Light Scattering by Individual and Groups of Spheroidal Particles

Diploma paper
by
Peter Alsholm

Lund Reports on Atomic Physics, LRAP-200
Lund, August 1996

Abstract

In order to simulate the light propagation in illuminated samples containing suspensions of spheroidal particles, taking into consideration size, shape, concentration and different orientations, a Monte Carlo computer program was developed. The program, based on an old Monte Carlo program, was able to treat samples of finite extension laterally, as well as in depth. The computations of the angular scattering of each particle, were made by separate programs based on the T-matrix theory. Modifications of existing programs for this purpose were performed, enabling scattering calculations for particles with larger size parameters. Simulations were made by using a sample with properties similar to human blood, *i.e.* a suspension of erythrocytes, approximated by oblate spheroids, as scattering particles. These results were then finally compared with measurements and results obtained by the original Monte Carlo program.

Contents

Abstract **3**
1. Introduction **7**
2. Theory **8**
2.1. Scattering from one particle 8
2.1.1. Electric time-harmonic fields 8
2.1.2. The T-matrix method 8
2.2. Multiple scattering 11
2.2.1. The original Monte Carlo method 11
3. Material and methods **13**
3.1. The T-matrix computer programs 13
3.1.1. The original computer program T1 13
3.1.2. Modifications of program T1 14
3.1.3. The original computer program T3 15
3.1.4. Modifications of program T1 16
3.2. The Monte Carlo computer programs 16
3.2.1. The original Monte Carlo program 16
3.2.2. Modifications due to boundary conditions 17
3.2.3. Implementation of specific scattering objects 17
3.3. Testing program MCMLBOUN, due to boundary conditions 18
3.4. Testing limitations and accuracy 18
3.5. Evaluations and tests of the programs EXTENDT3 and TMMCML 19
3.6. Comparison with the original Monte Carlo model 23
4. Results **24**
4.1. Testing program MCMLBOUN, due to boundary conditions 24
4.2. Testing limitations and accuracy 25
4.3. Evaluations and tests of the programs EXTENDT3 and TMMCML 27
4.4. Comparison with the original Monte Carlo model 39
5. Discussion and conclusions **43**
5.1. Comments to the tests and evaluations 43
5.1.1. Testing program MCMLBOUN, due to boundary conditions 43
5.1.2. Testing limitations and accuracy 43
5.1.3. Evaluations and tests of the programs EXTENDT3 and TMMCML 43
5.1.4. Comparison with the original Monte Carlo model 46
5.2. Summary and concluding remarks 46
5.3. Future 47
6. Acknowledgements **48**
7. References **49**
Appendix A: The T-matrix computer programs **52**
A.1. Computer code listing including comments 52
A.1.1. EXTENDT1 52
A.1.2. EXTENDT3 65
A.1.3. MCDATA 72
A.2. Flow charts 80
A.2.1. EXTENDT1 80
A.2.2. EXTENDT3 81
A.2.3. MCDATA 82

Appendix B: The Monte Carlo computer programs 83
B.1. Computer code listing 83
B.1.1. MCMLBOUN 83
B.1.1.1. mcml.h 83
B.1.1.2. mcmlmain.c 85
B.1.1.3. mcmlio.c 87
B.1.1.4. mcmlgo.c 99
B.1.2. TMMCML 109
B.1.2.1. mcml.h 109
B.1.2.2. mcmlmain.c 111
B.1.2.3. mcmlio.c 113
B.1.2.4. mcmlgo.c 127
B.2. List of specific modifications 138
B.2.1. Modifications from MCML to MCMLBOUN 138
B.2.2. Modifications from MCMLBOUN to TMMCML 139
B.3. Flow charts 140
Appendix C: How to run the programs 144
C.1. EXTENDT1 144
C.2. EXTENDT3 144
C.3. MCDATA 144
C.4. MCMLBOUN 145
C.5. TMMCML 145
Appendix D: Special functions 147
D.1. Bessel and Hankel functions 147
D.2. Associated Legendre functions 147

1. Introduction

The knowledge of light interacting with biological tissue is important when developing new diagnostic methods and medical treatments utilizing lasers. The more these methods are refined, the more detailed studies of the propagation of light in tissue are required. In diagnosing diseases such as cancer and arteriosclerosis, using lasers, the fluorescence, inelastic or elastic scattered light, transmitted through or reflected by the tissue is analyzed. Distinctions in the obtained spectra originate from deviations in optical properties, which makes it possible to differentiate between healthy and malignant tissue. When laser light is used to treat medical disorders, *e.g.* in obliteration of port wine stains or cancer cell destruction using photodynamic therapy (PDT)[1], the energy of the laser light is transformed into heat energy or induces chemical reactions. In order to optimize the results of these treatments, the understanding of light propagation properties in tissue is essential.

When light propagates, two processes - absorption and scattering - occur to various extents due to the medium. A common way of describing the optical properties of the medium is by the absorption coefficient, μ_a, the scattering coefficient, μ_s, and the anisotropy coefficient, g. The absorption and scattering coefficients describe the probability per unit path length of a photon being absorbed or scattered, respectively. The g-factor is defined by the mean value of the cosine of the scattering angle. These optical parameters can be used as input parameters for Monte Carlo simulations[2], to obtain the distribution of light transmitted through or reflected by the medium. In the Monte Carlo method, the paths of a large number of photons are simulated in a randomized manner using the known geometry and optical coefficients.

In a previous work[3], the optical properties of human blood were measured in conjunction with slow heating. Sudden changes in g-factor and scattering coefficient were observed simultaneously as the red blood cells were transformed from spheroidal into spherical shape. Other authors[4,5] have studied the light scattering by one erythrocyte and indicated that the cell shape influences the propagation of light. The scattering process in the Monte Carlo model has also been criticized[6], when simulating the light distribution in human blood.

The purpose of this work is to study the angular scattering by one spheroidal particle, the influence of size, shape, concentration and orientation of spheroidal particles distributed in a homogenous non-absorbing medium and to develop a computer program taking these characteristics into consideration, when evaluating the propagation of light. The study includes in particular particles with the size of erythrocytes. For calculations of the angular scattering by one spheroidal particle, the T-matrix method is an effective tool, described and used in this work. Nevertheless, the idea is that the developed program will be used not only for investigations of light propagating in human blood, but also in any tissue or media in general, using other shapes and sizes of the scattering particles. Interesting results of investigations of the optical and printing properties of paper, correlated to the structures of fibres and pores[7], have also given inspiration to this work and might be another application of this model.

2. Theory

2.1. Scattering from one particle

2.1.1. Electric time-harmonic fields

The time-dependent electric field of a uniform plane wave in space can be expressed as

$$\mathbf{E}(t,z) = \mathrm{Re}(\mathbf{E}(kz)\, e^{-i\omega t}) \qquad (2.1)$$

Re( ) extracts the real part of a complex variable.

The complex field **E**(kz) is written as

$$\mathbf{E}(kz) = E_0 e^{i(kz-\varphi)}\mathbf{e}_e \; ; \; k = 2\pi n/\lambda_0 \qquad (2.2)$$

where z is the position in the direction of propagation $\mathbf{e}_z$, φ is the phase at z = 0, n represents the complex index of refraction and λ_0 the free space wavelength. The notation $\mathbf{e}_e$ symbolizes the unity vector in the direction of the **E**-field ($\mathbf{e}_e \perp \mathbf{e}_z$).

The intensity (time-averaged power per unit area) is determined by

$$\frac{1}{2}\,|\,\mathrm{Re}(\mathbf{E}\times\mathbf{H}^*)\,| = \frac{1}{2}\,|\mathbf{E}|^2\,\mathrm{Re}\left(\frac{1}{\eta^*}\right) \qquad (^*\text{ stands for the complex conjugate}) \qquad (2.3)$$

H represents the magnetic field and is proportional to **E**. The two fields are perpendicular to each other and to the propagating direction. Thus, the intensity is proportional to $|\mathbf{E}|^2$. The complex variable η symbolizes the impedance of the medium.

When a plane wave is scattered by a particle, a scattered wave is generated. Far from the particle, the time-independent scattered electric field can be expressed as

$$\mathbf{E}(k\mathbf{r}) = \mathbf{E}_0\,\frac{e^{ikr}}{r} \qquad (2.4)$$

and is hence termed the far-field. The amplitude of the near-field has a stronger r-dependence and decays faster.

2.1.2. The T-matrix method

Assuming a particle is illuminated by the incident electric field $\mathbf{E}_i$. This field then induces an internal field $\mathbf{E}_{int}$ and a scattered field denoted by $\mathbf{E}_s$. The incident field is a plane-wave in the **k**-direction:

$$\mathbf{E}_i(k\mathbf{r}) = E_0 e^{i\mathbf{k}\cdot\mathbf{r}}\,\mathbf{e}_e \; ; \quad \mathbf{k} = k\mathbf{e}_k \qquad (2.5)$$

This expression can be expanded as

$$\mathbf{E}_i(k\mathbf{r}) = E_0 \sum_{\nu} D_\nu[a_\nu \mathbf{M}_\nu(k\mathbf{r}) + b_\nu \mathbf{N}_\nu(k\mathbf{r})] \qquad (2.6)$$

where $\nu = (\sigma,m,n)$ is the triple index for spherical harmonic functions (σ indicates whether the harmonic function is odd (o) or even (e), m represents the azimuthal mode and n the expansion-coefficient). $\mathbf{M}_\nu$ and $\mathbf{N}_\nu$ are Bessel-radial dependent vector spherical functions in the two polarization directions and D_ν is a normalization constant. The incident field expansion coefficients a_ν and b_ν are obtained by evaluating different combinations of trigonometric functions and associated Legendre-functions $P_n^m(\cos\theta)$. For example, a_{emn} is determined by

$$a_{emn} = 4\, i^n \mathbf{e}_e \cdot [-\mathbf{e}_\theta \sin(m\phi)\, m \frac{P_n^m(\cos\theta)}{\sin\theta} - \mathbf{e}_\phi \cos(m\phi) \frac{d}{d\theta}(P_n^m(\cos\theta))] \qquad (2.7)$$

where θ and ϕ indicate the direction of $\mathbf{k}$ and $\mathbf{e}_\theta$ and $\mathbf{e}_\phi$ are unity vectors in these directions. The coefficients a_{omn}, b_{emn} and b_{omn} are expressed in a similar way. Jackson[8] has made a more detailed description about spherical wave-expansion of plane waves.

In the same way as the incident field, the internal and scattered field can be expanded:

$$\mathbf{E}_{int}(mk\mathbf{r}) = E_0 \sum_{\mu} D_\mu[c_\mu \mathbf{M}_\mu(mk\mathbf{r}) + d_\mu \mathbf{N}_\mu(mk\mathbf{r})] \qquad (2.8)$$

$$\mathbf{E}_s(k\mathbf{r}) = E_0 \sum_{\nu} D_\nu[f_\nu \mathbf{M}_\nu(k\mathbf{r}) + g_\nu \mathbf{N}_\nu(k\mathbf{r})] \qquad (2.9)$$

The vector spherical functions **M** and **N** have a Bessel-radial dependence in equation (2.8) and a Hankel-radial dependence in equation (2.9). The coefficients c_μ, d_μ, f_ν and g_ν are unknown and the refractive index of the particle relative the surrounding medium is denoted by m.

Because of the linearity of Maxwell's equations, the relation between the different expansion coefficients is linear. Thus, when the coefficients a_ν and b_ν are known as well as the shape and size of the particle, it is possible to determine c_μ and d_μ :

$$\begin{bmatrix} & & \\ & \text{A - matrix} & \\ & & \end{bmatrix} \begin{bmatrix} c_\mu \\ \\ d_\mu \end{bmatrix} = -i \begin{bmatrix} a_\nu \\ \\ b_\nu \end{bmatrix} \qquad (2.10)$$

f_ν and g_ν are calculated in the same way:

$$\begin{bmatrix} f_\nu \\ \\ g_\nu \end{bmatrix} = -i \begin{bmatrix} & & \\ & \text{B - matrix} & \\ & & \end{bmatrix} \begin{bmatrix} c_\mu \\ \\ d_\mu \end{bmatrix} \qquad (2.11)$$

Thus, this relation can be expressed as

$$\begin{bmatrix} f_\nu \\ \\ g_\nu \end{bmatrix} = -\begin{bmatrix} \text{B - matrix} \end{bmatrix}\begin{bmatrix} \text{A - matrix} \end{bmatrix}^{-1}\begin{bmatrix} a_\nu \\ \\ b_\nu \end{bmatrix} = -\begin{bmatrix} \text{T - matrix} \end{bmatrix}\begin{bmatrix} a_\nu \\ \\ b_\nu \end{bmatrix} \qquad (2.12)$$

where T-matrix is a shorter name for the so called transition-matrix. Obviously, the T-matrix can be used to calculate f_ν and g_ν and thus the scattered field.

The elements of the A- and B- matrices consist of surface-integrals containing products of spherical functions. The integrals have to be numerically determined over the surface of the particle for every $\nu = (\sigma,m,n)$ combination. If the particle is axi-symmetric simplifications can be made so that the surface integrals can be reduced to one-dimensional integrals containing combinations of Bessel-, Hankel-, associated Legendre- and trigonometric functions. As the elements are independent of the incident and scattered fields, it's enough to compute the T-matrix once for a specific scattering particle at a fixed wavelength. In the case of a sphere, the T-matrix will be diagonal and explicit expressions can be obtained for the diagonal elements. These expressions will be identical with the ones obtained in Mie-scattering.

When $\mathbf{E}_s$ is evaluated, the far-field $\mathbf{F}$ is defined by

$$\mathbf{E}_s(k\mathbf{r}) = \mathbf{F}(\theta,\phi)\,\frac{e^{ikr}}{r} \qquad (2.13)$$

in the limit $kr \rightarrow \infty$.

F_θ and F_ϕ are the two components of $\mathbf{F}$. By replacing the Hankel-functions with asymptotic expressions (in the $kr \rightarrow \infty$ limit), an explicit form of $\mathbf{F}$ can be obtained[9]. For example, for $\mathbf{e}_\theta$ incident polarization, F_θ is evaluated by

$$F_\theta = \frac{i^{-(n+1)}}{k}\cos(m\phi)\; m\left(\frac{P_n^m(\cos\theta)}{\sin\theta}\right) D_{mn} f_{omn} +$$
$$+ \frac{i^{-n}}{k}\cos(m\phi)\;\left[n\cos\theta\left(\frac{P_n^m(\cos\theta)}{\sin\theta}\right) - (n+m)\left(\frac{P_{n-1}^m(\cos\theta)}{\sin\theta}\right)\right] D_{mn} g_{emn} \qquad (2.14)$$

The differential scattering cross section, expressing the scattered intensity in a specific direction relative the intensity of the incident field, can then be determined:

$$\sigma_d(\theta,\phi) = |\mathbf{F}(\theta,\phi)/E_0|^2 \qquad (2.15)$$

The total scattering cross section includes all scattering directions and is given by

$$\sigma_s = \pi k^{-2} \sum_\nu D_\nu \left(|f_\nu|^2 + |g_\nu|^2 \right) \qquad (2.16)$$

By using the optical theorem[10], the total extinction cross section σ_e, which describes how much the incident field is influenced by the particle, can be expressed as

$$\sigma_e = 4\pi k^{-2} \mathrm{Im}(\mathbf{e}_0 \cdot k\mathbf{F}/E_0) \qquad (2.17)$$

F is evaluated in the direction straight forward and $\mathbf{e}_0$ is the polarization unity vector of the incident field. Im( ) extracts the imaginary part.

Finally, the absorption cross section is defined by

$$\sigma_a = \sigma_e - \sigma_s \qquad (2.18)$$

By dividing the cross sections by πa^2, where a is a characteristic dimension of the particle, normalized cross sections can be calculated.

Other authors[9,11-12] have described the T-matrix method more extensively and specific properties of the T-matrix are analyzed by Waterman[13].

2.2. Multiple scattering

2.2.1. The original Monte Carlo method

Light propagation in turbid media can be simulated using Monte Carlo methods[2]. In those, photons or photon packages are sent on a random walk through the tissue. The phase and polarization of the photons are not taken into consideration in this method, due to their random behaviour after multiple scattering in the tissue. The wave phenomena of the photon are thus ignored and the photon is treated as a particle. The path of every photon package is simulated until it is emerged or absorbed. When a large amount of photon packages are used, good statistic results of the light distribution in the tissue can be obtained.

The computer code MCML, written by Wang and Jacques[2], is a program based on the Monte Carlo method, where the distribution of absorbed light in a medium, containing layers with different optical parameters and refractive indices, can be computed as well as the distributions of reflected and transmitted light.

When the photon package enters the media, it is assigned an initial weight W. This weight will be reduced each time the photon package interacts with the media. The fraction ΔW which is subtracted from the initial weight is given by

$$\Delta W = \frac{\mu_a}{\mu_a + \mu_s} W \qquad (2.19)$$

The path length, S, between the points of interaction is

$$S = \frac{-\ln(1-R)}{\mu_a + \mu_s} \qquad (2.20)$$

where R represents a random number between 0 and 1.

In every interaction point a new direction of the photon package is obtained. The azimuthal angle ϕ is calculated by

$$\phi = 2\pi R \tag{2.21}$$

where R is assigned a new random value between 0 and 1. In order to determine the deflection angle θ, the Henyey-Greenstein phase function[14] is utilized. This function, often used in tissue optics, is in fact a probability distribution expressed as

$$p(\cos\theta) = \frac{1-g^2}{4\pi(1+g^2-2g\cos\theta)^{3/2}} \tag{2.22}$$

where the anisotropy coefficient g is in the range between -1 and 1. Complete back and forward scattering are described by $g = -1$ and $g = 1$, respectively and $g = 0$ represents isotropic scattering. The deflection angle θ, assumed to be distributed according to the Henyey-Greenstein distribution can then be determined by using a new random value R, between 0 and 1:

$$\cos\theta = \frac{1+g^2-\left(\frac{1-g^2}{1-g+2gR}\right)^2}{2g} \tag{2.23}$$

A special routine is used in the program to determine when the photon package is totally absorbed. To ensure the conservation of energy, a roulette routine is used if the photon weight falls below a particular threshold value. The photon package survives at a certain probability, m, and is then assigned the value of m times the old weight. It is otherwise terminated. The MCML-program finally also takes care of the processes occurring when a photon package hits a boundary between two layers with different optical properties, using Snell's law and Fresnel's formulas[15]. The Monte Carlo method is discussed more rigorously by Keijzer[16].

3. Material and methods

3.1. The T-matrix computer programs

A series of computer programs, written in FORTRAN, which can be utilized for determination of the scattering and absorption characteristics of the finite-thickness slab, infinite circular cylinder, general axi-symmetric particle and sphere, is developed by Barber and Hill[9]. In this work two of their programs, T1 and T3, were modified to be able to compute the angular scattering of oblate spheroids with larger size parameters as for example the erythrocyte. A short description of the originally programs as well as the necessary modifications are described below.

3.1.1. The original computer program T1

Input parameters to T1 are:

2a :	The diameter of the spheroidal particle along the symmetry axis.
2b :	The diameter of the spheroidal particle perpendicular to the symmetry axis.
$x = ka = 2\pi na/\lambda_0$:	The size parameter.
aovrb = a/b :	Axial ratio.
mr, mi :	The real and imaginary part of the relative index of refraction.
nrank :	Number of expansion coefficients (number of terms in eq. 2.6).
ntheta :	Number of integration points in the one-dimensional integrals in the calculation of the elements of the A- and B-matrices.
case 0 :	The program T1 checks the convergence over ntheta.
case 1 :	The program T1 checks the convergence over nrank.
case 2 :	The program T1 checks the convergence over m. (m is the number of azimuthal modes).

In order to evaluate a correct T-matrix, a sufficient amount of expansion coefficients, integration points and azimuthal modes are needed depending on the size and shape of the scattering particle[17]. Large particles and oblate- and prolate-shaped particles with a considerable axial ratio require larger values of nrank and ntheta, compared to small and spherical-shaped particles. The possibility of obtaining a converged solution depends on the computer capacity and the number of precision variables.

The program T1 first checks the number of expansion coefficients, integration points and azimuthal modes needed to obtain a converged solution for a specific scattering particle in a specific surrounding medium and then it determines the T-matrix. Figure 3.1 shows a cross sectional picture of one possible scattering particle. The particle must be axi-symmetric and fulfil the equation

$$r(\theta) = \frac{a}{\sqrt{\cos^2\theta + (\frac{a}{b})^2 \sin^2\theta}} \qquad (3.1)$$

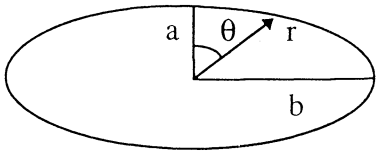


Figure 3.1. A cross sectional picture of a scattering oblate-shaped spheroid. The particle is rotational symmetric around its semiminor axis. The diameter of the particle along this axis equals 2a.

When determining convergence over nrank and ntheta for a specific particle, the incident field is assumed along the symmetry axis and the azimuthal mode m can be set to one, as all the other azimuthal modes then are zero. Convergence is verified when the angular scattering outputs for nrank and nrank-1 or ntheta and ntheta+4, respectively, correspond with a certain accuracy.

The obtained values of nrank and ntheta are then used when evaluating convergence over m. In this case (2), the incident field is assumed along a direction 45 degrees from the symmetry axis (the most complicated case). The program calculates and compares the angular scattering for increasing m-values until convergence is obtained or m > nrank (which means that no converged solution exists). By the use of case 2, the T-matrix is determined, but convergence over nrank and ntheta must first have been obtained to be sure that the determination is correct. The number of rows (or columns) in this matrix equals twice nrank. When the T-matrix finally is stored it can be used to compute various cross sections and resulting scattered fields in different directions.

3.1.2. Modifications of program T1

Using a larger T-matrix, more integration-points and extended precision variables, converged solutions are possible for particles with larger size parameters and axial ratios. Extended precision uses a significance of 32 digits instead of normally 8 digits. The extended precision variables are necessary, as the round-off-errors in the ill-conditioned process of calculating the integrals in the A- and B-matrices (2.12) become increasingly significant for these kind of particles. The DEC OSF/1 compiler, used in this work, applies extended precision for real variables, but only double precision (16 digits) for each part of a complex variable. With these facts in mind, some modifications of the T1-program have been done:

- The maximum size of the matrices used to evaluate the T-matrix has been increased.
- The real variables have been declared real*16 (extended precision).
- The complex variables have been split up into two real variables, through critical parts of the program.
- The accuracy in the integration routine has been increased, as there are values with more digits to pay attention to.
- The accuracy when a solution is accepted as being converged has been increased. The original program requires an accuracy of 1.0% in 8 of 10 angles, whereas the modified version requires 0.1% in all angles.

However, one consequence of these modifications is that the index of refraction must be real, which results in that no absorption cross section can be evaluated. This can be adjusted by splitting all complex variables in the program into two real ones.

To prove the correctness of the modifications, comparing tests were performed for particles which have a converged solution in both the original and the modified program. Additional tests were made to check that the obtained T-matrices still fulfil the condition of energy conservation[11]: $T^*T + Re(T) = 0$ (* symbolizes the complex conjugate). The modified computer program is called EXTENDT1 and is able to calculate accurate T-matrices, even for particles with the dimension of a human blood cell.

3.1.3. The original computer program T3

Program T3 uses the T-matrix, obtained from program T1, to calculate the differential scattering cross sections in all scattering directions. The input parameters to this program are:

angint: The angle between the symmetry axis of the particle and the incident wave.
anginp: The orientation of the particle relative a reference frame in the plane perpendicular to the symmetry axis.
anglab: The orientation of the polarization direction.
npnts: The number of points in which the angular scattering is evaluated is $npnts^2$.

The normalized differential scattering cross sections as a function of the scattering angles θ and ϕ are put in a vector and stored in the file T3.DAT. The graphical representation is a planar grid as shown in figure 3.2. The deflection angle θ is mapped along the radius r in the graph ($\theta = 0°$ when $r = 0$ and $\theta = 180°$ when $r = 1$) and ϕ is mapped in the normal way using cylindrical coordinates. The logarithm of the normalized differential scattering cross section at each grid-point is calculated and stored in the vector starting in the corner (X,Y) = (-1,-1), continuing by increasing X until X = 1 (constant Y), followed by the same procedure for the next Y-value etc. The value of the grid points outside the unit circle is set to the logarithm of the back scattering value.

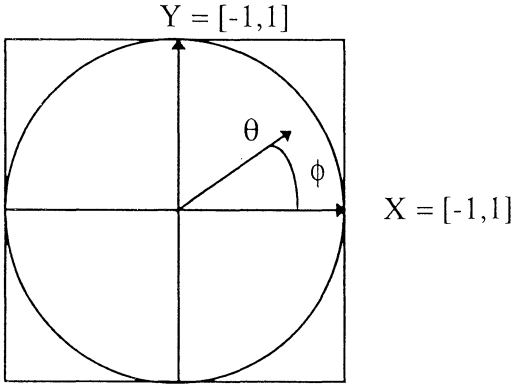


Figure 3.2. The representation of the scattering angles (θ,φ) on a planar grid.

3.1.4. Modifications of program T3

The modified version of T3, called EXTENDT3, is able to use the modified T-matrix. No further modifications have been made in this program.

In program MCDATA, an additional modified version of T3, probability distributions of the scattering angles are computed instead of the normalized differential scattering cross sections. One probability distribution is obtained for each of a certain number of incident angles. The number of incident angles, uniformly distributed between 0° and 90°, is an input parameter (nbrangin) and the variable anginp from program T3 is initially always set to zero. The input parameter npnts is replaced by the two parameters nbrthsc and nbrphsc, which describe the number of spherical coordinate angles θ and ϕ, respectively, uniformly distributed in space, in which the scattering probability shall be evaluated for every incident angle. Thus, these variables indicate the resolution of the probability distribution. When evaluating the distribution for one incident angle, the normalized differential scattering cross sections are first determined for all combinations of θ and ϕ, and the probability of scattering in a certain direction is obtained by normalization with the sum of the scattering in all other directions, and with the relative area this scattering direction represents. The obtained probability values are then stored in the file MC.DAT, making it easy to implement a program utilizing these values. A further description of the technical details of the implementation is found in Appendix C.

3.2. The Monte Carlo computer programs

3.2.1. The original Monte Carlo program

As mentioned before, the standard C computer code MCML[2] can be used to simulate the distributions of reflected, absorbed and transmitted light of a specified illuminated medium. The medium consists of a number of infinitely wide parallel layers, each with particular optical properties. The optical parameters (μ_a, μ_s and g), index of refraction and thickness of each layer are input parameters to the program, as well as the number of photon packages and variables defining the resolution of the light distribution. The variables dr and da represent the radial and angular resolution, respectively, and dz is the resolution along the axis parallel to the incident beam.

The output file, obtained from the program, contains the angularly and spatially resolved fractions as well as the total fractions of reflected, absorbed and transmitted light. Other terms, used in the output file are the reflectance per unit area and transmittance per unit area. These terms actually represent the probabilities of detection of a photon per unit area[2]. The reflectance per unit area at a specific distance r from the centre of the sample is thus calculated by dividing the total photon weight detected at this distance by the total number of photon package and the area of an annular ring $\Delta a = 2\pi r\, dr$. The transmittance per unit area is calculated in a corresponding way. Specular reflectance is the amount of light, reflected from the outer boundary of the sample, without interacting the with medium. The remaining reflectance is called diffuse reflectance.

3.2.2. Modifications due to boundary conditions

To be able to model the light distribution of samples, which have a finite extension, a modified version of the computer program is needed. The modified version MCMLBOUN assumes that the sample is laterally surrounded by glass layers, as shown in figure 3.3. The processes occurring when the photon package hits the surrounding glass are the same as when it hits a boundary between two layers. As glass is non-scattering, photons propagating in this glass, away from the sample, will never turn back. The lateral dimension of the sample and the refractive index of the surrounding glass are new input parameters.

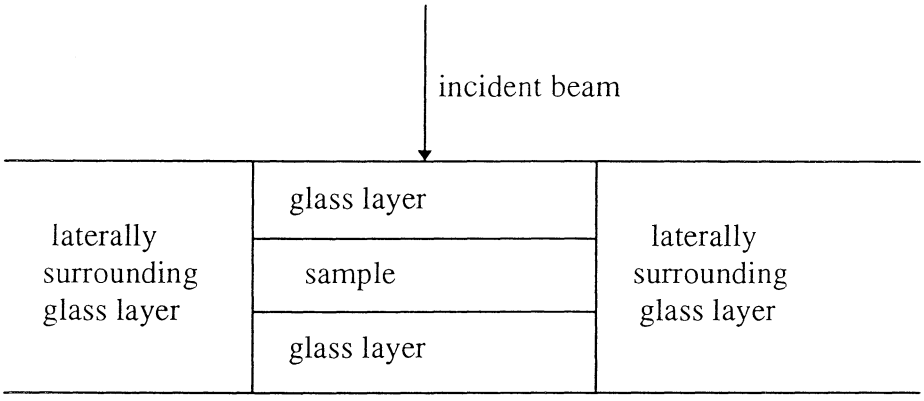


Figure 3.3. The cross sectional geometry for a sample and surrounding glass layers, used in program MCMLBOUN.

3.2.3. Implementation of specific scattering objects

An additional modified version, TMMCML, has been developed within this work. This version is based on the assumption of specific scattering objects, spheroidal particles, are distributed in the medium. The optical parameters in the original version are replaced, in the input file as well as in the program, by the orientation of the scattering objects and the mean distance between them. Standard deviations of these new parameters, assuming Gaussian distributions, are also defined in the input file.

The two main differences between this version and the original one, are the process of the photon package making a new step and the calculation of the new photon direction from an interaction point. The interaction points are in this case equal to the scattering particles. The step of the photon package before a new interaction occurs is obtained from a Gaussian distribution[18], where the mean distance and standard deviation are defined in the input file as mentioned before. A new scattering object is hit and the angle between the symmetry axis of the scattering spheroid and the incident direction of the photon package is evaluated. A new direction of the photon package is then obtained using a random number and the data base in MC.DAT. This data base contains the probability of scattering in different directions for several incident angles, as mentioned in chapter 3.1.4. The name of the data base, as well as the resolution of incident and scattering directions are defined by the user as input parameters. If the scattering objects are spheres, only one incident angle is needed. These modifications also require a modification of the absorbing process. A new variable, the normalized absorption cross section, is defined in the input

file and specifies the relative weight of the incident photon package being absorbed by the scattering object. The input parameters of one layer in program TMMCML is summarized below.

meanstep :	The mean distance between the particles.
meanstepdev :	The standard deviation of this distance.
abscross :	The normalized absorption cross section. (The part of the light absorbed by one particle.)
bcux, bcuy, bcuz :	The direction of the direction of the particle symmetry axis (the bcuz-direction is parallel with the incident field).
st. dev :	The standard deviation of these directions.
d :	The thickness

3.3. Testing program MCMLBOUN, due to boundary conditions

Monte Carlo simulations, using the program MCMLBOUN accounting for lateral boundary effects, were performed for four samples with identical optical properties but different lateral extensions. The results were compared with each other and with results from an original Monte Carlo simulation in order to investigate the influence of boundary conditions, as the original Monte Carlo method assumes an infinite medium, but the extension of experimental samples are finite. The optical properties were chosen to be similar to those of human tissue[19] and the thickness of the sample was chosen close to experimentally used values[20]. The sizes of the samples were 10000 cm × 10000 cm, 5 cm × 3 cm, 1 cm × 1 cm and 0.1 cm × 0.1 cm. Table 3.1 shows further input parameters.

# No. of photons	500000				
# dz, dr	0.25	0.008			
# No. of dz, dr, da	1	75	30		
# Number of layers	3				
# layer	n	mua	mus	g	thickness
# n for medium above	1				
# layer 1	1.5	0	0	0	0.1
# layer 2	1.4	2.6	267	0.95	0.05
# layer 3	1.5	0	0	0	0.1
# n for medium below	1				
# n for surrounding glass	1.5				

Table 3.1. Input parameters of the tests in section 3.3, investigating the influence of boundary conditions. The variables mua and mus represent the absorption and scattering coefficients μ_a and μ_s, n is the refractive index and g is the anisotropy factor. The radial and angular resolutions are described by dz, dr and da. The length unit is always cm.

3.4. Testing limitations and accuracy

Several different comparisons have been made, using the modified programs EXTENDT1, EXTENDT3 and MCDATA. The calculations were mainly performed for particles with sizes and properties similar to erythrocytes and a surrounding medium

similar to blood plasma. The parameters (chapter 3.1.1 and 3.1.3), common in all the tests using these programs, were thus n = 1.345 (refractive index of blood plasma[21]), mr = 1.045 (refractive index of a blood cell relative the blood plasma[22]), mi = 0 (the imaginary refractive index of a blood cell, which is negligible when evaluating the scattered field[23]) and λ_0 = 632.8 nm (wavelength of a He-Ne laser[24]). The variable ntheta (number of integration points) was always assigned the value of twice nrank.

In this section, tests were made to study the limits in size and shape of the particle in order to obtain a converged solution, as well as the influence of different polarization directions of the incident light and the number of incident and scattering angles needed to obtain a sufficient accuracy. In the last-mentioned study, the program TMMCML was executed for four different values of the variable nbrthsc, which represents the number of scattering deflection angles θ, in which the differential scattering cross section shall be determined. The values assigned to this variable were 46, 91, 181 and 721, respectively, and the remaining input parameters of these simulations are specified in table 3.2. The scattering particles were spheres with a volume equal to a blood cell. This volume equals 94 μm^3 [22], which leads to ka = 37.67, a/b = 1 and requires nrank = 50, where nrank represents the number of expansion coefficients in eq. 2.6, 2.8 and 2.9. The orientation of polarization (anglab) was set to zero as well as the incident angles (angint and anginp), which are arbitrary in the case of a sphere.

# nbrangin and nbrphsc	1	37	
# No. of photons	100000		
# dz, dr	102	4	
# No. of dz, dr, da	1	75	30
# x_size and y_size	5E+06	5E+06	

# Number of layers	3				
# layer	n	m-step	m-dev.	abs.	thickness
# n for medium above	1				
# layer 1	1.5	0	0	0	1
# layer 2	1.345	0.1	0.02	0.0001	100
# layer 3	1.5	0	0	0	1
# n for medium below	1				
# n for surrounding glass	1.5				

Table 3.2. Input parameters of the TMMCML-tests in section 3.4, performed to study the necessary accuracy of the variable nbrthsc (number of scattering deflection angles θ). The length unit is cm.

3.5. Evaluations and tests of the programs EXTENDT3 and TMMCML

Test 1. In order to investigate the influence of particle size on the light scattering by one particle as well as on the macroscopical light distribution, executions of the programs EXTENDT3 and TMMCML were made for spheres of three different sizes as scattering particles. One sphere with a volume equal to a blood cell (V = 94 μm^3, ka = 37.67, a/b = 1 and nrank ≈ 50), one larger sphere (V = 390 μm^3, ka = 60 and nrank ≈ 100) and one

smaller sphere (V = 14 μm^3, ka = 20 and nrank ≈ 30). The three variables angint, anginp (incident angles) and anglab (polarization direction) were set to zero. The variable npnts was assigned the value of 181 (the number of evaluated scattering angles is $npnts^2$). The normalized differential scattering cross sections in various directions for the spheres were calculated using program EXTENDT3. The logarithms of the obtained cross sections were then plotted in three-dimensional graphs utilizing a small MATLAB program. The input parameters of program TMMCML, used for the calculation of the macroscopical light distribution, are found in table 3.3. The g-factor (average value of the cosine of the scattering deflection angle θ) was calculated for the three spheres using a small FORTRAN program.

# nbrangin, nbrthsc and nbrphsc	1	181	37
# No. of photons	100000		
# dz, dr	102	4	
# No. of dz, dr, da	1	75	30
# x_size and y_size	5E+06	5E+06	

# Number of layers	3				
# layer	n	m-step	m-dev.	abs.	thickness
# n for medium above	1				
# layer 1	1.5	0	0	0	1
# layer 2	1.345	0.1	0.02	0.0001	100
# layer 3	1.5	0	0	0	1
# n for medium below	1				
# n for surrounding glass	1.5				

Table 3.3. Input parameters of the TMMCML-simulations in test 1, made to study the influence of particle size on the macroscopic light distribution. The length unit is cm.

Test 2. The influence of shape and orientation of scattering particles were investigated in this test. The normalized differential scattering cross sections in various directions for an oblate spheroid with the dimension of an erythrocyte were calculated using program EXTENDT3. The values of V = 94 μm^3 and b = 3.91 μm[22] result in a/b = 0.3754 and ka = 19.60. These dimensions require nrank = 66 (number of expansion coefficients). Computations were made for three different incident angles, angint = 0°, 45° and 90°. The variables anginp and anglab were set to zero and npnts = 181. Three-dimensional graphs were plotted to show the angular scattering and the small FORTRAN program, mentioned before, was used to find the anisotropy factor g, in the different cases. The influence of particle shape and orientation on the macroscopical light distribution was analyzed by comparing the output data from program TMMCML, using spheres and the recently described oblate spheroids in various orientations as scattering particles. The volume of the spheres were equal to human blood cells (94 μm^3) and further properties were described in test 1. One simulation was made with the spheres and six with different orientations of the spheroids, described in table 3.4. Remaining input parameters were unchanged from test 1, except the variable nbrangin, which had a value of 91 in the case of spheroids. The absorption cross section was assumed to be independent of the shape of the particle[25].

	spheroid 1	spheroid 2	spheroid 3	spheroid 4	spheroid 5	spheroid 6	sphere
# bcux	rand.	1	0	0	1	1	sphere
# bcuy	rand.	0	1	0	1	1	sphere
# bcuz	rand.	0	0	1	1	1	sphere
# st. dev.		0	0	0	0	0.5	sphere

Table 3.4. The orientation of the scattering spheroids in the seven samples in test 2. The parameters (bcux, bcuy, bcuz) represent the direction of the symmetry axis of the spheroids and this axis is parallel with the incident light beam in the case of (bcux, bcuy, bcuz) = (0, 0, 1). The standard deviation of this direction is denoted by st. dev. in the table and rand. indicates randomized orientation. If st. dev. equals zero, all spheroids have exactly the same direction. In case of spheres, no orientation is needed.

The outlines of a normal blood cell[22] and an oblate spheroid with an equal volume are drawn in figure 3.4 and 3.5.

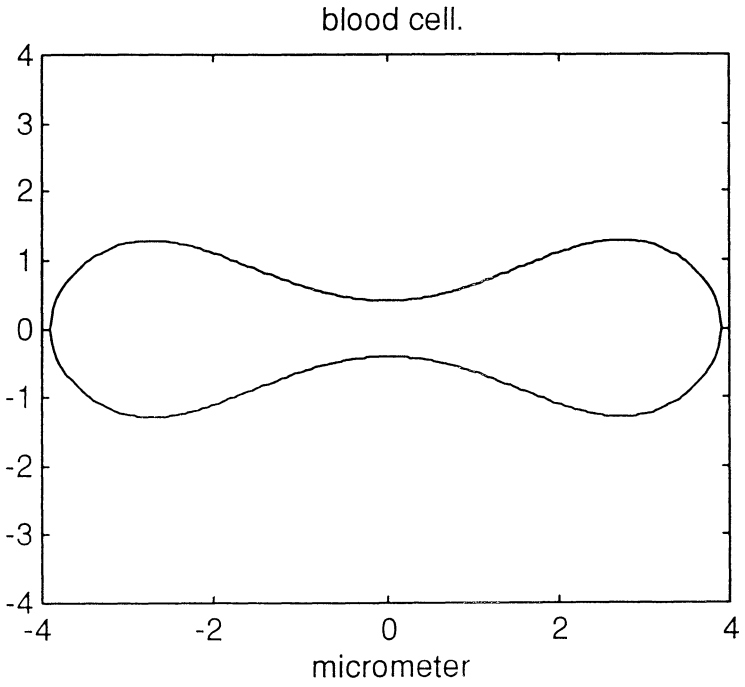


Figure 3.4. The dimension of a blood cell.

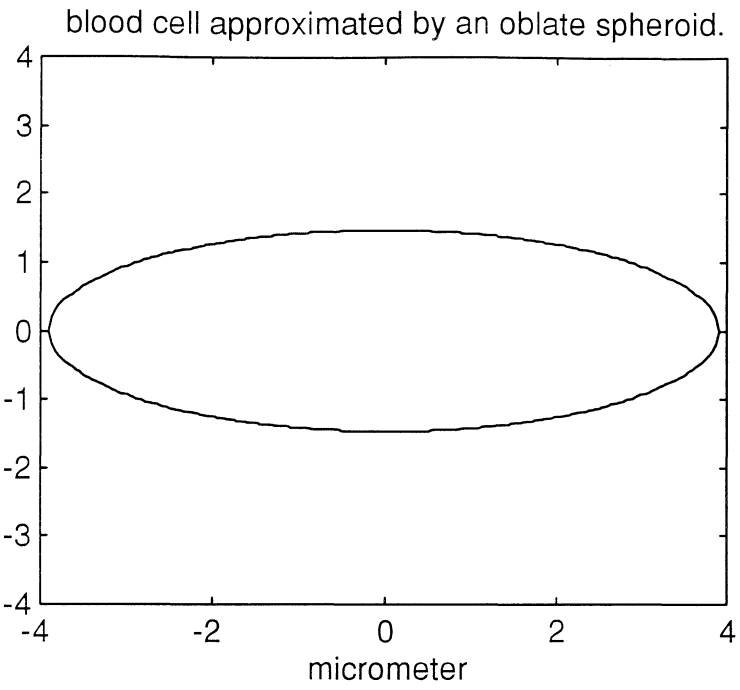


Figure 3.5. The dimension of a blood cell approximated by a volume equivalent oblate spheroid.

Test 3 was performed to investigate the influence of the distance between the scattering particles on the transmitted and reflected light obtained from TMMCML-simulations. Two different distances were used. In the first case, the average distance was assigned a value of 0.2 cm and in the second a value of 0.05 cm. The standard deviation was 20% of the average distance in both cases. The same particles and orientations as in the previous test were utilized. Table 3.5. displays the remaining input parameters.

# dz, dr	7	0.1	
# No. of dz, dr, da	1	75	30
# x_size and y_size	5000	5000	

# Number of layers	3		
# layer	n	abs.	thickness
# n for medium above	1		
# layer 1	1.5	0	1
# layer 2	1.345	0.004	5
# layer 3	1.5	0	1
# n for medium below	1		
# n for surrounding glass	1.5		

Table 3.5. Input parameters of TMMCML-simulations in test 3, investigating the influence of the distance between the scattering particles on the light distribution. The length unit is cm.

3.6. Comparison with the original Monte Carlo model

A comparison between an original Monte Carlo simulation and a simulation with scattering spheres using TMMCML was made, where the extension and thickness of the sample were very large, in order to obtain multiple scattering for all photons. The properties of the spheres were still unchanged (volume equal to an erythrocytes). The mean distance between two interaction points and the absorption during an interaction were identical in the two simulations and the g-factor in the original Monte Carlo program was assigned the mean value of the cosine of the scattering deflection angle θ (0.9940), obtained by a small FORTRAN program from the scattering probability distribution of one sphere. In order to save time, the original Monte Carlo program was used instead of MCMLBOUN, as the lateral extension of the samples were assumed to be large. Table 3.6a and 3.6b present the input parameters.

# No. of photons	250000		
# dz, dr	202	6	
# No. of dz, dr, da	1	75	30

# Number of layers	3				
# One line for each layer	n	mua	mus	g	thickness
# n for medium above	1				
# layer 1	1.5	0	0	0	1
# layer 2	1.345	0.0001	9.9999	0.994	200
# layer 3	1.5	0	0	0	1
# n for medium below	1				

Table 3.6a. Input parameters of the original Monte Carlo simulations in section 3.6.

# nbrangin, nbrthsc and nbrphsc	1	181	37
# No. of photons	250000		
# dz, dr	202	6	
# No. of dz, dr, da	1	75	30
# x_size and y_size	5E+06	5E+06	

# Number of layers	3				
# layer	n	m-step	m-dev.	abs.	thickness
# n for medium above	1				
# layer 1	1.5	0	0	0	1
# layer 2	1.345	0.1	0.02	1.0E-05	200
# layer 3	1.5	0	0	0	1
# n for medium below	1				
# n for surrounding glass	1.5				

Table 3.6b. Input parameters of the TMMCML-simulations in section 3.6.

4. Results

The results and comparisons of the calculations and simulations, using the modified computer programs will be presented in this chapter. In all diagrams and tables, the length unit is cm. Reflectance per unit area has consequently the unit cm^{-2}.

4.1. Testing program MCMLBOUN, due to boundary conditions

The distributions of reflected and transmitted light obtained when executing the program MCMLBOUN for four samples with different lateral extensions are shown in figures 4.1 and 4.2, as well as the corresponding result from an original Monte Carlo simulation. Table 4.1 presents the total reflected, absorbed and transmitted parts and table 4.2 indicates the time of each simulation. The difference in reflected and transmitted light between the original Monte Carlo simulation and the two largest samples (10000 cm × 10000 cm and 5 cm × 3 cm) in the MCMLBOUN-simulations never exceeds 3%, except close to origo. Regarding the total amounts, this deviation is much less than 1%. From the results of the simulations using samples with sizes less than 1 cm × 1 cm a larger deviation can be observed, up to 25% at some radial distances.

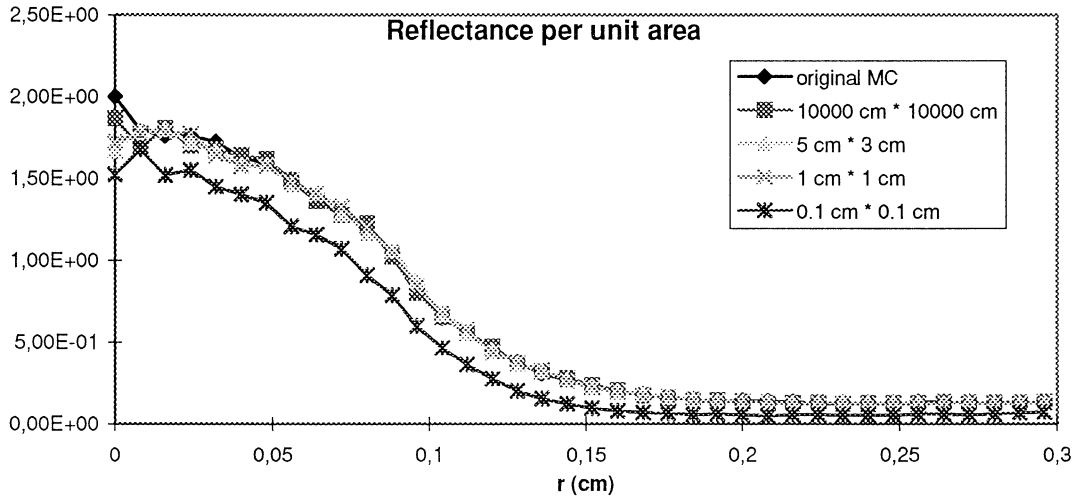


Figure 4.1. The reflectance per unit area versus the radial distance from the centre of the sample. The filled rhombs correspond to the result of the original Monte Carlo program. The other symbols correspond to results of program MCMLBOUN, where the sizes of the samples are described in the figure. The used parameter were $\mu_a = 2.6\ cm^{-1}$, $\mu_s = 267\ cm^{-1}$ *and* $g = 0.95$. *The thickness was 0.05 cm.*

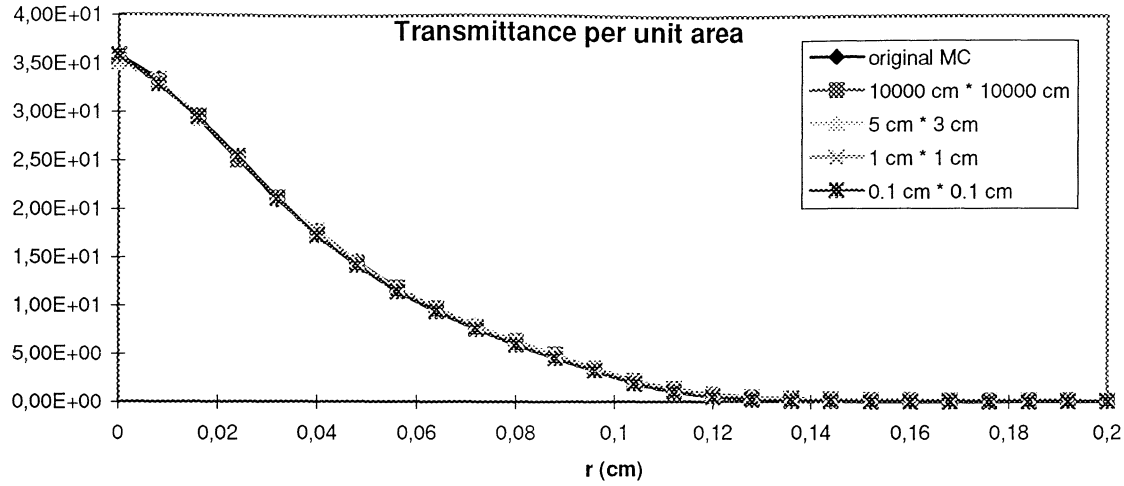


Figure 4.2. The transmittance per unit area versus the radial distance from the centre of the sample. The filled rhombs correspond to the result of the original Monte Carlo program. The other symbols correspond to results of program MCMLBOUN, where the sizes of the samples are described in the figure. The used parameter were $\mu_a = 2.6\ cm^{-1}$, $\mu_s = 267\ cm^{-1}$ *and* $g = 0.95$. *The thickness was 0.05 cm.*

	orig. MC (infinite)	10000 cm* 10000 cm	5.0 cm* 3.0 cm	1.0 cm* 1.0 cm	0.1 cm* 0.1 cm
# Specular reflectance	0.04110	0.04110	0.04110	0.04110	0.04110
# Diffuse reflectance	0.16110	0.16059	0.15965	0.13306	0.06595
# Absorbed fraction	0.30816	0.30813	0.30671	0.26503	0.15045
# Transmittance	0.48965	0.49018	0.48900	0.46251	0.39259

Table 4.1. The parts of reflected, absorbed and transmitted light for the five different sizes of the samples used in the tests described in section 3.3, investigating the influence of boundary conditions.

orig. MC (infinite)	10000 cm* 10000 cm	5.0 cm* 3.0 cm	1.0 cm* 1.0 cm	0.1 cm* 0.1 cm
t=2152s	t=3683s	t=3092s	t=2167s	t=1108s

Table 4.2. The simulation time for the five different sizes of the samples used in the tests described in section 3.3, investigating the influence of boundary conditions.

4.2. Testing limitations and accuracy

When testing which sizes and shapes of particles, possible for obtaining a converged solution in program EXTENDT1, the minimal axial ratio obtained for ka = 19.6 was a/b = 0.34, the largest size parameter for a/b = 0.3754 was ka = 22 and the highest possible value of the relative refractive index for this kind of particles was mr = 1.06. Converged

solutions were obtained for prolate particles with about 2% larger volumes than the corresponding oblate particles.

Tests were also performed indicating that the angular scattering for two incident angles separated by 1° deviate less than 1%. Results obtained with different polarization directions of the incident light showed the same small deviation.

The obtained radial resolutions of the reflectance per unit area and transmittance per unit area from the TMMCML-simulations when testing the output resolution needed by changing the variable nbrthsc (the number of scattering deflection angles θ) are shown in figures 4.3 and 4.4. The total amounts are shown in table 4.3. The differences in total amounts of reflected and transmitted light are less than 2%, comparing nbrthsc = 46 and nbrthsc = 721. Investigating radial resolution of the light, the deviation is below 6%, except close to origo.

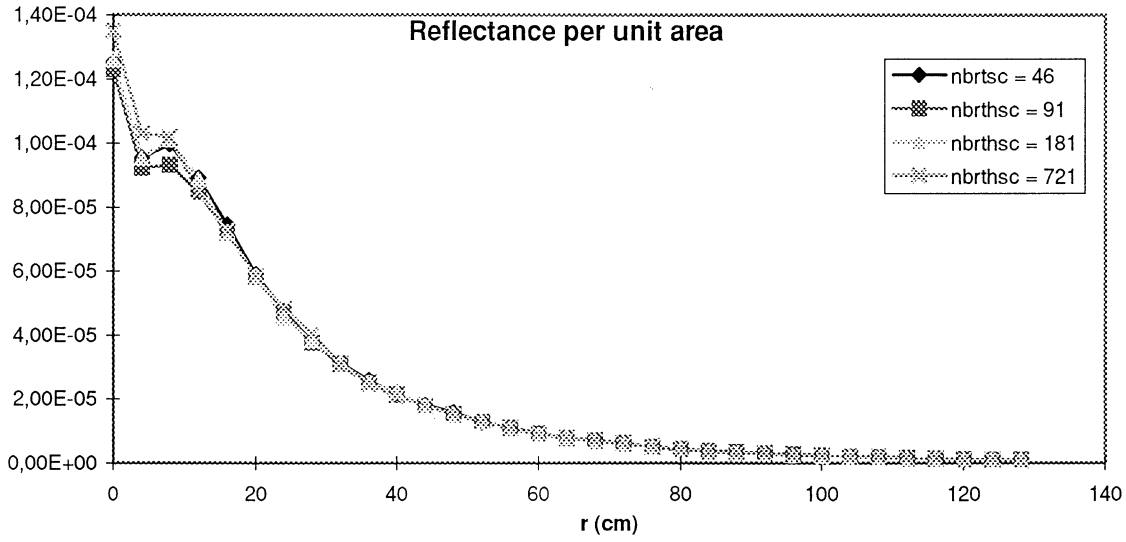


Figure 4.3. The reflectance per unit area versus the radial distance from the centre of the sample in the tests described in section 3.4, investigating the accuracy needed. The symbols in the graph correspond to nbrthsc = 46, 91, 181 and 721, respectively, where nbrthsc represents the number of scattering deflection angles θ. The mean distance between the particles was 0.1 cm and the sample thickness was 100 cm.

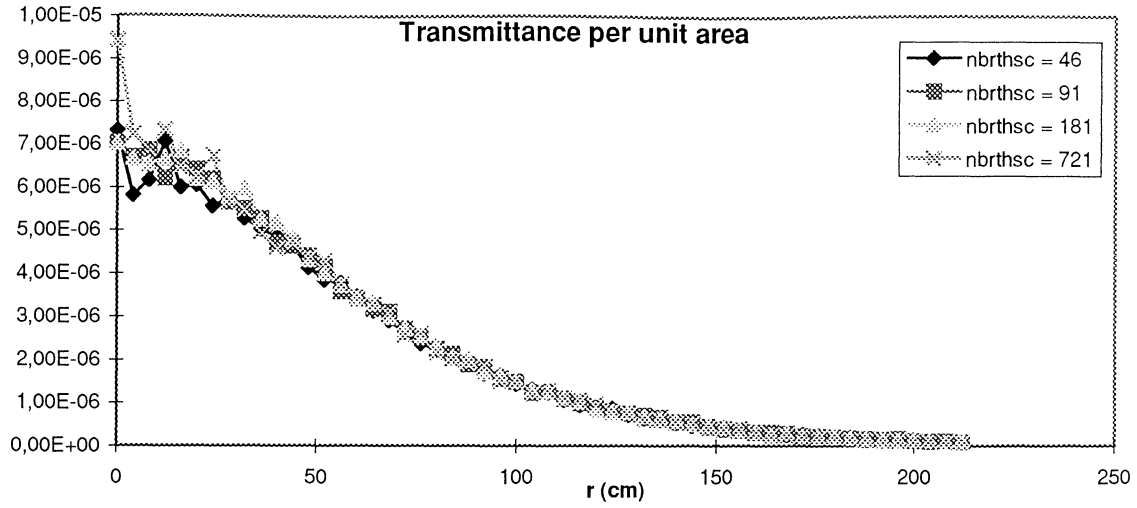


Figure 4.4. The transmittance per unit area versus the radial distance from the centre of the sample in the tests described in section 3.4, investigating the accuracy needed. The symbols in the graph correspond to nbrthsc = 46, 91, 181 and 721, respectively, where nbrthsc represents the number of scattering deflection angles θ. The mean distance between the particles was 0.1 cm and the sample thickness was 100 cm.

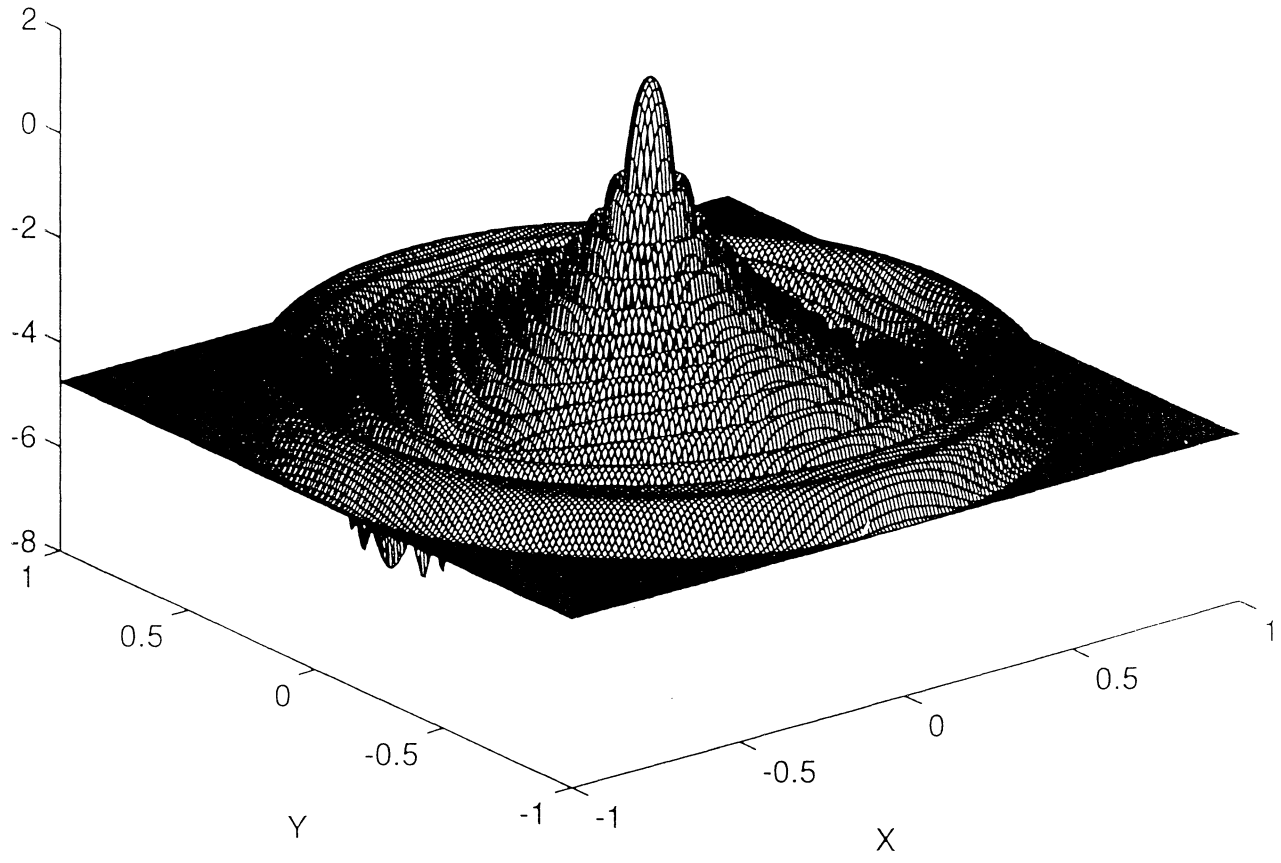
	nbrthsc=46	nbrthsc=91	nbrthsc=181	nbrthsc=721
# Specular reflectance	0.04274	0.04274	0.04274	0.04274
# Diffuse reflectance	0.51734	0.51296	0.51136	0.51557
# Absorbed fraction	0.28436	0.28490	0.28513	0.28324
# Transmittance	0.15556	0.15941	0.16077	0.15846

Table 4.3. The parts of reflected, absorbed and transmitted light for the four values of nbrthsc in the tests described in section 3.4, investigating the accuracy needed of the variable nbrthsc (the number of scattering deflection angles θ).

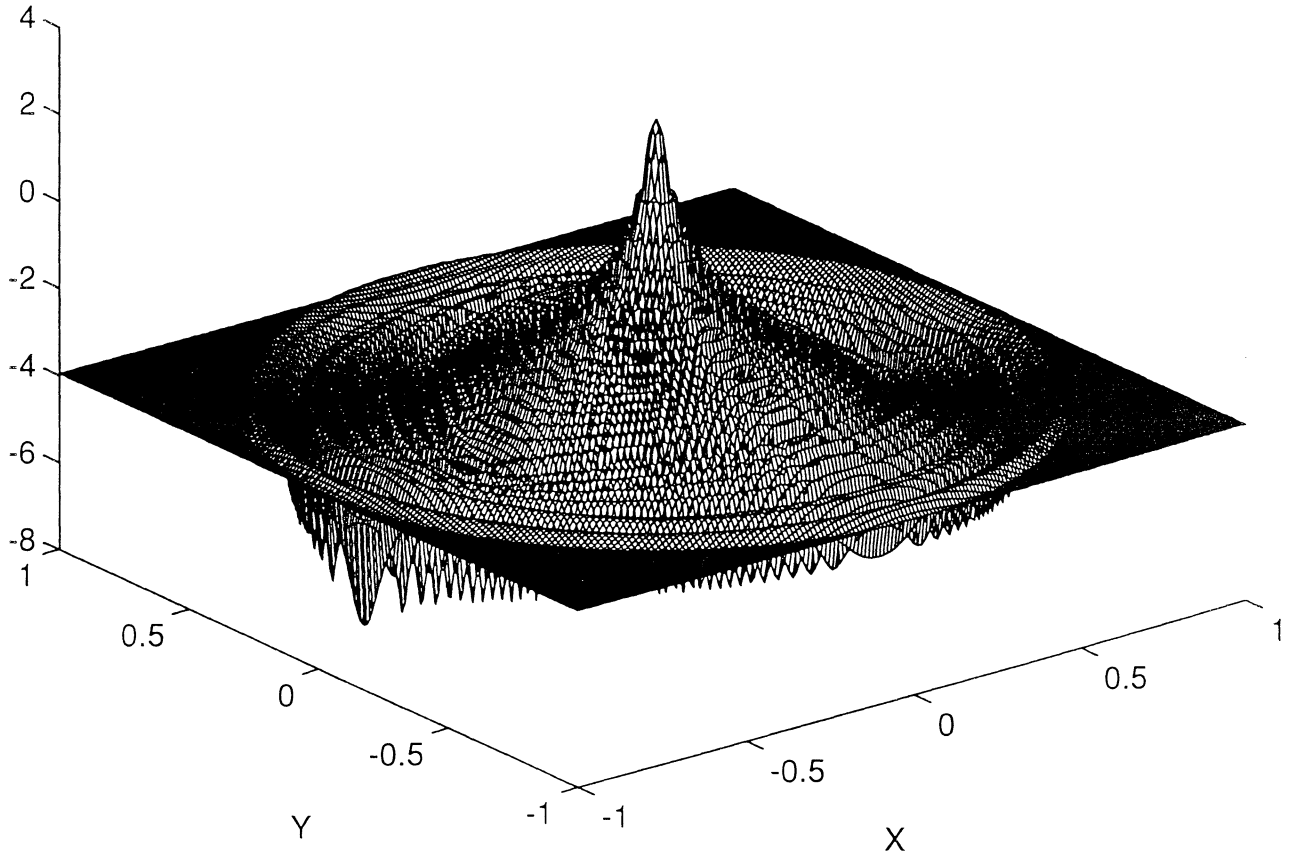
4.3. Evaluations and tests of the programs EXTENDT3 and TMMCML

Test 1. This test was made in order to investigate the influence of particle size on light scattering and distribution. The three following pages contain three-dimensional graphs, with the logarithm of the normalized differential scattering cross section plotted versus the scattering angles according to the rules described in section 3.1.3 (θ along the radius), for the spheres with size parameters ka = 20.0 (V = 14 μm^3), ka = 37.7 (V = 94 μm^3) and ka = 60.0 (V = 390 μm^3), respectively. Figure 4.5 shows the scattering probability versus the scattering deflection angle θ for these three spheres and the radially resolved reflectance and transmittance per unit area obtained from the TMMCML-simulations are presented in figures 4.6 and 4.7. Corresponding total amounts are found in table 4.4. The g-values obtained from the FORTRAN program were g = 0.9891 (ka = 20), g = 0.9940 (ka = 37.7) and g = 0.9920 (ka = 60).

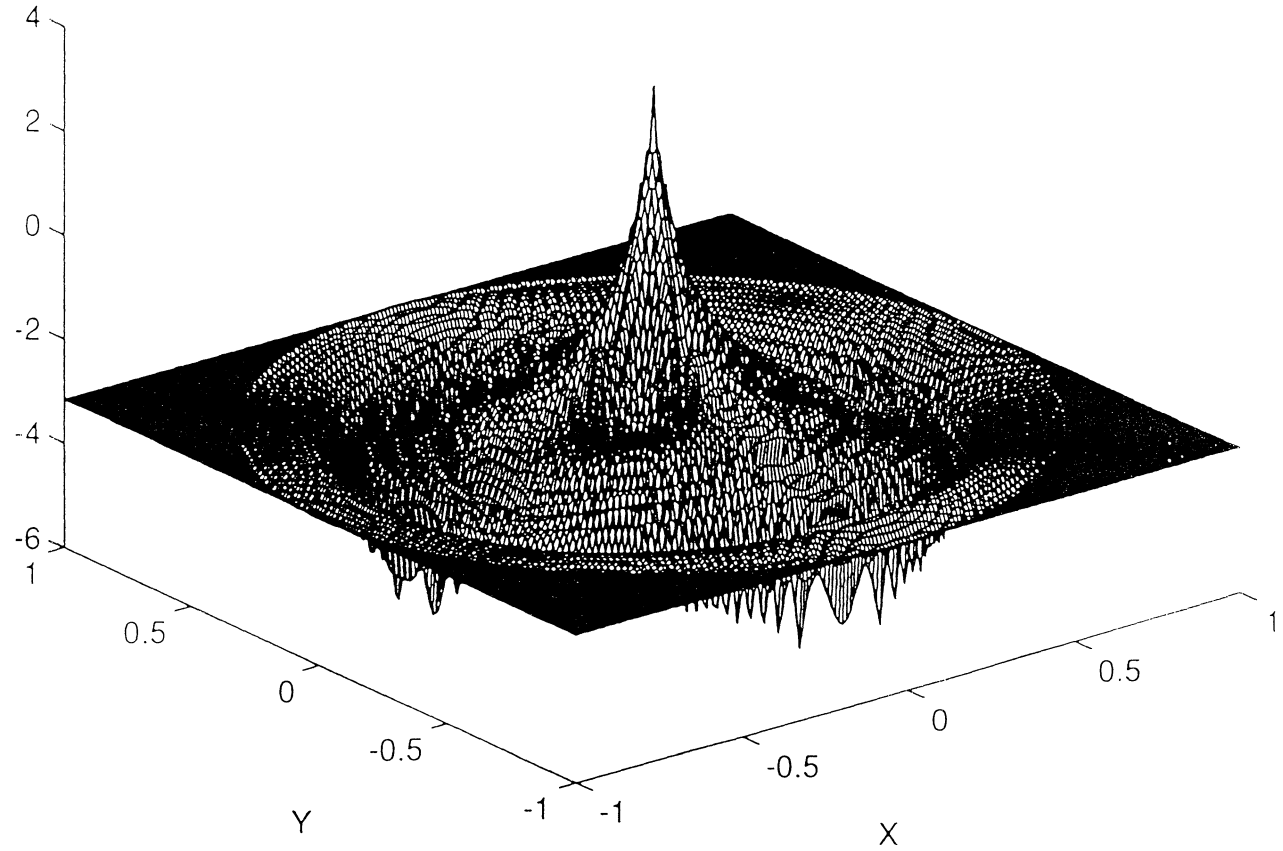
TEST 1
Logarithm of the normalized differential scattering cross section of a sphere.
The volume is 14 μm^3.



TEST 1
Logarithm of the normalized differential scattering cross section of a sphere.
The volume is 94 μm^3.



TEST 1
Logarithm of the normalized differential scattering cross section of a sphere.
The volume is 390 μm^3.



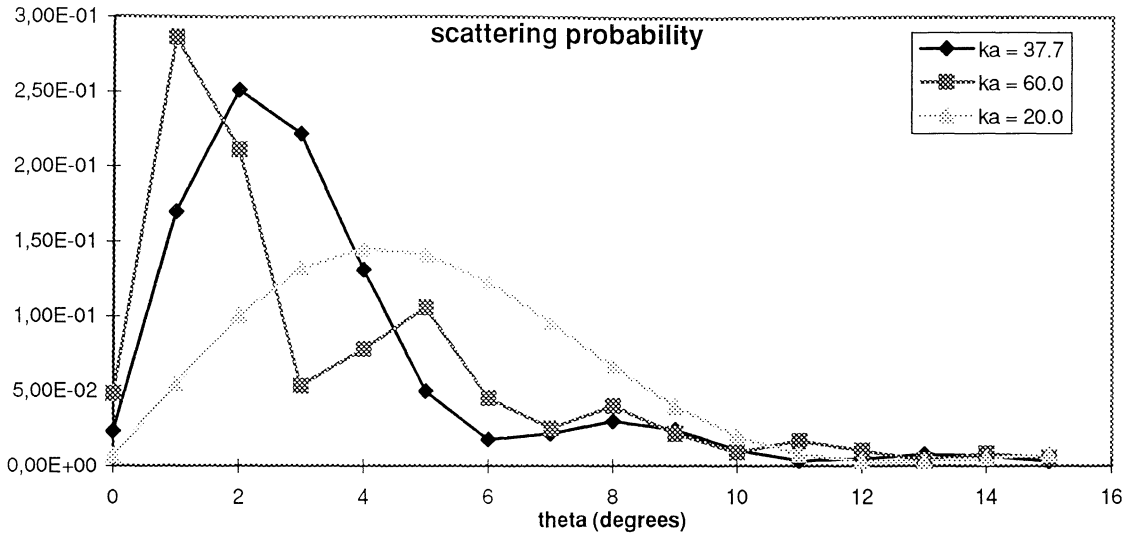


Figure 4.5. The scattering probability versus the deflection angle θ, for three spheres with size parameters ka = 37.7, 60.0 and 20.0, respectively (test 1). θ = 0° represents the scattering probability between 0° and 0.5°, whereas θ = 1° represents the interval between 0.5° and 1.5° etc.

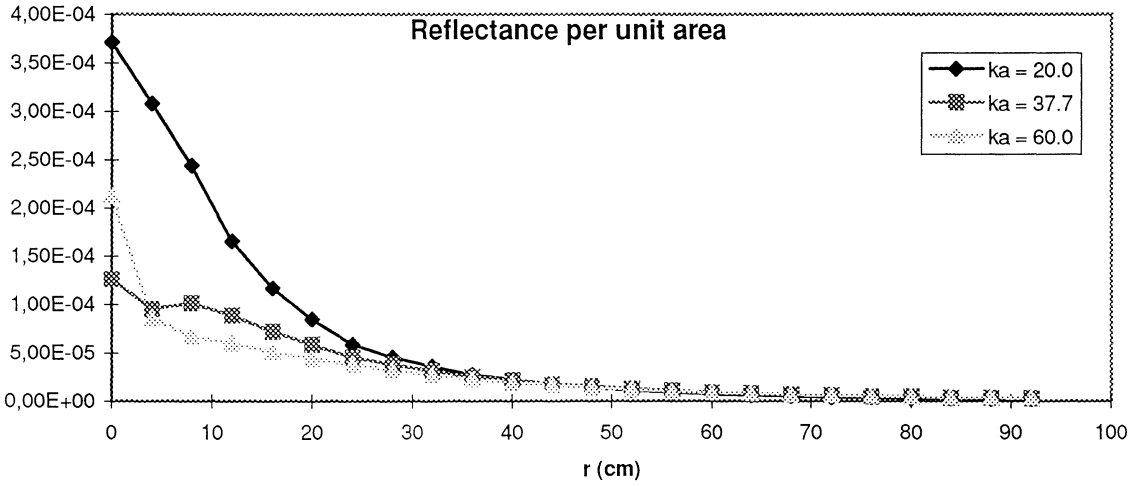


Figure 4.6. The reflectance per unit area versus the radial distance from the centre of the sample obtained from the TMMCML-simulations in test 1. The rhombs, squares and triangles correspond to scattering spheres of volumes 14μm³ (ka = 20.0), 94 μm³ (ka = 37.7) and 390 μm³ (ka = 60.0), respectively. The mean distance between the particles was 0.1 cm and the sample thickness was 100 cm.

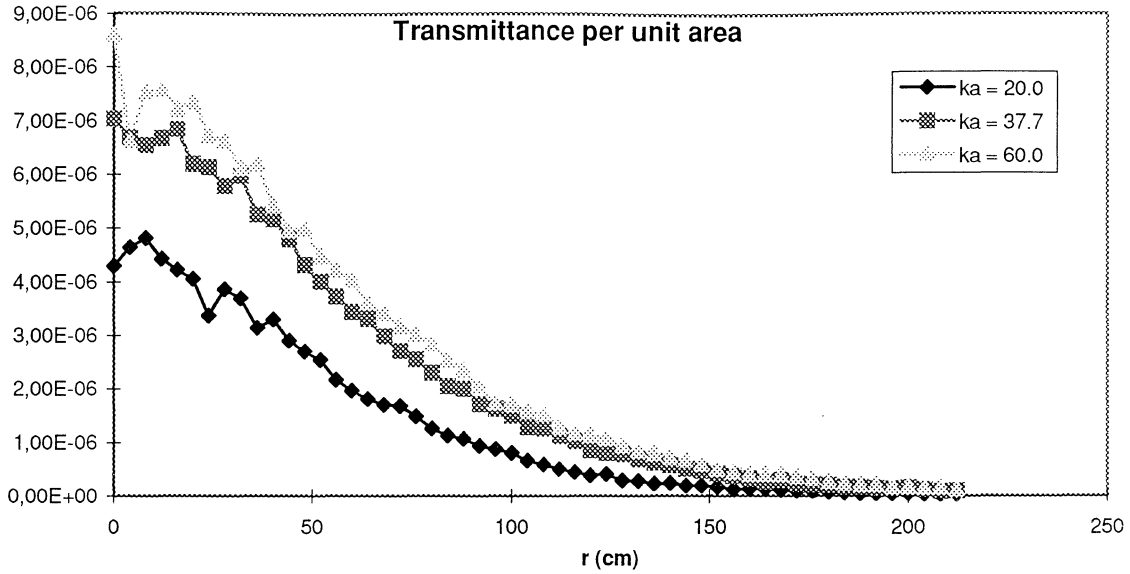
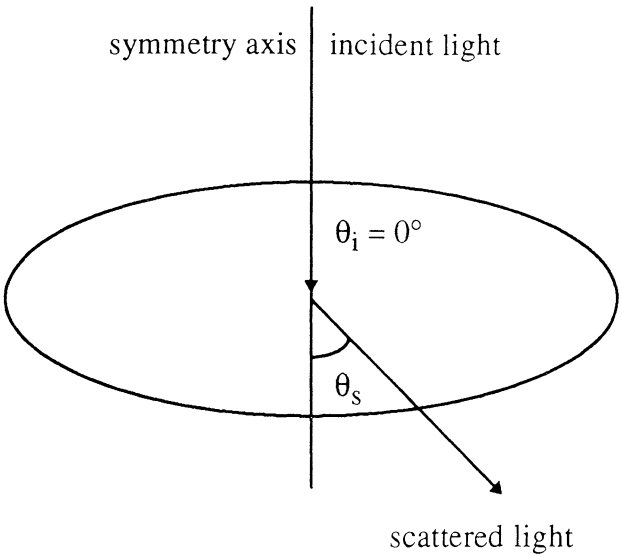


Figure 4.7. The transmittance per unit area versus the radial distance from the centre of the sample obtained from the TMMCML-simulations in test 1. The rhombs, squares and triangles correspond to scattering spheres of volumes 14μm^3 (ka = 20.0), 94 μm^3 (ka = 37.7) and 390 μm^3 (ka = 60.0), respectively. The mean distance between the particles was 0.1 cm and the sample thickness was 100 cm.

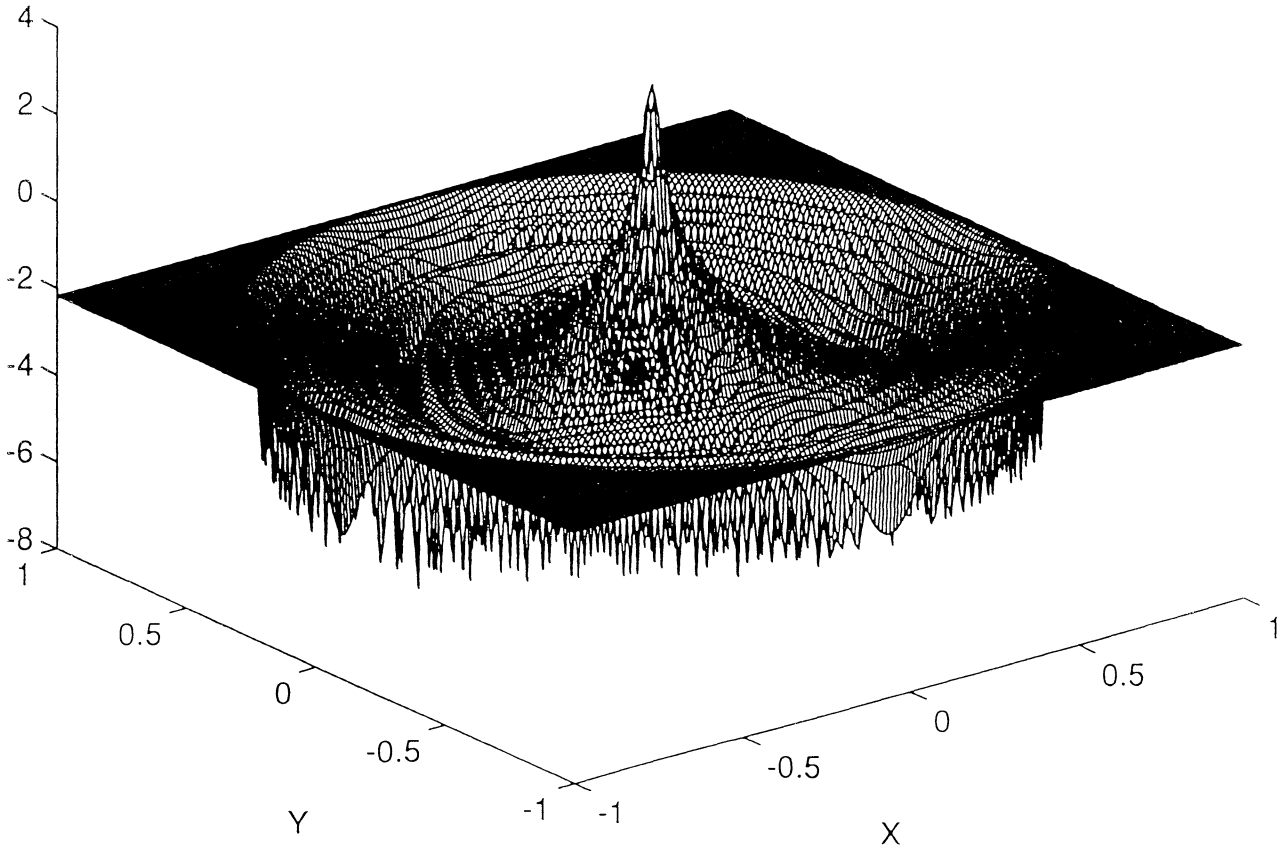
	ka = 20.0	ka = 37.7	ka = 60.0
#Specular reflectance	0.04274	0.04274	0.04274
#Diffuse reflectance	0.60380	0.51136	0.47667
#Absorbed fraction	0.26809	0.28513	0.28862
#Transmittance	0.08538	0.16077	0.19198

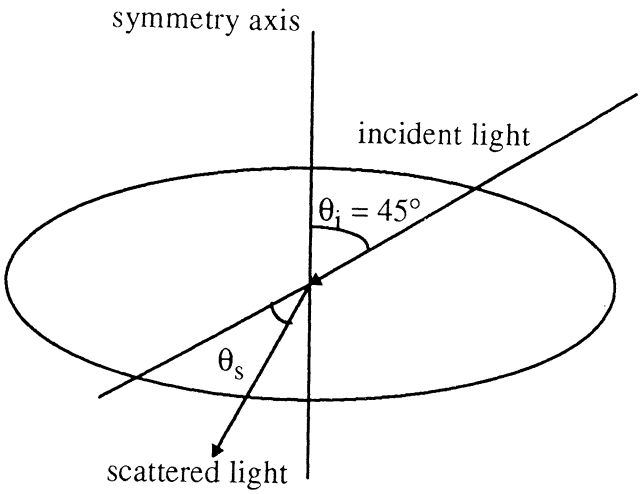
Table 4.4. The parts of reflected, absorbed and transmitted light obtained by TMMCML-simulations in test 1, investigating the influence of particle size on the light distribution.

Test 2. In this test the influence of particle shape and orientation on light scattering and distribution was studied. Three-dimensional graphs, with the logarithm of the normalized differential scattering cross section plotted versus the scattering angles according to the rules described in section 3.1.3, are presented on the three following pages for an oblate spheroid with the dimension of a human blood cell. The incident angles are 0°, 45° and 90°, respectively. These pictures show much smaller variations for the scattering azimuthal angle ϕ, than for the deflection angle θ. Figure 4.8 shows the scattering probability versus the scattering deflection angle θ, for these three cases and the obtained corresponding g-values were 0.9967, 0.9960 and 0.9887, respectively.



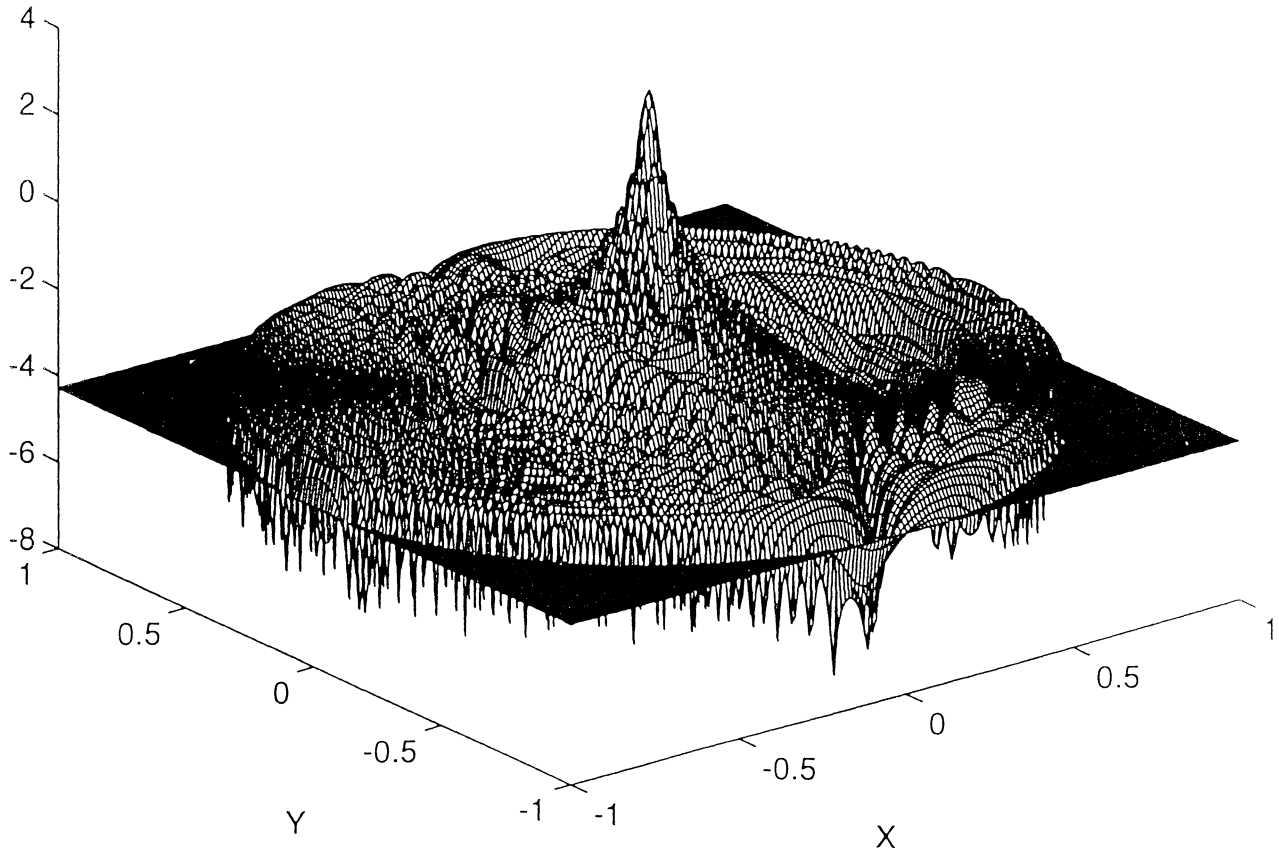
TEST 2
Logarithm of the normalized differential scattering cross section of an oblate spheroid.
The incident angle is 0 °.

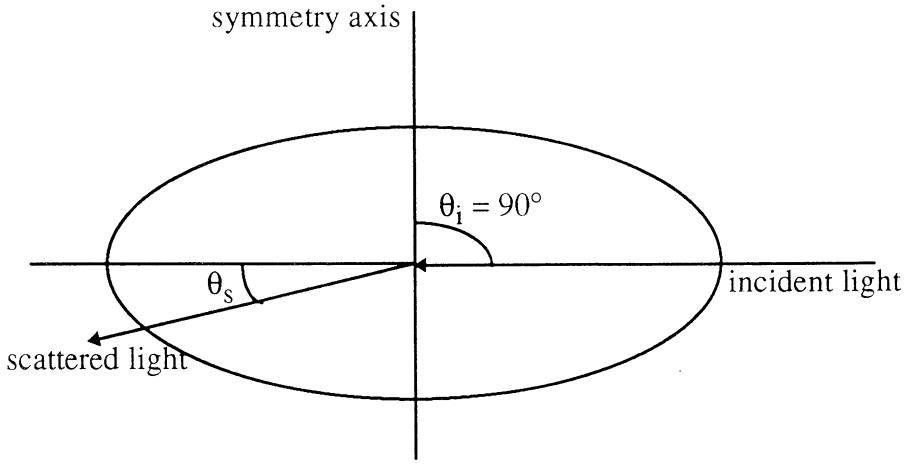




TEST 2

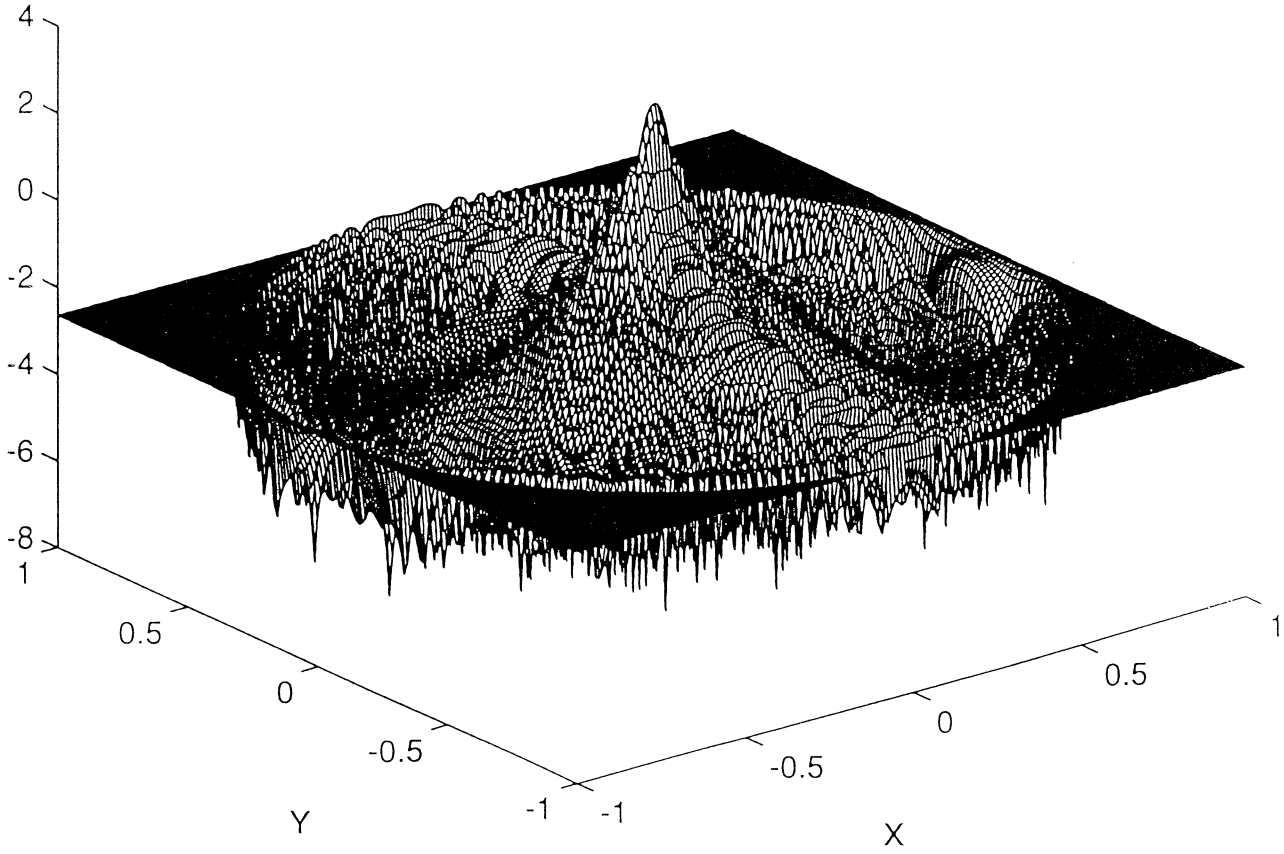
Logarithm of the normalized differential scattering cross section of an oblate spheroid.
The incident angle is 45 °.





TEST 2

Logarithm of the normalized differential scattering cross section of an oblate spheroid.
The incident angle is 90 °.



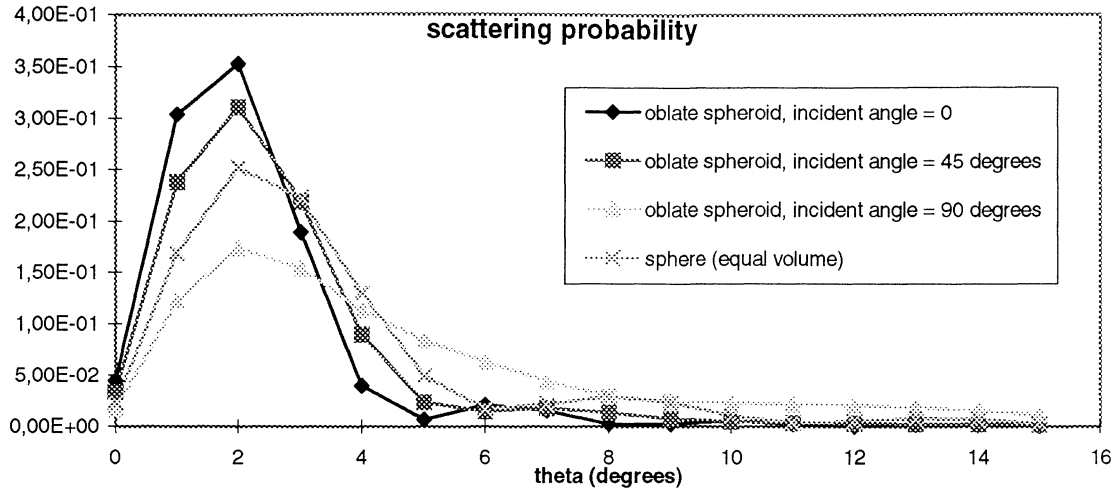


Figure 4.8a. The scattering probability versus the deflection angle θ for three different orientations of an oblate spheroid with the dimension of a blood cell. The incident angles are 0°, 45° and 90°, respectively. The cross symbols represent the scattering probability by a sphere with an equal volume (94 μm³). θ = 0° represents the scattering probability between 0° and 0.5°, whereas θ = 1° represents the interval between 0.5° and 1.5° etc.

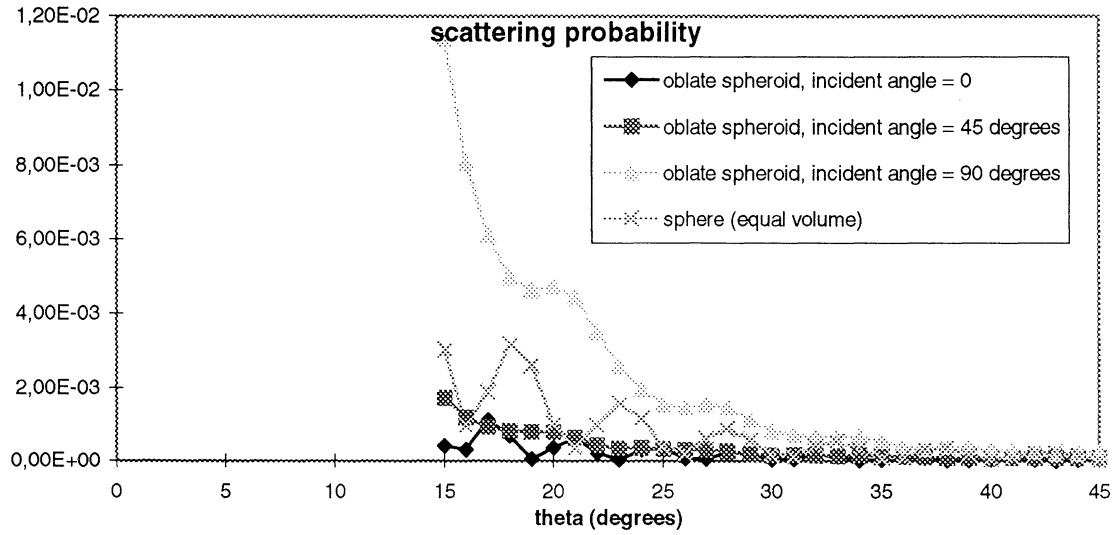


Figure 4.8b. Continuation of figure 4.8a for larger radial distances.

The radially resolved transmittance per unit area obtained from the TMMCML-simulations for the different orientations of the spheroids are presented in figure 4.9. Corresponding total amounts are found in table 4.5. The different symbols in the graph in figure 4.9 and the different columns in table 4.5 represent the different orientations of the scattering spheroids.

Orientation	rando-mized	(1,0,0)	(0,1,0)	(0,0,1)	(1,1,1)	(1,1,1) st.dev.=0.5	spheres
#Specular reflectance	0.04274	0.04274	0.04274	0.04274	0.04274	0.04274	0.04274
#Diffuse reflectance	0.00777	0.00772	0.00854	0.02942	0.00870	0.00870	0.51136
#Absorbed fraction	0.12497	0.15070	0.15059	0.11829	0.13419	0.12746	0.28513
#Transmit-tance	0.82453	0.79885	0.79814	0.80956	0.81437	0.8211	0.16077

Table 4.5. The parts of reflected, absorbed and transmitted light obtained by the TMMCML-simulation in test 2, using different orientations of the scattering spheroidal particles. Each column represents one sample containing spheroidal particles and the particles have different orientations in the different samples. The orientation of the particle symmetry axis is described by the three values on the first row, where the direction (0, 0, 1) is parallel with the incident light. If there is a deviation in the orientation, this is denoted by st.dev. The last column corresponds to scattering spheres with an equal volume.

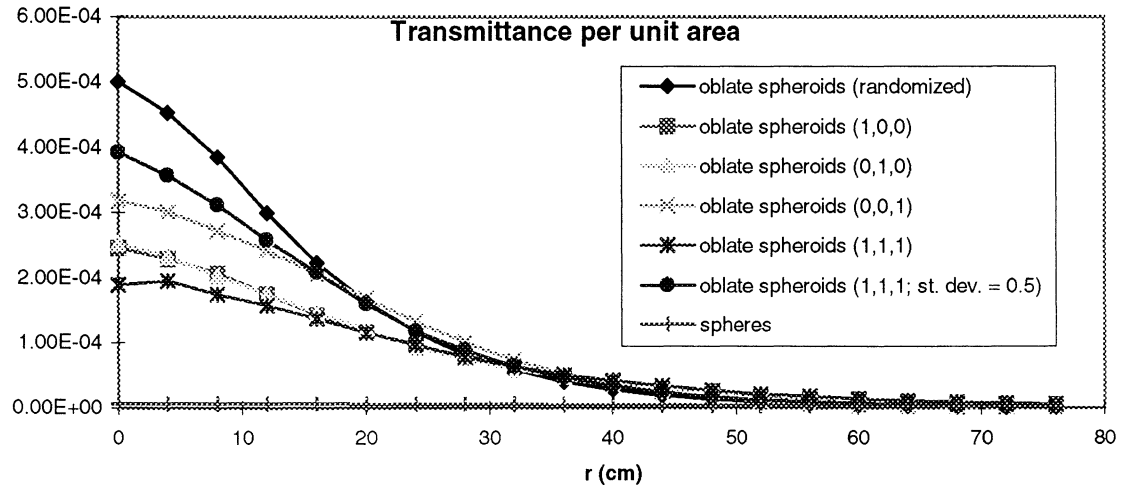


Figure 4.9a. The transmittance per unit area versus the radial distance from the centre of the sample in the TMMCML-simulations in test 2, using samples containing spheroids in different orientations in the different cases. The different symbols represent the orientations of the scattering spheroidal particles. The small strokes in the lower part of the figure represent the transmittance per unit area obtained by a simulation using spheres of equal volume as scattering particles.

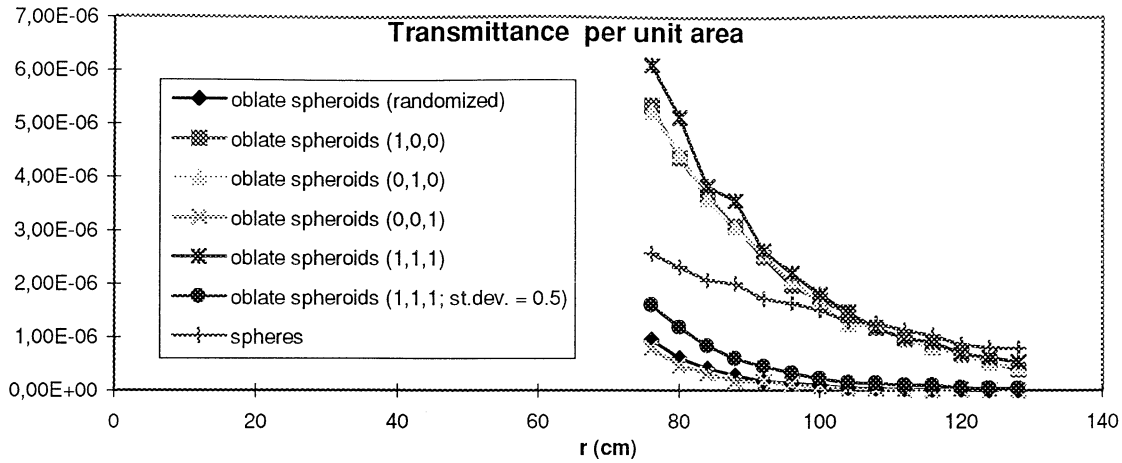


Figure 4.9b. Continuation of figure 4.9a for larger radial distances.

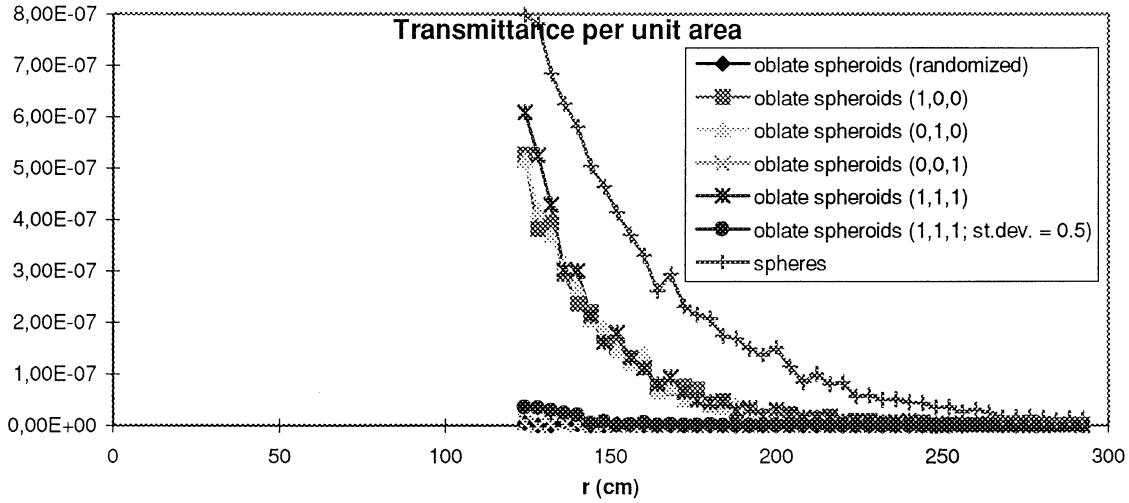


Figure 4.9c. Continuation of figure 4.9b for larger radial distances.

Test 3. The amounts of reflected, absorbed and transmitted light obtained by two TMMCML simulations, using different average distances, 0.2 cm and 0.05 cm, between the scattering particles, are shown in tables 4.6 and 4.7.

Orientation	rando-mized	(1,0,0)	(0,1,0)	(0,0,1)	(1,1,1)	(1,1,1) st.dev.=0.5	spheres
#Specular reflectance	0.04274	0.04274	0.04274	0.04274	0.04274	0.04274	0.04274
#Diffuse reflectance	0.03362	0.03247	0.03223	0.04042	0.03131	0.03433	0.05911
#Absorbed fraction	0.12335	0.14856	0.15162	0.10265	0.12277	0.12501	0.14840
#Trans-mittance	0.80030	0.77623	0.77341	0.81419	0.80319	0.79792	0.74975

Table 4.6. The parts of reflected, absorbed and transmitted light obtained by the TMMCML-simulations in test 3, investigating the influence of the distance between the scattering particles on the light distribution. The results correspond to a simulation where meanstep = 0.2 cm. Different orientations of the scattering spheroidal particles were used. The last column corresponds to scattering spheres with an equal volume. The sample thickness was 5 cm.

Orientation	rando-mized	(1,0,0)	(0,1,0)	(0,0,1)	(1,1,1)	(1,1,1) st.dev.=0.5	spheres
#Specular reflectance	0.04274	0.04274	0.04274	0.04274	0.04274	0.04274	0.04274
#Diffuse reflectance	0.00829	0.00753	0.00756	0.02516	0.00923	0.00876	0.06356
#Absorbed fraction	0.41788	0.47328	0.47316	0.37935	0.42021	0.42293	0.58636
#Trans-mittance	0.53110	0.47645	0.47654	0.55277	0.52783	0.52558	0.30735

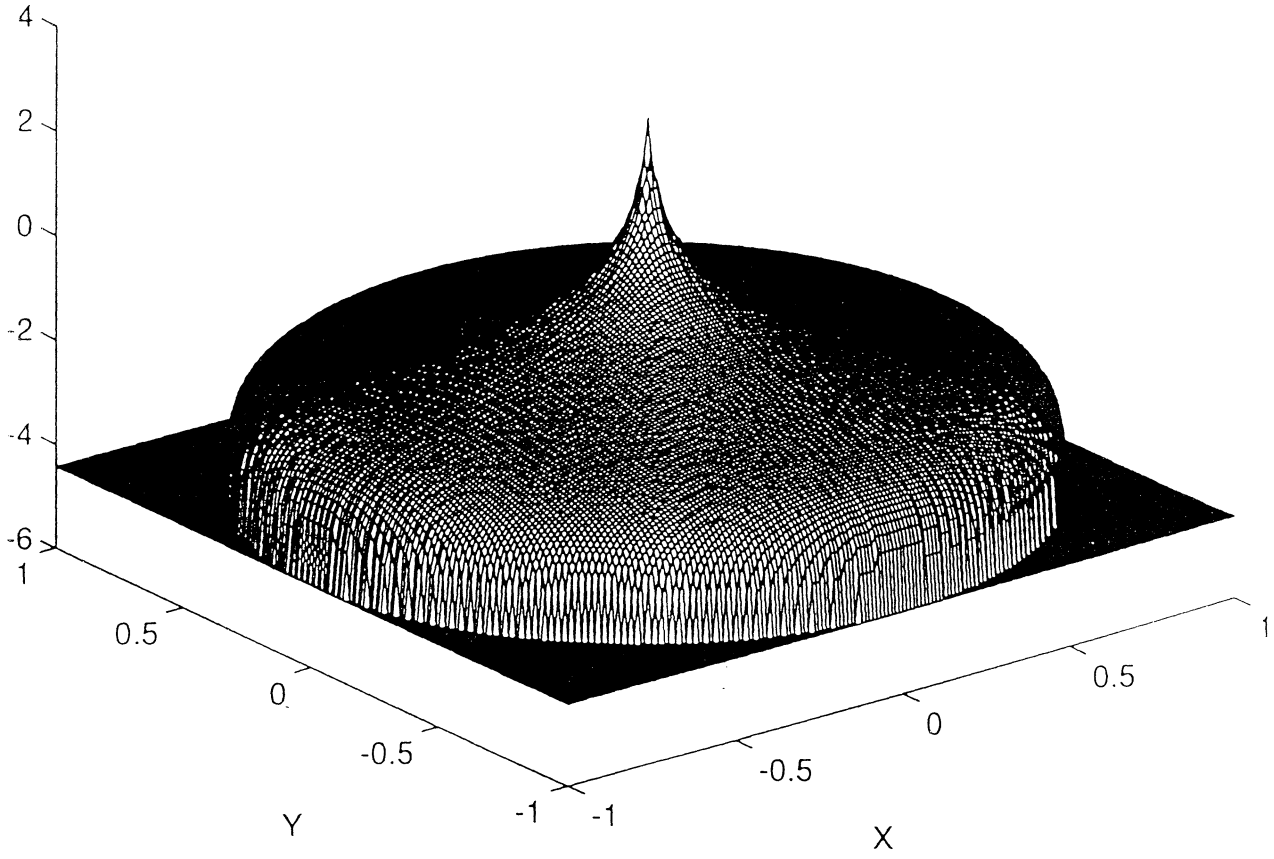
Table 4.7. The parts of reflected, absorbed and transmitted light obtained by the TMMCML-simulations in test 3, investigating the influence of the distance between the scattering particles on the light distribution. The results correspond to a simulation where meanstep = 0.05 cm. Different orientations of the scattering spheroidal particles were used. The last column corresponds to scattering spheres with an equal volume. The sample thickness was 5 cm.

Obviously, the reflected and transmitted parts of the light increase when the distance between the scattering particles increases. The absorbed part decreases.

4.4. Comparison with original Monte Carlo model

A three-dimensional picture representing the Henyey-Greenstein phase function is shown on next page. The representation of the scattering angles on a planar grid was described in section 3.1.3 (the scattering deflection angle θ along the radius). The probability values in each grid point are normalized in order to obtain a normalized total scattering cross section equal to the sphere with a volume of an erythrocyte ($\sigma_s/\pi a^2 = 3.1$).

Henyey-Greenstein distribution for g = 0.994.



The radial resolutions of the reflected and transmitted light for the simulation of the original Monte Carlo program are shown in figures 4.10 and 4.11, as well as the results from the corresponding TMMCML-simulation, using spheres as scattering particles. The reflected, absorbed and transmitted parts of the light are displayed in table 4.8.

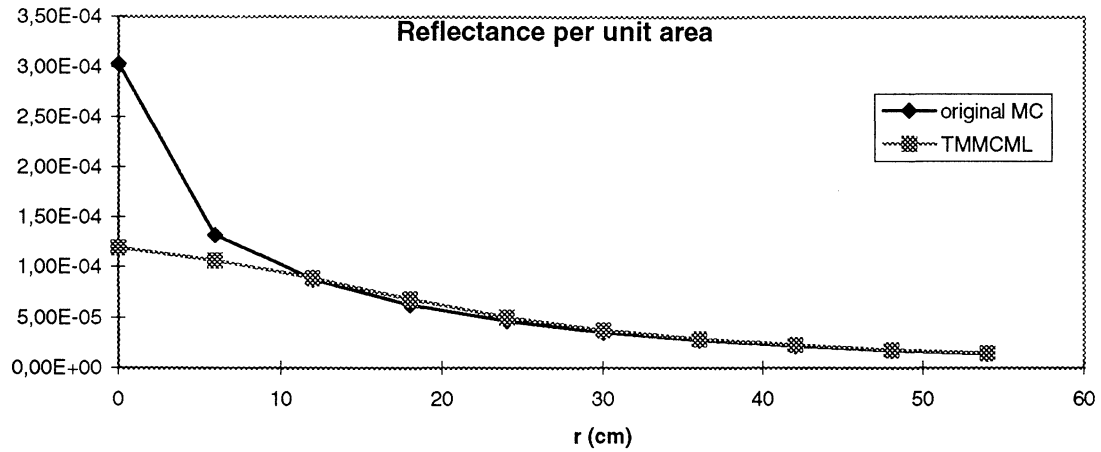


Figure 4.10a. The reflectance per unit area versus the radial distance from the centre of the sample, comparing the original Monte Carlo model with the TMMCML program, using spheres as scattering particles. The rhombs correspond to the original MCML program and the squares to TMMCML. The sample thickness was 200 cm and the optical properties in the original program were μ_a *= 0.0001* cm^{-1}, μ_s *= 9.9999* cm^{-1} *and g = 0.9940. Corresponding parameters in the modified version were meanstep = 0.1 cm (meanstepdev = 0.02 cm) and abscross = 0.00001.*

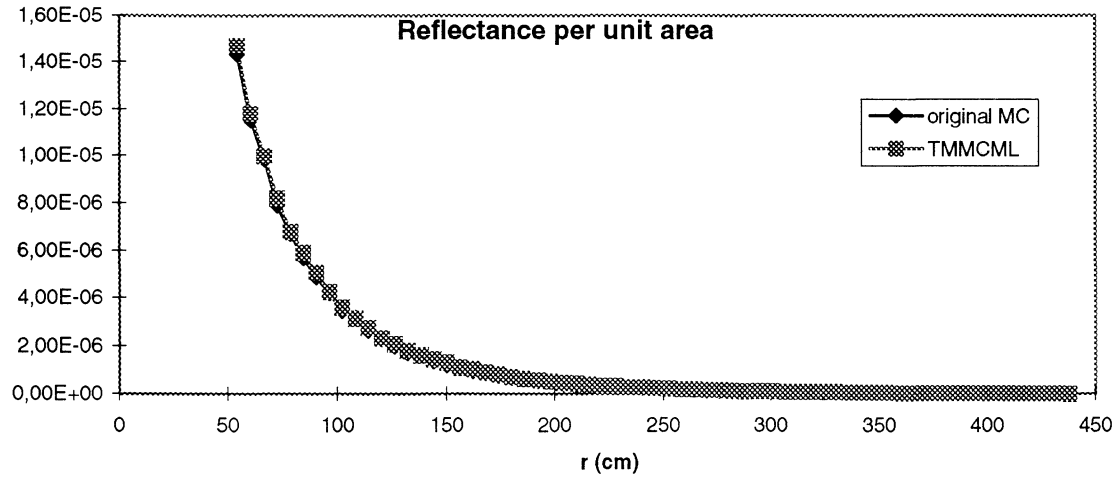


Figure 4.10b. Continuation of figure 4.10a for larger radial distances.

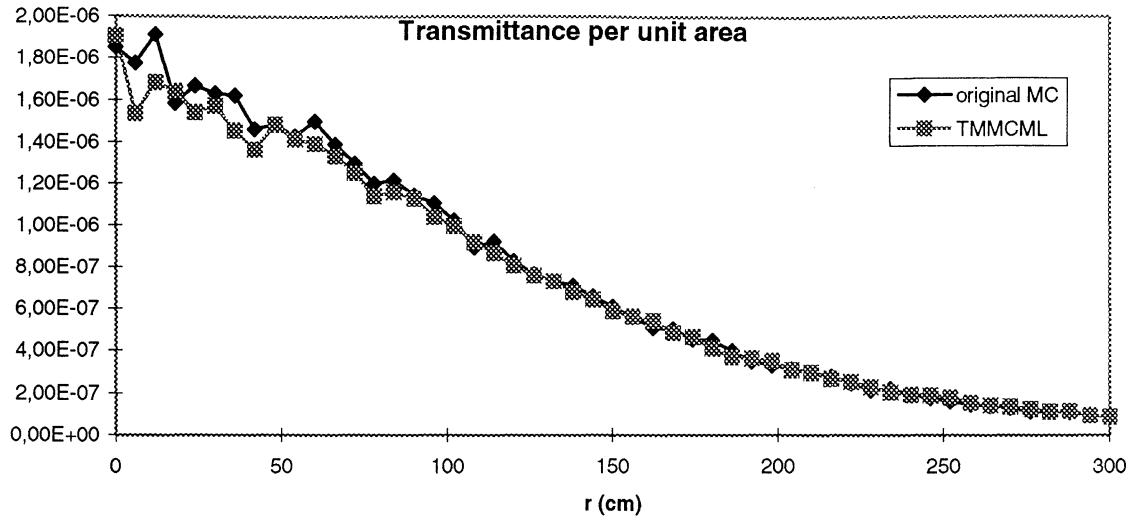


Figure 4.11. The transmittance per unit area versus the radial distance from the centre of the sample, comparing the original Monte Carlo model with the TMMCML program, using spheres as scattering particles. The rhombs correspond to the original MCML program and the squares to TMMCML.

	original MC	TMMCML
# Specular reflectance	0.04274	0.04274
# Diffuse reflectance	0.74134	0.73988
# Absorbed fraction	0.07153	0.07399
# Transmittance	0.14439	0.14340

Table 4.8. The parts of reflected, absorbed and transmitted light obtained by simulations of the original Monte Carlo program and the TMMCML program, using spheres as scattering particles.

The difference in transmitted and reflected light is less than 5% at all radial distances except close to origo. The simulation time in the modified version is almost four times that used by the original program (34.0 hours for the modified program and 9.6 hours for the original program).

5. Discussion and conclusions

5.1. Comments to the tests and evaluations

5.1.1. Testing program MCMLBOUN, due to boundary conditions

The conclusion of this test, when investigating the influence of the lateral extension of the sample on the light distribution, is that for normal experimentally used tissue samples of common thickness, in the order of millimetres, the extension of the sample ought to be at least a few centimetres in each direction, if the original Monte Carlo program, assuming infinitely wide layers, should be a correct model. Higher values of μ_s and lower values of g require larger samples. The refractive indices of the sample and the surrounding medium influence the necessary size of the sample as well, as a big difference between these two indices results in more light reflected back towards the surrounding medium into the sample instead of escape in the borders. The tests confirm reasonable results by the program MCMLBOUN and the fact that this modified version requires more simulation time for samples, possible to approximate as infinite than the original program. The increase in simulation time depends on more necessary computations in MCMLBOUN, as the distance from the position of the photon package to three different boundaries, instead of one in the original program, must be calculated for every new step. The small deviations in reflectance per unit area close to origo in figure 4.1, depend on the low number of photons emerged in this area.

5.1.2. Testing limitations and accuracy

These tests, investigating needed accuracy, assure that it is sufficient to assign the number of scattering deflection angles, nbrthsc, a value of 46, when the scattering particles are spheres. The results show also that the number of incident angles, nbrangin does not need to exceed 91 and that the polarization direction of the incident light is arbitrary. With these facts in mind together with the smaller dependence of the azimuthal scattering angle ϕ (test 2 in section 4.2) and information of the computer capacity, reasonable input parameters to program TMMCML are nbrangin = 91, nbrthsc = 181 and nbrphsc = 37. The last parameter represents the number of azimuthal scattering angles ϕ. The small deviations close to origo in the distributions of transmitted and reflected light (figures 4.3 and 4.4) depend on the low number of emerged photons in these areas. Finally, the limits of size and shape of the scattering particles in order to obtain converged solutions by program EXTENDT1 agree very well with those obtained by Mishchenko *et. al.*[26], and are much better than the values achieved by Hill *et. al.*[27], probably using non-extended variables.

5.1.3 Evaluations and tests of the programs EXTENDT3 and TMMCML

Test 1 shows the well-known phenomenon[28], that large spheres tend to scatter more light in the forward direction than small ones, comparing three spheres with size parameters ka = 20.0, 37.7 and 60.0 (V = 14 μm^3, 94 μm^3 and 390 μm^3). The three-dimensional figures on pages 28-30, as well as the scattering probability distribution in figure 4.5 indicate this property. The obtained g-values (0.9891, 0.9940 and 0.9920) for the three spheres, respectively, show that this value is lower for the largest sphere than for the sphere with a size parameter of 37.7. The g-factor, which only describes the mean value of the cosine of

the scattering deflection angle θ, is thus no sufficient quantity of describing the distribution of light scattered by a particle. Neither is the forward scattering. The oscillating patterns of the scattering probability distributions depend on interference phenomena. The TMMCML-simulations made for the three mentioned spheres, as scattering particles, were mainly performed to check the program TMMCML rather than to obtain any exact results. The mean distance between two scattering particles and the absorption in each of these particles were assumed to be equal in these three simulations. The first assumption results in different volume proportions of particles in relation to surrounding medium and the second is wrong, while the absorption cross section is volume dependent[29]. Nevertheless, the TMMCML-simulations manifest the fact, that more light is forward scattered when larger particles are used (figures 4.6 and 4.7). The high value of reflectance per unit area in origo (figure 4.6) for the largest sphere is probably due to the higher complete back scatter value for this particle. The program TMMCML yields thus reasonable results in well accordance with general scattering theory[28].

When comparing the results of scattering by an oblate spheroidal particle, for three different incident angles (0°, 45° and 90°) in **test 2**, it is evident that a smaller incident angle results in more light scattered straight forward than a larger incident angle. This is shown in the three-dimensional figures on pages 33-35, the scattering probability distribution in figure 4.8, as well as indicated by the obtained g-values (0.9967, 0.9960 and 0.9891 in the three cases, respectively) and is due to the larger area of the particle perpendicular to the direction of the incident wave for a smaller incident angle[30]. The back scattering for an incident angle of 0° is about 100 times higher than that for the case of 45°. When the incident angle is 90°, the back scattering is also quite high. These results may depend on the higher reflectance in straight opposite direction, which occurs when the light falls perpendicular towards a surface, than for the case of slant incident light, *e.g.* 45°. However, influence on the probability of back scattering of particle size and shape is very irregular[30] and is hard to predict. Another discovery, from the three-dimensional figures is that the scattering dependence of the azimuthal angle ϕ, seems to be much less than that of the deflection angle θ. Not totally negligible, though.

Several interesting facts are shown in the results of the TMMCML-simulations in **test 2**, using oblate spheroids in different orientations as scattering particles. The most remarkable property is the much lower amounts of reflected light and higher amounts of transmitted light, obtained when the oblate spheroids were used as scattering objects compared to when spheres were used, irrespective of the orientations of the spheroids (see table 4.5). This observation has earlier been described and discussed in case of randomly oriented spheroids[25]. The total amount of absorbed light, which indicates the interaction sites and events of the photons in the medium is also much less in the case of spheroids. The results presented in table 4.5, show that the absorbed fraction of light using spheroids was about half the fraction obtained, when using spheres as scattering particles.

When comparing two simulations with the same orientation of the particles with an uncertainty (normal distribution with a given standard deviation) in the direction in one of them (black stars and filled circles in figure 4.9), the total transmittance and the light transmitted close to origo are higher in the standard deviation case. Far from origo, the non-standard deviation case yields more transmitted light. This is probably a consequence of once the light has been scattered in a quite large angle, it tends to continue in this direction as it is more probable. In the standard deviation case, the photon package will

have a higher probability to hit a spheroid at an angle with lower g-factor (*i.e.* at an incident angle close to 90°), than in the non-standard deviation case. The probability of scattering in wide angles will therefore be higher and more scattering events result in wide scattering angles. This can lead to that it is more rare that photons are transmitted far from the centre of the sample.

The results of two simulations using spheroids oriented with their symmetry axis in the two directions perpendicular to the incident beam, respectively (squares and triangles in figure 4.9) are almost identical, which is expected due to the axial symmetry of the particle. If the symmetry axis of the spheroids is parallel to the incident light (grey stars in figure 4.9), both the total reflectance and the total transmittance will be higher. This is in agreement with the results from three-dimensional pictures (pages 33-35) of the angular scattering by an oblate spheroid for different incident angles. They show higher values of forward scattering as well as back scattering for an incident angle of 0° than for other angles (45° and 90°).

The fact that a sample with randomly oriented spheroids (filled rhombs in figure 4.9) results in more light transmitted straight forward than for spheroids with their symmetry axis parallel to the incident light (grey stars in figure 4.9) seems rather confusing at first sight, as the highest g-factor is obtained for an incident angle of 0°. Nevertheless, taking into consideration that the total amount of transmitted light is lower in the latter case, due to the high probability of back scattering and the facts mentioned before, that once the light has been scattered in quite a large angle, it tends to continue in this direction more probable in the latter case than in the first, this phenomenon can seem reasonable. Even more surprising seems a comparison between the results of the TMMCML-simulations using spheres (small grey strokes in figure 4.9) and spheroids oriented with their symmetry axis in the direction perpendicular to the incident beam, as scattering particles. Results from **test 1** and **test 2,** obtained using a small FORTRAN program, shows g = 0.9940 for the spheres and g = 0.9891 for the spheroids, when the incident angle equals 90°. Thus, less light would actually be scattered in the forward direction using spheroids oriented with the symmetry axis perpendicular to the incident beam than in the case with spheres. However, the anisotropy factor, g, describes only the mean value of the cosine of the deflection scattering angle θ, but not the scattering probability distribution. In fact, the scattering probability of a scattering angle between 5° and 35° is much higher for this kind of spheroids than for spheres, which can be seen in figure 4.8. The incident angle is around 90° only in the first interaction points and causing other directions of the incident angle at later interactions. The explanation concerning samples with no deviations in the orientations of the spheroids is also most likely in this case, due to the high transmission far from origo, using the spheroids as scattering particles (grey stars in figure 4.9).

Comparing the results of **test 3**, where two different distances (0.2 cm and 0.05 cm) between the particles in TMMCML-simulations, reasonable results are obtained *i.e.* more absorption and less transmission for the shorter particle distance due to more interaction occasions. The higher amounts of reflected light for samples containing spheroids, with larger distance in-between are probably due to higher amounts of light reflected in the lowest boundary, which is transmitted through the sample once more.

5.1.4. Comparison with original Monte Carlo model

When simulating multiple light scattering by executing TMMCML with scattering spheres and the original Monte Carlo program with identical values of the g-factor, the mean distance between two interaction points and the absorption during an interaction, the results show that light distribution is independent the scattering phase function Henyey-Greenstein versus T-matrix computation for spheres if the photons are scattered many times. The deviation in the distribution of reflected light close to origo (figure 4.10), is mainly caused by photons, not multiple scattered. The small deviations in the transmittance diagram (figure 4.11) might be due to the low number of photons in this areas. The difference in simulation time indicates the more complicated computations of the scattering events, required by the TMMCML program.

5.2. Summary and concluding remarks

The modified programs have been used to simulate the distributions of transmitted and reflected light of samples with a finite extension containing spheroidal scattering particles and the results presented here correspond well with those by others within this field[25,26,28,30].

One purpose of this work was to get a more accurate model for predicting light propagation in human blood. The main disadvantage of the developed model is the large distance between the scattering particles needed due to the assumption of far-field (eq. 2.13). The original Monte Carlo model used, is a conventional way of describing light propagation in tissue[20,31,32], in spite of its insufficient description on the microscopic level. Monte Carlo simulations, using the Henyey-Greenstein function, has been proved to be in a good agreement to measured values of reflected and transmitted light in human tissue[33]. The most important assumption by the Monte Carlo model is that the light scatters according to the Henyey-Greenstein function.

In the tests made during this work, no agreement have been obtained between the results of the program TMMCML with oblate spheroids as scattering particles and the original Monte Carlo program with corresponding input parameters. One explanation for the case of short distances between the blood cells (normal human blood) might be that this distance is much too small to assume that the incident field of one cell is equal to the scattered far-field from the previous cell. Nevertheless, this cannot be the whole truth, as there is no agreement when the distance between the cells is increased (**Test 2**). One possible reason to this fact is that the original Monte Carlo method and the Henyey-Greenstein phase function maybe are no longer valid in extremely diluted blood, in spite of a sample thickness resulting in multiple scattering. Thus, the modified model would in this case be a better description of the light propagation. The Monte Carlo model might even not be valid for blood at all, in spite of its good agreement for tissue in general. Another reason to the mentioned disagreements could be that the shape deviation between a blood cell and an oblate spheroid might influence the scattering process too much to make such an approximation. This can be due to different distribution of light scattering, different diffraction and occurrence of interference[20] or some resonance phenomenon taking place. Finally, the assumptions made in the work such as angular resolution of incident and scattered field, disregarding of the light polarization and neglected dependence of particle light absorption on the angular scattering, might have influenced the results.

Measurements made on slightly diluted blood[5], also indicate higher values of reflectance in proportion to transmittance than ever obtained by TMMCML simulations, with corresponding sample sizes and distances between the scattering spheroids. However, no measurements are made at the low concentrations required for accurate T-matrix computations. Nevertheless, some optical measurements on human blood qualitatively agree with the simulations in this work. The transmitted light for instance decrease when the blood cells were transformed from spheroidal into spherical shape[3]. This is in line with the results obtained from the simulations (figure 4.5) showing less transmitted light from the sample with spherical particles than from the one containing volume equivalent spheroids. Agreements is also obtained between measurements showing more reflected and less transmitted light with increasing blood flow[34], when the cells are less randomly oriented and the results from simulations with totally randomized versus well aligned spheroids showing more transmitted light for the former (Table 4.5).

5.3. Future

As mentioned before, the computer programs, developed during this work, can be used to evaluate the scattering in various directions from large spheroidal particles, as well as the distributions of transmitted and reflected light of a sample containing a lot of such particles in different orientations. These programs can hopefully be of use when such results are needed to interpret measurements and to understand which particle parameters influence the distributions of scattered light. The number of possible shapes and sizes of the scattering particles has increased considerably due to this work, and the application field has thus expanded.

In order to continue the investigations of scattering by blood cells and light distribution in human blood, some possible further works are suggested:

- To establish an explicit expression, $r(\theta)$ (see figure 3.1), of the shape of the blood cell (figure 3.4) and to obtain a converged solution for such a particle by the program T1. The first part would be possible, modifying an available implicit expression[22], while the convergence probably requires more advanced computers and further programming efforts.
- Another modification of the programs T1 and T2[9], in order to obtain the absorption cross sections for different incident angles and the angular scattering without neglecting the absorption. Measured values of the imaginary refractive index of a blood cell are available[23] and using a compiler with extended precision for complex variables, the necessary modifications should be few.
- Measurements of transmitted and reflected light, from a sample containing blood, or another suspensions with spheroidal particles diluted to such an extension that the assumption of far-field, described earlier, is fulfilled. The idea would be to compare the results obtained by the program TMMCML with measured values.

6. Acknowledgements

First of all, I would like to thank my supervisor Annika Nilsson for her good advices, positive attitude to the emerged problems and for always having time for all my questions.

Special thanks to Anders Karlsson at the Department of Electromagnetic Theory, Lund Institute of Technology, who shared his knowledge about the T-matrix method and the belonging computer programs.

Many thanks to Stefan Andersson-Engels for taking part in stimulating discussions, as well as the outlining of this project and to Jörgen Carlsson who helped me, with the large-scaled calculations on his computer.

I am grateful to my family, who has supported me during my work.

Finally, I would like to thank Professor Sune Svanberg, who woke my interest for applications of atomic physics and all the other staff members at the division, for helping me solving different kind of problems and many pleasant conversations.

7. References

1. B. W. Henderson and T. J. Dougherty, *Photodynamic Therapy - Basic principle and clinical applications* (Marcel Dekker, INC. New York, USA, 1992).

2. L. Wang and S. L. Jacques, "Monte Carlo modeling of light transport in multi-layered tissues in standard C", (1992).

3. A. M. K. Nilsson, G. W. Lucassen, W. Verkruysse, S. Andersson-Engels and M. J. C. van Gemert, "Changes in optical properties of human whole blood in vitro due to slow heating", submitted to Photochem. Photobiol. (1996).

4. J. M. Steinke and A. P. Shepherd, "Comparison of Mie theory and the light scattering of red blood cells", Appl. Opt. **27,** 4027-4033 (1988).

5. S. Kashima, A. Sohda, Y. Yagyu and T. Ohsawa, "Determination of deformability of erythrocytes by change in scattering cross section", Japanese Journal of Applied Physics part 1 **34,** 680-682 (1994).

6. H. Hahn, A. Roggan, D. Schädel, U. Stock, H. Bäumler and F. Wondrazek, "Die Optischen Eigenschaften von dicken Schichten zirkulierenden Humanblutes", Minimal Invasive Medizin **7,** 79-90 (1996).

7. J. Carlsson, P. Hellentin, L. Malmqvist and W. Persson, "The propagation of light in paper: modelling and Monte Carlo simulations", submitted to JPPS (1995).

8. J. D. Jackson, *Classical electrodynamics* (John Wiley & Sons, Inc. USA, 1975).

9. P. W. Barber and S. C. Hill, *Light scattering by particles: Computational Methods.* (World Scientific Publishing Co. Pte. Ltd. Signapore, 1990).

10. G. Kristensson, *Spridnings- och antennteori* (Lund, 1995) (in Swedish).

11. P. C. Waterman, "New formulation of acoustic scattering", The Journal of the Acoustial society of America **45,** 1417-1429 (1968).

12. W. E. Vargas, "Visible and near infrared optical properties of pigmented films", Dissertation thesis **UPTEC 95 112R** (Institute of Technology, Uppsala university, Uppsala, 1995).

13. P. C. Waterman, "Symmetry, unitarity and geometry in electromagnetic scattering", Physical review D **3,** 825-839 (1970).

14. L. G. Henyey and J. L. Greenstein, "Diffuse radiation in the galaxy", Astrophysics Journal **93,** 70-83 (1941).

15. E. Hecht, *Optics* (Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1987).

16. M. Keijzer, "Light transport for medical laser treatments", Ph.D. Thesis (Technische Universiteit Delft, the Netherlands, 1993).

17. M. I. Mishchenko, "Light scattering by size-shape distributions of randomly oriented axially symmetric particles of a size comparable to a wavelength", Appl. Opt. **32,** 4652-4666 (1993).

18. W. H. Press, S. A. Teukolsky, W. T. Vetterling and B.P. Flannery, *Numerical recipes in C* (Cambridge University Press, Cambridge, 1992).

19. W. F. Cheong, S. A. Prahl and A. J. Welch, "A review of the optical properties of biological tissues", IEEE J. Quant. Electr. **26,** 2166-2185 (1990).

20. R. Graaf, A. C. M. Dassel, M. H. Koelink, F. F. M. de Mul, J. G. Aarnoudse and W. G. Zijlstra, "Optical properties of human dermis *in vitro* and *in vivo*", Appl. Opt. **32,** 435-447 (1993).

21. G. J. Streekstra, A. G. Hoekstra, E.-J. Nijhof and R. M. Heethaar, "Light scattering by red blood cells in ektacytometry: Fraunhofer versus anomalous diffraction", Appl. Opt. **32,** 2266-2272 (1993).

22. E. Evans and Y. Fung, "Improved measurements of the erythrocyte geometry", Microvascular research **4,** 335-347 (1972).

23. L. Reynolds, C. Johnson and A. Ishimaru, "Diffuse reflectance from a finite blood medium: applications to the modeling of fiber optic catheters", Appl. Opt. **15,** 2059-2067 (1976).

24. S. Svanberg, *Atomic and molecular spectroscopy* (Springer Verlag, Heidelberg, Germany, 1992).

25. S. Asano and M. Sato, "Light scattering by randomly oriented spheroidal particles", Appl. Opt. **19,** 962-974 (1980).

26. M. I. Mishchenko and L. D. Travis, "T-matrix computations of light scattering by large spheroidal particles", Optics Comm. **109,** 16-21 (1994).

27. S. C. Hill, A. C. Hill and P. W. Barber, "Light scattering by size/shape distributions of soil particles and spheroids", Appl. Opt. **23,** 1025-1031 (1984).

28. P. Latimer and B. E. Pyle, "Light scattering at various angles. Theoretical predictions of the effects of particle volume changes", Biophysical Journal **12,** 764-773 (1971).

29. H. C. van de Hulst, *Light scattering by small particles* (Wiley, New York, 1957).

30. S. Asano, "Light scattering properties of spheroidal particles", Appl. Opt. **18,** 712-723 (1979).

31. A. M. K. Nilsson, R. Berg and S. Andersson-Engels, "Measurements of the optical properties of tissue in conjunction with photodynamic therapy", Appl. Opt. **34,** 4609-4619 (1995).

32. A. Roggan, H. Albrecht, K. Dörschel, O. Minet and G. J. Müller, "Experimental set-up and Monte-Carlo model for the determination of optical tissue properties in the wavelength range 330-1100 nm", SPIE **2323,** 21 -46 (1995).

33. S. L. Jacques, C. A. Alter and S. A. Prahl, "Angular dependence of HeNe laser light scattering by human dermis", Lasers Life Sci. **1,** 309-333 (1987).

34. L. G. Lindberg and P. Å. Öberg, "Optical Properties of blood in motion", Opt. Eng. **32,** 253-257 (1993).

35. G. Arfken, *Mathematical methods for physicists* (Academic Press, INC. San Diego, California, 1985).

APPENDIX A: The T-matrix computer programs

A.1. Computer code listing including comments

The computer code listing of the three modified T-matrix computer programs, includes some comments of the specific modifications. Remarks are inserted in the programs EXTENDT1 and MCDATA where modifications of the T1- and EXTENDT3-programs have been made. When changing program T3 to EXTENDT3, only general modifications have been made and they are all described in the beginning of program EXTENDT3. Original comments in program T1 and T3, which are not valid in the modified versions, are removed.

A.1.1. EXTENDT1

```
      program EXTENDT1
c     .....................................................................
c     .  Light Scattering by Particles: Computational Methods             .
c     .  by P.W. Barber and S.C. Hill                                      .
c     .  copyright (c) 1990 by World Scientific Publishing Co Pte Ltd      .
c     .                                                                    .
c     .  Modifications 1996 by P. Alsholm and A. Nilsson                   .
c     .....................................................................
c     .  calculate the scattering by axisymmetric dielectric objects       .
c     .  convergence test over three parameters:                           .
c     .    nrank : number of terms (matrix order)                          .
c     .    ntheta: number of integration steps                             .
c     .    nm    : number of azimuthal modes m                             .
c     .  inputs:  nrank  = matrix order                                    .
c     .           ntheta = integration steps                               .
c     .           ic = convergence case                                    .
c     .                ntheta ( = 0 ), nrank ( = 1 ), or nm ( = 2 )        .
c     .           x = size parameter (ka)                                  .
c     .           aovrb = a/b ratio                                        .
c     .           cm = complex index of refraction, (real,imag)            .
c     .                (imag is positive for absorption)                   .
c     .  ..................................................................
c     .  .               EXTENDT1 :                                        .
c     .  .  An accuracy of 32 digits is needed to obtain converged         .
c     .  .  solutions for large scatterers.                                .
c     .  .  The DEC OSF/1 compiler uses 32 digits for real                 .
c     .  .  variables(real*16) and (16+16) digits for each part of a       .
c     .  .  complex variable(double complex). We have to split the         .
c     .  .  complex variables into two real variables.                     .
c     .  .        These modifications have been made:                      .
c     .  . * no imaginary part of refractive index (erased everywhere)     .
c     .  . * all real variables are declared real*16                       .
c     .  . * all variables which have no imaginary part, if the            .
c     .  .   refraction index is real, are also declared real*16           .
c     .  . * all complex variables which are used before returning         .
c     .  .   from prcssm are split up in a real and an imaginary part.     .
c     .  .   Both parts are declared real*16                               .
c     .  . * at the end of prcssm, the critical region is passed and       .
c     .  .   the complex variables can be declared double complex          .
c     .  ..................................................................
c     .                                                                    .
c     .  EXTENDT1 : arrays are expanded to nrank = 200 and                 .
c     .             ntheta = 1000 maximum                                  .
c     ......................................................................
c     .                                -1          -1                      .
c     .  calculate [T] = [B] * [A]   =  { [A]' * [B]'}'                    .
c     .                                                                    .
c     .    use transposed matrices to permit the efficient                 .
c     .       -1                                        -1                 .
c     .     [ ]  * [ ]  operation rather than [ ] * [ ]                    .
c     .                                                                    .
c     .    (1)  obtain the transposed matrices [A]' and [B]'               .
c     .                                  -1                                .
c     .    (2)  calculate [T]' = [A]' * [B]'                               .
c     .                                                                    .
c     .    (3)  transpose [T]' to obtain [T]                               .
c     .                                                                    .
c     ......................................................................
c     .                       EXTENDT1 :                                   .
```

```
c     .                                                              .
c     . The size of a,b,a2,b2,aa2,bb2,are,aim                        .
c     .             bre,bim,breold,areold,tmat is (2*nrankmax)^2     .
c     . The size of cmxnrm,ls,ab1,ab2,fg1,fg2  is (nrankmax)         .
c     . The size of bslcmp,hankelre,hankelim   is (nrankmax+1)       .
c     . The size of asc,wt                     is (nthetamax)        .
c     . The common-blocks have been changed and split up,            .
c     . in order to prevent compiler warnings                        .
c     ................................................................

c     ................................................................
c     . EXTENDT1 : 0.0 is changed to 0.q0                            .
c     .            cmplx(0.0,0.0) is changed to cmplx(0.d0,0.d0)      .
c     .            1.0e-05 is changed to 1.0q-05  etc.               .
c     ................................................................
      implicit real*16 (a-h,o-z)
      double complex a,b
      common dtr,rtd,pi
      common /mtxcom/ nrank,nranki,a(400,400),b(400,400)
      common /mtxcom2/ cmxnrm(200)
      common /cmvcom/ nm,kmv
      common/cmvcom2/ cmv,twm,prodm
      common /thtcom/ theta,sinth,costh,x
      common /uvccom/ angint,anginp,utheta,uphi,aovrb,cmr,cmi,ic
      common /mtxcom3/ a2(400,400),b2(400,400)
      real*16 wt(1000),asc(1000),pnmllg(201),bslcmp(201)
      real*16 hankelre(201),hankelim(201),aa2(400,400),bb2(400,400)
c     ..........................................................
c     .  set program constants                                   .
c     .  EXTENDT1 : no imaginary refraction index                .
c     .             nrankmax,nthetamax: new variables            .
c     .             better accuracy of pi,dtr etc.               .
c     ..........................................................
      nrankmax = 200
      nthetamax = 1000
      pi = 3.14159265358979323846264338327795
      dtr =pi/180.q0
      rtd=180.q0/pi
      pi2 = pi/2.q0
      write(6,200)
      write(6,*) 'enter nrank, ntheta, case'
      read(5,*) nrank,ntheta,ic
      nranki = nrank+1
      nr2 = 2*nrank
      write(6,210)
      write(6,*) 'enter x, aovrb, mr'
      read(5,*) x,aovrb,cmr
c     ..........................................................

c     .  set the index of refraction                             .
c     .  for an exp(-iwt) time variation                         .
c     . EXTENDT1 : The refraction index is always real           .
c     .            cmi is erased                                 .
c     .            cm, dcn etc. are declared real*16             .
c     ..........................................................
      cm = cmr
      dcn = cm**2
c     ................................................................
c     .  utheta and uphi are the amplitudes of the incident field in .
c     .    the theta (parallel) and phi (perpendicular) directions   .
c     ................................................................
      utheta = 1.q0/sqrt(2.q0)
      uphi = 1.q0/sqrt(2.q0)
c     ................................................................
c     .  convergence over m (ic = 2) requires nrank azimuthal modes,  .
c     .    a nonsymmetric orientation, and a file for the matrices    .
c     .   EXTENDT1 : status='unknown' is added in the open arguments, .
c     .              to prevent compiler warnings                     .
c     ................................................................
      mtxsav = 0
      if(ic.eq.2) then
        nm = nrank
        angint = pi/4.q0
        anginp = pi/4.q0
        write(6,220)
        write(6,*) 'enter 1 (save [T]) or 2 (save [A])'
        read(5,*) mtxsav
        if(mtxsav.eq.1) then
          open(unit=11,file='t',form='unformatted',status='unknown')
        else
          open(unit=11,file='a',form='unformatted',status='unknown')
        end if
        rewind 11
c     ................................................................
c     .  convergence over ntheta (ic = 0) or nrank (ic = 1) requires  .
c     .    one azimuthal mode (m = 1) and a symmetric orientation     .
c     ................................................................
      else
        nm = 1
        angint = 0.q0
        anginp = 0.q0
      end if
      write(6,230) nm,nrank,ntheta,ic
      write(6,240) x,aovrb,cmr
      angtpr = angint*rtd
      angppr = anginp*rtd
      write(6,250) utheta,uphi,angtpr,angppr
```

```
c     ..........................................
c     .  convergence over ntheta (ic = 0) requires  .
c     .    solutions for two values of ntheta       .
c     ..........................................
      it = 1
      if(ic.eq.0) it = 2
      do 160 icon = 1,it
c     ..................................................
c     .  set the integration points and weighting values  .
c     ..................................................
        call gauss(wt,asc,ntheta,0.q0,pi2)
c     ........................................
c     .  enter a loop for each azimuthal mode m  .
c     ........................................
        do 150 im = 1,nm
c     ............................
c     .  set m-dependent variables  .
c     ............................
          kmv = im-1
          if(nm.eq.1) kmv = 1
          cmv = real(kmv)
          cm2 = cmv**2
          prodm = 1.q0
          if(kmv.eq.0) then
            em = 1.q0
          else
            em = 2.q0
            quanm = cmv
            do 10 i = 1,kmv
              quanm = quanm+1.q0
              prodm = quanm*prodm/2.q0
10          continue
          end if
          qem = 2.q0/em
          twm = 2.q0*cmv
c     ................................................
c     .  set indices for matrix compression when n < m  .
c     .    note: useful only when m > 1                 .
c     ................................................
          ij = kmv-1
          if(ij.lt.0) ij = 0
          ijt = 2*ij
          ns = nrank-ij
          ns2 = 2*ns
c     ........................................
c     .  initialize all matrix elements to zero  .
c     .  EXTENDT1 : Different initialization     .
c     ........................................
            do iii = 1,2*nrankmax
              do jjj = 1,2*nrankmax
                a2(iii,jjj) = 0.q0
                b2(iii,jjj) = 0.q0
              enddo
            enddo
c     ..........................................
c     .  enter a loop to integrate over the surface  .
c     .    (theta is the integration variable)       .
c     ..........................................
            do 50 ithta = 1,ntheta
              theta = asc(ithta)
              sinth = sin(theta)
              costh = cos(theta)
c     ..............................................
c     .  calculate the associated Legendre functions  .
c     .    at each integration point                  .
c     ..............................................
              call genlgp(pnmllg,nranki)
c     ..................................
c     .  calculate kr and it's derivative  .
c     .    at each integration point       .
c     ..................................
c     ..........................................................
c     .  current shape descriptor is for a spheroid:           .
c     .    a < b : oblate, a = 0 : sphere, a > b : prolate     .
c     .    dimension = 2a (along the symmetry axis) x 2b       .
c     .                                                        .
c     .  EXTENDT1 : (comment)                                  .
c     .  All axisymmetric particles can be treated,            .
c     .     (not only prolates and oblates)                    .
c     .  but it will be harder to obtain converged solutions.  .
c     .  The expression of r(theta) is needed.                 .
c     .  qb = r(theta)/a                                       .
c     .  dckr = x*(d(qb)/dtheta)                               .
c     .  x = k*a (input parameter)                             .
c     .  2a is the dimension along the symmetry axis.          .
c     .  2a is NOT the dimension along semimajor axis!!        .
c     ..........................................................
              qb = 1.q0/sqrt(costh**2+(aovrb*sinth)**2)
              ckr = x*qb
              dckr = -x*costh*sinth*(aovrb**2-1.q0)*qb**3
c     ..................................................................
c     .  calculate the Hankel functions (real argument) and            .
c     .    Bessel functions (complex argument) at each                 .
c     .    integration point                                           .
c     . EXTENDT1 : two variables, hankelre and hankelim, are used      .
c     .            instead of hankel and hbkre,hbkim instead of hbk    .
```

```
c     . EXTENDT1 : bslcmp is always real, while refraction index is   .
c     .            real. In T1 bslcmp was declared complex            .
c     . EXTENDT1 : dcn,h,bk,bbk,sa,sb,ckpr are real                   .
c     ..................................................................
            call besh(ckr,hankelre,hankelim,nranki)
            ckpr = cm*ckr
            call besj(ckpr,bslcmp,nranki)
            d = dckr*sinth
            wtsin = 2.q0*wt(ithta)*sinth
c     ..........................................................
c     .  enter a loop for each row of the matrix               .
c     . EXTENDT1: h is the imaginary hankel-element-quota,     .
c     .           while it's only used in the a2-calculation   .
c     .           (the imaginary part of A)                    .
c     ..........................................................
            do 40 irow = 1,nrank
              if(irow.le.ij) go to 40
              irow1 = irow+nrank
              crow = real(irow)
              crowm = cmv+crow
              crow1 = crow+1.q0
              br = hankelre(irow)/hankelre(irow+1)
              h = hankelim(irow)/hankelim(irow+1)
c     ..................................................
c     .  enter a loop for each column of the matrix  .
c     ..................................................
              do 30 icol = 1,nrank
                if(icol.le.ij) go to 30
                icol1 = icol+nrank
                ccol = real(icol)
                ccolm = cmv+ccol
                ccol1 = ccol+1.q0
c     ....................................
c     .  calculate variable combinations  .
c     ....................................
                crij = crow+ccol
                crssij = crow*ccol
                cmcrco = cm2+qem*crssij*costh**2
                pnr0c0 = pnmllg(irow)*pnmllg(icol)
                pnr0c1 = pnmllg(irow)*pnmllg(icol+1)
                pnr1c0 = pnmllg(irow+1)*pnmllg(icol)
                pnr1c1 = pnmllg(irow+1)*pnmllg(icol+1)
                b1a = crow*costh*pnr1c1-crowm*pnr0c1
                b1b = ccol*costh*pnr1c1-ccolm*pnr1c0
                bk = cm*bslcmp(icol)/bslcmp(icol+1)
                bbk = hankelre(irow+1)*bslcmp(icol+1)*wtsin
                hbkre = hankelre(irow+1)*bslcmp(icol+1)*wtsin
                hbkim = hankelim(irow+1)*bslcmp(icol+1)*wtsin
```

```
c     ..........................................................
c     .  calculate the [A]' and  [B]' submatrices, e.g.,      .
c     .            _                                _          .
c     .           |                |                |          .
c     .           |              ,                 , |          .
c     .           | (K + cm * J)   |-(I + cm * L)   |          .
c     .     ,     |                                 |          .
c     .  [A]  =   |- - - - - - - -|- - - - - - - -|          .
c     .           |              ,                 , |          .
c     .           | (L + cm * I)   | (J + cm * K)   |          .
c     .           |                                 |          .
c     .           |_               |                _|          .
c     .                                                        .
c     .  EXTENDT1 : a2 and b2 are used instead and             .
c     .                     A = cmplx(b2,a2)                   .
c     .                     B = cmplx(b2,0.0)                  .
c     .           Thus, modifications have been made           .
c     .           in the a2-expressions(hbkim instead of hbk)  .
c     ..........................................................
c     ..................................................................
c     .  simplification for mirror-symmetric particles:              .
c     .  if (irow+icol) is even then the I and L elements are zero  .
c     .  if (irow+icol) is odd  then the J and K elements are zero  .
c     ..................................................................
      if(mod((irow+icol),2).ne.0) then
c     ..................................................
c     .  increment the -(I + cm * L)' submatrix element  .
c     ..................................................
        if(kmv.ne.0) then
          b1 = b1a+b1b
          sa = pnr1c1*
     1         (crow*crow1*bk+ccol*ccol1*h-crssij*(crij+2.q0)/ckr)*d
          sa = (ckr*(1.q0+h*bk)-ccol*h-crow*bk+crssij/ckr)*b1*ckr+sa
          a2(icol-ij,irow1-ijt)=a2(icol-ij,irow1-ijt)+cmv*sa*hbkim
          sb = sa+(br-h)*(ccol*ccol1*pnr1c1*d+(bk*ckr-ccol)*b1*ckr)
          b2(icol-ij,irow1-ijt) = b2(icol-ij,irow1-ijt)+cmv*sb*bbk
c     ..................................................
c     .  increment the (L + cm * I)' submatrix element  .
c     ..................................................
          c = (dcn-1.q0)*b1*ckr**2
          a2(icol1-ijt,irow-ij)=a2(icol1-ijt,irow-ij)-cmv*(sa+c)
     1    *hbkim/cm
          b2(icol1-ijt,irow-ij) = b2(icol1-ijt,irow-ij)-
     1                              cmv*(sb+c)*bbk/cm
        end if
      else
c     ..................................................
c     .  increment the (J + cm * K)' submatrix element  .
```

```
c     ...............................................
        a12 = cmcrco*pnr1c1-qem*(crow*ccolm*costh*pnr1c0+
     1  ccol*crowm*costh*pnr0c1-crowm*ccolm*pnr0c0)
        a22 = a12*ckr**2
        b1a = ccol*ccol1*b1a
        b1b = crow*crow1*b1b
        sa = (ckr*(bk-dcn*h)+dcn*crow-ccol)*a12*ckr+(b1a-dcn*b1b)*qem*d
        a2(icol1-ijt,irow1-ijt)=a2(icol1-ijt,irow1-ijt)+sa*hbkim/cm
        b2(icol1-ijt,irow1-ijt) = b2(icol1-ijt,irow1-ijt)
     1                          +(sa-(br-h)*dcn*a22)*bbk/cm
c     ...............................................
c     .  increment the (K + cm * J)' submatrix element  .
c     ...............................................
        sa = (ckr*(bk-h)+crow-ccol)*a12*ckr+(b1a-b1b)*qem*d
        a2(icol-ij,irow-ij) = a2(icol-ij,irow-ij)+sa*hbkim
        b2(icol-ij,irow-ij)=b2(icol-ij,irow-ij)+(sa-(br-h)*a22)*bbk
      end if
30          continue
40        continue
50      continue
c     ....................................
c     .  calculate the normalization factor  .
c     ....................................
        do 70 irow = 1,nrank
          if(irow.le.ij) go to 70
          crow = real(irow)
          if(kmv.eq.0) then
            cmxnrm(irow-ij)=(2.q0*crow+1.q0)/(4.q0*crow*(crow+1.q0))
          else
            if(irow.lt.kmv) then
              cmxnrm(irow-ij) = 1.q0
            else
              ib = irow-kmv+1
              ie = irow+kmv
              fctki = 1.q0
              fprod = real(ib)
              do 60 l = ib,ie
                fctki = fctki*fprod
                fprod = fprod+1.q0
60            continue
                cmxnrm(irow-ij) =  em*(2.q0*crow+1.q0)/
     1                             (4.q0*crow*(crow+1.q0)*fctki)
            end if
          end if
70      continue
c     ...............................................
c     .  convergence over nrank (ic = 1) requires that  .
c     .    the [A]'and [B]' matrices be stored for      .
c     .    a second iteration with nrank = nrank - 1    .
c     . EXTENDT1 : a2(*,*) and b2(*,*) are stored       .
c     .            instead of [A]' and [B]'             .
c     ...............................................

          if(ic.eq.1) then
            do 80 ii = 1,nr2
              do 75 jj = 1,nr2
                aa2(ii,jj) = a2(ii,jj)
                bb2(ii,jj) = b2(ii,jj)
75            continue
80          continue
          end if
c     .....................................................
c     .  for convergence over nm (ic = 2) save the        .
c     .    [A] matrix and normalization constants         .
c     .    for later use in internal field programs       .
c     .   EXTENDT1 : [A]' is stored in the a2(*,*)- and   .
c     .              b2(*,*)-arrays.                      .
c     .                                                   .
c     .              transpose and store in the a(*,*) array  .
c     .....................................................
          if(ic.eq.2.and.mtxsav.eq.2) then
            do 90 ir = 1,ns2
              do 85 jr = 1,ns2
                a(ir,jr) = cmplx(b2(jr,ir),a2(jr,ir))
85            continue
90          continue
            write(11) aa
          end if
c     ....................................
c     .                        -1         .
c     .  calculate [T]' = [A]' * [B]'     .
c     ....................................
          call prcssm(ns2)
c     ....................................
c     .  [T]' is stored in the b(*,*) array  .
c     .    transpose [T]' to obtain [T]      .
c     ....................................
          do 100 ir = 1,ns2
            do 95 jr = 1,ns2
              a(ir,jr) = b(jr,ir)
95          continue
100       continue
c     ...............................................
c     .  for convergence over nm (ic = 2) save the     .
c     .    [T] matrix and normalization constants      .
c     .    for later use in scattering programs        .
```

```
c     .  the [T] matrix is stored in the a(*,*) array  .
c     ...................................................
          if(ic.eq.2.and.mtxsav.eq.1) write(11) a,cmxnrm
c     ...................................................
c     .  calculate the angular scattering and efficiencies  .
c     ...................................................
          call addprc
c     ...................................................
c     .  convergence over nrank (ic = 1) requires a second  .
c     .    solution with nrank = nrank-1                    .
c     ...................................................
          if(ic.eq.1) then
c     ...............................................
c     .  restore the original [A]' and [B]' matrices  .
c     .  EXTENDT1 : a2(*,*) and b2(*,*) are restored  .
c     ...............................................
            do 110 ii = 1,nr2
              do 105  jj = 1,nr2
                a2(ii,jj) = aa2(ii,jj)
                b2(ii,jj) = bb2(ii,jj)
105           continue
110         continue
c     .........................................
c     .  delete the last row and column of each    .
c     .    submatrix to reduce nrank to nrank - 1  .
c     .  EXTENDT1 : the a2- and b2- submatrices    .
c     .             are used instead of A and B    .
c     .........................................
            nr2m = nr2-1
            nr2 = 2*(nrank-1)
            do 120 ir = 1,nr2m
              do 115 jr = nrank,nr2
                a2(ir,jr) = a2(ir,jr+1)
                b2(ir,jr) = b2(ir,jr+1)
115           continue
120         continue
            do 130 jr = 1,nr2
              do 125 ir = nrank,nr2
                a2(ir,jr) = a2(ir+1,jr)
                b2(ir,jr) = b2(ir+1,jr)
125           continue
130         continue
            nranki = nrank
            nrank = nrank-1
            write(6,230) nm,nrank,ntheta,ic
c     ...................................
c     .                    -1             .
c     .  calculate [T]' = [A]' * [B]'     .
c     ...................................
          call prcssm(nr2)
c     ...................................
c     .  [T]' is stored in the b(*,*) array  .
c     .    transpose [T]' to obtain [T]      .
c     ...................................
            do 140 ir = 1,nr2
              do 135 jr = 1,nr2
                a(ir,jr) = b(jr,ir)
135           continue
140         continue
c     ...................................................
c     .  calculate the angular scattering and efficiencies  .
c     ...................................................
            call addprc
          end if
150     continue
c     .........................................
c     .  convergence over ntheta (ic = 0) requires  .
c     .    a solution for a second value of ntheta  .
c     .........................................
          if(ic.eq.0.and.icon.eq.1) then
            ntheta = ntheta+4
            write(6,230) nm,nrank,ntheta,ic
          end if
160     continue
        stop
200     format('.....................................',/,
     1       '.  convergence test - save the matrices  .',/,
     2       '.    [T] is written to file ''t''          .',/,
     3       '.    [A] is written to file ''a''          .',/,
     4       '.    parameters are in file ''case''       .',/,
     5       '.....................................',//,
     6       'nrank: number of terms',/,
     7       'ntheta: number of integration steps',/,
     8       'convergence case: ntheta (0), nrank (1), nm (2)',/)
210     format(/,'x: size parameter',/,
     1       'aovrb: a/b ratio',/,
     2       'index of refraction:',/)
220     format(/,'save [T] or [A] matrix',/)
230     format(/,'nm',i8,3x,'nrank',i6,3x,'ntheta',i8,3x,'ic',i12)
240     format('ka',f8.3,3x,'aovrb',f6.3,3x,'mr',f12.3,3x)
250     format('utht',f6.3,3x,'uphi',f7.3,3x,'angint',f8.3,3x,
     1'anginp',f8.3)
        end
        subroutine gauss(wt,asc,n,aa,bb)
c     ...............................................................
c     .  calculate abscissas and weights for n-point Gaussian       .
```

```
c     .    quadrature over integration limits of aa to bb             .
c     .  aa  = lower integration limit                                 .
c     .  bb  = upper integration limit                                 .
c     .  n   = number of integration points                            .
c     .  asc = abscissas                                               .
c     .  wt  = weights                                                 .
c     .                                                                .
c     .  a well-documented program which performs the same function    .
c     .    is given in Chapter 4.5 of W.H. Press, B.P. Flannery,       .
c     .    S.A. Teukolsky and W.T. Vetterling, Numerical Recipes -     .
c     .    The Art of Scientific Computing (Cambridge University       .
c     .    Press, Cambridge, 1986)                                     .
c     .                                                                .
c     .  the results (either program) compare with Table 25.4 in M.    .
c     .    Abramowitz and I.A. Stegun, Handbook of Mathematical Func-  .
c     .    tions (National Bureau of Standards, Washington, 1964)      .
c     ..................................................................
      implicit real*16 (a-h,o-z)
      real*16 wt(n),asc(n)
      data pi/3.14159265358979323846264338327 95/
      data const,tol/.148678816357,1.0q-31/
c     ..........................................................
c     .  the tolerance (tol) may be decreased to 1.0d-10 or     .
c     .    less when double precision variables are used        .
c     .  EXTENDT1: the tolerance (tol) is decreased to 1.0q-31  .
c     ..........................................................
      data c1,c2,c3,c4/.125,-.0807291666,.2460286458,-1.824438767/
      if(n.eq.1) then
        asc(1) = 1.q0/sqrt(3.q0)
        wt(1) = 1.q0
        return
      end if
      cn = real(n)
      ndiv2 = n/2
      np1 = n+1
      cnn1 = cn*(cn+1.q0)
      appfct = 1.q0/sqrt((cn+.5)**2+const)
      con1 = .5*(bb-aa)
      con2 = .5*(bb+aa)
      do 30 k = 1,ndiv2
        b = (real(k)-.25)*pi
        bisq = 1.q0/b**2
        bfroot = b*(1.q0+bisq*(c1+bisq*(c2+bisq*(c3+c4*bisq))))
        xi = cos(appfct*bfroot)
10      x = xi
        pm2 = 1.q0
        pm1 = x
        do 20 in = 2,n
          rn = real(in)
          p = ((2.q0*rn-1.q0)*x*pm1-(rn-1.q0)*pm2)/rn
          pm2 = pm1
          pm1 = p
20      continue
        pm1 = pm2
        aux = 1.q0/(1.q0-x**2)
        der1p = cn*(pm1-x*p)*aux
        der2p = (2.q0*x*der1p-cnn1*p)*aux
        ratio = p/der1p
        xi = x-ratio*(1.q0+ratio*der2p/(2.q0*der1p))
      if(abs(xi-x).gt.tol) go to 10
      asc(k) = -xi
      wt(k) = 2.q0*(1.q0-xi**2)/(cn*pm1)**2
      asc(np1-k) = -asc(k)
      wt(np1-k) = wt(k)
30    continue
      if(mod(n,2).ne.0) then
        asc(ndiv2+1) = 0.q0
        nm1 = n-1
        nm2 = n-2
        prod = cn
        do 40 k = 1,nm2,2
          prod = prod*real(nm1-k)/real(n-k)
40      continue
        wt(ndiv2+1) = 2.q0/prod**2
      end if
      do 50 k = 1,n
        asc(k) = con1*asc(k)+con2
        wt(k) = con1*wt(k)
50    continue
      return
      end
      subroutine genlgp(pnmllg,nc)
c     ..........................................................
c     .  calculate associated Legendre functions (argument     .
c     .    cos(theta)) divided by sin(theta) for each          .
c     .    azimuthal mode m                                    .
c     .  generate first two orders by formula and remaining    .
c     .    orders by recursion                                 .
c     .                                                        .
c     .  pnmllg = associated Legendre function/sin(theta)      .
c     .  sinth = sin(theta), costh = cos(theta)                .
c     .  nc = number of orders (0 to nc-1)                     .
c     .  the order of the associated Legendre functions is     .
c     .    incremented by one in the pnmllg(*) array           .
c     ..........................................................
      implicit real*16 (a-h,o-z)
```

```
      common /cmvcom/ nm,kmv
      common /cmvcom2/ cmv,twm,prodm
      common /thtcom/ theta,sinth,costh,x
      real*16 pnmllg(nc)
      dtwm = twm+1.q0
c     ..............................................................
c     .  calculate for the special case theta = 0 degrees or 180   .
c     .    degrees - note that for theta = 180 degrees, costh = -1,.
c     .    but theta is set to 0 degrees prior to subroutine entry .
c     ..............................................................
c     ..........................................................
c     .  if theta = 0 degrees and m.ne.1 all functions are zero  .
c     ..........................................................
      if(theta.eq.0.and.kmv.ne.1) then
        do 10 ilg = 1,nc
          pnmllg(ilg) = 0.q0
10      continue
        return
      end if
c     ..........................................................
c     .  if theta = 0 degrees and m.eq.1 calculate orders 0,1,2  .
c     ..........................................................
      if(theta.eq.0.and.kmv.eq.1) then
        pnmllg(1) = 0.q0
        pnmllg(2) = 1.q0
        pla = 1.q0
        plb = 3.q0*costh
        pnmllg(3) = plb
        ibeg = 4
c     ..............................................................
c     .  if theta.ne.0 degrees and m.eq.0 calculate orders 0 and 1  .
c     ..............................................................
      else if(kmv.eq.0) then
        pla = 1.q0/sinth
        plb = costh*pla
        pnmllg(1) = pla
        pnmllg(2) = plb
        ibeg = 3
c     ..........................................................
c     .  if theta.ne.0 degrees and m.ne.0 calculate first two  .
c     .    nonzero orders - the associated Legendre function   .
c     .    is zero for orders less than the azimuthal mode m   .
c     ..........................................................
      else
        do 20 ilg = 1,kmv
          pnmllg(ilg) = 0.q0
20      continue
        pla = prodm*sinth**(kmv-1)
        pnmllg(kmv+1) = pla
        plb = dtwm*costh*pla
        pnmllg(kmv+2) = plb
        ibeg = kmv+3
      end if
c     ..............................................
c     .  recur upward to obtain all remaining orders  .
c     ..............................................
      if(ibeg.gt.nc) return
      cnmul = real(2*ibeg-3)
      cnm = 2.q0
      cnmm = dtwm
      do 30 ilgr = ibeg,nc
        plc = (cnmul*costh*plb-cnmm*pla)/cnm
        pnmllg(ilgr) = plc
        pla = plb
        plb = plc
        cnmul = cnmul+2.q0
        cnm = cnm+1.q0
        cnmm = cnmm+1.q0
30    continue
      return
      end
      subroutine besh(x,hankelre,hankelim,nc)
c     ..................................................
c     .  calculate Hankel functions                    .
c     .  bj = Bessel function of the first kind        .
c     .  by = Bessel function of the second kind       .
c     .  x = real argument                             .
c     .  nc = number of orders (0 to nc-1)             .
c     .  the order of the functions is incremented by  .
c     .  one in the bj(*),by(*) and hankelre/im(*) arrays .
c     .                                                .
c     .  arrays are set for nc = 201 (nrank+1) maximum .
c     .  EXTENDT1 : hankelre and hankelim are used instead .
c     .             of hankel in T1                    .
c     ..................................................
      implicit real*16 (a-h,o-z)
      real*16 hankelre(nc),hankelim(nc)
      real*16 bj(201),by(201),t(3)
c     ..........................................
c     .  by(*) calculation - obtain the zeroeth and  .
c     .                      first order functions   .
c     ..........................................
      a = sin(x)/x
      by(1) = -cos(x)/x
      by(2) = by(1)/x-a
c     ..............................................................
```

```
c     .  obtain the higher order functions by upward recursion  .
c     ...........................................................
      do 10 n = 3,nc
        rn = real(n-2)
        by(n) = (2.q0*rn+1.q0)*by(n-1)/x-by(n-2)
10    continue
c     ................................................
c     .  bj(*) calculation - set the starting order  .
c     .                      for downward recursion  .
c     ................................................
      nst = nc+int((101.q0+x)**.5)
c     ................................................
c     .  the t(*) array is used to recur down to the     .
c     .    two highest order functions that are needed   .
c     .  set starting values for the two highest orders  .
c     .    nst and nst-1                                 .
c     ................................................
      t(3) = 0.q0
      t(2) = 1.0q-35
c     ................................................
c     .  recur downward to obtain orders nc-1 and nc-2  .
c     ................................................
      do 20 i = nst-1,nc-1,-1
        ri = real(i)
        t(1) = (2.q0*ri+1.q0)*t(2)/x-t(3)
        t(3) = t(2)
        t(2) = t(1)
20    continue
c     ..........................................
c     .  continue downward recursion to order zero  .
c     ..........................................
      bj(nc) = t(3)
      bj(nc-1) = t(2)
      do 30 i = nc-2,1,-1
        ri = real(i)
        bj(i) = (2.q0*ri+1.q0)*bj(i+1)/x-bj(i+2)
30    continue
c     ................................................
c     .  calculate the scale factor and the functions  .
c     ................................................
      alpha = a/bj(1)
      do 40 k = 1,nc
        hankelre(k) = bj(k)*alpha
        hankelim(k) = by(k)
40    continue
      return
      end
      subroutine besj(z,bslcmp,nc)
```

```
c     ................................................
c     .  calculate Bessel functions of the first kind      .
c     .  bslcmp = Bessel function of the first kind        .
c     .                                                    .
c     .  nc = number of orders (0 to nc-1)                 .
c     .  the order of the Bessel functions is incremented  .
c     .    by one in the bslcmp(*) array                   .
c     .  EXTENDT1 : bslcmp is declared real (no absorption).
c     .             z is also real (no absorption), x = z  .
c     ................................................
      implicit real*16 (a-h,o-z)
      real*16 bslcmp(nc),t(3)
c     ..........................................................
c     .  set the starting order for downward recursion          .
c     . EXTENDT1 : nc has been added to nst to prevent          .
c     .            that the loop(10) will not be entered        .
c     ..........................................................
      x = abs(z)
      nst = nc+int(x+4.05*x**.3333+2.q0+(101.q0+x)**.5)
c     ................................................
c     .  the t(*) array is used to recur down to the     .
c     .    two highest order functions that are needed   .
c     .  set starting values for the two highest orders  .
c     .    nst and nst-1                                 .
c     ................................................
      t(3) = 0.q0
      t(2) = 1.0q-35
c     ................................................
c     .  recur downward to obtain orders nc-1 and nc-2  .
c     ................................................
      do 10 i = nst-1,nc-1,-1
        ri = real(i)
        t(1) = (2.q0*ri+1.q0)*t(2)/z-t(3)
        t(3) = t(2)
        t(2) = t(1)
10    continue
c     ..........................................
c     .  continue downward recursion to order zero  .
c     ..........................................
      bslcmp(nc) = t(3)
      bslcmp(nc-1) = t(2)
      do 20 i = nc-2,1,-1
        ri = real(i)
        bslcmp(i) = (2.q0*ri+1.q0)*bslcmp(i+1)/z-bslcmp(i+2)
20    continue
c     ................................................
c     .  calculate the scale factor and the functions  .
c     ................................................
```

```
      alpha = sin(z)/(z*bslcmp(1))
      do 30 k = 1,nc
        bslcmp(k) = bslcmp(k)*alpha
30    continue
      return
      end
      subroutine prcssm(n)
c     ..............................................................
c     .                  -1                                        .
c     .  calculate [T]' = [A]' * [B]' using Gauss-Jordan reduction .
c     ..............................................................
      implicit real*16 (a-h,o-z)
      double complex a,b
      common /mtxcom/ nr,nri,a(400,400),b(400,400)
      common /mtxcom2/ cmxnrm(200)
      common/mtxcom3/ a2(400,400),b2(400,400)
      real*16 ls(400),are(400,400),aim(400,400),bre(400,400)
      real*16 bim(400,400),areold(400,400),breold(400,400)
c     ..............................................................
c     .  EXTENDT1: are,aim,bre,bim are used (are = real(a) etc.) .
c     ..............................................................
      do ii = 1,n
        do jj = 1,n
           are(ii,jj) = b2(ii,jj)
           aim(ii,jj) = a2(ii,jj)
           bre(ii,jj) = b2(ii,jj)
           bim(ii,jj) = 0.q0
         enddo
      enddo
c     ..........................
c     .  start reduction of [a]  .
c     ..........................
      do 60 i = 1,n
c     ..........................................................
c     .  search for the maximum element in the ith row of [a]  .
c     .  EXTENDT1 : areijmax, aimijmax (real*16) are used      .
c     .             instead of aijmax (complex) in T1          .
c     ..........................................................
        areijmax = are(i,1)
        aimijmax = aim(i,1)
        jmax = 1
        do 10 j = 2,n
          if((are(i,j)**2+aim(i,j)**2).gt.(areijmax**2+aimijmax**2))
     1    then
            areijmax = are(i,j)
            aimijmax = aim(i,j)
            jmax = j
          end if
10      continue
          con = areijmax**2+aimijmax**2
c     ..........................................................
c     .  normalize the ith row of [a] and [b] by aijmax        .
c     .                                                        .
c     . EXTENDT1: con,areold,breold are used in order to simplify .
c     . ( T1: a(i,j) = a(i,j)/aijmax ; b(i,j) = b(i,j)/aijmax) )   .
c     ..........................................................
        do 20 j = 1,n
            areold(i,j)=are(i,j)
            breold(i,j)=bre(i,j)
            are(i,j)=(are(i,j)*areijmax+aim(i,j)*aimijmax)/con
            aim(i,j)=(aim(i,j)*areijmax-areold(i,j)*aimijmax)/con
            bre(i,j)=(bre(i,j)*areijmax+bim(i,j)*aimijmax)/con
            bim(i,j)=(bim(i,j)*areijmax-breold(i,j)*aimijmax)/con
20      continue
c     ..........................................................
c     .  use row transformations to obtain zeros above and     .
c     .    below the jmax element of the ith row of [a] -      .
c     .    apply the same row transformations to [b]           .
c     ..........................................................
        do 50 k = 1,n
          if(k.ne.i) then
            aratre = -are(k,jmax)
            aratim = -aim(k,jmax)
            do 30 j = 1,n
              areold(i,j) = are(i,j)
              if(abs(are(i,j)**2+aim(i,j)**2).gt.0.q0) then
                are(k,j) = aratre*are(i,j)-aratim*aim(i,j)+are(k,j)
                aim(k,j) = aratim*areold(i,j)+aratre*aim(i,j)+aim(k,j)
              end if
30          continue
            are(k,jmax) = 0.q0
            aim(k,jmax) = 0.q0


            do 40 j = 1,n
              breold(i,j)=bre(i,j)
              if(abs(bre(i,j)**2+bim(i,j)**2).gt.0.q0) then
                bre(k,j) = aratre*bre(i,j)-aratim*bim(i,j)+bre(k,j)
                bim(k,j) = aratim*breold(i,j)+aratre*bim(i,j)+bim(k,j)
              end if
40          continue
          end if
50      continue
c     ..........................................................
c     .  store row counter (i) in array ls(*), such that       .
c     .    ls(*) contains the location of the pivot (unity)    .
```

```
c     .    element of each column (after reduction)          .
c     ....................................................
        ls(jmax) = i
60    continue
c     ....................................................
c     .  the reduction of [a] is complete - perform       .
c     .    row interchanges as indicated in array ls(*)   .
c     ....................................................
      do 90 i = 1,n
        k = i
c     ....................................................
c     .  put the integer value in ls(k) into k            .
c     .                                                   .
c     .  if k is less than i, then that row has already been  .
c     .    involved in an interchange so iterate k = ls(k)    .
c     .    until a value of k greater than i (corresponding   .
c     .    to a row stored above the ith row) is obtained     .
c     ....................................................
70      k = ls(k)
        if(k.lt.i) go to 70
        if(k.gt.0) then
          do 80 j=1,n
            aratre = bre(i,j)
            aratim = bim(i,j)
            bre(i,j) = bre(k,j)
            bim(i,j) = bim(k,j)
            bre(k,j) = aratre
            bim(k,j) = aratim
80        continue
        end if
90    continue
c     ....................................................
c     .  EXTENDT1 : the critical region is passed         .
c     .             and the double complex                .
c     .             T-matrix can be stored in b(*,*)      .
.
c     ....................................................
        do i = 1,n
          do j = 1,n
            b(i,j) = cmplx(bre(i,j),bim(i,j))
          enddo
        enddo
c     ....................................
c     .  [T]' is stored in the b(*,*) array  .
c     ....................................
      return
      end
      subroutine genang(n,tang,pang,cosb,sinb)
```

```
c     ....................................................
c     .  calculate scattering angles in the particle frame corres-  .
c     .    ponding to scattering angles in the laboratory frame     .
c     .  calculate the elements of the transform matrix for the     .
c     .    polarization vectors                                      .
c     ....................................................
      implicit real*16 (a-h,o-z)
      common dtr,rtd,pi
      common /uvccom/ angint,anginp,utheta,uphi,aovrb,cmr,cmi,ic
      real*16 tang(n),pang(n),cosb(n),sinb(n)
c     ....................................................
c     .  dltang calculation assumes scattering over 180.0 degrees  .
c     ....................................................
      dltang = pi/real(n-1)
      sint = sin(angint)
      cost = cos(angint)
      sinp = sin(anginp)
      cosp = cos(anginp)
c     ....................................................
c     .  calculate scattering angles in the particle frame corres-  .
c     .    ponding to scattering angles in the laboratory frame     .
c     ....................................................
      do 10 i = 1,n
        ts = dltang*real(i-1)
        ps = 0.q0
        sints = sin(ts)
        costs = cos(ts)
        cosps = cos(ps)
        x = -cosp*cost*sints*cosps+sint*costs
        y = sinp*sints*cosps
        z = cosp*sint*sints*cosps+cost*costs
c     ....................................................
c     .  calculate the theta scattering angles in the particle frame  .
c     ....................................................
        r = sqrt(x**2+y**2)
        tang(i) = 0.q0
        if(z.ne.0.q0.or.r.ne.0.q0) tang(i) = atan2(r,z)
c     ....................................................
c     .  calculate the phi scattering angles in the particle frame  .
c     ....................................................
        pang(i) = 0.q0
        if(x.ne.0.q0.or.y.ne.0.q0) pang(i) = atan2(y,x)
        if(pang(i).lt.0.q0) pang(i) = 2.q0*pi+pang(i)
c     ....................................................
c     .  calculate the elements of the transform matrix for the      .
c     .    polarization vectors                                      .
c     .                                                              .
c     .  sint  = sine   (particle theta orientation angle)           .
```

```
c     .  cost  = cosine (particle theta orientation angle)          .
c     .  sinp  = sine   (particle phi orientation angle)            .
c     .  cosp  = cosine (particle phi orientation angle)            .
c     .                                                             .
c     .  sints = sine   (theta scattering angle in laboratory frame)  .
c     .  costs = cosine (theta scattering angle in laboratory frame)  .
c     .  cosps = cosine (phi scattering angle in laboratory frame)    .
c     .                                                             .
c     .  pang(i) = phi scattering angle in particle frame           .
c     ..............................................................
        cosb(i) = cosps*sinp*cost*sin(pang(i))
     1           -cosps*cosp*cos(pang(i))
        sinb(i) = costs*cosps*cosp*cost*sin(pang(i))
     1          +sints*sint*sin(pang(i))+costs*cosps*sinp*cos(pang(i))
10    continue
      return
      end
      subroutine addprc
c     ..................................................
c     .  convergence test the differential scattering   .
c     .    cross section and calculate the scattering,  .
c     .    absorption and extinction efficiencies       .
c     .  calculations are performed in the particle     .
c     .    frame and written in the laboratory frame    .
c     ..................................................
      implicit real*16 (a-h,o-z)
      double complex tmat,b,ci,c1,c2,cim,ab1(400),ab2(400)
      double complex acans(10,2),fg1(400),fg2(400)
      common dtr,rtd,pi
      common /mtxcom/ nrank,nranki,tmat(400,400),b(400,400)
      common /mtxcom2/ cmxnrm(200)
      common /cmvcom/ nm,kmv
      common /cmvcom2/ cmv,twm,prodm
      common /thtcom/ theta,sinth,costh,x
      common /uvccom/ angint,anginp,utheta,uphi,aovrb,cmr,cmi,ic
      real*16 tang(10),pang(10),cosb(10),sinb(10),pnmllg(201)
      real*16 oldh(10),oldv(10),clrtot(40)
      equivalence (acans,clrtot)
      logical first
      data first/.true./
c     ..............................................................
c     .  set constants and calculate the coordinate transformation  .
c     .    variables the first time the subroutine is entered       .
c     ..............................................................
      if(first) then
        ci = (0.d0,1.d0)
        snorm = 1.q0/(pi*x**2)
        call genang(10,tang,pang,cosb,sinb)
c     ..................................................
c     .  transform the incident field polarization vector   .
c     .    from the laboratory frame to the particle frame  .
c     .    uu1 = theta component and uu2 = phi component    .
c     ..................................................
        uu1 = cosb(1)*utheta-sinb(1)*uphi
        uu2 = sinb(1)*utheta+cosb(1)*uphi
      end if
c     ..............................................................
c     .  clear the result accumulators when ic = 0 or ic = 1 and  .
c     .    the first time the subroutine is entered for ic = 2    .
c     ..............................................................
      if(ic.le.1.or.kmv.eq.0) then
        s = 0.q0
        do 10 i = 1,40
          clrtot(i) = 0.q0
10      continue
        nr2 = 2*nrank
      end if
c     ..................................................
c     .  set indices for matrix compression when n < m  .
c     .    note: useful only when m > 1                 .
c     ..................................................
      ij = kmv-1
      if(ij.lt.0) ij = 0
      ijt = 2*ij
      ns = nrank-ij
      ns2 = 2*ns
c     ..........................................................
c     .  calculate the associated Legendre functions for the  .
c     .    incident angle = tang(1) in the particle frame     .
c     ..........................................................
      theta = tang(1)
      costh = cos(theta)
      sinth = sin(theta)
      if(tang(1).eq.pi) then
        theta = 0.q0
        costh = -1.q0
      end if
      call genlgp(pnmllg,nranki)
c     ..................................................
c     .  calculate the incident field coefficients:    .
c     .    ab1(*) = theta polarization                 .
c     .    ab2(*) = phi polarization                   .
c     .  note: ab1(*) and ab2(*) used in scattered     .
c     .        field programs are i times ab1(*) and   .
c     .        ab2(*) used in internal field programs  .
c     ..................................................
```

```
      do 20 n = 1,nrank
        if(n.le.ij) go to 20
        np = n+nrank
        cn = real(n)
        n1 = n+1
        c1 = 4.q0*ci**n
        c2 = 4.q0*ci**n1
        p1 = cn*costh*pnmllg(n1)-(cn+cmv)*pnmllg(n)
        p2 = cmv*pnmllg(n1)
        ab1(n-ij) = c1*p2*uu1
        ab1(np-ijt) = -c2*p1*uu1
        ab2(n-ij) = -c1*p1*uu2
        ab2(np-ijt) = c2*p2*uu2
20    continue
c     ..........................................
c     .  the scattered field coefficients = -[T]  .
c     .    times the incident field coefficients  .
c     .  fg2(ns+1) to fg2(ns2) is -g(omn)         .
c     ..........................................
      do 40 i = 1,ns2
        fg1(i) = 0.q0
        fg2(i) = 0.q0
        do 30 j = 1,ns2
          fg1(i) = fg1(i)-tmat(i,j)*ab1(j)
          fg2(i) = fg2(i)-tmat(i,j)*ab2(j)
30      continue
40    continue
c     ..............................................................
c     .  evaluate the scattered field at each scattering angle      .
c     .    theta = tang(*) and phi = pang(*) in the particle frame  .
c     .  results are accumulated in acans(*,*) for each            .
c     .    azimuthal mode m                                        .
c     .  acans = kF, where F is a component of the vector          .
c     .    far-field amplitude                                     .
c     ..............................................................
      do 60 iu = 1,10
c     ..............................................
c     .  calculate the associated Legendre functions  .
c     .    at each scattering angle                   .
c     ..............................................
        theta = tang(iu)
        sinth = sin(theta)
        costh = cos(theta)
        if(tang(iu).eq.pi) then
          theta = 0.q0
          costh = -1.q0
        end if
        call genlgp(pnmllg,nranki)
        phi = cmv*pang(iu)
        cosphi = cos(phi)
        sinphi = sin(phi)
        do 50 n = 1,nrank
          if(n.le.ij) go to 50
          cn = real(n)
          np = n+nrank
          n1 = n+1
          p1 = cn*costh*pnmllg(n1)-(cn+cmv)*pnmllg(n)
          p2 = cmv*pnmllg(n1)
          aa = sinphi*p1
          bb = cosphi*p1
          cc = sinphi*p2
          dd = cosphi*p2
          cim = ci**(-n1)
          nij = n-ij
          npijt = np-ijt
c     ..........................................
c     .  solve for the theta polarized scattered  .
c     .    field in the particle frame            .
c     ..........................................
          acans(iu,1) = acans(iu,1)+cim*(fg1(nij)*dd+ci*fg1(npijt)*bb
     1                  -fg2(nij)*cc-ci*fg2(npijt)*aa)*cmxnrm(nij)
c     ..........................................
c     .  solve for the phi polarized scattered  .
c     .    field in the particle frame          .
c     .  fg2(npijt) is -g(omn) per (3.12b)      .
c     ..........................................
          acans(iu,2) = acans(iu,2)-cim*(fg1(nij)*aa+ci*fg1(npijt)*cc
     1                  +fg2(nij)*bb+ci*fg2(npijt)*dd)*cmxnrm(nij)
50      continue
60    continue
c     ..........................................................
c     .  write the scattering results in the laboratory frame  .
c     .    h = horizontal (parallel) or theta polarization     .
c     .    v = vertical (perpendicular) or phi polarization    .
c     ..........................................................
      write(6,100) kmv
      nconv = 0
      mconv = 0
      do 70 jup = 1,10
        ang = 20.*real(jup-1)
        h = snorm*abs(cosb(jup)*acans(jup,1)+sinb(jup)*acans(jup,2))**2
        v = snorm*abs(cosb(jup)*acans(jup,2)-sinb(jup)*acans(jup,1))**2
        write(6,110) ang,h,v
c     ..................................................
c     .  test for convergence at each scattering angle  .
c     ..................................................
```

```
        if(.not.first) then
          if(abs(1.q0-oldh(jup)/h).le.1.0q-03) nconv = nconv+1
          if(abs(1.q0-oldv(jup)/v).le.1.0q-03) mconv = mconv+1
        end if
        oldh(jup) = h
        oldv(jup) = v
70    continue
c     ..........................................................
c     .  calculate the normalized (pi*a**2) extinction (ex),   .
c     .    scattering (sc) and absorption (ab) cross sections  .
c     ..........................................................
c     ..............................................................
c     .  the ex calculation can be performed in either             .
c     .    reference frame - use the particle frame                .
c     . EXTENDT1 : imag instead of aimag while double complex is used .
c     ..............................................................
      ex = (imag(acans(1,1))*uu1+imag(acans(1,2))*uu2)
     1  *4.q0/x**2
c     ..........................................................
c     .  the sc calculation is independent of the reference frame  .
c     .    results are accumulated for each azimuthal mode m       .
c     ..........................................................
      do 80 n = 1,ns
        np = nms
        s = s+(abs(fg1(n))**2+abs(fg1(np))**2
     1        +abs(fg2(n))**2+abs(fg2(np))**2)*cmxnrm(n)
80    continue
        sc = s/x**2
c     ................................................
c     .  absorption cs = extinction cs - scattering cs  .
c     ................................................
      ab = ex-sc
      if(ab.lt.1.0q-08) ab = 0.q0
      write(6,120) sc,ab,ex
c     ..........................................................
c     .  test for complete convergence of                      .
c     .    solution at 8 out of 10 angles                      .
c     .    for each polarization                               .
c     .    if converged - stop                                 .
c     .    if not converged - return                           .
c     .   EXTENDT1 : status='unknown' is added in the open argument .
c     .              to prevent compiler warnings              .
c     .   EXTENDT1 : requires solution in all 10 angles        .
c     .              with an accuracy of 0.001                 .
c     ..........................................................
      if(nconv.ge.10.and.mconv.ge.10) then
        write(6,*) '*** solution has converged ***'
        if(ic.eq.2) then
          open(unit=12,file='case',status='unknown')
          rewind 12
          nt = kmv+1
          write(12,130) nt,nrank,x,aovrb,cmr
          close(unit=12)
          close(unit=11)
        end if
      else
        first = .false.
        return
      end if
      stop
100   format(/,12x,'***** m =',i3,' *****',/,
     1         2x,'differential scattering cross section',/,
     1         1x,'angle',9x,'parallel',8x,'perpendicular',/)
110   format(f6.2,2(5x,1pe13.4))
120   format(/,'scat xsect = ',1pe13.4,/,' abs xsect = ',
     1e13.4,/,' ext xsect = ',e13.4)
130   format(2i4,4f8.4)
      end
```

A.1.2. EXTENDT3

```
      program EXTENDT3
c     ..........................................................
c     .  Light Scattering by Particles: Computational Methods  .
c     .  by P.W. Barber and S.C. Hill                          .
c     .  copyright (c) 1990 by World Scientific Publishing Co Pte Ltd .
c     .                                                        .
c     .  Modifications 1996 by P. Alsholm and A. Nilsson       .
c     ..........................................................
c     ..........................................................
c     .  calculate the differential scattering cross section in all  .
c     .    directions by projecting the surface of a spherical       .
c     .    surface onto a rectangular coordinate system.             .
c     .    thsc: the theta scattering angle, is projected onto the   .
c     .          radius r in the rectangular plane                   .
c     .          r = thsc/180 degrees   (r varies from 0 to 1)       .
c     .          when r > 1, set scattering to backscatter value     .
c     .    phsc: the phi scattering angle, is the azimuthal angle in .
c     .          the spherical and rectangular coordinate systems    .
```

```
c     .                                                                .
c     .                                |                               .
c     .                      x     x  |  x      x                     .
c     .                                |                               .
c     .                      x     x  |  x      x                     .
c     .                    ----------------------- y                  .
c     .                      x     x  |  x      x                     .
c     .                                |                               .
c     .                      x     x  |  x      x                     .
c     .                                |                               .
c     .                                x                               .
c     .                                                                .
c     .  start at (x,y) = (-1,-1) and continue in the + x direction    .
c     .    then increment y by dlt and continue at (-1,-1+dlt)         .
c     .                                                                .
c     .  inputs:  angint = theta orientation in the laboratory frame   .
c     .           anginp = phi orientation in the laboratory frame     .
c     .           anglab = angle of the E-field polarization vector    .
c     .                    in the x-y plane                            .
c     .           npnts  = number of grid points = npnts x npnts       .
c     ..................................................................

c     ..........................................
c     .          EXTENDT3:                     .
c     . complex ----> double complex           .
c     . real     ----->  real*16               .
c     . alog ----> qlog                        .
c     . .0 ----> .q0 (cmplx(0.d0,0.d0))        .
c     . nrankmax = 200                         .
c     . common-blocks are changed              .
c     . open-arguments are changed             .
c     ..........................................
      implicit real*16 (a-h,o-z)
      double complex tmat,acans
      common dtr,rtd,pi
      common /mtxcom/ nrank,nranki,tmat(400,400)
      common /mtxcom2/ cmxnrm(200,201)
      common /cmvcom/ nm,kmv
      common /cmvcom2/ cmv,twm,prodm
      common /thtcom/ theta,sinth,costh,x
      common /uvccom/ angint,anginp,utheta,uphi,ic
      common /uvccom2/ thsc
      common /outcom/ total
      dimension cmx(200),ac(1000)
c     ..........................
c     .  set program constants  .
c     ..........................
      dtr = .017453292519943
      rtd = 57.2957795131
      pi = 3.14159265358979
c     ..................................................
c     .  read from file:                               .
c     .        the T-matrix and normalization constants .
c     .        nrank = number of terms (matrix order)   .
c     .        nm    = number of azimuthal modes m      .
c     .        x     = size parameter (ka)              .
c     ..................................................
      open(unit=11,file='t',form='unformatted',status='unknown')
      rewind 11
      open(unit=12,file='case',status='unknown')
      rewind 12
      read(12,100) nm,nrank,x
      close(unit=12)
      nranki = nrank+1
      write(6,110)
      write(6,*) 'enter angint,anginp,anglab,npnts'
      read(5,*) angint,anginp,anglab,npnts
c     ..............................................................
c     .  calculation for angint = 0 or 180 degrees uses m = 1 only  .
c     .    set the do 30 and do 50 loop indices for m = 1 only      .
c     .    read the stored arrays once to skip over the T-matrix    .
c     .    and normalization constants for m = 0                    .
c     ..............................................................
       if(angint.eq.0.q0.or.angint.eq.180.q0) then
        nl = 2
        nh = 2
        ic = 1
        read(11) tmat,cmx
c     ..........................................................
c     .  calculation for angint not 0 or 180 degrees ( all m )  .
c     .    set the do 30 and do 50 loop indices for all m       .
c     ..........................................................
      else
        nl = 1
        nh = nm
        ic = 2
      end if
      angint = angint*dtr
      anginp = anginp*dtr
      anglab = anglab*dtr
      tp = anginp
c     ..................................................................
c     .  utheta and uphi are the amplitudes of the incident field in  .
c     .    the theta (parallel) and phi (perpendicular) directions    .
c     ..................................................................
      utheta = cos(anglab)
```

```
      uphi = sin(anglab)
c     ..........................................
c     .  calculate the scattered field expansion  .
c     .    coefficients and the backscatter       .
c     .  enter a loop for each azimuthal mode m   .
c     ..........................................
      do 30 im = nl,nh
c     ............................
c     .  set m-dependent variables  .
c     ............................
        kmv = im-1
        cmv = real(kmv)
        prodm = 1.q0
        if(kmv.gt.0) then
          quanm = cmv
          do 10 i = 1,kmv
            quanm = quanm+1.q0
            prodm = quanm*prodm/2.q0
10        continue
        end if
        twm = 2.q0*cmv
        ij = kmv-1
        if(ij.lt.0) ij = 0
        ns = nrank-ij
c     ..........................................................
c     .  read the T-matrix and normalization constants for each m  .
c     ..........................................................
        read(11) tmat,cmx
        do 20 k = 1,ns
          cmxnrm(k,im) = cmx(k)
20      continue
c     ....................................................
c     .  calculate the scattered field expansion coefficients  .
c     .    and sum the backscatter over all azimuthal modes    .
c     ....................................................
        call addprc
30    continue
      close(unit=11)
c     ..................................................
c     .  calculate the logarithm of the total backscatter  .
c     .    differential scattering cross section           .
c     ..................................................
      bkscat = qlog10(total)
      open(unit=9,file='t3.dat',status='unknown')
      rewind 9
c     ..................................................
c     .  calculate the differential scattering cross      .
c     .    section over a rectangular grid corresponding  .
c     .    to the surface of a sphere                     .
c     ..................................................
      dlt = 2.q0/real(npnts-1)
c     ..................................
c     .  set starting y grid value (yg)  .
c     .  enter loop to vary y values     .
c     ..................................
      yg = -1.q0
      do 70 ig = 1,npnts
c     ..................................
c     .  set starting x grid value (xg)  .
c     .  enter loop to vary x values     .
c     ..................................
        xg = -1.q0
        do 60 jg = 1,npnts
          r = sqrt(xg**2+yg**2)
          if(r.lt.1.q0) then
c     ..........................................................
c     .  calculate the theta and phi scattering angles for radii  .
c     .    within the unit circle on the rectangular grid         .
c     ..........................................................
            thsc = r*pi
            phsc = 0.q0
            if(abs(xg).gt.0.q0.or.abs(yg).gt.0.q0) phsc = atan2(yg,xg)
c     ..........................................................
c     .  rotate the phi orientation angle anginp with phsc so that  .
c     .    the scattering at the angle phsc can be obtained in      .
c     .    the phi = zero degree plane in the laboratory frame      .
c     ..........................................................
            anginp = tp-phsc
            if(anginp.lt.0.q0) anginp = anginp+2.q0*pi
c     ..........................................
c     .  enter a loop for each azimuthal mode m  .
c     ..........................................
            do 50 im = nl,nh
c     ............................
c     .  set m-dependent constants  .
c     ............................
              kmv = im-1
              cmv = real(kmv)
              prodm = 1.q0
              if(kmv.gt.0) then
                quanm = cmv
                do 40 i = 1,kmv
                  quanm = quanm+1.q0
                  prodm = quanm*prodm/2.q0
40              continue
              end if
```

```
            twm = 2.q0*cmv
c     ..........................................................
c     .  calculate the differential scattering cross section at    .
c     .    each grid point - prcess is an alternate entry to       .
c     .    subroutine addprc for scattered field calculations      .
c     .    using the previously calculated expansion coefficients  .
c     ..........................................................
            call prcess
50         continue
c     ........................................
c     .  calculate the logarithm of the total     .
c     .    differential scattering cross section  .
c     .    at (xg,yg) and store in array ac(*)    .
c     ........................................
           ac(jg) = qlog10(total)
         else
c     ................................................
c     .  set the differential scattering cross section   .
c     .    to the backscatter value for all grid points  .
c     .    on or outside the unit circle                 .
c     ................................................
           ac(jg) = bkscat
         end if
c     ....................
c     .  increment x value  .
c     ....................
         xg = xg+dlt
60     continue
c     ..................................
c     .  write out data for all xg values  .
c     .    (for given yg value)            .
c     ..................................
       write(9,120) (ac(j),j=1,npnts)
c     ....................
c     .  increment y value  .
c     ....................
       yg = yg+dlt
70    continue
      close(unit=9)
      stop
100   format(2i4,f8.4)
110   format('................................................',/,
     1       '.  calculate scattered intensity in all directions  .',/,
     2       '.  output is written to t3.dat                      .',/,
     3       '................................................',//,
     4       'angint: theta orientation in the lab frame, degrees',/,
     5       'anginp: phi orientation in the lab frame, degrees'/,
     6       'anglab: angle of the E-field polarization',/,
     7       '        vector in the x-y plane, degrees',/,
     8       'npnts: number of grid points (npnts x npnts)',/)
120   format(e14.6)
      end
      subroutine genlgp(pnmllg,nc)
c     ..........................................................
c     .  calculate associated Legendre functions (argument    .
c     .    cos(theta)) divided by sin(theta) for each         .
c     .    azimuthal mode m                                   .
c     .  generate first two orders by formula and remaining   .
c     .    orders by recursion                                .
c     .                                                       .
c     .  pnmllg = associated Legendre function/sin(theta)     .
c     .  sinth = sin(theta), costh = cos(theta)               .
c     .  nc = number of orders (0 to nc-1)                    .
c     .  the order of the associated Legendre functions is    .
c     .    incremented by one in the pnmllg(*) array          .
c     ..........................................................
      implicit real*16 (a-h,o-z)
      common /cmvcom/ nm,kmv
      common /cmvcom2/ cmv,twm,prodm
      common /thtcom/ theta,sinth,costh,x
      dimension pnmllg(nc)
      dtwm = twm+1.q0
c     ..............................................................
c     .  calculate for the special case theta = 0 degrees or 180   .
c     .    degrees - note that for theta = 180 degrees, costh = -1,.
c     .    but theta is set to 0 degrees prior to subroutine entry .
c     ..............................................................
c     ..........................................................
c     .  if theta = 0 degrees and m.ne.1 all functions are zero  .
c     ..........................................................
      if(theta.eq.0.and.kmv.ne.1) then
        do 10 ilg = 1,nc
          pnmllg(ilg) = 0.q0
10      continue
        return
      end if
c     ..........................................................
c     .  if theta = 0 degrees and m.eq.1 calculate orders 0,1,2  .
c     ..........................................................
      if(theta.eq.0.and.kmv.eq.1) then
        pnmllg(1) = 0.q0
        pnmllg(2) = 1.q0
        pla = 1.q0
        plb = 3.q0*costh
        pnmllg(3) = plb
        ibeg = 4
```

```
c     ..............................................................
c     .  if theta.ne.0 degrees and m.eq.0 calculate orders 0 and 1  .
c     ..............................................................
      else if(kmv.eq.0) then
        pla = 1.q0/sinth
        plb = costh*pla
        pnmllg(1) = pla
        pnmllg(2) = plb
        ibeg = 3
c     ..........................................................
c     .  if theta.ne.0 degrees and m.ne.0 calculate first two  .
c     .    nonzero orders - the associated Legendre function   .
c     .    is zero for orders less than the azimuthal mode m   .
c     ..........................................................
      else
        do 20 ilg = 1,kmv
          pnmllg(ilg) = 0.q0
20      continue
        pla = prodm*sinth**(kmv-1)
        pnmllg(kmv+1) = pla
        plb = dtwm*costh*pla
        pnmllg(kmv+2) = plb
        ibeg = kmv+3
      end if
c     ...............................................
c     .  recur upward to obtain all remaining orders  .
c     ...............................................
      if(ibeg.gt.nc) return
      cnmul = real(2*ibeg-3)
      cnm = 2.q0
      cnmm = dtwm
      do 30 ilgr = ibeg,nc
        plc = (cnmul*costh*plb-cnmm*pla)/cnm
        pnmllg(ilgr) = plc
        pla = plb
        plb = plc
        cnmul = cnmul+2.q0
        cnm = cnm+1.q0
        cnmm = cnmm+1.q0
30    continue
      return
      end
      subroutine genang(n,tang,pang,cosb,sinb)
c     ..............................................................
c     .  calculate scattering angles in the particle frame corres-  .
c     .    ponding to scattering angles in the laboratory frame     .
c     .  calculate the elements of the transform matrix for the     .
c     .    polarization vectors                                     .
c     ..............................................................
      implicit real*16 (a-h,o-z)
      common dtr,rtd,pi
      common /uvccom/ angint,anginp,utheta,uphi,ic
      common /uvccom2/ thsc
      sint = sin(angint)
      cost = cos(angint)
      sinp = sin(anginp)
      cosp = cos(anginp)
c     ..............................................................
c     .  calculate scattering angles in the particle frame corres-  .
c     .    ponding to scattering angles in the laboratory frame     .
c     ..............................................................
      ps = 0.q0
      sints = sin(thsc)
      costs = cos(thsc)
      cosps = cos(ps)
      x = -cosp*cost*sints*cosps+sint*costs
      y = sinp*sints*cosps
      z = cosp*sint*sints*cosps+cost*costs
c     ..............................................................
c     .  calculate the theta scattering angles in the particle frame  .
c     ..............................................................
      r = sqrt(x**2+y**2)
      tang = 0.q0
      if(z.ne.0.q0.or.r.ne.0.q0) tang = atan2(r,z)
c     ..............................................................
c     .  calculate the phi scattering angles in the particle frame  .
c     ..............................................................
      pang = 0.q0
      if(x.ne.0.q0.or.y.ne.0.q0) pang = atan2(y,x)
      if(pang.lt.0.q0) pang = 2.q0*pi+pang
c     ...............................................
c     .  calculate the elements of the transform matrix  .
c     .    matrix for the polarization vectors           .
c     .  the transform matrix is needed only for the     .
c     .    incident polarization vector ( case n = 1 )   .
c     ...............................................
      if(n.eq.1) then
        cosb = -cosp
        sinb = sinp
      end if
      return
      end
      subroutine addprc
c     ..............................................................
c     .  calculate the scattered field expansion coefficients and   .
c     .    the differential scattering cross section at angles      .
```

```
c     .    tang, pang in the particle frame corresponding to       .
c     .    angles thsc, phsc in the laboratory frame               .
c     .  entry at addprc calculates the incident field expansion   .
c     .    coefficients and scattered field expansion coefficients .
c     .    which only need to be calculated once for each m        .
c     .  entry at prcess uses the previously calculated scattered  .
c     .    field expansion coefficients to calculate the           .
c     .    differential scattering cross section                   .
c     .  calculations are performed in the particle frame          .
c     ..............................................................
      implicit real*16 (a-h,o-z)
      double complex tmat,ab1(400),ab2(400),ci,c1,c2,cim,acans
      double complex fg1(400,201),fg2(400,201)
      common dtr,rtd,pi
      common /mtxcom/ nrank,nranki,tmat(400,400)
      common /mtxcom2/ cmxnrm(200,201)
      common /cmvcom/ nm,kmv
      common /cmvcom2/ cmv,twm,prodm
      common /thtcom/ theta,sinth,costh,x
      common /uvccom/ angint,anginp,utheta,uphi,ic
      common /uvccom2/ thsc
      common /outcom/ total
      dimension pnmllg(201),acans(2)
      logical first
      data first /.true./
c     ..............................................................
c     .  on first entry set the constants and:                     .
c     .    (1) calculate the incident field direction thtinc       .
c     .          in the particle frame                             .
c     .    (2) transform the incident field polarization           .
c     .          vector from the laboratory frame to the           .
c     .          particle frame                                    .
c     .          uu1 = theta component and uu2 = phi component     .
c     .    (3) set the backscatter angle thsc to 180 degrees       .
c     ..............................................................
      if(first) then
        ci = (0.d0,1.d0)
        nr2 = 2*nrank
        snorm = 1.q0/(pi*x**2)
        thsc = 0.q0
        call genang(1,tang,pang,cosb,sinb)
        thtinc = tang
        uu1 = cosb*utheta-sinb*uphi
        uu2 = sinb*utheta+cosb*uphi
        thsc = pi
        first = .false.
      end if
c     ..............................................
c     .  set indices for matrix compression when n < m  .
c     .    note: useful only when m > 1                 .
c     ..................................................
      ij = kmv-1
      if(ij.lt.0) ij = 0
      ijt = 2*ij
      ns = nrank-ij
      ns2 = 2*ns
c     ..............................................................
c     .  calculate the associated Legendre functions for the       .
c     .    incident angle = thtinc in the particle frame           .
c     ..............................................................
      theta = thtinc
      sinth = sin(theta)
      costh = cos(theta)
      call genlgp(pnmllg,nranki)
c     ..................................................
c     .  calculate the incident field coefficients:    .
c     .    ab1(*) = theta polarization                 .
c     .    ab2(*) = phi polarization                   .
c     .  note: ab1(*) and ab2(*) used in scattered     .
c     .        field programs are j times ab1(*) and   .
c     .        ab2(*) used in internal field programs  .
c     ..................................................
      do 10 n = 1,nrank
        if(n.le.ij) go to 10
        np = n+nrank
        cn = real(n)
        n1 = n+1
        c1 = 4.d0*ci**n
        c2 = 4.d0*ci**n1
        p1 = cn*costh*pnmllg(n1)-(cn+cmv)*pnmllg(n)
        p2 = cmv*pnmllg(n1)
        ab1(n-ij) = c1*p2*uu1
        ab1(np-ijt) = -c2*p1*uu1
        ab2(n-ij) = -c1*p1*uu2
        ab2(np-ijt) = c2*p2*uu2
10    continue
c     ..................................................
c     .  the scattered field coefficients = -[T] times   .
c     .    the incident field coefficients               .
c     .  store the coefficients for all azimuthal modes  .
c     .  fg2(ns+1) to fg2(ns2) is -g(omn)                .
c     ..................................................
      do 30 i = 1,ns2
        fg1(i,kmv+1) = 0.d0
        fg2(i,kmv+1) = 0.d0
        do 20 j = 1,ns2
```

```
            fg1(i,kmv+1) = fg1(i,kmv+1)-tmat(i,j)*ab1(j)
            fg2(i,kmv+1) = fg2(i,kmv+1)-tmat(i,j)*ab2(j)
20        continue
30      continue
c       ..........................................................
c       .  alternate entry to calculate the differential scattering  .
c       .    cross section using the previously calculated           .
c       .    scattered field expansion coefficients                  .
c       ..........................................................
        entry prcess
c       ..........................................................
c       .  clear the result accumulators and calculate the coordinate  .
c       .    transformation variables for each new grid point          .
c       ..........................................................
        if(ic.eq.1.or.kmv.eq.0) then
          acans(1) = 0.q0
          acans(2) = 0.q0
          call genang(2,tang,pang,cosb,sinb)
        end if
c       ..................................................
c       .  set indices for matrix compression when n < m  .
c       .    note: useful only when m > 1                 .
c       ..................................................
        ij = kmv-1
        if(ij.lt.0) ij = 0
        ijt = 2*ij
c       ..........................................................
c       .  evaluate the scattered field at each scattering angle  .
c       .    theta = tang and phi = pang in the particle frame    .
c       .  results are accumulated in acans(*) for each           .
c       .    azimuthal mode m                                     .
c       .  acans = kF, where F is a component of the vector       .
c       .    far-field amplitude                                  .
c       ..........................................................
c       ................................................
c       .  calculate the associated Legendre functions  .
c       .    at each scattering angle                   .
c       ................................................
        theta = tang
        sinth = sin(theta)
        costh = cos(theta)
        if(tang.eq.pi) then
          theta = 0.q0
          costh = -1.q0
        end if
        call genlgp(pnmllg,nranki)
        phi = cmv*pang
        sinphi = sin(phi)
        cosphi = cos(phi)
        do 40 n = 1,nrank
          if(n.le.ij) go to 40
          np = n+nrank
          n1 = n+1
          cn = real(n)
          p1 = cn*costh*pnmllg(n1)-(cn+cmv)*pnmllg(n)
          p2 = cmv*pnmllg(n1)
          aa = sinphi*p1
          bb = cosphi*p1
          cc = sinphi*p2
          dd = cosphi*p2
          cim = ci**(-n1)
          nij = n-ij
          npijt = np-ijt
c       ..........................................
c       .  solve for the theta polarized scattered  .
c       .    field in the particle frame            .
c       ..........................................
        acans(1) = acans(1)+cim*(fg1(nij,kmv+1)*dd+ci*fg1(npijt,kmv+1)*bb
     1  -fg2(nij,kmv+1)*cc-ci*fg2(npijt,kmv+1)*aa)*cmxnrm(nij,kmv+1)
c       ..........................................
c       .  solve for the phi polarized scattered  .
c       .    field in the particle frame          .
c       .  fg2(npijt) is -g(omn) per (3.12b)      .
c       ..........................................
        acans(2) = acans(2)-cim*(fg1(nij,kmv+1)*aa+ci*fg1(npijt,kmv+1)*cc
     1  +fg2(nij,kmv+1)*bb+ci*fg2(npijt,kmv+1)*dd)*cmxnrm(nij,kmv+1)
40      continue
c       ..........................................................
c       .  normalize the converged results and calculate the total       .
c       .    differential scattering cross section (both polarizations)  .
c       .    in the particle frame (same as in the laboratory frame)     .
c       ..........................................................
        if(ic.eq.1.or.kmv.eq.(nm-1)) then
          total = snorm*(abs(acans(1))**2+abs(acans(2))**2)
        end if
        return
        end
```

A.1.3. MCDATA

```
      program MCDATA
c     ..............................................................
c     .  Light Scattering by Particles: Computational Methods      .
c     .  by P.W. Barber and S.C. Hill                              .
c     .  copyright (c) 1990 by World Scientific Publishing Co Pte Ltd .
c     .                                                            .
c     .  Modifications 1996 by P. Alsholm and A. Nilsson           .
c     ..............................................................
c     ..............................................................
c     . The original program T3 calculates the differential        .
c     . scattering cross section in all directions by projecting   .
c     . the surface of a spherical surface onto a rectangular      .
c     . coordinate system. (See comments in the T3-code)           .
c     .                                                            .
c     . In the modified version MCDATA the probability of scattering .
c     . in all direction is evaluated for an amount of incident    .
c     . theta angles. The incident phi angle (anginp) is always zero .
c     . Output is written to mc.dat                                .
c     . Comments are included, where MCDATA differs from EXTENDT3  .
c     .                                                            .
c     .  Inputs to MCDATA:                                         .
c     .    nbrangint = number of different theta orientations in the .
c     .                laboratory frame (number of incident angles) .
c     .                ([0,dth,90];  dth = 90/(nbrangint-1)        .
c     .    anglab    = angle of the E-field polarization vector    .
c     .                in the x-y plane                            .
c     .    nbrthsc   = number of thsc (theta scattering angle)     .
c     .                ([0,dthsc,180) dthsc = 180/(nbrthsc-1))     .
c     .    nbrphsc   = number of phsc (phi scattering angle)       .
c     .                ([0,dphsc,180) dphsc = 180/(nbrphsc-1))     .
c     ..............................................................

c     ..............................................................
c     .          MCDATA :                                          .
c     . one loop for all angint = [0,dth,90]; dth = 90/(nbrangint-1) .
c     . nbrangint is a new variable                                .
c     .                                                            .
c     . algoritm:                                                  .
c     .                                                            .
c     . for angint = 0,dth,90                                      .
c     .  bkscat (thsc=180)                                         .
c     .  for thsc = 0,dthsc,180-dthsc   dthsc=180/(nbrthsc-1)      .
c     .   if thsc = 0 then                                         .
c     .    forwscat                                                .
c     .   else                                                     .
c     .    for phsc = 0,dphsc,180       dphsc=180/(nbrphsc-1)      .
c     .      ac(th,ph) = probability of scattering in this direction .
c     .    end                                                     .
c     .  end                                                       .
c     . end                                                        .
c     .                                                            .
c     .  The precision of the output is increased.                 .
c     .  The size and dimension of the array ac() is increased.    .
c     .  A new commonblock is created containing the variable      .
c     .   first in order to reset this variable for every new angint .
c     .  first is declared in the main program as well             .
c     ..............................................................
      implicit real*16 (a-h,o-z)
      double complex tmat,acans
      logical first
      common dtr,rtd,pi
      common /mtxcom/ nrank,nranki,tmat(400,400)
      common /mtxcom2/ cmxnrm(200,201)
      common /cmvcom/ nm,kmv
      common /cmvcom2/ cmv,twm,prodm
      common /thtcom/ theta,sinth,costh,x
      common /uvccom/ angint,anginp,utheta,uphi,ic
      common /uvccom2/ thsc
      common /outcom/ total
      common /newblock/ first
      dimension cmx(200),ac(361,361)
c     ........................
c     .  set program constants  .
c     ........................
      dtr = .017453292519943
      rtd = 57.2957795131
      pi = 3.14159265358979
c     ................................................
c     .  read from file:                               .
c     .       the T-matrix and normalization constants .
c     .       nrank = number of terms (matrix order)   .
c     .       nm    = number of azimuthal modes m      .
c     .       x     = size parameter (ka)              .
c     .                                                .
c     . MCDATA: new input parameters                   .
c     .         new variables (dtheta, dphiold, forwelem).
```

```
c     ................................................
      open(unit=12,file='case',status='unknown')
      rewind 12
      read(12,100) nm,nrank,x
      close(unit=12)
      nranki = nrank+1
      write(6,110)
      write(6,*) 'enter nbrangint,anglab,nbrthsc,nbrphsc'
      read(5,*) nbrangint,anglab,nbrthsc,nbrphsc
      dtheta = pi/(nbrthsc - 1)
      dphiold = pi/(nbrphsc - 1)
      forwelem = 2*pi*(1.d0 - cos(dtheta/2))
c     ................................................:...............
c     .  calculation for angint = 0 or 180 degrees uses m = 1 only  .
c     .    set the do 30 and do 50 loop indices for m = 1 only      .
c     .    read the stored arrays once to skip over the T-matrix    .
c     .    and normalization constants for m = 0                    .
c     .                                                             .
c     .              MCDATA:                                        .
c     . one loop (80) for each incident angle                       .
c     . nbrangint = 1 is used for spheres (angint doesn't matter)   .
c     . nbrangint <> 1, angint is uniformly distributed [0,90]      .
c     . the t-file is opened and closed inside each loop            .
c     . anginp is set to zero                                       .
c     . angint is written to stdout                                 .
c     . first is set to 'true' in the beginning of each loop        .
c     ...............................................................
      open(unit=9,file='mc.dat',status='unknown')
      rewind 9
      do 80 iangint = 1,nbrangint
        if (nbrangint.eq.1) then
          angint = 0.q0
        else
          angint = 90.q0*(iangint-1)/(nbrangint-1)
        end if
        anginp = 0.q0
        write(*,*) (angint)
        open(unit=11,file='t',form='unformatted',status='unknown')
        rewind 11
        first = .true.
        if(angint.eq.0.q0) then
          nl = 2
          nh = 2
          ic = 1
          read(11) tmat,cmx
c     ..........................................................
c     .  calculation for angint not 0 or 180 degrees ( all m )  .
c     .    set the do 30 and do 50 loop indices for all m       .
c     .                                                         .
c     . MCDATA: angint is shifted to prevent underflow          .
c     ..........................................................
        else
          nl = 1
          nh = nm
          ic = 2
          angint = angint - 1.0q-3
        end if
        angint = angint*dtr
        anginp = anginp*dtr
        anglab = anglab*dtr
        tp = anginp
c     ..........................................................
c     .  utheta and uphi are the amplitudes of the incident field in  .
c     .    the theta (parallel) and phi (perpendicular) directions    .
c     ..........................................................
        utheta = cos(anglab)
        uphi = sin(anglab)
c     ..........................................
c     .  calculate the scattered field expansion  .
c     .    coefficients and the backscatter       .
c     .  enter a loop for each azimuthal mode m   .
c     ..........................................
        do 30 im = nl,nh
c     ............................
c     .  set m-dependent variables  .
c     ............................
          kmv = im-1
          cmv = real(kmv)
          prodm = 1.q0
          if(kmv.gt.0) then
            quanm = cmv
            do 10 i = 1,kmv
              quanm = quanm+1.q0
              prodm = quanm*prodm/2.q0
10          continue
          end if
          twm = 2.q0*cmv
          ij = kmv-1
          if(ij.lt.0) ij = 0
          ns = nrank-ij
c     ..........................................................
c     .  read the T-matrix and normalization constants for each m  .
c     ..........................................................
          read(11) tmat,cmx
          do 20 k = 1,ns
            cmxnrm(k,im) = cmx(k)
```

```
20        continue
c     ..........................................................
c     .  calculate the scattered field expansion coefficients  .
c     .    and sum the backscatter over all azimuthal modes    .
c     ..........................................................
          call addprc
30      continue
        close(unit=11)
c     ..........................................................
c     .  calculate the logarithm of the total backscatter     .
c     .    differential scattering cross section              .
c     .                                                       .
c     . MCDATA: no logarithm.                                 .
c     ..........................................................
        bkscat = total
c     ..........................................................
c     .  calculate the differential scattering cross sections .
c     ..........................................................
c     ...........................................
c     . MCDATA: enter a loop to vary thsc values   .
c     ...........................................
        do 70 ig = 1,nbrthsc-1
c     ...........................................
c     . MCDATA: enter a loop to vary phsc values    .
c     ...........................................
          do 60 jg = 1,nbrphsc
c     ..........................................................
c     .  calculate the theta and phi scattering angles        .
c     .                                                       .
c     .  MCDATA: the combination thsc = 0.0 and phsc <> 0.0   .
c     .          is not necessary.                            .
c     ..........................................................
          if (ig.ne.1.or.jg.eq.1) then
            thsc = pi*(ig-1)/(nbrthsc-1)
            phsc = pi*(jg-1)/(nbrphsc-1)
c     ..........................................................
c     .  rotate the phi orientation angle anginp with phsc so that  .
c     .    the scattering at the angle phsc can be obtained in      .
c     .    the phi = zero degree plane in the laboratory frame      .
c     ..........................................................
            anginp = tp-phsc
            if(anginp.lt.0.q0) anginp = anginp+2.q0*pi
c     ...........................................
c     .  enter a loop for each azimuthal mode m  .
c     ...........................................
            do 50 im = nl,nh
c     ............................
c     .  set m-dependent constants  .
c     ............................
            kmv = im-1
            cmv = real(kmv)
            prodm = 1.q0
            if(kmv.gt.0) then
              quanm = cmv
              do 40 i = 1,kmv
                quanm = quanm+1.q0
                prodm = quanm*prodm/2.q0
40            continue
            end if
            twm = 2.q0*cmv
c     ..........................................................
c     .  calculate the differential scattering cross section at    .
c     .    each grid point - prcess is an alternate entry to       .
c     .    subroutine addprc for scattered field calculations      .
c     .    using the previously calculated expansion coefficients  .
c     ..........................................................
            call prcess
50        continue
c     ..........................................................
c     .                MCDATA:                                 .
c     .           calculate the total                          .
c     .       differential scattering cross section            .
c     .      at (thsc,phsc) and store in array ac(*,*)         .
c     .                                                       .
c     . if thsc = 0 : forward scattering                       .
c     . else        : justify the intensity according to the area  .
c     .               with the factor dphiold/dphinew          .
c     .               dphiold: the phi angle between two phsc  .
c     .               dphinew: the phi angle for a surface element .
c     .                        with an area equal to the area of   .
c     .                        the forward (and back) scattering   .
c     .                        surface element (forwelem)          .
c     .                        dphinew is theta-dependent          .
c     . the factor 2 is multiplied to ac(*,*) while this value     .
c     . symbolizes the scattering of both phi and 360-phi          .
c     . This factor is not used for phi=0, phi=180 or theta=0      .
c     ..........................................................
          if (ig.eq.1) then
            forwscat = total
          else
            i = ig - 1
            dphinew = forwelem/(cos((i-0.5)*dtheta) -
     1                          cos((i+0.5)*dtheta))
            ac(ig,jg) = 2*total*dphiold/dphinew
            if (jg.eq.1.or.jg.eq.nbrphsc) then
              ac(ig,jg) = ac(ig,jg)/2
```

```
              end if
            end if
          end if
c     ........................
c     .  MCDATA:             .
c     .  increment phsc value .
c     ........................
60      continue
c     ..............................
c     .  MCDATA: increment thsc value .
c     ..............................
70    continue

c     ..................................................................
c     . THIS PART IS ONLY FOR MCDATA :                                 .
c     . ac(*,*) contains differential scattering cross-sections (dscs) .
c     .       in different scattering angels                           .
c     . bkscat and forwscat symbolizes back- and forward scattering    .
c     . sum : summation of all dscs                                    .
c     . divide ac by sum to get a probability of scattering in         .
c     .        different scattering angels                             .
c     . check is used to be sure that the probability of scattering    .
c     .         in all directions together is exactly 1.q0             .
c     .                                                                .
c     ..................................................................
      sum = bkscat
      do i = 2,nbrthsc-1
        do j = 1,nbrphsc
          sum = sum + ac(i,j)
        enddo
      enddo
      sum = sum + forwscat
      bkscat = bkscat/sum
      val = bkscat
      do i = 2,nbrthsc-1
        do j = 1,nbrphsc
          ac(i,j) = ac(i,j)/sum + val
          val = ac(i,j)
        enddo
      enddo
      forwscat = forwscat/sum + val
      check = forwscat
      do i = 2,nbrthsc-1
        do j = 1,nbrphsc
          ac(i,j) = ac(i,j)/check
        enddo
      enddo
      bkscat = bkscat/check
c     ..................................................................
c     . MCDATA: write out the backscatter value and the ac() values .
c     .     forwscat = 1.q0-ac(nbrthsc-1,nbrphsc)                    .
c     ..................................................................
      write(9,120) bkscat
      do i = 2,nbrthsc-1
        write(9,120) (ac(i,j),j=1,nbrphsc)
      enddo

80    continue
      close(unit=9)
      stop
100   format(2i4,f8.4)
110   format('.....................................................',/,
     1       '.  calculate scattered intensity in all directions  .',/,
     2       '.  output is written to mc.dat                      .',/,
     3       '.....................................................',//,
     4       'nbrangint: number of different incident theta angles',/,
     5       'anglab: angle of the E-field polarization',/,
     6       '        vector in the x-y plane, degrees',/,
     7       'nbrthsc: number of scattering angles (theta)',/,
     8       'nbrphsc: number of scattering angles (phi)',/)
c     ..........................................................
c     . MCDATA: the precision of the ouput is increased        .
c     ..........................................................
120   format(e14.9)
      end
      subroutine genlgp(pnmllg,nc)
c     ..........................................................
c     .  calculate associated Legendre functions (argument    .
c     .    cos(theta)) divided by sin(theta) for each         .
c     .    azimuthal mode m                                   .
c     .  generate first two orders by formula and remaining   .
c     .    orders by recursion                                .
c     .                                                       .
c     .  pnmllg = associated Legendre function/sin(theta)     .
c     .  sinth = sin(theta), costh = cos(theta)               .
c     .  nc = number of orders (0 to nc-1)                    .
c     .  the order of the associated Legendre functions is    .
c     .    incremented by one in the pnmllg(*) array          .
c     ..........................................................
      implicit real*16 (a-h,o-z)
      common /cmvcom/ nm,kmv
      common /cmvcom2/ cmv,twm,prodm
      common /thtcom/ theta,sinth,costh,x
      dimension pnmllg(nc)
      dtwm = twm+1.q0
c     ..................................................................
```

```
c     .  calculate for the special case theta = 0 degrees or 180     .
c     .    degrees - note that for theta = 180 degrees, costh = -1,  .
c     .    but theta is set to 0 degrees prior to subroutine entry   .
c     ..............................................................
c     ..........................................................
c     .  if theta = 0 degrees and m.ne.1 all functions are zero  .
c     ..........................................................
      if(theta.eq.0.and.kmv.ne.1) then
        do 10 ilg = 1,nc
          pnmllg(ilg) = 0.q0
10      continue
        return
      end if
c     ..........................................................
c     .  if theta = 0 degrees and m.eq.1 calculate orders 0,1,2  .
c     ..........................................................
      if(theta.eq.0.and.kmv.eq.1) then
        pnmllg(1) = 0.q0
        pnmllg(2) = 1.q0
        pla = 1.q0
        plb = 3.q0*costh
        pnmllg(3) = plb
        ibeg = 4
c     ..........................................................
c     .  if theta.ne.0 degrees and m.eq.0 calculate orders 0 and 1  .
c     ..........................................................
      else if(kmv.eq.0) then
        pla = 1.q0/sinth
        plb = costh*pla
        pnmllg(1) = pla
        pnmllg(2) = plb
        ibeg = 3
c     ..........................................................
c     .  if theta.ne.0 degrees and m.ne.0 calculate first two  .
c     .    nonzero orders - the associated Legendre function   .
c     .    is zero for orders less than the azimuthal mode m   .
c     ..........................................................
      else
        do 20 ilg = 1,kmv
          pnmllg(ilg) = 0.q0
20      continue
        pla = prodm*sinth**(kmv-1)
        pnmllg(kmv+1) = pla
        plb = dtwm*costh*pla
        pnmllg(kmv+2) = plb
        ibeg = kmv+3
      end if
c     ..............................................
c     .  recur upward to obtain all remaining orders  .
c     ..............................................
      if(ibeg.gt.nc) return
      cnmul = real(2*ibeg-3)
      cnm = 2.q0
      cnmm = dtwm
      do 30 ilgr = ibeg,nc
        plc = (cnmul*costh*plb-cnmm*pla)/cnm
        pnmllg(ilgr) = plc
        pla = plb
        plb = plc
        cnmul = cnmul+2.q0
        cnm = cnm+1.q0
        cnmm = cnmm+1.q0
30    continue
      return
      end
      subroutine genang(n,tang,pang,cosb,sinb)
c     ..............................................................
c     .  calculate scattering angles in the particle frame corres-  .
c     .    ponding to scattering angles in the laboratory frame     .
c     .  calculate the elements of the transform matrix for the     .
c     .    polarization vectors                                     .
c     ..............................................................
      implicit real*16 (a-h,o-z)
      common dtr,rtd,pi
      common /uvccom/ angint,anginp,utheta,uphi,ic
      common /uvccom2/ thsc
      sint = sin(angint)
      cost = cos(angint)
      sinp = sin(anginp)
      cosp = cos(anginp)
c     ..............................................................
c     .  calculate scattering angles in the particle frame corres-  .
c     .    ponding to scattering angles in the laboratory frame     .
c     ..............................................................
      ps = 0.q0
      sints = sin(thsc)
      costs = cos(thsc)
      cosps = cos(ps)
      x = -cosp*cost*sints*cosps+sint*costs
      y = sinp*sints*cosps
      z = cosp*sint*sints*cosps+cost*costs
c     ..............................................................
c     .  calculate the theta scattering angles in the particle frame  .
c     ..............................................................
      r = sqrt(x**2+y**2)
      tang = 0.q0
```

```
      if(z.ne.0.q0.or.r.ne.0.q0) tang = atan2(r,z)
c     ..............................................................
c     .  calculate the phi scattering angles in the particle frame  .
c     ..............................................................
      pang = 0.q0
      if(x.ne.0.q0.or.y.ne.0.q0) pang = atan2(y,x)
      if(pang.lt.0.q0) pang = 2.q0*pi+pang
c     ................................................
c     .  calculate the elements of the transform matrix  .
c     .    matrix for the polarization vectors           .
c     .  the transform matrix is needed only for the     .
c     .    incident polarization vector ( case n = 1 )   .
c     ................................................
      if(n.eq.1) then
        cosb = -cosp
        sinb = sinp
      end if
      return
      end
      subroutine addprc
c     ..............................................................
c     .  calculate the scattered field expansion coefficients and   .
c     .    the differential scattering cross section at angles      .
c     .    tang, pang in the particle frame corresponding to        .
c     .    angles thsc, phsc in the laboratory frame                .
c     .  entry at addprc calculates the incident field expansion    .
c     .    coefficients and scattered field expansion coefficients  .
c     .    which only need to be calculated once for each m         .
c     .  entry at prcess uses the previously calculated scattered   .
c     .    field expansion coefficients to calculate the            .
c     .    differential scattering cross section                    .
c     .  calculations are performed in the particle frame           .
c     .                                                             .
c     . MCDATA: the variable first can be set in the main program   .
c     .         a new commonblock for the variable first is created .
c     .         first is declared in another way                    .
c     ..............................................................
      implicit real*16 (a-h,o-z)
      double complex tmat,ab1(400),ab2(400),ci,c1,c2,cim,acans(2)
      double complex fg1(400,201),fg2(400,201)
      logical first
      common dtr,rtd,pi
      common /mtxcom/ nrank,nranki,tmat(400,400)
      common /mtxcom2/ cmxnrm(200,201)
      common /cmvcom/ nm,kmv
      common /cmvcom2/ cmv,twm,prodm
      common /thtcom/ theta,sinth,costh,x
      common /uvccom/ angint,anginp,utheta,uphi,ic
      common /uvccom2/ thsc
      common /outcom/ total
      common /newblock/ first
      dimension pnmllg(201)
c     ..............................................................
c     .  on first entry set the constants and:                      .
c     .    (1) calculate the incident field direction thtinc       .
c     .          in the particle frame                             .
c     .    (2) transform the incident field polarization           .
c     .          vector from the laboratory frame to the           .
c     .          particle frame                                    .
c     .          uu1 = theta component and uu2 = phi component     .
c     .    (3) set the backscatter angle thsc to 180 degrees       .
c     ..............................................................
      if(first) then
        ci = (0.d0,1.d0)
        nr2 = 2*nrank
        snorm = 1.q0/(pi*x**2)
        thsc = 0.q0
        call genang(1,tang,pang,cosb,sinb)
        thtinc = tang
        uu1 = cosb*utheta-sinb*uphi
        uu2 = sinb*utheta+cosb*uphi
        thsc = pi
        first = .false.
      end if
c     ..................................................
c     .  set indices for matrix compression when n < m  .
c     .    note: useful only when m > 1                 .
c     ..................................................
      ij = kmv-1
      if(ij.lt.0) ij = 0
      ijt = 2*ij
      ns = nrank-ij
      ns2 = 2*ns
c     ..........................................................
c     .  calculate the associated Legendre functions for the   .
c     .    incident angle = thtinc in the particle frame       .
c     ..........................................................
      theta = thtinc
      sinth = sin(theta)
      costh = cos(theta)
      call genlgp(pnmllg,nranki)
c     ..............................................
c     .  calculate the incident field coefficients:  .
c     .    ab1(*) = theta polarization               .
c     .    ab2(*) = phi polarization                 .
c     .  note: ab1(*) and ab2(*) used in scattered   .
```

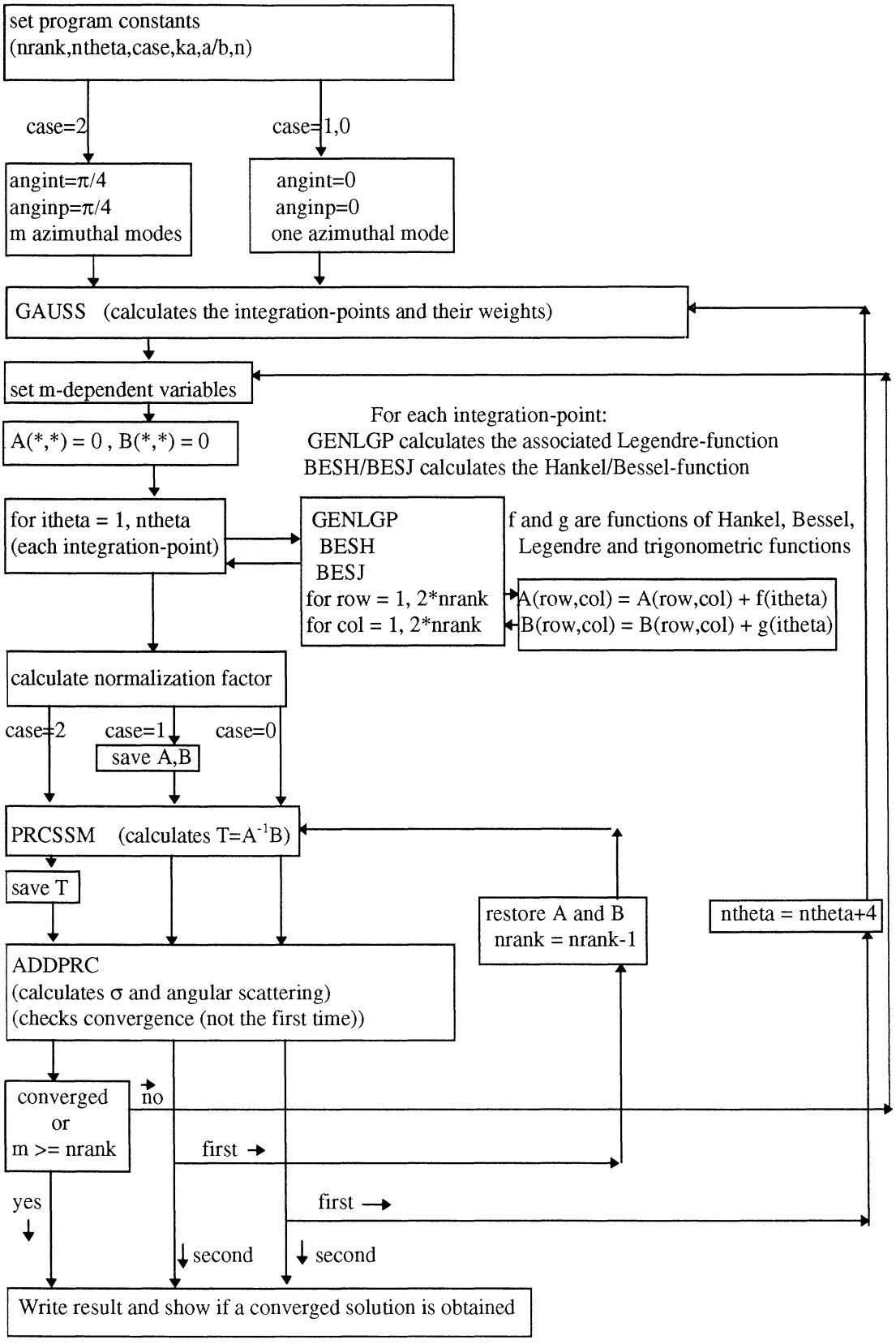
```
c     .        field programs are j times ab1(*) and    .
c     .        ab2(*) used in internal field programs   .
c     ..........................................
      do 10 n = 1,nrank
        if(n.le.ij) go to 10
        np = n+nrank
        cn = real(n)
        n1 = n+1
        c1 = 4.d0*ci**n
        c2 = 4.d0*ci**n1
        p1 = cn*costh*pnmllg(n1)-(cn+cmv)*pnmllg(n)
        p2 = cmv*pnmllg(n1)
        ab1(n-ij) = c1*p2*uu1
        ab1(np-ijt) = -c2*p1*uu1
        ab2(n-ij) = -c1*p1*uu2
        ab2(np-ijt) = c2*p2*uu2
10    continue
c     ..................................................
c     .  the scattered field coefficients = -[T] times   .
c     .    the incident field coefficients               .
c     .  store the coefficients for all azimuthal modes  .
c     .  fg2(ns+1) to fg2(ns2) is -g(omn)                .
c     ..........................................
      do 30 i = 1,ns2
        fg1(i,kmv+1) = (0.d0,0.d0)
        fg2(i,kmv+1) = (0.d0,0.d0)
        do 20 j = 1,ns2
          fg1(i,kmv+1) = fg1(i,kmv+1)-tmat(i,j)*ab1(j)
          fg2(i,kmv+1) = fg2(i,kmv+1)-tmat(i,j)*ab2(j)
20      continue
30    continue
c     ..........................................................
c     .  alternate entry to calculate the differential scattering  .
c     .    cross section using the previously calculated           .
c     .    scattered field expansion coefficients                  .
c     ..........................................................
        entry prcess
c     ..............................................................
c     .  clear the result accumulators and calculate the coordinate  .
c     .    transformation variables for each new grid point          .
c     ..............................................................
      if(ic.eq.1.or.kmv.eq.0) then
        acans(1) = (0.d0,0.d0)
        acans(2) = (0.d0,0.d0)
        call genang(2,tang,pang,cosb,sinb)
      end if
c     ..................................................
c     .  set indices for matrix compression when n < m  .
```

```
c     .    note: useful only when m > 1                .
c     ..................................................
      ij = kmv-1
      if(ij.lt.0) ij = 0
      ijt = 2*ij
c     ..........................................................
c     .  evaluate the scattered field at each scattering angle  .
c     .    theta = tang and phi = pang in the particle frame    .
c     .  results are accumulated in acans(*) for each           .
c     .    azimuthal mode m                                     .
c     .  acans = kF, where F is a component of the vector       .
c     .    far-field amplitude                                  .
c     ..........................................................
c     ..................................................
c     .  calculate the associated Legendre functions  .
c     .    at each scattering angle                   .
c     ..................................................
      theta = tang
      sinth = sin(theta)
      costh = cos(theta)
      if(tang.eq.pi) then
        theta = 0.q0
        costh = -1.q0
      end if
      call genlgp(pnmllg,nranki)
      phi = cmv*pang
      sinphi = sin(phi)
      cosphi = cos(phi)
      do 40 n = 1,nrank
        if(n.le.ij) go to 40
        np = n+nrank
        n1 = n+1
        cn = real(n)
        p1 = cn*costh*pnmllg(n1)-(cn+cmv)*pnmllg(n)
        p2 = cmv*pnmllg(n1)
        aa = sinphi*p1
        bb = cosphi*p1
        cc = sinphi*p2
        dd = cosphi*p2
        cim = ci**(-n1)
        nij = n-ij
        npijt = np-ijt
c     ..............................................
c     .  solve for the theta polarized scattered   .
c     .    field in the particle frame             .
c     ..............................................
      acans(1) = acans(1)+cim*(fg1(nij,kmv+1)*dd+ci*fg1(npijt,kmv+1)*bb
     1  -fg2(nij,kmv+1)*cc-ci*fg2(npijt,kmv+1)*aa)*cmxnrm(nij,kmv+1)
```

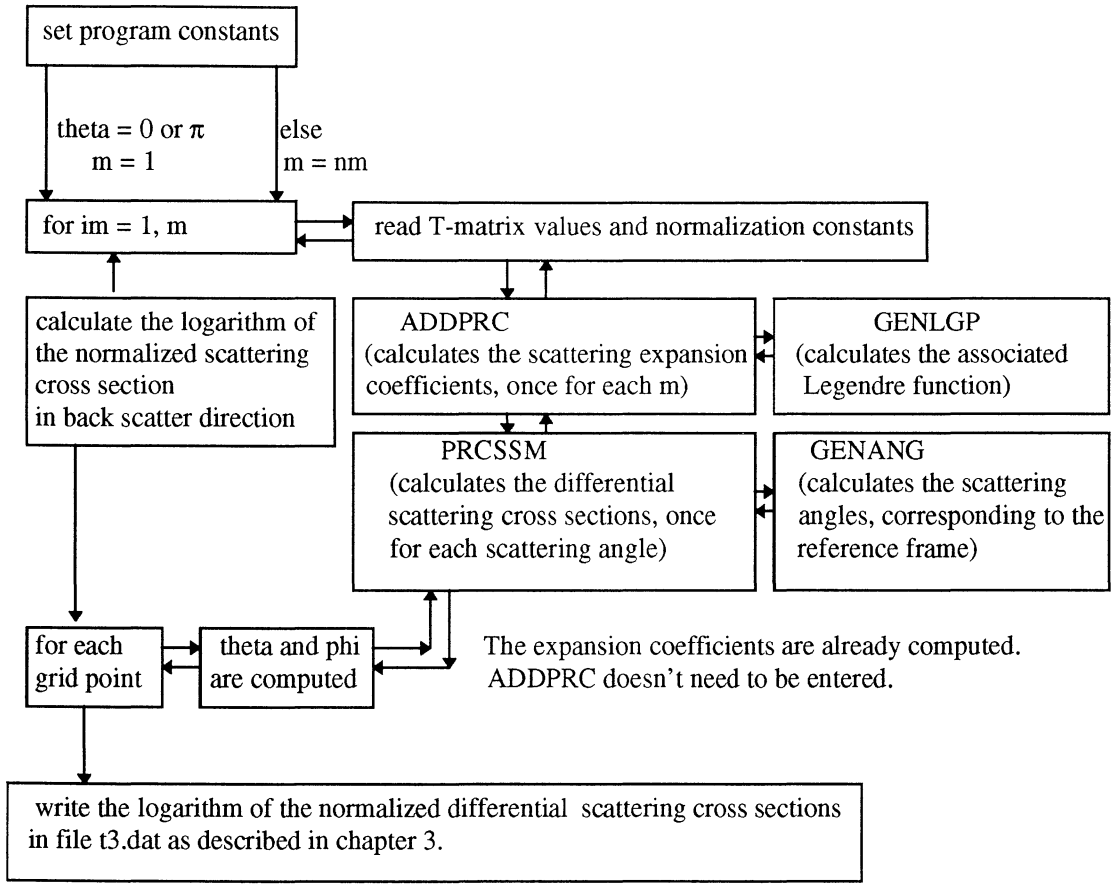
```
c     ........................................
c     .  solve for the phi polarized scattered  .
c     .    field in the particle frame          .
c     .  fg2(npijt) is -g(omn) per (3.12b)      .
c     ........................................
      acans(2) = acans(2)-cim*(fg1(nij,kmv+1)*aa+ci*fg1(npijt,kmv+1)*cc
     1  +fg2(nij,kmv+1)*bb+ci*fg2(npijt,kmv+1)*dd)*cmxnrm(nij,kmv+1)
40    continue
c     ..............................................................
c     .  normalize the converged results and calculate the total       .
c     .    differential scattering cross section (both polarizations)  .
c     .    in the particle frame (same as in the laboratory frame)     .
c     ..............................................................
      if(ic.eq.1.or.kmv.eq.(nm-1)) then
        total = snorm*(abs(acans(1))**2+abs(acans(2))**2)
      end if
      return
      end
```

A.2. Flow charts

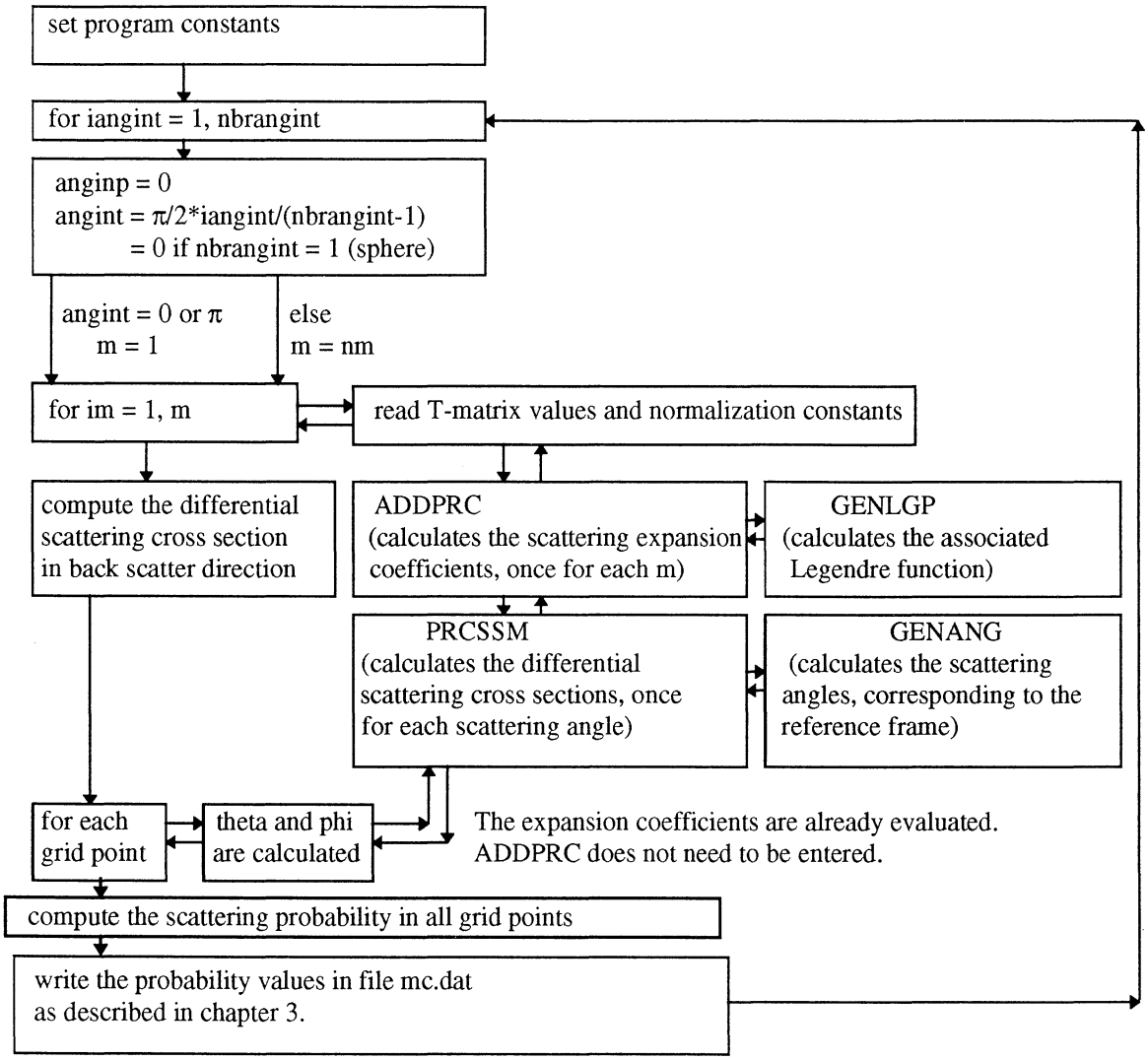
A.2.1. EXTENDT1



A.2.2. EXTENDT3



A.2.3. MCDATA


set program constants
for iangint = 1, nbrangint
anginp = 0
angint = π/2*iangint/(nbrangint-1)
= 0 if nbrangint = 1 (sphere)
angint = 0 or π
m = 1
else
m = nm
for im = 1, m
read T-matrix values and normalization constants
compute the differential scattering cross section in back scatter direction
ADDPRC
(calculates the scattering expansion coefficients, once for each m)
GENLGP
(calculates the associated Legendre function)
PRCSSM
(calculates the differential scattering cross sections, once for each scattering angle)
GENANG
(calculates the scattering angles, corresponding to the reference frame)
for each grid point
theta and phi are calculated
The expansion coefficients are already evaluated.
ADDPRC does not need to be entered.
compute the scattering probability in all grid points
write the probability values in file mc.dat
as described in chapter 3.

APPENDIX B: The Monte Carlo computer programs

B.1. Computer code listing

B.1.1. BOUNMCML

B.1.1.1. mcml.h

```
/****************************************************************
 *      Monte Carlo simulation of photon distribution in
 *      multi-layered turbid media in ANSI Standard C.
 ****
 *      Starting Date: 10/1991.
 *      Current Date:  6/1992.
 *
 *      Lihong Wang, Ph. D.
 *      Steven L. Jacques, Ph. D.
 *      Laser Biology Research Laboratory - 17
 *      M.D. Anderson Cancer Center
 *      University of Texas
 *      1515 Holcombe Blvd.
 *      Houston, TX 77030
 *      USA
 *
 *      This program was based on:
 *      (1) The Pascal code written by Marleen Keijzer and
 *      Steven L. Jacques in this laboratory in 1989, which
 *      deals with multi-layered turbid media.
 *
 *      (2) Algorithm for semi-infinite turbid medium by
 *      S.A. Prahl, M. Keijzer, S.L. Jacques, A.J. Welch,
 *      SPIE Institute Series Vol. IS 5 (1989), and by
 *      A.N. Witt, The Astrophysical journal Supplement
 *      Series 35, 1-6 (1977).
 *
 *      Major modifications include:
 *              . Conform to ANSI Standard C.
 *              . Removal of limit on number of array elements,
 *                because arrays in this program are dynamically
 *                allocated. This means that the program can accept
 *                any number of layers or gridlines as long as the
 *                memory permits.
 *              . Avoiding global variables whenever possible. This
 *                program has not used global variables so far.
 *              . Grouping variables logically using structures.
 *              . Top-down design, keep each subroutine clear &
 *                short.
 *              . Reflectance and transmittance are angularly
 *                resolved.
 ****
 * Modifications 1996 by P. Alsholm and A. Nilsson including
 * lateral boundaries.
 ****
 *      General Naming Conventions:
 *      Preprocessor names: all capital letters,
 *              e.g. #define PREPROCESSORS
 *      Globals: first letter of each word is capital, no
 *              underscores,
 *              e.g. short GlobalVar;
 *      Dummy variables:  first letter of each word is capital,
 *              and words are connected by underscores,
 *              e.g. void NiceFunction(char Dummy_Var);
 *      Local variables:  all lower cases, words are connected
 *              by underscores,
 *              e.g. short local_var;
 *      Function names or data types:  same as Globals.
 *
 ****
 *      Dimension of length: cm.
 *
 ****/

#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <time.h>
#include <string.h>
#include <ctype.h>

#define PI 3.1415926
#define WEIGHT 1E-4             /* Critical weight for roulette. */
```

```
#define CHANCE 0.1              /* Chance of roulette survival. */
#define STRLEN 256              /* String length. */

#define Boolean char

#define SIGN(x) ((x)>=0 ? 1:-1)

/***************** Stuctures *****************************/

/****
 *      Structure used to describe a photon packet.
 ****/
typedef struct {
  double x, y ,z;       /* Cartesian coordinates.[cm] */
  double ux, uy, uz;     /* directional cosines of a photon. */
  double w;                     /* weight. */
  Boolean dead;                 /* 1 if photon is terminated. */
  short layer;          /* index to layer where the photon */
                                /* packet resides. */
  double s;                     /* current step size. [cm]. */
  double sleft;                 /* step size left. dimensionless [-]. */
} PhotonStruct;

/****
 * Structure used to describe the geometry and optical
 * properties of a layer.
 * z0 and z1 are the z coordinates for the upper boundary
 * and lower boundary respectively.
 *
 * cos_crit0 and cos_crit1 are the cosines of the
 * critical angle of total internal reflection for the
 * upper boundary and lower boundary respectively.
 * cos_critxy is the cosine of the critical angle of
 * total internal reflection for the boundary to the
 * surrounding glass.
 * They are set to zero if no total internal reflection
 * exists.
 * They are used for computation speed.
 ****/

typedef struct {
  double z0, z1;           /* z coordinates of a layer. [cm] */
  double n;               /* refractive index of a layer. */
  double mua;       /* absorption coefficient. [1/cm] */
  double mus;       /* scattering coefficient. [1/cm] */
  double g;                 /* anisotropy. */

  double cos_crit0, cos_crit1, cos_critxy;
} LayerStruct;
```

```
/****
 *  Input parameters for each independent run.
 *
 *  z and r are for the cylindrical coordinate system. [cm]
 *  a is for the angle alpha between the photon exiting
 *  direction and the surface normal. [radian]
 *
 *  The grid line separations in z, r, and alpha
 *  directions are dz, dr, and da respectively.  The numbers
 *  of grid lines in z, r, and alpha directions are
 *  nz, nr, and na respectively.
 *
 *  The member layerspecs will point to an array of
 *  structures which store parameters of each layer.
 *  This array has (number_layers + 3) elements. One
 *  element is for a layer.
 *  The layers 0 and (num_layers + 1) are for top ambient
 *  medium and the bottom ambient medium respectively.
 *  The layer (num_layers + 2) is for the surrounding glass.
 ****/
typedef struct {
  char   out_fname[STRLEN];    /* output file name. */
  char   out_fformat;          /* output file format. */
                                                  /* 'A' for ASCII, */
                                                  /* 'B' for binary. */
  long   num_photons;          /* to be traced. */
  double Wth;                          /* play roulette if photon */
                                                  /* weight < Wth.*/

  double dz;                           /* z grid separation.[cm] */
  double dr;                           /* r grid separation.[cm] */
  double da;                           /* alpha grid separation. */
                                                  /* [radian] */
  short nz;                                  /* array range 0..nz-1. */
  short nr;                                  /* array range 0..nr-1. */
  short na;                                  /* array range 0..na-1. */

  short num_layers;                    /* number of layers. */

  double x_size, y_size;    /* dimension of the sample. */

  LayerStruct * layerspecs;   /* layer parameters. */
} InputStruct;

/****
 *      Structures for scoring physical quantities.
 *      z and r represent z and r coordinates of the
 *      cylindrical coordinate system. [cm]
 *      a is the angle alpha between the photon exiting
 *      direction and the normal to the surfaces. [radian]
```

```
 *      See comments of the InputStruct.
 *      See manual for the physcial quantities.
 ****/
typedef struct {
  double    Rsp;       /* specular reflectance. [-] */
  double ** Rd_ra;     /* 2D distribution of diffuse */
                               /* reflectance. [1/(cm2 sr)] */
  double *  Rd_r;      /* 1D radial distribution of diffuse */
                               /* reflectance. [1/cm2] */
  double *  Rd_a;      /* 1D angular distribution of diffuse */
                               /* reflectance. [1/sr] */
  double    Rd;              /* total diffuse reflectance. [-] */

  double ** A_rz;      /* 2D probability density in turbid */
                               /* media over r & z. [1/cm3] */
  double *  A_z;       /* 1D probability density over z. */
                               /* [1/cm] */
  double *  A_l;       /* each layer's absorption */
                               /* probability. [-] */
  double    A;         /* total absorption probability. [-] */

  double ** Tt_ra;     /* 2D distribution of total */
                               /* transmittance. [1/(cm2 sr)] */
  double *  Tt_r;      /* 1D radial distribution of */
                               /* transmittance. [1/cm2] */
  double *  Tt_a;      /* 1D angular distribution of */
                               /* transmittance. [1/sr] */
  double    Tt;              /* total transmittance. [-] */
} OutStruct;

/***********************************************************
 *      Routine prototypes for dynamic memory allocation and
 *      release of arrays and matrices.
 *      Modified from Numerical Recipes in C.
 ****/
double  *AllocVector(short, short);
double **AllocMatrix(short, short,short, short);
void     FreeVector(double *, short, short);
void     FreeMatrix(double **, short, short, short, short);
void     nrerror(char *);
```

B.1.1.2. mcmlmain.c

```
/***********************************************************
 *      main program for Monte Carlo simulation of photon
 *      distribution in multi-layered turbid media.
 *
 ****/

/****
 *      THINKCPROFILER is defined to generate profiler calls in
 *      Think C. If 1, remember to turn on "Generate profiler
 *      calls" in the options menu.
 ****/
#define THINKCPROFILER 0

/* GNU cc does not support difftime() and CLOCKS_PER_SEC.*/
#define GNUCC 0

#if THINKCPROFILER
#include <profile.h>
#include <console.h>
#endif

#include "mcml.h"

/*      Declare before they are used in main(). */
FILE *GetFile(char *);

short ReadNumRuns(FILE* );
void ReadParm(FILE* , InputStruct * );
void CheckParm(FILE* , InputStruct * );
void InitOutputData(InputStruct, OutStruct *);
void FreeData(InputStruct, OutStruct *);
double Rspecular(LayerStruct * );
void LaunchPhoton(double, LayerStruct *, PhotonStruct *);
void HopDropSpin(InputStruct  *,PhotonStruct *,OutStruct *);
void SumScaleResult(InputStruct, OutStruct *);
void WriteResult(InputStruct, OutStruct, char *);


/***********************************************************
 *      If F = 0, reset the clock and return 0.
 *
 *      If F = 1, pass the user time to Msg and print Msg on
 *      screen, return the real time since F=0.
 *
 *      If F = 2, same as F=1 except no printing.
 *
 *      Note that clock() and time() return user time and real
 *      time respectively.
 *      User time is whatever the system allocates to the
 *      running of the program;
 *      real time is wall-clock time.  In a time-shared system,
```

```
 *      they need not be the same.
 *
 *      clock() only hold 16 bit integer, which is about 32768
 *      clock ticks.
 ****/
time_t PunchTime(char F, char *Msg)
{
#if GNUCC
  return(0);
#else
  static clock_t ut0;  /* user time reference. */
  static time_t  rt0;  /* real time reference. */
  double secs;
  char s[STRLEN];

  if(F==0) {
    ut0 = clock();
    rt0 = time(NULL);
    return(0);
  }
  else if(F==1)  {
    secs = (clock() - ut0)/(double)CLOCKS_PER_SEC;
    if (secs<0) secs=0;       /* clock() can overflow. */
    sprintf(s, "User time: %8.0lf sec = %8.2lf hr.  %s\n",
            secs, secs/3600.0, Msg);
    puts(s);
    strcpy(Msg, s);
    return(difftime(time(NULL), rt0));
  }
  else if(F==2) return(difftime(time(NULL), rt0));
  else return(0);
#endif
}

/*************************************************************
 *      Print the current time and the estimated finishing time.
 *
 *      P1 is the number of computed photon packets.
 *      Pt is the total number of photon packets.
 ****/
void PredictDoneTime(long P1, long Pt)
{
  time_t now, done_time;
  struct tm *date;
  char s[80];

  now = time(NULL);
  date = localtime(&now);
  strftime(s, 80, "%H:%M %x", date);
  printf("Now %s, ", s);
  done_time = now +
                        (time_t) (PunchTime(2,"")*(Pt-P1)/(double)P1);
  date = localtime(&done_time);
  strftime(s, 80, "%H:%M %x", date);
  printf("End %s\n", s);
}

/*************************************************************
 *      Report estimated time, number of photons and runs left
 *      after calculating 10 photons or every 1/10 of total
 *      number of photons.
 *
 *      Num_Runs is the number of runs left.
 *      Pi is the index to the current photon, counting down.
 *      Pt is the total number of photons.
 ****/
void ReportStatus(short Num_Runs,long Pi,long Pt)
{
  if(Pt-Pi == 10 || Pi*10%Pt == 0 && Pi != Pt) {
    printf("%ld photons & %hd runs left, ", Pi, Num_Runs);
    PredictDoneTime(Pt-Pi, Pt);
  }
}

/*************************************************************
 *      Report time and write results.
 ****/
void ReportResult(InputStruct In_Parm, OutStruct Out_Parm)
{
  char time_report[STRLEN];

  strcpy(time_report, " Simulation time of this run.");
  PunchTime(1, time_report);

  SumScaleResult(In_Parm, &Out_Parm);
  WriteResult(In_Parm, Out_Parm, time_report);
}

/*************************************************************
 *      Get the file name of the input data file from the
 *      argument to the command line.
 ****/
void GetFnameFromArgv(int argc,
                                        char * argv[],
                                        char * input_filename)
{
  if(argc>=2) {                        /* filename in command line */
    strcpy(input_filename, argv[1]);
  }
```

```
  else
    input_filename[0] = '\0';
}


/***********************************************************
 *      Execute Monte Carlo simulation for one independent run.
 ****/
void DoOneRun(short NumRuns, InputStruct *In_Ptr)
{
  register long i_photon;
       /* index to photon. register for speed.*/
  OutStruct out_parm;          /* distribution of photons.*/
  PhotonStruct photon;

#if THINKCPROFILER
  InitProfile(200,200); cecho2file("prof.rpt",0, stdout);
#endif

  InitOutputData(*In_Ptr, &out_parm);
  out_parm.Rsp = Rspecular(In_Ptr->layerspecs);
  i_photon = In_Ptr->num_photons;
  PunchTime(0, "");

  do {
    ReportStatus(NumRuns, i_photon, In_Ptr->num_photons);
    LaunchPhoton(out_parm.Rsp, In_Ptr->layerspecs, &photon);
    do  HopDropSpin(In_Ptr, &photon, &out_parm);
    while (!photon.dead);
  } while(--i_photon);

#if THINKCPROFILER
  exit(0);
#endif

  ReportResult(*In_Ptr, out_parm);
  FreeData(*In_Ptr, &out_parm);
}

/***********************************************************
 *      The argument to the command line is filename, if any.
 *      Macintosh does not support command line.
 ****/
char main(int argc, char *argv[])
{
  char input_filename[STRLEN];
  FILE *input_file_ptr;
  short num_runs;      /* number of independent runs. */
  InputStruct in_parm;

  ShowVersion("Version 1.0, 1992");
  GetFnameFromArgv(argc, argv, input_filename);
  input_file_ptr = GetFile(input_filename);
  CheckParm(input_file_ptr, &in_parm);
  num_runs = ReadNumRuns(input_file_ptr);

  while(num_runs--)  {
    ReadParm(input_file_ptr, &in_parm);
       DoOneRun(num_runs, &in_parm);
  }

  fclose(input_file_ptr);
  return(0);
}
```

B.1.1.3. mcmlio.c

```
/***********************************************************
 *      Input/output of data.
 ****/

#include "mcml.h"

/***********************************************************
 *      Structure used to check against duplicated file names.
 ****/
struct NameList {
  char name[STRLEN];
  struct NameList * next;
};

typedef struct NameList NameNode;
typedef NameNode * NameLink;


/***********************************************************
 *      Center a string according to the column width.
```

```
 ****/
char * CenterStr(short  Wid,
                 char * InStr,
                 char * OutStr)
{
  size_t nspaces;      /* number of spaces to be filled */
                       /* before InStr. */

  nspaces = (Wid - strlen(InStr))/2;
  if(nspaces<0) nspaces = 0;

  strcpy(OutStr, "");
  while(nspaces--)  strcat(OutStr, " ");

  strcat(OutStr, InStr);

  return(OutStr);
}

/*************************************************************
 *      Print some messages before starting simulation.
 *      e.g. author, address, program version, year.
 ****/
#define COLWIDTH 80
void ShowVersion(char *version)
{
  char str[STRLEN];

  CenterStr(COLWIDTH,
       "Monte Carlo Simulation of Multi-layered Turbid Media",
       str);
  puts(str);
  puts("");

  CenterStr(COLWIDTH, "Lihong Wang, Ph. D.", str);
  puts(str);

  CenterStr(COLWIDTH, "Steven L. Jacques, Ph. D.", str);
  puts(str);

  CenterStr(COLWIDTH,
       "Laser Biology Research Laboratory - 17",str);
  puts(str);

  CenterStr(COLWIDTH, "M.D. Anderson Cancer Center", str);
  puts(str);

  CenterStr(COLWIDTH, "University of Texas", str);
  puts(str);

  CenterStr(COLWIDTH, "Houston, TX 77030", str);
  puts(str);

  CenterStr(COLWIDTH, "Fax: (713)792-3995", str);
  puts(str);
  puts("");

  CenterStr(COLWIDTH, version, str);
  puts(str);

  CenterStr(COLWIDTH, "modified 1996, PA - mcmlboun", str);
  puts(str);
  puts("");

  puts("\n\n\n\n");
}
#undef COLWIDTH

/*************************************************************
 *      Get a filename and open it for reading, retry until
 *      the file can be opened.  '.' terminates the program.
 *
 *      If Fname != NULL, try Fname first.
 ****/
FILE *GetFile(char *Fname)
{
  FILE * file=NULL;
  Boolean firsttime=1;

  do {
    if(firsttime && Fname[0]!='\0') {
         /* use the filename from command line */
      firsttime = 0;
    }
    else {
      printf("Input filename(or . to exit):");
      scanf("%s", Fname);
      firsttime = 0;
    }

    if(strlen(Fname) == 1 && Fname[0] == '.')
      exit(1);                /* exit if no filename entered. */

    file = fopen(Fname, "r");
  }  while(file == NULL);

  return(file);
}

/*************************************************************
```

```
 *      Kill the ith char (counting from 0), push the following
 *      chars forward by one.
 ****/
void KillChar(size_t i, char * Str)
{
  size_t sl = strlen(Str);

  for(;i<sl;i++) Str[i] = Str[i+1];
}

/*************************************************************
 *      Eliminate the chars in a string which are not printing
 *      chars or spaces.
 *
 *      Spaces include ' ', '\f', '\t' etc.
 *
 *      Return 1 if no nonprinting chars found, otherwise
 *      return 0.
 ****/
Boolean CheckChar(char * Str)
{
  Boolean found = 0;   /* found bad char. */
  size_t sl = strlen(Str);
  size_t i = 0;

  while(i<sl)
    if (Str[i]<0 || Str[i]>255)
      nrerror("Non-ASCII file\n");
    else if(isprint(Str[i]) || isspace(Str[i]))
      i++;
    else {
      found = 1;
      KillChar(i, Str);
      sl--;
    }

  return(found);
}

/*************************************************************
 *      Return 1 if this line is a comment line in which the
 *      first non-space character is "#".
 *
 *      Also return 1 if this line is space line.
 ****/
Boolean CommentLine(char *Buf)
{
  size_t spn, cspn;

  spn = strspn(Buf, " \t");
  /* length spanned by space or tab chars. */

  cspn = strcspn(Buf, "#\n");
  /* length before the 1st # or return. */

  if(spn == cspn)      /* comment line or space line. */
       return(1);
  else                          /* the line has data. */
       return(0);
}

/*************************************************************
 *      Skip space or comment lines and return a data line only.
 ****/
char * FindDataLine(FILE *File_Ptr)
{
  char buf[STRLEN];

  buf[0] = '\0';
  do {  /* skip space or comment lines. */
    if(fgets(buf, 255, File_Ptr) == NULL)  {
      printf("Incomplete data\n");
      buf[0]='\0';
      break;
    }
    else
      CheckChar(buf);
  } while(CommentLine(buf));

  return(buf);
}

/*************************************************************
 *      Skip file version, then read number of runs.
 ****/
short ReadNumRuns(FILE* File_Ptr)
{
  char buf[STRLEN];
  short n=0;

  FindDataLine(File_Ptr); /* skip file version. */

  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') nrerror("Reading number of runs\n");
  sscanf(buf, "%hd",&n);
  return(n);
}


/*************************************************************
```

```
 *      Read the file name and the file format.
 *
 *      The file format can be either A for ASCII or B for
 *      binary.
 ****/
void ReadFnameFormat(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in file name and format. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading file name and format.\n");
  sscanf(buf, "%s %c",
       In_Ptr->out_fname, &(In_Ptr->out_fformat) );
  if(toupper(In_Ptr->out_fformat) != 'B')
       In_Ptr->out_fformat = 'A';
}


/*********************************************************
 *      Read the number of photons.
 ****/
void ReadNumPhotons(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in number of photons. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading number of photons.\n");
  sscanf(buf, "%ld", &In_Ptr->num_photons);
  if(In_Ptr->num_photons<=0)
       nrerror("Nonpositive number of photons.\n");
}

/*********************************************************
 *      Read the members dz and dr.
 ****/
void ReadDzDr(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in dz, dr. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') nrerror("Reading dz, dr.\n");
  sscanf(buf, "%lf%lf", &In_Ptr->dz, &In_Ptr->dr);
  if(In_Ptr->dz<=0) nrerror("Nonpositive dz.\n");
  if(In_Ptr->dr<=0) nrerror("Nonpositive dr.\n");
}
```

```
/*********************************************************
 *      Read the members nz, nr, na.
 ****/
void ReadNzNrNa(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in number of dz, dr, da. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading number of dz, dr, da's.\n");
  sscanf(buf, "%hd%hd%hd",
       &In_Ptr->nz, &In_Ptr->nr, &In_Ptr->na);
  if(In_Ptr->nz<=0)
       nrerror("Nonpositive number of dz's.\n");
  if(In_Ptr->nr<=0)
       nrerror("Nonpositive number of dr's.\n");
  if(In_Ptr->na<=0)
       nrerror("Nonpositive number of da's.\n");
  In_Ptr->da = 0.5*PI/In_Ptr->na;
}

/*********************************************************
 *      Read the members x_size and y_size.
 ****/
void ReadXSizeYSize(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in x_size, y_size. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') nrerror("Reading x_size, y_size.\n");
  sscanf(buf, "%lf%lf", &In_Ptr->x_size, &In_Ptr->y_size);
  if(In_Ptr->x_size<=0) nrerror("Nonpositive x_size.\n");
  if(In_Ptr->y_size<=0) nrerror("Nonpositive y_size.\n");
}

/*********************************************************
 *      Read the number of layers.
 ****/
void ReadNumLayers(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in number of layers. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading number of layers.\n");
  sscanf(buf, "%hd", &In_Ptr->num_layers);
```

```
  if(In_Ptr->num_layers<=0)
       nrerror("Nonpositive number of layers.\n");
}


/************************************************************
 *      Read the refractive index n of the ambient.
 ****/
void ReadAmbient(FILE *File_Ptr,
                              LayerStruct * Layer_Ptr,
                              char *side)
{
  char buf[STRLEN], msg[STRLEN];
  double n;

  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') {
    sprintf(msg, "Rading n of %s ambient.\n", side);
       nrerror(msg);
  }

  sscanf(buf, "%lf", &n );
  if(n<=0) nrerror("Wrong n.\n");
  Layer_Ptr->n = n;
}

/************************************************************
 *      Read the parameters of one layer.
 *
 *      Return 1 if error detected.
 *      Return 0 otherwise.
 *
 *      *Z_Ptr is the z coordinate of the current layer, which
 *      is used to convert thickness of layer to z coordinates
 *      of the two boundaries of the layer.
 ****/
Boolean ReadOneLayer(FILE *File_Ptr,
                                    LayerStruct * Layer_Ptr,
                                    double *Z_Ptr)
{
  char buf[STRLEN];
  double d, n, mua, mus, g;    /* d is thickness. */

  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') return(1); /* error. */

  sscanf(buf, "%lf%lf%lf%lf%lf", &n, &mua, &mus, &g, &d);
  if(d<0 || n<=0 || mua<0 || mus<0 || g<0 || g>1)
    return(1);                /* error. */

  Layer_Ptr->n = n;
  Layer_Ptr->mua = mua;
  Layer_Ptr->mus = mus;
  Layer_Ptr->g   = g;
  Layer_Ptr->z0        = *Z_Ptr;
  *Z_Ptr += d;
  Layer_Ptr->z1        = *Z_Ptr;

  return(0);
}

/************************************************************
 *      Read and set the parameters of the surrounding glass.
 *
 *      Return 1 if error detected.
 *      Return 0 otherwise.
 *
 *      *Z_Ptr is the z coordinate of the current layer, which
 *      is used to convert thickness of layer to z coordinates
 *      of the two boundaries of the layer.
 ****/
Boolean ReadSurrGlass(FILE *File_Ptr,
                                    LayerStruct * Layer_Ptr,
                                    double *Z_Ptr)
{
  char buf[STRLEN];
  double n;

  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') return(1); /* error. */

  sscanf(buf, "%lf", &n);
  if(n<=0)
    return(1);                /* error. */

  Layer_Ptr->n = n;
  Layer_Ptr->mua = 0.0;
  Layer_Ptr->mus = 0.0;
  Layer_Ptr->z0 = 0.0;
  Layer_Ptr->z1 = *Z_Ptr;

  return(0);
}

/************************************************************
 *      Read the parameters of one layer at a time.
 ****/
void ReadLayerSpecs(FILE *File_Ptr,
                                    short Num_Layers,
                                    LayerStruct ** Layerspecs_PP)
```

```
{
  char msg[STRLEN];
  short i=0;
  double z = 0.0;      /* z coordinate of the current layer. */

  /* Allocate an array for the layer parameters. */
  /* layer 0 and layer Num_Layers + 1 are for ambient. */
  *Layerspecs_PP = (LayerStruct *)
    malloc((unsigned) (Num_Layers+3)*sizeof(LayerStruct));
  if (!(*Layerspecs_PP))
    nrerror("allocation failure in ReadLayerSpecs()");

  ReadAmbient(File_Ptr, &((*Layerspecs_PP)[i]), "top");
  for(i=1; i<=Num_Layers; i++)
    if(ReadOneLayer(File_Ptr, &((*Layerspecs_PP)[i]), &z)) {
      sprintf(msg, "Error reading %hd of %hd layers\n",
                    i, Num_Layers);
        nrerror(msg);
    }
  ReadAmbient(File_Ptr, &((*Layerspecs_PP)[i]), "bottom");
  if(ReadSurrGlass(File_Ptr, &((*Layerspecs_PP)[i+1]), &z)) {
      sprintf(msg, "Error reading surrounding glass");
        nrerror(msg);
    }
}

/*************************************************************
 *      Compute the critical angles for total internal
 *      reflection according to the relative refractive index
 *      of the layer.
 *      All layers are processed.
 ****/
void CriticalAngle( short Num_Layers,
                                    LayerStruct ** Layerspecs_PP)
{
  short i=0;
  double n1, n2;

  for(i=1; i<=Num_Layers; i++)  {
    n1 = (*Layerspecs_PP)[i].n;
    n2 = (*Layerspecs_PP)[i-1].n;
    (*Layerspecs_PP)[i].cos_crit0 = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;

    n2 = (*Layerspecs_PP)[i+1].n;
    (*Layerspecs_PP)[i].cos_crit1 = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;
    n2 = (*Layerspecs_PP)[Num_Layers+2].n;
    (*Layerspecs_PP)[i].cos_critxy = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;
  }
  n1 = (*Layerspecs_PP)[Num_Layers+2].n;   /* surrounding glass. */
  n2 = (*Layerspecs_PP)[0].n;
    (*Layerspecs_PP)[Num_Layers+2].cos_crit0 = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;
  n2 = (*Layerspecs_PP)[Num_Layers+1].n;
    (*Layerspecs_PP)[Num_Layers+2].cos_crit1 = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;
}

/*************************************************************
 *      Read in the input parameters for one run.
 ****/
void ReadParm(FILE* File_Ptr, InputStruct * In_Ptr)
{

  In_Ptr->Wth = WEIGHT;

  ReadFnameFormat(File_Ptr, In_Ptr);
  ReadNumPhotons(File_Ptr, In_Ptr);
  ReadDzDr(File_Ptr, In_Ptr);
  ReadNzNrNa(File_Ptr, In_Ptr);
  ReadXSizeYSize(File_Ptr, In_Ptr);
  ReadNumLayers(File_Ptr, In_Ptr);

  ReadLayerSpecs(File_Ptr, In_Ptr->num_layers,
                                &In_Ptr->layerspecs);
  CriticalAngle(In_Ptr->num_layers, &In_Ptr->layerspecs);
}

/*************************************************************
 *      Return 1, if the name in the name list.
 *      Return 0, otherwise.
 ****/
Boolean NameInList(char *Name, NameLink List)
{
  while (List != NULL) {
        if(strcmp(Name, List->name) == 0)
          return(1);
        List = List->next;
  };
  return(0);
}

/*************************************************************
 *      Add the name to the name list.
 ****/
void AddNameToList(char *Name, NameLink * List_Ptr)
{
  NameLink list = *List_Ptr;
```

```
  if(list == NULL) {   /* first node. */
        *List_Ptr = list = (NameLink)malloc(sizeof(NameNode));
        strcpy(list->name, Name);
        list->next = NULL;
  }
  else {                              /* subsequent nodes. */
        /* Move to the last node. */
        while(list->next != NULL)
          list = list->next;

        /* Append a node to the list. */
        list->next = (NameLink)malloc(sizeof(NameNode));
        list = list->next;
        strcpy(list->name, Name);
        list->next = NULL;
  }
}

/*************************************************************
 *      Check against duplicated file names.
 *
 *      A linked list is set up to store the file names used
 *      in this input data file.
 ****/
Boolean FnameTaken(char *fname, NameLink * List_Ptr)
{
  if(NameInList(fname, *List_Ptr))
        return(1);
  else {
        AddNameToList(fname, List_Ptr);
        return(0);
  }
}

/*************************************************************
 *      Free each node in the file name list.
 ****/
void FreeFnameList(NameLink List)
{
  NameLink next;

  while(List != NULL) {
        next = List->next;
        free(List);
        List = next;
  }
}

/*************************************************************
 *      Check the input parameters for each run.
 ****/
void CheckParm(FILE* File_Ptr, InputStruct * In_Ptr)
{
  short i_run;
  short num_runs;      /* number of independent runs. */
  NameLink head = NULL;
  Boolean name_taken;/* output files share the same */
                                        /* file name.*/
  char msg[STRLEN];

  num_runs = ReadNumRuns(File_Ptr);
  for(i_run=1; i_run<=num_runs; i_run++)  {
    printf("Checking input data for run %hd\n", i_run);
    ReadParm(File_Ptr, In_Ptr);

        name_taken = FnameTaken(In_Ptr->out_fname, &head);
        if(name_taken)
          sprintf(msg, "file name %s duplicated.\n",
                                        In_Ptr->out_fname);

    free(In_Ptr->layerspecs);
        if(name_taken) nrerror(msg);
  }
  FreeFnameList(head);
  rewind(File_Ptr);
}


/*************************************************************
 *      Allocate the arrays in OutStruct for one run, and
 *      array elements are automatically initialized to zeros.
 ****/
void InitOutputData(InputStruct In_Parm,
                                        OutStruct * Out_Ptr)
{
  short nz = In_Parm.nz;
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  short nl = In_Parm.num_layers;
  /* remember to use nl+2 because of 2 for ambient. */

  if(nz<=0 || nr<=0 || na<=0 || nl<=0)
    nrerror("Wrong grid parameters.\n");

  /* Init pure numbers. */
  Out_Ptr->Rsp = 0.0;
  Out_Ptr->Rd  = 0.0;
  Out_Ptr->A   = 0.0;
  Out_Ptr->Tt  = 0.0;
```

```
  /* Allocate the arrays and the matrices. */
  Out_Ptr->Rd_ra = AllocMatrix(0,nr-1,0,na-1);
  Out_Ptr->Rd_r  = AllocVector(0,nr-1);
  Out_Ptr->Rd_a  = AllocVector(0,na-1);

  Out_Ptr->A_rz  = AllocMatrix(0,nr-1,0,nz-1);
  Out_Ptr->A_z   = AllocVector(0,nz-1);
  Out_Ptr->A_l   = AllocVector(0,nl+1);

  Out_Ptr->Tt_ra = AllocMatrix(0,nr-1,0,na-1);
  Out_Ptr->Tt_r  = AllocVector(0,nr-1);
  Out_Ptr->Tt_a  = AllocVector(0,na-1);
}

/*************************************************************
 *      Undo what InitOutputData did.
 *  i.e. free the data allocations.
 ****/
void FreeData(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short nz = In_Parm.nz;
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  short nl = In_Parm.num_layers;
  /* remember to use nl+2 because of 2 for ambient. */

  free(In_Parm.layerspecs);

  FreeMatrix(Out_Ptr->Rd_ra, 0,nr-1,0,na-1);
  FreeVector(Out_Ptr->Rd_r, 0,nr-1);
  FreeVector(Out_Ptr->Rd_a, 0,na-1);

  FreeMatrix(Out_Ptr->A_rz, 0, nr-1, 0,nz-1);
  FreeVector(Out_Ptr->A_z, 0, nz-1);
  FreeVector(Out_Ptr->A_l, 0,nl+1);

  FreeMatrix(Out_Ptr->Tt_ra, 0,nr-1,0,na-1);
  FreeVector(Out_Ptr->Tt_r, 0,nr-1);
  FreeVector(Out_Ptr->Tt_a, 0,na-1);
}

/*************************************************************
 *      Get 1D array elements by summing the 2D array elements.
 ****/
void Sum2DRd(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  short ir,ia;
  double sum;

  for(ir=0; ir<nr; ir++)  {
    sum = 0.0;
    for(ia=0; ia<na; ia++) sum += Out_Ptr->Rd_ra[ir][ia];
    Out_Ptr->Rd_r[ir] = sum;
  }

  for(ia=0; ia<na; ia++)  {
    sum = 0.0;
    for(ir=0; ir<nr; ir++) sum += Out_Ptr->Rd_ra[ir][ia];
    Out_Ptr->Rd_a[ia] = sum;
  }

  sum = 0.0;
  for(ir=0; ir<nr; ir++) sum += Out_Ptr->Rd_r[ir];
  Out_Ptr->Rd = sum;
}

/*************************************************************
 *      Return the index to the layer according to the index
 *      to the grid line system in z direction (Iz).
 *
 *      Use the center of box.
 ****/
short IzToLayer(short Iz, InputStruct In_Parm)
{
  short i=1;   /* index to layer. */
  short num_layers = In_Parm.num_layers;
  double dz = In_Parm.dz;

  while( (Iz+0.5)*dz >= In_Parm.layerspecs[i].z1
       && i<num_layers) i++;

  return(i);
}

/*************************************************************
 *      Get 1D array elements by summing the 2D array elements.
 ****/
void Sum2DA(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short nz = In_Parm.nz;
  short nr = In_Parm.nr;
  short iz,ir;
  double sum;

  for(iz=0; iz<nz; iz++)  {
    sum = 0.0;
    for(ir=0; ir<nr; ir++) sum += Out_Ptr->A_rz[ir][iz];
```

```
    Out_Ptr->A_z[iz] = sum;
  }

  sum = 0.0;
  for(iz=0; iz<nz; iz++) {
    sum += Out_Ptr->A_z[iz];
    Out_Ptr->A_l[IzToLayer(iz, In_Parm)]
          += Out_Ptr->A_z[iz];
  }
  Out_Ptr->A = sum;
}

/*************************************************************
 *      Get 1D array elements by summing the 2D array elements.
 ****/
void Sum2DTt(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  short ir,ia;
  double sum;

  for(ir=0; ir<nr; ir++)  {
    sum = 0.0;
    for(ia=0; ia<na; ia++) sum += Out_Ptr->Tt_ra[ir][ia];
    Out_Ptr->Tt_r[ir] = sum;
  }

  for(ia=0; ia<na; ia++)  {
    sum = 0.0;
    for(ir=0; ir<nr; ir++) sum += Out_Ptr->Tt_ra[ir][ia];
    Out_Ptr->Tt_a[ia] = sum;
  }

  sum = 0.0;
  for(ir=0; ir<nr; ir++) sum += Out_Ptr->Tt_r[ir];
  Out_Ptr->Tt = sum;
}

/*************************************************************
 *      Scale Rd and Tt properly.
 *
 *      "a" stands for angle alpha.
 ****
 *      Scale Rd(r,a) and Tt(r,a) by
 *       (area perpendicular to photon direction)
 *              x(solid angle)x(No. of photons).
 *      or
 *              [2*PI*r*dr*cos(a)]x[2*PI*sin(a)*da]x[No. of photons]
 *      or
 *              [2*PI*PI*dr*da*r*sin(2a)]x[No. of photons]
 ****
 *      Scale Rd(r) and Tt(r) by
 *              (area on the surface)x(No. of photons).
 ****
 *      Scale Rd(a) and Tt(a) by
 *              (solid angle)x(No. of photons).
 ****/
void ScaleRdTt(InputStruct In_Parm, OutStruct *      Out_Ptr)
{
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  double dr = In_Parm.dr;
  double da = In_Parm.da;
  short ir,ia;
  double scale1, scale2;

  scale1 = 4.0*PI*PI*dr*sin(da/2)*In_Parm.num_photons;
       /* The factor (ir+0.5)*sin(2a) to be added. */

  for(ir=0; ir<nr; ir++)
    for(ia=0; ia<na; ia++) {
      scale2 = 1.0/((ir+0.5)*sin(2.0*(ia+0.5)*da)*scale1);
      Out_Ptr->Rd_ra[ir][ia] *= scale2;
      Out_Ptr->Tt_ra[ir][ia] *= scale2;
    }

  scale1 = 2.0*PI*dr*dr*In_Parm.num_photons;
       /* area is 2*PI*[(ir+0.5)*dr]*dr.*/
       /* ir+0.5 to be added. */

  for(ir=0; ir<nr; ir++) {
    scale2 = 1.0/((ir+0.5)*scale1);
    Out_Ptr->Rd_r[ir] *= scale2;
    Out_Ptr->Tt_r[ir] *= scale2;
  }

  scale1  = 2.0*PI*da*In_Parm.num_photons;
    /* solid angle is 2*PI*sin(a)*da. sin(a) to be added. */

  for(ia=0; ia<na; ia++) {
    scale2 = 1.0/(sin((ia+0.5)*da)*scale1);
    Out_Ptr->Rd_a[ia] *= scale2;
    Out_Ptr->Tt_a[ia] *= scale2;
  }

  scale2 = 1.0/(double)In_Parm.num_photons;
  Out_Ptr->Rd *= scale2;
  Out_Ptr->Tt *= scale2;
}
```

```
/************************************************************
 *      Scale absorption arrays properly.
 ****/
void ScaleA(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short nz = In_Parm.nz;
  short nr = In_Parm.nr;
  double dz = In_Parm.dz;
  double dr = In_Parm.dr;
  short nl = In_Parm.num_layers;
  short iz,ir;
  short il;
  double scale1;

  /* Scale A_rz. */
  scale1 = 2.0*PI*dr*dr*dz*In_Parm.num_photons;
       /* volume is 2*pi*(ir+0.5)*dr*dr*dz.*/
       /* ir+0.5 to be added. */
  for(iz=0; iz<nz; iz++)
    for(ir=0; ir<nr; ir++)
      Out_Ptr->A_rz[ir][iz] /= (ir+0.5)*scale1;

  /* Scale A_z. */
  scale1 = 1.0/(dz*In_Parm.num_photons);
  for(iz=0; iz<nz; iz++)
    Out_Ptr->A_z[iz] *= scale1;

  /* Scale A_l. Avoid int/int. */
  scale1 = 1.0/(double)In_Parm.num_photons;
  for(il=0; il<=nl+1; il++)
    Out_Ptr->A_l[il] *= scale1;

  Out_Ptr->A *=scale1;
}

/************************************************************
 *      Sum and scale results of current run.
 ****/
void SumScaleResult(InputStruct In_Parm,
                                    OutStruct * Out_Ptr)
{
  /* Get 1D & 0D results. */
  Sum2DRd(In_Parm, Out_Ptr);
  Sum2DA(In_Parm,  Out_Ptr);
  Sum2DTt(In_Parm, Out_Ptr);

  ScaleRdTt(In_Parm, Out_Ptr);
  ScaleA(In_Parm, Out_Ptr);
}
```

```
/************************************************************
 *      Write the version number as the first string in the
 *      file.
 *      Use chars only so that they can be read as either
 *      ASCII or binary.
 ****/
void WriteVersion(FILE *file, char *Version)
{
  fprintf(file,
       "%s \t# Version number of the file format.\n\n",
       Version);
  fprintf(file, "####\n# Data categories include: \n");
  fprintf(file, "# InParm, RAT, \n");
  fprintf(file, "# A_l, A_z, Rd_r, Rd_a, Tt_r, Tt_a, \n");
  fprintf(file, "# A_rz, Rd_ra, Tt_ra \n####\n\n");
}

/************************************************************
 *      Write the input parameters to the file.
 ****/
void WriteInParm(FILE *file, InputStruct In_Parm)
{
  short i;

  fprintf(file,
       "InParm \t\t\t# Input parameters. cm is used.\n");

  fprintf(file,
       "%s \tA\t\t# output file name, ASCII.\n",
       In_Parm.out_fname);
  fprintf(file,
       "%ld \t\t\t\t# No. of photons\n", In_Parm.num_photons);

  fprintf(file,
       "%G\t%G\t\t\t# dz, dr [cm]\n", In_Parm.dz,In_Parm.dr);
  fprintf(file, "%hd\t%hd\t%hd\t\t# No. of dz, dr, da.\n",
         In_Parm.nz, In_Parm.nr, In_Parm.na);
  fprintf(file, "%G\t%G\t\t\t# x_size and y_size.\n\n",
         In_Parm.x_size, In_Parm.y_size);

  fprintf(file,
       "%hd\t\t\t\t\t# Number of layers\n",
       In_Parm.num_layers);
  fprintf(file,
       "#n\tmua\tmus\tg\td\t# One line for each layer\n");
  fprintf(file,
       "%G\t\t\t\t\t# n for medium above\n",
       In_Parm.layerspecs[0].n);
  for(i=1; i<=In_Parm.num_layers; i++)  {
```

```
    LayerStruct s;
    s = In_Parm.layerspecs[i];
    fprintf(file, "%G\t%G\t%G\t%G\t%G\t# layer %hd\n",
            s.n, s.mua, s.mus, s.g, s.z1-s.z0, i);
  }
  fprintf(file, "%G\t\t\t\t\t# n for medium below\n",
        In_Parm.layerspecs[i].n);
  fprintf(file, "%G\t\t\t\t\t# n for surrounding glass\n\n",
        In_Parm.layerspecs[i+1].n);
}
/*************************************************************
 *      Write reflectance, absorption, transmission.
 ****/
void WriteRAT(FILE * file, OutStruct Out_Parm)
{
  fprintf(file,
        "RAT #Reflectance, absorption, transmission. \n");
        /* flag. */

  fprintf(file,
        "%-14.6G \t#Specular reflectance [-]\n", Out_Parm.Rsp);
  fprintf(file,
        "%-14.6G \t#Diffuse reflectance [-]\n", Out_Parm.Rd);
  fprintf(file,
        "%-14.6G \t#Absorbed fraction [-]\n", Out_Parm.A);
  fprintf(file,
        "%-14.6G \t#Transmittance [-]\n", Out_Parm.Tt);

  fprintf(file, "\n");
}

/*************************************************************
 *      Write absorption as a function of layer.
 ****/
void WriteA_layer(FILE * file,
                  short Num_Layers,
                  OutStruct Out_Parm)
{
  short i;

  fprintf(file,
        "A_l #Absorption as a function of layer. [-]\n");
        /* flag. */

  for(i=1; i<=Num_Layers; i++)
    fprintf(file, "%12.4G\n", Out_Parm.A_l[i]);
  fprintf(file, "\n");
}

/*************************************************************
 *      5 numbers each line.
 ****/
void WriteRd_ra(FILE * file,
                short Nr,
                short Na,
                OutStruct Out_Parm)
{
  short ir, ia;

  fprintf(file,
        "%s\n%s\n%s\n%s\n%s\n%s\n", /* flag. */
        "# Rd[r][angle]. [1/(cm2sr)].",
        "# Rd[0][0], [0][1],..[0][na-1]",
        "# Rd[1][0], [1][1],..[1][na-1]",
        "# ...",
        "# Rd[nr-1][0], [nr-1][1],..[nr-1][na-1]",
        "Rd_ra");

  for(ir=0;ir<Nr;ir++)
    for(ia=0;ia<Na;ia++)  {
      fprintf(file, "%12.4E ", Out_Parm.Rd_ra[ir][ia]);
      if( (ir*Na + ia + 1)%5 == 0) fprintf(file, "\n");
    }

  fprintf(file, "\n");
}

/*************************************************************
 *      1 number each line.
 ****/
void WriteRd_r(FILE * file,
               short Nr,
               OutStruct Out_Parm)
{
  short ir;

  fprintf(file,
        "Rd_r #Rd[0], [1],..Rd[nr-1]. [1/cm2]\n");   /* flag. */

  for(ir=0;ir<Nr;ir++) {
    fprintf(file, "%12.4E\n", Out_Parm.Rd_r[ir]);
  }

  fprintf(file, "\n");
}

/*************************************************************
 *      1 number each line.
 ****/
void WriteRd_a(FILE * file,
```

```
                          short Na,
                          OutStruct Out_Parm)
{
  short ia;

  fprintf(file,
       "Rd_a #Rd[0], [1],..Rd[na-1]. [sr-1]\n");    /* flag. */

  for(ia=0;ia<Na;ia++) {
    fprintf(file, "%12.4E\n", Out_Parm.Rd_a[ia]);
  }

  fprintf(file, "\n");
}

/************************************************************
 *      5 numbers each line.
 ****/
void WriteTt_ra(FILE * file,
                              short Nr,
                              short Na,
                              OutStruct Out_Parm)
{
  short ir, ia;

  fprintf(file,
         "%s\n%s\n%s\n%s\n%s\n%s\n", /* flag. */
         "# Tt[r][angle]. [1/(cm2sr)].",
         "# Tt[0][0], [0][1],..[0][na-1]",
         "# Tt[1][0], [1][1],..[1][na-1]",
         "# ...",
         "# Tt[nr-1][0], [nr-1][1],..[nr-1][na-1]",
         "Tt_ra");

  for(ir=0;ir<Nr;ir++)
    for(ia=0;ia<Na;ia++)  {
      fprintf(file, "%12.4E ", Out_Parm.Tt_ra[ir][ia]);
      if( (ir*Na + ia + 1)%5 == 0) fprintf(file, "\n");
    }

  fprintf(file, "\n");
}

/************************************************************
 *      5 numbers each line.
 ****/
void WriteA_rz(FILE * file,
                           short Nr,
                           short Nz,
                           OutStruct Out_Parm)
{
  short iz, ir;

  fprintf(file,
         "%s\n%s\n%s\n%s\n%s\n%s\n", /* flag. */
         "# A[r][z]. [1/cm3]",
         "# A[0][0], [0][1],..[0][nz-1]",
         "# A[1][0], [1][1],..[1][nz-1]",
         "# ...",
         "# A[nr-1][0], [nr-1][1],..[nr-1][nz-1]",
         "A_rz");

  for(ir=0;ir<Nr;ir++)
    for(iz=0;iz<Nz;iz++)  {
      fprintf(file, "%12.4E ", Out_Parm.A_rz[ir][iz]);
      if( (ir*Nz + iz + 1)%5 == 0) fprintf(file, "\n");
    }

  fprintf(file, "\n");
}

/************************************************************
 *      1 number each line.
 ****/
void WriteA_z(FILE * file,
                           short Nz,
                           OutStruct Out_Parm)
{
  short iz;

  fprintf(file,
       "A_z #A[0], [1],..A[nz-1]. [1/cm]\n");       /* flag. */

  for(iz=0;iz<Nz;iz++) {
    fprintf(file, "%12.4E\n", Out_Parm.A_z[iz]);
  }

  fprintf(file, "\n");
}

/************************************************************
 *      1 number each line.
 ****/
void WriteTt_r(FILE * file,
                           short Nr,
                           OutStruct Out_Parm)
{
  short ir;

  fprintf(file,
```

```
        "Tt_r #Tt[0], [1],..Tt[nr-1]. [1/cm2]\n"); /* flag. */

  for(ir=0;ir<Nr;ir++) {
    fprintf(file, "%12.4E\n", Out_Parm.Tt_r[ir]);
  }

  fprintf(file, "\n");
}

/*************************************************************
 *      1 number each line.
 ****/
void WriteTt_a(FILE * file,
                          short Na,
                          OutStruct Out_Parm)
{
  short ia;

  fprintf(file,
        "Tt_a #Tt[0], [1],..Tt[na-1]. [sr-1]\n"); /* flag. */

  for(ia=0;ia<Na;ia++) {
    fprintf(file, "%12.4E\n", Out_Parm.Tt_a[ia]);
  }

  fprintf(file, "\n");
}

/*************************************************************
 ****/
void WriteResult(InputStruct In_Parm,
                                OutStruct Out_Parm,
                                char * TimeReport)
{
  FILE *file;

  file = fopen(In_Parm.out_fname, "w");
  if(file == NULL) nrerror("Cannot open file to write.\n");

  if(toupper(In_Parm.out_fformat) == 'A')
        WriteVersion(file, "A1");
  else
        WriteVersion(file, "B1");

  fprintf(file, "# %s", TimeReport);
  fprintf(file, "\n");

  WriteInParm(file, In_Parm);
  WriteRAT(file, Out_Parm);
        /* reflectance, absorption, transmittance. */

  /* 1D arrays. */
  WriteA_layer(file, In_Parm.num_layers, Out_Parm);
  WriteA_z(file, In_Parm.nz, Out_Parm);
  WriteRd_r(file, In_Parm.nr, Out_Parm);
  WriteRd_a(file, In_Parm.na, Out_Parm);
  WriteTt_r(file, In_Parm.nr, Out_Parm);
  WriteTt_a(file, In_Parm.na, Out_Parm);

  /* 2D arrays. */
  WriteA_rz(file, In_Parm.nr, In_Parm.nz, Out_Parm);
  WriteRd_ra(file, In_Parm.nr, In_Parm.na, Out_Parm);
  WriteTt_ra(file, In_Parm.nr, In_Parm.na, Out_Parm);

  fclose(file);
}
```

B.1.1.4. mcmlgo.c

```
/*************************************************************
 *      Launch, move, and record photon weight.
 ****/

#include "mcml.h"

#define STANDARDTEST 0
  /* testing program using fixed rnd seed. */

#define PARTIALREFLECTION 0
  /* 1=split photon, 0=statistical reflection. */

#define COSZERO (1.0-1.0E-12)
  /* cosine of about 1e-6 rad. */

#define COS90D  1.0E-6
  /* cosine of about 1.57 - 1e-6 rad. */


/*************************************************************
 *      A random number generator from Numerical Recipes in C.
 ****/
```

```
#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC 1.0E-9

float ran3(int *idum)
{
  static int inext,inextp;
  static long ma[56];
  static int iff=0;
  long mj,mk;
  int i,ii,k;

  if (*idum < 0 || iff == 0) {
    iff=1;
    mj=MSEED-(*idum < 0 ? -*idum : *idum);
    mj %= MBIG;
    ma[55]=mj;
    mk=1;
    for (i=1;i<=54;i++) {
      ii=(21*i) % 55;
      ma[ii]=mk;
      mk=mj-mk;
      if (mk < MZ) mk += MBIG;
      mj=ma[ii];
    }
    for (k=1;k<=4;k++)
      for (i=1;i<=55;i++) {
        ma[i] -= ma[1+(i+30) % 55];
        if (ma[i] < MZ) ma[i] += MBIG;
      }
    inext=0;
    inextp=31;
    *idum=1;
  }
  if (++inext == 56) inext=1;
  if (++inextp == 56) inextp=1;
  mj=ma[inext]-ma[inextp];
  if (mj < MZ) mj += MBIG;
  ma[inext]=mj;
  return mj*FAC;
}

#undef MBIG
#undef MSEED
#undef MZ
#undef FAC


/*************************************************************
 *      Generate a random number between 0 and 1.  Take a
 *      number as seed the first time entering the function.
 *      The seed is limited to 1<<15.
 *      We found that when idum is too large, ran3 may return
 *      numbers beyond 0 and 1.
 ****/
double RandomNum(void)
{
  static Boolean first_time=1;
  static int idum;     /* seed for ran3. */

  if(first_time) {
#if STANDARDTEST /* Use fixed seed to test the program. */
    idum = - 1;
#else
    idum = -(int)time(NULL)%(1<<15);
        /* use 16-bit integer as the seed. */
#endif
    ran3(&idum);
    first_time = 0;
    idum = 1;
  }

  return( (double)ran3(&idum) );
}

/*************************************************************
 *      Compute the specular reflection.
 *
 *      If the first layer is a turbid medium, use the Fresnel
 *      reflection from the boundary of the first layer as the
 *      specular reflectance.
 *
 *      If the first layer is glass, multiple reflections in
 *      the first layer is considered to get the specular
 *      reflectance.
 *
 *      The subroutine assumes the Layerspecs array is correctly
 *      initialized.
 ****/
double Rspecular(LayerStruct * Layerspecs_Ptr)
{
  double r1, r2;
        /* direct reflections from the 1st and 2nd layers. */
  double temp;

  temp =(Layerspecs_Ptr[0].n - Layerspecs_Ptr[1].n)
          /(Layerspecs_Ptr[0].n + Layerspecs_Ptr[1].n);
  r1 = temp*temp;
```

```
  if((Layerspecs_Ptr[1].mua == 0.0)
  && (Layerspecs_Ptr[1].mus == 0.0))  { /* glass layer. */
    temp = (Layerspecs_Ptr[1].n - Layerspecs_Ptr[2].n)
                /(Layerspecs_Ptr[1].n + Layerspecs_Ptr[2].n);
    r2 = temp*temp;
    r1 = r1 + (1-r1)*(1-r1)*r2/(1-r1*r2);
  }

  return (r1);
}

/*****************************************************
 *      Initialize a photon packet.
 ****/
void LaunchPhoton(double Rspecular,
                              LayerStruct  * Layerspecs_Ptr,
                              PhotonStruct * Photon_Ptr)
{
  Photon_Ptr->w = 1.0 - Rspecular;
  Photon_Ptr->dead = 0;
  Photon_Ptr->layer = 1;
  Photon_Ptr->s = 0;
  Photon_Ptr->sleft= 0;

  Photon_Ptr->x       = 0.0;
  Photon_Ptr->y       = 0.0;
  Photon_Ptr->z       = 0.0;
  Photon_Ptr->ux      = 0.0;
  Photon_Ptr->uy      = 0.0;
  Photon_Ptr->uz      = 1.0;

  if((Layerspecs_Ptr[1].mua == 0.0)
  && (Layerspecs_Ptr[1].mus == 0.0))  { /* glass layer. */
    Photon_Ptr->layer  = 2;
    Photon_Ptr->z      = Layerspecs_Ptr[2].z0;
  }
}

/*****************************************************
 *      Choose (sample) a new theta angle for photon propagation
 *      according to the anisotropy.
 *
 *      If anisotropy g is 0, then
 *              cos(theta) = 2*rand-1.
 *      otherwise
 *              sample according to the Henyey-Greenstein function.
 *
 *      Returns the cosine of the polar deflection angle theta.
 ****/
double SpinTheta(double g)
{
  double cost;

  if(g == 0.0)
    cost = 2*RandomNum() -1;
  else {
    double temp = (1-g*g)/(1-g+2*g*RandomNum());
    cost = (1+g*g - temp*temp)/(2*g);
  }
  return(cost);
}


/*****************************************************
 *      Choose a new direction for photon propagation by
 *      sampling the polar deflection angle theta and the
 *      azimuthal angle psi.
 *
 *      Note:
 *      theta: 0 - pi so sin(theta) is always positive
 *      feel free to use sqrt() for cos(theta).
 *
 *      psi:   0 - 2pi
 *      for 0-pi  sin(psi) is +
 *      for pi-2pi sin(psi) is -
 ****/
void Spin(double g,
                PhotonStruct * Photon_Ptr)
{
  double cost, sint;   /* cosine and sine of the */
                                                  /* polar deflection angle theta.
*/
  double cosp, sinp;   /* cosine and sine of the */
                                                  /* azimuthal angle psi. */
  double ux = Photon_Ptr->ux;
  double uy = Photon_Ptr->uy;
  double uz = Photon_Ptr->uz;
  double psi;

  cost = SpinTheta(g);
  sint = sqrt(1.0 - cost*cost);
       /* sqrt() is faster than sin(). */

  psi = 2.0*PI*RandomNum(); /* spin psi 0-2pi. */
  cosp = cos(psi);
  if(psi<PI)
    sinp = sqrt(1.0 - cosp*cosp);
         /* sqrt() is faster than sin(). */
  else
    sinp = - sqrt(1.0 - cosp*cosp);
```

```
  if(fabs(uz) > COSZERO)  {   /* normal incident. */
    Photon_Ptr->ux = sint*cosp;
    Photon_Ptr->uy = sint*sinp;
    Photon_Ptr->uz = cost*SIGN(uz);
         /* SIGN() is faster than division. */
  }
  else  {              /* regular incident. */
    double temp = sqrt(1.0 - uz*uz);
    Photon_Ptr->ux = sint*(ux*uz*cosp - uy*sinp)
                                    /temp + ux*cost;
    Photon_Ptr->uy = sint*(uy*uz*cosp + ux*sinp)
                                    /temp + uy*cost;
    Photon_Ptr->uz = -sint*cosp*temp + uz*cost;
  }
}

/*************************************************************
 *      Move the photon s away in the current layer of medium.
 ****/
void Hop(PhotonStruct *       Photon_Ptr)
{
  double s = Photon_Ptr->s;

  Photon_Ptr->x += s*Photon_Ptr->ux;
  Photon_Ptr->y += s*Photon_Ptr->uy;
  Photon_Ptr->z += s*Photon_Ptr->uz;
}

/*************************************************************
 * Return the photon step size in glass.
 *
 * The step size is the distance between the current
 * position and the nearest boundary in the photon direction.
 *
 * uz != 0
 *
 ****/
short StepSizeInGlass(PhotonStruct *  Photon_Ptr,
                                      InputStruct  *  In_Ptr)
{
  double dl_b;       /* distance to nearest boundary. */
  double dl_xbound;  /* distance to boundary perp. to x-direction. */
  double dl_ybound;  /* distance to boundary perp. to y-direction. */
  double dl_zbound;  /* distance to boundary perp. to z-direction. */
  short  layer = Photon_Ptr->layer;
  double ux = Photon_Ptr->ux;
  double uy = Photon_Ptr->uy;
  double uz = Photon_Ptr->uz;
  short hitcase;     /* the boundary which is hit.  */

  /* hitcase = 1,2 or 3 for boundary perp. to x-, y- or z-direction. */

  if(uz>0.0)
    dl_zbound = (In_Ptr->layerspecs[layer].z1
                          - Photon_Ptr->z)/uz;  /* dl_zbound>0. */
  else if(uz<0.0)
    dl_zbound = (In_Ptr->layerspecs[layer].z0
                          - Photon_Ptr->z)/uz;  /* dl_zbound>0. */
  dl_b = dl_zbound;
  hitcase = 3;
  if(ux != 0.0) {
    dl_xbound = ((In_Ptr->x_size)*SIGN(ux)/2 - Photon_Ptr->x)/ux;
    if (dl_xbound<dl_b) {
      dl_b = dl_xbound;
      hitcase = 1;
    }
  }
  if(uy != 0.0) {
    dl_ybound = ((In_Ptr->y_size)*SIGN(uy)/2 - Photon_Ptr->y)/uy;
    if (dl_ybound<dl_b) {
      dl_b = dl_ybound;
      hitcase = 2;
    }
  }
  Photon_Ptr->s = dl_b;

  return(hitcase);
}

/*************************************************************
 *  Return the photon step size in surrouning glass,
 *
 * The step size is the distance between the current
 * position and the boundary, perp. to z-direction,
 * in the photon direction.
 *
 * uz != 0
 ****/
void StepSizeInSurrGlass(PhotonStruct *  Photon_Ptr,
                                         InputStruct  *  In_Ptr)
{
  double dl_b;       /* distance to boundary. */
  short  layer = Photon_Ptr->layer;
  double uz = Photon_Ptr->uz;

  if(uz>0.0)
    dl_b = (In_Ptr->layerspecs[layer].z1
                          - Photon_Ptr->z)/uz;  /* dl_b>0. */
  else if(uz<0.0)
    dl_b = (In_Ptr->layerspecs[layer].z0
```

```
                    - Photon_Ptr->z)/uz;   /* dl_b>0. */

  Photon_Ptr->s = dl_b;
}

/*************************************************************
 *      Pick a step size for a photon packet when it is in
 *      tissue.
 *      If the member sleft is zero, make a new step size
 *      with: -log(rnd)/(mua+mus).
 *      Otherwise, pick up the leftover in sleft.
 *
 *      Layer is the index to layer.
 *      In_Ptr is the input parameters.
 ****/
void StepSizeInTissue(PhotonStruct * Photon_Ptr,
                                    InputStruct  * In_Ptr)
{
  short  layer = Photon_Ptr->layer;
  double mua = In_Ptr->layerspecs[layer].mua;
  double mus = In_Ptr->layerspecs[layer].mus;

  if(Photon_Ptr->sleft == 0.0) {  /* make a new step. */
    double rnd;

    do rnd = RandomNum();
      while( rnd <= 0.0 );    /* avoid zero. */
      Photon_Ptr->s = -log(rnd)/(mua+mus);
  }
  else {       /* take the leftover. */
       Photon_Ptr->s = Photon_Ptr->sleft/(mua+mus);
       Photon_Ptr->sleft = 0.0;
  }
}

/*************************************************************
 * Check if the step will hit a boundary.
 * Return 1, 2 or 3 if a boundary is hit.
 * Return 0 otherwise.
 *
 * If the projected step hits the boundary, the members
 * s and sleft of Photon_Ptr are updated.
 ****/
short HitBoundary(PhotonStruct *  Photon_Ptr,
                                   InputStruct  *  In_Ptr)
{
  double dl_b;       /* distance to nearest boundary. */
  double dl_xbound;  /* distance to boundary perp. to x-direction. */
  double dl_ybound;  /* distance to boundary perp. to y-direction. */
  double dl_zbound;  /* distance to boundary perp. to z-direction. */
```



```
short  layer = Photon_Ptr->layer;
double ux = Photon_Ptr->ux;
double uy = Photon_Ptr->uy;
double uz = Photon_Ptr->uz;
short hitcase;      /* the boundary which is hit. */
 /* hitcase = 1,2 or 3 for boundary perp. to x-, y- or z-direction. */
 /* hitcase = 0 if no boundary is hit */
double maxdl_xybound = sqrt(pow(In_Ptr->x_size,2.0)
                      + pow(In_Ptr->y_size,2.0)
                      + pow(In_Ptr->layerspecs[layer].z1
                            - In_Ptr->layerspecs[layer].z0,2.0));
                      /* The longest possible path in this layer. */

if(uz>0.0)
  dl_zbound = (In_Ptr->layerspecs[layer].z1
                      - Photon_Ptr->z)/uz;   /* dl_zbound>0. */
else if(uz<0.0)
  dl_zbound = (In_Ptr->layerspecs[layer].z0
                      - Photon_Ptr->z)/uz;   /* dl_zbound>0. */
if (uz == 0.0)
  dl_b = maxdl_xybound;
else  {
  dl_b = dl_zbound;
  hitcase = 3;
}
if(ux != 0.0) {
  dl_xbound = ((In_Ptr->x_size)*SIGN(ux)/2 - Photon_Ptr->x)/ux;
  if (dl_xbound<dl_b) {
    dl_b = dl_xbound;
    hitcase = 1;
  }
}
if(uy != 0.0) {
  dl_ybound = ((In_Ptr->y_size)*SIGN(uy)/2 - Photon_Ptr->y)/uy;
  if (dl_ybound<dl_b) {
    dl_b = dl_ybound;
    hitcase = 2;
  }
}
if(Photon_Ptr->s > dl_b) {
       /* crossing. */
  double mut = In_Ptr->layerspecs[layer].mua
                              + In_Ptr->layerspecs[layer].mus;

  Photon_Ptr->sleft = (Photon_Ptr->s - dl_b)*mut;
  Photon_Ptr->s     = dl_b;
}
else
  hitcase = 0;
```

```
  return(hitcase);
}

/*************************************************************
 * Drop photon weight inside the tissue (not glass).
 *
 * The photon is assumed not dead.
 *
 * The weight drop is dw = w*mua/(mua+mus).
 *
 * The dropped weight is assigned to the absorption array
 * elements.
 ****/
void Drop(InputStruct  *      In_Ptr,
                PhotonStruct *       Photon_Ptr,
                OutStruct *          Out_Ptr)
{
  double dwa;          /* absorbed weight.*/
  double x = Photon_Ptr->x;
  double y = Photon_Ptr->y;
  short  iz, ir;       /* index to z & r. */
  short  layer = Photon_Ptr->layer;
  double mua, mus;

  /* compute array indices. */
  iz = (short)(Photon_Ptr->z/In_Ptr->dz);
  if(iz>In_Ptr->nz-1) iz=In_Ptr->nz-1;

  ir = (short)(sqrt(x*x+y*y)/In_Ptr->dr);
  if(ir>In_Ptr->nr-1) ir=In_Ptr->nr-1;

  /* update photon weight. */
  mua = In_Ptr->layerspecs[layer].mua;
  mus = In_Ptr->layerspecs[layer].mus;
  dwa = Photon_Ptr->w * mua/(mua+mus);
  Photon_Ptr->w -= dwa;

  /* assign dwa to the absorption array element. */
  Out_Ptr->A_rz[ir][iz]       += dwa;
}

/*************************************************************
 *      The photon weight is small, and the photon packet tries
 *      to survive a roulette.
 ****/
void Roulette(PhotonStruct * Photon_Ptr)
{
  if(Photon_Ptr->w == 0.0)
    Photon_Ptr->dead = 1;
  else if(RandomNum() < CHANCE) /* survived the roulette.*/
    Photon_Ptr->w /= CHANCE;
  else
    Photon_Ptr->dead = 1;
}

/*************************************************************
 *      Compute the Fresnel reflectance.
 *
 *      Make sure that the cosine of the incident angle a1
 *      is positive, and the case when the angle is greater
 *      than the critical angle is ruled out.
 *
 *      Avoid trigonometric function operations as much as
 *      possible, because they are computation-intensive.
 ****/
double RFresnel(double n1,     /* incident refractive index.*/
                                double n2,      /* transmit refractive index.*/
                                double ca1,     /* cosine of the incident */
                                                /*      angle.      0<a1<90
degrees. */
                                double * ca2_Ptr)  /* pointer to the */
                                                /* cosine of the transmission */
                                                /* angle. a2>0. */
{
  double r;

  if(n1==n2) {                          /** matched boundary. **/
    *ca2_Ptr = ca1;
    r = 0.0;
  }
  else if(ca1>COSZERO) {        /** normal incident. **/
    *ca2_Ptr = ca1;
    r = (n2-n1)/(n2+n1);
    r *= r;
  }
  else if(ca1<COS90D)  {        /** very slant. **/
    *ca2_Ptr = 0.0;
    r = 1.0;
  }
  else  {                                       /** general. **/
    double sa1, sa2;
          /* sine of the incident and transmission angles. */
    double ca2;

    sa1 = sqrt(1-ca1*ca1);
    sa2 = n1*sa1/n2;
    if(sa2>=1.0) {
          /* double check for total internal reflection. */
      *ca2_Ptr = 0.0;
      r = 1.0;
```

```
    }
    else  {
      double cap, cam; /* cosines of the sum ap or */
                                          /* difference am of the two */
                                          /* angles. ap = a1+a2 */
                                          /* am = a1 - a2. */
      double sap, sam; /* sines. */

      *ca2_Ptr = ca2 = sqrt(1-sa2*sa2);

      cap = ca1*ca2 - sa1*sa2; /* c+ = cc - ss. */
      cam = ca1*ca2 + sa1*sa2; /* c- = cc + ss. */
      sap = sa1*ca2 + ca1*sa2; /* s+ = sc + cs. */
      sam = sa1*ca2 - ca1*sa2; /* s- = sc - cs. */
      r = 0.5*sam*sam*(cam*cam+cap*cap)/(sap*sap*cam*cam);
               /* rearranged for speed. */
    }
  }
  return(r);
}

/*************************************************************
 *      Record the photon weight exiting the first layer(uz<0),
 *      no matter whether the layer is glass or not, to the
 *      reflection array.
 *
 *      Update the photon weight as well.
 ****/
void RecordR(double                  Refl,   /* reflectance. */
                       InputStruct  *       In_Ptr,
                       PhotonStruct *       Photon_Ptr,
                       OutStruct *   Out_Ptr)
{
  double x = Photon_Ptr->x;
  double y = Photon_Ptr->y;
  short  ir, ia;       /* index to r & angle. */

  ir = (short)(sqrt(x*x+y*y)/In_Ptr->dr);
  if(ir>In_Ptr->nr-1) ir=In_Ptr->nr-1;

  ia = (short)(acos(-Photon_Ptr->uz)/In_Ptr->da);
  if(ia>In_Ptr->na-1) ia=In_Ptr->na-1;

  /* assign photon to the reflection array element. */
  Out_Ptr->Rd_ra[ir][ia] += Photon_Ptr->w*(1.0-Refl);

  Photon_Ptr->w *= Refl;
}

/*************************************************************
 *      Record the photon weight exiting the last layer(uz>0),
 *      no matter whether the layer is glass or not, to the
 *      transmittance array.
 *
 *      Update the photon weight as well.
 ****/
void RecordT(double          Refl,
                       InputStruct  *       In_Ptr,
                       PhotonStruct *       Photon_Ptr,
                       OutStruct *   Out_Ptr)
{
  double x = Photon_Ptr->x;
  double y = Photon_Ptr->y;
  short  ir, ia;       /* index to r & angle. */

  ir = (short)(sqrt(x*x+y*y)/In_Ptr->dr);
  if(ir>In_Ptr->nr-1) ir=In_Ptr->nr-1;

  ia = (short)(acos(Photon_Ptr->uz)/In_Ptr->da);
  if(ia>In_Ptr->na-1) ia=In_Ptr->na-1;

  /* assign photon to the transmittance array element. */
  Out_Ptr->Tt_ra[ir][ia] += Photon_Ptr->w*(1.0-Refl);

  Photon_Ptr->w *= Refl;
}

/*************************************************************
 * Decide whether the photon will be transmitted or
 * reflected on the upper boundary (uz<0) of the current
 * layer.
 *
 * If "layer" is the first or surrounding layer,
 * the photon packet will be partially transmitted
 * and partially reflected if
 * PARTIALREFLECTION is set to 1,
 * or the photon packet will be either transmitted or
 * reflected determined statistically if PARTIALREFLECTION
 * is set to 0.
 *
 * Record the transmitted photon weight as reflection.
 *
 * If "layer" is not the first or surrounding
 * layer and the photon packet is transmitted,
 * move the photon to "layer-1".
 *
 * Update the photon parmameters.
 ****/
void CrossUpOrNot(InputStruct  *      In_Ptr,
                                PhotonStruct *       Photon_Ptr,
```

```
                                OutStruct *          Out_Ptr)
{
  double uz = Photon_Ptr->uz; /* z directional cosine. */
  double uz1;  /* cosines of transmission alpha, always positive. */
  double r=0.0;        /* reflectance. */
  short  layer = Photon_Ptr->layer;
  double ni = In_Ptr->layerspecs[layer].n;
  double nt = In_Ptr->layerspecs[layer-1].n;

  /* Get r. */
  if( - uz <= In_Ptr->layerspecs[layer].cos_crit0)
    r=1.0;                  /* total internal reflection. */
  else if (layer == In_Ptr->num_layers + 2)    /* surr. glass. */
    r = RFresnel(In_Ptr->layerspecs[layer].n,
                 In_Ptr->layerspecs[0].n, -uz, &uz1);
  else
    r = RFresnel(In_Ptr->layerspecs[layer].n,
                 In_Ptr->layerspecs[layer-1].n, -uz, &uz1);

  if (r == 1.0 && layer == In_Ptr->num_layers + 2)
    Photon_Ptr->dead = 1;      /* total internal reflection. */
                           /* in surrounding glass. */

#if PARTIALREFLECTION
  if((layer == 1 || layer == In_Ptr->num_layers + 2)
      && r<1.0) {    /* partially transmitted. */
    Photon_Ptr->uz = -uz1;    /* transmitted photon. */
    RecordR(r, In_Ptr, Photon_Ptr, Out_Ptr);
    Photon_Ptr->uz = -uz;     /* reflected photon. */
  }
  else if(RandomNum() > r) {/* transmitted.  */
    Photon_Ptr->layer--;
    Photon_Ptr->ux *= ni/nt;
    Photon_Ptr->uy *= ni/nt;
    Photon_Ptr->uz = -uz1;
  }
  else                                 /* reflected. */
    Photon_Ptr->uz = -uz;
#else
  if(RandomNum() > r) {               /* transmitted.  */
    if(layer == 1 || layer == In_Ptr->num_layers + 2)  {
      Photon_Ptr->uz = -uz1;
      RecordR(0.0, In_Ptr, Photon_Ptr, Out_Ptr);
      Photon_Ptr->dead = 1;
    }
    else {
      Photon_Ptr->layer--;
      Photon_Ptr->ux *= ni/nt;
      Photon_Ptr->uy *= ni/nt;
      Photon_Ptr->uz = -uz1;
    }
  }
  else                                          /* reflected. */
    Photon_Ptr->uz = -uz;
#endif

 if (r == 1.0 && layer == In_Ptr->num_layers + 2)
    Photon_Ptr->dead = 1;
    /* total internal reflection in surrounding glass. */
}

/*************************************************************
 * Decide whether the photon will be transmitted  or be
 * reflected on the bottom boundary (uz>0) of the current
 * layer.
 *
 * If the photon is transmitted, move the photon to
 * "layer+1". If "layer" is the last or surrounding layer,
 * record the transmitted weight as transmittance.
 * See comments for CrossUpOrNot.
 *
 * Update the photon parmameters.
 ****/
void CrossDnOrNot(InputStruct  *      In_Ptr,
                                PhotonStruct *       Photon_Ptr,
                                OutStruct *          Out_Ptr)
{
  double uz = Photon_Ptr->uz; /* z directional cosine. */
  double uz1;  /* cosines of transmission alpha. */
  double r=0.0;        /* reflectance. */
  short  layer = Photon_Ptr->layer;
  double ni = In_Ptr->layerspecs[layer].n;
  double nt = In_Ptr->layerspecs[layer+1].n;

  /* Get r. */
  if( uz <= In_Ptr->layerspecs[layer].cos_crit1)    r=1.0;          /*    total
internal reflection. */
  else if (layer == In_Ptr->num_layers + 2)    /* surr. glass. */
    r = RFresnel(In_Ptr->layerspecs[layer].n,
                 In_Ptr->layerspecs[layer-1].n, uz, &uz1);
  else r = RFresnel(In_Ptr->layerspecs[layer].n,
                 In_Ptr->layerspecs[layer+1].n, uz, &uz1);

#if PARTIALREFLECTION
  if((layer == In_Ptr->num_layers || layer == In_Ptr->num_layers + 2)
     && r<1.0) {
    Photon_Ptr->uz = uz1;
    RecordT(r, In_Ptr, Photon_Ptr, Out_Ptr);
    Photon_Ptr->uz = -uz;
  }
```

```
  else if(RandomNum() > r) {/* transmitted. */
    Photon_Ptr->layer++;
    Photon_Ptr->ux *= ni/nt;
    Photon_Ptr->uy *= ni/nt;
    Photon_Ptr->uz = uz1;
  }
  else                                    /* reflected. */
    Photon_Ptr->uz = -uz;
#else
  if(RandomNum() > r) {               /* transmitted. */
    if(layer == In_Ptr->num_layers ||
       layer == In_Ptr->num_layers + 2) {
      Photon_Ptr->uz = uz1;
      RecordT(0.0, In_Ptr, Photon_Ptr, Out_Ptr);
      Photon_Ptr->dead = 1;
    }
    else {
      Photon_Ptr->layer++;
      Photon_Ptr->ux *= ni/nt;
      Photon_Ptr->uy *= ni/nt;
      Photon_Ptr->uz = uz1;
    }
  }
  else                                    /* reflected. */
    Photon_Ptr->uz = -uz;
#endif
}

/*************************************************************
 * Decide whether the photon will be transmitted or be
 * reflected on the boundary perpendicular
 * to the x-direction
 *
 * If the photon is transmitted, move the photon to the
 * surrounded glass layer.
 *
 * Update the photon parmameters.
 ****/
void CrossXOrNot(InputStruct  *       In_Ptr,
                                 PhotonStruct *       Photon_Ptr,
                                 OutStruct *          Out_Ptr)
{
  double ux = Photon_Ptr->ux; /* x directional cosine. */
  double ux1;  /* cosines of transmission alpha. */
  double r=0.0;       /* reflectance. */
  short  layer = Photon_Ptr->layer;
  double ni = In_Ptr->layerspecs[layer].n;
  double nt = In_Ptr->layerspecs[In_Ptr->num_layers + 2].n;

  /* Get r. */
  if( fabs(ux) <= In_Ptr->layerspecs[layer].cos_critxy)
    r=1.0;            /* total internal reflection. */
  else r = RFresnel(In_Ptr->layerspecs[layer].n,
        In_Ptr->layerspecs[In_Ptr->num_layers+2].n, fabs(ux), &ux1);


  if(RandomNum() > r) {              /* transmitted to surrounding glass. */
    Photon_Ptr->layer = In_Ptr->num_layers+2;
    Photon_Ptr->ux = ux1*SIGN(ux);
    Photon_Ptr->uy *= ni/nt;
    Photon_Ptr->uz *= ni/nt;
    }
  else                                          /* reflected. */
    Photon_Ptr->ux = -ux;
}

/*************************************************************
 * Decide whether the photon will be transmitted or be
 * reflected on the boundary perpendicular
 * to the y-direction
 *
 * If the photon is transmitted, move the photon to the
 * surrounded glass layer.
 *
 * Update the photon parmameters.
 ****/
void CrossYOrNot(InputStruct  *       In_Ptr,
                                 PhotonStruct *       Photon_Ptr,
                                 OutStruct *          Out_Ptr)
{
  double uy = Photon_Ptr->uy; /* x directional cosine. */
  double uy1;  /* cosines of transmission alpha. */
  double r=0.0;       /* reflectance. */
  short  layer = Photon_Ptr->layer;
  double ni = In_Ptr->layerspecs[layer].n;
  double nt = In_Ptr->layerspecs[In_Ptr->num_layers + 2].n;

  /* Get r. */
  if( fabs(uy) <= In_Ptr->layerspecs[layer].cos_critxy)
    r=1.0;            /* total internal reflection. */
  else r = RFresnel(In_Ptr->layerspecs[layer].n,
        In_Ptr->layerspecs[In_Ptr->num_layers+2].n, fabs(uy), &uy1);


  if(RandomNum() > r) {              /* transmitted to surrounding glass. */
    Photon_Ptr->layer = In_Ptr->num_layers+2;
    Photon_Ptr->uy = uy1*SIGN(uy);
    Photon_Ptr->ux *= ni/nt;
    Photon_Ptr->uz *= ni/nt;
    }
```

```
  else                                          /* reflected. */
    Photon_Ptr->uy = -uy;
}
/*************************************************************
 ****/
void CrossOrNot(InputStruct  *       In_Ptr,
                             PhotonStruct * Photon_Ptr,
                             OutStruct    * Out_Ptr,
                             short          hitcase)
{
  if (hitcase == 1)
    CrossXOrNot(In_Ptr, Photon_Ptr, Out_Ptr);
  else if (hitcase == 2)
    CrossYOrNot(In_Ptr, Photon_Ptr, Out_Ptr);
  else if (hitcase == 3) {
    if(Photon_Ptr->uz < 0.0)
      CrossUpOrNot(In_Ptr, Photon_Ptr, Out_Ptr);
    else
      CrossDnOrNot(In_Ptr, Photon_Ptr, Out_Ptr);
  }
}

/*************************************************************
 *      Move the photon packet in glass layer.
 *      Horizontal photons are killed because they will
 *      never interact with tissue again.
 ****/
void HopInGlass(InputStruct  * In_Ptr,
                             PhotonStruct * Photon_Ptr,
                             OutStruct    * Out_Ptr)
{
  short  hitcase; /* the boundary which is hit. */

  if(Photon_Ptr->uz == 0.0)
    /* horizontal photon in glass is killed. */
    Photon_Ptr->dead = 1;
  else {
    if(Photon_Ptr->layer == In_Ptr->num_layers + 2) {
          /* surrounding glass */
      StepSizeInSurrGlass(Photon_Ptr, In_Ptr);
      hitcase = 3;        /* always in surrounding glass */
      Hop(Photon_Ptr);
      CrossOrNot(In_Ptr, Photon_Ptr, Out_Ptr, hitcase);
    }
    else {  /* glass layer which is not the surrounding glass layer */
      hitcase = StepSizeInGlass(Photon_Ptr, In_Ptr);
      Hop(Photon_Ptr);
      CrossOrNot(In_Ptr, Photon_Ptr, Out_Ptr, hitcase);
    }
  }
}

/*************************************************************
 *      Set a step size, move the photon, drop some weight,
 *      choose a new photon direction for propagation.
 *
 *      When a step size is long enough for the photon to
 *      hit an interface, this step is divided into two steps.
 *      First, move the photon to the boundary free of
 *      absorption or scattering, then decide whether the
 *      photon is reflected or transmitted.
 *      Then move the photon in the current or transmission
 *      medium with the unfinished stepsize to interaction
 *      site. If the unfinished stepsize is still too long,
 *      repeat the above process.
 ****/
void HopDropSpinInTissue(InputStruct  *  In_Ptr,
                                         PhotonStruct *  Photon_Ptr,
                                         OutStruct    *  Out_Ptr)
{
  short hitcase; /* the boundary which is hit. */

                 /* hitcase = 0 if no boundary is hit */

  StepSizeInTissue(Photon_Ptr, In_Ptr);

  hitcase = HitBoundary(Photon_Ptr, In_Ptr);

  if(hitcase>0) {
    Hop(Photon_Ptr);   /* move to boundary plane. */
    CrossOrNot(In_Ptr, Photon_Ptr, Out_Ptr, hitcase);
  }
  else {
    Hop(Photon_Ptr);
    Drop(In_Ptr, Photon_Ptr, Out_Ptr);
    Spin(In_Ptr->layerspecs[Photon_Ptr->layer].g,
              Photon_Ptr);
  }
}

/*************************************************************
 ****/
void HopDropSpin(InputStruct  *  In_Ptr,
                                 PhotonStruct *  Photon_Ptr,
                                 OutStruct    *  Out_Ptr)
{
  short layer = Photon_Ptr->layer;

  if((In_Ptr->layerspecs[layer].mua == 0.0)
  && (In_Ptr->layerspecs[layer].mus == 0.0))
```

```
      /* glass layer. */
    HopInGlass(In_Ptr, Photon_Ptr, Out_Ptr);
  else
    HopDropSpinInTissue(In_Ptr, Photon_Ptr, Out_Ptr);

  if( Photon_Ptr->w < In_Ptr->Wth && !Photon_Ptr->dead)
    Roulette(Photon_Ptr);
}
```

B.1.2. TMMCML

B.1.2.1. mcml.h

```
/****************************************************
 *      Monte Carlo simulation of photon distribution in
 *      multi-layered media containing scattering particles
 *      in ANSI Standard C.
 ****
 *      Starting Date: 10/1991.
 *      Current Date:  8/1996.
 *
 *      Lihong Wang, Ph. D.
 *      Steven L. Jacques, Ph. D.
 *      Laser Biology Research Laboratory - 17
 *      M.D. Anderson Cancer Center
 *      University of Texas
 *      1515 Holcombe Blvd.
 *      Houston, TX 77030
 *      USA
 *
 *      This program was based on:
 *      (1) The Pascal code written by Marleen Keijzer and
 *      Steven L. Jacques in this laboratory in 1989, which
 *      deals with multi-layered turbid media.
 *
 *      (2) Algorithm for semi-infinite turbid medium by
 *      S.A. Prahl, M. Keijzer, S.L. Jacques, A.J. Welch,
 *      SPIE Institute Series Vol. IS 5 (1989), and by
 *      A.N. Witt, The Astrophysical journal Supplement
 *      Series 35, 1-6 (1977).
 *
 *      Major modifications include:
 *              . Conform to ANSI Standard C.
 *              . Removal of limit on number of array elements,
 *                because arrays in this program are dynamically
 *                allocated. This means that the program can accept
 *                any number of layers or gridlines as long as the
 *                memory permits.
 *              . Avoiding global variables whenever possible.  This
 *                program has not used global variables so far.
 *              . Grouping variables logically using structures.
 *              . Top-down design, keep each subroutine clear &
 *                short.
 *              . Reflectance and transmittance are angularly
 *                resolved.
 ****
 *  Modifications when including scattering particles made 1996 by
 *  P. Alsholm and A. Nilsson.
 ****
 *      General Naming Conventions:
 *      Preprocessor names: all capital letters,
 *              e.g. #define PREPROCESSORS
 *      Globals: first letter of each word is capital, no
 *              underscores,
 *              e.g. short GlobalVar;
 *      Dummy variables:  first letter of each word is capital,
 *              and words are connected by underscores,
 *              e.g. void NiceFunction(char Dummy_Var);
 *      Local variables:  all lower cases, words are connected
 *              by underscores,
 *              e.g. short local_var;
 *      Function names or data types:  same as Globals.
 *
 ****
 *      Dimension of length: cm.
 *
 ****/

#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <time.h>
#include <string.h>
#include <ctype.h>

#define PI 3.1415926
#define WEIGHT 1E-4             /* Critical weight for roulette. */
#define CHANCE 0.1              /* Chance of roulette survival. */
```

```
#define STRLEN 256              /* String length. */

#define Boolean char

#define SIGN(x) ((x)>=0 ? 1:-1)

/****************** Stuctures *****************************/

/****
 *      Structure used to describe a photon packet.
 ****/
typedef struct {
  double x, y ,z;       /* Cartesian coordinates.[cm] */
  double ux, uy, uz;     /* directional cosines of a photon. */
  double w;                     /* weight. */
  Boolean dead;                 /* 1 if photon is terminated. */
  short layer;          /* index to layer where the photon */
                                /* packet resides. */
  double s;                     /* current step size. [cm]. */
  double sleft;                 /* step size left. dimensionless [-]. */
} PhotonStruct;

/****
 * Structure used to describe the geometry and optical
 * properties of a layer.
 * z0 and z1 are the z coordinates for the upper boundary
 * and lower boundary respectively.
 *
 * cos_crit0 and cos_crit1 are the cosines of the
 * critical angle of total internal reflection for the
 * upper boundary and lower boundary respectively.
 * cos_critxy is analogous for the boundary to the
 * surrounding glass.
 * They are set to zero if no total internal reflection
 * exists.
 * They are used for computation speed.
 ****/
typedef struct {
  double z0, z1;        /* z coordinates of a layer. [cm] */
  double n;                     /* refractive index of a layer. */
  double abscross;       /* absorption cross section. [-] */
  double meanstep;       /* the step length. [cm]*/
  double meanstepdev;    /* deviation of step length. [cm] */
  double bcux;      /* orientation of blood cell. */
  double bcuy;      /* orientation of blood cell. */
  double bcuz;      /* orientation of blood cell. */
  double dev;       /* deviation of that orientation. [0.0-1.0] */

  double cos_crit0, cos_crit1, cos_critxy;
} LayerStruct;
```

```
/****
 * Input parameters for each independent run.
 *
 * z and r are for the cylindrical coordinate system. [cm]
 * a is for the angle alpha between the photon exiting
 * direction and the surface normal. [radian]
 *
 * The grid line separations in z, r, and alpha
 * directions are dz, dr, and da respectively.  The numbers
 * of grid lines in z, r, and alpha directions are
 * nz, nr, and na respectively.
 *
 * The member layerspecs will point to an array of
 * structures which store parameters of each layer.
 * This array has (number_layers + 3) elements. One
 * element is for a layer.
 * The layers 0 and (num_layers + 1) are for top ambient
 * medium and the bottom ambient medium respectively.
 * The layer (num_layers + 2) is for the surrounding glass.
 ****/
typedef struct {
  char   out_fname[STRLEN];    /* output file name. */
  char   out_fformat;          /* output file format. */
                                                /* 'A' for ASCII, */
                                                /* 'B' for binary. */
  char   dat_fname[STRLEN]; /* data base file name. */
  short  nbrangint;    /* data base parameter, number of incident angels.  */
  short  nbrthsc;      /* data base parameter, number of scattering theta. */
  short  nbrpssc;      /* data base parameter, number of scattering psi.   */
  double ** valmatrix;       /* data base values. */
  long   num_photons;          /* to be traced. */
  double Wth;                           /* play roulette if photon */
                                                /* weight < Wth.*/

  double dz;                            /* z grid separation.[cm] */
  double dr;                            /* r grid separation.[cm] */
  double da;                            /* alpha grid separation. */
                                                /* [radian] */
  short nz;                                     /* array range 0..nz-1. */
  short nr;                                     /* array range 0..nr-1. */
  short na;                                     /* array range 0..na-1. */

  short num_layers;                     /* number of layers. */

  double x_size, y_size;    /* dimension of the sample. [cm] */

  LayerStruct * layerspecs;   /* layer parameters. */
} InputStruct;
```

```
/****
 *      Structures for scoring physical quantities.
 *      z and r represent z and r coordinates of the
 *      cylindrical coordinate system. [cm]
 *      a is the angle alpha between the photon exiting
 *      direction and the normal to the surfaces. [radian]
 *      See comments of the InputStruct.
 *      See manual for the physcial quantities.
 ****/
typedef struct {
  double    Rsp;       /* specular reflectance. [-] */
  double ** Rd_ra;     /* 2D distribution of diffuse */
                       /* reflectance. [1/(cm2 sr)] */
  double *  Rd_r;      /* 1D radial distribution of diffuse */
                       /* reflectance. [1/cm2] */
  double *  Rd_a;      /* 1D angular distribution of diffuse */
                       /* reflectance. [1/sr] */
  double    Rd;              /* total diffuse reflectance. [-] */

  double ** A_rz;      /* 2D probability density in turbid */
                       /* media over r & z. [1/cm3] */
  double *  A_z;       /* 1D probability density over z. */
                       /* [1/cm] */
  double *  A_l;       /* each layer's absorption */
                       /* probability. [-] */
  double    A;         /* total absorption probability. [-] */

  double ** Tt_ra;     /* 2D distribution of total */
                       /* transmittance. [1/(cm2 sr)] */
  double *  Tt_r;      /* 1D radial distribution of */
                       /* transmittance. [1/cm2] */
  double *  Tt_a;      /* 1D angular distribution of */
                       /* transmittance. [1/sr] */
  double    Tt;              /* total transmittance. [-] */
} OutStruct;

/***********************************************************
 *      Routine prototypes for dynamic memory allocation and
 *      release of arrays and matrices.
 *      Modified from Numerical Recipes in C.
 ****/
double  *AllocVector(short, short);
double **AllocMatrix(short, short,short, short);
void     FreeVector(double *, short, short);
void     FreeMatrix(double **, short, short, short, short);
void                                nrerror(char                                *);
```

B.1.2.2. mcmlmain.c

```
/***********************************************************
 *      main program for Monte Carlo simulation of photon
 *      distribution in multi-layered turbid media.
 *
 ****/

/****
 *      THINKCPROFILER is defined to generate profiler calls in
 *      Think C. If 1, remember to turn on "Generate profiler
 *      calls" in the options menu.
 ****/
#define THINKCPROFILER 0

/* GNU cc does not support difftime() and CLOCKS_PER_SEC.*/
#define GNUCC 0

#if THINKCPROFILER
#include <profile.h>
#include <console.h>
#endif

#include "mcml.h"

/*      Declare before they are used in main(). */
void ShowVersion(char *);
FILE *GetFile(char *);
short ReadNumRuns(FILE* );
void ReadParm(FILE* , InputStruct * );
void CheckParm(FILE* , InputStruct * );
void ReadDataBase(FILE* , InputStruct * );
void InitOutputData(InputStruct, OutStruct *);
void FreeData(InputStruct, OutStruct *);
double Rspecular(LayerStruct * );
void LaunchPhoton(double, LayerStruct *, PhotonStruct *);
void HopDropSpin(InputStruct  *,PhotonStruct *,OutStruct *);
void SumScaleResult(InputStruct, OutStruct *);
void WriteResult(InputStruct, OutStruct, char *);


/***********************************************************
```

```
 *      If F = 0, reset the clock and return 0.
 *
 *      If F = 1, pass the user time to Msg and print Msg on
 *      screen, return the real time since F=0.
 *
 *      If F = 2, same as F=1 except no printing.
 *
 *      Note that clock() and time() return user time and real
 *      time respectively.
 *      User time is whatever the system allocates to the
 *      running of the program;
 *      real time is wall-clock time.  In a time-shared system,
 *      they need not be the same.
 *
 *      clock() only hold 16 bit integer, which is about 32768
 *      clock ticks.
 ****/
time_t PunchTime(char F, char *Msg)
{
#if GNUCC
  return(0);
#else
  static clock_t ut0;  /* user time reference. */
  static time_t  rt0;  /* real time reference. */
  double secs;
  char s[STRLEN];

  if(F==0) {
    ut0 = clock();
    rt0 = time(NULL);
    return(0);
  }
  else if(F==1)  {
    secs = (clock() - ut0)/(double)CLOCKS_PER_SEC;
    if (secs<0) secs=0;       /* clock() can overflow. */
    sprintf(s, "User time: %8.0lf sec = %8.2lf hr.  %s\n",
            secs, secs/3600.0, Msg);
    puts(s);
    strcpy(Msg, s);
    return(difftime(time(NULL), rt0));
  }
  else if(F==2) return(difftime(time(NULL), rt0));
  else return(0);
#endif
}

/*************************************************************
 *      Print the current time and the estimated finishing time.
 *
 *      P1 is the number of computed photon packets.
 *      Pt is the total number of photon packets.
 ****/
void PredictDoneTime(long P1, long Pt)
{
  time_t now, done_time;
  struct tm *date;
  char s[80];

  now = time(NULL);
  date = localtime(&now);
  strftime(s, 80, "%H:%M %x", date);
  printf("Now %s, ", s);

  done_time = now +
                      (time_t) (PunchTime(2,"")*(Pt-P1)/(double)P1);
  date = localtime(&done_time);
  strftime(s, 80, "%H:%M %x", date);
  printf("End %s\n", s);
}

/*************************************************************
 *      Report estimated time, number of photons and runs left
 *      after calculating 10 photons or every 1/10 of total
 *      number of photons.
 *
 *      Num_Runs is the number of runs left.
 *      Pi is the index to the current photon, counting down.
 *      Pt is the total number of photons.
 ****/
void ReportStatus(short Num_Runs,long Pi,long Pt)
{
  if(Pt-Pi == 10 || Pi*10%Pt == 0 && Pi != Pt) {
    printf("%ld photons & %hd runs left, ", Pi, Num_Runs);
    PredictDoneTime(Pt-Pi, Pt);
  }
}

/*************************************************************
 *      Report time and write results.
 ****/
void ReportResult(InputStruct In_Parm, OutStruct Out_Parm)
{
  char time_report[STRLEN];

  strcpy(time_report, " Simulation time of this run.");
  PunchTime(1, time_report);

  SumScaleResult(In_Parm, &Out_Parm);
  WriteResult(In_Parm, Out_Parm, time_report);
}
```

```
/************************************************************
 *      Get the file name of the input data file from the
 *      argument to the command line.
 ****/
void GetFnameFromArgv(int argc,
                                    char * argv[],
                                    char * input_filename)
{
  if(argc>=2) {                      /* filename in command line */
    strcpy(input_filename, argv[1]);
  }
  else
    input_filename[0] = '\0';
}


/************************************************************
 *      Execute Monte Carlo simulation for one independent run.
 ****/
void DoOneRun(short NumRuns, InputStruct *In_Ptr)
{
  register long i_photon;
        /* index to photon. register for speed.*/
  OutStruct out_parm;          /* distribution of photons.*/
  PhotonStruct photon;

#if THINKCPROFILER
  InitProfile(200,200); cecho2file("prof.rpt",0, stdout);
#endif

  InitOutputData(*In_Ptr, &out_parm);
  out_parm.Rsp = Rspecular(In_Ptr->layerspecs);
  i_photon = In_Ptr->num_photons;
  PunchTime(0, "");

  do {
    ReportStatus(NumRuns, i_photon, In_Ptr->num_photons);
    LaunchPhoton(out_parm.Rsp, In_Ptr->layerspecs, &photon);
    do  HopDropSpin(In_Ptr, &photon, &out_parm);
    while (!photon.dead);
  } while(--i_photon);

#if THINKCPROFILER
  exit(0);
#endif

  ReportResult(*In_Ptr, out_parm);
  FreeData(*In_Ptr, &out_parm);
}

/************************************************************
 *      The argument to the command line is filename, if any.
 *      Macintosh does not support command line.
 ****/
char main(int argc, char *argv[])
{
  char input_filename[STRLEN];
  FILE *input_file_ptr;
  short num_runs;      /* number of independent runs. */
  InputStruct in_parm;

  ShowVersion("Version 1.0, 1992");
  GetFnameFromArgv(argc, argv, input_filename);
  input_file_ptr = GetFile(input_filename);
  CheckParm(input_file_ptr, &in_parm);
  num_runs = ReadNumRuns(input_file_ptr);

  while(num_runs--)  {
    ReadDataBase(input_file_ptr, &in_parm);
    ReadParm(input_file_ptr, &in_parm);
       DoOneRun(num_runs, &in_parm);
  }

  fclose(input_file_ptr);
  return(0);
}
```

B.1.2.3. mcmlio.c

```
/************************************************************
 *      Input/output of data.
 ****/

#include "mcml.h"
```

```
/************************************************************
 *      Structure used to check against duplicated file names.
 ****/
struct NameList {
  char name[STRLEN];
  struct NameList * next;
```

```
};

typedef struct NameList NameNode;
typedef NameNode * NameLink;


/*************************************************************
 *      Center a string according to the column width.
 ****/
char * CenterStr(short  Wid,
                                char * InStr,
                                char * OutStr)
{
  size_t nspaces;      /* number of spaces to be filled */
                                     /* before InStr. */

  nspaces = (Wid - strlen(InStr))/2;
  if(nspaces<0) nspaces = 0;

  strcpy(OutStr, "");
  while(nspaces--)  strcat(OutStr, " ");

  strcat(OutStr, InStr);

  return(OutStr);
}

/*************************************************************
 *      Print some messages before starting simulation.
 *      e.g. author, address, program version, year.
 ****/
#define COLWIDTH 80
void ShowVersion(char *version)
{
  char str[STRLEN];

  CenterStr(COLWIDTH,
        "Monte Carlo Simulation of Multi-layered Turbid Media",
        str);
  puts(str);
  puts("");

  CenterStr(COLWIDTH, "Lihong Wang, Ph. D.", str);
  puts(str);

  CenterStr(COLWIDTH, "Steven L. Jacques, Ph. D.", str);
  puts(str);

  CenterStr(COLWIDTH,
        "Laser Biology Research Laboratory - 17",str);
  puts(str);

  CenterStr(COLWIDTH, "M.D. Anderson Cancer Center", str);
  puts(str);

  CenterStr(COLWIDTH, "University of Texas", str);
  puts(str);

  CenterStr(COLWIDTH, "Houston, TX 77030", str);
  puts(str);

  CenterStr(COLWIDTH, "Fax: (713)792-3995", str);
  puts(str);
  puts("");

  CenterStr(COLWIDTH, version, str);
  puts(str);

  CenterStr(COLWIDTH, "Modified 1996, PA - tmmcml", str);
  puts(str);
  puts("");

  puts("\n\n\n\n");
}
#undef COLWIDTH

/*************************************************************
 *      Get a filename and open it for reading, retry until
 *      the file can be opened.  '.' terminates the program.
 *
 *      If Fname != NULL, try Fname first.
 ****/
FILE *GetFile(char *Fname)
{
  FILE * file=NULL;
  Boolean firsttime=1;

  do {
    if(firsttime && Fname[0]!='\0') {
          /* use the filename from command line */
      firsttime = 0;
    }
    else {
      printf("Input filename(or . to exit):");
      scanf("%s", Fname);
      firsttime = 0;
    }

    if(strlen(Fname) == 1 && Fname[0] == '.')
      exit(1);                /* exit if no filename entered. */
```

```
    file = fopen(Fname, "r");
  }  while(file == NULL);

  return(file);
}

/*************************************************************
 *      Kill the ith char (counting from 0), push the following
 *      chars forward by one.
 ****/
void KillChar(size_t i, char * Str)
{
  size_t sl = strlen(Str);

  for(;i<sl;i++) Str[i] = Str[i+1];
}

/*************************************************************
 *      Eliminate the chars in a string which are not printing
 *      chars or spaces.
 *
 *      Spaces include ' ', '\f', '\t' etc.
 *
 *      Return 1 if no nonprinting chars found, otherwise
 *      return 0.
 ****/
Boolean CheckChar(char * Str)
{
  Boolean found = 0;   /* found bad char. */
  size_t sl = strlen(Str);
  size_t i=0;

  while(i<sl)
    if (Str[i]<0 || Str[i]>255)
      nrerror("Non-ASCII file\n");
    else if(isprint(Str[i]) || isspace(Str[i]))
      i++;
    else {
      found = 1;
      KillChar(i, Str);
      sl--;
    }

  return(found);
}

/*************************************************************
 *      Return 1 if this line is a comment line in which the
 *      first non-space character is "#".
 *
 *      Also return 1 if this line is space line.
 ****/
Boolean CommentLine(char *Buf)
{
  size_t spn, cspn;

  spn = strspn(Buf, " \t");
  /* length spanned by space or tab chars. */

  cspn = strcspn(Buf, "#\n");
  /* length before the 1st # or return. */

  if(spn == cspn)      /* comment line or space line. */
       return(1);
  else                          /* the line has data. */
       return(0);
}

/*************************************************************
 *      Skip space or comment lines and return a data line only.
 ****/
char * FindDataLine(FILE *File_Ptr)
{
  char buf[STRLEN];

  buf[0] = '\0';
  do {  /* skip space or comment lines. */
    if(fgets(buf, 255, File_Ptr) == NULL)  {
      printf("Incomplete data\n");
      buf[0]='\0';
      break;
    }
    else
      CheckChar(buf);
  } while(CommentLine(buf));

  return(buf);
}

/*************************************************************
 *      Skip file version, then read number of runs.
 ****/
short ReadNumRuns(FILE* File_Ptr)
{
  char buf[STRLEN];
  short n=0;

  FindDataLine(File_Ptr);    /* skip the fil version. */
```

```
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') nrerror("Reading number of runs\n");
  sscanf(buf, "%hd",&n);
  return(n);
}

/************************************************************
 * Place the data base values in a matrix.
 *
 * Valmatrix (i,j)  j = [1,nbrangint]
 *   The different incident angles [0,PI/2].
 *
 *   i = [1,num_rows]: num_rows = (nbrthsc-2)*nbrpssc-1)
 *
 * Valmatrix(1,*) : back scattering probability.
 *
 * Valmatrix(2,*) - Valmatrix(1,*) :
 *                  probability of scattering in direction
 *                  theta = 1/(nbrthsc-1), psi = 0.
 *
 *
 * Valmatrix(n,*) - Valmatrix(n-1,*) :
 *                  probability of scattering in direction
 *                  theta = ((n-2)%nbrpssc+1)/(nbrthsc-1),
 *                  psi   = ((n-2)/nbrpssc)/(nbrthsc-1).
 *
 *
 * Valmatrix((num_rows,*) : 1 - forward scattering probability.
 *
 * For forward and back scattering, psi is arbitrary.
 ********************/
void PlaceDataInMatrix(InputStruct * In_Ptr)
{
  short   nbrthsc = In_Ptr->nbrthsc;
  short   nbrpssc = In_Ptr->nbrpssc;
  short   thetacol = 1;
  short   num_thetacol = In_Ptr->nbrangint;
  short   row = 0;
  short   num_rows;
  double  value;
  char    buf[STRLEN];
  FILE *  Datfile_Ptr;

  num_rows = (nbrthsc-2)*nbrpssc + 1;
  /* number of data values for each incident angle according */
  /* to the remarks in MCDATA.FOR.  */

  In_Ptr->valmatrix = AllocMatrix(1,num_rows,1,num_thetacol);

  Datfile_Ptr = fopen(In_Ptr->dat_fname,"r");
  if (num_thetacol==1) {           /* sphere. */
    for (row=1; row <= num_rows; row++) {
      strcpy(buf, FindDataLine(Datfile_Ptr));
      if(buf[0]=='\0')
            nrerror("Reading data base value.\n");
      sscanf(buf, "%lf", &value);
      In_Ptr->valmatrix[row][thetacol] = value;
    }
  }
  else  {                   /* spheroid. */
    for (thetacol=1; thetacol<=num_thetacol; thetacol++)
      for (row=1; row <= num_rows; row++) {
        strcpy(buf, FindDataLine(Datfile_Ptr));
        if(buf[0]=='\0')
              nrerror("Reading data base value.\n");
        sscanf(buf, "%lf", &value);
        In_Ptr->valmatrix[row][thetacol] = value;
      }
  }
  fclose(Datfile_Ptr);
}

/************************************************************
 * Read the data base.
 ********************/
void ReadDataBase(FILE* File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in file name, nbrangint. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading data base parameters.\n");
  sscanf(buf, "%s %hd", In_Ptr->dat_fname, &In_Ptr->nbrangint);
  if(In_Ptr->nbrangint<=0)
       nrerror("Nonpositive number of angles.\n");
  /** read in nbrthsc, nbrpssc. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading data base parameters.\n");
  sscanf(buf, " %hd %hd", &In_Ptr->nbrthsc, &In_Ptr->nbrpssc);
  if(In_Ptr->nbrthsc<=1 || In_Ptr->nbrpssc<=1)
       nrerror("Wrong number of angles.\n");
  PlaceDataInMatrix(In_Ptr);
}

/************************************************************
 *      Read the file name and the file format.
 *
```

```
 *      The file format can be either A for ASCII or B for
 *      binary.
 ****/
void ReadFnameFormat(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in file name and format. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading file name and format.\n");
  sscanf(buf, "%s %c",
       In_Ptr->out_fname, &(In_Ptr->out_fformat) );
  if(toupper(In_Ptr->out_fformat) != 'B')
       In_Ptr->out_fformat = 'A';
}


/*************************************************************
 *      Read the number of photons.
 ****/
void ReadNumPhotons(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in number of photons. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading number of photons.\n");
  sscanf(buf, "%ld", &In_Ptr->num_photons);
  if(In_Ptr->num_photons<=0)
       nrerror("Nonpositive number of photons.\n");
}

/*************************************************************
 *      Read the members dz and dr.
 ****/
void ReadDzDr(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in dz, dr. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') nrerror("Reading dz, dr.\n");
  sscanf(buf, "%lf%lf", &In_Ptr->dz, &In_Ptr->dr);
  if(In_Ptr->dz<=0) nrerror("Nonpositive dz.\n");
  if(In_Ptr->dr<=0) nrerror("Nonpositive dr.\n");
}

/*************************************************************
 *      Read the members nz, nr, na.
 ****/
void ReadNzNrNa(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in number of dz, dr, da. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading number of dz, dr, da's.\n");
  sscanf(buf, "%hd%hd%hd",
       &In_Ptr->nz, &In_Ptr->nr, &In_Ptr->na);
  if(In_Ptr->nz<=0)
       nrerror("Nonpositive number of dz's.\n");
  if(In_Ptr->nr<=0)
       nrerror("Nonpositive number of dr's.\n");
  if(In_Ptr->na<=0)
       nrerror("Nonpositive number of da's.\n");
  In_Ptr->da = 0.5*PI/In_Ptr->na;
}

/*************************************************************
 *      Read the members x_size and y_size.
 ****/
void ReadXSizeYSize(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in x_size, y_size. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') nrerror("Reading x_size, y_size.\n");
  sscanf(buf, "%lf%lf", &In_Ptr->x_size, &In_Ptr->y_size);
  if(In_Ptr->x_size<=0) nrerror("Nonpositive x_size.\n");
  if(In_Ptr->y_size<=0) nrerror("Nonpositive y_size.\n");
}

/*************************************************************
 *      Read the number of layers.
 ****/
void ReadNumLayers(FILE *File_Ptr, InputStruct *In_Ptr)
{
  char buf[STRLEN];

  /** read in number of layers. **/
  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0')
       nrerror("Reading number of layers.\n");
  sscanf(buf, "%hd", &In_Ptr->num_layers);
  if(In_Ptr->num_layers<=0)
       nrerror("Nonpositive number of layers.\n");
```

```
}

/*************************************************************
 *      Read the refractive index n of the ambient.
 ****/
void ReadAmbient(FILE *File_Ptr,
                                LayerStruct * Layer_Ptr,
                                char *side)
{
  char buf[STRLEN], msg[STRLEN];
  double n;

  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') {
    sprintf(msg, "Rading n of %s ambient.\n", side);
       nrerror(msg);
  }

  sscanf(buf, "%lf", &n );
  if(n<=0) nrerror("Wrong n.\n");
  Layer_Ptr->n = n;
}

/*************************************************************
 *      Read the parameters of one layer.
 *
 *      Return 1 if error detected.
 *      Return 0 otherwise.
 *
 *      *Z_Ptr is the z coordinate of the current layer, which
 *      is used to convert thickness of layer to z coordinates
 *      of the two boundaries of the layer.
 ****/
Boolean ReadOneLayer(FILE *File_Ptr,
                                     LayerStruct * Layer_Ptr,
                                     double *Z_Ptr)
{
  char buf[STRLEN];
  double d;          /* thickness. */
  double n;          /* index of refraction. */
  double meanstep;   /* mean distance between particles. */
  double meanstepdev;/* standard deviation. */
  double abscross;       /* absorbed part in each particle. */
  double bcux, bcuy, bcuz; /* orientation of the particle. */
  double dev;   /* deviation [0.0-1.0]; 1.0 : no orientation at all.*/
  double temp;

  strcpy(buf, FindDataLine(File_Ptr));
  if(buf[0]=='\0') return(1); /* error. */

  sscanf(buf,"%lf%lf%lf%lf%lf%lf%lf%lf",   &n,   &meanstep,   &meanstepdev,
&abscross, &bcux, &bcuy, &bcuz, &dev, &d);
  if(d<0 || n<=0 || meanstep<0 || meanstepdev<0 ||
      abscross<0 || dev<0 || dev>1)
    return(1);               /* error. */

  Layer_Ptr->n     = n;
  Layer_Ptr->meanstep = meanstep;
  Layer_Ptr->meanstepdev = meanstepdev;
  Layer_Ptr->abscross = abscross;

  temp = sqrt(bcux*bcux+bcuy*bcuy+bcuz*bcuz);
  if (temp == 0.0) {  /* no orientation. */
    Layer_Ptr->dev    = 1.0;
    Layer_Ptr->bcux   = 0.0;
    Layer_Ptr->bcuy   = 0.0;
    Layer_Ptr->bcuz   = 0.0;
  }
  else {
    Layer_Ptr->bcux   = bcux/temp;  /* normalization. */
    Layer_Ptr->bcuy   = bcuy/temp;
    Layer_Ptr->bcuz   = bcuz/temp;
    Layer_Ptr->dev    = dev;
  }

  Layer_Ptr->z0       = *Z_Ptr;
  *Z_Ptr += d;
  Layer_Ptr->z1       = *Z_Ptr;

  return(0);
}

/*************************************************************
 *      Read and set the parameters of the surrounding glass.
 *
 *      Return 1 if error detected.
 *      Return 0 otherwise.
 *
 *      *Z_Ptr is the z coordinate of the current layer, which
 *      is used to convert thickness of layer to z coordinates
 *      of the two boundaries of the layer.
 ****/
Boolean ReadSurrGlass(FILE *File_Ptr,
                                        LayerStruct * Layer_Ptr,
                                        double *Z_Ptr)
{
  char buf[STRLEN];
  double n;

  strcpy(buf, FindDataLine(File_Ptr));
```

```
  if(buf[0]=='\0') return(1); /* error. */

  sscanf(buf, "%lf", &n);
  if(n<=0)
    return(1);                /* error. */

  Layer_Ptr->n = n;
  Layer_Ptr->meanstep = 0.0;
  Layer_Ptr->z0       = 0.0;
  Layer_Ptr->z1       = *Z_Ptr;

  return(0);
}

/*************************************************************
 *      Read the parameters of one layer at a time.
 ****/
void ReadLayerSpecs(FILE *File_Ptr,
                                    short Num_Layers,
                                    LayerStruct ** Layerspecs_PP)
{
  char msg[STRLEN];
  short i=0;
  double z = 0.0;      /* z coordinate of the current layer. */

  /* Allocate an array for the layer parameters. */
  /* layer 0 and layer Num_Layers + 1 are for ambient. */
  /* Num_Layers + 2 is for the surrounding glass. */
  *Layerspecs_PP = (LayerStruct *)
    malloc((unsigned) (Num_Layers+3)*sizeof(LayerStruct));
  if (!(*Layerspecs_PP))
    nrerror("allocation failure in ReadLayerSpecs()");

  ReadAmbient(File_Ptr, &((*Layerspecs_PP)[i]), "top");
  for(i=1; i<=Num_Layers; i++)
    if(ReadOneLayer(File_Ptr, &((*Layerspecs_PP)[i]), &z)) {
      sprintf(msg, "Error reading %hd of %hd layers\n",
                        i, Num_Layers);
          nrerror(msg);
    }
  ReadAmbient(File_Ptr, &((*Layerspecs_PP)[i]), "bottom");
  if(ReadSurrGlass(File_Ptr, &((*Layerspecs_PP)[i+1]), &z)) {
      sprintf(msg, "Error reading surrounding glass");
          nrerror(msg);
    }
}

/*************************************************************
 *      Compute the critical angles for total internal
 *      reflection according to the relative refractive index
 *      of the layer.
 *      All layers are processed.
 ****/
void CriticalAngle( short Num_Layers,
                                        LayerStruct ** Layerspecs_PP)
{
  short i=0;
  double n1, n2;

  for(i=1; i<=Num_Layers; i++)  {
    n1 = (*Layerspecs_PP)[i].n;
    n2 = (*Layerspecs_PP)[i-1].n;
    (*Layerspecs_PP)[i].cos_crit0 = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;

    n2 = (*Layerspecs_PP)[i+1].n;
    (*Layerspecs_PP)[i].cos_crit1 = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;
    n2 = (*Layerspecs_PP)[Num_Layers+2].n;
    (*Layerspecs_PP)[i].cos_critxy = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;
  }
  n1 = (*Layerspecs_PP)[Num_Layers+2].n;   /* surrounding glass */
  n2 = (*Layerspecs_PP)[0].n;
    (*Layerspecs_PP)[Num_Layers+2].cos_crit0 = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;
  n2 = (*Layerspecs_PP)[Num_Layers+1].n;
    (*Layerspecs_PP)[Num_Layers+2].cos_crit1 = n1>n2 ?
              sqrt(1.0 - n2*n2/(n1*n1)) : 0.0;
}

/*************************************************************
 *      Read in the input parameters for one run.
 ****/
void ReadParm(FILE* File_Ptr, InputStruct * In_Ptr)
{

  In_Ptr->Wth = WEIGHT;

  ReadFnameFormat(File_Ptr, In_Ptr);
  ReadNumPhotons(File_Ptr, In_Ptr);
  ReadDzDr(File_Ptr, In_Ptr);
  ReadNzNrNa(File_Ptr, In_Ptr);
  ReadXSizeYSize(File_Ptr, In_Ptr);
  ReadNumLayers(File_Ptr, In_Ptr);

  ReadLayerSpecs(File_Ptr, In_Ptr->num_layers,
                                &In_Ptr->layerspecs);
  CriticalAngle(In_Ptr->num_layers, &In_Ptr->layerspecs);
}
```

```
/*************************************************************
 *      Return 1, if the name in the name list.
 *      Return 0, otherwise.
 ****/
Boolean NameInList(char *Name, NameLink List)
{
  while (List != NULL) {
       if(strcmp(Name, List->name) == 0)
         return(1);
       List = List->next;
  };
  return(0);
}

/*************************************************************
 *      Add the name to the name list.
 ****/
void AddNameToList(char *Name, NameLink * List_Ptr)
{
  NameLink list = *List_Ptr;

  if(list == NULL) {   /* first node. */
       *List_Ptr = list = (NameLink)malloc(sizeof(NameNode));
       strcpy(list->name, Name);
       list->next = NULL;
  }
  else {                              /* subsequent nodes. */
       /* Move to the last node. */
       while(list->next != NULL)
         list = list->next;

       /* Append a node to the list. */
       list->next = (NameLink)malloc(sizeof(NameNode));
       list = list->next;
       strcpy(list->name, Name);
       list->next = NULL;
  }
}

/*************************************************************
 *      Check against duplicated file names.
 *
 *      A linked list is set up to store the file names used
 *      in this input data file.
 ****/
Boolean FnameTaken(char *fname, NameLink * List_Ptr)
{
  if(NameInList(fname, *List_Ptr))
       return(1);
  else {
       AddNameToList(fname, List_Ptr);
       return(0);
  }
}

/*************************************************************
 *      Free each node in the file name list.
 ****/
void FreeFnameList(NameLink List)
{
  NameLink next;

  while(List != NULL) {
       next = List->next;
       free(List);
       List = next;
  }
}

/*************************************************************
 *      Check the input parameters for each run.
 ****/
void CheckParm(FILE* File_Ptr, InputStruct * In_Ptr)
{
  short i_run;
  short num_runs;      /* number of independent runs. */
  NameLink head = NULL;
  Boolean name_taken;/* output files share the same */
                                        /* file name.*/
  char msg[STRLEN];

  num_runs = ReadNumRuns(File_Ptr);
  for(i_run=1; i_run<=num_runs; i_run++)  {
    printf("Checking input data for run %hd\n", i_run);
    FindDataLine(File_Ptr);  /* skip the data base parameters.*/
    FindDataLine(File_Ptr);  /*            "                  */
    ReadParm(File_Ptr, In_Ptr);

       name_taken = FnameTaken(In_Ptr->out_fname, &head);
       if(name_taken)
         sprintf(msg, "file name %s duplicated.\n",
                                        In_Ptr->out_fname);

    free(In_Ptr->layerspecs);
       if(name_taken) nrerror(msg);
  }
  FreeFnameList(head);
  rewind(File_Ptr);
}
```

```
/*************************************************************
 *      Allocate the arrays in OutStruct for one run, and
 *      array elements are automatically initialized to zeros.
 ****/
void InitOutputData(InputStruct In_Parm,
                                    OutStruct * Out_Ptr)
{
  short nz = In_Parm.nz;
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  short nl = In_Parm.num_layers;
  /* remember to use nl+2 because of 2 for ambient. */

  if(nz<=0 || nr<=0 || na<=0 || nl<=0)
    nrerror("Wrong grid parameters.\n");

  /* Init pure numbers. */
  Out_Ptr->Rsp = 0.0;
  Out_Ptr->Rd  = 0.0;
  Out_Ptr->A   = 0.0;
  Out_Ptr->Tt  = 0.0;

  /* Allocate the arrays and the matrices. */
  Out_Ptr->Rd_ra = AllocMatrix(0,nr-1,0,na-1);
  Out_Ptr->Rd_r  = AllocVector(0,nr-1);
  Out_Ptr->Rd_a  = AllocVector(0,na-1);

  Out_Ptr->A_rz  = AllocMatrix(0,nr-1,0,nz-1);
  Out_Ptr->A_z   = AllocVector(0,nz-1);
  Out_Ptr->A_l   = AllocVector(0,nl+1);

  Out_Ptr->Tt_ra = AllocMatrix(0,nr-1,0,na-1);
  Out_Ptr->Tt_r  = AllocVector(0,nr-1);
  Out_Ptr->Tt_a  = AllocVector(0,na-1);
}

/*************************************************************
 * Undo what InitOutputData did.
 * i.e. free the data allocations.
 ****/
void FreeData(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short num_thetacol=In_Parm.nbrangint;
  short num_rows;
  short nz = In_Parm.nz;
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  short nl = In_Parm.num_layers;
  /* remember to use nl+2 because of 2 for ambient. */

  num_rows = (In_Parm.nbrthsc-2)*In_Parm.nbrpssc + 1;
  /* number of data values for each incident angle according */
  /* to the remarks in MCDATA.FOR.                           */

  free(In_Parm.layerspecs);

  FreeMatrix(Out_Ptr->Rd_ra, 0,nr-1,0,na-1);
  FreeVector(Out_Ptr->Rd_r, 0,nr-1);
  FreeVector(Out_Ptr->Rd_a, 0,na-1);

  FreeMatrix(Out_Ptr->A_rz, 0, nr-1, 0,nz-1);
  FreeVector(Out_Ptr->A_z, 0, nz-1);
  FreeVector(Out_Ptr->A_l, 0,nl+1);

  FreeMatrix(Out_Ptr->Tt_ra, 0,nr-1,0,na-1);
  FreeVector(Out_Ptr->Tt_r, 0,nr-1);
  FreeVector(Out_Ptr->Tt_a, 0,na-1);

  FreeMatrix(In_Parm.valmatrix,1,num_rows,1,num_thetacol);
}

/*************************************************************
 *      Get 1D array elements by summing the 2D array elements.
 ****/
void Sum2DRd(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  short ir,ia;
  double sum;

  for(ir=0; ir<nr; ir++)  {
    sum = 0.0;
    for(ia=0; ia<na; ia++) sum += Out_Ptr->Rd_ra[ir][ia];
    Out_Ptr->Rd_r[ir] = sum;
  }

  for(ia=0; ia<na; ia++)  {
    sum = 0.0;
    for(ir=0; ir<nr; ir++) sum += Out_Ptr->Rd_ra[ir][ia];
    Out_Ptr->Rd_a[ia] = sum;
  }

  sum = 0.0;
  for(ir=0; ir<nr; ir++) sum += Out_Ptr->Rd_r[ir];
  Out_Ptr->Rd = sum;
}
```

```
/*************************************************************
 *      Return the index to the layer according to the index
 *      to the grid line system in z direction (Iz).
 *
 *      Use the center of box.
 ****/
short IzToLayer(short Iz, InputStruct In_Parm)
{
  short i=1;   /* index to layer. */
  short num_layers = In_Parm.num_layers;
  double dz = In_Parm.dz;

  while( (Iz+0.5)*dz >= In_Parm.layerspecs[i].z1
       && i<num_layers) i++;

  return(i);
}

/*************************************************************
 *      Get 1D array elements by summing the 2D array elements.
 ****/
void Sum2DA(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short nz = In_Parm.nz;
  short nr = In_Parm.nr;
  short iz,ir;
  double sum;

  for(iz=0; iz<nz; iz++)  {
    sum = 0.0;
    for(ir=0; ir<nr; ir++) sum += Out_Ptr->A_rz[ir][iz];
    Out_Ptr->A_z[iz] = sum;
  }

  sum = 0.0;
  for(iz=0; iz<nz; iz++) {
    sum += Out_Ptr->A_z[iz];
    Out_Ptr->A_l[IzToLayer(iz, In_Parm)]
         += Out_Ptr->A_z[iz];
  }
  Out_Ptr->A = sum;
}

/*************************************************************
 *      Get 1D array elements by summing the 2D array elements.
 ****/
void Sum2DTt(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  short ir,ia;
  double sum;

  for(ir=0; ir<nr; ir++)  {
    sum = 0.0;
    for(ia=0; ia<na; ia++) sum += Out_Ptr->Tt_ra[ir][ia];
    Out_Ptr->Tt_r[ir] = sum;
  }

  for(ia=0; ia<na; ia++)  {
    sum = 0.0;
    for(ir=0; ir<nr; ir++) sum += Out_Ptr->Tt_ra[ir][ia];
    Out_Ptr->Tt_a[ia] = sum;
  }

  sum = 0.0;
  for(ir=0; ir<nr; ir++) sum += Out_Ptr->Tt_r[ir];
  Out_Ptr->Tt = sum;
}

/*************************************************************
 *      Scale Rd and Tt properly.
 *
 *      "a" stands for angle alpha.
 ****
 *      Scale Rd(r,a) and Tt(r,a) by
 *       (area perpendicular to photon direction)
 *             x(solid angle)x(No. of photons).
 *      or
 *             [2*PI*r*dr*cos(a)]x[2*PI*sin(a)*da]x[No. of photons]
 *      or
 *             [2*PI*PI*dr*da*r*sin(2a)]x[No. of photons]
 ****
 *      Scale Rd(r) and Tt(r) by
 *             (area on the surface)x(No. of photons).
 ****
 *      Scale Rd(a) and Tt(a) by
 *             (solid angle)x(No. of photons).
 ****/
void ScaleRdTt(InputStruct In_Parm, OutStruct *      Out_Ptr)
{
  short nr = In_Parm.nr;
  short na = In_Parm.na;
  double dr = In_Parm.dr;
  double da = In_Parm.da;
  short ir,ia;
  double scale1, scale2;

  scale1 = 4.0*PI*PI*dr*sin(da/2)*In_Parm.num_photons;
       /* The factor (ir+0.5)*sin(2a) to be added. */
```

```
  for(ir=0; ir<nr; ir++)
    for(ia=0; ia<na; ia++) {
      scale2 = 1.0/((ir+0.5)*sin(2.0*(ia+0.5)*da)*scale1);
      Out_Ptr->Rd_ra[ir][ia] *= scale2;
      Out_Ptr->Tt_ra[ir][ia] *= scale2;
    }

  scale1 = 2.0*PI*dr*dr*In_Parm.num_photons;
       /* area is 2*PI*[(ir+0.5)*dr]*dr.*/
       /* ir+0.5 to be added. */

  for(ir=0; ir<nr; ir++) {
    scale2 = 1.0/((ir+0.5)*scale1);
    Out_Ptr->Rd_r[ir] *= scale2;
    Out_Ptr->Tt_r[ir] *= scale2;
  }

  scale1  = 2.0*PI*da*In_Parm.num_photons;
    /* solid angle is 2*PI*sin(a)*da. sin(a) to be added. */

  for(ia=0; ia<na; ia++) {
    scale2 = 1.0/(sin((ia+0.5)*da)*scale1);
    Out_Ptr->Rd_a[ia] *= scale2;
    Out_Ptr->Tt_a[ia] *= scale2;
  }

  scale2 = 1.0/(double)In_Parm.num_photons;
  Out_Ptr->Rd *= scale2;
  Out_Ptr->Tt *= scale2;
}

/*************************************************************
 *      Scale absorption arrays properly.
 ****/
void ScaleA(InputStruct In_Parm, OutStruct * Out_Ptr)
{
  short nz = In_Parm.nz;
  short nr = In_Parm.nr;
  double dz = In_Parm.dz;
  double dr = In_Parm.dr;
  short nl = In_Parm.num_layers;
  short iz,ir;
  short il;
  double scale1;

  /* Scale A_rz. */
  scale1 = 2.0*PI*dr*dr*dz*In_Parm.num_photons;
       /* volume is 2*pi*(ir+0.5)*dr*dr*dz.*/
       /* ir+0.5 to be added. */
  for(iz=0; iz<nz; iz++)
    for(ir=0; ir<nr; ir++)
      Out_Ptr->A_rz[ir][iz] /= (ir+0.5)*scale1;

  /* Scale A_z. */
  scale1 = 1.0/(dz*In_Parm.num_photons);
  for(iz=0; iz<nz; iz++)
    Out_Ptr->A_z[iz] *= scale1;

  /* Scale A_l. Avoid int/int. */
  scale1 = 1.0/(double)In_Parm.num_photons;
  for(il=0; il<=nl+1; il++)
    Out_Ptr->A_l[il] *= scale1;

  Out_Ptr->A *=scale1;
}

/*************************************************************
 *      Sum and scale results of current run.
 ****/
void SumScaleResult(InputStruct In_Parm,
                                        OutStruct * Out_Ptr)
{
  /* Get 1D & 0D results. */
  Sum2DRd(In_Parm, Out_Ptr);
  Sum2DA(In_Parm,  Out_Ptr);
  Sum2DTt(In_Parm, Out_Ptr);

  ScaleRdTt(In_Parm, Out_Ptr);
  ScaleA(In_Parm, Out_Ptr);
}

/*************************************************************
 *      Write the version number as the first string in the
 *      file.
 *      Use chars only so that they can be read as either
 *      ASCII or binary.
 ****/
void WriteVersion(FILE *file, char *Version)
{
  fprintf(file,
       "%s \t# Version number of the file format.\n\n",
       Version);
  fprintf(file, "####\n# Data categories include: \n");
  fprintf(file, "# InParm, RAT, \n");
  fprintf(file, "# A_l, A_z, Rd_r, Rd_a, Tt_r, Tt_a, \n");
  fprintf(file, "# A_rz, Rd_ra, Tt_ra \n####\n\n");
}

/*************************************************************
```

```
 *      Write the input parameters to the file.
 ****/
void WriteInParm(FILE *file, InputStruct In_Parm)
{
  short i;

  fprintf(file,
       "InParm \t\t\t# Input parameters. cm is used.\n");

  fprintf(file,
       "%s \tA\t# output file name, ASCII.\n",
       In_Parm.out_fname);
  fprintf(file,
       "%s \t\t\t# data base file name\n",
       In_Parm.dat_fname);
  fprintf(file,
       "%hd\t%hd\t%hd\t\t\t# data base param.\n",
        In_Parm.nbrangint, In_Parm.nbrthsc, In_Parm.nbrpssc);
  fprintf(file,
       "%ld \t\t\t\t# No. of photons\n", In_Parm.num_photons);

  fprintf(file,
       "%G\t%G\t\t\t# dz, dr [cm]\n", In_Parm.dz,In_Parm.dr);
  fprintf(file, "%hd\t%hd\t%hd\t\t# No. of dz, dr, da.\n",
         In_Parm.nz, In_Parm.nr, In_Parm.na);
  fprintf(file, "%G\t%G\t\t\t# x_size and y_size.\n\n",
         In_Parm.x_size, In_Parm.y_size);

  fprintf(file,
       "%hd\t\t\t\t\t\t\t\t# Number of layers\n",
       In_Parm.num_layers);
  fprintf(file,
       "#n\tmstep\tdev\tabs\tbcux\tbcuy\tbcuz\tdev\td\t# layer\n");
  fprintf(file,
       "%G\t\t\t\t\t\t\t\t# n for medium above\n",
       In_Parm.layerspecs[0].n);
  for(i=1; i<=In_Parm.num_layers; i++)  {
    LayerStruct s;
    s = In_Parm.layerspecs[i];
    fprintf(file, "%G\t%G\t%G\t%G\t%.3G\t%.3G\t%.3G\t%G\t%G\t# layer %hd\n",
                      s.n, s.meanstep, s.meanstepdev, s.abscross,
                      s.bcux, s.bcuy, s.bcuz, s.dev, s.z1-s.z0, i);
  }
  fprintf(file, "%G\t\t\t\t\t\t\t\t# n for medium below\n",
       In_Parm.layerspecs[i].n);
  fprintf(file, "%G\t\t\t\t\t\t\t\t# n for surrounding glass\n\n",
       In_Parm.layerspecs[i+1].n);
}

/*********************************************************
```

```
 *      Write reflectance, absorption, transmission.
 ****/
void WriteRAT(FILE * file, OutStruct Out_Parm)
{
  fprintf(file,
       "RAT\t\t\t #Reflectance, absorption, transmission. \n");
       /* flag. */

  fprintf(file,
       "%-14.6G \t#Specular reflectance [-]\n", Out_Parm.Rsp);
  fprintf(file,
       "%-14.6G \t#Diffuse reflectance [-]\n", Out_Parm.Rd);
  fprintf(file,
       "%-14.6G \t#Absorbed fraction [-]\n", Out_Parm.A);
  fprintf(file,
       "%-14.6G \t#Transmittance [-]\n", Out_Parm.Tt);

  fprintf(file, "\n");
}

/*********************************************************
 *      Write absorption as a function of layer.
 ****/
void WriteA_layer(FILE * file,
                                  short Num_Layers,
                                  OutStruct Out_Parm)
{
  short i;

  fprintf(file,
       "A_l #Absorption as a function of layer. [-]\n");
       /* flag. */

  for(i=1; i<=Num_Layers; i++)
    fprintf(file, "%12.4G\n", Out_Parm.A_l[i]);
  fprintf(file, "\n");
}

/*********************************************************
 *      5 numbers each line.
 ****/
void WriteRd_ra(FILE * file,
                                short Nr,
                                short Na,
                                OutStruct Out_Parm)
{
  short ir, ia;

  fprintf(file,
        "%s\n%s\n%s\n%s\n%s\n%s\n", /* flag. */
```

```
          "# Rd[r][angle]. [1/(cm2sr)].",
          "# Rd[0][0], [0][1],..[0][na-1]",
          "# Rd[1][0], [1][1],..[1][na-1]",
          "# ...",
          "# Rd[nr-1][0], [nr-1][1],..[nr-1][na-1]",
          "Rd_ra");

  for(ir=0;ir<Nr;ir++)
    for(ia=0;ia<Na;ia++)  {
      fprintf(file, "%12.4E ", Out_Parm.Rd_ra[ir][ia]);
      if( (ir*Na + ia + 1)%5 == 0) fprintf(file, "\n");
    }

  fprintf(file, "\n");
}

/************************************************************
 *     1 number each line.
 ****/
void WriteRd_r(FILE * file,
                        short Nr,
                        OutStruct Out_Parm)
{
  short ir;

  fprintf(file,
        "Rd_r #Rd[0], [1],..Rd[nr-1]. [1/cm2]\n");    /* flag. */

  for(ir=0;ir<Nr;ir++) {
    fprintf(file, "%12.4E\n", Out_Parm.Rd_r[ir]);
  }

  fprintf(file, "\n");
}

/************************************************************
 *     1 number each line.
 ****/
void WriteRd_a(FILE * file,
                        short Na,
                        OutStruct Out_Parm)
{
  short ia;

  fprintf(file,
        "Rd_a #Rd[0], [1],..Rd[na-1]. [sr-1]\n");     /* flag. */

  for(ia=0;ia<Na;ia++) {
    fprintf(file, "%12.4E\n", Out_Parm.Rd_a[ia]);
  }
```

```
  fprintf(file, "\n");
}

/************************************************************
 *     5 numbers each line.
 ****/
void WriteTt_ra(FILE * file,
                                short Nr,
                                short Na,
                                OutStruct Out_Parm)
{
  short ir, ia;

  fprintf(file,
          "%s\n%s\n%s\n%s\n%s\n%s\n", /* flag. */
          "# Tt[r][angle]. [1/(cm2sr)].",
          "# Tt[0][0], [0][1],..[0][na-1]",
          "# Tt[1][0], [1][1],..[1][na-1]",
          "# ...",
          "# Tt[nr-1][0], [nr-1][1],..[nr-1][na-1]",
          "Tt_ra");

  for(ir=0;ir<Nr;ir++)
    for(ia=0;ia<Na;ia++)  {
      fprintf(file, "%12.4E ", Out_Parm.Tt_ra[ir][ia]);
      if( (ir*Na + ia + 1)%5 == 0) fprintf(file, "\n");
    }

  fprintf(file, "\n");
}

/************************************************************
 *     5 numbers each line.
 ****/
void WriteA_rz(FILE * file,
                            short Nr,
                            short Nz,
                            OutStruct Out_Parm)
{
  short iz, ir;

  fprintf(file,
          "%s\n%s\n%s\n%s\n%s\n%s\n", /* flag. */
          "# A[r][z]. [1/cm3]",
          "# A[0][0], [0][1],..[0][nz-1]",
          "# A[1][0], [1][1],..[1][nz-1]",
          "# ...",
          "# A[nr-1][0], [nr-1][1],..[nr-1][nz-1]",
          "A_rz");
```

```
  for(ir=0;ir<Nr;ir++)
    for(iz=0;iz<Nz;iz++)  {
      fprintf(file, "%12.4E ", Out_Parm.A_rz[ir][iz]);
      if( (ir*Nz + iz + 1)%5 == 0) fprintf(file, "\n");
    }

  fprintf(file, "\n");
}

/*************************************************************
 *      1 number each line.
 ****/
void WriteA_z(FILE * file,
                        short Nz,
                        OutStruct Out_Parm)
{
  short iz;

  fprintf(file,
       "A_z #A[0], [1],..A[nz-1]. [1/cm]\n");       /* flag. */

  for(iz=0;iz<Nz;iz++) {
    fprintf(file, "%12.4E\n", Out_Parm.A_z[iz]);
  }

  fprintf(file, "\n");
}

/*************************************************************
 *      1 number each line.
 ****/
void WriteTt_r(FILE * file,
                        short Nr,
                        OutStruct Out_Parm)
{
  short ir;

  fprintf(file,
       "Tt_r #Tt[0], [1],..Tt[nr-1]. [1/cm2]\n"); /* flag. */

  for(ir=0;ir<Nr;ir++) {
    fprintf(file, "%12.4E\n", Out_Parm.Tt_r[ir]);
  }

  fprintf(file, "\n");
}

/*************************************************************
 *      1 number each line.
 ****/
void WriteTt_a(FILE * file,
                        short Na,
                        OutStruct Out_Parm)
{
  short ia;

  fprintf(file,
       "Tt_a #Tt[0], [1],..Tt[na-1]. [sr-1]\n"); /* flag. */

  for(ia=0;ia<Na;ia++) {
    fprintf(file, "%12.4E\n", Out_Parm.Tt_a[ia]);
  }

  fprintf(file, "\n");
}

/*************************************************************
 ****/
void WriteResult(InputStruct In_Parm,
                              OutStruct Out_Parm,
                              char * TimeReport)
{
  FILE *file;

  file = fopen(In_Parm.out_fname, "w");
  if(file == NULL) nrerror("Cannot open file to write.\n");

  if(toupper(In_Parm.out_fformat) == 'A')
       WriteVersion(file, "A1");
  else
       WriteVersion(file, "B1");

  fprintf(file, "# %s", TimeReport);
  fprintf(file, "\n");

  WriteInParm(file, In_Parm);
  WriteRAT(file, Out_Parm);
       /* reflectance, absorption, transmittance. */

  /* 1D arrays. */
  WriteA_layer(file, In_Parm.num_layers, Out_Parm);
  WriteA_z(file, In_Parm.nz, Out_Parm);
  WriteRd_r(file, In_Parm.nr, Out_Parm);
  WriteRd_a(file, In_Parm.na, Out_Parm);
  WriteTt_r(file, In_Parm.nr, Out_Parm);
  WriteTt_a(file, In_Parm.na, Out_Parm);

  /* 2D arrays. */
  WriteA_rz(file, In_Parm.nr, In_Parm.nz, Out_Parm);
```

```
  WriteRd_ra(file, In_Parm.nr, In_Parm.na, Out_Parm);
  WriteTt_ra(file, In_Parm.nr, In_Parm.na, Out_Parm);
```

B.1.2.4. mcmlgo.c

```
/***********************************************************
 *      Launch, move, and record photon weight.
 ****/

#include "mcml.h"

#define STANDARDTEST 0
  /* testing program using fixed rnd seed. */

#define PARTIALREFLECTION 0
  /* 1=split photon, 0=statistical reflection. */

#define COSZERO (1.0-1.0E-12)
  /* cosine of about 1e-6 rad. */

#define COS90D  1.0E-6
  /* cosine of about 1.57 - 1e-6 rad. */


/***********************************************************
 *      A random number generator from Numerical Recipes in C.
 ****/
#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC 1.0E-9

float ran3(int *idum)
{
  static int inext,inextp;
  static long ma[56];
  static int iff=0;
  long mj,mk;
  int i,ii,k;

  if (*idum < 0 || iff == 0) {
    iff=1;
    mj=MSEED-(*idum < 0 ? -*idum : *idum);
    mj %= MBIG;
    ma[55]=mj;
    mk=1;
    for (i=1;i<=54;i++) {
      ii=(21*i) % 55;
      ma[ii]=mk;
      mk=mj-mk;
      if (mk < MZ) mk += MBIG;
      mj=ma[ii];
    }
    for (k=1;k<=4;k++)
      for (i=1;i<=55;i++) {
        ma[i] -= ma[1+(i+30) % 55];
        if (ma[i] < MZ) ma[i] += MBIG;
      }
    inext=0;
    inextp=31;
    *idum=1;
  }
  if (++inext == 56) inext=1;
  if (++inextp == 56) inextp=1;
  mj=ma[inext]-ma[inextp];
  if (mj < MZ) mj += MBIG;
  ma[inext]=mj;
  return mj*FAC;
}

#undef MBIG
#undef MSEED
#undef MZ
#undef FAC


/***********************************************************
 *      Generate a random number between 0 and 1.  Take a
 *      number as seed the first time entering the function.
 *      The seed is limited to 1<<15.
 *      We found that when idum is too large, ran3 may return
 *      numbers beyond 0 and 1.
 ****/
double RandomNum(void)
{
  static Boolean first_time=1;
  static int idum;     /* seed for ran3. */

  if(first_time) {
    fclose(file);
}
```

```
#if STANDARDTEST /* Use fixed seed to test the program. */
    idum = - 1;
#else
    idum = -(int)time(NULL)%(1<<15);
         /* use 16-bit integer as the seed. */
#endif
    ran3(&idum);
    first_time = 0;
    idum = 1;
  }

  return( (double)ran3(&idum) );
}

/*************************************************************
 * Compute the specular reflection.
 *
 * If the first layer is a turbid medium, use the Fresnel
 * reflection from the boundary of the first layer as the
 * specular reflectance.
 *
 * If the first layer is glass, multiple reflections in
 * the first layer is considered to get the specular
 * reflectance.
 *
 * The subroutine assumes the Layerspecs array is correctly
 * initialized.
 ****/
double Rspecular(LayerStruct * Layerspecs_Ptr)
{
  double r1, r2;
       /* direct reflections from the 1st and 2nd layers. */
  double temp;

  temp =(Layerspecs_Ptr[0].n - Layerspecs_Ptr[1].n)
        /(Layerspecs_Ptr[0].n + Layerspecs_Ptr[1].n);
  r1 = temp*temp;

  if(Layerspecs_Ptr[1].meanstep == 0.0)  { /* glass layer. */
    temp = (Layerspecs_Ptr[1].n - Layerspecs_Ptr[2].n)
                /(Layerspecs_Ptr[1].n + Layerspecs_Ptr[2].n);
    r2 = temp*temp;
    r1 = r1 + (1-r1)*(1-r1)*r2/(1-r1*r2);
  }

  return (r1);
}

/*************************************************************
 * Initialize a photon packet.
 ****/
void LaunchPhoton(double Rspecular,
                                  LayerStruct  * Layerspecs_Ptr,
                                  PhotonStruct * Photon_Ptr)
{
  Photon_Ptr->w               = 1.0 - Rspecular;
  Photon_Ptr->dead = 0;
  Photon_Ptr->layer = 1;
  Photon_Ptr->s = 0;
  Photon_Ptr->sleft= 0;

  Photon_Ptr->x = 0.0;
  Photon_Ptr->y = 0.0;
  Photon_Ptr->z = 0.0;
  Photon_Ptr->ux = 0.0;
  Photon_Ptr->uy = 0.0;
  Photon_Ptr->uz = 1.0;

  if(Layerspecs_Ptr[1].meanstep == 0.0)  { /* glass layer. */
    Photon_Ptr->layer  = 2;
    Photon_Ptr->z      = Layerspecs_Ptr[2].z0;
  }
}

/*************************************************************
 * Calculate a value from a normal (Gaussian) distribution,
 * according to Numerical Recipes in C.
 * Meanvalue = 0, standard deviation = dev.
 ****/
double GaussDevStep(double dev)
{
  double fac, r, v1, v2, gaussdev;
  static double StepGset;
  static short StepIset = 0;

  if(StepIset == 0) {
    do {
      v1 = 2*RandomNum() - 1.0;
      v2 = 2*RandomNum() - 1.0;
      r = v1*v1+v2*v2;
      }
    while (r>=1.0 || r==0.0);
    fac = sqrt(-2.0*dev*dev*log(r)/r);
    StepGset = v1*fac;
    gaussdev = v2*fac;
    StepIset = 1;
    }
  else {
    StepIset = 0;
    gaussdev = StepGset;
```

```
  }
  return(gaussdev);
}

/*************************************************************
 * Calculate a value from a normal (Gaussian) distribution,
 * according to Numerical Recipes in C.
 * Meanvalue = 0, standard deviation = dev.
 ****/
double GaussDevSpin(double dev)
{
  double fac, r, v1, v2, gaussdev;
  static double SpinGset;
  static short SpinIset = 0;

  if(SpinIset == 0) {
    do {
      v1 = 2*RandomNum() - 1.0;
      v2 = 2*RandomNum() - 1.0;
      r = v1*v1+v2*v2;
      }
    while (r>=1.0 || r==0.0);
    fac = sqrt(-2.0*dev*dev*log(r)/r);
    SpinGset = v1*fac;
    gaussdev = v2*fac;
    SpinIset = 1;
    }
  else {
    SpinIset = 0;
    gaussdev = SpinGset;
  }
  return(gaussdev);
}

/*************************************************************
 * Choose a new direction for photon propagation by
 * sampling the polar deflection angle theta and the
 * azimuthal angle psi.
 *
 * Note:
 *  theta: 0 - pi so sin(theta) is always positive
 *  feel free to use sqrt() for cos(theta).
 *
 *  psi:   0 - 2pi
 *  for 0-pi  sin(psi) is +
 *  for pi-2pi sin(psi) is -
 ****/
void Spin(InputStruct * In_Ptr,
                PhotonStruct * Photon_Ptr)
{
short layer = Photon_Ptr->layer;

double cost, sint;
/* cosine and sine of the polar deflection angle theta. */
double cosp, sinp;
/* cosine and sine of the azimuthal angle psi. */
double ux = Photon_Ptr->ux;     /* photon direction. */
double uy = Photon_Ptr->uy;     /* photon direction. */
double uz = Photon_Ptr->uz;     /* photon direction. */
double bcux ;                   /* spheroid direction. */
double bcuy ;                   /* spheroid direction. */
double bcuz ;                   /* spheroid direction. */
double angin;                   /* the incident angle. */
double theta;                   /* scattering angle. */
double psi;                     /* scattering angle. */
double dev = In_Ptr->layerspecs[layer].dev;
short  thetacol;
short  thetacolshort;
double thetacoldouble;
double rnd = RandomNum();
short  num_rows;
short  num_thetacol = In_Ptr->nbrangint;
short  nbrthsc = In_Ptr->nbrthsc;
short  nbrpssc = In_Ptr->nbrpssc;
short  ipsi, itheta;
short  i;
double temp;

num_rows = (nbrthsc-2)*nbrpssc + 1;
/* number of data values for each incident angle according */
/* to the remarks in MCDATA.FOR.                           */

if (num_thetacol == 1)  /* sphere. */
  thetacol = 1;         /* independent of incident angle. */
else  {                 /* spheroid. */
  if (dev==1.0) {       /* randomly oriented spheroids. */
    do {
      bcux = cos(PI*RandomNum());
      bcuy = cos(PI*RandomNum());
      bcuz = cos(PI*RandomNum());
      temp = sqrt(bcux*bcux+bcuy*bcuy+bcuz*bcuz);
    }
    while (temp == 0.0);
  }
  else {                /* orientation and deviation. */
    bcux = In_Ptr->layerspecs[layer].bcux;
    bcuy = In_Ptr->layerspecs[layer].bcuy;
    bcuz = In_Ptr->layerspecs[layer].bcuz;
    temp = 1.0;
    if (dev != 0.0)  {
```

```
        do {
          bcux = bcux + GaussDevSpin(dev);
          bcuy = bcuy + GaussDevSpin(dev);
          bcuz = bcuz + GaussDevSpin(dev);
          temp = sqrt(bcux*bcux+bcuy*bcuy+bcuz*bcuz);
        }
        while (temp == 0.0);
      }
    }
    angin = acos((ux*bcux+uy*bcuy+uz*bcuz)/temp);

/* the incident angle angin is the angle between the photon direction */
          /* and the symmetry axis of the particle. */

    if (angin>PI/2)
      angin = PI-angin;

    thetacolshort=(angin/(PI/2))*(num_thetacol-1)+1;  /* integer value. */
    thetacoldouble=(angin/(PI/2))*(num_thetacol-1)+1; /* real value.    */
    if (thetacoldouble - thetacolshort < 0.5)
      thetacol = thetacolshort;
    else
      thetacol = thetacolshort + 1;
  }

  /* The discrete scattering directions are evaluated and */
              /* spread out continuously. */

  if (rnd<In_Ptr->valmatrix[num_rows][thetacol]){/* forw. scattering. */
    theta = PI*0.5*RandomNum()/(nbrthsc-1);
    psi = PI*RandomNum();                       /* psi is arbitrary. */
  }
  else if (rnd<In_Ptr->valmatrix[1][thetacol])  { /* back scattering. */
    theta = PI - PI*0.5*RandomNum()/(nbrthsc-1);
    psi = PI*RandomNum();                       /* psi is arbitrary. */
  }
  else {
    i = 2;
    while (In_Ptr->valmatrix[i][thetacol]<rnd)
      i++;

    itheta   = (i-2)/nbrpssc+1;
      /* itheta = [0,1,nbrthsc-1] ==> theta = [0,1/(nbrthsc-1),PI]  */
    ipsi   = (i-2)%nbrpssc;
      /* ipsi = [0,1,nbrpssc-1] ==> psi = [0,1/(nbrpssc-1),PI]    */
    theta = PI*(itheta+(RandomNum()-0.5))/(nbrthsc-1);
    psi = PI*(ipsi+(RandomNum()-0.5))/(nbrpssc-1);
  }

  if (RandomNum() > 0.5)
    psi = 2*PI - psi; /* symmetry property */

  cost = cos(theta);
  sint = sqrt(1.0 - cost*cost);
      /* sqrt() is faster than sin(). */

  cosp = cos(psi);
  if(psi<PI)
     sinp = sqrt(1.0 - cosp*cosp);
           /* sqrt() is faster than sin(). */
  else
      sinp = - sqrt(1.0 - cosp*cosp);

  if(fabs(uz) > COSZERO)  {   /* normal incident. */
    Photon_Ptr->ux = sint*cosp;
    Photon_Ptr->uy = sint*sinp;
    Photon_Ptr->uz = cost*SIGN(uz);
         /* SIGN() is faster than division. */
  }
  else  {              /* regular incident. */
    double temp = sqrt(1.0 - uz*uz);
    Photon_Ptr->ux = sint*(ux*uz*cosp - uy*sinp)
                                        /temp + ux*cost;
    Photon_Ptr->uy = sint*(uy*uz*cosp + ux*sinp)
                                        /temp + uy*cost;
    Photon_Ptr->uz = -sint*cosp*temp + uz*cost;
  }
}

/*************************************************************
 * Move the photon s away in the current layer of medium.
 ****/
void Hop(PhotonStruct *       Photon_Ptr)
{
  double s = Photon_Ptr->s;

  Photon_Ptr->x += s*Photon_Ptr->ux;
  Photon_Ptr->y += s*Photon_Ptr->uy;
  Photon_Ptr->z += s*Photon_Ptr->uz;
}

/*************************************************************
 * Return the photon step size in glass and the
 * boundary which is hit.
 *
 * The step size is the distance between the current
 * position and the nearest boundary in the photon direction.
 *
 * uz != 0
 *
```

```
 ****/
short StepSizeInGlass(PhotonStruct *  Photon_Ptr,
                                      InputStruct  *  In_Ptr)
{
  double dl_b;       /* distance to nearest boundary. */
  double dl_xbound;  /* distance to boundary perp. to x-direction. */
  double dl_ybound;  /* distance to boundary perp. to y-direction. */
  double dl_zbound;  /* distance to boundary perp. to z-direction. */
  short  layer = Photon_Ptr->layer;
  double ux = Photon_Ptr->ux;
  double uy = Photon_Ptr->uy;
  double uz = Photon_Ptr->uz;
  short hitcase;     /* the boundary which is hit. */
  /* hitcase = 1,2 or 3 for boundary perp. to the */
             /* x-, y- or z-directions. */

  /* distance to the nearest boundary. */
  if(uz>0.0)
    dl_zbound = (In_Ptr->layerspecs[layer].z1
                      - Photon_Ptr->z)/uz;   /* dl_zbound>0. */
  else if(uz<0.0)
    dl_zbound = (In_Ptr->layerspecs[layer].z0
                      - Photon_Ptr->z)/uz;   /* dl_zbound>0. */
  dl_b = dl_zbound;
  hitcase = 3;
  if(ux != 0.0) {
    dl_xbound = ((In_Ptr->x_size)*SIGN(ux)/2 - Photon_Ptr->x)/ux;
    if (dl_xbound<dl_b) {
      dl_b = dl_xbound;
      hitcase = 1;
    }
  }
  if(uy != 0.0) {
    dl_ybound = ((In_Ptr->y_size)*SIGN(uy)/2 - Photon_Ptr->y)/uy;
    if (dl_ybound<dl_b) {
      dl_b = dl_ybound;
      hitcase = 2;
    }
  }
  Photon_Ptr->s = dl_b;

  return(hitcase);
}

/*************************************************************
 * Return the photon step size in surrouning glass,
 *
 * The step size is the distance between the current
 * position and the boundary, perp. to z-direction,
 * in the photon direction.
 *
 *  uz != 0
 ****/
void StepSizeInSurrGlass(PhotonStruct *  Photon_Ptr,
                                         InputStruct  *  In_Ptr)
{
  double dl_b;       /* distance to boundary. */
  short  layer = Photon_Ptr->layer;
  double uz = Photon_Ptr->uz;

  if(uz>0.0)
    dl_b = (In_Ptr->layerspecs[layer].z1
                      - Photon_Ptr->z)/uz;   /* dl_b>0. */
  else if(uz<0.0)
    dl_b = (In_Ptr->layerspecs[layer].z0
                      - Photon_Ptr->z)/uz;   /* dl_b>0. */

  Photon_Ptr->s = dl_b;
}

/*************************************************************
 * Pick a step size for a photon packet when it is in
 * tissue.
 * If the member sleft is zero, make a new step.
 * Otherwise, pick up the leftover in sleft.
 *
 * Layer is the index to layer.
 * In_Ptr is the input parameters.
 *
 * scale is used instead of mut in the original MCML.
 ***/
void StepSizeInTissue(PhotonStruct * Photon_Ptr,
                                       InputStruct  * In_Ptr)
{
  short  layer = Photon_Ptr->layer;
  double meanstep = In_Ptr->layerspecs[layer].meanstep;
  double scale = In_Ptr->layerspecs[layer].meanstep;
  double meanstepdev = In_Ptr->layerspecs[layer].meanstepdev;
  double stepdev;         /* standard deviation of meanstep. */

  do
    stepdev = GaussDevStep(meanstepdev);
  while (stepdev<-meanstep/2); /* meanstep+stepdev must not be to small. */
  meanstep = meanstep + stepdev;

  if(Photon_Ptr->sleft == 0.0) {  /* make a new step. */
       Photon_Ptr->s = meanstep;
  }
  else {       /* take the leftover. */
       Photon_Ptr->s = Photon_Ptr->sleft/scale;
```

```
        Photon_Ptr->sleft = 0.0;
    }
}

/*************************************************************
 * Check if the step will hit a boundary.
 * Return 1, 2 or 3 if a boundary is hit.
 * Return 0 otherwise.
 *
 * If the projected step hits the boundary, the members
 * s and sleft of Photon_Ptr are updated.
 *
 * scale is used instead of mua+mut in the original program.
 ****/
short HitBoundary(PhotonStruct *  Photon_Ptr,
                                    InputStruct  *  In_Ptr)
{
  double dl_b;       /* distance to nearest boundary. */
  double dl_xbound;  /* distance to boundary perp. to x-direction. */
  double dl_ybound;  /* distance to boundary perp. to y-direction. */
  double dl_zbound;  /* distance to boundary perp. to z-direction. */
  short  layer = Photon_Ptr->layer;
  double ux = Photon_Ptr->ux;
  double uy = Photon_Ptr->uy;
  double uz = Photon_Ptr->uz;
  short hitcase;     /* the boundary which is hit. */
   /* hitcase = 1,2 or 3 for boundary perp. to the x-, y- or z-directions. */
   /* hitcase = 0 if no boundary is hit. */
  double maxdl_xybound = sqrt(pow(In_Ptr->x_size,2.0)
                       + pow(In_Ptr->y_size,2.0)
                       + pow(In_Ptr->layerspecs[layer].z1
                             - In_Ptr->layerspecs[layer].z0,2.0));
                       /* The longest possible path in this layer. */

  if(uz>0.0)
    dl_zbound = (In_Ptr->layerspecs[layer].z1
                       - Photon_Ptr->z)/uz;   /* dl_zbound>0. */
  else if(uz<0.0)
    dl_zbound = (In_Ptr->layerspecs[layer].z0
                       - Photon_Ptr->z)/uz;   /* dl_zbound>0. */
  if (uz == 0.0)
    dl_b = maxdl_xybound;
  else  {
    dl_b = dl_zbound;
    hitcase = 3;
  }
  if(ux != 0.0) {
    dl_xbound = ((In_Ptr->x_size)*SIGN(ux)/2 - Photon_Ptr->x)/ux;
    if (dl_xbound<dl_b) {
      dl_b = dl_xbound;
      hitcase = 1;
    }
  }
  if(uy != 0.0) {
    dl_ybound = ((In_Ptr->y_size)*SIGN(uy)/2 - Photon_Ptr->y)/uy;
    if (dl_ybound<dl_b) {
      dl_b = dl_ybound;
      hitcase = 2;
    }
  }
  if(Photon_Ptr->s > dl_b) {
        /* crossing. */
    double scale = In_Ptr->layerspecs[layer].meanstep;

    Photon_Ptr->sleft = (Photon_Ptr->s - dl_b)*scale;
    Photon_Ptr->s     = dl_b;
  }
  else
    hitcase = 0;

  return(hitcase);
}

/*************************************************************
 * Drop photon weight inside the tissue (not glass).
 *
 * The photon is assumed not dead.
 *
 * The weight drop is dwa = abscross*w.
 *
 * The dropped weight is assigned to the absorption array
 * elements.
 ****/
void Drop(InputStruct  *      In_Ptr,
                PhotonStruct *       Photon_Ptr,
                OutStruct  *         Out_Ptr)
{
  double dwa;          /* absorbed weight.*/
  double x = Photon_Ptr->x;
  double y = Photon_Ptr->y;
  short  iz, ir;       /* index to z & r. */
  short  layer = Photon_Ptr->layer;
  double abscross;

  /* compute array indices. */
  iz = (short)(Photon_Ptr->z/In_Ptr->dz);
  if(iz>In_Ptr->nz-1) iz=In_Ptr->nz-1;

  ir = (short)(sqrt(x*x+y*y)/In_Ptr->dr);
  if(ir>In_Ptr->nr-1) ir=In_Ptr->nr-1;
```

```
  /* update photon weight. */
  abscross = In_Ptr->layerspecs[layer].abscross;
  dwa = Photon_Ptr->w * abscross;
  Photon_Ptr->w -= dwa;

  /* assign dwa to the absorption array element. */
  Out_Ptr->A_rz[ir][iz]       += dwa;
}

/*************************************************************
 *      The photon weight is small, and the photon packet tries
 *      to survive a roulette.
 ****/
void Roulette(PhotonStruct * Photon_Ptr)
{
  if(Photon_Ptr->w == 0.0)
    Photon_Ptr->dead = 1;
  else if(RandomNum() < CHANCE) /* survived the roulette.*/
    Photon_Ptr->w /= CHANCE;
  else
    Photon_Ptr->dead = 1;
}

/*************************************************************
 *      Compute the Fresnel reflectance.
 *
 *      Make sure that the cosine of the incident angle a1
 *      is positive, and the case when the angle is greater
 *      than the critical angle is ruled out.
 *
 *      Avoid trigonometric function operations as much as
 *      possible, because they are computation-intensive.
 ****/
double RFresnel(double n1,     /* incident refractive index.*/
                double n2,     /* transmit refractive index.*/
                double ca1,    /* cosine of the incident */
                               /*     angle.     0<a1<90
degrees. */
                double * ca2_Ptr)  /* pointer to the */
                               /* cosine of the transmission */
                               /* angle. a2>0. */
{
  double r;

  if(n1==n2) {                      /** matched boundary. **/
    *ca2_Ptr = ca1;
    r = 0.0;
  }
  else if(ca1>COSZERO) {      /** normal incident. **/
    *ca2_Ptr = ca1;
    r = (n2-n1)/(n2+n1);
    r *= r;
  }
  else if(ca1<COS90D)  {      /** very slant. **/
    *ca2_Ptr = 0.0;
    r = 1.0;
  }
  else  {                                   /** general. **/
    double sa1, sa2;
         /* sine of the incident and transmission angles. */
    double ca2;

    sa1 = sqrt(1-ca1*ca1);
    sa2 = n1*sa1/n2;
    if(sa2>=1.0) {
         /* double check for total internal reflection. */
      *ca2_Ptr = 0.0;
      r = 1.0;
    }
    else  {
      double cap, cam; /* cosines of the sum ap or */
                                              /* difference am of the two */
                                              /* angles. ap = a1+a2 */
                                              /* am = a1 - a2. */
      double sap, sam; /* sines. */

      *ca2_Ptr = ca2 = sqrt(1-sa2*sa2);

      cap = ca1*ca2 - sa1*sa2; /* c+ = cc - ss. */
      cam = ca1*ca2 + sa1*sa2; /* c- = cc + ss. */
      sap = sa1*ca2 + ca1*sa2; /* s+ = sc + cs. */
      sam = sa1*ca2 - ca1*sa2; /* s- = sc - cs. */
      r = 0.5*sam*sam*(cam*cam+cap*cap)/(sap*sap*cam*cam);
               /* rearranged for speed. */
    }
  }
  return(r);
}

/*************************************************************
 *      Record the photon weight exiting the first layer(uz<0),
 *      no matter whether the layer is glass or not, to the
 *      reflection array.
 *
 *      Update the photon weight as well.
 ****/
void RecordR(double                  Refl,   /* reflectance. */
                        InputStruct  *       In_Ptr,
                        PhotonStruct *       Photon_Ptr,
```

```
                        OutStruct *   Out_Ptr)
{
  double x = Photon_Ptr->x;
  double y = Photon_Ptr->y;
  short  ir, ia;       /* index to r & angle. */

  ir = (short)(sqrt(x*x+y*y)/In_Ptr->dr);
  if(ir>In_Ptr->nr-1) ir=In_Ptr->nr-1;

  ia = (short)(acos(-Photon_Ptr->uz)/In_Ptr->da);
  if(ia>In_Ptr->na-1) ia=In_Ptr->na-1;

  /* assign photon to the reflection array element. */
  Out_Ptr->Rd_ra[ir][ia] += Photon_Ptr->w*(1.0-Refl);

  Photon_Ptr->w *= Refl;
}

/***********************************************************
 *      Record the photon weight exiting the last layer(uz>0),
 *      no matter whether the layer is glass or not, to the
 *      transmittance array.
 *
 *      Update the photon weight as well.
 ****/
void RecordT(double           Refl,
                        InputStruct  *       In_Ptr,
                        PhotonStruct *       Photon_Ptr,
                        OutStruct *   Out_Ptr)
{
  double x = Photon_Ptr->x;
  double y = Photon_Ptr->y;
  short  ir, ia;       /* index to r & angle. */

  ir = (short)(sqrt(x*x+y*y)/In_Ptr->dr);
  if(ir>In_Ptr->nr-1) ir=In_Ptr->nr-1;

  ia = (short)(acos(Photon_Ptr->uz)/In_Ptr->da);
  if(ia>In_Ptr->na-1) ia=In_Ptr->na-1;

  /* assign photon to the transmittance array element. */
  Out_Ptr->Tt_ra[ir][ia] += Photon_Ptr->w*(1.0-Refl);

  Photon_Ptr->w *= Refl;
}

/***********************************************************
 * Decide whether the photon will be transmitted or
 * reflected on the upper boundary (uz<0) of the current
 * layer.
 *
 * If "layer" is the first or surrounding layer,
 * the photon packet will
 * be partially transmitted and partially reflected if
 * PARTIALREFLECTION is set to 1,
 * or the photon packet will be either transmitted or
 * reflected determined statistically if PARTIALREFLECTION
 * is set to 0.
 *
 * Record the transmitted photon weight as reflection.
 *
 * If "layer" is not the first or surrounding
 * layer and the photon packet is transmitted,
 * move the photon to "layer-1".
 *
 * Update the photon parmameters.
 ****/
void CrossUpOrNot(InputStruct  *      In_Ptr,
                                      PhotonStruct *       Photon_Ptr,
                                      OutStruct *          Out_Ptr)
{
  double uz = Photon_Ptr->uz; /* z directional cosine. */
  double uz1;  /* cosines of transmission alpha, always positive. */
  double r=0.0;        /* reflectance. */
  short  layer = Photon_Ptr->layer;
  double ni = In_Ptr->layerspecs[layer].n;
  double nt = In_Ptr->layerspecs[layer-1].n;

  /* Get r. */
  if( - uz <= In_Ptr->layerspecs[layer].cos_crit0)
    r=1.0;                  /* total internal reflection. */
  else if (layer == In_Ptr->num_layers + 2)     /* surr. glass. */
    r = RFresnel(In_Ptr->layerspecs[layer].n,
                 In_Ptr->layerspecs[0].n, -uz, &uz1);
  else r = RFresnel(In_Ptr->layerspecs[layer].n,
                 In_Ptr->layerspecs[layer-1].n, -uz, &uz1);

  if (r == 1.0 && layer == In_Ptr->num_layers + 2)
    Photon_Ptr->dead = 1;
      /* total internal reflection in surrounding medium. */

#if PARTIALREFLECTION
  if((layer == 1 || layer == In_Ptr->num_layers + 2)
      && r<1.0) {      /* partially transmitted. */
    Photon_Ptr->uz = -uz1;    /* transmitted photon. */
    RecordR(r, In_Ptr, Photon_Ptr, Out_Ptr);
    Photon_Ptr->uz = -uz;     /* reflected photon. */
  }
  else if(RandomNum() > r) {/* transmitted. */
    Photon_Ptr->layer--;
```

```
    Photon_Ptr->ux *= ni/nt;
    Photon_Ptr->uy *= ni/nt;
    Photon_Ptr->uz = -uz1;
  }
  else                              /* reflected. */
    Photon_Ptr->uz = -uz;
#else
  if(RandomNum() > r) {             /* transmitted. */
    if(layer==1 || layer == In_Ptr->num_layers + 2)  {
      Photon_Ptr->uz = -uz1;
      RecordR(0.0, In_Ptr, Photon_Ptr, Out_Ptr);
      Photon_Ptr->dead = 1;
    }
    else {
      Photon_Ptr->layer--;
      Photon_Ptr->ux *= ni/nt;
      Photon_Ptr->uy *= ni/nt;
      Photon_Ptr->uz = -uz1;
    }
  }
  else                                      /* reflected. */
    Photon_Ptr->uz = -uz;
#endif
}

/***********************************************************
 * Decide whether the photon will be transmitted  or be
 * reflected on the bottom boundary (uz>0) of the current
 * layer.
 *
 * If the photon is transmitted, move the photon to
 * "layer+1". If "layer" is the last or surrounding layer,
 * record the transmitted weight as transmittance.
 * See comments for CrossUpOrNot.
 *
 * Update the photon parmameters.
 ****/
void CrossDnOrNot(InputStruct  *     In_Ptr,
                  PhotonStruct *     Photon_Ptr,
                  OutStruct *        Out_Ptr)
{
  double uz = Photon_Ptr->uz; /* z directional cosine. */
  double uz1;  /* cosines of transmission alpha. */
  double r=0.0;       /* reflectance. */
  short  layer = Photon_Ptr->layer;
  double ni = In_Ptr->layerspecs[layer].n;
  double nt = In_Ptr->layerspecs[layer+1].n;

  /* Get r. */
  if( uz <= In_Ptr->layerspecs[layer].cos_crit1)
    r=1.0;            /* total internal reflection. */
  else if (layer == In_Ptr->num_layers + 2)     /* surr. glass. */
    r = RFresnel(In_Ptr->layerspecs[layer].n,
                 In_Ptr->layerspecs[layer-1].n, uz, &uz1);
  else r = RFresnel(In_Ptr->layerspecs[layer].n,
                 In_Ptr->layerspecs[layer+1].n, uz, &uz1);

#if PARTIALREFLECTION
  if(layer == In_Ptr->num_layers || layer == In_Ptr->num_layers + 2)
      && r<1.0) {
    Photon_Ptr->uz = uz1;
    RecordT(r, In_Ptr, Photon_Ptr, Out_Ptr);
    Photon_Ptr->uz = -uz;
  }
  else if(RandomNum() > r) {/* transmitted. */
    Photon_Ptr->layer++;
    Photon_Ptr->ux *= ni/nt;
    Photon_Ptr->uy *= ni/nt;
    Photon_Ptr->uz = uz1;
  }
  else                                          /* reflected. */
    Photon_Ptr->uz = -uz;
#else
  if(RandomNum() > r) {               /* transmitted. */
    if(layer == In_Ptr->num_layers || layer == In_Ptr->num_layers + 2) {
      Photon_Ptr->uz = uz1;
      RecordT(0.0, In_Ptr, Photon_Ptr, Out_Ptr);
      Photon_Ptr->dead = 1;
    }
    else {
      Photon_Ptr->layer++;
      Photon_Ptr->ux *= ni/nt;
      Photon_Ptr->uy *= ni/nt;
      Photon_Ptr->uz = uz1;
    }
  }
  else                                          /* reflected. */
    Photon_Ptr->uz = -uz;
#endif
}

/***********************************************************
 * Decide whether the photon will be transmitted  or be
 * reflected on the boundary perpendicular
 * to the x-direction.
 *
 * If the photon is transmitted, move the photon to the
 * surrounded glass layer.
 *
 * Update the photon parmameters.
```

```
 ****/
void CrossXOrNot(InputStruct  *       In_Ptr,
                                PhotonStruct *       Photon_Ptr,
                                OutStruct *          Out_Ptr)
{
  double ux = Photon_Ptr->ux; /* x directional cosine. */
  double ux1;  /* cosines of transmission alpha. */
  double r=0.0;        /* reflectance. */
  short  layer = Photon_Ptr->layer;
  double ni = In_Ptr->layerspecs[layer].n;
  double nt = In_Ptr->layerspecs[In_Ptr->num_layers+2].n;

  /* Get r. */
  if( fabs(ux) <= In_Ptr->layerspecs[layer].cos_critxy)
    r=1.0;             /* total internal reflection. */
  else r = RFresnel(In_Ptr->layerspecs[layer].n,
           In_Ptr->layerspecs[In_Ptr->num_layers+2].n, fabs(ux), &ux1);


  if(RandomNum() > r) {              /* transmitted to surrounding glass. */
    Photon_Ptr->layer = In_Ptr->num_layers+2;
    Photon_Ptr->ux = ux1*SIGN(ux);
    Photon_Ptr->uy *= ni/nt;
    Photon_Ptr->uz *= ni/nt;
    }
  else                                        /* reflected. */
    Photon_Ptr->ux = -ux;
}

/*************************************************************
 * Decide whether the photon will be transmitted  or be
 * reflected on the boundary perpendicular
 * to the y-direction.
 *
 * If the photon is transmitted, move the photon to the
 * surrounded glass layer.
 *
 * Update the photon parmameters.
 ****/
void CrossYOrNot(InputStruct  *       In_Ptr,
                                PhotonStruct *       Photon_Ptr,
                                OutStruct *          Out_Ptr)
{
  double uy = Photon_Ptr->uy; /* x directional cosine. */
  double uy1;  /* cosines of transmission alpha. */
  double r=0.0;        /* reflectance. */
  short  layer = Photon_Ptr->layer;
  double ni = In_Ptr->layerspecs[layer].n;
  double nt = In_Ptr->layerspecs[In_Ptr->num_layers+2].n;

  /* Get r. */
  if( fabs(uy) <= In_Ptr->layerspecs[layer].cos_critxy)
    r=1.0;             /* total internal reflection. */
  else r = RFresnel(In_Ptr->layerspecs[layer].n,
           In_Ptr->layerspecs[In_Ptr->num_layers+2].n, fabs(uy), &uy1);


  if(RandomNum() > r) {              /* transmitted to surrounding glass. */
    Photon_Ptr->layer = In_Ptr->num_layers+2;
    Photon_Ptr->uy = uy1*SIGN(uy);
    Photon_Ptr->ux *= ni/nt;
    Photon_Ptr->uz *= ni/nt;
    }
  else                                        /* reflected. */
    Photon_Ptr->uy = -uy;
}
/*************************************************************
 ****/
void CrossOrNot(InputStruct  *       In_Ptr,
                                PhotonStruct * Photon_Ptr,
                                OutStruct    * Out_Ptr,
                                short          hitcase)
{
  if (hitcase == 1)
    CrossXOrNot(In_Ptr, Photon_Ptr, Out_Ptr);
  else if (hitcase == 2)
    CrossYOrNot(In_Ptr, Photon_Ptr, Out_Ptr);
  else if (hitcase == 3) {
    if(Photon_Ptr->uz < 0.0)
      CrossUpOrNot(In_Ptr, Photon_Ptr, Out_Ptr);
    else
      CrossDnOrNot(In_Ptr, Photon_Ptr, Out_Ptr);
  }
}

/*************************************************************
 *      Move the photon packet in glass layer.
 *      Horizontal photons are killed because they will
 *      never interact with tissue again.
 ****/
void HopInGlass(InputStruct  * In_Ptr,
                                PhotonStruct * Photon_Ptr,
                                OutStruct    * Out_Ptr)
{
  short  hitcase; /* the boundary which is hit (1,2 or 3). */

  if(Photon_Ptr->uz == 0.0)
    /* horizontal photon in glass is killed. */
    Photon_Ptr->dead = 1;
```

```
  if(Photon_Ptr->layer == In_Ptr->num_layers + 2) {
        /* surrounding glass. */
    StepSizeInSurrGlass(Photon_Ptr, In_Ptr);
    hitcase = 3;        /* always in surrounding glass. */
    Hop(Photon_Ptr);
    CrossOrNot(In_Ptr, Photon_Ptr, Out_Ptr, hitcase);
  }
  else {  /* glass layer which is not the surrounding glass layer. */
    hitcase = StepSizeInGlass(Photon_Ptr, In_Ptr);
    Hop(Photon_Ptr);
    CrossOrNot(In_Ptr, Photon_Ptr, Out_Ptr, hitcase);
  }
}

/*************************************************************
 *      Set a step size, move the photon, drop some weight,
 *      choose a new photon direction for propagation.
 *
 *      When a step size is long enough for the photon to
 *      hit an interface, this step is divided into two steps.
 *      First, move the photon to the boundary free of
 *      absorption or scattering, then decide whether the
 *      photon is reflected or transmitted.
 *      Then move the photon in the current or transmission
 *      medium with the unfinished stepsize to interaction
 *      site.  If the unfinished stepsize is still too long,
 *      repeat the above process.
 ****/
void HopDropSpinInTissue(InputStruct  *  In_Ptr,
                                         PhotonStruct *  Photon_Ptr,
                                         OutStruct    *  Out_Ptr)
{
  short hitcase; /* the boundary which is hit (1,2,3).  */
                 /* hitcase = 0 if no boundary is hit. */

  StepSizeInTissue(Photon_Ptr, In_Ptr);

  hitcase = HitBoundary(Photon_Ptr, In_Ptr);

  if(hitcase>0) {
    Hop(Photon_Ptr);   /* move to boundary plane. */
    CrossOrNot(In_Ptr, Photon_Ptr, Out_Ptr, hitcase);
  }
  else {
    Hop(Photon_Ptr);
    Drop(In_Ptr, Photon_Ptr, Out_Ptr);
    Spin(In_Ptr, Photon_Ptr);
  }
}

/*************************************************************
 ****/
void HopDropSpin(InputStruct  *  In_Ptr,
                                 PhotonStruct *  Photon_Ptr,
                                 OutStruct    *  Out_Ptr)
{
  short layer = Photon_Ptr->layer;

  if(In_Ptr->layerspecs[layer].meanstep == 0.0)
       /* glass layer. */
    HopInGlass(In_Ptr, Photon_Ptr, Out_Ptr);
  else
    HopDropSpinInTissue(In_Ptr, Photon_Ptr, Out_Ptr);

  if( Photon_Ptr->w < In_Ptr->Wth && !Photon_Ptr->dead)
    Roulette(Photon_Ptr);
}
```

B.2. List of specific modifications

B.2.1. Modifications from MCML to MCMLBOUN

The input file:

- The lateral extension of the plane layers of the sample, x_size and y_size, are defined on a row below the row defining the variables nz, nr and na.
- The refractive index of the surrounding glass is defined on a row next to the rows containing the parameters of the different layers and ambients.

mcml.h:

- A variable cos_critxy is declared (data type: double) in the structure of Layerstruct.
- The two variables x_size and y_size are declared (data type: double) in the structure of Inputstruct.

mcmlio.c:

- The function ShowVersion shows a modified version.
- A new function ReadXSizeYSize, reading the variables x_size and y_size, is written after the function ReadNzNrNa.
- An additional function ReadSurrGlass, reading and setting the parameters of the surrounding glass, is included after the function ReadOneLayer.
- The number of elements in the array named Layerspecs is increased by one due to the surrounding glass. The argument of the malloc commando in the function ReadLayerSpecs is thus changed.
- The function ReadLayerSpecs calls the function ReadSurrGlass. The surrounding glass is thus defined as the layer with index (num_layers + 2).
- The calculation of the critical angles for total internal reflection for the different layers relative the surrounding glass (cos_critxy) are added in the function CriticalAngle as well as the critical angles for the surrounding glass relative the ambient.
- The function ReadXSizeYSize is called after the function ReadNzNrNa from the function ReadParm.
- A modification of the function WriteInParm, arranges the new input parameters in the output file.

mcmlgo.c:

- The data type of the function StepSizeInGlass is changed to short. The function returns the values (1, 2, 3) due to the hit boundary (perpendicular to x-, y- or z-axis). The implementation of the path length to the boundary is not modified.
- The function HitBoundary is modified in the same way, but when no boundary was hit, the value returned is zero.
- In order to handle photon packages propagating in the surrounding glass, a function called StepSizeInSurrGlass has been written.
- The two functions CrossUpOrNot and CrossDnOrNot are modified to work in the surrounding glass too. Photon packages which are totally reflected in the surrounding glass are killed in CrossUpOrNot.
- Two new functions, similar to the two previous mentioned, check if the photon packages are reflected or transmitted when hitting the boundary to the surrounding

glass. This functions are called CrossXOrNot and CrossYOrNot and are introduced directly after CrossDnOrNot.
- A new argument, hitcase (data type: short), is declared in the function CrossOrNot. If hitcase equals 1 or 2, CrossXOrNot or CrossYOrNot is called respectively. If hitcase equals 3, the function continues as in the original version.
- The function HopInGlass calls StepSizeInGlass or StepSizeInSurrGlass depending on the position of the photon package. When the function CrossOrNot is called, the value of the variable hitcase obtained from the function StepSizeInGlass, is included in the argument.
- The new variable hitcase is used in a similar way in the function HopDropSpinInTissue, but its value is obtained from the function HitBoundary. The function CrossOrNot is called when hitcase differs from zero.

mcmlmain.c:

- The function ShowVersion is declared in the main program in order to avoid compiler warnings.

B.2.2. Modifications from MCMLBOUN to TMMCML

The input file:

- The name of the file with the data base, and the values of the variables (datatype: short) nbrangint, nbrthsc and nbrpssc are written before the normal input parameters of each run. The three variables represent the number of incident angles (θ_i), scattered deflection angles (θ_s) and scattered azimuthal angles (φ_s), respectively, uniformly distributed. These values must agree with those connected to the data base, to obtain the right result.
- The mean distance between the scattering objects (meanstep), the standard deviation of this value (meanstepdev), the normalized absorption cross section (abscross), the direction of the symmetry axis of the scattering objects (bcux, bcuy, bcuz) and the standard deviation of this direction (dev) are written on a new row for each layer. The length unit is cm.

mcml.h:

- The parameters belonging to the data base (dat_fname, nbrangint, nbrthsc, nbrpssc and valmatrix) are declared in the structure of InputStruct. The array dat_fname (data type: char) contains the name of the data base file. The two-dimensional array valmatrix (data type: double) is used to store the values from the data base.
- The parameters of each layer (datatype: double) are declared in the structure of LayerStruct, replacing of the old ones (mua, mus and g).

mcmlio.c:

- The function ShowVersion shows a modified version.
- The new parameters are read instead of the old ones in the function ReadOneLayer and are written to the output file by the modified function WriteInParm.
- The function ReadSurrGlass assigns meanstep = 0 instead of $\mu_a = \mu_s = 0$.
- Two new functions, ReadDataBase and PlaceDataInMatrix, allocate memory space for a two-dimensional array (matrix), read the data base values and put them in the array.

- In the function CheckParm a commando FindDataLine is added to prevent the data base to be read twice. This saves time but the data base parameters will not be checked before the program starts.
- To free the allocated memory used to the data base values, another FreeMatrix commando is added in the function FreeData.

mcmlgo.c:

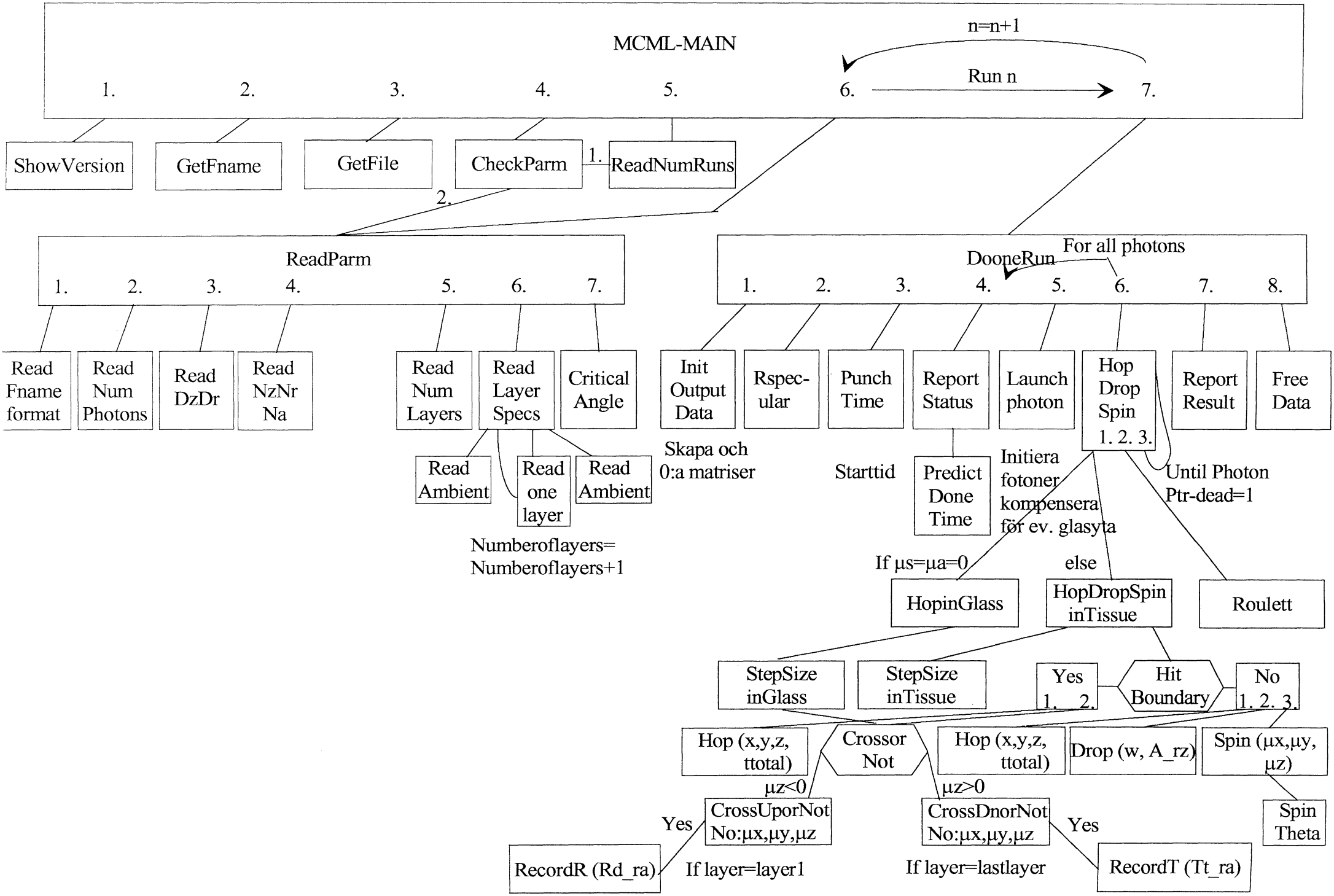
- The functions Rspecular, LaunchPhoton and HopDropSpin check if meanstep = 0 instead of $\mu_a = \mu_s = 0$.
- By using the variables meanstep and meanstepdev and the new function StepGaussDev, the function StepSizeInTissue evaluates a step length of the photon package instead of the old expression using μ_a and μ_s. In this function and in the function HitBoundary a variable scale is defined, corresponding to the variable μ_t in the original program when calculating the value of the variable sleft.
- The function Drop decreases the weight of the photon package by a factor abscross instead of the quotient μ_a/μ_t.
- The function Spin is completely changed. The function SpinTheta is removed and the new function SpinGaussDev handles the Gaussian distributed directions of the scattering objects.
- The variable g in the argument of Spin is replaced by the structure variable In_Ptr. This modification is also made in the call from the function HopDropSpinInTissue.

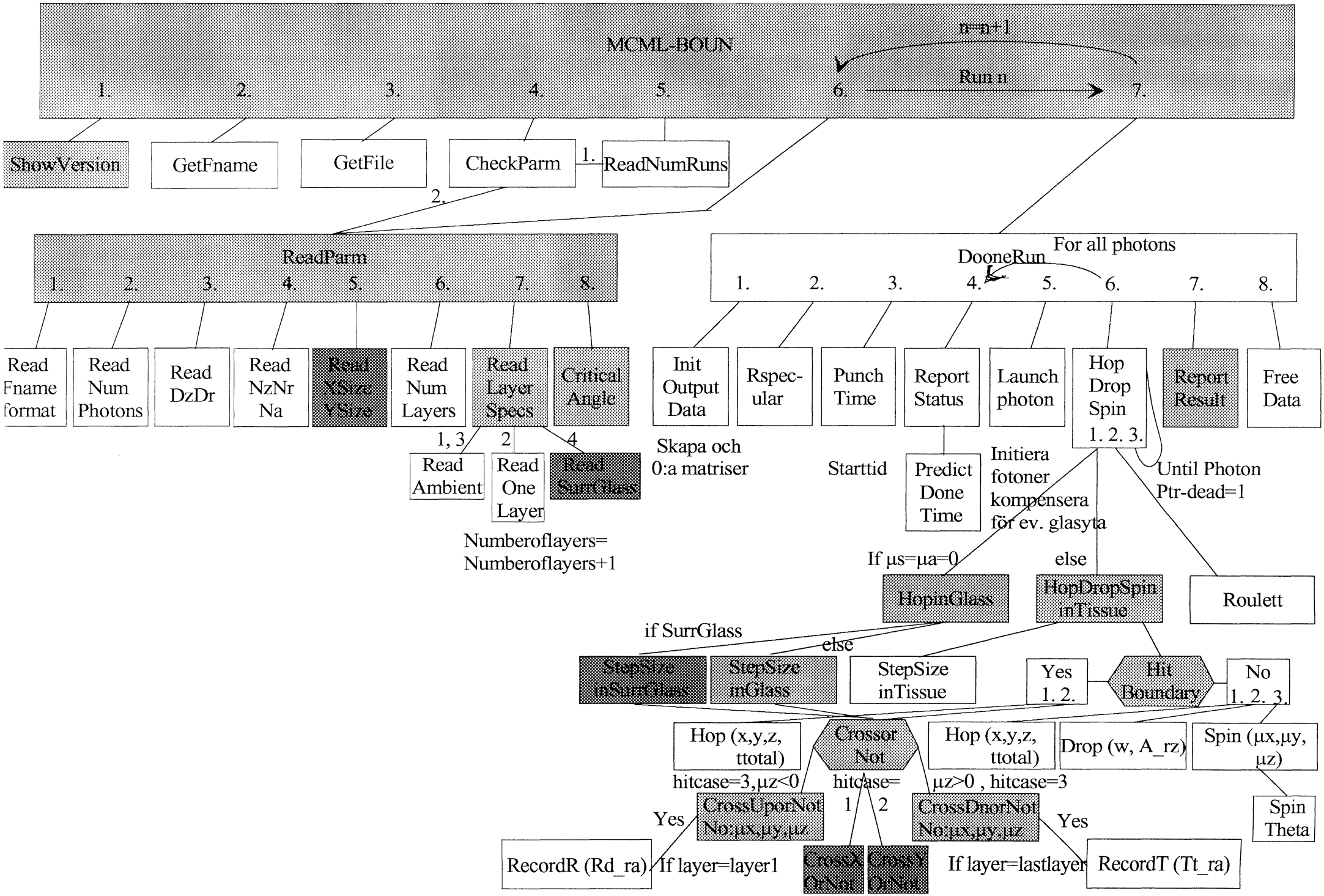
mcmlmain.c:

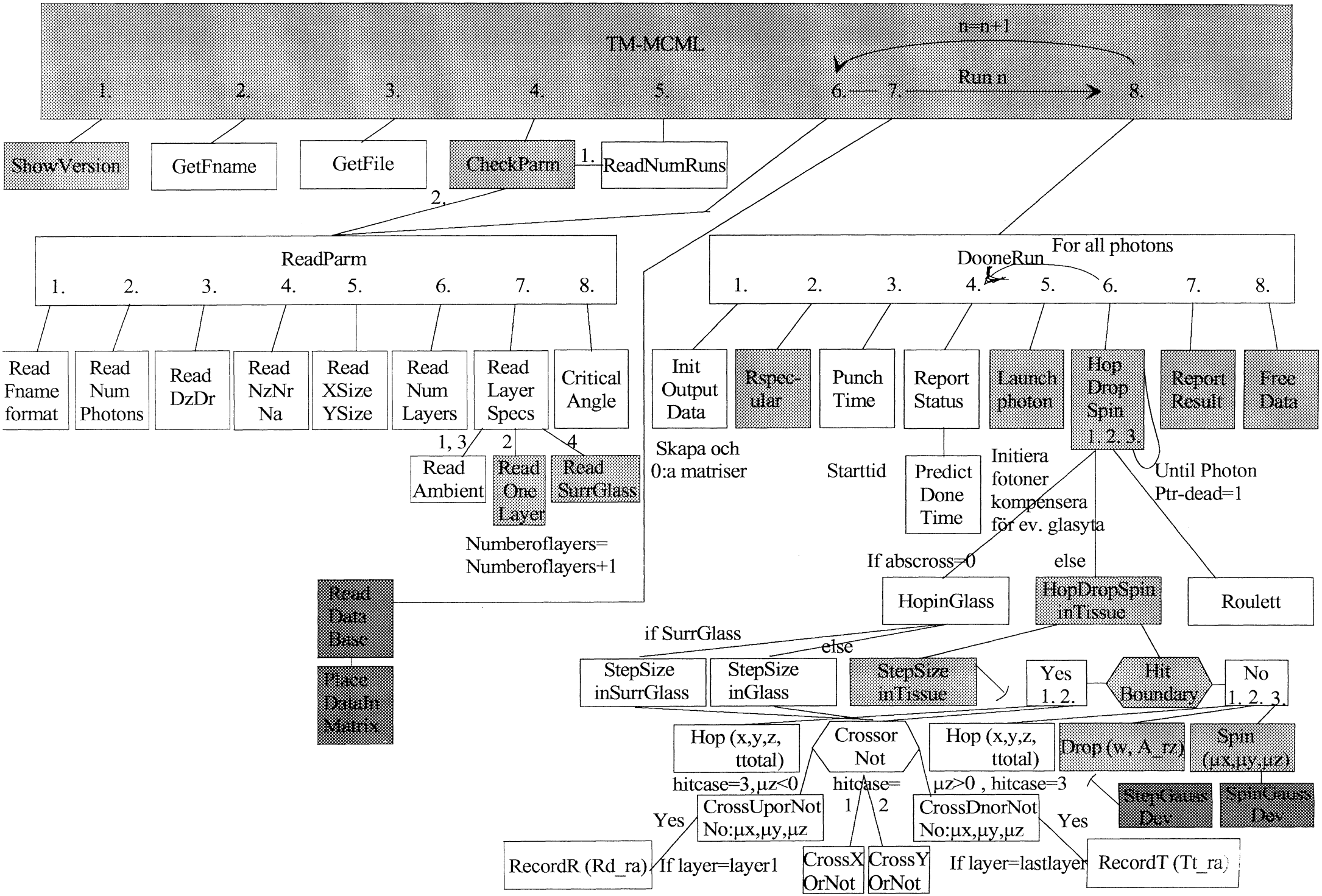
- The function ReadDataBase is declared and called.

B.3. Flow charts

Flow charts of the Monte Carlo programs are presented on the following three pages. The chart of the original Monte Carlo program is first shown, followed by the two modified versions MCMLBOUN and TMMCML. Squares symbolizing new routines in the modified versions are filled and squares for modified routines are shaded.


MCML-MAIN
n=n+1
1.
2.
3.
4.
5.
6.
Run n
7.
ShowVersion
GetFname
GetFile
CheckParm
1.
ReadNumRuns
2.
ReadParm
1.
2.
3.
4.
5.
6.
7.
Read Fname format
Read Num Photons
Read DzDr
Read NzNr Na
Read Num Layers
Read Layer Specs
Critical Angle
Read Ambient
Read one layer
Read Ambient
Numberoflayers= Numberoflayers+1
DooneRun
For all photons
1.
2.
3.
4.
5.
6.
7.
8.
Init Output Data
Rspec-ular
Punch Time
Report Status
Launch photon
Hop Drop Spin
1. 2. 3.
Report Result
Free Data
Skapa och 0:a matriser
Starttid
Predict Done Time
Initiera fotoner kompensera för ev. glasyta
Until Photon Ptr-dead=1
If μs=μa=0
else
HopinGlass
HopDropSpin inTissue
Roulett
StepSize inGlass
StepSize inTissue
Yes
1. 2.
Hit Boundary
No
1. 2. 3.
Hop (x,y,z, ttotal)
Crossor Not
Hop (x,y,z, ttotal)
Drop (w, A_rz)
Spin (μx,μy, μz)
μz<0
μz>0
Yes
CrossUporNot No:μx,μy,μz
CrossDnorNot No:μx,μy,μz
Yes
Spin Theta
RecordR (Rd_ra)
If layer=layer1
If layer=lastlayer
RecordT (Tt_ra)


MCML-BOUN
1.
2.
3.
4.
5.
6.
7.
n=n+1
Run n
ShowVersion
GetFname
GetFile
CheckParm
1.
ReadNumRuns
2.
ReadParm
1.
2.
3.
4.
5.
6.
7.
8.
Read Fname format
Read Num Photons
Read DzDr
Read NzNr Na
Read XSize YSize
Read Num Layers
Read Layer Specs
Critical Angle
1, 3
2
4
Read Ambient
Read One Layer
Read SurrGlass
Numberoflayers= Numberoflayers+1
For all photons
DooneRun
1.
2.
3.
4.
5.
6.
7.
8.
Init Output Data
Rspec-ular
Punch Time
Report Status
Launch photon
Hop Drop Spin 1. 2. 3.
Report Result
Free Data
Skapa och 0:a matriser
Starttid
Predict Done Time
Initiera fotoner kompensera för ev. glasyta
Until Photon Ptr-dead=1
If μs=μa=0
else
HopinGlass
HopDropSpin inTissue
Roulett
if SurrGlass
else
StepSize inSurrGlass
StepSize inGlass
StepSize inTissue
Yes 1. 2.
Hit Boundary
No 1. 2. 3.
Hop (x,y,z, ttotal)
Crossor Not
Hop (x,y,z, ttotal)
Drop (w, A_rz)
Spin (μx,μy, μz)
hitcase=3,μz<0
hitcase= 1 2
μz>0 , hitcase=3
Yes
CrossUporNot No:μx,μy,μz
CrossDnorNot No:μx,μy,μz
Yes
Spin Theta
RecordR (Rd_ra)
If layer=layer1
CrossX OrNot
CrossY OrNot
If layer=lastlayer
RecordT (Tt_ra)


TM-MCML
n=n+1
1.
2.
3.
4.
5.
6.
7.
Run n
8.
ShowVersion
GetFname
GetFile
CheckParm
1.
ReadNumRuns
2.
ReadParm
1.
2.
3.
4.
5.
6.
7.
8.
Read Fname format
Read Num Photons
Read DzDr
Read NzNr Na
Read XSize YSize
Read Num Layers
Read Layer Specs
Critical Angle
1, 3
2
4
Read Ambient
Read One Layer
Read SurrGlass
Numberoflayers= Numberoflayers+1
Read Data Base
Place Datain Matrix
For all photons
DooneRun
1.
2.
3.
4.
5.
6.
7.
8.
Init Output Data
Rspec-ular
Punch Time
Report Status
Launch photon
Hop Drop Spin 1. 2. 3.
Report Result
Free Data
Skapa och 0:a matriser
Starttid
Predict Done Time
Initiera fotoner kompensera för ev. glasyta
Until Photon Ptr-dead=1
If abscross=0
else
HopinGlass
HopDropSpin inTissue
Roulett
if SurrGlass
else
StepSize inSurrGlass
StepSize inGlass
StepSize inTissue
Yes 1. 2.
Hit Boundary
No 1. 2. 3.
Hop (x,y,z, ttotal)
Crossor Not
Hop (x,y,z, ttotal)
Drop (w, A_rz)
Spin (μx,μy,μz)
hitcase=3,μz<0
hitcase= 1
2
μz>0 , hitcase=3
StepGauss Dev
SpinGauss Dev
Yes
CrossUporNot No:μx,μy,μz
CrossDnorNot No:μx,μy,μz
Yes
RecordR (Rd_ra)
If layer=layer1
CrossX OrNot
CrossY OrNot
If layer=lastlayer
RecordT (Tt_ra)

APPENDIX C: How to run the programs

A description of running the programs will be given in order to prevent mistakes and wrong solutions. The characteristics of the data base MC.DAT will be discussed in detail in order to make it easier for other users to do further modifications. An extensive description of the modifications of the Monte Carlo programs is made in appendix B.2. This appendix will present only examples of input files of these programs.

C.1. EXTENDT1

The input parameters are the same as in the original T1 program (Chapter 3.1.1.).

1. Choose values for nrank and ntheta. The experience from this work is to choose nrank slightly less than twice the size parameter of an equal volume sphere and ntheta = 2*nrank.
2. Check convergence over nrank (case 1), ntheta (case 0).
3. If convergence was not obtained, restart from 1.
4. Check convergence over m (case 2), save the T-matrix (alternative 1).
5. If convergence was obtained, the T-matrix is stored in file t.

C.2. EXTENDT3

The EXTENDT3 program was in this work only used for controls and to plot three-dimensional pictures. The input and output parameters are exactly the same as in program T3, described in chapter 3.1.3.

C.3. MCDATA

1. Write the input parameters. The variables nbrangin, nbrthsc and nbrphsc are previously described (section 3.1.4). The variable anglab represents the polarization direction and has hardly any influence on the angular scattering intensities, when the difference between the refractive indices of the particle and the surrounding medium is much less than unity[30].
2. The differential scattering cross sections are computed in all directions. The scattering angles θ and φ vary between 0° and 180° with an interval of $d\theta = 180°/(nbrthsc-1)$ and $d\varphi = 180°/(nbrphsc-1)$, respectively. When θ equals 0° or 180°, the cross section only need to be evaluated for one azimuthal angle (φ = 0°). As these variables are discrete, one direction is assumed to represent all the directions closed by. Each direction is assigned an area corresponding to the surface area in that direction interval of a surrounding sphere. Therefore, the differential scattering cross sections are normalized by a factor:

$$\frac{d\varphi\ (\cos(\theta - \frac{d\theta}{2}) - \cos(\theta + \frac{d\theta}{2}))}{2\pi} \cdot \frac{1}{1-\cos(\frac{d\theta}{2})} = \frac{\text{area of this direction}}{\text{area of forward direction}}$$

All values representing directions when φ differ from 0° and 180° are multiplied by a factor of two, as they represent both φ and 360°-φ. This implementation, according to symmetry properties is made to save time and memory allocation.
3. The modified differential cross sections are then multiplied by a new normalization factor, in order to obtain a probability function, where the sum of all variables equals one.
4. Step 2. and 3. are repeated for every new incident angle.

5. The probability data are stored in MC.DAT using a specific algorithm:

```
for angin = 1 to nbrangin   (θi = 0 to 90°)
   sum = probability of back scattering
   write sum
   for theta = 2 to nbrthsc-1 (θs = dθ to 180°-dθ, dθ = 180°/(nbrthsc-1))
     for phi = 1 to nbrphsc  (φs = 0 to 180°)
       sum = sum + probability of scattering in this direction
       write sum
     end
   end
   the probability of forward scattering (θs = 0) is thus 1 - sum
end
```

C.4. MCMLBOUN

Input file of MCMLBOUN:

```
####
# This is the input file (bountest.mci) for testing the mcmlboun
# using fixed rnd seed.  6/25/96.
####

1.0                                    # file version
1                                      # number of runs

### Specify data for run 1
tmtest1.mco   A                        # output filename, ASCII/Binary
1000                                   # No. of photons
1       0.1                            # dz, dr
7       75      30                     # No. of dz, dr & da.
1000    1000                           # x_size and y_size

3                                              # No. of layers
# n     mua     mus     g       d            # One line for each layer
1.0                                            # n for medium above.
1.5     0       0       0       1            # glass
1.345   1       249     0.99    5            # layer 1
1.5     0       0       0       1            # glass
1.0                                            # n for medium below.
1.5                                            # n for surrounding glass
```

C.5. TMMCML

The most important thing to remember, running this program, is to write the same values for the number of different angles (nbrangin, nbrthsc, nbrpssc) in the input file as those used when obtaining the data base MC.DAT. Assign meanstep = 0 when a glass layer or another non-absorbing and non-scattering layer is required.

Input file of TMMCML:

```
####
#  This is the input file (tmtest.mci) for testing the mcml
#  using fixed rnd seed.  6/25/96.
####

1.0                                          # file version
1                                            # number of runs

### Specify data for run 1
mc.dat          1                            # data base file name, nbrangin
181     37                                   # nbrthsc, nbrphsc
tmtest1.mco     A                            # output filename, ASCII/Binary
100                                          # No. of photons
1       1                                    # dz, dr
8       75      30                           # No. of dz, dr & da.
5000    5000                                 # x_size and y_size

3                                                       # No. of layers
# n     mstep   dev     abs     bcux    bcuy    bcuz    dev     d  # One line for each layer
1.0                                                     # n for medium above.
1.5     0       0       0       0       0       0       0       1  # glass
1.345   0.1     0.02    0.004   1       1       0       0.2     10 # layer 1
1.5     0       0       0       0       0       0       0       1  # glass
1.0                                                     # n for medium below.
1.5                                                     # n for surrounding glass
```

APPENDIX D: Special functions

When physical problems related to wave propagation and scattering theory at spherical symmetry are to be solved mathematically, some complicated functions seem to turn up very often. Some of the functions, which have got their own names, are used in this work and a short mathematical description will be given. More detailed information and theory are given by Arfken[35].

D.1. Bessel and Hankel functions

The Bessel function $j_n(z)$ is the solution of the equation

$$z^2 \frac{d^2}{dz^2} f_n(z) + 2z \frac{d}{dz} f_n(z) + (z^2 - n(n+1)) f_n(z) = 0 \qquad (D.1)$$

which is finite in origo.
The Hankel functions of the first and second kind are described by

$$h_n^{(1)}(z) = j_n(z) + i n_n(z) \qquad (D.2a)$$

$$h_n^{(2)}(z) = j_n(z) - i n_n(z) \qquad (D.2b)$$

where the Neumann function $n_n(z)$ is the solution of the differential equation (D.1) which is infinite in origo.
The Bessel function can be expressed as

$$j_n(z) = 2^n z^n \sum_{k=0}^{\infty} \frac{(-1)^k (k+n)!}{k!(2k+2n+1)!} z^{2k} \qquad (D.3)$$

and has the property

$$j_n(-z) = (-1)^n j_n(z) \qquad (D.4)$$

The expression of the Neumann function is similar and the two functions can be represented in easier asymptotic expressions when the arguments are small.

For n = 0 the functions are described by

$$j_0(z) = \frac{\sin z}{z} \quad ; \quad n_0(z) = -\frac{\cos z}{z} \quad \text{and thus} \quad h_0^{(1)} = -\frac{i}{z} e^{iz} \qquad (D.5\ a,b,c)$$

D.2. Associated Legendre functions

The associated Legendre functions $P_n^m(\cos\theta)$ are parts of the spherical harmonic functions $Y_n^m(\theta,\phi)$ according to the relation

$$Y_n^m(\theta,\phi) = c_{nm} P_n^m(\cos\theta) e^{im\phi} \qquad (D.6)$$

where $P_n^m(\cos\theta)$ is the solution of the differential equation

$$\frac{1}{\sin\theta}\frac{d}{d\theta}(\sin\theta\frac{d}{d\theta}(P_n^m(\cos\theta)))+(n(n+1)-\frac{m^2}{\sin^2\theta})P_n^m(\cos\theta)=0 \qquad (D.7)$$

which results in the following relations:

$$P_n^m(\cos\theta)=\sin^m\theta\frac{d^m}{d(\cos\theta)^m}P_n^0(\cos\theta) \quad \theta\in[0,\pi] \qquad (D.8)$$

$$P_n^m(\cos\theta)=0 \quad \text{when } |m|>1 \qquad (D.9)$$

$$P_n^m(-\cos\theta)=(-1)^{n+m}P_n^m(\cos\theta) \qquad (D.10)$$

$$P_n^m(\theta=0)=\delta_{m0} \qquad (D.11)$$

Two examples of associated Legendre functions of lower orders are expressed as

$$P_1^1(\cos\theta)=\sqrt{1-\cos^2\theta}=\sin\theta \qquad (D.12a)$$

$$P_4^3(\cos\theta)=105\cos\theta(1-\cos^2\theta)^{3/2}=105\cos\theta\sin^3\theta \qquad (D.12b)$$