SANDIA REPORT
SAND2025-11083O
Printed October 14, 2025

Sandia National Laboratories

Sierra/SolidMechanics 5.26 ITAR User Manual

Sierra Solid Mechanics Team
Computational Solid Mechanics and Structural Dynamics Department
Engineering Sciences Center

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

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

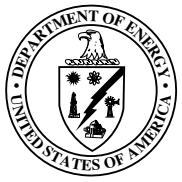
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

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

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods



1 Front Matter

Abstract

This is an addendum to the Sierra/SolidMechanics User's Guide that documents additional capabilities available only in alternate versions of the Sierra/SolidMechanics (Sierra/SM) code. These alternate versions are enhanced to provide capabilities that are regulated under the U.S. Department of State's International Traffic in Arms Regulations (ITAR) export control rules. The ITAR regulated codes are only distributed to entities that comply with the ITAR export control requirements. The ITAR enhancements to Sierra/SM include material models with an energy-dependent pressure response (appropriate for very large deformations and strain rates) and capabilities for blast modeling. This document is an addendum only; the standard Sierra/SolidMechanics User's Guide should be referenced for most general descriptions of code capability and use.

Acknowledgements

This document is the result of the collective effort of a number of individuals. This document was originally written primarily by Arne Gullerud, John Carpenter, and Bill Scherzinger. The current development team responsible for the Sierra/SolidMechanics codes includes: Frank N. Beckwith, Michael R. Buche, Gabriel J. de Frias, Scott O. Gampert, Kevin L. Manktelow, Mark T. Merewether, Scott T. Miller, Matt Mosby, Krishen J. Parmar, Matt G. Rand, Ryan T. Schlinkman, Timothy R. Shelton, Jesse D. Thomas, Jeremy Trageser, Benjamin C. Treweek, Michael G. Veilleux, and Ellen B. Wagman. This document is maintained by this team.

Presto_ITAR Release Notes

Release notes for Zapotec and Vivace are compiled alongside release notes for Sierra/SM (Adagio).

Main Contents

Contents

List of Tables

List of Figures

1.1 Introduction

This document is an addendum to the Sierra/SolidMechanics User's Guide. The standard user's guide describes the general input structure and most of the commands that are permissible in Sierra/SM and should be referenced for most documentation and usage guidelines. This addendum describes additional capabilities that are available only in ITAR versions of Sierra/SM, i.e., enhanced versions of Sierra/SM that include additional capabilities that make them regulated under the U.S. Department of State's International Traffic in Arms Regulations (ITAR). These enhanced codes are only distributed to entities that comply with the ITAR export control requirements.

The capabilities in the enhanced Sierra/SM codes that have been indicated as being ITAR restricted are, in general, only applicable to explicit transient dynamics. These capabilities deal with material response under very high rates of loading and/or deformation or with blast modeling. Most of the material response capabilities have been adopted from other export-controlled codes, such as EPIC and CTH. Some material capabilities, such as the ideal gas material model, are not explicitly export controlled but are similar in structure to the export-controlled capabilities. These capabilities are only available in the ITAR-controlled version of Sierra/SM (Presto_ITAR) and are thus documented here.

1.1.1 Document Overview

This document describes the ITAR restricted capabilities within the Sierra Solid Mechanics codes. Highlights of the document contents are as follows:

- Section 1.2 presents material models that are included in the Presto_ITAR version of Sierra/SM. These include materials from the Modular Material Models (MMM) interface (from EPIC) and CTH, as well as native implementations. These material models have a pressure response that is dependent on the energy within the element. This chapter also describes how energy deposition is enabled within the code.
- Section 1.3 describes element features that support the energy-dependent material models, such as internal iterations to resolve nonlinear energy-pressure relations.
- Section 1.4 describes a specialized boundary condition based on the ConWep code to simulate the blast pressure from an explosive.
- Section 1.5 presents the variables available for output from the Sierra/SM ITAR material models.
- Section 1.6 documents Zapotec, the two-way coupling between CTH and Sierra/SM (explicit, ITAR version).

1.1.2 Running The Code

There are two Sierra/SM ITAR codes: `Presto_ITAR` and `Zapotec`. The command to run any of these executables is essentially the same. For example, the command to run a basic Presto_ITAR analysis is:

```
sierra presto_itar -i sierra_input.i
```

Note that the capabilities defined in this addendum are only available when running the relevant executable (`presto_itar` or `zapotec`) and are not available when running the basic `adagio` executable. However, generally all analyses that run with the `adagio` executable will also run with the `presto_itar` executable.

The Sierra command also optionally takes many more options to specify the number of processors, queues to use, output log file names, etc. See the Sierra command documentation for a full description of capabilities.

1.1.3 Obtaining Support

Support for all SIERRA Mechanics codes, including Sierra/SM ITAR, can be obtained by contacting the SIERRA user support team by email at sierra-help@sandia.gov.

1.2 Materials

This chapter describes material models that exist in `Presto_ITAR` but not in standard `Adagio`. In general, all material models that have an explicit pressure dependence on energy are available only in the ITAR export-controlled version of the code. The material models documented in this manual are broken into three groups:

- **Modular Material Models (MMM)**: The MMM models are a select set of models extracted from the EPIC code and put into a common interface. They include a range of models that are widely used in modeling materials in the mild shock regime in a Lagrangian framework. See Reference [11] for more information.
- **CTH Models**: These are material models that exist within the CTH code base. This does not include all of the models in CTH; only those that directly compute a stress are included. These models include the ability to reference SESAME equation-of-state models to handle some level of phase change under very large deformations.
- **Standard Equation-of-State (EOS) Models**: These are implementations of standard EOS models within the LAME material model package [24].

All material models documented here are only available in `presto_itar` and not in the standard `adagio` executable. Only the commands specific to these models are provided here.

General information about conventions and commands for usage of material models is provided in the Sierra/SolidMechanics User's Guide.

Additional information in this section describes how to deposit energy into the elements. Only energy-dependent materials such as those described in this document have the capability to respond to deposited energy.

1.2.1 Modular Material Model (MMM) Specifications

A set of material models known as Modular Material Model (MMM) subroutines has been developed to be portable across a variety of codes, as described in [11]. These models have been made available in `Presto_ITAR`.

The following MMM models are provided in `Presto_ITAR`:

- Bodner-Partom strength model with Mie-Gruneisen EOS
- Holmquist-Johnson-Cook concrete model
- Hull concrete model
- Johnson-Cook strength model with Mie-Gruneisen EOS and Johnson-Cook failure model
- Johnson-Holmquist ceramic model
- Johnson-Holmquist-Beissel ceramic model
- Mechanical Threshold Stress (MTS) strength model with Mie-Gruneisen EOS
- Mechanical Threshold Stress (MTS) strength model with Mie-Gruneisen EOS and the TEPLA continuum level damage model
- Zerilli-Armstrong strength model for BCC metals with Mie-Gruneisen EOS
- Zerilli-Armstrong strength model for FCC metals with Mie-Gruneisen EOS

The inputs for these models are documented in the subsections below. A full description of the theory and implementation of the models is available in [11].

Properties for these models (excluding TEPLA) can be set via a *begin initial conditions* command block. Statistical (Weibull) distributions of properties are able to be used through the standard Sierra/SolidMechanics commands. Spaces in property names are replaced by underscores in the initial conditions, e.g., `SHEAR MODULUS` would become `SHEAR_MODULUS`.

1.2.1.1 MMM Models with Temperature

The Bodner-Partom, Johnson-Cook, MTS, TEPLA+MTS, and both Zerilli-Armstrong MMM models evolve the element temperature field. Prescribed temperatures and thermal strains are disallowed with these models. Uniform initial temperature can be set through the `INIT TEMPERATURE` property. An initial element temperature field must be provided either through a *begin initial temperature* command or from a transfer. The presence of an element temperature supersedes the specification of `INIT TEMPERATURE`.

1.2.1.2 Consistent Equation of State Initialization

For certain scenarios, a preload simulation is performed as setup for an explicit dynamics simulation that will use MMM models. Material models appropriate for preloading are very frequently different than the equation of state-based MMM models, and oftentimes include thermal strains and/or other temperature effects. Consistent initialization of the equation of state is necessary to prevent non-physical or spurious material response. The temperature specified at the end of a preload is almost never the same temperature that the equation of state will calculate. As such, we implement the theory outlined in [23] to properly initialize the equation of state. This involves augmenting the temperature equation with an `energy offset` as well as calculating a non-zero initial internal energy.

MMM models will use this procedure to consistently initialize the EOS if three element variables are specified: `initial_density`, `initial_pressure`, and `initial_temperature`. If present, the MMM model will use these three fields to compute the energy offset and initial internal energy. Some useful commands for utilizing this capability are:

```
begin user variable initial_pressure
  type = element real
  initial value = 0.0
end
begin user variable initial_density
  type = element real
  initial value = 0.0
end
begin user variable initial_temperature
  type = element real
  initial value = 0.0
end

begin initial condition
  block = block_1
  read variable = element_density
  initialize variable name = initial_density
  variable type = element
  time = last
```

(continues on next page)

(continued from previous page)

```
end initial condition

begin initial condition
  block = block_1
  read variable = pressure
  initialize variable name = initial_pressure
  variable type = element
  time = last
end initial condition

begin initial condition
  block = block_1
  read variable = temperature
  initialize variable name = initial_temperature
  variable type = element
  time = last
end initial condition
```

For some models, you may need the following to output an element density field:

```
begin function ElementDensity
  type = analytic
  expression variable: m = element element_mass
  expression variable: v = element volume
  evaluate expression = "m/v"
end

begin user output
  compute element element_density as function ElementDensity
end
```

Consistent initialization for multiple integration point elements, e.g., total Lagrange elements, requires single values of density, pressure, and temperature per element. Values for each integration point are not permitted.

1.2.1.3 Bodner-Partom Strength Model with Mie-Gruneisen EOS

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL BPSTRESS_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
```

(continues on next page)

(continued from previous page)

```
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    ABS ZERO TEMP = <real>ABS_ZERO_TEMP
    INIT TEMPERATURE = <real>INIT_TEMPERATURE
    SPECIFIC HEAT = <real>SPECIFIC_HEAT
    INIT STATE VAR Z0 = <real>INIT_STATE_VAR_Z0
    MAX RATE D0 = <real>MAX_RATE_D0
    MAX STATE VAR Z1 = <real>MAX_STATE_VAR_Z1
    STRAIN HARD PAR ALPHA = <real>STRAIN_HARD_PAR_ALPHA
    STRAIN HARD PAR M0 = <real>STRAIN_HARD_PAR_M0
    STRAIN HARD PAR M1 = <real>STRAIN_HARD_PAR_M1
    STRAIN RATE EXP N0 = <real>STRAIN_RATE_EXP_N0
    THERM SOFT PAR N1 = <real>THERM_SOFT_PAR_N1
    GRUN COEF = <real>GRUN_COEF
    MIEGRU COEF K2 = <real>MIEGRU_COEF_K2
    MIEGRU COEF K3 = <real>MIEGRU_COEF_K3
    MAX TENS PRESS = <real>MAX_TENS_PRESS
  END [PARAMETERS FOR MODEL BPSTRESS_MMM]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This specification activates the Bodner-Partom Stress model with a Mie-Gruneisen EOS. The expression for the yield function of this model is:

$$\sigma = Z\left(-\left(\frac{2n}{n+1}\right)\ln\left(\frac{\sqrt{3}\dot{\epsilon_p}}{2D_0}\right)\right)^{\frac{1}{2n}} \tag{1.1}$$

where n, Z, and m are defined by

$$n = n_0 + \frac{n_1}{T} \tag{1.2}$$

$$Z = Z_1 - (Z_1 - Z_0)\exp\left(\frac{m - m_0 - m_1}{\alpha} - m_0 W_p\right) \tag{1.3}$$

$$m = m_0 + m_1\exp(-\alpha W_p) \tag{1.4}$$

where $\dot{\epsilon_p}$ is the equivalent plastic strain rate, D_0 is the maximum allowable equivalent plastic strain rate, T is the absolute temperature, and W_p is the plastic work per initial volume. Z_0, Z_1, n_0, n_1, m_0, m_1, α, and D_0 are all material constants.

The pressure response is described by a cubic Mie-Gruneisen model:

$$P = \left(K_1\mu + K_2\mu^2 + K_3\mu^3\right)\left(1 - \frac{\Gamma\mu}{2}\right) + \Gamma E_s(1+\mu) \tag{1.5}$$

where $\mu = \frac{V_0}{V} - 1$, Γ is the Gruneisen coefficient, V_0 and V are the initial and current volumes, respectively, K_1 is the elastic bulk modulus, and K_2 and K_3 are material constants.

The Bodner-Partom command block starts with the input line:

```
BEGIN PARAMETERS FOR MODEL BPSTRESS_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL BPSTRESS_MMM]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- Only two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following command lines are required:
  - The yield stress of the material is defined with the `YIELD STRESS` command line.
  - The initial density of the material is defined with the `INITIAL DENSITY` command line. Set this equal to the density specified with the `DENSITY` command line.
  - The temperature at absolute zero is defined with the `ABS ZERO TEMP` command line.
  - The specific heat is defined with the `SPECIFIC HEAT` command line.
  - The material parameters Z0, D0, Z1, ALPHA, M0, M1, N0, and N1 are defined with the corresponding command lines listed above.
  - The Gruneisen parameter Gamma is defined with the `GRUN COEF` command line.
  - The K2 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K2` command line.
  - The K3 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K3` command line.
  - The maximum permitted tensile pressure is defined with the `MAX TENS PRESS` command line.

Output variables available for this model are listed in Table 1.2.

1.2.1.4 Holmquist-Johnson-Cook Concrete Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL HJCCONCRETE_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    COMP STREN FC = <real>COMP_STREN_FC
    DAMAGE COEF D1 = <real>DAMAGE_COEF_D1
    DAMAGE EXP D2 = <real>DAMAGE_EXP_D2
    INIT SHEAR MODULUS = <real>INIT_SHEAR_MODULUS
    MAX STRESS = <real>MAX_STRESS
    MAX TENS PRESS T = <real>MAX_TENS_PRESS_T
    MIN FAIL STRAIN = <real>MIN_FAIL_STRAIN
    PCRUSH = <real>PCRUSH
    PLOCKI = <real>PLOCKI
    PRESS COEF K1 = <real>PRESS_COEF_K1
    PRESS COEF K2 = <real>PRESS_COEF_K2
    PRESS COEF K3 = <real>PRESS_COEF_K3
    PRESS HARD COEF B = <real>PRESS_HARD_COEF_B
    PRESS HARD EXP N = <real>PRESS_HARD_EXP_N
    STRAIN RATE COEF C = <real>STRAIN_RATE_COEF_C
    UCRUSH = <real>UCRUSH
    ULOCK = <real>ULOCK
    YIELD STRESS A = <real>YIELD_STRESS_A
   END [PARAMETERS FOR MODEL HJCCONCRETE_MMM]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This specification activates the Holmquist-Johnson-Cook concrete model. This model has a yield surface defined by:

$$\sigma = [A(1-D) + BP^{*n}][1 + C\ln\dot{\epsilon}^*] \tag{1.6}$$

where A, B, n, and C are material constants. Additionally, $\dot{\epsilon}^* = \dot{\epsilon}/\dot{\epsilon}_0$, where $\dot{\epsilon}$ is the total equivalent strain rate and $\dot{\epsilon}_0 = 1.0\,\mathrm{s}^{-1}$. P^* is the pressure normalized by f'_c (the uniaxial compressive strength at $\dot{\epsilon}^* = 1.0$). The value of σ can be limited to σ_{max} if specified in the input file. D is the damage term, which is computed through the equation:

$$D = \sum \frac{\Delta\epsilon_p + \Delta\mu_p}{\epsilon_p^f + \mu_p^f} \tag{1.7}$$

where ϵ denotes equivalent plastic strain, μ denotes plastic volumetric strain, ^{f} indicates values at failure, and Δ indicates change over a step. The combined failure strain $\epsilon_p^f + \mu_p^f$ is set to $D_1(P^* + T^*)^{D_2}$, where D_1 and D_2 are material constants, P^* was defined previously, and $T^* = T/f_c'$, where T is the maximum permitted tensile pressure.

The compressive pressure response is dependent upon the values of volumetric crush (μ_{crush}) and lock (μ_{lock}), where $\mu = V_0/V - 1$, and V and V_0 are the current and initial volumes. At strains below μ_{crush}, the bulk modulus is constant and equal to P_{crush}/μ_{crush}. At volume strains above μ_{lock}, the material is considered to be fully compressed with no voids, and is described as:

$$P = K_1\bar{\mu} + K_2\bar{\mu}^2 + K_3\bar{\mu}^3 \tag{1.8}$$

where $\bar{\mu} = (\mu - \mu_{lock})/(1 + \mu_{lock})$. Between μ_{crush} and μ_{lock}, voids are crushed out of the material, and a linear fit is made between the states at μ_{crush} and μ_{lock}.

The tensile pressure response is defined as $P = K\mu$ before μ_{crush}, $P = K_1\bar{\mu}$ after μ_{plock} (note this is different than μ_{lock}), and is linearly interpolated between the states at μ_{crush} and μ_{plock} when between these values. A limit is placed on the tensile pressure by the expression $P_{max} = T(1 - D)$, using the T described previously.

The command block for this model starts with the input line:

```
BEGIN PARAMETERS FOR MODEL HJCCONCRETE_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL HJCCONRETE_MMM]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- Only two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following command lines are required:
  - The yield stress of the material is defined with the `YIELD STRESS A` command line.
  - The initial density of the material is defined with the `INITIAL DENSITY` command line. Set this equal to the density specified with the `DENSITY` command line.
  - The line `COMP STREN FC` sets the value of f_c'.

- The material constants D_1 and D_2 are used in the damage evolution equation.
- The initial shear modulus is set through the command `INIT SHEAR MODULUS`.
- The maximum permitted equivalent compressive stress is set by the command `MAX STRESS`.
- The maximum permitted tensile stress (T) is set by the command `MAX TEN PRESS T`.
- The minimum failure strain is set with the command `MIN FAIL STRAIN`.
- The pressure and volumetric strain at crush are set with the commands `PCRUSH` and `UCRUSH`, respectively.
- The pressure and volumetric strain at volumetric locking (fully dense material) are set with the commands `PLOCKI` and `ULOCK`, respectively.
- The fully dense compressive pressure constants K_1, K_2, and K_3 are specified through the related command lines.
- The yield function material constants B, n, and C are specified through the related command lines.

Output variables available for this model are listed in Table 1.5. More information about this model is available in [11].

1.2.1.5 Hull Concrete Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL HULLCONCRETE_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    KLOCK = <real>KLOCK
    MAX STRESS = <real>MAX_STRESS
    MAX TENS PRESS T = <real>MAX_TENS_PRESS_T
    PCRUSH = <real>PCRUSH
    PRESS COEF K1 = <real>PRESS_COEF_K1
    PRESS COEF K2 = <real>PRESS_COEF_K2
    PRESS COEF K3 = <real>PRESS_COEF_K3
```

(continues on next page)

(continued from previous page)

```
    PRESS HARD COEF B = <real>PRESS_HARD_COEF_B
    STRAIN RATE COEF C = <real>STRAIN_RATE_COEF_C
    UCRUSH = <real>UCRUSH
    ULOCK = <real>ULOCK
    YIELD STRESS A = <real>YIELD_STRESS_A
  END [PARAMETERS FOR MODEL HULLCONCRETE_MMM ]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This specification activates the Hull concrete model. This model has a yield surface defined by:

$$\sigma = [A + BP][1 + C \ln \dot{\epsilon}^*] \tag{1.9}$$

where A, B, and C are material constants. Additionally, $\dot{\epsilon}^* = \dot{\epsilon}/\dot{\epsilon}_0$, where $\dot{\epsilon}$ is the total equivalent strain rate and $\dot{\epsilon}_0 = 1.0s^{-1}$. The value of σ can be limited to σ_{max} if specified in the input file.

The compressive pressure response is dependent upon the values of volumetric crush (μ_{crush}) and lock (μ_{lock}), where $\mu = V_0/V - 1$, and V and V_0 are the current and initial volumes. At strains below μ_{crush}, the bulk modulus is constant and equal to P_{crush}/μ_{crush}. Between μ_{crush} and μ_{lock},

$$P = P_{crush} + K_1\bar{\mu} + K_2\bar{\mu}^2 + K_3\bar{\mu}^3 \tag{1.10}$$

where $\bar{\mu} = \mu - \mu_{crush}$. At volume strains above μ_{lock},

$$P = K_{lock}(\mu - \mu_0) \tag{1.11}$$

where μ_0 is the volumetric strain after unloading down to $P = 0$ from $\mu = \mu_{lock}$.

The command block for this model starts with the input line:

```
BEGIN PARAMETERS FOR MODEL HULLCONCRETE_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL HULLCONCRETE_MMM]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- Only two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.

  - Lambda is defined with the `LAMBDA` command line.

- The following command lines are required:
  - The yield stress of the material is defined with the `YIELD STRESS A` command line.
  - The initial density of the material is defined with the `INITIAL DENSITY` command line. Set this equal to the density specified with the `DENSITY` command line.
  - The maximum permitted equivalent compressive stress is set by the command `MAX STRESS`.
  - The maximum permitted tensile stress (T) is set by the command `MAX TEN PRESS T`.
  - The pressure and volumetric strain at crush are set with the commands `PCRUSH` and `UCRUSH`, respectively.
  - The volumetric strain at volumetric locking (fully dense material) is set with the command `ULOCK`.
  - The pressure constants K_1, K_2, and K_3 are specified through the related command lines.
  - The yield function material constants B and C are specified through the related command lines.

Output variables available for this model are listed in Table Table 1.6. More information about this model is available in [11].

1.2.1.6 Johnson-Cook Strength Model with Mie-Gruneisen EOS and Johnson-Cook Failure Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL JCSTRESS_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    ART VIS CL = <real>Linear_Artificial_Bulk_Viscosity
    ART VIS CQ = <real>Quadratic_Artificial_Bulk_Viscosity
    INIT TEMPERATURE = <real>INIT_TEMPERATURE
    MELT TEMPERATURE = <real>MELT_TEMPERATURE
```

(continues on next page)

(continued from previous page)

```
    ROOM TEMPERATURE = <real>ROOM_TEMPERATURE
    SPECIFIC HEAT = <real>SPECIFIC_HEAT
    JCF MODEL = { NONE | ORIGINAL | MODIFIED | LAME }
    JCF D1 = <real>JCF_D1
    JCF D2 = <real>JCF_D2
    JCF D3 = <real>JCF_D3
    JCF D4 = <real>JCF_D4
    JCF D5 = <real>JCF_D5
    JCF EFMIN = <real>JCF_EFMIN
    JCF KSTAR = <real>KSTAR
    JCF LAMBDA = <real>LAMBDA
    JCF LFAIL = <real>JCF_LFAIL
    JCF PFAIL = <real>JCF_PFAIL
    JCF WM = <real>JCF_WM
    JCF REFVOL = <real>JCF_REFVOL
    JCF ICSEED = <integer> JCF_ICSEED
    JCF ITSEED = <integer> JCF_ITSEED
    MAX STRESS = <real>MAX_STRESS
    PRESS HARD COEF = <real>PRESS_HARD_COEF
    STRAIN HARD COEF = <real>STRAIN_HARD_COEF
    STRAIN HARD EXP = <real>STRAIN_HARD_EXP
    STRAIN RATE COEF = <real>STRAIN_RATE_COEF
    STRAIN RATE MODEL = { LOG | POWER }
    THERM SOFT EXP = <real>THERM_SOFT_EXP
    MIEGRU FORM = { CUBIC | USUP }
    GRUN COEF = <real>GRUN_COEF
    MIEGRU COEF K2 = <real>MIEGRU_COEF_K2
    MIEGRU COEF K3 = <real>MIEGRU_COEF_K3
    MIEGRU CSBULK = <real>CSBULK
    MIEGRU SLOPE = <real>SLOPE
    MAX TENS PRESS = <real>MAX_TENS_PRESS
    CRITICAL FAILURE PARAMETER = <real> d_crit
    #
    #  MODULAR_FAILURE Failure model definitions
    #
    PRESSURE MULTIPLIER          = PRESSURE_INDEPENDENT | WILKINS
                                   | USER_DEFINED (PRESSURE_INDEPENDENT)
    LODE ANGLE MULTIPLIER        = LODE_ANGLE_INDEPENDENT |
                                   WILKINS (LODE_ANGLE_INDEPENDENT)
    TRIAXIALITY MULTIPLIER       = TRIAXIALITY_INDEPENDENT | JOHNSON_
↪COOK
                                   | USER_DEFINED (TRIAXIALITY_
↪INDEPENDENT)
    RATE FAIL MULTIPLIER         = RATE_INDEPENDENT | JOHNSON_COOK
                                   | USER_DEFINED (RATE_INDEPENDENT)
```

(continues on next page)

(continued from previous page)

```
    TEMPERATURE FAIL MULTIPLIER = TEMPERATURE_INDEPENDENT | JOHNSON_
↪COOK
                                | USER_DEFINED (TEMPERATURE_
↪INDEPENDENT)
    #
    # Individual multiplier definitions
    #
    PRESSURE MULTIPLIER = WILKINS
    WILKINS ALPHA       = <real> alpha
    WILKINS PRESSURE    = <real> B
    #
    PRESSURE MULTIPLIER = USER_DEFINED
    PRESSURE MULTIPLIER FUNCTION = <string> pressure_multiplier_fun_
↪name
    #
    LODE ANGLE MULTIPLIER = WILKINS
    WILKINS BETA          = <real> Beta
    #
    TRIAXIALITY MULTIPLIER = JOHNSON_COOK
    JOHNSON COOK D1        = <real> D_1
    JOHNSON COOK D2        = <real> D_2
    JOHNSON COOK D3        = <real> D_3
    #
    TRIAXIALITY MULTIPLIER = USER_DEFINED
    TRIAXIALITY MULTIPLIER FUNCTION = <string> triax_multiplier_fun_
↪name
    #
    RATE FAIL MULTIPLIER = JOHNSON_COOK
    JOHNSON COOK D4      = <real> D_4
    REFERENCE RATE       = <real> dot_epsilon_0
    #
    RATE FAIL MULTIPLIER = USER_DEFINED
    RATE FAIL MULTIPLIER FUNCTION = <string> rate_fail_multiplier_fun_
↪name
    #
    TEMPERATURE FAIL MULTIPLIER = JOHNSON_COOK
    JOHNSON COOK D5             = <real> D_5
    REFERENCE TEMPERATURE       = <real> T_ref
    MELTING TEMPERATURE         = <real> T_melt
    #
    TEMPERATURE FAIL MULTIPLIER          = USER_DEFINED
    TEMPERATURE FAIL MULTIPLIER FUNCTION = <string> temp_multiplier_
↪fun_name
    #
  END [PARAMETERS FOR MODEL JCSTRESS_MMM ]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This specification activates the Johnson-Cook Stress model with a Mie-Gruneisen EOS and the Johnson-Cook failure model. This is a widely used material model, and fits for a range of materials can be found in the literature. Several options turn on and off slight modifications to the model, and the failure portion can be used or turned off. The failure model also includes an option to randomly perturb the failure strains for the model, permitting the adding of material non-heterogeneity into analyses.

The Johnson-Cook Stress model has a yield function described by:

$$\sigma = [A + B\epsilon_p^n][1 + C \ln \dot{\epsilon}^*][1 - T^{*m}] + \alpha P \tag{1.12}$$

Where ϵ_p is the equivalent plastic strain, $\dot{\epsilon}^* = \dot{\epsilon}/1.0\,\mathrm{sec}^{-1}$, P is the hydrostatic pressure, $T^* = (T - T_{room})/(T_{melt} - T_{room})$ where T refers to temperature, and A, B, C, n, m, and α are material constants. The stress can be capped to a user-specified maximum.

The strain rate dependence can also take on a power-law form, where the expression $[1 + C \ln \dot{\epsilon}^*]$ is replaced with $[\dot{\epsilon}^{*C}]$.

The Johnson-Cook Stress model also has the capability to compute material failure. Once failed, the model provides resistance only to hydrostatic pressure. Material failure occurs when the damage D is greater than 1.0. Note that a value of D less than 1.0 has no effect on the computed stresses in the model. D accumulates according to the equation:

$$D = \sum \left(\frac{\Delta \epsilon_p}{\epsilon_p^f} \right) \tag{1.13}$$

where $\Delta\epsilon_p$ is the increment of plastic strain over a time step, and ϵ_p^f is the failure strain. The failure strain is described by the expression:

$$\epsilon_p^f = [D_1 + D_2 \exp(D_3 \sigma^*)][1 + D_4 \ln \dot{\epsilon_p}][1 + D_5 T^*] \tag{1.14}$$

where σ^* is the mean pressure divided by the von Mises equivalent stress, T^* is the normalized temperature described earlier, $\dot{\epsilon}_p$ is the plastic strain rate, and D_1 through D_5 are material constants. Note that the failure strain for a material point changes if the loading or temperature changes.

For high tensile stresses, the failure strain is handled differently. In the original J-C failure model, the failure strain is capped at ϵ_{min}^f once the stress reaches σ_{spall}^*, which is defined as the user-specified σ_{spall} normalized by the von Mises stress. The transition to this cap starts at a normalized tensile stress of $\sigma^* > 1.5$, at which point it varies linearly to the cap values. Alternatively, a modified version accumulates damage for tensile pressures as:

$$D = \frac{\sum (\sigma^* - 1)^\lambda \Delta t}{K^*} \tag{1.15}$$

where λ and K^* are material constants. This is activated once the mean tensile pressure exceeds the threshold σ_{m0}.

Statistical variation of the failure parameters can also be added through this model. See below for the commands which activate this.

The command block starts with the input line:

```
BEGIN PARAMETERS FOR MODEL JCSTRESS_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL JCSTRESS_MMM]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- Only two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following command lines are required:
  - The yield stress of the material, shown as A in the equations above, is defined with the `YIELD STRESS` command line.
  - The initial density of the material is defined with the `INITIAL DENSITY` command line. Set this equal to the density specified with the `DENSITY` command line.
  - Extra linear artificial bulk viscosity can be defined with the `ART VIS CL` command line; this value should generally be set to zero.
  - Extra quadratic artificial bulk viscosity can be defined with the `ART VIS CQ` command line; this value should generally be set to zero.
  - The room temperature is defined with the `ROOM TEMPERATURE` command line.
  - The melt temperature is defined with the `MELT TEMPERATURE` command line.
  - The specific heat is defined with the `SPECIFIC HEAT` command line.
  - The hardening constant B is specified with the command `STRAIN HARD COEF`.
  - The hardening exponent n is specified with the command `STRAIN HARD EXP`.
  - The exponent on the temperature m is specified with the command `THERM SOFT EXP`.

- The term α is specified with the command `PRESS HARD COEF`.
- A limit on the yield stress can be specified using the `MAX STRESS` command line.
- The form of the rate dependence is chosen with the command `STRAIN RATE MODEL``â??choose LOG for the traditional form, and POWER for the power law version. In both cases, the material parameter :math:`C` which controls the rate effect is specified with the command line ``STRAIN RATE COEF`.
- The type of failure model is defined with the `JCF MODEL` command line. If the value is NONE, then no failure model is used. The original version of the Johnson-Cook Failure model with its original treatment of spall is chosen with the `ORIGINAL` keyword. The MODIFIED value chooses the modified version of the spall model. Specifying the value LAME chooses the modular failure model; commands for this model are explained below.

- The Johnson-Cook failure model parameters D_1, D_2, D_3, D_4, and D_5 are defined with their corresponding commands, each of which begin with the `JCF` command word.
- The spall cap for the original Johnson-Cook Failure model is specified with the commands `JCF PFAIL` and `JCF EFMIN` for the spall stress (σ_{spall}) and minimum failure strain (ϵ^f_{min}), respectively.
- The spall behavior for the modified Johnson-Cook Failure model is specified with the commands `JCF KSTAR` and `JCF LAMBDA` for K^* and λ, respectively. The command `JCF PFAIL` specifies the threshold mean tensile pressure (σ_{m0}) after which the spall model is used for failure.
- The command `JCF LFAIL` controls whether the stress will be decayed if a damage > 1.0 is reached. If this value is set to zero, no failure will occur, though damage will still be computed. A value of 1 will cause the stress to go to zero once damage > 1.0.
- A Weibull modulus-based variability is available through the `JCSTRESS_MMM` model. This capability is activated if the value given for the Weibull modulus using the command `JCF WM` is a value greater than zero. `JCF REFVOL` defines a representative element size, such as the average size of elements where failure is expected. The commands `JCF ICSEED` and `JCF ITSEED` serve as seeds for the random number generator.
- `JCSTRESS_MMM` permits the choice of two different implementations of the Mie-Gruneisen model. The command `MIEGRU FORM` chooses the version.
  - The Gruneisen parameter Gamma is defined with the `GRUN COEF` command line.
  - If `MIEGRU FORM` is chosen as `CUBIC`, then the cubic version of Mie-Gruneisen is chosen. The following commands are active:
    * The K_2 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K2` command line.

    * The K_3 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K3` command line.
  - If `MIEGRU FORM` is chosen as `USUP`, then the linear $U_s - U_p$ version of Mie-Gruneisen is chosen. The following commands are active:
    * The initial bulk sound speed is defined with the `MIEGRUN CSBULK` command line.
    * The slope of the $U_s - U_p$ relation (S) is defined with the `MIEGRUN SLOPE` command line.
- The maximum permitted tensile pressure is defined with the `MAX TENS PRESS` command line.
- `JCSTRESS_MMM` supports the modular failure model. The following is a description of the corresponding commands:
  - The command `CRITICAL FAILURE PARAMETER` is used to specify the critical failure parameter, used for modular failure.
  - The modular failure model multipliers are specified with the commands `PRESSURE MULTIPLIER`, `LODE ANGLE MULTIPLIER`, `TRIAXIALITY MULTIPLIER`, `RATE FAIL MULTIPLIER`, and `TEMPERATURE FAIL MULTIPLIER`. The definitions listed above show the supported and default (`INDEPENDENT`) options for these multipliers.
  - Additional parameters need to be specified for some modular failure multipliers:
    * If an `INDEPENDENT` multiplier is used, no additional parameters are necessary.
    * When the Wilkins model is used for the pressure multiplier, `WILKINS ALPHA` and `WILKINS PRESSURE` must be specified.
    * For the Lode angle multiplier, just the `WILKINS BETA` parameter is required for the Wilkins model.
    * When the Johnson-Cook model is used for the triaxiality multiplier, the D_1, D_2, and D_3 parameters must be specified. Each of these parameters are specified with the corresponding commands, beginning with the `JOHNSON COOK` command word. These parameters must be specified, even if they already have been specified beginning with the `JC` command word.
    * One must define D_4 with `JOHNSON COOK D4` when the Johnson-Cook model is used for the rate fail multiplier. This is even if `JC D4` has already been defined. Additionally, the reference rate must be specified with `REFERENCE RATE`.
    * The temperature fail multiplier requires D_5 when the Johnson-Cook model is used for it, specified with `JOHNSON COOK D5`. This is specified in this case, in addition to `JC D5`. Reference temperature and melting temperature are also required, defined with `REFERENCE TEMPERATURE` and `MELTING`

TEMPERATURE, respectively. One must define these parameters for this case in addition to ROOM TEMPERATURE and MELT TEMPERATURE described above.

* For multipliers that support user-defined functions (pressure, triaxiality, rate fail, and temperature fail), the functions need to be specified with the multiplier name followed by the FUNCTION command word.

Output variables available for this model are listed in Table Table 1.9. More information about this model is available in [11].

1.2.1.7 Johnson-Holmquist Ceramic Models

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL JH1CERAMIC_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    BULKING CNST = <real>BULKING_CNST
    DAMAGE CNST DP1 = <real>DAMAGE_CNST_DP1
    FSTRENGTH MAX = <real>FSTRENGTH_MAX
    FSTRENGTH SLOPE = <real>FSTRENGTH_SLOPE
    MAX FAIL STRAIN = <real>MAX_FAIL_STRAIN
    PRESS COEF K2 = <real>PRESS_COEF_K2
    PRESS COEF K3 = <real>PRESS_COEF_K3
    STRAIN RATE COEF = <real>STRAIN_RATE_COEF
    STRENGTH CNST P1 = <real>STRENGTH_CNST_P1
    STRENGTH CNST P2 = <real>STRENGTH_CNST_P2
    STRENGTH CNST S1 = <real>STRENGTH_CNST_S1
    STRENGTH CNST S2 = <real>STRENGTH_CNST_S2
    MAX TENS PRESS = <real>MAX_TENS_PRESS
  END [PARAMETERS FOR MODEL JH1CERAMIC_MMM]

  BEGIN PARAMETERS FOR MODEL JH2CERAMIC_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
```

(continues on next page)

(continued from previous page)

```
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    BULKING CNST = <real>BULKING_CNST
    DAMAGE COEF D1 = <real>DAMAGE_COEF_D1
    DAMAGE EXP D2 = <real>DAMAGE_EXP_D2
    FSTRENGTH COEF B = <real>FSTRENGTH_COEF_B
    FSTRENGTH EXP M = <real>FSTRENGTH_EXP_M
    FSTRENGTH MAX NORM = <real>FSTRENGTH_MAX_NORM
    HEL = <real>HEL
    MIN FAIL STRAIN = <real>MIN_FAIL_STRAIN
    PRESS COEF K2 = <real>PRESS_COEF_K2
    PRESS COEF K3 = <real>PRESS_COEF_K3
    STRAIN RATE COEF = <real>STRAIN_RATE_COEF
    STRENGTH COEF A = <real>STRENGTH_COEF_A
    STRENGTH EXP N = <real>STRENGTH_EXP_N
    MAX TENS PRESS = <real>MAX_TENS_PRESS
  END [PARAMETERS FOR MODEL JH2CERAMIC_MMM]

  BEGIN PARAMETERS FOR MODEL JH3CERAMIC_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    BULKING CNST = <real>BULKING_CNST
    DAMAGE COEF D1 = <real>DAMAGE_COEF_D1
    DAMAGE EXP D2 = <real>DAMAGE_EXP_D2
    FSTRENGTH COEF B = <real>FSTRENGTH_COEF_B
    FSTRENGTH EXP M = <real>FSTRENGTH_EXP_M
    FSTRENGTH MAX NORM = <real>FSTRENGTH_MAX_NORM
    HEL = <real>HEL
    MIN FAIL STRAIN = <real>MIN_FAIL_STRAIN
    PRESS COEF K2 = <real>PRESS_COEF_K2
    PRESS COEF K3 = <real>PRESS_COEF_K3
    STRAIN RATE COEF = <real>STRAIN_RATE_COEF
    STRENGTH COEF A = <real>STRENGTH_COEF_A
    STRENGTH EXP N = <real>STRENGTH_EXP_N
    MAX TENS PRESS = <real>MAX_TENS_PRESS
   END [PARAMETERS FOR MODEL JH3CERAMIC_MMM ]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This specification activates the Johnson-Holmquist Ceramic models 1 through 3. The three models differ slightly in how they handle failure. More information is available in [11].

The command block starts with the input line:

```
BEGIN PARAMETERS FOR MODEL JH#CERAMIC_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL JH#CERAMIC_MMM]
```

where # is 1, 2, or 3.

In the above command blocks:

- The density of the material is defined with the DENSITY command line.
- Only two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the YOUNGS MODULUS command line.
  - Poisson's ratio is defined with the POISSONS RATIO command line.
  - The bulk modulus is defined with the BULK MODULUS command line.
  - The shear modulus is defined with the SHEAR MODULUS command line.
  - Lambda is defined with the LAMBDA command line.
- The following command lines are required:
  - The yield stress of the material is defined with the YIELD STRESS command line.
  - The initial density of the material is defined with the INITIAL DENSITY command line. Set this equal to the density specified with the DENSITY command line.
  - The maximum permitted tensile pressure is defined with the MAX TENS PRESS command line.
  - The remaining command lines are described in [11].

Output variables available for these models are listed in Table 1.7. More information about these models is available in [11]] and [[9].

1.2.1.8 Johnson-Holmquist-Beissel Ceramic Models

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL JHB1CERAMIC_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
```

(continues on next page)

(continued from previous page)

```
  POISSONS RATIO = <real>poissons_ratio
  SHEAR MODULUS = <real>shear_modulus
  LAMBDA = <real>lambda
  YIELD STRESS = <real>yield_stress
  INIT DENSITY = <real>init_density
  BULKING CNST = <real>BULKING_CNST
  DAMAGE COEF D1 = <real>DAMAGE_COEF_D1
  DAMAGE EXP N = <real>DAMAGE_EXP_N
  FSTRENGTH CNST PF = <real>FSTRENGTH_CNST_PF
  FSTRENGTH CNST SF = <real>FSTRENGTH_CNST_SF
  FSTRENGTH MAX = <real>FSTRENGTH_MAX
  MAX FAIL STRAIN = <real>MAX_FAIL_STRAIN
  PRESS COEF K2 = <real>PRESS_COEF_K2
  PRESS COEF K3 = <real>PRESS_COEF_K3
  STRAIN RATE COEF = <real>STRAIN_RATE_COEF
  STRENGTH CNST PI = <real>STRENGTH_CNST_PI
  STRENGTH CNST SI = <real>STRENGTH_CNST_SI
  STRENGTH MAX = <real>STRENGTH_MAX
  MAX TENS PRESS = <real>MAX_TENS_PRESS
END [PARAMETERS FOR MODEL JHB1CERAMIC_MMM ]

BEGIN PARAMETERS FOR MODEL JHB2CERAMIC_MMM
  TWO MU = <real>two_mu
  YOUNGS MODULUS = <real>youngs_modulus
  BULK MODULUS = <real>bulk_modulus
  POISSONS RATIO = <real>poissons_ratio
  SHEAR MODULUS = <real>shear_modulus
  LAMBDA = <real>lambda
  YIELD STRESS = <real>yield_stress
  INIT DENSITY = <real>init_density
  BULKING CNST = <real>BULKING_CNST
  DAMAGE COEF D1 = <real>DAMAGE_COEF_D1
  DAMAGE EXP N = <real>DAMAGE_EXP_N
  FSTRENGTH CNST PF = <real>FSTRENGTH_CNST_PF
  FSTRENGTH CNST SF = <real>FSTRENGTH_CNST_SF
  FSTRENGTH MAX = <real>FSTRENGTH_MAX
  HYSTERESIS CNST = <real>HYSTERESIS_CNST
  MAX FAIL STRAIN = <real>MAX_FAIL_STRAIN
  PHASE TRAN P1 = <real>PHASE_TRAN_P1
  PHASE TRAN P2 = <real>PHASE_TRAN_P2
  PHASE2 KP1 = <real>PHASE2_KP1
  PHASE2 KP2 = <real>PHASE2_KP2
  PHASE2 KP3 = <real>PHASE2_KP3
  PHASE2 UPZERO = <real>PHASE2_UPZERO
  PRESS COEF K2 = <real>PRESS_COEF_K2
```

(continues on next page)

(continued from previous page)

```
    PRESS COEF K3 = <real>PRESS_COEF_K3
    STRAIN RATE COEF = <real>STRAIN_RATE_COEF
    STRENGTH CNST PI = <real>STRENGTH_CNST_PI
    STRENGTH CNST SI = <real>STRENGTH_CNST_SI
    STRENGTH MAX = <real>STRENGTH_MAX
    MAX TENS PRESS = <real>MAX_TENS_PRESS
  END [PARAMETERS FOR MODEL JHB2CERAMIC_MMM]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This specification activates the Johnson-Holmquist-Beissel Ceramic models 1 and 2. The two models differ slightly in how they handle failure. More information is available in [11].

The command block starts with the input line:

```
BEGIN PARAMETERS FOR MODEL JHB#CERAMIC_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL JHB#CERAMIC_MMM]
```

where # is either 1 or 2.

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- Only two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following command lines are required:
  - The initial density of the material is defined with the `INITIAL DENSITY` command line. Set this equal to the density specified with the `DENSITY` command line.
  - The maximum permitted tensile pressure is defined with the `MAX TENS PRESS` command line.
  - The remaining command lines are described in [11].

Output variables available for these models are listed in Table 1.8. More information about these models is available in [11]] and [[9].

1.2.1.9 Mechanical Threshold Stress (MTS) Strength Model with Mie-Gruneisen EOS

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL MTSSTRESS_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    ABS ZERO TEMP = <real>ABS_ZERO_TEMP
    INIT TEMPERATURE = <real>INIT_TEMPERATURE
    MELT TEMPERATURE = <real>MELT_TEMPERATURE
    SPECIFIC HEAT = <real>SPECIFIC_HEAT
    ABS ZERO SHRMOD SM0 = <real>ABS_ZERO_SHRMOD_SM0
    BURGVEC MAG = <real>BURGVEC_MAG
    CNST ALPHA = <real>CNST_ALPHA
    CNST BOLTZ = <real>CNST_BOLTZ
    CNST CAPA = <real>CNST_CAPA
    CNST PINV = <real>CNST_PINV
    CNST PINVI = <real>CNST_PINVI
    CNST PINVS = <real>CNST_PINVS
    CNST QINV = <real>CNST_QINV
    CNST QINVI = <real>CNST_QINVI
    CNST QINVS = <real>CNST_QINVS
    DISLOC CNST HF0 = <real>DISLOC_CNST_HF0
    DISLOC CNST HF1 = <real>DISLOC_CNST_HF1
    DISLOC CNST HF2 = <real>DISLOC_CNST_HF2
    DISLOC CNST SIGA = <real>DISLOC_CNST_SIGA
    DISLOC CNST SIGI = <real>DISLOC_CNST_SIGI
    DISLOC CNST SIGS = <real>DISLOC_CNST_SIGS
    INIT STATE VAR SIG0 = <real>INIT_STATE_VAR_SIG0
    NORM ACT ENRGY G0 = <real>NORM_ACT_ENRGY_G0
    NORM ACT ENRGY G0I = <real>NORM_ACT_ENRGY_G0I
    NORM ACT ENRGY G0S = <real>NORM_ACT_ENRGY_G0S
    REF STN RAT EDOT0 = <real>REF_STN_RAT_EDOT0
    REF STN RAT EDOTI = <real>REF_STN_RAT_EDOTI
    REF STN RAT EDOTS = <real>REF_STN_RAT_EDOTS
    REF STN RAT EDOTS0 = <real>REF_STN_RAT_EDOTS0
    SAT TH STS SIGS0 = <real>SAT_TH_STS_SIGS0
    SHRMOD CNST SM1 = <real>SHRMOD_CNST_SM1
```

(continues on next page)

(continued from previous page)

```
    SHRMOD CNST SM2 = <real>SHRMOD_CNST_SM2
    GRUN COEF = <real>GRUN_COEF
    MIEGRU COEF K2 = <real>MIEGRU_COEF_K2
    MIEGRU COEF K3 = <real>MIEGRU_COEF_K3
    MAX TENS PRESS = <real>MAX_TENS_PRESS

    FAILURE MODEL = NONE | MODULAR_FAILURE (NONE)

    CRITICAL FAILURE PARAMETER = <real> d_crit
    #
    #  MODULAR_FAILURE Failure model definitions
    #
    PRESSURE MULTIPLIER         = PRESSURE_INDEPENDENT | WILKINS
                                  | USER_DEFINED (PRESSURE_INDEPENDENT)
    LODE ANGLE MULTIPLIER       = LODE_ANGLE_INDEPENDENT |
                                  WILKINS (LODE_ANGLE_INDEPENDENT)
    TRIAXIALITY MULTIPLIER      = TRIAXIALITY_INDEPENDENT | JOHNSON_
↪COOK
                                  | USER_DEFINED (TRIAXIALITY_
↪INDEPENDENT)
    RATE FAIL MULTIPLIER        = RATE_INDEPENDENT | JOHNSON_COOK
                                  | USER_DEFINED (RATE_INDEPENDENT)
    TEMPERATURE FAIL MULTIPLIER = TEMPERATURE_INDEPENDENT | JOHNSON_
↪COOK
                                  | USER_DEFINED (TEMPERATURE_
↪INDEPENDENT)
    #
    # Individual multiplier definitions
    #
    PRESSURE MULTIPLIER = WILKINS
    WILKINS ALPHA       = <real> alpha
    WILKINS PRESSURE    = <real> B
    #
    PRESSURE MULTIPLIER = USER_DEFINED
    PRESSURE MULTIPLIER FUNCTION = <string> pressure_multiplier_fun_
↪name
    #
    LODE ANGLE MULTIPLIER = WILKINS
    WILKINS BETA          = <real> beta
    #
    TRIAXIALITY MULTIPLIER = JOHNSON_COOK
    JOHNSON COOK D1        = <real> D_1
    JOHNSON COOK D2        = <real> D_2
    JOHNSON COOK D3        = <real> D_3
    #
```

(continues on next page)

(continued from previous page)

```
    TRIAXIALITY MULTIPLIER = USER_DEFINED
    TRIAXIALITY MULTIPLIER FUNCTION = <string> triax_multiplier_fun_
↪name
    #
    RATE FAIL MULTIPLIER = JOHNSON_COOK
    JOHNSON COOK D4      = <real> D_4
    REFERENCE RATE       = <real> dot_epsilon_0
    #
    RATE FAIL MULTIPLIER = USER_DEFINED
    RATE FAIL MULTIPLIER FUNCTION = <string> rate_fail_multiplier_fun_
↪name
    #
    TEMPERATURE FAIL MULTIPLIER = JOHNSON_COOK
    JOHNSON COOK D5             = <real> D_5
    REFERENCE TEMPERATURE       = <real> T_ref
    MELTING TEMPERATURE         = <real> T_melt
    #
    TEMPERATURE FAIL MULTIPLIER          = USER_DEFINED
    TEMPERATURE FAIL MULTIPLIER FUNCTION = <string> temp_multiplier_
↪fun_name
    #
  END [PARAMETERS FOR MODEL MTSSTRESS_MMM]

END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This specification activates the Mechanical Threshold Stress (MTS) model with a cubic Mie-Gruneisen EOS. The MTS model has a yield function defined by:

$$\sigma = \hat{\sigma}_a + \frac{G}{G_0}(s_{th}\hat{\sigma} + s_{th,i}\hat{\sigma}_i + s_{th,s}\hat{\sigma}_s) \tag{1.16}$$

where $\hat{\sigma}$ is the mechanical threshold stress (defined below), $\hat{\sigma}_a$, $\hat{\sigma}_i$, and $\hat{\sigma}_s$ are constants representing dislocation interactions corresponding to long-range barriers, interstitial atoms, and solute atoms, and G_0 is the shear modulus at absolute zero. The shear modulus at other temperatures is defined as:

$$G = G_0 - \frac{b_1}{\exp(b_2/T) - 1} \tag{1.17}$$

where b_1 and b_2 are material constants and T is the absolute temperature.

The s_{th} terms have the general form:

$$s_{th} = \left[1 - \left(\frac{kT\ln(\dot{\epsilon}_0/\dot{\epsilon})}{Gb^3 g_0}\right)^{\frac{1}{q}}\right]^{\frac{1}{p}} \tag{1.18}$$

where k is the Boltzmann constant, b is the magnitude of the Burger's vector, g_0 is a normalized activation energy, $\dot{\epsilon}_0$ is a reference strain rate, and p and q are exponential constants. For $s_{th,i}$ and $s_{th,s}$, the equation is identical but with different constants.

The update of the mechanical threshold stress $\hat{\sigma}$ is governed by:

$$\hat{\sigma}_{t+\Delta t} = \hat{\sigma}_t + \frac{\delta\hat{\sigma}}{\delta\epsilon_p}(\dot{\epsilon}_p \Delta t) \tag{1.19}$$

where:

$$\frac{\delta\hat{\sigma}}{\delta\epsilon_p} = \Theta_0 \left[1 - \frac{\tanh\left(\alpha \frac{\hat{\sigma}}{\hat{\sigma}_s}\right)}{\tanh(\alpha)} \right] \tag{1.20}$$

$$\Theta_0 = a_0 + a_1 \ln(\dot{\epsilon}) + a_2 \sqrt{\dot{\epsilon}} \tag{1.21}$$

$$\hat{\sigma}_s = \hat{\sigma}_{so} \left(\frac{\dot{\epsilon}}{\dot{\epsilon}_{so}} \right)^{kT/Gb^3 A} \tag{1.22}$$

where A, α, a_0, a_1, and a_2 are material constants, $\hat{\sigma}_{so}$ is the saturation threshold stress, and $\dot{\epsilon}_{so}$ is a reference strain rate.

The command block starts with the input line:

```
BEGIN PARAMETERS FOR MODEL MTSSTRESS_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL MTSSTRESS_MMM]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- Only two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following command lines are required:
  - The yield stress of the material is defined with the `YIELD STRESS` command line.
  - The initial density of the material is defined with the `INITIAL DENSITY` command line. Set this equal to the density specified with the `DENSITY` command line.
  - The temperature at absolute zero is defined with the `ABS ZERO TEMP` command line.

- The melt temperature is defined with the `MELT TEMPERATURE` command line.
- The specific heat is defined with the `SPECIFIC HEAT` command line.
- The shear modulus at absolute zero (G_0) is defined with the `ABS ZERO SHRMOD SM0` command line.
- The magnitude of the Burgers vector (b) is defined with the `BURGVEC MAG` command line.
- The material constant α is defined with the `CNST ALPHA` command line.
- The material constant A is defined with the `CNST CAPA` command line.
- The material constant a_0 is defined with the `DISLOC CNST HF0` command line.
- The material constant a_1 is defined with the `DISLOC CNST HF1` command line.
- The material constant a_2 is defined with the `DISLOC CNST HF2` command line.
- The Boltzmann constant (k) is defined with the `CNST BOLTZ` command line.
- The dislocation interaction constant $\hat{\sigma}_a$ is defined with the `DISLOC CNST SIGA` command line.
- The dislocation interaction constant $\hat{\sigma}_i$ is defined with the `DISLOC CNST SIGI` command line.
- The dislocation interaction constant $\hat{\sigma}_s$ is defined with the `DISLOC CNST SIGS` command line.
- The $1/p$ exponent in the equation for s_{th} is defined with the `CNST PINV` command line.
- The $1/p$ exponent in the equation for $s_{th,i}$ is defined with the `CNST PINVI` command line.
- The $1/p$ exponent in the equation for $s_{th,s}$ is defined with the `CNST PINVS` command line.
- The $1/q$ exponent in the equation for s_{th} is defined with the `CNST QINV` command line.
- The $1/q$ exponent in the equation for $s_{th,i}$ is defined with the `CNST QINVI` command line.
- The $1/q$ exponent in the equation for $s_{th,s}$ is defined with the `CNST QINVS` command line.
- The g_0 value in the equation for s_{th} is defined with the `NORM ACT ENRGY G0` command line.
- The g_0 value in the equation for $s_{th,i}$ is defined with the `NORM ACT ENRGY G0I` command line.

- The g_0 value in the equation for $s_{th,s}$ is defined with the `NORM ACT ENRGY G0S` command line.
- The $\dot{\epsilon}_0$ value in the equation for s_{th} is defined with the `REF STN RAT EDOT0` command line.
- The $\dot{\epsilon}_0$ value in the equation for $s_{th,i}$ is defined with the `REF STN RAT EDOTI` command line.
- The $\dot{\epsilon}_0$ value in the equation for $s_{th,s}$ is defined with the `REF STN RAT EDOTS` command line.
- The initial value for the mechanical threshold stress $\hat{\sigma}$ is defined with the `INIT STATE VAR SIG0` command line.
- The value for $\hat{\sigma}_{so}$ in the equation for the saturation stress $\hat{\sigma}_s$ is defined with the `SAT TH STS SIGS0` command line.
- The value for $\dot{\epsilon}_{so}$ in the equation for the saturation stress $\hat{\sigma}_s$ is defined with the `REF STN RAT EDOTS0` command line.
- The material constant b_1 in the temperature shear modulus equation is defined with the `SHRMOD CNST SM1` command line.
- The material constant b_2 in the temperature shear modulus equation is defined with the `SHRMOD CNST SM2` command line.
- The Gruneisen parameter Gamma is defined with the `GRUN COEF` command line.
- The K_2 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K2` command line.
- The K_3 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K3` command line.
- The maximum permitted tensile pressure is defined with the `MAX TENS PRESS` command line.
- The type of failure model is defined with the `FAILURE MODEL` command line. If the value is NONE, then no failure model is used. Specifying the value MODULAR FAILURE chooses the modular failure model; commands for this model are explained below.

- **Modular Failure Model**: - The command `CRITICAL FAILURE PARAMETER` is used to specify the critical failure parameter, used for modular failure. - The modular failure model multipliers are specified with the commands `PRESSURE MULTIPLIER`, `LODE ANGLE MULTIPLIER`, `TRIAXIALITY MULTIPLIER`, `RATE FAIL MULTIPLIER`, and `TEMPERATURE FAIL MULTIPLIER`. The definitions listed above show the supported and default (`INDEPENDENT`) options for these multipliers.

More information about this model is available in [11].

1.2.1.10 Mechanical Threshold Stress Strength Model with Mie-Gruneisen EOS and TEPLA Damage Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL TEPLA_MTSSTRESS_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    ABS ZERO TEMP = <real>ABS_ZERO_TEMP
    INIT TEMPERATURE = <real>INIT_TEMPERATURE
    MELT TEMPERATURE = <real>MELT_TEMPERATURE
    SPECIFIC HEAT = <real>SPECIFIC_HEAT
    ABS ZERO SHRMOD SM0 = <real>ABS_ZERO_SHRMOD_SM0
    ALPHA11 = <real>ALPHA11
    ALPHA21 = <real>ALPHA21
    ALPHA22 = <real>ALPHA22
    ALPHA31 = <real>ALPHA31
    ALPHA32 = <real>ALPHA32
    ALPHA33 = <real>ALPHA33
    ALPHA41 = <real>ALPHA41
    ALPHA42 = <real>ALPHA42
    ALPHA43 = <real>ALPHA43
    ALPHA44 = <real>ALPHA44
    ALPHA51 = <real>ALPHA51
    ALPHA52 = <real>ALPHA52
    ALPHA53 = <real>ALPHA53
    ALPHA54 = <real>ALPHA54
    ALPHA55 = <real>ALPHA55
    BURGVEC MAG = <real>BURGVEC_MAG
    CNST ALPHA = <real>CNST_ALPHA
    CNST BOLTZ = <real>CNST_BOLTZ
    CNST CAPA = <real>CNST_CAPA
    CNST PINV = <real>CNST_PINV
    CNST PINVI = <real>CNST_PINVI
    CNST PINVS = <real>CNST_PINVS
    CNST QINV = <real>CNST_QINV
    CNST QINVI = <real>CNST_QINVI
    CNST QINVS = <real>CNST_QINVS
    DISLOC CNST HF0 = <real>DISLOC_CNST_HF0
```

(continues on next page)

(continued from previous page)

```
    DISLOC CNST HF1 = <real>DISLOC_CNST_HF1
    DISLOC CNST HF2 = <real>DISLOC_CNST_HF2
    DISLOC CNST SIGA = <real>DISLOC_CNST_SIGA
    DISLOC CNST SIGI = <real>DISLOC_CNST_SIGI
    DISLOC CNST SIGS = <real>DISLOC_CNST_SIGS
    E11 = <real>E11
    E21 = <real>E21
    E22 = <real>E22
    E31 = <real>E31
    E32 = <real>E32
    E33 = <real>E33
    E41 = <real>E41
    E42 = <real>E42
    E43 = <real>E43
    E44 = <real>E44
    E51 = <real>E51
    E52 = <real>E52
    E53 = <real>E53
    E54 = <real>E54
    E55 = <real>E55
    E61 = <real>E61
    E62 = <real>E62
    E63 = <real>E63
    E64 = <real>E64
    E65 = <real>E65
    E66 = <real>E66
    FAIL POR PHIF = <real>FAIL_POR_PHIF
    FAIL SURF GAMA0 = <real>FAIL_SURF_GAMA0
    FAIL SURF GAMA1 = <real>FAIL_SURF_GAMA1
    FAIL SURF GAMA2 = <real>FAIL_SURF_GAMA2
    ICOMP = <real>ICOMP
    INIT POR PHI0 = <real>INIT_POR_PHI0
    INIT STATE VAR SIG0 = <real>INIT_STATE_VAR_SIG0
    LENGTH SCALE = <real>LENGTH_SCALE
    NORM ACT ENRGY G0 = <real>NORM_ACT_ENRGY_G0
    NORM ACT ENRGY G0I = <real>NORM_ACT_ENRGY_G0I
    NORM ACT ENRGY G0S = <real>NORM_ACT_ENRGY_G0S
    ORTHO = <real>ORTHO
    REF STN RAT EDOT0 = <real>REF_STN_RAT_EDOT0
    REF STN RAT EDOTI = <real>REF_STN_RAT_EDOTI
    REF STN RAT EDOTS = <real>REF_STN_RAT_EDOTS
    REF STN RAT EDOTS0 = <real>REF_STN_RAT_EDOTS0
    RODRIGUES ANGLE = <real>RODRIGUES_ANGLE
    RODRIGUES X = <real>RODRIGUES_X
    RODRIGUES Y = <real>RODRIGUES_Y
```

(continues on next page)

(continued from previous page)

```
    RODRIGUES Z = <real>RODRIGUES_Z
    SAT TH STS SIGS0 = <real>SAT_TH_STS_SIGS0
    SHRMOD CNST SM1 = <real>SHRMOD_CNST_SM1
    SHRMOD CNST SM2 = <real>SHRMOD_CNST_SM2
    VOID GROW PAR QG1 = <real>VOID_GROW_PAR_QG1
    VOID GROW PAR QG2 = <real>VOID_GROW_PAR_QG2
    VOID GROW PAR QG3 = <real>VOID_GROW_PAR_QG3
    GRUN COEF = <real>GRUN_COEF
    MIEGRU COEF K2 = <real>MIEGRU_COEF_K2
    MIEGRU COEF K3 = <real>MIEGRU_COEF_K3
  END [PARAMETERS FOR MODEL TEPLA_MTSSTRESS_MMM ]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This specification activates the Mechanical Threshold Stress (MTS) strength model with a cubic Mie-Gruneisen EOS and the TEPLA continuum level damage model. This model is an extension of the standard MTS model, as described in Section 1.2.1.9. The extensions provide an ability to initialize the porosity, pressure, failure porosity, flow stress, rotation, and stretch arrays and the specification of an orthotropic yield function. They also modify the MTS model to include the effect of evolving porosity (void growth) through an extended Gurson model. More information on this model is available in [11]] and [[20].

The command block starts with the input line:

```
BEGIN PARAMETERS FOR MODEL TEPLA_MTSSTRESS_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL TEPLA_MTSSTRESS_MMM]
```

Most of the commands for this material are identical to those defined in Section 1.2.1.9. In addition:

- The command `ORTHO` specifies that the material is orthotropic if set to 1, or isotropic if set to 0.
- The terms described by the commands `ALPHA11` through `ALPHA55` define the plastic shape tensor components.
- The terms described by the commands `E11` through `E66` represent the elastic stiffness tensor for an orthotropic material.
- The Rodrigues vector for the orthotropic yield surface is defined by the commands `RODRIGUES [X|Y|Z].`
- The Rodrigues angle is the angle of rotation around the Rodrigues vector, and is defined by the command `RODRIGUES ANGLE`.
- The initial porosity is defined by the command `INIT POR PHI0`.

- The final porosity at failure is given by the command `FAIL POR PHIF`.
- The command `ICOMP` toggles pore growth; if it is 0, then pores can grow, whereas if it is 1, pores do not grow.
- The commands `VOID GROW PAR QG[1|2|3]` define the coefficients for the Tvergaard porosity evolution equation.
- The length scale for the over-stress formulation is specified by the command `LENGTH SCALE`.
- The commands `FAIL SURF GAMA[0|1|2]` define the material constants in the expression for the failure strain.

More information about this model is available in [11].

1.2.1.11 Zerilli-Armstrong Strength Model for BCC Metals with Mie-Gruneisen EOS

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL ZABCCSTRESS_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    ABS ZERO TEMP = <real>ABS_ZERO_TEMP
    INIT TEMPERATURE = <real>INIT_TEMPERATURE
    SPECIFIC HEAT = <real>SPECIFIC_HEAT
    STRAIN HARD COEF C5 = <real>STRAIN_HARD_COEF_C5
    STRAIN HARD EXP N = <real>STRAIN_HARD_EXP_N
    STRAIN RATE COEF C1 = <real>STRAIN_RATE_COEF_C1
    STRAIN RATE COEF C4 = <real>STRAIN_RATE_COEF_C4
    THERM SOFT COEF C3 = <real>THERM_SOFT_COEF_C3
    YIELD STRESS C0 = <real>YIELD_STRESS_C0
    GRUN COEF = <real>GRUN_COEF
    MIEGRU COEF K2 = <real>MIEGRU_COEF_K2
    MIEGRU COEF K3 = <real>MIEGRU_COEF_K3
    MAX TENS PRESS = <real>MAX_TENS_PRESS
  END [PARAMETERS FOR MODEL ZABCCSTRESS_MMM]
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
```

This specification activates the Zerilli-Armstrong strength model for BCC metals with a Mie-Gruneisen EOS. The expression for the yield function of this model is:

$$\sigma = C_0 + C_1 \exp(-C_3 T + C_4 T \ln \dot{\epsilon}) + C_5 \epsilon_p^n \tag{1.23}$$

where ϵ_p is the equivalent plastic strain, T is the absolute temperature, $\dot{\epsilon}$ is the equivalent total strain rate, and C_0, C_1, C_3, C_4, C_5, and n are material constants.

The pressure response is described by a cubic Mie-Gruneisen model, see equation (1.5) for more details.

The command block starts with the input line:

```
BEGIN PARAMETERS FOR MODEL ZABCCSTRESS_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL ZABCCSTRESS_MMM]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- Only two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following command lines are required:
  - The yield stress of the material is defined with the `YIELD STRESS` command line.
  - The initial density of the material is defined with the `INITIAL DENSITY` command line. Set this equal to the density specified with the `DENSITY` command line.
  - The temperature at absolute zero is defined with the `ABS ZERO TEMP` command line.
  - The specific heat is defined with the `SPECIFIC HEAT` command line.
  - The material constants C_0, C_1, C_3, C_4, C_5, and n are defined with the corresponding command lines above.
  - The Gruneisen parameter Gamma is defined with the `GRUN COEF` command line.

- The K_2 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K2` command line.
- The K_3 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K3` command line.
- The maximum permitted tensile pressure is defined with the `MAX TENS PRESS` command line.

More information about this model is available in [11].

1.2.1.12 Zerilli-Armstrong Strength Model for FCC Metals with Mie-Gruneisen EOS

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
 BEGIN PARAMETERS FOR MODEL ZAFCCSTRESS_MMM
    TWO MU = <real>two_mu
    YOUNGS MODULUS = <real>youngs_modulus
    BULK MODULUS = <real>bulk_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    INIT DENSITY = <real>init_density
    ABS ZERO TEMP = <real>ABS_ZERO_TEMP
    INIT TEMPERATURE = <real>INIT_TEMPERATURE
    SPECIFIC HEAT = <real>SPECIFIC_HEAT
    STRAIN HARD COEF C2 = <real>STRAIN_HARD_COEF_C2
    STRAIN HARD EXP N = <real>STRAIN_HARD_EXP_N
    STRAIN RATE COEF C4 = <real>STRAIN_RATE_COEF_C4
    THERM SOFT COEF C3 = <real>THERM_SOFT_COEF_C3
    YIELD STRESS C0 = <real>YIELD_STRESS_C0
    GRUN COEF = <real>GRUN_COEF
    MIEGRU COEF K2 = <real>MIEGRU_COEF_K2
    MIEGRU COEF K3 = <real>MIEGRU_COEF_K3
    MAX TENS PRESS = <real>MAX_TENS_PRESS
  END [ PARAMETERS FOR MODEL ZAFCCSTRESS_MMM ]
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
```

This specification activates the Zerilli-Armstrong strength model for FCC metals with a Mie-Gruneisen EOS. The expression for the yield function of this model is:

$$\sigma = C_0 + C_2\epsilon_p^n \exp(-C_3T + C_4T \ln \dot{\epsilon}) \tag{1.24}$$

where ϵ_p is the equivalent plastic strain, T is the absolute temperature, $\dot{\epsilon}$ is the equivalent total strain rate, and C_0, C_2, C_3, C_4, and n are material constants.

The pressure response is described by a cubic Mie-Gruneisen model, see equation (1.5) for more details.

The command block starts with the input line:

```
BEGIN PARAMETERS FOR MODEL ZAFCCSTRESS_MMM
```

and terminates with an input line of the following form:

```
END [PARAMETERS FOR MODEL ZAFCCSTRESS_MMM]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- Only two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following command lines are required:
  - The yield stress of the material is defined with the `YIELD STRESS` command line.
  - The initial density of the material is defined with the `INITIAL DENSITY` command line. Set this equal to the density specified with the `DENSITY` command line.
  - The temperature at absolute zero is defined with the `ABS ZERO TEMP` command line.
  - The specific heat is defined with the `SPECIFIC HEAT` command line.
  - The material constants C_0, C_2, C_3, C_4, and n are defined with the corresponding command lines above.
  - The Gruneisen parameter Gamma is defined with the `GRUN COEF` command line.
  - The K_2 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K2` command line.
  - The K_3 parameter for the MMM cubic Mie-Gruneisen model is defined with the `MIEGRUN COEF K3` command line.
  - The maximum permitted tensile pressure is defined with the `MAX TENS PRESS` command line.

More information about this model is available in [11].

1.2.2 CTH Model Specifications

This section describes material models that have been ported from CTH to the LAME material library [24]. Because of the ITAR export-control restrictions on these models, they are maintained separately from the standard LAME material library and only linked in with `Presto_ITAR`.

> **Warning:** Support for CTH material models in `Presto_ITAR` is currently at an experimental level. As such, not all features may be fully implemented or tested, and the analyst should use these models with caution.

Note: The algorithms that apply when these energy-dependent models are in use are currently in a state of flux as they are being upgraded to the state-of-the-art. This transformation has currently been applied only to the midpoint-increment uniform-gradient hexahedron element. Attempting to use these models with any other element will likely result in code failure.

Implementation of the CTH material models departs from the typical behavior found for other material models present in `Presto_ITAR`. Generally, this allows the CTH models to be more flexible in the material behaviors they can represent, particularly for high strain rate, energy-dependent materials. The main differences are in the treatment of the energy update, modularity, and parameter specification.

For energy-dependent material models, such as those from this section, Section 1.2.1, and Section 1.2.3, the internal energy is updated using a second-order, implicit equation, see [27]. For the traditional `Presto_ITAR` models of Section 1.2.1 and Section 1.2.3, the energy update is performed as part of the material model. Additionally, all the models assume materials behave under the Mie-Gruneisen assumption that pressure is linearly dependent upon the internal energy. This allows these models to explicitly solve the implicit energy equation. While this provides for an easy solution, it limits the types of material behavior that can be modeled. The CTH models break from the Mie-Gruneisen assumption, allowing an arbitrary dependence of the pressure on the internal energy. This motivates several changes in how the elements treat these materials.

The energy update equation is a function of the host code in that its form and method of solution are code dependent. From a theoretical perspective, a material has no knowledge of such an equation. Additionally, for portability between codes, a material model should not solve such an equation since it would possibly have to be different for every code in which it was used. For this reason, the energy equation update was not pushed into the CTH material models. Instead, it is computed in the element itself. In the future, other material models from Section 1.2.1 and Section 1.2.3 may also have the energy update extracted from them, leading to less code duplication, better consistency across models, and better maintainability.

Not only was the energy update extracted from the material models for the CTH models, but the assumption of a Mie-Gruneisen form also had to be removed. This requires one to perform an iterative solve of the energy equation to be self-consistent, since the explicit solution is no longer

viable. The initial guess to the solution is based upon a predictor method for the hydrodynamic work. Later iterations include a fully implicit solve of the hydrodynamic and deviatoric work. For information on controlling the iterative solution of the energy equation, see Section 1.3.1.1.

The CTH models also depart from the other Presto_ITAR material models in that they adopt the concept of modularity. Typically, a solid might have an equation of state model and a yield model. Models from Section 1.2.1 and Section 1.2.3 explicitly couple these models together. Thus, if one wants to use an already implemented yield model with a new equation of state, then one has to write a new material model which couples them together. On the other hand, the CTH models are modular (although not completely) in that if a given model adheres to a certain interface, it may be used as a drop-in replacement for other models using the same interface. Thus, only the new submodel has to be implemented. Currently, there is a single implementation of a CTH modular model in Presto_ITAR, the CTH_EP model of Section 1.2.2.3.

One side effect of the modularity concept is that not all models compute a stress. Those that do not cannot be called directly from an element, and hence cannot be used as the material model for an element. See, for example, the CTH_JO model of Section 1.2.2.4. On the other hand, equation of state models, such as the CTH_MGR model of Section 1.2.2.1, do compute a stress and so they can not only be used as a submodel in a modular model such as CTH_EP, but may also be used directly as an element material model.

Parameter parsing behavior has also been modified from the standard Presto_ITAR practice in the CTH models. Unlike most of the material models, which require all parameters to be specified, the CTH models have default values for most parameters. Additionally, the CTH models introduce the concept of material parameter libraries. These libraries are essentially lookup tables for the parameters of predefined materials. Thus, one need only specify a material model, such as CTH_KSES, and a material name like MATLABEL = ALUMINUM. All the parameters are then automatically loaded. Note that if a predefined material is specified, one may override library values by additionally specifying the desired properties. When no library material is specified, this is essentially what occurs, as the entry MATLABEL = USER is implicitly specified to read the default parameters from the material library.

Many models are unit independent, in that any set of parameters with a consistent set of units will work correctly with such models. This is the case for most of the models in Presto_ITAR. However, with the CTH models, this assumption is broken for certain equation of state models as well as by the use of material libraries. Thus, all CTH models must specify a system of units. Note that while this is only required for full models and not submodels, submodel parameters should be specified in units consistent with their parent model. In general, the unit declarations have the form given by the following block.

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL <string>mod_name
    # SI    = International system of units
    # IPS   = inch-pound-seconds
    # CGSK  = centimeters-grams-seconds-kelvin
    # CGSEV = centimeters-grams-seconds-electron volts
```

(continues on next page)

(continued from previous page)

```
    UNIT SYSTEM = <string>SI|IPS|CGSK|CGSEV|SESAME|SSHOCK(SI)
    LENGTH UNIT = <real>length_unit(1.0)
    MASS UNIT = <real>mass_unit(1.0)
    TIME UNIT = <real>time_unit(1.0)
    TEMPERATURE UNIT = <real>temperature_unit(1.0)
    AMOUNT UNIT = <real>amount_unit(1.0)
    CURRENT UNIT = <real>current_unit(1.0)
    LUMINOSITY UNIT = <real>luminosity_unit(1.0)
    ...
  END [PARAMETERS FOR MODEL <string>mod_name]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The `UNIT SYSTEM` command line may be used to specify an overall system of units to use. The values of that system may then be overridden for specific units by the other commands. The `...` denotes all the other parameters of the model, which have been omitted here. The meaning of a unit command is the value required to convert from SI to the desired unit system. Thus, for example, if one has a problem where length is measured in centimeters, one would specify `LENGTH UNIT = 1.e2`, since there are one hundred centimeters in a meter. Once the unit system has been specified in this manner, all the model parameters must be entered in this system.

Paths to the material libraries, as well as certain tabular data required by the CTH SESAME models, must be specified in the user input as well. Specific parameters are available for setting the names of data files in the model input. These may be relative or absolute paths. *Do not put the paths or file names in quotes.* Additionally, the models recognize the existence of the environment variable `CTHPATH`. When `CTHPATH` is undefined, the default path for all CTH data is relative to the current directory. When `CTHPATH` is defined, then SESAME table data is searched for relative to the directory `CTHPATH/data/`. Also, in this case material libraries are first searched for relative to the working directory and upon failure of that, relative to the directory `CTHPATH/data/`. If a model cannot find its material library file, it will throw a fatal error.

1.2.2.1 Mie-Gruneisen Model (CTH_MGR)

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL CTH_MGR
    {unit parameters}
    MATLABEL = <string>material_label(USER)
    EOS DATAFILE = <string>eos_data_file(EOS_data)
    R0 = <real>density
    T0 = <real>temperature(298.0)
    CS = <real>sound_speed
    S1 = <real>us_up_slope(0.0)
    G0 = <real>gruneisen_parameter(0.0)
```

(continues on next page)

(continued from previous page)

```
    CV = <real>heat_capacity
    ESFT = <real>energy_shift(0.0)
    RP = <real>porous_density(0.0)
    PS = <real>crushup_pressure(1.e9)
    PE = <real>elastic_pressure(0.0)
    CE = <real>elastic_sound_speed(0.0)
    NSUB = <real>num_subcycles(10.0)
    S2 = <real>us_up_quadratic(0.0)
    TYP = <real>model_type(1.0)
    RO = <real>density_alias
    TO = <real>temperature_alias
    S = <real>s1_alias
    GO = <real>g0_alias
    B = <real>low_pressure_coefficient(0.0)
    XB = <real>low_pressure_constant(1.e-4)
    NB = <real>low_pressure_power(1.0)
    PWR = <real>alpha_power(2.0)
  END [PARAMETERS FOR MODEL CTH_MGR]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Mie-Gruneisen material model describes the nonlinear pressure-volume (or equivalently pressure-density) response of solids or fluids in terms of a reference pressure-volume curve and deviations from the reference curve in energy space. The reference curve is taken to be the experimentally determined principal Hugoniot, which is the locus of end states that can be reached by a shock transition from the ambient state. For details about this model, see [14].

For Mie-Gruneisen energy-dependent materials, the Mie-Gruneisen command block begins with the input line:

```
BEGIN PARAMETERS FOR MODEL CTH_MGR
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR MODEL CTH_MGR]
```

In the above command blocks:

- The `{unit parameters}` line is a placeholder for the unit block described in Section 1.2.2.
- The `MATLABEL` command line specifies the name of a material parameter library entry from which to take default values for the other parameters. This name is searched for under the model name `MGR` in the data file specified by the command line `EOS DATAFILE`.
- The command lines `R0` (or `RO`), `CS`, and `CV` are required inputs to this model. Alternatively, one may specify a non-default `MATLABEL` command line. All other values are optional and may be left unspecified if the defaults are acceptable.

- The initial density for the Hugoniot is defined with the `R0` command line. If the material is porous, the `RP` command line defines the initial density and `R0` is the ambient density for the nonporous material.

For information about the CTH Mie-Gruneisen model, consult [14].

1.2.2.2 SESAME Tabular EOS Model (CTH_KSES)

Note: The SESAME tabular interface currently reads tables from a platform-specific binary table format. Production of this format from the ASCII tables requires use of the `bcat` code, which is not built by default. If a current CTH installation is available, then one may use that installation's data by setting the `CTHPATH` environment variable, see Section 1.2.2.

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL CTH_KSES
    {unit parameters}
    MATLABEL = <string>material_label(USER)
    EOS DATAFILE = <string>eos_data_file(EOS_data)
    EOS = <real>eos_number
    SR = <real>scaling_factor(1.0)
    R0 = <real>density(table value)
    T0 = <real>temperature(table value)
    RMIN = <real>min_tension_density(0.8*R0)
    ZNUC = <real>avg_atomic_number(table value)
    ATWT = <real>avg_atomic_weight(table value)
    RP = <real>porous_density(0.0)
    PS = <real>crushup_pressure(1.e9)
    PE = <real>elastic_pressure(0.0)
    CE = <real>elastic_sound_speed(0.0)
    NSUB = <real>num_subcycles(10.0)
    ESFT = <real>energy_shift(table specific)
    TYP = <real>model_type(1.0)
    RO = <real>density_alias
    TO = <real>temperature_alias
    CLIP = <real>temperature_clip(0.0)
    PWR = <real>alpha_power(2.0)
    FEOS = <string>sesame_file
  END [PARAMETERS FOR MODEL CTH_KSES]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The SESAME tabular EOS model represents the thermodynamic state of a material through tabular representations of the pressure and internal energy as functions of density and temperature. Such tables may represent behavior as simple as an ideal gas to extremely complicated multi-phase behaviors. For more information on the implementation of this model,

consult [14]. Information on the SESAME format may be obtained from [17].

For SESAME materials, the SESAME command block begins with the input line:

```
BEGIN PARAMETERS FOR MODEL CTH_KSES
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR MODEL CTH_KSES]
```

In the above command blocks:

- The `{unit parameters}` line is a placeholder for the unit block described in Section 1.2.2.
- The `MATLABEL` command line specifies the name of a material parameter library entry from which to take default values for the other parameters. This name is searched for under the model name `SES` in the data file specified by the command line `EOS DATAFILE`.
- The command lines `EOS` and `FEOS` are required inputs to this model. Alternatively, one may specify a non-default `MATLABEL` command line. All other values are optional and may be left unspecified if the defaults are acceptable.
- The command lines `R0`, `T0`, `ZNUC`, and `ATWT` default to the values given in the specified table.
- The command line `ESFT` defaults to a value such that the internal energy of the specified table will be strictly positive for all states. Care should be taken if setting this to a non-default value as one may break assumptions on the positivity of the internal energy present in other areas of the code.
- For a porous material, the `RP` command line defines the initial density and `R0` becomes the ambient density for the nonporous material.
- The command line `CLIP` sets a delta in temperature from the edge of the table to which off-table temperatures are returned. In this implementation, extrapolation off of the tabulated region of a SESAME table can produce unphysical behavior. Thus, it is recommended to set `CLIP` to a non-zero value. The default, `CLIP = 0.0`, is to not clip off-table temperatures. The temperature delta is taken as the absolute value of `CLIP`. Setting a negative value suppresses error messages generated by this process.

For information about the SESAME tabular EOS model, consult [14].

1.2.2.3 Elastic-Plastic Modular Model (CTH_EP)

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL CTH_EP
    {unit parameters}
    EOS MODEL = <string>CTH_KSES|CTH_MGR
    {eos model parameters}
    YIELD MODEL = <string>CTH_JO|CTH_ST|CTH_ZE|NONE(NONE)
    {yield model parameters}
    FRACTURE MODEL = <string>CTH_JFRAC|NONE(NONE)
    {fracture model parameters}
    RHOL = <real>lower_density(0.0)
    RHOU = <real>upper_density(0.0)
  END [PARAMETERS FOR MODEL CTH_EP]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Elastic-Plastic Modular model combines an EOS, yield, and fracture model in the manner that CTH employs. In particular, the yield models are all of the "traditional" version which compute a yield stress and shear modulus. The resultant stress is calculated from a radial return plasticity model. Density degradation of the yield stress is applied when the density lies between the upper and lower density limits.

For the Elastic-Plastic Modular model, the `CTH_EP` command block begins with the input line:

```
BEGIN PARAMETERS FOR MODEL CTH_EP
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR MODEL CTH_EP]
```

In the above command blocks:

- The `{unit parameters}` line is a placeholder for the unit block described in Section 1.2.2.
- The `{eos model parameters}`, `{yield model parameters}`, and `{fracture model parameters}` lines are placeholders for all the parameters of the desired EOS, yield, and fracture models, respectively.
- An EOS model must be specified by the command line `EOS MODEL`. All other inputs are optional.
- Density degradation of the yield stress is performed only when the command lines `RHOL` and `RHOU` are specified and satisfy `RHOU > RHOL > 0`.

Output variables available for this model are listed in Table 1.3.

Note: The `CTH_EP` model does not yet implement a failure model. Thus, while the available

fracture model does compute the damage of the material, this information is not acted upon.

1.2.2.4 Johnson-Cook Viscoplastic Model (CTH_JO)

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL <string>combined_model
    YIELD MODEL = CTH_JO
    VP DATAFILE = <string>vp_data_file(VP_data)
    YIELD MATLABEL = <string>yield_material_label(USER)
    AJO = <real>parameter_a(0.0)
    BJO = <real>parameter_b(0.0)
    CJO = <real>parameter_c(0.0)
    MJO = <real>exponent_m(0.0)
    NJO = <real>exponent_n(0.0)
    TJO = <real>melt_temperature(0.0)
    POISSON = <real>poissons_ratio(0.0)
  END [PARAMETERS FOR MODEL <string>combined_model]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Johnson-Cook Viscoplastic model updates the material yield stress based upon the plastic strain, plastic strain rate, and the temperature. For more details about this model, see [26].

Since the Johnson-Cook model updates only the yield stress for a material, it must be used in combination with a plasticity model and equation of state. Currently, this means it must be used as a submodel of the Elastic-Plastic Modular model, see Section 1.2.2.3.

In the above command blocks:

- The `combined_model` must currently be `CTH_EP`.
- The `YIELD MATLABEL` command line specifies the name of a material parameter library entry from which to take default values for the other parameters. This name is searched for under the model name `JO` in the data file specified by the command line `VP DATAFILE`.

For information about the Johnson-Cook model, consult [26].

1.2.2.5 Zerilli-Armstrong Plasticity Model (CTH_ZE)

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL <string>combined_model
    YIELD MODEL = CTH_ZE
    VP DATAFILE = <string>vp_data_file(VP_data)
    YIELD MATLABEL = <string>yield_material_label(USER)
    C1ZE = <real>constant_c1(0.0)
```

(continues on next page)

(continued from previous page)

```
    C2ZE = <real>constant_c2(0.0)
    C3ZE = <real>constant_c3(0.0)
    C4ZE = <real>constant_c4(0.0)
    C5ZE = <real>constant_c5(0.0)
    AZE = <real>constant_a(0.0)
    NZE = <real>constant_n(0.0)
    POISSON = <real>poissons_ratio(0.0)
  END [PARAMETERS FOR MODEL <string>combined_model]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Zerilli-Armstrong Plasticity model updates the material yield stress based upon the plastic strain, plastic strain rate, and the temperature. For more details about this model, see [26].

Since the Zerilli-Armstrong model updates only the yield stress for a material, it must be used in combination with a plasticity model and equation of state. Currently, this means it must be used as a submodel of the Elastic-Plastic Modular model, see Section 1.2.2.3.

In the above command blocks:

- The `combined_model` must currently be `CTH_EP`.
- The `YIELD MATLABEL` command line specifies the name of a material parameter library entry from which to take default values for the other parameters. This name is searched for under the model name `ZE` in the data file specified by the command line `VP DATAFILE`.

For information about the Zerilli-Armstrong model, consult [26].

1.2.2.6 Steinberg-Guinan-Lund Plasticity Model (CTH_ST)

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL <string>combined_model
    YIELD MODEL = CTH_ST
    VP DATAFILE = <string>vp_data_file(VP_data)
    YIELD MATLABEL = <string>yield_material_label(USER)
    R0ST = <real>initial_density(0.0)
    TM0ST = <real>melt_temperature(0.0)
    ATMST = <real>melt_law_constant_a(0.0)
    GM0ST = <real>gruneisen_constant(0.0)
    AST = <real>shear_modulus_constant_a(0.0)
    BST = <real>shear_modulus_constant_b(0.0)
    NST = <real>work_hardening_constant_n(0.0)
    C1ST = <real>yield_stress_constant_c1(0.0)
    C2ST = <real>yield_stress_constant_c2(0.0)
    G0ST = <real>initial_shear_modulus(0.0)
    BTST = <real>work_hardening_constant_b(0.0)
    EIST = <real>initial_equivalent_plastic_strain(0.0)
```

(continues on next page)

(continued from previous page)

```
    YPST = <real>peierls_stress(0.0)
    UKST = <real>activation_energy(0.0)
    YSMST = <real>athermal_yield_stress(0.0)
    YAST = <real>athermal_prefactor(0.0)
    Y0ST = <real>initial_yield_stress(0.0)
    YMST = <real>max_yield_stress(0.0)
    POISSON = <real>poissons_ratio(0.0)
  END [PARAMETERS FOR MODEL <string>combined_model]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Steinberg-Guinan-Lund Plasticity model updates the material yield stress and shear modulus based upon the plastic strain, plastic strain rate, the density, and the temperature. For more details about this model, see [30].

Since the Steinberg-Guinan-Lund model updates only the yield stress and shear modulus for a material, it must be used in combination with a plasticity model and equation of state. Currently, this means it must be used as a submodel of the Elastic-Plastic Modular model, see Section 1.2.2.3.

In the above command blocks:

- The `combined_model` must currently be `CTH_EP`.
- The `YIELD MATLABEL` command line specifies the name of a material parameter library entry from which to take default values for the other parameters. This name is searched for under the model name `ST` in the data file specified by the command line `VP DATAFILE`.

For information about the Steinberg-Guinan-Lund model, consult [30].

1.2.2.7 Johnson-Cook Fracture Model (CTH_JFRAC)

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  BEGIN PARAMETERS FOR MODEL <string>combined_model
    FRACTURE MODEL = CTH_JFRAC
    FRACTURE MATLABEL = <string>fracture_material_label(USER)
    FRACTURE DATAFILE = <string>fracture_data_file(VP_data)
    JFD1 = <real>parameter_d1(0.0)
    JFD2 = <real>parameter_d2(0.0)
    JFD3 = <real>parameter_d3(0.0)
    JFD4 = <real>parameter_d4(0.0)
    JFD5 = <real>parameter_d5(0.0)
    JFTM = <real>melt_temperature(0.0)
    JFPF0 = <real>initial_fracture_pressure(0.0)
    DYLDRD = <real>strength_degradation_damage(0.0)
    DPFRD = <real>stress_degradation_damage(0.0)
```

(continues on next page)

(continued from previous page)

```
    YLDFLR = <real>minimum_yield_strength(0.0)
    FRCFLR = <real>minimum_fracture_stress(0.0)
    JFWM = <real>weibull_flag(0.0)
    JFIC = <real>random_seed_one(0.0)
    JFIT = <real>random_seed_two(0.0)
    JFVREF = <real>failure_volume(0.0)
    JFOUT = <real>output_message_flag(0.0)
  END [PARAMETERS FOR MODEL <string>combined_model]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Johnson-Cook Fracture model is a scalar damage model for predicting the failure of materials based upon the plastic strain, plastic strain rate, and yield stress. This model is completely independent of the similarly named Johnson-Cook Viscoplastic model, see Section 1.2.2.4. For details about this model, see [25].

Since the Johnson-Cook fracture model only calculates a damage, it must be used in combination with a plasticity model, equation of state, and a failure model. Currently, this means it must be used as a submodel of the Elastic-Plastic Modular model, see Section 1.2.2.3.

In the above command blocks:

- The `combined_model` must currently be `CTH_EP`.
- The `FRACTURE MATLABEL` command line specifies the name of a material parameter library entry from which to take default values for the other parameters. This name is searched for under the model name `JFRAC` in the data file specified by the command line `FRACTURE DATAFILE`.
- The Weibull modulus capability is currently unimplemented.

Output variables available for this model are listed in Table 1.4.

For more information about the Johnson-Cook Fracture model, consult [25].

1.2.3 Equation-of-State Model Specifications

This section describes material models that are applicable only for use in `Presto_ITAR`. The algorithms that apply when these energy-dependent models are in use are currently in a state of flux as they are being upgraded to the state-of-the-art. This transformation has currently been applied only to the midpoint-increment uniform-gradient hexahedron element. When using this element with EOS models, the new algorithms are chosen by default.

1.2.3.1 Mie-Gruneisen Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN
    RHO_0 = <real>density
    C_0 = <real>sound_speed
    SHUG = <real>const_shock_velocity
    GAMMA_0 = <real>ambient_gruneisen_param
    POISSR = <real>poissons_ratio
    Y_0 = <real>yield_strength
    PMIN = <real>mean_stress(REAL_MAX)
  END [PARAMETERS FOR MODEL MIE_GRUNEISEN]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Mie-Gruneisen material model describes the nonlinear pressure-volume (or equivalently pressure-density) response of solids or fluids in terms of a reference pressure-volume curve and deviations from the reference curve in energy space. The reference curve is taken to be the experimentally determined principal Hugoniot, which is the locus of end states that can be reached by a shock transition from the ambient state. For details about this model, see [27].

For Mie-Gruneisen energy-dependent materials, the Mie-Gruneisen command block begins with the input line:

```
BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR MODEL MIE_GRUNEISEN]
```

In the above command blocks:

- The `thermal strain option` is used to define thermal strains. See the Sierra/SolidMechanics User's Guide for further information on defining and activating thermal strains.
- The ambient density, ρ_0, is defined with the `RHO_0` command line. The ambient density is the density at which the mean pressure is zero, not necessarily the initial density.

- The ambient bulk sound speed, c_0, is defined by the C_0 command line. The ambient bulk sound speed is also the first constant in the shock-velocity-versus-particle-velocity relation $D = c_0 + Su$, where u is the particle velocity. (See the following description of the SHUG command line for the definition of S.)
- The second constant in the shock-velocity-versus-particle-velocity equation, S, is defined by the SHUG command line. The shock-velocity-versus-particle-velocity relation is $D = c_0 + Su$, where u is the particle velocity. (See the previous description of the C_0 command line for the definition of c_0.)
- The ambient Gruneisen parameter, Γ_0, is defined by the GAMMA_0 command line.
- Poisson's ratio, ν, is defined by the POISSR command line. Poisson's ratio is assumed constant.
- The yield strength, y_0, is defined by the Y_0 command line. The yield strength is zero for the hydrodynamic case.
- The fracture stress is defined by the PMIN command line. The fracture stress is a mean stress or pressure, so it must be negative or zero. This is an optional parameter; if not specified, the parameter defaults to REAL_MAX (no fracture).

For information about the Mie-Gruneisen model, consult [27].

1.2.3.2 Mie-Gruneisen Power-Series Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES
    RHO_0 = <real>density
    C_0 = <real>sound_speed
    K1 = <real>power_series_coeff1
    K2 = <real>power_series_coeff2
    K3 = <real>power_series_coeff3
    K4 = <real>power_series_coeff4
    K5 = <real>power_series_coeff5
    GAMMA_0 = <real>ambient_gruneisen_param
    POISSR = <real>poissons_ratio
```

(continues on next page)

(continued from previous page)

```
    Y_0 = <real>yield strength
    PMIN = <real>mean_stress(REAL_MAX)
  END [PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Mie-Gruneisen power-series model describes the nonlinear pressure-volume (or equivalently pressure-density) response of solids or fluids in terms of a reference pressure-volume curve and deviations from the reference curve in energy space. The reference curve is taken to be the experimentally determined principal Hugoniot, which is the locus of end states that can be reached by a shock transition from the ambient state. The Mie-Gruneisen power-series model is very similar to the Mie-Gruneisen model, except that the Mie-Gruneisen model bases the Hugoniot pressure-volume response on the assumption of a linear shock-velocity-versus-particle-velocity relation, while the Mie-Gruneisen power-series model uses a power-series expression. For details about this model, see [27].

For Mie-Gruneisen power-series energy-dependent materials, the Mie-Gruneisen power-series command block begins with the input line:

```
BEGIN PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR MODEL MIE_GRUNEISEN_POWER_SERIES]
```

In the above command blocks:

- The `thermal strain option` is used to define thermal strains. See the Sierra/SolidMechanics User's Guide for further information on defining and activating thermal strains.
- The ambient density, ρ_0, is defined with the `RHO_0` command line. The ambient density is the density at which the mean pressure is zero, not necessarily the initial density.
- The ambient bulk sound speed, c_0, is defined by the `C_0` command line.
- The power-series coefficients k_1, k_2, k_3, k_4, and k_5 are defined by the command lines `K1`, `K2`, `K3`, `K4`, and `K5`, respectively. Only the nonzero power-series coefficients need be input, since coefficients not specified will default to zero.
- The ambient Gruneisen parameter, Γ_0, is defined by the `GAMMA_0` command line.
- Poisson's ratio, ν, is defined by the `POISSR` command line. Poisson's ratio is assumed constant.
- The yield strength, y_0, is defined by the `Y_0` command line. The yield strength is zero for the hydrodynamic case.
- The fracture stress is defined by the `PMIN` command line. The fracture stress is a mean stress or pressure, so it must be negative or zero. This is an optional parameter; if not

specified, the parameter defaults to REAL_MAX (no fracture).

For information about the Mie-Gruneisen power-series model, consult [27].

1.2.3.3 JWL (Jones-Wilkins-Lee) Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL JWL
    RHO_0 = <real>initial_density
    D = <real>detonation_velocity
    E_0 = <real>init_chem_energy
    A = <real>jwl_const_pressure1
    B = <real>jwl_const_pressure2
    R1 = <real>jwl_const_nondim1
    R2 = <real>jwl_const_nondim2
    OMEGA = <real>jwl_const_nondim3
    XDET = <real>x_detonation_point
    YDET = <real>y_detonation_point
    ZDET = <real>z_detonation_point
    TDET = <real>time_of_detonation
    B5 = <real>burn_width_const(2.5)
  END [PARAMETERS FOR MODEL JWL]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The JWL model describes the pressure-volume-energy response of the gaseous detonation products of HE (High Explosive). For details about this model, see [27].

For JWL energy-dependent materials, the JWL command block begins with the input line:

```
BEGIN PARAMETERS FOR MODEL JWL
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR MODEL JWL]
```

In the above command blocks:

- The `thermal strain option` is used to define thermal strains. See the Sierra/SolidMechanics User's Guide for further information on defining and activating thermal strains.
- The initial density of the unburned explosive, ρ_0, is given by the `RHO_0` command line.
- The detonation velocity, D, is given by the `D` command line.
- The initial chemical energy per unit mass in the explosive, E_0, is given by the `E_0` command line. Note, this value has NO effect on the actual behavior of the model in terms of stresses returned or energy generated. The `E_0` term affects only what initial energy value is printed in the output log file. The energy generated by the JWL material is determined by the `A, B, R1, R2, and D` constants. Most compilations of JWL parameters give E_0 in units of energy per unit volume, rather than energy per unit mass. Thus, the tabulated value must be divided by ρ_0, the initial density of the unburned explosive.
- The JWL constants with units of pressure, A and B, are given by the `A` and `B` command lines, respectively.
- The dimensionless JWL constants, R_1, R_2, and ω, are given by the `R1`, `R2`, and `OMEGA` command lines, respectively.
- The x-coordinate of the detonation point, x_D, is given by the `XDET` command line.
- The y-coordinate of the detonation point, y_D, is given by the `YDET` command line.
- The z-coordinate of the detonation point, z_D, is given by the `ZDET` command line.
- The time of detonation, t_D, is given by the `TDET` command line.
- The burn-width constant, B_5, is given by the `B5` command line. The burn-width constant has a default value of 2.5.

For information about the JWL model, consult [27].

1.2.3.4 JWL (Jones-Wilkins-Lee) Model with Multiple Detonation Points

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
```

(continues on next page)

(continued from previous page)

```
  BEGIN PARAMETERS FOR MODEL JWL_MULTIPOINT
    RHO_0 = <real>initial_density
    D = <real>detonation_velocity
    E_0 = <real>init_chem_energy
    A = <real>jwl_const_pressure1
    B = <real>jwl_const_pressure2
    R1 = <real>jwl_const_nondim1
    R2 = <real>jwl_const_nondim2
    OMEGA = <real>jwl_const_nondim3
    B5 = <real>burn_width_const(2.5)
    XDET = <real>x_detonation_point... (up to 100 values)
    YDET = <real>y_detonation_point... (up to 100 values)
    ZDET = <real>z_detonation_point... (up to 100 values)
    TDET = <real>time_of_detonation... (up to 100 values)
  END [PARAMETERS FOR MODEL JWL_MULTIPOINT]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The JWL MULTIPOINT model describes the pressure-volume-energy response of the gaseous detonation products of HE (High Explosive). The mechanics for this model is identical to the JWL model, but this version permits up to 100 detonation points. Each detonation point can have its own detonation time. For details about this model, see [27].

For JWL MULTIPOINT energy-dependent materials, the `JWL_MULTIPOINT` command block begins with the input line:

```
BEGIN PARAMETERS FOR MODEL JWL_MULTIPOINT
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR MODEL JWL_MULTIPOINT]
```

In the above command blocks:

- The `thermal strain option` is used to define thermal strains. See the Sierra/SolidMechanics User's Guide for further information on defining and activating thermal strains.
- The initial density of the unburned explosive, ρ_0, is given by the `RHO_0` command line.
- The detonation velocity, D, is given by the `D` command line.
- The initial chemical energy per unit mass in the explosive, E_0, is given by the `E_0` command line. Note, this value has NO effect on the actual behavior of the model in terms of stresses returned or energy generated. The `E_0` term affects only what initial energy value is printed in the output log file. The energy generated by the JWL MULTIPOINT material is determined by the `A, B, R1, R2, and D` constants. Most compilations of JWL parameters give E_0 in units of energy per unit volume, rather than energy per unit

mass. Thus, the tabulated value must be divided by ρ_0, the initial density of the unburned explosive.

- The JWL constants with units of pressure, A and B, are given by the `A` and `B` command lines, respectively.
- The dimensionless JWL constants, R_1, R_2, and ω, are given by the `R1`, `R2`, and `OMEGA` command lines, respectively.
- The x-coordinates of the detonation points, x_D, are given by the `XDET` command line. Note that the number of detonation points specified should be the same number specified in the y and z coordinate locations as well as the detonation times.
- The y-coordinates of the detonation points, y_D, are given by the `YDET` command line. Note that the number of detonation points specified should be the same number specified in the x and z coordinate locations as well as the detonation times.
- The z-coordinates of the detonation points, z_D, are given by the `ZDET` command line. Note that the number of detonation points specified should be the same number specified in the x and y coordinate locations as well as the detonation times.
- The times of detonation for the detonation points, t_D, are given by the `TDET` command line. The detonation times can be different for each detonation point. Note that the number of detonation points specified should be the same number specified for the x, y, and z coordinate locations.
- The burn-width constant, B_5, is given by the `B5` command line. The burn-width constant has a default value of 2.5.

For information about the JWL model, consult [27].

1.2.3.5 Ideal Gas Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL IDEAL_GAS
    RHO_0 = <real>initial_density
    C_0 = <real>initial_sound_speed
    GAMMA = <real>ratio_specific_heats
```

(continues on next page)

(continued from previous page)

```
  END [PARAMETERS FOR MODEL IDEAL_GAS]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The ideal gas model provides a material description based on the ideal gas law. For details about this model, see [27].

For ideal gas materials, the ideal gas command block begins with the input line:

```
BEGIN PARAMETERS FOR MODEL IDEAL_GAS
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR MODEL IDEAL_GAS]
```

In the above command blocks:

- The `thermal strain option` is used to define thermal strains. See the Sierra/SolidMechanics User's Guide for further information on defining and activating thermal strains.
- The initial density, ρ_0, is given by the `RHO_0` command line.
- The initial sound speed, c_0, is given by the `C_0` command line.
- The ratio of specific heats, γ, is given by the `GAMMA` command line.

For information about the ideal gas model, consult [27].

1.2.4 Energy Deposition

```
BEGIN PRESCRIBED ENERGY DEPOSITION
  #
  # block set commands
  BLOCK = <string_list>block_names
  INCLUDE ALL BLOCKS
  REMOVE BLOCK
  #
  # function commands
  T FUNCTION = <string>t_func_name
  X FUNCTION = <string>x_func_name
  Y FUNCTION = <string>y_func_name
  Z FUNCTION = <string>z_func_name
  #
  # input mesh command
  READ VARIABLE = <string>mesh_var_name
  #
```

(continues on next page)

(continued from previous page)

```
  # user subroutine commands
  ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
  # other user subroutine command lines
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
END [PRESCRIBED ENERGY DEPOSITION]
```

The PRESCRIBED ENERGY DEPOSITION command block applies a set quantity of energy to energy-dependent material models for a given set of element blocks. Energy deposition defines a specific energy deposited (energy per unit mass) the code computes the actual energy added to each element by multiplying this applied specific energy by the element mass.

Energy deposition represents a particular type of boundary condition, and thus this command block follows the general specification of command blocks used to specify boundary conditions. See the Sierra/SolidMechanics User's Guide for more information on general boundary condition specification. The PRESCRIBED ENERGY DEPOSITION command block must appear in the region scope.

There are three options for defining the energy deposition for a set of elements: with standard SIERRA functions, with a mesh variable in the input mesh file, and by a user subroutine. If the energy deposition is a reasonably simple description and can be defined using the standard SIERRA functions, the function option is recommended. If the energy deposition requires a more complex description, it is necessary to use either the input mesh option or the user subroutine option. Only one of the three options can be specified in the command block.

The PRESCRIBED ENERGY DEPOSITION command block contains four groups of commands: block set, function, input mesh, and user subroutine. Each of these command groups, with the exception of the T FUNCTION command line, is basically independent of the others. Following are descriptions of the different command groups.

1.2.4.1 Block Set Commands

The block set commands portion of the PRESCRIBED ENERGY DEPOSITION command block defines a set of element blocks associated with the prescribed energy deposition and can include some combination of the following command lines:

```
BLOCK = <string_list>block_names
INCLUDE ALL BLOCKS
REMOVE BLOCK
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of blocks. See the Sierra/SolidMechanics User's Guide for more information about the use of these command lines for creating a set of blocks used in the command block. Either the `BLOCK` command line or the `INCLUDE ALL BLOCKS` command line must be present in the command block.

1.2.4.2 Function Commands

If the function option is used, either the T function or a set of T, X, Y, and Z function command lines must be included in the command block.

Following are the command lines related to the function option:

```
T FUNCTION = <string>t_func_name
X FUNCTION = <string>x_func_name
Y FUNCTION = <string>y_func_name
Z FUNCTION = <string>z_func_name
```

Each of the above command lines references a function name (defined in the SIERRA scope in a `DEFINITION FOR FUNCTION` command block). All the functions referenced in these four command lines must appear in the SIERRA scope.

The `T FUNCTION` command line gives the name of the user-defined T function. The T function describes how the applied input energy dose is integrated over time t. The T function should be 0 at the start time and 1 at the time at which all energy is deposited. The T function must be monotonically increasing over the time it is defined. The T function describes the total percentage of energy that is deposited at a given time.

The `X FUNCTION`, `Y FUNCTION`, and `Z FUNCTION` command lines define three functions, which we will denote as X, Y, and Z, respectively. The X, Y, and Z functions describe the total amount of energy to be deposited in an element as a function of position. Suppose we have element A with centroid (A_x, A_y, A_z) and mass M. The total energy that will have been deposited in element A at time t is given by:

$$E_A = M_A X(A_x) Y(A_y) Z(A_z) T(t) \tag{1.25}$$

where E_A is the total energy deposited.

1.2.4.3 Input Mesh Command

If the input mesh option is used, the quantity of specific energy deposited for each element will be read from an element variable defined in the mesh file.

Following is the command line related to the input mesh option:

```
READ VARIABLE = <string>mesh_var_name
```

The string `mesh_var_name` must match the name of an element variable in the mesh file that defines the energy deposition. Suppose that the total specific energy to be deposited for element A is $v(A)$. The quantity of energy deposited at time t is then given by:

$$E_A = M_A v(A) T(t) \tag{1.26}$$

The T function in Equation (1.26) is the same as that described in Section 1.2.4.2.

1.2.4.4 User Subroutine Commands

The user subroutine option allows for a very general description of the energy deposition, but this option requires that you write a user subroutine to implement this capability. The subroutine will be called by `adagio` at the appropriate time to generate the energy deposition.

Energy deposition uses an element subroutine signature. The subroutine returns one value per element for all the elements selected by use of the `block set commands`. The returned value is the specific energy flux at an element at a given time. The output flags array is ignored. The total energy deposited in an element is found by a time integration of the returned subroutine specific energy fluxes times the element mass. See the Sierra/SolidMechanics User's Guide for more information about user subroutines.

Following are the command lines related to the user subroutine option:

```
ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the `ELEMENT BLOCK SUBROUTINE` command line. The string `subroutine_name` is the name of a FORTRAN subroutine written by the user. The other command lines listed here (`SUBROUTINE DEBUGGING OFF`, `SUBROUTINE DEBUGGING ON`, `SUBROUTINE REAL PARAMETER`, `SUBROUTINE INTEGER PARAMETER`, and `SUBROUTINE STRING PARAMETER`) are described in the Sierra/SolidMechanics User's Guide.

1.2.4.5 Output Variables

When using prescribed energy deposition a few output variables become available:

- `specific_internal_energy` is an element variable that is the energy per unit mass that was applied by the boundary condition.
- `deposited_internal_energy` is the actual energy deposited by the boundary condition. This value is `specific_internal_energy` times the element mass.

1.3 Elements

This chapter describes additional information in the elements that are relevant to the energy-dependent material models described in this document. General information about the elements used in the SIERRA Solid Mechanics codes can be found in the Sierra/SolidMechanics User's Guide.

1.3.1 Finite Element Model

```
BEGIN FINITE ELEMENT MODEL <string>mesh_descriptor
  ...
  BEGIN PARAMETERS FOR BLOCK [<string list>block_names]


  END [PARAMETERS FOR BLOCK <string list>block_names]
END [FINITE ELEMENT MODEL <string>mesh_descriptor]
```

The only elements in the SIERRA Solid Mechanics codes that support energy-dependent material models are:

- **Eight-node, uniform-gradient hexahedron**: Only the midpoint-increment formulation [29] supports equation of state (EOS) models. This element is the most heavily tested with EOS models, and is the one currently recommended for use in this regime. This element is the only one that currently supports the use of CTH material models.
- **Four-node tetrahedron**: The regular element-based formulation and the node-based formulation for the four-node tetrahedron both support EOS models (but not the CTH models). However, both of these element formulations have problems with EOS models. The regular 4-node tetrahedral element is subject to volumetric and shear locking, which can lead to erroneous results. Recent investigations using the node-based tetrahedral element have shown problems with the computation of pressure, which is vital to the EOS computations. Thus it is recommended to avoid using either of these elements with EOS materials.

- **Smoothed particle hydrodynamics (SPH) elements**: These are one-dimensional elements. These elements can be used with EOS models (but not the CTH models). These elements are subject to numerical (non physics-based) failure in tension for all materials, so should be used with caution. Some analyses using SPH for explosives have shown marked deviation from expected behavior, so close comparisons should be made to other approaches, such as using uniform-gradient hexahedral elements.
- **Eight-node, total lagrange hexahedron**: Supports equation of state (EOS) models including the use of CTH material models.
- **Ten-node, total lagrange composite tetrahedron**: Supports equation of state (EOS) models including the use of CTH material models. Mid-edge nodes can sometimes exhibit spurious behavior when the material model is only an equation of state (purely dilatational, no deviatoric response).

1.3.1.1 Descriptors of Element Blocks

```
BEGIN PARAMETERS FOR BLOCK [<string list>block_names]
  LINEAR BULK VISCOSITY =
    <real>linear_bulk_viscosity_value(0.06)
  QUADRATIC BULK VISCOSITY =
    <real>quad_bulk_viscosity_value(1.20)
  MAX ENERGY ITERATIONS = <integer>max_energy_iterations(1)
  ENERGY ITERATION TOLERANCE = <real>energy_iteration_tolerance(1.0e-5)
  ELEMENT NUMERICAL FORMULATION = <string>OLD|NEW(OLD)
END [PARAMETERS FOR BLOCK <string list>block_names]
```

The finite element model consists of one or more element blocks. Associated with an element block or group of element blocks will be a `PARAMETERS FOR BLOCK` command block, which is also referred to in this document as an *element-block command block*. The basic information about the element blocks (number of elements, topology, connectivity, etc.) is contained in a mesh file. Specific attributes for an element block must be specified in the input file. The general commands for this block are described in the Sierra/SolidMechanics User's Guide, but several commands are of particular use when employing EOS models.

Linear and Quadratic Bulk Viscosity

```
LINEAR BULK VISCOSITY =
  <real>linear_bulk_viscosity_value(0.06)
QUADRATIC BULK VISCOSITY =
  <real>quad_bulk_viscosity_value(1.20)
```

The linear and quadratic bulk viscosity are set with these two command lines. These terms assist with the handling of strong discontinuities in stress, such as those found in a shock front. Setting

these parameters to a level that is too low will cause the simulation to exhibit excess noise ("ringing") in the simulation. Setting these too large, however, can cause excessive smearing of the discontinuity.

For more information, consult the documentation for the elements [16] for a description of the bulk viscosity parameters.

Energy Iterations

```
MAX ENERGY ITERATIONS =
  <integer>max_energy_iterations(1)
ENERGY ITERATION TOLERANCE =
  <real>energy_iteration_tolerance(1.0e-5)
```

When using an energy-dependent material model, the internal energy is updated using a second-order, implicit equation that includes terms for pressure-volume, entropy, and deposited work. The pressure-volume work is broken into hydrodynamic and deviatoric parts. Historically, `Presto_ITAR` has solved this equation under the assumption of a Mie-Gruneisen material, where the pressure is linearly dependent upon the internal energy, see [27]. Section 1.2.3 and Section 1.2.1 contain examples of materials which use this assumption. In these models, the energy equation is solved explicitly inside of the material model itself. However, the recent addition of more general material models (see Section 1.2.2) resulted in the need to remove this dependency. Also, for general portability of material models, the energy update was extracted from the material models and placed into the element for these general models.

Due to the implicit nature of the energy equation used by `Presto_ITAR`, an iteration is required to make the new state self-consistent. The `MAX ENERGY ITERATIONS` setting controls the maximum number of iterations performed in the self-consistent loop. When using a legacy material model, or a model from Section 1.2.2 that is purely hydrodynamic, the default value of `1` is sufficient. For the former models, this recovers the legacy behavior. In the case of the latter models, an isentropic predictor method is used that allows for an explicit solution of the implicit energy equation. For more general models from Section 1.2.2, one should set `MAX ENERGY ITERATIONS` to a value of at least `2`. This provides for a minimal amount of convergence in the energy equation.

The convergence criteria for exiting the self-consistent loop which calculates the implicit energy update may be set via the command `ENERGY ITERATION TOLERANCE`. For planar shock problems, the default value is typically reached after two or three iterations. Convergence to full double precision tolerance typically takes up to six or seven iterations. A warning message will be printed if the self consistent loop fails to converge to the desired tolerance within the maximum allowed number of iterations.

Element Numerical Formulation

```
ELEMENT NUMERICAL FORMULATION = <string>OLD|NEW(OLD)
```

For calculation of the critical time step, it is necessary to determine a characteristic length for each element. In one dimension, the correct characteristic element length is the distance between the two nodes of the element. In higher dimensions, this length is usually taken to be the minimum distance between any of the nodes in the element. However, some finite element codes, primarily those based on Pronto3D [29], use as a characteristic length an eigenvalue estimate based on work by Flanagan and Belytschko [7]. That characteristic length provides a stable time step, but often is far more conservative than the minimum distance between nodes. For a cubic element with side length equal to 1, and thus also surface area of each face and volume equal to 1, the minimum distance between nodes is 1. However, the eigenvalue estimate is $\frac{1}{\sqrt{3}}$, which is only 58% of the minimum distance. As the length of the element is increased in one direction while keeping surfaces in the lateral direction squares of area 1, the eigenvalue estimate asymptotes to $\frac{1}{\sqrt{2}}$ for very long elements. If the length is decreased, the eigenvalue estimate asymptotes to the minimum distance between nodes for very thin elements. In this case, the eigenvalue estimate is always more conservative than the minimum distance between nodes. However, consider an element whose cross section in one direction is not a square but a trapezoid with one side length much greater than the other. Assume the large side length is 1 and the other side length is arbitrarily small, ε. In this case, the minimum distance between nodes becomes ε, creating a small and inefficient time step. However, the eigenvalue estimate is related to the length across the middle of the trapezoid, which for the conditions stated is $\frac{1}{2}$. Since both distances provide stable time steps, and one or the other can be much larger in various circumstances, the most efficient calculation is obtained by using the maximum of the two lengths, either the eigenvalue estimate or the minimum distance between nodes, to determine the time step.

By using the maximum of the lengths, the computed critical time step will be very nearly unstable, and the `TIME STEP SCALE FACTOR` command line should be used to provide a margin of safety; the scale factor should not be greater than 0.9, and sometimes it may need to be reduced further. Although this time step estimate is closer to the critical value and provides better accuracy and efficiency, a smaller scale factor may need to be specified for stability. For this reason, the choice of approach to use is left to the user and is determined by the command line:

```
ELEMENT NUMERICAL FORMULATION = <string>OLD|NEW(OLD)
```

If the input parameter is `OLD`, only the eigenvalue estimate is used; `NEW` means that the maximum of the two lengths is used. The default is `OLD` so that users will have to specifically choose the new approach and be aware of the scale factor for the time step.

The `ELEMENT NUMERICAL FORMULATION` command line is applicable to energy-dependent material models, where the determination of the characteristic length is affected. This length scale will change both the artificial viscosity and the critical time step. Further details about the critical time step calculations and the use of this command line are available in [27].

1.3.2 Element Sections

Element sections are defined by section command blocks. There are currently nine different types of section command blocks. The section command blocks appear in the SIERRA scope, at the same level as the FINITE ELEMENT MODEL command block. No special parameters in the sections are required for the use of EOS models. However, there are some inputs in the SPH section that can be useful for explosives computations. The relevant section from the standard user's guides is duplicated here, with a few additional comments.

1.3.2.1 SPH Section

```
BEGIN SPH SECTION <string>sph_section_name
  DENSITY FORMULATION = <string>MATERIAL|KERNEL(MATERIAL)
END [SPH SECTION <string>sph_section_name]
```

SPH (smoothed particle hydrodynamics) is useful for modeling fluids or for modeling materials that undergo extremely large distortions. One must be careful when using SPH for modeling. SPH tends to exhibit both accuracy and stability problems, particularly in tension. An SPH particle interacts with other nearest-neighbor SPH particles based on radius properties of all the elements involved; SPH particles react with other elements, such as tetrahedra, hexahedra, and shells, through contact. You should consult [28] regarding the theoretical background for SPH. The full set of commands for the SPH section are listed in the SIERRA Solid Mechanics user's guides.

The DENSITY FORMULATION command can be used to define the way in which the particle radii are updated. For the default option MATERIAL, the material densities and nodal masses are used to compute a volume associated with a particle at a given time. The radius is then updated to be the cube root of that volume. The alternative option KERNEL computes the particle densities based off of the SPH particles' masses and the SPH kernel density function. The KERNEL option may be necessary if large expansion of particles is expected (for example, modeling large density changes in gases). The MATERIAL option generally changes particle densities and thus radii less than the KERNEL option, so is appropriate for analyses that do not have large density fluctuations. The KERNEL option is often necessary for EOS models for explosives (such as JWL) or for shocks in gaseous materials.

1.4 Boundary Conditions

This chapter documents a specialized pressure boundary condition that is currently only available in the Presto_ITAR version of the Sierra/SM code. Refer to the Sierra/SM User's Guide for documentation of other boundary conditions.

1.4.1 Blast Pressure

```
BEGIN BLAST PRESSURE <string>name
  SURFACE = <string list>surface_ids
  REMOVE SURFACE = <string list>surface_id
  BLOCK = <string list>block_ids
  REMOVE BLOCK = <string list>block_ids
  INCLUDE ALL BLOCKS

  BURST TYPE = <string>SURFACE|AIR
  TNT MASS IN LBS = <real>tnt_mass_lbs
  BLAST TIME = <real>blast_time
  BLAST LOCATION = <real>loc_x <real>loc_y <real>loc_z
  ATMOSPHERIC PRESSURE IN PSI = <real>atmospheric_press
  AMBIENT TEMPERATURE IN FAHRENHEIT = <real>temperature
  FEET PER MODEL UNITS = <real>feet
  MILLISECONDS PER MODEL UNITS = <real>milliseconds
  PSI PER MODEL UNITS = <real>psi
  PRESSURE SCALE FACTOR = <real>pressure_scale(1.0)
  IMPULSE SCALE FACTOR = <real>impulse_scale(1.0)
  POSITIVE DURATION SCALE FACTOR = <real>duration_scale(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
  BLOCKING SURFACE CALCULATION = {OFF|ON}(OFF)
END [BLAST PRESSURE <string>name]
```

The `BLAST PRESSURE` command block is used to apply a pressure load resulting from a conventional explosive blast. This boundary condition is based on [15]] and [[22], and Sachs scaling is implemented to match ConWep ([35]).

> **Warning:** The data that `BLAST PRESSURE` utilizes has been updated to match data from ConWep 2.1.0.8 and no longer matches the curves reported in [15] or [22].

Angle of incidence is accounted for by transitioning from reflected pressure to incident pressure according to:

$$P_{total} = P_{ref} \cos\theta + P_{inc}(1 - \cos\theta) \tag{1.27}$$

where θ is the angle between the face normal vector and the direction to the blast from the face, P_{total} is the total pressure, P_{ref} is the reflected portion of the pressure, and P_{inc} is the incident portion of the pressure. P_{ref} and P_{inc} are based on Friedlander's equation, as described in [22].

The `BLAST PRESSURE` command block can be used for surfaces that have faces derived from solid elements (eight-node hexahedra, four-node tetrahedra, eight-node tetrahedra, etc.), membranes, and shells.

The BLAST PRESSURE command block can also be used for particle-like elements if the particle elements are created through the use of element death particle conversion. The surfaces must be defined on the original solid elements.

If θ is greater than 90 degrees (i.e., the face is pointing away from the blast), only P_{inc} is applied to the face. In this case, the face variable cosa, which contains $\cos\theta$, is set to zero.

This boundary condition is applied to the surfaces in the finite element model specified by the SURFACE command line or the exterior of blocks of elements via the BLOCK or INCLUDE ALL BLOCKS command line. (Any surface specified on the REMOVE SURFACE command line is then removed from this set.)

Warning: Exercise caution when using INCLUDE ALL BLOCKS, as the boundary condition will be applied to all faces. This may result in nonzero forces on faces with normals pointing away from the blast. It is recommended that users verify that the surface normals point in the correct direction. Additionally, when using shell elements, both faces of the element are included in the calculations. To address this issue, enable blocking by using the command BLOCKING SURFACE CALCULATION = ON, which assumes that faces with normals facing away from the blast are covered.

Table 1.1 Face Variables for Blast Pressure Boundary Condition

Variable Name	Type	Comments
pressure	Real	Current total pressure. Varies in time.
normal	Vector_3D	Face normal vector.
incident_pressure	Real	Peak incident pressure.
reflected_pressure	Real	Peak reflected pressure.
alpha	Real	Decay coefficient α.
beta	Real	Decay coefficient β.
cosa	Real	Cosine of θ.
arrival_time	Real	Time for arrival of blast at face.
positive_duration	Real	Duration of blast at face.

The face variables listed above will only be output when specifying SURFACE = <string list>surface_ids and will not be output when using INCLUDE ALL BLOCKS.

The type of burst load is specified with the BURST TYPE command, which can be SURFACE or AIR. The SURFACE option is used to define a hemispherical burst, while the AIR option is used for a spherical burst.

The equivalent amount of TNT (in pounds) is defined with the TNT MASS IN LBS command. The time at which the explosive is detonated is defined using the BLAST TIME command. This can be negative, and can be used to start the analysis at the time when the blast reaches the structure, saving computational time. The location of the blast is defined with the BLAST LOCATION command. Both BLAST TIME and BLAST LOCATION should be specified in the unit system of the model.

The current ambient pressure and temperature are defined using the ATMOSPHERIC PRESSURE IN PSI and AMBIENT TEMPERATURE IN FAHRENHEIT commands, respectively. As

implied by the command names, these must be supplied in units of pounds per square inch and degrees Fahrenheit.

Because of the empirical nature of this method for computing an explosive load, appropriate conversion factors for the unit system used in the model must be supplied. The commands `FEET PER MODEL UNITS`, `MILLISECONDS PER MODEL UNITS`, and `PSI PER MODEL UNITS` are used to specify the magnitude of one foot, one millisecond, and one pound per square inch in the unit system of the model.

All of the commands listed above are required. Scaling factors can optionally be applied to modify the peak pressure, the impulse, and the duration of the loading. The `PRESSURE SCALE FACTOR` command scales the peak value of both the reflected and incident portions of the applied pressure. The `IMPULSE SCALE FACTOR` command scales the impulse of the reflected and incident portions of the applied pressure. The `POSITIVE DURATION SCALE FACTOR` command scales the duration of the reflected and incident portions of the applied pressure. Each of these scaling factors only affects the quantity that it modifies; for example, scaling the pressure does not affect the impulse or duration.

The `ACTIVE PERIODS` and `INACTIVE PERIODS` commands can optionally be used to activate or deactivate this boundary condition for certain time periods.

Optionally, a surface blocking calculation may be performed to determine whether certain surfaces shadow others from the blast loading. This calculation is enabled by the command `BLOCKING SURFACE CALCULATION = ON`. The blocking surface calculation determines the percentage of each face that is exposed to the blast source point. The incident and reflected pressure of the blast is then scaled by the uncovered area of the face.

For information about the blast pressure model, consult [15]], [[22]], and [[35].

1.5 Output Variables for Material Models

Most material models have state variables that can be output upon request. State variables can be accessed by name or index, although most of the time they are accessed by name. They are only accessed by index under special circumstances. Refer to the Sierra/SolidMechanics User's Guide for more information on how state variables are requested for output. Tables of state variables for the material models that are only available in Sierra/SolidMechanics are provided below. These tables contain the indices and names used to access the state variables.

Table 1.2 State Variables for Bodner-Partom Model (Section 1.2.1.3)

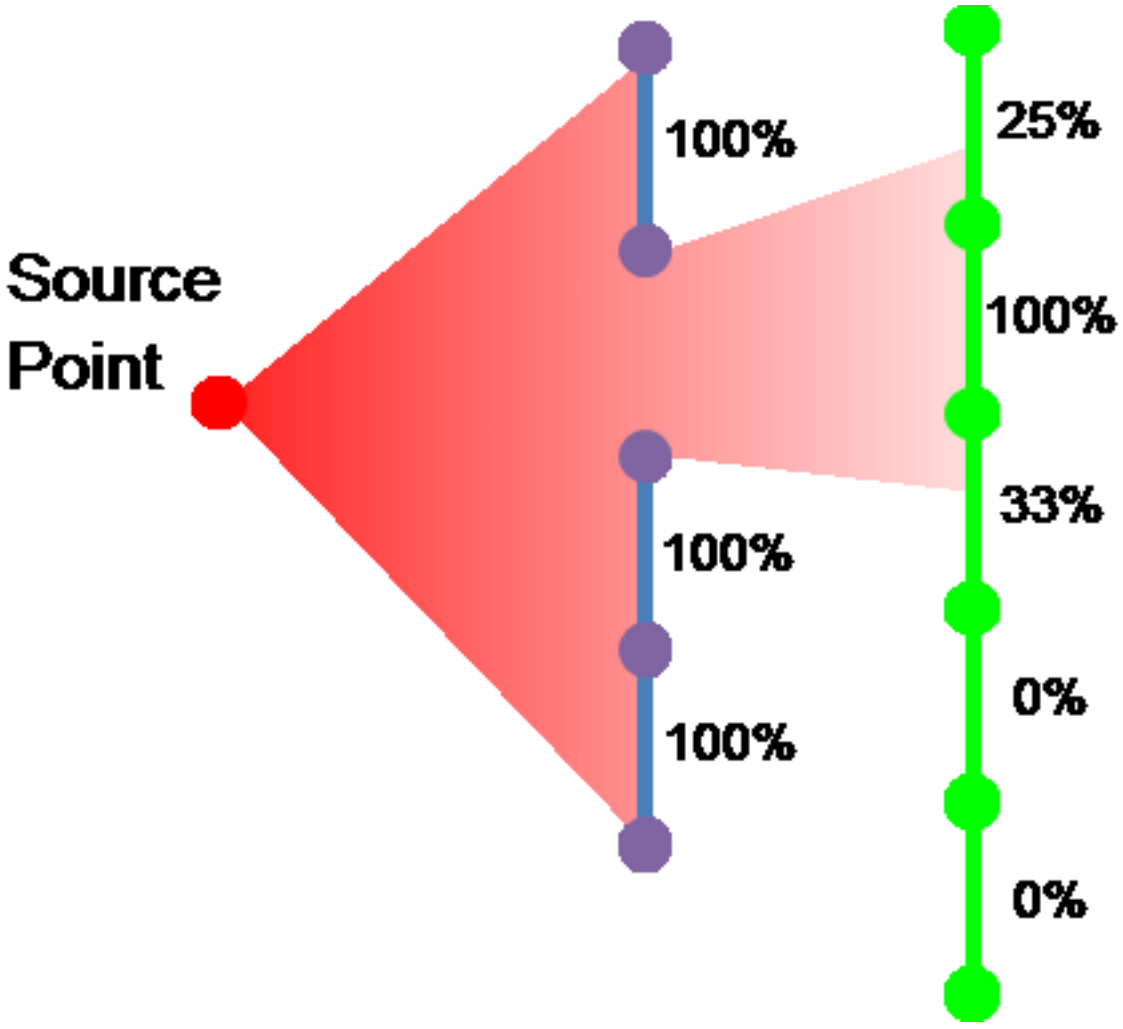


Fig. 1.1 Example Blocking Surface Calculation.

Index	Name
0	FAILURE_FLAG
1	EQPS
2	PLASTIC_WORK
3	INTERNAL_ENERGY
4	EQPS_RATE
5	BULK_VISCOSITY
6	SQ_SOUND_SPEED
7	INITIAL_VOLUME
8	VOLUME_STRAIN
9	PRESSURE
10	ELEMENT_LENGTH
11	EQUIVALENT_STRESS
12	TEMPERATURE

Table 1.3 State Variables for CTH_EP Model (Section 1.2.2.3)

Index	Name
0	EQPS
1	EQDOT

Table 1.4 State Variables for CTH_JFRAC Model (Section 1.2.2.7)

Index	Name
0	DAMAGE
1	FAILURE_FRACTION
2	FAILURE_THRESHOLD

Table 1.5 State Variables for Holmquist-Johnson-Cook Concrete Model (Section 1.2.1.4)

Index	Name
0	FAILURE_FLAG
1	EQPS
2	PLASTIC_WORK
3	INTERNAL_ENERGY
4	EQPS_RATE
5	BULK_VISCOSITY
6	SQ_SOUND_SPEED
7	INITIAL_VOLUME
8	VOLUME_STRAIN
9	VOLUME_STRAIN_PER_CURRENT_VOLUME
10	PRESSURE
11	ELEMENT_LENGTH
12	EQUIVALENT_STRESS
13	MAX_VOLUMETRIC_STRAIN
14	DAMAGE

Table 1.6 State Variables for Hull Concrete Model (Section 1.2.1.5)

Index	Name
0	FAILURE_FLAG
1	EQPS
2	PLASTIC_WORK
3	INTERNAL_ENERGY
4	EQPS_RATE
5	ARTIFICIAL_VISCOSITY
6	SQ_SOUND_SPEED
7	VOLUME_STRAIN
8	PRESSURE
9	ELEMENT_LENGTH
10	EFFECTIVE_STRESS
11	MAX_VOL_STRAIN_CUR_VOL

Table 1.7 State Variables for Johnson-Holmquist Ceramic Models (Section 1.2.1.7)

Index	Name
0 & FAILURE_FLAG	
1 & EQPS	
2 & PLASTIC_WORK	
3 & INTERNAL_ENERGY	
4 & EQPS_RATE	
5 & BULK_VISCOSITY	
6 & SQ_SOUND_SPEED	
7 & INITIAL_VOLUME	
8 & VOLUME_STRAIN	
9 & VOLUME_STRAIN_PER_CURRENT_VOLUME	
10 & PRESSURE	
11 & ELEMENT_LENGTH	
12 & EQUIVALENT_STRESS	
13 & DAMAGE	
14 & BULKING_PRESSURE	
15 & Z_FORCE	

Table 1.8 State Variables for Johnson-Holmquist-Beissel Ceramic Models (Section 1.2.1.8)

Index	Name
0 & FAILURE_FLAG	
1 & EQPS	
2 & PLASTIC_WORK	
3 & INTERNAL_ENERGY	
4 & EQPS_RATE	
5 & BULK_VISCOSITY	
6 & SQ_SOUND_SPEED	
7 & INITIAL_VOLUME	
8 & VOLUME_STRAIN	
9 & VOLUME_STRAIN_PER_CURRENT_VOLUME	
10 & PRESSURE	
11 & ELEMENT_LENGTH	
12 & EQUIVALENT_STRESS	
13 & DAMAGE	
14 & BULKING_PRESSURE	
15 & MAX_VOLUME_STRAIN	
16 & Z_FORCE	

Table 1.9 State Variables for Johnson-Cook Model (Section 1.2.1.6)

Index	Name
0 & FAILURE_FLAG	
1 & EQPS	
2 & PLASTIC_WORK	
3 & INTERNAL_ENERGY	
4 & EQPS_RATE	
5 & BULK_VISCOSITY	
6 & SQ_SOUND_SPEED	
7 & INITIAL_VOLUME	
8 & VOLUME_STRAIN	
9 & VOLUME_STRAIN_CURRENT	
10 & PRESSURE	
11 & SBAR	
12 & EQUIVALENT_STRESS	
13 & TEMPERATURE	
14 & DAMAGE	
15 & INITIAL_FAIL_STRAIN	

1.6 Zapotec

1.6.1 Introduction

Coupled Euler-Lagrange solution approaches are well suited for modeling problems involving penetration and blast loading on structures. Such problems are characterized by the transient, coupled interaction between bodies and/or problem domains. Furthermore, these problems are often found to have physical domains best modeled using different solution approaches. For example, in an earth penetration problem, the soil is best modeled using an Eulerian solution approach due to the large material deformations involved. Conversely, the penetrator is best modeled using a Lagrangian approach as structural response is of primary interest. A coupled approach utilizes the strengths of the two solution approaches to address problems not readily solved by either method alone. In this report, we describe the development of Zapotec, a coupled Eulerian-Lagrangian computer code for solving the aforementioned class of problems.

Zapotec couples the CTH (current version 13.0) [18], [10], [2]] and `Presto_ITAR` [[33], [32] codes. CTH, an Eulerian shock physics code, performs the Eulerian portion of the analysis, while `Presto_ITAR`, an explicit finite element code, performs the Lagrangian analysis. Zapotec controls the coupling between the two codes. In a Zapotec analysis, both CTH and `Presto_ITAR` are run concurrently. For a given time step, Zapotec maps the current configuration of a Lagrangian body and its state onto the fixed Eulerian mesh. Any overlapping Lagrangian material is inserted into the Eulerian mesh with the updated mesh data passed back to CTH. Once the material insertion is complete, the external loading on the Lagrangian material surfaces is then determined from the stress state in the Eulerian mesh. These loads are passed back to `Presto_ITAR` as a set of external nodal forces. Once the coupled treatment is complete, both CTH and `Presto_ITAR` are run independently over the next time step.

The Zapotec methodology and user inputs required to perform a Zapotec analysis are provided in the remainder of this report. Sections Section 1.6.2 and Section 1.6.3 provide background regarding the Zapotec algorithm and its implementation. Section Section 1.6.4 describes the user inputs for Zapotec. Examples are available in the Zapotec Examples Manual [31].

1.6.2 Methodology

1.6.2.1 Background

The solution approach taken by Zapotec can be described as a loose coupling between two pre-existing codes, CTH and `Presto_ITAR`. CTH performs the Eulerian portion of the analysis, while `Presto_ITAR` performs the Lagrangian calculations. The two codes are run concurrently with the appropriate portions of a problem solved on their respective computational domains. A brief description of the two codes follows. It should be noted that Zapotec only supports 3D problem development, *i.e.*, the code does not support 1D or 2D applications. This should not pose any difficulties, as the majority of real applications are 3D and symmetry boundary conditions can be applied to emulate many 1D and 2D configurations.

Before continuing, it should be noted that there are numerous references to Lagrangian methods throughout this report. There are many Lagrangian methods (*e.g.*, finite elements, finite difference, meshless methods, *etc.*). Any reference to Lagrangian methods will really refer to the finite element method since the Zapotec coupling is with `Presto_ITAR`, an explicit finite element code. Likewise, any reference to Eulerian methods will really refer to the solution procedure utilized by CTH.

CTH is an Eulerian shock physics code that utilizes a two-step approach for the solution of the conservation equations [18], [10], [2]. The two-step solution approach first involves a Lagrangian step, where the Eulerian mesh is allowed to deform. The Lagrangian step is followed by a remap step. The remap algorithm advects material quantities (*i.e.*, the volume flux, mass, momentum, and energy) from the deformed Lagrangian configuration back into the fixed Eulerian configuration. The Lagrangian step taken by CTH is transparent to Zapotec, *i.e.*, this is done internally within CTH. Thus, the Eulerian domain, as observed by Zapotec, is composed of a fixed, rectilinear mesh. The reader is directed to [18]] and [[10] for a more thorough discussion of the CTH methodology.

`Presto_ITAR` is an explicit Lagrangian, finite element code developed for modeling transient solid mechanics problems involving large deformations and contact [33], [32]. The numerical formulation utilizes an updated Lagrangian approach whereby the reference state at each time step is updated to coincide with the current configuration. Although the `Presto_ITAR` formulation accommodates several element types, Zapotec supports only a limited set. The supported element types are the Flanagan-Belytschko 8-node constant strain hexahedral element [8], 4-noded and 10-noded tetrahedral solid elements, and the 4-node constant strain quadrilateral shell element. These elements require hourglass control to eliminate the energy-less modes of deformation. SPH and mass particles in `Presto_ITAR` are also supported in Zapotec; however, particles created by conversion upon element death are not currently supported. Particles must be in the input mesh. The reader is directed to [32] for a thorough discussion of the `Presto_ITAR` methodology.

For an Eulerian method, the mesh is fixed in space with material allowed to move through the mesh. This is advantageous for modeling problems involving large material deformations and/or diffusion and mixing of gaseous materials. However, the solution scheme presents some difficulties for material interface tracking and modeling complex material response, particularly that involving history-dependent materials. With a Lagrangian approach, the mesh deforms with the material. As a result, boundary and contact conditions are well defined. The major weakness of a Lagrangian method lies with mesh deformation, where severe element distortion degrades accuracy and can potentially lead to a failure of the calculation due to mesh entanglement. A coupled approach can overcome the weaknesses associated with the two methods, allowing for solution of a class of problems not readily solved by either method alone.

For the class of problems of interest here (impact and blast loading on structures), the degree of mesh distortion is expected to be severe. In most cases, severe element distortion can be expected along contact and/or material interfaces. The current technology within `Presto_ITAR` for handling large mesh distortions relies upon an element death algorithm. Element death is a technique commonly used to remove highly distorted elements from the calculation once a user-prescribed death criterion is met. This technique goes by many names including the eroding interface algorithm within the EPIC code [13]] and the Slidesurfaces with Adaptive New

Definitions (SAND) algorithm within DYNA [[34]. Element death allows the extension of the finite element method to problems that would not normally be considered. It is an engineering algorithm that has its limitations. The algorithm can be sensitive to both mesh density and death criteria. Poor choices can lead to excessive removal of elements and a loss of solution accuracy for some applications [5], [12].

Zapotec provides an experimental capability for accommodating element death. This capability is referred to as donation. When an element dies, instead of discarding it, the material from the element is "donated" to the Eulerian domain and is henceforth treated as an Eulerian material. As an example, consider the ballistic impact of a long, blunt tungsten rod against a steel target. In a Zapotec analysis, the projectile would be modeled as a Lagrangian material, while the target is treated as Eulerian. During the course of penetration, the nose of the projectile mushrooms, as projectile material flows backwards following the surface of the crater formed in the target (see Fig. 1.2). This process is referred to as erosion. It is difficult to model the eroding material using the Lagrangian finite element method alone. Elements within the region of the eroding material become highly distorted, making it necessary to invoke element death to keep the calculation running. The consequence of using element death is that elements are removed from the contact surface, leading to a somewhat less than accurate treatment of the contact problem. With Zapotec's donation algorithm, dying `Presto_ITAR` elements are donated to CTH, allowing them to be treated as Eulerian materials for the duration of the analysis.

1.6.2.2 Nomenclature

Before discussing the specifics of the Zapotec methodology, it is useful to first define commonly used variable names and associated nomenclature. As a general rule, plain text will denote scalar quantities, while bold face denotes either vector or tensor quantities. A list of commonly used variables and their definitions are as follows:

Table 1.10 Commonly Used Variables

Variable	Definition
m	Mass
A	Area
V	Volume
ρ	Density
ϕ	Volume fraction
c	Sound speed
p	Pressure defined positive in compression ($p = -1/3\text{trace}(\boldsymbol{\sigma})$)
$\boldsymbol{s}$	Deviatoric stress
$\boldsymbol{\sigma}$	Cauchy stress defined positive in tension ($\boldsymbol{\sigma} = \boldsymbol{s} - p\boldsymbol{I}$)
$\boldsymbol{P}$	Linear momentum
e	Internal energy per unit mass
E	Internal energy
$\boldsymbol{n}$	Surface normal
$\boldsymbol{t}$	Traction vector ($\boldsymbol{t} = \boldsymbol{\sigma} \cdot \boldsymbol{n}$)
$\boldsymbol{v}$	Velocity
$\boldsymbol{f}$	Force
μ	Coulomb friction coefficient

In the ensuing discussion, subscripts will be associated with the above quantities to denote the domain and/or type of quantity. A list of subscript definitions follows:

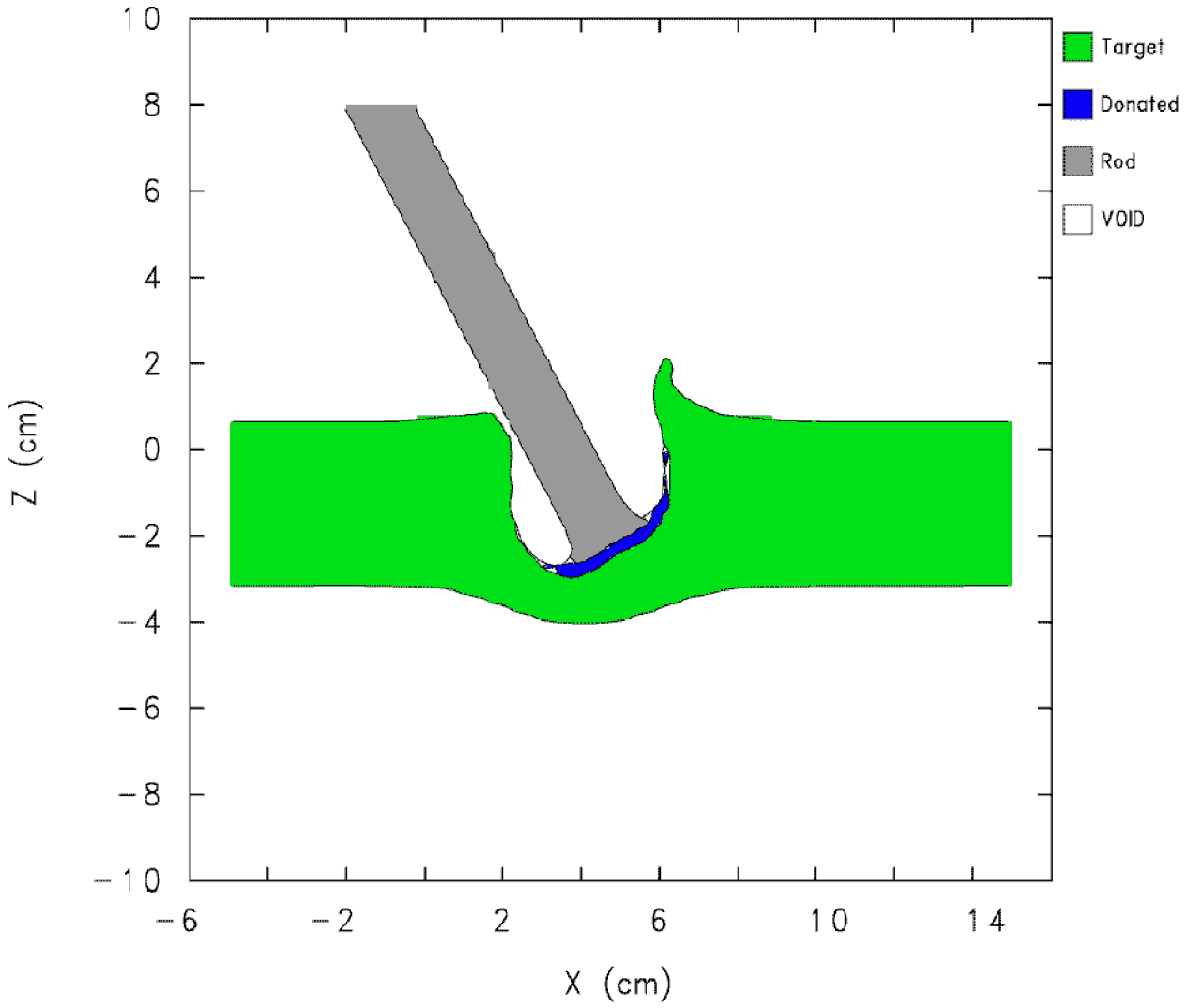


Fig. 1.2 Illustration of donation.

Table 1.11 Subscript Definitions

Subscript	Definition
ref	Reference state
L	Lagrangian problem domain
E	Eulerian problem domain
O	Overlap region ($L \cap E$)
l	Refers to a specific Lagrangian element
e	Refers to a specific Eulerian cell ($e = e(i, j, k)$), where i, j, k are cell indices
void	Void material within Eulerian domain
m	Material number
n	Normal component
t	Tangential component

The total number of materials in a problem (N_{mat}) is the summation of the number of Eulerian materials (N_{E}) and the number of Lagrangian materials inserted into the Eulerian mesh (N_{IL}). The latter need not be the same as the number of actual Lagrangian finite element blocks in the inputs for `Presto_ITAR` (N_{L}). It is important to note the difference between the two quantities. It is often advantageous to map the Lagrangian finite element blocks into a reduced set of CTH materials to reduce memory requirements (often down to only one CTH material). This is permitted since all state data associated with the Lagrangian materials are independently mapped into the Eulerian mesh.

In the Zapotec terminology, an inserted Lagrangian material will be referred to as a placeholder material within CTH. Eulerian CTH materials are classed as either permanent or donated. The relationship of these materials with respect to the CTH data storage location is illustrated in Table 1.12. When considering donation, some care is required in setting aside storage for Lagrangian materials that will be donated to the Eulerian domain. It is required that the number of permanent Eulerian materials ($N_{\text{E,permanent}}$) along with their constitutive data be defined first. Material definitions for donated and placeholder materials follow.

For many problems, the use of a single placeholder material is sufficient; however, there are situations where the user should define multiple placeholder materials. These situations arise when the problem contains multiple Lagrangian materials, with the materials having vastly differing characteristics. Difficulties can arise with ensuring consistency of pressure and energy reference states between `Presto_ITAR` and CTH. Also, inaccuracies can arise in the treatment of over-filled cells. Issues related to both will be discussed in more detail in the next section, which describes the material insertion algorithm.

Table 1.12 Classification of Materials for a Zapotec Analysis

Material Type	Classification	Description	CTH Storage Location for Material
Eulerian	Permanent	True Eulerian material	$1 \le m \le N_{\text{E,permanent}}$
Eulerian	Donated	L material donated to E	$N_{\text{E,permanent}} < m \le N_{\text{E}}$
Lagrangian	Placeholder	Inserted Lagrangian	$N_{\text{E}} < m \le (N_{\text{E}} + N_{\text{IL}})$

1.6.2.3 Overview of the Algorithm

The Zapotec coupling algorithm is summarized in Table 1.13 and Fig. 1.3. In the coupled treatment, Zapotec maps the current configuration of a Lagrangian body onto the fixed Eulerian mesh. Any overlapping Lagrangian material is inserted into the Eulerian mesh with the updated mesh data passed back to CTH. Once the material insertion is complete, the external loading on the Lagrangian material surfaces is determined from the stress state in the Eulerian mesh. These loads are passed back to `Presto_ITAR` as a set of external nodal forces. Once the coupled treatment is complete, both CTH and `Presto_ITAR` are run independently over the next time step.

Table 1.13 Summary of the Zapotec coupling algorithm

Step	Description
1	Remove pre-existing Lagrangian material from the CTH mesh
2	Get updated Lagrangian data
3	Insert Lagrangian material into the CTH mesh: a. Compute volume overlaps b. Map Lagrangian data (mass, momentum, pressure, sound, speed, internal energy) c. Pass updated mesh data back to CTH
4	Compute external forces on Lagrangian surface: a. Determine surface overlaps b. Compute surface tractions based on Eulerian stress state c. Compute normal force on element surface d. If friction, compute tangential force e. Distribute forces to nodes and pass data back to `Presto_ITAR`
5	Execute `Presto_ITAR` and CTH

Zapotec controls the time synchronization for CTH and `Presto_ITAR`. The Zapotec time step is taken to be the same as the CTH time step. This implies the `Presto_ITAR` time step is scaled back whenever it is larger than that for CTH. In practice, this generally occurs at start-up of the calculation. In most cases, the `Presto_ITAR` stable time step is smaller than that for CTH due to the finer resolution typically encountered for the finite element mesh. Zapotec permits subcycling of the Lagrangian code to improve computational efficiency and accuracy. The time synchronization associated with the subcycling algorithm is illustrated in Fig. 1.3. The external nodal forces computed by Zapotec at t_n are applied to the Lagrangian surfaces for the remainder of the CTH time step (*i.e.*, at each of the Lagrangian subcycles).

Special care is required when Lagrangian material donation is invoked. If a Lagrangian element dies during the course of subcycling, it is necessary to redefine the Lagrangian material boundaries and immediately donate the material to the Eulerian domain. In this situation, the stable time step taken by CTH is scaled back to coincide with the time at which element death occurred. In practice, such strict control of the time step by donation is not fully necessary, and can lead to excessively small time steps. Reasonable solutions are possible by allowing several deaths to occur before cutting back the time step. See the **death_threshold** and **max_don_subcycle** commands in Section 1.6.4.2 for user options and further discussion.

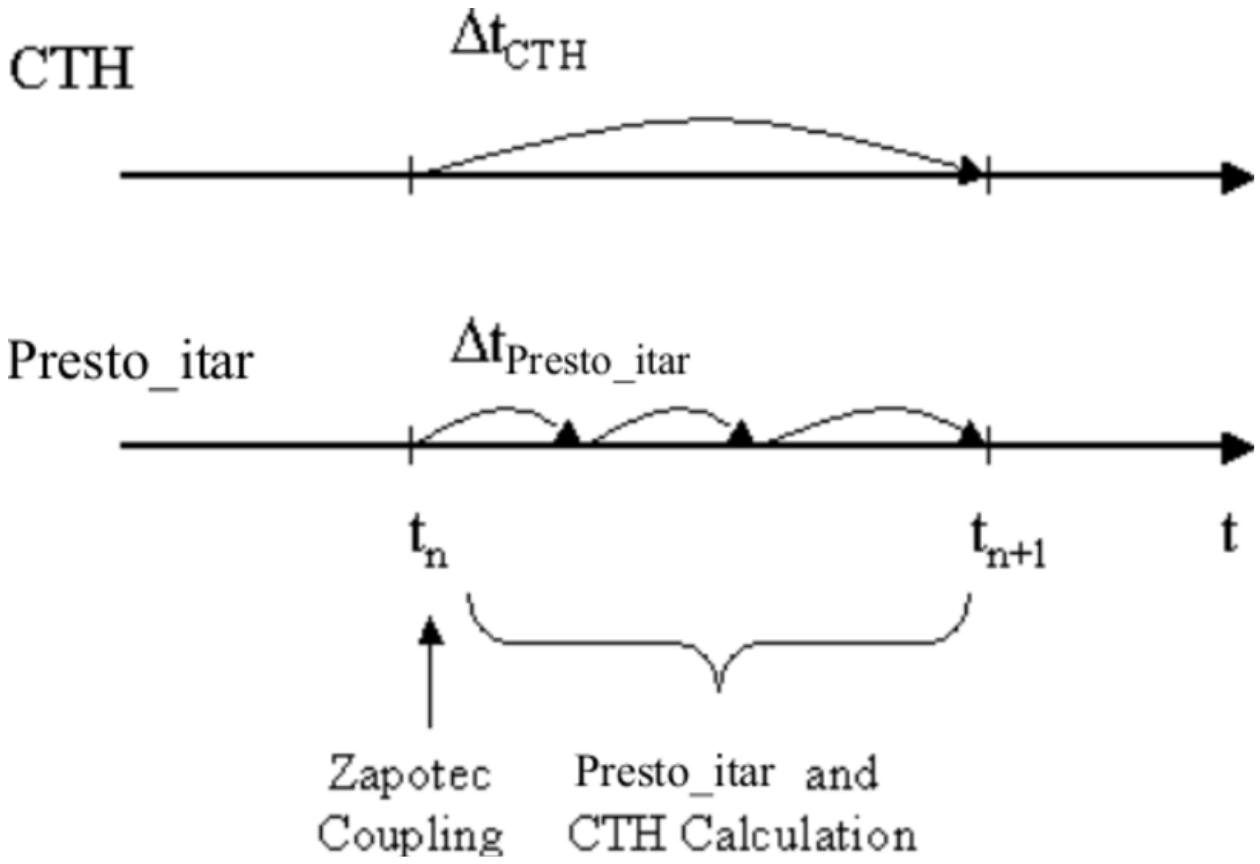


Fig. 1.3 Time synchronization of CTH and `Presto_ITAR`.

1.6.2.4 Material Insertion Algorithm

In the coupling algorithm, Zapotec first removes any pre-existing Lagrangian material from the Eulerian mesh, *i.e.*, material inserted during the previous time step. The data removed include any Lagrangian placeholder data in the CTH database as well as any momentum contribution to cells in the Eulerian mesh. Zapotec then maps the current configuration of the Lagrangian materials onto the fixed Eulerian mesh. Any overlapping Lagrangian material is inserted into the Eulerian mesh with updates made to Eulerian cell data so that momentum and mass are conserved. A description of the insertion algorithm is provided in this section. Although Zapotec allows the insertion of multiple Lagrangian materials into the Eulerian mesh, the ensuing discussion will only consider insertion of a single Lagrangian material for simplicity.

Prior to insertion, the Lagrangian finite elements are converted into a set of 4-node tetrahedral elements for Zapotec, as the determination of the volume overlap is much simpler with this geometry. For example, the 8-node hexahedral element utilized by `Presto_ITAR` is split into six 4-node tetrahedral elements. The element conversion process and associated data development is discussed in detail in Section 1.6.2.7. A description of the volume overlap computation is provided in [19]. The important point made here is that the Lagrangian material inserted into the Eulerian mesh is now represented by a region composed of tetrahedral elements. One should also note that these tetrahedral elements are only used to transfer Lagrangian material information to the Eulerian mesh and should not be confused with a true finite element.

Steps in the Algorithm

Step 1: Compute Volume Overlap Compute the volume overlap of a Lagrangian tetrahedral element with a rectangular Eulerian cell as follows:

$$V_0 = V_l \cap V_e$$

Step 2: Gather Lagrangian Data for Insertion into the Eulerian Cell The material pressure and sound speed for Lagrangian materials inserted into the Eulerian mesh is based upon a volume-weighted average of these quantities. The treatment of the deviatoric stress varies slightly since only a single deviatoric stress state is specified for an individual CTH cell, regardless of the number of materials present in the cell. In this case, the deviatoric stress state is based upon a mass-weighted average of the deviatoric stresses for all materials present in the cell. The Lagrangian momentum and energy inserted into an Eulerian cell is weighted by the volume fraction overlap ϕ_o, which is the fraction of a Lagrangian element overlapping an Eulerian cell.

The overlap data associated with a single Lagrangian element are computed as follows:

$$
\begin{aligned}
m_o &= \rho_l V_o \\
\phi_o &= V_o / V_l \\
c'_o &= c_l V_o \\
p_l &= -1/3 \text{trace}(\boldsymbol{\sigma}_l) \\
\boldsymbol{s}_l &= \boldsymbol{\sigma}_l + \boldsymbol{I} p_I \\
p'_o &= V_o (p_{ref} + p_l) \\
\boldsymbol{P}_o &= \phi_o \boldsymbol{P}_l \\
E_o &= \phi_o (E_l + E_{ref})
\end{aligned}
$$

where the prime ($'$) is used to denote weighted quantities and $\boldsymbol{I}$ is the identity tensor. Stress is defined positive in tension, while pressure is defined positive in compression.

Steps 1 and 2 are repeated for all Lagrangian elements overlapping the Eulerian mesh. The net result is a list of the above quantities that will be inserted into individual cells within the Eulerian mesh.

Step 3: Gather Eulerian Data from the CTH Database Determine the material and void volumes from the stored CTH volume fraction data ($\phi_{e,m}$ and $\phi_{e,void}$), convert material pressures and cell sound speed to volume-weighted quantities, and specific internal energy to total internal energy.

$$
\begin{aligned}
V_{e,m} = \phi_{e,m} V_e, \quad \text{where } m &= 1, \ldots N_E \\
V_{e,void} &= \phi_{e,void} V_e \\
c'_e &= c_e V_e \\
p'_{e,m} &= p_{e,m} V_{e,m} \\
E_{e,m} &= e_{e,m} m_{e,m}
\end{aligned}
$$

Step 4: Distribute the Momentum Associated with Eulerian Materials to the Cell Centers The procedure is illustrated in Fig. 1.4 for a one-dimensional example. The procedure is based on the half-index shifted (HIS) algorithm proposed by [4] as a more accurate approach for recovering the Eulerian cell velocities. The HIS-based algorithm is consistently implemented within both CTH and Zapotec. The data on the primary mesh used for the momentum distribution is outlined in Fig. 1.4 (a), where the data are comprised of the face-centered velocities $\boldsymbol{v}$ and the element-centered mass m. For clarity, the subscript e associated with the Eulerian data has been dropped. The Lagrangian mass and momentum overlapping the $i$textsuperscript{th} Eulerian cell is denoted by m_o and $\boldsymbol{P}_o$, respectively. The Lagrangian material insertion and subsequent recovery of the CTH face-centered velocities are described below. The distribution of momentum to the cell centers of the mesh is illustrated in Fig. 1.4 (b) and (c), where one notes that each face-centered velocity contributes to the cell-centered momenta of cells sharing a face. For example, the velocity on face i provides a contribution to the momenta of the forward and rear cells (cells i and $i-1$, respectively). The momentum shift to the forward and rear cells are referred to as the plus- and minus-shifted momenta, respectively.

Step 5: Insert the Lagrangian Data into the Eulerian Cell

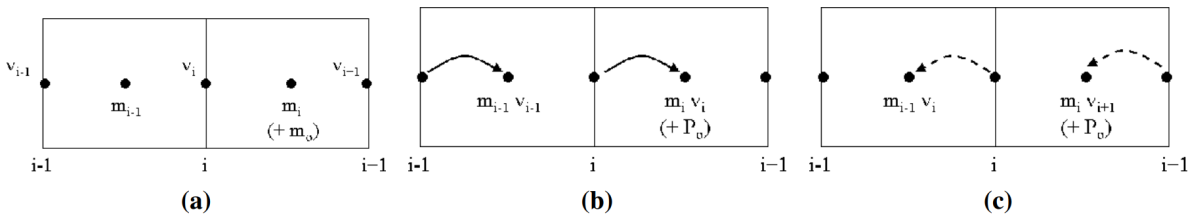


Fig. 1.4 Momentum insertion and recovery of face-centered velocities: (a) Primary mesh data before momentum shift, (b) Plus-shifted cell-centered momentum, P^+, (c) Minus-shifted cell-centered momentum, P^-.

The Lagrangian overlap data is mapped into the Lagrangian placeholder position, which is denoted by the subscript m. The Lagrangian momentum is inserted into the centered and staggered meshes as depicted in Fig. 1.4 (b) and Fig. 1.4 (c). The insertion is summarized as follows:

$$
\begin{aligned}
m_{e,m} \leftarrow m_{e,m} + m_o \quad \text{where } m = N_{E+1}, \ldots, N_{mat} \\
V_{e,m} &\leftarrow V_{e,m} + V_o \\
V_{e,void} &\leftarrow V_{e,void} - V_o \\
c'_e &\leftarrow c'_e + c'_o \\
p'_{e,m} &\leftarrow p'_{e,m} + p'_o \\
\boldsymbol{P}_e &\leftarrow \boldsymbol{P}_e + \boldsymbol{P}_o \\
E_{e,m} &\leftarrow E_{e,m} + E_o
\end{aligned}
$$

Step 5 is repeated for all Lagrangian elements overlapping the Eulerian cell. The net result is an accumulation of the inserted mass, momentum, and energy; volume-weighted sound speed and pressure for the Eulerian cell.

Step 6: Recover the Face-Centered Velocities for the Eulerian Mesh

The procedure is illustrated for a simple one-dimensional example in Fig. 1.4. As shown, the velocity recovery requires access to the HIS momenta in adjacent cells.

Step 7: Adjust Over-Filled Cells

Before recovering the Eulerian data and returning it to the CTH database, we must first consider the case of an over-filled cell, where the volume of inserted Lagrangian material along with the resident Eulerian material exceeds the Eulerian cell volume. In theory, the coupling would result in a fully compliant interface between Eulerian and Lagrangian materials. In practice, this is not the case. There will be small amounts of overlap between Eulerian and Lagrangian materials, leading to over-filled cells as part of the material insertion step. Zapotec currently uses a cutout approach in which the overlapping material (both Eulerian and Lagrangian) is removed from the problem. The cutout methodology is summarized in the following.

First, determine a weighting factor $w_{e,m}$ for each material m used to correct the material volumes

such that the total volume of materials within the cell does not exceed the total cell volume:

$$w_{e,m} = V_{e,m}/K_m \quad \text{where } m = 1, \ldots, N_{\text{mat}}$$

$$\varphi_{e,m} = w_{e,m} / \sum_{m=1}^{N_{\text{mat}}} (w_{\text{e,m}})$$

where K is the initial bulk modulus of the material.

Then, determine the material volume correction and update the current material volume:

$$\Delta V_{e,m} = \varphi_{e,m} V_{e,void}$$
$$V_{e,m} \leftarrow V_{e,m} + \Delta V_{e,m}$$

This scheme will lead to loss of mass from the problem. The algorithm as implemented in the code has substantial complexity as it attempts to minimize the negative effects of over-filled cell resolution.

The central issue is the loss of Eulerian material which serves to either resist or push on the Lagrangian material it is contacting. To avoid excess mass loss and ensure compliance at the Euler/Lagrange interface, a penalty force algorithm has been developed. The magnitude of this force is proportional to the overfilled cell volume, and the net intended effect of its application is to reduce the amount of mass loss and ensure compliance between the two material types. The penalty algorithm is experimental, and in its current incarnation can lead to spurious forces applied at odd times. The details of the penalty algorithm are discussed below, but great care should be taken in its use.

In many cases, the errors associated with overfilled cells do not appear to greatly affect solution quality. However, more metrics available to the user are needed to better assess the effect of these algorithms on code solutions. This will be the subject of future development work.

Step 8: Recover the Eulerian Data to be Returned to the CTH Database

This involves recovering the volume-weighted cell sound speed and material pressures, the volume-weighted cell pressure, the mass-weighted deviatoric stresses, the specific internal energy, and material volume/void fractions:

$$c_{\text{e}} = c'_{\text{e}} / \sum_{m=1}^{N_{\text{mat}}} V_{\text{e,m}},$$

$$p_{\text{e}} = \sum_{m=1}^{N_{\text{mat}}} p'_{e,m} / \sum_{m=1}^{N_{\text{mat}}} V_{\text{e,m}},$$

$$e_{\text{e,m}} = E_{\text{e,m}}/m_{\text{e,m}},$$

$$f_{\text{e,m}} = V_{\text{e,m}}/V_{\text{e}},$$

$$f_{\text{e,void}} = V_{\text{e,void}}/V_{\text{e}}.$$

Remark 1: Although the internal energy associated with the Lagrangian overlapping material is inserted into the CTH database, it is not actually used within CTH since the CTH EOS is never called for placeholder Lagrangian materials.

1.6.2.5 Material Donation Algorithm

The `Presto_ITAR` code provides an element death capability to remove elements from the Lagrangian calculation once a prescribed death criterion is reached. Zapotec has the ability to donate the newly dead Lagrangian elements to the Eulerian problem domain. Essentially, the donated elements form (or become part of) a new Eulerian material. Donated Lagrangian material is inserted into the Eulerian mesh using the same algorithm for intact Lagrangian materials. The only difference between donation and insertion is that the internal energy passed back to CTH is actually used in the equation of state calculations. As such, it is important to ensure consistency of the material state computed by `Presto_ITAR` and that passed into CTH for future calculations.

The donation algorithm ensures consistency of the pressure-energy state by iterating on the internal energy. This is done by successive calls to the CTH equation of state routine, whereby a bisection method is used to iterate on the internal energy until the pressures computed by CTH and `Presto_ITAR` are consistent. Material donation is an important algorithm useful for many Zapotec applications. However, great care should be taken to choose material definitions carefully to ensure consistency between the Lagrangian and Eulerian domains.

> **Warning:** Donation will be wrong if it occurs before insertion. An element must go through insertion before being donated to be correctly contributed to the CTH domain. This means element death occurring at initialization will be wrong.

1.6.2.6 Force Application Algorithm

General Algorithm

Once the material insertion is complete, the external loading on a Lagrangian material surface is determined from the stress state in the neighboring Eulerian material. This procedure is illustrated in Fig. 1.5. The cell overlapped by the Lagrangian material surface is referred to as the mixed Euler/Lagrange (E/L) cell. The adjacent, or forward, cell is usually comprised of only Eulerian material. The location of the forward cell is determined using the outward pointing normal associated with the Lagrangian surface element. The Eulerian stress is derived from a weighted average of the stress in the mixed E/L and forward cells. By default, a linear weighting based on the volume fraction of Eulerian material in the mixed E/L cell (Φ_{mix}) is assumed. Once the Eulerian stress state is determined, it is straightforward to determine the traction $\boldsymbol{t}$ on the surface element using the element's outward normal $\boldsymbol{n}_{\text{l}}$. In turn, the normal force $\boldsymbol{f}_n$ applied to the surface element is derived from the traction and overlapping element surface area A_{o}. The element-centered force is then distributed to the nodes. This process is repeated for all surface elements, resulting in a set of nodal forces which are passed back to `Presto_ITAR` as an update. Once the coupled treatment is complete, both CTH and `Presto_ITAR` are run independently over the next time step with their updated data.

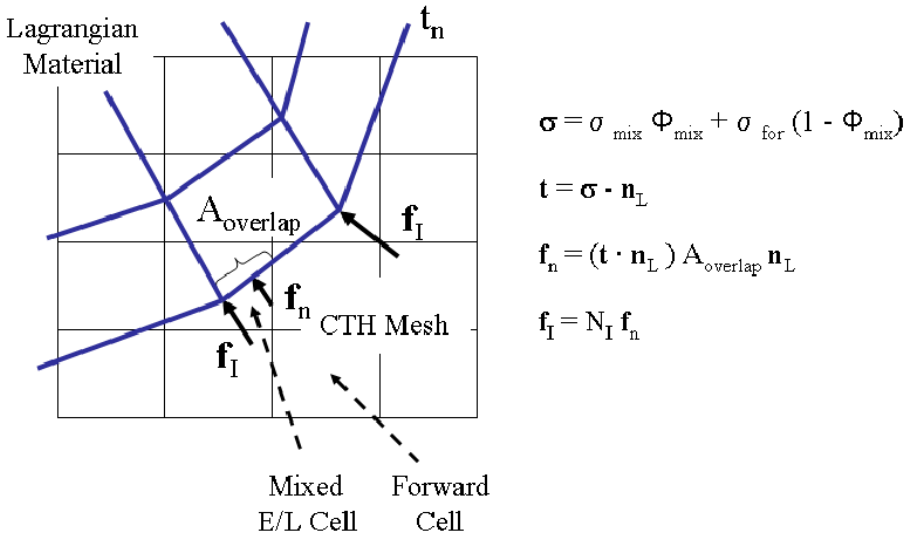


Fig. 1.5 Illustration of force application.

Zapotec makes several assumptions regarding the determination of the applied normal force f_n. For inserted Lagrangian materials, the force calculation assumes that a compressive loading ($\sigma_n < 0$) results from contact between the Lagrangian and Eulerian materials along the interface. Conversely, if the Eulerian material is in a state of tension (*i.e.*, $\sigma_n > 0$), the contacting materials are assumed to be separating and no load is transmitted to the Lagrangian material. The normal force applied to the Lagrangian material is calculated as follows:

$$\begin{aligned} \boldsymbol{f}_n &= \sigma_n A_o \boldsymbol{n} \quad \text{if } \sigma_n < 0, \\ \boldsymbol{f}_n &= \boldsymbol{0} \qquad\quad \text{if } \sigma_n \geq 0. \end{aligned}$$

Zapotec has the capability to evaluate frictional contact, where friction has the effect of retarding the relative motion between the Lagrangian and Eulerian materials. The friction treatment is based on a Coulomb friction model, where the user can prescribe either a constant coefficient of friction between materials or utilize a velocity-dependent friction model. The model is discussed in more detail in the next section.

For a Zapotec analysis, the user must prescribe *a priori* the Lagrangian material surfaces that can interact with Eulerian materials. Lagrangian material surfaces are defined using finite element blocks with the block Exodus ID, referred to in the Zapotec input deck as “materials”. The discrete Eulerian/Lagrangian material boundary is represented by a triangular faceted surface of the discretized Lagrangian domain, and is communicated from `Presto_ITAR` to Zapotec. The boundary conversion to a faceted surface, and associated data development, is discussed in Section 1.6.2.7.

Friction Treatment

The friction treatment is outlined in the following.

Step 1: Assess Friction

If friction is considered, compute the tangential friction force $\boldsymbol{f}_t$ applied to the surface of the Lagrangian element. First, determine the direction of sliding between the Lagrangian and Eulerian materials as follows:

$$\boldsymbol{v}_s = \boldsymbol{v}_r - (\boldsymbol{n} \cdot \boldsymbol{v}_r)\boldsymbol{n},$$

where $\boldsymbol{v}_s$ is the sliding velocity and $\boldsymbol{v}_r$ is the relative velocity between the Lagrangian and Eulerian materials. The sliding direction $\boldsymbol{s}$ is derived from the sliding velocity as:

$$\boldsymbol{s} = \boldsymbol{v}_s / |\boldsymbol{v}_s|.$$

The determination of the relative velocity can be problematic, a consequence of difficulties in recovering the Eulerian velocity. An interpolation of the face-centered Eulerian velocities is required to determine the velocity at a point of contact (assumed to be at the centroid of the Lagrangian surface element). This was considered in the original implementation; however, some unforeseen problems were encountered. In particular, the area intersection algorithm did not readily permit the interpolation of velocities outside of the overlapped cell (this occurs when the element overlaps multiple cells with the surface element centroid residing in a neighboring cell). In practice, this limitation leads to unreasonable (and non-physical) values for the interpolated velocities. In addition, resolution of the Eulerian mesh was found to significantly affect the interpolated velocities.

To avoid these difficulties, several simplifying assumptions were made for the determination of the relative velocity vector. For the case of a Lagrangian body moving into an Eulerian material, the relative velocity is based on the Lagrangian velocity at the centroid of the element:

$$\boldsymbol{v}_r = -\boldsymbol{v}_{\mathrm{L}}$$

This simple assumption is reasonable since the Lagrangian material is generally stronger than the surrounding Eulerian material and its surface is well defined. For the case of an Eulerian material impacting a stationary (or rearward moving) Lagrangian material, the relative velocity is computed as:

$$\boldsymbol{v}_r = \boldsymbol{v}_{\mathrm{E}} - \boldsymbol{v}_{\mathrm{L}},$$

where $\boldsymbol{v}_{\mathrm{L}}$ is the Lagrangian velocity at the surface element centroid and $\boldsymbol{v}_{\mathrm{E}}$ is taken as the cell-centered (*i.e.*, average) velocity for the mixed E/L cell.

Once the sliding direction is determined, the tangential friction force $\boldsymbol{f}_t$ is computed as:

$$\boldsymbol{f}_t = \min(\mu\sigma_n A_{\mathrm{o}}, f_{\mathrm{t,max}})\boldsymbol{s} \equiv f_t \boldsymbol{s},$$

where $f_{\text{t,max}}$ is the maximum allowed tangential force which serves to limit the relative motion at the interface to a tied boundary condition (*i.e.*, perfect sticking condition). The maximum allowed tangential force is computed as:

$$f_{t,max} = |\boldsymbol{v}_s| m/\Delta t,$$

where m and Δt are the lumped nodal mass at the element centroid and the time step, respectively.

Step 2: Apply Frictional Forces to Surface Nodes

Distribute the element-centered external forces to the nodes of the Lagrangian surface element:

$$\boldsymbol{f}_I = N_I(\boldsymbol{f}_n + \boldsymbol{f}_t),$$

where I refers to the nodes on the surface element ($I = 1, 2, 3$) and N is an interpolation function ($N = 1/3$ for the triangular surface elements). The nodal forces are accumulated over the Lagrangian surface and passed back to `Presto_ITAR` as a set of external nodal forces.

1.6.2.7 Special Topics

Lagrangian Code Interface

Zapotec accesses the Lagrangian data at every time step. A summary of the data used by Zapotec is provided in Table 1.14. The data are categorized as node, element, or surface element data. With each data type, there are two associated variables for the size (quantity) of data on a given processor. For example, consider the nodal variables `nsizel` and `max_nsizel`. The variable `nsizel` reflects the actual number of nodes on a processor while `max_nsizel` reflects the maximum number of nodes allowed on any processor.

The nodal coordinates and velocities (*e.g.*, the data residing in `xl` and `xvel`) are passed directly to Zapotec. The element data, however, require some manipulation prior to being used by Zapotec. The data manipulation is performed at the Lagrangian code interface. At present, Zapotec supports a limited set of `Presto_ITAR` element types. These are the Flanagan-Belytschko 8-node constant strain hexahedral element and the 4-node constant strain quadrilateral shell element [32]. A general description of the data manipulation follows. Shell elements require special treatment, which is discussed in the next section.

The parent (or original) finite elements are converted into a set of 4-node tetrahedral elements for Zapotec, as the determination of the volume overlap is much simpler with this geometry. For example, the 8-node hexahedral element utilized by `Presto_ITAR` is split into six 4-node tetrahedral elements. For a particle element, the radius is used to create an icosahedron around the particle center. The icosahedron is then converted into twenty 4-node tetrahedral elements for Zapotec. A similar operation is performed for the Lagrangian material surface definitions, where the original discrete surface is converted into a list of 3-node triangular surface elements. The conversion process requires the development of new connectivity arrays for both the element and

surface lists. Connectivity arrays relate the global node IDs to the local ordering of nodes for a given element type. The dimensions for the element-based arrays (`lsizel` and `nsrfl`) reflect the "on the fly" creation of the tetrahedral/triangular elements for Zapotec.

The newly created tetrahedral elements inherit data from the parent element (*e.g.*, the stress state, density, and internal energy of the parent element is inherited by the newly created tetrahedral elements). A mapping is generated to track the inheritance of the parent data (see `map_lparent`). The only additional data generated for the tetrahedral elements are the element volumes and momenta. Element momentum is not typically stored (or computed) in finite element codes. However, the Zapotec material insertion/donation algorithm requires these data for the update of Eulerian mesh velocities. The element momentum is computed on the fly and reflects the momentum associated with each of the newly created tetrahedral elements. The momentum for a given tetrahedral element is computed as follows:

$$\boldsymbol{P}_{\text{tet}} = \rho_l V_{\text{tet}} \boldsymbol{v}_{\text{avg}},$$

where $\boldsymbol{P}_{\text{tet}}$ is the element momentum, ρ_l is the density of the parent element, V_{tet} is the computed tetrahedral element volume, and $\boldsymbol{v}_{\text{avg}}$ is the average velocity (computed as the average of the nodal velocities for the four nodes associated with the tetrahedral element).

The lumped nodal mass vector `xmass` passed to Zapotec differs from that stored by `Presto_ITAR`. The data passed to Zapotec reflect the lumped mass associated with the newly created tetrahedral elements, while the data stored in `Presto_ITAR` reflect the lumped mass associated with the parent elements.

Table 1.14 Lagrangian Data Used Within Zapotec

Zapotec Variable Name	Description
(max_) nsizel	(max) number of nodes
(max_) lsizel	(max) number of 4-node tet elements
(max_) lparent	(max) number of parent elements
msrfl	number of surface elements
max_msizel	max number of surface elements
xl, yl, zl	nodal coordinates, dimensions (1:max_nsizel)
xvel, yvel, zvel	nodal velocities, dimensions (1:max_nsizel)
xmass	nodal mass, dimensions (1:max_nsizel)
nodexl	connectivity for inserted 4-node tet elements, dimensions (1:4, 1:max_lsizel)
matxl	element material ID, dimensions (1:max_lsizel)
map_lparent	Parent element to tet map (1:max_lsizel)
sxl, syl, szl	element total stress, dimensions (1:max_lparent)
sxyl, sxzl, syzl	
denl	element density, dimensions (1:max_lparent)
csl	element sound speed, dimensions (1:max_lparent)
esl	element internal energy, dimensions (1:max_lparent)
idonate	element status flag, dimensions (1:max_lsizel):
	= 0, element is alive
	= 1, element has just died, time to donate element
	= 2, element is already dead
nsrfxl	connectivity for surface triangular elements, dimensions (1:3, 1:max_msizel)

`Presto_ITAR` tracks the status of an element when element death is invoked. The `Presto_ITAR` element status ranges from zero to one, with zero denoting that an element has died and will be (or has been) removed from the calculation. The user can prescribe the number of time steps required to kill an element, with the status decremented until a zero value is reached.

Zapotec needs this information when the donation algorithm is invoked. The donation vector `idonate` tracks the element status, differentiating newly dead elements (`idonate = 1`) from those which have already died and been donated (`idonate = 2`).

Treatment of Shell Elements

Thin-shell structures pose significant challenges for coupled methods. The major difficulty is tracking material interfaces by the Eulerian code. Typically, material interface reconstruction algorithms utilized by Eulerian hydrocodes rely on volume fraction information to approximate the location and orientation of material interfaces and do not explicitly track the material interfaces. Such interface reconstruction algorithms tend to work well for solid or thick-skinned bodies whose material thickness spans several Eulerian cells; however, problems arise when the material thickness is on the order of the Eulerian cell width or less. In these situations, there is no information from either the upstream or downstream cells from which to reconstruct the material interfaces. A simple alternative for avoiding this problem is to refine the Eulerian mesh such that several cells span the thickness of the thin body. Unfortunately, this approach is impractical for most 3D applications as memory requirements and analysis times become unmanageable.

A shell reconstruction algorithm has been implemented into Zapotec to accommodate thin-shell structures where the Lagrangian domain is discretized using shell elements. Using this algorithm, the shell thickness is allowed to be smaller than the corresponding Eulerian cell width. The shell reconstruction algorithm creates an effective material interface for the Eulerian code by *lifting* the shell geometry into 3 dimensions using the shell *thickness*. A mathematical *lifting* operator is the right inverse to the more familiar *trace* operator. In the finite element mesh, the shell element has an attribute of *thickness* with the surface geometry defined by the location of the vertex nodes.

"Fluffing" can be used to increase the shell's volumetric representation by using an effective thickness rather than the actual thickness. Once the shell has a volume, it can then be inserted into the Eulerian mesh. The external loading on the outer surface of the lifted shell is then evaluated and passed back to `Presto_ITAR` as a set of external nodal forces. The user can control both the direction for fluffing the shell and the side on which loads are applied. It should be noted that using fluffed shell elements may return inaccurate results due to exploding elements. User options are discussed in Section 1.6.4.2. At present, the shell algorithm only supports 4-node quadrilateral shell elements.

The following procedure outlines the shell reconstruction algorithm for a single Lagrangian shell element:

Step 1: Determine the Effective Shell Thickness

The effective shell thickness is denoted by $t_{\text{eff,shell}}$. By default, Zapotec assumes an effective shell thickness based on the minimum effective CTH cell width for all cells in the mesh. The effective thickness is computed as:

$$t_{\text{eff,shell}} = l_{\text{cell}} = \sqrt{w_x^2 + w_y^2 + w_z^2},$$

where w_x, w_y, and w_z are the widths of Eulerian cells along the x, y, and z coordinate axes. The default effective thickness fluffs the shell to span at least two CTH cells.

Zapotec provides a number of options for scaling the effective shell thickness. First, a user-defined scale factor can be applied to the computed effective shell thickness as follows:

$$t_{\text{eff,shell}} = (\text{scale}) l_{\text{cell}},$$

where "scale" is a user input described later in Section 1.6.4.2 (see the `scale` command). The user also has the option to explicitly define the effective shell thickness (see the `tscale` command). This latter option allows the user to define an effective shell thickness that is independent of the CTH mesh resolution and is recommended over using "scale". This is also recommended if the shell is smaller than the CTH cell size. Early development of the shell reconstruction algorithm assumed the effective shell thickness should span at least two CTH cells to avoid any "bleed-through" of Eulerian material. Recent tests of the algorithm indicate that bleed-through of material does not occur for blast applications, even when the effective shell thickness is an order of magnitude less than the CTH cell size. Thus, in many applications, the effective shell thickness can be specified as the true shell thickness using the `tscale` command. A special case exists for donated shell elements, where the effective thickness will automatically be defined as the true thickness.

Step 2: Create the Fluffed Shell

This procedure is illustrated in Fig. 1.6 using a simple 2D example. In this step, the thin shell element is essentially transformed into a hexahedral element, having a significantly larger volume. In order to complete the transformation process, it is necessary to determine a direction for extending the shell thickness. This direction is based upon the average surface normal, $\boldsymbol{n}$, which is computed as:

$$\begin{aligned} r' &= (\boldsymbol{x}_2 - \boldsymbol{x}_1) + (\boldsymbol{x}_3 - \boldsymbol{x}_4) \\ \boldsymbol{s}' &= (\boldsymbol{x}_4 - \boldsymbol{x}_1) + (\boldsymbol{x}_3 - \boldsymbol{x}_2) \\ \boldsymbol{r} &= \boldsymbol{r}'/|\boldsymbol{r}'| \\ \boldsymbol{s} &= \boldsymbol{s}'/|\boldsymbol{s}'| \\ \boldsymbol{n} &= \boldsymbol{r} \times \boldsymbol{s} \end{aligned}$$

Once the average surface normal is determined, it is a straightforward procedure to determine the corners of the fluffed shell. In the algorithm, new nodes are created at the corners of the fluffed shell, which in turn, are used to define the top and bottom surfaces of the fluffed shell (see Fig. 1.6 for definitions). It is assumed that the outward pointing average surface normal defines the top surface of the fluffed shell.

Step 3: Convert the Fluffed Shell into Tetrahedral Elements

Convert the fluffed shell (hexahedron) into six tetrahedral elements for insertion into the Eulerian mesh. The procedure is comparable to that used for solid elements with some exceptions. First, shell elements have stresses computed at each of the through-thickness integration points. The insertion algorithm requires a single stress state. By default, the algorithm assumes a stress-free

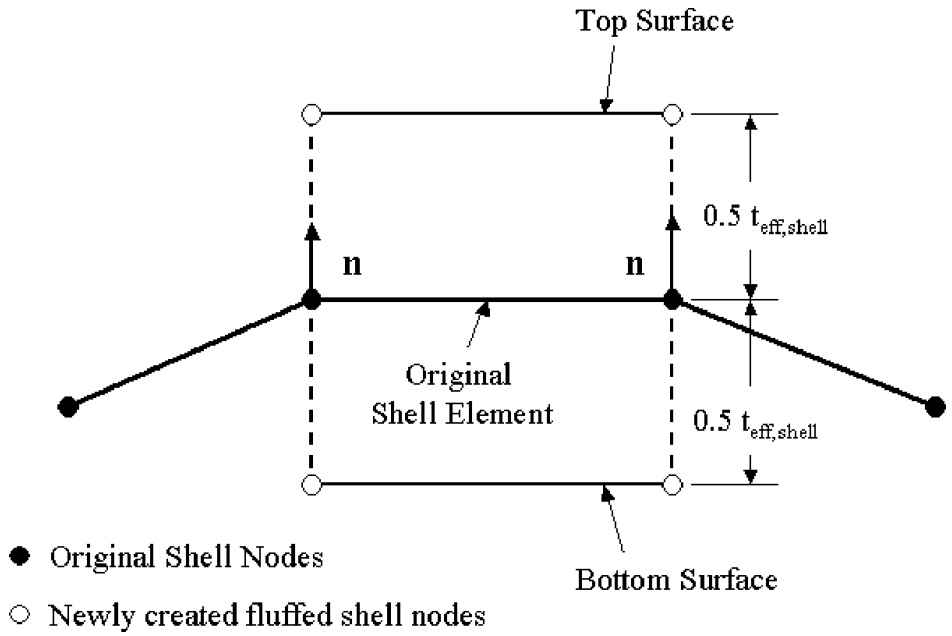


Fig. 1.6 Illustration of shell reconstruction algorithm.

shell that is at its reference energy and density state. The fluffed shell geometry is based on the current configuration. The user has the option to override the default and use either a stress state based on an average over all integration points or the stress state at an individual integration point (see the `integ` command). In general, the shell element will have 3 to 5 integration points through the thickness, with the integration points ordered from the bottom to the top of the shell element.

In addition, the material density can be modified to conserve the inserted mass and momentum (see the `conserve` command), where the inserted density is scaled as follows:

$$\rho_{\text{inserted}} = \rho_{\text{shell}} V_{\text{shell}} / V_{\text{inserted}}.$$

When the conservation option is invoked, the shell density ρ_{shell} is the current shell density. One issue that arises with the conservation option is that exceedingly low-density material can be inserted into the CTH mesh. The inserted density and energy state will be inconsistent (as far as CTH is concerned). It is unclear as to how this affects the CTH calculation. Inaccuracies associated with inserting inconsistent material state data are thought to outweigh any inaccuracies with the mass and momentum insertion. Thus, it is generally recommended that conservation not be applied.

Step 4: Define Exterior Surface for the Fluffed Shell

The user can define the surface (*e.g.*, top, bottom, or both) that can interact with Eulerian materials. The external loading on the prescribed surface of a fluffed shell is evaluated as outlined in Section 1.6.2.6. The applied nodal forces are mapped from the surface nodes of the fluffed shell onto the nodes of the original shell element. The mapped forces are then returned to `Presto_ITAR` as a set of external nodal forces.

Remark 1: As a convenience, surface definitions for the fluffed shells are generated automatically within Zapotec. The surface definitions are used for the force application. There is no need for the user to develop shell surface definitions, such as side sets, in the model file generation.

Remark 2: The shell reconstruction algorithm is useful for creating an artificial material interface for thin-shell structures interacting with Eulerian materials. It should not be considered as a substitute for contact between Lagrangian materials.

Remark 3: Zapotec makes no attempt to check for overlapping fluffed shell elements. This can potentially lead to errors when inserting materials with highly curved thin-shell structures or structures having irregular sections (*e.g.*, T-sections). To illustrate this problem, consider the curved structure shown in Fig. 1.7. Fluffed shell elements will overlap on the inside of the circular structure. Conversely, there will be gaps in the material insertion on the outside of the circular structure. Thus, the distribution of the shell data in the CTH mesh will not be correct.

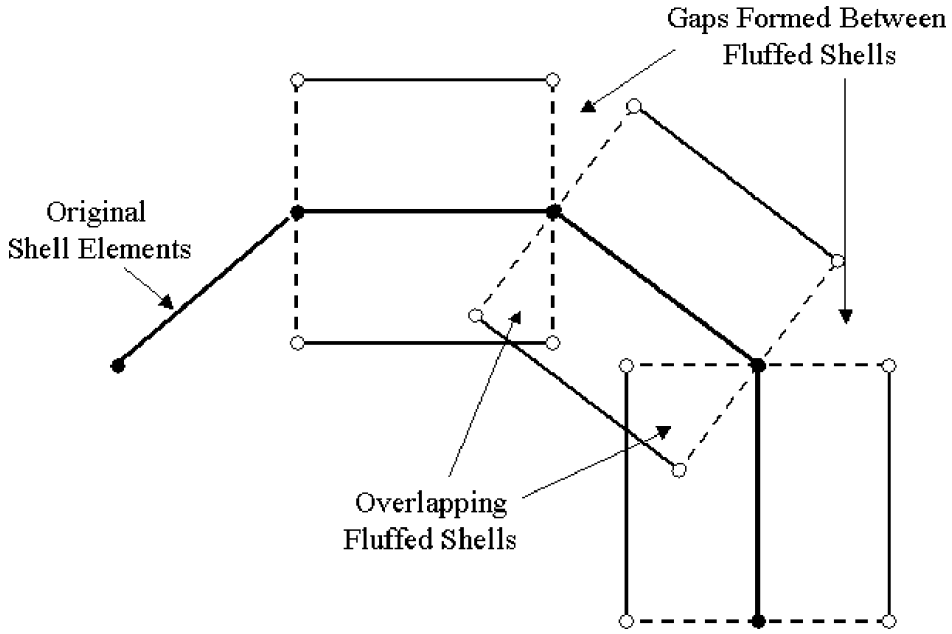


Fig. 1.7 Illustration of shell reconstruction algorithm for curved structures.

Remark 4: Zapotec makes no attempt to check for overlaps between fluffed shell elements and other materials in the problem. For example, consider the joint between a thin-shell structure and a solid body composed of hexahedral elements, which is modeled as a simple pinned connection (see Fig. 1.8). If the shell structure deforms and starts to collapse onto the solid body, it is possible for fluffed shell elements to overlap portions of the solid body. This can lead to over-filled cells in the vicinity of the joint. In turn, this can lead to over-compressed materials in this region. There is no ready fix for this problem and the user should exercise care with interpretation of results when this situation arises.

Remark 5: The default treatment for the shell insertion is the recommended option. One would like to consider the current state of the shell for insertion; however, this has proven troublesome in practice. Difficulties arise when the parent shell element becomes highly distorted, either through distention, hourglassing, or warpage. The state data associated with the highly distorted parent

shell element is inaccurate. This inaccurate data gets passed to CTH during the material insertion step. The inserted data can affect the accuracy of the CTH calculation as it moves forward over the next time step. In practice, poor states are noted in localized regions of the CTH mesh, which in turn, affect the time step computation. The net effect is usually a degradation of the CTH stable time step. Recent testing of the algorithm suggests the insertion of a "stress-free" shell does not affect the accuracy of the Zapotec calculation and avoids time step issues in CTH.

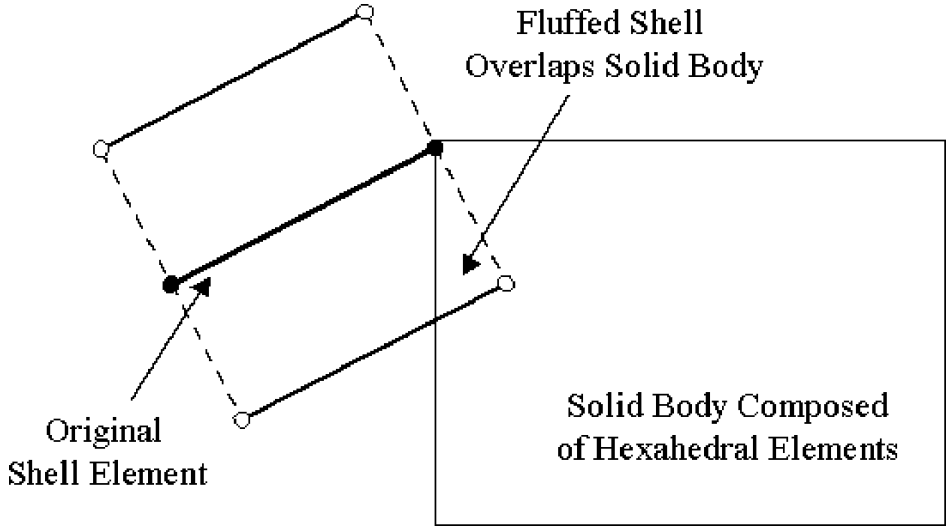


Fig. 1.8 Illustration of fluffed shell overlapping a solid structure.

Penalty Force Algorithm

The mismatch of velocity fields in the Eulerian and Lagrangian domains is thought to be a prime contributor to the "overfilled cell problem." One approach to mitigating the overfilled cell problem is to apply a penalty force to the surface of a Lagrangian material. Penalty methods are widely used in finite element analysis, generally for the imposition of displacement or contact constraints. The penalty algorithm can be viewed as a set of interface springs, providing an approximation for a displacement (or contact) constraint along the interface. The net effect is to move the Lagrangian material slightly to reduce the amount of overfilling. The penalty force is in addition to that computed during the force application step. Although there is no unique way of calculating a penalty force, this force should exhibit the characteristic that its magnitude increases with the degree of overfilling.

The penalty algorithm is experimental and can lead to spurious forces at various times in the analysis. It should be used with great care in its current form. Usually, it is recommended to leave the penalty algorithm off or to at least reduce its scale factor to prevent erroneous forces from adversely affecting the solution.

In the Zapotec implementation, a penalty pressure is first computed using the volume-weighted bulk modulus of the Eulerian materials residing in the cell, denoted K_{eul}, and the overfilled cell

volume V_o as follows:

$$p_{\text{penalty}} = c_{\text{penalty}} K_{\text{eul}} \frac{V_o}{V_{\text{eul}}},$$

where c_{penalty} is a user-prescribed constant coefficient and V_{eul} is the volume of Eulerian material residing in the cell. The bulk modulus used in the algorithm is the initial bulk modulus of the Eulerian material. This could be an issue if the bulk modulus changes significantly with the volumetric strain.

The penalty pressure is applied to the computed Eulerian stress tensor, which in turn, is used in computing the traction vector and normal force. This is outlined as follows:

$$\begin{aligned} \boldsymbol{\sigma}'_{\text{E}} &= \boldsymbol{\sigma}_{\text{E}} - p_{\text{penalty}} \boldsymbol{I}, \\ \boldsymbol{t} &= \boldsymbol{\sigma}'_{\text{E}} \cdot \boldsymbol{n}_L, \\ \sigma_n &= \boldsymbol{t} \cdot \boldsymbol{n}_L, \\ \boldsymbol{f}_n &= \sigma_n A_o \boldsymbol{n}. \end{aligned}$$

The penalty pressure (hence force) is directly proportional to the user-defined penalty coefficient c_{penalty}. Recent testing of the algorithm suggests coefficients ranging between 0.1 and 0.2 are reasonable for most applications.

1.6.3 Parallel Implementation

The material insertion, donation, and force application algorithms are implemented in parallel. The parallel implementation is load-balanced and allows for efficient computation when running large simulations. A parallel analysis requires partitioning of the Lagrangian and Eulerian meshes among the processors. The load balance is not straightforward since the mesh partitioning and manner of partitioning differs for CTH and `Presto_ITAR`. CTH utilizes a block-based approach for mesh decomposition [3]. `Presto_ITAR` maintains a different mesh decomposition based on the initial mesh decomposition, and in some cases maintains a second decomposition for contact surfaces [32]. An illustration of the different domain decompositions is provided in Fig. 1.9. A static, graph-based method is used to decompose the entire `Presto_ITAR` mesh at startup of the calculation. This mesh decomposition is maintained for the duration of the calculation. Contact surfaces require special treatment as they evolve over time. For some contact options in `Presto_ITAR`, a geometric-based decomposition is utilized. Further details regarding the parallelization of CTH and `Presto_ITAR` can be found in [3]] and the `Presto_ITAR` theory manual [[32], respectively.

The material insertion step requires determining the volume overlap of Lagrangian elements with Eulerian cells on each processor. An efficient parallel implementation of this volume overlap calculation depends on the balanced division of work among processors. If the overlap calculation is performed only on processors where overlaps occur, then many processors will be idle. To overcome this load imbalance, some of the work from processors with overlaps is distributed to idle processors. In the Zapotec terminology, processors having overlaps are termed masters, while idle processors are termed workers. The load balance algorithm for the material insertion step (*i.e.*, volume overlap calculation) is summarized as follows:

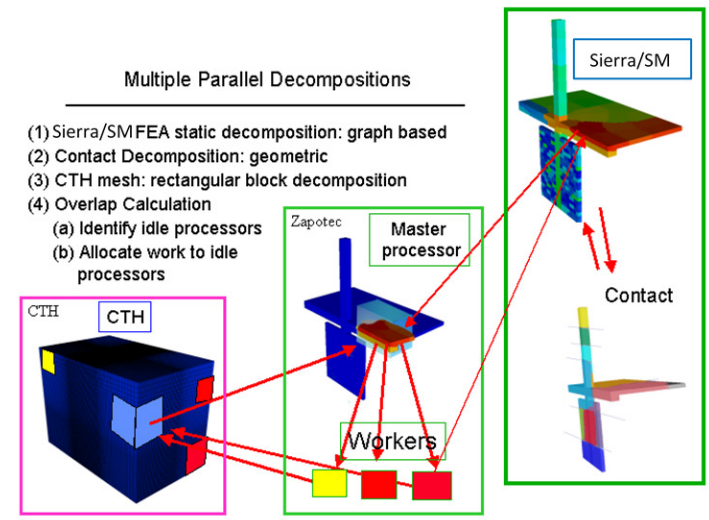


Fig. 1.9 Mesh decomposition and load balance for Zapotec coupling algorithm.

1. Develop a bounding box about Lagrangian elements on each processor to determine the geometric extents of Lagrangian elements owned on each processor. Broadcast the bounding box information to all processors. Each processor compares the bounding box for their Eulerian mesh against all of the Lagrangian bounding boxes to develop a list of the portions of the Lagrangian mesh that overlap with the owned portion of the Eulerian domain. Note that mesh overlaps can vary over the duration of the calculation due to the ability of the Lagrangian bodies to move through the fixed Eulerian mesh.
2. Perform the volume overlap calculation to identify the volume overlap of individual Lagrangian elements with each Eulerian cell.
    a. Identify processors having volume overlaps (termed masters).
    b. Identify idle processors (termed workers).
    c. Assign workers to masters. Workers are assigned in pairs to each master. If there is an abundance of workers, then multiple pairs can be assigned to a given master processor. If there are not enough workers available to provide a master with at least two workers, then the master processor will perform its own overlap calculation.
    d. Broadcast the master-worker assignments globally.
    e. Communicate the Eulerian mesh coordinate information for a master processor to its workers. If there are multiple workers available for a master processor, portions of the Lagrangian data are communicated and put into temporary memory on the assigned workers.
    f. Master and worker processors perform their assigned volume overlap calculations.
    g. As workers complete their overlap calculations, they communicate the results back to their master.

3. Once all of the overlap data is received from the workers, the master processors complete the material insertion step, *i.e.*, map the Lagrangian material mass, momentum, stress, *etc.* into the Eulerian mesh.

For many applications, the number of mesh overlaps will be larger than the number of processors. As a result, the master-worker pairings cannot be done at once. In this situation, we perform the parallel volume overlap calculations in a series of steps. In each step, the master and worker processors are identified and the overlap calculations are distributed. Each master is first given two workers. Excess workers are then assigned to the master processors with the largest number of remaining mesh overlaps that will not be satisfied this step. This process is repeated until all workers are assigned or no master processor has a mesh overlap to process. If there are still workers available, then they are assigned in a round-robin method, one for each mesh overlap. In this manner, the number of steps needed to complete the volume overlap calculations is reduced.

The process of material donation of dead elements from the Lagrangian mesh into the Eulerian mesh is similar to that of material insertion. The division of labor for master and worker processors is simpler in this case since there are fewer volume overlaps to consider. First, each processor determines the number of donated Lagrangian elements that it owns and broadcasts this information to all other processors. Then, each processor traverses its list of mesh overlaps and filters out insertion elements, *i.e.*, only donated elements remain in the list. The procedure for performing the volume overlap calculation follows that for material insertion.

The external loading algorithm determines the forces exerted on the surface of the Lagrangian elements by the Eulerian materials and has many similarities to the material insertion algorithm. As discussed previously, Lagrangian material surfaces are discretized using triangular elements, and thus the volume overlap computed for material insertion is replaced with area intersection for external loading. Once the Lagrangian area intersection is determined, the applied forces are computed based on the stress state in the Eulerian material.

The external loading algorithm also requires additional parallel communication steps. Each master processor maintains an array of force vectors. As the forces on the surfaces overlapping with the Eulerian cells are computed, the master processes accumulate them into a vector corresponding to the Lagrangian processor that owns the surface mesh. At the end of each step, the master processors send the forces to the appropriate Lagrangian processors, where they are accumulated in the locally-owned force vector. After all of the surfaces have been processed, Zapotec allows CTH and `Presto_ITAR` to continue.

1.6.3.1 Improved Material Insertion Algorithm

The material insertion algorithm previously discussed, including the master-worker load balancing, is known to scale poorly. Once a (simulation-dependent) processor count is reached, addition of more HPC resources can actually *increase* the runtime. An alternative material insertion algorithm is available to address these performance and scalability issues. The improved insertion scheme is conceptually simpler than the standard algorithm discussed previously. Both methods begin by checking for bounding box overlaps. The new method then sends data for each CTH block to the `Presto_ITAR` subdomains that overlap with it. Volume overlap computations

are performed on the Presto_ITAR partitions, and results are sent back to the CTH decomposition for insertion into CTH. The rationale behind this alternative is that typically all (or at least most) of the finite element domain is immersed in the CTH domain. If the finite element decomposition is well-balanced, then overlap computations are expected to be load balanced as well. Conversely, degraded performance may be expected when a finite element decomposition is not load balanced.

The improved insertion algorithm for placeholder material is used by default starting in Sierra version 5.8. Material donation and force application continue to use the algorithm described in Section 1.6.3. If the legacy material insertion algorithm is desired, place the keyword use_legacy_insertion in the Zapotec input file.

An alternative algorithm to compute the volumetric intersection between Sierra elements and CTH cells is available starting in Sierra version 5.22. This algorithm uses the r3d [21] library to perform the computational geometry. Initial results indicate that significant speedups can occur when the CTH cell size is smaller than the Sierra element size. Performance for other cell size ratios is comparable to the current algorithm in Zapotec. The r3d algorithm can be used by placing the keyword use_r3d in the Zapotec input file.

1.6.4 User Instructions

Five input files are required for a Zapotec analysis: (1) the CTH input file, (2) the Presto_ITAR input file, (3) the ExodusII binary model file used by Presto_ITAR, (4) the Zapotec input file, and (5) a Zapotec list file containing the file names of the Presto_ITAR and CTH input files. Each of the input files utilizes a keyword input structure for program control. Only the Zapotec list file and the keyword inputs for the Zapotec input file are discussed herein. The user is directed to the CTH and Presto_ITAR User Manuals for a discussion of their respective input file structures. The input file names are generally arbitrary, but Zapotec uses the input file named zapotec.inp by default and emits an error if the file does not exist and another file name is not specified in the list file.

This section details the syntax for the Zapotec List File and the Zapotec Input file. The final sections detail visualization of Zapotec results and managing restarts.

1.6.4.1 Zapotec List File

The Zapotec list file is a simple file with pointers to which input files to use for CTH and Presto_ITAR. An example is shown below:

```
sierra input deck  = presto_file.i
cth input deck     = cth.inp
zapotec input deck = zapotec.inp
cth run id         = h  ! used as suffix for cth outputs
lag_mass_per_eul_mass = 0.001 ! mass scaling: 0.001 = kg/g
lag_len_per_eul_len = 0.01 ! length scaling: 0.01 = m/cm
```

Global mass and length scaling are also specified in this file. The Sierra/SM model must be scaled to CTH units (CGSeV) before interacting with the CTH model.

1.6.4.2 Zapotec Input File Keywords

The Zapotec input file utilizes a free-field format, is case insensitive, and allows embedded comments. Comments begin with ***, $, %,** and/or **!** characters. Comments are allowed anywhere on a command line, with any data following a comment character ignored by the free-field reader. Program control is issued using a keyword structure. In general, a keyword is followed by data. In the following descriptions, keywords are given in bold face print, with keyword data represented by angled brackets (*i.e.*, `< . . . >`). Accepted abbreviations for keywords, if any, follow the keyword in square brackets (*i.e.*, `[ . . . ]`). Where appropriate, default values for the keyword data are specified in curly brackets (*i.e.*, `{ . . . }`). There are also keyword groups that are of a tree-form, whose general structure must be maintained. An example of this is the keyword `plot`, which is followed by the `time` and `interval` keywords.

The input data are not order dependent. Also, keywords can be repeated within the input file. The free-field reader will override previous specifications of a particular keyword, *i.e.*, the reader follows the "last wins" policy. For example, if `num_eulerian` is specified more than once, the last assignment will be the one used by Zapotec.

Program Control

num_eulerian [num_eul, num_cth] = `<integer>` {0}

Number of Eulerian (CTH) materials in the problem. This number is the summation of the permanent CTH and donated Lagrangian materials in the problem. Void material is not included in the count of CTH materials. The user is required to input this information.

stop_time = `<real>` {1.0e20}

stop_cycle = `<integer>` {1e9}

Stop time or cycle for the Zapotec analysis. This value will control the problem duration. Note it is necessary to set stop times/cycles in the CTH and `Presto_ITAR` input files to larger values. This will allow problem control by Zapotec. Note, the user can specify the stop time or cycle, but not both.

max_subcycles [max_sub] = `<integer>` {50}

Maximum number of Lagrangian subcycles allowed. If **max_subcycles** is set to zero, then no Lagrangian subcycling will be allowed and the CTH and `Presto_ITAR` time steps are constrained to be identical.

coupling

frequency = `<integer>` {1} **tcutoff** = `<value>` **ncutoff** = `<integer>`

Two command structures exist within the Zapotec input deck to control coupling and de-coupling. The first, older-style syntax begins with the keyword **coupling**. The keyword **frequency** specifies the coupling frequency, which defaults to do a coupled calculation at every cycle. The **frequency** option allows for less-frequent coupled calculations. This is sometimes useful for problems where the loading on the Lagrangian structure does not change significantly over time. This option should be used with care, as the solution accuracy is highly contingent on the coupled interaction between Eulerian and Lagrangian materials. A less-frequent coupling can miss much of the physical interaction, leading to a poor solution. The keyword **tcutoff** specifies a time at which CTH is shut down and the `Presto_ITAR` solution continues, while the keyword **ncutoff** specifies the number of time steps over which the decoupling occurs.

multistep

time = `<real>`

type = `<analysis type>`

ncutoff = `<integer>` {1}

The **multistep** keyword enables more complete and varied control for a multi-step calculation (as opposed to the **coupling** keyword). This option allows the user to toggle between coupled and independent calculations at various times. The keyword **type** indicates the type of analysis to be undertaken. The available type options are described in Table 1.15. If the **Lagrangian** type is specified, then the loading derived from the last coupled treatment will be abruptly cut off. This can be overridden by specifying the **ncutoff** option. This allows the loading to decrease linearly over the prescribed number of time steps. The **Lagrangian** type is particularly useful for applications involving air blast loading on structures, where the positive phase duration of the blast pressure pulse can be much shorter than the response time of the structure. In this situation, it is useful to capture the initial coupled interaction between the blast and the structure, then turn off the coupled interaction after the pressure pulse had dropped to a negligible value. The Lagrangian calculation can then proceed to assess the late-time response of the structure.

Example:

```
stop_time = 20.0e-3                  !analysis termination time
multistep
  time = 0.0     type = independent !run CTH and Sierra/SM
  time = 10.0e-3 type = coupled     !independently to 10 msec,
                                    !afterwards perform coupled
                                    !calculation
```

Table 1.15 Description of multistep types

Multistep Type	Description	Related Option
Coupled	Coupled calculations	None
Independent	Allows CTH and `Presto_ITAR` to run concurrently, but does not assess the coupled interaction between the two computational domains	None
Lagrangian	Cuts off the CTH portion of the analysis, allowing `Presto_ITAR` to continue until the Zapotec stop time is reached. By default, the loads prescribed during the force application step will be terminated over a single time step. This can be overridden by specifying the ncutoff option below this command line.	ncutoff
Eulerian	Cuts off the `Presto_ITAR` portion of the analysis, allowing CTH to continue until the Zapotec stop time is reached.	None
Constant	Similar to the Lagrangian option, except that the loads prescribed during the last force application step are applied for the duration of the calculation. The ncutoff option will be ignored.	None

Material Mapping and Insertion

use_legacy_insertion

This keyword, placed anywhere in the Zapotec input file, activates the legacy insertion algorithm that performs all related computations in the CTH/Eulerian domain. This algorithm is slower and its use is generally not recommended.

use_r3d

This keyword, placed anywhere in the Zapotec input file, activates the use of the *R3d* intersection algorithm that can provide significant performance improvements in certain cases. In particular, this algorithm provides greater performance when the CTH discretization is finer than the SIERRA/SM discretization. This option may become default in the future.

Keywords to control the insertion of Lagrangian material into the CTH domain, as described in Section 1.6.3.1.

material_map

eulerian [eul] = `<integer>` **lagrangian** [lag] = `<integer>`

The material map is a required input for Zapotec that provides a mapping between the `Presto_ITAR` material IDs (given by the ExodusII block ID) and the corresponding CTH placeholder material(s). The keyword **eulerian** refers to the Eulerian (CTH) material IDs, while **lagrangian** refers to the Lagrangian (`Presto_ITAR`) block IDs. CTH material IDs are numbered in sequential order as defined in the CTH input file. The **eulerian/lagrangian** keyword lines can be repeated as needed. There is no order dependence for these keywords (*i.e.*, the keyword **lagrangian** can be specified before **eulerian**); however, it is required that the data be input in Euler/Lagrange pairs on the same line in the input file. It is not required that every Lagrangian element block be included in the mapping, but note that any left out of the coupling will not be represented in the CTH domain. This can lead to strange code failures. Care should be taken if blocks are left out. It is generally recommended to include all finite element blocks in the mapping.

insert

xmin = `<real>` {-1.0e20}

xmax = <real> {1.0e20}

ymin = <real> {-1.0e20}

ymax = <real> {1.0e20}

zmin = <real> {-1.0e20}

zmax = <real> {1.0e20}

Bounds for Lagrangian material insertion. Only Lagrangian elements residing within the prescribed bounding box will be considered for insertion into the CTH mesh. This keyword command set can be useful for reducing the volume overlap calculations in the coupling algorithm.

Example:

```
material_map
  eul = 3 lag = 10 !map Lagrangian matl 10 into CTH placeholder matl 3
  lag = 20 eul = 3 !map Lagrangian matl 20 into CTH placeholder matl 3
  eul = 3 lag = 30 !map Lagrangian matl 30 into CTH placeholder matl 3
```

External Loading Specification

cthf_insert

xmin = <value>

xmax = <value>

ymin = <value>

ymax = <value>

zmin = <value>

zmax = <value>

Bounding box for inserting Lagrangian material surface data into the coupled calculation. Useful for reducing the amount of area overlap calculations.

force

material [mat] = <integer> <integer> .....

weight = <real> {1.0}

option = <integer> {0}

scale = <real> {1.0}

ignore_gaps {off}

ignore_all_gaps {off}

ignore_contact_surfaces {off}

penalty = <integer> {1}

pen_coeff = <real> {0.1}

filename = <character>

void = <integer> {2}

Control for force application on Lagrangian materials.

The keyword **material** refers to the definition of the Lagrangian material (Exodus block IDs). When setting up the Lagrangian problem, the user must identify the Lagrangian surfaces that can overlap the Eulerian mesh. In turn, the Zapotec force application algorithm utilizes these surface definitions to assess the interaction between the Lagrangian and Eulerian materials.

The keyword **material** sets the Lagrangian materials, *i.e.*, ExodusII blocks, that can interact with the Eulerian domain. The material surface definition is automatically generated within the code and comprises surface elements residing on the exterior boundary of a Lagrangian material. This includes surfaces residing on a plane of symmetry.

Material surfaces are automatically regenerated when elements die in the Presto_ITAR portion of the calculation. The automated material surface generation is invoked as follows:

material = <Lagrangian Material ID> <Lagrangian Material ID> .....

where any number of Lagrangian material (block) IDs can be input.

Blocks can be excluded from the list of material (block) IDs, so that no forces are applied in the coupling. Note that this can lead to unexpected results, so it is generally recommended that all finite element blocks are included in the coupling.

Zapotec allows different approaches for determination of the Eulerian stress state, which are outlined in Table 1.16. The default algorithm, discussed previously in Section 1.6.2.6, derives the Eulerian stress state from the weighted material pressure for the mixed Euler/Lagrange (E/L) and forward cells combined with the deviatoric stress from the forward cell (see Fig. 1.5). By default, a linear weighting is assumed between the mixed and forward cells. The user can modify the weighting factor (defined as w_E in Fig. 1.5) using the keyword **weight**. The default approach with a linear weight has been found to work well for problems involving solid materials (*e.g.*, penetration applications where both the Lagrangian and Eulerian materials are solids). For applications involving fluid-structure or blast-structure interaction, **force option** 2 is recommended. For bonded materials, **force option** 3 will automatically be invoked. This will not affect the **force option** specified for contacting Euler-Lagrange materials.

Table 1.16 Options for Determination of Eulerian Stress State

Force Option	Method for Determining Eulerian Stress State
0 (Default)	Derived from weighted material pressure in mixed and forward cells combined with deviatoric stress from forward cell, *i.e.*, $\boldsymbol{\sigma} = -p_{\text{weighted}}\boldsymbol{I} + \boldsymbol{s}_{\text{forward}}$
1	Derived from pressure and deviatoric stress in the forward cell only, *i.e.*, $\boldsymbol{\sigma} = -p_{\text{forward}}\boldsymbol{I} + \boldsymbol{s}_{\text{forward}}$
2	Derived from pressure in the forward cell only, *i.e.*, $\boldsymbol{\sigma} = -p_{\text{forward}}\boldsymbol{I}$. Note: All shear stresses are set to zero
3	Derived from the pressure and deviatoric stress in the mixed E/L cell. This option is automatically invoked for bonded materials.

The keyword **scale** allows the user to scale the applied normal force on the Lagrangian materials. It is a global option that scales all forces applied to all Lagrangian materials in the problem. Great care should be exercised with its use since it scales all forces applied to a Lagrangian body.

The keyword **ignore_gaps** sets the stress state to zero when the forward cell contains a Lagrangian material different than itself. To zero the stress when the forward cell contains any Lagrangian material, use the keyword **ignore_all_gaps**. For applications where there are small gaps between Lagrangian bodies (*e.g.*, consider the gap between a door and its door frame), it is possible to over-compress Eulerian material that flows into the gap. This can lead to excessive and unrealistic loading on the Lagrangian structure, which is avoided with the **ignore_gaps** option. Note that there are other cases in which the use of **ignore_gaps** can lead to force asymmetries and apparent mesh biasing, so it is recommended to only use this command if it is needed for a specific condition.

The command **ignore_contact_surfaces** is another option to control the application of forces onto the Lagrangian material. Models that have curved surfaces in contact will sometimes have a spurious CTH force applied to them, or surfaces will receive spurious force application as they come into contact with each other. This command turns off CTH force application to element faces when all nodes of the face are in Lagrangian contact (*contact_status = 1*). Shell elements in contact are handled as a special case. The `contact_shell_facet_status` nodal variable is used to only load the shell from planar sides that are not in Lagrangian contact.

Since **ignore_contact_surfaces** uses the nodal *contact_status* field to determine whether or not to add CTH forces for the face, this command can be sensitive to the relationship between CTH cell edge length and the search tolerances specified for contact. As two Lagrangian elements are coming into contact, the CTH material between them can exert large forces as it is compressed. Adjusting contact search tolerances can allow the CTH force to be ignored prior to actually coming into contact. Contact search tolerances equal to a fraction (*e.g.*, one-half or one-quarter) of the smallest CTH edge length have produced reasonable results in a limited number of use cases.

An additional penalty force can be applied to the Lagrangian structure. This loading is in addition to the traditional loading derived from the surface traction. The penalty treatment is invoked with the **penalty** and **pen_coeff** commands. A penalty force is generally needed to add an additional correction for overfilled cells. The penalty treatment is described in Section 1.6.2.7. Note that this is an experimental capability, and should be used with caution.

There are inherent ambiguities in the application of forces when void is present in the mixed Euler/Lagrange cell. Since material interfaces are not explicitly tracked in CTH, it is not possible to determine if an Eulerian material is directly in contact with the inserted Lagrangian material. The **void** feature allows a smooth decrease in the applied loading based on the volume fraction of void in the mixed cell. The applied forces are "corrected" as follows:

$$f_n^{\text{corrected}} = f_{\text{n}} \left(1 - \phi_{\text{v}}\right)^{\text{void}},$$

where ϕ_{v} is the void volume fraction, f_n is the magnitude of the normal force, and void is the user-defined input. The default is a quadratic decay in the forces.

The keyword **filename** provides the full name of the printed global output file generated for Lagrangian materials. Examples of output include the global forces exerted on the Lagrangian materials, average linear and angular velocities, and global bounds of the Lagrangian mesh.

Remark: Materials composed of shell elements require special treatment and their surface definitions should not be specified using the **material** command. All force application data for shells should be specified using the **loads** keyword within the **shell** keyword set described in Section 1.6.4.2.

friction

global = <real> {0}

constant = <real> {0}

eulerian = <integer> **lagrangian** = <integer

lagrangian = <integer> **eulerian** = <integer>

velocity = <integer>

eulerian = <integer> **lagrangian** = <integer>

lagrangian = <integer> **eulerian** = <integer>

Specification of frictional surfaces between Lagrangian and Eulerian materials. The keyword **global** refers to the assignment of a global (constant) Coulomb friction coefficient for all Lagrangian and Eulerian material interfaces in the problem. The keyword **constant** is used for the assignment of a constant friction coefficient between specific Lagrangian and Eulerian materials. The keyword **velocity** refers to the assignment of a velocity-dependent function for the friction coefficient, where the associated data is a function ID. In order to use a velocity-dependent friction model, the user must prescribe the friction coefficient as a function of sliding velocity using the **function** keyword set (see below). As discussed previously, the keyword **lagrangian** refers to the Lagrangian material (block) IDs, while **eulerian** refers to the Eulerian material IDs. Additionally, the **lagrangian/eulerian** keywords for each material pair can be in any order and the listing of pairs can be in any order, but the listing must be pairwise. Note also that keywords can be mixed. For example, a user can prescribe a global friction coefficient for the problem and then override the global value by assigning friction coefficients between specific materials. When using mixed keywords, it is not required that the **global** keyword precede the other keyword assignments.

function = <Function ID>

<x-value> <y-value>

<x-value> <y-value>

…

Function definition. The keyword **function** is used to define a piece-wise linear function, specified via a set of x-y data pairs. If the independent quantity is outside the range of the x-values input, then the dependent data is extrapolated as illustrated in Fig. 1.10. There are no limitations on the number of functions or number of data pairs. The only requirement is that the function ID be unique and the x-data be monotonically increasing.

Example:

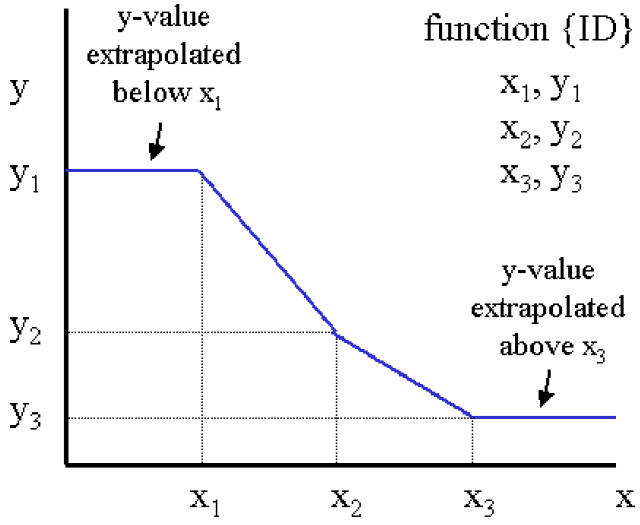


Fig. 1.10 Example of function definition.

```
force
  file = force.dat           !Summary data written to force.dat
  material = 10, 20, 30, 40 !Force application Lag materials
  penalty = 1                !Add in penalty force
  p_coeff = 5                !with coefficient of 5
friction
  global = 0.10       !Globally defined Coulomb friction coef.
  velocity 10         !function 10 defines the friction
                      !versus velocity data
    eul = 3 lag = 30 !Overrides previous constant
                      !friction assignment
function 10
  100.0  0.10         !data for function 10
  200.0  0.04
  400.0  0.02
```

Plot/Restart Control

plot

time = <real> **interval** = <real>

cycle = <integer> **interval** = <integer>

Plot specification. The **plot** keyword triggers Presto_ITAR and CTH to write spatial results (plot) data at selected times or cycles to their respective results files. The keywords **time/cycle** and **interval** refer to the initial plot time/cycle and plot frequency following the specified plot time/cycle. The **time/cycle** keyword lines can be repeated any number of times, but the two cannot be mixed (*i.e.*, the user cannot specify both time and cycle plot dumps). Although plot frequency

is controlled via Zapotec, the spymaster capability in CTH is not yet reliably controlled, thus it is a best practice to duplicate the plot time/cycle identifiers in the CTH input file using the **spy** keyword set. The `Presto_ITAR` input file needs to have an output specification block to define output variables, but it does not require that an output frequency be specified. If it does specify an output frequency, Zapotec requests additional output according to its input file as necessary.

restart

time = `<real>` **interval** = `<real>`

cycle = `<integer>` **interval** = `<integer>`

Restart specification. By default, both `Presto_ITAR` (if a restart block is defined in its input) and CTH will write a restart file at the end of an analysis. However, it is often useful to write intermediate restart files. The **restart** keyword triggers `Presto_ITAR` and CTH to write intermediate restart data at selected times or cycles to their respective restart files. The **times/cycle** keyword lines were described above and can be repeated any number of times; however, the two keyword types cannot be mixed (*i.e.*, the user cannot specify both time and cycle restart dumps). Although restart timing is controlled via Zapotec, the user must specify corresponding restart identifiers in the CTH input file using **restt** keyword set, though Zapotec controls its output. Similarly to plotting, the `Presto_ITAR` input file must define the restart block, but output is controlled by Zapotec. If a restart output frequency is specified in the `Presto_ITAR` input, Zapotec will output additional restart steps as needed.

Restart files can be written based on elapsed wall clock time with **wall_clock_restart = HR:MIN:SEC**. This functionality works in conjunction with the **restart** keyword; restart files will be written when either specification is true.

Effective use of restart requires careful coordination between input files. See Section 1.6.4.4 for details on executing a restart.

Example:

```
plot
  time = 0.0e-3  interval = 1.0e-3   !plot time and frequency
  time = 10.0e-3 interval = 2.0e-3
  time = 30.0e-3 interval = 2.5e-3
restart
  time = 20.0e-3 interval = 20.0e-3  !restart time and frequency
wall_clock_restart = 4:00:00 ! write restart files every 4 hours
```

Shell Element Specification

shell = <integer>

fluff = <character> {bottom}

scale = <real> {1.0}

tscale = <real> {none}

integ = <integer> {-1}

conserve = <integer> {0}

loads = <character> {top}

Shell element data. Insertion of thin shells into the Eulerian mesh requires some manipulation of the data to assure proper interface tracking. In essence, the Lagrangian shell element is "fluffed" up to have an effective thickness (and volume) for insertion into the CTH mesh. The data manipulation is done in the Zapotec interface routines with the Lagrangian code and does not affect the original finite element mesh data.

In the finite element model development, material blocks composed of shell elements are assigned unique IDs. The user is required to identify shell element materials (blocks) that will be inserted in the CTH mesh. This is in addition to their specification in the material and donation maps. The shell element material is identified using the keyword **shell** as follows:

shell = <Lagrangian material ID>

Additional (optional) inputs can be specified for manipulation of data for the prescribed shell material.

The **fluff** keyword is used to determine which side(s) of the shell is to be extended by *one-half* the effective shell thickness, where:

fluff = <top> or <bottom> or <both> or <none>

Sides not selected for extension will be extended by *one-half the actual shell thickness*. The top surface is defined by the outward surface normal for the shell element (see Fig. 1.6 for a description of the surface definitions). The outward surface normal is based on a right-hand rule with respect to the counter-clockwise ordering of nodes on the shell element. The default is to extend only the bottom surface of the shell element.

The user can also scale the effective shell thickness using either the **scale** or **tscale** commands. The **scale** command bases the effective thickness of the shell element on the characteristic length of a CTH cell and is applied as follows:

$$t_{\text{eff,shell}} = (\text{scale}) l_{\text{cell}},$$

where $t_{\text{eff,shell}}$ is the effective shell thickness and l_{cell} is the characteristic cell length, taken as the minimum length over all cells within the CTH mesh. The **tscale** command allows the user to directly input the effective shell thickness. This option is useful when the user wishes to maintain

a constant effective shell thickness while considering a varying CTH mesh resolution, *e.g.*, when conducting a mesh resolution study.

By default, the algorithm assumes the fluffed shell is stress-free and is at its reference density and energy state (**integ** = -1). The user has the option to override the default and modify the state data. The **integ** command is used to modify the stress state, which can be based on either an average over all integration points or the stress state at an individual integration point. An **integ** value of zero (0) uses an average over all integration points. Any positive value refers to a specific integration point. In general, the shell element will have 3 to 5 integration points through the thickness, with the integration points running from the bottom to the top of the shell element.

The inserted material density can be modified to conserve the mass and momentum using the **conserve** keyword, where the inserted density is scaled as follows:

$$\rho_{\text{inserted}} = \rho_{\text{shell}} V_{\text{shell}} / V_{\text{inserted}}.$$

When the **conserve** option is invoked, the shell density ρ_{shell} is the current shell density rather than the reference density.

As a convenience, surface definitions for the fluffed shells are generated automatically within Zapotec. The surface definitions are needed in the force application step of the coupled algorithm. There is no need for the user to develop shell surface definitions, such as side sets, in the model file generation. The keyword **loads** is used to specify the side(s) to apply external forces as follows:

loads = `<top>` or `<bottom>` or `<both>` or `<none>`

The default is to apply external forces to the top surface of the fluffed shell (see Fig. 1.6 for description of the surface definitions).

All force application data for shells are specified using the **loads** keyword only (*i.e.*, shell materials should not be included in the **material** command under the **force** keyword set). At the end of the Zapotec coupled treatment, the external loads on the shell are transferred back to the Lagrangian code as a set of external nodal forces. This requires that the forces on the fluffed shell surfaces be mapped back onto the original shell element configuration.

Remark: In some applications, it is desirable to exclude groups of shell elements from being subjected to any externally applied loads (external loading is always evaluated for the defined Lagrangian material surfaces). An example would be a shell element covering a hexahedral element face, where the faces and associated face nodes are coincident for the two element types. In this situation, one may want to insert the shell into the Eulerian mesh to conserve mass and momentum, but disregard applying external loads to the shell surface. This can be accomplished by specifying **loads = none**. If this is not done, there will be a duplication of the Lagrangian surface definitions (one associated with the underlying hexahedral element and the other associated with the covering shell element whose surface definition was automatically created by Zapotec). In this case, the duplicated surface definitions will lead to twice the applied loading on the Lagrangian material.

Example:

```
shell   10       !Lagrangian material ID for shell block
  scale = 1.0    !Scale factor for effective shell thickness
  loads = top    !Force application on top surface of shell element
  fluff = bottom !Fluff shell in direction of the bottom surface of
                 !the shell element
```

Donation

death_threshold = <integer>

Threshold for number of Presto_ITAR elements that must die in a Presto_ITAR time step before donation will occur. The default is 0, *i.e.*, donation will be incurred whenever a Presto_ITAR element dies. This threshold can be set to some finite number (*e.g.*, between 2 to 5) to avoid the excessively small CTH times steps incurred by donation. This will be at the cost of losing mass from the problem.

donation_map

eulerian [eul] = <integer> **lagrangian** [lag] = <integer>

Donated material map. The donation map provides a mapping between the ID of the Presto_ITAR material to be donated and the corresponding CTH donated material ID. It is customary to utilize this option if element death is invoked within Presto_ITAR. With this option, a dead Presto_ITAR element is donated to the CTH mesh. Once the Lagrangian element is donated, the donated material is then treated as a CTH material. The donation map is only specified if donation is desired in the problem, *i.e.*, the user has the option to allow element death within Presto_ITAR without donation by not including the associated material in the donation map. The **eulerian/lagrangian** keyword lines were described previously and can be repeated as needed.

donation_off = <Lagrangian material ID> <time>

Allows the user to turn off donation for a given Lagrangian material at a specified time.

don_insert

xmin = <value>

xmax = <value>

ymin = <value>

ymax = <value>

zmin = <value>

zmax = <value>

Bounding box for inserting donated Lagrangian material into the coupled calculation. Useful for reducing the amount of volume overlap calculations during material insertion/donation.

max_don_subcycle = <value>

Number of Lagrangian subcycles to allow before donation occurs. Useful for increasing the stable CTH time step. This option should be used with care since the configuration of the "dead" Lagrangian element will continue to change over time. Note that if the "dead" element is inverted in the current configuration, it will not be donated and the mass associated with that element will be lost. This lost mass is accumulated in the global variable zap_lost_mass available for output in Presto_ITAR. We recommend that the value of **max_don_subcycle** not exceed 3. The default is zero, which results in the CTH stable time step being cut back to the time at which element death occurs.

Example:

```
material_map
  eul = 3 lag = 10 !map Lag matl 10 into CTH placeholder matl 3
  lag = 20 eul = 3 !map Lag matl 20 into CTH placeholder matl 3
  eul = 3 lag = 30 !map Lag matl 30 into CTH placeholder matl 3
donation_map
  eul = 2 lag = 30 !donate "dying" elements from Lag material 30
                   !donate them to CTH matl 2
```

Miscellaneous

ambient_pres = <real> {0.0}

Ambient (reference) pressure associated with the Lagrangian materials. Presto_ITAR assumes a zero pressure for ambient conditions. This can differ from the value used by CTH for the Eulerian materials. This option allows the user to specify a consistent ambient pressure for the Lagrangian materials in the problem. Specifying any non-zero value of **ambient_pres** acts to zero out any pressures with magnitude less than **ambient_pres**, including fluctuating pressure loads which may not be related to ambient conditions. For example, deltas in pressure across a shell will not produce load on the shell if they are lower than **ambient_pres**.

cpu_stop = <real> {1.0e20}

CPU stop time in seconds. This is a useful option when batch processing a calculation with set CPU time limits. Zapotec will automatically force both CTH and Presto_ITAR to output a restart at this time. However, currently these restart files may be written at inconsistent times. It is a current best practice to write frequent restarts and not rely on this command to make a usable restart at the end of the queue time.

cutoff_placeholder = <volume fraction>

Allows the user to remove trace amounts of Lagrangian material from those cells containing *only* placeholder material. Useful if you are desperate and cannot get past a CTH DTMIN error.

eliminate_excess {FALSE}

Removes overlapped Eulerian material at the startup of the calculation. This option is useful when modeling complex Lagrangian material geometries overlapping a large, homogeneous Eulerian domain. For example, consider the problem of an air blast loading on a multi-room concrete structure (see Fig. 1.11). The air is best modeled as an Eulerian material, while the structure is best modeled as Lagrangian. Inserting individual air spaces into the Eulerian mesh would be a difficult and time-consuming process. This can be avoided using the **eliminate_excess** option. With this option, the user can create a large Eulerian region composed of air and then overlay the Lagrangian structure onto the air space. Any overlapped Eulerian material is automatically removed from the problem. It should be noted that currently the algorithm employed by this keyword is not entirely accurate—some Eulerian material may be left behind inside the inserted Lagrangian material. Although this can cause small spurious pressures at time zero, they are completely removed in subsequent steps as the Lagrangian material is re-inserted. These small errors can be alleviated by using the **remove exodus** command in CTH's diatom to pre-remove the finite element mesh from the Eulerian domain.

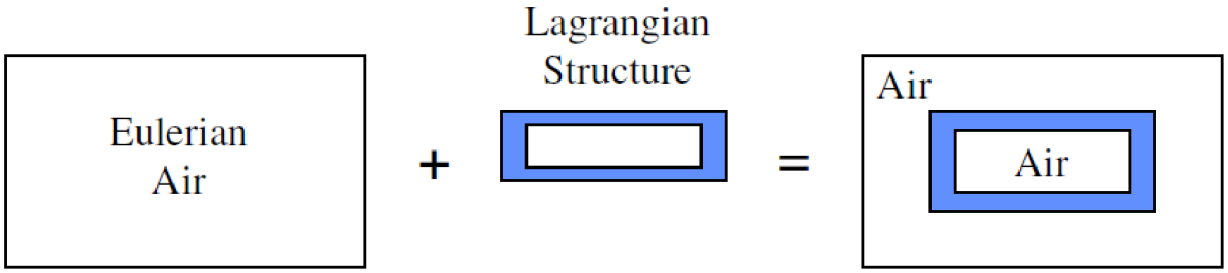


Fig. 1.11 Illustration of the eliminate_excess option.

donation_vmag_threshold = <real> {1.0e4}

Do not donate elements whose velocity magnitude is below this threshold limit.

insert_material_and_finish {FALSE}

Inserts the Lagrangian blocks specified by the **material_map** into CTH, writes CTH output, and stops program execution. No time advancement is performed. This option is mostly a debugging command that allows the user to inspect grid resolution, etc. It can be very useful when checking a model that uses length scaling.

mda1 {FALSE}

The **mda1** keyword triggers the following set of options, all of which can be set separately:

```
cutoff_placeholder = 1.0e-9
donation_vmag_threshold = 1.0e4
max_don_subcycle = 3
```

exit

All keywords following the keyword **exit** are ignored.

resize lsizel = <integer> {0}

resize msizel = `<integer>` {0}

resize nsizel = `<integer>` {0}

Resize array bounds. Zapotec estimates the array bounds for the Lagrangian element, surface element, and node data described in Section 1.6.2.7. On rare occasions, the actual bounds can exceed those estimated, resulting in termination of Zapotec at startup. When Zapotec terminates, an error message is written to the output file indicating the problem array. The user can resize the array bounds using the above commands, where the keyword following **resize** indicates the array type and the third input is a numerical value indicating the new array bound. The keywords **lsizel**, **msizel**, and **nsizel** refer to the number of volume elements, surface elements, and nodes, respectively.

shield = `<lag ID> <eul ID>`

Material shielding. In some instances, one Lagrangian material may shield another. For example, consider the situation depicted in Fig. 1.12. Here, we have a thin Lagrangian structure surrounding an interior Lagrangian material. If both materials are identified for force application, then loads will be incorrectly applied to the interior, shielded material. The force application algorithm does not check to see if any other Lagrangian material resides in either the mixed E/L or forward cells. In this situation, it is necessary to identify shielded Lagrangian materials. This is done using the **shield** command, whose inputs are the shielded Lagrangian material ID `<lag ID>` and its mapped placeholder material location `<eul ID>` in the CTH database. In the implementation, if the material surface has the prescribed Lagrangian material ID and the mixed E/L contains other placeholder material and any shielded material, then the Lagrangian material is assumed to be shielded. With this option, the material surface must be defined using the **material** option in the **force** keyword set.

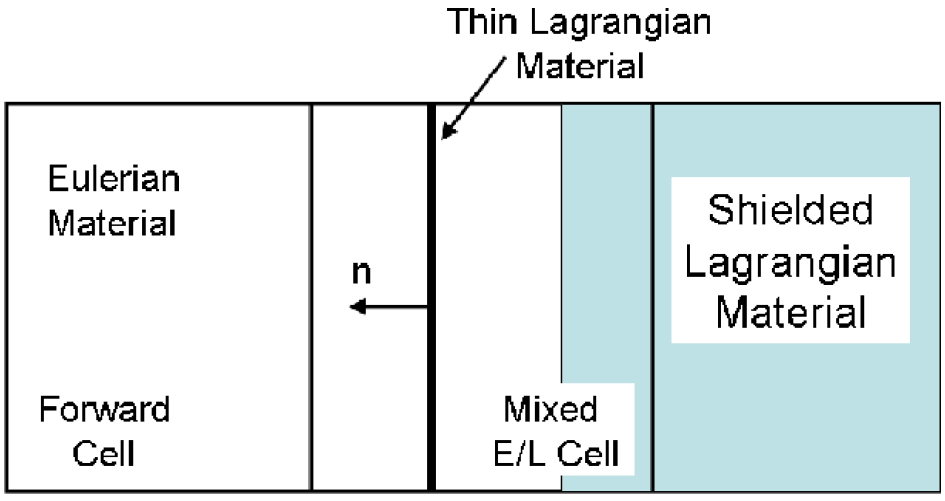


Fig. 1.12 Illustration of shielded Lagrangian materials.

timestep = `<integer>`

Frequency to write time step information to the printed output file. By default, Zapotec will write a reduced set of time step information to the output file, the frequency of which is determined internally.

Debug Information

debug

dodonate

doinsert

lput

my_pe = `<integer>` {default is all processors}

istart = `<integer>` {0}

timestep = `<integer>` {10}

Debug control for various levels in the coupling algorithm. Keywords **driver**, **dolink**, and **docalcs** provide top-level information regarding time step control and calls to sub-level procedures. Keywords **dodonate**, **doinsert**, **lover**, and **lput** provide detailed information regarding material insertion. Note, printed output from these options can be lengthy and is generally recommended for developer use only. The keyword **my_pe** can be used to identify debug information for a specific processor. The keyword **istart** indicates the first Zapotec cycle at which debug information is to be written. The keyword **timestep** writes time step information to an auxiliary file, *debug_timestep.out*. The **timestep** keyword is followed by the frequency at which data is written to this file.

1.6.4.3 Visualization

Visualization of Zapotec spatial and temporal results is possible using the ParaView visualization software [1]. Details on how to appropriately visualize Zapotec results using ParaView are included in the Examples Manual [31]. Programs specifically designed for visualizing CTH results, such as SPYPLT and SPYHIS [6], are discussed in their respective user manuals as well as the CTH documentation. Insertion of tracer points in the Eulerian domain and history output for the Lagrangian domain is discussed in the CTH and `Presto_ITAR` documentation, respectively.

1.6.4.4 Executing Restarts

In order to restart a Zapotec run, each of the input decks must be set up to process restarts correctly and must be correctly modified upon restart. The steps for this are described in the next sections.

Ensure Input Files Are Configured for Restart at the Start

Before you run the problem, it is important to make sure that your input files are consistent such that they write out restart files. The Zapotec input file specifies a restart frequency and will force both Sierra/SM and CTH to write out compatible restart files at the specified frequency. No additional input is needed in the CTH file, but a restart block is required in the Sierra/SM input.

As described in Section 1.6.4.2, the restart frequency is specified in the *zapotec.inp* file using the **restart** keyword:

```
restart
  time = 0.0 interval = 1.0e-3
```

In the Sierra/SM file, specify restart using the restart data block in the region block. No restart frequency is needed, since Zapotec will control the restart frequency. An example of this in a Sierra/SM input deck is:

```
begin restart data restart
  database name = run.rst
end restart data restart
```

It is also good to note the CTH run ID, which is specified in the Zapotec list file. It is recommended to start with the run id 'a', so that subsequent restarts can update the id through the alphabet (b, c, d, etc.). In the Zapotec list file, the run id is specified as:

```
cth run id = a
```

One other important item of note: CTH appears to have an internal limit on the length of the path to restart files. If the restart path is over 80 characters, CTH may fail with one or more restart database errors. To avoid this, keep the path to the CTH restart files (as well as the CTH restart file names) shorter than 80 characters.

Determine the Time, Cycle, and CTH Run ID to Use for Restart

When you are ready to restart the analysis, look at the *zapotec.out* file from the previous run, which reports the simulation times where restart files are written (an example is shown below):

```
CTH restart file written: cycle, time, no. dumps:   4543  4.97E-05  2
CTH plot file written: cycle, time:                 4543  4.97E-05
SIERRA/SM restart written at cycle, Lag step, time: 4543 4543 4.97E-05
SIERRA/SM plot written at cycle, Lag step, time:    4543 4543 4.97E-05
```

Of note is the first line, which specifies the cycle and time at which a CTH restart file was written. Keep note of this cycle number. Also keep note of the CTH run ID (specified in the Zapotec list file).

When restarting a simulation, the log file will be overwritten. To prevent this, the best practice is to rename *zapotec.out* to *zapotec.out.<runid>* (e.g., *zapotec.out.a* if the run ID is *a*) after the run completes. In this way, the log file is not overwritten (preserving the restart times and cycle numbers) and the log file is associated with the corresponding CTH run ID.

Edit the CTH File to Restart at the Given Cycle

In the CTH input file, add a restart specification that lists the name of the restart file to read from and which cycle to use for restart. The example lines below show the form of the syntax. In this example, the previous run ID had been *a*, so all the CTH restart files were called "rscta". Also, we will restart at the cycle number shown in the previous step, cycle 4543:

```
restart
  file = 'rscta'
  cycle = 4543
endrestart
```

Change the CTH Run ID in the Zapotec List File

In this example, we had previously run with the ID *a*. Now change the line to be:

```
cth run id = b
```

This will preserve all the CTH output and results files from the previous run and generate new ones with the run ID *b* (e.g., *rsctb*, *spctb*, *octb*, etc.). Note that CTH does try not to overwrite files, and thus if you keep the CTH run ID the same, it will likely generate files with an appended identifier. However, this can be confusing as to which file is the latest, so it is highly recommended to change the CTH run ID.

Specify the Restart Time in the Presto_ITAR Input File

Add the following line in the `Presto_ITAR` input file just inside the "begin sierra" block:

```
restart time = <time>
```

In our example, *<time> = 4.97E-05*. Sierra is generally careful about restart files and will append extra characters to the file names and create new files when needed, so explicitly specifying a new restart file name is not generally necessary. Thus, this is all you generally need. If you would rather control this naming yourself, you can change restart names using the *input database* and *output database* keywords as shown above.

```
begin restart data restart
  input database name = run.rst
  output database name = run_b.rst
end restart data restart
```

Run the Same Zapotec Command Line as Before

Restart should now activate correctly. Zapotec will check to make sure the CTH and Sierra/SM restart times are very close and will error out if they are not. If you get this error, check to make sure the restart times you specified in the CTH and Sierra/SM restart files were the same and that there were restarts actually written at those times. To check the times actually in the restart files, use the *cthed* tool to look at the CTH restart files and *grope* or *blot* to look at the times written in the `Presto_ITAR` restart files.

This page left blank

References

[1] Utkarsh Ayachit. *The ParaView Guide: A Parallel Visualization Application*. Kitware, 2015. ISBN 978-1930934306.

[2] R.L. Bell, M.R. Baer, R.M. Brannon, D.A. Crawford, M.G. Elrick, E.S. Hertel, S.A. Silling, and P.A. Taylor. *CTH User's Manual and Input Instructions, Version 6.0*. Sandia National Laboratories, Albuquerque, NM, November 2000.

[3] R.L. Bell, M.G. Elrick, and E.S. Hertel. Multi-processing cth: porting legacy fortran code to mp hardware. In *Nuclear Explosives Code Developers' Conference*. San Diego, CA, October 22-25 1996.

[4] D.J Benson. Momentum advection on a staggered mesh. *Journal of Computational Physics*, 100:143–162, 1992.

[5] G.C. Bessette and D.L. Littlefield. Analysis of loading on long-rod penetrators by oblique moving plates. In *Proceedings of the Conference of the American Physical Society Topical Group on Shock Compression of Condensed Matters*, 937–940. New York, 1998.

[6] D.A. Crawford. *Spymaster User's Guide, Version 1.01*. Sandia National Laboratories, Albuquerque, NM, February 2002.

[7] D.P. Flanagan and T. Belytschko. A uniform strain hexahedron and quadrilateral with orthogonal hourglass control. *International Journal for Numerical Methods in Engineering*, 17:679–706, 1981. \href http://dx.doi.org/10.1002/nme.1620170504doi.

[8] D.P. Flanagan and T. Belytschko. A uniform strain hexahedron and quadrilateral with orthogonal hourglass control. *International Journal for Numerical Methods in Engineering*, 17:679–706, 1981.

[9] T. J. Holmquist G.R. Johnson and S.R. Beissel. Response of aluminum nitride (including a phase change to large strains. *Journal Applied Physics*, 2003.

[10] E.S. Hertel, R.L. Bell, M.G. Elrick, A.V. Farnsworth, G.I. Kerley, J.M. McGlaun, S.V. Petney, S.A. Silling, P.A. Taylor, and L. Yarrington. Cth: a software family for multi-dimensional shock physics analysis. Technical Report SAND92-2089C, Sandia National Laboratories, Albuquerque, NM, July 1993.

[11] G.R. Johnson and S.R. Beissel. Modular material model subroutines for explicit lagrangian computer codes. Technical Report ARL-CR-556, Network Computing Services, Inc., Minneapolis, MN, February 2005.

[12] G.R. Johnson and J.A. Schonhardt. Some parametric sensitivity analyses for high velocity impact computations. *Nuclear Engineering Design*, 138:75–91, 1992.

[13] G.R. Johnson and R.A. Stryk. Eroding interface and improved tetrahedral element algorithms for high-velocity impact computations in three dimensions. *International Journal of Impact Engineering*, 5:411–421, 1987.

[14] E.S. Hertel Jr. and G.I. Kerley. CTH reference manual: the equation of state package. Technical Report SAND98-0947, Sandia National Laboratories, Albuquerque, NM, 1998.

[15] C.N. Kingery and G. Bulmash. Airblast parameters from TNT spherical air burst and hemispherical surface burst. Technical Report ARBBRL-TR-02555, Ballistic Research Laboratory, Aberdeen Proving Ground, MD, April 1984.

[16] T.A. Laursen, S.W. Attaway, and R.I. Zadoks. SEACAS theory manuals: part III. finite element analysis in nonlinear solid mechanics. Technical Report SAND98-1760/3, Sandia National Laboratories, Albuquerque, NM, 1999. \href http://infoserve.sandia.gov/sand_doc/1998/981760-3.pdfpdf.

[17] S.P. Lyon and J.D. Johnson. SESAME: the Los Alamos National Laboratory equation of state database. Technical Report LA-UR-92-3407, Los Alamos National Laboratory, 1992.

[18] J.M. McGlaun, S.L. Thompson, and M.G. Elrick. Cth: a three-dimensional shock wave physics code. *International Journal of Impact Engineering*, 10:351–360, 1990.

[19] D.P. Peterson. Finding the volume common to a block and a tetrahedron, more quickly. Technical Report Unpublished SAND Report, Sandia National Laboratories, Albuquerque, NM, April 1995.

[20] A. Picklesimer. The joint DoD/DoE munitions technology program, progress report for FY01, dynamic properties of materials. Technical Report LA-14015-PR, Los Alamos National Laboratory, February 2003.

[21] Devon Powell and Tom Abel. An exact general remeshing scheme applied to physically conservative voxelization. *Journal of Computational Physics*, 297:340–356, 2015.

[22] G. Randers-Pehrson and K.A. Bannister. Airblast loading model for DYNA2D and DYNA3D. Technical Report ARL-TR-1310, Army Research Laboratory, March 1997.

[23] W.M. Scherzinger. Consistent Modeling for Constitutive and EOS Models. Technical Report SAND2022-2411CTF, Sandia National Laboratories, 2022.

[24] W.M. Scherzinger and D.C. Hammerand. Constitutive models in LAME. Technical Report SAND2007-5873, Sandia National Laboratories, Albuquerque, NM, September 2007. \href http://infoserve.sandia.gov/sand_doc/2007/075873.pdfpdf.

[25] S. Silling. Use of the Johnson-Cook fracture model in CTH. Technical Report Memo, Sandia National Laboratories, Albuquerque, NM, 1996.

[26] S.A. Silling. CTH reference manual: viscoplastic models. Technical Report SAND91-0292, Sandia National Laboratories, Albuquerque, NM, 1991.

[27] J.W. Swegle. SIERRA: PRESTO theory documentation: energy dependent materials version 1.0. Technical Report SAND2009-3801P, Sandia National Laboratories, Albuquerque, NM, October 2001.

[28] J.W. Swegle, S.W. Attaway, M.W. Heinstein, F.J. Mello, and D.L. Hicks. An analysis of smoothed particle hydrodynamics. Technical Report SAND93-2513, Sandia National

Laboratories, Albuquerque, NM, March 1994. \href http://infoserve.sandia.gov/sand_doc/1993/932513.pdfpdf.

[29] L.M. Taylor and D.P. Flanagan. Pronto3D: a three-dimensional transient solid dynamics program. Technical Report SAND87-1912, Sandia National Laboratories, Albuquerque, NM, March 1989. \href http://infoserve.sandia.gov/sand_doc/1987/871912.pdfpdf.

[30] P.A. Taylor. CTH reference manual: the Steinberg-Guinan-Lund viscoplastic model. Technical Report SAND92-0716, Sandia National Laboratories, Albuquerque, NM, 1992.

[31] SIERRA Solid Mechanics Team. *Zapotec 3.0 Example Problems Manual*. Sandia National Laboratories, Albuquerque, NM, September 2016.

[32] SIERRA Solid Mechanics Team. *Sierra/SolidMechanics 5.0 Theory Manual*. Sandia National Laboratories, Albuquerque, NM, April 2021.

[33] SIERRA Solid Mechanics Team. *Sierra/SolidMechanics 5.0 User's Guide*. Sandia National Laboratories, Albuquerque, NM, April 2021.

[34] R.G. Whirley and B.E. Engelmann. Slidesurfaces with adaptive new definitions (sand) for transient analysis. *New Methods in Transient Analysis*, PVP 246/AMD 143:65–71, 1992.

[35] Protective Design Center, United States Army Corps of Engineers. Conwep 2.1.0.8. \href https://pdc.usace.army.mil/software/conweplink.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.