Do you have this one? PWV

C/C9800401

CRADA Final Report
for
CRADA Number C9800401

Automotive Underhood Thermal Management Analysis Using 3-D Coupled Thermal-Hydrodynamic Computer Models: Thermal Radiation Modeling

Sreekanth Pannala
Eduardo D'Azevedo
Thomas Zacharia
Oak Ridge National Laboratory

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
UT-Battelle, LLC
for the
U.S. Department of Energy
Under contract DE-AC05-00OR22725

PROTECTED CRADA INFORMATION
This product contains Protected CRADA Information which was produced on February 26, 2002 under CRADA No. C9800401 and is not to be further disclosed for a period of five (5) years from the date it was produced except as expressly provided for in the CRADA.

A. Abstract:

The goal of the radiation modeling effort was to develop and implement a radiation algorithm that is fast and accurate for the underhood environment. As part of this CRADA, a net-radiation model was chosen to simulate radiative heat transfer in an underhood of a car. The assumptions (diffuse-gray and uniform radiative properties in each element) reduce the problem tremendously and all the view factors for radiation thermal calculations can be calculated once and for all at the beginning of the simulation. The cost for online integration of heat exchanges due to radiation is found to be less than 15% of the baseline CHAD code and thus very manageable. The off-line view factor calculation is constructed to be very modular and has been completely integrated to read CHAD grid files and the output from this code can be read into the latest version of CHAD. Further integration has to be performed to accomplish the same with STAR-CD.

The main outcome of this effort is to obtain a highly scalable and portable simulation capability to model view factors for underhood environment (for e.g. a view factor calculation which took 14 hours on a single processor only took 14 minutes on 64 processors). The code has also been validated using a simple test case where analytical solutions are available. This simulation capability gives underhood designers in the automotive companies the ability to account for thermal radiation - which usually is critical in the underhood environment and also turns out to be one of the most computationally expensive components of underhood simulations.

This report starts of with the original work plan as elucidated in the proposal in section B. This is followed by Technical work plan to accomplish the goals of the project in section C. In section D, background to the current work is provided with references to the previous efforts this project leverages on. The results are discussed in section 1E. This report ends with conclusions and future scope of work in section F.

B. Radiation Modeling Effort as described in the Original Proposal:

B.1 First Year

- **Initial Model development**
  - Develop the radiation heat transfer model and perform preliminary integration into the CHAD code.

B.2 Second Year

- **Model Refinement and Integration**
  - Complete the radiation model integration
  - Work with ANL and other partners in performing an evaluation of condenser models and development needs for implementation in CHAD

B.3 Third Year

- **Verification and Validation of Commercial Version of the Code**

- Work in co-operation with ANL and other participants to improve the models
- Verify and Validate new models within the framework of the release version of the CHAD code

C. Technical Work Plan:

A stand-alone FORTRAN 90 code to calculate view factors for radiation model is developed at ORNL under a previous CRADA by Williams et al. (Ref. 3). This is based on the FACET code developed at LLNL by Shapiro (Ref. 4). The integration to CHAD has been accomplished by another routine that computes the source terms to the energy equation. Some of the tasks, which need to be performed for the completion of tasks listed along with work plan, are:

a) Evaluate the current thermal radiation algorithm as compared to others in the literature to see whether the net-radiation model is efficient and accurate enough for underhood simulations.

b) Parallelize the stand-alone routine to have reduced wall-clock times for view-factor calculations. With the use of SCALAPACK routines available and HPF compilers, this step can be easily achieved. The parallelization procedure will be done using Shared Memory Parallilization (SMP) in the beginning and later extend to Distributed Memory Parallelization (DMP) using Message Passing Interface (MPI).

c) Validation and verification of the view factor calculations and if possible the validation of thermal radiation model in totality coupled to CHAD or STAR-CD.

d) Explore more intelligent and inbuilt approach to lump computational elements into radiation elements rather than the need to do them manually. On a same note, a provision should be provided to only deal with the most critical elements based on criteria like (line-of-sight, surface properties like temperature & radiation properties). The problem being addressed is similar to the ones found in the studies of Computational Geometry and Radiosity in Computer Graphics. A rigorous survey of available literature in those fields can give enough insight addressing this problem.

Work plan described above is expected to produce a working and validated thermal radiation model completely integrated into the CHAD code with possible migration to STAR-CD. The capability will not only be helpful in studying the Underhood problem but also in any flow where heat-transfer rates due to thermal radiation can be significant.

D. Background on thermal radiation models and Net-Radiation Model Implementation Details:

D.1 Comparison of thermal Radiation Models commonly used:

Over the past 3-4 decades thermal radiation modeling has received attention – this is due to both the need for some applications and also the ability to incorporate thermal radiation effects with the advent of super computers. The most popular choices for simulating radiative heat transfer are (Ref. 1 & 2):

a) Net-Radiation Model

b) The method of Spherical Harmonics

c) The method of Discrete Ordinates

d) Zonal Methods

e) Monte Carlo Methods

One of the main hindrances for modeling thermal radiation is the exorbitant computational cost associated with modeling the full set of radiative transport equations. Thermal radiation not only depends on the surface properties but also the characteristics of the incident rays and the properties of the surfaces where the radiation is being emitted. In some instances, using Discrete Ordinates methods, the cost of the thermal radiation model can reach 90% of the total solver time. This essentially means the computational cost increases by 10 times just by adding radiation model to the basic solver. In Underhood environment several assumptions can be made and thus a net-radiation model can be applied. More details of this model are described in the next section but an important observation is that addition of the radiation model only increased the computational time of the baseline solver by 15% and thus making radiation modeling very affordable. When the assumptions as detailed in the following section fail, Net-Radiation model does not offer the best framework to model thermal radiation. In other words, this model is severely restricted but the assumptions made are usually valid over a wide range of problems of practical interest – including HVAC applications.

D.2 Net-Radiation Model for GRAY-DIFFUSE Surfaces

In the previous section various methods used for thermal radiation model are mentioned and of them Net-Radiation model is the simplest. Consider two surfaces are listed in Fig. 1. If the following assumptions can be made:

- The temperature is constant over each surface but can be different from others
- The radiative surface properties are uniform
- Diffuse-Gray assumption: The emissivity, abosorptivity, reflectivity and transmissitivity are independent of direction and wavelength
- All the radiation properties are at most dependent on surface temperatures
- The geometric view factors are same as that of black-body and do not vary with time for a fixed geometry
- Air is assumed to be radiatively non-participant medium
- The semi-transparent surfaces are assumed to be symmetric such that radiative properties are same on both sides.

The view factor between surfaces I and J reduces to the following equation (for more details see Ref. 1):

$$F_{IJ} = \frac{1}{A_I}\int_{A_I}\int_{A_J}\frac{\cos\beta_I \cos\beta_J \, dA_I dA_J}{\pi r^2} \quad \text{.......... Equation 1}$$

The main advantage of the above assumptions is that these view factors can once and for all be computed at the beginning of the simulation and can be re-used. This drastically cuts down the expense of the thermal radiation calculations. In the following sections more detailed information will be provided about how these view factors are computed and also how the view factors are used to compute the heat transfer due to radiation.

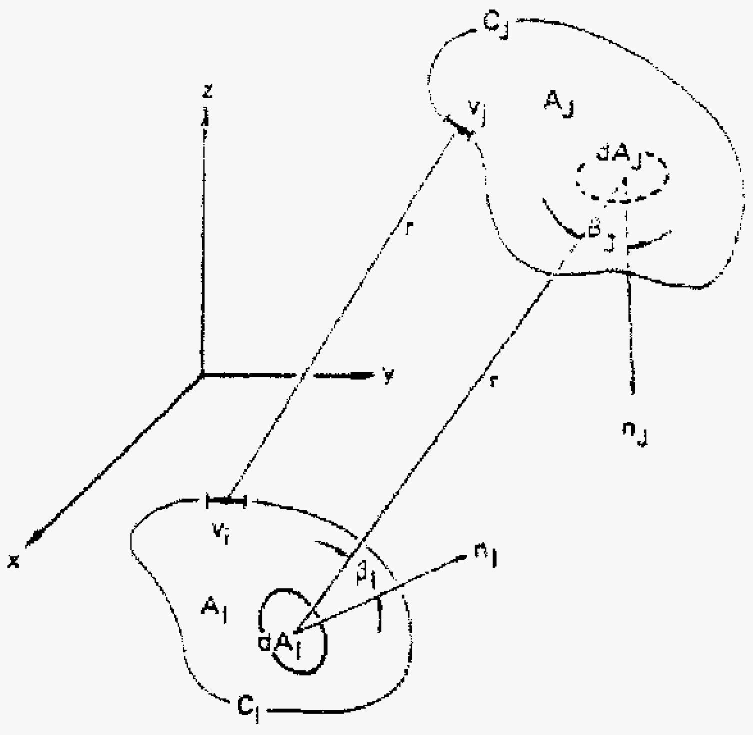


Figure 1 View Factor Calculation

D.3 Implementation Details:

In the previous section an overview of the net-radiation model is given. In this section, the implementation of the net-radiation model is reported (for more details see Ref. 3 & 4). The view factors are computed offline using CHADVIEW code. This is a recast of FACET code (Ref. 4). Here is a brief description of the CHADVIEW code.

D.3.1 Stand-alone CHADVIEW code (Appendix A):

- This code calculates geometric view factors (F_{IJ})
- It is recast in Fortran 90 and memory is dynamically allocated
- Interface to read CHAD mesh and input files has been implemented

- Provides input to online routine User_Srcs and LU factors for efficient linear solver

The geometric view factors can be calculated from Eq. 1. The straight way of integration is to discretize in the following manner.

$$F_{IJ} \cong \frac{1}{A_I}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\cos\beta_i \cos\beta_j \mathrm{A_i A_j}}{\pi \mathrm{r}^2} \text{ ....} \text{Equation 2}$$

This procedure turns out to be very expensive and inaccurate. It has been found that alternate procedures can strike a balance between accuracy and speed. For two non-adjacent surfaces with no shadowing – one of the integration can be performed analytically and arrive at the following expression for contour integration. This is extremely efficient compared to Area Integration and is reported in Ref. 4.

$$F_{IJ} \cong \frac{1}{2\pi A_I}\sum_{i=1}^{n}\sum_{j=1}^{n}\ln \mathrm{r_{ij}}\, \hat{\mathrm{v}}_\mathrm{i} \bullet \hat{\mathrm{v}}_j \text{ ....} \text{Equation 3}$$

Jonnavithula (private communication) at adapco, NY found that the above expression is not dimensionally correct. On further exploration, it was found that in most of the literature, above equation was referred to but no point was made about the dimensional inconsistency. It turns out that irrespective of what scaling one uses, the above formula is correct as a contour integral of a constant is zero. When the surfaces are adjacent, contour integration does not work as it leads to a singularity. Mitalas and Stephenson (Ref. 5) have performed analytical integration and have come up with the following expression for view factors.

$$F_{IJ} \cong \frac{1}{2\pi A_I}\sum_{p=1}^{4}\sum_{q=1}^{4}\Phi(p,q)\sum_{j=1}^{n}\left[\left(T\cos\phi \ln \mathrm{T} + \mathrm{S}\cos\theta \ln \mathrm{S} + \mathrm{U}\omega - \mathrm{R}\right)\left|\hat{v}_j\right|\right]_{p,q} \text{ ....} \text{Equation 4}$$

Only when the above conditions are not met – resort to Area Integration is taken. More details of the implementation are given in Ref. 4.

D.3.2 Subroutine User_Srcs

Once the view factor information is obtained – it is used in the following expression to calculate the heat fluxes from thermal radiation.

$$\sum_{j=1}^{N}\left(\frac{\delta_{i,j}}{\varepsilon_j} - \frac{1-\varepsilon_j-\tau_j}{\varepsilon_j}F_{i-j}\right)\tilde{q}_j = \sum_{j=1}^{N}F_{i-j}\left(\sigma\left(T_k^4-(1-\tau_j)T_j^4\right)-\tau_j q_{e,j}\left(\frac{\delta_{i,j}}{\varepsilon_j}-\frac{1-\tau_j}{\varepsilon_j}\right)\right)$$

where $\tilde{q}_j$ is defined as:

for semi - transparent surfaces: ..Equation 5

$$\tilde{q}_j = q_j - \varepsilon_j \sigma T_j^4 + (\varepsilon_j + \tau_j) q_{e,j}$$

for opaque and transparent surfaces: $\tilde{q}_j = q_j$

In summary this routine provides and details are available in Ref. 3:

- Online integration of thermal radiation model into CHAD

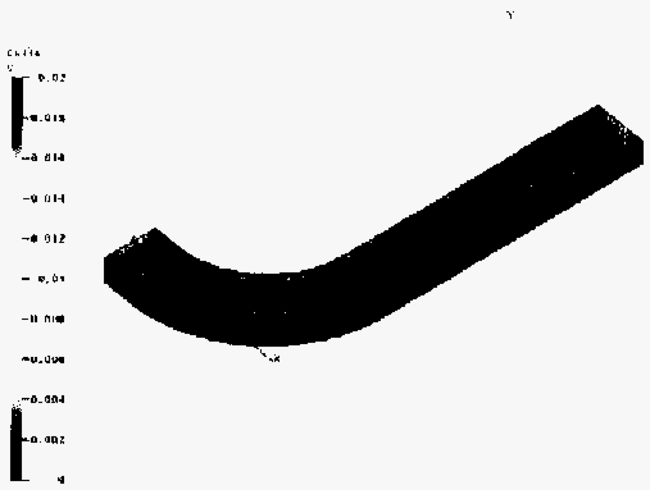
- It is called from subroutine sources in CHAD to obtain surface heat fluxes due to radiation
- Capable of handling partially transparent and transparent surfaces with external radiation loads

E. Results:

The algorithms implemented in the stand-alone code are presented in the previous sections. In this section, the results obtained are reported. First experiments were to establish the computational cost of radiation model and also with respect to the cost of the online integration of the radiation model with the baseline CHAD solver. This is followed by a report on the SMP parallelization and then the DMP parallelization. A simple validation is presented at the end. The present CHADVIEW code is not integrated completely into STAR-CD format and thus direct comparisons to STAR results were not performed.

E.1 Simple experimentation of Square Duct with CHAD and CHADVIEW:

The first set of results is for a simple case of curved duct with square cross-section. The computational details of the problem along with the various timings are given in Fig. 2. From these details one can conclude that view factor computations are most expensive part of the CHADVIEW routine and Contour Integration is called most number of times. The computational cost also increases as the number of sub-divisions of the element increases. It has been also found that running this problem with additional contributions from thermal radiation only adds 15% cost to the baseline CHAD solver. This is very reasonable compared to the computational expense of other radiation solvers.



Problem Size: 4116 Nodes, 3168 Cells
1764 Surface Nodes

View Factor Computations:
3 Subdivisions:
- View Factor Compt. : 272 Secs
- LU Factorization: 76 Secs

7 Subdivisions:
- View Factor Compt. : 2241 Secs
- LU Factorization: 63 Secs
- LI = 880538; MS = 511; AI = 134872

Figure 2 Curved Duct of Square Cross-section (with radiation)

E.2 SMP Parallelization:

The results from the previous section indicate that the view factor calculations can be expensive and they are even more computational intensive as the number of surface elements increase. The first approach to address this issue is to parallelize this code so that the view factor calculations can be performed on multiple processors. The first step in this direction was to have a shared memory implementation. The IO and was performed by the root processor. The code was profiled and the most intensive calculations were divided for different processors while the less intensive calculations were performed by the root processor. The parallel performance of this SMP code is shown in Fig. 3. There is near ideal scale-up till 4 processors and the parallel performance degrades at 8 processors. This is typical of SMP parallelization and this might be aggravated by the fact that some less intensive work is only performed by the root. Based on this study it was decided to have a distributed version of the code and the results are reported in the next section.

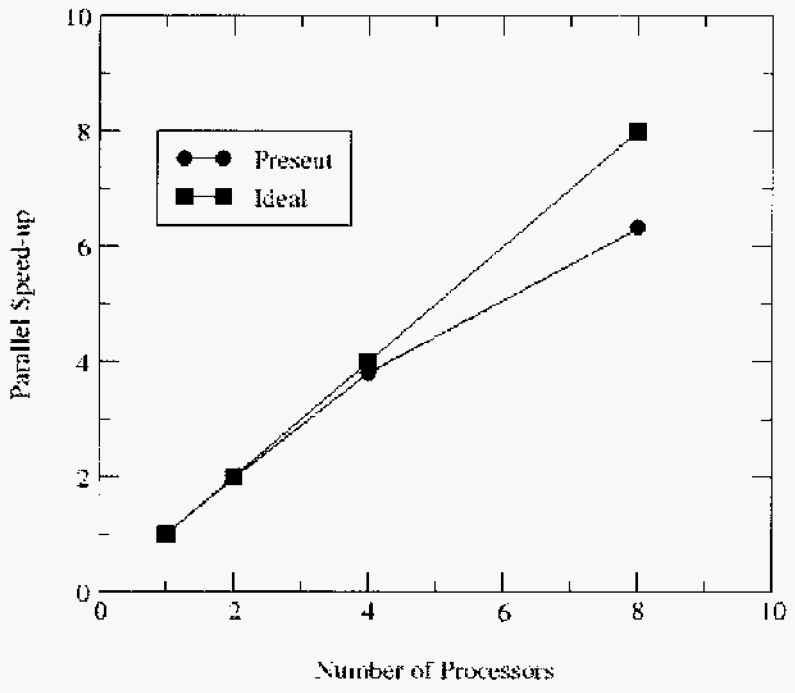


Figure 3 Parallel Scale-up for the SMP version of CHADVIEW

E.3 DMP Parallelization:

The CHADVIEW code has been parallelized for shared memory systems and has been reported in the previous section. The shared memory parallelization – even though easier to implement lacks in the area of high efficiency at large number of processors and also very restrictive to certain computer hardware. Here a brief overview of the parallelization strategy is given along with some results showing the parallel performance. The following strategy has been adopted to port the CHADVIEW to distributed memory systems using MPI:

- The IO operations are handled by the root processor
- The surface element information is processed by the root processor and on all the processors
- Equally distribute the view-factor computations on all the processors
- The view-factor information is collected to the root processor using MPI allreduce (message size is controlled for efficient operation).
- The LU factorization is carried out using ScaLAPACK

View factors are obtained for the earlier sample problem of curved duct and are plotted in Figure 4. The parallel performance is near ideal all the way upto 16 processors unlike the SMP version of the code. It can be also noted that the most intensive part of the code (view factor calculations) is highly scalable. A more detailed profile of the code is given in Table 1. Here it can be observed that view factor calculations are very highly scalable and even for such a small problem ScaLAPACK performs reasonably well for the LU factorization.

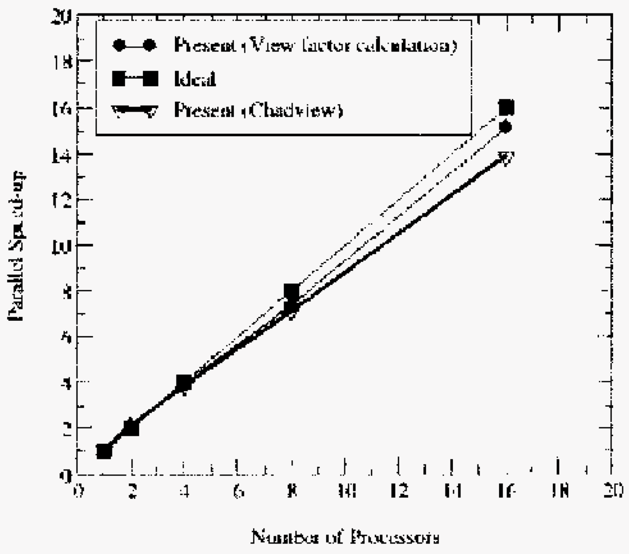


Figure 4 Parallel Scale-up for the DMP version of CHADVIEW

No. of Procs. / Chadview Routines	1	2	4	8	16
Initialization & Data Input	1.3	1.92	2.71	2.60	2.6
Area*view factor computation	726	340	187	98	48
Reciprocity calc. & output	.55	.54	.50	.48	.49
Row-sums calculation	.28	.27	.27	.26	.27
LU Factorization of AMAT	7.16	4.25	2.99	2.53	1.75

Table 1 Profile of the DMP version of CHADVIEW

To test the distributed parallel version further – a problem with 4,800 surface nodes is chosen. This is a simple cube with thermal radiation. The parallel performance is shown in Fig. 5 and the super-linear scale-up is observed till 32 processors. Further it scaled well upto 64 processors. It is worthwhile to note that the calculation that would have taken 14 hours on single processor only takes 14 minutes on 64 processors.

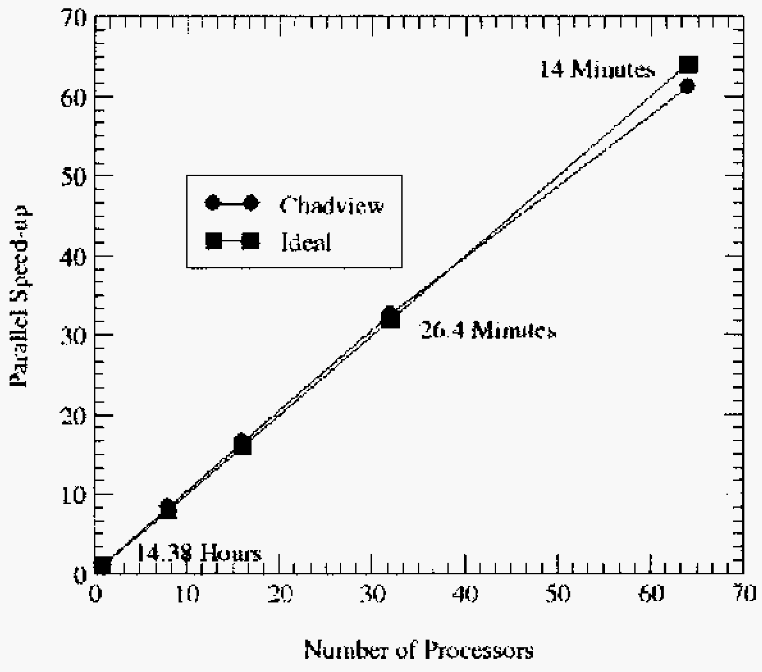


Figure 5 Parallel Scale-up for the DMP version of CHADVIEW for the Cube problem.

E.4 Simple Validation:

In the absence of detailed experimental data – a simple validation has been performed. A cube with enclosing walls is used as an example. The problem is shown in Fig. 6. Analytical solution yields a value of 0.2 for the view factor between the bottom surface and the top surface or bottom surface and any of the side surfaces. Figure 7, plots the error for both the surfaces. A trapezoidal rule was used for contour integration and the perpendicular surfaces employed the Mitalas and Stephenson method as the contour integration gave rise to higher error. As you can note from the figure that the error does not decrease as a second order scheme and it stagnates after 5 sub-divisions. A 2nd order quadrature integration was also employed with similar result.

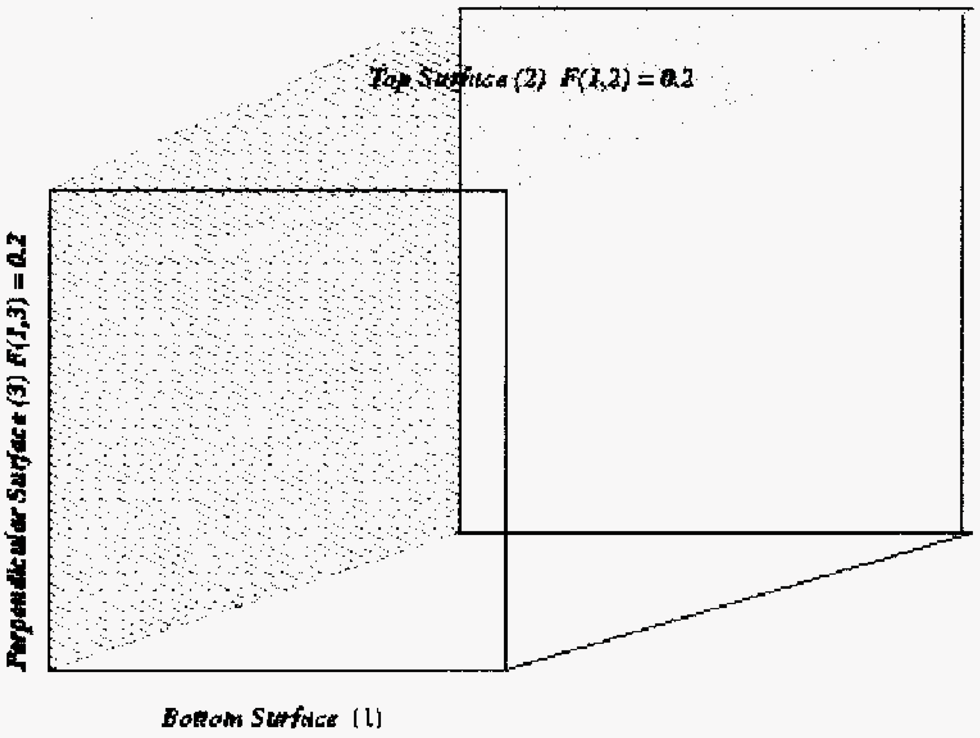

Top Surface (2) F(1,2) = 0.2
Perpendicular Surface (3) F(1,3) = 0.2
Bottom Surface (1)

Figure 6 Simple Validation case of a cube

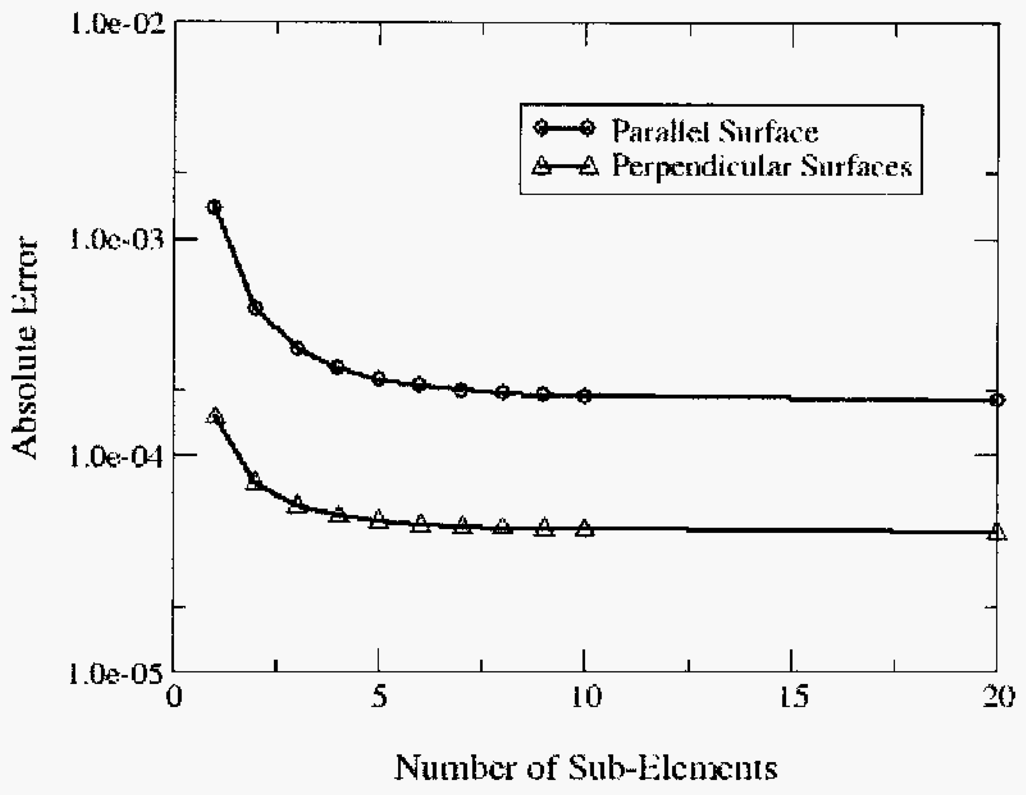


Figure 7 Number of sub-elements vs. Absolute error for the Cube Problem

Comparing Fig. 7 with Fig. 8, where computational time is plotted versus number of sub-elements, one can conclude that for this problem one can use just a few sub-elements to get reasonably accurate view factors without incurring too much computational cost. More detailed studies like these on real problems will provide valuable database to determine semi-automatic patching and sub-dividing algorithms for view factor calculations.

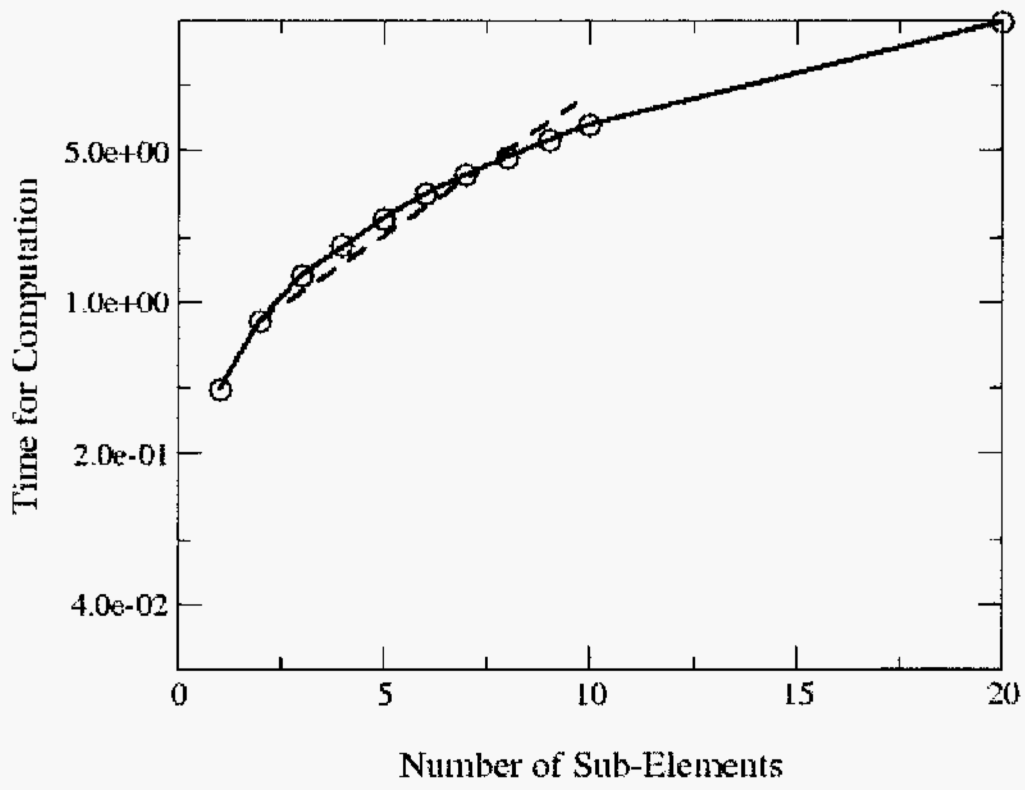


Figure 8 Computational time versus number of sub-elements.

F. Conclusions and future scope of work:

The goal of the radiation modeling effort was to develop and implement a radiation algorithm that is fast and accurate for the underhood environment. As part of this CRADA, a net-radiation model was used to simulate radiative heat transfer in an underhood of a car. The cost for online integration of heat exchanges due to radiation is found to be less than 15% of the baseline CHAD code and thus very manageable. The off-line view factor calculation is constructed to be very modular and has been completely integrated to read CHAD grid files and the output from this code can be read into the latest version of CHAD. Further integration has to be performed to accomplish the same with STAR-CD.

The final outcome of this effort is a highly scalable and portable simulation capability to model view factors for underhood environment (for e.g. a view factor calculation which took 14 hours on a single processor only took 14 minutes on 64 processors). The code has also been validated using a simple test case where analytical solution is available.

The future scope of work includes more close integration with STAR-CD and testing real underhood geometries. Detailed validations against experiments would be very useful to examine the validity of various assumptions used and the accuracy of the thermal radiation calculations. A considerable effort can be used in the area of better algorithms for shadow calculations and also in the areas of developing rule-based algorithms for patching and sub-dividing elements. A prelude to such an effort are given in the following section

G. Exploration of alternate algorithms for fast shadow calculations and patching:

In this section some overview of the alternate algorithms for shadow calculations and patching will be reported. Based on limited literature search and the understanding of the thermal radiation model – these conclusions are arrived at. The algorithms are not tested to have complete understanding of the gains they offer.

G.1 Shadow Computations:

Here is a brief outline of how shadow computations can be performed in serial. This procedure will reduce the computational cost. This is due to the fact that the algorithm being proposed is of order N^2 whereas the original algorithm implemented in CHADVIEW is N^3.

- Extract wall surfaces
- Create Hierarchy of the surfaces based on number of surface nodes
- Use ray tracing to sweep through surfaces in hierarchical fashion (from the nearest objects to the farthest)
    - Start with the nearest objects. Determine the shadow region and the lighted region with respect to the object picked and source.
    - No more computations in the leeward side and the objects in the shadow of this object (refer Fig. 9). This is the case because the objects in the leeward side do not see the current source.
    - This procedure is of order N^2 because one has to only march through points on the object surfaces which are lighted instead of all the surfaces in the enclosure as it is currently set. The cost is coming up with a suitable parallel algorithm.
- Once the line-of-sight matrix is built, the view factor computation is relatively straightforward like before.

Use visibility mark and visibility interface to perform this operation in parallel (Fig. 10 and Ref. 6). Here rays are shot in all directions to determine the shadow/lighted region. This information is marked on the visibility mask and projected to a visibility interface – a plane dividing the regions on different processors. This information is communicated to all the processors and these are associated with each source. Each processor will march through the information sent from all the sources relevant to it, to determine what the final shadow/lighted region. This procedure can be used to extend the above outlined algorithm for parallel computations.

The proposed method has not been implemented but added to this report for any future reference.

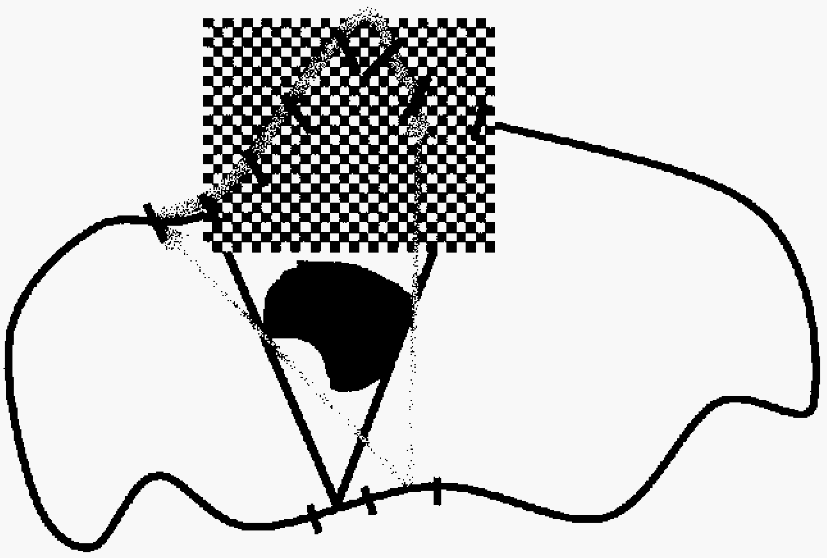
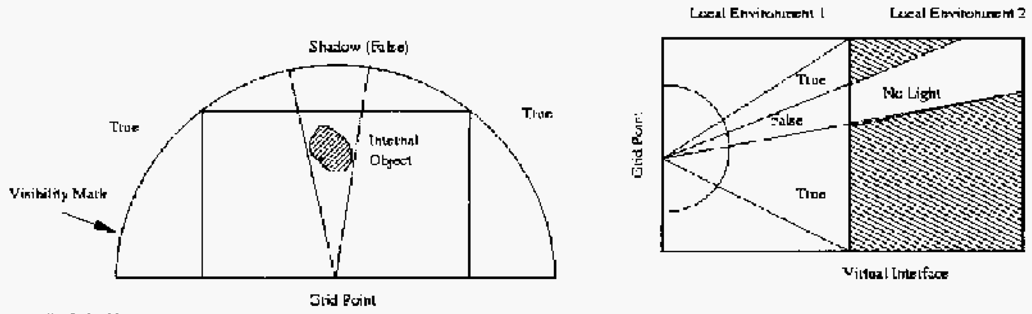


Figure 9 Reusing the shadow information from the green sweep in the blue sweep.



•Initialize

•Choose a grid point on the largest surface (source) and shoot a ray through the interior surfaces belonging to that processor

•Send Visibility Masks to neighbors

•Read sources arriving from other processor, put them in queue

•Termination

Repeat Hierarchically based on number of surface nodes

Figure 10 Visibility Mask and Visibility Interface

G.2 Grouping elements or defining patches:

The industry partners have expressed interest in more user-friendly ways of grouping elements or defining patches to reduce computational expense. Currently, in commercial software only has provisions to prescribe the patches manually and this process can be very tedious and error prone. Here is a methodology proposed to address that:

- Cannot be done apriori (has to have built-in information regarding shadows and topography of the complex geometry)
- Set of user controllable rules to determine the patches
    - Determine the most contributing view factor
    - Look at neighboring elements to determine how the view factor varies with respect to the element identified in the previous step
    - Patch nodes on user specified tolerance

The above procedure can be an automated way to arrive at patches. It is also conceivable that if the problem warrants more expensive radiation modeling approaches – the above procedure can be used as a preprocessing step to improve the efficiency of the radiation model by identifying the most contributing elements.

H. References:

1. R. Siegel and J. R. Howell, *Thermal Radiation Heat Transfer, 3d ed.*, Hemisphere Publishing, Washington, D. C., 1992

2. M. F. Modest, *Radiative Heat Transfer*, McGraw-Hill Series in Mechanical Engineering, McGraw-Hill Inc., 1993

3. P. T. Williams, *A Radiation Heat Transfer Model for Thermal Management Applications*, ORNL Draft Report, 1998.

4. A. B. Shapiro, FACET – A Radiation View Factor Computer Code for Axisymmetric, 2D Planar, and 3D Geometries with Shadowing, DOE Report Number UCID—19887, Lawrence Livermore National Laboratory, 1983

5. G. P. Mitalas and D. G. Stephenson, *FORTRAN IV Programs to Calculate Radiant Interchange Factors*, National Research Council of Canada, Division of Building Research, Ottawa, Canada, DBR-25, 1996.

6. B. Arnaldi, Thierry Priol, L. Renambot, X. Pueyo, *Visibility Masks for Solving Complex Radiosity Computations on Multiprocessors*, **23**, pp. 887-897, Parallel Computing, 1997.

Appendix A: Source code of CHADVIEW

```
!**********************************************************************
!                                                  *
!                  CHADVIEW                        *
!                                                  *
! CHADVIEW is based on the view factor code FACET written by       *
! Dr. A. B. Shapiro of Lawrence Livermore National Laboratory.     *
! Ref.: A. B. Shapiro, "FACET - A Radiation View Factor Computer   *
! Code for Axisymmetric, 2D Planar, and 3D Geometries with Shadowing", *
! Methods Development Group, Mechanical Engineering Dept.,         *
! August, 1983, Rpt. UCID-19887.                                   *
!                                                  *
! The following modifications were made to FACET to produce CHADVIEW.  *
! 1  Remove subroutines relating to axisymmetric and 2D planar     *
!    geometries.                                   *
! 2. Remove all subroutines relating to the creation of familied   *
!    direct access files.                          *
! 3. Recast entire code into FORTRAN 90.           *
! 4. Added dynamic memory management.              *
! 5. Added coding from CHAD plus new coding to provide interface   *
!    with CHAD input and mesh files.               *
! 6. Added coding to calculate LU factorization of AMAT using      *
!    subroutines from LAPACK.                      *
! 7. Minor modification to FTW's version and some debugging        *
! 8. Open-MP parallel directives for shared memory parallelization *
! 9. Modified to use built in BLAS for greater efficiency          *
! 10. Completely in sync with 10-98 version of CHAD                *
!                                                  *
! Paul T. Williams, Ph.D., P.E.                    *
! Computational Engineering Section                *
! Computational Physics and Engineering Division   *
! Oak Ridge National Laboratory                    *
! P. O. Box 2008                                   *
! Building 6011, MS 6415                           *
! Oak Ridge, Tennessee 37831-6415                  *
!**********************************************************************
!***  Modified by Sreekanth Pannala ***************************************
!***  Oak Ridge National Laboratory                ***
!***  P. O. Box 2008                               ***
!***  Building 6012, MS 6367                       ***
!***  Oak Ridge, Tennessee 37831-6367              ***
!***                                               ***
!**********************************************************************
!                                                  *
!**********************************************************************
!
!**********************************************************************
    module setreal_h
!**********************************************************************
!   PURPOSE:
!     Set precision for type real variables
!**********************************************************************
    implicit none
!   integer, parameter :: setpr = selected_real_kind(6,37)   ! single
!   integer, parameter :: setpr = selected_real_kind(15,307) ! double
    integer, parameter :: setpr = selected_real_kind(15,307) ! NEW CHAD VERS
!..................................................
    end module setreal_h


    module element_h
!**********************************************************************
```

```
!  PURPOSE:
!    Defines connectivity within an element among vertices,
!    connections, and faces.
!  OUTPUT VARIABLES:
!    Refer to the following figure and definitions for unambiguous
!    definitions of output variables.
!                  7____________6
!                 /|        /|
!                / |       / |
!               /  |      /  |
!             8/__________/5 |
!              | |      | |
!              |        | |
!              | |      | |
!              |        | |
!              | 3|.    ..._|2
!              |        | /
!              | /      | /
!              |        |/
!              |/ . . ..___|/
!              4        1
!          Face No.   Face      Vertices
!          ________   ______    _______
!             1       Left      3-4-8-7
!             2       Right      1-2-6-5
!             3       Front     4-1-5-8
!             4       Back       2-3-7-6
!             5       Bottom      3-2-1-4
!             6       Top        8-5-6-7
!
!    Edge No. Connection   Edge No. Connection   Edge No. Connection
!                                                 -- . . -----------
!      1    1-2         5    1-5        9    5-6
!      2    2-3         6    2-6       10    6-7
!      3    3-4         7    3-7       11    7-8
!      4    4-1         8    4-8       12    8-5
!    Note: Each connection vector in the above table is defined to
!        be originating from the first vertex and pointing
!        towards the second vertex.  For example, connection 1
!        is from vertex 1 to vertex 2.  The median-mesh boundary
!        areas are computed consistently with this convention.
!    ivertf    = Four vertices associated with each face of
!              the element in counterclockwise direction while
!              viewing from the outside of the element.
!              (see the first table).
!.................................................................
!
!
!  Variable indices for do loops and local temporaries.
   integer  ib      ,ib1      ,ib2      ,ib3      &
           if1     ,if2      ,if3      ,if4      , &
           if5     ,if6      ,iface    , &
           iv      ,iv1      ,iv2      ,iv3      , &
           iv4     ,iv5      ,iv6
   save ib,iface
!.................................................................
!  Face variables.
   integer  ivertf   (4,    6)
   save     ivertf
!.................................................................
!  Median-mesh boundary variables.
   integer  ivertb   (2,   *2)
   save     ivertb
!.................................................................
!  Define face variables.
   data ivertf / 3, 4, 8, 7,  1, 2, 6, 5,  4, 1, 5, 8,  2, 3, 7, 6  &
             3, 2, 1, 4,  8, 5, 6, 7/
!  Define median-mesh boundary variables.
   data ivertb /     1, 2,      2, 3,      3, 4,      4, 1,&
                 1, 5,      2, 6,      3, 7,      4, 8,&
```

```
                    5, 6,      6, 7,      7, 8,      8, 5/
!............ .......... ............................ ..........
      end module element_h


!*************************************************************************
      module global_h
!*************************************************************************
!
!     Global variables.
!.......... ..... ........ .. ..... .. .. ... ..... .............
!     Parameters.
      integer   nf_tec    ,nf_mesh    ,nf_in      ,nf_out     ,&
            nf_log     ,nf_abs      ,&
            ntp_free    ,ntp_freeslip,ntp_inflow ,ntp_int     ,&
            ntp_missing ,ntp_noslip  ,ntp_outflow ,ntp_piston  ,&
            ntp_tmpsrc  ,ntp_uvwsrc
      parameter (nf_tec      =12        nf_mesh     =11        ,&
            nf_in      =10        ,nf_out      =14        ,&
            nf_log      =6         ,nf_abs      =25        ,&
            ntp_free    =11        ,ntp_freeslip=21        ,&
            ntp_inflow  =41        ,ntp_int     =0         ,&
            ntp_missing =-1        ,ntp_noslip  =31        ,&
            ntp_outflow =51        ,ntp_piston  =91        &
            ntp_tmpsrc  =71        ,ntp_uvwsrc  =61&
           )
      end module global_h


!*************************************************************************
      module primary_h
!*************************************************************************
      use setreal_h
       real(setpr)        , dimension(3,6) :: cpuio
       integer                           :: numnp,numel,ncp:
       integer                             maxl,maxa,maxb,nblk,ndiv
       integer                           :: jin, jout, jlog, jabs
!
       save cpuio
!
      end module primary_h

      module compar

       include 'mpif.h'

!       Variables to be declared for parallel information. Added by
!       Sreekanth on 10/25/00.

!       myPE - my processor Id (it varies from 0 to nproc-1)
!
!       numPEs - total number of nodes

       integer :: myPE, numPEs

!       mpierr - used for error checking

       INTEGER :: mpierr

       integer :: nodesi, nodesj, nodesk

!       root represents the 'root' processor. For now it is defaulted to
!       zero

       integer :: root

       data root /0/

!      declaration for storing filebasename, e.g. xxxx000.dat
       CHARACTER(len=3) :: fbname
```

```
    end module compar

     module parallel_mpl

!     A module to carry out init, finalize and check for any parallel errors

     use compar
     implicit none

     contains

     subroutine parallel_init()

     integer :: ierr


     call MPI_Init(ierr)
     call MPI_Check( 'parallel_init:MPI_Init ', ierr)

     call MPI_COMM_SIZE( MPI_COMM_WORLD, numPEs, ierr )
     call MPI_Check( 'parallel_init:MPI_Comm_size ', ierr )

     call MPI_COMM_RANK( MPI_COMM_WORLD, myPE, ierr )
     call MPI_Check( 'parallel_init:MPI_Comm_size ', ierr )

     return
     end subroutine parallel_init

     subroutine parallel_fin()

     integer :: ierr

     call MPI_Finalize(ierr)
     call MPI_Check( 'parallel_init:MPI_Finalize ', ierr)

     return
     end subroutine parallel_fin

     subroutine MPI_Check( msg, ierr )
     character(len=*),intent(in) :: msg
     integer, intent(in) :: ierr

     character(len=512) :: errmsg
     integer :: resultlen

     if (ierr .ne. MPI_SUCCESS ) then
          call MPI_Error_string( ierr, errmsg, resultlen )
          print*, 'Error: ', msg
          print*, errmsg(1:resultlen)
          stop '** ERROR ** '
     endif

     return
     end subroutine MPI_Check


     end module parallel_mpi
!*******************************************************************************
    program CHADVIEW
!*******************************************************************************
!   PURPOSE:
!     A computer program that will read in CHAD input and mesh files
!     and calculate geometric view factors for future use by the
!     CHAD radiation heat transfer model.
!   INPUT ARGUMENTS:
!     Not applicable, main program.
!   OUTPUT ARGUMENTS:
```

```
!     Not applicable, main program.
!*****************************************************************************
    use setreal_h
    use global_h
    use compar
    use parallel_mpi
!                                        ------------------------------------
!...........................................................................
!   Problem size.
    integer   nelemax    ,neles      ,nnodemax    ,nnodes
!...........................................................................
    character *120       meshfile    ,run_label
!___________________________________________________________________________
!   Local variables.
    integer   I          ,iargC      ,nargs
    character *24        dstr
    character *120       infile  line
    character *20        outfile
    integer, dimension(16)             :: error
!_________________________________
!   Global variables.
    logical     radiation, shading
    integer     ndiv
    real (setpr) sigma, solar
    save        radiation, shading, sigma, solar
!                 -----------------------------------------------------------
!   Nodal arrays.
    integer,    dimension(:),allocatable :: NODETYPE
    real (setpr),dimension(:),allocatable :: X
    real (setpr),dimension(:),allocatable :: Y
    real (setpr),dimension(:),allocatable :: Z
    real (setpr),dimension(:),allocatable :: EMISSIVITY
    real (setpr),dimension(:),allocatable :: TRANSMISSIVITY
!___________________________________________________________________________
!   Elemental arrays.
    integer, dimension(:),allocatable :: ELETYPE ! ELETYPE(nelemax)
!__________
!   Vertex arrays.
    integer, dimension(:,:),allocatable :: NODES ! NODES(8,nelemax)
    integer, dimension(:,:),allocatable :: nds   ! nds(5,nelemax)
!...........................................................................
!***************************
! *** Interface Prototypes ***
!***************************
    interface
      subroutine check_alloc(string,error nerr,nf_out)
        character*(*) string
        integer nerr,nf_out
        integer error(nerr)
      end subroutine check_alloc
      subroutine check_dealloc(string,error,nerr,nf_out)
        character*(*) string
        integer nerr,nf_out
        integer error(nerr)
      end subroutine check_dealloc
      subroutine fopen_write( fin, u, ilog)
        character*20, intent(in) :: fin
        integer, intent (in)     :: u
        integer, optional        :: ilog
      end subroutine fopen_write
      subroutine read_input(infile,ndiv,meshfile,run_label,radiation,&
        shading,sigma,solar)
        use setreal_h
        integer   ndiv
        logical   radiation   shading
        character **20 infile  ,meshfile    ,run_label
        real  (setpr)   sigma, solar
      end subroutine read_input
      subroutine setup (ntp_free,ntp_freeslip,ntp_inflow,&
              ntp_int,ntp_missing,ntp_noslip,ntp_outflow,&
```

```
            ntp_piston,ntp_tmpsrc,ntp_uvwsrc,shading,ndiv,&
            nelemax,neles,nnodemax,nnodes,meshfile,run_label, &
            nf_tec,nf_mesh,nf_in,nf_out,nf_log,nf_abs,&
            NODETYPE,X,Y,Z,EMISSIVITY,TRANSMISSIVITY,&
            ELETYPE,NODES,nds)
       use setreal_h
       logical   shading
       integer   nf_tec     ,nf_mesh    ,nf_in     ,nf_out    , &
              nf_log     ,nf_abs     ,&
              ntp_free   ,ntp_freeslip,ntp_inflow  ,ntp_int    ,&
              ntp_missing ,ntp_noslip  ,ntp_outflow ,ntp_piston  , &
              ntp_tmpsrc  ,ntp_uvwsrc
        character*120 meshfile    ,run_label
       integer   ndiv       ,nelemax    ,neles      ,nnodemax    , &
              nnodes
       integer   NODETYPE     (nnodemax)
       real (setpr) X          (nnodemax),Y          (nnodemax),&
               Z          (nnodemax),EMISSIVITY  (nnodemax),&
              TRANSMISSIVITY(nnodemax)
       integer   ELETYPE     (nelemax )
       integer   NODES      (8,nelemax ), nds      (5,nelemax)
       end subroutine setup
      subroutine facet90(nnodemax,nelemax,nf_tec,nf_in,nf_out,nf_log,&
               nf_abs,sigma,solar,EMISSIVITY,TRANSMISSIVITY)
        use setreal_h
        integer              :: nnodemax,nelemax
        integer              :: nf_tec,nf_in,nf_out,nf_log,nf_abs
        real (setpr)              :: sigma, solar
        real (setpr), dimension(*) :: EMISSIVITY
        real (setpr), dimension(*) :: TRANSMISSIVITY
      end subroutine facet90
       subroutine version(codename,dstr)
        character *8       codename
        character *24      dstr
       end subroutine version
!
!*************************
     end interface
!*************************
!.................................................
!    ************************
!    Initialize timing utility
!    ************************
     call timing (0)
!    ************************
!    Initialize the program.
!    ************************


!    Initialize MPI & get ranks & total PEs employed
     call parallel_init

!    Generate file basename for LOG files
     i100 = int(myPE/100)
     i10  = int((myPE-i100*100)/10)
     i1   = int((myPE-i100*100-i10*10)/1)

     i100 = i100 + 48
     i10  = i10  + 48
     i1   = i1        + 48

     fbname=char(i100)//char(i10)//char(i1)

!    ************************
!    First: Create the output file.
!    ************************
     outfile  = 'chad_view'//fbname//'.out'
     call fopen_write( outfile, nf_out, ilog=nf_log)
!    **********************************************************************
!    Print code version information and the date and time of the run.
!   ****************************************************************
    dstr=' '
!     *** Date and time (dstr) will be returned by subroutine
!     *** VERSION.  This blank initialization is being used as a
!     *** flag for VERSION indicating that this is the start of the
!     *** program and not the end.
    call version('CHADVIEW',dstr)
!   ****************************************
!   Get program arguments (input file name).
!   ****************************************
    nargs=iargc()
    if(nargs.LE.0) then
      infile ='chad.in'
    elseif(nargs.EQ.1) then
      call getarg(1,infile)
    else
      if(myPE.eq.root) then
      write(nf_log,"(//,13x,&
       '*****Usage: chadview.x [<data file>]*****',//)")
       endif
      stop
    endif
!   ********************************
!   Read the user data file (INFILE).
!   ********************************
    call read_input(infile,ndiv,meshfile,run_label,radiation,&
       shading,sigma,solar)
    if (.NOT.radiation) then
    if(myPE.eq.root) then
    write (nf_log,'(//" RADIATION flag has not been set to .TRUE.",&
           " in CHAD input file: ",a30/ &
           " Job aborted.")') infile
    endif
    write (nf_out,'(//" RADIATION flag has not been set to .TRUE.",&
           " in CHAD input file: ",a30/ &
           " Job aborted.")') infile
     stop
    endif

!   ********************************************
!   Read the user problem size (from meshfile).
!   ********************************************

!   *******************
!   Open the mesh file.
!   *******************
    open(nf_mesh,file=meshfile,action='read',status='old',iostat=ios)
    call check_ios ('read_mesh',meshfile,ios,nf_out)

      read(nf_mesh,"(a)") line
      read(line,*) nelemax,neles,nnodemax,nnodes

    rewind(nf_mesh)
    close(nf_mesh)

!   ********************
!   Allocate HEAP memory.
!   ********************
     allocate ( NODETYPE  (nnodemax),     stat=error(1)  )
     allocate ( X         (nnodemax),     stat=error(2)  )
     allocate ( Y         (nnodemax),     stat=error(3)  )
     allocate ( Z         (nnodemax),     stat=error(4)  )
     allocate ( EMISSIVITY(nnodemax),     stat=error(5)  )
     allocate ( TRANSMISSIVITY(nnodemax), stat=error(6)  )
     allocate ( ELETYPE   ( nelemax),     stat=error(7)  )
     allocate ( NODES     (8,nelemax),    stat=error(8)  )
     allocate ( nds       (5,nelemax),    stat=error(9)  )
!.....is there sufficient memory to solve the problem?
     call check_alloc('CHADVIEW',error,9,nf_out)
!   ****************************************************************
!    Now that we know the problem size, call SETUP to create the
!    input file for the FACET90 program.
!    ************************************************************
     call setup(ntp_free,ntp_freeslip,ntp_inflow,&
            ntp_int,ntp_missing,ntp_noslip,ntp_outflow,&
            ntp_piston,ntp_tmpsrc,ntp_uvwsrc,shading,ndiv,&
            nelemax,neles,nnodemax,nnodes,meshfile,run_label, &
            nf_tec,nf_mesh,nf_in,nf_out,nf_log,nf_abs,&
            NODETYPE,X,Y,Z,EMISSIVITY,TRANSMISSIVITY,&
            ELETYPE,NODES,nds)
!    ***********************
!    Deallocate HEAP memory
!    ***********************
      deallocate ( NODETYPE,    stat=error(1)  )
      deallocate ( X        ,    stat=error(2)  )
      deallocate ( Y        ,    stat=error(3)  )
      deallocate ( Z        ,    stat=error(4)  )
      deallocate ( ELETYPE ,    stat=error(5)  )
      deallocate ( NODES         stat=error(6)  )
      deallocate ( nds      ,    stat=error(7)  )
!.....deallocation successful?
      call check_dealloc('CHADVIEW',error,7,nf_out)
!    ************************************************
!    Calculate geomtric view factors for 3D enclosure
!    ************************************************
     call facet90(nnodemax,nelemax,nf_tec,nf_in,nf_out,nf_log,nf_abs,&
             sigma,solar,EMISSIVITY,TRANSMISSIVITY)
!    *********************************
!    Deallocate remaining HEAP memory
!    *********************************
     deallocate ( EMISSIVITY    ,   stat=error(1)  )
     deallocate ( TRANSMISSIVITY,   stat=error(2)  )
     call check_dealloc('CHADVIEW',error,2,nf_out)
!    ***********************************************
!    Print the date and time of program termination.
!    ***********************************************
     call version('CHADVIEW',dstr)
!    ***********************************************
!    Close the output file and exit the program
!    ***********************************************
     close (unit=nf_out,status='keep')

!    Finalize and terminate MPI
     call parallel_fin

     stop
     end program CHADVIEW


     subroutine check_alloc(string,error,nerr,nf_out)
!*************************************************************************c
!    PURPOSE:
!  Check for errors in HEAP memory allocation
!*************************************************************************c
     implicit none
     character*(*) string
     character*5  :: a_format = "(aNN)"
     logical lerr
     integer nerr,nf_out,i
     integer error(nerr)
!
     lerr = .FALSE.
     write (a_format(3:4), fmt="(i2.2)") len_trim(string)
     loop_error: do i=1,nerr
       if ( error(i) .EQ. 0 ) cycle loop_error
       write (nf_out, advance='NO', &
            fmt="(/' Allocation error! in ')")
       write (nf_out,fmt=a_format) string
       write (nf_out,fmt="(' Error(',i2,')= ',i4)") i,error(i)
       write (nf_out,&
           fmt="(' HEAP memory allocation unsuccessful!'/)")
       lerr = .TRUE.
     end do loop_error
     if (lerr) then
       write (nf_out,fmt="(' Execution aborted!')")
       stop
     endif
     write (nf_out, advance='NO', &
           fmt="(//' HEAP memory allocation successful in ')")
     write (nf_out,fmt=a_format) string
!
     end subroutine check_alloc



     subroutine check_dealloc(string,error,nerr,nf_out)
!**********************************************************************c
!    PURPOSE:
! Check for errors in HEAP memory deallocation
!**********************************************************************c
     implicit none
     character*(*) string
     character*5 :: a_format = "(aNN)"
     logical lerr
     integer nerr,nf_out,i
     integer error(nerr)
!
     lerr = .FALSE.
     write (a_format(3:4), fmt="(i2.2)") len_trim(string)
     loop_error: do i=1,nerr
       if ( error(i) .EQ. 0 ) cycle loop_error
       write (nf_out, advance='NO', &
            fmt="(/' Deallocation error! in ')")
       write (nf_out,fmt=a_format) string
       write (nf_out,fmt="(' Error(',i2,')=',i4)") i,error(i)
       write (nf_out,  &
            fmt="(' HEAP memory deallocation unsuccessful!'//)")
       lerr = .TRUE.
     end do loop_error
     if (lerr) then
       write (nf_out,fmt="(' Execution aborted!')")
       stop
     endif
     write (nf_out, advance='NO', &
           fmt="(//' HEAP memory deallocation successful in ')")
     write (nf_out,fmt=a_format) string
!
     end subroutine check_dealloc



     subroutine check_ios (string1,string2,ios,nf_out)
!**********************************************************************c
!    PURPOSE:
! Check for IOSTAT errors opening files
!**********************************************************************c
     implicit none
     character*(*) string1
     character*(*) string2
     character*5 :: a1_format = "(aNN)"
     character*5 :: a2_format = "(aNN)"
     integer ios,nf_out
!
     if (ios .NE. 0) then
       write (a1_format(3:4), fmt="(i2.2)") len_trim(string1)
       write (a2_format(3:4), fmt="(i2.2)") len_trim(string2)
       write (nf_out, advance='NO' &
            fmt="(//'****IOSTAT ERROR in ')")
       write (nf_out,fmt=a1_format)       string1
       write (nf_out, advance='NO' &
        fmt="(//'Could not open ')")
     write (nf_out,fmt=a2_format)       string2
     write (nf_out,fmt="('IOS = ',I6)")  ios
     write (nf_out,fmt="('Execution aborted!')")
     write (nf_out,fmt="(//)")
     stop
    endif
!
    end subroutine check_ios




    subroutine get_unit(nf_unit)
!*********************************************************************c
!   PURPOSE:
!     Find an unused unit number
!*********************************************************************c
    implicit none
    integer nf_unit,i
    logical connected
!
    connected = .TRUE.
    inquire( unit=nf_unit,opened=connected)
    unit_loop : do i=1,73
     if( .NOT.connected ) EXIT unit_loop
     nf_unit = nf_unit + 1
     inquire( unit=nf_unit,opened=connected)
    end do unit_loop
!
    end subroutine get_unit


    subroutine setup(ntp_free,ntp_freeslip,ntp_inflow,  &
            ntp_int,ntp_missing,ntp_noslip,ntp_outflow,  &
            ntp_piston,ntp_tmpsrc,ntp_uvwsrc,shading,ndiv,  &
            nelemax,neles,nnodemax,nnodes,meshfile,run_label, &
            nf_tec,nf_mesh,nf_in,nf_out,nf_log,nf_abs, &
            NODETYPE,X,Y,Z,EMISSIVITY,TRANSMISSIVITY, &
            ELETYPE,NODES,nds)
!*********************************************************************c
!   PURPOSE:
! Create an input file for the FACET90 code based on the CHAD input file.
!*********************************************************************c
!______________________________________________________________________________
    use setreal_h
    use element_h
    use compar
!..............................................................
! ***************************
! *** Interface Prototypes ***
! ***************************
    interface
    subroutine read_mesh(nf_mesh,nf_log,meshfile,nelemax,neles,  &
                  nnodemax,nnodes,NODETYPE,  &
                  X,Y,Z,ELETYPE,NODES,EMISSIVITY,  &
                  TRANSMISSIVITY)
     use setreal_h
     integer    nf_mesh    ,nf_log     , &
            nelemax    ,neles      ,nnodemax    ,nnodes
     character *120        meshfile
     integer    NODETYPE    (nnodemax)
     real (setpr) X         (nnodemax),Y          (nnodemax), &
             Z          (nnodemax),EMISSIVITY  (nnodemax), &
            TRANSMISSIVITY(nnodemax)
     integer    ELETYPE     (nelemax )
     integer    NODES       (8,nelemax )
    end subroutine read_mesh
    subroutine fopen_write( fn, u, ilog)
```

```
      character*20 intent(in) :: fin
      integer, intent (in)    :: u
      integer, optional       :: ilog
     end subroutine fopen_write
!**************************
     end interface
!**************************
!__________________________________________________________________________
!    Global variables (scalars and serial arrays) being passed
!    through the argument list.
!.........................................................
!    Parameters.
     logical   shading
     integer   nf_tec     ,nf_mesh    ,nf_in      ,nf_out      &
              nf_log     ,nf_abs     , &
              ntp_free   ,ntp_freeslip,ntp_inflow ,ntp_int    , &
              ntp_missing ,ntp_noslip ,ntp_outflow ,ntp_piston  , &
              ntp_tmpsrc ,ntp_uvwsrc
!.........................................................
!    Problem size.
     integer   nelemax    ,neles      ,nnodemax    ,nnodes
!.........................................................
     character *120 meshfile, run_label
     character*20   chadviewin,chadtecplot
!.........................................................
!    Local arrays
     logical  cross_flow, window
     integer, dimension(16)              :: error
     integer ndim,nummat,numnp,numel,ndiv,nblk,nrot,icheck,ncpl,ibug
     integer kn0,nmiss,inc,i
     integer                            :: nface,jface,iele
     integer,  dimension(4)             :: iivert
     integer,  dimension(4,6)           :: NFACES
     save numnp,numel,kn0,nmiss,inc,i
!.........................................................
!    Nodal arrays.
     integer, dimension(nnodemax)    :: NODETYPE  ! NODETYPE(nnodemax)
     real (setpr),dimension(nnodemax) :: X        ! X(nnodemax)
     real (setpr),dimension(nnodemax) :: Y        ! Y(nnodemax)
     real (setpr),dimension(nnodemax) :: Z        ! Z(nnodemax)
     real (setpr),dimension(nnodemax) :: EMISSIVITY! EMISS(nnodemax)
     real (setpr),dimension(nnodemax) :: TRANSMISSIVITY !T(nnodemax)
!.........................................................
!    Elemental arrays
     integer, dimension(nelemax)        :: ELETYPE   ! ELETYPE(nelemax)
!.........................................................
!    Vertex arrays.
     integer, dimension(8,nelemax)      :: NODES     ! NODES(8,nelemax)
     integer, dimension(5,nelemax)      :: nds       ! nds(5,nelemax)
!.........................................................
!    *****************
!    Read the mesh data
!    *****************
      call read_mesh(nf_mesh,nf_log,meshfile,nelemax,neles  &
                   nnodemax,nnodes,NODETYPE, &
                   X,Y,Z,ELETYPE,NODES,EMISSIVITY, &
                   TRANSMISSIVITY)
!    ***********************************************************
!    Set EMISSIVITY and TRANSMISSIVITY at cross_flow boundaries
!    ***********************************************************
     where( NODETYPE.EQ.ntp_inflow .OR. NODETYPE.EQ.ntp_outflow)
                    EMISSIVITY     = 1.0
                    TRANSMISSIVITY = 0.0
     end where
!    ******************************
!    Create the CHAD_VIEW input file
!    ******************************
     chadviewin = 'chad_view'//fbname//'.in'
     call fopen_write( chadviewin, nf_in, ilog=nf_log)
!
```

```
!   ****************
!   write title card
!   ****************
    write (nf_in,'(" CHAD_VIEW input file: ",a50)')run_label
!   ***************************
!   initialize some control data
!   ***************************
    ndim   = 3
    nummat = 2
    numnp  = nnodes
    nblk   = 0
    nrot   = 0
    icheck = 0
    ncpl   = 0
    ibug   = 0
    kn0    = 0
    nmiss  = 0
    inc    = 0
!   *********************************************************
!   determine enclosure element definitions: calculate numel
!   *********************************************************
    numel = 0
!   loop over all elements in CHAD mesh
    element_loop : do iele = 1,nelemax
!
     nface = 0
!    loop over 6 faces of each element
     face_loop1: do jface = 1,6
      iivert(:) = NODES( ivertf(:,jface),iele)
!     test each face for boundary faces
      if ( NODETYPE(iivert(1)) .GT. 0   .AND.  &
           NODETYPE(iivert(2)) .GT. 0   .AND.  &
           NODETYPE(iivert(3)) .GT. 0   .AND.  &
           NODETYPE(iivert(4)) .GT. 0)   THEN
       nface = nface + 1
       NFACES(:,nface) = iivert(:)
      endif
     end do face_loop1
!     test for a solid or an interior element
      if ( (nface.EQ.0) .OR. (nface.EQ.6) ) cycle element_loop
!
       face_loop2: do jface = 1,nface
        numel = numel + 1
        cross_flow = .FALSE.
        window     = .FALSE.
        do iface = 1,4
          nds(iface,numel) = NFACES(5-iface,jface)
          if ( NODETYPE(nds(iface,numel)) .EQ. ntp_inflow  &
          .OR. NODETYPE(nds(iface,numel)) .EQ. ntp_outflow )  &
          cross_flow = .TRUE.
          if ( TRANSMISSIVITY(nds(iface,numel)) .GT. 0.0)  &
           window    = .TRUE.
        end do
        nds(5,numel)      = 1   ! standard wall element
        if (cross_flow) then
          nds(5,numel)    = 2   ! cross-flow element
        elseif (window) then
          nummat          = 3
          nds(5,numel)    = 3   ! window element
        endif
       end do face_loop2
!
    end do element_loop
!   ***************************
!   Check for internal shading
!   ***************************
    if (shading) nblk   = numel
!   ******************
!   write control card
!   ******************
```

```
    write (nf_in,140) ndim,nummat,numnp,numel,ndiv,nblk,nrot,icheck, &
              ncpl,ibug
!   ******************
!   write nodal data
!   ******************
    do i=1,numnp
     write (nf_in,160) i,X(i),Y(i),Z(i),kn0
    end do
!   ***********************************
!   write enclosure element definitions
!   ***********************************
    do i=1,numel
     write (nf_in,190) i,(nds(j,i),j=1,5),nmiss,inc
    end do
!   **********************************
!   write blocking surface definitions
!   **********************************
    if (shading) write (nf_in,230) 1,numel-1,1
!   *************************************
!   rewind and close chadview input file
!   *************************************
    rewind (nf_in)
    close (unit=nf_in, status='keep')
!   ******************************
!   Create the TECPLOT input file
!   ******************************
    chadtecplot= 'chad_tecplot'//fbname//'.in'
    call fopen_write( chadtecplot, nf_tec, ilog=nf_log)
!
    write (nf_tec,200) numnp,numel
    do i=1,numnp
     write (nf_tec,210) X(i),Y(i),Z(i),EMISSIVITY(i), &
              TRANSMISSIVITY(i),NODETYPE(i)/10
    end do
    do i=1,numel
     write (nf_tec,220) (nds(j,i),j=1,4)
    end do
!   *************************
!   close TECPLOT input file
!   *************************
    close (unit=nf_tec, status='keep')
!
 140 format (10i6)
 160 format (i6,5x,1p,3e12.5,i6)
 190 format (8i6)
 200 format ('TITLE= ',1h",'CHAD_VIEW ENCLOSURE DEFINITION',1h"/ &
         'VARIABLES= ',1h",'X',1h",',',2x,1h",'Y',1h",',',2x, &
                   1h",'Z',1h",2x,1h",'EMIS',1h",',',2x, &
                   1h",'TRANS',1h",',',2x,1h",'TYPE',1h"/   &
         'ZONE  N=',i6,' E=',i6,' F=FEPOINT, ET=QUADRILATERAL')
 210 format (1p,5(2x,e12.5),0p,2x,i3)
 220 format (4(1x,i6))
 230 format (3i6)
    end subroutine setup




!**********************************************************************c
!***                                                ***c
!***                  subroutine fdate                     ***c
!***                                                ***c
!***    The subroutine fdate creates a time and date string      ***c
!***    using the FORTRAN 90 intrinsic function date_and_time     ***c
!***                                                ***c
!***    Paul T. Williams                                   ***c
!***                                                ***c
!**********************************************************************c
!
```

```
    subroutine fdate (date_time)
    character*24  date_time
    integer      elements(8)
    character*3  months(12)
    data months / 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', &
                 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec' /
!
    call date_and_time ( VALUES=elements )
    invalid: if ( elements(1) .NE. -HUGE(0) ) then
      century: if ( elements(1) .LT. 2000 ) then
                 elements(1) = elements(1) - 1900
             else
                 elements(1) = elements(1) - 2000
      end if century
      write ( date_time,100 ) elements(3), &
                 months ( elements(2) ), &
                       elements(1), elements(5), &
                       elements(6), elements(7)
100    format ( i2.2, 1x, a3, 1x, i2.2, 1x, &
             i2.2, ':' i2.2, ':' i2.2 )
    else invalid
      date_time = ' '
    end if invalid
    end subroutine fdate




    subroutine read_char(var,label,var_default,ifile)
!*********************************************************************c
!   PURPOSE:
!     Reads character variables from the input file.
!   INPUT ARGUMENTS:
!     var        = Variable to be read.
!     label      = Label to seek on the input file for reading VAR.
!     var_default = Default value of the variable.  VAR will be set
!                 to this value if not found on the input file.
!     ifile      = Input file unit number.
!   OUTPUT ARGUMENTS:
!     var        = Set to the desired value.
!*********************************************************************c
!
    use setreal_h
    use global_h
    use compar
!.............................................................
! ****************************
! *** Interface Prototypes ***
! ****************************
    interface
      subroutine read_file(line,ibeg,error,label,ifile)
        logical    error
        integer    ibeg       ,ifile
        character  *(*)        line        ,label
      end subroutine read_file
! ****************************
    end interface
! ****************************
!
!____________________________________________________________
!   Global variables being passed through the argument list.
    integer    ifile
    character  *(*)        var        ,label       ,var_default
!________________________________________________________________________
!   Local variables.
    logical    error
    integer    ibeg
    character  *120        line       ,string
!________________________________________________________________________
!*********************************************************************c
```

```
!     ******************************************************************
!     Read data file until the line beginning with label is found
!     Return entire line in variable LINE.  ERROR is .TRUE. if label was
!     not found in data file
      call read_file(line,ibeg,error,label,ifile)
!     ******************************************************************
!     If LABEL was not found, set VAR to its default; otherwise
!     extract its value from LINE.
      if(error) then
        var=var_default
        write(nf_out,"(1x,a20,19x,a37,' (default)')") label,var
      else
        call read_string(string,ibeg,error,line)
        if(error) then
          var=var_default
          write(nf_out,"(1x,a20,19x,a37,' (default)')") label,var
        else
          var=string
          write(nf_out,"(1x,a20,19x,a)") label,var
        endif
      endif
!     ******************************************************************
      end subroutine read_char




      subroutine read_file(line,ibeg,error,label,ifile)
!*********************************************************************c
!     PURPOSE:
!       Reads the entire file to search for LABEL.
!     INPUT ARGUMENTS:
!       label       = Label to seek on the input file.
!       ifile       = Input file unit number.
!     OUTPUT ARGUMENTS:
!       line        = Line from input file containing LABEL.
!       ibeg        = Position of space in line following LABEL.
!       error       = Error flag.
!*********************************************************************c
      implicit none
!.....................................................................
! ***************************
! *** Interface Prototypes ***
! ***************************
      interface
        subroutine read_nxtlne(line,end,ifile)
          logical   end
          integer   ifile
          character *(*)       line
        end subroutine read_nxtlne
        subroutine read_label(label,ibeg,error,line)
          logical   error
          integer   ibeg
          character *(*)       line       ,label
        end subroutine read_label
! ***************************
      end interface
! ***************************
!---------------------------------------------------------------------
!     Global variables being passed through the argument list.
      logical   error
      integer   ibeg       ,ifile
      character *(*)       line       ,label
!---------------------------------------------------------------------
!     Local variables.
      logical   looking
      character *120       labell
!---------------------------------------------------------------------
!     ******************************************************************
!   Rewind the input file and initialize some data.
    rewind(ifile)
    error  =.FALSE.
    looking=.TRUE.
!   *****************************************************************
!
!   Look until LABEL is found or EOF is reached.
    do while(looking)
      ibeg = 1
      call read_nxtlne(line,error,ifile)
      if(error) then
        looking=.FALSE.
      else
        call read_label(labell,ibeg,error,line)
        if(labell.EQ.label) looking = .FALSE.
      endif
    enddo
!   *****************************************************************
    end subroutine read_file




    subroutine read_input(infile,ndiv,meshfile,run_label,radiation,  &
       shading,sigma,solar)

!*******************************************************************c
!   PURPOSE:
!     Reads the user input file.
!   INPUT ARGUMENTS:
!     infile     = File name of the user input data (entered on the
!                  command line).
!     nf_in      = User-input file (INFILE) unit number.
!   OUTPUT ARGUMENTS:
!     ndiv       = number of subdivisions for radiation surfaces
!     meshfile   = File name for the mesh data.
!     run_label  = Character string for case title
!     radiation  = logical flag to signal radiation heat transfer
!     shading    = logical flag to signal if internal shading exists
!     sigma      = Stefan-Boltzmann constant in correct units
!     solar      = solar insolation for windows in correct units
!*******************************************************************c
    use setreal_h
    use global_h
    use compar
!...................................................................
! ***************************
! *** Interface Prototypes ***
! ***************************
    interface
      subroutine check_ios (string1,string2,ios,nf_out)
        character*(*) string1
        character*(*) string2
        integer ios,nf_out
      end subroutine check_ios
      subroutine read_char(var,label,var_default,ifile)
        integer   ifile
        character *(*)  var  ,label   ,var_default
      end subroutine read_char
      subroutine read_int(var,label,var_default,ifile)
        integer   var        ,ifile       ,var_default
        character  *(*)        label
      end subroutine read_int
      subroutine read_logicl(var,label,var_default,ifile)
        integer   ifile
        logical var, var_default
        character  *(*)  label
      end subroutine read_logicl
      subroutine read_real(var,label,var_default,ifile)
        use setreal_h
      integer   ifile
      real (setpr)  var, var_default
      character  *(*)  label
     end subroutine read  real
!****************************
    end interface
!****************************
!___________________________________________________________________________
!   Global variables being passed through the argument list.
    logical   radiation  ,shading
    integer   ndiv
    character  *(*)       infile     ,meshfile    ,run_label
    real (setpr)      sigma, solar
!___________________________________________________________________________
!   Local variables.
    logical   restart  default        ,radiation  default       , &
              shading  default
    integer   nelemax_default        ,neles_default          , &
              nnodemax  default       .nnodes  default         , &
              ndiv  default
    integer   ios
    character  *120 meshfile_default   ,runlabel_default
    real (setpr)      sigma_default     , solar_default
!___________________________________________________________________________
!   ****************************
!   Initialize some variables.
!   ****************************
!   Define default values.
    ndiv_default     =5           ! default number of subdivisions
    meshfile_default = chad.msh'
    runlabel_default =       '
    radiation_default=.FALSE.
    shading_default  =.FALSE.
!   Stefan-Boltzmann constant = 5.67051E-08 W/m**2-K**4
!   Stefan-Boltzmann constant = 5.67051E-05 ergs/s-cm**2-K**4
    sigma_default    = 5.67051E-08  ! default Stefan_Boltzmann constant (MKS)
!   sigma_default    = 5.67051E-05  ! default Stefan_Boltzmann constant (cgs)
    solar_default    = 0.0         ! default solar insolation
!   *****************
!   Open input file.
!   *****************
    open(nf_in,file=infile,action='read',status='old',iostat=ios)
    call check_ios ('read_input',infile,ios,nf_out)
!   **********************************************************************
!   Write a marker indicating the beginning of input-file reading phase.
!   **********************************************************************
    write(*,"(/,1x,71('  '))")
!   ************************************************
!   Read one variable at a time from the input file.
!   ************************************************
    call read_char  (run_label,'RUN_LABEL',runlabel_default ,nf_in)
    call read_char  (meshfile ,'MESHFILE' ,meshfile_default ,nf_in)
    call read_logic(radiation,'RADIATION',radiation_default,nf_in)
    call read_logic(shading  ,'SHADING'  ,shading_default  ,nf_in)
    call read_int   (ndiv     ,'NDIVRAD'  ,ndiv_default     ,nf_in)
    call read_real  (sigma    ,'SIGMA'    ,sigma_default    ,nf_in)
    call read_real  (solar    ,'SOLAR     ,solar_default    nf_in)
!   *****************
!   Close input file
!   *****************
    close(nf  in,status='keep')
!   *******************************************************************
!   Write a marker indicating the end of input-file reading phase.
!   *******************************************************************
    write(*,"(/,1x,71('_'))")
!   *******************************************************************
    end subroutine read_input
    subroutine read_int(var,label,var_default,ifile)
!*********************************************************************c
!   PURPOSE:
!     Reads integer variables from the input file.
!   INPUT ARGUMENTS:
!     var        = Variable to be read.
!     label      = Label to seek on the input file for reading VAR.
!     var_default = Default value of the variable.  VAR will be set
!                  to this value if not found on the input file
!     ifile      = Input file unit number.
!   OUTPUT ARGUMENTS:
!     var        = Set to the desired value.
!*********************************************************************c
    use setreal_h
    use global_h
    use compar
!.............................................................
! ****************************
! *** Interface Prototypes ***
! ****************************
    interface
      subroutine read_file(line,ibeg,error,label,ifile)
        logical    error
        integer    ibeg       ,ifile
        character  *(*)       line        label
      end subroutine read_file
      subroutine read_label(label,ibeg,error,line)
        logical    error
        integer    ibeg
        character  *(*) line,label
      end subroutine read_label
! ****************************
    end interface
! ****************************
!____________________________________________________________________________
!   Global variables being passed through the argument list.
    integer   var        ,ifile      ,var_default
    character *(*)       label
!____________________________________________________________________________
!   Local variables
    logical   error
    integer   ibeg
    character *120       line       ,string
!
!   *****************************************************************
!   Read data file until the line beginning with label is found.
!   Return entire line in variable LINE   ERROR is .TRUE. if label was
!   not found in data file.
    call read_file(line,ibeg,error,label,ifile)
!   *****************************************************************
!   If LABEL was not found, set VAR to its default; otherwise
!   extract its value from LINE.
    if(error) then
      var=var_default
      write(nf_out,"(1x,a20,2x,i37,' (default)')") label,var
    else
      call read_label(string,ibeg,error,line)
      if(error) then
        var=var_default
        write(nf_out,"(1x,a20,2x,i37,' (default)')") label,var
      else
        read(string,*) var
        write(nf_out,"(1x,a20,2x,i37)") label,var
      endif
    endif
!   *****************************************************************
    end subroutine read_int
    subroutine read_label(label,ibeg,error,line)
!************************************************************************c
!   PURPOSE:
!     Extracts the LABEL (containing no spaces) from the input file.
!   INPUT ARGUMENTS:
!     ibeg       = Position from where to start the search.
!     line       = Line from input file containing LABEL.
!   OUTPUT ARGUMENTS:
!     label      = Label to seek on the input file.
!     ibeg       = Position of space in line following LABEL.
!     error      = Error flag.
!************************************************************************c
    implicit none
!_______________________________________________________________________
!   Global variables being passed through the argument list.
    logical    error
    integer    ibeg
    character  *(*)       line        ,label
!_______________________________________________________________________
!   Local variables.
    integer    ich        ,linelen
!_______________________________________________________________________
!   *********************
!   Initialize some data.
!   *********************
    error  =.TRUE.
    label  =''
    linelen=len(line)
!   ****************************
!   Find the beginning of string.
!   ****************************
    do ich=ibeg,linelen
      if(line(ich:ich).NE.' ') then
        error=.FALSE.
        exit
      endif
    enddo
    ibeg=ich
!   **********************
!   Find the end of string.
!   **********************
    do ich=ibeg,linelen
      if(line(ich:ich).EQ.' ') exit
    enddo
!   ***************************************************
!   Extract LABEL and the position of space following it.
!   ***************************************************
    label(1:ich-ibeg)=line(ibeg:ich-1)
    ibeg=ich
!   *****************************************************************
    end subroutine read_label



    subroutine read_logic(var,label,var_default,ifile)
!************************************************************************c
!   PURPOSE:
!     Reads logical variables from the input file.
!   INPUT ARGUMENTS:
!     var        = Variable to be read.
!     label      = Label to seek on the input file for reading VAR.
!     var_default = Default value of the variable.  VAR will be set
!                   to this value if not found on the input file.
!     ifile      = Input file unit number.
!   OUTPUT ARGUMENTS:
!     var         = Set to the desired value.
!**********************************************************************c
    use setreal_h
    use global_h
    use compar
!........................................................................
! ****************************
! *** Interface Prototypes ***
! ****************************
    interface
      subroutine read_file(line,ibeg,error,label,ifile)
        logical    error
        integer    ibeg        ,ifile
        character  *(*)        line        ,label
      end subroutine read_file
      subroutine read_label(label,ibeg,error,line)
        logical    error
        integer    ibeg
        character  *(*)  line,label
      end subroutine read_label
! ****************************
    end interface
! ****************************
!
!_____________________________________________________________________________
!   Global variables being passed through the argument list.
    logical    var        ,var_default
    integer    ifile
    character  *(*)        label
!_____________________________________________________________________________
!   Local variables.
    logical    error
    integer    ibeg
    character  *120        line        ,string
!........................................................................
!   ******************************************************************
!   Read data file until the line beginning with label is found.
!   Return entire line in variable LINE.  ERROR is .TRUE. if label was
!   not found in data file.
    call read_file(line,ibeg,error,label,ifile)
!   ******************************************************************
!   If LABEL was not found, set VAR to its default; otherwise
!   extract its value from LINE.
    if(error) then
      var=var_default
      write(nf_out,"(1x,a20,2x,l37,' (default)')") label,var
    else
      call read_label(string,ibeg,error,line)
      if(error) then
        var=var_default
        write(nf_out,"(1x,a20,2x,l37,' (default)')") label,var
      else
        read(string,*) var
        write(nf_out,"(1x,a20,2x,l37)") label,var
      endif
    endif
!   ******************************************************************
    end subroutine read_logicl




    subroutine read_mesh(nf_mesh,nf_log,meshfile,nelemax,neles, &
                 nnodemax,nnodes,NODETYPE,X,Y,Z, &
                 ELETYPE,NODES,EMISSIVITY,TRANSMISSIVITY)
!**********************************************************************c
!   PURPOSE:
!     Reads mesh data from MESHFILE.
!   INPUT ARGUMENTS:
!     nf_mesh       = Mesh-input file (MESHFILE) unit number.
!     meshfile      = File name for the mesh data.
!     nelemax       = Maximum value of an element number in the
!                     problem.
!     neles         = Total number of elements.
!     nnodemax      = Maximum value of a node number in the problem.
!     nnodes        = Total number of nodes.
!   OUTPUT ARGUMENTS:
!     NODETYPE      = Node type (positive value implies a real node).
!     X             = X-coordinate of node.
!     Y             = Y-coordinate of node.
!     Z             = Z-coordinate of node
!     ELETYPE       = Element type.  Hexahedron=12, tetrahedron=6,
!                     pyramid=8,prism=9,etc.(basically the number
!                     of nonzero-length connections).
!     NODES         = List of nodes associated with an element.
!     EMISSIVITY    = nodal emissivities
!     TRANSMISSIVITY = nodal transmissivities
!*********************************************************************c
!
    use setreal_h
    use element_h
!_______________________________________________________________________
!   Global variables being passed through the argument list.
    integer   nf_mesh    ,nf_log
    character *(*)       meshfile
    integer   NODETYPE(*)
    real (setpr)  X(*),Y(*),Z(*)
    real (setpr)  EMISSIVITY(*),TRANSMISSIVITY(*)
    integer      ELETYPE    (  nelemax )
    integer      NODES      ( 8,nelemax )
!_______________________________________________________________________
!   Local variables.
    integer   i          ,ibb       ,ios       , &
              iw1        ,iw2       ,j         ,nbade     , &
              nelemax    ,neles     ,nnodemax  ,nnodes,  &
              nparents, nconns, nspl
    character *24        name
    logical   FLAGB      (12,nelemax )
!_______________________________________________________________________
!   ******************
!   Open the mesh file.
!   ******************
    open(nf_mesh,file=meshfile,action='read',status='old',iostat=ios)
    call check_ios ('read_mesh',meshfile,ios,nf_out)
!   ************************
!   Read problem dimensions.
!   ************************
    read(nf_mesh,*) nelemax,neles,nnodemax,nnodes,  &
             nparents,nconns,nspl
!   ***************************************
!   Read standard nodal and elemental data.
!   ***************************************
    read(nf_mesh,*) name
    if(name(1:1).NE.'X') then
      write(nf_log,"(//,' *****Error reading X.*****',//)")
      stop
    endif
    read(nf_mesh,*) (X(i),i=1,nnodemax)
    read(nf_mesh,*) name
    if(name(1:1).NE.'Y') then
      write(nf_log,"(//,' *****Error reading Y.*****',//)")
      stop
    endif
    read(nf_mesh,*) (Y(i),i=1,nnodemax)
    read(nf_mesh,*) name
    if(name(1:1).NE.'Z') then
      write(nf_log,"(//,' *****Error reading Z.*****',//)")
      stop
    endif
    read(nf_mesh,*) (Z(i),i=1,nnodemax)
    read(nf_mesh,*) name
    if(name(1 6).NE 'NODESV') then
      write(nf_log,"(//,' *****Error reading NODES.*****',//)")
      stop
    endif
    read(nf_mesh,*) ((NODES(i,j),i=1,8),j=1,nelemax)
!   ****************************************************************
!   Define element type based on foregoing element data.  The element
!   type is defined as the number of nonzero-length edges.  The
!   minimum allowable element type is 6 (a tetrahedron).  If this is
!   not the case, print a message and abort the code.
!   ____________________________________________
!   Define a boundary flag in elements, such that it is .FALSE. at
!   dudded connections
!   ........ .. ......... ..............................................
!   First set the flag to .TRUE. everywhere.
    FLAGB=.TRUE.
!   ............... .. ......................... ............. ...........
!   Set the flag to .FALSE. where a connection has the same node
!   numbers on both sides.
    do ib=1,12
      iv1=ivertb(1,ib)
      iv2=ivertb(2,ib)
      WHERE(NODES(iv1,:).EQ.NODES(iv2,:)) FLAGB(ib,:)=.FALSE.
    enddo
!   .. ................................ ......... .......................
!   Now set the flag to .FALSE. at duplicate connections.  For
!   example, a tetrahedron could be defined by merging nodes 3 and 4,
!   and nodes 5 through 8.  In this case, connection 3-7 is same as
!   connection 4-8.  One of these must be eliminated to achieve
!   uniqueness.
    do ib=1,11
      iv1=ivertb(1,ib)
      iv2=ivertb(2,ib)
      do ibb=ib+1.12
        ivv1=ivertb(1,ibb)
        ivv2=ivertb(2,ibb)
        WHERE((MIN(NODES(iv1 ,:),NODES(iv2 ,:)).EQ.  &
              MIN(NODES(ivv1,:),NODES(ivv2,:)) )  &
             .AND.  &
             (MAX(NODES(iv1 ,:),NODES(iv2 ,:)).EQ.  &
              MAX(NODES(ivv1,:),NODES(ivv2,:)) ))  &
              FLAGB(ibb,:)=.FALSE.
      enddo
    enddo
!   ......................................... ............. ..............
!   Define element type by counting .TRUE. connections in each
!   element.  Make sure all element types are greater than or equal
!   to 6.
    ELETYPE=COUNT(FLAGB DIM=1)
    nbade=count(ELETYPE.LT.6)
    if(nbade.NE.0) then
      write(*,"(//,' *****There are',i6, &
                ' zero-volume elements in MESHFILE.*****',// &
            )" &
          ) nbade
      stop
    endif
!   ..... ...................... . ........ .............. ............. ...
!   __________ . .                                        . . ______________________
!   *************************
!   Read the remaining data.
!   *************************
    NODETYPE_loop : do
     read(nf_mesh,fmt='(a)',end=997) name
     if(name(1:8) .EQ. 'NODETYPE') exit NODETYPE_loop
    end do NODETYPE_loop
    read(nf_mesh,*) (NODETYPE(i),i=1,nnodemax)
```

```
!
    EMISSIVITY_loop : do
      read(nf_mesh,fmt='(a)',end=998) name
      if(name(1:10) .EQ. 'EMISSIVITY') exit EMISSIVITY_loop
    end do EMISSIVITY_loop
    read(nf_mesh,*) (EMISSIVITY(i),i=1,nnodemax)
!
    TRANSMISSIVITY_loop : do
      read(nf_mesh,fmt='(a)',end=999) name
      if(name(1:14) .EQ. 'TRANSMISSIVITY') exit TRANSMISSIVITY_loop
    end do TRANSMISSIVITY_loop
    read(nf_mesh,*) (TRANSMISSIVITY(i),i=1,nnodemax)
!
!   ********************
!   Close the mesh file.
!   ********************
    close(nf_mesh)
!   ******************************************************************
    return
 997  continue
      write(nf_log,"(//,' *****Error reading NODETYPE.*****',//)")
      stop
 998  continue
      write(nf_log,"(//,' *****Error reading EMISSIVITY.*****',//)")
      stop
 999  continue
    write(nf_log,"(//,' *****Error reading TRANSMISSIVITY.*****',//)")
      stop
      return
    end subroutine read_mesh




    subroutine read_nxtline(line,end,ifile)
!*******************************************************************c
!   PURPOSE
!     Reads the next active line from the input file by skipping
!     comments and blank lines.
!   INPUT ARGUMENTS:
!     ifile      = Input file unit number
!   OUTPUT ARGUMENTS:
!     line       = Line from input file
!     end        = Flag for end-of-file.
!*******************************************************************c
    implicit none
!
!   Global variables being passed through the argument list.
    logical    end
    integer    ifile
    character  *(*)       line
!______________________________________________________________________
!   Local variables
!______________________________________________________________________
!   *********************
!   Initialize some data.
!   *********************
    line=''
    end =.TRUE.
!   ****************************
!   Search for a relevant line.
!   ****************************
    do while(end)
      read (ifile,"(a)",end=999) line
      if(line(1:1).NE.'#'.AND.line(1:1).NE.' ') end=.FALSE.
    enddo
 999 continue
    end subroutine read_nxtline
```

```
      subroutine read_real(var,label,var_default,ifile)
!*************************************************************************c
!     PURPOSE:
!       Reads floating-point variables from the input file.
!     INPUT ARGUMENTS:
!       var         = Variable to be read.
!       label       = Label to seek on the input file for reading VAR.
!       var_default = Default value of the variable.  VAR will be set
!                     to this value if not found on the input file
!       ifile       = Input file unit number.
!     OUTPUT ARGUMENTS:
!       var         = Set to the desired value.
!*************************************************************************c
      use setreal_h
      use global_h
      use compar
!..............................................................
! ****************************
! *** Interface Prototypes ***
! ****************************
      interface
        subroutine read_file(line,ibeg,error,label,ifile)
          logical    error
          integer    ibeg       ,ifile
          character  *(*)       line       ,label
        end subroutine read_file
        subroutine read_label(label,ibeg,error,line)
          logical    error
          integer    ibeg
          character  *(*)  line,label
        end subroutine read_label
! ****************************
      end interface
! ****************************
!_____________________________________________________________________________
!     Global variables being passed through the argument list.
      integer          ifile
      real (setpr)     var        ,var_default
      character  *(*)  label
!________________________________________
!     Local variables.
      logical    error
      integer    ibeg
      character  *120       line       ,string
!_____________________________________________________________________________
!     *******************************************************************
!     Read data file until the line beginning with label is found.
!     Return entire line in variable LINE.  ERROR is .true. if label was
!     not found in data file.
      call read_file(line,ibeg,error,label,ifile)
!     *******************************************************************
!     If LABEL was not found, set VAR to its default; otherwise
!     extract its value from LINE.
      if(error) then
        var=var_default
        write(nf_out,"(1x,a20,2x,1pe37.14,' (default)')") label,var
      else
        call read_label(string,ibeg,error,line)
        if(error) then
          var=var_default
          write(nf_out,"(1x,a20,2x,1pe37.14,' (default)')") label,var
        else
          read(string,*) var
          write(nf_out,"(1x,a20,2x,1pe37.14)") label,var
        endif
      endif
```

```
!   ******************************************************************
    end



    subroutine read_string(string,ibeg,error,line)
!*********************************************************************c
!   PURPOSE:
!     Reads a string from input line and places it in a character
!     variable.
!   INPUT ARGUMENTS:
!     ibeg      = Position from where to start the search.
!     line      = Input line.
!   OUTPUT ARGUMENTS:
!     string    = Desired data placed in the character string.
!     ibeg      = Position of space in line following the string
!     error     = Error flag.
!*********************************************************************c
    implicit none
!   ------------------------------------------------------------------
!   Global variables being passed through the argument list.
    logical   error
    integer   ibeg
    character*(*)       string     ,line
!---------------------------------------------------------------------
!   Local variables.
    integer   ich       ,iend       ,linelen
!---------------------------------------------------------------------
!   ******************************************************************
!   Initialize some data.
    error  = .TRUE.
    string =''
    linelen=len(line)
!   ***********************************
!   Find the beginning of input string.
!   ***********************************
    do ich=ibeg,linelen
      if(line(ich:ich).NE.' ') then
        error=.FALSE.
        exit
      endif
    enddo
    if(line(ich:ich).EQ.'"'.OR.line(ich:ich).EQ."'") ich=ich+1
    ibeg = ich
!   ******************************************************************
!   Find the end of string by starting at end and moving backwards.
!   ******************************************************************
    do ich=linelen,ibeg,-1
      if(line(ich:ich).NE.' ') exit
    enddo
    if(line(ich:ich).EQ.'"'.OR.line(ich:ich).EQ."'") ich=ich-1
    iend=ich
!   ******************************************************
!   Extract the string and position of space following it.
!   ******************************************************
    string(1:iend-ibeg+1)=line(ibeg:iend)
    ibeg=iend+1
!   ******************************************************************
    end subroutine read_string



    subroutine version(codename,dstr)
!*********************************************************************c
!   PURPOSE:
!     Prints version information and the date and time of the run.
!   INPUT ARGUMENTS:
!     codename   = Name of the code.
!     dstr       = Date and time of the run.  As an input argument,
!               this is a flag that indicates it is the start
!               of the run if dstr is blank.  A nonblank value
!               Indicates that the run is successfully completed.
!   OUTPUT ARGUMENTS:
!     dstr      = Date and time of the run.
!*****************************************************************************c
    use setreal_h
    use global_h
    use compar
!............................................................
!***************************
!*** Interface Prototypes ***
!***************************
    interface
      subroutine fdate (date_time)
        character*24  date_time
      end subroutine fdate
!***************************
    end interface
!***************************
!____________________________________________________________________________
!   Global variables being passed through the argument list.
    character *8        codename
    character *(*)      dstr
!   ..........________________________________________________________________
!   Local variables.
    character *8        codedate   ,codemach   ,codetime   , &
             codeuser   ,rundate    ,runstatus  ,runtime
!____________________________________________________________________________
!   **********************
!   Initialize some data.
!   **********************
    if(dstr.EQ.' ') then
      runstatus='start'
    else
      runstatus='finish'
    endif
!   **********************************
!   Determine the run-time information
!   **********************************
    call fdate(dstr)
    codetime='10:30:00'
    codedate='10/27/00'
    codeuser='pannala '
    codemach='COMPAQ'
!   *****************************************************
!   Print version and run-time (start/finish) information.
!   *****************************************************
    if(runstatus.EQ.'start') then

      write(nf_out,  &
        '(//,13x,"************************",  &
          /,13x,"*                        *",  &
          /,13x,"*    This version of ",a8," was made:      *",  &
          /,13x,"*        on date       ",a8,"         *",  &
          /,13x,"*        at time       ",a8,"         *",  &
          /,13x,"*        by user       ",a8,"         *",  &
          /,13x,"*        on machine    ",a8,"         *",  &
          /,13x,"*                        *",  &
          /,13x,"*    This run is being made on:            *",  &
          /,13x,"*        ",a24,"        *",  &
          /,13x,"*                        *",  &
          /,13x,"************************",  &
         //  &
        )'  &
         ) codename,codedate,codetime,codeuser,codemach,dstr

      if(myPE.eq.root) then
      write(nf_log,  &
        '(//,13x,"************************",  &
          /,13x,"*                                    *",  &
          /,13x,"*    This version of ",a8," was made:     *",  &
          /,13x,"*         on date       ",a8,"          *",  &
          /,13x,"*         at time       ",a8,"          *",  &
          /,13x,"*         by user       ",a8,"          *",  &
          /,13x,"*         on machine    ",a8,"          *",  &
          /,13x,"*                                    *",  &
          /,13x,"*    This run is being made on:           *",  &
          /,13x,"*         ",a24,"          *",  &
          /,13x,"*                                    *",  &
          /,13x,"************************************",  &
          //  &
         )'  &
          ) codename,codedate,codetime,codeuser,codemach,dstr
        endif
        else

        write(nf_out,  &
          '(//,13x,"************************************",  &
          /,13x,"*                                    *",  &
          /,13x,"*    Run successfully finished on:        *",  &
          /,13x,"*         ",a24,"          *",  &
          /,13x,"*                                    *",  &
          /,13x,"************************************",  &
          //  &
         )'  &
          ) dstr

        if(myPE.eq.root) then
        write(nf_log,  &
          '(//,13x,"************************************",  &
          /,13x,"*                                    *",  &
          /,13x,"*    Run successfully finished on:        *",  &
          /,13x,"*         ",a24,"          *",  &
          /,13x,"*                                    *",  &
          /,13x,"************************************",  &
          //  &
         )'  &
          ) dstr
        endif
        endif
!     ***********************************************************************
      end

!***********************************************************************
      subroutine facet90(nnodemax,nelemax,nf_tec,nf_in,nf_out,nf_log, &
                 nf_abs,sigma,solar,EMISSIVITY,TRANSMISSIVITY)
!***********************************************************************
      use setreal_h
      use primary_h
      use compar
!***********************************************************************
      integer nnodemax,nelemax
      integer nf_tec,nf_in,nf_out,nf_log,nf_abs
      integer, dimension(:,:), allocatable :: nds   ! nds(6,numel)
      integer, dimension(:,:), allocatable :: nocpl ! nocpl(2,ncpl+1)
      integer, dimension(:,:), allocatable :: nia   ! nia(4,ndiv*ndiv)
      integer, dimension(:,:), allocatable :: nja   ! nja(4,ndiv*ndiv)
      integer, dimension(:),   allocatable :: kblk  ! kblk(numel)
      integer, dimension(:),   allocatable :: kset  ! kset(numel)
      real (setpr),dimension(:,:),allocatable :: xnd   ! xnd(3,numnp)
      real (setpr),dimension(:),  allocatable :: garea ! garea(numel)
      real (setpr),dimension(:,:),allocatable :: xil   ! xil(3,maxl+1)
      real (setpr),dimension(:,:),allocatable :: xjl   ! xjl(3,maxl+1)
      real (setpr),dimension(:,:),allocatable :: xia   ! xia(3,maxa)
      real (setpr),dimension(:,:),allocatable :: xja   ! xja(3,maxa)
      real (setpr),dimension(:,:),allocatable :: xng   ! xng(3,numel)
      real (setpr),dimension(:,:),allocatable :: xcg   ! xcg(3,numel)
      real (setpr),dimension(:) allocatable :: frow  ! frow(numel)
      real (setpr),dimension(:).allocatable :: f !f((numel**2+numel)/2)
```

```
    real (setpr),dimension(:),allocatable :: emisf ! emisf(numel)
    real (setpr),dimension(:),allocatable :: tranf ! tranf(numel)
    real (setpr),dimension(:),allocatable :: rowerr! rowerr(numel)
!
    real (setpr),    dimension(4)        :: emisn
    real (setpr),    dimension(4)        :: transn
    real (setpr)                         :: sigma, solar
    integer idamax
    external idamax
!..... ........... ............... ............ ........... .......
! ***************************
! *** Interface Prototypes ***
! ****************************
    interface
     subroutine check_alloc(string,error,nerr,nf_out)
      character*(*) string
      integer nerr,nf_out,i
      integer error(nerr)
     end subroutine check_alloc
     subroutine check_dealloc(string,error,nerr,nf_out)
      character*(*) string
      integer nerr,nf_out,i
      integer error(nerr)
     end subroutine check_dealloc
     subroutine datain(nds,nocpl,nia,nja,kblk,kset,xnd,garea,xil,xjl, &
                xia,xja,xng,xcg,frow,f)
      use setreal_h
      integer, dimension(6,*)     :: nds   ! nds(6,numel)
       integer, dimension(2,*)     :: nocpl ! nocpl(2,ncpl+1)
      integer, dimension(4,*)     :: nia   ! nia(4,ndiv*ndiv)
      integer, dimension(4,*)     :: nja   ! nja(4,ndiv*ndiv)
      integer, dimension(*)       :: kblk  ! kblk(numel)
      integer, dimension(*)       :: kset  ! kset(numel)
      real (setpr),dimension(3,*) :: xnd   ! xnd(3,numnp)
      real (setpr),dimension(*)   :: garea ! garea(numel)
      real (setpr),dimension(3,*) :: xil   ! xil(3,maxl+1)
      real (setpr),dimension(3,*) :: xjl   ! xjl(3,maxl+1)
      real (setpr),dimension(3,*) :: xia   ! xia(3,maxa)
      real (setpr),dimension(3,*) :: xja   ! xja(3,maxa)
      real (setpr),dimension(3,*) :: xng   ! xng(3,numel)
      real (setpr),dimension(3,*) :: xcg   ! xcg(3,numel)
      real (setpr),dimension(*)   :: frow  ! frow(numel)
      real (setpr),dimension(*)   :: f     ! f((numel*2+numel)/2)
     end subroutine datain
     subroutine factor(f,emisf,tranf,garea,numel,nf_out,nf_log)
       use setreal_h
      integer                     :: numel
      real (setpr),dimension(*)   :: f     ! f((numel*2+numel)/2)
      real (setpr),dimension(*)   :: tranf ! tranf(numel)
      real (setpr),dimension(*)   :: emisf ! emisf(numel)
      real (setpr),dimension(*)   :: garea ! garea(numel)
      integer nf_out,nf_log
     end subroutine factor
     subroutine fopen_read( fin, u, ilog)
      character*20, intent(in) :: fin
      integer, intent (in)     :: u
      integer, optional        :: ilog
     end subroutine fopen_read
     subroutine fopen_write( fin, u, ilog)
      character*20, intent(in) :: fin
      integer, intent (in)     :: u
      integer, optional        :: ilog
     end subroutine fopen_write
     subroutine timing (k)
      integer                  :: k
     end subroutine timing
     subroutine uopen_write( fin, u, ilog)
      character*20, intent(in) :: fin
      integer, intent (in)     :: u
      integer, optional        :: ilog
      end subroutine uopen_write
      subroutine viewr (nnodemax,nelemax,nds,sigma,solar,  &
                 garea,f,emisf,tranf,rowerr)
       use setreal_h
       integer                :: nnodemax,nelemax
       integer, dimension(6,*)   :: nds
       real (setpr)            :: sigma, solar
       real (setpr), dimension(*) :: garea
       real (setpr), dimension(*) :: f
       real (setpr), dimension(*) :: emisf
       real (setpr), dimension(*) :: tranf
       real (setpr), dimension(*) :: rowerr
       end subroutine viewr
      subroutine view3d(nds,nocpl,nia nja,kblk,kset,xnd,garea,xil,xjl,  &
            xia,xja,xng,xcg,frow,f)
       use setreal_h
       integer, dimension(6,*)     :: nds   ! nds(6,numel)
         integer, dimension(2,*)      :: nocpl ! nocpl(2,ncpl+1)
       integer, dimension(4,*)     :: nia   ! nia(4.ndiv*ndiv)
       integer, dimension(4,*)     :: nja   ! nja(4.ndiv*ndiv)
       integer, dimension(*)       :: kblk  ! kblk(numel)
       integer, dimension(*)       :: kset  ! kset(numel)
       real (setpr), dimension(3,*) :: xnd   ! xnd(3,numnp)
       real (setpr), dimension(*)   :: garea ! garea(numel)
       real (setpr), dimension(3,*) :: xil   ! xil(3,maxl+1)
       real (setpr), dimension(3,*) :: xjl   ! xjl(3,maxl+1)
       real (setpr), dimension(3,*) :: xia   ! xia(3,maxa)
       real (setpr), dimension(3,*) :: xja   ! xja(3,maxa)
       real (setpr), dimension(3,*) :: xng   ! xng(3,numel)
       real (setpr), dimension(3,*) :: xcg   ! xcg(3,numel)
       real (setpr), dimension(*)   :: frow  ! frow(numel)
       real (setpr), dimension(*)   :: f     ! f((numel**2+numel)/2)
      end subroutine view3d
!***************************
     end interface
!***************************
!    ***************************
!    variable declaration section
!    ***************************
     character*6 ,     dimension(15)  :: title
     character*8 ,     dimension(5)   :: head
     character*20                     :: nin,nout,nabs
     integer,          dimension(19)  :: error
     integer                          :: nrot
     integer                          :: i,icheck
     integer                          :: ndim,nummat,ibug
     real (setpr),     dimension(*)   :: EMISSIVITY
     real (setpr),     dimension(*)   :: TRANSMISSIVITY
!    ********************
!    open chad_view files
!    ********************
     nin  = 'chad_view'//fbname//'.in'
     nabs = 'chad_view.bin'
     nout = 'chad_view'//fbname//'.out'
     jin  = nf_in
     jout = nf_out
     jlog = nf_log
     jabs = nf_abs
     call fopen_read(  nin, jin , ilog=jlog)
     call uopen_write( nabs, jabs, ilog=jlog)
!    ***************************
!    read title and control data
!    ***************************
     call timedat (head(1),head(2),head(3))
     read (jin,90) (title(i),i=1,12)
     head(4)='chadview'
     head(5)='02/14/96'
!    write (jout,110) (title(i),i=1,12) (head(i),i=1,3),head(5)
     write (jout,170) nin,nout,nabs
     write (jout,180)
```

```
!    write (jlog,110) (title(i),i=1,12),(head(i),i=1,3),head(5)
     if(myPE.eq.root) then
     write (jlog,170) nin,nout,nabs
     write (jlog,180)
     endif
     read (jin,140) ndim,nummat,numnp,numel,ndiv,nblk,nrot,icheck, &
                 ncpl,lbug
!.....set default values
     if (ndiv.EQ.0) ndiv=5
     write (jout,150) ndim,nummat,numnp,numel,ndiv,nblk
     if(myPE.eq.root) then
     write (jlog,150) ndim,nummat,numnp,numel,ndiv,nblk
     endif
!    ************
!    set pointers
!    ************
     maxl  = 4*ndiv
     maxa  = (ndiv+1)**2
     maxb  = (numel*numel+numel)/2
!    ********************
!    allocate HEAP memory
!    ********************
       allocate ( nds(6,numel),      stat=error(1)  )
       allocate ( nocpl(2,ncpl+1),  stat=error(2)  )
       allocate ( nia(4,ndiv*ndiv), stat=error(3)  )
       allocate ( nja(4,ndiv*ndiv), stat=error(4)  )
       allocate ( kblk(numel),      stat=error(5)  )
       allocate ( kset(numel),      stat=error(6)  )
       allocate ( xnd(3,numnp),     stat=error(7)  )
       allocate ( garea(numel),     stat=error(8)  )
       allocate ( xil(3,maxl+1),    stat=error(9)  )
       allocate ( xjl(3,maxl+1),    stat=error(10) )
       allocate ( xia(3,maxa),      stat=error(11) )
       allocate ( xja(3,maxa),      stat=error(12) )
       allocate ( xng(3,numel),     stat=error(13) )
       allocate ( xcg(3,numel),     stat=error(14) )
       allocate ( frow(numel),      stat=error(15) )
       allocate ( f(maxb),          stat=error(16) )
       allocate ( emisf(numel),     stat=error(17) )
       allocate ( tranf(numel),     stat=error(18) )
       allocate ( rowerr(numel),    stat=error(19) )
! ....is there sufficient memory to solve the problem?
       call check_alloc('FACET90',error,19,nf_out)
!    ************************
!    read remaining input data
!    ************************
     call datain(nds,nocpl,nia,nja,kblk,kset,xnd,garea,xil,xjl, &
              xia,xja,xng,xcg,frow,f)
!    ************************
!    initialize f to zero
!    ************************
     f(:) = 0.0

     call timing (1)
     if (icheck .EQ. 0) then
!    ***************************************************
!... determine face emissivities and transmissivities for CHAD
!    ***************************************************
     do i=1,numel
        emisn(1) = EMISSIVITY(nds(1,i))
        emisn(2) = EMISSIVITY(nds(2,i))
        emisn(3) = EMISSIVITY(nds(3,i))
        emisn(4) = EMISSIVITY(nds(4,i))
        transn(1)= TRANSMISSIVITY(nds(1,i))
        transn(2)= TRANSMISSIVITY(nds(2,i))
        transn(3)= TRANSMISSIVITY(nds(3,i))
        transn(4)= TRANSMISSIVITY(nds(4,i))
       if ( nds(6,i) .EQ. 1) then
!       opaque wall element
        emisf(i) = 0.25*(emisn(1)+emisn(2)+emisn(3)+emisn(4))
```

```
          tranf(i) = 0.0
        elseif ( nds(6,i) .EQ. 2) then
!         blackbody cross-flow element
          emisf(i) = 1.0
          tranf(i) = 0.0
        elseif ( nds(6,i) .EQ. 3) then
!         partially transmitting window element
          emisf(i) = emisn(idamax(4,transn,1))
          tranf(i) = transn(idamax(4,transn,1))
        endif
      enddo
!     **********************
!.....calculate view factors
!     **********************
         if(myPE.eq.root) then
        write (jlog,120)
         endif
        call view3d(nds,nocpl,nia,nja,kblk,kset,xnd,garea,xil,xjl,  &
                xia,xja,xng,xcg,frow,f)
        call timing (2)
!     ***************************************************
!.....calculate row sum errors and write out binary file
!     ***************************************************
        if(myPE.eq.root) then
         write (jlog,130)
         call viewr (nnodemax,nelemax,nds,sigma,solar,  &
                garea,f,emisf,tranf,rowerr)
        endif
       else
        write(jout,250)
         if(myPE.eq.root) then
        write(jlog,250)
         endif
       endif
!     **********************************
!.....Carry out LU factorization of AMAT
!     **********************************
      call factor(f,emisf,tranf,garea,numel,nf_out,nf_log)
!     ********************
!.....release HEAP memory
!     ********************
      deallocate ( nds,    stat=error(1)  )
      deallocate ( nocpl,  stat=error(2)  )
      deallocate ( nia,    stat=error(3)  )
      deallocate ( nja,    stat=error(4)  )
      deallocate ( kblk,   stat=error(5)  )
      deallocate ( kset,   stat=error(6)  )
      deallocate ( xnd,    stat=error(7)  )
      deallocate ( garea,  stat=error(8)  )
      deallocate ( xil,    stat=error(9)  )
      deallocate ( xjl,    stat=error(10) )
      deallocate ( xia,    stat=error(11) )
      deallocate ( xja,    stat=error(12) )
      deallocate ( xng,    stat=error(13) )
      deallocate ( xcg,    stat=error(14) )
      deallocate ( f,      stat=error(16) )
      deallocate ( emisf,  stat=error(17) )
      deallocate ( tranf,  stat=error(18) )
      deallocate ( rowerr, stat=error(19) )
!.....Was deallocation successful?
      call check_dealloc('FACET90',error,19,nf_out)
      close (unit=jin,status='delete')
      close (unit=jabs,status='keep')
      call timing(5)
!     ************************
!.....print timing information
!     ************************
      write (jout,160)
      write (jout,190) cpuio(1,1)
      write (jout,200) cpuio(1,2)
```

```
    write (jout,210) cpuio(1,3)
    write (jout,220) cpuio(1,4)
    write (jout,225) cpuio(1,5)
    write (jout,230) cpuio(1,6)
    if(myPE.eq.root) then
    write (jlog,160)
    write (jlog,190) cpuio(1,1)
    write (jlog,200) cpuio(1,2)
    write (jlog,210) cpuio(1,3)
    write (jlog,220) cpuio(1,4)
    write (jlog,225) cpuio(1,5)
    write (jlog,230) cpuio(1,6)
    endif

!
!   ****************
!.....FORMAT STATMENTS
!   ****************
  90 format (12a6)
 110 format(1x//12a6//15x,'time= ',a8,'   date= ',a8,'   machine= ',a8, &
    //' this version of chad  view compiled ',a8//' input phase')
 120 format (' solution phase'/5x,'calculating upper triangular', &
          ' [a*f] matrix')
 130 format (5x,'writing binary file')
 140 format (10i6)
 150 format(' GEOMETRY CODE                  =',i6,/ &
         ' NUMBER OF MATERIALS            =',i6,/ &
         ' NUMBER OF NODES                =',i6,/ &
         ' NUMBER OF ENCLOSURE SURFACES   =',i6,/ &
         ' NUMBER OF SUBSURFACE DIVISIONS =' i6,/ &
         ' NUMBER OF SHADING SURFACES     =',i6,//)
 160 format (//' ********************************'/ &
           ' ********* T I M I N G *********'/ &
           ' ********************************'/)
 170 format(//10x,' INPUT  FILE NAME      (text)     - ',a15/ &
           10x,' OUTPUT FILE NAME      (text)     - ',a15/ &
           10x,' VIEW FACTOR FILE NAME (binary)   - ',a15//)
 180 format(' ****************************************************'/ &
         ' *********** C O N T R O L   D A T A ************'/ &
         ' ****************************************************'/)
 190 format( ' initialization & data input',5x,1p,e12.5,' sec')
 200 format( ' area * view factor calc.   ',5x,1p,e12.5,' sec')
 210 format( ' reciprocity calc. & output',5x,1p,e12.5,' sec')
 220 format( ' row-sums calculation       ',5x,1p,e12.5,' sec')
 225 format( ' LU factorization of AMAT   ',5x,1p,e12.5,' sec')
 230 format(/' total time used            ',5x,1p,e12.5,' sec')
 250 format(//' data check only')
    end subroutine facet90


    integer function idamax(n,dx,incx)
!   ****************************************************************
!   Finds the index of element having max  absolute value.
!   Jack Dongarra, LINPACK, 3/11/78.
!   modified 3/93 to return if incx .LE. 0.
!   modified 12/3/93, array(1) declarations changed to array(*)
!   ****************************************************************
    use setreal_h
    real (setpr)   dx(*),dmax
    integer i,incx,ix,n
!
    idamax = 0
    if( n.LT.1 .OR. incx.LE.0 ) return
    idamax = 1
    if(n.EQ.1)return
    if(incx.EQ.1)go to 20
!
!     code for increment not equal to 1
!
    ix = 1
```

```
      dmax = abs(dx(1))
      ix = ix + incx
      do 10 i = 2,n
        if(abs(dx(ix)).LE.dmax) go to 5
        idamax = i
        dmax = abs(dx(ix))
    5   ix = ix + incx
   10 continue
      return
!
!     code for increment equal to 1
!
   20 dmax = abs(dx(1))
      do 30 i = 2,n
        if(abs(dx(i)).LE.dmax) go to 30
        idamax = i
        dmax = abs(dx(i))
   30 continue
      return
      end


!***********************************************************************
      subroutine couple (nocpl,iseg,jseg,icouple)
!***********************************************************************
      use setreal_h
      use primary_h
!***********************************************************************
      integer, dimension(2,ncpl+1)     :: nocpl
      integer                          :: iseg,jseg
      integer                          :: icpl
      integer                          :: icouple
!***********************************************************************
      DATA icpl /1/
      save icpl
!***********************************************************************
      icouple = 1
      if ( icpl .LE. ncpl ) then
        if ( iseg.EQ.nocpl(1,icpl) .AND. &
             jseg.EQ.nocpl(2,icpl) )  then
          icpl   = icpl + 1
          icouple = 0
        endif
      endif
!
      end subroutine couple



!     ******************************************************************
      subroutine datain(nds,nocpl,nia,nja,kblk,kset,xnd,garea,xil,xjl, &
                        xia,xja,xng,xcg,frow,f)
!     ******************************************************************
      use setreal_h
      use primary_h
      use compar
!.............................................................
! ****************************
! *** Interface Prototypes ***
! ****************************
      interface
        subroutine geom3d (x,y,z,xc,yc,zc,xn,yn,zn,area)
          use setreal_h
          real (setpr), dimension(4)     :: x
          real (setpr), dimension(4)     :: y
          real (setpr), dimension(4)     :: z
          real (setpr)                   :: xc, yc, zc, xn, yn, zn, area
        end subroutine geom3d
```

```
! *****************************
    end interface
! *****************************
!***************************************************************************
      integer, dimension(6,*)      :: nds   ! nds(6,numel)
        integer, dimension(2,*)     :: nocpl ! nocpl(2,ncpl+1)
      integer, dimension(4,*)      :: nia   ! nia(4,ndiv*ndiv)
      integer, dimension(4,*)      :: nja   ! nja(4,ndiv*ndiv)
      integer, dimension(*)        :: kblk  ! kblk(numel)
      integer, dimension(*)        :: kset  ! kset(numel)
      real (setpr), dimension(3,*) :: xnd   ! xnd(3,numnp)
      real (setpr), dimension(*)   :: garea ! garea(numel)
      real (setpr), dimension(3,*) :: xil   ! xil(3,maxl+1)
      real (setpr), dimension(3,*) :: xjl   ! xjl(3,maxl+1)
      real (setpr), dimension(3,*) :: xia   ! xia(3,maxa)
      real (setpr), dimension(3,*) :: xja   ! xja(3,maxa)
      real (setpr), dimension(3,*) :: xng   ! xng(3,numel)
      real (setpr), dimension(3,*) :: xcg   ! xcg(3,numel)
      real (setpr), dimension(*)   :: frow  ! frow(numel)
      real (setpr), dimension(*)   :: f     ! f((numel**2+numel)/2)
      real (setpr), dimension(4)   :: x, y, z
      real (setpr)                 :: dx, dy, dz, xnum
      integer                      :: nold, kn0, np, kn
      integer                      :: n, k  kk, num, numn
      integer                      :: i, j, m, nmiss, inc
      integer                      :: l, jj
    data nold,kn0,np/0,0,2/
!*****************************************
!.....input and generate node point data
!*****************************************
    if(myPE.eq.root) then
    write (jlog,250)
    endif
    n = 0
    kn0 = 0
    nold = 1
    nodal_data: do
      if ( n .GE. numnp) exit nodal_data
      kn = max( kn0, 1)
      read (jin,160) n,xnd(1,n),xnd(2,n),xnd(3,n),kn0
      num  = (n-nold)/kn
      numn = num - 1
      if (numn .GE. 1) then
       xnum = real(num)
       dx   = (xnd(1,n) - xnd(1,nold)) / xnum
       dy   = (xnd(2,n) - xnd(2,nold)) / xnum
       dz   = (xnd(3,n) - xnd(3,nold)) / xnum
       k    = nold
       do 20 j=1,numn
        kk      = k
        k       = k + kn
        xnd(1,k) = xnd(1,kk) + dx
        xnd(2,k) = xnd(2,kk) + dy
        xnd(3,k) = xnd(3,kk) + dz
 20     continue
      endif
      nold = n
    end do nodal_data
!
!**************************
! ....write node point data
!**************************
    write (jout,170)
    do 40 i=1,numnp
     write (jout,180) i,(xnd(j,i),j=1,3)
 40 continue
!************************************
!.....input and generate surface data
!************************************
    if(myPE.eq.root) then
```

```
     write (jlog,260)
     endif
     m = 0
     surface_data: do
       if ( m .GE. numel ) exit surface_data
       read (jin,190) m,(nds(j,m),j=1,5),nmiss,inc
       nds(6,m) = max(nds(5,m), 1)
       nds(5,m) = nds(1,m)
       if (nmiss .GT. 0) then
         inc = max( inc, 1)
         do 70 i=1,nmiss
           l = m
           m = m+1
           do 60 j=1,4
             nds(j,m) = nds(j,l) + inc
  60         continue
           nds(6,m) = nds(5,l)
           nds(5,m) = nds(1,l)
  70     continue
       endif
     end do surface_data
!****************************
!.....calculate surface area
!****************************
     np = 4
     do 100 j=1,numel
       do 90 jj=1,np
         x(jj) = xnd(1,nds(jj,j))
         y(jj) = xnd(2,nds(jj,j))
         z(jj) = xnd(3,nds(jj,j))
  90   continue
       call geom3d(x,y,z,xcg(1,j),xcg(2,j),xcg(3,j), &
                 xng(1,j),xng(2,j),xng(3,j),garea(j))
 100 continue
!.... write surface data
     write (jout,200)
     do 130 j=1,numel
       write (jout,210) j,(nds(i,j),i=1,4),nds(6,j),garea(j)
 130 continue
!****************************
!.....input blocking surfaces
!****************************
     if (nblk .GT. 0) then
       if(myPE.eq.root) then
       write (jlog,270)
       endif
       write (jout,220)
       m = 0
       block_surface : do
         if ( m .GE. nblk ) exit block_surface
         m = m + 1
         read (jin,230)  kblk(m),nmiss,inc
         if (nmiss .GT. 0) then
           inc = max( inc, 1)
           do 141 i=1,nmiss
             m = m + 1
             kblk(m) = kblk(m-1) + inc
 141       continue
         endif
       do 143 i=1 nblk
         write (jout,240) i,kblk(i)
 143   continue
       end do block_surface
     endif
!**************************************************
!.....input surface pairs that cannot intercouple
!**************************************************
     if (ncpl .GT. 0) then
       if(myPE.eq.root) then
       write (jlog,280)
```

```
        endif
       write (jout,290)
       m = 0
       intercouple: do
         if (m .GE. ncpl) exit intercouple
         m = m + 1
         read (jin,190) nocpl(1,m),nocpl(2,m),nmiss,inc
         if (nmiss .GT. 0) then
           inc = max(inc, 1)
           do 156 i=1,nmiss
             m = m + 1
             nocpl(1,m) = nocpl(1,m-1)
             nocpl(2,m) = nocpl(2,m-1) + inc
 156       continue
         endif
       end do intercouple
       do 158 i=1,ncpl
         write (jout,300) i,nocpl(1,i),nocpl(2,i)
 158   continue
     endif
!
 160 format (i6,5x,3e12.0,i6)
 170 format(// '*************************************************'/ &
            '************* N O D A L   D A T A *************'/ &
            '*************************************************'// &
            ' node',6x 'x1',10x,'x2',10x,'x3',/)
 180 format (i6,1p,3e12.4)
 190 format (8i6)
 200 format(// '*************************************************'/ &
            '*********** S U R F A C E   D A T A ***********'/ &
            '*************************************************'// &
            ' ele #   n1   n2   n3   n4  mat',12x,'area',/)
 210 format (6i6,1pe20.8)
 220 format(// '*************************************************'/ &
            '******* B L O C K I N G  S U R F A C E S ********'/ &
            '*************************************************'// &
            '         index',5x,'surface'/)
 230 format (3i6)
 240 format (12x,i6,5x,i6)
 250 format(5x,'reading node data')
 260 format(5x,'reading surface data')
 270 format(5x,'reading obstructing surfaces')
 280 format(5x,'reading surface pairs that cannot intercouple')
 290 format(// '*************************************************'/ &
            '********  SURFACES THAT CANNOT COUPLE  ***********'/ &
            '*************************************************'// &
            '          pair   iseg   jseg'/)
 300 format(12x,i6,2x,i6,2x,i6)
!
     end subroutine datain



!*************************************************************************
     subroutine edge (xnd,nds,iseg,jseg,iedge)
!*************************************************************************
!  subroutine to determine if two quadrilaterals have an adjoint edge*
!       iedge=1  not adjoint                                          *
!       iedge=2  adjoint                                              *
!*************************************************************************
     use setreal_h
     use primary_h
!*************************************************************************
     real (setpr), dimension(3,numnp) :: xnd
     real (setpr), dimension(3)       :: dif
     integer, dimension(6,numel)      :: nds
     integer                          :: iseg,jseg,iedge
     integer                          :: k,i,j,nodei,nodej
     real (setpr), parameter          :: small = 1.0E-06
!*************************************************************************
```

```
    iedge = 1
    k    = 0
    do 20 i=1,4
      nodei = nds(i,iseg)
      do 10 j=1 4
        nodej = nds(j,jseg)
        dif  = abs( xnd(:,nodei) - xnd(:,nodej) )
        if ( maxval(dif) .LE. small ) k = k + 1
 10   continue
 20 continue
    if (k  EQ. 2) iedge = 2
!
    end subroutine edge




!***************************************************************************c
!*** file: fopen_read.f                                                 ***c
!***                                                                    ***c
!***            open a read  only formatted file                        ***c
!*** input:                                                             ***c
!***    fin = file name                                                 ***c
!***    u   = unit number                                               ***c
!***    ilog= output unit number for reporting (optional;defaults to 6)***c
!***                                                                    ***c
!*** p. t. williams                                                     ***c
!***                                                                    ***c
!***************************************************************************c
    subroutine fopen_read( fin, u, ilog)
!***************************************************************************c
    character*20, intent(in) :: fin
    integer, intent (in)     :: u
    integer, optional        :: ilog
    integer                  :: iout
!   declare open and inquire statement variables
    character acC*10, act*09, blnk*10, del*10, dir*07  &
            fmt*09, fm*11,  fn*15,   pad*03, pos*06, r*07, &
            rw*07,  seq*07, sta*07,  unf*11  w*7
    integer ios,  nr, num,  recl
    logical  ex, nmd, od
!*******************************
!   declare local subprograms.
!*******************************
    logical f90openeh
    external f90openeh
!***************************************
!   check to see if ilog is present
!***************************************
    iout = 6
    if ( present( ilog ) ) iout = ilog
!*****************************************************
!   initialize open and inquire statement variables.
!*****************************************************
    acC = ''
    act = ''
    blnk = ''
    del  = ''
    dir  = ''
    ex   = .FALSE.
    fm   = ''
    fmt  = ''
    ios  = 0
    fn   = ''
    nmd  = .FALSE.
    nr   = 0
    num  = 0
    od   = .FALSE.
    pad  = ''
    pos  = ''
```

```
      r   = ' '
      recl = 0
      rw   = ' '
      seq  = ' '
      sta  = ' '
      unf  = ' '
      w    = ' '
!
!     ****************************************************
!     open the file
      open ( unit = u  file=fin, form='formatted',  &
           action='read',  iostat=ios)
!     ****************************************************
!
      if ( .NOT. f90openeh( 'open    ', ios, iout ) ) goto 1000
!
!     get the status of the file
      inquire( unit=u,        iostat=ios   &
            access=acc,      action=act, blank=blnk,   delim=del,  &
            direct=dir,     exist=ex,   form=fm,     formatted=fmt,  &
            name=fn,        named=nmd.  nextrec=nr,   number=num,  &
            opened=od,      pad=pad,    position=pos, read=r,  &
            readwrite=rw,   recl=recl,  sequential=seq,  &
            unformatted=unf, write=w)
!
      if ( .NOT. f90openeh( 'inquire', ios, iout) ) goto 1000
!
!     display inquire keyword values
      write (iout,600)
 600  format (/1h , 'inquire keyword values  .. ')
      write (iout,700) acc, act, blnk, del. dir. ex, &
                 fin,  fm,  fmt, ios,  fn, nmd
 700  format ( 1h , 'access     ', a10,    5x, 'action      ',  a9, &
           / 1h , 'blank      ', a10,    5x, 'delim       ', a10, &
           / 1h , 'direct     ',  a4,   11x, 'exist       ',  l1, &
           / 1h , 'file       ', a15,        'form        ', a11, &
           / 1h , 'formatted  ',  a9,    6x, 'iostat      ',i5.5, &
           / 1h , 'name       ', a15,        'named       ',  l1)
      write (iout,800) nr,  num, od, pad, pos. r, &
                 rw,  recl  seq, unf, u  w
 800  format ( 1h , 'nextrec    ',i5.5,   10x, 'number      ' i5 5, &
           / 1h , 'opened     ',  l1,   14x, 'pad         ',  a3, &
           / 1h , 'position   ',  a6,    9x, 'read        ',  a7, &
           / 1h , 'readwrite  ',  a7,    8x, 'recl        ',i10.10, &
           / 1h , 'sequential ',  a7,    8x, 'unformatted ', a11, &
           / 1h , 'unit       ',i5.5,   10x, 'write       ', a7)
      goto 999
!
!     error handling
 1000 continue
      write (iout,900) fin
 900  format (/1h ,'error in fopen_read' file = ',a15)
      stop
!
 999  continue
      end subroutine fopen_read



!*******************************************************************c
!*** file: fopen_write.f                                         ***c
!***                                                             ***c
!***           open a write_only formatted file                  ***c
!*** input:                                                      ***c
!***    fin = file name                                          ***c
!***    u   = unit number                                        ***c
!***    ilog= output unit number for reporting (optional;defaults to 6)***c
!***                                                             ***c
!*** p. t. williams                                              ***c
```

```
!***                                               ***c
!*************************************************************************c
   subroutine fopen_write( fin, u, ilog)
!*************************************************************************c
   character*20, intent(in) :: fin
   integer, intent (in)     :: u
   integer, optional        :: ilog
   integer                  :: iout
!  declare open and inquire statement variables
   character acC*10, act*09, blnk*10, del*10, dir*07,  &
         fmt*09, fm*11, fn*15   pad*03, pos*06, r*07,  &
         rw*07,  seq*07, sta*07,  unf*11, w*7
   integer ios,  nr, num,  recl
   logical  ex, nmd, od
!*********************************************************************
!  declare local subprograms.
   logical f90openeh
   external f90openeh
!*********************************************************************
!  check to see if ilog is present
   iout = 6
   if ( present( ilog ) ) iout = ilog
!  initialize open and inquire statement variables.
   acC = ' '
   act = ' '
   blnk = ' '
   del  = ' '
   dir  = ' '
   ex   = .FALSE.
   fm   = ' '
   fmt  = ' '
   ios  = 0
   fn   = ' '
   nmd  = .FALSE.
   nr   = 0
   num  = 0
   od   = .FALSE.
   pad  = ' '
   pos  = ' '
   r    = ' '
   recl = 0
   rw   = ' '
   seq  = ' '
   sta  = ' '
   unf  = ' '
   w    = ' '
!
!  ***********************************************
!  open the file
   open ( unit = u, file=fin, form='formatted'  &
       action='readwrite', status='unknown', iostat=ios)
!  ***********************************************
!
   if ( .NOT. f90openeh( 'open   ', ios, iout ) ) goto 1000
!
!  get the status of the file
   inquire( unit=u,       iostat=ios, &
        access=acc,     action=act, blank=blnk,   delim=del, &
        direct=dir,     exist=ex,  form=fm,      formatted=fmt, &
        name=fn,        named=nmd,  nextrec=nr,  number=num, &
        opened=od,      pad=pad,    position=pos, read=r, &
        readwrite=rw,   recl=recl,  sequential=seq, &
        unformatted=unf, write=w)
!
   if ( .NOT. f90openeh( 'inquire', ios, iout) ) goto 1000
!
!  display inquire keyword values
   write (iout,600)
600 format (/1h , 'inquire keyword values ... ')
   write (iout,700) acc, act, blnk, del, dir, ex, &
```

```
                fin,  fm,  fmt, ios,  fn, nmd
 700  format ( 1h , 'access     ', a10,    5x, 'action      ',  a9, &
          / 1h   'blank      ', a10,    5x, 'delim       ', a10, &
          / 1h   'direct     ',  a4,   11x, 'exist       ',  l1, &
          / 1h , 'file       ', a15,        'form        ', a11, &
          / 1h , 'formatted  ',  a9,    6x, 'iostat      ',i5.5, &
          / 1h , 'name       ', a15,        'named       ',  l1)
     write (iout,800) nr,  num. od, pad, pos, r, &
                rw,  red, seq, unf, u, w
 800  format ( 1h , 'nextrec    ',i5.5,   10x, 'number      ',i5.5, &
          / 1h   'opened     ',  l1.   14x, 'pad         ',  a3, &
          / 1h , 'position   ',  a6,    9x, 'read        ',  a7, &
          / 1h , 'readwrite  ',  a7,    8x, 'recl        ',i10.10, &
          / 1h , 'sequential ',  a7     8x, 'unformatted ', a11, &
          / 1h , 'unit       ',i5.5,   10x, 'write       ', a7)
      goto 999
!
!     error handling
 1000  continue
      write (iout,900) fin
 900   format (/1h ,'error in fopen_write! file = ',a15)
      stop
!
 999   continue
      end subroutine fopen_write


!*********************************************************************
     subroutine fwrite (frow,f,iseg)
!*********************************************************************
     use setreal_h
     use primary_h
!*********************************************************************
     real (setpr), dimension(numel)                    :: frow
     real (setpr), dimension( (numel*numel+numel)/2 ) :: f
     integer                                           :: nw,iseg
     integer                                           :: jseg,loca
     integer                                           :: i
!*********************************************************************
!    declare local subprograms
     integer indx
     external indx
!*********************************************************************
     nw  = numel - iseg + 1
     do 100 i=1,nw
      jseg = iseg + i - 1
      loca = indx(iseg,jseg)
      f(loca) = frow(i)
 100  continue
!
     end subroutine fwrite



!*********************************************************************c
     logical function f90openeh ( statement, ios, iout)
!*********************************************************************c
     character*7 statement
     integer   ios
     integer   iout
!
     f90openeh = .FALSE.
     if ( ios .GT. 0 ) then
       write ( iout, 100 ) statement, ios
     elseif ( ios .EQ. 0) then
       f90openeh = .TRUE.
     else
       write (iout, 200 ) statement, ios
     endif
```

```
!
 100  format( / ' error! ', a7, ' statment (', &
               ' iostat=',i5.5, ')' )
 200  format ( / 1h , a7,  statement iostat = ,i6.5,'?')
!
      end function f90openeh



!***********************************************************************
      subroutine geom3d (x,y,z,xc,yc,zc,xn,yn,zn,area)
!***********************************************************************
      use setreal_h
      use global_h
      use compar
      real (setpr), dimension(4) :: x
      real (setpr), dimension(4) :: y
      real (setpr), dimension(4) :: z
      real (setpr)              :: xc, yc, zc, xn, yn, zn, area, s, t
      real (setpr)              :: third
      real (setpr), parameter   :: half = 0.5
!***********************************************************************
      third = 1.0/3.0
!.....calculate unit normal vector
      xn = (y(3)-y(2))*(z(1)-z(2))-(z(3)-z(2))*(y(1)-y(2))
      yn = (z(3)-z(2))*(x(1)-x(2))-(x(3)-x(2))*(z(1)-z(2))
      zn = (x(3)-x(2))*(y(1)-y(2))-(y(3)-y(2))*(x(1)-x(2))
      s  = sqrt( xn*xn + yn*yn + zn*zn )
      xn = xn / s
      yn = yn / s
      zn = zn / s
!.....calculate area
      t = sqrt(((y(3)-y(4))*(z(4)-z(1))-(z(3)-z(4))*(y(4)-y(1)))**2  &
            +((z(3)-z(4))*(x(4)-x(1))-(x(3)-x(4))*(z(4)-z(1)))**2  &
            +((x(3)-x(4))*(y(4)-y(1))-(y(3)-y(4))*(x(4)-x(1)))**2)
      area = half * ( s + t )
!.....calculate centroid
      xc = third * (x(1) + x(3) + 1./(s+t) * (s*x(2) + t*x(4)) )
      yc = third * (y(1) + y(3) + 1./(s+t) * (s*y(2) + t*y(4)) )
      zc = third * (z(1) + z(3) + 1./(s+t) * (s*z(2) + t*z(4)) )
!
      end subroutine geom3d




!***********************************************************************
      subroutine grida (xnd,nds,xpa,nda,kseg)
!***********************************************************************
!***    subdivide surface area for area integration                 ***
!***********************************************************************
      use setreal_h
      use primary_h
!***********************************************************************
!***             variable declaration section                   ***
!***********************************************************************
      real (setpr), dimension(3,numnp) :: xnd
      real (setpr), dimension(3,maxa)  :: xpa
      real (setpr), dimension(3)       :: dd
      real (setpr)                     :: div
      integer, dimension(6,numel)      :: nds
      integer, dimension(4,ndiv*ndiv)  :: nda
      integer, dimension(5)            :: loci
      integer                          :: ndivm1,ndivp1,n,i,j
      integer                          :: ido,jdo,l1,l2,jm,jj,jm1
      integer                          :: nel nelm1,inc,kseg
!***********************************************************************
      div    = real( ndiv )
      ndivm1 = ndiv - 1
      ndivp1 = ndiv + 1
```

```
!.....set corner points
     loci(1) = 1
     loci(2) = 1 + ndiv
     loci(3) = maxa
     loci(4) = maxa - ndiv
     do 10 i=1,4
       n      = nds(i,kseg)
       j      = loci(i)
       xpa(:,j) = xnd(:,n)
  10 continue
! ....set point on sides 1-4 and 2-3
     do 30 ido=1,2
       if (ido.EQ.1) then
         l1 = loci(1)
         l2 = loci(4)
       else
         l1 = loci(2)
         l2 = loci(3)
       endif
       dd = (xpa(:,l2) - xpa(:,l1)) / div
       jm = l1
       do 20 jj=1,ndivm1
         j      = l1 + jj*ndivp1
         xpa(:,j) = xpa(:,jm) + dd
         jm     = j
  20   continue
  30 continue
!. ...set interior points
     do 50 ido=1,ndivp1
       l1 = loci(1) + (ido-1)*ndivp1
       l2 = loci(2) + (ido-1)*ndivp1
       dd = (xpa(:,l2) - xpa(:,l1)) / div
       do 40 jj=1,ndivm1
         j      = l1 + jj
         jm1    = j - 1
         xpa(:,j) = xpa(:,jm1) + dd
  40   continue
  50 continue
!.....determine subregion node numbering
     nel = 0
     inc = -1
     do 70 ido=1,ndiv
       nel       = nel + 1
       inc       = inc + 1
       nda(1,nel) = 1 + inc*ndivp1
       nda(2,nel) = nda(1,nel) + 1
       nda(3,nel) = nda(2,nel) + ndivp1
       nda(4,nel) = nda(1,nel) + ndivp1
       do 60 jdo=2,ndiv
         nelm1     = nel
         nel       = nel+1
         nda(:,nel) = nda(:,nelm1)+1
  60   continue
  70 continue
!
     end subroutine grida



!*************************************************************************
     subroutine gridl (xnd,nds,xpl,kseg)
!*************************************************************************
!***    subdivide surface contour for line integration           ***
!*************************************************************************
     use setreal_h
     use primary_h
!*************************************************************************
!***            variable declaration section                   ***
!*************************************************************************
```

```
     real (setpr), dimension(3,numnp)  :: xnd
     real (setpr), dimension(3,maxl+1) :: xpl
     real (setpr), dimension(3)        :: dd
     real (setpr)                      :: div
     integer, dimension(6,numel)       :: nds
     integer, dimension(5)             :: loci
     integer                           :: kseg,i,j,n
     integer                           :: ndivm1,jm1,jj
!*************************************************************************
!.....set corner points
     div = real( ndiv )
     j  = 1 - ndiv
     do 10 i=1,5
      j      = j + ndiv
      loci(i)  = j
      n      = nds(i,kseg)
      xpl( :,j) = xnd(:,n)
  10 continue
! ....set inbetween points
     do 30 i=1,4
      ndivm1    = ndiv-1
      dd        = (xpl(:,loci(i+1))-xpl(:,loci(i))) / div
      do 20 jj=1,ndivm1
       j      = loci(i) + jj
       jm1     = j - 1
       xpl(:,j) = xpl(:,jm1) + dd
  20   continue
  30 continue
     end subroutine gridl


!*************************************************************************
     integer function indx(iseg,jseg)
!*************************************************************************
     use setreal_h
     use primary_h
!*************************************************************************
     integer                   :: iseg,jseg
     integer                   :: irow,jcol
!*************************************************************************
     if ( jseg .GE. iseg) then
         irow = iseg
         jcol = jseg
     else
         irow = jseg
         jcol = iseg
     endif
!
     indx =  ((irow-1)*((2*numel) - irow))/2 + jcol
!
     end function indx



!*************************************************************************
     subroutine obstr (xnd,nds,xng,xcg,kblk,kset,iseg,jseg,nset)
!*************************************************************************
! this subroutine inspects all the specified obstructing surfaces      *
! as possible shadowing surfaces between the current view factor        *
! surfaces iseg & jseg.  a subset of obstructing surfaces is          *
! formed.  when the view factor is calculated between iseg & jseg      *
! this subset is examined for the shadowing sufaces.                  *
!*************************************************************************
     use setreal_h
     use primary_h
!.........................................................................
! ****************************
! *** Interface Prototypes ***
! ****************************
     interface
```

```
       subroutine sectn (xnd,nds,xcg,xng,iseg,jseg,kseg,iflag)
        use setreal_h
        real (setpr), dimension(3,*) :: xnd
        real (setpr), dimension(3,*) :: xng
        real (setpr), dimension(3,*) :: xcg
        integer, dimension(6,*)      :: nds
        integer                      :: iseg,jseg,kseg,iflag
       end subroutine sectn
! ***************************
      end interface
! ***************************
!
      real (setpr), dimension(3,numnp) :: xnd
      real (setpr), dimension(3,numel) :: xng
      real (setpr), dimension(3,numel) :: xcg
      real (setpr)                     :: dot
!
      integer, dimension(6,numel)      :: nds
      integer, dimension(numel)        :: kblk
      integer, dimension(numel)        :: kset
      integer                          :: iseg,jseg,nset
      integer                          :: i,kount,kseg
      integer                          :: iflag
!
      real (setpr), parameter          :: small = 1.0E-06
!
      do 10 i=1,nblk
        kset(i) = 0
   10 continue
      kount_loop: do kount=1,nblk
        kseg = kblk(kount)
        if (iseg.EQ.kseg .OR. jseg.EQ.kseg) cycle kount_loop
          dot = dot_product(xng(:,kseg),(xcg(:,iseg)-xcg(:,jseg)))
        if (abs(dot) .GT. small) then
          call sectn (xnd,nds,xcg,xng,iseg,jseg,kseg,iflag)
          if (iflag .EQ. 0) cycle kount_loop
          endif
        nset      = nset + 1
        kset(nset) = kseg
      end do kount_loop
!
      end subroutine obstr



!***********************************************************************
      subroutine sectn (xnd,nds,xcg,xng,iseg,jseg,kseg,iflag)
!***********************************************************************
! this subroutine determines if a line connecting the centroids of    *
! surfaces iseg & jseg intersects surface kseg.                        *
!        iflag=0  no intersection                                     *
!        iflag=1  intersection                                         *
!***********************************************************************
      use setreal_h
      use primary_h
!***********************************************************************
      real (setpr), dimension(3,numnp)  :: xnd
      real (setpr), dimension(3,numel)  :: xng
      real (setpr), dimension(3,numel)  :: xcg
      real (setpr), dimension(5,3)      :: v
      real (setpr), dimension(3)        :: ax
      real (setpr), dimension(3)        :: xi
      real (setpr)                      :: c1,tnum,tden,t
      real (setpr)                      :: xl1,xl2,xl3,angle
      real (setpr)                      :: den,dot,adot
      real (setpr)                      :: twopi
      integer, dimension(6,numel)       :: nds
      integer                           :: iseg,jseg,kseg,iflag
      integer                           :: n,i
      real (setpr), parameter          :: small = 1.0E-06
      real (setpr), parameter          :: zero  = 0.0
      real (setpr), parameter          :: one   = 1.0
!**********************************************************************
      twopi = 8.0*atan(1.0)
      iflag = 0
      n     = nds(2,kseg)
!.....determine intersection point
      ax    = xcg(:,iseg) - xcg(:,jseg)
      tden  = dot_product( xng(:,kseg), ax)
      if (abs(tden) .LT. small) return
      c1    = dot_product( xng(:,kseg), xnd(:,n) )
      tnum  = dot_product( xng(:,kseg), xcg(:,iseg) ) - c1
      t     = tnum/tden
      xi    = xcg(:,iseg) - ax*t
!.....is intersection point between surfaces iseg & jseg?
      xl1   = sqrt(dot_product(xi-xcg(:,iseg),xi-xcg(:,iseg)))
      xl2   = sqrt(dot_product(xi-xcg(:,jseg),xi-xcg(:,jseg)))
      xl3   = sqrt(dot_product(ax,ax))
      if ((abs(xl3-xl2-xl1)/xl3) .GT. small) return
!.....is intersection point within quadrilateral kseg?
      v(1,:) = xnd(:,nds(1,kseg)) - xi
      v(2,:) = xnd(:,nds(2,kseg)) - xi
      v(3,:) = xnd(:,nds(3,kseg)) - xi
      v(4,:) = xnd(:,nds(4,kseg)) - xi
      v(5,:) = v(1,:)
      angle  = zero
      do 20 i=1,4
         den = sqrt(dot_product(  v(i,:),   v(i,:)))* &
               sqrt(dot_product(v(i+1,:), v(i+1,:)))
       if (den .LT. small) goto 30
       dot = dot_product( v(i,:), v(i+1,:) ) / den
       dot = min( one, max(-one,dot) )
       adot  = acos(dot)
       angle = angle + adot
  20 continue
     if (abs(angle-twopi) .GT. small) return
  30 iflag = 1
!
     end subroutine sectn




!**********************************************************************
     subroutine see (xnd,nds,xng,iseg,jseg,isee,iedge)
!**********************************************************************
! determine self shadowing between two surfaces                       *
!    isee=-1  partial shadowing                          *
!    isee= 0  total shadowing                           *
!    isee=+1  no shadowing                               *
!**********************************************************************
      use setreal_h
      use primary_h
!.....................................................
! ****************************
! *** Interface Prototypes ***
! ****************************
     interface
       subroutine edge (xnd,nds,iseg,jseg,iedge)
         use setreal_h
         real (setpr), dimension(3,*) :: xnd
         integer, dimension(6,*)      :: nds
         integer                      :: iseg,jseg,iedge
       end subroutine edge
! ****************************
     end interface
! ****************************
! *********************************************************************
     real (setpr), dimension(3,nummp) :: xnd
```

```
      real (setpr), dimension(3,numel) :: xng
      real (setpr), dimension(3)       :: xij
      integer  dimension(6,numel)      :: nds
      integer                    :: iseg,jseg,isee
      integer                    :: k,iper,iedge
      integer                    :: i,j,nodei,nodej
      real (setpr)               :: dotn,dotij,dotji
      real (setpr), parameter    :: zero = 0.0
!*******************************************************************
      k   = 0
!     iper = 0
!.....if surfaces iseg & jseg are perpendicular - set iper=1
!     dotn = dot_product( xng(:,iseg), xng(:,jseg) )
!     if ( abs(dotn) .LT. 1.e-06 ) iper = 1
!.....if surfaces iseg & jseg share a common edge - set iedge=2
      call edge (xnd,nds,iseg,jseg,iedge)
!.....check 16 corner point vector dot products with surface normals
      do 20 i=1,4
       nodei = nds(i,iseg)
       do 10 j=1,4
        nodej = nds(j,jseg)
        xij   = xnd(:,nodej) - xnd(:,nodei)
        dotij = dot_product( xij, xng(:,iseg) )
        dotji = dot_product(-xij, xng(:,jseg) )
        if (dotij.GT.zero .AND. dotji.GT.zero) k = k + 1
  10    continue
  20 continue
!.....set appropriate flags
      if ( k.EQ.0 ) then
       isee =  0                    ! no see  (total shadowing)
      elseif (k.NE.0 .AND. iedge.EQ.2) then
       isee =  1                    ! can see (adjoint)
      elseif ( k.EQ.16) then
       isee =  1                    ! can see (no shadowing)
      else
       isee = -1                    ! partial
      endif
!
      end subroutine see



!*******************************************************************
      subroutine timedat(timd,dat,mach)
!*******************************************************************
!***  return the current time, date, and machine letter          ***
!***  output                                      ***
!***   timd      the time of day in the form hh:mm:ss            ***
!***   dat       the date in the form mo/da/yr                 ***
!***   mach      the machine letter                          ***
!*******************************************************************
      character*8         :: timd, dat, mach
      integer             :: i,j,k
      integer, dimension(8)  :: elements
!*******************************************************************
      mach='IBMR6000'
      call date_and_time( values=elements )
      i = elements(2)
      j = elements(3)
      century: if ( elements(1) .LT. 2000) then
                 elements(1) = elements(1) - 1900
             else
                 elements(1) = elements(1) - 2000
      end if century
      k = elements(1)
      write(dat,'(i2,"/",i2,"/",i2)') i,j,k
      write(timd,'(i2,":",i2,":",i2)') &
           elements(5), &
           elements(6), &
           elements(7)
```

```
!
     end subroutine timedat

!*************************************************************************
    subroutine timeuse(tim)
!*************************************************************************
    use setreal_h
    real (setpr), dimension(3) :: tim
    real*4,      dimension(2) :: etimer
    integer                :: it1, it2, it3
!*************************************************************************
!   output arguments
!    tim(1),cpu   total cpu time used
!    tim(2),tio   total i/o time used
!    tim(3),sys   total system time used
!*************************************************************************
!   call lib$stat_timer(2,it1)                        vms
!   call lib$stat_timer(3,it2)                        vms
!   call lib$stat_timer(4,it3)                        vms
!   tim(1)=.01*float(it1)                            vms
!   tim(2)=.001*float(it2+it3)                         vms
!   tim(3)=0.                                    vms
    call system_clock(count=it1,count_rate=it2,count_max=it3)
    tim(1) = real(it1)/real(it2)
    tim(2) = real(it3)/real(it2)
!
    tim(1) = etime(etimer)
!
    end subroutine timeuse



!*************************************************************************
    subroutine timing (k)
!*************************************************************************
!*** accumulate cpu, io & sys solution times                     ***
!*************************************************************************
    use setreal_h
    use primary_h
    use compar
!*************************************************************************
    integer                  :: k
    real (setpr), dimension(3)    :: tp, t
    save tp,t
!*************************************************************************
    call timeuse (t)
    if (k .EQ. 0) then
!......initialize if k=0
      cpuio = 0.0
    else
!......accumulate cpu, io, & sys solution times
    if(t(1)-cpuio(1,6).gt.0) then
     cpuio(1,k) = cpuio(1,k) + t(1) - cpuio(1,6)
     cpuio(1,6) = cpuio(1,6) + cpuio(1,k)
    else
     cpuio(1,k) = cpuio(1,k) + t(2) - cpuio(1,6)
     cpuio(1,6) = cpuio(1,6) + cpuio(1,k)
    endif
    endif
    end subroutine timing



!*************************************************************************
!*** file: uopen_write.f                        ***
!***                                    ***
!***      open a write_only unformatted binary file            ***
!*** input:                                  ***
```

```
!*** fin = file name                                  ***
!*** u   = unit number                                ***
!*** ilog= output unit number for reporting (optional;defaults to 6)***
!***                                                  ***
!*** p. t. williams                                   ***
!***                                                  ***
!**********************************************************************
      subroutine uopen_write( fin, u, ilog)
!**********************************************************************
      character*20, intent(in) :: fin
      integer, intent (in)     :: u
      integer, optional        :: ilog
      integer                  :: iout
!     declare open and inquire statement variables
      character acC*10, act*09, blnk*10, del*10, dir*07, &
            fmt*09, fm*11, fn*15,  pad*03, pos*06, r*07, &
            rw*07,  seq*07, sta*07,  unf*11, w*7
      Integer ios,  nr, num,  recl
      logical  ex, nmd, od
!     declare local subprograms.
      logical f90openeh
      external f90openeh
!**********************************************************************
!     check to see if ilog is present
      iout = 6
      if ( present( ilog ) ) iout = ilog
!     initialize open and inquire statement variables.
      acC  = ''
      act  = ''
      blnk =  ''
      del  = ''
      dir  = ''
      ex   = .FALSE.
      fm   = ''
      fmt  = ''
      ios  = 0
      fn   = ''
      nmd  = .FALSE.
      nr   = 0
      num  = 0
      od   = .FALSE.
      pad  = ''
      pos  = ''
      r    = ''
      recl = 0
      rw   = ''
      seq  = ''
      sta  = ''
      unf  = ''
      w    = ''
!     ************************************************
!     open the file
      open ( unit = u, file=fin, form='unformatted', &
           action='write', status='unknown', iostat=ios)
!     ************************************************
      if ( .NOT. f90openeh( 'open   ', ios, iout ) ) goto 1000
!     get the status of the file
      inquire( unit=u,        iostat=ios,  &
            access=acc,     action=act, blank=blnk,   delim=del,  &
            direct=dir,     exist=ex,  form=fm,     formatted=fmt,  &
            name=fn,        named=nmd,  nextrec=nr,   number=num,  &
            opened=od,      pad=pad,    position=pos, read=r,  &
            readwrite=rw,   recl=recl,  sequential=seq,  &
            unformatted=unf, write=w)
!
      if ( .NOT. f90openeh( 'inquire', ios, iout) ) goto 1000
!
!     display inquire keyword values
      write (iout,600)
 600  format (/1h , 'inquire keyword values .. ')
```

```
    write (iout,700) acc, act, bink, del, dir, ex,  &
                fin, fm, fmt, ios, fn, nmd
 700  format ( 1h , 'access     ', a10,    5x, 'action     ',  a9,  &
         / 1h , 'blank      ' a10,    5x, 'delim      ', a10,  &
         / 1h , 'direct     ', a4,   11x, 'exist      ', l1,  &
         / 1h . 'file       ', a15,       'form       ', a11,  &
         / 1h , 'formatted  ', a9,    6x, 'iostat     ',i5.5,  &
         / 1h , 'name       ', a15,       'named      ', l1)
    write (iout,800) nr, num od, pad, pos, r,  &
                rw, recl, seq, unf, u, w
 800  format ( 1h , 'nextrec    ',i5.5,   10x, 'number     ',i5.5,  &
         / 1h , 'opened     ', l1,   14x, 'pad        ', a3,  &
         / 1h , 'position   ', a6,    9x, 'read       ', a7,  &
         / 1h , 'readwrite  ', a7,    8x, 'recl       ',i10.10,  &
         / 1h , 'sequential ', a7     8x, 'unformatted ', a11,  &
         / 1h , 'unit       ',i5.5,   10x, 'write      ', a7)
     goto 999
!
!    error handling
 1000  continue
     write (iout,900) fin
 900   format (/1h ,'error in uopen_write! file = ',a15)
     stop
!
 999   continue
     end subroutine uopen_write


!**********************************************************************
    subroutine viewaa (xnd,nds,xia,xja,nia,nja,xng,kset,frow, &
                iseg,jseg,ndiv2,nset)
!**********************************************************************
! double area summation algorithm for 3d geometries                          *
!**********************************************************************
    use setreal_h
    use primary_h
!......................................................................
! ****************************
! *** Interface Prototypes ***
! ****************************
    interface
      subroutine geom3d (x,y,z,xc,yc,zc,xn,yn,zn,area)
        use setreal_h
        real (setpr), dimension(4)     :: x
        real (setpr), dimension(4)     :: y
        real (setpr), dimension(4)     :: z
        real (setpr)                   :: xc, yc, zc, xn, yn, zn, area
      end subroutine geom3d
      subroutine sectn (xnd,nds,xcg,xng,iseg,jseg,kseg,iflag)
        use setreal_h
        real (setpr), dimension(3,*)   :: xnd
        real (setpr), dimension(3,*)   :: xng
        real (setpr), dimension(3,*)   :: xcg
        integer, dimension(6,*)        :: nds
        integer                        :: iseg,jseg,kseg,iflag
      end subroutine sectn
! ****************************
    end interface
! ****************************
!**********************************************************************
    real (setpr), dimension(3,numnp)  :: xnd
    real (setpr), dimension(3,maxa)   :: xia
    real (setpr), dimension(3,maxa)   :: xja
    real (setpr), dimension(3,numel)  :: xng
    real (setpr), dimension(numel)    :: frow
    real (setpr), dimension(4)        :: x
    real (setpr), dimension(4)        :: y
    real (setpr), dimension(4)        :: z
    real (setpr), dimension(3,2)      :: xcl
    real (setpr), dimension(3)        :: r, vni, vnj
    real (setpr)                    :: fij,areai,areaj
    real (setpr)                    :: rmag,cosbi,cosbj
    real (setpr)                    :: invpi
    integer, dimension(6,numel)       :: nds
    integer, dimension(4,ndiv*ndiv)   :: nia
    integer, dimension(4,ndiv*ndiv)   :: nja
    integer, dimension(numel)         :: kset
    integer, dimension(4)             :: ln
    integer                         :: nset,iflag,indexi
    integer                         :: ndiv2, i,j,k
    integer                         :: iseg,jseg,kseg
!*********************************************************************
!   write(*,*) 'viewaa',iseg,jseg,nset

    invpi = 1.0/(4.0*atan(1.0))
    fij  = 0.0
!
    outer_loop: do i=1,ndiv2
     ln = nia(:,i)
     do ii = 1,4
     x(ii) = xia(1,ln(ii))
     y(ii) = xia(2,ln(ii))
     z(ii) = xia(3,ln(ii))
     enddo
     call geom3d (x,y,z,xcl(1,1),xcl(2,1),xcl(3,1),vni(1), &
              vni(2),vni(3),areai)
!
     inner_loop: do j=1,ndiv2
      ln = nja(:,j)
      do jj = 1,4
      x(jj)  = xja(1,ln(jj))
      y(jj)  = xja(2,ln(jj))
      z(jj)  = xja(3,ln(jj))
      enddo
      call geom3d (x,y,z,xcl(1,2),xcl(2,2),xcl(3,2),vnj(1), &
               vnj(2),vnj(3),areaj)
      if (nset .NE. 0) then
!.........can differential surface i see differential surface j
!........ considering the subset of third surface obstructions
      do 30 k=1,nset
       kseg = kset(k)
       call sectn (xnc,nds,xcl,xng,1,2,kseg,iflag)
       if (iflag.EQ.1) cycle inner_loop
  30   continue
      endif
!.........calculate: r vector; cos bi; cos bj; dfij
       r     = xcl(:,2) - xcl(:,1)
       rmag  = sqrt(dot_product(r,r))
       cosbi = dot_product(r,vni)/rmag
       cosbj =-dot_product(r,vnj)/rmag
      if ((cosbi.GT.0.0).AND.(cosbj.GT.0.0))          &
       fij  = fij + invpi*cosbi*cosbj*areai*areaj/(rmag*rmag)
!
     end do inner_loop
!
    end do outer_loop
!
    indexi = jseg - iseg + 1
    frow(indexi) = fij
!
    end subroutine viewaa

!*********************************************************************
    subroutine viewcc (xil,xjl,frow,iseg,jseg)
!*********************************************************************
! contour integration algorithm for 3d geometries
!*********************************************************************
    use setreal_h
    use primary_h
!*********************************************************************
```

```
      real (setpr), dimension(3,maxl+1) :: xil
      real (setpr), dimension(3,maxl+1) :: xjl
      real (setpr), dimension(numel)    :: frow
      real (setpr)                      :: f,dxi,dyi,dzi,dxhi,dyhi,dzhi
      real (setpr)                      :: dxj,dyj,dzj,r,rlog
      real (setpr)                      :: twopi
      real (setpr), parameter           :: half = 0.5
      integer                           :: iseg,jseg
      integer                           :: i,j,indexi
!***********************************************************************
      twopi  = 8.0*atan(1.0)
      f      = 0.0
      do 20 i=1,maxl
        dxi  = xil(1,i+1) - xil(1,i)
        dyi  = xil(2,i+1) - xil(2,i)
        dzi  = xil(3,i+1) - xil(3,i)
        dxhi = dxi * half
        dyhi = dyi * half
        dzhi = dzi * half
        do 10 j=1,maxl
          dxj = xjl(1,j+1) - xjl(1,j)
          dyj = xjl(2,j+1) - xjl(2,j)
          dzj = xjl(3,j+1) - xjl(3,j)
          r   = sqrt( (xjl(1,j) + dxj*half - xil(1,i) - dxhi)**2 +  &
                  (xjl(2,j) + dyj*half - xil(2,i) - dyhi)**2 + &
                  (xjl(3,j) + dzj*half - xil(3,i) - dzhi)**2)
          rlog = log(r)
          f    = f + rlog*(dxi*dxj + dyi*dyj + dzi*dzj)
   10   continue
   20 continue
      indexi = jseg - iseg + 1
      frow(indexi) = f / twopi
!
      end subroutine viewcc



!***********************************************************************
      subroutine viewms (xnd,nds,frow,iseg,jseg)
!***********************************************************************
!***   Mitalas & Stephenson Method for 3D Geometries                  ***
!***********************************************************************
      use setrea_h
      use primary_h
!***********************************************************************
      real (setpr), dimension(3,numnp) :: xnd
      real (setpr), dimension(numel)   :: frow
      real (setpr), dimension(4)       :: ai,bi,gi
      real (setpr), dimension(4)       :: aj,bj,gj
      real (setpr), dimension(4)       :: xleni, xlenj
      real (setpr), dimension(3)       :: dr,ds,dt,dx, p
      real (setpr)                     :: fij,r,s,t,xdiv,dl,summ
      real (setpr)                     :: costh,cosph,theta,phi,omega
      real (setpr)                     :: v,add,eta
      real (setpr)                     :: pi, twopi
!
      integer, dimension(6,numel)      :: nds
      integer                          :: iend,jend,i,j,k,n1,n2
      integer                          :: n1j,n2j,indexi
      integer                          :: iseg,jseg
      integer                          :: ndiv_new
!***********************************************************************
      ndiv_new  = ndiv*5
      pi        = 4.0*atan(1.0)
      twopi     = 8.0*atan(1.0)
      iend      = 4
      jend      = 4
      do 10 i=1,iend
        n1      = nds(i,iseg)
        n2      = nds(i+1,iseg)
      dx      = xnd(:,n2) - xnd(:,n1)
      xleni(i) = sqrt(dot_product(dx,dx))
      ai(i)   = dx(1) / xleni(i)
      bi(i)   = dx(2) / xleni(i)
      gi(i)   = dx(3) / xleni(i)
  10 continue
    do 20 j=1,jend
      n1      = nds(j,jseg)
      n2      = nds(j+1,jseg)
      dx      = xnd(:,n2) - xnd(:,n1)
      xlenj(j) = sqrt(dot_product(dx,dx))
      aj(j)   = dx(1) / xlenj(j)
      bj(j)   = dx(2) / xlenj(j)
      gj(j)   = dx(3) / xlenj(j)
  20 continue
    fij = 0.0
    do 60 j=1,jend
      n1j     = nds(j,jseg)
      n2j     = nds(j+1,jseg)
      dr      = xnd(:,n2j) - xnd(:,n1j)
      r       = sqrt(dot_product(dr,dr))
      do 50 i=1,iend
        eta    = ai(i)*aj(j) + bi(i)*bj(j) + gi(i)*gj(j)
        n1     = nds(i,iseg)
        n2     = nds(i+1,iseg)
        xdiv   = real(ndiv_new)
        dx     = (xnd(:,n2) - xnd(:,n1)) / xdiv
        dl     = xleni(i) / xdiv
        p      = xnd(:,n1) - dx/2.0
        summ   = 0.0
        do 30 k=1,ndiv_new
          p     = p   + dx
          ds    = p   - xnd(:,n1j)
          s     = sqrt(dot_product(ds,ds))
          dt    = p   - xnd(:,n2j)
          t     = sqrt(dot_product(dt,dt))
          costh  = dot_product(ds, dr) / (r*s)
          cosph  = -dot_product(dt, dr) / (r*t)
          costh  = min( 1.0,max(-1.0,costh))
          cosph  = min( 1.0,max(-1.0,cosph))
          theta  = acos(costh)
          phi    = acos(cosph)
          omega  = pi - theta - phi
          v      = s*sin(theta)
          add    = dl*(t*cosph*log(t) + s*costh*log(s) + v*omega - r)
          summ   = summ + add
  30    continue
        fij = fij + eta*summ
  50  continue
  60 continue
    indexi      = jseg - iseg + 1
    frow(indexi) = fij / twopi
!
    end subroutine viewms


!*************************************************************************
    subroutine viewr (nnodemax,nelemax,nds,sigma,solar,  &
                  garea,f,emisf,tranf,rowerr)
!*************************************************************************
!*** subroutine to calculate remaining view factors by reciprocity   ***
!*************************************************************************
    use setreal  h
    use primary  h
!*************************************************************************
    integer                              :: nnodemax,nelemax
    integer, dimension(6,numel)              :: nds
    integer, dimension(5)                  :: jdum
!.......... ............... .......... ........... ..............
    real (setpr)                          :: sigma, solar
```

```
     real (setpr), dimension(numel)              :: garea
     real (setpr), dimension(numel)              :: emisf
     real (setpr), dimension(numel)              :: tranf
     real (setpr), dimension(numel)              :: rowerr
     real (setpr), dimension( (numel*numel+numel)/2 ) :: f
!.............................................................
     integer                               :: k,kk,l
     integer                               :: i,j,ndo,nleft
     integer,     dimension(1)                  :: imax
     real (setpr), dimension(5)                  :: fdum
     real (setpr)                           :: avgerr
     real (setpr)                           :: maxerr
     real (setpr)                           :: norm1err
     real (setpr)                           :: norm2err
     real (setpr)                           :: stderr
     real (setpr)                           :: summ
!*******************************
!.....declare local subprograms
!*******************************
     integer indx
     external indx
!**********************************************************************
!*******************************
!.....create binary file
!*******************************
     write (jabs) nnodemax
     write (jabs) nelemax
     write (jabs) numel
     write (jabs) sigma
     write (jabs) solar
     do 10 i=1,numel
      write(jabs) (nds(j,i),j=1,4)
 10   continue
     write (jabs) garea
     write (jabs) emisf
     write (jabs) tranf
     write (jabs) f
     call timing (3)
!*******************************
!.....determine row sum error
!*******************************
     write (jout,110)
     do 60 i=1,numel
      summ = 0.0
      do 50 j=1,numel
       kk  = indx( i,j)
       summ= summ + f(kk)/garea(i)
  50   continue
      rowerr(i) = abs(1. - summ)
      write (jout,120) i,rowerr(i)
  60 continue
!****************************************
!.....determine row sum error statistics
!****************************************
     maxerr   = maxval( rowerr )
     imax     = maxloc( rowerr )
     norm1err = sum( rowerr )
     norm2err = sqrt( dot_product(rowerr,rowerr) )
     avgerr   = norm1err/real(numel)
     rowerr   = rowerr - avgerr
     stderr   = sqrt( dot_product(rowerr,rowerr)/real(numel) )
     write (jout,125) maxerr,imax(1),norm1err,norm2err,  &
                 avgerr,stderr
!*******************************
!.....write view factor matrix
!*******************************
!    write (jout,150)
!    ndo   = numel / 5
!    nleft = numel - 5*ndo
!    do 100 k=1,numel
!     write (jout,140) k
!     do 80 i=1,ndo
!       do 70 l=1,5
!         j      = (i-1)*5 + l
!         kk     = indx(k,j)
!         jdum(l) = j
!         fdum(l) = f(kk)/garea(k)
! 70    continue
!       write (jout,130) (jdum(l),fdum(l),l=1,5)
! 80  continue
!.......last line
!     if (nleft.LE.0) go to 100
!     do 90 l=1,nleft
!       j      = ndo*5 + l
!       kk     = indx(k,j)
!       jdum(l) = j
!       fdum(l) = f(kk)/garea(k)
! 90  continue
!    write (jout,130) (jdum(l),fdum(l),l=1,nleft)
! 100 continue
    call timing (4)
!
 110 format(// '**************************************************'/ &
           '********** R O W   S U M   E R R O R **********'/ &
           '**************************************************'// &
           '              row    1.-row sum'/)
 120 format (15x,i5,1pe12.4)
 125 format( //1x,'**********************************************' &
           /1x,'******** Row-Sum Error Statistics **********' &
           /1x,'**********************************************' &
           /1x,'   sup norm = ',1p,e10.3,' at row =',i5 &
           /1x,'   L1 norm  = ',  e10.3 &
           /1x,'   L2 norm  = ',  e10.3 &
           /1x,'   mean     = ',  e10.3 &
           /1x,'   std-dev  = ',  e10.3)
 130 format (5(i4,1x,1p e10.3))
 140 format (i4,1x,'  th row')
 150 format(// '**************************************************'/ &
           '********* V I E W   F A C T O R S ***********'/ &
           '**************************************************'//)
    end subroutine viewr




!*************************************************************************
    subroutine view3d(nds,nocpl,nia,nja,kblk,kset,xnd,garea,xil,xjl, &
            xia,xja,xng,xcg,frow,f)
!*************************************************************************
!*** determine view factors for 3d geometries                         ***
!*************************************************************************

    use setreal_h
    use primary_h
    use compar
    implicit none
!.... .................. ........................... ...... ......... ..
! ******************************
! *** Interface Prototypes ***
! ******************************

    interface
      subroutine couple (nocpl,iseg,jseg,icouple)
        integer, dimension(2,*)      :: nocpl
        integer                      :: iseg,jseg,icouple
      end subroutine couple
      subroutine edge (xnd,nds,iseg,jseg,iedge)
        use setreal_h
        real (setpr), dimension(3,*) :: xnd
        integer, dimension(6,*)      :: nds
      integer                  :: iseg,jseg,iedge
    end subroutine edge
    subroutine fwrite (frow,f,iseg)
      use setreal_h
      real (setpr), dimension(*)  :: frow
      real (setpr), dimension(*)  :: f
      integer                  :: iseg
    end subroutine fwrite
    subroutine grida (xnd,nds,xpa,nda,kseg)
      use setreal_h
      real (setpr), dimension(3,*) :: xnd
      real (setpr), dimension(3,*) :: xpa
      integer, dimension(6,*)      :: nds
      integer, dimension(4,*)      :: nda
      integer                  :: kseg
    end subroutine grida
    subroutine gridl (xnd,nds,xpl,kseg)
      use setreal_h
      real (setpr), dimension(3,*) :: xnd
      real (setpr), dimension(3,*) :: xpl
      integer, dimension(6,*)      :: nds
      integer                  :: kseg
    end subroutine gridl
    subroutine obstr (xnd,nds,xng,xcg,kblk,kset,iseg,jseg,nset)
      use setreal_h
      real (setpr), dimension(3,*) :: xnd
      real (setpr), dimension(3,*) :: xng
      real (setpr), dimension(3,*) :: xcg
      integer, dimension(6,*)      :: nds
      integer, dimension(*)        :: kblk
      integer, dimension(*)        :: kset
      integer                  :: iseg,jseg,nset
    end subroutine obstr
    subroutine see (xnd,nds,xng,iseg,jseg,isee,iedge)
      use setreal_h
      real (setpr), dimension(3,*) :: xnd
      real (setpr), dimension(3,*) :: xng
      integer, dimension(6,*)      :: nds
      integer                  :: iseg,jseg,isee,iedge
    end subroutine see
    subroutine viewaa (xnd,nds,xia,xja,nia,nja,xng,kset,frow, &
                iseg,jseg,ndiv2,nset)
      use setreal_h
      real (setpr), dimension(3,*) :: xnd
      real (setpr), dimension(3,*) :: xia,xja
      real (setpr), dimension(3,*) :: xng
      real (setpr), dimension(*)   :: frow
      integer, dimension(6,*)      :: nds
      integer, dimension(4,*)      :: nia,nja
      integer, dimension(*)        :: kset
      integer                  :: iseg,jseg,ndiv2,nset
    end subroutine viewaa
    subroutine viewcc (xil,xjl,frow,iseg,jseg)
      use setreal_h
      real (setpr), dimension(3,*) :: xil,xjl
      real (setpr), dimension(*)   :: frow
      integer                  :: iseg,jseg
    end subroutine viewcc
    subroutine viewms (xnd,nds,frow,iseg,jseg)
      use setreal_h
      real (setpr), dimension(3,*) :: xnd
      real (setpr), dimension(*)   :: frow
      integer, dimension(6,*)      :: nds
      integer                  :: iseg,jseg
    end subroutine viewms
!****************************
   end interface
!****************************
!**********************************************************************
!***     variable declaration section                         ***
```

```
!****************************************************************************
      integer, dimension(6,numel)     :: nds   ! nds(6,numel)
        integer, dimension(2,*)      :: nocpl ! nocpl(2,ncpl+1)
      integer, dimension(4,ndiv*ndiv)     :: nia   ! nia(4,ndiv*ndiv)
      integer, dimension(4,ndiv*ndiv)     :: nja   ! nja(4,ndiv*ndiv)
      integer, dimension(numel)       :: kblk  ! kblk(numel)
      integer, dimension(numel)       :: kset  ! kset(numel)
      real (setpr), dimension(3,numnp) :: xnd   ! xnd(3,numnp)
      real (setpr), dimension(numel)   :: garea ! garea(numel)
      real (setpr), dimension(3,maxl+1) :: xil   ! xil(3,maxl+1)
      real (setpr), dimension(3,maxl+1) :: xjl   ! xjl(3,maxl+1)
      real (setpr), dimension(3,maxa) :: xia   ! xia(3,maxa)
      real (setpr), dimension(3,maxa) :: xja   ! xja(3,maxa)
      real (setpr), dimension(3,numel) :: xng   ! xng(3,numel)
      real (setpr), dimension(3,numel) :: xcg   ! xcg(3,numel)
      real (setpr), dimension(numel)   :: frow  ! frow(numel)
      real (setpr), dimension((numel**2+numel)/2) :: f     ! f((numel**2+numel)/2)
      real*8      , dimension((numel**2+numel)/2) :: ftemp ! ftemp(numel,numel)
      real (setpr), dimension((numel**2+numel)/2) :: fsum  ! fsum(numel,numel)
      real*8      , dimension((numel**2+numel)/2) :: ftsum ! fsum(numel,numel)
      integer                  :: sendtype, recvtype, sendcnt
      integer                  :: ndiv2,iseg,jseg
      integer                  :: isee,iedge,nset
      integer                  :: icouple,indexi
      integer                  :: iviewcc,iviewms,iviewaa
      integer                  :: icount, ierr
      save   iedge, isee
!...........................................................
!     Parameters.
      real (setpr), parameter :: one  = 1.0
      real (setpr), parameter :: zero = 0.0
!****************************************************************************
    ndiv2   = ndiv*ndiv
    iviewcc = 0
    iviewms = 0
    iviewaa = 0
    icount  = -1
!**********************************
!.....view factor iseg outer  loop
!**********************************
!$DOACROSS
!$&LOCAL(iseg,jseg,indexi,nia,nja, xil, xjl, xia, xja, frow,
!$&  icouple, nset, isee, iedge)
!$&REDUCTION(iviewcc,iviewaa,iviewms)
!$&MP_SCHEDTYPE=GSS
    outer_loop: do iseg=1,numel-1
       if ( mod(iseg,10) .EQ. 0) then
         if(myPE.eq.root) then
        write(jlog,'(i6," surfaces have been processed. ",$)')  &
                  iseg
        write(jlog,'(" viewcc = ",i10," viewms = ",i10,  &
                  " viewaa = ",i10)')iviewcc,iviewms,iviewaa
        endif
       endif
        frow     = 0.0
       frow( 1 ) = 0.0
       call gridl (xnd,nds,xil,iseg)
       call grida (xnd,nds,xia,nia,iseg)
!************************************
!...... .view factor jseg inner_loop
!************************************
       inner_loop: do jseg=iseg+1,numel
         nset = 0
         if (ncpl .GT. 0) then
!.........  ...check if iseg and jseg couple
           call couple(nocpl,iseg,jseg,icouple)
           if ( icouple .EQ. 0) then
             indexi = jseg - iseg + 1
             frow( indexi ) = 0.0
             cycle inner_loop
```

```
            endif
          endif
           icouple = icouple + 1
           if(mod(icouple,numPEs).ne.myPE) cycle inner_loop
!*********************************************************************
!....can surface i see surface j ignoring third surface obstructions
!*********************************************************************
          call see (xnd,nds,xng,iseg,jseg,isee,iedge)
!          write(*,*) 'view3d',iseg,jseg,isee,iedge,nset
!.......... isee = 0 >> total self-blockage
!...... ... isee = 1 >> no self-blockage
!...... ... isee =-1 >> partial self-blockage
!... ...... iedge= 1 >> iseg and jseg do not share a common edge
! .. ...... iedge= 2 >> iseg and jseg share a common edge
          if (isee .EQ. 0) then
!*********************************************************************
! ............surface iseg and jseg cannot see each other
!*********************************************************************
            indexi = jseg - iseg + 1
            frow( indexi ) = 0.0
          elseif ((nblk.EQ.0) .AND. (isee.EQ.1)) then
!*********************************************************************
!....... .....no third surface blockage - - use contour integration
!*********************************************************************
           if (iedge.EQ.1) then
             iviewcc = iviewcc + 1
             call gridl  (xnd,nds,xjl,jseg)
             call viewcc (xil,xjl,frow,iseg,jseg)
           elseif (iedge.EQ.2) then
             iviewms = iviewms + 1
             call viewms (xnd,nds,frow,iseg,jseg)
           endif
          elseif ((nblk.EQ.0) .AND. (isee.EQ.-1)) then
!*********************************************************************
!. ...... ....partial self-blockage     - - use area integration
!*********************************************************************
           iviewaa = iviewaa + 1
           call grida (xnd,nds,xja,nja,jseg)
           call viewaa (xnd,nds,xia,xja,nia,nja,xng,kset,frow,  &
                    iseg,jseg,ndiv2,nset)
          else
!*********************************************************************
!.............identify the subset of the obstructing surfaces k that
!..... ...... obstruct the view between surfaces i and j
!*********************************************************************
           call obstr (xnd,nds,xng,xcg,kblk,kset,iseg,jseg,nset)
           if ((nset.EQ.0) .AND. (isee.EQ.1)) then
!..............no third surface blockage - - use contour integration
             if (iedge.EQ.1) then
              iviewcc = iviewcc + 1
              call gridl  (xnd,nds,xjl,jseg)
              call viewcc (xil,xjl,frow,iseg,jseg)
             elseif (iedge.EQ.2) then
              iviewms = iviewms + 1
              call viewms (xnd,nds,frow,iseg,jseg)
             endif
           else
!......... .....third surface blockage - - use area integration
             iviewaa = iviewaa + 1
             call grida (xnd,nds,xja,nja,jseg)
             call viewaa (xnd,nds,xia,xja,nia,nja,xng,kset,frow,  &
                       iseg,jseg,ndiv2,nset)
           endif
          endif
!*******************************************
!... ...end of view factor jseg inner_loop
!*******************************************
         end do inner_loop
!*********************************************************************
! .......after having calculated the necessary elements in row Iseg,
!       now store in global array
!$PAR CRITICALSECTION
        call fwrite (frow,f,iseg)
!$PAR ENDCRITICALSECTION
!*********************************************
! ......end of view factor iseg outer_loop
!*********************************************
      end do outer_loop
!*********************************************
!......pick up last element of f matrix
!*********************************************
!C$PAR CRITICALSECTION
      frow( * ) = 0.0
      call fwrite (frow,f,numel)
!C$PAR ENDCRITICALSECTION
!**************************************************
!......Sum the information from all the processors
!**************************************************

      recvtype = MPI_DOUBLE_PRECISION
      sendtype = recvtype
      sendcnt  = maxo
      if(kind(one).eq.kind(1.0d0)) then
        call MPI_Allreduce( f  fsum, sendcnt, sendtype, MPI_SUM,  &
                  MPI_COMM_WORLD, ierr )
      f = fsum
      else
      ftemp = dble(f)
        call MPI_Allreduce( ftemp, ftsum, sendcnt, sendtype, MPI_SUM,  &
                  MPI_COMM_WORLD, ierr )
      f = ftsum
      endif

      if(myPE.eq.root) then
      write(jlog,'(i6," surfaces have been processed. ",$)')numel
      write(jlog,'(" viewcc = ",i10 " viewms = ",i10,  &
               " viewaa = ",i10)')iviewcc,iviewms iviewaa
      endif
!
     end subroutine view3d



!***************************************************************************
     subroutine factor(f,emisf,tranf,garea,numel,nf_out,nf_log)
!***************************************************************************
!    PURPOSE:
!      Carries out an LU factorization of AMAT
!    INPUT ARGUMENTS:
!      nf_log = unit number for log file
!
!...............................................................
     use setreal_h
     use compar
! ***************************
! *** Interface Prototypes ***
! ***************************
     interface
       subroutine check_alloc(string,error,nerr,nf_out)
         character*(*) string
         integer nerr,nf_out
         integer error(nerr)
       end subroutine check_alloc
       subroutine check_dealloc(string,error,nerr,nf_out)
         character*(*) string
         integer nerr,nf_out
         Integer error(nerr)
       end subroutine check_dealloc
       subroutine check_ios (string1,string2,ios,nf_out)
```

```
        character*(*) string1
        character*(*) string2
        integer ios,nf_out
      end subroutine check_ios
      subroutine get_unit(nf_unit)
        integer nf_unit
      end subroutine get_unit
! ****************************
    end interface
! ****************************
!______________________________________________________________________
!   Local variables.
!
    integer nf_bin,nf_fact,nf_log,nf_out,i,j,k,ios
    integer, dimension(15)           :: error
    logical exists,connected
!
    integer numel,nnodemax,nelemax,maxb,info
    real (setpr)                     :: sigma, solar
    real (setpr)                     :: deltakj,Fkj
!.....................................................
!   Allocatable arrays
    integer    ,dimension(:,:),allocatable  :: nds  ! nds(4,numel)
    integer    ,dimension(:) ,allocatable :: IPIV ! IPIV(numel)
    real (setpr),dimension(numel)  :: garea! garea(numel)
    real (setpr),dimension((numel*numel + numel)/2)   :: f
    real (setpr),dimension(numel)  :: emisf! emisf(numel)
    real (setpr),dimension(numel)  :: tranf! tranf(numel)
    real (setpr),dimension(:,:),allocatable :: AMAT   !AMAT(numel,numel)
    real       ,dimension(:,:),allocatable :: AMAT_S !Single Precision
    real (setpr),dimension(numel)  :: emisf_inv
    real (setpr),dimension(numel)  :: garea_inv
!.....................................................
!   External functions
    integer  indx
    external indx
!.....................................................
!   Parameters.
    real (setpr), parameter :: one  = 1.0
    real (setpr), parameter :: zero = 0.0
    logical     , parameter :: SCALAPACK = .TRUE.
!.....................................................
!   Data statements
    data exists/.FALSE./,connected/.FALSE./
    data nf_bin/26/,nf_fact/27/
!_____________________________________________________
!
!   *********************
!   allocate HEAP memory
!   *********************
      maxb = (numel*numel + numel)/2
      allocate ( AMAT(numel,numel), stat=error( 1)  )
      allocate (       IPIV(numel), stat=error( 2)  )
      allocate ( AMAT_S(numel,numel), stat=error( 3)  )
!.....is there sufficient memory to solve the problem?
      call check_alloc('FACTOR',error,3,nf_out)

        PIV = 0

!   **************
!   Construct AMAT
!   **************
      if(myPE.eq.root) then
      write (nf_log,'(" CONSTRUCTING AMAT.")')
       endif
      emisf_inv = one/emisf
      garea_inv = one/garea
      row_loop: do k=1,numel
        column_loop: do j=1,numel
          if (k .EQ. j) then
        deltakj = one
       else
        deltakj = zero
       endif
       Fkj = f(ndx(k,j))*garea_inv(k)
       AMAT(k,j) = (deltakj*emisf_inv(j)) - &
             (one-emisf(j)-tranf(j))*emisf_inv(j)*Fkj
     end do column_loop
    end do row_loop

    if (scalapack) then

    call tmatrix(AMAT, IPIV, numel)

    elseif (myPE.eq.root) then
!   *****************************************************************
!   Carry out LU factorization of AMAT
!   Check for the real-type and use appropriate built in BLAS/LINPACK
!   otherwise copy to single precision and call built-in libraries
!   *****************************************************************
     if(myPE.eq.root) then
    write (nf_log,'(" BLOCKED LU FACTORIZATION BEGUN ")')
     endif
     if(kind(one).eq.kind(1.0d0)) then
     call dgetrf( numel,numel,AMAT,numel,IPIV,info )
     elseif(kind(one).eq.kind(1.0)) then
     call sgetrf( numel,numel,AMAT,numel,IPIV,info )
     else
     AMAT_S = real(AMAT)
     call sgetrf( numel,numel,AMAT_S,numel,IPIV,info )
     AMAT   = AMAT_S
     endif
     if(myPE.eq.root) then
    write (nf_log,'(" BLOCKED LU FACTORIZATION COMPLETED.")')
     endif
!   ************************
!   Check solution flag INFO
!   ************************
!   INFO = 0:  successful exit
!      < 0:  if INFO = -i, the i-th argument had an illegal value
!      > 0:  if INFO = i, U(i,i) is exactly zero. The factorization
!            has been completed, but the factor U is exactly
!            singular, and division by zero will occur if it is used
!            to solve a system of equations
    if ( info .NE. 0) then
     write(nf_log,'(//" ****ERROR in FACTOR!")')
     write(nf_log,'( /" LU factorization of AMAT failed.")')
     write(nf_log,'( /" INFO = ",i6)') info
     write(nf_log,'(" CHADVIEW execution aborted!")')
     write(nf_log,"(//)")
     stop
    endif

!   **************************
!   write out LU factorization
!   **************************
    call get_unit(nf_fact)
    open (unit = nf_fact,file='factor.bin',form='unformatted', &
        action='write', status='unknown', iostat=ios)
    call check_ios ('FACTOR','factor.bin',ios,nf_out)
    write (nf_log,'(" Writing LU factorization to factor.bin")')
    write (nf_fact) IPIV
    write (nf_fact) AMAT
    rewind (nf_fact)
    close (unit=nf_fact,status='keep')

    endif
!   ********************
!   deallocate HEAP memory
!   ********************
```

```
       deallocate (     AMAT, stat=error( 1)  )
       deallocate (     IPIV, stat=error( 2)  )
       deallocate (    AMAT_S, stat=error(3)  )
       call check_dealloc('FACTOR',error,3,nf_out)
!
      end subroutine factor


      subroutine tmatrix(Amat, IPIV, numel)
!     ----------------------------
!     simple test program to test scalapack
!     ----------------------------
      implicit none
!     -----------------------
!     storage for parallel scalapack
!     -----------------------
      INTEGER          DLEN_
      PARAMETER        ( DLEN_ = 9 )
      INTEGER          CTXT_, M_, N_, MB_, NB_
      PARAMETER        ( CTXT_ = 2, M_ = 3, N_ = 4, MB_ = 5, NB_ = 6 )
      INTEGER          RSRC_, CSRC_, LLD_
      PARAMETER        ( RSRC_ = 7, CSRC_ = 8, LLD_ = 9 )

      doubleprecision          :: t1,t2
      doubleprecision,external    mpi_wtime

      integer :: numel

      doubleprecision, parameter :: one = 1.0d0
      doubleprecision, parameter :: zero = 0.0d0
      doubleprecision          :: alpha,beta, norm2

      doubleprecision, parameter :: UNDEFINED = -999.999
      doubleprecision, dimension(numel,numel) :: Amat
      doubleprecision, allocatable, dimension(:) :: A

      integer, dimension(DLEN_) :: descA,descX,descB

      doubleprecision, allocatable, dimension(:) :: work
      integer, allocatable, dimension(:)        :: pivot
      integer     ,dimension(numel)            :: IPIV ! IPIV(numel)

      integer :: m,n,  mb,nb,  rsrc,csrc, lld,icontxt, info
      integer :: myrow,mycol,  nprow,npcol,  myid,nproc
      integer :: istart,iend,isize,  Loc_row,Loc_col,nrhs
      integer :: Asize, vsize

      doubleprecision  :: Aij,Xij
      character(len=4) :: suffix

      integer :: i,j,  irow,icol, lr,lc,ipos
      integer :: ia,ja,  incX,incY, ineed
      logical :: ismine, isok, is_small

      integer, external :: numroc

      integer :: irpmt,icpmt, noutA,noutB,noutX


!     ---------------------
!     setup parallel environment
!     ---------------------

      call blacs_pinfo( myid, nproc )
      if (nproc .lt. 1) then
        write(*,'("blacs_pinfo returns: myid,nproc ",2(1x,i9))') myid,nproc
        stop "blacs not setup "
      endif
      call blacs_get(-1,0,icontxt)
```

```
!  ---------------
!  setup processor grid
!  ---------------
   nprow = -1
   npcol = -1
   do nprow=int(sqrt(dble(nproc)))+1,1,-1
      npcol = nproc/nprow
      if (nprow*npcol.eq.nproc) exit
   enddo

   call blacs_gridinit( icontxt, 'row-major', nprow,npcol)
   call blacs_gridinfo( icontxt, nprow, npcol, myrow, mycol)

   if (myid.eq.0) then
      write(*,'("blacs started: nprow,npcol,nproc ",3(1x,i9))') &
     nprow,npcol nproc
   endif


!  ----------------------------------
!  may need to open different files like 'out38.001'
!  for processor '001'
!  ----------------------------------
   suffix = '.000'
   suffix(4:4) = char(ichar('0') + mod( myid,    10))
   suffix(3:3) = char(ichar('0') + mod( myid/10, 10))
   suffix(2:2) = char(ichar('0') + mod( myid/100,10))


!  ----------
!  setup descriptor
!  ----------
   m = numel
   n = m
   mb = 50
   nb = 50
   rsrc = 0
   csrc = 0

   if (myid.eq.0) then
     write(*,'("m,n ",2(1x,i6)," mb,nb ",2(1x,i6))') &
     m,n,  mb,nb
   endif

   lld = max(1.numroc( m. mb, rsrc,myrow, nprow ))

   call descinit( descA, m,n, mb nb, rsrc,csrc, icontxt, lld, info )
   if (info.ne.0) then
     write(*,'("** descinit for descA returns info = ",1x.i9)') info
     stop '** error ** '
   endif

   call descinit( descX, m,1, mb,nb, rsrc,csrc, icontxt, lld, info )
   if (info.ne.0) then
     write(*,'("** descinit for descX returns info = ",1x,i9)') info
     stop '** error ** '
   endif

   call descinit( descB, m,1, mb,nb, rsrc,csrc, icontxt, lld, info )
   if (info.ne.0) then
     write(*,'("** descinit for descB returns info = ",1x,i9)') info
     stop '** error ** '
   endif
!  ----------
!  check storage
!  ----------
   Loc_row = max(1,numroc(m,mb,rsrc,myrow,nprow))
   Loc_col = max(1 numroc(n,nb,rsrc,mycol,npcol))
   ineed = Loc_row * Loc_col
   Asize = ineed+1
```

```
      vsize = Loc_row+1

      allocate (A(Asize))
      allocate (work(vsize))
      allocate (pivot(vsize))

      do i=1,Asize
       A(i) = UNDEFINED
      enddo

!     ----------------------
!     initialize lower part of matrix
!     ----------------------
      do j=1,n
      do i=j,n
         call infog2l( i,j,  descA  nprow,npcol  myrow,mycol, &
                    lr,lc,  irow,icol )
        ismine = (irow .eq. myrow).and.(icol .eq. mycol)
        if (ismine) then
        ipos = lr + (lc-1)*descA(LLD_)
        A(ipos) = Amat(i,j)
        endif
      enddo
      enddo


!     --------------------------------
!     copy lower part to upper triangular part
!     --------------------------------

      call blacs_barrier( icontxt, 'All')
      t1 = mpi_wtime()

      do j=1,n
       istart = j
       iend = m
       isize = iend - istart + 1
        incX = 1
        incY = descA(M_)

        if (isize .ge. 1) then
        call PDCOPY( isize, A,istart,j,descA, incX, &
                     A,j,istart,descA, incY )
        endif
       enddo

      call blacs_barrier( icontxt  'All')
      t2 = mpi_wtime()
      if (myid.eq.0) then
        write(*,'("time for transpose copy is ",1x,1pd14.4)') (t2-t1)
      endif

!     ---------------
!     perform factorization
!     ---------------
      call blacs  barrier( icontxt, 'All')
      t1 = mpi  wtime()

       info = 0
      ia = 1
      ja = 1
      call PDGETRF( m,n, A,ia,ja,descA, pivot, info )
      if (info.ne.0) then
        write(*,'("PDGETRF returns info = ",1x,i9)') info
        stop "** error ** "
      endif

      call blacs_barrier( icontxt, 'All')
      t2 = mpi  wtime()
```

```
    if (myid.eq.0) then
      write(*,'(''time for PDGETRF is '',1pe14.4)') (t2-t1)
    endif

!   ----------------
!   stop parallel environment
!   ----------------
    call blacs_gridexit( icontxt )


    return
    end subroutine tmatrix
```

C/C9800401

INTERNAL DISTRIBUTION

1-2.	C. A. Valentine
3.	Sreekanth Pannala
4.	Eduardo D'Azevedo
5.	Thomas Zacharia
6.	Central Research Library
7.	ORNL Laboratory Records - RC
8.	ORNL Laboratory Records - OSTI

EXTERNAL DISTRIBUTION

9.	Steve MacDonald, adapco, 60 Broadhollow Road, Melville, NY 11747
10.	Dr. Gary Strumolo, Ford Motor Company, 2000 Rotunda Drive, Mail Drop 2122, Dearborn, MI 48121
11.	Dr. Richard Sun, DaimlerChrysler, 800 Chrysler Drive, East, MC:481-33-01, Auburn Hills, MI 48326-2757
12.	Dr. V. Sumantran, General Motors, 30500 Mount Road, MC: 480106256, Warren, MI 48090-9055
13.	Dr. Adrian Tentner, ANL, 9700 South Cass Avenue, Argonne, Illinois 60439
14.	Dr. David Weber, ANL, 9700 South Cass Avenue, Argonne, Illinois 60439
15.	Dr. Hank Domanus, ANL, 9700 South Cass Avenue, Argonne, Illinois 60439
16.	Dr. Constantine Tzanos, ANL, 9700 South Cass Avenue, Argonne, Illinois 60439