Layered-SQT usage

M. Grady Saunders
mgs8033@rit.edu

1. Introduction

This document describes `layered-sqt`, a command-line tool for simulating the Bidirectional Scattering Distribution Function (BSDF) which emerges from a layered assembly. In this context, a *layered assembly* is a theoretical construction consisting of N layers separated by $N + 1$ participating media for $N \geq 1$. A layer is an infinite plane which is offset along (and normal to) the z-axis, and which is associated with a constituent BSDF that describes how light is scattered upon intersection. A medium is thought to occupy the space between adjacent layers or, in boundary cases, the spaces above and below the top and bottom layers respectively. See figure 1 for clarification.

The *emergent BSDF* is the BSDF observed in the limit as one backs infinitely far away from the assembly or, identically, as the assembly shrinks to an infinitesmial point. To ensure that the emergent BSDF is well-defined, `layered-sqt` requires that participating media be homogeneous, i.e., that scattering properties be independent of spatial location. Furthermore, for simplicity/tractability, `layered-sqt` does not account for wavelength-dependence.

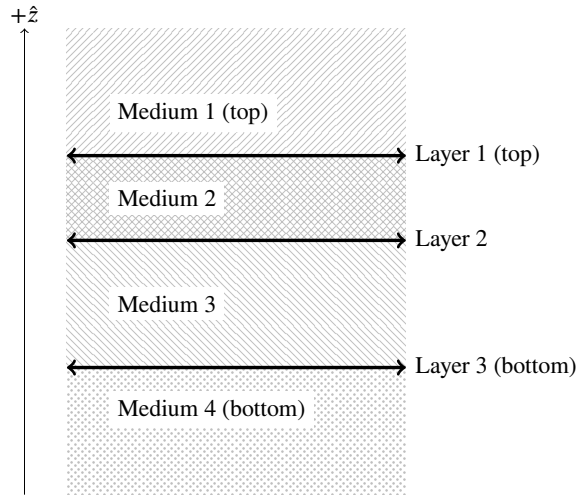

Figure 1: An example diagram of a layered assembly with 3 layers and 4 participating media.

1.1. Basics of LSQT format

The structure of a layered assembly is easy enough to convey in plain-text with rudimentary syntax, so this is the format `layered-sqt` accepts as input. By convention, we refer to this format as "LSQT format", and we suffix associated filenames with the extension `.lsqt` (though this suffix is not strictly required for the program to run).

An LSQT file is therefore a line-by-line plain-text description of a layered assembly from top to bottom. So, the first line describes the top medium, the second line describes the top layer beneath the top medium, the third line describes the medium beneath the top layer, and so on until the bottom medium. That being the case, odd-numbered lines describe media and even-numbered lines describe layers. For media as well as layers, the general syntax is

```
Name key1=val1 key2=val2
```

where `Name` is described by keyword arguments `key1` and `key2`—importantly, this syntax is whitespace-delimited, so keyword arguments of the form `key=val` must not contain whitespace. It is also worth mentioning that keyword arguments may appear in any order.

1.2. Basics of program usage

As `layered-sqt` is a command-line tool, it runs on the command line, whereby it scans command-line arguments for (optional) configuration flags and a (required) input filename appearing somewhere as a positional argument, i.e., an argument not consumed by a flag. It then parses the input file, simulates the emergent BSDF, and writes the results to a RAW-format file ready for conversion to SQT-format via `raw2sqt`. For instance,

```
$ ./layered-sqt example.lsqt -p 50000 \
  -o example.raw
```

simulates a layered assembly as described in `example.lsqt` with 50,000 paths, and writes the emergent BSDF in plain-text RAW-format to `example.raw`. To convert the BSDF to SQT-format, which is necessary for use in DIRSIG, run

```
$ ./raw2sqt example.raw
```

which will write a new file `example.sqt`.

Two of the most common flags appear above, namely `-p` (or `--path-count`) to specify the number of paths used in the simulation and `-o` (or `--output`) to specify the output filename. To see a list of all acceptable flags with brief descriptions and default values, pass `-h` (or `--help`), or simply run `layered-sqt` with no filename. As an aside, `layered-sqt` verifies parameters specified in command-line flags as well as keyword arguments specified in the input LSQT file. In the event that something has an unreasonable value, the program issues an error and fails in a controlled manner.

Lastly, `layered-sqt` recognizes the single dash filename "-" as standard input. So, it is possible to pipe the (presumably LSQT format) output of a script into `layered-sqt` directly, if this happens to be convenient. As a trivial example,

```
$ cat example.lsqt | \
  ./layered-sqt -p 50000 -o example.raw -
```

is equivalent to just passing `example.lsqt`.

2. Tutorial

This section provides a series of progressively more interesting "learn-by-doing" tutorials, which should naturally introduce the concepts and features in `layered-sqt`. To get started, create a working directory `lsqt-tutorial` and link the relevant programs for easy access. From one of the Carlson CIS servers, type the following commands:

```
$ mkdir lsqt-tutorial && cd lsqt-tutorial
$ TMP_PATH=/cis/phd/mgs8033/layered-sqt/bin
$ ln -s $TMP_PATH/layered-sqt
$ ln -s $TMP_PATH/layered-sqt-lssview
$ ln -s /dirs/pkg/dirsig/bin/raw2sqt
```

2.1. Hello, world

Let's start with something mind-numbingly simple—a 60% reflective Lambertian surface. First, create and enter a sub-directory `tutorial1`.

```
$ mkdir tutorial1 && cd tutorial1
```

Using whatever text-editing program you prefer, create a plain-text LSQT file `Lambertian.lsqt` with three lines.

```
1 Medium
2 Layer z=0 Lambertian fR=0.6
3 Medium
```

As an aside, LSQT format input is syntax-highlighted in this document. However, syntax-highlighting files/add-ons are not available (yet) for any text editor.

Now, run `layered-sqt` on `Lambertian.lsqt`,

```
$ ../layered-sqt Lambertian.lsqt
```

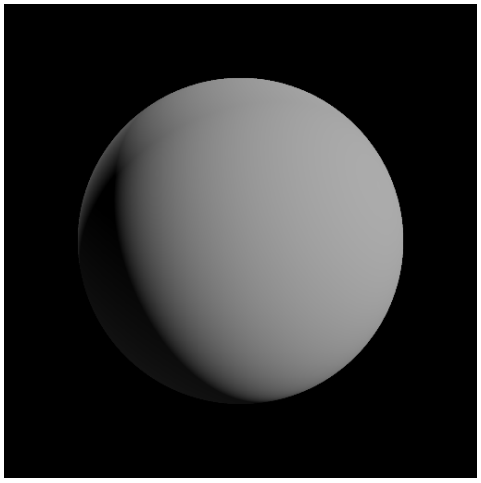

Figure 2: Image preview of a 60% reflective Lambertian BRDF output by `layered-sqt-lssview`, as in tutorial §2.1.

which should write two new files: `Lambertian.lsqt.lss` and `Lambertian.lsqt.raw`. As stated in the introduction, we may convert the plain-text RAW file to a binary SQT file for use with DIRSIG by running `raw2sqt`,

```
$ ../raw2sqt Lambertian.lsqt.raw
```

which generates the final SQT file `Lambertian.lsqt.sqt` suitable for use in a DIRSIG scene.

Now, what is the LSS file? *LSS* is an initialism for *LSQT-Slice*, which is the name of the internal file format `layered-sqt` uses to store simulation data, and it is useful 1) for previewing the layered BSDF *without setting up and rendering a DIRSIG scene*, and 2) for simulating a layered BSDF progressively, with multiple runs of `layered-sqt`.

Run `layered-sqt-lssview` to preview the Lambertian BRDF.

```
$ ../layered-sqt-lssview Lambertian.lsqt.lss
```

This should write a new file, `Lambertian.lsqt.lss.png`, which is a 512x512 rendering of the BSDF applied to a ball, shown in figure 2. When connected to the CIS server with X-window support (from a Linux machine, log-in with `ssh -X`), this image may be viewed by running `eog`.

```
$ eog Lambertian.lsqt.lss.png
```

Importantly, `layered-sqt-lssview` is *not* a full-blown path-tracer, and may not perfectly represent how the BSDF will appear in DIRSIG. It only accounts for the direct (first bounce) contributions of a few directional light sources, and it further uses tone-mapping and sRGB correction, such that the output image is not suitable for any radiometric analysis. The intended use of this preview image is to determine if a simulated BSDF is suitably convergent/noise-free.

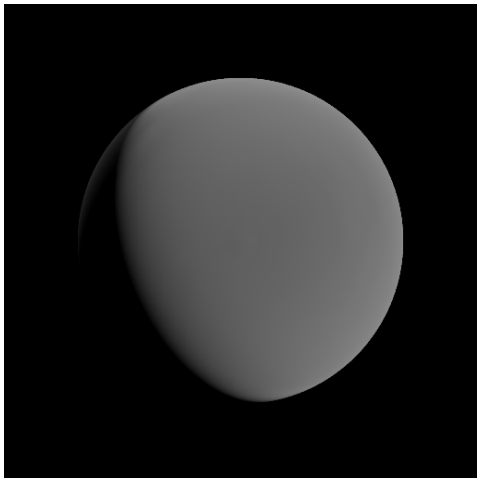

Figure 3: Image preview of a 60% reflective Lambertian BRDF with a rough microsurface output by `layered-sqt-lssview`, as in tutorial §2.2.

For more information on the Lambertian BSDF, refer to the documentation in §3.2.2.

2.2. Hello, world, with roughness

Let's expand on §2.1 by simulating a 60% reflective Lambertian BRDF with a rough microsurface—we assume that, microscopically, the surface geometry is not perfectly smooth, but is instead characterized by a distribution of normals about the primary surface normal. See the documentation in §3.2.3 for a more detailed explanation.

First, return to the `lsqt-tutorial` directory, and create and enter a new subdirectory `tutorial2`.

```
$ cd ..
$ mkdir tutorial2 && cd tutorial2
```

As before, create a new plain-text LSQT file `Rough.lsqt` with three lines.

```
1 Medium
2 Layer z=0 MicrosurfaceLambertian fR=0.6
      alpha=2.4
3 Medium
```

Above, `fR=0.6` specifies that the Lambertian BRDF is 60% reflective, and `alpha=2.4` sets the roughness parameter α to be 2.4—this is very rough. Note that setting `alpha=0` is effectively equivalent to using `Lambertian` with `fR=0.6` and `fT=0`.

Now, run `layered-sqt` on `Rough.lsqt` to generate files `Rough.lsqt.lss` and `Rough.lsqt.raw`. This should take longer to simulate than before, due to the computational complexity of the microsurface model.

```
$ ../layered-sqt Rough.lsqt
```

When this is complete, use `layered-sqt-lssview` to generate `Rough.lsqt.lss.png`, shown in figure 3.

```
$ ../layered-sqt-lssview Rough.lsqt.lss
```

Notice that the microsurface roughness causes increased reflectance at grazing angles, and decreased reflectance elsewhere.

2.3. A simple substrate

Let's now consider a simple two-layer assembly to model a substrate. We imagine a 60% reflective diffuse Lambertian surface with a glossy dielectric coating. Before continuing, again return to the `lsqt-tutorial` directory, and create and enter a new subdirectory `tutorial3`.

```
$ cd ..
$ mkdir tutorial3 && cd tutorial3
```

Then create a new plain-text LSQT file `Substr.lsqt` with five lines as below.

```
1 Medium
2 Layer z=1 MicrosurfaceDielectric alpha=0.2
3 Medium eta=1.4
4 Layer z=0 Lambertian fR=0.6
5 Medium
```

This assembly has two layers, appearing on lines 2 and 4. The first layer is positioned at z-height 1 with a `MicrosurfaceDielectric`—this is similar to `MicrosurfaceLambertian`, but it applies the microsurface ideology to the dielectric Fresnel mirror BSDF instead of the Lambertian BRDF. This layer forms the surface of our glossy dielectric coating. The second layer is positioned at z-height 0 with a `Lambertian` as in tutorial §2.1.

The medium between these layers on line 3 is attributed a refractive index η (`eta`) of 1.4. Note that *every* medium in an LSQT file has a refractive index, which defaults to 1 if left unspecfied. Note also that `MicrosurfaceDielectric` depends on the refractive indices of the media above and below the layer to which it is assigned, as it generalizes the dielectric Fresnel mirror BSDF.

Run `layered-sqt` as usual and preview the resulting LSS file with `layered-sqt-lssview`.

```
$ ../layered-sqt Substr.lsqt
$ ../layered-sqt-lssview Substr.lsqt.lss
```

Now, upon viewing `Substr.lsqt.lss.png`, we might suspect that something has gone wrong. As shown in figure 4, there is quite a bit of low frequency noise and artifacting. What is going on?

Behind the scenes, `layered-sqt` estimates values of the emergent BSDF at a finite number of sample directions using Monte Carlo integration, i.e., integration by averaging

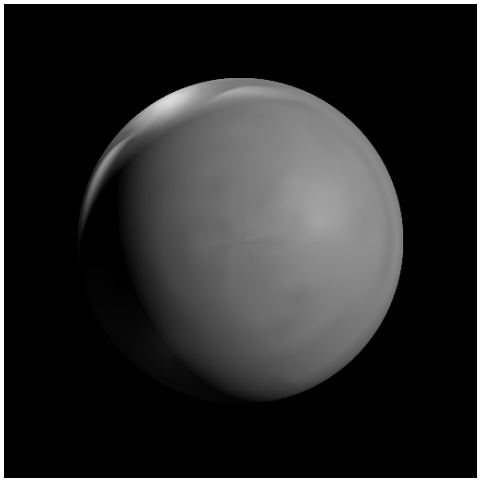

Figure 4: Image preview of the simple substrate BRDF in tutorial §2.3, as output from running `layered-sqt` with default settings. As is evident, the BRDF would benefit from more samples (higher resolution), and more simulated paths (less noise).

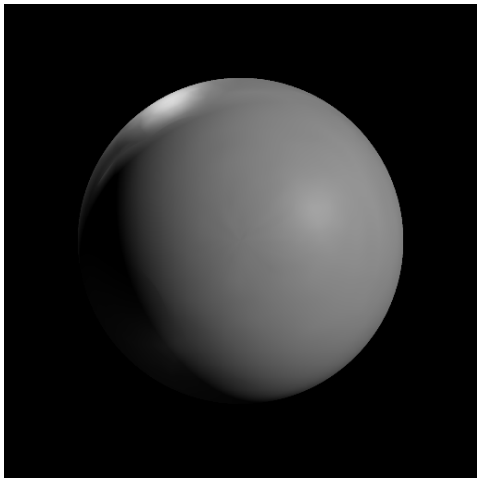

Figure 5: Image preview of the simple substrate BRDF in tutorial §2.3, as output from running `layered-sqt` with higher settings. In particular, this uses 200 incident directions (instead of the default 80) and 100,000 paths (instead of the default 10,000).

random evaluations. So, two central parameters controlling simulation fidelity are 1) the number of sample directions and 2) the number of random evaluations used to estimate the BSDF at each sample direction. By default, `layered-sqt` uses 80 sample directions and 10,000 random evaluations of potential light paths to estimate the emergent BSDF.

We can change the number of directions ω_i on the command line using the flag `-wi` (or `--wi-count`), and we can change the number of paths using the flag `-p` (or `--path-count`). In general, we should think to increase the number of directions to increase *resolution*, and we should think to increase the number of paths to reduce *noise*.

In the case of our simple substrate BRDF, let's increase the number of directions from 80 to 200, and the number of paths from 10,000 to 100,000. Run `layered-sqt` with the following flags:

```
$ ../layered-sqt -R -wi=200 \
  -p=100000 -rp=0.2 Substr.lsqt
```

This should take a minute or two to compute. Upon previewing the BRDF with `layered-sqt-lssview`, we see something much more sensible, as shown in figure 5.

The flag `-R` (or `--restart`) tells `layered-sqt` to ignore the LSS file and restart the simulation from scratch. If the LSS file exists, then `layered-sqt` will ignore all flags except `-p`/`--path-count` by default, and add this number of paths to the existing simulation data (it will also display a message to remind us that is doing this). So, `-R` is necessary to regenerate sample directions.

We have specified the number of sample directions with `-wi`, but it remains a non-trivial problem to determine what these directions ought to be—`layered-sqt` implements a presumably novel algorithm to form a so-called *Redundancy Reduced Sample Set (RRSS)* of directions by initially sampling *more* directions than the user requests, then choosing the subset of those directions that minimizes a quantitative measure of "redundancy" with respect to a desirable density, being the normalization of the non-cosine-weighted BSDF.

The flag `-rp` (or `--rrss-path-frac`) thus specifies the fraction of paths (as specified by `-p`) used to initially estimate the emergent BSDF in forming the RRSS of directions ω_i. There is also the flag `-rx` (or `--rrss-oversampling`), to specify the RRSS oversampling multiplier. (We discuss this flag for completeness, but in practice it is never really necessary to specify `-rx`, as the default of 4 is generally sufficient.) For example,

```
$ layered-sqt \
  -wi=100 -p=20000 -rp=0.25 -rx=5 File.lsqt
```

samples $100 \times 5 = 500$ directions initially, and uses $20000 \times 0.25 = 5000$ paths to estimate the emergent BSDF at each of these 500 directions, then forms an RRSS containing 100 of these 500 directions, and lastly uses $20000 - 5000 = 15000$ paths to improve the estimates of the emergent BSDF at each of these 100 directions (such that the total number of paths is 20000).

In general, increasing `-rp` and `-rx` leads to better sample placement, at the (potentially extreme) cost of computation time. We discuss practical simulation strategies using these flags, including incremental simulation with multiple runs of `layered-sqt`, over the remaining tutorial sections.

2.4. Adding dust

Let's now add a layer of back-scattering dust to the surface of the substrate BRDF in tutorial §2.3. As usual, return to the `lsqt-tutorial` directory, and create and enter a new subdirectory `tutorial4`.

```
$ cd ..
$ mkdir tutorial4 && cd tutorial4
```

Next create another plain-text LSQT file `Dusty.lsqt` with three layers and thus seven lines.

```
1 Medium
2 Layer z=2 Null
3 Medium mus=0.2 HenyeyGreenstein g=-0.4
4 Layer z=1 MicrosurfaceDielectric alpha=0.2
5 Medium eta=1.4
6 Layer z=0 Lambertian fR=0.2
7 Medium
```

We attribute a `Null` to the topmost layer appearing on line 2, such that this layer only exists to separate media, and no scattering happens at the layer itself. The medium above on line 1 is vacuum as usual. The medium below on line 3 models our layer of dust. Lines 4 through 7 are just lines 2 through 5 from the previous section, except we have reduced f_R from 60% to 20% to exaggerate the dust effect.

Similarly to refractive index η (`eta`), *every* medium in an LSQT file has two volume scattering parameters—the scattering coefficient μ_s (`mus`) and the absorption coefficient μ_a (`mua`), which default to zero if left unspecified. Scattering and absorption events happen according to Beer's Law in a homogeneous medium. That is, the probability of scattering within a particular distance d is given by an exponential distribution $1-\exp(-\mu_s d)$, where μ_s is the distribution parameter, and analogously for absorbing within a particular distance d and μ_a. We thus identify the units of μ_s and μ_a as "inverse distance", and we interpret the reciprocals $1/\mu_s$ and $1/\mu_a$ as the mean distances between scattering and absorption events respectively. Note that in the limit $\mu_s = \mu_a = 0$ (the default values), the mean distances tend to infinity, such that scattering and absorption events cease to exist.

To simulate a thin layer of back-scattering dust, we attribute a scattering coefficient μ_s (`mus`) of 0.2 and a `Henyey-Greenstein` with shape parameter g of -0.4 (more on this in a moment). We choose μ_s to be 0.2 such that the mean distance between scattering events is $1/\mu_s = 5$ units. This is somewhat large in comparison to the thickness of the dust layer, which is 1 unit (the difference of the z-heights of layers on lines 2 and 4), so the dust appears suitably "thin". When plucking this number out of the sky, this is the line of reasoning to use.

The Henyey-Greenstein phase function is widely used in computer graphics due to its simplicity and intuitive parameterization (although it was originally intended to model scattering of interstellar dust clouds). The phase function is equipped with a single shape parameter $g \in (-1, +1)$, which is identically the mean scattering cosine. That is, it tends to forward scatter as $g \to +1$ and back scatter as $g \to -1$. See the documentation in §3.1.1 for more information.

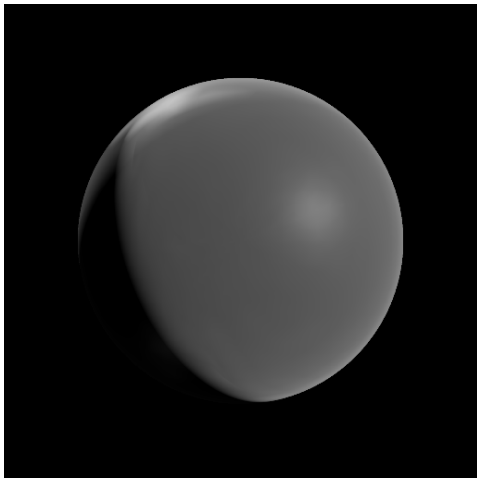

Figure 6: Image preview of the modified simple substrate BRDF with back-scattering dust in tutorial §2.4, where the bottom Lambertian BRDF layer is reduced to 20% reflectance to exaggerate the dust effect.

Now run `layered-sqt` on `Dusty.lsqt` with the same flags as in the previous section.

```
$ ../layered-sqt -wi=200 \
  -p=100000 -rp=0.2 Dusty.lsqt
```

Notice that we do not need `-R` this time because no LSS file exists yet. As before, this should take a minute or two to compute. The image preview as output by `layered-sqt-lssview` is shown in figure 6.

3. Documentation

This section documents the properties of participating media and scattering layers in more detail, and enumerates the available scattering models for media and layers, and how to specify them in LSQT format. Note that "phase function", "BRDF", and "BSDF" are specific types of scattering models. In particular,

- a *phase function* is a scattering model which accounts for volume-scattering within a participating medium,
- a *BRDF*, i.e., a *Bidirectional Reflectance Distribution Function*, is a scattering model which accounts for only reflection at a surface, and
- a *BSDF*, i.e., a *Bidirectional Scattering Distribution Function*, is a scattering model which accounts for reflection and transmission at a surface.

Note that phase functions are normalized by definition, as volume-absorbtion is accounted for by a separate parameter. BRDFs and BSDFs, however, account for absorption as well as scattering, and so are only normalized if the model is non-absorbing (in other words, perfectly energy conserving). To state all of this more rigorously, a phase function p is normalized with respect to integration over incident directions ω_i, such that

$$\int_{\mathcal{S}^2} p(\omega_o \to \omega_i)\, \mathrm{d}\omega_i = 1.$$

However, a BRDF/BSDF f is only normalized as such if it is non-absorbing,

$$\int_{\mathcal{S}^2} f(\omega_o \to \omega_i)\, \mathrm{d}\omega_i = 1 \iff f \text{ is non-absorbing.}$$

As this ought to suggest—this document follows the convention that BRDFs and BSDFs contain an implicit cosine-weighting with respect to incident direction. This is often written explicitly elsewhere in the literature, such that the above normalization condition is written as

$$\int_{\mathcal{S}^2} f(\omega_o \to \omega_i) \left|\cos\theta_i\right| \mathrm{d}\omega_i = 1.$$

We effectively make the substitution $f \leftarrow f|\cos\theta_i|$.

3.1. Participating media

A medium is characterized by its absolute index of refraction $\eta > 0$, its absorption coefficient $\mu_a \geq 0$, its scattering coefficient $\mu_s \geq 0$, and a local phase function model.

To specify a medium, simply type the name `Medium` followed by (optional) keywords arguments `eta`, `mua`, and `mus` for η, μ_a, and μ_s respectively. Keyword arguments for a medium have default values `eta=1`, `mua=0`, `mus=0`, such that any may be omitted for brevity. If $\mu_s > 0$, the keyword arguments must be followed by the name of the desired phase function, followed by any keyword arguments for the phase function. If $\mu_s = 0$, it is not necessary to specify a phase function since no scattering occurs.

As the defaults may suggest, `layered-sqt` considers that vacuuum is just a medium with refractive index 1 which neither absorbs nor scatters. So,

```
Medium
```

specifies vacuum.

3.1.1. Henyey-Greenstein phase

The Henyey-Greenstein phase function is given by

$$p(\omega_o \to \omega_i) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 + 2g\omega_o \cdot \omega_i)^{3/2}}$$

with shape parameter $g \in (-1, 1)$. It may be helpful to note that g is the mean cosine of the scattered direction with respect to the incident direction, such that p becomes forward scattering as $g \to 1$, back scattering as $g \to -1$, and uniformly/isotropically scattering as $g \to 0$. See d'Eon's write-up [1, p. 19] for more information.

To specify the Henyey-Greenstein phase in LSQT format, use the name `HenyeyGreenstein` followed by the (optional) keyword argument `g` for g, which is zero by default. For example,

```
Medium mus=1 HenyeyGreenstein g=-0.22
```

specifies a moderately back-scattering medium.

3.1.2. Rayleigh phase

The general form of the Rayleigh phase function is given by

$$p(\omega_o \to \omega_i) = \frac{3}{16\pi}\left[\frac{1+3\gamma}{1+2\gamma} + \frac{1-\gamma}{1+2\gamma}(\omega_o \cdot \omega_i)^2\right]$$

where

$$\gamma = \frac{\rho}{2-\rho}$$

where, in turn, $\rho \in [-1, 1]$ is a shaping parameter known as the *depolarization factor*. The simpler, more common form of the Rayleigh phase function is given by

$$p(\omega_o \to \omega_i) = \frac{3}{16\pi}(1 + (\omega_o \cdot \omega_i)^2),$$

which is obtained by setting $\rho = 0$. See d'Eon's write-up [1, p. 19] for more information.

The Rayleigh phase function prefers parallel scattering to perpendicular scattering. That is to say, scattering in directions perpendicular to the direction of propagation is diminished, and scattering is symmetric in terms of forward- versus back-scattering. This effect is maximized as $\rho \to -1$ and minimized as $\rho \to +1$. Note that p becomes uniformly/isotropically scattering in the limiting case $\rho = +1$.

To specify the Rayleigh phase function in LSQT format, use the name `Rayleigh` followed by the (optional) keyword argument `rho` for ρ, which is zero by default. For example,

```
Medium mus=1 Rayleigh rho=-0.5
```

specifies a medium with an exaggerated "Rayleigh effect" relative to the common form where $\rho = 0$.

3.1.3. SGGX phase

The Symmetric GGX (SGGX) phase function, given by Heitz et al. [5], is based on *microflake theory*, wherein we assume volume scattering happens due to surface scattering with infinitely many microscopic disjoint flakes. The general form of the phase function is

$$p(\omega_o \to \omega_i) = \int_{\mathcal{S}^2} p_m(\omega_m, \omega_o \to \omega_i) D_{\omega_o}(\omega_m)\, \mathrm{d}\omega_m$$

where p_m is the microscopic phase function assigned to each flake, and D_{ω_o} is the distribution of *visible* flake normals for a particular viewing/outgoing direction ω_o.

The distribution of flake normals D is defined by its projected areas with respect to three orthogonal directions. In particular,

$$D(\omega_m) = \frac{1}{\pi\sqrt{\det \boldsymbol{S}}(\omega_m \cdot \boldsymbol{S}^{-1}\omega_m)^2}$$

where $\boldsymbol{S}$ is a 3×3 real symmetric matrix. The eigenvalues of $\boldsymbol{S}$ are squared projected areas, and the eigenvectors of $\boldsymbol{S}$ and the orthogonal coordinate directions. The distribution of *visible* normals D_{ω_o} is obtained from D as

$$D_{\omega_o}(\omega_m) = \frac{D(\omega_m)}{\sigma(\omega_o)}\langle\omega_o, \omega_m\rangle$$

where σ yields projected area along ω_o and

$$\langle\omega_o, \omega_m\rangle = \begin{cases} \omega_o \cdot \omega_m & (\omega_o, \omega_m) \text{ in same hemisphere,} \\ 0 & \text{otherwise.} \end{cases}$$

So, unlike the phase functions in the previous sections, the SGGX phase function has a global orientation, meaning that p depends on the specific values of directions ω_o and ω_i, and not just on the angle between them.

Due to the constraint in `layered-sqt` that the emergent BSDF must be isotropic, we cannot allow all configurations of $\boldsymbol{S}$. If $\boldsymbol{S}$ is written as the eigendecomposition $\boldsymbol{S} = \boldsymbol{Q}^\top \boldsymbol{\Lambda} \boldsymbol{Q}$, then 1) we fix $\boldsymbol{Q} = \boldsymbol{I}$, the identity matrix, and we 2) require that the eigenvalues of $\boldsymbol{\Lambda}$ describing projected area in the XY plane be identical. We thus obtain

$$\boldsymbol{S} = \begin{bmatrix} A_\parallel^2 & 0 & 0 \\ 0 & A_\parallel^2 & 0 \\ 0 & 0 & A_\perp^2 \end{bmatrix}$$

with free parameters $A_\parallel > 0$ and $A_\perp > 0$.

To specify the SGGX phase function in LSQT format, use the name `Sggx` followed by (optional) keyword arguments `Apara` for $A_\parallel$, `Aperp` for $A_\perp$, and `type` for the micro-phase function type (either `Specular` or `Diffuse`). For example,

```
Medium mus=1 Sggx Apara=0.5 type=Diffuse
```

By default, `Apara=1`, `Aperp=1`, and `type=Specular`. Note that only the ratio of $A_\parallel$ to $A_\perp$ matters, so it always acceptable to fix $A_\perp = 1$ and just specify $A_\parallel$ (or vice versa).

3.2. Layers

A layer is characterized by its z-height and a local BSDF model. As different BSDF models require different parameters, there is a different "type" of layer for each BSDF model implemented in `layered-sqt`.

Every layer accepts z-height as a parameter, so every line describing a layer must begin with the name `Layer` followed by the (required) keyword argument `z`. This is followed in turn by the name of the desired BSDF, followed by any keyword arguments for the BSDF. For example,

```
Layer z=1 Null
```

specifies a layer at z-height 1 with a null BSDF. As explained in the next sub-section, a null BSDF is a special case which accepts no additional keyword arguments.

The implementation requires that layers appear in top-to-bottom order, such that the z-heights of subsequent layers in an LSQT file are strictly decreasing. If this condition is violated, the implementation issues an error and exits.

3.2.1. Null BSDF

At times, it is desirable to separate participating media *without* scattering at a layer. This is possible in `layered-sqt` by assigning a null BSDF with name `Null`, which may be interpreted as a 100% transmissive directional delta function in the direction a ray is already traveling. This is useful, for example, to model a layer of dust on top of a surface. To do so, we might specify the following LSQT file

```
1 Medium
2 Layer z=1 Null
3 Medium mus=1 HenyeyGreenstein g=0.2
4 Layer z=0 Lambertian fR=1
5 Medium
```

which describes a layer of forward-scattering "dust" on top of a 100% reflective Lambertian surface.

3.2.2. Lambertian BSDF

The Lambertian BSDF models a uniformly scattering surface which may reflect, transmit, or both. That is, the (cosine-weighted) scattering function is

$$f(\omega_o \to \omega_i) = \frac{|\cos\theta_i|}{\pi} \begin{cases} f_R & (\omega_o, \omega_i) \text{ in same hemisphere} \\ f_T & \text{otherwise} \end{cases}$$

where f_R and f_T specify the amount of incident energy reflected and transmitted respectively. Both f_R and f_T should be non-negative numbers such that $f_R + f_T \leq 1$ for this to be physically plausible. In the event that $f_R + f_T = 1$, this is perfectly energy-conserving.

To specify the Lambertian BSDF in LSQT format, use the name `Lambertian` followed by (optional) keyword arguments `fR` and `fT` for f_R and f_T respectively. For example,

```
Layer z=2.2 Lambertian fR=0.8 fT=0.1
```

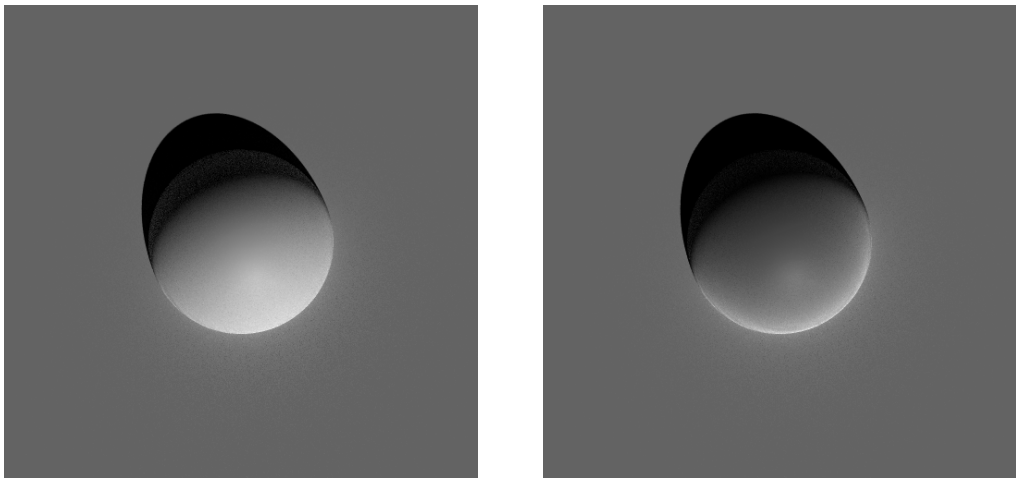

Figure 7: A sphere with an LSQT BRDF on a Lambertian ground plane rendered from above with DIRSIG5. The LSQT BRDF models a dusty dielectric substrate—from top to bottom, we have a vacuum medium, a null BSDF layer, a slightly back-scattering medium with $g = -0.2$, a decently smooth microsurface dielectric BSDF layer, a transparent medium with $\eta = 1.5$, and a Lambertian BRDF layer. On the left, the Lambertian layer has $f_R = 0.8$. On the right, the Lambertian layer has $f_R = 0.4$, such that the backscattering due to "dust" is more obvious. All other parameters are the same in both images.

specifies a Lambertian surface at z-height 2.2 which is 80% reflective, 10% transmissive, and 10% absorptive. By default, `fR=1` and `fT=0`.

3.2.3. Microsurface Lambertian BSDF

A microsurface is thought to be an infinitesimally thin cloud of microfacets, where each facet is thought to scatter light according to another, simpler BSDF. The cloud is characterized geometrically by 1) a distribution of the slopes of the facets and 2) a distribution of the heights of the facets, which are assumed to uncorrelated, such that the facets are discontinuous. The distribution of slopes is parameterized by its so-called roughness α (more generally, anisotropic roughness α_x, α_y). As $\alpha \to \infty$, the distribution of slopes widens and the emergent BSDF appears rougher. As $\alpha \to 0$, the distribution of slopes collapses, recovering the initial, simpler BSDF in the limiting case $\alpha = 0$.

As given in [4], the microsurface Lambertian BSDF is a multiple scattering (stochastic) microfacet model wherein facets are assumed to be Lambertian scatterers. This is particularly useful for representing rough diffuse surfaces. The implementation in `layered-sqt` is parameterized by roughness $\alpha > 0$, the Lambertian BSDF coefficients $f_R \in [0, 1]$ and $f_T \in [0, 1]$, and the number of local stochastic process iterations $n_{\text{iter}} \geq 1$. For rougher microsurfaces (say, $\alpha > 0.4$), it is typically more efficient to increase n_{iter} rather than the global path count.

To specify the microsurface Lambertian BSDF in LSQT format, use the name `MicrosurfaceLambertian` followed by keyword arguments `alpha`, `fR`, `fT`, and `iter_count` for α, f_R, f_T, n_{iter} respectively. By default, `alpha=0.5`, `fR=1`, `fT=0`, and `iter_count` is chosen to be 1, 2, 4, or 6 depending on `alpha`. For reference, $\alpha < 0.1$ is very smooth, $0.1 < \alpha < 0.8$ is somewhat rough, and $0.8 < \alpha$ is very rough. Furthermore, multiple scattering may be disabled by the keyword argument

```
use_multiple_scattering=false
```

in which case only the single scattering term is computed. It is important to note that, if multiple-scattering interactions are ignored, then $f_R + f_T = 1$ does *not* guarantee that the BSDF is perfectly energy conserving. For small roughness values (say, $\alpha < 0.1$), the energy carried by multiple scattering interations is insignificant, and so this may not be a concern. For large roughness values however, multiple scattering is important to prevent energy loss.

3.2.4. Microsurface dielectric BSDF

The microsurface dielectric BSDF follows the same logical construction as the microsurface Lambertian BSDF, except the constituent BSDF assigned to the facets is the delta Fresnel mirror BSDF. The implementation in `layered-sqt` is parameterized by roughness $\alpha > 0$, a scaling factor for the Fresnel mirror BRDF $k_R \in [0, 1]$, a scaling factor for the Fresnel mirror BTDF $k_T \in [0, 1]$, and the number of local stochastic process iterations $n_{\text{iter}} \geq 1$.

To specify the microsurface dielectric BSDF in LSQT format, use the name `MicrosurfaceDielectric` followed by keyword arguments `alpha`, `kR`, `kT`, and `iter_count` for α, k_R, k_T, and n_{iter} respectively. By default, `alpha=0.5`,

`kR=1`, `kT=1`, and `iter_count` is chosen to be 1, 2, 4, or 6 depending on `alpha`.

Unlike the Lambertian case, the single-scattering term is directly computable due to the delta function in the dielectric Fresnel BSDF—such that disabling multiple-scattering renders `iter_count` irrelevant. This leads to noticeably faster simulations, so much so that multiple-scattering is disabled by default. Energy loss is less noticeable than the Lambertian case for moderately rough surfaces, but is still an issue for very rough surfaces (say $\alpha > 0.8$). As such, multiple-scattering may be enabled at the user's discretion by the keyword argument

```
use_multiple_scattering=true
```

to ensure proper energy conservation.

3.2.5. Microsurface conductive BRDF

TODO

3.2.6. Oren-Nayar diffuse BRDF

TODO

References

[1] E. d'Eon. *A Hitchhiker's Guide to Multiple Scattering. An Incomplete Collection of Mostly Steady-State Monoenergetic Neutral-Particle Problem Solutions*. 2016. URL: http://www.eugenedeon.com/project/a-hitchhikers-guide-to-multiple-scattering/.

[2] A. A. Goodenough and S. D. Brown. "DIRSIG5: Next-Generation Remote Sensing Data and Image Simulation Framework". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 10.11 (Nov. 2017), pp. 4818–4833. ISSN: 2151-1535. DOI: 10.1109/JSTARS.2017.2758964.

[3] E. Heitz. "Understanding the masking-shadowing function in microfacet-based BRDFs". In: *Journal of Computer Graphics Techniques (JCGT)* 3.2 (June 2014), pp. 48–107. ISSN: 2331-7418. URL: http://jcgt.org/published/0003/02/03/paper.pdf.

[4] E. Heitz et al. "Multiple-scattering microfacet BSDFs with the Smith model". In: *ACM Transactions on Graphics* 35.4 (July 2016), pp. 1–58. ISSN: 0730-0301. DOI: 10.1145/2897824.2925943. URL: https://doi.org/10.1145/2897824.2925943.

[5] E. Heitz et al. "The SGGX Microflake Distribution". In: *ACM Trans. Graph.* 34.4 (July 2015). ISSN: 0730-0301. DOI: 10.1145/2766988. URL: https://doi.org/10.1145/2766988.