Preparing the Generalized Harvey-Shack rough surface scattering method for use with the Discrete Ordinates Method

Villads Egede Johansen[1, *]

[1] *Technical University of Denmark, Department of Mechanical Engineering, 2800 Kgs. Lyngby,Denmark*

compiled: August 1, 2017

The paper shows how to implement the Generalized Harvey-Shack (GHS) method for isotropic rough surfaces discretized in a polar coordinate system and approximated using Fourier series. This is in particular relevant for the use of the GHS method as boundary condition for radiative transfer problems in slab geometries, where the Discrete Ordinates Method can be applied to solve the problem. Furthermore, such an implementation is a more convenient discretization of the problem than the traditional direction cosine space that has its strengths in analytical problems and intuitive understanding (mainly due to its translation invariance). A computer implementation of scattering from a Gaussian rough surface with Gaussian autocovariance written in Python is included at the end of the paper.

OCIS codes: (290.5835) Scattering, Harvey;(240.5770) Roughness; (290.1483) BSDF, BRDF, and BTDF; (010.5620) Radiative transfer; (110.7050) Turbid media

http://dx.doi.org/10.1364/XX.99.099999

1. Introduction

Scattering from rough surfaces continues to be an active area of research in optics [1]. A multitude of approaches to solve roughness scattering problems exists [2], all with their advantages and disadvantages – most often tailored to a range of applications. One set of often used methods is based on single scattering theories [1], which historically have been popular due to their simplicity. However, there is a natural limit to which effects these theories can capture since multiple scattering will occur when surface perturbations get more prominent, and the famous Rayleigh criterion is no longer fulfilled.

The best approach to overcome these issues and still keep a simple formulation seems to be the recent Generalized Harvey-Shack (GHS) method [3], which is based on the principle of non-paraxial scalar diffraction theory [4], that significantly improves scattering predictions for large angle scattering. Harvey has at several occasions [3, 5–7] indicated its superior predictions compared to paraxial scalar diffraction theory and older single scattering rough surface theories. This also includes results showing good fit with experimental data for moderately rough surfaces.

When the rough surface calculation in itself is a part of a larger model [8–12] and results are needed within reasonable time, GHS is therefore a good approach due to its accuracy and low computational cost compared to simple scalar methods like Rayleigh-Rice (low accuracy, no energy conservation) and full wave methods (high computational cost). A sketch of such a model is shown in Figure 1(a). In the paper it will be shown how GHS can be implemented for use in the radiative transfer model when solved using the so-called Discrete Ordinates Method (DOM) [9, 13, 14]. This enables GHS to be used efficiently when for example determining reflection from turbid media in slab geometries with rough surfaces. Furthermore, this approach seems more suitable for implementing the GHS method in general.

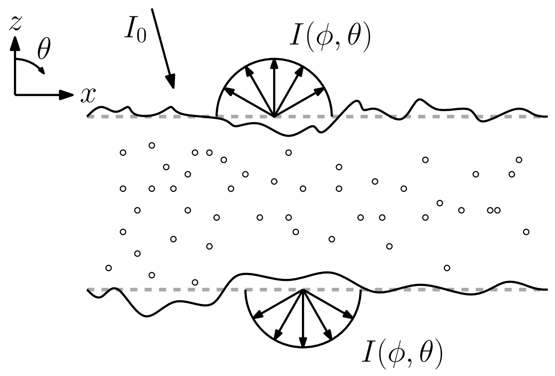
A secondary purpose of the paper is to present a minimalistic, ready-to-use implementation of the GHS method, in order to make it accessible for researchers in need of a scattering model for rough surfaces. An implementation of the GHS method is therefore given in Appendix A.

The rest of the paper is organized as follows. Section 2 introduces notation that will be used subsequently, Section 3 gives a short description of the DOM and radiative transfer, Section 4 introduces the necessary knowledge on rough surfaces needed for the GHS method, and Section 5 states all the relevant equations needed to implement GHS. After these recapitulations follows Section 6 which reformulates GHS in polar coordinates and Fourier series, thus having prepared it for DOM, Section 7 which explains and comments on the Python implementation of the GHS method presented in Appendix A, and Section 8 which concludes on the reformulation and implementation.

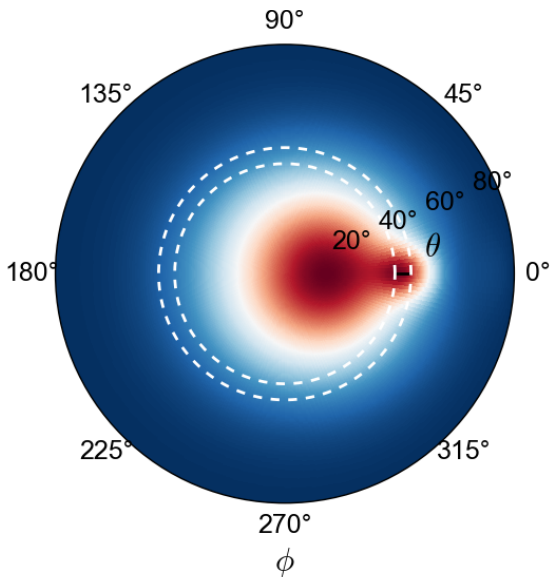
2. Notation

To be consistent with similar writings [6], all variables subscripted i refer to the incident angle, a subscript of o refers to the specular reflection angle (meaning $\theta_i =$

* Corresponding author: vejo@mek.dtu.dk



(a)



(b)

Fig. 1. (color online) (a) Example of a slab geometry with rough surfaces and a scattering/turbid medium in between (here depicted by particles). This model is for example used to represent human tissue [9], the ocean [15] and paint [16]. (b) Example of reflected intensity from such a system. Under the assumption that the rough surfaces as well as the turbid medium scatters isotropically, the corresponding reflection will be symmetric in the ϕ-direction. This is shown here by dashed lines indicating constant ϕ circles as well as a black bar showing the symmetry line.

$-\theta_o$), and s refers to scattered angles. Furthermore, a hatted variable means that it is scaled with respect to the electromagnetic wavelength of interest, λ. For example $\hat{x} = x/\lambda$, $\hat{\sigma} = \sigma/\lambda$. The Fourier transform operator $\mathcal{F}$ is defined as

$$\mathcal{F}\left\{H(\hat{x},\hat{y})\right\}(\alpha,\beta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} H(\hat{x},\hat{y})e^{-2\pi i(\hat{x}\alpha+\hat{y}\beta)}\mathrm{d}\hat{x}\mathrm{d}\hat{y}. \tag{1}$$

In this text $\hat{x},\hat{y}$ are always the non-transformed variables. In many cases some variables are uniquely defined by others, and not all variables are therefore written as independent variables of a function. This is for example the case where $\gamma_i = -\gamma_o$, $\gamma = \cos\theta$ or the condition in (12).

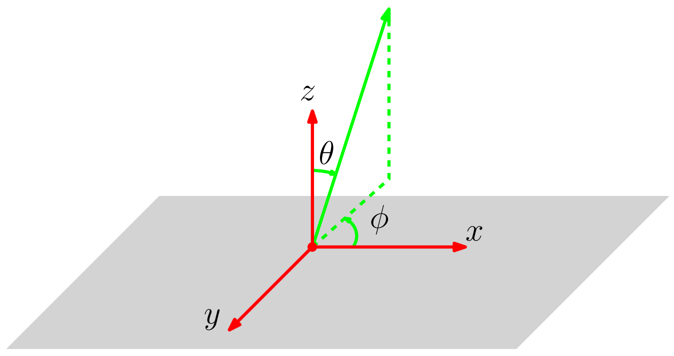


Fig. 2. (color online) Coordinate system used in this text.

3. Radiative transfer for slab geometries

The source free, isotropic radiative transfer equation (RTE) for a slab geometry as seen in Figure 1(a) describes the angular redistribution/transfer of radiance u with respect to the optical depth $\tau \propto z$ in a turbid/scattering medium described by the phase function p such that

$$-\mu\frac{\mathrm{d}u(\tau,\mu,\phi)}{\mathrm{d}\tau} = -u(\tau,\mu,\phi) + \frac{\Omega}{4\pi}\int_{-1}^{1}\int_{0}^{2\pi} p(\mu',\phi',\mu,\phi)u(\tau,\mu',\phi')\mu'\mathrm{d}\mu'\mathrm{d}\phi', \tag{2}$$

where $\mu = \cos\theta$, ϕ is as defined in Figure 2 and Ω is the so-called albedo. This equation can for example be used to describe light scattering of paint or skin. To cover the effect of the rough boundaries in Figure 1(a), the boundary conditions for the RTE is normally modified to take this into account [9, 17, 18]. Whereas rough surfaces for e.g. ocean analysis have received much attention [17, 18], it is hard to find implementations for optically rough surfaces [9]. The author has not been able to find any implementation that can be based on actual roughness measurements and which preserves a collimated beam component in case of low roughness. The GHS method satisfies both criteria, and can therefore be useful as a rough boundary condition for these cases.

3.A. The Discrete Ordinates Method

The Discrete Ordinates Method (DOM) was introduced by Chandrasekhar [13, 19] to solve the RTE for plane-parallel/slab geometries. The method has later been modified for efficient, stable numerical implementations [14, 20] and also applied to problems of higher dimensionality [19].

DOM discretizes μ in its Gauss points (ordinates) to facilitate numerical integration [13]. By defining the Gauss points x_i and their corresponding weights w_i, integrals can be approximated by Gaussian quadrature:

$$\int_{-1}^{1} f(x) \approx \sum_{i=1}^{N} w_i f(x_i). \tag{3}$$

Most modern math-oriented programming languages contain methods for calculating Gauss points and their corresponding weights. The integration rule can of course be scaled.

At the same time, the radiance is linearized using a Fourier series expansion in the ϕ direction to express the solution. For a 2π periodic function f that can be expressed as a Fourier series it holds that

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}(a_n \cos nx + b_n \sin nx), \tag{4}$$

where

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx, \quad n = 0, 1, 2, \ldots, \tag{5}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin nx, \quad n = 1, 2, 3, \ldots. \tag{6}$$

In order to obtain a finite linear system, the summation is truncated at a certain order N. This is justified since, if the absolute squared integral of the derivative of f is finite, the error rolls off with at least a factor of $1/\sqrt{N}$ [21, p. 127].

For isotropic cases, the solution is symmetric in ϕ with respect to the plane of incidence, which means that $b_n = 0\ \forall n \in \mathbb{N}$. The symmetri is sketched in Figure 1(b). To be able to impose the rough surface boundary conditions, it is necessary to discretize the GHS method in the same way.

4. Isotropic rough surfaces

All rough surfaces described by the GHS method are assumed to be isotropic and to have a Gaussian surface height distribution (with a zero mean height) [6]. This means that a root-mean-square (rms) surface roughness σ_s can be associated with the roughness properties of the surface, since it is then the standard deviation of the height distribution. Furthermore, a so-called surface autocovariance (ACV) function can be ascribed to the surface [6]. The ACV does not need to have a Gaussian distribution. Since not all features contribute to light scattering (like scratches or other surface "finish" effects), the ACV is also used to calculate the so-called relevant roughness as described below, which corrects for this effect.

4.A. Relative roughness

Considering the surface power spectral density (PSD) function, which is the Fourier transform of the ACV, it is found that some of these spatial frequencies will only contribute to evanescent modes [6], which do not contain any energy. This happens when the spatial frequencies are greater than $1/\lambda$ [6], and for a given incident angle θ_i, the relevant rms surface roughness is therefore limited to

$$\sigma_{rel} = \sqrt{\int_{-1/\lambda+f_o}^{1/\lambda+f_o} \int_{-\sqrt{1/\lambda^2-(f_x-f_o)^2}}^{\sqrt{1/\lambda^2-(f_x-f_o)^2}} PSD(f_x, f_y)\mathrm{d}f_y\mathrm{d}f_x}, \tag{7}$$

where

$$f_o = \frac{-\sin\theta_i}{\lambda}. \tag{8}$$

The above expressions can be understood as integrating the PSD in a disc with radius $1/\lambda$ centered at f_o instead of integrating over the whole f_x, f_y-plane. This ensures that only roughness that contributes to scattering is considered. A more elaborate description is found in [6] and [3].

4.B. Rough surface with Gaussian ACV

As a verification example for the implementation, a rough surface described by a Gaussian ACV is used so that the results can be compared against a reference [6].The ACV is given by

$$C_s(\hat{x}, \hat{y}) = \hat{\sigma}_s^2 e^{-\hat{r}^2/\ell_c^2}, \tag{9}$$

where $\hat{r} = \sqrt{\hat{x}^2 + \hat{y}^2}$ and ℓ_c is the correlation length of the surface [6]. The Fourier transform of this circular symmetric function can be looked up in most books or web pages treating two-dimensional Fourier transforms, and the result is that

$$PSD(f_r) = \pi\ell_c^2\sigma_s^2 e^{-(\pi\ell_c f_r)^2}, \tag{10}$$

where $f_r = \sqrt{f_x^2 + f_y^2}$.

5. Overview of the GHS method

Before deriving the GHS method for polar coordinates, the relevant equations for calculating light reflection from isotropic rough surfaces will be stated [6]. A thorough treatment of GHS for rough surfaces is found in [3] and [6] alongside few validation examples on experimental data. A numerical validation of the method is available in [7].

Defining a traditional spherical coordinate system as sketched in Figure 2 where $\phi \in [0, 2\pi[$, $\theta \in [0, \pi[$, the so-called direction cosines can be defined as

$$\alpha = \sin\theta\cos\phi, \quad \beta = \sin\theta\sin\phi, \quad \gamma = \cos\theta. \tag{11}$$

Note that this description is overdetermined since three variables are used to describe two variables – for example, this implies that

$$\gamma = \mathrm{sign}(\pi/2 - \theta)\sqrt{1 - \alpha^2 + \beta^2}, \tag{12}$$

where $\mathrm{sign}(x)$ is 1 for $x \geq 0$ and -1 otherwise.

Having defined the coordinate system, it is now possible to describe the reflected or transmitted intensity using GHS. For simplicity only reflection will be considered here, and in [3, 6] the few considerations needed for calculating transmittance are described. The reflected intensity is given as [6]

$$I(\theta_i,\phi_i,\theta_s,\phi_s,\lambda) = \cos(\theta_s) BRDF(\theta_i,\phi_i,\theta_s,\phi_s,\lambda), \tag{13}$$

where $BRDF$ is the so-called Bidirectional Reflection Distribution Function (BRDF), which describes the distribution of reflected radiance for light incident at (θ_i,ϕ_i). The BRDF generated by the rough surface is then described by [6]

$$BRDF(\theta_i,\phi_i,\theta_s,\phi_s,\lambda) = R(\theta_i)\ ASF(\theta_i,\phi_i,\theta_s,\phi_s)|_\lambda\,, \tag{14}$$

where R is Fresnel reflection calculated for a given incident angle, and ASF is the so-called angle spread function. The ASF can be represented using direction cosines, $ASF(\alpha_i,\beta_i,\alpha_s,\beta_s)$. Without loosing generality $\alpha_i = 0$ is assumed. This is legal due to the isotropic surface assumption that makes the reflection from the rough surface indifferent to rotation in the ϕ direction. The ASF is then defined as [6]

$$\begin{aligned} ASF(\beta_i,\alpha_s,\beta_s) = {} & A(\gamma_i,\gamma_o)\delta(\alpha_s,\beta_s-\beta_o) \\ & + K(\gamma_i)S(\beta_i,\alpha_s,\beta_s). \end{aligned} \tag{15}$$

Here, A describes the energy left in the specular reflection and is given as

$$A(\gamma_i,\gamma_s) = e^{-[2\pi(n_1\gamma_i - n_2\gamma_s)\hat{\sigma}_{rel}]^2}, \tag{16}$$

where $\hat{\sigma}_{rel} = \sigma_{rel}/\lambda$ was calculated in the unscaled version in Section 4.A and n_1, n_2 are the refractive indices of the material from the background medium and the rough surface material respectively. For reflection $n_2 := -n_1$. The S describes the distribution of the scattered radiance and is given as

$$S(\beta_i,\alpha_s,\beta_s) = B(\gamma_i,\gamma_s)\mathcal{F}\left\{G(\hat{x},\hat{y},\gamma_i,\gamma_s)e^{-i2\pi\beta_o\hat{y}}\right\}, \tag{17}$$

where

$$B(\gamma_i,\gamma_s) = 1 - A(\gamma_i,\gamma_s) \tag{18}$$

and

$$G(\hat{x},\hat{y},\gamma_i,\gamma_s) = \frac{e^{[2\pi(n_1\gamma_i-n_2\gamma_s)\sigma_{rel}/\sigma_s]^2 C_s(\hat{x},\hat{y})} - 1}{e^{[2\pi(n_1\gamma_i-n_2\gamma_s)\hat{\sigma}_{rel}]^2} - 1}, \tag{19}$$

where C_s is the ACV function of the rough surface. A concrete choice of ACV for the later implementation was introduced in Section 4.B.

K in (15) is used to normalize S by assuming no energy is lost in evanescent modes, and also making sure that energy not left in the specular reflection is in the scattered term, and is therefore defined as

$$K(\gamma_i) = B(\gamma_i,\gamma_o)\left(\int_{-1}^{1}\int_{-\sqrt{1-\alpha^2}}^{\sqrt{1-\alpha^2}} S(\beta_i,\alpha,\beta)\mathrm{d}\alpha\mathrm{d}\beta\right)^{-1}, \tag{20}$$

where the integral is equivalent to integrating over the hemisphere $\phi \in [0, 2\pi[$, $\theta \in [0, \pi/2[$.

5.A. Calculating S using translated Hankel transforms

For normal incidence, $\beta_i = 0$, S is a circular symmetric (that is, constant wrt. ϕ) function due to the isotropy assumption. This means that the Fourier transform in (17) can be calculated efficiently by applying the Hankel transform of order zero, and this feature is preferable to retain. Therefore, S_{rad} is defined such that

$$S_{rad}(\beta_i,\alpha_s,\beta_s) = B(\gamma_i,\gamma_s)\mathcal{F}\left\{G(\hat{x},\hat{y},\gamma_i,\gamma_s)\right\}, \tag{21}$$

which can then be calculated efficiently using a Hankel transform. Details are shown in the implementation. Furthermore, the modulation (exponential function) in (17) corresponds to a translation in the transformed domain, which means S can be expressed as

$$S(\beta_i,\alpha_s,\beta_s) = S_{rad}(\beta_i,\alpha_s,\beta_s-\beta_o). \tag{22}$$

This will later on make it easy to calculate S, which is best seen in the actual implementation in Appendix A. By this we have a complete and efficient description of the BRDF of the rough surface. The task is then to convert the method such that it is represented by its Fourier components in a μ,ϕ coordinate system suitable for DOM.

6. Reformulating GHS for DOM

Reformulating the GHS method in a μ,ϕ coordinate system gives cumbersome expressions, and a $\rho = \sin\theta, \phi$ system is preferred. This also places $\theta = 0°$ at the origin, which is more intuitive. In a discretized version, this is not a problem, as long as all point ρ_i are chosen to match a Gauss point μ_i. Integrals still have to be reformulated to fit the Gauss points of μ, which will also be shown in the following.

The extra step is needed to comply with DOM, but *if the calculations are not to be used with DOM, the author recommends to discretize ρ in its Gauss points instead to get a simpler implementation.*

6.A. Updating coordinates and translation

In this section the problem is reformulated in terms of ρ,θ. To begin with, note that

$$\beta_o = \sin\theta_o = \rho_o, \quad \text{for } \phi = \pi/2, \tag{23}$$

so that β_o-translation directly can be interchanged with ρ_o-translation along the $\phi = \pi/2$ line. (This would not have been possible if choosing μ). A Cartesian translation of the radial component in a polar coordinate system is done by updating the radial component to

$$\rho' = \sqrt{\rho^2 + \rho_o^2 - 2\rho\rho_o \cos\phi}, \tag{24}$$

where ρ_o is the length of the translation and ρ' is the updated variable. This transformation also involves ϕ, since such a translation naturally breaks the circular symmetry (for all but the constant case).

The relation between the direction cosines and ρ, ϕ are given by

$$\alpha = \rho\cos\phi, \quad \beta = \rho\sin\phi, \quad \gamma = \text{sign}(\rho)\sqrt{1-\rho^2}. \tag{25}$$

If the changes are introduced in (21), then it is easy to calculate

$$S_r(\rho_i, \rho_s) := S_{rad}\left(\beta_i = \rho_i, \alpha_s = 0, \beta_s = \rho_s\right) \tag{26}$$

Note that there is no dependence of ϕ_s for S_r due to the isotropic scattering, and it is therefore chosen to put $\phi_s = \pi/2$. This then means that S can be expressed as

$$S(\rho_i, \rho_s, \phi_s) = S_r\left(\rho_i, \sqrt{\rho_s^2 + \rho_o^2 - 2\rho_s\rho_o\cos\phi_s}\right). \tag{27}$$

6.B. Fourier series expansion

Due to DOM, the solution has to be expanded into its Fourier series components. From Section 3 it is found that the b_n's all are zero. Expanding the Fourier series, it can be written as

$$S(\rho_i, \rho_s, \phi_s) = \frac{1}{2}s_0(\rho_i, \rho_s) + \sum_{k=1}^{\infty} s_k(\rho_i, \rho_s)\cos k\phi_s, \tag{28}$$

where

$$s_k(\rho_i, \rho_s) = \frac{2}{\pi}\int_0^{\pi} S(\rho_i, \rho_s, \phi)\cos k\phi \mathrm{d}\phi, \tag{29}$$

where the integral will be solved numerically by sampling ϕ at Gauss points and then use the Gaussian quadrature rule as described in Section 3.

6.C. Calculating K

For calculating K, the integral defined in (20) now needs to be converted to the same coordinate system. The Jacobian associated with shifting to polar coordinates is given by

$$\begin{aligned} J &= \frac{\partial\alpha}{\partial\rho}\frac{\partial\beta}{\partial\phi} - \frac{\partial\alpha}{\partial\phi}\frac{\partial\beta}{\partial\rho} = \phi\sin\theta\cos^2\phi + \sin\theta\sin^2\phi \\ &= \sin\theta = \rho, \end{aligned} \tag{30}$$

which means that K can be calculated as

$$\begin{aligned} K(\gamma_i) &= B(\gamma_i, \gamma_o)\left(\int_0^1\int_0^{2\pi} S(\rho_i, \rho, \phi_s)\rho \mathrm{d}\phi \mathrm{d}\rho\right)^{-1} \\ &= B(\gamma_i, \gamma_o)\left(\pi\int_0^1 s_0(\rho_i, \rho)\rho \mathrm{d}\rho\right)^{-1}. \end{aligned} \tag{31}$$

The last equation is obtained since

$$\int_0^{2\pi} s_k(\rho_i, \rho_s)\cos k\phi_s = 0 \quad \text{for } k = 1, 2, \ldots. \tag{32}$$

As is known from DOM, information on the energy of S is stored in the zero order term, whereas all higher order terms only are used to determine the shape of the reflection.

6.D. Calculating K by μ integration

For the DOM purpose, it is necessary to integrate over μ instead of ρ. Remembering that $\rho = \sqrt{1-\mu^2}$ the Jacobian for this coordinate shift is

$$J = \frac{\partial\rho}{\partial\mu} = -\frac{\mu}{\sqrt{1-\mu^2}}. \tag{33}$$

This means that (31) can be expressed as

$$\begin{aligned} K(\gamma_i) &= B(\gamma_i, \gamma_o)\left(\pi\int_0^1 s_0(\rho_i, \sqrt{1-\mu^2})\rho\frac{\mu}{\sqrt{1-\mu^2}}\mathrm{d}\mu\right)^{-1} \\ &= B(\gamma_i, \gamma_o)\left(\pi\int_0^1 s_0(\rho_i, \sqrt{1-\mu^2})\mu \mathrm{d}\mu\right)^{-1}. \end{aligned} \tag{34}$$

6.E. Calculating relative roughness in polar coordinates

Even though this is not strictly necessary, it is convenient to convert the integral of σ_{rel} in (7) to an integral in polar coordinates, since the same principle then can be applied. The integral in (7) corresponds to the integral of a disc with radius $1/\lambda$ of the PSD translated by f_o, and can therefore be calculated – using the same principle as when calculating K – as

$$\sigma_{rel} = \sqrt{\int_0^{1/\lambda}\int_0^{2\pi} PSD_r(f_r, \phi_r, f_o)f_r \mathrm{d}\phi_r \mathrm{d}f_r}, \tag{35}$$

where

$$PSD_r(f_r, \phi_r, f_o) = PSD\left(\sqrt{f_r^2 + f_o^2 - 2f_rf_o\cos(\phi_r)}\right). \tag{36}$$

This finishes all reformulation needed for implementation, and the s_k's which are necessary coefficients for the DOM are now found. To ease implementation for others and to test the above reformulation, a numerical implementation is presented in the follwing.

7. Implementation

In Appendix A, a Python implementation of the above method is given. To run the program, save the code in Appendix A.1 and A.2 as, respectively, `cosineHarveyShack.py` and `HankelTransformSimple.py` in the same folder, and run `cosineHarveyShack.py` using Python. In

Linux the command would be

```
python cosineHarveyShack.py
```

and then one of the results shown in Figure 3(a) will appear (with less formating). The code requires `numpy` and `matplotlib` installed. Older versions of `numpy` does not contain the function `leggauss` to find Gauss points and weights, in this case the code tries to fall back on a `scipy` implementation of the same function, which requires that `scipy` is also installed. The code has been tested under Ubuntu Linux 14.04 64-bit in Python version 2.7.6 and 3.4.0, and Windows 7 64-bit using Anaconda 2.0.1. Since Anaconda and other Python implementations are available for OS X as well, the software is expected to run on Mac.

The code is written to be compact, straight-forward to read and ready to use and extend. This means that a few performance tweaks have been left out due to simplicity, where the most notable is lack of Fresnel reflection and efficient calculation of the Hankel transform, which will be explained in Section 7.C. In the following sections, the implementation of the code is explained.

7.A. A note on `numpy` matrix multiplication for Matlab users

For Matlab users inexperienced with `numpy` for Python, it is worth noticing that Python array/matrix indexing starts at 0 instead of 1. Furthermore there is no distinct `*` and `.*` operator. Instead, `*` means element-wise multiplication if the operands are arrays and matrix multiplication if the operands are matrices. In case the arrays have different (but compatible) shapes, the result is a higher order array. Since all `numpy` data structures in this code are initialized as arrays, `*` and `/` are always performed element-wise.

7.B. The main program explained

line 1–10 Import of modules: `numpy` for calculation, `matplotlib` for visualization. A function, `leggauss` to calculate Gauss points and weights from either `numpy` or `scipy`. Note that `HankelLibSimple` is the custom module in Appendix A.2.

line 11–23 `main()` is the main routine which is run when the program is run – see line 112–113. Input parameters are defined here:

`lam`	λ, wavelength;
`sigmas`	σ_s, rms roughness;
`lc`	ℓ_c, roughness correlation length;
`thetai`	θ_i, incident angle;
`n1`	n_1, refractive index for material 1;
`N`	N, number of Gauss points for μ;
`Nf`	N_f, number of Fourier expansion terms.

line 24–30 Discretization of the coordinate system in Gauss points. `phi` is transposed such that when multiplied by `rho` or `mu` a matrix multiplication will be returned.

line 31–36 Calculation of σ_{rel} based on (7) – using the Gauss quadrature rule for integration.

line 37–47 Precalculation and initialization of variables for later use. Note in particular that `cosk` is a matrix containing $\cos(k\phi)$ for all discretized points of k and ϕ. This is used for the Fourier expansion calculations.

line 48–51 Setup of the Hankel transform class for repeated use in the following loop.

line 52–61 First part of loop over ρ_s, since (21) and (27) dictates a new Hankel transform for every value of ρ_s (but not ϕ_s). Note how `Cs` in line 58 is evaluated using the radial coordinate system defined by the `HankelTransform` class. `fht` is short for *forward Hankel transform.*

line 62–68 Last part of loop over ρ_s. Here `Srad` (S_r) is interpolated onto the relevant part of `rhom` (ρ'). Since the interpolation requires monotone, increasing values, the coordinates are first sorted, then interpolated, and then sorted back. In line 67 `sk` is calculated from (29) by the Gauss quadrature rule.

line 69–72 Calculation of K using (20), and B using (18).

line 73–86 Evaluation of S in the plane of incident light using (28) (where $\phi = \pi$ is needed for negative angles) followed by a construction of the angular axis, conversion to intensity using (13), and then plotting.

line 87–98 Evaluation and plotting of intensities on the whole upper hemisphere using same principle as above.

line 99–104 Function for calculating N Gauss points and weights on the interval $[a, b]$.

line 105–111 Function for calculating G using (19), where an approximation is used for large exponents to avoid overflow.

line 112–113 This ensures that `main()` is executed when the code is run.

7.C. Hankel transform

The Hankel transform class is a minimalistic implementation of the matlab code from [22] which is based on the work found in [23]. For completeness, both a forward and inverse Hankel transform method is given even though the latter is never used. The math behind the code is explained in [22, 23] – therefore only the usage is explained here.

line 1–9 Credits to the original paper, load of modules and special functions: `jn` is the Bessel function of order n, and `jn_zeros` returns the zeros of a Bessel function of order n.

line 10–13 Beginning of the `HankelTransform` class constructor which takes the following arguments

`Rmax`	max domain value for untransformed axis;
`order`	the order of the Hankel transform;
`N`	number of (equidistant) discretization points.

line 14–16 Calculation of zeros for a Bessel function of order `order`. Calculation of the discretization of the radius axis in untransformed domain and similar for the transformed (“frequency”) domain. The domains are

stored so that they can be fetched by the user afterwards – see line 49 and onwards in the main file for an example of use.

line 17–24 Calculation of relevant matrices for the conversion, see [22, 23] for a derivation of the results.

line 25–27 A function that returns the Hankel transform of order `order` of the function `f` given as an array calculated in the radius array points from line 14.

line 28–29 A function that returns the inverse Hankel transform of order `order` of the function `F` given as an array calculated in the frequency array points from line 15.

7.D. Modification and comments

Since the implementation is kept simple, a few comments on how to modify the code for speed or usage are given here.

First of all, the Fresnel reflection from equation (14) is left out for simplicity. This is easily implemented by first calculating the Fresnel reflection for a given input angle and n_1, n_2 and afterwards multiply this with K just after line 70. Secondly, for repeated use, there is no reason to re-initialize the `HankelTransform` class more than once. Also, the Bessel function zeros calculated in the `HankelTransform` class can be pre-calculated and stored in a `numpy` (or matlab) data file, which is also shown in [22]. Other sizes that only should be calculated once for repeated use include `phi`, `wphi`, `mu`, `wmu`, `rho`. All in all, the code is executed fast, and without visualization (but with initialization of all previous mentioned variables) a runtime of less than 0.5 s should be expected on a normal desktop computer.

For ACV's with a larger dynamic range than a Gaussian ACV, it is proposed [6] to use FFTlog or similar to obtain good precision.

The author finds that for a general implementation of the GHS method, a ρ, ϕ coordinate system is more appropriate than the current Cartesian discretization in the α, β plane [6, 24]. See for example [24, Fig. 2] where only $\sim$ 20% of the computational domain contains information, since the rest corresponds to "intensities" of complex angles ($\sqrt{\alpha^2 + \beta^2} > 1$). Furthermore it makes circular integrals more precise. For these reasons, a decrease in CPU time and memory consumption can be expected.

7.E. Verification

To verify the code, results from [6] for a rough Gaussian surface with Gaussian ACV was used. The number of Fourier series terms was put to 20, and μ was discretized in 100 points. The result is presented in Figure 3(a) and is shown to match. This confirms that the code is implemented correctly. The result for 40° scattering angle in the complete ρ, ϕ plane is presented in Figure 3(f).

It can in general be expected that the narrower the peak of the scattered energy is, the more Fourier terms will be needed to accurately resolve the shape of the response, since higher frequencies are required. This is illustrated in Figure 3(b), where the same result with $\theta_i = 40°$ is calculated for $N_f = 2, 5, 10, 15, 99$. Some of their corresponding ρ, ϕ plots are seen in Figure 3(c-f).From this it can be seen that using $N_f = 2$ gives a negative (hence infeasible) solution in some areas since the best approximation using only one cosine cannot counteract that effect. Using $N_f = 5$ still gives negative contributions even though that is not seen in the specular plot in Figure 3(b). Already at $N_f = 10$ the result resembles the reference ($N_f = 99$) quite well, and for $N_f = 15$, the result is not distinguishable from the reference solution.

8. Conclusion

The paper presents a method for calculating rough surface reflectance based on the General Harvey-Shack (GHS) method discretized using the Discrete Ordinates Method (DOM). The implementation is seen to reproduce results from [6]. Since DOM requires a polar coordinate system, a such is proposed and implemented in a way that exploits the isotropy assumption used for the rough surfaces under consideration. This makes it possible to efficiently use the GHS rough surface scattering method for radiative transfer problems. Furthermore it is shown how it is possible to keep the desired shift-invariance property from the direction cosine space in this new setting. It is also shown how the proposed discretization makes use of all data point in the discretized space as opposed to the approaches in [6, 24].

A Python implementation is presented in the appendix of the paper with the purpose of passing on a ready-to-use code in a freely available programming language, which hopefully will make the GHS method as accessible as possible, so that researchers can benefit from its improved accuracy compared to many other single scattering theories.

Acknowledgments

The author is grateful for the support from the Danish National Technology Foundation through the ODAAS project.

References

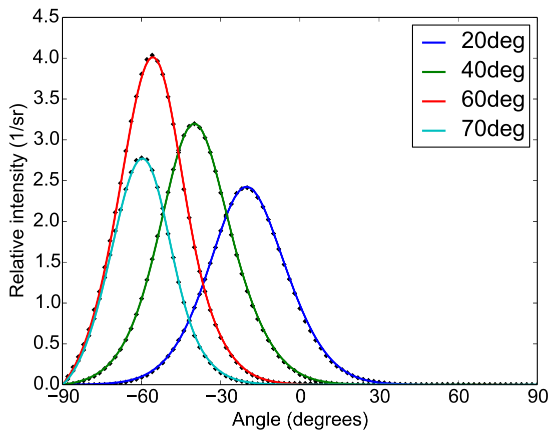
[1] I. Simonsen., Optics of surface disordered systems. *The European Physical Journal Special Topics*, 181(1):1–103, June 2010.

[2] JR. Colin et al., *Light Scattering and Nanoscale Surface Roughness*. Springer, 2007.

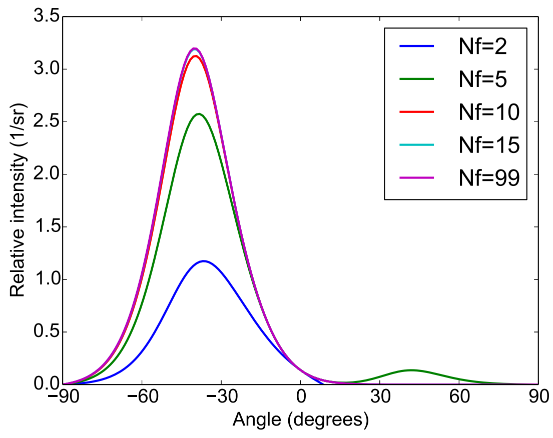
[3] JE. Harvey et al. Scattering from Moderately Rough Interfaces between two arbitrary media. 7794(1951):77940V–77940V–11, August 2010.

[4] JE. Harvey et al., Diffracted radiance: a fundamental quantity in nonparaxial scalar diffraction theory. *Applied optics*, 38(31):6469–81, November 1999.

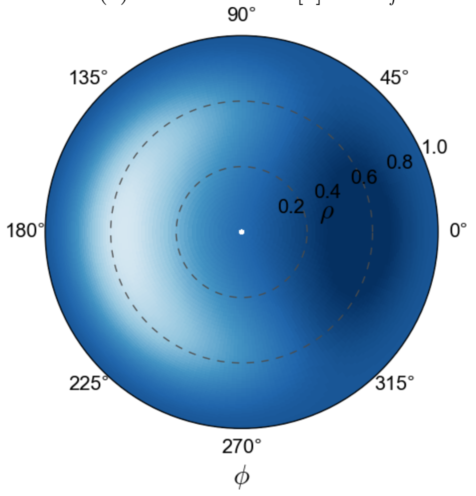
[5] S. Schröder et al., Modeling of light scattering in different regimes of surface roughness. *Optics express*, 19(10):9820–35, May 2011.



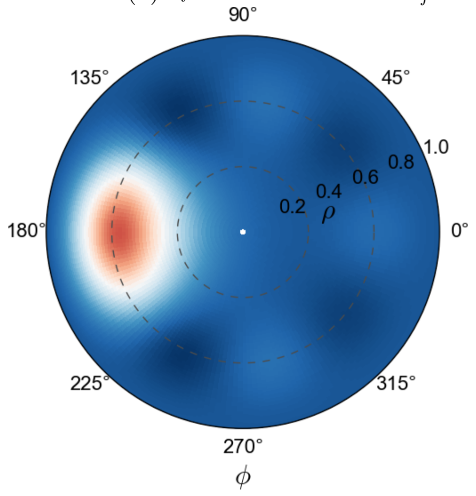
(a) Verification of [6] for $N_f = 20$



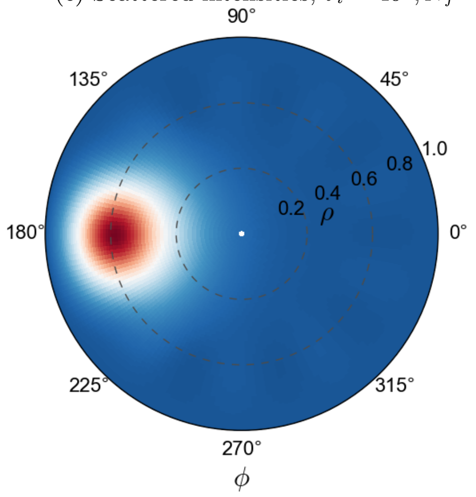
(b) $\theta_i = 40°$ for different N_f



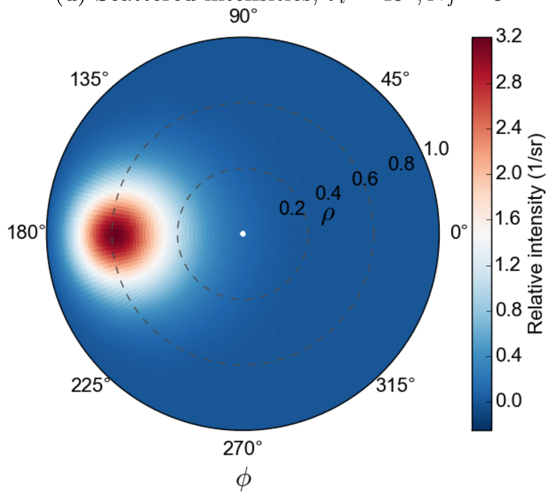
(c) Scattered intensities, $\theta_i = 40°, N_f = 2$



(d) Scattered intensities, $\theta_i = 40°, N_f = 5$



(e) Scattered intensities, $\theta_i = 40°, N_f = 10$



(f) Scattered intensities, $\theta_i = 40°, N_f = 15$

Fig. 3. (color online) Results calculated from the code in Appendix A – all settings are the same except for θ_i and N_f. (a) verifies the implementation by reproducing the results from [6], which is indicated by blakc diamonds. Mismatch is due to error in the digitizing process. (b) shows the results of calculating the same example for different number of cosine terms, N_f. (c-f) shows the reflected intensities for the whole ρ, ϕ range.

[6] A. Krywonos et al., Linear systems formulation of scattering theory for rough surfaces with arbitrary incident and scattering angles. *JOSA A*, 28(6):1121–1138, 2011.

[7] N. Choi and JE. Harvey, Numerical validation of the generalized HarveyShack surface scatter theory. *Optical Engineering*, 52(11):115103, November 2013.

[8] S. Faÿ et al., Light trapping enhancement for thin-film silicon solar cells by roughness improvement of the ZnO front TCO. *16th EC Photovoltaic Solar Energy Conference*, pages 361–364, 2000.

[9] J. Stam, An illumination model for a skin layer bounded by rough surfaces. *Rendering Techniques 2001*, 2001.

[10] AB. Murphy, Modified KubelkaMunk model for calculation of the reflectance of coatings with optically-rough surfaces. *Journal of Physics D: Applied Physics*, 39(16):3571–3581, August 2006.

[11] S. Schröder et al., Scattering of Roughened TCO Films-Modeling and Measurement. *Optical Interference Coatings*, 1–3, 2010.

[12] M. Elias et al., Influence of interface roughness on surface and bulk scattering. 27(6):1265–1273, 2010.

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

[14] K. Stamnes et al., Numerically stable algorithm for discrete-ordinate-method radiative transfer in multiple scattering and emitting layered media. *App. Opt.*, 27(12):2502–9, June 1988.

[15] Z. Jin and K. Stamnes, Radiative transfer in nonuniformly refracting layered media: atmosphere-ocean system. *Applied Optics*, 33(3):431–42, January 1994.

[16] M. Elias, Physics, colour and art: a fruitful marriage. *JAIC-Journal of the International Colour Association*, pages 25–35, 2012.

[17] GE. Thomas, K. Stamnes, *Radiative Transfer in the Atmosphere and Ocean.* Cambridge University Press, 2002.

[18] Z. Jin et al., Modeling of light scattering from micro- and nanotextured surfaces. *Appl. Opt.*, 45(28):7443–55, October 2006.

[19] ST. Thynell, Discrete-ordinates method in radiative heat transfer. *International journal of engineering science*, 36:1651–75, 1998.

[20] WJ. Wiscombe, The delta-M method: Rapid yet accurate radiative flux calculations for strongly asymmetric phase functions. *Journal of the Atmospheric Sciences*, 34:1408–22, September 1977.

[21] O. Christensen, *Differentialligninger og uendelige rækker.* Institut for Matematik, Danmarks Tekniske Universitet, 2009.

[22] M. Guizar, Integer order Hankel transform `http://www.mathworks.com/matlabcentral/fileexchange/6570-integer-order-hankel-transform`, December 2004.

[23] M. Guizar-Sicairos and J- C. Gutiérrez-Vega, Computation of quasi-discrete Hankel transforms of integer order for propagating optical wave fields. *Journal of the Optical Society of America. A, Optics, image science, and vision*, 21(1):53–8, January 2004.

[24] D. Domine et al., Modeling of light scattering from micro- and nanotextured surfaces. *Journal of Applied Physics*, 107(4):044504–8, February 2010.

Appendix A: Code

Here follows the code for a Python program to implement the above GHS method for polar coordinates. The source files are also available by e-mailing the author.

1. cosineHarveyShack.py

The following is the main file for the program. Note that it is dependent on `HankelLibSimple.py`.

```python
#(c) Villads Egede Johansen, 2014, vejo@mek.dtu.dk
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt

from numpy import cos,sin,arcsin,exp,pi,sqrt,multiply
try:    from numpy.polynomial.legendre import leggauss
except: from scipy.special.orthogonal import p_roots as leggauss
from HankelLibSimple import HankelTransform

def main():
   #############################
   # INPUT PARAMETERS
   lam = 10.6e-6
   sigmas = 2.27e-6
   lc = 20.9e-6
   thetai = -40 /180*pi
   n1 = 1.0
   n2 = -n1

   N = 100
   Nf = 20

   #############################
   # SETUP
   phi,wphi = calcGauss(2*N,0,pi)
   phi = phi[np.newaxis].T
   mu,wmu = calcGauss(N,0,1)
   rho = sqrt(1-mu**2)[::-1]

   f, wf = calcGauss(99,0,1/lam)
   fo = -sin(thetai)/lam
   fm = sqrt(f**2+fo**2-2*f*fo*(cos(phi)))
   PSD = pi* lc**2 *sigmas**2 *exp(-(pi*lc*fm)**2)
   sigmarel = sqrt(( (PSD*f).dot(wf).dot(2*wphi)))

   sigmashat = sigmas/lam
   sigmarelhat = sigmarel/lam
   lchat = lc/lam
   gammai = cos(thetai)

   rhom = sqrt(rho**2+sin(thetai)**2
         +2*rho*sin(thetai)*(cos(phi)))

   cosk =cos(multiply(np.arange(Nf),phi))
   sk = np.zeros((Nf,N))

   #Setup Hankel transform class
   ht = HankelTransform(Rmax=50,order=0,N=201)
   vrad,rrad = ht.v, ht.r

   #############################
   # Calculations
   for n in range(N):
      gammas = mu[::-1][n]
      A = exp(-(2*pi*(n1*gammai-n2*gammas)*sigmarelhat)**2)
      B = 1-A
      Cs = sigmashat**2 *exp(-rrad**2/lchat**2)
      G = calcG(gammai,gammas,sigmarel,sigmarelhat,sigmas,Cs,n1,n2)
      Srad = ht.fht(G)

      _rhom = rhom[:,n]
      idxNew = np.argsort(_rhom)
      rhoOrdered = _rhom[idxNew]
      idxBack = np.argsort(idxNew)
      Stot = B*np.interp(rhoOrdered,vrad,Srad)[idxBack]
      sk[:,n] = 2/pi*multiply(Stot,cosk.T).dot(wphi)

   K = 1/(pi*sk[0][::-1] *mu).dot(wmu)
   B = 1-exp(-(2*pi*(n1*gammai-n2*gammai)*sigmarelhat)**2)
   K = B*K

   #############################
   # VISUALIZATION
   Slinepos = 0.5*sk[0] + sum([ sk[k]*cos(k*0)
                              for k in range(1,sk.shape[0])])
   Slineneg = 0.5*sk[0] + sum([ sk[k]*cos(k*pi)
                              for k in range(1,sk.shape[0])])
   angles = arcsin(rho)
   angles = np.concatenate([-angles[::-1],angles])
   I = cos(angles) *K*np.concatenate([Slineneg[::-1],Slinepos])

   plt.plot(angles/pi*180,I,lw=2)
   plt.xlim(-90,90), plt.ylim(0)
   plt.show()

   phifull = np.linspace(0,2*pi,2*N+1)[np.newaxis].T
   cosk =cos(multiply(np.arange(Nf),phifull))
   f = 0.5*sk[0] + sum([ multiply(sk[k,:],cosk[:,k][np.newaxis].T)
                       for k in range(1,sk.shape[0])])
   f = mu[::-1]*K*f

   ax = plt.subplot(1,1,1, projection="polar", aspect=1.)
   cax=ax.pcolormesh(phifull.ravel(), rho, f.T)
   plt.gcf().colorbar(cax)
   plt.show()


def calcGauss(N,a,b):
   p,w = leggauss(N)
   p = (b-a)/2.*p + (b+a)/2.
   w = (b-a)/2.*w
   return p,w

def calcG(gammai,gammas,sigmarel,sigmarelhat,sigmas,Cs,n1,n2):
   exponent1 = (2*pi*(n1*gammai-n2*gammas)*sigmarel/sigmas)**2 *Cs
   exponent2 = (2*pi*(n1*gammai-n2*gammas)*sigmarelhat)**2
   if exponent2 > 50:
      return exp(exponent1-exponent2)
   return (exp(exponent1)-1)/(exp(exponent2)-1)

if __name__ == "__main__":
   main()
```

2. HankelLibSimple.py

The following is a minimalistic implementation of a class used to calculate Hankel transforms. The code is based on the approach presented in [23].

```python
#(c) Villads Egede Johansen, 2014, vejo@mek.dtu.dk
#Port of matlab code by Manuel Guizar Sicairos based on
#  M. Guizar-Sicairos and J. C. Gutierrez-Vega, Computation of quasi-
     discrete
#  Hankel transforms of integer order for propagating optical wave fields,
#  J. Opt. Soc. Am. A 21, 53-58 (2004).
from __future__ import division
import numpy as np
from scipy.special import jn,jn_zeros

class HankelTransform:
   def __init__(self,Rmax,order=0,N=500):
      c = jn_zeros(order,N+1)

      self.r = c[:N] *Rmax/c[N]        #Radius vector
      self.v = c[:N] /(2*np.pi*Rmax)   #Frequency vector

      Jn,Jm = np.meshgrid(c[:N],c[:N])

      self._C = 2/c[N] *jn(order,Jn*Jm/c[N]) / (
            abs(jn(order+1,Jn)) * abs(jn(order+1,Jm)))

      self._m1 = abs(jn(order+1,c[:N]))/Rmax
      self._m2 = self._m1*Rmax/c[N] /(2*np.pi*Rmax)

   def fht(self,f):
      return np.dot(self._C,f/self._m1)*self._m2

   def iht(self,F):
      return np.dot(self._C,F/self._m2)*self._m1
```