LA-13406

High Explosive Programmed Burn in the Flag Code

Los Alamos
NATIONAL LABORATORY

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

Edited by Patricia W. Mendius, Group CIC-1
Prepared by M. Ann Nagy, Group X-CI

An Affirmative Action/Equal Opportunity Employer

LA-13406

UC-741
Issued: February 1998

High Explosive Programmed Burn in the Flag Code

David Mandell
Donald Burton
Carl Lund

Los Alamos
NATIONAL LABORATORY
Los Alamos, New Mexico 87545

HIGH EXPLOSIVE PROGRAMMED BURN IN THE FLAG CODE

by

David Mandell, Donald Burton, and Carl Lund

ABSTRACT

The models used to calculate the programmed burn high-explosive lighting times for two- and three-dimensions in the FLAG code are described. FLAG uses an unstructured polyhedra grid. The calculations were compared to exact solutions for a square in two dimensions and for a cube in three dimensions. The maximum error was 3.95 percent in two dimensions and 4.84 percent in three dimensions. The high explosive lighting time model described has the advantage that only one cell at a time needs to be considered.

I. INTRODUCTION

This report describes the high explosive (HE) programmed burn model implemented into the FLAG code. In the programmed burn model, the times at which the HE in each mesh cell detonates is calculated a priori by calculating the arrival times of waves emanating from the prescribed detonation point or points. The HE chemical energy is then sourced into the hydrodynamics. The times at which the mesh cells are burned in two- and three-dimensions are found from the model developed by Lund (Lund, 1986).

FLAG (Burton, 1992,1994) is a one-, two-, and three-dimensional Lagrangian hydrodynamics code. The code contains a number of hydrodynamics, material strength, and equation of state models. Additional models are being added, including finite-element, arbitrary Lagrange-Eulerian (ALE), free Lagrange hydrodynamics, and others. In two dimensions the HE burn times must be found at the vertices of triangles, and in three dimensions the burn times need to be calculated at the vertices of tetrahedron. In three dimensions FLAG uses arbitrary polyhedral, that are broken up, for the internal calculations, into tetrahedron. FLAG is coded in object-oriented Fortran.

In the next section, the general equations for the Lund model are presented. In the following sections, the detailed two-dimensional and three-dimensional models are presented. Then the high-explosive equation of state is presented. Next the procedure for sourcing the HE chemical energy into the hydrodynamics equations is discussed. Results and comparisons to analytical solutions are then presented and discussed. Coding and sample FLAG input files are shown in the Appendices.

II. LUND MODEL

The first step in the programmed burn calculation is to find the times that the high explosive lights at each mesh cell vertex. This step is taken at the beginning of the calculation. A number of algorithms are available to calculate these burn times. We are using the Lund model, which is described below.

The scalar burn time field, t, is given by the equation

$$|\nabla \mathbf{t}| = \frac{1}{\mathbf{D}} \quad , \tag{1}$$

where D is the high explosive detonation velocity. Values of D for a number of explosives are given in standard references (Dobratz and Crawford, 1985). In two dimensions there are three unknowns – the time, t, and the derivatives of t with respect to the two coordinate directions, in the finite-difference solution. In three dimensions there is an additional unknown – the derivative of t with respect to the third coordinate. Taylor expansions from one vertex to the other vertices provide the additional equation needed to find all the unknowns. The details of the solutions for two and three dimensions are given in the following sections.

III. TWO-DIMENSIONAL MODEL

Since it is easier to visualize the two-dimensional (2D) model than the three-dimensional model (3D), the 2D model will be described first. Initially all of the vertex times are set to a very large number. Once the triangles containing the detonator(s) are found and those triangle's vertices are lit, the following procedure is used to calculate the times at which the HE will light every other vertex in the mesh. For each triangle in the mesh

1) Since two of the vertices must have known times, skip the triangle during the current iteration if two of the vertices have the initial large time. Otherwise pick the vertex with the maximum time, which may be the initial large time, and calculate a new Lund trial time as described below. Triangles with known times are obtained from the cells with the detonators during the first iteration, as described above, or from calculations of previous triangles. If two vertices have identical maximum times, either one can be chosen to be recalculated.

2) The Lund trial time must satisfy local causality, which is described below. Local causality basically means that the calculated time must have come from the other two vertices of the triangle,

3) Calculate the direct times from the two known vertices,

4) The new time is the minimum of the old time, the Lund trial time, and the two direct times. The new time must be greater than the accepted times of the other two vertices since it is assumed that the detonation wave is coming from that side of the triangle,

5) Iterate until no vertex time is decreasing any longer.

A. Finding the Cell(s) Containing the Detonator(s)

The first step in determining the times for which the HE at each mesh vertex lights is to determine which cell contains the detonator, the position where the HE begins to burn, if there is only one detonator. If there are multiple detonators the procedure is done for each detonator. We need to find the triangle containing the user-specified detonations point. The surrounding vertices of the triangle are then lit by using the direct distance from the detonator to each vertex.

The area of a triangle can be found from the cross product of the vectors forming two sides of the triangle. Thus, if all of the areas of the triangles formed from the vectors to the vertices and the detonation point are positive, the detonation point is within the triangle. Consider a triangle with vertices at points 1, 2, and 3, and a detonation point at point p. The four vectors to the vertices of the triangle and to the detonator are

$$\bar{\mathbf{r}}_1 = \mathbf{x}_i\hat{\mathbf{i}} + \mathbf{y}_i\hat{\mathbf{j}} \ , \tag{2}$$

where i is 1, 2, or 3, and the detonator is at

$$\bar{\mathbf{p}} = \mathbf{p}_x\hat{\mathbf{i}} + \mathbf{p}_y\hat{\mathbf{j}} \ , \tag{3}$$

in two dimensions. The following cross products must be positive for the detonation point to be within the triangle

$$\begin{aligned} &\left(\mathbf{p} - \bar{\mathbf{r}}_1\right)\mathbf{X}\left(\bar{\mathbf{p}} - \bar{\mathbf{r}}_2\right) \ , \\ &\left(\mathbf{p} - \bar{\mathbf{r}}_2\right)\mathbf{X}\left(\bar{\mathbf{p}} - \bar{\mathbf{r}}_3\right) \ , \end{aligned} \tag{4}$$

and

$$\left(\bar{\mathbf{p}} - \bar{\mathbf{r}}_3\right)\ \mathbf{X}\ \left(\bar{\mathbf{p}} - \bar{\mathbf{r}}_1\right) \ . \tag{5}$$

Once it is determined that the detonator is in the triangle being examined, the lighting times for the three vertices are found from the distance from $\bar{\mathbf{p}}$ to each vertex. For example, for vertex 1, the time at which the HE will light is

$$\mathbf{t}_1 = \sqrt{(\mathbf{p_x} - \mathbf{x}_1)^2 + (\mathbf{p_y} - \mathbf{y}_1)^2} \; / \, \mathbf{D} + \mathbf{t}_{\text{det}} \quad , \tag{6}$$

where D is again the HE detonation velocity and t_{det} is the detonator lighting time, usually zero.

B. Two-Dimensional Lund Model

In 2D the HE burn time at one vertex of a triangle, t_0, can be calculated by the Lund model if the times at the other two vertices, t_1 and t_2, are known from calculations of other triangles, or from the detonator calculations. The calculated Lund time is a trial solution which must satisfy a number of criteria, discussed below, before it can be accepted as the vertex HE lighting time.

1. Burn Times. In 2D, three equations are needed in order to determine the burn time, t_0, at one vertex of the triangle, by the Lund algorithm. In addition to t_0, the unknowns in the finite-difference solution are the two derivatives of the time with respect to the coordinates x and y. From Eq. (1), we have

$$\left(\frac{\partial t}{\partial x}\right)^2 + \left(\frac{\partial t}{\partial y}\right)^2 = \frac{1}{\mathbf{D}^2} \quad , \tag{7}$$

The other two equations are found from taking a Taylor series about the vertex for which we are calculating the time

$$t_1 = \mathrm{t}_0 + \frac{\partial \mathrm{t}}{\partial \mathrm{x}}(\mathrm{x}_1 - \mathrm{x}_0) + \frac{\partial \mathrm{t}}{\partial \mathrm{y}}(\mathrm{y}_1 - \mathrm{y}_0) \tag{8}$$

and

$$t_2 = \mathrm{t}_0 + \frac{\partial \mathrm{t}}{\partial \mathrm{x}}(\mathrm{x}_2 - \mathrm{x}_0) + \frac{\partial \mathrm{t}}{\partial \mathrm{y}}(\mathrm{y}_2 - \mathrm{y}_0) \tag{9}$$

The derivatives are evaluated at the t_0 vertex. Equations (7)-(9) are solved for the three unknowns. It should be noted that Eq. (7) is a nonlinear equation. A quadratic equation results, this giving two solutions for t_0. The larger value is chosen.

2. Local Causality. In order for the Lund trial time to be an acceptable time, local causality must be satisfied. Local causality says that the detonation wave passing through node 0 must have come from nodes 1 and 2. Mathematically, the gradiant of t_0 must pass between nodes 1 and 2, and this will be true if the cross products of the gradient of the time at 0 with the vectors of the sides have opposite signs.

$$\nabla \mathbf{t} \; \mathbf{X} \; (\overline{\mathbf{r}_1} - \overline{\mathbf{r}_0}) \;\; ,$$

and

$$\nabla \mathbf{t} \; \mathbf{X} \; (\overline{\mathbf{r}_2} - \overline{\mathbf{r}_0}) \;\; ,$$

must have opposite signs.

IV. THREE-DIMENSIONAL MODEL

The three-dimensional model is similar to the two-dimensional model but the algebra needed to obtain the solutions is considerably more complicated. In 2D, Mathematica (Wolfram, 1996) was used to verify the algebra. In 3D Mathematica was used to obtain the solutions and to produce the Fortran coding. In this case four equations are needed to find the unknown HE lighting time and the three derivatives of the time with respect to the three coordinate directions. Prior to solving a given tetrahedron, the two-dimensional solutions for waves traveling within the faces must be obtained.

A. Finding the Cell(s) Containing the Detonator(s)

In 3D it is necessary to loop over each detonator, find the tetrahedron containing the detonator, and then light the four vertices of the tetrahedron. The lighting of the vertices surrounding the detonator is again done by direct lighting.

In 2D we determined if a detonation point was within a triangle by looking at the signs of vector areas obtained by taking the cross product of vectors to the vertices and the detonation point. The corresponding algorithm in 3D involves volumes, obtained by dotting cross products of the position vectors into the vector area of a face of the tetrahedron. Figure 1 shows a tetrahedron, containing the nomenclature.

The vector areas of the four faces of the tetrahedron are

$$\overline{A_1} = (\overline{r_3} - \overline{r_2}) \; X \; (\overline{r_4} - \overline{r_2}) \;\; ,$$

$$\overline{A_2} = (\overline{r_4} - \overline{r_1}) \; X \; (\overline{r_3} - \overline{r_1}) \;\; ,$$

$$\overline{A_3} = (\overline{r_4} - \overline{r_2}) \; X \; (\overline{r_1} - \overline{r_2}) \;\; ,$$

and

$$\overline{A_4} = (\overline{r_1} - \overline{r_2}) \; X \; (\overline{r_3} - \overline{r_2}) \;\; ,$$

where the areas represent the outward normal of the tetraderal face opposite the vertex designated. For example, A_1 is the area of the face opposite vertex 1.

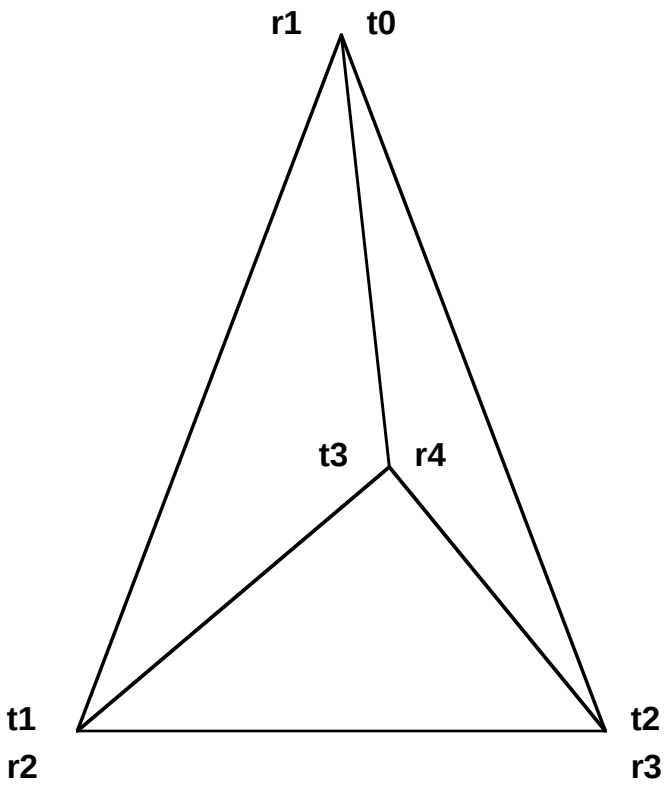

Fig. 1. Tetrahedral nomenclature.

The detonator is within the tetrahedron if the following four volumes are all positive. The vector position of the detonator is $\bar{\mathbf{p}}$.

$$(\bar{\mathbf{r}}_i - \bar{\mathbf{p}}) \bullet \overline{\mathbf{A}_i} \ ,$$

where i takes on the values 1, 2, 3, and 4.

B. Three-Dimensional Lund Model

1. Burn Times. The three-dimensional solution for the time, t_0, is similar to the two-dimensional method described above, except that there are now four equations for four unknowns, and the algebra is considerably more involved. In addition waves traveling within each face of the tetrahedron must be considered.

Since some tetraheral faces are not parallel to coordinate planes, it is useful to transform the faces into a two-dimensional coordinate system. Figure 2 shows a tetrahedral face with a two-dimensional coordinate system (xp, yp) attached to the vertex were we are calculating a new high explosive burn time.

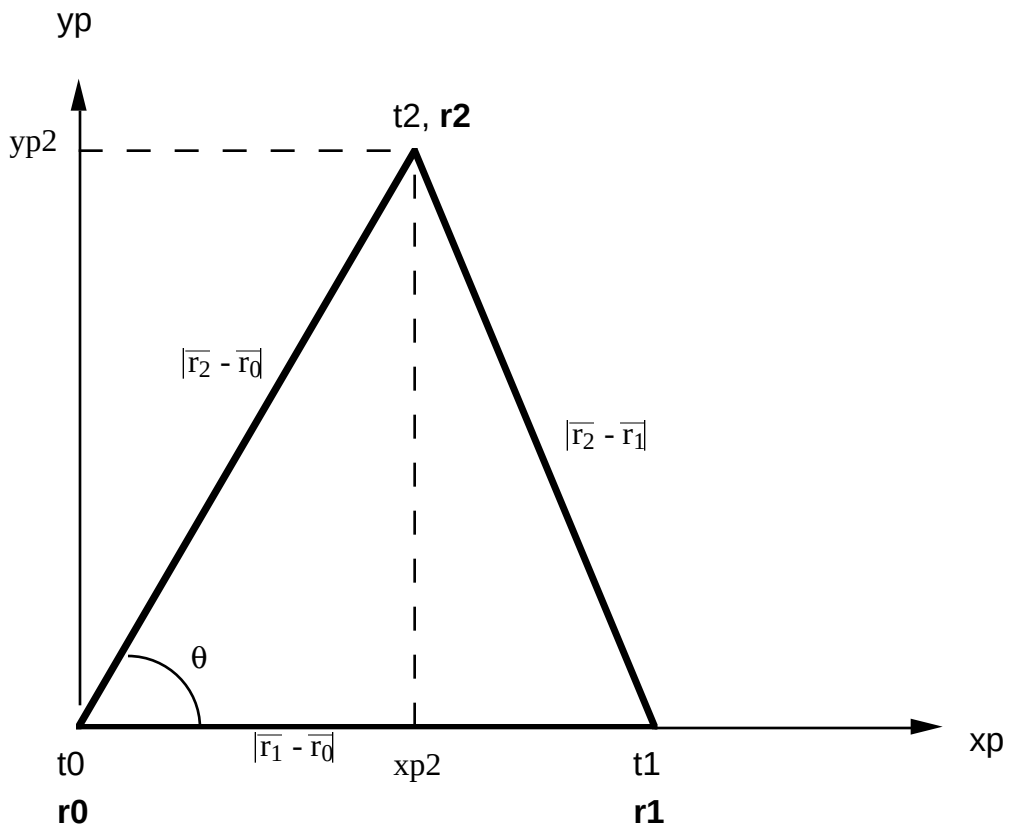

Fig. 2. Transformed coordinate system for tetrahedral face.

The lengths of the sides of the triangle are shown. The vectors r are given in terms of the original three-dimensional coordinate system. The angle theta is given by

$$\cos\theta = \frac{\left(\overline{\mathbf{r}_2} - \overline{\mathbf{r}_0}\right) \bullet \left(\overline{\mathbf{r}_1} - \overline{\mathbf{r}_0}\right)}{\left|\overline{\mathbf{r}_2} - \overline{\mathbf{r}_0}\right| \left|\overline{\mathbf{r}_1} - \overline{\mathbf{r}_0}\right|} \quad .$$

The coordinates of the triangle in the new coordinate system are

$$\text{xp0} = 0.0,\ \text{yp0} = 0.0,\ \text{xp1} = \overline{\mathbf{r}_1} - \overline{\mathbf{r}_0}, \text{yp1} = 0.0,\ \text{xp2} = (\overline{\mathbf{r}_2} - \overline{\mathbf{r}_0})\cos\theta \ ,$$

and

$$\text{yp2} = (\overline{\mathbf{r}_2} - \overline{\mathbf{r}_0})\sin\theta \ .$$

With these tranformed coordinates, the previously described two-dimensional Lund solution is used to solve each face of the tetrahedron for the HE burn time for the vertex having the maximum time, from previous calculations. The accepted solution for any

vertex is the minimum acceptable time from the face and tetrahedral solutions. This minimum time must be greater than the times for the other three vertices.

2. Local Causality. In three dimensions, local causality requires that the vector given by the gradient of the burn time must pass through the opposite face of the terahedron. This requirement means that

$$\nabla t_0 = \frac{\partial t}{\partial x}\hat{i} + \frac{\partial t}{\partial y}\hat{j} + \frac{\partial t}{\partial z}\hat{k}$$

must intersect the plane given by Ax + By + Cz + D = 0. A, B, and C can be found from the cross product of the vectors of the sides of the tetrahedral face (Wexler, 1962):

$$(\bar{\mathbf{r}}_3 - \bar{\mathbf{r}}_1)\ X\ (\bar{\mathbf{r}}_2 - \bar{r}_1) = A\hat{i} + B\hat{j} + C\hat{k}\ ,$$

and $D = -Ax_2 - By_2 - Cz_2$, where x_2, y_2, and z_2 are the coordinates at point 2 of the tetrahedron.

Three equations are needed to find the intercept point of the plane and the gradient of the time. The above equation of the plane is one of the equations needed, and the two equations for the gradient line are the additional equations needed:

$$\frac{x - x_1}{\frac{\partial t}{\partial x}} = \frac{y - y_1}{\frac{\partial t}{\partial y}} = \frac{z - z_1}{\frac{\partial t}{\partial z}}\ ,$$

where x_1, y_1, and z_1 are the coordinates of the vertex where the time t_0 has been calculated, and the derivatives are also evaluated at that vertex.

Simultaneous solution of the above three equations gives the coordinates, xi, yi, and zi, where the gradient intercepts the plane. It is next necessary to determine if the intercept point is within or outside the face of the tetrahedron. This determination is done in a similar manner to the method used to determine if the detonator is within the tetrahedron.

V. EQUATION OF STATE

The Jones-Wilkins-Lee (JWL) equation of state is used for the detonation products (Lee et al., 1968). In the current version of FLAG, the high explosive pressure is zero prior to HE detonation, given by the following equation after the HE is fully burned, and linearly increased during the time the HE is burning.

$$P = A\left(1 - \frac{\omega}{R_1 V}\right)e^{-R_1 V} + B\left(1 - \frac{\omega}{R_2 V}\right)e^{-R_2 V} + \frac{\omega E}{V}$$

The material constants R_1, R_2, A, B, and ω are given, for a number of explosives, in the report by Dobratz and Crawford. V is the relative specific volume, E is the HE chemical energy, and P is the pressure of the HE detonation products.

VI. ENERGY SOURCES

The high explosive energy must be coupled to the hydrodynamic equations by including the HE energy in the force and work terms. This energy is linearly added to the hydrodynamic equations from the minimum to the maximum HE burn times calculated. The fraction of energy added each time step is

$$\frac{\Delta t}{t_{\max} - t_{\min}} E_0 \ ,$$

where t_{min} and t_{max} are minimum and maximum HE cell lighting times, and E_0 is the HE chemical energy.

VII. RESULTS

Two- and three-dimensional test problems were run in order to verify that the Lund model was implemented into the code correctly, and to evaluate the model. Simple geometries were run first so that the HE lighting times could be compared to exact solutions. For a square of HE in 2D or a cube of HE in 3D, lit at one corner at time zero, the exact solution is the distance from the detonator point to a vertex divided by the detonation velocity. Sample input files are given in Appendices C and D.

A. Two-Dimensional Results

Contours of constant HE lighting times, obtained by using the Lund model, for a 1-cm by 1- cm square of HE, lit at one corner, and having 10 by 10 cells, are shown in Fig. 3a. The corresponding exact solution is shown in Fig. 3b, and the percent error is shown in Fig. 4. The maximum error is about 3.95 percent. The results show that the Lund model is working correctly in the FLAG code in two dimensions. The input file for the Lund model for a 2-cell by 2-cell (3 by 3 lines) geometry is given in Appendix C.

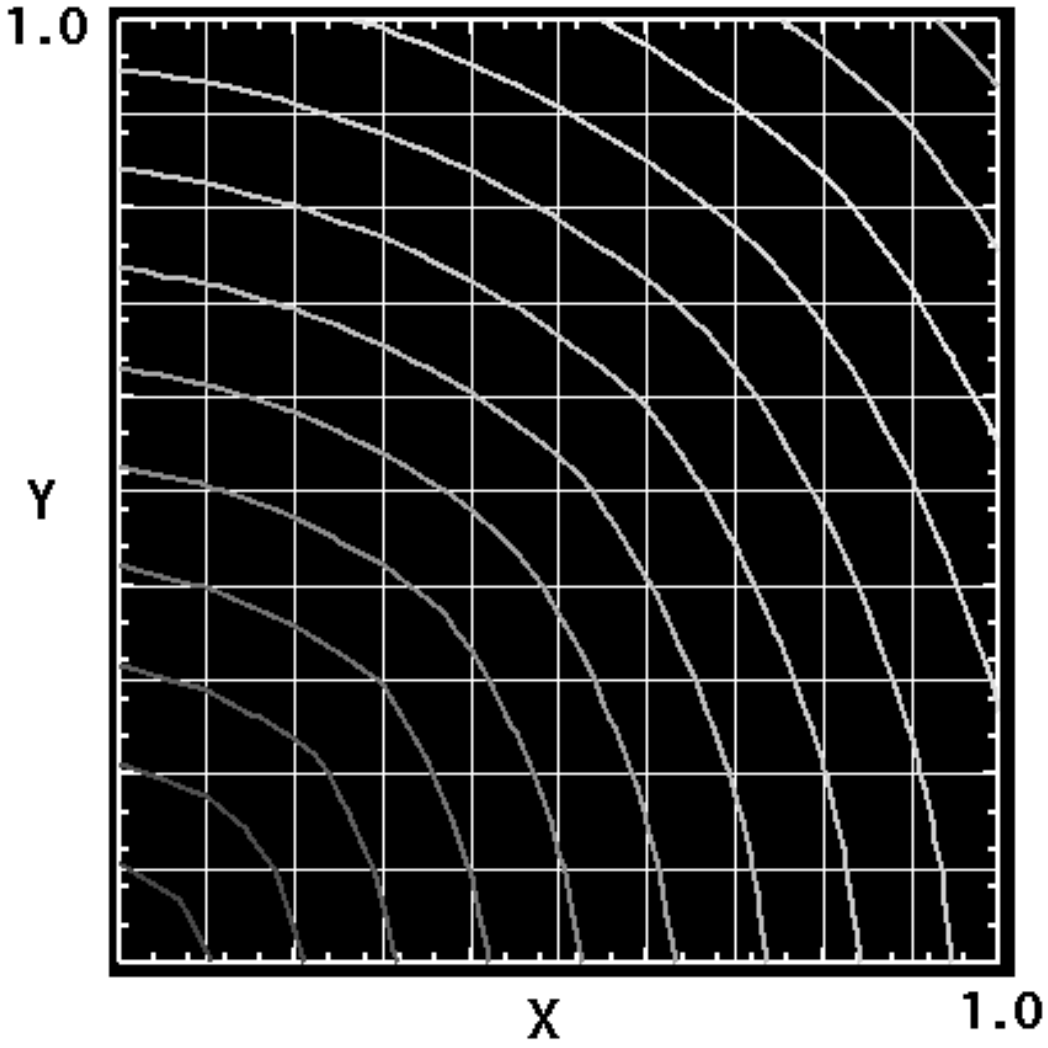

Fig. 3a. HE burn times, phet, for a square lit at one corner,using the Lund model.

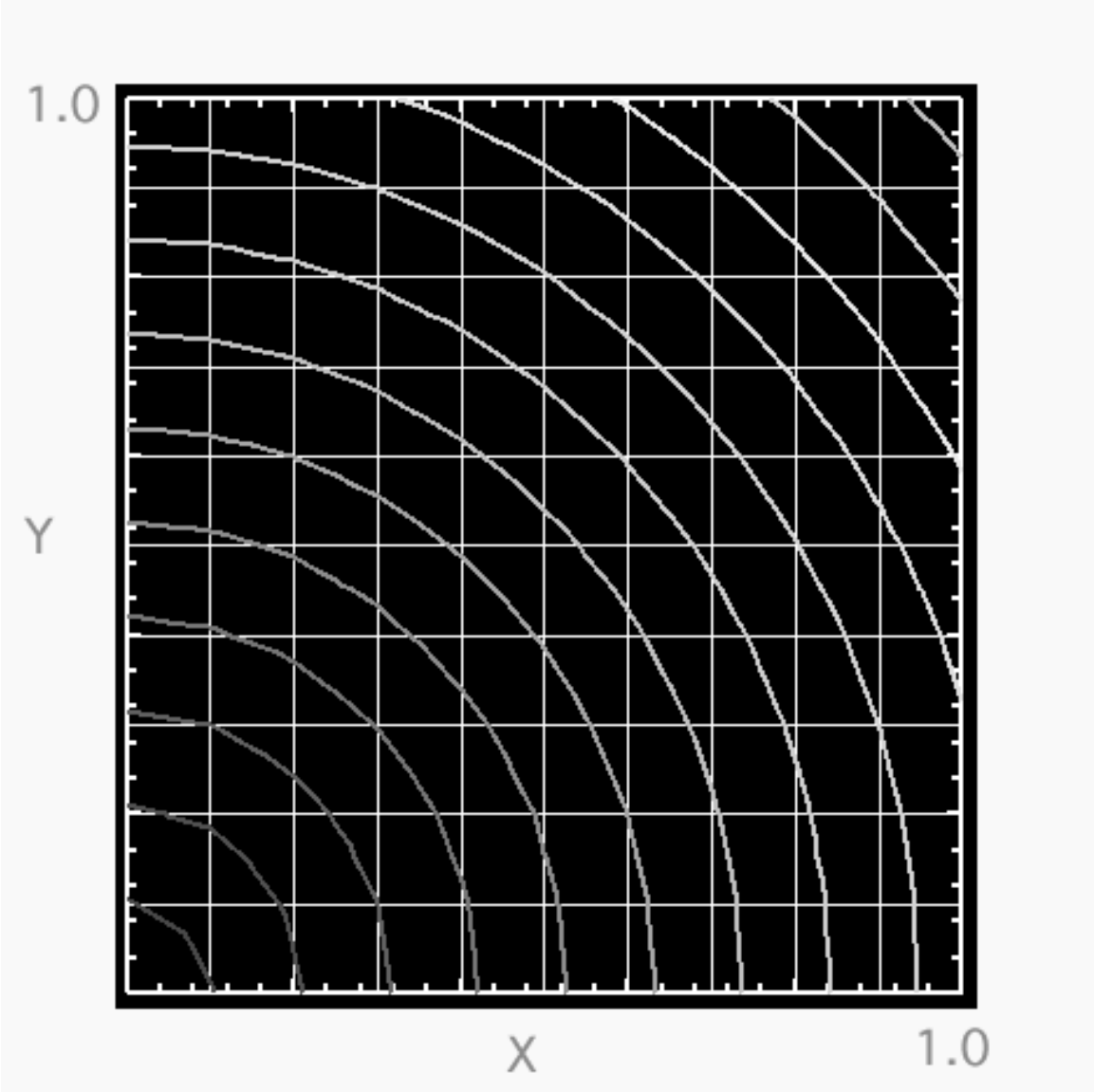

Fig. 3b. Exact HE burn times, phet.

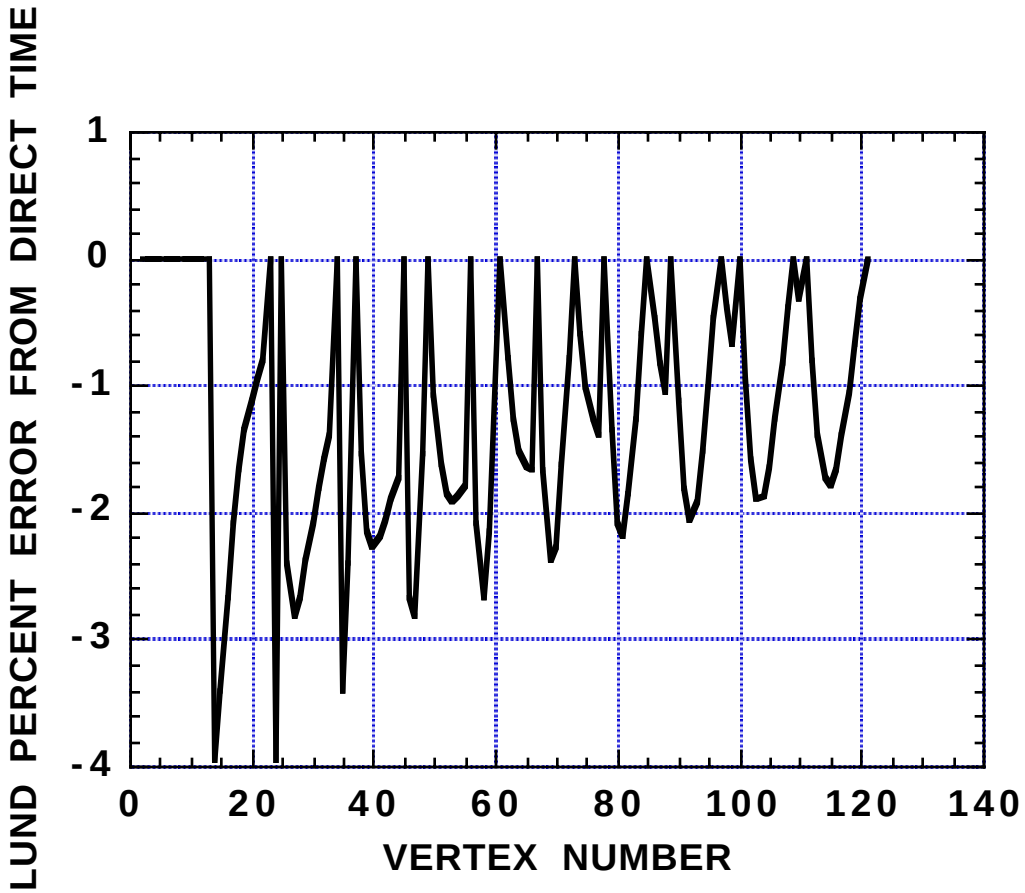

Fig. 4. Percent error between the exact (direct) solution and the Lund model for the 2D square.

To test the coupling of the HE model with the hydrodynamics, a cylinder of PBX-9501 explosive, lit at the origin was calculated. The resulting pressure wave at 3 microseconds is shown in Fig. 5. The mesh consisted of 10 by 50 cells and was run on a workstation. The Chapman-Jouguet (CJ) pressure for PBX-9501 is 370 kilobars. The maximum calculated pressure was about 278 kilobars. The calculated pressure would approach the measured value if a finer zoned calculation were run. We did not perform this calculation because of disc limitations on the workstation used for the calculations.

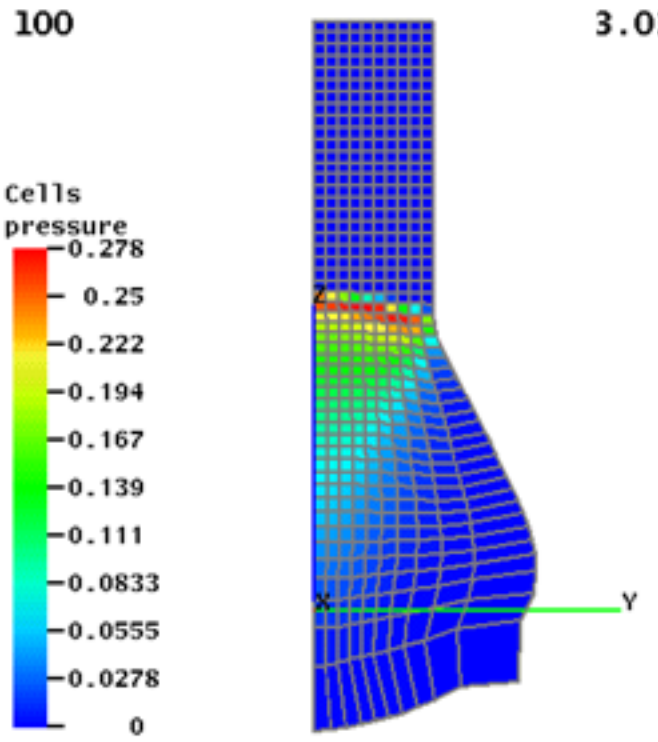

Fig. 5. Calculated 2D pressure contours for a cylinder of PBX-9501 at 3 microseconds. The Lund model was used to calculate the HE lighting times.

B. Three-Dimensional Results

The two test problems discussed above were converted to three-dimensional problems. The results for the error in the cube HE lighting times are shown in Fig. 6. The maximum error was about 4.84 percent. Results for the PBX-9501 cylinder are shown in Figs. 7 and 8. One quarter of the cylinder was used in the calculation, and the results were reflected about the axes using the GMV graphics package (Ortega, 1995).

The results verify the 3D coding.

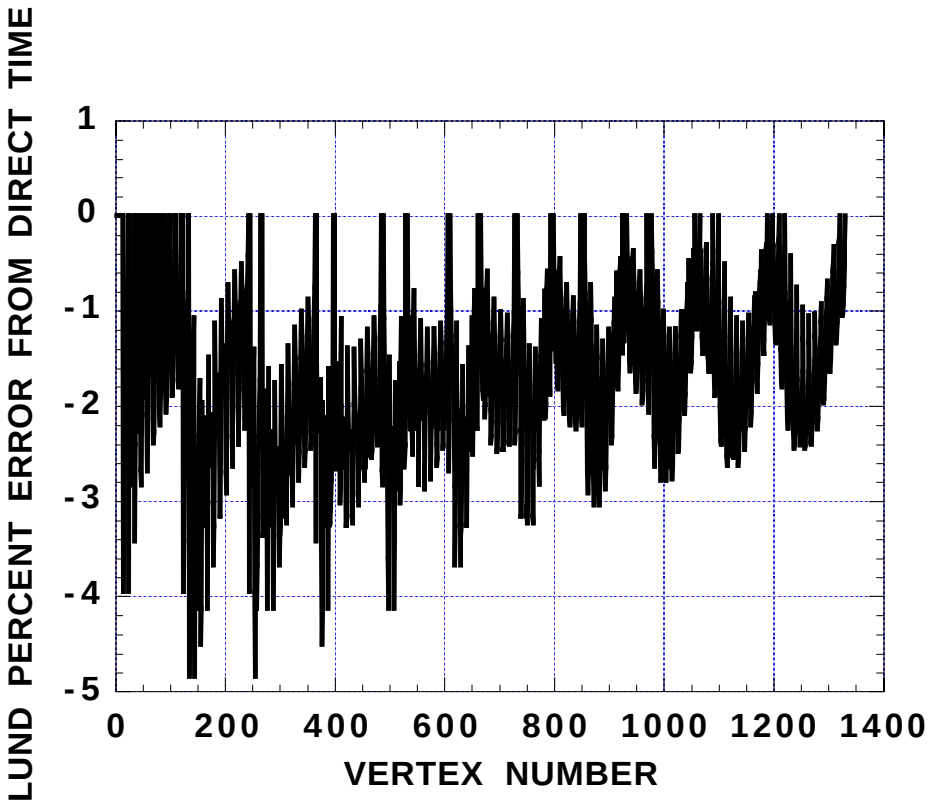

Fig. 6. Percent error between the exact (direct) solution and the Lund model for the 3D cube.

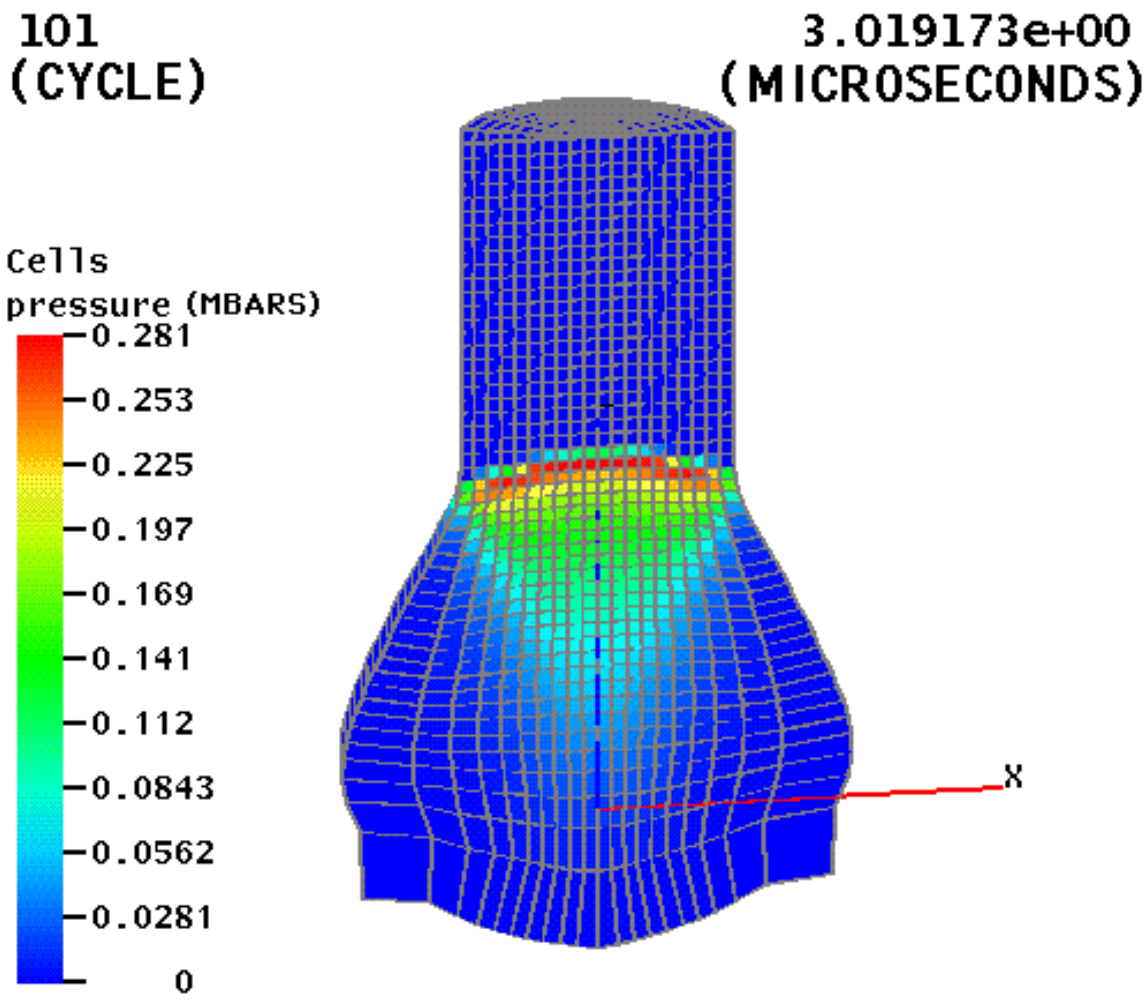

Fig. 7. Pressure contours at 3 microseconds for a PBX-9501 cylinder, lit at the origin. Half the cylinder is shown. The HE burn times were obtained by using the Lund model.

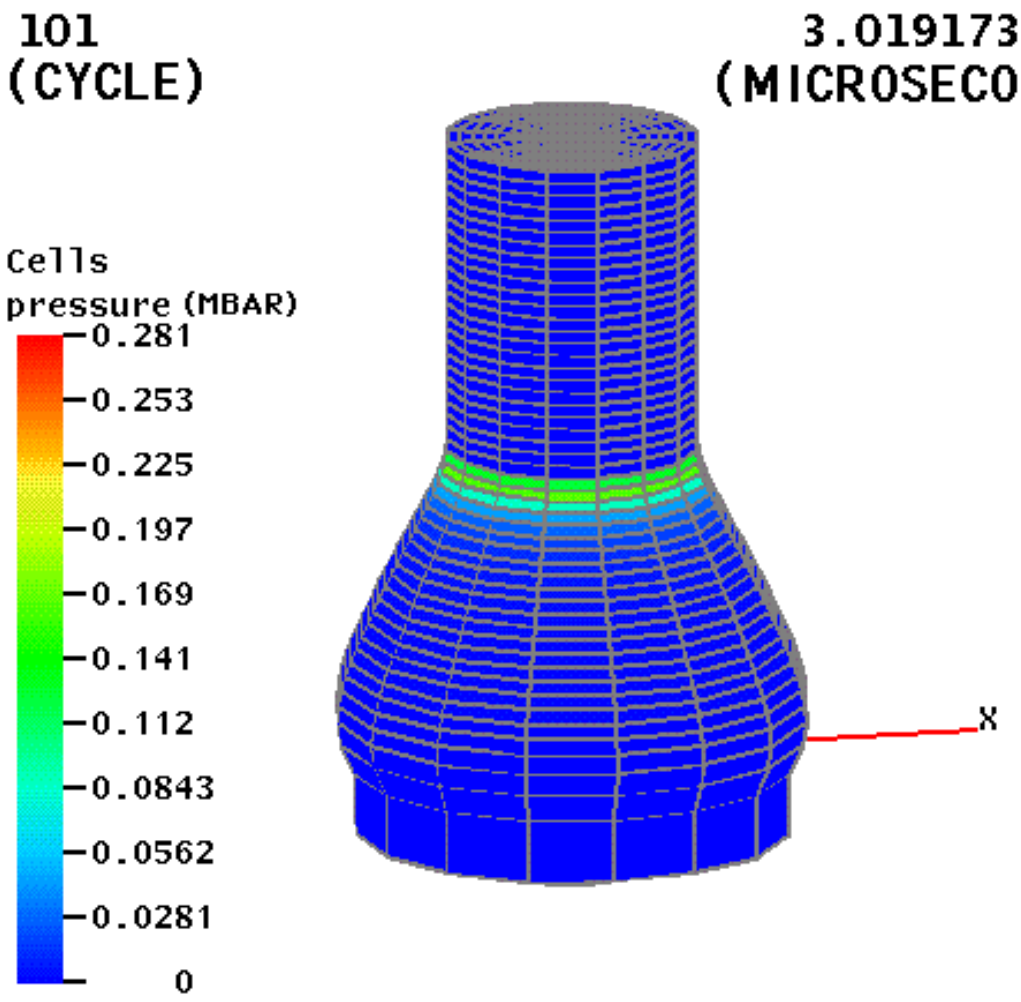

Fig. 8. Pressure contours at 3 microseconds for a PBX-9501 cylinder, lit at the origin. The entire cylinder is shown. The Lund model was used.

ACKNOWLEDGMENTS

We would like to thank Robert Hotchkiss, Computational Science Methods Group, for his help with the vector algebra and analytical geometry methods. We would also like to thank Frank Ortega, Computational Science Methods Group, for help in using his General Mesh Viewer (GMV) graphics package.

REFERENCES

1. Donald E. Burton, "Connectivity Structures and Differencing Techniques for Staggered-Grid Free-Lagrange Hydrodynamics," Lawrence Livermore National Laboratory Report UCRL–JC–110555 (1992).
2. Donald E. Burton, Consistent Finite-Volume Discretization of Hydrodynamic Conservation Laws for Unstructured Grids, Lawrence Livermore National Laboratory Report UCRL–JC–118788 (1994).
3. B. M. Dobratz and P. C. Crawford, "LLNL Explosives Handbook—Properties of Chemical Explosives and Explosive Simulants," Lawrence Livermore National Laboratory Report UCRL–52997 (January 31, 1985).
4. E. L. Lee, H. C. Hornig, and J. W. Kury, "Adiabatic Expansion of High Explosive Detonation Products," Lawrence Livermore National Laboratory Report UCRL–50422 (May 2, 1968).
5. Carl Lund, Private Communication (1986).
6. Frank Ortega, "GMV—General Mesh Viewer User's Manual," Los Alamos National Laboratory report LAUR–95–2986 (1995).
7. Stephen Wolfram, *The Mathematica Book*, Third Edition, Cambridge University Press, NY (1996).
8. Charles Wexler, *Analytic Geometry A Vector Approach*, Addison-Wesley Pub. Co., Reading, MA (1962).

APPENDIX A

TWO-DIMENSIONAL FORTRAN CODING

In order to aid other computational physicsts who wish to use the Lund high explosive burn model, the coding used in FLAG is presented in this Appendix and the following Appendix. The two-dimensional coding is presented in this Appendix. FLAG is written in object-oriented Fortran. A database manager controls the classes, which are designated by dd in the following coding. The "access" command in the coding brings in the variables indicated. The written coding is preprocessed to insert the pointers and other Fortran 77 constructs.

I. LOCATION OF 2D DETONATORS

```
c-------------------------------------------------------------

      subroutine HEDet2D(dd)
c
c     Find the cells containing detonators.
c     Calculate the lighting times around the detonators for 2D
c
      access /dd/ kk4ll, kkdll,
     1            kk3ll,kksll, kksl, kkdl,
     1            dxt(kk4ll,kkdll),
     1            kkpll,px(kk3ll,kkpll),
     1            kkzll,zx(kk3ll,kkzll),
     1            phet(kkpll),pheto(kkpll),
     1            zhet(kkzll),zheto(kkzll),
     1            kstyp(kksll),
     1            kksp1(kksll),kksp2(kksll),kksz(kksll),
     1            detvel(kksll),
     1            idebug,zero

      integer s,p1,p2,z,d,inside

      real*8 xdet,ydet,tdet,
     1       x1,x2,x3,y1,y2,y3,
     1      pr1cpr2,pr2cpr3,pr3cpr1

      data inside/0/
c
c     loop over all detonators
c
      do d=1,kkdl

        xdet = dxt(1,d)
        ydet = dxt(2,d)
        tdet = dxt(3,d)
c
```

```
c     loop over sides
c
        do s=1,kksl

          if(kstyp(s).eq.1) then

            p1 = kksp1(s)
            p2 = kksp2(s)
            z  = kksz(s)

            x1 = px(1,p1)
            x2 = px(1,p2)
            x3 = zx(1,z)
            y1 = px(2,p1)
            y2 = px(2,p2)
            y3 = zx(2,z)

c
c     Cross the vector differences of the detonation position
c     and the point positions in all combinations. If the
c     three results are all positive, the detonation point is
c     within the side.
c
            pr1cpr2 = (xdet-x1)*(ydet-y2)-(xdet-x2)*(ydet-y1)
            pr2cpr3 = (xdet-x2)*(ydet-y3)-(xdet-x3)*(ydet-y2)
            pr3cpr1 = (xdet-x3)*(ydet-y1)-(xdet-x1)*(ydet-y3)

            if(idebug .eq. 2) then
              write(*,*)
               write(*,*) '-------------------------------'
              write(*,*)
               write(*,*) 'HEDet2D: Side = ',s
                 write(*,*) 'p1=',p1,  ' p2 = ',p2,' z = ',z
                write(*,*) 'x1 = ',x1,' y1 = ',y1
                write(*,*) 'x2 = ',x2,' y2 = ',y2
                write(*,*) 'x3 = ',x3,' y3 = ',y3
               write(*,*) 'xdet = ',xdet,' ydet = ',ydet
               write(*,*) 'pr1cpr2 = ',pr1cpr2,' pr2cpr3 = ',
     1                    pr2cpr3,' pr3cpr1 = ',pr3cpr1
              write(*,*)
               write(*,*) '-------------------------------'
              write(*,*)
            endif

           if(pr1cpr2 .ge. zero .and.
     1        pr2cpr3 .ge. zero .and.
     1        pr3cpr1 .ge. zero       ) then

              phet(p1) = sqrt( (xdet-x1)**2 + (ydet-y1)**2 ) /
     1                   detvel(s) + tdet
              phet(p2) = sqrt( (xdet-x2)**2 + (ydet-y2)**2 ) /
     1                   detvel(s) + tdet
              zhet(z)  = sqrt( (xdet-x3)**2 + (ydet-y3)**2 ) /
     1                   detvel(s) + tdet
```

```
c
c     Another detonation point could be closer to the point than
c     the current detonation point so take the minimum time.
c
                phet(p1)  = amin1(phet(p1),pheto(p1))
                pheto(p1) = phet(p1)
                phet(p2)  = amin1(phet(p2),pheto(p2))
                pheto(p2) = phet(p2)
                zhet(z)   = amin1(zhet(z),zheto(z))
                zheto(z)  = zhet(z)

                inside = 1

                if(idebug .eq. 1) then
                   write(*,*)
                    write(*,*) '*********************************'
                   write(*,*)
                    write(*,*) 'HEDet2D:'
                      write(*,*) 'p1=',p1,' phet(p1)=',phet(p1),'
     1                          p2=',p2,
     1                          ' phet(p2) = ',phet(p2),' z=',z,
     1                          ' zhet=',zhet(z), ' s=',s
                   write(*,*)
                endif

              endif

            endif

          enddo

        enddo

       if(inside .ne. 1 )
      1   call Fatal('HEDet2D: Detonators must be inside mesh')

       return
       end
```

II. LUND 2D CALCULATION

```
c-------------------------------------------------------------

      subroutine HELund2D(dd)
c
c     Calculate the Lund HE burn times for 2D
c

      access /dd/ kk4ll,fnul,
     1            kk3ll,kksll, kksl,
     1            kkpll,px(kk3ll,kkpll),
     1            kkzll,zx(kk3ll,kkzll),
     1            phet(kkpll),pheto(kkpll),
```

```
     1           zhet(kkzll),zheto(kkzll),
     1           kstyp(kksll),
     1           kksp1(kksll),kksp2(kksll),kksz(kksll),
     1           detvel(kksll),
     1           mheiter,zero,
     1           idebug

      integer s,p1,p2,z,d,
     1        heiter,
     1        success,bad,
     1        i1,i2,i3,flag

      real*8 x1,x2,x3,y1,y2,y3,
     1       tmax,t21,t31,t1,t2,t0tmp,
     1        x10,y10,x20,y20,x12,y12,d1,disc,
     1        ax,ay,bx,by,a,b,c,
     1       gx,gy,gc10,gc20,
     1      ihe

c
c    Iterate until the HE burn times are no longer decreasing
c
     do heiter=1,mheiter

c
c    loop over "sides"
c
        do s=1,kksl

           if(kstyp(s).eq.1 .and. detvel(s) .ne. fnul) then

              p1 = kksp1(s)
              p2 = kksp2(s)
              z  = kksz(s)

              x1 = px(1,p1)
              x2 = px(1,p2)
              x3 = zx(1,z)
              y1 = px(2,p1)
              y2 = px(2,p2)
              y3 = zx(2,z)
c
c    Two of the three times must be known to use the Lund method
c
              ihe = 0
              if(phet(p1) .eq. fnul) ihe = ihe + 1
              if(phet(p2) .eq. fnul) ihe = ihe + 1
              if(zhet(z)  .eq. fnul) ihe = ihe + 1
              if(ihe .lt. 2) then
c
c    Find the maximum HE burn time and recalculate that point
c    from the other two points of the triangle
c
                 tmax = amax1( phet(p1), phet(p2), zhet(z) )
```

```
            if(phet(p1).ge.phet(p2).and.phet(p1).ge.zhet(z)) then
c
c     Time at p1 is being recalculated
c
                i1 = p1
                i2 = p2
                i3 = z
                flag = 0
              elseif(phet(p2).ge.phet(p1).and.phet(p2).gt.
     1              zhet(z)) then
c
c     Time at p2 is being recalculated
c
                i1 = p2
                i2 = p1
                i3 = z
                flag = 0
              elseif(zhet(z).gt.phet(p1).and.zhet(z).ge.
     1              phet(p2)) then
c
c     Time at z is being recalculated
c
                i1 = z
                i2 = p1
                i3 = p2
                flag = 1

              else
                cycle
              endif

              if(flag .eq. 0) then
c
c     Either point p1 or point p2 is being recalculated.
c
                t1  = phet(i2)
                t2  = zhet(i3)
                x10 = px(1,i2)-px(1,i1)
                y10 = px(2,i2)-px(2,i1)
                x20 = zx(1,i3)-px(1,i1)
                y20 = zx(2,i3)-px(2,i1)
                t21 = sqrt(x10**2+y10**2)/detvel(s)+phet(i2)
                t31 = sqrt(x20**2+y20**2)/detvel(s)+zhet(i3)
                x12 = px(1,i2)-zx(1,i3)
                y12 = px(2,i2)-zx(2,i3)
              else
c
c     Point z is being recalculated.
c
                t1  = phet(i2)
                t2  = phet(i3)
                x10 = px(1,i2)-zx(1,i1)
                y10 = px(2,i2)-zx(2,i1)
```

```
      x20 = px(1,i3)-zx(1,i1)
      y20 = px(2,i3)-zx(2,i1)
     t21 = sqrt(x10**2+y10**2)/detvel(s)+phet(i2)
     t31 = sqrt(x20**2+y20**2)/detvel(s)+phet(i3)
      x12 = px(1,i2)-px(1,i3)
      y12 = px(2,i2)-px(2,i3)
    endif

    d1 = x20*y10-x10*y20
    ax= - y12/d1
    ay=   x12/d1
    bx=  (t2*y10 - t1*y20)/d1
    by= -(t2*x10 - t1*x20)/d1
    a = ax**2+ay**2
    b = 2*(ax*bx+ay*by)
    c = (bx**2+by**2-1/detvel(s)**2)
    disc=b**2-4.*a*c

    if (disc.ge.0) then


      t0tmp= (-b + sqrt(disc))/(2.*a)

      gx = ax*t0tmp+bx
      gy = ay*t0tmp+by

      gc10 = gx*y10-gy*x10
      gc20 = gx*y20-gy*x20

      if(gc10*gc20 .lt. zero) then

        if(flag .eq. 0) then
          if(t0tmp.gt.phet(i2).and.t0tmp.gt.
     1          zhet(i3)) then
            phet(i1) = amin1(t0tmp,t21,t31,
     1              phet(i1),pheto(i1))
          else
            phet(i1) = amin1(t21,t31,phet(i1),
     1              pheto(i1))
          endif
        else
          if(t0tmp.gt.phet(i2).and.t0tmp.gt.
     1          phet(i3)) then
            zhet(i1) = amin1(t0tmp,t21,t31,zhet(i1),
     1              zheto(i1))
          else
            zhet(i1) = amin1(t21,t31,zhet(i1),
     1              zheto(i1))
          endif
        endif
      else


        if(flag .eq. 0) then
```

```
                phet(i1) = amin1(t21,t31,phet(i1),pheto(i1))

              else
                zhet(i1) = amin1(t21,t31,zhet(i1),zheto(i1))
              endif

            endif

          else
              if(flag .eq. 0) then
                phet(i1) = amin1(t21,t31,phet(i1),pheto(i1))

              else
                zhet(i1) = amin1(t21,t31,zhet(i1),zheto(i1))
              endif

          endif

            if( idebug .eq. 1 ) then
              write(*,*)
               write(*,*) '-----------------------------------'
              write(*,*)
              write(*,*) 'HELund2D: HE ITERATION NUMBER',heiter
                 write(*,*) 'p1=',p1,' p2=',p2,' z=',z
                 write(*,*) 's=',s,' phet(p1)=',phet(p1),'
     1                    phet(p2)=',
     1                    phet(p2),'  zhet = ',zhet(z)
                  write(*,*) 'x1=',x1,' y1=',y1,' x2 = ',x2,'
     1                    y2 = ',y2,
     1                      ' x3 = ',x3,' y3 = ',y3
               write(*,*) 'tmax = ',tmax
                write(*,*) 'pheto(p1)=',pheto(p1),' pheto(p2)=',
     1                   pheto(p2),
     1                    ' zheto(z) =',zheto(z)
                write(*,*) 't0tmp = ',t0tmp,' t21 = ',t21,'
     1                    t31 = ',t31
                 write(*,*) 'i1=',i1,' i2=',i2,' i3=',i3,
     1                    ' flag=',flag
               write(*,*) 'disc = ',disc
                 write(*,*) 'a = ',a,' b = ',b,' c = ',c
                 write(*,*) 'Gx = ',gx,' Gy = ',gy,' gc10 = ',gc10,
     1                    ' gc20 = ',gc20
                write(*,*) 'x10 = ',x10,' y10 =',y10
                write(*,*) 'x20 = ',x20,' y20 = ',y20
                 write(*,*) 'x12 = ',x12,' y12 = ',y12,' d1=',d1
             endif

           endif

        endif

      enddo
```

```
      bad = 0

      do s=1,kksl

        if(kstyp(s).eq.1) then

          p1 = kksp1(s)
          if( phet(p1) .lt. fnul) then
             bad = 1
             exit
          endif

        endif

      enddo

      if(bad .eq. 0) call Fatal('HELund2D: BAD TIME ITERATION')

      success = 0


      do s=1,kksl

        if(kstyp(s).eq.1 .and. detvel(s) .ne. fnul) then

          p1 = kksp1(s)
          p2 = kksp2(s)
          z  = kksz(s)

          if( phet(p1) .lt. pheto(p1) ) then
             success = success + 1
             pheto(p1) = phet(p1)
          endif
          if( phet(p2) .lt. pheto(p2) ) then
             success = success + 1
             pheto(p2) = phet(p2)
          endif
          if( zhet(z) .lt. zheto(z) ) then
             success = success + 1
             zheto(z) = zhet(z)
          endif

c          write(*,*) 'HELUND2D: HE ITERATION NUMBER',heiter
c           write(*,*) 's = ',s,' phet(p1) = ',phet(p1),' phet(p2) = ',
c    1                 phet(p2),'  zhet = ',zhet(z)

        endif

      enddo

      if(success .eq. 0 ) then
        write(*,*)
        write(*,*) '--------- HE LUND 2D BURN ----------'
        write(*,*)
```

```
      write(*,*) 'NUMBER OF ITERATIONS = ',heiter
      write(*,*) 'MAXIMUM ITERATIONS ALLOWED = ',mheiter
      write(*,*)
       write(*,*) '----------------------------------------'
      write(*,*)
      exit
     endif

    enddo

    if(success .ne. 0 )
   1   call Fatal('HELund2D: HE iteration did not converge ')

    return
    end
```

APPENDIX B

THREE-DIMENSIONAL CODING

In order to aid other computational physicists who wish to use the Lund high explosive burn model, the 3D coding used in FLAG is presented in this Appendix. FLAG is written in object-oriented Fortran. A database manager controls the classes, which are designated by dd in the following coding. The "access" command in the coding brings in the variables indicated. The written coding is preprocessed to insert the pointers and other Fortran 77 constructs.

I. LOCATION OF 3D DETONATORS

```
c-------------------------------------------------------------

      subroutine HEDet3D(dd)
c
c     Find the cells containing detonators.
c     Calculate the lighting times around the detonators for 3D
c
      access /dd/ kk4ll, kkdll,
     1            kk3ll,kksll, kksl, kkdl,
     1            dxt(kk4ll,kkdll),
     1            kkpll,px(kk3ll,kkpll),
     1            kkzll,zx(kk3ll,kkzll),
     1            kkfll,fx(kk3ll,kkfll),
     1            phet(kkpll),pheto(kkpll),
     1            zhet(kkzll),zheto(kkzll),
     1            fhet(kkfll),fheto(kkfll),
     1            kstyp(kksll),
     1            kksp1(kksll),kksp2(kksll),kksz(kksll),kksf(kksll),
     1            detvel(kksll),
     1            idebug,zero

      integer s,p1,p2,z,f,d,inside,totin

      real*8 xdet,ydet,zdet,tdet,
     1       x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,z3,z4,
     1      r1pa1,r2pa2,r3pa3,r4pa4
c
c     totin is the runnig total of the number of detonators
c     inside the mesh
c
      totin=0

c
c     loop over all detonators
c
      do d=1,kkdl
```

```
        xdet = dxt(1,d)
        ydet = dxt(2,d)
        zdet = dxt(3,d)
        tdet = dxt(4,d)
c
c     loop over sides
c
        do s=1,kksl

          if(kstyp(s).eq.1) then

            p1 = kksp1(s)
            p2 = kksp2(s)
            z  = kksz(s)
            f  = kksf(s)

            x1 = px(1,p1)
            x2 = px(1,p2)
            x3 = zx(1,z)
            x4 = fx(1,f)

            y1 = px(2,p1)
            y2 = px(2,p2)
            y3 = zx(2,z)
            y4 = fx(2,f)

            z1 = px(3,p1)
            z2 = px(3,p2)
            z3 = zx(3,z)
            z4 = fx(3,f)

       r1pa1 =
     -  ((y2-ydet)*(-(x3*z2)+x4*z2+x2*z3-x4*z3-x2*z4+x3*z4)+
     -   (x2-xdet)*(y3*z2-y4*z2-y2*z3+y4*z3+y2*z4-y3*z4)+
     -   (x3*y2 - x4*y2 - x2*y3 + x4*y3 + x2*y4 - x3*y4)*(z2 - zdet))

       r2pa2 =
     -  ((y1-ydet)*(x3*z1-x4*z1-x1*z3+x4*z3+x1*z4-x3*z4)+
     -   (x1-xdet)*(-(y3*z1)+y4*z1+y1*z3-y4*z3-y1*z4+y3*z4)+
     -   (-(x3*y1)+x4*y1+x1*y3-x4*y3-x1*y4+x3*y4)*(z1-zdet))

       r3pa3 =
     -  ((y1-ydet)*(-(x2*z1)+x4*z1+x1*z2-x4*z2-x1*z4+x2*z4)+
     -   (x1-xdet)*(y2*z1-y4*z1-y1*z2+y4*z2+y1*z4-y2*z4)+
     -   (x2*y1-x4*y1-x1*y2+x4*y2+x1*y4-x2*y4)*(z1-zdet))


       r4pa4 =
     -  ((y1-ydet)*(x2*z1-x3*z1-x1*z2+x3*z2+x1*z3-x2*z3)+
     -   (x1-xdet)*(-(y2*z1)+y3*z1+y1*z2-y3*z2-y1*z3+y2*z3)+
     -   (-(x2*y1)+x3*y1+x1*y2-x3*y2-x1*y3+x2*y3)*(z1-zdet))

          inside = 0
           if(r1pa1 .ge. zero .and. r2pa2 .ge. zero .and.
```

```
     1        r3pa3 .ge. zero .and. r4pa4 .ge. zero ) then
              inside = 1
              totin  = totin + 1
            endif

            if(idebug .eq. 2) then
               write(*,*)
               write(*,*) '------------------------------'
               write(*,*)
               write(*,*) 'HEDet3D: Side = ',s
                  write(*,*) 'p1=',p1,  ' p2 = ',p2,' z = ',z,' f = ',f
                  write(*,*) 'x1 = ',x1,' y1 = ',y1,' z1 = ',z1
                  write(*,*) 'x2 = ',x2,' y2 = ',y2,' z2 = ',z2
                  write(*,*) 'x3 = ',x3,' y3 = ',y3,' z3 = ',z3
                  write(*,*) 'x4 = ',x4,' y4 = ',y4,' z4 = ',z4
                 write(*,*) 'xdet = ',xdet,' ydet = ',ydet,' zdet = ',
     1                zdet
                 write(*,*) 'r1pa1 = ',r1pa1,' r2pa2 = ',
     1                  r2pa2,' r3pa3 = ',r3pa3,' r4pa4 = ',
     1                r4pa4
                write(*,*) 'inside = ',inside,' totin = ',totin
               write(*,*)
               write(*,*) '------------------------------'
               write(*,*)
            endif

            if(inside .eq. 1) then

               phet(p1) = sqrt( (xdet-x1)**2 + (ydet-y1)**2
     1                     +(zdet-z1)**2 ) /
     1                  detvel(s) + tdet
               phet(p2) = sqrt( (xdet-x2)**2 + (ydet-y2)**2
     1                     +(zdet-z2)**2 ) /
     1                  detvel(s) + tdet
               zhet(z)   = sqrt( (xdet-x3)**2 + (ydet-y3)**2
     1                      +(zdet-z3)**2 ) /
     1                  detvel(s) + tdet
               fhet(f)   = sqrt( (xdet-x4)**2 + (ydet-y4)**2
     1                      +(zdet-z4)**2 ) /
     1                  detvel(s) + tdet
c
c     Another detonation point could be closer to the point than
c     the current detonation point so take the minimum time.
c
               phet(p1)  = amin1(phet(p1),pheto(p1))
               pheto(p1) = phet(p1)
               phet(p2)  = amin1(phet(p2),pheto(p2))
               pheto(p2) = phet(p2)
               zhet(z)   = amin1(zhet(z),zheto(z))
               zheto(z)  = zhet(z)
               fhet(f)   = amin1(fhet(f),fheto(f))
               fheto(f)  = fhet(f)

               if(idebug .eq. 3) then
```

```
            write(*,*)
             write(*,*) '-----------------------------------'
            write(*,*)
            write(*,*) ' DETONATOR NUMBER ',kkdl
            write(*,*)
           write(*,*) 'DETONATOR POSITION AND TIME:'
            write(*,*)
              write(*,*) 'x = ',dxt(1,kkdl),' y = ',dxt(2,kkdl),
     1              ' z = ',dxt(3,kkdl),' TIME = ',dxt(4,kkdl)
            write(*,*)
             write(*,*) 'SIDE ',s
            write(*,*)
            write(*,*) 'POINT p1:'
              write(*,*) '          x = ',x1,' y = ',y1,' z = ',z1
            write(*,*) '         BURN TIME = ',phet(p1)
            write(*,*) 'POINT p2:'
              write(*,*) '          x = ',x2,' y = ',y2,' z = ',z2
            write(*,*) '         BURN TIME = ',phet(p2)
            write(*,*) 'ZONE:'
              write(*,*) '          x = ',x3,' y = ',y3,' z = ',z3
            write(*,*) '         BURN TIME = ',zhet(z)
            write(*,*) 'FACE:'
              write(*,*) '          x = ',x4,' y = ',y4,' z = ',z4
            write(*,*) '         BURN TIME = ',fhet(f)
           endif

           if(idebug .eq. 1) then
              write(*,*)
               write(*,*) '*********************************'
              write(*,*)
              write(*,*) 'HEDet3D:'
                 write(*,*) 'p1=',p1,' phet(p1)=',phet(p1),'
     1                    p2=',p2,
     1                    ' phet(p2) = ',phet(p2),' z=',z,
     1                    ' zhet=',zhet(z), ' s=',s
              write(*,*)
           endif

         endif

       endif

     enddo

   enddo

   if(totin .eq. 0)
  1   call Fatal('HEDet3D: Detonators must be inside mesh')

   write(*,*)

   return
   end
```

II. 3D LUND CODING

```
c-------------------------------------------------------------

      subroutine HELund3D(dd)
c
c     Calculate the Lund HE burn times for 3D
c

      access /dd/ kk4ll,fnul,
     1            kk3ll,kksll, kksl,
     1            kkpll,px(kk3ll,kkpll),
     1            kkzll,zx(kk3ll,kkzll),
     1            kkfll,fx(kk3ll,kkfll),
     1            phet(kkpll),pheto(kkpll),
     1            zhet(kkzll),zheto(kkzll),
     1            fhet(kkfll),fheto(kkfll),
     1            kstyp(kksll),
     1            kksp1(kksll),kksp2(kksll),kksz(kksll),kksf(kksll),
     1            detvel(kksll),
     1            mheiter,zero,
     1            idebug

       integer s,p1,p2,z,f,d,
     1         heiter,
     1         success,bad,
     1         i1,i2,i3,i4,flag

       real*8 x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,z3,z4,
     1        tmax,t21,t31,t41,t1,t2,t3,t0tmp,
     1        x10,y10,z10,x20,y20,z20,x30,y30,z30,
     1        x12,y12,d1,disc,
     1        ax,ay,az,bx,by,a,b,c,dface,
     1        gx,gy,gz,gc10,gc20,
     1        c1,c2,c3,c4,c5,c6,
     1        xi,yi,zi,
     1        r4pcr3pda,r2pcr4pda,r3pcr2pda,
     1        ihe,tmp

c
c     Iterate until the HE burn times are no longer decreasing
c
      do heiter=1,mheiter

c
c     loop over "sides"
c
         do s=1,kksl

            if(kstyp(s).eq.1 .and. detvel(s) .ne. fnul) then

               p1 = kksp1(s)
               p2 = kksp2(s)
               z  = kksz(s)
```

```
            f  = kksf(s)

c
c     Calculate the time for the tet faces
c
c     Face f-p1-z
            call dtetface(fhet(f),fheto(f),fx(1,f),fx(2,f),fx(3,f),
     1           phet(p1),pheto(p1),px(1,p1),px(2,p1),px(3,p1),
     1           zhet(z),zheto(z),zx(1,z),zx(2,z),zx(3,z),
     1           zero,detvel(s))

c     Face f-p2-z
            call dtetface(fhet(f),fheto(f),fx(1,f),fx(2,f),fx(3,f),
     1           phet(p2),pheto(p2),px(1,p2),px(2,p2),px(3,p2),
     1           zhet(z),zheto(z),zx(1,z),zx(2,z),zx(3,z),
     1           zero,detvel(s))

c     Face f-p1-p2
            call dtetface(fhet(f),fheto(f),fx(1,f),fx(2,f),fx(3,f),
     1           phet(p1),pheto(p1),px(1,p1),px(2,p1),px(3,p1),
     1           phet(p2),pheto(p2),px(1,p2),px(2,p2),px(3,p2),
     1           zero,detvel(s))

c     Face p1-p2-z
            call dtetface(phet(p1),pheto(p1),px(1,p1),
     1           px(2,p1),px(3,p1),
     1           phet(p2),pheto(p2),px(1,p2),px(2,p2),px(3,p2),
     1           zhet(z),zheto(z),zx(1,z),zx(2,z),zx(3,z),
     1           zero,detvel(s))

c
c     Three of the four times must be known to use the Lund method
c     in 3D
c
            ihe = 0
            if(phet(p1) .eq. fnul) ihe = ihe + 1
            if(phet(p2) .eq. fnul) ihe = ihe + 1
            if(zhet(z)  .eq. fnul) ihe = ihe + 1
            if(fhet(f)  .eq. fnul) ihe = ihe + 1
            if(ihe .lt. 2) then
c
c     Find the maximum HE burn time and recalculate that point
c     from the other three points of the tet
c
               if(phet(p1).ge.phet(p2).and.phet(p1).ge.zhet(z)
     1              .and.phet(p1).ge.fhet(f)) then
c
c     Time at p1 is being recalculated
c
                  i1 = p1
                  i2 = p2
                  i3 = z
                  i4 = f
                  flag = 0
```

```
                x1 = px(1,i1)
                x2 = px(1,i2)
                x3 = zx(1,i3)
                x4 = fx(1,i4)

                y1 = px(2,i1)
                y2 = px(2,i2)
                y3 = zx(2,i3)
                y4 = fx(2,i4)

                z1 = px(3,i1)
                z2 = px(3,i2)
                z3 = zx(3,i3)
                z4 = fx(3,i4)

                ax = y3*z2-y4*z2-y2*z3+y4*z3+y2*z4-y3*z4
                ay = -(x3*z2)+x4*z2+x2*z3-x4*z3-x2*z4+x3*z4
                az = x3*y2-x4*y2-x2*y3+x4*y3+x2*y4-x3*y4

            elseif(phet(p2).ge.phet(p1).and.phet(p2).gt.
     1              zhet(z).and.phet(p2).ge.fhet(f)) then
c
c     Time at p2 is being recalculated
c
                i1 = p2
                i2 = p1
                i3 = z
                i4 = f
                flag = 3

                x1 = px(1,i1)
                x2 = px(1,i2)
                x3 = zx(1,i3)
                x4 = fx(1,i4)

                y1 = px(2,i1)
                y2 = px(2,i2)
                y3 = zx(2,i3)
                y4 = fx(2,i4)

                z1 = px(3,i1)
                z2 = px(3,i2)
                z3 = zx(3,i3)
                z4 = fx(3,i4)

                ax = -(y3*z2)+y4*z2+y2*z3-y4*z3-y2*z4+y3*z4
                ay = x3*z2-x4*z2-x2*z3+x4*z3+x2*z4-x3*z4
                az = -(x3*y2)+x4*y2+x2*y3-x4*y3-x2*y4+x3*y4

            elseif(zhet(z).gt.phet(p1).and.zhet(z).ge.
     1              phet(p2).and.zhet(z).gt.fhet(f)) then
c
c     Time at z is being recalculated
```

```
c
                i1 = z
                i2 = p1
                i3 = p2
                i4 = f
                flag = 1

                x1 = zx(1,i1)
                x2 = px(1,i2)
                x3 = px(1,i3)
                x4 = fx(1,i4)

                y1 = zx(2,i1)
                y2 = px(2,i2)
                y3 = px(2,i3)
                y4 = fx(2,i4)

                z1 = zx(3,i1)
                z2 = px(3,i2)
                z3 = px(3,i3)
                z4 = fx(3,i4)

                ax = y3*z2-y4*z2-y2*z3+y4*z3+y2*z4-y3*z4
                ay = -(x3*z2)+x4*z2+x2*z3-x4*z3-x2*z4+x3*z4
                az = x3*y2-x4*y2-x2*y3+x4*y3+x2*y4-x3*y4

             elseif(fhet(f).gt.phet(p1).and.fhet(f).gt.phet(p2)
     1               .and.fhet(f).gt.zhet(z)) then
c
c     Time at f is being recalculated
                i1 = f
                i2 = p1
                i3 = p2
                i4 = z
                flag = 2

                x1 = fx(1,i1)
                x2 = px(1,i2)
                x3 = px(1,i3)
                x4 = zx(1,i4)

                y1 = fx(2,i1)
                y2 = px(2,i2)
                y3 = px(2,i3)
                y4 = zx(2,i4)

                z1 = fx(3,i1)
                z2 = px(3,i2)
                z3 = px(3,i3)
                z4 = zx(3,i4)

                ax = -(y3*z2)+y4*z2+y2*z3-y4*z3-y2*z4+y3*z4
                ay = x3*z2-x4*z2-x2*z3+x4*z3+x2*z4-x3*z4
                az = -(x3*y2)+x4*y2+x2*y3-x4*y3-x2*y4+x3*y4
```

```
            else
              cycle
            endif

            if(flag .eq. 0 .or. flag .eq. 3) then
c
c     Either point p1 or point p2 is being recalculated.
c
              t1  = phet(i2)
              t2  = zhet(i3)
              t3  = fhet(i4)

              x10 = px(1,i2)-px(1,i1)
              y10 = px(2,i2)-px(2,i1)
              z10 = px(3,i2)-px(3,i1)
              x20 = zx(1,i3)-px(1,i1)
              y20 = zx(2,i3)-px(2,i1)
              z20 = zx(3,i3)-px(3,i1)
              x30 = fx(1,i4)-px(1,i1)
              y30 = fx(2,i4)-px(2,i1)
              z30 = fx(3,i4)-px(3,i1)

              t21 = sqrt(x10**2+y10**2+z10**2)/
     1              detvel(s)+phet(i2)
              t31 = sqrt(x20**2+y20**2+z20**2)/
     1              detvel(s)+zhet(i3)
              t41 = sqrt(x30**2+y30**2+z30**2)/
     1              detvel(s)+fhet(i4)
              x12 = px(1,i2)-zx(1,i3)
              y12 = px(2,i2)-zx(2,i3)
            elseif(flag .eq. 1) then
c
c     Point z is being recalculated.
c
              t1  = phet(i2)
              t2  = phet(i3)
              t3  = fhet(i4)

              x10 = px(1,i2)-zx(1,i1)
              y10 = px(2,i2)-zx(2,i1)
              z10 = px(3,i2)-zx(3,i1)
              x20 = px(1,i3)-zx(1,i1)
              y20 = px(2,i3)-zx(2,i1)
              z20 = px(3,i3)-zx(3,i1)
              x30 = fx(1,i4)-zx(1,i1)
              y30 = fx(2,i4)-zx(2,i1)
              z30 = fx(3,i4)-zx(3,i1)

              t21 = sqrt(x10**2+y10**2+z10**2)/
     1              detvel(s)+phet(i2)
              t31 = sqrt(x20**2+y20**2+z20**2)/
     1              detvel(s)+phet(i3)
              t41 = sqrt(x30**2+y30**2+z30**2)/
```

```
     1                detvel(s)+fhet(i4)
                    x12 = px(1,i2)-px(1,i3)
                    y12 = px(2,i2)-px(2,i3)
                  else
c
c     Point f is being recalculated.
c
                    t1 = phet(i2)
                    t2 = phet(i3)
                    t3 = zhet(i4)

                    x10 = px(1,i2)-fx(1,i1)
                    y10 = px(2,i2)-fx(2,i1)
                    z10 = px(3,i2)-fx(3,i1)
                    x20 = px(1,i3)-fx(1,i1)
                    y20 = px(2,i3)-fx(2,i1)
                    z20 = px(3,i3)-fx(3,i1)
                    x30 = zx(1,i4)-fx(1,i1)
                    y30 = zx(2,i4)-fx(2,i1)
                    z30 = zx(3,i4)-fx(3,i1)

                    t21 = sqrt(x10**2+y10**2+z10**2)/
     1                    detvel(s)+phet(i2)
                    t31 = sqrt(x20**2+y20**2+z20**2)/
     1                    detvel(s)+phet(i3)
                    t41 = sqrt(x30**2+y30**2+z30**2)/
     1                    detvel(s)+zhet(i4)
                  endif

                  d1 = -x30*y20*z10 + x20*y30*z10
     1                 + x30*y10*z20 - x10*y30*z20 -
     1                 x20*y10*z30 + x10*y20*z30
c
                  c1 = y20*z10 - y30*z10 - y10*z20 + y30*z20 +
     1                 y10*z30 - y20*z30
                  c2 = -t3*y20*z10 + t2*y30*z10 + t3*y10*z20 -
     1                 t1*y30*z20 - t2*y10*z30 + t1*y20*z30

                  c3 = -x20*z10 + x30*z10 + x10*z20 -
     1                 x30*z20 - x10*z30 + x20*z30
                  c4 = t3*x20*z10 - t2*x30*z10 - t3*x10*z20 +
     1                 t1*x30*z20 + t2*x10*z30 - t1*x20*z30

                  c5 = x20*y10 - x30*y10 - x10*y20 + x30*y20 +
     1                 x10*y30 - x20*y30
                  c6 = -t3*x20*y10 + t2*x30*y10 + t3*x10*y20 -
     1                 t1*x30*y20 - t2*x10*y30 + t1*x20*y30

                  disc =
     1                 (2*c1*c2 + 2*c3*c4 + 2*c5*c6)**2*
     1                 detvel(s)**4 - 4*(c1**2 + c3**2 + c5**2)
     1                 *detvel(s)**2*(-d1**2 + c2**2*detvel(s)**2 +
     1                 c4**2*detvel(s)**2 + c6**2*detvel(s)**2)
```

```
            if (disc.ge.zero) then

              disc = sqrt(disc)
              t0tmp =
     1          ((-2*c1*c2 - 2*c3*c4 - 2*c5*c6)*detvel(s)**2 +
     1          disc)/(2.*(c1**2 + c3**2 + c5**2)*detvel(s)**2)
c
c     The derivatives of the time w.r.t. x, y and z are:
c
              gx = (t0tmp * c1 + c2) / d1
              gy = (t0tmp * c3 + c4) / d1
              gz = (t0tmp * c5 + c6) / d1
c
c     Check local causality. t0tmp is a possible time only if
c     causality is satisfied. This means the gradient of the time
c     t0 must pass within the opposite face of the tet.
c
c     Calculate the eq. of the plane for the tet face opposite
c     the time being calc. using the 3 points of the tet face.
c     ( A x + B y + C z + D = 0 )
c

              a = ax
              b = ay
              c = az

              dface = -a*x2-b*y2-c*z2
c
c     Solve the equation for the plane and the equations for
c     the line representing the gradient of the time at t0 to
c     give the x, y, and z coodinates where the two intersect.
c     This point is at xi, yi, and zi.
c
              tmp = (A*gx+B*gy+C*gz)

              if( tmp .ne. zero ) then
                xi = -(dface*gx-B*gy*x1-C*gz*x1+B*gx*
     1               y1+C*gx*z1)/tmp

                yi = -(dface*gy+A*gy*x1-A*gx*y1-C*gz*
     1               y1+C*gy*z1)/tmp

                zi = -(dface*gz+A*gz*x1+B*gz*y1-A*gx*
     1               z1-B*gy*z1)/tmp

              endif
c
c     Determine if the intercept point (xi, yi, zi)
c     is within the triangle forming the tet side.
c
c     r4pcr3pda is the vector at point 4 - the vector of the
c     intercept point crossed with the vector at point 3 -
c     the intercept point, and that quantity dotted into the
c     vector of the area of the point where the time, t0, is
```

```
c     being calculated.
c
                if( tmp .ne. zero) then
                  r4pcr3pda =
     1               az*(x4*y3-xi*y3-x3*y4+xi*y4+x3*yi-x4*yi)+
     1               ay*(-(x4*z3)+xi*z3+x3*z4-xi*z4-x3*zi+x4*zi)+
     1               ax*(y4*z3-yi*z3-y3*z4+yi*z4+y3*zi-y4*zi)

                  r2pcr4pda =
     1               az*(-(x4*y2)+xi*y2+x2*y4-xi*y4-x2*yi+x4*yi)+
     1               ay*(x4*z2-xi*z2-x2*z4+xi*z4+x2*zi-x4*zi)+
     1               ax*(-(y4*z2)+yi*z2+y2*z4-yi*z4-y2*zi+y4*zi)

                  r3pcr2pda =
     1               az*(x3*y2-xi*y2-x2*y3+xi*y3+x2*yi-x3*yi)+
     1               ay*(-(x3*z2)+xi*z2+x2*z3-xi*z3-x2*zi+x3*zi)+
     1               ax*(y3*z2-yi*z2-y2*z3+yi*z3+y2*zi-y3*zi)

                endif


                 if(flag .eq. 2 .or. flag .eq. 3 ) then
                  r4pcr3pda = - r4pcr3pda
                  r2pcr4pda = - r2pcr4pda
                  r3pcr2pda = - r3pcr2pda
                endif

                 if(r4pcr3pda .ge. zero .and.
     1              r2pcr4pda .ge. zero .and.
     1              r3pcr2pda .ge. zero .and.
     1             tmp .ne. zero ) then

                   if(flag .eq. 0 .or. flag .eq. 3) then
                     if(t0tmp.gt.phet(i2).and.t0tmp.gt.
     1                   zhet(i3).and.t0tmp.gt.
     1                   fhet(i4)) then
                         phet(i1) = amin1(t0tmp,t21,t31,t41,
     1                          phet(i1),pheto(i1))
                     else
                          phet(i1) = amin1(t21,t31,t41,
     1                          phet(i1),pheto(i1))
                     endif
                   elseif(flag .eq. 1) then
                     if(t0tmp.gt.phet(i2).and.t0tmp.gt.
     1                   phet(i3).and.t0tmp.gt.
     1                   fhet(i4)) then
                       zhet(i1) = amin1(t0tmp,t21,t31,t41,
     1                          zhet(i1),zheto(i1))
                     else
                       zhet(i1) = amin1(t21,t31,t41,
     1                          zhet(i1),zheto(i1))
                     endif
                   else
                     if(t0tmp.gt.phet(i2).and.t0tmp.gt.
```

```
1                    phet(i3).and.t0tmp.gt.zhet(i4)) then
                   fhet(i1) = amin1(t0tmp,t21,t31,t41,
1                           fhet(i1),fheto(i1))
                 else
                   fhet(i1) = amin1(t21,t31,t41,
1                           fhet(i1),fheto(i1))
                 endif
               endif
             else


               if(flag .eq. 0 .or. flag .eq. 3) then

                 phet(i1) = amin1(t21,t31,t41,
1                           phet(i1),pheto(i1))

               elseif(flag .eq. 1) then
                 zhet(i1) = amin1(t21,t31,t41,
1                           zhet(i1),zheto(i1))
               else
                 fhet(i1) = amin1(t21,t31,t41,
1                           fhet(i1),fheto(i1))
               endif

             endif

           else
               if(flag .eq. 0 .or. flag .eq. 3) then
                 phet(i1) = amin1(t21,t31,t41,
1                           phet(i1),pheto(i1))

               elseif(flag .eq. 1) then
                 zhet(i1) = amin1(t21,t31,t41,
1                           zhet(i1),zheto(i1))
               else
                 fhet(i1) = amin1(t21,t31,t41,
1                           fhet(i1),fheto(i1))
               endif

           endif

             if( idebug .eq. 1 ) then
               write(*,*)
                write(*,*) '-----------------------------------'
               write(*,*)
               write(*,*) 'HELund3D: HE ITERATION NUMBER',heiter
                   write(*,*) 'p1=',p1,' p2=',p2,' z=',z,' f = ',f
                 write(*,*) 's=',s,' phet(p1)=',phet(p1),'
1                          phet(p2)=',
1                           phet(p2),'  zhet = ',zhet(z),
1                          ' fhet(f) = ',fhet(f)
                 write(*,*) 'x1=',x1,' y1=',y1,' z1 = ',z1
                 write(*,*) 'x2=',x2,' y2=',y2,' z2 = ',z2
                 write(*,*) 'x3=',x3,' y3=',y3,' z3 = ',z3
```

```
                write(*,*) 'x4=',x4,' y4=',y4,' z4 = ',z4
               write(*,*) 'pheto(p1)=',pheto(p1),' pheto(p2)=',
     1                pheto(p2),
     1                 ' zheto(z) =',zheto(z),' fheto(f) =',
     1                fheto(f)
               write(*,*) 't0tmp = ',t0tmp,' t21 = ',t21,'
     1                 t31 = ',t31,' t41 = ',t41
                write(*,*) 'i1=',i1,' i2=',i2,' i3=',i3,' i4=',
     1                 i4,' flag=',flag
               write(*,*) 'disc = ',disc,' d1 = ',d1
                write(*,*) 'ax = ',ax,' ay = ',ay,' az = ',az
                write(*,*) 'Gx = ',gx,' Gy = ',gy,' gz = ',gz
              write(*,*) 'r4pcr3pda = ',r4pcr3pda
              write(*,*) 'r2pcr4pda = ',r2pcr4pda
              write(*,*) 'r3pcr2pda = ',r3pcr2pda
                write(*,*) 'a = ',a,' b = ',b,' c = ',c
               write(*,*) 'dface = ',dface,' tmp = ',tmp
                write(*,*) 'xi = ',xi,' yi = ',yi,' zi = ',zi
                write(*,*) 'c1 = ',c1,' c2 = ',c2,' c3 = ',c3
                write(*,*) 'c4 = ',c4,' c5 = ',c5,' c6 = ',c6

            endif

          endif

        endif

     enddo

  bad = 0

  do s=1,kksl

    if(kstyp(s).eq.1) then

      p1 = kksp1(s)
      if( phet(p1) .lt. fnul) then
         bad = 1
         exit
      endif

    endif

  enddo

  if(bad .eq. 0) call Fatal('HELund3D: BAD TIME ITERATION')

  success = 0

  do s=1,kksl

    if(kstyp(s).eq.1 .and. detvel(s) .ne. fnul) then

      p1 = kksp1(s)
```

```
            p2 = kksp2(s)
            z  = kksz(s)
            f  = kksf(s)

            if( phet(p1) .lt. pheto(p1) ) then
               success = success + 1
               pheto(p1) = phet(p1)
            endif
            if( phet(p2) .lt. pheto(p2) ) then
               success = success + 1
               pheto(p2) = phet(p2)
            endif
            if( zhet(z) .lt. zheto(z) ) then
               success = success + 1
               zheto(z) = zhet(z)
            endif
            if( fhet(f) .lt. fheto(f) ) then
               success = success + 1
               fheto(f) = fhet(f)
            endif

c           write(*,*) 'HELUND3D: HE ITERATION NUMBER',heiter
c             write(*,*) 's = ',s,' phet(p1) = ',phet(p1),' phet(p2) = ',
c    1                phet(p2),'  zhet = ',zhet(z)

          endif

         enddo

         if(success .eq. 0 ) then
            write(*,*)
            write(*,*) '--------- HE LUND 3D BURN ----------------'
            write(*,*)
            write(*,*) 'NUMBER OF ITERATIONS = ',heiter
            write(*,*) 'MAXIMUM ITERATIONS ALLOWED = ',mheiter
            write(*,*)
            write(*,*) '---------------------------------------'
            write(*,*)
            exit
         endif

        enddo

        if(success .ne. 0 )
       1   call Fatal('HELund3D: HE iteration did not converge ')

        return
        end
```

III. TETRADEDRAL FACE CALCULATION

```
c-------------------------------------------------------------

      subroutine dtetface(t1,t1old,x1,y1,z1,
     1                    t2,t2old,x2,y2,z2,
     1                    t3,t3old,x3,y3,z3,
     1                   zero,detvel)
c
c     This routine is the driver for the calculation of
c     the tet face waves.
c
      real*8 t1,t1old,x1,y1,z1,t2,t2old,x2,y2,z2,
     1       t3,t3old,x3,y3,z3,zero,detvel

      if( t1 .gt. t2 .and. t1.gt.t3 ) then
          call tetface(t1,x1,y1,z1,x2,y2,z2,x3,y3,z3,
     1                 t2,t3,zero,detvel,t1,t1old)

c
c     Calculate time at point 2 of the triangle
c
      elseif( t2.gt.t1 .and. t2.gt.t3) then
          call tetface(t2,x2,y2,z2,x1,y1,z1,x3,y3,z3,
     1                 t1,t3,zero,detvel,t2,t2old)

      else
c
c     Calculate time at point 3 of the triangle
c
          call tetface(t3,x3,y3,z3,x1,y1,z1,x2,y2,z2,
     1                 t1,t2,zero,detvel,t3,t3old)

      endif

      return
      end

c-------------------------------------------------------------
      subroutine tetface(tface,x0,y0,z0,x1,y1,z1,x2,y2,z2,
     1                   t1,t2,zero,detvel,tnew,told)
c
c     This subroutine calculates the 2D solution for tet
c     faces. For each tet, 3 2D solutions are obtained in
c     sequence. This is done in order to take into account
c     waves that could propagate within a tet face.
c
      real*8 tface,x0,y0,z0,x1,y1,z1,x2,y2,z2,
     1       t1,t2,zero,detvel,tnew,told,
     1       t10,t20,t0tmp,
     1        x10,y10,x20,y20,x12,y12,
     1        d1,ax,ay,bx,by,a,b,c,disc,gx,gy,
     1       gc10,gc20,dotprod,dist1,dist2,theta
c
```

```
c     Convert the 3D coodinate system (x,y,z) to
c     a 2D system (xp,yp) within the tet face. The origin
c     is at the vertex where the time t0 is being calculated.
c
      dotprod = (x2-x0)*(x1-x0)+(y2-y0)*(y1-y0)+(z2-z0)*(z1-z0)
      dist2 = sqrt((x2-x0)**2+(y2-y0)**2+(z2-z0)**2)
      dist1 = sqrt((x1-x0)**2+(y1-y0)**2+(z1-z0)**2)
      theta = acos(dotprod/(dist1*dist2))
      t10 = dist1/detvel+t1
      t20 = dist2/detvel+t2

      x10 = dist1
      y10 = zero
      x20 = dist2 * cos(theta)
      y20 = dist2 * sin(theta)

      x12 = x10-x20
      y12 =    -y20
c
c     Solve the 2D Lund equations.
c

            d1 = x20*y10-x10*y20
            ax= - y12/d1
            ay=   x12/d1
            bx=  (t2*y10 - t1*y20)/d1
            by= -(t2*x10 - t1*x20)/d1
            a = ax**2+ay**2
            b = 2*(ax*bx+ay*by)
            c = (bx**2+by**2-1/detvel**2)
            disc=b**2-4.*a*c

            if (disc.ge.0) then


              t0tmp= (-b + sqrt(disc))/(2.*a)

              gx = ax*t0tmp+bx
              gy = ay*t0tmp+by

              gc10 = gx*y10-gy*x10
              gc20 = gx*y20-gy*x20

c
c     Check 2D local causality
c
              if(gc10*gc20 .lt. zero) then

                 if(t0tmp .gt. t1 .and. t0tmp .gt. t2)
     1              then
                    tface = amin1(tface,t0tmp,t10,t20,
     1                    tnew,told)
                 else
                    tface = amin1(tface,t10,t20,tnew,told)
```

```
                endif

             else

                tface = amin1(tface,t10,t20,tnew,told)

             endif

          else
             tface = amin1(tface,t10,t20,tnew,told)
          endif

c                write(*,*) '^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^'
c               write(*,*) 'dist1=',dist1,' dist2=',dist2
c               write(*,*) 'theta=',theta,' t10 = ',t10
c                write(*,*) 't20=',t20,' x10=',x10,' y10=',y10
c                write(*,*) 'x20=',x20,' y20=',y20,' x12=',x12
c               write(*,*) 'y12=',y12,' t0tmp=',t0tmp
c               write(*,*) 'gc10=',gc10,' gc20=',gc20
c               write(*,*) 'tnew=',tnew,' told=',told
c               write(*,*) 't1=',t1,' t2=',t2,' tface=',tface
c                write(*,*) '^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^'
      return
      end
```

APPENDIX C

TWO-DIMENSIONAL SAMPLE INPUT

3 by 3 Input

```
$-----------------------------------------------------------------------
$-----------------------------------------------------------------------
$-----------------------------------------------------------------------
$ LUND HE burn test problem
$-----------------------------------------------------------------------
$-----------------------------------------------------------------------
$-----------------------------------------------------------------------
$  GLOBAL

$---  global level parameters
      mk /global
        title="LUND.2D burn Test"

$-------------------------------------
$  MESH = cartesian
$     Create a mesh and call it 'Grid'

      mk /global/mesh
      mk /global/mesh/geometry/cart2

$     Generate own grid
      mk /global/mesh/zoner/kl

$          Lmax  k3
$       p4+------+p3
$      ^ |      |
$   K1 ^ |      | Kmax
$      L |      |
$       p1+------+p2
$        k1   L1    K>>>>>

      integer N, nk, nl
      N=2
      kk3ll=2

      real p1(N),p2(N),p3(N),p4(N),alf(N)
      real rk1(N),rk2(N),rk3(N),rk4(N)
$     NOTE: rk1 ... are real since AJAX call implementation
$          currently passes only reals

      kmax=3
      lmax=3
      rk1= 1  1
      rk3= 3 3
      alf= 1. 1.

      p1=  0 0
      p2=  1 0
```

```
      p3=  1 1
      p4=  0 1
      call G2Block dd alf rk1 rk3 p1 p2 p3 p4

$--------------------------------------
$  HYDRO parameters

$ universe
      mk /global/mesh/func(univ)/universe

$--------------------------------------
$  REGIONS

      mk /global/mesh/kregion(Universe)/onefunc
        fname="univ"
$--------------------------------------
$ DETONATION TIMES
      mk /global/mesh/heburn/hedet

       kkdll= 10

       dxt=0.0 0.0 0.0

       idebug = 0

$--------------------------------------
$ HIGH EXPLOSIVE EDGE LIGHTING
$     mk /global/mesh/heburn/heedge
$       mheiter= 20
$       idebug = 0
$--------------------------------------
$ HIGH EXPLOSIVE LUND LIGHTING
       mk /global/mesh/heburn/helund
         mheiter= 100
         idebug = 0
$--------------------------------------
$ HIGH EXPLOSIVE DIRECT LIGHTING
$     mk /global/mesh/heburn/hedirect
$       mheiter= 20
$       idebug = 0

$      kkdll= 10

$       dxt=0. 0. 0.
$--------------------------------------
$  MATERIALS

$     Create a material and call it 'mat1'
      mk /global/mesh/mat(he1)/mhe/gamma
        r0=one                    $ reference density (at node /mat)
         g=5./3.
        detvelhe=1.0
        heenergy=.1111111111
```

```
      mk /global/mesh/mat(he1)/initmat
        region="Universe"
        density=1.
        energy=1.
        volfrac=one                $ default value

$-------------------------------------
$  POB and WIN parameters
      cd /global/mesh/heburn
        alias phet phet
        alias zhet zhet

      mk /global/mesh/pob/win
        ncolors=40
        mk map
          scale(:,:)=0 1 0 1
$         coord = "px"           $ default is px
        mk ../wire
          wiretype="ZoneEdge"     $ default is "ZoneEdge"
        mk ../contour
          on="on"
          name="phet"
          vmin=0.
          vmax=2.
$       mk ../contour
$         on="off"
$         name="_zr"
$         vmin=0.
$         vmax=64.
$       mk ../gour
$         on="off"
$         name="_zr"
$         vmin=0.
$         vmax=64.
$       mk ../vector
$         on="off"
$         labels="off"
$         name="_pu"
$         vmin=0.
$         vmax=0.
$       mk ../insert
$         on="off"

        mk ../fin

$-------------------------------------
$  EXECUTION


cd /global
      call ttyon

send TestHEBurn
send VIEW
```

```
cd /global/mesh/heburn/helund
print zhet
print phet
tty

return
```

APPENDIX D

THREE-DIMENSIONAL SAMPLE INPUT

3 by 3 by 3 Input

cat 3D.flg

```
$-----------------------------------------------------------------------
$-----------------------------------------------------------------------
$-----------------------------------------------------------------------
$ HE burn test problem
$-----------------------------------------------------------------------
$-----------------------------------------------------------------------
$-----------------------------------------------------------------------
$  GLOBAL

$---  global level parameters
      mk /global
        title="LUND.3D burn Test"

$-------------------------------------
$  MESH = cylindrical

      mk /global/mesh(Grid)
      mk /global/mesh(Grid)/geometry/axis2

$     Generate own grid
      mk /global/mesh(Grid)/zoner/kl

$           Lmax  k3
$       p4+------+p3
$     ^ |      |
$   K1 ^ |      | Kmax
$      L |      |
$       p1+------+p2
$        k1   L1    K>>>>>

      integer N, nk, nl
      N=2
      kk3ll=2

      real p1(N),p2(N),p3(N),p4(N),alf(N)
      real rk1(N),rk2(N),rk3(N),rk4(N)
$     NOTE: rk1 ... are real since AJAX call implementation
$           currently passes only reals

      kmax=3
      lmax=3
      rk1= 1  1
      rk3= 3 3
      alf= 1. 1.

      p1=  0 0
      p2=  1 0
```

```
      p3=  1 1
      p4=  0 1
      call G2Block dd alf rk1 rk3 p1 p2 p3 p4

$---  Create a 3D mesh (its default name is 'mesh.0')
 $     Create 3D FROM 2D MESH by translation
       mk /global/mesh/geometry/cart3
       mk /global/mesh/zoner/trans2to3
         trans=0 .5 0
         mmax=3
         mesh2d="Grid"

$--------------------------------------
$  HYDRO parameters

$ universe
      mk /global/mesh/func(univ)/universe

$--------------------------------------
$  REGIONS

      mk /global/mesh/kregion(Universe)/onefunc
        fname="univ"
$--------------------------------------
$ DETONATION TIMES
      mk /global/mesh/heburn/hedet

        dxt=0. 0. 0. 0.

        idebug = 0

$--------------------------------------
$ HIGH EXPLOSIVE EDGE LIGHTING
$     mk /global/mesh/heburn/heedge
$       mheiter= 20
        idebug = 0
$--------------------------------------
$ HIGH EXPLOSIVE LUND LIGHTING
       mk /global/mesh/heburn/helund
$        mheiter= 100
         idebug = 0
$--------------------------------------
$ HIGH EXPLOSIVE DIRECT LIGHTING
$     mk /global/mesh/heburn/hedirect
$       mheiter= 20
$       idebug = 0

$      kkdll= 10

$       dxt=0. 0. 0.
$--------------------------------------
$  MATERIALS

$     Create a material and call it 'mat1'
```

```
    mk /global/mesh/mat(he1)/mhe/gamma
      r0=one                    $ reference density (at node /mat)
       g=5./3.
      detvelhe=1.0

    mk /global/mesh/mat(he1)/initmat
      region="Universe"
      density=1.
      energy=1.
      volfrac=one               $ default value

$-------------------------------
    mk /global/mesh/output/gmv
      vars="phet" "zhet" "fhet"
$-------------------------------------
$  POB and WIN parameters
    cd /global/mesh/heburn
      alias phet phet
      alias zhet zhet
      alias fhet fhet

    mk /global/mesh/pob/win
      ncolors=40
      mk map
        scale(:,:)=0 1 0 1
$       coord = "px"            $ default is px
      mk ../wire
       wiretype="ZoneEdge"     $ default is "ZoneEdge"
      mk ../contour
       on="on"
       name="phet"
       vmin=0.
       vmax=2.
$     mk ../contour
$       on="off"
$      name="_zr"
$       vmin=0.
$        vmax=64.
$     mk ../gour
$       on="off"
$      name="_zr"
$       vmin=0.
$        vmax=64.
$     mk ../vector
$       on="off"
$      labels="off"
$      name="_pu"
$       vmin=0.
$        vmax=0.
$     mk ../insert
$        on="off"

      mk ../fin
```

```
$-------------------------------------
$  EXECUTION


cd /global
      call ttyon

$ send GMV
send TestHEBurn
$ send VIEW
 send GMV

cd /global/mesh/heburn/helund

print fhet

print zhet

print phet

tty


return
```

Los Alamos
NATIONAL LABORATORY
Los Alamos, New Mexico 87545