SANDIA REPORT
SAND2009-4181
Unlimited Release
Printed July 2009

Solar Mechanics Thermal Response Capabilities

Dean Dobranich

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2009-4181
Unlimited Release
Printed July 2009

Solar Mechanics Thermal Response Capabilities

Dean Dobranich
Thermal and Fluid Processes

Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-0346

Abstract

In many applications, the thermal response of structures exposed to solar heat loads is of interest. Solar mechanics governing equations were developed and integrated with the Calore thermal response code via user subroutines to provide this computational simulation capability. Solar heat loads are estimated based on the latitude and day of the year. Vector algebra is used to determine the solar loading on each face of a finite element model based on its orientation relative to the sun as the earth rotates. Atmospheric attenuation is accounted for as the optical path length varies from sunrise to sunset. Both direct and diffuse components of solar flux are calculated. In addition, shadowing of structures by other structures can be accounted for. User subroutines were also developed to provide convective and radiative boundary conditions for the diurnal variations in air temperature and effective sky temperature. These temperature boundary conditions are based on available local weather data and depend on latitude and day of the year, consistent with the solar mechanics formulation. These user subroutines, coupled with the Calore three-dimensional thermal response code, provide a complete package for addressing complex thermal problems involving solar heating. The governing equations are documented in sufficient detail to facilitate implementation into other heat transfer codes. Suggestions for improvements to the approach are offered.

Acknowledgements

Many thanks to Ronald C. Dykhuizen for his advice and for directing me to reference material regarding the solar mechanics approach developed in this work. Appreciation is also extended to Ron and to Samuel R. Subia for technical review of this report.

Contents

Figures

Introduction

Thermal response codes enable the simulation of heat transfer to and within a structure. It is often necessary to include solar loading as a heat flux boundary condition, accounting for the trajectory of the sun as it passes over the structure. Such a solar mechanics capability was developed via user subroutines for the Calore[1] thermal response code. This finite element code provides three-dimensional transient heat conduction capabilities allowing specification of heat flux, radiative, and convective boundary conditions, along with specification of thermal radiation enclosures.

The solar mechanics capability enables calculation of the solar flux on all surfaces of a structure as a function of time depending on the latitude and day of the year. Both direct and diffuse flux components are computed, accounting for atmospheric attenuation. Vector algebra is used to calculate the orientation-dependent flux on each exposed finite element face of the structure. In addition, shadowing of surfaces by other surfaces is accounted for. Along with the solar mechanics user subroutine, two other companion subroutines were developed. One subroutine specifies the diurnal variation of air temperature to be used for convective boundary conditions on the structure, via Newton's law of cooling. Input for this subroutine includes yearly minimum and maximum air temperatures for the geographical location of interest along with latitude and day of the year. The other subroutine uses this diurnal air temperature to calculate an effective sky temperature to be used for radiative boundary conditions on the structure. Together, these three Calore user subroutines provide a general-purpose solar mechanics thermal response capability useful for a variety of engineering problems. The solar mechanics governing equations are documented in sufficient detail to enable their implementation into other thermal response codes as desired.

The governing equations for the solar mechanics implementation are described in detail in the next section, with subsections for each of the contributing terms. One subsection provides a brief description of a possible implementation for the reflected component of solar flux, although this implementation was not exercised because of limitations in the user subroutine capability. The following section provides several example problems that demonstrate the solar mechanics capability. This is followed by a comments and summary section in which comments are offered regarding the solar mechanics capability along with recommendations for future efforts.

[1] "Calore—A Computational Heat Transfer Program," SAND2008-0098P, February 12, 2008.

Governing Equations

The basic equation governing the calculation of absorbed solar flux is based on an adaptation of a method outlined in the ASHRAE Handbook.[2] The governing equation, as developed here, expresses absorbed solar heat flux as a function of several factors and is given as

$$q''_{abs} = \alpha q''_{inc} = \alpha q''_{am1} f_{ext} \left(f_{or} f_{clear} f_{sh} + f_{dif} \right) + \alpha q''_{refl} \qquad (1)$$

where q''_{abs} is the absorbed solar heat flux on a surface, α is the solar absorptivity of the surface, q''_{inc} is the total incident solar heat flux, q''_{am1} is the apparent solar heat flux on a horizontal surface at sea level if the sun were directly overhead, corresponding to an air mass of unity, f_{ext} is a factor that accounts for atmospheric extinction (i.e., absorption), f_{or} is a factor that accounts for the orientation of the surface with respect to the position of the sun, f_{clear} is a clearness factor that accounts for cloud cover, smog, elevation, etc. and equals unity for a clear day (a value as high as 1.15 can be selected for a very clear day, and a day with 20% cloud cover, for example, corresponds to a value of 0.8), f_{sh} is a factor to account for shadowing and equals zero or unity depending on whether the surface is shadowed or not, f_{dif} is a factor that accounts for the diffuse component of solar flux as a result of atmospheric scattering, ranging from 0.05 to 0.15 depending on time of the year and other atmospheric conditions, with 0.1 a typical clear-day value, and q''_{refl} is the reflected component of the total incident flux, which depends on the direct and diffuse components.* Two of the factors, f_{clear} and f_{dif}, are specified by the analyst to reflect local atmospheric conditions.# Determination of the other factors requires additional orientation-dependent computations as presented in following subsections.

---

[2] ASHRAE Fundamentals Handbook (SI), 1997.

* The reflected component of incident flux is not included in this implementation because of limitations in the user-subroutine capability within Calore. However, suggestions for its implementation are offered in a subsequent subsection.

These two factors could be implemented as time-dependent factors to account for changing atmospheric conditions throughout the day or year. For example, it might be desirable to account for daily changing cloud coverage. As currently implemented, the two factors represent constant average values. The clearness factor can be considered simply as a correction factor to the direct component of solar flux, allowing the user to easily modify the resulting flux as desired. The diffuse factor can also be modified by the user to reflect atmospheric conditions of interest.

Solar Flux Yearly Variation

Air mass is a commonly used term in the solar community that is a relative measure of the optical path length through the atmosphere. An air mass of unity corresponds to the path length at sea level if the sun were directly overhead. (This does not necessarily correspond to the solar altitude at solar noon, where solar noon refers to the time when the sun is at its maximum altitude angle, which depends on latitude and the day of the year.) An air mass of zero indicates a location above the atmosphere.

The solar heat flux on a horizontal surface for an air mass of unity is based on worldwide average measurements throughout the year as provided in Reference 2. This data, based on 1964 measurements on the 21st day of each month, was fit to a sine function as given by

$$q''_{am1} = 1160 + 74 \sin\left[\frac{360^\circ}{365}(J + 88)\right] \text{ W/m}^2 \tag{2}$$

where J is the Julian day ranging from 1 to 365, with 1 corresponding to January 1st. A plot of this equation is provided in Figure 1. The green vertical dashed lines indicate the approximate times in the northern hemisphere for the spring equinox (J = 81), the summer solstice (J = 172), the autumn equinox (J = 265), and the winter solstice (J = 355). The figure also includes the solar flux outside the atmosphere, where the air mass equals zero. The solar flux changes as a result of the varying distance between the earth and the sun throughout the year. The earth is closest to the sun on January 3rd (J = 3) and furthest from the sun on July 4th (J = 185). The exoatmospheric flux is not used in the method to determine absorbed surface flux but is presented as data of interest. The related equation[3] is given as

$$q''_{exo} = 1373 + 45.31 \sin\left[\frac{360^\circ}{365}(J + 10)\right] \text{ W/m}^2 \tag{3}$$

[3] W. M. Rohsenow, J. P. Hartnett, and E. N. Ganic, **Handbook of Heat Transfer Applications**, 2nd Edition, McGraw-Hill, Inc., 1973.

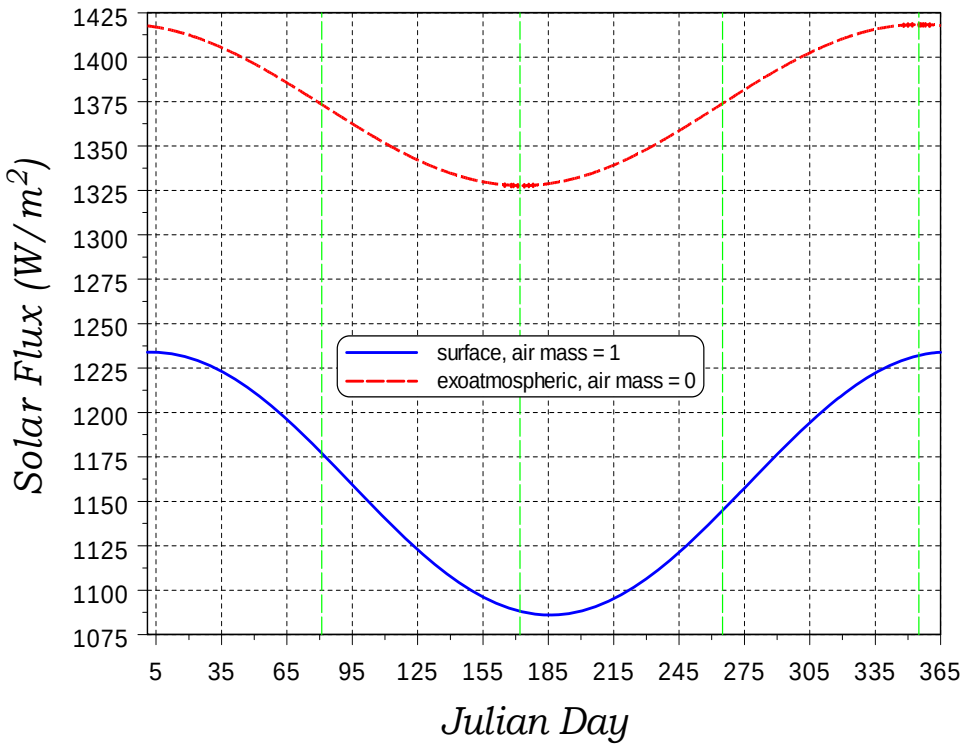

Figure 1. Solar Flux on a Horizontal Surface (air mass of unity)

Extinction Factor

The first factor in Equation (1), the extinction factor, is estimated using

$$f_{ext} = e^{-c_{ext} am} = e^{\frac{-c_{ext}}{\sin(\theta_{alt})}} \tag{4}$$

where c_{ext} is the atmospheric extinction coefficient, am is air mass, and θ_{alt} is the solar altitude, which is the angle from horizontal of a vector directed at the sun that varies between 0° at sunrise (and sunset) and a maximum of 90° at solar noon. The extinction coefficient is an analyst-specified parameter. Worldwide clear-day values[2] range from 0.142 in January to 0.207 in July. A value of 0.1 is representative of a very clear day (i.e., low humidity and no smog) while values closer to 1 are representative of very cloudy days. A value of 0.2 is typical for a clear day. Reductions in the extinction coefficient below 0.1 may be appropriate for local elevations significantly above sea level. The inverse sine of the solar altitude provides an estimate of the path length (i.e., air mass, *am*) through the atmosphere as the sun rises and sets. More accurate formulas are available in the literature to estimate the path length as the solar altitude approaches zero. However, because the solar flux is low early morning and late evening, the simpler formula is used with division by zero prevented by limiting the solar altitude to a minimum of 2°. The

extinction factor is plotted in Figure 2 as a function of solar altitude for three values of extinction coefficient.

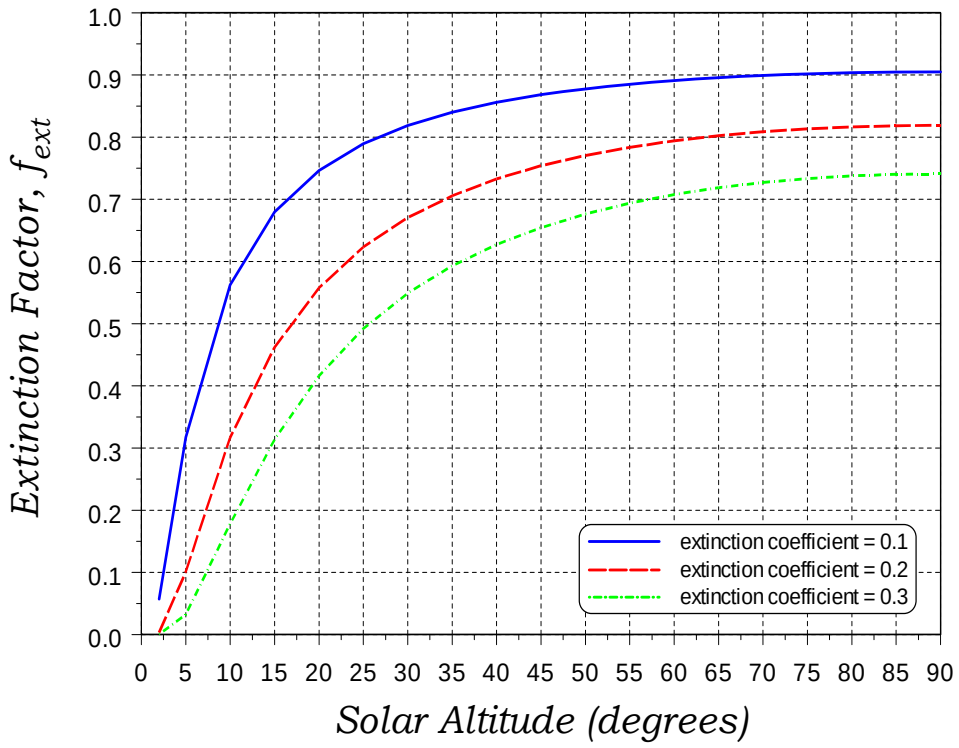

Figure 2. Extinction Factor as a Function of Solar Altitude

Orientation Factor

The next factor, f_{or}, accounts for the orientation of each exposed surface with respect to the position of the sun. Vector algebra is used to compute this factor for each finite element face in the structure model. The orientation factor is found as the dot product of the face surface normal vector and the sun vector. Thus

$$f_{or} = \mathbf{F} \bullet \mathbf{S} = F_x S_x + F_y S_y + F_z S_z \tag{5}$$

where **F** and **S** are the face normal unit vector and the sun unit vector, respectively, and the components of each vector in a rectangular coordinate system are given by F and S with x, y, z subscripts. For example, the **F** vector is written as

$$\mathbf{F} = F_x \hat{i} + F_y \hat{j} + F_z \hat{k} \tag{6}$$

where $\hat{i}$, $\hat{j}$, and $\hat{k}$ are the unit vectors along the three orthogonal coordinate directions. A unit face vector can be constructed such that

$$\mathbf{F} = l\hat{i} + m\hat{j} + n\hat{k} \tag{7}$$

where *l*, *m*, and *n* are the direction cosines given as

$$l = \frac{F_x}{F} \qquad m = \frac{F_y}{F} \qquad n = \frac{F_z}{F} \tag{8}$$

where the magnitude of vector **F** is given by the Pythagorean Theorem as

$$F = \left(F_x^2 + F_y^2 + F_z^2\right)^{1/2} \tag{9}$$

The normal vector for each finite element face is found as the cross product of two vectors on the face surface. Thus

$$\mathbf{F} = \mathbf{A} \times \mathbf{B} = \left(A_y B_z - A_z B_y\right)\hat{i} - \left(A_x B_z - A_z B_x\right)\hat{j} + \left(A_x B_y - A_y B_x\right)\hat{k} \tag{10}$$

where the two vectors, **A** and **B**, are created based on the coordinates of the finite element integration points. (Nodal coordinates could also be used but are not conveniently available in Calore user subroutines.) Thus, one vector can be constructed from integration point 1 to integration point 2, and the second vector constructed from integration point 1 to integration point 3. For example, vector **A** is determined as

$$\mathbf{A} = A_x\hat{i} + A_y\hat{j} + A_z\hat{k} = \left(x_2 - x_1\right)\hat{i} + \left(y_2 - y_1\right)\hat{j} + \left(z_2 - z_1\right)\hat{k} \tag{11}$$

where *x*, *y*, and *z* specify the coordinates of the integration points. Vector **B** is formed similarly, replacing the "2" subscript with "3". Thus

$$\mathbf{B} = \left(x_3 - x_1\right)\hat{i} + \left(y_3 - y_1\right)\hat{j} + \left(z_3 - z_1\right)\hat{k} \tag{12}$$

It should be noted that this choice of surface vectors **A** and **B** is consistent with the finite element numbering convention used in Calore and ensures that the resulting normal vector points outward.

The face vector **F** is thus constructed by first creating vectors **A** and **B** based on Equations (11) and (12), and then forming the cross product based on Equation (10). This provides the components of vector **F**, as used in Equation (6), which is then converted to a unit vector based on direction cosines using Equations (7), (8), and (9).

Face vectors must be created for every participating element face in the finite element model. These values can be computed once and stored for subsequent use provided the geometry is static. The **S** vector, derived next, changes with the position of the sun and must be reevaluated every time step before performing the dot product operation of Equation (5).

The Sun Vector

Next, a vector specifying the direction to the sun as a function of time must be created. It is convenient to adopt a Ptolemaic view in which the sun is assumed to revolve around the earth at the location of interest. Using

spherical coordinates, the sun vector can be expressed with two angles, the solar altitude, θ_{alt}, and the solar azimuth, ϕ_{azi}. The solar altitude is the angle from a local horizontal plane and a line to the center of the sun. The altitude varies from 0° to a maximum of 90°. The solar azimuth is the angle measured from either due south or due north to a line formed by a projection of the sun line (a.k.a. the sun or solar vector) onto the horizontal plane. The azimuth varies between -90° and 90°, and equals zero at solar noon when the sun lies on the north-south axis. A diagram depicting the two angles is provided in Figure 3. This figure also shows the global Cartesian coordinate system used for development of the sun vector.

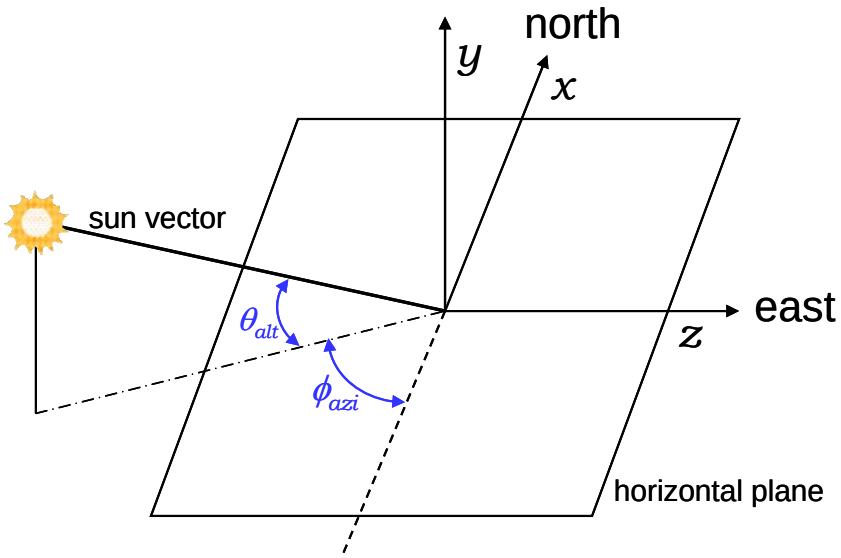

Figure 3. Diagram Depicting Solar Angles and Global Coordinate System

Two quantities are needed before the two solar angles can be computed. These two quantities are the declination and the hour angle. The declination, which changes throughout the year as the earth revolves around the sun, is the angle between the earth-sun line and the equatorial plane of the earth. The north-south rotation axis of the earth is tilted 23.45° relative to the plane of the earth's orbit around the sun. Thus, the declination varies from -23.45° at the winter solstice to 23.45° at the summer solstice. (The solstice terms are defined for the northern hemisphere.) The declination is zero at the two equinoxes. The declination, δ, as a function of the Julian day (J) is approximated by

$$\delta = 23.45^{\circ} \sin\left[\frac{360^{\circ}}{365}(J + 284)\right] \tag{13}$$

The declination as a function of Julian day is plotted in Figure 4. Note that the declination changes continuously throughout the day in the evaluation of the solar angles. Thus, J is allowed to take on non-integer values.

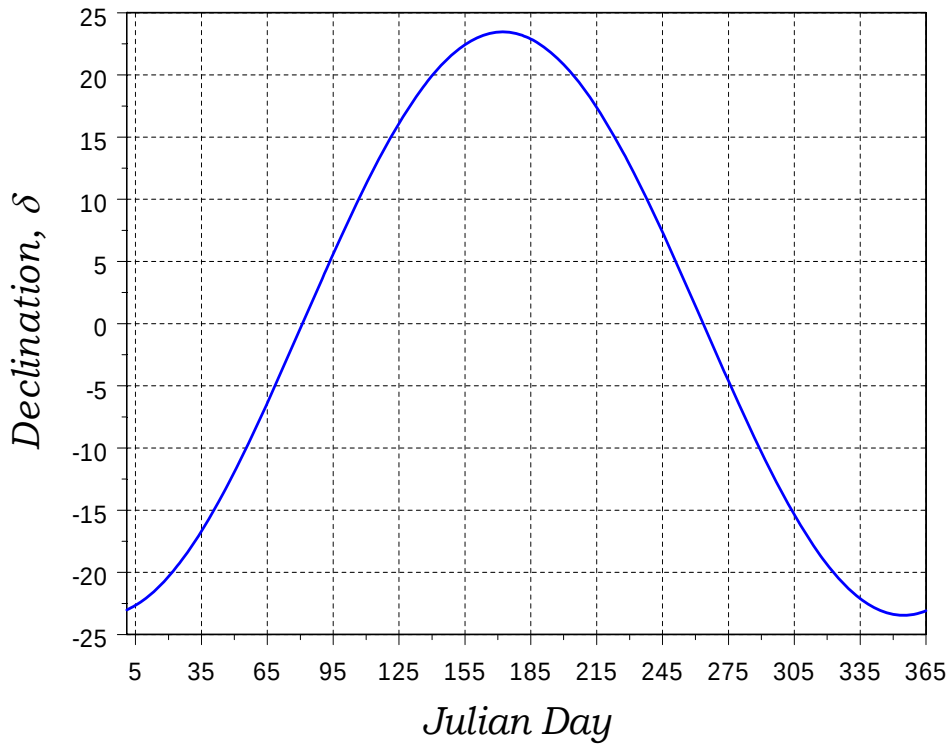

Figure 4. Declination as a Function of Julian Day

The hour angle is the angular distance that the earth rotates given a rotation rate of 15° per hour. The hour angle, ω, is defined as

$$\omega = 15^{\circ}\left(12 - t\right) \tag{14}$$

where t is the time in hours between 0 and 24. With this definition, the hour angle is positive before solar noon and negative after solar noon. Recall that solar noon is the time when the sun is at its maximum altitude, independent of time zone, daylight savings time, etc.

The solar altitude is given by

$$\sin\left(\theta_{alt}\right) = \cos\left(\lambda\right)\cos\left(\delta\right)\cos\left(\omega\right) + \sin\left(\lambda\right)\sin\left(\delta\right) \tag{15}$$

where λ is the latitude measured from the equator, with positive values north and negative values south. Rather than performing an arcsine evaluation to solve for θ_{alt} via Equation (15), an alternate equation for solar altitude is used to avoid difficulties when the sine of the angle is negative. The alternate equation is given as

$$\theta_{alt} = \arctan\left[\frac{\sin\left(\theta_{alt}\right)}{\cos\left(\theta_{alt}\right)}\right] \tag{16}$$

where the numerator term [$\sin(\theta_{alt})$] is calculated from Equation (15), but the denominator term [$\cos(\theta_{alt})$] is calculated using the following trigonometric relation

$$\cos\left(\theta_{alt}\right) = \left[1 - \sin^2\left(\theta_{alt}\right)\right]^{1/2} \tag{17}$$

It turns out that the solar altitude, θ_{alt}, is never needed in the formation of the sun vector, only the individual sine and cosine terms given by Equations (15) and (17), respectively.

The second angle needed to specify the sun vector is the solar azimuth, which is calculated as

$$\phi_{azi} = \arctan\left(\frac{x_{azi}}{y_{azi}}\right) \tag{18}$$

where x_{azi} is given by

$$x_{azi} = \sin(\omega)\cos(\delta) \tag{19}$$

and y_{azi} is given by

$$y_{azi} = \cos(\lambda)\sin(\delta) - \cos(\omega)\cos(\delta)\sin(\lambda) \tag{20}$$

Division by zero in Equation (18) is prevented by limiting the minimum value of y_{azi} to a small number (1.0E-26). The two solar angles are plotted in Figure 5, for Albuquerque, NM on June 21st. The two vertical green dashed lines indicate the time of sunrise and sunset. Note that the data points were plotted at ½-hr intervals such that the minimum and maximum values and the transitions are not exactly captured. The maximum solar altitude is 78.4°, which indicates that on the summer solstice, the sun is not directly overhead at solar noon. The solar azimuth is complicated as the sun passes from quadrant to quadrant of the global coordinate system (see Figure 3). The sun rises in the northeast quadrant at an azimuth of 60° measured from the positive *x* axis with clockwise defined as the positive direction. Thus, the sun rises 30° north of due east. The sun then passes into the southeast quadrant at approximately 8.4 hr at which time the azimuth is measured from the negative *x* axis. The azimuth equals zero at solar noon when the altitude is maximum. The sun continues through the southwest quadrant into the northwest quadrant, setting 30° north of due west. The trajectory of the sun is better illustrated in Figure 6, which shows the sun track on December 22nd and on June 21st. This plot was created with a commercial program[4] that was used to help validate the solar mechanics implementation.

---

[4] SunPlot3D, version 1.1, 2001, Maui Solar Energy Software Corporation.

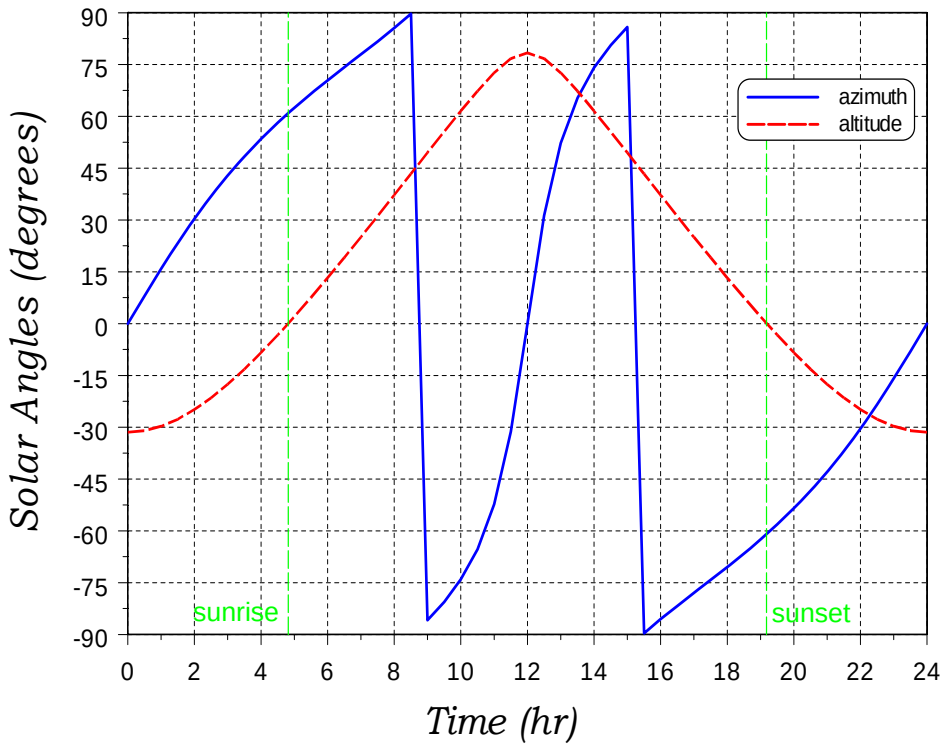

Figure 5. Solar Angles for Albuquerque, NM on June 21st

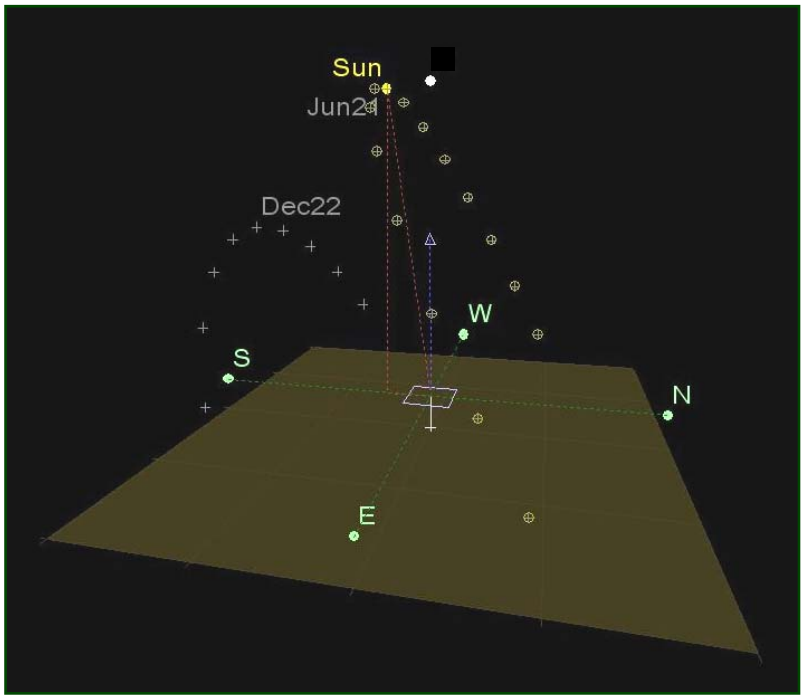

Figure 6. Trajectory of the Sun for Albuquerque, NM on June 21st

The Daylight Equations

The hour angle and the solar altitude equations are used to determine the duration of daylight, from which the time of sunrise and sunset are calculated as a function of day of the year and latitude. At sunrise, the solar altitude equals zero and Equation (15) provides the corresponding hour angle, $\omega_{sunrise}$. Thus

$$\omega_{sunrise} = \arccos\left[-\tan(\lambda)\tan(\delta)\right] \tag{21}$$

Dividing this angle (in units of degrees) by the earth's rotation rate of 15°/hr yields the elapsed time between sunrise and solar noon. The duration of daylight is then twice this time difference. Thus

$$\Delta t_{daylight} = \frac{2}{15^{\circ}}\omega_{sunrise} \tag{22}$$

In determining solar angles, declination is evaluated continuously as a function of time. However, in the evaluation of the daylight equations, declination is calculated only for integer values of the Julian day, *J*, yielding constant values for the duration of daylight and for the times of sunrise and sunset for each day.

For Polar latitudes above the Arctic Circle (greater than 66.568°) and below the Antarctic Circle (less than -66.561°), the sun never sets or rises for one or more days of the year. To account for this, the argument of the arccos function in Equation (21) must be restricted to the interval -1 to 1 (inclusive) to prevent undefined function evaluations. With this restriction, Equation (22) correctly calculates the duration of daylight as either 24 or 0 hours for the Polar latitudes for the appropriate days of the year.

For illustrative purposes, the duration of daylight for four latitudes (the first of which, 35° N, corresponds to Albuquerque, NM) is plotted in Figure 7. This figure shows that for Albuquerque, there are approximately 9.64 hr of daylight on December 21st and 14.37 hr of daylight on June 21st. Latitudes closer to the equator (e.g., 15° N) have less variation in daylight duration. The figure also includes results for the polar latitude of 75° N to demonstrate the situation in which 24-hours of daytime or 24-hours of nighttime prevail for many days of the year. Finally, a southern latitude (20° S) is included in the figure to illustrate the reversed seasons in which June 21st is the shortest day of the year. Of note is that during the two equinoxes, the duration of daylight is 12 hr for all latitudes.

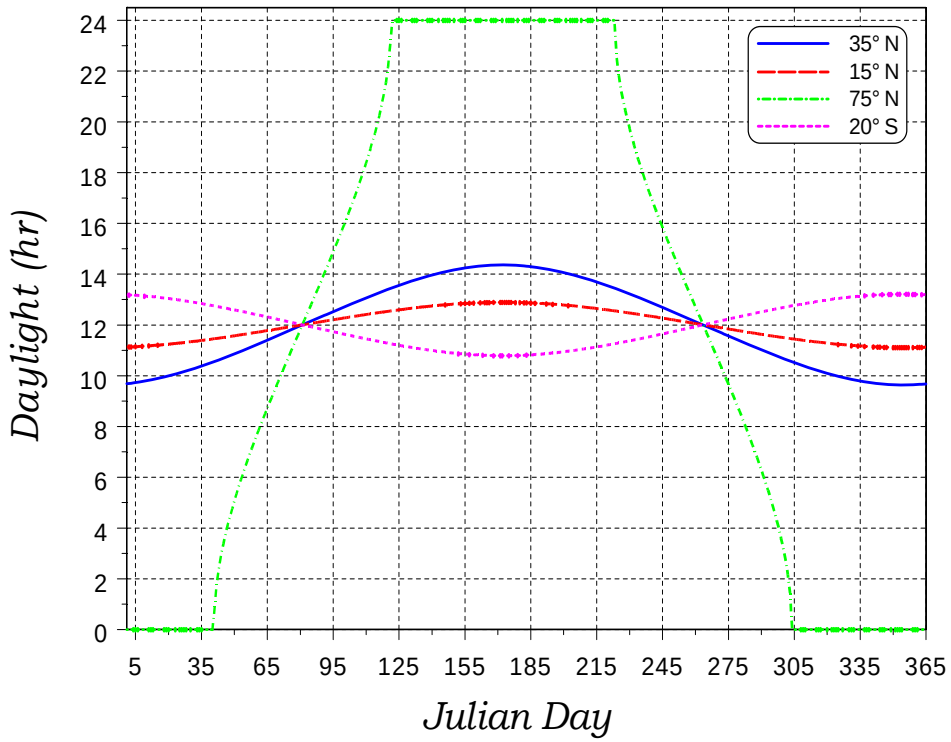

Figure 7. Duration of Daylight for Four Latitudes

Using the daylight equations, the times of sunrise and sunset relative to solar noon are then simply

$$t_{sunrise} = 12 - \frac{\Delta t_{daylight}}{2} \quad \text{and} \quad t_{sunset} = 12 + \frac{\Delta t_{daylight}}{2} \tag{23}$$

In the implementation of the solar mechanics equations, the solar flux is set to zero when the time is before sunrise or after sunset, i.e., nighttime. A 24-hr repeating clock is used in the subroutine implementation. Likewise, a 365-day repeating calendar is used for the Julian day.

Transformation to Direction Cosines

The two solar angles must now be transformed into direction cosines in a Cartesian coordinate system to enable execution of the vector dot product operation given by Equation (5). The global coordinate system, as was shown in Figure 3 (page 7), is based on the assumption that the positive x axis points due north and the positive z axis points due east, with the sun overhead in the positive y direction. Finite element models must be constructed to correspond to this coordinate system. The m direction cosine (j component) is determined as

$$m_s = \sin(\theta_{alt}) \tag{24}$$

The l (i component) direction cosine and n (k component) direction cosine for the sun vector depend on the coordinate-system quadrant as the sun passes overhead. If the azimuth is less than zero and the time is past solar noon, or if the azimuth is greater than zero and the time is before solar noon, then the sun is in the northern sky and the direction cosines are

$$l_s = \cos\left(\theta_{alt}\right)\cos\left(\phi_{azi}\right) \tag{25}$$

$$n_s = \cos\left(\theta_{alt}\right)\sin\left(\phi_{azi}\right) \tag{26}$$

where the s subscript refers to the sun and the direction cosine nomenclature is consistent with that presented in Equations (7) and (8). When the sun is in the southern sky, the direction cosines are

$$l_s = -\cos\left(\theta_{alt}\right)\cos\left(\phi_{azi}\right) \tag{27}$$

$$n_s = -\cos\left(\theta_{alt}\right)\sin\left(\phi_{azi}\right) \tag{28}$$

An exception occurs when the time exactly equals solar noon (i.e., 12 hr). In this case

$$l_s = s_y \cos\left(\theta_{alt}\right) \quad \text{with } s_y \text{ determined as the sign of } y_{azi} \tag{29}$$

$$n_s = 0 \tag{30}$$

The value of y_{azi} was determined from Equation (20). A negative value of l_s occurs when the sun is in the southern sky.

The unit sun vector, pointing towards the sun, is now given as

$$\mathbf{S} = l_s\hat{i} + m_s\hat{j} + n_s\hat{k} \tag{31}$$

Figure 8 provides a plot of the three direction cosines for Albuquerque, NM on June 21st.

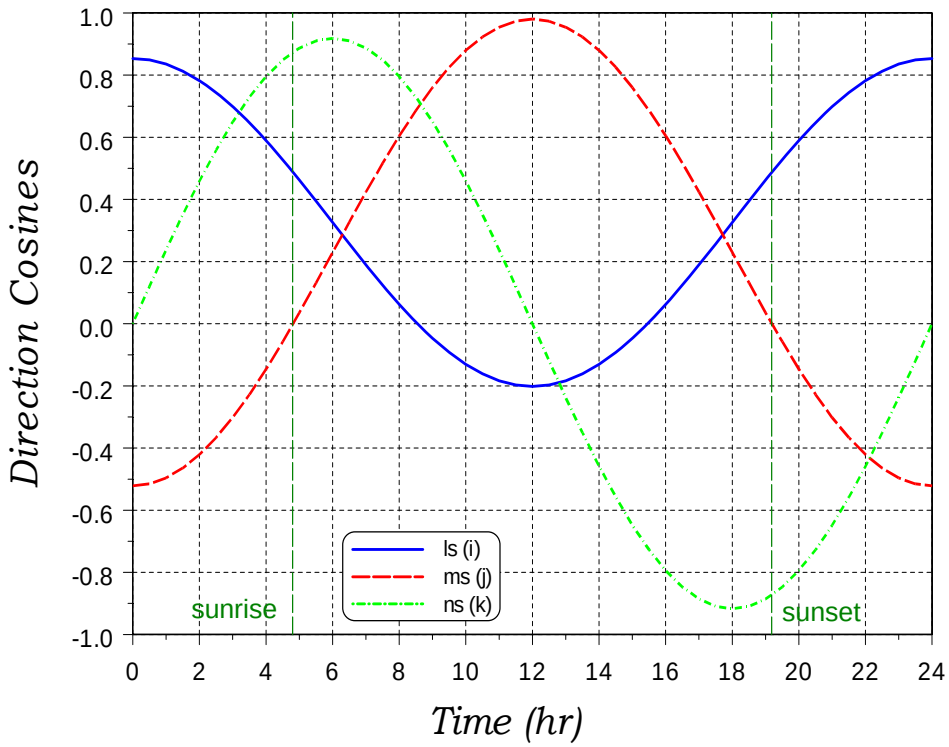

Figure 8. Direction Cosines for Albuquerque, NM on June 21st

With the components of both the **F** and **S** vectors determined, the value of f_{or} can be calculated for every exposed finite element face [Equation (5)]. A negative value of f_{or} indicates that the surface is not exposed directly to the sun and the direct component of solar flux is set to zero. With this vector algebra approach, the direct solar flux on every face is automatically determined without the need for the analyst to specify the orientation of each face.

Shadow Factor

The final factor from Equation (1) to be determined is f_{sh}, the shadow factor. In general, every element face has the potential to cast a shadow on every other element face in the model at any daylight time. Thus, the search for shadowing is a computationally intensive operation. In addition, the search for shadowed surfaces must be performed throughout daylight hours as the sun traverses the sky, adding significant computational burden. Sophisticated shadowing algorithms are available in the computer graphics and gaming communities, but the expense of their implementation was considered beyond currently available resources. Therefore, a simplified approach was implemented in which only potential shadowing structures are identified by the analyst. Taking advantage of the analyst's knowledge of the geometry greatly reduces the computational burden associated with the determination of shadowing. This simplified approach also has the advantage

of allowing the specification of "pseudo" shadowing structures that are not part of the finite element model. This is desirable if the thermal response of a structure is not of interest but the shadow it casts influences the thermal response of other structures.

A potential shadowing structure, referred to as a wall, is defined by specifying the coordinates of the top two corner locations of the wall, with the simplifying assumption that the wall extends to the lowest point of the structure. An image of an example finite element model created to demonstrate shadowing is provided in Figure 9. The top two corners of the blue wall are labeled points 1 and 2. This is an example of a "thin" structure in which specification of a single wall is adequate to capture its shadowing effect. In specifying the coordinates of the two points, it is important to select points just inside the structure to prevent shadowing of the actual finite element structure by the pseudo wall. If the structure is not thin, specification of multiple pseudo walls would be required. For example, a cube would require specification of four pseudo walls, one for each side. To capture the shadows cast by a curved surface, such as the red cylinder, specification of two perpendicular walls passing through the center (as indicated by the two cyan lines) works quite well. Thus, complex structures can be accommodated with the creative selection of multiple pseudo walls.

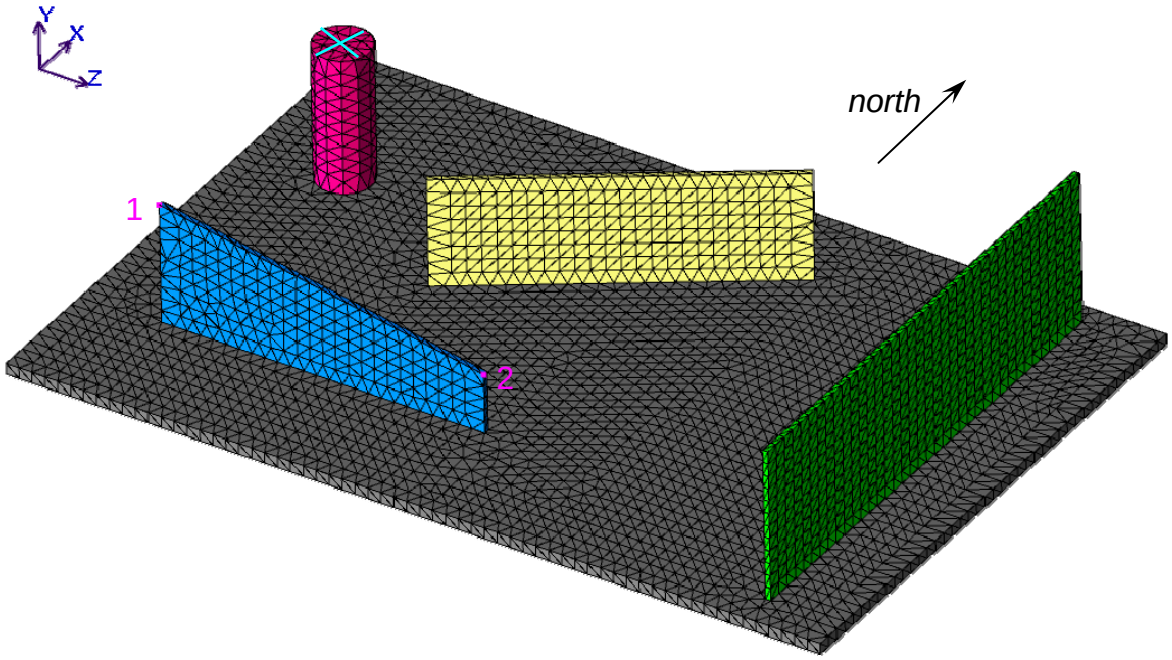

Figure 9. Example Finite Element Model for Demonstration of Shadowing

Each face on the surface of the finite element model is considered in the determination of the shadow factors. The coordinates of the face centroid are determined as the algebraic average of the integration point coordinates. For additional resolution, the shadow factor for each integration point could be

determined separately. However, this adds significant computational burden; therefore, the centroid approach was adopted for simplicity. Using the current specification of the sun vector (i.e., solar direction cosines), a straight line is constructed on the *x*-*z* plane passing through the face centroid location. Recall from Figure 3 that the positive *x* axis is defined as north and the positive *z* axis is defined as east in this solar-mechanics implementation. In terms of the solar direction cosines, the slope of this sun line is

$$s_s = \frac{l_s}{n_s} \tag{32}$$

and the intercept is determined by one of the following equations depending on the choice of abscissa and ordinate axes selected for the wall line, the derivation of which is described next.

$$b_s = x_f - s_s z_f \tag{33}$$

or

$$b_s = z_f - \frac{1}{s_s} x_f \tag{34}$$

where b_s is the sun line intercept, x_f and z_f are the *x* and *z* coordinates of the face centroid, respectively, and s_s is the sun line slope. Equation (33) is used if the *x* axis is the ordinate. Figure 10 provides a sample reference diagram to accompany the following derivation of the wall line equations.

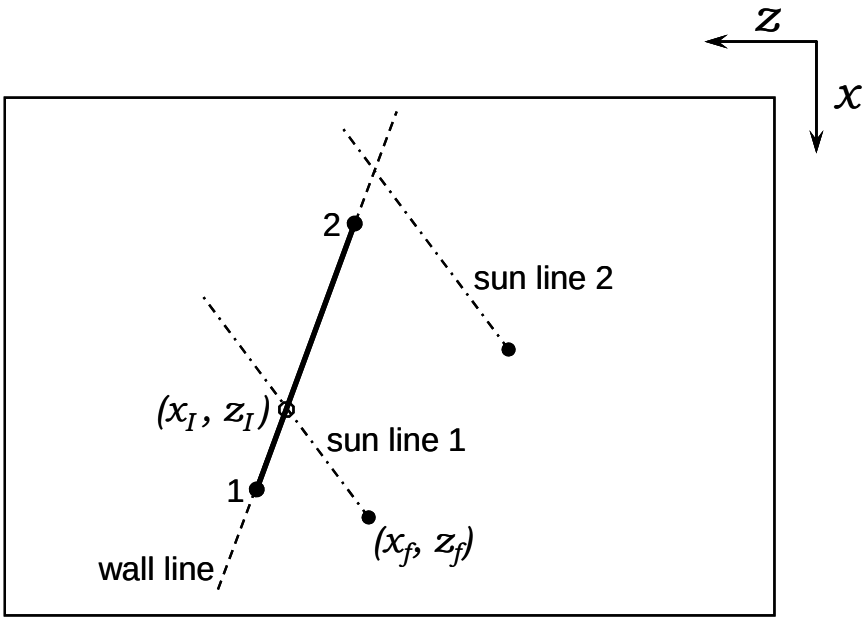

Figure 10. Sample Diagram for Derivation of Shadow Equations

For every defined pseudo wall, a wall line in the *x*-*z* plane is determined based on the coordinates of the two specified upper corner points. First, however, based on the orientation of the wall, selection of which axis is to be the abscissa and which the ordinate is made. The *x* and *z* distances between the two wall corner points are given as

$$\Delta x = x_1 - x_2 \quad \text{and} \quad \Delta z = z_1 - z_2 \tag{35}$$

If $|\Delta z|$ is greater than $|\Delta x|$, then the *x* axis is the ordinate and the *z* axis is the abscissa. Otherwise, the *z* axis is the ordinate and the *x* axis is the abscissa. For $|\Delta z|$ greater than $|\Delta x|$, the slope of the wall line is given as

$$s_w = \frac{\Delta x}{\Delta z} \tag{36}$$

and the intercept is

$$b_w = x_1 - s_w z_1 \tag{37}$$

where the *w* subscript refers to the wall. The coordinates of the intersection of the sun line and the wall line are then given by the next two equations.

$$z_I = \frac{b_s - b_w}{s_w - s_s} \tag{38}$$

$$x_I = s_w z_I + b_w \tag{39}$$

For $|\Delta x|$ greater than or equal to $|\Delta z|$, the slope of the wall line is given as

$$s_w = \frac{\Delta z}{\Delta x} \tag{40}$$

and the intercept is

$$b_w = z_1 - s_w x_1 \tag{41}$$

The coordinates of the intersection of the sun line and the wall line are then given by the next two equations.

$$x_I = \frac{b_s - b_w}{s_w - \dfrac{1}{s_s}} \tag{42}$$

$$z_I = s_w x_I + b_w \tag{43}$$

where the *I* subscript refers to the intersection point. If the intersection point lies between the bounds of the two wall corner points, then shadowing of the element face by this wall is possible. Figure 10 provides two notional sun

lines corresponding to two different face centroids. For sun line 1, the intersection lies between the bounds whereas it does not for sun line 2. For sun line 1, it is still necessary to determine if the face is shadowed depending on the height of the wall at the intersection location.

In this shadowing implementation, the two corner points of a pseudo wall are assumed to be connected with a straight line. If the top contour of the wall is curved, it would be necessary to split the wall into multiple walls to approximate the curvature. The height of the wall at the wall intersection point is given by

$$h = y_2 \frac{L_1}{L_1 + L_2} + y_1 \left(1 - \frac{L_1}{L_1 + L_2} \right) \tag{44}$$

where

$$L_1 = \left[\left(x_1 - x_I \right)^2 + \left(z_1 - z_I \right)^2 \right]^{\frac{1}{2}} \tag{45}$$

and

$$L_2 = \left[\left(x_2 - x_I \right)^2 + \left(z_2 - z_I \right)^2 \right]^{\frac{1}{2}} \tag{46}$$

Next, the height at which the sun line intersects the wall is determined based on the direction cosine corresponding to the solar altitude, as depicted in Figure 11. h_I must be less than h for shadowing to occur.

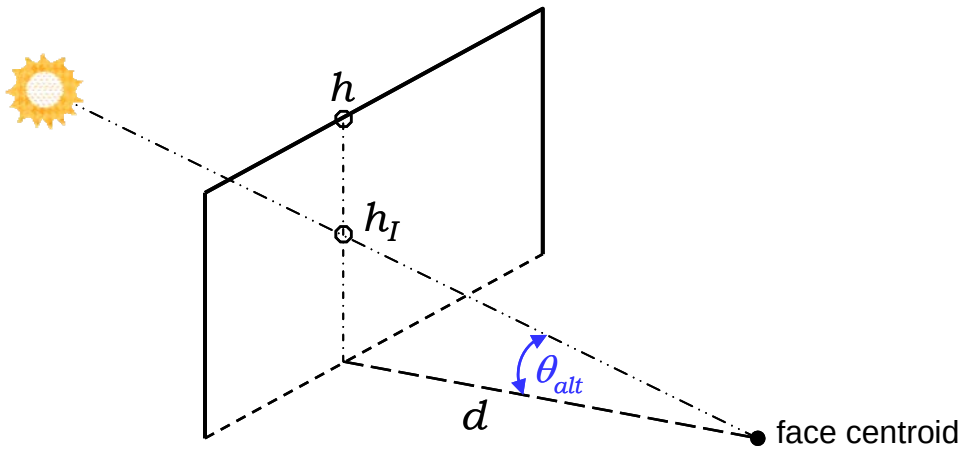

Figure 11. Diagram of Sun Line Intersection with a Wall

The intersection height is given by

$$h_I = dm_s + y_f \tag{47}$$

where m_s was defined in Equation (24) and d is given by

$$d = \left[\left(x_I - x_f \right)^2 + \left(z_I - z_f \right)^2 \right]^{\frac{1}{2}} \tag{48}$$

and y_f is the y coordinate of the element face. Before shadowing can be confirmed, it is also necessary to consider the position of the face relative to the wall and the time of day. If the face is east of the wall and the time is past solar noon, then the face is shadowed and the shadow factor, f_{sh}, is set to zero. Likewise, if the face is west of the wall and the time is before solar noon, the face is shadowed. The shadow factor is set to unity if shadowing is not detected. Once a face is determined to be shadowed, there is no need to check for shadowing by additional walls.

Reflected Component of Total Incident Solar Flux

As mentioned at the beginning of the Governing Equations Section (page 2), the reflected component of incident solar flux is not included because of limitations in the user-subroutine capability within Calore. However, a formulation for the reflected flux component is offered here to support potential future solar mechanics implementation in other heat transfer codes.

The total incident solar flux on a surface is the sum of direct, diffuse, and reflected flux components, expressed as

$$q''_{inc} = q''_{dir} + q''_{dif} + q''_{refl} = \underbrace{q''_{am1} f_{ext} \left(f_{or} f_{clear} f_{sh} \right)}_{direct} + \underbrace{q''_{am1} f_{ext} f_{dif}}_{diffuse} + q''_{refl} \tag{49}$$

where q''_{dir} and q''_{dif} are the direct and diffuse components [both extracted from Equation (1)], and q''_{refl} is the reflected component of solar flux. It should be noted that the reflected component does not provide an additional independent source of solar flux. Instead, it provides an additional flux contribution associated with incident solar flux that is <u>not</u> absorbed by other surfaces or reflected into the sky. Thus, the reflected solar flux for each element face <u>is a function of</u> the direct and diffuse fluxes on all other faces. For surfaces that reflect diffusely (i.e., no specular reflections*), the reflected flux for each element face can be determined by first solving for the total incident flux using

$$q''_{inc,i} = \left(q''_{dir} + q''_{dif} \right)_i + \underbrace{\sum_{j=1}^{N} \left(q''_{dir} + q''_{dif} \right)_j \rho_j F_{j-i}}_{reflected} \tag{50}$$

* A similar (although much more complex) formulation can be derived for structures that include specular surfaces, but is not addressed here.

where the *i* and *j* subscripts indicate face number, *N* is the total number of finite element faces, ρ_j is the solar reflectivity of face *j*, (one minus the solar absorptivity, α) and F_{j-i} is the view factor from surface *j* to surface *i*. Such an equation for every face in the model is constructed with the direct and diffuse flux components first determined using the equations presented previously. The system of equations given by Equation (50) is then solved simultaneously to yield the total incident flux. If desired, the reflected component of flux can be backed out based on Equation (49). Thus, the same view factors for the structure partial radiation enclosure temperature solution, as discussed in the next subsection, are used to determine the reflected flux component.

Diurnal Air and Sky Temperatures

With the absorbed solar flux on each element face calculated based on the latitude and the time of the year, it is necessary to consistently specify the diurnal air and sky temperatures. That is, the diurnal variations in air and sky temperatures must reflect the changing length of day throughout the year. Air temperature, which is needed for the specification of convective heat transfer boundary conditions via Newton's law of cooling, reflects local weather conditions. Maximum and minimum daily air temperatures are readily available for most cities and towns in the world. One source of such data can be found at www.weatherbase.com. A screen capture of such data for Albuquerque, NM is provided in Figure 12.

Albuquerque, New Mexico

Elevation: 1623 meters Latitude: 35 03N Longitude: 106 37W

Average Temperature — Years on Record: 48

	YEAR	Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.
°C	13	1	5	8	13	18	23	26	25	21	14	7	2

Average High Temperature — Years on Record: 48

	YEAR	Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.
°C	21	8	11	16	21	26	32	33	31	28	22	13	8

Average Low Temperature — Years on Record: 48

	YEAR	Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.
°C	6	-5	-2	---	5	10	15	18	17	13	6	---	-4

Highest Recorded Temperature — Years on Record: 48

	YEAR	Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.
°C	41	20	24	29	31	36	41	40	38	37	32	25	22

Lowest Recorded Temperature — Years on Record: 48

	YEAR	Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.
°C	-27	-27	-20	-13	-7	-2	4	11	10	2	-6	-21	-21

Figure 12. High and Low Air Temperatures for Albuquerque, NM

The average high and low temperature data for each month of the year are used to fit cosine distributions given by

$$T_{high} = \bar{T}_{high} + A_{high} \cos\left[\frac{360^\circ}{365}\left(J + O_{high}\right)\right] \tag{51}$$

$$T_{low} = \bar{T}_{low} + A_{low} \cos\left[\frac{360^\circ}{365}\left(J + O_{low}\right)\right] \tag{52}$$

with the average high and low temperatures, and the amplitudes (A) for the high- and low-temperature data given by

$$\bar{T}_{high} = \frac{T_{high}^{\max} + T_{high}^{\min}}{2} \tag{53}$$

$$A_{high} = \frac{T_{high}^{\max} - T_{high}^{\min}}{2} \tag{54}$$

$$\bar{T}_{low} = \frac{T_{low}^{\max} + T_{low}^{\min}}{2} \tag{55}$$

$$A_{low} = \frac{T_{low}^{\max} - T_{low}^{\min}}{2} \tag{56}$$

where J is the Julian day [as was first defined in Equation (2)], and O_{high} and O_{low} are offsets based on the day of the year in which the maximum high and maximum low temperatures occur, respectively. Thus

$$O_{high} = 365 - J_{high}^{\max} \tag{57}$$

$$O_{low} = 365 - J_{low}^{\max} \tag{58}$$

For Albuquerque, $J_{high}^{\max}$ is approximately 192 days, and $J_{low}^{\max}$ is approximately 195 days. (The offset is analogous to the phase angle in electrical engineering and serves to shift the cosine curve to the left or to the right.) The weather data, along with the fitted cosine distributions, is plotted in Figure 13, which indicates that the cosine functions provide a reasonable approximation of the data. This was found to be the case for about a dozen other cities across the world, providing partial validation of this approximate approach.

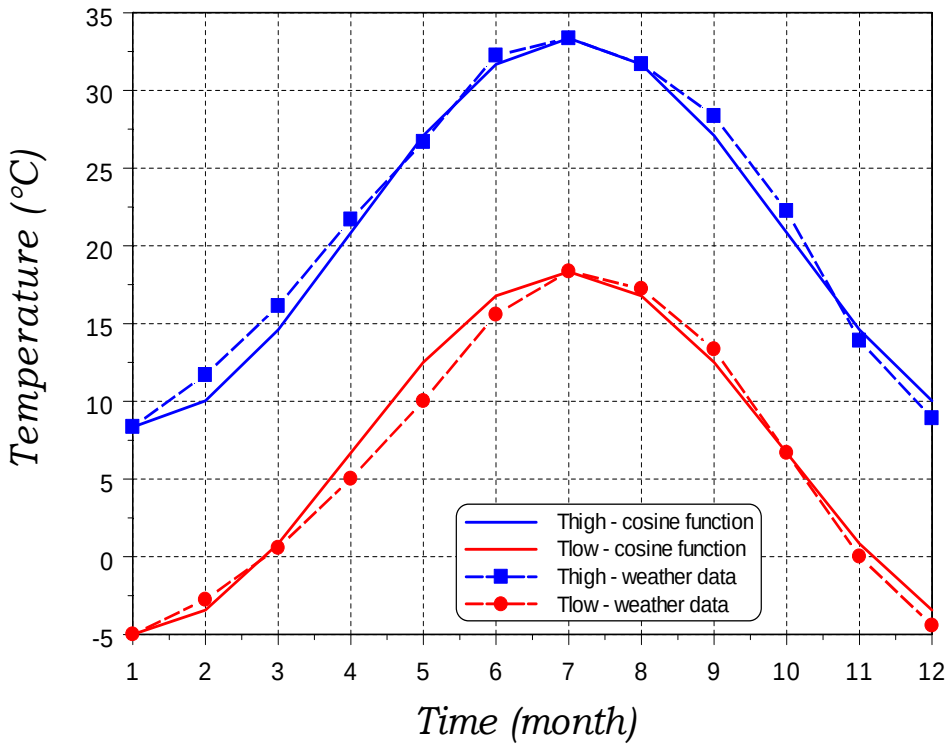

Figure 13. Yearly Air Temperature Variations for Albuquerque, NM

With the yearly variation defined by the fitted cosine distributions, the diurnal variation for any given day is determined as

$$T_{air} = \bar{T}_{air} + A\cos\left[\frac{360^{\circ}}{24}(t+O)\right] \tag{59}$$

where $\bar{T}_{air}$ is the average air temperature given by

$$\bar{T}_{air} = \frac{T_{high} + T_{low}}{2} \tag{60}$$

and the amplitude, A, is given by

$$A = \frac{T_{high} - T_{low}}{2} \tag{61}$$

where the high- and low-temperature values are determined using Equations (51) and (52), respectively, evaluated at the current value of J, t is time in units of hours, and O is the offset determined as

$$O = \bar{O} + \tau\cos\left[\frac{360^{\circ}}{24}(t+\bar{O})\right] \tag{62}$$

where $\bar{O}$ is the average offset given by

$$\bar{O} = \frac{\Delta t_{daylight}}{2} \tag{63}$$

where $\Delta t_{daylight}$ is the number of hours of daylight calculated using Equation (22), which is a function of latitude, τ is the lag time in units of hours, and t is time also in units of hours. The lag time accounts for the fact that often the minimum temperature of the day occurs after sunrise and the maximum temperature occurs before sunset. A typical value of τ is about 1 hr, but can vary between about ½ hr to 2 hr. If τ is set to zero, then O is constant and the temperature cosine curve is not skewed. Otherwise, the offset varies with a cosine distribution, skewing the diurnal temperature curve. It should be noted that the presence of heat sinks/sources can alter the local air temperature variations—no attempt is made to account for this effect.

The sky temperature is also needed for thermal simulations involving solar mechanics. The sky temperature is used for far-field radiative boundary conditions and represents an effective temperature, accounting for the fact that the air is not perfectly transparent to infrared radiation. If the air were transparent, the sky temperature would be approximately 3 K, which is the background temperature of the universe. Water vapor, ozone, carbon dioxide, other trace elements, and particulate matter associated with air pollution contribute to the absorption of radiation through the atmosphere. There are numerous empirical relations that allow the determination of sky temperature as a function of various parameters such as air temperature and absolute humidity. One such relation for sky temperature[5] is given by

$$T_{sky} = \left[T_{air}^4 \left(1 - 0.261 e^{-7.77*10^{-4}(273 - T_{air})^2} \right) \right]^{\frac{1}{4}} \tag{64}$$

where the air temperature must be specified in the absolute temperature units of Kelvin. This relation is a function only of air temperature but provides reasonable agreement with data across the globe. For the Calore implementation, the sky temperature is specified as the far-field reference temperature in a partial radiation enclosure boundary condition based on a gray-body approach. For such a boundary condition, view factors are calculated for all element faces, including the associated view factor to the environment, represented as a black surface at the reference (i.e., sky) temperature. Thus, the fact that some element faces may have only a partial view of the sky is accounted for in the implementation. The view factors for the partial radiation enclosure can also be used to determine the reflected component of solar flux, as was described in the previous subsection.

---

[5] Sherwood B. Inso and Ray D. Jackson, "Thermal Radiation from the Atmosphere," Journal of Geophysical Research, Vol. 74, No. 23, October 20, 1969.

It should be noted that for a partial radiation enclosure, the specified surface emissivities are those for infrared radiation (IR), corresponding to thermal radiation. Recall that surface absorptivity, α, is specified for wavelengths associated with solar radiation. In a gray-body radiation enclosure implementation, such as that used in the Calore thermal response code, absorptivity and emissivity are assumed equal. However, the solar absorptivity and the IR emissivity often are not equal for some surfaces. The solar mechanics approach implemented here allows separate specification of solar absorptivity such that radiation transfer at the two wavelengths is appropriately accounted for.

The air temperature (assuming a lag time of 1 hr) and the associated sky temperature are plotted in Figure 14 for Albuquerque, NM on June 21st (J = 172). On this day, with solar noon defined as 12 hr, sunrise occurs at 4.82 hr (4:49 AM) and sunset occurs at 19.18 hr (7:11 PM). It should be noted that calculation of sunrise and sunset is strongly dependent on the accuracy of the declination equation [Equation (13)], which is an approximation for ease of implementation. The sunrise/sunset times are consistent with values from a variety of other sources that indicate variations of several minutes. Introduction of the lag term results in the skewed cosine function distribution. An air-temperature curve is also included in which the lag time was set to zero for comparison to the skewed distribution. Obviously, the actual temperature variations throughout the day will not be as smooth as provided by these well-behaved functions for air and sky temperature.

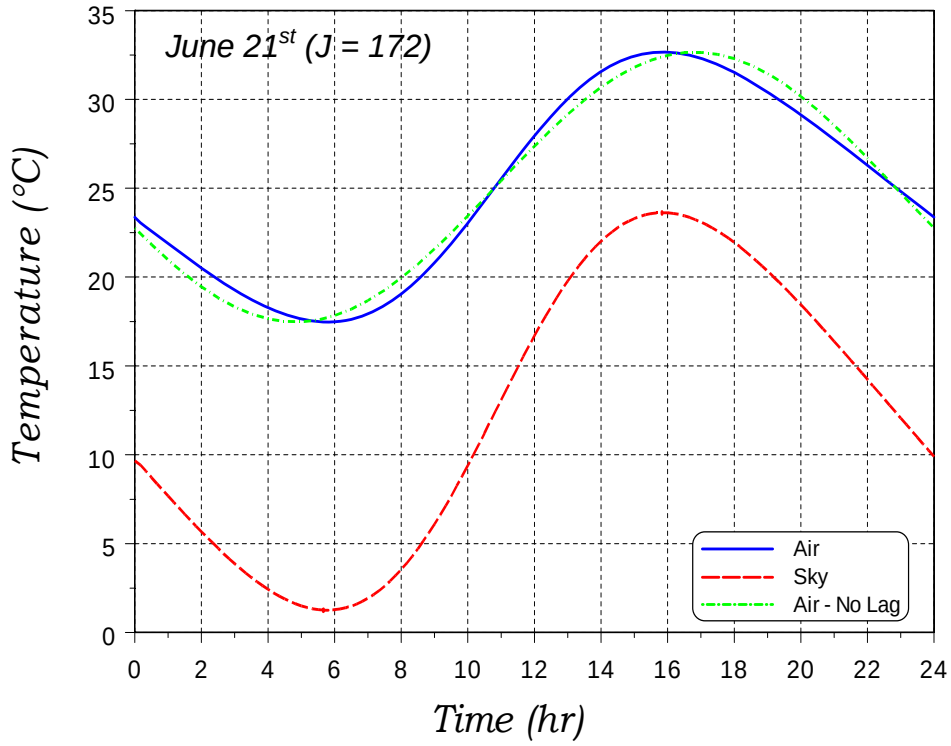

Figure 14. Diurnal Air and Sky Temperatures for Albuquerque, NM

Figure 15 compares the smooth temperature curve associated with the simulation for Albuquerque, NM, to measured hourly temperature data on June 21st for two years, 1990 and 2008. The temperature data was found at www.wunderground.com/history/, which is one of several such databases. The comparison indicates that the simulation approach provides a reasonable reproduction of the data, approximately capturing the lag time effect. The hourly measured data obviously reflects changing local conditions for the particular day that are not captured with the smooth cosine distribution. Also, the yearly temperature data is based on measurements averaged over many years. Therefore, one can only expect the minimum and maximum temperatures for any particular day to be representative of the actual. The premise of the simulation approach is to provide time varying air temperature consistent with the solar mechanics implementation, enabling meaningful comparisons of thermal response at different times of the year and at different geographical locations. An advantage of a general implementation of the solar mechanics equations is that an analyst would not be restricted to using the cosine distribution to specify air temperature, and could instead use tabular data if desired.

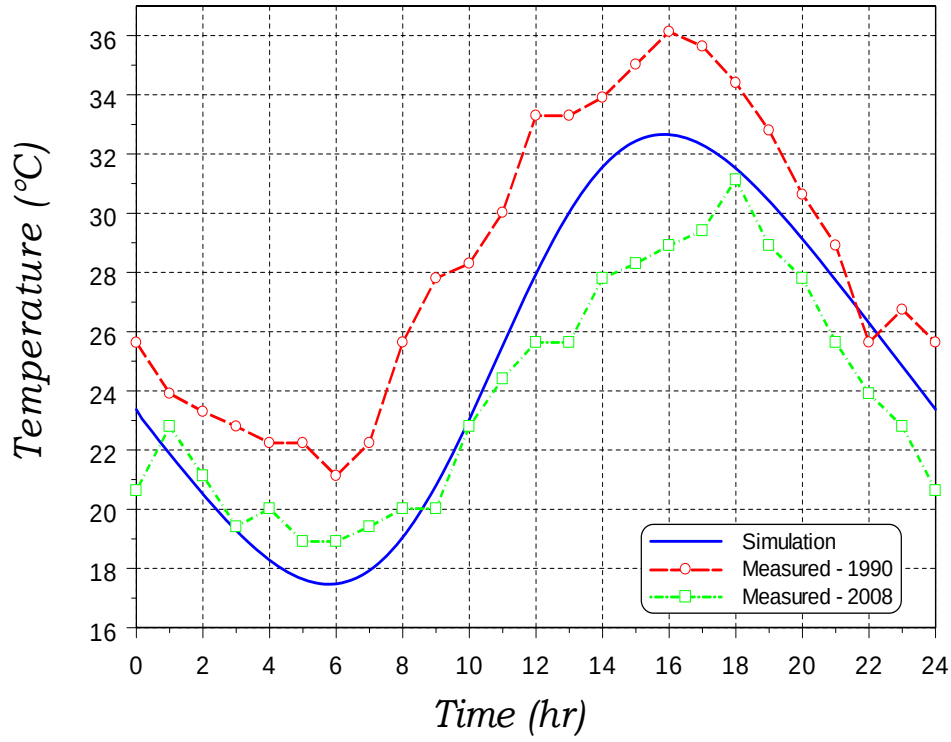

Figure 15. Comparison to Hourly Temperature Measurements

Solar Mechanics Example Problems

Solar mechanics results for a few example problems are provided for demonstration purposes. The examples include solar flux as a function of day of year and latitude. In addition, an example for solar flux incident on vertical and horizontal surfaces is provided along with comparison to measured values on a horizontal surface. An example problem is also included to demonstrate shadowing.

Figure 16 provides the calculated solar flux incident on a horizontal plate in Albuquerque, NM, which is at a latitude of 35° north. The figure demonstrates the variation in peak flux occurring on four different days throughout the year. The summer solstice occurs on day 172, the winter solstice occurs on day 355, and the two equinoxes occur on days 81 and 265. The differences in peak fluxes and daylight time are clearly demonstrated.

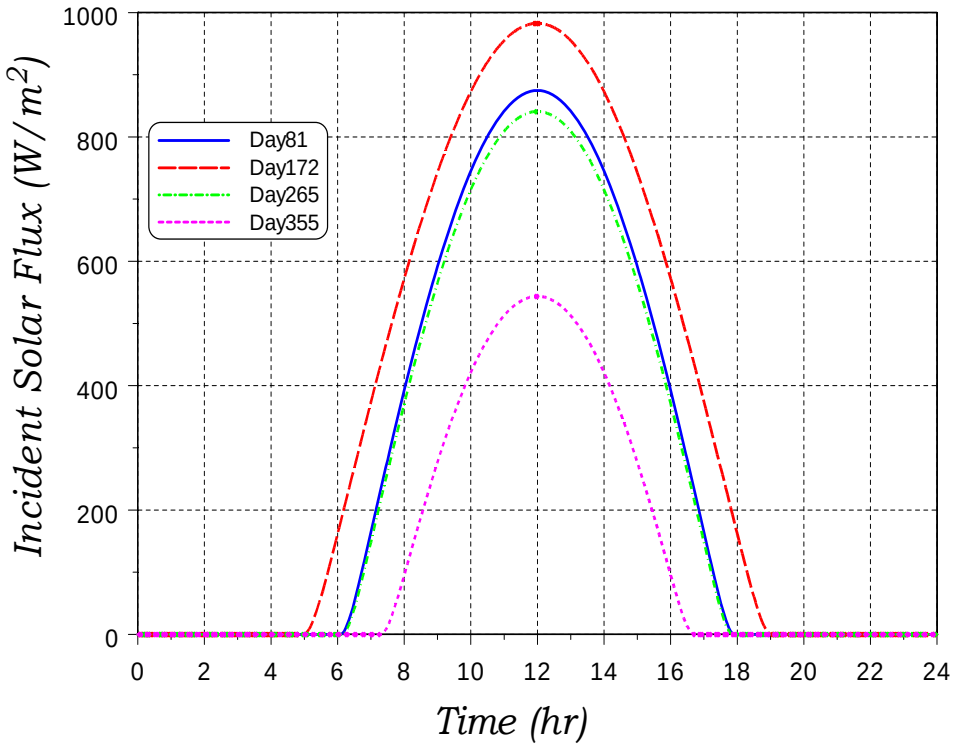

Figure 16. Incident Solar Flux, Various Days, Albuquerque, NM (Latitude=35° N)

Figure 17 provides the incident solar flux on June 21st for cities at four different latitudes. Death Valley, CA, which is the farthest north location, exhibits the longest day, and Singapore, which is near the equator, exhibits the shortest day.

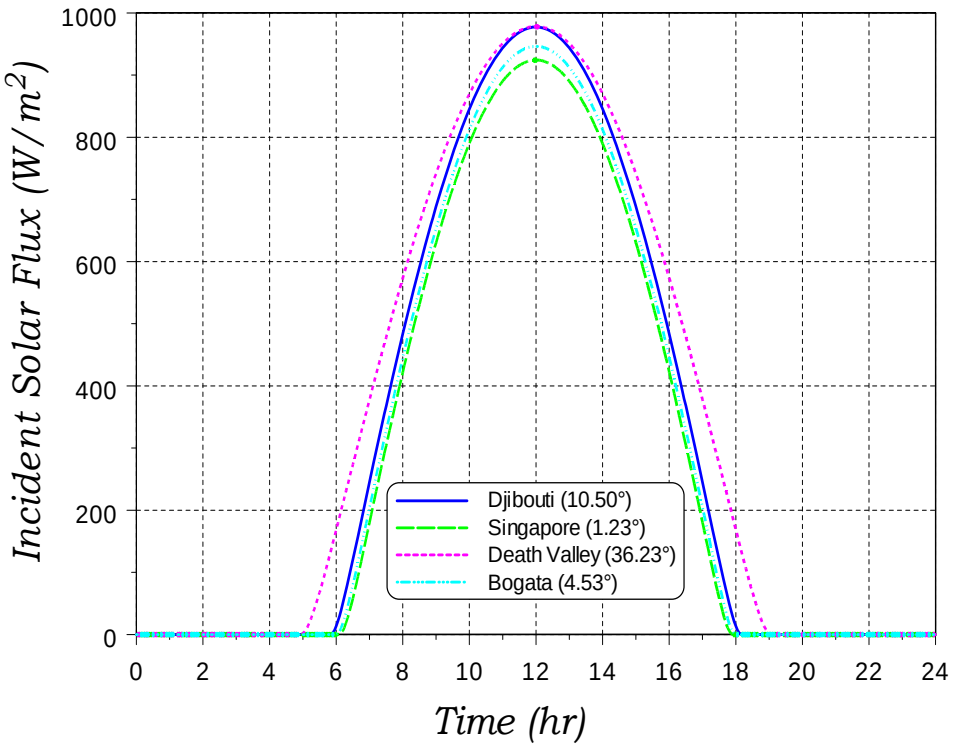

Figure 17. Incident Solar Flux, Various Latitudes, June 21st

For Albuquerque, NM, on June 21st, the incident solar flux on both sides of a horizontal plate and on both sides of a vertical plate is provided in Figure 18. For this example, the two plates are considered standalone such that reflections or shadowing influences are not present. The calculated flux includes both direct and diffuse components. For the bottom surface of the horizontal plate, there is no direct component, only the diffuse component. Recall that the diffuse flux is a result of scattering in the atmosphere, which is considered to be isotropic. Thus, all surfaces, regardless of orientation, receive the diffuse component of solar flux. For the back face of the vertical plate (i.e., west facing), there is no direct component for times before solar noon. Likewise, for the front face, there is no direct component for times after solar noon. The peak total flux on the two sides of the vertical plate is not as great as the peak total flux on the top surface of the horizontal plate because of greater atmospheric attenuation early and late in the day when the optical path length through the atmosphere is greater than it is at noon.

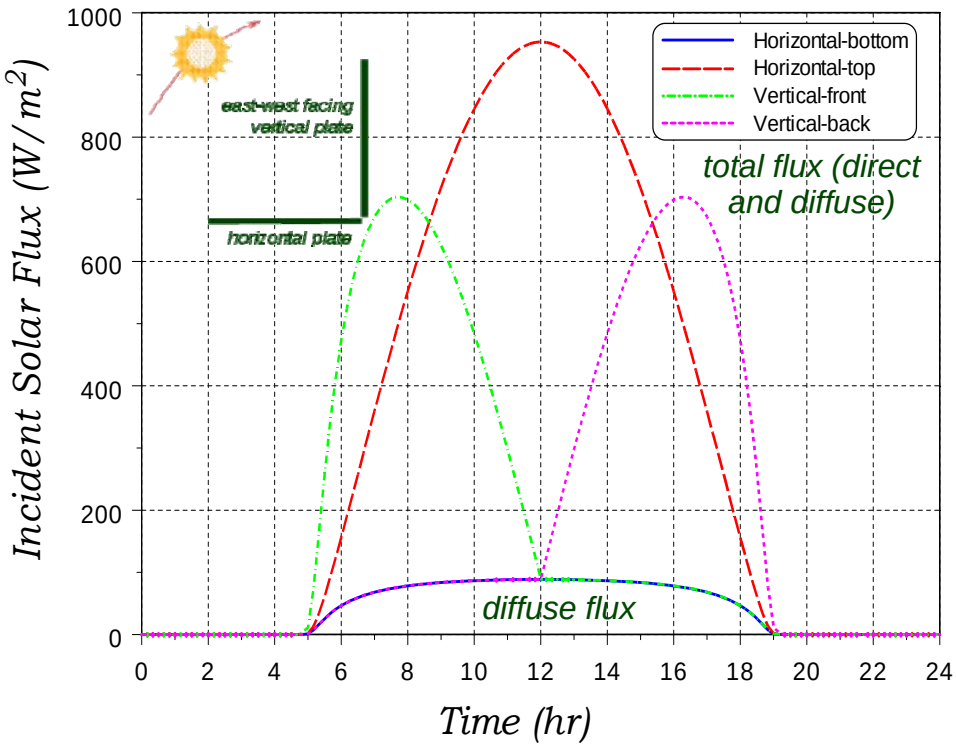

Figure 18. Incident Solar Flux, Albuquerque, NM, June 21st

The calculated total and diffuse incident solar fluxes on a horizontal surface for Albuquerque, NM are compared to the corresponding measured flux values. The flux data, for June 21st of the year 2005, was found at http://rredc.nrel.gov/solar/old_data/nsrdb/. For the simulation, the extinction factor was assumed equal to 0.2; the diffuse factor was assumed equal to 0.1; and the clearness factor was set to unity. Better agreement could be achieved if desired by modifying these input parameters. The comparison shows that the measured diffuse component is larger than calculated, indicating that a larger value of the diffuse factor may be warranted. Also, the calculated flux during sunrise and sunset does not quite match the data, indicating a discrepancy in the simplified model. A more sophisticated model for extinction would be needed to capture this response, although this may not be warranted because of the very low relative flux contributions at these times. Otherwise, the agreement between the simulation and the data is reasonable for this single-day comparison. As with the temperature model, solar mechanics was implemented with the major premise that the model provides solar flux predictions that can be used to consistently compare thermal response simulations at different times of the year at different geographical locations. The available input constants allow the user to modify the results to reflect specific local atmospheric conditions if desired.

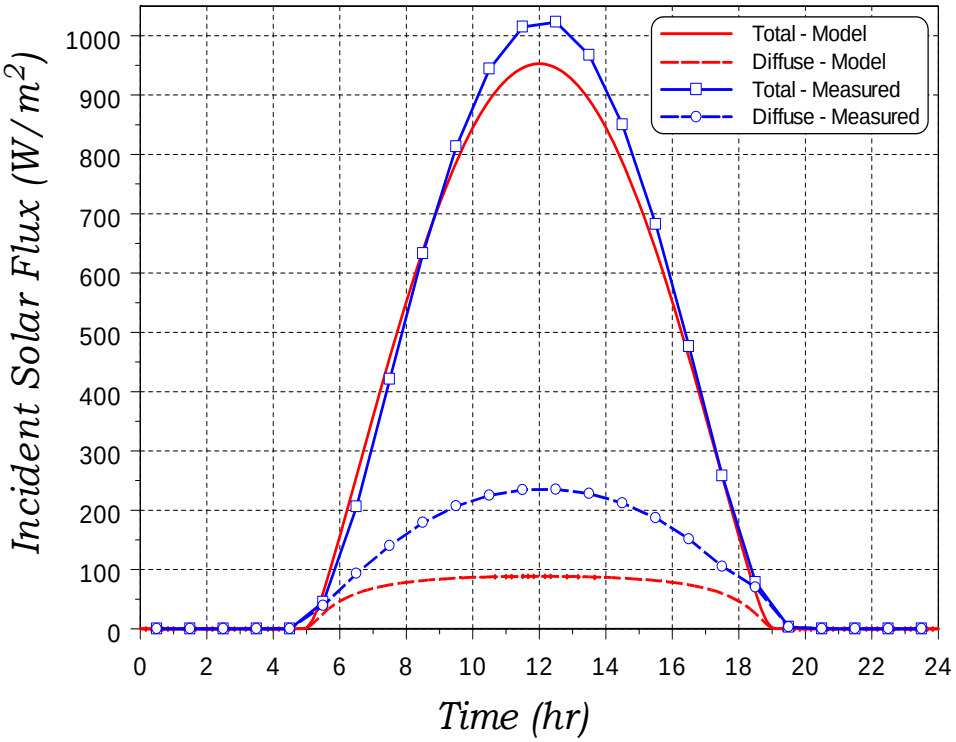

Figure 19. Comparison to Hourly Solar Flux Measurements

The example finite element model presented in Figure 9 (page 15) was used to demonstrate the influence of shadows on calculated absorbed solar flux. For simplicity, the structure was assumed to be all steel with a solar absorptivity of 0.7 and an infrared emissivity of 0.2. Recall that for the Calore implementation, the positive z axis points due east. The calculation is for Albuquerque, NM at a latitude of approximately 35° N, and begins on June 21st. Six images at different times of the day are provided in Figure 20. The varying flux level on the curved surface of the cylinder is readily apparent. The figure also shows the changing position of the shadows on the multi-wall structure. Ragged edges on some of the shadows are a result of the relative coarseness of the mesh. The temperature distribution on the structure, as provided in Figure 21 at 13 and 18 hours, is smoother but still reflects the remnants of the shadows at the end of the day.

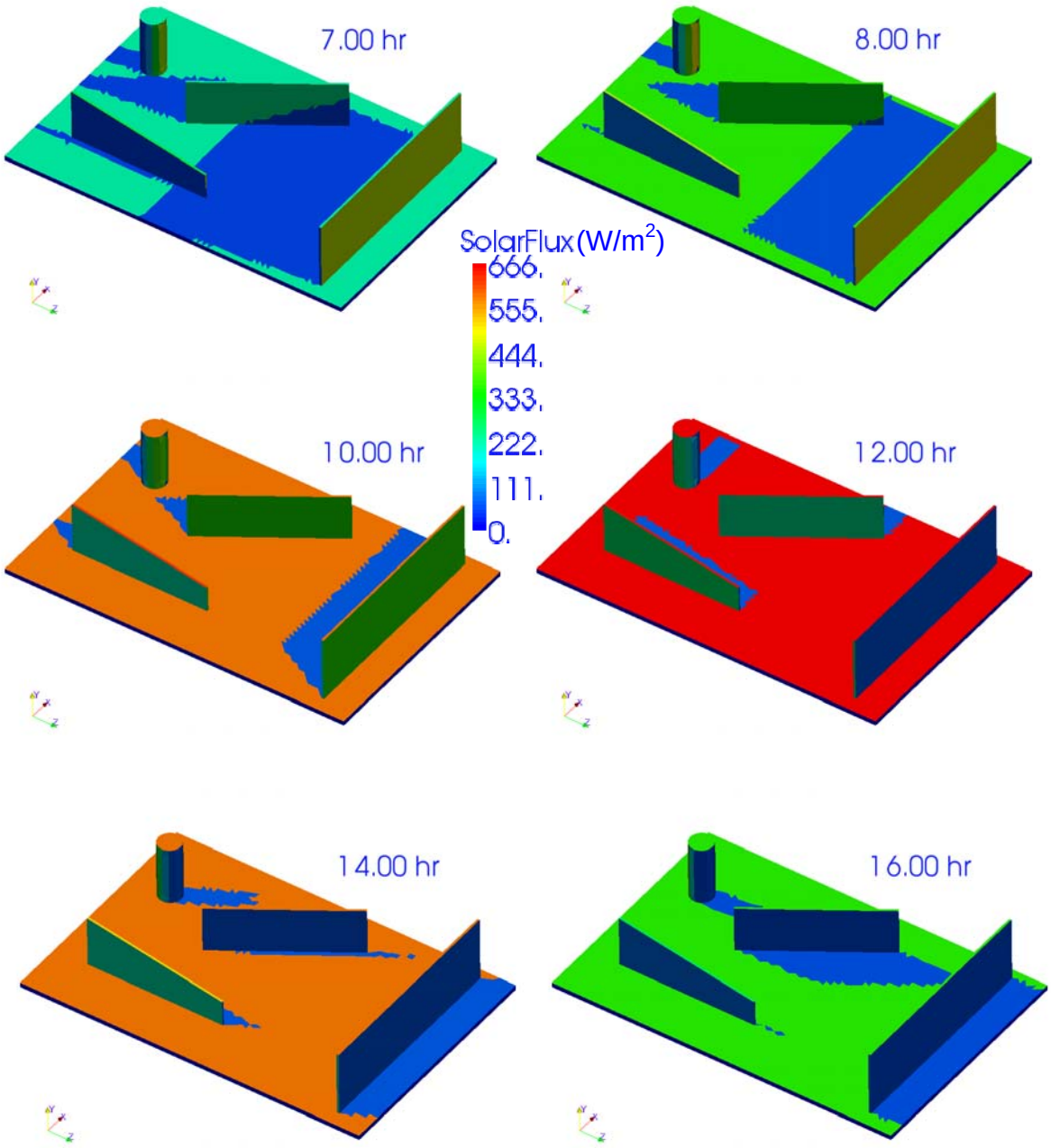

Figure 20. Absorbed Solar Flux Example Problem Images

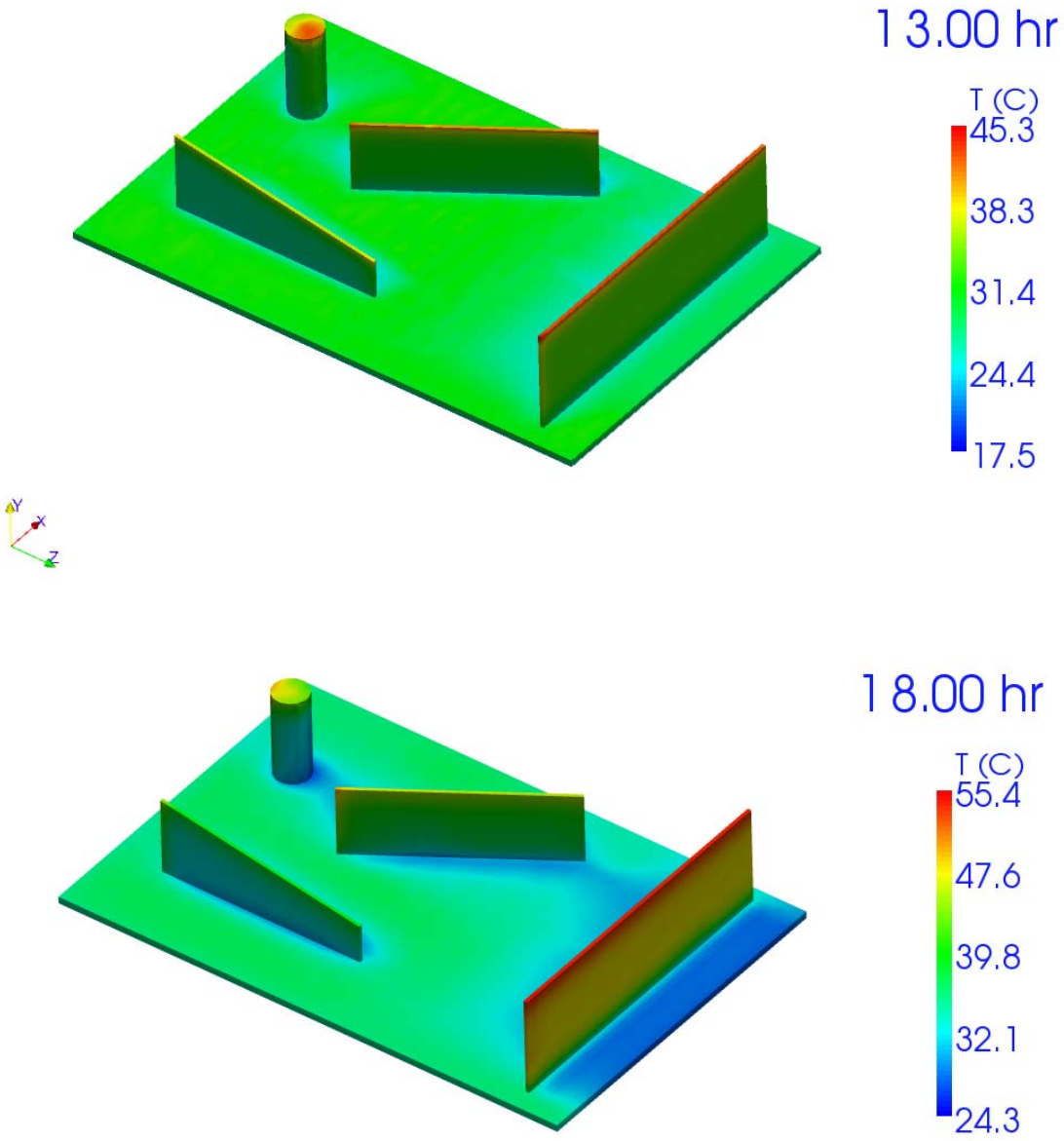

Figure 21. Temperature Distributions for Example Problem

Comments and Summary

An approach has been developed to enable simulation of the thermal response of structures in which incident solar heat flux is an important thermal load. This approach was implemented into the three-dimensional Calore thermal response code via a user subroutine for solar mechanics, along with subroutines for the diurnal variation of air temperature and effective sky temperature.

The selected solar mechanics approach is an adaptation of one of several similar approaches found in the solar literature. A basic premise of the approach is that the normal clear day solar loading is of primary interest, although analyst-specified factors can be modified to approximate non-clear day weather conditions. The different approaches found in the literature all seem to yield similar solar mechanics results. However, it should be noted that the results should not be expected to match identically because of differences in approximations used to develop the governing equations. Using commercial and internet-based software, limited validation of the approach with regard to peak solar flux, solar trajectories, and times of sunrise and sunset indicate reasonable agreement. In addition, most of the governing equations were implemented within an Excel spreadsheet, available from the author upon request, before implementation into the FORTRAN user subroutines. However, limited resources prevented thorough validation of the approach. No attempt was made to validate the shadowing capability, other than verifying that the predicted shadows were reasonable and consistent with the solar trajectories.

There are three parameters in the solar mechanics approach related to atmospheric conditions that must be specified by the analyst: the extinction coefficient (c_{ext}), the diffuse factor (f_{dif}), and the clearness factor (f_{clear}). The extinction coefficient is representative of the absorption of solar radiation whereas the diffuse factor is representative of scattering. It is expected that the two parameters are correlated, although no such correlation was available in the original approach reference. The clearness factor is essentially a correction factor on the direct component of flux as implemented in the adapted approach. Again, no correlation with the other atmospheric parameters is readily available. Further research regarding these parameters and their implementation is warranted.

Three features of note in the developed approach are (1) the use of vector algebra to automatically determine the orientation factor, accounting for the orientation of each finite element face relative to the position of the sun, (2) the capability to account for shadows cast by “pseudo” walls, and (3) integration of the solar mechanics capability with the capability to specify diurnal variations in air temperature (relying on published weather data) and the related effective sky temperature as functions of latitude and day of the year, consistent with the solar mechanics implementation.

The approach is sufficiently developed for implementation into other thermal response codes if desired. In addition to formal validation, other future development efforts could address a more general and sophisticated shadowing capability, inclusion of a reflected flux contribution, and additional options for calculation of the effective sky temperature (several empirical correlations are readily available in the literature). Because of restrictions in the user-subroutine capability, only a single value of solar absorptivity can be specified for each implementation of a solar boundary condition. The ability to specify solar absorptivity as a material or surface property is desirable in a general implementation. Restrictions in the user-subroutine capability also prevented inclusion of the reflected flux contribution, which also is desirable in a general implementation. The developed approach, as currently implemented via Calore user subroutines, provides a useful capability for engineering problems involving solar loading.

References

1) "Calore—A Computational Heat Transfer Program," SAND2008-0098P, February 12, 2008.

2) ASHRAE Fundamentals Handbook (SI), 1997.

3) W. M. Rohsenow, J. P. Hartnett, and E. N. Ganic, **Handbook of Heat Transfer Applications**, 2nd Edition, McGraw-Hill, Inc., 1973.

4) SunPlot3D, version 1.1, 2001, Maui Solar Energy Software Corporation.

5) Sherwood B. Inso and Ray D. Jackson, "Thermal Radiation from the Atmosphere," Journal of Geophysical Research, Vol. 74, No. 23, October 20, 1969.

Appendix: Calore User Subroutines for Solar Mechanics

For ease of readability, angles in all of the equations presented in the text were expressed in units of degrees. However, in the FORTRAN implementation presented in this appendix, all operations are performed with angles in units of radians.

Solar Mechanics:

```
c  ======================================================================
c
      subroutine Solar_flux_shadows(faceID, nelemf, nip, coords, t,
     * flux, ierror)
c
c  absorbed Solar flux on an object oriented arbitrarily with respect to the sun.
c   the flux is adjusted as the sun rises and sets based on the direction cosines.
c   the assumed coordinate system is based on the sun traveling in the y-z plane with
c   sunrise and sunset at y = 0, and the sun is somewhere overhead for y > 0.
c   thus, the positive z axis points east and the positive x axis points north.
c   (SolarFlux is written as a user face variable for subsequent output.)
c
c  this subroutine accounts for shadows with an approximate method in which the
c   user must specify the coordinates of the corners of shadowing pseudo walls.
c   currently, only the top location of the wall is specified, assuming that the
c   wall extends down to the ground (y=0 or some common elevation).
c
c   created by Dean Dobranich, 3/2009
c
      implicit none
c
      integer ierror, nip, nelemf
      integer faceID(nelemf)
      double precision flux(nelemf, nip)
      double precision coords(3, nelemf, nip)
      double precision t(nelemf, nip)
c
c     ierror: user defined error code for calore to test. zero means success.
c     nip :  number of gauss integration points per face
c     nelemf:  number of element faces in workset
c     flux:  array of length nelemf containing the values of the
c           heat flux that the subroutine calculates at
c           integration points of element faces
c     coords: coords of the integration points on each element face.
c     t:  array of temperatures at the integration points on each element face
c
      integer i, j, nr, found
      double precision time, time_24hr, pi, pio2, zero, one
      double precision t_sunrise, t_sunset, solar_flux, absorpt
      double precision daylight, gamma, beta, f
      double precision xa, xb, xc, ya, yb, yc, za, zb, zc
      double precision v1i, v1j, v1k
      double precision v2i, v2j, v2k
      double precision v3i, v3j, v3k, v3m
      double precision vsi, vsj, vsk, vsm
      double precision JulianDay, latitude, declination
      double precision hour_angle_sunrise, hour_angle
```

```
      double precision surface_flux, fext, fdif, fclear, ext_coef
      double precision shadow_dt1, shadow_dt2
c
      integer k, ni, nwalls, m, n, mo, ko
      double precision slope_solar, intercept_solar
      double precision slope_wall, intercept_wall
      double precision dx1, dx2, dz, dx
      double precision xe, ye, ze, x, z
      double precision l1, l2, wf, h, d, hi
c     double precision solar_alt, solar_azi, rotation_rate, arg
      double precision solar_azi, rotation_rate, arg
      double precision cos_ha, cos_lat, sin_lat, cos_dec, sin_dec
      double precision sin_alt, cos_alt, x_azi, y_azi
c
      double precision small
c
      parameter (pi=3.14159265358979323846264433d0, pio2=0.5d0*pi)
      parameter (zero=0.0d0, one=1.0d0, small=1.0d-26)
      parameter (rotation_rate=15.0d0*pi/180.0d0)

c  nr is number of real data variables read in from .i file, ni is number of integers
      integer max_num_walls
      parameter (max_num_walls=8)
      parameter (nr=8+max_num_walls*6, ni=1)
      double precision rdata(nr)
      integer idata(ni)
      double precision wall_coords(max_num_walls,2,3)
      logical possible
c
      ierror = 0
c
c   idata(1) => number of walls that may shadow a structure
c
c   rdata(1) => Julian day (fractional day permissible, i.e., 3.5 indicates half way
c               through the third day of the year; progresses with simulation time)
c   rdata(2) => latitude (+ for northern hemisphere)
c   rdata(3) => solar absorptivity of surface
c   rdata(4) => ext_coef (extinction coefficient through atmosphere, typically = 0.2)
c   rdata(5) => fdif (fraction of solar flux that is diffuse, typically = 0.1)
c   rdata(6) => fclear (clearness factor, direct solar flux component multiplier;
c               = 1.0 for clear day, 0.8 for 20% cloud cover, etc.)
c   rdata(7) => shadow_dt1 (time when structure is in shadow relative to sunrise)
c   rdata(8) => shadow_dt2 (time when structure is in shadow relative to sunset)
c               thus, if time is less that sunrise + shadow_dt1 or greater than
c               sunset - shadow_dt2, then only diffuse flux is applied
c   rdata(9-m) => x,y,z coordinates of walls 1 through max_num_walls with 2 coordinates per wall
c                 where m = 8 + max_num_walls*6
c
c
c  get current simulation time and convert to hours
      call acal_get_time(time)
      time = time/3600.0d0
c  use a 24 hour repeating clock
      time_24hr = mod(time,24.0d0)
c
      call acal_get_instance_int_data(ni,idata)
```

```
      nwalls = idata(1)
c
      call acal_get_instance_real_data(nr,rdata)
      JulianDay = mod(rdata(1) + time/24.0d0, 365.0d0)
      latitude = rdata(2)*pi/180.0d0
      absorpt = rdata(3)
      ext_coef = rdata(4)
      fdif = rdata(5)
      fclear = rdata(6)
      shadow_dt1 = rdata(7)
      shadow_dt2 = rdata(8)

c  rdata must be entered in input file for all max_num_walls, but only nwalls assigned and used
c   m is counter for points 1 and 2 of wall; n is pointer for x, y, z (1, 2, 3) axis
      ko = 0
      do k = 1, nwalls
        mo = 0
        do m = 1, 2
          do n = 1, 3
            wall_coords(k,m,n) = rdata(8+ko+mo+n)
          enddo
          mo = mo + 3
        enddo
        ko = ko + 6
      enddo
c
c  declination (radians) [23.45 degrees = 0.40927971 radians]
c   truncate Julian Day for calculation of sunrise time
      declination = 0.40927971d0*dsin(pi/182.5d0*
     &             (dint(JulianDay)+284.0d0))
      arg = -dtan(latitude)*dtan(declination)
      arg = max(arg,-one)
      arg = min(arg,one)
c  argument of arccos must be between -1 and 1, inclusive (bounds would be
c   exceeded for polar angles when daylight may equal 0 or 24 hours)
      hour_angle_sunrise = dacos(arg)
      daylight = 24.0d0/pi*hour_angle_sunrise
      t_sunrise = 12.0d0 - daylight*0.5d0
      t_sunset = t_sunrise + daylight
c
c  recalculate declination for determining solar angles
      declination = 0.40927971d0*dsin(pi/182.5d0*(JulianDay+284.0d0))
c
c  if time >= t_sunset or time < t_sunrise, set flux to zero (night time)
      if (time_24hr .ge. t_sunset .or. time_24hr .lt. t_sunrise) then

        do i = 1, nelemf
          do j = 1, nip
            flux(i,j) = zero
          end do
          call acal_put_real_face_var(zero,faceID(i),9,
     &                                "SolarFlux",found)
        end do

      else
```

```
c  solar_flux initially is the flux on a surface normal to the sun for m = 1 (1 air mass)
       solar_flux = 1160.0d0 + 74.0d0 * dsin(pi/182.5d0 *
     &              (JulianDay + 88.0d0))
c
c  determine position of sun in sky based on hour_angle, latitude, and declination
       hour_angle = rotation_rate * (12.0d0 - time_24hr)
       cos_ha = dcos(hour_angle)
       cos_lat = dcos(latitude)
       sin_lat = dsin(latitude)
       cos_dec = dcos(declination)
       sin_dec = dsin(declination)
       sin_alt = cos_lat*cos_dec*cos_ha + sin_lat*sin_dec
       cos_alt = dsqrt(1.0d0 - sin_alt**2)
c  find the solar altitude angle (varies from 0 up to a max of 90 and back to 0)
c   solar altitude is never used directly so comment out unless desired to print
c        solar_alt = datan(sin_alt/max(small,cos_alt))
c
c  find the solar azimuthal angle (varies between -90 and +90 degrees)
       x_azi = dsin(hour_angle)*cos_dec
       y_azi = cos_lat*sin_dec - cos_ha*cos_dec*sin_lat
       y_azi = dsign(max(dabs(y_azi),small),y_azi)
       solar_azi = datan(x_azi/y_azi)
c
c  create a unit vector representing the direction of the sun
c   a negative value of vsi indicates that the sun is in the southern sky
       vsj = sin_alt
c  x-z plane coordinates depend on time of day and sign of azimuthal angle
       if (time_24hr .eq. 12.0d0) then
         vsi = dsign(cos_alt,y_azi)
         vsk = zero
       elseif (solar_azi*(time_24hr - 12.0d0) .lt. zero) then
c  sun is in the northern sky
         vsi = cos_alt*dcos(solar_azi)
         vsk = cos_alt*dsin(solar_azi)
       else
c  sun is in the southern sky
         vsi = -cos_alt*dcos(solar_azi)
         vsk = -cos_alt*dsin(solar_azi)
       endif
c
c  determine atmospheric extinction factor (minimum horizon angle of 2 degrees (0.0349 radians))
c   an atmosphere mass of 1 corresponds to when the sun is at its zenith
c   the inverse sine of the solar altitude angle accounts for the greater
c   distance through the atmosphere at lower angles
       fext = dexp(-ext_coef/max(dabs(sin_alt),0.0349d0))
c
c  apply aborptivity and account for extinction from atmosphere
       solar_flux = solar_flux * absorpt * fext
c
c ---
c  check for shadow time (simple approximate way to account for blocking structures)
       if (time_24hr .gt. t_sunset - shadow_dt2 .or.
     &     time_24hr .lt. t_sunrise + shadow_dt1) then
c
c  only diffuse component is applied when shadowed
         surface_flux = solar_flux * fdif
```

```
c
        do i = 1, nelemf
          do j = 1, nip
            flux(i,j) = surface_flux
          end do
          call acal_put_real_face_var(surface_flux,faceID(i),9,
     &                                "SolarFlux",found)
        end do
c
        return
       endif
c ---
c
c  determine direct and diffuse flux levels for all element faces
       do i = 1, nelemf
c
c  determine two vectors on element surface using the integration point coordinates
         xa = coords(1, i, 1)
         ya = coords(2, i, 1)
         za = coords(3, i, 1)
         xb = coords(1, i, 2)
         yb = coords(2, i, 2)
         zb = coords(3, i, 2)
         xc = coords(1, i, 3)
         yc = coords(2, i, 3)
         zc = coords(3, i, 3)

         v1i = xb - xa
         v1j = yb - ya
         v1k = zb - za

         v2i = xc - xa
         v2j = yc - ya
         v2k = zc - za

c  vector 3 is normal to vectors 1 and 2 that lie on the surface of the element face,
c   found by the cross product
         v3i = v1j*v2k - v1k*v2j
         v3j = v1k*v2i - v1i*v2k
         v3k = v1i*v2j - v1j*v2i

c  vector magnitude
         v3m = dsqrt(v3i**2 + v3j**2 + v3k**2)

c  unit vector (direction cosine components)
         v3i = v3i/v3m
         v3j = v3j/v3m
         v3k = v3k/v3m

c  now find the dot product of the sun vector, vs, with the surface normal vector, v3
c   both vs and v3 are unit vectors, so the dot product is the fraction of solar flux on the surface

         f = v3i*vsi + v3j*vsj + v3k*vsk

c  a negative f indicates the surface of the element face is not visible; thus set direct flux to zero
         if (f .gt. zero) then
```

```
c  check for shadowing by any pseudo walls (if shadowed, set f = 0.0)
c   find element centroid coordinates
          if (nwalls .gt. 0) then
            slope_solar = vsi/dsign(max(dabs(vsk),small),vsk)
            xe = coords(1, i, 1)
            ye = coords(2, i, 1)
            ze = coords(3, i, 1)
            do j = 2, nip
              xe = xe + coords(1, i, j)
              ye = ye + coords(2, i, j)
              ze = ze + coords(3, i, j)
            end do
            xe = xe/nip
            ye = ye/nip
            ze = ze/nip
          endif
c
          do k = 1, nwalls
c
c  perform a first check to eliminate element faces that are not shadowed
c   note that a negative value of vsi indicates that the sun is in the southern sky
c
c  find x distances from element face centroid to each of the two defining wall points
c                         wall#, point#, x-y-z (1-2-3)
             dx1 = xe - wall_coords(k,1,1)
             dx2 = xe - wall_coords(k,2,1)
c  if sun is in the northern sky and the element face is north of wall, no shadows are possible
             if (dx1 .gt. zero .and. dx2 .gt. zero
     &          .and. vsi .gt. zero) go to 5
c  if sun is in the southern sky and the element face is south of wall, no shadows are possible
             if (dx1 .lt. zero .and. dx2 .lt. zero
     &          .and. vsi .lt. zero) go to 5
c
c  now check to see if face is possibly shadowed based on its x-z location and the sun vector
c
c  determine equation of wall line in the x-z plane (this could be pulled outside of face do loop)
c   also find equation of the solar line in the x-z plane passing through the face centroid
c   then find x, z coordinates of intersection of the two lines
             dz = wall_coords(k,1,3) - wall_coords(k,2,3)
             dx = wall_coords(k,1,1) - wall_coords(k,2,1)
             possible = .false.
             if (dabs(dz) .gt. dabs(dx)) then
               slope_wall = dx/dz
               intercept_wall = wall_coords(k,1,1) -
     &                          slope_wall*wall_coords(k,1,3)
               intercept_solar = xe - slope_solar*ze
               z = (intercept_solar - intercept_wall)/
     &              (slope_wall - slope_solar)
               x = slope_wall*z + intercept_wall
c  if z intersection lies between bounds of the wall line, then shadowing is possible
               if (z.lt.max(wall_coords(k,1,3),wall_coords(k,2,3))
     &           .and. z.gt.min(wall_coords(k,1,3),wall_coords(k,2,3)))
     &           possible = .true.
             else
               slope_wall = dz/dx
```

```
            intercept_wall = wall_coords(k,1,3) -
     &                       slope_wall*wall_coords(k,1,1)
            intercept_solar = ze - 1.0d0/slope_solar*xe
            x = (intercept_solar - intercept_wall)/
     &           (slope_wall - 1.0d0/slope_solar)
            z = slope_wall*x + intercept_wall
c  if x intersection lies between bounds of the wall line, then shadowing is possible
            if (x.lt.max(wall_coords(k,1,1),wall_coords(k,2,1))
     &       .and. x.gt.min(wall_coords(k,1,1),wall_coords(k,2,1)))
     &       possible = .true.
          endif
c
          if (possible) then
c
c  determine height of wall at intersection of sun vector with the wall
c  once an element face is found to be shadowed, no need to check additional walls, exit do (go to 10)
c
            l1 = dsqrt((wall_coords(k,1,1) - x)**2 +
     &           (wall_coords(k,1,3) - z)**2)
            l2 = dsqrt((wall_coords(k,2,1) - x)**2 +
     &           (wall_coords(k,2,3) - z)**2)
            wf = l1/(l1+l2)
            h = wf*wall_coords(k,2,2) +
     &          (1.0d0 - wf)*wall_coords(k,1,2)
c
c  find height of wall at intersection with solar line assuming common bottom location
            d = dsqrt((x - xe)**2 + (z - ze)**2)
            hi = d * vsj + ye
            if (hi .lt. h) then
c   for times before solar noon:
              if (time_24hr .lt. 12.0d0) then
                if (ze .lt. z) then
                  f = zero
                  go to 10
                endif
c   for times after solar noon:
              elseif (time_24hr .gt. 12.0d0) then
                if (ze .gt. z) then
                  f = zero
                  go to 10
                endif
c   for time = solar noon (z wont discriminate so need to look at x coordinate):
              else
                if (vsi .lt. zero .and. xe .gt. x) then
                  f = zero
                  go to 10
                endif
                if (vsi .gt. zero .and. xe .lt. x) then
                  f = zero
                  go to 10
                endif
              endif
            endif
          endif
c
    5     continue
```

```
c  no shadowing, continue with do loop to check next wall
c
          end do
   10       continue
c
c  determine total surface flux accounting for orientation
c   apply fdif to account for diffuse flux component and apply clearness factor
c   to modify direct component (only the direct component is affected by clearness)
          surface_flux = solar_flux * (f * fclear + fdif)

        else
c
c  only diffuse component is applied
          surface_flux = solar_flux * fdif

        endif
c
c  assign fluxes
        do j = 1, nip
          flux(i,j) = surface_flux
        end do
        call acal_put_real_face_var(surface_flux,faceID(i),9,
     &                              "SolarFlux",found)
c
       end do
c
     endif
c
c
     return
     end
c
c   ================================================================================
```

Diurnal Air Temperature:

```
c
     subroutine DiurnalAirT(faceID, nelem, nint, coords, t,
     &                      refTemp, ierror)
c
c  calculate diurnal cycle air temperature for convective reference temperature
c   determine sunrise time based on latitude and day of year, and account for yearly variation
c   based on weather data, which can be obtained from such sites as weatherbase.com
c   created by Dean Dobranich, 1/2009
c
     implicit none

     integer nelem
     integer nint
     integer faceID(nelem)
     double precision coords(3, nelem, nint)
     double precision t(nelem, nint)
     double precision refTemp(nelem, nint)
     integer ierror

c
```

```
c     ierror:  (output)user defined error code for calore to test.
c              zero means success.
c     nelem:   (input)number of elements in workset
c     nint:    (input)number of elements in workset
c     refTemp: (output)array containing the magnitude of the reference
c              temperature that the subroutine calculates at a face
c     coords:  (input) coords of the integration points on each element.
c     t:       (input)array of temperatures at the integration points
c              on each element

      integer i, j, nr
      double precision time, time_24hr, time_days
      double precision JulianDay, latitude, declination, hour_angle
      double precision daylight, Tref_max, Tref_min, period
      double precision T_high_max, T_high_min, JD_high_max
      double precision T_low_max, T_low_min, JD_low_max
      double precision T_high_avg, T_high_amp, T_low_avg, T_low_amp
      double precision Tref_avg, Tref_amplitude
      double precision offset_avg, offset
      double precision Lag, refT, arg
      double precision pi, twopi_o_period, one
c
      parameter (pi=3.14159265358979323846264433d0)
      parameter (twopi_o_period=2.0d0*pi/24.0d0)
      parameter (one=1.0d0)

c  nr is number of real data variables read in from .i file
      parameter (nr=9)
      double precision rdata(nr)
c
      ierror = 0
c
c  Tref = Tref_avg + Tref_amplitude*cos(2*pi/period*(t + offset))
c   with offset = offset_avg + Lag*(cos(2*pi/period*(time + offset_avg))
c   rdata(1) => Julian day (fractional day permissible, i.e., 3.5 indicates half way
c               through the third day of the year; progresses with simulation time; max of 365)
c   rdata(2) => latitude (+ for northern hemisphere), used with day to find sunrise time
c   rdata(3) => T_high_max (maximum high daily temperature throughout the year)
c   rdata(4) => T_high_min (minimum high daily temperature throughout the year)
c   rdata(5) => JD_high_max (Julian day corresponding to max high daily temperature)
c   rdata(6) => T_low_max (maximum low daily temperature throughout the year)
c   rdata(7) => T_low_min (minimum low daily temperature throughout the year)
c   rdata(8) => JD_low_max (Julian day corresponding to max low daily temperature)
c   rdata(9) => Lag (time interval after sunrise that air temperature is minimum; also
c               the time interval before sunset that air temperature is maximum; usually ~1)

c
c  get current simulation time and convert to hours
      call acal_get_time(time)
      time = time/3600.0d0
      time_days = time/24.0d0
c  use a 24 hour repeating clock
      time_24hr = mod(time, 24.0d0)

      call acal_get_instance_real_data(nr,rdata)
      JulianDay = mod(rdata(1) + time_days, 365.0d0)
```

```
      latitude = rdata(2)*pi/180.0d0
      T_high_max = rdata(3)
      T_high_min = rdata(4)
      JD_high_max = rdata(5)
      T_low_max = rdata(6)
      T_low_min = rdata(7)
      JD_low_max = rdata(8)
      Lag = rdata(9)

c  declination (radians); 23.45 degrees = 0.40927971 radians
c   truncate Julian Day for calculation of sunrise time
      declination = 0.40927971d0*dsin(pi/182.5d0*
     &             (dint(JulianDay)+284.0d0))
      arg = -dtan(latitude)*dtan(declination)
      arg = max(arg,-one)
      arg = min(arg,one)
c  argument of arccos must be between -1 and 1, inclusive (bounds would be
c   exceeded for polar angles when daylight may equal 0 or 24 hours)
      hour_angle = dacos(arg)
c      hour_angle = dacos(-dtan(latitude)*dtan(declination))
      daylight = 24.0d0/pi*hour_angle

c  set cosine function for yearly T variation based on min and max high and low Ts
      T_high_avg = (T_high_max + T_high_min)*0.5d0
      T_high_amp = (T_high_max - T_high_min)*0.5d0
      T_low_avg = (T_low_max + T_low_min)*0.5d0
      T_low_amp = (T_low_max - T_low_min)*0.5d0
c
      offset = 365.0d0-JD_high_max
      Tref_max = T_high_avg + T_high_amp*
     &          (dcos(pi/182.5d0*(JulianDay + offset)))
      offset = 365.0d0-JD_low_max
      Tref_min = T_low_avg + T_low_amp*
     &          (dcos(pi/182.5d0*(JulianDay + offset)))

c  now set cosine function for daily T variations
c
c  Tref = Tref_avg + Tref_amplitude*cos(2*pi/period*(t + ov))
c   with ov = offset_avg + Lag*(cos(2*pi/period*(time + offset_avg))
      Tref_avg = (Tref_max + Tref_min)*0.5d0
      Tref_amplitude = (Tref_max - Tref_min)*0.5d0
      offset_avg = daylight*0.5d0
c
c  calculate refT vs time
c   offset shifts the cosine distribution to the left
      offset = offset_avg + Lag*
     &         (dcos(twopi_o_period*(time_24hr + offset_avg)))
      refT = Tref_avg + Tref_amplitude*
     &       (dcos(twopi_o_period*(time_24hr + offset)))

c
c  pass reference temperature to Calore for subsequent output (add as user variable in Calore)
      call acal_lupdate_global_real_var(refT,8,"Tref-air")
c
c  assign Tref
      do j = 1, nint
```

```
      do i = 1, nelem
        refTemp(i,j) = refT
      end do
    end do

    return
    end

c   ================================================================

c
    subroutine getDiurnalAirT(faceID, nelem, nint, coords, t,
    &                         refTemp, ierror)
c
c  get diurnal cycle air temperature for convective reference temperature
c   this subroutine simply gets the user variable Tref-air, which is determined
c   by the DiurnalAirT subroutine. This avoids having to recalculate this quantity
c   for multiple convection boundary conditions.
c   created by Dean Dobranich, 1/2009


    implicit none

    integer nelem
    integer nint
    integer faceID(nelem)
    double precision coords(3, nelem, nint)
    double precision t(nelem, nint)
    double precision refTemp(nelem, nint)
    integer ierror

c
c     ierror:  (output)user defined error code for calore to test.
c              zero means success.
c     nelem:   (input)number of elements in workset
c     nint:    (input)number of elements in workset
c     refTemp: (output)array containing the magnitude of the reference
c              temperature that the subroutine calculates at a face
c     coords:  (input) coords of the integration points on each element.
c     t:       (input)array of temperatures at the integration points
c              on each element

    integer i, j
    double precision refT
c
    ierror = 0

c
c  the following call retrieves Tref-air from a user subroutine that calculates
c   the convective reference temperature; this user variable must be available
    call acal_get_global_real_var(refT,8,"Tref-air")
c
c  assign Tref
    do j = 1, nint
      do i = 1, nelem
        refTemp(i,j) = refT
```

```
      end do
     end do

     return
     end
c
```

Sky Temperature:

```
c
     subroutine Sky_T(faceID, nelem, nint, coords, t,
    &               refTemp, ierror)
c
c  calculate sky temperature based on the diurnal cycle air temperature
c   used as the reference temperature for far-field radiative boundary conditions
c   reference: Sherwood B. Inso and Ray D. Jackson
c            "Thermal Radiation from the Atmosphere"
c             Journal of Geophysical Research, Vol. 74, No. 23, October 20, 1969.
c   created by Dean Dobranich, 12/2008
c
     implicit none

     integer nelem
     integer nint
     integer faceID(nelem)
     double precision coords(3, nelem, nint)
     double precision t(nelem, nint)
     double precision refTemp(nelem, nint)
     integer ierror

c
c    ierror:  (output)user defined error code for calore to test.
c             zero means success.
c    nelem:   (input)number of elements in workset
c    nint:    (input)number of elements in workset
c    refTemp: (output)array containing the magnitude of the reference
c             temperature that the subroutine calculates at a face
c    coords:  (input) coords of the integration points on each element.
c    t:       (input)array of temperatures at the integration points
c             on each element

     integer i, j
     double precision refT, skyT
c
     ierror = 0

c
c  the following call retrieves Tref-air from a user subroutine that calculates
c   the convective reference temperature; this user variable must be available
     call acal_get_global_real_var(refT,8,"Tref-air")
c
     skyT = (refT**4*(1.0d0 - 0.261d0*dexp(-7.77d-4*
    &       (273.0d0 - refT)**2)))**0.25d0
c
c  pass reference temperature to Calore for subsequent output (add as user variable in Calore)
     call acal_lupdate_global_real_var(skyT,8,"Tref-sky")
```

```
c
c  assign Tref
      do j = 1, nint
        do i = 1, nelem
          refTemp(i,j) = skyT
        end do
      end do

      return
      end
#   ================================================================
#
```

Sample Input:

```
#  ================================================================

      Begin Heat Flux Boundary Condition SolarFlux
        add surface surface_3  #exposed surfaces, not including bottom
        #add surface surface_1  #top only

        element subroutine is Solar_flux_shadows

#
#  qpp = absorbed total solar flux vs time; includes direct and diffuse components
#   rdata(1) => Julian day (fractional day permissible, i.e., 3.5 indicates half way
#               through the third day of the year; progresses with simulation time)
#   rdata(2) => latitude (+ for northern hemisphere)
#   rdata(3) => solar absorptivity of surface
#   rdata(4) => ext_coef (extinction coefficient through atmosphere, typically = 0.2)
#   rdata(5) => fdif (fraction of solar flux that is diffuse, typically = 0.1)
#   rdata(6) => fclear (clearness factor, direct solar flux component multiplier;
#               = 1.0 for clear day, 0.8 for 20% cloud cover, etc.)
#   rdata(7) => shadow_dt1 (time when structure is in shadow relative to sunrise)
#   rdata(8) => shadow_dt2 (time when structure is in shadow relative to sunset)
#               thus, if time is less that sunrise + shadow_dt1 or greater than
#               sunset - shadow_dt2, then only diffuse flux is applied
#   rdata(9-m) => x,y,z coordinates of walls 1 through max_num_walls with 2 coords per wall
#                 where m = 8 + max_num_walls*6 (currently max_num_walls = 8)
#

        #real data 172.0, 10.0, 0.8, 0.2, 0.1, 1.0, 0.0, 0.0
        integer data 2   #nwalls (data for max_num_walls must be entered but only nwalls used)
        real data 218.0, 30.0, 0.8, 0.2, 0.1, 1.0, 0.0, 0.0  \$
                  0.0 1.6 0.6  0.55 1.2 0.6 \$
                  0. 1.4 0.  0. 1.6 0.35 \$
                  0. 0. 0.  0. 0. 0. \$
                  0. 0. 0.  0. 0. 0. \$
                  0. 0. 0.  0. 0. 0. \$
                  0. 0. 0.  0. 0. 0. \$
                  0. 0. 0.  0. 0. 0. \$
                  0. 0. 0.  0. 0. 0.

        integrated power output q_Solar
        integrated flux output qpp_Solar
      end
```

```
#
#  ============================================================
#
     Begin Convective Flux Boundary Condition BoxC
       Add Surface surface_5001 #exposed surfaces, including bottom

       Convective coefficient = 2.85  #for a 1 m high vertical plate

       Reference Temperature Subroutine is DiurnalAirT

#  Tref = Tref_avg + Tref_amplitude*cos(2*pi/period*(t + offset))
#   with offset = offset_avg + Lag*(cos(2*pi/period*(time + offset_avg))
#   rdata(1) => Julian day (fractional day permissible, i.e., 3.5 indicates half way
#               through the third day of the year; progresses with simulation time; max of 365)
#   rdata(2) => latitude (+ for northern hemisphere), used with day to find sunrise time
#   rdata(3) => T_high_max (maximum high daily temperature throughout the year)
#   rdata(4) => T_high_min (minimum high daily temperature throughout the year)
#   rdata(5) => JD_high_max (Julian day corresponding to high daily temperature)
#   rdata(6) => T_low_max (maximum low daily temperature throughout the year)
#   rdata(7) => T_low_min (minimum low daily temperature throughout the year)
#   rdata(8) => JD_low_max (Julian day corresponding to low daily temperature)
#   rdata(9) => Lag (time interval after sunrise that air temperature is minimum; also
#               the time interval before sunset that air temperature is maximum; usually ~1 hr)

       real data 218.0 30.0 311.4833 300.3722 196.0 305.9278 297.0389 196.0 1.0

       integrated power output qc_box
       integrated flux output qcpp_box
     End Convective Flux Boundary Condition BoxC
#
#  ============================================================
#
     Begin Radiative Flux Boundary Condition BoxR  #
       add surface surface_2 surface_3  #bottom and sides

       emissivity = 0.8
       Radiation Form Factor is 1.0

# this subroutine relies on the DiurnalAirT subroutine to calculate air temperature
#  that assigns the Tref-air user variable
       Reference Temperature Subroutine is Sky_T

       integrated power output qr_box
       integrated flux output qrpp_box
     End
#
#  ============================================================
#
```

Distribution:

3	MS-0346	01514	D. Dobranich
1	MS-0826	01512	M. T. Valley
1	MS-0836	01514	T. L. Aselage

Electronic copy only:

1	MS-0899	09536	Technical Library
1	MS-0382	01514	D. R. Noble
1	MS-0382	01541	S. W. Bova
1	MS-0382	01541	S. R. Subia
1	MS-0382	01543	M. W. Glass
1	MS-0382	01541	B. Hassan
1	MS-0382	01541	T. O. Okusanya
1	MS-0405	00426	R. D. Waters
1	MS-0614	02547	S. A. Whalen
1	MS-0734	06337	J. R. Tillerson
1	MS-0735	06313	G. T. Klise
1	MS-0828	01544	A. R. Black
1	MS-0828	01544	V. J. Romero
1	MS-0836	01510	J. S. Lash
1	MS-0836	01514	R. L. Akau
1	MS-0836	01514	R. C. Dykhuizen
1	MS-0836	01514	N. D. Francis
1	MS-0836	01514	R. E. Hogan, Jr.
1	MS-0968	02611	B. D. Boughton
1	MS-1033	06335	C. J. Hanley


Sandia National Laboratories