FINAL REPORT

submitted to

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Space Science Branch
ATTN: Eric L. Christiansen
Houston, TX 77058

for research entitled

DESIGN OF ORBITAL DEBRIS SHIELDS FOR OBLIQUE HYPERVELOCITY IMPACT

Submitted by

Eric P. Fahrenthold
Department of Mechanical Engineering
University of Texas
Austin, TX 78712

February 14, 1994

FINAL REPORT

submitted to

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Space Science Branch
ATTN: Eric L. Christiansen
Houston, TX 77058

for research entitled

DESIGN OF ORBITAL DEBRIS SHIELDS FOR OBLIQUE HYPERVELOCITY IMPACT

Submitted by

Eric P. Fahrenthold
Department of Mechanical Engineering
University of Texas
Austin, TX 78712

February 14, 1994

ABSTRACT

A new impact debris propagation code has been written to link CTH simulations of space debris shield perforation to the Lagrangian finite element code DYNA3D, for space structure wall impact simulations. This software (DC3D) simulates debris cloud evolution using a nonlinear elastic-plastic deformable particle dynamics model, and renders computationally tractable the supercomputer simulation of oblique impacts on Whipple shield protected structures. Comparison of three dimensional, oblique impact simulations with experimental data shows good agreement over a range of velocities of interest in the design of orbital debris shielding.

Source code developed during this research is provided on the enclosed floppy disk. An abstract based on the work described in this report has been submitted to the 1994 Hypervelocity Impact Symposium.

ACKNOWLEDGMENTS

This work was funded under the NASA Regional Universities Grant Program. The assistance of Eric L. Christiansen (NASA Technical Officer) and Jeanne Lee Crews of the Space Science Branch of Johnson Space Center has been greatly appreciated. Additional support was provided by Cray Research, Inc. under the Cray University Research and Development Grant Program.

SOFTWARE COPYRIGHT NOTICE

TABLE OF CONTENTS

1. Introduction

The design of the space station Freedom and similar structures for earth orbit must include provisions for the effects of hypervelocity impact, which may arise as a result of space debris or micrometeorite encounters. This problem can be expected to become increasingly important as longer duration space missions are launched, increasing the exposure time of orbiting systems. Such missions provide increased probability of impact damage while placing greater reliability demands on vehicles and structures. Existing light gas gun facilities used in the study of hypervelocity impact effects do not generally allow for tests at velocities above ten kilometers per second, suggesting the use of computer simulation methods for orbital debris shield design at those velocities.

The accomplishment of design goals for the space station and similar structures depends in part upon the development and verification of computationally tractable models capable of describing oblique hypervelocity impact effects at velocities beyond existing experimental capabilities. Experience to date has shown that the extreme CPU time and memory requirements of standard Eulerian hydrocodes (McGlaun et al., 1990) make their use in direct simulation of three dimensional impacts on space debris shields impractical, even given supercomputer resources. In addition, current Eulerian hydrocodes do not in general rigorously account for material history effects on the failure of space structures under impact debris loading. Conventional Lagrangian finite element codes (Goudreau and Hallquist, 1982), on the other hand, are not suitable for use in simulating the shield perforation portion of the impact problem. Mesh distortion effects greatly reduce the size of the time step used in the calculations, and mandate frequent rezoning. As a result of the preceding difficulties, the only general simulation technique demonstrated to date has involved the systematic linking of Eulerian hydrocodes (of shield perforation) to Lagrangian finite element models (of debris cloud evolution and debris impact on the protected structure). This approach has been developed and implemented by the principal investigator (Fahrenthold, 1993).

The development of particle-based debris cloud evolution models [e.g. Fritts et al. (1985), Trease et al. (1990), Monaghan (1988), and Trease (1988)], offers an opportunity to directly simulate complete three-dimensional debris impact problems on existing supercomputers. Particle-based methods address the previously discussed shortcomings of Eulerian and Lagrangian codes in two ways: (1) they effectively eliminate the mesh distortion problems of Lagrangian finite element codes, and (2) they greatly improve on the CPU time efficiency of Eulerian hydrocodes while allowing for accurate tracking of material history dependent effects such as plastic deformation. As an application which has severely taxed the capabilities of conventional Eulerian and Lagrangian computer codes, orbital debris shield design is well suited to capitalize on the strengths of new particle-based modeling techniques. Hence the research presented here has: (1) developed a debris cloud evolution code which links shield perforation and wall impact simulations, (2) conducted three dimensional, supercomputer based simulations of shield impact, debris cloud evolution, and wall impact problems, and (3) evaluated and validated the computer models using data from experiments conducted at NASA Johnson Space Center. This computer simulation methodology allows for the modeling of impacts at velocities beyond ten kilometers per second which are very difficult to duplicate in the laboratory.

2. Methodology

The problem of hypervelocity impact on space structures has been an object of research since the 1950's, as reflected in the summary of Hypervelocity Impact Symposia presented during recent conferences in that series (e.g. Anderson, 1986). However both experimental and analytical research work has accelerated in recent years, with most recent NASA interest focused on debris effects on the space shuttle and planned space station. A significant data base exists for impacts at velocities below ten kilometers per second (Tower et al., 1987), including studies aimed specifically at NASA applications such as space shuttle windows (Schneider and Stilp, 1987) and debris shield

design (Yew and Kendrick, 1986). However difficulties with conducting experiments at higher velocities, relevant to both the space station and future programs, have limited the ability to evaluate new designs in a laboratory setting.

The preceding facts suggest a combined experimental and analytical approach to micrometeorite and debris shield design, using experimental data at velocities below ten kilometers per second to critique and verify computer models, which then provide a basis for higher velocity design calculations. The simulation work described here was conducted on a Cray Y-MP/864 supercomputer at the University of Texas System Center for High Performance Computing.

The principal investigator has made extensive use of existing supercomputer based Eulerian and Lagrangian hydrocodes to evaluate their use in orbital debris shield design. In general Eulerian codes are best suited to hypervelocity impact simulations where impact pressures are sufficiently large to render material strength effects generally unimportant. An Eulerian code is best used to predict initial debris cloud mass and distribution as a function of projectile material type and velocity, shield material type and geometry, and other parameters. The protected space structure must be designed to avoid spallation or fracture under the debris cloud impact. Since spallation and fracture processes are very stress and strain history dependent phenomena (Yew and Taylor, 1992, and Grady and Kipp, 1987), and since Lagrangian hydrocodes are best suited to trace stress and strain history in solid materials, a Lagrangian hydrocode model of debris cloud impact on the inner wall is most appropriate. The new modeling approach outlined here links both parts of the impact event using a new analysis methodology, significantly reducing computer resource requirements for oblique impact calculations.

3. Particle Dynamics Model

a. Introduction

This section describes a new modeling approach combining Lagrangian bond graphs (Fahrenthold and Wargo, 1994) with a selected finite element discretization

scheme to allow for direct simulation of the dynamic evolution of debris particles arising from hypervelocity impact.

b. Kinematics

The homogeneous deformation field associated with the finite element discretization employed here allows the position **x** and velocity **v** of any mass particle "P" in a particular element to be written in the form (Malvern, 1969)

$$\mathbf{x} = \mathbf{F}(t)\,(\mathbf{r} - \mathbf{r}_c) + \mathbf{c}(t) \;\; ; \;\; \mathbf{v} = \dot{\mathbf{F}}(t)\,(\mathbf{r} - \mathbf{r}_c) + \dot{\mathbf{c}}(t) \qquad (1a,b)$$

where $\mathbf{c}(t)$ and $\dot{\mathbf{c}}(t)$ are the position and velocity of the element center of mass "C", $\mathbf{r}$ and $\mathbf{r}_c$ are the position of P and C in the reference (initial undeformed) configuration of the body, and $\mathbf{F}$ is the deformation gradient tensor. Note that since $\mathbf{r}$ and $\mathbf{r}_c$ in equations (1) are constants, the motion of any particle P in the element is determined by the motion of C and by the time dependent components of the second order tensor $\mathbf{F}$, related to the rate of deformation ($\mathbf{D}$) and velocity gradient ($\mathbf{L}$) tensors by

$$\mathbf{D} = (1/2)\,[\,\mathbf{L} + \mathbf{L}^T\,] \;\; ; \;\; \mathbf{L} = \dot{\mathbf{F}}\mathbf{F}^{-1} \qquad (2a,b)$$

where the superscripts "-1" and "T" denote the inverse and the transpose. In the special case where $\mathbf{L}$ is the skew-symmetric tensor whose axial vector is the angular velocity, equation (1b) represents rigid body motion (Casey, 1983, and Fahrenthold and Wargo, 1991a and b).

c. Kinetic energy

The homogeneous deformation kinematics of equations (1) allows the kinetic co-energy (T^*) of a single element of fixed mass "m" and variable volume "V" to be expressed in the form

$$T^* = (1/2) \int_V \rho\,\mathbf{v}\cdot\mathbf{v} \;\; dV \qquad (3a)$$

$$= (1/2)\,\{ \int_V \rho\,\dot{\mathbf{c}}\cdot\dot{\mathbf{c}} \; dV + \int_V \rho\,\dot{\mathbf{F}}(\mathbf{r}-\mathbf{r}_c)\cdot\dot{\mathbf{F}}(\mathbf{r}-\mathbf{r}_c) \; dV + 2\dot{\mathbf{c}}\cdot\dot{\mathbf{F}} \int_V \rho(\mathbf{r}-\mathbf{r}_c) \; dV \qquad (3b)$$

$$= (1/2)\,[\, m\dot{\mathbf{c}}\cdot\dot{\mathbf{c}} + \mathrm{tr}(\dot{\mathbf{F}}^T\dot{\mathbf{F}}\,\mathbf{J})\,] \qquad (3c)$$

where ρ is the density, "tr" is the trace operator, $\mathbf{J}$ is an inertia tensor,

$$\mathbf{J} = \int_V \rho_o(\mathbf{r}-\mathbf{r}_c)\otimes(\mathbf{r}-\mathbf{r}_c)\ dV_o \tag{3d}$$

and conservation of mass requires

$$dm = \rho\, dV = \rho_o\, dV_o \tag{3e}$$

with ρ_o the density in the reference configuration. Note that the third term in equation (3b) is zero by definition of the center of mass. Since **J** is a constant tensor defined in the reference configuration, the element momenta **p** and **H** are defined by

$$\mathbf{p} = \partial T^*/\partial \dot{\mathbf{c}} = m\dot{\mathbf{c}}\ ;\ \mathbf{H} = \partial T^*/\partial \dot{\mathbf{F}} = \dot{\mathbf{F}}\mathbf{J} \tag{4a,b}$$

Since the kinetic energy (T) of the finite element is

$$T = \mathbf{p}\cdot\dot{\mathbf{c}} + \mathrm{tr}(\mathbf{H}^T\dot{\mathbf{F}}) - T^* \tag{5a}$$

it follows that

$$T = (1/2)\,[\, m^{-1}\,\mathbf{p}\cdot\mathbf{p} + \mathrm{tr}\{(\mathbf{H}\mathbf{J}^{-1})^T\mathbf{H}\}\,] \tag{5b}$$

The preceding results demonstrate that kinetic energy storage in a single finite element may be modeled using the bond graph multiports shown in Figure 1. The kinetic energy function may be represented in the most familiar form by the introduction of a fourth order inertia tensor $\underset{\approx}{\mathbf{G}}$, defined by

$$\underset{\approx}{\mathbf{G}} = \frac{\partial^2 T^*}{\partial\dot{\mathbf{F}}\partial\dot{\mathbf{F}}} \tag{6a}$$

so that

$$T^* = (1/2)\,[\, m\dot{\mathbf{c}}\cdot\dot{\mathbf{c}} + \dot{\mathbf{F}}:\underset{\approx}{\mathbf{G}}:\dot{\mathbf{F}}\,]\ ;\ T = (1/2)\,[\, m^{-1}\,\mathbf{p}\cdot\mathbf{p} + \mathbf{H}:\underset{\approx}{\mathbf{G}}^{-1}:\mathbf{H} \tag{6b,c}$$

Now the constitutive relation (4b) takes the form

$$\mathbf{H} = \underset{\approx}{\mathbf{G}}\,\dot{\mathbf{F}} \tag{6d}$$

d. Internal energy

The preceding discussion of inertia effects must be augmented by an RC network description of internal energy storage and energy dissipation in the material. Unlike the inertia multiports, the capacitance and resistance multiports required to model a particular material must be formulated for each material type. To illustrate this procedure, this section considers the large strain deformation of an elastic-viscoplastic material (Haupt,

1985). This case emphasizes the applicability of the bond graph modeling methodology described here to very complex engineering materials (Fahrenthold and Wu, 1988), and hence to system dynamics and impact dynamics problems of a very general nature. Note that the conventional assumptions of infinitesimal strain and linear stress-strain behavior, included in the vast majority of mechanical vibrations models, are not adopted here. Since the large strains and the relatively complex material response considered here often calls for special purpose finite element code development work, the example material selected demonstrates the simplicity of the proposed modeling methodology, as compared to conventional displacement-based finite element analysis.

Before formulating a stored energy function for an elastic-plastic material, it is appropriate to first define the system kinematics. A very general description of the large strain kinematics of elastic-plastic deformation is provided by a multiplicative decomposition of the deformation gradient tensor **F** in the form (Haupt, 1985)

$$\mathbf{F} = \mathbf{F}^e \mathbf{F}^p \;\; ; \;\; \dot{\mathbf{F}} = \dot{\mathbf{F}}^e \mathbf{F}^p + \mathbf{F}^e \dot{\mathbf{F}}^p \qquad (7a,b)$$

where $\mathbf{F}^e$ describes the elastic deformation of the solid material from the unloaded plastically deformed configuration and $\mathbf{F}^p$ relates the unloaded plastically deformed configuration of the material to the original undeformed reference configuration. Hence the rate of change of **F** may be decomposed as in equation (7b) into a first term governing the rate of elastic energy storage and a second term governing the rate of plastic energy dissipation. In this case, the presence of both complex kinematics and energy dissipation effects highlights the value of bond graphs in nonlinear system modeling.

Since most elastic-plastic modeling work begins with a Helmholtz free energy function and then derives the internal energy, that procedure is followed here. The elastic-plastic kinematics just described are associated with a Helmholtz free energy density function of the form (Dashner, 1986)

$$\psi = \psi(\mathbf{F}^e, \mathbf{F}^p, \theta) \tag{7c}$$

Here ψ is assumed to take the conventional functional form (Bowen, 1989)

$$\psi = (1/\rho_\alpha)\ \{\ (\lambda/2)\ \mathrm{tr}(\mathbf{E}^e)^2 + \mu\ \mathrm{tr}(\mathbf{E}^{e2}) - \beta(\theta-\theta_o)\ \mathrm{tr}(\mathbf{E}^e) - [(c\rho_\alpha)/(2\theta_o)](\theta-\theta_o)^2\ \} \tag{7d}$$

where λ and μ are Lame constants for an isotropic solid, c is the specific heat, θ_o is a reference temperature, ρ_α is the density in the unloaded plastically deformed configuration, and

$$\mathbf{E}^e = (1/2)\ [\ \mathbf{F}^{eT}\mathbf{F}^e - \mathbf{I}\ ]\ ;\ \beta = k\ (3\lambda+2\mu)\ ;\ \rho_o/\rho_\alpha = \det(\mathbf{F}^p) \tag{7e,f,g}$$

with "k" the thermal expansion coefficient and "det" the determinant operator. For an element of mass "m", this corresponds to a (conserved) internal energy function

$$U = m\psi + \theta S \tag{8a}$$

where S is the total entropy, defined by the thermodynamic identity

$$S = -\frac{\partial(m\psi)}{\partial\theta} = (m/\rho_\alpha)\ \{\ c\rho_\alpha\ [\ (\theta/\theta_o) - 1] + \beta\ \mathrm{tr}(\mathbf{E}^e)\ \} \tag{8b}$$

Equations (7) and (8) may be combined to yield

$$U = (m/\rho_o)\ \det(\mathbf{F}^p)\ \{\ \mu\ \mathrm{tr}(\mathbf{E}^{e2}) + (\lambda/2)\ \mathrm{tr}(\mathbf{E}^e)^2 + \beta\theta_o\ \mathrm{tr}(\mathbf{E}^e)\ \} +$$

$$[(cm\theta_o)/2]\ \{\ [\ 1 + S/(mc) - (\beta/(c\rho_o))\ \det(\mathbf{F}^p)\ \mathrm{tr}(\mathbf{E}^e)\ ]^2 - 1\ \} \tag{8c}$$

The functional form for the internal energy

$$U = U(\mathbf{F}^e, \mathbf{F}^p, S) \tag{9a}$$

leads to the multiport capacitor shown in Figure 2, where in this case

$$\mathbf{K}^e = \frac{\partial U}{\partial \mathbf{F}^e}\Big|_{\mathbf{F}^p,S} = (m/\rho_\alpha)\ \mathbf{F}^e\{\ [\lambda+((\beta^2\theta_o)/(c\rho_\alpha))]\ \mathrm{tr}(\mathbf{E}^e)\ \mathbf{I} +$$

$$2\mu\mathbf{E}^e - [(\beta\theta_o S)/(mc)]\ \mathbf{I}\ \} \tag{9b}$$

$$\mathbf{K}^p = \frac{\partial U}{\partial \mathbf{F}^p}\Big|_{\mathbf{F}^e,S} = (m/\rho_\alpha)\ \{\ [(\lambda/2)+((\beta^2\theta_o)/(c\rho_\alpha))]\ \mathrm{tr}(\mathbf{E}^e)^2 +$$

$$\mu\ \mathrm{tr}(\mathbf{E}^{e2}) - [(\beta\theta_o S)/(mc)]\ \mathrm{tr}(\mathbf{E}^e)\ \}\ \mathbf{F}^{p-T} \tag{9c}$$

$$\theta = \frac{\partial U}{\partial S}\Big|_{\mathbf{F}^e,\mathbf{F}^p} = \theta_o\ \{\ 1 + S/(mc) - [\beta/(c\rho_\alpha)]\ \mathrm{tr}(\mathbf{E}^e)\ \} \tag{9d}$$

e. Plastic deformation

The material description is completed by defining the plastic constitutive relations. For simplicity, the rate form

$$\mathbf{K}^d = \eta\,(m/\rho_\alpha)\,\mathbf{L}^p \;;\; \mathbf{L}^p = \dot{\mathbf{F}}^p\mathbf{F}^{p-1} \tag{10a,b}$$

is adopted here, with η a viscosity coefficient. The second order tensor $\mathbf{K}^d$ (dimensionally an extensive chemical potential) is the effort power conjugate to $\mathbf{L}^p$.

A model for the elastic-viscoplastic material just defined is shown in Figure 2. Note that a transformer with fourth order tensor modulus $\underset{\approx}{\mathbf{M}}$, defined by components

$$M_{ijrs} = \delta_{ir}F^{p-1}_{sj} \tag{11}$$

is introduced in order to conform to the fundamental kinematic relation (10b).

f. Bond graph model

The bond graph structure of Figure 2 may be directly augmented with inertia multiports representing the kinetic energy contributions associated with **p** and **H**. Finally if the appropriate RC network model representing elastic-viscoplastic materials is introduced, the result is the complete element-level bond graph shown in Figure 3, representing the "ith" element of the system. Assuming adiabatic deformation, thermal energy is stored, as indicated by the thermomechanical coupling shown in Figure 3. Note that in Figure 3 transformers with moduli

$$M^{e(i)}_{ijkl} = \delta_{jl}F^{e(i)-1}_{ik} \;;\; M^{p(i)}_{ijkl} = \delta_{ik}F^{p(i)-1}_{lj} \tag{12a,b}$$

are introduced in accordance with the kinematics of equations (7a and b).

g. State equation derivation

The causally augmented bond graph of Figure 3 and the constitutive relations previously defined yield (Rosenberg and Karnopp, 1983) state equations for the "ith" element of the form:

$$\dot{\mathbf{p}}^{(i)} = \mathbf{0} \tag{13a}$$

$$\dot{\mathbf{H}}^{(i)} = -\,\underset{\approx}{\mathbf{M}}^{p(i)T}\,\mathbf{K}^{e(i)}(\mathbf{F}^{e(i)}, \mathbf{F}^{p(i)}, S^{(i)}) \tag{13b}$$

$$\dot{\mathbf{F}}^{e(i)} = \underset{\approx}{\mathbf{M}}^{p(i)}\,\underset{\approx}{\mathbf{G}}^{(i)-1}\,\mathbf{H}^{(i)} - \{\rho_\alpha^{(i)}/(\eta m^{(i)})\}\,\underset{\approx}{\mathbf{M}}^{p(i)}\,\underset{\approx}{\mathbf{M}}^{e(i)-1}\,\underset{\approx}{\mathbf{M}}^{(i)-1}\mathbf{K}^{d(i)} \tag{13c}$$

$$\dot{\mathbf{F}}^{p(i)} = \{\rho_\alpha^{(i)}/(\eta m^{(i)})\}\,\underset{\approx}{\mathbf{M}}^{(i)-1}\,\mathbf{K}^{d(i)} \tag{13d}$$

$$\dot{S}^{(i)} = [1/\theta^{(i)}(\mathbf{F}^{e(i)}, \mathbf{F}^{p(i)}, S^{(i)})]\,\{\rho_\alpha^{(i)}/(\eta m^{(i)})\}\,\mathrm{tr}\{\,\mathbf{K}^{d(i)T}\mathbf{K}^{d(i)}\,\} \tag{13e}$$

where

$$\mathbf{K}^{d(i)} = \underset{\approx}{\mathbf{M}}^{(i)-T}\,[\,\underset{\approx}{\mathbf{M}}^{e(i)-T}\,\underset{\approx}{\mathbf{M}}^{p(i)T}\,\mathbf{K}^{e(i)}(\mathbf{F}^{e(i)}, \mathbf{F}^{p(i)}, S^{(i)}) - \mathbf{K}^{p(i)}(\mathbf{F}^{e(i)}, \mathbf{F}^{p(i)}, S^{(i)})] \tag{13f}$$

and the functions $\mathbf{K}^{e(i)}$, $\mathbf{K}^{p(i)}$, and $\theta^{(i)}$ are defined by equations (9). Note that

$$M^{e(i)-1}_{ijkl} = \delta_{jl}F^{e(i)}_{ik} \;;\; M^{p(i)}_{ijkl} = \delta_{ik}F^{p(i)-1}_{lj} \;;\; M^{(i)-1}_{ijkl} = \delta_{ik}F^{p(i)}_{lj} \tag{14a,b,c}$$

Equations (13) are nonlinear equations in the state variables: $\mathbf{p}^{(i)}$, $\mathbf{H}^{(i)}$, $\mathbf{F}^{e(i)}$, $\mathbf{F}^{p(i)}$, and $S^{(i)}$.

h. Conclusion

The outlined modeling methodology may be implemented numerically, for a variety of internal energy functions and plasticity models, for use in engineering analysis and design. The source code listing at Appendix G represents an isothermal, variable compressibility implementation using the plasticity theory of Green and Naghdi (1971).

4. Comparison to Experiments

a. Introduction

Three dimensional simulation of hypervelocity impacts on space structures places extreme demands on even supercomputer resources. To reduce the computer time and memory requirements of oblique impact simulations, a three-dimensional, deformable particle dynamics model of the type just described has been coded and linked to an Eulerian hydrocode and a Lagrangian structural code, and applied in the simulation of oblique hypervelocity impacts on Whipple shield protected structures. Comparison of the results to experimental data shows good agreement at a computer time and memory cost much less than that associated with conventional hydrocode calculations.

This section describes evaluation of the preceding modeling approach using data from Whipple shield impact experiments conducted at NASA Johnson Space Center, including CPU time requirements for the simulation of representative oblique impact problems.

b. Methodology

The deformable particle dynamics model of debris cloud evolution is referred to here by the title DC3D. This numerical model is used in combination with the Eulerian hydrocode CTH (McGlaun et al., 1990) and the structural finite element code DYNA3D (Goudreau and Hallquist, 1982) as follows. An Eulerian simulation is first employed to model impact on the shield. Then post-processing of the velocity, mass density, and void space distribution data from the Eulerian simulation is used to establish the initial state of the elastic-plastic model of the debris cloud. Numerical integration of the particle dynamics model DC3D, using established system dynamics modeling techniques (Rosenberg and Karnopp, 1983) provides a thermodynamically rigorous yet computationally efficient basis for predictions of debris cloud evolution over the relatively large spaces which normally separate shields from the space structure which they protect. Output from the debris cloud evolution model then provides a basis for the simulation of impact on the wall plate, using DYNA3D. Input data for the wall impact simulation is generated automatically by DC3D, based on the final state of the debris cloud propagation model. By using a momentum deposition representation of the debris cloud loading on the wall plate (calculated from the DC3D results), meshing of each debris particle in DYNA3D can be avoided, yielding additional reductions in computer time and memory requirements.

c. Oblique impact example

The analysis procedure just discussed may be illustrated by considering an oblique impact simulation for a Whipple shield configuration. Specifically, consider the oblique (23 degree) impact of a 15/64 inch diameter aluminum (2017-T4) sphere, at 7.1 kilometers per second, on a 0.063 inch thick aluminum (7075-T6) shield, protecting a

0.125 inch thick aluminum (7075-T6) wall plate. The shield to wall plate spacing is 4.0 inches. Figure 4 shows the CTH simulation results for impact on the shield, at four microseconds after impact. (Note that due to symmetry, all of the figures discussed here depict only one-half of the physical system modeled.) The code DC3D was then employed to: (1) post-processes the CTH results to formulate a debris cloud model, (2) propagate the debris to the wall plate, and (3) generate a DYNA3D input file for use in modeling impact of the debris cloud on the wall plate. DC3D uses a variable step method for numerical integration of the first order state equations describing the elastic-plastic debris cloud dynamics, and commercial plotting routines for graphical representation of the simulation results. Figure 5 shows the predicted impact momentum distribution on the wall plate. Finally a DYNA3D simulation of the wall plate impact was conducted, using a momentum deposition approach to represent the effects of the debris. Figures 6 and 7 show front and rear views of the wall plate impact simulation, including extensive spallation observed in the experiment.

By making appropriate use of Eulerian and Lagrangian codes for those parts of the simulation where they are both accurate and computationally tractable, this approach does not suffer from the limiting assumptions of many previous attempts to model shield impact problems (e.g. Swift et al., 1983). In addition, that portion of the total impact simulation described by the debris cloud model incorporates arbitrarily nonuniform, three-dimensional velocity, density, and void space distributions not admitted by many simplified debris models published to date (Grady and Passman, 1990). In summary, the approach used here makes use of the known strengths of available codes while providing an essential improvement in computational efficiency for oblique impact simulation. Such an improvement is essential before computer codes can provide a practical analytical design tool.

The preceding example demonstrates the feasibility of the modeling approach. Its computational efficiency is such that computer resource requirements are relatively

modest, considering the three dimensional nature of the simulations. Typical CPU time requirements are listed in Table 1.

d. Ballistic limit calculations

Finally it is worthwhile to compare some hypervelocity impact simulation results with ballistic limit calculations based on published experimental research. Figure 8 depicts a typical Ballistic limit curve, and indicates the velocities and impact particle diameters used in the simulations. Figures 9 through 20 show results of the simulation of oblique Whipple shield impacts at 7 and 10 kilometers per second, using the debris propagation code and the general modeling methodology developed under this project. The cases represented are listed in Table 2.

Note that the simulations were conducted for particle diameters above and below the expected ballistic limits for the two velocities. The simulations are consistent with the ballistic limit curve data at 7 km/sec. At 10 km/sec, the simulations suggest slightly less wall damage than might be expected from the ballistic limit curve. Computer time requirements are very modest, considering the complexity of the problems.

5. Conclusion

The outlined research has addressed fundamental problems relevant to the development of space vehicles and structures for a variety of missions. It makes use of state of the art supercomputer resources while applying the newest numerical modeling techniques, designed to make oblique impact simulations computationally tractable. It included numerical implementation of a nonlinear, elastic-plastic debris cloud dynamics model important to accurate shield impact calculations. The resulting simulation capability can provide an important adjunct to experimental work on space shield design for a variety of future missions. The outlined work has been coordinated directly with NASA JSC experimental research efforts, and the final source code has been provided to NASA. It is therefore suggested that the proposed research has addressed important objectives of the NASA Regional Universities Grant Program.

REFERENCES

Anderson, C.E., Jr., ed., 1986, *Hypervelocity Impact: Proceedings of the 1986 Symposium*, Pergammon Press, Oxford.

Bowen, R.M., 1989, Introduction to Continuum Mechanics for Engineers, Plenum Press, New York.

Budge, K.G., and Peery, J.S., 1993, "RHALE: a MMALE shock physics code written in C++," *International Journal of Impact Engineering*, Vol. 14, pp. 107-120.

Casey, J., 1983, "A Treatment of Rigid Body Dynamics," *Journal of Applied Mechanics*, Vol. 50, pp. 905-907 (see also errata listed in Vol. 51, p. 227).

Christiansen, E.L., 1990, "Advanced Meteoroid and Debris Shielding Concepts," AIAA 90-1336, presented at the AIAA/NASA/DOD Orbital Debris Conference, Baltimore, April 16-19.

Cour-Palais, B.G., and Crews, J.L., 1990, "A Multi-Shock Concept for Spacecraft Shielding," *International Journal of Impact Engineering*, Vol. 10, pp. 135-146.

Dashner, P.A., 1986, "Invariance Considerations in Large Strain Elasto-plasticity," *Journal of Applied Mechanics*, Vol. 53, pp. 55-60.

Fahrenthold, E.P., 1993, "A Lagrangian Model for Debris Cloud Dynamics Simulation" *International Journal of Impact Engineering*, Vol. 14, pp. 229-240.

Fahrenthold, E.P., and Wargo, J.D., 1991a, "Vector and Tensor Based Bond Graphs for Physical Systems Modeling," *Journal of the Franklin Institute*, Vol. 328, No. 5/6, pp. 833-853.

Fahrenthold, E.P., and Wargo, J.D., 1991b, "Vector Bond Graph Analysis of Mechanical Systems," *Journal of Dynamic Systems, Measurement and Control*, Vol. 113, No. 3, pp. 344-353.

Fahrenthold, E.P., and Wargo, J.D., 1994, "Lagrangian Bond Graphs for Solid Continuum Dynamics Modeling," *Journal of Dynamic Systems, Measurement, and Control*, in press.

Fahrenthold, E.P., and Wu, A., 1988, "Bond Graph Modeling of Continuous Solids in Finite Strain Elastic-Plastic Deformation," *Journal of Dynamic Systems, Measurement, and Control*, Vol. 110, pp. 284-287.

Fritts, M.J., Crowley, W.P., and Trease, H.E., eds., 1985, The Free-Lagrange Method, Springer-Verlag, New York.

Goudreau, G.L., and Hallquist, J.O., 1982, "Recent Developments in Large-Scale Finite Element Hydrocode Technology," *Computer Methods in Applied Mechanics and Engineering*, Vol. 33, pp. 725-757.

Grady, D.E., and Kipp, M.E., 1987, "Dynamic Rock Fragmentation," in *Fracture Mechanics of Rock*, ed. B.K. Atkinson, Academic Press, San Diego, pp. 429-475.

Grady, D.E., and Passman, S.L., 1990, "Stability and Fragmentation of Ejecta in Hypervelocity Impact," *International Journal of Impact Engineering*, Vol. 10, pp. 197-212.

Green, A.E., and Naghdi, P.M., 1971, *Int. J. of Eng. Science*, Vol. 9, pp. 1219.

Haupt, P., 1985, "On the Concept of an Intermediate Configuration and Its Application to a Representation of Viscoelastic-Plastic Material Behavior," *International Journal of Plasticity*, Vol. 1, pp. 303-316.

Malvern, L.E., 1969, Introduction to the Mechanics of a Continuous Medium, Prentice-Hall, Englewood Cliffs, New Jersey.

McGlaun, J.M., Thompson, S.L., and Elrick, M.G., 1990, "CTH: A Three Dimensional Shock Wave Physics Code," *International Journal of Impact Engineering*, Vol. 10, pp. 351-360.

Melosh, H.J., Ryan, E.V., and Asphaug, E., 1992, "Dynamic Fragmentation in Impacts," *Journal of Geophysical Research*, Vol. 97, No. E9, pp. 14,735-14,759.

Monaghan, J.J., 1988, "An Introduction to SPH," *Computer Physics Communications*, Vol. 48, pp. 89-96.

Rosenberg, R.C., and Karnopp, D.C., 1983, Introduction to Physical System Dynamics, McGraw-Hill, New York.

Schneider, E., and Stilp, A., 1987, "Micrometeorite Impact Simulation at EMI - A Review," International Journal of Impact Engineering, Vol. 5, pp. 561-568.

Stellingwerf, R.F., and Wingate, C.A., 1993, "Impact Modeling with Smooth Particle Hydrodynamics," *International Journal of Impact Engineering*, Vol. 14, pp. 707-718.

Swift, H.F, Bamford, R., and Chem, R., 1983, "Designing Space Vehicle Shields for Meteoroid Protection: A New Analysis," *Advances in Space Research*, Vol. 2, pp. 219-234.

Tower, M.M., Jackson, G.L., Farris, L.K., and Haight, C.H., 1987, "Hypervelocity Impact Testing Using an Electromagnetic Railgun Launcher," International Journal of Impact Engineering, Vol. 5, pp. 635-644.

Trease, H.E., 1988, "Three-Dimensional Free-Lagrange Hydrodynamics," *Computer Physics Communications*, Vol. 48, pp. 39-50.

Trease, H.E., Fritts, M.J., and Crowley, W.P., eds., 1990, Advances in the Free-Lagrange Method, Springer-Verlag, New York.

Yew, C.H., and Kendrick, R.B., 1986, "A Study of Damage in Composite Panels Produced by Hypervelocity Impact," *International Journal of Impact Engineering*, Vol. 5, No. 1-4, pp. 729-738.

Yew, C.H., and Taylor, P.A., 1992, "A Simple Theory of Dynamic Fragmentation," Sandia National Laboratories, Albuquerque, New Mexico.

Table 1. Example Oblique Impact Simulations

shield to wall spacing:	4.0-8.0 inches
shield thickness:	0.063-0.071 inches
wall thickness:	0.125-0.160 inches
impact velocity:	7.1-7.6 km/sec
impact obliquity:	23.0-58.4 degrees
impact mass:	0.202-0.376 grams
material:	aluminum

Simulation type	Computer system	Code	Memory	CPU time
shield impact	Cray Y-MP/864	CTH	< 64 MB	2.0-6.0 hrs
debris evolution	IBM RS/6000*	DC3D	< 16 MB	0.5-3.0 days
wall impact	Cray Y-MP/864	DYNA3D	< 64 MB	0.5-1.0 hrs

*The CPU time range given is for a Model 320, a very low performance system.

Table 2. Ballistic Limit Simulations

bumper thickness	=	0.127 cm
wall thickness	=	0.3175 cm
impact obliquity	=	25 degrees
material	=	Al 6061-T6

(also see the ballistic limit plot in Figure 8)

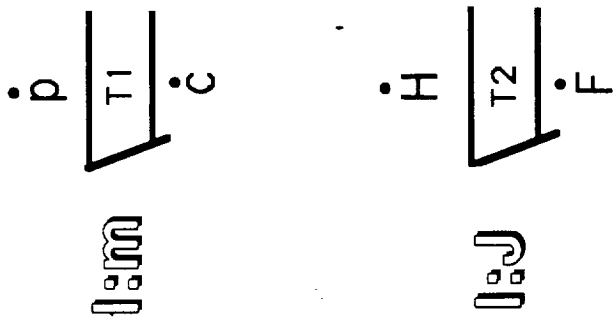
Simulation #	Impact velocity	Particle diameter	Figures
7a	7 km/sec	0.575 cm	9 through 11
7b	7 km/sec	0.475 cm	12 through 14
10a	10 km/sec	0.530 cm	15 through 17
10b	10 km/sec	0.430 cm	18 through 20

CPU time requirements

bumper impact simulations (CTH):	1.80-2.11 CPU hrs, Cray YMP
debris transport calculations (DC3D):	3.34-5.67 CPU hrs, IBM RS/6000
wall impact calculations (DYNA3D):	0.27-0.28 CPU hrs, Cray YMP

List of Figures

Figure 1



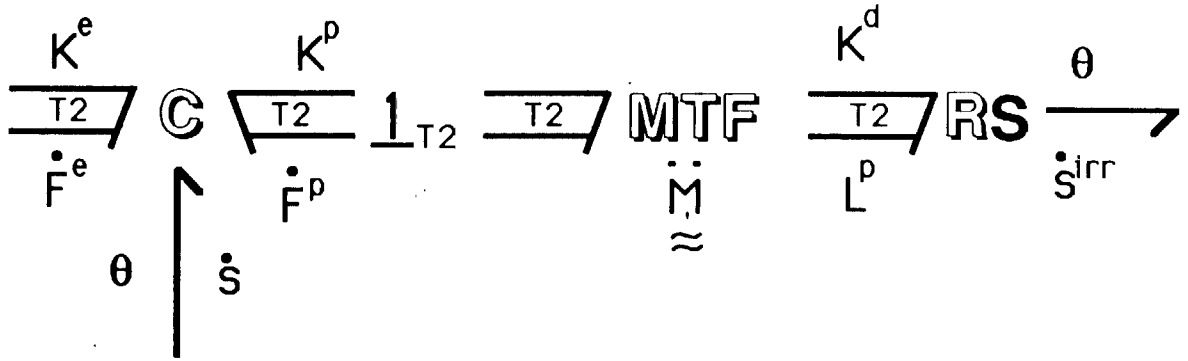
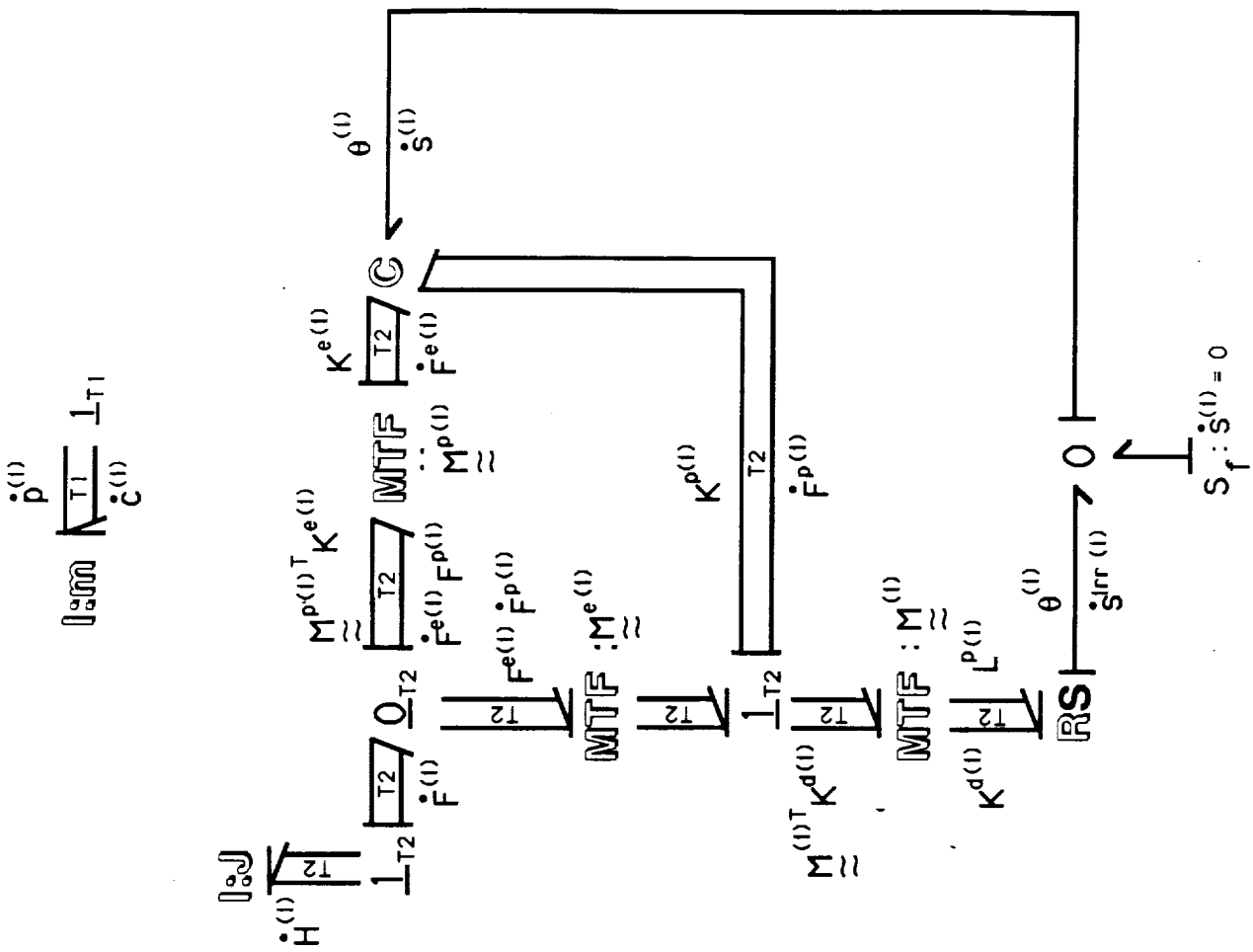
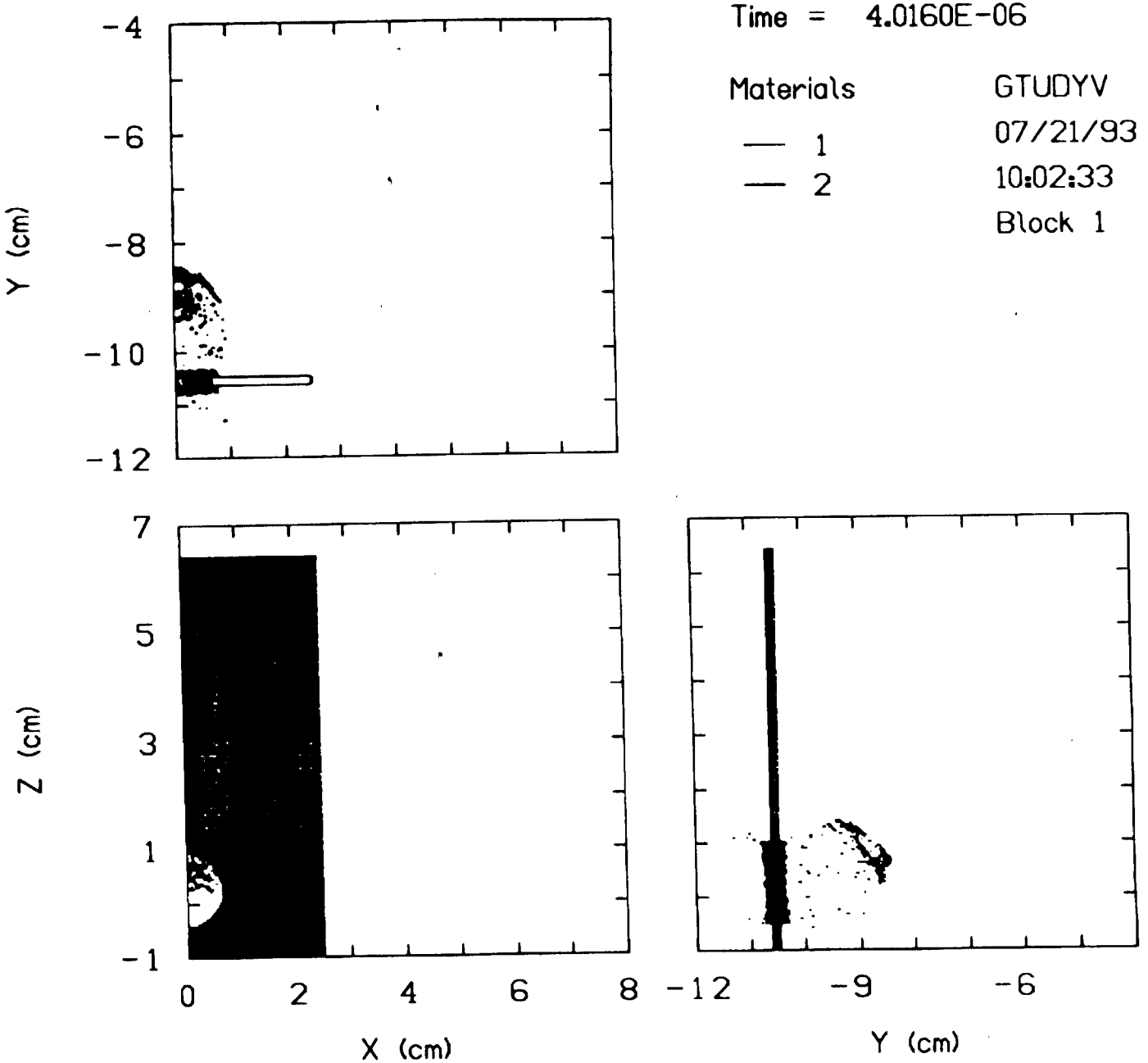


Figure 2

Figure 3

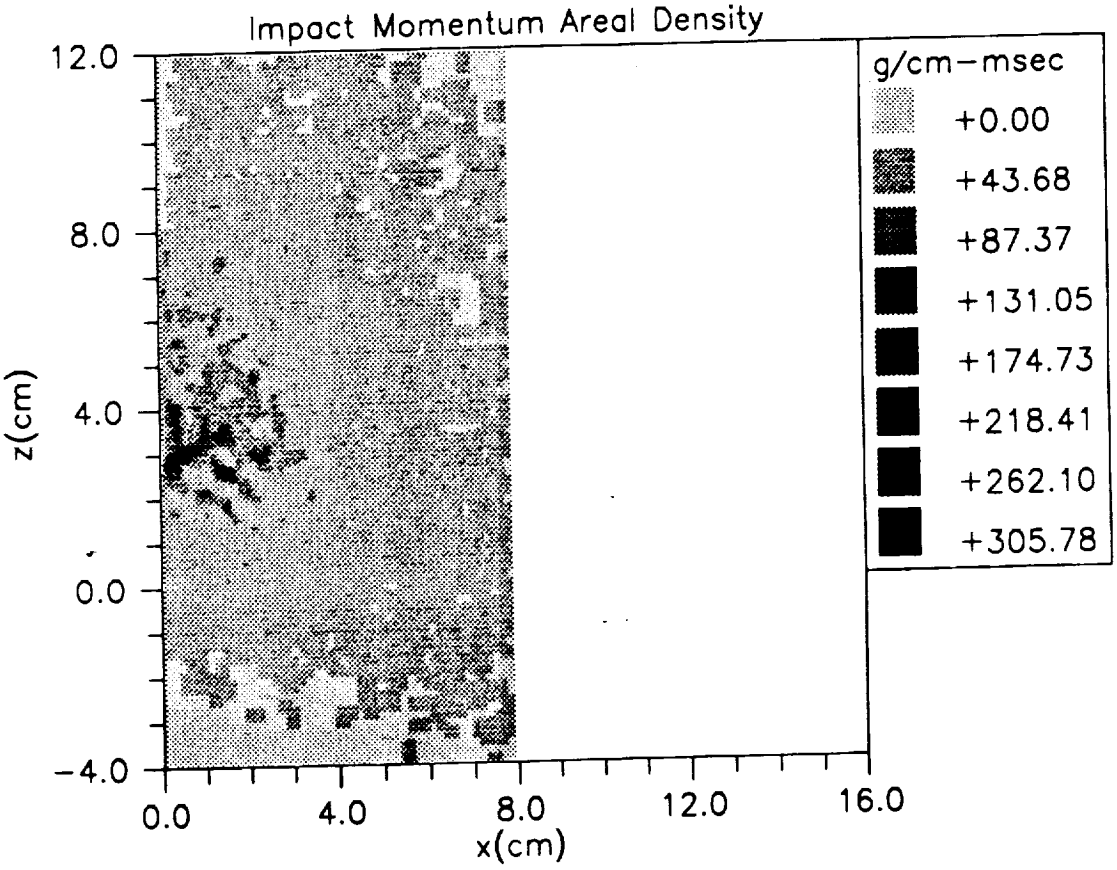




Oblique whipple shield impact (B324)

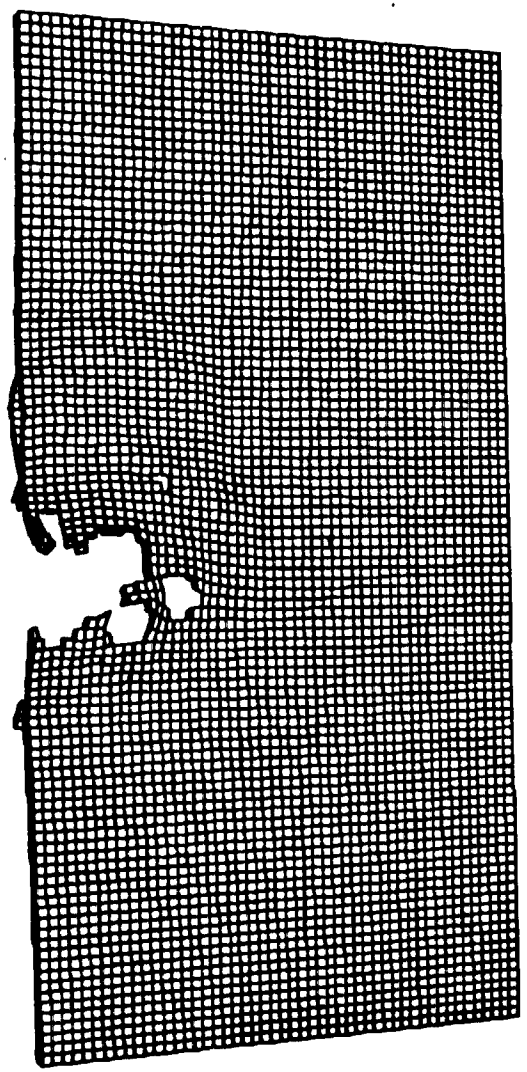
Figure 4

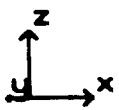
Figure 5


Impact Momentum Areal Density
g/cm−msec
+0.00
+43.68
+87.37
+131.05
+174.73
+218.41
+262.10
+305.78
12.0
8.0
4.0
0.0
−4.0
z(cm)
0.0
4.0
8.0
12.0
16.0
x(cm)

Debris Cloud Wall Impact Model
time = 0.49974E+02

B324: front surface



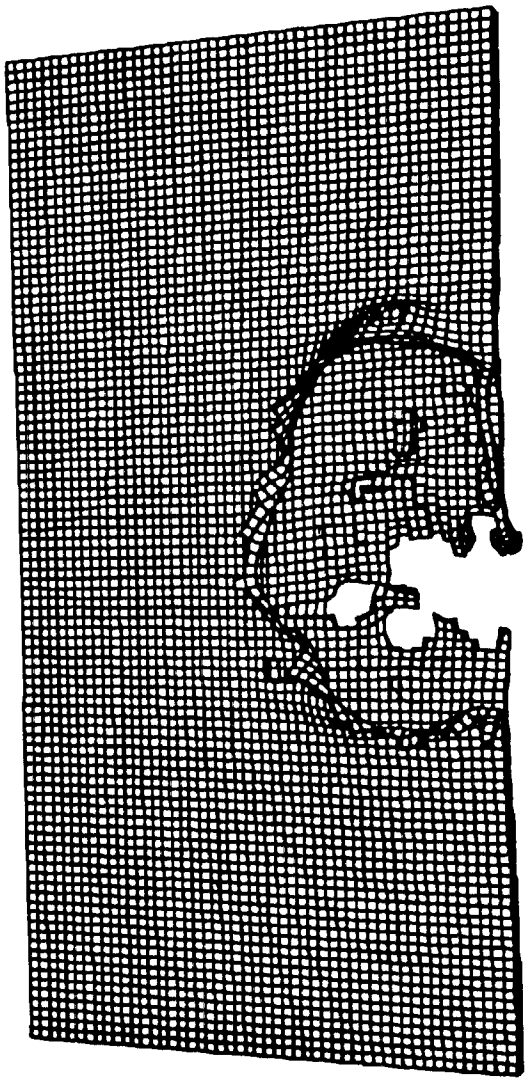


disp. scale factor = 0.100E+01 (default)

Figure 6

Debris Cloud Wall Impact Model
time = 0.49974E+02

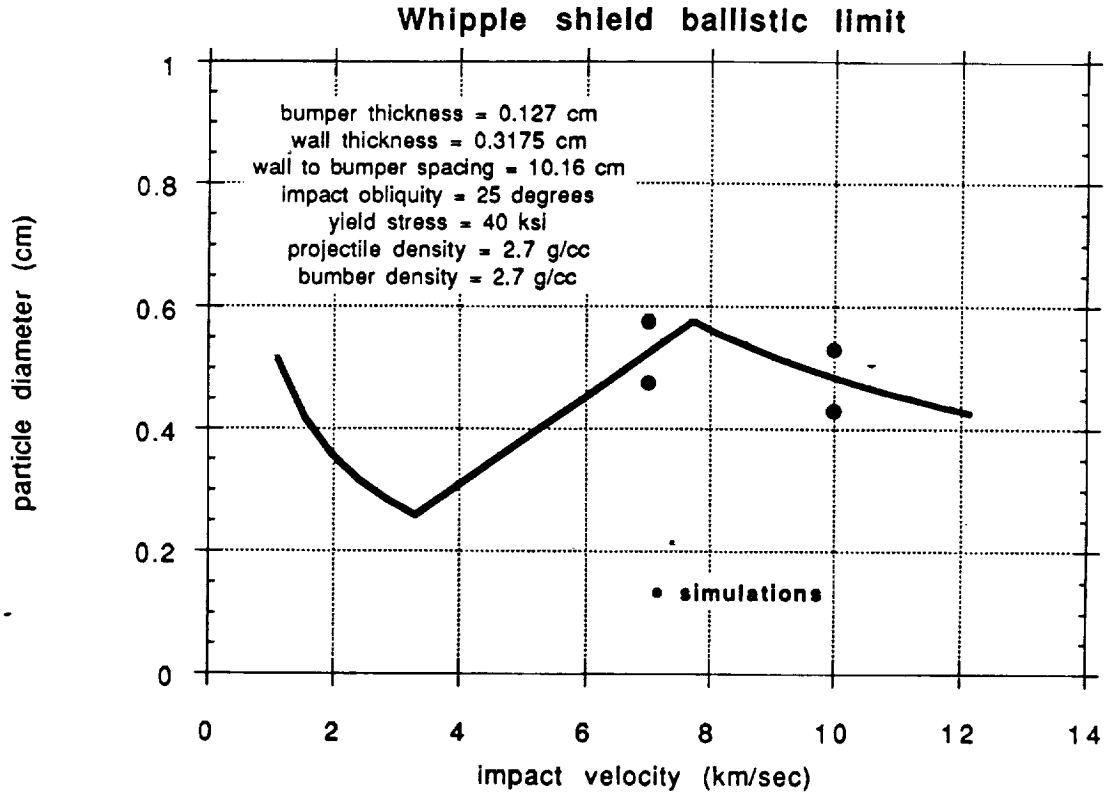
B324: rear surface

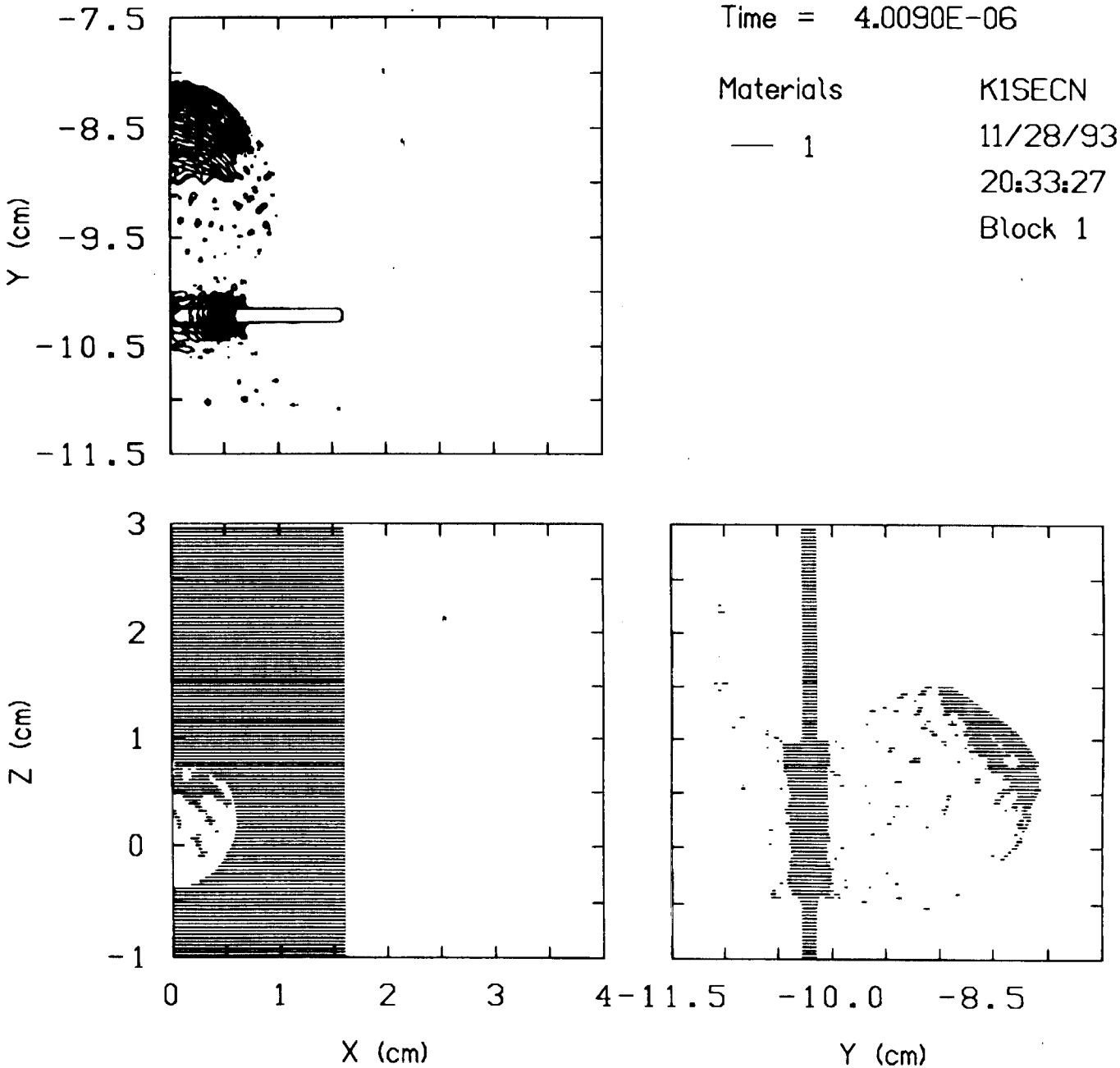


disp. scale factor = 0.100E+01 (default)

Figure 7

Figure 8

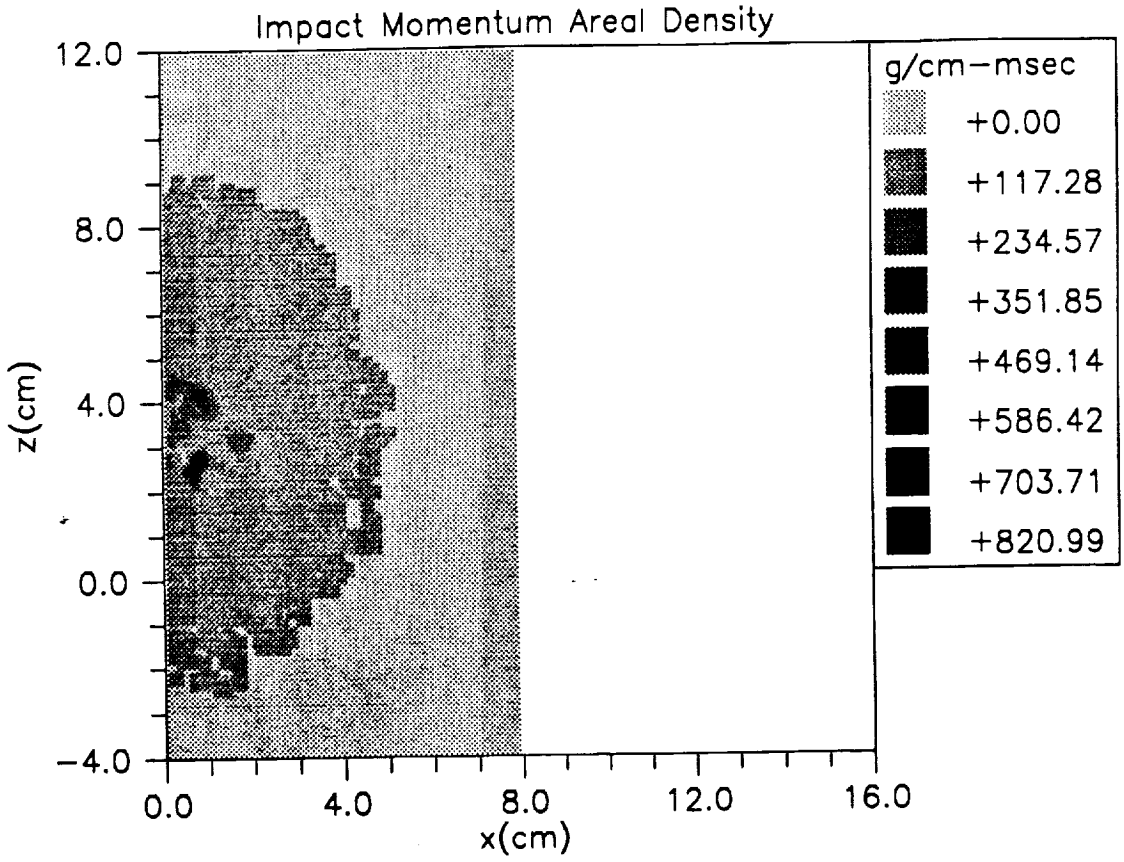

Whipple shield ballistic limit
bumper thickness = 0.127 cm
wall thickness = 0.3175 cm
wall to bumper spacing = 10.16 cm
impact obliquity = 25 degrees
yield stress = 40 ksi
projectile density = 2.7 g/cc
bumber density = 2.7 g/cc
• simulations
particle diameter (cm)
impact velocity (km/sec)
0
0.2
0.4
0.6
0.8
1
0
2
4
6
8
10
12
14



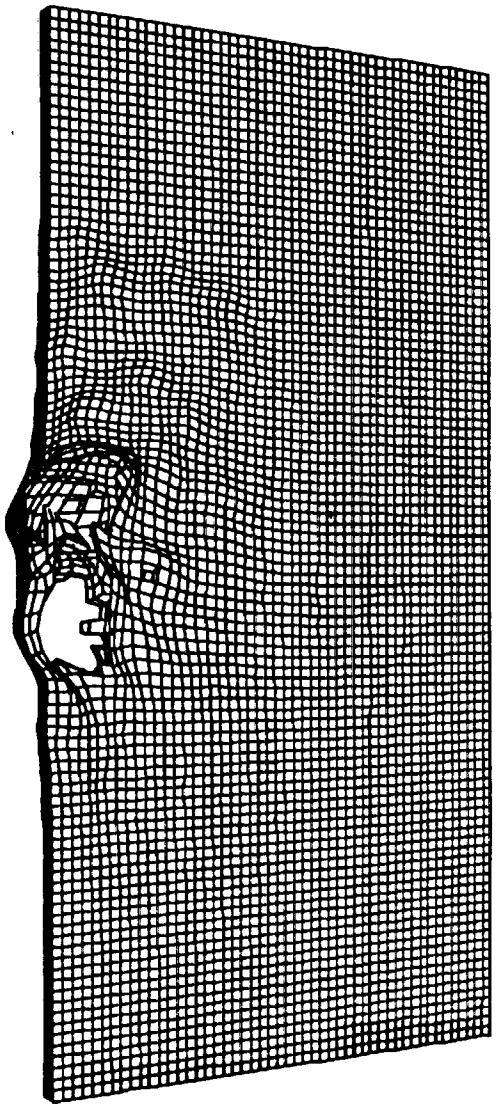
Oblique whipple shield impact (7a)

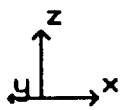
Figure 9

Figure 10


Impact Momentum Areal Density
g/cm−msec
+0.00
+117.28
+234.57
+351.85
+469.14
+586.42
+703.71
+820.99
12.0
8.0
4.0
0.0
−4.0
z(cm)
0.0
4.0
8.0
12.0
16.0
x(cm)

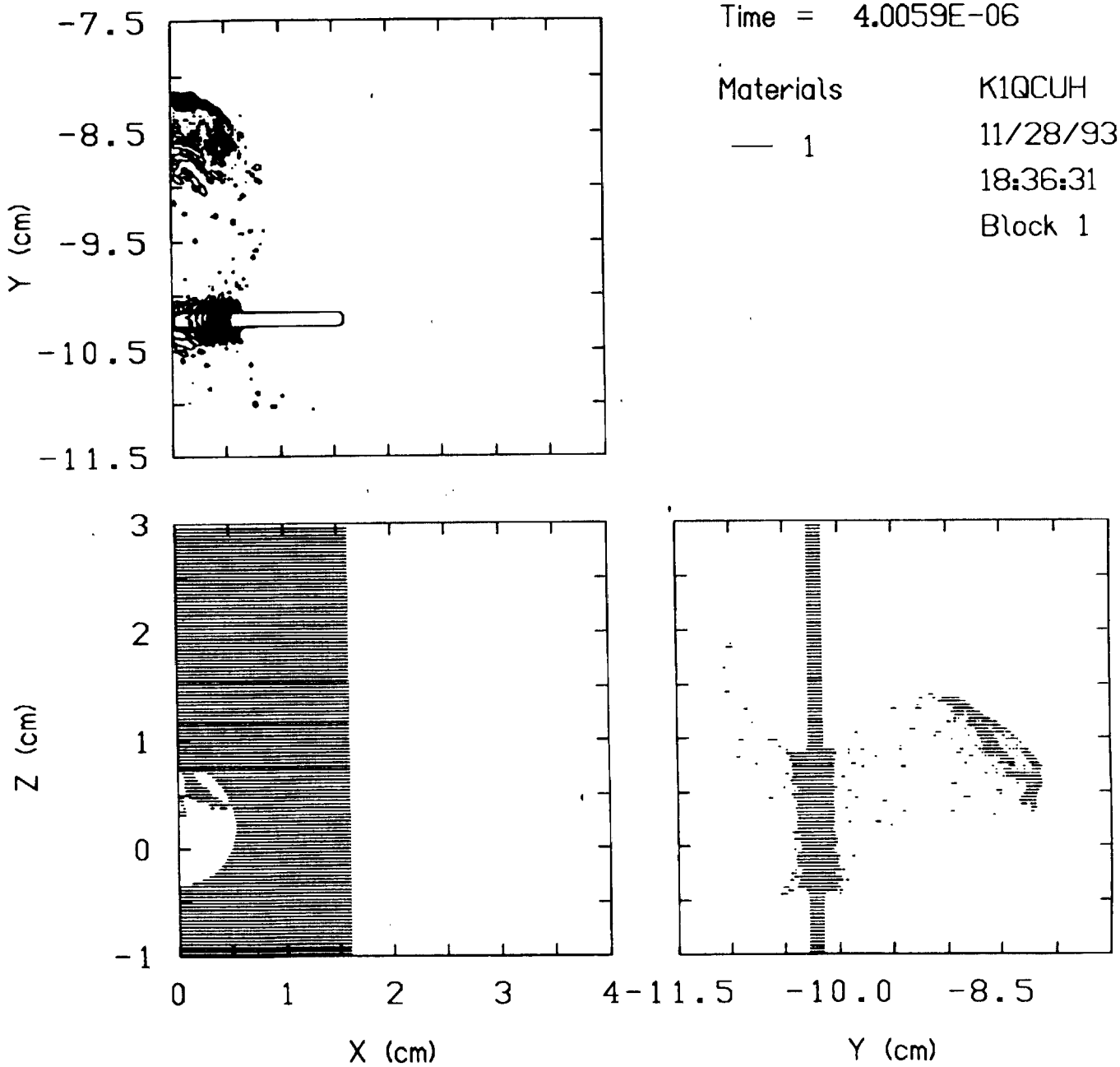
Debris Cloud Wall Impact Model
time = 0.99799E+01





disp. scale factor = 0.100E+01 (default)

Figure 11



Oblique whipple shield impact (7b)

Figure 12

Figure 13

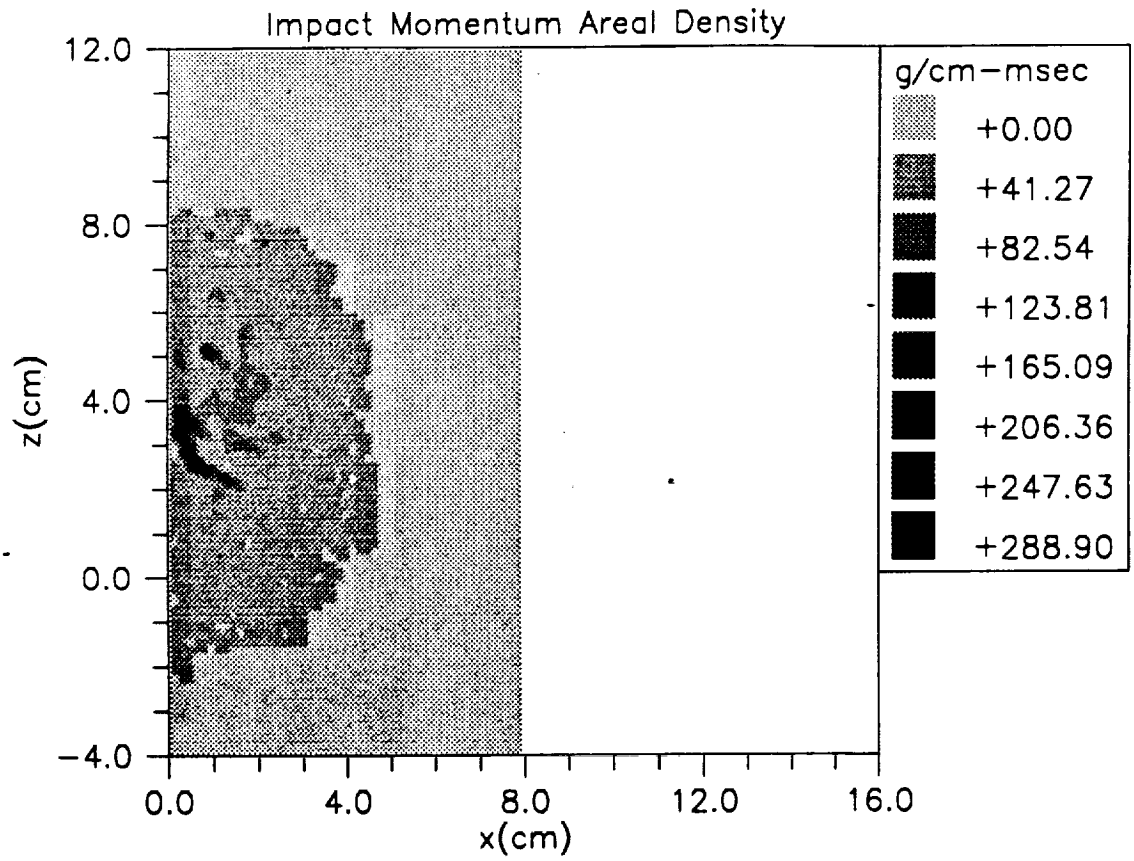

Impact Momentum Areal Density
12.0
8.0
4.0
0.0
-4.0
z(cm)
0.0
4.0
8.0
12.0
16.0
x(cm)
g/cm-msec
+0.00
+41.27
+82.54
+123.81
+165.09
+206.36
+247.63
+288.90

Figure 14

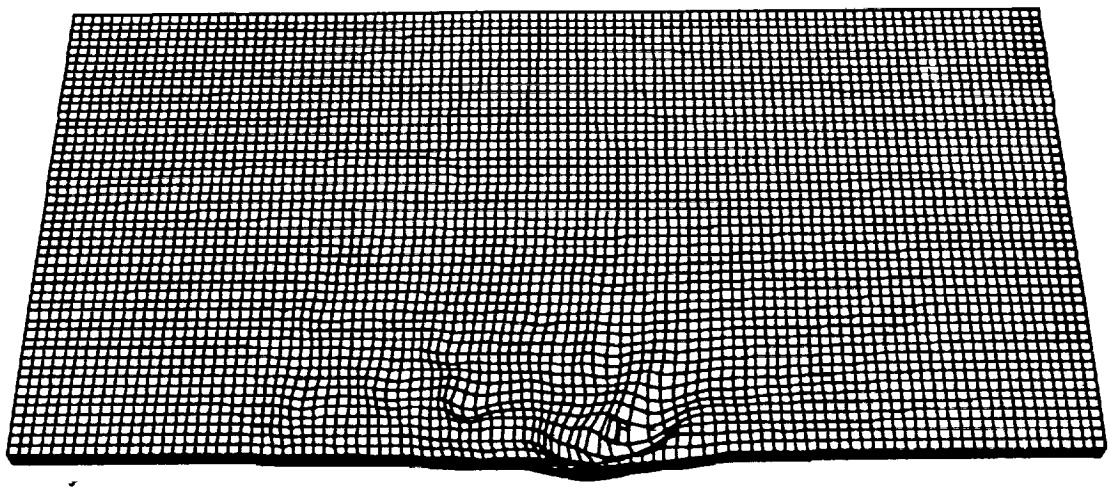
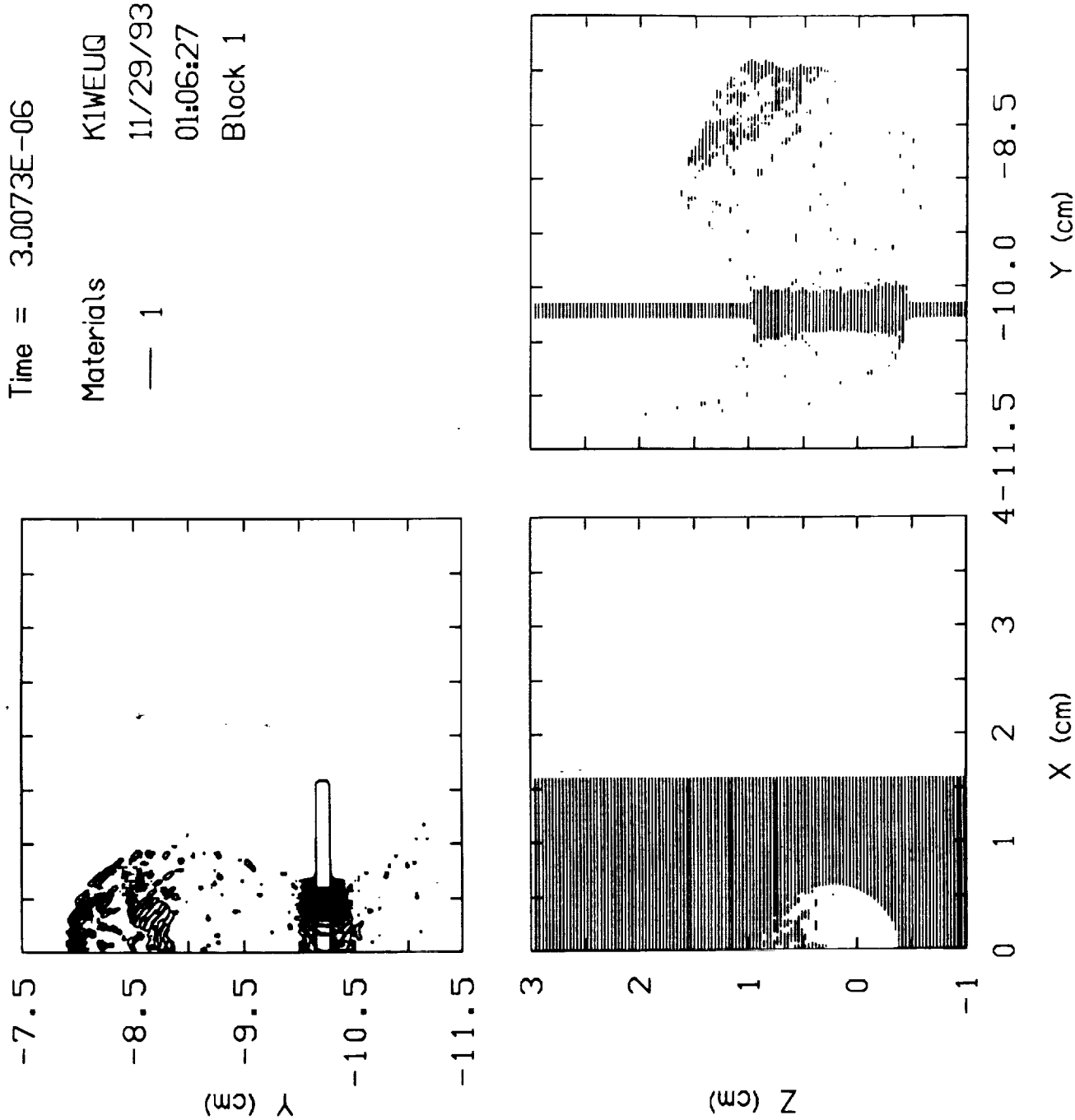
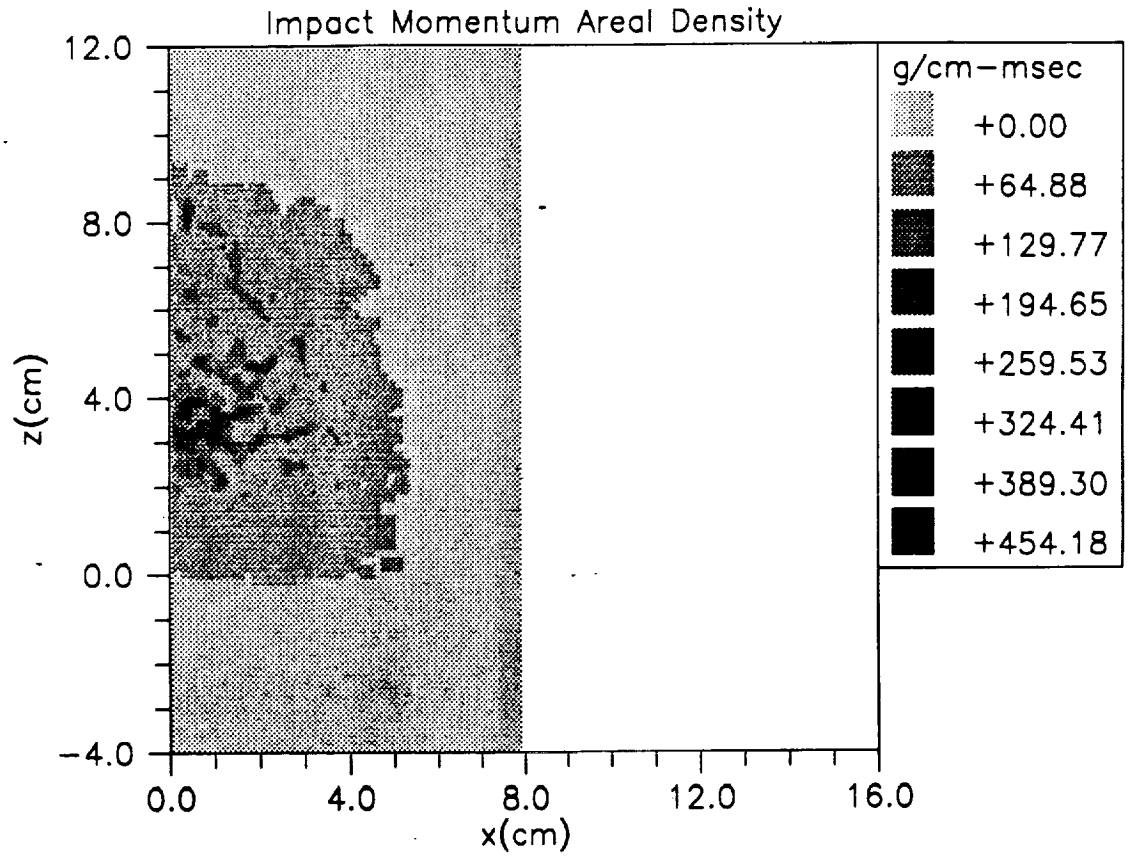

Debris Cloud Wall Impact Model
time = 0.99984E+01
disp. scale factor = 0.100E+01 (default)
x
z
y

Figure 15

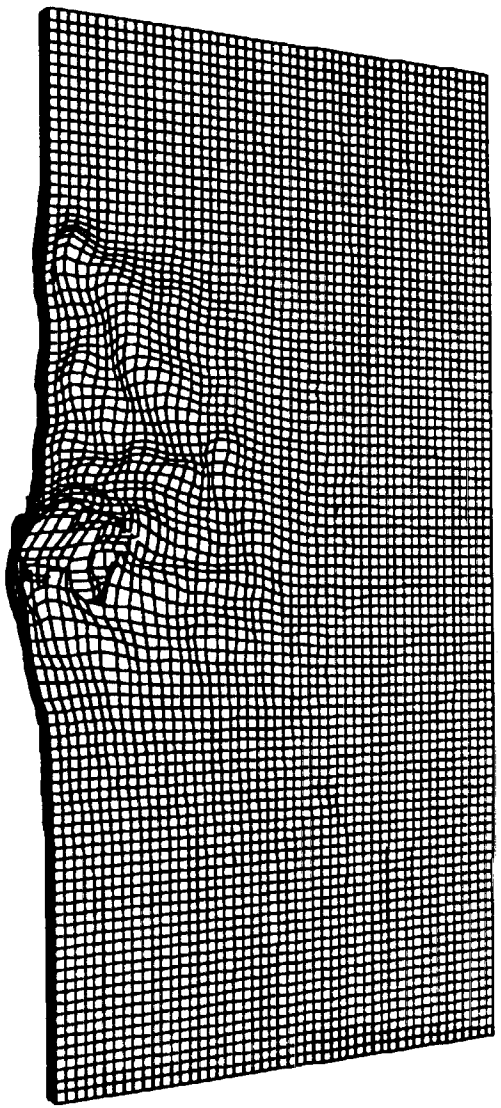


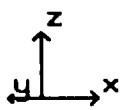
Oblique whipple shield impact (10a)

Figure 16


Impact Momentum Areal Density
g/cm-msec
+0.00
+64.88
+129.77
+194.65
+259.53
+324.41
+389.30
+454.18
12.0
8.0
4.0
0.0
-4.0
z(cm)
0.0
4.0
8.0
12.0
16.0
x(cm)

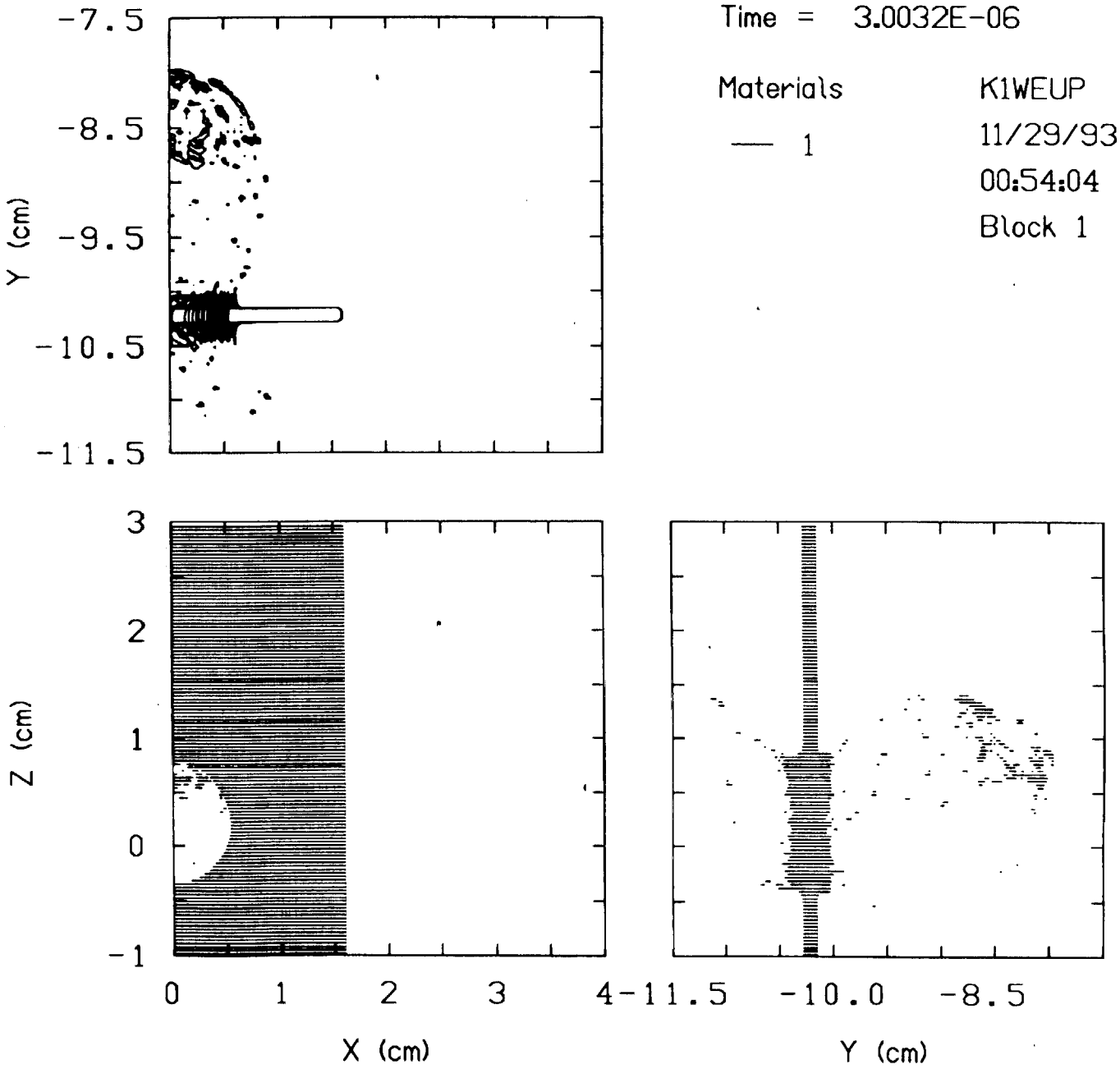
Debris Cloud Wall Impact Model
time = 0.99761E+01





disp. scale factor = 0.100E+01 (default)

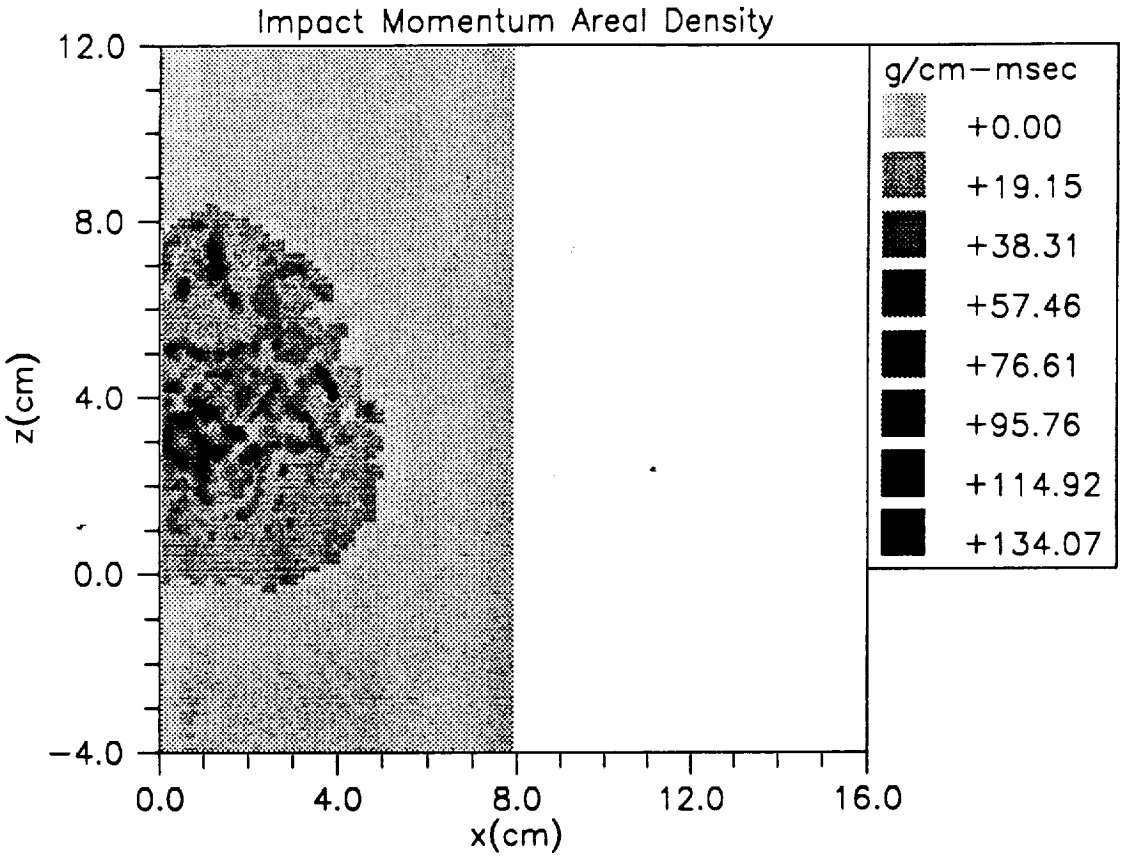
Figure 17



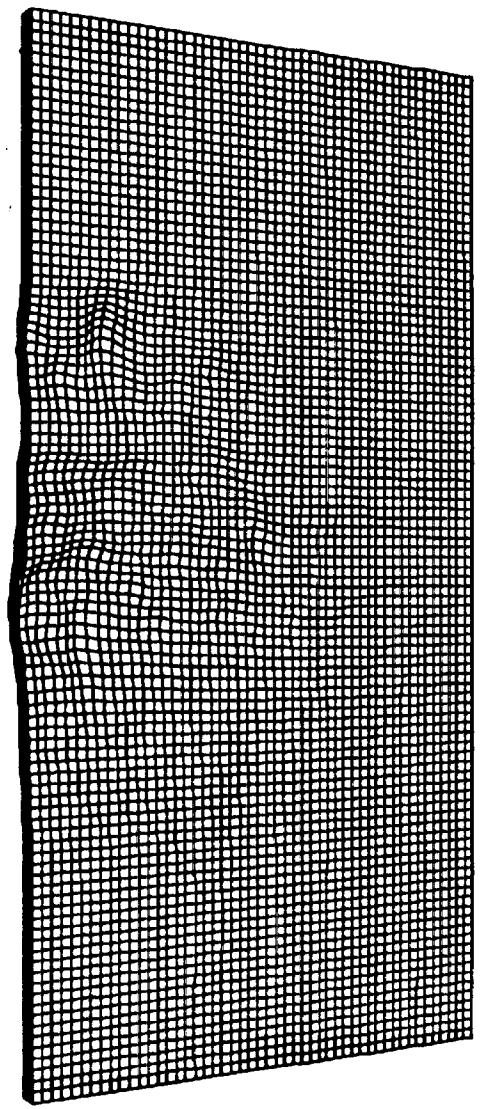
Oblique whipple shield impact (10b)

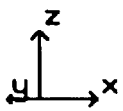
Figure 18

Figure 19


Impact Momentum Areal Density
g/cm−msec
+0.00
+19.15
+38.31
+57.46
+76.61
+95.76
+114.92
+134.07
12.0
8.0
4.0
0.0
−4.0
z(cm)
0.0
4.0
8.0
12.0
16.0
x(cm)

Debris Cloud Wall Impact Model
time = 0.99903E+01





disp. scale factor = 0.100E+01 (default)

Figure 20

APPENDIX A

Example Input File (CTH simulation)

```
*****************************************************************************
*
*eor* cgenin
*
*****************************************************************************
*
*  cthgen input
*
*****************************************************************************
*
*  Title record
*
Oblique whipple shield impact (7a)
*
*****************************************************************************
*
*  control records
*
control
  ep
* mmp
endc
*
*****************************************************************************
*
*  edit records
*
edit
  block 1
    expanded
  endb
ende
*
*****************************************************************************
*
*  mesh records
*
mesh
  block 1  geom=3dr  type=e
    x0    0.0
      x1  n=50   w=1.5875    rat=1.
    endx
    y0  -11.5
      y1  n=125  w=3.96875   rat=1.
    endy
    z0   -1.000
      z1  n=125  w=3.96875   rat=1.
    endz
    xact =   0.0       1.00
    yact = -11.5     -10.40
    zact =  -0.50      1.00
  endb
endm
*
*****************************************************************************
*
*  material insertion records
*
```

```
insertion of material
*
  block 1
*
    package projectile
      material 1
      numsub 49
      velocities  xvel 0.0  yvel 6.3441545e5  zvel 2.9583278e5
      insert sphere
        center  0.0  -10.5745  0.0
        r       0.2875
      endi
    endp
*
    package shield
      material 1
      numsub 49
      velocities  xvel 0.  yvel 0.  zvel 0.
      insert box
        x1 0.0 y1 -10.287 z1 -1.00 x2 1.5875 y2 -10.16 z2 2.96875
      endi
    endp
*
  endb
*
endi
*
*****************************************************************************
*
*  eos records
*
* eos  num 2
eos
  mat1   mgrun eos=6061-t6_al
* mat1   mgrun eos=2024-t4_al
* mat2   mgrun eos=7075-t6_al
*
* aneos1 -1  'Aluminum library'  lib=6  type=4  rhug=-1.  thug=-1.
*
ende
*
*****************************************************************************
*
*  material strength records
*
epdata
* matep 1  jfrac=7
* matep 2  jfrac=8
  matep 1  st=6061-t6_aluminum
* matep 1  st=2024-t4_aluminum
* matep 2  st=7075-t6_aluminum
* matep 1  st=user  r0=1.0
* matep 2  st=user  r0=1.0
  mix = 5
ende
*
*****************************************************************************
*
```

```
endinput
*
* end of cthgen input
*
*******************************************************************************
*
*eor* cthin
*
*******************************************************************************
*
*  cth input
*
*******************************************************************************
*
*  Title record
*
Oblique whipple shield impact (7a)
*
*******************************************************************************
*
*  control records
*
restart
  nu = 1
endr
*
control
  tst = 1.0e-6
  nsc = 5000
  cpshift = 60.
*  mmp
endc
*
convect
  convection = 1
endc
*
*******************************************************************************
*
*  time step records
*
mindt
  time = 0.  dt = 1.e-12
endn
*
maxdt
  time=0.       dt=0.5e-10
  time=3.e-10   dt=1.
endx
*
*******************************************************************************
*
*  tracer records
*
tracer
*  add  0.0  -10.528  0.0
*  add  0.0  -10.368  0.0
*  add  0.0  -10.287  0.0
```

```
*  add  0.0    -9.368
*  add  0.0    -0.127
*  add  0.0    -9.368  to  1.0  -9.368  n  3
*  add  0.0    -0.127  to  1.0  -0.127  n  9
endt
*
*******************************************************************************
*
*  edit records
*
edit
  shortt
    time=0.      dtf=5.e-6
    time=1.e-5   dtf=1.e-5
    time=1.e-4   dtf=1.e-4
  ends
  longt
    time=0.      dtf=5.e-6
    time=1.e-5   dtf=1.e-5
    time=1.e-4   dtf=1.e-4
  endl
  plott
    time=0.        dtf=0.5e-6
    time=15.0e-6   dtf=1.0e-6
    time=30.0e-6   dtf=2.0e-6
  endp
  histt
    time = 0.  dtf = 1.e-8
*    htracer1
*    htracer2
*    htracer3
*    bxyz 1  0.0 -8.01 0.0
*    bxyz 1  0.2 -8.01 0.0
*    bxyz 1  0.4 -8.01 0.0
*    bxyz 1  0.6 -8.01 0.0
*    bxyz 1  0.8 -8.01 0.0
*    bxyz 1  1.0 -8.01 0.0
  endh
ende
*
*
*  boundary condition records
*
boundary
   bhydro
     block 1
       bxb=0  bxt=1  byb=1  byt=1  bzb=1  bzt=1
     endb
   endh
endb
*
* end of cth input
*
*******************************************************************************
*
*eor* hisinp
*
*******************************************************************************
```

```
*
*  hisplt input
*
*************************************************************************
*
bottom=off
plot time cpu          v2=dt           v3=etot
*plot time pressure.1  v2=pressure.2  v3=pressure.3
*plot time yvelocity.1 v2=yvelocity.2 v3=yvelocity.3
*plot time zvelocity.1 v2=zvelocity.2 v3=zvelocity.3
*plot time pressure.4  v2=pressure.5  v3=pressure.6
*plot time pressure.7  v2=pressure.8  v3=pressure.9
*
* end of hisplt input
*
*************************************************************************
*
*eor* pltinp
*
*************************************************************************
*
*  cthplt input
*
*************************************************************************
*
units cgsk
bottom=off
*
*color table = 6
*color material  =  16   112
*color nmaterial =  31   127
*3dplot
*tcolor = 6
*
*3dline,  mat=0
3dortho, mat=0
*
*2dfix  x=0.0
*2dplot if tracer dots=density=3.0
*
* end of cthplt input
*
*************************************************************************
```

APPENDIX B

Example Input File (CTH simulation restart)

```
*******************************************************************************
*
*eor* cgenin
*
*******************************************************************************
*
*  cthgen input
*
*******************************************************************************
*
*  Title record
*
Oblique whipple shield impact (7a)
*
*******************************************************************************
*
*  control records
*
control
  ep
* mmp
endc
*
*******************************************************************************
*
*  edit records
*
edit
  block 1
    expanded
  endb
ende
*
*******************************************************************************
*
*  mesh records
*
mesh
  block 1  geom=3dr  type=e
    x0    0.0
      x1  n=50   w=1.5875    rat=1.
    endx
    y0  -11.5
      y1  n=125  w=3.96875   rat=1.
    endy
    z0   -1.000
      z1  n=125  w=3.96875   rat=1.
    endz
    xact =   0.0       1.00
    yact = -11.5     -10.40
    zact =  -0.50      1.00
  endb
endm
*
*******************************************************************************
*
*  material insertion records
*
```

```
insertion of material
*
  block 1
*
    package projectile
      material 1
      numsub 49
      velocities  xvel 0.0  yvel 6.3441545e5  zvel 2.9583278e5
      insert sphere
        center  0.0  -10.5745  0.0
        r       0.2875
      endi
    endp
*
    package shield
      material 1
      numsub 49
      velocities  xvel 0.  yvel 0.  zvel 0.
      insert box
        x1 0.0 y1 -10.287 z1 -1.00 x2 1.5875 y2 -10.16 z2 2.96875
      endi
    endp
*
  endb
*
endi
*
*****************************************************************************
*
*  eos records
*
* eos  num 2
eos
  mat1   mgrun eos=6061-t6_al
* mat1   mgrun eos=2024-t4_al
* mat2   mgrun eos=7075-t6_al
*
* aneos1 -1  'Aluminum library'  lib=6  type=4  rhug=-1.  thug=-1.
*
ende
*
*****************************************************************************
*
*  material strength records
*
epdata
* matep 1  jfrac=7
* matep 2  jfrac=8
  matep 1  st=6061-t6_aluminum
* matep 1  st=2024-t4_aluminum
* matep 2  st=7075-t6_aluminum
* matep 1  st=user  r0=1.0
* matep 2  st=user  r0=1.0
  mix = 5
ende
*
*****************************************************************************
*
```

```
endinput
*
* end of cthgen input
*
*******************************************************************************
*
*eor* cthin
*
*******************************************************************************
*
*  cth input
*
*******************************************************************************
*
*  Title record
*
Oblique whipple shield impact (7a)
*
*******************************************************************************
*
*  control records
*
restart
  newfile
  nu = 3
endr
*
control
  tst = 4.0e-6
  nsc = 5000
  cpshift = 60.
*  mmp
endc
*
convect
  convection = 1
endc
*
*******************************************************************************
*
*  time step records
*
mindt
  time = 0.  dt = 1.e-12
endn
*
maxdt
  time=0.       dt=0.5e-10
  time=3.e-10   dt=1.
endx
*
*******************************************************************************
*
*  tracer records
*
tracer
*  add  0.0  -10.528  0.0
*  add  0.0  -10.368  0.0
```

```
*  add  0.0  -10.287  0.0
*  add  0.0   -9.368
*  add  0.0   -0.127
*  add  0.0   -9.368  to  1.0  -9.368  n  3
*  add  0.0   -0.127  to  1.0  -0.127  n  9
endt
*
*******************************************************************************
*
*  edit records
*
edit
  shortt
    time=0.     dtf=5.e-6
    time=1.e-5  dtf=1.e-5
    time=1.e-4  dtf=1.e-4
  ends
  longt
    time=0.     dtf=5.e-6
    time=1.e-5  dtf=1.e-5
    time=1.e-4  dtf=1.e-4
  endl
  plott
    time=0.       dtf=1.5e-6
    time=15.0e-6  dtf=1.0e-6
    time=30.0e-6  dtf=2.0e-6
  endp
  histt
    time = 0.  dtf = 1.e-8
*    htracer1
*    htracer2
*    htracer3
*    bxyz 1  0.0 -8.01 0.0
*    bxyz 1  0.2 -8.01 0.0
*    bxyz 1  0.4 -8.01 0.0
*    bxyz 1  0.6 -8.01 0.0
*    bxyz 1  0.8 -8.01 0.0
*    bxyz 1  1.0 -8.01 0.0
  endh
ende
*
*
*  boundary condition records
*
boundary
   bhydro
     block 1
       bxb=0  bxt=1  byb=1  byt=1  bzb=1  bzt=1
     endb
   endh
endb
*
* end of cth input
*
*******************************************************************************
*
*eor* hisinp
*
```

```
*************************************************************************
*
*  hisplt input
*
*************************************************************************
*
bottom=off
plot time cpu          v2=dt           v3=etot
*plot time pressure.1  v2=pressure.2  v3=pressure.3
*plot time yvelocity.1 v2=yvelocity.2 v3=yvelocity.3
*plot time zvelocity.1 v2=zvelocity.2 v3=zvelocity.3
*plot time pressure.4  v2=pressure.5  v3=pressure.6
*plot time pressure.7  v2=pressure.8  v3=pressure.9
*
* end of hisplt input
*
*************************************************************************
*
*eor* pltinp
*
*************************************************************************
*
*  cthplt input
*
*************************************************************************
*
units cgsk
bottom=off
*
*color table = 6
*color material  =  16   112
*color nmaterial =  31   127
*3dplot
*tcolor = 6
*
*3dline,  mat=0
3dortho, mat=0
*
*2dfix  x=0.0
*2dplot if tracer dots=density=3.0
*
* end of cthplt input
*
*************************************************************************
```

APPENDIX C

Example Input File (DC3D)

```
* dc3d input file
*
*    itype      jtype
           2          1
*       i1         i2
           2         33
*       j1         j2
          64        112
*       k1         k2
          17         80
*          xlow         delta x
           0.0        0.031750
*          ylow         delta y
      -11.5000        0.031750
*          zlow         delta z
      -1.0000         0.031750
*          tcut            tmin          dencut
           25.0             0.0            0.01
*    idmax      jdmax      kdmax    idmaxn     jdmaxn     kdmaxn
          51          5        101        51          5        101
*         xdmin           xdmax         xdminn          xdmaxn
            0.0            8.00            0.0            4.00
*         ydmin           ydmax         ydminn          ydmaxn
            0.0          0.3175            0.0          0.3175
*         zdmin           zdmax         zdminn          zdmaxn
          -4.00           12.00             2.0            10.0
*           vmu           vlamb         denref             eta
          0.259           0.503          2.703          1.0e-3
*          zeta             eps         ymxref
         1.0e-9          1.0e-6         1.0e-6
*      mdfactor
           1.0
```

APPENDIX D

Example Input File Header (DYNA3D)

```
Debris Cloud Wall Impact Model                                          88 large
    2     25763     20001
    0     0    0    0    0    0    0    0    0    0
                                  e20.0
    0                                      1          0          0          0 1619
 100.0e-01     -2.e+4    50.e-01                    1.0e-18
    0
    0
    0
    1   13      2.703
e-p w/failure
     0.276 2.900e-03  3.88e-03  1.500e-0  -12.e+33
   7.20e-1



    2    1         1.0
elastic
       1.0
       0.3
```

APPENDIX E

Plotting Routine Source Code (DCMPLT)

```
c     program dcmplt
c
      dimension x(500),y(500),z(500),
     1 zz(500,500),cval(10)
c
      external grctr,efsplt
c     external grctr,egsgl,efsplt
c
      open(1,file='dcpost.plt')
c     open(2,file='dc3din')
c     open(9,file='dcmplt.out')
c
c     do 11 i=1,500
c     x(i)=0.0
c     y(i)=0.0
c     z(i)=0.0
c 11 continue
c
c     scale=1.0e+6
      scale=1.0e+3
c
      jmax=1
      kmax=1
      zmax=0.0
c
      do 10 i=1,500000
      read(1,101,end=98) j,k,xx,yy,pmag
  101 format(2i6,3e15.3)
      x(j)=xx
      y(k)=yy
      zz(j,k)=pmag*scale
      if(zz(j,k).gt.zmax) zmax=zz(j,k)
      if(j.gt.jmax) jmax=j
      if(k.gt.kmax) kmax=k
   10 continue
c
   98 dummy=1.0
c
      iunit=0
      ldzz=500
c     iopt=1+2+4
      iopt=2+8
      ncv=8
c
      cval(1)=1.0e-33
      cval(2)=zmax
c
      call grctr(jmax,kmax,x,y,zz,ldzz,iopt,ncv,cval)
      call egsgl('!contour_plot use$', 'dcmplt.d1$')
      call egsgl('.1 use$',            'dcmplt.d2$')
      call egsgl('.1 viewport$', 0.1,0.9,0.1,0.9)
      call efsplt(iunit, ' ')
c
      stop
      end
```

APPENDIX F

Post-processor Source Code (DCPOST)

```
      program dcpost
c
c**** program dcpost
c
      dimension iflg(100000),px(100000),py(100000),pz(100000),
     1 jenew(100000)
      dimension xcen(500),zcen(500),emom(500,500)
c
      character*8 ch0,ch1,ch2,ch3,ch4,ch5,ch6,ch7,ch8,ch9
c
      open(1,file ='dc3din')
      open(12,file='dc3d.dyn')
      open(61,file='dc3d.mom')
c     open(13,file='dcbase')
c
      open(21,file='dcpost.out')
      open(22,file='dcpost.plt')
c
c     time=1.0
      time=0.0
c
c     factor=0.75
c     factor=1.0
c
      read(1,110) itype,jtype
  110 format(///2i10)
      read(1,111) i1,i2,j1,j2,k1,k2
  111 format(/2i10)
      read(1,112) xlow,deltx,ylow,delty,zlow,deltz
  112 format(/2e15.6)
      read(1,113) tcut
  113 format(/e15.3)
      read(1,114) idmax,jdmax,kdmax,idmaxn,jdmaxn,kdmaxn
  114 format(/6i10)
      read(1,115) xdmin,xdmax,xdminn,xdmaxn,
     1            ydmin,ydmax,ydminn,ydmaxn,
     2            zdmin,zdmax,zdminn,zdmaxn
  115 format(/4e15.3)
      read(1,116) factor
  116 format(/////e15.3)
c
      ncnt=idmax*jdmax*kdmax+(idmax-1)*(jdmax-1)*(kdmax-1)+
     1     8+1+5
c
c     if(jtype.ne.1) go to 777
      if(jtype.eq.0) go to 777
c
      do 14 i=1,ncnt
      read(12,202)  ch0,ch1,ch2,ch3,ch4,ch5,ch6,ch7,ch8,ch9
  202 format(10a8)
c
      if(jtype.eq.3) go to 14
c
      write(21,203) ch0,ch1,ch2,ch3,ch4,ch5,ch6,ch7,ch8,ch9
  203 format(10a8)
   14 continue
c
  777 dummy=1.0
```

```
c
      do 10 j=1,100000
      iflg(j)=0
      px(j)=0.0
      py(j)=0.0
      pz(j)=0.0
   10 continue
c
      do 11 i=1,500000
c     read(12,102,end=75) j,dpx,dpy,dpz
      read(61,102,end=75) j,dpx,dpy,dpz
  102 format(i8,3e10.3)
      iflg(j)=1
      px(j)=px(j)+dpx*factor
      py(j)=py(j)+dpy*factor
      pz(j)=pz(j)+dpz*factor
   11 continue
c
   75 dummy=1.0
c
      ridmax=idmax
      rkdmax=kdmax
      deldx=(xdmax-xdmin)/(ridmax-1.0)
      deldz=(zdmax-zdmin)/(rkdmax-1.0)
      ridmxn=idmaxn
      rkdmxn=kdmaxn
      deldxn=(xdmaxn-xdminn)/(ridmxn-1.0)
      deldzn=(zdmaxn-zdminn)/(rkdmxn-1.0)
c
      do 710 k=1,kdmax-1
      do 711 i=1,idmax-1
      rk=k
      ri=i
      zcen(k)=zdmin+deldz/2.0+(rk-1.0)*deldz
      xcen(i)=xdmin+deldx/2.0+(ri-1.0)*deldx
      emom(i,k)=0.0
c
      jelem=i+(idmax-1)*(k-1)
      jenew(jelem)=0
      if(xcen(i).lt.xdminn) go to 711
      if(xcen(i).gt.xdmaxn) go to 711
      if(zcen(k).lt.zdminn) go to 711
      if(zcen(k).gt.zdmaxn) go to 711
      inew=int((xcen(i)-xdminn)/deldxn)+1
      knew=int((zcen(k)-zdminn)/deldzn)+1
      jenew(jelem)=inew+(idmaxn-1)*(knew-1)
c
  711 continue
  710 continue
c
      do 12 j=1,100000
c
      jout=j
      if(jtype.eq.3) jout=jenew(j)
      if(jout.eq.0) go to 12
c
      if(iflg(j).eq.0) go to 12
      write(21,201) jout,px(j),py(j),pz(j),time
```

```
  201 format(i8,4e10.3)
c
      if(jtype.eq.3) go to 12
c
      iel=mod(j,idmax-1)
      kel=j/(idmax-1)+1
      pmag=(px(j)**2+py(j)**2+pz(j)**2)**0.5
      emom(iel,kel)=pmag/(deldx*deldz)
c     emom(iel,kel)=pmag
c     riel=iel
c     rkel=kel
c     xi=xdmin+deldx+(riel-1.0)*deldx
c     zi=zdmin+deldz+(rkel-1.0)*deldz
c     write(22,212) iel,kel,xi,zi,pmag,time
c 212 format(2i6,4e15.3)
c
   12 continue
c
      if(jtype.eq.3) go to 99
c
      do 721 k=1,kdmax-1
      do 722 i=1,idmax-1
      write(22,213) i,k,xcen(i),zcen(k),emom(i,k)
  213 format(2i6,3e15.5)
  722 continue
  721 continue
c
   99 dummy=2.0
c
      stop
      end
```

APPENDIX G

Analysis Source Code (DC3D)

```
      program dc3d
c
c**** program dc3d
c
      common/atype/itype,jtype
      common/mesh/i1,i2,j1,j2,k1,k2,deltx,delty,deltz,
     1 xlow,ylow,zlow
      common/waldat/ywall,tcut,tmin,dencut,
     1 idmax,jdmax,kdmax,xdmin,xdmax,ydmin,ydmax,zdmin,zdmax
      common/walnew/idmaxn,jdmaxn,kdmaxn,
     1 xdminn,xdmaxn,ydminn,ydmaxn,zdminn,zdmaxn
      common/props/den,denref,rmass,vmu,vlamb,eta,zeta,phi,rj(3,3)
      common/intpar/epsint,ymxref
c
      open(1,file='dc3din')
c
      open(7,file='pltdat/dc3d.plt1')
      open(8,file='pltdat/dc3d.plt2')
      open(4,file='dc3d.cth')
      open(9,file='dc3d.dyn')
c
      open(61,file='dc3d.mom')
c
      open(10,file='dc3d.dbg')
c
      read(1,110) itype,jtype
  110 format(///2i10)
      read(1,111) i1,i2,j1,j2,k1,k2
  111 format(/2i10)
      read(1,112) xlow,deltx,ylow,delty,zlow,deltz
  112 format(/2e15.6)
      read(1,113) tcut,tmin,dencut
  113 format(/3e15.3)
      read(1,114) idmax,jdmax,kdmax,idmaxn,jdmaxn,kdmaxn
  114 format(/6i10)
      read(1,115) xdmin,xdmax,xdminn,xdmaxn,
     1            ydmin,ydmax,ydminn,ydmaxn,
     2            zdmin,zdmax,zdminn,zdmaxn
  115 format(/4e15.3)
      read(1,116) vmu,vlamb,denref,eta
  116 format(/4e15.3)
      read(1,117) zeta,epsint,ymxref
  117 format(/3e15.3)
c
      ywall=ydmin
c
c     call rdavs
c
c     if(itype.eq.2) call dyngen
      if(itype.eq.2.and.jtype.eq.1) call dyngen
      if(itype.eq.2.and.jtype.eq.2) call dyngen
      if(itype.eq.2.and.jtype.eq.3) call dyngen
      if(jtype.eq.2) go to 99
      if(jtype.eq.3) go to 99
c
      call rdavs
c
      ifst=2
```

```
      jfst=2
      kfst=2
c
c     ilast=2
c     jlast=2
c     klast=2
c
      ilast=i2-i1
      jlast=j2-j1
      klast=k2-k1
c
      lfst=1
      llast=5
c
      do 10 i=ifst,ilast
      do 11 j=jfst,jlast
      do 12 k=kfst,klast
      do 13 l=lfst,llast
      call dcelem(i,j,k,l)
   13 continue
   12 continue
   11 continue
   10 continue
c
   99 dummy=1.0
c
      stop
      end
c
      subroutine dyngen
c
c**** subroutine dyngen
c
      common/atype/itype,jtype
      common/walnew/idmaxn,jdmaxn,kdmaxn,
     1 xdminn,xdmaxn,ydminn,ydmaxn,zdminn,zdmaxn
      common/waldat/ywall,tcut,tmin,dencut,
     1 idmax,jdmax,kdmax,xdmin,xdmax,ydmin,ydmax,zdmin,zdmax
c
      dimension x(8),y(8),z(8),xn(501),yn(501),zn(501)
c
      if(jtype.ne.3) go to 88
      idmax=idmaxn
      jdmax=jdmaxn
      kdmax=kdmaxn
      xdmin=xdminn
      ydmin=ydminn
      zdmin=zdminn
      xdmax=xdmaxn
      ydmax=ydmaxn
      zdmax=zdmaxn
   88 dummy=1.0
c
      ridmax=idmax
      delx=(xdmax-xdmin)/(ridmax-1.0)
      rjdmax=jdmax
      dely=(ydmax-ydmin)/(rjdmax-1.0)
      rkdmax=kdmax
```

```
      delz=(zdmax-zdmin)/(rkdmax-1.0)
c
      do 10 i=1,idmax
      ri=i
      xn(i)=(ri-1.0)*delx
   10 continue
c
      do 11 j=1,jdmax
      rj=j
      yn(j)=(rj-1.0)*dely
   11 continue
c
      do 12 k=1,kdmax
      rk=k
      zn(k)=(rk-1.0)*delz
   12 continue
c
c     define the nodes
c
c     nconst=0
c
      do 20 j=1,jdmax
      do 21 k=1,kdmax
      do 22 i=1,idmax
c
      nconst=0
      if(i.eq.1)      nconst=1
c
      go to 761
c
      if(i.eq.idmax) nconst=4
      if(k.eq.1)     nconst=5
      if(k.eq.kdmax) nconst=5
      if(i.eq.1.and.k.eq.1)          nconst=7
      if(i.eq.1.and.k.eq.kdmax)      nconst=7
      if(i.eq.idmax.and.k.eq.1)      nconst=7
      if(i.eq.idmax.and.k.eq.kdmax) nconst=7
c
  761 dummy=1.0
c
      nnode=i+idmax*(k-1)+idmax*kdmax*(j-1)
      write(9,101) nnode,nconst,xn(i),yn(j),zn(k)
c 101 format(i5,i5,3e20.8)
  101 format(i8,i5,3e20.8)
   22 continue
   21 continue
   20 continue
c
      nconst=0
c
      x(1)=xdmin
      y(1)=ydmin-1.0
      z(1)=zdmin
      x(2)=xdmin+1.0
      y(2)=ydmin-1.0
      z(2)=zdmin
      x(3)=xdmin+1.0
      y(3)=ydmin-1.0
```

```
      z(3)=zdmin+1.0
      x(4)=xdmin
      y(4)=ydmin-1.0
      z(4)=zdmin+1.0
      x(5)=xdmin
      y(5)=ydmin-2.0
      z(5)=zdmin
      x(6)=xdmin+1.0
      y(6)=ydmin-2.0
      z(6)=zdmin
      x(7)=xdmin+1.0
      y(7)=ydmin-2.0
      z(7)=zdmin+1.0
      x(8)=xdmin
      y(8)=ydmin-2.0
      z(8)=zdmin+1.0
c
      do 40 l=1,8
      nnode=idmax+idmax*(kdmax-1)+idmax*kdmax*(jdmax-1)+l
      write(9,103) nnode,nconst,x(l),y(l),z(l)
c 103 format(i5,i5,3e20.8)
  103 format(i8,i5,3e20.8)
   40 continue
c
c     define the elements
c
      nmat=1
      ngen=0
c
      do 30 j=1,jdmax-1
      do 31 k=1,kdmax-1
      do 32 i=1,idmax-1
      nelem=i+(idmax-1)*(k-1)+(idmax-1)*(kdmax-1)*(j-1)
      n1=i+idmax*(k-1)+idmax*kdmax*(j-1)
      n2=n1+1
      n3=n2+idmax
      n4=n3-1
      n5=n1+idmax*kdmax
      n6=n5+1
      n7=n6+idmax
      n8=n7-1
c     write(9,102) nelem,nmat,ngen,n5,n6,n7,n8,n1,n2,n3,n4
c 102 format(11i5)
      write(9,102) nelem,nmat,n5,n6,n7,n8,n1,n2,n3,n4
  102 format(i8,i5,8i8)
   32 continue
   31 continue
   30 continue
c
      nmat=2
c
      nelem=idmax-1+(idmax-1)*(kdmax-1-1)+
     1 (idmax-1)*(kdmax-1)*(jdmax-1-1)+1
      n1=idmax+idmax*(kdmax-1)+idmax*kdmax*(jdmax-1)+1
      n2=n1+1
      n3=n1+2
      n4=n1+3
      n5=n1+4
```

```
      n6=n1+5
      n7=n1+6
      n8=n1+7
c     write(9,104) nelem,nmat,ngen,n1,n2,n3,n4,n5,n6,n7,n8
c 104 format(11i5)
      write(9,104) nelem,nmat,n1,n2,n3,n4,n5,n6,n7,n8
  104 format(i8,i5,8i8)
c
c     write(9,110)
c 110 format('    1    1   11',/,'    1    0',/,'    1')
c     idmp1=idmax+1
c     idmp2=idmax+2
c     write(9,111) idmp2,idmp1,n4,n3,n2,n1
c 111 format('    1    0    1    2',2i5,
c    1     /,'    1    0',4i5)
c
      write(9,110)
  110 format('       1       1   11',/,'    1    0',/,'    1')
      idmp1=idmax+1
      idmp2=idmax+2
      write(9,111) idmp2,idmp1,n4,n3,n2,n1
  111 format('       1       1       2',2i8,
     1     /,'       1',4i8)
c
      return
      end
c
      subroutine rdavs
c
c**** subroutine rdavs
c
      common/mesh/i1,i2,j1,j2,k1,k2,deltx,delty,deltz,
     1 xlow,ylow,zlow
c
      common/cthdat/  velx(75,75,75),vely(75,75,75),
     1 velz(75,75,75),pres(75,75,75),temp(75,75,75),
     2 gama(75,75,75),spen(75,75,75),vmas(75,75,75),
     3 vphi(75,75,75),cspd(75,75,75)
c
      common/cthxyz/xcth(100),ycth(100),zcth(100)
c
      common/vellng/vfx(75,75,75),vfy(75,75,75),vfz(75,75,75)
c
      character*1 chdum
c
      open(2,file='avsout')
      open(3,file='avschk')
c
c     calculate the nodal coordinates
c
      do 20 i=i1,i2
      vi=i
      xcth(i-i1+1)=xlow+(vi-1.0)*deltx-deltx
   20 continue
c
      do 21 j=j1,j2
      vj=j
      ycth(j-j1+1)=ylow+(vj-1.0)*delty-delty
```

```
   21 continue
c
      do 22 k=k1,k2
      vk=k
      zcth(k-k1+1)=zlow+(vk-1.0)*deltz-deltz
   22 continue
c
c     read in the mesh data
c
      do 400 idum=1,27
      read(2,901) chdum
  901 format(a1)
  400 continue
c
      do 10 k=1,k2-k1+1
      do 11 j=1,j2-j1+1
      do 12 i=1,i2-i1+1
      read(2,902)  velx(i,j,k),vely(i,j,k),velz(i,j,k),
     1 pres(i,j,k),temp(i,j,k),gama(i,j,k),spen(i,j,k)
      read(2,902)  vmas(i,j,k),vphi(i,j,k),
     1 cspd(i,j,k)
c
c     unit conversion factors
c
      velx(i,j,k)=velx(i,j,k)/1.0e+6
      vely(i,j,k)=vely(i,j,k)/1.0e+6
      velz(i,j,k)=velz(i,j,k)/1.0e+6
      pres(i,j,k)=pres(i,j,k)/1.0e+12
c
  902 format(7e11.3)
   12 continue
   11 continue
   10 continue
c
      do 30 k=1,k2-k1+1
      do 31 j=1,j2-j1+1
      do 32 i=1,i2-i1+1
c     write(3,903) velx(i,j,k),vely(i,j,k),velz(i,j,k),
c    1 pres(i,j,k),temp(i,j,k),gama(i,j,k),spen(i,j,k)
c     write(3,903) vmas(i,j,k),vphi(i,j,k),cspd(i,j,k),
c    1 xcth(i),ycth(j),zcth(k)
c 903 format(7e11.3)
   32 continue
   31 continue
   30 continue
c
c     calculate the nodal velocities
c
      do 40 k=2,k2-k1+1
      do 41 j=2,j2-j1+1
      do 42 i=2,i2-i1+1
      wta=vmas(i,j,k-1)
      wtb=vmas(i,j-1,k-1)
      wtc=vmas(i,j-1,k)
      wtd=vmas(i,j,k)
      va=velx(i,j,k-1)
      vb=velx(i,j-1,k-1)
      vc=velx(i,j-1,k)
```

```
      vd=velx(i,j,k)
      if(wta+wtb+wtc+wtd.gt.0.0) then
       vfx(i,j,k)=(wta*va+wtb*vb+wtc*vc+wtd*vd)/
     1            (wta+wtb+wtc+wtd)
      else
       vfx(i,j,k)=0.0
      endif
c
c     write(10,910) wta,wtb,wtc,wtd,va,vb,vc,vd
c 910 format(8e10.2)
c
      wta=vmas(i-1,j,k-1)
      wtb=vmas(i,j,k-1)
      wtc=vmas(i,j,k)
      wtd=vmas(i-1,j,k)
      va=vely(i-1,j,k-1)
      vb=vely(i,j,k-1)
      vc=vely(i,j,k)
      vd=vely(i-1,j,k)
      if(wta+wtb+wtc+wtd.gt.0.0) then
       vfy(i,j,k)=(wta*va+wtb*vb+wtc*vc+wtd*vd)/
     1            (wta+wtb+wtc+wtd)
      else
       vfy(i,j,k)=0.0
      endif
c
c     write(10,911) wta,wtb,wtc,wtd,va,vb,vc,vd
c 911 format(8e10.2)
c
      wta=vmas(i,j,k)
      wtb=vmas(i,j-1,k)
      wtc=vmas(i-1,j-1,k)
      wtd=vmas(i-1,j,k)
      va=velz(i,j,k)
      vb=velz(i,j-1,k)
      vc=velz(i-1,j-1,k)
      vd=velz(i-1,j,k)
      if(wta+wtb+wtc+wtd.gt.0.0) then
       vfz(i,j,k)=(wta*va+wtb*vb+wtc*vc+wtd*vd)/
     1            (wta+wtb+wtc+wtd)
      else
       vfz(i,j,k)=0.0
      endif
c
c     write(10,912) wta,wtb,wtc,wtd,va,vb,vc,vd
c 912 format(8e10.2)
c
   42 continue
   41 continue
   40 continue
c
      return
      end
c
      subroutine dcelem(ii,jj,kk,ll)
c
c**** subroutine dcelem
c
```

```
      common/atype/itype,jtype
      common/waldat/ywall,tcut,tmin,dencut,
     1 idmax,jdmax,kdmax,xdmin,xdmax,ydmin,ydmax,zdmin,zdmax
      common/props/den,denref,rmass,vmu,vlamb,eta,zeta,phi,rj(3,3)
      common/ndata/x1(3),x2(3),x3(3),x4(3),
     1             v1(3),v2(3),v3(3),v4(3),xc(3)
c
      dimension h(3,3),f(3,3),e(3,3),ep(3,3),p(3),
     1 y(33),yt(33,101)
      dimension param(50),time(101)
c
      external fcn
c
      call eldata(ii,jj,kk,ll)
c
c     rden=den/denref
c     if(rden.gt.1000.0) go to 99
c     if(rden.lt.0.01)   go to 99
c     if(phi.gt.0.999)   go to 99
c
      denblk=(1.-phi)*den/denref
c     if(denblk.lt.0.01) go to 99
      if(denblk.lt.dencut) go to 99
c
      call nddata(ii,jj,kk,ll)
c
c     call jcalc(den,x1,x2,x3,x4,xc,rj)
      call jcalc(x1,x2,x3,x4,xc,rj)
c
      call initlz(h,f,ep,p)
c
c     write(10,558) rmass
c 558 format(/,'inertia tensor ( mass = ',e11.3,' )',/)
c
      do 555 io=1,3
      do 556 jo=1,3
c     write(10,557) rj(io,jo)
c 557 format(e20.8)
  556 continue
  555 continue
c
c     set the state vector
c
      do 10 i=1,3
      do 11 j=1,3
      y(3*(i-1)+j)    =  h(i,j)
      y(3*(i-1)+j+9)  =  f(i,j)
      y(3*(i-1)+j+18) = ep(i,j)
c     write(10,1000) h(i,j),f(i,j),ep(i,j)
c1000 format(3e15.4)
   11 continue
   10 continue
c
      do 12 i=1,3
      y(i+27)= p(i)
      y(i+30)=xc(i)
c     write(10,1001) p(i),xc(i)
c1001 format(2e15.4)
```

```
   12 continue
c
c     set up and call the integration routine
c
      ido=1
      neq=33
      tol=1.0e-6
c
      do 20 i=1,50
      param(i)=0.0
   20 continue
c
      param(1)=1.0e-6
c     param(3)=1.0e-3
      param(4)=100000
c     param(9)=1.0e-1
c     param(10)=3
      param(12)=2
c
      do 32 i=1,neq
      yt(i,1)=y(i)
   32 continue
c
c     iend=2
      iend=1
      riend=iend
c
      tarr=1.0e+33
      vceny=p(2)/rmass
      avceny=abs(vceny)
      if(avceny.gt.0.0)
     1 tarr=abs((ywall-y(32))/vceny)
c     if(itype.eq.2.and.tarr.gt.tcut) go to 99
c     if(itype.eq.2.and.tarr.lt.tmin) go to 99
c     if(tarr.gt.tcut) go to 99
      if(tarr.ge.tcut) go to 99
      if(tarr.lt.tmin) go to 99
c
c     if(ii.ne.3)  go to 99
c     if(jj.ne.38) go to 99
c     if(kk.ne.44) go to 99
c
      delt=tarr/riend
c     if(itype.eq.2) delt=tarr/riend
c     if(itype.eq.1) delt=0.2
c
      t=0.0
      time(1)=0.0
c
      do 30 i=1,iend
c
      ri=i
c     t=(ri-1.0)*delt
      tend=t+delt
c
c     call ivprk(ido,neq,fcn,t,tend,tol,param,y)
c     call ivpbs(ido,neq,fcn,t,tend,tol,param,y)
c     call ivpag(ido,neq,fcn,fcnj,aaa,t,tend,tol,param,y)
```

```
c
c     call ifdcrk(ido,neq,t,tend,tol,param,y)
      call ivdcrk(ido,neq,t,tend,tol,param,y)
c
c     write(10,101) tarr,t,tend,y(32)
c 101 format(5e15.3)
c
      do 31 j=1,neq
      yt(j,i+1)=y(j)
   31 continue
c
      t=tend
      time(i+1)=tend
c
   30 continue
c
c     write cth input data
c
      if(itype.eq.1)
     1 call cthout(ii,jj,kk,ll,y)
      if(itype.eq.2)
     1 call dynout(ii,jj,kk,ll,y,tarr)
c
      ido=3
c     call ivprk(ido,neq,fcn,t,tend,tol,param,y)
c     call ivpbs(ido,neq,fcn,t,tend,tol,param,y)
c     call ivpag(ido,neq,fcn,fcnj,aaa,t,tend,tol,param,y)
c
c     call prtout(iend,time,yt)
c
      go to 99
c
      write(7,501) ii,jj,kk,ll,yt(31,1),yt(32,1),yt(33,1)
      write(8,501) ii,jj,kk,ll,
     1 yt(31,iend+1),yt(32,iend+1),yt(33,iend+1)
  501 format(4i6,3e15.4)
c
   99 return
      end
c
      subroutine dynout(ip,jp,kp,lp,y,tarr)
c
c**** subroutine dynout
c
      common/props/den,denref,rmass,vmu,vlamb,eta,zeta,phi,rj(3,3)
      common/ndata/x1(3),x2(3),x3(3),x4(3),
     1             v1(3),v2(3),v3(3),v4(3),xc(3)
      common/waldat/ywall,tcut,tmin,dencut,
     1 idmax,jdmax,kdmax,xdmin,xdmax,ydmin,ydmax,zdmin,zdmax
c
      dimension y(33),f(3,3),xt1(3),xt2(3),xt3(3),xt4(3),xt5(3),
     1 ptc(3),id(5),jd(5),kd(5)
c
      c1=(denref/den)**(-1./3.)
c
      do 10 i=1,3
      do 11 j=1,3
      f(i,j)=y(3*(i-1)+j+9)
```

```
c      write(4,101) i,j,f(i,j)
c 101  format(i5,i5,e15.6)
   11  continue
   10  continue
c
       call detcal(f,detf)
       denout=denref/detf
c
c      momentum enhancement
c
       ridmax=idmax
       rjdmax=jdmax
       rkdmax=kdmax
       tmass=denref*((xdmax-xdmin)/(ridmax-1.0))*
     1  ((ydmax-ydmin)/(rjdmax-1.0))*((zdmax-zdmin)/(rkdmax-1.0))
       enhmom=((tmass+rmass)/rmass)**0.5
c      enhmom=1.0
c
       do 12 i=1,3
       ptc(i)=enhmom*y(27+i)/5.0
   12  continue
c
       do 20 i=1,3
       xt1(i)=0.0
       xt2(i)=0.0
       xt3(i)=0.0
       xt4(i)=0.0
       xt5(i)=0.0
   20  continue
c
       do 23 i=1,3
       do 21 j=1,3
       xt1(i)=xt1(i)+c1*f(i,j)*(x1(j)-xc(j))
       xt2(i)=xt2(i)+c1*f(i,j)*(x2(j)-xc(j))
       xt3(i)=xt3(i)+c1*f(i,j)*(x3(j)-xc(j))
       xt4(i)=xt4(i)+c1*f(i,j)*(x4(j)-xc(j))
   21  continue
   23  continue
c
       do 22 i=1,3
       xt1(i)=xt1(i)+y(30+i)
       xt2(i)=xt2(i)+y(30+i)
       xt3(i)=xt3(i)+y(30+i)
       xt4(i)=xt4(i)+y(30+i)
       xt5(i)=y(30+i)
   22  continue
c
c      calculate the element numbers
c
c      nelmax=(idmax-1)*(jdmax-1)*(kdmax-1)
c
       ridmax=idmax
       delx=(xdmax-xdmin)/(ridmax-1.0)
c      rjdmax=jdmax
c      dely=(ydmax-ydmin)/(rjdmax-1.0)
       rkdmax=kdmax
       delz=(zdmax-zdmin)/(rkdmax-1.0)
c
```

```
      id(1)=int((xt1(1)-xdmin)/delx)+1
      kd(1)=int((xt1(3)-zdmin)/delz)+1
      id(2)=int((xt2(1)-xdmin)/delx)+1
      kd(2)=int((xt2(3)-zdmin)/delz)+1
      id(3)=int((xt3(1)-xdmin)/delx)+1
      kd(3)=int((xt3(3)-zdmin)/delz)+1
      id(4)=int((xt4(1)-xdmin)/delx)+1
      kd(4)=int((xt4(3)-zdmin)/delz)+1
      id(5)=int((xt5(1)-xdmin)/delx)+1
      kd(5)=int((xt5(3)-zdmin)/delz)+1
c
c     write(9,501) ip,jp,kp,lp
c 501 format(4i5)
c
      do 303 ic=1,5
      ia=id(ic)
      if(ia.gt.idmax-1) go to 303
      if(ia.lt.1) go to 303
      ka=kd(ic)
      if(ka.gt.kdmax-1) go to 303
      if(ka.lt.1) go to 303
      nelem=ia+(idmax-1)*(ka-1)
c     write(9,503) nelem,ptc(1),ptc(2),ptc(3),tarr
      write(61,503) nelem,ptc(1),ptc(2),ptc(3),tarr
c 503 format(i5,4e10.3)
  503 format(i8,4e10.3)
  303 continue
c
      go to 301
c
      write(9,506) xt1(1),xt1(2),xt1(3)
  506 format(3e20.8)
      write(9,507) xt2(1),xt2(2),xt2(3)
  507 format(3e20.8)
      write(9,508) xt3(1),xt3(2),xt3(3)
  508 format(3e20.8)
      write(9,598) xt4(1),xt4(2),xt4(3)
  598 format(3e20.8)
      write(9,597) xt5(1),xt5(2),xt5(3)
  597 format(3e20.8)
c
  301 dummy=1.0
c
      return
      end
c
      subroutine cthout(ip,jp,kp,lp,y)
c
c**** subroutine cthout
c
      common/props/den,denref,rmass,vmu,vlamb,eta,zeta,phi,rj(3,3)
      common/ndata/x1(3),x2(3),x3(3),x4(3),
     1             v1(3),v2(3),v3(3),v4(3),xc(3)
c
      dimension y(33),f(3,3),xt1(3),xt2(3),xt3(3),xt4(3),vtc(3)
c
      nummat=1
      numsub=49
```

```
c
      c1=(denref/den)**(1./3.)
      c2=(1.-phi)**(1./3.)
      c3=c2/c1
c
      do 10 i=1,3
      do 11 j=1,3
      f(i,j)=y(3*(i-1)+j+9)
c     write(4,101) i,j,f(i,j)
c 101 format(i5,i5,e15.6)
   11 continue
   10 continue
c
      call detcal(f,detf)
      denout=denref/detf
c
      do 12 i=1,3
      vtc(i)=1.0e+6*y(27+i)/rmass
   12 continue
c
      do 20 i=1,3
      xt1(i)=0.0
      xt2(i)=0.0
      xt3(i)=0.0
      xt4(i)=0.0
c     write(4,102) x1(i),x2(i),x4(i),x4(i),xc(i)
c 102 format(5e15.6)
   20 continue
c
      do 23 i=1,3
      do 21 j=1,3
      xt1(i)=xt1(i)+c3*f(i,j)*(x1(j)-xc(j))
      xt2(i)=xt2(i)+c3*f(i,j)*(x2(j)-xc(j))
      xt3(i)=xt3(i)+c3*f(i,j)*(x3(j)-xc(j))
      xt4(i)=xt4(i)+c3*f(i,j)*(x4(j)-xc(j))
   21 continue
   23 continue
c
c     go to 301
c
      do 22 i=1,3
      xt1(i)=xt1(i)+y(30+i)
      xt2(i)=xt2(i)+y(30+i)
      xt3(i)=xt3(i)+y(30+i)
      xt4(i)=xt4(i)+y(30+i)
   22 continue
c
c 301 dummy=1.0
c
      write(4,501) ip,jp,kp,lp
  501 format(5x,'package ',i3,',',i3,',',i3,',',i3)
      write(4,502) nummat,numsub
  502 format(7x,'material ',i3,/7x,'numsub ',i3)
      write(4,532) denout
  532 format(7x,'density = ',e20.8)
      write(4,503) vtc(1)
  503 format(7x,'xvel = ',e20.8)
      write(4,504) vtc(2)
```

```
  504 format(7x,'yvel = ',e20.8)
      write(4,594) vtc(3)
  594 format(7x,'zvel = ',e20.8)
      write(4,505)
  505 format(7x,'insert pyramid')
      write(4,506) xt1(1),xt1(2),xt1(3)
  506 format(9x,'point  = ',e15.6,', ',e15.6,', ',e15.6)
      write(4,507) xt2(1),xt2(2),xt2(3)
  507 format(9x,'point  = ',e15.6,', ',e15.6,', ',e15.6)
      write(4,508) xt3(1),xt3(2),xt3(3)
  508 format(9x,'point  = ',e15.6,', ',e15.6,', ',e15.6)
      write(4,598) xt4(1),xt4(2),xt4(3)
  598 format(9x,'vertex = ',e15.6,', ',e15.6,', ',e15.6)
      write(4,509)
  509 format(7x,'endinsert')
      write(4,591)
  591 format(5x,'endpackage')
c
      return
      end
c
      subroutine prtout(iend,time,yt)
c
c**** subroutine prtout
c
      dimension yt(33,101)
c
      dimension time(101),out(18,101),f(3,3),e(3,3),ep(3,3),
     1 c(3,3)
c
c     print output
c
c     open(9,file='pltdat/dc3d.out')
c
      open(11,file='pltdat/h11.plt')
      open(12,file='pltdat/h12.plt')
      open(13,file='pltdat/h13.plt')
      open(14,file='pltdat/h21.plt')
      open(15,file='pltdat/h22.plt')
      open(16,file='pltdat/h23.plt')
      open(17,file='pltdat/h31.plt')
      open(18,file='pltdat/h32.plt')
      open(19,file='pltdat/h33.plt')
c
      open(20,file='pltdat/f11.plt')
      open(21,file='pltdat/f12.plt')
      open(22,file='pltdat/f13.plt')
      open(23,file='pltdat/f21.plt')
      open(24,file='pltdat/f22.plt')
      open(25,file='pltdat/f23.plt')
      open(26,file='pltdat/f31.plt')
      open(27,file='pltdat/f32.plt')
      open(28,file='pltdat/f33.plt')
c
      open(29,file='pltdat/ep11.plt')
      open(30,file='pltdat/ep12.plt')
      open(31,file='pltdat/ep13.plt')
      open(32,file='pltdat/ep21.plt')
```

```
      open(33,file='pltdat/ep22.plt')
      open(34,file='pltdat/ep23.plt')
      open(35,file='pltdat/ep31.plt')
      open(36,file='pltdat/ep32.plt')
      open(37,file='pltdat/ep33.plt')
c
      open(38,file='pltdat/p1.plt')
      open(39,file='pltdat/p2.plt')
      open(40,file='pltdat/p3.plt')
c
      open(41,file='pltdat/xc1.plt')
      open(42,file='pltdat/xc2.plt')
      open(43,file='pltdat/xc3.plt')
c
      open(44,file='pltdat/detf.plt')
      open(45,file='pltdat/i1e.plt')
      open(46,file='pltdat/j2e.plt')
      open(47,file='pltdat/i1ep.plt')
      open(48,file='pltdat/j2ep.plt')
      open(49,file='pltdat/detc.plt')
      open(50,file='pltdat/out7.plt')
      open(51,file='pltdat/out8.plt')
      open(52,file='pltdat/out9.plt')
c
c     print out the states
c
c     do 80 i=1,iend+1
c     write(9,111) (yt(j,i),j=1,33)
c 111 format(11e15.6)
c  80 continue
c
c     calculate output variables
c
      do 60 i=1,18
      do 61 j=1,iend+1
      out(i,j)=0.0
   61 continue
   60 continue
c
      do 610 i=1,iend+1
c
      do 611 j=1,3
      do 612 k=1,3
      f(j,k) =yt(3*(j-1)+k+9,i)
      ep(j,k)=yt(3*(j-1)+k+18,i)
  612 continue
  611 continue
c
      call detcal(f,detval)
      call ecalc(f,e)
c
      do 614 j=1,3
      do 615 k=1,3
      if(j.eq.k) then
       deljk=1.0
      else
       deljk=0.0
      endif
```

```
      c(j,k)=deljk+2.0*e(j,k)
  615 continue
  614 continue
c
      call detcal(c,detc)
c
      out(1,i)=detval
      out(2,i)=(1./3.)*(e(1,1)+e(2,2)+e(3,3))
      out(4,i)=(1./3.)*(ep(1,1)+ep(2,2)+ep(3,3))
      out(6,i)=detc
c
  610 continue
c
      do 52 j=1,9
      j1=j
      j2=j+9
      j3=j+18
      k1=j+10
      k2=j+19
      k3=j+28
      do 51 i=1,iend+1
      write(k1,112) time(i),yt(j1,i)
      write(k2,112) time(i),yt(j2,i)
      write(k3,112) time(i),yt(j3,i)
  112 format(2e20.8)
   51 continue
   52 continue
c
      do 53 j=1,3
      j1=j+27
      j2=j+30
      k1=j+37
      k2=j+40
      do 54 i=1,iend+1
      write(k1,102) time(i),yt(j1,i)
      write(k2,102) time(i),yt(j2,i)
  102 format(2e20.8)
   54 continue
   53 continue
c
      do 55 j=1,9
      jj=j+43
      do 56 i=1,iend+1
      write(jj,113) time(i),out(j,i)
  113 format(2e20.8)
   56 continue
   55 continue
c
c     do 50 i=1,iend+1
c     out1(i)=yt(19,i)+yt(23,i)+yt(27,i)
c     out2(i)=abs(yt(19,i)-yt(23,i))
c     out3(i)=abs(yt(23,i)-yt(27,i))
c     out4(i)=abs(yt(19,i)-yt(27,i))
c     write(11,101) time(i),out1(i),out2(i),out3(i),out4(i)
c 101 format(5e15.6)
c  50 continue
c
      return
```

```
      end
c
      subroutine detcal(a,deta)
c
c**** subroutine detcal
c
      dimension a(3,3)
c
      deta=a(1,1)*(a(2,2)*a(3,3)-a(3,2)*a(2,3))-
     1     a(1,2)*(a(2,1)*a(3,3)-a(2,3)*a(3,1))+
     2     a(1,3)*(a(2,1)*a(3,2)-a(3,1)*a(2,2))
c
      return
      end
c
      subroutine eldata(ii,jj,kk,ll)
c
c**** subroutine eldata
c
      common/props/den,denref,rmass,vmu,vlamb,eta,zeta,phi,rj(3,3)
c
      common/mesh/i1,i2,j1,j2,k1,k2,deltx,delty,deltz,
     1 xlow,ylow,zlow
c
      common/cthdat/  velx(75,75,75),vely(75,75,75),
     1 velz(75,75,75),pres(75,75,75),temp(75,75,75),
     2 gama(75,75,75),spen(75,75,75),vmas(75,75,75),
     3 vphi(75,75,75),cspd(75,75,75)
c
c     rmass=vmas(ii,jj,kk)
c
      den=gama(ii,jj,kk)
      phi=vphi(ii,jj,kk)
c
c     denref=2.7
c     dmge=0.0
c     vmu  =(1.-dmge)*1.0
c     vlamb=(1.-dmge)*1.0
c     eta=1.0
c
c     write(10,100) velx(ii,jj,kk),vely(ii,jj,kk),velz(ii,jj,kk),
c    1 pres(ii,jj,kk),temp(ii,jj,kk),gama(ii,jj,kk),spen(ii,jj,kk),
c    2 vmas(ii,jj,kk),vphi(ii,jj,kk),cspd(ii,jj,kk)
c 100 format(/7e11.3,/3e11.3,/)
c
      return
      end
c
      subroutine nddata(ii,jj,kk,ll)
c
c**** subroutine nddata
c
      common/ndata/x1(3),x2(3),x3(3),x4(3),
     1             v1(3),v2(3),v3(3),v4(3),xc(3)
c
      common/mesh/i1,i2,j1,j2,k1,k2,deltx,delty,deltz,
     1 xlow,ylow,zlow
c
```

```
      common/cthdat/  velx(75,75,75),vely(75,75,75),
     1 velz(75,75,75),pres(75,75,75),temp(75,75,75),
     2 gama(75,75,75),spen(75,75,75),vmas(75,75,75),
     3 vphi(75,75,75),cspd(75,75,75)
c
      common/cthxyz/xcth(100),ycth(100),zcth(100)
c
      common/vellng/vfx(75,75,75),vfy(75,75,75),vfz(75,75,75)
c
c     set cell coordinates
c
      xn1=xcth(ii)
      xn2=xcth(ii+1)
      xn3=xcth(ii+1)
      xn4=xcth(ii)
      xn5=xcth(ii)
      xn6=xcth(ii+1)
      xn7=xcth(ii+1)
      xn8=xcth(ii)
c
      yn1=ycth(jj)
      yn2=ycth(jj)
      yn3=ycth(jj+1)
      yn4=ycth(jj+1)
      yn5=ycth(jj)
      yn6=ycth(jj)
      yn7=ycth(jj+1)
      yn8=ycth(jj+1)
c
      zn1=zcth(kk)
      zn2=zcth(kk)
      zn3=zcth(kk)
      zn4=zcth(kk)
      zn5=zcth(kk+1)
      zn6=zcth(kk+1)
      zn7=zcth(kk+1)
      zn8=zcth(kk+1)
c
c     set element coordinates
c
      go to (301,302,303,304,305) ll
c
  301 x1(1)=xn1
      x1(2)=yn1
      x1(3)=zn1
      x2(1)=xn2
      x2(2)=yn2
      x2(3)=zn2
      x3(1)=xn4
      x3(2)=yn4
      x3(3)=zn4
      x4(1)=xn5
      x4(2)=yn5
      x4(3)=zn5
      go to 306
c
  302 x1(1)=xn3
      x1(2)=yn3
```

```
            x1(3)=zn3
            x2(1)=xn4
            x2(2)=yn4
            x2(3)=zn4
            x3(1)=xn2
            x3(2)=yn2
            x3(3)=zn2
            x4(1)=xn7
            x4(2)=yn7
            x4(3)=zn7
            go to 306
      c
        303 x1(1)=xn4
            x1(2)=yn4
            x1(3)=zn4
            x2(1)=xn2
            x2(2)=yn2
            x2(3)=zn2
            x3(1)=xn7
            x3(2)=yn7
            x3(3)=zn7
            x4(1)=xn5
            x4(2)=yn5
            x4(3)=zn5
            go to 306
      c
        304 x1(1)=xn8
            x1(2)=yn8
            x1(3)=zn8
            x2(1)=xn5
            x2(2)=yn5
            x2(3)=zn5
            x3(1)=xn7
            x3(2)=yn7
            x3(3)=zn7
            x4(1)=xn4
            x4(2)=yn4
            x4(3)=zn4
            go to 306
      c
        305 x1(1)=xn6
            x1(2)=yn6
            x1(3)=zn6
            x2(1)=xn7
            x2(2)=yn7
            x2(3)=zn7
            x3(1)=xn5
            x3(2)=yn5
            x3(3)=zn5
            x4(1)=xn2
            x4(2)=yn2
            x4(3)=zn2
            go to 306
      c
        306 dummy=1.0
      c
      c     set cell velocities
      c
```

```
      vxn1=vfx(ii,jj,kk)
      vxn2=vfx(ii+1,jj,kk)
      vxn3=vfx(ii+1,jj+1,kk)
      vxn4=vfx(ii,jj+1,kk)
      vxn5=vfx(ii,jj,kk+1)
      vxn6=vfx(ii+1,jj,kk+1)
      vxn7=vfx(ii,jj+1,kk+1)
      vxn8=vfx(ii+1,jj+1,kk+1)
c
      vyn1=vfy(ii,jj,kk)
      vyn2=vfy(ii+1,jj,kk)
      vyn3=vfy(ii+1,jj+1,kk)
      vyn4=vfy(ii,jj+1,kk)
      vyn5=vfy(ii,jj,kk+1)
      vyn6=vfy(ii+1,jj,kk+1)
      vyn7=vfy(ii,jj+1,kk+1)
      vyn8=vfy(ii+1,jj+1,kk+1)
c
      vzn1=vfz(ii,jj,kk)
      vzn2=vfz(ii+1,jj,kk)
      vzn3=vfz(ii+1,jj+1,kk)
      vzn4=vfz(ii,jj+1,kk)
      vzn5=vfz(ii,jj,kk+1)
      vzn6=vfz(ii+1,jj,kk+1)
      vzn7=vfz(ii,jj+1,kk+1)
      vzn8=vfz(ii+1,jj+1,kk+1)
c
c     set nodal velocities
c
      go to (331,332,333,334,335) ll
c
  331 v1(1)=vxn1
      v1(2)=vyn1
      v1(3)=vzn1
      v2(1)=vxn2
      v2(2)=vyn2
      v2(3)=vzn2
      v3(1)=vxn4
      v3(2)=vyn4
      v3(3)=vzn4
      v4(1)=vxn5
      v4(2)=vyn5
      v4(3)=vzn5
      go to 336
c
  332 v1(1)=vxn3
      v1(2)=vyn3
      v1(3)=vzn3
      v2(1)=vxn4
      v2(2)=vyn4
      v2(3)=vzn4
      v3(1)=vxn2
      v3(2)=vyn2
      v3(3)=vzn2
      v4(1)=vxn7
      v4(2)=vyn7
      v4(3)=vzn7
      go to 336
```

```
c
  333 v1(1)=vxn4
      v1(2)=vyn4
      v1(3)=vzn4
      v2(1)=vxn2
      v2(2)=vyn2
      v2(3)=vzn2
      v3(1)=vxn7
      v3(2)=vyn7
      v3(3)=vzn7
      v4(1)=vxn5
      v4(2)=vyn5
      v4(3)=vzn5
      go to 336
c
  334 v1(1)=vxn8
      v1(2)=vyn8
      v1(3)=vzn8
      v2(1)=vxn5
      v2(2)=vyn5
      v2(3)=vzn5
      v3(1)=vxn7
      v3(2)=vyn7
      v3(3)=vzn7
      v4(1)=vxn4
      v4(2)=vyn4
      v4(3)=vzn4
      go to 336
c
  335 v1(1)=vxn6
      v1(2)=vyn6
      v1(3)=vzn6
      v2(1)=vxn7
      v2(2)=vyn7
      v2(3)=vzn7
      v3(1)=vxn5
      v3(2)=vyn5
      v3(3)=vzn5
      v4(1)=vxn2
      v4(2)=vyn2
      v4(3)=vzn2
      go to 336
c
  336 dummy=2.0
c
c     set initial coordinates
c
      go to 337
c
      x1(1)=0.0
      x1(2)=0.0
      x1(3)=0.0
      x2(1)=1.0
      x2(2)=0.0
      x2(3)=0.0
      x3(1)=0.5
      x3(2)=sqrt(0.75)
      x3(3)=0.0
```

```
      x4(1)=0.5
      x4(2)=sqrt(0.75)/3.
      x4(3)=1.0
c
c     set initial velocities
c
      v1(1)=1.0
      v1(2)=0.0
      v1(3)=0.0
      v2(1)=0.0
      v2(2)=0.0
      v2(3)=0.0
      v3(1)=0.0
      v3(2)=0.0
      v3(3)=0.0
      v4(1)=0.0
      v4(2)=0.0
      v4(3)=0.0
c
  337 dummy=3.0
c
      do 350 ip=1,3
c     write(10,340) x1(ip),x2(ip),x3(ip),x4(ip)
c 340 format(4e15.3)
  350 continue
c     write(10,341)
c 341 format(' ')
      do 352 ip=1,3
c     write(10,342) v1(ip),v2(ip),v3(ip),v4(ip)
c 342 format(4e15.3)
  352 continue
c
c 337 dummy=3.0
c
      return
      end
c
      subroutine initlz(h,f,ep,p)
c
c**** subrouitne initlz
c
      common/props/den,denref,rmass,vmu,vlamb,eta,zeta,phi,rj(3,3)
      common/ndata/x1(3),x2(3),x3(3),x4(3),
     1             v1(3),v2(3),v3(3),v4(3),xc(3)
c
c
      dimension h(3,3),f(3,3),ep(3,3),p(3)
c
c     set initial conditions
c
      f(1,1)=1.0
      f(1,2)=0.0
      f(1,3)=0.0
      f(2,1)=0.0
      f(2,2)=1.0
      f(2,3)=0.0
      f(3,1)=0.0
      f(3,2)=0.0
```

```
      f(3,3)=1.0
c
      detf=denref/den
c
      f(1,1)=detf**(1./3.)
      f(2,2)=detf**(1./3.)
      f(3,3)=detf**(1./3.)
c
c     detf=f(1,1)*(f(2,2)*f(3,3)-f(3,2)*f(2,3))-
c    1     f(1,2)*(f(2,1)*f(3,3)-f(2,3)*f(3,1))+
c    2     f(1,3)*(f(2,1)*f(3,2)-f(3,1)*f(2,2))
c
c     den=denref/detf
c
c     write(10,103) den
c 103 format(e20.8)
c
      ep(1,1)=0.0
      ep(1,2)=0.0
      ep(1,3)=0.0
      ep(2,1)=0.0
      ep(2,2)=0.0
      ep(2,3)=0.0
      ep(3,1)=0.0
      ep(3,2)=0.0
      ep(3,3)=0.0
c
c     h(1,1)=0.0
c     h(1,2)=0.0
c     h(1,3)=0.0
c     h(2,1)=0.0
c     h(2,2)=0.0
c     h(2,3)=0.0
c     h(3,1)=0.0
c     h(3,2)=0.0
c     h(3,3)=0.0
c
c     p(1)=0.0
c     p(2)=0.0
c     p(3)=0.0
c
      call ivcalc(h,p)
c
c     xc(1)=0.5
c     xc(2)=sqrt(0.75)/3.
c     xc(3)=1.0/4.0
c
      return
      end
c
      subroutine fcn(neq,t,y,yprime)
c
c**** subroutine fcn
c
      common/props/den,denref,rmass,vmu,vlamb,eta,zeta,phi,rj(3,3)
      common/ndata/x1(3),x2(3),x3(3),x4(3),
     1             v1(3),v2(3),v3(3),v4(3),xc(3)
c
```

```
      dimension y(33),yprime(33)
c
      dimension h(3,3),f(3,3),fi(3,3),e(3,3),ep(3,3),p(3),
     1 rji(3,3),rm(3,3,3,3),rk(3,3),rkp(3,3),c(3,3),ci(3,3),
     2 vg(3,3),dd(3,3),rn(3,3,3,3),vkv(3,3),xct(3)
c
      dimension hd(3,3),fd(3,3),ed(3,3),epd(3,3),pd(3),xcd(3)
c
c     define variables
c
      do 10 i=1,3
      do 11 j=1,3
      h(i,j) = y(3*(i-1)+j)
      f(i,j) = y(3*(i-1)+j+9)
      ep(i,j)= y(3*(i-1)+j+18)
   11 continue
   10 continue
c
      do 13 i=1,3
      do 14 j=1,3
      c(i,j)=0.0
      do 15 ia=1,3
      c(i,j)=c(i,j)+f(ia,i)*f(ia,j)
   15 continue
   14 continue
   13 continue
c
      ninv=3
      ldc=3
      ldcinv=3
c
c     call linrg(ninv,c,ldc,ci,ldcinv)
      iflag=1
      call matv03(iflag,ninv,c,ci)
c
      do 12 i=1,3
      p(i) = y(i+27)
      xct(i)= y(i+30)
   12 continue
c
      call ecalc(f,e)
c
c     write(10,104)
c 104 format('subroutine fcn')
c
c     call jcalc(den,x1,x2,x3,x4,rj)
c
c     write(10,105)
c 105 format('subroutine fcn jcalc')
c
c     do 201 ii=1,3
c     do 202 jj=1,3
c     write(10,102) rj(ii,jj)
c 102 format(e20.8)
c 202 continue
c 201 continue
c
      call jinv(rj,rji)
```

```
c
c     calculate transformer modulus
c
      do 20 i=1,3
      do 21 j=1,3
      do 22 ia=1,3
      do 23 ib=1,3
      if(i.eq.ib) then
       delib=1.0
      else
       delib=0.0
      endif
      if(j.eq.ib) then
       deljb=1.0
      else
       deljb=0.0
      endif
      rm(i,j,ia,ib)=0.5*(delib*f(ia,j)+f(ia,i)*deljb)
   23 continue
   22 continue
   21 continue
   20 continue
c
      do 30 i=1,3
      do 31 j=1,3
      if(i.eq.j) then
       delij=1.0
      else
       delij=0.0
      endif
c
c     rk(i,j)=(rmass/denref)*(2.*vmu*(e(i,j)-ep(i,j))+
c    1 delij*vlamb*((e(1,1)+e(2,2)+e(3,3))-
c    2 (ep(1,1)+ep(2,2)+ep(3,3))))
c     rkp(i,j)=-rk(i,j)
c
      call detcal(f,rvl)
      rk(i,j)=  (rmass/((1.-phi)*denref))*2.*vmu*((e(i,j)-
     1 (1./3.)*delij*(e(1,1)+e(2,2)+e(3,3)))-ep(i,j))
     2 -(rmass/((1.-phi)*denref))*(vlamb+(2./3.)*vmu)*(1./rvl-1.)
     3 *(1./rvl)*ci(i,j)
      rkp(i,j)=-(rmass/((1.-phi)*denref))*2.*vmu*((e(i,j)-
     1 (1./3.)*delij*(e(1,1)+e(2,2)+e(3,3)))-ep(i,j))
c
   31 continue
   30 continue
c
      do 50 i=1,3
      do 51 j=1,3
      fd(i,j)=0.0
      do 52 ia=1,3
      fd(i,j)=fd(i,j)+h(i,ia)*rji(ia,j)
   52 continue
   51 continue
   50 continue
c
      ninv=3
      ldf=3
```

```
      ldfinv=3
c
c     call linrg(ninv,f,ldf,fi,ldfinv)
      iflag=2
      call matv03(iflag,ninv,f,fi)
c
      do 300 i=1,3
      do 301 j=1,3
      vg(i,j)=0.0
      do 302 ia=1,3
      vg(i,j)=vg(i,j)+fd(i,ia)*fi(ia,j)
  302 continue
  301 continue
  300 continue
c
      do 303 i=1,3
      do 304 j=1,3
      dd(i,j)=0.5*(vg(i,j)+vg(j,i))
  304 continue
  303 continue
c
c     calculate transformer modulus
c
      do 220 i=1,3
      do 221 j=1,3
      do 222 ia=1,3
      do 223 ib=1,3
      if(i.eq.ib) then
       delib=1.0
      else
       delib=0.0
      endif
      if(j.eq.ib) then
       deljb=1.0
      else
       deljb=0.0
      endif
      rn(i,j,ib,ia)=0.5*(delib*fi(ia,j)+fi(ia,i)*deljb)
  223 continue
  222 continue
  221 continue
  220 continue
c
c     zeta=1.0
      call detcal(f,rvl)
c
      do 231 i=1,3
      do 232 j=1,3
      vkv(i,j)=0.0
      do 233 ia=1,3
      do 234 ib=1,3
      if(ia.eq.ib) then
       delab=1.0
      else
       delab=0.0
      endif
c     vkv(i,j)=vkv(i,j)+zeta*rn(ia,ib,i,j)*dd(ia,ib)
      vkv(i,j)=vkv(i,j)+zeta*(rmass*rvl/((1.-phi)*denref))
```

```
     1 *rn(ia,ib,i,j)*
c    1 delab*(1./3.)*(dd(1,1)+dd(2,2)+dd(3,3))
c    2 delab*(1./3.)*amin1(0.0,dd(1,1)+dd(2,2)+dd(3,3))
     2 dd(ia,ib)
     3 *(0.5*(dd(1,1)**2+dd(1,2)**2+dd(1,3)**2+
     4        dd(2,1)**2+dd(2,2)**2+dd(2,3)**2+
     5        dd(3,1)**2+dd(3,2)**2+dd(3,3)**2))**2
  234 continue
  233 continue
  232 continue
  231 continue
c
      do 40 i=1,3
      do 41 j=1,3
      hd(i,j)=0.0
      do 42 ia=1,3
      do 43 ib=1,3
      hd(i,j)=hd(i,j)-rm(ia,ib,i,j)*rk(ia,ib)
   43 continue
   42 continue
      hd(i,j)=hd(i,j)-vkv(i,j)
   41 continue
   40 continue
c
      do 60 i=1,3
      do 61 j=1,3
      epd(i,j)=-(1./eta)*rkp(i,j)
   61 continue
   60 continue
c
      do 70 i=1,3
      pd(i)=0.0
      xcd(i)=(1./rmass)*p(i)
   70 continue
c
c     set the vector yprime
c
      do 80 i=1,3
      do 81 j=1,3
      yprime(3*(i-1)+j)    =  hd(i,j)
      yprime(3*(i-1)+j+9)  =  fd(i,j)
      yprime(3*(i-1)+j+18) = epd(i,j)
   81 continue
   80 continue
c
      do 82 i=1,3
      yprime(i+27)= pd(i)
      yprime(i+30)=xcd(i)
   82 continue
c
      do 201 ii=1,3
      do 202 jj=1,3
c     write(10,102) f(ii,jj),fd(ii,jj),h(ii,jj),hd(ii,jj)
c 102 format(5e15.3)
  202 continue
  201 continue
c
      return
```

```
      end
c
      subroutine ecalc(f,e)
c
c**** subroutine ecalc
c
      dimension f(3,3),e(3,3),c(3,3)
c
c     calculate e
c
      do 10 i=1,3
      do 11 j=1,3
c
c     e(i,j)=0.0
      c(i,j)=0.0
      if(i.eq.j) then
       delij=1.0
      else
       delij=0.0
      endif
c
      do 12 ia=1,3
      c(i,j)=c(i,j)+f(ia,i)*f(ia,j)
   12 continue
c
      e(i,j)=0.50*(c(i,j)-delij)
c
   11 continue
   10 continue
c
      return
      end
c
      subroutine jinv(z,zi)
c
c**** subroutine jinv
c
c     inverts the symmetric 3x3 matrix z
c
      dimension z(3,3),zi(3,3)
c
      z1=z(1,1)
      z2=z(2,2)
      z3=z(3,3)
      z4=z(1,2)
      z5=z(2,3)
      z6=z(1,3)
c
      det=z1*(z2*z3-z5*z5)-z4*(z4*z3-z5*z6)+z6*(z4*z5-z2*z6)
c
       zi(1,1)=+(1./det)*(z2*z3-z5*z5)
       zi(2,1)=-(1./det)*(z4*z3-z5*z6)
       zi(3,1)=+(1./det)*(z4*z5-z2*z6)
       zi(1,2)=-(1./det)*(z4*z3-z5*z6)
       zi(2,2)=+(1./det)*(z1*z3-z6*z6)
       zi(3,2)=-(1./det)*(z1*z5-z4*z6)
       zi(1,3)=+(1./det)*(z4*z5-z2*z6)
       zi(2,3)=-(1./det)*(z1*z5-z4*z6)
```

```
      zi(3,3)=+(1./det)*(z1*z2-z4*z4)
c
      return
      end
c
      subroutine jref(g,rjp)
c
c**** subroutine jref
c
      common/props/den,denref,rmass,vmu,vlamb,eta,zeta,phi,rj(3,3)
      common/rdata/xx1(3),xx2(3),xx3(3),xx4(3),xxc(3),vref
c
      dimension g(3,3),rjp(3,3)
c
      detg=g(1,1)*(g(2,2)*g(3,3)-g(3,2)*g(2,3))-
     1     g(1,2)*(g(2,1)*g(3,3)-g(2,3)*g(3,1))+
     2     g(1,3)*(g(2,1)*g(3,2)-g(3,1)*g(2,2))
c
c     rmass=(1.0-phi)*den*vref*detg
      rmass=abs((1.0-phi)*den*vref*detg)
c
c     calculate the reference inertia tensor
c
      h1 = sqrt((xx1(1)-xxc(1))**2+(xx1(2)-
     1 xxc(2))**2+(xx1(3)-xxc(3))**2)
      h2 = sqrt((xx2(1)-xxc(1))**2+(xx2(2)-
     1 xxc(2))**2+(xx2(3)-xxc(3))**2)
      h3 = sqrt((xx3(1)-xxc(1))**2+(xx3(2)-
     1 xxc(2))**2+(xx3(3)-xxc(3))**2)
      h4 = sqrt((xx4(1)-xxc(1))**2+(xx4(2)-
     1 xxc(2))**2+(xx4(3)-xxc(3))**2)
c
      ep11 = (xx1(1)-xxc(1))/h1
      ep12 = (xx1(2)-xxc(2))/h1
      ep13 = (xx1(3)-xxc(3))/h1
c
      ep21 = (xx2(1)-xxc(1))/h2
      ep22 = (xx2(2)-xxc(2))/h2
      ep23 = (xx2(3)-xxc(3))/h2
c
      ep31 = (xx3(1)-xxc(1))/h3
      ep32 = (xx3(2)-xxc(2))/h3
      ep33 = (xx3(3)-xxc(3))/h3
c
      ep41 = (xx4(1)-xxc(1))/h4
      ep42 = (xx4(2)-xxc(2))/h4
      ep43 = (xx4(3)-xxc(3))/h4
c
      rjp11=(rmass/20.)*((h1**2)*ep11*ep11+(h2**2)*ep21*ep21
     1 +(h3**2)*ep31*ep31+(h4**2)*ep41*ep41)
      rjp12=(rmass/20.)*((h1**2)*ep11*ep12+(h2**2)*ep21*ep22
     1 +(h3**2)*ep31*ep32+(h4**2)*ep41*ep42)
      rjp13=(rmass/20.)*((h1**2)*ep11*ep13+(h2**2)*ep21*ep23
     1 +(h3**2)*ep31*ep33+(h4**2)*ep41*ep43)
c
      rjp21=(rmass/20.)*((h1**2)*ep12*ep11+(h2**2)*ep22*ep21
     1 +(h3**2)*ep32*ep31+(h4**2)*ep42*ep41)
      rjp22=(rmass/20.)*((h1**2)*ep12*ep12+(h2**2)*ep22*ep22
```

```
     1 +(h3**2)*ep32*ep32+(h4**2)*ep42*ep42)
      rjp23=(rmass/20.)*((h1**2)*ep12*ep13+(h2**2)*ep22*ep23
     1 +(h3**2)*ep32*ep33+(h4**2)*ep42*ep43)
c
      rjp31=(rmass/20.)*((h1**2)*ep13*ep11+(h2**2)*ep23*ep21
     1 +(h3**2)*ep33*ep31+(h4**2)*ep43*ep41)
      rjp32=(rmass/20.)*((h1**2)*ep13*ep12+(h2**2)*ep23*ep22
     1 +(h3**2)*ep33*ep32+(h4**2)*ep43*ep42)
      rjp33=(rmass/20.)*((h1**2)*ep13*ep13+(h2**2)*ep23*ep23
     1 +(h3**2)*ep33*ep33+(h4**2)*ep43*ep43)
c
      rjp(1,1)=rjp11
      rjp(1,2)=rjp12
      rjp(1,3)=rjp13
      rjp(2,1)=rjp21
      rjp(2,2)=rjp22
      rjp(2,3)=rjp23
      rjp(3,1)=rjp31
      rjp(3,2)=rjp32
      rjp(3,3)=rjp33
c
      return
      end
c
      subroutine jcalc(x1,x2,x3,x4,xc,rj)
c     subroutine jcalc(den,x1,x2,x3,x4,xc,rj)
c
c**** subroutine jcalc
c
      common/rdata/xx1(3),xx2(3),xx3(3),xx4(3),xxc(3),vref
c
      dimension x1(3),x2(3),x3(3),x4(3),rj(3,3),
     1 rjp(3,3),a(12,12),b(12),s(12),ainv(12,12),
     2 g(3,3),ginv(3,3),xc(3),am1(12,12),am2(12,12)
c
c     set the reference tetrahedron
c
      xx1(1)=0.0
      xx1(2)=0.0
      xx1(3)=0.0
      xx2(1)=1.0
      xx2(2)=0.0
      xx2(3)=0.0
      xx3(1)=0.5
      xx3(2)=sqrt(0.75)
      xx3(3)=0.0
      xx4(1)=0.5
      xx4(2)=sqrt(0.75)/3.
      xx4(3)=1.0
c
      xxc(1)=0.5
      xxc(2)=sqrt(0.75)/3.
      xxc(3)=1.0/4.0
c
      vref=(1./3.)*(1./2.)*sqrt(0.75)
c
c     set the right hand side vector
c
```

```
      b(1) =x1(1)
      b(2) =x1(2)
      b(3) =x1(3)
      b(4) =x2(1)
      b(5) =x2(2)
      b(6) =x2(3)
      b(7) =x3(1)
      b(8) =x3(2)
      b(9) =x3(3)
      b(10)=x4(1)
      b(11)=x4(2)
      b(12)=x4(3)
c
c     set the coefficient matrix
c
      a(1,1) =xx1(1)-xxc(1)
      a(1,2) =xx1(2)-xxc(2)
      a(1,3) =xx1(3)-xxc(3)
      a(1,4) =0.0
      a(1,5) =0.0
      a(1,6) =0.0
      a(1,7) =0.0
      a(1,8) =0.0
      a(1,9) =0.0
      a(1,10)=1.0
      a(1,11)=0.0
      a(1,12)=0.0
c
      a(2,1) =0.0
      a(2,2) =0.0
      a(2,3) =0.0
      a(2,4) =xx1(1)-xxc(1)
      a(2,5) =xx1(2)-xxc(2)
      a(2,6) =xx1(3)-xxc(3)
      a(2,7) =0.0
      a(2,8) =0.0
      a(2,9) =0.0
      a(2,10)=0.0
      a(2,11)=1.0
      a(2,12)=0.0
c
      a(3,1) =0.0
      a(3,2) =0.0
      a(3,3) =0.0
      a(3,4) =0.0
      a(3,5) =0.0
      a(3,6) =0.0
      a(3,7) =xx1(1)-xxc(1)
      a(3,8) =xx1(2)-xxc(2)
      a(3,9) =xx1(3)-xxc(3)
      a(3,10)=0.0
      a(3,11)=0.0
      a(3,12)=1.0
c
      a(4,1) =xx2(1)-xxc(1)
      a(4,2) =xx2(2)-xxc(2)
      a(4,3) =xx2(3)-xxc(3)
      a(4,4) =0.0
```

```
      a(4,5) =0.0
      a(4,6) =0.0
      a(4,7) =0.0
      a(4,8) =0.0
      a(4,9) =0.0
      a(4,10)=1.0
      a(4,11)=0.0
      a(4,12)=0.0
C
      a(5,1) =0.0
      a(5,2) =0.0
      a(5,3) =0.0
      a(5,4) =xx2(1)-xxc(1)
      a(5,5) =xx2(2)-xxc(2)
      a(5,6) =xx2(3)-xxc(3)
      a(5,7) =0.0
      a(5,8) =0.0
      a(5,9) =0.0
      a(5,10)=0.0
      a(5,11)=1.0
      a(5,12)=0.0
C
      a(6,1) =0.0
      a(6,2) =0.0
      a(6,3) =0.0
      a(6,4) =0.0
      a(6,5) =0.0
      a(6,6) =0.0
      a(6,7) =xx2(1)-xxc(1)
      a(6,8) =xx2(2)-xxc(2)
      a(6,9) =xx2(3)-xxc(3)
      a(6,10)=0.0
      a(6,11)=0.0
      a(6,12)=1.0
C
      a(7,1) =xx3(1)-xxc(1)
      a(7,2) =xx3(2)-xxc(2)
      a(7,3) =xx3(3)-xxc(3)
      a(7,4) =0.0
      a(7,5) =0.0
      a(7,6) =0.0
      a(7,7) =0.0
      a(7,8) =0.0
      a(7,9) =0.0
      a(7,10)=1.0
      a(7,11)=0.0
      a(7,12)=0.0
C
      a(8,1) =0.0
      a(8,2) =0.0
      a(8,3) =0.0
      a(8,4) =xx3(1)-xxc(1)
      a(8,5) =xx3(2)-xxc(2)
      a(8,6) =xx3(3)-xxc(3)
      a(8,7) =0.0
      a(8,8) =0.0
      a(8,9) =0.0
      a(8,10)=0.0
```

```
      a(8,11)=1.0
      a(8,12)=0.0
c
      a(9,1) =0.0
      a(9,2) =0.0
      a(9,3) =0.0
      a(9,4) =0.0
      a(9,5) =0.0
      a(9,6) =0.0
      a(9,7) =xx3(1)-xxc(1)
      a(9,8) =xx3(2)-xxc(2)
      a(9,9) =xx3(3)-xxc(3)
      a(9,10)=0.0
      a(9,11)=0.0
      a(9,12)=1.0
c
      a(10,1) =xx4(1)-xxc(1)
      a(10,2) =xx4(2)-xxc(2)
      a(10,3) =xx4(3)-xxc(3)
      a(10,4) =0.0
      a(10,5) =0.0
      a(10,6) =0.0
      a(10,7) =0.0
      a(10,8) =0.0
      a(10,9) =0.0
      a(10,10)=1.0
      a(10,11)=0.0
      a(10,12)=0.0
c
      a(11,1) =0.0
      a(11,2) =0.0
      a(11,3) =0.0
      a(11,4) =xx4(1)-xxc(1)
      a(11,5) =xx4(2)-xxc(2)
      a(11,6) =xx4(3)-xxc(3)
      a(11,7) =0.0
      a(11,8) =0.0
      a(11,9) =0.0
      a(11,10)=0.0
      a(11,11)=1.0
      a(11,12)=0.0
c
      a(12,1) =0.0
      a(12,2) =0.0
      a(12,3) =0.0
      a(12,4) =0.0
      a(12,5) =0.0
      a(12,6) =0.0
      a(12,7) =xx4(1)-xxc(1)
      a(12,8) =xx4(2)-xxc(2)
      a(12,9) =xx4(3)-xxc(3)
      a(12,10)=0.0
      a(12,11)=0.0
      a(12,12)=1.0
c
c     call inverse solver
c
      ninv=12
```

```
      lda=12
      ldainv=12
c
c     write(10,106)
c 106 format('sub jcalc before solver')
c
c     do 108 iii=1,12
c     do 109 jjj=1,12
c     write(10,110) a(iii,jjj)
c 110 format(e20.8)
c 109 continue
c 108 continue
c
c     call linrg(ninv,a,lda,ainv,ldainv)
      iflag=3
      call matv12(iflag,ninv,a,ainv)
c     call amod1(a,am1)
c     call matinv(ninv,am1,am2)
c     call amod2(am2,ainv)
c
c     write(10,107)
c 107 format('sub jcalc after solver')
c
c     calculate the solution vector
c
      do 10 i=1,12
      s(i)=0.0
      do 11 j=1,12
      s(i)=s(i)+ainv(i,j)*b(j)
   11 continue
c     write(4,876) i,b(i),s(i)
c 876 format(1x,i5,2e15.3)
   10 continue
c
c     set the tensor g
c
      g(1,1)=s(1)
      g(1,2)=s(2)
      g(1,3)=s(3)
      g(2,1)=s(4)
      g(2,2)=s(5)
      g(2,3)=s(6)
      g(3,1)=s(7)
      g(3,2)=s(8)
      g(3,3)=s(9)
c
      xc(1)=s(10)
      xc(2)=s(11)
      xc(3)=s(12)
c
c     invert the tensor g
c
c     call inverse solver
c
      ninv=3
      ldg=3
      ldginv=3
c
```

```
c     call linrg(ninv,g,ldg,ginv,ldginv)
      iflag=4
      call matv03(iflag,ninv,g,ginv)
c
      call jref(g,rjp)
c
c     calculate the inertia tensor
c
      do 20 i=1,3
      do 21 j=1,3
      rj(i,j)=0.0
      do 22 ia=1,3
      do 23 ib=1,3
      rj(i,j)=rj(i,j)+ginv(ia,i)*rjp(ia,ib)*ginv(ib,j)
   23 continue
   22 continue
   21 continue
   20 continue
c
      return
      end
c
c
      subroutine ivcalc(h,p)
c
c**** subroutine ivcalc
c
      common/props/den,denref,rmass,vmu,vlamb,eta,zeta,phi,rj(3,3)
      common/rdata/xx1(3),xx2(3),xx3(3),xx4(3),xxc(3),vref
      common/ndata/x1(3),x2(3),x3(3),x4(3),
     1             v1(3),v2(3),v3(3),v4(3),xc(3)
c
      dimension a(12,12),b(12),s(12),ainv(12,12),
     1 g(3,3),h(3,3),p(3),am1(12,12),am2(12,12)
c
c     set the right hand side vector
c
      b(1) =v1(1)
      b(2) =v1(2)
      b(3) =v1(3)
      b(4) =v2(1)
      b(5) =v2(2)
      b(6) =v2(3)
      b(7) =v3(1)
      b(8) =v3(2)
      b(9) =v3(3)
      b(10)=v4(1)
      b(11)=v4(2)
      b(12)=v4(3)
c
c     set the coefficient matrix
c
      a(1,1) =x1(1)-xc(1)
      a(1,2) =x1(2)-xc(2)
      a(1,3) =x1(3)-xc(3)
      a(1,4) =0.0
      a(1,5) =0.0
      a(1,6) =0.0
```

```
      a(1,7) =0.0
      a(1,8) =0.0
      a(1,9) =0.0
      a(1,10)=1.0
      a(1,11)=0.0
      a(1,12)=0.0
C
      a(2,1) =0.0
      a(2,2) =0.0
      a(2,3) =0.0
      a(2,4) =x1(1)-xc(1)
      a(2,5) =x1(2)-xc(2)
      a(2,6) =x1(3)-xc(3)
      a(2,7) =0.0
      a(2,8) =0.0
      a(2,9) =0.0
      a(2,10)=0.0
      a(2,11)=1.0
      a(2,12)=0.0
C
      a(3,1) =0.0
      a(3,2) =0.0
      a(3,3) =0.0
      a(3,4) =0.0
      a(3,5) =0.0
      a(3,6) =0.0
      a(3,7) =x1(1)-xc(1)
      a(3,8) =x1(2)-xc(2)
      a(3,9) =x1(3)-xc(3)
      a(3,10)=0.0
      a(3,11)=0.0
      a(3,12)=1.0
C
      a(4,1) =x2(1)-xc(1)
      a(4,2) =x2(2)-xc(2)
      a(4,3) =x2(3)-xc(3)
      a(4,4) =0.0
      a(4,5) =0.0
      a(4,6) =0.0
      a(4,7) =0.0
      a(4,8) =0.0
      a(4,9) =0.0
      a(4,10)=1.0
      a(4,11)=0.0
      a(4,12)=0.0
C
      a(5,1) =0.0
      a(5,2) =0.0
      a(5,3) =0.0
      a(5,4) =x2(1)-xc(1)
      a(5,5) =x2(2)-xc(2)
      a(5,6) =x2(3)-xc(3)
      a(5,7) =0.0
      a(5,8) =0.0
      a(5,9) =0.0
      a(5,10)=0.0
      a(5,11)=1.0
      a(5,12)=0.0
```

```
c
      a(6,1) =0.0
      a(6,2) =0.0
      a(6,3) =0.0
      a(6,4) =0.0
      a(6,5) =0.0
      a(6,6) =0.0
      a(6,7) =x2(1)-xc(1)
      a(6,8) =x2(2)-xc(2)
      a(6,9) =x2(3)-xc(3)
      a(6,10)=0.0
      a(6,11)=0.0
      a(6,12)=1.0
c
      a(7,1) =x3(1)-xc(1)
      a(7,2) =x3(2)-xc(2)
      a(7,3) =x3(3)-xc(3)
      a(7,4) =0.0
      a(7,5) =0.0
      a(7,6) =0.0
      a(7,7) =0.0
      a(7,8) =0.0
      a(7,9) =0.0
      a(7,10)=1.0
      a(7,11)=0.0
      a(7,12)=0.0
c
      a(8,1) =0.0
      a(8,2) =0.0
      a(8,3) =0.0
      a(8,4) =x3(1)-xc(1)
      a(8,5) =x3(2)-xc(2)
      a(8,6) =x3(3)-xc(3)
      a(8,7) =0.0
      a(8,8) =0.0
      a(8,9) =0.0
      a(8,10)=0.0
      a(8,11)=1.0
      a(8,12)=0.0
c
      a(9,1) =0.0
      a(9,2) =0.0
      a(9,3) =0.0
      a(9,4) =0.0
      a(9,5) =0.0
      a(9,6) =0.0
      a(9,7) =x3(1)-xc(1)
      a(9,8) =x3(2)-xc(2)
      a(9,9) =x3(3)-xc(3)
      a(9,10)=0.0
      a(9,11)=0.0
      a(9,12)=1.0
c
      a(10,1) =x4(1)-xc(1)
      a(10,2) =x4(2)-xc(2)
      a(10,3) =x4(3)-xc(3)
      a(10,4) =0.0
      a(10,5) =0.0
```

```
      a(10,6) =0.0
      a(10,7) =0.0
      a(10,8) =0.0
      a(10,9) =0.0
      a(10,10)=1.0
      a(10,11)=0.0
      a(10,12)=0.0
c
      a(11,1) =0.0
      a(11,2) =0.0
      a(11,3) =0.0
      a(11,4) =x4(1)-xc(1)
      a(11,5) =x4(2)-xc(2)
      a(11,6) =x4(3)-xc(3)
      a(11,7) =0.0
      a(11,8) =0.0
      a(11,9) =0.0
      a(11,10)=0.0
      a(11,11)=1.0
      a(11,12)=0.0
c
      a(12,1) =0.0
      a(12,2) =0.0
      a(12,3) =0.0
      a(12,4) =0.0
      a(12,5) =0.0
      a(12,6) =0.0
      a(12,7) =x4(1)-xc(1)
      a(12,8) =x4(2)-xc(2)
      a(12,9) =x4(3)-xc(3)
      a(12,10)=0.0
      a(12,11)=0.0
      a(12,12)=1.0
c
c     call inverse solver
c
      ninv=12
      lda=12
      ldainv=12
c
c     write(10,106)
c 106 format('sub jcalc before solver')
c
c     do 108 iii=1,12
c     do 109 jjj=1,12
c     write(10,110) a(iii,jjj)
c 110 format(e20.8)
c 109 continue
c 108 continue
c
c     call linrg(ninv,a,lda,ainv,ldainv)
      iflag=5
      call matv12(iflag,ninv,a,ainv)
c     call amod1(a,am1)
c     call matinv(ninv,am1,am2)
c     call amod2(am2,ainv)
c
c     write(10,107)
```

```
c 107 format('sub jcalc after solver')
c
c     calculate the solution vector
c
      do 10 i=1,12
      s(i)=0.0
      do 11 j=1,12
      s(i)=s(i)+ainv(i,j)*b(j)
   11 continue
   10 continue
c
c     set the tensor g
c
      const=(denref/den)**(1./3.)
c
      g(1,1)=s(1)*const
      g(1,2)=s(2)*const
      g(1,3)=s(3)*const
      g(2,1)=s(4)*const
      g(2,2)=s(5)*const
      g(2,3)=s(6)*const
      g(3,1)=s(7)*const
      g(3,2)=s(8)*const
      g(3,3)=s(9)*const
c
      p(1)=s(10)*rmass
      p(2)=s(11)*rmass
      p(3)=s(12)*rmass
c
c     calculate the tensor h
c
      do 20 i=1,3
      do 21 j=1,3
      h(i,j)=0.0
      do 22 ia=1,3
      h(i,j)=h(i,j)+g(i,ia)*rj(ia,j)
   22 continue
   21 continue
   20 continue
c
      return
      end
c
      subroutine ifdcrk(ido,neq,t,tend,tol,param,y)
c
      dimension y(33),yprime(33),param(50)
      dimension f1(33),f2(33),f3(33),f4(33),y1(33),y2(33),y3(33),
     1 y4(33)
c
c     external fcn
c
      nstep=1000
      rnstep=nstep
      delt=(tend-t)/rnstep
c
      do 10 i=1,nstep
c
      ri=i
```

```
      t1=t+(ri-1.0)*delt
      t2=t1+delt/2.0
      t3=t1+delt/2.0
      t4=t1+delt
c
      do 21 j=1,neq
      y1(j)=y(j)
   21 continue
c
      call fcn(neq,t1,y1,f1)
c
      do 22 j=1,neq
      y2(j)=y(j)+(delt/2.0)*f1(j)
   22 continue
c
      call fcn(neq,t2,y2,f2)
c
      do 23 j=1,neq
      y3(j)=y(j)+(delt/2.0)*f2(j)
   23 continue
c
      call fcn(neq,t3,y3,f3)
c
      do 24 j=1,neq
      y4(j)=y(j)+delt*f3(j)
   24 continue
c
      call fcn(neq,t4,y4,f4)
c
      do 30 j=1,neq
      y(j)=y(j)+(delt/6.0)*(f1(j)+2.0*f2(j)+2.0*f3(j)+f4(j))
   30 continue
c
   10 continue
c
      return
      end
c
      subroutine ivdcrk(ido,neq,tbeg,tend,tol,param,y)
c
      common/intpar/epsint,ymxref
c
      dimension y(33),yw(33),ymax(33),param(50),err(33)
      dimension f1(33),f2(33),f3(33),f4(33),y1(33),y2(33),y3(33),
     1 y4(33)
c
c     eps=1.0e-6
c     ymxref=1.0e-6
c
      eps=epsint
c
      delmin=1.0e-24
      nstep=1000000
c
      rnstep=nstep
      delt=(tend-tbeg)/rnstep
c     delmin=delt/10.0
c
```

```
      do 11 i=1,neq
      ymax(i)=amax1(ymxref,abs(y(i)))
   11 continue
c
      t=tbeg
c
  100 t1=t
      if(t+delt.gt.tend) delt=tend-t
      t2=t+delt/2.0
      t3=t+delt
c
      do 91 i=1,neq
      y1(i)=y(i)
   91 continue
c
      call rkstep(ido,neq,t1,t3,tol,param,y1,y2)
      call rkstep(ido,neq,t1,t2,tol,param,y1,y3)
      call rkstep(ido,neq,t2,t3,tol,param,y3,y4)
c
      errmax=0.0
      do 12 i=1,neq
      err(i)=abs(y4(i)-y2(i))/(eps*ymax(i))
      errmax=amax1(errmax,err(i))
   12 continue
c
      if(errmax.le.0.0) delt=2.0*delt
      if(errmax.gt.1.0) go to 101
      if(errmax.gt.0.0) delt=0.99*delt*errmax**(-1.0/5.0)
c
      do 21 i=1,neq
      y(i)=y4(i)
      ymax(i)=amax1(ymax(i),abs(y(i)))
   21 continue
c
      t=t3
      if(t.ge.tend)       go to 99
      if(delt.lt.delmin) go to 98
      go to 100
c
  101 delt=delt/2.0
      if(delt.lt.delmin) go to 98
      go to 100
c
   98 ido=98
      write(10,110) ido,t,delt
  110 format(1x,i5,2e15.3)
c 110 format(1x,'time step less than minimum, ido =',i5)
      return
c
   99 ido=99
c     write(10,120) ido,t,delt
c 120 format(1x,i5,2e15.3)
      return
c
      end
c
      subroutine rkstep(ido,neq,tbeg,tend,tol,param,y,yout)
c
```

```
      dimension y(33),param(50),yout(33)
      dimension f1(33),f2(33),f3(33),f4(33),y1(33),y2(33),y3(33),
     1 y4(33)
c
      delt=tend-tbeg
      t1=tbeg
      t2=t1+delt/2.0
      t3=t1+delt/2.0
      t4=t1+delt
c
      do 21 j=1,neq
      y1(j)=y(j)
   21 continue
c
      call fcn(neq,t1,y1,f1)
c
      do 22 j=1,neq
      y2(j)=y(j)+(delt/2.0)*f1(j)
   22 continue
c
      call fcn(neq,t2,y2,f2)
c
      do 23 j=1,neq
      y3(j)=y(j)+(delt/2.0)*f2(j)
   23 continue
c
      call fcn(neq,t3,y3,f3)
c
      do 24 j=1,neq
      y4(j)=y(j)+delt*f3(j)
   24 continue
c
      call fcn(neq,t4,y4,f4)
c
      do 30 j=1,neq
      yout(j)=y(j)+(delt/6.0)*(f1(j)+2.0*f2(j)+2.0*f3(j)+f4(j))
   30 continue
c
      return
      end
c
      subroutine matv12(iflag,n,rm,rmi)
c
c     gauss-jordon elimination
c
      dimension rm(12,12),rmw(12,12),rmi(12,12),chk(12,12),
     1 v(12),vw(12),itrak(12)
c
c     open(1,file='gauss.inp')
c     open(2,file='gauss.out')
c
c     write(10,101) n
c 101 format(/'n =',i5)
c
c     do 40 i=1,n
c     do 41 j=1,n
c     write(10,422) rm(i,j)
c 422 format(e15.3)
```

```
c  41 continue
c  40 continue
c
c     do 42 i=1,n
c     read(1,423) v(i)
c 423 format(e15.3)
c     vw(i)=v(i)
c  42 continue
c
      do 20 i=1,n
c     itrak(i)=i
      do 21 j=1,n
      rmw(i,j)=rm(i,j)
      chk(i,j)=0.0
      rmi(i,j)=0.0
      if(i.eq.j) rmi(i,j)=1.0
   21 continue
   20 continue
c
      do 10 i=1,n
c
      isav=i
      pmag=abs(rmw(i,i))
      do 301 irow=i,n
      tmpnum=abs(rmw(irow,i))
      if(tmpnum.gt.pmag) isav=irow
  301 continue
c     itrak(i)=isav
      do 302 jcol=1,n
      hold=rmw(i,jcol)
      rmw(i,jcol)=rmw(isav,jcol)
      rmw(isav,jcol)=hold
      hold=rmi(i,jcol)
      rmi(i,jcol)=rmi(isav,jcol)
      rmi(isav,jcol)=hold
  302 continue
c
      pivot=rmw(i,i)
c     vw(i)=vw(i)/pivot
c
      do 11 jj=1,n
      rmi(i,jj)=rmi(i,jj)/pivot
      rmw(i,jj)=rmw(i,jj)/pivot
   11 continue
c
      do 12 k=1,n
      if(k.eq.i) go to 12
      scale=rmw(k,i)
c     vw(k)=vw(k)-scale*vw(i)
      do 13 j=1,n
      rmi(k,j)=rmi(k,j)-scale*rmi(i,j)
      rmw(k,j)=rmw(k,j)-scale*rmw(i,j)
   13 continue
   12 continue
c
   10 continue
c
      go to 99
```

```
c
      chksum=0.0
      do 50 i=1,n
      do 51 j=1,n
      do 52 k=1,n
      chk(i,j)=chk(i,j)+rm(i,k)*rmi(k,j)
   52 continue
      if(i.eq.j) chk(i,j)=chk(i,j)-1.0
      chksum=chksum+chk(i,j)**2
   51 continue
   50 continue
c
      if(iflag.ne.4) go to 99
      ichk=0
      if(chksum.ge.1.0e-12) ichk=-999
      write(4,222) ichk,iflag,n,chksum
  222 format(1x,3i5,e15.3)
c
      do 30 i=1,n
      do 31 j=1,n
      write(4,223) i,j,rm(i,j),rmw(i,j),rmi(i,j),chk(i,j)
  223 format(1x,2i5,4e15.3)
   31 continue
   30 continue
c
   99 dummy=1.0
c
c     do 44 i=1,n
c     write(2,223) i,v(i),vw(i)
c 223 format(1x,i5,2e15.3)
c  44 continue
c
      return
      end
c
      subroutine matv03(iflag,n,rm,rmi)
c
c     gauss-jordon elimination
c
      dimension rm(3,3),rmw(3,3),rmi(3,3),chk(3,3),
     1 v(3),vw(3),itrak(3)
c
c     open(1,file='gauss.inp')
c     open(2,file='gauss.out')
c
c     write(10,101) n
c 101 format(/'n =',i5)
c
c     do 40 i=1,n
c     do 41 j=1,n
c     write(10,422) rm(i,j)
c 422 format(e15.3)
c  41 continue
c  40 continue
c
c     do 42 i=1,n
c     read(1,423) v(i)
c 423 format(e15.3)
```

```
c      vw(i)=v(i)
c   42 continue
c
       do 20 i=1,n
c      itrak(i)=i
       do 21 j=1,n
       rmw(i,j)=rm(i,j)
       chk(i,j)=0.0
       rmi(i,j)=0.0
       if(i.eq.j) rmi(i,j)=1.0
    21 continue
    20 continue
c
       do 10 i=1,n
c
       isav=i
       pmag=abs(rmw(i,i))
       do 301 irow=i,n
       tmpnum=abs(rmw(irow,i))
       if(tmpnum.gt.pmag) isav=irow
   301 continue
c      itrak(i)=isav
       do 302 jcol=1,n
       hold=rmw(i,jcol)
       rmw(i,jcol)=rmw(isav,jcol)
       rmw(isav,jcol)=hold
       hold=rmi(i,jcol)
       rmi(i,jcol)=rmi(isav,jcol)
       rmi(isav,jcol)=hold
   302 continue
c
       pivot=rmw(i,i)
c      vw(i)=vw(i)/pivot
c
       do 11 jj=1,n
       rmi(i,jj)=rmi(i,jj)/pivot
       rmw(i,jj)=rmw(i,jj)/pivot
    11 continue
c
       do 12 k=1,n
       if(k.eq.i) go to 12
       scale=rmw(k,i)
c      vw(k)=vw(k)-scale*vw(i)
       do 13 j=1,n
       rmi(k,j)=rmi(k,j)-scale*rmi(i,j)
       rmw(k,j)=rmw(k,j)-scale*rmw(i,j)
    13 continue
    12 continue
c
    10 continue
c
       go to 99
c
       chksum=0.0
       do 50 i=1,n
       do 51 j=1,n
       do 52 k=1,n
       chk(i,j)=chk(i,j)+rm(i,k)*rmi(k,j)
```

```
   52 continue
      if(i.eq.j) chk(i,j)=chk(i,j)-1.0
      chksum=chksum+chk(i,j)**2
   51 continue
   50 continue
c
      if(iflag.ne.4) go to 99
      ichk=0
      if(chksum.ge.1.0e-12) ichk=-999
      write(4,222) ichk,iflag,n,chksum
  222 format(1x,3i5,e15.3)
c
      do 30 i=1,n
      do 31 j=1,n
      write(4,223) i,j,rm(i,j),rmw(i,j),rmi(i,j),chk(i,j)
  223 format(1x,2i5,4e15.3)
   31 continue
   30 continue
c
   99 dummy=1.0
c
c     do 44 i=1,n
c     write(2,223) i,v(i),vw(i)
c 223 format(1x,i5,2e15.3)
c  44 continue
c
      return
      end
```