2017 MCRT Improvements

Matt S. Mitchell

December 14, 2017

Contents

1 Introduction

At the beginning of this project, I was provided with a list of deficiencies and bugs within the MCRT program. Some of these issues were minor and simple to fix, while others were more complex. This project is a compilation of these improvements which were made to the MCRT program during the Fall 2017 semester for MAE 5823. The subsequent sections describe these improvements.

2 Compiler

It was mentioned that the 2015 MCRT which was distributed to the class may have been compiled in 'debug' mode. To test this, I took the MCRT 2015 code and compiled it in 'debug' and 'release' modes using an Intel FORTRAN compiler. I then compared the run time for an example input file for both versions. In release mode, I also compared run times for several different levels of optimization. The gfortran compiler was also tested using a cross compiled version (Windows exe compiled on a Linux machine) and a pure Linux version (Linux exe compiled on Linux). The gfortran versions were compiled using O2 optimization in release mode. The results of this comparison are shown in Figure 1.

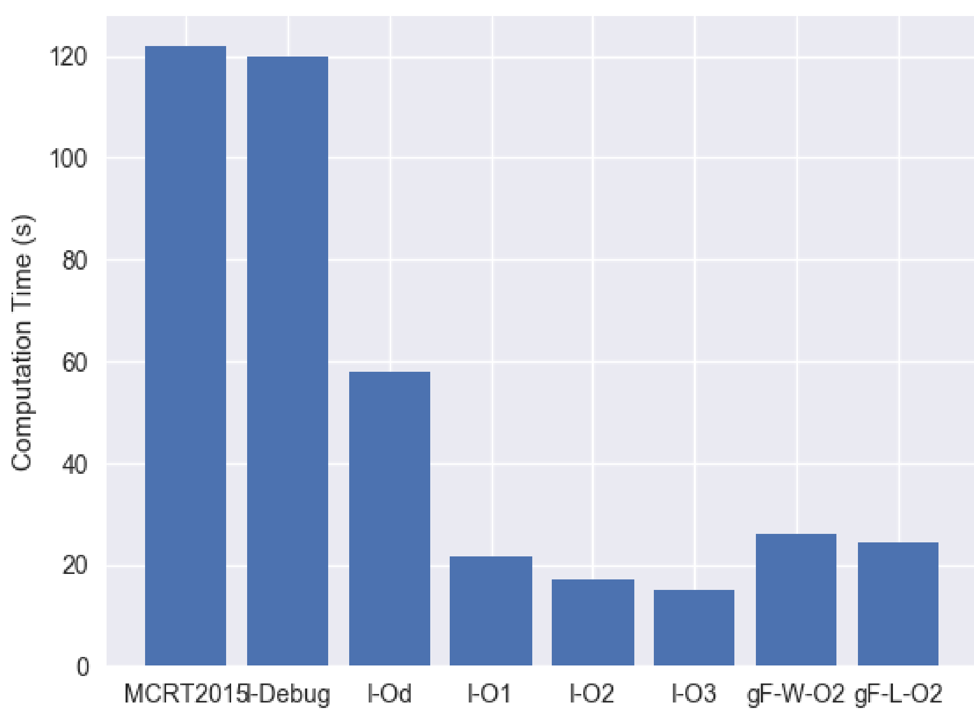

Figure 1: Compiler comparisons

No change in results was observed between the cases. All tests were run using the Sauna test file used repeatedly during this semester, with 1×10^6 rays diffusely emitted per surface.

3 Random Seed

I noticed that if the program was run repeatedly, the results would be identical. Due to the fact that Monte-Carlo methods should be stochastic, meaning that the inputs are quasi-random, we expect the results to change between runs. However, the with enough data points the results should approach the same value. It was determined that the seed for the random number generator was fixed at a constant value. This was changed to use a random seed which is generated from the current computer clock time. After applying this, MCRT outputs vary between runs, but approach the same values given enough data points, which is how the program is expected to behave.

4 Input Structure Modifications

Input processing for the input.vs3 file required a rigid input structure. First, a list of verticies; then, a single comment line; another list of surfaces; then, another single comment line; a list of surface types; then finally an end of file indicator, "End of Data."

This input structure was not flexible and did not allow for extra user comments in the input file. To address this, I overhauled the input processor for this file to allow any number of comment lines. The list of verticies must be written to the file before the list of surfaces, which in turn must be listed before the surface types. However, now the vertex and surface lists may be entered out of order. The surface types are also defaulted to the DIF type if the user omit them from the file. The end of file indicator is also no longer needed. Additional information regarding the input structure can be found in Mitchell (2017).

5 Status Messages

MCRT 2015 would give no messages output to the terminal indicating to the user its simulation status. I have added a number of status messages which will output to the terminal at one time intervals and after completing simulating each surface. An example of this is shown below in Figure 2.

6 Combined Surface Reporting

Some large surfaces need to be broken down into quadrilaterals or triangles in order to be simulated by MCRT. After the simulation is complete, though, MCRT like View3D (Walton, 2002) allows users to combine these subsurfaces into larger surfaces to determine the distribution factors of the actual large surface, not just the simplified subsurfaces. The code used to combine the distribution factors from the smaller subsurfaces into the original larger surfaces was producing incorrect results. These issues were corrected.

7 Generalized Surface Types

A previous team to work on MCRT (Holman et al., 2012) attempted to add a number of different surface types. Upon testing, however, it was determined that the surface types were not functioning as expected. The reporting schemes were also complicated, with specular and diffusely emitted rays being reported, supposedly, in different output files.

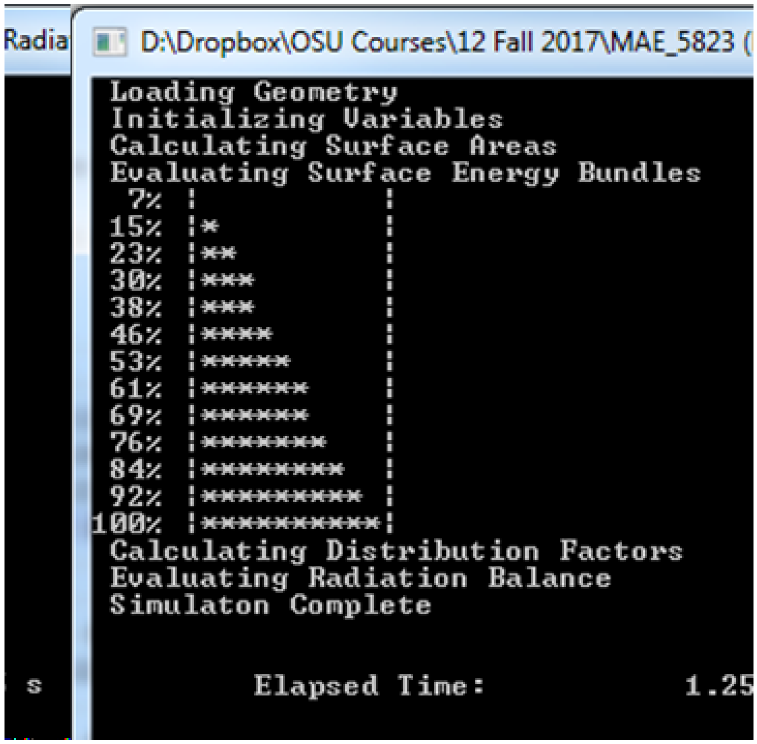

Figure 2: Status message example.

To correct this, I overhauled the surface types allowed in MCRT. The total number of surface types was reduced from six to three, and the remaining surface types were generalized to handle specular and diffuse emission and reflection per user specified parameters. Testing of these surface types is described in Section 8. Additional details regarding the surface type input values can be found in the MCRT documentation.

8 Testing

To verify the new surface types function as expected, a few test cases were run. These are outlined here. All cases were run with an L-shaped enclosure, which can be seen in Figure 3. RTVT (Crews, 2005) loads the enclosure wireframe with the positive z direction pointed downwards, Figures 4–8, which are images of the ray paths visualized using RTVT, will appear inverted due to this.

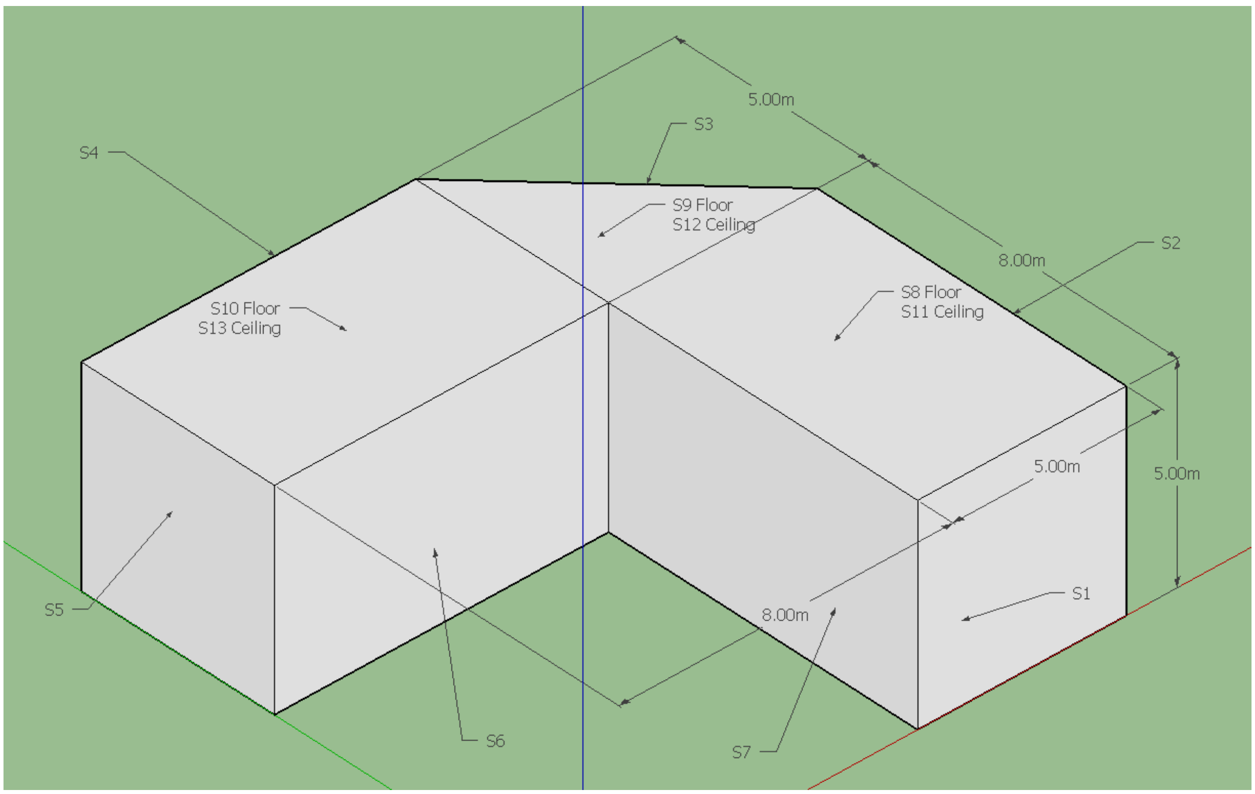

Figure 3: L-shape Enclosure

8.1 Case 1

- Surface 1: 100% diffuse emission
- Surface 3: Absorptivity set to 0.999

For this case, we expect all ray emitted from 1 to be emitted specularly directly to 3, then to be absorbed. From Figure 4, this appears to function as expected.

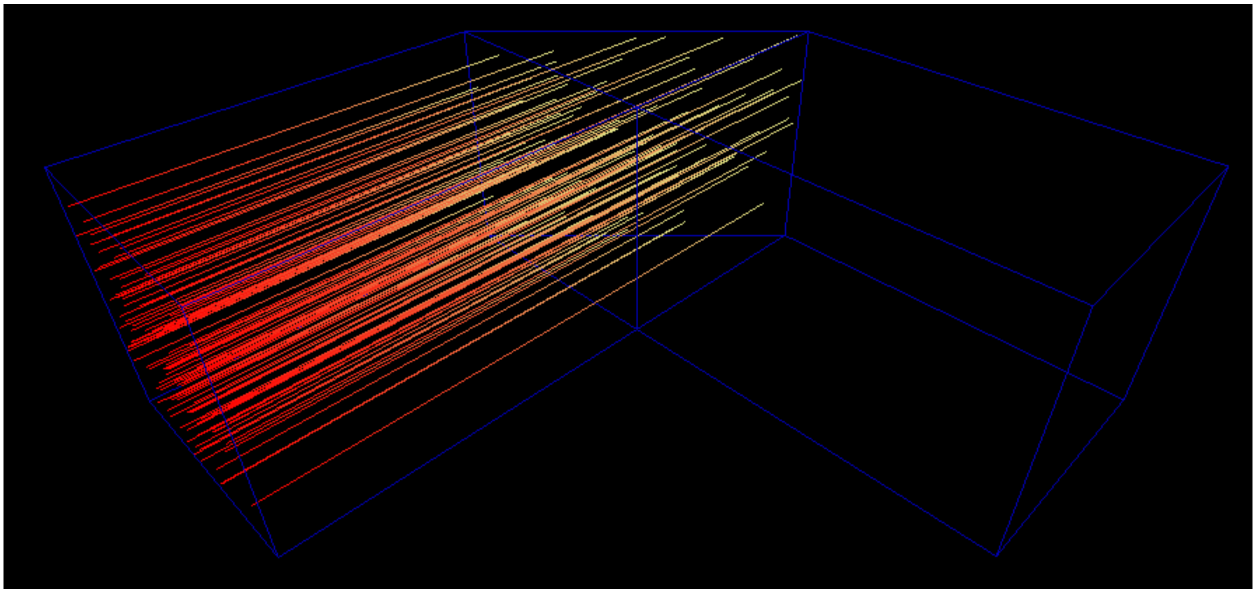

Figure 4: Case 1

8.2 Case 2

- Surface 1: 100% specular emission
- All other surfaces: Absorptivity set to 0.999

For this case, we expect all rays emitted from 1 to be diffuse and to be absorbed immediately by other surfaces. From Figure 5, this appears to function as expected.

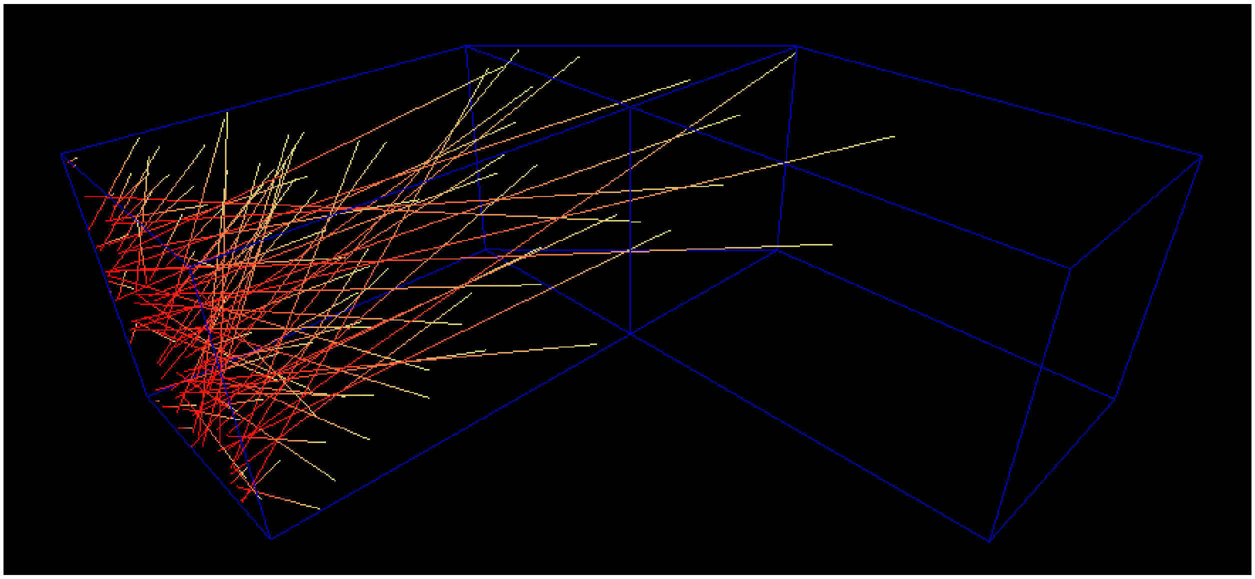

Figure 5: Case 2

8.3 Case 3

- Surface 1: 100% specularly emitted rays
- Surface 3: Absorptivity set to 0.1; Fraction of reflected rays set to be 100%
- Surface 5: Absorptivity set to 0.999

For this case, we expect all rays to be emitted from 1 specularly directly to 3. Most (what are not absorbed) will be reflected specularly to 5 and be absorbed. From Figure 6, this appears to function as expected.

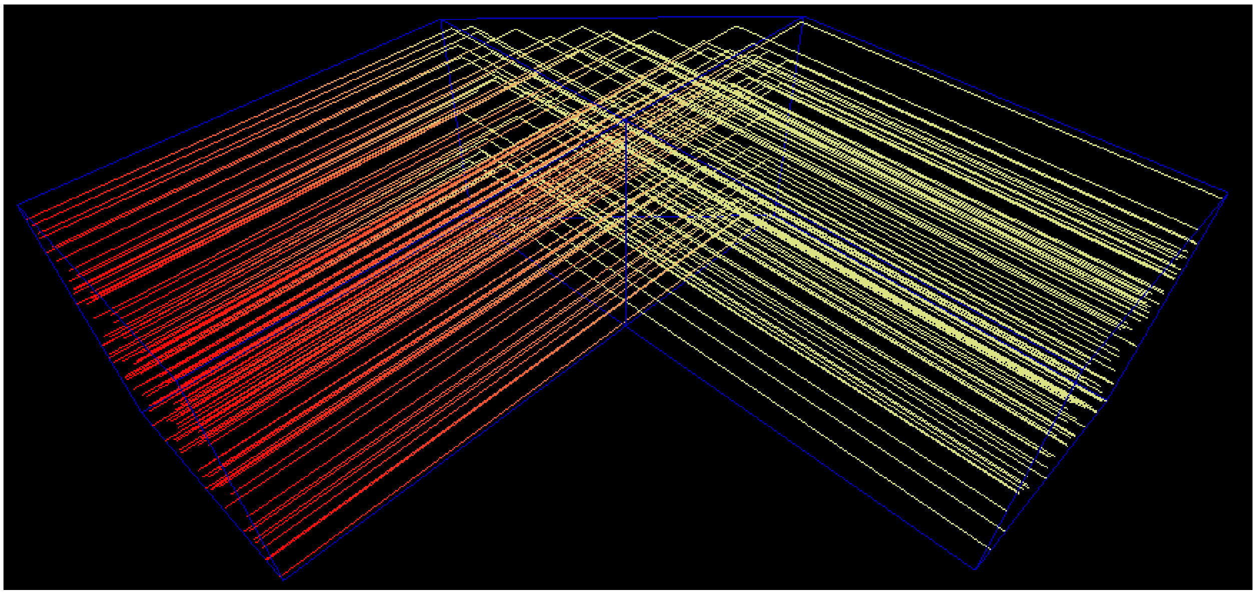

Figure 6: Case 3

8.4 Case 4

- Surface 1: 100% specularly emitted rays
- Surface 3: Absorptivity set 0.1; Fraction of reflected rays set to be 0%
- All other surfaces: Absorptivity set to 0.999

For this case, we expect all rays to be emitted from 1 specularly directly to 3. Most (what are not absorbed) will be reflected diffusely and be absorbed. From what we see in Figure 7, this appears to function as expected.

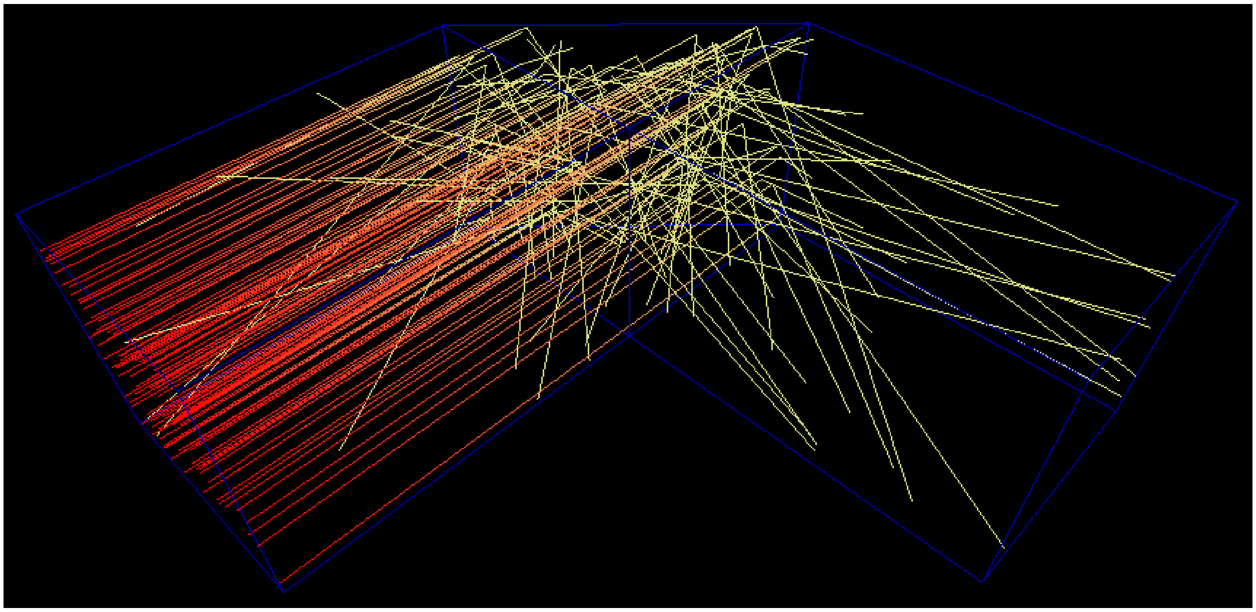

Figure 7: Case 4

8.5 Case 5

- Surface 3 set to "SDRO"

For this case, we expect surface 3 to emit no rays. Viewing the RTVT output result in Figure 8, this appears to function as expected.

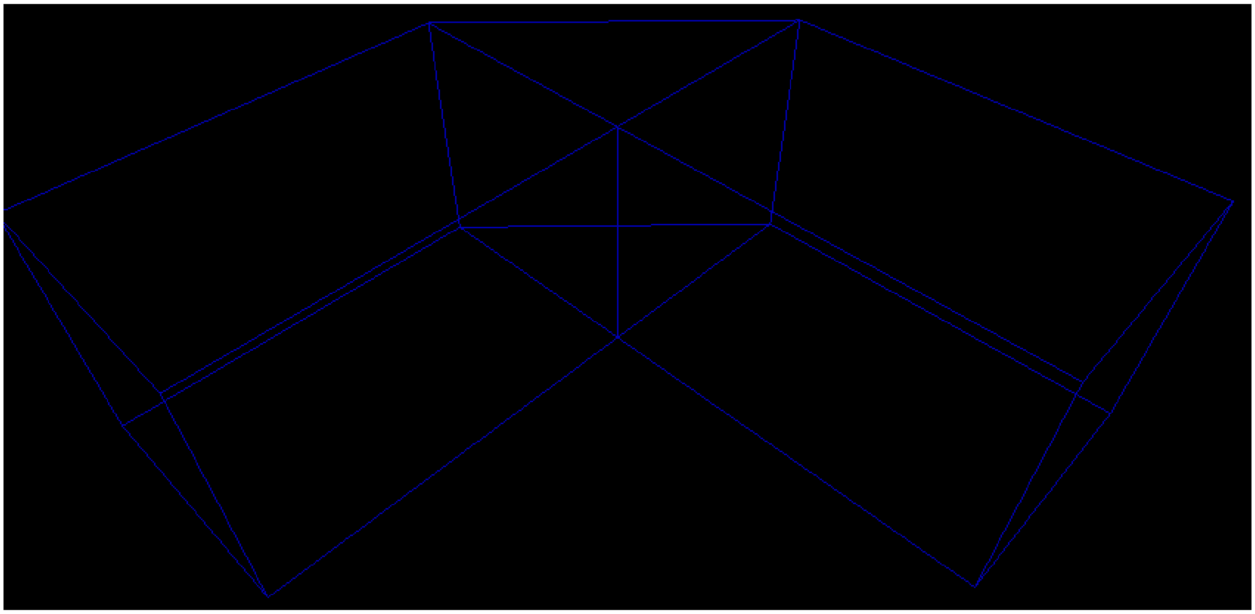

Figure 8: Case 5

9 Continuous Integration

MCRT was tested by Nigusse (2004) initially, then later by Holman et al. (2012). One problem, however, with this intermittent testing approach is that is isn't standardized or continued after the project is passed on to the next person. As a first step to remedy this issue, I have hosted the code on GitHub (`https://github.com/BETSRG/MCRT/tree/master`) and incorporated a continuous integration (CI) testing service, Travis-CI (`https://travis-ci.org/BETSRG/MCRT`) to build MCRT each time a commit is made to the GitHub repository. Travis builds MCRT using gfortran. If the build fails, CI emails the users that the build is not functional. Additional testing should be incorporated into this, however.

10 Documentation

Finally, I noticed that there was essentially no information describing how MCRT is "supposed" to work. The student reports are informative, but are fixed at one point in time and may not be relevant to current code. To remedy this, I developed some documentation to describe input/output syntax and explain how the current program is intended to work. In the future, any changed to the program's functionality need to be incorporated into the documentation.

References

Crews, B. (2005). *RTVT: Ray Tracing Visualization Tool*. Oklahoma State Univsersity, Stillwater, OK.

Holman, J., Spitler, R., and Sikha, S. (2012). Monte Carlo ray tracing program improvements. Technical report, Oklahoma State University, Stillwater, OK. Semester project final report for MAE 5823.

Mitchell, M. S. (2017). *MCRT Program Documentation*. Oklahoma State University, Stillwater, OK.

Nigusse, B. A. (2004). Radiation heat transfer in enclosure using Monte-Carlo method. Technical report, Oklahoma State University, Stillwater, OK. Semester project final report for MAE 5823.

Walton, G. N. (2002). Calculation of obstructed view factors by adaptive integration. Technical Report NISTIR 6925, NIST, Gaithersburg, MD.

Appendix—Test Case Input Files

Listing 1: Case 1 Input File

```
V   1   8   0   0
V   2  13   0   0
V   3  13   8   0
V   4   8  13   0
V   5   0  13   0
V   6   0   8   0
V   7   8   8   0
V   8   8   0   5
V   9  13   0   5
V  10  13   8   5
V  11   8  13   5
V  12   0  13   5
V  13   0   8   5
V  14   8   8   5
!   #  v1  v2  v3  v4 base cmb  emit   name        surface data
S   1   8   9   2   1   0   0   0.90   South-Wall-Bottom-(SDE)
S   2   9  10   3   2   0   0   0.90   East-Wall
S   3  10  11   4   3   0   0   0.999  Angle-Wall
S   4  11  12   5   4   0   0   0.90   North-Wall
S   5  12  13   6   5   0   0   0.90   West-Wall-Top
S   6  13  14   7   6   0   0   0.90   South-Wall-Top
S   7  14   8   1   7   0   0   0.90   West-Wall-Bottom
S   8   1   2   3   7   0   0   0.90   South-Floor
S   9   7   3   4   0   0   8   0.90   Angle-Floor
S  10   4   5   6   7   0   8   0.90   North-Floor
S  11  14  10   9   8   0   0   0.90   South-Ceiling
S  12  14  11  10   0   0  11   0.90   Angle-Ceiling
S  13  13  12  11  14   0  11   0.90   North-Ceiling
T  1   SDE 0 1 0 1 0.5
T  3   SDRO 0
```

Listing 2: Case 2 Input File

```
V   1   8   0   0
V   2  13   0   0
V   3  13   8   0
V   4   8  13   0
V   5   0  13   0
V   6   0   8   0
V   7   8   8   0
V   8   8   0   5
V   9  13   0   5
V  10  13   8   5
V  11   8  13   5
V  12   0  13   5
V  13   0   8   5
V  14   8   8   5
!   #  v1  v2  v3  v4 base cmb  emit   name           surface data
S   1   8   9   2   1   0   0   0.999  South-Wall-Bottom-(SDE)
S   2   9  10   3   2   0   0   0.999  East-Wall
S   3  10  11   4   3   0   0   0.999  Angle-Wall
S   4  11  12   5   4   0   0   0.999  North-Wall
S   5  12  13   6   5   0   0   0.999  West-Wall-Top
S   6  13  14   7   6   0   0   0.999  South-Wall-Top
S   7  14   8   1   7   0   0   0.999  West-Wall-Bottom
S   8   1   2   3   7   0   0   0.999  South-Floor
S   9   7   3   4   0   0   8   0.999  Angle-Floor
S  10   4   5   6   7   0   8   0.999  North-Floor
S  11  14  10   9   8   0   0   0.999  South-Ceiling
S  12  14  11  10   0   0  11   0.999  Angle-Ceiling
S  13  13  12  11  14   0  11   0.999  North-Ceiling
T  1   SDE 0 1 0 0 0.5
T  3   SDRO 0
```

Listing 3: Case 3 Input File

```
V   1   8   0   0
V   2  13   0   0
V   3  13   8   0
V   4   8  13   0
V   5   0  13   0
V   6   0   8   0
V   7   8   8   0
V   8   8   0   5
V   9  13   0   5
V  10  13   8   5
V  11   8  13   5
V  12   0  13   5
V  13   0   8   5
V  14   8   8   5
!   #  v1  v2  v3  v4  base  cmb  emit   name        surface  data
S   1   8   9   2   1    0    0   0.90   South-Wall-Bottom-(SDE)
S   2   9  10   3   2    0    0   0.90   East-Wall
S   3  10  11   4   3    0    0   0.1    Angle-Wall
S   4  11  12   5   4    0    0   0.90   North-Wall
S   5  12  13   6   5    0    0   0.999  West-Wall-Top
S   6  13  14   7   6    0    0   0.90   South-Wall-Top
S   7  14   8   1   7    0    0   0.90   West-Wall-Bottom
S   8   1   2   3   7    0    0   0.90   South-Floor
S   9   7   3   4   0    0    8   0.90   Angle-Floor
S  10   4   5   6   7    0    8   0.90   North-Floor
S  11  14  10   9   8    0    0   0.90   South-Ceiling
S  12  14  11  10   0    0   11   0.90   Angle-Ceiling
S  13  13  12  11  14    0   11   0.90   North-Ceiling
T  1   SDE 0 1 0 1 0.5
T  3   SDRO 1
```

Listing 4: Case 4 Input File

```
V   1   8   0   0
V   2  13   0   0
V   3  13   8   0
V   4   8  13   0
V   5   0  13   0
V   6   0   8   0
V   7   8   8   0
V   8   8   0   5
V   9  13   0   5
V  10  13   8   5
V  11   8  13   5
V  12   0  13   5
V  13   0   8   5
V  14   8   8   5
!   #  v1  v2  v3  v4  base  cmb  emit   name      surface data
S   1   8   9   2   1   0    0    0.999  South−Wall−Bottom−(SDE)
S   2   9  10   3   2   0    0    0.999  East−Wall
S   3  10  11   4   3   0    0    0.1    Angle−Wall
S   4  11  12   5   4   0    0    0.999  North−Wall
S   5  12  13   6   5   0    0    0.999  West−Wall−Top
S   6  13  14   7   6   0    0    0.999  South−Wall−Top
S   7  14   8   1   7   0    0    0.999  West−Wall−Bottom
S   8   1   2   3   7   0    0    0.999  South−Floor
S   9   7   3   4   0   0    8    0.999  Angle−Floor
S  10   4   5   6   7   0    8    0.999  North−Floor
S  11  14  10   9   8   0    0    0.999  South−Ceiling
S  12  14  11  10   0   0   11    0.999  Angle−Ceiling
S  13  13  12  11  14   0   11    0.999  North−Ceiling
T  1   SDE  0  1  0  1  0.5
T  3   SDRO  0
```

Appendix—Source Code

Listing 5: DistributionFactors Module

```
MODULE DistributionFactors

USE Global
USE EnclosureGeometry
USE EnergyBundleLocation
USE IntersectionEnergySurface
USE EnergyAbsorbedReflected

IMPLICIT NONE
CONTAINS

SUBROUTINE RadDistributionFactors
!******************************************************************************
!
! PURPOSE:             Calculating the radiation distribution factor
!
!
!
!******************************************************************************
    IMPLICIT NONE
    INTEGER :: I, J, K, L, N_SCMB, IOS
    INTEGER, ALLOCATABLE, DIMENSION(:) :: NumEmitted
    INTEGER, ALLOCATABLE, DIMENSION(:, :) :: NAEnergy_cmb
    REAL(Prec2), ALLOCATABLE, DIMENSION(:) :: Area_cmb_temp, Emit_cmb_temp

    ALLOCATE(NumEmitted(NSurf), STAT = IOS)
    ALLOCATE(RAD_D_F(NSurf, NSurf), STAT = IOS)
    ALLOCATE(NAEnergy_cmb(NSurf, NSurf), STAT = IOS)
    ALLOCATE(Area_cmb_temp(NSurf), Emit_cmb_temp(NSurf), STAT = IOS)

    RAD_D_F = 0

    ! Populate array for orignial surfaces
    DO I = 1, NSurf
        DO J = 1, NSurf
            IF (TCOUNTA(I) .EQ. 0) THEN
                RAD_D_F(I, J) = 0
            ELSE
                RAD_D_F(I, J) = REAL(NAEnergy(I, J)) / REAL(TCOUNTA(I))
            ENDIF
        END DO
    END DO

    ! Now combine surfaces

    N_SCMB = 0
    ! Identify number of surface combinations
    DO I = 1, NSurf
        DO J = 1, NSurf
            IF (I == CMB(J))THEN
                N_SCMB = N_SCMB + 1
            ENDIF
        END DO
    END DO

    ! Number of surfaces after combined
    NSurf_cmb = NSurf - N_SCMB

    ALLOCATE(RAD_D_F_cmb(NSurf_cmb, NSurf_cmb), STAT = IOS)
    ALLOCATE(Area_cmb(NSurf_cmb), Emit_cmb(NSurf_cmb), STAT = IOS)

    ! Copy over to arrays we can edit
    NAEnergy_cmb = NAEnergy

    ! Initialize arrays
    RAD_D_F_cmb = 0
    NumEmitted = 0

    ! Combine count
    DO I = 1, NSurf

        IF (CMB(I) .gt. 0) THEN
            NumEmitted(I) = 0
        ELSE
        NumEmitted(I) = TCOUNTA(I)
        ENDIF
```

```fortran
    DO J = 1, NSurf
        IF (I == CMB(J)) THEN
            NumEmitted(I) = NumEmitted(I) + TCOUNTA(J)
        ENDIF
    END DO
END DO

! Combine columns
DO I = 1, NSurf
    DO J = 1, NSurf
        IF (CMB(J) .gt. 0 ) THEN
            ! Diffuse rays
            NAEnergy_cmb(I, CMB(J)) = NAEnergy_cmb(I, CMB(J)) + NAEnergy_cmb(I, J)
            NAEnergy_cmb(I, J) = 0
        ENDIF
    END DO
END DO

! Combine rows
DO I = 1, NSurf
    DO J = 1, NSurf
        IF (CMB(I) .gt. 0 ) THEN
            ! Diffuse rays
            NAEnergy_cmb(CMB(I), J) = NAEnergy_cmb(CMB(I), J) + NAEnergy_cmb(I, J)
            NAEnergy_cmb(I, J) = 0
        ENDIF
    END DO
END DO

! Copy to new reduced arrays
K = 0
DO I = 1, NSurf
    IF (CMB(I) .gt. 0) THEN
        CYCLE
    ELSE
        K = K + 1
        L = 0
        DO J = 1, NSurf
            IF (CMB(J) .gt. 0) THEN
                CYCLE
            ELSE
                L = L + 1

                IF (NumEmitted(I) == 0) THEN
                    RAD_D_F_cmb(K, L) = 0
                ELSE
                    RAD_D_F_cmb(K, L) = REAL(NAEnergy_cmb(I, J)) / REAL(NumEmitted(I))
                END IF
            END IF
        END DO
    END IF
END DO

! Combined surface areas
! Combined surface emittances
! Use area weighting
Area_cmb = 0
Area_cmb_temp = 0
Emit_cmb = 0
Emit_cmb_temp = 0

DO I = 1, NSurf
    IF (CMB(I) == 0) THEN
        Area_cmb_temp(I) = Area(I)
        Emit_cmb_temp(I) = Emit(I) * Area(I)
    ELSE
        Area_cmb_temp(CMB(I)) = Area_cmb_temp(CMB(I)) + Area(I)
        Emit_cmb_temp(CMB(I)) = Emit_cmb_temp(CMB(I)) + Emit(I) * Area(I)
    END IF
END DO

J = 0
DO I = 1, NSurf
    IF (Area_cmb_temp(I) == 0) THEN
        CYCLE
    ELSE
        J= J + 1
        Area_cmb(J) = Area_cmb_temp(I)
        Emit_cmb(J) = Emit_cmb_temp(I) / Area_cmb(J)
    END IF
END DO

RETURN
```

```
END SUBROUTINE RadDistributionFactors

END MODULE DistributionFactors
```

Listing 6: EnclosureGeometry Module

```
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
USE StringUtility
IMPLICIT NONE

CONTAINS

SUBROUTINE CalculateGeometry()

    IMPLICIT NONE
    INTEGER :: I, VertIndex, SurfIndex, TypeIndex, IOS
    CHARACTER (Len = 12) :: ReadStr
    REAL(Prec2) TotalReflec

    NVertex = 0
    NSurf = 0

    ! Count verticies and surfaces so we can allocate arrays
    DO
        ReadStr = ''
        READ (2, *, IOSTAT = IOS) ReadStr
        IF (StrLowCase(TRIM(ReadStr)) == "v") THEN
            NVertex = NVertex + 1
        ELSE IF (StrLowCase(TRIM(ReadStr)) == "s") THEN
            NSurf = NSurf + 1
        END IF

        IF (IS_IOSTAT_END(IOS)) THEN
            EXIT
        ENDIF
    END DO

    REWIND(2)

    !  Allocate the size of the array
    ALLOCATE(V(NVertex), XS(NVertex), YS(NVertex), ZS(NVertex), STAT = IOS)
    ALLOCATE(SNumber(NSurf), SVertex(NSurf, NSurf), SType(NSurf), BaseP(NSurf), CMB(NSurf), Emit(NSurf), SurfName(NSurf), STAT = IOS)
    ALLOCATE(DirectionX(NSurf), DirectionY(NSurf), DirectionZ(NSurf), SpecReflec(NSurf), DiffReflec(NSurf), FracSpecEmit(NSurf),
          FracSpecReflec(NSurf))

    ! Initialize arrays
    DirectionX = 0
    DirectionY = 0
    DirectionZ = 0
    SpecReflec = 0
    DiffReflec = 0
    FracSpecEmit = 0
    FracSpecReflec = 0

201 FORMAT(A1, ' ', I2, 3(' ', f6.3))
203 FORMAT(A1, 5('  ', I2), 1('  ',f6.3), 1('  ',I2), 1('  ',f6.3), A15)

    ! Now, read all data into arrays
    DO
        ReadStr = ''
        READ (2, *, IOSTAT = IOS) ReadStr
        IF (StrLowCase(TRIM(ReadStr)) == "v") THEN

            ! Read in vertex information
            BACKSPACE(2)
            READ (2, *) ReadStr, VertIndex, XS(VertIndex), YS(VertIndex), ZS(VertIndex)
            V(VertIndex) = VertIndex

            ! Write vertex data for RTVT
```

```
            IF (WriteLogFile) THEN
                WRITE(4, 201, ADVANCE = 'YES') TRIM(ReadStr), VertIndex, XS(VertIndex), YS(VertIndex), ZS(VertIndex)
            END IF

        ELSE IF (StrLowCase(TRIM(ReadStr)) == "s") THEN

            ! Read in surface information
            BACKSPACE(2)
            READ (2, *) ReadStr, SurfIndex, (SVertex(SurfIndex, I), I = 1, 4), BaseP(SurfIndex), CMB(SurfIndex), Emit(SurfIndex), SurfName(
                  SurfIndex)
            SNumber(SurfIndex) = SurfIndex

            ! Write surface data for RTVT
            IF (WriteLogFile) THEN
                WRITE(4, 203) TRIM(ReadStr), SurfIndex, (SVertex(SurfIndex, I), I = 1, 4), BaseP(SurfIndex), CMB(SurfIndex), Emit(SurfIndex
                      ), SurfName(SurfIndex)
            END IF

        ELSE IF (StrLowCase(TRIM(ReadStr)) == "t") THEN

            ! Read in surface type information
            BACKSPACE(2)
            READ(2, *) ReadStr, TypeIndex, SType(TypeIndex)
            BACKSPACE(2)
            IF (StrUpCase(SType(TypeIndex)) == "SDE") THEN
                READ(2, *) ReadStr, SNumber(TypeIndex), SType(TypeIndex), DirectionX(TypeIndex), DirectionY(TypeIndex), DirectionZ(
                      TypeIndex), FracSpecEmit(TypeIndex), FracSpecReflec(TypeIndex)
            ELSE IF (StrUpCase(SType(TypeIndex)) == "SDRO") THEN
                READ(2, *) ReadStr, SNumber(TypeIndex), SType(TypeIndex), FracSpecReflec(TypeIndex) ! Reading in specular and diffuse
                      reflection
            ELSE
                READ(2, *) ReadStr, SNumber(TypeIndex), SType(TypeIndex)
            END IF
        END IF

        IF (IS_IOSTAT_END(IOS)) THEN
            EXIT
        ENDIF
    END DO

    ! Check for out of range specular/emitting fractions
    DO I = 1, NSurf
        IF (FracSpecEmit(I) .gt. 1) THEN
            FracSpecEmit(I) = 1
        ELSE IF (FracSpecEmit(I) .lt. 0) THEN
            FracSpecEmit(I) = 0
        END IF

        IF (FracSpecReflec(I) .gt. 1) THEN
            FracSpecReflec(I) = 1
        ELSE IF (FracSpecReflec(I) .lt. 0) THEN
            FracSpecReflec(I) = 0
        END IF
    END DO

    ! If no surface types were provided, set them to DIF here
    ! Update reflectance/emittance values and fractions here
    DO I = 1, NSurf
        IF (StrUpCase(SType(I)) == "SDE") THEN
            TotalReflec = 1 - Emit(I)
            DiffReflec(I) = TotalReflec * (1 - FracSpecReflec(I))
            SpecReflec(I) = TotalReflec * FracSpecReflec(I)
        ELSE IF (StrUpCase(SType(I)) == "SDRO") THEN
            TotalReflec = 1 - Emit(I)
            DiffReflec(I) = TotalReflec * (1 - FracSpecReflec(I))
            SpecReflec(I) = TotalReflec * FracSpecReflec(I)
        ELSE
            SType(I) = "DIF"
            DiffReflec(I) = 1 - Emit(I)
        END IF
    END DO

    CLOSE(Unit = 2)

END Subroutine  CalculateGeometry

SUBROUTINE CalculateSurfaceEquation()
!******************************************************************************
!
!  SUBROUTINE:    CalculateSurfaceEquation
!
!  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), 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))
       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 CalculateSurfaceEquation

SUBROUTINE SurfaceNormal(Vx, Vy, Vz)
!******************************************************************************
!
!  PURPOSE:        Determine normal unit vector of the surfaces in the enclosure
!
!
!******************************************************************************
    IMPLICIT NONE
```

```fortran
    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

    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 CalculateAreaSurfaces()
!******************************************************************************
!
!  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 CalculateAreaSurfaces

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 AllocateAndInitArrays()
!******************************************************************************
!
!  PURPOSE:         Allocates the arrays
!
!
!******************************************************************************

    IMPLICIT NONE
    INTEGER :: I, J, IOS

    ALLOCATE(NAEnergy(NSurf, NSurf))
    ALLOCATE(TCOUNTA(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 (EmittedUV(NSurf, 3), STAT = IOS )

    NAEnergy = 0
    TCOUNTA = 0
    XLS = 0
    YLS = 0
    ZLS = 0
    XP = 0
    YP = 0
    ZP = 0
    Intersection = 0
    Xo = 0
    Yo = 0
    Zo = 0
    EmittedUv = 0

END SUBROUTINE AllocateAndInitArrays

END MODULE EnclosureGeometry
```

Listing 7: EnergyAbsorbedReflected Module

```
MODULE EnergyAbsorbedReflected


USE Global
```

```fortran
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_incident
    REAL(Prec2) :: XPVal

    !   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_incident = Rand(6)
    ReflectedSpec = .false.

    ! Write point data to RTVT file
    IF (WriteLogFile) THEN
        IF (ReflecCount == 0) THEN
101         FORMAT(A1, 1('␣', I2, 3('␣', f6.3)))
            WRITE(4, 101, ADVANCE = 'NO')'P', SIndex, XLS(SIndex), YLS(SIndex), ZLS(SIndex)
        END IF
    END IF

    IF (R_incident .lt. DiffReflec(SInter)) THEN
        ! Reflect Diffusely
        Reflected = .true.
        ReflecCount = ReflecCount + 1
    ELSE IF ((DiffReflec(SInter) .lt. R_incident) .and. (R_incident .lt. (DiffReflec(SInter) + SpecReflec(SInter)))) THEN
        ! Reflect Specularly
        Reflected = .true.
        ReflectedSpec = .true.
        ReflecCount = ReflecCount + 1
    ELSE
        ! Absorb
        RayAbsorbed = .true.
        NAEnergy(SIndexRef, SInter) = NAEnergy(SIndexRef, SInter) + 1
    END IF

    ! Write point data to RTVT file
    IF (WriteLogFile) THEN
        IF (RayAbsorbed) THEN
111         FORMAT(1('␣', I2, 3('␣', f6.3)), '␣' '0')
            XPVal = XP(SIndex, SInter)
            WRITE(4, 111, ADVANCE = 'YES') SInter, XPVal, YP(SIndex, SInter), ZP(SIndex, SInter)
        ELSE
121         FORMAT(1('␣', I2, 3('␣', f6.3)))
            WRITE(4, 121, ADVANCE = 'NO') SInter, XP(SIndex, SInter), YP(SIndex, SInter), ZP(SIndex, SInter)
        END IF
    ENDIF

    PrevSurf = SIndex
    SIndex = SInter

END SUBROUTINE AbsorptionReflection
END MODULE EnergyAbsorbedReflected
```

Listing 8: EnergyBalance Module

```fortran
 MODULE EnergyBalance

USE Global
USE EnclosureGeometry
USE EnergyBundleLocation
USE IntersectionEnergySurface
USE EnergyAbsorbedReflected
```

```
USE DistributionFactors

IMPLICIT NONE
CONTAINS

SUBROUTINE RadiationBalance
!*******************************************************************************
!
! PURPOSE:          Calculating the net radiation flux at each surface using
!                   the gray view factor or the radiation distribution factor
!
!
!*******************************************************************************
    IMPLICIT NONE
    INTEGER  :: I, J, LWL, UPL, IOS
    INTEGER, ALLOCATABLE, DIMENSION(:) :: EB
    REAL(Prec2) :: SIGMA, EBSUM, T

    SIGMA = 5.67E-8     ! Stephan 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 RadiationBalance

END MODULE EnergyBalance
```

Listing 9: EnergyBundleLocation Module

```
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
```

```fortran
!               the appropriate subroutine. If the fourth vertex index is zero
!               then the polygon is triangular, else it is rectangular
!
!*******************************************************************************
    CALL RANDOM_NUMBER(Rand)
    IF(PolygonIndex(SIndex) .eq. 4)THEN
        CALL RectangularSurface()
    ELSEIF( PolygonIndex(SIndex) .eq. 3)THEN
        CALL TriangularSurface()
    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
```

```fortran
    IMPLICIT NONE
    INTEGER :: J
    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)
        ENDIF

        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.'
        ENDIF
    ENDIF
END SUBROUTINE RectangularSurface

SUBROUTINE InitializeSeed()
!******************************************************************************
!
!   PURPOSE:    Initialization of seed for the random number generator
!
!
!******************************************************************************
    IMPLICIT NONE
    INTEGER :: i, n, clock
    INTEGER, DIMENSION(:), ALLOCATABLE :: seed

    CALL RANDOM_SEED(size = n)
    ALLOCATE(seed(n))

    CALL SYSTEM_CLOCK(COUNT = clock)

    seed = clock + 104729 * (/ (i - 1, i = 1, n) /)
    CALL RANDOM_SEED(PUT = seed)

    DEALLOCATE(seed)
END SUBROUTINE InitializeSeed

SUBROUTINE TangentVectors()
!*******************************************************************************
!
!   PURPOSE:    Determines unit tangent vectors on a surface in the enclosure
!
!
!*******************************************************************************
!   UV_X(3)          Unit vector along X-direction
```

```fortran
!   UV_Y(3)          Unit vector along Y-direction
!   UV_Z(3)          Unit vector along Z-direction
!   TUV1(3)          Unit vector tangent to the source point on a surface
!   TUV2(3)          Unit vector tangent to the source point on a surface
!                    and normal to the TUV1 tangent vector
!                    The tangent vectors are used for reference in defining the angle
!                    Thus, need to be determined once for each surface
!   SmallestRealNo  The smallest machine number

    IMPLICIT NONE
    INTEGER :: I, J, K, IOS, INDEX
    REAL(Prec2) ::  UV_x(3), UV_y(3), UV_z(3), V(3), TUV1(3), TUV2(3), VDOT(3)
    REAL(Prec2) ::  SmallestRealNo, NV, xx

    ! define the smallest machine number
    SmallestRealNo = EPSILON(0.0d0)

    ALLOCATE(Tan_V1(NSurf, 3), Tan_V2(NSurf, 3), STAT = IOS)
    DO I = 1, 3
        Tan_V1(SIndex, I) = 0.0
        Tan_V2(SIndex, I) = 0.0
        V(I) = NormalUV(SIndex, I)
        UV_x(I) = 0.0
        UV_y(I) = 0.0
        UV_z(I) = 0.0
    END DO

    UV_x(1) = 1.0
    UV_y(2) = 1.0
    UV_z(3) = 1.0

    ! The first tangent vector is determined first as follows
    VDOT(1) = DOT_PRODUCT(V, UV_x)
    VDOT(2) = DOT_PRODUCT(V, UV_y)
    VDOT(3) = DOT_PRODUCT(V, UV_z)
    IF((1.0 - ABS(VDOT(1))) .gt. SmallestRealNo)THEN
        CALL CrossProduct(V, UV_x, TUV1)
    ELSEIF((1.0 - ABS(VDOT(2))) .gt. SmallestRealNo)THEN
        CALL CrossProduct(V, UV_y, TUV1)
    ELSE
        CALL CrossProduct(V, UV_z, TUV1)
    ENDIF

    NV = 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
    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)

    IF (.not. Reflected) THEN
        ! Emitted rays
        IF ((REAL(BIndex) / REAL(NBundles)) .le. FracSpecEmit(SIndex)) THEN
            ! Emit Specularly
            EmittedUV(SIndex, 1) = DirectionX(SIndex)
            EmittedUV(SIndex, 2) = DirectionY(SIndex)
            EmittedUV(Sindex, 3) = DirectionZ(SIndex)
        ELSE
            ! Emit Diffusely
            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
    ELSE
        ! Reflected rays
        IF (ReflectedSpec) THEN
            ! Specularly reflected rays
            SurfNorm(1) = NormalUV(SIndex, 1)  !Surface normal unit vector
            SurfNorm(2) = NormalUV(SIndex, 2)
            SurfNorm(3) = NormalUV(SIndex, 3)

            InVecDirec(1) = - EmittedUV(PrevSurf, 1)
            InVecDirec(2) = - EmittedUV(PrevSurf, 2)
            InVecDirec(3) = - EmittedUV(PrevSurf, 3)

            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
            ! Diffusely reflected rays
            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 IF
END SUBROUTINE  DirectionEmittedEnergy

SUBROUTINE CheckDirection()

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

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

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

END SUBROUTINE

END MODULE EnergyBundleLocation
```

Listing 10: Global Module

```
MODULE Global
!       OKlahoma State University
!       School of Mechanical And Aerospace Engineering
!
!
!       PURPOSE:  Global Data for Program Monte Carlo Method
!

IMPLICIT NONE
SAVE
```

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

INTEGER :: NSurf                        ! Number of Surfaces
INTEGER :: NSurf_cmb                    ! Number of Surfaces after combination
INTEGER :: SIndex                       ! Surface counting Index
INTEGER :: SIndexRef                    ! Surface counting Index Reference
INTEGER :: NVertex                      ! Number of Vertices
INTEGER :: NBundles                     ! Number of Energy Bundles Emitted

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(:) :: 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(:)  :: Emit_cmb     ! Emissivities of combined 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(:) :: SurfName   ! Name of Surfaces
LOGICAL :: Reflected
LOGICAL :: ReflectedSpec
LOGICAL, ALLOCATABLE, DIMENSION(:) :: Intersects ! Surface Intersection Flag
LOGICAL :: WriteLogFile                         ! Flag to indicate whether log file should be written

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_F_cmb ! Diffuse Radiation Distribution Factor for combined surfaces

REAL(Prec2), ALLOCATABLE, DIMENSION(:) ::   Area         ! Area of a surface
REAL(Prec2), ALLOCATABLE, DIMENSION(:) ::   Area_cmb     ! Area of a combined surfaces

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 = 4), ALLOCATABLE, DIMENSION(:) ::  SType ! 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

    REAL(Prec2), ALLOCATABLE, DIMENSION(:) ::  FracSpecEmit  !- Fraction of total rays emitted which are specular
    REAL(Prec2), ALLOCATABLE, DIMENSION(:) ::  FracSpecReflec     !- Fraction of total rays reflected which are specular

    INTEGER :: PrevSurf
    INTEGER :: BIndex       !- Bundle index
    LOGICAL :: RayAbsorbed  !- Has bundle been absorbed flag
    INTEGER :: ReflecCount

 END MODULE Global
```

Listing 11: IntersectionEnergySurface Module

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

  USE Global
  USE EnclosureGeometry
  USE EnergyBundleLocation
  USE EnergyAbsorbedReflected

  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 IntersectionPoints & SingleOutIntersection
!
!****************************************************************************

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

    CALL IntersectionPoints()
    CALL SingleOutIntersection()

END SUBROUTINE CheckingIntersection

SUBROUTINE IntersectionPoints()
!****************************************************************************
!
!  SUBROUTINE:  IntersectionPoints
!
!  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
        ENDIF
    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 IntersectionPoints

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
    REAL(Prec2) XPVal

    !   SIMIN        the closest intersection distance
    !   SIMAX        Maximum real number
    !   Assign the maximum REAL number to SIMAX

    SIMAX  = 10000000000000000.0
```

```fortran
    Allocate(SIINTER(NSurf), STAT = IOS)

    !  Calculates the vector position of the intersection point
    DO Index = 1, NSurf
        IF (Index .ne. SIndex) THEN
            XPVal = XLS(SIndex) + SI(Index) * EmittedUV(SIndex, 1)
            XP(SIndex, Index) = XPVal
            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
            ENDIF
        ENDIF
    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 IntersectionEnergySurface
```

Listing 12: MainMonteCarlo Program

```
PROGRAM MainMonteCarlo
    ! 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
    ! Matt Mitchell for MAE 5823, December 2017

    USE Global
```

```fortran
USE EnclosureGeometry
USE EnergyBundleLocation
USE IntersectionEnergySurface
USE EnergyAbsorbedReflected
USE DistributionFactors
USE EnergyBalance
USE Output

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

! Initialize the CPU time
CALL CPU_TIME(TIME1)

OPEN(Unit = 2, file = 'input.vs3', status = 'unknown', Action = 'READ', IOSTAT = IOS)
OPEN(Unit = 3, file = 'MC-Output.txt', status = 'unknown', IOSTAT = IOS)     ! Diffuse bundles and distribution factors
OPEN(Unit = 4, file = 'logfile.dat', status = 'unknown', IOSTAT = IOS)       ! Ray emission, reflection, and absorption points
OPEN(Unit = 7, File = 'input.TK', status = 'unknown', IOSTAT = IOS)          ! Surface temperatures
OPEN(Unit = 8, File = 'parameters.txt', status = 'old', IOSTAT = IOS)        ! Geometry and ray data for RTVT
OPEN(Unit = 12, File = 'MC-Output.csv', status = 'unknown', IOSTAT = IOS)     ! csv file with diffuse distribution factors

! Read simulation parameters
READ(8, *) NBundles
READ(8, *) logfileint
CLOSE(Unit = 8)

IF (logfileint == 1) THEN
    WriteLogFile = .true.
ELSE
    WriteLogFile = .false.
ENDIF

WRITE(*, *) "Loading Geometry"
CALL CalculateGeometry()

WRITE(*, *) "Initializing Variables"
CALL InitializeSeed()
CALL AllocateAndInitArrays()

WRITE(*, *) "Calculating Surface Areas"
DO SIndex = 1, NSurf
    CALL CalculateSurfaceEquation()
    CALL CalculateAreaSurfaces()
    CALL TangentVectors()
END DO

WRITE(*, *) "Evaluating Surface Energy Bundles"

DO SIndexRef = 1, NSurf

    ! This surface only reflects, so we don't need to compute emitted values
    IF (SType(SIndexRef) == "SDRO" ) THEN
        CYCLE
    ELSE

        ! Run for all bundles
        DO BIndex = 1, NBundles
            TCOUNTA(SIndexRef) = TCOUNTA(SIndexRef) + 1
            SIndex = SIndexRef
            Reflected = .False.
            RayAbsorbed = .false.
            ReflecCount = 0
            ! Run until absorbed
            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 (RayAbsorbed) THEN
                    EXIT
                END IF
            END DO
        END DO
    END IF
```

```
        ! Update progress bar
        CALL Progress(SIndexRef, NSurf)

    END DO

    ! Calculate the radiation distribution factor
    WRITE(*, *) "Calculating Distribution Factors"
    CALL RadDistributionFactors()

    ! Calculate the heat balance of the enclosure
    WRITE(*, *) "Evaluating Radiation Balance"
    CALL RadiationBalance()

    WRITE(*, *) "Simulaton Complete"

    ! Calculate the CPU Time
    CALL CPU_TIME(TIME2)

    ! Write Results to a file
    CALL PrintViewFactorHeatFlux

    CLOSE(UNIT = 2)
    CLOSE(Unit = 3)
    CLOSE(Unit = 4)
    CLOSE(Unit = 7)
    CLOSE(Unit = 12)
    STOP
END PROGRAM MainMonteCarlo
```

Listing 13: Output Module

```
MODULE Output

USE Global
USE EnclosureGeometry
USE EnergyBundleLocation
USE IntersectionEnergySurface
USE EnergyAbsorbedReflected
USE DistributionFactors
USE EnergyBalance

IMPLICIT NONE
CONTAINS

SUBROUTINE Progress(Surf, TotNumSurfs)
    ! Writes status bar
    ! Original source from here: https://software.intel.com/en-us/forums/intel-fortran-compiler-for-linux-and-mac-os-x/topic/270155

    IMPLICIT NONE
    INTEGER(KIND = 4) :: K, Surf, TotNumSurfs, PercentComplete
    CHARACTER(len = 17) :: bar="???% |          |"
    PercentComplete = FLOOR(100 * REAL(Surf) / REAL(TotNumSurfs))
    WRITE(unit = bar(1:3), fmt = "(i3)") PercentComplete
    DO K = 1, FLOOR(10 * REAL(Surf) / REAL(TotNumSurfs))
        bar(6 + K : 6 + K)="*"
    END DO
    ! print the progress bar.
    WRITE(*, fmt = "(a1,a1,a17)") '+', CHAR(13), bar
    RETURN
END SUBROUTINE progress

SUBROUTINE PrintViewFactorHeatFlux()
    !********************************************************************************
    !
    ! PURPOSE:           Prints View Factors, Radiation Heat Flux 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)
    END DO
```

```
1001 FORMAT(2x, 100(x, I8), I10)

     WRITE(3, 1002)
     WRITE(6, 1002)

1002 FORMAT(//)

     DO Index = 1, NSurf_cmb
        WRITE(3, 102)(RAD_D_F_cmb(Index, J), J = 1, NSurf_cmb)   ! Diffuse distribution factors
     END DO

    ! Write csv file for combined surface
302  FORMAT(f10.6, 100(',', f10.6))
    WRITE(12, 302)(Area_cmb(I), I = 1, NSurf_cmb)
    DO I = 1, NSurf_cmb
        WRITE(12, 302)(RAD_D_F_cmb(I, J), J = 1, NSurf_cmb)
    END DO
    WRITE(12, 302)(Emit_cmb(I), I = 1, NSurf_cmb)

    ! Writing the rest of the outputs for MCOutput.txt
102  FORMAT(4x, 100(2x, f10.6))

     WRITE(3, 103)'Index', 'SurfName', '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, SurfName(Index), TS(Index), Emit(Index), QFLUX(Index), Q(Index)
104     FORMAT(7x, I3, 8x, A12, 4x, F7.2, 8x, F5.2, 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 PrintViewFactorHeatFlux
 END MODULE Output
```

Listing 14: StringUtility Module

```
MODULE StringUtility

    ! Original source: http://computer-programming-forum.com/49-fortran/4075a24f74fcc9ce.htm

    IMPLICIT NONE
    PRIVATE
    PUBLIC :: StrUpCase
    PUBLIC :: StrLowCase
    CHARACTER( * ), PRIVATE, PARAMETER :: LOWER_CASE = 'abcdefghijklmnopqrstuvwxyz'
    CHARACTER( * ), PRIVATE, PARAMETER :: UPPER_CASE = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
CONTAINS

FUNCTION StrUpCase ( Input_String ) RESULT ( Output_String )
    ! -- Argument and result
    CHARACTER( * ), INTENT( IN )     :: Input_String
    CHARACTER( LEN( Input_String ) ) :: Output_String
    ! -- Local variables
    INTEGER :: i, n
    ! -- Copy input string
    Output_String = Input_String
    ! -- Loop over string elements
    DO i = 1, LEN( Output_String )
        ! -- Find location of letter in lower case constant string
        n = INDEX( LOWER_CASE, Output_String( i:i ) )
        ! -- If current substring is a lower case letter, make it upper case
        IF ( n /= 0 ) Output_String( i:i ) = UPPER_CASE( n:n )
    END DO
END FUNCTION StrUpCase

FUNCTION StrLowCase ( Input_String ) RESULT ( Output_String )
    ! -- Argument and result
    CHARACTER( * ), INTENT( IN )     :: Input_String
    CHARACTER( LEN( Input_String ) ) :: Output_String
    ! -- Local variables
    INTEGER :: i, n
    ! -- Copy input string
    Output_String = Input_String
    ! -- Loop over string elements
    DO i = 1, LEN( Output_String )
```

```fortran
        ! -- Find location of letter in upper case constant string
        n = INDEX( UPPER_CASE, Output_String( i:i ) )
        ! -- If current substring is an upper case letter, make it lower case
        IF ( n /= 0 ) Output_String( i:i ) = LOWER_CASE( n:n )
    END DO
END FUNCTION StrLowCase

END MODULE StringUtility
```

Listing 15: Resource File

```fortran
!MS$FREEFORM
! Microsoft Developer Studio generated include file.
! Used by Script1.rc
!
      integer, parameter :: IDD_DIALOG1                    = 101
```