# USERS'MANUAL for MONT3D-Code Version 2.4

**Article**

CITATIONS
0

READS
545

**3 authors**, including:

P.J. Burns
Colorado State University
**48** PUBLICATIONS **701** CITATIONS

# USERS' MANUAL

**for**

# MONT3D - Code Version 2.4

**by**

**Charles N. Zeeb, Patrick J. Burns, and Klemens Branner**
**Department of Mechanical Engineering**
**Colorado State University**
**Fort Collins, CO 80523**
**(970) 491-7479, 5778, and 7479**
**E-mail Addresses:**
**czeeb@lamar.colostate.edu**
**pburns@colostate.edu**
**and**
**klem@lamar.colostate.edu**

**and**

**John Dolaghan**
**Lawrence Livermore National Laboratory**
**E-mail Address: dolaghan@llnl.gov**

**April, 1999**

# Table of Contents

# List of Figures



# List of Tables

# CHAPTER 1 INTRODUCTION

This manual encompasses the 3-D Monte Carlo radiative exchange factor computer code "MONT3D." This chapter gives a quick overview of the code, particularly its latest version, 2.4. The chapter begins with the background of the code. This is followed by a theoretical formulation of exchange factors calculated by the code. Next is a brief discussion of view factors. This is followed by a short discussion of the MONT3D implementation. The chapter ends with an overview of the rest of the manual.

## 1.1 Background

MONT3D was developed in the Department of Mechanical Engineering at Colorado State University (CSU) beginning with the work of Scott Statton in 1983 [Statton, 1983], continuing with the work of James D. Maltby [Maltby, 1987; Maltby, 1990], Charles N. Zeeb [Zeeb, 1997; Zeeb and Burns, 1999], and Klemens Branner [Branner, 1999]. The code is used to calculate radiative exchange factors for enclosures with a nonparticipating medium. The focus is complex geometries rather than sophisticated physics. A simple geometry which may be simulated is shown in Figure 1.1. The 3-D code is capable of simulating geometries modeled as assemblages of generalized quadrilaterals (and triangles), which are constrained to be flat, but may otherwise be in any orientation. Curved surfaces must be approximated by a sufficient number of flat surfaces, sometimes termed "faceting," to "capture" the curvature. Surfaces may absorb photons, or they may reflect and/or transmit them specularly, semi-specularly and/or diffusely. All exterior surfaces must be non-transmissive (it is left to the user to ensure this). All material incident radiative properties may be explicit functions of the incident photon angle, and be dependent upon energy through the band wavelength formulation. Donald L. Brown of Lawrence Livermore National Laboratory (LLNL) and Katherine Bryan of Oak Ridge National Laboratory have exhaustively exercised the 3-D code, checking it for validity.

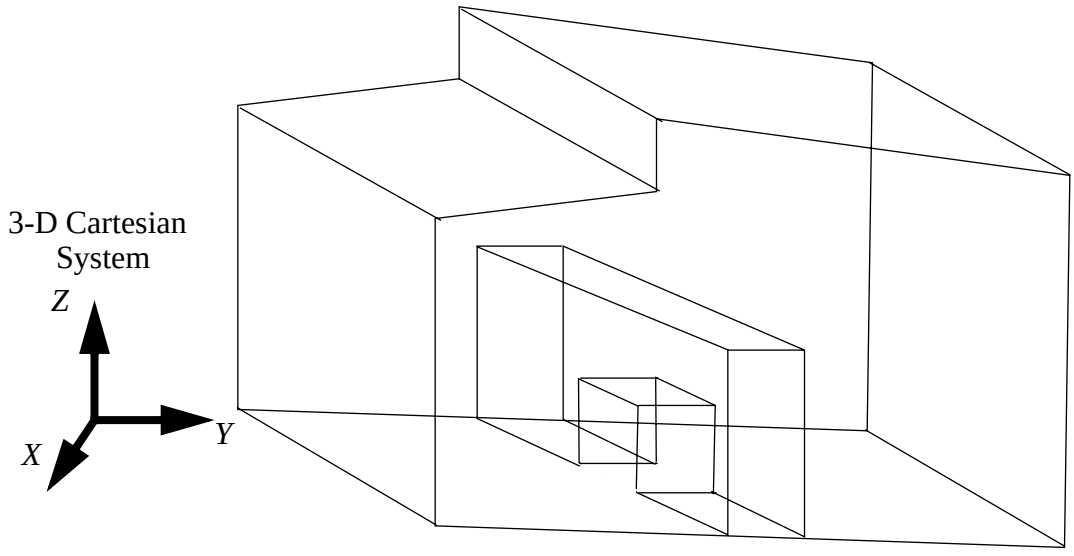



Figure 1.1: Example Geometry Which Can be Simulated

The code has recently been updated extensively. The core of the code has been recoded, improving the efficiency of the code by around 44% in test cases [Zeeb, 1997; Zeeb and Burns, 1999]. Also, the 3-D code has been parallelized to run under the PVM [Geist et al., 1994] environment, with very good success. Furthermore, a new material model, which includes semi-specular components of reflectance and transmittance, has been implemented in the 3-D code. In order to use these enhanced capabilities to the greatest effectiveness, additional detail is required in the input file, which has been changed from the older format to the one described herein. However, input files in the "old" format will still work, as the changes are backwards compatible.

The current version of the code can be run on any computer which possesses a fully featured FORTRAN 77 or Fortran 90 compiler, and has been extensively tested on Sun, Hewlett Packard, IBM, DEC, SGI, Intel, and Macintosh architectures. A robust *Makefile* has been implemented which allows the code to be compiled on virtually any Unix platform, although in some cases some nonessential functions will be lost. More detail on compiling the code for Unix, Microsoft Windows, and Macintosh platforms is provided in Chapter 5. Should compilation problems occur, one of the authors should be contacted.

In order to serve users better, a World Wide Web site for the code exists at http://www.colostate.edu/~pburns/monte.html. The WWW site will include general information about the code, contact information for the authors, the current version of the manual and other documents of interest.

A number of related publications exist wherein the code has been applied to various problems, including a detailed test of the Separator Development Facility (SDF) [Maltby, 1987], the calculation of radiative exchange in passive solar enclosures [Maltby et al., 1986], and application to the Laser Isotope Separation (LIS) process [Burns and Pryor, 1989]. Work has been done to include the modeling of combustion gases, particularly carbon dioxide and water vapor, in MONT2D [Zeeb, 1996]. Five other publications of interest are Pryor and Burns, 1986; Burns et al., 1990; Maltby and Burns, 1991; Burns et al., 1992; and Schweitzer et al., 1993. Also a comprehensive review article of the MONTE codes is in press [Burns and Pryor, 1999]. Finally, an alternative approach for diffuse view factors is the FACET code of Shapiro [Shapiro, 1983].

A number of related codes are available in the public domain. These include the postprocessor SMOOTH [Dolaghan et al., 1992] - which smooths the results and establishes reciprocity; the graphical postprocessor MPLOT [Nagesh and Burns, 1994], which runs on Sun workstations and under Microsoft Windows; MONT2D [Maltby et al., 1994] - an older 2-D version of the MONT3D code which simulates 2-D and axisymmetric geometries; MONT3E [Crockett et al., 1990] - a code for simulating electron transport in three dimensions in the presence of a spatially nonuniform magnetic field; SPUT3D [Dolaghan, 1991] - a code for simulating molecular redistribution during sputtering in three-dimensional enclosures; and MONT3V [Dolaghan, 1996] - a code for simulating rarefied gas dynamics. Information on these codes may be obtained from the authors of this manual.

## 1.2 Theoretical Formulation

MONT3D is formulated in the Monte Carlo style where a large number of photons are emitted from each surface and "traced" until each is absorbed by another surface. Using the subscripts *i* and *j* to denote the emitting and absorbing surfaces, respectively, and the superscript *k* to denote the wavelength band, it can be shown that:

$$\dot{Q}_{i \to j}^{k} = \frac{\varepsilon_i^k N_{ij}^k}{N_i^k} \sigma T_i^4 F_i^k A_i \tag{1.1}$$

where:

$\dot{Q}_{i \to j}^{k}$ = one way rate of radiative heat transfer emitted from surface *i* and absorbed by surface *j* in wavelength band *k* (W).

$\varepsilon_i^k$ = hemispherical emittance of surface *i* in wavelength band *k.*

$N_{ij}^k$ = number of photons emitted in wavelength band *k* from surface *i* and absorbed by surface *j.*

$N_i^k$ = total number of photons emitted in wavelength band *k* from surface *i.*

$\sigma$ = Stefan-Boltzman constant, $5.669 \text{x} 10^{-8} (\text{W} \bullet \text{m}^{-2} \bullet \text{K}^{-4})$.

$T_i$ = absolute temperature of surface *i* (K).

$F_{\text{i}}^{\text{k}}$ = fraction of the blackbody energy of surface *i* in wavelength band *k.*

$A_i$ = area of surface *i* ($\text{m}^2$).

The ordinary definitions of the exchange factor $E_{ij}^k$ and the exchange fraction $F_{ij}^k$ are as follows:

$$E_{ij}^k = \varepsilon_i^k F_{ij}^k = \varepsilon_i^k \frac{N_{ij}^k}{N_i^k} \tag{1.2}$$

The following rules apply to these quantities:

$$\sum_j E_{ij}^k = \varepsilon_i^k \tag{1.3}$$

or:

$$\sum_j F_{ij}^k = 1 \tag{1.4}$$

Equations (1.3) and (1.4) express conservation of energy (photons) since all photons must be absorbed by a surface. Given that there must be zero net heat flow between isothermal surfaces, the second law of thermodynamics (entropy principle) follows from Equation (1.1) as:

$$E_{ij}^{k} A_i = E_{ji}^{k} A_j \tag{1.5}$$

or:

$$\varepsilon_i^k F_{ij}^k A_i = \varepsilon_j^k F_{ji}^k A_j \tag{1.6}$$

The net radiative exchange in wavelength band *k* from surface *i* to surface *j* is then:

$$\dot{Q}_{ij}^{k} = E_{ij}^{k} A_i \sigma \left( F_i^k T_i^4 - F_j^k T_j^4 \right) \tag{1.7}$$

or:

$$\dot{Q}_{ij}^{k} = \varepsilon_i^k F_{ij}^k A_i \sigma \left( F_i^k T_i^4 - F_j^k T_j^4 \right) \tag{1.8}$$

Either of the relations expressed as Equations (1.5) or (1.6) may be used to test the exchange factors or the exchange fractions for consistency (convergence). Indeed, either of these relations may be used to manipulate the values in the matrix if either of Equations (1.3) or (1.4) is used as a constraint.

## 1.3 View Factors

The code may be used to compute view factors, valid for diffuse reflectances independent of incident angle. To do so, all reflectances and transmittances must be set to 0.0, resulting in an emittance of 1.0 (“black”) for all surfaces. In this limiting case, exchange factors for blackbody surfaces are equal to view factors.

## 1.4 MONT3D Implementation

MONT3D is typically used as a “preprocessor” for thermal balance studies. As such, the matrix of exchange factors output from the program is used as input to a thermal analysis code. MONT3D is designed to be compatible with the thermal analysis code, TOPAZ3D [Shapiro, 1985], developed at Lawrence Livermore National Laboratory by Art Shapiro. SMOOTH [Dolaghan et al., 1992] is a postprocessor designed to take advantage of reciprocity to improve the accuracy of the exchange factors. SMOOTH operates on the output of MONT3D and produces output compatible with TOPAZ3D.

Fundamentally, the geometry and the material properties are the only quantities necessary to establish the exchange factors. The exchange factors result from the interaction between the geometry and the material properties in a complex fashion, and are unique to a particular geometry/mate-

rial property combination. Therefore, it is not possible to extend a set of exchange factors calculated for a particular geometry/material property combination to another geometry/material property combination, even if only the geometry or only the material properties vary. After any changes the problem must be rerun. However, if view factors are calculated (all properties black), arbitrary diffuse reflectances may be included in the thermal analysis code since view factors are dependent on geometry alone. Note that diffuse exchange with finite reflectances calculated from the present codes may yield different answers than those obtained using the radiosity/irradiation approach, as the assumption in the radiosity approach is uniform radiosity/irradiation, which is not a mandatory condition for the present code. Specularity or material property dependence on incident angle may not be modeled using view factors.

In addition to radiation exchange, this code can also be used to simulated some special cases of other transport problems in enclosures. If the Knudsen number is much less than 1, rarefied gas dynamics problems can be simulated (a vacuum vessel). Also, some simple cases of molecular sputtering in an enclosure can be simulated.

## 1.5 Overview of the Manual

In Chapter 2, the general background required to run MONT3D is presented. Chapter 3 presents, in great detail, the format of the input file (data deck) that MONT3D requires. A sample problem is used to demonstrate how to run MONT3D in Chapter 4. The more technical details about the MONT3D implementation are covered in Chapter 5.

# CHAPTER 2 COMPUTER CODE BACKGROUND

This chapter presents the background information necessary to run MONT3D. The first three sections define the geometry of a radiating enclosure in terms of nodes and radiating surfaces. Next, there is a note on the surface concatenation option. The material interactions supported by the code and how materials are specified by the user is the subject of the fifth section. The sixth section explains the available shading algorithms, while the seventh discusses accuracy versus the number of photons emitted. Next, debugging and visualization information generated by the program is covered. This is followed by information on the restart capability, whereby a run may proceed from a previously calculated state. In the tenth section, the parallel execution of MONT3D is covered. The chapter concludes with a discussion of the pseudo-random number generator.

## 2.1 Nodes

The description of the geometry of the radiating enclosure begins with the specification of the nodes (also called nodal points) of the enclosure. A node is a point in three-dimensional space defined by three coordinates in a Cartesian coordinate system $\{X, Y, Z\}$, as depicted in Figure 2.1. Each node, $N$, in the enclosure is assigned a node number which must be positive and a member of the closed, full set $N \varepsilon \{1,\text{NUMNP}\}$, where NUMNP is the total number of nodes for the problem. Once the nodes of the enclosure have been specified, radiating surfaces can be defined by specifying sets of nodal points. See Sections 3.2.1 and 3.5 for additional detail on the node input format.

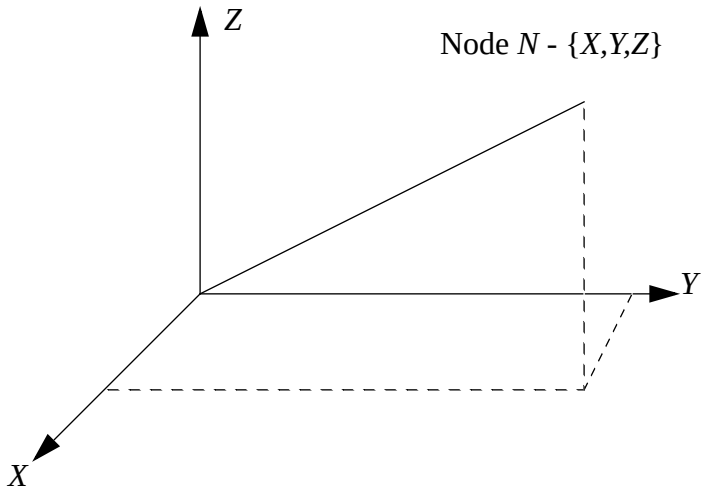
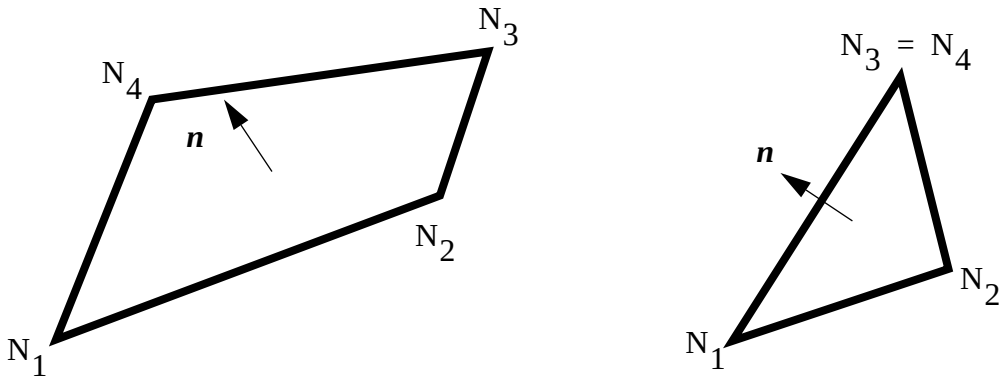



Figure 2.1: Nodal Coordinate System

## 2.2 Surfaces

Three-dimensional radiating surfaces consist of planar quadrilaterals or triangles and are defined by specifying four nodal points. If four distinct points are specified, then the surface is a generalized quadrilateral, as shown in Figure 2.2(a). If it is desired to use triangular surfaces, then the last two node numbers must be identical as shown in Figure 2.2(b). The outward normal for both types of surfaces is defined consistent with the right-hand rule, i.e. if the fingers of the right hand are



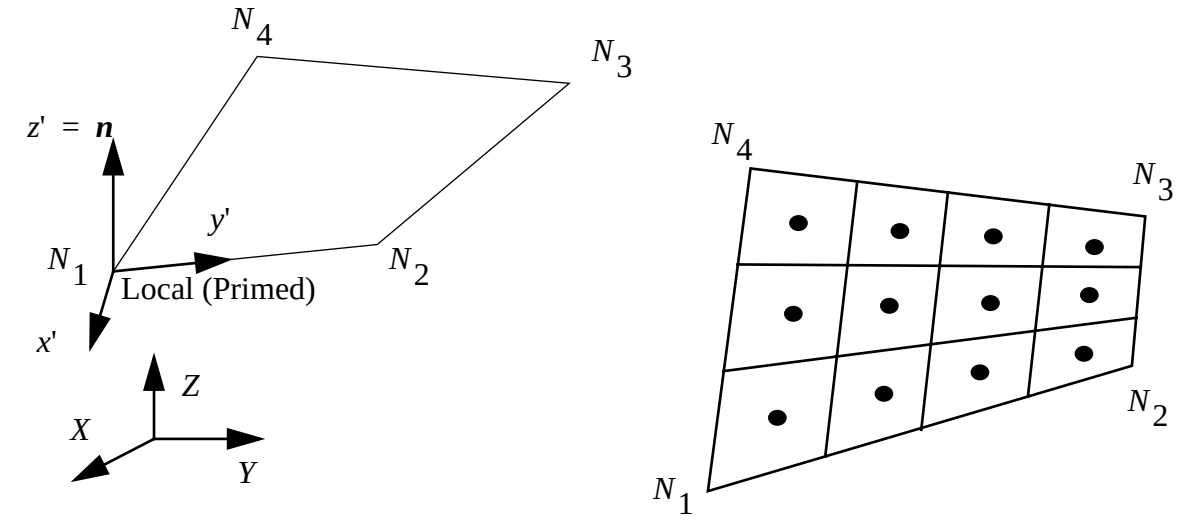

(a) 3-D Quadrilateral (b) 3-D Triangle

Figure 2.2: Radiating Surface Geometries

curled in the direction of increasing nodal point number $(N_1 \rightarrow N_4)$, then the thumb of the right hand indicates the direction of the outward normal. This results in a counterclockwise convention for nodal point numbering, when "viewing" the radiating side of the surface from above. Photons are only emitted from the "front" of the surface (i.e. in the direction in which the surface normal points); the "back" side of the surface is transparent as far as the code is concerned. When specifying the surfaces, they must be convex (for more discussion of convexity, see Section 2.3). For quadrilaterals, the four nodes should be coplanar (to within a small tolerance), but the code is able to handle non-planar quadrilaterals; see Section 2.3 for more details. Additional information on the surface input format is given in Section 3.6.

Each radiating surface is assigned a material number, and all material properties are independent of spatial position on a single surface. More detail on the material properties is given below in Section 2.5.

Referring to Figure 2.3(a), one observes the local (primed) coordinate system with vertex at $N_1$, and axes as shown. Each surface is divided into equally spaced regions, and emission occurs from the centroids of these subsurfaces. The user determines how many subsurfaces each surface in the geometry will be divided into by specifying NDIVX and NDIVY, the number of subdivisions in the x' and y' directions; see Section 3.2.2 for the input specification of these variables. Figure 2.3(b) depicts this for four subdivisions in the y' direction and three subdivisions in the x' direction (here, editorial license has been taken as, in general, x' is not aligned with the lines $N_1$ - $N_4$ or $N_2$ - $N_3$, as it is normal to the y' and z' axes). The dots in Figure 2.3(b) indicate the photon emission points. The number of photons emitted from each point is scaled by the area of its subsurface to eliminate bias that would otherwise result in "hot spots." At least 10 divisions in x' and y' are recommended for uniform surface emission (default is 5). Additionally, at least 100 photons should be emitted from each emission point per loop (specified by the variable NPHTN for each surface) to render insignificant round-off problems in scaling photon emissions to subsurface areas. More detail about specifying the number of photons to emit from a surface is given in Section 2.7.




(a) Global - Local Coordinate Systems

(b) Photon Emission Points

Figure 2.3: Three-Dimensional Surface Details

## 2.3 Split Surfaces, Planarity and Convexity

As mentioned above, while surfaces in MONT3D are supposed to be planar and convex (a convex surface is usually defined as one with no interior angles greater than 180˚), the code can handle non-planar surfaces. It does this by dividing non-planar quadrilaterals into two triangles which are planar and convex by definition. Since the photons emitted by the original surface are proportioned to the two split surfaces and the results for the two split surfaces are combined in the final output, this division is totally transparent to the user. In fact, it is often easy for split surfaces to go unnoticed by the user because MONT3D only issues warnings for these surfaces, not errors.

This is unfortunate because while there is little harm in splitting a slightly non-planar surface, extremely non-planar surfaces may be modelled very badly; they may even be modelled as two overlapping triangles. An example for a *planar* surface is shown in Figure 2.4 where the quadrilateral in the figure is split into the triangles defined by nodes 1-2-3 and 1-3-4. The triangle 1-2-3 is not part of the original quadrilateral and the triangle 1-3-4 completely covers the triangle 1-2-3. While MONT3D would automatically reject a concave quadrilateral such as the one in the figure, it may not catch all ill-defined non-planar split surfaces; this will be discussed in greater detail below. This problem has become more severe recently, since, due to user request, the convexity test has been relaxed, particularly for non-planar surfaces.

If surfaces are non-coplanar, unpredictable results can occur. The user is advised to examine split surfaces carefully. Often the wisest course is to redefine the surfaces to make them planar and convex. Most of the potential problems mentioned above are closely related to the concepts of planarity and convexity. For this reason, the planarity and convexity tests done by MONT3D and how split surfaces relate to them are described below. There are also potential problem related to the number of photons emitted from each emission point which will also be described below.

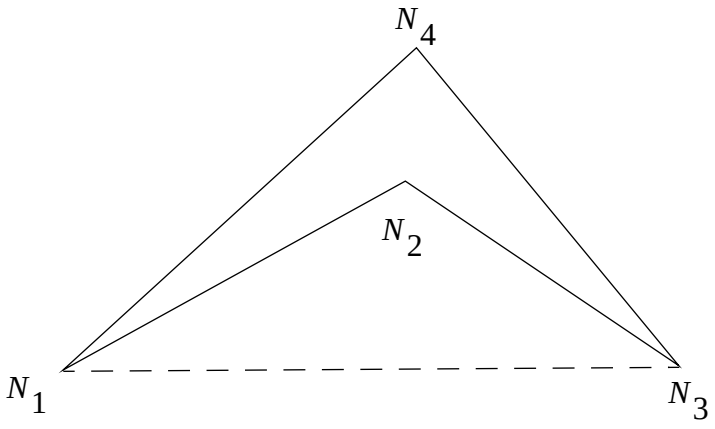



Figure 2.4: Planar Quadrilateral Divided into Two Overlapping Triangles

To assess the planarity of a surface, the dot product of the surface normals of the two triangles defined by the nodes 1-2-3 and 1-3-4 is calculated. If this dot product differs from one by more than a predefined tolerance (SPLITOL, see Section 3.2.3 for the input specification of this variable), the surface is divided into two planar triangles. It should be realized that this results in a geometry different from the original one - i.e. a lack of consistency exists in the original versus simulated geometry. Indeed, greater differences in non-planarity result in greater inconsistencies with the original geometry. If these aberrations are large, unpredictable and unintended results may occur. Indeed, there is no substitute for a well-defined geometry. On the other hand, it should be realized that if the value of SPLITOL is too high, non-planar surfaces will not be split and excessive lost photons and other such problems can occur. When splitting a surface, MONT3D arbitrarily splits the quadrilateral along the line from node 1 to node 3. While in certain cases it may be "better" to split the quadrilateral along the line from node 2 to node 4, there is no known way for MONT3D to determine which way is better, so the surface always splits along the line from node 1 to node 3.

MONT3D also tests for convexity. A common definition for convexity is that no interior angle can exceed 180˚, but a complete definition is much more complex; see Schorn and Fisher [1994]. By definition, a convex polygon is planar, so the convexity tests are not totally applicable to non-planar quadrilaterals. This is unfortunate because it is possible to split a concave quadrilateral into two overlapping triangles, perhaps similar to those shown in Figure 2.4. To try to prevent this, MONT3D applies modified convexity tests to non-planar surfaces. These tests probably do not catch all overlapping triangles, but we know of no foolproof test for non-planar quadrilaterals.

The convexity test in MONT3D has two parts. For the first convexity test, a check is done that the sum of the lengths of each pair of non-adjacent sides is shorter than the sum of the lengths of the two surface diagonals. If this is not the case, the surface is rejected whether it has been split or not and MONT3D exits with an error. For the second convexity test, a check is done of the sum of the areas of triangles comprising the surface for both the cases of splitting along the lines from node 1 to node 3, and from node 2 to node 4. For a planar surface, if the areas do not match within a certain tolerance, the quadrilateral is considered concave and MONT3D exits with an error. The tolerance is given as the percent difference of the areas (PDA) being compared. This tolerance (*pdamax*) is a parameter which can be set by the user; see Section 5.5.2 for more details. Its default

value is 0.1 (a 0.1% difference between the two areas). Non-planar split surfaces often fail this test, but when they do, only a warning is issued.

It should be realized that the PDA for a surface is a good measure of the lack of consistency involved in modelling the surface as two split surfaces or as convex. The larger this value, the greater the chance that the surface is modelled incorrectly, for example, as overlapping triangles. Since MONT3D accepts any non-planar set of split surfaces no matter how large their PDA value, the user must make sure that the split surfaces, particularly those with high PDA values, are properly defined. The MONT3D output file does include a warning that lists all split surfaces with a PDA greater than *pdamax* and their PDA values. The user should consult this list to determine which surfaces may require redefinition.

When a planar surface fails the second convexity test, there are three ways to fix the problem. The only safe way to fix the problem is to redefine the surface so that it is convex. More dangerous solutions are to lower SPLITOL enough to mark the surface as non-planar or to increase *pdamax* enough that the surface is marked as convex. Neither solution is recommended.

When a surface is split, it should emit as many photons per loop as it would if it were not split. To ensure this, while the number of subdivisions in the x and y directions are the same for each split surface, the number of photons emitted per emission point per loop (NPHTN) is divided between the two surfaces weighted by the area of each surface. Frequently, this causes at least one of the split surfaces to emit many fewer than the suggested 100 photons per emission point leading to round-off errors. There may even be significant round-off error in the division of the photons per emission point between the two split surfaces. Therefore, for split surfaces, the user should set NPHTN to values greater than 100. The MONT3D output file lists the area and the number of photons emitted per emission point for all surfaces including split surfaces. More detail on setting NPHTN and other such topics is given in Sections 2.7 and 2.2.

## 2.4 Surface Concatenation

Previous versions of MONT3D permitted surface concatenation. Current versions do not due to the limitations associated with error checking and restart (it is impossible to recover restart information from concatenated information, as some information is lost during the concatenation process). If it is desired, concatenation can be done *a posteriori* by operating on the exchange factor matrices.

## 2.5 Material Properties

The current version of MONT3D has a new material model. The new material model extends the older material model in two respects. First, semi-specular outgoing directional distributions are included for both transmission and reflection. Secondly, material input has been simplified. The input formats for the new and old material models are not compatible. MONT3D will read material property input in either the new or the old input format. The input control variable, NUMMAT - the number of materials, is used to control the format of the material properties to be input. If negative, the new format is used; if positive, the old format is used. See Section 3.2.1 for the input specification of NUMMAT.

The material properties are defined in terms of a local spherical coordinate system. Figure 2.5 defines the cone angle, $\theta$, and the azimuthal angle, $\phi$, with $x'$ and $y'$ in the plane of the surface.

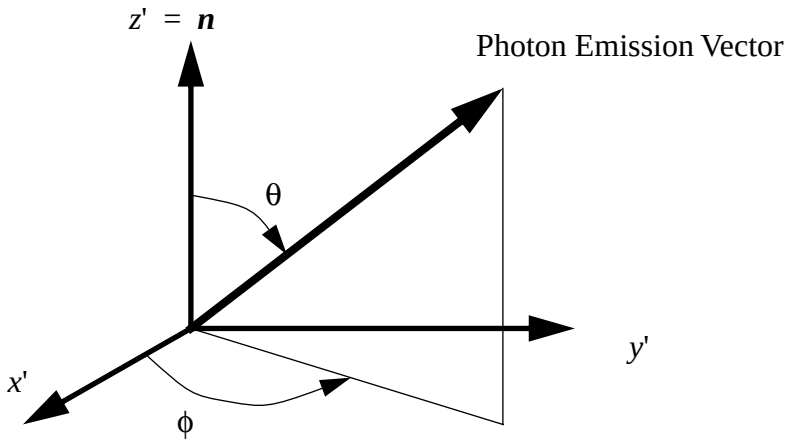



Figure 2.5: Local Material Coordinate System

Material properties are presented below in terms of material property curves, outgoing directional distributions, and emission. Information about the input format for the new material model is given in Sections 3.8-3.10. The old material model is covered in detail in Appendix A.

### 2.5.1 Material Property Curves

Material properties describe the interaction of a particle with a material. Material properties are independent of incoming azimuthal angle, $\phi$, but may be dependent upon incoming cone angle, $\theta$. The properties are defined as constant (gray) within a particular radiative band *k*, so that Kirchoff's law applies within each band. Explicitly:

$$\varepsilon^{k}(\theta) = \alpha^{k}(\theta) = 1 - \rho_{d}^{k}(\theta) - \rho_{s}^{k}(\theta) - \rho_{ss}^{k}(\theta) - \tau_{d}^{k}(\theta) - \tau_{s}^{k}(\theta) - \tau_{ss}^{k}(\theta) \qquad (2.1)$$

where:

$\varepsilon^{k}(\theta)$ = emittance in wavelength band *k* at outgoing cone angle, $\theta$

$\alpha^{k}(\theta)$ = absorptance in wavelength band *k* at incident cone angle, $\theta$

$\rho_{d}^{k}(\theta)$ = diffuse reflectance in wavelength band *k* at incident cone angle, $\theta$

$\rho_{s}^{k}(\theta)$ = specular reflectance in wavelength band *k* at incident cone angle, $\theta$

$\rho_{ss}^{k}(\theta)$ = semi-specular reflectance in wavelength band *k* at incident cone angle, $\theta$

$\tau_{d}^{k}(\theta)$ = diffuse transmittance in wavelength band *k* at incident cone angle, $\theta$

$\tau_s^k(\theta)$ = specular transmittance in wavelength band *k* at incident cone angle, θ

$\tau_{ss}^k(\theta)$ = semi-specular transmittance in wavelength band *k* at incident cone angle, θ

Hereinafter, we drop the explicit dependence upon wavelength band *k*, and carry an implicit dependence. It should be noted that all properties may be considered in terms of probability, i.e. $\rho_d(\theta)$ is the probability that a photon of incident angle θ will be diffusely reflected, $\rho_s(\theta)$ is the probability that a photon of incident angle θ will be specularly reflected, etc. Furthermore, the total reflectance in band *k* is the sum of the specular, semi-specular and diffuse components, and similarly for the total transmittance. The fundamental difference among the specular, semi-specular and diffuse components is the shape of the outgoing directional distributions for these components (covered in Section 2.5.2 below).

Since material properties are both constant in each wavelength band and independent of ϕ, the emittance and absorptance for a given wavelength band and θ are equal by virtue of Kirchoff's law. The absorptance and emittance here are determined as the complement of the other properties. Thus, specification of the diffuse, specular and semi-specular reflectances and transmittances as functions of the incident angle uniquely defines the incident properties within one wavelength band for a material. In the case of ordinary emission, the directional emittance is determined from the absorptance. However, other possibilities for emission exist, as described below in Section 2.5.3.

Figure 2.6 depicts the material property curves as functions of the incident cone angle, $\theta_i$ within a wavelength band *k*. At any particular value of incident cone angle, $\theta_i$, an incident photon has the

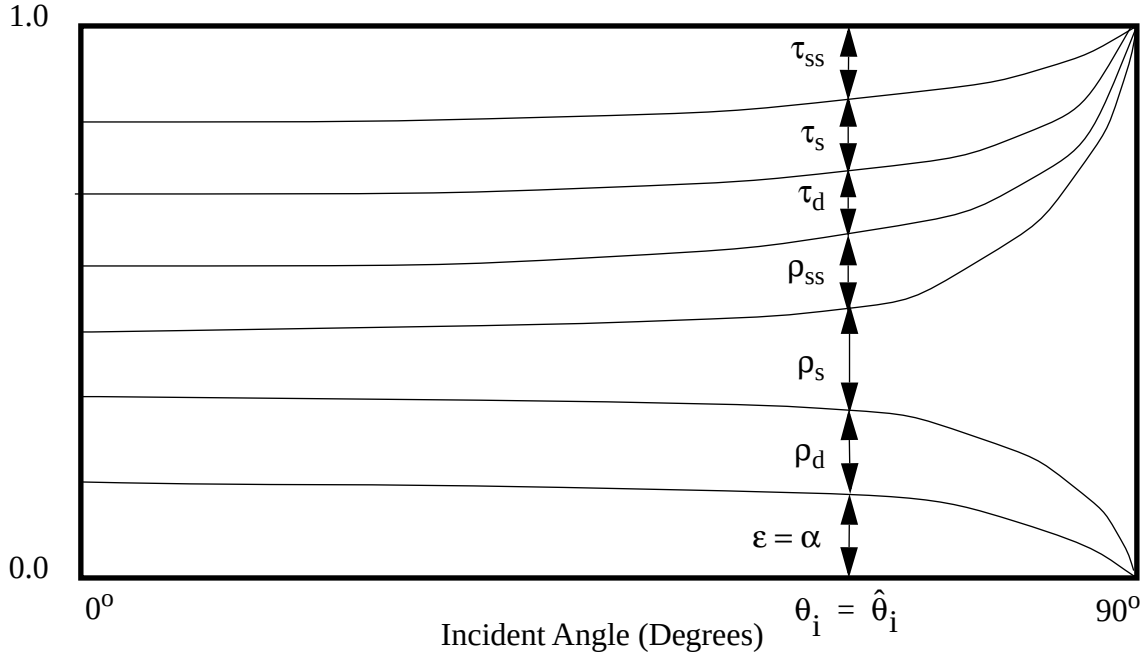



Figure 2.6: Material Properties vs. Incident Cone Angle

following probabilities: $\rho_d(\theta_i)$, $\rho_s(\theta_i)$, $\rho_{ss}(\theta_i)$, $\tau_d(\theta_i)$, $\tau_s(\theta_i)$, $\tau_{ss}(\theta_i)$, and $1-\rho_d(\theta_i)-\rho_s(\theta_i)-\rho_{ss}(\theta_i)-\tau_d(\theta_i)-\tau_s(\theta_i)-\tau_{ss}(\theta_i)$ for diffuse, specular, and semi-specular reflection, diffuse, specular, and semi-specular transmission, and absorption (or emission), respectively. Curves of material properties must be input for every wavelength band for each material. In the old model, these curves had to be input by point value as a function of cone angle. The new material model allows each of these properties to be input as a single, constant value, or by reference to a curve which must subsequently be input by point value as a function of cone angle. In the case of point value input, the computer code parabolically interpolates between each three successive points entered. In this case, care must be taken to: (1) include bounding points of $\theta = 0^o$ and $\theta = 90^o$ (since no extrapolation is done), and (2) to include enough points, varying smoothly, to result in good interpolation (i.e., discontinuous jumps must be input as "steep" parabolas with three non-coincident points used to define the jump). If curves are input by point value, then it is strongly suggested that the plot file (file suffix .plt) and MPLOT [Nagesh and Burns, 1994] be used to view the curves created, to ascertain that the interpolation is physically reasonable.

### 2.5.2 Outgoing Directional Distributions

Photon/material interactions of seven types may occur. These outgoing directional distributions will be discussed briefly below. A more detailed discussion of these distributions is given by Burns and Pryor [1999]. In the current discussion, for purposes of illustration, these outgoing directional distributions will be presented graphically for reflection only, and depicted in cross-section. To establish a basis for the ensuing discussion, the definition of the bidirectional reflectance is recalled as $\rho''(\theta_i, \phi_i, \theta_o, \phi_o)$, where the double prime indicates that it is dependent upon two directions, incident $(\theta_i, \phi_i)$ and outgoing $(\theta_o, \phi_o)$.

Figure 2.7(a) depicts the traditional model for reflection, consisting of a purely specular component and a purely diffuse component. The current model in MONT3D includes three enhancements to the traditional model. First, the diffuse distribution may be weighted in $\cos(\theta)$ to allow control of the shape of the outgoing directional distribution. Second, a semi-specular outgoing distribution is added. The semi-specular distribution is "spread" in space to represent more realistic materials. Third, the semi-specular distribution may be modeled using an offset about the specular angle, $\Delta\theta_o(\theta_i)$, which usually tends toward the grazing from the specular angle. These features have been observed by Torrance and Sparrow [1966], and are shown conceptually in cross-section in Figure 2.7(b).

Three components of reflectance are modelled: weighted diffuse, specular and semi-specular. Each is discussed below.

The weighted diffuse component exhibits a symmetric shape of revolution about the surface normal. The weighting exponent, $r_d$, controls how much the distribution tends toward the normal direction. The probability distribution functions (PDF's) for energy (not intensity) for the weighted diffuse distribution are:

$$PDF(\theta_o) = (r_d + 1)\cos^{r_d}(\theta_o)\sin(\theta_o) \tag{2.2}$$

$$PDF(\phi_o) = 1/(2\pi) \tag{2.3}$$

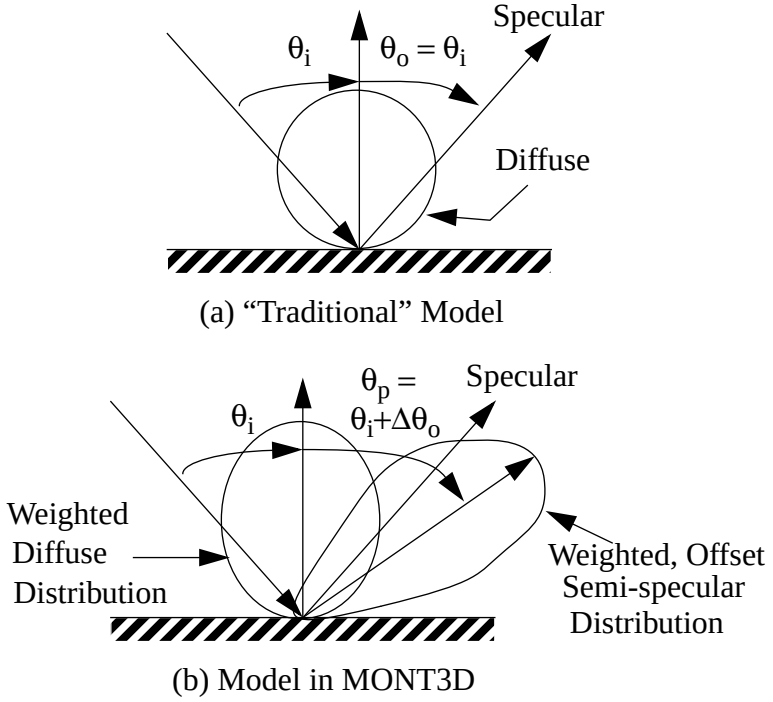



Figure 2.7: Cross-sectional Views of Reflectance Models

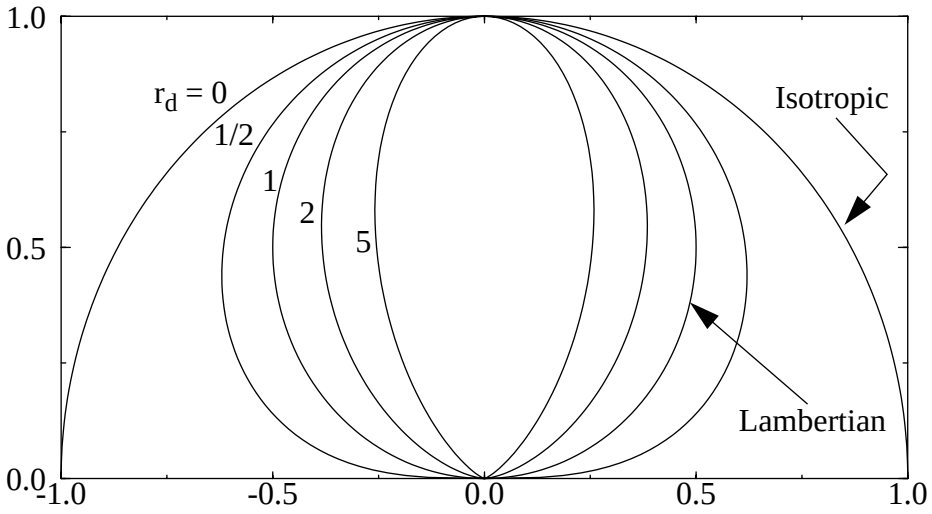



Figure 2.8: Cross-sectional View of Weighted Diffuse, Normalized PDF’s

The coefficient ($r_d$+1) in Equation (2.2) effects normalization of the distribution when integrated over $\theta$ from 0 to $\pi/2$. The normalized diffuse reflectance distributions are shown in Figure 2.8, with $r_d$ as a parameter. Note that $r_d = 1$ represents the standard diffuse (Lambertian) distribution, and $r_d = 0$ represents an isotropic distribution. Values of $r_d$ exceeding 1 bias the distribution toward the

normal, termed “over cosine,” while values less than one bias the distribution toward the grazing angle, termed “under cosine.”

For specular reflection, the photon’s directional component normal to the surface is reversed, and the component parallel to the surface remains unchanged. No specular offset angle is modelled in this distribution.

The semi-specular reflectance distribution takes this specular distribution and: (1) adds an offset cone angle $\Delta\theta_o(\theta_i)$ to yield the “preferred” outgoing direction, and (2) spreads the distribution in space, including a weighting coefficient, by revolving the distribution about the “preferred” outgoing direction. The “preferred” outgoing cone angle is determined as the specular angle plus the semi-specular offset angle:

$$\theta_p = \theta_i + \Delta\theta_o(\theta_i) \tag{2.4}$$

The same type of distributions shown in Figure 2.8 for the diffuse outgoing distributions are used for the semi-specular outgoing distribution, except: (1) the distribution is scaled in θ to disallow penetration into the surface i.e., the distribution is forced to zero at the grazing angle, and (2) symmetry about the preferred outgoing direction, rather than about the surface normal, is applied. Here, the weighting coefficient, $r_{ss}$, controls the tendency of the distribution to go toward the preferred outgoing direction. Figure 2.9 depicts cross-sectional views of normalized outgoing distributions about preferred outgoing directions of $30^o$ and $70^o$. The distributions shown in Figure 2.9 are revolved about the preferred outgoing direction, spreading the distribution in cone and azimuthal angles. It should be noted that as $r_{ss}$ approaches infinity, the semi-specular distribution approaches a specular distribution at the preferred outgoing angle.

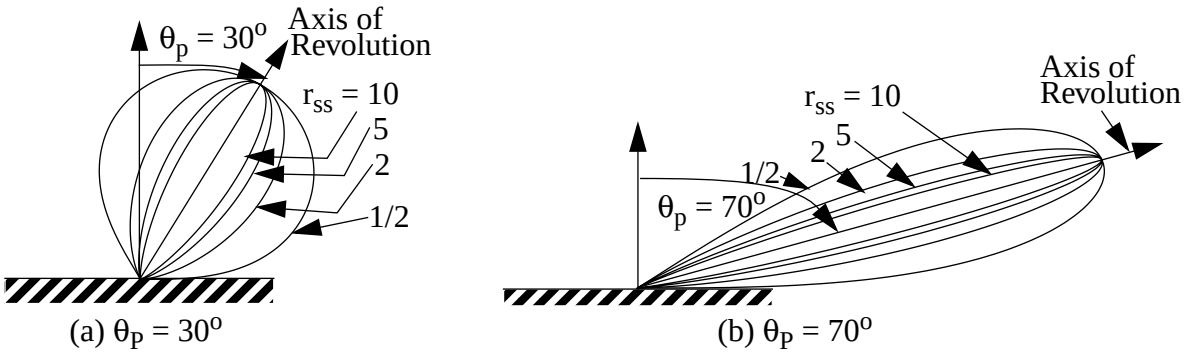



Figure 2.9: Normalized, Weighted Semi-specular Directional Distributions (Shown in Cross-section)

These same three distributions apply to transmission, except that the distributions pass through the surface instead of being reflected from it. These distributions for transmission allow approximate modelling of semi-transparent materials, such as glass, and translucent materials, such as lighting diffusers. Although the material models in MONT3D permit accurate modelling of energy, no refraction is modelled, introducing an error in direction. However, in many instances the directional changes induced by refraction are small, and may be neglected with good accuracy.

Care should be taken when modelling transmission. All exterior surfaces of a geometry must be totally non-transmissive. If this is not the case, photons transmitted through the exterior surfaces will be lost. MONT3D does not check for this and no warning will be issued during the input phase if this error occurs.

In addition to being reflected or transmitted, the photon may be absorbed. In summary, seven possibilities for incident photon disposition exist: three components of reflection, three components of transmission, and absorption. For each material in each band, the diffuse and semi-specular weighting coefficients, $r_d$ and $r_{ss}$, for both reflection and transmission must be input. Furthermore, for semi-specular reflection and transmission, the offset angle as a function of incident angle $\Delta\theta_o(\theta_i)$ must also be input. More details on the input format are given in Sections 3.8-3.10.

### 2.5.3 Emission

In the new material model, emission type is defined using the variable, IETP. Emission of one of three types may be specified, depending on the whether IETP is 0 (the default), 1 or 2, as discussed below. See Section 3.8.2 for additional details. Emission is specified quite differently in the old material model. Appendix A, specifying old material model and input format, is provided for reference. Hereinafter in this section, only the new material model will be presented.

**Standard Emission - IETP = 0.** This is the type of emission deriving from Kirchoff's Law. At each angle, the emittance is equal to the absorptance - determined as the complement of the reflectance and transmittance. This type of emission adheres to the second law of thermodynamics, and is recommended for radiative transfer. If this type of emission is used, no additional input for emission is required.

**Beam Emission - IETP = 1.** This type is intended to model "beam" or collumnated radiation. Emission of this type is accomplished along the direction specified for this material by EBX, EBY, and EBZ - the directional components in global coordinates of the emission vector. Here, emission is unidirectional - all photons emitted from this material type travel in the direction specified by the global components, only the point of emission changes. Different material types must be defined for each global direction of emission desired. When specifying a material of this type, care must be taken to ensure that, as discussed in Section 2.2, the beam emission will be from the "front" of the surface, not the "back." In other words, the dot product of the emission direction and the surface normal must be greater than 0 for all surfaces that use the material. If this is not the case, the code exits with an error. Note that all interactions (absorptions, transmissions, reflections) will depend only on the incident material properties defined for this material.

**Function Emission - IETP = 2.** Here, emission is accomplished using computer code supplied by the user; more details are given in Section 3.8.2. Either EBX, EBY, and EBZ, or both $\theta$ and $\phi$ must be specified (the example in the code supplies a function only for $\theta$, emission is uniformly distributed in $\phi$). The user is cautioned that emission via this option is accomplished using the accept-reject method, and many trials may occur for each photon emission if the magnitude of the user-supplied function is small. Photon/material interactions are determined by the incident material properties defined for this material.

### 2.5.4 Cases Where the Reciprocity Relations Do Not Hold

Care must be taken when using some of the more advanced material property features of MONT3D. Unless the directional model for emission is consistent with the directional model for surface properties, the reciprocity relations, Equations (1.5) and (1.6), may no longer hold. The other conservation relations still hold, however. Conditions in which the two models do not match include: function and beam emission; diffuse reflectance and transmittance with $r_d$ not equal to 1 and any case involving semi-specular reflectance or transmittance. When modelling radiative transfer, it is suggested that the user avoid the above conditions. The reciprocity relations also are not observed when the diffuse reflectance varies as a function of angle. Surprisingly, they DO hold when the specular reflectance varies as a function of angle. No tests have been done to assess if reciprocity holds when diffuse or specular transmittance varies by angle. In the future, investigations will be done to determine how MONT3D can be modified to observe reciprocity relations under the above conditions.

## 2.6 Shading

If any shading exists in the geometry, a photon's path may intersect a number of surfaces. The distance from the emission point to the different intersection points must then be computed, and the closest point chosen as the true intersection point (the surface first encountered). For large problems, this results in significant expense. To reduce execution time for large problems with shading, the grid shading algorithm [Margolies, 1986] has been implemented. Here, the geometry is divided into a series of grid cells, resulting in rectangular parallelepipeds. The photon is traced from grid cell to grid cell, and a search is done within each grid cell only over those surfaces which exist either wholly or partly within that cell. The situation is depicted in Figure 2.10 for a 2-D geometry.

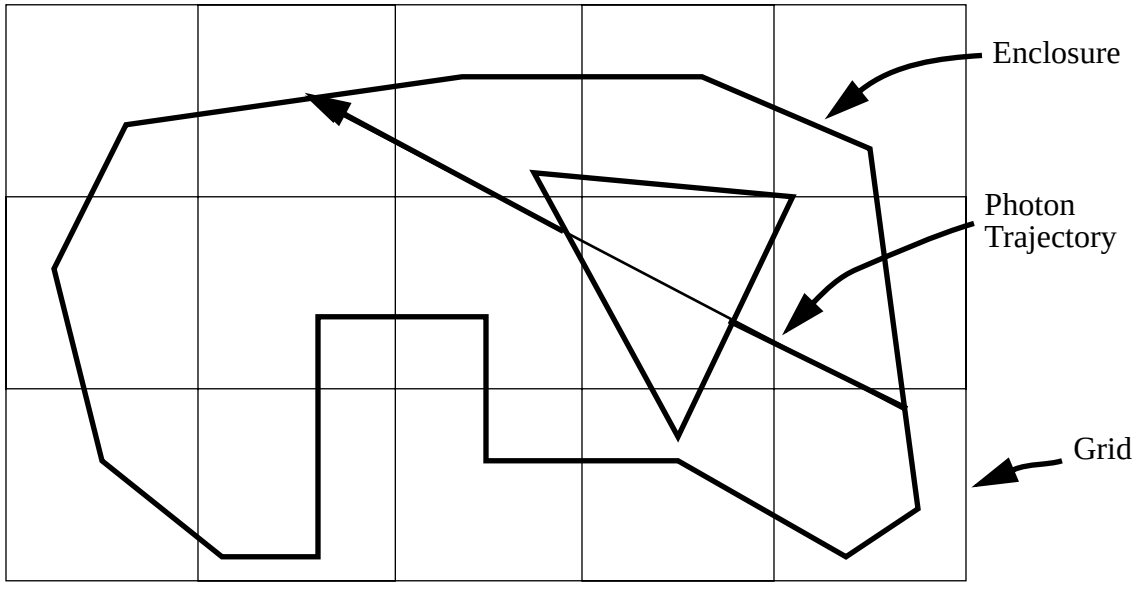



Figure 2.10: 2-D Illustration of the Grid Shading Algorithm

In the recent recoding of MONT3D [Zeeb, 1997; Zeeb and Burns, 1999], the grid tracing routine was further optimized and use of this algorithm has resulted in speed-ups in execution time in tracing, compared to no grid, of factors of 18 to factors of 81 for large geometries (1,000 to 5,000 surfaces). The optimum grid differs from problem to problem, and must be empirically determined by the user. However, a good starting guess for a large geometry (1,000 to 5,000 surfaces) is about 15,000 grid cells.

Grid coordinates are defined separately for each axis. Two options exist for defining the grid coordinates along each axis: (1) a uniform grid generated by the program, with equally spaced grids, or (2) a user-defined grid. For option (1), the user need specify only the number of grids along the axis, while for option (2) the user must also specify the grid coordinate locations along the axis. See Sections 3.3 and 3.11 for additional details.

## 2.7 Number of Photons, Convergence and Accuracy

The number of photons emitted in each photon loop is specified for each surface in each band by the relation: # of photons = NDIVX x NDIVY x NPHTN. Multiple photon emission loops may be done per surface; more detail is given below and in Sections 2.2, 3.2, and 3.6. As Monte Carlo techniques are statistical in nature, “enough” photons must be emitted from each surface to yield a statistically accurate result. This number depends upon the geometry and, to some extent, upon the material properties. As a general rule, greater numbers of surfaces require greater numbers of photons. Execution time increases linearly with number of photons. For small problems (about 20 surfaces), it has been found that on the order of 20,000 photons per surface (not subsurface) are required to achieve exchange factors accurate to within about 5%. It is typical to observe convergence in a particular exchange factor as shown in Figure 2.11. The user is cautioned that false convergence may be indicated when comparing two values on the curve as shown. It is therefore wise to check the entire matrix of exchange factors for consistency at several numbers of photons.

To estimate the number of photons required to achieve a given level of accuracy, Table 2.1 is provided. The table gives, for each exchange fraction $F_{ij}^{k}$, the number of photons, $N_{i}^{k}$, which must be emitted from a surface to achieve 95% confidence that the exchange fraction is within: 1%, 2%, 5%, 10%, and 50% of the exact answer. The numbers of photon emissions per surface are calculated from the formula for confidence intervals, $C_{ij}^{k}$, for the exchange fraction from surface *i* to surface *j* in wavelength band *k* ($F_{ij}^{k}$), derived by Maltby [Maltby, 1990]:

$$C_{ij}^{k} = z\sqrt{\frac{1 - F_{ij}^{k}}{N_{i}^{k} F_{ij}^{k}}} \tag{2.5}$$

where *z* is taken from the standard normal tables [Kreyszig, 1993], and is 1.96 for 95% confidence. Equation (2.5) yields the fractional accuracy in $F^{k}{}_{ij}$ (n.b., 100 times this value is the percent accuracy).

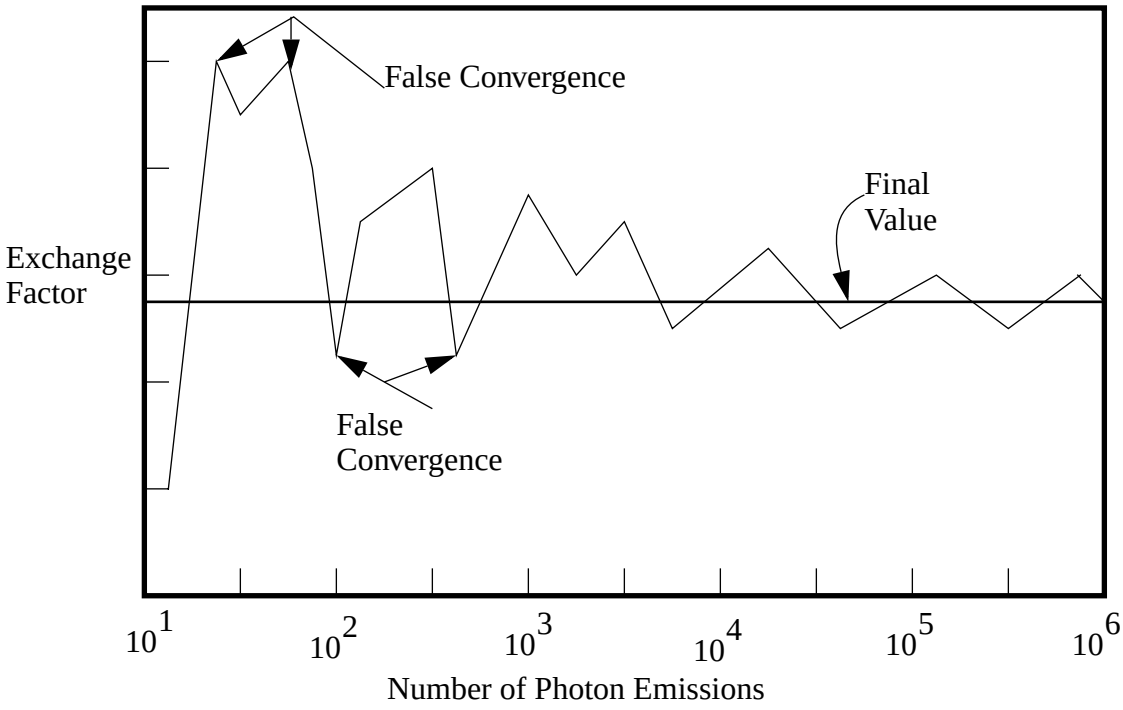



Figure 2.11: Convergence vs. Number of Photon Emissions

Table 2.1: Accuracy in Exchange Fractions

| Exchange Fraction | Level of Accuracy | | | | |
|---|---|---|---|---|---|
| | 1% | 2% | 5% | 10% | 50% |
| $10^{-3}$ | 38,377,584 | 9,594,396 | 1,535,103 | 383,776 | 15,351 |
| $10^{-2}$ | 3,803,184 | 950,796 | 152,127 | 38,032 | 1,521 |
| $10^{-1}$ | 345,744 | 86,436 | 13,830 | 3,457 | 138 |

MONT3D is formulated to attain a specified accuracy for each row *i* of the exchange factor matrix. The program is constructed to loop over successive emissions from each surface if a preset accuracy tolerance is not met after a full surface emission. To explain this, we first note that Equation (2.5) provides the confidence interval for only element *ij* of the exchange fraction matrix, when emitting additional photons actually increases the accuracy of all elements in row *i*. Equation (2.5) is modified to account for this with the rationale that exchange fractions affect the accuracy proportional to their size. Thus, we weight each confidence interval by its exchange fraction, sum and then average by dividing this amount by the total number of surfaces, NSURF, to yield the *ad hoc* row confidence factor for row *i*, $C_i^k$:

$$C_i^k = \frac{1}{\text{NSURF}} \sum_j C_{ij}^k F_{ij}^k = \frac{z}{\text{NSURF}} \sum_j \sqrt{\frac{F_{ij}^k (1 - F_{ij}^k)}{N_i^k}} \tag{2.6}$$

If the confidence factor for emissions from surface *i* in band *k* is not met after a full surface emission loop, then the program continues to perform full surface emission loops until either the specified confidence factor is achieved, or a maximum number of loops over the surface, NPLOOPS, have occurred. This feature can be used with the restart option to effect a specified accuracy for surfaces, traded off against CPU usage.

The user has quite a bit of control over how many emissions occur for each surface. While NDIVX, NDIVY, and NLOOPS are the same for all surfaces, NPHTN and the confidence factor can be specified for each surface; see Sections 3.2, 3.4, and 3.6 for additional detail. It should also be noted that the value of *z* used by the code for calculation of the row confidence factor can be set by a parameter; see Section 5.5.2 for more details. The default value is 1.96 which represents 95% confidence.

A rough guess of the size of the exchange fractions is the reciprocal of the number of surfaces in the input file. This yields the "average" exchange fraction size, since the sum of any row of the exchange fraction matrix is 1. The number of photons required to be emitted to achieve an "average" level of accuracy may then be estimated from Equation (2.5) or obtained from Table 2.1 through interpolation or extrapolation.

Errors in the temperatures calculated from radiative flux balances are smaller than errors in the exchange fractions due to the fourth-root dependence of temperature upon radiative flux. For small errors, one may expect the errors in temperatures to be about one-fourth of the errors in fluxes. Emitting an equal number of photons from each surface may result in a waste of computer time, since some surfaces contribute little to the radiative exchange. A better approach would be to apportion the numbers of emissions to each surface based upon its estimated power output. This is an approach which requires judgement gained through experience with specific problems, since the power outputs are generally not known *a priori*. In any case, the above approach provides a "potentiometer" which can be used judiciously to adjust solution accuracy.

## 2.8 Aid in Debugging and Visualizing MONT3D Geometries

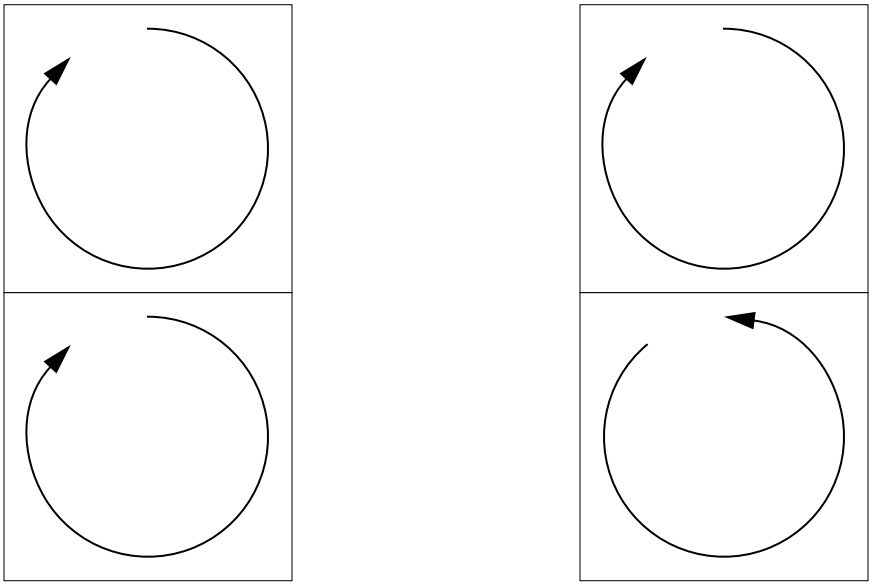
MONT3D generates several files useful in debugging and visualizing geometries. These files are: the plot file, the leaks file, the lost photon file, and the trajectory file. The various files are described below. All can be visualized using the stand-alone graphics program MPLOT [Nagesh and Burns, 1994]. The actual formats of each of these files are presented in detail in Appendix B.

The plot file (file suffix .plt) containing all the information in the input file (geometry and material properties) is written to disk during the input phase. This file may be used with MPLOT to display the geometry and the material property curves. This file must be read by MPLOT before viewing the results from any of the other files listed above; otherwise, MPLOT will have no geometry over which to display the results.

An error in the specification of the geometry often results in a "leak" or hole in the enclosure, through which photons may be transmitted and "lost." Leaks may be caused due to disjoint surfaces, missing surfaces, incorrect node numbering on a surface, misplaced nodal points, or insuffi-

cient precision in specifying coordinates. During the input phase, the geometry is checked for leaks, and results are written to the output file. Additionally, an ASCII file of potential leaks (file suffix .lks), identified by type (severity of leak), is also created. This file may be used with MPLOT to highlight surfaces and sides of surfaces which have been identified as potential problems or leaks. The three types of errors identified are:

**Error 1- Reversed Edge:** This error occurs when two surfaces share the same edge. An example is given in Figure 2.12. In Figure 2.12(a), the edge between the two surfaces is correct and both surfaces have normals pointing into the page. In Figure 2.12(b), the edge is reversed and the normals of the two surfaces point in opposite directions. Recall from Section 2.2 that surfaces are transparent to photons which hit the back side of the surface. This type of error usually results in the enclosure losing excessive photons resulting in an error termination.



(a) correct edge (b) reversed edge

(Arrows represent direction of increasing node number)

Figure 2.12: Example of a Reversed Edge

**Error 2 - No Match Found:** This error occurs when no match is found connecting at least one side of that surface to the side of another surface. This may or may not be acceptable.

**Error 3 - Slip Surface:** This error occurs when an edge goes through a node point instead of terminating at it. Although this is sometimes an actual error, it often is not. If the slip surface creates no “holes” in the geometry, then it should not cause the enclosure to lose photons.

When photons are lost, the endpoints of each photon ray are written to a separate file (file suffix .lst), which may then be read by the MPLOT program. Then trajectories of the lost photons may be displayed on the geometry. Because there is no terminus of the ray, a fictitious endpoint is used.

An option is also available to write trajectory information to an output file (file suffix .trc), which may be subsequently read and plotted by the program MPLOT; see Section 3.2.2 for more details. This feature is useful in obtaining a "feel" for the underlying physical processes, and to ascertain that the simulation is proceeding as planned. There is a copious amount of information written to the output file during the exercise of this option, so it is suggested that the number of photons emitted per surface be less than about 100.

Recent versions of MPLOT allow the user to view convergence information on the exchange matrix values for a specified surface. Unfortunately, MPLOT can not read the new exchange matrix file format used by latest version of MONT3D and this feature is not available for MONT3D versions 2.4 and later.

For more information, the reader is urged to consult the MPLOT documentation [Nagesh and Burns, 1994].

## 2.9 Restart Capability

The code has been designed to be restarted from a previously computed state. For example, the code may experience a "crash" during execution (for any of a host of reasons). Alternatively, the code may run to completion, and subsequent examination of the answers indicates that they are not sufficiently accurate. In either case, it is desirable to begin a new simulation from the last state available to take advantage of previous work. This prevents waste of computer resources in recomputing information already available. A simulation may be restarted multiple times, until the desired level of accuracy is attained.

To effect this, the current state of the solution must periodically be written to disk, so that it will be available for a restart run. How often this is done is controlled by the input variable NEBLOCK. All information required for restart is written to a restart file (suffix .rst) after emission has been completed from each "block" of |NEBLOCK| surfaces; see Section 3.2.1 for more information. Chapter 4 includes an example of a restart run.

## 2.10 Parallel Version

MONT3D now supports parallel execution implemented in PVM [Greist, et al., 1994]. The parallel version uses a master-worker model. The basic operation of the parallel code is as follows. The master process reads the input file and checks it for errors. If no errors are found, the master spins off a user specified number of worker processes (NWPROC; See Section 3.2.2) and passes to them the input file name and other required information. Next, all the worker processes read the input file in an abbreviated input phase. Since the input file has already been checked by the master, many of the checks, such as the time consuming check of the validity of the geometry, can be skipped during the worker input stage. As soon as a worker completes the input phase, it sends a "worker ready" message to the master. The master then sends a block of |NEBLOCK| surfaces to the worker. The worker performs the work associated with emissions from these surfaces. Once it completes these emissions, it writes the results to the disk and sends a message to the master indicating it has completed that block, and the master issues another block of surfaces to the worker, if any are left. This cycle continues until all emissions have been completed. If a worker encounters a fatal error, it aborts and the block it is working on is done later by another worker. If all workers abort due to fatal errors or if a block of surfaces results in fatal errors for two different workers, the

entire run is aborted and the program terminates. After all emissions are done, the master combines the results from the workers into one exchange matrix file and the program terminates.

The parallel version of MONT3D has been tested on a wide variety of platforms. PVM (Parallel Virtual Machine) was originally designed to combine a number of heterogeneous computers into a parallel virtual machine using the TCP/IP networking protocol. The master and the workers can be totally different types of machines and each machine must be accessible via the network. A diagram of this topology is shown in Figure 2.13. Each of the switches in the diagram may be either

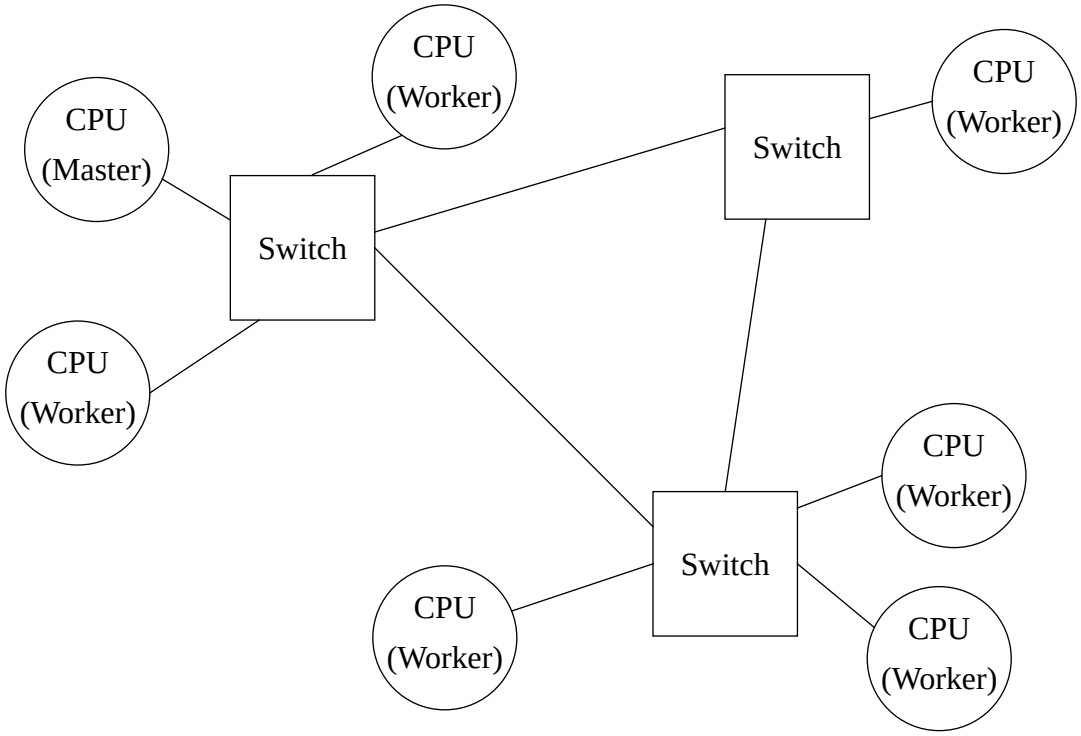



Figure 2.13: Example Master-Worker Architecture

a switch between subnets or a gateway (router) between different networks. The parallel code has been shown to work on a homogeneous network of Hewlett Packard workstations and a heterogeneous network of three Sun workstations and a Linux workstation. Excellent results running the code on a tightly couple cluster of eight DEC Alpha CPU's have also been obtained. The only requirement for the parallel version of the code is that all CPU's must use a common file system such as NFS, as the workers must be able to access the scratch (file suffix .scr) file created by the master and the master and the workers must be able to access the block exchange matrix (file suffix .bni) files. More information about the parallel version of the code is given in Section 5.6.

## 2.11 Pseudo-Random Numbers

Random Number Generators (RNG's) are fundamental components of Monte Carlo analyses, without which Monte Carlo simulation would be virtually impossible. In the codes described herein, uniformly distributed pseudo-random (also termed just random) numbers are used to determine: (1) outgoing directions of particles, and (2) interactions of particles with surfaces. Mathematically, the sequence of random numbers used to effect a Monte Carlo model should possess the following properties: (1) the sequences of random numbers should be serially *uncorrelated,* 2) the generator should be of *long period,* 3) the sequence of random numbers should be *uniform, and unbiased*, and 4) the generator should be *efficient*.

In the past, linear, congruential generators (LCG's) were typically used to produce pseudo-random numbers. However, a new class of generators, lagged Fibonacci generators (LFG's), have recently emerged. LFG's have better properties than LCG's, are more efficient, and are readily amenable to parallelization. An overview of these types of generators is presented, followed by some implementation details. Additional material may be found on-line [Burns and Pryor, 1995; Zeeb and Burns, 1997] and from other sources [Anderson, 1990; Brent, 1992; Burns and Pryor, 1999; Marsaglia, 1985].

### 2.11.1 Lagged Fibonacci Generators

Lagged Fibonacci pseudo-random number generators are based upon the Fibonacci sequence [Golomb, 1982]. The Fibonacci sequence is generalized to a family of pseudo-random number generators of the form:

$$X_n = (X_{n-l} + X_{n-k}) \bmod m \qquad l > k > 0 \tag{2.7}$$

where mod is the integer remainder function and $l$ initial integer values, $X_0$, ..., $X_{l-1}$, are needed in order to compute the next element in the sequence. In this expression the "lags" are $k$ and $l$, so that the current value, $X_n$, is determined by the value of $X$ $k$ places ago and $l$ places ago. In addition, for most applications of interest, $m$ is a power of two. That is, $m = 2^M$. Random real numbers, $R_n$, are generated from the above integer numbers by dividing the integer $X_n$ by $m$, viz.

$$R_n = \frac{X_n}{m} \text{ yielding } 0 \le R_n < 1 \tag{2.8}$$

This type of pseudo-random number generator has been extensively tested for randomness properties using Marsaglia's [1985] DIEHARD tests, and has been given high marks. The only deficiency found is related to what Marsaglia terms the Birthday Spacings test. Preliminary work by Brent [1992] suggests that LFG's pass the Birthday Spacings test if $l$ is greater than 100. It should be noted that LCG's did not do nearly as well on the tests. A firm theoretical understanding of the cycle structure of these generators has been established in a series of two papers by a group at the Center for Computing Sciences [Pryor et al., 1994; Mascagni et al., 1995].

With proper choice of $k$, $l$, and the first $l$ values of $X$, the period, $P$, of this generator is equal to $(2^l - 1) \times 2^{(M-1)}$. For example, for a small generator with $l = 17$, and $M = 31$, the period, $P$, is huge, $\sim 1.4 \times 10^{14}$. The only condition on the first $l$ values is that at least one of them must be odd. Still, for some applications, one should refrain from using more than $2^l - 1$ of the numbers generated by

these generators because for a generator with period, $P$, $R_n$ and $R_{n+P/2i}$ differ by at most $i$ bits ($0 < i < M$) [Brent, 1992]. Examples of four commonly used versions of these generators (LFG($l$,$k$,$M$)) are:

1) LFG(17, 5, 31): P ~ $2^{47}$ (~ 1.4 x $10^{14}$); $2^l$ ~ 1.3 x $10^5$

2) LFG(55, 24, 31): P ~ $2^{85}$ (~ 3.9 x $10^{25}$); $2^l$ ~ 3.6 x $10^{16}$

3) LFG(127, 97, 31): P ~ $2^{157}$ (~ 1.8 x $10^{47}$); $2^l$ ~ 1.7 x $10^{38}$

4) LFG(607, 273, 31): P ~ $2^{224}$ (~ 5.7 x $10^{191}$); $2^l$ ~ 5.3 x $10^{182}$

Even using the more restrictive $2^l$ -1 constraint instead of the full period, all but the LFG (17, 5, 31) have random series much longer than a LCG's maximum period (about $2^{31}$ ~ 2.1 x $10^9$). The most severe drawback of an LFG is the fact that $l$ words of memory are required to be kept current, and this is not significant in our application.

It is necessary to fill the initial state to start the sequence. For the LFG, an initial state of $l$ words is needed. When implementing the code in parallel, it is important that each worker process have a sequence or cycle of random numbers that is independent of all the other processes. Initializing separate cycles is addressed by Mascagni, et al. [1995], where they describe a canonical form for initializing Fibonacci generators. This canonical form is determined solely by $l$ and $k$, and is independent of $M$. In general, the canonical form for initializing Fibonacci generators requires the $l^{\text{th}}$ word be set to all zero bits, and the least significant bits of all words in the register set to zero, with the exception of one or two characteristic bits that depend on $l$ and $k$. Each combination of the remaining ($l$ - 1)($m$ -1) bits, called equivalence classes (EC's), specifies a different cycle of random numbers. No two EC's share the same cycle. Cuccaro [1996] has extensively tested initializing the EC's bits with a binary shift register generator (i.e. an LFG with $M$ = 1) and found excellent randomness properties across parallel sequences with this method, even when the initial seeds for each sequence are very similar. This is the method employed in MONT3D. Additional detail may be found in Burns and Pryor [1999] and Zeeb and Burns [1997].

### 2.11.2 Implementation Details

Experience has shown that using LFG (17, 5, 31) can lead to about 0.1% systematic error in some applications [Zeeb and Burns, 1997; Zeeb and Romero, 1999], probably due to the fact that, as mentioned above, correlations appear after about 130,000 ($2^{17}$ - 1) random numbers are generated. This error is generally well below the accuracy inherent in Monte Carlo simulations. Furthermore, these correlations probably have little effect on all but the most simple of problems.

However, to ensure high accuracy runs, we have chosen to implement LFG(127, 97, 31), which has the smallest value of $l$ that passes all of Marsaglia's DIEHARD tests, including the Birthday Spacings test. Correlations do not appear among the random numbers until after about 1.7 x $10^{38}$ random numbers have been generated, which is more than adequate for any problem in the foreseeable future. It is possible to compile the code so that it implements a different length LFG generator, see Section 5.5.2 for more details.

The RNG is initialized as follows. First, the initial seed, $X_0$, which is either given by the user, generated from the time and date, or set to the default value, is obtained. If the initial seed is generated from the time and date, then the following formula is used:

$$X_0 = \text{isec}*2^{25} + \text{imin}*2^{19} + \text{ihr}*2^{14} + \text{iday}*2^{9} + \text{imon}*2^{5} + \text{mod}(\text{iyr}, 2^{5}) \quad (2.9)$$

where mod is the integer remainder function and the integer variables on the right-hand side are the second of the minute (0-59), the minute of the hour (0-59), the hour of the day (0-23), the day of the month (1-31), the month of the year (1-12), and the year of the century (00-99). More detail on specifying the initial seed is given in Section 3.2.1.

Next, the initial seed is used in the binary shift register to fill the initial state of the LFG according to the canonical form as described by Burns and Pryor [1999]. Also, each binary shift register is stepped through 64 bits before starting to fill the rectangular region in the canonical form, to avoid starting at a "flat spot" on the cycle. See Pryor et al. [1994] for a discussion of these flat spots.

When executing the code in parallel, the lowest-order bits of the initial 32-bit seed (usually obtained from Equation (2.9) above) are masked with the worker process number - 1, as shown in Figure 2.14. In the figure, ten bits, which are enough bits for up to 1,024 worker processes, are masked. The exact number masked depends on the number of worker processes being used. Note that when a time generated seed is used, the most significant bits derive from the time, and are the same for all tasks, but vary from run to run as does the time at the start of the run. Since all RNG's are initialized using the binary shift register, this methodology produces parallel, independent streams of very high quality random numbers on each worker process.

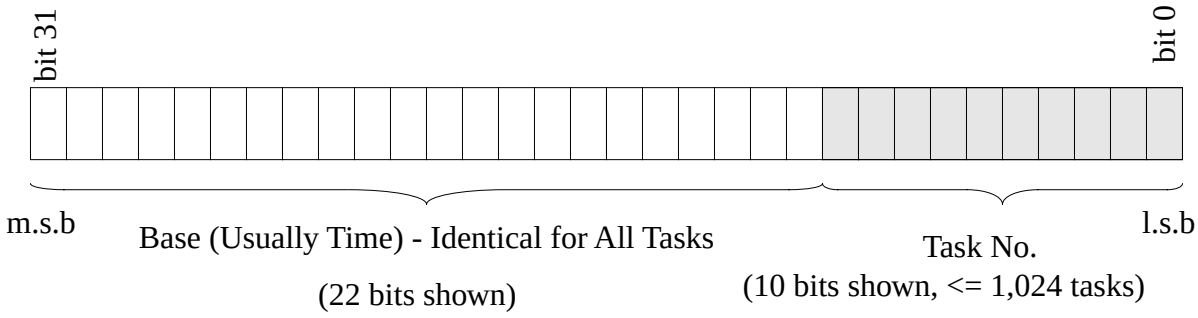



Figure 2.14: Bit Structure of 32 Bit Initial Seed for Parallel Processes

### 2.11.3 Effect of Different Random Sequences on Results

Every different initial seed creates a different sequence of random numbers, so different answers are obtained. However, if "enough" photons are emitted to achieve convergence, then the answers, whatever the initial seed, will be within statistical convergence error. It should be noted that it is not possible to traverse the same sequence of random numbers restarting from a completed run involving emission from more than one surface. Thus, if a run is done with 20,000 photons emitted initially from each surface, the answers will be different than if an initial run is done with 10,000 photons per surface followed by a restart with 10,000 additional photons emitted. However, the comments above pertaining to convergence do apply. If "enough" photons are emitted, then the answers will converge (within a statistical tolerance) to a final state independent of the order of emissions.

# CHAPTER 3 INPUT DECK

The following pages contain the instructions necessary to enable the user to construct the input file (data deck) required by MONT3D. Input lines, generally referred to as "cards," are limited to 80 columns in width. Cards with the character "&" in column 1 can be placed anywhere in the data deck for use as comment cards or as spaces; all such cards are ignored. A sample input file is shown in Section 4.1.

Some care may be required when entering variables in the input deck. How the code handles blanks (spaces) in the input file depends on the compiler used. Some compilers may make the code read blanks as zeroes. When blanks are read as zeroes, blanks to the right side of the variable become significant. For example, if 6 is entered with one space after it, it may be interpreted as 60. When blanks are read as zeroes, variables should be entered right-justified. For most compilers, blanks in the input file are ignored and each variable may be entered anywhere within the columns specified for it in the description below. If all the columns are empty, the variable is read in as zero. It should be noted that for maximum compatibility, input files should be right-justified. The example input file in Section 4.1 is right-justified.

The "Format" column in the card description below gives the FORTRAN format used to read the variable. The only information in that column that may be beneficial to the casual user is that the letter in the format specifies the type of the variable: A for character, E for floating point and I for integer. It should be noted that floating point numbers may be entered in many formats including: as integers (1), in decimal format (1. or .010 or 0.001) or in scientific notation (3.07e-3 or 0.307E-2).

Default values (marked as DEFAULT:) are used if the input file contains zeroes or blanks (read as zero by FORTRAN). If no default value is given, then zero is the default value.

## 3.1 Title Card

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-48 | 6A8 | Heading to appear on output | |

## 3.2 Control Cards

### 3.2.1 Card 1

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-5 | I5 | Number of dimensions (NDIM) | 1 |
| 6-10 | I5 | Number of nodal points (NUMNP) | |
| 11-15 | I5 | Number of surfaces (NSURF) | |
| 16-20 | I5 | Number of materials (NUMMAT) | 2 |
| 21-25 | I5 | Number of wavelength bands (NBANDS) | 3 |
| 26-30 | I5 | Number of photons emitted per band per subsurface (NPHTON) | 4 |

| | | | |
|---|---|---|---|
| 31-35 | I5 | Maximum number of reflections allowed per photon before a warning is issued (NREFS)<br>(DEFAULT: NREFS = 100) | 5 |
| 36-40 | I5 | Maximum number of warnings before the run is aborted (NWARNS)<br>(DEFAULT: NWARNS = 50) | 5 |
| 41-45 | I5 | Maximum number of lost photons (NLOST) | 6 |
| 46-50 | I5 | Surface (emitter) increment for writing restart information (NEBLOCK)<br>NEBLOCK < 0: Restart run from previously stored state or crash<br>NEBLOCK > 0: New run<br>Restart information written to disk after every \|NEBLOCK\| surfaces<br>(DEFAULT: NEBLOCK = 10) | 7 |
| 51-60 | I10 | Initial seed for the random number generator (INSEED)<br>INSEED < 0: The internal, constant value for the initial seed is used.<br>INSEED = 0: The initial seed is obtained from the time<br>INSEED > 0: The value INSEED is used for the initial seed | 8 |

**Notes:**

1. NDIM should be equal to 3 for MONT3D. Any other value causes the code to issue a warning and continue.
2. If NUMMAT is negative, the new material model is used. Otherwise, the old material model is used. See Sections 2.5 and 3.8-3.10 and Appendix A for more details.
3. Radiative properties are defined in wavelength bands, and are constant within a given wavelength band.
4. The number of photons emitted per surface in each band is equal to NPHTN * NDIVX * NDIVY, where NPHTN is the number of photons emitted from each emission point. NPHTON is the default value of NPHTN for all surfaces, which may be overridden for any particular surface(s) during the surface input. See Sections 2.2, 2.7, 3.2.2, and 3.6 for more detail.
5. The maximum number of reflections allowed before a run is aborted is NREFS * NWARNS. For the parallel version of the code, each worker process aborts when it reaches this number. If a worker process aborts due to this error, the other processes continue to work and the aborting process's block is done by another worker later in the run. See Sections 2.10 and 5.6 for more detail.
6. Occasionally, due to precision problems, a photon is "lost" (i.e., no receiving surface is found for a given photon). NLOST specifies the number of such occurrences before the run is aborted. For the parallel version of the code, each worker process aborts when it loses this number of photons. As in Note 5 above, the other worker processes continue and the aborted block is finished by another worker.

7. Each MONT3D run involves emitting from NBANDS*NSURF emitters or surfaces. After emission and tracing from every |NEBLOCK| emitters, the exchange matrix and the restart files are written to disk. Thus, it is possible to restart only from every |NEBLOCK| states. Selecting a small value of |NEBLOCK| will ensure that restart information is written frequently to the output file. Still, if NEBLOCK is set too low, the file writing overhead may become significant. If NEBLOCK is negative, the previous state is read from the restart and exchange matrix files. Restart may be of two types: (1) where the previous state is incomplete (i.e., the previous run was interrupted), or (2) the previous state is not converged. More detail is given in Section 2.9 and Chapter 4. NEBLOCK in versions of MONT3D prior to version 2.4 was known as NINCR.
8. For more details on the initial seed and the RNG in general, see Sections 2.11, 5.5.2, and 5.6.5.

### 3.2.2 Card 2

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-5 | I5 | Type of geometry (IGEOM) | 1 |
| 6-10 | 5I1 | Output print control code toggles, IPRINT(I): | 2 |
| 6 | I1 | IPRINT(1) = 1 - exchange fractions are written to output file | 3 |
| 7 | I1 | IPRINT(2) = 1 - lost photons are written to console and output file | 4 |
| 8 | I1 | IPRINT(3) - currently unused, reserved for future use | |
| 9 | I1 | IPRINT(4) = 1 - complete material property information is written to output file | 5 |
| 10 | I1 | IPRINT(5) = 1 - surfaces wholly or partially in each grid cell are written to output file | |
| 11-15 | I5 | Number of concats (NUMCAT) | 6 |
| 16-20 | I5 | Data check code (IDATA):<br>IDATA = 0: Normal execution<br>IDATA = 1: Data check only, execution stops after input phase | |
| 21-25 | I5 | Number of x' emission points per surface (NDIVX)<br>(DEFAULT: NDIVX = 5) | 7 |
| 26-30 | I5 | Number of y' emission points per subsurface (NDIVY)<br>(DEFAULT: NDIVY = 5) | 7 |
| 31-35 | I5 | Shading in geometry (NSHADE):<br>NSHADE ≥ 0: Shading, distance algorithm<br>NSHADE < 0: Shading, grid shading algorithm | 8 |
| 36-40 | I5 | Trajectory control code (ITRACES):<br>ITRACES > 0: Trajectory information written to disk file (*.trc* extension) | 9 |

| | | | |
|---|---|---|---|
| | | ITRACES = 0: Trajectory information not written to disk file | |
| 41-45 | I5 | Number of worker processes (NWPROC) | 10 |
| 46-50 | I5 | Maximum number of photon convergence loops (NPLOOPS) | 11, 12 |
| | | (DEFAULT: NPLOOPS = 1) | |
| 51-55 | 5I1 | Parallel option code (IPAROPT(I)) as follows: | 13 |
| 51 | I1 | IPAROPT(1) - worker spawning options | 14 |
| | | IPAROPT(1) = 0, workers can spawn on any CPU | |
| | | IPAROPT(1) = 1, workers can not spawn on master CPU | |
| | | IPAROPT(1) = 2, xterms with debug sessions are created for each worker | |
| | | IPAROPT(1) = 3, IPAROPT(1) = 1 and 2 combined | |
| | | Other array elements are reserved for future use | |
| 56-60 | I5 | Number of material property curves (NMACV) | 15 |
| 61-65 | I5 | Number of semi-specular offset angle curves (NDTCV) | 15 |

**Notes:**

1. This option is not available in this version. This space is being retained in the input file for backward compatibility in format. Any value entered will be ignored.
2. If IPRNT(I) = 1, the specified information is written. Otherwise, if IPRNT(I) = 0, the information is not written.
3. If IPRNT(1) = 1, then exchange fractions are printed in the output file. If IPRNT(1) = 0, they are written only to the exchange matrix file.
4. Lost photon information is always written to the lost photon file (file suffix .lst) where it can be viewed using the MPLOT program (see Section 2.8 for more detail). If IPRNT(2) = 1, information about the lost photons is also written to the console and the output file.
5. When IPRNT(4) is equal to 1, all material information, the material property curves; the semi-specular preferred angle curves; and the thetar arrays, are printed to the output file in the form in which they are stored internally. Angles are always displayed in degrees, even if they are stored internally as radians. Material property curves are stored so that each property includes the cumulative sum of all properties below it. The order in which the properties are stored is $\alpha$ (=$\varepsilon$), $\rho_d$, $\rho_s$, $\rho_{ss}$, $\tau_d$, $\tau_s$, and $\tau_{ss}$. The cumulative sum of all the material properties, $\tau_{ss}$ ($\tau_s$ for the old material model) is equal to 1 at all angles. It should also be noted that while the user enters offset angles, $\Delta\theta_o(\theta_i)$ for the semi-specular reflectance and transmittance, the code stores these as the preferred outgoing angle, $\theta_p = \theta_i + \Delta\theta_o(\theta_i)$. The thetar arrays are used to determine the outgoing cone angle of emission from a uniformly distributed random number between 0 and 100. For a detailed discussion of the emission algorithm, see Burns and Pryor [1989].
6. This option is not available in this version. This space is being retained in the input file for backward compatibility in format. If present, NUMCAT must be 0 or MONT3D will terminate with an error.

7. The number of photons emitted per loop for each surface in each band is NPHTN*NDIVX*NDIVY. See Sections 2.2, 2.7, 3.2.1, and 3.6 for more detail.
8. Currently, only the grid shading algorithm is used in this code. If NSHADE is set to zero or greater, a single grid cell is used, which is equivalent to the distance algorithm for shading. More details are given in Sections 2.6, 3.3, and 3.11.
9. The program outputs trajectory information to the *.trc* file if this option is set. This file is used to "view" the trajectories using the MPLOT program [Nagesh and Burns, 1994], useful in establishing physical intuition. If chosen, a copious amount of information is printed; the user is therefore cautioned to select this option only for very few photon emissions per surface. See Section 2.8 for more detail.
10. This option is used only by the parallel version of the code; it is ignored in the single processor version of the code. If NWPROC is less than 1 for the parallel code, a single processor run is done. More detail about the parallel code is given in Sections 2.10 and 5.6.
11. NPLOOPS is the maximum number of full surface photon convergence loops done per surface. In each full surface emission, NPHTN photons are emitted from each subsurface; see Sections 2.2 and 2.7 and the description of NDIVX and NDIVY above. If convergence to the specified tolerance for the surface (see Sections 2.7, 3.4, and 3.6) is not attained within NPLOOPS full surface emissions, then a warning is printed to the screen, and execution continues with the next surface.
12. If 0 is entered for NPLOOPS, an extremely large default convergence tolerance is established ($= 1 \times 10^{10}$), thereby ensuring that convergence occurs after one full surface emission loop. Surfaces that have their convergence tolerance set to this large value will not emit any photons in any restart runs since they are converged.
13. The IPAROPT array is used only in the parallel version of the code. Currently only IPAROPT(1) is used; the other array elements are reserved for future enhancements of the code.
14. IPARAOPT(1) controls how workers are spawned. If the value entered for this option is less than 0 or greater than 3, MONT3D will terminate with an error. For more details, see Section 5.6.
15. NMACV and NDTCV are only used by the new material model (NUMMAT < 0). They specify how many material property and specular offset angle (del theta) curves are to be read. See Sections 2.5, 3.2.1, and 3.8-3.10 for more details.

### 3.2.3 Card 3

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-10 | E10.0 | Scale for X (XSCALE) | 1 |
| | | (DEFAULT: XSCALE = 1.0) | |
| 11-20 | E10.0 | Shift for X (XSHIFT) | 1 |
| 21-30 | E10.0 | Scale for Y (YSCALE) | 1 |
| | | (DEFAULT: YSCALE = 1.0) | |
| 31-40 | E10.0 | Shift for Y (YSHIFT) | 1 |

| | | | |
|---|---|---|---|
| 41-50 | E10.0 | Scale for Z (ZSCALE)<br>(DEFAULT: ZSCALE = 1.0) | 1 |
| 51-60 | E10.0 | Shift for Z (ZSHIFT) | 1 |
| 61-70 | E10.0 | Increment of cone angle in degrees for integration (DELT)<br>(DEFAULT: DELT = 0.01 degrees) | 2 |
| 71-80 | E10.0 | Tolerance for splitting non-planar surfaces (SPLITOL)<br>(DEFAULT: SPLITOL = 0.0001 degree) | 3, 4 |

**Notes:**

1. These factors can be used when one needs to scale from one unit to another (i.e. meters to centimeters) or when one wishes to shift the coordinate system. Since there is no inherent length scale, scaling all axes equally has no effect; nor does shifting of any axis(es).

   The results of scaling and shifting are:

   X = X * XSCALE + XSHIFT,

   Y = Y * YSCALE + YSHIFT, and

   Z = Z * ZSCALE + ZSHIFT

2. DELT = $\Delta\theta$: the increment used in numerically integrating the cumulative distribution function for emission versus cone angle, used to determine the cone angle distribution of emitted photons. The range is: 1E-7 $\leq$ DELT $\leq$ 0.1. If a value outside this range is entered for DELT, then DELT is set to the default value of 0.01. It should be noted that very small values of DELT result in the consumption of excessive computer time. For a detailed discussion of the emission algorithm, see Burns and Pryor [1989].
3. The range of acceptable SPLITOL values is: 1E-20 $\leq$ SPLITOL $\leq$ 0.01. If a positive value outside this range is entered for SPLITOL, SPLITOL is set to the default value of 0.0001. For more information about SPLITOL and split surfaces, see Section 2.3.
4. It is possible to force the code to split all quadrilateral surfaces into triangles. To turn this feature on, enter a negative value of SPLITOL such that -0.01 $\leq$ SPLITOL < 0.

## 3.3 Grid Dimensions (Shading)

Condition(s): NSHADE < 0 (entered in Section 3.2.2), otherwise, omit card

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-5 | I5 | Number of grid cells in the X-direction (NGX) | 1 |
| 6-10 | I5 | Number of grid cells in the Y-direction (NGY) | 1 |
| 11-15 | I5 | Number of grid cells in the Z-direction (NGZ) | 1 |

**Notes:**

1. Cell divisions along each axis are uniform unless a negative number is entered for the NG variable. In that case, the user must enter the grid coordinates for that axis as specified in Section 3.11. It should be noted that in versions of MONT3D prior to version 2.4, all axes must be user defined if NGX is negative; otherwise all axes are uniformly divided. Input files designed for versions of MONT3D prior to 2.4 may not be forwards compatible here.

## 3.4 Default Convergence Tolerance

Condition(s): NPLOOPS > 0 (entered in Section 3.2.2), otherwise, omit card

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-10 | E10.0 | Default convergence tolerance for photon emissions (ERRDEF) | 1 |

**Notes:**

1. This is the default tolerance for convergence of the surface exchange fractions; see Section 2.7 for more detail. This may be overridden for (a) particular surface(s) during surface input; see Section 3.6. If 0 is entered for ERRDEF, then any surface using the default tolerance will complete the full NPLOOPS surface emission loops. If NPLOOPS is equal to 0, ERRDEF is set to $1 \times 10^{10}$; see Section 3.2.2 for details.

## 3.5 Nodal Point Data

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-5 | I5 | Node point number (N) | 1 |
| 6-10 | I5 | Increment in number of points to be generated (INC) | 2 |
| 11-30 | E20.0 | X-coordinate: X(N). | |
| 31-50 | E20.0 | Y-coordinate: Y(N). | |
| 51-70 | E20.0 | Z-coordinate: Z(N). | |

**Notes:**

1. Node points can be input in any order. All nodes from 1 to NUMNP (entered in Section 3.2.1) inclusive must be input or generated as described in the next note. More detail on nodes is given in Section 2.1.
2. Nodal points are generated in increments of INC from *the previous node input to the current node*. The coordinates are obtained by linearly interpolating all coordinates between the ones input on the previous card and the present ones. Care must be taken such that there are an integer number of generated nodes between the present node number and the one input on the previous card. It should be noted that, if the program generates nodes, they should not be input elsewhere.

## 3.6 Surface Data

| Cols. | Format | Entry | Notes |
|---|---|---|---|
| 1-5 | I5 | Surface number (N) | 1 |

| | | | |
|---|---|---|---|
| 6-10 | I5 | Node $N_1$: NODES (1,N) | 2 |
| 11-15 | I5 | Node $N_2$: NODES (2,N) | |
| 16-20 | I5 | Node $N_3$: NODES (3,N) | |
| 21-25 | I5 | Node $N_4$: NODES (4,N) | |
| 26-30 | 5X | Skip | |
| 31-35 | I5 | Number of surfaces to be generated after current surface: NMISS | 3 |
| 36-40 | I5 | Increment of generation: INC | 3 |
| 41-45 | 5X | Skip | |
| 46-50 | I5 | Surface material number: MATNUM(N) | |
| 51-60 | 10X | Skip | |
| 61-65 | I5 | Number of photons (NPHTN) | 4 |
| 66-70 | I5 | Photon increment (INCP) | 5 |
| 71-80 | E10.0 | Convergence tolerance for surface: ERRMAX(N) | 6 |
| | | (DEFAULT for surface: ERRMAX(N) = ERRDEF) | 7 |

**Notes:**

1. Surfaces can be input in any order. All surfaces from 1 to NSURF (entered in Section 3.2.1) inclusive must be input or generated as described in the notes below.
2. The outward normal must be such that the right-hand rule as explained in Section 2.2 applies. Unpredictable and erroneous errors may result if this convention is not adhered to for all surfaces.
3. NMISS additional surfaces are generated by successively incrementing surface numbers by 1 and all 4 node numbers by INC.
4. NPHTN different from 0 overrides NPHTON (entered in Section 3.2.1), the number of photons per subsurface division. If NPHTN is negative, no photons will be emitted from the surface. For more details, see Sections 2.2 and 2.7.
5. Similar to INC in the node input above. For each missing surface, i, (i = 1 to NMISS) generated, i*INCP additional photons emissions are added to NPHTN.
6. ERRMAX(N) is used as explained in Section 2.7. Loops over full surface emissions are done until either the specified number of loops, NPLOOPS (entered in Section 3.2.2), have occurred, or convergence to within tolerance, $C_i$, defined in Equation (2.6), is achieved, whichever comes first.
7. If no value of ERRMAX(N) is input or a value less than or equal to 0 is read, then this defaults to the global value, ERRDEF, input as described in Section 3.4.

## 3.7 Wavelength Band Data

**CARDS 1 to (NBANDS-1)/8**

Condition(s): NBANDS > 1 (entered in Section 3.2.1), otherwise, omit card(s)

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-80 | 8E10.0 | Wavelength breakpoint N (micrometers) | 1 |

**Notes:**

1. The first (1) and last (NBANDS + 1) wave breakpoints are assumed to be 0.0 and $\infty$ ($1 \times 10^{10}$) in micrometers and should not be input. The remaining breakpoints between 0 and $1 \times 10^{10}$ should be input in order from 2 to NBANDS. No more than eight breakpoints should be input per card (10 columns per value). More than one card may be required.

## 3.8 Material Type Data

Material type data cards are input by band for each material in order. For example, if three materials are used in two bands, then all material type data are read in for material 1, band 1; then material 1, band 2; then material 2, band 1; etc. MN and IB are used as checks to ensure that the values are entered in the right order. If the information is not entered in the correct order, the program detects an error and terminates. All material type data cards must be input before any material property curves are input. Sections 3.8-3.10 describe the new material model, which is used if NUMMAT (entered in Section 3.2.1) is less than 0. Otherwise, the old material model is used, with input as described in Appendix A. More detail on the material model is given in Section 2.5. A sample input file using the new material model is given in Section 4.1.

### 3.8.1 Card 1

Condition(s): Repeat for every material

| Cols. | Format | Item | Note(s) |
|---|---|---|---|
| 1-5 | I5 | Material number (MN) | 1 |
| 6-37 | A20 | Material name | |

**Note:**

1. The material number must be between 1 and |NUMMAT|, inclusive.

### 3.8.2 Card 2

Condition(s): Repeat the pair of cards 2 and 3 for every band for each material; unless IB = 0, in which case the information is only entered once. See Note 1 below.

| Cols. | Format | Item | Note(s) |
|---|---|---|---|
| 1-5 | I5 | Band number (IB) | 1 |
| 6-10 | I5 | Emission type (IETP) | 2, 3, 4 |
| | | IETP = 0 - "standard" emission according to $\varepsilon(\theta)$ | |
| | | IETP = 1 - "beam" emission | |
| | | IETP = 2 - "function" emission | 5 |
| 11-20 | E10.0 | Global X-component of beam emission vector (EBX) | 6, 7 |
| 21-30 | E10.0 | Global Y-component of beam emission vector (EBY) | 6, 7 |
| 31-40 | E10.0 | Global Z-component of beam emission vector (EBZ) | 6, 7 |
| 41-50 | E10.0 | Cosine power for weighted diffuse reflectance (RRHOD) | 8 |
| | | (DEFAULT: 1.) | |
| 51-60 | E10.0 | Cosine power for semi-specular reflectance (RRHOSS) | 8 |
| | | (DEFAULT: 1.) | |
| 61-70 | E10.0 | Cosine power for weighted diffuse transmittance (RTAUD) | 8 |
| | | (DEFAULT: 1.) | |
| 71-80 | E10.0 | Cosine power for semi-specular transmittance (RTAUSS) | 8 |
| | | (DEFAULT: 1.) | |

**Notes:**

1. If IB is 0 for the first band of the material, then it is assumed that all material properties for that material are the same for all bands. No cards are read for the other bands.
2. IETP must be between 0 and 2, inclusive. More detail is given in Sections 2.5.3 and 2.5.4.
3. If a material has 0 emittance for all values of $\theta$, then no photons are emitted from surfaces of that type unless IETP = 1 or 2.
4. As explained in Section 2.5.4, if IETP equals 1 or 2, the reciprocity relations, Equations (1.5) and (1.6), may no longer hold.
5. If IETP = 2, emission is accomplished using computer code supplied by the user (function *fcn* in subroutine *getang* in the file *m32s.f*).
6. For regular emission (IETP = 0), these values are ignored. They are read in if IETP = 1 (beam emission) or if IETP = 2 (the function for function emission may require them). Values are normalized internally, to yield a magnitude of 1. If all components entered are equal to 0 and IETP does not equal 0, the program exits with an error.

7. When using beam emission (IETP = 1), care must be taken to ensure that all emission is from the “front” of the surface; i.e. the dot product of the emission vector and the surface normal for each surface that uses this material must be greater than 0. If this is not the case, the code will exit with an error. No check is made when the beam emission variables are used for function emission (IETP = 2).
8. These are the coefficients, $r_d$ and $r_{ss}$, described in Section 2.5.2. If one of these variables equals 0, then it will be set to 1 (standard diffuse distribution). If any of these variables is less than 0, then it is set to 0 (isotropic distribution). Note that, as discussed in Section 2.5.4, if $r_d$ does not equal 1 (standard diffuse distribution) or semi-specular properties are used, the reciprocity relations, Equations (1.5) and (1.6), may no longer hold.

### 3.8.3 Card 3

Condition(s): Repeat the pair of cards 2 and 3 for every band input for every material

| Cols. | Format | Item | Note(s) |
|---|---|---|---|
| 1-10 | E10.0 | Diffuse reflectance (RHOD) | 1, 2 |
| 11-20 | E10.0 | Specular reflectance (RHOS) | 1, 2 |
| 21-30 | E10.0 | Semi-specular reflectance (RHOSS) | 1, 2 |
| 31-40 | E10.0 | Diffuse transmittance (TAUD) | 1, 2 |
| 41-50 | E10.0 | Specular transmittance (TAUS) | 1, 2 |
| 51-60 | E10.0 | Semi-specular transmittance (TAUSS) | 1, 2 |
| 61-70 | E10.0 | Semi-specular offset angle for reflectance (DTHRSS) | 3 |
| 71-80 | E10.0 | Semi-specular offset angle for transmittance (DTHTSS) | 3 |

**Notes:**

1. If a positive, definite value (0 or greater) is input, then this is taken as the constant value for this material property, for all values of $\theta$. If a negative value is input, it is taken as the negative of the curve index for this property and the rounded absolute value must be between 1 and NMACV (entered in Section 3.2.2), inclusive. For example, if the values for this property are included in curve 3, a value that rounds to -3 (preferably -3.0) must be input. Material property curves are read after all material information has been read; see Section 3.9 below. Positive values must be between 0. and 1., inclusive. For more detail, see Section 2.5.1.
2. As explained in Section 2.5.4, if semi-specular properties are used, the reciprocity relations, Equations (1.5) and (1.6), may no longer hold. The reciprocity relations also do not hold if the diffuse reflectance varies as a function of angle. They DO hold if the specular reflectance varies as a function of angle. No tests have been done to see if reciprocity relations hold when the specular or diffuse transmittance vary as a function of angle.
3. The user enters the offset angle which is the $\Delta\theta_o(\theta_i)$ quantity defined in Equation (2.4). Values are in degrees. The code stores the results internally as $\theta_p$. If 0. is entered, $\Delta\theta_o(\theta_i)$ is 0. Other positive values can not be entered because $\Delta\theta_o(\theta_i)$ must be between $-\theta_i$ and $90 - \theta_i$ for all angles,

$\theta_i$. If a negative value is input, it is taken as the negative of the curve index for this offset angle curve and the rounded absolute value must be between 1 and NDTCV (entered in Section 3.2.2), inclusive. For example, if the values for this property are included in curve 3, a value that rounds to -3 (preferably -3.0) must be input. Offset angle (del theta) curves are read after all material property curves have been read; see Section 3.10 below.

## 3.9 Material Property Curves Input

Condition(s): NMACV > 0 (entered in Section 3.2.2), otherwise, omit card(s)

Material property curves must be entered in order from 1 to NMACV using the format given below.

### 3.9.1 Card 1

| Cols | Format | Item | Note(s) |
|---|---|---|---|
| 1-5 | I5 | Material property curve number (NCMA) | |
| 6-10 | I5 | Number of points to be read in for the curve (NP) | 1 |

**Notes:**

1. At least three points must be input. Points must be input at angles of 0 and 90 degrees, plus at least one other intermediate angle.

### 3.9.2 Cards 2 to NP+1

| Cols. | Format | Item | Note(s) |
|---|---|---|---|
| 1-10 | E10.0 | Angle for curve value in degrees (ANGLE) | 1, 2 |
| 11-20 | E10.0 | Material property curve value (VALUE) | 1, 3 |

**Notes:**

1. Values must be given for 0 and 90 degrees.
2. ANGLE values must be between 0. and 90., inclusive.
3. VALUE must be between 0. and 1., inclusive.

## 3.10 Semi-specular Offset Angle Curves Input

Condition(s): NDTCV > 0 (entered in Section 3.2.2), otherwise, omit card(s)

Semi-specular offset angle curves must be entered in order from 1 to NDTCV using the format given below.

### 3.10.1 Card 1

| Cols | Format | Item | Note(s) |
|---|---|---|---|
| 1-5 | I5 | Semi-specular offset angle curve number (NCDT) | |
| 6-10 | I5 | Number of points to be read in for the curve (NP) | 1 |

**Notes:**

1. At least three points must be input. Points must be input at angles of 0 and 90 degrees, plus at least one other intermediate angle.

### 3.10.2 Cards 2 to NP+1

| Cols. | Format | Item | Note(s) |
|---|---|---|---|
| 1-10 | E10.0 | Angle for curve value in degrees (ANGLE) | 1, 2 |
| 11-20 | E10.0 | Semi-specular offset angle value in degrees (VALUE) | 1, 3 |

**Notes:**

1. Values must be given for 0 and 90 degrees.
2. ANGLE values must be between 0. and 90., inclusive.
3. VALUE must be between -ANGLE and (90 - ANGLE), so that the preferred outgoing angle is between 0 and 90 degrees, inclusive.

## 3.11 User Grid Input

It should be noted that depending on the values of NGX, NGY, and NGZ entered, forwards compatibility may not exist for input files create for versions of MONT3D earlier than 2.4. For additional explanation of the variables NSHADE, and NGX, NGY and NGZ, see Sections 3.2.2 and 3.3, respectively.

### 3.11.1 User X-grid Coordinates

Condition(s): NSHADE < 0 and NGX < 0, otherwise, omit card(s)

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-80 | 8E10.0 | X-grid coordinates XG(N) | 1 |

**Notes:**

1. For N = 1 to NGX+1. No more than eight values should be input per card (10 columns per value). More than one card may be required.

### 3.11.2 User Y-grid Coordinates

Condition(s): NSHADE < 0 and NGY < 0, otherwise, omit card(s)

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-80 | 8E10.0 | Y-grid coordinates YG(N) | 1 |

**Notes:**

1. For N = 1 to NGY+1. No more than eight values should be input per card (10 columns per value). More than one card may be required.

### 3.11.3 User Z-grid Coordinates

Condition(s): NSHADE < 0 and NGZ < 0, otherwise, omit card(s)

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-80 | 8E10.0 | Z-grid coordinates ZG(N) | 1 |

**Notes:**

1. For N = 1 to NGZ+1. No more than eight values should be input per card (10 columns per value). More than one card may be required.

# CHAPTER 4 PROGRAM EXECUTION

This chapter illustrates how MONT3D is executed in a Unix environment. The first section details the sample input files used in the examples below. The second section explains the commands and results obtained from a normal execution and a restart execution of the code for this input file. The next two sections show the screen output for the normal and restart executions described in the previous section. The final section addresses the machine independence of MONT3D results and its implications during restart runs.

## 4.1 Input File “box.in”

Figure 4.1 shows a 3-D geometry for analysis by MONT3D. Comparison of the picture with the input file “box.in” shown below illustrates the right-hand rule for 3-D surfaces. The inner surfaces are modeled by two different types of materials which are mixed specular and diffuse reflecting and whose properties vary with incident angle. This file demonstrates the “new” material model. The first material, which is used for the sides of the box, has the diffuse reflectance entered as a constant value (0.2), while the specular reflectance is entered as the curve shown in Figure 4.2. The second material, which is used for the top and bottom of the box, uses the same curve for its diffuse reflectance and has a constant value for the specular reflectance (0.1). Due to the simplicity of the geometry and the small number of surfaces it contains, grid shading is not used. It should be noted that even moderately complex geometries will benefit from grid shading, see Section 2.6 for more details. A complete description of the input format is given in Chapter 3.

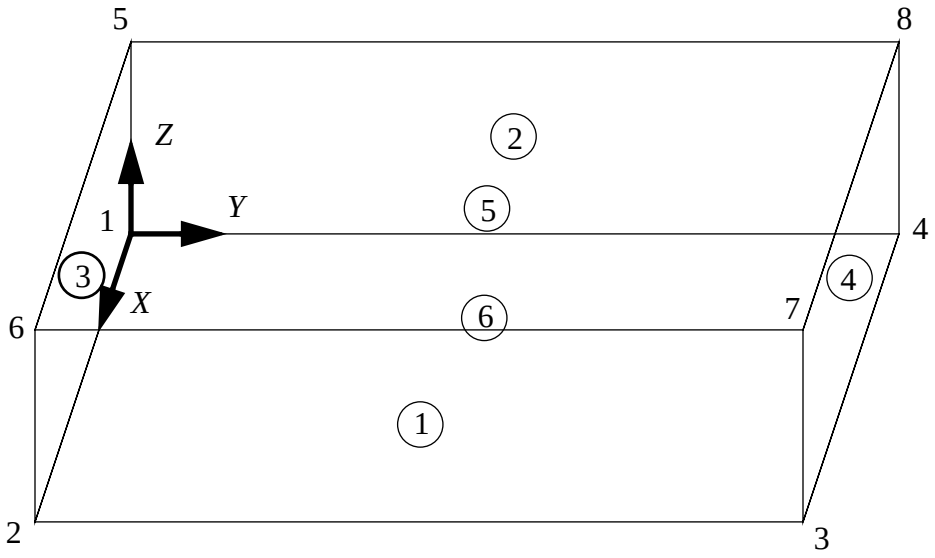



Figure 4.1: 3-D Geometry of File “box.in”

The input file is listed below and is also available at the MONT3D WWW site mentioned in Section 1.1. It should be noted that comment cards are used throughout the input file to identify the input variables and where they should be entered. As shown in Chapter 3, each variable is restricted

to being entered in a certain range of non-overlapping columns on a certain card. The last letter of each variable name on the comment cards marks the last column that can be used to input that variable (except for IGEOM). If the code reads blanks as zeroes (see the beginning of Chapter 3 for more detail), then all variables should be right-justified and should line up as shown in the file below. For maximum compatibility, all input files should be right-justified.

```
&                                          TITLE (48 CHARACTERS MAX)
3-D BOX TEST PROBLEM
&NDIM     NSURF    NBANDS     NREFS     NLOST         INSEED
&    NUMNP    NUMMAT    NPHTON    NWARNS   NEBLOCK
    3    8    6   -2    1  100  100  100   10   -2         0
&IGEOM   NUMCAT     NDIVX    NSHADE    NWPROC   IPAROPT     NDTCV
&    IPRNT     IDATA     NDIVY   ITRACES   NPLOOPS     NMACV
    110000    0    0   10   10    1    0    0   1000000    1    0
&   XSCALE    XSHIFT    YSCALE    YSHIFT    ZSCALE    ZSHIFT      DELT   SPLITOL
       1.0       0.0       1.0       0.0       1.0       0.0      0.01    0.0001
& GRID NUMBERS (READ IF NSHADE < 0)
& NGX  NGY  NGZ
&   5    5    5
&   ERRDEF (if NPLOOPS<>0)
    2.0e-3
& NODES
&   N  INC                  X                  Y                  Z
    1    0                0.0                0.0                0.0
    2    0               20.0                0.0                0.0
    3    0               20.0               30.0                0.0
    4    0                0.0               30.0                0.0
    5    0                0.0                0.0               10.0
    6    0               20.0                0.0               10.0
    7    0               20.0               30.0               10.0
    8    0                0.0               30.0               10.0
& ELEMENTS
&   N   N1   N2   N3   N4      NMISS INC        MN           NPHT INCP    ERRMAX
    1    1    2    3    4         0    0         2              0    0      0.00
    2    5    8    7    6         0    0         2              0    0      0.00
    3    1    5    6    2         0    0         1              0    0      0.00
    4    3    7    8    4         0    0         1              0    0      0.00
    5    1    4    8    5         0    0         1              0    0      0.00
    6    2    6    7    3         0    0         1              0    0      0.00
& WAVELENGTH BAND BREAKPOINTS (READ IF NBANDS > 1)
&      BP1       BP2       BP3       BP4       BP5       BP6       BP7       BP8
&      5.0
& MATERIALS (NEW MODEL, NUMMAT < 0)
&  MN            MATNAME
    1   Sample Material 1
&  IB IETP       EBX       EBY       EBZ     RRHOD    RRHOSS     RTAUD    RTAUSS
    0    0        0.        0.        0.        0.        0.        0.        0.
&     RHOD      RHOS     RHOSS      TAUD      TAUS     TAUSS    DTHRSS    DTHTSS
       0.2       -1.        0.        0.        0.        0.        0.        0.
&  MN            MATNAME
```

```
    2   Sample Material 2
&  IB IETP       EBX       EBY       EBZ      RRHOD    RRHOSS     RTAUD    RTAUSS
    0    0        0.        0.        0.        0.        0.        0.        0.
&     RHOD      RHOS     RHOSS      TAUD      TAUS     TAUSS    DTHRSS    DTHTSS
        -1       0.1        0.        0.        0.        0.        0.        0.
&MATERIAL PROPERTY CURVES (NUMMAT < 0, NMACV > 0)
& NCP   NP
    1    6
&     ANGLE    VALUE
       0.0      0.40
      30.0      0.38
      50.0      0.35
      70.0      0.20
      80.0      0.20
      90.0      0.80
& SEMI-SPECULAR DEL THETA CURVES (IF NDTCV > 0)
& USER DEFINED GRID (READ IF NGX < 0)
& User X grid (Read if NGX < 0)
&      XG1       XG2       XG3       XG4       XG5       XG6       XG7       XG8
&    -0.01      0.01         1         5         9.    10.01
& User Y grid (Read if NGY < 0)
&      YG1       YG2       YG3       YG4       YG5       YG6       YG7       YG8
&    -0.01      0.01         1         3         9     20.01
& User Z grid (Read if NGZ < 0)
&      ZG1       ZG2       ZG3       ZG4       ZG5       ZG6       ZG7       ZG8
&    -0.01      0.01         1      1.01         7     30.01
```

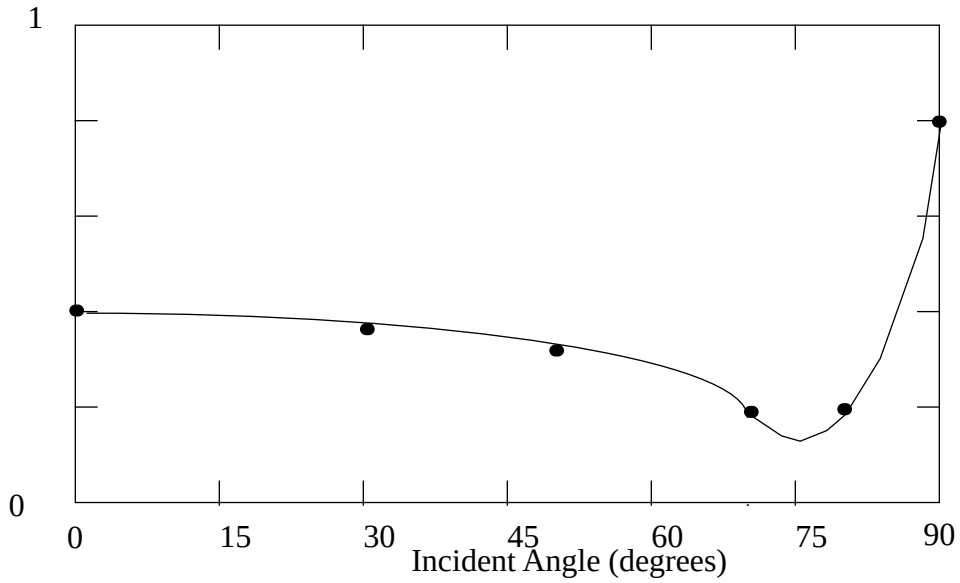



Figure 4.2: Material Property Curve for File ‘‘box.in’’

## 4.2 Execution of File “box.in”

The following details the execution of the 3-D code using the file “box.in.” The output shown below in Section 4.3 is output to the screen. The Unix prompt is “czeeb(n)%”, where “n” is the command number. All input from the user is shown bold and italicized.

The executable file here is *mont3d* (note that Unix is case sensitive), and is invoked by typing its name - shown at the first prompt - “czeeb(1)%.” All subsequent output up to the “czeeb(2)%” prompt is from the program MONT3D. The user types in only the prefix of the input file, which MUST have the extension “in.” In this case, the file is “box.in,” and the user enters “box.” If the program cannot find the input file, an error diagnostic is printed and the program exits. Additional detail on the files used by MONT3D can be found in Chapter 5.

The program proceeds with reading of the input file. The run’s title is printed (here, “3-D BOX TEST PROBLEM”), along with the version number (here, “2.4f1”), the date of last modification (here, “04-15-99”), and the language of the source code (always “f77”). Next, the control portion of the input is read. This establishes the numbers of nodes, surfaces (sometimes referred to as elements or surface elements), wavelength bands, materials, grids, etc. which are to be read by succeeding subroutines. Then, the input subroutines are executed one-by-one. As each subroutine is entered, the name of the subroutine is printed to the output file. This assists in debugging input files. For example, if the program “crashes” after the message “in nodin3” is printed, there is almost certainly a problem with the nodal input (or possibly, the cards are out of order). Another item that should be noted is the “Memory Allocation and Usage” section listed between the subroutines “gridin” and “order.” By setting the memory allocation parameters to the “used” values, it is possible to run the code using the smallest memory allocation possible. More details on changing the code’s parameters are given in Section 5.5. Finally, the successful completion of the entire input phase is indicated by the message, “input phase complete.”

The solution phase occurs next. After the specified number of full surface emission loops for each surface are completed, a message is printed indicating whether the exchange fractions have converged (see Section 2.7). In addition, the wavelength band, surface number, total number of emitted photons (summed over all full surface emissions), and the calculated error are printed. Here, none of the surfaces converge to the stringent error tolerance of 0.002, but all come close.

Next, after all the surfaces emissions have been completed, the code creates the exchange matrix file (file suffix *.nij*) as shown by the two statements that both start with “Now writing...” This phase is usually fast unless there is a large number of surfaces. Next, overall program statistics are printed, including the error statistics; various timings for the run (in seconds of CPU time); total number of photons emitted (and traced to absorption); and the performance metrics, number of photons per CPU second for both the entire run and just for the solution phase. The timing information is divided into three phases. The input and solution phases are self-explanatory. The clean-up phase is the time the program spends after the solution phase creating the exchange matrix file.

Finally, upon completion of the run, the Unix prompt is issued. To find out more about the run at this point, the output file *box.out* is available. Furthermore, the plot file *box.plt* can be used with the MPLOT program [Nagesh and Burns, 1994] to view the geometry and material properties, etc. For other more complicated geometries which have “holes” and other such problems, the lost photon file (file suffix *.lst*), the leaks file (file suffix *.lks*), and the optional trace file (file suffix *.trc*) may

also be used with MPLOT to further analyze the geometry. For more details, see Sections 2.8 and 5.3 and the MPLOT documentation.

Section 4.4 contains the screen output generated by a restart run of box.in. The only modification to file *box.in* was to change NEBLOCK from 2 for the initial run to -2 for the restart run. It is possible to change a number of other variables before beginning a restart run. Basically, any variable that does not change the description of the geometry or material properties, such as NPHTON, can be changed for a restart run. It should be noted that when doing the restart run, the exchange factor file, *box.nij,* and the restart file, *box.rst*, must be present. If the run restarted from a crash, the block file (*box.blk*) and any block nij files (file suffix *.bni*) must also be present. The restart run in Section 4.4 also demonstrates MONT3D's command line interface. The "-f" (family option) specifies the name to be used for all files not specifically named by any other command option. More details about the command line interface are given in Section 5.4. Convergence was achieved in the restart run.

## 4.3 Screen Output During Execution of File "box.in"

```
czeeb(1)% mont3d
  ****** mont3d ******
  Enter prefix for disk files -    20 characters or less

     input file MUST have extension .in
box
 3-D BOX TEST PROBLEM                              v. 2.4f1 04-15-99 f77

 in nodin3
 in surfin3
 in wavin
 in matinnew
   reading material property curves
 in curveset
 in cumdis
 in fileset
 in gridin



       M e m o r y   A l l o c a t i o n   a n d   U s a g e

Category                                        Allocated       Used

 1. Nodes (inod)                                15000 words      8 words (  0%)
 2. Surfaces (isrf)                              5000 words      6 words (  0%)
 3. Materials (imat)                               30 words      2 words (  7%)
 4. Wavelength bands (ibnd)                         5 words      1 words ( 20%)
 5. Restart surface block size (iblk)             200 words      2 words (  1%)
 6. Restart blocks (itblk)                       1000 words      3 words (  0%)
 7. Index digits for filename (iimag)               4 words      1 words ( 25%)
 8. Grid divisions per axis (incg)                 60 words      1 words (  2%)
```

```
 9. Surfaces including split surfaces (isrfs)    7000 words      6 words (  0%)
10. Surface segments in grid cells (iseg)      100000 words      6 words (  0%)


 in order
 in graf
 in bplane2d



        i n p u t   p h a s e   c o m p l e t e




                band  surf.  iter.       npht.        error         tol.
 not converged -     1      1     10      100000   0.2173E-02   0.2000E-02
 not converged -     1      2     10      100000   0.2175E-02   0.2000E-02
 restart file written
 not converged -     1      3     10      100000   0.2125E-02   0.2000E-02
 not converged -     1      4     10      100000   0.2123E-02   0.2000E-02
 restart file written
 not converged -     1      5     10      100000   0.2136E-02   0.2000E-02
 not converged -     1      6     10      100000   0.2136E-02   0.2000E-02
 restart file written

Now writing exchange numbers from each block nij file (.bni)
to a temporary binary file (.tni)

Now writing the exchange numbers from the temporary
binary file (.tni) to the .nij file




     normal termination




 Convergence information for band   1
         0 out of the     6 completed surfaces did not emit
     and     0 out of the    6 emitting  surfaces converged.
for the emitting surfaces:
        Average error = 0.2145E-02
        Minimum error = 0.2123E-02   on surface    4
        Maximum error = 0.2175E-02   on surface    2
```

```
                    s o l u t i o n   t i m e   l o g


     time for input phase                    =  0.15902E+00 secs
     time for solution phase                 =  0.10073E+02 secs
     time for cleaning up                    =  0.13693E-01 secs
     total run time                          =  0.10245E+02 secs
     total number of photons emitted         =  0.60000E+06
     total number of photons lost            =         0
     photons per second (solution phase)     =    59567.793
     photons per second (total run)          =    58563.584

     all results are given in CPU time
czeeb(2)%
```

## 4.4 Screen Output During Restart Execution of File “box.in”

```
czeeb(2)% mont3d -f box
 3-D BOX TEST PROBLEM                              v. 2.4f1 04-15-99 f77

 in nodin3
 in surfin3
 in wavin
 in matinnew
   reading material property curves
 in curveset
 in cumdis
 in fileset
 in gridin



       M e m o r y   A l l o c a t i o n   a n d   U s a g e

Category                                        Allocated       Used

 1. Nodes (inod)                                 15000 words      8 words (  0%)
 2. Surfaces (isrf)                               5000 words      6 words (  0%)
 3. Materials (imat)                                30 words      2 words (  7%)
 4. Wavelength bands (ibnd)                          5 words      1 words ( 20%)
 5. Restart surface block size (iblk)              200 words      2 words (  1%)
 6. Restart blocks (itblk)                        1000 words      3 words (  0%)
 7. Index digits for filename (iimag)                4 words      1 words ( 25%)
 8. Grid divisions per axis (incg)                  60 words      1 words (  2%)
 9. Surfaces including split surfaces (isrfs)     7000 words      6 words (  0%)
10. Surface segments in grid cells (iseg)       100000 words      6 words (  0%)


 in order
```

```
 in graf
 in bplane2d


        i n p u t   p h a s e   c o m p l e t e




                band  surf.  iter.       npht.        error         tol.
    converged -     1      1      2      120000   0.1984E-02   0.2000E-02
    converged -     1      2      2      120000   0.1984E-02   0.2000E-02
 restart file written
    converged -     1      3      2      120000   0.1940E-02   0.2000E-02
    converged -     1      4      2      120000   0.1938E-02   0.2000E-02
 restart file written
    converged -     1      5      2      120000   0.1949E-02   0.2000E-02
    converged -     1      6      2      120000   0.1949E-02   0.2000E-02
 restart file written

Now writing exchange numbers from each block nij file (.bni)
to a temporary binary file (.tni)

Now writing the exchange numbers from the temporary
binary file (.tni) to the .nij file




     normal termination




 Convergence information for band   1
        0 out of the     6 completed surfaces did not emit
    and     6 out of the    6 emitting  surfaces converged.
    for the emitting surfaces:
       Average error = 0.1957E-02
       Minimum error = 0.1938E-02   on surface    4
       Maximum error = 0.1984E-02   on surface    1

                s o l u t i o n   t i m e   l o g


    time for input phase                  =  0.16846E+00 secs
```

```
     time for solution phase                =  0.20285E+01 secs
     time for cleaning up                   =  0.11734E-01 secs
     total run time                         =  0.22087E+01 secs
     total number of photons emitted        =  0.12000E+06
     total number of photons lost           =         0
     photons per second (solution phase)    =    59155.912
     photons per second (total run)         =    54329.894

     all results are given in CPU time
czeeb(3)%
```

## 4.5 Machine Independence of MONT3D

Much effort has been put into making MONT3D platform independent. Of all the files used by MONT3D, only one is binary (the temporary binary exchange matrix file), and it only exists during the input phase at the beginning of the run and clean-up phase at the end of a run. Since all results are stored as ASCII files, runs started on one machine can be restarted on a different machine, provided the files are available on the other machine (usually accomplished via ftp). This is true even if the run is restarting from a crash. Furthermore, the code can switch between single processor and parallel execution, or even change the number of worker processes being used in a parallel run, even if restarting from a crash.

# CHAPTER 5 IMPLEMENTATION DETAILS

This chapter covers how MONT3D is implemented. The first two sections describe the MONT3D source code and explain how to compile it on various platforms. The next two sections cover the files generated and used by MONT3D, and the command line used to name several of these files. The fifth section discusses the use of parameter statements in the code, including how they affect memory allocation. The sixth section discusses the parallel version of the code. Unix batch execution by scripts is the subject of the seventh section. The last section briefly discuss precision in MONT3D.

If any problems with the code are encountered, one of the authors should be contacted.

## 5.1 MONT3D Source Files

In this section, the various files included in the MONT3D source code are described. Much work has been done recently to increase the portability of MONT3D. Furthermore, the ability to compile a parallel version of the code implemented in PVM [Geist et al., 1994] has been added. Both of these features have been implemented through the use of "stubs." Stubs are multiple files which contain the same subroutine and function names, each with or without certain options. Features can be turned on and off by compiling the code using selected stubs.

All nonportable parts of the MONT3D code (the command line, the timing, and the time and date functions) have been separated into stubs. If an architecture does not support a feature, it can be compiled with a stub that omits the feature. Alternatively, it is possible to create a stub that implements that feature specifically for that platform.

The only potentially nonportable aspects of MONT3D are the bitwise operators used in the random number generator. The bitwise operator functions used are those defined in the Fortran 90 standard. While these bitwise operations functions are not part of the ANSI FORTRAN 77 standard, these subroutines have been supported on every platform tested so far, and appear to be included in all current FORTRAN 77 compilers as extensions.

The parallel portion of the code is also implemented as a stub. The only difference between the parallel and single processor versions of the code is which stub is used.

In the subsections below, various files are described, starting with the files common to all versions. Next, the stubs to support the nonportable features: command line, timing, and date and time, are covered. Finally, the parallel stubs are discussed. Further detail is also be given in the next section which covers compiling MONT3D for various platforms.

### 5.1.1 Files Common to All Versions

Every compilation of MONT3D requires the following files:

**m30s.f:** the main program, routines called by subroutines in multiple source code files, and other miscellaneous routines.

**m311s.f:** the main input routine and all the subroutines required to input nodes, surfaces, and wavelength bands.

**m312s.f:** the input routines for entering material properties and material property curves.

**m313s.f:** the input routines for entering and setting up the grid and post-processing of the input data.

**m32s.f:** the solve routines.

**mont3d.par:** parameters that control the sizes of arrays and other quantities (see Section 5.5 below).

**mont3d.com:** all common variables not specific to the parallel version of the code.

### 5.1.2 Command Line

The command line allows the user individually to specify file names for most of the files generated by MONT3D. If the command line is disabled, all command line input is ignored and the user must specify the base file name by entering it when prompted during program execution. Additional detail is given in Section 5.4.

Three stubs are available for the command line:

**m3comline.f:** activates the command line. Requires the nonstandard subroutines *iargc* and *getarg* to be supported by the compiler. Tested on Sun, SGI, DEC Alpha, Hewlett Packard, IBM RS /6000, and Macintosh (Absoft FORTRAN) platforms.

**m3nocom.f:** deactivates the command line for platforms, such as the NAG f90 compiler under Linux, which do not support the subroutines *iargc* and *getarg*.

**m3pccom.f:** activates the command line when compiling the code under Microsoft Windows (Microsoft FORTRAN).

### 5.1.3 Timing Information

As shown in Chapter 4, MONT3D prints some timing information at the end of each run. There are two types of timing information that can be collected. For single processor runs, the timing information desired is usually the CPU time. The CPU time includes both the user time, which is the time the program itself uses in execution, and the system time, which is the time the system uses for activities such as I/O and swapping the program in and out of memory. The CPU time does not include the time the processor spends on other programs or processes. For parallel runs, the most useful timing information is clock time, which is the actual time recorded from the clock. The clock time includes the time spent on other processes besides MONT3D.

If timing is disabled, then MONT3D reports no timing information. On Unix systems, the clock, user, and system times for the run can be obtained using the Unix *time* command; perform a *man* on *time* for more information.

All these stubs work to some degree with the Microsoft Windows (Microsoft FORTRAN) version of the code. The suggest stub for compiling the Windows version of the code is *m3f90timing.f*. For more details on Windows timing, see Section 5.2.2.

Four stubs are available for timing:

**m3ctiming.f:** allows the code to collect timing information about the run in clock time. Requires that the nonstandard subroutine *time* be supported. Mainly used for the parallel version of the code. Tested on Sun, DEC Alpha, IBM RS/6000, and Windows (Microsoft FORTRAN) platforms.

**m3etiming.f:** allows the code to collect timing information about the run in CPU time. Requires that the nonstandard subroutine *etime* be supported. Mainly used for the single processor version of the code. Tested on Sun, SGI, DEC Alpha, Hewlett Packard, and Macintosh (Absoft FORTRAN) platforms. More detail on the use of this stub under Microsoft Windows (Microsoft FORTRAN) is given in Section 5.2.2.

**m3f90timing.f:** allows the code to collect clock time using the Fortran 90 subroutine. It is used by the parallel and single processor versions of the code on computers such as the IBM RS/6000 which do not support the subroutine *etime*. It is also the preferred timing stub to use for Microsoft Windows (Microsoft FORTRAN).

**m3notiming.f:** deactivates timing for platforms which do not support any of the timing routines listed above.

### 5.1.4 Time and Date Information

Time and date information is rather important because it is used to create time-generated seeds for the random number generator; see Sections 2.11 and 3.2.1 for additional detail. If time and date information are disabled, then the time-generated seed is an internal constant. To obtain different random number sequences, the user must supply different initial seeds in the input file.

Three stubs are available for time and date information:

**m3f0date.f:** allows the code to access the time and date of the run. Requires that the Fortran 90 subroutine *date_and_time* be supported by the compiler. Tested on the IBM RS/6000 and Microsoft Windows (Microsoft FORTRAN) platforms.

**m3fdate.f:** allows the code to access the time and date of the run. Requires that the nonstandard subroutine *fdate* be supported by the compiler. Tested on Sun, SGI, DEC Alpha, Hewlett Packard, and Macintosh (Absoft FORTRAN) platforms.

**m3nodate.f:** deactivates time and date information for platforms which do not support any date subroutine. The date and time are set to the default value: 16:53:49 on March 12, 1970.

### 5.1.5 Parallel Implementation

Two stubs that control parallel implementation of MONT3D are available:

**m3nopara.f:** implements the single processor version of the code. This has compiled on every platform on which it has been tested.

**m3pvm.f:** implements a PVM version of MONT3D. The file *m3pvm.inc* contains the PVM specific parameters and common variables and is required. The user's environment variables PVM_ROOT and PVM_ARCH must be defined for this file to compile successfully. The PVM version has been compiled on Sun, Hewlett Packard, IBM RS /6000, DEC Alpha and Linux (NAG f90 compiler) platforms.

More information about compiling is given in the below. Also, more information on running the parallel code is given in Section 5.6 below.

## 5.2 Compiling MONT3D

### 5.2.1 Unix

For Unix, a Makefile that has been tested on several platforms is available to compile the code. Different instances of the program can be created by typing "make *target*" in the directory that contains the source code files and the Makefile, where *target* is a keyword that determines which version of MONT3D is created. Currently, there are nine options for *target:*

**m3d:** creates the single processor version of MONT3D with all optional features (command line, timing, and time and date) enabled using *m3comline.f, m3etiming.f,* and *m3fdate.f.* The timing results generated are CPU time. This has been compiled on Sun, SGI, DEC Alpha and Hewlett Packard architectures.

**m3dibm:** creates the single processor version of MONT3D and activates the command line and the Fortran 90 versions of the clock timing and time and date functions using the files *m3comline.f, m3f90timing.f,* and *m3f90date.f.* The timing results generated are clock time. This version compiles on the IBM RS/6000 platform.

**m3df90:** creates the single processor version of MONT3D and activates the Fortran 90 versions of the clock timing and time and date functions but not the command line. It uses the files *m3nocom.f, m3f90timing.f,* and *m3f90date.f.* The timing results generated are clock time. This version can be compiled on any full featured Fortran 90 compiler.

**m3dansi:** creates the single processor version of MONT3D with all optional features listed above turned off. The files used are *m3nocom.f, m3notiming.f,* and *m3nodate.f.* We believe it to be mostly ANSI compliant except for bitwise operations required for the random number generator as mentioned above.

**m3dpvm:** creates the parallel PVM implementation of MONT3D with all features activated using *m3comline.f, m3ctiming.f,* and *m3fdate.f.*The timing results generated are clock time. It has been compiled on Sun, DEC Alpha, and Hewlett Packard workstations.

**m3dpvmibm:** creates the parallel PVM implementation of MONT3D and activates the command line and the Fortran 90 versions of the clock timing and time and date functions using the files *m3comline.f, m3f90timing.f,* and *m3f90date.f.* The timing results generated are clock time. This version compiles on the IBM RS/6000 platform.

**m3dpvmf90:** creates the parallel PVM implementation of MONT3D activates the Fortran 90 versions of the clock timing and time and date functions but not the command line. It uses using the files *m3nocom.f, m3f90timing.f,* and *m3f90date.f.* The timing results generated are clock time. This version can be compiled on any full featured Fortran 90 compiler.

**m3dpvmansi:** creates the parallel PVM implementation of MONT3D with all optional features deactivated. It uses the files *m3nocom.f, m3notiming.f,* and *m3nodate.f.*

**clean:** deletes all the object files in the current directory. This option should be used whenever the user wants to start the compilation over from scratch. If code does not seem to compile correctly, then the user should try using the clean target and then recompiling the code.

The name of the MONT3D executable generated by the Makefile is specified by macros in the Makefile. The name of the single processor version of the code is determined by the Makefile

macro M3DSERNAME which has a default value of *mont3d*. The name of the PVM version is set by the macro M3DPARANAME which has a default value of *m3dpvm*. For the worker processes to spawn correctly, the name of the PVM version of executable must match the parameter *m3dparaname* which is declared in the file *mont3d.par*. The default value for *m3dparaname* is *m3dpvm*. More detail is given in Sections 5.5.2, 5.6.1, and 5.6.4.

When compiling the parallel versions of the code, the user's environment variables PVM_ROOT and PVM_ARCH must be defined for this compilation to succeed.

When compiling the PVM version of the code on various heterogeneous architectures, different executables can be used on different machines as long as they are the parallel versions of the executables. For example in the inhomogeneous network of the Suns and Linux workstation mentioned earlier, the *m3dpvm* target was compiled on the Suns, while the *m3dpvmansi* target was compiled under Linux. Currently, the command line, timing, and time and date functions are used only by the master in the parallel code. Therefore, the command line and time and date functions are supported if the master process's executable supports them.

To compile MONT3D on various architectures, specifying the correct target may not be enough. There are three macro definitions in the Makefile that may have to be set as well. The Makefile lists default values for these macros for Sun, SGI, IBM RS /6000, DEC Alpha, and Hewlett Packard platforms and for the NAG f90 compiler under Linux. These macros are:

**FC:** specifies the FORTRAN compiler. This is usually set to "f77." For the Hewlett Packard platform, this must be set to "fort77." Of course, if a Fortran 90 compiler is used, this should be set to "f90."

**FFLAGS:** specifies the FORTRAN compiler flags. Some platforms require special flags. Flags are specified in the Makefile to allow the user to compile the code on several platforms and control optimization. If a person is knowledgeable about optimization options for his specific computers, he may want to change the default settings for these flags. FFLAGS can be used to compile the code with almost any options desired - check the compiler's documentation for specifics. Popular choices are optimizing (-O option), *debugging* (-g option), *array bounds checking* (-C option), and *profiling* (-p or -pg option).

**LPVMFLAGS:** specifies the libraries for the PVM code which are used during linking. The directory to search for the PVM libraries and the libraries *pvm3* and *fpvm3* must always be specified. Some platforms may require other libraries. For example, the Sun version also requires the libraries *nsl* and *socket*. Appropriate values of LPVMFLAGS are listed in the Makefile for the Sun, Hewlett Packard, IBM RS/6000 and DEC platforms, and for the NAG f90 compiler under Linux. Values for other platforms can be found by compiling the sample programs that are included with the PVM distribution.

The authors have worked hard to make MONT3D as portable as possible. They would like to be informed about troubles/successes when compiling the code on other platforms.

### 5.2.2 Microsoft Windows

The code has been compiled under Microsoft Windows using Microsoft FORTRAN Powerstation.The files which are needed to compile the code under Windows are: *m30s.f, m311s.f, m312s.f, m313s.f, m32s.f, m3nopara.f, m3pccom.f, m3f90timing.f, m3f90date.f, mont3d.par,* and *mont3d.com*. These files have compiled successfully on every Windows machine we have tested so

far. However, it is possible to use other timing stubs. The code is essentially the same if either *m3f90timing.f* or *m3ctiming.f* is used. Both stubs cause timing results to be given in clock time. While *m3etiming.f* compiles in all test cases, sometimes the timing function does not work and zero seconds is reported for all the timings in the timing summary. In our limited testing, the *etime* function works when the code is compiled and run under Windows NT, but not under Windows 95 or 98. It is highly likely that *etime* reports clock timing instead of CPU timing on most Windows machines, so there is probably no advantage to using etime instead of one of the clock time functions.

### 5.2.3 Macintosh

The Macintosh version has been compiled on a 233 MHz 604e PowerPC chip using the Absoft f77 (and f90) compilers. A full-featured version of the code can be obtained by compiling the following files: *m30s.f, m311s.f, m312s.f, m313s.f, m32s.f, m3nopara.f, m3comline.f, m3etime.f, m3fdate.f, mont3d.par,* and *mont3d.com*. The compilation also requires the Absoft *unixlib.o* file which is usually in the *AbsoftLibraries* folder in the *Libraries* folder in the *MPW* folder. The code must be compiled with the *fold to uppercase* option (-N109). The *general optimization* (-O) and *604 PowerPC specific optimizations* (-Q92, if applicable) options both improve performance. Best performance is obtained compiling the code as an *MPW Tool* so it can run in the *MPW Shell* program. Performance is further increased by using a minimum of extensions and control panels.

Many timings have been done of the 233 MHz 604e PowerPC Macintosh and performance has been found to be very good. This particular Macintosh is almost 5 times faster than a 36 MHz Sun Sparc-10 workstation.

## 5.3 Files Generated and Used by MONT3D

This section briefly describes the files used by MONT3D. For specificity, a prefix of *fn* is used. Table 5.1 gives the files used (either pre-existing or generated during execution) by MONT3D. Files listed as temporary are deleted by the end of a successful run; most are deleted at the end of the run.

Almost all the files listed in Table 5.1 are ASCII files, and may be read and printed with no difficulty. Files for unit 13 are binary files, and are only read by MONT3D itself. The Unix utility *od* (octal dump) may be used to examine binary files (perform a *man* on *od* for instructions). The description of the input file format is given in Chapter 3 and most of the other files are described in Appendix B.

## 5.4 Specifying File Names

MONT3D must have the names specified for all of the files shown in Table 5.1. This section explains the naming conventions and methods of specifying these file names.

There are two methods of specifying the file names; default and command line. If no name(s) are specified on the command line (or the command line is disabled, see Section 5.1.2), the code will query the user via the console for a base file name (*fn*). This base name is used as the prefix for all files in the run. The command line method of specifying the file names allows the user more flexibility in defining file names. Most file names can be specified independently according to the conventions in Table 5.2.

Table 5.1: MONT3D Files

| Unit | Name | Function |
|---|---|---|
| 1 | fn.scr | Temporary input file with comment cards "stripped away." File is generated by code. This is the file actually read during the input phase. |
| 2 | fn.rst | Restart file required for restarting from a crash or a completed run. |
| 3 | fn.plt | Plot file, to be used with the program MPLOT [Nagesh and Burns, 1994]. Contains geometrical and material property information. |
| 4 | fn.lst | Contains lost photon trajectories, if any. This file can be viewed by MPLOT [Nagesh and Burns, 1994]. For the parallel version, a separate file, fnxx.lst, is generated for each worker process xx. |
| 5 | stdin | Standard input (keyboard). |
| 6 | stdout | Standard output (screen). |
| 7 | fn.in | Input file with comment cards, in the format described in Chapter 3. This file is used to generate the file of unit 1. |
| 8 | fn.nij | Exchange matrix file which contains the results used by the thermal analysis code TOPAZ3D [Shapiro, 1985]. See Section 1.4 for more details. |
| 9 | fn.trc | Trajectory file (written if ITRACES $\neq 0$, see Section 3.2.2). To be used with MPLOT [Nagesh and Burns, 1994], to plot particle trajectories. For the parallel version, a separate trace file, fnxx.trc, is generated for each worker process xx. |
| 10 | fn.out | ASCII output file. Contains echo of all input and other information as determined by IPRNT (see Section 3.2.2). The parallel version also generates for each process xx a temporary output file, fnxx.out, that is deleted at the end of a completed run. If the run exits abnormally, these files may be used for debugging. |
| 11 | fn.lks | Leaks file. To be used with MPLOT [Nagesh and Burns, 1994] to identify potential leaks. |
| 12 | fnxx.bni | Temporary ASCII file which contains the exchange matrix for the $xx^{th}$ block of surfaces. The full exchange matrix (.nij) file only exists at the beginning and end of a run. During the run, the block exchange matrix files are used instead. |
| 13 | fn.tni | Temporary binary exchange matrix file used to convert back and forth between the exchange matrix (.nij) file and the block exchange matrix (.bni) files. |
| 14 | fn.blk | Temporary ASCII file containing block information required for restarting from a crash. |

Table 5.2: Command Line File Control

| File name to be specified | Preceded on command line by |
|---|---|
| Restart file name | -r, -R, r=, or R= |
| Plot file name | -p, -P, p=, or P= |
| Lost photon trajectory file name | -m, -M, m=, or M= |
| Input file name | -i, -I, i=, or I= |
| Absorption exchange matrix file name | -e, -E, e=, or E= |
| Trajectory file name | -t, -T, t=, or T= |
| Output file name | -o, -O, o=, or O= |
| Leaks file name | -l, -L, l=, or L= |
| Family file name | -f, -F, f=, or F= |

Several conventions bear emphasizing. If even one file name is specified on the command line, the code will not query the user for a base file name. If the *f* option is used, any file not explicitly specified will assume the naming convention indicated in Table 5.1. Those file names explicitly specified will override this default. If the *f* option is not used, then, as a minimum, the names of the input file, output file and exchange matrix file must be specified independently on the command line with the other files deriving their base name from the input file name. The maximum length of the base name is set by the parameter, *iflen*. The default value is 20 characters. Indices are always appended to the end of the base name. The number of digits used for the indices are controlled by the parameter, *iimag*. More details on these parameters is given in Section 5.5.

To help clarify the command line conventions, the following examples and explanations are offered.

**Example 1:** %mont3d -i alakazam -Okaboom e= ardvark M=toasted

MONT3D will expect the input file named *alakazam* to exist. MONT3D will create the following files; the lost photon trajectory file named *toasted (toasted01, toasted02,* ... up to the number of worker processes if this is the parallel version), the absorption exchange matrix file named *ardvark*, the output file named *kaboom* (additional temporary files named *kaboom01, kaboom02,* ... up to the number of worker processes for the parallel version). All other files will use *alakazam* for their base name.

**Example 2:** %mont3d -f calendar

MONT3D will expect the input file *calender.in* to exist. MONT3D will create all files using the base name *calendar.*

**Example 3:** %mont3d -f box -e mi5run

MONT3D will expect the input file *box.in* to exist. The exchange matrix files will use the base name *mi5run*. All the other files will use the base name *box.*

## 5.5 Parameter Statements and Memory Allocation

MONT3D uses static common blocks for storage, making the code more robust and easier to edit. Unfortunately, this also requires that the sizes of arrays be pre-specified. The sizes of various arrays are pre-specified with parameter statements. Several other features of the code are also specified by parameter statements.

All parameters in MONT3D are stored in one file, *mont3d.par*. To change a parameter, the value need be changed only in the file, *mont3d.par*, and all files must be recompiled. MONT3D has many parameters; any parameters not listed in this section should not be altered.

### 5.5.1 Parameters Specifying Array Sizes

These parameters establish the maximum sizes of various arrays, such as number of surfaces, number of wavelength bands, number of materials, etc. If the numbers are set too low, MONT3D will issue an error message informing the user that insufficient storage space is available, and terminate. It should be noted that the larger the arrays, the more memory the program uses. If the arrays become too large, the program may exceed available physical memory and the execution speed will severely degrade because virtual memory is used (where the program is swapped to and from disk).

The array-sizing parameters used in MONT3D and their default values are:

| | |
|---|---|
| iblk (maximum value for \|NEBLOCK\|, the number of surfaces in a restart block): | 200 |
| ibnd (maximum value for NBANDS, the number of wavelength bands): | 5 |
| iimag (maximum number of index digits for file names; see Section 5.5.2): | 4 |
| imat (maximum value for NUMMAT, the number of materials): | 30 |
| incg (maximum number of grid divisions along any axis; see Section 3.3): | 60 |
| inod (maximum value for NUMNP, the number of nodes): | 15,000 |
| iseg (maximum number of surfaces [segments] in all grid cells): | 100,000 |
| isrf (maximum value for NSURF, the number of surfaces): | 5,000 |
| isrfs (maximum number of surfaces including split surfaces): | 7,000 |
| itblk (maximum number of restart blocks in a run): | 1,000 |
| iwproc (maximum value for NWPROC, the number of worker processes): | 64 |

Most of the parameters listed above are rather straightforward. The value of several input variables, most of which are defined in Sections 3.2 and 3.3, must be compatible with the memory limits set by the values of the above parameters.

Results in MONT3D are stored in memory for a block of |NEBLOCK| surfaces before being written to disk; see Sections 2.9 and 3.2.1. |NEBLOCK| can not be larger than *iblk*.

When the grid shading option is used in MONT3D (see Section 2.6), the program stores in one long array a list of all surfaces which are completely or partially in each grid cell. The length of this array cannot be larger than *iseg*.

As explained in Section 2.3, if the nodes of a quadrilateral surface are not coplanar, MONT3D may have to split the surface into two planar triangles. Every time a surface is split, storage is required for an additional "split" surface. While the maximum number of surfaces that can be specified is *isrf*, the maximum number of surfaces including split surfaces is specified by *isrfs*.

For a geometry with NBANDS*NSURF emitters, NEMIT, there are NEMIT/|NEBLOCK| restart blocks in a run (round the result up). This number can not be larger than *itblk*.

It is a fairly simple process for the user to compile MONT3D for each geometry so that it uses the smallest amount of memory possible. If memory on a machine is limited, this may allow the code to run without having to use virtual memory. The way this is done is as follows. The user should first run a data check only (IDATA = 1, see Section 3.2.2), which will only go through the input phase of the code. As shown in Chapter 4, during the input phase, a "Memory Allocation and Usage" section is printed. All parameters listed in that section should be set to the value in the "used" column of that section and the code should be recompiled. The newly compiled code will use the minimum memory required for the given geometry.

### 5.5.2 Other Parameters

The other parameters used in MONT3D and their default values are:

| | |
|---|---|
| iflen (maximum length of a file's base name): | 20 |
| ilarg (maximum length of a command line argument): | 30 |
| iseeddef (default seed for RNG; see Section 3.2.1): | 19,895,660 |
| lflen, lflag, lfl, lfs (parameters for RNG, see below) | |
| m3dparaname (name of the code's parallel version; see Sections 5.2.1 and 5.6): | m3dpvm |
| pdamax (maximum allowed value for percent area difference; see Section 2.3): | 0.1 |
| zee (*z* value from normal tables, default = 95% confidence): | 1.96 |

There are a few parameters controlling file name length and input that should be explained. *iflen* is the maximum possible length for the base name (file name without indices or suffix) for a file. *ilarg* is the maximum length of a single argument on a command line or the base name entered interactively. If the base file name obtained this way is longer than *iflen*, the base name is clipped to *iflen*. Having *ilarg* larger than *iflen* may prevent possible array bounds errors on some systems. *iimag* is another file parameter that needs explanation. In creating the block exchange matrix (.bni) files, MONT3D must append the block number to the file's base name. Furthermore, several files generated in the parallel version of the code need the worker process number appended to the base name. *iimag* specifies the maximum number of digits which can be appended to the base name. For example, for 50 processes, *iimag* should be 2; while for 100 blocks, *iimag* should be 3.

The length of the lagged Fibonnaci generator can be changed if desired. As shown by Pryor et al. [1994], it takes four constants to describe a LFG fully. Besides *l* and *k* mentioned in Section 2.11, two constants, L and S, are needed to specify the format of the canonical seed. The code defines these four constants in the file *mont3d.par* as *lflen* (for LF length), *lflag*, *lfl*, and *lfs*. Several sets of (*l*, *k*, L, S) are given by Pryor et al. [1994]. Popular values are (55, 24, 1, 11), (127, 97, 1, 21) and

(607, 273, 1, 105) [Pryor, 1997]. To change the LFG, the user need change only these constants and recompile the code.

*zee* is the constant *z* defined in Section 2.7. For 95% confidence in the results, *z* is 1.96 (the default value). Other values of *z* as a function of D, the percent confidence, are given in the normal distribution tables (see Kreyszig [1993]).

## 5.6 Parallel Version

### 5.6.1 Running the Parallel Version

Much information about setting up PVM and running PVM programs is given by Geist et al. [1994]. The discussion below gives several tips on how to run the PVM version of MONT3D but for more complete information, see Geist. For an overview of the master-worker model used by MONT3D, see Section 2.10. It should be noted that several shell environment variables including the path, $PVM_ROOT, and $PVM_ARCH must be set correctly for the PVM code to run.

There are two ways to create the PVM virtual machine. The first way is to type the command *pvm* and then use the *add* command at the prompt *pvm>*. A second way is to create a hostfile. If the hostfile is given as an argument to the *pvm* command, then a PVM virtual machine will be created using the commands and options in the hostfile. A hostfile allows the user to specify a virtual machine much more easily than manually using the PVM command *add*. A hostfile also has other benefits, as will be shown below. More detail about hostfiles is given by Geist [1994].

The parallel virtual machine is created on the machine on which the master process will run. After the parallel virtual machine is created, the user just executes MONT3D executable (usually called *m3dpvm*) on the master machine. Just like the single processor version, the parallel version can be run interactively or using the command line as explained in Section 5.4. Once the input file is checked for errors, the master then spawns NWPROC (see Section 3.2.2) worker processes on the virtual machine. The PVM daemon on the master selects the CPU's on which to spawn the processes; more on this below.

To run the parallel version of MONT3D successfully, it is important that certain directories be correctly specified. The path for the executable file must be known. For the master, the path for the executable must be in the user's path or must be explicitly given on the command line. The path for executables for each of the worker processes is set to $HOME/pvm3/bin/$PVM_ARCH by default. $HOME is the user's home directory and $PVM_ARCH is an environment variable describing the platform being used. One way to change the directories for executables for each CPU in the virtual machine is to use the "ep=" option (executable path) in the PVM hostfile.

The working directory, where the code will execute and look for files, must also be specified. For the code to run properly, all processes must use the same directory so they can share disk files. For the master, the working directory is the directory in which the program is run. For the worker processes, the default value is $HOME. The working directory for each CPU in the virtual machine can be changed using the "wd=" option in the PVM hostfile. If $HOME is different for the user's accounts on different machines, the user must change the working directory for the machines so that all use the same directory.

When each worker spawns, it attempts to run a specifically named executable, It is very important that the executable be named properly, or the executable will not be found and the process will

exit in error. The default name for the executable is *m3dpvm* but it can be changed by changing the parameter *m3dparaname* and recompiling the code. More detail is given in Sections 5.2.1, 5.5.2, and 5.6.4.

When debugging the code, it is helpful to have separate debugging sessions for each worker process. If IPAROPT(1) (see Section 3.2.2) is set to 2 or 3, then an *xterm* with a *dbx* session (on some systems, it may be some other debugger) is created for each worker process. For this to work, the DISPLAY variable must be set properly for each worker process. To accomplish this, the DISPLAY environment variable on the master must be set correctly and the variable PVM_EXPORT on the master must be set to include the DISPLAY environment variable.

When running the parallel version of the code, it should be remembered that even when restarting from a crash, it is possible to switch to different machines, change the number of worker processes or even switch between the parallel and single processor versions of the code. More detail is given in Section 4.5.

### 5.6.2 Worker Processes

It should be noted that in the parallel environment, the master and workers are just Unix processes. While it is possible to have multiple worker processes on one CPU, it is generally not recommended. On a Unix workstation, only one process is using the CPU at a time. When more than one process is executed on a CPU, CPU cycles are wasted in the overhead of switching between processes. For the master-worker model used in MONT3D, one worker process on a CPU is always more efficient than two worker processes on the same CPU. There is only one time when a user might want two processes on the same CPU. Since the master process does so little work during the solution phase, the minimum overall wall clock time may be obtained by placing a worker process on the CPU that has the master process. Research into the benefits of having a worker on the master CPU is currently being done. The benefit is probably greater as the total number of CPU's used on a problem decreases.

PVM does not allow much control over which processes run on which CPU. To determine the CPU on which to spawn a process, PVM usually uses a list of the CPU's in the order that they were added to the virtual machine. The master CPU is at the end of the list. Multiple processes are put on one CPU only if NWPROC, the number of worker processes specified by the user, is greater than the number of CPU's in the virtual machine. Since having more than one worker process on a CPU is inefficient, NWPROC should always be less than or equal to the number of CPU's in the virtual machine. If NWPROC is equal to the number of CPU's, then a worker process will spawn on the master CPU. If the user wants to insure that a worker does not spawn on the master CPU, he should set the input variable IPAROPT(1) to 1 (use IPAROPT(1) = 3 if debugging); see Section 3.2.2 for more detail.

It is possible to do batch jobs as described in Section 5.7 with the parallel version of the code. It should be noted that if more than one run is done in succession using the same virtual machine, it will rotate through the list of CPU's and sooner or later a worker process will spawn on the master CPU. If the user wishes to prevent this, IPAROPT(1) should be set to 1.

### 5.6.3 Files

The file structure for the parallel version of the code requires a bit of explanation. Most of the files listed Table 5.1 are generated and used by the master process. This includes the restart (.rst)

file, the plot (.plt) file, the exchange matrix (.nij) file, the leaks (.lks) file, the temporary binary exchange matrix (.tni) file, and the block (.blk) file. The scratch (.scr) file is generated by the master, but used by both the master and worker processes. The input (.in) file is, of course, supplied by the user. The block exchange matrix (.bni) files are generated and used by both the master and the workers.

A lost photon (.lst) file and a trace (.trc) file are generated by each worker. Each file has the worker process's ID appended to the base name. The ID number is a number between 1 and NWPROC that is supplied to the worker by the master. For example, if the base name of the run is fn, the third worker process will create the files with names such as fn03.lst and fn03.trc. The number of digits in the index is controlled by the parameter *iimag*; see Section 5.5.2. Each of these files can be viewed individually using MPLOT [Nagesh and Burns, 1994].

A single output (.out) file is generated by the master process, but temporary output files indexed by worker ID (i.e. fn03.out) are also generated by each worker process. These temporary output files are deleted at the end of a successful run. Their main purpose is to provide information useful for tracking down problems if the run terminates due to errors.

### 5.6.4 Errors

While most errors from the master are very similar to that of the single processor version of the code, PVM errors and some worker errors need some additional explanation. Spawning errors have special error messages and may occur when the processes are trying to spawn. The other types of worker errors described below have a generic error message which indicates the basic error type, which worker had the error on which block, and other information. If a worker encounters any error, it terminates. As stated in Section 2.10, the first time a block of surfaces fails, the block is sent to another worker. The second time the block fails, the entire run is aborted. As described above, if the run terminates from errors, output files for each process are generated. These can be used to get further information about the errors.

**Spawning errors:** Errors in spawning often means that one or more machines or the communication between them are down. The error can either occur on the master, which is called a system error, or it can occur on one or more workers when they fail to start their copy of MONT3D. If a system error occurs or all the workers fail to start MONT3D, the run exits with an error. If some of the workers fail to start the program, a serious warning is issued but the run continues with the remaining workers. To help identify the spawning problem(s), the code prints out the PVM error code(s) for either the system error or for each failing worker. Each error code is a number that represents a PVM error condition. Which code specifies which condition is given in the file *fpvm3.h* which can usually be found in the directory $PVM_ROOT/include. More detail of these spawning errors can be found in the PVM documentation about the function *pvmfspawn*.

A particularly common spawning error is when all the workers fail to start their copy of MONT3D. This usually occurs due to one of three causes. First, the MONT3D executable may not be in the directory PVM searches for executables; see Section 5.6.1 for more detail. Second, the worker process may not be able to start the MONT3D executable because the executable is named incorrectly. When each worker spawns, it attempts to run the executable with the name specified by the parameter *m3dparaname*. Therefore, for the parallel code to run, the executable must be named as specified by *m3dparaname* parameter which has a default value of *m3dpvm*. For more

detail, see Sections 5.2.1, 5.5.2, and 5.6.1. Third, the executable can be compiled for a platform other than the one on which the worker process is spawned.

**General error:** General errors are a catch-all category. If the block number is 0, the error occurred during the input phase for the worker. Check the process's output file for more details.

**Not Enough Memory Allocated on Worker:** When working on a heterogeneous architecture, at least some of the workers will use a different version of the executable than the master. In those cases, it is possible for a worker executable not to have enough memory allocated even if the master executable does. If this error occurs, check the worker process's output file to determine which parameters defining memory allocation need to be increased. For more detail, see Section 5.5.

**Emission Point Not in Grid:** This error occurs when an emission point from a surface is not within the grid. To our knowledge, this error has never occurred for any geometry. If it does occur, there may be serious problems in the geometry specification.

**Maximum Number of Photons Lost per Worker Process**: This error occurs when more then NLOST (input format described in Section 3.2.1) photons are lost per worker process.

**Maximum Number of Reflection Warnings per Process**: This error occurs for a process when there are more then NWARNS warnings of a photon reflecting more than NREFS times (both described in Section 3.2.1).

**PVM Error:** A PVM error occurs when a PVM function exits with an error. If no worker process is mentioned, then the error occurred on the master. To our knowledge, this type of error has not yet occurred. Quite a bit of information is printed for this type of error, but it is mainly intended to be given to the code's authors so they can determine the problem. For any one trying to track down the problems themselves, the error message includes the following information. First, it gives the PVM function that caused the error. Second, the item marked "error code" specifies which PVM error condition occurred. The error code numbers are described in *fpvm.h* as discussed above in spawning errors. The "mestag" item gives the message tag ID. Each mestag number is equivalent to a message tag parameter defined in the file *m3pvm.inc*. The PVM version of the code passes six different types of messages. The message tag parameters are used to identify these six message types for both the master and the workers. If the error occurred while packing or unpacking data for communication, the last number marked "item" indicates for which item, from the first to the last data item, the error occurred.

### 5.6.5 Random Numbers

As discussed in Section 2.11.2, each worker process is supplied with an unique seed generated by the master process, which the worker then masks with the process number to obtain the initial seed for that processor for the run. While the random number state is saved for single processor restart runs, this is not the case for the parallel version. Instead, the workers receive a totally new seed from the master for each restart run. The reason for this is that the runs are done asynchronously under PVM. Since exact reproduction is impossible, a different set of random seeds is used for each parallel restart run, and CPU time and disk space are saved by not saving the random states.

When parallel restart runs are done, care must be taken to ensure that the random number streams are uncorrelated with those of the previous run. Since MONT3D uses canonical seeds generated by a binary shift register, if the initial seeds used by the processes differ by even one bit, then good

statistical behavior between the sequences is assured. If the default seed (INSEED < 0; see Section 3.2.1) is used for two consecutive continuing runs, then the same initial seeds will be used for each run and the same set of random number sequences will be used in both runs which may cause problems. Similarly, if the user specifies a specific random number seed for INSEED (INSEED > 0), a different seed must be entered for each run.

On the other hand, a different time-generated seed is produced for nominally every second in a 32 year period. Therefore, if time generated seeds (INSEED = 0) are used, independent streams of random numbers are virtually assured. It should be noted that time generated seeds vary only with time if the code supports the time and date functions (see Section 5.1.4). If time and date functions are not supported, the user must enter different values of INSEED for each restart run. It should be noted that the time-generated seed is generated on the master process only. If the master's version of MONT3D supports time and date functions, then time generated seeds will work properly.

## 5.7 Unix Batch Execution Using Scripts

This section describes the use of a Unix shell script to submit multiple MONT3D runs to be executed sequentially. This has the advantage of running only one job at a time, thereby avoiding the overhead involved with swapping jobs in memory in and out of the CPU (our tests have shown that the overhead under Unix in so doing is prohibitively large). The shell script in Figure 5.1 is an example which runs MONT3D three times for three separate problems (input files *tc1*, *tc2*, and *tc3*). In this example line one executes *mont3d* and redirects standard output to the file *run1*. The second line causes the script to pause until line one has finished. When the first run has completed the process continues with line three, etc.

```
mont3d -f tc1 > run1
wait
mont3d -f tc2 > run2
wait
mont3d -f tc3 > run3
```

Figure 5.1: Contents of Script File "submit"

The script file can be created with any ASCII editor and given any valid Unix name (here the file is named *submit*). The files referred to in the script (here *tc1, run1,* etc.) can also have any valid Unix name. The files *tc1*, *tc2*, and *tc3* must be the MONT3D input files (all must have the .in extension) to be used for the run.

Once the script is created it must have execute permission before it can be run. This is accomplished for the file *submit* with the Unix command *chmod*:

```
%chmod 744 submit
```

where % is the Unix prompt.

The script can be run in the background by typing the script name followed by "&." If commands are run in the C shell, then to run *submit* in the background and redirect all its output to standard output and standard error to the file, *submit.out*, type:

```
%submit >& submit.out &
```

where the ">&" redirects both standard output and standard error to the file submit.out. Alternately, it can be submitted using the Unix commands *at* or *batch*. The syntax of any of these Unix commands can be obtained by referring to the Unix *man*(ual) pages.

## 5.8 Precision

MONT3D is coded to compile and run using 64-bit floating point precision. For MONT3D, it is desirable to use 64-bit precision, as the specification of coplanar surfaces is particularly susceptible to precision errors. Also, 3-D photon tracing generally requires 64 bits to perform intersection calculations with the requisite accuracy.

# REFERENCES


Anderson, S. L., 1990. “Random Number Generators on Vector Supercomputers and Other Advanced Architectures,” *SIAM Review, 32,* pp. 221-251.

Branner, K., 1999. *An Enhanced Material Model for Radiative Heat Transfer via Monte Carlo*, M.S. Thesis, Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523. (In progress).

Brent, R. P., 1992. “Uniform Random Number Generators for Supercomputers, “*Proceedings Fifth Australian Supercomputing Conference*, SASC Organizing Committee, 1992, pp. 95-104. (Unpublished).

Burns, P. J., and Pryor, D. V., 1989. “Vector and Parallel Monte Carlo Radiative Heat Transfer,” *Numerical Heat Transfer, Part B: Fundamentals*, *16*, pp. 20-42.

Burns, P. J., Maltby, J. D., and Christon, M. A., 1990. “Large-Scale Surface to Surface Transport for Photons and Electrons via Monte Carlo,” *Computing Systems in Engineering*, Vol. 1 No. 1, pp. 75-99.

Burns, P. J., Loehrke, R. I., Dolaghan, J. S., and Maltby, J. D., 1992. “Photon Tracing in Axisymmetric Enclosures,” *Developments in Radiative Heat Transfer, HTD-Vol. 203*, pp. 93-100, ASME, New York.

Burns, P. J., and Pryor, D. V., 1995. “Random Numbers,” Available as a WWW document, URL = http://csep1.phy.ornl.gov/CSEP/RN/RN.html.

Burns, P. J., and Pryor, D. V., 1999. “Surface Radiative Transport at Large Scale via Monte Carlo,” Vol. 9 of *Annual Review of Heat Transfer*, Begell House, New York, NY. (in press).

Cuccaro, S. A., 1996. Personal communication, Center for Computer Sciences, 17100 Science Drive, Bowie, MD 20715.

Crockett, D. V., Maltby, J. D., and Burns, P. J., 1990. “User’s Manual for MONT3E - A Three-Dimensional Electron-Tracing Code with Non-Uniform Magnetic Field, Release 5.0,” Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523.

Dolaghan, J. S., 1991. *A Monte Carlo Simulation of Molecular Redistribution in an Enclosure due to Sputtering*, M.S. Thesis, Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523.

Dolaghan, J. S., Loehrke, R. I., and Burns, P. J., 1992. “User’s Manual for SMOOTH,” Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523.

Dolaghan, J. S., 1996. *A Monte Carlo Simulation in Rarefied Gas Dynamics with Application to Physical Vapor Deposition*, Ph.D. Dissertation, Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Mancheck, R., and Sunderam, V., 1994. *PVM: Parallel Virtual Machine - A User’s Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA. Also available as a WWW document, URL = http://www.netlib.org/pvm3/book/pvm-book.html.

Golomb, S. W., 1982. *Shift Register Sequence*s (Revised Ed.), Aegean Park Press.

Kreyszig, E., 1993. *Advanced Engineering Mathematics, 7th Ed.*, John Wiley & Sons, New York.

Maltby, J. D., Burns, P. J., and Winn, C. B., 1986. "Monte Carlo Simulation of Radiative Heat Transport in Passive Solar Buildings," *Proceedings of the 1986 American Solar Energy Society Conference,* Boulder, Colorado (June 9-11, 1986).

Maltby, J. D., 1987. *Three-Dimensional Simulation of Radiative Heat Transfer by the Monte Carlo Method*, M.S. Thesis, Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523.

Maltby, J. D., 1990. *Analysis of Electron Heat Transfer via Monte Carlo Simulation*, Ph.D. Dissertation, Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523.

Maltby, J. D., and Burns, P. J., 1991. "Performance, Accuracy and Convergence in a Three-Dimensional Monte Carlo Radiative Heat Transfer Simulation," *Numerical Heat Transfer, Part B; Fundamentals*, Vol. 16, pp. 191-209.

Maltby, J. D., Zeeb, C. N., Dolaghan, J., and Burns, P. J., 1994. "User's Manual for MONT2D - Version 2.6 and MONT3D - Version 2.3," Department of Mechanical Engineering, Colorado State University, Fort Collins, CO.

Margolies, D., 1986. Personal communication, LLNL.

Marsaglia, G., 1985. "A Current View of Random Number Generators", *Computing Science and Statistics: Proceedings of the XVIth Symposium on the Interface*, L. Billard (ed.) Elsevier Science Publishers, B. V. (North Holland) pp. 3-10.

Mascagni, M., Cuccaro, S., Pryor, D., and Robinson, M., 1995. "A Fast, High Quality, and Reproducible Parallel Lagged-Fibonacci Pseudorandom Number Generator," *Journal of Computational Physics*, Vol. 119, pp. 211-219.

Nagesh, S., and Burns, P. J., 1994. "User's Manual for the Program MPLOT - Version 3.3," Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523.

Pryor, D. V., and Burns, P. J., July 21-25 1986. "A Parallel Monte Carlo Model for Radiative Heat Transfer," presented at the 1986 SIAM Meeting, Boston, MA.

Pryor, D. V., Cuccaro, S. A., Mascagni, M., and Robinson, M. L., 1994. "Implementation of a Portable and Reproducible Parallel Pseudorandom Number Generator," *Supercomputing '94 Proceedings*, IEEE Computer Society Press, Los Alamitos, CA, pp. 311-319.

Pryor, D. V., 1997. Personal communication, Center for Computer Sciences, 17100 Science Drive, Bowie, MD 20715.

Schorn, P., and Fisher, F., 1994. "Testing the Convexity of a Polygon," In *Graphic Gems IV,* Heckbert, P. S., editor, AP Professional, San Diego, CA, pp. 7-15.

Schweitzer, R., McHugh, J., Burns, P. J., and Zeeb, C. N., 1993. "Daylighting Design via Monte Carlo with a Corresponding Scientific Visualization," Supercomputing '93 Proceedings, IEEE Computer Society Press, Los Alamitos, CA pp. 250-259

Shapiro, A. B., 1983. "FACET - A Radiation View Factor Computer Code for Axisymmetric, Two-Dimensional Planer, and Three-Dimensional Geometries with Shading," Lawrence Livermore National Laboratory, UCID-19887.

Shapiro, A. B., 1985. “TOPAZ3D - A Three-Dimensional Finite Element Heat Transfer Code,” Lawrence Livermore National Laboratory, UCID-20484.

Statton, E. S., 1983. *MONTE - A Two-Dimensional Monte Carlo Radiative Heat Transfer Code*, M.S. Thesis, Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523.

Torrance, K., and Sparrow, E. M., 1966. “Off-specular Peaks in the Directional Distribution of Reflected Thermal Radiation,” *J. Heat Transfer*, Vol. 88, pp. 223-230.

Zeeb, C. N., 1996. *Two-dimensional Heat Transfer in Combustion Gases Via Monte Carlo*, M.S. Thesis at Colorado State University, Department of Mechanical Engineering, Fort Collins, CO 80523.

Zeeb, C. N., 1997. “Application of Ray Tracing Techniques to Radiative Heat Transfer Via Monte Carlo” Available as a WWW document., URL =http://www.colostate.edu/~pburns/monte/documents.html.

Zeeb, C. N., and Burns, P. J., 1997. “Random Number Generator Recommendation,” Report prepared for Sandia National Laboratories, Albuquerque, NM. Available as a WWW document., URL =http://www.colostate.edu/~pburns/monte/documents.html.

Zeeb, C. N., and Burns, P. J., 1999. “Performance Enhancements of Monte Carlo Particle Tracing Algorithms for Large, Arbitrary Geometries” To be presented at the 1999 ASME National Heat Transfer Conference, Albuquerque, NM, August 15-17, 1999.

Zeeb, C. N., and Romero, V. J., 1999. “LAYMC: A Monte Carlo Code for Modeling Photon Transport in Layered Participating Media — Theory and User Manual version 2A”, to be published as a Sandia National Laboratories report (unlimited distribution), (Expected 1999).

# APPENDIX A OLD MATERIAL MODEL

This appendix covers the old material model. The first section of this appendix gives an overview of the old material model. This is followed by a section on the specification of outgoing angles for diffuse and specular reflection and transmission. The next four sections describe the seven material types supported by the old material model. The last two sections cover the input card format for entering in the material type data and the material property curves using the old material model.

## A.1 Overview

The old material model is only used if NUMMAT, the number of material, (input specification in Section 3.2.1) is greater than 0. If NUMMAT is less than 0, the new material model, which is described in Sections 2.5 and 3.8-3.10, is used. More detail about the input specifications for the old model are given below in Sections A.7 and A.8.

The old material model has material types defined for each band. Seven different material types are available to provide flexibility to the material model. These material types allow different combinations of photon/material interactions to be defined. Table A.1 summarizes the material types and indicates which material property curves must be entered. Note that while the old material model supports specular and (weighted) diffuse reflection ($\rho_s$ and $\rho_d$) and transmission ($\tau_s$ and $\tau_d$), semi-specular interactions are only supported by the new material model.

Table A.1: Material Property Summary

| Type | Emission | Interactions | Input Curves |
|---|---|---|---|
| 2 | Function | $\tau_s(\theta)+\rho_s(\theta)+\rho_d(\theta) + \alpha(\theta) = 1$ | 3: $\tau_s(\theta)$, $\rho_s(\theta)$, $\rho_d(\theta)$ |
| 1 | Beam | $\tau_s(\theta)+\rho_s(\theta)+\rho_d(\theta)+\alpha(\theta) = 1$ | 3: $\tau_s(\theta)$, $\rho_s(\theta)$, $\rho_d(\theta)$ |
| 0 | $\varepsilon(\theta)$ | $\tau_s(\theta)+\rho_s(\theta)+\rho_d(\theta)+\alpha(\theta) = 1$ | 3: $\tau_s(\theta)$, $\rho_s(\theta)$, $\rho_d(\theta)$ |
| -1 | “ | $\tau_d(\theta)+\rho_s(\theta)+\rho_d(\theta)+\alpha(\theta) = 1$ | 3: $\tau_d(\theta)$, $\rho_s(\theta)$, $\rho_d(\theta)$ |
| -2 | “ | $\tau_s(\theta)+\tau_d(\theta)+\rho_s(\theta)+\rho_d(\theta)+\alpha(\theta) = 1$ | 4: $\tau_s(\theta)$, $\tau_d(\theta)$, $\rho_s(\theta)$, $\rho_d(\theta)$ |
| -3 | None | Perfect specular reflection | None |
| -4 | None | Perfect diffuse reflection | None |

In addition to photon/material interactions, the material type also determines the type, if any, of emission that will take place. Just as in the new material model, three possible emission modes are modeled emission according to user input function, beam emission, and normal emission. More detail is given below. It is possible to specify materials for which the reciprocity relations do not hold; more detail is given in Section 2.5.4.

It should be noted that unless the material type is -3 or -4, the curves listed in Table A.1 must be entered by point value as a function of angle. The format is given in Section A.8. The computer code parabolically interpolates between each three successive points entered, just as it does with curves entered in by point value for the new model. Section 2.5.1 gives several tips about entering curves by point value.

## A.2 Outgoing Angles for Diffuse and Specular Interactions

Figure A.1 shows the conventions for θ and ϕ for diffuse and specular interactions. The cone angle, θ, is always defined from the surface normal, and the azimuthal angle, ϕ, is positive counterclockwise (when viewed from above) from the surface x'-axis. The x'-axis is defined perpendicular to the ray joining nodes 1 and 2 (y'-axis) and the surface normal ***n*** (z'-axis) according to a right-hand rule, as shown in Figure A.1. Care should be taken to number the nodes of a surface correctly to achieve the desired emission direction. This coordinate system is used by both the new and old material models.

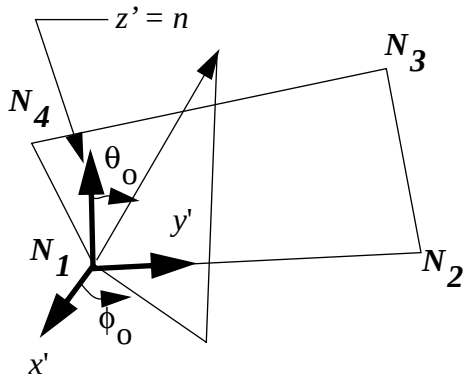



Figure A.1: Conventions for Outgoing Angles

## A.3 Material Type 2, Emission According to a User-Supplied Function

For material type 2, emission occurs according to a user-supplied function. This is exactly the same as setting IETP in the new material model to 2. Photon/material interactions are determined by the three input curves listed in Table A.1 above.

## A.4 Material Type 1, Beam Emission

An option (material type 1) is available in the code to simulate beam radiation, with all primary photon emissions from a specified material occurring in a fixed direction. To use this option, fixed values of $\phi_o$ and $\theta_o$ for a material must be entered by the user, in a format described below. It should be noted that this is very different than the beam emission defined by the new material model. The new material model defines the beam direction by an emission vector in GLOBAL coordinates. The $\phi_o$ and $\theta_o$ defined in the old material model are relative to the LOCAL coordinate system for the surface. For the old material model, the emission direction for the beam depends on the orientation of the surface.

Just as with the new material model, all interactions (absorptions, transmissions, reflections) will depend only on the material property curves defined for that material. Again, the user should be cautious when specifying the beam radiation option, since the reciprocity relations, Equations (1.5) and (1.6) may no longer be valid. The other conservation relations still hold.

## A.5 Material Types 0 Thorough -2, Normal Emission

For material type 0 through -2, emission is a function of $\theta_o$, so all of the photon/material interactions are determined by the input curves. For material 0, three curves are read and $\tau_d(\theta_i)$ is set to zero. Material type -1 also requires three input curves with $\tau_s(\theta_i)$ set to zero. For material type -2, all four curves are input.

## A.6 Material Types -3 and -4, Perfect Mirrors

Material types -3 and -4 are perfect mirrors (perfect reflectors). Material type -3 is completely specular, and material type -4 is completely diffuse. For surfaces of this material type, no emission occurs, and “zero” exchange numbers are written into the exchange matrix file.

## A.7 Material Type Data Cards

If the old material property model is used, the material type data cards should be entered right after the wavelength band data, which is where the new material model property curve information is currently entered. See Chapter 3 for more details.

Material type data cards are input by band for each material in order. For example, if three materials are used in two bands, then all material type data are read in for material 1, band 1; then material 1, band 2; then material 2, band 1; etc. MN and IB are used as checks to ensure that the values being entered are in the right order. If the information is not entered in the correct order, the program detects an error and terminates. All cards must be input before any material property curves are input (these cards control which material property curves are read).

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-5 | I5 | Material type: MTYPE(N,IB): domain - [-4,2] | 1, 2, 3 |
| | | | 4, 5 |
| | | | 6, 7 |
| 6-15 | E10.0 | Outgoing cone angle $\theta_o$ for beam radiation: THSET(MN,IB) | 1 |
| 16-25 | E10.0 | Outgoing surface azimuth angle $\phi_o$ for beam radiation: PHISET(MN,IB) | 1 |
| 26-35 | E10.0 | R dependence of diffuse reflectance: RDIFFR(MN,IB) | 5, 6 |
| | | (DEFAULT: RDIFFRD = 1) | 8 |
| 36-45 | E10.0 | R dependence of diffuse transmittance: RDIFFT(MN,IB) | 5, 6 |
| | | (DEFAULT: RDIFFT = 1) | 8 |
| 46-50 | I5 | Material number (MN) | 5, 6 |
| 51-55 | I5 | Wavelength band number (IB) | 5, 6 |

**Notes:**

1. THSET and PHISET are ignored unless MTYPE =1, or MTYPE = 2 and the user supplied function requires them.

2. If MTYPE(N) = 2, then emission occurs according to the user supplied function *fcn* in subroutine *getang* in the file *m32s.f*.
3. If a material has 0 emittance for all values of θ, then no photons are emitted from surfaces of that type unless MTYPE = 1 or 2.
4. Material property curves are required for all material types except -3 and -4.
5. If the values of MN and IB are 0 (or blank) for the first band of a material then RDIFFR and RDIFFT are set to 1 (standard diffuse re-emission) and the material type data values are used for all bands in that material and no other material type data cards are read in for that material. This has been done for compatibility with earlier versions which do not support r dependence or multiple material types.
6. If MN is not 0 but IB is for the first band of the material then it is assumed that the material type data are constant for this material for all bands and no other material type data cards are read in for this material. The material property curves must still be entered for each band.
7. As discussed in Section 2.5.4, if MTYPE = 1 or 2, the reciprocity relations, Equations (1.5) and (1.6), may no longer hold.
8. If either RDIFFR or RDIFFT = 0, then it will be set to 1 (standard diffuse distribution.) If either RDIFFR or RDIFFT are < 0, then it will be set to 0 (isotropic distribution.) Note that, as discussed in Section 2.5.4, if either of these variables do not equal 1 (standard diffuse distribution) the reciprocity relations, Equations (1.5) and (1.6), may no longer hold.

## A.8 Material Property Curves Cards

The material property curves should be entered right after the material property cards described above. These cards do have a different format than the curve cards used by the new material model and should be used instead of them.

Material property curves are input by band for each material, except for materials of type -3 and -4, where no curves are input. Curves must be input in order. For example, if three materials are used in two bands, then all material type data are read in for material 1, band 1; then material 1, band 2; then material 2, band 1; etc. MN and IB are used as checks to ensure that the values being entered are in the right order. If the information is not entered in the correct order, the program detects an error and terminates.

As discussed in Section 2.5.4, if any curve except the specular reflectance varies as a function of angle, the reciprocity relationships may not hold.

### A.8.1 Specular Transmittance

Condition(s): MTYPE = 2, 1, 0, or -2 (omit otherwise)

**CARD 1**

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-16 | 2A8 | Name of curve (e.g., specular trans): (CN1, CN2) | |
| 17-21 | I5 | Material number (MN) | 1 |
| 22-26 | I5 | Wavelength band number (IB) | 1 |

| | | | |
|---|---|---|---|
| 27-31 | I5 | Curve number (NJ) | 1, 2 |
| 32-36 | I5 | Number of points to be input for this material (NP) | 3 |

**CARDS 2 to NP+1**

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-10 | E10.0 | Cone angle - $\theta$ | 3 |
| 11-20 | E10.0 | Specular transmittance: $\rho_s(\theta)$ | |

**Notes:**

1. These values are used to check that the data are input properly by material number, band number, and curve number.
2. A curve number of 1 must be input for specular transmittance.
3. NP number of cards must be input for the curve.

### A.8.2 Diffuse Transmittance

Condition(s): MTYPE = -1 or -2 (omit otherwise)

**CARD 1**

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-16 | 2A8 | Name of curve (e.g., diffuse trans): (CN1, CN2) | |
| 17-21 | I5 | Material number (MN) | 1 |
| 22-26 | I5 | Wavelength band number (IB) | 1 |
| 27-31 | I5 | Curve number (NJ) | 1, 2 |
| 32-36 | I5 | Number of points to be input for this material (NP) | 3 |

**CARDS 2 to NP+1**

| Cols. | Format | Entry | Note |
|---|---|---|---|
| 1-10 | E10.0 | Cone angle - $\theta$ | 3 |
| 11-20 | E10.0 | Diffuse transmittance: $\tau_d(\theta)$ | |

**Notes:**

1. These values are used to check that the data are input properly by material number, band number, and curve number.
2. For diffuse transmittance, curve number = 1 for MTYPE = -1 and curve number = 2 for MTYPE = -2.
3. NP number of cards must be input for the curve.

### A.8.3 Specular Reflectance

Condition(s): MTYPE $\neq$ -3 or -4 (omit otherwise)

**CARD 1**

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-16 | 2A8 | Name of curve (e.g., specular reflectance): (CN1, CN2) | |
| 17-21 | I5 | Material number (MN) | 1 |
| 22-26 | I5 | Wavelength band number (IB) | 1 |
| 27-31 | I5 | Curve number (NJ) | 1, 2 |
| 32-36 | I5 | Number of points to be input for this material (NP) | 3 |

**CARDS 2 TO NP+1**

| Cols. | Format | Entry | Note |
|---|---|---|---|
| 1-10 | E10.0 | Cone angle - $\theta$ | 3 |
| 11-20 | E10.0 | Specular reflectance: $\rho_s(\theta)$ | |

**Notes:**

1. These values are used to check that the data are input properly by material number, band number, and curve number.
2. For specular reflectance, curve number = 2 for MTYPE = 2 to -1 and curve number = 3 for MTYPE = -2.
3. NP number of cards must be input for the curve.

### A.8.4 Diffuse Reflectance

Condition(s): MTYPE $\neq$ -3 or -4 (omit otherwise)

**CARD 1**

| Cols. | Format | Entry | Note(s) |
|---|---|---|---|
| 1-16 | 2A8 | Name of curve (e.g. diffuse reflectance): (CN1, CN2) | |
| 17-21 | I5 | Material number (MN) | 1 |
| 22-26 | I5 | Wavelength band number (IB) | 1 |
| 27-31 | I5 | Curve number (NJ) | 1, 2 |
| 32-36 | I5 | Number of points to be input for this material (NP) | 3 |

**CARDS 2 TO NP+1**

| Cols. | Format | Entry | Note |
|---|---|---|---|
| 1-10 | E10.0 | Cone angle - $\theta$ | 3 |
| 11-20 | E10.0 | Diffuse reflectance: $\rho_d(\theta)$ | |

**Notes:**

1. These values are used to check that the data are input properly by material number, band number, and curve number.
2. For diffuse reflectance, curve number = 3 for MTYPE = 2 to -1 and curve number = 4 for MTYPE = -2.
3. NP number of cards must be input for the curve.

# APPENDIX B FILE FORMATS

As discussed in Section 5.3, MONT3D generates a large number of files. This appendix gives the format of most of these files. The files are presented in order of their unit numbers, see Section 5.3 for more detail. This appendix should only be of interest to those who plan to write programs that work with the files created by MONT3D. The input file format, which is of much more general interest, is described in Chapter 3.

Files are ASCII unless otherwise specified. The ASCII FORTRAN format specifications given below are for a single record. Each record starts on a new line and if the record is larger than the specification, the format repeats as needed starting on a new line each time.

## B.1 Restart File (Suffix .rst, Unit 2)

The restart file contains the information MONT3D requires to restart from a completed or crashed run. If the program is restarting from a crash, additional information stored in the block (.blk) file described below is also required. All the code for the reading and writing of this file can be found in the subroutine *rstfile* in the file *m30s.f*.

### B.1.1 IPARFLG

**1 Record of Length 1**

**FORTRAN Format Specification: 7(i10, 1x)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Parallel run flag (IPARFLG) | 1 |

**Notes:**

1. IPARFLG is 0 for a single processor run, 1 for a parallel run.

### B.1.2 Photon Emission Counts

**1 Record of Length NSURF*NBANDS**

**FORTRAN Format Specification: 7(i10, 1x)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Number of photons emitted from each surface in each band | 1 |

**Notes:**

1. The photon counts are written in order from 1 to NSURF, for each band 1 to NBAND.

### B.1.3 Random Number Generator Information

**Only Written/Read if IPARFLG = 0**

**1 Record of Length *lflen* + 2**

**FORTRAN Format Specification: 7(i10, 1x)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | The ISEED array of seeds which is of length *lflen* | 1 |
| Integer | First tap value | 1 |
| Integer | Second tap value | 1 |

**Notes:**

1. The code uses a lagged Fibonnaci RNG of length *lflen*. The parameter *lflen* is usually 127 but can be changed, see Sections 2.11 and 5.5.2 for more details.

### B.1.4 Block Information

**1 Record of Length NBLOCKS**

**FORTRAN Format Specification: 7(i10, 1x)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | NBKFIN array | 1 |

**Notes:**

1. The NBKFIN array is NBLOCKS long where NBLOCKS is the number of surface emission blocks for the run, see Sections 2.9 and 3.2.1 for more details. NBKFIN is 1 if the block has been successfully completed; otherwise, it is 0.

## B.2 Plot File (Suffix .plt, Unit 3)

The plot file contains the basic information MPLOT [Nagesh and Burns, 1994] requires to plot the geometry, see Section 2.8 for more details. All the code for the writing of this file can be found in the subroutine *graf* in the file *m313s.f*.

### B.2.1 Header

**1 Record of Length 72**

**FORTRAN Format Specification: (10a8)**

| Format | Entry | Note(s) |
|---|---|---|
| Character | File header | 1 |

**Notes:**

1. This is a character string of 72 characters. The first 48 characters are the title for the run which is entered on the first line of the input file; see Section 3.1 for more details. The last 24 charac-

ters are three 8 character variables, which are, in order, the version number of the code, the date when the code was last modified, and the language of the source code (always “f77”). The header is the first line printed to the screen during a MONT3D run; see the screen output in Chapter 4 for an example. It is also used as a page header in the output file.

### B.2.2 Control Information

**1 Record of Length 3**

**FORTRAN Format Specification: 16i5**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Number of surfaces including split surfaces (NSPTR) | 1 |
| Integer | Number of materials (NUMMAT) | |
| Integer | Number of wavelength bands (NBANDS) | |

**Notes:**

1. While split surfaces are transparent to the user, the plot file contains information about them, not the original unsplit surfaces.

### B.2.3 Limiting Dimensions for the Geometry

**1 Record of Length 6**

**FORTRAN Format Specification: 6(1x,f12.7)**

| Format | Entry | Note(s) |
|---|---|---|
| Real | Minimum X value (XLO) | 1 |
| Real | Maximum X value (XHI) | 1 |
| Real | Minimum Y value (YLO) | 1 |
| Real | Maximum Y value (YHI) | 1 |
| Real | Minimum Z value (ZLO) | 1 |
| Real | Maximum Z value (ZHI) | 1 |

**Notes:**

1. Each of the hi values is slightly larger than the maximum value for the geometry and each lo value is slightly less than the minimum value for the geometry.

### B.2.4 Surface Information

The following records are repeated for each of the NSPTR surfaces. The surfaces are listed in order of surface number.

**Material Information**

**1 Record of Length 2**

**FORTRAN Format Specification: 16i5**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Surface number | |
| Integer | Surface's material number | |

**Node Information**

**4 Records of Length 3**

**FORTRAN Format Specification: 6(1x,f12.7)**

| Format | Entry | Note(s) |
|---|---|---|
| Real | X value of node i | 1 |
| Real | Y value of node i | 1 |
| Real | Z value of node i | 1 |

**Notes:**

1. The X, Y, and Z coordinates are listed for each of the four nodes of the surface. One node is listed per line.

### B.2.5 Material Information

The following records are repeated 4*NUMMAT*NBANDS times. The records are repeated for the four cumulative property types from the old material model in the following order: specular transmittance ($\tau_s$), diffuse transmittance ($\tau_d$) + $\tau_s$, specular reflectance ($\rho_s$) + $\tau_d$ + $\tau_s$, and diffuse reflectance ($\rho_d$) + $\rho_s$ + $\tau_d$ + $\tau_s$. This series of material property records is repeated from 1 to NUMMAT times in each band in order of material number which in turn is repeated from 1 to NBANDS for each band in order of band number.

**Curve Name**

**1 Record of Length 16**

**FORTRAN Format Specification: a16**

| Format | Entry | Note(s) |
|---|---|---|
| Character | Curve name | |

**Curve Values**

**1 Record of Length 91**

**FORTRAN Format Specification: f10.7**

| Format | Entry | Note(s) |
|---|---|---|
| Real | Curve values | 1 |

**Notes:**

1. Curve values are listed in order from 0 to 90 degrees.

### B.2.6 Number of Records in the Binary Exchange Matrix File

**1 Record of Length 1**

**FORTRAN Format Specification: i10**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Number of records in file | 1 |

**Notes:**

1. This value was used by MPLOT [Nagesh and Burns, 1994] to read the exchange matrix information from older versions of MONT3D which used familied binary files. Due to the new exchange matrix file format used by MONT3D, this value no longer has any meaning and is always set to 1.

## B.3 Lost Photon File (Suffix .lst, Unit 4)

The lost photon file contains the trajectory (photon ray end point) information for each lost photon and can be used with MPLOT [Nagesh and Burns, 1994] to display the trajectories, see Section 2.8 for more details. All the code for the writing of this file can be found in the subroutine *grdint* in the file *m32s.f*.

The following series of records are repeat for each lost photon. Records are written in order of lost photon number.

### B.3.1 Header Card

**1 Record of Length 5**

**FORTRAN Format Specification: 5(3x, i7)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Lost photon number (LOST) | |
| Integer | Event number (IEVENT) | 1 |
| Integer | Number of points to be plotted along the particle's trajectory (NPNTS) | 2 |
| Integer | Subdivision number (ND) | 3, 4 |
| Integer | Number of surface where last emitted/reflected or transmitted (LERT) | 3, 4 |

**Notes:**

1. IEVENT is always set to 1.
2. NPNTS is always 2, and indicates that the trajectory just includes the starting and ending points.

3. In the MPLOT description of this file [Nagesh and Burns, 1994], the last two numbers written to this record are supposed to be an integer, MAT, and a real, E0. Neither of these two variables apply to MONT3D so they are replaced by ND and LERT.
4. LERT is the surface number of the last surface with which the photon was in contact, whether that was as an emission, a reflection or a transmission. As described in Section 2.2, each surface is divided into NDIVX x NDIVY subdivisions. ND gives the number of the subsection from which the lost photon was emitted. It ranges from 0 to NDIVX x NDIVY - 1. It should be realized that if a photon is reflected or transmitted before it is lost, LERT is generally not the surface that emitted the photon.

### B.3.2 Photon Ray's Starting Point

**1 Record of Length 3**

**FORTRAN Format Specification: 3(2x, e10.4)**

| Format | Entry | Note(s) |
|---|---|---|
| Real | X coordinate of photon ray's starting point | |
| Real | Y coordinate of photon ray's starting point | |
| Real | Z coordinate of photon ray's starting point | |

### B.3.3 Photon Ray's Ending Point

**1 Record of Length 3**

**FORTRAN Format Specification: 3(2x, e10.4)**

| Format | Entry | Note(s) |
|---|---|---|
| Real | X coordinate of photon ray's ending point | |
| Real | Y coordinate of photon ray's ending point | |
| Real | Z coordinate of photon ray's ending point | |

## B.4 Exchange Matrix File (Suffix .nij, Unit 8)

As discussed in Section 1.4, thermal balance codes such as TOPAZ3D [Shapiro, 1985] require the exchange matrix information generated by MONT3D as input. To be compatible with TOPAZ3D, this file may need to be processed with the program SMOOTH [Dolaghan et al., 1992], which smooths the full matrix of exchange numbers into an upper triangle of numbers which obey reciprocity. All the code for the reading and writing of this file can be found in the subroutines *nijfile* and *nijheader* in the file *m30s.f*.

### B.4.1 Header Card

**1 Record of Length 5**

**FORTRAN Format Specification: 7(i10, 1x)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Geometry code (NDIM) | 1 |

| | | |
|---|---|---|
| Integer | Number of surfaces (NSURF) | |
| Integer | Factor code (IFACT) | 2 |
| Integer | Number of wavelength bands (NBANDS) | |
| Integer | Number of materials (NUMMAT) | |

**Notes:**

1. NDIM is 3.
2. IFACT is set to 2, and indicates that the file contains exchange numbers as opposed to view factors.

### B.4.2 Surface Areas

**1 Record of Length NSURF**

**FORTRAN Format Specification: 6(e12.5, 1x)**

| Format | Entry | Note(s) |
|---|---|---|
| Real | Surface areas from 1 to NSURF | |

### B.4.3 Surface Emittances

**NBAND Records Each of Length NSURF**

**FORTRAN Format Specification: 6(e12.5, 1x)**

| Format | Entry | Note(s) |
|---|---|---|
| Real | Hemispherical surface emittances, $\bar{\varepsilon}_i^k$, from 1 to NSURF for band *k* | 1, 2 |

**Notes:**

1. The surface emittances are averaged over the hemisphere (i.e. θ).
2. The surface emittances are written in order from 1 to NSURF, for each band 1 to NBAND. A new record (line) is started for each band.

### B.4.4 Photon Number Matrix

**NBAND Sets of NSURF Records Each of Length NSURF**

**FORTRAN Format Specification: 7(i10, 1x)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Numbers of absorbed photons emitted by surface *i* absorbed by all surfaces *j* for band *k* | 1, 2 |

**Notes:**

1. Each record contains the number of photons absorbed by each surface *j* ranging from 1 to NSURF for each surface *i* in the band *k*. It should be noted that the file contains results for all surfaces, even those that do not emit (which therefore have all zeroes for results). A new record (line) is started for each surface. The records for all surfaces in order from 1 to NSURF in band *k* specify an exchange matrix for band *k*. The exchange matrices for the 1 to NBANDS bands are written in order to the file.
2. This file contains the full exchange matrices. The program SMOOTH [Dolaghan et al., 1992] may be used to process these files into an upper triangle (including the diagonal) of numbers of photons which have been smoothed to obey reciprocity.

## B.5 Trajectory File (Suffix .trc, Unit 9)

The trajectory file contains the trajectory (photon ray end point) information for each photon and can be used with MPLOT [Nagesh and Burns, 1994] to display the trajectories, see Section 2.8 for more details. All the code for the writing of this file can be found in the subroutine *surfloop* in the file *m32s.f*.

The following series of records are repeat for each photon trajectory. Records are written in chronological order.

### B.5.1 Header Card

**1 Record of Length 5**

**FORTRAN Format Specification: 5(1x,i5)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Photon number (IHIST) | 1 |
| Integer | Event number (IEVENT) | 2 |
| Integer | Number of points to be plotted along the particle's trajectory (NPNTS) | 3 |
| Integer | Material number of emitting surface | 4 |
| Integer | Wavelength band number in which the photon is being traced | 4 |

**Notes:**

1. IHIST starts at 1 and is incremented 1 for each new photon emitted. It is not incremented when a photon is lost and re-emitted.
2. IEVENT is set to 1 when a photon is emitted and is incremented by 1 every time it is reflected or transmitted. IEVENT is reset to 1 when a photon is lost and re-emitted.
3. NPNTS is always 2, and indicates that the trajectory includes just the starting and ending points.
4. In the MPLOT description of this file [Nagesh and Burns, 1994], the last two number written to this record are supposed to be an integer, MAT, and a real, E0. Neither of these two variables apply to MONT3D so they are replaced by the values listed above.

### B.5.2 Photon Ray's Starting Point

**1 Record of Length 3**

**FORTRAN Format Specification: 3(2x, e10.4)**

| Format | Entry | Note(s) |
|---|---|---|
| Real | X coordinate of photon ray's starting point | |
| Real | Y coordinate of photon ray's starting point | |
| Real | Z coordinate of photon ray's starting point | |

### B.5.3 Photon Ray's Ending Point

**1 Record of Length 3**

**FORTRAN Format Specification: 3(2x, e10.4)**

| Format | Entry | Note(s) |
|---|---|---|
| Real | X coordinate of photon ray's ending point | |
| Real | Y coordinate of photon ray's ending point | |
| Real | Z coordinate of photon ray's ending point | |

## B.6 Leaks File (Suffix .lks, Unit 11)

The leaks file contains information about potential leaks. It can be used with MPLOT [Nagesh and Burns, 1994] to display these potential leaks, see Section 2.8 for more details. All the code for the writing of this file can be found in the subroutine *order* in the file *m313s.f*.

The following record is repeated as needed for each potential leak.

### B.6.1 Leak Information

**1 Record of Length 3**

**FORTRAN Format Specification: (1x, i2, 1x, i5, 1x, i2)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Type of leak (ITYPEL) | 1, 2 |
| Integer | Number of the surface associated with the leak | |
| Integer | Number of the side associated with the leak | 3 |

**Notes:**

1. ITYPEL is 1 for a reversed edge, 2 for no match found with any other surface, 3 for a slip surface. If ITYPEL is equal to 4 then the line is a continuation card that contains additional information about the last leak.
2. The number of records required for each type of error varies. While only one record is required for ITYPEL equal 2, two are required for ITYPEL equal 1 since the reversed edge is shared by two surfaces. For ITYPEL equal 3, up to 15 records may be required since a slip surface may

touch many surfaces. The limit of 15 is a hardcoded limit in MONT3D itself. When more surfaces than this are involved with the slip surface, they are not written to the leaks file.

3. Side n is between nodes n and n + 1 for the surface. Side 4 is between nodes 4 and 1.

## B.7 Block Exchange Matrix File (Suffix .bni, Unit 12)

The block exchange matrix files are used to hold the exchange factor results during a run. There is one of these indexed files for each block of surfaces for which results have been calculated on this or any previous run. The block exchange matrix files are combined into a regular exchange matrix file at the end of a MONT3D run. Since load balancing has not been implemented yet, blocks are determined as follows. A list is kept of all surfaces ordered by band and by surface number within band. This list is broken sequentially into blocks of size |NEBLOCK|; see Sections 2.9 and 3.2.1 for more information. All the code for the reading and writing of this file can be found in the subroutine *nijfile* in the file *m30s.f*.

### B.7.1 Exchange Information for the Block

**|NEBLOCK| Records of Length NSURF**

**FORTRAN Format Specification: 7(i10, 1x)**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Numbers of absorbed photons emitted by surface *i* absorbed by all surfaces *j* for band *k* | 1 |

**Notes:**

1. Each record contains the number of photons absorbed by each surface *j* ranging from 1 to NSURF for the surface *i* in the band *k*. The *i* and *k* values for a block are determined using the formula mentioned above. It should be noted that the file contains results for all surfaces, even those that do not emit (which therefore have all zeroes for results). A new record (line) is started for each emitting surface.

## B.8 Temporary Exchange Matrix File (Suffix .tni, Unit 13)

The temporary exchange matrix file is used to hold the exchange matrix results when converting back and forth between the regular exchange matrix file and the block exchange matrix files. This file only exists for a short time during the input and clean up stages of a MONT3D run. This file is binary direct access file with record length 4*NSURF (since all results written to disk are 4 byte integers). All the code for the reading and writing of this file can be found in the subroutine *nijfile* in the file *m30s.f*.

### B.8.1 Photon Number Matrix

**NBAND Sets of NSURF Records Each of Length NSURF**

| Format | Entry | Note(s) |
|---|---|---|
| Integer | Numbers of absorbed photons emitted by surface *i* absorbed | |

| | | |
|---|---|---|
| | by all surfaces *j* for band *k* | 1 |

**Notes:**

1. Each record contains the number of photons absorbed by each surface *j* ranging from 1 to NSURF for each surface *i* in the band *k*. It should be noted that the file contains results for all surfaces, even those that do not emit (which therefore have all zeroes for results). A new record (line) is started for each surface. The records for all surfaces in order from 1 to NSURF in band *k* specify an exchange matrix for band *k*. The exchange matrices for the 1 to NBANDS bands are written in order to the file.

## B.9 Block File (Suffix .blk, Unit 14)

The block file contains the additional information MONT3D requires to restart from a crashed run. The file is generated at the start of a MONT3D run and deleted at the end of a successful MONT3D run. To restart from a crashed run, the block file must be present as it is now how MONT3D realizes that a restart from a crash is occurring. All the code for the reading and writing of this file can be found in the subroutine *blkfile* in the file *m313s.f*. The block file was mainly created to keep track of the surfaces in blocks for load balancing which has not been implemented yet. Without load balancing, there is not much need for the information in this file. This file contains several variables used internally by MONT3D for restart. It has little or no usefulness outside of MONT3D and requires quite a bit of explanation. If the user needs to know about this file, one of the authors should be contacted.