

Chapter 6

Monte Carlo simulations of fluorescence in turbid media.

Steven L. Jacques

Oregon Medical Laser Center, Providence St. Vincent Medical Center

Oregon Health and Science University

Portland, Oregon

Table of Contents

1. Introduction
2. The general solution
3. The Monte Carlo solution
4. Example simulations
5. The program `mcfluor.c`
 - 5.1 Header
 - 5.2 `main()`
 - 5.3 Excitation
 - 5.4 Background fluorescence
 - 5.5 Fluorescent heterogeneity
6. The subroutines
 - 6.1 `mcsub()`
 - 6.2 `Rfresnel()`
 - 6.3 `SaveFiles()`
 - 6.4 `RandomNumber()`
 - 6.5 Memory allocation routines
7. Listing of `mcfluor.c`

1. Introduction

This chapter presents a Monte Carlo program written in ANSI Standard C that simulates the penetration of excitation light into a light-scattering medium such as biological tissue, and the generation and escape of fluorescence due to fluorophore distributed in the tissue.

The program uses a simple steady-state Monte Carlo subroutine, `mcsub()`, that launches photons as either (1) a collimated beam of variable beam radius, (2) a focused Gaussian beam with variable beam radius, focal depth, and focal waist radius, or (3) an isotropic point source at a chosen depth. The results returned by the subroutine are the concentration of light within the tissue and the escaping light at the tissue surface.

The concentration of light within the tissue is reported as the “fluence rate” $F(z,r)$ in units of $[W/cm^2]$ per W of incident power, which equals the energy density of light in a local volume, $[J/cm^3]$, times the speed of light, $[cm/s]$. The escaping light is reported as the “flux density” $J(r)$ in units of $[W/cm^2]$ per W of incident power, which equals the power per unit surface area escaping the medium across a local area of the air/medium surface. The parameters z and r indicate depth and radial position $[cm]$, respectively.

The program uses `mcsub()` to solve three aspects of the problem: (1) the penetration of excitation light throughout the tissue, (2) the generation and escape of fluorescence due to a uniform background fluorophore concentration, and (3) the generation and escape of fluorescence due to a localized heterogeneity of extra fluorophore concentration. The escaping flux density and fluence rate of excitation light are denoted by J_x and F_x . The escaping flux density and fluence rate of fluorescent emission light are denoted by J_f and F_f .

The history of this Monte Carlo simulation began with a Monte Carlo simulation written by the author in 1984 that implemented the Henyey-Greenstein scattering function and a mismatched air/tissue boundary. The program was improved by incorporating the step size selection based on a random number as used by Wilson and Adam 1983 [1] who first applied a Monte Carlo method to light transport in biological tissues using isotropic scattering. Keijzer et al. 1989 [2] improved on the tracking of photon trajectories during simulations of photon transport in tissue and transformed the program close to its present form. Keijzer et al. 1989 [3] applied the Monte Carlo program to simulating the generation and escape of tissue autofluorescence. Drawing from the work of Witt 1977 [4], Prahl et al. 1989 [5] translated Keijzer’s work from cylindrical coordinates to Cartesian coordinates to yield simpler code. Prahl et al. implemented the selection of photon deflection angles using random numbers for Henyey-Greenstein scattering. Prahl et al. brought the program to its present form. Wang et al. (1992) [6] and (1995) [7] converted the work of Prahl et al. into modularized ANSI Standard C code for multilayered tissues called MCML that has been widely promulgated. The MCML program by Wang et al. stimulated broad use of Monte Carlo simulations in the biomedical optics community. Jacques (1998) [8] reduced the MCML code to its simplest form, called `mc321.c`, to provide a simulation of light propagation from point, line, and planar sources of light. For this chapter, the `mc321.c` code was converted to a subroutine called `mcsub()` that provides for three dimensional photon propagation from a collimated beam, a focused Gaussian beam, or an isotropic point source with a surface boundary with mismatch refractive indices such as an air/tissue surface. The Gaussian launch method was developed by Gareau et al. (2001) [9].

In summary, this chapter illustrates the use of the Monte Carlo method to model the generation and escape of fluorescence within a tissue with both homogeneous and heterogeneous fluorophore concentrations. Listings of the controlling program `mcfluor.c` and the subroutine `mcsub()` are provided.

Further reading on Monte Carlo simulations can be found at <http://omlc.orgi.edu/classroom/ece532/class4/index.html>. Monte Carlo software can be downloaded from <http://omlc.orgi.edu/software/mc/index.html>, including a listing of the `mcfluor()` program of this chapter.

2. The general solution

Consider a simple problem. A narrow beam of collimated light irradiates a tissue orthogonal to the tissue surface. The light penetrates the tissue and experiences multiple scattering events that deflect the photons. Each photon initially has a trajectory pointing down into the tissue, but after numerous scattering events the photon's trajectory becomes randomized. All the photons experience this transition from an initially directed launching to an eventually randomized trajectory. Once all photon trajectories are randomized, the net flow of light is directed along gradients in photon concentration, in which case diffusion theory is useful for modeling the light distribution. However, the Monte Carlo method is especially useful in modeling the initial transition from directed to randomized trajectories. Much of the fluorescence generation occurs during this transition. Hence, Monte Carlo simulation is an attractive method for simulating generation and escape of fluorescence in a tissue.

A generic mathematical description of the problem describes the escaping flux density of fluorescence, J_f [W/cm²], at a position \underline{r} on the tissue surface:

$$J_f(\underline{r}) = P_o \int_{\text{volume}} T_x(\underline{r}') \epsilon C(\underline{r}') Y T_f(\underline{r}, \underline{r}') dV(\underline{r}') \quad (1)$$

where

\underline{r}'	[cm]	is a vector that specifies the position of the incremental volume $dV(\underline{r}')$ within the tissue,
$dV(\underline{r}')$	[cm ³]	is the incremental volume at \underline{r}' of the volume integration,
P_o	[W]	is the incident power of the excitation beam,
$T_x(\underline{r}')$	[cm ⁻²]	is the transport of source power at the excitation wavelength such that $P_o T_x(\underline{r}')$ yields the local fluence rate at \underline{r}' ,
ϵ	[cm ⁻¹ M ⁻¹]	is the extinction coefficient of the fluorophore (using base e),
$C(\underline{r}')$	[M]	is the concentration of fluorophore at \underline{r}' ,
Y	[W/W]	is the power yield, W fluorescence per W excitation absorbed by fluorophore,
$T_f(\underline{r}, \underline{r}')$	[cm ⁻²]	is the transport of fluorescence power from \underline{r}' to yield the escaping energy density at the tissue surface at position \underline{r} ,
$J_f(\underline{r})$	[W/cm ²]	is the flux density of escaping fluorescence at the surface at position \underline{r} .

The transport factor T_x accounts for how the beam is launched into the tissue, eg. as a collimated beam or a focused Gaussian beam, including consideration of any specular reflectance at the air/tissue surface, and how that excitation light spreads throughout the tissue. The lumped factor ϵC is the absorption coefficient of the fluorophore at the excitation wavelength, defined using base e such that transmission T

through a pathlength L is given as $T = \exp(-\epsilon CL)$. Note that the literature usually reports the extinction coefficient using base 10 such that $T = 10^{-\epsilon CL}$, so the ϵ of this chapter equals the literature's ϵ times $\ln(10)$, i.e. 2.3-fold larger. The factor Y is the power yield of W fluorescence per W excitation absorbed by fluorophore. The incremental volume of the integration, dV , has units of cm^3 . The product $P_o T_x \epsilon C Y dV$ has units of power [W] and serves as an isotropic point source of fluorescent power. The transport factor T_f accounts for how this fluorescent power distributes throughout the tissue and escapes at the surface, including consideration of total internal reflectance at the air/tissue surface. The final result is the escaping flux density, J_f [W/cm^2].

Similarly, the fluence rate F_f [W/cm^2] at a position \underline{r} within the tissue is expressed:

$$F_f(\underline{r}) = P_o \int_{\text{volume}} T_x(\underline{r}') \epsilon C(\underline{r}') Y T_f(\underline{r}, \underline{r}') dV(\underline{r}') \quad (2)$$

where in this case $T_f(\underline{r}, \underline{r}')$ indicates the transport of local fluorescent power generated at position \underline{r}' to an observation position \underline{r} within the medium to yield the fluence rate $F_f(\underline{r})$ [W/cm^2]. The only difference between equations (1) and (2) is that the former calculates the escape of photons out the surface while the latter calculates the concentration of photons at a position within the tissue. Because the units of J_f and F_f are the same, the equations look identical. Only the factor $T_f(\underline{r}, \underline{r}')$ differs between the two equations.

3. The Monte Carlo solution

To implement equations 1 and 2 using Monte Carlo simulations, the program `mcfluor.c` uses a Monte Carlo subroutine called `mcsub()`. In the first step, `mcfluor.c` uses `mcsub()` to launch and propagate excitation photons into a medium with the optical properties of the excitation wavelength yielding the net statistical result for the transport factor $T_x(\underline{r}')$ [cm^2]. The program does not specify P_o but rather refers to P_o as the “incident power”. Therefore, `mcsub()` returns the factor $T_x(\underline{r}')$ that can be called the fluence rate of excitation per W of incident power, $F_x(\underline{r}')$ [W/cm^2 per W of incident power] or [$\text{W}/\text{cm}^2/\text{W}$]. $F_x(\underline{r}')$ is the distributed excitation light that will excite fluorescence.

Next, `mcfluor.c` considers the fluorescence that is generated from a uniform background of fluorophore concentration throughout the medium. The simulation uses `mcsub()` to launch fluorescence photons into a medium with the optical properties appropriate for the fluorescence emission wavelength. The photons are launched as an isotropic point source of fluorescence from the position \underline{r}' . The result returned by `mcsub()` is the impulse response $T_f(\underline{r}, \underline{r}')$ [$\text{W}/\text{cm}^2/\text{W}$]. The impulse response is then multiplied by the fluorescent power associated with the incremental volume $dV(\underline{r}')$, which is $T_x(\underline{r}') \epsilon C Y dV(\underline{r}')$ [W per W of incident power] or [W/W]. The result, $T_x(\underline{r}') \epsilon C Y dV(\underline{r}') T_f(\underline{r}, \underline{r}')$, is the incremental contribution to $F_f(\underline{r})$ from the incremental volume $dV(\underline{r}')$. Iteratively, the program uses `mcsub()` to launch fluorescence power from each of the incremental volumes $dV(\underline{r}')$, multiply the result by the local fluorescent

power in $dV(r')$, and accumulate these contributions into a final total fluence rate of fluorescence, $F_f(r)$ [$W/cm^2/W$]. Similarly, during these iterative calculations, the escaping flux density of fluorescence, $J_f(r)$ [$W/cm^2/W$], is also accumulated. These iterative accumulations are equivalent to the integration of equations 1 and 2.

The subroutine `mcsb()` propagates photons in Cartesian coordinates (x,y,z) , however the results are recorded in cylindrical coordinates (z,r) because of the cylindrical symmetry of the problem. Therefore, the final result is reported as $J_f(r)$ and $F_f(z,r)$ where z is the depth position and r is the radial position of an observation point. Throughout calculations, for example when launching a photon, a radial position r is sometimes assigned to a position x with $y = 0$, which is appropriate since the model has cylindrical symmetry.

Finally, `mcfluor.c` considers the fluorescence that is generated from a local small heterogeneity of extra fluorophore. The heterogeneity is characterized as an extra fluorescence in a small volume V_h at a specific position $\underline{r}_h = (x_h, y_h, z_h)$ with ϵ , C and Y given values of ϵ_h , C_h and Y_h . The program determines the fluorescent power source to be $T_x(\underline{r}_h) \epsilon_h C_h Y_h V_h$ [W/W]. The impulse response is obtained by `mcsb()` launching a fluorescent isotropic point source at $\underline{r}' = (0,0,z_h)$ to yield $T_f(\underline{r}', \underline{r}'')$ where \underline{r}'' is the position of an observation point relative to the fluorescent source. The points of observation are placed along the x axis with $y = 0$. The \underline{r}'' equals the line from the heterogeneity at \underline{r}_h to some point of observation \underline{r} along the x axis (with $y = 0$). Let this line be called \underline{r}_{ho} . Then the contribution of the fluorescent heterogeneity to fluorescence observed at a position \underline{r} is determined:

$$F_f(\underline{r}) = T_x(\underline{r}_h) \epsilon_h C_h Y_h V_h T_f(\underline{r}', \underline{r}_{ho})$$

where $T_f(\underline{r}', \underline{r}_{ho})$ is the impulse response $F(z,r)$ [$W/cm^2/W$] returned by `mcsb()` after launching an isotropic point source at $(0,0,z_h)$. Without an underline, z and r denote depth and radial positions, respectively.

Similarly, the contribution of the fluorescent heterogeneity to escaping flux density observed at a position r on the surface along the x axis ($y = 0$) is determined:

$$J_f(r) = T_x(\underline{r}_h) \epsilon_h C_h Y_h V_h T_f(\underline{r}', \underline{r}_{ho})$$

where $T_f(\underline{r}', \underline{r}_{ho})$ is the $J(r)$ returned by `mcsb()` after launching an isotropic point source at $(0,0,z_h)$.

The program `mcfluor.c` illustrates the calculation of fluorescence from a single small heterogeneity that is chosen to have a spherical shape. The program can be adapted to consider the contribution from several such heterogeneities. Superposition of many such heterogeneities can yield the response to a larger irregular region of extra fluorescence above the background fluorescence. Keep in mind that the heterogeneity problem breaks the cylindrical symmetry. Another caveat is that the added absorption of this fluorophore should not significantly increase the overall absorption coefficients at

either the excitation or emission wavelength. Otherwise, one no longer has a homogeneous problem for propagation of excitation and fluorescent emission and the user may need to modify `mcsb()` to consider the local absorption of the heterogeneity.

4. Example simulations

To illustrate the use of the program `mcfluor.c`, three example simulations are shown. The first example shown in Figure 1 is the launching of an isotropic point source of excitation light from a depth of 0.5 cm. The optical properties at the excitation and fluorescent wavelengths are

<u>property</u>		<u>excitation</u>	<u>fluorescence</u>
Absorption coefficient	μ_a	1 cm^{-1}	0.2 cm^{-1}
Scattering coefficient	μ_s	10 cm^{-1}	5 cm^{-1}
Anisotropy	g	0.90	0.90

The fluorophore properties are $\epsilon C = 0.1 \text{ cm}^{-1}$ and $Y = 1$. The refractive index of the medium is $n_1 = 1.33$ with an external air boundary ($n_2 = 1.00$). The figure shows the fluence rates of excitation and fluorescence, $F_x(z,r)$ and $F_f(z,r)$, and the escaping flux densities, $J_x(r)$ and $J_f(r)$. Also shown is the ratio of escaping fluorescence to escaping excitation, J_f/J_x , which increases versus radial distance illustrating that fluorescence spreads more easily than excitation due to the lower absorption and scattering properties at the fluorescent wavelength relative to the excitation wavelength.

The second example is the launching of a focused Gaussian beam for the same optical properties. The $1/e$ beam radius is 0.3 cm. The focus would be located at a depth of 0.3 cm if the refractive indices of the external and internal media were matched. The $1/e$ beam radius at the focus would be 0.0010 cm under matched boundary conditions. The program accounts for specular reflectance and refraction at the entry from air into medium. Figure 2 shows the results, similar to Figure 1.

The third example is the launching of a collimated beam. The radius of the beam is 0.3 cm. The program accounts for the specular reflectance at the entry from air into medium. Figure 3 shows the results, similar to Figure 1. For this example, the scattering coefficients are 10-fold greater than those in examples 1 and 2.

The fourth example is the fluorescence from a fluorescent heterogeneity, with optical properties like Figs. 1 and 2. In this example, the excitation was from a 0.0500-cm-radius collimated beam. The heterogeneity was a spherical object of radius 0.0100 cm characterized as $\epsilon_h C_h = 0.1 \text{ cm}^{-1}$, $Y_h = 1$. The location of the heterogeneity was $(x_h, y_h, z_h) = (0.1, 0.15, 0.3) \text{ [cm]}$. Figure 4 shows the flux density of escaping fluorescence at the surface, $J_f(\underline{r})$, and the distribution of fluence rate within the volume, $F_f(\underline{r})$.

Fig. 1 here

Fig. 2 here

Fig. 3 here

Fig. 4 here

5. The program `mcfluor.c`

The overall organization of the program `mcfluor.c` is shown below. Each part is discussed in more detail in the following sections. The `main()` routine within `mcfluor.c` calls a Monte Carlo subroutine, `mcsub()` that encapsulates the basic Monte Carlo algorithm. The program `mcfluor.c` uses `mcsub()` to (1) determine the transport of excitation light into the medium, (2) the generation and escape of fluorescence due to a uniform background concentration of fluorophore, and (3) the escape of fluorescence due to a localized heterogeneity of extra fluorophore. The program initially makes two calls to `mcsub()` with only 999 photons launched, once at the excitation wavelength and once at the fluorescence wavelength, so as to estimate the time of completion of the simulations.

```
/* mcfluor.c */
header
declare subroutines
main() {

    Set up variables and arrays.
    Specify USER CHOICES of program parameters.

    Call Monte Carlo subroutine with 999 photons,
    once for excitation and once for fluorescence wavelengths.
    Make estimate of completion time for simulation.

    Setup excitation transport.
    Call Monte Carlo subroutine.
    Save escaping flux Jx(r) and fluence rate distribution Fx(z,r).

    Setup background fluorescence.
    Iteratively call Monte Carlo subroutine, convolving against Fx(z,r).
    Save escaping flux Jf(r) and fluence rate distribution Ff(z,r).

    Setup local fluorescent heterogeneity.
    Call Monte Carlo subroutine, convolve against Fx at heterogeneity.
    Save J(r) and F(z,r) due to heterogeneity.

}
list subroutines
```

5.1 Header

The initial header portion of `mcfluor.c` sets up the `main()` program. The initial `#include` commands incorporate supporting standard C program files that are needed by the program. The `#define` commands cause global substitutions of the first argument by the second argument, i.e. `USERLABEL` is replaced by the string “an

example Monte Carlo simulation” and `BINS` is replaced by the value `101`. Thus the user can conveniently change the number of bins used by arrays in the program, and can specify a label that prints out during the run.

The subroutine declarations inform `main()` of the availability of subroutines listed at the end of the `mcfluor.c`:

`mcsub()` executes a modular Monte Carlo simulation,
`RFresnel()` evaluates the value of internal reflectance at the tissue/air surface,
`SaveFile()` saves the escaping flux density $J(r)$ and fluence rate $F(z,r)$ to a file,
`RandomGen()` is the random number generator,

and four subroutines for allocation of memory:

`nrrror()` used if error encountered during memory allocation,
`*AllocVector()` allocates memory for a 1-D array,
`**AllocMatrix()` allocates memory for a 2-D array,
`FreeVector()` frees the memory allocated for a 1-D array,
`FreeMatrix()` frees the memory allocated for a 2-D array.

5.2 `main()`

The `main()` program organizes the user’s problem, calls the modular subroutine `mcsub()`, interprets the results, and saves the results to files.

The `main()` program begins with declarations of the variables used in `main()`. The first set of variables are USER CHOICES where the user can specify the optical properties of the medium at the excitation and fluorescence wavelengths, and the properties of the fluorophore. In particular, the product of the fluorophore’s extinction coefficient and concentration, ϵC , are specified by the lumped parameter `eC` which has the units of an absorption coefficient [cm^{-1}]. The energy yield `Y` specifies the W of fluorescence per W of excitation absorbed by fluorophore [W/W] or [dimensionless]. The program is usually run with `Y` set to 1. Then if one wishes to know the results for a lower `Y` value, such as `Y = 0.01`, one simply multiplies the results for flux J_f and fluence rate F_f for fluorescence by that lower `Y` value.

The parameter `mcflag` determines whether photons will be launched as a collimated flat-field beam (`mcflag = 0`), as an approximation to a focused Gaussian beam (`mcflag = 1`), or as an isotropic point source (`mcflag \geq 2`). For flat or Gaussian beams, the parameter `radius` determines the radius of the beam incident at the surface. In the case of the focused Gaussian beam at the surface of the medium, `radius` is the radial position where the beam irradiance drops to $1/e$ the central peak value. For the focused Gaussian beam, an additional pair of parameters describes the focus point under conditions of matched boundary conditions. The parameter `waist` is the $1/e$ radius of the Gaussian beam at the focus depth, and the parameter `zfocus` is the depth position of the focus, both for a matched boundary condition. However, when a Gaussian beam is launched into a tissue with a mismatched boundary condition, `mcsub()` will calculate the specular reflectance and the refraction of the beam at the external medium/internal

medium interface. The parameters `xs`, `ys`, `zs` are used when `mcflag` ≥ 2 to describe the position of isotropic launching. When `mcflag` equals 0, 1 or 2, the subroutine `mcsb()` will print out progress reports to the user during the Monte Carlo simulation. If `mcflag` > 2 , then `mcsb()` will behave as if `mcflag` = 2 but will omit any printouts, and `mcfluor.c` will later use `mcflag` = 3 when iteratively computing contributions due to distributed point sources of fluorescence. The number of photons to be launched by `mcsb()` is set by `Nphotons`. In summary,

	<code>mcflag</code>	Radius	<code>waist</code>	<code>zfocus</code>	<code>xs</code>	<code>ys</code>	<code>zs</code>
Collimated	0	+	-	-	-	-	-
Focused Gaussian	1	+	+	+	-	-	-
Isotropic point	≥ 2	-	-	-	+	+	+

where + means “uses” and – means “ignores.”

The user chooses the number of photons to be launched by `mcsb()` by specifying the parameter `Nruns`. The user chooses `Nruns` and `mcmain()` calculates the appropriate `Nphotons` requested of `mcsb()`. There are `Nruns`* 10^6 excitation photons launched, `Nruns`*100 background fluorescent photons launched from each of $(\text{BINS}-1)^2$ bins, and `Nruns`* 10^6 fluorescent photons launched for the heterogeneity. If `Nruns` = 1 and `BINS` = 101, there will be 10^6 excitation photons launched, and $100(\text{BINS}-1)^2 = 10^6$ background fluorescent photons launched, and 10^6 fluorescence photons launched for the heterogeneity. Choosing `NRUNS` = 10 will cause 10-fold more excitation and fluorescent photons to be launched. Hence, the user can conveniently vary the number of photons being launched, for example choosing `NRUNS` = 0.1 for a quick initial run lasting 3 min then `NRUNS` = 10 for a final run lasting 5 hr.

The user also chooses the size of the bins for depth, `dz`, and radial position, `dr`, appropriate for the cylindrical coordinates used to record escaping flux density $J(r)$ and fluence rate $F(z,r)$. The values $(\text{BINS}-1)*dz$ and $(\text{BINS}-1)*dr$ specify the total depth and radial extent over which results are stored. The indices `[iz][ir]` specify the particular z and r bins, respectively. The last bins, `iz` = `BINS` and `ir` = `BINS`, are used to collect any overflow consisting of photons that migrate beyond the extent of the bins.

The remaining variables are determined by the program and the user need not specify their values. All units are in [cm] or variations such as $[\text{cm}^{-1}]$ or $[\text{cm}^2]$. The declaration and memory allocation of the 1D arrays (`Jx`, `Jf`, `temp1`) and the 2D arrays (`Fx`, `Ff`, `temp2`) employ memory allocation subroutines described in section 6.5.

There is a printout of the USER CHOICES so that the user can be reminded of the conditions of a simulation when reviewing a printout. A final initialization step sets all arrays to zero.

5.3 Excitation

This part of program considers the penetration of excitation light into the tissue or medium. The control parameters for the Monte Carlo subroutine to be used for the excitation were set in the USER CHOICES section above.

The `mcsub()` routine is called. The arguments of the subroutine include the optical properties of the medium at the excitation wavelength (`muax`, `musx`, `gx`, `n1`, `n2`), the number of r and z bins and their bin sizes and the number of photons to be launched (`NR`, `NZ`, `dr`, `dz`, `Nphotons`), the control parameters (`mcflag`, `xs`, `ys`, `zs`, `radius`, `waist`, `zfocus`), and the pointers to arrays `Jx[ir]` and `Fx[iz][ir]` for escaping flux density $J_x(r)$ [W/cm²/W] and fluence rate $F_x(z,r)$ [W/cm²/W] for the excitation light that are returned by the subroutine. The subroutine `mcsub()` records its results in cylindrically symmetric radial coordinates of z and r. The results have been normalized by `Nphotons` so the same answer is obtained regardless of the value of `Nphotons` used, although for a low choice of `Nphotons` the results are more noisy.

Finally, the flux density `Jx[ir]` and fluence rate `Fx[iz][ir]` are sent to the subroutine `SaveFile()` that saves the data along with the appropriate r and z positions of each bin based on the function arguments (`NR`, `NZ`, `dr`, `dz`). The parameter `mcflag` is also an argument of `SaveFile()` and specifies the name of the files to be saved. For the case of `mcflag = 0` the saved files are called “J0.dat” and “F0.dat”, for the case of `mcflag = 1` the saved files are called “J1.dat” and “F1.dat”, and so forth. The format of J0.dat and F0.dat are discussed in section 6.3.

5.4 Background Fluorescence

A medium often has some uniform background fluorophore concentration. The amount of fluorescence generated in each bin of medium is proportional to the local fluence rate of excitation. The next section of the program considers this background fluorescence.

In the previous section, `mcsub()` computed the fluence rate of excitation, `Fx[iz][ir]`. The local generation of fluorescence in the bin `[iz][ir]` equals `Fx[iz][ir]*eC*Y`, where `eC` is the product of the extinction coefficient ϵ [cm⁻¹ M⁻¹] and the concentration C [M], and `Y` is the power yield [W emission per W excitation absorbed by fluorophore]. The factor `Fx[iz][ir]*eC*Y` is the power density of fluorescence [W/cm³/W] that acts as a local source.

Because the Monte Carlo subroutine uses cylindrical symmetry, the launching of fluorescent photons from any point within an annular bin will have the same effect in terms of migration in r and z space. Hence, the total fluorescent source power associated with the bin `[iz][ir]` is the local fluorescence power density multiplied by the

volume of the annular bin. Each annular bin has a volume equal to $2*PI*r*dr*dz$ [cm³], where $r = (ir - 0.5)*dr$. Consequently, the total fluorescent source for each bin is $Fx[iz][ir]*2*PI*(ir - 0.5)*dr*dr*dz*eC*Y$.

The program uses these fluorescent sources associated with each bin $[iz][ir]$ to weight the results from launching fluorescent photons from the bin using `mcsub()` operating with the optical properties of the medium at the fluorescent wavelength, `muaf`, `musf`, `gf`. The program sets the control parameter `mcflag` to 3 which causes `mcsub()` to launch an isotropic point source from position `xs`, `ys`, `zs`, just like setting `mcflag` to 2, but setting `mcflag` to 3 causes `mcsub()` to avoid printing out progress reports to the user. The ACCUMULATION LOOP sequentially sets `xs`, `ys`, `zs` to each of the positions of the $[iz][ir]$ bins. Because of cylindrical symmetry, `xs` and `zs` are set to the r and z position of the bin, respectively, and `ys` is always set to 0. The number of photons requested of `mcsub()` is `100*Nruns` photons per bin. The LOOP does not launch fluorescent photons due to excitation power deposited in the overflow bins $iz = NZ$, $ir = NR$. Thus, there is an error due to ignoring excitation that has migrated beyond the extent covered by the $[iz][ir]$ bins. The user should select values of `NZ`, `NR`, `dz`, `dr` such that nearly all the deposition of excitation light occurs within the bins so as to minimize excitation energy deposition in the overflow bins.

For each $[iz][ir]$ bin, the subroutine `mcsub()` returns $J(r)$ as `temp1[ir]` and $F(z,r)$ as `temp2[iz][ir]`. These values are multiplied by the strength of the fluorescent power source in that bin, then accumulated in `Jf[ir]` and `Qf[iz][ir]`:

```
Jf[iir] += temp1[iir]*Fx[iz][ir]*2*PI*(ir-0.5)*dr*dr*eC*Y;

Ff[iiz][iir] += temp2[iiz][iir]*Fx[iz][ir]*2*PI*(ir-0.5)
               *dr*dr*dz*eC*Y;
```

After all the $[iz]$ rows of bins have accumulated for a given $[ir]$ column, a progress report may be printed out for the user. After completion of the accumulation of fluorescence generated by all bins, `Jf[]` and `Ff[][]` are saved to the files `J3.dat` and `F3.dat` using the `SaveFile()` subroutine.

5.5 Fluorescent heterogeneity

A common fluorescence problem to be solved by Monte Carlo is the magnitude of fluorescent signal from a fluorescent heterogeneity within the tissue. A small heterogeneity can be modeled as a point source of fluorescent source whose power of fluorescence [W] is equal to the product of the local fluence rate of excitation $Fx[iz][ir]$, the extra amount of fluorophore absorption and fluorescent yield, specified by the product $\epsilon_h C_h Y_h$ or `heC*hY`, due to the fluorophore heterogeneity above the background fluorescence, and the volume of the heterogeneity here specified by a spherical volume of radius `hrad`. This fluorescent power will scale the impulse response to that small heterogeneity located at position (xh, yh, zh) . The impulse response is acquired by a call to `mcsub()` for an isotropic source located at the depth position of the

heterogeneity, $z_s = z_h$, with x_s and y_s equal to 0 which returns J as `temp1[ii]` and F as `temp2[iiz][ii]`. In summary, the observed J and F due to the fluorescent heterogeneity is calculated:

```
Jf[iir] = temp1[ii]*Fx[iz][ir]*temp4;
Ff[iiz][iir] = temp2[iiz][ii]*Fx[iz][ir]*temp4;
```

where

```
temp4 = 4.0/3*PI*hrad*hrad*hrad*heC*hY;
```

and `[ii]` denotes the radial distance from the heterogeneity to the point of observation. The point of observation is denoted by `[iiz]` and `[iir]`. The position of the heterogeneity is denoted by `[iz]` and `[ir]`.

If a larger and/or irregularly shaped fluorescent heterogeneity is required, then the user can superimpose the fluorescence from many small fluorophore heterogeneities to create the result for the larger or irregular heterogeneity.

6. The subroutines

6.1 `mcsub()`

The Monte Carlo simulation is executed using a subroutine called `mcsub()` that considers a semi-infinite medium with an upper surface boundary. The subroutine has the arguments:

<code>mua</code>	absorption coefficient
<code>mus</code>	scattering coefficient
<code>g</code>	anisotropy of scattering
<code>n1</code>	refractive index of internal medium
<code>n2</code>	refractive index of external medium
<code>NR</code>	number of r bins
<code>NZ</code>	number of z bins
<code>dr</code>	incremental size of r bins
<code>dz</code>	incremental size of r bins
<code>Nphotons</code>	number of photons to be launched
<code>mcflag</code>	control of the launch as collimated, focused, or isotropic
<code>xs,ys,zs</code>	location of isotropic launching
<code>radius</code>	radius of collimated beam or 1/e radius of focused Gaussian beam
<code>waist</code>	radius of Gaussian beam at the focal point
<code>zfocus</code>	depth position of the focal point for Gaussian beam
<code>*J</code>	pointer to 1D array of flux density escaping at surface, <code>J[ir]</code>
<code>**F</code>	pointer to 2D array of fluence rate, <code>F[iz][ir]</code>

The subroutine follows an algorithm that is depicted in Figure 5. The program begins with `SETUP`, declaring and initializing the various variables. In particular, `J[ir]` and `F[iz][ir]` are set initially to values of zero.

The **LAUNCH** do-loop proceeds to launch the requested number of photons, **Nphotons**. If a photon escapes at the surface or is terminated by the **ROULETTE** procedure, then a new photon is launched. Each photon launching sets the initial photon weight **W** to a value $1.0 - \text{rsp}$ where **rsp** is the specular reflectance at the air/tissue surface. The photon launching is executed as either a collimated beam (**mcflag** = 0), a focused Gaussian beam (**mcflag** = 1), or an isotropic point source (**mcflag** ≥ 2).

Launching collimated beam

For a collimated beam, the radial position **r** of launch is selected based on a random number, **rnd**, and is assigned to the coordinate **x** while **y** and **z** are assigned the value 0:

```
x = radius*sqrt(rnd);
y = 0;
z = 0;
```

where **radius** is the beam radius provided as an argument of **mcsb()**. The photon trajectory is specified by the cosine of the angle of the trajectory relative to each of the x, y, and z axes, and these cosine(angle) values are called **ux**, **uy**, and **uz**, respectively. A collimated beam would have **uz** equal to 1 while **ux** and **uy** equal 0:

```
ux = 0;
uy = 0;
uz = 1;
```

The value of the specular reflectance, **rsp**, is calculated based on the refractive indices **n1** and **n2**:

$$r_{sp} = \left(\frac{n_1 - n_2}{n_1 + n_2} \right)^2$$

Launching focused Gaussian beam

For a focused Gaussian beam, the radial position of launch is determined:

```
x = radius*sqrt(-log(rnd));
y = 0;
z = 0;
```

where **radius** is the 1/e radius of the Gaussian beam at the tissue surface. Note that **log()** is a base e logarithm function. The focus of the Gaussian beam for a matched boundary condition is specified by **zfocus** and **waist** which is the 1/e radius of the beam at **zfocus**. The ratio **waist/radius** is used to scale the launch position **x** at the surface to yield a radial position **xfocus** at the depth **zfocus**, and the trajectory is oriented inward toward the central axis pointing to the position (**xfocus, 0, zfocus**):

```
xfocus = waist/radius*x;
```

The program then computes the trajectory required for launching at the surface at position $(x, 0, 0)$ toward the focus at position $(xfocus, 0, zfocus)$, characterized by ux, uy, uz . This trajectory is for the case of matched boundary conditions.

```
temp = sqrt((x - xfocus)*(x - xfocus) + zfocus*zfocus);
sintheta = -(x - xfocus)/temp;
costheta = zfocus/temp;
ux = sintheta;
uy = 0.0;
uz = costheta;
```

This incident trajectory is subsequently modified if there is a mismatched boundary condition. The refractive indices of the internal medium (the tissue), $n1$, and the external medium (the air), $n2$, determine the amount of specular reflectance that occurs upon entry into the tissue and the refraction that changes the photon trajectory. The subroutine `RFresnel()` determines the `rsp` for the angle of launch that is selected. The cosine of the angle of transmission across the surface boundary into the tissue is returned as the variable `uz` whose address is denoted in the argument for `RFresnel()` as `&uz`. A new `ux` is calculated:

```
rsp = RFresnel(n2, n1, costheta, &uz);
ux = -sqrt(1.0 - uz*uz);
```

Each photon is launched at a different angle and experiences a different `rsp`. The refraction at the mismatched boundary where $n1 > n2$ causes the focal point to move deeper into the tissue. Figure 6a illustrates the launching of a Gaussian beam that would focus at `zfocus` = 0.0300 cm under matched boundary conditions and Fig. 6b illustrates how the actual depth of focus has increased to 0.0390 cm due to refraction at the air/tissue surface.

Fig. 6 here

Launching isotropic point source

For an isotropic point source, the position of launch is specified by `xs, ys, zs` in the argument for `mcsb()`. The trajectory is isotropic and so has no preferential direction, and is specified:

```
costheta = 1.0 - 2.0*RandomGen(1,0,NULL);
sintheta = sqrt(1.0 - costheta*costheta);
psi = 2.0*PI*RandomGen(1,0,NULL);
cospsi = cos(psi);
if (psi < PI)
    sinpsi = sqrt(1.0 - cospsi*cospsi);
else
    sinpsi = -sqrt(1.0 - cospsi*cospsi);
ux = sintheta*cospsi;
uy = sintheta*sinpsi;
uz = costheta;
```

The value $\sin(\psi)$ is calculated as `sinpsi = sqrt(1.0 - cospsi*cospsi)` since the `sqrt()` function is faster than the `sin()` function. Because the launch point is within the tissue, there is no specular reflectance and `rsp` is set equal to zero.

For each photon launched, the specular reflectance `rsp` is calculated by one of the three methods above. Then the initial weight W of that photon is set to `1.0 - rsp`. Hence, a total weight of $W = 1.0$ is delivered to the tissue but only `1.0 - rsp` actually enters the tissue. The specular reflectance from each launching is accumulated as a total `Rsptot`:

```
Rsptot += rsp;
```

The resulting `J[ir]` will not include specular reflectance. The photon's status is initiated as `photon_status = ALIVE` where `ALIVE` has a Boolean value of 1.

Once a photon is launched, it enters the PROPAGATION CYCLE. The HOP section sets the stepsize `s` that the photon takes and updates the current photon position `(x, y, z)` based on the current trajectory `(ux, uy, uz)`:

```
s = -log(rnd)/mut;  
x += s*ux;  
y += s*uy;  
z += s*uz;
```

The ESCAPE? step asks `if (z <= 0)` and if yes then the photon is attempting to escape out the top surface of the medium. The ESCAPE section checks for total internal reflectance by calling a random number `rnd` and asking

```
if (rnd > RFresnel(n1, n2, -uz, &uz1))
```

If yes, then the photon has escaped the tissue and its weight W is added to the current escaping flux `J[ir]`. The ESCAPE section resets the photon position then takes a partial step size to just reach the surface. The radial position `r` is calculated based on the `x` and `y` positions and the choice of `ir` is made by the equivalent of an absolute value function, `ir = (long)(r/dr) + 1`, with the minimum value being 1 to indicate the first bin. Then the photon is terminated by setting `photon_status = DEAD`, where `DEAD` has a Boolean value of 0. The photon will bypass the following DROP-SPIN-ROULETTE section and reach the end of the PROPAGATION CYCLE. Because the `photon_status = DEAD` a new photon is launched.

If no, then the photon does not escape but is internally reflected, accomplished by setting `z = -z`. The `photon_status` remains `ALIVE` so the photon can enter the DROP-SPIN-ROULETTE section.

If there is no escape, the photon enters the DROP section that causes the current weight W to decrement by an amount that depends on the `albedo = mus/(mua + mus)`:

```

    absorb = W*(1 - albedo);
    W -= absorb;

```

Then the value `absorb` is added to the current bin `F[iz][ir]`. The SPIN section causes the trajectory of the photon to deviate by an angle θ specified as `costheta` = $\cos(\theta)$ based on sampling the Henyey-Greenstein scattering function using a random number `rnd`:

```

    rnd = RandomGen(1,0,NULL);
    if (g == 0.0)
        costheta = 2.0*rnd - 1.0;
    else if (g == 1.0)
        costheta = 1.0;
    else {
        temp = (1.0 - g*g)/(1.0 - g + 2*g*rnd);
        costheta = (1.0 + g*g - temp*temp)/(2.0*g);
    }

```

Also, an azimuthal angle `psi` for the trajectory change is chosen:

```

    psi = 2.0*PI*RandomGen(1,0,NULL);

```

These angles of deviation are used to calculate a new trajectory assigned to `(ux, uy, uz)`.

The ROULETTE section provides a means of terminating a photon based on absorption. A value `THRESHOLD` was set equal to $1e-4$ at the beginning of the subroutine. If the weight `W` drops below `THRESHOLD`, then the photon is either terminated or its weight `W` is increased and propagation continues. A random number `rnd` is obtained and compared with a value `CHANCE` set to 0.1 in this program. The program asks `if rnd < CHANCE`, and if yes then the weight is increased by a factor `1/CHANCE` or 10-fold. Propagation continues and the photon returns to the top of the PROPAGATION CYCLE. If no, then the photon is terminated by setting `photon_status = DEAD`. This method statistically conserves photon energy but terminates photons 9 out of 10 times that `W` drops below `THRESHOLD`. At the end of the PROPAGATION CYCLE a new photon is launched.

After the PROPAGATION CYCLE has launched `Nphotons`, the simulation is complete. The subroutine normalizes the values in `J[ir]` by the area of each `[ir]` bin to yield the escaping flux density [$W/cm^2/W$]. The subroutine normalizes the values in `F[iz][ir]` by each bin volume to yield the density of power deposition [$W/cm^3/W$], and further normalizes by the absorption coefficient `mua` [cm^{-1}] to yield the fluence rate [$W/cm^2/W$]. The subroutine returns `J[ir]` and `F[iz][ir]` to the calling program `mcmain()`. At this time, the total amount of weight associated with escaping flux is accumulated in the parameter `temp`.

The subroutine normalizes `temp` by `Nphotons` to yield the total amount of escaping flux [W/W]. The subroutine also normalizes `Rsptot` by `Nphotons` to yield

the value of specular reflectance `rsp`. In addition, the total amount of absorbed photon weight was accumulated by the parameter `Atot` and is now normalized by `Nphotons` to yield [W/W] of absorbed power. These totals for specular reflectance, absorbed power, and escaping flux are printed out in the progress report to the user, and their sum equals unity illustrating conservation of photon energy by the algorithm. These normalized parameters could easily be returned to `mcmain()` as values if the user modified `mcsb()`.

Testing `mcsb()`

To test `mcsb()`, the total escaping flux, J_{tot} [W/W], was computed:

$$J_{tot} = \sum_{ir=1}^{Nr} J[ir] 2\pi(ir - 0.5) dr^2$$

for the optical properties: $\mu_a = 1 \text{ cm}^{-1}$, $\mu_s = 9 \text{ cm}^{-1}$, $g = 0.0$, yielding an albedo = 0.90. The medium was semi-infinite and the surface boundary condition was matched. This problem has been simulated by Prahl (1988) [10] who reported in his Table 3-1 the result to be 0.4149 and that this value matched the published value of van de Hulst (1980) [11] for this problem. The `mcsb()` was run ten times with ten different `Seed` values for initializing the random number generator (see section 6.4) to yield a mean and standard deviation of 0.41507 ± 0.00018 ($n = 10$).

Also, an example problem was run which could be compared with diffusion theory. The optical properties were `mua` = 1 cm^{-1} , `mus` = 100 cm^{-1} , `g` = 0.90. The refractive indices of the internal and external media were 1.33 and 1.00, respectively, simulating a water/air interface. An isotropic point source was launched at a depth of `zs` = $1 / (\text{mua} + \text{mus} * (1 - g))$ or 0.09091 cm.

Figure 7 shows the comparison of the `mcsb()` result and diffusion theory as outlined by Farrel et al. (1992) [12] using the extrapolated boundary condition:

$$J(r) = \frac{1}{4\pi} \left(z_o \left(\frac{1}{r_1} + \frac{1}{\delta} \right) \frac{\exp(-r_1/\delta)}{r_1} + (z_o + 4AD) \left(\frac{1}{r_2} + \frac{1}{\delta} \right) \frac{\exp(-r_2/\delta)}{r_2} \right)$$

where

$$r_1 = \sqrt{r^2 + z_o^2}$$

$$r_2 = \sqrt{r^2 + (z_o + 4AD)^2}$$

$$z_o = 1 / (\mu_a + \mu_s(1 - g))$$

$$D = z_o / 3$$

$$A = \frac{1 + r_i}{1 - r_i}$$

$$r_i \approx 0.668 + 0.0636n + 0.710/n - 1.440n^2$$

where r_i is the total internal reflectance at the tissue surface and n is the ratio n_2/n_1 . The difference between Monte Carlo and diffusion theory is characterized by the ratio $(DT - MC)/MC$ where DT is the $J(r)$ calculated by diffusion theory and MC is the $J(r)$ calculated by `mcsub()`. The ratio is in the range of -0.20 to 0.10 , except near $r = 0$ where DT is far too low. This is the same behavior predicted by the MCML code [6].

Fig. 7 here

6.2 RFresnel()

The subroutine `RFresnel()` was prepared by Lihong Wang as part of the MCML code [6]. The subroutine computes the Fresnel reflectance, r_i , at an interface between two media with refractive indices `ni` and `nt` where `ni` is the medium from which the incident photon arrives at angle θ_i and `nt` is the medium into which the photon transmits at angle θ_t :

$$r_i = \frac{1}{2} \left(\frac{\sin^2(\theta_i - \theta_t)}{\sin^2(\theta_i + \theta_t)} + \frac{\tan^2(\theta_i - \theta_t)}{\tan^2(\theta_i + \theta_t)} \right)$$

where

$$\frac{\sin(\theta_t)}{\sin(\theta_i)} = \frac{n_i}{n_t}$$

The angle of incidence is specified as `ca1` = $\cos(\text{angle of incidence})$.

In the subroutine, the angle of transmission is calculated based on Snell's Law by the subroutine and the $\cos(\text{angle of transmission})$ is returned as the value of the variable `ca2_Ptr`:

```
RFresnel(ni, nt, ca1, *ca2_Ptr)
```

During photon launch as a focused Gaussian beam, the incident medium refractive index is assigned the value of the external medium (`ni` = `n2`) and the transmitted medium refractive index is assigned the value of the internal medium (`nt` = `n1`). The specular reflectance `rsp` and the $\cos(\text{angle of transmission})$, `uz`, are calculated by the subroutine call:

```
rsp = RFresnel(n2, n1, costheta, &uz);
```

During photon propagation as photons attempt to escape the medium, they are tested for the occurrence of total internal reflectance. In this case, the incident medium refractive index is the value of the internal medium (`ni` = `n1`) and the transmitted medium refractive index is assigned the value of the external medium (`nt` = `n2`). The incident $\cos(\text{angle})$ is the negative of the current value `uz` which is negative because the photon is escaping, so `-uz` is positive. The transmitted angle `uz1` is not used, but does

specify the $\cos(\text{angle of transmission})$ for the escaping photon and could be used to document the angle of escape. The test for photon escape is phrased:

```
if (rnd > RFresnel(n1, n2, -uz, &uz1))
```

and if true then there is escape and if false then there is total internal reflectance.

6.3 SaveFiles()

The `SaveFiles()` subroutine saves two files, `J[ir]` and `F[iz][ir]`, along with the appropriate values of `r[ir]` and `z[iz]`. The names of the files depends on the values of the argument `Nfile`:

```
SaveFile(*J, **F, Nfile, NR, NZ, dr, dz)
```

where `NR` and `NZ` are the number of bins and `dr` and `dz` are the incremental bin sizes. If `Nfile` equals 1, then the names of the files are `J1.dat` and `F1.dat`. If `Nfile` equals 2, then the names of the files are `J2.dat` and `F2.dat`, and so on.

The values of `r[ir]` and `z[iz]` are calculated:

```
r[ir] = (ir - 0.5)*dr
```

```
z[iz] = (iz - 0.5)*dz
```

which are the midpoints of each bin. The calculation for `r[ir]` is based on the expectation value for `r` within the `[ir]` bin, assuming that the general form of the `J[ir]` and `F[iz][ir]` responses versus `r` is $1/r$. Then the expectation value is:

$$\langle r \rangle = \frac{\int_a^b r \frac{1}{r} 2\pi r dr}{\int_a^b \frac{1}{r} 2\pi r dr} = \frac{\pi(b^2 - a^2)}{2\pi(b - a)} = \frac{b + a}{2}$$

which is the midpoint of the `[ir]` bin. A more correct assignment of `r[ir]` would involve using the true behavior of `J` and `F` versus `r`, but this leads to iteratively reconsidering an assignment after an initial determination of the behavior. The user is better advised to simply run the `mcsusb()` with smaller values of `dr` and `dz` if the user wishes to refine the estimate of behavior `J` and `F` at small `r`. One caveat is that the sizes of bins near the central `z` axis are smaller and few photons are collected in such bins. Therefore, the data for `J` and `F` near the central `z` axis often may be noisy.

The file `J.dat` is a file with two columns and `NR` rows. The first column is `r[ir]`. The second column is `J[ir]`. The file `F.dat` is a file with `NR+1` columns and `NZ+1` rows. The first element (1,1) is ignored and set to 0. The first column, rows 2 to `NZ+1`, lists the values of `z[iz]`. The first row, columns 2 to `NR+1`, lists the values of

`r[ir]`. The remaining array, rows 2 to `NZ+1`, columns 2 to `NR+1`, holds the values of `F[iz][ir]`.

6.4 RandomNumber ()

The random number generator subroutine was prepared by Lihong Wang as part of the MCML code [6]:

```
RandomGen(Type, Seed, *Status)
```

The generator is initialized by the call with `Type` set to 0 and `Seed` set to a long integer ($0 < \text{Seed} < 32000$), for example set equal to 1:

```
RandomGen(0,1,NULL);
```

Subsequently, a random number `rnd` is generated by the call with `Type` set to 1:

```
rnd = RandomGen(1,0,NULL);
```

In some cases, such as when a command must evaluate `log(rnd)`, it is important to exclude the value `rnd = 0`. In such cases, the call is phrased:

```
while ((rnd = RandomGen(1,0,NULL)) <= 0.0);
```

6.5 Memory allocation routines

The memory allocation routines were also prepared by Lihong Wang as part of the MCML code [6]. The routines are

```
nrerror(error_text[]);  
*AllocVector(nl, nh);  
**AllocMatrix(nrl, nrh, ncl, nch);  
FreeVector(*v, nl, nh);  
FreeMatrix(**m, nrl, nrh, ncl, nch);
```

They are used in declaring 1D and 2D arrays at the beginning of the program `mcmain()` and in freeing the memory allocated for these arrays at the end of `mcmain()`. They allow the arrays to be addressed by indices ranging from 1 to `NR` and 1 to `NZ`. Their use is shown in `mcmain()`.

7. Listing of `mcfluor.c`

```
/*
 * mcfluor.c
 */

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

/*****
 **** USER CHOICES ****
 *****/
#define USER_LABEL "an example Monte Carlo simulation"
#define BINS 101
/*****

/*****
 * DECLARE SUBROUTINES
 *****/
/* The Monte Carlo subroutine */
void mcsub(double mua, double mus, double g, double n1, double n2,
           long NR, long NZ, double dr, double dz, double Nphotons,
           int mcflag, double xs, double ys, double zs,
           double radius, double waist, double zfocus,
           double *J, double **F);

/* Computes internal reflectance at tissue/air interface */
double RFresnel(double n1, double n2, double ca1, double *ca2_Ptr);

/* Saves surface escape R(r) and fluence rate distribution F(z,r) */
void SaveFile(double *J, double **F, int Nfile,
              long NR, long NZ, double dr, double dz);

/* Random number generator
   Initiate by RandomGen(0,1,NULL)
   Use as rnd = RandomGen(1,0,NULL) */
double RandomGen(char Type, long Seed, long *Status);

/* Memory allocation routines
 * from MCML ver. 1.0, 1992 L. V. Wang, S. L. Jacques,
 * which are modified versions from Numerical Recipes in C. */
void nrerror(char error_text[]);
double *AllocVector(short n1, short nh);
double **AllocMatrix(short nrl, short nrh, short ncl, short nch);
void FreeVector(double *v, short nl, short nh);
void FreeMatrix(double **m, short nrl, short nrh, short ncl, short nch);

/*****
 * MAIN PROGRAM
 *****/
int main() {
```

```

/*****
/**** USER CHOICES *****/
/*****
/* excitation */
double muax = 1.0; /* excitation absorption coeff. [cm^-1] */
double musx = 100.0; /* excitation scattering coeff. [cm^-1] */
double gx = 0.90; /* excitation anisotropy [dimensionless] */
double n1 = 1.33; /* refractive index of medium */
double n2 = 1.00; /* refractive index outside medium */
short mcflag = 0; /* 0 = collimated, 1 = focused Gaussian,
2 = isotropic pt */
double radius = 0.0; /* used if mcflag = 0 or 1 */
double waist = 0.0; /* used if mcflag = 1 */
double zfocus = 0.0; /* used if mcflag = 1 */
double xs = 0.0; /* used if mcflag = 2 */
double ys = 0.0; /* used if mcflag = 2 */
double zs = 0.090909; /* used if mcflag = 2 */
/* background fluorescence */
double muaf = 5.0; /* fluorescence absorption coeff. [cm^-1] */
double musf = 50.0; /* fluorescence scattering coeff. [cm^-1] */
double gf = 0.0; /* fluorescence anisotropy [dimensionless] */
double eC = 1.0; /* ext. coeff. x conc of fluor [cm^-1] */
double Y = 1.0; /* Energy yield for fluorescence [W/W] */
/* heterogeneity */
double xh = 0.2; /* heterogeneity */
double yh = 0.15; /* heterogeneity */
double zh = 0.3; /* heterogeneity */
double heC = 0.1; /* extra eC of heterogeneity */
double hY = 1.0; /* energy yield of heterogeneity */
double hrad = 0.01; /* radius of spherical heterogeneity */
/* other parameters */
double Nruns = 0.1; /* number photons launched = Nruns x 1e6 */
double dr = 0.0100; /* radial bin size [cm] */
double dz = 0.0100; /* depth bin size [cm] */
/*****
/*****

char label[1];
double PI = 3.1415926;
double Nphotons;
long ir, iz, iir, iiz, ii;
double temp, temp3, temp4, r, r1, r2; /* dummy variables */
double start_time, finish_time1, finish_time2, finish_time3; /* for
clock() */
double timeA, timeB;
time_t now;
double *Jx, *Jf, *temp1;
double **Fx, **Ff, **temp2;
long NR = BINS; /* number of radial bins */
long NZ = BINS; /* number of depth bins */
Jx = AllocVector(1,BINS);
Jf = AllocVector(1,BINS);
temp1 = AllocVector(1,BINS);
Fx = AllocMatrix(1, BINS, 1, BINS); /* for absorbed excitation */
Ff = AllocMatrix(1, BINS, 1, BINS); /* for absorbed fluor */
temp2 = AllocMatrix(1, BINS, 1, BINS); /* dummy matrix */

```

```

strcpy(label, USER_LABEL);
printf("\n|-----\n");
printf(  "| | %s\n",label);
printf(  "| |-----\n\n");

start_time = clock();
now = time(NULL);
printf("%s\n", ctime(&now));

if (1) { /* Switch printout ON=1 or OFF=0 */
  /* print out summary of parameters to user */
  printf("----- USER CHOICES ----- \n");
  printf("EXCITATION\n");
  printf("muax      = %1.3f\n",muax);
  printf("musx      = %1.3f\n",musx);
  printf("gx        = %1.3f\n",gx);
  printf("n1        = %1.3f\n",n1);
  printf("n2        = %1.3f\n",n2);
  printf("mcflag    = %d\n",mcflag);
  printf("radius    = %1.4f\n",radius);
  printf("waist     = %1.4f\n",waist);
  printf("zfocus    = %1.4f\n",radius);
  printf("xs        = %1.4f\n",xs);
  printf("ys        = %1.4f\n",ys);
  printf("zs        = %1.4f\n",zs);
  printf("BACKGROUND FLUORESCENCE\n");
  printf("muaf      = %1.3f\n",muaf);
  printf("musf      = %1.3f\n",musf);
  printf("gf        = %1.3f\n",gf);
  printf("eC        = %1.3f\n",eC);
  printf("Y         = %1.3f\n",Y);
  printf("FLUORESCENT HETEROGENEITY\n");
  printf("xh        = %1.4f\n",xh);
  printf("yh        = %1.4f\n",yh);
  printf("zh        = %1.4f\n",zh);
  printf("heC       = %1.4f\n",heC);
  printf("hY        = %1.4f\n",hY);
  printf("hrad     = %1.4f\n",hrad);
  printf("OTHER\n");
  printf("Nruns    = %1.1f \n", Nruns);
  printf("dr       = %1.4f\n",dr);
  printf("dz       = %1.4f\n",dz);
  printf("-----\n\n");
}

/* Initialize arrays */
for (ir=1; ir<=NR; ir++) {
  Jx[ir]    = 0.0;
  Jf[ir]    = 0.0;
  templ[ir] = 0.0;
  for (iz=1; iz<=NR; iz++) {
    Fx[iz][ir] = 0.0;
    Ff[iz][ir] = 0.0;
    temp2[iz][ir] = 0.0;
  }
}

/*****

```

```

* Time estimate for completion
*****/
/* EXCITATION */
timeA = clock();
mcsub(muax, musx, gx, n1, n2, /* CALL THE MONTE CARLO SUBROUTINE */
      NR, NZ, dr, dz, 999,
      mcflag, xs, ys, zs,
      radius, waist, zfocus,
      Jx, Fx); /* returns Jx, Fx */
timeB = clock();
temp3 = (timeB - timeA)/CLOCKS_PER_SEC/60/999; /* min per photon EX */
printf("%5.3e min/EXphoton \n", temp3);
/* EMISSION */
temp = NZ/2*dz; /* zs is midway */
timeA = clock();
mcsub(muaf, musf, gf, n1, n2, /* CALL THE MONTE CARLO SUBROUTINE */
      NR, NZ, dr, dz, 999,
      3, 0, 0, temp,
      radius, waist, zfocus,
      Jx, Fx); /* returns Jx, Fx */
timeB = clock();
temp4 = (timeB - timeA)/CLOCKS_PER_SEC/60/999; /* min per photon EM */
printf("%5.3e min/EMphoton \n", temp4);
printf("estimated completion time = %5.2f min\n\n", 0.07 + (temp3 +
2*temp4)*1e6*Nruns);

/*****
* EXCITATION
*****/
printf("EXCITATION\n");
Nphotons = 1e6*Nruns;

mcsub(muax, musx, gx, n1, n2, /* CALL THE MONTE CARLO SUBROUTINE */
      NR, NZ, dr, dz, Nphotons,
      mcflag, xs, ys, zs,
      radius, waist, zfocus,
      Jx, Fx); /* returns Jx, Fx */

/* SAVE EXCITATION */
SaveFile(Jx, Fx, mcflag, NR, NZ, dr, dz);
printf("-----\n");

finish_time1 = clock();
printf("-----\n");
printf("Elapsed Time for excitation = %5.3f min\n",
      (double)(finish_time1-start_time)/CLOCKS_PER_SEC/60);
now = time(NULL);
printf("%s\n", ctime(&now));

/*****
* BACKGROUND FLUORESCENCE
*****/
printf("BACKGROUND FLUORESCENCE\n");
mcflag = 3;

/* Accumulate Monte Carlo fluorescence due to each bin source

```

```

        weighted by strength of absorbed excitation in each bin
        source, Fx[iz][ir]. Do not include the last bins [NZ][NR]
        which are for overflow. */
temp = 0.0; /* count total number of photons launched */
for (ir=1; ir<NR; ir++) {
    temp3 = 0.0; /* number of photons launched in current [ir] row */
    temp4 = eC*Y*2*PI*(ir - 0.5)*dr*dr*dz; /* fluorescence conversion
*/
    for (iz=1; iz<NZ; iz++) {

        Nphotons = 100*Nruns;
        temp += Nphotons;
        temp3 += Nphotons;

        /* Set to launch as isotropic point source at bin [iz][ir]
*/
        r2 = ir*dr;
        r1 = (ir-1)*dr;
        r = 2.0/3*(r2*r2 + r2*r1 + r1*r1)/(r2 + r1);
        xs = r;
        ys = 0;
        zs = (iz - 0.5)*dz;

        /* CALL THE MONTE CARLO SUBROUTINE */
        mcsub(muaf, musf, gf, n1, n2,
            NR, NZ, dr, dz, Nphotons,
            mcflag, xs, ys, zs,
            radius, waist, zfocus,
            temp1, temp2);

        /* Accumulate Monte Carlo results */
        for (iir=1; iir<=NR; iir++) {
            Jf[iir] += temp1[iir]*Fx[iz][ir]*temp4;
            for (iiz=1; iiz<=NZ; iiz++)
                Ff[iiz][iir] += temp2[iiz][iir]*Fx[iz][ir]*temp4;
        } /* end iir */

        } /* end iz */

    /* Print out progress for user */
    if (ir < 10)
        printf("%1.0f fluor photons \t@ ir = %ld \t total = %1.0f\n",
            temp3,ir, temp);
    else if (fmod((double)ir,10)==0)
        printf("%1.0f fluor photons \t@ ir = %ld \t total = %1.0f\n",
            temp3,ir, temp);

} /* end ir */

printf("%1.0f fluor photons \t@ ir = %ld \t total = %1.0f\n",
    temp3,ir, temp);
printf("%1.0f fluorescent photons total\n",temp);
finish_time2 = clock();
printf("Elapsed Time for background fluorescence = %5.3f min\n",
    (double)(finish_time2-finish_time1)/CLOCKS_PER_SEC/60);

/* SAVE BACKGROUND FLUORESCENCE */
SaveFile(Jf, Ff, mcflag, NR, NZ, dr, dz);

```

```

printf("-----\n");

/*****
 * HETEROGENEOUS FLUORESCENCE *
 *****/
printf("FLUORESCENT HETEROGENEITY\n");
mcflag = 2; /* isotropic pt source, give progress reports. */
Nphotons = Nruns*1e6;

/* Initialize arrays Jf[] and Ff[][] */
for (ir=1; ir<=NR; ir++) {
    Jf[ir] = 0.0;
    for (iz=1; iz<=NR; iz++)
        Ff[iz][ir] = 0.0;
}

/* Fluorescent heterogeneity at (xh,yh,zh)
 * as a small sphere with specified radius.
 * Note that results are Jf(x), F(z,x) in y=0 plane.
 * Launch at xs = 0, ys = 0, zs = zs */

/* CALL THE MONTE CARLO SUBROUTINE -> fluorescent impulse response */
xs = 0; ys = 0; zs = zh;
mcsub(muaf, musf, gf, n1, n2,
      NR, NZ, dr, dz, Nphotons,
      mcflag, xs, ys, zs,
      radius, waist, zfocus, /* ignored */
      temp1, temp2); /* temp1 = J_imp(r), temp2 = F_imp(z,r) */

/* Convolve impulse response against fluorescent source at (xh,yh,zh).
 * Weight by Fx[iz][ir]*4/3*PI*hrad*hrad*hrad*heC*hy
 * which is fluorescent power of heterogeneity [W/W]. */
temp4 = 4.0/3*PI*hrad*hrad*hrad*heC*hy;
ir = (long)(sqrt( xh*xh + yh*yh )/dr) + 1; /* ir, heterog-origin */
iz = (long)(zh/dz) + 1; /* iz, for Fx[iz][ir] */
for (iir=1; iir<=NR; iir++) { /* for Jf[iir], Ff[iiz][ir] */
    r2 = iir*dr; /* radial position of observer */
    r1 = (iir-1)*dr;
    r = 2.0/3*(r2*r2 + r2*r1 + r1*r1)/(r2 + r1);
    temp = sqrt( (r - xh)*(r - xh) + yh*yh ); /* heterog-observer */
    ii = (long)(temp/dr) + 1; /* for temp1[ii], temp2[iz][ii] */
    if (ii > NR) ii = NR;
    Jf[iir] += temp1[ii]*Fx[iz][ir]*temp4;
    for (iiz=1; iiz<=NZ; iiz++) /* for Ff[iiz][ir], temp2[iz][ii] */
        Ff[iiz][iir] += temp2[iiz][ii]*Fx[iz][ir]*temp4;
} /* end iir */

/* SAVE HETEROGENEITY FLUORESCENCE */
mcflag = 4; /* save as J4.dat, F4.dat */
SaveFile(Jf, Ff, mcflag, NR, NZ, dr, dz);

finish_time3 = clock();
printf("Elapsed Time for fluorescent heterogeneity = %5.3f min\n",
      (double)(finish_time3-finish_time2)/CLOCKS_PER_SEC/60);
printf("-----\n");

printf("-----\n");
now = time(NULL);

```

```

printf("%s\n", ctime(&now));
printf("Elapsed Time TOTAL = %15.3f min\n",
      (double)(finish_time3-start_time)/CLOCKS_PER_SEC/60);

FreeVector(Jx, 1, BINS);
FreeVector(Jf, 1, BINS);
FreeVector(temp1, 1, BINS);
FreeMatrix(Fx, 1, BINS, 1, BINS);
FreeMatrix(Ff, 1, BINS, 1, BINS);
FreeMatrix(temp2, 1, BINS, 1, BINS);

return(1);
}

/*****
 *
 * SUBROUTINES
 *
 *****/

/*****
 * The Monte Carlo SUBROUTINE
 *
 *****/

void mcsub(double mua, double mus, double g, double n1, double n2,
          long NR, long NZ, double dr, double dz, double Nphotons,
          int mcflag, double xs, double ys, double zs,
          double radius, double waist, double zfocus,
          double *J, double **F)
{
/* Constants */
double PI = 3.1415926;
short ALIVE = 1; /* if photon not yet terminated */
short DEAD = 0; /* if photon is to be terminated */
double THRESHOLD = 0.0001; /* used in roulette */
double CHANCE = 0.1; /* used in roulette */

/* Variable parameters */
double mut, albedo, absorb, rsp, Rsptot, Atot;
double rnd, xfocus;
double x,y,z, ux,uy,uz,uz1, uxx,uyy,uzz, s,r,W,temp;
double psi,costheta,sintheta,cospsi,sinpsi;
long iphoton, ir, iz, CNT;
short photon_status;

/**** INITIALIZATIONS *****/
RandomGen(0,1,NULL); /* initiate with seed = 1, or any long integer. */
CNT = 0;
mut = mua + mus;
albedo = mus/mut;
Rsptot = 0.0; /* accumulate absorbed photon weight */
Atot = 0.0; /* accumulate specular reflectance per photon */

/* initialize arrays to zero */
for (ir=1; ir<=NR; ir++) {
    J[ir] = 0.0;
    for (iz=1; iz<=NZ; iz++)
        F[iz][ir] = 0.0;
}

```

```

/*=====
===== RUN N photons =====
* Launch N photons, initializing each one before propagation.
=====*/
for (iphoton=1; i photon<=Nphotons; i photon++) {

    /* Print out progress for user if mcflag < 3 */
    temp = (double)iphoton;
    if ((mcflag < 3) & (temp >= 1000)) {
        if (temp<10000) {
            if (fmod(temp,1000)==0) printf("%1.0f    photons\n",temp);
        }
        else if (temp<100000) {
            if (fmod(temp,10000)==0) printf("%1.0f    photons\n",temp);
        }
        else if (temp<1000000) {
            if (fmod(temp,100000)==0) printf("%1.0f    photons\n",temp);
        }
        else if (temp<10000000) {
            if (fmod(temp,1000000)==0) printf("%1.0f    photons\n",temp);
        }
        else if (temp<100000000) {
            if (fmod(temp,10000000)==0) printf("%1.0f    photons\n",temp);
        }
    }

    /**** LAUNCH
        Initialize photon position and trajectory.
        Implements an isotropic point source.
    *****/

    if (mcflag == 0) {
        /* UNIFORM COLLIMATED BEAM INCIDENT AT SURFACE */
        /* Launch at (r,z) = (radius*sqrt(rnd), 0).
        * Due to cylindrical symmetry, radial launch position is
        * assigned to x while y = 0.
        * radius = radius of uniform beam. */
        /* Initial position */
        rnd = RandomGen(1,0,NULL);
        x = radius*sqrt(rnd);
        y = 0;
        z = 0;
        /* Initial trajectory as cosines */
        ux = 0;
        uy = 0;
        uz = 1;
        /* specular reflectance */
        temp = n1/n2; /* refract index mismatch, internal/external */
        temp = (1.0 - temp)/(1.0 + temp);
        rsp = temp*temp; /* specular reflectance at boundary */
    }
    else if (mcflag == 1) {
        /* GAUSSIAN BEAM AT SURFACE */
        /* Launch at (r,z) = (radius*sqrt(-log(rnd)), 0).
        * Due to cylindrical symmetry, radial launch position is
        * assigned to x while y = 0.
        * radius = 1/e radius of Gaussian beam at surface.
        * waist = 1/e radius of Gaussian focus.

```

```

    * zfocus = depth of focal point. */
/* Initial position */
while ((rnd = RandomGen(1,0,NULL)) <= 0.0); /* avoids rnd = 0 */
x = radius*sqrt(-log(rnd));
y = 0.0;
z = 0.0;
/* Initial trajectory as cosines */
/* Due to cylindrical symmetry, radial launch trajectory is
   * assigned to ux and uz while uy = 0. */
xfocus = waist/beam*x;
temp = sqrt((x - xfocus)*(x - xfocus) + zfocus*zfocus);
sintheta = -(x - xfocus)/temp;
costheta = zfocus/temp;
ux = sintheta;
uy = 0.0;
uz = costheta;
/* specular reflectance and diffraction */
rsp = RFresnel(n2, n1, costheta, &uz); /* new uz */
ux = -sqrt(1.0 - uz*uz); /* new ux */
}
else {
/* ISOTROPIC POINT SOURCE AT POSITION xs,ys,zs */
/* Initial position */
x = xs;
y = ys;
z = zs;
/* Initial trajectory as cosines */
costheta = 1.0 - 2.0*RandomGen(1,0,NULL);
sintheta = sqrt(1.0 - costheta*costheta);
psi = 2.0*PI*RandomGen(1,0,NULL);
cospsi = cos(psi);
if (psi < PI)
    sinpsi = sqrt(1.0 - cospsi*cospsi);
else
    sinpsi = -sqrt(1.0 - cospsi*cospsi);
ux = sintheta*cospsi;
uy = sintheta*sinpsi;
uz = costheta;
/* specular reflectance */
rsp = 0.0;
}

W          = 1.0 - rsp; /* set photon initial weight */
Rsptot    += rsp; /* accumulate specular reflectance per photon */
photon_status = ALIVE;

/*****
***** HOP_ESCAPE_SPINCYCLE *****/
* Propagate one photon until it dies by ESCAPE or ROULETTE.
*****/
do {

/**** HOP
* Take step to new position
* s = stepsize
* ux, uy, uz are cosines of current photon trajectory
*****/
    while ((rnd = RandomGen(1,0,NULL)) <= 0.0); /* avoids rnd = 0 */

```

```

s = -log(rnd)/mut; /* Step size. Note: log() is base e */
x += s*ux; /* Update positions. */
y += s*uy;
z += s*uz;

/* Does photon ESCAPE at surface? ... z <= 0? */
if (z <= 0) {
    rnd = RandomGen(1,0,NULL);
    /* Check Fresnel reflectance at surface boundary */
    if (rnd > RFresnel(n1, n2, -uz, &uz1)) {
        /* Photon escapes at external angle, uz1 = cos(angle) */
        x -= s*ux; /* return to original position */
        y -= s*uy;
        z -= s*uz;
        s = fabs(z/uz); /* stepsize to reach surface */
        x += s*ux; /* partial step to reach surface */
        y += s*uy;
        r = sqrt(x*x + y*y); /* find radial position r */
        ir = (long)(r/dr) + 1; /* round to 1 <= ir */
        if (ir > NR) ir = NR; /* ir = NR is overflow bin */
        J[ir] += W; /* increment escaping flux */
        photon_status = DEAD;
    }
    else z = -z; /* Total internal reflection. */
}

if (photon_status == ALIVE) {
    /***** SPINCYCLE = DROP_SPIN_ROULETTE *****/
    /***** SPINCYCLE = DROP_SPIN_ROULETTE *****/

    /**** DROP
    * Drop photon weight (W) into local bin.
    *****/
    absorb = W*(1 - albedo); /* photon weight absorbed at this step */
    W -= absorb; /* decrement WEIGHT by amount absorbed */
    Atot += absorb; /* accumulate absorbed photon weight */
    /* deposit power in cylindrical coordinates z,r */
    r = sqrt(x*x + y*y); /* current cylindrical radial position */
    ir = (long)(r/dr) + 1; /* round to 1 <= ir */
    iz = (long)(fabs(z)/dz) + 1; /* round to 1 <= iz */
    if (ir >= NR) ir = NR; /* last bin is for overflow */
    if (iz >= NZ) iz = NZ; /* last bin is for overflow */
    F[iz][ir] += absorb; /* DROP absorbed weight into bin */

    /**** SPIN
    * Scatter photon into new trajectory defined by theta and psi.
    * Theta is specified by cos(theta), which is determined
    * based on the Henyey-Greenstein scattering function.
    * Convert theta and psi into cosines ux, uy, uz.
    *****/
    /* Sample for costheta */
    rnd = RandomGen(1,0,NULL);
    if (g == 0.0)
        costheta = 2.0*rnd - 1.0;
    else if (g == 1.0)
        costheta = 1.0;
    else {

```

```

        temp = (1.0 - g*g)/(1.0 - g + 2*g*rnd);
        costheta = (1.0 + g*g - temp*temp)/(2.0*g);
    }
    sintheta = sqrt(1.0 - costheta*costheta); /*sqrt faster sin()*/

/* Sample psi. */
psi = 2.0*PI*RandomGen(1,0,NULL);
cospsi = cos(psi);
if (psi < PI)
    sinpsi = sqrt(1.0 - cospsi*cospsi); /*sqrt faster */
else
    sinpsi = -sqrt(1.0 - cospsi*cospsi);

/* New trajectory. */
if (1 - fabs(uz) <= 1.0e-12) { /* close to perpendicular. */
    uxx = sintheta*cospsi;
    uyy = sintheta*sinpsi;
    uzz = costheta*((uz)>=0 ? 1:-1);
}
else { /* usually use this option */
    temp = sqrt(1.0 - uz*uz);
    uxx = sintheta*(ux*uz*cospsi - uy*sinpsi)/temp + ux*costheta;
    uyy = sintheta*(uy*uz*cospsi + ux*sinpsi)/temp + uy*costheta;
    uzz = -sintheta*cospsi*temp + uz*costheta;
}

/* Update trajectory */
ux = uxx;
uy = uyy;
uz = uzz;

/**** CHECK ROULETTE
* If photon weight below THRESHOLD, then terminate photon using
* Roulette technique. Photon has CHANCE probability of having
its
* weight increased by factor of 1/CHANCE,
* and 1-CHANCE probability of terminating.
*****/
if (W < THRESHOLD) {
    rnd = RandomGen(1,0,NULL);
    if (rnd <= CHANCE)
        W /= CHANCE;
    else photon_status = DEAD;
}

}/*****
**** END of SPINCYCLE = DROP_SPIN_ROULETTE *
*****/

}
while (photon_status == ALIVE);
/*****
***** END of HOP_ESCAPE_SPINCYCLE *****
***** when photon_status == DEAD) *****
*****/

/* If photon dead, then launch new photon. */
} /*===== End RUN N photons =====

```

```

=====*/
/*****
* NORMALIZE
*   J[ir]      escaping flux density [W/cm^2 per W incident]
*             where bin = 2.0*PI*r[ir]*dr [cm^2].
*   F[iz][ir] fluence rate [W/cm^2 per W incident]
*             where bin = 2.0*PI*r[ir]*dr*dz [cm^3].
*****/
temp = 0.0;
for (ir=1; ir<=NR; ir++) {
    r = (ir - 0.5)*dr;
    temp += J[ir]; /* accumulate total escaped photon weight */
    J[ir] /= 2.0*PI*r*dr*Nphotons; /* flux density */
    for (iz=1; iz<=NZ; iz++)
        F[iz][ir] /= 2.0*PI*r*dr*dz*Nphotons*mua; /* fluence rate */
}

if (mcflag < 2) {
    printf("Specular = %5.6f\n", Rsptot/Nphotons);
    printf("Absorbed = %5.6f\n", Atot/Nphotons);
    printf("Escaped = %5.6f\n", temp/Nphotons);
    printf("total    = %5.6f\n", (Rsptot + Atot + temp)/Nphotons);
}

} /***** END SUBROUTINE *****/

/*****
*   FRESNEL REFLECTANCE
*   Computes reflectance as photon passes from medium 1 to
*   medium 2 with refractive indices n1,n2. Incident
*   angle a1 is specified by cosine value ca1 = cos(a1).
*   Program returns value of transmitted angle a1 as
*   value in *ca2_Ptr = cos(a2).
*****/
double RFresnel(double n1, /* incident refractive index.*/
                double n2, /* transmit refractive index.*/
                double ca1, /* cosine of the incident */
                    /* angle a1, 0<a1<90 degrees. */
                double *ca2_Ptr) /* pointer to the cosine */
                    /* of the transmission */
                    /* angle a2, a2>0. */
{
double r;

if(n1==n2) { /** matched boundary. **/
    *ca2_Ptr = ca1;
    r = 0.0;
}
else if(ca1>(1.0 - 1.0e-12)) { /** normal incidence. **/
    *ca2_Ptr = ca1;
    r = (n2-n1)/(n2+n1);
    r *= r;
}
else if(ca1< 1.0e-6) { /** very slanted. **/
    *ca2_Ptr = 0.0;
    r = 1.0;
}
}

```

```

    }
else {
    /** general. **/
    double sa1, sa2; /* sine of incident and transmission angles. */
    double ca2;      /* cosine of transmission angle. */
    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 sum ap or diff 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);
} /***** END SUBROUTINE *****/

```

```

/*****
* SAVE RESULTS TO FILES
* to files named "Ji.dat" and "Fi.dat" where i = mcflag.
* Saves "Ji.dat" in following format:
* Saves r[ir] values in first column, (2:NR,1) = (rows,cols).
* Saves Ji[ir] values in second column, (2:NR,2) = (rows,cols).
* Saves "Fi.dat" in following format:
* The upper element (1,1) is filled with zero, and ignored.
* Saves z[iz] values in first column, (2:NZ,1) = (rows,cols).
* Saves r[ir] values in first row, (1,2:NZ) = (rows,cols).
* Saves Fi[iz][ir] in (2:NZ, 2:NR).
*****/

```

```

void SaveFile(double *R, double **F, int Nfile,
              long NR, long NZ, double dr, double dz)

```

```

{
    double r, z, r1, r2;
    long ir, iz;
    char name[20];
    FILE* target;

    /* SAVE flux density J(r) */
    sprintf(name, "mcJ%d.dat", Nfile);
    target = fopen(name, "w");
    for (ir=1; ir<=NR; ir++) {
        r2 = ir*dr;
        r1 = (ir-1)*dr;
        r = 2.0/3*(r2*r2 + r2*r1 + r1*r1)/(r1 + r2);
        fprintf(target, "%5.5f \t%5.12f\n", r, R[ir]);
    }
    fclose(target);
}

```

```

/* SAVE fluence rate F(z,r) */
sprintf(name, "mcF%d.dat",Nfile);
target = fopen(name, "w");
fprintf(target, "%5.5f", 0.0); /* ignore upperleft element of matrix */
for (ir=1; ir<=NR; ir++) {
    r2 = ir*dr;
    r1 = (ir-1)*dr;
    r = 2.0/3*(r2*r2 + r2*r1 + r1*r1)/(r1 + r2);
    fprintf(target, "\t %5.5f", r);
}
fprintf(target, "\n");
for (iz=1; iz<=NZ; iz++) {
    z = (iz - 0.5)*dz; /* z values for depth in 1st column */
    fprintf(target, "%5.5f", z);
    for (ir=1; ir<=NR; ir++)
        fprintf(target, "\t %5.12f", F[iz][ir]);
    fprintf(target, "\n");
}
fclose(target);
} /***** END SUBROUTINE *****/

```

```

/*****
*      RANDOM NUMBER GENERATOR
*      A random number generator that generates uniformly
*      distributed random numbers between 0 and 1 inclusive.
*      The algorithm is based on:
*      W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P.
*      Flannery, "Numerical Recipes in C," Cambridge University
*      Press, 2nd edition, (1992).
*      and
*      D.E. Knuth, "Seminumerical Algorithms," 2nd edition, vol. 2
*      of "The Art of Computer Programming", Addison-Wesley, (1981).
*
*      When Type is 0, sets Seed as the seed. Make sure 0<Seed<32000.
*      When Type is 1, returns a random number.
*      When Type is 2, gets the status of the generator.
*      When Type is 3, restores the status of the generator.
*
*      The status of the generator is represented by Status[0..56].
*      Make sure you initialize the seed before you get random
*      numbers.
*****/

```

```

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

```

```

double RandomGen(char Type, long Seed, long *Status)
{
    static long i1, i2, ma[56]; /* ma[0] is not used. */
    long      mj, mk;
    short     i, ii;

    if (Type == 0) { /* set seed. */
        mj = MSEED - (Seed < 0 ? -Seed : Seed);

```

```

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 (ii = 1; ii <= 4; ii++)
    for (i = 1; i <= 55; i++) {
        ma[i] -= ma[1 + (i + 30) % 55];
        if (ma[i] < MZ)
            ma[i] += MBIG;
    }

i1 = 0;
i2 = 31;
}
else if (Type == 1) {          /* get a number. */
    if (++i1 == 56)
        i1 = 1;
    if (++i2 == 56)
        i2 = 1;
    mj = ma[i1] - ma[i2];
    if (mj < MZ)
        mj += MBIG;
    ma[i1] = mj;
    return (mj * FAC);
}
else if (Type == 2) {          /* get status. */
    for (i = 0; i < 55; i++)
        Status[i] = ma[i + 1];
    Status[55] = i1;
    Status[56] = i2;
}
else if (Type == 3) {          /* restore status. */
    for (i = 0; i < 55; i++)
        ma[i + 1] = Status[i];
    i1 = Status[55];
    i2 = Status[56];
}
else
    puts("Wrong parameter to RandomGen().");
return (0);
}
#undef MBIG
#undef MSEED
#undef MZ
#undef FAC
/***** end subroutine *****/

/*****
* MEMORY ALLOCATION
* REPORT ERROR MESSAGE to stderr, then exit the program
* with signal 1.
*****/

```

```

****/
void nrerror(char error_text[])
{
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

/*****
 * MEMORY ALLOCATION
 * by Lihong Wang for MCML version 1.0 code, 1992.
 * ALLOCATE A 1D ARRAY with index from nl to nh inclusive.
 * Original matrix and vector from Numerical Recipes in C
 * don't initialize the elements to zero. This will
 * be accomplished by the following functions.
****/
double *AllocVector(short nl, short nh)
{
    double *v;
    short i;
    v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
    if (!v) nrerror("allocation failure in vector()");
    v -= nl;
    for(i=nl;i<=nh;i++) v[i] = 0.0;    /* init. */
    return v;
}

/*****
 * MEMORY ALLOCATION
 * ALLOCATE A 2D ARRAY with row index from nrl to nrh
 * inclusive, and column index from ncl to nch inclusive.
****/
double **AllocMatrix(short nrl,short nrh,
                    short ncl,short nch)
{
    short i,j;
    double **m;
    m=(double **) malloc((unsigned) (nrh-nrl+1) *sizeof(double*));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;
    for(i=nrl;i<=nrh;i++) {
        m[i]=(double *) malloc((unsigned) (nch-ncl+1) *sizeof(double));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    for(i=nrl;i<=nrh;i++)
        for(j=ncl;j<=nch;j++) m[i][j] = 0.0;
    return m;
}

/*****
 * MEMORY ALLOCATION
 * RELEASE MEMORY FOR 1D ARRAY.
****/
void FreeVector(double *v,short nl,short nh)
{
    free((char*) (v+nl));
}

```

```

/*****
 * MEMORY ALLOCATION
 *   RELEASE MEMORY FOR 2D ARRAY.
 *****/
void FreeMatrix(double **m,short nrl,short nrh, short ncl,short nch)
{
short i;
for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
free((char*) (m+nrl));
}

```

¹ B.C. Wilson, G. Adam, A Monte Carlo model for the absorption and flux distributions of light in tissue, *Medical Physics* 10:824-830 (1983).

² Keijzer M, SL Jacques, SA Prah, AJ Welch: Light distributions in artery tissue: Monte Carlo simulations for finite-diameter laser beams. *Lasers Surg.Med.* 9:148-154, (1989).

³ Keijzer M, R Richards- Kortum, SL Jacques, MS Feld: Fluorescence spectroscopy of turbid media: autofluorescence of the human aorta. *Applied Optics* 28:4286-4292, (1989).

⁴ A.N. Witt, Multiple scattering in reflection nebulae I. A Monte Carlo approach, *The Astrophysical Journal Supplement Series* 35:1-6 (1977).

⁵ S.A. Prah, M. Keijzer, S.L. Jacques, A.J. Welch, "A Monte Carlo model of light propagation in tissue.," in "Dosimetry of laser radiation in medicine and biology," SPIE Institute Series Vol IS5:102-111, ed. G.J. Müller, D.H. Sliney (1989).

⁶ L.-H. Wang, S.L. Jacques, "Monte Carlo modeling of light transport in multi-layered tissues in Standard C," publ. Univ. of Texas M. D. Anderson Cancer Center (1992). Download from <http://omlc.ogi.edu/software/mc/index.html>.

⁷ L.-H. Wang, S. L. Jacques, L.-Q. Zheng: MCML - Monte Carlo modeling of photon transport in multi-layered tissues. *Computer Methods and Programs in Biomedicine*, **47**, 131-146, (1995).

⁸ S. L. Jacques: Light distributions from point, line, and plane sources for photochemical reactions and fluorescence in turbid biological tissues. *Photochem. Photobiol.* 67:23-32 (1998).

⁹ D.S. Gareau, S.L. Jacques, Transcutaneous imaging of green fluorescent protein expression in cartilage of mice, *Proceedings of the SPIE*, 4617B (2001).

¹⁰ S.A. Prah, "Light Transport in Tissue," Ph.D. dissertation, Univ. of Texas at Austin, Texas, (1988).

¹¹ H.C. van de Hulst, "Multiple Light Scattering Volume II," Academic Press, New York (1980).

¹² T.J. Farrel, M.S. Patterson, B. Wilson, A diffusion theory model of spatially resolved, steady-state diffuse reflectance for the noninvasive determination of tissue optical properties in vivo, *Med. Phys.* 19:881-888 (1992).