MAE 5823

Monte Carlo Ray Tracing Program Improvements

Team One Final Report

John Holman, Rachel Spitler, and Sudha Sikha

12/4/2012

The purpose of this project was to modify the original MCRT program in order to incorporate different surface types. The distribution factor for each surface was calculated based on the Monte Carlo Ray Tracing algorithm and the direction handling algorithm for the specular emitting surfaces. The most major change in the program was the addition of an algorithm that allowed for specular emission, reflection, and absorption. The original MCRT program only worked if the surface model was either a rectangle or a triangle. The modified MCRT program works for all types of quadrilaterals and triangles, as well as properly handling the additional surface types. The program was tested for a number of different surface types and assumed geometry, and the results were visualized based on the RTVT (Ray Tracing Visualizing Tool).

Contents

1. Introduction

Monte Carlo methods involve using random numbers to generate results over a large distribution. The Monte Carlo Ray Tracing (MCRT) program uses the Monte Carlo method to determine the direction of rays being emitted from a surface. Each of these rays represents a "bundle" of energy, which will be tracked around the enclosure. The purpose of this program is to determine the effects of radiation heat transfer in the enclosure by recording what happens to each energy bundle that is emitted. With the MCRT methodology, very complicated situations can be examined, situations that would have been very difficult to handle using any other method. With the advances in computer technology, the computationally expensive Monte Carlo Method is becoming more and more prevalent in radiation evaluation. This leads to an improvement in functionality of MCRT programs in general, allowing for more accurate and informative analyses.

This paper outlines a number of changes to an existing MCRT program. The modifications to the program include, but are not limited to, handling of spectral radiation, the addition of surface types beyond simple gray diffuse, and an improved emission algorithm. These changes are verified for accuracy using multiple test cases. Also, further improvements to the program are suggested.

1.1 Objectives

The objective of this project was to increase the functionality to an existing MCRT program written by Bereket Nigusse (Nigusse 2004). The main functional change was to be the addition of specular radiation to the program, for the modeling of light from the sun. Along with the inclusion of specular radiation, the addition of surface types, such as gray diffuse, specular and diffuse emitting, specular and diffuse reflecting, specular reflecting only, and diffuse reflecting only, was required. Also, in the original program, the quadrilateral emission algorithm only worked for rectangles, so it needed to be modified so that non-rectangular quadrilaterals could be used.

2. Literature Review

There are many applications for Monte Carlo Ray tracing. Some of the less complicated applications are those for modeling simple radiation problems. Mirhosseini and Saboonchi cover the use of a MCRT program to determine the view factor from a 3D strip to a circular cylinder for applications in heating and cooling processes (Mirhosseini and Saboonchi 2011). Hong and Welty use a variety of surfaces to examine the distribution of heat flux within an enclosure (Hong and Welty 1999). Shuai, Tan, and Xia cover the application of the MCRT algorithm to evaluate stray radiation reaching optical devices (Xia, Shuai and Tan 2004). Similar to the case of stray optical radiation is the case of far infrared radiation. Tanaka, et al. wrote a paper covering the use of MCRT to evaluate this specific type of radiation involved in the heating in different types of airflow (Tanaka, et al. 2006).

Participating media is a very integral part of many heat transfer situations. MCRT can be utilized to further evaluate this phenomenon. Soucasse, Riviére, and Soufani examine this application for quasi-isothermal media by computing the difference between the radiative field and the equilibrium radiative field for the medium (Soucasse, Riviere and Soufiani 2012). Wang and Modest also examine the

phenomenon of participating media by examining non-gray radiation in inhomogeneous media (Wang and Modest 2007).

Insulation is often regarded as a conduction element, but on a microscopic level, it functions on principles of radiation. Arambakam, Vahedi Tafreshi, and Pourdeyhimi examine steady state heat radiation in fibrous insulation using the MCRT method (Arambakam, Vahedi Tafreshi and Pourdeyhimi 2012). Schweiger, Costa, and Segarra examine another form of insulation by using the MCRT algorithm to investigate the radiative heat transfer component of honeycomb transparent insulation (Schweiger, et al. 1999).

There are a multitude of other applications for MCRT in the context of radiation heat transfer. Maurente, Vielmo, and Franca examine use MCRT to solve for the radiation heat transfer in media with spectrally dependent properties, utilizing the absorption line blackbody distribution function to represent the spectrally dependent properties (Maurente, Vielmo and Franca 2007). Luo, Hauer, and Day examine the radiation heat transfer loads in a cryopump using MCRT for further design (Luo, Hauer and Day 2012). Wang, et al. examine the use of MCRT to investigate radiation interaction with jet flames (A. Wang, M. F. Modest, et al. 2007). MCRT can also be used to examine natural phenomena, such as wild fires; Bilaud, et al. use MCRT to determine the safe distance from the leading edge of the wildfire (Billaud, et al. 2010). Solar reactors are another good application for MCRT; in their paper, Villafán-Vidales et al. use MCRT to simulate a cavity solar reactor (Villafan-Vidales, Dehesa-Carrasco and Romero-Paredes 2008).

3. Original Program

The original MCRT program was written in 2004 by Bereket Nigusse (Nigusse 2004). This program could calculate the distribution factors and heat flux through each surface of an enclosure with rectangular or triangular surfaces. These values were calculated by simulating the path and behavior of a specified number of energy bundles randomly distributed around each surface. The user could implement these values in another program, or simply use them to calculate the radiation from one surface to another by hand. The heat flux values were created using the calculated distribution factors and the inputted surface temperatures from a separate file. These values were also balanced so that the net heat gain from each surface was 0.

3.1 Main Algorithm

The main algorithm for the MCRT program is shown below in Figure 1.

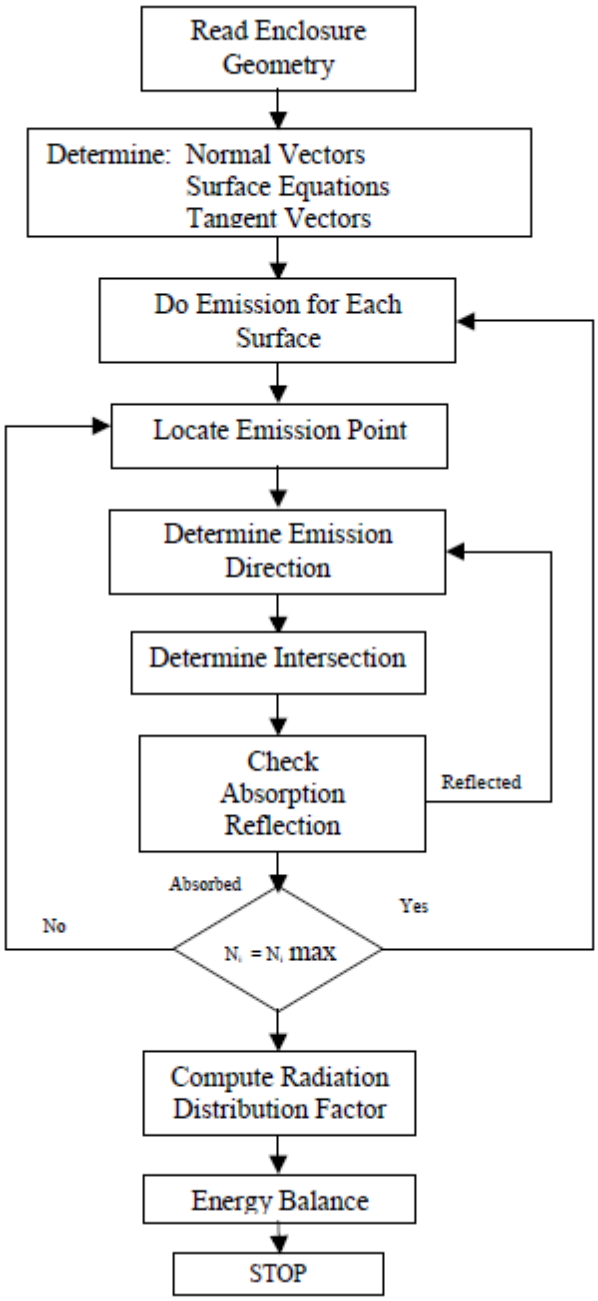

Figure 1: MCRT Program Algorithm (Nigusse 2004)

The most integral points in the algorithm are the determination of the absorption or reflection of each bundle. This is the "turning point" of the algorithm, determining whether to emit the next bundle or to follow the current one. The location of the bundle at each point of its journey through the enclosure is logged by the surfaces that it interacts with. The counters for each surface are then used to determine the distribution factors from one surface to another. These values are output in matrix form for easy comprehension and application by the user. Before any of these operations happen, though, the input file must be read and the basic information about each surface must be determined. This information

includes the location of the surface and the normal vector to the surface. These values are used to determine whether or not the energy bundles interact with each surface, and, if they do, what happens to the bundle. The portion of this algorithm that makes it a Monte Carlo Method is the incorporation of random numbers into the emission and absorption algorithms. Each of these random numbers play a role in determining the location and direction of the bundle and, in the case of absorption, whether or not the ray is absorbed. All of these components together form an useful algorithm for the calculation of distribution factors in complex geometric enclosures.

3.2 Capabilities

The capabilities of the original MCRT program are very limited. The original program could only calculate and determine the effects of diffuse radiation inside the enclosure. Also, the program could not use any shapes that were not a rectangle or a triangle. In addition, the surfaces had to be located on one of the major axes (x, y, z) or "rotated" about one of the major axes. If any of these criteria were not met, then the program would crash and return values for bundles being emitted outside of the enclosure that go off into "space."

4. Changes to MCRT Program

This section outlines the improvements and changes made to the code in order to fulfill the required objectives.

4.1 Surface Definitions

The objectives of the project called for the addition of new surface types to be used in the program. The surface definitions given in the project assignment were followed. There were five surfaces: DIF, SDE, SDR, SRO, and DRO.

DIF (Diffuse emitting) is the normal surface type for the MCRT program. It models a gray diffuse surface that both emits and reflects.

SDE (Specular and Diffuse Emitting) denotes a surface that emits both diffusely and specularly, but only reflects diffuse irradiation. For this surface type, the specular emission direction needs to be specified for the surface in the form of a direction vector.

SDR (Specular and Diffuse Reflection) is a surface that emits diffusely, but reflects both diffuse and specular irradiation. This surface requires the explicit definition of both the specular and diffuse reflectance of the surface.

DRO (Diffuse Reflection Only) represents a surface that is non-emitting, but reflects diffuse irradiation.

SRO (Specular Reflection Only) denotes a surface that is also non-emitting, but reflects specular and diffuse irradiation. This surface requires the explicit definition of the specular and diffuse reflectance.

4.2 Input Processing

A new subroutine was added to bring in the additional surface information. After the original code read in the first two information blocks, the new subroutine was called to read in the first two entries, surface number and type, in each line of the surface type block. Once the end was reached, the code went back to the beginning and skipped over lines until it got to the surface type block again. Based on what the surface type was, the program either read in the extra information needed or went on to the next line. Figure 2 shows the flow chart for this logic.

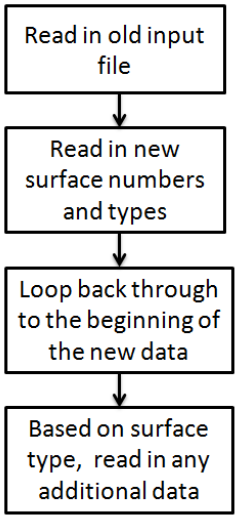

Figure 2: Flow Chart for Input Processing

4.3 Emission Algorithm

For diffuse emission, the original MCRT program generated random emission points and locations, which were utilized for all diffuse emission in the update. For the cases of specular emission, though, a specific direction was listed in the input file; the actual points are still randomly generated, but the direction remains constant for every emission.

The original code would only work if the quadrilateral was defined as a rectangle, otherwise points would be drawn outside of the shape. Modifications were made in how the surfaces were defined, so that any convex quadrilateral with sets of vertices in the same xyz plane can be used. Instead of automatically assuming that every surface of the enclosure was defined by three or four vertices that remained in the same xyz plane, the modified code generates three vectors to check. The three vectors are two vectors along the edges from the first vertex and then one from the first to third vertex.

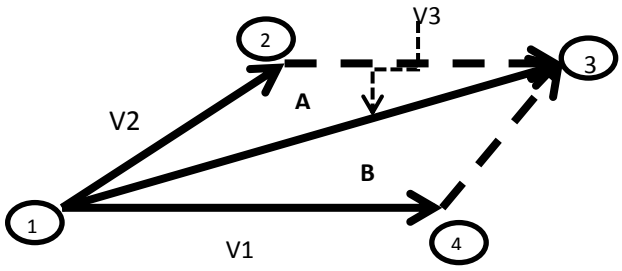

Figure 3: Quadrilateral Algorithm Example

The emission point of a triangle was determined using the following set of equations from Dr. Spitler's notes (Spitler 2012).

$$P_e = P_o + u\vec{a} + v\vec{b}$$

Equation 1: Random Emission Point

$$u = 1 - \sqrt{1 - R_1}$$

Equation 2: Random Number Component

$$v = (1 - u)R_2$$

Equation 3: Random Number Component

Where:

P_e = Point of Emission
P_o = Point of Origin
a = Vector
b = Vector
R_1 = Random Number
R_2 = Random Number

Then, instead of treating a rectangular surface as a single piece, it is treated as two triangles. To determine which triangle will emit the point, a random number is selected. If the random number is greater than 0.5, then the point will be emitted from the top triangle, triangle "A." If the random number is less than or equal to 0.5, then the point will be emitted from the bottom triangle or triangle "B." After the triangle is selected, then the vector from the quadrilateral (V1 or V2) and the bisection vector (V3) are used in conjunction with the triangular emission algorithm to determine the point of emission.

4.4 Specular Reflection Algorithms

The specular ray is emitted from the first surface and is intersected by a second surface. This second surface either absorbs or reflects the ray; if it reflects, the new reflection direction must be calculated. The difficult part of this procedure was determining the angle in three directions. The relationship between the incoming ray, the normal vector to the surface, and the outgoing ray was listed by Nancy Pollard (Pollard 2004) in the following format:

$$r = 2(I \cdot n)n - I$$

Equation 4: Reflection of a Ray

Where:

r = the reflected ray leaving,
I = the negative of the incoming ray, and
n = the normal vector to the surface.

I is taken as the negative of the incoming ray since the equation considers *I* an outgoing ray, as seen in Figure 3.

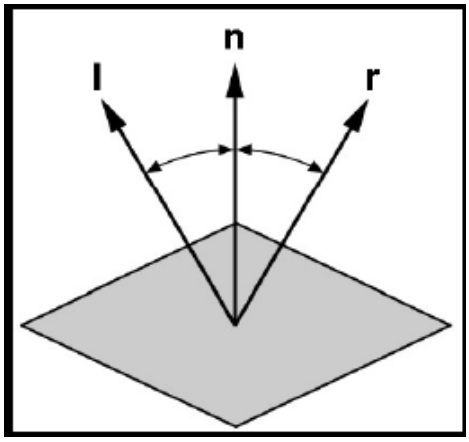

Figure 4. Diagram of Ray Reflection (Pollard 2004)

The above relationship was implemented in the MCRT code for cases of both specular reflection and re-reflection. In the case of rereflection, the incoming ray to the surface is the same as the reflected ray leaving the previous surface.

4.5 Specular Distribution Factors

Distribution factors for the diffuse rays were calculated in the original MCRT program by merely taking the number of rays absorbed by any given surface and dividing by the number of rays emitted from the emitting surface. The specular distribution factors were calculated the same way as the diffuse ones; the only difference was that three different sets of specular distribution factors had to be calculated instead of just one, as in the diffuse case. The three different sets were for total specular radiation, reflected specular radiation, and non-reflected specular radiation. The total specular radiation distribution factors counted all the specular rays that were emitted, whether they were absorbed without bouncing or after bouncing off of surfaces. The reflected specular radiation distribution factors only accounted for the specular rays that were absorbed after being reflected once or more. The non-reflected specular radiation factors accounted only for the specular rays that were absorbed the first time they hit a surface. As such, the addition of the number of reflected specular rays and the number of non-reflected specular rays result in the number of total specular rays.

4.6 Output Files

The output files generated are listed below, along with an explanation of each:

MCOutput.txt – This includes the total number of diffuse energy bundles emitted from and absorbed by each surface, the total diffuse distribution factors, and the general surface properties from the input file.

It also reports the heat flux and heat transfer rate for each surface; however, these have not been validated for the new revision.

SpecularDF.out – This includes the total number of specular energy bundles emitted from and absorbed by each surface, and the total specular distribution factors. This file reports the overall specular bundles and distribution factors, regardless of whether they are absorbed on the first intersection, or after reflection or re-reflection.

SpecReflecDF.out – This includes the number of reflected specular energy bundles emitted from and absorbed by each surface, and the reflected distribution factors. This file does not report the number of bundles or the distribution factors for the rays that are absorbed without reflection, only for those that are reflected once or more before absorption.

SpecWRDF.out – This includes the number of direct specular energy bundles emitted from and absorbed by each surface, and the direct specular distribution factors. This file does not report the bundles or distribution factors for rays that are reflected or re-reflected, only the number of bundles and distribution factors for the rays that are absorbed by the first surface they intersect with.

Logfile.out – This reports the emission, reflection, and absorption points for every ray emitted. This file can be combined with the first two blocks of the input file to create an input file for the RTVT program. The user can choose to not generate this file by including a 0 on the second line of the parameters.txt file; likewise, the user can choose to generate this file by replacing the 0 with a 1. Either a 0 or a 1 must occur on the second line of the parameters.txt file for the program to run, though.

5. Verification

To evaluate the program so that it could be verified for the users, some test cases were needed for validation. The following cases were introduced to provide verification that the program works with many different cases, particularly with varied geometries.

5.1 General Issues

One of the most common errors that occurred was the emission and reflection of rays outside of the enclosure. This was solved by fixing how the surfaces were defined; before, the program was not checking to make sure that the emission or intersection point was actually on the surface it was supposed to be.

Specular rays were reflecting back along the same direction they were being emitted from; this was fixed by the correct implementation of the ray reflection algorithm, as shown earlier.

The program was also checked for a maximum number of surfaces that the program could handle. Only one variable, a surface property array, was found hardcoded to limit the number of surfaces; this was fixed by allocating the array using the total number of surfaces. Another team had a structure with at least 80 surfaces, and the modified MCRT program was able to handle that scenario. On a practical level,

the simulation time would put an effective limitation on the number of surfaces used, but the code itself should be able to handle almost any enclosure.

5.2 Test Cases

Several test cases were run in order to verify the program and its output distribution factors, particularly for the specular surfaces. A log file is created by the program after every run with the corresponding input file. The log file actually shows the rays' direction from one surface to the other surface. This can be visualized by a ray tracing visualization tool (RTVT). The RTVT program reads a file with inputs such as the vertices, surfaces, and the emission, reflection, and absorption points for every ray listed in the log file in the .dat format. The tool generates a line image of the enclosure, and then the emitted and reflected rays within the enclosure can be viewed. Using this tool, one can view the specular and diffuse rays reflected and emitted from the surfaces.

Some test cases were created for this project – different input files with different geometry and surfaces. The input files were run through the modified MCRT program and the corresponding output log files are analyzed using the RTVT program. The rays emitted and the rays reflected for all these test cases lie within the enclosure.

5.2.1 Trapezoidal

One of the cases used to test the MCRT program was that of a square pyramid with the tip chopped off. With this configuration, the sides become trapezoids, something that the original MCRT program would not be able to handle. Using the RTVT program to visualize the path of the rays, the figures below were generated.

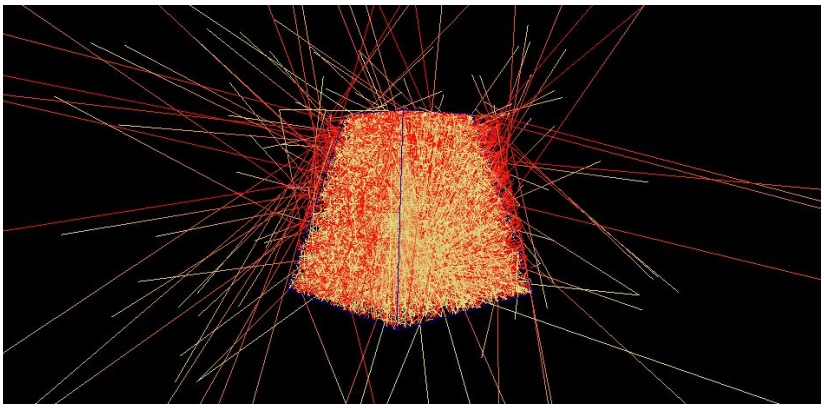

Figure 5: Trapezoidal Verification Case with old MCRT

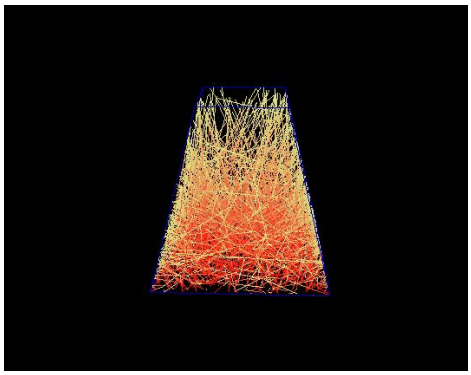

Figure 6: Trapezoidal Verification Case with new MCRT

These figures plainly show the improvements in the program. Using the old MCRT program and algorithms, there are a significant number of rays leaving the enclosure, whereas in the newly modeled enclosure, there are no rays leaving. The new algorithms obviously improve the accuracy of the simulation.

5.2.2 Equinox House

The Equinox House is a low energy, sustainable dwelling in Illinois (Newell Instruments Inc. 2012). This presented an interesting case for the MCRT program to model. An input file for a building with similar geometry to the house was created for testing purposes.

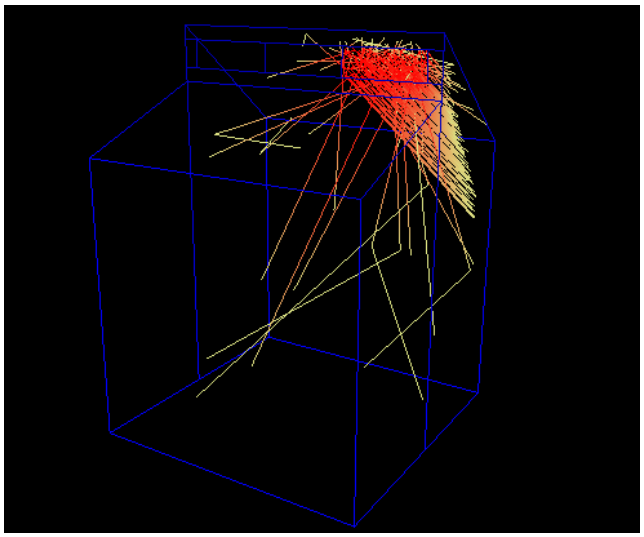

Figure 7: Specular Irradiation from one Surface in Equinox House Example

This figure shows that the specular rays from one of the windows (originating end is red) are absorbed by the slanted roof and side wall of the enclosure, while the diffuse rays are absorbed and reflected. A more in-depth RTVT picture would better show that the rays from the window are almost completely absorbed by the slanted roof and wall behind them, confirming a common-sense result as all the surfaces besides the windows are modeled as diffuse-gray here.

6. Conclusions and Recommendations

At the completion of any project, there are certain things that are learned and certain things that can be done in the future to improve the work from the project.

6.1 Future Work

Although many things have been added to increase the functionality of the MCRT program, there is much more that can be done to improve its usability and allow for a wider array of implementations.

6.1.1 Improvements for Debugging

One idea to improve debugging of the input file is to include a subroutine that prints out a log file with just the normal vectors at the center of every surface. This file can then be run in RTVT and, if any of the vectors are pointed the wrong direction, will indicate to the user that the surface vertices are defined in the wrong order.

In addition, there should be some form of error handling in the code, where the program outputs a warning or error message when something goes wrong. This message would tell the user where the error occurred and why there is an error. For instance, the program could return a message telling the user that there is an error in their input file, in the area of the extra information for one of the surfaces.

There is also room for improvement in the input file processing. The program should be able to read input files with extra lines or spaces, or at least produce an error telling the user where the input file is wrong. Also, the format of the input file could be changed to make it more streamlined, allowing for shorter, smaller input files to be used.

Additionally, the radiation heat flux and heat transfer calculations should be verified and fixed as needed. In the original version of the program, the calculations appeared to be accurate. However, in the modified version of the program the heat transfer rates are output, but they are not accurate. This is a nice addition to the distribution factors that are traditionally output and should be included in the output in future versions.

6.1.2 Restructuring and Rewriting Code

One of the other major improvements to the code could be a major re-write of the original source code. There are many places where the code could be reduced to accomplish the same goal while decreasing the computation time. This reduction in computation time would allow for the program to be used on larger enclosures and for more complex applications.

6.1.3 Other Surface Types

In the project assignment two additional surface types were listed, but not assigned for this semester. These two surface types are defined here and should be considered in future work.

SND (Specular and Non-uniform Diffuse emitting) would be a surface that emits both specularly and diffusely, but the diffuse emission would be non-uniform. In addition to the direction vector for the specular emission, a distribution factor for the diffuse emission would be needed.

TAD (Transmitting, Angle Dependent) would only emit diffuse radiation, but would depend on the ray's incidence angle to compute reflection, transmission, and absorption.

6.2 Conclusions

Based on the validation performed and the use of the program by project teams in MAE 5823, the modified program works in its intended capacity. It calculates the diffuse and specular distribution factors within the enclosure and handles both diffuse and specular rays. However, there is definitely more work to be done with the program to improve its functionality. When combined with the RTVT program, this program provides a good visualization of typically invisible phenomena that occur within an enclosure.

Works Cited

Arambakam, R., H. Vahedi Tafreshi, and B. Pourdeyhimi. 2012. Analytical Monte Carlo Ray Tracing simulation of radiative heat transfer through bimodal fibrous insulations with translucent fibers. *International Journal of Heat and Mass Transfer*, 55: 7234-7246.

Billaud, Y., A. Kaiss, J. L. Consalvi, and B. Porterie. 2010. Monte Carlo estimation of thermal radiation from wildland fires. *International Journal of Thermal Sciences*, 50: 2-11.

Hong, Seung-Ho, and James R. Welty. 1999. Monte Carlo simulation of radiation heat transfer in a three-dimensional enclosure containing a circular cylinder. *Numerical Heat Transfer, Part A: Applications: An International Journal of Computation and Methodology*, 36:4: 1-17.

Luo, Xueli, Volker Hauer, and Christian Day. 2012. Monte Carlo calculation of the thermal radiation heat load of the ITER pre-production cryopump. *Fusion Engineering and Design*, 87: 603-607.

Maurente, A., H.A. Vielmo, and F.H.R. Franca. 2007. A Monte Carlo implementation to solve radiation heat transfer in non-uniform media with spectally dependent properties. *Journal of Quantitative Spectroscopy & Radiative Transfer*, 108: 295-307.

Mirhosseini, M., and A. Saboonchi. 2011. View factor calculation using the Monte Carlo method for a 3D element to circular cylinder. *International Communications in Heat and Mass Transfer*, 38: 821-826.

Naeimi, H., and F. Kowsary. 2011. Simplex ray-object intersention algorithm as ray tracer for Monte Carlo simulations in radiative heat transfer analysis. *International Communications in Heat and Mass Transfer*, 38: 646-651.

Newell Instruments Inc. 2012. Equinox House. http://buildequinox.com/projects/equinox-house/

Nigusse, Bereket A. 2004. Radiation Heat Transfer in Enclosure Using the Monte Carlo Method. Project Report, Stillwater, OK.

Pollard, Nancy. 2004. Course 15-462: Computer Graphics, Spring 2004 Slides. http://graphics.cs.cmu.edu/nsp/course/15-462/Spring04/slides/13-ray.pdf

Schweiger, H., A. Oliva, M. Costa, and C.D. Perez Segarra. 1999. A Monte Carlo method for the simulation of transient radiation heat transfer: Application to compound honeycomb transparent insulation. *Numerical Heat Transfer, Part B: Fundamentals: An International Journal of Computation and Methodology*, 35:1: 113-136.

Soucasse, Laurent, Philippe Riviere, and Anouar Soufiani. 2012. Monte Carlo Methods for Radiative Transfer in Quasi-Isothermal Participating Media. *Journal of Quantitave Spectroscopy & Radiative Transfer*, 1-9.

Spitler, Jeffrey D. 2012. Monte Carlo Ray Tracing Methods in Radiation Heat Transfer. Class Notes. Oklahoma State University, Stillwater, OK.

Tanaka, F., K. Morita, K. Iwisaki, P. Verboven, N. Scheerlinck, and B. Nicolai. 2006. Monte Carlo simulation of far infrared radiation heat transfer: theoretical approach. *Journal of Food Process Engineering*, 29: 1-13.

Villafan-Vidales, H.I., Arancibia-Bulnes, C.A., U. Dehesa-Carrasco, and H. Romero-Paredes. 2008. Monte Carlo radiative transfer simulation of a cavity solar reactor for the reduction of cerium oxide. *International Journal of Hydrogen Energy*, 34: 115-124.

Wang, Anquan, and Michael Modest. 2007. Spectral Monte Carlo Models for Nongray Radiation in Inhomogenious Participating Media. *International Journal of Heat and Mass Transfer*, 50: 3877-3889.

Wang, Anquan, Michael F. Modest, Daniel C. Haworth, and Liangyu Wang. 2007. Monte Carlo simulation of radiative heat transfer and turbulence interactions in methane/air jet flames. *Journal of Quantitative Spectroscopy & Radiative Transfer*, 109: 269-279.

Xia, Xin-Lin, Yong Shuai, and He-Ping Tan. 2004. Calculation Techniques with the Monte Carlo method in stray radiation evaluation. *Journal of Quantitative Spectroscopy & Radiative Transfer*, 95: 1-11.

Appendix A: Modified Code

```
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 Specular Trials
      INTEGER :: NTrialsD                         ! Number of Diffuse 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 :: SpIndex      !JH: Specular Index, 1 for specular Radiation,
0 for diffuse
      INTEGER ::  TCountSpecR     !RS: Determines whether or not a reflected
ray is absorbed
!
!
    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(:,:) :: NAEnergyS       ! Total Absorbed
Energy Counter, Specular
    INTEGER , ALLOCATABLE,DIMENSION(:,:) :: NAEnergyR       ! Reflected and
Rereflected Energy Counter, Specular
    INTEGER , ALLOCATABLE,DIMENSION(:,:) :: NAEnergyWR      ! Unreflected
Energy Counter, Specular
```

```
!
      INTEGER , ALLOCATABLE,DIMENSION(:) :: TCOUNTA   ! Number of absorbed
energy bundle
      INTEGER , ALLOCATABLE,DIMENSION(:) :: TCOUNTR   ! Number of reflected
energy bundle
      INTEGER , ALLOCATABLE,DIMENSION(:) :: TCOUNTRR  ! Number of rereflected
energy bundle
      INTEGER , ALLOCATABLE,DIMENSION(:) :: NTOTAL    ! Total Number of
Energy bundles emitted
      INTEGER , ALLOCATABLE,DIMENSION(:) :: NTACMB    ! Total Number of
Energy bundles emitted

      ! after surface combinations

    Integer, allocatable,dimension(:) :: TSpecA         !Total Number of
specular bundles absorbed on first bounce
    Integer, allocatable,dimension(:) :: TSpecR         !Total Number of
specular bundles reflected
    Integer, allocatable, dimension(:) :: TSpecRR       !Total Number of
specular bundles rereflected
!
      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  ! Reference Point
      REAL(Prec2) :: Rand(7)  ! 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
      Logical :: WriteLogFile  ! Flag to indicate whether log file
                               ! should be written.

! JDS 11-10-2006
!
      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  ! Diffuse
Radiation Distribution Factor
    REAL(prec2), ALLOCATABLE, DIMENSION(:,:) ::  RAD_D_S    ! Specular
Radiation Distribution Factor
    REAL(prec2), ALLOCATABLE, DIMENSION(:,:) ::  RAD_D_R    ! Reflected
Specular Radiation Distribution Factor
    REAL(prec2), ALLOCATABLE, DIMENSION(:,:) ::  RAD_D_WR   ! Non-Reflected
Specular 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 in Surface
equation
!
    CHARACTER(LEN=3), ALLOCATABLE, DIMENSION(:) ::  SurfaceType  ! Surface
Type Array
    REAL (prec2), ALLOCATABLE, DIMENSION(:) ::  DirectionX   ! X Vector
Coordinates for SDE type
    REAL (prec2), ALLOCATABLE, DIMENSION(:) ::  DirectionY   ! Y Vector
Coordinates for SDE type
    REAL (prec2), ALLOCATABLE, DIMENSION(:) ::  DirectionZ   ! Z Vector
Coordinates for SDE type
    REAL (prec2), ALLOCATABLE, DIMENSION(:) ::  SpecReflec   ! Specular
Reflectance
    REAL (prec2), ALLOCATABLE, DIMENSION(:) ::  DiffReflec   ! Diffuse
Reflectance

    INTEGER :: NCount   !Counter for specular reflection
    INTEGER :: NCountd  !Counter for diffuse reflection
    INTEGER :: OldSurface   !Keeps track of the previous emitting surface for
the case of rereflections

 END MODULE Global

PROGRAM Main_MonteCarlo
    !Program and Modules created by Bereket Nigusse, Fall 2004 for MAE 5823
    !Program and Modules updated and modified November 2012 by
    !John Holman, Rachel Spitler, and Sudha Sikha for MAE 5823

   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, logfileint
!
!  Initialize the CPU time
   CALL CPU_TIME(TIME1)
!
!  Assign number of Energy Bundles emitted per surface
!   JDS 11-9-2006 Replace this fixed number with an input from
!   a file, set below.
!   NBundles = 1000000

! JDS 11/08/2006 Use generic filenames:
!                                     For input:            input.vs3
!                                     For output:           MCoutput.txt
```

```
!                                     For temperatures in K: input.TK
!                                     For number of bundles: parameters.txt
    Open (Unit=2, file='input.vs3',status='unknown',Action='Read',IOSTAT=IOS)
      Open (Unit=3, file='MCoutput.txt',status='unknown',IOSTAT = IOS)
!Diffuse bundles and distribution factors
      Open (Unit=4, file='logfile.out',status='unknown',IOSTAT = IOS)
!Ray emission, reflection, and absorption points
      Open (Unit=7, File='input.TK',status='unknown',IOSTAT = IOS)
!Surface temperatures
    OPEN (Unit=6, File='SpecularDF.out',status='unknown', IOSTAT=IOS)
!Total Specular bundles and distribution factors
    OPEN (Unit=9, File='SpecReflecDF.out',status='unknown', IOSTAT=IOS)
!Reflected and rereflected Specular bundles and distribution factors
    OPEN (Unit=10, File='SpecWRDF.out',status='unknown', IOSTAT=IOS)    !Non-
Reflected AKA absorbed on first intersection Specular bundles and
distribution factors
    OPEN (Unit=11, File='DebugFile.txt',status='unknown', IOSTAT=IOS)
!Lists rays that are not finding intersection points and whether they're
reflected or not

!   JDS 11-9-2006 Read Nbundles from file so that it can be changed
!   without recompiling.
!   JDS 11-20-2006 Read flag to determine whether or not logfile should
!   be written for use with RTVT

      Open (Unit=8, File='parameters.txt',status='old',IOSTAT = IOS)
      Read (8,*)NBundles
      read (8,*) logfileint
      if (logfileint == 1) then
         WriteLogFile=.true.
         else
         WriteLogFile=.false.
         end if
      Close(Unit = 8)
!
!
    Call CalculateGeometry()
    Call InitializeSeed()
    Call AllocateArrays()
      Call InitializeArrays()
!
    Do SIndex = 1, NSurf
      CALL Calculate_SurfaceEquation()
        CALL Calculate_Area_Surfaces()
      CALL TangentVectors()
    END DO
!
!  Initialize the logical variable for the first emitted energy bundle
   Reflected = .False.
!
    DO SIndexR = 1, NSurf
      SIndex  = SIndexR
!
!  The counter only counts the emitted and absorbed energy
             Ntrials = 0
 If (SurfaceType(SIndex) .EQ. 'SDE') then
```

```
    SpIndex = 1 !JH: This begins the Specular ray tracing

 Specular:    DO
!  Calculating source locations for each energy bundle
              Call EnergySourceLocation()
!
!  Calculate the direction of the 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()

!  If the number of absorbed energy bundles is the same as the number
!  of emitted energy bundles, exit the specular emission loop
!
                  If(NTrials==NBundles) Exit Specular
 END DO Specular

 SpIndex = 0 !JH: This begins the diffuse ray tracing for SDE surfaces

Ntrialsd=0
 Diffuse1:    DO
!  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()

!  Detrmine whether the enrgy bundle is absorbed or reflected
                CALL AbsorptionReflection()

!  If the number of absorbed energy bundles is the same as the
!  number of emitted energy bundles, exit the diffuse emission loop
!
                  If(NTrialsd==NBundles) Exit Diffuse1
 END DO Diffuse1

 ElseIF (SurfaceType(SIndex) .EQ. "DIF" .OR. SurfaceType(SIndex) .EQ. "SDR")
THEN
     NTrialsd=0
    !JH: This is the standard diffuse ray tracing
     Diffuse2:    DO

!  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 enrgy bundle is absorbed or reflected
                CALL AbsorptionReflection()

!  If the number of absorbed energy bundles is the same as the
!  number of emitted energy bundles, exit the diffuse emission loop
!
                  If(NTrialsd==NBundles) Exit Diffuse2
     END DO Diffuse2
 End if
    END DO
!
!  Calculate the radiation distribution factor
    Call Rad_Distribution_Factors()
!
!  Calculate the heat balance of the enclosure
      Call Radiation_Balance
!
!  Calculate the CPU Time
      CALL CPU_TIME(TIME2)

!  Write Results to a file
      Call Print_ViewFactor_HeatFlux

!
     Close(Unit = 3)
       Close(Unit = 4)
       Close(Unit = 7)
     CLOSE(Unit = 6)
     CLOSE(Unit =10)
     CLOSE(Unit =11)
       STOP
END PROGRAM Main_MonteCarlo

  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
      INTEGER, ALLOCATABLE, DIMENSION(:,:) :: NAEnergyDummy
!
!     NTA               =     Number of total energy bundles absorbed in the
enclosure
!                             for an energy emitted from a given surface
!     NTR               =   Number of total energy bundles reflected in the
enclosure
!                             for energy bundles emitted from a given surface
!
!     NTRR        =   Number of energy bundles re-reflected in the enclosure
!                             for energy bundles emitted from a given surface
!     NTAcmb            =     Number of total energy bundles absorbed in the
enclosure
!                             for energy bundles emitted from a given surface
after
!                             surface combination
!     NTRcmb            =     Number of energy bundles reflected in the
enclosure
!                             for energy bundles emitted from a given surface
after
!                             surface combination

      ALLOCATE (NTA(NSurf),NTR(NSurf),NTRR(NSurf),COMBSURF(NSurf),      &
                          NAEnergyDummy(NSurf, NSurf), STAT = IOS)
!
!   Identify number of surface 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  !Distribution Factors for Diffuse Rays
        DO Index = 1, NSurf
          IF (Real(NTA(I)) .EQ. 0) THEN
            RAD_D_F(I,Index)=0.0000
          ELSE
              RAD_D_F(I,Index)=NAEnergy(I,Index)/Real(NTA(I))
         END IF
      END DO
    END DO

    DO I =1, NSurf      !Distribution Factors for Specular Rays
        DO Index = 1, NSurf
           IF (Real(TSpecA(I)) .EQ. 0) THEN
            RAD_D_S(I,Index)=0.0000
          ELSE
              RAD_D_S(I,Index)=NAEnergyS(I,Index)/Real(TSpecA(I))
        END IF
      END DO
    END DO

    DO I =1, NSurf    !Distribution Factors for Reflected Specular Rays
        DO Index = 1, NSurf
           IF ((Real(TSpecR(I))+Real(TSpecRR(I))) .EQ. 0) THEN
            RAD_D_R(I,Index)=0.0000
          ELSE

RAD_D_R(I,Index)=NAEnergyR(I,Index)/(Real(TSpecR(I))+Real(TSpecRR(I))) !RS:
NAEnergyR is reflected energy
        END IF
      END DO
    END DO

    DO I =1, NSurf    !Distribution Factors for Non-Reflected (those absorbed
at the first intersection point) Specular Rays
        DO Index = 1, NSurf
           IF ((REAL(TSpecA(I))-Real(TSpecR(I))) .EQ. 0) THEN
            RAD_D_WR(I,Index)=0.0000
          ELSE
              RAD_D_WR(I,Index)=NAEnergyWR(I,Index)/(REAL(TSPecA(I))-
Real(TSpecR(I))) !RS: NAEnergyWR is non-reflected energy
        END IF
      END DO
    END DO

      Return
  END SUBROUTINE Rad_Distribution_Factors
```

```
!
  END MODULE Distribution_Factors

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
      Character (Len=12)  ErrorMessage
      Character (Len = 12) :: SubTitle
      Character (Len = 3)  :: Dummy
      Logical ReadFile
    CHARACTER (LEN = 12) :: SubTitle2   !RS: Second subtitle

!   The filename for the vertex and surface parameters of the rectangular
!   surface Enclosure
!   Reads numer of vertices and number of surfaces to allocate the array's
size
    i = 0;  j = 0
    Do
        Read (2,*)Dummy
        If(Trim(Dummy) == "v" .or. Trim(Dummy) == "V")Then
         i = i + 1
      Elseif(Trim(Dummy) == "!" )Then
           NVertex = i
      Elseif(Trim(Dummy) == "s" .or. 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)
    ALLOCATE(SurfaceType(NSurf),DirectionX(NSurf),DirectionY(NSurf), &  !RS:
Allocating arrays for surface types and directions
            DirectionZ(NSurf),SpecReflec(NSurf),DiffReflec(NSurf))

        !JH: Loop to set specular reflectances to 1 initially
    DO J=1,NSurf
        SpecReflec(J)=1
    End Do
!
    DO J = 1, NVertex
             Read (2,*)Vertex(J),V(J),XS(J),YS(J),ZS(J)
    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)
    END DO

    READ (2,*) SubTitle2    !RS: Deals with a second subtitle

    CALL SurfaceTypePropertiesIn    !RS: Reading in the surface properties
block

    CLOSE(Unit=2)

      END Subroutine  CalculateGEometry
!
!
!
    SUBROUTINE Calculate_SurfaceEquation()
!**********************************************************************
**
!
!  SUBROUTINE:    Calculate_Surface_Equation
!
!  PURPOSE:       Determines the coefficients 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 vertix
!  Y                 y  - coordinate of a vertix
!  Z                 z  - coordinate of a vertix


      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 coefficients 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))


      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) :: NV(SIndex), Vector(3) !Norm_V,
        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 cross 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
!     JDS 11-8-06 attempt to eliminate Norm_V linking problem
!       NV(SIndex) = Norm_V(Vector)
        NV(Sindex)=sqrt(DOT_PRODUCT(Vector,Vector))

!     Converts/Normalizes the normal vector to get the unit vector
        DO J = 1, 3
          NormalUV(SIndex,J) = Vector(J)/NV(SIndex)
      END DO

    END SUBROUTINE SurfaceNormal
!
!
!
      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 surface in the
enclosure
!   LT                 The three sides of a triangular surface in 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)))
        ENDIF

      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
   INTEGER :: I !Loop counter
   INTEGER :: J !Loop counter

   ALLOCATE(NAEnergy(NSurf,NSurf),RAD_D_F(NSurf,NSurf),RAD_D_S(NSurf,NSurf),
RAD_D_R(Nsurf,NSurf),RAD_D_WR(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)

ALLOCATE(TSpecA(NSurf),TSpecR(NSurf),TSpecRR(NSurf),NAEnergyS(NSurf,NSurf),NA
EnergyR(NSurf,NSurf),NAEnergyWR(NSurf,NSurf))    !RS !JH

   !Setting Specular Counter arrays to 0
    Do i = 1,NSurf  !JH
       TSpecA(i)=0
       TSpecR(i)=0
       TSpecRR(i)=0
        do j = 1,NSurf
           NAEnergyS(i,j) = 0
           NAEnergyR(i,j)=0
           NAEnergyWR(i,j)=0
        end do
    end do

   END SUBROUTINE AllocateArrays


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

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

!  Initialize absorbed 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
```

```
    SUBROUTINE SurfaceTypePropertiesIn()
!**********************************************************************
**
!
!  PURPOSE:       Reads in the surface type and properties
!
!**********************************************************************
**

   INTEGER :: I !RS: Loop Counter

   DO I = 1, NSurf
    READ(2,*)SNumber(I),SurfaceType(I)  !RS: Reading in the surface numbers
and types
   END DO

   I=1  !RS: Resetting I

   REWIND(2)    !RS: Going back to the beginning of the input file

   DO I=1,(NSurf+NVertex+2) !Surfaces, Vertices, and two subtitles
       READ(2,*)    !RS: Reading through the file until we get to the surface
properties block
   END DO

   I=1  !RS: Resetting I

   DO I=1,NSurf
    SELECTCASE(SurfaceType(I))  !RS: Dealing with the different surface cases
        CASE("SDE")
            READ(2,*)SNumber(I),SurfaceType(I), DirectionX(I), DirectionY(I),
DirectionZ(I) !RS: Reading in the direction vector
        CASE("SDR")
            READ(2,*)SNumber(I),SurfaceType(I), SpecReflec(I), DiffReflec(I)
!RS: Reading in specular and diffuse reflection
            SpecReflec(I)=1-SpecReflec(I)   !RS: Changing it to absorptance
for the absorption or reflection calculations
        CASE("DRO")
            READ(2,*)SNumber(I),SurfaceType(I) !RS: Nothing more to read in
here.
        CASE("SRO")
            READ(2,*) SNumber(I),SurfaceType(I),SpecReflec(I), DiffReflec(I)
!RS: Reading in specular and diffuse reflection
            SpecReflec(I)=1-SpecReflec(I)   !RS: Changing it to absorptance
for the absorption or reflection calculations
        CASE DEFAULT
            READ(2,*)SNumber(I),SurfaceType(I) !RS: Nothing more to read in
here either
        END SELECT
   END DO

   END SUBROUTINE

  End MODULE EnclosureGeometry

  MODULE EnergyAbsorbed_Reflected
```

```
  USE Global
  USE EnclosureGeometry
  USE EnergyBundleLocation


  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 generated is used to verify whether the
!                     intercepted energy is absorbed or reflected by comparing
!                     it with surface absorptance

!    JDS 11-10-2006 added all the " if (WriteLogFile) then" blocks to control
whether
!    or not a log file is written.  Also changed format statements to remove
commas
!    so that RTVT could actually read the file.
!
    R_absorbed = Rand(6)
!
IF (SpIndex .eq. 1) then    !Specular energy
     IF(R_absorbed < SpecReflec(SInter))Then
        NAEnergyS(SIndexR,SInter) = NAEnergyS(SIndexR,SInter) + 1   !RS:
Total number of energy bundles absorbed
        TSpecA(SIndexR) = TSpecA(SIndexR) + 1    !RS: Total Number of energy
bundles absorbed by surface

!   count the number of energy bundles absorbed and emitted
        IF (NCount .NE. 1) THEN !RS: Non-reflected rays
           NAEnergyWR(SIndexR,SInter)=NAEnergyWR(SindexR,Sinter)+1  !RS:
Total Number of energy bundles absorbed without reflection

             if (WriteLogFile) then
            Write(4,101,
ADVANCE='YES')'P',SIndex,XLS(SIndex),YLS(SIndex),ZLS(SIndex),SInter,XP(SIndex
,SInter),YP(SIndex,SInter),ZP(SIndex,SInter)
          end if
```

```
101      Format(A1,2(' ',I2,3(' ',f6.3)),' ' '0')

        ELSE    !RS: Rays reflected or rereflected
         if (WriteLogFile) then

Write(4,111,ADVANCE='YES')SInter,XP(SIndex,SInter),YP(SIndex,SInter),ZP(SInde
x,SInter)
         end if

111      Format(1(' ',I2,3(' ',f6.3)),' ' '0')

         NCount=0   !RS: Debugging: Resetting the counter

         TSpecR(SIndexR)=TSpecR(SIndexR)+1  !RS: Total number of reflected or
reflected rays absorbed by surface

         NAEnergyR(SIndexR,SInter)=NAEnergyR(SIndexR,SInter)+1  !RS: Total
number of reflected or rereflected rays absorbed

        END IF

         NTrials = NTrials + 1  !RS: Overall number of rays absorbed from the
emitting surface

        IF (SIndex ==1 .and. SInter == 2)Then
           count = count + 1

        ENDIf

          SIndex = SIndexR
          REF_IND = 0
          Reflected = .False.   !RS: Setting the reflection flag to false
since the array has now been absorbed
        TCountSpecR = 0 !RS:Resetting Reflection Flag

     ELSE
        IF(SIndex == SIndexR .and. REF_IND == 0)Then    !RS: Reflected Rays

         NCount=1   !RS: Marking a reflection

         TCountSpecR=1  !RS: Setting a flag

        if (WriteLogFile) then
           Write(4,112,
ADVANCE='NO')'P',SIndex,XLS(SIndex),YLS(SIndex),ZLS(SIndex),SInter,XP(SIndex,
SInter),YP(SIndex,SInter),ZP(SIndex,SInter)
        end if

112     Format(A1,2(' ',I2,3(' ',f6.3)))

        ELSEIF(REF_IND == 1)Then    !RS: Rereflected rays

         TCountSpecR=2  !RS: Debugging: Setting a flag

           if (WriteLogFile) then
            Write(4,102,ADVANCE='NO')SInter,
XP(SIndex,SInter),YP(SIndex,SInter),ZP(SIndex,SInter)
```

```
           end if

102     format(' ',1(I2,3(' ',f6.3)))

        END IF

        OldSurface=SIndex  !RS: Keeping track of the emitting surface for
each bounce
        SIndex=SInter
             Reflected= .True.  !RS: Setting a flag for reflection
         REF_IND=1  !RS: A flag to say the ray has already been reflected at
least once

     ENDIF
Else    !Diffuse Energy
    IF(R_absorbed < Emit(SInter))Then
        NAEnergy(SIndexR,SInter) = NAEnergy(SIndexR,SInter) + 1
          TCOUNTA(SIndexR) = TCOUNTA(SIndexR) + 1

!   count the number of energy bundles absorbed and emitted

        IF (NCountd .NE. 1) THEN    !RS: Marking as a non-reflected ray
              if (WriteLogFile) then
             Write(4,101,
ADVANCE='YES')'P',SIndex,XLS(SIndex),YLS(SIndex),ZLS(SIndex),SInter,XP(SIndex
,SInter),YP(SIndex,SInter),ZP(SIndex,SInter)
          end if
        ELSE
              if (WriteLogFile) then

Write(4,111,ADVANCE='YES')SInter,XP(SIndex,SInter),YP(SIndex,SInter),ZP(SInde
x,SInter)
              end if

             NCountd=0  !RS: Resetting the reflection counter

        END IF

         NTrialsd = NTrialsd + 1

        IF (SIndex ==1 .and. SInter == 2)Then
           count = count + 1
        ENDIf

          SIndex = SIndexR
          REF_IND = 0
          Reflected = .False.  !RS: Setting the reflection flag to false
since the array has now been absorbed

     ELSE
        IF(SIndex == SIndexR .and. REF_IND == 0)Then
         TCOUNTR(SIndexR) = TCOUNTR(SIndexR) + 1

         NCountd = 1    !RS: Counting as a reflected surface

            if (WriteLogFile) then
```

```
                Write(4,112,
ADVANCE='NO')'P',SIndex,XLS(SIndex),YLS(SIndex),ZLS(SIndex),SInter,XP(SIndex,
SInter),YP(SIndex,SInter),ZP(SIndex,SInter)
             end if
         ELSEIF(REF_IND == 1)Then
            TCOUNTRR(SIndexR)= TCOUNTRR(SIndexR) + 1

            if (WriteLogFile) then
           Write(4,102,ADVANCE='NO')SInter,
XP(SIndex,SInter),YP(SIndex,SInter),ZP(SIndex,SInter)
                  end if

         END IF

         SIndex=SInter
              Reflected= .True.
          REF_IND=1

     ENDIF
Endif

  END SUBROUTINE AbsorptionReflection
  END MODULE EnergyAbsorbed_Reflected

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, 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

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 is
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 a reflected energy bundle, no need to calculate the emission
point
  IF(Reflected)Then

    XLS(SIndex) = Xo(SInter)
```

```
    YLS(SIndex) = Yo(SInter)
      ZLS(SIndex) = Zo(SInter)

  ELSE
    DO J = 1, 3 !Calculates emission point
           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))

    !Generating random numbers
    !The following equations are from Dr. Spitler's notes, Monte Carlo Ray
Tracing in Radiation Heat Transfer
    Randu=1-SQRT(1-Rand(1))
    Randv=(1-Randu)*Rand(2)

    XLS(SIndex)=X(1)+Randu*Vedge1(1)+Randv*Vedge2(1)
    YLS(SIndex)=Y(1)+Randu*Vedge1(2)+Randv*Vedge2(2)
    ZLS(SIndex)=Z(1)+Randu*Vedge1(3)+Randv*Vedge2(3)

  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)
    REAL(prec2), ALLOCATABLE, DIMENSION(:) :: SurfaceE
      REAL(Prec2), DIMENSION(4) :: X, Y, Z
    REAL(Prec2), DIMENSION(:,:) :: Vedge1(3), Vedge2(3), Vedge3(3) ! Dividing
the rectangles into triangles
    REAL(Prec2) :: Randu, Randv
!
! 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)

  ELSE
    DO J = 1, 4 !Otherwise, determine emission location
           VS(J) = SVertex(SIndex,J)
            X(J) = XS(VS(J))
            Y(J) = YS(VS(J))
              Z(J) = ZS(VS(J))
    END DO

    Vedge1(1) = (X(2) - X(1))
      Vedge1(2) = (Y(2) - Y(1))
      Vedge1(3) = (Z(2) - Z(1))

      Vedge2(1) = (X(4) - X(1))
      Vedge2(2) = (Y(4) - Y(1))
      Vedge2(3) = (Z(4) - Z(1))

    Vedge3(1) = (X(3) - X(1))
      Vedge3(2) = (Y(3) - Y(1))
      Vedge3(3) = (Z(3) - Z(1))

    !The following equations are from Dr. Spitler's notes, Monte Carlo Ray
Tracing in Radiation Heat Transfer
    Randu=1-SQRT(1-Rand(1))
    Randv=(1-Randu)*Rand(2)

    IF(Rand(7) .GT. 0.5) THEN
         XLS(SIndex)=X(1)+Randu*Vedge1(1)+Randv*Vedge3(1)
         YLS(SIndex)=Y(1)+Randu*Vedge1(2)+Randv*Vedge3(2)
         ZLS(SIndex)=Z(1)+Randu*Vedge1(3)+Randv*Vedge3(3)
    ELSE
        XLS(SIndex)=X(1)+Randu*Vedge2(1)+Randv*Vedge3(1)
        YLS(SIndex)=Y(1)+Randu*Vedge2(2)+Randv*Vedge3(2)
        ZLS(SIndex)=Z(1)+Randu*Vedge2(3)+Randv*Vedge3(3)
    END IF

    IF (XLS(SIndex) .LT. 0 .OR. YLS(SIndex) .LT. 0 .OR. ZLS(SIndex) .LT. 0)
THEN
        WRITE(*,*) 'Error! Check the vertices on your input file.'
    END IF

   ENDIF
  END SUBROUTINE RectangularSurface
```

```
  SUBROUTINE InitializeSeed()
!**********************************************************************
**
!
!   PURPOSE:      Initialization of seed for the random Number generator
!
!
!**********************************************************************
**
      IMPLICIT NONE
    INTEGER :: K
      INTEGER, DIMENSION(:) ::  SEEDARRAY(7), OLDSEED(7)
!    Sets K = N
      K = 7
    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 -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
!    JDS 11-8-06 attempt to eliminate linking problem with Norm_V
!      NV = Norm_V(TUV1)
     NV=sqrt(DOT_PRODUCT(TUV1,TUV1))
     DO J = 1, 3
        TUV1(J) = TUV1(J)/NV
      Tan_V1(SIndex,J) = TUV1(J)
     END DO
!
!  The second tangent vector is given by the cross product of the surface
normal
!  vector and the first tangent vector
     Call CrossProduct(V, TUV1,TUV2)
     DO J = 1, 3
      Tan_V2(SIndex,J) = TUV2(J)
     END DO
!
  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, J
      REAL(Prec2) :: Theta, Phi, Pi, DotTheta, MagVec !Theta1, Theta2,
```

```
    INTEGER, DIMENSION (:) :: VS(4)
    REAL(Prec2), DIMENSION(:,:) :: InVecDirec(3), SurfNorm(3)   !RS: Incoming
Vector Direction and Surface Normal
    REAL(Prec2), DIMENSION(4) :: X, Y, Z
!
!  Calculate emitted energy bundle direction angles
    Pi = 4.*Atan(1.)
      Theta = asin(sqrt(Rand(4)))
    Phi   = 2.*Pi*Rand(5)
ALLOCATE (EmittedUV(NSurf,3), STAT = IOS )
! Calculate the unit vector in the direction of the emitted energy bundle
If (Spindex .eq. 1) then    !RS: For Specular Rays
    IF (Reflected) Then !RS: If the rays are being reflected off of another
surface

            SurfNorm(1)=NormalUV(SIndex,1)  !Surface normal unit vector
            SurfNorm(2)=NormalUV(SIndex,2)
            SurfNorm(3)=NormalUV(SIndex,3)

            !Taking the incoming direction from the specified emission
direction
            IF (TCountSpecR .EQ. 1) THEN  !If the ray is being reflected for
the first time
                MagVec=SQRT(DirectionX(OldSurface)**2 +
DirectionY(OldSurface)**2 + DirectionZ(OldSurface)**2)
                InVecDirec(1)=-DirectionX(OldSurface)/MagVec  !I is negative
since the ray is incoming
                InVecDirec(2)=-DirectionY(OldSurface)/MagVec
                InVecDirec(3)=-DirectionZ(OldSurface)/MagVec
            ELSE    !If the ray is being rereflected
                InVecDirec(1)=-EmittedUV(OldSurface,1)
                InVecDirec(2)=-EmittedUV(OldSurface,2)
                InVecDirec(3)=-EmittedUV(OldSurface,3)
            END IF

            DotTheta=DOT_PRODUCT(InVecDirec,SurfNorm)   !Dot product of the
incoming ray and surface normal

            !r=2(I dot n)n -I   !Page 5, Nancy Pollard, 2004,
http://graphics.cs.cmu.edu/nsp/course/15-462/Spring04/slides/13-ray.pdf

            EmittedUV(SIndex,1)=2*DotTheta*SurfNorm(1)-InVecDirec(1)
            EmittedUV(SIndex,2)=2*DotTheta*SurfNorm(2)-InVecDirec(2)
            EmittedUV(SIndex,3)=2*DotTheta*SurfNorm(3)-InVecDirec(3)

    ELSE    !If Not Reflected
        EmittedUV(SIndex,1) = DirectionX(SIndex)
        EMittedUV(SIndex,2) = DirectionY(SIndex)
        EmittedUV(Sindex,3) = DirectionZ(SIndex)
    END IF

    CALL CheckDirection

Else
      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)
End If

  END SUBROUTINE  DirectionEmittedEnergy

 SUBROUTINE CheckDirection

  !RS:Debugging: Trying to set direction=0 if it doesn't exist

  IF (EMittedUV(SIndex,1) .LT. (-10E10) .OR. EmittedUV(SIndex,1) .GT.
(10E10)) THEN
      EMittedUV(SIndex,1)=0
  END IF

  IF (EMittedUV(SIndex,2) .LT. (-10E10) .OR. EmittedUV(SIndex,2) .GT.
(10E10)) THEN
      EMittedUV(SIndex,2)=0
  END IF

  IF (EMittedUV(SIndex,3) .LT. (-10E10) .OR. EmittedUV(SIndex,3) .GT.
(10E10)) THEN
      EMittedUV(SIndex,3)=0
  END IF

  END SUBROUTINE

End Module EnergyBundleLocation

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


  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 multiplier 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
with 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 their 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
!  and 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
     IF (UNV_DOT_EUV .EQ. 0) THEN
         SI(Index)=0.0  !RS: In the case of division by 0
     END IF
    ELSE
     SI(Index) = 0.0
    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:       Selects 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 distance
!     SIMAX       Maximum real number
!   Assign the maximum Real number to SIMAX

    SIMAX  = 10000000000000000.0

   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) = 1  !0 means no intersection, 1 means there
is Inter.
       Else
        Intersection(SIndex,Index) = 0
       Endif
      Else
        Intersection(SIndex,Index) = 0
      INTersects(SIndex)=.FALSE.    !RS: Setting the intersection flag to
false for cases when it's the emission surface
    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)
        EndIF

!  Eliminate intersection point on the back side of emission
        IF(SI(Scount) > 0.0 .and. Intersection(SIndex,Scount) == 1)THen
          SIINTER(Scount) = SI(Scount)
      Else
           SIINTER(Scount) = SIMAX
      ENDIF
      END DO

!  Assign the minimum distance from intersection point
    SIMIN = MINVAL(SIINTER)
```

```
!  Determine intersection by selecting the closest point
    DO I =1, Nsurf
     IF (INTersects(I))Then
        IF(SIINTER(I) == SIMIN) Then
           SInter = I
        END IF
       END IF
    END DO

  END SUBROUTINE SingleOutIntersection
!
!
  SUBROUTINE IntersectionRectangle(Index)
!**************************************************************************
**
!
!  SUBROUTINE:    IntersectionRectangle
!
!  PURPOSE: Finds intersection point (if any) for rectangular surface
!               JDS: Should also work for any trapezoidal or convex 4-sided
!                    polygon
!
!
!**************************************************************************
**
!
!       Modifications:
!       24 November 2012 - JDS: clean up internal documentation whilst trying
to
!                                figure out what is going on!
!
!       Input variables:
!       Index   =  index of surface that is being tested for possible
intersection
!       Note: Current ray information is stored in Global variables:
!             Sindex: emitting (or reflecting) surface index
!             Intersection(i,j)=1 if the ray emitted from the ith surface
intersects the plane of
!             the jth surface; else =0
!             (JDS: if this only applies to the current ray, why is it stored
in an array?
!               We shouldn't even call this subroutine if it doesn't
intersect.)
!              XP,YP,ZP hold x,y,z coordinates of intersection on the plane,
previously determined
!
!     UNV   = Unit normal vector of the rectangular surface
!       V_Int     = Vector from one vertex to the intersection (on plane of
surface) point
!     V_edge      = Vector along the edges of the surfaces defined in
consistent
!              direction
!     VcpS  = Cross product vector between the edges and intersection vector
!       VcpN    = Dot product of VcpS and the surface unit normal 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), Vedge1(3), Vedge2(3),
Vedge3(3), Vedge4(3)
   REAl(Prec2) SIMIN
!  checks whether the point of intersection of the surface's plane is within
the
!  enclosure
!  Assign surface its corresponding vertices
! (JDS: Shouldn't this be done once globally?)
     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 the surface edges using the vertices of the
surfaces
! (JDS: Shouldn't this be done once globally?)

    IF(Index .ne. SIndex) 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))
          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
      ENDIF

!  Eliminate intersection point outside the surface domain
      IF(VcpN(Index,1)> 0.0 .and. VcpN(Index,4) > 0.0 .and. VcpN(Index,2)>
0.0 .and. VcpN(Index,3)> 0.0) THEN
        SInter = Index
            INTersects(Index) = .True.
```

```
!  Save the intersection point coordinates

       Xo(SInter) = XP(SIndex,Index)
         Yo(SInter) = YP(SIndex,Index)
         Zo(SInter) = ZP(SIndex,Index)

! JDS: One possible problem - if intersection is on vertex or edge, it will
be "false"

    ELSE
            INTersects(Index) = .false.
            Intersection(SIndex,Index) = 0
      ENDIF
  END SUBROUTINE IntersectionRectangle
!
!
  SUBROUTINE IntersectionTriangle(Index)
!**********************************************************************
**
!
!  SUBROUTINE:    IntersectionTriangle
!
!  PURPOSE:       Selects the exact intersection points for triangular
surfaces
!
!
!**********************************************************************
**
!
!     UNV         = Unit normal vector of the surfaces
!   V_Int   = Vector from the vertices 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
       END DO
    ELSE
      ENDIF

!  Eliminate intersection point outside the surface domain
    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
         INTersects(Index) = .True.
         Intersection(SIndex,Index) = 1

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

    ELSE
        INTersects(Index) = .false.
        Intersection(SIndex,Index) = 0
      ENDIF
  END SUBROUTINE IntersectionTriangle

  END MODULE IntersectionEnergy_Surface

Module OutPut

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

```
!
  IMPLICIT NONE
  CONTAINS
!
  SUBROUTINE Print_ViewFactor_HeatFlux()
!**********************************************************************
**
!
! PURPOSE:              Prints View Factors, Radiation Heat Fluxe and Heat
Transfer
!                             Rate at Each Surface
!
!
!**********************************************************************
**
    IMPLICIT NONE
    INTEGER       :: I,J,k,Index


!  Write the Title of the Program and Output data
    Write(3,101)'Monte Carlo Method','PURPOSE:','Calculates The View &
                Factors Using Monte Carlo Method','and','The Net Radiation &
                      Heat Flux at Each Surface'
101  Format(//,15x,A30,///,14x,A25,//,14x,A52,/,36x,A3,/,11x,A50,//)


    DO k =1, NSurf
        Write(3,1001)NAEnergy(k,:),TCOUNTA(k)
      Write(6,1001)NAEnergyS(k,:),TSpecA(k) !JH !Writing the number of total
specular rays absorbed at each surface
      Write(9,1001)NAEnergyR(k,:),TSpecR(k) !Writing the number of reflected
specular rays absorbed at each surface
      Write(10,1001)NAEnergyWR(k,:),(TSpecA(k)-TSpecR(k))   !Writing the
number of specular rays absorbed on first contact at each surface
      END DO

1001 Format(2x,100(x,I8),I10)

     Write(3,1002)
     WRITE(6,1002)
     WRITE(9,1002)
     WRITE(10,1002)
1002 Format(//)
       DO Index = 1, NSurf
          Write(3,102)(RAD_D_F(Index,J), J=1, NSurfcmb)   !Diffuse
distribution factors
        WRITE(6,102)(RAD_D_S(Index,J), J=1, NSurfcmb)   !Total specular
distribution factors
        WRITE(9,102)(RAD_D_R(Index,J), J=1, NSurfcmb)   !Reflected specular
distribution factors
        WRITE(10,102)(RAD_D_WR(Index,J), J=1, NSurfcmb) !Absorbed at first
intersection specular distribution factors
     END DO

     !Writing the rest of the outputs for MCOutput.txt
102  Format(4x,100(2x,f8.6))
```

```
     Write(3,103)'Index','SURF_NAME','Temperature','Emissivity','Heat Flux',
&
                    'Heat Transfer Rate'
103 Format(///,8x,A5,2x,A10,6x,A12,2x,A12,4x,A12,8x,A19)
       DO Index = 1, NSurf

Write(3,104)Index,SURF_NAME(Index),TS(Index),EMIT(Index),QFLUX(Index) &
                        ,Q(Index)
104  Format(7x,I3,8x,A12,4x,F7.2,8x,F4.2,8x,ES12.3,10x,ES12.3)
       END DO

       Write(*,107)'Elapsed Time:',TIME2-TIME1,'s'
       Write(3,107)'Elapsed Time:',TIME2-TIME1,'s'
107  Format(//,8x,A14,1x,F14.2,x,A1)

  END SUBROUTINE Print_ViewFactor_HeatFlux
 END Module OutPut
```

Appendix B: Trapezoidal Case Input File

```
V 1   0   0   0
V 2   1   0   0
V 3   1   1   0
V 4   0   1   0
V 5 0.25 0.25  1
V 6 0.75 0.25  1
V 7 0.75 0.75  1
V 8 0.25 0.75  1
!  #       v1  v2    v3  v4    base  cmb  emit name
S  1     1   2   3   4   0     0   0.9  Base
S  2     6   2   1   5   0     0   0.9  Front Face
S  3     7   3   2   6   0     0   0.9  Right Face
S  4     8   4   3   7   0     0   0.9  Back Face
S  5     5   1   4   8   0     0   0.9  Left Face
S  6     8   7   6   5   0     0   0.9  Top
!# Type
 1  DIF
 2  DIF
 3  DIF
 4  DIF
 5  DIF
 6  DIF
```

Appendix C: Equinox House Input File

```
V 1 0 0 0
v 2 5 0 0
v 3 5 2.5 0
v 4 5 5 0
v 5 5 5 5
v 6 5 2.5 6
v 7 5 0 5
v 8 0 0 5
v 9 0 5 0
v 10 0 2.5 6
v 11 5 2.5 7
v 12 0 2.5 7
v 13 0 5 5
v 14 3.25 2.5 6.25
v 15 3.25 2.5 6.75
v 16 4.75 2.5 6.75
v 17 4.75 2.5 6.25
v 18 0.25 2.5 6.25
v 19 0.25 2.5 6.75
v 20 1.75 2.5 6.75
v 21 1.75 2.5 6.25
v 22 5 2.5 6.25
v 23 5 2.5 6.75
v 24 0 2.5 6.75
v 25 0 2.5 6.25
v 26 0 2.5 0
!  #   v1  v2  v3  v4 base cmb  emit  name      surface data
S 1    1   8   7   2 0    0    0.9    face
s 2   2    7   6   3 0    0    0.9    SE side
s 3   3    11  5   4 0    0    0.9    NE side
s 4   9    4   5   13 0   0    0.9    back side
s 5   9    13  12  26 0   0    0.9    NWside
s 6   26   10  8   1 0    0    0.9    SW side
s 7   1    2   4   9 0    0    0.9    base
s 8   6    7   8   10 0   0    0.9    S roof
s 9   5    11  12  13  0   0    0.9    N roof
s 10  11  23  24  12  0   0    0.9    Top perpend ceiling
s 11  6   10  25  22  0   0    0.9    bottomceliinh
s 12  18  25  24  19  0   0    0.9    W perpen
s 13  21  20  15  14  0   0    0.9    centre perp
s 14  22  17  16  23  0   0    0.9    E perpe
s 15  18  19  20  21  0   0    0.9    W window
s 16  14  15  16  17  0   0    0.9    E window
!# Type
1   DIF
2   DIF
```

3 DIF
4 DIF
5 DIF
6 DIF
7 DIF
8 DIF
9 DIF
10 DIF
11 DIF
12 DIF
13 DIF
14 DIF
15 SDE 0.2 0.2 -.3
16 SDE 0.2 0.2 -.3
End of Data