AR-010-149

DSTO-TR-0500

DSTO

# A MONTE-CARLO SIMULATION OF LIGHT PROPAGATION IN SEA WATER

Mike Brennan

] APPROVED FOR PUBLIC RELEASE



DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

# A MONTE-CARLO SIMULATION OF LIGHT PROPAGATION IN SEA WATER

*Mike Brennan*

**Land Space and Optoelectronics Division**
**Electronics and Surveillance Research Laboratory**

DSTO-TR-0500

**ABSTRACT**

The design and implementation of a suite of programs for Monte Carlo simulation of light propagation in turbid media is described. The program has been tailored to simulate the propagation of the green laser in the RAN Laser Airborne Depth Sounder (LADS) through turbid water. The paper describes the Monte Carlo program in detail, particularly how the inherent multiple scattering problem is interpreted for incorporation into a single scattering simulation. Assumptions made in the implementation of the program are discussed. Results of some initial simulations are presented, together with data obtained during a recent LADS sortie for comparison with the simulations. This paper forms part of the formal documentation for the simulation suite.



DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

*Published by*

*DSTO Electronics and Surveillance Research Laboratory*
*PO Box 1500*
*Salisbury SA Australia 5108*

*Telephone: (61) (08) 82595555*
*Fax: (61) (08) 82596567*

*AR No. AR-010-149*
*March 1997*

**APPROVED FOR PUBLIC RELEASE**

# A MONTE-CARLO SIMULATION OF LIGHT PROPAGATION IN SEA WATER

## Executive Summary (U)

This is an interim report on Monte Carlo simulations of photon propagation in turbid water. Application of the method to simulating the performance of laser bathymetric systems (The Royal Australian Navy LADS system in particular) is considered in detail. Initial results suggest that the method qualitatively describes the so-called "depth bias" measured by such systems (the tendency of the laser hydrographic systems to record depths greater than the actual depth in turbid water). Future extensions to the method are proposed to quantify this bias, which will yield a better understanding of the factors affecting the measured depth bias and may suggest a method for improving the existing depth bias model for the RAN LADS system.

The simulation described may be generalised to the study of a wide range of propagation problems in the opto-electronics area.

# Authors

**Mike Brennan**
Land, Space and Optoelectronics Division

*In 1982 Mike Brennan commenced a PhD in Atomic Physics at Flinders University, in which he studied electron transport in gases under the influence of crossed electric and magnetic fields, using experimental and Monte Carlo techniques.*
*After completing his PhD in 1986, he began post doctoral work in Zurich at the Swiss Federal Institute of Technology, where Monte Carlo and experimental techniques were extended to study strongly electron attaching gases.*
*Mike returned to Australia in 1988 as a School Post-Doctoral Fellow in the Research School of Physical Sciences at the Australian National University. At the ANU, Mike studied low energy electron collisions, using a time-of-flight electron spectrometer, Monte Carlo and electron swarm techniques. In 1990 he was appointed as a Research Fellow in the Electron Physics Group, where his work on low energy electron physics continued.*
*In October 1995 Mike joined LSOD of DSTO as a Research Scientist to conduct Monte Carlo simulations of the Laser Airborne Depth Sounder and research other problems related to underwater light propagation.*
*The work presented in this report builds on 15 years of experience in numerical and experimental physics.*

# Contents

# 1. INTRODUCTION

The software described in this report has been written for Monte Carlo simulation of anisotropic transport of visible light in turbid sea water, including air-sea and sea-floor boundaries. The principal motivation for writing this software is to provide a simulation tool for studying the propagation of light from the typically green laser in laser bathymetric systems in turbid water. The aim is to construct a simulation with sufficient detail to study the parameters which effect the 'depth bias' (the result of an increase in optical path length due to scattering) experienced by laser bathymetric systems. By studying the propagation of the green laser beam through simulation, it is hoped that methods of reducing the uncertainty of the depth determinations of laser bathymetric systems can be identified. Monte Carlo techniques are preferred in this case for two main reasons.

1. Monte Carlo techniques provide an effective treatment for the major physical boundary conditions imposed by
   - a laser pulse of variable spatio-temporal width,
   - a complex air-sea interface, inevitably involving wave action,
   - a desire to simulate a non-trivial sea-floor topography.
2. The output from Monte Carlo simulations can be simply tailored to provide intuitive descriptions of phenomena, using probability distribution histograms for example.

An attempt has been made to construct the code from discrete, function specific modules, with the intention that the program may be simply modified to simulate the widest possible variety of light scattering problems. Many Monte Carlo codes exist which simulate light transport in water [see for example - Joelson & Kattawar 1996, Poole *et al* 1981, Gordon 1982, Kaijser 1990, Koerber 1996, Fernee 1995 and Groenhuis 1983]. However, the application specific nature of the previous codes has made the task of altering them difficult. Therefore, new code has been written. The operation of each module has been tested using benchmarks and documented. These tests are described below. Short of an analytic solution of the transport equations for a specific case, or another independent method, this approach is aimed at establishing the highest confidence in the output of the simulations.

The type of Monte Carlo algorithm chosen for this study differs from most of the methods used in the papers cited above. Briefly, a 'simple' Monte Carlo algorithm has been chosen, over more sophisticated methods (so called semi-analytic codes), which bias the choice of photon paths which are followed in detail (forward scattering for example) to enhance the sampling of a particular piece of the sample space which is determined *a-priori* as 'interesting'. Scattering from water and suspended matter are treated separately - A variable beam attenuation coefficient and volume scattering function is included for each, together with a rate for absorption. Post collision velocity cosines of each "photon" in the simulation are determined on the basis of the shape of the Volume Scattering Function as described later in this work. Apart from this weighting, no other biasing of photon paths has been attempted.

If successful, the results of this simulation will be used to guide an extensive experimental trials program, which will determine how an operational bathymetric system (The RAN Laser Airborne Depth Sounder - LADS) can be modified to best measure, and thus compensate for, errors and uncertainties introduced into depth soundings due to water turbidity. It is important that the simulation generate output with a clearly defined statistical merit to facilitate objective comparison of results

from different input parameters, such as the density of suspended scatterers for example. A further justification of the 'simple' (or brute-force) approach can be made in the case of laser-bathymetry, where, while the 'merit' of the depth determination comes principally from photons which are highly forward scattered (supporting the use of semi-analytic methods), a significant and important feature of received signal comes from photons backscattered from water and suspended matter at depths shallower than the true bottom. Indeed, this contribution is enhanced in more turbid water and analysis of this feature in future laser-bathymetric systems is likely to provide a method of extending the operational range to more turbid waters [Billard 1986] and is best studied using an algorithm similar to that presented here; one which does not bias the paths followed and one which can be analysed using naive statistical methods for determining merit.

This paper details the elements of the simulation. In presenting the algorithms and code for the simulation as a technical report at a relatively early stage of development, the intention is to lay the foundation for informed discussion and criticism of this numerical interpretation of the physics of light propagation in turbid media and the interpretation of the LADS system in particular. The other function of this report is to act as formal documentation for the simulation software. This paper is the first in a series of reports documenting the details of the simulation and results. To achieve these goals, the paper begins with a functional description of the RAN LADS system, highlighting elements which need to be considered and solved by any successful simulation of these problems. Section two continues with a discussion of the major assumptions implicit in the interpretation of the multiple scattering problem presented in this paper and a brief description of the code. In section three, the implementation of each module in the main program is examined as a guide to the program structure. A description of the processing code and output is given. In section four, initial results are presented, primarily as a benchmark for future reference. They do, however, reveal some interesting information concerning measurement of depth. Since initial debugging began, several important modifications to the program (technical and scientific) have been identified. These are discussed in section five. No simulation is complete without data against which it can be tested. LADS data, with the automatic gain controller disengaged, was obtained during a sortie over the Sahul Banks in the Timor Sea, west of Darwin, for qualitative comparison with the simulations. Examples of this data are given and discussed in section six. The version of the simulation code presented here is not a 'complete' simulation. The code presented here assumes a receiver with an infinite aperture and infinite field-of-view, FOV. However, it has been written with a view to including variable apertures and finite FOVs, and to extending the simulation to include representations of the polarisation state of the laser, sea-surface wave action and non-trivial sea-floor topologies. The way in which the implementation of each of these is planned is detailed in section seven. In section eight, a brief guide to the simulation input is given. This is followed by acknowledgments, references and program listings.

# 2. FUNCTIONAL DESCRIPTIONS

## 2.1 The Laser Airborne Depth Sounder (LADS)

The simulation described in this document has been specially tailored to provide a numerical analogue of the RAN Laser Airborne Depth Sounder system. The interpretation is based around the following functional description of the LADS

system. Each of the words and phrases in *italics* below is interpreted in the simulation, as discussed in the remainder of the document.

The LADS system is based around a *pulsed laser*, which is *scanned about the nadir* position, *parallel to the aircraft wings*, flown at *500 m*, producing a *raster* with *3 m radius spots* at the *sea surface*, with 10 m spacing. *Light* from the beam either *reflects* back to the aircraft, or *enters the water* and *propagates through the water* to the *sea floor*. The *intensity* of the beam is *attenuated*, due to *absorption* and *scattering* from the *water* and *scattering from particles* suspended in the water.

The time difference in the *recorded Time-of-Flight* of *light returned* to a *receiver* in the aircraft by *reflection* from the *sea surface* and the *sea floor*, yields a measure of the *depth*. The received Time-of-Flight signal also contains contributions from light which is *backscattered* at approximately 180 degrees, following scattering from water, or suspended matter, which may be used to estimate the turbidity of the water sample in the optical path.

## 2.2 The Simulation

Radiation transport through a liquid at visible wavelengths, is a multiple scattering problem. That is, the radiation interacts with the light as a light field, across a distributed front, as it propagates through the liquid. In addition, there is interaction of the radiation with discrete, suspended particles to be considered. In contrast, Monte Carlo techniques of the type described here are, strictly speaking, only applicable to the (efficient) simulation of a series of local, binary encounters - ie. between two 'particles'. Thus, in order to simulate the transport problems associated with LADS, it is important to note the following points as approximations:

Optical properties of a medium, such as refraction, transmission, reflection, polarisation and absorption of light, are descriptions of aspects of the macroscopic interaction of the light field with the bulk medium through which it is travelling, rather than descriptions of the various microscopic photon-molecule interactions. All are consequences of multiple scattering across a distributed wave front. However, refraction, transmission, reflection, polarisation and photon absorption have well defined interpretations in Ray Optics, through Snell's Law, the Fresnel formulae and the like. In this simulation therefore, the light field is represented by 'rays' which have directions chosen using pseudo-random numbers, weighted by probability distributions for the relevant processes. The manner in which the ray vectors are chosen is described in the remainder of this document. For simplicity, these individual rays are referred to as 'photons' in the remainder of this document.

In short then, an initial ray is chosen at random with weighting determined by the profile of the beam leaving the source. Ray optics describes the propagation between scattering events. When scattering occurs, again according to a weighted probability, wave optics is used to describe the angular distribution of the scattered field. The simulated photon maintains its identity during scattering, but has new direction cosines, chosen at random with weighting determined by the angular distribution of the scattered field.

Note that a 'Monte Carlo photon' is not precisely a classical ray because wave optics is used to describe scattering and the photon maintains its identity during scattering. However, in contrast to the electron transport code on which this simulation is based [Brennan, 1991], no explicit reference is made to the microscopic description (photon - molecule scattering cross sections) of the interaction.

To minimise arithmetic in the simulation, all calculations are in SI units, conversions to and from other units are made either at the start of the simulation, or in the post processing software. The speed of the photons in air is set to be the group velocity of the light field in air: $3x10^8$ $ms^{-1}$. Similarly, the speed of the photons in water is set to be $2.25x10^8$ $ms^{-1}$, where the ratio of the two velocities is the assumed non-complex index of refraction for the sea water. The following Cartesian geometry is assumed: $z = 0$ at mean sea level. The altitude of the photon source (the aircraft) is positive. The source travels with the plane, notionally in the positive $y$ direction, although it is fixed during the simulation. Photons propagate initially from the source at $z = h$ in the negative $z$ direction, at some angle to the vertical, towards the sea floor at some user defined depth, $d$, (currently fixed at 50 m) on the negative $z$ axis (Figure 1). Their positions in $r$ (the radius wrt to an origin at the nadir position defined as in Figure 1), $z$ and $t$ (time) are recorded periodically. Those which are reflected, or back scattered, propagate back towards the altitude of the source. The positions of the photons in $x$, $y$ and $t$ are recorded at the point where they intersect the plane containing the source, which is parallel to the mean sea surface. The beam scanning function of LADS is simulated by setting the initial photon velocities at a user specified angle, $\theta$, in the positive $x$ direction with respect to the nadir position.

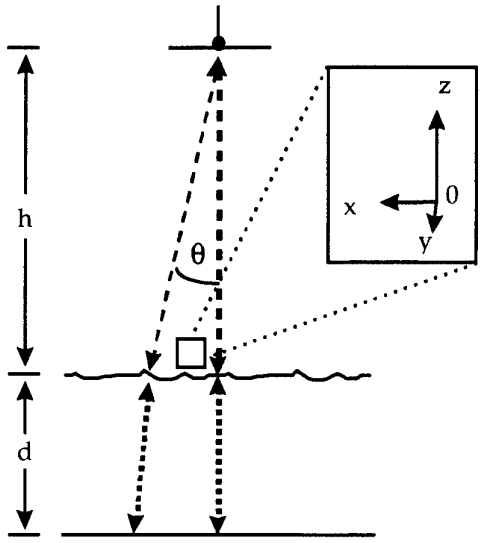


**Figure 1.** Schematic representation of the LADS system, showing the coordinate system used in the simulation. The laser source and receiver are co-located at the bottom of the aircraft. In the simulation, the laser beam is assumed to be 'scanned-out' away for nadir, at a variable angle θ, in the positive x direction only.

The fundamental output of the simulation is histograms describing the probability distributions for spatially resolved Time-of-Flight (ToF) spectra at the receiver, and in the water. For efficient simulation of the ToF spectra it is necessary to convert the characteristic attenuation coefficients from a spatial measurement, expressed in units of $m^{-1}$, to the time domain. The rate at which the laser beam is attenuated is characterised in the simulation by a 'mean collision frequency', set in the following manner:

We note that for processes in a scattering medium, the $z$ dependence of the beam intensity, $I$, due to process, $p$, is given by

$$I(z) = I_0 \exp(-p_z z),$$

where $p_z$ is the appropriate rate coefficient; the inverse of the mean free path, expressed in $m^{-1}$. In this paper, by analogy with statistical mechanics, we relate the mean free path, $1/p_z$, and the mean free time for the same process, $p_{MFT}$ through the average velocity, in this case $v_{group}$,

$$v_{group} = \frac{1/p_z}{p_{MFT}}.$$

This relationship allows us to base the simulation naturally around the basic unit of a mean free time, for the purpose of tracking the photons and interrogating their positions in phase space.

In addition to refraction at the sea-air interface, the simulation considers three possible interactions between the light and the sea-water;

- Absorption: The mean free time for the absorption process is determined from the absorption coefficient as described by the equations above. When this process is 'chosen', the photon is 'lost' from the simulation and a new photon initiated.
- Scattering from water: The mean free time for this process is set in the same manner. In the initial coding, a Henyey - Greenstein function with a user defined value for the angular form, *g*, [as in Bergougnoux *et al* 1996] is used to determine the post-collision velocity cosines. The Henyey - Greenstein function has been used previously to describe scattering from high forward scattering systems such as clouds [Bergougnoux *et al* 1996] and has been adopted in this simulation so that the dependence of the simulation on variations in the shape of the volume scattering function may be assessed. At present, Rayleigh scattering from pure water is also approximated (poorly) by this function. Future representation of the interaction with water in terms of Rayleigh scattering is discussed later.
- Scattering from suspended matter: In this simulation, a single component suspended scatterer is assumed. A single mean free time and anisotropy are used. Again a Henyey - Greenstein function is used to represent the distribution of the anisotropy.

That is, the effective density and properties of the water as an absorber and scatterer are set separately, as is the density of the scatterer. The sum of the mean free times for the above processes sets the overall mean free time for the simulation. Individual treatment of the scattering from separate components of the medium represents a relatively novel approach, in that the literature suggests that most contemporary codes use a single volume scattering function to represent the scattering from all processes in water. This novel approach allows us to simply vary the 'turbidity' of the simulation by increasing the attenuation coefficient for the suspended matter, thereby decreasing the mean free time for collisions with suspended particles. Further details are given in the next section.

The temporal width of the laser beam is variable, characterised as a temporal 'flat top' with a user defined width. The beam divergence is a user defined variable. Initial photon velocity cosines are set by observing that the spot size is the point at which the beam intensity at the sea surface has been reduced by a factor $e^1$. The spatial distribution is modelled by a Gaussian, characterised by a user defined spot radius.

The sea state is initially modelled as flat. Snell's Law is used to determine the components of the photon velocities on entry and exit of the photons from the sea surface. Photons are reflected from, and refracted into, the sea surface at entry and exit using Fresnel's Equations. Reflection from the sea surface is considered to be specular. The bottom is also modelled as flat. Photons are reflected from the bottom

according to a user defined albedo. Photons reflected from the bottom have a Lambertian distribution. Photon positions are recorded in $x$, $y$ and $t$ boxes at the source altitude and in $r$, $z$ and $t$ in the bulk of the water. Thus histograms are formed, representing the photon distribution.

# 3. STRUCTURE OF THE SIMULATION

The most important, but obvious, tenet of any Monte Carlo simulation is to design code which performs the minimum number of calculations per photon in the simulation, allowing the maximum number of photons to be simulated in a given time, thereby minimising the uncertainty in the results of a simulation. Following this principle, two programs have been developed in the initial release of the simulation. The first, described in section 3.2, is designed to perform the simulation efficiently, storing intermediate results as histograms, in arrays and periodically writing to a disk file. The second, described in section 3.3, reads and processes the results of the simulation. Splitting the simulation into discrete programs serves two purposes. First, by creating a program focussed on conducting the simulation and determining basic data for output and subsequently conducting any analysis of the data in another program, the efficiency of the simulation is increased, due to the absence of computational overhead associated with the analysis. Secondly, if the data storage from a single run can be optimised, while remaining generic to the simulation (i.e. unprocessed), the data can be analysed later for details which may not be immediately apparent during the first analysis. Storing of generic data also allows for a degree of parallelism in the simulation, in that uniquely 'seeded' runs for the same parameter set may be conducted on several machines and the output simply combined before further analysis.

## 3.1 Choice of Programming Language

Fortran90 was chosen as the development language for two significant reasons. Firstly, it allowed simple migration of Fortran77 code, previously developed for electron transport simulations by the author [Brennan 1991]. Secondly, Fortran90 has been developed to take advantage of, and extend, many of the sophisticated C and C++ functions for handling arrays, the principal output of this type of simulation. Microsoft Fortran Powerstation 'Development Studio', version 4, was used as a development environment under Windows95. DEC fortran90 and dbx for the DECalpha, running OSF v3.2 were used in the workstation environment. The DECalpha is the target platform. Critical sections of the calculation, where round-off cannot be tolerated, are performed in double precision. Elsewhere, single and integer precision are used to enhance performance.

## 3.2 The Simulation Code

In this section, source code file names appear in *italics*, program subroutines and functions are **BOLD**. Fortran90 elements are in CAPS. The simulation code is compiled from modules contained in four source files. The discrete source code structure was adopted to aid editing and debugging in the Windows95 environment. The first file, *modules.f90*, contains Fortran90 modules, which replace COMMON blocks in Fortran90 as the preferred method of transporting data between program units. The second file, listed in appendix 2, contains the main program and the subroutines and functions called directly by the main program. A third file, *level1.f90*, contains subroutines and functions used by the subroutines which main calls. The hierarchy continues to *level2.f90* which contains routines called by units in *level1.f90*. The

hierarchy is truncated at this point, so *level2.f90* also contains functions which are used by other units in *level2.f90*. The implementation of important subroutines in the main file are discussed below.

### 3.2.1 MAIN (main.f90)

The main program begins by initialising five random seeds and initialising sensitive variables to a known state. Independent pseudo random sequences are used for determining the frequency of collision and the results of the collision, to ensure that there is no underlying correlation between a series of processes. The user is interrogated for input defining the parameters for the simulation - rate coefficients, laser beam properties and the like. Details of the input may be found in section 8.1.1 The program then calls DEP_VAR, which takes the user input, performs unit conversions and calculates mean free times as described in section 2 and determines relative weights for the scattering processes, which are reported to the screen. In the subroutine, DEP_VAR, the minimum time required for photons in the simulation to reach the water surface and to return to the source height is calculated. This value, dependent on source altitude, beam scan angle and the temporal width of the laser is used to minimise the storage requirements for the simulation as described later. The program proper then begins.

The primary control loop, executed by the main program, performs a test 'WHILE' the number of simulated photons is smaller than the total desired in the simulation.

```
sim:    DO WHILE (m_photons.LT.n_total)
```

Because the simulations typically run for many days, and some intermediate monitoring is desired, OUTPUT routines are called regularly and an output counter reset.

```
        IF(i_output.EQ.n_output)THEN
          CALL OUTPUT
          i_output=0
! reset - incremented in BEGIN
        END IF
```

On the first pass, the program falls through this conditional test and BEGINS a new photon. The BEGIN routine, described in detail later, ultimately returns the initial phase space information for the photons at entry to the water, and increments counters appropriately.

```
        CALL BEGIN
```

Since BEGIN passes the photon from the source to the point of entry to the water surface, there is a likelihood that the photon will be lost from the simulation due to surface reflection. In this case the program CYCLES control to the primary loop to begin another photon. While this is roughly 2% of all simulated photons, the computation cost is much smaller than this because it occurs at the start of a photon path, which has consumed the minimum computational effort.

```
        IF(ph_finish)THEN
          CYCLE sim
        ELSE
```

Otherwise, the simulation of the photon path continues and a time variable t0, used in the traditional sense in the evaluation of Newton's equations of motion, is set to the elapsed time for the photon to reach the water from the source.

```
          t0 = t
        END IF
```

Control then passes to the secondary loop, which determines the time to the next collision for the photon under consideration, according to the previously calculated total mean free time and monitors and updates the phase space coordinates of the

photons. Control is held by this loop until one of the subroutines reports the photon 'lost', either to absorption (water and bottom), or to the altitude of the source, at which point the program CYCLES to the main loop.

```
between: DO                    ! between collision, until 'lost' loop
```

The rate at which the intensity of a beam of photons is attenuated with respect to time, due to collisions with particles in a medium of uniform density is governed by the rate equation:

$$\frac{dI}{dt} = -\frac{I}{\tau_{MFT}},$$

where $\tau_{MFT}$ is the mean free time between collisions. The solution is an exponential function, as stated in section 2. The time interval between any two collisions can be simulated using a uniform pseudo-random deviate, by finding the appropriate function which maps the uniform probability of the pseudo-random generator onto the probability of the function in question. In this case, for an exponential distribution, characterised by a rate $\tau_{MFT}$, the mapping function is a natural logarithm weighted by $\tau_{MFT}$:

$$\Delta t = -t_{MFT}\,\ell n(\mathbf{R}),$$

where **R** is a uniform pseudo-random number, **R**, drawn from the exclusive interval 0.0< R< 1.0.

```
delta_t = - time_mean*DLOG(DBLE(R))
```

The time of the collision is put on an absolute scale with respect to the center of the laser pulse,

```
tck = t + delta_t
```

To monitor the temporal motion of the photon beam as it collides, reflects and subsequently diffuses, the phase positions of the photons are updated between collisions and recorded at fixed intervals. To achieve this, control is passed to a third loop between collisions, which compares the time of the next collision with the time of the next interrogation point or update slice.

```
slices:   DO                   ! trace loop
! time to next interrogation slice
          tcut = DBLE(it + 1)*dt_slice
```

In the present simulation, the interrogation interval has been arbitrarily fixed at 0.5 ns, which is significantly smaller than the temporal resolution of the LADS system (2 ns). In a time interval of 0.5 ns, the photon propagates 0.11 m in the direction given by the velocity components. In a 'worst' case ( V = Vz) this represents an intrinsic minimum depth resolution of 0.05 m. That is, the simulation is completely insensitive to variations in parameters which change the total 'in-water' path by less than 0.11 meters. However, the LADS data analysis algorithms, which will be ultimately used analyse the simulated 'return waveforms' to determine 'simulated' depth, use a 50% constant fraction discrimination technique to derive the depth. This method has an intrinsic dependence on the 'timing-granularity' or the recorded signal. At some point, this dependence should be investigated, by varying the sampling interval.

There are three possible outcomes from the comparison against the fixed interval:

```
IF(tck.LT.tcut)THEN
```

The photon crosses no interrogation point before collision, in which case the photon proceeds immediately to collision, modifying the time variable, *t*, to equal the time at collision:

```
t = tck
```

Using this time, and the time, $t_0$, from the last phase space update, Newtonian physics gives the current $x$, $y$, $z$ coordinates

```
CALL WHERE
```

WHERE is also the routine in which bottom and surface reflections are handled. It is possible to 'lose' a photon from the simulation at this point, .ie. before collision, so provision must be made to cycle out to the primary loop in this event.

```
IF(ph_finish)THEN
   CYCLE sim  ! next photon
END IF
```

If the photon has survived to this point, the routine COLLIDE is called, in which the collision outcomes are determined. That is, the photon is either absorbed, or scattered from either water, or the suspended scatterer.

```
CALL COLLIDE
```

If absorbed, then control must be returned to the main loop.

```
IF(ph_finish)THEN
   CYCLE sim  ! next photon
END IF
```

Otherwise, control returns to the secondary loop to determine a new time of collision.

```
CYCLE between ! cycle to top of loop
```

The second possible outcome from the test of the time of collision against the time of the next interrogation, is that the photon collides exactly at the fixed time when it is interrogated:

```
ELSE IF(tck.EQ.tcut)THEN
```

It is necessary to update counters and the time, before calling WHERE and subsequently COLLIDE

```
! update slice counter
   it = it + 1
   t = tcut
   CALL WHERE
   IF(ph_finish)THEN
      CYCLE sim  ! next photon
   END IF
```

Immediately before calling the collision subroutine, where the photon may be lost, the subroutine TRACE_WATER is called, in which the r and z coordinates of the photon at time t are added to the appropriate histogram. Note that it is correct to call TRACE_WATER after WHERE, not before, because WHERE really deals with the motion between two times, whereas the trace and subsequent collisions occur at the end of a given interval.

```
   CALL TRACE_WATER
   CALL COLLIDE
   IF(ph_finish)THEN
      CYCLE sim  ! next photon
   END IF
   CYCLE between ! cycle to top of loop
ELSE
```

The third possibility is that the photon crosses at least one interrogation slice before colliding, in which case, the same procedure as for the previous possibility is invoked, but, no collision occurs and the simulation falls out to the base of this loop to check the time of collision against the next (new) interrogation time.

```
   it = it + 1
   t = tcut
   CALL WHERE
   IF(ph_finish)THEN
      CYCLE sim  ! next photon
   END IF
```

```
        CALL TRACE_WATER
    END IF
```

Note that in the simulation the interrogation interval of 0.5 ns is much smaller than typical mean free times between collisions, which are of the order of several nanoseconds at least. Thus, the third case is executed more often. While this ordering makes for easy reading, in future releases the order of the cases will be changed to 3, 1, 2 which will have an effect on the program execution, removing several conditional statement evaluations for the vast majority of collisions.

This completes the controlling loops and all that remains for the main program is to complete a redundant final output to file and screen.

### 3.2.2 Subroutine BEGIN (main.f90)

This subroutine, called from the main control loop only, initiates a photon. Initially, the controlling counters are incremented, the timing counters reset and the phase space coordinates set to a defined state in air at mean sea-level. Provision has been made at this point to include surface action, by modifying z at surface entry, calling the function SURFACE, which returns zero in this implementation:

```
z = z + SURFACE()
```

The distribution in the time-of-arrival of the laser pulse at the sea surface, due to the temporal and spatial widths of the laser, is calculated in the subroutine **LASER**. The routine **SPOT_DEV** is called by **LASER** to modify the $x$ and $y$ positions at entry, previously determined by the scan angle, to simulate the laser foot print on the sea surface (a Gaussian spatial distribution in two dimensions), using the following algorithm:

$$dx = \frac{r_s}{\sqrt{-2\ln(1.0-0.14)}}\sqrt{-2\ln[u_1]}\cos[2\pi u_2]$$

$$dy = \frac{r_s}{\sqrt{-2\ln(1.0-0.14)}}\sqrt{-2\ln[u_2]}\cos[2\pi u_1]$$

where $r_s$ is the radius of the laser spot; defined in laser optics as a disk of radius $r_s$ which encompasses 86% of the beam intensity, ie. the 1/e width.

Once the position (x, $y$, z) that the simulated photon strikes the water has been determined, the routine **SURFACE_ENTRY** is called to determine whether the photon is specularly reflected, or refracted into the water.

```
CALL SURFACE_ENTRY
```

The probability that a simulated photon is reflected, or refracted, at the air/sea interface requires a prior knowledge of the 'in air' velocity vector, the angle of incidence and the degree of polarisation of the beam. Until this time, the treatment of the photon has only required a knowledge of position and time. In **SURFACE_ENTRY** the velocity components of the photon, immediately before entry into the water, are determined for the first time by calculating the distance travelled to the water surface, including wave action (null in this release) and using the group velocity of the photon in air and the previously determined relative time of release of the photon from the source. At this point, the time-of-flight array index, *it*, is determined. Despite the fact that the calculations for the air/sea are performed, usually, at most twice for each simulated photon, this section of code is relatively CPU intensive, as a number of trigonometric functions and square roots must been evaluated. Thus, some optimisation is required.

As an initial step, the cosine of the angle of incidence with respect to the z axis, $\cos\theta_i$, is determined. Note that modifications to the program, which include wave action, will require modifications to this section to calculate the normal to the wave surface. Snell's Law is used to calculate the cosine of the angle of propagation, $\cos\theta_t$, of the photon w.r.t. to the local surface normal, *if refracted.*

$$\cos\theta_t = 1/n\sqrt{n^2 - \sin^2\theta_i}\ .$$

```
costh = ABS(Vz)/3.d8
sinth_sq = 1.0 - costh*costh
n_i = 1.0
n_t = water_n
costh_i = costh
costh_t = 1.0/water_n*SQRT(water_n_sq - sinth_sq)
```

It is efficient, even necessary, to perform the calculation for the refracted angle at this point and store the result, because the Fresnel Formulae, given below, define the ratio of the reflected and refracted fractions at the interface, for light of arbitrary incident angle and polarisation, in terms of the refractive index of both regions, $n_i$ and $n_t$, and $\cos\theta_i$ and $\cos\theta_t$:

Reflection coefficient for beam component with polarisation parallel to the plane of incidence:

$$r_{para} = \left(n_t\cos\theta_i - n_i\cos\theta_t\right) \Big/ \left(n_t\cos\theta_i + n_i\cos\theta_t\right)\ .$$

Reflection coefficient for beam component with polarisation perpendicular to the plane of incidence:

$$r_{perp} = \left(n_i\cos\theta_i - n_t\cos\theta_t\right) \Big/ \left(n_i\cos\theta_i + n_t\cos\theta_t\right)\ .$$

Transmission coefficient for beam component with polarisation parallel to the plane of incidence:

$$t_{para} = \left(2n_i\cos\theta_i\right) \Big/ \left(n_t\cos\theta_i + n_i\cos\theta_t\right)\ .$$

Transmission coefficient for beam component with polarisation perpendicular to the plane of incidence:

$$t_{perp} = \left(2n_i\cos\theta_i\right) \Big/ \left(n_i\cos\theta_i + n_t\cos\theta_t\right)\ .$$

For an unpolarised beam, as is in this case, the fraction of the reflected flux, or the reflectance, surf_reflect, is

$$surf\_reflect = \frac{1}{2}\left(\left|r_{para}\right|^2 + \left|r_{perp}\right|^2\right).$$

The components $r_{para}$ and $r_{perp}$ are determined in **SURFACE_ENTRY** by calls to functions, **R_PARA** and **R_PERP**, found in *level2.f90*. A random number, r1, on the interval [0,1] is generated and forms a test against the calculated probability of reflection from the interface, for a particular photon trajectory.

```
IF(r1.gt.surf_reflect)THEN
```

If the photon is not reflected, it is refracted into the water and the components of the 'in-water' photon velocity are given by Snell's Law in 3 dimensions:

$$V_z = \frac{-V_{group}^{air}}{n_t}\sqrt{n_t^2 - \sin^2\theta_i},$$

$$V_y = V_y / n_t^2,$$

$$V_x = V_x / n_t^2,$$

where $V_{group}$ is the photon group velocity in water and $\theta_i$ is the incident angle to the surface normal, calculated previously. The routine **TRACE_WATER** is then called to add the initial position of the photon to the appropriate histogram.

```
Vz = -2.25d8/water_n*SQRT(water_n_sq - sinth_sq)
Vy = Vy/water_n_sq
Vx = Vx/water_n_sq
CALL TRACE_WATER
ELSE
```

If the test determines that the photon is reflected at the air/sea interface, then the reflection is considered to be specular. That is, the sign of the z component of velocity, $V_z$, is reversed and the photon propagates to the source altitude, where **TO_PLANE** is called to record the time and position of arrival at the source and end the simulation of that photon.

```
Vz = -Vz
CALL TO_PLANE
END IF
```

At this point, control is returned to **BEGIN**, which in turn returns control to the primary control loop of the **MAIN** program.

### 3.2.3 Subroutine WHERE (main.f90)

As indicated previously in this section, subroutine **WHERE**, is called exclusively from the third level control loop in the main program. The main function of the routine is to calculate the position and velocity of the photon at time t, each time an interrogation point is reached (every 0.5 ns). The updated phase space information is returned through the module **PHASE**. This subroutine also provides treatments for photons which strike the sea floor and those which may strike the sea/air interface.

Newtonian mechanics forms the basis for this subroutine. The time interval between the current and latest previous call to **WHERE** is calculated.

```
delta_t = t - t0
```

Because there is possibility that photons strike either the sea floor, or sea/air interface, intermediate values of $x$, $y$ and z must be calculated to test against criteria which define the local profile and position of the sea surface and sea floor.

```
z_int = Vz*delta_t + z
y_int = Vy*delta_t + y
x_int = Vx*delta_t + x
```

Note that in the present release, with trivial sea surface and floor topologies, it is not strictly necessary to calculate intermediate values for $x$ and $y$. However, to assist in the development of the code to include these features, intermediate values are calculated as shown above. In addition, values for the local surface and bottom are calculated by calls to functions, which currently return 0 and 50 m respectively.

```
local_s = surface()
local_b = bottom()
IF((z_int.LT.local_s).AND.(z_int.GT.local_b)) THEN
```

The intermediate value of z is tested against the local values for surface and bottom. The most frequent occurrence is that the calculated position of the photon lies

between the local surface and bottom values. In this case, $x$, $y$ and $z$ are set to the intermediate values, the velocity components remaining unchanged.

```
z = z_int
y = y_int
x = x_int
```

Having satisfied the most frequent condition, the loop is complete and control passes back to the main program after setting $t_0$ equal to the current value of t in preparation for the next call to **WHERE**.

```
ELSE IF(z_int.lt.local_b) THEN
```

The next most frequent occurrence is that the photon strikes the bottom within the time interval t - $t_0$. Since the probability of reflection from a realistic sea floor is considerably less than 50%, it is most efficient, on average, to throw a random variable to determine whether the photon is absorbed, before proceeding to calculate the outcome of the reflection.

```
IF(r1.ge.bott_reflect)THEN
```

If the photon is absorbed by the sea floor, the flag ph_finish is set and control jumps to the end of the subroutine, setting $t_0$ = t as above.

```
    ph_finish = .TRUE.
    GOTO 666
END IF
```

Alternatively, the photon is reflected and the fraction of the time interval required for the photon to intersect the local bottom is determined. In addition $x$, $y$ and $z$ are determined for the time of intersection.

```
t_int = (local_b-z)/Vz
z = local_b
y = Vy*t_int + y
x = Vx*t_int + x
```

Lambertian, or diffuse, scattering from the ocean floor implies that regardless of the angle of incidence, the velocity vectors of the scattered photons are uniformly distributed on a half sphere. The value of $\cos\theta$ of the scattered photon is thus, uniformly distributed on the interval [0-1] and may be chosen directly from the pseudo random deviate. The azimuthal angle $\varphi$ is chosen, using an uncorrelated deviate, normalised to $2\pi$ radians. The corresponding components of the velocity vector may now be calculated and normalised to the 'in-water' group velocity ($2.25 \times 10^8$ $ms^{-1}$).

```
costh = r1
sinth = SQRT(1.0d0 - costh*costh)
phi=6.283185308d0*r2
Vz=2.25d8*costh
Vx=2.25d8*sinth*COS(phi)
Vy=2.25d8*sinth*SIN(phi)
```

The position at the end of the time interval, $t$ - $t_0$, is calculated using the new velocity components.

```
t_int = delta_t - t_int
z = Vz*t_int + z
y = Vy*t_int + y
x = Vx*t_int + x
ELSE IF(z_int.GT.local_s) THEN
```

Treatment of the sea/air interface is similar to that of the interaction with the sea-floor. If the intermediate value of z is found to be greater than the local sea surface, an intercept time is calculated, along with values for the surface position.

```
t_int = (local_s-z)/Vz
z = local_s
y = Vy*t_int + y
x = Vx*t_int + x
CALL surface_exit(t_int)
```

The possibility of refraction through the interface, to the source altitude, must be considered at the surface, by calling **SURFACE_EXIT** at the time of intercept. The routine **SURFACE_EXIT** performs the complimentary calculations to **SURFACE_ENTRY.** If the photon is reflected, either as a consequence of total internal reflection, for angles greater than 53°, or on the basis of a calculated probability for smaller angles, control is returned to **WHERE** and the position at the end of the interval $t-t_0$ is calculated. In future, this may involve un-physical paths through the sea/air interface, depending on the model used for the wave action.

```
IF(.NOT.ph_finish)THEN
   t_int = delta_t - t_int
   z = Vz*t_int + z
   y = Vy*t_int + y
   x = Vx*t_int + x
```

For very shallow bottoms, it is possible that in the remaining time, the photon strikes the bottom. In this case the photon is ignored.

```
            IF(z.le.bottom())THEN
! Ignore the rest of this photon
               WRITE(*,*)' A surface reflected photon has struck the bottom'
               ph_finish = .TRUE.
               WRITE(*,*)' z = ',REAL(z)
            END IF
         END IF
      ELSE IF(z_int.EQ.local_b)THEN
```

It is possible that the calculated intermediate value of z is equal to either the local surface, or bottom. If these cases are not treated, then the photon is effectively lost from the simulation, either 'floating' or 'drowning', not satisfying the condition that a photon be between the two boundaries. A simple treatment is to adjust the depth by an appropriately small amount, in this case 1 mm.

```
! set the depth to depth +0.1 mm
         z = local_b + 0.0001
         y = y_int
         x = x_int
      ELSE
! photon is relaxing on the surface ...
! set the depth to depth -0.1 mm
! may need checking for V normalisation 02/08/96
         z = local_s - 0.0001
         y = y_int
         x = x_int
      END IF
```

The subroutine ends, setting $t_0 = t$ in preparation for the next call.

```
666   t0 = t
      RETURN
      END
```

### 3.2.4 Subroutine COLLIDE (main.f90)

**COLLIDE** is the most computationally intensive section of the simulation. Called exclusively from second level control loop of the main program at the time of collision, **COLLIDE** determines which component of the water / suspension medium the photon strikes, the type of collision and the appropriate outcomes, including new velocity vectors for those photons which are anisotropically scattered. The possible processes are represented by relative probabilities, calculated from the characteristic coefficients in **DEP_VAR** at the start of the simulation. A random number, RCOLL, is generated and compared against the calculated fractions representing the processes.

```
      IF(Rcoll.GT.s_coeff) THEN          ! greater than scat ... absorb
```

Some of the photons are absorbed, a diagnostic counter is incremented and the photon 'finished' flag set to true. Control returns to the main program.

```
        xabsorb = xabsorb  +  1.d0
        ph_finish = .true.
        RETURN
    ELSE IF(Rcoll.GT.w_coeff) THEN ! greater than water ... suspended
```

If scattering from the suspension is 'chosen', the mean value of cos(θ), which defines the broad shape of the Henyey - Greenstein function, is set to the user defined value for the scatterer and the appropriate diagnostic counter is incremented.

```
        g_ = g_scatt
        xscatt = xscatt  +  1.d0
    ELSE
```

Since the same function is used to describe the angular scattering from both water and suspension, the same procedure is followed if scattering from water is chosen.

```
        g_ = g_water
        xwater = xwater  +  1.d0
    END IF
```

Seven steps are required to return the new velocity cosines for the scattered photon. Clearly, the calculation of the initial choice of scattering angles can be optimised using look-up tables, as discussed later. However, it is not obvious how the transformations which place these otherwise arbitrary angles in the frame of reference defined by the relationship of source and sea-surface may be optimised. In the case of electron scattering from a gas for example, isotropic scattering is often assumed as an approximation to the random thermal motion of the gas molecules. Isotropic scattering, which is quick to calculate can not be assumed here, a full treatment is required.

The theta dependence of the differential scattering cross section, $d\sigma/d\theta$, (or volume scattering function for multi-component suspensions) has the same dependence as the probability density for cos(θ). Thus, a uniform deviate may be appropriately transformed, using the functional form for $d\sigma/d\theta$ to randomly choose a value for the scattering angle, cos(θ). The transformation function between the random number R1 and cos(θ) is analytic for the Henyey - Greenstein function, which explains its widespread acceptance in random media simulation. A random number is used, together with the mean value of cos(θ), g, to determine a value for cos(θ) in the simulation with the correct anisotropic dependence.

```
R1 = randti(ks , rks)
ak = ak + DBLE(r1)
callsk = callsk + 1.0d0
costh = 1.d0/2.d0/g_*(1.d0+g_*g_ - (1.d0-g_*g_)*(1.d0-g_*g_)/      &
        (1.d0-g_+2.d0*g_*r1)/(1.d0-g_+2.d0*g_*R1))
```

The relationship $\cos^2(\theta) + \sin^2(\theta) = 1$ is used to determine the value of sin(θ).

```
IF(ABS(costh).EQ.1.0)THEN
    sinth = 0.d0
ELSE
    sinth = DSQRT(1.0d0  -  costh*costh)
END IF
```

Thus far, an element of solid angle, in effect an annulus between θ and θ+dθ, centred about the direction of the photon velocity immediately before scattering, has been chosen, into which the photon must scatter. To uniquely determine the φ dependence and thus the new scattered velocity cosines, while preserving the anisotropy of the scattering function, it is necessary to first find the component of the scattered velocity vector which is parallel to the incident. That is to scale cos(θ) to $V_f/V_i$.

```
vxp=vx*costh
```

```
vyp=vy*costh
vzp=vz*costh
```

Then define a non zero length random vector in space.

```
rcosth=(1.d0-2.d0*r1)
rsinth=SQRT(1.d0-rcosth*rcosth)

r1 = randti(ms , rms)
am = am + DBLE(r1)
callsm = callsm + 1.0d0
r1=6.283185308d0*r1
rx=rsinth*COS(r1)
ry=rsinth*SIN(r1)
rz=rcosth
```

Now, the random vector and the initial velocity define an arbitrary, but known plane, from which the azimuthal angle, $\varphi$, may be determined. First the components of the random vector which lie parallel and orthogonal to $V_i$ are determined:

```
c=(vx*rx+vy*ry+vz*rz)/5.0625d16
rxp=vx*c
ryp=vy*c
rzp=vz*c

rxo=rx-rxp
ryo=ry-ryp
rzo=rz-rzp
c=SQRT(rxo*rxo+ryo*ryo+rzo*rzo)
```

The orthogonal components are scaled to the appropriate length for the scattered velocity in water:

```
c=2.25d8*sinth/c
vxo=rxo*c
vyo=ryo*c
vzo=rzo*c
```

Simple scalar addition of the orthogonal and parallel components gives the velocity components in the frame of reference defined by our simulation.

```
vx=(vxo+vxp)
vy=(vyo+vyp)
vz=(vzo+vzp)
```

While not strictly necessary, the group velocity is calculated to check for internal consistency and control returns to the main program.

```
v=SQRT(vx*vx+vy*vy+vz*vz)
RETURN
END
```

### 3.2.5 Subroutine TRACE_WATER (main.f90)

Once called from the inner most control loop in the main program, subroutine **TRACE_WATER**, determines which cell in the 'in-water' histogram the current phase space information of the photon contributes to. Other relevant monitors and flags are updated. A similar routine **TRACE_PLANE** deals with the special case of the photons at z=plane, ie it accumulates the histograms at the source altitude. In order to optimally pack the array space for non-nadir simulations, photon histories are recorded in a histogram centred at $x$ = plane*sin(scan_angle), $y$ = 0. An array index, *ir*, representing the radius is determined by subtracting the 'center' in $x$ [x = plane*sin(scan_angle)] from the photon's absolute $x$ coordinate and combining this with the $y$ coordinate appropriately.

```
x_int = x - x_center
r = SQRT(x_int*x_int + y*y)
```

Given the 1 meter boxing used in this simulation, *ir* is then found simply:

```
ir = INT(r)+1
```

If the value of r is greater than that allowed for in the static array size of this program (10 meters), *ir* is set to the maximum array index and the diagnostic flag, *xr_hot*, representing the number of photons having radii larger than that allowed for, is incremented.

```
IF(ir.gt.ir_rzt)THEN
   ir = ir_rzt
   xr_hot = xr_hot + 1.0
END IF
```

A similar treatment is used for the histogram index in the z direction. Note that since for sub-surface z values, z is negative, it is necessary to form the index on the absolute value of z.

```
iz = INT(ABS(z))+1
```

A value of *iz* which is greater than the maximum z index of the array in this version of the code represents a serious coding error, as somehow photons have reached depths greater than the bottom. The user is alerted to the fact that this has occurred, *iz* is set to the maximum allowable index of the array and the simulation proceeds.

```
IF(iz.gt.jz_rzt)THEN
   iz = jz_rzt
   WRITE(*,*)' ALERT: iz = 50'
END IF
```

At the time that any photon enters the water, a zero reference timing index, *it_entry*, is determined. The appropriate offset time is determined for the 'in-water' histogram.

```
it_water = it - it_entry + 1
IF(it_water.gt.kt_rzt)THEN
   it_water = kt_rzt
   xtw_hot = xtw_hot + 1.0
END IF
```

Following appropriate end-treatment for long lived photons, the histogram is modified and arrays which monitor the maximum temporal and radial extents are updated if necessary. Control then returns to the main program.

```
box_rzt(ir, iz, it_water) = box_rzt(ir, iz, it_water) + 1
IF(t.lt.t_max(1)) t_max(1)=t
IF(t.gt.t_max(2)) t_max(2)=t
IF(r.gt.r_max(1)) r_max(1)=r
RETURN
  END
```

## 3.3 The Processing Code

The processing code comprises two files: the module file *modules.f90*, which is identical to that used in the simulation code, and the main program, as listed in sect on 10.5.

The main processing code is relatively simple in structure. The user is interrogated for the names and paths of the files containing the output from the simulation and the run parameters. The program then reads the data from the histogram files for both the 'in-air' and 'in-water' cases. As a development aid, the number of array elements which contain 'non-zero' data are counted for both cases and reported to the screen. This gives a measure of the efficiency with which data is being stored. The data concerning the number of photons simulated is read from the run parameter file to allow the user to check that the content of the files do indeed reflect the data that the user wishes to process.

The possible options for data analysis are listed at the start of a loop:

```
qu:   DO
        WRITE(*,*)' '
        WRITE(*,*)' What operation do you wish to perform?'
        WRITE(*,*)' Write out ASCII ToF histogram at plane height?          ... 1'
        WRITE(*,*)' Write out ASCII ToF ln(data) in water?          .. 2'
        WRITE(*,*)' Write out ASCII R,Z in-water data for given time(s)?    ... 3'
        WRITE(*,*)' Quit from Case loop?                                    ... 0'
        READ(*,*)iqu
        SELECT CASE (iqu)
```

On the basis of the answer to the questions above, one of 4 CASE statements is executed:

In case one, the in-air ToF histogram is written as an array of coordinate pairs, representing elapsed time from start of pulse and the corresponding histogram height. In case two, the ordering of the z dependence of the in-water *r,z,t* histogram is inverted, so that in a two dimensional plot, shallow depths (z, a small negative number) appear at the top of the page, rather than near the origin at z = 0. Similarly, the radial data are written out so that contour plots are across the diameter, centred at r = 0. The dynamic range of the in-water histogram is compressed by taking the natural logarithm of the height of the histogram to facilitate display on a 256 discrete colour map. Note that the 'end treatment' for the in-water data, implies that the last box in the ToF histogram is meaningless and will usually be higher than the trend of the adjacent boxes suggest, as all photons at larger radii are recorded in the last box. The third option allows the user to select specific time intervals for which the R,Z in-water data is written to file.

# 4. INITIAL RESULTS

In the process of developing and debugging the code, benchmark tests were performed on individual modules to verify their correct function. The functions RANDT and RANDTI which generate the uniform pseudo-random deviates on the exclusive and inclusive intervals (0 - 1) and [0 - 1], have been used extensively elsewhere by the author (see for example Brennan [1992]), and have only been tested superficially in this work. It is noted that the algorithms of Tootill *et al* [1973], which are used to generate the random numbers with a $2^{607}$ period are now considered to represent a minimum standard for uniformity and randomness [Brent 1996]. The subroutine SPOT_DEV, which generates the $x$ and $y$ offsets for a Gaussian beam profile of known (given) spot size was tested by generating $10^8$ pseudo-random pairs and comparing the results by eye for a 3 meter spot beam, against the expected Gaussian distribution on a log scale. The agreement was sufficiently impressive to a diameter of 6 meters (ie. 2x spot size) that no rigorous $\chi^2$ testing was attempted.

The integrity of other subroutines and functions in the simulation and analysis programs were tested during debugging using hand checking for individual photons throughout representative paths. Most of the program functionality has been tested to date, with the exception of the simulation of non-nadir beams, which awaits the enhancement of the data storage routines (section 5). Figure 2., above, shows the results of a 'special case' test of the validity of the Snell's Law and Fresnel formulae algorithms. $2.5 \times 10^4$ photons, propagating from a delta function source at 500 m, orthogonal to the sea-surface, with zero beam divergence, were simulated. Scattering and absorption processes from both water and the suspended matter were calculated, but the results suppressed (that is - no scattering, or absorption). In this test, the sea-floor was modelled as a non-absorbing/specular reflector, as opposed to a more realistic partially absorbing Lambertian scatter used elsewhere in

this simulation. This approach preserves the "zero angle scattering" nature of the test. In the test, as elsewhere, the Air/Water interface treatment was complete for unpolarised rays, the transmitted and reflected fractions given by the Fresnel formulae (2.04% reflected at nadir for $n_{12}$ = 1.3333). At nadir, Snell's Law was calculated, but returned only $v_z = - c/n_{12}$ for the group velocity in water. In the figure, the peak at 3333.333 ns, with a height of 2.04% of the simulated photons, corresponds to photons which propagated to the surface and were immediately reflected back to the source. The next peak, at 3777.777 ns, corresponds to photons which were refracted into the water, then striking the perfectly reflective bottom and propagating back to the sea-surface, being transmitted through the sea/air interface and returning to the source altitude. A further peak is evident 444.444 ns later, corresponding to those photons which have been reflected at the sea/air transition at 3777.777 ns, propagating once more to the bottom and back, exiting at this time to the source altitude. In this particular simulation this process was faithfully repeated out to 5111 ns. The amplitude of the peaks generated were within one standard deviation of the expected values.

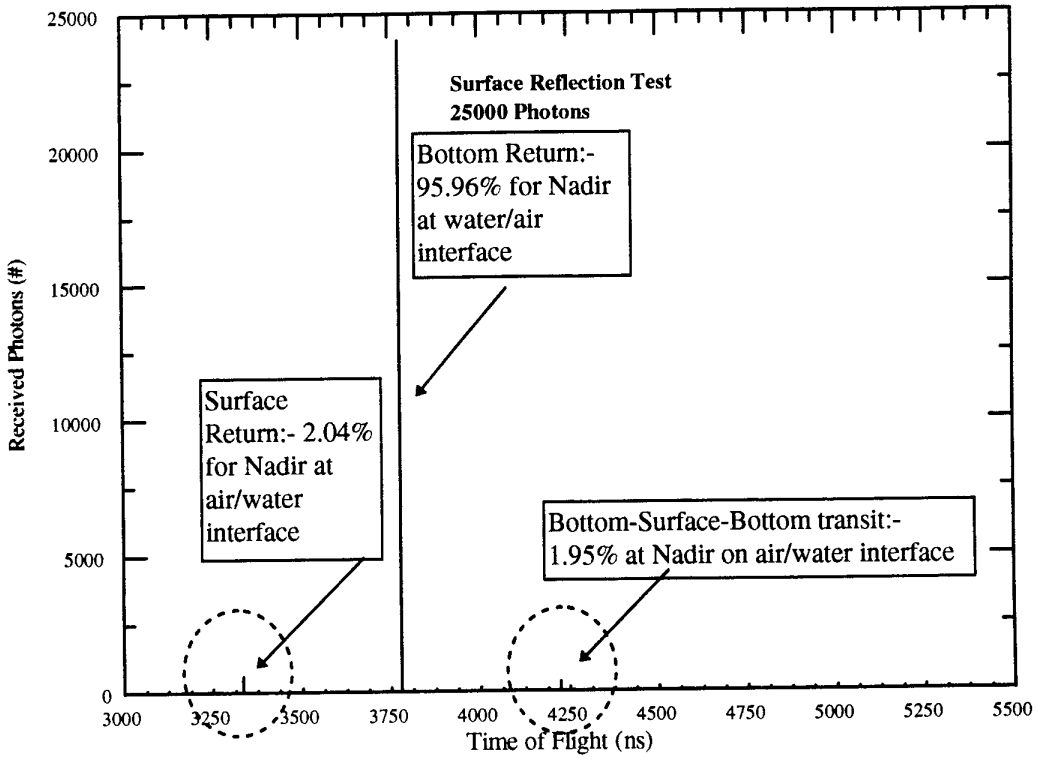


**Figure 2.** Test run for Snell's Law and Fresnel formulae algorithms. Peaks correspond to reflections from interfaces. $2.5 \times 10^4$ photons propagating from delta function source at 500 m at nadir with zero beam divergence. Scattering and absorption processes are suppressed. Sea-floor is modelled as a non-absorbing / specular reflector. Air/Water interface treatment is complete for unpolarised rays.

The series of figures 3a-c show simulated spectra for all photons arriving at the $x$-y plane containing the photon source for three key test source configurations. In effect they are LADS simulations for an infinite receiving aperture and an infinite field of view (FOV). In each, $10^8$ photons where initiated from a source 500 m above the sea surface. The sea floor was modelled as a non-absorbing, diffuse scatterer, 50 m below the surface. In 3a and 3b the source was a delta function in time and space. The scattering coefficients were from table 3 of Joelson and Kattawar [1996] for $\lambda$ = 532 nm. For the data in Figure 3a the water scattering coefficient was 0.0022 $m^{-1}$,

the suspended matter scattering coefficient: 0.0035 $m^{-1}$ and the absorption coefficient: 0.0544 $m^{-1}$. The surface reflected photons appear at the same time as in Figure 2 (3333.333 ns). The growth of low level backscatter is due to the forward scattering dominated Henyey - Greenstein distribution ($g_w = g_{sus} = 0.95$). The sharp step at 3777.777 ns corresponds to photons suffering no collisions throughout the path (or at least very forward scattered). Long path photons (those suffering several collisions) then appear as a decaying tail. Note again that the spectrum is integrated over all $x$ and $y$ at the source height. This explains the long decaying tail after the first bottom reflection, in contrast to real LADS data, where the finite acceptance angle of the green receiver and the finite FOV sharply truncates the tail of the observed spectra.

Figure 3b is similar to 3a, except the suspended matter scattering coefficient was set to 0.1955 $m^{-1}$, corresponding to 'coastal waters'. Significantly more backscatter following the surface pulse is evident in this case. The bottom reflection appears as a small step in the data plotted on a log scale. Comparison of Figures 3a and 3b reveals that while the bottom feature in Figure 3b begins at the same time as in Figure 3a (as expected, given that the depth is the same), the larger scattering coefficient delays the peak of the bottom feature. As discussed in section 3.2.1, the LADS data analysis algorithms, use a 50% constant fraction discrimination technique to derive the depth.

Figure 3b suggests that the LADS data processing would therefore determine the 'bottom' to be midway between the actual bottom at the foot of the step and the peak of the feature, that is somewhat delayed in time. This is evidence that the 'depth bias' can be successfully modelled by this simulation.

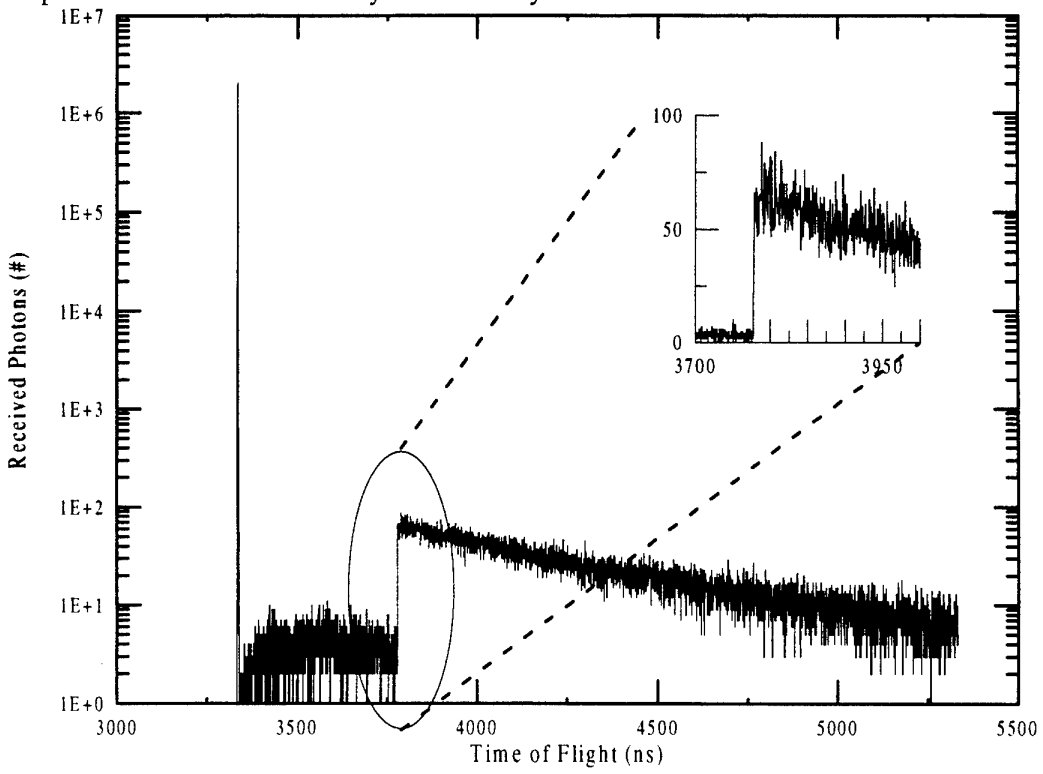


**Figure 3a.** Test run for $10^8$ photons propagating from delta function source at 500 m at nadir with zero beam divergence. Water scattering coefficient: 0.0022 $m^{-1}$. Suspended matter scattering coefficient: 0.0035 $m^{-1}$. Absorption coefficient: 0.0544 $m^{-1}$.

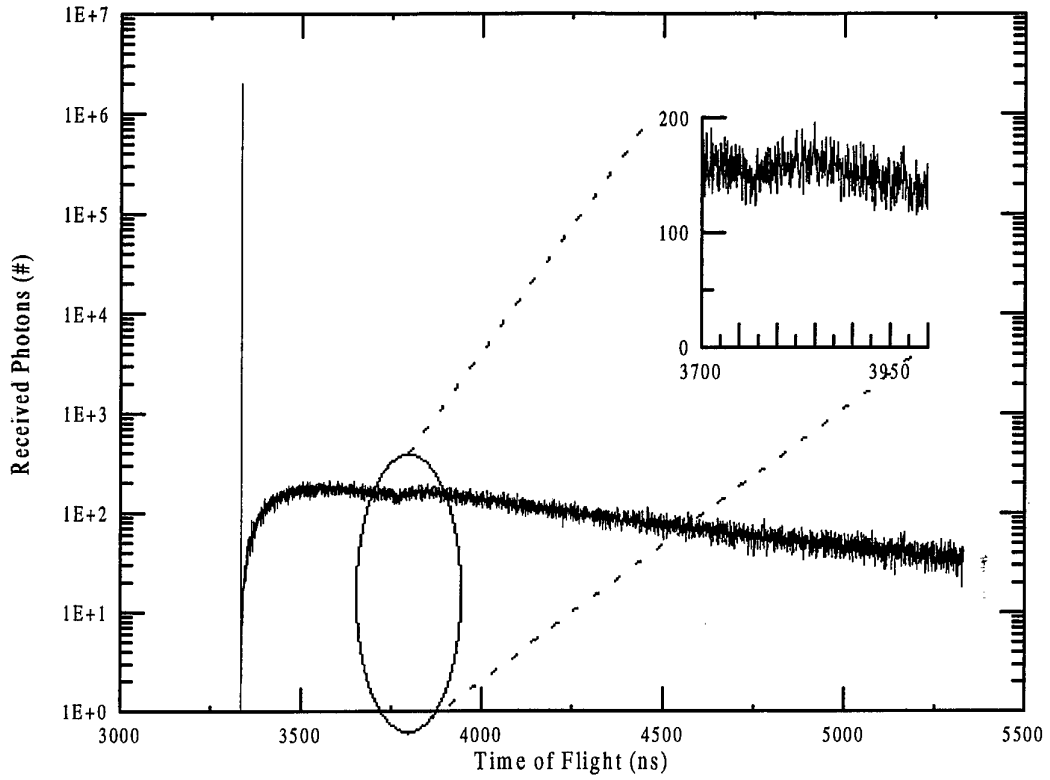


**Figure 3b.** Simulation details as for Figure 3a., except: Suspended matter scattering coefficient: 0.1955 $m^{-1}$. The highlighted bottom return feature is slightly delayed w.r.t. Figure 3a.

Figure 3c is again similar to 3a, except that a 10 ns full width 'flat top' source is used, as evidenced by the finite width of the surface reflected pulse, which is approximately an order of magnitude smaller in amplitude - the 2.04% of reflected photons now distributed over 10 ns, as opposed to a delta function in time. Interestingly, the simulation correctly predicts an identical minimum source-bottom-source transit time, and close inspection reveals a somewhat wider 'step' corresponding to the bottom return.

Figure 4 shows the evolution of the density profile of a pulse of photons for the same attenuation parameters as in Figure 3a, as they propagate from $r = 0$, $z = 0$ at $t = 0$, towards the bottom, which they first strike between $t = 215$ ns and $t = 230$ ns. After 425 ns the 'front' of the simulated pulse can be seen approaching the surface on the 'way up'. The front appears as a slight change in shading below -5 m. The lighter shading at shallow depths for this frame represents photons which have been backscattered several times, and have not been reflected from the bottom. The dark band (high concentration) at $r = 10$ m, evident for times > 50 ns represents photons at larger radii. The positions of these photons are tracked in the simulation, thus they may contribute to the return signal, but the distribution is only recorded out to 10 m radius. The granularity evident at $t = 50$ ns is due to the fact that the density distribution is recorded in 1m x 1m boxes. At $t = 50$ ns, there are a small number of photons in front of the 'head' of the pulse. This is due to small imperfections in modelling the temporal 'flat top' of the laser pulse, which will be attended to in later releases.

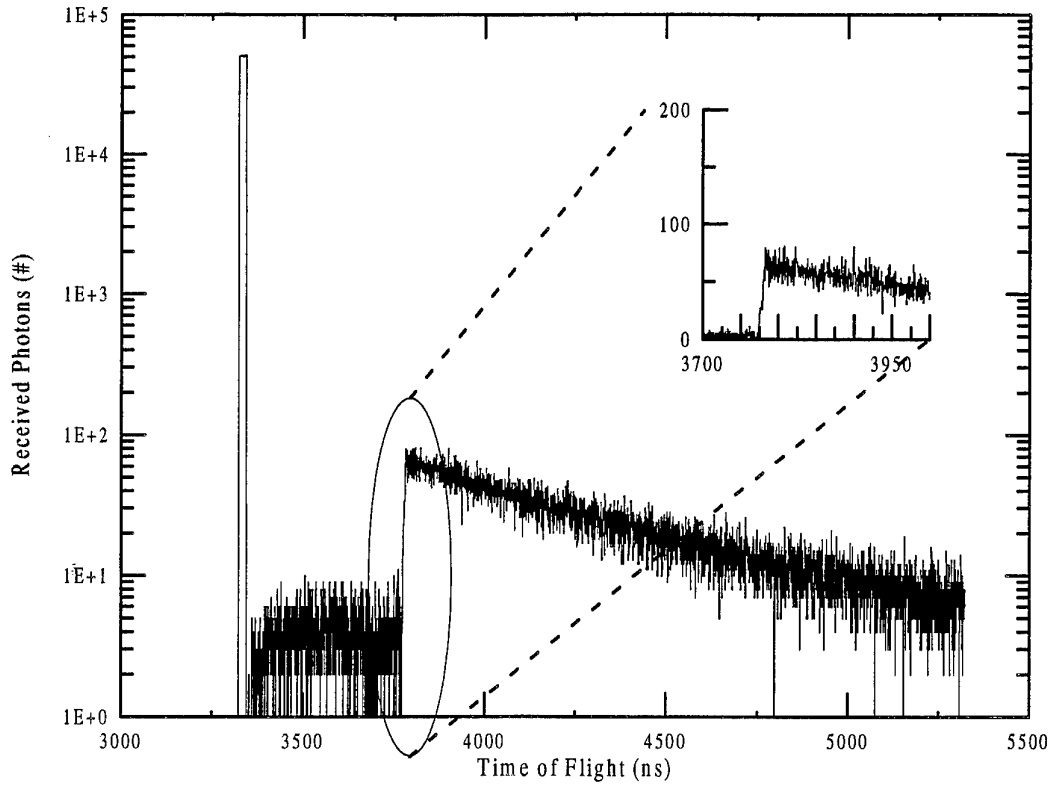


**Figure 3c.** Simulation details as for Figure 3a., except: photons propagate from 'flat top' source with 10 ns full width, at 500 m at nadir with zero beam divergence.

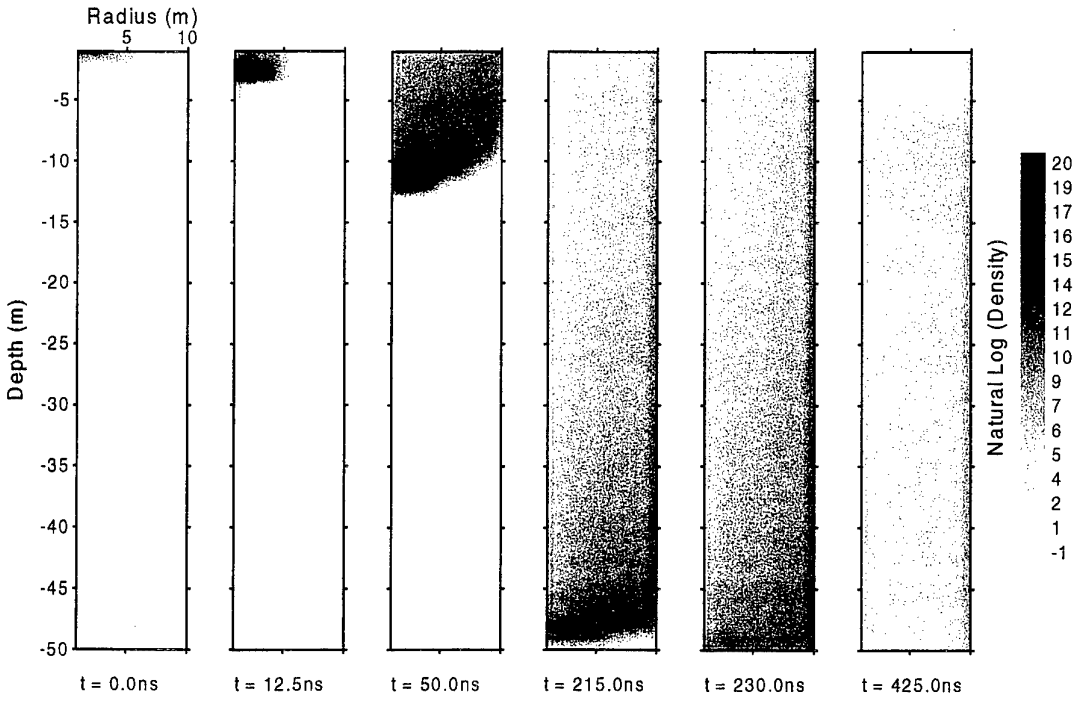


**Figure 4.** Evolution of the R-Z density profile for photons in water. 'Flat top' source, with 10 ns full width, at 500 m at nadir with 3m beam spot size at surface. Water scattering coefficient: 0.0022 $m^{-1}$. Suspended matter scattering coefficient: 0.0035 $m^{-1}$. Absorption coefficient: 0.0544 $m^{-1}$

# 5. IMPROVEMENTS TO THE SIMULATION

While the initial results provide some evidence of the numerical integrity of the simulation code, they have also highlighted some weaknesses. Several of these were identified and immediately corrected during the initial phase. Several weaknesses remain, however. Four of these are discussed below, along with solutions, which will be implemented shortly.

## 5.1 Rayleigh Scattering from Water

Close inspection of the data used to produce the frames in Figure 4, reveals that as the pulse 'head' approaches the bottom ($t$ = 215 ns), there is a significant depopulation of photons at small radial values. At later times still, the radial distribution of photons may be likened to a 'smoke ring' - a doughnut shape, propagating to wider radii with time. While experience suggests that this is un-physical behaviour, this is a characteristic of the Henyey - Greenstein scattering model and the initial Gaussian distribution of photon velocities. The Henyey - Greenstein distribution is used to model the angular scattering distribution from both water and suspended mater. Because this distribution is strongly forward biased, there is an unrealistically small backward scattering contribution, as indicated by figures 3a and 3c for example, which would 're-populate' the distribution at small radii, once the pulse head has passed. Light scattering from water is governed by fluctuations in the random uniform distribution of water molecules. This is well described by Rayleigh scattering and higher order processes.

The angular dependence of the differential scattering cross section for Rayleigh scattering follows a $1+ \cos^2\theta$ distribution, which is peaked at backward and forward angles. This dependence gives rise to the small backward peak in volume scattering functions characteristic of ocean waters, as shown in figure 5, where the angular dependence of a volume scattering measured in deep, clean, ocean water (850 - 870 fathoms, Lat: 24°29'N, Long: 77°33'W, [Petzold 1972]), is compared to a Henyey - Greenstein function and an appropriately normalised Rayleigh Scattering function.

Clearly, while a Henyey - Greenstein distribution satisfactorily describes the scattering for forward angles, at angles > 90 degrees, a Rayleigh distribution is required. In principle, any function can be used to describe the angular probability distribution of scattered photons, with varying degrees of 'realism', in the subroutine COLLIDE, described in section 3.2. To improve the efficiency of the code, the probability distribution for $\cos(\theta)$ for suspended scatterers, now evaluated in COLLIDE, using a Henyey - Greenstein function, is to be represented by a table of cubic spline coefficients. Each spline function represents an equal (small) fraction of the integral of the function from $0 \leq \theta \leq \pi$, which will be used to give an interpolated value for $\cos(\theta)$, depending on a random variable, as in the present case. The existence of the Henyey - Greenstein distribution will provide a definitive test for the integrity of this piece of code. A similar table will be used to represent $P(\cos(\theta))$ for the Rayleigh distribution for scattering from water. After the impact of these changes are assessed, the Henyey - Greenstein distribution for scattering from the suspended matter may be superseded by a more realistic measured volume scattering function. To achieve this, an appropriate Rayleigh dependence would be subtracted from a fitted functional representation of the discrete measured data. The principal advantage of using real volume scattering functions in this way, is that it represents a closer approximation to a multi-component suspension, while retaining the computational advantages of the single suspended scatterer representation.

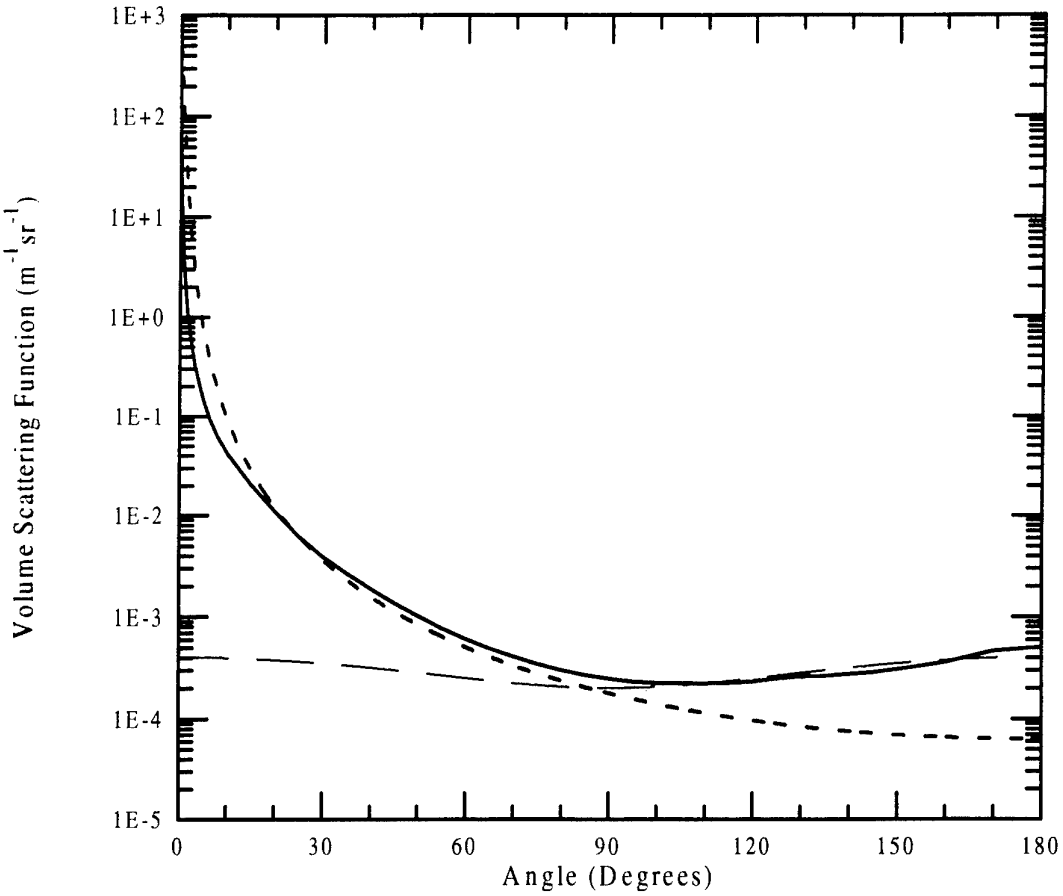


**Figure 5.** Measured angular dependence of the Volume Scattering Function ━━ for 'deep ocean water' (850 - 870 fathoms: station 8, Petzold 1972) compared with a Henyey - Greenstein distribution function for <cosθ> = 0.99 • • • , and a Rayleigh Scattering distribution - - - -

## 5.2 Finite Acceptance Angle Receiver

The shape of the decay of the signal following the appearance of the bottom in Figures 3a - 3c, is due to loss of signal through absorption alone, as the signal from an infinite field-of-view, is received to an infinite aperture at the source altitude. In reality, the LADS system has an effective receiving aperture of 180 mm, centred at the source and either a FOV which is closely matched to the initial laser spot size (6 mrad); the so-called 'narrow' FOV, or one of 40 mrad; the 'wide' FOV. The effect of the finite aperture - FOV combination is to sharply truncate the tail of the returned signal, as photons propagate through the water, away from the field of view of the aperture, to positions where they are significantly less likely to be scattered into the receiving aperture. Exact duplication of the physical receiving aperture / FOV in the simulation would result in an inevitable increase in the statistical uncertainty of the returned signal. The following approach is proposed, in part as a compromise between the slow statistical convergence of the simulation and the physical constraints of the LADS system.

Three receiver fields of view are defined from the center of the laser spot on the mean sea surface; a point $x_c$ on the x-axis

$$x_c = h \tan \theta$$

where h is the height of the source above the mean sea surface, and θ is the angle of the beam with respect to nadir. (see Figure 1.). The diameter of the first, 'narrow'

FOV, is identical to the spot size of the probe beam. The diameter of the second, or 'wide' FOV is identical to that found in the LADS system: defined by a 40 mrad wide cone, or 20m diameter spot at 500 meters. The third FOV is infinite in extent.

At the point of exit from the water, the radial position of the photon w.r.t. $x_c$ is used to assess which of the receiver fields of view: narrow, wide, or infinite, the exiting photon belongs to. The photon then propagates to the source altitude, where the final radius w.r.t. the source position, is compared with disks (apertures) of radius 0.1 m, 1 m, 10 m, 100 m and infinite. A similar comparison is made for apertures of the same dimensions, centred on $x_c$, (ie. Immediately above the point of entry into the water). The time of arrival of the photon at the source altitude is then added to the histogram for each, and any, of the arrays defined by the three FOVs and ten possible apertures.

In order to minimise simulation time spent following 'long path' photons, photons which have survived unabsorbed for longer than three 'direct path' surface-bottom-surface transits will be terminated and considered to have exited from the infinite FOV to the infinite aperture at that time, plus the source-surface-source transit with $x$ = 1000m and $y$ = 1000m. This should prove a considerable saving in simulation time, based on the evidence of Figure 4.

### 5.3 Data Storage

The changes mooted in the previous section require a review of the data structures used for storing the simulation data. Firstly, the maximum radius for the 'in water' arrays should be increased from the present 10 m to 20 m, to match the wide FOV 'in-air' data. Secondly, the array structures for the 'in-air' data need to be changed to reflect the change in the recorded data: 10x3 dimensional arrays with a histogram of Time-of-Flight data for the three FOVs in three columns, 'binned' in 0.5 ns elements.

The magnitude of the 'depth bias', or increase in optical path due to scattering, which this program is designed to study is of the order of one meter in 30 - 50 meters depth. The present 'in-water' data structure stores the density distribution profiles on a 1meter x 1meter grid. This was sufficient for the initial testing described in this report, but is not sufficiently sensitive to study the dependence of the depth bias on the turbidity. In order to perform such studies it is necessary to increase the resolution of the boxes to 0.1 meters. However, the size of the present type of data structure increases prohibitively. This problem can be overcome by utilising a new feature in Fortran90, that of variable structure arrays which have sizes which can be allocated and reallocated dynamically. Using this feature and pointers, an optimally dense data structure can be created, rather than the present structures, which were less than 40% filled in the initial tests. In keeping with the present practice, photons achieving a radius of greater than 20 meters, or surviving longer than three times the direct surface-bottom-surface transit will be boxed at 20 m. These approaches, while making the data structures somewhat less intuitive to interpret, will also reduce the number of elements to be written to file periodically, hence increasing overall execution speed. The approach of using pointers and variable structure arrays will also simplify efficient data storage for simulations for depths less than the current fixed value of 50 meters.

## 6. COMPARISONS WITH SURVEY DATA

During a LADS sortie (#235) over the Sahul Banks in the Timor Sea, west of Darwin, in June 1996, a single line (Frame 84) was flown in clear water, with the automatic

gain control disengaged, to obtain data for qualitative comparison with the simulations. The values of the photomultiplier gain control parameters, G2 and G3, were set manually, such that the bottom return signals were un-saturated for depths deeper than about 30m. The gain was constant and fixed for all scan angles. The surface returns were allowed to saturate. At present it is not possible to directly compare the simulation with the Sahul Banks data, because the LADS system has a finite receiving aperture, whereas the present simulation integrates across an infinite receiver at the source. However, some features in the real data are worthy of note, as they may bear on the future direction of the simulations. It is clear from examples of data on flat bottoms, below 30m, that there is little, if any, degradation of the bottom return signal strength at angles away from nadir. This behaviour, may indicate the importance of relatively small amplitude wave action in the amplitude of the returned signal for scan angles other than nadir (The trial was conducted in relatively flat, but windy seas.

It was discovered during the sortie, that it was difficult to maintain constant gain across the time of the depth return, using independent manual control of G2 and G3. The exercise has recently been successfully repeated.

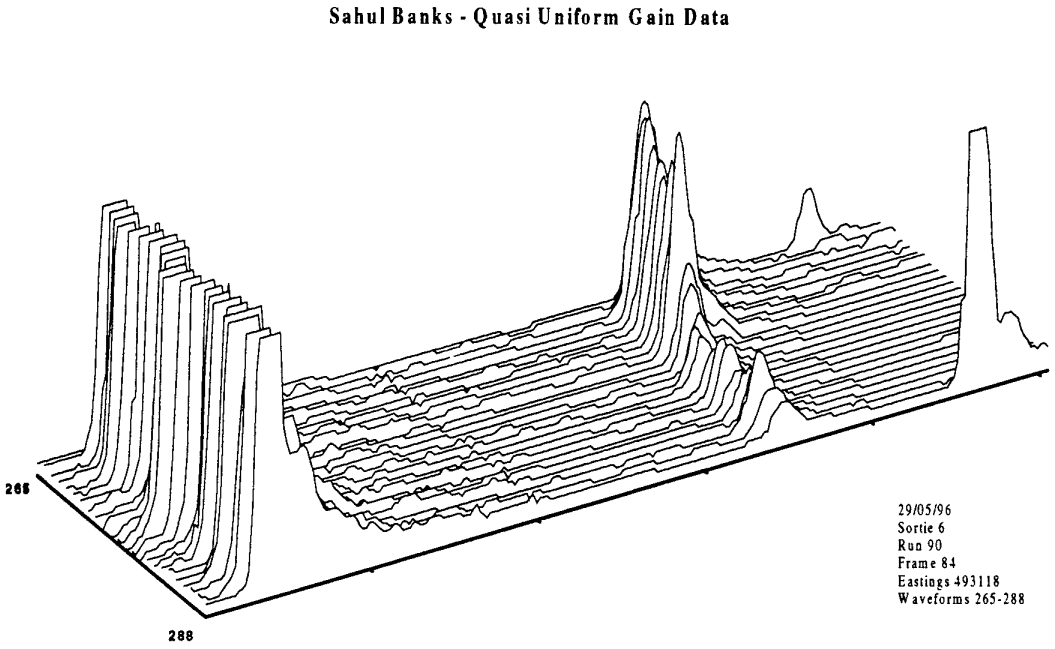


**Figure 6a.** LADS swath data. 29/5/96 Sortie #235, Run 90, Frame 84, Waveforms 265 - 288. Note the saturated surface returns and the large dynamic range in the bottom returns from depths of ~ 35m. The deep pulses on waveforms 265 and 288 are calibration points injected into the green receiver.

**Figures 6a and 6b show data from two swath scans over a shoaling bottom, separated by approximately 150 meters. Examined together, it is evident that the larger amplitude returns (adjacent waveforms 265 and 169), are from a flatter, slightly deeper bottom, whereas the shallower waveforms (adjacent waveforms 288 and 192) are smaller in amplitude. It is likely that the change in amplitude is due to an intrinsic decrease in bottom albedo with decreasing depth. This data provides an interesting point of comparison for the simulations with non-trivial bottom topographies.**

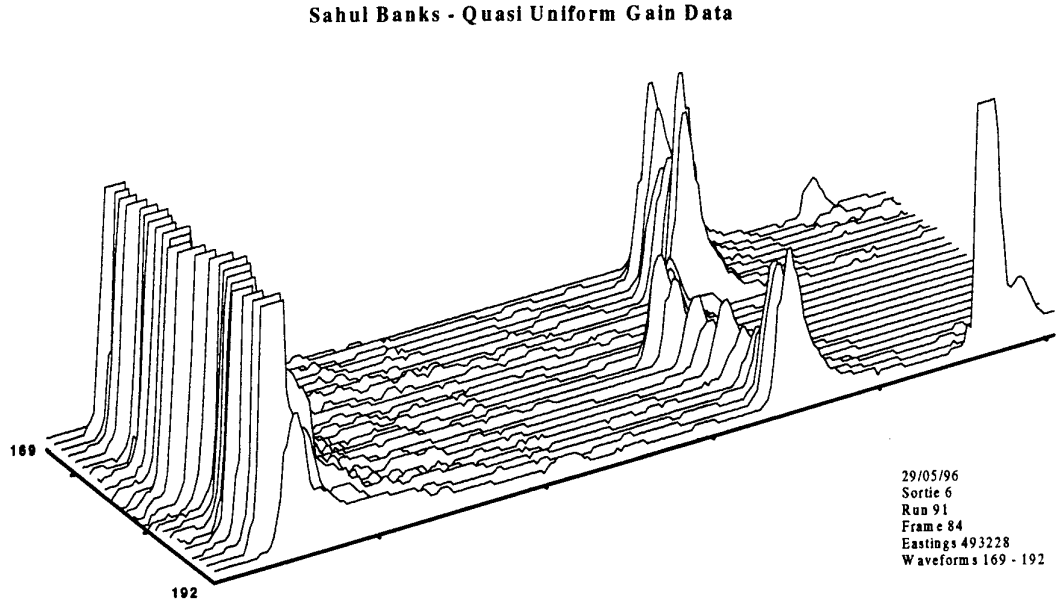


**Figure 6b.** LADS swath data. 29/5/96 Sortie #235, Frame 84, Run 91, Waveforms 169 - 192. Note that the smaller bottom returns occur at shoaler depths than the larger returns for waveforms 169 - 175.

# 7. EXTENSIONS TO THE SIMULATION

Three additional features are planned for the simulation. The proposed interpretation of each of these concepts is discussed below.

## 7.1 Provision for Wave Action

The function **SURFACE** (Appendix 3) is used to define the z value of the surface when called. At present it is a trivial function, returning zero (the mean surface level). At least three considerations are required to introduce wave action into this function. The first is a definition of the wave itself. A simple proposal, is to define a wave in terms of a single dimension in the $x$ direction (parallel to the scanned beam). The wave is defined by user input as a sinusoidal function, with a maximum amplitude, a wavelength in $x$ and phase, relative to the nadir position in $x$. A photon which is projected to strike the mean sea surface at $x = x_0$, from either above or below the surface, instead strikes the surface at a height defined by the wave. At present it is not expected that an exact re-calculation of the corresponding $x$ position will be required, given the arbitrary nature of the approximation. The principal focus is on the change in z and the surface normal, due to wave action. This approach should be particularly suitable for long wavelength swells. The local derivative of the wave surface will define the tangent and the angle of incidence for the purpose of Snell's Law.

An alternative description, which may be both faster to implement computationally and provide a more realistic description, is to describe the distribution of wave slopes at any point and time using a two dimensional Gram Charlier distribution, together with a Pierson-Moskowitz spectrum which are reported [Gambling 1973, Cox & Munk 1954, Dietrich & Wegener 1996] to provide a useful description of the distribution of ocean wave slopes and amplitudes. This approach could be

implemented by choosing a suitably weighted random number. In this case, some distribution should be used for defining the tangent to the wave surface.

## 7.2 Provision for Non-Trivial Bottoms

The ability of LADS to survey shallow and reef areas is an important operational advantage in Australia's coastal waters. The LADS surveys of reef areas are characterised in part by a significant number of soundings from isolated single shoals, due to coral heads ('bommies') and other narrow features. While LADS has surveyed many of these features, it is desirable that the simulation is able to at least qualitatively reproduce the waveforms observed from such features. With this extension to the simulation, it is possible to investigate issues relating to so-called 'bottom coverage'. It is necessary for example to investigate how the returned signal from a sharp bottom feature varies, depending on the whether beam strikes the side, foot or top of the feature. By analysing the shape of the returned signal it may be possible to determine if LADS has recorded the 'shoalest' depth, or whether the feature deserves further overflight. In addition, it will be possible to investigate the dependence and sensitivity of the LADS system to depth, spots size and spot spacing. Specifically, how the interplay of these parameters may effect the ability of the system to resolve separate adjacent features and to investigate to what degree the presence of a sharp bottom feature 'shadows' the adjacent bottom.

A simple method for providing an irregular bottom in the simulation is to prompt the user for 2 sets of 4 (x, *y*, z) triplets, defining the coordinates of the base and top of the feature respectively. In the bulk of the medium, that is below the surface and above the shoalest depth of the bottom feature, the simulation proceeds as before, regardless of the bottom. At depths deeper than the shoalest, the program must test whether the interim $x$ and $y$ coordinates calculated in **WHERE** lie inside or outside of the feature, probably, most economically defined by the base coordinates. If the interim coordinates lie within the feature, then, as now for the 'flat bottom', the program determines the time to intercept the plane defining one side of the feature and scattering takes place at that time, with the scattering surface defined as the appropriate plane of the feature. The process of transforming between coordinate frames is somewhat computationally expensive, however, by providing earlier conditional tests, this is usually performed at most once per photon. Note that different scattering albedos for the feature may be defined.

## 7.3 Scattering of Polarised Light / Depolarisation of Light

In the present LADS system, the polarisation of the source beam plays an important role. The light from the green laser is polarised orthogonal to the wings of the plane. The reflection from the sea surface is approximately specular. Thus, the direction of polarisation of the fraction of the beam reflected back to the receiver is preserved. A polarising filter, which is oriented orthogonally to the direction of polarisation of the returned beam, is placed in front of the 'green beam' receiver. This has the effect of strongly reducing the intensity of polarised light reaching the receiver, and thus reduces the problem of immediate saturation of the receiver electronics by the surface reflected light. However, if the suspended particles are large compared with the wavelength of the incident radiation and sufficiently smooth, the light backscattered at 180 degrees is also specular and thus photons reaching the receiver which have been backscattered from suspended particles will also be strongly attenuated. This is of concern because the strength of the backscattered signal and the shape of its subsequent decay is used as a measure of the turbidity of the water and provides a important parameter in depth bias modelling.

The degree and direction of polarisation of light may be defined by the Stokes parameters. In general, scattering of the beam results in both a reduction of the degree of polarisation and a change in the state of polarisation. A description of scattering of polarised light in terms of incident and scattered Stokes parameters can be found in Bohren and Huffman [1983].

To simulate polarisation of the incident and returned beams the following approach is proposed. The light from the source is considered to be 100% linearly polarised in the $y$ direction. A photon exiting the source will carry the appropriate Stokes vector (1,1,0,0). Each scattering event (including surface and bottom reflections) modifies the Stokes vector appropriately, according to the scattering, or Muller matrices. In the present simulation, if the photon is returned to the receiver, the position of the photon in $x$, $y$, and t is boxed and added to a 3D histogram. In the extension to polarisation, the $x$, $y$, t histogram at the receiver will be extended to include the Stokes vector. Thus, average values for each of the parameters as a function of space and time can be obtained. From this information, the required polarisation state may be readily derived in a modified post-simulation algorithm.

# 8. PROGRAM INPUT

## 8.1 Simulation Code

### 8.1.1 Explicit

The initial input to the simulation is divided into sections. At present these comprise a File information section, followed by Beam, Water and Bottom information sections. When wave action or other features discussed above are included, then appropriate input sections will be introduced. In the File information section, the user is asked for the names of 4 files in which information is to be stored:

- Run information output: This is essentially the file which indicates the status of the simulation - the number of photons output, CPU time consumed, collisions processed etc. The information in this file is largely self explanatory.
- Boxed $x$, $y$, t output at plane .
- Boxed r, z, t output in water, where r is the radius: $\sqrt{x^2 + y^2}$
- Run parameter file name: This is the auxiliary file read by the processing program, containing information concerning the run.

Input defining the beam parameters is then requested. The user enters

- The aircraft height in meters.
- The full temporal width of laser beam in nanoseconds.
- The beam scan angle from nadir in degrees. Note that in the LADS system, the maximum angle is 15 degrees, however, no such limit is set in the simulation.
- The beam spot diameter at the sea surface meters, representing the width of a Gaussian beam encompassing 86% of the total beam intensity.

Input defining the attenuation and scattering of the light in water follows. The user enters

- The photon absorption coefficient, a, in $m^{-1}$
- The water scattering coefficient, $b_{ray}$, in $m^{-1}$
- A value for average cosine for water, as represented by the Henyey-Greenstein distribution (typically 0.95 - 0.99).
- The suspended matter scattering coefficient, $b_{sus}$, in $m^{-1}$.

- A value for average cosine for the suspended scatterer, as represented by the Henyey-Greenstein distribution (typically 0.95 - 0.99).
- The bottom depth in meters is entered, together with the reflectivity of the bottom, expressed as a percentage
- Lastly the user is prompted for the total number of photons to be simulated and the number of photons to be processed between calls to the routines which save intermediate data to file.

The simulation can be run either interactively as describe above, or as a batch job. In the later case, under Unix, the appropriate command is:

nohup photon.exe < input.file > log.file &,

where photon.exe is the name of the executable file, input.file is a file containing the information otherwise entered interactively through stdin and log.file contains the output re-directed from stdout.

### 8.1.2 Implicit

In this version of the simulation, the form of the scattered photon angular distribution is given by a single analytic function. The functional form used is a Henyey-Greenstein distribution, with a variable, separate, width for both photon-water and photon-suspended matter collisions, given by a user input value for g, the mean value of $\cos\theta$. Henyey-Greenstein distributions have been widely for describing the $\theta$ dependence of light scattering from suspended matter used [see for example Groenhuis *et al* 1983 and Bergougnoux *et al* 1996 and refs therein]. As described above, the $\phi$ dependence is uniform over $2\pi$. Comparison with measured Volume Scattering functions for selected ocean waters [Petzold 1972] suggests that values of g between 0.8 and 0.99 give an appropriate representation of scattering of 532 nm radiation from suspended matter. This proved a useful approximation for scattering from water as well during testing. However, such a function significantly under-estimates the backscattering from the random fluctuations in water, which is governed by Rayleigh scattering (section 5.1).

# 9. Acknowledgments

Many people have contributed to the successful development of the code. However, special thanks are due to Ralph Abbot, Derek Bertilone, David Cartwright, Dallas Lane, Martin O'Connor and Bob Whatmough for their contributions to the interpretation of the physics presented, and to Bruce Bennett, Dave Hemming, Richard Watts and the staff of CISU for their efforts in assisting with the purchase, installation and optimisation of the DECalpha used in the simulations.

# 10. References

Bergougnoux, L., Misguich-Ripault, J., Firpo, J-L., and André, J. (1996) Monte-Carlo calculation of backscattered light intensity by suspension: comparison with experimental data, *Applied Optics*, **35**, 1735

Billard, B. (1986) Remote sensing of scattering coefficient for airborne laser hydrography, *Applied Optics*, **25**, 2099

Bohren, C.F. and Huffman, D.R., (1983), Absorption and scattering of light by small particles, Wiley Interscience (Brisbane)

Brennan, M.J., (1991) Optimization of Monte-Carlo codes using null collision techniques for experimental simulation at low E/N, *IEEE Trans. Plasma Science,* **19,** 256

Brent, R., (1996) *private communication*

Cox, C. and Munk, W. (1954) Measurement of the roughness of the sea surface from photographs of the sun's glitter, *J. Opt. Soc. Am.* **44,** 838

Fernee, M., (1995) Report on LADS depth bias simulation using Monte-Carlo simulations, *LSOD-95-09-WP*, ESRL

Gambling,D.J., (1973) Sun glitter on the surface of the ocean in the infrared spectral region, *WRE-TN-892* , ESRL

Gordon, H.R., (1982) Interpretation of airborne oceanic lidar: effects of multiple scattering, *Applied Optics*, **21**, 2996

Groenhuis, R.A.J., Ferwerda, H.A., and Ten Bosch, J.J., (1983) Scattering and absorption of turbid materials determined from reflection measurements I: Theory, *Applied Optics*, **22**, 2456

Joelson, B.D. and Kattawar, G.W. (1996) Multiple scattering effects on the remote sensing of the speed of sound in the ocean by Brillouin scattering, *Applied Optics*, **35,** 2693

Kaijser, T.J., (1990) A Monte Carlo simulation of airborne hydrographic laser systems, FOA report C 30578-8.1.

Koerber, B., (1996) Mathematical modelling of optical image propagation in water, *DSTO-TR-0150* , ESRL

Petzold, T.J., (1972), Volume scattering functions for selected ocean waters, *SIO Ref. 72-78* (Scripps Institute of Oceanography, San Diego, Calif., 1972).

Poole, L.R., Venable, D.D. & Campbell, J.W. (1981) Semi-analytic Monte Carlo radiative transfer model for oceanographic lidar systems, *Applied Optics*, **20**, 3653

Tootill, J.P., Robinson W.D. and Eagle J., (1973). An asymptotically random Tausworthe sequence. *J.A.C.M.* **20**, 469

Dietrich, C.R. and Wegener, M., (1996). Generation of random sea surfaces satisfying a Cox-Munk distribution for wave slopes and Pierson-Moskowitz spectrum for wave height. *Pre-print*

# Appendix 1

The program in this appendix is current as of 05-August-96.

## modules.f90

```
! ********************* MODULE SECTION ********************
!
      MODULE PH_LOGIC
         SAVE
         LOGICAL(1) ph_finish
      END MODULE PH_LOGIC
! **********************
      MODULE BOXES
         SAVE
         INTEGER(4) it_entry, ir_entry
! Max dimensions are referenced in main and several other routines
         INTEGER(4), PARAMETER :: ix_plane = 25, jy_plane = 2, kt_plane = 4000
         INTEGER(4), PARAMETER :: ir_rzt = 10, jz_rzt = 50, kt_rzt = 4000
! Boxed arrays for plane and water
         INTEGER(4) boxplane(ix_plane, jy_plane, kt_plane)
         INTEGER(4) box_rzt(ir_rzt, jz_rzt, kt_rzt)
! center position of photon cone in water
! ... x_center = plane*sin(scan_angle)
         REAL(8)x_center
! minimum ToF for returned photon
         REAL(8) plane_mint, water_mint
      END MODULE BOXES
! **********************
      MODULE RAN_PAR
         SAVE
         REAL(8) ai, aj, ak, al, am
! running sums of random numbers
         REAL(8) callsi, callsj, callsk,callsl, callsm
! number of calls to generator
         INTEGER(2) is, js, ks, ls, ms
! seeds
         INTEGER(4) ris(607), rjs(607), rks(607), rls(607), rms(607)
! random sequence arrays
      END MODULE RAN_PAR
! **********************
      MODULE MONITORS
         SAVE
! process monitors:
! number of total, absorbing, water and suspended scattering events
! and number of end treatments for radial and temporal boxes at the plane
! (x?p_hot) and in the water (x?w_hot)
         REAL(8) xtotal, xabsorb, xwater, xscatt, xr_hot, xx_hot, xy_hot, &
                 xtp_hot, xtw_hot
      END MODULE MONITORS
! **********************
      MODULE PHASE
         SAVE
         INTEGER(4) it
         REAL(8) x, y, z, Vx, Vy, Vz, V, t, t0
      END MODULE PHASE
! **********************
      MODULE OUT_PAR
         SAVE
```

```
          INTEGER(4) i_output, m_photons, n_total, n_output
        END MODULE OUT_PAR
! *********************
        MODULE FILES
          SAVE
          CHARACTER(40) finfo, fPlane, frzt, frun
        END MODULE FILES
! *********************
        MODULE EXTREMA
          SAVE
! maximum radial extent in water, max x and y at plane, min and max
! temporal extents in water (1,2) and plane (3,4)
! note: x and y max are allowed to be negative
          REAL(4) r_max(3), t_max(4)
        END MODULE EXTREMA
! *********************
        MODULE BEAM_PAR
          SAVE
          REAL(8) plane, laser_width, scan_angle, spot, sigma
          REAL(8) laser_time
        END MODULE BEAM_PAR
! *********************
        MODULE SURFACE_PAR
          SAVE
!         REAL(8) surf_reflect
        END MODULE SURFACE_PAR
! *********************
        MODULE WATER_PAR
          SAVE
! total attenuation coefficient
          REAL(8) atten_coeff
! calculated mean free time in water
          REAL(8) time_mean
          REAL(8) a_coeff, w_coeff, g_water, s_coeff, g_scatt
! values of scattering coeffs for output
          REAL(4) a_out, w_out, gw_out, s_out, gs_out
        END MODULE WATER_PAR
! *********************
        MODULE BOTTOM_PAR
          SAVE
          REAL(8) depth, bott_reflect
        END MODULE BOTTOM_PAR
! *********************
        MODULE SEA_AIR
          SAVE
          REAL(8) costh, sinth_sq, water_n, water_n_sq
! refractive index and cos theta for incident and transmitted media
          REAL(4) n_i, n_t, costh_i, costh_t
        END MODULE sea_air
! *********************
        MODULE SLICE
! temporal slice/bin size
          REAL(8) dt_slice
          DATA dt_slice/5.d-10/
        END MODULE SLICE
!
! ******************* END MODULE SECTION ******************
```

# Appendix 2

The program in this appendix is current as of 05-August-96.

## 960805.f90

```
! 960805.f90
! Anisotropic program for simulation of photon transport in sea water
! including air-sea and sea floor boundaries
! provision is made for future extension to consider periodic surface
! topology and arbitrary bottom topology
!
! The program has been written with the following features ...
!   All calculations are in SI units ... meters and seconds
!   The speed of the photons in air is set to be 3x10^8 m/s
!   The speed of the photons in water is set to be 2.25x10^8 m/s,
!     where the ratio 1.3333333 is the assumed index of refraction
!     for the sea water
!   The following full Cartesian geometry (x,y,z) has been adopted:
!     z = 0 at mean sea level. z = +plane at plane height.
!   Photons propagate initially in the -z direction, are reflected and
!       propagate to +plane.
!   Plane notionally propagates in the +y direction
!   Photons are scanned out in the +x direction.
!   The mean collision frequency is set by a grand total collision cross
!    -section composed of the following:
!   There are three possible interactions of the photons with the water
!   - Absorption: mean free time set by mean free path for process x
!         lambda_ = 1/coeff_x ... a = 0.05 '20 meter water'
!   - note that I(z) = I_0 exp(-lambda_x z) is analogous to
!                     I(t) = I_0 exp(-MFT_x t)
!       where lambda is the mean free distance
!       where the speed of the photons in the absorbing media implies
!
!                 v = lambda_a/MFT_a
!
!   - Scattering from water: ... MFT set as above ... Anisotropy from
!       Henyey-Greenstein, using average cos theta g ~ 0.95 (variable)
!   - Scattering from suspended matter: - characterised by single MFT
!       set as above
!
!   - That is the effective density and properties of the water as an
!       absorber and scatterer are set separately, as is the density of
!       the scatterer
!
!   Laser beam temporal width is variable ...
!     characterised as a temporal 'flat top'
!   Beam divergence is variable
!     Initial photon velocity cosines set by observing that the spot
!     size is 2x the spot radius, which represents the 1/e point for
!     the photon intensity
!     The photon distribution is modelled by a Gaussian characterised
!     by the spot radius
!
!   Sea state is flat ... Snell's Law is used on photon entry and exit
!** Photons are reflected from the sea surface at entry and exit
!       using Fresnel's Equations.
!
!   Bottom state is flat ...
```

```
!** Photons are reflected from the bottom with a variable probability
!       Cosine distribution is used in theta and 0-2Pi in Phi
!
!   Photon positions are recording in r,z,t in boxes, ...
!       -       boxplane x (1 m boxes), y (1 meter boxes)
!                  and t (.5 ns boxes) at z = plane meters
!                  t is offset to be at box 1 for the first non-zero time
!       -       box_rzt: 1m x 1m x0.5ns, centred around water entry point
!
! created for DECalpha May-July 1996 MJB
! in collaboration with Robert Whatmough, Derek Bertilone and Ralph Abbot
! change log started 060696
!       060696 Variables are described in declaration and/or first
!       assignment
!       170796 1 m boxes in z for initial debugging
!       180796 split monolithic program file into the following components
!               1) date-named file ... contains main and routines called by
!                  main.
!               2) modules.f90 contains modules
!               3) level1.f90 contains routines called by subroutines in
!                  main program
!               2) level2.f90 contains routines called by subroutines in
!                  level1.f90 and other routines in level2.f90
!               A daily directory structure contains files
!               first attempt at array compacting ... at plane, it = 1
!                 corresponds to a min ToF for given beam config
!               10 m boxes in x and y at plane for debugging ...
!       200796 First 'release' version ... cos theta set to 1 in scattering
!               and at bottom (phi = 0) for debugging
!       210796 Introduce parameters for array dimension
!       250796 Timing boxing verified through testing
!       260796 Correctly treat total internal reflection in level1.f90
!               Edited for Specular bottom 13:16
!       300796 Back to diffuse bottom
!       020896 Added treatment for z = local_b in where
!               cleaned up a few errors and bugs found since 300796
!               NOT under RCS yet!
!
! modules used
        USE ph_logic
        USE boxes
        USE ran_par
        USE monitors
        USE phase
        USE out_par
        USE files
        USE extrema
        USE beam_par
        USE water_par
        USE bottom_par
        USE sea_air
        USE slice
!       USE portlib
        IMPLICIT NONE
! main program bits and pieces
! loop indices
        INTEGER(4) i,j,k
! loop index
        INTEGER(4) ii
! time between collisions
```

```
      REAL(8) delta_t
! absolute time of next collision
      REAL(8) tck
! absolute time of next time bin
      REAL(8) tcut
! exclusive random number function
      REAL(4) randt
! random variable - standard random number intermediate
      REAL(4) r1
! machine specific timing
      REAL(4) tarray(2)
      REAL(4) cpu, old
! timing
      CHARACTER(8) tbuf
      CHARACTER(9) dbuf
      EXTERNAL ETIME
      REAL(4) ETIME
! fail safe handlers
      INTEGER(2) ifail1,ifail2
!
! Start the clock
      CALL date(dbuf)
      CALL time(tbuf)
      WRITE(*,*)tbuf , ' on ' , dbuf
      cpu = ETIME(tarray)
! Read in seeds and initialise random sequences
11      OPEN(unit = 15 , file = 'seeds.dat' ,
     &
          STATUS = 'unknown' , err = 1011)
      READ(15,*)is , js , ks , ls , ms
      CLOSE(15)
      IF((is.eq.js).OR.(is.eq.ks).OR.(is.eq.ls).OR.(is.eq.ms))THEN
        WRITE(*,*)'Check last run: Seeds Equal!!'
        STOP
      END IF
      IF((js.eq.ks).OR.(js.eq.ls).OR.(js.eq.ms))THEN
        WRITE(*,*)'Check last run: Seeds Equal!!'
        STOP
      END IF
      IF((ls.eq.ms))THEN
        WRITE(*,*)'Check last run: Seeds Equal!!'
        STOP
      END IF
12      OPEN(UNIT = 15 , FILE = 'ris607.dat' ,
     &
          FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
      DO 5000 ii = 1 , 607
        READ(15)ris(ii)
5000  END DO
      CLOSE(15)
      OPEN(UNIT = 15 , FILE = 'rjs607.dat' ,
     &
          FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
      DO 5010 ii = 1 , 607
        READ(15)rjs(ii)
5010  END DO
      CLOSE(15)
      OPEN(UNIT = 15 , FILE = 'rks607.dat' ,
     &
          FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
```

```
      DO 5020 ii = 1 , 607
         READ(15)rks(ii)
5020  CONTINUE
      CLOSE(15)
      OPEN(UNIT = 15 , FILE = 'rls607.dat' ,
     &
           FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
      DO 5030 ii = 1 , 607
         READ(15)rls(ii)
5030  CONTINUE
      CLOSE(15)
      OPEN(UNIT = 15 , FILE = 'rms607.dat' ,
     &
           FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
      DO 5032 ii = 1 , 607
         READ(15)rms(ii)
5032  CONTINUE
      CLOSE(15)
20       FORMAT(I1)
      WRITE(*,*)'*** ENTER THE FOLLOWING FILE INFORMATION ***'
      WRITE(*,*)'ENTER RUN INFORMATION OUTPUT FILE NAME'
      READ(*,30)finfo
      WRITE(*,*)finfo
      WRITE(*,*)'ENTER BOXED x,y,t OUTPUT FILE NAME FOR PLANE HEIGHT'
      READ(*,30)fplane
      WRITE(*,*)fplane
      WRITE(*,*)'ENTER BOXED r,z,t OUTPUT FILE NAME FOR WATER'
      READ(*,30)frzt
      WRITE(*,*)frzt
      WRITE(*,*)'ENTER RUN PARAMETER FILE NAME'
      READ(*,30)frun
      WRITE(*,*)frun
30       FORMAT(A40)
!
!  **********************VARIABLES SET*****************************
!
      ai = 0.0d0                !
      aj = 0.0d0                !}
      ak = 0.0d0                !}
      al = 0.0d0                !}
      am = 0.0d0                !}
      callsi = 0.0d0            !}random check
      callsj = 0.0d0            !}
      callsk = 0.0d0            !}
      callsl = 0.0d0            !
      callsm = 0.0d0            !
      xtotal = 0.0d0            ! total collisions monitor
      xabsorb = 0.0d0              ! absorbing collisions monitor
      xwater = 0.0d0               ! scattering collisions with water
                                ! monitor
      xscatt = 0.0d0            ! scattering collisions with suspension
                                ! monitor
      xr_hot = 0.0d0            ! end treatment radial monitor
      xx_hot = 0.0d0            ! end treatment X monitor
      xy_hot = 0.0d0            ! end treatment Y monitor
      xtp_hot = 0.0d0              ! end treatment temporal monitor at
                                ! plane
      xtw_hot = 0.0d0              ! end treatment temporal monitor in
                                ! water
      m_photons=0               ! total number of photons modelled
```

```
      i_output=0                ! number of modelled photons since the
                                ! last write
      water_n = 3.0d8/2.25d8    ! refractive index of water
      water_n_sq = water_n*water_n
! set exit time of photon from laser to zero
      laser_time = 0.d0
! **** SETUP ARRAYS ****
      DO i = 1, ix_plane
         DO j = 1, jy_plane
            DO k = 1, kt_plane
               boxplane(i, j, k)= 0
            END DO
         END DO
      END DO
      DO i = 1, ir_rzt
         DO j = 1, jz_rzt
            DO k = 1, kt_rzt
               box_rzt(i, j, k)= 0
            END DO
         END DO
      END DO
      DO i = 1, 3
         r_max(i) = 0.
      END DO
      t_max(1) = 3.e8
      t_max(2) = 0.
      t_max(3) = 3.e8
      t_max(4) = 0.
!
! ***************** ENTER RUN PARAMETERS *****************
!
      WRITE(*,*)' ENTER THE FOL'.OWING PARAMETERS:'
      WRITE(*,*)' FOR THE BEAM:'
      WRITE(*,*)''
      WRITE(*,*)' PLANE HEIGHT IN METERS'
      READ(*,*)plane
      WRITE(*,*)REAL(plane)
      WRITE(*,*)' FULL TEMPORAL WIDTH OF LASER BEAM (ns)'
      READ(*,*)laser_width
      laser_width = laser_width*1.d-9
      WRITE(*,*)REAL(laser_width)
      WRITE(*,*)' BEAM SCAN ANGLE FROM NADIR IN DEGREES - LADS Max: 15'
      READ(*,*)scan_angle
      WRITE(*,*)REAL(scan_angle)
      WRITE(*,*)' BEAM SPOT DIAM AT SURFACE (86% INTENSITY)(METERS)'
      READ(*,*)spot
      WRITE(*,*)REAL(spot)
!
!     WRITE(*,*)''
!     WRITE(*,*)' FOR THE SEA SURFACE :'
!     WRITE(*,*)''
      WRITE(*,*)' FOR THE WATER :'
      WRITE(*,*)''
      WRITE(*,*)' PHOTON ABSORPTION COEFFICIENT'
      READ(*,*)a_coeff
      a_out = REAL(a_coeff)
      WRITE(*,*)REAL(a_coeff)
      WRITE(*,*)' WATER SCATTERING COEFFICIENT'
      READ(*,*)w_coeff
      w_out = REAL(w_coeff)
```

```
      WRITE(*,*)REAL(w_coeff)
      WRITE(*,*)' AVERAGE COSINE FOR WATER'
      READ(*,*)g_water
      gw_out = REAL(g_water)
      WRITE(*,*)REAL(g_water)
      WRITE(*,*)' SUSPENDED MATTER SCATTERING COEFFICIENT'
      READ(*,*)s_coeff
      s_out = REAL(s_coeff)
      WRITE(*,*)REAL(s_coeff)
      WRITE(*,*)' AVERAGE COSINE FOR SCATTERER'
      READ(*,*)g_scatt
      gs_out = REAL(g_scatt)
      WRITE(*,*)REAL(g_scatt)
      WRITE(*,*)''
      WRITE(*,*)' FOR THE BOTTOM :'
      WRITE(*,*)''
      WRITE(*,*)' BOTTOM DEPTH IN METERS'
      READ(*,*)depth
      WRITE(*,*)REAL(depth)
      WRITE(*,*)' PERCENTAGE PHOTON REFLECTANCE FROM BOTTOM'
      READ(*,*)bott_reflect
      bott_reflect = bott_reflect/100.d0
      WRITE(*,*)REAL(bott_reflect)*100.
      WRITE(*,*)
      WRITE(*,*)'*********************************************************'
      WRITE(*,*)
      WRITE(*,*)' ENTER TOTAL NUMBER OF PHOTONS TO BE SIMULATED'
      READ(*,*)n_total
      WRITE(*,*)n_total
      WRITE(*,*)' ENTER NUMBER OF SIMULATED PHOTONS BETWEEN OUTPUTS'
      READ(*,*)n_output
      WRITE(*,*)n_output
! **** SETUP DEPENDENT VARIABLES ****
      CALL dep_var
!
! ***************** CALCULATION BEGINS *****************
!
      cpu = ETIME(tarray)
      CALL time(tbuf)
      old = cpu
      WRITE(*,*)'Calculation Start          :'                    &
              , cpu/60. , 'CPU (min) @ ' , tbuf
! MAIN LOOP
! ************************************************************************
! ************************************************************************
! ************************************************************************
slrn:  DO WHILE (m_photons.LT.n_total)    ! logic check 11/7/96
! ************************************************************************
! ************************************************************************
! ************************************************************************
! note: m_photons traces the total number of modelled photons
! go to output if necessary
        IF(i_output.EQ.n_output)THEN ! logic check 11/7/96
          cpu = ETIME(tarray)
          CALL time(tbuf)
          WRITE(*,*)'Outputing to Files       :'                  &
                  , cpu/60. , 'CPU (min) @ ' , tbuf
          CALL output
          i_output=0
! reset - incremented in BEGIN
```

```
            cpu = ETIME(tarray)
            old = cpu
          END IF
! begin the photon, find initial phase space position at entry below
! mean sea surface, increment counters etc.
          CALL BEGIN
          IF(ph_finish)THEN
!           the photon has been reflected at sea surface, or otherwise
!           finished start new photon - ie. drop through to end of loop
            CYCLE sim          ! next photon
          ELSE
! set t0 = t
            t0 = t
          END IF
          cpu = ETIME(tarray)
          CALL time(tbuf)
! ************************************************************************
! ************************************************************************
between: DO                    ! between collision, until 'lost' loop
! ************************************************************************
! ************************************************************************
! throw delta_t to next collision
            r1 = randt(is , ris)
            ai = ai + DBLE(r1)
            callsi = callsi + 1.0d0
! update random monitors
            delta_t =  - time_mean*DLOG(DBLE(r1))
! total simulated time to next collision for this photon
!
            tck = t + delta_t
! ************************************************************************
slices:  DO                ! trace loop
! ************************************************************************
! time to next interrogation slice
              tcut = DBLE(it + 1)*dt_slice
              IF(tck.LT.tcut)THEN
! photon crosses no slice before collision - no problem
                t = tck
! time equals time collision
                CALL WHERE
                IF(ph_finish)THEN
                  CYCLE sim  ! next photon
                END IF
! go straight to collision
                CALL collide
                IF(ph_finish)THEN
                  CYCLE sim  ! next photon
                END IF
! - choose next collision time
                CYCLE between ! cycle to top of loop
              ELSE IF(tck.EQ.tcut)THEN
! collides at time slice
! update slice counter
                it = it + 1
! time equals time at 'next' slice
                t = tcut
                CALL WHERE
                IF(ph_finish)THEN
                  CYCLE sim  ! next photon
                END IF
```

```
! trace photon at slice
                    CALL trace_water
                    CALL collide
                    IF(ph_finish)THEN
                       CYCLE sim  ! next photon
                    END IF
! leave the loop - choose next collision time
                    CYCLE between ! cycle to top of loop
                 ELSE
! photon cross at least one time slice before collision
! update it
                    it = it + 1
                    t = tcut
                    CALL WHERE
! Note, tck remains abs time to collision
                    IF(ph_finish)THEN
                       CYCLE sim  ! next photon
                    END IF
                    CALL trace_water
                 END IF
              END DO slices
           END DO between
        END DO sim
!
! ***************** CALCULATION FINISH *****************
!
        cpu = ETIME(tarray)
        WRITE(*,*)'Outputing to Files        :',cpu/60.,'CPU (min) @ ' ,
     &                                                  tbuf
        CALL output
        cpu = ETIME(tarray)
        CALL time(tbuf)
        WRITE(*,*)'STOP                      :',cpu/60.,'CPU (min) @ ' ,
     &                                                  tbuf
        WRITE(*,*)' '
        WRITE(*,*)'In this simulation there were:'
        WRITE(*,*)'In',REAL(xabsorb),' Absorption Events'
        WRITE(*,*)'In',REAL(xwater),' Scattering Events from Water'
        WRITE(*,*)'In',REAL(xscatt),' Scattering Events from Suspension'
        WRITE(*,*)'In',REAL(xtotal),' Total Collisions'
!
        WRITE(*,*)' Maximum r extent in water: ', r_max(1),' (m)'
        WRITE(*,*)' Maximum x extent at Plane: ', r_max(2),' (m)'
        WRITE(*,*)' Maximum y extent at Plane: ', r_max(3),' (m)'
        WRITE(*,*)' Minimum temporal extent in water: ', t_max(1),' (s)'
        WRITE(*,*)' Maximum temporal extent in water: ', t_max(2),' (s)'
        WRITE(*,*)' Calculated minimum ToF to Plane : ', REAL(plane_mint),' (s)'
        WRITE(*,*)' Minimum temporal extent at Plane: ', t_max(3),' (s)'
        WRITE(*,*)' Maximum temporal extent at Plane: ', t_max(4),' (s)'
        IF(xr_hot.ne.0.d0)THEN
           WRITE(*,*)'  ', REAL(xr_hot), ' Too Hot Radial Boxes in Water'
        END IF
        IF(xx_hot.ne.0.d0)THEN
           WRITE(*,*)'  ', REAL(xx_hot), ' Too Hot X Boxes at Plane'
        END IF
        IF(xy_hot.ne.0.d0)THEN
           WRITE(*,*)'  ', REAL(xy_hot), ' Too Hot Y Boxes at Plane'
        END IF
```

```
      IF(xtp_hot.ne.0.d0)THEN
         WRITE(*,*)' ',REAL(xtp_hot),' Too Hot Temporal Boxes at Plane'
      END IF
      IF(xtw_hot.ne.0.d0)THEN
         WRITE(*,*)' ',REAL(xtw_hot),' Too Hot Temporal Boxes in Water'
      END IF
      GOTO 2000

! ********** Error on network OPEN traps **********
1011  WRITE(*,*)'Error opening seeds.dat'
      ifail1 = ifail1 + 1
      CALL SLEEP(1) ! Portlib Routine
      IF(ifail1.gt.50)GOTO 2000
      GOTO 11
1021  WRITE(*,*)'Error opening r607.dat'
      CALL SLEEP(1)
      ifail2 = ifail2 + 1
      IF(ifail2.gt.50)GOTO 2000
      GOTO 12
2000  STOP
      END
!
! ********************************************************************
!
      SUBROUTINE DEP_VAR
! Set up variables which depend on user input characteristics
! modules used
      USE phase
      USE ran_par
      USE beam_par
      USE sea_air
      USE boxes
      USE water_par
      USE slice
      IMPLICIT NONE
! distance from wave included surface to water or plane
      REAL(8) dist
! functions
      REAL(8) surface
! intermediate working variables
      REAL(8) plane_int, water_int
!
! convert scan_angle to SI
      scan_angle = scan_angle* 3.14159265359d0/180.0d0
! set center of 'in' water array
      x_center = plane*SIN(scan_angle)
! estimate minimum time for photon to reach water for efficient
! boxing. -
!  corresponds to an 'early exit' photon travelling
!  down along the scan angle to the surface.
      dist = plane - surface()
      water_int = dist/3.0d8 - laser_width
! Now find the nearest integral multiple of dt_slice which is less than
! or equal to water_int
      water_mint = 0.d0
      DO WHILE (water_mint.le.water_int)
         water_mint = water_mint + dt_slice
      END DO
      water_mint = water_mint - dt_slice
! estimate minimum time for photon to reach plane height for efficient
```

```
! boxing. -
!  corresponds to an 'early exit' photon travelling
!  down along the scan angle to the surface, being immediately reflected
!  vertically to plane height
        dist = SQRT((plane - surface())*(plane - surface())                &
                    + plane*SIN(scan_angle)*plane*SIN(scan_angle))         &
                + plane - surface()
        plane_int = dist/3.0d8 - laser_width
! Now find the nearest integral multiple of dt_slice which is less than
! or equal to plane_int
        plane_mint = 0.d0
        DO WHILE (plane_mint.le.plane_int)
           plane_mint = plane_mint + dt_slice
        END DO
        plane_mint = plane_mint - dt_slice
! convert spot to radius
        spot = spot/2.d0
        sigma = spot/SQRT(-2.d0*LOG(1.d0 - 0.86d0))
!
! ***************** TOTAL MEAN COLLISION FREQUENCY *****************
!
        atten_coeff = a_coeff + w_coeff + s_coeff
! Relative attenuations
        a_coeff = a_coeff/atten_coeff
        w_coeff = w_coeff/atten_coeff
        s_coeff = s_coeff/atten_coeff
! Mean Free Time between collisions, as per header comments
        time_mean = 1.d0/(atten_coeff*2.25d08)
! note: speed of light in water = 2.25d8 m/s
        WRITE(*,*)'*************************************************'
        WRITE(*,*)' Total Attenuation = ', REAL(atten_coeff),' /meter'
        WRITE(*,*)' Mean Free Time = ', 1.e9*REAL(time_mean),' ns'
        WRITE(*,*)' Expected random Distribution'
        WRITE(*,*)' Scattering by Water:                0 - ', REAL(w_coeff)
        s_coeff = w_coeff + s_coeff
        WRITE(*,*)' Scattering by Suspension:' ,REAL(w_coeff),' - ', REAL(s_coeff)
        a_coeff = s_coeff + a_coeff
        WRITE(*,*)' Absorption by Water:  ',REAL(s_coeff),' - ', REAL(a_coeff)
        WRITE(*,*)'*************************************************'
        WRITE(*,*)'*************************************************'

        RETURN
        END
!
! ********************************************************************
!
        SUBROUTINE BEGIN
! This routine begins a photon -
! Increments m_photon and i_output
!
! modules used
        USE ph_logic
        USE boxes
        USE phase
        USE out_par
        IMPLICIT NONE
! local surface function
        REAL(8) surface
!
        m_photons = m_photons + 1
```

```
        i_output = i_output + 1
! set photon finished flag false
        ph_finish = .FALSE.
! Initialize phase info for debugging
        x = 0.d0
        y = 0.d0
        z = 0.d0
        Vx= 0.d0
        Vy= 0.d0
        Vz= 0.d0
        V = 3.0d8
        t = 0.d0
        it = 0
        it_entry = 0
!
        z = z + surface()
        CALL laser
        CALL surface_entry
!
        RETURN
        END
!
! ***********************************************************************
!
        SUBROUTINE WHERE
! This subroutine calculates the position and velocity of the
! photon at time t and returns the phase info through the module
! phase
!
! modules used
        USE ph_logic
        USE phase
        USE ran_par
        USE bottom_par
        IMPLICIT NONE
! elapsed time since last call to WHERE
        REAL(8) delta_t
! local variables
        REAL(8) local_s, local_b, z_int, y_int, x_int, t_int
! scattering angle components
        REAL(8) costh, sinth, phi
! boundary functions
        REAL(8) surface, bottom
! inclusive random number function
        REAL(4) randti
! random variable
        REAL(4) r1, r2
!
! calculate delta_t first and then intermediate x, y, z
        delta_t = t - t0
        z_int = Vz*delta_t + z
        y_int = Vy*delta_t + y
        x_int = Vx*delta_t + x
! NB: this is compatible with wave action and bottom topography as
! described in log Book 1, page 32
        local_s = surface()
        local_b = bottom()
        IF((z_int.LT.local_s).AND.(z_int.GT.local_b)) THEN
! there is nothing more to do - set intermediates to final
! most common occurrence
```

```
            z = z_int
            y = y_int
            x = x_int
        ELSE IF(z_int.lt.local_b) THEN
! first, determine if photon is reflected ... saves time
            r1 = randti(is , ris)
            ai = ai + DBLE(r1)
            callsi = callsi + 1.0d0
            IF(r1.ge.bott_reflect)THEN
! photon is absorbed ... set ph_finish, exit to end
                ph_finish = .TRUE.
                GOTO 666
            END IF
! find time to intersection with bottom, reflect, then carry
! photon for the rest of the time associated with call
            t_int = (local_b-z)/Vz
            z = local_b
            y = Vy*t_int + y
            x = Vx*t_int + x
! set random variables
            r1 = randti(js , rjs)
            aj = aj + DBLE(r1)
            callsj = callsj + 1.0d0
            r2 = randti(is , ris)
            ai = ai + DBLE(r2)
            callsi = callsi + 1.0d0
! determine cos theta for scattered vector ... uniform on 0,1
! ... only positive Vz on bottom reflection, from NB: FLAT BOTTOM
            costh = r1
            sinth = SQRT(1.0d0 - costh*costh)
            phi=6.283185308d0*r2
            Vz=2.25d8*costh
            Vx=2.25d8*sinth*COS(phi)
            Vy=2.25d8*sinth*SIN(phi)
! take the photon on to next collision with new Vx, Vy, Vz
! note that technically, the photon path
! may be unphysical ... ie. may cross an interface here
! watch in debug later
            t_int = delta_t - t_int
            z = Vz*t_int + z
            y = Vy*t_int + y
            x = Vx*t_int + x
            IF(z.ge.surface())THEN
! Ignore the rest of this photon
                WRITE(*,*)' A bottom reflected photon has exited the surface'
                WRITE(*,*)' z = ',REAL(z)
                ph_finish = .TRUE.
            END IF
        ELSE IF(z_int.GT.local_s) THEN
! find time to intersection with surface, call SURFACE_EXIT and
! take appropriate action
! NB: t is set if exit
            t_int = (local_s-z)/Vz
! t at surface
            z = local_s
            y = Vy*t_int + y
            x = Vx*t_int + x
            CALL surface_exit(t_int)
            IF(.NOT.ph_finish)THEN
! take the photon on to next collision with new Vx, Vy, Vz
```

```
! note that technically, if very wavy, the photon path
! may be unphysical ... ie. may cross an interface here
! watch in debug later
                t_int = delta_t - t_int
                z = Vz*t_int + z
                y = Vy*t_int + y
                x = Vx*t_int + x
                IF(z.le.bottom())THEN
! Ignore the rest of this photon
                    WRITE(*,*)' A surface reflected photon has struck the bottom'
                    ph_finish = .TRUE.
                    WRITE(*,*)' z = ',REAL(z)
                END IF
            END IF
        ELSE IF(z_int.EQ.local_b)THEN
! set the depth to depth +0.1 mm and this will do
! may need checking for V normailisation 02/08/96
            z = local_b + 0.0001
            y = y_int
            x = x_int
        ELSE
! photon is relaxing on the surface ...
! set the depth to depth -0.1 mm and this will do
! may need checking for V normailisation 02/08/96
            z = local_s - 0.0001
            y = y_int
            x = x_int
        END IF
666     t0 = t
        RETURN
        END
!
! ***********************************************************************
!
        SUBROUTINE COLLIDE
!
! This subroutine determines which collision the photon has
!
! modules used
        USE ph_logic
        USE ran_par
        USE phase
        USE monitors
        USE water_par
        IMPLICIT NONE
! random number variables
        REAL(4) randti
! inclusive (0-1) function return
        REAL(4) Rcoll, r1
! scattering angle components
        REAL(8) costh, sinth
! variables for defining random vector and hence absolute velocity
! cosines wrt to known x,y,z
        REAL(8) rcosth, rsinth
! random dangles
        REAL(8) rx, ry, rz
! speed scalars
        REAL(8) c
! parallel components
        REAL(8) rxp, ryp, rzp, Vxp, Vyp, Vzp
```

```
! orthogonal components
      REAL(8) rxo, ryo, rzo, Vxo, Vyo, Vzo
! local value for average cosine
      REAL(8) g_
!
      Rcoll = randti(js , rjs)
! THROW DICE TO DETERMINE PROBABILITY
      aj = aj + DBLE(Rcoll)
      callsj = callsj + 1.0d0
      xtotal = xtotal  +  1.d0
!
!   COMPARING PROBABILITIES FOR PROCESSES AGAINST Rcoll
!
      IF(Rcoll.GT.s_coeff) THEN        ! greater than scat ... absorb
         xabsorb = xabsorb  +  1.d0
! absorbing collision ...
         ph_finish = .true.
! finish flag set
         RETURN
! return
      ELSE IF(Rcoll.GT.w_coeff) THEN ! greater than water ... suspended
! scattering from suspension
         g_ = g_scatt
! set average cosine for scattering
         xscatt = xscatt  +  1.d0
      ELSE
! scattering from water
         g_ = g_water
         xwater = xwater  +  1.d0
      END IF
!
! 1. find costh and sinth according to Henyey-Greenstein distribution
!
      R1 = randti(ks , rks)
      ak = ak + DBLE(r1)
      callsk = callsk + 1.0d0
      costh = 1.d0/2.d0/g_*(1.d0+g_*g_ - (1.d0-g_*g_)*(1.d0-g_*g_)/        &
            (1.d0-g_+2.d0*g_*r1)/(1.d0-g_+2.d0*g_*r1))
!
      IF(ABS(costh).EQ.1.0)THEN
         sinth = 0.d0
      ELSE
         sinth = DSQRT(1.0d0  -  costh*costh)
      END IF
!
! 2. find the component of Vf which is parallel to Vi
!    (ie scale to costh and vf/vi)
!
      vxp=vx*costh
      vyp=vy*costh
      vzp=vz*costh
!
! 3. Define a random vector in xyz space
!

 200  r1 = randti(ls , rls)
      al = al + DBLE(r1)
      callsl = callsl + 1.0d0
      rcosth=(1.d0-2.d0*r1)
      rsinth=SQRT(1.d0-rcosth*rcosth)
```

```
      r1 = randti(ms , rms)
      am = am + DBLE(r1)
      callsm = callsm + 1.0d0
      r1=6.283185308d0*r1
      rx=rsinth*COS(r1)
      ry=rsinth*SIN(r1)
      rz=rcosth
!
! 4. find the components of the random vector that is parallel to Vi
!
      c=(vx*rx+vy*ry+vz*rz)/5.0625d16
      rxp=vx*c
      ryp=vy*c
      rzp=vz*c
!
! 5. find the components of the random vector that is orthogonal to Vi
!
      rxo=rx-rxp
      ryo=ry-ryp
      rzo=rz-rzp
      c=SQRT(rxo*rxo+ryo*ryo+rzo*rzo)
      IF (c.LT.1.e-5)GOTO 200
!
! 6. Scale to get components of Vf orthog to Vinit
!
      c=2.25d8*sinth/c
      vxo=rxo*c
      vyo=ryo*c
      vzo=rzo*c
!
! 7. Add perp and orthog components for final components
!
! for debugging  comment out the three lines below ... no scattering
      vx=(vxo+vxp)
      vy=(vyo+vyp)
      vz=(vzo+vzp)
      v=SQRT(vx*vx+vy*vy+vz*vz)
!
      RETURN
      END
!
! ***********************************************************************
!
      SUBROUTINE TRACE_WATER
!
! This routine traces the motion of the photons through the
! water
! photons are traced in r,z,t about x = plane*sin(scan_angle), y = 0
! note: this really only deals with the 'in water' motion of the
! photons TRACE_PLANE deals with the special case of the photons at
! z=plane
!
! modules used
      USE phase
      USE monitors
      USE boxes
      USE extrema
      IMPLICIT NONE
      INTEGER(4) ir, iz, it_water
```

```
! working intermediates
      REAL(8) r, x_int
!
! ******************** WRITE INTO ARRAYS **********************
!
      x_int = x - x_center
      r = SQRT(x_int*x_int + y*y)
      ir = INT(r)+1
      IF(ir.gt.ir_rzt)THEN
         ir = ir_rzt
         xr_hot = xr_hot + 1.0
      END IF
! note boxing of iz includes abs(z) ... wave action will blur!
      iz = INT(ABS(z))+1
      IF(iz.gt.jz_rzt)THEN
         iz = jz_rzt
         WRITE(*,*)' ALERT: iz = 50'
      END IF
!
! NOTE: THIS MEANS THAT AT Z = 0 IT_WATER = 1 ... MAY NEED SOME THOUGHT
!
      it_water = it - it_entry + 1
      IF(it_water.gt.kt_rzt)THEN
         it_water = kt_rzt
         xtw_hot = xtw_hot + 1.0
      END IF
      box_rzt(ir, iz, it_water) = box_rzt(ir, iz, it_water) + 1
      IF(t.lt.t_max(1)) t_max(1)=t
      IF(t.gt.t_max(2)) t_max(2)=t
      IF(r.gt.r_max(1)) r_max(1)=r
      RETURN
      END

!
! ***********************************************************************
!
      SUBROUTINE OUTPUT
! This routine outputs arrays of information gathered in trace
! to named files
! Also updates information files and save the random number info
!
! modules used
      USE files
      USE monitors
      USE extrema
      USE beam_par
      USE water_par
      USE bottom_par
      USE out_par
      USE ran_par
      USE boxes
!     USE portlib
      IMPLICIT NONE
! WRITE loop indices
      INTEGER(4) i
! as in main
      REAL(4) tarray(2)
! as in main
      EXTERNAL ETIME
      REAL(4) ETIME
```

```
      REAL(4) cpu
! as in main
      CHARACTER(8) tbuf
! fail safe handlers
      INTEGER(2) ifail1, ifail2
!
      CALL time(tbuf)
      cpu = ETIME(tarray)
      WRITE(*,*)'Writing to files'
      WRITE(*,*)m_photons,' photons in',cpu/60.,'CPU (min)@ ',tbuf
!*********************************
      CALL write_boxed
!*********************************
      OPEN(UNIT = 10 , FILE = finfo , STATUS = 'unknown')
! for PC based program
!     OPEN(UNIT= 10,FILE='otest.txt',status='unknown')
      WRITE(10,*)'Calculation at:'
      WRITE(10,*)REAL(plane),' m PLANE HEIGHT'
      WRITE(10,*)REAL(laser_width)*1.e9,' ns LASER WIDTH'
      WRITE(10,*)REAL(scan_angle)*180./3.14159265359,' degrees SCAN ANGLE'
      WRITE(10,*)REAL(spot)*2.,' m SPOT DIAMETER'
      WRITE(10,*)REAL(depth),' m BOTTOM DEPTH'
      WRITE(10,*)REAL(bott_reflect)*100.,' % BOTTOM REFLECTANCE'
      WRITE(10,*)' '
      WRITE(10,*)'ABSORPTION COEFF   : ', a_out
      WRITE(10,*)'H2O SCATTERING COEFF: ', w_out
      WRITE(10,*)'H2O AVERAGE COSINE  : ', gw_out
      WRITE(10,*)'SUS SCATTERING COEFF: ', s_out
      WRITE(10,*)'SUS AVERAGE COSINE  : ', gs_out
      WRITE(10,*)' Total Attenuation = ', REAL(atten_coeff),' /meter'
      WRITE(10,*)' Mean Free Time = ', 1.e9*REAL(time_mean),' ns'
      WRITE(10,*)' '
      WRITE(10,*)m_photons,' photons in',cpu/60.,'CPU (min) @ ', tbuf
      WRITE(10,*)' '
      WRITE(10,*)'In this simulation there were:'
      WRITE(10,*)'In',REAL(xabsorb),' Absorption Events'
      WRITE(10,*)'In',REAL(xwater) ,' Scattering Events from Water'
      WRITE(10,*)'In',REAL(xscatt),' Scattering Events from Suspension'
      WRITE(10,*)'In',REAL(xtotal),' Total Collisions'
!
      WRITE(10,*)' Maximum r extent in water: ', r_max(1),' (m)'
      WRITE(10,*)' Maximum x extent at Plane: ', r_max(2),' (m)'
      WRITE(10,*)' Maximum y extent at Plane: ', r_max(3),' (m)'
      WRITE(10,*)' Minimum temporal extent in water: ', t_max(1),' (s)'
      WRITE(10,*)' Maximum temporal extent in water: ', t_max(2),' (s)'
      WRITE(10,*)' Calculated minimum ToF to plane : ', REAL(plane_mint),' (s)'
      WRITE(10,*)' Minimum temporal extent at Plane: ', t_max(3),' (s)'
      WRITE(10,*)' Maximum temporal extent at Plane: ', t_max(4),' (s)'
      IF(xr_hot.ne.0.d0)THEN
         WRITE(10,*)'  ', REAL(xr_hot), ' Too Hot Radial Boxes in Water'
      END IF
      IF(xx_hot.ne.0.d0)THEN
         WRITE(10,*)'  ', REAL(xx_hot), ' Too Hot X Boxes at Plane'
      END IF
      IF(xy_hot.ne.0.d0)THEN
         WRITE(10,*)'  ', REAL(xy_hot), ' Too Hot Y Boxes Plane'
      END IF
      IF(xtp_hot.ne.0.d0)THEN
         WRITE(10,*)'  ',REAL(xtp_hot),' Too Hot Temporal Boxes at Plane'
      END IF
```

```
      IF(xtw_hot.ne.0.d0)THEN
         WRITE(10,*)'  ',REAL(xtw_hot),' Too Hot Temporal Boxes in Water'
      END IF
!
      WRITE(10,*)'PLANE HEIGHT OUTPUT IN ' , fplane
      WRITE(10,*) ix_plane, jy_plane, kt_plane
      WRITE(10,*)'R,Z,T OUTPUT IN ' , frzt
      WRITE(10,*) ir_rzt, jz_rzt, kt_rzt
      CLOSE(10)
      OPEN(UNIT = 10 , FILE = frun , STATUS = 'unknown')
! for PC based program
!     OPEN(UNIT= 10,FILE='rtest.txt',status='unknown')
      WRITE(10,*)m_photons, n_total, n_output
      WRITE(10,*)plane_mint, water_mint
      WRITE(10,*)plane, depth, spot
      WRITE(10,*)atten_coeff, time_mean
      WRITE(10,*)a_coeff, w_coeff, s_coeff
      WRITE(10,*)g_water, g_scatt
      WRITE(10,*)bott_reflect
      WRITE(10,*)is , js , ks , ls , ms
      IF(callsi.eq.0.)callsi = 1.0
      IF(callsj.eq.0.)callsj = 1.0
      IF(callsk.eq.0.)callsk = 1.0
      IF(callsl.eq.0.)callsi = 1.0
      IF(callsm.eq.0.)callsm = 1.0
      WRITE(10,*)ai/callsi , aj/callsj , ak/callsk
      WRITE(10,*)al/callsl , am/callsm
      WRITE(10,*)callsi , callsj , callsk , callsl, callsm
      WRITE(10,50)finfo , fplane , frzt
      WRITE(10,*) ix_plane, jy_plane, kt_plane
      WRITE(10,*) ir_rzt, jz_rzt, kt_rzt
 50   FORMAT(5(1x , A40 , /))
      CLOSE(10)
 11   OPEN(UNIT=15,FILE='seeds.dat',STATUS='unknown',err=1011)
      WRITE(15,*)is, js, ks, ls, ms
      CLOSE(15)
 12   OPEN(UNIT = 15 , FILE = 'ris607.dat' ,                                  &
      FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
      DO i = 1 , 607
         WRITE(15)ris(i)
      END DO
      CLOSE(15)
      OPEN(UNIT = 15 , FILE = 'rjs607.dat' ,                                  &
           FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
      DO i = 1 , 607
         WRITE(15)rjs(i)
      END DO
      CLOSE(15)
      OPEN(UNIT = 15 , FILE = 'rks607.dat' ,                                  &
           FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
      DO i = 1 , 607
         WRITE(15)rks(i)
      END DO
      CLOSE(15)
      OPEN(UNIT = 15 , FILE = 'rls607.dat' ,                                  &
           FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
      DO i = 1 , 607
         WRITE(15)rls(i)
      END DO
      CLOSE(15)
```

```
      OPEN(UNIT = 15 , FILE = 'rms607.dat' ,                              &
           FORM = 'UNFORMATTED' , STATUS = 'unknown' , err = 1021)
      DO i = 1 , 607
         WRITE(15)rms(i)
      END DO
      CLOSE(15)
      cpu = ETIME(tarray)
      CALL time(tbuf)
      WRITE(*,*)'Written Files              :',cpu/60.,'CPU (min) @ ',    &
                                                  tbuf
      GOTO 2000
! ***************** Error on network OPEN traps *******************
1011  WRITE(*,*)'Error opening seeds.dat'
      CALL SLEEP(1)
      ifail1 = ifail1 + 1
      IF(ifail1.gt.50)GOTO 2000
      GOTO 11
1021  WRITE(*,*)'Error opening r607.dat'
      CALL SLEEP(1)
      ifail2 = ifail2 + 1
      IF(ifail2.gt.50)GOTO 2000
      GOTO 12
2000  RETURN
      END
!
! ***********************************************************************
!
      REAL(4) FUNCTION RANDT(J,R)
! This FORTRAN-callable function returns a pseudo-random REAL *4
! number uniformly distributed over the exclusive interval (0,1).
! Coded in December, 1982 by
!       W A Alford,
!       School Computer Unit,
!       Research School of Physical Sciences,
!       Australian National University,
!       Canberra, ACT
!
! Modified April 1989 (MJB) to remove possibility of generation of 0.0
! and to retain the seeding between runs
! Mainly cosmetic modification July 1996 (MJB) for f90 compatibility
!
! The FORTRAN call to the REAL *4 function RANDT is of the form:
!
!               y = RANDT (j)
!
! The coding is based on the following paper:
!
! TOOTILL J.P., ROBINSON W.D. and EAGLE J., 1973. 'An asymptotically
!       random Tausworthe sequence.' J.A.C.M. 20(3), 469-481.
!
! In the abstract to this paper they make the following statement.
!
! 'An asymptotically random 23-bit number sequence of astronomic period
!  607
! 2    -1 is presented. An initialisation program is required to provide
! 607 starting values, after which the sequence can be generated from
! a three-term recurrence of the Fibonacci type. In addition to
! possessing the theoretically demonstrable randomness properties
! associated with Tausworthe sequences, the sequence possesses equi-
! distribution and multi-dimensional uniformity properties vastly in
```

```
! excess of anything that has been shown for conventional congruent-
! lally generated sequences.. The claimed randomness properties do not
! necessarily extend to subsequences, though it is not yet known which
! particular subsequences are at fault. Accordingly, the sequence is
! at present suggested only for simulations with no fixed dimensional-
! ity requirements.'
!
! Some statistical tests (frequency, serial, auto-correlation, gap and
! runs tests) on the output from this coding engender confidence in
! these assertions and show that this pseudo-random number generator is
! better than a generalised Fibonacci sequence type generator coded by
! R Brent in use at the Australian National University. In terms of
! speed on average this generator is comparable to the linear
! congruential and generalised Fibonacci sequence type generators.
!
! The initialisation of array R comes from subroutine INIT23 in the
! above paper. The statistical properties of the output from this
! coding is highly dependent upon the initial set of 19 arbitrary
! 32-bit integers used in subroutine INIT23. In this case they were
! derived from a generalised Fibonacci sequence type generator with lag
! 127 referred to above and then subroutine RECUR in the above paper
! was called once again because it was observed that the first 607
! output numbers were statistically poor but everything thereafter was
! acceptable. Several other sets of initial 19 arbitrary 32-bit
! integers were tried but in terms of the statistical properties of the
! output sequences this set was the best found. There may be even
! better sets of initial 19 arbitrary 32-bit integers (or 38 arbitrary
! 16-bit integers for subroutine INIT16). This critical fact was not
! mentioned in the above paper and suggested good sets of initial
! arbitrary integers would have been appreciated in the above paper
! (initial attempts at finding these were disenchanting). The actual
! values in array R used here are those for the first 607 terms of the
! output (ie. after another call to subroutine RECUR).
!
      IMPLICIT NONE
      INTEGER(2) I, j
      INTEGER(4) r(607)
! Should we generate the next 607 terms of the sequence?
5     j=j+1
      IF (j.le.607) GOTO 40
!
! Generate the next 607 terms of the sequence. The sequence is based on
!
!                              607  334
! the primitive trinomial x   +x +1  over GF(2).
!
10    DO 20 j=1,273
         r(i)=r(i).XOR.r(i+334)
20    END DO
      DO 30 i=274,607
         r(i)=r(i).XOR.r(i-273)
30    END DO
! Re-initialise J
      j=1
! Scale the result to (0,1) by dividing by 2**23 (as arbitrary 23-bit
! integers are produced)
!X    randt=REAL(R(J))/REAL(2**23)
! modification to get random numbers on the exclusive interval (0,1)
! 16777216 = 2**24
40    randt=REAL(r(j)+r(j)+1)/REAL(16777216)
```

```
!X      IF(randt.eq.0.0)GOTO 5
        RETURN
        END
!
! ***********************************************************************
!
        REAL(4) FUNCTION RANDTI(j,r)
! This FORTRAN-callable function returns a pseudo-random REAL *4
! number uniformly distributed over the inclusive interval (0,1).
! description as above
!
        INTEGER(2) I, j
        INTEGER(4) r(607)
! Should we generate the next 607 terms of the sequence?
 5      j =j +1
        IF (j.le.607) GO TO 40
!
! Generate the next 607 terms of the sequence. The sequence is based on
! the
!                          607  334
! primitive trinomial       x   +x +1  over GF(2).
!
 10     DO 20 i=1,273
           r(i)=r(i).XOR.r(i+334)
 20     END DO
        DO 30 i=274,607
           r(i)=r(i).XOR.r(i-273)
 30     END DO
! Re-initialise J
        j=1
! Scale the result to (0,1) by dividing by 2**23 (as arbitrary 23-bit
! integers are produced)
! 16777216 = 2**24
! 8388608 = 2**23
 40     randti=REAL(r(j))/8388608
        RETURN
        END
```

# Appendix 3

The program in this appendix is current as of 05-August-96.

## level1.f90

```
        SUBROUTINE LASER
! This routine determines at what value of time, t, the photon is
! initiated by the laser, determines x and y a sea surface
!
! modules used
        USE beam_par
        USE phase
        USE ran_par
        USE boxes
        IMPLICIT NONE
! increments to x, y due to laser width
        REAL(8) dx, dy
! inclusive random number function
        REAL(4) randti
! intermediate random var
```

```
      REAL(4) r1
!
      IF(laser_width.lt.1.d-16)THEN
! RETURN a delta t of 0.
         laser_time = 0.d0
      ELSE
! throw a random fraction of the laser width for inclusion in laser
! 'air time'
         r1 = randti(ls , rls)
         al = al + DBLE(r1)
         callsl = callsl + 1.0d0
         laser_time = (r1 - 0.5)*laser_width
! note that a positive value of laser_time, actually means that the
! photon is emitted 'early' Rt t = 0.
      END IF
!
! Determine x and y coord for photon on surface at time of entry
! (NB: THIS ASSUMES Z =0 for the moment)
! Two components ... y = 0 and x = z sin (scan_angle)
!
      x = plane*SIN(scan_angle)
!                ... x = x+dx y = y+dy from spot size
      CALL spot_dev(dx,dy)
      x = x + dx
      y = dy
!
      RETURN
      END
!
! ***********************************************************************
!
      SUBROUTINE SPOT_DEV(dx,dy)
! This routine returns dx and dy, the increment/decrement to x and y
! due to the finite spot_zize of the Gaussian beam
!
! modules used
      USE beam_par
      USE ran_par
      IMPLICIT NONE
      REAL(4) r1, r2
      REAL(8) dx, dy
! exclusive random number function
      REAL(4) randt, randti
!
      IF(spot.lt.0.001d0)THEN
         dx = 0.d0
         dy = 0.d0
      ELSE
         r1 = randt(js , rjs)
         aj = aj + DBLE(r1)
         callsj = callsj + 1.0d0
         r2 = randti(ks , rks)
         ak = ak + DBLE(r1)
         callsk = callsk + 1.0d0
         dx = sigma*SQRT(-2.*LOG(r1))*COS(6.28318530718d0*r2)
         dy = sigma*SQRT(-2.*LOG(r1))*SIN(6.28318530718d0*r2)
      END IF
!
      RETURN
      END
```

```
!
! ****************************************************************
!
      SUBROUTINE SURFACE_ENTRY
! The photon is either reflected (Vz is inverted) and the program calls
! to_plane for boxing, or the photon is refracted into the water
! according to Snell's law n_1*sin(th_1) = n_2*sin(th_2), where n_1 = 1
! for air and n_2 = 1.333333 for sea water
! Velocity components are assigned non-zero values for the first time
! photon is 'traced'
!
! modules used
      USE phase
      USE ran_par
!     USE surface_par
      USE beam_par
      USE sea_air
      USE boxes
      USE slice
      IMPLICIT NONE
! Fresnel Formulae components
      REAL(4) r_para, r_perp
! reflected fraction
      REAL(4) surf_reflect
! inclusive random number function
      REAL(4) randti
! random intermediate
      REAL(4) r1
! radius squared at z = 0 for the moment
      REAL(8) rsq_z0
! distance from wave included surface to plane
      REAL(8) dist
! local surface height
      REAL(8) top_bound
! functions
      REAL(8) surface
!
! determine 'air' time for photon
      rsq_z0 = x*x + y*y
      top_bound = z + surface()
      dist = SQRT(rsq_z0 + (plane-top_bound)*(plane-top_bound))
      t = dist/3.0d8 + laser_time
! photon enters the water between it and it + 1
! need to set 'it' to 'previous slice' so that TCUT in main is set to
! next slice
      it = INT(t/dt_slice)
      it_entry = it
! Assign in air velocity vector for photon
      Vz = (z-plane)/t
      Vy = y/t
      Vx = x/t
!
! Does the photon reflect from the surface?
! determine
      costh = ABS(Vz)/3.d8
      sinth_sq = 1.0 - costh*costh
      n_i = 1.0
      n_t = water_n
      costh_i = costh
      costh_t = 1.0/water_n*SQRT(water_n_sq - sinth_sq)
```

```
! determine the fraction of the reflected flux
      surf_reflect=0.25*(r_para()+r_perp())*(r_para()+r_perp())
! test surf_reflect against a random number
      R1 = randti(ms , rms)
      am = am + DBLE(r1)
      callsm = callsm + 1.0d0
!
      IF(r1.gt.surf_reflect)THEN
! no: Snell's law gives V components in the water  (log book 1 pg 37)
         Vz = -2.25d8/water_n*SQRT(water_n_sq - sinth_sq)
         Vy = Vy/water_n_sq
         Vx = Vx/water_n_sq
! record initial position in water
         CALL trace_water
      ELSE
! yes: reverse Vz and go to plane of plane
         Vz = -Vz
         CALL to_plane
      END IF
!
      RETURN
      END
!
! ***********************************************************************
!
      SUBROUTINE SURFACE_EXIT(t_int)
! Note that this is re-entrant in principle... Snell's law is used.
! The photon is either reflected (Vz is inverted) and the program
! returns to WHERE, or the photon is refracted into the air
! Calls to_plane for boxing if exit
! Should not modify ph_finish
!
! modules used
      USE phase
      USE ran_par
!     USE surface_par
      USE beam_par
      USE sea_air
      IMPLICIT NONE
! fraction of t - t0 from WHERE required to take photon to surface
      REAL(8) t_int
! Intermediate variable for 1/water_n_sq
      REAL(8) t_inv
! Fresnel Formulae components
      REAL(4) r_para, r_perp
! reflected fraction
      REAL(4) surf_reflect
! inclusive random number function
      REAL(4) randti
! random intermediate
      REAL(4) r1
!
! Does the photon reflect from the surface?
! determine
      costh = ABS(Vz)/2.25d8
      sinth_sq = 1.0 - costh*costh
      t_inv = 1.d0/water_n_sq
! determine if theta > theta_critical => total internal  reflection
      IF(sinth_sq.LT.t_inv)THEN
! most common occurrence is reflection/refraction
```

```
            n_i = water_n
            n_t = 1.0
            costh_i = costh
            costh_t = water_n*SQRT(t_inv - sinth_sq)
! determine the fraction of the reflected flux
            surf_reflect=0.25*(r_para()+r_perp())*(r_para()+r_perp())
! test surf_reflect against a random number
            r1 = randti(ms , rms)
            am = am + DBLE(r1)
            callsm = callsm + 1.0d0
        ELSE
! use r1 below as 100% switch when theta is large enough for
! total internal reflection ... push to ELSE below and reflect
            r1 = -2.0
        END IF
        IF(r1.gt.surf_reflect)THEN
! yes: Snell's law gives V components in the air
            Vz = 3.0d8*water_n*SQRT(t_inv - sinth_sq)
            Vy = Vy*water_n_sq
            Vx = Vx*water_n_sq
! time at surface exit is t0 + t_int
            t = t0 + t_int
! go to plane
            CALL to_plane
        ELSE
! no: reverse Vz and go back to WHERE
            Vz = -Vz
        END IF
!
        RETURN
        END
!
! ***********************************************************************
!
        REAL(8) FUNCTION SURFACE
! Calculates the local surface topography
! should have the same sense as plane ... ie. waves have positive height
! troughs are negative
! at the present (14/07/96) no wave action
!
        surface = 0.0d0
!
        RETURN
        END
!
! ***********************************************************************
!
        REAL(8) FUNCTION BOTTOM
! Calculates the local surface topography
! at the present (14/07/96) flat bottom
        USE bottom_par
        bottom = -depth
!
        RETURN
        END
!
! ***********************************************************************
!
        SUBROUTINE WRITE_BOXED
! This routine writes out the boxed information
```

```
! Made modular to ease the future move to run length limited or
! similar compression of data
!
! modules used
      USE boxes
      USE files
      USE out_par
      IMPLICIT NONE
      INTEGER(4) i,j,k
!
! photons to date
      OPEN(UNIT= 10,FILE=fplane,form='UNFORMATTED',status='unknown')
! for PC based program
!     OPEN(UNIT= 10,FILE='ptest.bin',form='UNFORMATTED',status='unknown')
      DO i = 1, ix_plane
         DO j = 1, jy_plane
            WRITE(10)(boxplane(i, j, k), k = 1, kt_plane)
         END DO
      END DO
      CLOSE(10)
      OPEN(UNIT=15,FILE=frzt,form='UNFORMATTED',status='unknown')
! for PC based program
!     OPEN(UNIT= 15,FILE='wtest.bin',form='UNFORMATTED',status='unknown')
      DO 20 i = 1 , ir_rzt
         DO 15 j = 1, jz_rzt
            WRITE(15)(box_rzt(i, j, k), k = 1, kt_rzt)
15          END DO
20       END DO
      CLOSE(15)
!
      RETURN
      END
```

# Appendix 4

The program in this appendix is current as of 05-August-96.

## level2.f90

```
      SUBROUTINE TO_PLANE
! This routine  solves the equation to determine where the photon
! intercepts the plane of the plane boxes the photon there
! sets .ph_finish. flag true
! z should have already been modified to include surface action at
! this point
!
! modules used
      USE ph_logic
      USE phase
      USE beam_par
      IMPLICIT NONE
      REAL(8) air_time
!
! determine time to reach plane height
      air_time = (plane-z)/Vz
! determine x, y, z, t at plane
      x = x + Vx*air_time
      y = y + Vy*air_time
      t = t + air_time
      ph_finish = .true.
```

```
        CALL trace_plane
!
        RETURN
        END
!
! ********************************************************************
!
        SUBROUTINE TRACE_PLANE
! This routine traces the time history of the photons and the radial
! extent at the height of the plane
! Note Absolute x and y are used
!
! modules used
        USE phase
        USE monitors
        USE boxes
        USE extrema
        USE slice
        IMPLICIT NONE
        INTEGER(4) ix, iy, it_plane
!
! ******************* WRITE INTO ARRAYS **********************
!
        ix = INT(ABS(x/10.d0)) + 1
        iy = INT(ABS(y/10.d0)) + 1
        IF(ix.gt.25)THEN
            ix = 25
            xx_hot = xx_hot + 1
        END IF
        IF(iy.gt.2)THEN
            iy = 2
            xy_hot = xy_hot + 1
        END IF
        it_plane = INT((t-plane_mint)/dt_slice) + 1
        IF(it_plane.gt.4000) THEN
            it_plane = 4000
            xtp_hot = xtp_hot + 1.0
        END IF
        IF(it_plane.lt.1) THEN
            write(*,*)' Error in time boxing at plane:    t ',t
            write(*,*)' Calculated minimum ToF: plane_mint ', plane_mint
            write(*,*)' it_plane: ', it_plane
! temp debug 030896
            it_plane = 1
        END IF
        boxplane(ix,iy,it_plane) = boxplane(ix,iy,it_plane) + 1
        IF(t.lt.t_max(3)) t_max(3)=t
        IF(t.gt.t_max(4)) t_max(4)=t
        IF(ABS(x).gt.r_max(2)) r_max(2)=x
        IF(ABS(y).gt.r_max(3)) r_max(3)=y
        RETURN
        END
!
! ********************************************************************
!
        REAL(4) FUNCTION R_PARA()
! Fresnel formula for reflection amplitude of component parallel
! to incident polarisation
        USE sea_air
        IMPLICIT NONE
```

```
! Hecht and Zajac pg 74
      r_para=ABS((n_t*costh_i-n_i*costh_t)/(n_t*costh_i+n_i*costh_t))
!
      RETURN
      END
!
! ****************************************************************
!
      REAL(4) FUNCTION R_PERP()
! Fresnel formula for reflection amplitude of component perpendicular
! to incident polarisation
      USE sea_air
      IMPLICIT NONE
! Hecht and Zajac pg 74
      r_perp = ABS((n_i*costh_i-n_t*costh_t)/(n_i*costh_i+n_t*costh_t))
!
      RETURN
      END
!
! ****************************************************************
!
      REAL(4) FUNCTION T_PARA()
! Fresnel formula for transmission amplitude of component parallel
! to incident polarisation
      USE sea_air
      IMPLICIT NONE
! Hecht and Zajac pg 74
      t_para = ABS(2.*n_i*costh_i/(n_t*costh_i + n_i*costh_t))
!
      RETURN
      END
!
! ****************************************************************
!
      REAL(4) FUNCTION T_PERP()
! Fresnel formula for transmission amplitude of component perpendicular
! to incident polarisation
      USE sea_air
      IMPLICIT NONE
! Hecht and Zajac pg 74
      t_perp = ABS(2.*n_i*costh_i/(n_i*costh_i + n_t*costh_t))
!
      RETURN
      END
```

# Appendix 5

The program in this appendix is current as of 02-September-96.

### 960902p.f90

```
! phoproc
!
! Processes boxed data from photest series
! Created July 1996 for Pentium MJB
! ported August 1996 to DECalpha MJB
!
! Change log
! 240796     changed to output explicit histogram style x,y data
```

```
! 130896    created r,z,t data output for AVS
! 190896    changed in water r,z,t data to log representation
!           AVS only has 256 colours to represent the data
! 200896    Introduced time slicer for water data ...
! 210896    Changed r,z,t water data output to give ln(0) = 0
! 220896    Change both the water r,z,t and r,z at output to
!           give data s.t. z = 0 is at the top!
! 020996    19 boxes in diameter attempt in ln(water)
!
! modules used
      USE boxes
      USE files
      USE extrema
      USE slice
      IMPLICIT NONE
! function case integer
      INTEGER(2) iqu
! number of time slices, slice index
      INTEGER(4) nslice, i_sl
! main program bits and pieces
! Array for x and y summed ToF at Plane
      INTEGER(4) ToF(kt_plane)
! photons simulated and output at frequency ....
      INTEGER(4) m_photons, n_total, n_output
! loop indices
      INTEGER(4) i,j,k
! non-zero element counter
      REAL(4) non_zero
! x axis    time variable
      REAL(4) x_time
! ToF output file name
      CHARACTER(40) fToF
! ASCII water data filename
      CHARACTER(40) fwater
! timing
      CHARACTER(8) tbuf
      CHARACTER(9) dbuf
!
      CALL date(dbuf)
      CALL time(tbuf)
      WRITE(*,*)tbuf , ' on ' , dbuf
!
! Read in files as written
      WRITE(*,*)'*** ENTER THE FOLLOWING FILE INFORMATION ***'
      WRITE(*,*)'ENTER RUN INFORMATION OUTPUT FILE NAME'
      READ(*,30)finfo
      WRITE(*,*)finfo
      WRITE(*,*)'ENTER BOXED x,y,t OUTPUT FILE NAME FOR PLANE HEIGHT'
      READ(*,30)fplane
      WRITE(*,*)fplane
      WRITE(*,*)'ENTER BOXED r,z,t OUTPUT FILE NAME FOR WATER'
      READ(*,30)frzt
      WRITE(*,*)frzt
      WRITE(*,*)'ENTER RUN PARAMETER FILE NAME'
      READ(*,30)frun
      WRITE(*,*)frun
30    FORMAT(A40)
      DO k = 1, kt_plane
         ToF(k) = 0
      END DO
```

```
!
! Read in data arrays
!
      CALL READ_BOXED
! Manipulate
!
      non_zero = 0.
      DO k = 1, kt_plane
         DO i = 1, ix_plane
            DO j = 1, jy_plane
               ToF(k) = Tof(k) + boxplane(i, j, k)
               IF(boxplane(i, j, k).GT.0.)non_zero = non_zero+1.
            END DO
         END DO
      END DO
      write(*,*)'The total number of elements in array BOXPLANE is  ', &
                REAL(kt_plane*ix_plane*jy_plane)
      write(*,*)'The number of non-zero elements in array BOXPLANE is ',&
                non_zero
      non_zero = 0.
      DO i = 1 , ir_rzt
         DO j = 1, jz_rzt
            DO k = 1, kt_rzt
               IF(box_rzt(i, j, k).GT.0.)non_zero = non_zero+1.
            END DO
         END DO
      END DO
      write(*,*)'The total number of elements in array BOX_RZT is ',  &
                REAL(ir_rzt*jz_rzt*kt_rzt)
      write(*,*)'The number of non-zero elements in array BOX_RZT is ', &
                non_zero

!
! Find minimum flight time in run parameter file
      OPEN(UNIT = 10 , FILE = frun , STATUS = 'unknown')
      READ(10,*)m_photons, n_total, n_output
      WRITE(*,*)m_photons,' of ',n_total,                              &
                ' Photons simulated in run to be analysed'
      READ(10,*)plane_mint
!     READ(10,*)plane, depth, spot
!     READ(10,*)a_coeff, w_coeff, s_coeff
!     READ(10,*)g_water, g_scatt
!     READ(10,*)bott_reflect
!     READ(10,*)is , js , ks , ls , ms
!     IF(callsi.eq.0.)callsi = 1.0
!     IF(callsj.eq.0.)callsj = 1.0
!     IF(callsk.eq.0.)callsk = 1.0
!     IF(callsl.eq.0.)callsl = 1.0
!     IF(callsm.eq.0.)callsm = 1.0
!     READ(10,*)ai/callsi , aj/callsj , ak/callsk
!     READ(10,*)al/callsl , am/callsm
!     READ(10,*)callsi , callsj , callsk , callsl, callsm
!     READ(10,50)finfo , fplane , frzt
!
      CLOSE(10)
qu:   DO
         WRITE(*,*)' '
         WRITE(*,*)' What operation do you wish to perform?'
         WRITE(*,*)' Write out ASCII ToF histogram at plane height? ... 1'
         WRITE(*,*)' Write out ASCII ToF ln(data) in water?          ... 2'
```

```
          WRITE(*,*)' Write out ASCII R,Z data for given time(s)? ... 3'
          WRITE(*,*)' Quit from Case loop?                        ... 0'
          READ(*,*)iqu
          SELECT CASE (iqu)
          CASE (1)
             WRITE(*,*)'ENTER PLANE TOF OUTPUT FILE NAME'
             READ(*,30)fToF
             WRITE(*,*)fToF
             OPEN(UNIT= 10,FILE=fToF,status='unknown')
             DO k = 1, kt_plane
! Calculate absolute time in nanoseconds for x axis
                x_time = 1.e9*(plane_mint + REAL(k)*dt_slice)
! write x, y histogram to file
                WRITE(10,*)(x_time-REAL(1.e9*dt_slice)),' ', ToF(k)
                WRITE(10,*)x_time,' ', ToF(k)
             END DO
             CLOSE(10)
          CASE (2)
             WRITE(*,*)'ENTER FILE NAME FOR ASCII WATER DATA OUTPUT'
             READ(*,30)fwater
             WRITE(*,*)fwater
             OPEN(UNIT= 10,FILE=fwater,status='unknown')
             DO i = ir_rzt, 1, -1
                DO j = jz_rzt , 1 , -1
                   DO k = 1, kt_rzt
                      IF(box_rzt(i, j, k).LT.1)THEN
                         WRITE(10,*)' 0.0'
                      ELSE
                         WRITE(10,*)LOG(REAL(box_rzt(i, j, k)))
                      END IF
                   END DO
                END DO
             END DO
             DO i = 2, ir_rzt
                DO j = jz_rzt , 1 , -1
                   DO k = 1, kt_rzt
                      IF(box_rzt(i, j, k).LT.1)THEN
                         WRITE(10,*)' 0.0'
                      ELSE
                         WRITE(10,*)LOG(REAL(box_rzt(i, j, k)))
                      END IF
                   END DO
                END DO
             END DO
             CLOSE(10)
          CASE (3)
             WRITE(*,*)'How many slices do you which to write out?'
             READ(*,*)nslice
             DO i_sl = 1, nslice
                WRITE(*,*)'Enter time array index for R, Z slice number ',i_sl
                READ(*,*)k
                WRITE(*,*)'ENTER FILE NAME FOR ASCII R, Z SLICE DATA OUTPUT'
                READ(*,30)fwater
                WRITE(*,*)fwater
                OPEN(UNIT= 10,FILE=fwater,status='unknown')
                DO i = 1 , ir_rzt
                   DO j = 1, jz_rzt
                      IF(box_rzt(i, j, k).LT.1)THEN
                         WRITE(10,*)i, -j, ' 1.0'
                      ELSE
```

```
                        WRITE(10,*)i, -j, REAL(box_rzt(i, j, k))
                    END IF
                END DO
            END DO
            CLOSE(10)
        END DO
    CASE (0)
        EXIT qu
    CASE DEFAULT
        CYCLE qu
    END SELECT
END DO qu
!
    END
!
! ********************************************************************
!
    SUBROUTINE READ_BOXED
! This routine reads in the boxed information ... as WRITE BOXED above
! Made modular to ease the future move to run length limited or
! similar compression of data
!
! modules used
    USE boxes
    USE files
    IMPLICIT NONE
    INTEGER(4) i,j,k
!
! read photon data as written above
    OPEN(UNIT= 10,FILE=fplane,form='UNFORMATTED',status='unknown')
!   OPEN(UNIT= 10,FILE='ptest.bin',form='UNFORMATTED',status='unknown')
    DO i = 1, ix_plane
        DO j = 1, jy_plane
            READ(10)(boxplane(i, j, k), k = 1, kt_plane)
        END DO
    END DO
    CLOSE(10)
!   OPEN(UNIT= 15,FILE='wtest.bin',form='UNFORMATTED',status='unknown')
    OPEN(UNIT=15,FILE=frzt,form='UNFORMATTED',status='unknown')
    DO i = 1 , ir_rzt
        DO j = 1, jz_rzt
            READ(15)(box_rzt(i, j, k), k = 1, kt_rzt)
        END DO
    END DO
    CLOSE(15)
!
    RETURN
    END
```

DISTRIBUTION LIST

A MONTE-CARLO SIMULATION OF LIGHT PROPAGATION IN SEA WATER
Mike Brennan

## AUSTRALIA

### 1. DEFENCE ORGANISATION

**a. Task Sponsor**
**RAN Hydrographer**
OIC LADS

**b. S&T Program**
Chief Defence Scientist
FAS Science Policy } shared copy
AS Science Corporate Management
Counsellor Defence Science, London (Doc Data Sheet )
Counsellor Defence Science, Washington (Doc Data Sheet )
Scientific Adviser to MRDC Thailand (Doc Data Sheet )
Director General Science Policy Development
Director General Scientific Advisers and Trials/Scientific Adviser Policy and Command (shared copy)
Navy Scientific Adviser
Scientific Adviser - Army (Doc Data Sheet and distribution list)
Air Force Scientific Adviser
Director Trials

**Aeronautical and Maritime Research Laboratory**
Director

**Electronics and Surveillance Research Laboratory**
Director

Chief, LSOD
Research Leader, Space & Surveillance Systems LSOD
Research Leader, Optoelectronics LSOD
Head, Systems Integration
Head, Image Processing & Propagation
Task Manager (NAV 95/024) Mr Ralph Abbot,
Author(s): Dr Mike Brennan (5 Copies)
Mr Dallas Lane LSOD
Mr Trevor Adams LSOD
Dr Derek Bertilone LSOD
Mr Martin O'Connor LSOD
Dr Brian Billard ITD
Dr Kim Brown EWD

**DSTO Library**
Library Fishermens Bend
Library Maribyrnong
Library DSTOS (2 copies)
Australian Archives

Library, MOD, Pyrmont (Doc Data sheet)

c. **Forces Executive**
Director General Force Development (Sea)
Director General Force Development (Land), (Doc Data Sheet)

d. **Navy**
SO (Science), Director of Naval Warfare, Maritime Headquarters Annex, Garden Island, NSW 2000. (Doc Data Sheet)

e. **Army**
ABCA Office, G-1-34, Russell Offices, Canberra (4 copies)

f. **Air Force**
No compulsory distribution

g. **S&I Program**
Defence Intelligence Organisation
Library, Defence Signals Directorate (Doc Data Sheet only)

h. **Acquisition and Logistics Program**
No compulsory distribution

i. **B&M Program (libraries)**
OIC TRS, Defence Regional Library, Canberra
Officer in Charge, Document Exchange Centre (DEC), 1 copy
*US Defence Technical Information Centre, 2 copies
*UK Defence Research Information Center, 2 copies
*Canada Defence Scientific Information Service, 1 copy
*NZ Defence Information Centre, 1 copy
National Library of Australia, 1 copy

2. **UNIVERSITIES AND COLLEGES**

Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering
Deakin University, Serials Section (M list), Deakin University Library
Senior Librarian, Hargrave Library, Monash University
Librarian, Flinders University
Prof John Thomas, School of Physics and Electronic Systems Eng., University of South Australia.
Prof David Booth, Dept of Applied Physics, Victoria University of Technology.
Prof George Kattawar, Texas A&M University, USA

3. **OTHER ORGANISATIONS**

NASA (Canberra)
AGPS
State Library of South Australia
Parliamentary Library, South Australia

## OUTSIDE AUSTRALIA

**4. ABSTRACTING AND INFORMATION ORGANISATIONS**

INSPEC: Acquisitions Section Institution of Electrical Engineers
Library, Chemical Abstracts Reference Service
Engineering Societies Library, US
Materials Information, Cambridge Science Abstracts
Documents Librarian, The Center for Research Libraries, US

**5. INFORMATION EXCHANGE AGREEMENT PARTNERS**

Acquisitions Unit, Science Reference and Information Service, UK
Library - Exchange Desk, National Institute of Standards and Technology, US

SPARES (10 copies)

**Total number of copies: 76**

**DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION**
**DOCUMENT CONTROL DATA**

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

A MONTE-CARLO SIMULATION OF LIGHT PROPAGATION IN SEA WATER

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

| | |
|---|---|
| Document | (U) |
| Title | (U) |
| Abstract | (U) |

4. AUTHOR(S)

Mike Brennan

5. CORPORATE AUTHOR

Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury SA Australia 5108

| 6a. DSTO NUMBER | 6b. AR NUMBER | 6c. TYPE OF REPORT | 7. DOCUMENT DATE |
|---|---|---|---|
| DSTO-TR-0500 | AR-010-149 | Technical Report | March 1997 |

| 8. FILE NUMBER | 9. TASK NUMBER | 10. TASK SPONSOR | 11. NO. OF PAGES | 12. NO. OF REFERENCES |
|---|---|---|---|---|
| D9505-10-145 | NAV 95/024 | RAN Hydrographer | 65 | 17 |

13. DOWNGRADING/DELIMITING INSTRUCTIONS

To be reviewed three years after date of publication

14. RELEASE AUTHORITY

Chief, Land Space and Optoelectronics Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

*Approved for public release*

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Monte Carlo method, propagation, turbidity, scattering, laser bathymetry

19. ABSTRACT

The design and implementation of a suite of programs for Monte Carlo simulation of light propagation in turbid media is described. The program has been tailored to simulate the propagation of the green laser in the RAN Laser Airborne Depth Sounder (LADS) through turbid water. The paper describes the Monte Carlo program in detail, particularly how the inherent multiple scattering problem is interpreted for incorporation into a single scattering simulation. Assumptions made in the implementation of the program are discussed. Results of some initial simulations are presented, together with data obtained during a recent LADS sortie for comparison with the simulations. This paper forms part of the formal documentation for the simulation suite.