Putting the Pieces Together:

A Physically(ish) Based Approach.

Lee Kerley : Sony Pictures Imageworks
lkerley@imageworks.com

Hello, my name is Lee Kerley and work at Sony Pictures Imageworks. Today I'm going to be talking about how we put these physically based pieces together to create materials, and tools, for artists to use in production.

Me

- Shader Writer/Lookdev Tools for 15+ years.
- Currently Shading Dev Lead at Imageworks.
- OSL in SPI Arnold for the past 10.
- Katana development for the past 10 years.
- Collecting credits on Spiderman movies.

SONY PICTURES imageworks

I’ve worked on one of two Spiderman movies through the years, and at least three of them were made using the technology I’m going to show you today.

What we are NOT going to talk about

- Using correct units.
- BXDFs.
- Sampling.
- Microfacets, Normals, Vectors or much else in the geometric world.
- See “Revisiting Physically Based Shading at Imageworks” PBS Course 2017 and “Physical Production Shaders with OSL” PBS Course 2012.

SONY PICTURES imageworks

I want to start out by telling you what we are NOT going to talk about today. There will be no discussion of using correct units, no mention of BXDFs or sampling, and nary a muttering of microfacet, or any geometric math.

At Imageworks, a lot of these things are handled inside the renderer and exposed in OSL as closures. These were discussed in the 2017 edition of this course by Chris Kulla and Alejandro Conty, as well as the earlier presentation in 2012 by Adam Martinez.

The OSL shaders are responsible for putting together those building blocks and delivering the renderer a description of the material.

Closures in SPI Arnold

“A Closure is an object that describes the procedure for evaluating the BSDF.”

Adam Martinez PBS Course 2012

- Coatings : clearcoat(), sheen(), thinfilm()
- standard() : diffuse, 2xGGX and metal
- dielectric() : glass with 2xGGX and internal volumetric properties
- bssrdf()
- hair()
- emission(), transparent(), holdout()

SONY PICTURES imageworks

A closure in OSL is a description of the BXDF that is then interpreted and implemented by the renderer.

The current set of closures used by the shaders has been really stable over the past five years since we introduced this latest generation of the shader library at Imageworks.

The coating closures, accept the closure beneath as arguments so the renderer can appropriately compensate for energy absorbed or reflected by those coatings.

The standard() closure incorporates a diffuse model, two GGX specular lobes and our metallic model, with blending weights provided to maintain energy conservation internally. My mental model for the standard closure is that it handles all the things that can happen on the “outside” of the surface.

Conversely, the dielectric() closure handles the things on the inside of the surface, so anything that is refractive like glass, volumetric or subsurface. The dielectric has a “Thin” and “Solid” mode, which we will hear more about later.

This selection of closures was largely motivated by the choice to be strict about blending configurations.

What we ARE going to talk about

- Taking the building blocks (OSL closures) and putting them together.
- Tools that we've built for the artists to give them as much creative freedom as we can, while still keeping them on the physically(ish) based straight and narrow.
- Horizontal layering at the material level.
- A tiny bit of math about how we blend our double GGX specular model.

SONY PICTURES imageworks

What we ARE going to talk about today is how we put those pieces together and how they are presented to the artists in a friendly, yet powerful way; allowing them the creative power to experiment quickly, while at the same time staying (mostly) within the bounds of physical correctness.

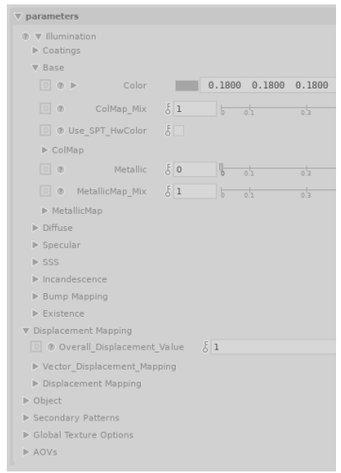
I'll show some of the tools we built, and also talk about how we hide away much of the energy conservation concerns from the artists in the tooling, so they can't break the rules too much.

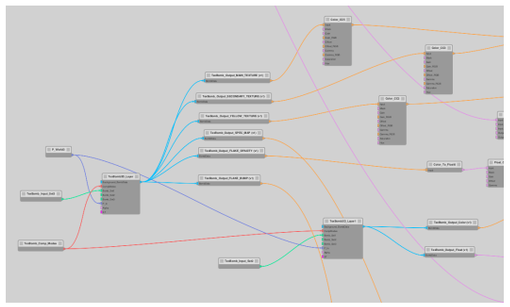
In contrast to the previous MaterialX presentation that included vertical and horizontal layering at the granularity of the BXDF, we have a fixed vertical layering strategy for the composition of the closures, and instead perform horizontal layering at the granularity of entire materials with OSL and the Katana tools we have developed.

Finally, I'll talk a little about the technical specifics of how we blend the double GGX lobes in a way that tries to maximize the preservation of the original artist intent, and a few other tricks we learned along the way.

What is a ‘Material’?

- Monolithic user interface with the ability to procedurally texture its inputs.





- PatternCreate : Node-based procedural texturing tool in Katana to modify material inputs. Nodes are OSL subgraphs.

SONY PICTURES imageworks

Firstly, what do I mean when I say “Material”? At Imageworks we have always presented lookdev and lighting artists with a monolithic material user interface. This has proven particularly useful as shading technology changed, including the adoption of Physically Based Shading, as it allowed us to evolve the technology beneath a familiar interface. This eases the burden for re-training and makes adoption of the new technology easier to roll out.

For a long time we had a large library of different materials with automatic texturing and some basic procedural support built in, but as shows needed increasing levels of visual complexity, there was more call for custom shader development on each show and it was clear that the team couldn’t scale in response to the work that was needed.

About seven years ago, we introduced a tool called PatternCreate that allowed artists to be able to use a nodegraph-based approach to texture the inputs to the materials with patterns. All of the pattern nodes are authored by the shading team in OSL and delivered to the artists in the form of a library. This allowed a lot of the bespoke production development to be completed by technical TDs while still allowing less technical artists to be able to interact with and use the material system without feeling overwhelmed.

What is a ‘Material’?

- Small set of base materials:
  - Constant (aka Emission).
  - Basic (aka Plastic).
  - Metal.
  - General (Basic+Metal+SSS with energy-conserving weights).
  - Glass (Refractive with internal volumetric scattering support).
  - Hair.
  - Feather/Skin/Eye - these are just convenient specializations of the materials above re-parameterized to give a consistent experience to the user.

SONY PICTURES imageworks

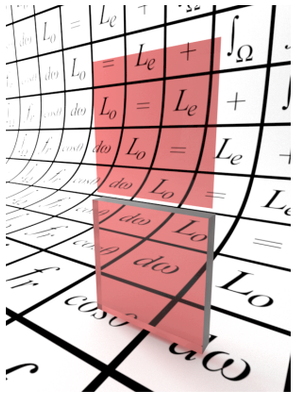
In the latest generation of the shader library we provide the artists a small set of base materials to start working from, these are built from the closure building blocks we are provided by Arnold in OSL.

The Glass and General materials are the two that use the dielectric closure, so they have a volumetric component to them. We have two different versions of each of these that can be used depending on how the asset is modeled, which we will see on the next slide.

The artists then use these materials in combination with PatternCreate to create rich material hierarchies inside of Katana.

Material Thickness

- Thin and Solid versions of the volumetric component.
- “Solid” model used on objects modeled with physical volume.
- “Thin" model used on single thickness surfaces.
- “Thin” model simulates thickness using user-provided “Thickness” material parameter.
- Materials with no volumetric component do not need a thickness model.



SONY PICTURES imageworks

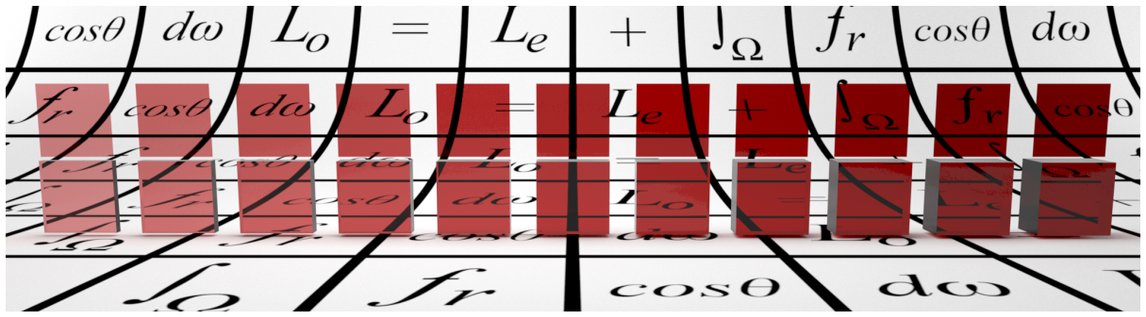
As I mentioned we have the idea of thickness as a differentiator between materials in the system.

<SLIDE>
Here we can see two models, a single poly plane, and a box. We use the “Thin” version of the volumetric shading component for the single poly plane and the “Solid” version for the box.
<SLIDE>
The “Thin” version simulates the thickness inside the material model, and the two versions appear visually similar when used with the correct geometry. Here we have dialed the “Thickness” parameter of the “Thin” model to the actual thickness of the box geometry, so their appearances match.

For materials that don’t model any volumetric effect, the thickness model is irrelevant. We call these “Auto” because they can automatically adopt the thickness model of other materials that they are combined with.

Nearly all the other material parameters are floating point and so can be interpolated, but the thickness model is a geometric decision. This is why we decided to introduce the separation here, so it isn’t possible to end up accidentally mixing the two or end up using the wrong model on the wrong geometry.

Material Thickness

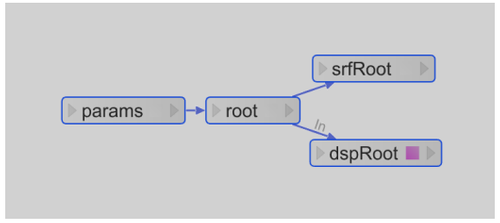


Here as we wedge the Thickness parameter of the “Thin” material to match the geometric thickness of a “Solid” material, their appearances will approximately match.

When we simulate the thickness in the “Thin” model we ignore the effects of ray bending due to the IOR at the medium boundaries. This is generally OK because we tend to use this only for things that would be relatively thin if they were modeled as solid objects.

Material Topology

- Encapsulates both surface and displacement shaders together which allows for easier sharing of patterns.
- Each material a small OSL network with the same topology.



SONY PICTURES imageworks

A material provides both a surface and a displacement shader to the renderer, but the user is not aware of the two separate pieces, because it is presented as a single monolithic interface.

The materials themselves are a small network of OSL nodes, and all have a very similar topology. Here we see a slightly simplified example.

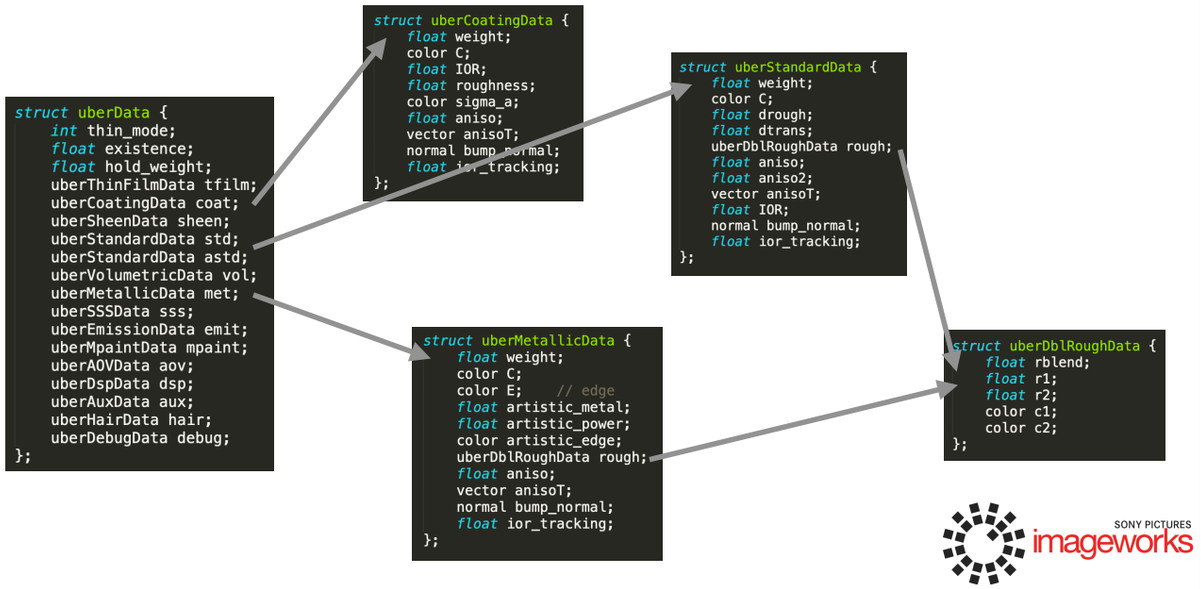
The ‘params’ node is the only node that differs per base material. It provides the material-specific logic.

The central ‘root’ node defines material elements that are common to all materials, such as coatings and AOVs.

The srfRoot and dspRoot nodes are responsible for taking the accumulated material data and constructing the surface and displacement shader outputs that the renderer expects.

The connections in the network are the UberData struct which we will se next.

UberData struct-o-structs



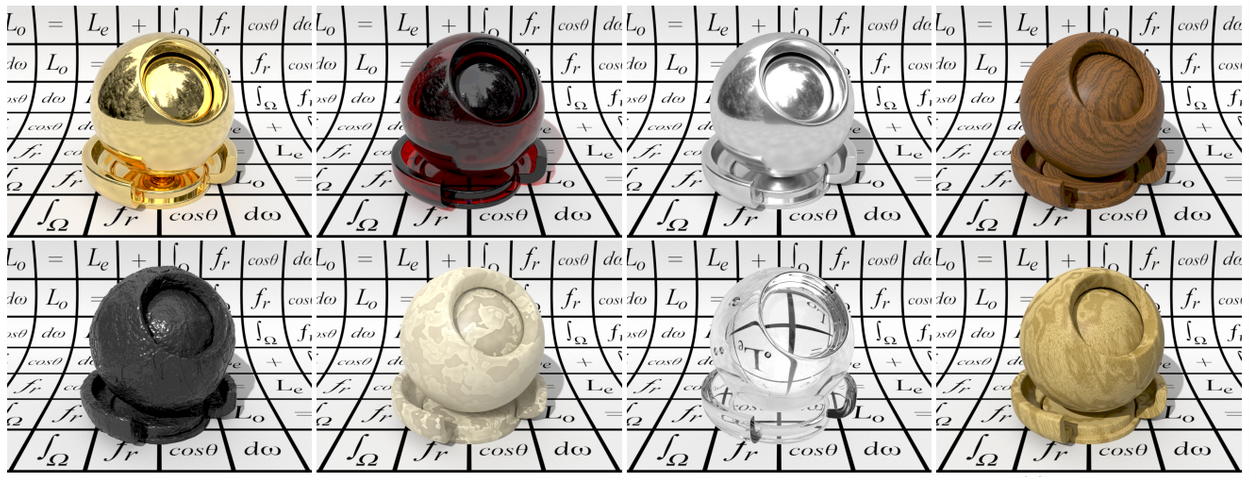
The UberData struct is the heart of the system, and it is actually a struct of structs.

Each sub-struct represents the one of the illumination building blocks in the system, and contains the data necessary to represent that component in the renderer. Loosely, the members of these structs map to the parameters of the closures, although this isn't always strictly true. There are times where we can derive some closure parameters from multiple members in the struct, or other times where we reuse a closure to represent a simpler struct and so a number of the parameters of the closure have default values not captured in the sub-struct. Our standard() closure actually contains our plastic and metal, but we choose to separate these in the struct to allow for better blending later on.

Each sub-struct has its own weight, which controls how much of that sub-struct will eventually be present in the material. These weights are controlled by inputs from the artist, but logic in the OSL "params" nodes ensures that the weights always sum to 1 for the illumination sub-structs.

The sub-structs are completely independent from each other. Each component might have its own copy of the surface normal, roughness or color parameters. This becomes really important when we start thinking about combining materials with each other as we want to make sure that some of these parameters don't pollute each other when blending.

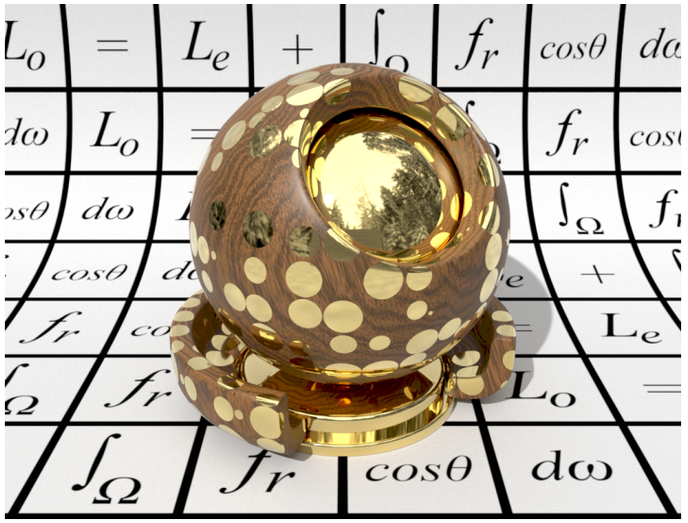
Materials



We can see here a small collection of materials artists created using these base materials with PatternCreate.

Shows will often create, or steal from another show, a large set of these materials to create a material library to start from.

Complex Materials



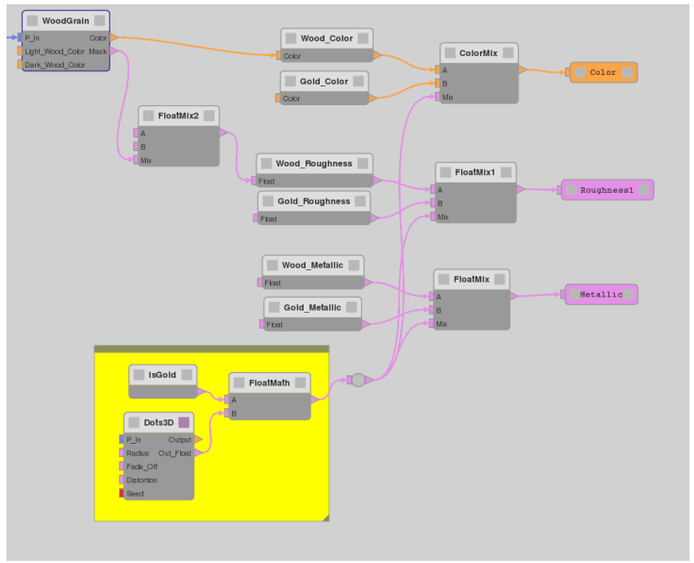
SONY PICTURES imageworks

A lot of asset lookdev can be completed using materials from these libraries, but from time to time there can be a need to combine more than one material together to complete an asset. Maybe the shape has been modeled such that two logically different components have been combined into a single mesh. Perhaps the handle and blade of a knife were merged, and we want a wood handle and a metal blade.

Or sometimes the artist just needs to separate out the material into different logical blocks for easier dialing.

Here we have a contrived non-production example where a wooden asset is being covered in gold spots. We need a single material we can assign to this shape that incorporates both sets of material properties.

Complex Materials



- Mixing two “materials” inside PatternCreate, by mixing the two sets of input parameters.
- Laborious to construct and error prone, easy to get wires crossed.

SONY PICTURES imageworks

In the previous version of the shader library, we would sometimes find that artists would reach for PatternCreate to perform the combining of the materials. This approach is effectively artist-created parameter blending.

Here we see the overly complicated pattern nodegraph, where really the only interesting blend logic is in the yellow backdrop at the bottom.

This approach is quite labour intensive, easy to make mistakes and also clutters the pattern graph.

What we need is a node-based tool, but for whole materials, and that’s how we ended up with MaterialCombine.

MaterialCombine

- Nodegraph-based material blending tool.
- Re-uses the pattern node library from PatternCreate.
- Parameter blending inspired system - uberData structs are blended, surface and displacement properties are blended synchronously.
- Library of “Combine” nodes are also provided to manipulate the combined material in a number of ways

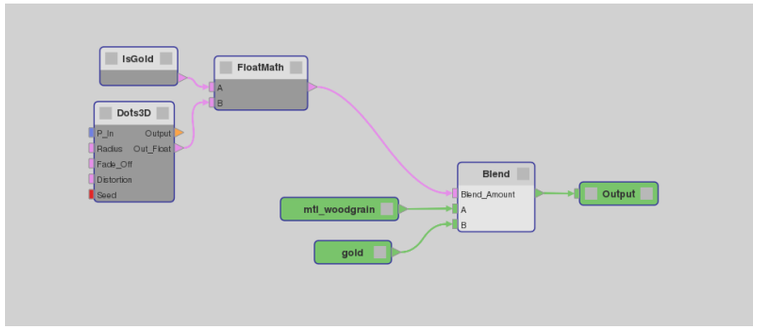
SONY PICTURES imageworks

MaterialCombine is a node-based material blending system. Because the system is built on top of the same technology as PatternCreate we get to re-use the pattern nodes we already have.

Its not strictly a parameter blending system. We don’t blend the input parameters of the materials, but instead we blend the contents of the uberData struct.

We’ve built a small library of “Combine” nodes that allow the blended material to be manipulated in more interesting ways than just ‘Over’.

MaterialCombine



SONY PICTURES imageworks

Getting back to our example from before, here we see the corresponding MaterialCombine graph that creates an identical image. The green nodes here represent entire materials, and we end up with a system that is much easier for the artists to comprehend. This means they are now able to craft much more complex combined materials.

Combine Nodes

- Library of “Combine” nodes are also provided to manipulate the combined material in a number of ways.
- Blend or Over to combine entire materials.
- Replace or blend in subcomponents from other materials, like Coatings, Displacement or even Existence.

SONY PICTURES imageworks

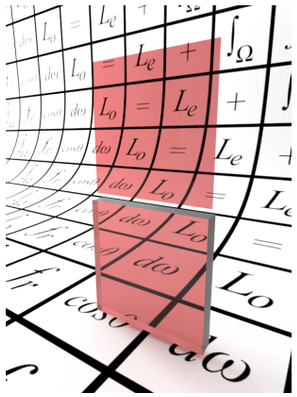
The Blend node is the heart of the entire system and provides a “simple” linear blend between two materials. It turns out that a simple linear blend isn’t quite as simple as it first seems, and we’ll get to that in a slide or two, but this node is really the workhorse of MaterialCombine.

We also have a slightly more familiar Over node which just uses the existence of the foreground layer to drive the mix of the Blend node.

Other combine nodes allow artists to blend in, completely replace or redefine a number of the subcomponents within the UberData struct. ReplaceSheen has proven useful for adding dust to the tops of existing materials really easily, and we often see artists stealing the displacement component from one base material and re-using it with a more complex combined surface material component.

Thickness Model

- Remember: “Thin” and “Solid” are not blend-able with each other.
- Other “Auto” materials that do not have a specific thickness model can be blended with anything.
- Hair is a separate material type that does not blend with anything other than Hair.
- Tools manage the enforcement of the rules, so artists don’t have to think about it too much.



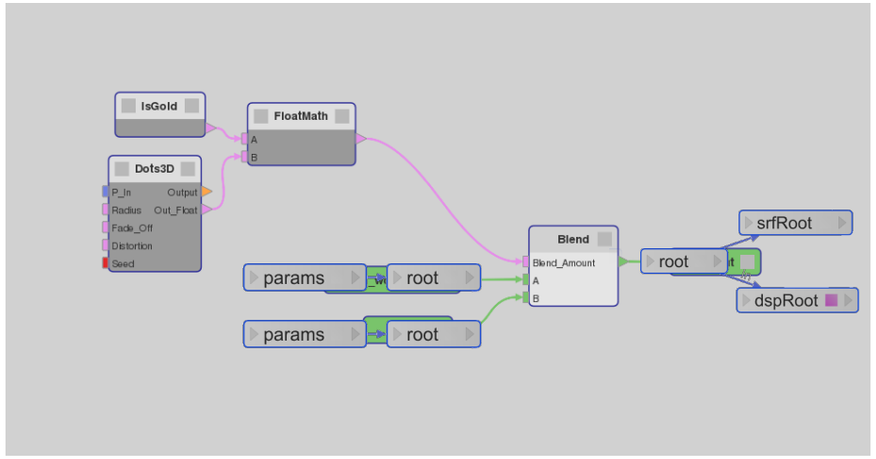
SONY PICTURES imageworks

We mentioned before that some of the materials are intended to be used with geometry modeled as a single surface, and others with geometry that has been modeled with an interior volume.

A combined material is still a single material, so it needs to have only one thickness model. Consequently, the tools enforce that Thin materials cannot be blended with Solid materials and vice-versa, and materials that do not have a thickness model, such as emission or plasti-metal, are allowed to be combined with anything.

The tools also automatically derives the appropriate thickness model based on the input materials that are used in the graph.

Connecting Materials



Next to talk a little about the mechanics of how we connect the materials.
Remember that the materials all share a common nodegraph topology.

<SLIDE>
Here we see a MaterialCombine graph and to connect the materials layers we slice that material topology in half and just keep the first part.
<SLIDE>
And we reuse the last part for the Output, since we still need to end up with one surface and one displacement shader
<SLIDE>

This allows us to use a common set of nodes to deliver the closures to the renderer no matter if the material is a simple single material or a combination of many.

Katana Nodes

- PatternCreate and MaterialCombine allow:
  - Inheritance of material data, to allow hierarchies of materials to be easily created and maintained.
  - Authoring of data at any point in the nodegraph.
  - Inspection of data downstream at any point.
  - Overriding of material parameters at specific geometry locations orthogonally to the material that is assigned.

SONY PICTURES imageworks

Both PatternCreate and MaterialCombine exist within Katana, which is a node-based look development and lighting package. This affords the artists a large amount of power over and above operations on a single material location.

Materials are be authored in a tree hierarchy, where child materials inherit their parent material configurations, allowing for sparse edits to create entirely new materials to be used.

A new material can be created, its Patterns or the MaterialCombine graph created or modified at any point in the Katana nodegraph. This means that the data can also be inspected at any point in the graph as well.

Also any parameter of any pattern, material or material layer inside a MaterialCombine graph can be overridden at any geometry location independent of the material assigned to that location. For example it is very easy for artists to just set Overall_Displacement_Scale to 0.5 for large parts of their scene to scale down displacement across a large number of assets with a vast number of different materials assigned.

Blending Materials

- Data pollution.
- Weighted mixing.
- Double GGX blending.

SONY PICTURES imageworks

As I said a little earlier, the Blend node is really the heart of the entire combining system. We don't blend the input parameters on the materials, but we blend the contents of the UberData struct.

<when we blend materials we need to take care that data from one component does not pollute another as we blend two materials together.

we also need to make sure that we're balancing the weights of each building block to maintain energy conversation.

Finally, I'll talk about a little trick we do with our double GGX lobe that means that when we blend, most of the time, we can do a lot better than the naive approach of linear blending.>

There are a few things that we need to take care with when we're blending that we'll talk about in a little more detail.

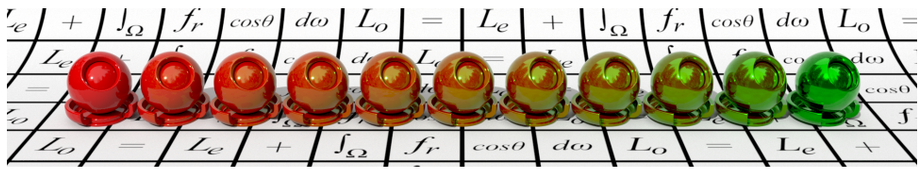
We need to take care that data from one component does not pollute another as we blend two materials together.

When we blend materials with different building blocks we need to make-sure we are balancing the relative amounts of each building block to maintain energy conservation within the material. If we add in metal without removing a corresponding amount of glass, we'll be passing the renderer a material with too much "material" in it and it will no longer be energy conserving.

Finally, I'll talk about a little trick we do with our double GGX lobe that means that when we blend, most of the time, we can do a lot better than the naive approach of linear blending.

Data Pollution

col: (1,0,0) col: (0.5,0.5,0) col: (0,1,0)



pl_col: (1,0,0)
mt_col: (0,1,0)

pl_col: (1,0,0)
mt_col: (0,1,0)

pl_col: (1,0,0)
mt_col: (0,1,0)

SONY PICTURES imageworks

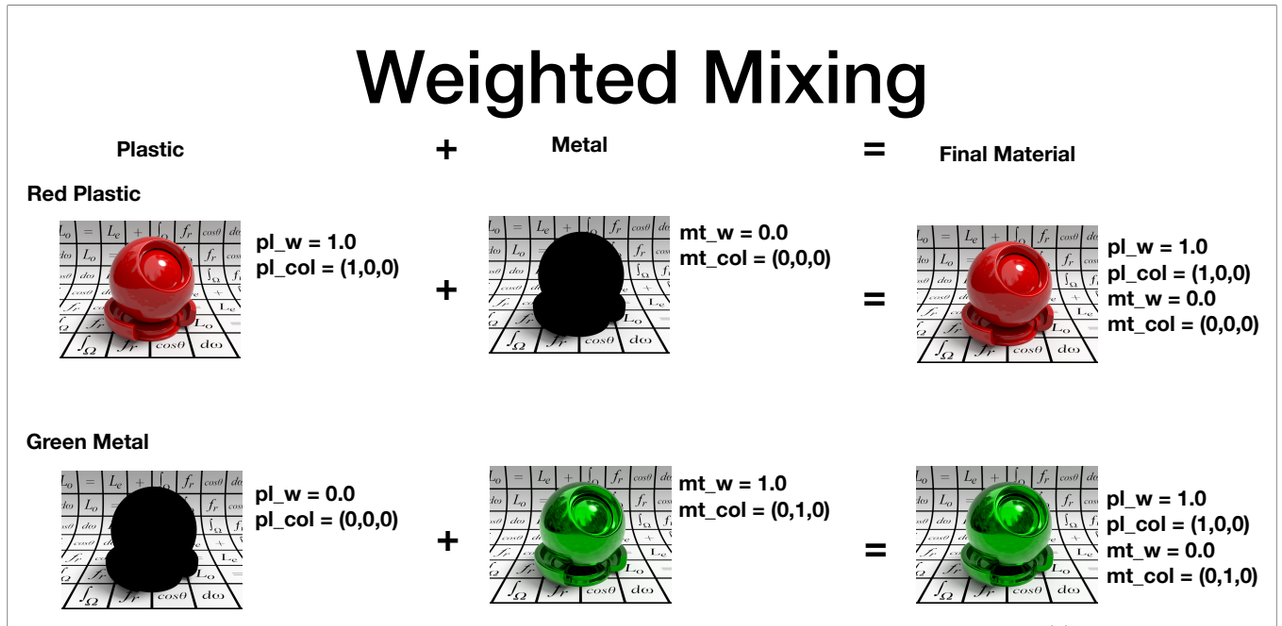
Within the UberData struct, each sub-struct is independent from the others. Some parameters, like Color or Surface Normal, are repeated in several sub-structs. This means we can blend those structs without incorrect cross-pollination of data.

Here we see an example where we are blending a Red Plastic with a Green Metal.

The top example demonstrates what happens when we don't separate the color in the metal and plastic sub-structs. You can see once we get to the middle of the blend the color looks like neither of the original colors. We get a polluted color because we've just done a simple linear blend between the two colors.

Below we see that if we keep the metal color and the plastic color separate, this gives us a material with a more pleasing color at the midpoint. This is because, by duplicating the color in each of the components, we now just need to blend the weights of those components. We just mix from being plastic to metal, but the plastic can stay Red for the entire blend.

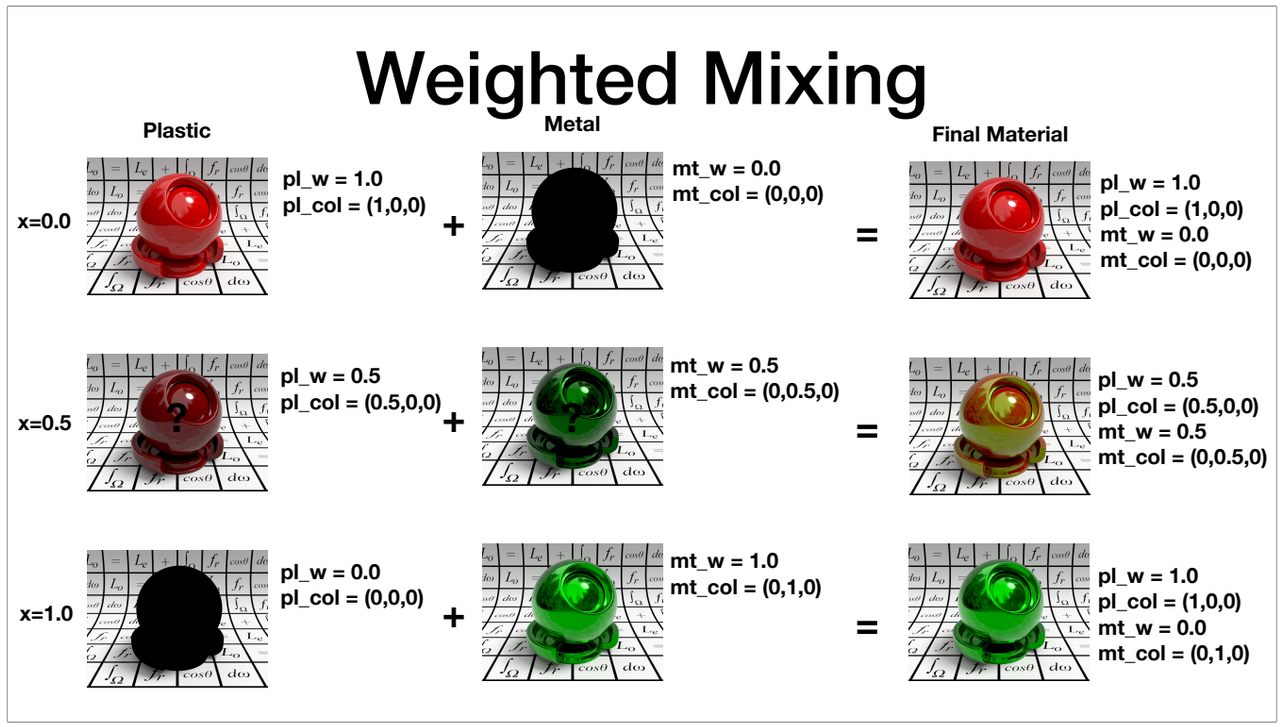
We will see more of this weighted mixing next…



We’re going to look at a simplified example of how we blend the different components of the UberData struct using their weights.

We’re just going to consider a material that can be metal or plastic, two components which are separate structs in our model. At the end they are just linearly summed to get the result, this is possible because the system ensures that the sum of the weights never exceeds 1.
<SLIDE>
Here we can see that the Red plastic material has a plastic weight of 1 and plastic color of red, but the metallic component has a weight of 0 and color of black(the default), because the artist never made a choice here.
<SLIDE>
The Green metal material is the complement of this, with a plastic weight of 0 and a metal weight of 1, and no plastic color specified.

We can see on the right-hand side the summed up final material.



Now we're going to blend these two materials together, using a blend amount x.

If we carry out a straight-forward linear blend of all the values using x, then we know we'll have the correct result at x=0 and x=1. But what do we get in the middle? We'd like it to look something like this….

<SLIDE>
But linearly interpolating the weight and the color using x leads to a result here that is darker than we expect it to be.

So instead we use weighted mixing to do a bit better. The crux of weighted mixing is the wmix() function.

wmix()

```
// this returns the amount we should mix parameters of A and B,
// knowing that A and B only exist according to their weights

float wmix(float weight_A, float weight_B, float x) {
    if (weight_A == weight_B)
        return x;  // both A and B exist in equally, just return x. aka LERP.
    if (weight_A == 0)
        return 1;  // A has nothing, take everything from B
    if (weight_B == 0)
        return 0;  // B has nothing, take everything from A

    // NOTE: the ifs above just avoid numerical singularities
    // but are otherwise continuous with this general formula
    float a = (1 - x) * weight_A;  // how much of A
    float b = x * weight_B;        // how much of B
    return b / (a + b);
}
```

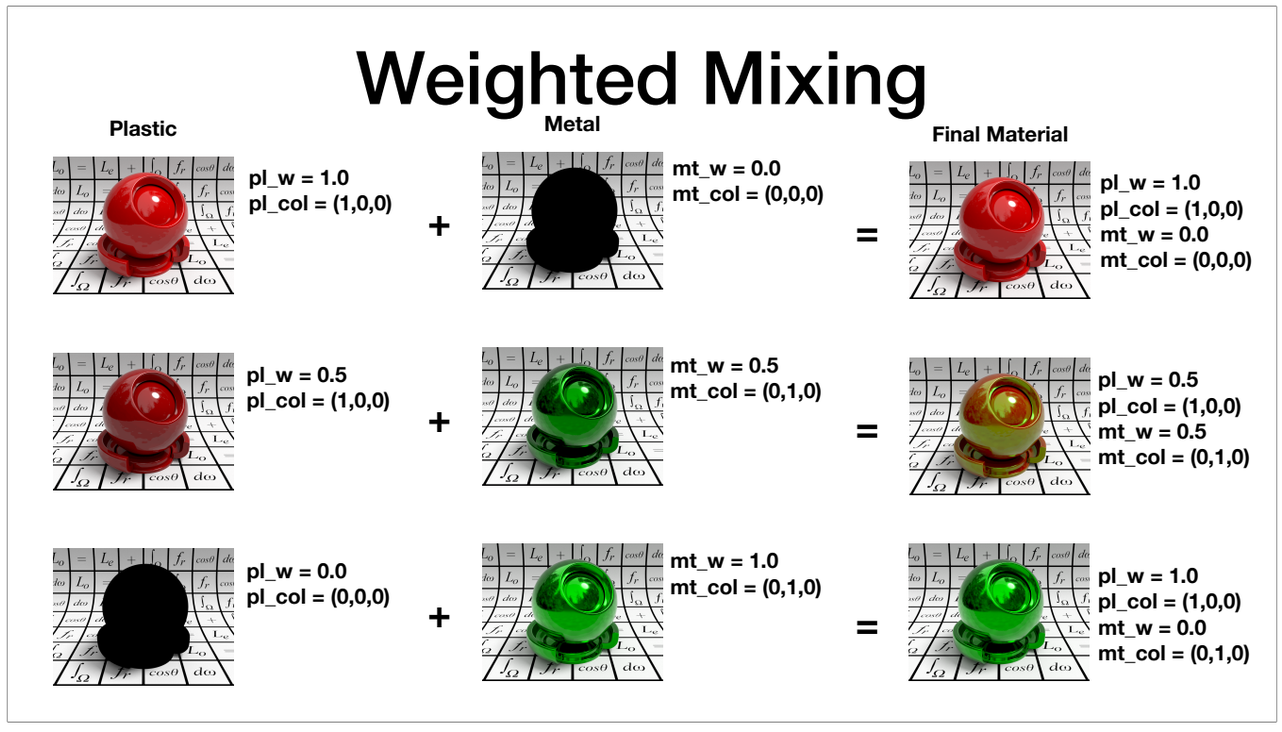
You can see here that the amount we take from A or B is not only affected by the overall blending weight x, but also by the weights of the respective sides as well.

We have a few early out cases for optimization. These optimizations also make it a little easier to understand the goals of the function. If the weights on both sides are the same then a regular linear interpolation is good, or if either side has a weight of 0, then we just want to use everything from the other side.

The generalized formula below is just the continuous version of the above optimizations, and basically says use more of the side that you have a higher weight for.

Once we have the wmix value for a sub-struct then in most cases we just perform a linear blend between the two using this newly calculated weight, but there are a few special cases that I'll touch on in a minute.

It might look like we have a long series of conditionals, but the run-time optimization inside of OSL will often optimize these away. In fact the run-time optimization in OSL is one of the key pieces of technology we needed to allow artists the freedom to author their own nodegraphs. We often see a reduction in the order of 90-95% of the instructions in the shader and without that we would end up with very large materials that would be too expensive to evaluate.

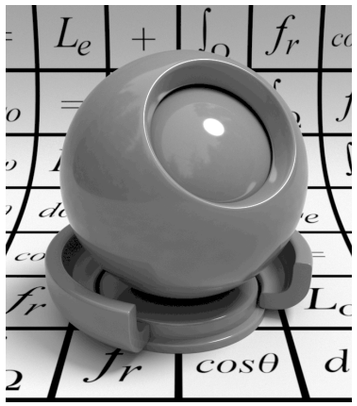


Concretely what that means in our simple blending case is that when we're blending the plastic component for x=0.5 we keep the original red plastic color. This is because the plastic weight of the green metal material is 0, and so our wmix() value just selects all of A. We just keep all the plastic component data untouched and only linearly blend the weight.

This gives us the result we want, which in this situation is exactly correct. In other more complex real-world solutions, the blend falls as close to the correct solution as we can get by blending.

Double GGX Lobes - Why?

- Allows us to fit more measured data.
- Linear blend between two energy-conserving lobes.
- More information presented in “Revisiting Physically Based Shading at Imageworks” PBS Course 2017.



Roughness = 0.1

SONY PICTURES imageworks

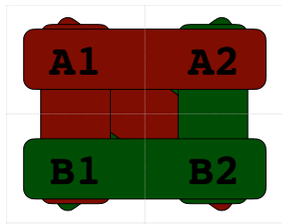
We have two GGX lobes in our model because it allows us to achieve more complex specular profiles without introducing more complex models. The two lobes are just blended linearly, so we’re always energy conserving.

Here I have sharp spec at roughness 0.1
<SLIDE>
And a broad spec roughness at 0.5
<SLIDE>
With two spec lobes I can blend these to together to retain my tight spec and still have a sense of that hazy spec underneath.
<SLIDE>
If we only had one spec lobe we would have to find some poor compromise, like here with the Roughness at 0.3.
<SLIDE>

This was covered in more detail by Chris Kulla in the 2017 course.

Blending Double GGX

- Two input materials *A* and *B* to be blended by a wmix() value *x*.
- Each has 2 GGX lobes, with roughness values *r1* and *r2* and a blend amount *rBlend* between the two.
- 3 permutations:
  - Option 1 : (A1, A2) & (B1, B2)
  - Option 2 : (A1, B1) & (A2, B2)
  - Option 3 : (A1, B2) & (B1, A2)



SONY PICTURES imageworks

One of the cases where we can do better than the simple linear blend of the data inside the sub-structs is with our double GGX lobes.

Our shading model has two GGX lobes, so we have two roughness values and a 0-1 blend between the two lobes in each material. If we have two input materials A and B that are being blended and have computed a wmix() value x, then there are a 4 lobes in total to consider. We need to reduce that to the two final output GGX lobes.

There are three different ways we can pair up these 4 lobes to make two final lobes.

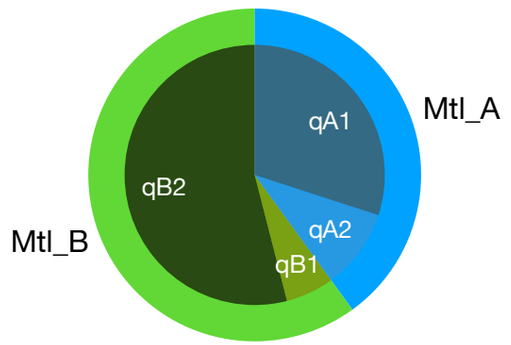
<SLIDE>
Option 1 : Blending the two lobes from A to create the first output lobe, and then the two lobes from B to create the second output lobe.
<SLIDE>
Option 2 : Blending both the first lobes from A and B to create the first output lobe, and then the second two from A and B to create the second.
<SLIDE>
Option 3 : Crossing both the lobes and the materials to create the output lobes.

So, now we just need to pick which of these three combinations is the “best”…

Blending Double GGX

- We have four GGX lobes *A1*, *A2*, *B1*, *B2*.
- Quantify the amount of each lobe using *x*, the mix value, and the *rblend* from each material, which represents how much of each lobe is used in each material.

```
// figure out how much of each
// roughness we have
qA1 = (1 - A.rBlend) * (1 - x);
qA2 = A.rBlend * (1 - x);
qB1 = (1 - B.rBlend) * x;
qB2 = B.rBlend * x;
```



x = 0.4
A.rblend = 0.25
B.rblend = 0.9

SONY PICTURES imageworks

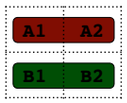
We have a simple example here where x is the wmix between material A and material B, and A.rblend and B.rblend are the mixing factors within the materials between the two GGX lobes.

To select which one of the three permutations is the best, we start be quantifying how much of each of the input lobes we have, by using x and the two rblend values.

By visualizing this, we can intuit that we'd like to end up with a mix that prioritizes retaining A1 and B2 as separate lobes, as they appear to be dominant.

Blending Double GGX

- Quantify the “cost” of each of the three options. This is the sum of the cost of the two output lobes.
- Each lobe “cost” is calculated by using the absolute difference between the input roughness and scaling by the quantity of each input lobe used.
- Absolute difference is a measure of how much error will be introduced by combining two roughnesses.
- Scaling by the quantity of each lobe means that if either input lobe is not present then we will get an error of 0, because the output lobe can just exactly adopt the other input lobe perfectly.
- Select the option with the lowest “cost” and then we blend the two lobes together using the quantities we already calculated and the wmix().




```
cost = abs(a.r1 - a.r2) * (qA1 * qA2) +
       abs(b.r1 - b.r2) * (qB1 * qB2);
```

SONY PICTURES imageworks

Next we calculate the quality of each of the options by measuring the “error” introduced.

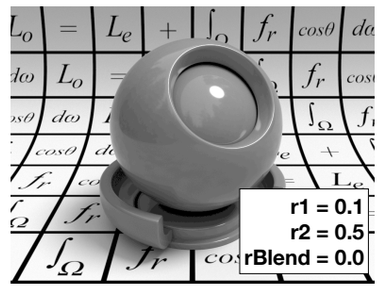
The cost of each lobe is calculated by considering the difference in roughness between the two input lobes that contribute to it, and then the overall cost is the sum of these costs scaled by how much of each lobe will exist.

Once we’ve calculated the cost of all three options.
Then we can choose the option with the lowest cost and we know which combination to use.

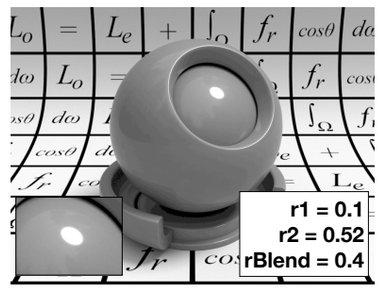
Once we’ve made the selection, we blend the two lobes together using the familiar wmix() weighting approach but this time with the quantities of each lobe we calculated earlier.

This is a bit of a simplification as our actual implementation of the cost function also incorporates the specular tint and an attempt to consider anisotropy changes.

Blending Double GGX



Opt 1 : (A1,A2) (B1,B2) Cost = 0+0.012



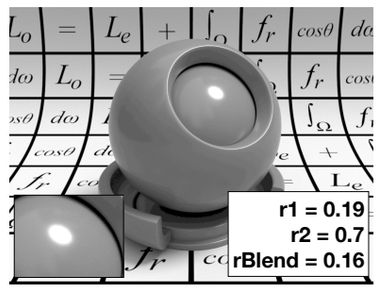
blendAmount = 0.4

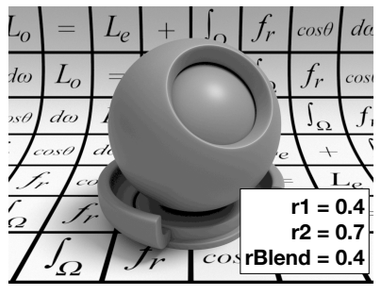
qA1 = 0.6
qA2 = 0.0

qB1 = 0.24
qB2 = 0.16

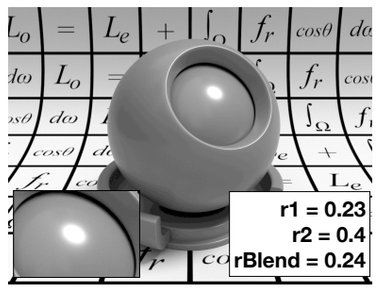
```
cost = abs(a.r1 - a.r2) * (qA1 * qA2) +
       abs(b.r1 - b.r2) * (qB1 * qB2);
```

Opt 2 : (A1,B1) (A2,B2) Cost = 0.043+0





Opt 3 : (A1,B2) (B1,A2) Cost = 0.057+0



Let’s run through a quick concrete example to make this a little clearer.
<SLIDE>
Here we have two materials A and B, and notice that on the A side we have an rBlend of 0, which means we’re only using one of the GGX lobes.
<SLIDE>
We are going to blend these two materials with a blend amount of 0.4, which means using the four quantity values for each of the lobes and cost function from the previous slide we can calculate the cost of each of the three options.
On the A side qA2 is zero because we’re only using one of the lobes, so that means the costs for the side that includes that lobe will always be zero.
<SLIDE>
The differences are a subtle, but hopefully you can see that the lowest-cost option here, option 1, is the one that retains the original tight spec that is dominant on the side we’re blending more from.

Blending Materials

- IOR - we convert to reflectance, to do a linear blend in that space and then convert back.
- Normals - we re-normalize after a linear blend to retain unit length. Very rarely are we blending normals on opposite sides of the surface because of how the data is structured, so we don’t need the cost of a SLERP.
- All the blending logic needs to be continuous.

SONY PICTURES imageworks

Other gotchas where we do a little more than just a simple linear blend include:

IOR, where we found linearly blending reflectance instead of IOR directly gave much better results, so we convert the IOR values, do the blend and then convert back.

We also cannot just linearly interpolate the normals, otherwise we might not have a unit-length vector. We considered a spherical LERP, but found if we took a little care with the directions of the vectors then we could just do a simple linear interpolation and just re-normalize afterwards, which saved us a bunch of trigonometry. Because of how the data is structured, we are nearly always blending two normals on the same side of a surface, so the LERP+normalization works well enough.

Probably the biggest lesson we learned was that the logic we use needs to be continuous; artists quickly stumble across the discontinuities if they exist. So while we strived to find the best ways to combine things, we would always work harder to find a way for there to not be any discontinuities across the blend region. Today we’ve seen both the roughness blending and the wmix() function as examples of this continuous-ness.

Future Work…

- Ever expanding library of nodes.
- Grow the nodegraph tooling to provide more familiar tools that artists depend on to manage complexity.
- Expand the Double GGX lobe to account for more anisotropy changes.
- Unify some of our materials to a more general-purpose uberShader model. Katana material hierarchy imposes some awkward organizational constraints.

SONY PICTURES imageworks

As far as next steps or future work goes, there is always more we can do.

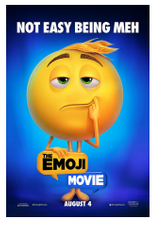
We regularly have requests for more nodes in the system, and it is often a balancing act between just saying yes, and weighing the cost of increased cognitive load on the users of a larger library of nodes.

There is also always work to be done on the graph-based tools themselves, adding more features to help the artists deal with increasing complexity in the materials they make and combine.

We do try a little to account for error introduced by anisotropy changes when blending the double GGX lobe, but there’s room for improvement here.

The other piece of work that’s on the horizon, is to work on merging some of those materials that are presented separately to the user and move to more of an uberShader model. This isn’t so much to make up for a short-coming in the system as it is a response to some of the awkwardness that katana imposes how the data is stored and the ability to reorganize data in a more dynamic way.

Wrapping Up..



SONY PICTURES imageworks

And that about wraps it up…

Today, hopefully I've given you an overview of how we put the physically based shading pieces together at Imageworks. The system I've presented today has been in use for the past five years, on a wide range of animated features and visual effects shows.

This has been the longest running generation of shader library in my time at Imageworks, and I think this is because we have found a good balance between keeping things on rails to create realistic and physically based images, while keeping the flexibility to easily explore creative ideas.

Thanks

- SPI Shader Developers past and present.
  - Ole Gulbrandsen, Daniel Greenstein, Roman Zulak, Derek Haase, Adam Martinez, Jay Reynolds, Andreas Bauer, Søren Ragsdale, Daniel Rolinek, Laurent Hamery, John Schlag, Laurence Treweek, Juan Carlos Moreno, Robert Durnin, Yasser Hammed and John Monos.
- SPI Arnold dev team and Katana dev team.
  - Chris Kulla, Clifford Stein, Alejandro Conty, Larry Gritz, Pascal Lecocq, Lucas Miller and Christian Gloor.
- Imageworks Dev Group Leadership team.

Thanks for your time, listening to this presentation. Thank you to the course organizers for inviting me. And finally, I would like to extend a special thank you to all the people mentioned here whose hard work has made all of this happen, after-all it takes a village to raise shader library.