Physically Based Shading

Joshua Bainbridge
MSc Computer Animation and Visual Effects,
Bournemouth University

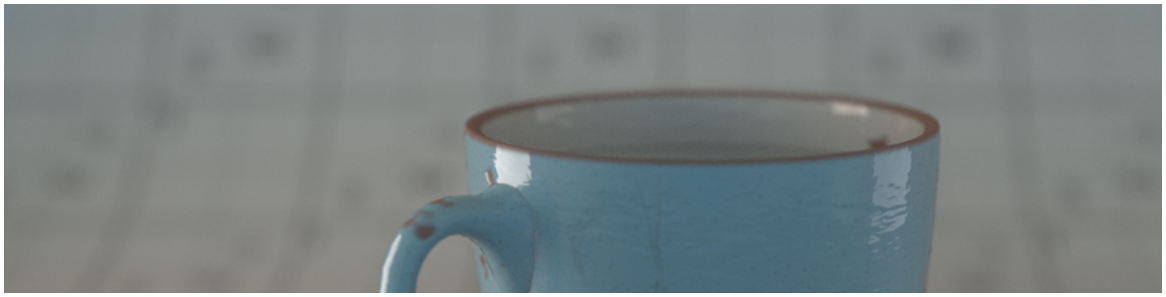

Figure 1: *Ceramic Cup 2015*

Abstract

This paper outlines the development of a generalised shading model and its implementation. The chosen example subject is a ceramic cup, demonstrating a layered approach to shader development. The design will be based upon physical laws reflecting modern trends in global illumination and as result should be robust in a variety of lighting conditions. The rendering context will be Photorealistic Renderman 18 accessed through the API and python bindings, while using standard RSL 2.0 for shader development.

Keywords: physically based, shading, global illumination

1 Reference

Once I had found a subject I started taking reference images and measurements. To do this I used a long focal length to minimise perspective distortion as well as natural lighting to get a good representation of colour. I then also photographed specific features of the

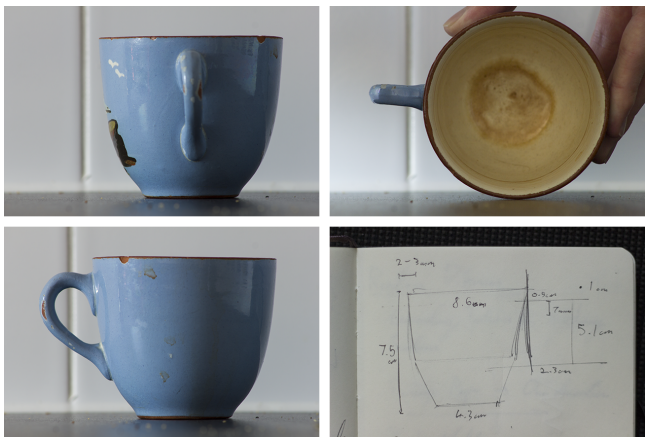

Figure 2: *Reference*

cup, such as chips, coffee stains and lime scale as seen in Figure 3. These are all things that will need to be synthesised when composing the shaders. The composition of the shader will need to reflect the basic elements that the mug is constructed from as well as the aforementioned features. These break down into the clay base as well as the ceramic glazing which is itself composed of two elements. The first of these being the the outer glass layer, usually formed of silicon dioxide which is an insulator (dielectric) material and as result does not alter the colour of reflection. The second is the pigmentation that has a diffused response and is composed of metal oxides that conduct electromagnetic energy a in wave-length dependent manner. This is what causes the change in colour on the surface. Construction of these elements will use a layering system where one can mask or reveal a layer beneath. As for the ceramic glazing, this will be contained within a single shader to allow for easy development when combining specular and diffuse distributions.

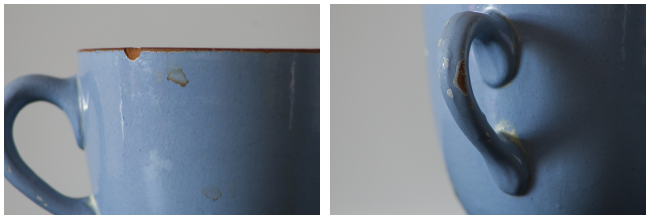

Figure 3: *Features*

2 Theory

In this project I will be using the ray-trace hider within Renderman while limiting the distribution of rays to a single path. This process is called path tracing and was originally developed in a paper called "The Rendering Equation" [Kajiya 1986] by Kajiya. The equation can be written in the form:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega f_r(x,\omega_i,\omega_o) L_i(x,\omega_i)(\omega_i \cdot n) d\omega_i$$

This describes a recursive integral of passing electromagnetic energy across the hemisphere of a single position on a surface. These sample positions then undergo convolution across a two dimensional array forming an image. It is often better to express this in terms of an integration of path space as described by Eric Veach [Veach 1998] rather than a recursive approach as this allows for a clearer explanation of advance techniques. Kajiya explained that the contribution of each vertex along a path is equal or less than the previous

vertex. This means that all vertices lower in a distributed tree have greater importance than the ones higher, resulting in more optimal sampling when branching is reduced.

Russian roulette [Szcsi et al. 2003] is an optimisation that introduces noise but can reduce computation considerably by reducing ray branching and give early termination of paths with minimal contribution. It involves randomly selecting between options based upon their probabilities and then dividing the resulting value by that probability resulting in an unbiased sample. This can be done when deciding on whether or not to terminate a path or when selecting a layer to sample from a material. Russian roulette has been implemented to reduce ray branching and optimise sampling in both these ways within the layering system.

The rendering equation can also be separated into two distinct integrals as described by Philip Dutre [Dutre et al. 2006] and is written as:

$$L(x \to \Theta) = L_e(x \to \Theta) + L_r(x \to \Theta)$$

$$L_r(x \to \Theta) = \int_{\Omega_x} f_r(x, \Psi \to \Theta) L(y \leftarrow -\Psi) cos(N_x, \Psi) d\omega_\Psi$$

$$= L_{direct} + L_{indirect}$$

$$L_{direct} = \int_A f_r(x, \overrightarrow{xy} \to \Theta) L(y \to \overrightarrow{yx}) V(x,y) G(x,y) dA_y$$

$$L_{indirect} = \int_{\Omega_x} f_r(x, \Psi \to \Theta) L_i(y \leftarrow -\Psi) cos(N_x, \Psi) d\omega_\Psi$$

This is used within the shaders to deterministically sample the lights directly resulting in what is commonly called next event estimation. Renderman's pipeline methods for RSL facilitate such separation making implementation within the shader relatively simple.

For direct sampling, different strategy can be used and combined optimally using multiple importance sampling. This is done when creating samples from both the shader as well as the light and weighting the two using a heuristic. The heuristic however must be chosen carefully as it can introduce variance when not optimal or when the sampling strategies used by both the light and/or the shader are poor representations of their evaluation. Multiple importance sampling can be written as

$$\langle I_j \rangle = \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(X_{i,j}) \frac{f(X_{i,j})}{p_i(X_{i,j})}$$

where $X_{i,j}$ represents the samples produced by each distribution strategy and $w_i(X_{i,j})$ is the heuristic that weights the samples contribution according to its probability density. Veach [Veach 1998] defined what he calls the power heuristic that is near optimal as long as the distributions of both the shader and light are good. Multiple importance sampling using Veach's power heuristic can be handled externally from the shading context in RSL as long as the shaders distribution function conforms to the standard interface.

This interface involves two methods. The first of which is a means to approximately create samples according to the BRDF and also return their probability. An example expressed below creates samples according to a cosine around the normal

$$x = \cos(2\pi r_1)\sqrt{1 - r_2}$$

$$y = \sin(2\pi r_1)\sqrt{(1 - r_2)}$$

$$z = \sqrt{r_2}$$

where r_i represents a uniform random variable. These equations are derived by establishing a pdf that approximates the BRDF and calculating its cdf. This is then inverted and sampled producing the correct distribution according the original pdf. Then all that is needed is to transform the two samples from spherical coordinates into cartesian space. The probability is the dot product between the sample direction and normal of the surface. It is written in the form:

$$\frac{\cos\theta}{\pi}$$

This distribution is then rotated around the direction of reflectance. Using this distribution could result in samples falling beneath the surface resulting in zero contribution making it less efficient than more advanced methods such as described by Walter [Walter et al. 2007]. The second method needed is the evaluation of the BRDF itself. Both of these methods are passed a structure that holds all samples and probabilities for both the lights and the shaders. This is then used for multiple importance sampling. The probability is also important to maintain an unbiased numerical integration using a Monte Carlo estimator because the sampled value must be multiplied by the inverse of the distribution probability.

Calculating the distribution at the surface requires evaluating the BDRF. There are many different models of BRDF but to be physically plausible it must be energy conserving and behave with reciprocity. This means that the surface cannot distribute more light than it receives and if the incident and reflecting directions were to be swapped, the evaluation would still be the same. Another property required by a physically plausible BRDF is that it take into account micro-facet theory. This is the evaluation of how much the surface normal may vary on a statistical level creating a more accurate distribution. There are two different types of micro-facet BRDFs, the first is a specular model where each facet acts as a perfect reflector, the second being a diffuse model were the statistical facets behave with a lambertian response.

Micro-facet BRDFs are generally based upon the same Cook-Torrance [Cook and Torrance 1982] model. This can be written as

$$f(l, v) = \frac{F(l,h)G(l,v,h)D(h)}{4(n.l)(n.v)}$$

where l and v are the look and view directions while h is the half-way vector. The functions F, G and D represent the Fesnel, geometric and distribution parts respectively. The half-way vector is calculated by normalizing the sum of both the light and the view vectors. The denominator is a normalization factor as described by Cook.

The Fresnel part describes how light reflects at grazing angles according to the densities of the current media and the surface. Here we use Shlick's approximation (for non-metal surfaces) that can be written as

$$F(l,h) = (\frac{n_1 - n_2}{n_1 + n_2})^2 + (1 - (\frac{n_1 - n_2}{n_1 + n_2})^2)(1 - \cos(l.h))^5$$

where n_1 and n_2 are the indices of refraction for the current medium and the surface material.

As for the distribution function, this is largely what defines the different BRDFs from each other and is the probability of a facet facing the half-way vector. We will use the Beckham model as it gives reasonable results for the computational cost. It can be written in the form

$$D(h) = \frac{e^{\frac{(n.h)^2 - 1}{r^2 (n.h)^2}}}{\pi r^2 (n.h)^4}$$

where e is the base of natural logarithms and n is the macro normal of the surface. The r variable represents the surface's roughness giving the distribution a wider lobe with higher values.

The geometric function represents the occlusion of micro-facets and has two parts. The first part is the visibility of the facet in the view direction. If the facet is not visible then it does not contribute any radiance. The second represents the visibility in the light direction and again if not visible, it wont contribute radiance. It can be written as

$$G(l, v, h) = min\left\{1, \frac{2(n.h)(n.v)}{(v.h)}, \frac{2(n.h)(n.l)}{(v.h)}\right\}$$

where n is the macro normal of the surface.

3 Implementation

For implementation I started by constructing the geometry from the reference and measurements. The model was built to scale as this would make a difference when lighting the scene. Accessing the Renderman API was done through the python bindings for multiple reasons. Primarily the choice was due to having experience using RIB and wanting to learn something new. It also allowed me to compile the shaders when running the script which made testing shader development easer. Lastly it meant I could write a parser to take an .obj and convert it to a format required by the API. I did not write the original python code for parsing the file but instead edited a sample from online, this has been cited in the source. The program works by parsing in a single .obj file as an argument (must have UVs) when running the script from bash.

Figure 4: *Loaded Objects*

I started lighting the scene by adding two standard physically based area lights. One on either side of the model with varying intensities and temperature. I then created a physical environment light that used one of the environment maps shipped with Renderman to save on space. It was important that the shaders for these light were built using the pipeline methods in RSL and had the correct interfaces to work with my own surface shaders.

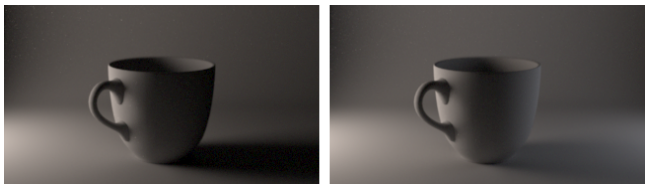

Figure 5: *Area lights*

For the render options I set the hider to ray-trace and the integration mode to path. This would later then be detected by the shaders and control the sampling count. Using "it" also allowed for progressive rendering which meant instant feedback when later developing the shaders. I also added slight depth of field to the camera for realism.

I didn't want to just create a single shader for representing a single object but rather look into how a composition of shaders could be developed for many objects. Anders Langlands wrote an interesting paper named "Physically Based Shader Design in Arnold" [Langlands 2014] where he outlined the idea of stacked BRDF design for

Figure 6: *Environment Light*

shader development. Here I have developed three shaders with the first being a basic diffuse. The second is a dielectric shader which is simply an extension. Lastly there is a layered shader that can take two co-shaders that conform to a specific pipeline specification, including itself. This allows for infinite layering of shaders and uses painted and procedural masks that are sampled according to russian roulette. As a result multiple layers will increase variance but will minimal impact on render time and iterative feedback.

3.1 Layered Shader

The layered shader uses the new pipeline methods which include construct, begin, displacement, prelighting and lighting. Construct is used to initiate values for that shader instance and begin is run every patch. Displacement is the same as in traditional shaders and mutates the surface normal and position. Finally prelighting is used for anything that can be cached and does not rely on camera or light information while lighting is the main method for computing the light contribution. Russian roulette is used to pick a shader based upon the mask and run it's own methods. This could also be another layered shader, repeating the process. Russian roulette is also used as an optimisation to perform early path termination.

The mask is a combination of both an optional texture as well as an fbm noise. This is useful as some layers may need a mask purely controlled with a texture while others may benefit from procedural techniques.

The shader can also change the normal direction based upon the mask as a height-field. This gives the impression of thickness to each layer. Using a method Maas [Maas 2006] describes, I find the difference between the geometric and smoothed normal before calculating the displaced normal. This difference is then added back to the normal to prevent faceting. This faceting is due to the displaced normal being calculated from the geometry and not the original data passed to the shader. This new normal is then passed to the co-shaders.

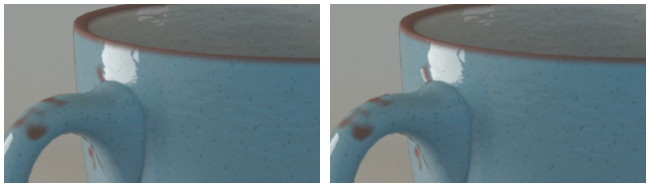

Figure 7: *Layer Bump Mapping*

3.2 Diffuse Shader

The diffuse shader implements an interface that can be integrated into Renderman's physically based workflow. It starts by constructing a more advanced shading context from the stdrsl library. This can then be passed to other functions defined outside the shader. Displacement is rather simple and just receives and passes the normal data directly to the context.

The prelighting method checks to see if a texture has been passed and if so gets the colour values. If required, it will also correct the texture and convert it from sRGB to linear colour space. As the BRDF being used is also from stdrsl it is initiated here with the surface colour. The model used for the BRDF is that described by Oren et al. in "Generalization of Lambert's Reflectance Model" [Oren and Nayar 1994].

Within the main lighting function, the lighting is divided as previously explained into both direct and indirect integrals. The direct lighting call passes in itself as a material which in turn access the BRDF member that was previously initiated. This also handles multiple importance sampling using Veach's power heuristic. Before continuing the path using indirect sampling the sample manager is first queried to gather a sample count from the render context. I added this because it meant the shaders could be used for distributed rendering as well as path tracing. If the the integration mode has been set to path then the sample manager will return one and the path won't branch. The method for indirect lighting is called using a standard cosine distribution for Lambertian response. Then both the direct and indirect lighting contributions are summed to produce the final colour.

3.3 Dielectric Shader

Creating the dielectric shader was simply an extension of the diffuse while adding a dielectric reflector and blending between the two using a Fresnel calculation.

The shader also perturbs the dielectric surface normal using another fbm noise to create subtle changes and imperfections in the specular reflection. A simple noise function is used when getting the diffuse colour to add pigment variation as this was a characteristic I noticed in the reference model. This shader also added a second BRDF

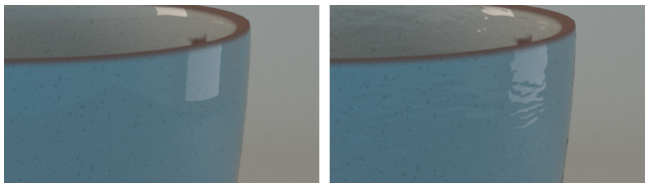

Figure 8: *Dielectric Bump Mapping*

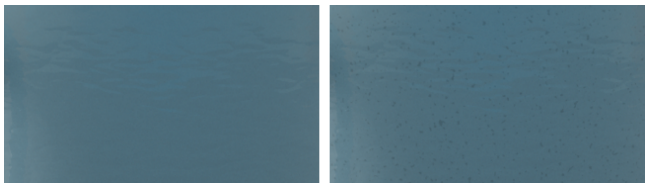

Figure 9: *Pigment Variation*

for the specular component. This needed initiating in the same way as the diffuse component with a possible texture lookup. I used the principled Disney model [McAuley et al. 2012] from stdrsl with a Beckham distribution. This works in the same way as described in the theoretical section but does not include the Fresnel term. It is calculated separately and used when summing both the specular and diffuse contributions. This is because light contacting the surface at greater angles have a higher possibility of refracting and as a result interacting with the diffused pigment below the dielectric surface.

The lighting method now also calls the indirect specular using the specular BRDF. This is passed a pointer to an array of structs that contain radiance information to be shared with the direct lighting call. As previously pointed out, these structs contain lighting information from both the lights and the shader to be used within the rendering engine to perform multiple importance sampling. Then the direct and indirect specular contributions are summed and weighted before being added to the equivalent diffuse contributions. This produces the final colour.

4 Conclusion

To create the finished cup I used the layer shader with a combination of both diffuse and dielectric co-shaders totalling seven shader instances. The four main layers being clay base, ceramic coating, lime scale and then coffee stains can be seen in Figure 10 in a clockwise order starting from the top left. Both textures and procedural noise in the layer was used to mask the layers. Overall I believe the project

Figure 10: *Layers*

was successful in reproducing the the cup in realistic way. With some more development these shaders could also be further generalised to a wider range of material types with greater control. After testing the cup in alternative lighting conditions, the shaders appear to respond in a believable way, integrating into their environment.

References

COOK, R. L., AND TORRANCE, K. E. 1982. A reflectance model for computer graphics. *ACM Trans. Graph. 1*, 1 (Jan.), 7–24.

DUTRE, P., BALA, K., BEKAERT, P., AND SHIRLEY, P. 2006. *Advanced Global Illumination*. AK Peters Ltd.

KAJIYA, J. T. 1986. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '86, 143–150.

LANGLANDS, A. 2014. Physically based shader design in arnold. In *SIGGRAPH 2014 Course: Physically Based Shading in Theory and Practice*.

MAAS, D. 2006. What the rispec never told you. In *ACM SIGGRAPH 2006 Courses*, ACM, New York, NY, USA, SIGGRAPH '06.

MCAULEY, S., HILL, S., HOFFMAN, N., GOTANDA, Y., SMITS, B., BURLEY, B., AND MARTINEZ, A. 2012. Practical physically-based shading in film and game production. In *ACM SIGGRAPH 2012 Courses*, ACM, New York, NY, USA, SIGGRAPH '12, 10:1–10:7.

OREN, M., AND NAYAR, S. K. 1994. Generalization of lambert's reflectance model. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '94, 239–246.

SZCSI, L., SZIRMAY-KALOS, L., AND KELEMEN, C. 2003. Variance reduction for russian-roulette. In *WSCG*.

VEACH, E. 1998. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA. AAI9837162.

WALTER, B., MARSCHNER, S. R., LI, H., AND TORRANCE, K. E. 2007. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, EGSR'07, 195–206.