Radiation Heat Transfer In Enclosure Using Monte-Carlo Method

SPECIAL PROJECT
Final Report

RADIATION HEAT TRANSFER
MAE 5823

Instructor

Dr J. D. Spitler

By
Bereket Asgedom Nigusse

December 6, 2004

1. Background

Radiation heat exchange analysis between surfaces most often estimated using the net radiation exchange method, or zonal method with either analytically or numerically determined view factors. The Monte Carlo Method, which is a statistical approach, can be applied to radiation exchange analysis. In general the Monte Carlo Method is suitable for real problems that do not have closed form solutions and also when the events or processes cannot be represented by algebraic expressions. The Monte Carlo Method uses probability distribution functions to represent the system variables in a real system. Thus, any problem, if its variables can be represented by any of the probability distribution functions, then Monte Carlo Method can be used to analyze the problem. The Monte Carlo Method can handle exceptionally accurate analysis of radiation exchange including directional, spectral and variable surface property problems (Mahan, 2002). Thus, the directional, spectral and variable surface property modeling capacity of Monte Carlo Method has attracted its application in thermal radiometer applications, jet engine exhaust plume infrared emission and prediction of jet plume spectral thermal radiation field (Mahan, 2002).

The Monte Carlo Method for radiation exchange analysis application has been low due to: (1) high computational time requirements, and (2) precision level required can be attained by more simpler net radiation or zonal method. Though radiation heat exchange analysis can be easily handled using the common techniques such as: net radiation method and the zonal method, these methods are based on certain assumptions that sometimes limit their applications (Howell, 1968). These assumptions include: surfaces are gray and diffuse, uniform surface properties and uniform surface temperature.

Thus, the Monte Carlo Method can be attractive for modeling radiation analysis involving complex geometries with variable surface properties and non-uniform surface temperature applications.

1.1 Objectives

Develop a Monte Carlo Method radiation heat transfer program for a gray enclosure consisting rectangular or triangular surfaces.

2. Literature Survey

2.1 Introduction

The Monte Carlo Method, which is a statistical based radiation heat exchange analysis model, can be applied to a gray enclosure to predict radiation distribution factor. The radiation distribution factor is simply the ratio of the number of counts of energy bundles absorbed by a given surface to the total number of counts of energy bundles emitted by a surface in the enclosure (Mahan, 2002; Henda, 2004). In the Monte Carlo Method, energy bundles emitted from a given surface in an enclosure may go through a series of reflections until finally being absorbed.

The energy bundle may be absorbed or reflected at the first surface it encounters in the enclosure. Thus, Monte Carlo Method, needs to track emitted energy bundle unit absorbed. The location of the emission point of an energy bundle on a surface and it direction are determined by making use of normalized uniform distribution random numbers. Moreover, the condition for absorption or reflection of the energy by the surfaces it strikes is determined by comparing the surface properties and normalized uniform distribution random numbers. The absorptance of the surface is compared with normalized uniform distribution random numbers generated to decide whether a given emitted energy bundle is absorbed or reflected at the surface it strikes.

Repeatedly emitting several million energy bundles from a given surface and tracing the energy bundle until it is finally absorbed by any one of the surfaces in the enclosure determine radiation distribution factor. The numbers of emitted energy bundles have to be such a high number that any one additional energy bundle emission should not affect the outcome of the radiation distribution factor to the accuracy of the results desired. Once the radiation distribution factor is determined the net radiation heat flux from a given surface can be computed by writing the energy balance.

2.2 The Monte Carlo Method

In this paper, Monte Carlo Method radiation heat transfer program development will be discussed. The Monte Carlo method program developed models radiation heat exchange analysis in a gray enclosure with rectangular and triangular surfaces. The net radiation energy emitted by surface *j* and absorbed by surface *i*, Q_{ij}, in an enclosure is given by (Mahan, 2002)

$$Q_{ij} = \varepsilon_j A_j \sigma T_j^4 D'_{ji} \tag{1}$$

Where D'_{ji} is the radiation distribution factor from surface *j* to surface *i*, A_j is area of surface *j*, ε_j is emissivity of surface *j,* and T_j is the absolute temperature of surface *j*. The energy emitted by surfaces in the enclosure and absorbed by the surface *i* in the enclosure is given by summing up the absorbed terms

$$Q_{i,a} = \sum_{j=1}^{N} \varepsilon_j A_j \sigma T_j^4 D'_{ji} \tag{2}$$

Where $Q_{i,a}$(W) is the absorbed energy by surface *i*. Using the rule of reciprocity eqn. 2 becomes

$$\varepsilon_i A_i D'_{ij} = \varepsilon_j A_j D'_{ji}$$

$$Q_{i,a} = \sum_{j=1}^{N} \varepsilon_i A_i \sigma T_j^4 D'_{ij} \tag{3}$$

The net radiant energy transfer at surface *i* is the difference between the emitted and absorbed energy and is given by Mahan (2002)

$$Q_i = \varepsilon_i A_i \sigma T_i^4 - \sum_{j=1}^{N} \varepsilon_i A_i \sigma T_j^4 D'_{ij} \tag{4}$$

The net radiation heat flux at surface *i* is given by

$$\dot{q}_i = \varepsilon_i \sigma T_i^4 - \varepsilon_i \sum_{j=1}^{N} \sigma T_j^4 D'_{ij} \tag{5}$$

Computation of the net radiation heat flux from eqn.5 is straight forward if the radiation distribution factor is known. The radiation distribution factor D'_{ij} is determined by simple ratio of number of counts of energy absorbed by a surface *j* to the total number of energy bundles emitted by the surface *i* (Mahan, 2002; Henda, 2004)

$$D'_{ij} = \frac{N_{ij}}{N_i} \tag{6}$$

Monte Carlo method is used to determine radiation distribution factor. The procedure used to determine radiation distribution factor will be described next.

2.3.1 Locating Emission Point on a Surface

The location of an emitted energy bundle on a particular surface on a Cartesian coordinate system is determined using a uniformly distributed probability distribution function. The x_e, y_e and z_e coordinates of the source point are determined from geometry of the surface and two normalized uniform distribution random numbers (Howell, 1968; Mahan, 2002).

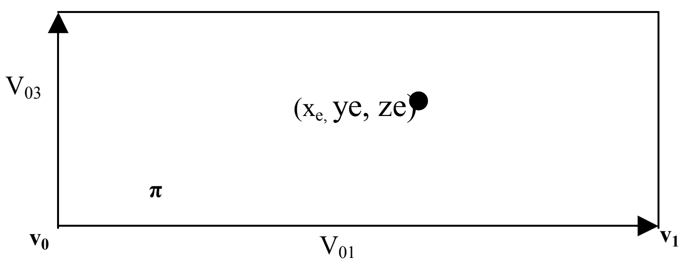


Figure-2.1 Nomenclature and representation of a rectangular surface

$$x_e = x_o + \Delta x R_x \qquad (7)$$

$$y_e = y_o + \Delta y R_y \qquad (8)$$

One of the points of the coordinates need to be determined from equation of a surface

$$Z_e = (D + Ax_e + Ay_e)/C \qquad (9)$$

Where *Δx* and *Δy* are the sides of the rectangular surfaces in *x* and *y* direction respectively, x_o, y_o and z_o are the reference base points of the global coordinate, R_x and R_y are the two normalized uniform distribution random numbers. The Monte Carlo Method program described in this paper uses built-in Fortran 90 function to generate the random numbers.

For triangular surfaces the random emission point selection follows different expression. The two random numbers must sum to less than 1.0. If the random numbers sum up to greater than 1.0 as shown in eqn. 10, then the random numbers need to be calculated by eqn. 11 (Mahan, 2002)

$$R_\alpha + R_\alpha \geq 1.0 \qquad (10)$$

$$R_\alpha = 1.0 - R_\alpha \qquad \text{and} \qquad R_\beta = 1.0 - R_\beta \qquad (11)$$

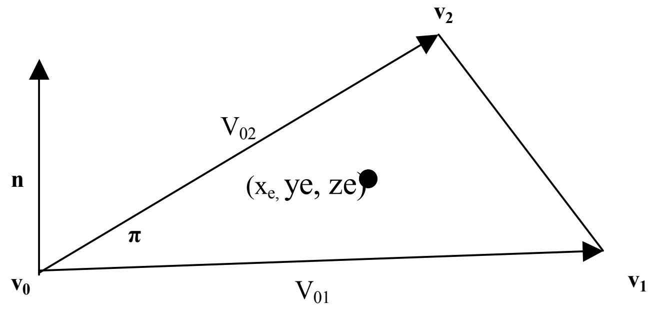


Figure-2.2 Nomenclature and representation of a triangular surface

$$x_e = x_o + R_\alpha (x_1 - x_o) + R_\beta (x_2 - x_o) \tag{12}$$

$$y_e = y_o + R_\alpha (y_1 - y_o) + R_\beta (y_2 - y_o) \tag{13}$$

Then the third coordinate needs to be determined from the surface equation as follows.

$$Z_e = (D + Ax_e + Ay_e) / C \tag{14}$$

2.3.2 Direction of The Emitted Energy Bundle

The direction of a given emitted energy bundle from a given surface is determined using normalized uniform distribution random numbers R_θ and R_Φ. The angle of departure of the emitted energy bundle is defined as the probability of the emitted energy is directed through angle $d\theta$ about θ is given by Howell (1968)

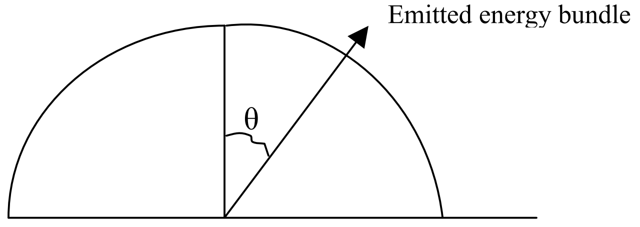


Figure-2.3 Nomenclature of the direction of emitted energy bundle

$$R_\theta = \frac{\int_0^{2\pi}\int_0^{\theta} \varepsilon I \sin\theta \cos\theta d\theta d\phi}{\varepsilon \sigma T^4} \qquad (15)$$

Where ε is the emissivity of the surface, *Φ* is the polar angle, *T* (K) is the surface temperature, *I* (W/m^2 sr) is the intensity of emitted energy bundle and σ (5.67x10^{-8} W/m^2 k^4) is the Stephan Boltzmann constant. Ignoring the polar angle dependence of angle *θ* for diffuse surfaces the eqn 15 simply reduces to (Howell, 1968)

$$R_\theta = \frac{2\pi\varepsilon\int_0^{\theta} I \sin\theta \cos\theta d\theta}{\varepsilon \sigma T^4} \qquad (16)$$

And using the definition of black body emissive power intensity

$$I = \frac{\sigma T^4}{\pi} \qquad (17)$$

Substituting eqn. (17) into eqn. (16) yields

$$R_\theta = 2\int_0^{\theta} \sin\theta \cos\theta d\theta \qquad (18)$$

Evaluating the integral yields

$$R_\theta = \sin^2\theta \qquad (19)$$

Thus, the probability of a given emitted energy bundle to be directed through an angle *θ* is given by Howell (1968; Modest, 1993)

$$\theta = \sin^{-1}(\sqrt{R_\theta}) \qquad (20)$$

The polar angle *Φ* is given by (Howell, 1968; Mahan, 2002; Haji-Sheikh, 1969; Modest, 1993)

$$\phi = 2\pi R_\phi \qquad (21)$$

These direction angles are defined relative to the two tangent vectors and the normal vector to the surface at the emission point. The direction of the emitted energy, which is defined based on the local tangent and normal vectors, needs to be transformed into the global coordinate system (Mahan, 2002). The first tangent vector is determined by cross product of the normal vector and a unit vector in either direction of the principal axis, X, Y, and Z. The unit vector can be any of the three axes direction except it should not be aligned to the normal vector (Mahan, 2002).

$$\vec{t}_1 = \vec{n} \times \vec{i} \tag{22}$$

The second tangent vector is determined by cross product of the normal vector and the first tangent vector (Mahan, 2002)

$$\vec{t}_2 = \vec{n} \times t_1 \tag{23}$$

Direction vector, $\vec{V}_e$, of the emitted energy bundle in terms of the tangent vectors t_1 and t_2 and the normal vector $\vec{n}$ is given by Mahan (2002)

$$V_{e,n} = \vec{n}\cos\theta \tag{24}$$

$$V_{e,t_1} = \vec{t}_1 \sin\theta\cos\phi \tag{25}$$

$$V_{e,t_2} = \vec{t}_2 \sin\theta\sin\phi \tag{26}$$

In terms of the global coordinate the direction of the unit vector of the emitted energy bundle becomes (Mahan, 2002)

$$V_{e,x} = (\vec{n}_x \cos\theta + \vec{t}_{1,x} \sin\theta\cos\phi + \vec{t}_{2,x} \sin\theta\sin\phi)\vec{i} \tag{27}$$

$$V_{e,y} = (\vec{n}_y \cos\theta + \vec{t}_{1,y} \sin\theta\cos\phi + \vec{t}_{2,y} \sin\theta\sin\phi)\vec{j} \tag{28}$$

$$V_{e,Z} = (\vec{n}_z \cos\theta + \vec{t}_{1,z} \sin\theta\cos\phi + \vec{t}_{2,z} \sin\theta\sin\phi)\vec{k} \tag{29}$$

The normal vector $\vec{n}$ to the surface can be determined by cross product of the edge vectors of the surface shown in figure-2.4 and is given by Mahan (2002)

$$\vec{n} = \frac{\vec{V}_{12} \times \vec{V}_{23}}{\left|\vec{V}_{12} \cdot \vec{V}_{23}\right|} \tag{30}$$

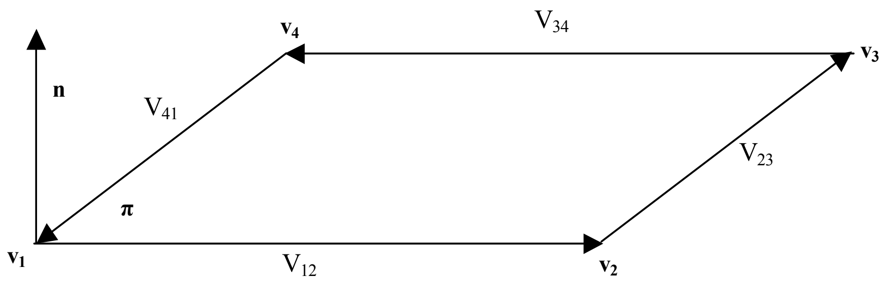


Figure-2.4 Vector notation of surface edges

2.3.3 Locating Point of Intersection

The point of intersection of the energy bundle with surfaces in the enclosure is determined by using vector algebra. Intersection between a three dimensional line and a plane containing the surface can be determined using vector algebra as follows. Consider a line ***L*** that represents the direction of the emitted energy bundle from a point source, $\mathbf{P_0}$, on one of the surfaces in the enclosure and intersects another surface ***π*** in the enclosure at point $P(S_I)$. In vector notation the line ***L*** in three-dimensional representation can be expressed in terms of the unit vector ***u*** in the direction of the emitted energy, scalar constant ***s*** and the emission source point P_0 (Sunday, http://softsurfer.com/Archive/algorithm)

$$L = P_0 + su \tag{31}$$

Where ***π*** is one of the surfaces of the enclosure, ***w*** is a vector from a point on the intercepting surface to the origin of the emitted energy vector, i.e., the energy bundle emission point. A vector position on the line of emitted energy is given by

$$P(s) = P_0 + su \tag{32}$$

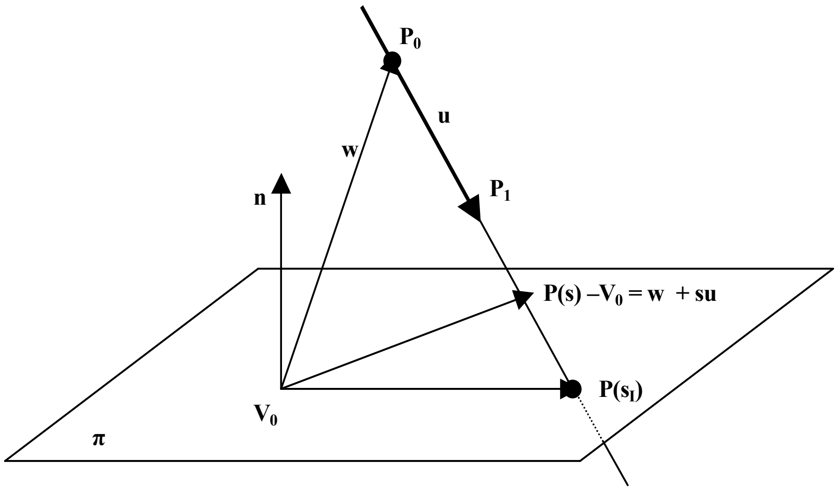


Figure-2.5 Intersection of a line and plane surface

The vector position from the reference point, **V₀**, on the intercepting surface and a point on the emitted energy line is given by vector sum of the vectors ***w*** and **P**(s)

$$P(s) - V_0 = w + su \tag{33}$$

At intersection point the vector, P(s)-V_0, which lies on the plane of surface ***π***, is perpendicular to the surface normal vector **n**. Thus, the dot product of the intercepting surface unit normal vector and the vector on the intercepting plane is zero, i.e.,

$$n \cdot (w + su) = 0 \tag{34}$$

Solving for ***s*** yields

$$s = \frac{-n \cdot w}{n \cdot u} \tag{35}$$

The vector position of the intersection point on a plane containing the surface ***π*** is given by

$$P(s_I) = P_0 + su \tag{36}$$

All intersection points on planes containing the surfaces of the enclosure can be determined as shown above; however, there can be only one intersection point. This intersection point needs to be identified from those possible intersection points. If the scalar ***s*** is negative then the intersection point is pointed on the reverse side of the emitted energy bundle direction and hence automatically eliminated. For all cases where the scalar ***s*** is greater than zero the vector algebra will be used to eliminate the intersection points outside the enclosure surfaces. If the dot product of the surface normal vector ***n***, and the emitted energy direction vector ***u*** is zero then the emitted energy bundle line is parallel to the surface; hence, there is no intersection. Moreover, if the intersection point lies within the domain of the surface then the cross product vector of a surface edge vector and a vector from a common point on the edge vector to the intersection point shall be always positive and should point in the same direction as the surface normal vector.

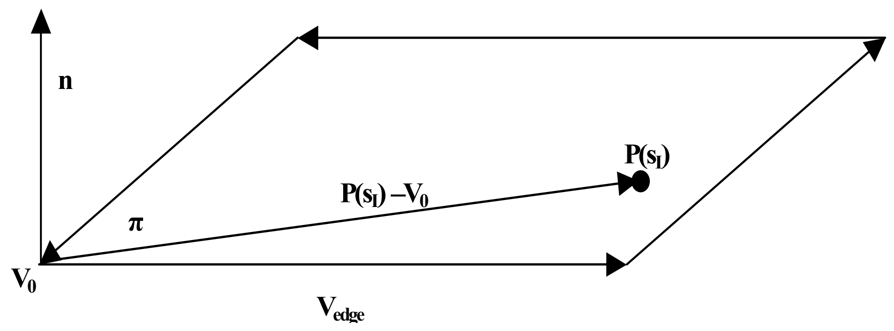


Figure-2.6 Vector notation of intersection point

2.3.4 Absorption or Reflection

To determine whether the intercepted energy has been absorbed or reflected again a normalized uniform distribution random number generated is compared with the surface absorptance by treating the surface property as a probability (Howell, 1968; Mahan, 2002). If the random number R_α is less than the absorptance, $\boldsymbol{\alpha}$, of the intercepting surface then the energy bundle is absorbed otherwise reflected (Mahan, 2002; Henda, 2004). Every time an energy bundle is absorbed the counter of that particular surface increments the number of absorbed energy bundles. If reflected for diffuse surfaces the direction of the reflected energy bundle is determined in similar way and traced until it is finally absorbed. The same process of energy bundle emitting and tracing until absorbed is repeated for every surface in the enclosure in enough number of times.

3. Methodology

The Monte Carlo Method of radiation distribution factor computation and radiation analysis program for an enclosure has been developed. The program assumes a gray enclosure. To improve the computational speed all constant parameters were calculated at the beginning of the program after having read the geometry file. These constant parameters include: Surface normal vector, surface tangent vectors, surface area and surface equation. The Monte Carlo Method radiation analysis for a gray enclosure was implemented using the program structure shown in figure-3.1. The flow chart of the program can be described step by step as follows:

1. Read the global coordinates of the enclosure geometry
2. Allocate the global variables and Initialize the energy bundle counter
3. Determine surface normal vectors, tangent vectors, surface equation, and surface area

Then Loop Calculations:

1. Determine the point of energy bundle emission
    - Triangular or Rectangular surface
2. Determine the direction of the emitted energy
3. Navigate through the enclosure to determine the intersection points
    - Triangular or Rectangular surface
    - Determine all possible Intersection Points
    - Consider only those in the front side of the energy bundle direction
    - Identify and eliminate those outside the surface domain
    - Then choose the closest point
4. Check on absorption and /reflection
    - Decide by comparing the surface absorptance with random number
5. Repeat steps 1- 4 to for all energy bundles emitted from a given surface
6. Then repeat steps 1-5 for all surfaces in the enclosure

4. Compute the radiation distribution factor for each surface in the enclosure
5. Make Energy Balance

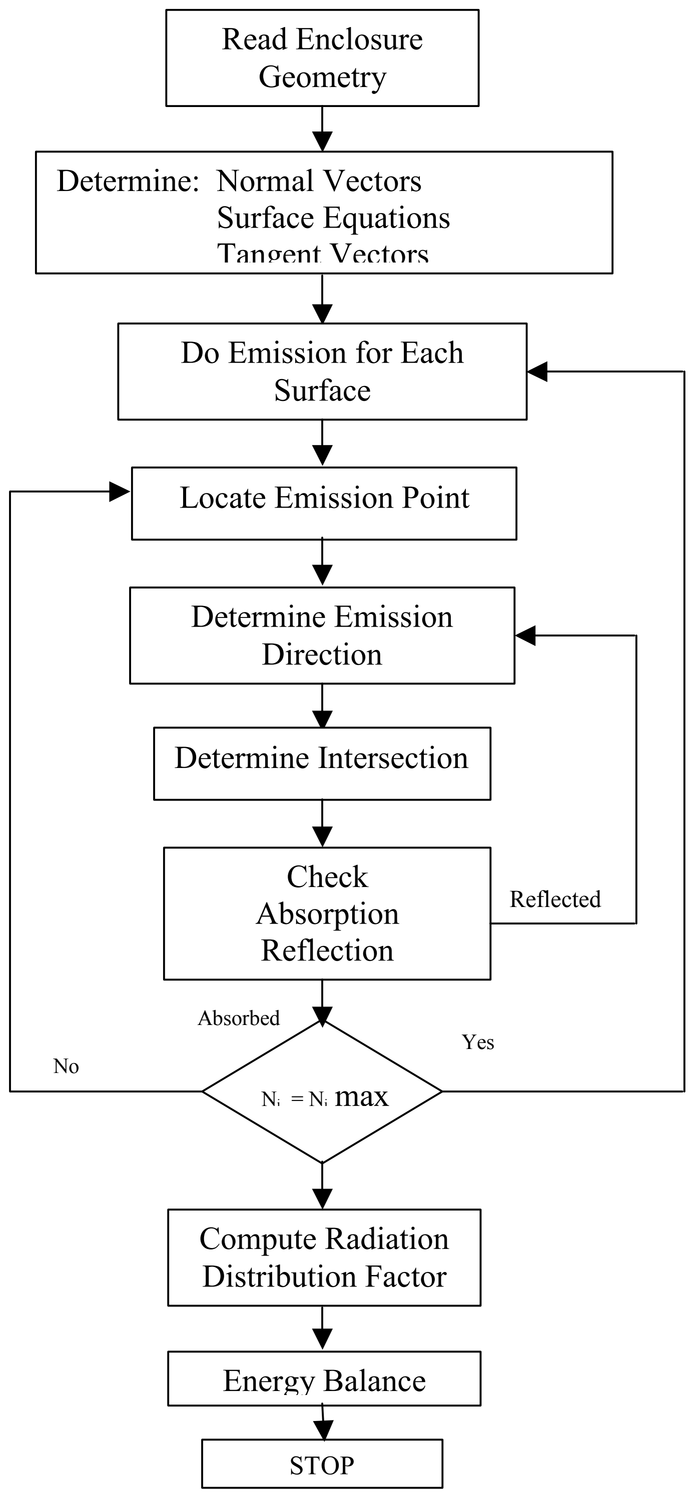


Figure-3.1 Monte Carlo Method Program Flow Chart

3.1 The Monte Carlo Program

The Fortran program developed to implement Monte Carlo Method has the following modeling capacity.

- Computes the view factors of a gray enclosure.
- The surfaces must be rectangular or triangular polygon
- Handles attic and L-shaped enclosures
- Calculates the heat flux and heat transfer at the surfaces

3.2 Program Requirements

The Monte Carlo Method program uses the same input file format as that of Walton's program. The vertices of the surfaces shall be defined in counterclockwise order when viewed from inside the enclosure. The emissivities will be read form the input file and the surface temperatures need to be supplied in a separate file with same format to that of Walton's program. The temperatures must be specified in absolute temperature scale.

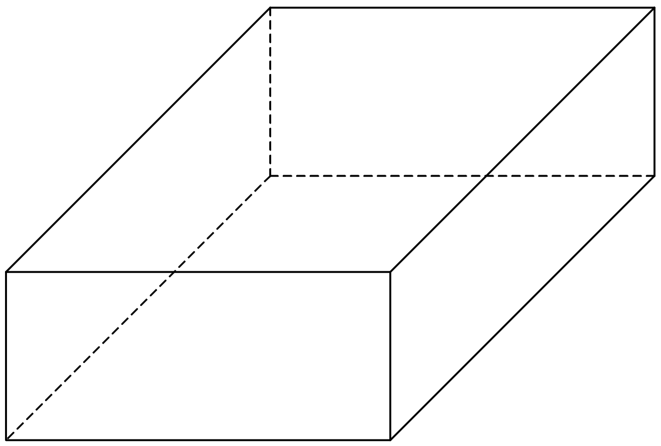
Temperature Input File: = 'xxx.TK'
Geometry Input File: = 'xxx.vs3'

3.3 Test Samples

The developed program has been tested for three different enclosures: a rectangular surfaces enclosure with geometry of 6 m long, 4 m wide and 3 m high, a barn type gray enclosure with rectangular and triangular surfaces, and L-shaped gray enclosure with rectangular surfaces. The input files of the enclosures in the Walton's input file format are shown next. However, the first three lines of the file will be removed. The first test sample is the simplest test case. The view factor calculated for this particular enclosure with emissivities of all surfaces set to 1.0 yields the "black body" view factors.

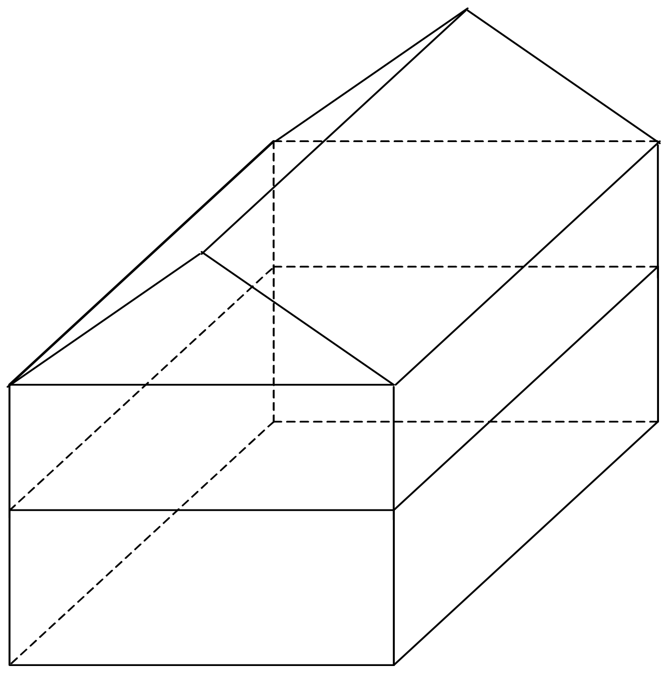
Test Sample 1 Input File

```
T Rectangular Box
C encl=1 out=1 list=2 emit=0
!     #      x      y      z      coordinates of vertices
v     1     0.0    0.0    0.0
v     2     6.0    0.0    0.0
v     3     6.0    4.0    0.0
v     4     0.0    4.0    0.0
v     5     0.0    0.0    3.0
v     6     6.0    0.0    3.0
v     7     6.0    4.0    3.0
v     8     0.0    4.0    3.0
!     #     v1     v2     v3     v4     base  cmb   emit  names
s     1     5      8      7      6      0     0     1.0   Roof
s     2     1      2      3      4      0     0     1.0   Floor
s     3     1      4      8      5      0     0     1.0   WestW
s     4     3      2      6      7      0     0     1.0   EastW
s     5     4      3      7      8      0     0     1.0   NorthW
s     6     1      5      6      2      0     0     1.0   SothW
End of data
```



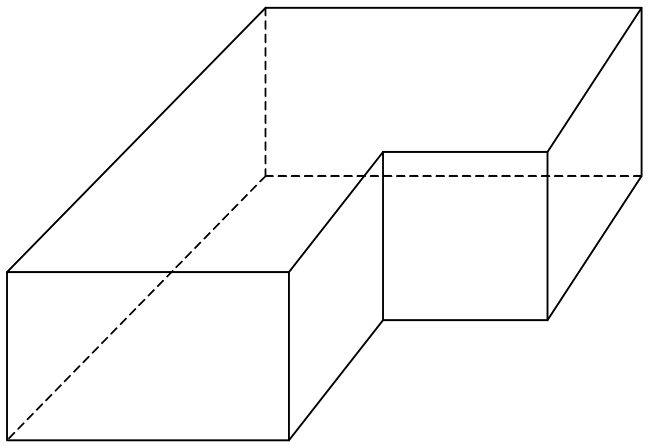
Test Sample 2 Input File

```
T Barn Geometry
C encl=1 out=0 list=2 emit=0
!  #   x      y      z      coordinates of vertices
v  1   0.    0.0   0.0
v  2   0.    0.0   6.2
v  3   0.0   0.0   9.4
v  4   5.0   0.0  10.6
v  5  10.0   0.0   9.4
v  6  10.    0.0   6.2
v  7  10.    0.0   0.0
v  8   0.   17.0   0.0
v  9   0.   17.0   6.2
v 10   0.0  17.0   9.4
v 11   5.   17.0  10.6
v 12  10.0  17.0   9.4
v 13  10.   17.0   6.2
v 14  10.   17.0   0.0
!  #   v1   v2 v3  v4  base cmb emit names
s  1    2    9  10   3   0   0   0.9  srf?1
s  2    3   10  11   4   0   0   0.9  srf?2
s  3   12    5   4  11   0   0   0.9  srf?3
s  4   13    6   5  12   0   0   0.9  srf?4
s  5   14    7   6  13   0   0   0.9  srf?5
s  6    7   14   8   1   0   0   0.9  srf?6
s  7    1    8   9   2   0   0   0.9  srf?7
s  8    7    1   2   6   0   0   0.9  srf?8
s  9    8   14  13   9   0   0   0.9  srf?9
s 10    6    2   3   5   0   8   0.9  srf?7b   ! surfaces that combine
s 11    5    3   4   0   0   8   0.9  srf?7c   ! surfaces that combine
s 12    9   13  12  10   0   9   0.9  srf?7b   ! surfaces that combine
s 13   10   12  11   0   0   9   0.9  srf?7c   ! surfaces that combine
End of data
```



Test Sample 3 Input File

```
T LSHAPE
C  encl=1 list=1
F  3
v     1     0.0   0.0   0.0
v     2     4.0   0.0   0.0
v     3     4.0   3.0   0.0
v     4     7.0   3.0   0.0
v     5     7.0   6.0   0.0
v     6     0.0   6.0   0.0
v     7     0.0   3.0   0.0
v     8     0.0   0.0   3.0
v     9     4.0   0.0   3.0
v    10     4.0   3.0   3.0
v    11     7.0   3.0   3.0
v    12     7.0   6.0   3.0
v    13     0.0   6.0   3.0
v    14     0.0   3.0   3.0
!     #     v1    v2    v3    v4    base  cmb   emit  names
s     1     1     8     9     2     0     0     0.9   South1
s     2     2     9    10     3     0     0     0.9   East1
s     3     3    10    11     4     0     0     0.9   South2
s     4     4    11    12     5     0     0     0.9   East2
s     5     5    12    13     6     0     0     0.9   North
s     6     1     6    13     8     0     0     0.9   West
s     7    14    13    12    11     0     0     0.9   CeilingNorth
s     8     7     4     5     6     0     0     0.9   FloorNorth
s     9     8    14    10     9     0     0     0.9   CeilingSouth
s     10    1     2     3     7     0     0     0.9   FloorSouth
End of file
```



4. Results and Discussion

The test sample view factors and surfaces heat fluxes were calculated using the Monte Carlo Method and Walton's program. The accuracy of the Monte Carlo method depends on the number of energy bundles emitted and theoretically it gives accurate results when the number of energy bundles emitted is infinite (Howell, 1968; Mahan, 2002; Modest, 1993). This will be demonstrated through the three test sample enclosures.

4.1 Results of Test Sample-1

The rectangular enclosure test sample–1 with all surfaces in the enclosure being black is the simplest case for Monte Carlo Method, as it does not involve any reflection. Any emitted energy is absorbed at the first surfaces where it strikes. The "black body" view factor calculated using Monte Carlo Method and Walton's program is shown in table-4.1 and table-4.2. The results show very good agreement for higher number of energy bundles emitted.

Table-4.1 Black Body view factors calculated using Walton's program for Test sample -1

	"Black Body" View Factors Using Walton's Program					
	1	2	3	4	5	6
1	0	0.341694	0.129616	0.129616	0.199537	0.199537
2	0.341694	0	0.129616	0.129616	0.199537	0.199537
3	0.259232	0.259232	0	0.087105	0.197215	0.197215
4	0.259232	0.259232	0.087105	0	0.197215	0.197215
5	0.266049	0.266049	0.131477	0.131477	0	0.204948
6	0.266049	0.266049	0.131477	0.131477	0.204948	0

Table-4.2 Black Body view factors calculated using Monte Carlo Method for Test sample -1

	"Black Body" Factors Using Monte Carlo Method (1 Million Energy Bundles Emitted)					
	1	2	3	4	5	6
1	0	0.340802	0.129843	0.129525	0.199311	0.200519
2	0.341911	0	0.129357	0.129870	0.199489	0.199373
3	0.259285	0.258661	0	0.086789	0.198010	0.197255
4	0.259161	0.259091	0.086732	0	0.197477	0.197539
5	0.265724	0.266281	0.131383	0.131578	0	0.205034
6	0.266182	0.265705	0.131388	0.131745	0.204980	0

Table-4.3 Relative error of "Black Body" view factors of Monte Carlo Method for Test sample -1

	Difference Between Walton's Program and Monte Carlo Method					
	1	2	3	4	5	6
1	0.0	0.00089	-0.00023	0.00009	0.00023	-0.00098
2	-0.00022	0.0	0.00026	-0.00025	0.00005	0.00016
3	-0.00005	0.00057	0.0	0.00032	-0.00079	-0.00004
4	0.00007	0.00014	0.00037	0.0	-0.00026	-0.00032
5	0.00032	-0.00023	0.00009	-0.00010	0.0	-0.00009
6	-0.00013	0.00034	0.00009	-0.00027	-0.00003	0.0

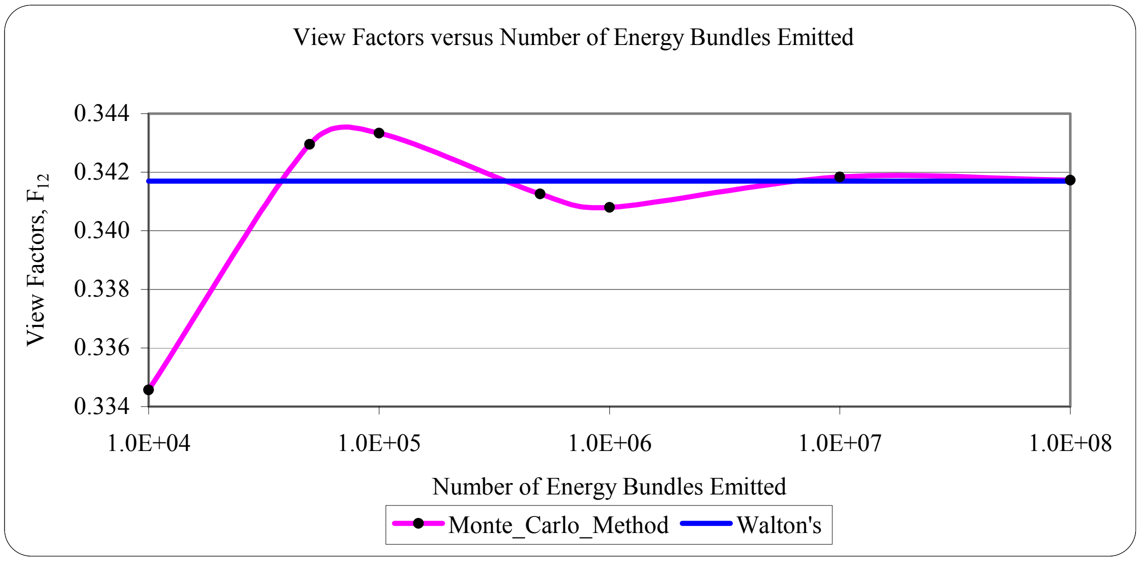


Figure-4.1 "Black Body" View Factor F_{12} versus number of energy bundles emitted

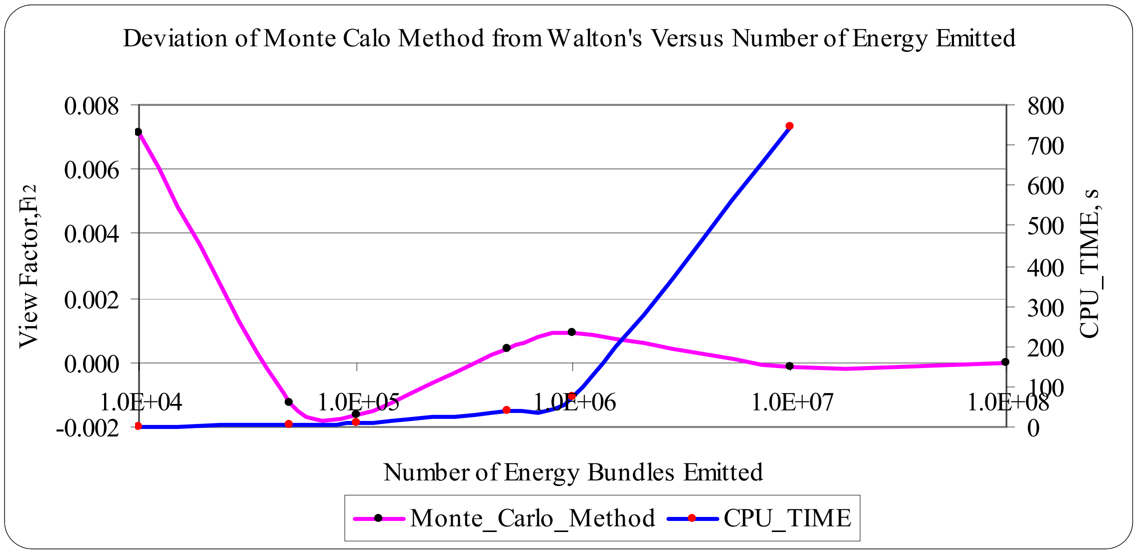


Figure-4.2 "Black Body" View Factor F_{12} error versus number of energy bundles emitted

For the rectangular surfaces gray enclosure, test sample-1, with all the surfaces assigned emissivities of 0.9, the view factors F_{12} and F_{16} convergence trend as a function of number of energy bundles emitted are shown in figure-4.3 and figure-4.4, respectively. As can be observed from graph the convergence rate for the gray surface is low compared to the black body enclosure due to the repeated reflections that happens in the case of gray surfaces.

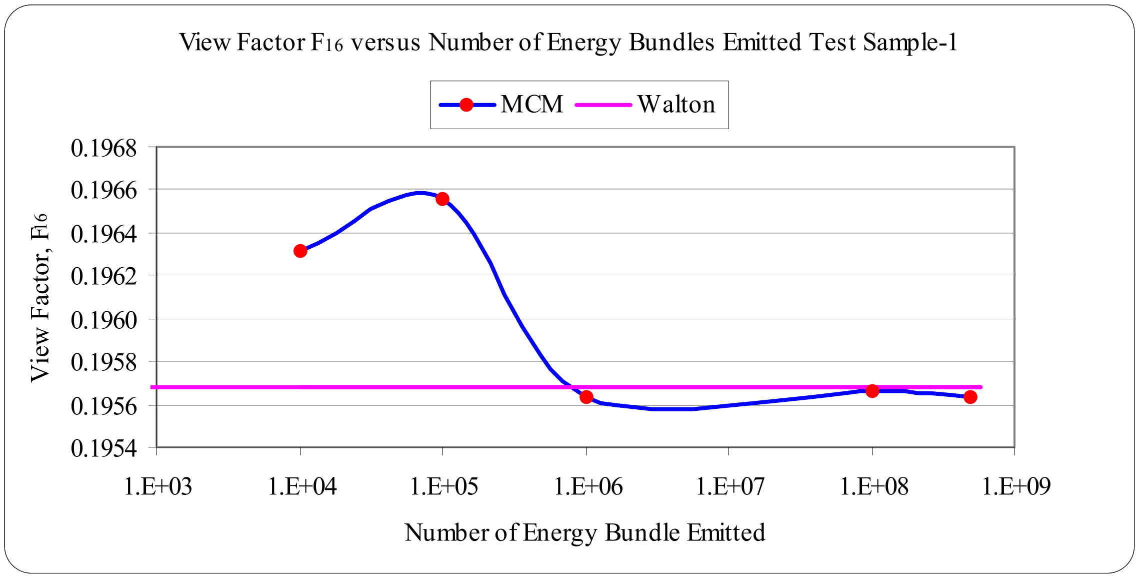


Figure-4.3 “Gray” View Factor F_{16} error versus number of energy bundles emitted

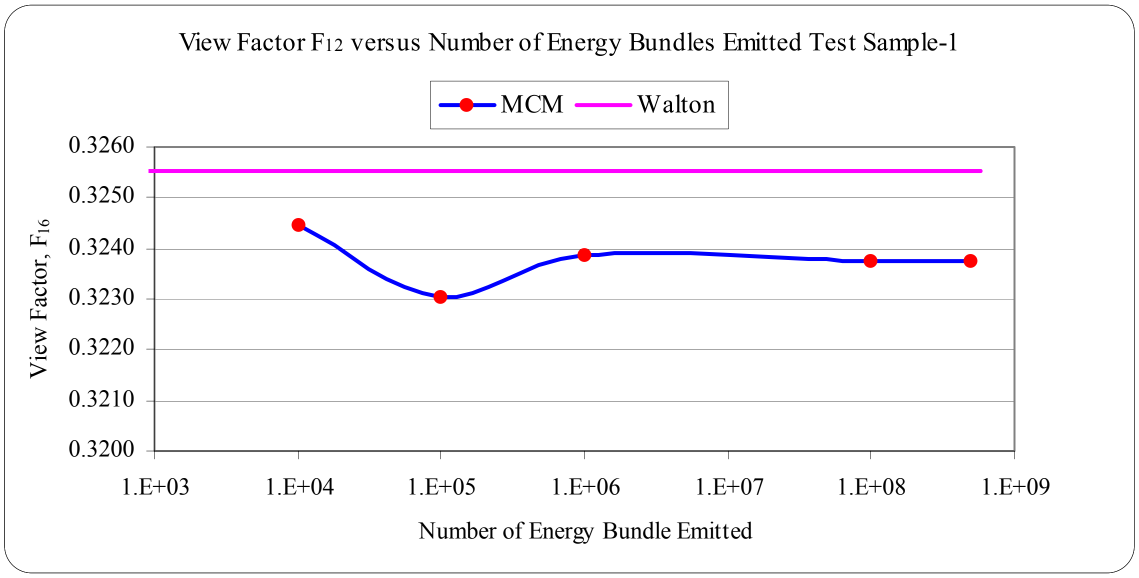


Figure-4.4 “Gray” View Factor F_{12} error versus number of energy bundles emitted

4.2 Results of Test Sample-2

The Test sample-2 is barn type enclosure made up of rectangular and triangular gray surfaces. For this test sample the gray view factors and the heat fluxes for each surfaces has been computed using Monte Carlo Method and Walton's Program. The view factors for the test sample are shown in table-4.6 through table-4.8. The view factors agree mostly up to three decimal places and in some cases only to two decimal places only. The heat fluxes $\dot{q}_{i,flux}$ calculated at each surface shows reasonable agreement with those calculated using Walton's program. Three thousand intersection points on one of the triangular surfaces in the Test Sample 3 enclosure is shown in figure-4.5.

Table-4.4 Heat Flux calculated for the test sample-2 (100 Millions Energy Bundle Emitted)

Surfaces	Surface Area m^2	Surface Temperature, K	q_{flux} (Walton's) W/m^2	q_{flux} (MCM) W/m^2	Δq_{flux} W/m^2
1	54.4	318.15	51.74	51.07	-0.67
2	87.4	320.15	77.41	76.99	-0.42
3	87.4	312.15	26.80	26.89	0.09
4	54.4	310.15	-3.32	-3.54	-0.21
5	105.0	307.15	-9.94	-10.06	-0.12
6	170.0	301.15	-56.90	-56.75	0.15
7	105.0	314.15	44.56	44.53	-0.03
8	62.0	306.15	-15.69	-15.39	0.30
9	62.0	304.15	-28.23	-27.88	0.35
10	32.0	306.15	-32.32	-32.03	0.29
11	6.0	306.15	-37.57	-37.23	0.34
12	32.0	304.15	-44.85	-44.50	0.35
13	6.0	304.15	-50.06	-49.58	0.48

Table-4.5 CPU Time for the two methods for Test Sample 2

Number of Emitted Energy Bundles	CPU TIME, s	
	Walton Program	Monte Carlo Method
10^6	0.1	70.1
10^7	0.1	700.8
10^8	0.1	6961.7

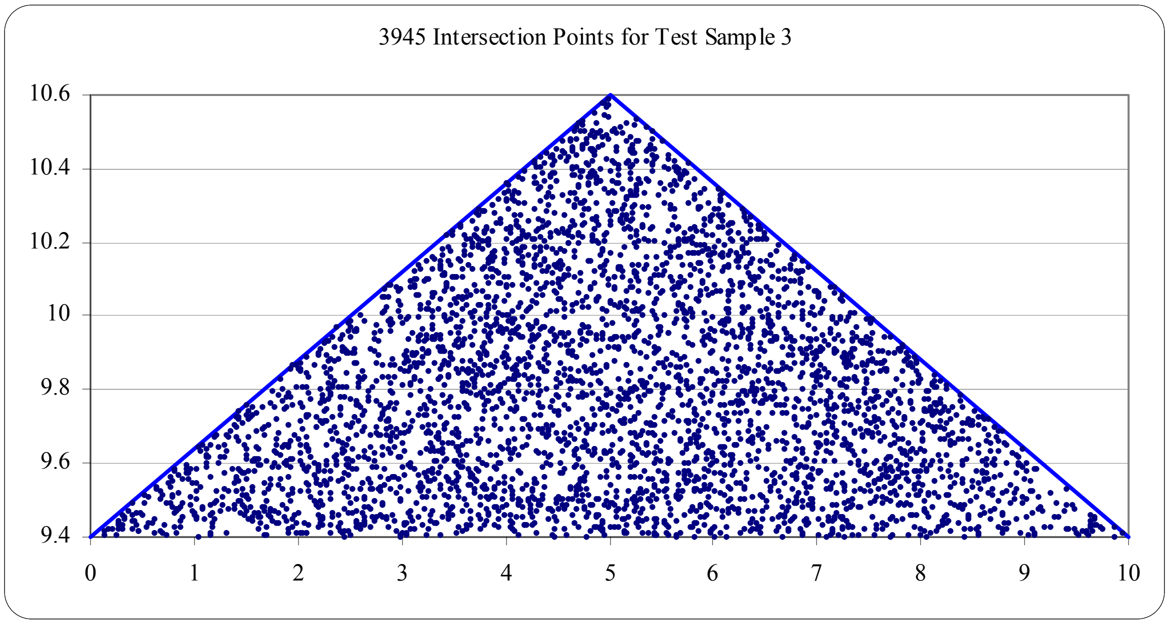


Figure4.5 Intersection Points on a triangular surface for Test Sample 3

Table-4.6 Gray view factors of test sample-2 using Walton's Program

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0.009195	0.222887	0.115088	0.099131	0.139796	0.137246	0.012970	0.050916	0.050916	0.071524	0.009403	0.071524	0.009403
2	0.138731	0.013637	0.034048	0.071634	0.118321	0.259493	0.086961	0.060607	0.060607	0.061312	0.016746	0.061312	0.016746
3	0.071634	0.034048	0.013637	0.138731	0.086961	0.259493	0.118321	0.060607	0.060607	0.061312	0.016746	0.061312	0.016746
4	0.099131	0.115088	0.222887	0.009195	0.012970	0.137246	0.139796	0.050916	0.050916	0.071524	0.009403	0.071524	0.009403
5	0.072428	0.098488	0.072385	0.006720	0.015843	0.288242	0.180038	0.104961	0.104961	0.026393	0.003479	0.026393	0.003479
6	0.043919	0.133410	0.133410	0.043919	0.178032	0.025611	0.178032	0.100975	0.100975	0.026721	0.004138	0.026721	0.004138
7	0.006720	0.072385	0.098488	0.072428	0.180038	0.288242	0.015843	0.104961	0.104961	0.026393	0.003479	0.026393	0.003479
8	0.044675	0.085438	0.085438	0.044675	0.177757	0.276867	0.177757	0.008732	0.061636	0.003654	0.000657	0.027958	0.004757
9	0.044675	0.085438	0.085438	0.044675	0.177757	0.276867	0.177757	0.061636	0.008732	0.027958	0.004757	0.003654	0.000657
10	0.121591	0.167459	0.167459	0.121591	0.086602	0.141958	0.086602	0.007080	0.054169	0.005050	0.000991	0.033174	0.006275
11	0.085252	0.243936	0.243936	0.085252	0.060877	0.117237	0.060877	0.006790	0.049157	0.005283	0.001170	0.033468	0.006764
12	0.121591	0.167459	0.167459	0.121591	0.086602	0.141958	0.086602	0.054169	0.007080	0.033174	0.006275	0.005050	0.000991
13	0.085252	0.243936	0.243936	0.085252	0.060877	0.117237	0.060877	0.049157	0.006790	0.033468	0.006764	0.005283	0.001170

Table-4.7 Gray view factors of test sample-2 using Monte Carlo Method for 100 million energy bundles emitted from each surface

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0.011991	0.223941	0.114413	0.098007	0.139715	0.136722	0.012344	0.050761	0.050784	0.071415	0.009234	0.071450	0.009225
2	0.139350	0.015384	0.033381	0.071175	0.117691	0.258869	0.087485	0.060307	0.060254	0.061316	0.016772	0.061239	0.016776
3	0.071207	0.033426	0.015401	0.139370	0.087495	0.258804	0.117651	0.060268	0.060279	0.061297	0.016775	0.061250	0.016777
4	0.098023	0.114403	0.223862	0.011981	0.012348	0.136835	0.139673	0.050807	0.050762	0.071438	0.009224	0.071435	0.009209
5	0.072094	0.097628	0.072548	0.006373	0.019494	0.287308	0.176810	0.104298	0.104231	0.026185	0.003420	0.026192	0.003419
6	0.043763	0.133069	0.133097	0.043738	0.178154	0.027458	0.178070	0.100825	0.100840	0.026455	0.004030	0.026458	0.004043
7	0.006373	0.072547	0.097631	0.072086	0.176841	0.287255	0.019519	0.104274	0.104264	0.026203	0.003418	0.026166	0.003423
8	0.044580	0.085000	0.084992	0.044597	0.177189	0.276393	0.177248	0.016597	0.057282	0.004410	0.000580	0.026615	0.004518
9	0.044606	0.085005	0.084998	0.044579	0.177222	0.276387	0.177262	0.057202	0.016628	0.026628	0.004500	0.004396	0.000586
10	0.121407	0.167337	0.167438	0.121403	0.086244	0.140473	0.086268	0.008538	0.051607	0.011385	0.002002	0.030288	0.005612
11	0.083570	0.244291	0.244363	0.083577	0.060069	0.114409	0.060108	0.006017	0.046537	0.010641	0.010518	0.030024	0.005876
12	0.121482	0.167339	0.167412	0.121380	0.086233	0.140506	0.086283	0.051605	0.008523	0.030245	0.005623	0.011367	0.002002
13	0.083570	0.244349	0.244400	0.083604	0.060022	0.114365	0.060053	0.046515	0.006020	0.030036	0.005885	0.010654	0.010528

Table-4.8 Relative Monte Carlo Method for 100 million energy bundles emitted from each surface for test sample-2

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-0.002796	-0.001054	0.000675	0.001124	0.000081	0.000524	0.000626	0.000155	0.000132	0.000109	0.000169	0.000074	0.000178
2	-0.000619	-0.001747	0.000667	0.000459	0.000630	0.000624	-0.000524	0.000300	0.000353	-0.000004	-0.000026	0.000073	-0.000030
3	0.000427	0.000622	-0.001764	-0.000639	-0.000534	0.000689	0.000670	0.000339	0.000328	0.000015	-0.000029	0.000062	-0.000031
4	0.001108	0.000685	-0.000975	-0.002786	0.000622	0.000411	0.000123	0.000109	0.000154	0.000086	0.000179	0.000089	0.000194
5	0.000334	0.000860	-0.000163	0.000347	-0.003651	0.000934	0.003228	0.000663	0.000730	0.000208	0.000059	0.000201	0.000060
6	0.000156	0.000341	0.000313	0.000181	-0.000122	-0.001847	-0.000038	0.000150	0.000135	0.000266	0.000108	0.000263	0.000095
7	0.000347	-0.000162	0.000857	0.000342	0.003197	0.000987	-0.003676	0.000687	0.000697	0.000190	0.000061	0.000227	0.000056
8	0.000095	0.000438	0.000446	0.000078	0.000568	0.000474	0.000509	-0.007865	0.004354	-0.000756	0.000077	0.001343	0.000239
9	0.000069	0.000433	0.000440	0.000096	0.000535	0.000480	0.000495	0.004434	-0.007896	0.001330	0.000257	-0.000742	0.000071
10	0.000184	0.000122	0.000021	0.000188	0.000358	0.001485	0.000334	-0.001458	0.002562	-0.006335	-0.001011	0.002886	0.000663
11	0.001682	-0.000355	-0.000427	0.001675	0.000808	0.002828	0.000769	0.000773	0.002620	-0.005358	-0.009348	0.003444	0.000888
12	0.000109	0.000120	0.000047	0.000211	0.000369	0.001452	0.000319	0.002564	-0.001443	0.002929	0.000652	-0.006317	-0.001011
13	0.001682	-0.000413	-0.000464	0.001648	0.000855	0.002872	0.000824	0.002642	0.000770	0.003432	0.000879	-0.005371	-0.009358

Table-4.9 Number of Energy Bundles emitted and absorbed for test sample-2

	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
1	107846	2013856	1030980	881946	1257873	1230036	111201	456008	456658	641726	82901	645279	82997	8999307
2	1252231	138488	301058	639652	1061364	2329707	785952	543545	542495	551833	151393	550535	151360	8999613
3	640535	301113	138865	1252905	787054	2330596	1058947	542920	541213	551083	151574	551592	151516	8999913
4	882137	1031003	2013475	108093	111000	1230807	1258354	457356	457835	642290	83154	643085	82755	9001344
5	649825	878604	654080	57281	175788	2584789	1591960	937741	938339	235035	30458	235224	30865	8999989
6	394824	1197794	1197730	393983	1602302	247335	1601294	909241	906392	238077	36559	238178	36230	8999939
7	57403	654453	878460	648465	1591858	2584458	175429	938134	938495	235095	31036	235765	30785	8999836
8	401855	762897	765761	401651	1594608	2487397	1595651	149395	514113	39582	5151	239942	40443	8998446
9	401186	765096	764996	401822	1596418	2486100	1596123	514750	149494	239220	40429	39791	5277	9000702
10	1094731	1503267	1506738	1092942	776017	1264701	777030	76430	464575	102491	18137	271994	50525	8999578
11	752965	2201648	2198345	750899	540976	1028919	541326	54259	418846	95464	94824	269667	52920	9001058
12	1093373	1505404	1506867	1092409	775490	1264590	777243	462854	76746	272753	50631	102743	18185	8999288
13	753536	2199123	2200427	752651	540459	1028132	540559	418398	54065	269856	53155	95897	94571	9000829

4.3 Results of Test Sample-3

The Test sample-3 is L-shaped black surfaces enclosure made up of rectangular surfaces. For this test sample the radiation distribution factors have been computed using Monte Carlo Method and Walton's Program. The view factors for the test sample are shown in table-4.12 through table-4.14. The view factors calculated using Walton's and Monte Carlo Method show good agreement. The fluxes also show good agreement.

Table-4.10 Heat Flux calculated for the test sample-2 (1 Million Energy Bundle Emitted)

Surfaces	Surface Area m^2	Surface Temperature, K	q_{flux} (Walton's) W/m^2	q_{flux} (MCM) W/m^2	Δq_{flux} W/m^2
1	12.0	318.15	75.66	75.09	0.57
2	9.0	320.15	88.73	88.56	0.17
3	9.0	312.15	20.46	20.57	-0.11
4	9.0	310.15	6.63	6.33	0.30
5	21.0	307.15	-16.86	-17.13	0.27
6	18.0	301.15	-59.67	-59.64	-0.03
7	21.0	314.15	46.40	46.57	-0.17
8	21.0	306.15	-24.08	-23.81	-0.27
9	12.0	304.15	-37.93	-37.62	-0.31
10	12.0	304.15	-44.63	-44.32	-0.31

Table-4.11 CPU Time for the two methods for Test Sample 3

Number of Emitted Energy Bundles	CPU TIME, s	
	Walton Program	Monte Carlo Method
10^6	0.05	40.73
10^7	0.05	407.92

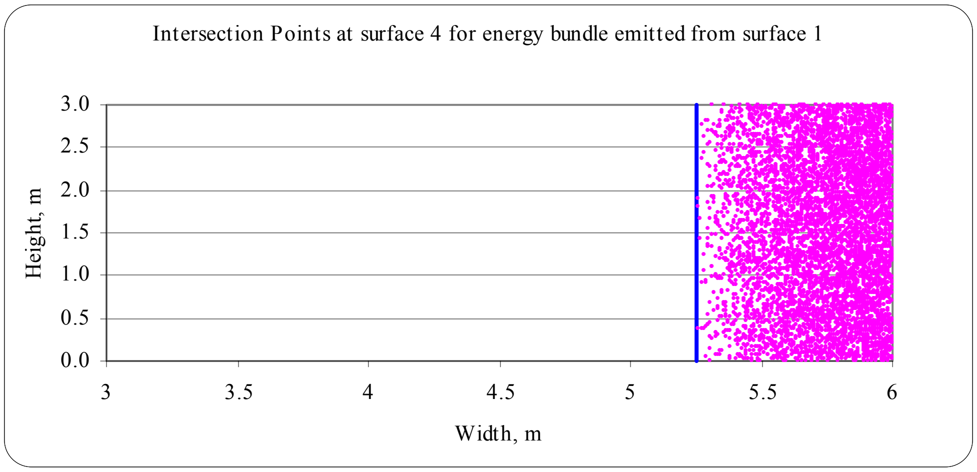


Figure-4.6 Intersection point on surface 4 for a million energy bundle emitted at surface 1

Table-4.12 “Black Body” view factors of test sample-3 using Walton’s Program

	1	2	3	4	5	6	7	8	9	10
1	0	0.162566	0	0.000639	0.112043	0.197200	0.045048	0.045048	0.218728	0.218728
2	0.216754	0	0	0	0.046186	0.204937	0.049277	0.049277	0.216784	0.216784
3	0	0	0	0.200042	0.293876	0.020163	0.24296	0.24296	0	0
4	0.000852	0	0.200042	0	0.236942	0.076218	0.236939	0.236939	0.006034	0.006034
5	0.064025	0.019794	0.125947	0.101547	0	0.132556	0.247513	0.247513	0.030554	0.030554
6	0.131467	0.102468	0.010081	0.038109	0.154648	0	0.148580	0.148580	0.133033	0.133033
7	0.025742	0.021119	0.104126	0.101545	0.247513	0.127354	0	0.301917	0	0.070685
8	0.025742	0.021119	0.104126	0.101545	0.247513	0.127354	0.301917	0	0.070685	0
9	0.218728	0.162588	0	0.004525	0.053469	0.199550	0	0.123699	0	0.237441
10	0.218728	0.162588	0	0.004525	0.053469	0.199550	0.123699	0	0.237441	0

Table-4.13 “Black Body” view factors of test sample-3 using Monte Carlo Method for one Million energy bundles emitted from each surface

	1	2	3	4	5	6	7	8	9	10
1	0	0.163036	0	0.000613	0.11163	0.197263	0.044721	0.045100	0.218241	0.219396
2	0.216043	0	0	0	0.046353	0.205451	0.049279	0.049130	0.216556	0.217188
3	0	0	0	0.200261	0.293993	0.020281	0.242660	0.242805	0	0
4	0.000776	0	0.199848	0	0.236924	0.076265	0.236851	0.237154	0.006083	0.006099
5	0.063923	0.019896	0.126006	0.101759	0	0.133011	0.246711	0.247441	0.030437	0.030816
6	0.131013	0.102932	0.010069	0.038071	0.155211	0	0.148773	0.148578	0.132247	0.133106
7	0.025795	0.021139	0.103418	0.101265	0.247874	0.127468	0	0.302460	0	0.070581
8	0.025409	0.021205	0.104508	0.101073	0.247530	0.126932	0.302917	0	0.070426	0
9	0.217941	0.162748	0	0.004611	0.053628	0.199354	0	0.123452	0	0.238266
10	0.217880	0.163191	0	0.004467	0.053133	0.199744	0.123558	0	0.238027	0

Table-4.14 Relative error of Monte Carlo Method for 1000000 energy bundles emitted from each surface for test sample-3

	1	2	3	4	5	6	7	8	9	10
1	0	-0.00047	0	0.000026	0.000413	-6.3E-05	0.000327	-5.2E-05	0.000487	-0.000668
2	0.000711	0	0	0	-0.000167	-0.000514	-2E-06	0.000147	0.000228	-0.000404
3	0	0	0	-0.000219	-0.000117	-0.000118	0.0003	0.000155	0	0
4	0.000076	0	0.000194	0	1.8E-05	-4.7E-05	8.8E-05	-0.000215	-4.9E-05	-0.000065
5	0.000102	-0.000102	-5.9E-05	-0.000212	0	-0.000455	0.000802	7.2E-05	0.000117	-0.000262
6	0.000454	-0.000464	1.2E-05	3.8E-05	-0.000563	0	-0.000193	2E-06	0.000786	-7.3E-05
7	-5.3E-05	-2E-05	0.000708	0.00028	-0.000361	-0.000114	0	-0.000543	0	0.000104
8	0.000333	-8.6E-05	-0.000382	0.000472	-1.7E-05	0.000422	-0.001	0	0.000259	0
9	0.000787	-0.00016	0	-8.6E-05	-0.000159	0.000196	0	0.000247	0	-0.000825
10	0.000848	-0.000603	0	5.8E-05	0.000336	-0.000194	0.000141	0	-0.000586	0

Conclusion

The following conclusions can be made from the observations of the simulation results:

- The Monte Carlo Method can be used to determine the “Black Body” or “Gray” view factors for an enclosure.
- Accuracy of the Monte Carlo Method depends on the number of energy bundles emitted.
- The computational time increases with the number of energy bundles emitted and the number of surfaces in an enclosure. For thirteen surfaces gray enclosure for one million and hundred million energy bundles emitted from each surface the computational CPU time were 0.0195 hrs (70.2 s) and 1.93 hrs (6961.67 s), respectively.
- Due to the high computational time it is not recommended for thermal radiation exchange analysis applications such as building radiation analysis.

References:

Howell, J.R. (1968). *Applications of Monte Carlo to heat transfer problems*. In Advances in Heat Transfer, Vol. 15, Academic press, New York, pp 1-50.

Haji-Sheikh, A. and E. M. Sparrow, *Probability distribution and error estimates for Monte Carlo solutions of radiation problems*. In progress in Heat and Mass Transfer, Vol. 2, Pergamon Press, Oxford, 1969, PP. 1-12

Henda, R. (2004). *Computer evaluation of exchange factors*. Chemical Engineering Education. Vol. (38), No. 2, pp 126-131

Kreyszing, E. "*Advanced Engineering Mathematics.*" 8th edition, John Wiley& Sons, Inc, New York, 1999.

Mahan, J. R. "*Radiation Heat Transfer: A Statistical Approach*." John Wiley & Sons, New York, 2002

Modest, M. F. "*Radiative Heat Transfer*". , New York: McGraw-Hill, Inc., 1993.

Sunday, D. (http://softsurfer.com/Archive/algorithm_0104/algorithm_0104B.htm)

Walton, G.N. (2002). *View3D Program*. Building Environmental Division, National Institute of Standards and Technology, Gaithersburg, USA.

APPENDIX

Monte Carlo Method Computer Program

Fortran 90

```
MODULE Global
!           OKlahoma State University
!           School of Mechanical And Aerospace Engineering
!
!
!           PURPOSE:  Global Data for Program Monte Carlo Method
!
IMPLICIT NONE
SAVE

INTEGER, PARAMETER :: Prec = SELECTED_REAL_KIND(P = 12)
INTEGER, PARAMETER :: Prec2 = SELECTED_REAL_KIND(P = 12)

INTEGER :: out                              ! Unit Number for Output file
INTEGER :: In                               ! Unit number for inout file

INTEGER :: NSurf                            ! Number of Surfaces
INTEGER :: NSurfcmb                         ! Number of Surfaces after combination
INTEGER :: NTrials                                  ! Number of Trials
INTEGER :: SIndex                                   ! Surface counting Index
INTEGER :: SIndexR                                  ! Surface counting Index Reference
INTEGER :: NVertex                                  ! Number of Vertices
INTEGER         :: NBundles                         ! Number of Energy Bundles Emitted
INTEGER         :: REF_IND                          ! One Reflection or rereflection index, 0 reflected or
                                                    ! 1 rereflected
INTEGER         :: N_SCMB                           ! Number of surfaces combined in the enclosure
!
!
INTEGER , ALLOCATABLE,DIMENSION(:,:) :: SVertex                 ! Vertices of A surface
INTEGER , ALLOCATABLE,DIMENSION(:)  :: SNumber                  ! Index of a surface
INTEGER , ALLOCATABLE,DIMENSION(:)  :: V                        ! vertex Index
INTEGER , ALLOCATABLE,DIMENSION(:)  :: SPlane           ! Plane of a Surface (x,y,z)
INTEGER         ,           :: SInter                   ! Index of Intercepted Surface
INTEGER , ALLOCATABLE,DIMENSION(:,:) :: NAEnergy        ! Absorbed Energy Counter
INTEGER , ALLOCATABLE,DIMENSION(:,:) :: NAEnergyCMB     ! Absorbed Energy Counter for
                                                    !combined surfaces
!
INTEGER , ALLOCATABLE,DIMENSION(:) :: TCOUNTA           ! Number of absorbed energy bunble
INTEGER , ALLOCATABLE,DIMENSION(:) :: TCOUNTR           ! Number of reflcted energy bunble
INTEGER , ALLOCATABLE,DIMENSION(:) :: TCOUNTRR          ! Number of rereflcted energy bunble
INTEGER , ALLOCATABLE,DIMENSION(:) :: NTOTAL            ! Total Number of Energy budles emitted
INTEGER , ALLOCATABLE,DIMENSION(:) :: NTACMB            ! Total Number of Energy budles emitted
                                                        ! after surface combinations
!
INTEGER , ALLOCATABLE,DIMENSION(:,:) :: Intersection    ! Surface Intersection Index
INTEGER , ALLOCATABLE,DIMENSION(:)  :: PolygonIndex     ! 3 is Triangle 4 is Rectangle
INTEGER , ALLOCATABLE,DIMENSION(:) ::CMB                ! Index for surfaces to be combined

REAL(Prec2), ALLOCATABLE,DIMENSION(:) :: EMIT           ! Emissivities of surfaces
REAL(Prec2), ALLOCATABLE,DIMENSION(:) :: TS             ! surface Temperature, K
REAL(Prec2), ALLOCATABLE,DIMENSION(:) :: BASEP          ! Refernce Point
REAL(Prec2)                           :: Rand(6)        ! Random number (0 - 1)
REAL(Prec2),  :: TIME1          ! Starting Time in s
REAL(Prec2),  :: TIME2          ! Finishing Time in s

CHARACTER (LEN=12), ALLOCATABLE,DIMENSION(:) :: SURF_NAME           ! Name of Surfaces
CHARACTER (LEN=12), ALLOCATABLE,DIMENSION(:) :: VERTEX              ! Name of Vertex
CHARACTER (LEN=12), ALLOCATABLE,DIMENSION(:) :: SURFACE             ! Index of Surfaces ("s")
Logical :: Reflected                        ! True reflected or false absorbed
Logical, ALLOCATABLE, DIMENSION(:) :: INTersects ! Surface INtersection Flag
```

```
!
REAL(prec2), ALLOCATABLE, DIMENSION(:,:) :: XP              ! Intersection Point x-coordinates
REAL(prec2), ALLOCATABLE, DIMENSION(:,:) :: YP              ! Intersection Point y-coordinates
REAL(prec2), ALLOCATABLE, DIMENSION(:,:) :: ZP              ! Intersection Point z-coordinates
REAL(prec2), ALLOCATABLE, DIMENSION(:)   :: SI              ! Scalar Vector Multiplier
REAL(prec2), ALLOCATABLE, DIMENSION(:)   :: SIPOS           ! Scalar Vector Multiplier
!
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  XLS              ! X coordinate of Source Location
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  YLS              ! Y coordinate of Source Location
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  ZLS              ! Z coordinate of Source Location
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  QFLUX            ! Net radiation flux at each surface
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  Q        ! Net radiation heat transfer at each
                                                  ! surface
!
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  XS               ! x - coordinate of a vertex
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  YS               ! y - coordinate of a vertex
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  ZS               ! z - coordinate of a vertex
!
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  Xo               ! x - coordinate of intersection point
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  Yo               ! y - coordinate of intersection point
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  Zo               ! z - coordinate of intersection point

REAL(prec2), ALLOCATABLE, DIMENSION(:,:) ::  NormalV        ! Normal Vectors of surfaces
REAL(prec2), ALLOCATABLE, DIMENSION(:,:) ::  NormalUV       ! Normal Unit Vectors of surfaces
REAL(prec2), ALLOCATABLE, DIMENSION(:,:) ::  EmittedUV      ! Unit Vector of emitted energy
REAL(prec2), ALLOCATABLE, DIMENSION(:,:) ::  Tan_V1                 ! Unit Vector tangent to the source S
REAL(prec2), ALLOCATABLE, DIMENSION(:,:) ::  Tan_V2                 ! Unit Vector tangent to the source S
REAL(prec2), ALLOCATABLE, DIMENSION(:,:) ::  RAD_D_F                ! Radiation Distribution Factor
!
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  WIDTH                    ! width of a surface
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  LENGTH                   ! Length of a surface
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  HEIGHT                   ! Height of a surface
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  Area                     ! Area of a Surface
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  AreaCMB                  ! Area of a Surface after combined
!
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  A                        ! Coefficient of X in Surface equation
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  B                        ! Coefficient of Y in Surface equation
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  C                        ! Coefficient of Z in Surface equation
REAL(prec2), ALLOCATABLE, DIMENSION(:) ::  D                        ! Constant of in Surface equation
!
!
 END MODULE Global
```

```
-----------------------------------------------------------------------------------------------------------------------------------------

MODULE EnclosureGeometry
!*****************************************************************************
!
!  MODULE:                    EnclosureGeometry
!
!  PURPOSE:                   Reads the enclosure Geometry (vertex and vertices coordinates
!             data) from a file for use in the program for surface equation
!                                            Determination
!
!*****************************************************************************
!
!
 USE Global
 IMPLICIT NONE
!
 CONTAINS
!
 SUBROUTINE CalculateGEometry()
!
 IMPLICIT NONE
!
            Integer :: I ,J,l, Openstatus, IOS, error, ErrorFlg, vr
            Character (Len=12)  ErrorMessage
            Character (Len = 12) :: SubTitle
            Character (Len = 3)  :: Dummy
            Logical ReadFile

!
!   The filename for the vertex and surface parameters of the rectangular
!   surface Enclosure
!   Reads numer of vertices and numebr od surfaces to allocate the arrays size
   i = 0;   j = 0
   Do
             Read (2,*)Dummy
             If(Trim(Dummy) == "v")Then
       i = i + 1
     Elseif(Trim(Dummy) == "!" )Then
               NVertex = i
     Elseif(Trim(Dummy) == "s")Then
       j = j + 1
     Else
               NSurf = j
              Exit
             Endif
            END DO

   Rewind(2)
!
!  Allocate the size of the array
   ALLOCATE(Vertex(NVertex),V(NVertex),XS(NVertex),YS(NVertex),ZS(NVertex)  &
                  ,STAT= IOS)
   ALLOCATE(SURFACE(NSurf),SNumber(NSurf),SVertex(NSurf,NSurf),BASEP(NSurf),&
                    CMB(NSurf),EMIT(NSurf),SURF_NAME(NSurf), STAT= IOS)
!
   DO J = 1, NVertex
                 Read (2,*)Vertex(J),V(J),XS(J),YS(J),ZS(J)
!                         Write(3,102) Vertex(J),V(J),XS(J),YS(J),ZS(J)
!102       Format(A3,2x,I2,3(2x,F7.2))
   End Do
!
```

```
        Read (2,*) SubTitle
!
        Do I = 1, NSurf
                Read (2,*)SURFACE(I),SNumber(I),(SVertex(I,J),J=1,4),BASEP(I),  &
                        CMB(I),EMIT(I),SURF_NAME(I)
!               Write(3,104)Surface(I),SNumber(I),(SVertex(I,J),J=1,4),BASEP(I),&
!                         CMB(I),EMIT(I),SURF_NAME(I)
!104    Format(A3,2x, I2,4(2x,I3),2x,f6.2,2x,f4.2,2x,f6.2,2x,A12)
  END DO
  CLOSE(Unit=2)
        END Subroutine  CalculateGEometry
!
!
!
  SUBROUTINE Calculate_SurfaceEquation()
!****************************************************************************
!
! SUBROUTINE:  Calculate_Surface_Equation
!
! PURPOSE:              Determines the coeffeciients of the surface equation using
!                               surface normal vector a point on the surface. The equation
!       is of the form Ax + By + Cz + D  = 0
!
!****************************************************************************

! Calculating the normal vector of the surfaces in the enclosure and the
! coefficients of the surface equation.  The equations is determined in
! cartesian coordinate system

  IMPLICIT NONE
  Integer :: I,J,k,m, IOS
        Integer, DIMENSION (:) :: VS(4)
        REAL(Prec2),  Dimension (4) :: X, Y, Z
        REAL(Prec2), Dimension (:,:) :: V_x(SIndex,2),V_y(SIndex,2),V_z(SIndex,2)

! V_x(SIndex,2)   Vectors on a surface used for normal vector determination
! V_y(SIndex,2)   Vectors on a surface used for normal vector determination
! V_z(SIndex,2)   Vectors on a surface used for normal vector determination
! X                       x  - coordinate of a vertice
! Y                       y  - coordinate of a vertice
! Z                       z  - coordinate of a vertice


        ALLOCATE (SPlane(NSurf),NormalV(NSurf,3),Width(NSurf),Length(NSurf), &
              Height(NSurf),NormalUV(NSurf,3),PolygonIndex(NSurf),STAT= IOS)

!  Assign the vertices of a surfaces their corresponding vertices
        DO J = 1, 4
          VS(J) = SVertex(SIndex,J)
  END DO
  DO J = 1, 4
          IF(VS(4) .ne. 0 .or. J < 4)Then
            X(J) = XS(VS(J))
            Y(J) = YS(VS(J))
            Z(J) = ZS(VS(J))
          ElseIF(VS(4) .eq. 0)Then
            X(4) = XS(VS(1))
            Y(4) = YS(VS(1))
            Z(4) = ZS(VS(1))
    ELSE
          Endif
  End Do
```

```
          IF(VS(4)==0)Then
             PolygonIndex(SIndex) = 3
   ELSE
             PolygonIndex(SIndex) = 4
   ENDIF
   DO I = 1, 2
             V_x(SIndex,I) = X(I+1) - X(I)
             V_y(SIndex,I) = Y(I+1) - Y(I)
             V_z(SIndex,I) = Z(I+1) - Z(I)
          End do
!
          Call SurfaceNormal(V_x,V_y,V_z)

!         Allocate size of the array for coefficients of surface equation
          ALLOCATE (A(NSurf),B(NSurf),C(NSurf),D(NSurf),STAT= IOS)
          DO J = 1, 4
             VS(J) = SVertex(SIndex,J)
                    IF(VS(4) .eq. 0)Then
      ELSE
               X(J) = XS(VS(J))
                              Y(J) = YS(VS(J))
                              Z(J) = ZS(VS(J))
                    ENDIF
   End Do
!         Calculates the coeffcients of the surface equation
            A(SIndex) = NormalUV(SIndex,1)
            B(SIndex) = NormalUV(SIndex,2)
            C(SIndex) = NormalUV(SIndex,3)
            D(SIndex) = -(X(1)*A(SIndex) + Y(1)*B(SIndex) + Z(1)*C(SIndex))
!           Write(*,105)'Surface Equation',SIndex,A(SIndex), B(SIndex), C(SIndex),&
!                  D(SIndex)
!105  Format(2x,A20,2x,I3,2x,4(2x,F6.2))
          END SUBROUTINE Calculate_SurfaceEquation
!
          SUBROUTINE SurfaceNormal(Vx,Vy,Vz)
!****************************************************************************
!
!  PURPOSE:             Determine normal unit vector of the surfaces in the enclosure
!
!
!****************************************************************************
    IMPLICIT NONE
    INTEGER :: I,J,k
            REAL(Prec2) :: Norm_V, NV(SIndex), Vector(3)
            REAL(Prec2), Dimension (:,:) :: Vx(SIndex,2),Vy(SIndex,2),Vz(SIndex,2)

!           Norm_V                  magnitude of a vector
!   NV(SIndex)     Magnitude of a normal vector of a surface SIndex
!   Vector(3)               Coefficients of a normal vector

!           Calculates the corss product of the vectors on a surface to determine the
!   surface Normal vector
    NormalV(SIndex,1) = Vy(SIndex,1)*Vz(SIndex,2) - Vz(SIndex,1)*Vy(SIndex,2)
    NormalV(SIndex,2) = Vz(SIndex,1)*Vx(SIndex,2) - Vx(SIndex,1)*Vz(SIndex,2)
            NormalV(SIndex,3) = Vx(SIndex,1)*Vy(SIndex,2) - Vy(SIndex,1)*Vx(SIndex,2)

            DO k =1, 3
              Vector(K) = NormalV(SIndex,k)
    END DO
            NV(SIndex) = Norm_V(Vector)
!           Write(*,104)'Magnitude of Vector',NV(SIndex)
104   Format(x,A20,2x,F6.2)
```

```
!    Normalizes the normal vector to get the unit vector
           DO J = 1, 3
             NormalUV(SIndex,J) = Vector(J)/NV(SIndex)
    END DO
!     Write(*,105)SIndex,NormalUV(SIndex,1),NormalUV(SIndex,2),NormalUV(SIndex,3)
!105   Format(x,I4,3(2x,F6.2))
          END SUBROUTINE SurfaceNormal
!
!
  SUBROUTINE Calculate_Length_Width_Height()
!****************************************************************************
!
!  PURPOSE:                  Determine the edge dimension of the surfaces in the enclosure
!
!
!****************************************************************************
!
   IMPLICIT NONE
   Integer :: I,J,k,m, IOS
          Integer, DIMENSION (:) :: VS(4)
          Real,  Dimension (4) :: X, Y, Z
!
!  Assign a surfaces thier corresponding vertices and coordinates and calculates
!  width,Length and Height
           DO J = 1, 4
             VS(J) = SVertex(SIndex,J)
               X(J) = XS(VS(J))
                           Y(J) = YS(VS(J))
                           Z(J) = ZS(VS(J))
    End Do
!         Determine the Width, Length or Height of the surfaces
    IF(NormalV(SIndex,1) == 0 .and. NormalV(SIndex,2) == 0.0)Then
            Length(SIndex) = abs(X(3) - X(1))
                  Width(SIndex)  = abs(Y(3) - Y(1))
                  Height(SIndex) = 0.0
    Elseif(NormalV(SIndex,1) == 0 .and. NormalV(SIndex,3) == 0.0)Then
            Length(SIndex) = abs(X(3) - X(1))
                  Height(SIndex) = abs(z(3) - z(1))
                  Width(SIndex)  = 0.0
    Elseif(NormalV(SIndex,2) == 0 .and. NormalV(SIndex,3) == 0.0)Then
            Width(SIndex)  = abs(y(3) - y(1))
                  Height(SIndex) = abs(z(3) - z(1))
                  Length(SIndex) = 0.0
    Else
                  Length(SIndex) = abs(x(3) - x(1))
            Width(SIndex)  = abs(y(3) - y(1))
                  Height(SIndex) = abs(z(3) - z(1))
           Endif
           Write(*,102)'W  H  L',SIndex,Length(SIndex),Width(SIndex),Height(SIndex)
102   Format(x,A10,2x,I2,3(2x,F6.2))
          END SUBROUTINE Calculate_Length_Width_Height
!
!
          SUBROUTINE Calculate_Area_Surfaces()
!****************************************************************************
!
!  PURPOSE:                  Determine areas of the surfaces in the enclosure
!
!
!****************************************************************************
!
```

```
   IMPLICIT NONE
   INTEGER :: I,J,IOS
            INTEGER, DIMENSION (:) :: VS(4)
            REAL(Prec2), DIMENSION(:) :: X(4),Y(4),Z(4)
            REAL(prec2), ALLOCATABLE,DIMENSION(:,:) :: LR, LT
            REAL(prec2), ALLOCATABLE,DIMENSION(:) :: S
!
!  LR                        Length and width of a rectangular surfaces in the enclosure
!  LT                        The three sides of a triangular surface n the enclosure
!  S                         A parameter used to calculate area for triangular surfaces
!                                    using the Heron's formula s = (LT(1) + LT(2) + LT(3))/2
!  VS                        Vertices of a surface
!  X, Y & Z          Are coordinates of a vertex


!  Assign the surfaces their corresponding vertices and coordinates and
!  and calculate areas of rectangular and triangular polygons
!
   ALLOCATE(LR(NSurf, 2), LT(NSurf, 3), S(NSurf),Area(NSurf), STAT = IOS)
!
            IF(PolygonIndex(SIndex) == 4)Then
             DO J = 1, 4
    VS(J) = SVertex(SIndex,J)
                  X(J) = XS(VS(J))
                                   Y(J) = YS(VS(J))
                                   Z(J) = ZS(VS(J))
    End Do
    DO I = 1, 2
                 LR(SIndex,I)=sqrt((X(I+1)-X(I))**2+(Y(I+1)-Y(I))**2+(Z(I+1)-Z(I))**2)
      END DO
      Area(SIndex) = LR(SIndex,1)*LR(SIndex,2)
!
   ELSEIF(PolygonIndex(SIndex) == 3)Then
             DO J = 1, 4
    VS(J) = SVertex(SIndex,J)
     IF(J < 4)Then
                  X(J) = XS(VS(J))
                                   Y(J) = YS(VS(J))
                                   Z(J) = ZS(VS(J))
      Elseif(J == 4)Then
        X(4) = XS(VS(1))
                                   Y(4) = YS(VS(1))
                                   Z(4) = ZS(VS(1))
      Endif
    END DO
               DO J = 1, 3
                        LT(SIndex,J)=SQRT((X(J+1)-X(J))**2+(Y(J+1)-Y(J))**2+(Z(J+1)-Z(J))**2)
      END DO
       S(SIndex) = (LT(SIndex,1)+LT(SIndex,2)+LT(SIndex,3))/2
                        Area(SIndex) = SQRT(S(SIndex)*(S(SIndex)-LT(SIndex,1)) &
                              *(S(SIndex)-LT(SIndex,2))*(S(SIndex)-LT(SIndex,3)))
!            Write(*,102)'Triangle A ',SIndex, Area(SIndex)
             ENDIF
!            Write(*,102)'W  H  L',SIndex, Area(SIndex)
!102   Format(x,A10,2x,I2,(2x,F6.2))
            END SUBROUTINE Calculate_Area_Surfaces
!
!
   SUBROUTINE CrossProduct(Vec1, Vec2, Vec)
!*************************************************************************
!
! PURPOSE:                  Calculates the crossProduct of two vectors
```

```
!
!
!**************************************************************************
   REAL(Prec2) :: Vec1(3),Vec2(3), Vec(3)
     Vec(1) = Vec1(2)*Vec2(3) - Vec1(3)*Vec2(2)
     Vec(2) = Vec1(3)*Vec2(1) - Vec1(1)*Vec2(3)
     Vec(3) = Vec1(1)*Vec2(2) - Vec1(2)*Vec2(1)
   END SUBROUTINE CrossProduct
!
!
  Function Norm_V(V)
!**************************************************************************
!
!  PURPOSE:                    Calculates the magnitude of a vector
!
!
!**************************************************************************
   IMPLICIT NONE
   REAL(Prec2) :: V(3), Norm_V
!  V(3)              the vector whose magnitude is to be determined
!  Norm_V                      is the magnitude of the vector V
!
   Norm_V = 0.0d0
              Norm_V = SQRT(DOT_PRODUCT(V,V))
  END Function Norm_V



 SUBROUTINE AllocateArrays()
!**************************************************************************
!
!  PURPOSE:                    Allocates the arrays
!
!
!**************************************************************************

  IMPLICIT NONE
  INTEGER :: IOS

  ALLOCATE(NAEnergy(NSurf,NSurf),RAD_D_F(NSurf,NSurf),STAT = IOS )
  ALLOCATE(TCOUNTA(NSurf),TCOUNTR(NSurf),TCOUNTRR(NSurf),NTOTAL(NSurf) &
                           ,STAT=IOS)
  ALLOCATE(XLS(NSurf),YLS(NSurf),ZLS(NSurf), STAT= IOS)
  ALLOCATE(XP(NSurf,NSurf),YP(NSurf,NSurf),ZP(NSurf,NSurf),                &
                Intersection(NSurf,NSurf), STAT = IOS)
  ALLOCATE(Xo(NSurf),Yo(NSurf),Zo(NSurf),INTersects(NSurf),STAT = IOS)

  END SUBROUTINE AllocateArrays


  SUBROUTINE InitializeArrays()
!**************************************************************************
!
!  PURPOSE:                    Initializes the arrays
!
!
!**************************************************************************

  IMPLICIT NONE
  INTEGER :: I, J, K, IOS

!  Initialize absrorbed and reflected energy bundle counter arrays
```

```
   DO J = 1, NSurf
    DO k = 1, NSurf
             NAEnergy(J,k) = 0
    END DO
             TCOUNTA(J) = 0; TCOUNTR(J) = 0; TCOUNTRR(J)= 0
   END DO
!

  END SUBROUTINE InitializeArrays

 End MODULE EnclosureGeometry
```

```
------------------------------------------------------------------------------------------------------------------------------------

Module EnergyBundleLocation

!*******************************************************************************
!          PURPSOE:          Locating the position of the Emitted or reflected EnergyBundle
!                                         on a surface and the direction of the ray
!
!          CREATED BY:     Bereket A. Nigusse               10.19.04
!
!
!!******************************************************************************

 USE GLOBAL
 USE EnclosureGeometry

 IMPLICIT NONE
 CONTAINS


 SUBROUTINE EnergySourceLocation()
!*******************************************************************************
!
!  Purpose:            Checks whether the surface rectangular or triangular, then calls
!          the appropriate subroutine. If the fourth vertex index is zero
!                                         then the polygon is triangular else it rectangular
!
!*******************************************************************************
   INTEGER :: I, J, K, IOS
   Call RANDOM_NUMBER(Rand)
          IF(PolygonIndex(SIndex) .eq. 4)Then
            CALL RectangularSurface()
          ELSEIF( PolygonIndex(SIndex) .eq. 3)Then
           CALL TriangularSurface()
   ELSE
          Endif
 END SUBROUTINE EnergySourceLocation


 SUBROUTINE TriangularSurface()
!******************************************************************************
!
!  Purpose:            Determines the location of the emitted energy on a triangular
!          surface randomly
!
!
!******************************************************************************
!  Rand                 Normalized uniform distribution Random numbers between 0 and 1
!  XLS                             Location of x-coordinate of the source on a particular surface
!  YLS                             Location of y-coordinate of the source on a particular surface
!  ZLS                             Location of z-coordinate of the source on a particular surface
!  VS(4)                The four vertices used to define a surface and are inputs
!  X, Y, Z                         The coordinates of a vertex

          IMPLICIT NONE
   INTEGER :: I, J, K, IOS
          INTEGER, DIMENSION (:) :: VS(4)
          REAL(Prec2), DIMENSION(4) :: X, Y, Z
          REAL(Prec2), DIMENSION(:,:) :: Vedge1(3), Vedge2(3)
          REAL(Prec2) :: Randu, Randv

! If it is reflected energy bundle no need to calculate the emission point
```

```
  IF(Reflected)Then
!   XLS(SIndex) = XP(SIndexR, SInter)
!   YLS(SIndex) = YP(SIndexR, SInter)
!   ZLS(SIndex) = ZP(SIndexR, SInter)

   XLS(SIndex) = Xo(SInter)
   YLS(SIndex) = Yo(SInter)
         ZLS(SIndex) = Zo(SInter)
!   write(*,112)SInter,XLS(SIndex),YLS(SIndex),ZLS(SIndex)
!112 format(x,I2,2x,3(2x,f6.3))
  ELSE
   DO J = 1, 3
            VS(J) = SVertex(SIndex,J)
            X(J) = XS(VS(J))
            Y(J) = YS(VS(J))
                  Z(J) = ZS(VS(J))
   END DO
!   Calculates two edge vectors for a triangular polygon
         Vedge1(1) = (X(2) - X(1))
         Vedge1(2) = (Y(2) - Y(1))
         Vedge1(3) = (Z(2) - Z(1))
!
         Vedge2(1) = (X(3) - X(1))
         Vedge2(2) = (Y(3) - Y(1))
         Vedge2(3) = (Z(3) - Z(1))
         IF ((Rand(1) + Rand(2)) .gt. 1.0)Then
          Rand(1) = 1.0 - Rand(1)
          Rand(2) = 1.0 - Rand(2)
   ELSE
         ENDIF
!        Determine the location of the source
   IF(NormalV(SIndex,1) == 0 .and. NormalV(SIndex,2) == 0.0)Then
          XLS(SIndex) = X(1) + Rand(1)*Vedge1(1) + Rand(2)*Vedge2(1)
          YLS(SIndex) = Y(1) + Rand(1)*Vedge1(2) + Rand(2)*Vedge2(2)
          ZLS(SIndex) = Z(1)
   Elseif(NormalV(SIndex,1) == 0 .and. NormalV(SIndex,3) == 0.0)Then
          YLS(SIndex) = Y(1)
          XLS(SIndex) = X(1) + Rand(1)*Vedge1(1) + Rand(2)*Vedge2(1)
          ZLS(SIndex) = Z(1) + Rand(1)*Vedge1(3) + Rand(2)*Vedge2(3)
   Elseif(NormalV(SIndex,2) == 0 .and. NormalV(SIndex,3) == 0.0)Then
          XLS(SIndex) = X(1)
          YLS(SIndex) = Y(1) + Rand(1)*Vedge1(2) + Rand(2)*Vedge2(2)
          ZLS(SIndex) = Z(1) + Rand(1)*Vedge1(3) + Rand(2)*Vedge2(3)
   Else
    IF(NormalV(SIndex,2) == 0 )Then
     XLS(SIndex) = X(1) + Rand(1)*Vedge1(1) + Rand(2)*Vedge2(1)
           YLS(SIndex) = Y(1) + Rand(1)*Vedge1(2) + Rand(2)*Vedge2(2)
           ZLS(SIndex)=-(D(SIndex)+A(SIndex)*XLS(SIndex)+B(SIndex)  &
                                        *YLS(SIndex))/C(SIndex)
    ELSEIF(NormalV(SIndex,1) == 0)Then
           XLS(SIndex) = X(1) + Rand(1)*Vedge1(1) + Rand(2)*Vedge2(1)
           YLS(SIndex) = Y(1) + Rand(1)*Vedge1(2) + Rand(2)*Vedge2(2)
           ZLS(SIndex)=-(D(SIndex)+A(SIndex)*XLS(SIndex)+B(SIndex) &
                 *YLS(SIndex))/C(SIndex)
    ELSEIF(NormalV(SIndex,3) == 0)Then
           YLS(SIndex) = Y(1) + Rand(1)*Vedge1(2) + Rand(2)*Vedge2(2)
           ZLS(SIndex) = Z(1) + Rand(1)*Vedge1(3) + Rand(2)*Vedge2(3)
           XLS(SIndex)=-(D(SIndex)+C(SIndex)*ZLS(SIndex)+B(SIndex) &
                 *YLS(SIndex))/A(SIndex)
          ENDIF
   Endif
```

```
!           Write(3,101)'XLs YLs ZLs',XLS(SIndex), YLS(SIndex), ZLS(SIndex)
!101 Format(x,A15,3(2x,F12.8))
  ENDIF
  END SUBROUTINE TriangularSurface


  SUBROUTINE RectangularSurface
!*****************************************************************************
!
!   Purpose:            Calculates the location of the emitted energy on a rectangular
!           surface randomly
!
!
!*****************************************************************************
!   Rand                Normalized uniform distribution Random numbers between 0 and 1
!   XLS                             Location of x-coordinate of the source on a particular surface
!   YLS                             Location of y-coordinate of the source on a particular surface
!   ZLS                             Location of z-coordinate of the source on a particular surface
!   VS(4)               The four vertices used to define a surface and are inputs
!   X, Y, Z                         The coordinates of a vertex

            IMPLICIT NONE
    INTEGER :: I, J, K, IOS
            Integer, DIMENSION (:) :: VS(4),  SurfaceE(13)
            REAL(Prec2), DIMENSION(4) :: X, Y, Z
!
! If the energy is reflected then its location will be the point of intersection
  IF( Reflected)Then
    XLS(SIndex) = Xo(SInter)
    YLS(SIndex) = Yo(SInter)
            ZLS(SIndex) = Zo(SInter)
!   write(*,112)SInter,XLS(SIndex),YLS(SIndex),ZLS(SIndex)
!112 format(x,I2,2x,3(2x,f6.3))
  ELSE
    DO J = 1, 4
               VS(J) = SVertex(SIndex,J)
                X(J) = XS(VS(J))
                Y(J) = YS(VS(J))
                      Z(J) = ZS(VS(J))
    END DO
 !          Determine Locaton of the energy source
     IF(NormalV(SIndex,1) == 0 .and. NormalV(SIndex,2) == 0.0)Then
                    XLS(SIndex) = X(1) + (X(3) - X(1))*Rand(1)
              YLS(SIndex) = Y(1) + (Y(3) - Y(1))*Rand(2)
              ZLS(SIndex) = Z(1)
     Elseif(NormalV(SIndex,1) == 0 .and. NormalV(SIndex,3) == 0.0)Then
              YLS(SIndex) = Y(1)
                    XLS(SIndex) = X(1) + (X(3) - X(1))*Rand(1)
              ZLS(SIndex) = Z(1) + (Z(3) - Z(1))*Rand(3)
     Elseif(NormalV(SIndex,2) == 0 .and. NormalV(SIndex,3) == 0.0)Then
                    XLS(SIndex) = X(1)
              YLS(SIndex) = Y(1) + (y(3) - y(1))*Rand(2)
              ZLS(SIndex) = Z(1) + (z(3) - z(1))*Rand(3)
     Else
             IF(NormalV(SIndex,2) == 0 )Then
                    XLS(SIndex) = X(1) + (x(3) - x(1))*Rand(1)
              YLS(SIndex) = Y(1) + (y(3) - y(1))*Rand(2)
              ZLS(SIndex)=-(D(SIndex)+A(SIndex)*XLS(SIndex)+B(SIndex)*YLS(SIndex)) &
                           /C(SIndex)
      ELSEIF(NormalV(SIndex,1) == 0)Then
                    XLS(SIndex) = X(1) + (x(3) - x(1))*Rand(1)
              YLS(SIndex) = Y(1) + (y(3) - y(1))*Rand(2)
              ZLS(SIndex)=-(D(SIndex)+A(SIndex)*XLS(SIndex)+B(SIndex)*YLS(SIndex)) &
                          /C(SIndex)
     ELSEIF(NormalV(SIndex,3) == 0)Then
                    ZLS(SIndex) = z(1) + (z(3) - z(1))*Rand(3)
              YLS(SIndex) = Y(1) + (y(3) - y(1))*Rand(2)
              XLS(SIndex)=-(D(SIndex)+C(SIndex)*ZLS(SIndex)+B(SIndex)*YLS(SIndex)) &
                          /A(SIndex)
             ENDIF
            Endif
           SurfaceE(SIndex) = D(SIndex)+A(SIndex)*XLS(SIndex)+B(SIndex)*YLS(SIndex) &
                        + C(SIndex)*ZLS(SIndex)
!   IF (SIndexR == 1 .and. SIndex ==1)Then
!           Write(3,101)'XLsYLsZLs',SIndexR,SIndex,XLS(SIndex),YLS(SIndex),ZLS(SIndex),&
!             SurfaceE(SIndex)
!101  Format(x,A15,2x,I2,2x,I2,2x,3(2x,F10.6),2x,F12.10)
!   Else
!           Endif
  ENDIF
 END SUBROUTINE RectangularSurface


 SUBROUTINE InitializeSeed()
!****************************************************************************
!
!  PURPOSE:       Initialization of seed for the random Number generator
!
!
!****************************************************************************
           IMPLICIT NONE
   INTEGER :: K
           INTEGER, DIMENSION(:) ::  SEEDARRAY(6), OLDSEED(6)
!   Sets K = N
           K = 6
   CALL RANDOM_SEED (SIZE = K)
!   Set user seed
   CALL RANDOM_SEED (PUT = SEEDARRAY(1:K))
!   Get current seed
   CALL RANDOM_SEED (GET = OLDSEED(1:K))
 END SUBROUTINE InitializeSeed


 SUBROUTINE TangentVectors()
!*****************************************************************************
!
!  PURPOSE:       Determines unit tangent vectors on a surface in the enclosure
!
!
!*****************************************************************************
!  UV_X(3)                   Unit vector along x-direction
!  UV_Y(3)                   Unit vector along Y-direction
!  UV_Z(3)                   Unit vector along Z-direction
!  TUV1(3)                   Unit vector tangent to the source point on a surface
!          TUV2(3)           Unit vector tangent to the source point on a surface
!          and normal to the TUV1 tangent vector
!                                     The tangent vectors are used for reference in defining the angle
!          Thus, need to be determined once for each surface
!SmallestRealNo The smallest machine number
!
   IMPLICIT NONE
   INTEGER :: I,J, K, IOS, INDEX
           REAL(Prec2) ::  UV_x(3), UV_y(3), UV_z(3),V(3), TUV1(3), TUV2(3),VDOT(3)
           REAL(Prec2) ::      SmallestRealNo,NV, xx
```

```
!
!           define the smallest machine number
            SmallestRealNo = EPSILON(0.0d0)
!
   ALLOCATE(Tan_V1(NSurf,3),Tan_V2(NSurf,3), STAT= IOS)
   DO I = 1, 3
    Tan_V1(SIndex, I) = 0.0
             Tan_V2(SIndex, I) = 0.0
    V(I)     = NormalUV(SIndex,I)
    UV_x(I) = 0.0
             UV_y(I) = 0.0
             UV_z(I) = 0.0
   END DO
             UV_x(1) = 1.0
             UV_y(2) = 1.0
             UV_z(3) = 1.0

!    The first tangent vector is determined first as follows
    VDOT(1) = DOT_PRODUCT(V,UV_x)
             VDOT(2) = DOT_PRODUCT(V,UV_y)
             VDOT(3) = DOT_PRODUCT(V,UV_z)
             If((1.0 - abs(VDOT(1))) .gt. SmallestRealNo)Then
               Call CrossProduct(V, UV_x, TUV1)
    ELseif((1.0 - abs(VDOT(2))) .gt. SmallestRealNo)Then
                        Call CrossProduct(V, UV_y, TUV1)
             Else
               Call CrossProduct(V, UV_z,TUV1)
             ENDIF
             NV = Norm_V(TUV1)
    DO J = 1, 3
              TUV1(J) = TUV1(J)/NV
     Tan_V1(SIndex,J) = TUV1(J)
    END DO
!
             Write(5,102)'Tan_V1', SIndex,Tan_V1(SIndex,1),Tan_V1(SIndex,2),Tan_V1(SIndex,3)
102  FORMAT(X,a10,2x,I2,2x,3(2X,F10.6))
!
!  The second tangent vector is given by the cross product of the surface normal
!  vector and the firtst tangent vector
    Call CrossProduct(V, TUV1,TUV2)
    DO J = 1, 3
     Tan_V2(SIndex,J) = TUV2(J)
    END DO
!
!            Write(5,103)'Tan_V2', SIndex,Tan_V2(SIndex,1),Tan_V2(SIndex,2),Tan_V2(SIndex,3)
!103  FORMAT(X,a10,2x,I2,2x,3(2X,F10.6))
  END SUBROUTINE TangentVectors
!
  SUBROUTINE DirectionEmittedEnergy()
!****************************************************************************
!
!   PURPOSE:       Determines the direction of the emitted energy bundle
!
!
!****************************************************************************
!   THETA                      The angle of the emitted energy bundle makes with the normal to
!                                        the surface
!   PHI                        Polar angle of the emitted energy bundle
!   Rand(4)                    Random number for zenith angle theta
!   Rand(5)                    Random number for azimuth angle phi
!
   IMPLICIT NONE
   INTEGER                  :: IOS
            REAL(Prec2)     :: Theta, Phi, Pi
!
!  Calculate emitted energy bundle direction angles
   Pi = 4.*Atan(1.)
            Theta = asin(sqrt(Rand(4)))
   Phi   = 2.*Pi*Rand(5)
!
! Calculate the unit vector in the direction of the emitted energy bundle
   ALLOCATE (EmittedUV(NSurf,3), STAT = IOS )
            EmittedUV(SIndex,1) = NormalUV(SIndex,1)*cos(Theta) + Tan_V1(SIndex,1) &
                  *sin(Theta)*cos(Phi) + Tan_V2(SIndex,1)*sin(Theta)*sin(Phi)
            EmittedUV(SIndex,2) = NormalUV(SIndex,2)*cos(Theta) + Tan_V1(SIndex,2) &
                  *sin(Theta)*cos(Phi) + Tan_V2(SIndex,2)*sin(Theta)*sin(Phi)
            EmittedUV(SIndex,3) = NormalUV(SIndex,3)*cos(Theta) + Tan_V1(SIndex,3) &
                  *sin(Theta)*cos(Phi) + Tan_V2(SIndex,3)*sin(Theta)*sin(Phi)

!           Write(*,105)'Angles',Theta,Phi,EmittedUV(SIndex,1),EmittedUV(SIndex,2) &
!           ,EmittedUV(SIndex,3)
!105 Format(x,A10,5(2x,f6.2))

  END SUBROUTINE  DirectionEmittedEnergy

End Module EnergyBundleLocation
```

```
------------------------------------------------------------------------------------------------------------------------------------------

 MODULE IntersectionEnergy_Surface
!***************************************************************************
!
!  MODULE:                    IntersectionEnergy_Surface
!
!  PURPOSE:                   Determines the point of intersection of the emitted energy &
!           the surfaces in the enclosure
!
!***************************************************************************


 USE Global
 USE EnclosureGeometry
 USE EnergyBundleLocation
 USE EnergyAbsorbed_Reflected

 IMPLICIT NONE
 CONTAINS

! Checking intersection point of emitted ray and surfaces in the enclosure
! the emitted ray navigates through the equation of surfaces

 SUBROUTINE CheckingIntersection

!***************************************************************************
!
!  SUBROUTINE:  CheckingIntersection
!
!  PURPOSE:                   Determines the point of intersection between the emitted
!           energy ray and the surfaces
!
!  CALLS:                     Subroutines Intersection_Points & SingleOutIntersection
!
!***************************************************************************


  IMPLICIT NONE
  INTEGER :: I, J, K, Index, IOS, InterCount

  CALL Intersection_Points()
  CALL SingleOutIntersection()

 END SUBROUTINE CheckingIntersection


 SUBROUTINE Intersection_Points()
!***************************************************************************
!
!  SUBROUTINE:  Intersection_Points
!
!  PURPOSE:                   Determines all possible points of intersection for the
!           surfaces in the enclosure
!
!***************************************************************************
!
  IMPLICIT NONE
          INTEGER :: I, J, K, Index,SCount, IOS, InterCount
          INTEGER, DIMENSION (:) :: VS(4)
```

```
            REAL(Prec2), DIMENSION(:) :: WV(3), UNV(3),EUV(3),W_V(3)
            REAL(Prec2) :: UNV_DOT_WV, UNV_DOT_EUV
            REAL(Prec2),  Dimension (:) :: X(4), Y(4), Z(4)
!
!   SI                                  Scalar multiplyer of emitted energy unit vector to locate
!                                               the intersection point
!           UNV                                 Unit vector normal to the surfaces
!           EUV                                 Unit vector in the direction of the emitted energy
!           WV                                  A vector from a point on a surface intersection the ray
!              to the source point the surface emitting the energy
!           W_V                                 Unit vector in the direction of the emitted energy
!           UNV_DOT_WV                  Dot product of UNV and WV vectors
!   UNV_DOT_EUV             Dot product of UNV and EUV vectors
!
   ALLOCATE(SI(NSurf),STAT = IOS )
!  Assign surfaces thier corresponding vertices and coordinates
   DO Index = 1, NSurf
    DO J = 1, 4
             VS(J) = SVertex(Index,J)
            IF(VS(4) .ne. 0 )Then
             X(J) = XS(VS(J))
             Y(J) = YS(VS(J))
             Z(J) = ZS(VS(J))
    ELSEIF(J .lt. 4)Then
             X(J) = XS(VS(J))
             Y(J) = YS(VS(J))
             Z(J) = ZS(VS(J))
            ELSE
            ENDIF
    End Do
!
!  Determine a vector between a point on a surface considered for intersection
!  the emitted energy source point
!
    IF(Index .ne. SIndex) Then
            WV(1) = -(XLS(SIndex) - X(1))
    WV(2) = -(YLS(SIndex) - Y(1))
            WV(3) = -(ZLS(SIndex) - Z(1))
!
!  Determine the dot product of the surfaces unit vector and vector WV
    DO I = 1, 3
     UNV(I) = NormalUV(Index,I)
             W_V(I) = WV(I)
             EUV(I) = EmittedUV(SIndex,I)
            END DO
    UNV_DOT_WV  = DOT_PRODUCT(UNV,W_V)
    UNV_DOT_EUV = DOT_PRODUCT(UNV,EUV)
    SI(Index) = UNV_DOT_WV/UNV_DOT_EUV
   ELSE
   ENDIF
    DO I = 1, 3
     UNV(I) = 0.0
             W_V(I) = 0.0
             EUV(I) = 0.0
            END DO
   END DO

  END SUBROUTINE Intersection_Points
```

```
!
 SUBROUTINE SingleOutIntersection()
!**************************************************************************
!
! SUBROUTINE:  SingleOutIntersection
!
! PURPOSE:                    Selectes the exact intersection points from the possible
!            intersection points
! USES:               Subroutine IntersectionTriangle(Scount) &
!            IntersectionRectangle(Scount)
!
!**************************************************************************
!
   IMPLICIT NONE
            INTEGER :: I, J, K, Index,Scount,IOS,InterCount
            INTEGER, DIMENSION (:) :: VS(4)
            REAL(Prec2), ALLOCATABLE,DIMENSION(:) :: SIINTER
            REAL(prec2) SIMIN, SIMAX

!  SIMIN              the closest intersection distnace
!           SIMAX                Maximum real number
!  Assign the maximum Real number to SIMAX

   SIMAX  = 1000000000000.0d0


  Allocate(SIINTER(NSurf), STAT = IOS)
! Calculates the vector position of the intersection point
  DO Index = 1, NSurf
   IF(Index .ne. SIndex)Then
     XP(SIndex,Index) = XLS(SIndex) + SI(Index)*EMittedUV(SIndex,1)
     YP(SIndex,Index) = YLS(SIndex) + SI(Index)*EMittedUV(SIndex,2)
             ZP(SIndex,Index) = ZLS(SIndex) + SI(Index)*EMittedUV(SIndex,3)

!
            IF(SI(Index) < 0.0)Then
     Intersection(SIndex,Index) = 0  !0 means no intersection, 1 means there is Inter.
            Else
             Intersection(SIndex,Index) = 1
            Endif
           Else
   Endif
  End DO

   DO Scount = 1, NSurf
     IF(PolygonIndex(Scount) .eq. 4 .and. Intersection(SIndex,Scount) == 1)Then
               Call IntersectionRectangle(Scount)
             ELSEIF(PolygonIndex(Scount) .eq. 3 .and. Intersection(SIndex,Scount) == 1)Then
               Call IntersectionTriangle(Scount)
     ELSE

             EndIF
           END DO

! Consider the positive length only
   DO I =1, NSurf
             IF(SI(I) > 0.0 )THen
               SIINTER(I) = SI(I)
     Else
               SIINTER(I) = SIMAX
     ENDIF
           END DO
```

```
! The intersection point is the closest point from possible intersection points
  SIMIN = MINVAL(SIINTER)


  DO I =1, Nsurf
   IF (INTersects(I))Then     ! Intersects True or false
           IF(SIINTER(I) == SIMIN) Then
             SInter = I

    If(SIndex == 1 .and. SInter == 4)Then
              Write(4,101)'Inters',SIndexR,SInter, XP(SIndexR,SInter),YP(SIndexR,I),&
                  ZP(SIndexR,SInter)
                    Write(4,101)'Inters',SIndexR,SInter, Xo(SInter),Yo(Sinter),Zo(SInter)
101 Format(X,A8,2x,I2,2X,I2,3(X,f12.4))
    Else
           Endif

           Endif

          ELSE
          END IF
  END DO



 END SUBROUTINE SingleOutIntersection!
!
 SUBROUTINE IntersectionRectangle(Index)
!***************************************************************************
!
! SUBROUTINE:  IntersectionRectangle
!
! PURPOSE:             Selectes the exact intersection points for recatngular surfaces
!
!
!***************************************************************************
!
!          UNV             = Unit normal vector of the surfaces
! V_Int  = Vector from the evrtices to the intersection point
!          V_edge   = Vector along the edges of the surfaces defined in consistent
!          direction
!          VcpS      = Cross product vector between the edges and intersection vector

  IMPLICIT NONE
  INTEGER :: I, J, K, Index,SCount,IOS, count
  INTEGER, DIMENSION (:) :: VS(4)
  REAL(Prec2),DIMENSION(:,:):: VcpS(NSurf,3),VcpN(NSurf,4)
  REAL(Prec2),DIMENSION(:) ::  V(3),X(4),Y(4),Z(4),V_edge(3),V_Int(3),Vcp(3) &
                     ,UNV(3)
  REAl(Prec2) SIMIN
! checks whether the point of intersection of the surfaces plane is within the
! enclosure
! Assign surfaces thier corresponding vertices and coordinates
   DO J = 1, 4
           VS(J) = SVertex(Index,J)
           X(J) = XS(VS(J))
           Y(J) = YS(VS(J))
           Z(J) = ZS(VS(J))
   End Do
!
! Determine a vector for of the surface edges using the vertices of the surfaces
```

```
    IF(Index .ne. SIndex .and.  Intersection(SIndex,Index) == 1) Then
             DO J = 1, 4
     If (J < 4 )Then
              V_edge(1) = (X(J+1) - X(J))
      V_edge(2) = (Y(J+1) - Y(J))
              V_edge(3) = (Z(J+1) - Z(J))
             Elseif(J == 4)Then
      V_edge(1) = (X(1) - X(4))
      V_edge(2) = (Y(1) - Y(4))
              V_edge(3) = (Z(1) - Z(4))
             Else
             Endif
!
!    Determine a vector from a vertex on the surface to the intersection point on
!    the plane of the same surface
     V_Int(1) = XP(SIndex,Index) - X(J)
             V_Int(2) = YP(SIndex,Index) - Y(J)
             V_Int(3) = ZP(SIndex,Index) - Z(J)
!
             Call CrossProduct(V_edge, V_Int,Vcp)
     DO I = 1, 3
      UNV(I) = NormalUV(Index,I)
     END DO
             VcpN(Index, J) = DOT_PRODUCT(Vcp,UNV)
     DO I = 1, 3
              VcpS(Index,I) = Vcp(I)
     END DO
    END DO
   ELSE
            ENDIF

!    Write(*,110)'Nm C Pord.',Index,VcpN(Index,1),VcpN(Index,2),VcpN(Index,3)
!110  format(x,A15,2x,I3,4(2x,f12.4))

! Confirm the itersection
            IF(VcpN(Index,1)> 0.0 .and. VcpN(Index,2)> 0.0 .and. VcpN(Index,3)> 0.0  &
              .and. VcpN(Index,4) > 0.0 .and. Intersection(SIndex,Index) == 1)Then
       SInter = Index

! Save the intersection point coordinates
      Xo(SInter) = XP(SIndex,Index)
              Yo(SInter) = YP(SIndex,Index)
              Zo(SInter) = ZP(SIndex,Index)

!    Write(*,110)'Nm C Pord.',Index,VcpN(Index,1),VcpN(Index,2),VcpN(Index,3)
!110  format(x,A15,2x,I3,4(2x,f12.4))


!    If(SINter ==1 .and. SIndexR == 13)Then
!              Write(*,101)'Inters',SIndex,SInter, XP(SIndex,Index),YP(SIndex,Index),&
!                   ZP(SIndex,Index)
!101  Format(X,A8,2x,I2,2X,I2,3(X,f12.4))
!     Else
!            Endif

!    Write(*,110)'Nm C Pord.',Index,VcpN(Index,1),VcpN(Index,2),VcpN(Index,3),SI(Index)
!110  format(x,A15,2x,I3,5(2x,f12.4))

   ELSE
            ENDIF
  END SUBROUTINE IntersectionRectangle
!
```

```
!
 SUBROUTINE IntersectionTriangle(Index)
!**************************************************************************
!
! SUBROUTINE:  IntersectionTriangle
!
! PURPOSE:              Selectes the exact intersection points for triangular surfaces
!
!
!**************************************************************************
!
!          UNV              = Unit normal vector of the surfaces
!  V_Int  = Vector from the evrtices to the intersection point
!          V_edge   = Vector along the edges of the surfaces defined in consistent
!          direction
!          VcpS      = Cross product vector between the edges and intersection vector

   IMPLICIT NONE
          INTEGER :: I, J, K, Index,SCount,IOS, count
          INTEGER, DIMENSION(:) :: VS(4)
   REAL(Prec2),DIMENSION(:,:):: VcpS(NSurf,3),VcpN(NSurf,4)
          REAL(Prec2),DIMENSION(:)::V(3),X(4),Y(4),Z(4),V_edge(3),V_Int(3),Vcp(3),UNV(3)
!
! check whether the point of intersection of the surfaces is within the enclosure
    DO J = 1, 3
            VS(J) = SVertex(Index,J)
            X(J) = XS(VS(J))
            Y(J) = YS(VS(J))
            Z(J) = ZS(VS(J))
    End Do
! Determine a vector for the surface edges using the vertices of the surfaces
   IF(Index .ne. SIndex .and.  Intersection(SIndex,Index) == 1) Then
          DO J = 1, 3
    If (J < 3 )Then
            V_edge(1) = (X(J+1) - X(J))
     V_edge(2) = (Y(J+1) - Y(J))
            V_edge(3) = (Z(J+1) - Z(J))
           Elseif(J == 3)Then
     V_edge(1) = (X(1) - X(3))
     V_edge(2) = (Y(1) - Y(3))
            V_edge(3) = (Z(1) - Z(3))
           Endif
! Determine a vector from a vertex on the surface to the intersection point on
! the plane of the same surface
     V_Int(1) = XP(SIndex,Index) - X(J)
            V_Int(2) = YP(SIndex,Index) - Y(J)
            V_Int(3) = ZP(SIndex,Index) - Z(J)
!
           Call CrossProduct(V_edge, V_Int,Vcp)
    DO I = 1, 3
     UNV(I) = NormalUV(Index,I)
    END DO
           VcpN(Index, J) = DOT_PRODUCT(Vcp,UNV)
    DO I = 1, 3
            VcpS(Index,I) = Vcp(I)
    END DO
!   Write(*,110)'Cross Porduct',Index,VcpS(Index,1),VcpS(Index,2),VcpS(Index,3)
          END DO
!   Write(*,110)'Nm C Pord.',Index,VcpN(Index,1),VcpN(Index,2),VcpN(Index,3)
!110  format(x,A15,2x,I3,4(2x,f12.4))
   ELSE
          ENDIF
   If(VcpN(Index,1) > 0.0 .and. VcpN(Index,2) > 0.0 .and. VcpN(Index,3) > 0.0 &
                .and. Intersection(SIndex,Index) == 1)Then
      SInter = Index

!    Write(*,*)'in ',SINdexR,SINDEX,SInter
!  Save the intersection point coordinates
      Xo(SInter) = XP(SIndex,Index)
               Yo(SInter) = YP(SIndex,Index)
               Zo(SInter) = ZP(SIndex,Index)
!       If(SInter == 1 .and. SIndex == 13)Then
!                 Write(4,101)'Inters',SIndex,SInter, XP(SIndex,Index),YP(SIndex,Index),&
!                      ZP(SIndex,Index)
!101 Format(X,A8,2x,I2,2X,I2,3(X,f12.4))
!       Else
!              Endif
   ELSE
           ENDIF
  END SUBROUTINE IntersectionTriangle
  END MODULE IntersectionEnergy_Surface
```

```
  MODULE EnergyAbsorbed_Reflected


  USE Global
  USE EnclosureGeometry
  USE EnergyBundleLocation
!  USE IntersectionEnergy_Surface


  IMPLICIT NONE


  CONTAINS

  SUBROUTINE AbsorptionReflection

!*******************************************************************************
!
! PURPOSE:                              Checking whether the energy bundle absorbed or reflected
!
!*******************************************************************************
```

```
   IMPLICIT NONE
   INTEGER                    :: I,J, K, IOS, count
           REAL(Prec2)  :: R_absorbed

!  R_absorbed                  Random number used generated for verifying whether the
!             intersecpted energy is absorbed or reflected
!
   R_absorbed = Rand(6)
!
    IF(R_absorbed < Emit(SInter))Then
      NAEnergy(SIndexR,SInter) = NAEnergy(SIndexR,SInter) + 1
              TCOUNTA(SIndexR) = TCOUNTA(SIndexR) + 1

!      IF(SIndex == SIndexR )THen
!      Write(4,101, ADVANCE='YES')'P',',',SIndex,',',XLS(SIndex),',',YLS(SIndex),','  &
!                 ,ZLS(SIndex),',',SInter,',',XP(SIndex,SInter),','&
!                                               ,YP(SIndex,SInter),',',ZP(SIndex,SInter)
!      Write(*,101, ADVANCE='YES')'P',',',SIndex,',',XLS(SIndex),',',YLS(SIndex),','  &
!                 ,ZLS(SIndex),',',SInter,',',XP(SIndex,SInter),','&
!                                               ,YP(SIndex,SInter),',',ZP(SIndex,SInter)
!101    Format(A2,A2,I2,3(A2,f6.3),A2,I2,3(A2,f6.3))
!      ELSE
!      Write(4,111,ADVANCE='YES')',',SInter,',',XP(SIndex,SInter),','&
!                                               ,YP(SIndex,SInter),',',ZP(SIndex,SInter)
!      Write(*,111,ADVANCE='YES')',',SInter,',',XP(SIndex,SInter),','&
!                                               ,YP(SIndex,SInter),',',ZP(SIndex,SInter)
!111    Format(A2,I2,3(A2,f6.3))

!      END IF
!      IF (SIndex ==1 .and. SInter == 2)Then
!        count = count + 1
!                     write(3,*)'count' , count
!                     Write(3,*)'NAEnergy(1,2)',NAEnergy(1,2)
!      ENDIf


              SIndex = SIndexR
              REF_IND = 0
              Reflected = .False.

    ELSE

      IF(SIndex == SIndexR .and. REF_IND == 0)Then
       TCOUNTR(SIndexR) = TCOUNTR(SIndexR) + 1
       REF_IND = 1
      Else
              TCOUNTRR(SIndexR) = TCOUNTRR(SIndexR) + 1
      ENDIF

!!       IF(Reflected)Then
!        Write(4,102,ADVANCE='NO')',',SInter,',',XP(SIndex,SInter),',' &
!                                               ,YP(SIndex,SInter),',',ZP(SIndex,SInter)
!
!       Write(*,102,ADVANCE='NO')',',SInter,',',XP(SIndex,SInter),',' &
!                                               ,YP(SIndex,SInter),',',ZP(SIndex,SInter)
!102    format(A2,I2,3(A2,f6.3))

!!       Else
!!         Write(4,112, ADVANCE='YES')'P',',',SIndex,',',XLS(SIndex),',',YLS(SIndex),','  &
!!                 ,ZLS(SIndex),',',SInter,',',XP(SIndex,SInter),','&
!!                                               ,YP(SIndex,SInter),',',ZP(SIndex,SInter)
!!         Write(*,112, ADVANCE='YES')'P',',',SIndex,',',XLS(SIndex),',',YLS(SIndex),','  &
!!                    ,ZLS(SIndex),',',SInter,',',XP(SIndex,SInter),',''&
!!                                                    ,YP(SIndex,SInter),',',ZP(SIndex,SInter), Rand(6)
!!112    Format(A2,A1,I2,3(A1,f6.3),A1,I2,3(A1,f6.3),x,f6.4)
!!        END IF
                        SIndex = SInter
                        Reflected = .True.
   ENDIF

 END SUBROUTINE AbsorptionReflection

 END MODULE EnergyAbsorbed_Reflected
```

```
 MODULE Distribution_Factors
!
!
 USE Global
 USE EnclosureGeometry
 USE EnergyBundleLocation
 USE IntersectionEnergy_Surface
 USE EnergyAbsorbed_Reflected
!
!
 IMPLICIT NONE
 CONTAINS
!
!
 SUBROUTINE Rad_Distribution_Factors
!******************************************************************************
!
! PURPOSE:                          Calculating the radiation distribution factor
!
!
!
!******************************************************************************
   IMPLICIT NONE
   INTEGER                    :: I,J,k,l, m,Index,IOS,NEACMB,NAreaCMB,N_C_S_CMB
           INTEGER, ALLOCATABLE, DIMENSION(:) :: NTA,NTR,NTRR,NTRcmb,NTRRcmb,CMBCOUNT
           INTEGER, ALLOCATABLE, DIMENSION(:) :: CMBSURFS, ICOMBSURF,COMBSURF
   REAL(Prec2) :: ECMB
           INTEGER, ALLOCATABLE, DIMENSION(:,:) :: NAEnergyDummy
!
!          NTA                        =          Number of total energy nundles absorbed in the enclosure
!                                                for an energy emitted from a given surafce
!          NTR                        =   Number of total energy nundles reflected in the enclosure
!                                                     for an energy bundles emitted from a given surafce
!
!           NTRR                =   Number of energy nundles re-reflected in the enclosure
!                                                     for an energy bundles emitted from a given surafce
!           NTAcmb              =          Number of total energy nundles absorbed in the enclosure
!                                                     for an energy bundles emitted from a given surafce after
!                                                     surface combination
!           NTRcmb              =          Number of energy nundles reflected in the enclosure
!                                                     for an energy bundles emitted from a given surafce after
!                                                     surface combination


            ALLOCATE (NTA(NSurf),NTR(NSurf),NTRR(NSurf),COMBSURF(NSurf),  &
                                           NAEnergyDummy(NSurf, NSurf), STAT = IOS)
!
!   Indentify number of surfaces combinations
            DO J =1, NSurf
             DO m =1, NSurf
             IF (J == CMB(m))Then
                        N_SCMB =  N_SCMB + 1
             ELSE
             ENDIF
   END DO
            END DO
!
            NSurfcmb = NSurf - N_SCMB                         ! Number of Surfaces after combined
!
            ALLOCATE (NTAcmb(NSurfcmb),NTRcmb(NSurfcmb),NTRRcmb(NSurfcmb), &
                   NAEnergyCMB(NSurfcmb, NSurfcmb),CMBCOUNT(NSurfcmb),  &
                                  ICOMBSURF(N_SCMB),CMBSURFS(N_SCMB),AreaCMB(NSurfcmb),STAT = IOS)
!
            DO I = 1, NSurf
             NTA(I) = 0
             NTR(I) = 0
             NTRR(I)= 0
   END DO
!
   DO I = 1, NSurf
             NTA(I) = TCOUNTA(I)
   END DO

            DO m = 1, NSurfcmb
             DO J = 1, NSurfcmb
              NAEnergyCMB(m,J) = 0
    END DO
             NTAcmb(m) = 0
             NTRcmb(m) = 0
             NTRRcmb(m)= 0
   END DO

   DO I =1, NSurf
             DO Index = 1, NSurf
              RAD_D_F(I,Index)=NAEnergy(I,Index)/Real(NTA(I))
    END DO
   END DO

            Return
  END SUBROUTINE Rad_Distribution_Factors
!
  END MODULE Distribution_Factors
```

```
------------------------------------------------------------------------------------------------------------------------------------------

MODULE EnergyBalance
!
 USE Global
 USE EnclosureGeometry
 USE EnergyBundleLocation
 USE IntersectionEnergy_Surface
 USE EnergyAbsorbed_Reflected
 USE Distribution_Factors
!
 IMPLICIT NONE
 CONTAINS
!
 SUBROUTINE Radiation_Balance
!*****************************************************************************
!
! PURPOSE:                          Calculating the net radiation flux at each surface using
!                                    the gray view factor or the radiation distribution factor
!
!*****************************************************************************
   IMPLICIT NONE
   INTEGER                 :: I,J,k,Index,IOS,LWL,UPL
         INTEGER, ALLOCATABLE, DIMENSION(:) :: Eb
   REAL(Prec2) :: SIGMA, EBSUM, T

         SIGMA = 5.67E-8              ! Stephane Boltzmann constant
!        EBSUM  =         Is product sum of emissivities and balck body emissive power
!                                For each surface
!  LWL          =         The lower surface index for which the temperatures to read is
!                                applicable
!  UPL          =         The upper surface index for which the temperatures to read is
!                                applicable
!  T            =         Temperature of the surfaces, K

         ALLOCATE(Ts(NSurf),EB(NSurf),QFLUX(NSurf),Q(NSurf),STAT = IOS)

!  Read and assign surface Temperatures
!        DO I = 1, NSurfcmb
   DO I = 1, NSurf
    Read(7,*)LWL, UPL, T
          IF(LWL == "0")EXIT
          DO J = LWL, UPL
           Ts(J) = T
          END DO
   END DO

         DO J = 1, NSurf
          EB(J) = SIGMA*(Ts(J)**4)
         END DO
   DO I =1, NSurf
         EBSUM = 0.0
         DO J = 1, NSurf
          EBSUM = EBSUM + RAD_D_F(I,J)*EB(J)
         END DO
          QFLUX(I) = EMIT(I)*EB(I) - EMIT(I)*EBSUM
          Q(I) = Area(I)*QFlux(I)
   END DO
```

```
  END SUBROUTINE Radiation_Balance
  END MODULE EnergyBalance
```

```
PROGRAM Main_MonteCarlo

   USE Global
   USE EnclosureGeometry
   USE EnergyBundleLocation
   USE IntersectionEnergy_Surface
   USE EnergyAbsorbed_Reflected
   USE Distribution_Factors
   USE EnergyBalance
   USE OutPut

   IMPLICIT NONE
   INTEGER                          ::I,J,k,IOS,Index
!
!  Initialize the CPU time
   CALL CPU_TIME(TIME1)
!
!  Assign number of Energy Bundles emitted per surface
!
   NBundles = 1000000
!
!    Open (Unit=2, file='Barn2.dat',status='old',Action='Read',IOSTAT=IOS)
!    Open (Unit=2, file='Barn1.dat',status='old',Action='Read',IOSTAT=IOS)
!    Open (Unit=2, file='Geometry1.dat',status='old',IOSTAT=IOS)
     Open (Unit=2, file='LShape.vs3',status='old',IOSTAT=IOS)

!            Open (Unit=3, file='Barn2.out',status='unknown',IOSTAT = IOS)
!            Open (Unit=3, file='Barn1.out',status='unknown',IOSTAT = IOS)
!   Open (Unit=3, file='Geometry1.out',status='unknown',IOSTAT = IOS)
             Open (Unit=3, file='LShape.out',status='unknown',IOSTAT = IOS)
!    Open (Unit=5, file='TangentV.out',status='unknown',IOSTAT = IOS)
             Open (Unit=4, file='Iters_point.out',status='unknown',IOSTAT = IOS)
             Open (Unit=7, File='barn.TK',status='old',IOSTAT = IOS)
!
!
    Call CalculateGeometry()
    Call InitializeSeed()
    Call AllocateArrays()
             Call InitializeArrays()
!
    Do SIndex = 1, NSurf
      CALL Calculate_SurfaceEquation()
              CALL Calculate_Area_Surfaces()
!             CAll Calculate_Length_Width_Height()
      CALL TangentVectors()
    END DO
!

!  Initiaize the logical variable for the first emitted energy bundle
   Reflected = .False.
!
    DO SIndexR = 1, NSurf
     SIndex  = SIndexR
     DO NTrials = 1, NBundles
!  Calculating source locations for each energy bundle
                Call EnergySourceLocation()
```

```
!
!  Calculate the direction of emitted energy bundle
               CALL DirectionEmittedEnergy()
!
!  Check the intersection points and determine the correct one
             CALL CheckingIntersection()
!
!  Determine whether the energy bundle is absorbed or reflected
             CALL AbsorptionReflection()
    END DO
   END DO
!
!  Calculate the radiation distribution factor
   CAll Rad_Distribution_Factors()
!
!  Calculate the heat balance of the enclosure
             Call Radiation_Balance
             CALL CPU_TIME(TIME2)

!  Write Results to a file
             Call Print_ViewFactor_HeatFlux

!
    Close(Unit = 3)
              Close(Unit = 4)
              Close(Unit = 5)
              Close(Unit = 7)
              STOP
END PROGRAM Main_MonteCarlo
```