Everything You Always Wanted to Know About `mia_material*` (* But Were Afraid to Ask)

by Håkan "Zap" Andersson

Figure 1: Catch the Dream

Abstract

As the original author of the popular *mental ray* shader `mia_material`[1], I will herein discuss its background, development history, features, and even its failures.

Today's topic: `mia_material`

`mia_material` is probably one of the most widespread physically plausible shaders in existence. As of today, is is integrated (in some form or another) into a large percentage of *Autodesk* products—including 10+ million *AutoCAD* installations—and it is the basis of the standardized *Autodesk materials*. It has been used in everything from spoon commercials and game cinematics to high-end feature films, and it has inspired many emulations, simulations and deconstructions from partners and competitors alike.

[1] Also known as the **Arch & Design Material** in certain product integrations.

Background: what is this thing anyway, and why was it made?

Traditionally, computer graphics software has been littered with very ad hoc shading models. In particular, DCC (Digital Content Creation) tools have had a tendency to throw "everything and the kitchen sink" at its poor users, leaving them very frustrated as they attempted to figure out how to glue things together and get a visually correct result.

It didn't help that a lot of this software had no concept of energy conservation, so creating a shader that emitted 500% of the incoming energy was all too easy to do—there were simply no safeguards. This, compounded by the interesting psycho-optical limitations of *how we see things*[2], made it surprisingly difficult to fine tune the rendering "by eye". To make matters worse, many DCC applications had color management (and hence gamma correction) turned *off* by default, which meant that the poor user had to try to light in a non-linear color space. Taken together, you can appreciate what an utter mess the business was in!

The Time, the Place, the People

The development of a solution to this problem started in 2005, as a proposal by *mental images*[3] for a new, easy-to-use physically plausible shading in *mental ray*, primarily targeting *Autodesk*'s 3ds Max DCC platform. The design was principly done by *mental images* but in strong cooperation with *Autodesk*, and was iterated over many months. Development was done on the *mental images* side[4] but weekly design iterations were done with the rendering development and product design leads at *Autodesk*[5].

So, who is "Mia"?

The actual working name of the shader was "The X material"[6], but as it was later folded into the larger "architectural" shading library, *mental images* shader naming convention dictated that it got stuck with the name `mia_material`, where "mi" stands for *mental images* and "a" for architectural.

Design Goals: what were we trying to achieve?

From a high-level perspective, the idea was simply to achieve good-looking and physically plausible shading with minimal effort. The user, starting with the default settings, should be able to get a pretty picture right "out of the box"[7]. For this reason, the aforementioned architectural shader library contained other shaders for things such as physical sun- and sky-lighting, exposure control, tone mapping, etc., that we will not delve deeper into here.

Some key design decisions were:

- **Conceptually *layering***: The user should be able to think about the shader as if *layering* things together, such as a clear coat on top of a diffuse layer on top of some kind of transparency.
- **Energy conservation**: No more "let's have 100% diffuse with 50% glossy reflection and 80% transparency".

[2] For more on this topic, see the `mry201` course, class #2, available at `fxphd.com`.
[3] *mental images GmbH* in Berlin is now known as the *NVIDIA ARC* (Advanced Rendering Center).
[4] Me (Zap Andersson), overseen by Matthias Senz.
[5] Pierre-Felix Breton and Daniel Levesque.
[6] To this day, the original name is reflected in some of the names in the source code.
[7] Assuming that they also used a physically plausible lighting scenario, a proper linear color space, etc.

- **Reflections and specular highlights are linked**: Most shaders of this era treated traditional Phong-style specular highlights and glossy reflections as completely disparate sets of knobs, wholly unrelated unless the user linked them manually. Reality doesn't work that way!

- **Fresnel everywhere**: Most reflectivity effects in reality follow an angle dependent curve. However, our human visual subsystem is specifically designed to tune this effect out. This makes it very difficult to intuit that reflections should depend on a Fresnel curve.

- **Easy parameters**: Visually understandable parameters in a range that can be texture mapped (i.e., *glossiness* in a 0-1 range rather than *roughness* going from 0 to infinity, or exponents going into the thousands.)

- **It has to look good**: Above all, images should look good. So visual *beauty* was actually valued higher than strict academic *accuracy*. For example, most of the shading models existing at the time were rejected simply based on a dislike of their visual appearance.

- **Special options**: Some specific options were deemed important, such as:

  - **Thin versus thick material**: treat the boundary either as an entry/exit surface to a volume, or an infinitely thin "soap bubble"-like shell.
  - **Cutouts**: removing parts of a surface with a texture map. For example, to turn a rectangle into a leaf.
  - **Metal mode**: metal reflects its color, dielectrics do not. This has to be an easy "switch".

- **Easy to use**: Things that were difficult to set up or do in *mental ray* had to be automatic.

  - **Photon and shadow shaders**: These work differently from surface shaders, and normally need special care to set up properly. In `mia_material`, they are simply built in and automatic.

- **(Reasonably) accurate**: the user should be able to input real-world values should get reasonably real-world results.

- **Performance**: finally, it had to be fast. At the time, most shaders that did something as "exotic" as *glossy reflections* caused ray counts to explode exponentially and quite literally ground your rendering to a halt.

The Shader: its features and behavior

The full documentation of the shader and all of its parameters is beyond the scope of this document, but a good reference can be found here: `http://download.autodesk.com/us/maya/2009help/mr/shaders/architectural/arch_mtl.html`

Material Layers

The material consists of four basic *layers*, and these are conceptually combined "on top" of each other as follows:

- Reflections / Coating
- Diffuse
- Refractions (referred to as *transparency* from this point on)
- Translucency

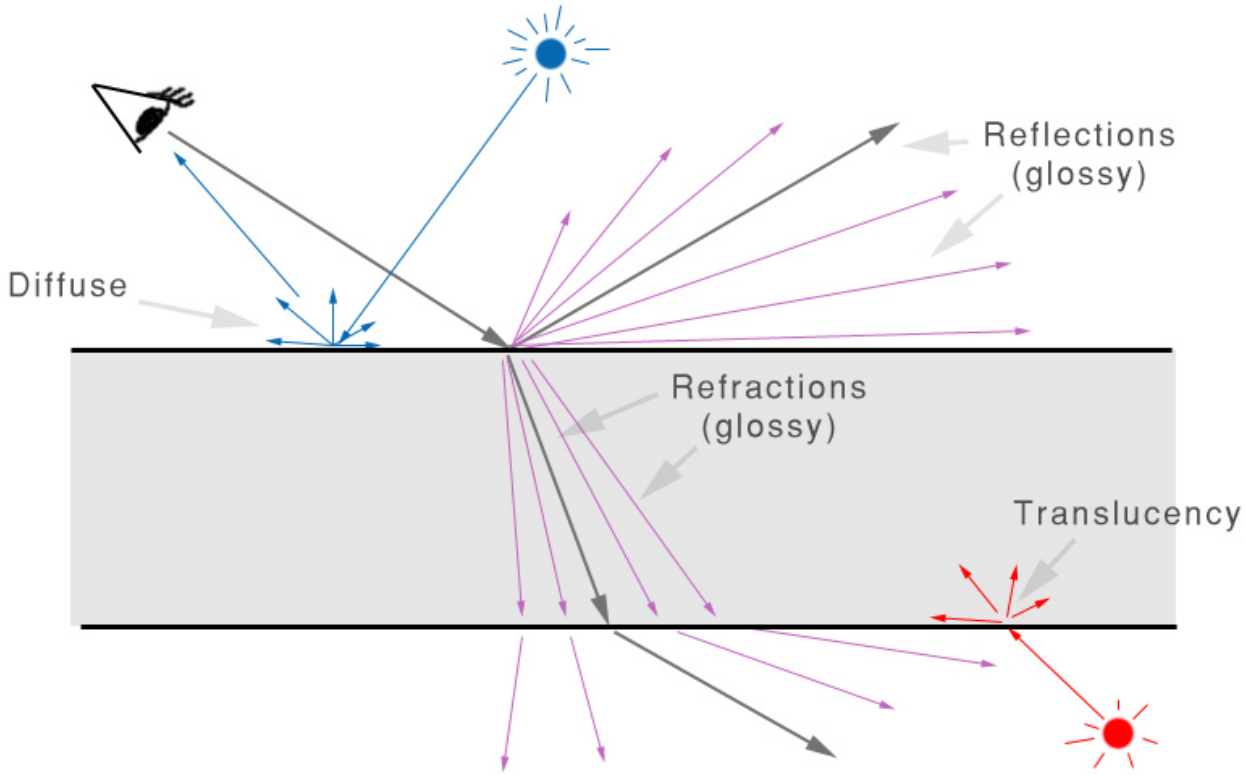

Figure 2: The various layers

This ordering becomes relevant for the energy conservation algorithm. For instance, since the reflection layer is conceptually on top, it "steals" energy[8] from anything below it.

[8] This is covered in more detail in the Energy Conservation section.

Angle-Dependent Reflectivity

Furthermore, the *reflection* layer has an always-on angle-dependent component, where reflectivity increases at increasingly grazing angles of incidence.

Figure 3: Angle-dependent reflectivity.

This feature can be switched between a strict Fresnel function (driven by the index of refraction) and a "hand tuned" curve, where one gives as inputs the reflectivity at normal incidence (R0), the reflectivity at grazing incidence (R90), and a curve exponent (E). Here is an example of the user interface in 3ds max:

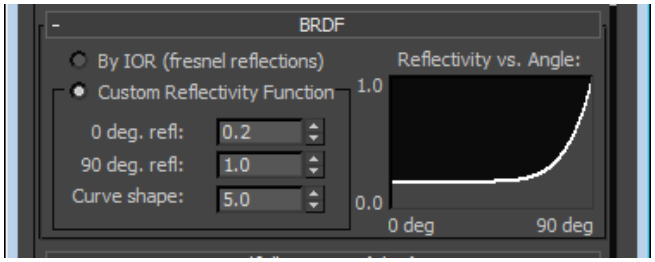

Figure 4: 3ds max UI example.

The actual reflectivity is modulated like this pseudocode snippet:

```
float dot_nv = dot(normal, eye);
float expfac = pow(1.0 - dot_nv, E);
float refl = R0 * (1.0 - expfac) + R90 * expfac;
```

Energy Conservation

Energy conservation follows by a fairly simple set of rules:

- Reflection takes energy from everything below it
- Transparency takes energy from diffuse
- Translucency is considered to be a "kind of" transparency, and therefore shares remaining energy with it[9]

In practice, when layer A "takes energy from" layer B, it means that layer B's contribution is multiplied by 1 minus the *contribution* of layer A. To avoid color shifting, this is done equally per channel, so the contribution is based on the luminance-weighted R, G, B components. Therefore, a red reflection does not just remove "red energy" (which would, to the user's surprise, tint underlying layers cyan). Instead, it subtracts the same amount from all channels. The value used is the amount of red multiplied by the luminance weight of the red channel (for the current color space). This is technically *wrong* but since most of the energy loss comes from the reflectivity layer—which is generally white—we rarely see problems. In pseudocode form, this amounts to:

```
float3 reflect = apply_view_dependent_curve(<reflectivity>)
float3 diffuse = <diffuse>;
float3 refract = <transparency>;
float3 transl  = <translucency>;
float  orig_transl = luminance(transl);

// Energy conservation math
diffuse *= (1 - luminance(refract)); // Reduce by transparency

transp  *= (1 - luminance(reflect)); // Reduce by reflectivity
diffuse *= (1 - luminance(reflect)); // Reduce by reflectivity
transl  *= refract;                  // Scale by transparency
refract *= (1 - orig_transl);        // Compensate by reducing by orig. translucency
```

At this point the variables **diffuse**, **reflect**, **refract** and **transl** contain the absolute weight of each of the components. Slightly more complex rules are used if the *metal* mode is enabled; reflectivity color is multiplied by the diffuse color, but the energy conservation operates as if reflectivity had retained its original color.

[9] This is admittedly a bit inconsistent and was caused by translucency being added later.

Shading Model

As mentioned earlier, visual quality was the main decider of the shading model of the various layers. A lot of work went into making these look nice, but also execute efficiently. The efficiency work is beyond the scope of this document[10], and probably irrelevant to most modern rendering, because it was done in ways particular to *mental ray* and particular to the fact that it had to be done *in the shaders*[11].

Transparency

The transparency layer just used standard *mental ray* glossy transparency functions[12] with no special sauce at all.

In theory there is probably a similar "problem" with transparency as we will be discussing in a moment with reflections, but transparency is hard for our brains to wrap themselves around. Even early research at Pixar shows that our eyes accept almost anything that is roughly the right color and distorted as "refraction", so no particular deep thinking went into this part of the shading at all[13].

Translucency

The translucency layer is even more simple. It is really intended to be a cheap fake for things like translucent leaves and similar, without having to do full sub-surface scattering and other complex shading effects. It is implemented simply as computing a standard Lambertian diffuse on the opposite side of the surface.

Diffuse

The diffuse model is Oren-Nayar, but with a twist. The visual problem we saw with a standard Lambertian is a harsh look of the terminator. This becomes especially evident when rendering in a proper linear color space[14]. The reason for this is simply the large discontinuity in the shading.

[10] Careful importance-driven ray-tree management and whatnot.

[11] Being located outside the Berlin office, I never actually had access to the closely guarded *mental ray* source code. It was locked in a vault not unlike the one seen in "Terminator 2".

[12] mi_transmission_dir_anisglossy_x() to be precice.

[13] Actually I've seen refraction in real life that doesn't look realistic *at all*!

[14] Sadly, we live in a world where a large amount of people think a Lambertian surface actually looks like what you see when displaying it with no gamma correction straight onto an sRGB display. I invite such people to go into a dark closet with a flashlight and a ping-pong ball, and if they don't change their views, well, at least we have them safely locked away in the closet.

Our eyes are edge-detectors and do this by detecting the second derivative (the change in the rate of change) of luminance. Since a Lambertian is just an abruptly cut off cosine, we perceive this as an edge. The problem is especially evident in point-lit scenes[15] and when the terminator is seen against a slightly different color:

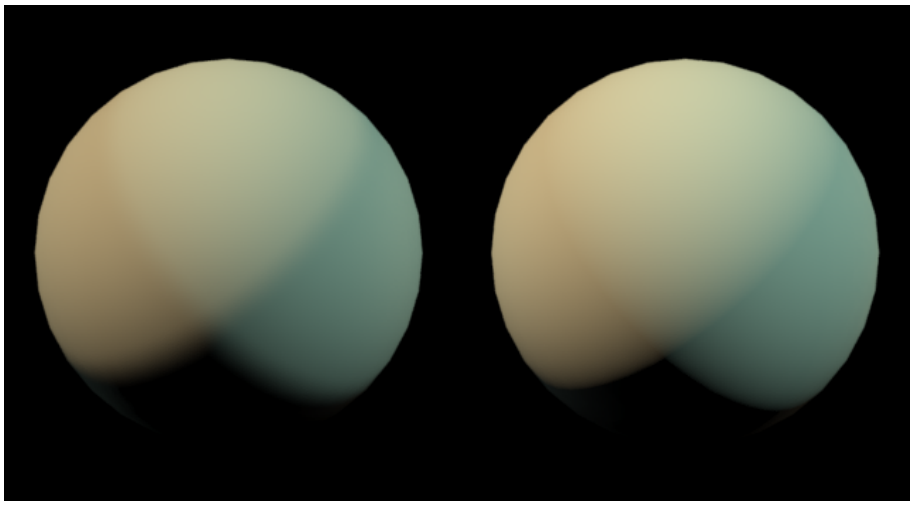

Figure 5: A `mia_material` diffuse (left) compared to a standard Lambert (right) under two-colored light.

The image above shows the (admittely very subtle) difference. On the right, the terminator regions appear *harsher* and reads more like lines to the eye. On the left, we see the `mia_material` version which smooths this out some.

The implementation is simply applying a smoothstep() function to the very lowest 25% region of the cosine, to smooth out the discontinuity some. It isn't entirely successful in an artificial scene such as the above, but does cover up the problem in most real-world scenes.

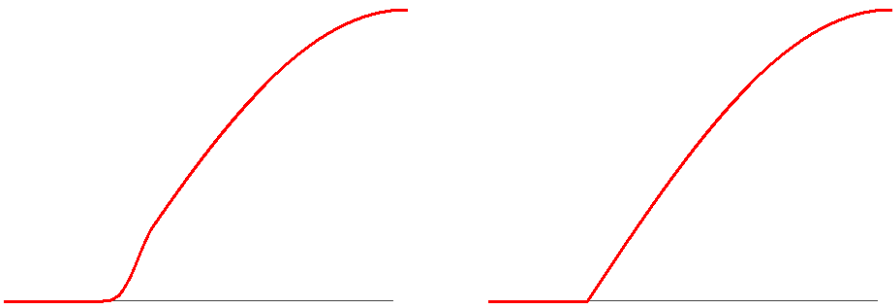

Figure 6: Cosine with a smoothstep (left) compared to a standard cosine (right).

[15] Area-lighting alleviates this problem quite a bit, but it is still there.

Reflections

With the Reflection layer, we found that many of the available models were grounded heavily in *theory*—such as a microfacet model, or similar—but that many of them failed the basic test of *looking good*. In essence, the "look" of glossy reflections and specular highlights of many available shading models simply did not satisfy us visually.

Two visual problems were especially evident: bad highlights, and poor "stretchiness" of reflections:

Bad Highlights

For most shading models, specular highlights looked too much like a Gaussian—i.e., a "fuzzy blob"—and didn't appear visually "glowy" enough. At the time (2005), very little available research existed to explain the difference; we simply relied on experience of what looked realistic.

Figure 7: This image shows an `mia_material` highlight (left) compared to a traditional Blinn-style highlight (right).

Notice how the `mia_material` highlight has a softer "spread", whereas the Blinn model fades out much more rapidly. No amount of tweaking of the Blinn shader's exponent will match the appearance of `mia_material`.

People have historically resorted to cheap tricks to combat this, such as combining multiple specular functions with different exponents; this is in fact the same trick utilized by `mia_material`. The highlight is a layering of multiple Ward highlights[16], where the base Ward exponent is computed using this formula:

```
base_exponent = pow(2.0, glossiness * 8.0);
```

Then, three highlights are created using `base_exponent`, `base_exponent/2.0`, and `base_exponent/4.0` and blended with weights 1/6, 1/3 and 1/2, respectively.

[16] Using *mental ray* API function mi_ward_anisglossy()

Reflection "Stretchiness"

A second issue with many existing shading models is how they fail to recreate the "stretchiness" of reflections to the extent that reality seems to do. As an example of this, let us consider a sunset over a lake. Why does the Sun's reflection turn into this vertical streak, rather than just a fuzzy highlight?

Figure 8: A stretchy highlight.

The answer to this question is simple: the normal varies across the lake surface effectively randomly with the many tiny waves on the surface. But what happens to the reflection direction as the normal varies on a horizontal surface such as this water?

- If the normal is varying *away* from or *towards* the viewer, the reflection direction varies *massively* up or down the sky.
- If the normal is varying *side to side*, it hardly affects the reflection direction *at all*.

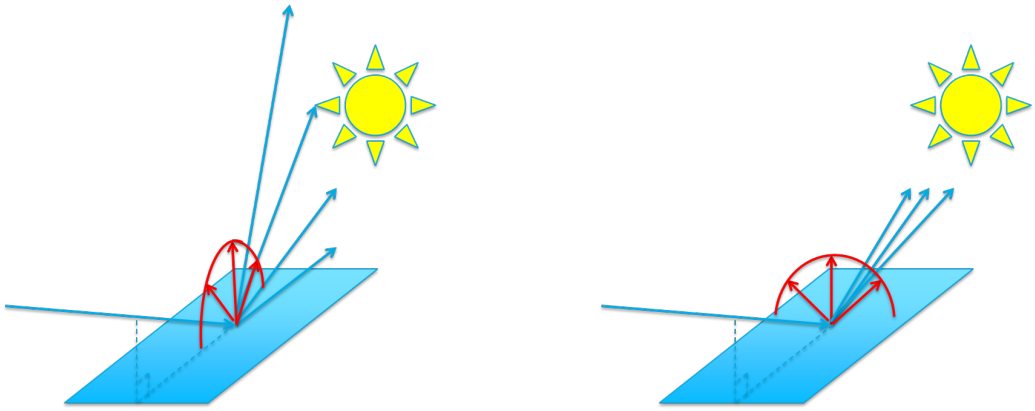

Figure 9: How reflection rays change as the normal vector changes.

This makes the reflection "lobe" extremely stretched out in the visually perpendicular direction to the surface's median normal. This effect seems to be under-represented in many available shading models - even the microfacet ones which should, in theory, represent this well.

The `mia_material` Stretchiness Solution

Most available helper functions in the *mental ray* shader API for generating rays for glossy reflections work by varying the reflection direction randomly. But as we discovered above, this does not yield visually interesting reflections at all; a uniform cone of glossy reflection rays is exactly what one does not want. `mia_material` solves this with some creative coding[17], "abusing" the *mental ray* API by using the functions intended for generating Quasi-Monte Carlo (QMC) stratified bunches of glossy reflection rays[18] to instead generate bunches of *modified normals*, and then recomputing the reflection ray for each new normal.

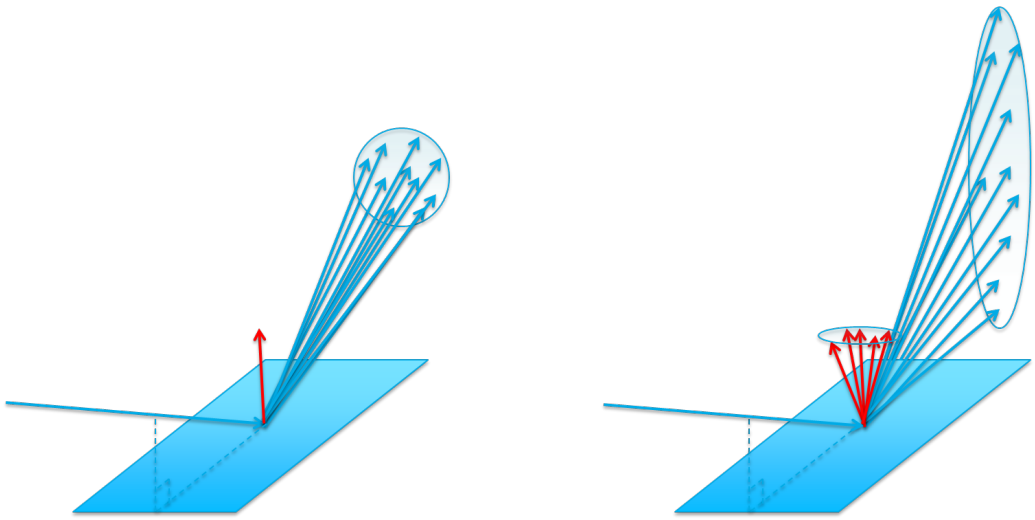

Figure 10: Jittering reflection direction vs. jittering normals.

When computing a reflection direction based on a shading normal that is very different to the

[17] "I'm not a mathematician, I'm a magician" - Me

[18] mi_reflection_dir_anisglossy_x()

geometry normal, there is always a risk of the reflection direction ending up below the plane, causing a self-intersection with the surface if one tries to trace that ray. Most renderers reject such rays, or treat them as black, which is often a cause of dark edges in glossy reflection models. In contrast, `mia_material` simply flips such rays to be above the plane instead, which seems to work well in practice.

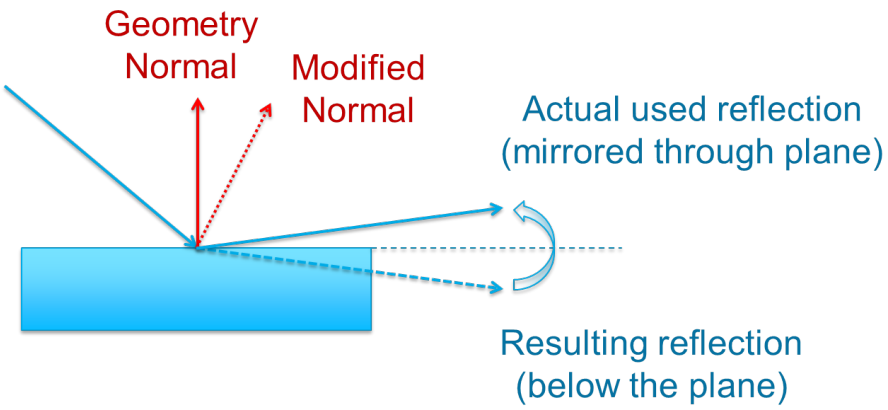

Figure 11: Solving reflection rays that go below the plane.

A visual comparison of the two methods:

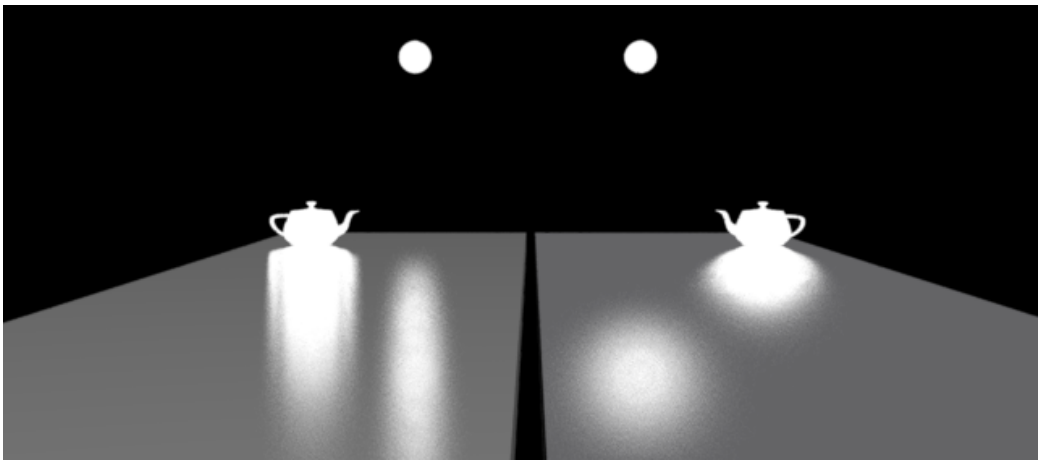

Figure 12: In this image, we see `mia_material` reflections on the left, and standard *mental ray* Ward reflections on the right.

The reflection of the sphere is properly stretched out on the left, but is just a fuzzy, vaguely Gaussian blob on the right. The reflection of the teapot on the right just looks *strange*.

However, all of the above caused one of the biggest problems with the shader, as I will discuss in the following section.

Failure is Always an Option

Life is not always optimal, and there are a couple of known issues with `mia_material`.

Highlights versus Reflections

A major flaw in `mia_material` is the simple fact that reflections and specular highlights do not match! They may be extremely beautiful individually, but they are supposed to express the same math in two different ways. Alas, they do not[19]:

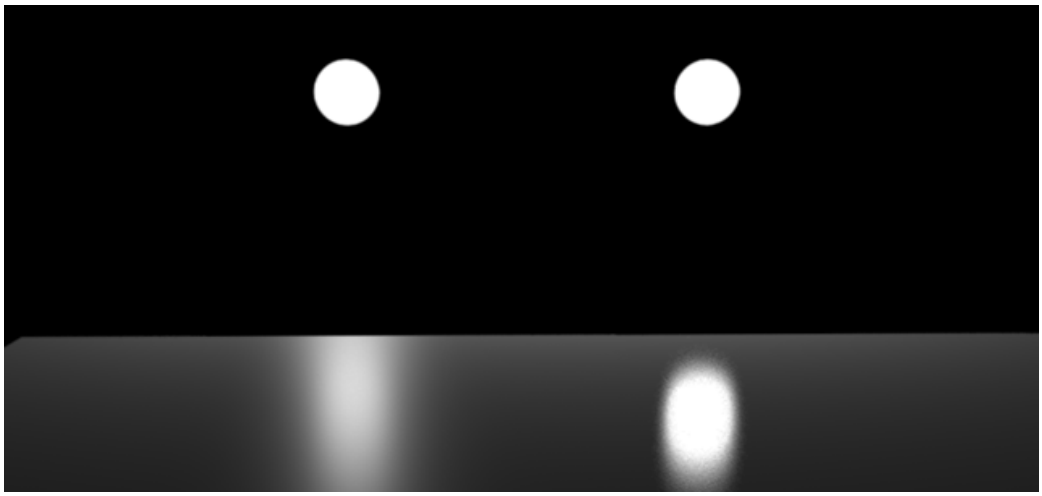

Figure 13: Specular highlight (left) vs. glossy reflection (right) at glossiness=0.54.

There are several reasons for this discrepancy:

- Highlights use the a sum of three Ward highlights with three different levels of glossiness
- Reflections only use a single function with a single glossiness, not three
- The unorthodox method to compute the reflection ray directions ends up being a different distribution than the highlights[20]

This has to be *the* issue to look out for if one is trying to match `mia_material` in some other renderer. Without cloning this behavior, it will be practically impossible to get an identical "look".

Conservation Math is "Wrong"

The Energy conservation math uses a slightly bizarre algorithm when a layer takes energy from another layer. As mentioned previously, it bases the reduction in energy on the luminance-weighted color value.

Without doing this, if one has a 100% red reflection layer there would be zero energy left for diffuse reflections, since 100% has already been taken. Or alternatively, it would only reduce the energy in the red channel, causing a cyan tint of the underlying layers. We felt either of those results would feel confusing to the user. Instead, `mia_material` considers the 100% red to be only 21%[21] of the total energy, and the remaining 79% can be used for diffuse.

[19] As I said, "I'm not a Mathematician, I'm a Magician"

[20] If it is any consolation, had the standard *mental ray* API function been used to create the reflections, they *still* would not have matched.

[21] Assuming sRGB color primaries, where the weight of the red channel is 21%.

Fixed Layers

Many real world materials need more than the four predefined layers of `mia_material` to be accurately portrayed. For example, car paint would need colored glossy reflection *and* white specular clear-coat reflections. This is impossible to achieve with a single `mia_material` instance, and whilst it is possible to blend multiple materials using a color mixing shader, this is highly inefficient[22]. It will also not work with shadow and photon shaders, which work inherently differently in *mental ray* and so can't just be mixed in that way.

Missing Effects

Finally, `mia_material` doesn't support subsurface scattering and a few other shading effects that may be desirable. So as a "do everything" material, it fails to actually do *everything*.

Tiny Bumps are Hard

Another more general problem that isn't specific to `mia_material` is the problem of small (subpixel) bumps getting prematurely filtered.

In reality, all glossiness effects are actually the result of some form of sub-pixel structure. But when a traditional bump or normal map gets filter *prior* to shading, this effect goes away with the filtering. It may look fine in a close up where every little bump is visible, but seen at a distance where for example every screen pixel covers 100 normal map pixels (causing the normal map to get heavily filtered) the effect breaks down and the material appears too shiny.

The `mia_material` lacks any solution to this problem. A few experiments were made to solve it, but none were ultimately successful. For example: instead of filtering 100 normals down to a *single* normal and shading that, we attempted to actually sample the light once, and re-using this light sample shade each of the 100 normals. This turned out to be way too slow to be usable, and was never actually added.

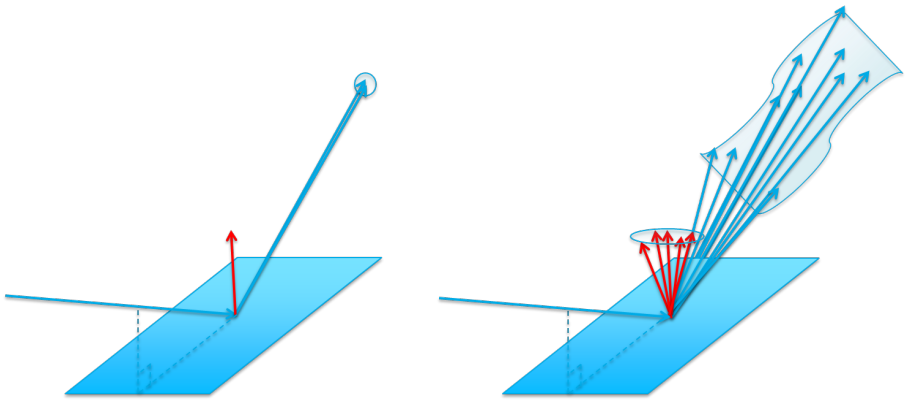

Figure 14: Filtering a normal-map down to a single normal (left) vs. shading all normals and filtering the result (right)

[22]Each material would run its own light loop and trace its own set of rays, quickly exploding render times.

Going Forward

The experience of creating `mia_material` and seeing it spread throughout the industry has been an interesting and rewarding experience. But what can we learn from all this?

Highlight and Reflection Appearance is Important

Our argument that the Gaussian or cosine-raised-to-a-power look of classic specular models was *wrong*, and that real-world highlights seem to have a falloff that appears much more *exponential* in nature was purely empirical. But in 2012 at least two papers were released that seem to vindicate our observations:

- Brent Burley, "Physically-Based Shading at Disney" from SIGGRAPH 2012[23]
- Joakim Löw, Joel Kronander, Anders Ynnerman and Jonas Unger, "BRDF Models for Accurate and Efficient Rendering of Glossy Surfaces" from Linköping University[24]

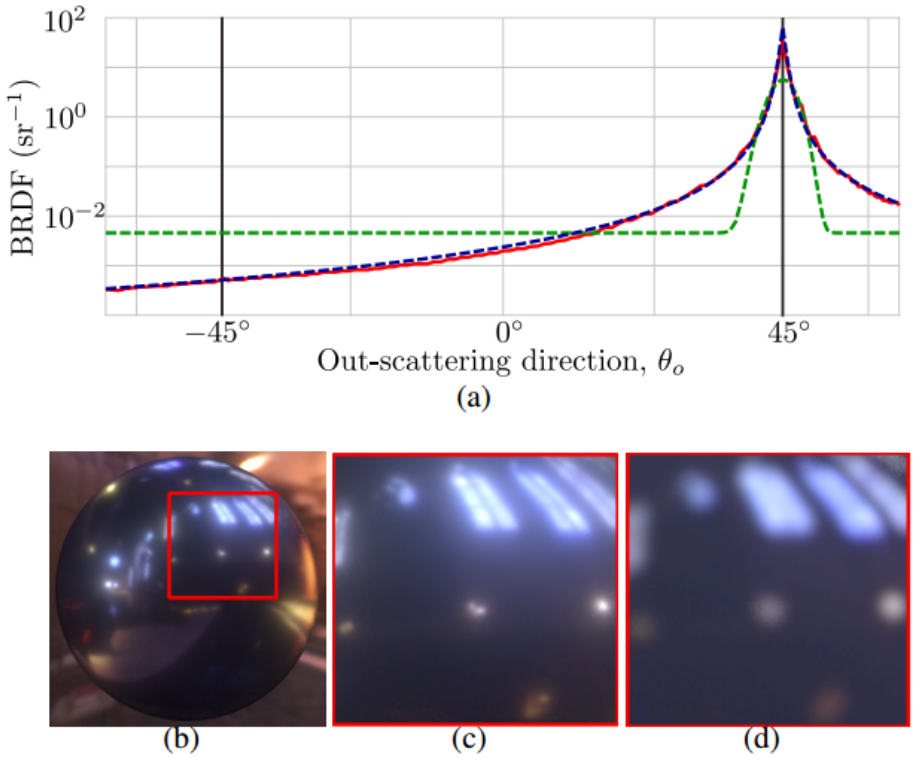

Figure 15: An important graph from the Löw et al. paper.

In Figure 15 (a), the red line is measured data, the blue line is the model proposed in the paper and the green line is Cook-Torrance. The difference between image (c) (their model) and image (d) (Cook-Torrance) shows *exactly* the visual problem we witnessed when studying many available glossy models empirically. Clearly, more research needs to go into this area.

[23] `http://blog.selfshadow.com/publications/s2012-shading-course/`
[24] `http://vcl.itn.liu.se/publications/2012/LKYU12/`

Reflection Stretchiness and Tiny Bumps

A very promising technique that solves *both* the problems of stretchy reflections and *also* handles the effect of tiny bumps seen at a distance can be found in this paper:

- Marc Olano and Dan Baker, "Lean Mapping" from Firaxis Games[25].

Flexibility is Important

The design of `mia_material` as a monolithic predefined set of layers was intentional, for performance as well as ease-of-use reasons, yet it is limiting for the end user. Something more flexible is needed, and in fact there is currently a clear industry move away from shaders as pieces of code, towards materials as a mixture of atomic BRDFs.

Fortunately, a successor to `mia_material` is currently being developed by *NVIDIA ARC*[26] based on a flexible and efficient layering model. These shaders are known as the MILA[27] shaders, and are based on code I wrote while still an *NVIDIA ARC* employee, but the shaders have been developed further since then. Curious *mental ray* users can find a beta download of these shaders at the *NVIDIA ARC* forum mentioned below.

Figure 16: Dirty glossy skin dipped in strawberry jam, rendered in the MILA shaders.

[25] `http://www.csee.umbc.edu/~olano/papers/lean/`
[26] The company formerly known as *mental images GmbH*.
[27] This stands for **m**ental **i**mages **la**yering.

Want to Know More?

More information on the mia_material and related shaders can be found in the documentation of the Architectural shader library. Another source of information is my blog at `http://www.mentalraytips.com`, as well as the user forums at NVIDIA Advanced Rendering Center, found at `http://forum.nvidia-arc.com/`.

Adventurous people can also follow **@MasterZap** on Twitter but be forewarned, it is *mostly* Instagram pictures of Sushi....