scattering compensation as for the conductor node.

(In the bottom image, the offline render version was done in Renderman, using an implementation of PxrSurface that preceded the introduction of energy compensation. So that is why the results at high roughness values are not matching with our GLSL version that includes energy compensation).



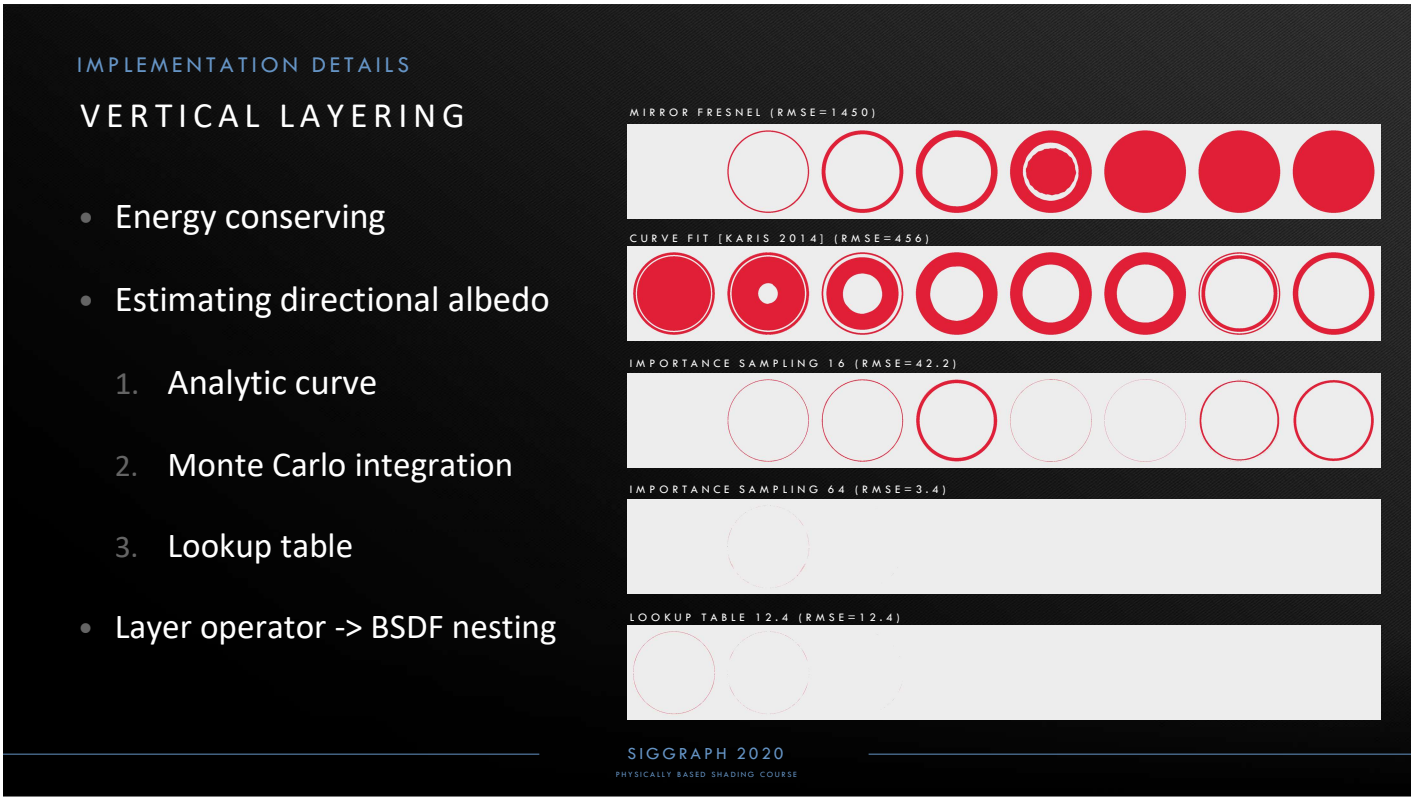
Being dielectrics, these nodes are layerable and can be placed as a coating on top of other BSDF nodes using the layer operator. We implement vertical layering by albedo scaling, where we estimate the top layer's directional albedo and scale the response from layers below by the inverse of this.

We support three different methods for estimating the albedo:
- Analytic curves, if we have a curve fit to approximate the albedo
- Monte Carlo integration on the fly at run time
- Lookup tables, precalculated using Monte Carlo integration

The wedges here show a furnace test comparison of these methods, with a dielectric GGX specular layered over a diffuse. The top image show results when using a single mirror Fresnel to approximate the reflectance. As expected, it only works for a very low roughness. With a curve fit to the directional albedo we can do better, improving the results a lot for high roughness values. The curve we use here is from Karis 2014. As future work we plan on finding a new curve with a better fit over the whole range.

Using Monte Carlo integration and importance sampling of the albedo, here with 64 samples, we get almost perfect results. But by precalculating this to a LUT we can get away with a similar result and avoid the runtime sampling. This is what we use by default in GLSL. For OSL we use Monte Carlo importance sampling at run time.

Karis 2014:
https://www.unrealengine.com/en-US/blog/physically-based-shading-on-mobile



Here's a comparison of the errors introduced to show this more clearly. (Red pixels show *any* deviation above a threshold, so the curve fit method is better than it looks, if we compare the error numbers instead).

With our code generator you can choose which method to use, depending on your accuracy vs. performance requirements.

It's worth noting that our layering description does not impose constraints on how to perform the vertical layering. Our code generator transforms the layer operator setup into a nesting of BSDF nodes, where each BSDF could have different strategies for how to implement its layering. So if better methods for layering are found, this can be implemented underneath the same description.

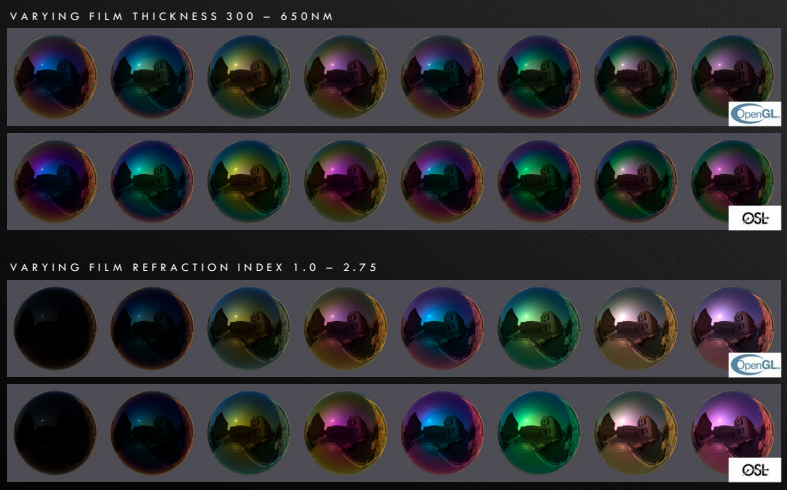
We also imagine that properties of the layering medium, like thickness and scattering coefficients, could be added as parameters on the layer node if needed in the future.

IMPLEMENTATION DETAILS

THIN-FILM

`thin_film_brdf`

- Modeling of Varying Iridescence
- Layerable over all microfacet BSDFs
- Implemented as a new Fresnel model



SIGGRAPH 2020
PHYSICALLY BASED SHADING COURSE

To render iridescence effects, we use the method from Belcour and Barla 2017. This works well for both real-time and offline rendering.

For users, this is also described as vertical layering, adding a thin-film BSDF node over another BSDF. But it's implemented differently. When such configurations are found in the graph, we switch the Fresnel term on the base BSDF to a model that takes iridescence into account (Airy reflectance).

This is supported on all microfacet nodes (dielectrics and conductors).

Belcour and Barla 2017:
https://belcour.github.io/blog/research/2017/05/01/brdf-thin-film.html

IMPLEMENTATION DETAILS

SHEEN

`sheen_brdf`

- Microfacet Sheen BRDF
- Layerable on any BSDF
- Using albedo scaling (LUT)
- Simplified for GLSL

SIGGRAPH 2020
PHYSICALLY BASED SHADING COURSE

Our sheen BRDF is based on the work by Imageworks [Conty and Kulla 2017].

This node is also layerable, as a specular back-scattering response on any other BSDF.

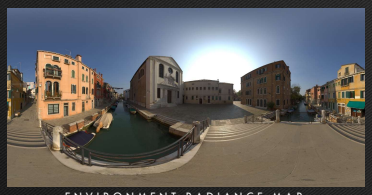
As with dielectrics, this is done with albedo scaling, here using a lookup table, both for OSL and GLSL. For GLSL, we have additional simplifications: ignoring the geometry term and use a smoother denominator in the microfacet expression [Neubelt and Pettineo 2013]. But the same NDF (Charlie Sheen) from Imageworks is used.

Conty and Kulla 2017:
http://www.aconty.com/pdf/s2017_pbs_imageworks_sheen.pdf

Neubelt and Pettineo 2013:
https://blog.selfshadow.com/publications/s2013-shading-course/rad/s2013_pbs_rad_notes.pdf

IMPLEMENTATION DETAILS

ENVIRONMENT LIGHTS IN GLSL



ENVIRONMENT RADIANCE MAP

- Main input is a latitude-longitude radiance map
- Ideal is to support all BSDF nodes with minimal precomputation

CHROME (STANDARD SURFACE)

Earlier we noted that environment lights require special consideration in real-time shading languages like GLSL, and in this section we'll provide some additional details on how they're handled during code generation.

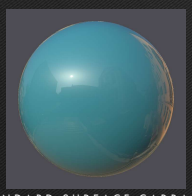
The main input to our environment lighting system is a latitude-longitude radiance map, with each texel representing the incoming radiance from the corresponding solid angle of the distant environment.

For our default GLSL code generator, which is used in real-time visualization tools such as MaterialXView and LookdevX, we would ideally like this radiance map to be sufficient in rendering any shading model required at run time, with a minimum of precomputed data for the variety of BSDF nodes that MaterialX supports.

IMPLEMENTATION DETAILS

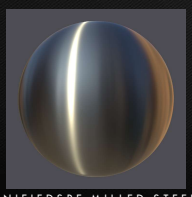
FILTERED IMPORTANCE SAMPLING

- Maintains independence from details of specular model
- User sample count controls tradeoff between fidelity and performance
- Extends to anisotropic surfaces with high sample counts



STANDARD SURFACE CARPAINT

UNIFIEDSRF BRASS



UNIFIEDSRF MILLED STEEL

For rendering specular nodes such as dielectric, conductor, and generalized Schlick, we've focused on a popular technique called *Filtered Importance Sampling*. Based on the 2008 paper by Křivánek and Colbert, this method integrates incoming lighting across the NDF of the surface by sampling the radiance map multiple times, using the mipmap chain of this texture to appropriately filter each lighting sample.

The main advantage of this approach is its independence from microfacet terms, as the lighting integral works for any NDF that can be importance sampled and can accommodate any Fresnel or geometric term that is required by a BSDF node.

The need to sample the environment light multiple times makes it more expensive than prefiltered approaches, but this can be mitigated by exposing a sample count to the user, giving them control over the tradeoff between visual fidelity and performance. In the MaterialX viewer, for example, we sample environment lights 16 times by default, but give the user a range from 4 to 1024 samples at run time, with the highest settings being useful when validating real-time renders against offline techniques.

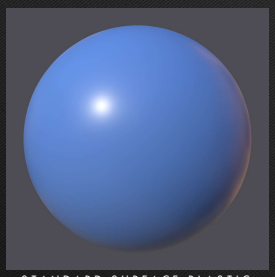
As seen in the milled steel example on this slide, it's also possible to approximate anisotropic surfaces with sufficiently high sample counts, with the results comparing favorably with offline renders.

Křivánek and Colbert 2008:
https://cgg.mff.cuni.cz/~jaroslav/papers/2008-egsr-fis/2008-egsr-fis-final-embedded.pdf

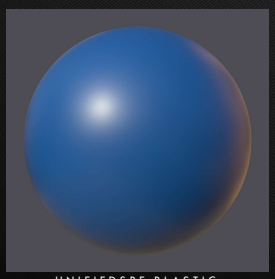
IMPLEMENTATION DETAILS

PREFILTERED IRRADIANCE MAPS

- Approximation is performant without a significant loss of fidelity
- Non-Lambertian diffuse lobes modulate by the directional albedo
- Runtime filtering supported by a C++ spherical harmonics library



STANDARD SURFACE PLASTIC

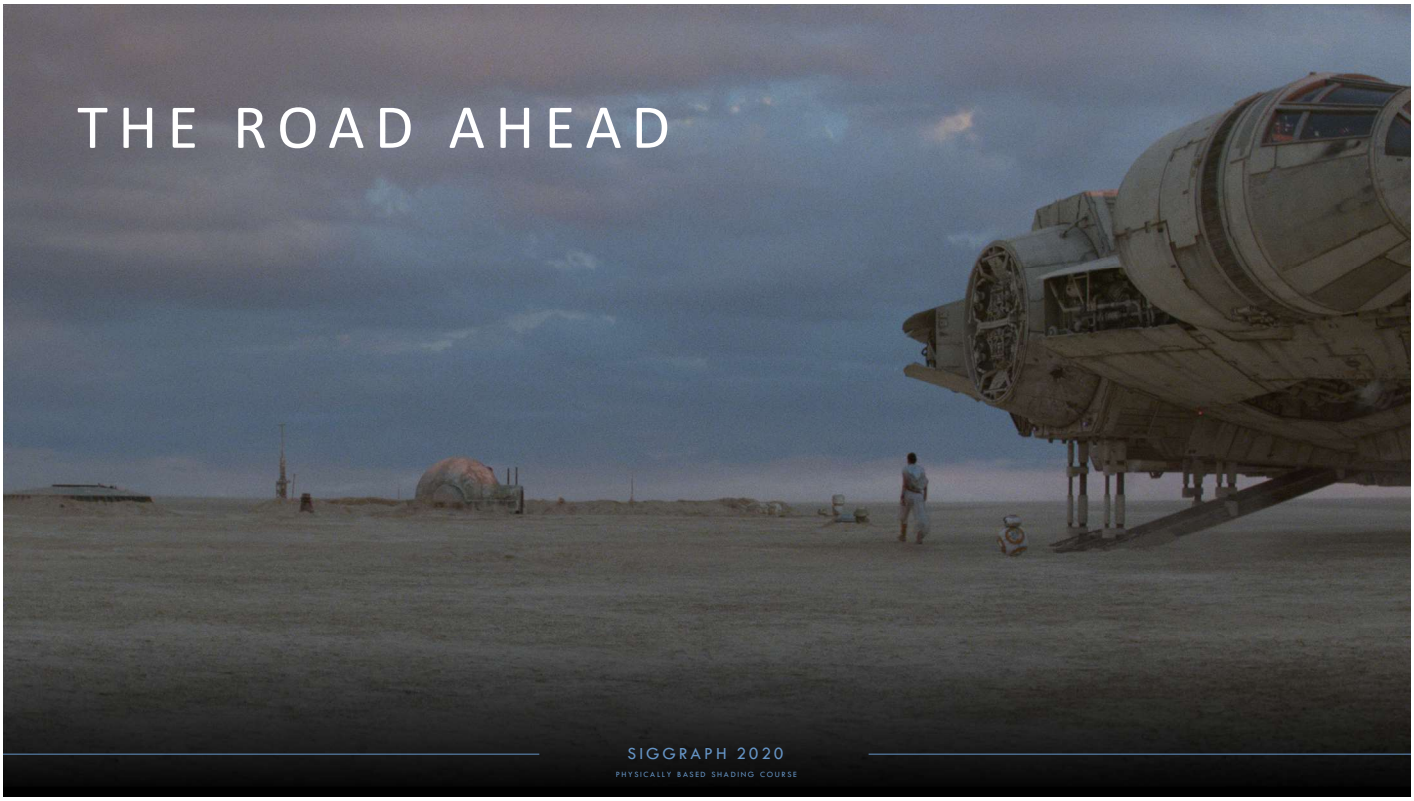


UNIFIEDSRF PLASTIC

For rendering diffuse nodes, while it's technically possible to use Filtered Importance Sampling with very high sample counts, we instead use the popular approximation of *Prefiltered Irradiance Maps*, which gives significantly better performance with only a small degree of approximation error.

Since prefiltered irradiance maps are only valid for Lambertian diffuse reflections, we approximate reflections from other diffuse models by modulating the prefiltered lookup by the directional albedo of the diffuse lobe for the given view direction. This approach is used, for example, in Lucasfilm's UnifiedSrf shader, which leverages the Burley diffuse node in MaterialX. We use the same method for our sheen lobe.

For user-specified environment lights that lack a prefiltered irradiance map, we provide a spherical harmonics library in C++ to generate them efficiently on demand. The spherical harmonics library also supports the separation of environment lights into their directional and ambient components, allowing run-time rendering of shadows from novel lighting environments. Examples of this usage can be found in the MaterialXView application within the MaterialX codebase on GitHub.



In this section, we'll provide a few thoughts on the road ahead, and discuss some current and upcoming projects that build upon the work we've presented today.

THE ROAD AHEAD

SHADER TRANSLATION GRAPHS

- Translation of material content between shading models
- Expressed as a graph that converts between input signals
- Used throughout this presentation for BB-8 and R2-D2

LUCASFILM UNIFIEDSRF

STANDARD SURFACE

SIGGRAPH 2020

PHYSICALLY BASED SHADING COURSE

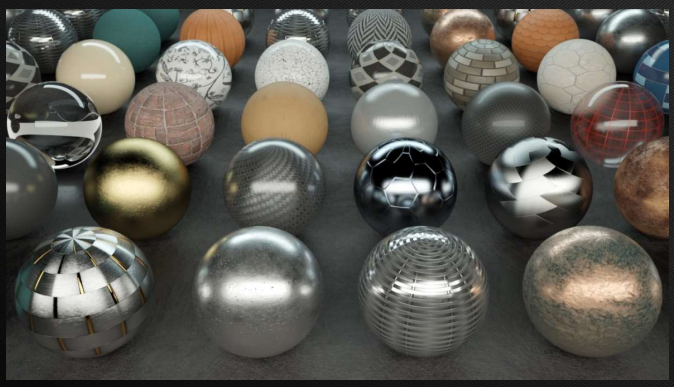
At Lucasfilm, one active area of research is the translation of material content between shading models, with that translation expressed as a MaterialX graph that converts signals between the expected inputs of the two BSDFs. This approach works together with texture baking, allowing the translated material content to be converted to flat textures for efficient rendering.

In the slides we've presented today, we've used this technique to translate BB-8 and R2-D2 from Lucasfilm's production shading model, UnifiedSrf, to the Standard Surface model for transport to external teams. Because both of these shading models can be expressed effectively as a graph of physically based shading nodes, it's straightforward to render them consistently within a single reference application like MaterialXView, where translation graphs between models can then be developed and tested.

THE ROAD AHEAD

ADDITIONAL SHADING LANGUAGES

- MDL (currently on GitHub)
- HLSL
- WebGL



NVIDIA MDL

A second area of research and development is the extension of MaterialX code generation to new shading languages beyond OSL and GLSL.

At the GPU Technology Conference earlier this year, Autodesk and NVIDIA presented a new MaterialX code generator for the Material Definition Language, or MDL, and the code for this project can now be found on GitHub.

THE ROAD AHEAD

ADDITIONAL SHADING LANGUAGES

- MDL (currently on GitHub)
- HLSL
- WebGL

HLSL IN 3DS MAX

Autodesk has also been developing early support for HLSL, and here we're seeing a glimpse of this work in a preview build of 3ds Max, where generated OSL has been cross compiled to HLSL for the real-time viewport.

THE ROAD AHEAD

ADDITIONAL SHADING LANGUAGES

- MDL (currently on GitHub)
- HLSL
- WebGL

WEBGL IN AUTODESK FORGE VIEWER

And in preparation for cloud-based workflows, the Autodesk Forge team has been working on code generation for WebGL, including new JavaScript bindings for the MaterialX API.

A third area that we're following closely is the development of *standardized material libraries* based on portable shading models expressed as BSDF graphs.

At Lucasfilm, we've long been developing a *Common Material Library* based on our production shading model, which for recent shows has been extended to the full set of look-development tools that our artists use.

Here you're seeing a small set of library presets that were developed for *Star Wars: The Rise of Skywalker*. On the left are hand-authored Imperial Tech materials used for sequences such as the Battle of Exegol, and on the right are sand and rock materials that were physically captured in the Jordan desert for use in building digital environments that could blend with principal photography.

As an illustration of how these presets were used in production, here’s a breakdown of the Pasaana Speeder Chase sequence, in which the terrain was constructed as a digital environment by Industrial Light & Magic.

Starting with the Jordan desert presets from our Common Material Library, ILM artists built the complex, layered materials for this CG environment using multiple look development packages, ultimately blending the results with both green-screened and on-location photographic elements.

Another broader example is Autodesk's upcoming *Standard Surface Library*, which is designed to provide consistent visuals for materials across the full suite of Autodesk tools, including not only media and entertainment packages like Maya and 3ds Max, but also architecture and design applications like Revit and Inventor.

THE ROAD AHEAD

USD AND MATERIALX

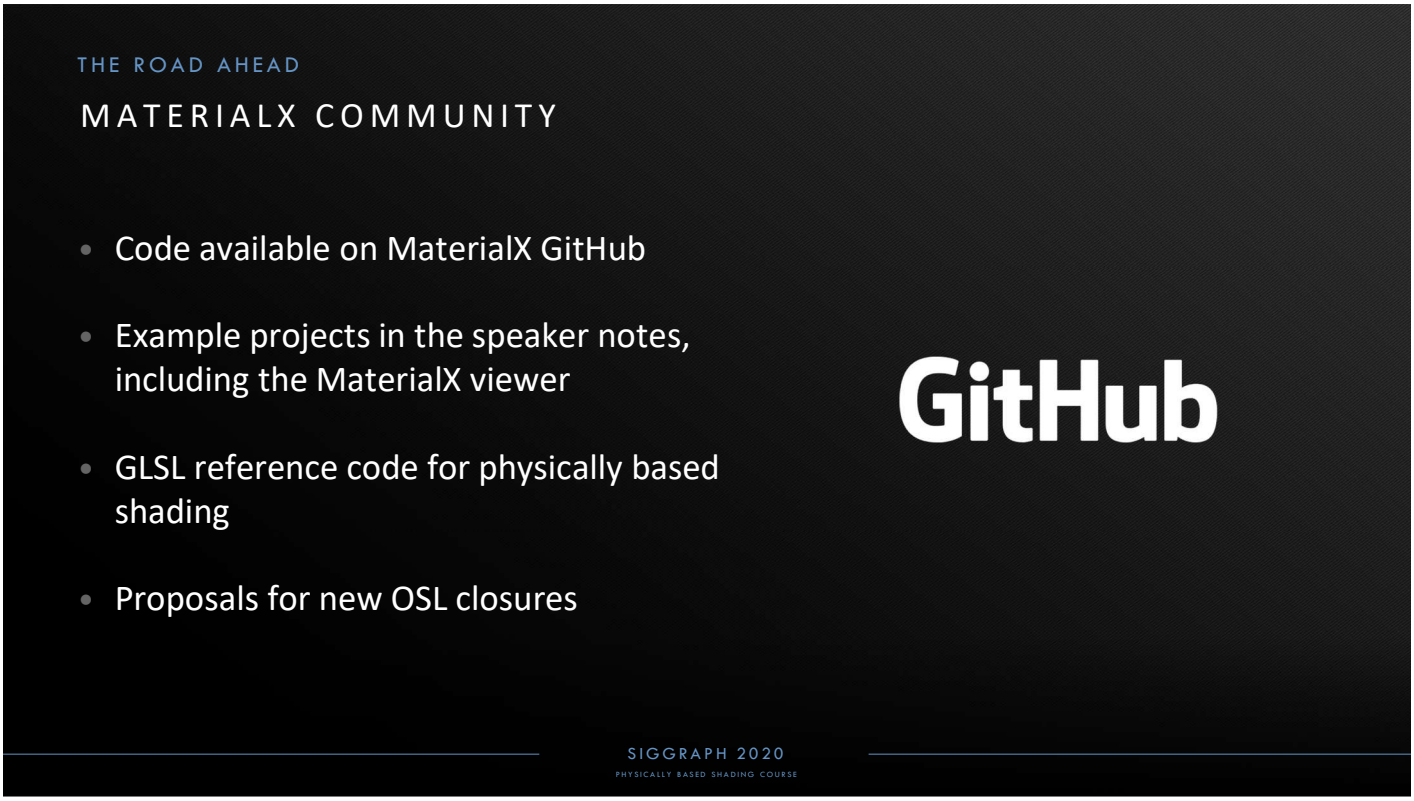
- First glimpse of MaterialX code generation in Hydra RenderMan
- Currently uses OSL code generation for pattern graphs
- Future goals include BSDF graphs and additional Hydra renderers

MATERIALX IN HYDRA RENDERMAN

Finally, we're excited about this recent project from our colleagues at Pixar, which takes a first step towards integrating MaterialX with their Unified Scene Description, or USD.

In this render you're seeing a first glimpse of this work, with OSL code generation used to display MaterialX assets in the RenderMan back end for Hydra.

In this initial prototype, code generation is only used for pattern graphs, but Pixar ultimately plans to extend this work to physically based shading graphs and additional Hydra renderers such as Storm.



For those who are interested in following up on the work we've discussed today, the code is all available on the MaterialX GitHub, and we greatly welcome contributions from the community.

We'll add links to example projects in our speaker notes, and one particularly good starting point is the MaterialX Viewer, which uses GLSL code generation for its real-time viewport, and supports the full set of physically based shading nodes.

Because our GLSL codebase needs to support all combinations of BSDFs and light types, it can also be a useful reference for real-time physically based shading, and we welcome ideas from the community on how to make it more effective and approachable for new readers.

On the OSL side, we're interested in proposing additional closures that would complete its support for the MaterialX physically based shading nodes. For developers who are interested in participating in this discussion, we'll provide links to that project in the speaker notes.

MaterialX viewer:
https://github.com/materialx/MaterialX/blob/master/documents/DeveloperGuide/Viewer.md

GLSL shading implementations:
https://github.com/materialx/MaterialX/tree/master/libraries/pbrlib/genglsl

OSL closure discussion:
https://github.com/autodesk-forks/OpenShadingLanguage/tree/adsk-contrib/dev

Thanks for watching our presentation today, and we'd like extend special thanks to the artists, producers, developers, and researchers who have made this work possible, including the many talented folks on this slide.