JF Nested Dielectric

A nice glass of Rainbow Juice.

JF Nested Dielectric is an open-sourced shader for the Arnold renderer by Solid Angle, based largely on the 2002 paper by Charles M. Schmidt and Brian Budge.

http://graphics.idav.ucdavis.edu/~bcbudge/deep/research/nested_dielectrics.pdf

JF Nested Dielectric was written by Jonah Friedman in collaboration with Psyop.

Special thanks to: Andy Jones, Tony Barbieri, Andy Gilbert, Todd Akita, and Nisa Foster.

Nesting Dielectrics

A few nested dielectric media (Water drops, plastic, beverage).

Nested Dielectric, at its core, is meant to solve one problem: Rendering multiple, different, touching refractive media. Examples of this are water drops on glass, glass in water, ice in water, bubbles in glass or water, or any combination of these sorts of effects.

The shading properties of refractive media are governed by two laws: Snell's law, and the Fresnel equations. These two equations have the same 3 inputs:

1. θ - angle between the incoming ray to the normal of the surface
2. n1 - Index of refraction of the medium the ray has just traversed
3. n2 - Index of refraction of the medium the ray is (maybe) about to refract into

Snell's law gives the angle of refraction.

The Fresnel Equations give the reflectance of the medium (and the inverse, 1 – reflectance, gives "refractance").

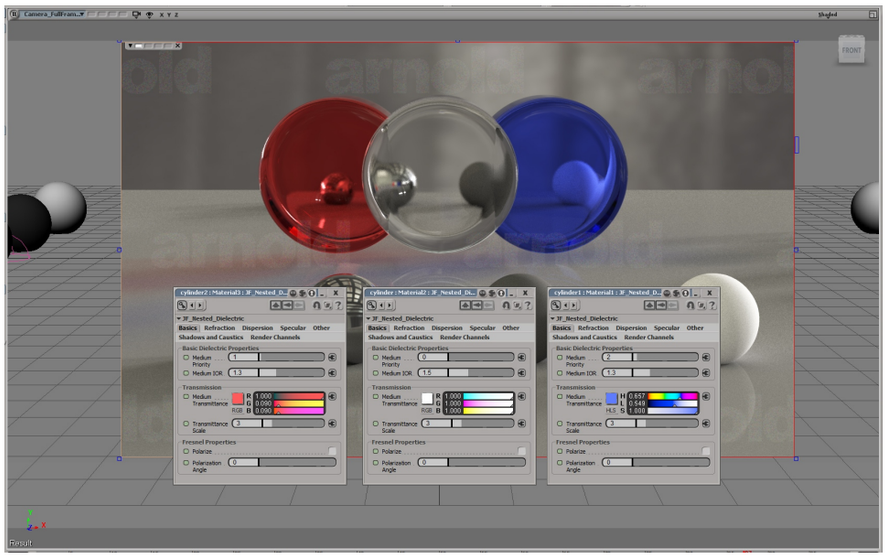
In order to evaluate the refractive and reflective properties of an interface between two materials correctly, you need to know the properties of both materials. Otherwise you will refract in the wrong direction, and reflect at the wrong power.

Imagine a glass of water. We have three media, and three interfaces. Air <--> Glass. Glass <---> Water, and Air <---> Water. When we trace the water surface at the top of the glass where it contacts air, we want very different material properties than when we're tracing the boundary between water and glass.

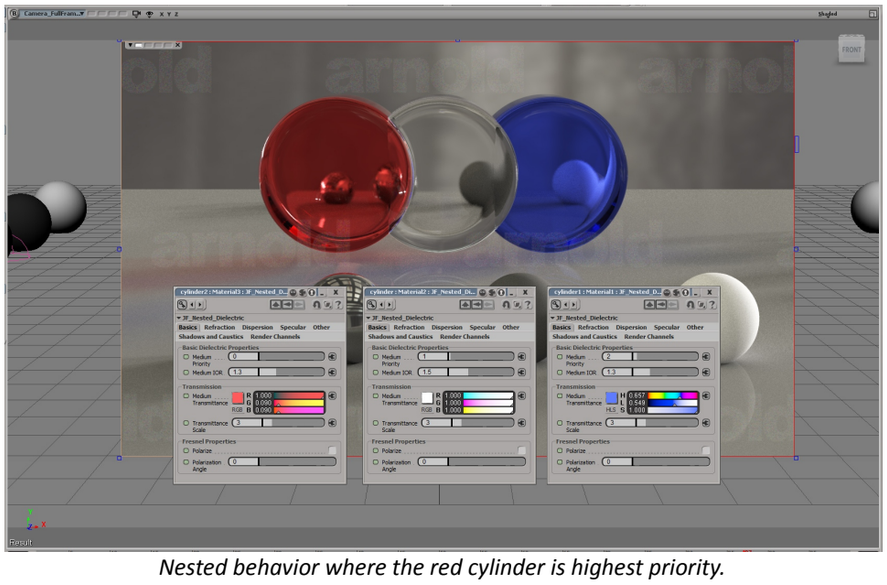
Demonstration of Nesting and Priority

Nested Dielectric works with priorities. If a space is occupied by two media, the ray tracing behaves as if only the higher priority one is occupying that space. The boundary between those two media traces correctly with both those material's properties.

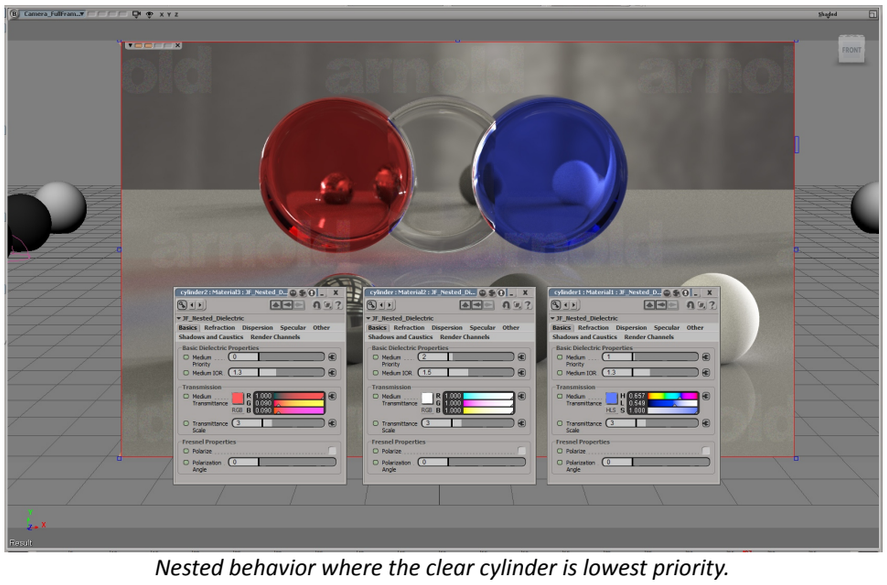
The models are the same in all three renders, only priorities are different. (Style of nesting demonstration copied from the original Nested Dielectric paper.)



Nested behavior where the central clear cylinder is the highest priority.



Nested behavior where the red cylinder is highest priority.



Nested behavior where the clear cylinder is lowest priority.

Nested Dielectric: Doing it right the easy way

Model every surface closed. Surfaces that are touching should be intersecting slightly. In a glass of water, the water should be slightly larger than the inside of the glass. These intersections will be resolved by priority.

How Priorities work: **Lowest number wins. Lower numbers are higher priority.**

Imagine an ice cube floating in a glass of water.

- Correct: Glass priority 0, ice is priority 1, water is priority 2:
  - We have a block of ice floating in water.
  - Inside the ice, the water surface doesn't exist.
  - This is what you want.
- Incorrect: Glass priority 0, ice is priority 2, water is priority 1:
  - Ice only exists above the surface of the water.
  - Underwater, ice does not exist.
  - This is quite strange.

A material being higher priority essentially means it wins when there is a conflict. If a ray finds itself inside both an ice cube and water, the ice should win, so the ice should be higher priority than the water. If it finds itself inside both a water droplet and glass, the glass should win.

Without Nested Dielectric: Doing it wrong the easy way

In our glass of water, we model it as above, with the water surface slightly larger than the glass it's in.

Problems: Interfaces between glass and water are wrong, either subtly or obviously. We have to trace two interfaces instead of 1:

The interface we want is: Glass -> Water.
The interfaces we get are: Air -> Water, Glass-> Air.

If both of those interfaces are both reflecting and refracting, that means the ray counts are doubling at every interface. Doubling our interfaces is more than a little inefficient. Turning off internal reflections helps, and then you've damaged the look.

Without Nested Dielectric: Doing it right the hard way

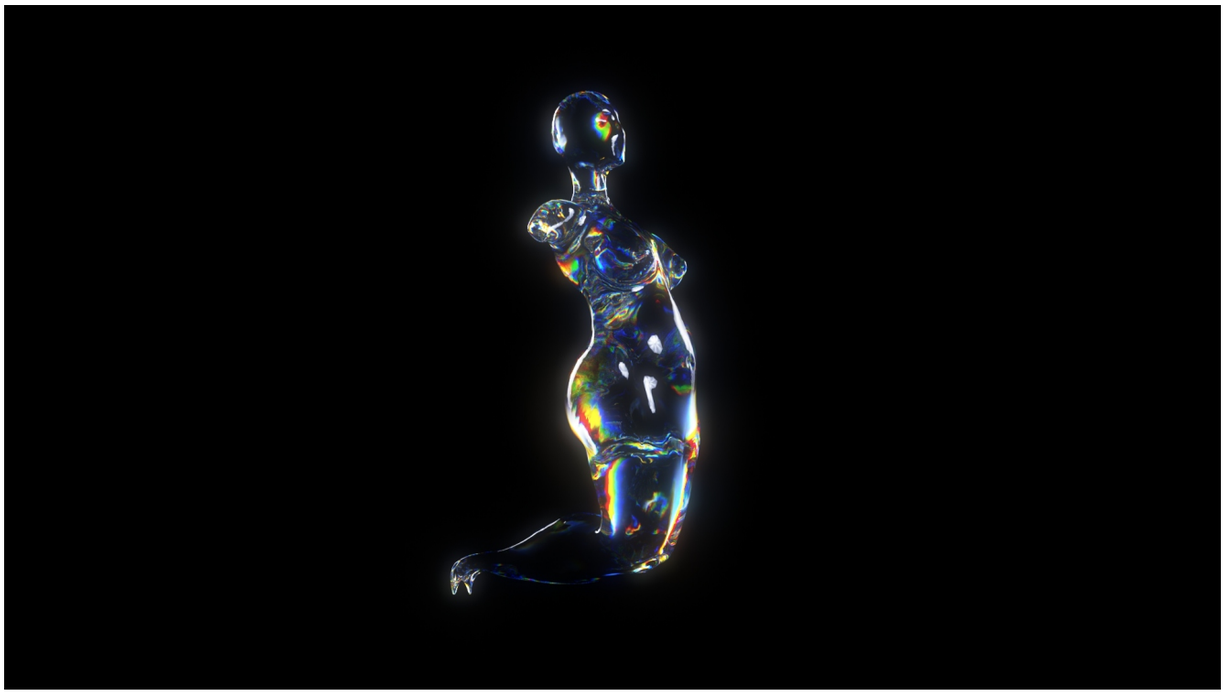
This would involve making three meshes instead of two. Instead of modeling the substances where water is a substance and so is glass, we instead model interfaces between substances.

1 - The glass until it reaches the water. The area of glass in contact with water is missing from this model.
2 - The water surface in the glass. It would look a bit like a flat disk.
3 - Boundary between glass and water. This would look like the missing piece of glass.

This will trace correctly with the right values on an ordinary shader. The glass-water boundary's IOR, if the normals are facing into the glass, should be Water IOR / Glass IOR, which would be below 1.0.

Problems: Computationally this is best but from a workflow perspective, this is a **disaster**. Water in a glass is about the most complicated nested dielectric scenario you can practically make. If you to go a step further and put ice in the water partially above the water line, you have to break up that surface too. If the water surface is animated, the glass needs to be too. Water drops on the glass would lead to dozens of tiny surfaces.

Dispersion



Model by Lauren Indovina.

Dispersion is a property of materials where the index of refraction is different at different wavelengths of light. This causes objects or lights in refractions to become separated into rainbow-like colors.

In JF Nested dielectric, dispersion works this way: When an object with the “disperse” setting on refracts for the first time, the refracted rays will each be assigned a random wavelength, between 370 and 780 nm. The wavelength chosen will influence the ray-tracing behavior from then on in the ray tree, changing the IORs used in those interfaces. Objects without the "disperse" setting on will also refract differently based on the wavelength of the ray, they just won't initiate dispersion on their own.

Disperse/Dispersion: The dispersion amount is the “spread” of IOR difference over wavelengths. For instance if dispersion is set to 0.1, and the base IOR is 1.5, the effective IOR will vary between 1.45 and 1.55.

Spectral Distributions: You can choose a spectral distribution from a list of presets. The spectra are defined as curves over the visible spectrum. Some are more continuous like daylight, and some more “spiky” like fluorescent. Daylight is the recommended distribution. It's modeled on the D65 curve. It's slightly easier to sample than the full spectrum, because it has less intensity at the fringes of the visible spectrum where the translated colors are very dim.

Gamuts: When wavelengths are converted back to colors, the main control for how this conversion is performed is the gamut. It should probably be set to the gamut of your working space. Rec.709 and sRGB share the same Gamut. Gamuts are ordered smallest to widest, with sRGB/R.709 being the smallest.

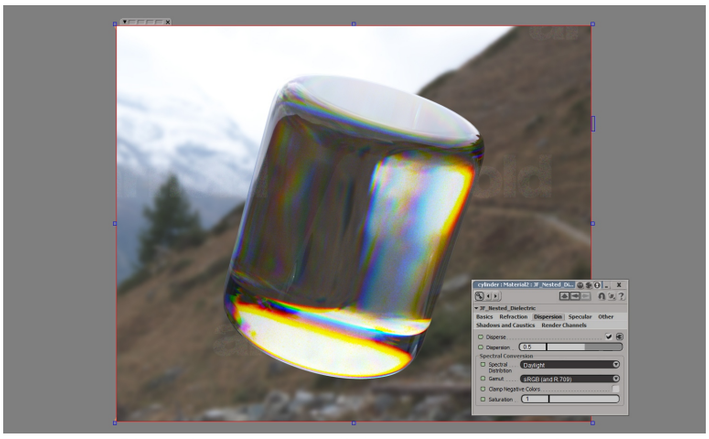
Negative colors: Colors generated this way can easily be way outside of the gamut. Colors outside of the gamut will have negative values. These negative values are not wrong- the colors in a dispersed rainbow will be so saturated that normal cameras can't render them properly and they may seem to burn into other colors. Even weak rainbows can seem to have extra saturation, and negative colors are the reason. Also, with negative colors you can actually convert the image back into a wider gamut and resolve the colors properly in Nuke (or whatever). However, negative colors can also simply be clamped if they cause problems, or noise.

Saturation: Decreasing saturation mixes the colors in the spectrum with white. This is non-physical. This mixing occurs before clamping negative colors, so decreasing saturation can mitigate negative colors as well.

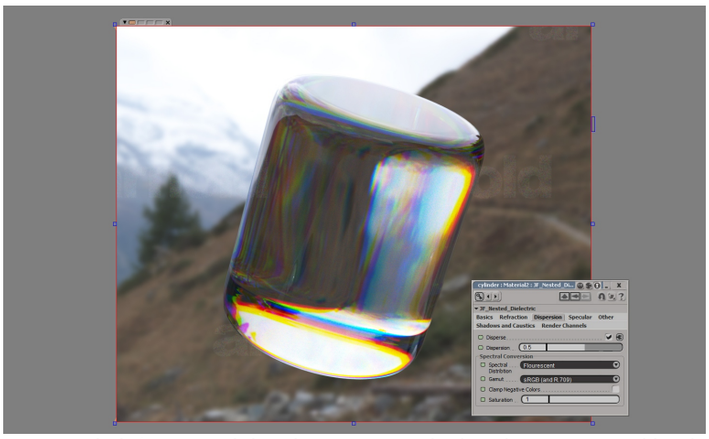
Technical description of Spectral LUT generation: Based on your choice of spectral distribution, an array of wavelengths is generated with wavelength samples in that distribution. Based on gamut and other settings about how to convert wavelengths to colors, an accompanying array of colors is generated. Together these two arrays are a “Spectral LUT”. When dispersing, the sampler chooses values from the Spectral LUT.

Spectral Distributions

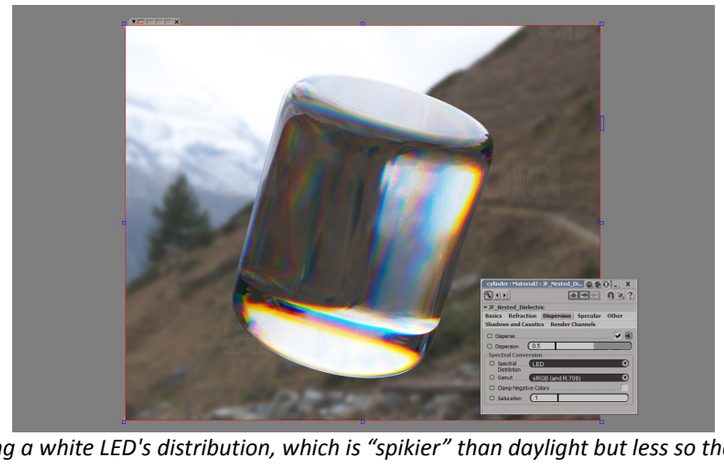
(Exagerated amount of dispersion)



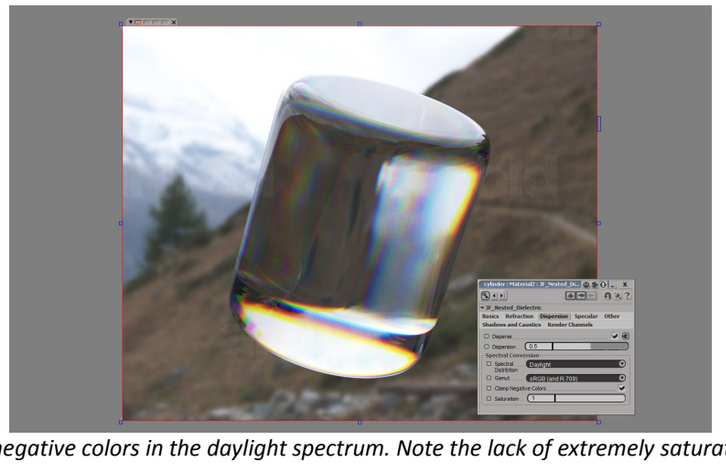
Dispersion using a daylight spectral distribution.



Dispersion using a fluorescent light's spectral distribution. Note the banding generated by the “spiky” distribution.

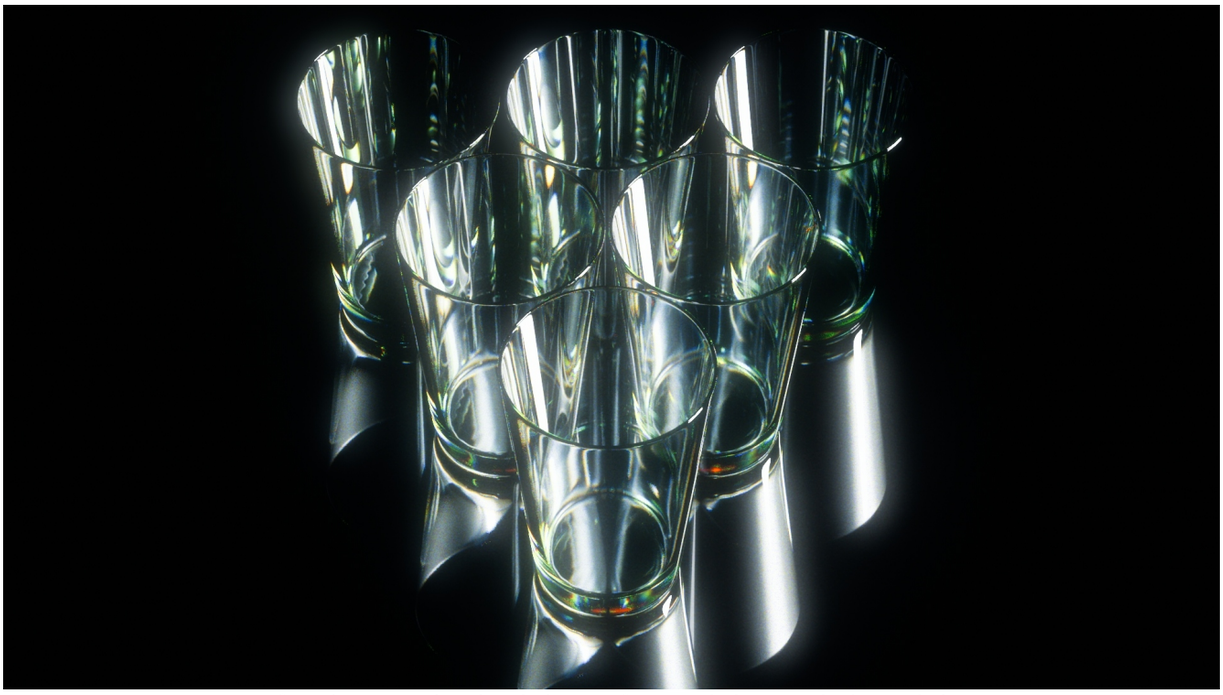


Dispersion using a white LED's distribution, which is “spikier” than daylight but less so than fluorescent.



Clamping negative colors in the daylight spectrum. Note the lack of extremely saturated colors.

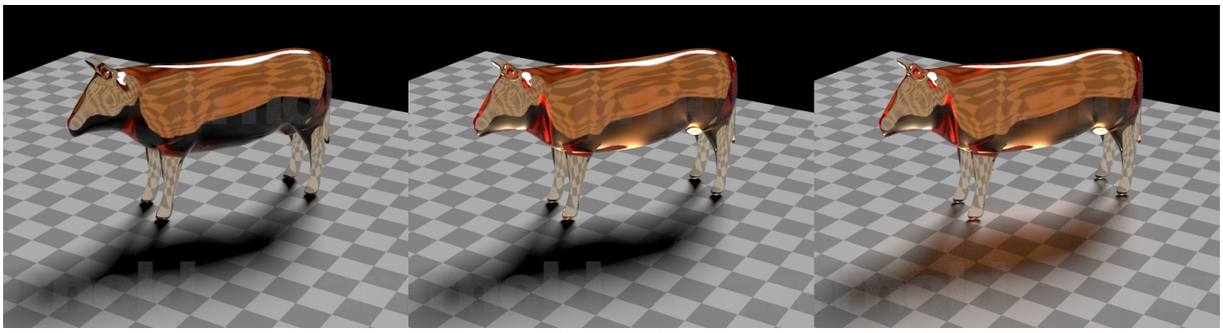
Direct Refraction and “BTDF”s



A few glasses rendered with direct refraction, lit by a single distant light.

Direct Refraction is refraction of lights. In ordinary refractive shaders you can only refract geometry and environments.

Blurry refraction of lights is accomplished by using the specular BRDFs native within Arnold to render both blurry refraction and refraction of lights. Using direct refraction becomes more tractable in a nested shader, because the book-keeping necessary to tell the shader what media it's inside also allows it to know when it's leaving all media, at which point direct refraction is performed.



From left to right: No direct refraction, with direct refraction, and with direct-refracted caustics.

Direct refraction also works with Arnold's skydome light. Generally speaking, using direct refraction of skydome lights instead of indirect refractions of the sky shader is only better when the refraction is quite blurry. The shader allows you to choose whether you want to use skies (environment shaders) or skydomes with two check boxes. Note that turning off refraction of skies in the shader will also disable volume shaders from being visible in refraction.

Depending on the lighting scenario, direct refraction settings often need tweaking to look good. Sharp refractions of small, hot light sources (such as the sun) can easily create very hot value, so it's very often desirable to embellish the roughness of refraction specifically for direct refraction.

There are somewhat complex roughness controls for direct refraction.

- Use Refraction BRDF
  - Use the same BRDF that ordinary indirect refraction is using.
  - Otherwise, a completely different BRDF can be specified.
- Roughness Offset
  - Adds this value to the roughness of the BTDF

- Ray Depth based roughness modification
  - Allows you to multiply or add roughness to the direct refraction, depending on the ray depth, so that deeper ray paths use rougher direct refraction.
- Second Lobe
  - Allows a second specular lobe for direct refraction, which can have a higher roughness to create a “glow” effect around the direct refraction, such as haziness or fingerprints on glass.

Direct refraction can also be used for ray-traced caustics. This won't work for sharp caustics because the sampling required is infeasible, but can work for blurry caustics.

Direct refraction of a skydome light with an HDR containing an unclamped sun.

BTDFs in JF Nested Dielectric are actually Arnold's native BRDFs, being used for blurry refraction. This allows for oddities such as Ward-Deur anisotropic refraction, which could be used to model something like “brushed glass”.

From left to right: Cook-Torrance, Refraction derived Ward in U, Refraction derived Ward in V, Ward with user tangents.

There are four “BTDF” options:

- Stretched Phong
- Cook-Torrance (note that this is not the micro-facet BTDF, but the specular BRDF re-purposed)
- Ward-Deur with refraction derived tangents
  - This allows you to control blurring of the refraction along or against the refracted angle.
- Ward-Deur with user tangents
  - You will need to supply tangents to the shader for this to work, in the “ward_tangent” port.

Blur Anisotropic Poles blurs the roughness of U and V of the shader when the normal is pointing along the same vector used to generate the tangents. In the example images above, it's being used in the Ward (refraction derived) in V image as otherwise the pole would be quite conspicuous.

Technical Topics

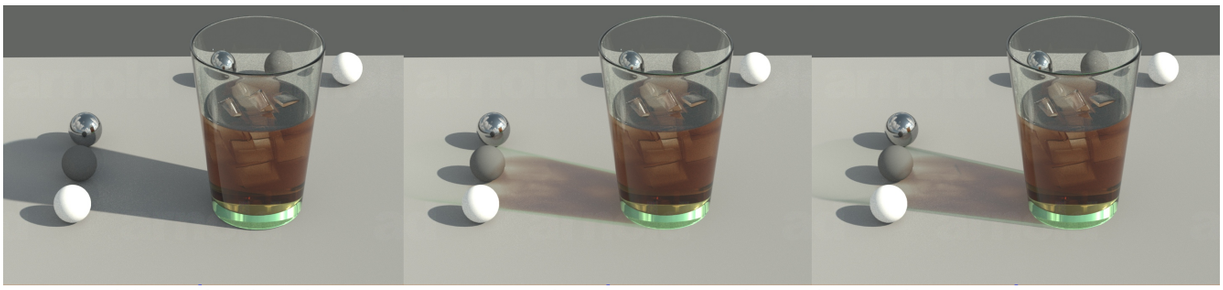
Interfaces, entrance and exits:

Valid Interfaces: Any time ray tracing encounters a interface, it may or may not be valid. A valid interface reflect and/or refract, an invalid one will allow the ray to pass without modifying its direction. Entrances means the ray is entering into a medium for the first time. Exits mean that it is leaving all media for open air.

There are three emission ports, which another shader can be plugged into. This can be used to add other shading effects.

- Emission At Interfaces: Emits/Evaluates at valid interfaces (not at invalid ones)
- Emit At Exits: Emits/Evaluates when the ray is leaving all media from open air.
- Emit at Entrances: Emits/Evaluates only when the ray is entering into media from open air.

Shadows:



From left to right: Black shadows, Transmission only, Transmission with outside Fresnels. No caustics are present.

For rendering with transparent shadows, multiple shadow modes are available. Note that the Arnold opacity flag must be off in order for transparent shadows to be rendered.

- Black Shadows: No light passes through. If caustics were rendered to fill in the shadows, this would be the physical option. Otherwise, transparent shadows could be thought of as a way to fake caustics.
- Transmission Only: Only the transmission color will be in the shadows.
- Transmission and outside Fresnels: Default and recommended setting for most cases. The shadow darkens on the edges where the material is more reflective and less refractive.
- Transmission and all Fresnels: Shadow darkens with interior Fresnels as well. This tends to cause visible artifacts, especially where total internal reflection occurs.

Polarizing Filter :

From left to right: No polarization, vertical polarizer, horizontal polarizer. Note the top and sides of the sphere.

The polarizer in this shader is meant to mimic a polarizing lens filter. By its nature it's very incomplete, because to work totally correctly it would also need polarized light sources, as well as tracking of polarization through the ray tree. This is a much simplified version.

There are actually two Fresnel equations, one for S polarized light and one for P polarized light, where one equation is for polarization in alignment with the surface, and the other is rotated 90 degrees

from that. For un-polarized light, the correct way to use the Fresnel equations is to assume an equal amount of P-polarized and S-polarized light, and simply use an average of the two equation's results.

To use a polarizing lens filter that's aligned vertically with the camera, we blend from the S polarized result to the P polarized result, using the alignment with the “filter” with the surface normal to do the blend. You can also rotate the filter.

This can somewhat mimic what happens when you use a polarized lens filter to, for instance, dramatically reduce reflections on water. However in the real world when photographing the ocean, the sky light is also polarized, causing a much more dramatic effect than we see with the shader, which must assume that all light is un-polarized.

Caustics:

At the moment only for the path traced caustics are available. Generally path-traced caustics are infeasible in production due to noise issues, but there is one exception: Caustics from direct refraction are sample-able enough to be feasible in production in many cases, to produce blurry caustics (not sharp ones).

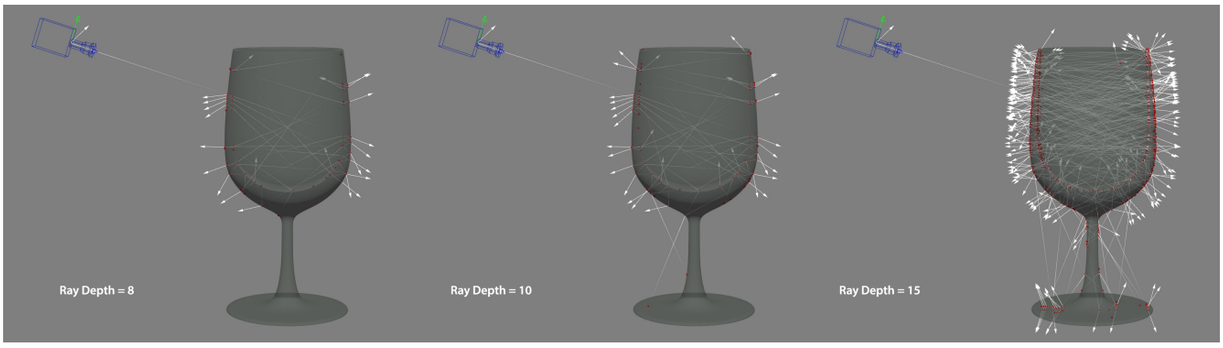
There is also a distance control. If the max distance is 10, then the caustics linearly fall off to have zero intensity beyond 10 units. Max distance set to -1 will disable the falloff.

Specular Ray Type:

The shader allows you to specify whether you'd like to use the “Reflected” or “Glossy” ray type for indirect speculars. This effects ray depth, and behavior with ray switches, such as the ray switch built into the sky shader.

If the rest of your scene is using glossy rays, and you use reflected rays, you can set a different ray depth for the reflections inside the glass than the ray depth of the glossy reflections in the scene. This is useful especially with sharp reflections inside glass, as you may only want one glossy bounce in the scene in general, but many more for internal reflections inside your glass.

Optimization



Visualization of the horror of unchecked ray tree growth, generated from one AA sample.

Why rendering dielectrics requires aggressive optimization techniques: By their nature, dielectrics cause branching ray trees. At most interfaces, the ray should both reflect and refract. That means that as ray depth increases, the number of rays increases exponentially. It's also hard to figure out ahead of time which ray paths will be important and which ones won't. For example, a weak reflection of a weak reflection will be a very weak reflection, and it may seem like it can't be very important. But it can be, if that ray happens to point at a hot light source, like the sun. If that's the case, a super deep ray path would appear in the render as a glint in the glass, a desirable property.

Disabling Internal Reflections is detrimental to the look of the render, but will cut down on the exponential growth of the ray trees drastically. Total internal reflection is not effected by this, because TIR by its nature reflects totally and does not cause branching in the ray tree.

Russian Roulette is an unbiased but noise-producing optimization technique. In this shader, it allows you to set a probability of reflection. If the probably is 33%, the intensity of the reflected ray will be 3x as strong, but only take place 33% of the time. This produces noise, especially in very deep ray paths, but has a huge effect on render times. With RR set to 50%, your ray count increases with 1.5 ^ (ray depth), instead of 2 ^ (ray depth), which is a huge difference with higher depths. With ray depth of 10, you can get 2^10 = 1024 rays from each AA sample. With Russian Roulette set to 0.5, this maximum number you should expect is 1.5^10 = ~ 58 rays.

Russian Roulette also allows you to render with higher AA samples with the same number of total rays, which can be very advantageous for glass renders as they often produce very small details such as glints or bright edges that can't be resolved with low AA samples.

Russian Roulette is disabled if probability <= 0 or probability >= 1.0.

Energy cutoff is a biased but noise-free optimization. The cumulative reflection, refraction, and transmission intensities are tracked through the generated ray tree, allowing culling of ray paths whose cumulative power falls below a very low threshold.

For example, if you have a weak reflection of a weak reflection of the sky, and the weak reflections both have an intensity of 0.05, the cumulative intensity of the reflection-of-reflection is 0.05 * 0.05 = 0.0025. When ray trees get deep, you can very easily get to very small fractions of the sample's intensity being transmitted back to the AA sample. Since the values we're dealing with are so small, the parameter is set as as an exponent of ten. The default value of -5, for instance, is 10^-5 = 0.00001.

These tiny fractions may matter though- if such a weak reflection finds the sun, it might be visible as a glint inside the glass. If you'd like to have no visible difference in your render, consider 0.002 could be a rule of thumb for the smallest pixel value that might matter. Given that, a rule of thumb for choosing a safe clamping threshold is determine the brightest thing that will be refracting, and then take (0.002 / brightest value). For an un-clamped sun with a value of 20,000, this value would be 0.000005, or about 10^-7.

License

This documentation and images in it are Copyright (c) 2014, Psyop Media Company, LLC and Jonah Friedman.

The binaries and source code for JF nested dielectric are released open sourced, under the BSD 3-clause license:

```
Copyright (c) 2014, Psyop Media Company, LLC and Jonah Friedman

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.
    * Neither the name of the <organization> nor the
      names of its contributors may be used to endorse or promote products
      derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```