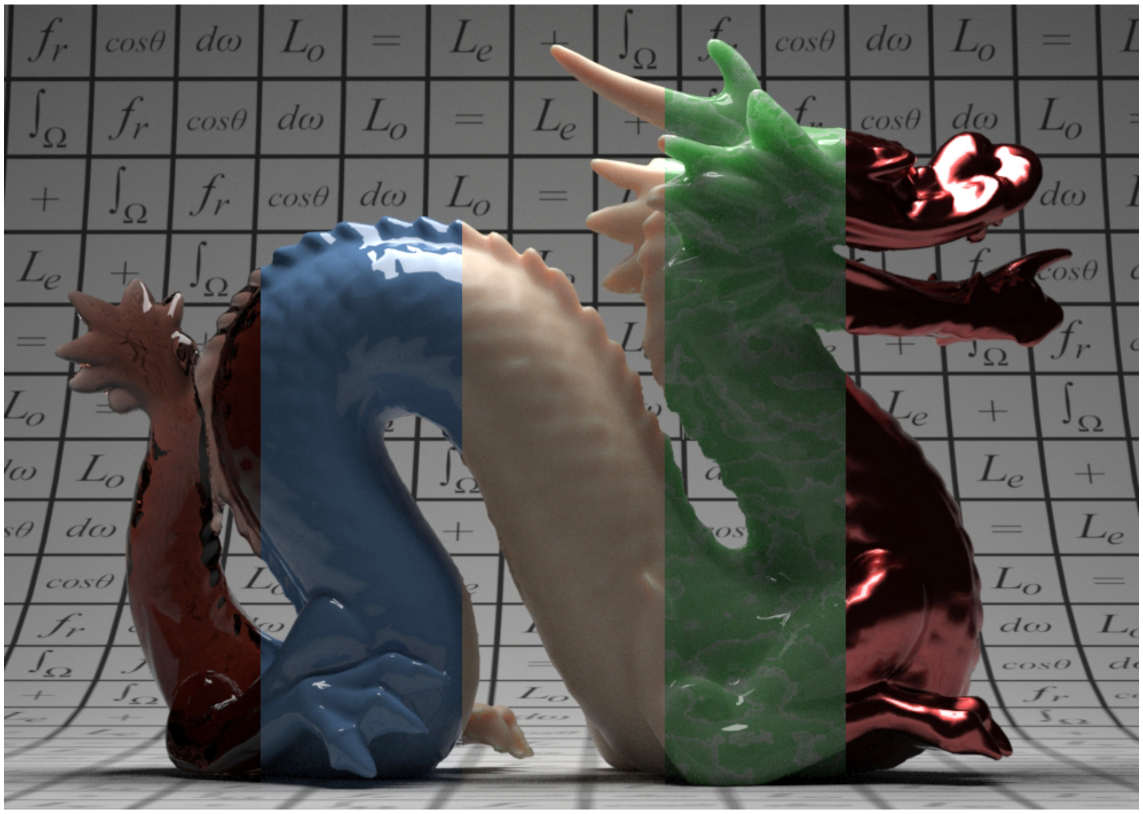
Physically Based Shader Design in Arnold

by Anders Langlands



Introduction

`alShaders` is an open-source, production shader library for Arnold. It was created as a hobby project, both as a learning exercise for the author to get to grips with the Arnold SDK, as well as to fill a gap in the Arnold toolset, since no production-quality library existed that was available or fully functional across all of the DCC applications supported by Arnold. It was made open source in order that others might learn from it and hopefully contribute to its development. Since its inception in 2012, it has seen action in many studios around the world and across a wide variety of work.

In this document, we will examine what constitutes a production shader library and examine the design choices that shaped the form `alShaders` would take. As we will see, many of those choices follow naturally from the design of the renderer itself, so we will also take a brief look at the design of Arnold. We will primarily focus on the design of the surface shader, `alSurface`, examining the way it is structured in order to create a simple-to-use, physically plausible shader. We will cover the outputs it generates and how they are intended to be used within a visual effects pipeline. We will also look at some of the tricks employed to reduce noise and render faster, even if it sometimes means breaking physical correctness.

Finally, we will see what other choices could have been made in its design, along with potential areas for improvement in the future, including potentially fruitful research avenues based on recent work within the graphics community.

Figure 1: Examples of work using `alShaders`. Images used with permission courtesy of (clockwise, from top-left): Nozon, Storm, Brett Sinclair, Phil Amelung & Daniel Hennies, Psyop, Pascal Floerks.

What Makes a Production Shader Library?

A full-featured shader library should be able to correctly shade the vast majority of assets that come through a typical production pipeline. This means that we need surface shaders to handle the most common materials: skin, fibers, metal, plastics, glass etc. Ideally these materials should be handled with as few different shaders as possible, so that users do not have to learn multiple interfaces (or make a judgment call about which shader is best for a particular material), and to reduce the amount of code maintenance required. In `alShaders` all of these materials are handled by `alSurface`, with the notable exception of fibers, for which we provide `alHair`. Multiple surface shaders can be layered together with `alLayer` to create more complex materials.

In order to create realistic materials, users must be able to generate patterns to drive material parameters, either by reading from textures or through procedural noise functions. In most Digital Content Creation (DCC) applications such as Maya, file-based textures are fairly deeply ingrained within the application's workflow, so we rely on translating the application's native functionality. For procedurals we provide several different noise types such as `alFractal` and `alCellNoise`.

These patterns must usually be remapped and blended together in a variety of ways in order to get the desired final appearance, so we provide nodes for basic mathematical operations on both colors and floats (`alCombineColor/Float`) as well as nodes for remapping their values in ways such as input/output range selection, contrast and gamma (`alRemapColor/Float`). Several nodes—in particuar the procedural patterns—include remapping controls directly so as to reduce the amount of nodes required for a given network, for the sake of performance and network legibility.

Finally, more complicated shading setups might require knowledge of the geometry of the scene such as surface curvature (`alCurvature`), normal directions and point positions (`alInputVector`), or selecting or caching subnetworks for efficiency (`alSwitch` and `alCache`, respectively).

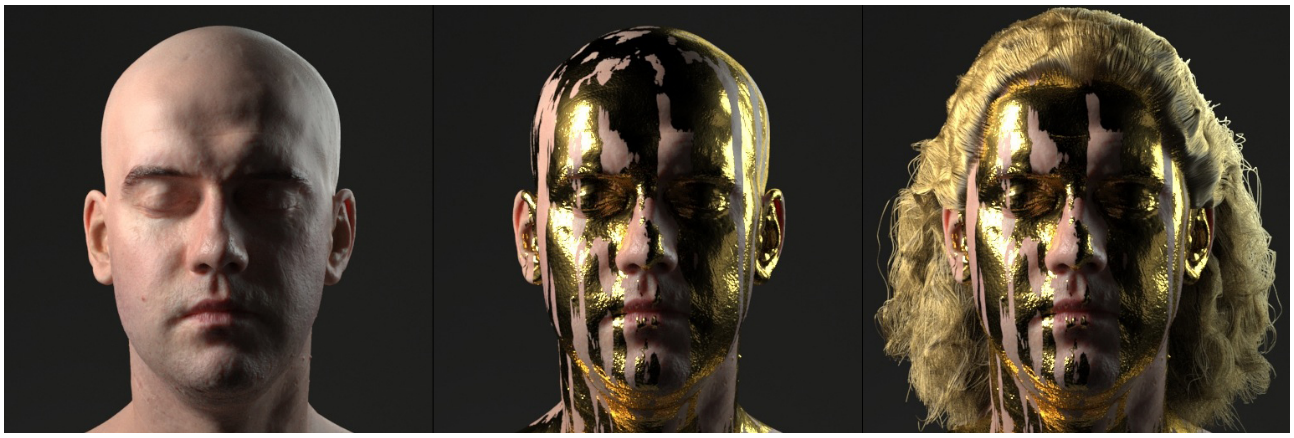


Figure 2: Different materials created using a combination of `alSurface`, `alLayer`, `alHair`, procedural noise, remapping and blending nodes. Head model by Lee Perry-Smith. Hair model by Cem Yuksel.

Design Philosophy

In this section we will examine the philosophy that guided the choices made in designing the shader library. Many of us in the visual effects industry have spent a significant amount of time trying to turn PRMan into a physically based renderer [1], often with limited success. That experience teaches us that it is better to play to a renderer's strengths (and manage its limitations), rather than trying to write a whole new renderer inside a shader.

Arnold

Arnold is an extremely fast, brute-force, unidirectional path tracer. It is targeted primarily at efficient rendering of the massive data sets typical in high-end visual effects production and is therefore tuned to offer the fastest possible ray tracing while keeping the lowest possible memory footprint.

While brute-force Monte Carlo may seem decidedly "old school" compared to the myriad advanced algorithms available to choose from today, it fits perfectly with the guiding principle behind much of Arnold's design, namely that CPU time is cheaper than artist time. While algorithms such as photon mapping and irradiance caching can generate high-quality results in a short time frame, they rely on the user understanding and correctly setting a whole host of extra parameters that can interact in unintuitive ways. Managing these parameters requires extra artist time, which is expensive and could be better spent making a shot look good. Furthermore, setting these parameters incorrectly could make a long render unusable due to flickering or other artifacts.

[1]Interestingly, Pixar have now taken a leaf out of Arnold's book and gone fully raytraced as well.

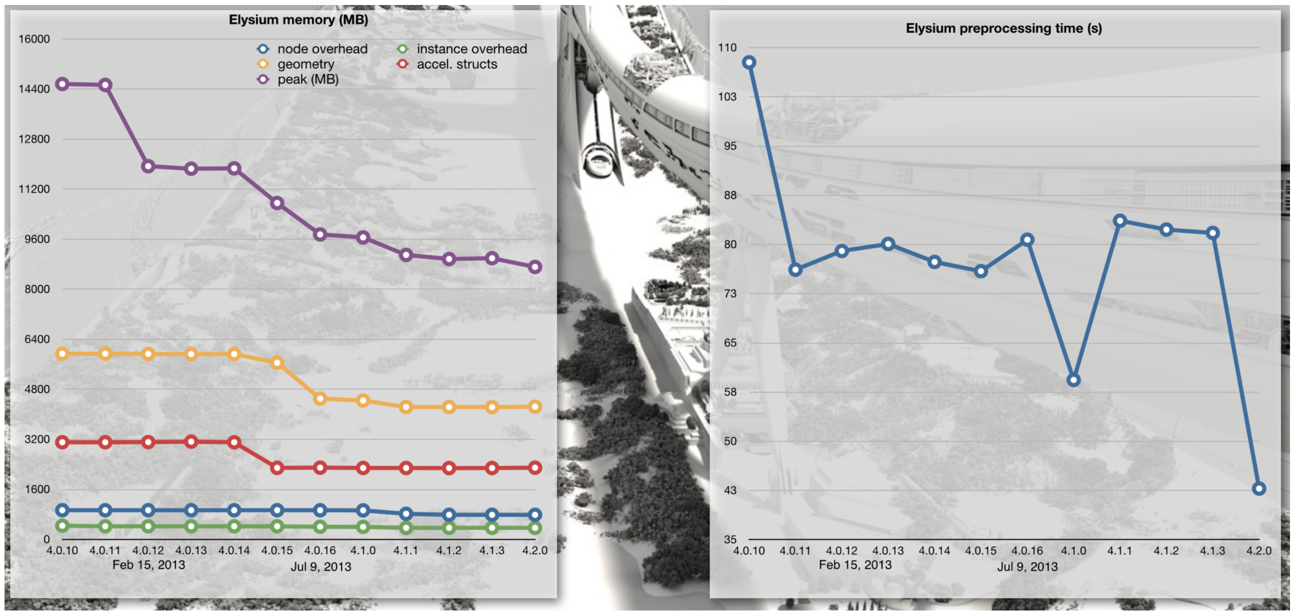
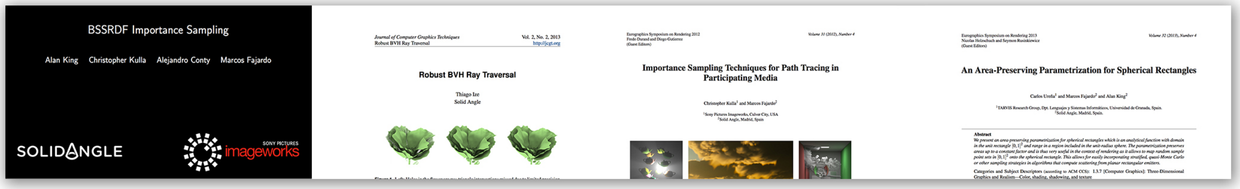


Figure 3: Arnold continually reduces both render time and memory usage with each release.

In contrast, the brute-force approach presents the user with a simple choice: take less samples for a faster, noisier result, or take more samples for a slower, cleaner result. Tuning a scene to get the desired level of noise in the desired amount of time then becomes a matter of setting a small number of parameters that interact with each other in very predictable ways, and can often be set per sequence—or even once for the entire show—and then forgotten about, leaving the lighting artist to concentrate on lighting.

In order to make the renderer fast enough to handle actual production workflows without any of the complex cheats out there, Arnold concentrates on multiple importance sampling (MIS) and highly efficient sample distribution in order to reduce the number of samples that need to be used for a clean result from any given scene.

The API itself is C and fairly low level, giving shader writers the choice of either relying on the built-in integration and MIS functions or write their own. Rays are traced strictly in order and Arnold provides a message-passing API to allow communication between different vertices of a single path.


BSSRDF Importance Sampling

Alan King Christopher Kulla Alejandro Conty Marcos Fajardo

SOLIDANGLE SONY PICTURES imageworks

Journal of Computer Graphics Techniques Vol. 2, No. 2, 2013
Robust BVH Ray Traversal http://jcgt.org

Robust BVH Ray Traversal

Thiago Ize
Solid Angle

Eurographics Symposium on Rendering 2012 Volume 31 (2012), Number 4
Fredo Durand and Diego Gutierrez
(Guest Editors)

Importance Sampling Techniques for Path Tracing in Participating Media

Christopher Kulla[1] and Marcos Fajardo[2]
[1]Sony Pictures Imageworks, Culver City, USA
[2]Solid Angle, Madrid, Spain

Eurographics Symposium on Rendering 2013 Volume 32 (2013), Number 4
Nicolas Holzschuch and Szymon Rusinkiewicz
(Guest Editors)

An Area-Preserving Parametrization for Spherical Rectangles

Carlos Ureña[1] and Marcos Fajardo[2] and Alan King[2]
[1]TARVIS Research Group, Dpt. Lenguajes y Sistemas Informáticos, Universidad de Granada, Spain.
[2]Solid Angle, Madrid, Spain.

Abstract
We present an area-preserving parametrization for spherical rectangles which is an analytical function with domain in the unit rectangle $[0,1]^2$ and range in a region included in the unit-radius sphere. The parametrization preserves areas up to a constant factor and is thus very useful in the context of rendering as it allows to map random sample point sets in $[0,1]^2$ onto the spherical rectangle. This allows for easily incorporating stratified, quasi-Monte Carlo or other sampling strategies in algorithms that compute scattering from planar rectangular emitters.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Figure 4: Examples of Solid Angle's published research on sampling techniques.

The Design of `alShaders`

These characteristics lead us naturally to a set of guiding design principles for the development of the library, ensuring that we use Arnold in the most effective way possible.

Don't Try to Turn Arnold Into Something It's Not: Understand the strengths of the renderer—

raytracing speed, sampling efficiency, etc.—and use those features to solve the rendering equation. In particular there should be no interpolation or precalculation of illumination.

Simplicity: Choose algorithms that use as few parameters as possible, or reparameterize to make the user interface simple.

Predictability: User parameters should drive changes in the shading result in a predictable way. If the user inputs a color value (say, with a texture map), then the shader result should be as close to that color as possible without introducing unexpected color shifts due to the algorithms used. Scalar parameters should be remapped wherever possible to produce a linear response to input.

Efficiency: Use multiple importance sampling for everything, and in particular use Arnold's built-in sampling routines wherever possible to allow the renderer to manage sample distribution efficiently. The user should not have to worry about edge cases. The renderer should perform reasonably well in every case and the user should not have to remember dozens of rules, such as: *if it's this type of scene, I have to enable this parameter.*

Scalability: We are aiming for production-quality rendering, so motion blur is a necessity. Consequently, We should use algorithms and techniques that work efficiently at high AA sample counts.

Plausibility: Finally, we take it for granted that all algorithms should be physically plausible. Moreover, Simplicity should apply here so that plausible materials are made as simple as possible. The user should not have to worry about whether they've achieved the look they want "in the right way". In other words, if it *looks* right, it *is* right.

The Layered Uber-Shader Model vs. The BSDF-Stack Model

The first question to answer when designing a new shader is: do you want it to be an uber shader or a stack shader? In recent times, BSDF-stack models have become more fashionable, while uber-shader models have fallen out of favour. In this section we will examine the pros and cons of each model as they relate to our stated design philosophy.

The BSDF-Stack Model

The BSDF stack is normally implemented as a series of BSDF shading nodes connected to a Stack shader that is responsible for handling the light and sampling loops and accumulating the results from each connected BSDF node.
This model offers a number of attractive features, namely:

Flexibilty: The design is extremely flexible, as the user can combine whatever BSDFs they like in order to create their materials.

Simple Interface: Only the required controls are exposed. If the user desires a material with only a single specular lobe, for example, they do not have to wade through a sea of unused parameters to get to the ones they are interested in.

Rapid Prototyping: It is easy to try out new BSDF models. If one wishes to deploy a new specular BRDF for testing or to solve a specific problem, all that is required is to create a new BSDF node to implement it. Existing shader networks can remain unchanged while the new BRDF is evaluated.

However, this extra flexbility comes at a cost:

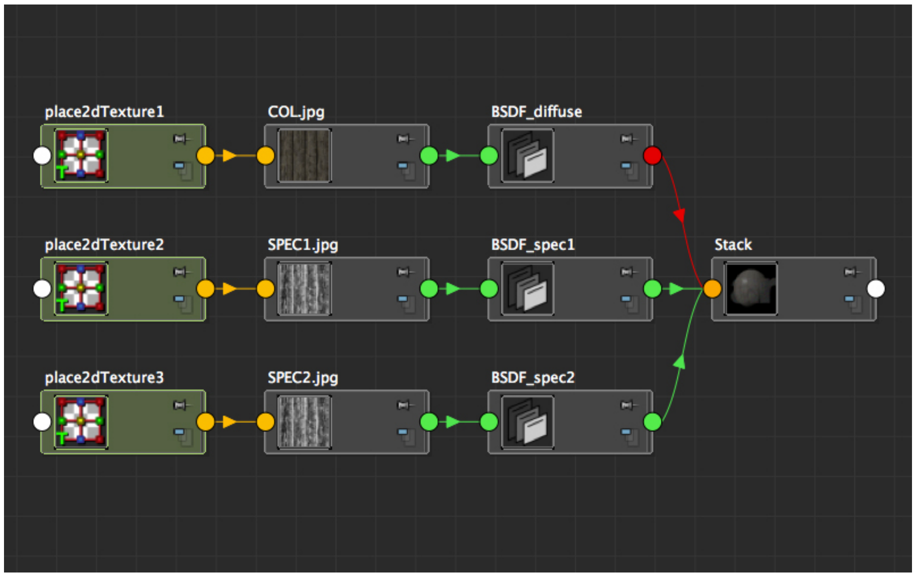


Figure 5: A BSDF-stack node graph representation

High Maintenance: BSDF stacks require a significant overhead to set up and maintain. There is a lot of clicking and connecting required just to create a simple plastic shader. This is often mitigated by creating a shader template system to quickly create basic material types, but then that system itself requires maintaining and keeping up to date with new shader versions - for example, if one wishes to switch specular BRDF globally for an improved model, all shader templates must now be updated.

Hard to Tame: BSDF stacks are harder to optimize and constrain for energy conservation. Since BSDF nodes operate conceptually in isolation, it requires more work on the programmer's part to optimize their operation and ensure energy conservation, either by adding lots of logic to the Stack, or by having the BSDF nodes communicate with each other, which breaks encapsulation.

Easy to Break: "Anything is possible" is not always a good thing; it is extremely easy to create completely nonsensical material models, or incredibly inefficient material stacks (consider 10 specular layers each with a weight of 0.1, for example). Either the programmer must detect and try to handle such cases, or the artists using the system must be trained and monitored not to break things. Essentially, one is trading extra overhead keeping an eye on what artists are doing for flexbility that is required in a tiny percentage of real-world cases.

The Layered Uber-Shader Model

The layered uber-shader model removes user choice in what layers are available and instead provides a single "chunk" of material. This is most often combined with a separate layer shader to allow users to blend multiple materials together to create more complex effects. All light and sampling loops and their accumulation are explicitly handled by the uber shader.

The advantages of this model are:

Easy to Maintain: It is quicker to set up and easier to maintain compared to the stack model. Creating a basic material is as simple as creating the shader and changing a couple of parameters. Shading networks are smaller and easier to understand.

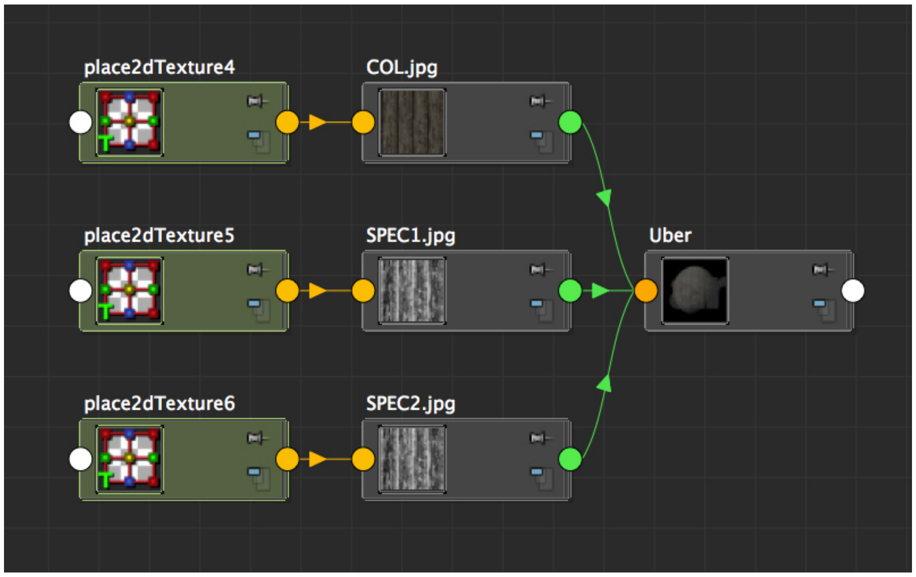


Figure 6: An uber-shader node graph representation

High Performance: It is easier to optimize for speed. Since everything in the light and sampling loops is completely under the main shader's control, results can be reused and BSDF layers balanced against one another without the need for complex control logic or message passing.

Robust: If implemented correctly, it should be nearly impossible for a user to create a non-physical or non-sensical material. This means less time debugging scenes and allows artists to experiment freely without fear of breaking anything.

But it does have some limitations:

Unfiltered Interface: Whether the artist is using them or not, all parameters are visible all the time, potentially making it harder to find the ones they want.

Dependencies: It is harder to implement new techniques. Much greater care must be taken when adding a new algorithm to ensure that existing scenes don't change look or degrade performance. Interface-breaking or look-changing rollouts must be managed carefully. On the plus side, it is easier to do like-for-like comparisons between old and new techniques simply by switching shader binaries.

Uncompromising: Sometimes you really do need four speculars! We are targeting the real world of production, where physical correctness occasionally has to be thrown out the window to satisfy a client. Fortunately, the cases where there is no other solution than to break physical plausibility are extremely rare.

Given our stated design goals of simplicity, predictability and efficiency, it is obvious that the uber-shader model is a better fit for our needs. In other renderers with different aims, or even in Arnold for a different, more specific workflow (if one were trying to build a system for researching new algorithms, for example), a BSDF stack might be a better choice.

The `alSurface` Layered Model

We will do a bit of hand waving here and simply state that in our experience, roughly 90% of materials used in production can be modeled well with a diffuse or transmissive base plus one specular lobe, and nearly everything else by adding an additional specular lobe. With that in mind, Figure 7 shows how these components are layered together in `alSurface`.

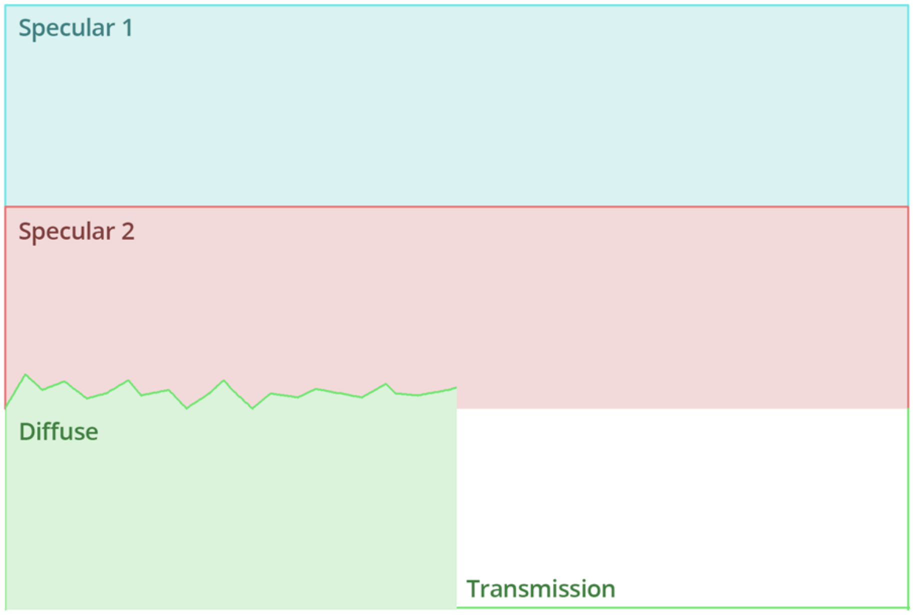


Figure 7: The layer model used by `alSurface`.

As shown in the diagram, we have either a semi-infinite diffuse (subsurface scattering) or transmissive base, covered by one or two (glossy) specular layers.

In the current model, the layers themselves have no thickness; they are simply an infinitely thin scattering interface that either reflects light, or transmits it to the layer immediately below it. Balancing different combinations of these layers can produce a huge variety of different effects.

In terms of parameters (see Figure 9), each layer has a `Strength` and a `Color` parameter, which are just scalar and vector multipliers on the layer's result, respectively. Each layer also has an `Advanced` section, which contains more low-level, non-look-based parameters for fine-tuning the shader's behaviour and performance.

Diffuse

The diffuse base layer uses either an Oren-Nayar BRDF [ON94] with an optional backfacing lobe for doing thin-surface translucency effects, or a multi-lobe, cubic BSSRDF. The user can linearly interpolate between the BRDF and BSSRDF with the `Mix` parameter as an additional control on how "scattery" they want the material to appear. The two are mixed linearly to ensure evergy conservation, and the BSSRDF is normalized so that the final albedo can be user controlled (or mapped) by the `Diffuse Color` parameter. We found this preferable to trying to directly invert the scattering parameters from the color map, since it guarantees the expected result in the simplest case and the user can map the scattering parameters separately to achieve more complex effects.

The BSSRDF provided by the Arnold API is a multi-lobe kernel using either cubic or Gaussian profiles. While Gaussians are more common in recent literature, we chose to use cubic profiles to match the result from Arnold's default standard shader, which users would already be familiar with. We use

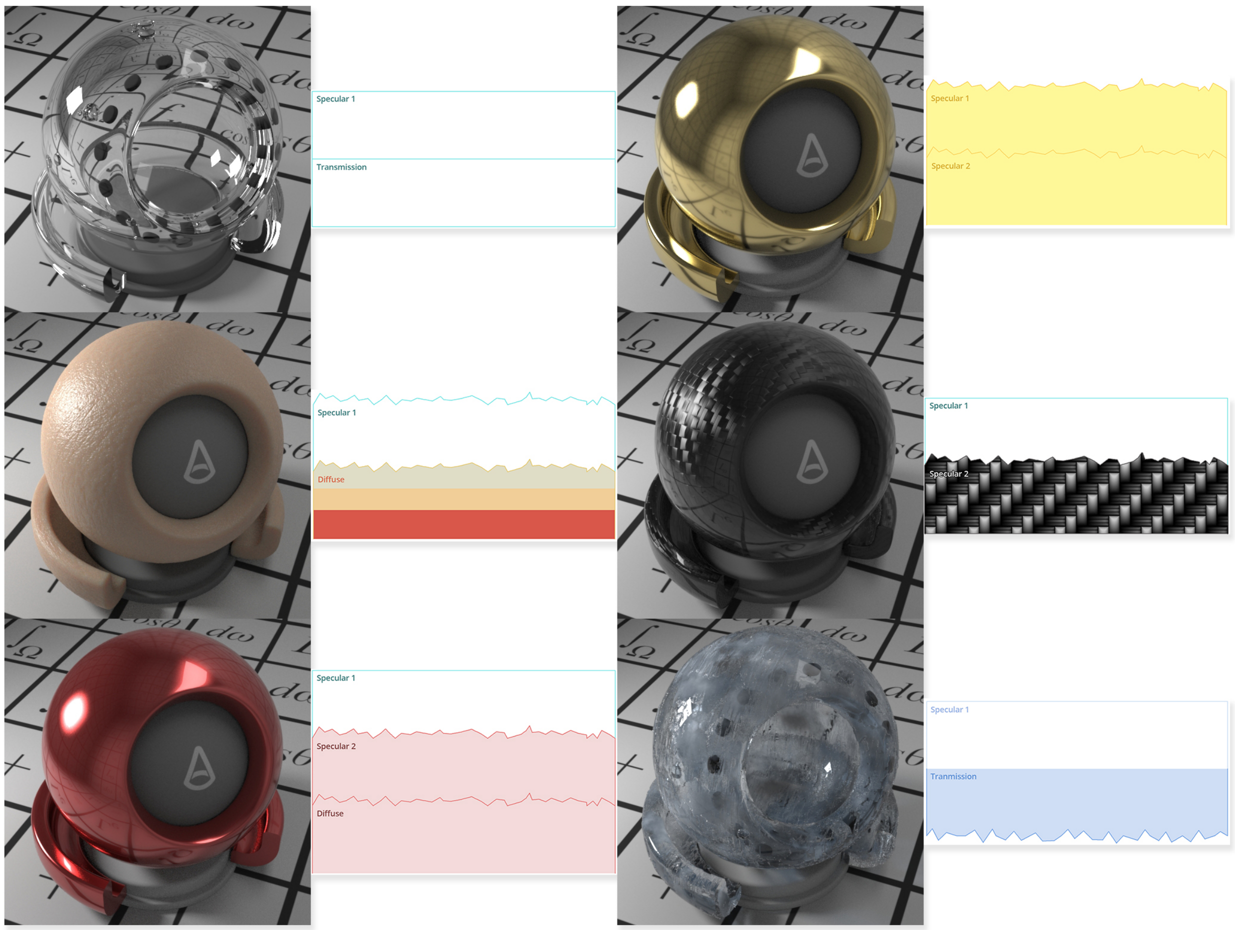
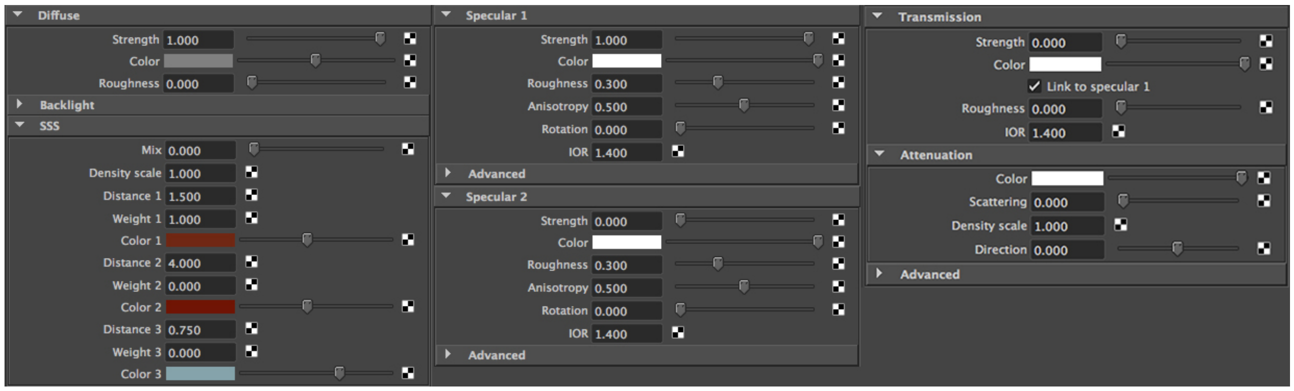


Figure 8: Some example material types and their corresponding layer structures.



Diffuse

Strength	1.000
Color	▨
Roughness	0.000

Backlight

SSS

Mix	0.000
Density scale	1.000
Distance 1	1.500
Weight 1	1.000
Color 1	🟥
Distance 2	4.000
Weight 2	0.000
Color 2	🟥
Distance 3	0.750
Weight 3	0.000
Color 3	▨

Specular 1

Strength	1.000
Color	⬜
Roughness	0.300
Anisotropy	0.500
Rotation	0.000
IOR	1.400

Advanced

Specular 2

Strength	0.000
Color	⬜
Roughness	0.300
Anisotropy	0.500
Rotation	0.000
IOR	1.400

Advanced

Transmission

Strength	0.000
Color	⬜
✓ Link to specular 1	
Roughness	0.000
IOR	1.400

Attenuation

Color	⬜
Scattering	0.000
Density scale	1.000
Direction	0.000

Advanced

Figure 9: alSurface parameters as displayed in Maya's Attribute Editor.

three lobes (although more are possible) for simplicity's sake. Arnold internally samples all the profiles using a very fast, fully raytraced, MIS algorithm [Kin+13]. The shape of each profile is controlled by a `Distance` and `Color` parameter, which together control the radius of the cubic, and the layers are blended with `Weight` parameters which are normalized before being used. `Density scale` just adds a multiplier on the radii in order to easily adapt an existing look to a scene with a different unit scale.

Specular

The specular layers use the Cook-Torrance BRDF provided by the Arnold API, which is an anisotropic Beckmann distribution sampled using a modified version of the Heitz and D'Eon [HD14] visible normal sampling technique. The `Roughness` parameter is squared before being used in order to give a more linear perceptual response (as in Burley [Bur12]). `Anisotropy` controls the stretchiness of the highlight: values less than 0.5 stretch in the U direction, and values greater than 0.5 stretch in the V direction. The frame can be rotated using the `Rotation` parameter.

Transmission

Transmission also uses a Beckmann microfacet distribution. By default, the `Link to Specular 1` parameter is enabled. This causes the transmission `Roughness` and `IOR` parameters to be linked to the top specular layer, which is the most common configuration for dielectrics such as glass. Breaking this link can be useful for modeling materials such as ice and glass with one frosted side.

All transmission rays can be subject to `Attenuation`, either simply through absorption (Beer's Law) or including single scattering in order to simulate liquids and other scattering transmissive materials. Users select the desired `Color` for the attenuation as well as the amount of `Scattering` (as opposed to just absorption) and the appropriate attenuation coefficients are calculated for them internally. Alternatively, users can use the controls in the `Advanced` section to specify attenuation coefficients directly if they wish to use measured data, for example.

Energy Conservation

The energy conservation model in alSurface is extremely simple, which makes it robust and easy to understand (Figure 10).

Each layer is modeled as a microfacet scattering interface, where light can either be reflected or transmitted (or absorbed), according to the Fresnel equations.

The shader is evaluated top down, proceeding to each layer in turn. Any energy that is not reflected at each interface is assumed to be transmitted, and is available to be reflected at the next interface. Note that since we assume the layers to be infinitely thin, we do not account for absorption within the layers themselves, nor do we model the light travelling back out of the surface. Modelling absorption and scattering within and between thin layers à la Weidlich and Wilkie [WW07] or Jakob et al. [Jak+14] could be investigated in the future.

Fresnel

We use the standard Fresnel function for all layers, specified by a user-defined index of refraction (IOR) value. In the case of dielectrics this is all that is necessary, and a reasonable approximation to reflectance for conductors can be achieved by specifying a high IOR (e.g. 10-1000) and selecting a specular color multiplier that matches the desired material. This is "good enough" for most cases (see Figure 11, and has the advantage that if the surface is textured, the final color of the material can be taken directly from that texture. It does not however capture the subtle color shift seen at glancing angles in many metals, and if this effect is required, we allow the user to specify measured reflectance data by connecting an `alFresnelConductor` node to the layer's `IOR` parameter. The angle-dependent Rec.709 reflectances for a selection of common metals are precalculated and stored in a lookup table in the shader, for fast access. The result can also be normalized in order to apply it to a textured specular color.

It is extremely important when calculating Fresnel to do so for the angle between the incident light and the normal of each microfacet (i.e., the half-angle vector), not just once based on the view

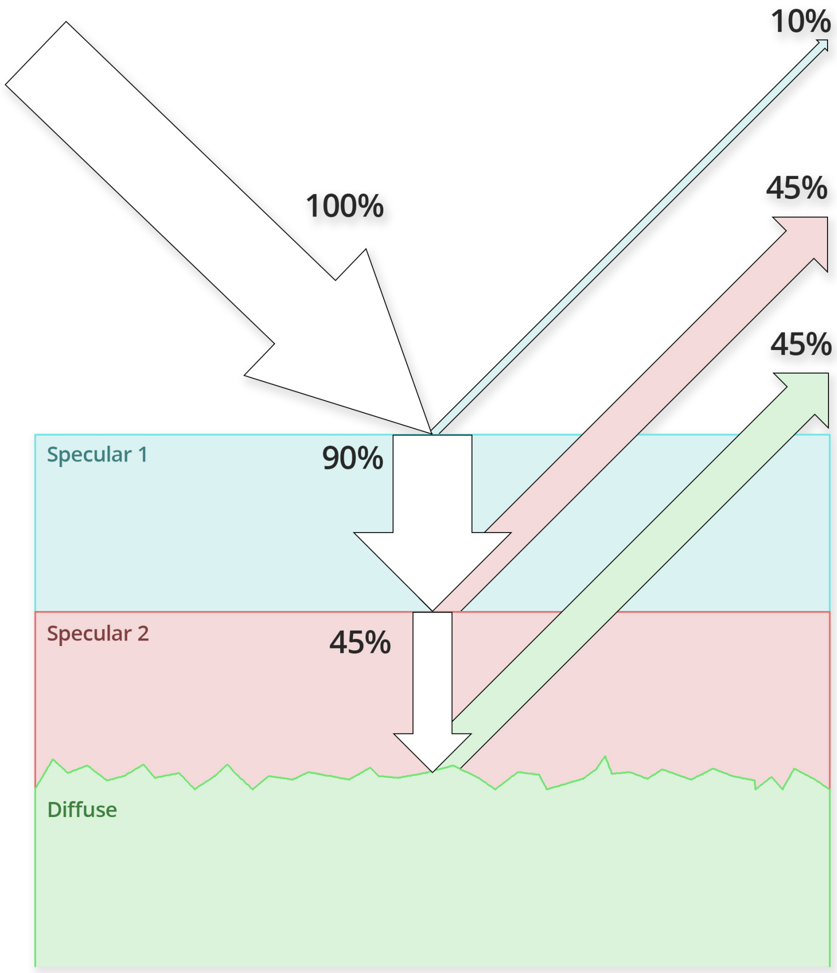


Figure 10: Interface-based energy conservation. Incident white light is attenuated by fresnel transmittance only, and reflected according to the parameters of each layer. If all layers are pure white, the shader will reflect 100% of the energy it receives.

direction. This ensures that the Fresnel falloff behaves correctly as the microsurface roughness changes (see Fig. 12).

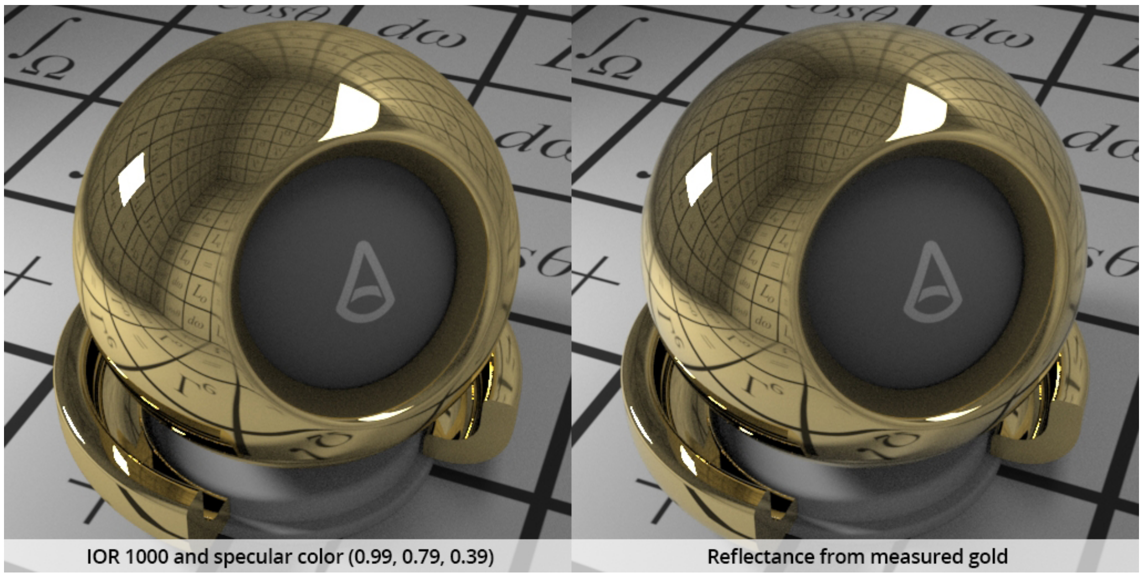

IOR 1000 and specular color (0.99, 0.79, 0.39)
Reflectance from measured gold

Figure 11: Using a high (real) IOR with an appropriate specular color gives a reasonable match to measured data, but lacks the color shifts visible at glancing angles.

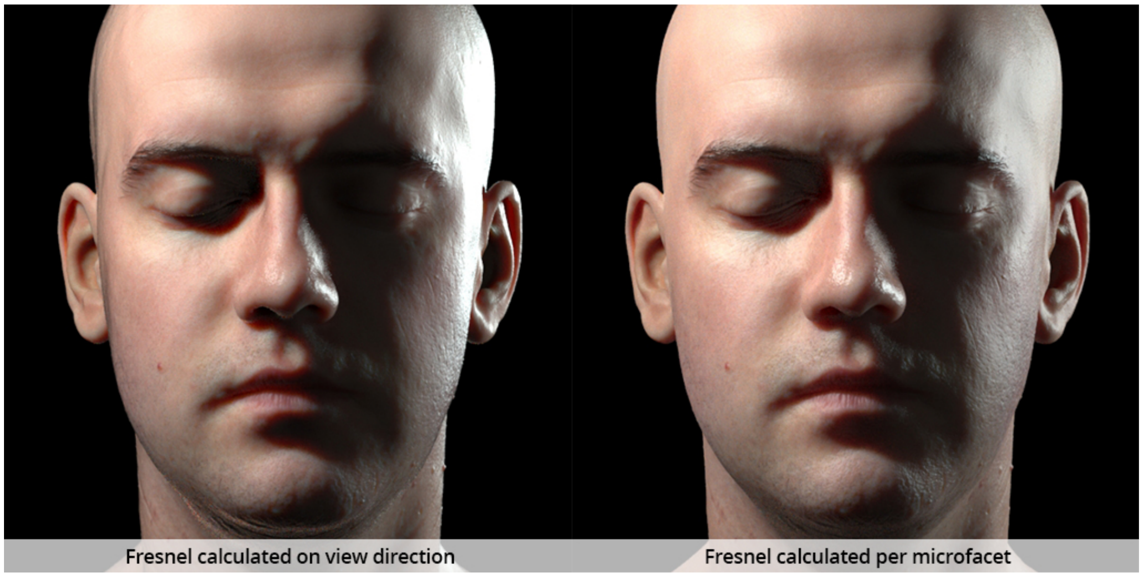

Fresnel calculated on view direction
Fresnel calculated per microfacet

Figure 12: Not calculating Fresnel per microfacet leads to artifically dull highlights at normal incidence and artifically bright highlights or dark egdes at glancing angles when used with a rough BSDF.

Efficiency Optimizations

Avoiding Caustic Paths

Caustics are generally defined as LSDE (Light Specular Diffuse Eye) paths, i.e., a specular bounce followed by a diffuse bounce before reaching the eye. These paths are notoriously difficult to solve in a unidirectional path tracer, so we explicitly disallow them in the shader by checking if the path we are evaluating is diffuse, and if so turning off specular evaluations.

However, we can also generate caustic paths from glossy bounces if the receiving surface has a higher roughness than the surface it is tracing against (Figure 13).

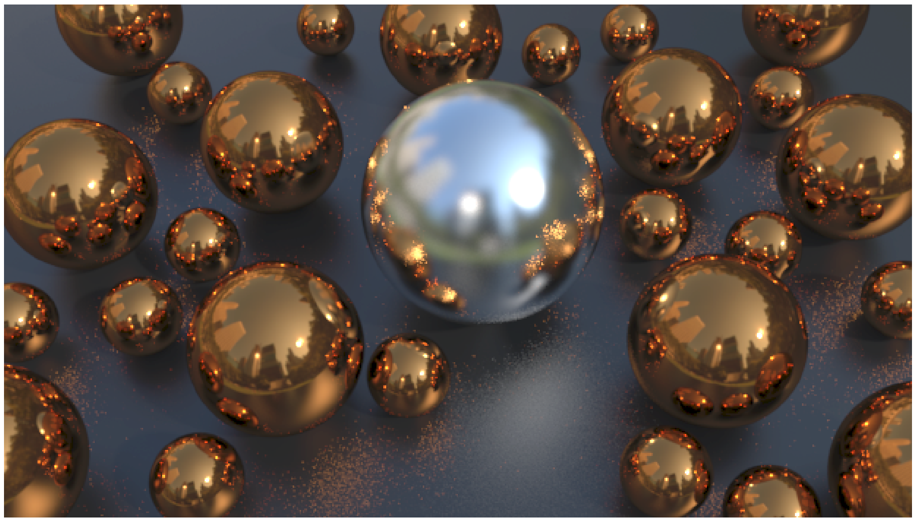


Figure 13: The floor is rougher than the silver sphere, which is rougher than the orange spheres. This results in “fireflies”.

We cannot simply disallow these paths as that would kill many glossy self-reflections entirely! Instead we combat this by passing the current specular roughness down the ray tree. At each bounce, the roughness of the current specular BRDF is clamped to be the same or larger than that at the previous bounce. This causes slight changes in the shape of reflected highlights, but the difference is rarely noticeable in practice and clears up the noise effectively (Figure 14).

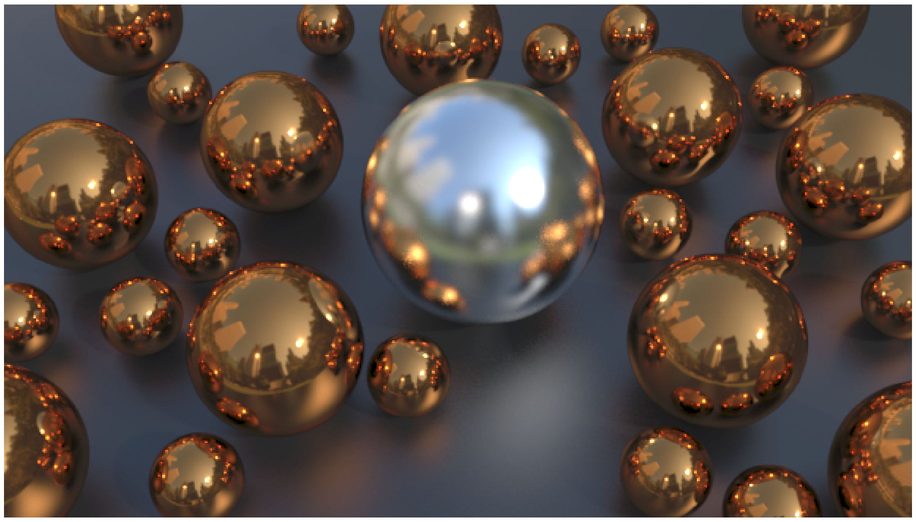


Figure 14: We can resolve the reflections properly by clamping the minimum roughness down the ray tree, with only a small change in look.

Russian Roulette

`alShaders` targets production rendering, which means rendering with motion blur. It is usually necessary to render with high camera sample counts (e.g. 64-144 samples per pixel) in order to resolve motion trails cleanly. Thus it is important to avoid doing redundant work as much as possible; the noise in many shading effects will clear up at these sample rates, so it is possible to undersample them to improve efficiency [Vea98].

The first candidate for Russian roulette sampling is splitting. Arnold will only split at the first bounce—at high sample rates most users will turn off splitting completely—but splitting still occurs because each BSDF in the shader is sampled individually. This isn't usually a problem for opaque surfaces, but for transmissive surfaces such as glass, where many bounces of both reflection and transmission may be needed to render correctly, this can lead to an explosion in render time.

To combat this, rather than evaluate both reflection and transmission rays at each intersection, we choose one or the other based on the Fresnel reflection coefficient. This dramatically reduces render time for the same number of camera samples, and (at realistic sample rates) no additional noise is discernable, as shown in Figure 15.

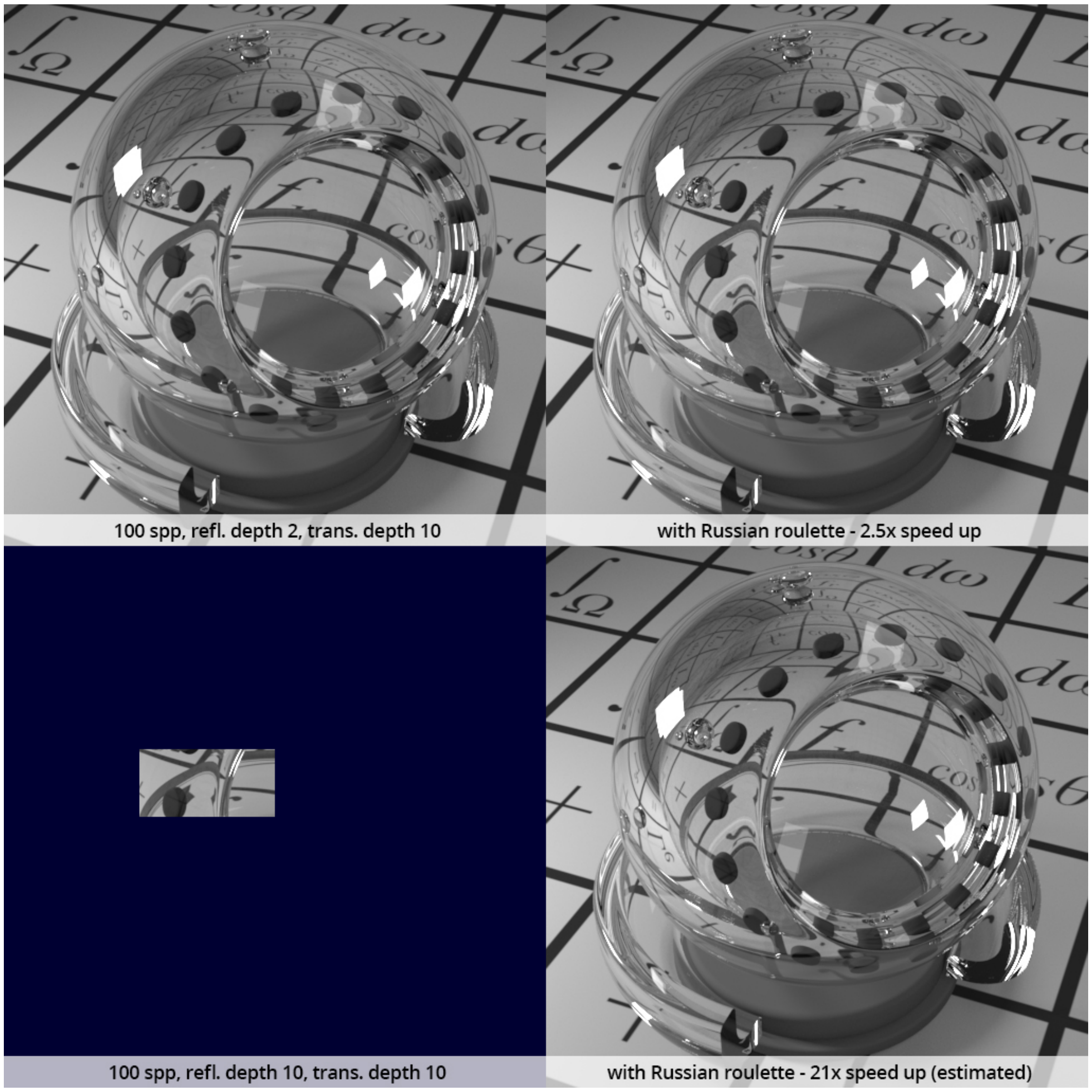

100 spp, refl. depth 2, trans. depth 10
with Russian roulette - 2.5x speed up
100 spp, refl. depth 10, trans. depth 10
with Russian roulette - 21x speed up (estimated)

Figure 15: Using Russian roulette to probabilistically choose between specular reflection and transmission dramatically cuts the rendering time. In the case where both reflection and transmission depths were set to 10, the render without Russian roulette took too long to measure.

The second candidate for Russian roulette is path termination. We track the throughput of the current path using message passing and probabilistically kill the path if it is unlikely that continuing will contribute much to the image. We use a modified version of the spectral optimization technique of Szécsi, Szirmay-Kalos, and Kelemen [SSK03] where our continuation probability p_c is:

$$p_c = \min\left(1, \sqrt{\frac{\text{hmax}(T_i \cdot w_i)}{\text{hmax}(T_i)}}\right) \tag{1}$$

Here, we use the horizontal component maximum hmax as the weighting function, T_i is the path throughput up to the current path vertex and w_i is the sample weight at the current vertex. We take the square root in order to change how quickly path termination "kicks in", which we found gives a nice speedup without introducing a significant amount of extra noise. If we could guarantee that extremely high sample rates were going to used, then using a linear mapping could improve efficiency, but we found it better to avoid surprising users with extremely noisy images at low sample rates.

Early path termination gives us a nice speedup for an equivalent sample rate, while only introducing a small amount of extra noise (Figure 16), and allows us to trace many more bounces at very little cost, since only paths that actually contribute to the image are continued (Figure 17).

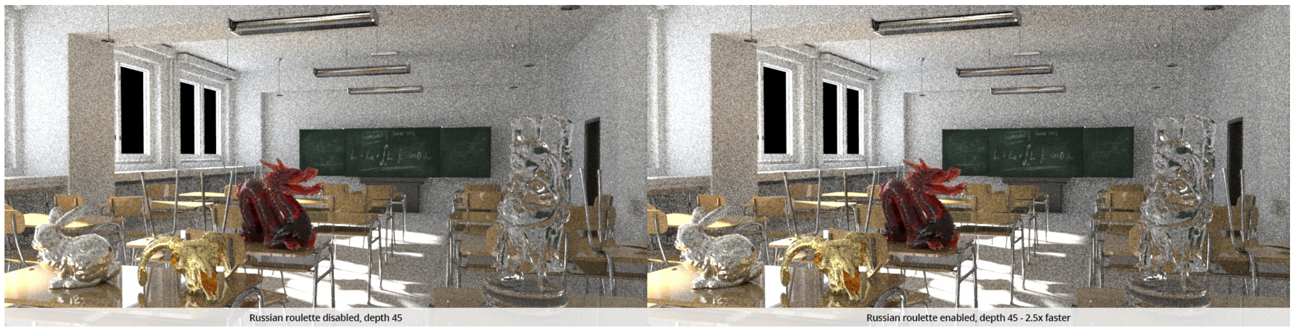

Russian roulette disabled, depth 45
Russian roulette enabled, depth 45 - 2.5x faster

Figure 16: Using Russian roulette for early path termination can significantly reduce render times, with a small increase in variance.

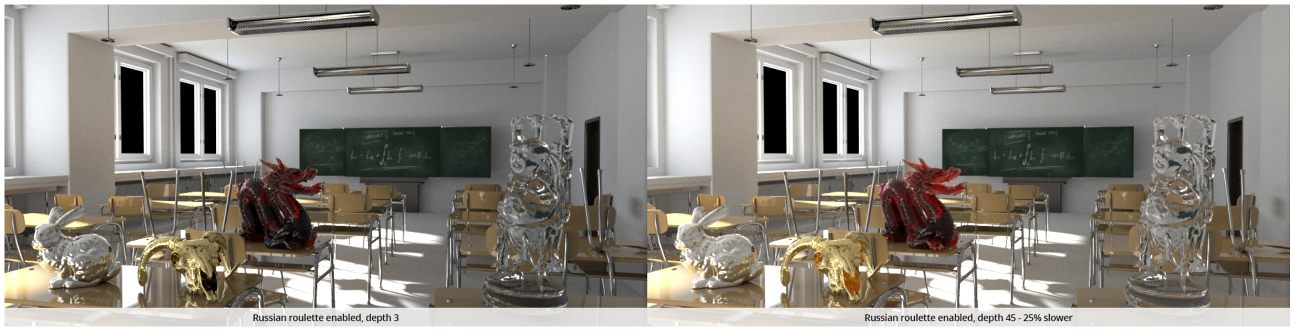

Russian roulette enabled, depth 3
Russian roulette enabled, depth 45 - 25% slower

Figure 17: Early path termination also allows arbitrarily high numbers of bounces to be used at little additional cost.

Sample Clamping

Although increases in fine-grained noise from Russian roulette are acceptable[2], fireflies can sometimes be introduced when a path is continued with extremely low probability. To combat this, we provide a control to optionally clamp the final sample weight to some reasonable maximum (e.g., 10-20). This can

[2]The noise can double as film grain and responds well to increased sample counts and simple denoising algorithms.

slightly darken the resulting image and clip highlights in certain circumstances, but it does effectively eliminate fireflies from difficult-to-sample paths (Figure 18).

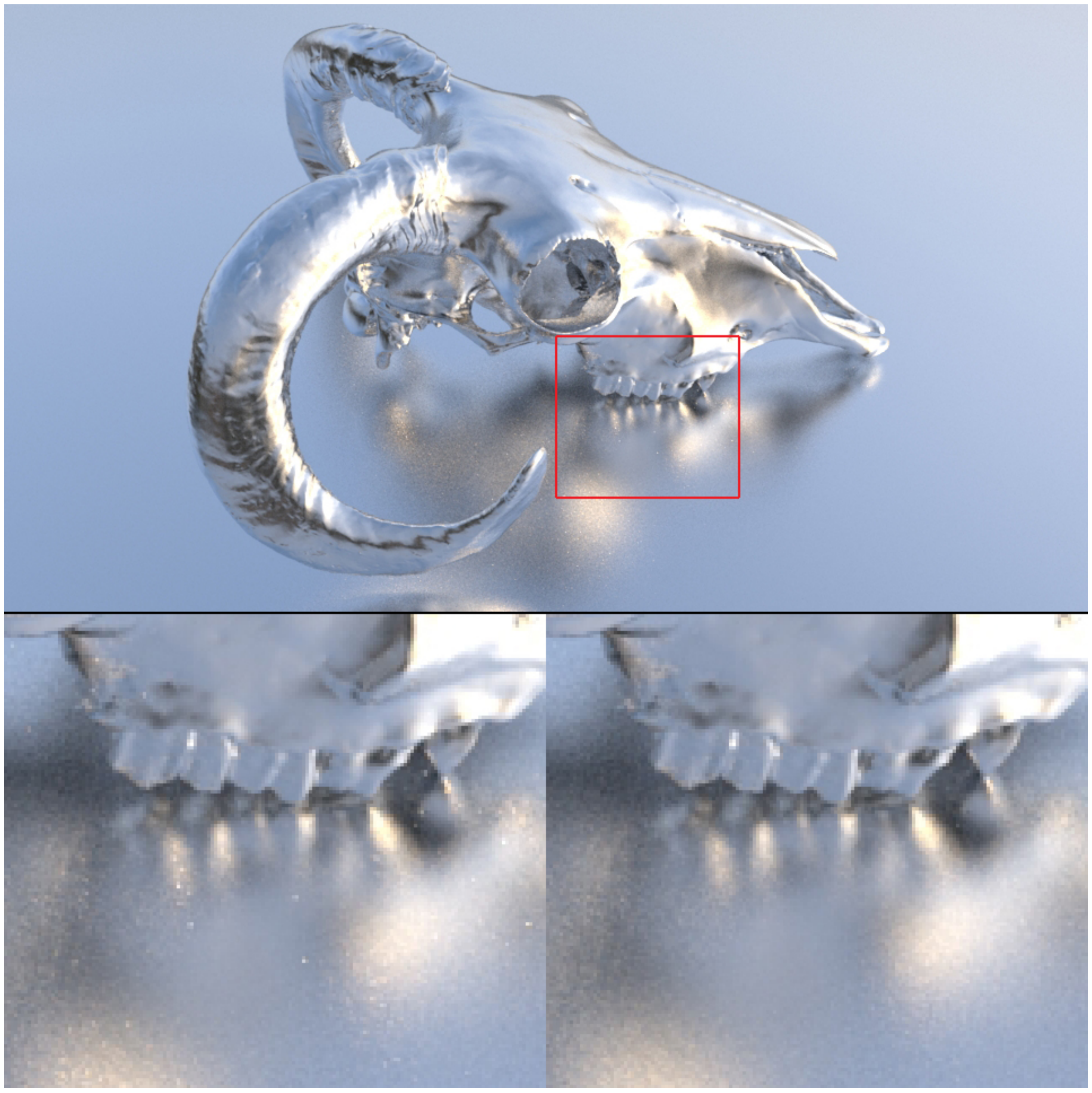


Figure 18: A render of a glossy metallic skull showing fireflies (top). Clamping indirect specular paths to a maximum of 10 events eliminates the bright pixels (bottom right) and in many cases does not otherwise affect the image.

Arbitrary Output Variables

Arbitrary Output Variables (AOVs) are an essential part of any production pipeline. Deciding what outputs to support can often be difficult as there are a bewildering array of workflows to choose from, and twice as many opinions on the best approach. Again, we focus on simplicity, and split our AOVs into four categories:

SHADING: these separate each shading layer into direct and indirect components (direct diffuse, indirect specular etc.).

LIGHT: up to 8 AOVs separating light sources (or groups of light sources) into individual outputs.

ID: up to 8 color AOVs that can be used for RGB mattes for compositing, or for plugging in arbitrary channels (e.g., noise patterns).

DATA: UVs, depth, facing-ratio passes etc.

Light Groups

While the shading outputs are mainly included for legacy reasons, we prefer light groups for rebalancing renders in "comp" (compositing). Rebalancing shading passes tends to break physical plausibility: we have gone to great lengths to ensure reflectance between different rough surface layers is carefully balanced using Fresnel, and these effects cannot easily be tweaked in 2D for a global illumination render without breaking realism.

In contrast, restricting balancing to light groups allows tweaking a render in comp while maintaining physical accuracy. In order to achieve this, lights are assigned to one of eight light groups via a user-defined integer parameter. We then track the contribution of each light group to the current traced path (multiplying by each BSDF as we go) and output its contribution in a separate AOV. The result is equivalent to performing separate renders for each light source individually, but dramatically less expensive and easier to set up. The resulting light group AOVs can then be added together to recreate the beauty render, or they can be multiplied by the compositing artist to change the color and brightness of each light group without needing to re-render the scene. Figure 19 shows an example render and its light groups, and Figure 20 some examples of physically correct scene rebalancing in a compositing package using light groups.

Conclusions and Future Work

`alShaders` provides a full-featured production shader library. The surface shader, `alSurface`, can realistically represent most commonly needed material types, and does so using a simple, easy-to-understand energy conservation model. It is efficient, making use of Russian roulette sampling techniques to reduce shading cost in high-sample renders, and provides a flexible set of AOVs that can be correctly rebalanced in comp. Being open source, it is easy for others to read and experiment with the code, and fork it for modification in their own pipelines.

`alShaders` is also a work in progress. The direction of future development of the surface shader will encompass a much more rigorous sampling and Russian roulette scheme for the different layers of the shader, rather than the ad hoc model proposed here. We are also keen to research an effective method of modeling the multiple scattering of light between microfacets to reclaim the energy lost of traditional single-scattering BSDFs.

Physically based rendering is a dynamic and exciting field and there are always new techniques to try. In particular, we are currently working on implementing the energy conserving, importance

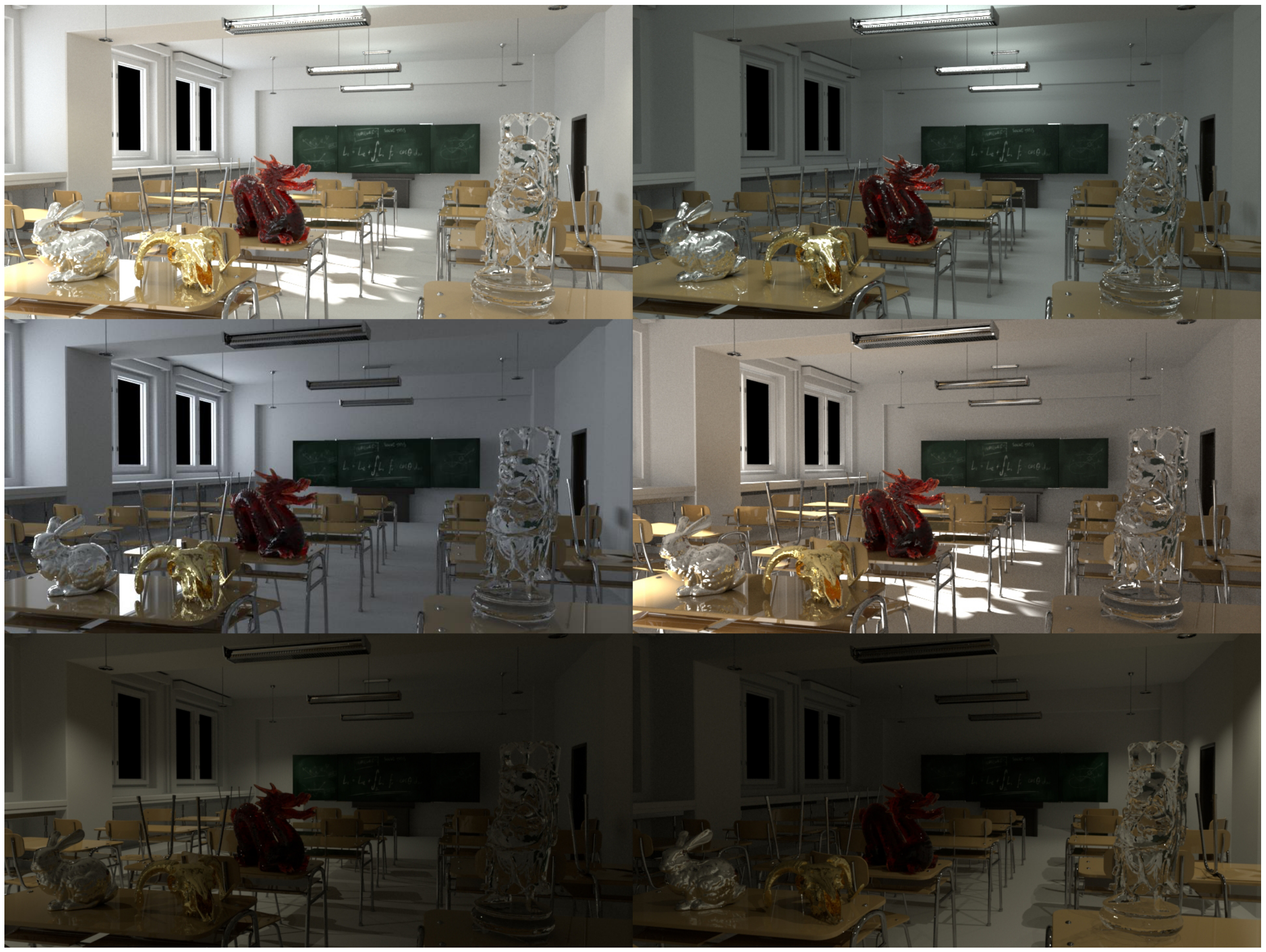


Figure 19: A beauty render (top left) and its associated light group AOVs. The sun and sky light are output individually while the practical light sources are grouped for easier control.

sampled hair model by d'Eon, Marschner, and Hanika [dMH13] as an efficient replacement for the "Franken"-Marschner model combination of Ou et al. [Ou+12] and Zinke et al. [Zin+08] currently used by `alHair`. The directional dipole by Frisvad, Hachisuka, and Kjeldsen [FHK13], is a simple alternative to traditional subsurface scattering kernels that gives improved results and is a good candidate for improved skin rendering. LEADR mapping by Dupuy et al. [Dup+13] is an excellent solution for preserving the look of surface microstructure at different screen sizes (a common problem in production), can be importance sampled efficently, and also opens an avenue to building custom microfacet distributions for user-defined microstructures, such as the warp and weft of cloth. Yan et al. [Yan+14] cover a similar area of research, but capture high-frequency changes in speculars that cannot be accounted for with current models, and could be a useful addition for cases such as metallic flakes in car paint, so as long as it can be implemented without overcomplicating the shader.

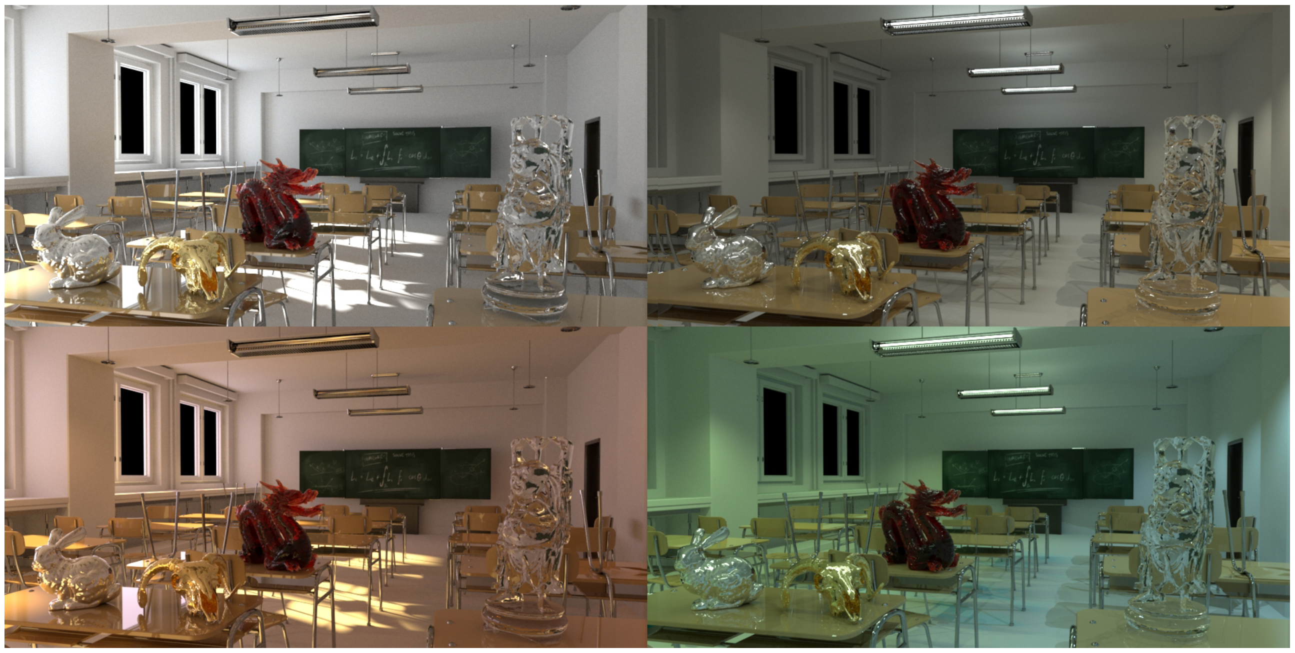


Figure 20: An example of the results of rebalancing the lights in the classroom scene in a compositing application to create different effects.

Acknowledgements

Marcos Fajardo and the rest of the Solid Angle team both for writing Arnold in the first place and for the guidance and support offered during the development of this library.

Espen Nordahl and Magnus Bruland at Storm for their contributions (`alCache`, `alSwitch`, `alTriplanar`, `alJitterColor`) and feedback on the library.

Jules Stevenson's Kettle shaders [Ste] were, to our knowledge, the first example of open source Arnold shader development and a great learning resource.

PBRT [PH10], by Matt Pharr and Greg Humphreys provides a never-ending resource for theory, as well as code snippets for theiving.

Bibliography

[Bur12] Brent Burley. “Physically-based Shading at Disney”. In: 2012. URL: `http://disney-animation.s3.amazonaws.com/library/s2012_pbs_disney_brdf_notes_v2.pdf`.

[dMH13] Eugene d’Eon, Steve Marschner, and Johannes Hanika. “Importance Sampling for Physically-based Hair Fiber Models”. In: *SIGGRAPH Asia 2013 Technical Briefs*. SA ’13. Hong Kong, Hong Kong: ACM, 2013, 25:1–25:4. ISBN: 978-1-4503-2629-2. DOI: `10.1145/2542355.2542386`. URL: `http://doi.acm.org/10.1145/2542355.2542386`.

[Dup+13] Jonathan Dupuy et al. “Linear Efficient Antialiased Displacement and Reflectance Mapping”. In: *ACM Transactions on Graphics* 32.6 (Nov. 2013), Article No. 211. DOI: `10.1145/2508363.2508422`. URL: `http://hal.inria.fr/hal-00858220`.

[FHK13] J. R. Frisvad, T. Hachisuka, and T. K. Kjeldsen. *Directional Dipole for Subsurface Scattering in Translucent Materials*. Manuscript. Aug. 2013. URL: `http://www2.imm.dtu.dk/pubdb/p.php?6646`.

[HD14] Eric Heitz and Eugene D’Eon. “Importance Sampling Microfacet-Based BSDFs using the Distribution of Visible Normals”. Anglais. In: *Computer Graphics Forum* (June 2014). URL: `http://hal.inria.fr/hal-00996995`.

[Jak+14] Wenzel Jakob et al. “A Comprehensive Framework for Rendering Layered Materials”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)* 33.4 (2014). URL: `http://www.cs.cornell.edu/projects/layered-sg14/layered.pdf`.

[Kin+13] Alan King et al. “BSSRDF importance sampling.” In: *SIGGRAPH Talks*. ACM, 2013, p. 48. ISBN: 978-1-4503-2344-4. URL: `https://www.solidangle.com/research/s2013_bssrdf_slides.pdf`.

[ON94] Michael Oren and Shree K. Nayar. “Generalization of Lambert’s Reflectance Model”. In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’94. New York, NY, USA: ACM, 1994, pp. 239–246. ISBN: 0-89791-667-0. DOI: `10.1145/192161.192213`. URL: `http://doi.acm.org/10.1145/192161.192213`.

[Ou+12] Jiawei Ou et al. “ISHair: Importance Sampling for Hair Scattering”. In: *ACM SIGGRAPH 2012 Talks*. SIGGRAPH ’12. Los Angeles, California: ACM, 2012, 28:1–28:1. ISBN: 978-1-4503-1683-5. DOI: `10.1145/2343045.2343084`. URL: `http://doi.acm.org/10.1145/2343045.2343084`.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN: 0123750792, 9780123750792.

[SSK03] László Szécsi, László Szirmay-Kalos, and Csaba Kelemen. “Variance reduction for Russian-roulette”. In: *A. Slack, Solid State Physics* 34 (2003), p. 2003. URL: `http://sirkan.iit.bme.hu/~szirmay/c29.pdf`.

[Ste] Jules Stevenson. *Kettle Shaders*. URL: https://bitbucket.org/Kettle/kettle_uber/wiki/Home.

[Vea98] Eric Veach. "Robust Monte Carlo Methods for Light Transport Simulation". AAI9837162. PhD thesis. Stanford, CA, USA, 1998. ISBN: 0-591-90780-1.

[WW07] Andrea Weidlich and Alexander Wilkie. "Arbitrarily Layered Micro-Facet Surfaces". In: *GRAPHITE 2007*. Perth, Australia: ACM, Dec. 2007, pp. 171–178. ISBN: 978-1-59593-912-8. URL: http://www.cg.tuwien.ac.at/research/publications/2007/weidlich_2007_almfs/.

[Yan+14] Ling-Qi Yan et al. "Rendering Glints on High-resolution Normal-mapped Specular Surfaces". In: *ACM Trans. Graph.* 33.4 (July 2014), 116:1–116:9. ISSN: 0730-0301. DOI: 10.1145/2601097.2601155. URL: http://doi.acm.org/10.1145/2601097.2601155.

[Zin+08] Arno Zinke et al. "Dual Scattering Approximation for Fast Multiple Scattering in Hair". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)* 27.3 (2008), 32:1–32:10. DOI: 10.1145/1360612.1360631. URL: http://doi.acm.org/10.1145/1360612.1360631.