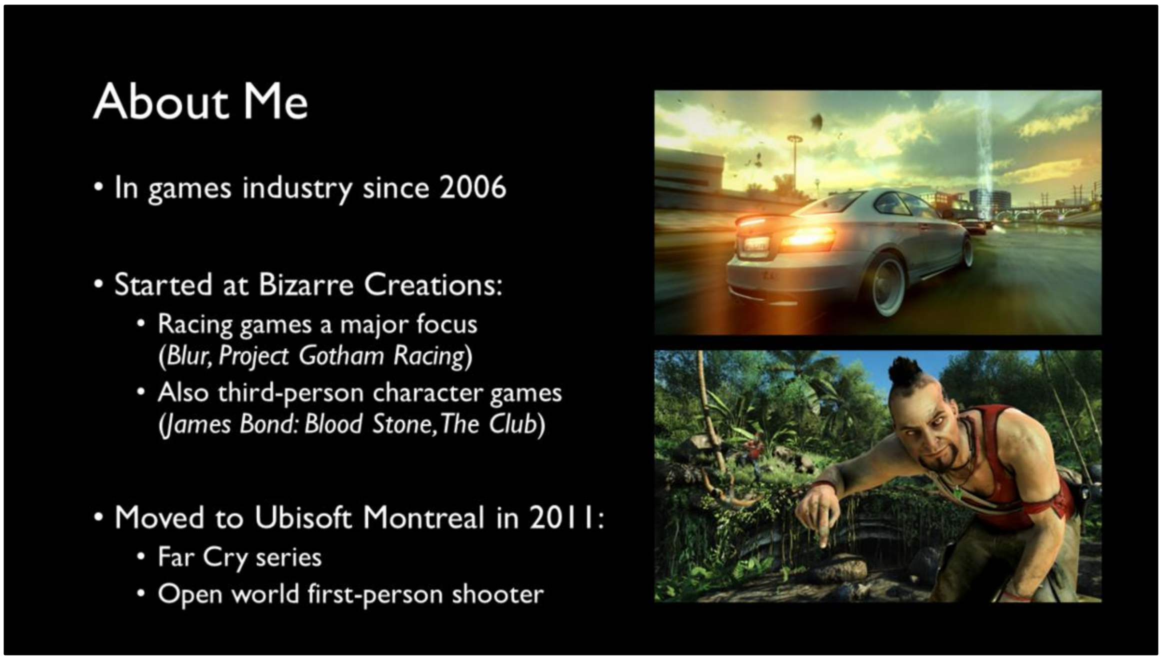
Advances in Rendering, Graphics Research and Video Game Production

Stephen McAuley, Technical Lead, Ubisoft Montreal

Thanks for the introduction, and hello everyone! It's a real honour to be invited to speak here at I3D. It's my third time attending the conference, and I've been lucky to see some fantastic papers presented every time I've attended and so far is no exception. As a resident of Montreal, I should also say welcome, I hope those of you visiting the city really enjoy your time here.

One of the main reasons we're here at I3D is to talk about real time 3D computer graphics, and I'm a representative of the video games industry where my role as a graphics programmer is to take the research presented at conferences like this, and to try to take them into production. I'm hoping that what I'm going to talk about today will give you a few more insights into how that works and the difficulties we face doing so. I'm going to take some research I've been doing very recently into improving our lighting and materials through multiscattering diffuse and specular BRDFs and area lights, and walk you through all the steps I've had to take so far on that journey. My goal is that I give researchers insight into how to make papers more useable and accessible for those of us in industry, but there should be a few tips in there for industry professionals of how to make the best use of research too.

I should begin though by telling you a little bit about me and the games that I've worked on to give some background to the challenge.



About Me

- In games industry since 2006
- Started at Bizarre Creations:
  - Racing games a major focus (*Blur, Project Gotham Racing*)
  - Also third-person character games (*James Bond: Blood Stone, The Club*)
- Moved to Ubisoft Montreal in 2011:
  - Far Cry series
  - Open world first-person shooter

My name is Stephen McAuley, and I've been in the games industry since 2006 since I finished university where I studied mathematics. I started in the UK at a company in Liverpool called Bizarre Creations, whose major focus was on racing games, such as Project Gotham Racing and Blur, but they also did some third person character games, which culminated in James Bond 007: Blood Stone. This was a really great place to start in games because I had some mentors who were very focused on visual quality, physically based shading and providing solid tools for artists.

These were all things I took to Ubisoft Montreal in 2011, where I started work on Far Cry 3, and I've been working on Far Cry games ever since. Now, Far Cry games are big open world first person shooters, and this was a big new challenge to me. Working on racing games was fun and you could do some stunning visuals, but in the grand scheme of things, they're relatively simple projects. You have one type of activity - driving cars, and the game is effectively on rails so you can generally predict what's going to happen. In an open world game, everything is much more complex and this poses a great deal more challenges for computer graphics.

Let's take a look at some of the challenges that we face.

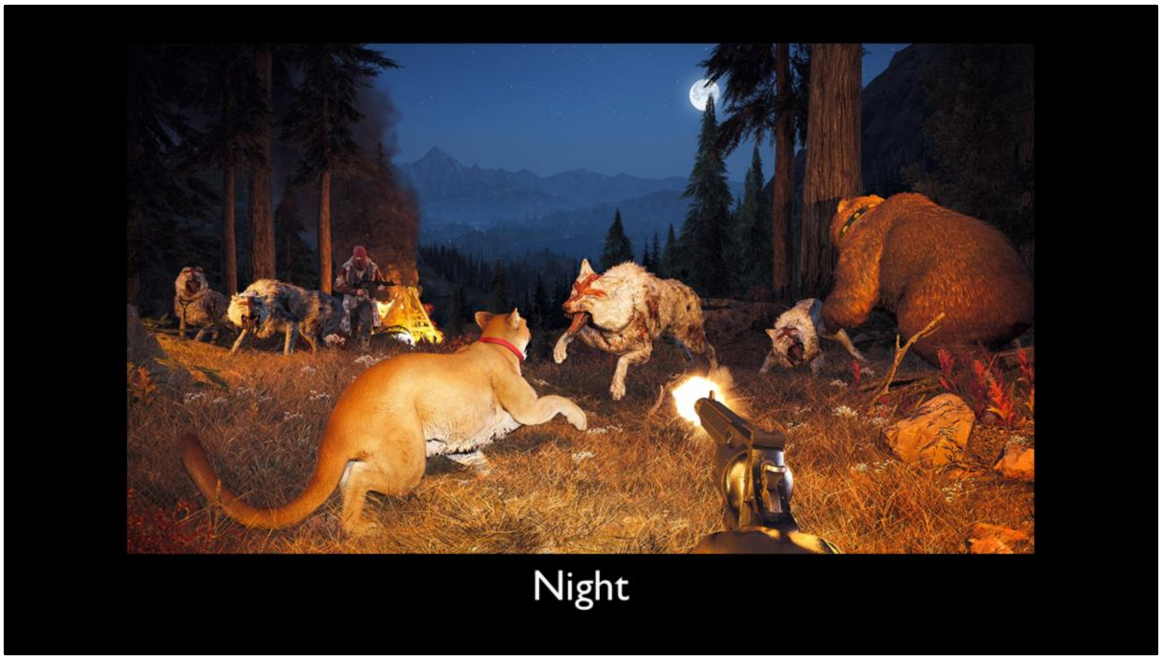
Obviously in an open world game you have huge outdoor environments. So if you're implementing a global illumination system, you need to figure out a way to store the data so it fits on disk and in memory for a world around 10km x 10km in size.

Whitetail
Indoors

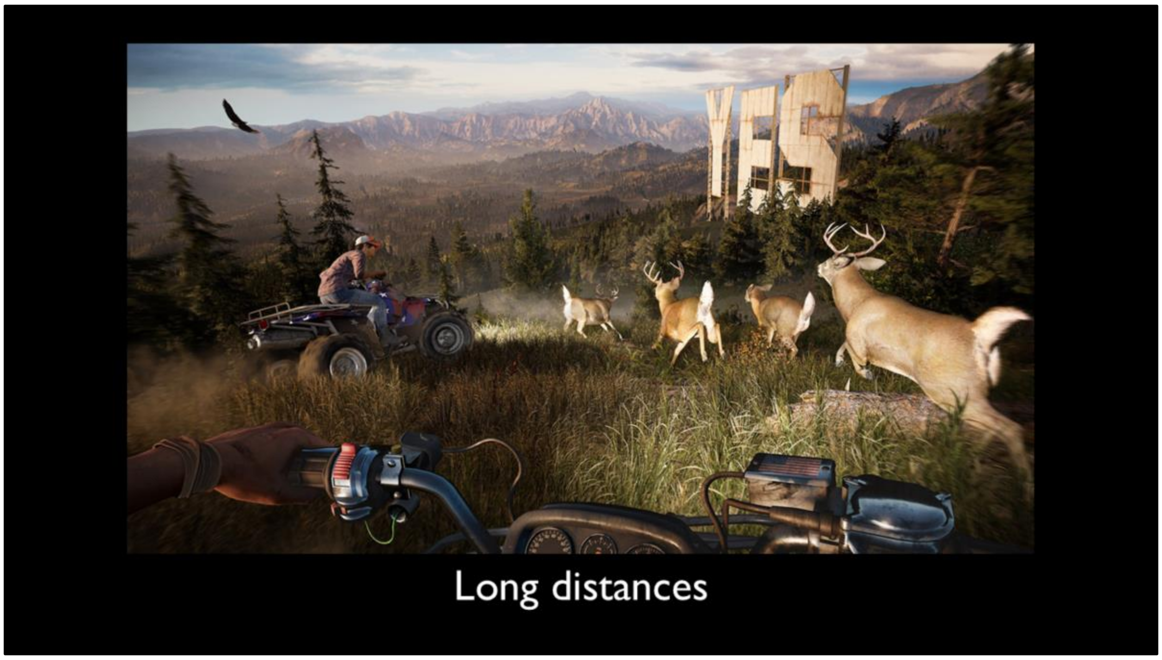
But we also have indoor environments, and here we want really detailed lighting and a much higher resolution global illumination than outdoors. This means that we’re going to favour adaptive, scalable solutions that adapt to our environments. Of course, interiors also open up more room for bugs like light leaks.

Day

Our lighting environment isn't static though. We have a dynamic time of day system, which means that we have day, but we also have night…

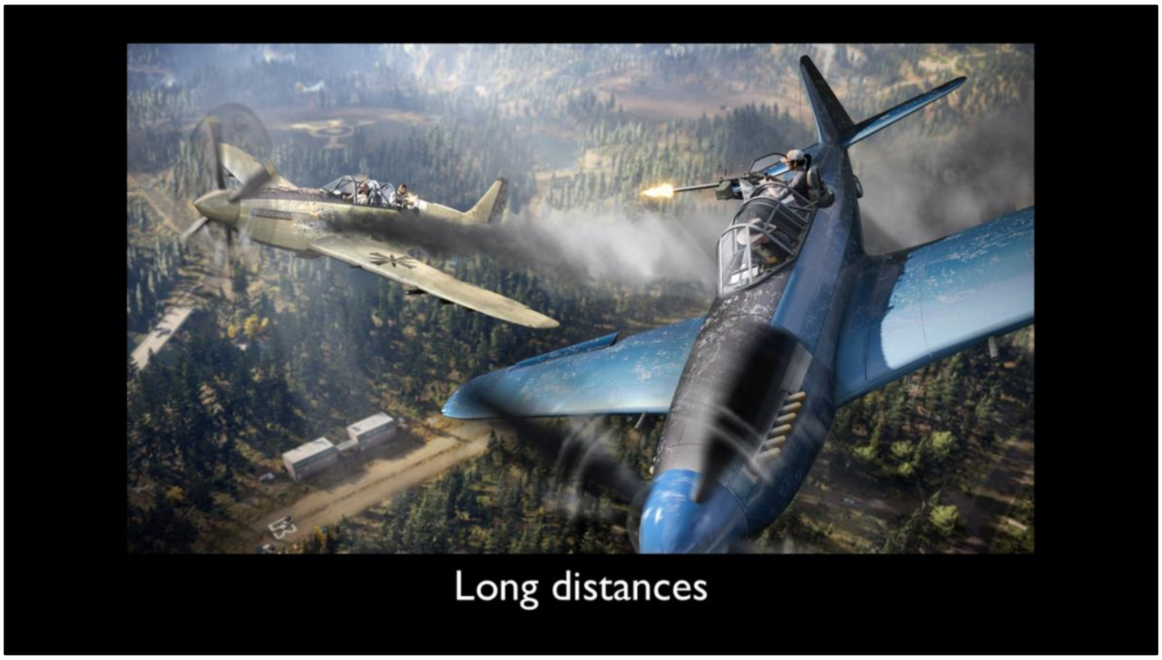


…and this gives our lighting artists placing lights into the scene, like you can see here, a huge challenge as they have to make things work and look realistic in a huge variety of conditions. Plus, you have to ensure that the lighting conditions change smoothly, which is really difficult at dawn and dusk where the light changes rapidly. Plus, we have to deal with contrast between interiors and exteriors at all times of day, which causes exposure problems and visibility issues for gameplay.



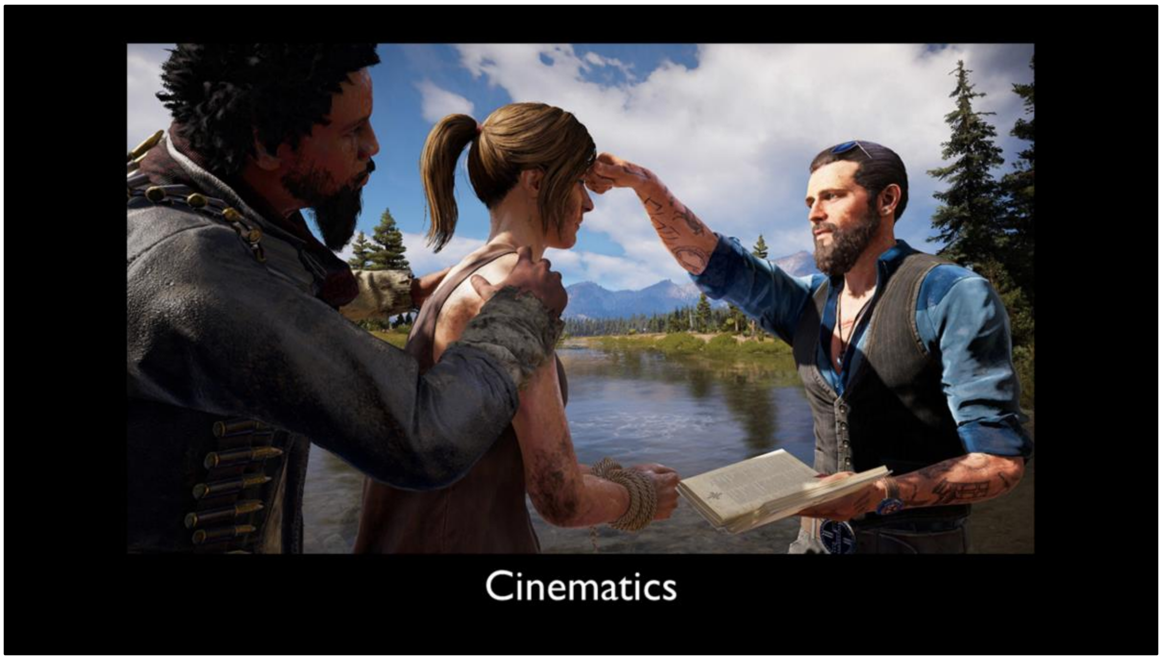
Long distances

We also have huge distances to cover, with approximately 10km plus view distances. This means that we need a variety of different shadowing solutions that will work at all distances, and we also need different fog solutions to ensure that we get detail close to the player but also the beautiful atmospheric scattering you want at long distances.



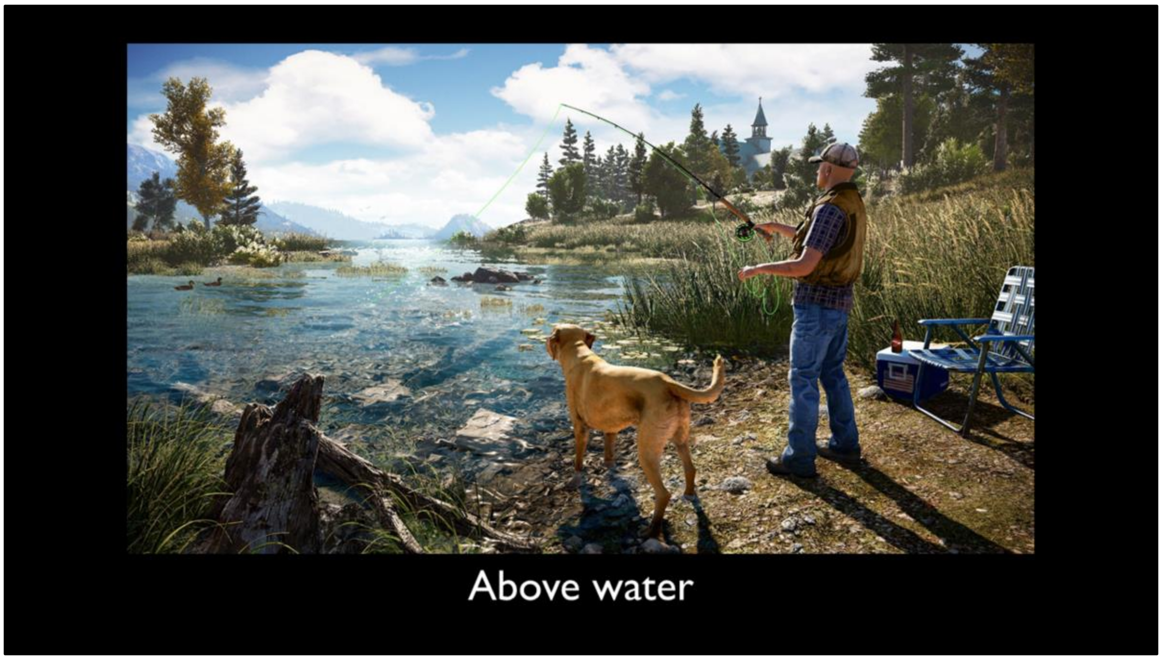
Long distances

And we have aeroplanes, so everything has to look good from up high too.



Far Cry is also a very story-driven game, so we have cinematics, where we need things to look really good close up. This is where we struggle with things like depth precision, and seeing all those wonderful shadow mapping artefacts we know and love.

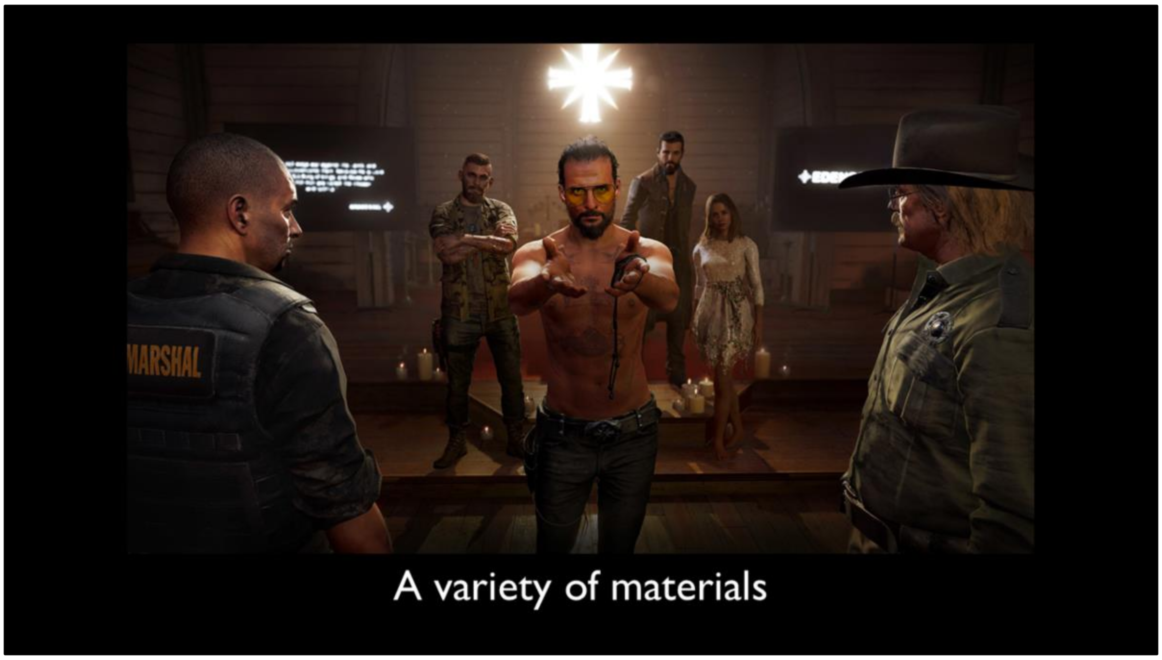
But this baptism scene from Far Cry 5 also reminds us of the challenges we have in our natural environment.



Most of the time we’re above water, and if we want refraction that poses problems of multiple layers of transparency, if we have transparent objects both above and beneath the water. But even worse, we can of course go under the water…

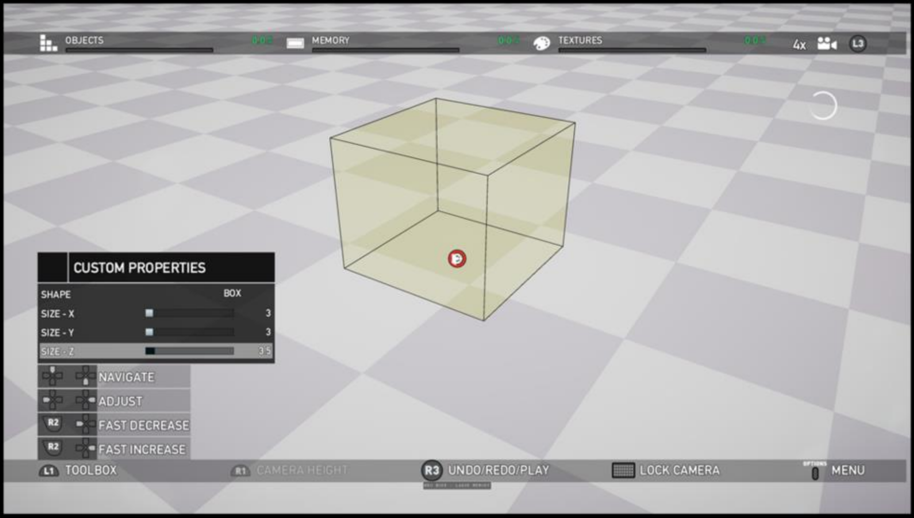
Underwater

…where our rendering order of objects is flipped, our assumptions change and we face new challenges. For example, do we want to take our volumetric fog solution for above water and use it for some cool underwater fogging? Do we run both and pay an extra performance cost? Do we turn one off and the other on? How do the transitions look when we do that? These are all things we have to ask ourselves.



A variety of materials

Then when it comes to shading, we have many different material types, from skin, hair and cloth on characters, to translucent leaves and multi-layered car paint. These all have to work and look good as they're all in the game and will be very prominent at certain points.

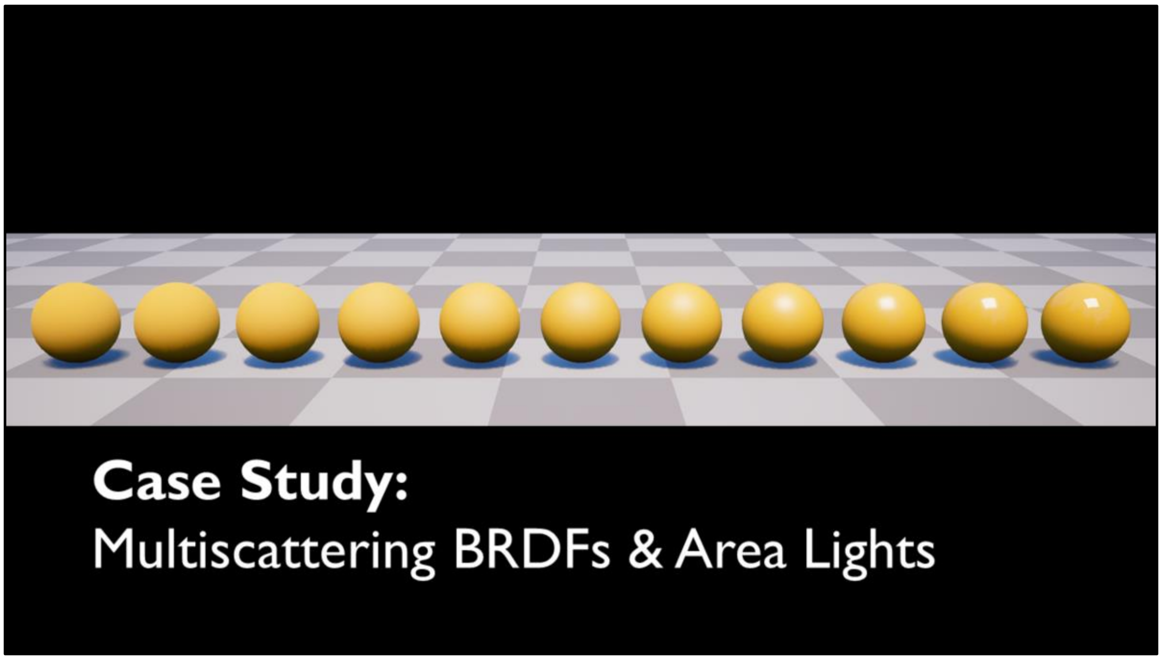


Dynamic geometry

The final complexity is our Editor. We have an Editor that our content creators use themselves to build the game, but many of our Far Cry games have also featured an In Game Editor where players can build their own levels. This means that we can't necessarily even rely on geometry being static – because there's a mode where players and our own developers can move things around at will. So if we cache our distant sun shadows, because we think the geometry won't move or the lighting won't change, well, one of our artists can just pick up the object in the Editor and move it, then change the time of day… and expect it to work and look good. If it doesn't, even if it isn't something the player will necessarily see, we'll still be hearing about it as it hinders the artist making the world.

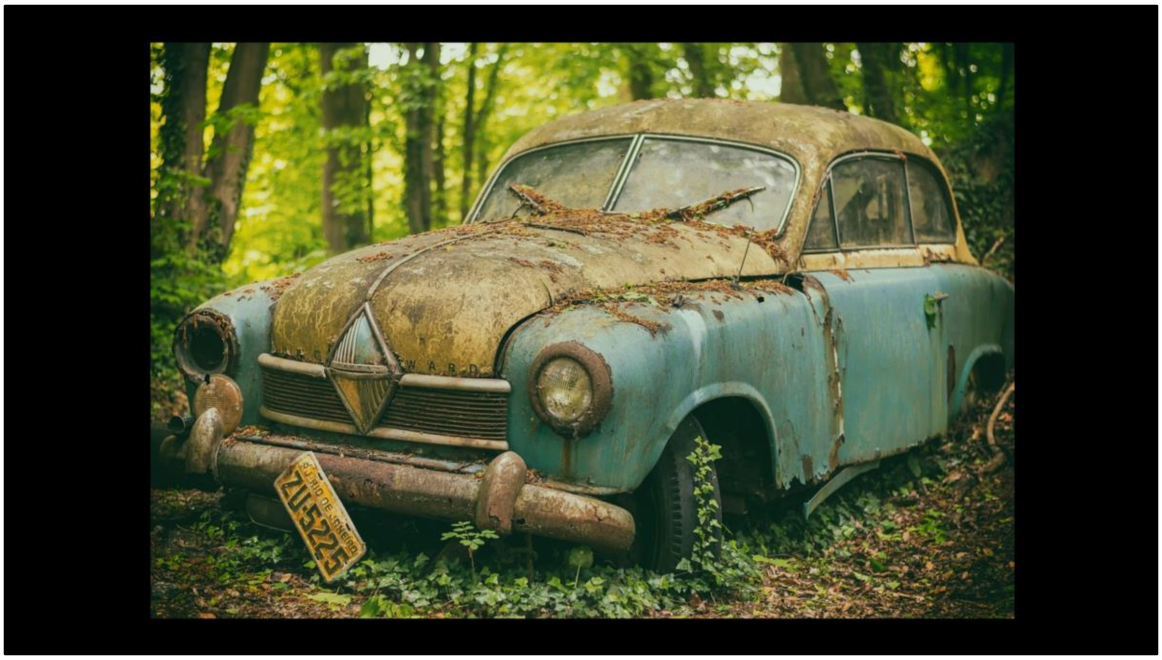
So I hope by now you're getting an idea of how difficult it can be to make things work in a Far Cry game, and the vast complexity that we face. To go back to that example of shadows, I might need a number of different shadowing solutions – for close to camera, from very far away from the camera, for sun light, for moon light, for local lights… and I probably need them to work to a degree with fast moving objects and fast moving time of day too. There are of course some permissible compromises to all these, but we're still going to have to *think* about them in order to compromise them, which means we still have to hold all these cases in our head…

…and that takes a lot of brain power and can be pretty exhausting at times!



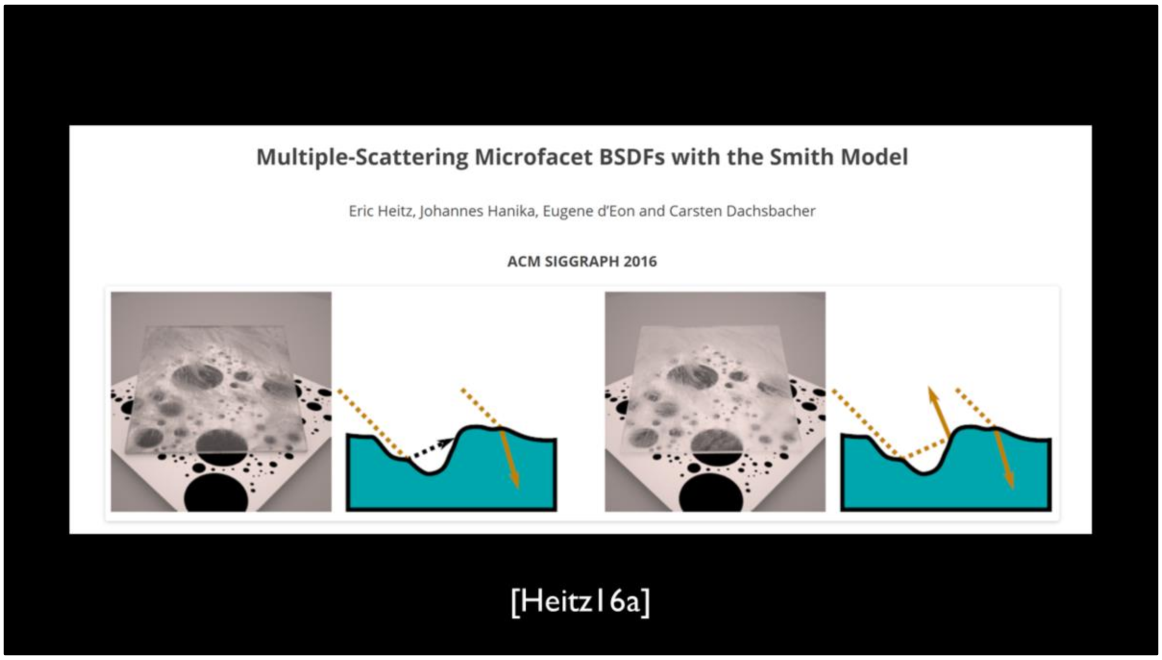
Case Study:
Multiscattering BRDFs & Area Lights

But now you have a background on the general things we have to think about, it's time to focus on a case study of what happens when we take some research and try to get it into our games. There's actually no better topic than the one I'm working on right now, which is on multiscattering BRDFs and area lights. It's posed a number of really interesting challenges, so we're going to walk through what has happened when I've implemented some research, what problems I've had to overcome, and see what insights we learn from that along the way.



At the end of Far Cry 5, I was feeling our material system was getting a little out of date. Sure, it used to be pretty good, but we were still pretty much based on the Disney BRDF model from 2012, but with Lambertian diffuse rather than the Disney diffuse model. A lot of research had been coming out and I thought it was about time I did an upgrade and saw what was out there, and what benefit it could bring us.

There were some problems mentioned by artists too. Generally people still wanted more specular being visible, and another wish was a slightly softer falloff for diffuse. Plus, of course everyone wants the game to look more realistic, and better materials and lighting are a key route to that.



Multiple-Scattering Microfacet BSDFs with the Smith Model

Eric Heitz, Johannes Hanika, Eugene d'Eon and Carsten Dachsbacher

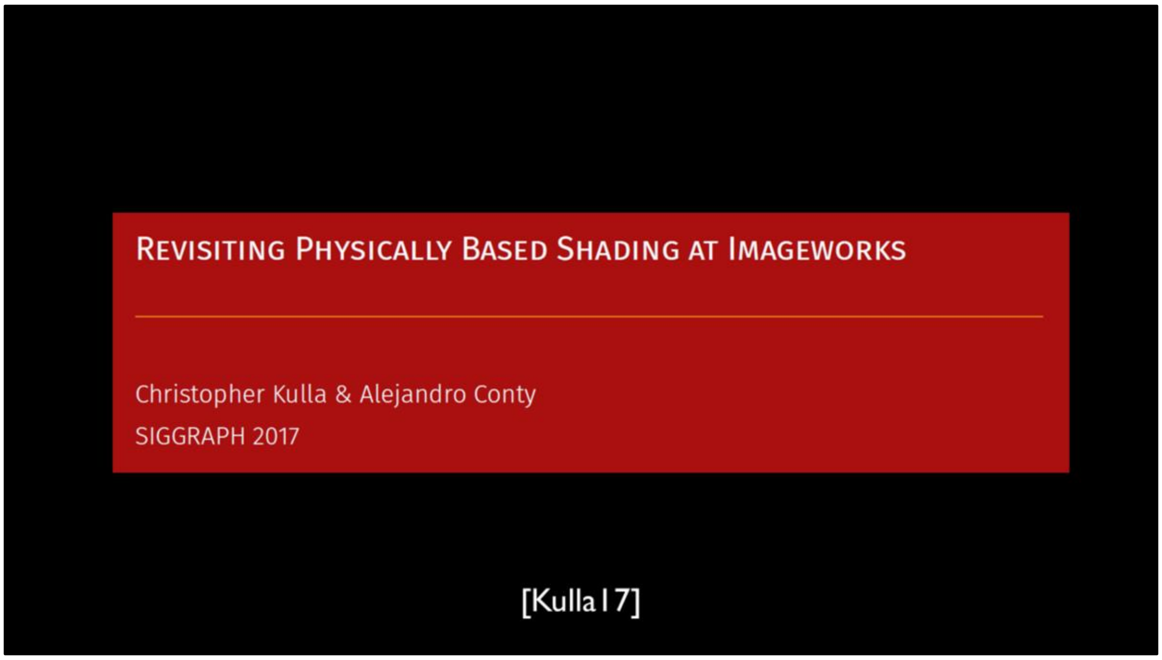
ACM SIGGRAPH 2016

[Heitz16a]

The paper that had interested me most in the last few years, was this excellent paper from Eric Heitz et. al. It was the first paper to really bring multiple scattering to my attention, and made me realise how much energy we're losing with our standard specular models. This was a revelation to me – I'd spent a lot of time evangelising energy conservation… only to find that actually, I'd been pretty happy to lose a lot.

I had absolutely no idea what to do about the problem though. The paper did some rather brute force solutions, but it didn't really provide a clear path to finding a real-time approximation.

Of course, someone was going to do that eventually…



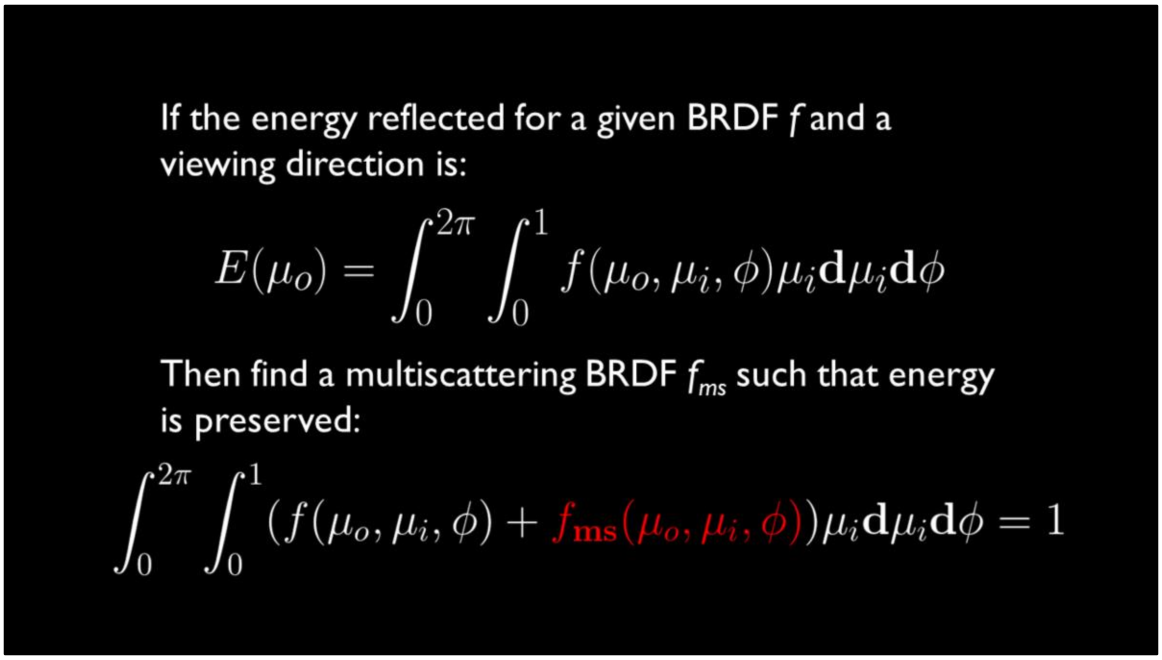
REVISITING PHYSICALLY BASED SHADING AT IMAGEWORKS

Christopher Kulla & Alejandro Conty

SIGGRAPH 2017

[Kulla17]

There's where Chris Kulla and Alejandro Conty from Sony Imageworks come in. They presented their approximation to multiscattering specular at SIGGRAPH 2017, by a simple trick designed to preserve energy. It reduced to some simple formulae and easily calculable LUTs, so this very much seemed like a winner.



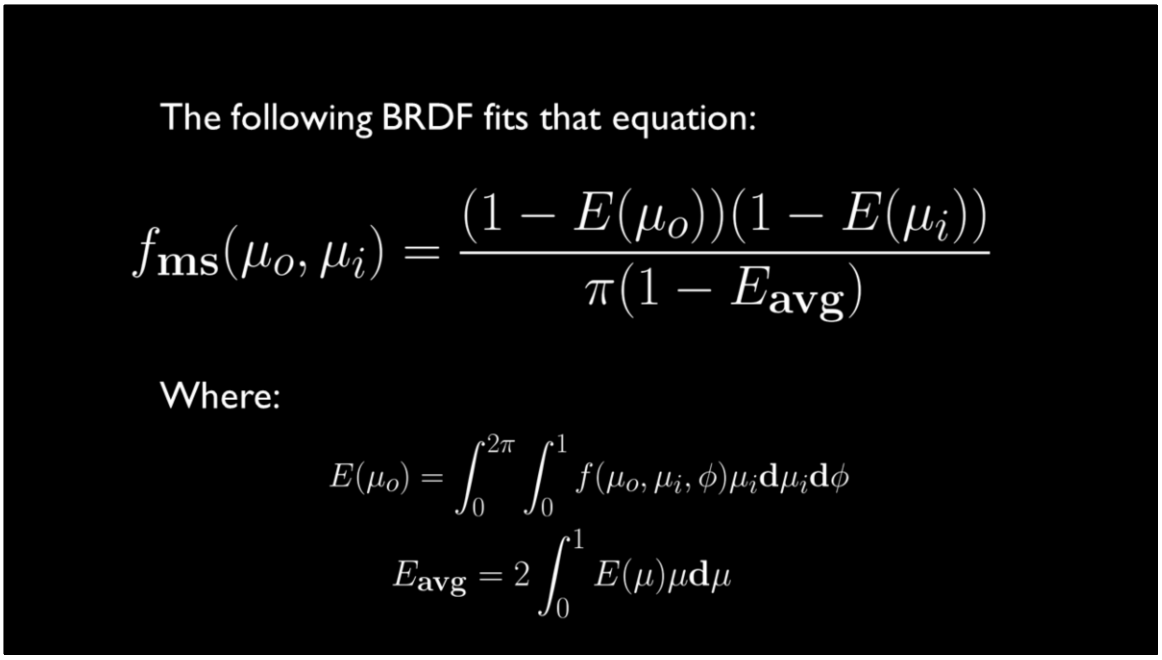
If the energy reflected for a given BRDF *f* and a viewing direction is:

$$E(\mu_o) = \int_0^{2\pi} \int_0^1 f(\mu_o, \mu_i, \phi)\mu_i \mathbf{d}\mu_i \mathbf{d}\phi$$

Then find a multiscattering BRDF f_{ms} such that energy is preserved:

$$\int_0^{2\pi} \int_0^1 (f(\mu_o, \mu_i, \phi) + f_{\mathbf{ms}}(\mu_o, \mu_i, \phi))\mu_i \mathbf{d}\mu_i \mathbf{d}\phi = 1$$

It’s a pretty simple idea. We observe that we need a BRDF that makes up for the missing lost energy.



The following BRDF fits that equation:

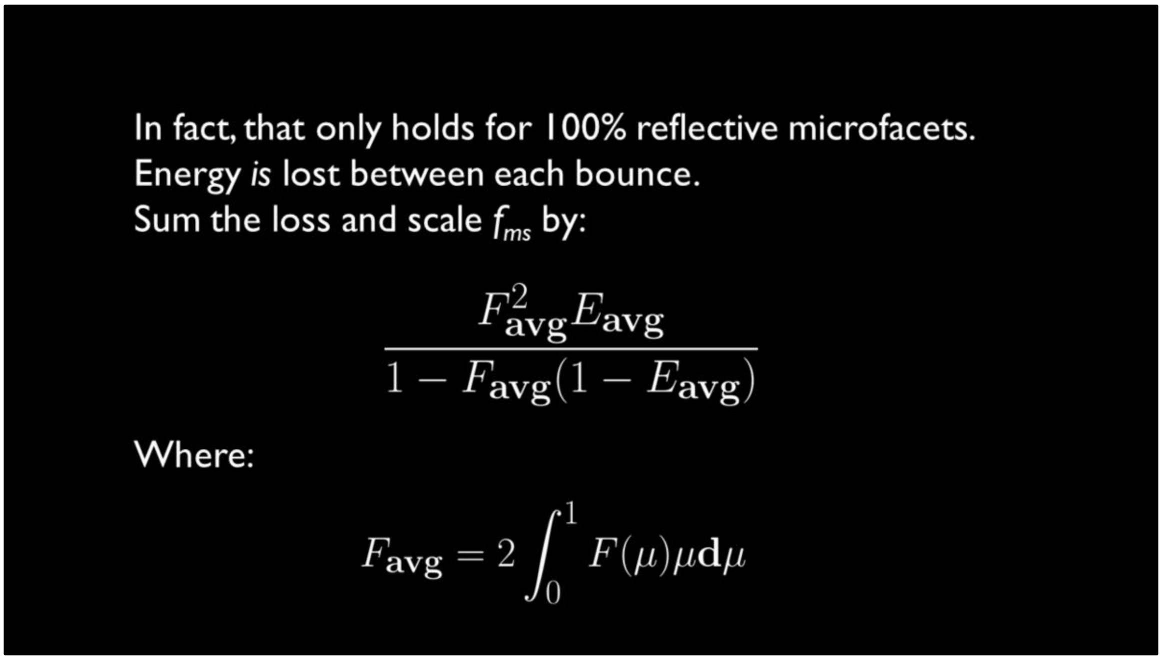
$$f_{\mathbf{ms}}(\mu_o, \mu_i) = \frac{(1 - E(\mu_o))(1 - E(\mu_i))}{\pi(1 - E_{\mathbf{avg}})}$$

Where:

$$E(\mu_o) = \int_0^{2\pi} \int_0^1 f(\mu_o, \mu_i, \phi)\mu_i \mathbf{d}\mu_i \mathbf{d}\phi$$

$$E_{\mathbf{avg}} = 2\int_0^1 E(\mu)\mu \mathbf{d}\mu$$

So we find a BRDF that fits, based upon the energy reflected in a given direction by a BRDF, and the average energy. This all works great…



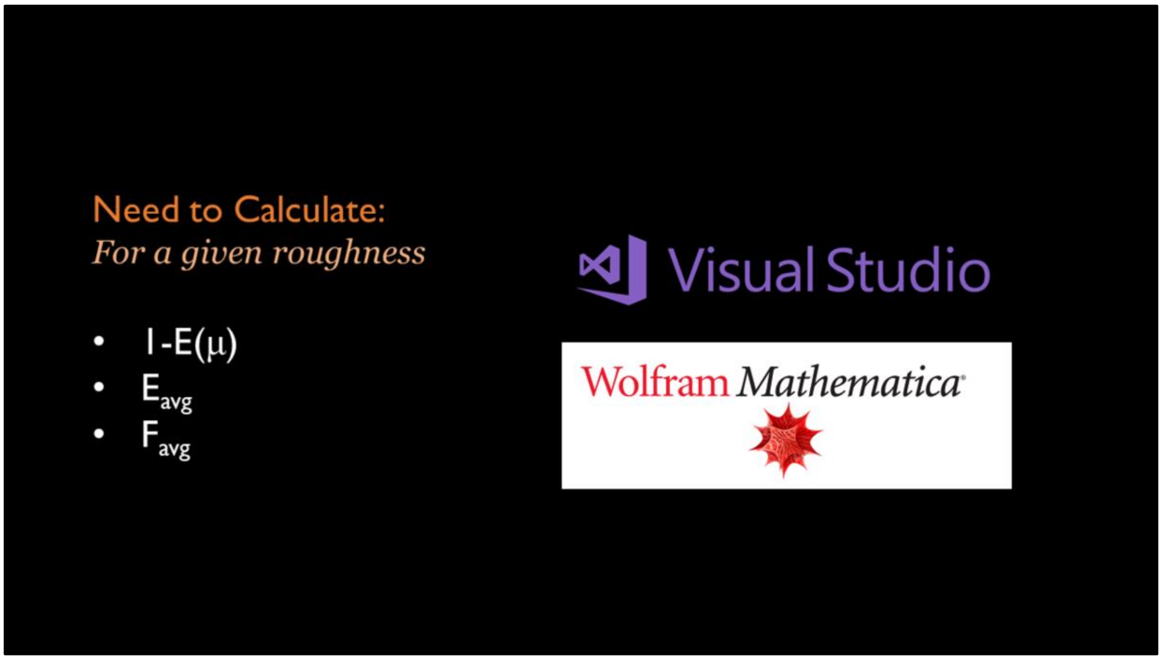
In fact, that only holds for 100% reflective microfacets.
Energy *is* lost between each bounce.
Sum the loss and scale f_{ms} by:

$$\frac{F_{\mathbf{avg}}^2 E_{\mathbf{avg}}}{1 - F_{\mathbf{avg}}(1 - E_{\mathbf{avg}})}$$

Where:

$$F_{\mathbf{avg}} = 2 \int_0^1 F(\mu)\mu \mathbf{d}\mu$$

…but actually we do want some lost energy, as energy is lost (absorbed by the surface) each bounce. So we sum the loss for infinite bounces and scale our multiscattering BRDF by the following equation. F here is of course the Fresnel, which gives us a fraction of the average energy lost at each bounce.

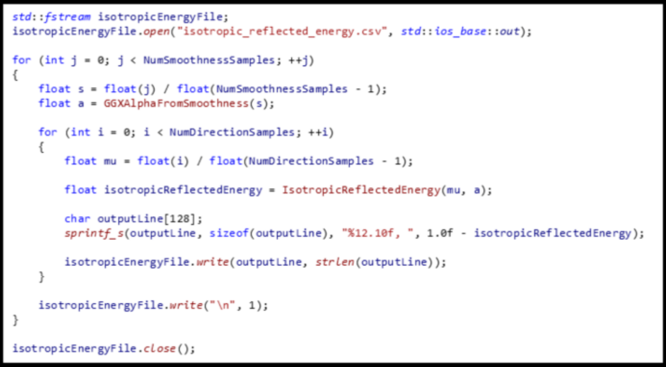

Need to Calculate:

For a given roughness

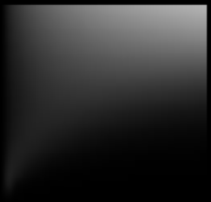
- $1-E(\mu)$
- E_{avg}
- F_{avg}

Visual Studio
Wolfram Mathematica

But what do we need to do if we need to implement this? Well, we need to calculate three things. One minus the energy in a given direction, for a given surface roughness, the average energy for a given roughness, and the average Fresnel for a given specular colour. This is going to involve some integration over the hemisphere and some fitting, so I pulled out Visual Studio and Mathematica.

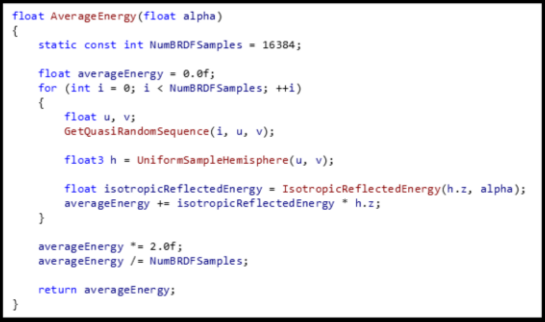


```
std::fstream isotropicEnergyFile;
isotropicEnergyFile.open("isotropic_reflected_energy.csv", std::ios_base::out);

for (int j = 0; j < NumSmoothnessSamples; ++j)
{
    float s = float(j) / float(NumSmoothnessSamples - 1);
    float a = GGXAlphaFromSmoothness(s);

    for (int i = 0; i < NumDirectionSamples; ++i)
    {
        float mu = float(i) / float(NumDirectionSamples - 1);

        float isotropicReflectedEnergy = IsotropicReflectedEnergy(mu, a);

        char outputLine[128];
        sprintf_s(outputLine, sizeof(outputLine), "%12.10f, ", 1.0f - isotropicReflectedEnergy);

        isotropicEnergyFile.write(outputLine, strlen(outputLine));
    }

    isotropicEnergyFile.write("\n", 1);
}

isotropicEnergyFile.close();
```

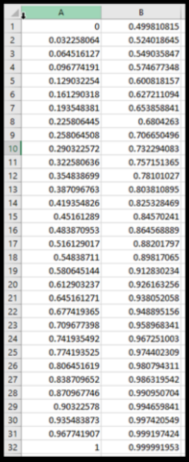


1-E(μ)

First I had to integrate the BRDF over the hemisphere to get one minus the reflected energy, and we store this in a 2D LUT parameterised by the cosine of the angle and the surface roughness (or smoothness, in our case).

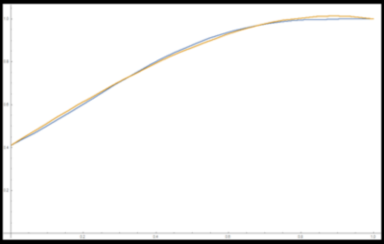


```
float AverageEnergy(float alpha)
{
    static const int NumBRDFSamples = 16384;

    float averageEnergy = 0.0f;
    for (int i = 0; i < NumBRDFSamples; ++i)
    {
        float u, v;
        GetQuasiRandomSequence(i, u, v);

        float3 h = UniformSampleHemisphere(u, v);

        float isotropicReflectedEnergy = IsotropicReflectedEnergy(h.z, alpha);
        averageEnergy += isotropicReflectedEnergy * h.z;
    }

    averageEnergy *= 2.0f;
    averageEnergy /= NumBRDFSamples;

    return averageEnergy;
}
```

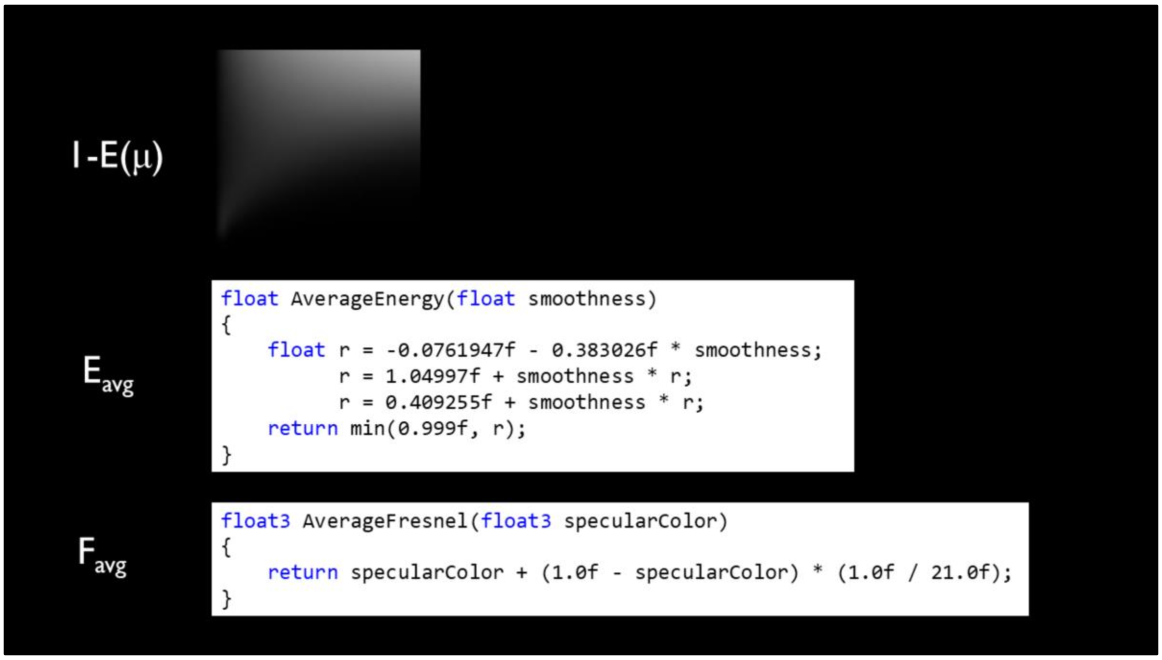


	A	B
1	0	0.499810815
2	0.032258064	0.524018645
3	0.064516127	0.549035847
4	0.096774191	0.574677348
5	0.129032254	0.600818157
6	0.161290318	0.627211094
7	0.193548381	0.653858841
8	0.225806445	0.6804263
9	0.258064508	0.706650496
10	0.290322572	0.732294083
11	0.322580636	0.757151365
12	0.354838699	0.78101027
13	0.387096763	0.803810895
14	0.419354826	0.825328469
15	0.45161289	0.84570241
16	0.483870953	0.864568889
17	0.516129017	0.88201797
18	0.54838711	0.89817065
19	0.580645144	0.912830234
20	0.612903237	0.926163256
21	0.645161271	0.938052058
22	0.677419365	0.948895156
23	0.709677398	0.958968341
24	0.741935492	0.967251003
25	0.774193525	0.974402309
26	0.806451619	0.980794311
27	0.838709652	0.986319542
28	0.870967746	0.990950704
29	0.90322578	0.994659841
30	0.935483873	0.997420549
31	0.967741907	0.999197424
32	1	0.999991953

E_{avg}



Then we needed to take that energy reflected in a given direction, and integrate that over the hemisphere to get the average energy for a given roughness. Then I took that data into Mathematica and found a fit.



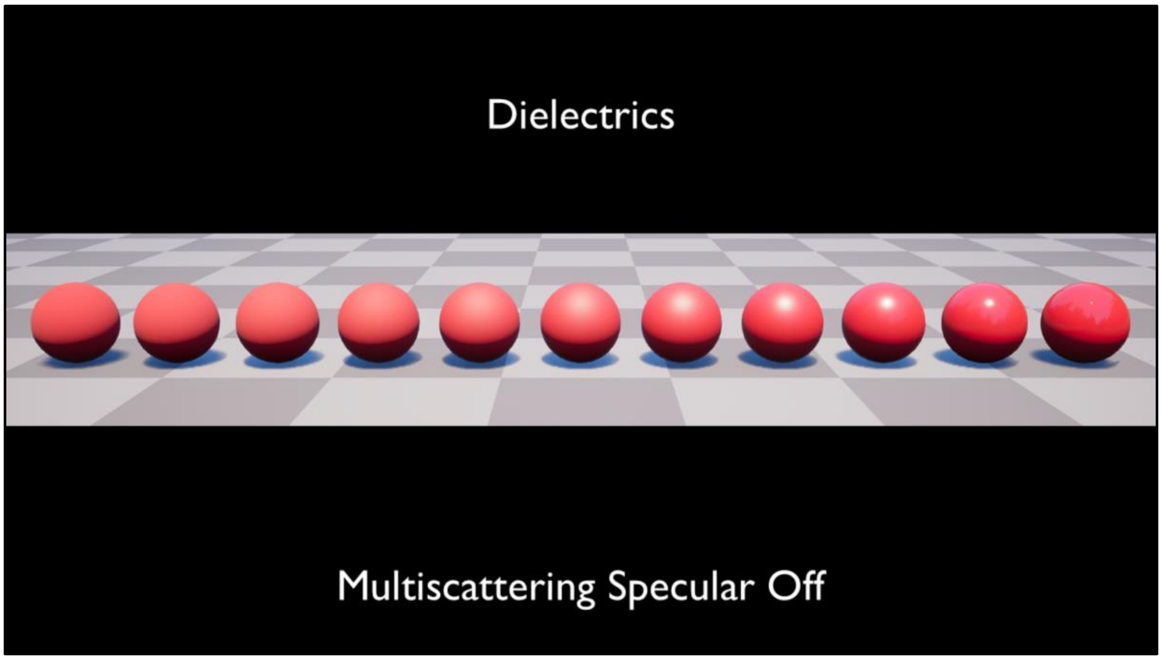
```
float AverageEnergy(float smoothness)
{
    float r = -0.0761947f - 0.383026f * smoothness;
          r = 1.04997f + smoothness * r;
          r = 0.409255f + smoothness * r;
    return min(0.999f, r);
}
```

```
float3 AverageFresnel(float3 specularColor)
{
    return specularColor + (1.0f - specularColor) * (1.0f / 21.0f);
}
```

Throw in a simple analytic integration of Schlick Fresnel to give the average Fresnel, and we end up with the following functions and look up tables we can use in our shaders…

So implementing this paper wasn’t particularly hard, but it did take knowledge of Mathematica and the time to write some C++ code to generate the LUTs.

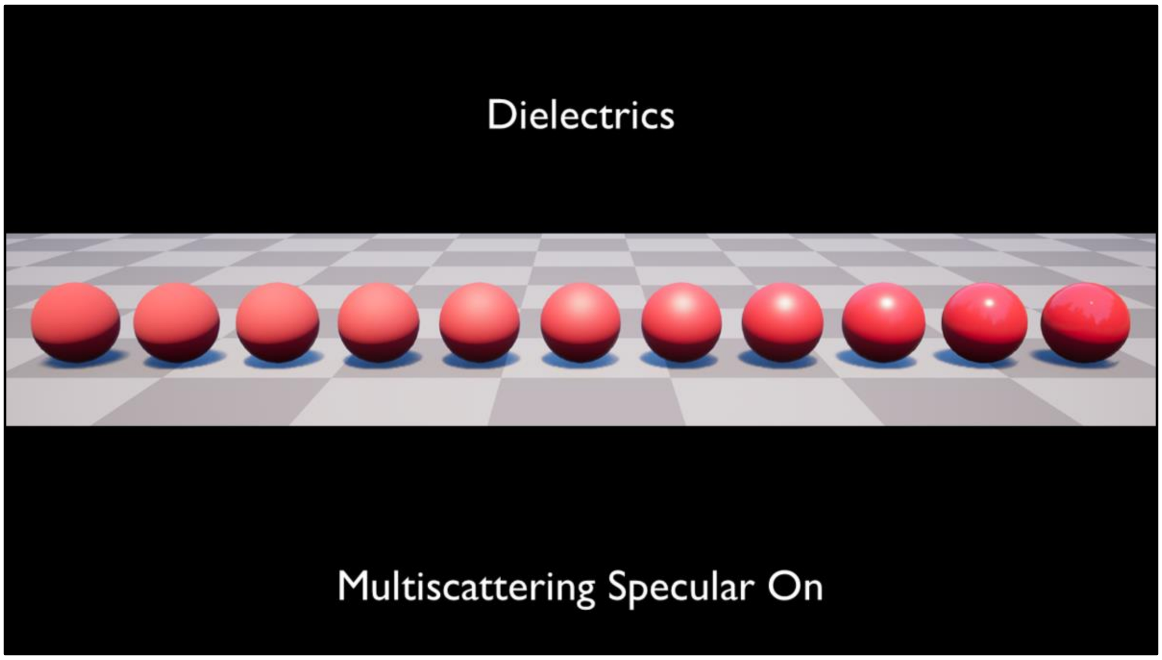
Let’s look at the results!



Dielectrics

Multiscattering Specular Off

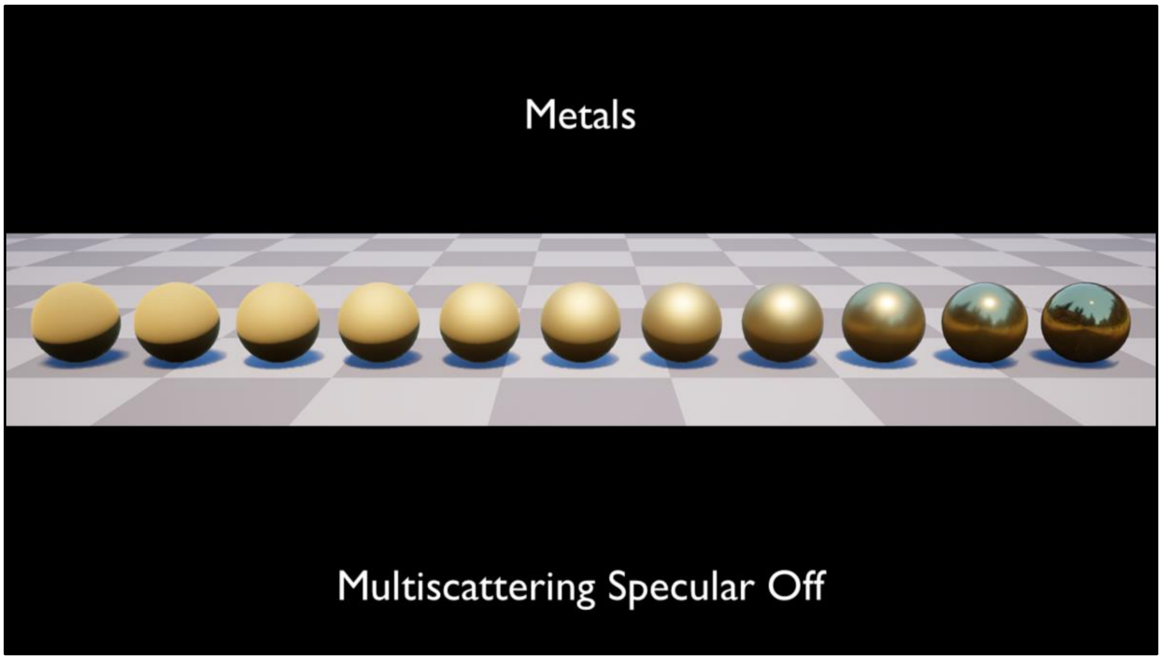
We start here with a row of spheres, smoothness increasing from left to right…


Dielectrics

Multiscattering Specular On

We turn multiscattering specular on… and… well… do you see any difference? Actually, if we toggle back and forth, we *do* see a small change on the roughest spheres, but it’s really hard to tell. This isn’t giving us the big win that we might have wanted.

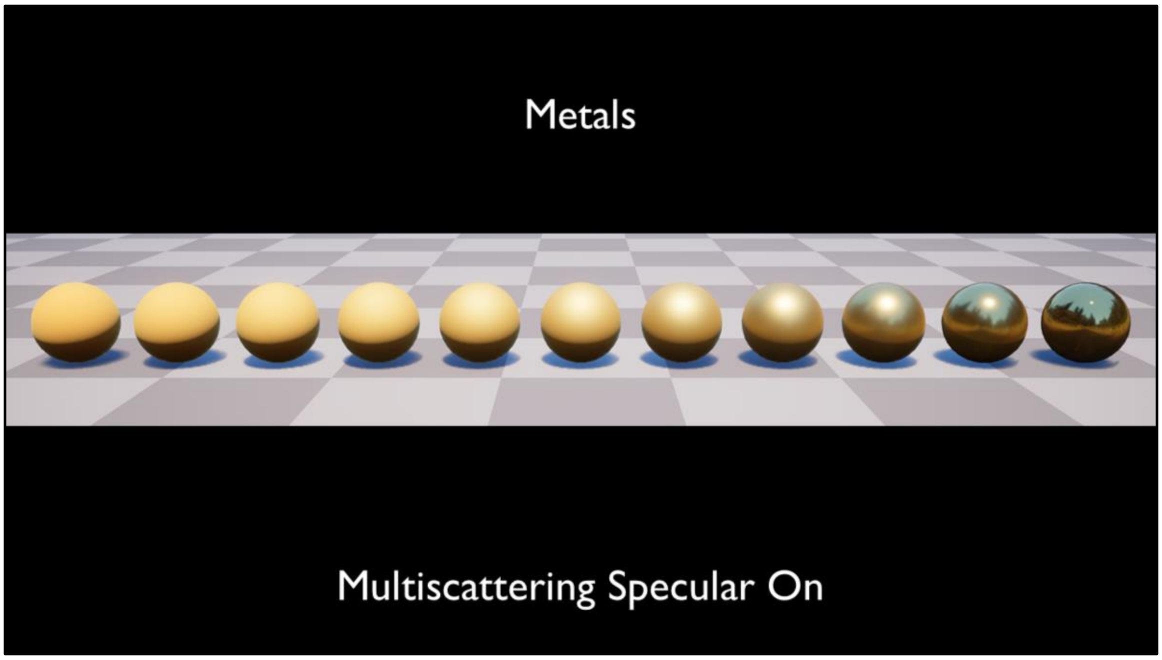
But before we give up, we should try this on metals.



Metals

Multiscattering Specular Off

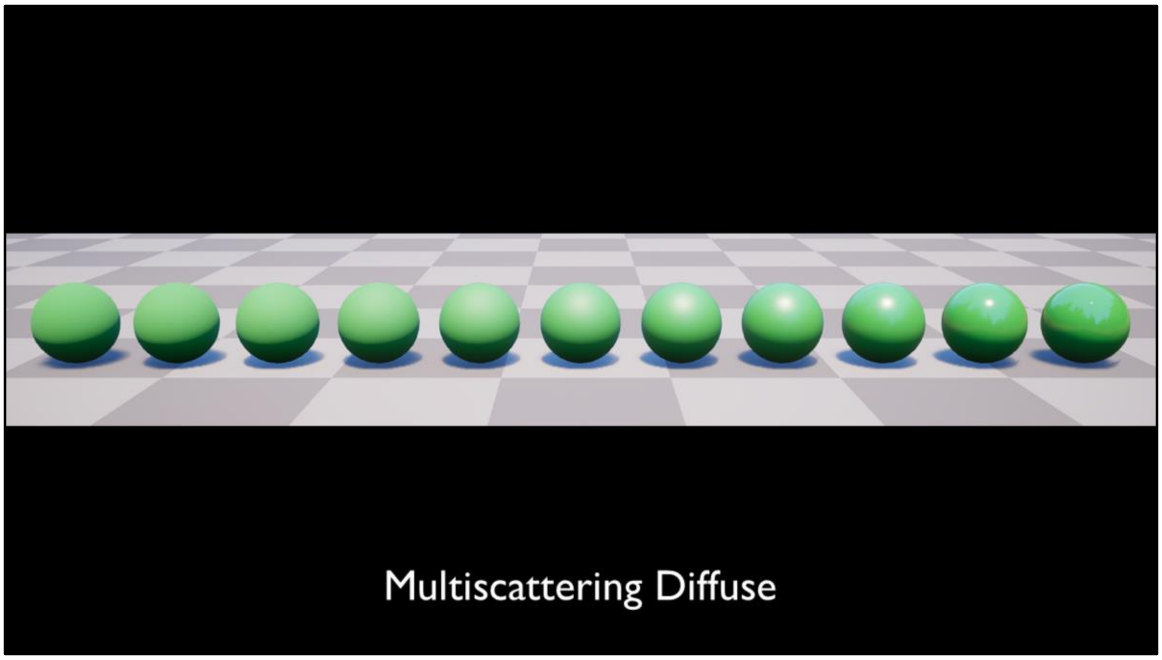
Again, we start with some gold spheres with the multiscattering specular off…



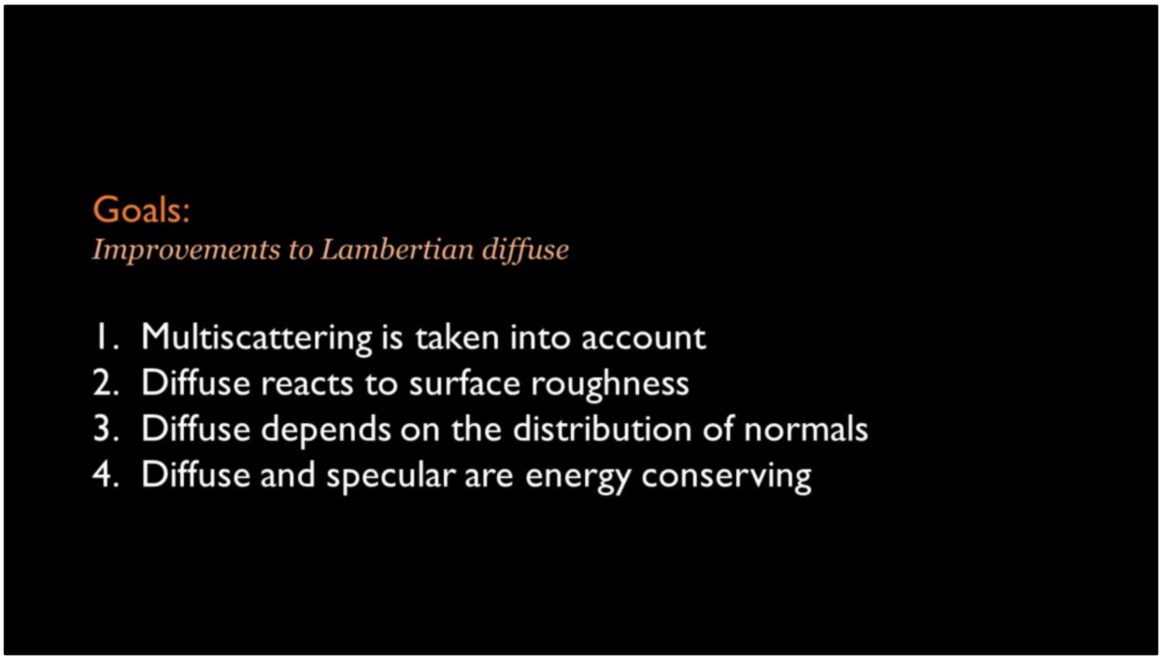
…and when we turn it on, wow, there's a huge difference here. Essentially, the greater specular reflectance means that less energy is being lost each bounce, so it's having a much bigger effect.

In fact, papers about multiscattering specular do mention this helps rough metals the most, so perhaps I should have paid more attention to those.

So this might not give the huge win on our overall image we're looking for. A typical Far Cry "postcard" shot involves lots of nature, such as trees, rocks and dirt that are dielectrics. But we do have vehicles and weapons that are metals, and this is going to improve their visuals a lot. In fact our lead artist in charge of them absolutely loves the effect this is giving, so that's one win we can be very happy with.


Multiscattering Diffuse

With multiscattering specular working successfully, we decided to switch and implement a new diffuse model.


Goals:
Improvements to Lambertian diffuse

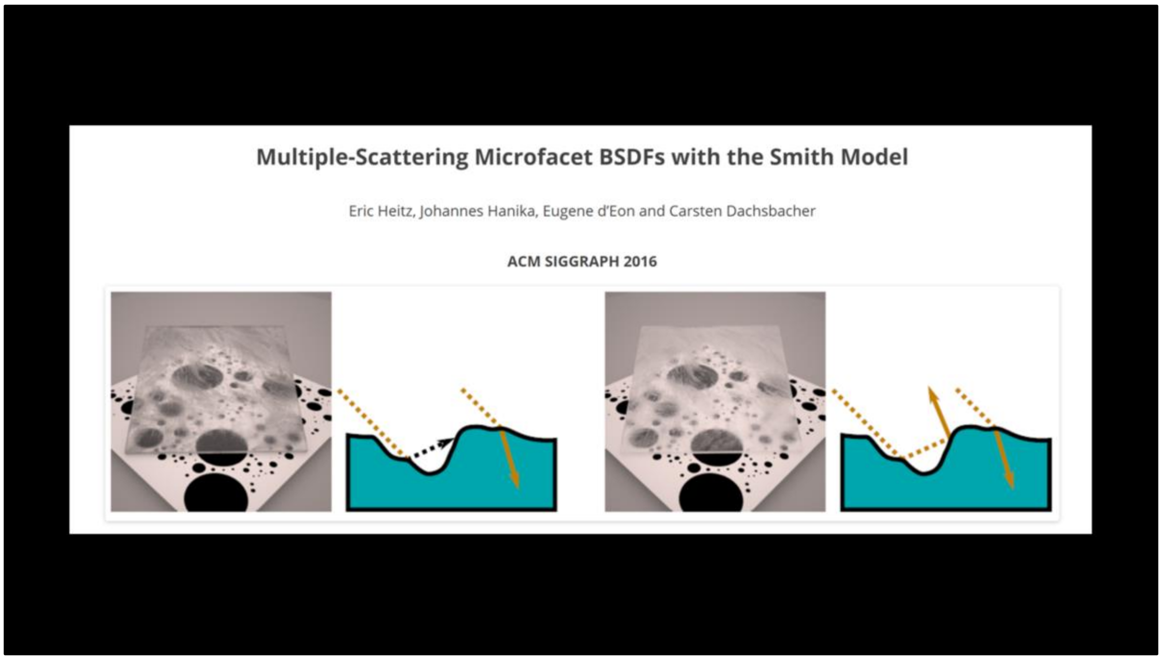
1. Multiscattering is taken into account
2. Diffuse reacts to surface roughness
3. Diffuse depends on the distribution of normals
4. Diffuse and specular are energy conserving

Lambertian diffuse is pretty poor, and we wanted the following improvements:

1. Multiscattering is taken into account – we've learned from specular that we're losing energy from single-scattering, can we solve this for diffuse too?
2. It reacts to the surface roughness. This should give us more interesting materials, and also hopefully more detail preservation in the distance. For instance, we have problems on objects such as rocks and tree trunks with strong normals. As they go into the distance the mip maps get flat and we no longer see the strong diffuse lighting we saw previously. We'd ideally like to do some sort of appearance filtering and this might fit in nicely with our existing appearance filtering for specular.
3. It reacts to the distribution of normals – if we're using a GGX distribution for specular, can we do the same for diffuse?
4. Diffuse vs specular is energy conserving. Maybe this means diffuse is too bright vs specular, which would diminish the amount of specular we're seeing, which is one of the original things we were looking to improve.

Material Advances in Call of Duty: WWII [Chan18]

Thankfully, someone else had come up with a model that fitted all those requirements. Danny Chan from Sledgehammer games gave an excellent presentation at SIGGRAPH last year which had a lot of great ideas for improving our materials in games, but he also included a multiscattering diffuse model that essentially met all our requirements.

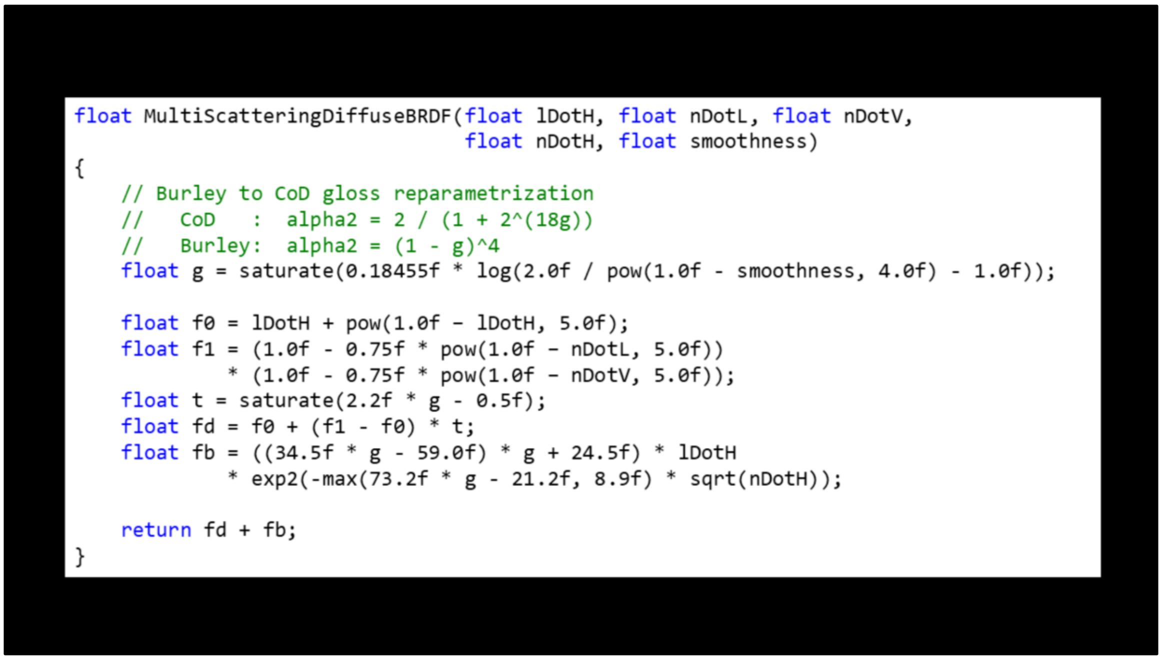


Multiple-Scattering Microfacet BSDFs with the Smith Model

Eric Heitz, Johannes Hanika, Eugene d'Eon and Carsten Dachsbacher

ACM SIGGRAPH 2016

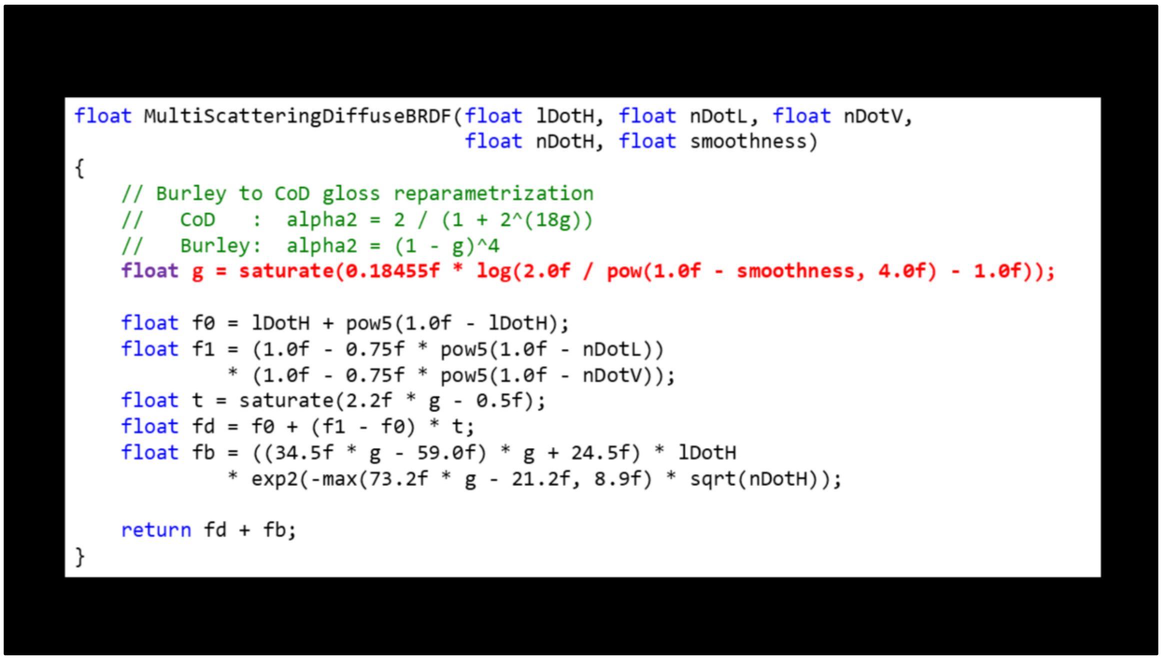
The base of the new diffuse model goes back to the same paper that inspired our interest in multiple scattering specular BRDFs. Danny Chan used the source code provided to generate data for a multiple-scattering diffuse model, and then found a fit.



```
float MultiScatteringDiffuseBRDF(float lDotH, float nDotL, float nDotV,
                                 float nDotH, float smoothness)
{
    // Burley to CoD gloss reparametrization
    //   CoD    :  alpha2 = 2 / (1 + 2^(18g))
    //   Burley:  alpha2 = (1 - g)^4
    float g = saturate(0.18455f * log(2.0f / pow(1.0f - smoothness, 4.0f) - 1.0f));

    float f0 = lDotH + pow(1.0f – lDotH, 5.0f);
    float f1 = (1.0f - 0.75f * pow(1.0f – nDotL, 5.0f))
             * (1.0f - 0.75f * pow(1.0f – nDotV, 5.0f));
    float t = saturate(2.2f * g - 0.5f);
    float fd = f0 + (f1 - f0) * t;
    float fb = ((34.5f * g - 59.0f) * g + 24.5f) * lDotH
             * exp2(-max(73.2f * g - 21.2f, 8.9f) * sqrt(nDotH));

    return fd + fb;
}
```

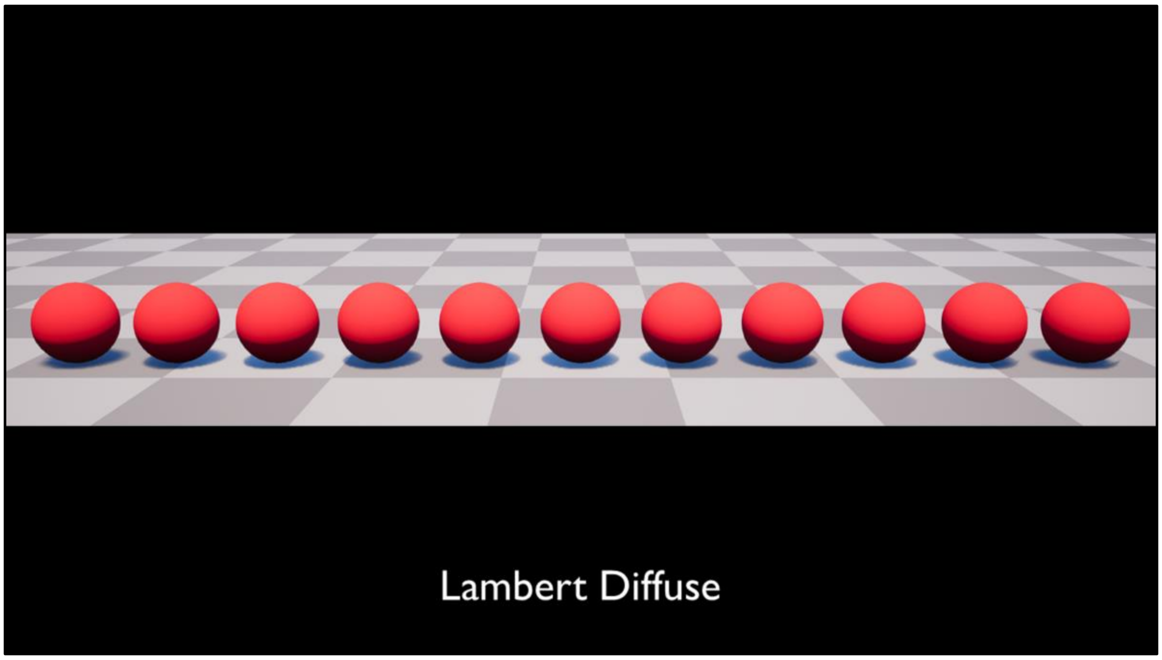
The fitting process as explained in the presentation is pretty complex, but at the end it results in a relatively simple function that we can use in our shaders.

[Thanks to William Bussière for this code]



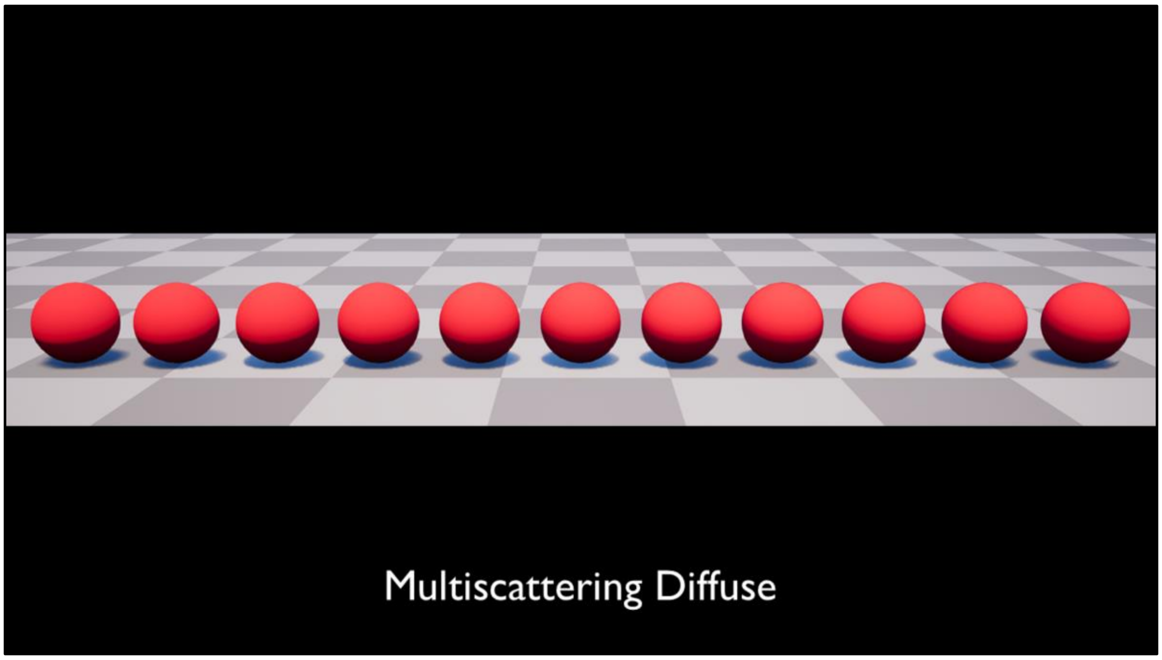
```
float MultiScatteringDiffuseBRDF(float lDotH, float nDotL, float nDotV,
                                 float nDotH, float smoothness)
{
    // Burley to CoD gloss reparametrization
    //   CoD    :  alpha2 = 2 / (1 + 2^(18g))
    //   Burley:  alpha2 = (1 - g)^4
    float g = saturate(0.18455f * log(2.0f / pow(1.0f - smoothness, 4.0f) - 1.0f));

    float f0 = lDotH + pow5(1.0f - lDotH);
    float f1 = (1.0f - 0.75f * pow5(1.0f - nDotL))
             * (1.0f - 0.75f * pow5(1.0f - nDotV));
    float t = saturate(2.2f * g - 0.5f);
    float fd = f0 + (f1 - f0) * t;
    float fb = ((34.5f * g - 59.0f) * g + 24.5f) * lDotH
             * exp2(-max(73.2f * g - 21.2f, 8.9f) * sqrt(nDotH));

    return fd + fb;
}
```

But let me highlight one line. In his paper, Danny fitted against a different parameterisation of surface smoothness than we do. Hence we have to add some extra instructions to convert between the two… or of course, we have to spend the time running the model ourselves and doing our own fit, which would be another significant amount of work.



Let’s look at some results. Here are spheres increasing in smoothness from left to right, which with Lambert diffuse has absolutely no effect – they all look the same.

Now let’s turn the multiscattering diffuse on…


Multiscattering Diffuse

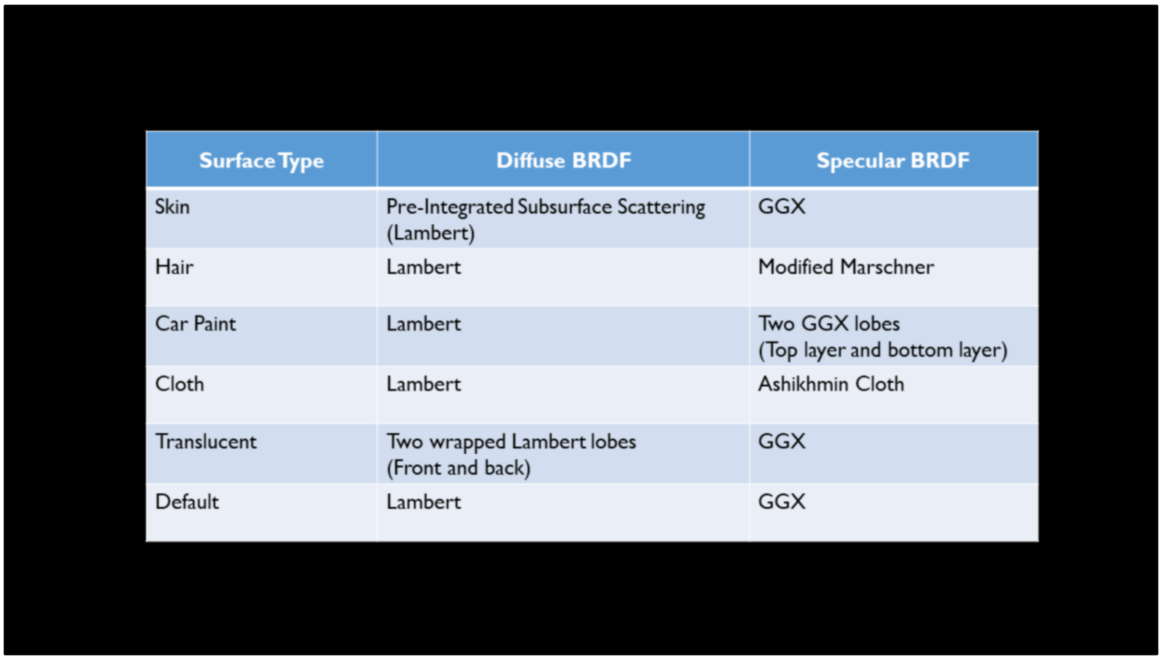
And we see some nice retroreflection happening for the rough spheres, and a softer falloff for the smoother spheres. In fact, one of our art directors always complained about the harsh falloff for diffuse, nearly preferring gamma-space rather than linear-space lighting because of the softness of the falloff. So this looks like it’s going to be something that’s going to help the game.

Another thing that’s interesting though is seeing how the ground plane in these pictures is changing, getting darker with the multiscattering diffuse. I point this out as this is actually one of the biggest effects of multiscattering diffuse you can see in the game. If you think of a postcard shot, with sky and terrain, then this is changing the ratio between those two and affects the whole image. We’ll have to see in the future how this benefits our lighting artists.

“Are we there yet?”

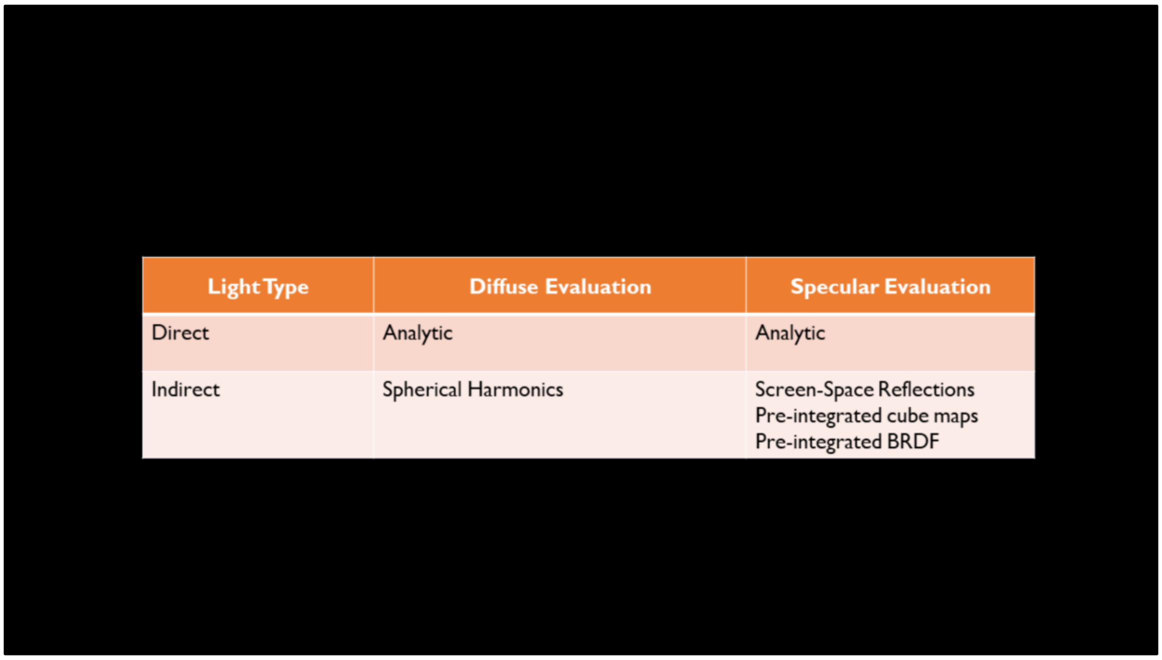
So we have our new diffuse and specular models, so we’re done, right?

Sadly… not… we’re not even close…



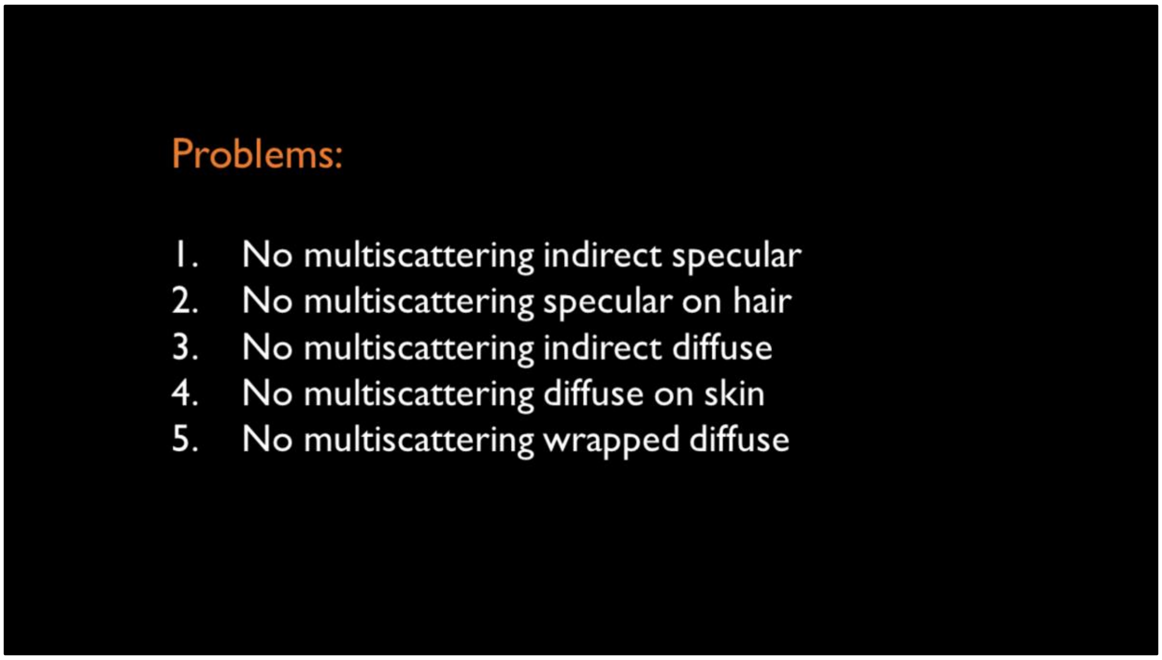
Surface Type	Diffuse BRDF	Specular BRDF
Skin	Pre-Integrated Subsurface Scattering (Lambert)	GGX
Hair	Lambert	Modified Marschner
Car Paint	Lambert	Two GGX lobes (Top layer and bottom layer)
Cloth	Lambert	Ashikhmin Cloth
Translucent	Two wrapped Lambert lobes (Front and back)	GGX
Default	Lambert	GGX

Far Cry actually has lots of BRDFs that we need to replace, not just GGX for specular and Lambert for diffuse. So if we want to upgrade our diffuse lighting, then ideally we'd update our pre-integrated subsurface scattering too, plus we need to find a way to mimic wrapped lighting for Lambert. For specular, if we want to be energy-preserving in all cases then we might need to do it for hair and cloth too.



Light Type	Diffuse Evaluation	Specular Evaluation
Direct	Analytic	Analytic
Indirect	Spherical Harmonics	Screen-Space Reflections Pre-integrated cube maps Pre-integrated BRDF

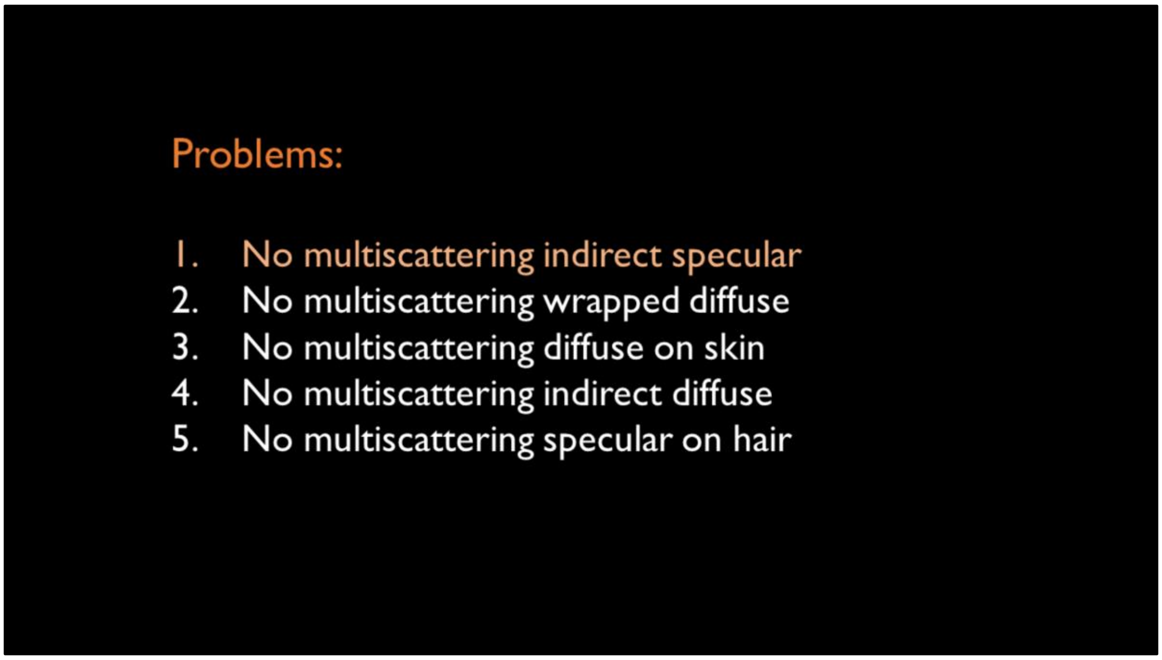
But we also have different light types. For direct lighting we have simple analytic evaluation, this is straightforward. But we also have spherical harmonics and cube maps for indirect lighting. Frankly, we're not always great at making our indirect lighting obey the BRDFs – we do nothing for evaluating the skin with SH right now, for instance. But it feels like we're missing a huge opportunity to increase the amount of specular lighting we have on direct lighting only, but not on the environment.



Problems:

1. No multiscattering indirect specular
2. No multiscattering specular on hair
3. No multiscattering indirect diffuse
4. No multiscattering diffuse on skin
5. No multiscattering wrapped diffuse

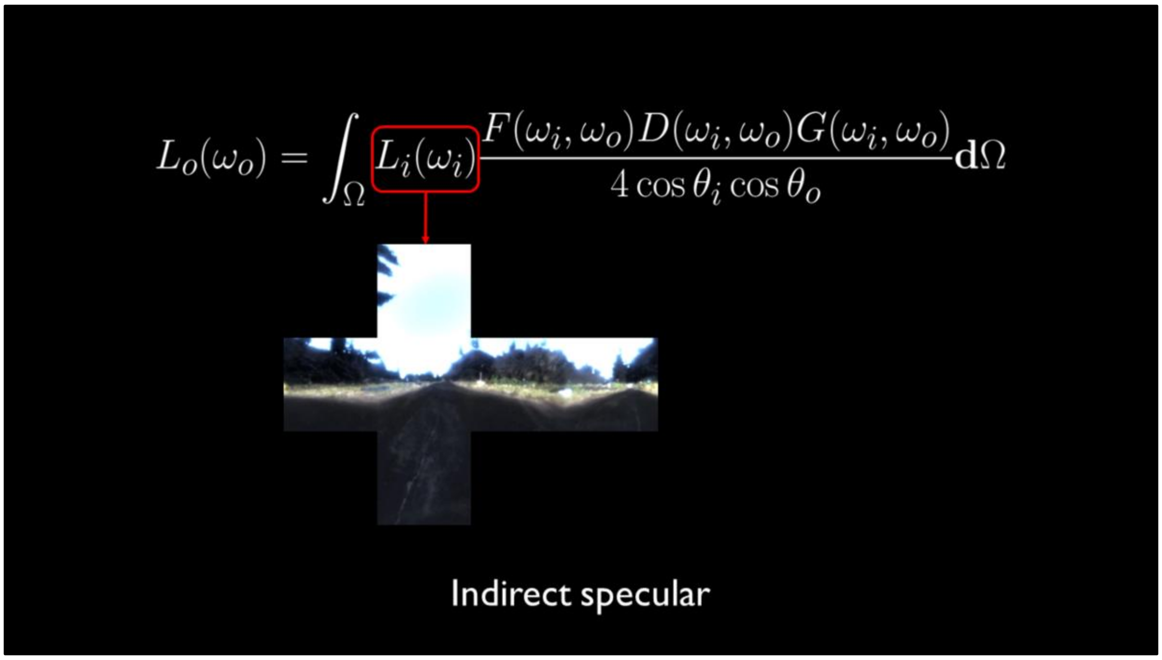
Let’s summarise our problems. We don’t have multiscattering BRDFs on our indirect lighting, our subsurface scattering BRDFs – which includes our wrapped diffuse and skin – and finally we don’t have a solution for hair. I’ll leave off cloth as it’s not so important – the BRDF is a bit of a hack anyway that almost compensates for energy loss in cloth materials. Also note that we don’t have a solution for indirect diffuse on skin currently either in our engine, so there are pre-existing problems that we’re already living with.



Problems:

1. No multiscattering indirect specular
2. No multiscattering wrapped diffuse
3. No multiscattering diffuse on skin
4. No multiscattering indirect diffuse
5. No multiscattering specular on hair

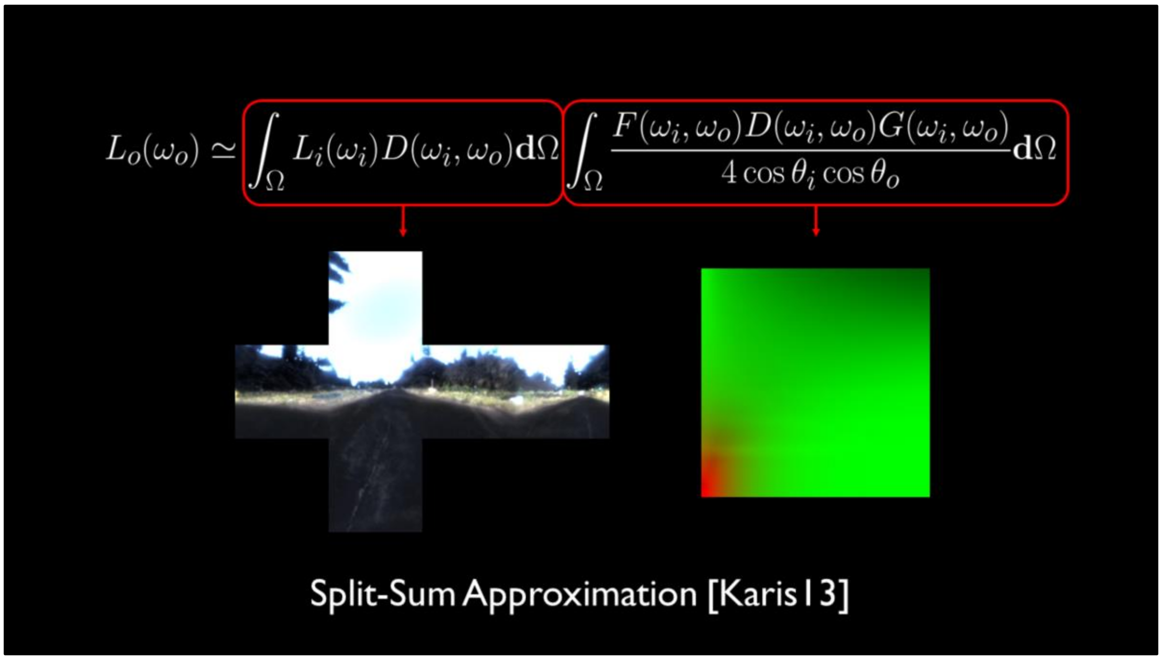
The first problem we'll try to solve is indirect specular, as this will almost certainly have the biggest impact on the game, particularly on our rough metals.



$$L_o(\omega_o) = \int_\Omega \boxed{L_i(\omega_i)} \frac{F(\omega_i,\omega_o)D(\omega_i,\omega_o)G(\omega_i,\omega_o)}{4\cos\theta_i\cos\theta_o}\mathbf{d}\Omega$$

Indirect specular

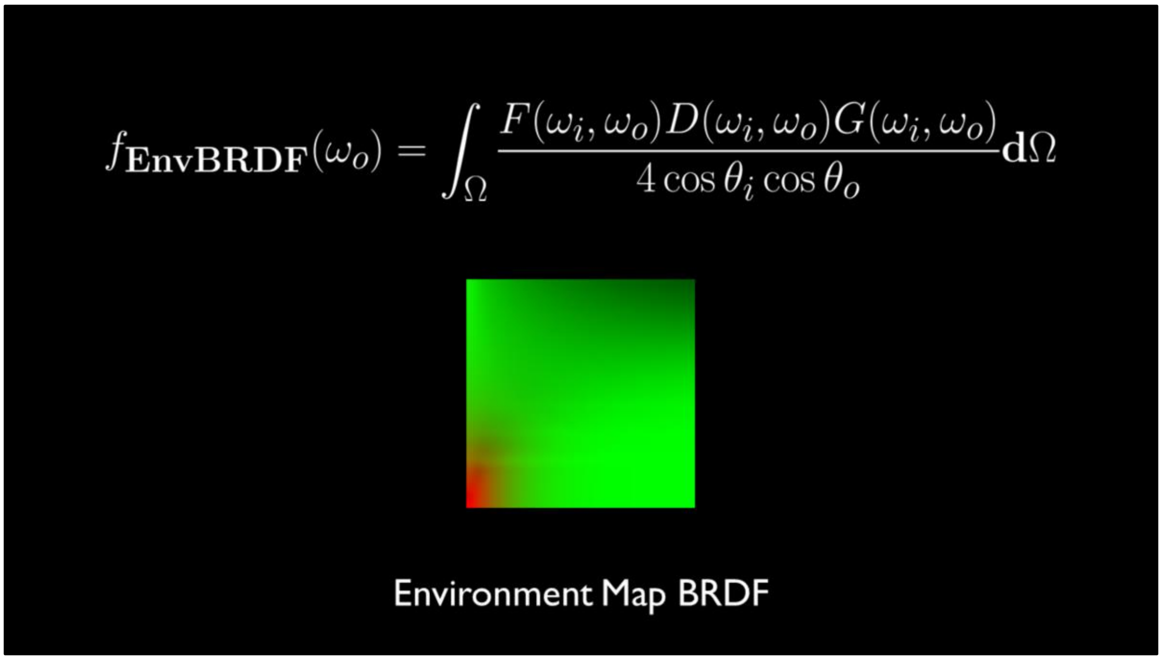
So let’s give it a go. This is the base formula we’re trying to solve for environment lighting. In this case, the environment light comes from a cubemap, and we’re integrating over the hemisphere. The problem is, for games we can’t do this integration per-pixel in real-time – it would require too many samples of the cubemap. So we have some approximations.



$$L_o(\omega_o) \simeq \boxed{\int_\Omega L_i(\omega_i) D(\omega_i, \omega_o) \mathbf{d}\Omega} \boxed{\int_\Omega \frac{F(\omega_i, \omega_o) D(\omega_i, \omega_o) G(\omega_i, \omega_o)}{4 \cos\theta_i \cos\theta_o} \mathbf{d}\Omega}$$

Split-Sum Approximation [Karis13]

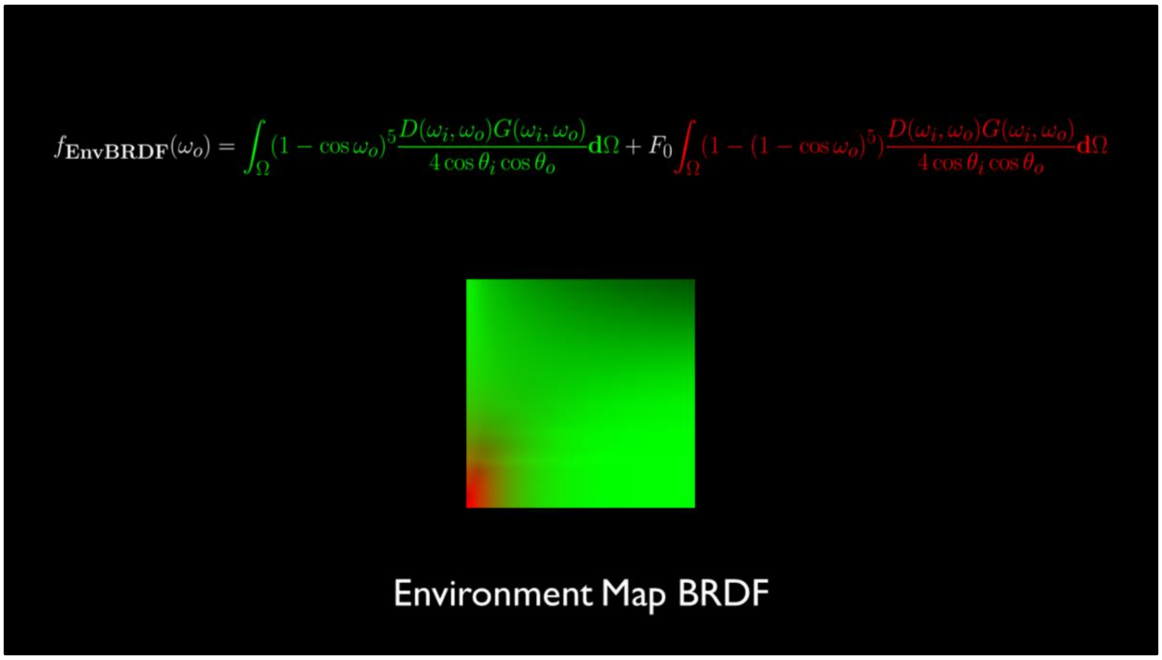
We approximate the result by using this split-sum integral, from Brian Karis' work in 2013. We pre-integrate the cubemap for different roughnesses (with results for higher roughnesses stored in the lower-mip levels of the cubemap), and we integrate the BRDF part into a LUT. This also works pretty well for screen-space reflections too – they just replace the pre-integrated cubemap, and we still use the same LUT.



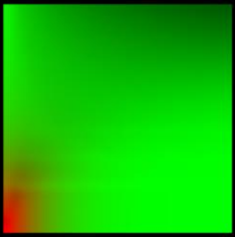
$$f_{\mathbf{EnvBRDF}}(\omega_o) = \int_\Omega \frac{F(\omega_i,\omega_o)D(\omega_i,\omega_o)G(\omega_i,\omega_o)}{4\cos\theta_i\cos\theta_o}\mathbf{d}\Omega$$

Environment Map BRDF

We call the LUT the environment BRDF, and if we use Schlick's Fresnel approximation, we're able to split it into two components…

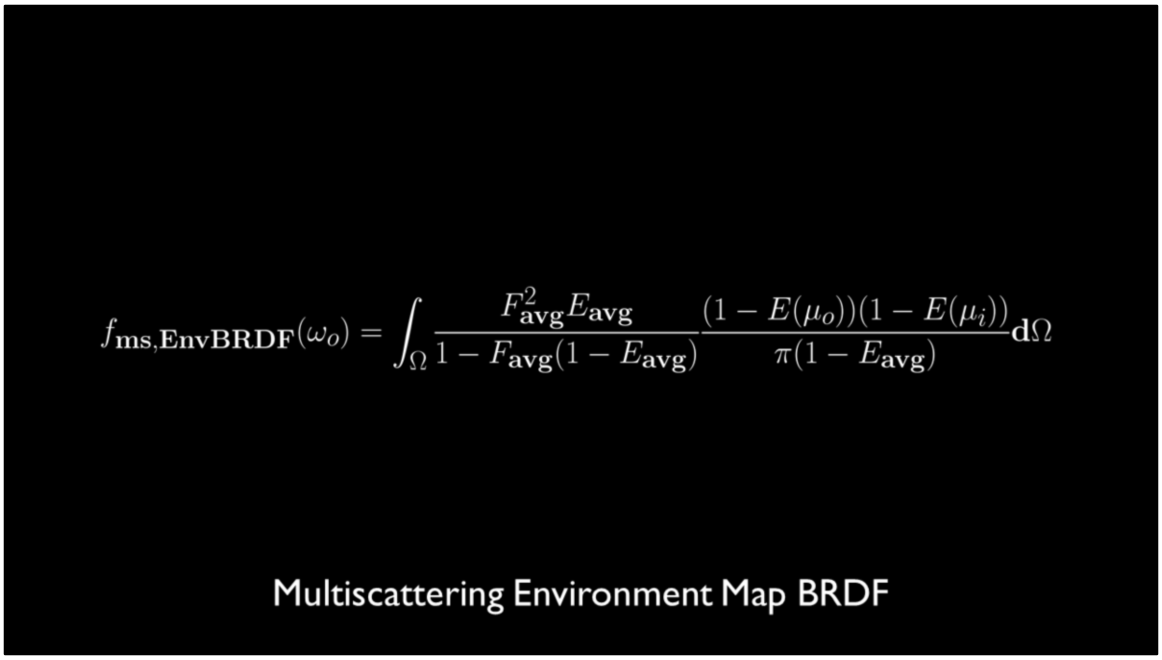


$$f_{\mathbf{EnvBRDF}}(\omega_o) = \int_\Omega (1-\cos\omega_o)^5 \frac{D(\omega_i,\omega_o)G(\omega_i,\omega_o)}{4\cos\theta_i\cos\theta_o}\mathbf{d}\Omega + F_0 \int_\Omega (1-(1-\cos\omega_o)^5) \frac{D(\omega_i,\omega_o)G(\omega_i,\omega_o)}{4\cos\theta_i\cos\theta_o}\mathbf{d}\Omega$$



Environment Map BRDF

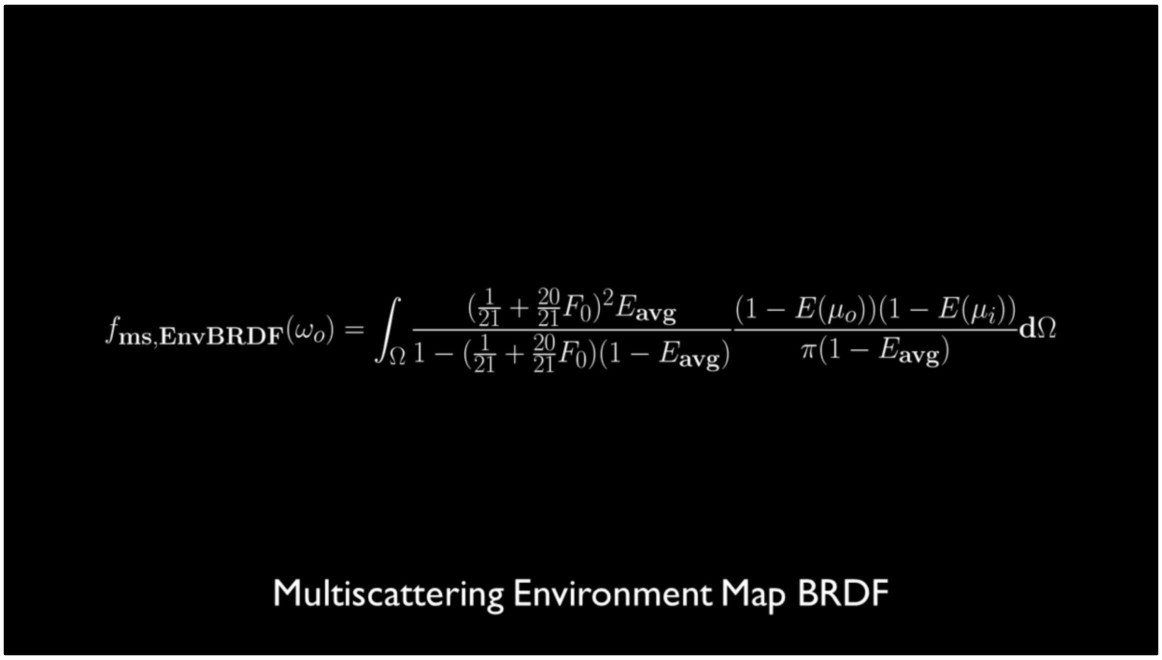
This means that the LUT is just two dimensions – the cosine of the outgoing angle (the view direction), and the roughness of the surface. The F0 term, the specular reflectance, is factored out.



$$f_{\mathbf{ms,EnvBRDF}}(\omega_o) = \int_\Omega \frac{F_{\mathbf{avg}}^2 E_{\mathbf{avg}}}{1 - F_{\mathbf{avg}}(1 - E_{\mathbf{avg}})} \frac{(1 - E(\mu_o))(1 - E(\mu_i))}{\pi(1 - E_{\mathbf{avg}})} \mathbf{d}\Omega$$

Multiscattering Environment Map BRDF

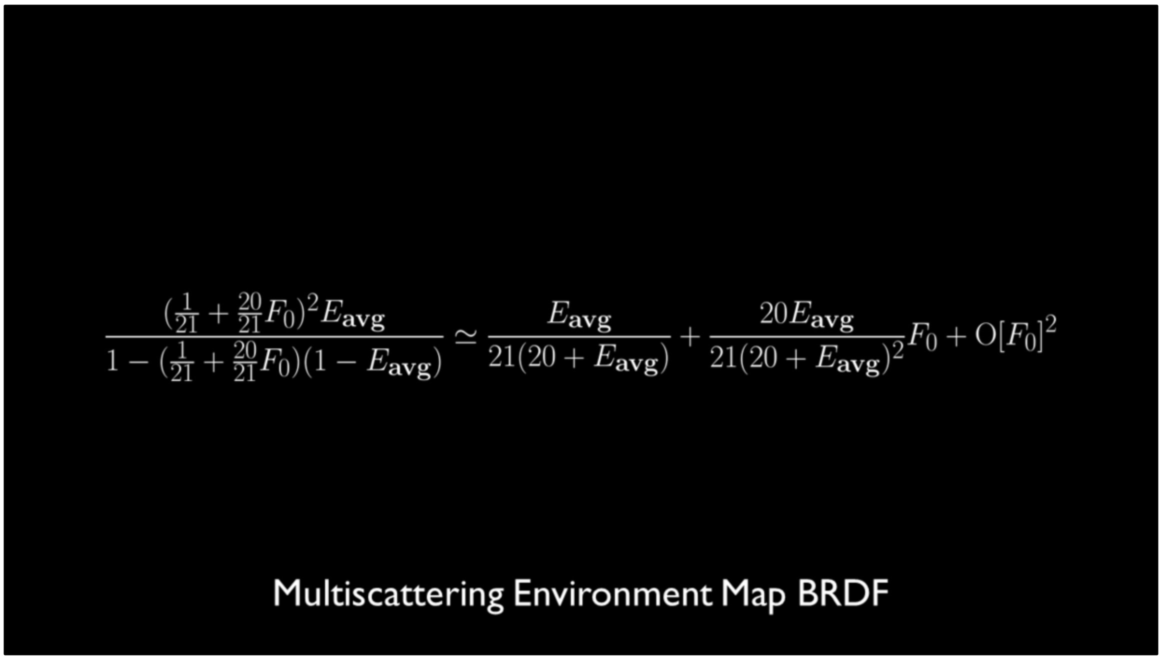
At first I thought adding the multiscattering BRDF into this would be easy. We're preintegrating, so we just add the MS BRDF into the BRDF we're integrating, at no extra cost! So this is what the multiscattering part of the environment BRDF looks like.



$$f_{\mathbf{ms,EnvBRDF}}(\omega_o) = \int_\Omega \frac{(\frac{1}{21}+\frac{20}{21}F_0)^2 E_{\mathbf{avg}}}{1-(\frac{1}{21}+\frac{20}{21}F_0)(1-E_{\mathbf{avg}})} \frac{(1-E(\mu_o))(1-E(\mu_i))}{\pi(1-E_{\mathbf{avg}})} \mathbf{d}\Omega$$

Multiscattering Environment Map BRDF

However, once we expand the average Fresnel term, we realise that things aren't so simple after all. If we're going to just add this to our existing environment BRDF, we want a nice linear dependency on F0… and we don't have that here. So I had an idea, why don't we do a power expansion of the function and see what we get?

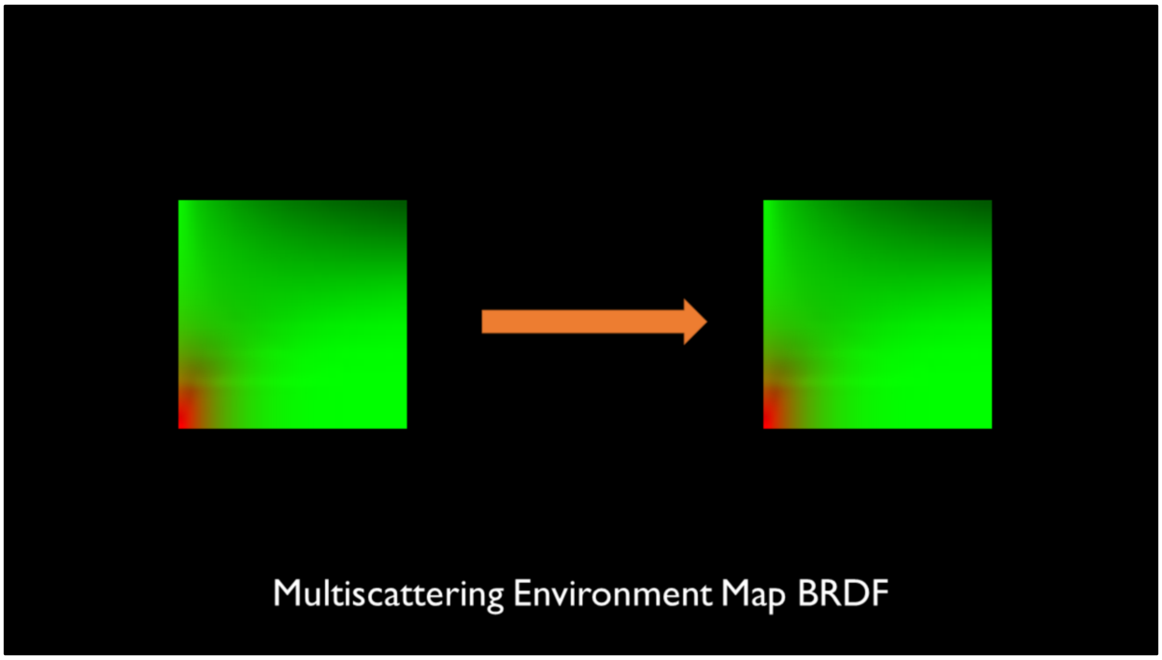


$$\frac{(\frac{1}{21}+\frac{20}{21}F_0)^2 E_{\mathbf{avg}}}{1-(\frac{1}{21}+\frac{20}{21}F_0)(1-E_{\mathbf{avg}})} \simeq \frac{E_{\mathbf{avg}}}{21(20+E_{\mathbf{avg}})} + \frac{20E_{\mathbf{avg}}}{21(20+E_{\mathbf{avg}})^2}F_0 + \mathrm{O}[F_0]^2$$

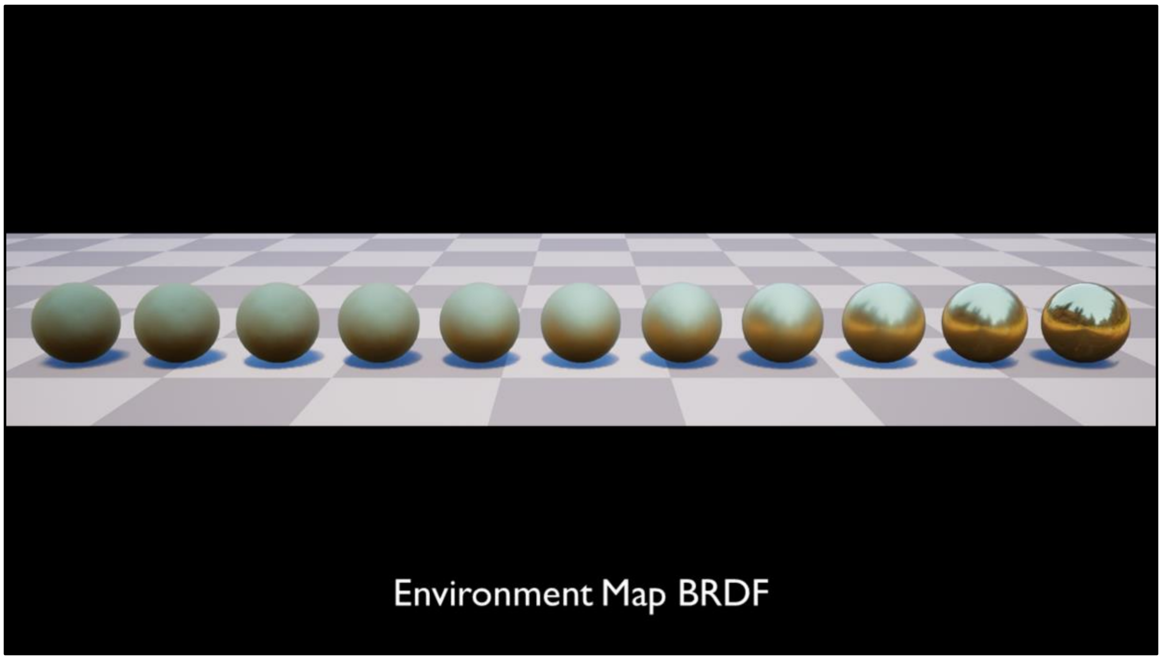
Multiscattering Environment Map BRDF

We get an approximation like this, which is nice and simple and we can add to our environment map BRDF. We could even take more power terms of F0 and bake them into our environment BRDF if we wanted, getting an environment BRDF of three or four components instead of two.

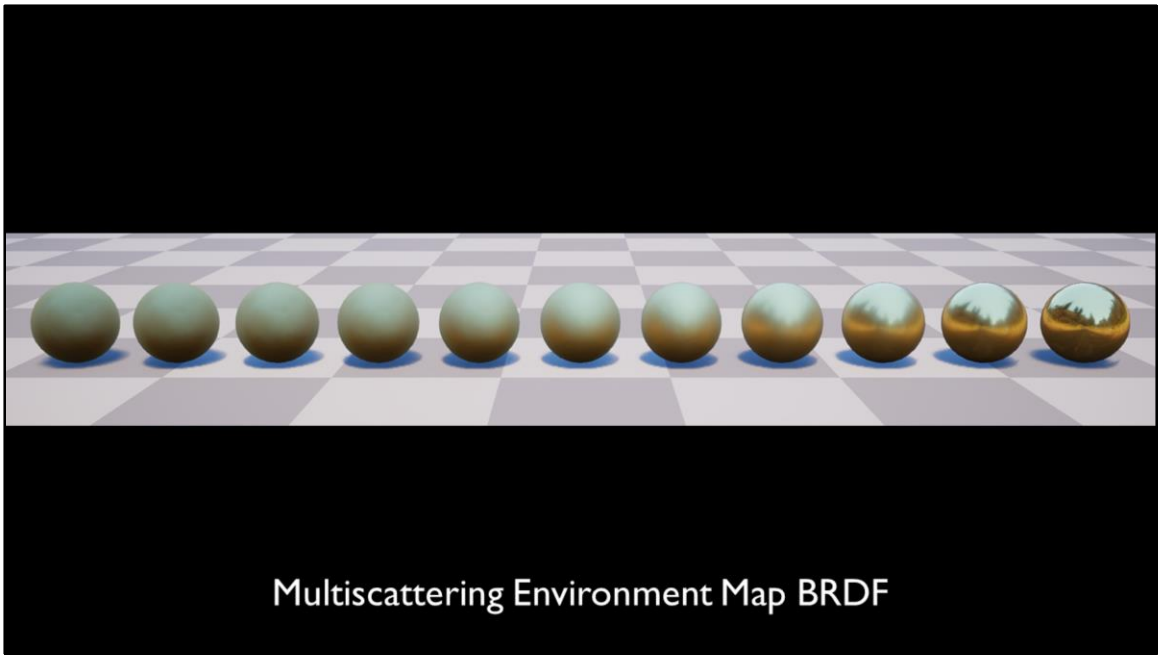
At the time, I thought that two coefficients would be enough. I thought that the F0 term for dielectrics tends to be less than 0.1, so the subsequent power terms wouldn’t really matter… at the time I guess I forgot that multiscattering specular makes the biggest difference for rough *metals*, with much higher specular reflectances…



This means that when we update the environment map BRDF with the multiscattering terms… umm… well, you can barely see any difference. In fact, in 8 bits I’m not sure there is a difference at all, but in 16 bits there definitely is… but you’ll just have to trust me.



If we see how this looks on metals, this is with the standard environment map BRDF…



…and this is with the multiscattering environment BRDF. We see a small difference for metals, but nowhere near the big difference we should be seeing, comparing for our results with a direct light.

So I was puzzling over a solution for this…

JCGT
@JCGT_announce

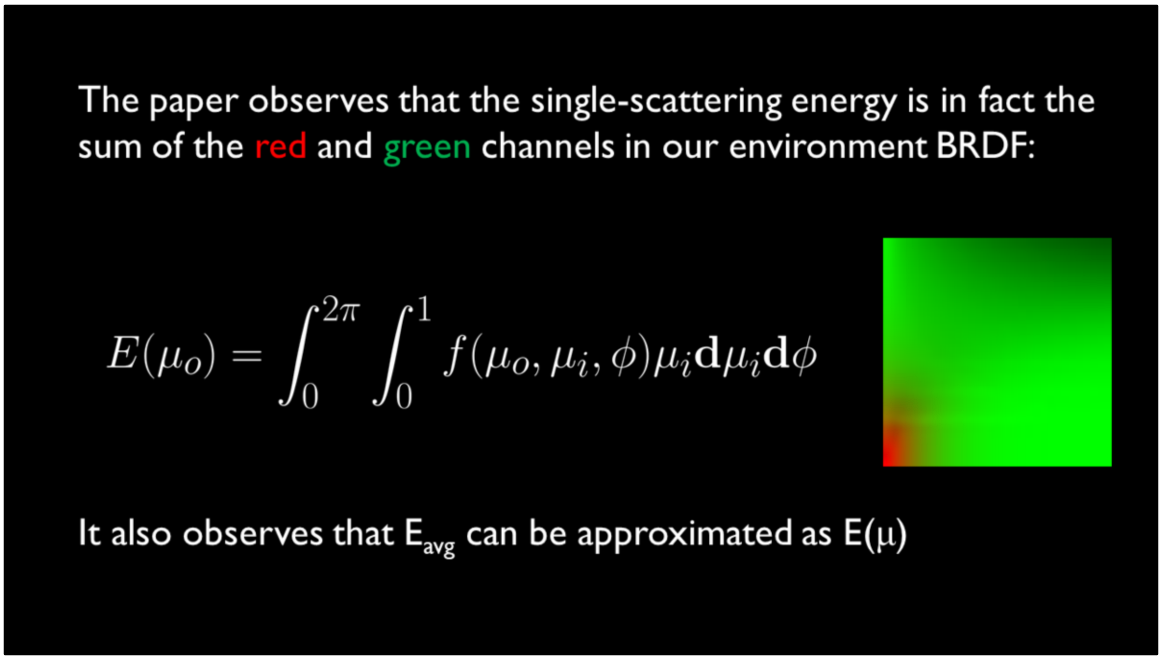
Fdez-Agüera, A Multiple-Scattering Microfacet Model for Real-Time Image Based Lighting. jcgt.org/published/0008...

11:47 AM - 22 Jan 2019

98 Retweets 225 Likes

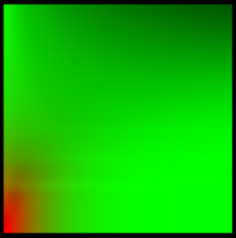
[FdezAgüera19]

… and around the same time this paper came out! It's always really exciting to see someone working on exactly the same things you were working on… and this was a paper solving a really relevant problem for games, so kudos to the author and JCGT. It's also pretty great to extend existing work, making Kulla and Conty's work applicable to real-time image based lighting.



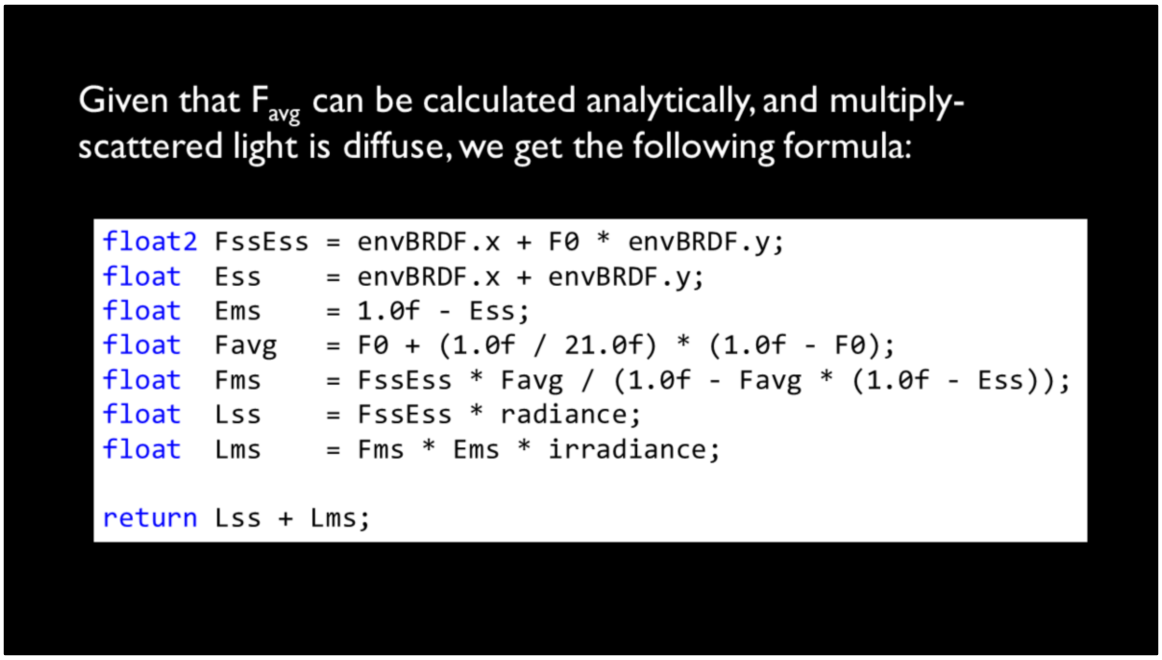
The paper observes that the single-scattering energy is in fact the sum of the red and green channels in our environment BRDF:

$$E(\mu_o) = \int_0^{2\pi} \int_0^1 f(\mu_o, \mu_i, \phi)\mu_i \mathbf{d}\mu_i \mathbf{d}\phi$$



It also observes that E_{avg} can be approximated as $E(\mu)$

This paper makes the clever observation (as have others) that the sum of the red and green channels in the environment BRDF is in fact the same as the outgoing energy in that direction. It also observes that this term can approximate the average energy.

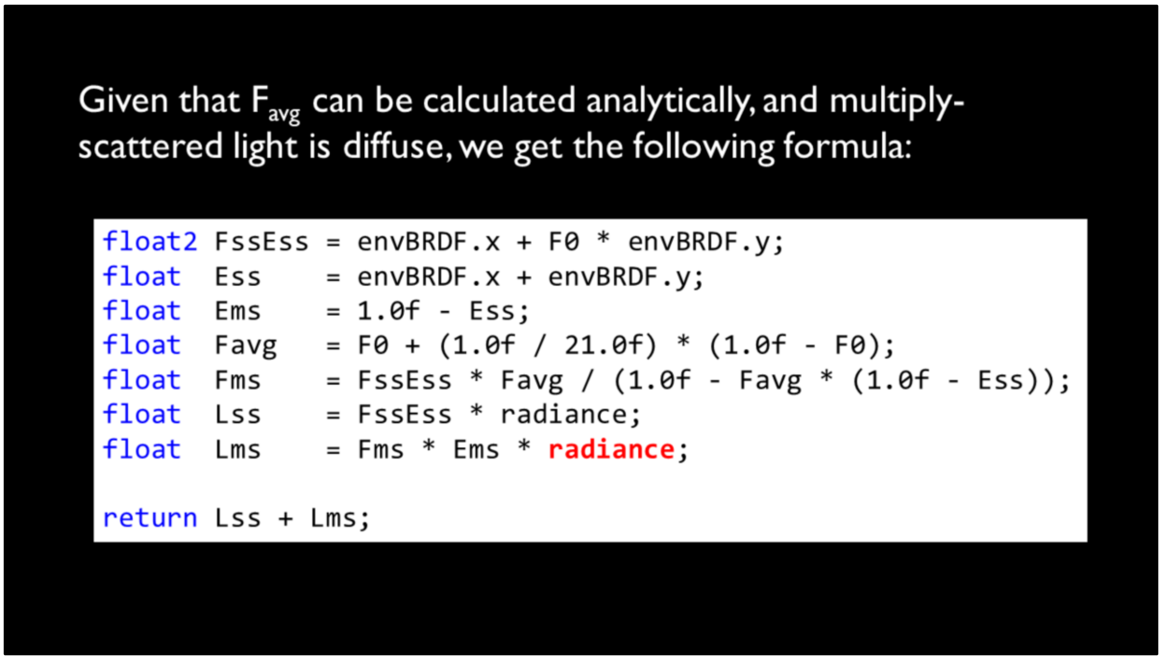


Given that F_{avg} can be calculated analytically, and multiply-scattered light is diffuse, we get the following formula:

```
float2 FssEss = envBRDF.x + F0 * envBRDF.y;
float  Ess    = envBRDF.x + envBRDF.y;
float  Ems    = 1.0f - Ess;
float  Favg   = F0 + (1.0f / 21.0f) * (1.0f - F0);
float  Fms    = FssEss * Favg / (1.0f - Favg * (1.0f - Ess));
float  Lss    = FssEss * radiance;
float  Lms    = Fms * Ems * irradiance;

return Lss + Lms;
```

The paper then arrives at the following simple formula for the multiple-scattering part of the lighting. It helps of course that terms like the average Fresnel are easily calculable. But the paper notes that multiply-scattered light is diffuse, so it decides to take the irradiance of the lighting rather than radiance, recommending sampling a lower mip-level of the cubemap or storing the irradiance in spherical harmonics.

Now, that's all well and good, but I'm a game developer and I'm lazy. I also want to save as many instructions, particularly texture samples, as possible, so I don't like the irradiance part very much.

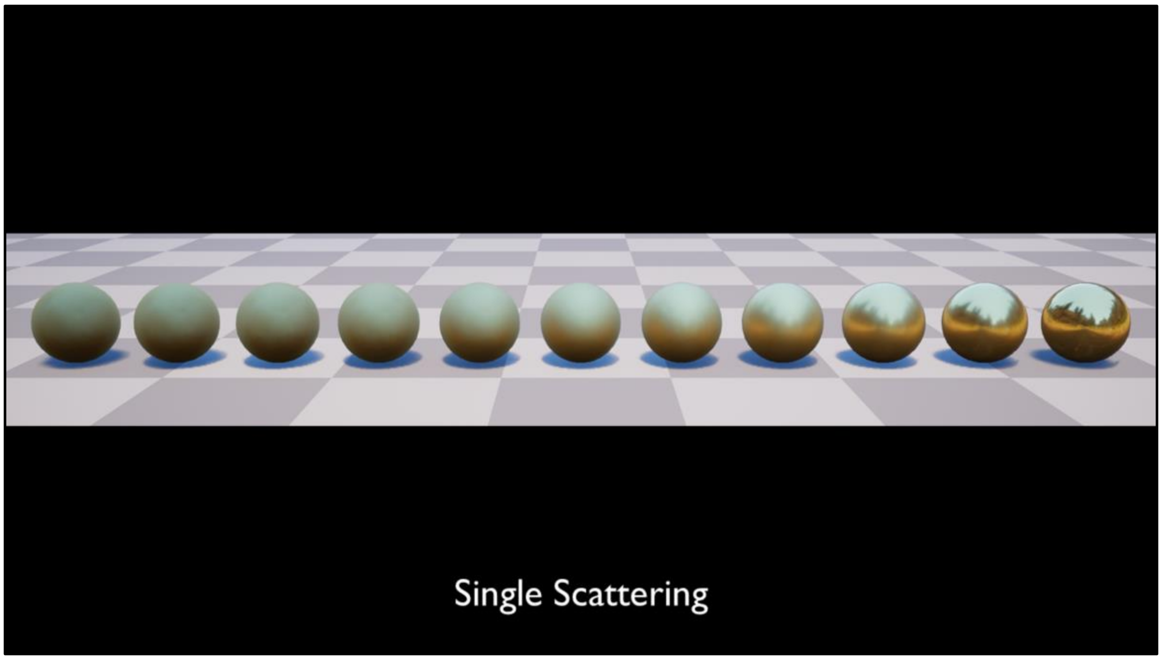


Given that F_{avg} can be calculated analytically, and multiply-scattered light is diffuse, we get the following formula:

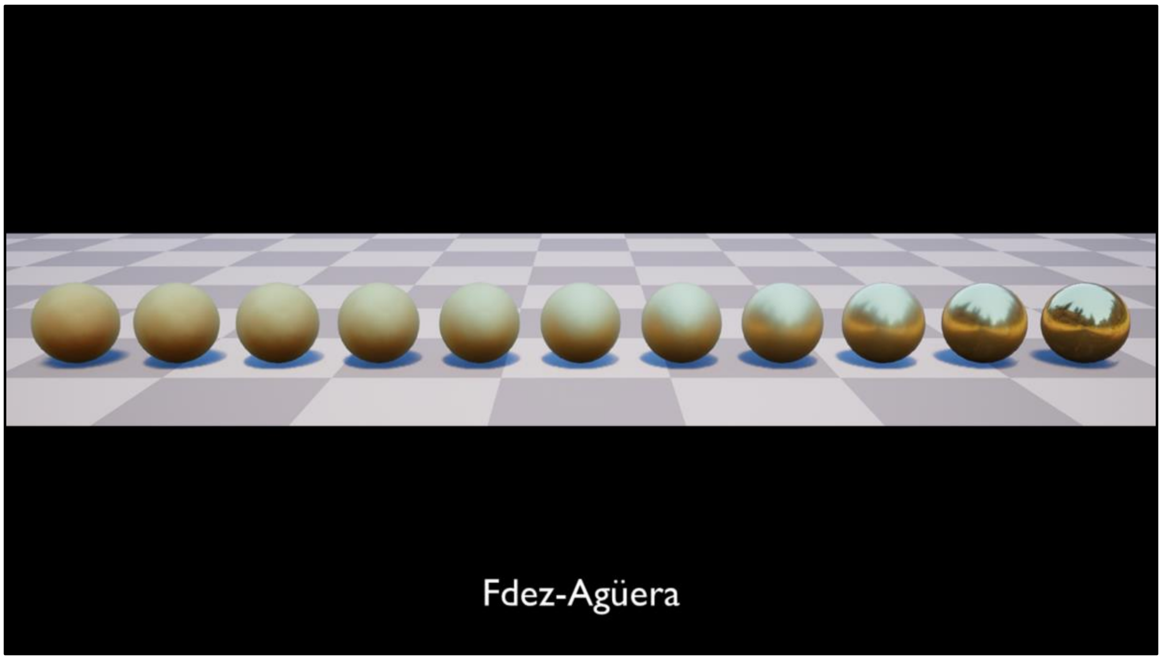
```
float2 FssEss = envBRDF.x + F0 * envBRDF.y;
float  Ess    = envBRDF.x + envBRDF.y;
float  Ems    = 1.0f - Ess;
float  Favg   = F0 + (1.0f / 21.0f) * (1.0f - F0);
float  Fms    = FssEss * Favg / (1.0f - Favg * (1.0f - Ess));
float  Lss    = FssEss * radiance;
float  Lms    = Fms * Ems * radiance;

return Lss + Lms;
```

We're getting the biggest impact of multiscattering on rough metals, this means that specular lighting we're sampling is pretty diffuse anyway, so maybe this doesn't matter too much. Perhaps we could just sub in the radiance for the irradiance and things will still look OK.

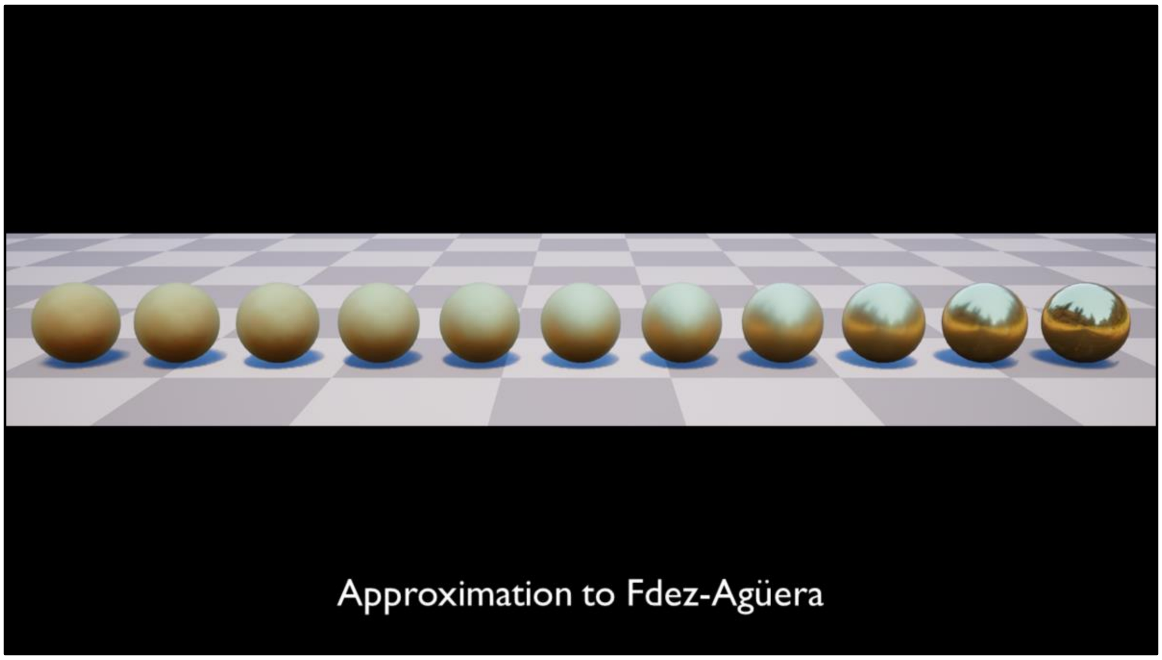
So let's look at some results:



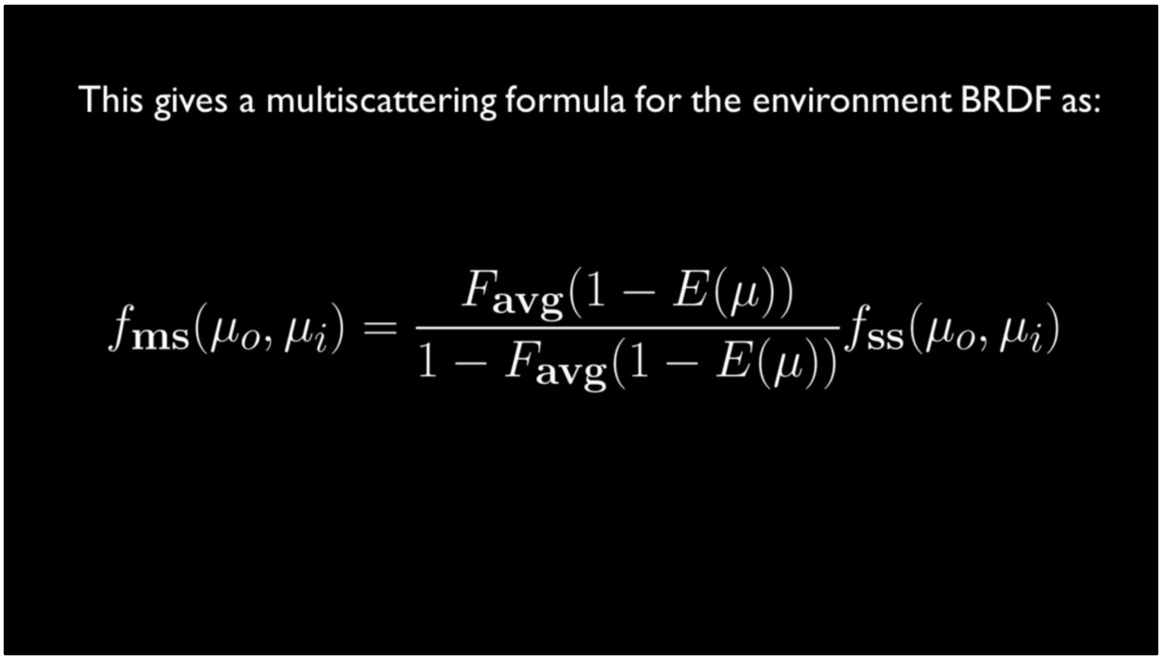
We’ll of course be looking at metals, and this is with the standard environment map BRDF…



This is the Fdez-Agüera approximation which is great, since now we're really compensating for that lost energy.

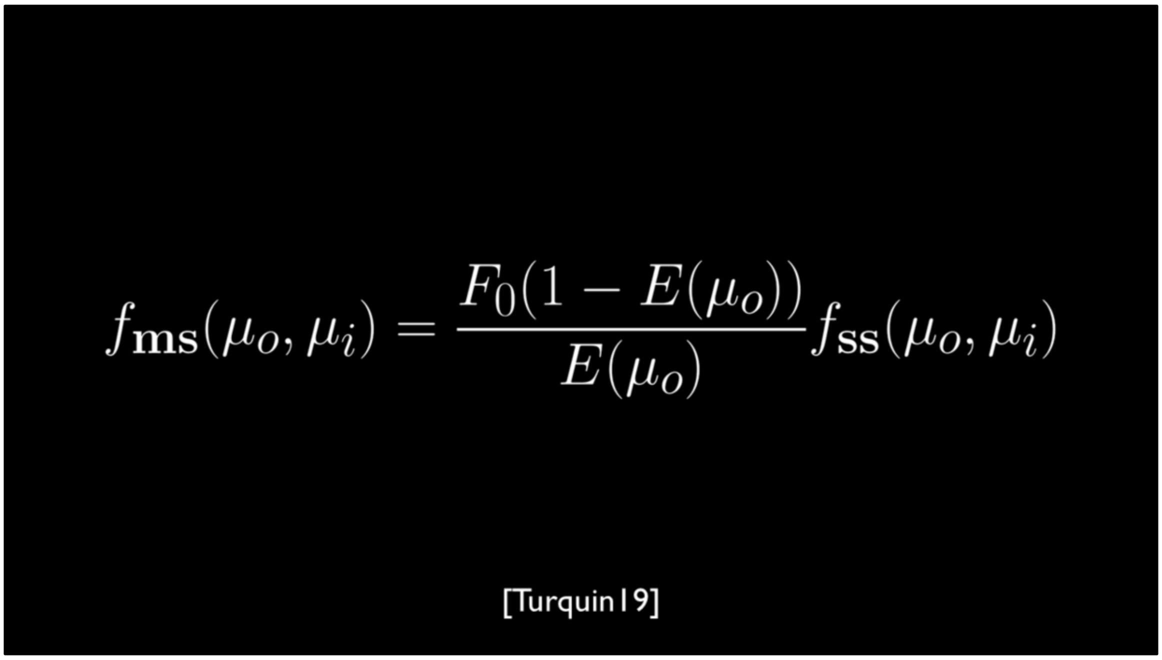


And here’s my approximation to Fdez-Agüera… which… looks virtually no different. It seems like my approximation was valid, at least in my case.


This gives a multiscattering formula for the environment BRDF as:

$$f_{\mathbf{ms}}(\mu_o, \mu_i) = \frac{F_{\mathbf{avg}}(1 - E(\mu))}{1 - F_{\mathbf{avg}}(1 - E(\mu))} f_{\mathbf{ss}}(\mu_o, \mu_i)$$

The great thing about this is it gives us a formula for a multiscattering BRDF that can just scale the single scattering BRDF, which is pretty awesome, as we might be able to apply it to other things. The funny thing with this is that after I'd figured this out…

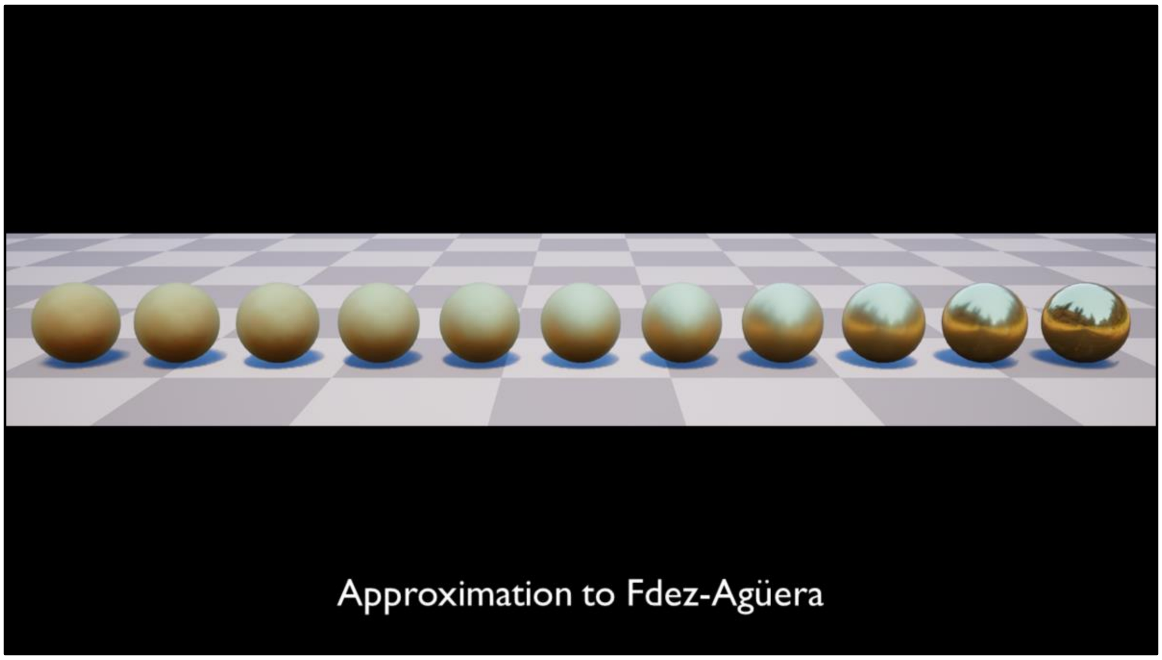


$$f_{\mathbf{ms}}(\mu_o, \mu_i) = \frac{F_0(1 - E(\mu_o))}{E(\mu_o)} f_{\mathbf{ss}}(\mu_o, \mu_i)$$

[Turquin19]

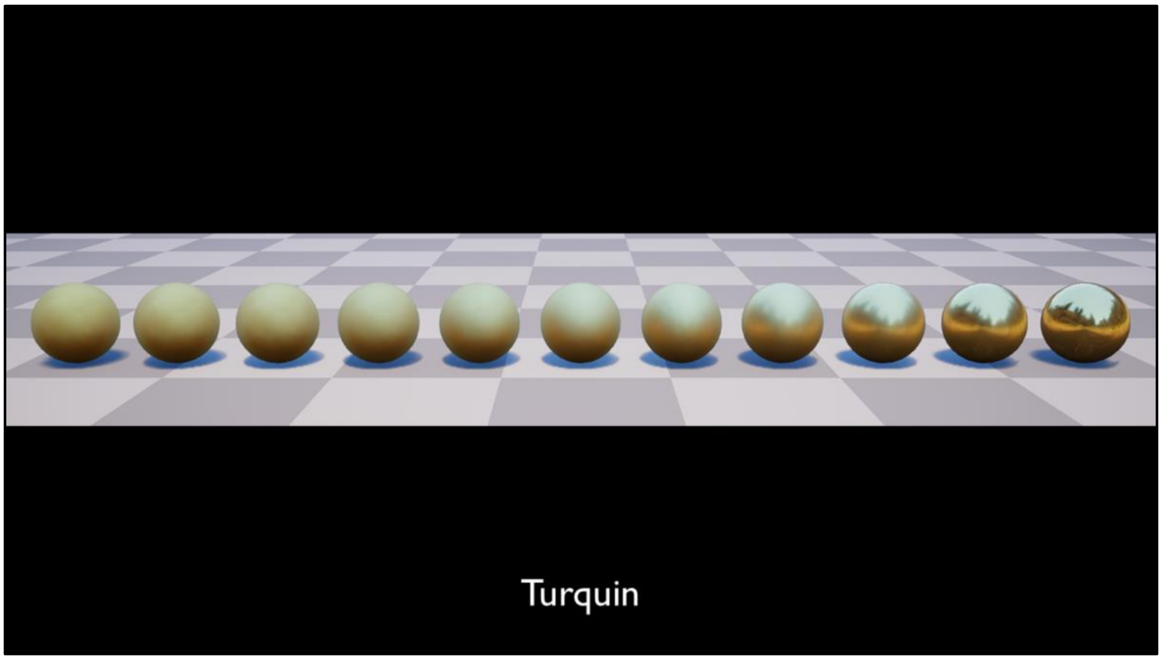
… a tech report from Emmanuel Turquin came out that observed from Heitz’s original paper that the multiscattering lobe looked very much like a scaled version of the single-scattering, and thus came up with this approximation. This isn’t a great deal different from what we just derived from the Fdez-Agüera. Again, this is awesome because we have someone else researching in the same area, coming up with a very pragmatic approximation that we can try out straight away. Another example of building upon some very successful existing work and contributing to the discussion.

Let’s see some results!



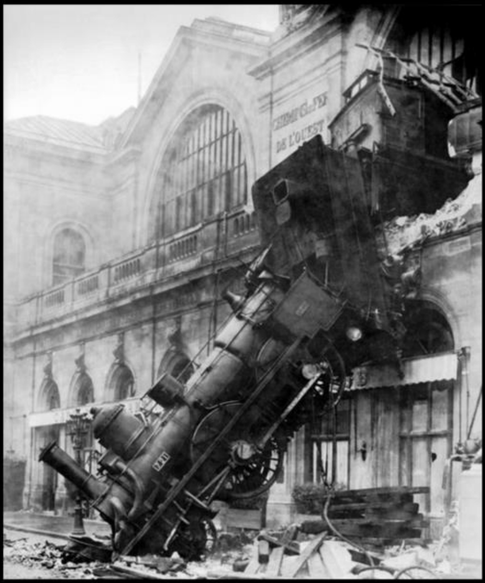
Approximation to Fdez-Agüera

This is my approximation to Fdez-Agüera



And this is Emmanuel Turquin’s. The two are pretty different, particularly in terms of colour, and Emmanuel Turquin’s seems to be a little bit brighter. I haven’t had the time to do a ground-truth comparison, but there are two options here!

I should also mention here that Stephen Hill has recently done some work in the same area, and in a recent blog post he advocates a four-term environment BRDF that solves the issue of multiscattering for environment lighting in a slightly different way. That’s well worth reading if you can, and if you want another solution to the problem.



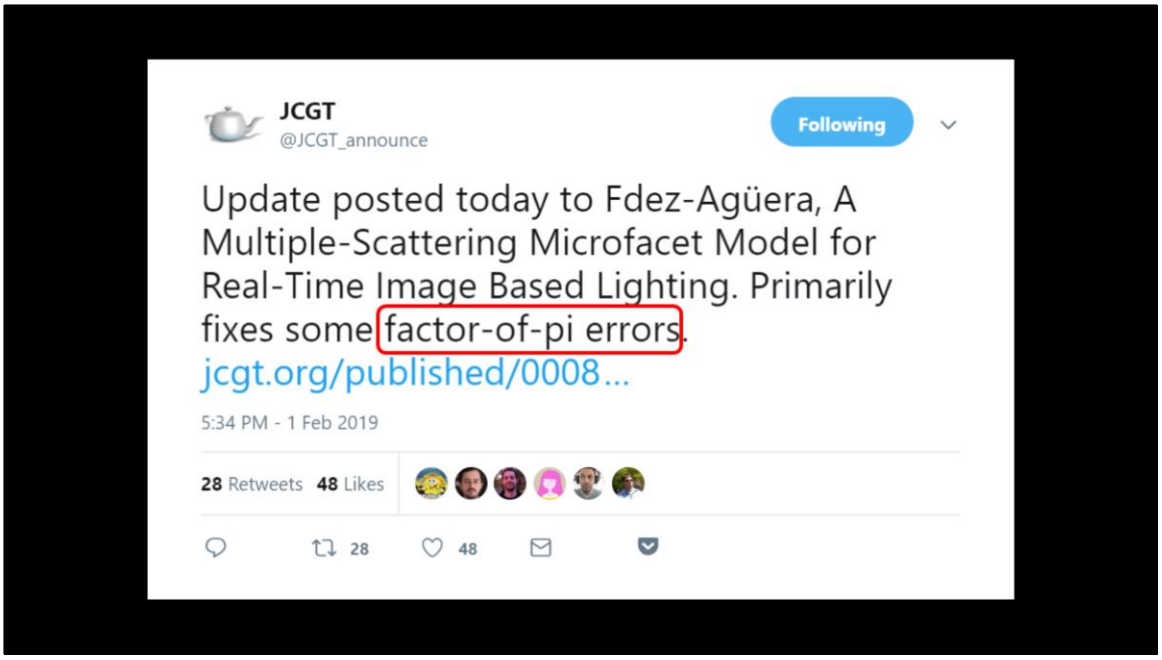
However, when I started, there was something in the Fdez-Agüera paper that very nearly derailed me completely. Let's look at one of the equations, calculating the average Fresnel.

$$F_{\text{avg}} = 2\boxed{\pi} \int_0^{\pi/2} (F_0 + (1 - F_0)(1 - \cos(\theta))^5) \sin\theta \cos\theta d\theta, \qquad (10)$$

$$F_{\text{avg}} = F_0 + \frac{\boxed{\pi}}{21}(1 - F_0). \qquad (11)$$

What really threw me was the added factor of pi, which was present in this and in other equations. When I'd been implementing the Sony Imageworks paper myself, I'd been dividing these integrals by pi as a normalisation factor, which gave me what seemed to be the correct results. But I already had enough self-doubt about my mathematical ability that I was a little unsure about what I'd done. So seeing these equations in a published peer-reviewed paper that were different than mine was really confusing, and I didn't know what to do. Was everything I'd done wrong?

Well, I plugged away, and became more convinced that the factor of pi was wrong. I was going to send an e-mail to the author querying… when I saw this tweet…


JCGT
@JCGT_announce

Following

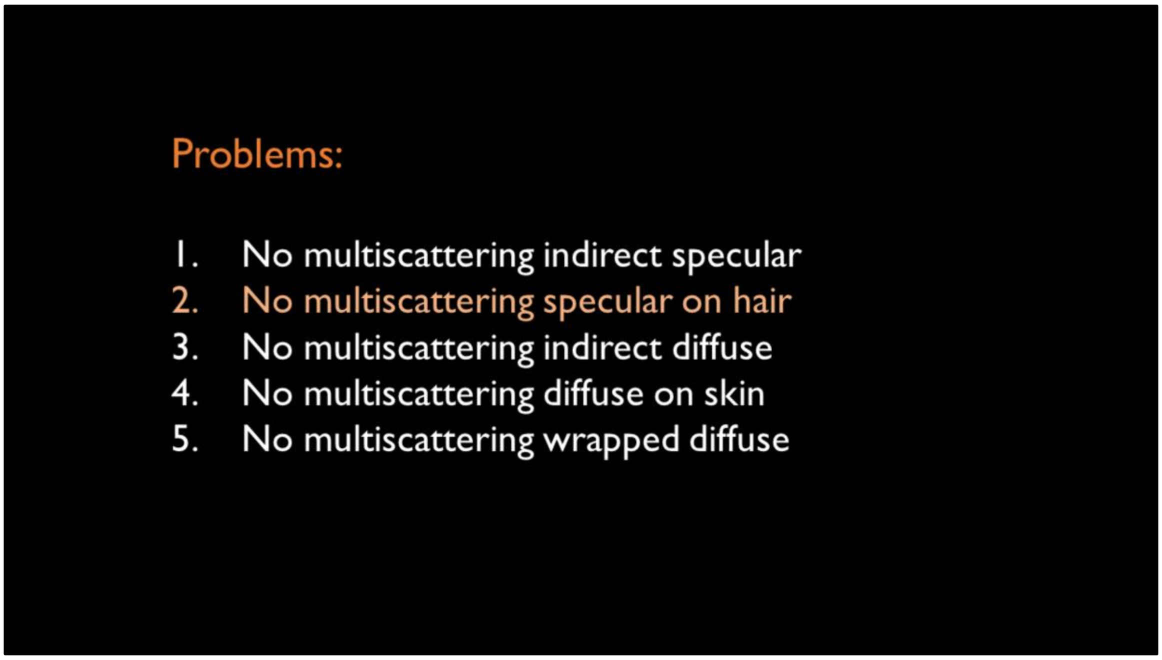
Update posted today to Fdez-Agüera, A Multiple-Scattering Microfacet Model for Real-Time Image Based Lighting. Primarily fixes some factor-of-pi errors. jcgt.org/published/0008…

5:34 PM - 1 Feb 2019

28 Retweets 48 Likes

28 48

I was so relieved! Thank you for the good Samaritan who reported it before I did. ☺



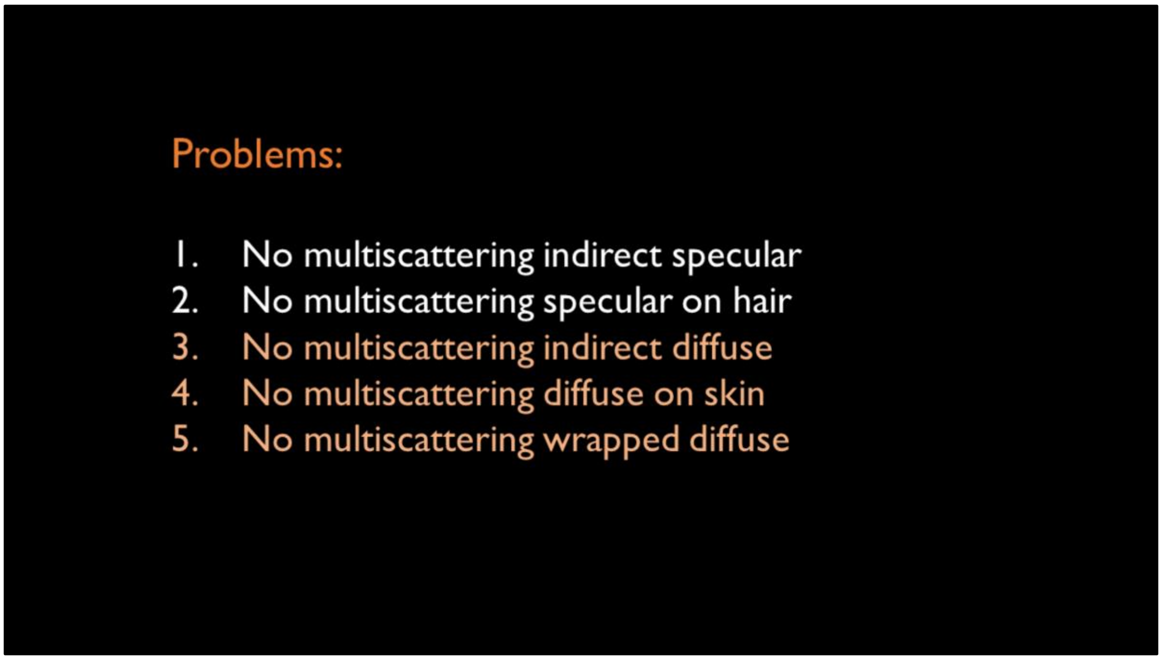
Problems:

1. No multiscattering indirect specular
2. No multiscattering specular on hair
3. No multiscattering indirect diffuse
4. No multiscattering diffuse on skin
5. No multiscattering wrapped diffuse

Let’s return to our list of problems. The next is about multiscattering specular on hair…

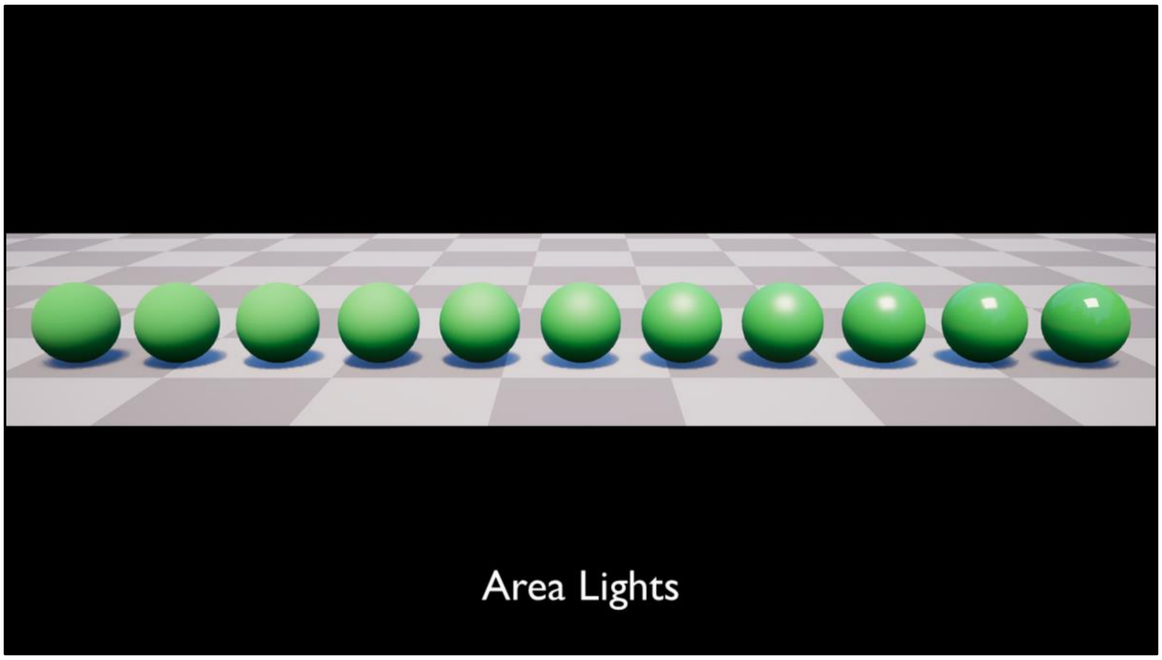
Sadly this is on our TO DO list for now. We could examine using our scaling factor on a single-scattering BRDF, but it's a little more tricky with the transmittance and second specular lobe. So for now I have to leave you hanging, as we're going to have to return to this at a later date.

Image credits: Sue Seecof, https://www.flickr.com/photos/126344637@N05/26013859361, licensed under https://creativecommons.org/licenses/by/2.0/


Problems:

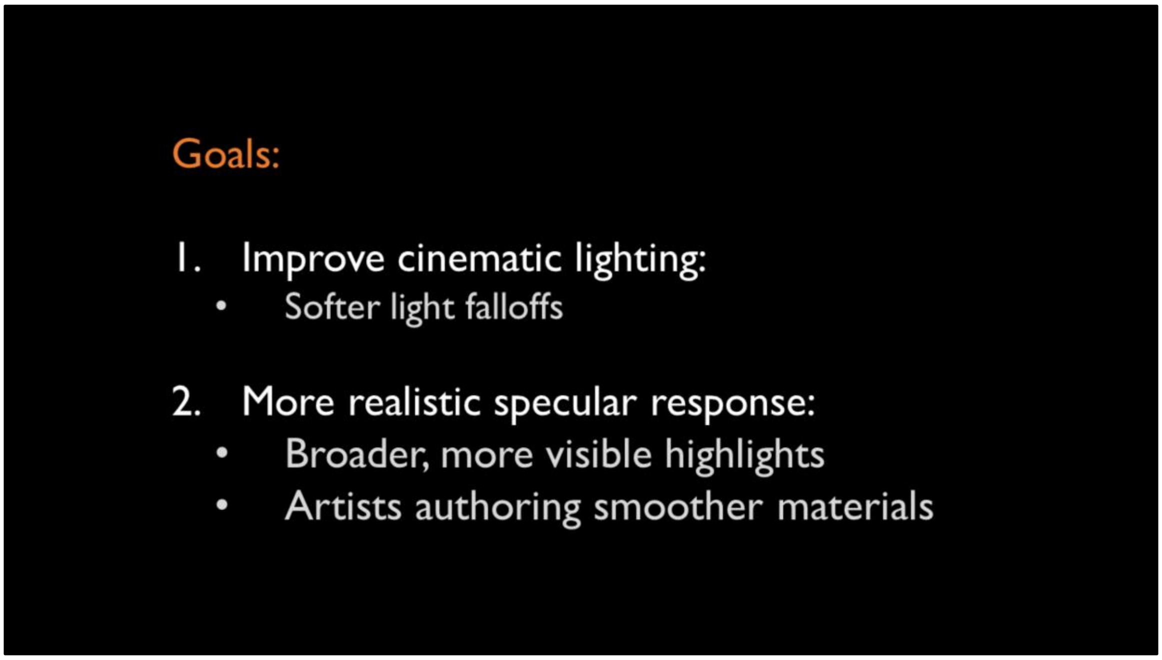
1. No multiscattering indirect specular
2. No multiscattering specular on hair
3. No multiscattering indirect diffuse
4. No multiscattering diffuse on skin
5. No multiscattering wrapped diffuse

But what about our final three problems – multiscattering for our subsurface scattering objects, and for indirect diffuse? Well, it turns out we'll address solutions for these as we move to our next topic…



…which is area lights.

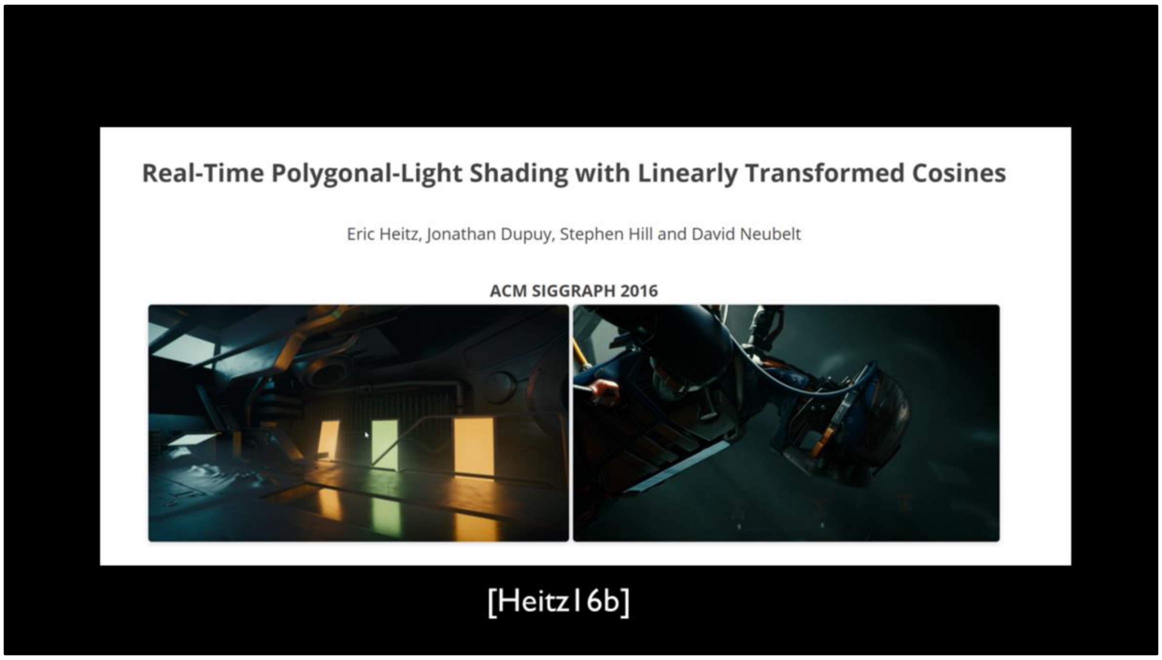
This is something that complements us improving our materials – now we want to look at improving the lights themselves.



Goals:

1. Improve cinematic lighting:
   - Softer light falloffs
2. More realistic specular response:
   - Broader, more visible highlights
   - Artists authoring smoother materials

We'd really like to give our lighting artists more options in cinematics, and having area lights should give them more control over the diffuse falloff via the size of the light. We'll also get more specular, with broader, more visible highlights. This could have an additional benefit where artists will start authoring smoother materials as 100% smooth surfaces will no longer look slightly odd with their infinitesimally small specular highlights.



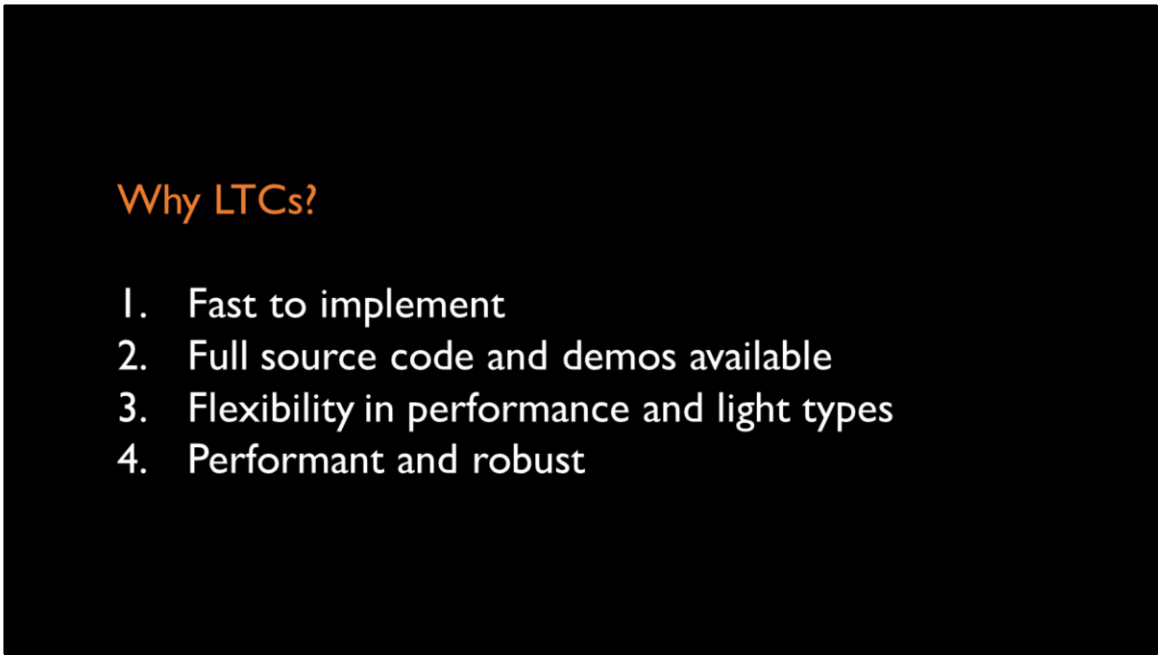
Real-Time Polygonal-Light Shading with Linearly Transformed Cosines

Eric Heitz, Jonathan Dupuy, Stephen Hill and David Neubelt

ACM SIGGRAPH 2016

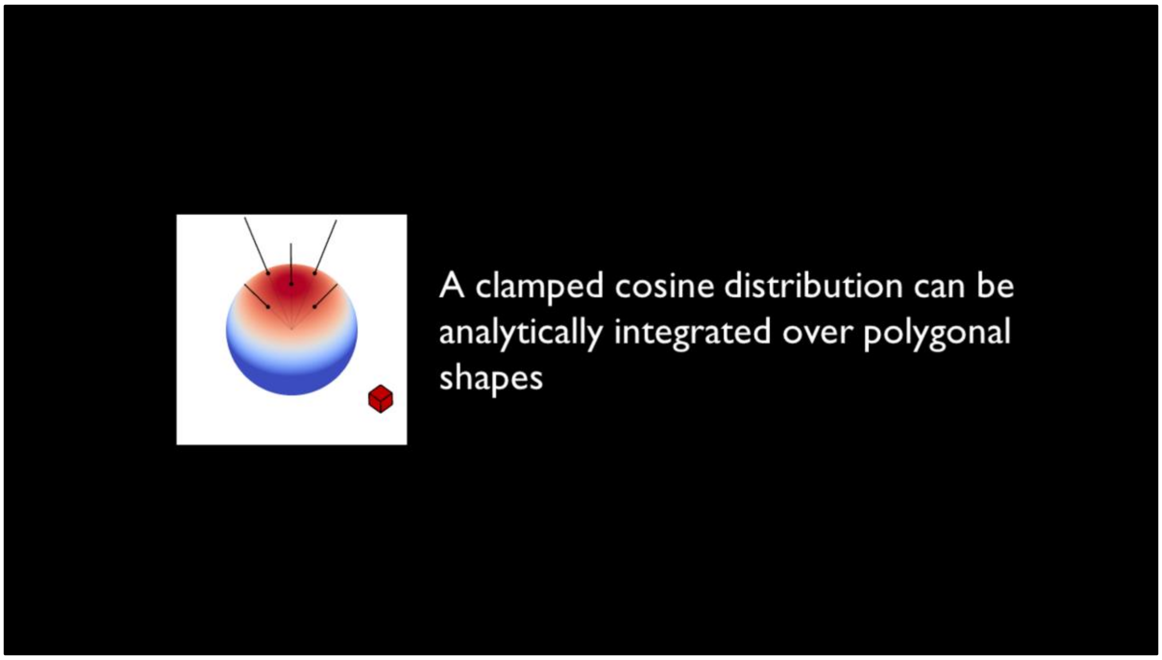
[Heitz16b]

When looking at areas lights, it was a no-brainer to investigate using Linearly-Transformed Cosines, a paper from Eric Heitz and others from 2016, which described a real-time method of shading with polygonal lights, which could technically work with any BRDF.


Why LTCs?

1. Fast to implement
2. Full source code and demos available
3. Flexibility in performance and light types
4. Performant and robust

We chose LTCs because we knew full source code was available, with demos, making them really quick to get in the game and try out. In fact, the most difficult bit was working out that I had to transpose matrices when converting the GLSL demo into HLSL. The full source code would also prove really useful later on when we had to make changes, because flexibility is important too. They already gave us quad lights, disk lights, line lights and texture mapped lights, plus a few scalable performance options, so that gave me a lot of confidence we could make things work. Plus, it was clear from the demos and the code that they were robust and suitable performance-wise for implementation in a game.

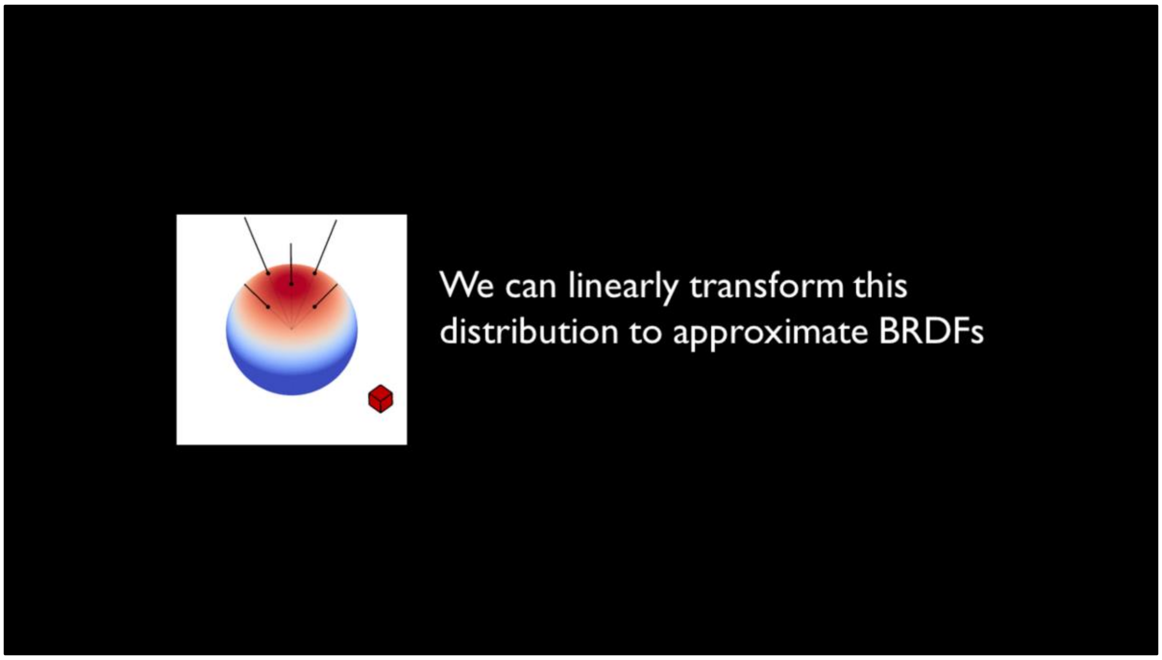


A clamped cosine distribution can be analytically integrated over polygonal shapes

Let’s explain briefly what LTCs are so everyone is on the same page...

LTCs are pretty simple to understand, which is the magic of the whole technique. You start by observing that to evaluate a polygonal area light, you need to integrate the polygon over a spherical distribution which is your BRDF. That’s a potentially unsolvable mathematical problem for many BRDFs, but we can observe that we have a clamped cosine distribution that CAN be analytically integrated over polygonal shapes.

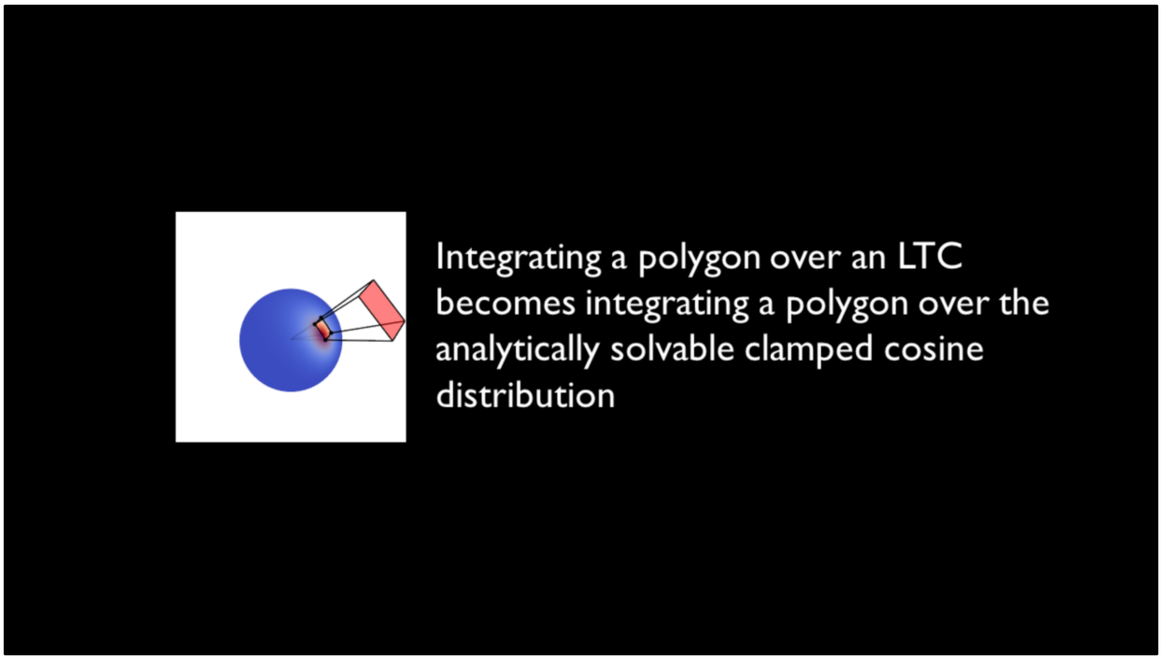
[Thanks to Eric Heitz for the diagram]



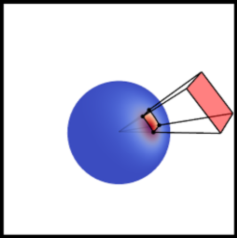
We can linearly transform this distribution to approximate BRDFs

And we can linearly transform this distribution to fit arbitrary BRDFs, while maintaining its ability to be integrated.

[Thanks to Eric Heitz for the diagram]

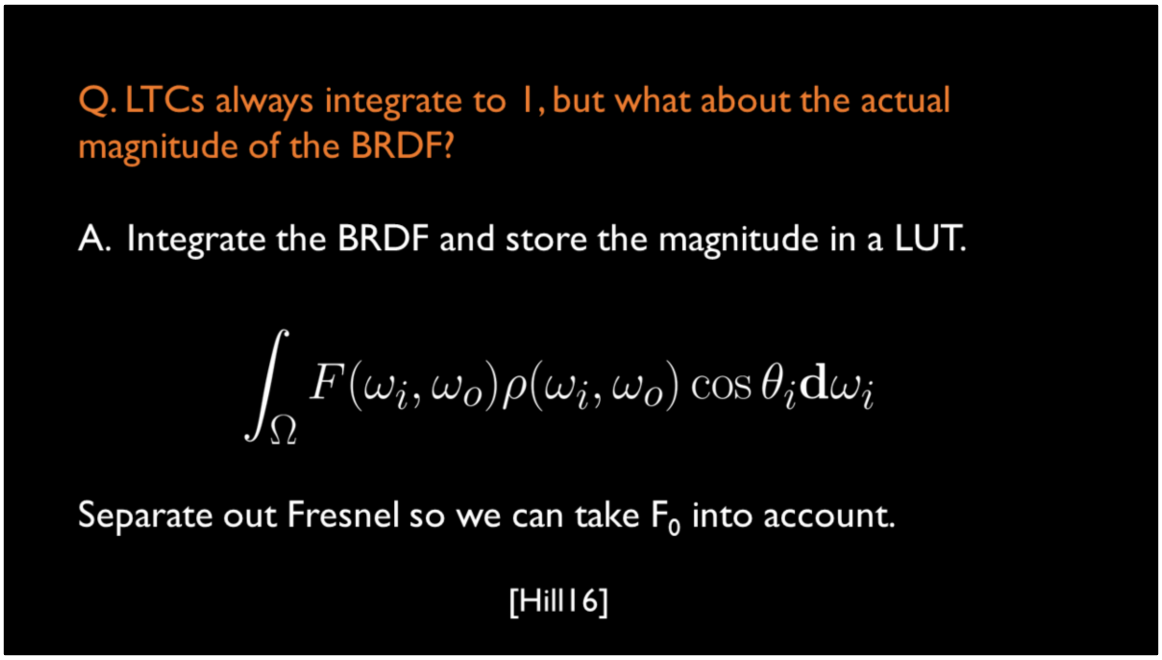


Integrating a polygon over an LTC becomes integrating a polygon over the analytically solvable clamped cosine distribution



So we want to fit our BRDF to a linearly transformed cosine, then when integrating over a polygon, we can apply the inverse transform and the answer just becomes integrating a polygon over a clamped cosine distribution, which we know how to solve.

[Thanks to Eric Heitz for the diagram]



Q. LTCs always integrate to 1, but what about the actual magnitude of the BRDF?

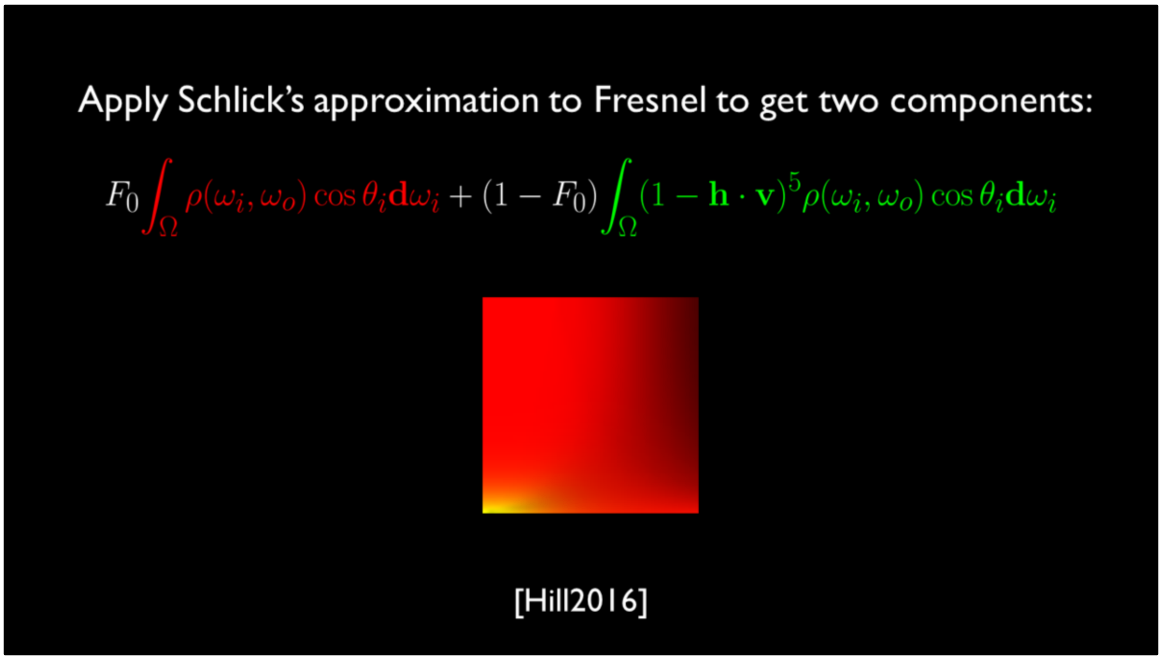
A. Integrate the BRDF and store the magnitude in a LUT.

$$\int_\Omega F(\omega_i, \omega_o)\rho(\omega_i, \omega_o) \cos\theta_i \mathbf{d}\omega_i$$

Separate out Fresnel so we can take F_0 into account.

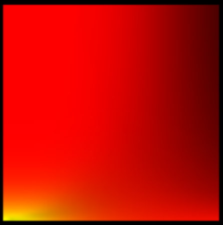
[Hill16]

But LTCs always integrate to 1, whereas BRDFs might integrate to something less than 1. We need to scale by the magnitude of the BRDF, and also take the Fresnel term into account. So we just integrate the BRDF over the hemisphere and store the data in a LUT…



Apply Schlick's approximation to Fresnel to get two components:

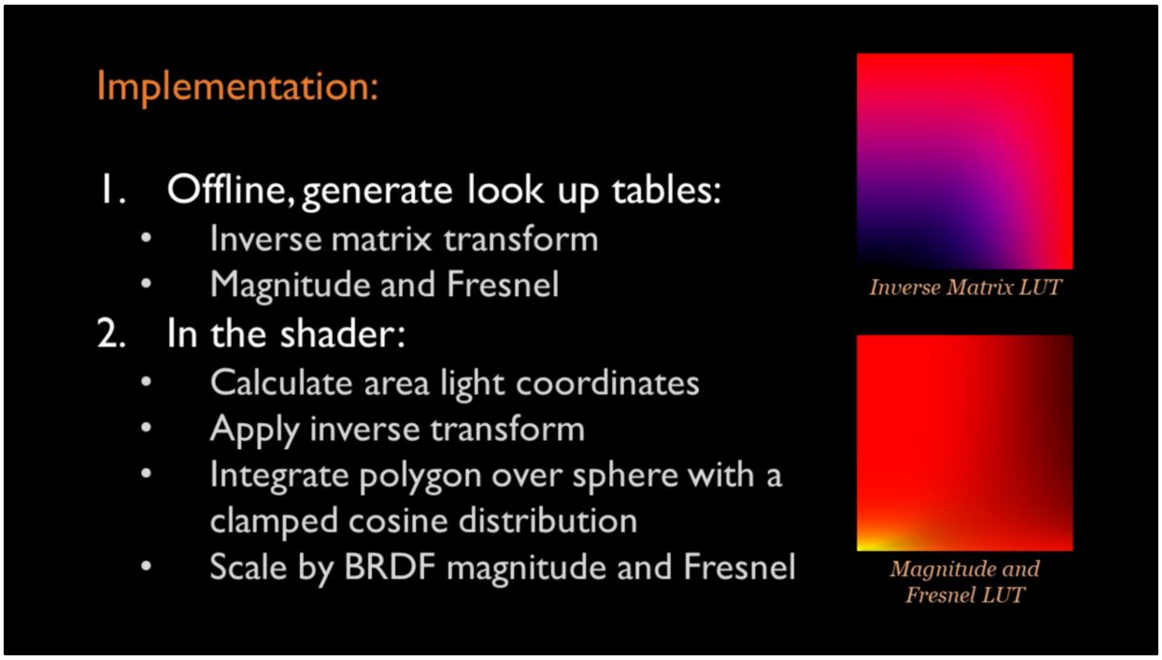
$$F_0 \int_\Omega \rho(\omega_i, \omega_o) \cos\theta_i \mathbf{d}\omega_i + (1 - F_0) \int_\Omega (1 - \mathbf{h} \cdot \mathbf{v})^5 \rho(\omega_i, \omega_o) \cos\theta_i \mathbf{d}\omega_i$$



[Hill2016]

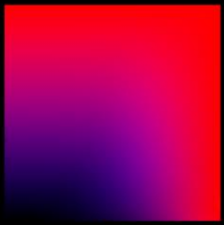
In fact, we can apply Schlick's approximation to Fresnel and get two components, which we can composite together based upon the surface's F0 term and scale the LTC by.

Now, this might look very familiar… precisely because it is. This is a slightly different parameterisation, but a very, very similar idea to that of the environment BRDF. That's going to be very helpful later.

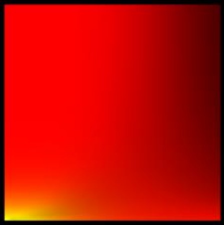


Implementation:

1. Offline, generate look up tables:
   - Inverse matrix transform
   - Magnitude and Fresnel
2. In the shader:
   - Calculate area light coordinates
   - Apply inverse transform
   - Integrate polygon over sphere with a clamped cosine distribution
   - Scale by BRDF magnitude and Fresnel

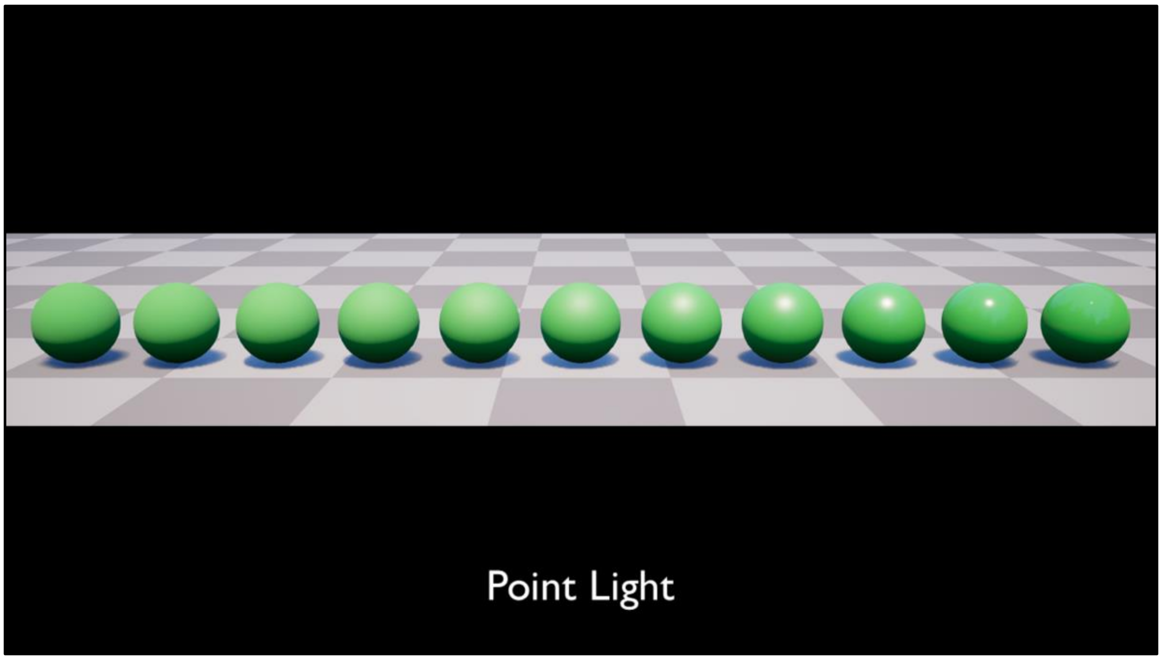


Inverse Matrix LUT

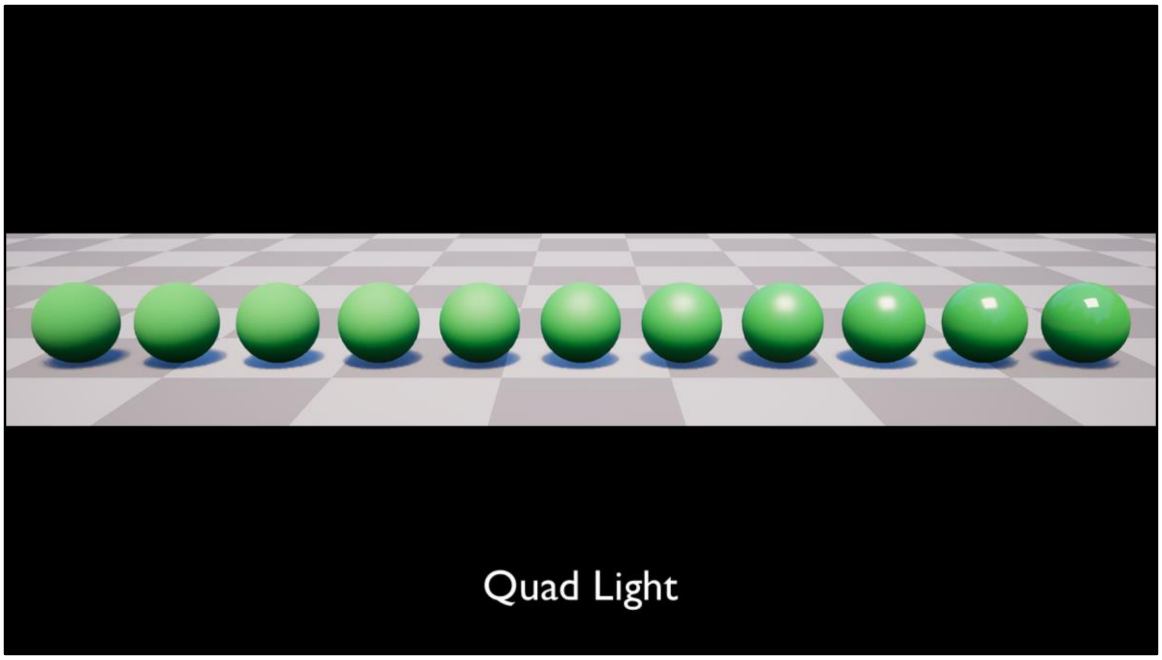


Magnitude and Fresnel LUT

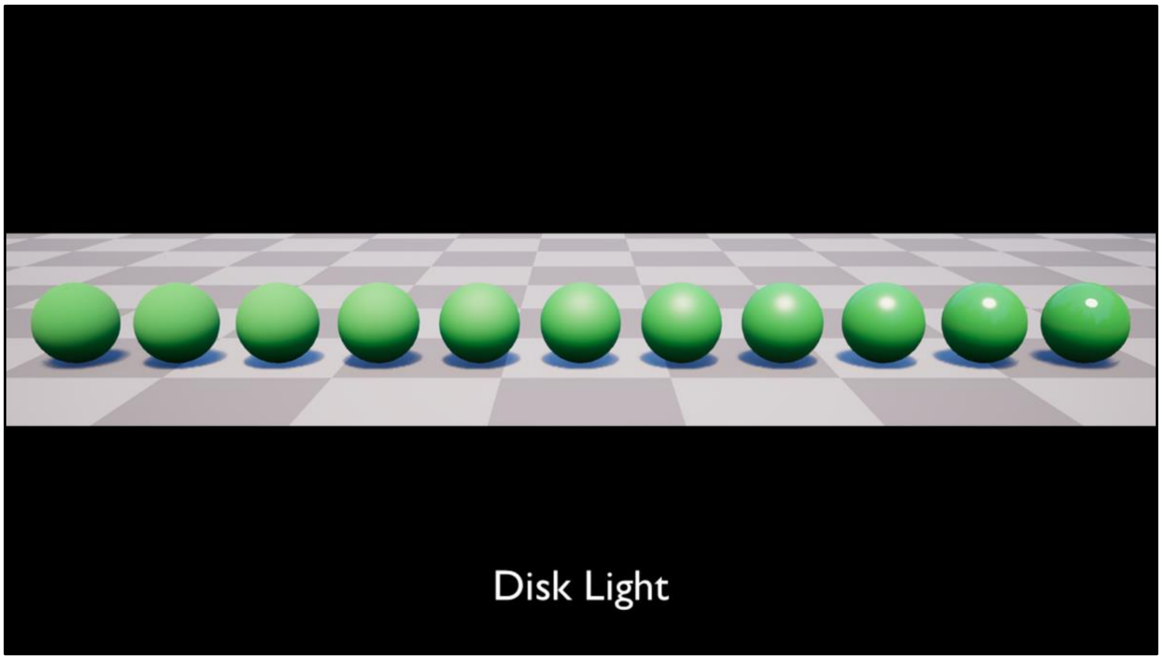
The implementation is relatively straightforward, thanks to all the excellent code and demos provided. Offline, you generate two look up tables for your BRDF, both parameterised on the outgoing angle and the surface roughness. The first contains coordinates of the inverse matrix transform, from the linearly transformed cosine approximation to your BRDF with that angle and roughness, back into a clamped cosine over the hemisphere. The second look up table is very similar to the environment BRDF that we discussed earlier, to scale the BRDF by its magnitude at a given viewing angle.



Let’s see how this looks with some dielectric spheres of increasing smoothness. I’m going to replace the sun light with a quad light:



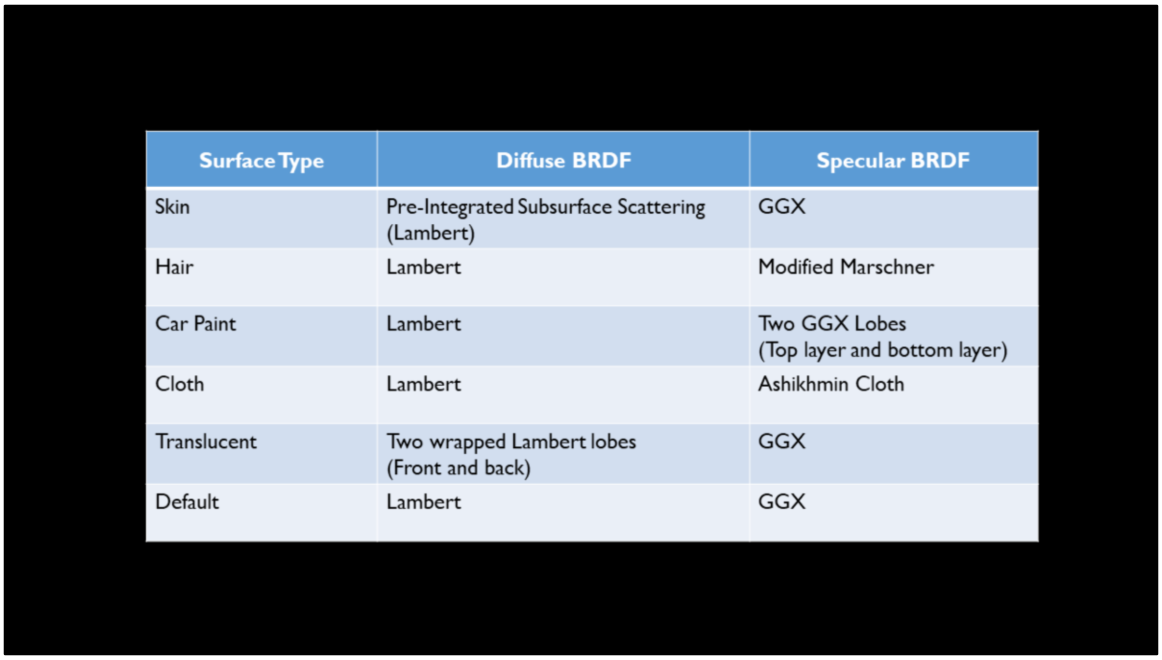
And that looks pretty awesome, particularly the specular highlight on smooth surfaces. We can also do a disk!



That also looks pretty cool!

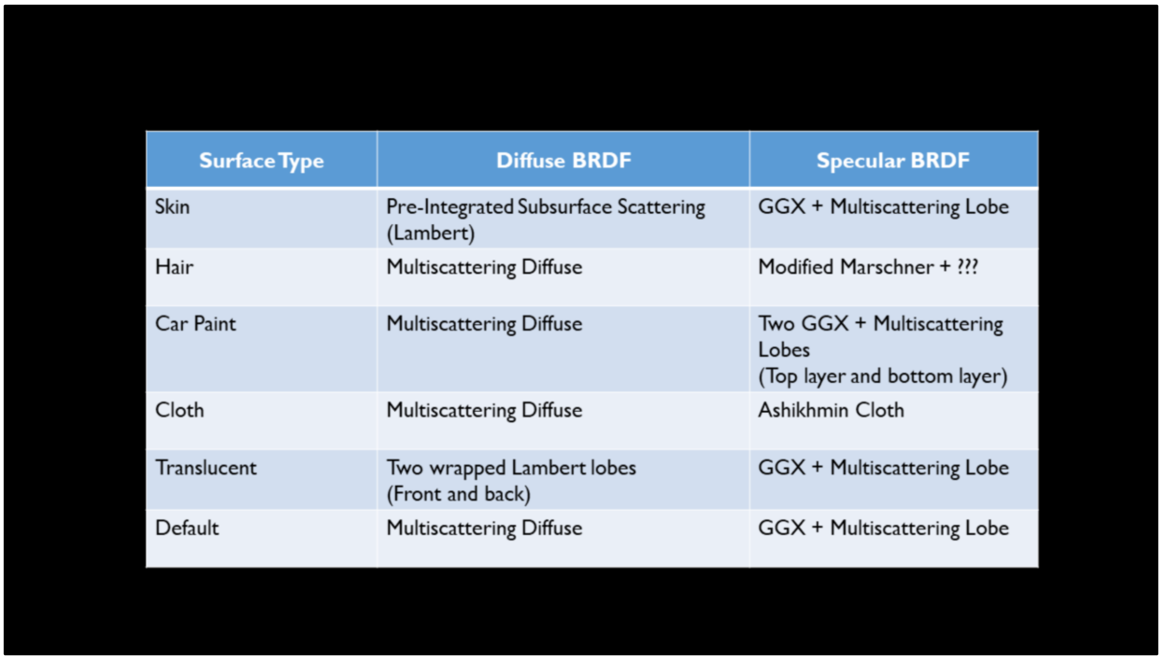
You'll see that the diffuse is changing too. Implementing LTCs for Lambertian diffuse is trivial, since Lambertian diffuse IS the clamped cosine over a sphere. We get a nice softer falloff from the size of the area light, which I know artists will love.

So we have area lights implemented for GGX and Lambert diffuse, so everything's great, right?!



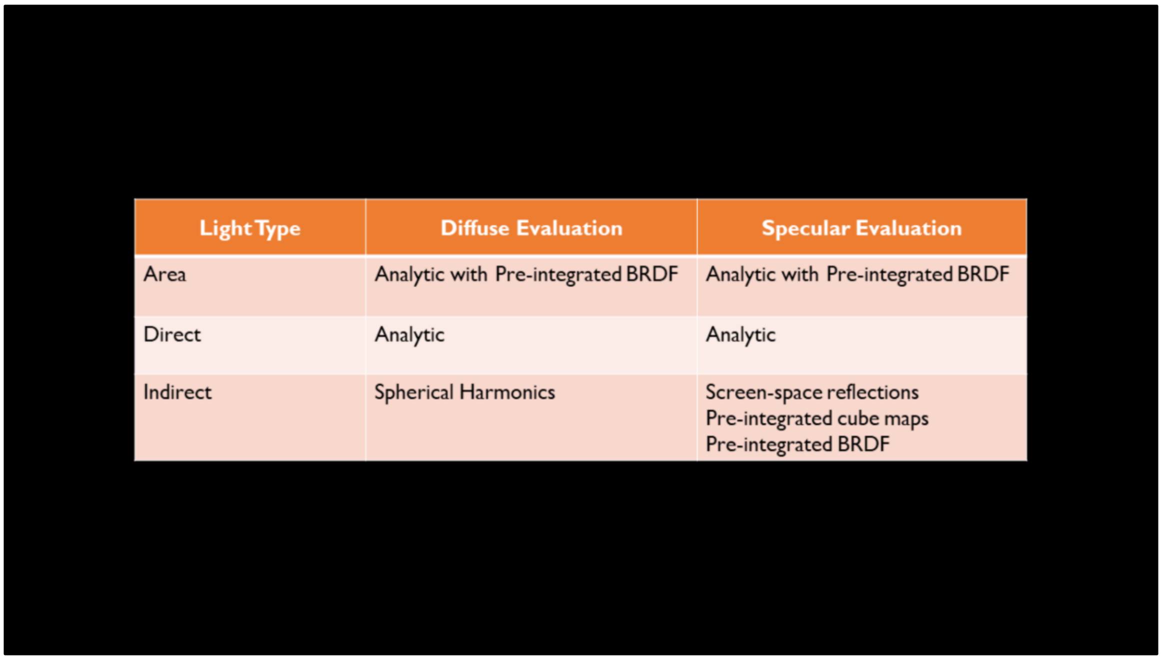
Surface Type	Diffuse BRDF	Specular BRDF
Skin	Pre-Integrated Subsurface Scattering (Lambert)	GGX
Hair	Lambert	Modified Marschner
Car Paint	Lambert	Two GGX Lobes (Top layer and bottom layer)
Cloth	Lambert	Ashikhmin Cloth
Translucent	Two wrapped Lambert lobes (Front and back)	GGX
Default	Lambert	GGX

But… we're back at the combination problem. We don't just have a GGX BRDF. The Lambert BRDF is trivial and already solved, since it is just a clamped cosine over the hemisphere, but what about the wrapped diffuse? The pre-integrated scattering? The hair and the cloth? These are similar problems to those we faced for our new diffuse and specular BRDFs.



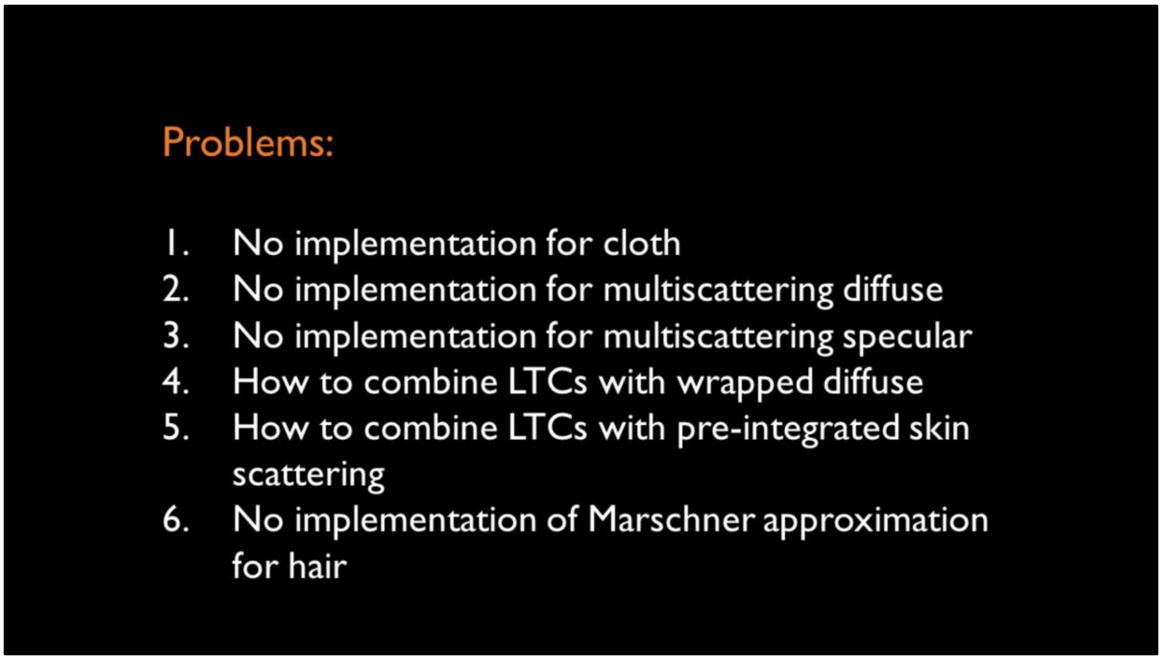
Surface Type	Diffuse BRDF	Specular BRDF
Skin	Pre-Integrated Subsurface Scattering (Lambert)	GGX + Multiscattering Lobe
Hair	Multiscattering Diffuse	Modified Marschner + ???
Car Paint	Multiscattering Diffuse	Two GGX + Multiscattering Lobes (Top layer and bottom layer)
Cloth	Multiscattering Diffuse	Ashikhmin Cloth
Translucent	Two wrapped Lambert lobes (Front and back)	GGX + Multiscattering Lobe
Default	Multiscattering Diffuse	GGX + Multiscattering Lobe

But in fact it’s got worse. Now we also have to make area lights work for multiscattering diffuse, and multiscattering specular too.



Light Type	Diffuse Evaluation	Specular Evaluation
Area	Analytic with Pre-integrated BRDF	Analytic with Pre-integrated BRDF
Direct	Analytic	Analytic
Indirect	Spherical Harmonics	Screen-space reflections Pre-integrated cube maps Pre-integrated BRDF

Essentially, we've just added another light type that we have to evaluate all our BRDFs against.



Problems:

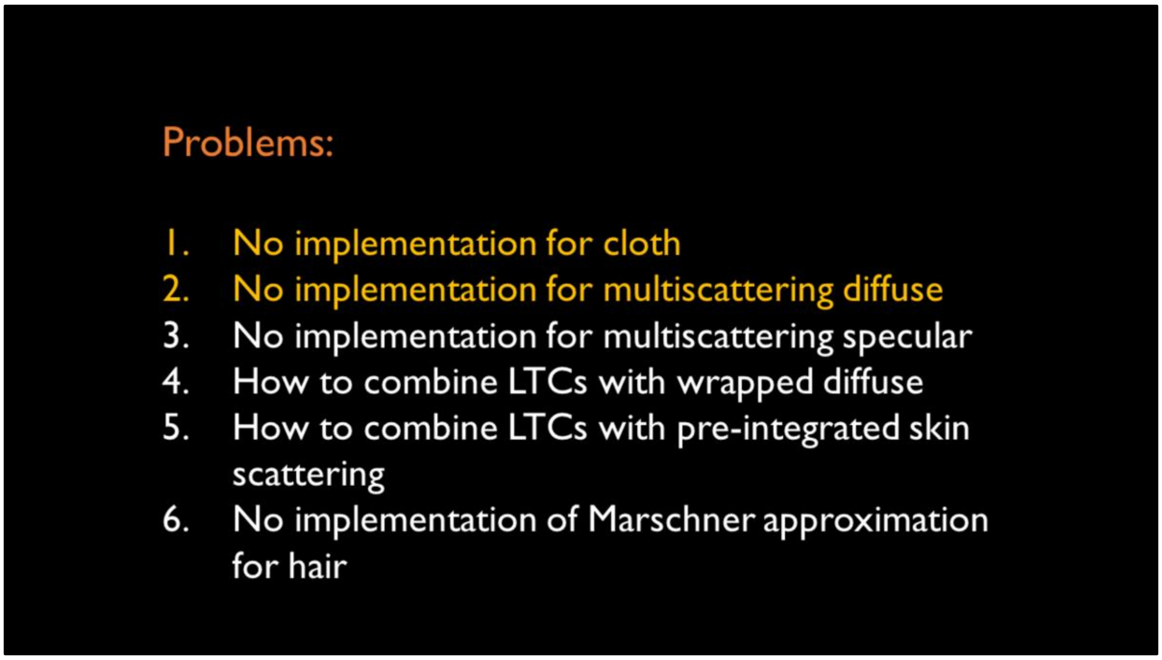
1. No implementation for cloth
2. No implementation for multiscattering diffuse
3. No implementation for multiscattering specular
4. How to combine LTCs with wrapped diffuse
5. How to combine LTCs with pre-integrated skin scattering
6. No implementation of Marschner approximation for hair

So let's summarise where we're at. We have no LTC implementations for three base BRDFs – the cloth, the multiscattering diffuse and the hair. We also don't have multiscattering specular, which we've just implemented for our non-area lighting path. Finally, we have no idea how to combine LTCs with wrapped diffuse and our pre-integrated skin scattering.

Source code available!

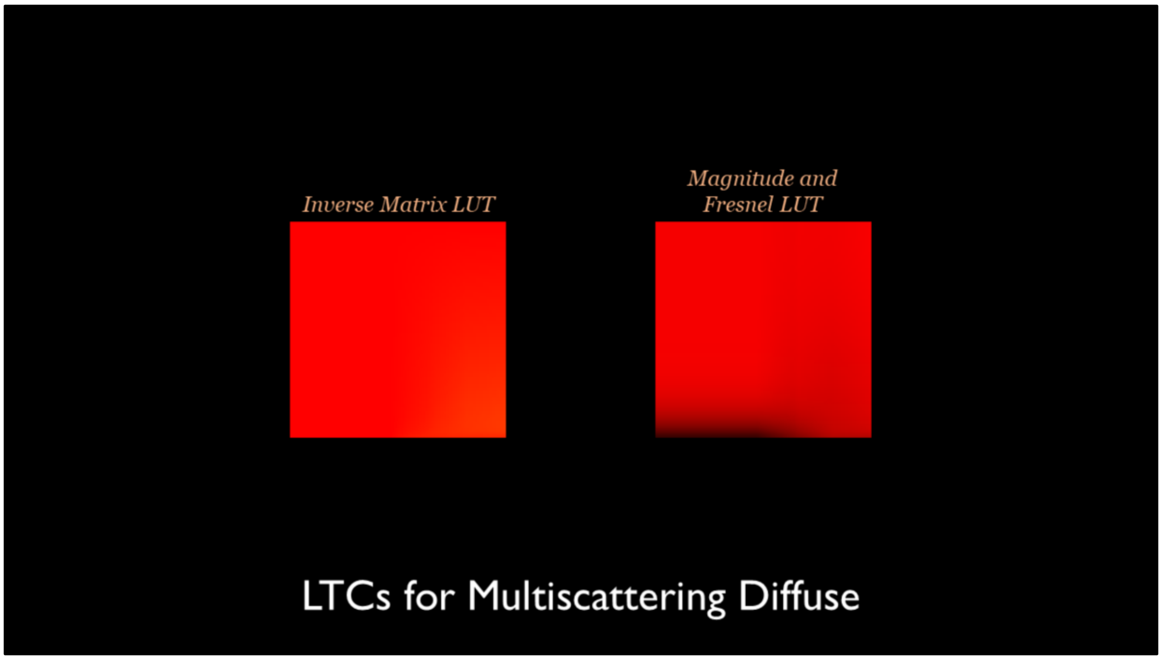
https://github.com/selfshadow/ltc_code

But we have cause for celebration! The source code for LTCs is available, most importantly, the code to fit LTCs against an arbitrary BRDF. So I can just plug my BRDFs into that, run the fitting and create the LUTs, and I should be good to go.


Problems:

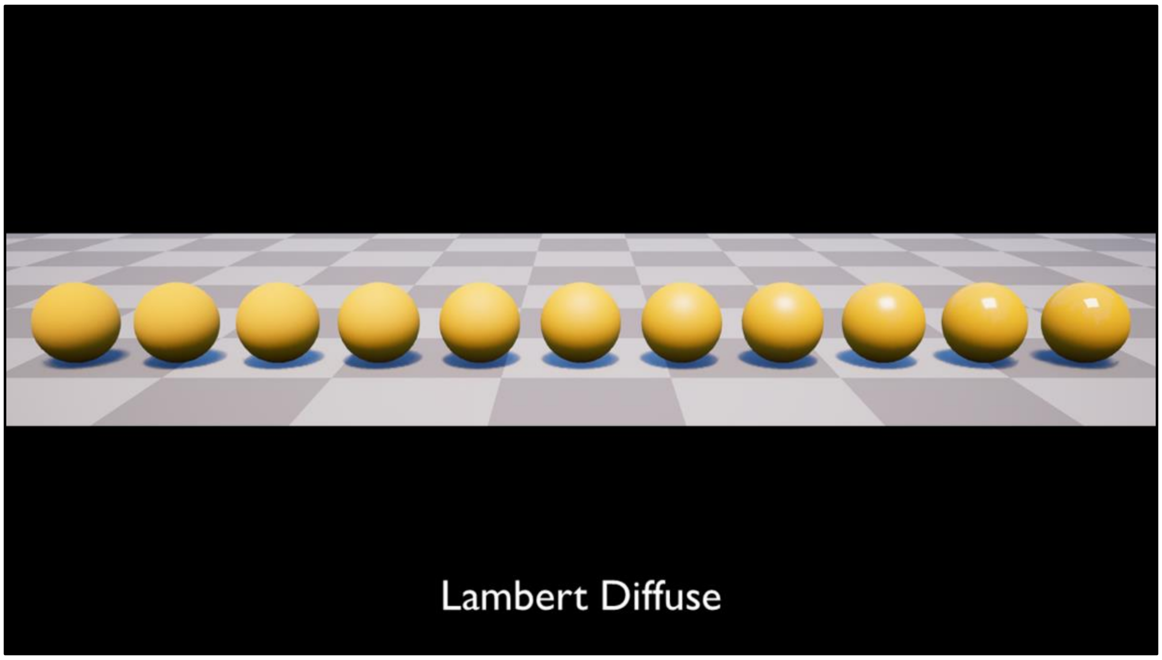
1. No implementation for cloth
2. No implementation for multiscattering diffuse
3. No implementation for multiscattering specular
4. How to combine LTCs with wrapped diffuse
5. How to combine LTCs with pre-integrated skin scattering
6. No implementation of Marschner approximation for hair

So let's first start to trying to get cloth and multiscattering diffuse working.

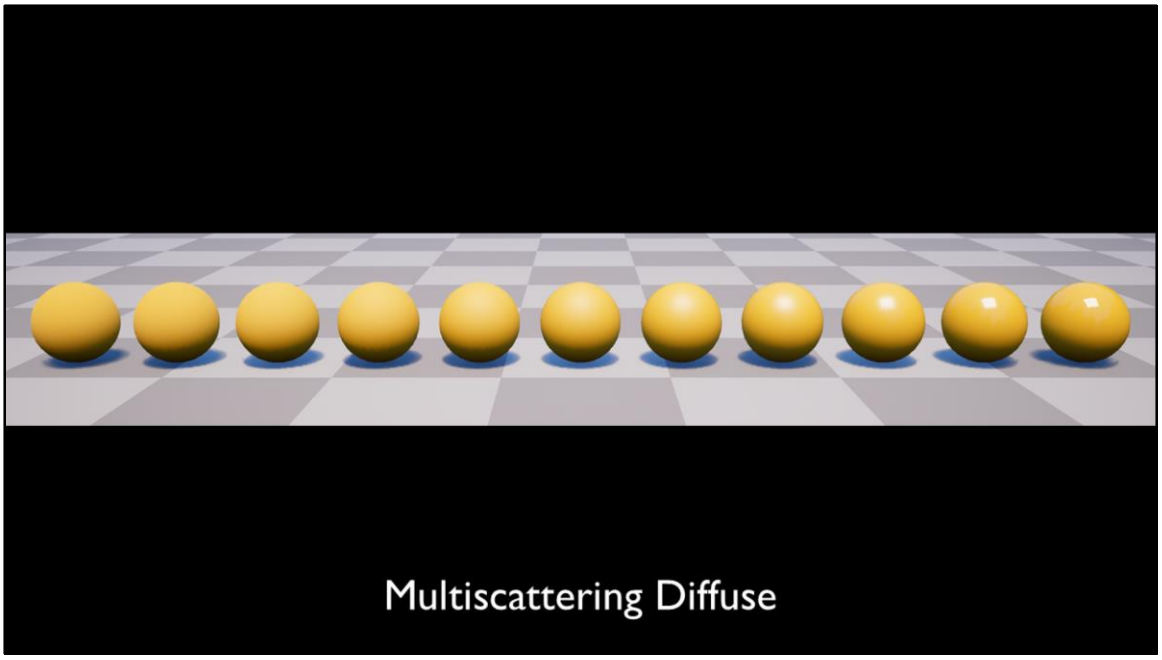


LTCs for Multiscattering Diffuse

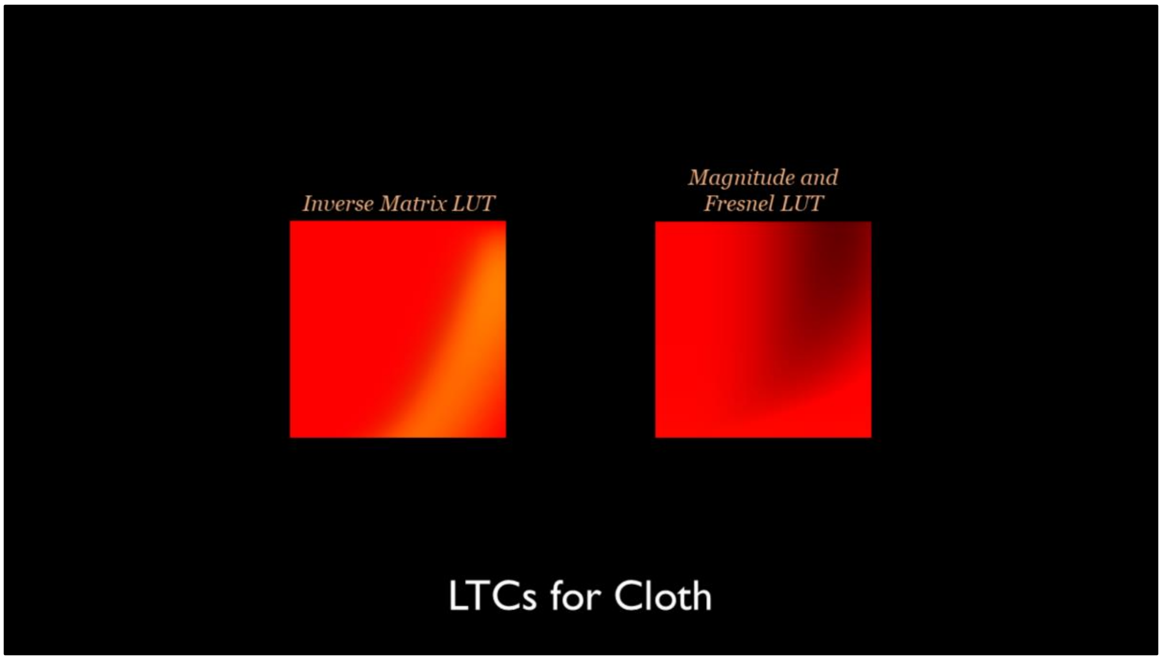
So start by generating our LUTs for multiscattering diffuse. Not much is going on in the inverse matrix compared to GGX, but you'd expect that as roughness and viewing angle have a much smaller effect for this BRDF.



Let’s start with our result for Lambertian diffuse…

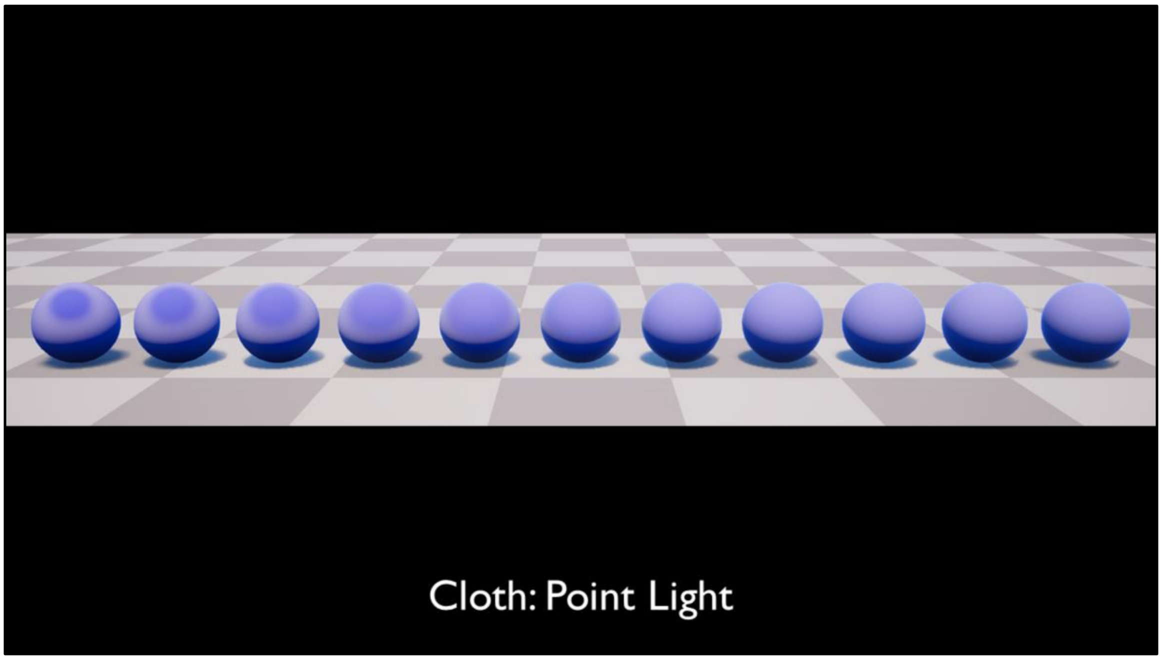


We can see the desired effect happening. The lighting is getting slightly flatter and more retroreflective for rough surfaces, but with a slightly stronger falloff for smooth surfaces. Smooth surfaces are also getting darker at a grazing viewing angle. This is great because the diffuse lighting on the material is responding exactly as for point lights, so we’ve achieved what we were looking for.

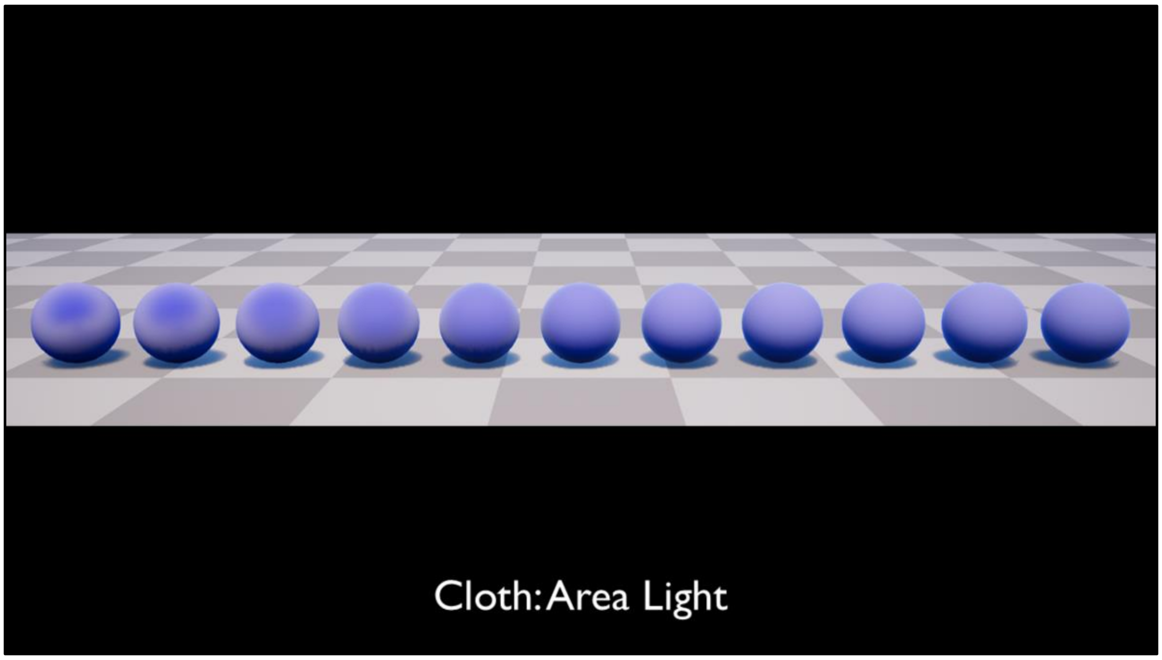


LTCs for Cloth

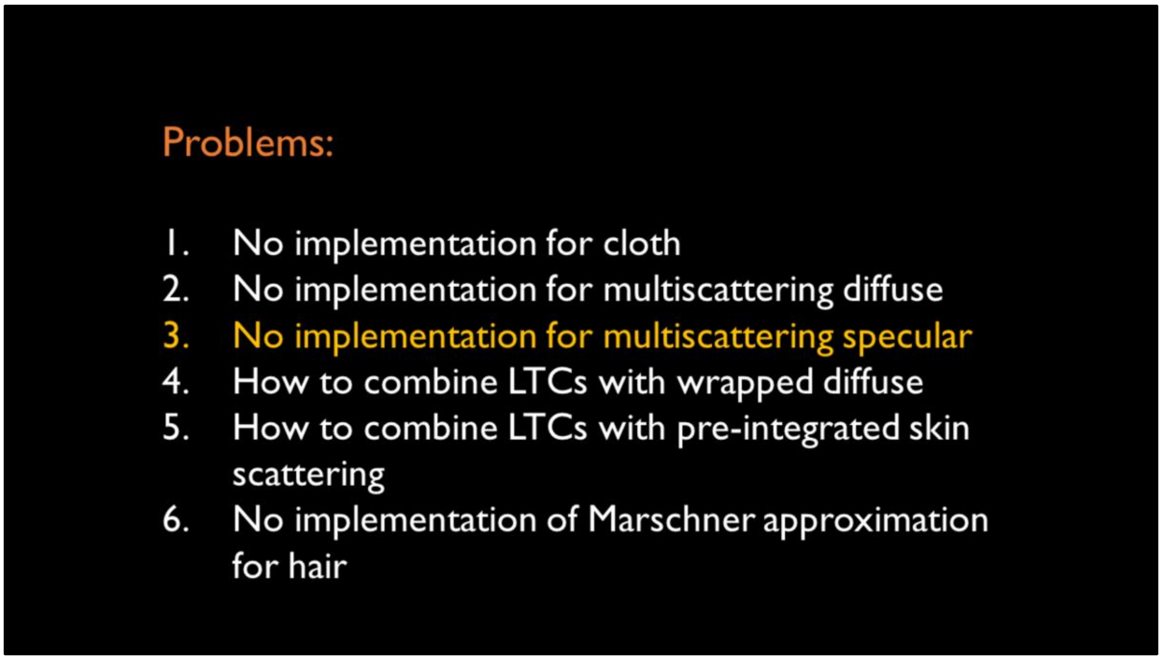
We can also try this for cloth! Let’s see how this works…



This time we'll compare against our point light cloth implementation. It's the rougher surfaces that have the biggest retroreflection and sharpest effect in our cloth model, which you can see here.



This seems pretty good! It's different for sure, it seems a little more diffuse. But that's what we'd expect since we have a light with a size now. This still means that we'll get the cloth effect on various materials, which is what we're looking for.

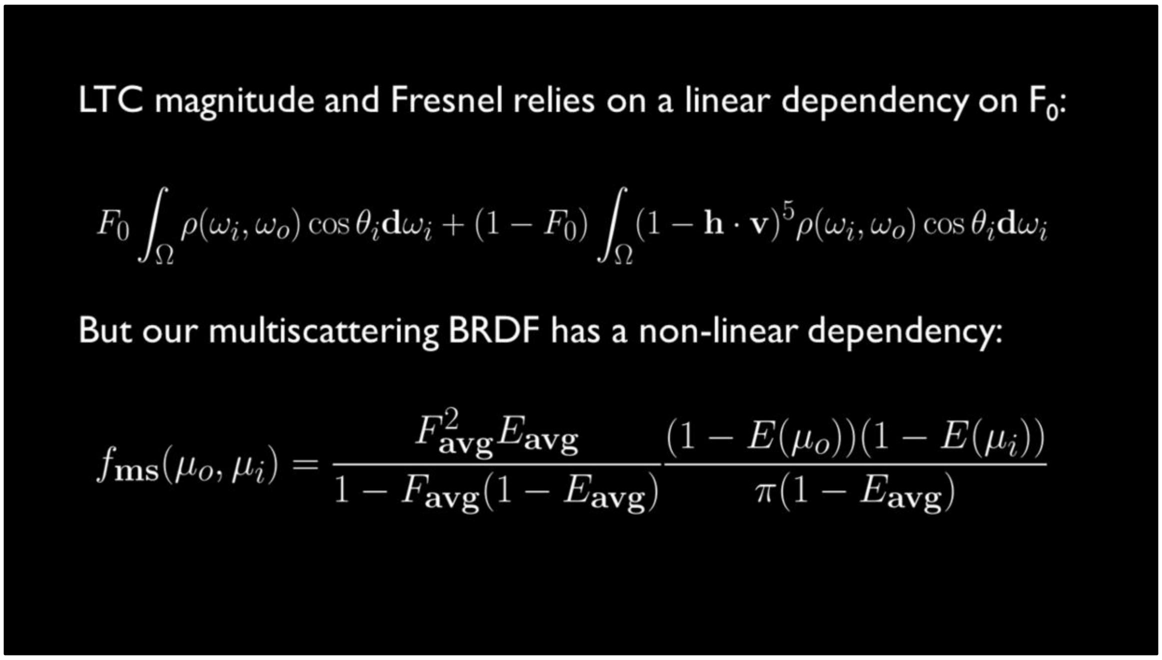


Problems:

1. No implementation for cloth
2. No implementation for multiscattering diffuse
3. No implementation for multiscattering specular
4. How to combine LTCs with wrapped diffuse
5. How to combine LTCs with pre-integrated skin scattering
6. No implementation of Marschner approximation for hair

The next problem is multiscattering specular. We've just implemented this new BRDF that we want, but we need to get it working with LTCs…

Now, at first we might think that this is exactly the same as cloth and multiscattering diffuse, just use the fitting code with a different BRDF, but actually this has the same problem as a topic we've already discussed – indirect specular and the environment BRDF.



LTC magnitude and Fresnel relies on a linear dependency on F_0:

$$F_0 \int_\Omega \rho(\omega_i, \omega_o) \cos\theta_i \mathbf{d}\omega_i + (1 - F_0) \int_\Omega (1 - \mathbf{h} \cdot \mathbf{v})^5 \rho(\omega_i, \omega_o) \cos\theta_i \mathbf{d}\omega_i$$

But our multiscattering BRDF has a non-linear dependency:

$$f_{\mathbf{ms}}(\mu_o, \mu_i) = \frac{F_{\mathbf{avg}}^2 E_{\mathbf{avg}}}{1 - F_{\mathbf{avg}}(1 - E_{\mathbf{avg}})} \frac{(1 - E(\mu_o))(1 - E(\mu_i))}{\pi(1 - E_{\mathbf{avg}})}$$

Previously, we were able to break the BRDF down into a linear dependency on F0, and store two components in a LUT. But once again, the multiscattering BRDF is dependent on F0, in a non-linear way, so we can't use this trick.

JCGT
@JCGT_announce
Following

Fdez-Agüera, A Multiple-Scattering Microfacet Model for Real-Time Image Based Lighting. jcgt.org/published/0008...

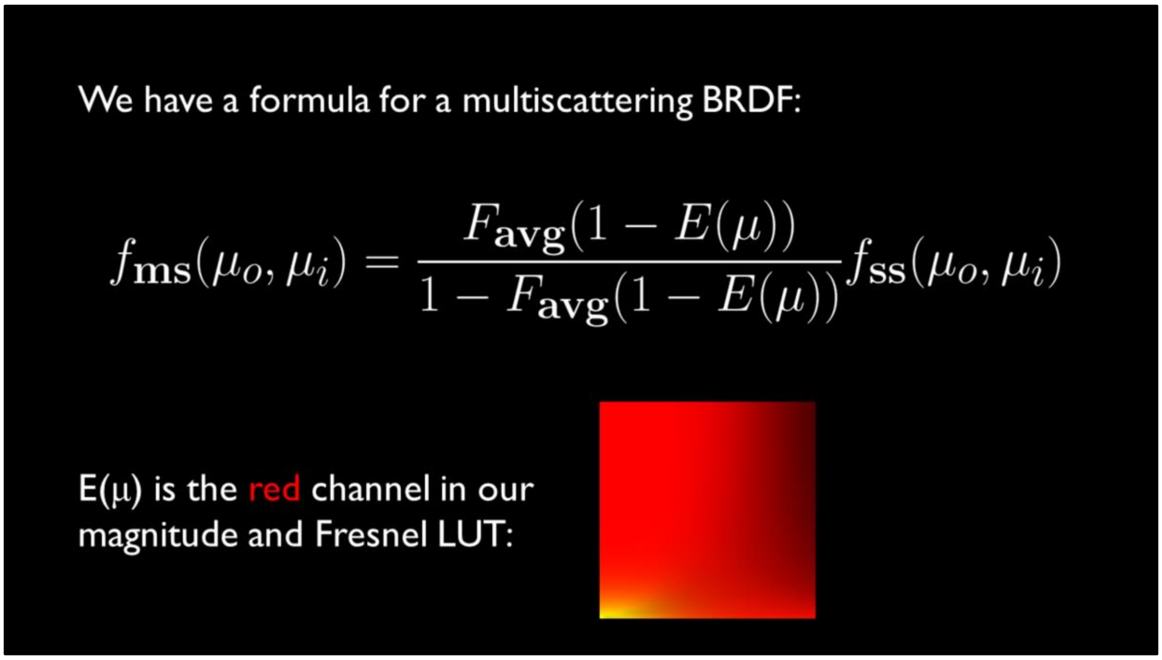
11:47 AM - 22 Jan 2019

98 Retweets 225 Likes

1 98 225

[FdezAgüera19]

But we have a solution! We can use the same solution as for the environment BRDF!

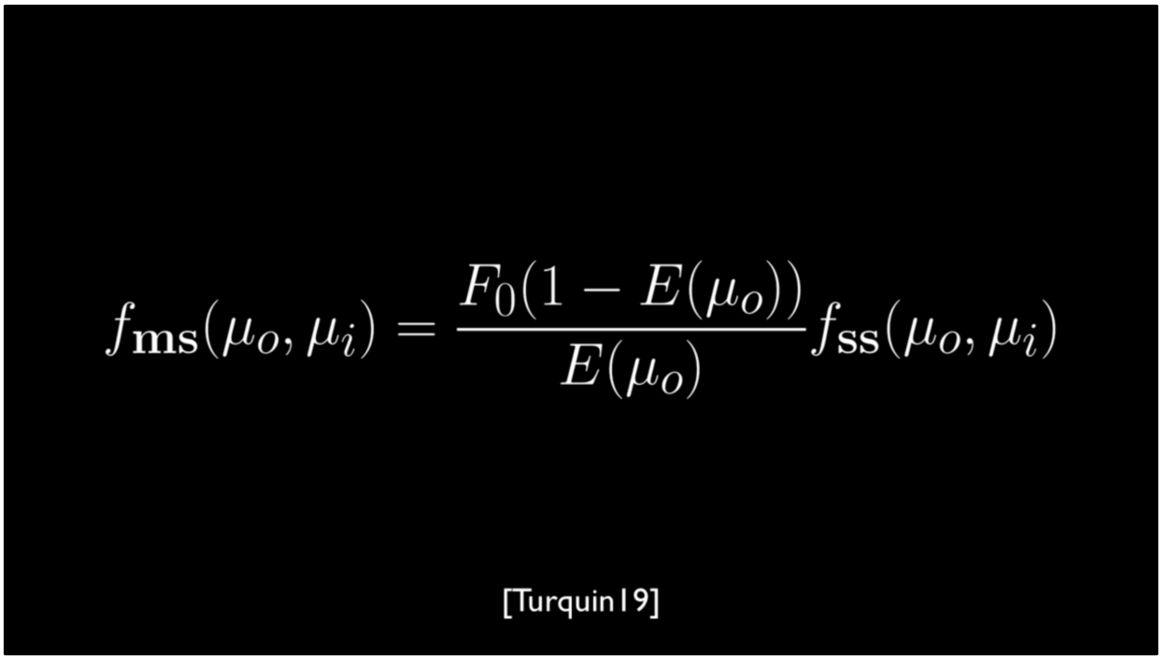


We have a formula for a multiscattering BRDF:

$$f_{\mathbf{ms}}(\mu_o, \mu_i) = \frac{F_{\mathbf{avg}}(1 - E(\mu))}{1 - F_{\mathbf{avg}}(1 - E(\mu))} f_{\mathbf{ss}}(\mu_o, \mu_i)$$

E(μ) is the red channel in our magnitude and Fresnel LUT:

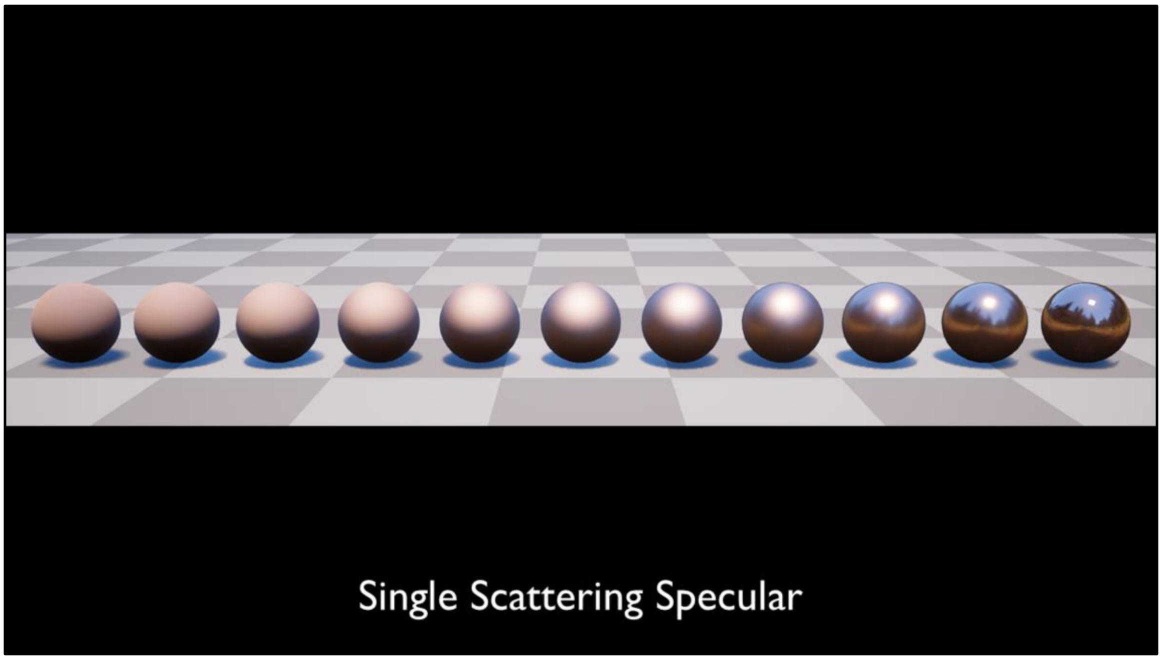
Now, this gives us a formula for a multiscattering BRDF that can just scale the single scattering BRDF, which is awesome, because it’s fast and simple and can be applied to our LTCs. We can get the energy term from our magnitude and Fresnel LUT, just like we could from our environment BRDF.



$$f_{\mathbf{ms}}(\mu_o, \mu_i) = \frac{F_0(1 - E(\mu_o))}{E(\mu_o)} f_{\mathbf{ss}}(\mu_o, \mu_i)$$

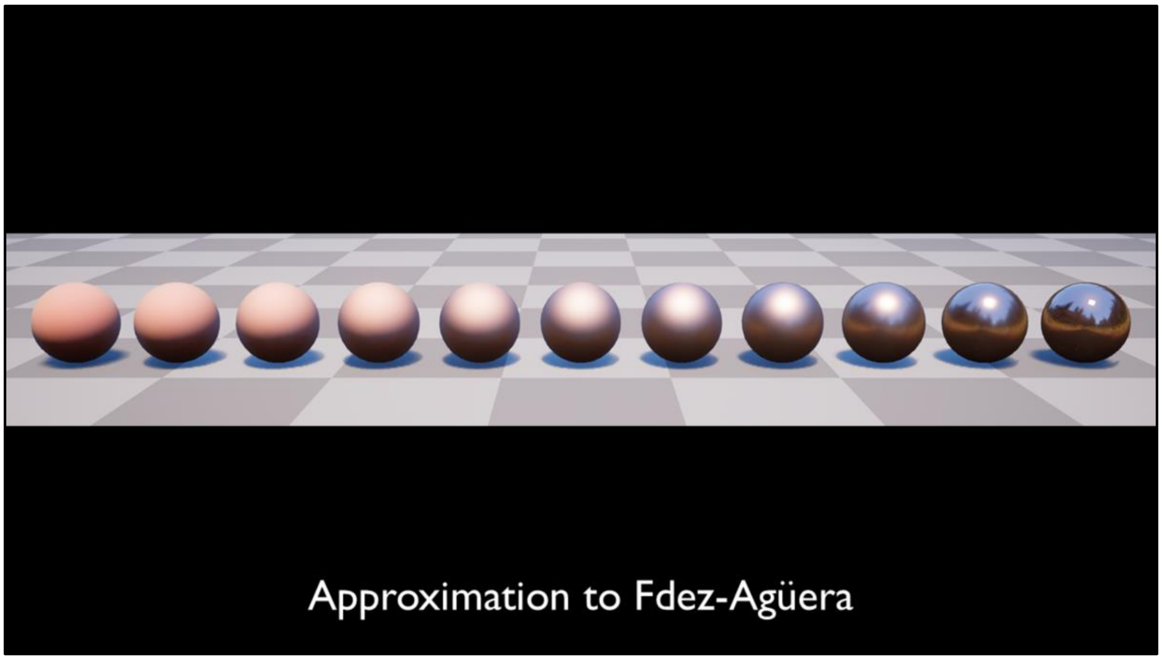
[Turquin19]

We could also use the formula from Emmanuel Turquin! Let’s try them out!

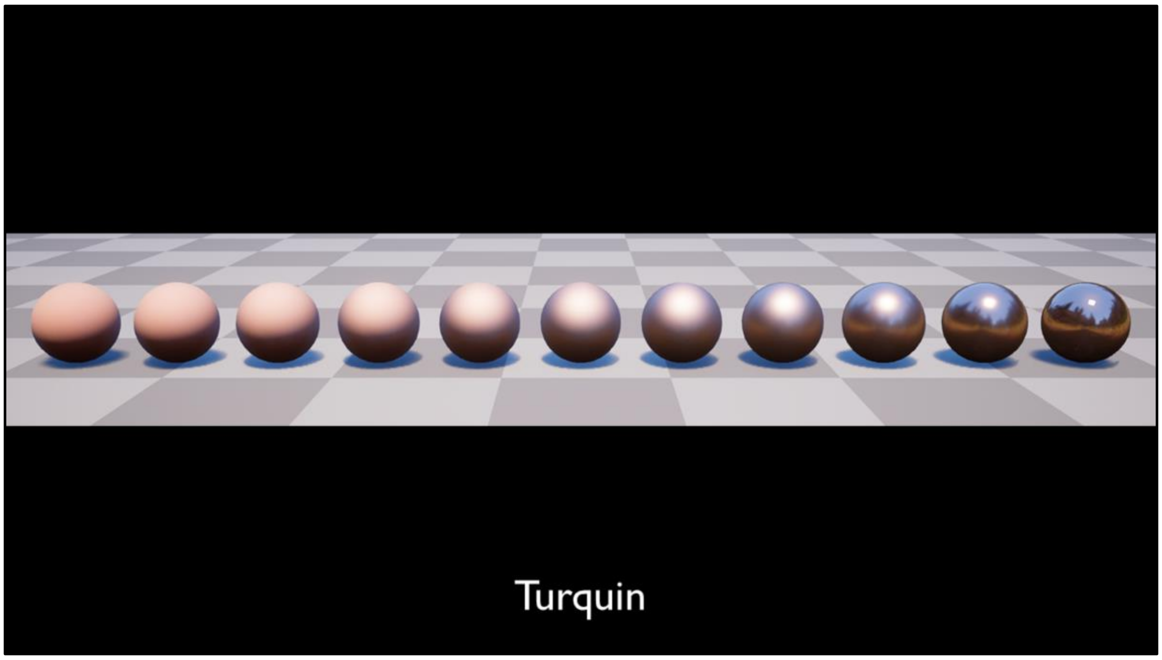


Single Scattering Specular

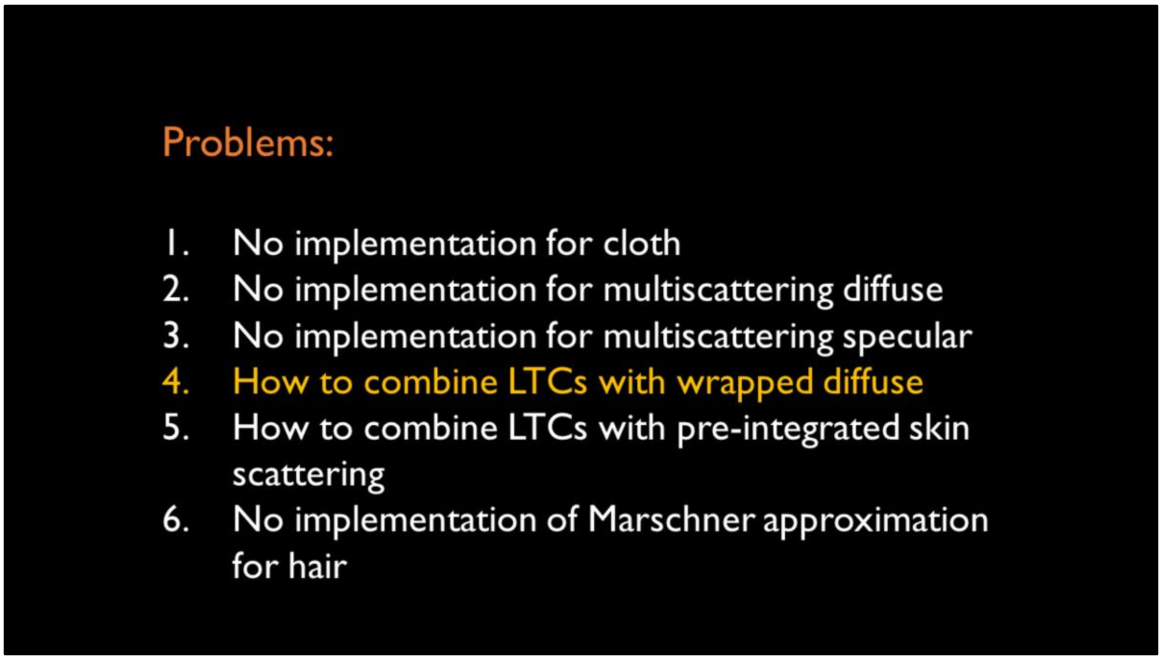
Let’s start by looking at single scattering specular.



Now let’s use my approximation to Fdez-Agüera to get a multiscattering BRDF.



And finally, let's look at Emmanuel Turquin's solution. Once again, this is slightly brighter but with slightly less colour than the Fdez-Agüera approach.

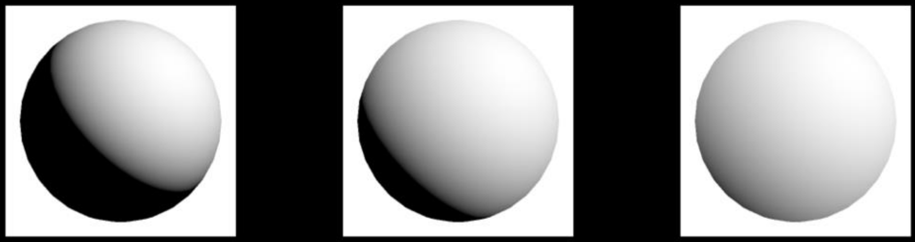


Problems:

1. No implementation for cloth
2. No implementation for multiscattering diffuse
3. No implementation for multiscattering specular
4. How to combine LTCs with wrapped diffuse
5. How to combine LTCs with pre-integrated skin scattering
6. No implementation of Marschner approximation for hair

The next problem to solve is wrapped lighting. What’s interesting is that we need to solve this problem for multiscattering diffuse as well as for area lights, so maybe there’s a solution that we can use for both.

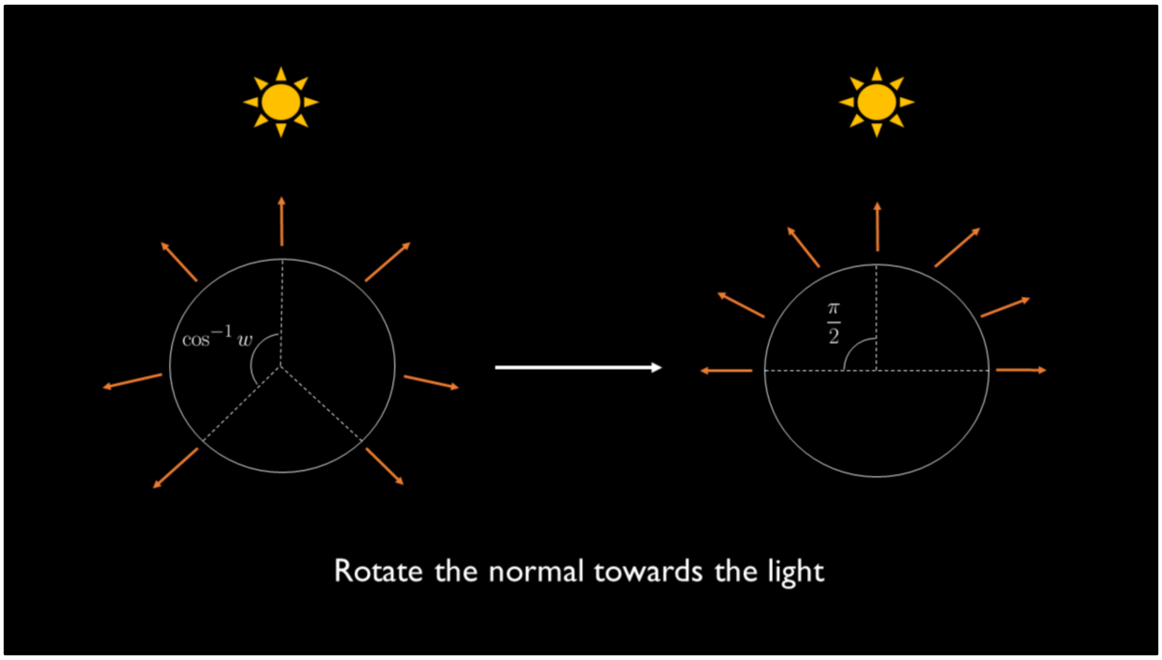
Wrapped Lambertian Diffuse: $\frac{1}{\pi}\frac{\cos\theta + w}{1+w}$



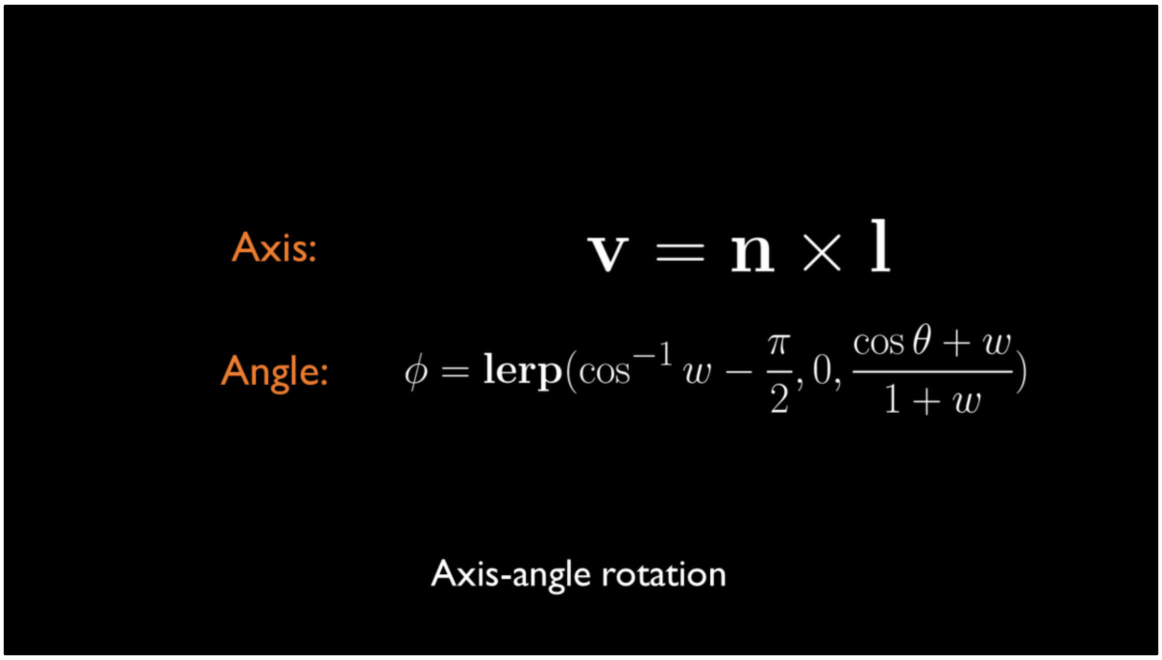
Let’s start with a quick recap of wrapped diffuse. We take Lambertian diffuse, which is the cosine of the incoming light angle, and we just wrap it around the sphere. The wrap factor varies between 0, which has no effect, to 1 which wraps the lighting all the way around.

This is great and all, but how is this going to work for other BRDFs? For example, our new multiscattering diffuse model has a term dependent on the incoming angle and another term on the outgoing angle. Do we need to wrap both? What about for area lights?

I had an idea… rather than manipulating the cosine of the incoming light angle, like we are here, why don’t we just change the normal directly?



If we rotate the normal towards the light, that will surely have the same effect? Then we can use that normal in all our normal lighting equations, with whatever BRDF we want.



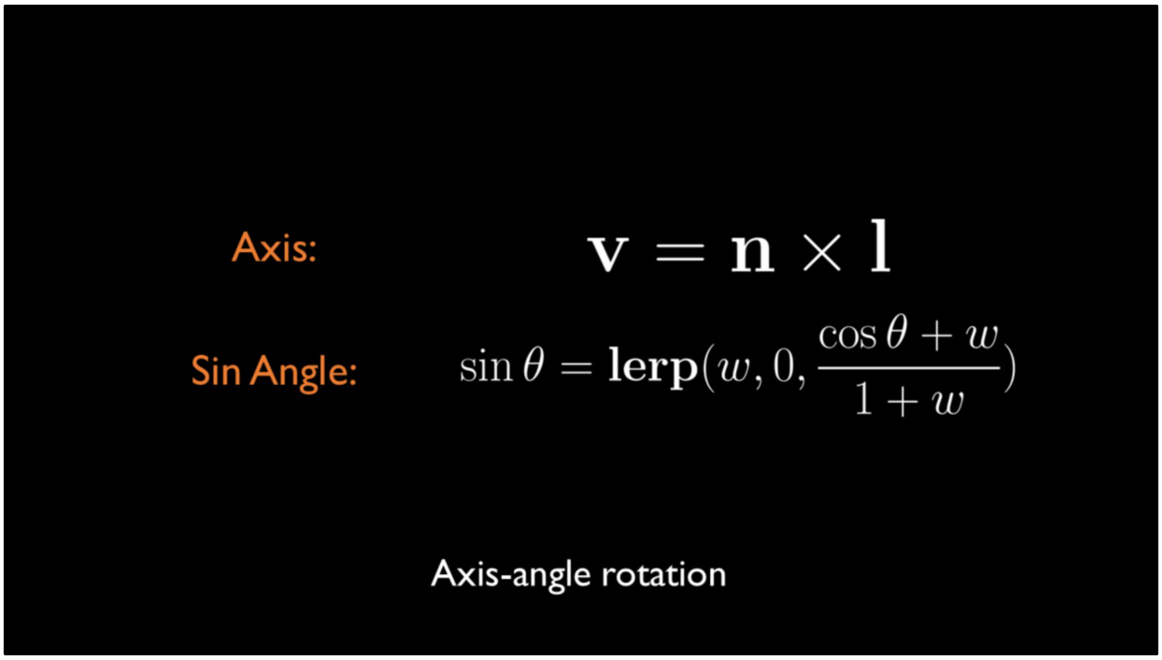
Axis: $$\mathbf{v} = \mathbf{n} \times \mathbf{l}$$

Angle: $$\phi = \mathbf{lerp}(\cos^{-1} w - \frac{\pi}{2}, 0, \frac{\cos\theta + w}{1 + w})$$

Axis-angle rotation

This boils down to just a simple axis-angle rotation. However, the rotation angle depends on how far we are around the sphere. When the normal is facing the light, when cos theta is 1, we don't want to rotate at all. On the other hand, when we're at the furthest point around the sphere we want to wrap, when cos theta is –w, then we want to rotate to the point where theta is pi/2.

However, in shader code we in fact want to avoid the costly inverse cosine, so instead we can blend between the sine of the angles…

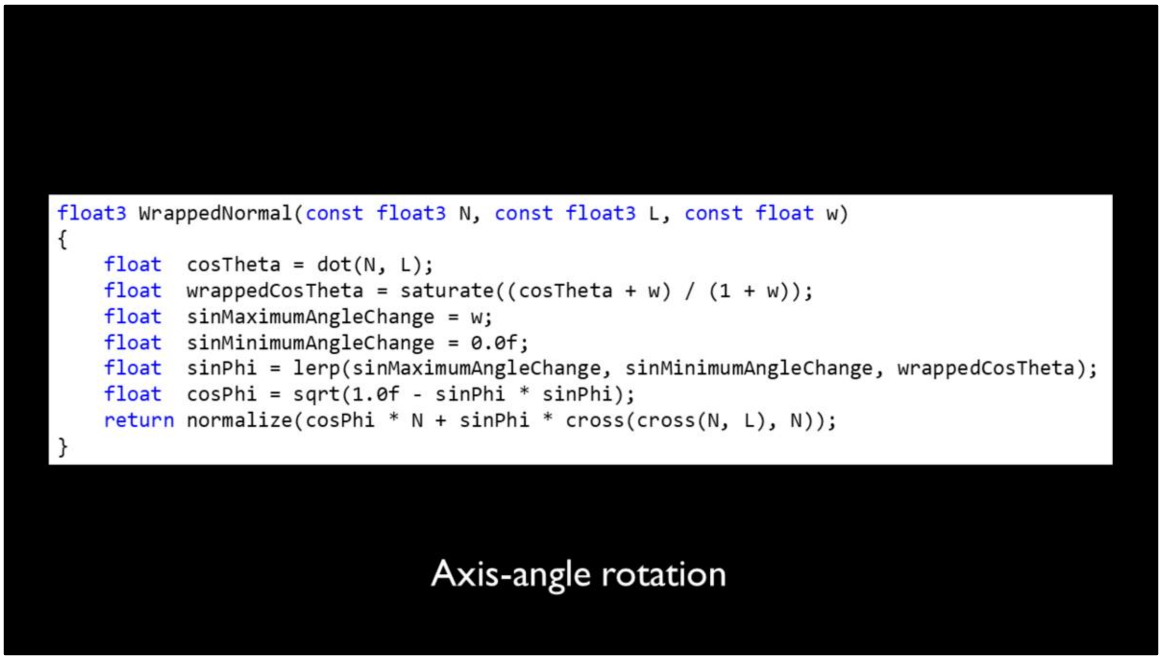


Axis: $$\mathbf{v} = \mathbf{n} \times \mathbf{l}$$

Sin Angle: $$\sin\theta = \mathbf{lerp}(w, 0, \frac{\cos\theta + w}{1 + w})$$

Axis-angle rotation

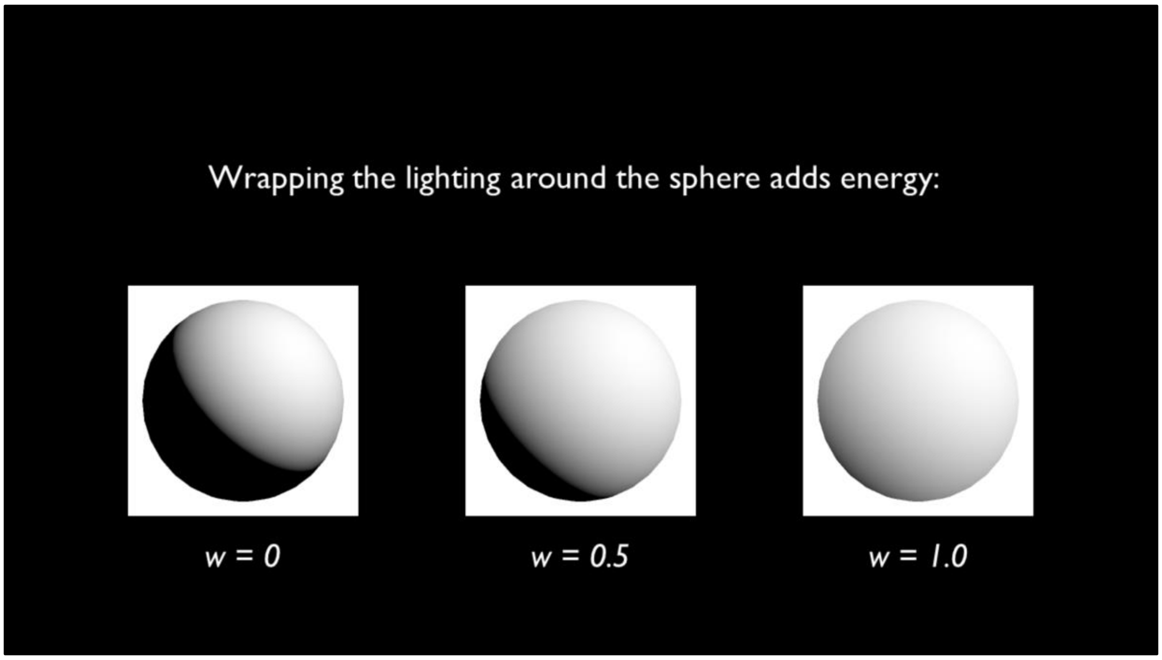
...which is much, much faster!



```
float3 WrappedNormal(const float3 N, const float3 L, const float w)
{
    float  cosTheta = dot(N, L);
    float  wrappedCosTheta = saturate((cosTheta + w) / (1 + w));
    float  sinMaximumAngleChange = w;
    float  sinMinimumAngleChange = 0.0f;
    float  sinPhi = lerp(sinMaximumAngleChange, sinMinimumAngleChange, wrappedCosTheta);
    float  cosPhi = sqrt(1.0f - sinPhi * sinPhi);
    return normalize(cosPhi * N + sinPhi * cross(cross(N, L), N));
}
```

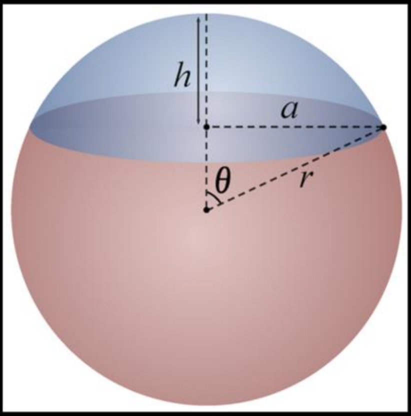
Axis-angle rotation

This ends up being a relatively simple shader function we can use to adjust our normal based upon the light direction and a wrapping factor.



Wrapping the lighting around the sphere adds energy:

But what about normalisation? If we're wrapping the lighting further around the sphere then we're effectively adding energy.

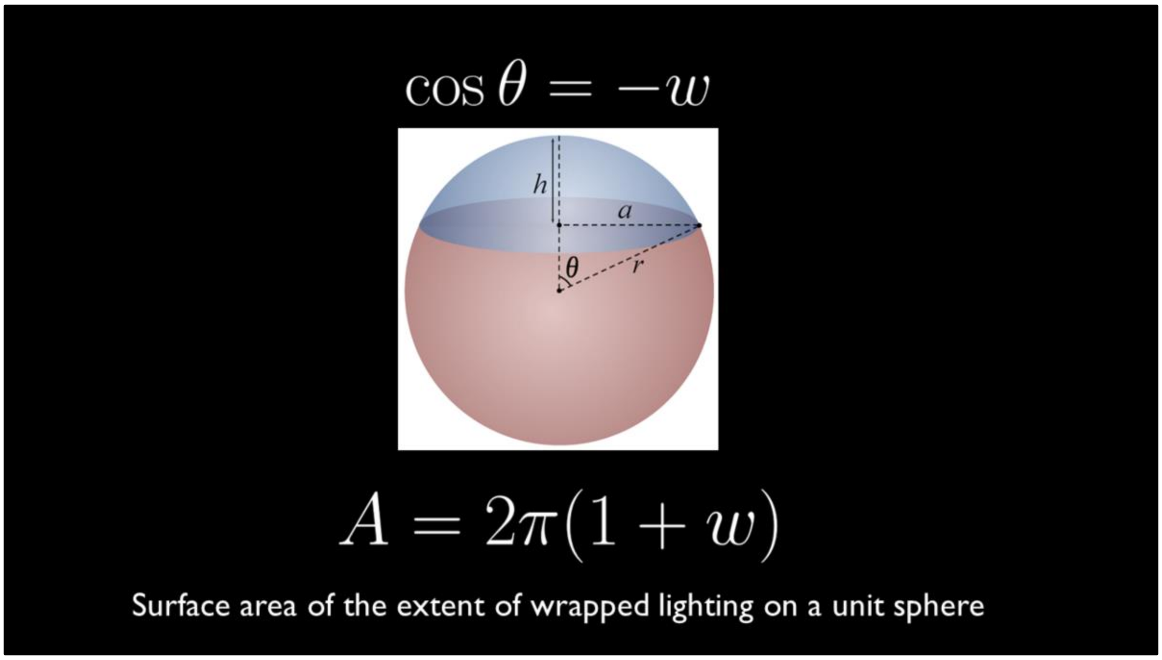


$$A = 2\pi r(1 - \cos\theta)$$

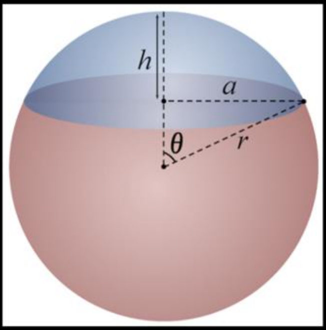
Surface area of spherical cap

Well, it turns out we can normalise by looking at the surface area of a spherical cap on the sphere.

Image attribution:
By Jhmadden - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=44784311



$$\cos\theta = -w$$

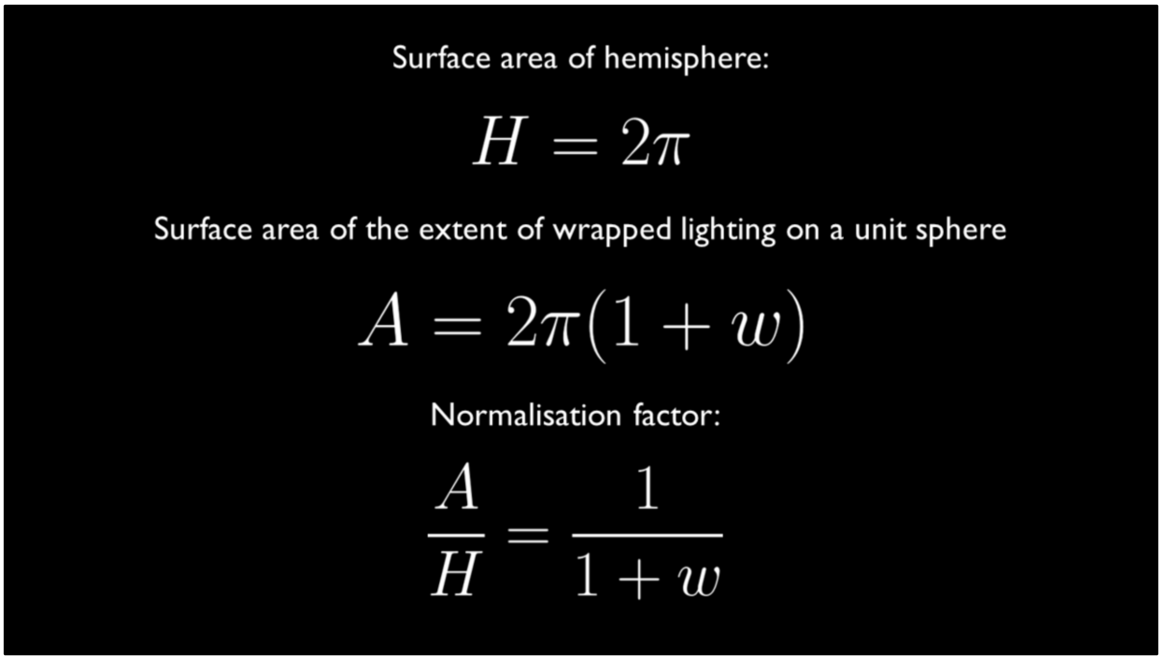


$$A = 2\pi(1 + w)$$

Surface area of the extent of wrapped lighting on a unit sphere

If we recall the definition of cos theta at the furthest extent around the sphere, and substitute that into the surface area of the spherical cap, then we get the following formula.

Image attribution:
By Jhmadden - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=44784311


Surface area of hemisphere:

$$H = 2\pi$$

Surface area of the extent of wrapped lighting on a unit sphere

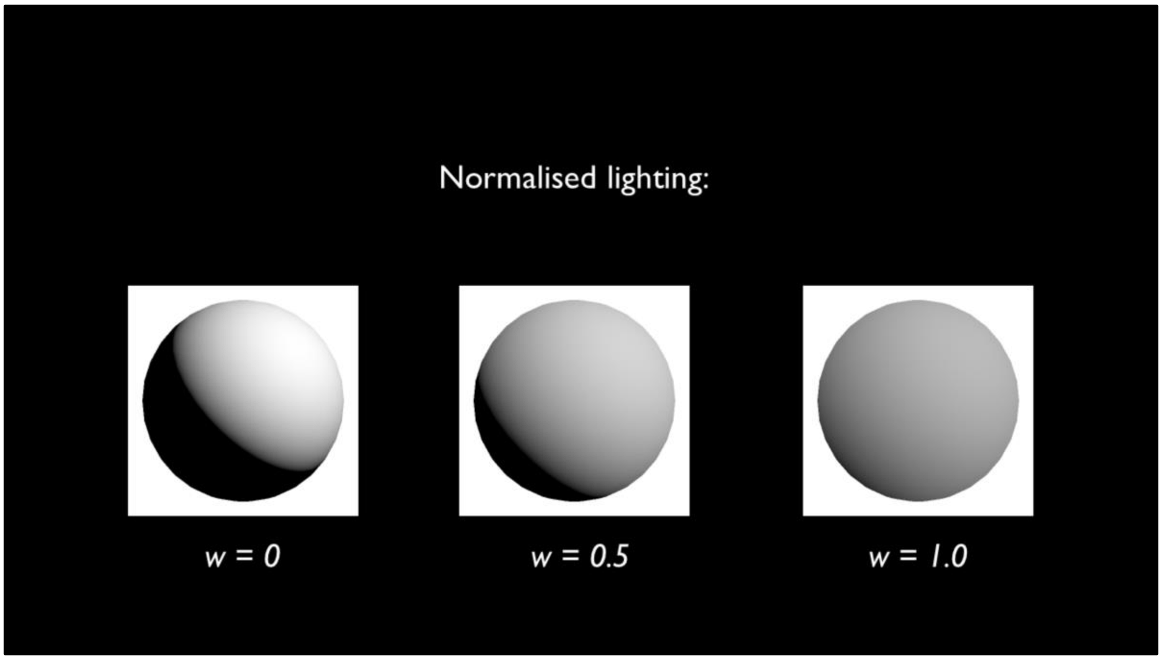
$$A = 2\pi(1 + w)$$

Normalisation factor:

$$\frac{A}{H} = \frac{1}{1 + w}$$

Then, if we compare that to the surface area of the hemisphere, then we get the following simple normalisation factor.

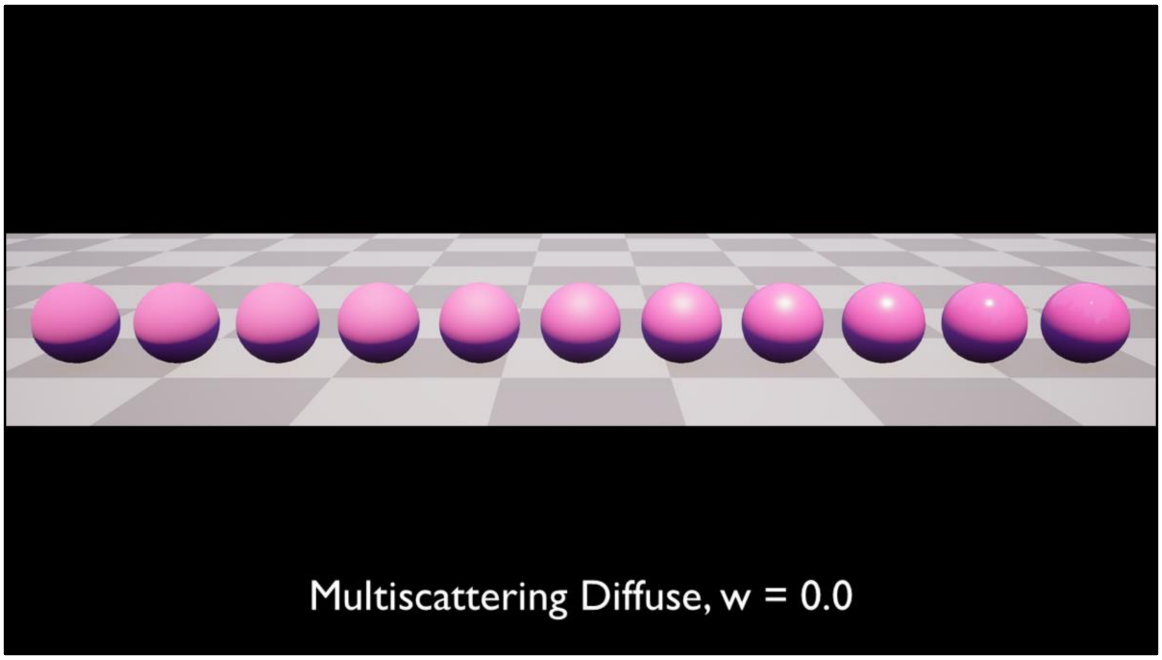
Funnily enough, I actually wrote a blog post on normalising wrapped Lambert diffuse lighting in the past and this is the normalisation factor I came up with then. It was kind of to my surprise that I suddenly found another justification for it, and one that now works in different circumstances.



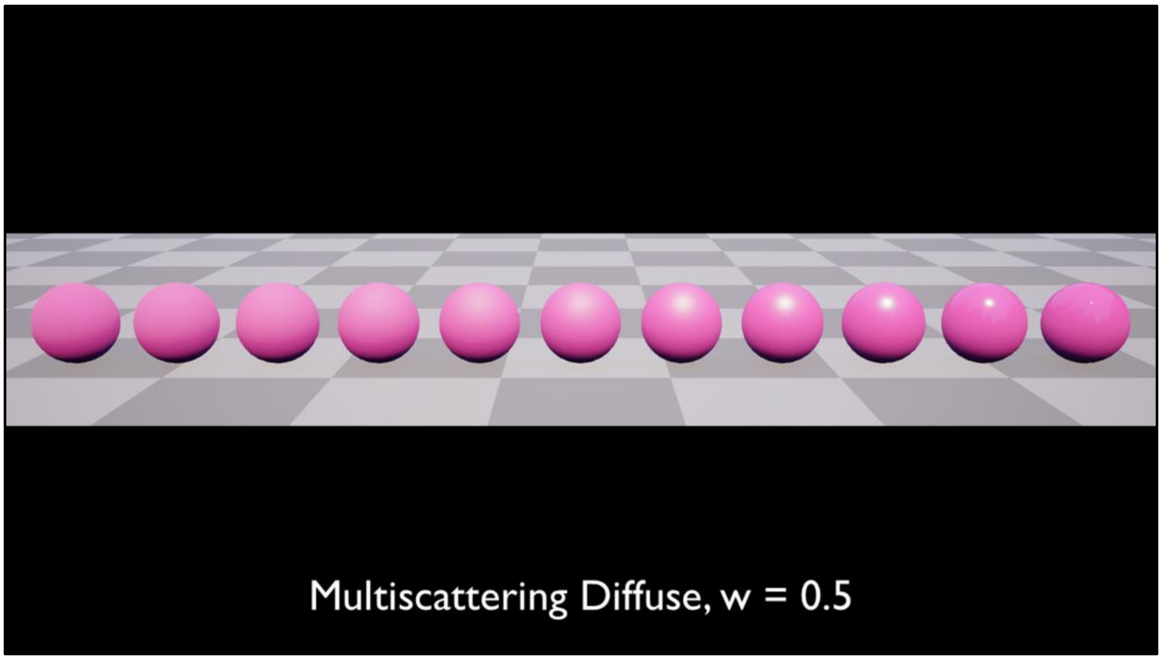
Here are the results, which look much better.

Now, I'll admit that this is a little inaccurate. As we're blending the sine of the angles instead of the angles for performance, the normalisation factor won't quite be right for the BRDF. But this is games so for performance's sake, we think this should be "good enough".

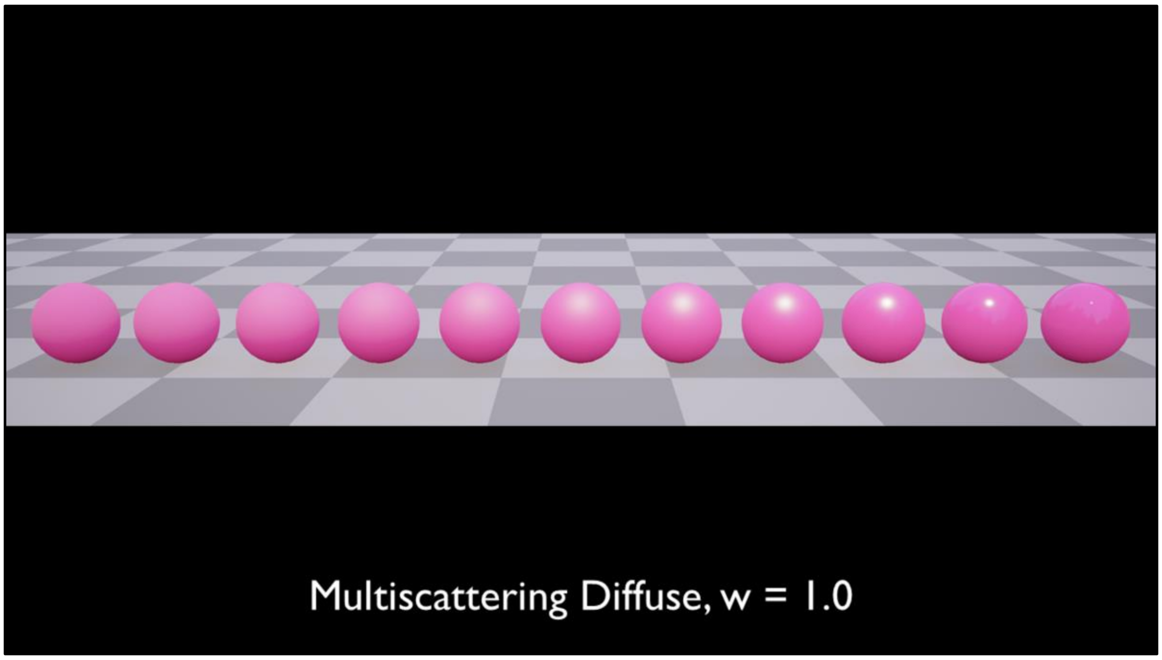
Let's give this a try!



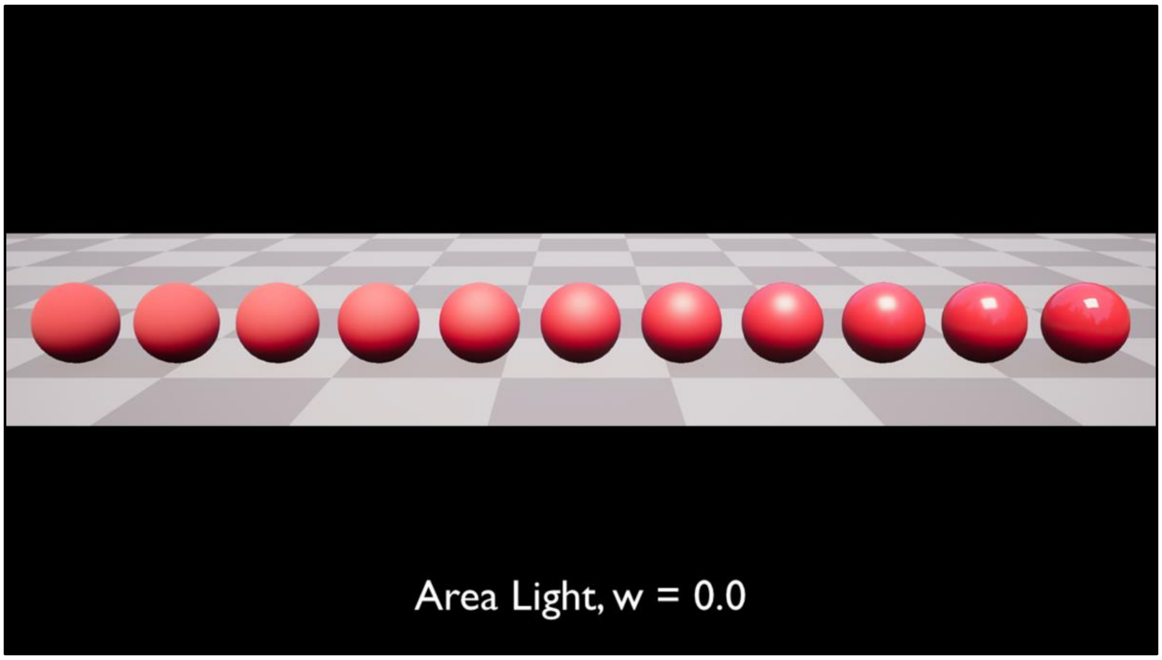
Let’s start by looking at point lights, and our base multiscattering diffuse model.



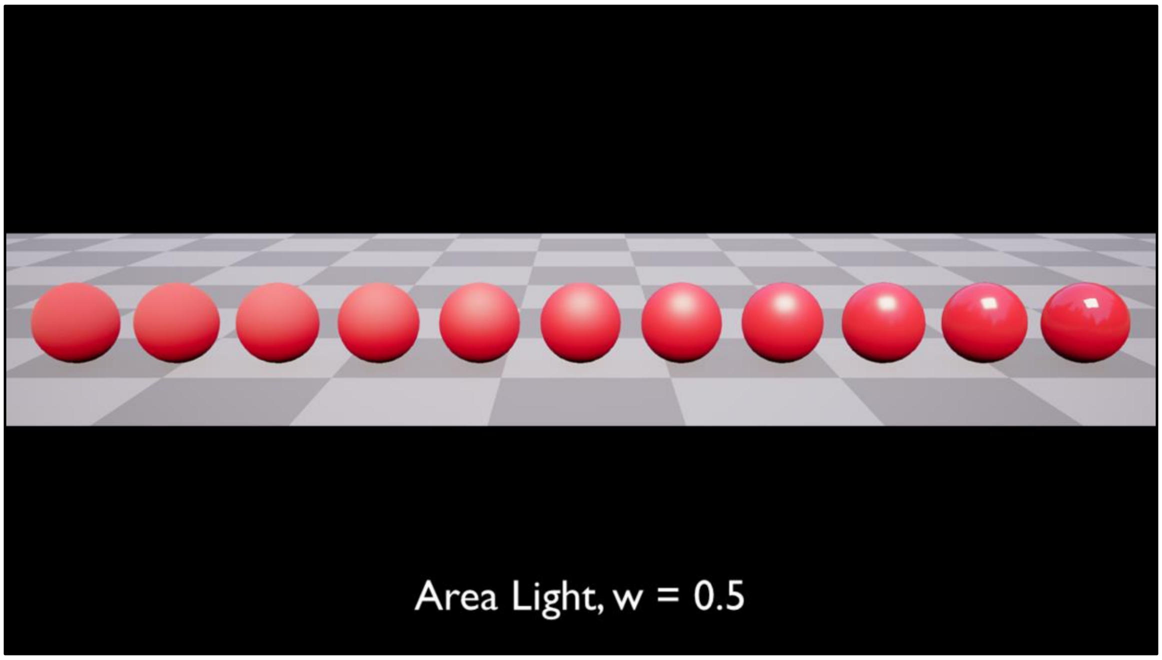
We can wrap 50% of the way around the sphere…


Multiscattering Diffuse, w = 1.0

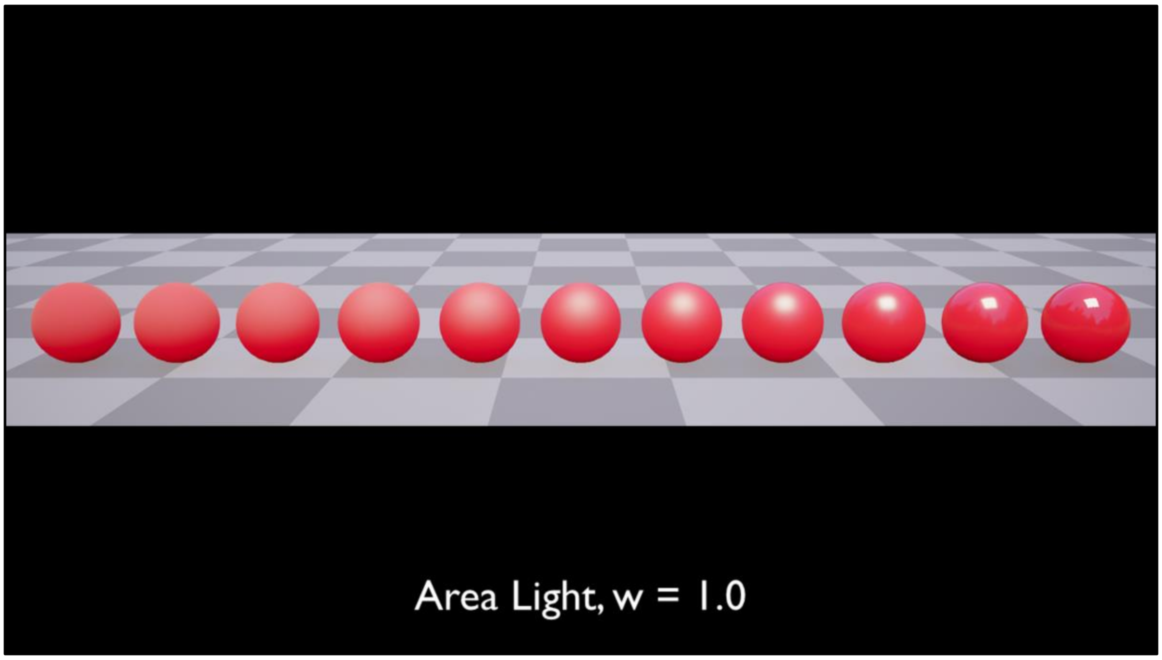
… and 100%, and everything seems to be holding up. The rougher spheres still feel rough with flatter lighting, which is exactly what we’re going for.



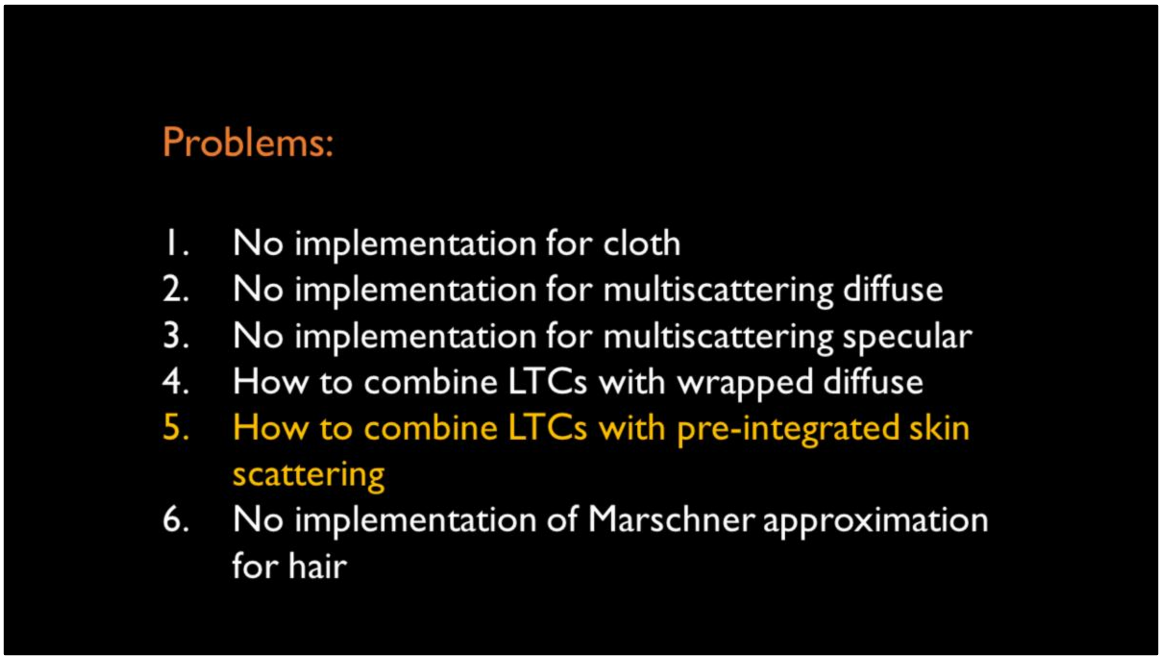
Next we'll check the result with area lights and LTCs.



Once again we can wrap 50% of the way around the sphere…



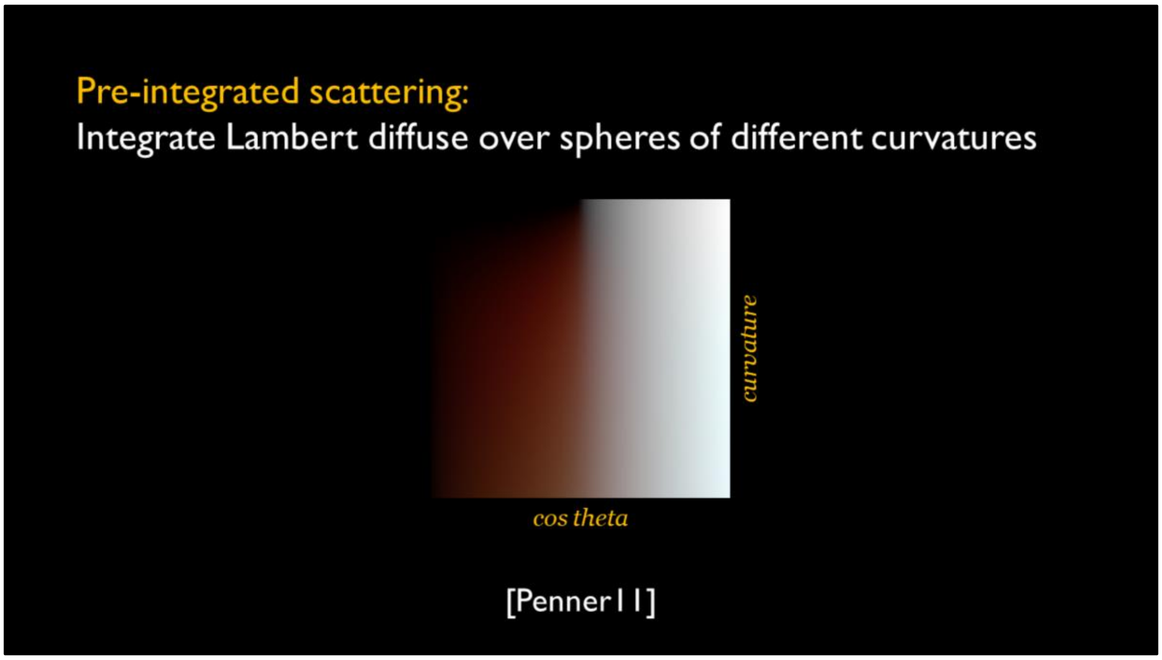
… and 100%, and we're still getting the results we expect. This is pretty awesome, now we can pretty much apply wrapped lighting to everything! Let's consider this problem solved.



Problems:

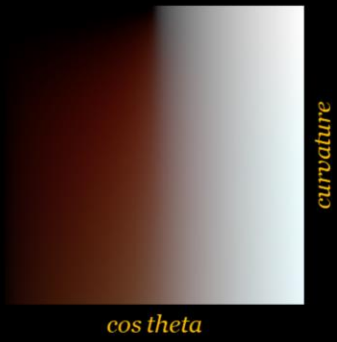
1. No implementation for cloth
2. No implementation for multiscattering diffuse
3. No implementation for multiscattering specular
4. How to combine LTCs with wrapped diffuse
5. How to combine LTCs with pre-integrated skin scattering
6. No implementation of Marschner approximation for hair

Now that we’ve done wrapped diffuse, let’s move onto looking at pre-integrated scattering, which in some ways is very similar…



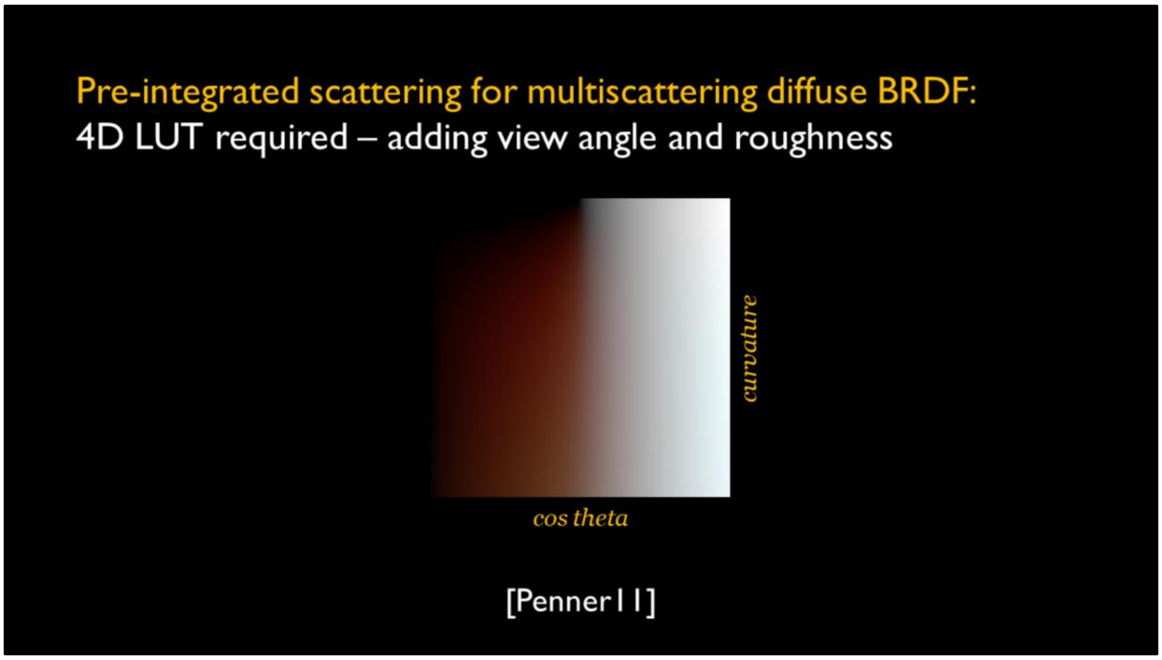
Pre-integrated scattering:

Integrate Lambert diffuse over spheres of different curvatures

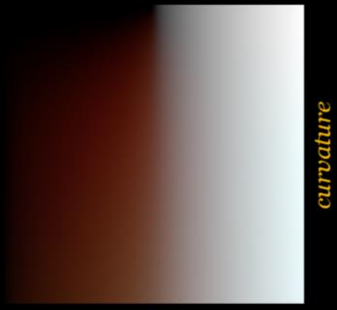


[Penner11]

Pre-integrated scattering for skin is a technique presented by Eric Penner at SIGGRAPH 2011. The trick behind pre-integrated scattering is to integrate Lambert diffuse over spheres of different curvatures, with a skin scattering diffusion profile.

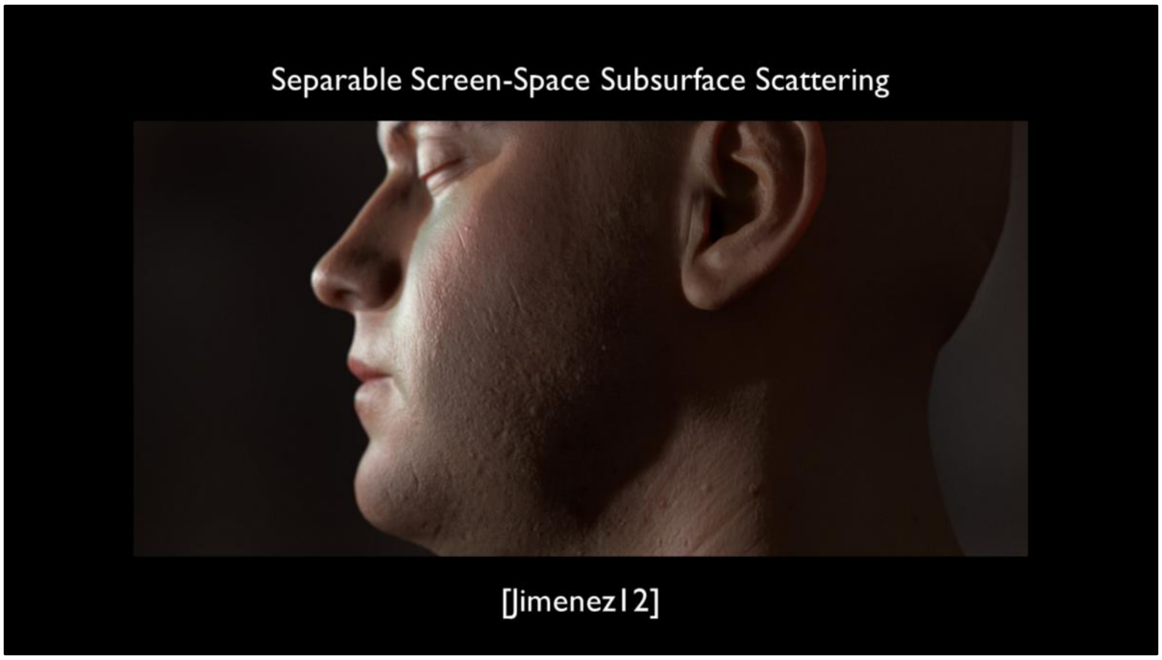


Pre-integrated scattering for multiscattering diffuse BRDF:
4D LUT required – adding view angle and roughness



[Penner11]

If we had to solve this for multiscattering diffuse, we'd have to integrate over the new BRDF… but that has two extra parameters compared with Lambert – viewing angle and roughness, which would make this an unfeasibly large 4D LUT. Of course, perhaps if we were to actually research this, maybe we could find a fit, or reduce the dimensionality… but this still isn't going to solve the answer for area lights.


Separable Screen-Space Subsurface Scattering

[Jimenez12]

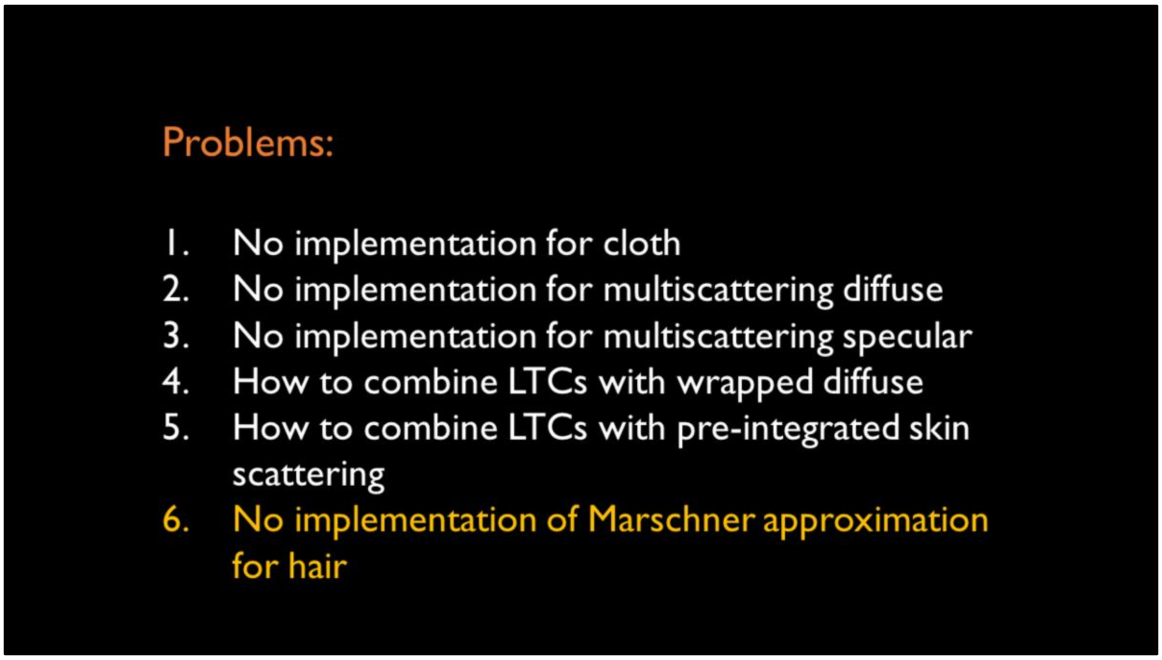
But there is another solution here to skin scattering used in games: separable screen-space subsurface scattering. This runs in screen-space and blurs an existing lighting buffer, so this technically works with any BRDF… and area lights too. Even better, this also will work with our indirect diffuse lighting, which is something that we're missing now.

It seems like back on Far Cry 3 when we had to choose between pre-integrated scattering and screen-space scattering, we bet on the wrong horse. Well, the first was much faster, both in implementation time and performance, but now…? It seems like the latter is much more flexible. Essentially, it's orthogonal to our choices of BRDF and lighting, and that orthogonality can often be really important in rendering techniques.

[Thanks to Jorge Jimenez for permission to use this image]

Sadly this is on our TO DO list for now, and it's something we'll have to return to at a later date. But it seems like it will solve our problems, and be more future proof.

Image credits: Sue Seecof, https://www.flickr.com/photos/126344637@N05/26013859361, licensed under https://creativecommons.org/licenses/by/2.0/

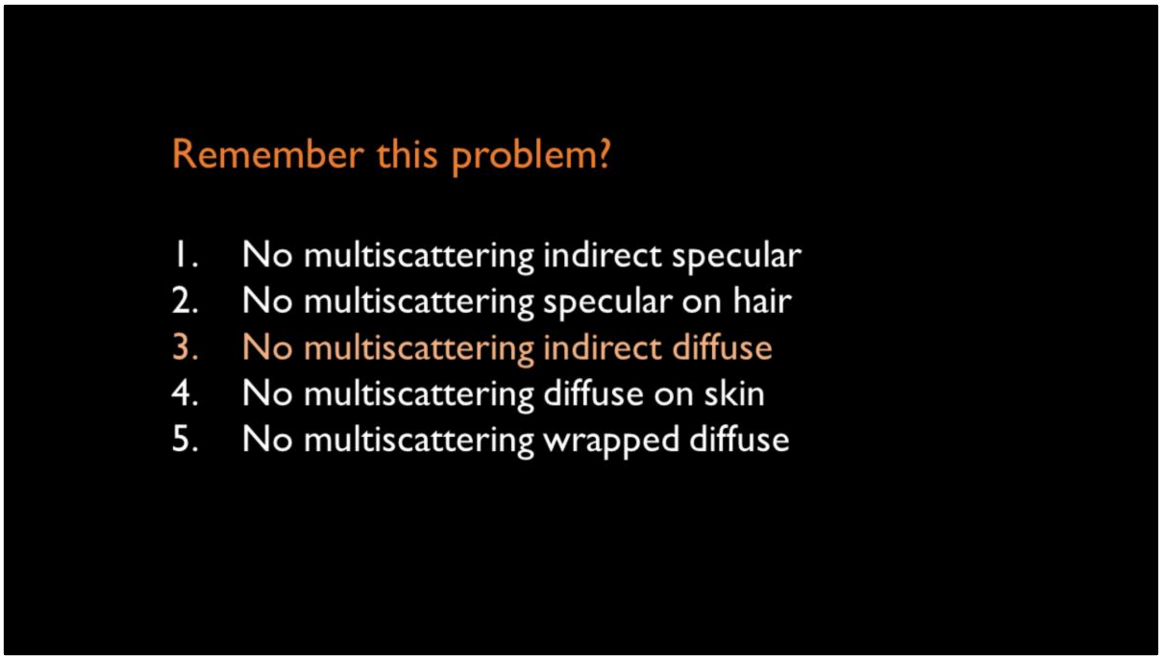

Problems:

1. No implementation for cloth
2. No implementation for multiscattering diffuse
3. No implementation for multiscattering specular
4. How to combine LTCs with wrapped diffuse
5. How to combine LTCs with pre-integrated skin scattering
6. No implementation of Marschner approximation for hair

Our final unsolved problem is how to apply LTCs to hair.

Well, sadly this is also on our TO DO list. Marschner hair has three lobes, essentially a reflection, a transmission, and a second-bounce reflection. Maybe this needs to be really expensive and we need to do LTC look ups for each lobe? Or perhaps we'll have to find a different area light approximation, or just fall back to our point light solution. But once again, like for multiscattering specular, there doesn't seem to be an intuitive answer for hair. Any help is of course appreciated!

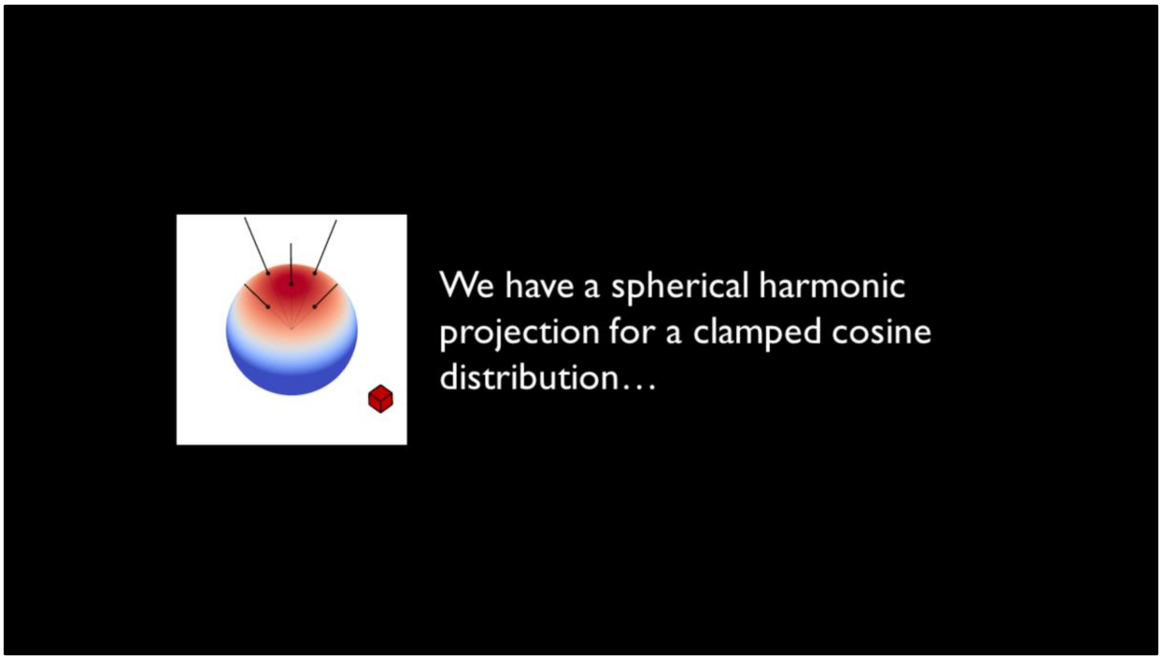
Image credits: Sue Seecof, https://www.flickr.com/photos/126344637@N05/26013859361, licensed under https://creativecommons.org/licenses/by/2.0/



Remember this problem?

1. No multiscattering indirect specular
2. No multiscattering specular on hair
3. No multiscattering indirect diffuse
4. No multiscattering diffuse on skin
5. No multiscattering wrapped diffuse

But there’s one more thing… remember this unsolved problem? How do we apply our multiscattering diffuse to our indirect lighting, which is stored in spherical harmonics?

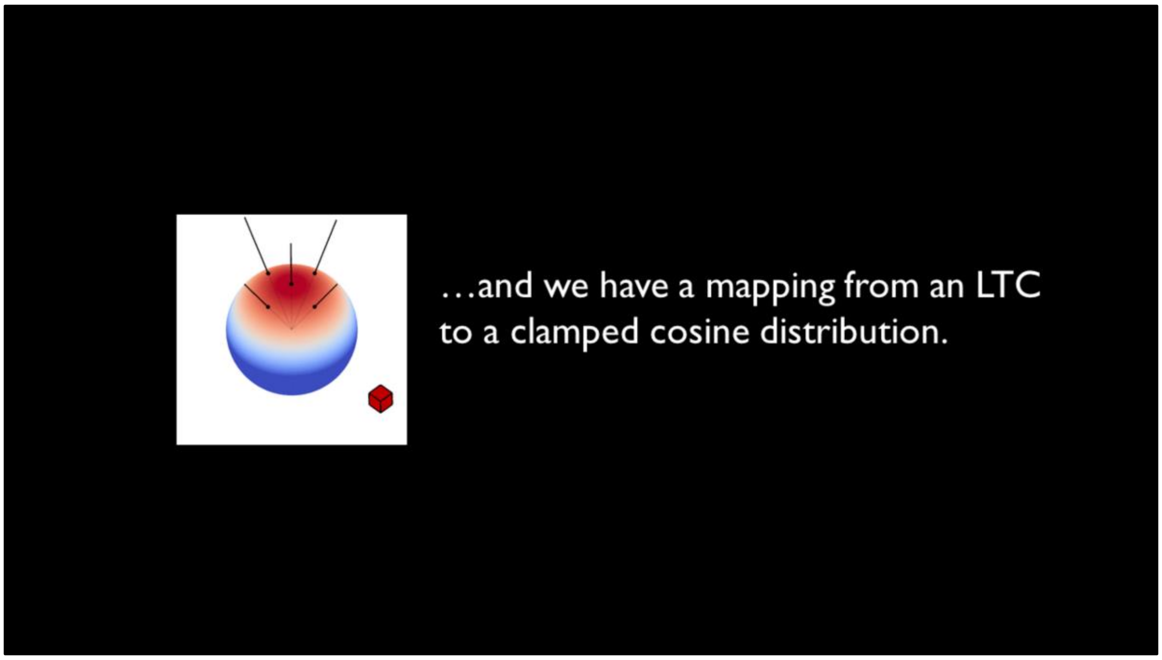


We have a spherical harmonic projection for a clamped cosine distribution…

Well… once again there's a link to a problem we've already solved…

We can project a clamped cosine distribution into spherical harmonics, and use that to evaluate lighting projected into SH. But what we want is to project any BRDF (in this case our multiscattering diffuse BRDF) into spherical harmonics.

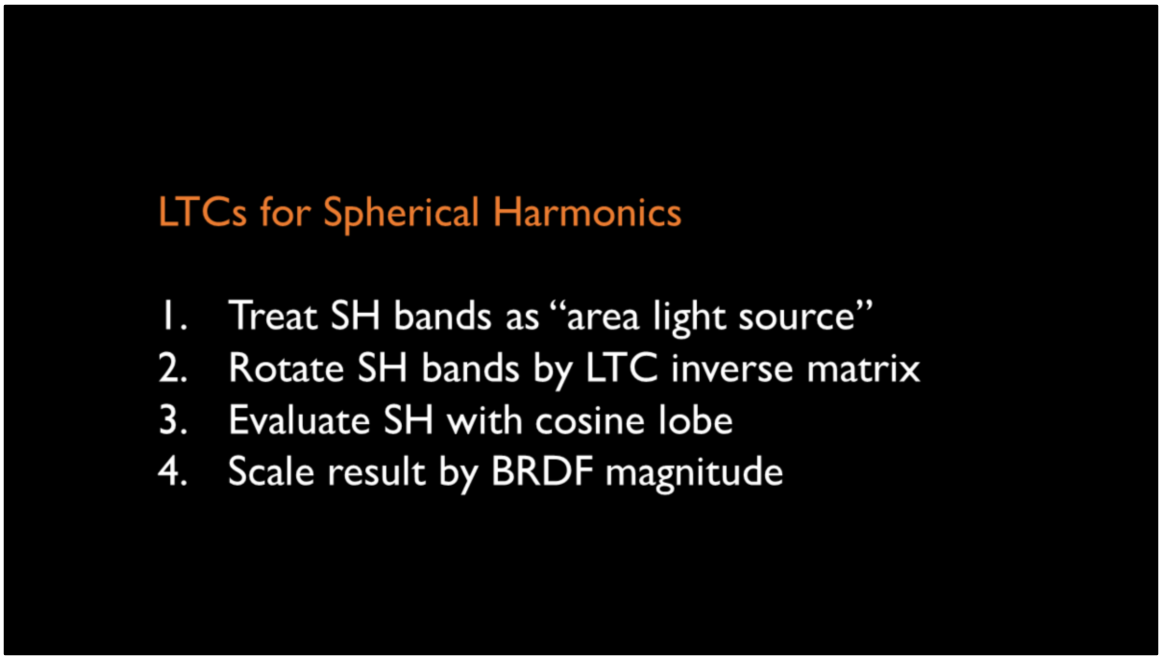
[Thanks to Eric Heitz for the diagram]



…and we have a mapping from an LTC to a clamped cosine distribution.

But it just so happens that now we have a mapping from a linear-transformed cosine, with which we can approximate a BRDF, to a clamped cosine distribution! So perhaps we can use that to solve our problem.

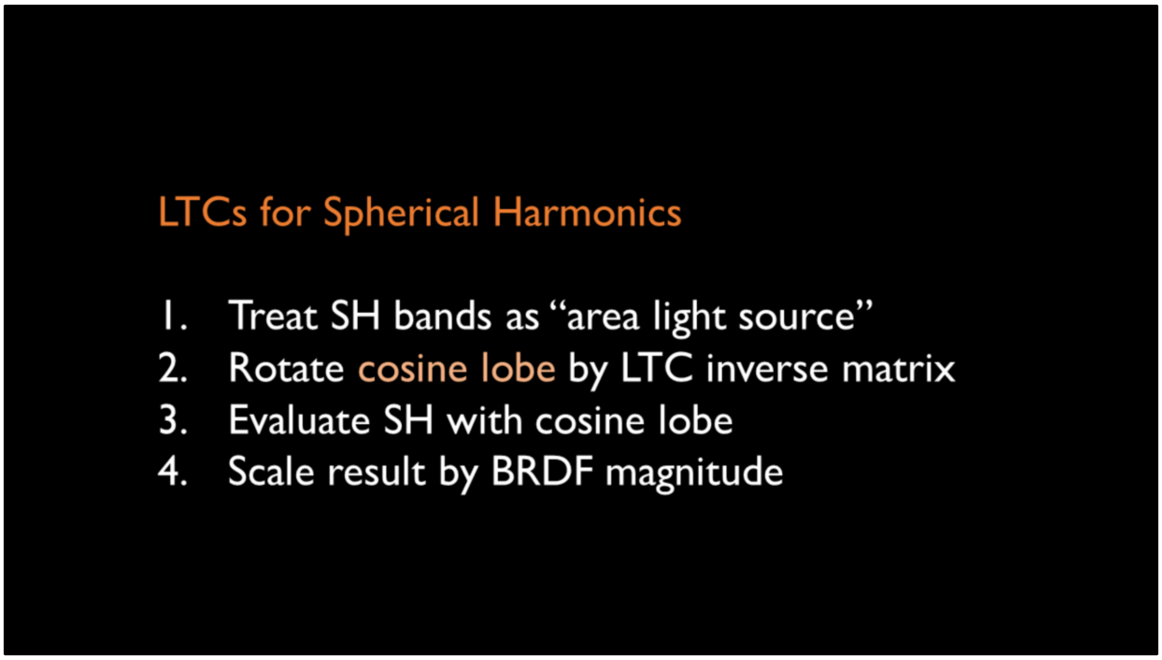
[Thanks to Eric Heitz for the diagram]



LTCs for Spherical Harmonics

1. Treat SH bands as "area light source"
2. Rotate SH bands by LTC inverse matrix
3. Evaluate SH with cosine lobe
4. Scale result by BRDF magnitude

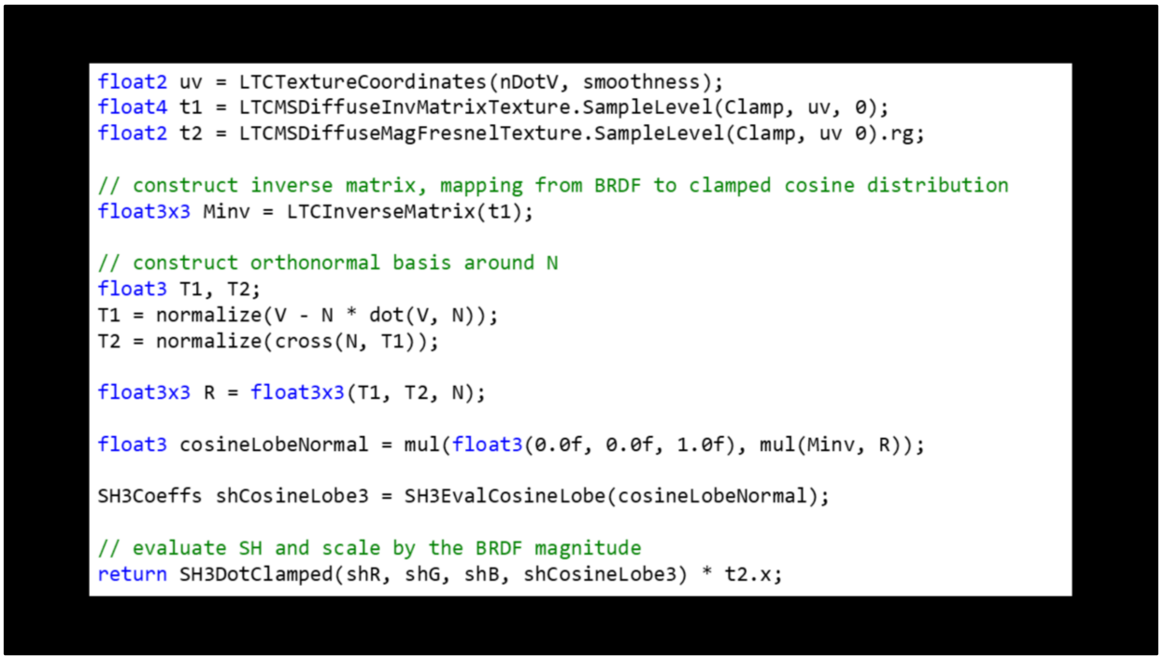
For area lights, we take the coordinates of our light source and rotate them using the LTC inverse matrix. We're going to treat our SH bands like an area light source and rotate them using the LTC inverse matrix, then evaluate by the cosine lobe. Then finally, we just scale the result by the BRDF magnitude like when we use LTCs for area lights.



LTCs for Spherical Harmonics

1. Treat SH bands as “area light source”
2. Rotate cosine lobe by LTC inverse matrix
3. Evaluate SH with cosine lobe
4. Scale result by BRDF magnitude

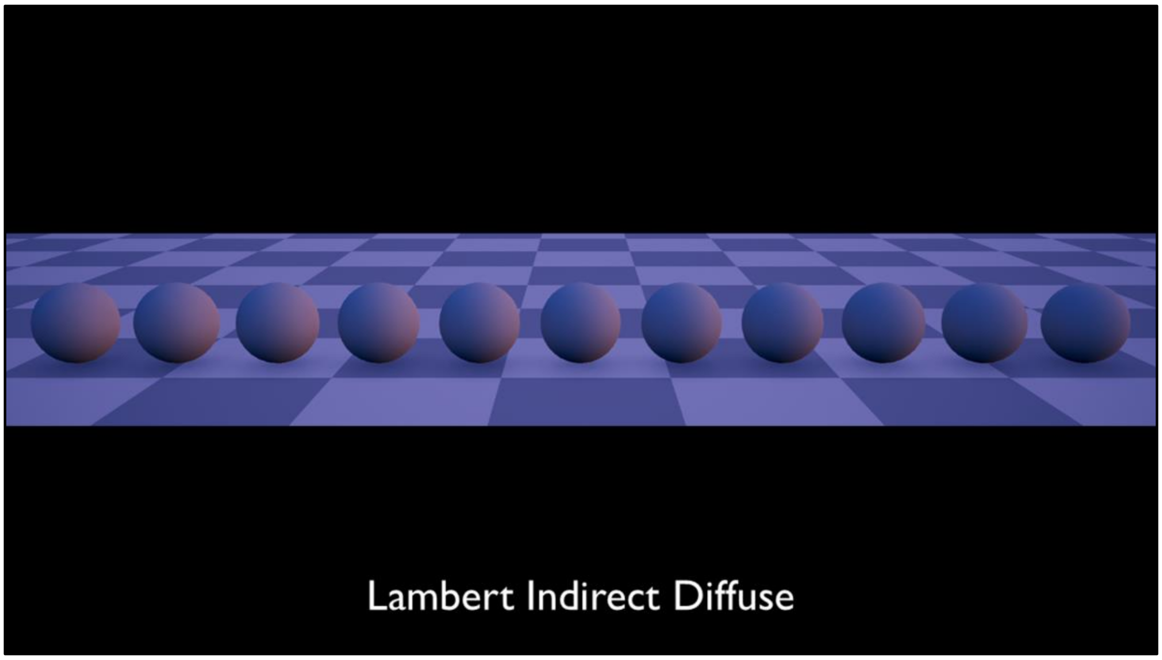
But, as SH is rotationally invariant, we can actually just rotate our cosine lobe instead, which is much faster!



```
float2 uv = LTCTextureCoordinates(nDotV, smoothness);
float4 t1 = LTCMSDiffuseInvMatrixTexture.SampleLevel(Clamp, uv, 0);
float2 t2 = LTCMSDiffuseMagFresnelTexture.SampleLevel(Clamp, uv 0).rg;

// construct inverse matrix, mapping from BRDF to clamped cosine distribution
float3x3 Minv = LTCInverseMatrix(t1);

// construct orthonormal basis around N
float3 T1, T2;
T1 = normalize(V - N * dot(V, N));
T2 = normalize(cross(N, T1));

float3x3 R = float3x3(T1, T2, N);

float3 cosineLobeNormal = mul(float3(0.0f, 0.0f, 1.0f), mul(Minv, R));

SH3Coeffs shCosineLobe3 = SH3EvalCosineLobe(cosineLobeNormal);

// evaluate SH and scale by the BRDF magnitude
return SH3DotClamped(shR, shG, shB, shCosineLobe3) * t2.x;
```

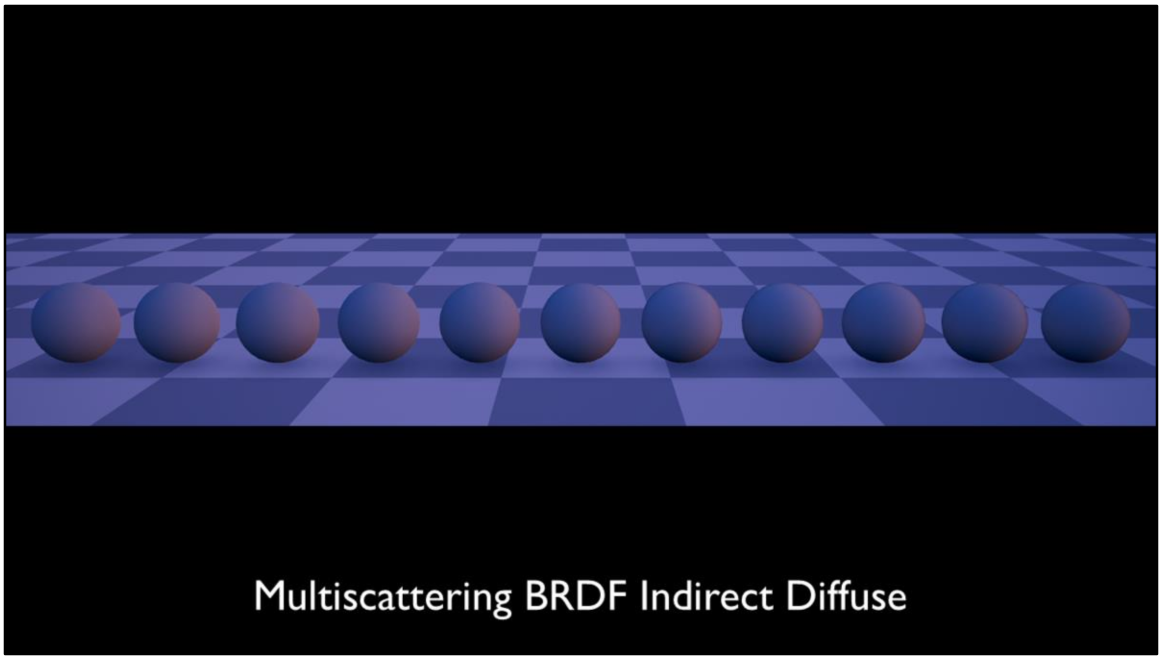
Here’s an example of some shader code. Note that as we’re rotating the SH also into the space of the normal with the matrix R, we’re rotating a cosine lobe that’s pointing straight up. That means that you can actually do some nice further optimisations to this code.

Now it’s time to look at some results.



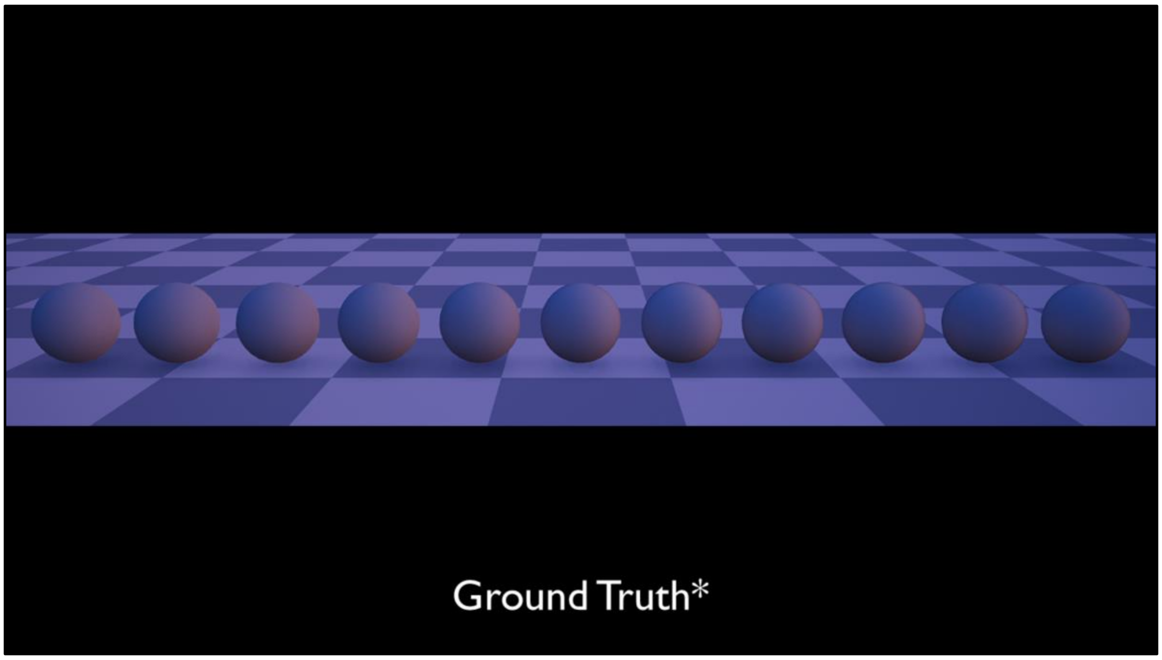
I’ve changed the lighting set up a bit for these shots. Here we are at dawn, with sky lighting only. You can see the sun is about to rise to our right hand side and the sky is a beautiful pink in that direction.

Now if we turn our multiscattering indirect diffuse on…



You can see we get flatter lighting on the rough left hand spheres, but then a stronger falloff for glancing viewing angles as the spheres get smoother towards the right. The ground plane is also getting a little darker, as we saw for the multiscattering direct diffuse, so this is definitely going to have an impact on our world lighting.

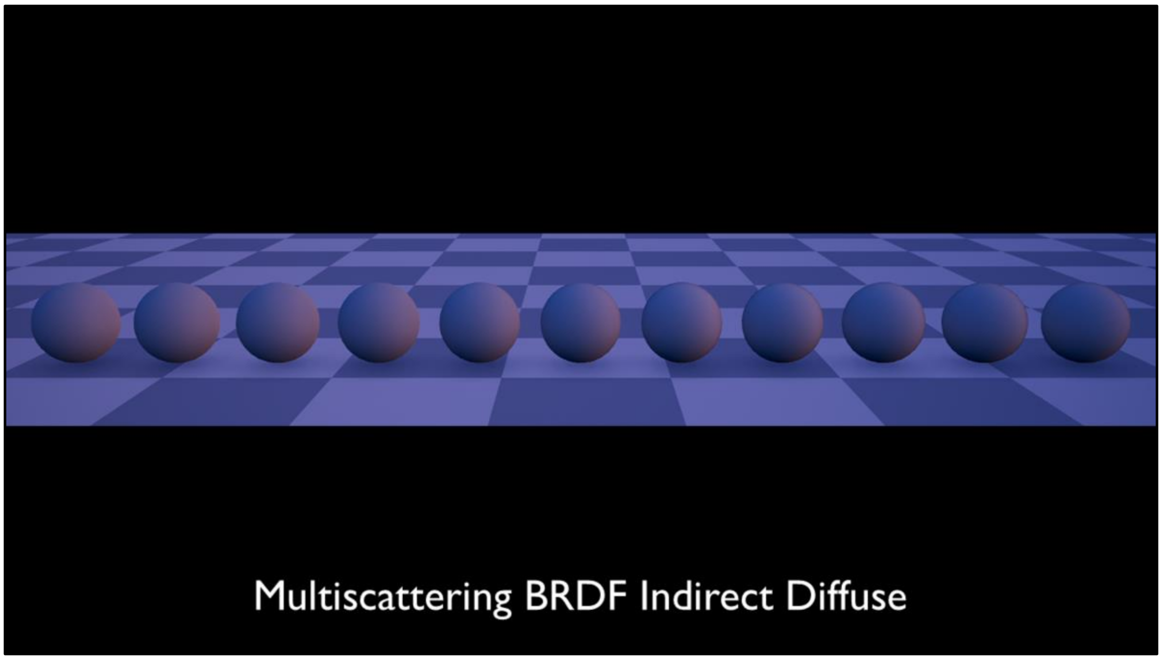
Let's compare against the ground truth…



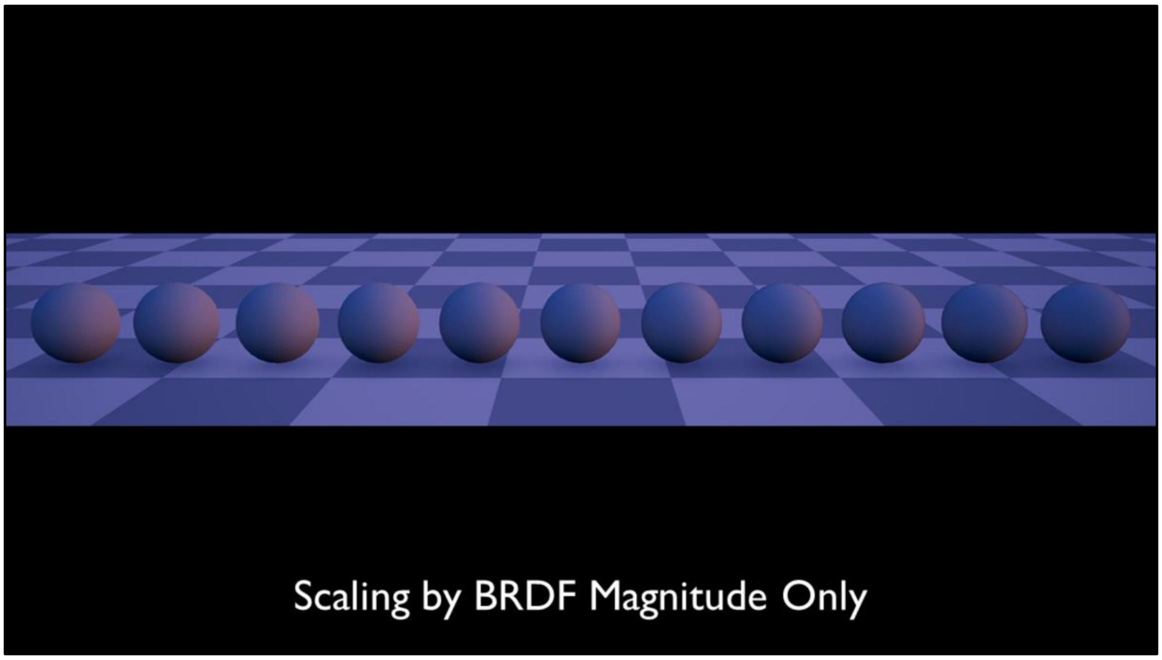
This is really close! In fact, the only real difference we see is that the bottom of the spheres are lighter for the ground truth, but if we were to compare Lambert we'd actually see the same – this is caused by SH ringing.

I put an asterix by ground truth because I always feel a little wary about claims of absolute truth in these situations. We generated this by sampling the SH over the hemisphere and scaling by the BRDF.

Let's go back to look at our approximation…

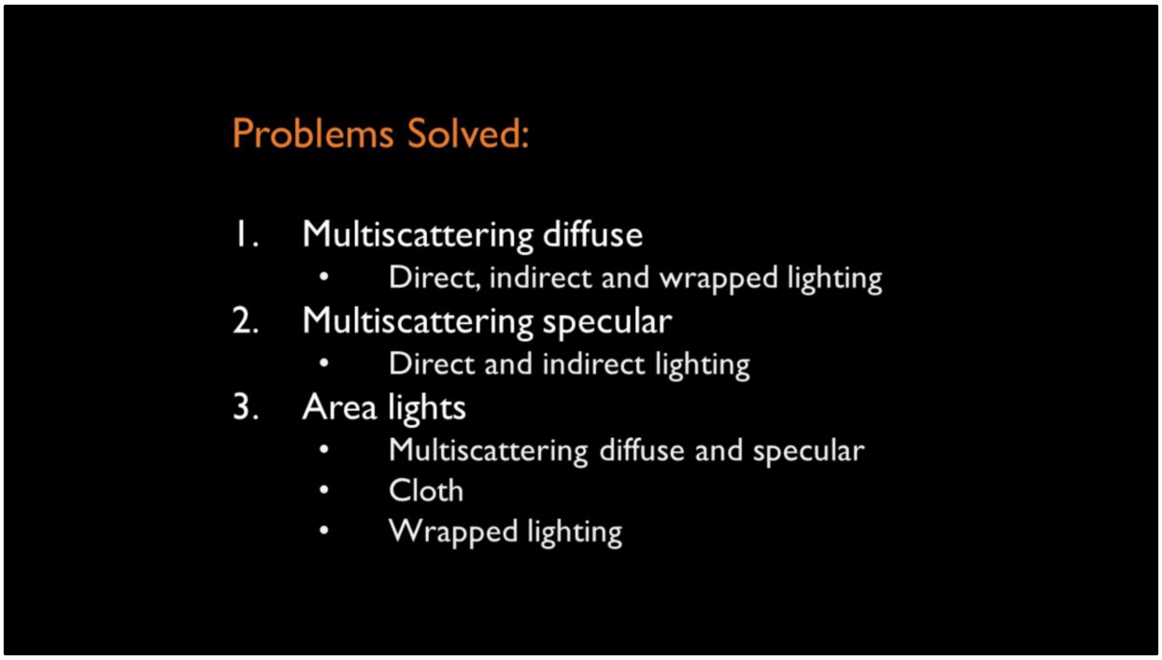


So this is great, but in case you feel those matrix multiplies and construction of an orthonormal basis might take too much time… I have a cheaper solution. Don’t rotate the SH at all, but still scale by the BRDF magnitude. Let’s compare.



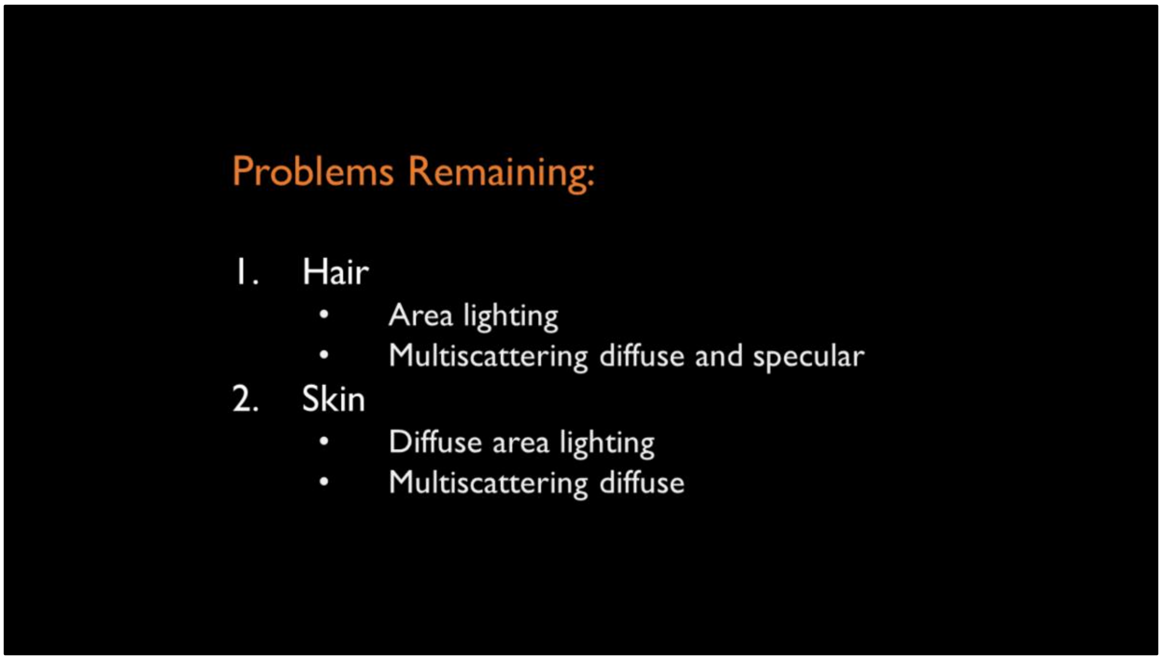
So the colours shift, but the overall intensity stays the same. This is a pretty extreme example of lighting being different in the SH too, so perhaps in general cases it doesn't matter so much. With scaling by the magnitude though you're still going to get that falloff at glancing viewing angles for smooth surfaces which gives more definition to the objects, as well as having that effect on the overall luminance of the scene that darkening the ground plane will have.

This needs a little further research, but I hope this is a promising avenue to go down!


Problems Solved:

1. Multiscattering diffuse
   - Direct, indirect and wrapped lighting
2. Multiscattering specular
   - Direct and indirect lighting
3. Area lights
   - Multiscattering diffuse and specular
   - Cloth
   - Wrapped lighting

Let’s conclude by recapping where we’re at. We’ve implementing multiscattering diffuse and specular, as well as area lights. We’ve made our BRDFs work for direct and indirect lighting, as well as area lighting. Wrapped lighting works in all circumstances, and we have an implementation of area lighting for cloth.



Problems Remaining:

1. Hair
   - Area lighting
   - Multiscattering diffuse and specular
2. Skin
   - Diffuse area lighting
   - Multiscattering diffuse

But sadly we still have the following problems left to solve. Nothing currently works with our hair shading, and neither for subsurface scattering on skin. At least for skin we have a plan of using separable screen-space subsurface scattering, but hair is definitely in the research area.

What have we learned?

So we're now at the end of our case study. We've tried to take some latest research into new multiscattering BRDFs and area lights, and put them into our game, and it's been a bit more complicated than we initially would have hoped. But hopefully this journey has given us a few insights that we can learn together about uniting research and game production.

Takeaway #1: Source code is invaluable

The first is that source code is invaluable. Having some simple shader code for Danny Chan’s multiscattering diffuse model was great, as we could implement it quickly. Having demos for area lights with LTCs also meant I had a working version of area lights in the game very quickly. But perhaps even more importantly, having the fitting code for LTCs was utterly invaluable because when I hit a hurdle and needed to work with more BRDFs than just Lambert diffuse and GGX, I was able to do so very easily.

Takeaway #2: Separate insight from implementation

This leads us nicely into Takeaway #2 – separating insight from implementation. The thing about LTCs is not that they allow us to get a real-time result of GGX specular with area lights, although that's what the demos show. The research isn't about that implementation detail. The research is about the insight that we can use LTCs to represent any BRDF. I just used that insight further myself to use LTCs to help with indirect diffuse with spherical harmonics.

This is good for us all to remember when writing papers but also when reading them. I remember a member of my team implementing a paper about a BRDF for the moon. It contained a BRDF from the sun light, but it also provided a sky light model too. Really, the sky light model was just there to make the image the researcher was rendering be complete. However, my colleague implemented both the sun model and the sky model, completely forgetting that we already had a much better sky model in game that was being applied. Now, that's the job of us in games to spot the real insights in paper, and to forget the peripheral information, but sometimes it's the job of researchers too to ensure the key insight in their paper is what stands out, not just some arbitrary implementation details.

That isn't to say we can't have papers 100% about implementation. In some ways the

Fdez-Agüera paper is a fantastic example of this – focusing on how to get something working in a particular case, and as a game developer I love these papers. *But* I did personally take the model and extended it to more use cases, looking at the insight it provided into other problems.

Takeaway #3: Build on successful existing work

I guess I was building upon existing work. As was Fdez-Agüera when looking at multiscattering for environment lighting, or Emmanuel Turquin when developing his own multiscattering model. The more research we have in a given area the better! It's even more important because if work is successful, we're likely using it in games. And doing research to push what's being used further really helps.

Even the concept of the environment map BRDF is something that has come up repeatedly. We were able to use it to get the energy of a BRDF in a given direction, and we also used a very similar concept to scale our LTCs. The fact that in games we already have some of this technology just makes implementing future papers based on that easier.

There is a word of caution here too. It's looking likely that in the near future I'm going to be using LTCs to evaluate BRDFs, because I want to use area lights. That means if I'm going to implement a BRDF, I really want it to work with LTCs, otherwise I'm going to have a problem. Building on successful existing work also means understanding what's already being used, and trying to make your research work with that. Or understanding that we might only come back to your research in 15 years once we've stopped doing LTCs and we're all on ray tracing.

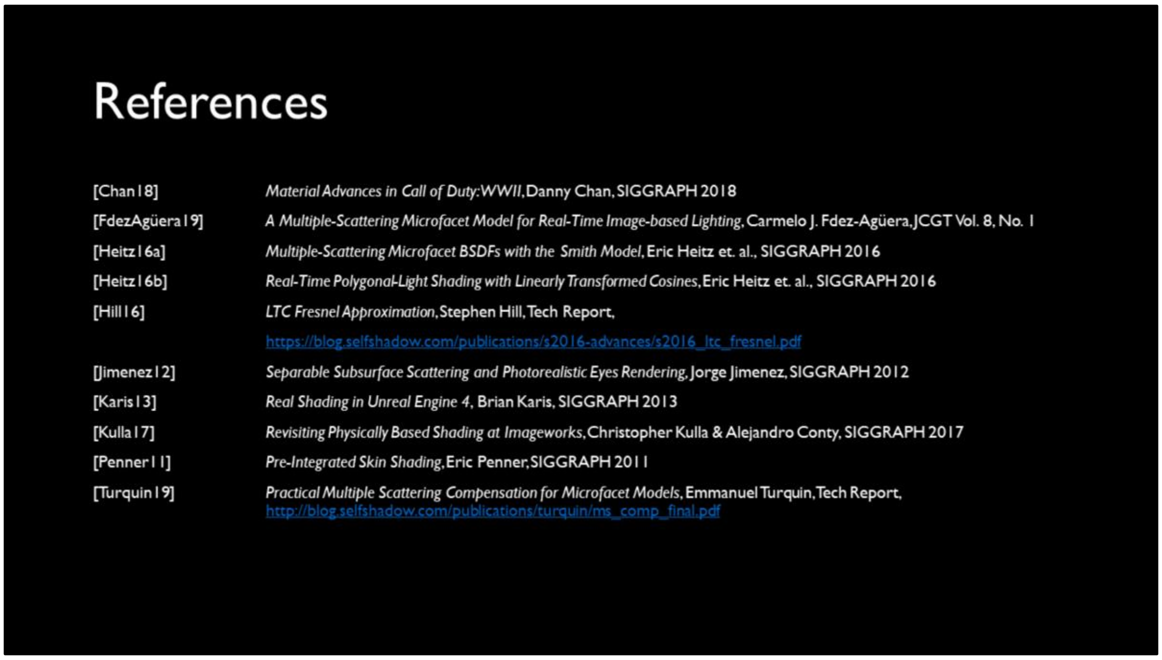
But the other benefit to building on existing work is that you might be solving problems that are within it. And today we hit a lot of problems.

Takeaway #4: Never underestimate implementation time

Never underestimate implementation time. And that’s coming from me, who was initially really pleasantly surprised with how quickly I got new BRDFs plus area lights into the game. Then I realised all the cases that I was missing. The skin, the hair, the cloth. How the new BRDFs didn’t necessarily play nicely with the area lights, and that those two strands of research were pretty independent that I had to try to unite. Still, there’s much more work to do – I need to implement a whole new technique for skin shading, check that it’s performant, and discover a solution for hair.

I say this to many people – to artists and producers I work with, to hardware manufacturers wanting us to implement their latest feature, to academics wanting us to implement their research, and to graphics programmers trying to get the latest feature into their engine. It often takes a lot longer than you think. Sure, the initial implementation might be simple, it might not take that much time. But I like to think that this is one of those 80:20 rules. The last 20% of the work very definitely takes 80% of the time, because our engines are incredibly complex and features don’t always play nicely together.

Thank you all for listening. A particular thanks to the I3D committee for inviting me to speak, everyone who's research inspired everything presented here, Eric Heitz and Jorge Jimenez for letting me use their pictures and diagrams, and finally Natalya Tatarchuk and Ulrich Haar for reviewing my slides. I hope you learned something about implementing research and rendering advances into game productions, and I hope those of you who are visiting Montreal enjoy our city. Have a great rest of the conference!



References

[Chan18] *Material Advances in Call of Duty: WWII*, Danny Chan, SIGGRAPH 2018

[FdezAgüera19] *A Multiple-Scattering Microfacet Model for Real-Time Image-based Lighting*, Carmelo J. Fdez-Agüera, JCGT Vol. 8, No. 1

[Heitz16a] *Multiple-Scattering Microfacet BSDFs with the Smith Model*, Eric Heitz et. al., SIGGRAPH 2016

[Heitz16b] *Real-Time Polygonal-Light Shading with Linearly Transformed Cosines*, Eric Heitz et. al., SIGGRAPH 2016

[Hill16] *LTC Fresnel Approximation*, Stephen Hill, Tech Report, https://blog.selfshadow.com/publications/s2016-advances/s2016_ltc_fresnel.pdf

[Jimenez12] *Separable Subsurface Scattering and Photorealistic Eyes Rendering*, Jorge Jimenez, SIGGRAPH 2012

[Karis13] *Real Shading in Unreal Engine 4*, Brian Karis, SIGGRAPH 2013

[Kulla17] *Revisiting Physically Based Shading at Imageworks*, Christopher Kulla & Alejandro Conty, SIGGRAPH 2017

[Penner11] *Pre-Integrated Skin Shading*, Eric Penner, SIGGRAPH 2011

[Turquin19] *Practical Multiple Scattering Compensation for Microfacet Models*, Emmanuel Turquin, Tech Report, http://blog.selfshadow.com/publications/turquin/ms_comp_final.pdf