`layerlab`: A computational toolbox for layered materials

by Wenzel Jakob

1 Introduction

A layered BSDF model describes the directional reflectance properties of a material whose internal structure consists of a stack of scattering and/or absorbing layers separated by smooth or rough interfaces. The bottom of the stack could be an opaque interface (such as a metal) or a transparent one. Such structural decompositions into layers and interfaces dramatically enlarge the size of the "language" that is available to describe materials, and for this reason they have been the focus of considerable interest in computer graphics in the last few years [WW07; WW11; Jak+14a].

In this document, we present `layerlab`, a Python-based toolbox for computations involving layered materials that implements a model recently proposed by Jakob et al. [Jak+14a]. The purpose of this document is to serve both as a gentle introduction to the underlying theory and as a hands-on tutorial on solving practical layering problems with this tool.

To follow the tutorial, download and compile the toolbox from GitHub:

```
$ git clone --recursive https://github.com/wjakob/layerlab
$ cd layerlab && cmake . && make -j 4
```

Next, launch Python and import the `layerlab` package.

```
>>> import layerlab as ll
```

We will also be using NumPy and matplotlib for visualizations.

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

The remainder is structured as follows: Section 2 motivates rendering of layered materials and discusses the key components. Section 3 discusses background material from linear transport theory. Section 4 introduces different "ingredient" layers that can be combined to build layered materials and Section 5 shows how an initial 1D solution can be extended to produce a full BSDF model. Finally, Section 6 covers practical considerations and showcases several experiments performed with `layerlab`.

2 Motivation

The most commonly used reflectance models in computer graphics describe the appearance of objects in terms of metal or dielectric interfaces with microscopic roughness. Real objects are often considerably more complex, with layers that might not fully hide the material below: glaze over ceramic, wall paint over primer, colored car paint with a clear coat, enamel on gold and silver jewelry, and layered biological structures such as leaves or skin.

Simulating these types of *layered* materials in renderings is surprisingly difficult: when a quantum of light enters the top layer, it can undergo a complex sequence of scattering events within individual layers and at layer boundaries; finally, the light is either absorbed or able to leave the material. The details of this intermediate scattering process are important, since they determine both the intensity and distribution of scattered light.

Most state-of-the-art rendering systems provide support for layered materials in some form, but the underlying simulations either entirely neglect internal scattering or rely on crude approximations.

Accurate models have been proposed for specific layer combinations, but it is important to realize that we do not have accurate computational models for scattering from any systems more complex than a single layer or a single interface. Even the simplest nontrivial system, of a single medium bounded by a smooth interface, is only roughly approximated by standard BRDF models. While it is possible to perform an accurate simulation of the random walk of light quanta individually using a brute force Monte Carlo simulation (e.g. in a path tracer), this is rarely done in practice because it tends to produce prohibitively high variance in renderings.

Recently, Jakob et al. [Jak+14a] proposed a model that supports general layered materials without the aforementioned drawbacks. In this method, the response of the layer to incident illumination is precomputed ahead of time and tabulated into a compact representation that can be queried and importance sampled at render time. This involves several major components:

- A sparse angular representation for isotropic BRDFs
- Techniques for projecting a range of standard models into this representation
- An efficient way of merging the representation of two layers into a combined form that accounts for all orders of internal scattering
- A correction to recover energy lost to multiple scattering in traditional microfacet models

We will go over each of these points and present experiments that demonstrate the capabilities of this approach.

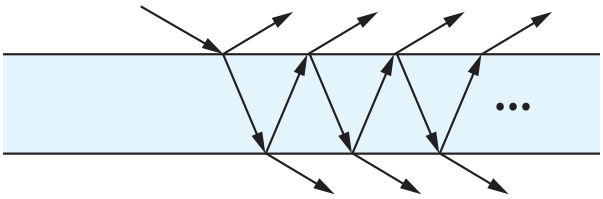
3 Background

Accounting for internal reflections in a stack of layers entails computing an integral over all possible trajectories along which light could flow through the material; the proposed model builds on existing tools from the radiative transport literature to solve this problem. In this section, we provide a brief overview of relevant background material, starting with a historical perspective.

3.1 Glass plates theory

Some of the earliest theoretical work on layered materials was conducted by Stokes [Sto60], who analyzed the combined reflection and transmission properties of a stack of glass plates. It will be instructive to review the mathematics underlying the simplest case of his analysis involving only a single plate.

Assuming geometric optics, unpolarized light and no absorption, the top interface of a glass plate illuminated by a ray of unit power reflects a portion R of the incident light and transmits another portion T into the material, where it goes on to encounter the bottom interface, reflecting back and forth with a fraction escaping at each event:



In this case, R and T are given by the Fresnel equations. Due to reciprocity, the reflection and transmission coefficients at the bottom interface are also equal to R and T.

Suppose that we want to determine the total reflectance and transmittance of the plate as a whole. To obtain these properties, we must sum over all light paths that eventually leave the plate, tracking the intermediate scattering events to compute the path's contribution to the sum.

$$\begin{aligned} R_{\text{tot}} &= R + TRT + \ \ldots & &= R + T^2 \sum_{i=0}^{\infty} R^{2i+1} & &= R + \frac{RT^2}{1-R^2}, \\ T_{\text{tot}} &= TT + TR^2T + \ \ldots & &= T^2 \sum_{i=0}^{\infty} R^{2i} & &= \frac{T^2}{1-R^2}. \end{aligned} \tag{1}$$

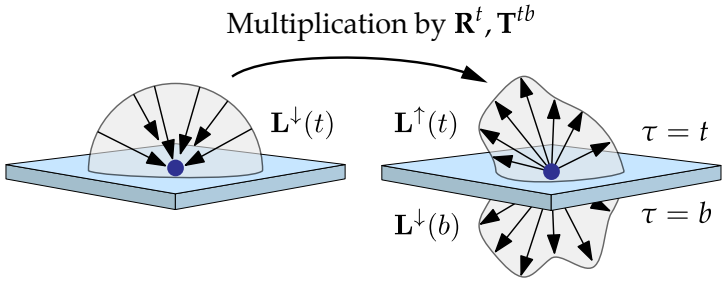
The resulting expressions are geometric series, which have a surprisingly simple explicit form. This is convenient, since it allows us to directly quantify the optical properties of the combination of both interfaces without having to sum over large numbers of internal scattering events.

3.2 Combining general layers

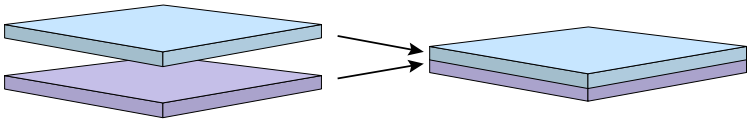
With this example in mind, let us move to a more general case. Rather than a smooth interface illuminated from a single direction, consider a slab of arbitrary composition illuminated by a radiance distribution expressed in some basis. (We leave the underlying discretization unspecified for now.) The linearity of light transport then allows us to write the scattered illumination projected onto the same basis using a matrix-vector product:

$$\begin{aligned} \mathbf{L}^{\uparrow}(t) &= \mathbf{R}^{t}\mathbf{L}^{\downarrow}(t) + \mathbf{T}^{bt}\mathbf{L}^{\uparrow}(b), \\ \mathbf{L}^{\downarrow}(b) &= \mathbf{R}^{b}\mathbf{L}^{\uparrow}(b) + \mathbf{T}^{tb}\mathbf{L}^{\downarrow}(t), \end{aligned} \tag{2}$$

where $\mathbf{L}^{\uparrow}(\tau)$ and $\mathbf{L}^{\downarrow}(\tau)$ are vectors describing the upwards and downwards radiance at depth τ with respect to the basis. The depths t and b correspond to the top and bottom surface, and the square matrices $\mathbf{R}^t, \mathbf{R}^b$ and $\mathbf{T}^{tb}, \mathbf{T}^{bt}$ describe the reflection and transmission for light arriving at the top and bottom, respectively:



The analogous question to the glass plate example is: given the *scattering matrices* of two distinct layers (i.e. $\mathbf{R}_1^t, \mathbf{R}_2^t, \mathbf{T}_1^{tb}, \mathbf{T}_2^{tb}$, etc.), what are the scattering matrices of the two layers stacked together?



The solution is similar, but with the $\mathbf{R}$ and $\mathbf{T}$ matrices replacing the scalars R and T. As in the scalar case, we sum over all possible sequences of reflections and transmissions, replacing geometric series by their closed-form solutions. Attention must be paid to the ordering of multiplications, since the

matrices generally do not commute. For a rigorous discussion of this derivation, we refer the reader to [GH69]. The final result of this computation, analogous to (1), are the so-called *adding equations*:

$$\begin{aligned}
\tilde{\mathbf{R}}^t &= \mathbf{R}_1^t + \mathbf{T}_1^{bt}(\mathbf{I} - \mathbf{R}_2^t\mathbf{R}_1^b)^{-1}\mathbf{R}_2^t\mathbf{T}_1^{tb} \\
\tilde{\mathbf{R}}^b &= \mathbf{R}_2^b + \mathbf{T}_2^{tb}(\mathbf{I} - \mathbf{R}_1^b\mathbf{R}_2^t)^{-1}\mathbf{R}_1^b\mathbf{T}_2^{bt} \\
\tilde{\mathbf{T}}^{tb} &= \mathbf{T}_2^{tb}(\mathbf{I} - \mathbf{R}_1^b\mathbf{R}_2^t)^{-1}\mathbf{T}_1^{tb} \\
\tilde{\mathbf{T}}^{bt} &= \mathbf{T}_1^{bt}(\mathbf{I} - \mathbf{R}_2^t\mathbf{R}_1^b)^{-1}\mathbf{T}_2^{bt}
\end{aligned} \tag{3}$$

Comparing Equations (1) and (3) reveals many similarities. For instance, the scalar inverse was replaced by a matrix inverse, the constant one was replaced by an identity matrix, and so on.

The adding equations are a key ingredient of `layerlab`, since they permit accurate computation of the scattering properties of stacks of layers that take all orders of internal scattering into account. Any layer that is expressed in a suitable basis can be combined with other layers, and the same approach even works for boundaries between layers that describe index of refraction changes.

To apply the adding equations in practice, we require a specific basis to tabulate the response of layers to incident illumination. Furthermore, we must be able to project the reflectance functions of layers and layer boundaries onto this basis. We discuss both aspects in turn.

3.3 Discretization

The directional response of a surface to illumination is characterized by the local illumination integral

$$L_o(\theta,\phi) = L_e(\theta,\phi) + \int_0^{2\pi}\int_0^{\pi} f(\theta,\phi,\theta',\phi')\, L_i(\theta',\phi')\, |\cos\theta'| \sin\theta' \,\mathrm{d}\theta'\,\mathrm{d}\phi', \tag{4}$$

which relates the outgoing, emitted, and incident radiance (L_o, L_e, and L_i, respectively) via the BSDF f. In the context of a layer located between depths $\tau \in [t, b]$, angles $\theta < \frac{\pi}{2}$ correspond to illumination on the top surface t, and angles $\theta > \frac{\pi}{2}$ cover illumination on the bottom surface b.

In this section, we show how to convert Equation (4) into a set of matrices that are usable with the adding equations (3). We will only consider non-emitting surfaces, hence $L_e = 0$. To facilitate the initial discussion, we will make a strong simplification and completely ignore the problem's dependence on azimuth angles ϕ. The simplified equation only involves a 1-dimensional integral over latitudes θ and can be understood as describing light transport in 2D flatland.

$$L_o(\theta) = \int_0^{\pi} f(\theta,\theta')\, L_i(\theta')\ |\cos\theta'| \sin\theta' \,\mathrm{d}\theta' \tag{5}$$

Instead of working with the latitude angles, it is generally more convenient and natural to integrate over cosines of latitudes, which avoid the spatial distortion of spherical coordinates near the poles (this removes the $\sin\theta'$ term).

$$L_o(\mu) = \int_{-1}^{1} f(\mu,\mu')\, L_i(\mu')\, |\mu'| \,\mathrm{d}\mu' \tag{6}$$

3.3.1 Computing integrals

To perform computer simulations, this equation must now be discretized in some manner. The discretization we'll employ turns the integral into a weighted sum that evaluates all terms at discrete locations $\{\mu_1, \ldots, \mu_n\} \in [-1, 1]$. The expression then reads

$$L_o(\mu) = \sum_{j=1}^{n} f(\mu,\mu_j)\, L_j(\mu_j)\, w_j\, |\mu_j|. \tag{7}$$

The way in which this discretization is done can have a fairly large impact on the accuracy of integrals, particularly when the integrand is smooth. Let's consider an artificial example where we want to integrate a high-order polynomial over the interval $[-1, 1]$.

```
>>> def integrand(x):
        return 6.3 * (x**2 - x**6 - x**8)
```

Placement of 100 uniform and equal-weight samples leads to a poor approximation with almost 15% relative error (the correct answer is `1.0`).

```
>>> np.sum(integrand(np.linspace(-1, 1, 100)) * 1/50)
0.85891040658351925
```

Discretized integrals in linear transport theory frequently rely on *Gaussian quadrature*, where the locations μ_i (referred to as *discrete ordinates* [Cha60]) and weights w_i are chosen to yield an exact result for polynomials up to a certain degree. Gauss-Lobatto points are one example, which include the endpoints ± 1 and maximize the order of exactly integrable polynomials subject to this constraint.

```
>>> mu, w = ll.quad.gaussLobatto(6)
>>> mu, w
(array([-1.        , -0.76505532, -0.28523152,  0.28523152,  0.76505532,  1.        ]),
 array([ 0.06666667,  0.37847496,  0.55485838,  0.55485838,  0.37847496,  0.06666667]))
```

For the previous example, they can be seen to provide the correct answer up to rounding errors with just 6 evaluation points.

```
>>> np.sum(integrand(mu) * w)
0.99999999999999989
```

Note that most commands in `layerlab` include extensive documentation—to obtain additional information on their usage simply invoke `help()` at any point during the tutorial. The namespace `ll.quad`, for instance, contains several additional quadrature techniques. Enter the following command to see an overview:

```
>>> help(ll.quad)
```

Once the set of ordinates and weights are chosen, we can proceed to convert the local surface integral (7) into a form that is suitable for use with the adding equations. For this, it will be convenient to introduce some extra notation:

$$\begin{aligned} \mathbf{L}_i &\coloneqq (L_i(\mu_k))_k \in \mathbb{R}^n & \mathbf{F} &\coloneqq (f(\mu_i,\mu_j))_{ij} \in \mathbb{R}^{n\times n} \\ \mathbf{L}_o &\coloneqq (L_o(\mu_k))_k \in \mathbb{R}^n & \mathbf{W} &\coloneqq (\delta_{ij} w_i)_{ij} \in \mathbb{R}^{n\times n} \\ & & \overline{\mathbf{M}} &\coloneqq (\delta_{ij}|\mu_i|)_{ij} \in \mathbb{R}^{n\times n} \end{aligned}$$

where δ_{ij} is the Kronecker delta. With these expressions, the fully discretized integral

$$L_o(\mu_i) = \sum_{j=1}^{n} f(\mu_i,\mu_j)\, w_j\, |\mu_j|\, L_j(\mu_j) \quad (i = 1,\ldots,n) \tag{8}$$

can be written as a matrix expression

$$\mathbf{L}_o = (\mathbf{F}\mathbf{W}\overline{\mathbf{M}})\,\mathbf{L}_i \tag{9}$$

The part in parentheses is the layer's scattering matrix, which must still be separated into the four blocks corresponding to reflection and transmission at the top and bottom interfaces, e.g.:

$$\left[\begin{array}{c|c} \mathbf{T}^{bt} & \mathbf{R}^{t} \\ \hline \mathbf{R}^{b} & \mathbf{T}^{tb} \end{array}\right] = \mathbf{F}\mathbf{W}\overline{\mathbf{M}}. \tag{10}$$

Note that the actual layout will depend on the chosen order of the latitude angle cosines $\mu_1, \ldots, \mu_n$.

The `Layer` class in `layerlab` provides an implementation of the discretization presented in this section as well as functionality for filling the matrix entries with content. The next section documents its usage in more detail.

4 Ingredient layers

`layerlab` supports a range of different "ingredient" layers that can be used to build more complex layered structures. The simplest layer is a diffusely reflecting opaque slab.

4.1 Diffuse surfaces

The following snippet creates a diffuse layer with albedo 0.5, using Equation (10) to compute the matrix entries from the definition of a diffuse BRDF.

```
>>> layer = ll.Layer(mu, w)
>>> layer.setDiffuse(0.5)
```

The four blocks of the layer's scattering matrix can be queried via the fields `layer[0].reflectionTop` and `layer[0].reflectionBottom` for the reflection case, plus `layer[0].transmissionTopBottom` and `layer[0].transmissionBottomTop` for transmission.

```
>>> layer[0].reflectionTop
[ 0.1582631,  0.2895543, 0.06666667;
  0.1582631,  0.2895543, 0.06666667;
  0.1582631,  0.2895543, 0.06666667]
```

The reason for the brackets in the expression `layer[0]` will be explained shortly in Section 5.

We can also compute the amount of reflected light when the layer is illuminated with a distribution of incident light in vector form, e.g. uniform illumination:

```
>>> Li = np.ones(3)
>>> np.dot(layer[0].reflectionTop, Li)
array([ 0.51448404,  0.51448404,  0.51448404])
```

The `Li` vector above has three entries matching the previously configured discretization with six ordinates for both sides. The resulting integral still has about 3% relative error due to the extremely low resolution.

The layer's BSDF also can be queried directly using the `Layer.eval(mu_o, mu_i)` method. The implementation of the latter function relies on cubic spline interpolation when it is invoked with arbitrary ordinate values that lie between the μ_i locations of the discretization:

```
>>> layer.eval(0.12, -0.34) * np.pi
0.5
```

Note the sign flip in the arguments to the `eval` function. `layerlab` uses the convention that positive μ values correspond to rays traveling deeper into the material, while negative values move towards the top. When a reflection occurs we thus have $\mu_i \cdot \mu_o < 0$. The above command performs a lookup into the layer's $\mathbf{R}^t$ block, which characterizes the reflection at the top interface.

4.2 Plotting layer scattering functions

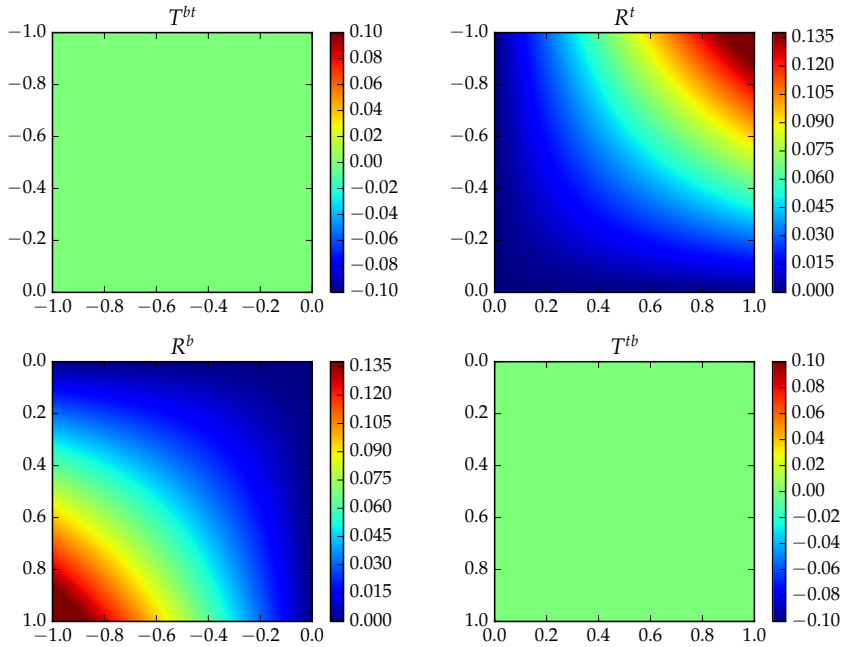
The `plot_layer` function in the following code snippet visualizes the contents of layer matrices. The function calls a vectorized form of `Layer.eval` many times and draws the resulting sequence of plots using Matplotlib's `imshow` function. The meaning of the `phi_d` argument can be ignored for now.

Note that the plot function multiplies the BSDF values with two cosine factors—this is needed to keep the results in a reasonable range for visualization (many BSDF models take on large values at grazing angles).

```
>>> def plot_layer(layer, num_samples = 200, phi_d = 0):
        fig = plt.figure(tight_layout = True)
        titles = [ '$T^{bt}$', '$R^{t}$', '$R^{b}$', '$T^{tb}$' ]
        for i in range(4):
            # Plot extents for subplot
            extent = [i%2-1, i%2, i//2-1, i//2]
            # Initialize points where the layer is evaluated
            mu_i = np.linspace(extent[0], extent[1], num_samples)
            mu_o = np.linspace(extent[2], extent[3], num_samples)
            mu_i_arg, mu_o_arg = np.meshgrid(mu_i, mu_o)

            # Evaluate the layer scattering function
            result = layer.eval(mu_o_arg, mu_i_arg, phi_d)
            # Scale by cosine factors to plot scattered energy
            result = np.array(result) * np.abs(mu_i_arg * mu_o_arg)

            # Plot result
            fig.add_subplot(2, 2, i + 1)
            plt.title(titles[i])
            plt.imshow(result, extent = extent, aspect = 'equal', origin='lower',
                vmin = 0, vmax = np.percentile(result, 99))
            plt.gca().invert_yaxis()
            plt.colorbar()
            plt.show()
```

We can immediately apply `plot_layer` to the previously computed diffuse layer, which creates zero-valued plots on the diagonal as there is no transmission.

```
plot_layer(layer)
```

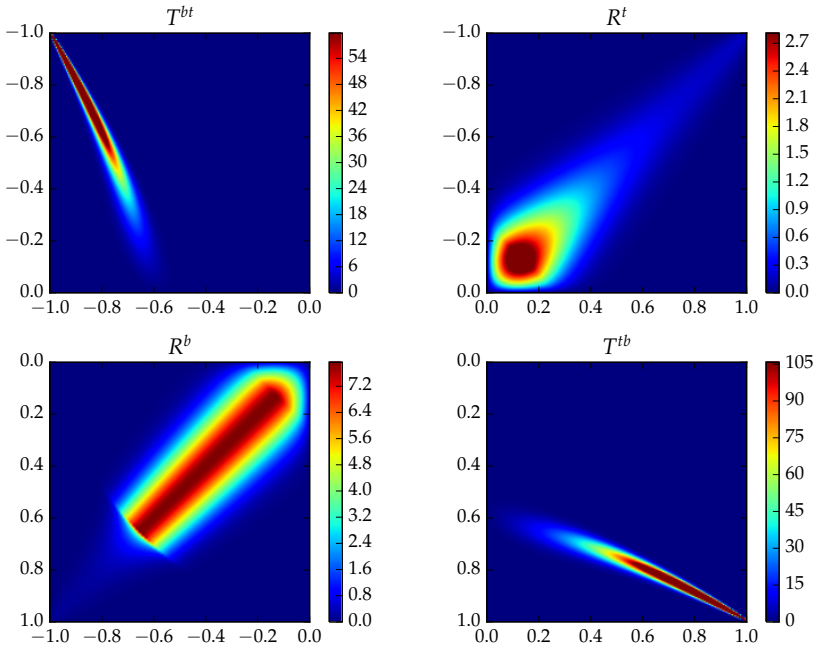


4.3 Microfacet layers

To model boundaries of dielectrics and conductors (i.e. metals), we use the microfacet model proposed by Walter et al. [Wal+07]. Microfacet models describe the interaction of light with random surfaces composed of microscopic dielectric or conducting facets that are oriented according to a microfacet distribution. Integration over this distribution then leads to simple analytic expressions that describe the expected reflection and transmission properties at a macroscopic scale.

The `Layer.setMicrofacet()` function initializes the layer with a microfacet BSDF for a given index of refraction `eta` and Beckmann roughness coefficient `alpha`. We briefly postpone a discussion of `conserveEnergy` and will explain the `m` parameter in Section 5. Note that since this layer's scattering function is considerably more detailed, we will require a higher number of ordinates.

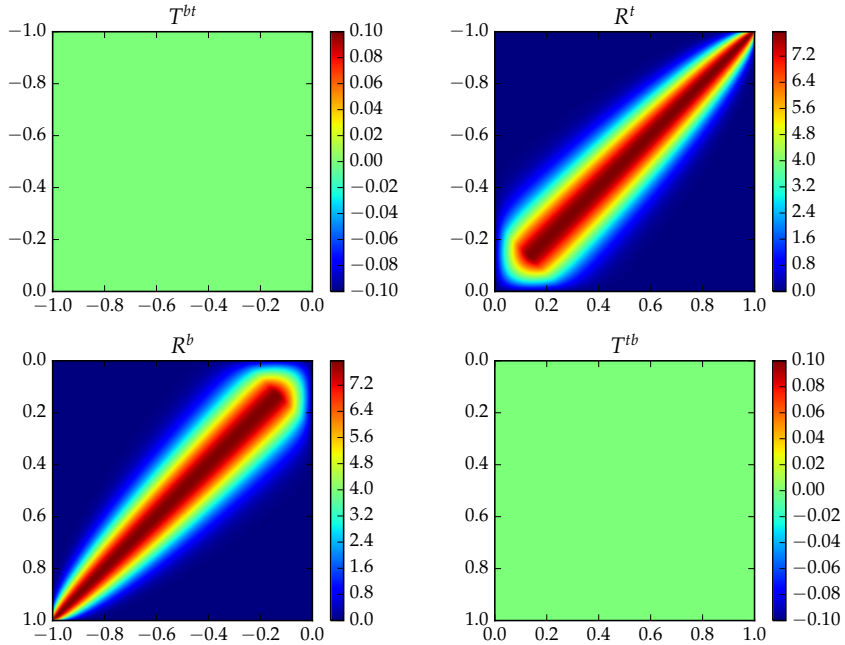
```
>>> mu, w = ll.quad.gaussLobatto(200)
>>> m = 200
>>> dielectric = ll.Layer(mu, w, m)
>>> dielectric.setMicrofacet(eta = 1.5, alpha = 0.1, conserveEnergy = False)
>>> plot_layer(dielectric)
```



The plots on the diagonal and off-diagonal illustrate the refractive and reflective behavior of the material, respectively: refracted light is slightly blurred due to the material's roughness, while reflected light undergoes a much stronger directional blur. The refracted light changes direction according to Snell's law. The reflection at the bottom interface is fairly strong until the critical angle, at which point it rapidly drops off.

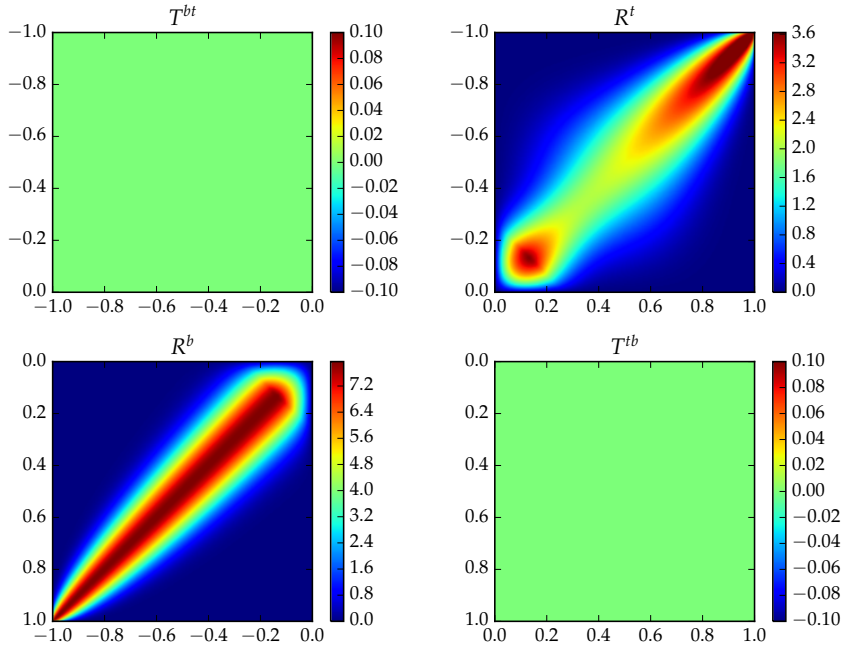
The `eta` parameter of `Layer.setMicrofacet()` actually accepts arbitrary complex valued indices of refraction. Calling the function with the imaginary value `0+1j` creates a perfectly reflective conducting layer without absorption:

```
>>> conductor = ll.Layer(mu, w, m)
>>> conductor.setMicrofacet(eta = 0+1j, alpha = 0.1, conserveEnergy = False)
>>> plot_layer(conductor)
```



Having computed dielectric and conductive layers, we can now perform our first layering experiment: the following call to `Layer.addToTop()` applies the adding equations (3) to the `dielectric` and `conductor` variables, storing the combination in `conductor`. The resulting layer describes a metal surface with a lacquer coating.

```
>>> conductor.addToTop(dielectric)
>>> plot_layer(conductor)
```

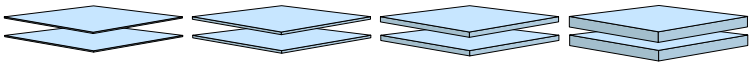


The matrix visualization shows zero-valued diagonals: the result is opaque due to the conductive input layer; the reflection at the bottom surface is also unchanged. The reflection at the top surface is interesting: it shows features of both ingredient layers. The reflection from the conductor is additionally blurred out since it must undego two refractions through the rough coating.

4.4 Isotropic and anisotropic medium layers

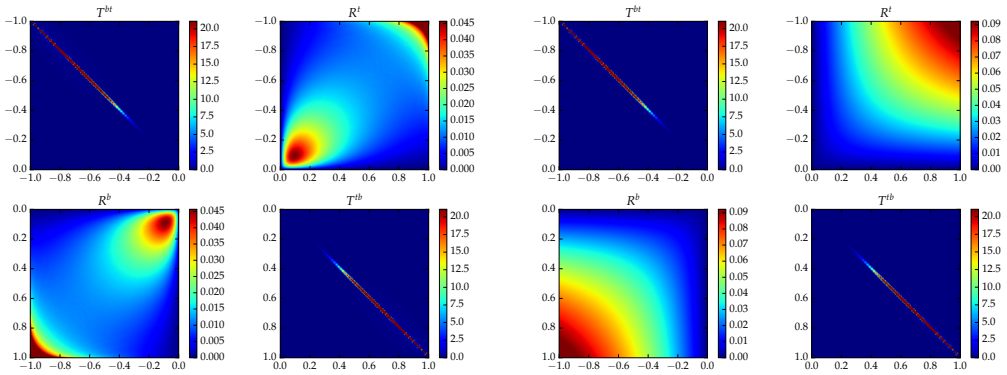
Several different techniques exist that can be used to solve for the scattering matrices of medium layers. The default in `layerlab` is the *adding-doubling* technique [Hul63].

Adding-doubling builds on the property that, as a function of optical depth, multiple scattering is a higher-order effect. For sufficiently thin layers, the portion of multiply scattered illumination is so minuscule that it can be neglected entirely. On the other hand, scattering matrices of layers with *at most* a single scattering event are easily obtained, since they admit an analytic solution. The idea of adding-doubling then is as follows: after computing the scattering matrices of a very thin layer (thin enough that multiple scattering can be neglected, we use $\tau \approx 10^{-15}$), the adding equations (3) are used to find the scattering matrices of a layer twice the thickness, by joining two identical layers. The layer is repeatedly doubled until it has the desired thickness.



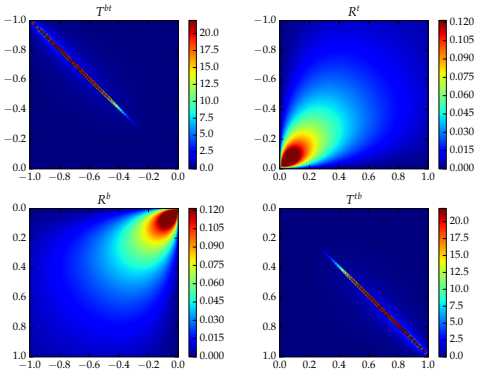
Since τ increases exponentially during doubling operations, even very thick layers can be processed rapidly. The snippet below instantiates three layers with a different Henyey-Greenstein anisotropy parameter g and expands them to an optical depth of $\tau = 2$.

```
>>> for x in [-0.9, 0, 0.9]:
>>>     medium = ll.Layer(mu, w, m)
>>>     medium.setHenyeyGreenstein(albedo = 0.95, g = x)
>>>     medium.expand(2)
>>>     plot_layer(medium)
```



(a) $g = -0.9$

(b) $g = 0$



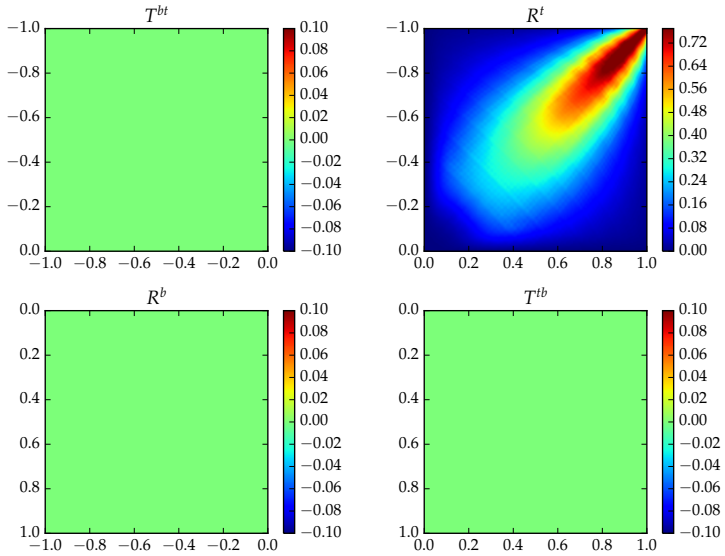
(c) $g = 0.9$

An analogous routine `setIsotropic()` exists for the special case of $g = 0$. Looking at the plots, we can see a set of high-valued diagonal entries, which correspond to attenuated light that has passed straight through the layer without being scattered. The magnitude of the diagonal entries diminishes towards $\mu = 0$, since light at these angles must travel an increasingly long distance to pass through the layer. For the $g = 0.9$ case, there is a faint, blurred transmission lobe around the diagonal, which corresponds to forward-scattered light. There is also significant variation in the angular shape of the reflection lobe between $g = -0.9$ and $g = 0.9$.

4.5 Measured layers

Finally, it is also possible to import measured materials into `layerlab` so that they can be composed with other layers. This is currently implemented for BRDF data files from the database of Matusik et al. [Mat+03].

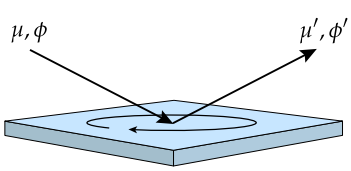
```
>>> golden_paint = ll.Layer(mu, w, m)
>>> golden_paint.setMatusik(filename = "gold-metallic-paint.binary", channel = 0)
>>> plot_layer(golden_paint)
```



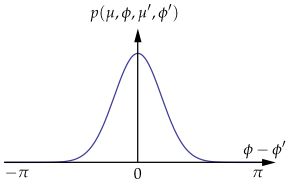
5 Accounting for the azimuth dependence

In Section 3, we introduced the mathematical tools to apply the adding equations in a flatland setting where the azimuthal dependence of the problem could be fully neglected. This is of course a severe approximation—in general, we will need to account for azimuth to obtain meaningful results. Unfortunately, reintroducing ϕ and ϕ' will increase the dimensionality of the BSDF from 2D to 4D, which greatly reduces the practicality of the tools introduced thus far since the discretizations will generally be too large to keep in memory.

We therefore restrict our analysis to surfaces that are isotropic in the sense of invariance with rotation around the normal (a), and add the further reasonable assumption of bilateral symmetry (b):



(a) Isotropy



(b) Bilateral symmetry

Together these imply that the BSDF of a layer is a three-dimensional function that depends on μ, μ', and the azimuth angle difference $\phi_d = |\phi - \phi'|$. This reduction in dimension certainly helps, but even a 3D layering problem will be challenging to solve using the discretization approach discussed so far.

5.1 Transitioning to frequency space

To address this issue, we expand all functions that depend on azimuth or azimuth difference in an even real-valued Fourier basis. In other words, a function $g(\phi)$ is now written as

$$g(\phi) = \sum_{l=0}^{\infty} g_l \cos(l(\phi - \phi')), \tag{11}$$

where g_l $(l = 0, \ldots, \infty)$ are the Fourier coefficients corresponding to different azimuthal oscillation *modes*. When starting from a smooth function g, the coefficients can be obtained using the following projection operator:

$$g_l = \mathcal{F}_l[g] = \frac{2 - \delta_{0l}}{\pi} \int_0^{\pi} g(\phi) \cos(l\phi) \mathrm{d}\phi. \tag{12}$$

Applying this projection operator to the incident and outgoing radiance function and the BSDF produces a set of coefficients that we will refer to as $L_{o,l}$, $L_{i,l}$, and f_l, respectively.

From a high level, it is not directly clear how this has made the problem any easier. However, consider what happens when we project the full local illumination integral onto this Fourier basis (see [Jak+14b] for a detailed derivation):

$$\mathcal{F}_l\Big[L_o(\mu, \phi)\Big] = \mathcal{F}_l\Big[\int_0^{2\pi} \int_{-1}^{1} f(\mu, \mu', \phi - \phi')\, L_i(\mu', \phi')\, |\mu'|\, \mathrm{d}\mu'\, \mathrm{d}\phi'\Big] \tag{13}$$

$$\Leftrightarrow \qquad L_{o,l}(\mu) = \pi(1 + \delta_{0l}) \int_{-1}^{1} f_l(\mu', \mu)\, L_{i,l}(\mu')\, |\mu'|\, \mathrm{d}\mu' \quad (l = 0, \ldots, \infty) \tag{14}$$

The crucial aspect to note about Equation (14) is that the dependence on μ and ϕ *decouples*. We're left with an infinite sequence of 2D flatland problems that *only* involve the μ variable. These are equivalent[1] to the types of simplified azimuth-independent problems that were solved in the previous section. However, the ingredients that go into these equations are not generally physically meaningful anymore. For instance, the equation with $l = 1$ describes the layer's response to incident light, which is positive at $\phi = 0$ and negative at $\phi = \pi$. If we just ignore this, solve the individual equations as usual and put the results back together in the form of Equation (11), we obtain a solution for the full problem that correctly accounts for azimuthal variation. This approach was pioneered in Chandrasekhar's study of stratified media [Cha60].

[1] Up to some minor constant factors.

At this point, we have reduced the original 4D integration problem first to 3D and then to an *infinite* sequence of much simpler 2D integration sub-problems. Of course, only a finite number of sub-problems can be considered in practice.

The magnitude of the Fourier modes f_l decreases so that a good approximation can be obtained by truncating the set of equations after some maximum index $l < m$. However, when the material's reflectance function includes sharp peaks (e.g. due to specular reflection), this constant m can unfortunately be a very large number (e.g. several thousands of 2D sub-problems).

The important question is thus: is it practical to solve a very large number of sub-problems, or would we have been off easier with the original 3D problem? This is where sparsity comes into play.

5.2 Making use of sparsity

We make the following important observations:

1. To represent very peaked reflectance functions, we require many Fourier coefficients, but this is only the case for a small set of elevation pairs (μ, μ'). For instance, in the case of specular reflection, the expansions have high frequencies only when $\mu \approx \mu'$ and low frequencies or even zeroes elsewhere.

2. Smoother reflectance functions are nonzero over many pairs (μ, μ'), but they are low frequency in the azimuth difference angle $\phi - \phi'$, and thus their Fourier series decay rapidly.

Scattering matrices of high frequency materials expressed in Chandrasekhar's directional basis are *sparse*. `layerlab` therefore relies on sparse linear algebra techniques, allowing it to go to very high orders to represent even mirror-like materials without ringing or other artifacts (Figure 2), while generating BSDF representations that require comparably little storage.

This also explains the indexing notation used to access `layer[0].reflectionTop` in Section 4. This command returns the zeroth Fourier mode, which describes the response to light that is uniform in azimuth. The 2D subproblem for each oscillation mode is completely independent from the others: when we invoked the `addToTop()` command to apply the adding equations, a sparse-aware implementation was executed 200 times in parallel for each one of them.

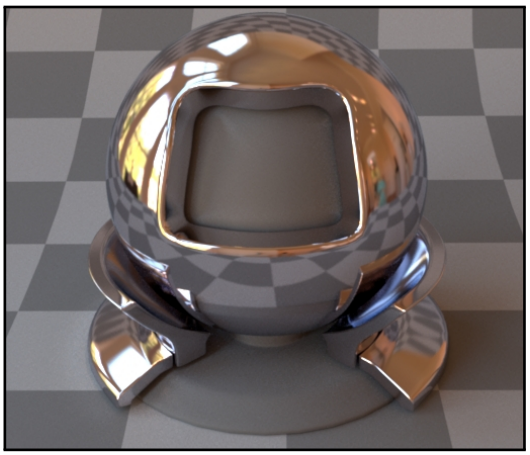


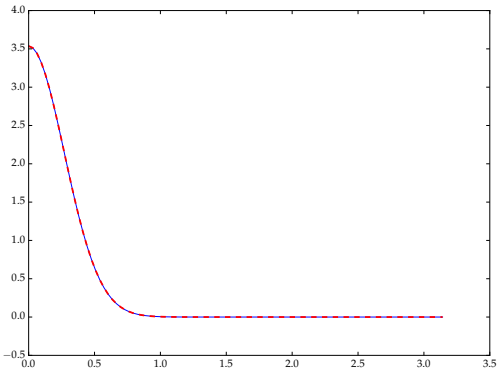
Figure 2: Frequency-based representations are usually impractical when trying to represent specular materials like this chrome object with Beckmann roughness $\alpha = 0.01$. This BSDF was projected into a basis with $m = 9763$ Fourier series terms and $n = 503$ discretizations in μ_i and μ_o, which would normally produce about 28 GiB of dense coefficient data. By exploiting sparsity, only 51.3 MiB of coefficients (0.19%) are required. Computing the discretization took 9.6 seconds; rendering took another 1.4 minutes.

Much of the technical contribution of this toolbox revolves around the efficient computation of Fourier projections of common models without having to perform a costly sequence of integrals of the form in Equation (12). For example, the `Layer.setMicrofacet()` function encountered before issues many calls to a helper function `microfacetFourierSeries()`, which relies on a fast recurrence relation to compute series coefficients for a microfacet reflectance model (see [Jak+14b] for details):

```
>>> coeffs = ll.microfacetFourierSeries(mu_o = -0.5, mu_i = 0.5, eta = 1.5, alpha = 0.3,
                                        n = 50, relerr = 1e-5)
>>> np.array(coeffs)
array([  3.82335157e-01,   7.36943424e-01,   6.59711216e-01,
         5.48716585e-01,   4.24219739e-01,   3.04993551e-01,
         2.04012100e-01,   1.27013796e-01,   7.36102434e-02,
         3.97010664e-02,   1.99084441e-02,   9.26369707e-03,
         3.98439027e-03,   1.57507016e-03,   5.62955372e-04,
         1.79824161e-04,   4.91424467e-05,   7.70838099e-06])
```

The resulting frequency space representation can be compared against the reference model implemented in `microfacet()`:

```
>>> phi_d = np.linspace(0, np.pi, 100)
>>> reference   = ll.microfacet(mu_o = -0.5, mu_i = 0.5, eta = 1.5, alpha = 0.3,
                                phi_d = phi_d)
>>> approximate = ll.fourier.evalFourier(coeffs, phi_d)
>>> plt.plot(phi_d, approximate, 'b')
>>> plt.plot(phi_d, reference, 'r', lw = 2, ls = 'dashed')
>>> plt.show()
```



6 Miscellaneous topics and experiments

In the remainder of the document, we show additional experiments and touch on some topics of practical relevance.

6.1 Validation against van de Hulst reference data

In his two-volume treatise on multiple scattering, van de Hulst [Hul80] provides a rich set of reference data for multiple scattering in isotropic and anisotropic slabs over a range of different parameters.

This is a useful source of information to assess the accuracy of our implementation. For instance, consider the highlighted row of Table 12 in Volume 1, which quantifies the reflectivity of an isotropic slab of optical depth $\tau = 2$ with an albedo of 0.9 and incident illumination arriving from $\mu = 0.9$.

TABLE 12 (*continued*)
Intensities out at Top

VECTOR	μ=0.0	μ=0.1	μ=0.3	μ=0.5	μ=0.7	μ=0.9	μ=1.0	AVERAGE N	FLUX U
b= 2.00000	μ_0 = 0.9								
FIRST ORDER	0.27778	0.25000	0.20830	0.17822	0.15528	0.13726	0.12965	0.18620	0.16280
SECOND ORDER	0.09310	0.11369	0.11532	0.10788	0.09887	0.09023	0.08624	0.10466	0.09992
THIRD ORDER	0.05233	0.06621	0.07375	0.07275	0.06878	0.06403	0.06166	0.06876	0.06829
SUMS a = 0.20	0.05976	0.05516	0.04696	0.04064	0.03565	0.03166	0.02996	0.04207	0.03720
a = 0.40	0.13059	0.12400	0.10834	0.09510	0.08417	0.07518	0.07129	0.09740	0.08728
a = 0.60	0.21937	0.21526	0.19432	0.17374	0.15553	0.13996	0.13310	0.17568	0.16009
a = 0.80	0.34275	0.35010	0.33013	0.30277	0.27533	0.25034	0.23901	0.30132	0.28063
a = 0.90	0.43124	0.45143	0.43807	0.40866	0.37554	0.34382	0.32910	0.40276	0.38040
a = 0.95	0.48841	0.51848	0.51175	0.48225	0.44593	0.40992	0.39296	0.47268	0.45008
a = 0.99	0.54410	0.58475	0.58602	0.55729	0.51820	0.47807	0.45891	0.54363	0.52137
a = 1.00	0.55987	0.60366	0.60744	0.57908	0.53926	0.49797	0.47819	0.56418	0.54210

We define the following function that evaluates a given layer structure and compares against Table 12.

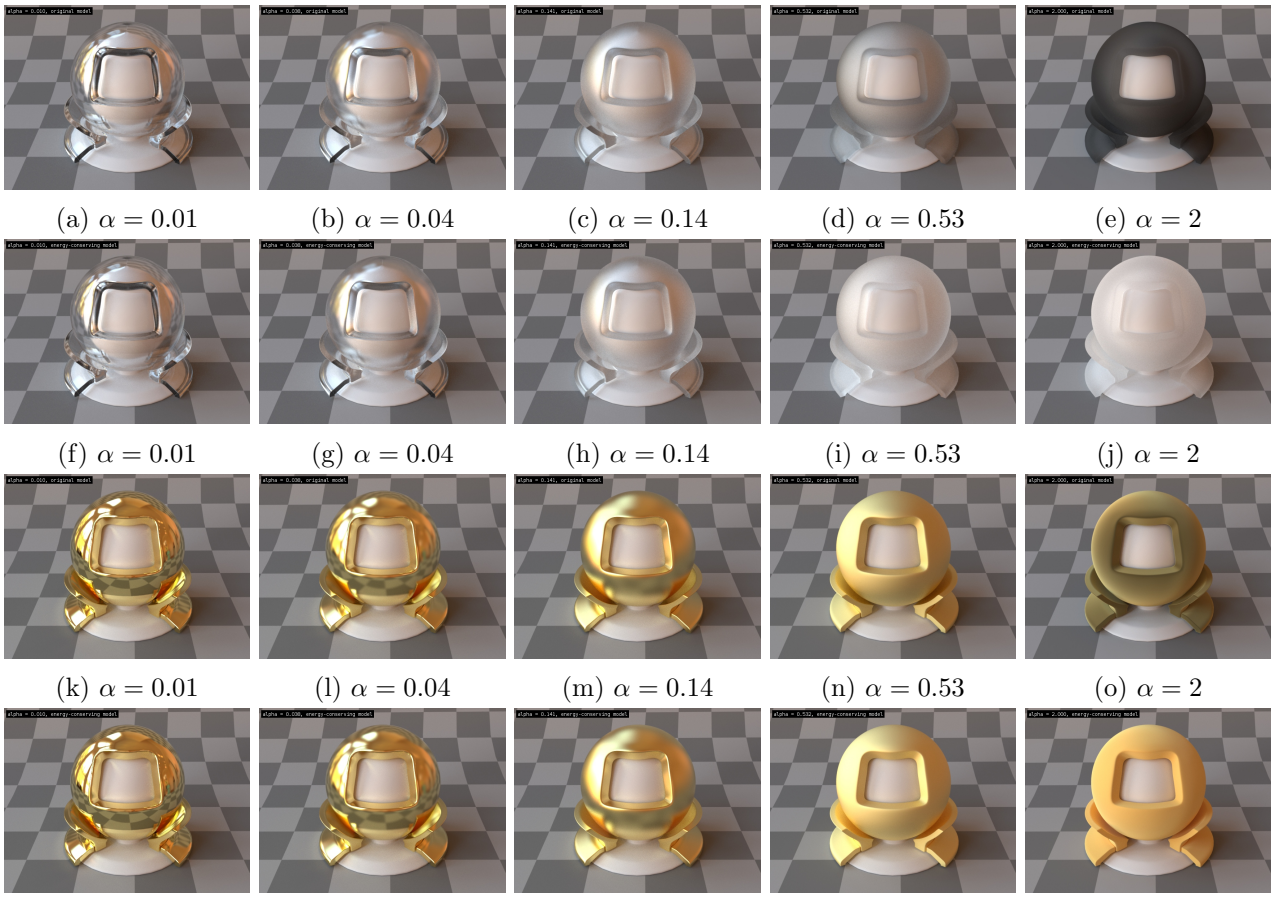
```
>>> def compare_against_vandehulst(l):
        ref = [ 0.43124, 0.45143, 0.43807, 0.40866, 0.37554, 0.34382, 0.32910]
        ref_mu = [0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0]

        for i in range(len(ref_mu)):
            result = l.eval(0.9, -ref_mu[i]) * np.pi
            print('mu_i = %.1f: van de Hulst = %.5f, computed = %.5f (rel. error = %.2e)'
                % (ref_mu[i], ref[i], result, np.abs(result-ref[i]) / ref[i]))
```

Now, we can recreate the same type of layer in `layerlab` and launch the comparison.

```
>>> mu, w = ll.quad.compositeSimpson38(10)
>>> layer = ll.Layer(mu, w)
>>> layer.setIsotropic(0.9)
>>> layer.expand(2)
>>> compare_against_vandehulst(layer)
mu_i = 0.0: van de Hulst = 0.43124, computed = 0.46310 (rel. error = 7.39e-02)
mu_i = 0.1: van de Hulst = 0.45143, computed = 0.45227 (rel. error = 1.87e-03)
mu_i = 0.3: van de Hulst = 0.43807, computed = 0.43807 (rel. error = 3.05e-07)
mu_i = 0.5: van de Hulst = 0.40866, computed = 0.40943 (rel. error = 1.87e-03)
mu_i = 0.7: van de Hulst = 0.37554, computed = 0.37624 (rel. error = 1.86e-03)
mu_i = 0.9: van de Hulst = 0.34382, computed = 0.34467 (rel. error = 2.47e-03)
mu_i = 1.0: van de Hulst = 0.32910, computed = 0.32957 (rel. error = 1.42e-03)
```

A finer discretization leads to extremely accurate results except for $\mu = 0$, i.e. light traveling parallel to the surface, where the order of convergence is lower.

```
>>> mu, w = ll.quad.compositeSimpson38(64)
>>> layer = ll.Layer(mu, w)
>>> layer.setIsotropic(0.9)
>>> layer.expand(2)
>>> compare_against_vandehulst(layer)
mu_i = 0.0: van de Hulst = 0.43124, computed = 0.43661 (rel. error = 1.24e-02)
mu_i = 0.1: van de Hulst = 0.45143, computed = 0.45143 (rel. error = 8.48e-06)
mu_i = 0.3: van de Hulst = 0.43807, computed = 0.43807 (rel. error = 3.03e-06)
mu_i = 0.5: van de Hulst = 0.40866, computed = 0.40866 (rel. error = 2.84e-06)
mu_i = 0.7: van de Hulst = 0.37554, computed = 0.37555 (rel. error = 1.81e-05)
mu_i = 0.9: van de Hulst = 0.34382, computed = 0.34382 (rel. error = 1.42e-06)
mu_i = 1.0: van de Hulst = 0.32910, computed = 0.32910 (rel. error = 8.30e-07)
```



(a) $\alpha = 0.01$ (b) $\alpha = 0.04$ (c) $\alpha = 0.14$ (d) $\alpha = 0.53$ (e) $\alpha = 2$

(f) $\alpha = 0.01$ (g) $\alpha = 0.04$ (h) $\alpha = 0.14$ (i) $\alpha = 0.53$ (j) $\alpha = 2$

(k) $\alpha = 0.01$ (l) $\alpha = 0.04$ (m) $\alpha = 0.14$ (n) $\alpha = 0.53$ (o) $\alpha = 2$

(p) $\alpha = 0.01$ (q) $\alpha = 0.04$ (r) $\alpha = 0.14$ (s) $\alpha = 0.53$ (t) $\alpha = 2$

Figure 3: Traditional microfacet models for dielectrics and conductors (rows 1 and 3) suffer from energy loss that is particularly problematic in simulations of layered materials. The proposed model adds a multiple scattering term that reintroduces this energy (rows 2 and 4).

6.2 Energy loss

One issue with currently used microfacet models is that they only account for a single scattering event at the microgeometry level; light that interacts with multiple facets is effectively ignored. These models thus incur an energy loss that grows steadily as the roughness of the interface is increased. In a simulation of layered materials with multiple rough internal boundaries, this loss is incurred many times due to interactions between layers, potentially removing significant amounts of energy.

`layerlab` can optionally apply an additive correction term to the microfacet model, which reintroduces any energy lost to multiple scattering. This term is approximate—in particular, it assumes that, following multiple interactions within the surface microgeometry, the scattered radiation emerges with an angular distribution that is close to diffuse. Figure 3 illustrates the visual effects of setting `conserveEnergy=True` when the roughness parameter `alpha` is set to a high value.

6.3 Exploring the space of multi-layered appearance models

Figures 4 and 5 show rendered results of layering experiments performed with `layerlab`. In Figure 4, a gold base layer was coated with a dielectric with an index of refraction of $\eta = 1.5$. The sequence of renderings shows the effects of varying the roughness of the two layers independently. Changing the roughness of the bottom layer generally has a stronger effect on the overall appearance, since the effect

Figure 4: A gold dragon coated with a lacquer layer (both with varying roughness). The small insets show the ingredient layers rendered on their own. **Top left**: smooth base and smooth coating. **Top right**: smooth base and rough coating. **Bottom left**: rough base and smooth coating. **Bottom right**: rough base and rough coating.

Figure 5: A shiny gold dragon with a layer of dust. **Top left**: thin layer of isotropic dust. **Top right**: thin layer of anisotropic dust. **Bottom left**: thick layer of isotropic dust. **Bottom right**: thick layer of anisotropic dust.

of this change is directionally magnified through the refractive coating.

In Figure 5, a dust layer of varying thickness and scattering anisotropy was applied to a shiny gold base; its anisotropy was modeled using the Henyey-Greenstein phase function. When the layer is thick and isotropic, little light reaches the bottom, hence the gold base is essentially invisible. When $g = 0.9$, the gold base becomes visible, as light can travel farther in a highly forward-scattering medium. When the layer is thin, it is interesting to note that the anisotropic medium blurs out the reflection from the gold base, while the reflection is merely attenuated in the isotropic case.

6.4 Spectral/RGB variation

Note that the `Layer` class is purely monochromatic and has no special support for RGB or spectral data. This is intentional—layering computations must be performed separately for each wavelength or color channel; the final set of reflectance data can then be exported to a RGB or spectrum-based rendering system.

When simulating colored metals such as gold or copper that will ultimately be exported into a RGB-based rendering one simple trick to get fairly good approximations entails convolving the complex-valued spectrally varying index of refraction with the CIE color matching curves and converting resulting XYZ values to linear RGB. The following values were obtained in this way for gold.

```
>>> layers = []
>>> for eta in [0.143552 + 3.98397j, 0.377438 + 2.38495j, 1.43825 + 1.60434j]:
>>>     l = ll.Layer(mu, w, 500)
>>>     l.setMicrofacet(eta = eta, alpha = 0.1)
>>>     # ... apply other modifications to 'l' (e.g. a coating) ...
>>>     layers.append(l)
```

6.5 Exporting BSDF data to rendering software

Finally, the `BSDFStorage.fromLayer()` or `BSDFStorage.fromLayerRGB()` functions can be used to export a `.bsdf` file that can be used with PBRT version 3 [PHJ16] and the Mitsuba renderer [Jak10].

```
>>> storage = ll.BSDFStorage.fromLayerRGB("gold.bsdf", layers[0], layers[1], layers[2])
Log: BSDFStorage::fromLayerGeneral(): merging 3 layers into "gold.bsdf" - analyzing sparsity
     pattern..
Log: Done. Number of coeff: 734778 / 13824000, sparsity=5.32%
Log: Creating sparse BSDF storage file "gold.bsdf":
Log:   Discretizations in mu  : 96
Log:   Max. Fourier orders    : 500
Log:   Color channels         : 3
Log:   Textured parameters    : 0
Log:   Basis functions        : 1
Log:   Uncompressed size      : 52.8 MiB
Log:   Actual size            : 2.9 MiB (reduced to 5.50%)
Trace: Creating memory-mapped file "gold.bsdf" (2.9 MiB)..
Log: Copying data into sparse BSDF file ..
Log: Computing cumulative distributions for importance sampling ..
Log: BSDFStorage::fromLayerGeneral(): Done.

>>> storage.close()
Trace: Unmapping "gold.bsdf" from memory
```

References

[Cha60] S. Chandrasekhar. *Radiative Transfer*. Dover Publications, 1960.

[GH69] I. Grant and G. Hunt. "Discrete space theory of radiative transfer. I. Fundamentals". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 313.1513 (1969), pp. 183–197.

[Hul63] H. C. van de Hulst. *A new look at multiple scattering*. Tech. rep. New York: Goddard Institute for Space Studies, NASA, 1963, 81pp.

[Hul80] H. van de Hulst. *Multiple light scattering*. Academic Press, 1980.

[Jak+14a] W. Jakob, E. D'Eon, O. Jakob, and S. Marschner. "A Comprehensive Framework for Rendering Layered Materials". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)* 33.4 (2014).

[Jak+14b] W. Jakob, E. D'Eon, O. Jakob, and S. Marschner. *A Comprehensive Framework for Rendering Layered Materials, Expanded Technical Report*. Tech. rep. ETH Zürich, 2014. URL: `http://www.cs.cornell.edu/projects/layered-sg14/layered-tr.pdf`.

[Jak10] W. Jakob. *Mitsuba renderer*. http://www.mitsuba-renderer.org. 2010.

[Mat+03] W. Matusik, H. Pfister, M. Brand, and L. McMillan. "A Data-Driven Reflectance Model". In: *ACM Transactions on Graphics* 22.3 (July 2003), pp. 759–769.

[PHJ16] M. Pharr, G. Humphreys, and W. Jakob. *Physically based rendering: From theory to implementation*. 3rd. Morgan Kaufmann, 2016.

[Sto60] G. G. Stokes. "On the Intensity of the Light Reflected from or Transmitted through a Pile of Plates". In: *Proceedings of the Royal Society of London* 11 (1860), pages.

[Wal+07] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. "Microfacet models for refraction through rough surfaces". In: *Proceedings of the 18th Eurographics conference on Rendering Techniques*. EGSR'07. Eurographics Association, 2007, pp. 195–206.

[WW07] A. Weidlich and A. Wilkie. "Arbitrarily layered micro-facet surfaces". In: *Proceedings of GRAPHITE '07*. ACM. 2007, pp. 171–178.

[WW11] A. Weidlich and A. Wilkie. "Thinking in Layers: Modeling with Layered Materials". In: *SIGGRAPH Asia 2011 Courses*. SA '11. New York, NY, USA: ACM, 2011.