WARP3D–Release 10.8

Dynamic Nonlinear Analysis of Solids Using A Preconditioned Conjugate Gradient Software Architecture

By

Kyle C. Koppenhoefer
Arne. S. Gullerud
Claudio Ruggieri
Robert H. Dodds, Jr.
University of Illinois

Brian E. Healy
Exxon Production Research Company

A Report on a Research Project Sponsored by the:

U.S. Nuclear Regulatory Commission
Office of Nuclear Regulatory Research
Division of Engineering
Washington, D.C.

NASA–Ames Research Center
Moffett Field, California
and
NASA–Langley Research Center
Langley, Virginia

University of Illinois
Urbana, Illinois
September 1998

ACKNOWLEDGEMENTS

This investigation was supported by grants principally from the Nuclear Regulatory Commission, the National Aeronautic and Space Administration (Ames and Langley Research Centers) and the Heavy Section Steel Technology Program (HSST) at the Oak Ridge National Laboratory.

Early research on the numerical algorithms and software architecture of WARP3D were supported by Grant SCCA 90–82144 from the Illinois Department of Commerce and Grant DE–FG02–85ER25001 from the Department of Energy made to the Center for Supercomputer Research and Development at the University of Illinois.

Computational support for software development and verification was provided by the Department of Civil Engineering HP workstation network made possible, in part, by grants from the Hewlett–Packard Company. Supercomputer (Cray–90) access was made available by the National Aerodynamic Simulation facility operated by the NASA–Ames research center. The support of all these organizations is gratefully acknowledged.

Recent research and development efforts on improved 3-D fracture models have been sponsored by NASA Grant NCC2–5126.

The authors express their appreciation to colleagues Prof. Nick Aravas (University of Pennsylvania), Prof. C. Fong Shih (Brown University), Prof. Alan Needleman (Brown University) and Dr. Sam Key (Sandia National Laboratory) for many helpful suggestions on the implementation of nonlinear constitutive models.

Contents

List of Figures

Chapter 1

Introduction

1.1 What is WARP3D?

This manual describes commands and theoretical background material necessary to use the WARP3D finite element code. WARP3D is under continuing development as a research code for the solution of very large-scale, 3-D solid models subjected to static and dynamic loads. Specific features in the code oriented toward the investigation of ductile fracture in metals include a robust finite strain formulation, a general J-integral computation facility (with inertia, face loading, thermal loading), very general 3-D element extinction and node release facilities to model crack growth, nonlinear material models including viscoplastic effects, and the Gurson-Tvergaard dilatant plasticity model for void growth.

The nonlinear, dynamic equilibrium equations are solved using an incremental-iterative, implicit formulation with full Newton iterations to eliminate residual nodal forces. Time history integration of the nonlinear equations of motion is accomplished with Newmark's β method. A central feature of WARP3D involves the use of a linear-preconditioned conjugate gradient (LPCG) solver implemented in an element-by-element format to replace a conventional direct linear equation solver. This software architecture dramatically reduces both the memory requirements and CPU time for very large, nonlinear solid models since formation of the assembled (dynamic) stiffness matrix is avoided. Analyses thus exhibit the numerical stability for large time (load) steps provided by the implicit formulation coupled with the low memory requirements characteristic of an explicit code. In addition to the much lower memory requirements of the LPCG solver, the CPU time required for solution of the linear equations during each Newton iteration is generally one-half or less of the CPU time required for a sparse, direct solver. All other computational aspects of the code (element stiffnesses, element strains, stress updating, element internal forces) are implemented in the element-by-element, blocked architecture. This greatly improves vectorization of the code on uni-processor hardware and enables straightforward parallel-vector processing of element blocks on multi-processor hardware (see Carey and Jiang [11], Flanagan and Taylor [25], Hughes, Ferencz, and Hallquist [43], Healy, Pecknold and Dodds [33] for detailed discussions of blocking strategies).

For models which prove difficult to analyze with the LPCG solver due to poor conditioning, e.g., thin shell structures modeled with solid elements, WARP3D provides a family of very efficient, sparse matrix solvers based on multi-minimum degree re-ordering of the equations. The sparse solvers dramatically reduce both memory and CPU times required for solution of the linearized equations compared to a traditional profile based solver; factors of 5-15 reduction in memory and CPU time are routinely found. Moreover, the sparse solver becomes very competitive with the LPCG solver on Unix workstations which have slow memory subsystems (relative to their floating point speeds and to the memory systems on Cray computers, see McCalpin [[59]). The LPCG solver performs far fewer floating point operations to obtain a solution but must repeatedly cycle over element matrices hundreds-to-thousands of times which increases the relative importance of memory access times. To

[†] Numbers in [ ] indicate references listed in Appendix B.

reach the maximum possible performance, WARP3D invokes sparse solvers especially tuned for each hardware platform. These solvers are usually provided in numerical libraries by the computer vendor. A "generic" sparse solver is available in all platforms to maintain portability.

Research continues to focus on the application of *nonlinear* pre-conditioned conjugate gradient (NLPCG) solvers for solution of large-scale, 3-D finite element models (see for example Biffle [7], [8] [9], Hughes, Ferencz, and Hallquist [43]). The *JAC* codes of Biffle [8] [9] employ NLPCG solvers for the analysis of large, quasi-static solid models. Experience with these codes quickly points out the dominant role played by the relative efficiency of numerical implementations for constitutive models to update stresses. In contrast to the LPCG approach during which stresses are updated *outside* the linear equation solving process, the material state requires updating *inside* each iteration of each load (time) step in the NLPCG approach. The number of NLPCG iterations per step can easily exceed 1000 for even moderate size problems. For simple constitutive models that may be *fully* vectorized, e.g., rate-independent Mises plasticity with a constant hardening, the NLPCG approach has the potential to be very robust and computationally efficient. However, for the increasingly complex nonlinear constitutive models employed in modeling ductile fracture, for example, the stress update routines become very difficult to vectorize and to date are partially vectorized (these models require multi-levels of local Newton solutions to update material state variables). Consequently, the potential benefits offered by NLPCG compared to LPCG are diminished severely. The architecture of WARP3D is designed to accommodate the NLPCG approach in the future should that evolution path for the code become advantageous.

Using WARP3D with the current LPCG strategy, 3-D models containing 30,000-50,000 elements are routinely analyzed on supercomputers (Crays). Models with 8,000 8-node brick elements fit in main memory on 64 MB desktop workstations. They solve with dramatically reduced elapsed times compared to commercial software since no spilling to disk occurs during equation solving coupled with the generally better CPU efficiency of the LPCG solver relative to a conventional direct solver.

WARP3D executes in batch and interactive modes. Traditional batch mode execution is most useful for large analyses on supercomputers which enforce job queuing policies. On Unix workstations, the code is often executed in background (&) mode for long jobs and then interactively during an analysis restart to obtain selected output. Options exist to write information files describing the solution status at completion of each Newton iteration during long analyses executed in batch mode.

WARP3D takes input data from a variety of sources under control of the user. A Patran-to-WARP3D translator program (*patwarp*) is also available to convert a Patran neutral file for the model into a WARP3D input file. Input commands to define the model, loading history, solution parameters, compute and output requests have a format-free, English-like structure. Input files may include extensive user comments and thus are generally self-documenting. Output consists of traditional printed displacements, strains, stresses, etc. in addition to nodal results files in standard Patran format (binary or ascii) written directly by WARP3D. A convenient restart capability provides the facility to segment a long job over multiple runs and to create analysis recovery files in the event of hardware failures or should the solution not converge.

This manual is organized as follows. The remainder of Chapter 1 provides an overview of WARP3D through discussion of an example problem, and background material on the formulation and solution of the governing equations. Chapter 2 describes the commands to define the finite element model, loading history, nonlinear/dynamic solution parameters, compute and output commands. Chapter 3 provides a detailed description of the currently available finite elements and material models. Chapter 4 describes the procedures and commands available to compute J-integrals using domain integral techniques. Chapter 5 discusses the procedures and commands to model crack growth. The appendices provide additional details such as the format of nodal results files generated for use in Patran.

A Note About Physical Units

WARP3D does not provide facilities for units conversions. Users are required to specify consistent physical units for all quantities defining the finite element model and loading.

1.2 Illustrative Problem

This section describes the nonlinear analysis of a pre–cracked Charpy–V–Notch (CVN) specimen subjected to impact loading typical of that experienced in a standard, constant velocity test. Figure 1.1 shows the finite element model, dimensions, boundary conditions and loading history. In this example, the 3–D model has one–layer of elements in the thickness direction with plane–strain constraints (w=0) imposed on all nodes. The model has 2008 nodes and 916 elements (8–node bricks with $\bar{B}$ modification). The model was developed and analyzed to support an investigation of crack tip inertia and viscoplastic effects on the near–tip stress fields which drive cleavage fracture in ferritic materials.

The analysis uses the small–strain kinematic formulation with viscoplastic material behavior. Rate–dependent properties characteristic of A533B steel at 100°C are specified. The uniaxial (tensile) inviscid response follows a power–law hardening model (n=10) after yield at σ_0; the viscoplastic response follows a power–law model with an exponent of 35 and a reference strain rate of $1/s$. Displacements imposed at the hammer impact point increase from zero as indicated in the figure to generate a constant velocity loading of 120 in/s after an elapsed time of 5 μs. The analysis covers 200 μs duration in 400 steps with a constant time increment of 0.5 μs. The remainder of this section describes features of the WARP3D input to define the model, loading history, request computations and output, and to compute J–integrals shortly after impact.

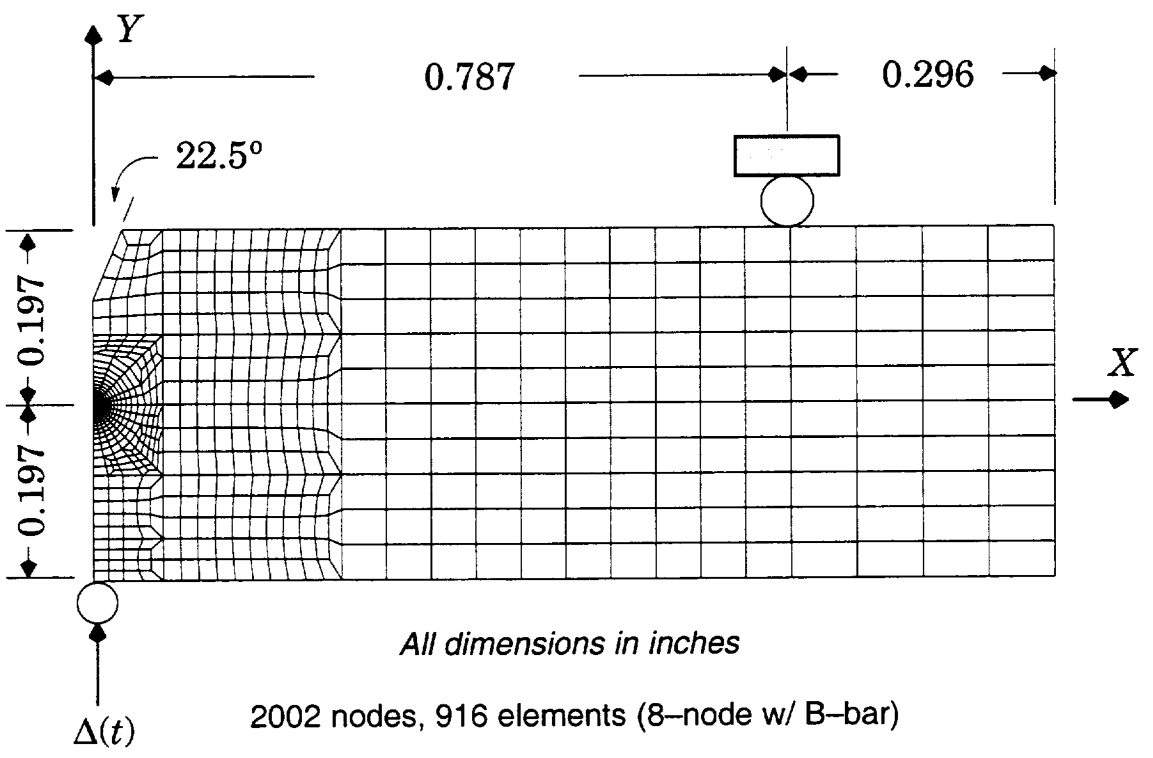
Input for the model begins with a structure command and material definitions.

```
c
c     example cvn analysis with WARP
c
c
structure cvn
c
c
material  a533b
    properties  mises e 30000 nu 0.3 yld_pt 60 n_power 10,
       ref_eps 1.0 m_power 35.0 rho  7.29275e-07
```

WARP3D commands are format free and may begin anywhere on the line. One or more blanks separate data items. A 'c' in column 1 denotes a comment line and is ignored by the input translator. A comma (,) at the end of a line indicates that the input for that command continues on the next line. In the above sequence, we assign a convenient name for the problem (cvn) which appears on all printed output and forms the initial part of some output file names. We define a material named *a533b* (any convenient id) and the 'type' of constitutive model as *mises*. Up to 10 materials may be defined as above for subsequent assignment to elements. User assignable properties for the model are specified as shown, with a keyword label followed by a data value. Keywords have easily interpreted names and may be given in any order. Decimal points are optional and may be omitted if not needed to specify the fractional part of a number. Some keywords specify "logical" data values; appearance of the keyword in the input sets the corresponding property value .true. Property *rho* denotes the mass density of the material.

Following the structure id and material definitions, the structure sizes and nodal coordinates are specified as illustrated below:

```
c
number of nodes   2002
number of elements   916
c
```



Material Properties

$E = 30{,}000\ ksi$	$n = 10$ (inviscid power – law hardening)
$\nu = 0.3$	$m = 35$ (viscoplastic power)
$\sigma_0 = 60\ ksi$ (inviscid)	$\dot{\epsilon}_{ref} = 1\ in/in/s$ (reference strain rate)
$\rho = 7.29275 \times 10^{-7}\ kip - s^2/in$	

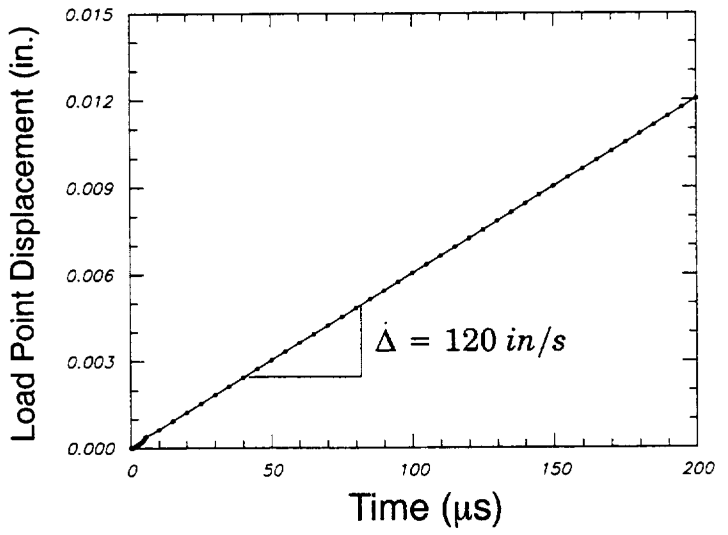


FIG. 1.1—*Pre–cracked Charpy specimen used in illustrative problem.*

```
*echo off
coordinates
     1   .100900006E+01  -.196999982E+00   .000000000E+00
     2   .108300006E+01  -.196999982E+00   .000000000E+00
                           .
                           .
```

The model sizes are required to properly allocate space for internal data arrays. The order of commands to define the sizes is immaterial, and a command of the form *number of nodes 2002 elements 916* applies as well. Nodes and elements must be numbered sequentially and must not have "holes" in the numbering. The **echo off* suppresses data echo of commands as read from the current input file. Various * commands may be specified at any point in the input stream to control the echo, switch to another file for input, etc. Coordinates for nodes are defined in the global *X–Y–Z* system with the origin located at a convenient location. Coordinates for nodes may be specified any number of times; the last specified set of coordinates are retained for analysis. The coordinates here were translated from a Patran neutral file for the model by the *patwarp* program and thus have the *E* format shown.

The 'incidences' define the connectivity of each element node to the corresponding structure node.

```
c
incidences
     1     5     1     4     8     6     2     3     7
     2     8     4    10    12     7     3     9    11
     3    12    10    14    16    11     9    13    15
     4    16    14    18    20    15    13    17    19
     5    20    18    22    24    19    17    21    23
     6    24    22    26    28    23    21    25    27
                           .
                           .
```

Chapter 3 describes the ordering of nodes on the element and the relationship of element nodes to the ordering of Gauss points. Elements may be entered in any order; the last specified set of incidences for the element applies in the analysis. The input translators perform extensive checks on the specified incidences to insure there are no gross errors (e.g., nodes with no elements attached).

The type of each element and the properties for each element are specified next.

```
c
elements
  1-916 type l3disop linear bbar material a533b order 2x2x2
```

In this example, all elements are the 8–node isoparametric (*l3disop*) with a small–strain kinematic formulation (*linear*). The $\overline{\mathrm{B}}$ modifications to prevent locking under plastic deformation are requested (*bbar*, a logical property). The previously defined material *a533b* is associated with these elements and the standard 2x2x2 Gauss integration is requested. Other element properties available invoke various output options. All elements have the same material and properties in this example. When this is not the case, any number of similar input lines may be defined to specify the properties. The *integerlist* construction (1–916 above) is convenient and may be used anywhere a list of integers is needed in the input stream. A more general example of an *integerlist* is: 1–400 by 2, 800–600 by –2, 3000–6000 492 496 ...

Each element in the model must be assigned to a "block" for computation. Blocking is required to support optimum vector/parallel operations on supercomputers and is retained for analyses conducted on Unix workstations. All elements in a block must be the same type (e.g. *l3disop*), have the same material, the same type of kinematic formulation, the same values of some element properties (e.g., integration order, $\bar{\boldsymbol{B}}$) and must not be connected to a common node in the model. This last restriction does not apply for analyses conducted on scalar computers (most Unix workstations) unless the conjugate gradient solver uses the Hughes–Winget preconditioner. The maximum number of elements per block varies with the computer hardware. On Crays, the block size is normally 128 to accommodate vector registers of 128 words in length. On workstations, the cache memory size dictates an optimum block size (usually 32–64).

In this example, the block size is 32; the block number is specified followed by the number of elements in the block and the first element in the block. Elements appearing in a block must be sequentially numbered with no holes. The *patwarp* program which converts a Patran neutral file to a WARP3D input file performs automatic blocking of the elements using a red–black algorithm. The input processors in WARP3D perform exhaustive checks to verify that the rules for blocking assignments are satisfied.

```
c
blocking
      1     32      1
      2     32     33
      3     32     65
      7     32    193
            .
            .
     28     32    865
     29     20    897
```

Nodal constraints in this analysis enforce the plane–strain conditions, the symmetry conditions (u=0) on the crack plane, the v=0 condition at the top, right roller support and the imposed loading to simulate a constant velocity response. A portion of the constraint input is shown below. The specified constraints are the *incremental* displacements imposed over the model during each load (time) step. Constraints may be re–defined as necessary between load steps. When modified, *all* constraints must again be specified. Nodes 499 and 503 in this model are the two nodes at the hammer impact point in the thickness direction. The constraints shown here are applied during load steps 1 and 2. Then a new set of constraints is defined for application in steps 3, 4 with the imposed increments at nodes 499 and 503 doubled in value. Similarly, during steps 5, 6 the v increment at nodes 499, 500 is 3.0E–05; during steps 7, 8 the v increment at nodes 499, 500 is 4.0E–05; and finally during steps 9–400 the v increment at nodes 499, 500 is 6.0E–05. The load point velocity (120 *in/s*) remains constant over steps 9–400 and is simply the imposed displacement increment / Δt (in this case 6.0E–0/5.5E–06). The slow increase in load point velocity minimizes spurious oscillations in the response.

```
constraints
      1        w       0.0
      2        w       0.0
      3        w       0.0
      4        w       0.0
      5        w       0.0
      6        w       0.0
               .
               .
c
c
```

```
      499       v    1.0e-5
      503       v    1.0e-5
```

Loads may be applied to the nodes and elements of a model. Element loads, which are dependent on the type of finite element, are converted to equivalent nodal loads by element processing routines. Nodal loads and element loads are grouped together to define *loading patterns.* The loading patterns define the spatial variation and reference amplitudes of loads on a model. Examples of loading patterns include dead load, an internal pressure, simple bending of a component or specified nodal–element temperatures.

A *nonlinear* loading condition is declared using previously defined patterns. The term *dynamic* may be used as a synonym for *nonlinear* if desired. A nonlinear/dynamic loading consists of a sequential number of load steps. An incremental–iterative solution is obtained for each load step. For dynamic analyses, a load step is the same as a time step. Each *load step* may consist of loading patterns combined with scalar multipliers. The scaled values of nodal forces (nodal loads and resulting equivalent nodal loads) for the patterns are applied as the new *incremental* load to the model during the step. Loading commands for this example are shown below.

```
c
loading null
  nodal loads
        401 force_y 0
c
 loading disp_ctrl
  dynamic
      step 1-2 null 1.0
      step 3-4 null 2.0
      step 5-6 null 3.0
      step 7-8 null 4.0
      step 9-400 null 6.0
c
```

In this analysis of the CVN specimen, no real "loadings" are needed since the model is loaded by enforced displacements. Nevertheless, a "dummy" loading pattern must be defined to satisfy the syntax requirements for the dynamic loading. Here, the dummy loading is assigned the id "null." The dynamic loading is assigned the id "disp_ctrl." All 400 steps are defined above although this is not required; additional steps may be defined later during the analysis. The scalar multipliers assigned to the pattern (1.0, 2.0, 3.0, 4.0, 6.0) above refer to the relative change in the magnitude of displacement increments. For displacement control loading, these multipliers come into use during extrapolation of displacements from step n to n+1 for accelerating convergence of the Newton iterations.

The user may specify values for a number of nonlinear/dynamic parameters that control the solution procedures In this example, we specify

```
c
 dynamic analysis parameters
  solution technique direct sparse sgi
  maximum iterations 5
  convergence test norm residual tol 0.0005
  nonconvergent solution stop
  time step 0.5e-6
  extrapolate on
  adaptive solution on
```

```
      material messages off
      batch messages on
    c
```

A few keywords describing the option are given followed by a required value(s). Some parameters have numerical values while others have *on, off* values and others just end with a keyword. Most parameters have suitable default values. A brief explanation of each parameter specified above follows:

- The linear equation solver is specified as *direct sparse*—an in-memory direct solver that employs sparse matrix technology to reorder the equations for very efficient decomposition. This solver is efficient for 2–D type models, such as this example, and for moderate size 3-D models. The primary equation solver for large 3–D models uses the linear, preconditioned conjugate gradient algorithm and is requested by the option *lpcg* rather than *direct.*
- The maximum number of Newton iterations to eliminate residual forces in each step is set to 5.
- The Newton convergence test specifies a tolerance of 0.05% on the Euclidean norm of the residual forces relative to the Euclidean norm of the current (total) load vector. Solutions that fail to converge cause termination of the analysis unless the default *stop* value for the *nonconvergent solutions* is changed to *continue.*
- The time step is 5 μs for use in Newmark's β method to integrate the dynamic equilibrium equations.
- *extrapolate on* invokes a nonlinear solution option which imposes the scaled displacement increment computed for step *n* on the model to start the solution for step *n*+1. This option greatly accelerates the convergence of Newton iterations for displacement controlled loading.
- The nonlinear *adaptive* strategy is requested; load steps are automatically sub–incremented and re–solved when the specified limit on the number of Newton iterations is reached without convergence. Two levels of adaptivity are available which subdivide, at most, a user specified step into 16 sub–steps. Adaptive solutions that do not converge are terminated and a restart file written.
- Material models (by default) issue messages which notify of first yielding, reversed yielding, and other state changes. These messages are suppressed with *material messages off.*
- This analysis is executed in "batch" mode (&) on a workstation. The *batch messages on* parameter requests that WARP3D write a solution status file following each Newton iteration. The file names are *wm_xxxx_yy* where *xxxx* denotes the step number and *yy* denote the Newton iteration. These files provide information about convergence of the solution.

The model, loading history and solution parameters are now defined. Commands to request an analysis and output of results are given. For the first 10 load steps the commands are:

```
    c
      compute displacements for loading disp_ctrl for step 1 2
    *echo off
    *input from 'forty'
    *echo on
    c
      compute displacements for loading disp_ctrl for step 3 4
    c
     echo off
    *input from 'sixty'
    *echo on
    c
      compute displacements for loading disp_ctrl for step 5 6
    c
```

```
*echo off
*input from 'eighty'
*echo on
c
  compute displacements for loading disp_ctrl for step 7 8
c
*echo off
*input from 'one-twenty'
*echo on
c
 compute displacements for loading disp_ctrl for step 9 10
c
 save to file 'cvn_step_10'
c
   output displacements node 798
   output velocity node 798
   output wide eformat strains elements 20-40
   output wide eformat stresses elements 20-40
   output accelerations for elements 100-200 by 2
   output internal_forces 109,110
   output internal_forces 499,503
   output patran binary displ stress strains velocity accelerations
*input from 'domain_define'
stop
```

Here, we request computation of results for load steps 1–2 and then switch the input stream to a file named *forty*. This file contains an entire new set of constraints for the model (the incremental displacements imposed on nodes 499, 503 are increased to 2.0e–5 from 1.0e–5). The first few and last few lines of the file *forty* are

```
constraints
     1      w     0.0
     2      w     0.0
     3      w     0.0
            .
            .
  2002      w     0.0
c
  499       v    2.0e-5
  503       v    2.0e-5
```

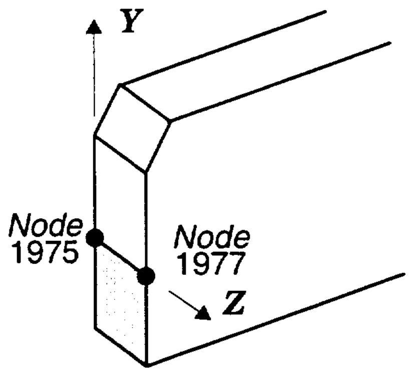
The * commands turn off the data echo while the new constraints are being read and then resume the data echo (this is just for convenience and may be omitted). The **input* command specifies the file name for input. We could just as easily have placed the contents of file *forty* in the current input file. The WARP3D input processors sense when the end–of–file condition on *forty* occurs and automatically resume reading from the previous input stream. This sequence of commands is repeated to continue the analysis through load step 10, and in the process ramp the imposed load point velocity to 120 *in/s*.

Following completion of the analysis for load step 10, we issue a *save to file* ... command which forces creation of an analysis restart file (sequential, binary) named *cvn_step_10*. The choice of file name resides with the user. This file enables resumption of the analysis at load step 11 in a future program execution (as illustrated subsequently).

Several *output* commands are defined to request printing (to the *current* output device) of nodal and element values (displacements, velocities, accelerations, strains, stresses). These results are displayed in tabular form with appropriate page and column headers. The *internal forces* are reactions at constrained nodal dof. The *output patran* ... command requests creation of *binary* (ascii is optional) files of nodal values written in the required format for direct post–processing by Patran. These files have the names *pbd#####* , for example, where *pbd* denotes 'patran binary displacements' and ##### indicates the load step number. Appendix A defines the format of Patran results files created by WARP3D.

Finally, an **input* command is specified to read more input from the file *domain_define*. This file contains the input commands

```
c
 domain one
  symmetric
  front nodes  1975 1977  linear
  normal plane nx 1 ny 0 nz 0
  q-values automatic rings  31-35
  print totals
  function type d
 compute domain integral
c
```



We define one "domain" for J–integral computation using the results for load step 10. A domain is defined by specifying an "id" (*one* in this example for output headers), the nodes in the domain along the crack front, the q–function interpolation order along the front, the orientation of the crack plane relative to the global coordinate system, the number and types of "rings" for J–evaluation and output options. The *rings 31–35* option requests that the first J–value be computed using elements in the 31st ring of elements enclosing the crack front. J–values are then computed over rings 32–35. Values for each ring are printed and statistics shown to assess the path (domain) independence of the values. The *symmetric* parameter causes the code to double J–values prior to printing.

In this (effectively) 2–D model, we request computation of a "through–thickness" average J–value by specifying *function type d*. In general 3–D models, we specify the sequence *domain ... compute domain integral* at each point on the crack front where J–values are required. WARP3D automatically determines that the analysis is dynamic and includes the inertia terms in J and crack face loadings if they are present as well.

The input file ends with a *stop* command which terminates program execution. *Restart files must be explicitly requested with the "save" command.*

To restart the analysis at load step 11 in a new execution of the program, the input file for this example begins with the commands

```
c
 retrieve from file 'cvn_step_10'
c
 output displacements 100-200
c
 dynamic analysis parameters
  maximum iterations 4
  convergence test norm residual tol 0.001
  material messages on
  batch messages off
c
 compute displacements for loading disp_ctrl for step 11-20
 save to file 'cvn_step_20'
c
   output displacements node 798
   output velocity node 798
```

```
        output wide eformat strains elements ...
           .
           .
           .
```

The *retrieve* command must be the first non–comment line in the restart file. WARP3D reads this file to restore all internal variables to their values at completion of load step 10. Another *output* command requests more results for step 10 and then several analysis parameters are modified. The analysis for steps 11–20 is requested and the computations are finished, another restart file is created, output commands to print results at step 20 issued, etc.

1.3 Manual Conventions

The input translators for WARP provide a problem oriented language command structure to simply specification of model and solution parameters. This section describes the conventions and notation employed throughout the manual to explain commands.

The appearance within a WARP command of a descriptor of the form

< integer >

implies that the user is to enter an item of data within that position in the statement of the class described by the descriptor (in the above example an integer). The command

number of nodes < integer >

implies that the word *nodes* is to be followed by an integer, such as 1000 or 6870, and that the statement entered by the user as input data should be of the form

```
number of nodes 6870
```

The following are definitions of most of the descriptors used within the language. Those not described below are explained when they first occur in the text.

< integer >	a series of digits optionally preceded by a plus or minus sign. Examples are 121, +300, –410.
< real >	a series of digits with a decimal point included, or series of digits with a decimal point followed by an exponential indicating a power of 10. Real numbers may be optionally signed. Examples are 1.0, –2.5, 4.3e–01.
< number >	is either a < real > or an < integer >. The input translator performs mode conversion as needed for internal storage.
< label >	is a series of letters and digits. The sequence must begin with a letter. Input translators also accept the character underbar, _ , as a valid letter. Labels may have the form big_cylinder, for example, to give the appearance of multiple words for readability.
< string >	is any textual information enclosed in apostrophes (') or quotes ("). An example is 'this is a string'.
< list >	is the notation used to indicate a sequence of positive integer values — usually node and element numbers. Lists generally contain two forms of data that may be intermixed with the same list. The first form of data is a series of integers optionally separated by commas. An example is 1, 3, 6, 10, 12. The second common form of a list implies a consecutive sequence of integers and consists of two integers separated by a hyphen. An example is 1–10, which implies all integers in the sequence 1 through 10. An extension of this form implies a constant increment, e.g., 1–10 by 2 implies 1, 3, 5, 7, 9. A third form, *all*, is sometimes permitted, and implies all physically meaningful integers. The forms of lists are often combined as in ... *nodes 1–100 by 3, 200–300, 500–300 by –3.*

Input to WARP appears as a sequence of English–like commands. Many of the words or phrases in these commands are optional and are permitted for readability or to specify options with a command. In the definition of each command, underlined words are required for proper operation of the input translators. If a portion of a word is underlined, only the underlined portion is required input. Items such as <integer> shown in the command definitions are not underlined but must always be replaced by an item of the specified class. For example, the command phrase defined by

$$\underline{\text{numb}}\text{er } (\underline{\text{of}}) \ \underline{\text{node}}\text{s} < \text{integer} >$$

can be shortened to

```
numb of node 10
```

if the user so desires.

In many instances, more than one word is acceptable in a given position within a command. The choices are listed one above the other in the command definition. The command definition

$$\underline{\text{comp}}\text{ute} \begin{Bmatrix} \underline{\text{displa}}\text{cements} \\ \underline{\text{domain}} \end{Bmatrix}$$

indicates that each of the following commands are acceptable

```
compute domain
compute displacements
comp displa
```

Optional words and phrases are enclosed with parentheses, (). In some commands, items may be repeated and/or multiple phrases may be combined on one data line. This is indicated in the command definition by enclosing the repeatable entries within brackets, [ ]. The command

$$< \text{integer} > \left[\begin{Bmatrix} \text{X} \\ \text{Y} \\ \text{Z} \end{Bmatrix} < \text{number} > \ (,) \right]$$

implies that the following sequences are valid:

```
1 x 10 y 10 z 15.3
2 x 15 z 30
30 z -42.5
```

In order to be more descriptive within the command definitions, actual data items (those denoted with <> in the definition) are sometimes described in terms of their physical meaning and followed by the type or class of data item which can be used in the command. For example the command,

$$\underline{\text{struc}}\text{ture} < \text{name of structure: label} >$$

implies that the data item following the word *structure* is the name of the structure and must a descriptor of type < label >. Examples of acceptable commands are

```
structure cylinder
struct big_block
```

while

```
structure 1a
```

is not acceptable since the name of the structure is not a label (*labels must begin with a letter*).

Continuation Lines

A comma (,) placed at the end of a line causes the subsequent data line to be considered a logical continuation of the current line. There is no limit on the number of continuation lines. Continuation can be invoked at any point in any command.

Comment Lines

Comments may be placed in the input following a Fortran style. The letter 'c' or 'C' appearing in physical column 1 of the data line marks it as a comment line. The line is read and (possibly) echoed by the input translator. The content is ignored and the next data line read.

Line Termination

Line termination is accomplished in one of three ways. First, the last column examined by the input translators is column 72. Secondly, after encountering the first data item on a card, the translators count blanks between data items. If 40 successive blanks are found, the remainder of the line is assumed blank. Finally, a $ indicates an end of line. Space following the $ is ignored by the input translators and is often used for short comments.

1.4 Nonlinear Equations of Motion

The structure occupies the configuration $\boldsymbol{B}_0$ at time $t = 0$ and evolves through time to the deformed configuration $\boldsymbol{B}$ at time t. In the $\boldsymbol{B}_0$ configuration, the structure is undeformed and at rest. In reaching the deformed configuration, the structure may displace in any manner, including simple rigid body translation or rotation in the absence of true deformation. This situation is illustrated in Fig. 1.2. The position vector $\boldsymbol{X}$ identifies a point in the undeformed configuration and $\boldsymbol{x}$ denotes the position vector of the same point in the deformed (current) configuration. The vector $\boldsymbol{d}$ is the displacement vector that takes the point from the initial to the deformed configuration. The coordinates of the structure in the reference configuration represent the geometry interpolated from the parametric coordinates in the isoparametric formulation. The nonlinear implementation of the finite element method in WARP3D employs a continuously updated formulation naturally suited for solids with only translational dof at the nodes. The expression of virtual work defining equilibrium and the equations of motion are defined and solved on the current, $\boldsymbol{B}$, configuration. Throughout the deformation history of the structure, this choice of reference configuration remains in effect.

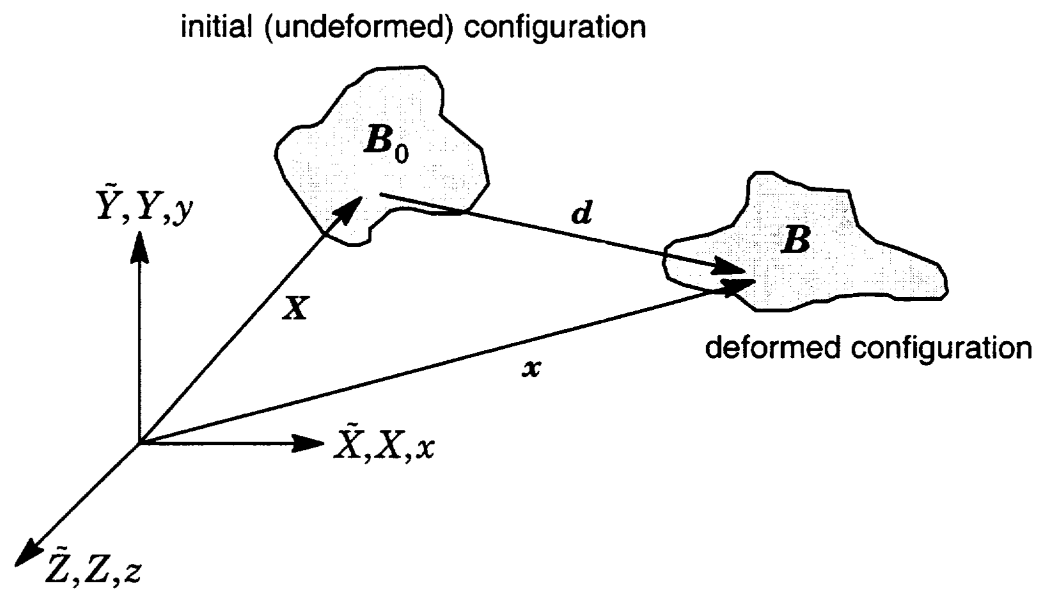


FIG. 1.2—*Definition of initial and current (deformed) configurations. Equations of motion are written on the deformed configuration.*

In the remainder of this section, the equations of motion are derived. Methods for solution of the resulting nonlinear algebraic equations are described in subsequent sections and followed by descriptions of the specific finite element formulations and the adopted formulation to model finite strains and rotations.

The weak formulation of momentum balance equations (virtual work) expressed in the current configuration is given by

$$\int_V \delta\boldsymbol{\epsilon}^T\boldsymbol{\sigma}\, dV - \int_V \delta\boldsymbol{d}^T\boldsymbol{f}\, dV - \sum_{i=1}^{m} \delta\boldsymbol{d}_i^T\boldsymbol{p}_i = 0 \qquad (1.1)$$

where V denotes the current volume, $\delta\boldsymbol{\epsilon}$ and $\boldsymbol{\sigma}$ are the virtual rate of deformation vector and the Cauchy stress vector, $\boldsymbol{f}$ is the body force vector per unit volume in the deformed configuration, and each $\boldsymbol{p}_i$ is a 3×1 vector of external forces acting at m discrete points (see Malvern [57] , Marsden and Hughes [58]). We use 6×1 vector forms of the symmetric tensors for $\delta\boldsymbol{\epsilon}$ and $\boldsymbol{\sigma}$. The operator δ denotes a small, arbitrary virtual variation. The virtual rate of deformation tensor and the Cauchy stress tensor form a work conjugate pair when defined on the current configuration.

External force vectors remain constant in magnitude and direction over a load step. The nodal forces $\boldsymbol{p}_i$ may comprise directly applied nodal forces and the (work) equivalent nodal forces due to specified surface tractions applied on element faces and other body forces, e.g., self–weight. Inertial D'Alembert forces arising from accelerations are given by

$$\boldsymbol{f} = -\rho\ddot{\boldsymbol{d}} \tag{1.2}$$

where ϱ is the mass density in the deformed configuration. By including acceleration forces in $\boldsymbol{f}$ and body forces due to self–weight in $\boldsymbol{p}_i$, Eq. (1.1) becomes

$$\int_V \delta\boldsymbol{\epsilon}^T\boldsymbol{\sigma} dV + \int_V \delta\boldsymbol{d}^T\rho\ddot{\boldsymbol{d}} dV - \sum_{i=1}^{m} \delta\boldsymbol{d}_i^T\boldsymbol{p}_i = 0 \ . \tag{1.3}$$

Following standard procedures (Cook et. al [16], Hughes [44]), Eq. (1.3) transforms from a purely continuum form to an (equivalent) finite element form as given below, beginning with integrations over each element to define the volume integral over the structure

$$\sum_{j=1}^{\#elem} \int_{V_e^j} \delta\boldsymbol{\epsilon}^T\boldsymbol{\sigma} dV_e + \sum_{j=1}^{\#elem} \int_{V_e^j} \delta\boldsymbol{d}^T\rho\ddot{\boldsymbol{d}} dV_e - \sum_{i=1}^{m} \delta\boldsymbol{d}_i^T\boldsymbol{p}_i = 0 \tag{1.4}$$

$$\sum_{j=1}^{\#elem} (\delta\boldsymbol{u}_e^T\boldsymbol{I}_e)_j + \sum_{j=1}^{\#elem} (\delta\boldsymbol{u}_e^T\boldsymbol{M}_e\ddot{\boldsymbol{u}}_e)_j - \sum_{i=1}^{m} \delta\boldsymbol{d}_i^T\boldsymbol{p}_i = 0 \tag{1.5}$$

$$\delta\boldsymbol{u}^T\left(\sum\boldsymbol{I}_e + \left(\sum\boldsymbol{M}_e\right)\ddot{\boldsymbol{u}} - \boldsymbol{P}\right) = 0 \tag{1.6}$$

where $\boldsymbol{u}$ is the global nodal displacement vector, $\boldsymbol{u}_e$ is an element nodal displacement vector, $\boldsymbol{I}_e$ is an element internal force vector, $\boldsymbol{M}_e$ is an element mass matrix, and $\boldsymbol{P}$ is the global external force vector. Subsequent sections outline procedures to compute the element internal force vector and the element mass matrix as well as the element tangent stiffness matrix. The summations in Eq. (1.6) denote the global assembly process. Since the $\delta\boldsymbol{u}$ are arbitrary in nature,

$$\sum\boldsymbol{I}_e + \left(\sum\boldsymbol{M}_e\right)\ddot{\boldsymbol{u}} - \boldsymbol{P} = 0 \ . \tag{1.7}$$

After performing the assembly processes implied by the Σ in Eq. (1.7), the global equation of motions become

$$\boldsymbol{I} + \boldsymbol{M}\ddot{\boldsymbol{u}} = \boldsymbol{P} \ . \tag{1.8}$$

The vectors have size $3 \times m$, where m denotes the number of structure nodes. Nonlinearity in $\boldsymbol{I}$ arises from the element internal force vectors (geometric and/or material effects) while $\boldsymbol{P}$ become nonlinear when tractions applied to element faces have constant orientation relative to the deformed face (e.g., pressure loads).

1.5 Dynamic Analysis: Newmark β Method

Numerical integration of the equations of motion in WARP3D is performed using a method attributed to Newmark [69]. This approach employs a two parameter family of equations that define the displacement, velocity, and acceleration at time t_{n+1} in terms of the displacement increment from t_n to t_{n+1} and the kinematic state at time t_n. These equations derive from successive application of the extended mean value theorem of differential calculus. Consider first the velocities at time t_n and t_{n+1}. Use of the extended mean value theorem for the first derivative leads to the equation

$$\dot{\boldsymbol{u}}_{n+1} = \dot{\boldsymbol{u}}_n + \Delta t\, \ddot{\boldsymbol{u}}_\gamma\,; \qquad \ddot{\boldsymbol{u}}_\gamma \in [\, \ddot{\boldsymbol{u}}_n\,,\, \ddot{\boldsymbol{u}}_{n+1}\,]\ . \tag{1.9}$$

Using the relationship

$$\ddot{\boldsymbol{u}}_\gamma = (1-\gamma)\ddot{\boldsymbol{u}}_n + \gamma\ddot{\boldsymbol{u}}_{n+1}\,; \qquad 0 \le \gamma \le 1 \tag{1.10}$$

Eq. (1.9) can be rewritten as

$$\dot{\boldsymbol{u}}_{n+1} = \dot{\boldsymbol{u}}_n + (1-\gamma)\Delta t\ddot{\boldsymbol{u}}_n + \gamma\Delta t\ddot{\boldsymbol{u}}_{n+1}\ . \tag{1.11}$$

Equation (1.11) provides an exact result for a given time interval if the parameter γ can be chosen correctly. Even so, the constant acceleration $\ddot{\boldsymbol{u}}_\gamma$ upon integration of Eq. (1.9) does not necessarily produce the correct displacement at time t_{n+1} in terms of the displacement and velocity at time t_n. Accordingly, the extended mean value theorem for the second derivative is invoked to yield

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta t\dot{\boldsymbol{u}}_n + \frac{\Delta t^2}{2}\ddot{\boldsymbol{u}}_\beta\,; \qquad \ddot{\boldsymbol{u}}_\beta \in [\, \ddot{\boldsymbol{u}}_n\,,\, \ddot{\boldsymbol{u}}_{n+1}\,]\ . \tag{1.12}$$

Again, a relationship having the form

$$\ddot{\boldsymbol{u}}_\beta = (1-2\beta)\ddot{\boldsymbol{u}}_n + 2\beta\ddot{\boldsymbol{u}}_{n+1}\,; \quad 0 \le 2\beta \le 1 \tag{1.13}$$

is employed to recast Eq. (1.12) as

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta t\dot{\boldsymbol{u}}_n + \frac{(1-2\beta)}{2}\Delta t^2\ddot{\boldsymbol{u}}_n + \beta\Delta t^2\ddot{\boldsymbol{u}}_{n+1}\ . \tag{1.14}$$

Equation (1.14) also provides an exact for a given time interval as long as the choice of the parameter β proves to be correct. Of course, in general it is impossible to choose either γ or β correctly without knowing the solution in advance, so that the approximation in the Newmark method lies in the choice of γ and β. Newmark showed that to avoid spurious damping in linear systems, the parameter γ should equal $1/2$. The pertinent equations of the Newmark method then become

$$\dot{\boldsymbol{u}}_{n+1} = \dot{\boldsymbol{u}}_n + \frac{\Delta t}{2}(\ddot{\boldsymbol{u}}_n + \ddot{\boldsymbol{u}}_{n+1}) \tag{1.15}$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta t\dot{\boldsymbol{u}}_n + \frac{(1-2\beta)}{2}\Delta t^2\ddot{\boldsymbol{u}}_n + \beta\Delta t^2\ddot{\boldsymbol{u}}_{n+1} \tag{1.16}$$

A wide variety of values for the parameter β are possible. For instance, setting β equal to zero leads to the second central difference method. A choice of $\beta = 1/6$ defines the linear acceleration method, wherein the acceleration is assumed to vary linearly over the time increment. The choice of $\beta = 1/4$ produces the constant average acceleration method. Newmark demonstrated that $\beta = 1/4$ renders the method unconditionally stable for linear problems; other choices must satisfy a time step constraint to maintain stability throughout the solution. For materially nonlinear problems, Schoeberle and Belytschko [79] established that the use of $\beta = 1/4$ leads to unconditional stability when nonlinear equilibrium iterations (Newton) are performed to satisfy an energy convergence criterion, and for nonlinear elastic problems Hughes [39] found much the same situation. In WARP3D, $\beta = 1/4$ is the default value although users can modify this value.

Use of the Newmark method leads to an implicit dynamic formulation in that the solution of a nontrivial system of equations is required to compute a displacement increment. Assuming that β does not equal zero, Eqs. (1.15, 1.16) are manipulated to the form

$$\Delta \boldsymbol{u}_{n+1} = \boldsymbol{u}_{n+1} - \boldsymbol{u}_n \tag{1.17}$$

$$\dot{\boldsymbol{u}}_{n+1} = \frac{1}{2\beta\Delta t}\Delta \boldsymbol{u}_{n+1} - \frac{(1-2\beta)}{2\beta}\dot{\boldsymbol{u}}_n - \frac{(1-4\beta)}{4\beta}\Delta t\ddot{\boldsymbol{u}}_n \tag{1.18}$$

$$\ddot{\boldsymbol{u}}_{n+1} = \frac{1}{\beta\Delta t^2}\Delta \boldsymbol{u}_{n+1} - \frac{1}{\beta\Delta t}\dot{\boldsymbol{u}}_n - \frac{(1-2\beta)}{2\beta}\ddot{\boldsymbol{u}}_n \tag{1.19}$$

Equations (1.17–1.19) are substituted into the equations of motion and into the chosen iterative nonlinear solution algorithm. The total displacement increment for the current time step is computed, $\Delta \boldsymbol{u}_{n+1}$, and that increment is back substituted into Eqs. (1.17–1.19) to define the velocity and acceleration for the current estimate of the solution at time t_{n+1}.

1.6 Solution of Nonlinear Equations: Newton's Method

Recalling the equation of motion, the residual load vector at any time is expressed as

$$\boldsymbol{R} = \boldsymbol{P} - \boldsymbol{I} - \boldsymbol{M}\ddot{\boldsymbol{u}} \tag{1.20}$$

where $\boldsymbol{P}$ is the external load vector, $\boldsymbol{I}$ is the internal force vector, $\boldsymbol{M}$ is the mass matrix, and $\boldsymbol{u}$ is the nodal displacement vector. The residual defines the out–of–balance force vector that arises from nonlinear effects in $\boldsymbol{I}$ and (possibly) $\boldsymbol{P}$ computed for the current estimate of the nodal displacements, $\boldsymbol{u}$. An iterative solution designed to drive the residual to zero is desired. Newton's method for nonlinear equations, illustrated in Fig. 1.3 for a static analysis, can be derived by assuming that there exists an approximate displacement state, $\tilde{\boldsymbol{u}}$, in the neighborhood of the exact solution for which a linear mapping represented by

$$\boldsymbol{R}(\boldsymbol{u}) = \boldsymbol{R}(\tilde{\boldsymbol{u}}) + d\boldsymbol{R}(\boldsymbol{u}) = \boldsymbol{R}(\tilde{\boldsymbol{u}}) + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{u}} d\boldsymbol{u} \tag{1.21}$$

is a good approximation to the residual load vector. The partial derivative in Eq. (1.21) represents the Jacobian matrix which maps the displacement vector to the residual load vector. Presumably, a better approximation, $\tilde{\boldsymbol{u}} + d\boldsymbol{u}$, is obtained by setting Eq. (1.21) to zero. The differential increment of the residual load vector (the mass matrix for a given time step is constant), is given by

$$d\boldsymbol{R} = d\boldsymbol{P} - d\boldsymbol{I} - \boldsymbol{M}d\ddot{\boldsymbol{u}} \ . \tag{1.22}$$

The external loads are assumed to remain constant in direction and magnitude over a load (time) step and thus $d\boldsymbol{P}$=$\boldsymbol{0}$ (loads can change between steps). By using Eq. (1.19) to define the differential acceleration in terms of Newmark's method and by introducing the structure *tangent* stiffness, we have

$$\boldsymbol{M}d\ddot{\boldsymbol{u}} = \frac{1}{\beta \Delta t^2}\boldsymbol{M}d\boldsymbol{u} \ , \tag{1.23}$$

$$d\boldsymbol{I} = \boldsymbol{K}_T d\boldsymbol{u} \ . \tag{1.24}$$

where $\boldsymbol{K}_T$ denotes the tangent stiffness matrix for the structure. Equation (1.22) can then be written in the form

$$d\boldsymbol{R} = -\boldsymbol{K}_T^d d\boldsymbol{u} \tag{1.25}$$

where

$$\boldsymbol{K}_T^d = \boldsymbol{K}_T + \frac{1}{\beta \Delta t^2}\boldsymbol{M} \tag{1.26}$$

defines the dynamic tangent stiffness. The use of $d\boldsymbol{R}$ from Eq. (1.25) in Eq. (1.21) yields

$$\boldsymbol{R}(\boldsymbol{u}) = \boldsymbol{R}(\tilde{\boldsymbol{u}}) - \boldsymbol{K}_T^d d\boldsymbol{u} \tag{1.27}$$

which demonstrates that the dynamic tangent stiffness is the negative of the Jacobian matrix relating the residual load vector to the displacement vector:

$$\boldsymbol{K}_T^d = -\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{u}} \ . (1.28)$$

Setting Eq. (1.27) to zero and rearranging defines

$$\boldsymbol{K}_T^d d\boldsymbol{u} = \boldsymbol{R}(\tilde{\boldsymbol{u}}) \tag{1.29}$$

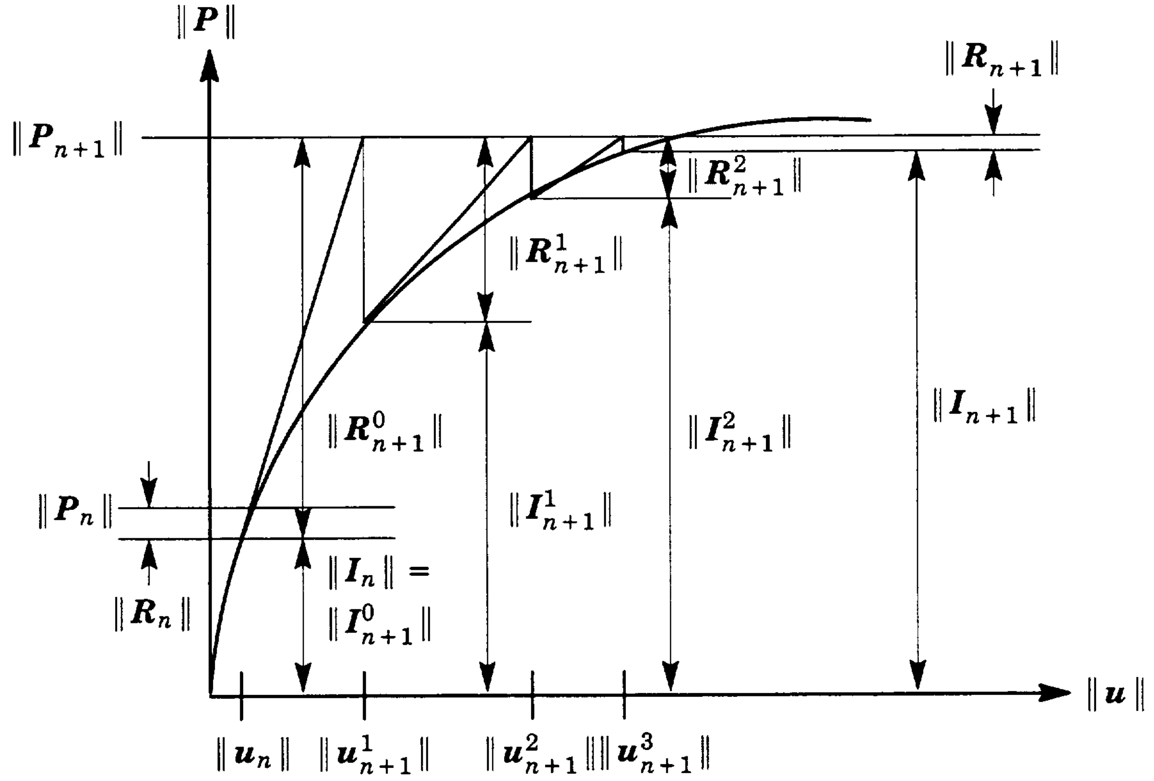


FIG. 1.3—*Illustration of Newton's method for a static analysis*

For finite, rather than differential, increments, the approximate form of Eq. (1.29) becomes

$$K_T^d \delta u_{n+1}^i = R_{n+1}^{i-1} \tag{1.30}$$

where δu_{n+1}^i denotes the (corrective) increment of displacement for the current iteration of the time step which advances the solution from n to n+1 and R_{n+1}^{i-1} denotes the residual load after the previous iteration. This residual is defined as

$$R_{n+1}^{i-1} = P_{n+1} - I_{n+1}^{i-1} - M\ddot{u}_{n+1}^{i-1} \tag{1.31}$$

or, after substitution of Eqs. (1.17–1.19), alternatively as

$$R_{n+1}^{i-1} = P_{n+1}^d - I_{n+1}^{i-1} - \frac{1}{\beta \Delta t^2} M \Delta u_{n+1}^{i-1} \tag{1.32}$$

where P_{n+1}^d is the applied load vector at time t_{n+1} modified by terms associated with Eqs. (1.17–1.19):

$$P_{n+1}^d = P_{n+1} + \frac{1}{\beta \Delta t} M\dot{u}_n + \frac{(1-2\beta)}{2\beta} M\ddot{u}_n \ . \tag{1.33}$$

The total change in displacement over the load step, through the current Newton iteration i for the step, is obtained from the summed corrective displacement vectors for the current step, i.e.,

$$\Delta \boldsymbol{u}_{n+1}^{i} = \sum_{k=1}^{i} \delta \boldsymbol{u}_{n+1}^{k} \tag{1.34}$$

with the updated estimate for the total displacements at step $n+1$ through iteration i is

$$\boldsymbol{u}_{n+1}^{i} = \boldsymbol{u}_n + \Delta \boldsymbol{u}_{n+1}^{i} \ . \tag{1.35}$$

The combination of Eqs. (1.30) and (1.32) defines the basic equation driving the iterative solution associated with the Newton method:

$$\boldsymbol{K}_T^d \delta \boldsymbol{u}_{n+1}^{i} = \boldsymbol{P}_{n+1}^{d} - \boldsymbol{I}_{n+1}^{i-1} - \frac{1}{\beta \Delta t^2} \boldsymbol{M} \Delta \boldsymbol{u}_{n+1}^{i-1} \tag{1.36}$$

WARP3D employs a *full* Newton scheme in which the tangent stiffness, $\boldsymbol{K}_T^d$, is updated before the solution of Eq. (1.36) at each iteration. Iterations continue until specified convergence criteria are met or until a specified limit on iterations is reached.

The residual load vector, the dynamic tangent stiffness, and the mass matrix are computed using the element computation algorithms discussed subsequently The solution of the linear simultaneous equations, Eq. (1.36), for the iterative displacement increment is performed by solvers discussed subsequently as well.

Convergence Criteria

Four convergence criteria are provided to support the Newton iterative solution method. They are:

$$1)\ \| \delta \boldsymbol{u}_{n+1}^{i} \| \le \delta_1 \| \delta \boldsymbol{u}_{n+1}^{1} \| \tag{1.37}$$

$$2)\ \| \boldsymbol{R}_{n+1}^{i} \| \le \delta_2 \| \boldsymbol{P}_{n+1} \| \tag{1.38}$$

$$3)\ \max\left(\ |(\delta u_{n+1}^{i})_k|,\ k = 1, N_{eq} \right) \le \delta_3 \| \delta \boldsymbol{u}_{n+1}^{1} \| \tag{1.39}$$

$$4)\ \max\left((R_{n+1}^{i})_k |, k = 1, N_{eq} \right) \le \delta_4 \times \bar{q} \tag{1.40}$$

Tests (2) and (4) include the current reactions for constrained degrees of freedom in the total applied load $\boldsymbol{P}$. $\bar{q}$ denotes an average force (internal, inertial, reactions, etc.) applied to nodes of the model (defined in section on input for convergence parameters). This makes possible the use of these two convergence tests for models loaded only by imposed displacements; otherwise $\boldsymbol{P} = 0$.

At present there are no mechanisms to control loading in the vicinity of limit points or to otherwise improve performance in such situations, e.g., Riks method.

Imposed Displacements and Temperatures

Non-zero imposed displacement increments and imposed temperature increments enter the equation solving process in the following manner. First, Eq. (1.33) is rewritten in the following form

$$\boldsymbol{P}_{n+1}^{d} = \boldsymbol{P}_n + \Delta \boldsymbol{P} + \frac{1}{\beta \Delta t} \boldsymbol{M} \dot{\boldsymbol{u}}_n + \frac{(1 - 2\beta)}{2\beta} \boldsymbol{M} \ddot{\boldsymbol{u}}_n \tag{1.41}$$

where $\Delta \boldsymbol{P}$ denotes the specified increments of nodal forces over $n \rightarrow n+1$ and the increment of work equivalent nodal forces arising from specified element loads (body forces and surface tractions). The incremental load vector to drive the first iteration of the Newton solution for step $n+1$, denoted $\boldsymbol{R}_0$, is then defined by

$$\boldsymbol{R}_0 = \boldsymbol{P}^d_{n+1} - \boldsymbol{I}_0 + \frac{1}{\beta\Delta t}\boldsymbol{M}\Delta\boldsymbol{u} \tag{1.42}$$

where $\Delta\boldsymbol{u}$ contains the specified, non-zero displacement increments and (optionally) the extrapolated displacements from the previous load step.

The internal force vector, $\boldsymbol{I}_0$, for this computation derives from the nodal displacements $\Delta\boldsymbol{u}$ and the imposed temperature increments as follows:

$$\boldsymbol{I}_0 = \sum_{j=1}^{\#elem} \int_{V_e^j} \delta\boldsymbol{\epsilon}^T\boldsymbol{\sigma}_0 dV_e \tag{1.43}$$

where the stress field $\boldsymbol{\sigma}_0$ is obtained through the operations

$$\Delta\boldsymbol{\epsilon}_0 = \mathbf{B}\Delta\boldsymbol{u} - \Delta\boldsymbol{\epsilon}_{th} \tag{1.44}$$

$$\boldsymbol{\sigma}_0 = \boldsymbol{\sigma}_n + \mathcal{C}\left(\boldsymbol{\sigma}_n, \Delta\boldsymbol{\epsilon}_{0,}\ldots\right) . \tag{1.45}$$

In the above, $\mathbf{B}$ denotes the incremental strain-displacement operator with $\Delta\boldsymbol{\epsilon}_{th}$ the specified thermal strain increment for the step. Here, $\mathcal{C}$ defines the constitutive operator which updates the stresses for a specified strain increment. The operators $\mathbf{B}$ and $\mathcal{C}$ reflect the specific element formulation, finite strains-rotations if required and the appropriate material consitutive model.

The incremental load vector defined in this manner ($\boldsymbol{R}_0$) is then used in Eq. (1.30) to compute the first estimate fo the displacement increment which advances the solution from $n \rightarrow n+1$,

$$\boldsymbol{K}^d_T\delta\boldsymbol{u}^1_{n+1} = \boldsymbol{R}_0 . \tag{1.46}$$

1.7 Linear Equation Solvers

Solution of the linear set of equations described by Eq. (1.36) is accomplished either by sparse direct solvers or by a linear preconditioned conjugate gradient (LPCG) solver. Two types of direct solvers are available: (1) an older in-memory version of Choleski factorization and back substitution based on profile storage of the upper-triangular portion of the dynamic tangent stiffness matrix for the structure, (2) a family of platform specific, sparse solvers based on multi–minimum degree re–ordering of the equilbrium equations. The sparse solvers use much less memory and CPU time compared to profile Choleski solver for larger models and approach the LPCG solver in efficiency on workstations which have slower memory systems. Use of the direct solvers is recommended for 3–D models which are essentially 2-D, e.g., a one or two-layer 3-D model to represent a plane–strain, plane–stress or shell structure.

The LPCG solver forms the basis for efficient solution of very large 3–D models in WARP3D. The solution using a LPCG algorithm involves the iterative improvement of an approximate nodal displacement vector, $\boldsymbol{u}$, through a sequence of matrix operations which vectorize naturally and which are amenable to parallel processing. The computational procedure is implemented in an element–by–element architecture which eliminates the need to assemble and store the dynamic tangent stiffness matrix for the structure. Consequently, the memory requirements for solution are dramatically reduced. Moreover, the CPU time required for the LPCG iterative solution frequently is one–half or less of the CPU time required for the direct solver. Both memory and CPU time reductions provided by the LPCG solver are of paramount importance on supercomputers (sometimes making the difference between practical and impractical storage/runtimes). Use of the LPCG solver on Unix workstations often enables the solution of relatively large problems in real memory with CPU time ≈ wallclock time. For such problems, the direct solver incurs a severe wallclock time overhead for virtual memory paging to swap the assembled stiffness matrix (often > 200–400 MB) to/from disk storage. Models with 7,500 8–node elements run in–memory on a 64 MB workstation using the LPCG solver with the diagonal preconditioner.

1.7.1 Linear Preconditioned Conjugate Gradient

As stated above, the linear preconditioned conjugate gradient algorithm can be used to solve the linear system of equations in a nonlinear iteration of Newton's method. In the following development, the linear system of equations is denoted by $\boldsymbol{Ax}=\boldsymbol{b}$, where $\boldsymbol{A}$ is understood to be the current estimate of the dynamic tangent stiffness and $\boldsymbol{b}$ the nonlinear residual. The matrix $\boldsymbol{B}$ represents the *preconditioning* matrix. The linear preconditioned conjugate gradient algorithm proceeds as follows:

1) *Initialize:*

```
$\boldsymbol{x}_0 = \boldsymbol{0}$

for $j = 1, N_{eq}$; if $j$ is a constrained dof,
        $r_j = 0$
else
        $r_j = b_j$
end if

$k = 1$
```

note: non–zero displacement constraints are placed in the total increment of displacement vector at the beginning of each step and corresponding residual entries are set to zero.

2) *Compute in order:*

$$\boldsymbol{z}_{k-1} = \boldsymbol{B}^{-1}\boldsymbol{r}_{k-1} \tag{1.47}$$

$$\beta_k = \frac{\boldsymbol{z}^T_{k-1}\boldsymbol{r}_{k-1}}{\boldsymbol{z}^T_{k-2}\boldsymbol{r}_{k-2}} \qquad (\beta_1 = 0) \tag{1.48}$$

$$\boldsymbol{p}_k = \boldsymbol{z}_{k-1} + \beta_k\boldsymbol{p}_{k-1} \quad (\boldsymbol{p}_0 = \boldsymbol{0}) \tag{1.49}$$

$$a_k = \frac{\boldsymbol{z}^T_{k-1}\boldsymbol{r}_{k-1}}{\boldsymbol{p}^T_k\boldsymbol{A}\boldsymbol{p}_k} \quad \text{(step length computation)} \tag{1.50}$$

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + a_k\boldsymbol{p}_k \tag{1.51}$$

$$\boldsymbol{r}_k = \boldsymbol{r}_{k-1} - a_k\boldsymbol{A}\boldsymbol{p}_k \tag{1.52}$$

3) *Check convergence:*

```
if $\|\boldsymbol{r}_k\| \le tol\,\|\boldsymbol{r}_0\|$ then
        LPCG solution converged
else
        if $k$ > iteration limit then
                LPCG solution did not converge
        else
                $k = k+1$
                return to (2)
        end if
end if
```

The costly operations in the above algorithm are represented by the preconditioning step, Eq. (1.47), and the matrix–vector product required by Eqs. (1.50) and (1.52). Performance of the preconditioning step is discussed below. Because the matrix $\boldsymbol{A}$ is never formed on the global level, the matrix–vector product is computed in blocks of similar, nonconflicting elements.

The key to the performance of the linear preconditioned conjugate algorithm is the choice of a preconditioning matrix, represented by the matrix $\boldsymbol{B}$ in Eq. (1.47). Defining the "A" norm as

$$\|\boldsymbol{x}\|_A = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \tag{1.53}$$

the rate of convergence in this norm is given by

$$\|\boldsymbol{x} - \boldsymbol{x}_k\|_A = \|\boldsymbol{x} - \boldsymbol{x}_0\|_A \left[\frac{1 - \sqrt{\kappa}}{1 + \sqrt{\kappa}}\right]^{2k} \tag{1.54}$$

where κ is the condition number

$$\kappa = \lambda_{\max}(\boldsymbol{B}^{-1}\boldsymbol{A})/\lambda_{\min}(\boldsymbol{B}^{-1}\boldsymbol{A}) \tag{1.55}$$

and $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum eigenvalues of $\boldsymbol{B}^{-1}\boldsymbol{A}$ (see Concus, et al. [15], Golub [26], Hughes et al. [43]). The preconditioning matrix should resemble the inverse of $\boldsymbol{A}$ so that $\varkappa$ approaches unity and convergence is enhanced, and it should also be a relatively trivial matter to invert the preconditioning matrix. Two preconditioners are available in WARP3D as outlined below.

Diagonal Preconditioner

The first and simplest preconditioning matrix is the diagonal preconditioner

$$\boldsymbol{B} = diag(\boldsymbol{A}) \tag{1.56}$$

which represents diagonal scaling or an acceleration of the Jacobi iterative method. Instead of using the current estimate of the dynamic tangent stiffness $\boldsymbol{A}$, it is also possible to employ the diagonal elements of the current estimate of the tangent stiffness or the mass matrix as the preconditioner, although no real advantage results since $\boldsymbol{A}$ must be available in some form (in WARP3D, upper triangular storage by element) to calculate the step length and the linear residual in Eqs. (1.50) and (1.52). The evaluation of Eq. (1.47) using the diagonal preconditioner is accomplished on the global level, as it consists of a simple vector multiply.

Hughes–Winget Preconditioner

The second preconditioner available is the Crout element–by–element preconditioner described by Hughes, et. al. [42], [43]. This preconditioner is an attractive one because it conforms to the element storage of data inherent in the finite element method and it provides an easily vectorizable algorithm for block and parallel processing. The preconditioner consists of the product decomposition

$$\boldsymbol{B} = \boldsymbol{W}^{1/2} \times \prod_{e=1}^{N_{el}} \boldsymbol{L}_p^e \times \prod_{e=1}^{N_{el}} \boldsymbol{D}_p^e \times \prod_{e=N_{el}}^{1} \boldsymbol{U}_p^e \times \boldsymbol{W}^{1/2} \tag{1.57}$$

where

$$\boldsymbol{W} = diag(\boldsymbol{A}) \tag{1.58}$$

and $\boldsymbol{L}_p^e$, $\boldsymbol{D}_p^e$, $\boldsymbol{U}_p^e$ are the lower triangular, diagonal, and upper triangular matrices of the Crout factorization of the corresponding Winget regularized element matrix defined by

$$\overline{\boldsymbol{A}}^e = \boldsymbol{I} + \boldsymbol{W}^{-1/2}(\boldsymbol{A}^e - \boldsymbol{W}^e)\boldsymbol{W}^{-1/2}; \quad \boldsymbol{W}^e = diag(\boldsymbol{A}^e) \tag{1.59}$$

The reverse element ordering in the upper triangular product of Eq. (1.57) insures that $\boldsymbol{B}$ is symmetric, and the Winget regularization dictates that the regularized element matrix be positive–definite. Since the regularized element matrix is also symmetric, $\boldsymbol{L}_p^e$ is the transpose of $\boldsymbol{U}_p^e$ and need not be computed or require additional storage. The upper triangular and diagonal matrix factors for a given element are computed by Eqs. (1.60)–(1.63), for each matrix column k as k varies from one to the number of element degrees of freedom.

$$U_{ik}^{e^*} = \overline{A}_{ik}^e ; \qquad i = 1,\ k-1 \tag{1.60}$$

$$U_{ik}^{e^*} = \overline{A}_{ik}^e - \sum_{j=1}^{i-1} U_{ji}^e U_{jk}^{e^*} ; \qquad i = 2,\ k-1 \tag{1.61}$$

$$U_{ik}^e = \frac{U_{ik}^{e^*}}{D_{ii}^e} ; \qquad i = 1,\ k-1 \tag{1.62}$$

$$D_{kk}^e = \overline{A}_{kk}^e - \sum_{j=1}^{k-1} U_{jk}^{e^*} U_{jk}^e ; \qquad i = 1,\ k-1 \tag{1.63}$$

The factorization is performed for all elements each time the matrix A is recomputed in the course of the nonlinear iterative solution. In practice, all element matrices are stored and manipulated in a compact upper triangular vector form. Performance of the element regularizations and factorizations is accomplished in blocks of similar, nonconflicting elements using the element computation algorithms.

The steps required to solve Eq. (1.47) given the preconditioning matrix of Eq. (1.57) are listed as follows:

1) *Global diagonal scaling:*

$$\boldsymbol{z}_0^* = \boldsymbol{W}^{-1/2}\boldsymbol{r}_{k-1} \tag{1.64}$$

2) *Element forward reduction:*

$$\boldsymbol{z}_i^* = (\boldsymbol{L}_p^i)^{-1}\boldsymbol{z}_{i-1}^* ; \qquad i = 1,\ N_{el} \tag{1.65}$$

3) *Element diagonal scaling:*

$$\hat{z}_0 = z_{N_{el}}^* \rightarrow \hat{\boldsymbol{z}}_i = (\boldsymbol{D}_p^i)^{-1}\hat{\boldsymbol{z}}_{i-1} ; \qquad i = 1,\ N_{el} \tag{1.66}$$

4) *Element back substitution:*

$$\tilde{\boldsymbol{z}}_{N_{el}+1} = \hat{\boldsymbol{z}}_{N_{el}} \rightarrow \tilde{\boldsymbol{z}}_i = (\boldsymbol{U}_p^i)^{-1}\tilde{\boldsymbol{z}}_{i+1} ; \ \ i = N_{el},\ 1 \tag{1.67}$$

5) *Global diagonal scaling:*

$$\boldsymbol{z}_{k-1} = \boldsymbol{W}^{-1/2}\tilde{\boldsymbol{z}}_1 \tag{1.68}$$

The element operations implied by Eqs. (1.65)–(1.67) are again executed in blocks of similar, nonconflicting elements. Element diagonal scaling is achieved at the global level through the equation

$$\hat{\boldsymbol{z}}_{N_{el}} = \hat{\boldsymbol{W}}^{-1}\boldsymbol{z}_{N_{el}}^* \tag{1.69}$$

where

$$\hat{\boldsymbol{W}}^{-1} = \prod_{e=1}^{N_{el}} \boldsymbol{D}_p^{e\,-1} \tag{1.70}$$

is premultiplied during the regularization and factorization procedure.

1.7.2 Direct Solvers

The sparse direct solvers dynamically allocate sufficient real memory to store only the required terms of the dynamic tangent stiffness matrix for the structure. Virtual memory (paging) facilities provided by the operating system permit solutions even when the memory required for data storage exceeds the available physical memory. The "wallclock" time increases dramatically for solutions that incur significant paging overhead.

The (old) profile-based direct solver uses a Choleski procedure to perform forward reduction of the load vector simultaneously with factorization of the dynamic tangent stiffness (see Zienkiewicz and Taylor [90]). Inner loops of the factorization, forward pass and the back pass steps are performed with calls to assembly language routines provided by the computer manufacturer to obtain maximum performance on each platform.

The "generic" sparse direct solver in WARP3D derives from the VSS solver system developed by the Computational Structural Mechanics Branch of the NASA Langley Research Center. The minimum degree re-ordering scheme dramatically reduces the real memory and CPU time required for solution of the equations. The solution procedure has several steps including: (1) assembly of only non-zero terms in the profile to exploit sparsity, (2) minimum degree re-ordering of the equations, (3) symbolic factorization to determine fill-in during decomposition, (4) numeric factorization and loadpass. Typical 3-D solids models analyzed with WARP3D often have only 2-5% non-zero terms in the profile with fill-in after re-ordering of 10-20% of the profile. Memory requirements with the sparse solver are thus often only one–fifth or less of those for the profile solver. Numeric factorization times are reduced, with the reductions becoming more dramatic as the number of equations increases. Further savings are realized in nonlinear solutions which maintain the same matrix sparsity during Newton iterations; the solution processors bypass the re-ordering and symbolic factorization steps.

Much research continues into improving the performance of sparse equation solvers. Best performance is obtained only by matching features of the specific computer architecture (memory hierarchy, cache size, vector lengths, etc.) to data structures and specific algorithms in the solver code (especially the numeric factorization routine). Computer vendors now supply optimized sparse solver libraries for their hardware. On the Cray C–90, T–90 computers, we use the Boeing Computer Services (BCSLIB) sparse solver. On SGI computers, we use the solver developed by Ed Rothberg's group at SGI which provides superb serial and parallel performance. For serial execution, the SGI solver can run very efficiently in out–of–core mode (helpful for executions on workstations). On HP workstations, the sparse solver provided in their MLIB product can be invoked within WARP3D.

The user can specify which solver to use during input of initial model. The choice of solver can be changed at any time during the solution. The (old) profile–based solver and the "generic" sparse solver are available on all platforms.

When the direct solvers are entered for the first time during program execution, various statistics about the solution are printed, including the actual number of equations to be solved (constrained dof do not appear in the assembled equations), the number of terms in the profile, the number of non-zero terms in the profile, etc.

As for all "node" based direct solvers, the ordering of structure nodes plays the critical role in determining the computational effort required for solution. Traditional node re-numbering procedures, e.g., reverse Cuthill–McGee and Gibbs-Poole-Stockmeyer should be used to re-order the nodes before creating the WARP3D input file. Experience indicates that such re-numberings also improve the performance of the sparse solver and are thus highly recommended.

Pre–processing programs should be used to re–number model nodes to minimize the profile before using the direct solvers.

1.7.3 Solver Summary

WARP3D offers two basic equation solving strategies: (1) linear, pre-conditioned conjugate gradients, and (2) direct solution via factorization. The user makes an initial selection during model definition but can change solvers at any time without incurring a memory penalty.

For the LPCG solver, there are two preconditioners available: (1) diagonal and (2) Hughes-Winget (HW). The diagonal preconditioner is very fast but generally requires more iterations for convergence. The HW preconditioner must be used on models with poor conditioning (e.g., shells modeled with solid elements).

The direct solvers may require much greater amounts of memory for solution of large models. However, the sparse solver technologies have reduced numeric factorization times and memory requirements tremendously from those of older "profiled" solvers. The reduced factorization times and large memories available today once again make direct solvers competitive for large 3-D models. WARP3D offers an older style "profile" solver and a "generic" sparse solver on all platforms. On certain platforms (Cray, SGI, HP), highly optimized sparse solvers are available for both serial and parallel execution. These solvers can attain as much as a 5-8 $\times$ reduction in solution time compared to the generic sparse solver and 20-30 $\times$ reduction compared to the profile based solver.

1.8 Element Formulations

Development of the finite element formulation for three dimensional isoparametric elements begins with interpolation of the element displacements and coordinates. The description that follows refers to the kinematic nonlinear formulation; simplifications to obtain the conventional linear kinematic formulation are straightforward.

All quantities are described relative to a fixed set of Cartesian axes, $\tilde{\boldsymbol{X}}$, defined at $t = 0$. Let $\boldsymbol{X}$ denote the Cartesian position vectors for material points at $t = 0$ (see Fig. 1.2). Position vectors for material points at time t are denoted $\boldsymbol{x}$. The displacements of material points are thus given by $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{X}$ and the material point velocities by $\dot{\boldsymbol{u}}$ (later we also use $\boldsymbol{v}$ to denote material point velocities). Components of $\boldsymbol{X}$, $\boldsymbol{x}$, $\boldsymbol{u}$ and $\dot{\boldsymbol{u}}$ are all defined using the basis vectors for axes $\tilde{\boldsymbol{X}}$. In static analyses we associate the time–like parameter t with a specified level of loading imposed on the model. Stress and deformation rates are thus defined with respect to the applied loading rather than with time.

1.8.1 Interpolating Functions

The velocity of a material point at t is interpolated from the nodal velocities using a conventional element interpolating ("shape") function matrix in the form

$$\dot{\boldsymbol{d}} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix}_{3\times 1} = \hat{\boldsymbol{N}} \begin{bmatrix} (\dot{\boldsymbol{u}}_e^x)_{n\times 1} \\ (\dot{\boldsymbol{u}}_e^y)_{n\times 1} \\ (\dot{\boldsymbol{u}}_e^z)_{n\times 1} \end{bmatrix}_{3n\times 1} = \hat{\boldsymbol{N}}\dot{\boldsymbol{u}}_e \tag{1.71}$$

where n here denotes the number of element nodes. Note the non–conventional ordering of nodal displacements in $\dot{\boldsymbol{u}}_e$ which facilitates vectorization of numerical computations. The coordinates of a material point in the configuration at time t are interpolated from the nodal coordinates at t using the same shape functions, resulting in the similar equation

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{3\times 1} = \hat{\boldsymbol{N}} \begin{bmatrix} (\boldsymbol{c}_e^x)_{n\times 1} \\ (\boldsymbol{c}_e^y)_{n\times 1} \\ (\boldsymbol{c}_e^z)_{n\times 1} \end{bmatrix}_{3n\times 1} = \hat{\boldsymbol{N}}\boldsymbol{c}_e \tag{1.72}$$

where $\boldsymbol{c}_e = \boldsymbol{c}_{e,t=0} + \boldsymbol{u}_e$. The element shape functions, one for each element node, are functions of the parametric variables ξ, η, and ζ. For convenience, they are grouped in the row vector

$$\boldsymbol{N} = \langle N_1 N_2 \dots N_n \rangle_{1\times n} \tag{1.73}$$

The shape function derivatives with respect to the parametric variables are represented by the row vectors

$$\boldsymbol{N},_\xi = \langle N_{1,\xi} N_{2,\xi} \dots N_{n,\xi} \rangle_{1\times n} \tag{1.74}$$

$$\boldsymbol{N},_\eta = \langle N_{1,\eta} N_{2,\eta} \dots N_{n,\eta} \rangle_{1\times n} \tag{1.75}$$

$$\boldsymbol{N},_\zeta = \langle N_{1,\zeta} N_{2,\zeta} \dots N_{n,\zeta} \rangle_{1\times n} \tag{1.76}$$

The element shape functions are collected in the element shape function matrix defined by

$$\hat{\boldsymbol{N}} = \begin{bmatrix} \boldsymbol{N} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{N} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{N} \end{bmatrix}_{3\times 3n} \tag{1.77}$$

1.8.2 Cartesian Derivatives

The Jacobian matrix relating differentials in parametric and Cartesian ($\boldsymbol{x}$) coordinates is given by

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}_{3\times 3} \tag{1.78}$$

with the inverse of the Jacobian matrix denoted by

$$\boldsymbol{\Gamma} = \boldsymbol{J}^{-1} \tag{1.79}$$

The gradients of velocity with respect to the $\boldsymbol{x}$ configuration are contained in the vector defined by

$$\dot{\boldsymbol{\Theta}} = \begin{Bmatrix} \dot{\boldsymbol{d}}_{,x} \\ \dot{\boldsymbol{d}}_{,y} \\ \dot{\boldsymbol{d}}_{,z} \end{Bmatrix} \equiv \begin{Bmatrix} \dot{\boldsymbol{\Theta}}_x \\ \dot{\boldsymbol{\Theta}}_y \\ \dot{\boldsymbol{\Theta}}_z \end{Bmatrix} = \begin{Bmatrix} \dot{u}_{,x} \\ \dot{v}_{,x} \\ \dot{w}_{,x} \\ \dot{u}_{,y} \\ \dot{v}_{,y} \\ \dot{w}_{,y} \\ \dot{u}_{,z} \\ \dot{v}_{,z} \\ \dot{w}_{,z} \end{Bmatrix}_{9\times 1} \tag{1.80}$$

The velocity gradients in parametric space constitute the vector

$$\dot{\boldsymbol{\Phi}} = \begin{Bmatrix} \dot{\boldsymbol{d}}_{,\xi} \\ \dot{\boldsymbol{d}}_{,\eta} \\ \dot{\boldsymbol{d}}_{,\zeta} \end{Bmatrix} = \begin{Bmatrix} \dot{u}_{,\xi} \\ \dot{v}_{,\xi} \\ \dot{w}_{,\xi} \\ \dot{u}_{,\eta} \\ \dot{v}_{,\eta} \\ \dot{w}_{,\eta} \\ \dot{u}_{,\zeta} \\ \dot{v}_{,\zeta} \\ \dot{w}_{,\zeta} \end{Bmatrix}_{9\times 1} \tag{1.81}$$

The two velocity gradient vectors are related by the equation

$$\dot{\boldsymbol{\Theta}} = \hat{\boldsymbol{\Gamma}}\dot{\boldsymbol{\Phi}} \tag{1.82}$$

where

$$\hat{\boldsymbol{\Gamma}} = \begin{bmatrix} \Gamma_{11}\boldsymbol{I}_3 & \Gamma_{12}\boldsymbol{I}_3 & \Gamma_{13}\boldsymbol{I}_3 \\ \Gamma_{21}\boldsymbol{I}_3 & \Gamma_{22}\boldsymbol{I}_3 & \Gamma_{23}\boldsymbol{I}_3 \\ \Gamma_{31}\boldsymbol{I}_3 & \Gamma_{32}\boldsymbol{I}_3 & \Gamma_{33}\boldsymbol{I}_3 \end{bmatrix}_{9\times 9} \tag{1.83}$$

where $\boldsymbol{I}_3$ denotes a 3×3 identity matrix. The velocity gradients in parametric space are expressed in terms of the nodal velocities by

$$\dot{\boldsymbol{\Phi}} = \boldsymbol{G}\dot{\boldsymbol{u}}_e \tag{1.84}$$

where

$$G = \begin{bmatrix} \hat{N}_{,\xi} \\ \hat{N}_{,\eta} \\ \hat{N}_{,\zeta} \end{bmatrix} = \begin{bmatrix} N_{,\xi} & 0 & 0 \\ 0 & N_{,\xi} & 0 \\ 0 & 0 & N_{,\xi} \\ N_{,\eta} & 0 & 0 \\ 0 & N_{,\eta} & 0 \\ 0 & 0 & N_{,\eta} \\ N_{,\zeta} & 0 & 0 \\ 0 & N_{,\zeta} & 0 \\ 0 & 0 & N_{,\zeta} \end{bmatrix}_{9 \times 3n} \tag{1.85}$$

1.8.3 *B* Matrix

At time t, we impose a compatible virtual displacement field on the the current (deformed) configuration. The corresponding virtual deformation is defined using the 6×1 vector form of the symmetric deformation tensor

$$\delta\epsilon = \begin{Bmatrix} \delta\epsilon_x \\ \delta\epsilon_y \\ \delta\epsilon_z \\ \delta\gamma_{xy} \\ \delta\gamma_{yz} \\ \delta\gamma_{xz} \end{Bmatrix} = \begin{Bmatrix} \delta u_{,x} \\ \delta v_{,y} \\ \delta w_{,z} \\ \delta u_{,y} + \delta v_{,x} \\ \delta v_{,z} + \delta w_{,y} \\ \delta w_{,x} + \delta u_{,z} \end{Bmatrix}_{6 \times 1} \tag{1.86}$$

where it is understood that, for example, that $\delta u_{,x} = \partial(\delta u)/\partial x$. In terms of the virtual nodal displacements, we write in conventional form

$$\delta\epsilon_{(6 \times 1)} = B_{(6 \times 3n)} \delta u_{e(3n \times 1)} \tag{1.87}$$

where the strain–displacement B matrix is constructed as follows. Define the Boolean matrix $\tilde{B}$ by

$$\tilde{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}_{6 \times 9} \tag{1.88}$$

which permits expression of the strain–displacement matrix B by

$$B_{(6 \times 3n)} = \tilde{B}_{(6 \times 9)} \hat{\Gamma}_{(9 \times 9)} G_{(9 \times 3n)} \tag{1.89}$$

The vectors and matrices presented in this section form the building blocks of the key element quantities determined below.

1.8.4 Internal Force Vector

The element internal force vector is derived from the internal virtual work term in Eq. (1.4) given by

$$\sum_{j=1}^{\#elem} \int_{V_e^j} \delta\epsilon^T \sigma dV_e + \sum_{j=1}^{\#elem} \int_{V_e^j} \delta d^T \rho \ddot{d} dV_e - \sum_{i=1}^{m} \delta d_i^T p_i = 0 \tag{1.90}$$

Using the virtual deformation expressed in terms of the element B matrix, we have (for a single element)

$$\int_{V_e} \delta\boldsymbol{\epsilon}^T \sigma dV_e = \delta\boldsymbol{u}_e{}^T \int_{V_e} \boldsymbol{B}^T \sigma dV_e \tag{1.91}$$

where again V_e denotes the element configuration at t, σ denotes the symmetric Cauchy stresses expressed in 6×1 vector format at t, and the $\boldsymbol{B}$ matrix is evaluated using coordinates of element nodes at t, $\boldsymbol{c}_e = \boldsymbol{c}_{e,t=0} + \boldsymbol{u}_e$. Using Eq. (1.6) we see that the element internal force vector is given by

$$\boldsymbol{I}_{e(3n\times 1)} = \int_{V_e} \boldsymbol{B}^T \sigma dV_e = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \boldsymbol{B}^T \sigma \,|\boldsymbol{J}|\, d\xi d\eta d\zeta \tag{1.92}$$

The global internal force vector is obtained through global assembly of the element internal force vectors.

1.8.5 Strain Increment for Stress Updating

Newton's method advances the global solution from time step n to n+1 through a series of iterative improvements to the solution at n+1. Let i denote the current Newton iteration for the solution at n+1, $\boldsymbol{u}^{(i)}_{n+1}$ the i^{th} estimate for the element nodal displacements at n+1 and $\boldsymbol{u}_n$ the converged solution for element nodal displacements at n. Using the mid–increment configuration, the i^{th} estimate for the (mechanical) strain increment over the step is given by

$$\Delta\boldsymbol{\epsilon}^{(i)} = \boldsymbol{B}_{n+\frac{1}{2}}\left(\boldsymbol{u}^{(i)}_{n+1} - \boldsymbol{u}_n\right) - \Delta\boldsymbol{\epsilon}_{th} \tag{1.93}$$

where the $\boldsymbol{B}$ matrix is evaluated using nodal coordinates $\boldsymbol{c}_e = \boldsymbol{x}_{n+1/2}$. The specified thermal strain increments over $n \to n+1$ are indicated by $\Delta\boldsymbol{\epsilon}_{th}$. The strain increment $\Delta\boldsymbol{\epsilon}^{(i)}$ is passed to the stress updating (constitutive) models, after rotation effects are neutralized as described in Section 1.9.4, to obtain the new estimate for the Cauchy stresses at n+1, $\sigma^{(i)}_{n+1}$.

Key and Krieg [49] and Nagtegaal and Veldpaus [66] have demonstrated that Eq. (1.93) defines a constant rate of logarithmic strain over the step. In a one–dimensional setting, integration of the strain rate to define a total strain measure using the mid–point rule above remains surprisingly accurate for very large increments. In multi–dimensional problems, the interpretation of logarithmic strain holds if the principal directions of strain rotate to match the rigid body motion. This rarely happens and thus accumulated increments of converged $\Delta\boldsymbol{\epsilon}$ values do not represent a valid total strain measure.

1.8.6 Tangent Stiffness Matrix

The element tangent stiffness matrix is defined in terms of the rate of the element internal force vector by

$$\dot{\boldsymbol{I}}_e = \left[\boldsymbol{K}_T\right]_e \dot{\boldsymbol{u}}_e \tag{1.94}$$

From Eq. (1.92) the rate of the element internal force vector is

$$\dot{\boldsymbol{I}}_e = \int_{V_e} \dot{\boldsymbol{B}}^T \sigma \, dV_e + \int_{V_e} \boldsymbol{B}^T \dot{\sigma} \, dV_e \tag{1.95}$$

The first term in Eq. (1.95) can be manipulated into the form (see Zienkiewicz and Taylor [91])

$$\int_{V_e} \dot{\boldsymbol{B}}^T \sigma \, dV_e = \left[\int_{V_e} \boldsymbol{G}^T \hat{\Gamma}^T \boldsymbol{M}_\sigma \hat{\Gamma} \boldsymbol{G} \, dV_e \right] \dot{\boldsymbol{u}}_e \tag{1.96}$$

where

$$\boldsymbol{M}_\sigma = \begin{bmatrix} \sigma_1 \boldsymbol{I}_3 & \sigma_4 \boldsymbol{I}_3 & \sigma_6 \boldsymbol{I}_3 \\ \sigma_4 \boldsymbol{I}_3 & \sigma_2 \boldsymbol{I}_3 & \sigma_5 \boldsymbol{I}_3 \\ \sigma_6 \boldsymbol{I}_3 & \sigma_5 \boldsymbol{I}_3 & \sigma_3 \boldsymbol{I}_3 \end{bmatrix}_{9 \times 9} \tag{1.97}$$

Eq. (1.96) defines the so–called "initial–stress" or geometric stiffness matrix

$$\left[\boldsymbol{K}_T^g \right]_e = \int_{V_e} \boldsymbol{G}^T \hat{\Gamma}^T \boldsymbol{M}_\sigma \hat{\Gamma} \boldsymbol{G} \, dV_e \tag{1.98}$$

The second term in Eq. (1.95) resolves to

$$\int_{V_e} \boldsymbol{B}^T \dot{\sigma} \, dV_e = \left[\int_{V_e} \boldsymbol{B}^T \boldsymbol{E} \boldsymbol{B} \, dV_e \right] \dot{\boldsymbol{u}} \tag{1.99}$$

where $\boldsymbol{E}$ (6×6) denotes the constitutive matrix relating the (spatial) rate of the deformation to the spatial rate of Cauchy stress, as in

$$\dot{\sigma} = \boldsymbol{E} \dot{\boldsymbol{\epsilon}} = \boldsymbol{E} \boldsymbol{B} \dot{\boldsymbol{u}}_e \tag{1.100}$$

Since $\dot{\sigma}$ does not vanish under motion corresponding to a rigid rotation (see Johnson and Bammann [47], Rubinstein and Atluri [78]), a rotation neutralized stress rate must be employed in development of the constitutive matrix, $\boldsymbol{E}$. In WARP3D, the Green–Naghdi [28] stress rate is used to formulate $\boldsymbol{E}$ (see Section 1.9.4 for the stress updating strategy).

Upon combining Eqs. (1.98) and (1.99), the element tangent stiffness matrix may be written as

$$\left[\boldsymbol{K}_T \right]_e = \int_{V_e} \left[\boldsymbol{G}^T \hat{\Gamma}^T \boldsymbol{M}_\sigma \hat{\Gamma} \boldsymbol{G} + \boldsymbol{B}^T \boldsymbol{E} \boldsymbol{B} \right] dV_e \tag{1.101}$$

$$= \int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} \left[\boldsymbol{G}^T \hat{\Gamma}^T \boldsymbol{M}_\sigma \hat{\Gamma} \boldsymbol{G} + \boldsymbol{B}^T \boldsymbol{E} \boldsymbol{B} \right] |\boldsymbol{J}| \, d\xi d\eta d\zeta \tag{1.102}$$

When required for the direct solver, the tangent stiffness matrix for the structure (in global coordinates) is obtained through the usual assembly of element matrices.

All deformation dependent quantities appearing in Eq. (1.102) refer to values for the i^{th} iteration of step n+1, i.e, $\boldsymbol{B}$ is evaluated using the nodal coordinates $\boldsymbol{x}_{n+1}^{(i)}$, the Cauchy stresses appearing in $\boldsymbol{M}_\sigma$ are $\sigma_{n+1}^{(i)}$ and $\boldsymbol{E}$ is the tangent modulus which advances the spatial rate of Cauchy stress from n to n+1 (i^{th} iteration) *consistent* with the stress updating procedure for the strain increment $\Delta \boldsymbol{\epsilon}^{(i)}$.

The stiffness formulations employed in WARP3D do not correspond to either of the traditional procedures, Total Lagrangian (T.L.) or Updated Lagrangian (U.L.), (see Bathe [6], Zienkiewicz and Taylor [91]). In T.L., the tangent stiffness is expressed using all deformation quantities relative to the configuration at $t = 0$. In U.L., the converged solution at n provides the reference configuration for all quantities needed in $[K_T]$. Both of these approaches require the inclusion of additional (nonlinear) terms in $\boldsymbol{B}$ and the use of $2nd$ Piola–Kirchoff stresses rather than the Cauchy stress.

The present formulation, with minor differences, follows closely that used in the NIKE codes (Hallquist [30], [31]).

1.8.7 Mass Matrix

The element consistent mass matrix is derived from the inertial virtual work term in Eq. (1.4) given by

$$\int_{V_e} \delta \boldsymbol{d}^T \rho \ddot{\boldsymbol{d}} dV_e \tag{1.103}$$

where integration is over the (current) deformed volume and ρ denotes the mass density per unit of deformed volume. Upon substitution of Eq. (1.71) and its second time derivative, noting that the shape functions are independent of time, Eq. (1.103) becomes

$$\int_{V_e} \delta \boldsymbol{d}^T \rho \ddot{\boldsymbol{d}} dV_e = \delta \boldsymbol{u}_e{}^T \left[\int_{V_e} \rho \hat{\boldsymbol{N}}^T \hat{\boldsymbol{N}} dV_e \right] \ddot{\boldsymbol{u}}_e \tag{1.104}$$

A comparison with Eq. (1.6) reveals that the element consistent mass matrix has the form

$$\boldsymbol{M}_e = \int_{V_e} \rho \hat{\boldsymbol{N}}^T \hat{\boldsymbol{N}} dV_e = \int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} \rho \hat{\boldsymbol{N}}^T \hat{\boldsymbol{N}} |\boldsymbol{J}| d\xi d\eta d\zeta \tag{1.105}$$

where $|\boldsymbol{J}|$ is evaluated using nodal coordinates at t. Considering the block diagonal structure of Eq. (1.77), the element consistent mass matrix is also block diagonal, and it is only necessary to compute the block diagonal mass matrix corresponding to one of the three continuum degrees of freedom and to assign this matrix to the other two nodal freedoms.

The mass density ϱ appearing in Eq. (1.105) corresponds to the current configuration, as the inertial body force acts there. It may be expressed in terms of the mass density in the undeformed ($t = 0$) configuration by

$$\rho_0 = \rho |\boldsymbol{F}| \tag{1.106}$$

where $|\boldsymbol{F}|$ denotes the determinant of the deformation gradient, $\boldsymbol{F} = \partial \boldsymbol{x} / \partial \boldsymbol{X}$. Using the relation $dV_e = |\boldsymbol{F}| dV_0$, and Eq. (1.106), the element consistent mass matrix may be expressed using quantities referenced to the $t = 0$ configuration

$$\boldsymbol{M}_e = \int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} \rho_0 \hat{\boldsymbol{N}}^T \hat{\boldsymbol{N}} |\boldsymbol{J}_0| d\xi d\eta d\zeta \tag{1.107}$$

where $|\boldsymbol{J}_0|$ is the determinant of the coordinate Jacobian at $t = 0$. The element consistent mass matrix defined by Eq. (1.107) is independent of time; consequently, the element tangent and secant consistent mass matrices are equal.

It is also possible to define a diagonal element lumped mass matrix. This is accomplished in the following manner (Hinton, et al. [37]):

1) Compute the diagonal terms of the block diagonal consistent mass matrix corresponding to one of the continuum degrees of freedom.
2) Accumulate the mass of these diagonal terms. Scale the diagonal terms by the ratio of the total element mass related to the continuum degree of freedom to the accumulated mass so that the total mass of the diagonal terms is correct. Assign the diagonal terms to the other two continuum degrees of freedom. This is the element lumped mass matrix.

Once again, either the global consistent or lumped mass matrix is found through assembly of the element matrices.

1.9 Finite Strain Plasticity

The theoretical basis and numerical implementation of a constitutive architecture suitable for finite strains and rotations are described in this section. The constitutive equations governing finite deformation are formulated using strains-stresses and their rates defined on an *unrotated* frame of reference. Unlike models based on the classical Jaumann [46] (or corotational) stress rate, the present model predicts physically acceptable responses for homogeneous deformations of exceedingly large magnitude. The associated numerical algorithms accommodate the large strain increments which may arise routinely in the implicit solution of the global equilibrium equations employed in WARP3D. The resulting computational framework divorces the finite rotation effects on strain-stress rates from integration of the rates to update the material response over a load (time) step. Consequently, all of the numerical refinements developed previously for small-strain plasticity (radial return, kinematic hardening, consistent tangent operators, dilatant plasticity models for continuum descriptions of void growth) are utilized without modification.

Two fundamental assumptions (and points of criticism, see Simo and Hughes [84]) underlie the present implementation of this framework in WARP3D: (1) additive decomposition of elastic and plastic strain rates expressed on the current configuration remains a valid description of the deformation, and (2) material elasticity maybe adequately represented by an isotropic, hypoelastic model. These assumptions require that plastic strains (and rates) greatly exceed elastic strains (and rates). Such conditions are easily realized in the study of ductile fracture in metals which possess large E/σ_0 ratios. For other materials, such as polymers, the *ad hoc* treatment of elasticity adopted here becomes unsuitable— at best. A multiplicative decomposition of the deformation gradient into elastic and plastic components, when coupled with a proper hyperelastic treatment of material elasticity, is clearly more appropriate (Moran, Ortiz and Shih [62], Simo and Ortiz [83]). Nevertheless, the essential features of the present finite-strain plasticity formulation provide the core technology adopted in large-scale finite element codes, including NIKE ([30] [31]), DYNA ([27]), PRONTO ([85] [86]), ABAQUS-Standard [35] and ABAQUS-Explicit ([36]).

The following sections describe the basis for the constitutive framework and the detailed, step-by-step implementation in WARP3D. Once the kinematic transformations have eliminated rotation effects on rates of tensorial quantities, the stress updating procedures for each constitutive model are those for the conventional small-strain formulation. Details of the usual small-strain computations are described in Chapter 5 for each of the material models currently available.

The reader interested in an extensive description, the numerical implementation details and the criticism of this finite-strain plasticity framework is referred to the monograph of Simo and Hughes [84], specifically Chapters 6 and 7.

1.9.1 Kinematics, Strain-Stress Measures

Development of the finite strain plasticity model begins with consideration of the deformation gradient

$$\boldsymbol{F} = \partial \boldsymbol{x} / \partial \boldsymbol{X}, \qquad \det(\boldsymbol{F}) = J > 0 \tag{1.108}$$

where $\boldsymbol{X}$ denotes the Cartesian position vectors for material points defined on the configuration at t=0. Position vectors for material points at time t are denoted $\boldsymbol{x}$ (configuration $\boldsymbol{B}$ in Fig. 1.4, after Flanagan and Taylor [24]). The displacements of material points are thus given by $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{X}$. The polar decomposition of $\boldsymbol{F}$ yields

$$\boldsymbol{F} = \boldsymbol{V}\boldsymbol{R} = \boldsymbol{R}\boldsymbol{U} \tag{1.109}$$

where V and U are the left- and right-symmetric, positive definite stretch tensors, respectively; R is a orthogonal rotation tensor. The principal values of V and U are the stretch ratios, λ_i, of the deformation. These two methods for decomposing the motion of a material point are illustrated in Fig. 1.4. In the initial configuration, B_0, we define an orthogonal reference frame at each material point such that the motion relative to these axes is only deformation throughout the loading history. With the RU decomposition, for example, these axes are "spatial" during the motion from B_0 to B_u; they are not altered by deformation of the material. However, during the motion from B_u to B these axes are "material"; they rotate with the body in a *local average sense* at each material point. Strain-stress tensors and their rates referred to these axes are said to be defined in the *unrotated* configuration (Johnson and Bammann [47] and Atluri [4]).

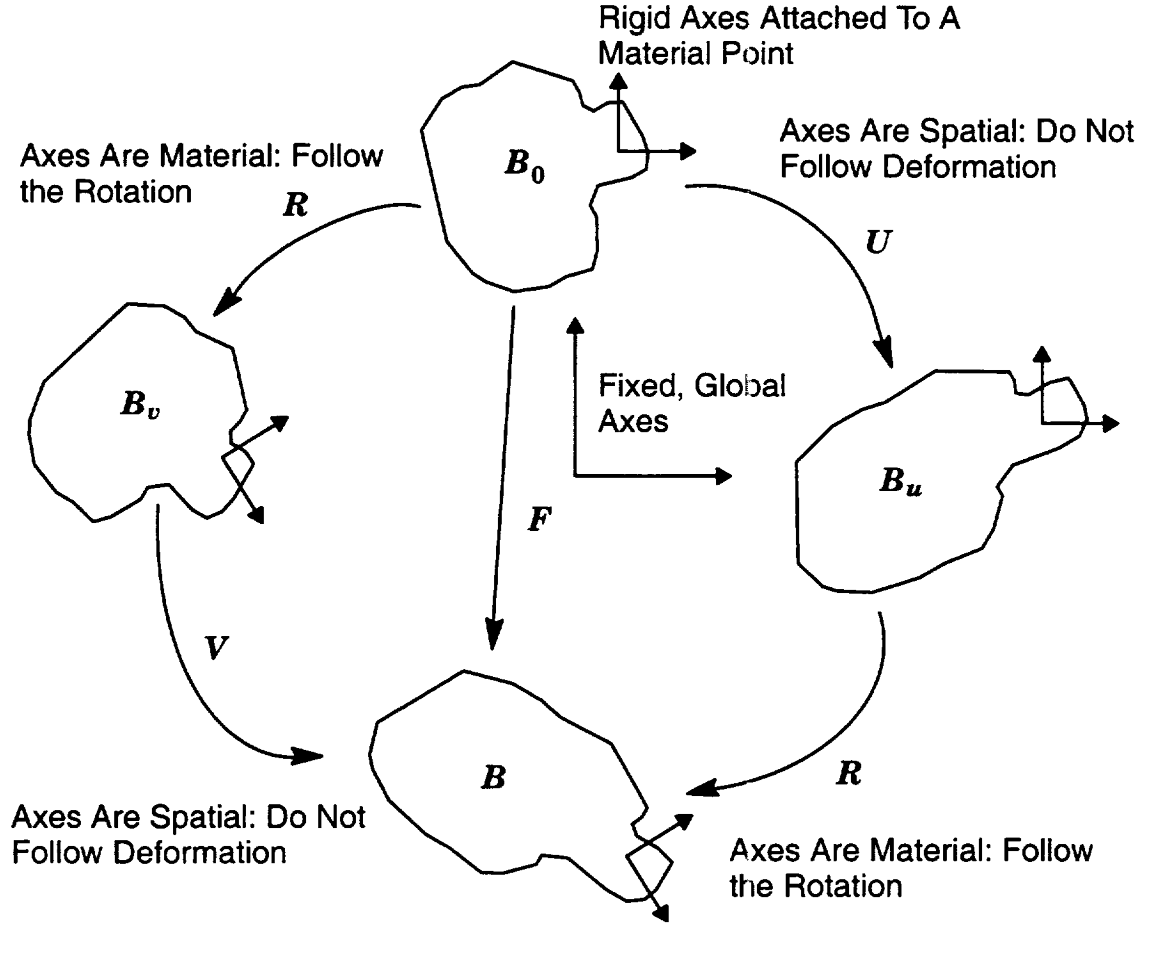

Rigid Axes Attached To A Material Point
Axes Are Material: Follow the Rotation
R
B_0
Axes Are Spatial: Do Not Follow Deformation
U
B_v
Fixed, Global Axes
B_u
F
V
B
R
Axes Are Spatial: Do Not Follow Deformation
Axes Are Material: Follow the Rotation

FIG. 1.4—*Motion of Model Using Polar Decomposition ([24])*

The material derivative of displacement with respect to an applied loading parameter is written as $v = \dot{x}$ (i.e., the material point velocity in dynamic analyses). The spatial gradient of this material derivative with respect to the current configuration is given by

$$L = \frac{\partial v}{\partial x} = \frac{\partial v}{\partial X}\frac{\partial X}{\partial x} = \dot{F}F^{-1} . \tag{1.110}$$

The symmetric part of $\boldsymbol{L}$ is the spatial rate of the deformation tensor, denoted $\boldsymbol{D}$; the skewsymmetric part, denoted $\boldsymbol{W}$, is the spin rate or the vorticity tensor. Thus,

$$\boldsymbol{L} = \boldsymbol{D} + \boldsymbol{W} \tag{1.111}$$

where

$$\boldsymbol{D} = \tfrac{1}{2}(\boldsymbol{L} + \boldsymbol{L}^T); \quad \boldsymbol{W} = \tfrac{1}{2}(\boldsymbol{L} - \boldsymbol{L}^T)\ . \tag{1.112}$$

$\boldsymbol{W}$ represents the rate of rotation of the principal axes of the spatial rate of deformation $\boldsymbol{D}$. When integrated over the loading history, the principal values of $\boldsymbol{D}$ are recognized as the logarithmic (true) strains of infinitesimal fibers oriented in the principal directions if the principal directions do not rotate. It is important to note that $\boldsymbol{D}$ and $\boldsymbol{W}$ have no sense of the deformation history; they are instantaneous rates.

Using the $\boldsymbol{RU}$ decomposition of $\boldsymbol{F}$, the spatial gradient $\boldsymbol{L}$ may be also written in the form

$$\boldsymbol{L} = \dot{\boldsymbol{R}}\boldsymbol{R}^T + \boldsymbol{R}\dot{\boldsymbol{U}}\boldsymbol{U}^{-1}\boldsymbol{R}^T \tag{1.113}$$

in which the following relations are used

$$\dot{\boldsymbol{F}} = \boldsymbol{R}\dot{\boldsymbol{U}} + \dot{\boldsymbol{R}}\boldsymbol{U} \tag{1.114}$$

and

$$\boldsymbol{F}^{-1} = (\boldsymbol{RU})^{-1} = \boldsymbol{U}^{-1}\boldsymbol{R}^{-1} = \boldsymbol{U}^{-1}\boldsymbol{R}^T\ . \tag{1.115}$$

The first term in Eq. (1.113) is the rate of rigid-body rotation at a material point and is denoted $\boldsymbol{\Omega}$ (see Dienes [20]). The spin rate $\boldsymbol{W}$ and $\boldsymbol{\Omega}$ are identical when the principal axes of $\boldsymbol{D}$ coincide with the principal axes of the current stretch $\boldsymbol{V}$ (this observation plays an essential role later in development of a linearized tangent operator). Simple extension and pure rotation satisfy this condition. The symmetric part of the second term in Eq. (1.113) is called the *unrotated* deformation rate tensor (sometimes the *rotation neutralized* deformation rate) and is denoted $\boldsymbol{d}$

$$\boldsymbol{d} = \tfrac{1}{2}(\dot{\boldsymbol{U}}\boldsymbol{U}^{-1} + \boldsymbol{U}^{-1}\dot{\boldsymbol{U}})\ . \tag{1.116}$$

The unrotated rate of deformation defines a material strain rate relative to the orthogonal reference frame indicated on configuration $\boldsymbol{B}$ in Fig. 1.4.

Using the orthogonality property of $\boldsymbol{R}$ that $d(\boldsymbol{R}^T\boldsymbol{R})/dt=0$

$$\boldsymbol{R}^T\dot{\boldsymbol{R}} + \dot{\boldsymbol{R}}^T\boldsymbol{R} \equiv \boldsymbol{0} \tag{1.117}$$

the unrotated deformation rate may be expressed in the simpler form as

$$\boldsymbol{d} = \boldsymbol{R}^T\boldsymbol{D}\boldsymbol{R}\ . \tag{1.118}$$

The principle of virtual displacements (Section 1.4) demonstrates that the spatial rate of deformation, $\boldsymbol{D}$, and the symmetric Cauchy (true) stress, $\boldsymbol{\sigma}$, are work conjugate in the sense that work per unit volume in the current configuration is given by $\sigma_{ij}D_{ij}$. Since components of both $\boldsymbol{D}$ and $\boldsymbol{\sigma}$ are defined relative to the fixed, global axes, the work conjugate stress measure for $\boldsymbol{d}$ on the unrotated configuration is given simply by

$$\boldsymbol{t} = \boldsymbol{R}^T\boldsymbol{\sigma}\boldsymbol{R} \tag{1.119}$$

where $\boldsymbol{t}$ is termed the *unrotated* Cauchy stress, i.e., $\boldsymbol{\sigma}$ is the tensor $\boldsymbol{t}$ expressed on the fixed global axes.

1.9.2 Selection of Strain and Stress Rates

The simplest form of a hypo-elastic constitutive relation is adopted to couple a materially objective stress rate with a work conjugate deformation rate. The Jaumann and Green-Naghdi objective rates of Cauchy stress are

$$\tilde{\sigma} = \dot{\sigma} - \boldsymbol{W}\sigma + \sigma\boldsymbol{W} = \boldsymbol{E} : \boldsymbol{D} \qquad \text{(Jaumann)} \tag{1.120}$$

$$\hat{\sigma} = \dot{\sigma} - \boldsymbol{\Omega}\sigma + \sigma\boldsymbol{\Omega} = \boldsymbol{E} : \boldsymbol{D} \qquad \text{(Green-Naghdi)} \tag{1.121}$$

where the modulus tensor $\boldsymbol{E}$ may depend linearly on the current stress tensor and on history dependent state variables ($\boldsymbol{E} : \boldsymbol{D}$ denotes$E_{ijkl}D_{kl}$). Once the objective stress rate is evaluated using $\boldsymbol{E} : \boldsymbol{D}$, the spatial rate of Cauchy stress, $\dot{\sigma}$, is found by computing $\boldsymbol{W}$ or $\boldsymbol{\Omega}$ and transposing the above equations. In a finite-element setting, these rate expressions are numerically integrated to provide incremental values of the Cauchy stress corresponding to load (time) steps.

When $\boldsymbol{D}$ vanishes both the Jaumann and Green-Naghdi rates predicted by the constitutive models also vanish; however, the two stress rates lead to different spatial rates of Cauchy stress since $\boldsymbol{W}$ and $\boldsymbol{\Omega}$ are generally not identical. Use of the spin tensor $\boldsymbol{W}$ in Eq. (1.120) causes the physically unreasonable (oscillatory) response predicted for the finite shear problem; the Green-Naghdi rate leads to a realistic response. However, the debate ofver physically meaningful stress rates continues.

The Jaumann rate is adopted extensively in finite element codes—the quantity $\boldsymbol{W}$ is readily available as a by-product of computing $\boldsymbol{D}$ whereas computation of $\boldsymbol{\Omega}$ requires polar decompositions of $\boldsymbol{F}$. Hughes and Winget [41] recognized that a constant spin rate $\boldsymbol{W}$ (and rotation rate $\boldsymbol{\Omega}$) limits the acceptable step sizes for implicit codes. They developed a numerical integration scheme for Eq. (1.120) that retains objectivity for rotation increments exceeding 30°. Such refinements, however, do not remove the fundamental cause ($\boldsymbol{W}$) of the oscillatory response in simple shear. Roy, Fossum and Dexter [77] recently implemented a 2-D, implicit finite-element code based on the Green-Naghdi rate as expressed in Eq. (1.121). They employed the Hughes-Winget procedure to integrate $\hat{\sigma}$ using $\boldsymbol{\Omega}$ computed from polar decompositions of $\boldsymbol{F}$ at the start and end of each load increment.

The Green-Naghdi rate may be written alternatively as the rate of *unrotated* Cauchy stress, $\dot{\boldsymbol{t}}$, expressed on the fixed, Cartesian axes

$$\hat{\sigma} = \boldsymbol{R}\dot{\boldsymbol{t}}\boldsymbol{R}^T = \boldsymbol{E} : \boldsymbol{D} \ . \tag{1.122}$$

Transformation of the spatial deformation rate $\boldsymbol{D}$ in this expression to the unrotated deformation rate $\boldsymbol{d}$ yields

$$\dot{\boldsymbol{t}} = \boldsymbol{E} : \left(\boldsymbol{R}^T\boldsymbol{D}\boldsymbol{R}\right) = \boldsymbol{E} : \boldsymbol{d} \ . \tag{1.123}$$

Constitutive computations, equivalent to the Green-Naghdi rate in Eq. (1.121), therefore can be performed using stress-strain rates defined on the unrotated configuration. Updated values of $\boldsymbol{t}$ are rotated via $\boldsymbol{R}$ to obtain the updated Cauchy stress at the end of a load increment. The numerical problems of integrating the rotation rates in Eqs. (1.120) and (1.121) are thus avoided. Moreover, tensorial state variables of the plasticity model, e.g., the back-stress for kinematic hardening, are also defined and maintained on the unrotated configuration and thus never require correction for finite rotation effects. Hallquist [30], [31] was apparently the first to recognize the simplicity derived from this constitutive framework and used it in the NIKE and DYNA codes. Later, this framework was adopted by Flanagan and Taylor for the PRONTO-2D [85] and PRONTO-3D [86] codes, by Biffle and Blandford for the JAC-2D [7] and JAC-3D [8] codes, and most recently in the commerical ABAQUS-EXPLICIT [36] code. The potential disadvantage of this constitutive framework is the numerical effort to compute $\boldsymbol{R}$ from the polar decomposition $\boldsymbol{F}=\boldsymbol{RU}$ at thousands of material points for each of many load steps. For explicit codes in which time steps are necessarily very small to maintain stability, an efficient (forward) integration scheme developed by Flanagan and Taylor [24] may be used to update $\boldsymbol{R}$ without the polar decomposition. The polar decomposition issue is discussed in the section on numerical procedures.

1.9.3 Elastic-Plastic Decomposition

Further developments require kinematic decomposition of the total strain rate $\boldsymbol{d}$ into elastic and plastic components. The multiplicative decomposition of the deformation gradient

$$\boldsymbol{F} = \boldsymbol{F}^e\boldsymbol{F}^p \tag{1.124}$$

appears most compatible with the physical basis of elastic-plastic deformation in crystalline metals (see, for example, Lee [52] and Asaro [3]). $\boldsymbol{F}^p$ represents plastic flow (dislocations) while $\boldsymbol{F}^e$ represents lattice distortion; rigid rotation of the material structure may be considered in either term. Substitution of this decomposition into the spatial rate of the displacement gradient Eq. (1.110) yields

$$\boldsymbol{L} = \dot{\boldsymbol{F}}^e\boldsymbol{F}^{-e} + \boldsymbol{F}^e\dot{\boldsymbol{F}}^p\boldsymbol{F}^{-p}\boldsymbol{F}^{-e} = \boldsymbol{L}^e + \boldsymbol{F}^e\boldsymbol{L}^p\boldsymbol{F}^{-e} \ . \tag{1.125}$$

We now impose the restriction that elastic strains remain vanishingly small compared to the unrecoverable plastic strains; a behavior closely followed by ductile metals having an elastic modulus orders of magnitude greater than the flow stress. Consequently, $\boldsymbol{F}^p$ and $\boldsymbol{F}^e$ are uniquely determined by unloading from a plastic state. This considerably simplifies the above expression and permits separate treatment of material elasticity and plasticity. Using the left polar decomposition and writing the stretch as the product of elastic and plastic parts yields

$$\boldsymbol{F} = \boldsymbol{F}^e\boldsymbol{F}^p = \boldsymbol{V}^e\boldsymbol{V}^p\boldsymbol{R} \tag{1.126}$$

Identifying the elastic deformation as

$$\boldsymbol{F}^e = \boldsymbol{V}^e \tag{1.127}$$

and using the small elastic strain assumption, we have

$$\boldsymbol{F}^e = \boldsymbol{I} + \boldsymbol{e}^e \approx \boldsymbol{I} \ . \tag{1.128}$$

Consequently, the expression for $\boldsymbol{L}$ is approximated by

$$\boldsymbol{L} \approx \boldsymbol{L}^e + \boldsymbol{L}^p \ . \tag{1.129}$$

As in Eq. (1.112), the symmetric part of this approximation for $\boldsymbol{L}$ is taken as $\boldsymbol{D}$ with the result that

$$\boldsymbol{D} \approx \boldsymbol{D}^e + \boldsymbol{D}^p \ . \tag{1.130}$$

Given the restriction of vanishingly small elastic strains, the multiplicative decomposition of the deformation gradient in Eq. (1.124) leads to the familiar additive decomposition of the spatial deformation rate $\boldsymbol{D}$ into elastic and plastic components. The transformation of $\boldsymbol{D}$ to the unrotated configuration using Eq. (1.118) provides the decomposition scheme needed for $\boldsymbol{d}$ as

$$\boldsymbol{d} = \boldsymbol{R}^T(\boldsymbol{D}^e + \boldsymbol{D}^p)\boldsymbol{R} = \boldsymbol{d}^e + \boldsymbol{d}^p \ . \tag{1.131}$$

Once the above transformation of elastic and plastic strain rates onto the unrotated configuration is accomplished, the remaining steps in development of the finite-strain plasticity theory are identical to those for classical small-strain theory.

If the elastic strains are not vanishingly small, the incrementally linear form of this hypo-elastic material model predicts hysteretic dissipation and residual stresses for some closed loading paths, for example, the path defined by finite extension→finite shear→tension unloading→shear unloading (Kojic and Bathe [51]). Uncoupled loading-unloading for extension and shear produces no residual stresses. For finite-strain plasticity of ductile metals having large modulus-to-yield stress ratios this situation is not a serious concern since plastic strains are commonly 50-100 times greater than the elastic strains.

1.9.4 Numerical Procedures

The global solution is advanced from time (load) t_n to t_{n+1} using an incremental-iterative Newton method. Iterations at t_{n+1} to remove unbalanced nodal forces are conducted under fixed external loading and no change in the prescribed displacements for displacement controlled loading. Each such iteration, denoted i, provides a revised estimate for the total displacements at t_{n+1}, denoted $\boldsymbol{u}^{(i)}_{n+1}$. Fully converged displacements at t_n are denoted $\boldsymbol{u}_n$. Following Pinsky, Ortiz and Pister [73] a mid-increment scheme is adopted in which deformation rates are evaluated on the intermediate configuration at $(1-\gamma)\boldsymbol{u}_n + \gamma\boldsymbol{u}^{(i)}_{n+1}$. The choice of $\gamma = 1/2$ represents a specific form of the generalized trapezoidal rule that is unconditionally stable and second-order accurate. Key and Krieg [49] have demonstrated the optimality of the mid-point configuration for integrating the rate of deformation and the resulting correspondence with logarithmic strain (for uniaxial conditions).

The following sections describe the computational processes performed at each material (Gauss) point to: 1) update stresses and to 2) provide a consistent tangent matrix for updating the global stiffness matrix. A brief discussion of the procedure to compute the polar decomposition of the deformation gradient is also provided.

Stress Updating Procedure

The computational steps are:

Step 1. Compute the deformation gradients at $n + 1/2$ and $n + 1$

$$\boldsymbol{F}^{(i)}_{n+1} = \frac{\partial\left(\boldsymbol{X} + \boldsymbol{u}^{(i)}_{n+1}\right)}{\partial \boldsymbol{X}}; \tag{1.132}$$

$$\boldsymbol{F}^{(i)}_{n+1/2} = \frac{\partial\left(\boldsymbol{X} + \boldsymbol{u}^{(i)}_{n+1/2}\right)}{\partial \boldsymbol{X}} \tag{1.133}$$

Step 2. Compute polar decompositions at $n + 1/2$ and $n + 1$

$$\boldsymbol{F}^{(i)}_{n+1} = \boldsymbol{R}^{(i)}_{n+1} \cdot \boldsymbol{U}^{(i)}_{n+1} \tag{1.134}$$

$$\boldsymbol{F}^{(i)}_{n+1/2} = \boldsymbol{R}^{(i)}_{n+1/2} \cdot \boldsymbol{U}^{(i)}_{n+1/2} \tag{1.135}$$

Step 3. Compute the i th estimate for the spatial deformation increment over the step from the $\boldsymbol{B}$ matrix for the element, see Eq. (1.93) and Section 1.8.5.

$$\Delta\boldsymbol{\epsilon}^{(i)} = \boldsymbol{B}^{(i)}_{n+1/2}\left(\boldsymbol{u}^{(i)}_{n+1} - \boldsymbol{u}_n\right) \tag{1.136}$$

$$\Delta \boldsymbol{D}^{(i)} \leftarrow \Delta \boldsymbol{\epsilon}^{(i)} \quad \text{(convert } 6\times 1 \text{ vector to symmetric tensor)} \tag{1.137}$$

This procedure, as compared to the more conventional scheme using Eqs. (1.110) and (1.112), provides a straightforward method to utilize the $\overline{\boldsymbol{B}}$ formulation (to replace $\boldsymbol{B}$) for finite strains thereby reducing volumetric locking in the element.

Step 4. Rotate the increment of spatial deformation to the unrotated configuration

$$\Delta \boldsymbol{d}^{(i)} = \boldsymbol{R}^{(i)T}_{n+1/2} \cdot \Delta \boldsymbol{D}^{(i)} \cdot \boldsymbol{R}^{(i)}_{n+1/2} \tag{1.138}$$

Step 5. The terms of the symmetric tensor $\Delta \boldsymbol{d}^{(i)}$ define the strain increments for use in a conventional small-strain model. Invoke the small-strain model to provide the i th estimate for the *unrotated* Cauchy stress at $n + 1$

$$\boldsymbol{t}^{(i)}_{n+1} \leftarrow \mathcal{C}\left(\boldsymbol{t}_n, H^j_n, \boldsymbol{q}_n, \Delta \boldsymbol{d}^{(i)}\right) \tag{1.139}$$

where $\mathcal{C}$ denotes the small-strain integration process (typically, an elastic-predictor, return mapping algorithm). The integration process requires the material state at n: the unrotated Cauchy stress ($\boldsymbol{t}_n$), a set of scalar state variables denoted by H^j_n, and a set of tensorial state variables denoted by $\boldsymbol{q}_n$ which are maintained on the unrotated configuration in the model history data.

Step 6. The unrotated Cauchy stress at $n + 1$ is transformed to the Cauchy stress at $n + 1$ required for subsequent computation of element internal forces

$$\sigma_{n+1} = \boldsymbol{R}_{n+1} \boldsymbol{t}_{n+1} \boldsymbol{R}^T_{n+1} \tag{1.140}$$

Key advantages of the above steps are the absence of half-angle rotations applied to stresses (and tensorial state variables) found in co-rotational rate formulations, Eqs. (1.120) and (1.121), and most importantly, the ability to use an existing small-strain constitutive model for *Step* 5 without modification since all quantities are referred to the unrotated configuration. The disadvantage is the need to perform two polar decompositions for the stress update at each material (Gauss) point.

Consistent Tangent Operators

Tangent operators, denoted here by $\boldsymbol{E}$, are needed to form new element stiffness matrices for the i th Newton iteration during solution for step n+1 as expressed in Eqs. (1.99) and (1.102). The operators couple increments of the spatial deformation tensor expressed on the current configuration with increments of the spatial Cauchy stress required by the fully updated formulation adopted in WARP3D. Because the incremental-iterative Newton solution at the global level uses *finite* increments of quantities to advance the solution from n to n+1, rather than simple rates $\times$ dt, the tangent operators should provide incremental, *secant* relationships.

For plasticity models implemented in a small-strain setting, Simo and Taylor [82] presented the first formalized procedures to develop the (secant) relationships and coined the phrase *consistent tangent operator*. For small-strains, *consistency* implies that the finite stress increment predicted by the tangent operator, $\boldsymbol{E}^c$, acting on a finite strain increment matches (to first order), the stress increment determined by the procedures used to integrate the plasticity rate equations over the step, i.e.,

$$\tau^{(i)}_{n+1} = \tau_n + \boldsymbol{E}^c : \left(\boldsymbol{\epsilon}^{(i)}_{n+1} - \boldsymbol{\epsilon}_n\right) = \tau_n + \int_{t_n}^{t_{n+1}} \dot{\tau}\, dt \tag{1.141}$$

where τ denotes the stress measure in the small-strain setting.

In the finite-strain framework adopted for WARP3D, the notion of a consistent tangent operator for the stress-update procedure on the unrotated configuration follows directly as (in matrix-vector form)

$$\{\boldsymbol{t}\}_{n+1}^{(i)} = \{\boldsymbol{t}\}_n + \{\Delta \boldsymbol{t}\}^{(i)} = \{\boldsymbol{t}\}_n + [\boldsymbol{E}^*]_{n+1}^{(i)}\{\Delta \boldsymbol{d}^{(i)}\} \tag{1.142}$$

where the * denotes the 6×6 consistent tangent operator defined on the unrotated configuration and the vector form of the symmetric, unrotated deformation tensor, $\Delta \boldsymbol{d}^{(i)}$, is used.

The needed form of the above relation for the fully updated solution strategy, expressed by Eq. (1.99), is

$$\sigma_{n+1}^{(i)} = \sigma_n + \boldsymbol{E}^c : \left(\boldsymbol{\epsilon}_{n+1}^{(i)} - \boldsymbol{\epsilon}_n\right) = \sigma_n + \int_{t_n}^{t_{n+1}} \dot{\sigma} dt \ . \tag{1.143}$$

where the spatial rate of Cauchy stress is integrated over $n \rightarrow n+1$. Using the Green-Naghdi rate of Cauchy stress from Eq. (1.121), the above expression becomes

$$\sigma_{n+1}^{(i)} = \sigma_n + \boldsymbol{E}^c : \left(\boldsymbol{\epsilon}_{n+1}^{(i)} - \boldsymbol{\epsilon}_n\right) = \sigma_n + \int_{t_n}^{t_{n+1}} \left(\hat{\sigma} + \boldsymbol{\Omega}\sigma - \sigma\boldsymbol{\Omega}\right) dt \ . \tag{1.144}$$

Simo and Hughes [84] and Cuitino and Ortiz [18] discuss the difficulty of constructing the consistent tangent operator implied above by $\boldsymbol{E}^c$ which includes potentially large-rotation effects over the step coupled with material stress increments caused by the deformation increment.

In the following we use a variation of the approximate linearization to define the transformation $[\boldsymbol{E}^*] \rightarrow [\boldsymbol{E}]$ employed in the NIKE codes and in ABAQUS. Computational experience indicates the procedure is quite robust and maintains good rates of convergence in the Newton iterations. We drop the iteration indicator (i) for simplicity and we use the vector form, $\Delta\boldsymbol{\epsilon}$, of the symmetric, spatial deformation tensor, $\Delta\boldsymbol{D}$. A mix of tensor and matrix-vector operations provides the most straightforward presentation.

The relationship between the tensor forms of the spatial deformation rate and the unrotated deformation rate, Eq. (1.118), is re-written in matrix-vector form as (using standard conversion of the rotation operation from tensor to matrix format)

$$\{\dot{\boldsymbol{\epsilon}}\} = [\boldsymbol{T}]\{\boldsymbol{d}\} \tag{1.145}$$

where the 6×6 matrix $[\boldsymbol{T}]$ is defined using $\boldsymbol{R}_{n+1}$. The terms of $[\boldsymbol{T}]$ are given by

$$[T] = \begin{bmatrix} R_{11}^2 & R_{12}^2 & R_{13}^2 & 2R_{11}R_{12} & 2R_{13}R_{12} & 2R_{11}R_{13} \\ R_{21}^2 & R_{22}^2 & R_{23}^2 & 2R_{21}R_{22} & 2R_{23}R_{22} & 2R_{21}R_{23} \\ R_{31}^2 & R_{32}^2 & R_{33}^2 & 2R_{31}R_{32} & 2R_{33}R_{32} & 2R_{31}R_{33} \\ R_{11}R_{21} & R_{12}R_{22} & R_{13}R_{23} & (R_{11}R_{22} + R_{21}R_{12}) & (R_{12}R_{23} + R_{13}R_{22}) & (R_{11}R_{23} + R_{13}R_{21}) \\ R_{21}R_{31} & R_{32}R_{22} & R_{23}R_{33} & (R_{21}R_{32} + R_{22}R_{31}) & (R_{22}R_{33} + R_{32}R_{23}) & (R_{21}R_{33} + R_{23}R_{31}) \\ R_{11}R_{31} & R_{12}R_{32} & R_{13}R_{33} & (R_{11}R_{32} + R_{12}R_{31}) & (R_{12}R_{33} + R_{13}R_{32}) & (R_{11}R_{33} + R_{31}R_{13}) \end{bmatrix} \tag{1.146}$$

The rate of unrotated Cauchy stress, Eq. (1.123), may then be written in matrix form as

$$\{\dot{\boldsymbol{t}}\} = [\boldsymbol{E}^*]\{\boldsymbol{d}\} = [\boldsymbol{E}^*][\boldsymbol{T}]^T\{\dot{\boldsymbol{\epsilon}}\} \ . \tag{1.147}$$

where orthogonality of the rotation matrix $[\boldsymbol{T}]$ is used. Note that $[\boldsymbol{E}^*]$ actually used in computations is the consistent tangent operator defined by Eq. (1.142). Now the Green-Naghdi stress rate in Eq. (1.122) becomes

$$\{\hat{\sigma}\} = [\boldsymbol{T}]\{\dot{\boldsymbol{t}}\} = [\boldsymbol{T}][\boldsymbol{E}^*][\boldsymbol{T}]^T\{\dot{\boldsymbol{\epsilon}}\} \tag{1.148}$$

and existing symmetries of $[\boldsymbol{E}^*]$ are preserved through the $[\boldsymbol{T}]$ transformation.

We invoke the relationship between the Green-Naghdi stress rate and the spatial rate of Cauchy stress rate given by Eq. (1.121). The left side of Eq. (1.121) is simply the symmetric tensor form of $\{\hat{\sigma}\}$ given above. To arrive at a tractable form for the $-\boldsymbol{\Omega}\sigma + \sigma\boldsymbol{\Omega}$ terms, the approximation $\boldsymbol{W} \doteq \boldsymbol{\Omega} = \dot{\boldsymbol{R}}\boldsymbol{R}^T$ is adopted. Nagtegaal and Veldpaus [66] demonstrated the validity of this approximation when the rate of logarithmic strain remains constant over the step, which is consistent with the present stress updating procedure. Moreover, they showed that the $-\boldsymbol{W}\sigma + \sigma\boldsymbol{W}$ terms could be re-cast in matrix form (using the $\boldsymbol{W} = \boldsymbol{L} - \boldsymbol{D}$ decomposition with $\boldsymbol{L}$ given by $\partial\boldsymbol{v}/\partial\boldsymbol{x}$ in Eq. (1.110)) as

$$-\boldsymbol{W}\sigma + \sigma\boldsymbol{W} \rightarrow [\boldsymbol{Q}]\{\dot{\boldsymbol{\epsilon}}\} \tag{1.149}$$

where the assumption of incompressibility becomes necessary to arrive at a symmetric form of $[\boldsymbol{Q}]$. The terms of $[\boldsymbol{Q}]$ are

$$[\boldsymbol{Q}] = \begin{bmatrix} 2\sigma_{11} & 0 & 0 & \sigma_{12} & 0 & \sigma_{13} \\ 0 & 2\sigma_{22} & 0 & \sigma_{12} & \sigma_{23} & 0 \\ 0 & 0 & 2\sigma_{33} & 0 & \sigma_{23} & \sigma_{13} \\ \sigma_{12} & \sigma_{12} & 0 & \frac{1}{2}(\sigma_{11}+\sigma_{22}) & \frac{1}{2}\sigma_{13} & \frac{1}{2}\sigma_{23} \\ 0 & \sigma_{23} & \sigma_{23} & \frac{1}{2}\sigma_{13} & \frac{1}{2}(\sigma_{22}+\sigma_{33}) & \frac{1}{2}\sigma_{12} \\ \sigma_{13} & 0 & \sigma_{13} & \frac{1}{2}\sigma_{23} & \frac{1}{2}\sigma_{12} & \frac{1}{2}(\sigma_{11}+\sigma_{33}) \end{bmatrix} \tag{1.150}$$

By expressing each term of Eq. (1.121) in matrix-vector form, the spatial rate of Cauchy stress is given by

$$\{\dot{\sigma}\} = \left[[\boldsymbol{T}][\boldsymbol{E}^*][\boldsymbol{T}]^T - [\boldsymbol{Q}]\right]\{\dot{\boldsymbol{\epsilon}}\} = [\boldsymbol{E}]\{\dot{\boldsymbol{\epsilon}}\} \ . \tag{1.151}$$

This expression defines the finite strain-rotation form of the tangent operator for use in construction in the element tangent stiffness in Eq. (1.102). This form is not a true consistent operator as the kinematic transformation uses the *rate* expressions at n+1 rather than the secant relationship from n to n+1. Use of the constitutive consistent $[\boldsymbol{E}^*]$ seems to be far more important for convergence.

The tangent operator defined in Eq. (1.151) appears in the NIKE-2D and NIKE-3D (implicit) codes which also adopt a Green-Naghdi stress rate and stress updating procedure followed here. However, the $[\boldsymbol{Q}]$ term is omitted in forming the element tangent stiffness such that $[\boldsymbol{E}] \doteq [\boldsymbol{E}^*]$. Our numerical experiments indicate that inclusion of $[\boldsymbol{Q}]$ is essential to maintain quadratic rates of convergence in the global Newton iterations when large portions of the model undergo nearly homogeneous deformation. In other instances, $[\boldsymbol{Q}]$ may be omitted as in the NIKE codes without a detrimental effect on convergence rates. The nonlinear solution parameters defineable in WARP3D enable the user to include-exclude the $[\boldsymbol{Q}]$ matrix.

Polar Decomposition

The polar decomposition $\boldsymbol{F}=\boldsymbol{RU}$ is a key step in the stress-updating algorithm and must be performed twice for each Gauss point for each stress update, i.e., at $n + 1/2$ and $n + 1$. The computational effort required for the polar decomposition should be insignificant relative to the element stiffness computation and the equation solving effort. For their explicit code, Flanagan and Taylor [24] developed an algorithm for the integration of $\dot{\boldsymbol{R}} = \boldsymbol{\Omega R}$ that maintains orthogonality of $\boldsymbol{R}$ for the very small displacement increments characteristic of explicit solutions. Numerical tests readily show their procedure fails for large displacement increments experienced with implicit global solutions. The following algorithm removes such approximations by providing an exact construction of $\boldsymbol{R}$ and $\boldsymbol{U}$ for arbitrary size load steps and yet remains computationally very efficient with the framework of an implicit solution.

Step 1. Compute the right Cauchy-Green tensor

$$\boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F} \tag{1.152}$$

and its square

$$\boldsymbol{C}^2 = \boldsymbol{C}^T\boldsymbol{C} \tag{1.153}$$

where only the upper-triangular form of the symmetric products (6 terms) are actually computed and stored.

Step 2. Compute the eigenvalues λ_1^2, λ_2^2 and λ_3^2 of $\boldsymbol{C}$. A Jacobi transformation procedure specifically designed for 3×3 matrices is used to extract the eigenvalues. For scalar computers, the do-loops are eliminated by explicitly coding each off-diagonal rotation form. Two or, at most, three sweeps are needed to obtained eigenvalues converged to a 10^{-6} tolerance. The procedure vectorizes easily since there are no transcendental functions to evaluate; the number of iterations is fixed at two or three for all material points in a contiguous block of elements.

Step 3. Compute invariants of $\boldsymbol{U}$ and the det($\boldsymbol{F}$)

$$I_U = \lambda_1 + \lambda_2 + \lambda_3 \tag{1.154}$$

$$II_U = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_1\lambda_3 \tag{1.155}$$

$$III_U = \lambda_1\lambda_2\lambda_3 = J = det(\boldsymbol{F}) \tag{1.156}$$

Step 4. Form the upper triangle of the symmetric, right stretch, $\boldsymbol{U}$, and it's symmetric inverse, $\boldsymbol{U}^{-1}$ (see Hoger and Carlson [38])

$$\boldsymbol{U} = \beta_1(\beta_2\boldsymbol{I} + \beta_3\boldsymbol{C} - \boldsymbol{C}^2) \tag{1.157}$$

where **I** denotes a unit tensor with the β coefficients defined by

$$\beta_1 = 1/(I_U II_U - III_U), \quad \beta_2 = I_U III_U, \quad \beta_3 = I_U^2 - II_U \tag{1.158}$$

Similarly, the inverse of **U** may be formed directly as

$$U^{-1} = \gamma_1(\gamma_2 I + \gamma_3 C + \gamma_4 C^2) \tag{1.159}$$

where the γ coefficients defined by

$$\gamma_1 = 1/III_u(I_U II_U - III_U), \quad \gamma_2 = I_U II_U^2 - III_U(I_U^2 + II_U), \tag{1.160}$$

$$\gamma_3 = -III_U - I_U(I_U^2 - 2II_U), \quad \gamma_4 = I_U \tag{1.161}$$

Step 5. Form $\boldsymbol{R}$ as the product

$$\boldsymbol{R} = \boldsymbol{F}\boldsymbol{U}^{-1} \tag{1.162}$$

Chapter 2

Model Definition

This chapter describes the commands to define a finite element model, to define a nonlinear/dynamic solution algorithm, to request an analysis for a number of load steps and to request output. Commands in this chapter are described in the recommended order of input:

- structure name and sizes (number of nodes and elements)
- definition of "materials" for association with elements in the model. Materials provide linear elastic properties, material density, nonlinear properties and a "type" of constitutive algorithm, e.g., rate–dependent Mises plasticity with isotropic hardening.
- the type of each finite element in the model, the kinematic formulation for the element (large or small displacements) and the values of any properties for the element, e.g., the order of numerical integration
- the *X*–*Y*–*Z* coordinates for all model nodes in the model global coordinate system
- the incidences for all elements in the model. Incidences define the connectivity of element nodes to model nodes
- the assignment of contiguous lists of elements to "blocks" for analysis. Blocking is required to support parallel/vector operations on supercomputers and is retained for analyses conducted on Unix workstations to beter utilize cache memory. All elements in a block must be the same type, have the same material model, the same type of kinematic formulation. For Crays and the LPCG solver with the Hughes–Winget preconditioner, elements in a block must not be connected to a common node.
- displacement constraints imposed on nodes of the model, either zero or non–zero.
- loading patterns for the model. Loading patterns consist of nodal forces; element body forces, face tractions, face pressures which are converted to equivalent nodal forces; nodal and element temperature changes relative to a zero reference state.
- a nonlinear/dynamic loading which defines the increment of load to be applied during each load/time step. Loading increments for a step are defined using the loading patterns.
- parameters to control the nonlinear/dynamic solution process, e.g., the time increment for dynamic analysis, the type of equation solver (direct, conjugate gradient), the maximum number of Newton iterations, adaptive loading parameters etc.
- parameters to control the type of crack growth (node release, cell extinction)
- a request to compute displacements for a list of load steps
- a request to output computed nodal and element results. Results for use by humans are directed to the current output device with appropriate pagination, headers, labels, etc.
- a request to output computed nodal and element results in the format defined by the PAtran modeling software. These results files are readable by Patran without further conversion.
- a request to compute and output values for the *J*–integral in fracture mechanics models
- a "save" command to write all current, essential data structures to a sequential binary file for later use to re–start an analysis.
- a "stop" command to terminate program execution

In typical analyses, multiple compute, output, *J*–integral and save commands appear in the input. Parameters to control the nonlinear/dynamic solution algorithm, e.g., the time step, may be modified between analyses for sets of load steps. Constraints can be modified between analyses for load steps to effect incremental changes in the boundary conditions.

Some model descriptors cannot be modified once defined. For example, the number of nodes and elements, the element types and properties, the coordinates, the incidences and the blocking cannot be altered.

2.1 Model Name and Sizes

The definition of a new finite element model begins with specification of an alphanumeric identifier. The identifier appears on all pages of output. The command has the form

$$\underline{\text{struct}}\text{ure} \quad < \text{name: label} >$$

The first eight characters of model names are recognized. Longer names are accepted on the command but truncated to the eight character limit.

The number of nodes and number of elements in the model must be specified prior to any other command related to nodal or elemental quantities. WARP uses the specified sizes to support checking of the input data as it is entered and to support exhaustive consistency checking of the structural model for errors prior to the first compute request. An example of such an error is a node with no elements attached. The model sizes are defined with a command having the form

$$\underline{\text{num}}\text{ber } (\underline{\text{of}}) \left[\left\{ \begin{array}{c} \underline{\text{nodes}} \\ \underline{\text{elem}}\text{ents} \end{array} \right\} < \text{size: integer } (,) > \right]$$

Examples of the above commands are:

```
structure bend_strip
  number of nodes 3450 elements 4230
```

and

```
structure bend_strip
  number of nodes 3450
  number of elements 4230
```

All node and element identifiers are positive integers beginning with the value 1. Nodes and elements must each be numbered sequentially.

Once specified, the number of nodes and elements cannot be modified through user commands.

Limits on Number of Nodes and Elements

The maximum number of nodes and elements permitted in a model varies with the version of WARP being executed and the computer executing the program. Typical limits are 25,000 nodes and 25,000 elements for a Unix workstation version and 100,000 elements and 100,000 nodes for a Cray and SGI (Power Challenge and Origin 2000) versions. These limits are easily changed through one line in the source code followed by a re–compilation on the hardware platform.

2.2 Material Definitions

Finite elements in a model are associated with "materials" from which they derive elastic properties, mass density and nonlinear characteristics, if necessary. Through the *material* command, the user specifies a convenient name for the material, the type of constitutive model (e.g., rate-dependent Mises) and the values of any properties required by the material model. Material definitions must precede the specification of element properties during input.

Some models provide an option to specify nonlinear response in the form of a piecewise-linear description, i.e., a tensile stress-strain curve. The *stress-strain curve* command is used to describe points on the piecewise-linear curve for use by the material model.

This section describes the *material* and *stress-strain curve* commands. When a *material* command references a *stress-strain curve*, there is no requirement that the referenced curve be defined previously. Consistency checks are performed prior to any computations.

2.2.1 Material Command

A *material* command on a separate line initiates the material definition sequence. Any number lines may follow to define the properties required for the material model. The definition of an element requires the following information:

The command syntax is

```
material  < material id:  label >

  properties  < model type: label >  [ < matl  prop:  label > (< value >) ]
```

The logical input line for the properties may be continued over multiple physical input lines with commas at any point. Subsequent sections in Chapter 3 define the "type" of material models currently available and the properties required for each model type.

An example of material specification is

```
material al2024t
  properties mises e 10350 nu 0.3 yld_pt 50.0 n_power 10,
                  rho 0.1254e-07 alpha 5.4e-06
```

In this example, the material is named "al2024t" and the computational model for the material is "mises" (one of the models described in Chapter 3). Keywords "e", "nu", "n_power" are properties of the *mises* model assignable by the user.

The following example refers to *stress-strain curve 3* for a piecewise-linear description of the uniaxial, tensile stress-strain curve

```
material a36
  properties mises e 30000 nu 0.3 curve 3 rho 0.1254e-07
```

Once defined, the specification for a material cannot be modified at any further point in the analysis.

2.2.2 Stress–Strain Curve Command

The uniaxial, tensile stress-strain response of certain materials requires a general segmental curve description for a realistic representation. Materials that exhibit a sharp yield point, a Luder's band and then strain hardening are classic examples not amenable to modeling with the power-law type curves. Figure 2.1 provides an example of a stress-strain curve described with a piecewise-linear model.

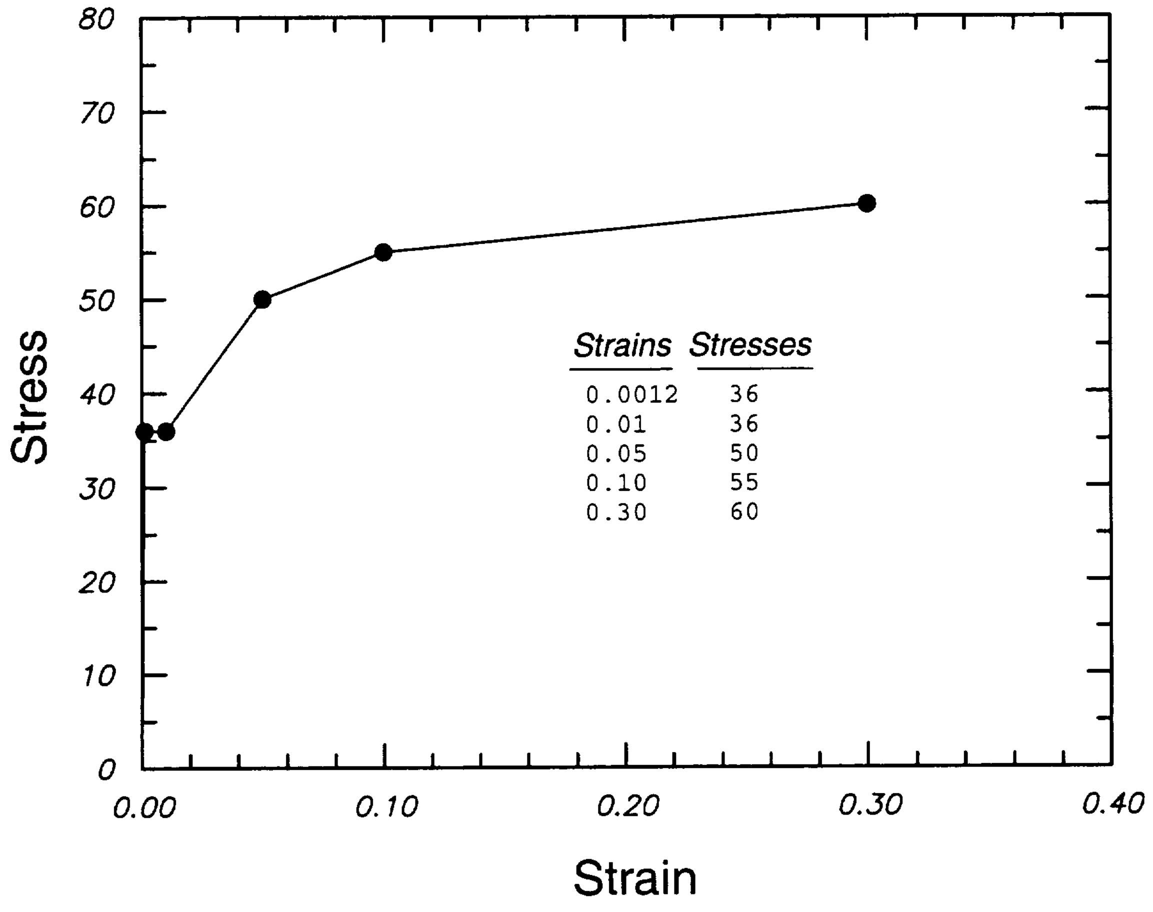


FIG. 2.1—*Example of piecewise-linear stress-strain curve.*

Points on such curves are specified with a simple command sequence *stress-strain curve* where each such curve required in the analysis is assigned an integer number for identification. The curve may then be referenced in a *material* command as described above. The command syntax is

```
stress(-strain) (curve) < curve number:  integer >

  [ < strain value:  numr > < stress value: numr >  (,) ]
```

Curve points are input as strain-stress pairs; use as many lines as needed to specify the points. Multiple pairs may be specified on a line. All strain-stress values must be positive. *Do not specify the (0,0) point on the curve.* The first point defines the yield strain and yield stress. Young's modulus specified in the *material* command *must* match the value implied by the yield strain-yield stress pair. After the last specified point, the response is assumed perfectly plastic.

The strain values input here are the *total* strains (not the plastic strains!). For large-deformation analyses, the values should correspond to the logarithmic strain-Cauchy stress; for small strain-analyses the values should be engineering strain-engineering stress.

A maximum of 10 curves may be specified for use in an analysis. Each curve may have up to 100 strain-stress pairs defined.

The above curve is described with the command sequence

```
stress-strain curve 3
  0.0012 36,  0.01 36,  0.05 50,
  0.10 55, 0.30 60
```

2.3 Element Types and Properties

The types of finite elements and their properties are specified prior to any compute requests. An *elements* command on a separate line initiates the element definition sequence. Any number lines may follow to define the types and properties of all elements in the model. The definition of an element requires the following information:

- the "type" of element (e.g. *l3disop*, *ts15isop*, etc.)
- the kinematic formulation (small or large displacements)
- reference to a previously defined "material" that defines elastic properties, mass denisty and the nonlinear properties (if required)
- a list of element property identifiers and associated values, e.g., the order of numerical integration.

The command syntax is

$$\underline{\text{elements}}$$

$$\text{< element nos.: list > } \underline{\text{type}} \text{ < element type: label >} \left\{ \begin{array}{c} \underline{\text{linear}} \\ \underline{\text{nonlinear}} \end{array} \right\} \text{(,)}$$

$$\underline{\text{material}} \text{ < matl. id: label > [< elem. prop: label > < value >]}$$

The logical input line may be continued over multiple physical input lines with commas at any point. Subsequent sections in Chapter 3 define the "type" of elements currently available and the properties available for each element type. Element properties typically have a property keyword followed by a value. Some element properties are "logical" values which take on "true" values by the presence of the keyword.

The keyword *linear* requests a conventional small displacement, small strain element formulation. This is the default formulation and is adopted if no specification is given. The keyword *nonlinear* requests a geometric nonlinear formulation that models large rotations and finite strains.

Every element must have an associated *material*. Materials must be specified prior to their use in element specification.

An example of elements specification is

```
elements
  1-40 type l3disop linear material a36 center_output bbar,
                            order 2x2x2
  500-1000, 1200-200 by -2 q3disop nonlinear material al_2024t,
                            order 14pt_rule long
```

Once defined, the specification for an element cannot be modified at any further point in the analysis.

2.4 Nodal Coordinates

The coordinates of nodes are specified relative to the global Cartesian reference axes. During model definition, the command *coordinates* initiates the translation of nodal coordinate data. Any number of *coordinates* commands may be given prior to a *compute* request. The existing coordinates for nodes are simply overwritten by any newly specified values. The input syntax is

```
coordinates (clear)

                          [ ( x )                               ]
< node number: integer >  [ { y } < coord. value: number > (,)  ]
                          [ ( z )                               ]


< node number: integer >  [ < coord. value: number > (,) ]
```

where the second form applies the default ordering of entries *X–Y–Z*. *When using the second form, coordinates not specified take on the last previously defined values.* For example, the sequence

```
coordinates
  4 3.2 5.2 6.4
  10 4.1
```

defines the *Y* coordinate of node 10 as 5.2 and the *Z* coordinate of node 10 as 6.4. This feature may be suppressed by appending the word *clear* to the *coordinates* command line. The default coordinates for every node are then 0.0 unless explicitly input. With this option for the above example, node 10 is assigned coordinates of 4.1, 0.0, 0.0 rather than 4.1, 5.2, 6.4.

The default *X–Y–Z* ordering for the second input form may be modified by the *default* command

```
coordinates
          [ ( x ) ]
default   [ { y } ]
          [ ( z ) ]


< node number: integer >  [ < coord. value: number > (,) ]
```

where any number of *default* commands may be given.

Some examples illustrating various options to define nodal coordinates are given below.

```
coordinates
  4 x 2.5 y 3.0 z 4.1
  10 z -20 y 40 x 20
```

```
11 -5.23 6.23
default z y x
  3  15.3  14.2   10.5
default x y z
  10 -13.5 10.5 -20.4
```

At any point during input of the coordinates, the *dump* command is available to request a listing of current coordinates for all nodes of the model.

```
coordinates
  4 x 2.5 y 3.0 z 4.1
  10 z -20 y 40 x 20
  11 -5.23 6.23
  dump
  default z y x
    3  15.3  14.2   10.5
  default x y z
    10 -13.5 10.5 -20.4
  dump
```

2.5 Element Incidences

Each node of an element in the model must be "mapped" onto the corresponding global node. Element *incidences* establish this correspondence. During model definition, the command *incidences* initiates the translation of element incidence data. Any number of *incidences* commands may be given prior to a *compute* request. The existing incidences for elements are simply overwritten by any newly specified values. The input syntax is

```
incidences

< element number: integer > [ < global node i: integer list > (,) ]
```

where <global node i> denotes the number of the global node to which the element node is attached. Note that the list of global node numbers may be specified as an integer list.

An example of the incidences command is

```
incidences
  1    13-20
  2    5 40 65 83 92 120 44 98
  3    140-144 178 162 183
```

The number of entries in the integer list must equal the number of nodes on the element (8 for *l3disop*, 12 for *ts12isop*, etc.). Error messages are issued by the input processor if the number of nodes is less than required, if a node number exceeds the number of structure nodes, etc. A warning message is issued if the same node appears more than once in the integer list.

The ordering of nodes for each element is shown in Chapter 3 where the element library is described.

2.6 Element Blocking

All element level computations in WARP proceed on a block-by-block basis to facilitate vectorization and parallel processing of element blocks. Each element must be assigned to a block. The maximum number of elements in a block is set by a compile-time variable in the WARP source code and is selected to optimize performance on specific types of computers. On a CRAY-90, for example, the block size is 128 since the vector processor units have registers each of length 128 words. On a Unix workstation, the block size impacts the efficient use of cache memory; large blocks cause severe thrashing in the cache. A typical block size for a workstation is 32, 64, or 128. Blocking improves computational performance of the code even on Unix workstations without vector hardware. Very efficient subroutines to perform common vector-matrix operations available on workstations provide the improved performance during element level operations.

The assignment of elements to blocks is most conveniently handled by the pre-processor software employed to create the finite element model. The *patwarp* program, for example, converts a Patran neutral file into a WARP input file and performs the element-to-block assignments. The block assignment commands have the form

```
blocking

 < block : integer > < block size: integer > < first element  in block: integer >
```

The following example input describes the blocking for a model with 520 elements.

```
blocking
      1   120    1
      2   112  121
      3   109  233
      4   100  342
      5    42  442
      6    24  484
      7    11  508
      8     2  519
```

The following rules define the proper assignment of elements to blocks. All elements in a block

- must be sequentially numbered
- must be the same type; e.g., *l3disop*
- must have the same kinematic formulation (*linear* or *nonlinear*)
- must have the same associated material
- must have the same integration order (e.g. $2 \times 2 \times 2$)
- must not share a common node if:
  - execution is on a vector/parallel computer (Cray, Convex)

    or
  - the Hughes-Winget pre-conditioner is selected for the conjugate gradient solver

This last requirement nearly always necessitates a re-numbering of the elements in the model to eliminate node conflicts within blocks. The *patwarp* pre-processor, for example, employs a simple "red-black" strategy to re-number elements before constructing the WARP input file.

The input translators perform checks to insure that blocking assignments follow these rules.

2.7 Nodal Constraints

WARP currently supports constraints applied to nodes: (1) in the global, Cartesian system and (2) in a *local* Cartesian system defined at selected nodes. The sequence is initiated with the *constraints* command. When the *constraints* command is encountered by the input translators, *all* previously defined constraints data are destroyed. Thus to modify constraints between load (time) steps, all the constraints must be specified — not just the constraints that have changed.

To define constraints in the global Cartesian system the input syntax is

constraints

$$\text{< node list: list >} \left[\begin{Bmatrix} \text{u} \\ \text{v} \\ \text{w} \end{Bmatrix} \text{(=) < constraint value: number > (,)} \right]$$

Examples of global constraints input include:

```
constraints
  1-100 by 3 w 4.3 v 0 u 0
  24 u = -1.3 w 0.0
```

2.7.1 Non–Global Constraints

The capability to specify constraints in non–global coordinates enables the analysis of skew supports, for example, that arise naturally in structural systems or in 3–D models of axi-symmetric structures. To define constraints in a non–global Cartesian system, consider the simple problem shown in the Fig. 2.2. Here the global and local Z axes are aligned but the local and global X, Y axes are not aligned. The user defines a 3×3 *rotation* matrix of direction cosines which transforms global vector quantities into the local coordinate system. The boundary condition shown is simply $u = 0$ in the local coordinate system. Transformation matrices are specified with the command sequence:

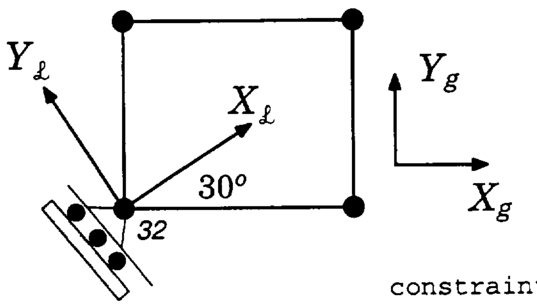
constraints

transformation matrix < node list: list >

$$\left[\begin{Bmatrix} \text{row\_1} \\ \text{row\_2} \\ \text{row\_3} \end{Bmatrix} \left[\text{< direction cosines:number>} \right] \right]$$

where any number of nodes may be associated with the specified transformation matrix; the *transformation matrix* command may be repeated as necessary within the *constraints* definition. In this example, the constraint is specified immediately following definition of the transformation matrix although this is not required.

The input system verifies that the rotation matrix specified is orthogonal and that the matrix pre–multiplied by its transpose is an identity matrix to within a tight tolerance.



$$\begin{bmatrix} X_\mathcal{L} \\ Y_\mathcal{L} \\ Z_\mathcal{L} \end{bmatrix} \begin{matrix} = \\ = \\ = \end{matrix} \underbrace{\begin{bmatrix} 0.86667 & 0.5 & 0.0 \\ -0.5 & 0.86667 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}}_{\textit{rotation matrix}} \begin{bmatrix} X_g \\ Y_g \\ Z_g \end{bmatrix}$$

```
constraints


  transformation matrix 32,
     row_1 0.86667 0.5 0.0,
     row_2 -0.5  0.8667 0.0,
     row_3 0.0 0.0 1.0
  32 u = 0
```

FIG. 2.2—*Example of Local Coordinate System for Constraint Specification*

Users are aware of such *local* coordinate systems only during the specification of constraints. Nodal loads and element loads are always specified in global coordinates. All nodal output quantities produced by WARP3D are in global coordinates.

2.7.2 Constraints in Nonlinear Analyses

In a nonlinear analysis, the currently defined constraints are interpreted as the *incremental displacement change* enforced over the next load (time) step. A non–zero constraint is enforced during the first iterative cycle for the load step. In subsequent iterations, no displacement change is permitted on the constrained displacements to maintain the value of the specified increment.

By default, the current set of constraints with a multiplier of 1.0 are imposed during each nonlinear load step. Alternatively, users can specify directly the constraint multipler in the definition of each load step (see Section 2.8.5).

2.7.3 Display of Current Constraint Data

Within the *constraints* command sequence, the *dump* command may be specified to request a display (listing) of the current constraints information taken from internal tables.

2.8 Loads (Including Temperatures, Displacements)

Loads and temperature changes may be applied to the nodes and elements of a model. Element loads, which are dependent on the type of finite element, and nodal temperatures are converted to equivalent nodal loads by element processing routines. Nodal loads and element loads are grouped together to define *loading patterns*. The loading patterns define the spatial variation and reference amplitudes of loads on a model. The *constraints* defined on the model also represent a *loading pattern* but with a "built-in" name, i.e., *constraints*. Examples of loading patterns include dead load, an internal pressure, a localized temperature increase and simple bending of a component.

Once loading patterns are defined, a *nonlinear* loading condition is defined. The term *dynamic* may be used as a synonym for *nonlinear* if desired. A nonlinear/dynamic loading consists of a sequential number of load steps. An incremental-iterative solution is obtained for each load step. For dynamic analyses, a load step is the same as a time step. Each *load step* may consist of loading patterns combined with scalar multipliers. The scaled values of nodal forces (nodal loads and resulting equivalent nodal loads) and constraints for the patterns are applied as the new *incremental* load to the model during the step.

A static linear analysis must be performed as the first step of a static nonlinear analysis. A static nonlinear analysis is solved as a dynamic analysis with: (1) a very large time increment or (2) zero mass for the model. The user selects one of the two procedures by setting the time increment and the model mass.

The first sections describe the commands to define nodal forces and element loads that construct a loading pattern. Commands are then defined to specify load steps in a nonlinear/dynamic analysis (or step 1 of a static, linear analysis).

2.8.1 Loading Patterns

A new loading pattern is defined through a command of the form

```
loading < loading identifier: label >
```

where the loading identifier is used in subsequent commands to identify the loading, for example, in compute and output requests. Only the first eight characters of the identifier are processed; all loading patterns must have unique identifiers.

When an existing loading pattern is referenced in this command, newly specified node and element loads are *added* to the previously specified loads for that loading pattern. If the command references an existing nonlinear loading condition, the previously defined information for all steps is destroyed and replaced by the newly specified input.

Specified temperature values represent *relative* changes from an aribitrary (uniform) reference temeperature for the model.

2.8.2 Nodal Loads

A sequence of nodal load definitions has the form

```
nodal (loads)

                    ┌ ⎧ force_x     ⎫                            ┐
                    │ ⎪ force_y     ⎪                            │
< node list: list > │ ⎨ force_z     ⎬ (=) < value: number > (,)  │
                    └ ⎩ temperature ⎭                            ┘
```

Nodal loads are additive; if the same node and direction appear in two different loading commands the sum of two loads is applied to the model. An example sequence to define a loading condition and a set of nodal forces is

```
loading unit_pull
  nodal loads
     1-40 60-90 force_z -2.3 force_x 14 temperature -42.3
     3240 3671 4510-5000 force_z -3.12
     35 temperature 145.0 force_x 2
```

In the above example, node 35 has a total force in the X-direction of 16 (14 from the first line + 2 from the last line), in addition to a net temperature change of 102.7.

2.8.3 Element Loads

A sequence of element load definitions has the form

```
element (loads)
   < elements: list >  < type of element loading >
   < elements: list >  < type of element loading >
                        •
                        •
```

where the <type of element loading> is either a body force, a face traction with constant direction, a face pressure, or a uniform temperature change for the entire element. The types of element loads and commands to define them are dependent on the type of element. Refer to Chapter 3 for this information.

When the analysis includes geometric nonlinear effects (large displacements), equivalent loads for the incrementally applied surface tractions are re-computed at the beginning of each load step using the current (deformed) geometry of the elements.

Nodal forces are always applied in the global coordinate system and are thus unaffected by the deformed geometry.

2.8.4 Step Loads

The loading type designated *dynamic* or *nonlinear* defines the combinations of pattern loads for each time step in a dynamic analysis or each load step in a static nonlinear analysis. These commands have the form

```
loading < loading identifier: label >
  nonlinear
      steps < steps: list > [ < pattern id: label > < multiplier: number > (,) ]
```

where the keyword *dynamic* may be substituted as a synonym for *nonlinear*. Nodal and element loads cannot be specified within a nonlinear/dynamic loading definition above. The multiplier value must follow each pattern id— a multiplier value is required input. As indicated, multiple pattern loads may be combined with different multipliers to define a load increment for a time step in a dynamic analysis or a load step in a static nonlinear analysis.

By default the existing constraint definitions are included in each load step with a multiplier of 1.0. The constraints used in solution for a load step are the constraints defined at the actual solution time for the step (users can re-define the constraints data at any time— the 1.0 multiplier applies to the currently defined constraints at step solution time). The user may include *constraints* as a loading pattern with a multiplier other than 1.0.

An example of this command sequence is

```
loading crush
  nonlinear
      steps 1-10 unit_pressure 2.3 unit_tens -1.2 constraints 1.0
      steps 11-200 pull 0.2 constraints 2.3
```

where the loading patterns *unit_pressure*, *unit_tens* and *pull* have been defined previously. Although the steps are defined in ascending sequence in the above example, the steps may be defined in any order; the final set of steps must comprise a sequential list.

Modifying Step Definitions

During the course of a nonlinear or dynamic analysis, it is often necessary to define additional steps or to modify the definition of steps yet to be analyzed. For example, previously defined, but unsolved, load steps may need to have a reduced multiplier based on current convergence properties.

Two approaches are available to perform this task. In the first approach, a new nonlinear/dynamic loading condition may be defined with the desired definition for the new/modified load steps. Subsequent compute requests then refer to this new loading. In the second approach, the existing nonlinear/dynamic loading condition is redefined. The input translators require that all load steps 1, 2, 3, ... be re-defined with this approach. A warning message is issued to the user about this feature when an existing nonlinear/dynamic loading condition is redefined.

2.8.5 Displacement Control Loading

A nonlinear/dynamic loading condition with appropriate step definitions must be always be specified for a model. This becomes a slight inconvenience when the model is loaded only by imposed non-zero displacements at selected nodes. The recommended procedure for displacement control loading is:

- define a loading pattern "dummy" with a meaningless, zero nodal force (put a force of 0.0 on one node)
- define the nonlinear/dynamic loading condition. All steps refer to the loading pattern "dummy" with a multiplier of 1.0.

This procedure forces the processing routines to create the necessary internal data structures required for an analysis. An example of these commands is

```
loading dummy
  nodal loads
```

```
          1 force_x 0.0

     loading crush
       nonlinear
          steps 1-100 dummy 1.0 constraints 1.3
```

Effects of Step Multipliers

The pattern multiplier (1.0 above) plays no role in the solution of displacement control loadings unless the *extrapolate* option of the nonlinear solution algorithm is invoked (extrapolate is *on* by default). When the extrapolate option is in effect, the incremental displacements computed from the solution over step n-1 to n are scaled and applied to the model to start the iterative (Newton) solution from n to n+1. The displacement scaling factor is computed from the specified step multipliers for steps n (say f_n) and n+1 (f_{n+1}) as f_{n+1}/f_n. Thus only the ratios of the multipliers are significant for displacement control with *extrapolate on*. When the non-zero constraints are modified during a displacement control analysis, the loading step multipliers must be modified accordingly by the user; otherwise the extrapolation ratio (f_{n+1}/f_n) is computed incorrectly.

To illustrate, consider the following example. Non-zero constraints are specified to load the model. The dummy loading pattern and nonlinear loading are defined as above with step multipliers of 1.0 for load steps 1-10. After step 10, the user modifies the constraints to reduced the imposed increment (uniformly) by one-half, possibly to reduce the number of Newton iterations for convergence in subsequent steps. Load steps 11, 12, 13, ... must have a multiplier of 0.5 for correct extrapolation. In step 11, the extrapolation multiplier is 0.5/1.0 = 0.5 while in steps 12, 13, ... the multiplier again becomes 1.0.

2.9 Solution Parameters

The nonlinear (and dynamic) computational procedure in WARP follows an incremental-iterative strategy with full Newton iterations to eliminate residual nodal forces caused by nonlinear behavior. The user has full control over the solution procedures through a wide range of parameters. Each of the parameters has a built-in default value which may be redefined by the user. The values of these parameters are declared by the user before computation begins for the first load step; those values remain in effect unless modified by the user as the solution progresses through the load steps. New values for these parameters may be defined whenever the input translators accept new input lines. The most current values of the parameters then control subsequent computations over load steps.

The specification of solution parameters begins with a command of the form

$$\left\{\begin{array}{l}\underline{\text{solution}}\\ \underline{\text{nonlinear}}\\ \underline{\text{dynamic}}\end{array}\right\}\ (\underline{\text{analysis}})\ \ (\underline{\text{para}}\text{meters})$$

and terminates whenever a command is given that does not define a parameter controlling the analysis. The following sections describe each of the parameters assignable by the user and the command syntax. An example defining values for selected solution parameters is shown below for reference.

```
dynamic analysis parameters
   solution technique lnpcg
   preconditioner ebe
   lnr_pcg conv test res tol 0.01
   maximum linear iterations 2000
   maximum iterations 10
   convergence test norm res tol .5
   time step 0.05
   trace solution on
   linear stiffness iteration one off
c
c
compute displacements for loading dead_live step 1-5
```

2.9.1 Linear Equation Solvers

The linearized set of equilibrium equations for the model is solved by one of three computational procedures. The first is a "direct" solver which assembles the upper-triangular stiffness matrix for the model (in profile format) and executes a conventional Choleski factorization, forward load pass and backward load pass. The second is a "direct" solver which employs sparse matrix technology with Choleski factorization, forward load pass and backward load pass. The multi-minimum degree re-ordering of the equations adopted in the sparse solver dramatically reduces memory and CPU requirements compared to the conventional direct solver. More efficient, platform specific versions of the sparse solvers are also available. The third is an iterative, element-by-element, linear preconditioned conjugate gradient solver (LPCG). This solver does not assemble the structural stiffness matrix and thereby greatly reduces memory requirements. A choice of two preconditioners is available: (1) a diagonal preconditioner which employs the diagonal terms of the dynamic stiffness for the model, and (2) Crout factorization of the "regularized" dynamic tangent stiffness (implemented on an element-by-element basis as outlined by Hughes-Winget).

Direct Solvers

The direct solvers provide an "exact" solution for the linearized equations within round-off features of the computer hardware. The direct solver is recommended for all problems smaller than a few hundred nodes and for all problems in which 3-D elements model a plane-stress, plane-strain or thin plate-shell type structures. Such models have very large in-plane dimensions and only one or two elements in the thickness direction.

The conventional direct solver which assembles the full upper-triangular profile becomes inefficient very quickly as the 3-D nature and size of the model increases— inefficient in terms of both required memory for the assembled stiffness and the factorization time. The *sparse* version of the direct solver should be used for larger models. On Unix workstations, the sparse direct solvers remain competitive with the conjugate gradient solver for very large models. Memory requirements for the sparse solver are many times smaller than those of the conventional direct solver.

The direct solver is the default computational procedure in WARP and is explicitly specified with the command

```
solution (technique) direct
```

To request the generic sparse solver available on all platforms, simply append the keyword *sparse* after the keyword direct.

```
solution (technique) direct sparse
```

To request the sparse solver on HP workstations installed, simply append the keyword *hp* after the keywords direct sparse.

```
solution (technique) direct sparse hp
```

The HP supplied sparse solver is highly tuned for the PA RISC architecture. It runs in-core only at present.

To request the sparse solver on Cray computers which have the Boeing BCSLIB installed, simply append the keyword *bcs* after the keywords direct sparse.

```
solution (technique) direct sparse bcs
```

WARP invokes the BCSLIB solver using an out-of-core algorithm with minimum memory use requested.

To request the vendor supplied sparse solver on SGI computers, simply append the keyword *sgi* after the keywords direct sparse.

For parallel execution on SGI computers, be sure to set the number of threads through an environment variable before running WARP3D. The SGI solver may also be executed in an out-of-core mode to reduce the real memory requirements (non-parallel execution only). Users specify the amount of real memory that the solver may allocate (in mega-bytes) and the

```
solution (technique)  direct sparse sgi
```

directory on which it writes scratch files. The commands to invoke these options are:

```
solver out(–of–core) { on  }
                     { off }

solver memory  <memory:numi>

solver scratch directory <directory:string>
```

where the full path name for the scratch directory must be specified (no ~ for example in the path name). The default scratch directory is /usr/tmp.

Conjugate Gradient Solver

The linear preconditioned conjugate gradient (LPCG) solver iteratively improves an initial estimate for the solution of the linearized equilibrium equations. The iterations continue until further changes in the displacement increments yield no significant improvement in the solution.

The element-by-element implementation of the LPCG solver eliminates construction of the assembled stiffness matrix. Memory requirements for the LPCG solver are thus many times smaller than for the direct solver. A 7,000 element/node model runs without (virtual memory) paging on a 64 MB Unix workstation.

The number of LPCG iterations required to converge on the correct displacement increment varies with the characteristics of the model. The very best convergence rate derives from a model of uniformly (cube) sized elements arranged in a cube. In this case, the diagonal preconditioner (DPC) provides a solution in a number of LPCG iterations less than the square root of the number of active nodal degrees of freedom (dof). The DPC performs exceptionally well in dynamic analyses with small time increments. Some models that exhibit very poor LPCG convergence with DPC in static loading converge very rapidly in dynamic loading. For non-uniform element sizes, large time increments in dynamic analyses, decreased "three-dimensionality" of the model and increased nonlinearity, the number of LPCG iterations may exceed 3-5 $\times$ (active no. of dof)$^{1/2}$.

Fracture mechanics models with focused meshes and orders of magnitude variations in element sizes define a very difficult configuration for the LPCG solver. For models at the extremes of these conditions, a solution may not be possible with the DPC. The Hughes-Winget preconditioner (HWPC) is available for LPCG solution in these models. The HWPC increases the computational cost per LPCG iteration by a factor of $\approx$ 2.1-2.3 $\times$ the cost per DPC iteration. The HWPC produces a converged solution in nearly all cases which fail with the DPC. Moreover, when both DPC and HWPC produce solutions, the number of LPCG iterations with HWPC is often 0.3-0.4 $\times$ the number of DPC iterations which yields a net reduction in total solution times. Numerical experiments with large models provide guidance on the optimum choice of a preconditioner.

The LPCG solver employs the following convergence test to assess the solution quality. Let r_0 be the residual vector for solution of the linear equations evaluated for the initial (estimated) displacement increment:

$$K_D \cdot \Delta u_0 - \Delta P = r_0$$

where K_D denotes the dynamic stiffness. The initial displacement vector Δu_0 is set to zero except those for non-zero terms of the user specified (current) constraints. The vector ΔP denotes the incremental load. During Newton iteration 1, ΔP contains the applied load over the step (including inertia effects); during subsequent iterations, ΔP contains the residual load. LPCG iterations continue until at the *k*th iteration with Δu_k available

$$\| r_k \| \le (user\ tol/100) \times \| r_0 \|$$

where $\| \; \|$ denote the Euclidean norm. Tolerance values are specified in (%); thus, a user tolerance of 0.01 (%) is reasonably strict and often used. Tolerance values of 0.001-1.0 have been used successfully in various models. The user specified tolerance exerts a dramatic impact on the required number of LPCG iterations and the total CPU time. Excessively tight tolerances do not provide real improvements in solutions. If the model has a linear elastic material and a kinematically linear formulation, the convergence tests performed after the first Newton iteration of a load step provide a very good indicator of the linear solution quality. An excessively large residual indicates that a smaller LPCG tolerance value is needed.

The approximate nature of LPCG solution provides an opportunity to balance accuracy and CPU time for linear equation solving with the number of Newton iterations required to eliminate residual forces arising from nonlinear behavior. During the first few Newton iterations of a load step, excessive accuracy during solution of the linear equations is often un-warranted as the force imbalances due to nonlinearity far exceed those due to remaining errors in displacement increments from the LPCG solver. The Newton iterations correct, simultaneously, the incremental displacement vector for the step due to nonlinear effects and due to small residuals in the LPCG solver. Experimentation with LPCG and Newton tolerance values in nonlinear analyses often yields substantial decreases in total solution times.

The program terminates execution if the specified number of LPCG iterations is exceeded (the default limit is 10 iterations).

The commands to specify a LPCG solver have the form

solution (technique) lnpcg

preconditioner (type) $\left\{ \begin{array}{c} \text{diagonal} \\ \text{hughes–winget} \end{array} \right\}$

lnpcg (convergence) (tests) residual (tolerance) < number >

maximum linear iterations < integer >

trace lnpcg_solution $\left\{ \begin{array}{c} \text{on} \\ \text{off} \end{array} \right\}$

An example is

```
nonlinear analysis parameters
   solution technique lnpcg
   preconditioner hughes-winget
   lnpcg conv test res tol 0.01
   maximum linear iterations 2000
```

•

•

During solution of a large nonlinear model, the LPCG solver with the DPC may converge very rapidly during early load steps when nonlinear effects remain small. Once the DPC requires an excessive number of LPCG iterations in later load steps, the preconditioner can be switched to *ebe*.

2.9.2 Dynamic Analysis Parameters

The time increment over each load step and the β factor for the Newmark time integration scheme are defined by the commands

```
time step  < number >
newmark beta < number >
```

The default time step size is 1000000 and the default value of the Newmark β factor is 1/4.

Static analyses in WARP are achieved by using a very large time step or by setting the model mass to zero. The time step must be a positive number. By setting a realistic time step for the analysis with a zero mass, analyses for viscoplastic effects may be performed without inertia effects.

2.9.3 Newton Iteration Parameters

The nonlinear solution in each load/time step is accomplished with a full Newton iterative procedure by default. The dynamic tangent stiffness is updated prior to each equilibrium iteration and at the beginning of the step. Newton iterations are numbered 1, 2, 3, ... where the increment of applied forces and imposed displacements comprise the load vector for iteration 1. During subsequent iterations, the load vector consists of the current (total) residual forces. Users may request use of the linear-elastic stiffness for the solution of iteration 1 with a command of the form

linear stiffness (for) iteration one $\left\{ \begin{matrix} \text{on} \\ \text{off} \end{matrix} \right\}$

This option enhances convergence when the incremental load during the step causes inelastic unloading. The default value is *off*.

Maximum Iteration Limit

The upper limit on Newton iterations is defined by

```
maximum iterations < integer >
```

The default limit is 10.

Minimum Iteration Limit

The minimum number of Newton iterations defaults to 2. This prevents the "extrapolated" displacement increments from being accepted as the solution (see Section 2.9.7). Such circumstances may develop due to insufficiently strict tolerances on the convergence tests. For linear analyses or a solution strategy with the displacement extrapolation option turned *off*, the minimum number of iterations may be set to 1.

```
minimum iterations < integer >
```

The default limit is 2.

Nonconvergent Solutions

By default, the program terminates execution if Newton iteration limit is reached without convergence. Users can request that program execution continue to the next load step with the command

$$\underline{\text{noncon}}\text{vergent } \underline{\text{solu}}\text{tions} \begin{Bmatrix} \underline{\text{stop}} \\ \underline{\text{cont}}\text{inue} \end{Bmatrix}$$

Convergence Tests

Four types of tests are available to assess convergence of the Newton iterations. Define the following quantities:

$\|\boldsymbol{R}_k\|$	Euclidean norm of the residual force vector for the model following solution of iteration k of the step
$max[abs\boldsymbol{R}_k^{(i)}]$	maximum (absolute) entry in the residual force vector for the model following solution for iteration k of the step (only active dof are considered)
$\|\boldsymbol{P}\|$	Euclidean norm of the total force vector applied to the model (includes reactions at constrained dof and inertia effects)
$\|\Delta\boldsymbol{u}_1\|$	Euclidean norm of the incremental displacement vector for the model computed during iteration 1 of the load step
$\|\Delta\boldsymbol{u}_k\|$	Euclidean norm of the incremental displacement vector for the model computed during iteration k of the load step
$max[abs\Delta\boldsymbol{u}_k^{(i)}]$	maximum (absolute) entry in the displacement vector for the model following solution for iteration k of the step
$\bar{q}$	the numerical average of all forces (absolute value) applied to the nodes including: internal element forces due to stresses, inertia forces, reaction forces and externally applied nodal/element forces

Using these quantities, the four convergence tests are defined as follows:

Test 1:	$\|\Delta\boldsymbol{u}_k\| \le (user\ tol/100) \times \|\Delta\boldsymbol{u}_1\|$
Test 2:	$\|\boldsymbol{R}_k\| \le (user\ tol/100) \times \|\boldsymbol{P}\|$
Test 3:	$max[abs\Delta\boldsymbol{u}_k^{(i)}] \le (user\ tol/100) \times \|\Delta\boldsymbol{u}_1\|$
Test 4:	$max[abs\boldsymbol{R}_k^{(i)}] \le (user\ tol/100) \times \bar{q}$

where $\|\ \|$ denote the Euclidean norm. Multiple convergence tests may be defined; convergence requires satisfaction of all tests. Tolerance values are specified in (%); thus, a user tolerance of 0.01 (%) is reasonably strict and often used. Tolerance values of 0.001-0.5 have been used successfully in various models. The user specified tolerance exerts a dramatic impact on the required number of Newton iterations and the total CPU time. Excessively tight tolerances do not provide real improvements in solutions.

Also note that these are *relative* tolerance tests and the choice of physical units affects the corresponding *absolute* tolerance. For example, a *user tol* of 0.01 that may be suitable for a problem with forces in units of *kips* might be absurdly stringent if the force units in the same problem are given in *pounds-force* instead. This has importance, for example, in fracture problems where the actual residual force values on nodes in the crack front region must be controlled carefully.

Commands to define the convergence tests are

where the test types parallel the four tests defined above. The command to define Tests 1 and 2 is:

$$\underline{\text{norm}} \begin{Bmatrix} \underline{\text{displ}}\text{acement} \\ \underline{\text{res}}\text{idual } (\underline{\text{load}}) \end{Bmatrix} \quad \underline{\text{tol}}\text{erance} \quad < \text{tolerance: number} >$$

$$\underline{\text{conv}}\text{ergence } (\underline{\text{tes}}\text{ts})\ [<\text{test type}>]$$

Similarly, command to define Tests 3 and 4 is:

$$\underline{\text{max}}\text{imum}\begin{Bmatrix}\underline{\text{displ}}\text{acement}\\ \underline{\text{res}}\text{idual } (\underline{\text{load}})\end{Bmatrix}\ \underline{\text{tol}}\text{erance}\quad < \text{tolerance: number} >$$

An example of convergence test commands is:

```
nonlinear analysis parameters
  maximum iterations 10
  convergence test norm res tol 0.01 maximum displ tol 0.01
  nonconvergent solutions continue
```

2.9.4 Adaptive Step Size Control

In a nonlinear analysis (static or dynamic), it is often difficult to estimate *a priori* the appropriate load step sizes which provide rapid convergence of the Newton iterations. WARP provides a simple facility to reduce automatically load step (and time step) sizes when the solution appears to be diverging or converging slowly. By default, the adaptive step size feature is not used.

The adaptive algorithm is very simple. When the user specified limit on Newton iterations is reached and the solution has not converged, the load step (and time step) is subdivided into four (4) equal increments and the solution for the load step restarted. Steps are not renumbered during this process so that output messages indicate four solutions of the same load step. The output messages indicate which *fraction* of the user specified load step is being analyzed, e.g., 0.25 to 0.5.

Material models may also request an immediate load step reduction when the adaptive solution strategy is enabled. State variable updating may experience convergence difficulties requiring a reduction in load step size.

In geometrically nonlinear analyses, unusually large displacement increments may lead to a zero or negative deformation Jacobian at Gauss points in elements. When this condition is detected during strain computation, and adaptive solution control is *on*, the solution processor terminates further computations and immediately reduces the load step size. When the adaptive option is *off*, the solution processor terminates execution of WARP.

If the solution does not converge in any one of the 4 subincrements, that subincrement is further subdivided into four more increments and the solution restarted. Only two such levels of step reduction are permitted; nonconverged solutions at the second level cause program termination. In many cases, the first level of step reduction is sufficient. In other cases, one or more of the 0.25 fractions must be subdivided to obtain convergence. The adaptive algorithm performs level two reduction only for the level one fractions that do not converge.

The command to control adaptive load step sizes is

$$\underline{\text{adap}}\text{tive } (\underline{\text{sol}}\text{ution})\begin{Bmatrix}\underline{\text{on}}\\ \underline{\text{off}}\end{Bmatrix}$$

When the adaptive procedure restarts the analysis for a load step or subincrement, it forces the first iteration to be resolved using the linear stiffness for the model. This is required since the current estimate for the solution at $n+1$ is not valid for use to recompute element matrices. The full Newton process resumes at the next iteration. WARP manager routines handle these processes automatically.

Adaptive load step control is strongly recommended for users attempting the nonlinear solution of new classes of problems until experience with the convergence characteristics are known. For parametric studies of problems with well known convergence characteristics, adaptive load step control should not be used as it often dramatically increases analysis run times (the code repeatedly learns what size steps converge!). Analyses run much faster when the user specifies load step sizes known to exhibit good convergence characteristics.

Non-Zero Constraints

When a load step is subdivided, the non-zero constraints (e.g., $\Delta u_{10} = 0.1$) imposed by the user are reduced by the same adaptive factors as the step load. The actual constraint values specified by the user and stored in program data structure are not modified. Rather, scaled values are imposed during the equation solving process.

2.9.5 Batch Status Messages

During solution of a large nonlinear problem in batch mode (e.g. on a Cray), it proves convenient to have occasional information about the progress of the solution (load step/iteration number, convergence rate, etc.) WARP provides an option to produce status messages independent of the normal (standard) output file for the job. A status file is updated after each equilibrium if each step. This file is named <structure id>.batch_messages. In Unix, users can invoke the *tail* command on this file during execution to examine the last few lines. The typical last few lines of this file can appear as:

```
newton convergence tests step: 100 iteration:   5 @ cpu:   43.6
-----------------------------------------------------------------
    completed fraction over step: 1.00000
    maximum residual force:    0.179549E+00 @ node:   1356
    test 2: norm of residual load vector:          0.13631E+01
            norm of total load vector:             0.23237E+01
            ratio*100:         58.66192
```

If the batch message file exists from a previous analysis, the new information overwrites the old file. By default, no batch message files are written. The command to control batch messages is

$$\underline{\text{batch}}\ (\underline{\text{messages}}) \begin{Bmatrix} \underline{\text{on}} \\ \underline{\text{off}} \end{Bmatrix}$$

2.9.6 CPU Time Limit

On some systems, batch jobs are executed with a user specified limit set on the CPU time for the job. If the WARP execution exceeds the CPU time limit, the program is aborted by the operating system and all results after the last written restart file are lost. Estimating the required CPU time for highly nonlinear problems may be very difficult, especially when similar problems have not been executed previously.

To help users with this problem, WARP provides its own cpu time limit feature. The user informs WARP of the allowable CPU time (in secs) for the job. At the beginning of the solution for load step $n+1$, WARP assumes that the solution time for the step is the same as the time required the solution of load step n. The total CPU time estimated to advance the solution through load step n+1 is computed using this procedure and compared to the user specified limit. If the estimated time exceeds 90% of the user limit, WARP writes a restart file named xxxxx_overtime_db for load step n and terminates the job (xxxxx denotes the structure name).

The command to control this option is:

```
cpu (time) (limit)  { on < limit: secs > }
                    { off                }
```

By default the cpu time limit feature is *off*.

2.9.7 Displacement Extrapolation

In nonlinear analyses, the use of an extrapolated displacement vector frequently enhances the convergence rate of the Newton iterations— especially for "smooth" responses in plasticity. The incremental displacements computed from the solution over step n-1 to n are scaled and applied to the model to start the iterative (Newton) solution from n to n+1. The displacement scaling factor is computed from the specified step multipliers for steps n (say f_n) and n+1 (f_{n+1}) as f_{n+1}/f_n. Alternatively, users may specify directly the multiplier value. Only one loading pattern (*constraints* do not count)is permitted in the step definition when the extrapolate option is in effect.

The extrapolated displacement vector is employed at the beginning of load step n+1 to compute a set of incremental nodal forces for application to the model during iteration 1. The strains/stresses/internal forces are updated for the extrapolated displacement vector but the material states are not retained for the next iteration. New nonlinear response (e.g., first time yielding) is prevented during this updating process.

When the non-zero constraints are modified during a displacement control analysis, the loading step multipliers must be modified accordingly by the user; otherwise the extrapolation ratio (f_{n+1}/f_n) is computed incorrectly. To illustrate, consider the following example. Non-zero constraints are specified to load the model. The dummy loading pattern and nonlinear loading are defined as above with step multipliers of 1.0 for load steps 1-10. After step 10, the user modifies the constraints to reduced the imposed increment (uniformly) by one-half, possibly to reduce the number of Newton iterations for convergence in subsequent steps. Load steps 11, 12, 13, ... must have a multiplier of 0.5 for correct extrapolation. In step 11, the extrapolation multiplier is 0.5/1.0=0.5 while in steps 12, 13, ... the multiplier again becomes 1.0.

The command to control displacement extrapolation is

```
extrapolate { on ( (multiply) (by) < scale factor: number > ) }
            { off                                              }
```

When the *multiply by* option is given, the user specified scale factor supercedes the computed scale factor.

Numerical experiments reveal significant improvements can be obtained in the Newton convergence rate for displacement controlled loading with minor or no effect for analyses conducted under load control. For this reason, *extrapolate on* is the system default. The *extrapolate* option is correctly processed when used with adaptive load step control. For simple linear, dynamic analyses, we recommend using *extrapolate off* to eliminate spurious iterations created by inaccuracies in the the extrapolation procedure.

2.9.8 Material Model Messages

The material models have built-in features to print status messages during stress update. An option is provided to suppress all such informative messages generated by material models. Messages about severe conditions in the material models are not suppressed with this option. For example, the material model may request an immediate load step reduction when adaptive load control is enabled. In such cases, the material model prints a message to this effect with the reason it requests a load step reduction. The command to control printing of informative material messages is:

material (messages) $\left\{ \begin{array}{c} \text{on} \\ \text{off} \end{array} \right\}$

Material messages are *on* by default.

2.9.9 Solution Status Messages

The nonlinear solution process for a step and iteration involves many processes such as stiffness update, strain update, stress update, convergence tests etc. WARP outputs messages indicating when these processes start-finish and the detailed results for convergence tests. For users familiar with the code, most all of these messages can be suppressed with the *show details* command which has the form:

show (details) $\left\{ \begin{array}{c} \text{on} \\ \text{off} \end{array} \right\}$

Detailed messages are *on* by default. This option has no effect on the batch message feature.

2.9.10 Residual Loads Printing

Residual forces at nodes may be printed during Newton's iterations to facilitate debugging of problems which exhibit unusual convergence. To request printing of residual loads, use the command

print residual (loads) (for) (iterations) < integer list >

Residual loads printing is *off* by default.

2.9.11 *B*-Bar Element Stabilization

The $\overline{\boldsymbol{B}}$ modification of the 8-node, trilinear element has the potential to introduce hourglass modes. Section 3.1.7 describes a simple procedure that can often suppress such modes. The user controls the amount of stabilization with the command

bbar (stabilization) (factor) <numr>

where the numerical value ranges from 0.0 (no stabilization) to 1.0 (no $\bar{\boldsymbol{B}}$). The default value for this factor is 0.0. Values not exceeding 0.10 are often used.

2.9.12 Consistent [Q] Matrix

The consistent tangent moduli for the incremental plasticity models (*mises, gurson*) include the so-called [Q] matrix for the finite strain formulation (see Section 1.9.4). This matrix most often enhances the convergence rate of global Newton iterations but there are occasionally instances when it slows convergence.

An option exists to omit the [Q] contribution under control of the user. The user controls this option with the command

consistent q(–matrix) $\left\{ \begin{array}{c} \underline{\text{on}} \\ \underline{\text{off}} \end{array} \right\}$

The default value is *on*.

2.10 Compute Requests

Solution For Load Steps

The nonlinear (and dynamic) solution for a series of one or more load steps is requested with the command

```
compute displacements (for) loading < nonlinear load id: label >
        (for) steps < integer list >
```

A comma may be used anywhere in the line for continuation. WARP compares the last step number solved against the list of steps provided. A list of steps for computation is generated from this process and computations initiated. For example, let steps 1–10 be analyzed in the first *compute* command. The second *compute* command requests computation for steps 20–25. WARP automatically inserts steps 11–19 into the list of steps for computation.

WARP verifies the data provided in this command for correctness, e.g., the nonlinear load must exist and the steps requested must be defined in that load step. When errors are encountered, the command is ignored and a new input line read.

Once this command is accepted and computations begin, the user cannot intervene in the solution process until the analysis for all steps in the list is completed.

Examples of *compute* commands are:

```
compute displa load test steps 1-20
compute displacements for loading crush for steps 15-30
```

Domain Integral (J)

Once the solution for a load step is available, a domain integral evaluation to compute the J–integral may be requested. The domain(s) for computation must be defined immediately prior to the *compute* request. Chapter 4 describes commands to define domains for J computation. The *compute* command has the form

```
compute domain (integral)
```

2.11 Output Requests

The output command provides computational results in three forms:

- printed output with page and column headers
- Patran (2.5 compatible) nodal result files in either binary or ASCII formats.
- Patran (2.5 compatible) element result files in either binary or ASCII formats.
- creation of a Patran neutral file (2.5 compatible) for the model (coordinates, incidences, constraints).

Output commands must be given immediately after completion of the solution for a load step. Once the solution for load step *n* converges, WARP immediately updates all internal variables to prepare for solution of step *n*+1; only results for load step *n* are available for output.

2.11.1 Printed Output

The command to request printed output has the form

$$\underline{\text{output}}\left[\left\{\begin{array}{c}\text{wide}\\ \text{eformat}\\ \text{precision}\end{array}\right\}\right] < \text{quantity: label} > (\text{for}) \left\{\begin{array}{c}\text{nodes}\\ \text{elements}\end{array}\right\} < \text{integer list} >$$

where

< quantity >	is *one* of the following *displacements, velocities, accelerations, strains, stresses, reactions*

The destination for printed output is the *current* output device specified by the user. The output device is either a disk file or the workstation display. The output file is declared using the standard output (i.e., the < file name) convention of Unix on the program invocation command or through the **output to <file>* command available in WARP (refer to Section 2.13 for a description of * commands).

By default, output routines which generate printed results format values to fit on an 8.5 in. x 11 in page oriented in portrait mode. The *wide* option permits extension of output up to 132 columns for eventual printing in the landscape orientation.

Numerical results are printed with an F12.6 format. An E12.5 format is requested with the *eformat* option. These *precision* option increases these fields to F26.16 and E26.16.

When a list of elements is specified for output of displacements, velocities, accelerations or internal forces, results are printed for the nodes of each element in the list (not the merged set of nodes for all elements in the list). Only lists of elements are permitted for output of strains and stresses. When the < integer list > of elements/nodes is omitted, the results are printed at *all* elements/nodes of the model.

Reactions are external forces required for equilibrium at constrained nodal dof; at unconstrained nodal dof, they are the remaining force unbalance due to nonlinear response and/or linear equation solving. These forces include the effects of inertia loading on the model. Separate algebraic sums of the *X, Y, Z* components of these forces are printed following the nodal results to assist in the checking of reactions.

All strain/stress quantities refer to the global Cartesian coordinate system for the model. The number of strain/stress items printed for each element and the number/location of the points with printed results are specified with element properties. For example, the ele-

(**S**) $\epsilon_x,\ \epsilon_y,\ \epsilon_z,\ \gamma_{xy},\ \gamma_{yz},\ \gamma_{xz}$

(**S**) $\epsilon_{eff} = \frac{\sqrt{2}}{3}\sqrt{(\epsilon_x - \epsilon_y)^2 + (\epsilon_y - \epsilon_z)^2 + (\epsilon_x - \epsilon_z)^2 + 1.5(\gamma_{xy}^2 + \gamma_{yz}^2 + \gamma_{xz}^2)}$

$I_1 = \epsilon_x + \epsilon_y + \epsilon_z$

$I_2 = \epsilon_{xy}^2 + \epsilon_{yz}^2 + \epsilon_{xz}^2 - \epsilon_x\epsilon_y - \epsilon_y\epsilon_z - \epsilon_x\epsilon_z$

$I_3 = \epsilon_x(\epsilon_y\epsilon_z - \epsilon_{yz}^2) - \epsilon_{xy}(\epsilon_{xy}\epsilon_z - \epsilon_{yz}\epsilon_{xz}) + \epsilon_{xz}(\epsilon_{xy}\epsilon_{yz} - \epsilon_y\epsilon_{xz})$

$\epsilon_1 \le \epsilon_2 \le \epsilon_3$ (Principal strain values)

l_1, m_1, n_1 (Cosines for direction 1)

l_2, m_2, n_2 (Cosines for direction 2)

l_3, m_3, n_3 (Cosines for direction 3)

(**S**)– value included with *short* output option. All values included with *long* option.

FIG. 2.3—*Strain values for output*

ment logical property *long* requests an extended set of strain/stress results at the output points. The additional quantities include principal values, maximum shear values, state variables supplied from the material models, etc. The *short* output option is the default. The location/number of strain points is specified with the element logical properties: *gausspts, nodpts* or *center_output*. Node point values are extrapolated from the Gauss point values. The center point values are numerical averages of Gauss point values. The default output location is *gausspts*. Figure 2.3 summarizes the element strain output quantities; Fig. 2.4 summarizes the element stress output quantities.

Several examples of output commands are

```
output wide eformat precision displacements for nodes 1-300 by 2

output stresses elements 900-1500 by 2 300-500

output accelerations for elements 20-40 100-300 by 3
```

2.11.2 Patran Compatible Result Files

The command to request Patran output files has the form

$$\underline{\text{output}}\quad \underline{\text{patran}} \begin{Bmatrix} \underline{\text{binary}} \\ \underline{\text{formatted}} \end{Bmatrix} \begin{Bmatrix} \underline{\text{nodal}} \\ \underline{\text{element}} \end{Bmatrix} \quad < \text{quantity: label} >$$

where

< quantity > is *one* of the following *displacements, velocities, accelerations, strains, stresses, reactions*

and *nodal* results are output by default. Both *binary* and *formatted* results are sequential files created with the Fortran *open* statement and written with ordinary Fortran *write* statements.

The Patran compatible results files are assigned names that begin with four letters followed by the 5 digit load step number. Nodal result files begin with the letter *wn*, element result files with the letter *we*. The *n* and *e* letters are followed by the letter *b* for binary files or the letter *f* for formatted files. The fourth letter in the file name denotes the physical quantities: '*d*' – displacements, '*v*' – velocities, '*a*' – accelerations, '*r*' – reaction forces, '*e*' – strains and '*s*' – stresses. Note that element results files are available only for strains and stresses. For example, the file *webs00005* contains element stress results for step 5 in a binary file.

Figure 2.5 summarizes the data column assignments for Patran strain/stress results files. The first six strain/stress values that appear at each model node in the Patran nodal results files are the numerical average for the contribution of each element at the node. The strain-stress invariants, principal values and directions are computed from these averaged nodal values. The effective strain, Mises equivalent stress, energy density and the three material model state variables are first extrapolated from the Gauss points to the nodes and then averaged.

Element result files for strains and stresses adopt the same column assignments listed in Fig. 2.5 for nodal result files. A single set of values given for each element is obtained by simple averaging of Gauss point values within each element. The first six strain/stress values that appear for the element are the numerical average for the contribution of each Gauss point. The strain-stress invariants, principal values and directions are computed

(*S*) $\sigma_x,\ \sigma_y,\ \sigma_z,\ \sigma_{xy},\ \sigma_{yz},\ \sigma_{xz}$

(*S*) $U_0 = \int_0^{\epsilon} \sigma \, d\epsilon$ (Work density)

(*S*) $\sigma_{vm} = \frac{1}{\sqrt{2}}\sqrt{(\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_x - \sigma_z)^2 + 6(\sigma_{xy}^2 + \sigma_{yz}^2 + \sigma_{xz}^2)}$

c_1, c_2, c_3 (State variables from material model)

$I_1 = \sigma_x + \sigma_y + \sigma_z$

$I_2 = \sigma_{xy}^2 + \sigma_{yz}^2 + \sigma_{xz}^2 - \sigma_x\sigma_y - \sigma_y\sigma_z - \sigma_x\sigma_z$

$I_3 = \sigma_x(\sigma_y\sigma_z - \sigma_{yz}^2) - \sigma_{xy}(\sigma_{xy}\sigma_z - \sigma_{yz}\sigma_{xz}) + \sigma_{xz}(\sigma_{xy}\sigma_{yz} - \sigma_y\sigma_{xz})$

$\sigma_1 \le \sigma_2 \le \sigma_3$ (Principal stress values)

l_1, m_1, n_1 (Cosines for direction 1)

l_2, m_2, n_2 (Cosines for direction 2)

l_3, m_3, n_3 (Cosines for direction 3)

(*S*)– value included with *short* output option. All values included with *long* option.

FIG. 2.4—*Stress values for output*

from these averaged values. The effective strain, Mises equivalent stress, energy density and the three material model state variables are the average of Gauss point values.

The MacNeal-Schwindler Corporation (developers of Patran) publishes specifications for the formatted and binary structures of these results files. Appendix A provides skeleton Fortran programs to read the binary and formatted forms of the nodal results files. WARP3D generated (ascii) versions of these files are compatible with Patran versions 2.x, 3.x and 5.x.

Please note the following:

- nodal results files contain result values *only* at model nodes. Strains and stresses are nodes are obtained using a two step process: (1) extrapolation of integration point values to element nodes and then (2) numerical averaging of all nodal values.
- invariants, principal values and direction cosines are computed using the averaged nodal results for the strain and stress components
- the effective strain (e_{eff}), *mises* effective stress (σ_{vm}), work density (U_0) and material model state variables (c_1, c_2, c_3) are the extrapolated and then averaged nodal values
- it is not possible, at present, to specify a list of nodes to appear in the Patran results files. Results are written for all nodes in the model.

With the release of Patran3 and subsequent versions, "template" files are employed by Patran to match the data columns in the results files with symbolic names for the physical quantities. This simplifies user interactions with the results processing features available in Patran (users no longer need to remember column numbers!). To support post–processing of WARP results in Patran, we provide a set of such "template" files. These are included in the WARP distribu–tion and should be copied into the appropriate Patran directory on your computer (usually the res_templates directory of the Patran installation). While executing Patran, the user then selects a set of result templates to use before importing the results files.

2.11.3 Patran Compatible Neutral File for Model

The command to request generation of a Patran neutral file has the form

```
output  patran  neutral  (<name of file: label or string>)
```

If a name for the neutral file is omitted the default name is "structure_name".neutral (e.g. beam.neutral). When the specified filename already exists, the current time (hr:min:sec) is appended to the filename (e.g. beam.neutral_12:00:01).

The following information about the model is included in the neutral file:

- neutral file header records required by Patran
- number of nodes and elements
- coordinates of all model nodes
- incidences for all model elements. The 8 and 20–node elements are defined as HEX/8/#, or HEX/20/# where the Patran configuration code (#) for the element is assigned the material number given to the element during WARP3D input. For the 9, 12 and 15 node transition elements, the elements are defined as HEX/20/# but with zeroes in the incidence lists for the missing mid-side nodes. The Patran configuration code (#) for the element is assigned the material number given to the element during WARP3D input.
- constraints (zero and non–zero) imposed on model nodes

At present, no loading information is written to the neutral file.

Data Column	Strain Value	Data Column	Stress Value
1	ϵ_x	1	σ_x
2	ϵ_y	2	σ_y
3	ϵ_z	3	σ_z
4	γ_{xy}	4	σ_{xy}
5	γ_{yz}	5	σ_{yz},
6	γ_{xz}	6	σ_{xz}
7	ϵ_{eff}	7	U_0
8	I_1	8	σ_{vm}
9	I_2	9	c_1
10	I_3	10	c_2
11	ϵ_1	11	c_3
12	ϵ_2	12	I_1
13	ϵ_3	13	I_2
14	l_1	14	I_3
15	m_1	15	σ_1
16	n_1	16	σ_2
17	l_2	17	σ_3
18	m_2	18	l_1
19	n_2	19	m_1
20	l_3	20	n_1
21	m_3	21	l_2
22	n_3	22	m_2
		23	n_2
		24	l_3
		25	m_3
		26	n_3

FIG. 2.5—*Column numbers for strain–stress results in Patran data files*

2.12 Analysis Restart

Create A Restart File

To maintain the highest possible performance, WARP allocates all data structures in memory during execution and does not use databases on magnetic disk to temporarily hold (*swap*) data arrays. At completion of load step *n*, the user may request creation of a binary (sequential) file of data arrays required to restart execution at that point in the solution. The default form of the *save* command is

save (structure) (< structure id: label >)

where the structure id is optional. If omitted, the last specified structure name is used. The data file created with this command has the name @_db where @ denotes the first 8 characters of the structure id.

An explicit name for the restart file may be specified with the command

save (to) file < file name: label *or* string >

where a <string> must be used if the file name starts with/or contains special characters. The optional phrase *structure* may also be included in this command to maintain readability.

Examples of the *save* command are

```
save
save structure cylindrical_bar
save to file bar_step_450
save to file '452_model_bar'
save structure bar to file '325_bar'
```

Restart file sizes increase with the model size and solution characteristics. For example, a 7200 node, 5700 element model using a large displacement formulation and the rate–dependent Mises model requires 52 MB of space for each restart file on a Cray.

In a typical analysis, the solution is advanced 10 to 50 load steps then a new restart file is requested. WARP can be executed later to output results for the load step in a restart file. The explicit naming feature enables creation of a series of unique restart files at various points in the analysis.

Access A Restart File

To restart execution of WARP, the first (non–comment) command must be

retrieve (structure) (< structure id: label >)

or using an explicit name for the restart file

retrieve (from) file < file name: label *or* string >

where a <string> must be used if the file name starts with/or contains special characters. The optional phrase *structure* may also be included in this command to maintain readability.

Once the restart file is opened and read into memory, WARP displays a message indicating the load step number *n* for the restart file and the time completed in the analysis (useful for dynamic analyses). Commands to request output, to analyze additional load steps, etc. may then be given as usual.

2.13 Utility (*) Commands

Several utility commands are provided to manipulate input–output files, to control command echo, etc. Each of these commands begins with an * and these commands may be given at any time during input.

**** Echo Command***

The * *echo* command controls the "echoing" of input commands to the current output device. By default, all commands are echoed. The * *echo* command has the form

$$* \ \underline{\text{echo}} \left\{ \begin{array}{l} \underline{\text{on}} \\ \underline{\text{off}} \end{array} \right\}$$

**** Input Command***

The * *input* command controls the location from which input commands are read for processing. By default, the input stream is the user's interactive display or the Unix *stdin* device. The input stream can be switched to a disk file or switched back to the interactive display

$$* \ \underline{\text{input}} \ (\underline{\text{from}}) \left\{ \begin{array}{l} \underline{\text{display}} \\ (\underline{\text{file}}) < \text{file name: label or string} > \end{array} \right\}$$

where the <string> form is required with file names not meeting the definition of a <label>. * *input from file* ... commands may be contained within referenced input files to create an input "stack" up to 10 levels deep. When an end–of–file condition is reached on the current file, the stack is popped to again read from the previous file. When reading of the last file completes, the input stream returns to the user's display. In a batch job, the program is terminated by the WARP command processor if an end–of–file condition occurs at the highest level.

**** Output Command***

The * *output* command controls the location (stream) to which usual WARP output is directed. By default, the output stream is the user's interactive display or the Unix *stdout* device. The output stream can be switched to a disk file or switched back to the interactive display

$$* \ \underline{\text{output}} \ (\underline{\text{to}}) \left\{ \begin{array}{l} \underline{\text{display}} \\ (\underline{\text{file}}) < \text{file name: label or string} > \end{array} \right\}$$

where the <string> form is required with file names not meeting the definition of a <label>.

**** Time Command***

The * *time* command outputs the elapsed CPU time ins seconds for the current job.

$$* \ \underline{\text{time}}$$

**** Reset Command***

When the WARP command processor interprets the command stream, errors of various types may be detected. When errors are encountered, the command processors set an internal flag .true. to prevent a *compute* command from attempting a solution. This internal flag can be set to the "no error" condition with the ∗ *reset* command, which has the form

```
        * reset
```

Chapter 3

Elements and Material Models

This chapter describes the elements and material models currently available. The formulations and computational procedures unique to the elements/models are outlined in detail sufficient for their proper use.

3.1 Solid Elements: *l3disop, ts9isop, ts12isop, ts15isop, q3disop*

These isoparametric elements provide the fundamental modeling capability in Warp3D. The 8-node element (*l3disop*) employs a conventional tri–linear displacement field. With the $\bar{B}$ modifications of Hughes [40], the element exhibits minimal volumetric locking under fully incompressible material response and exhibits slightly improved bending response. This element performs well under finite deformations encountered, for example, near severe discontinuities and near crack fronts. A simple stabilization scheme may be invoked should hourglassing modes appear (infrequently experienced in this elements). Unfortunately, the element exhibits shear locking when subjected to very strong bending fields.

The 20-node element provides a quadratic displacement field with the ability to model crack front singularities in a focused mesh with element shapes collapsed into wedges. With a reduced order of Gauss quadrature, this element accurately resolves strong bending fields without shear locking; moreover, the reduced integration order also eliminates volumetric locking under fully plastic deformation. Thin shell structures are modeled accurately with just one element through the thickness when combined with the reduced integration—unless the analysis requires precise resolution of through thickness yielding. In such cases multiple elements defined through the thickness locate Gauss points nearer the outer surfaces.

The 9, 12 and 15 node elements have selected "quadratic" edges which enable transitions between the 8 and 20-node elements while maintaining full displacement compatibility. If Patran is used to create the model, the *patwarp* program will convert user-defined 8-node elements into transition elements based on shared faces/edges between 8 and 20-node elements in the model.

The element formulations support geometrically nonlinear analysis (large displacements, rotations, finite strains), materially nonlinear analysis and combined geometric/material nonlinear analysis.

For dynamic analyses, the diagonal (lumped) mass matrix derives from the scaled terms of the consistent mass matrix.

All element computations take place in the global coordinate system for the model. Strains and stresses output by the model reference the global coordinate axes.

For modeling initially sharp crack fronts, these elements are frequently degenerated or collapsed into a wedge shape. While this modeling technique causes no problems for a small–strain analysis, difficulties in Newton convergence of the global solution can be experienced when the collapsed elements have the geometric nonlinear formulation. The remedy is to model the crack front as a very small tube (i.e. a *keyhole*) or to model the crack tip as an initially blunt notch with a root radius very small compared to the crack length or remaining ligament length.

3.1.1 Node and Integration Point Ordering

Figure 3.1 shows the ordering of nodes for the elements and the orientation of parametric axes (ξ, η, ζ). The 9, 12, 15 and 20-node elements retain the same numbering for the 8 corner nodes as defined for the *l3disop* element. The node ordering for the 20 node element follows that used in Abaqus for the same element.

These elements have mass, stiffness and internal forces computed using quadrature. Figure 3.2 tabulates the locations of integration points in parametric coordinates. The 8-node element is always evaluated using a conventional $2 \times 2 \times 2$ Gauss quadrature. The same $2 \times 2 \times 2$ is defined as the default order for the 9, 12, 15 and 20-node elements. For these elements, two additional integration rules are provided as options: a 9-point rule and a 14-point rule. The 9 and 14 point rules are especially useful for the 20-node brick to suppress hourglassing modes. Fracture mechanics models constructed with 20-node elements tend to develop hourglassing modes in those elements located on the crack plane just behind the front. Hourglass modes can also appear in those elements with multiple "free" faces, especially those subjected to applied loading. The 9 point and 14 point rules offer a remedy for the modes but at the cost of increased computation time and memory storage.

To eliminate potential errors due to strongly varying element shapes, the mass matrix and equivalent forces for applied body forces are evaluated with the 14-point rule for the 9, 12, 15 and 20-node elements. Similarly, the equivalent loads for applied face tractions are evaluated using a 3×3 Gauss rule. These procedures are applied independent of the integration order specified by the user. The user specified integration order is applied for stiffness and internal force computations and for strain-stress updating.

Element results are frequently output at the "center point" which corresponds to parametric location (0,0,0).

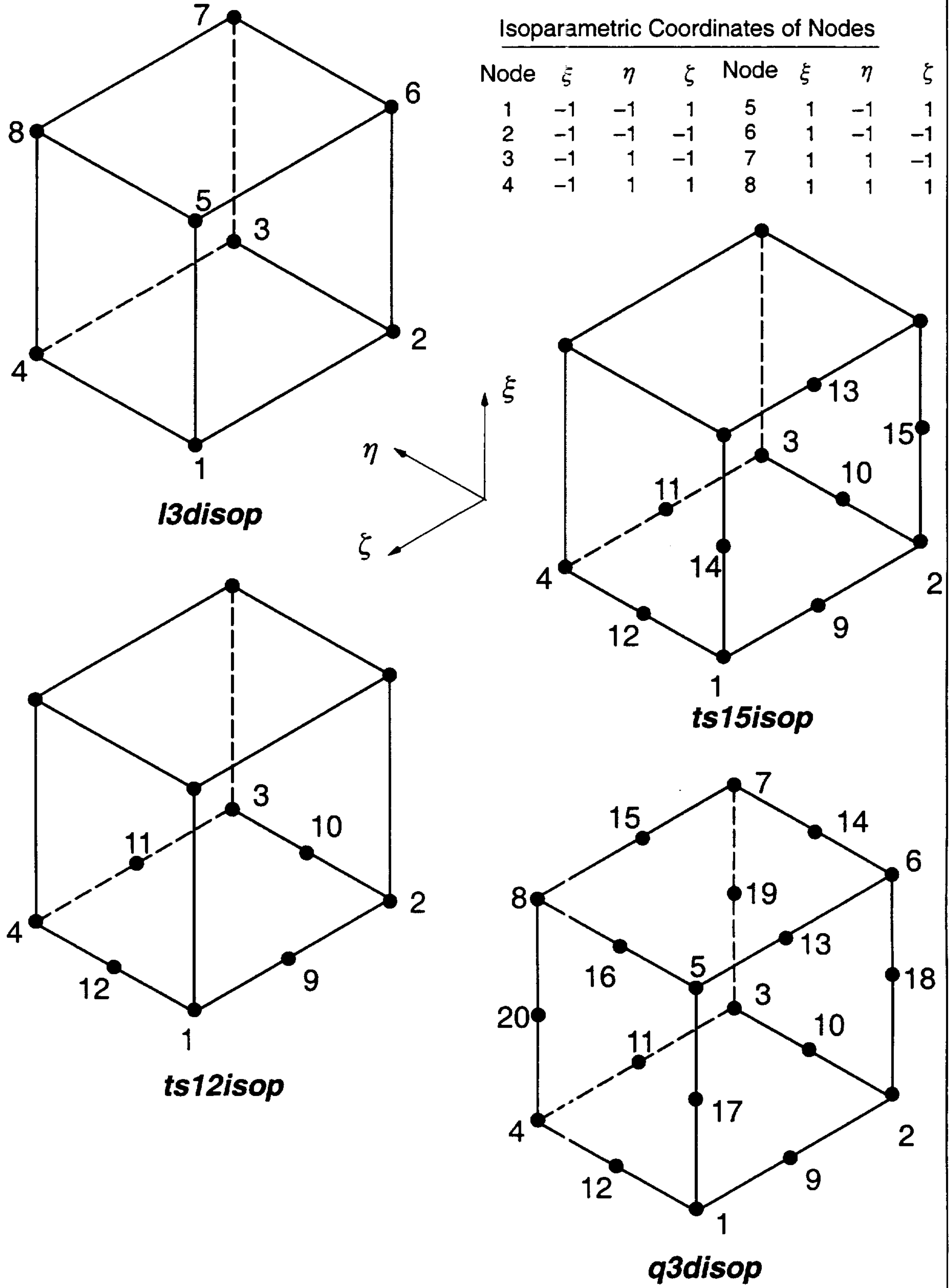
Isoparametric elements provide a powerful capability to model the geometry of irregularly shaped bodies. The parent element in parametric coordinates is mapped into the global Cartesian space using (current) coordinates of the nodes and the linear interpolation functions. The element behavior remains adequate unless the mapped shape becomes unreasonable (either the initial, undeformed shape or the current shape if geometric nonlinear analysis). Corner angles on each face must be $>0°$ and $<180°$. The best element response is obtained for angles within the range $90° \pm 30°$. Large aspect ratios should be avoided if possible. The best element behavior derives from a cubical shape; however, rectangular prism shapes with aspect ratios of 10–20 are commonly used without undue loss of accuracy, especially if the strain field varies gently in the "long" direction.

Element routines check for badly distorted elements by examining the determinant of the coordinate Jacobian at the integration points (using the current nodal coordinates for geometric nonlinear analysis). Zero or negative values indicate a severely distorted element. Messages identifying these problems are printed with information about the element.

For edges of elements having mid-side nodes, the nodes must be located intitially within a narrow range from the geometric center of the element edge.

3.1.2 Element Properties

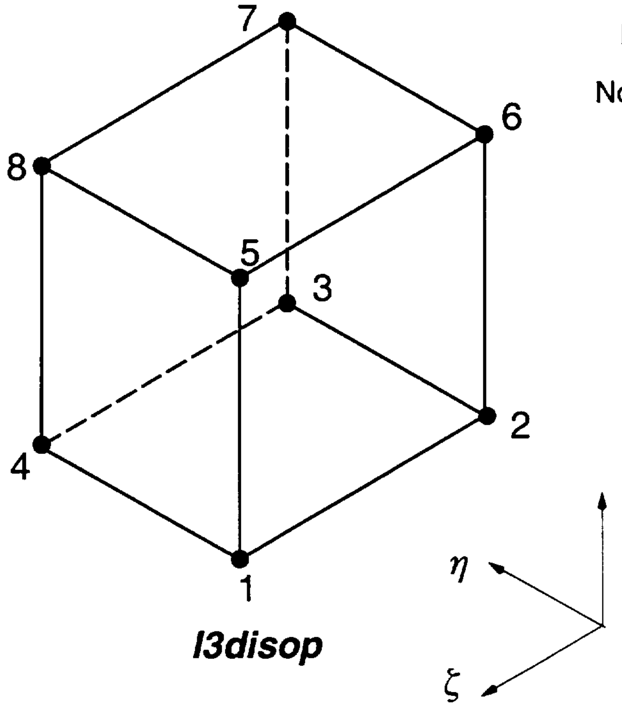
Table 3.1 summarizes the user–assignable values that control element behavior. Element properties are defined by the name of the property, a < label >, followed by a value. Logical



Isoparametric Coordinates of Nodes

Node	ξ	η	ζ	Node	ξ	η	ζ
1	−1	−1	1	5	1	−1	1
2	−1	−1	−1	6	1	−1	−1
3	−1	1	−1	7	1	1	−1
4	−1	1	1	8	1	1	1

FIG. 3.1—*Local node ordering for the isoparametric solid elements. Isoparametric coordinates for the element corner nodes are listed.*



Isoparametric Coordinates of Nodes

Node	ξ	η	ζ	Node	ξ	η	ζ
1	–1	–1	1	5	1	–1	1
2	–1	–1	–1	6	1	–1	–1
3	–1	1	–1	7	1	1	–1
4	–1	1	1	8	1	1	1

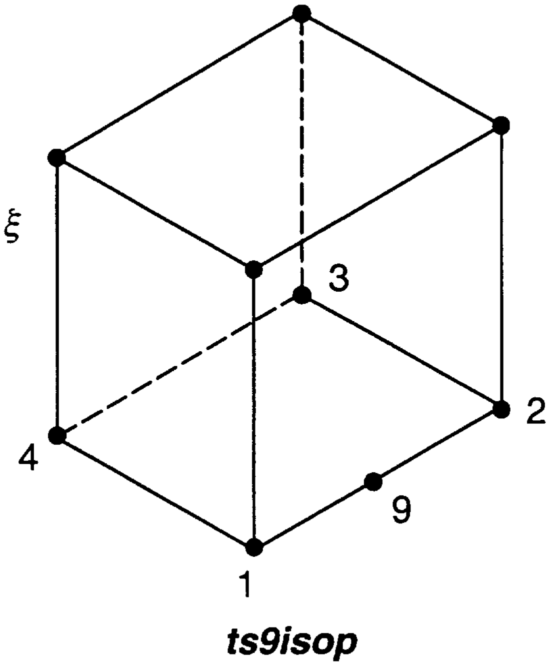
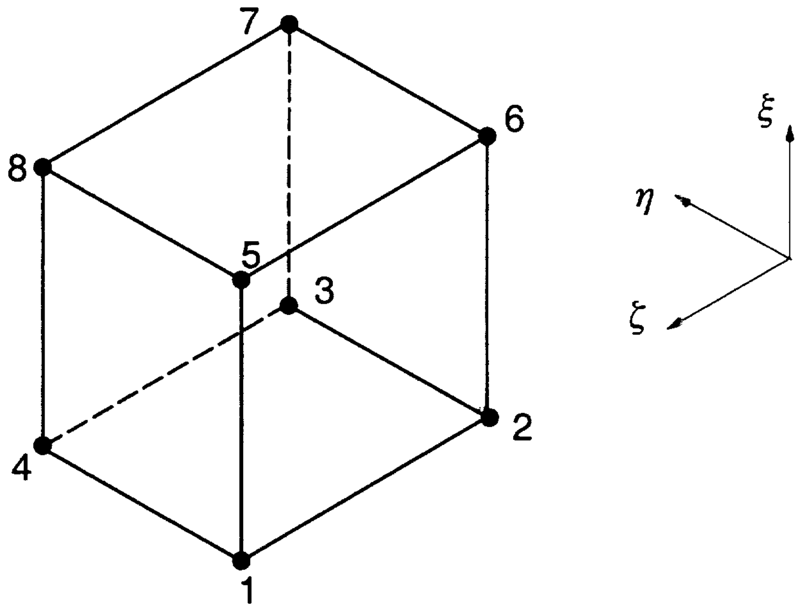


FIG. 3.1—*Local node ordering for the isoparametric solid elements. Isoparametric coordinates for the element corner nodes are listed.*

properties are set .true. simply by the appearance of the property name. The default behavior for the *l3disop* element is this: small–strain formulation, $2 \times 2 \times 2$ Gauss integration, $\overline{B}$ formulation, and output of a short list of strains–stresses at the Gauss points. For the *ts9isop, ts12isop, ts15isop* and *q3disop* elements, the default behavior is this: small–strain formulation, $2 \times 2 \times 2$ Gauss integration, and output of a short list of strains–stresses at the Gauss points.



$2 \times 2 \times 2$ and 9pt_rule

Coordinates of Gauss Pts.

Point	ξ	η	ζ
1	–a	–a	–a
2	–a	a	–a
3	a	–a	–a
4	a	a	–a
5	–a	–a	a
6	–a	a	a
7	a	–a	a
8	a	a	a
9	0	0	0

a=0.57735

14pt_rule

Coordinates of Pts.

Point	ξ	η	ζ
1	–a	0	0
2	a	0	0
3	0	–a	0
4	0	a	0
5	0	0	–a
6	0	0	a
7	–b	–b	b
8	–b	–b	–b
9	–b	b	–b
10	–b	b	b
11	b	–b	b
12	b	–b	–b
13	b	b	–b
14	b	b	b

a=0.795822 b=0.758787

FIG. 3.2—*Location of integration points in isoparametric coordinates.*

Element Property	Keyword	Mode	Default Value
Geometrically *linear* formulation	*linear*	Logical	True
Geometrically *nonlinear* formulation	*nonlinear*	Logical	False
Material associated with element	*material*	Label	none
Order of Gauss integration	*order*	String	$2\times2\times2$[†]
Use $\bar{B}$ formulation	*bbar*	Logical	True[*]
Do not use $\bar{B}$ formulation	*no_bbar*	Logical	False[*]
Output strains–stresses at Gauss points	*gausspts*	Logical	True
Output strains–stresses at element nodes	*nodpts*	Logical	False
Output strains–stresses at (0,0,0) in element	*center_output*	Logical	False
Output minimal set of strain–stress values	*short*	Logical	True
Output full set of strain–stress values	*long*	Logical	False

[†]9pt_rule and 14pt_rule available for 9, 12, 15, 20-node elements

[*]$\bar{B}$ is not available for 9, 12, 15 and 20-node elements

Table 3.1 Properties for *l3disop, ts9isop, ts12isop, ts15isop* and *q3disop* elements

3.1.3 Output Options

Printed strain–stress results may be obtained at the integration points (default), the element nodes or at the parametric centerpoint of the element (0,0,0). Figures 2.3 and 2.4 define each of the strain–stress values output by the element.

When the $2\times2\times2$ integration rule is specified, nodal values of σ_{ij}, ϵ_{ij} are computed by extrapolation of Gauss point values using linear, Lagrangian polynomials. When other integration orders are specified, the nodal values are defined simply as the mean values of the integration point values since no clear extrapolation procedure exists (values at all nodes of an element are identical). Values of invariants, principal values and directions are computed from these extrapolated nodal values. Values of effective strain, Mises stress, energy density and state variables dependent on the material model are simply extrapolated to element nodes.

The centerpoint values of σ_{ij}, ϵ_{ij} are the simple numerical average of Gauss point values. Values of invariants, principal values and directions are computed from these averaged, centerpoint values. Values of effective strain, Mises stress, energy density and state variables dependent on the material model are simply are simply averaged.

The *short* option requests printing of a reduced set of output values. The invariants, principal values and direction cosines are omitted. This is the default output option.

3.1.4 Mass Formulation

The element (diagonal) mass matrix is evaluated once at the start of computations for the first load step. Entries of the lumped mass are proportional to the diagonal entries of the element consistent mass. The proportionality factor is defined to preserve the total mass of the element, e.g., the sum of the diagonal terms for the $\ddot{v}_i$ accelerations equals the element mass. This procedure always generates positive values for the lumped mass and leads to optimal convergence rates with mesh refinement.

The element mass matrix for analysis is thus given by

$$m^e_{pq} = \begin{cases} \alpha\delta_{ij} \int_{V_e} \rho N_a N_b \, dV_e & a = b \\ 0 & a \neq b \end{cases} \tag{3.1}$$

where ρ denotes the mass density of the undeformed material. N_a denotes the usual linear interpolating functions for the element node a. The scaling factor α is given by

$$\alpha = \underbrace{\int_{V_e} \rho \, dV_e}_{\textit{total element mass}} \bigg/ \underbrace{\left[\sum_{a=1}^{nn} \int_{V_e} \rho N_a^2 \, dV_e \right]}_{\textit{sum of diagonal entries of consistent mass}} \tag{3.2}$$

where nn denotes the number of element nodes. As noted previously, a full integration order is emplyed to evaluate these integrals.

3.1.5 Element Loads

Loads available for these elements include body forces, face tractions, face pressures and uniform temperature changes. Imposed nodal temperatures can generate a non-uniform temperature field over the element. Nodal and element temperature loads may be active simultaneously and can be mixed with other types of element loadings. A sequence of element load definitions has the form

```
element (loads)
  < elements: list >  < type of element loading >
  < elements: list >  < type of element loading >
                      •
                      •
```

where the <type of element loading> is a body force, a face traction or a temperature.

Body Forces

Body forces are specified by the intensity (units of F/L^3) and the direction along one of the coordinate axes. The body force intensity is constant over the element. The body force loads are defined by the command

Face Tractions

Tractions applied to the faces of elements may have a direction along one of the global coordinate axes or a direction normal to the specified face. Figure 3.3 defines the face numbers.

$$\underline{\text{body}}\ (\underline{\text{forces}}) \left[\left\{ \begin{matrix} \text{bx} \\ \text{by} \\ \text{bz} \end{matrix} \right\} (=)\ <\text{force intensity: number}>\ (,) \right]$$

The commands define the loaded face of the element, the loading intensity (units of F/L^2), and the loading direction. When the traction is aligned with one of the coordinate axes, the command has the form

$$\underline{\text{face}}\ <\text{face number: integer}>\ (\underline{\text{trac}}\text{tions}) \left[\left\{ \begin{matrix} \text{tx} \\ \text{ty} \\ \text{tz} \end{matrix} \right\} (=)\ <\text{intensity: number}>\ (,) \right]$$

For a normal (pressure) loading, use a command of the form

$$\underline{\text{face}}\ <\text{face number: integer}>\ \underline{\text{press}}\text{ure}\ \ (=)\ <\text{intensity: number}>$$

where a positive value for the intensity denotes a load directed into the face, i.e., a positive pressure loads the face in compression.

Uniform Temperature

A uniform temperature change over the complete element may be imposed through element loads.The command has the form

Element loads are additive; if the same element and direction appear in two different loading commands the sum of two loads is applied to the model. An example sequence to define a loading condition and a set of element loads is

```
loading one
  element loads
     1-40 620-800 by 2 face 6 pressure -2.1 temperature 32.4
     140 face 3 tractions tx -0.5 ty 14.34 tz 42.6
     3256-4000 body forces bz 12.3 bx -32.4
     20 body force bx -3
```

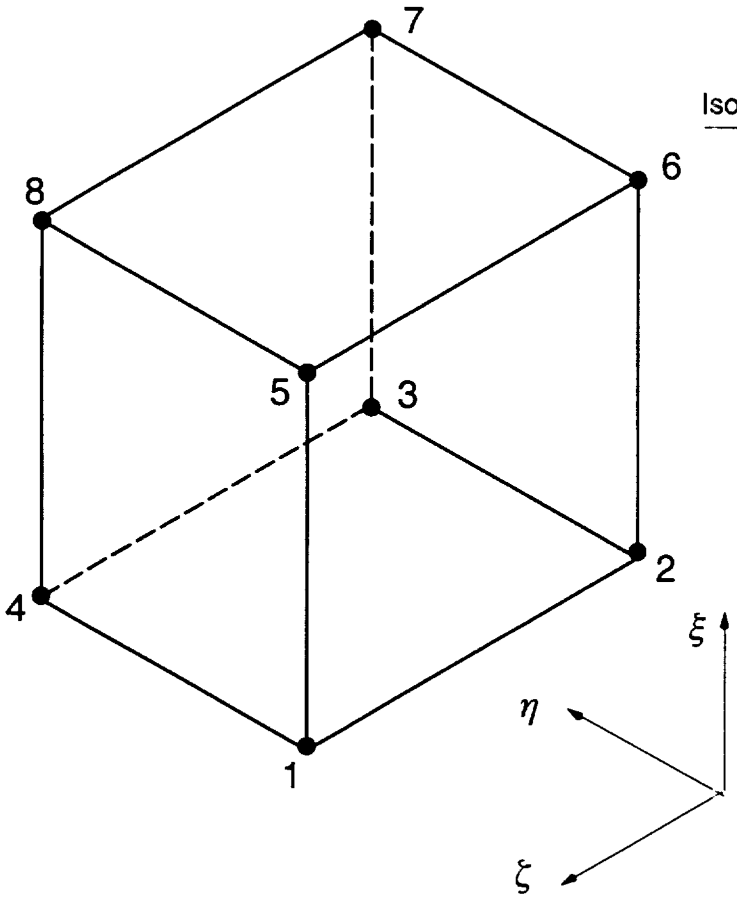
In the above example, element 20 has both a normal face pressure on face 6 and body forces in the x and y directions. Specifications for different loading types for a list of elements may be combined onto a single line if desired.

Large Displacement Effects on Loads

When the analysis includes geometric nonlinear effects (large displacements), equivalent loads for the incrementally applied surface tractions are re–computed at the beginning of each load step using the current (deformed) geometry of the elements.

3.1.6 Strains–Stresses for Geometric Nonlinear Formulation

The *nonlinear* property requests a geometrically nonlinear element formulation. Stress values output by the element are then the *Cauchy* (true) stresses. The Cauchy stress defines tractions over internal surfaces in the *deformed* configuration. The Cauchy stress components, σ_{ij}, are aligned with the global coordinate axes for the model. The symmetric Cauchy stress satisfies the equilibrium conditions



Isoparametric Coordinates of Nodes

Node	ξ	η	ζ
1	–1	–1	1
2	–1	–1	–1
3	–1	1	–1
4	–1	1	1
5	1	–1	1
6	1	–1	–1
7	1	1	–1
8	1	1	1

Face No.	Nodes
1	1–2–3–4
2	5–8–7–6
3	1–5–6–2
4	4–8–7–3
5	2–6–7–3
6	1–5–8–4

FIG. 3.3—*Face numbers for applying tractions to the isoparametric elements.*

temperature (=) < value: number > (,)

$$\int_S \boldsymbol{n} \cdot \sigma \, dS = \int_V \left(\frac{\partial}{\partial \boldsymbol{x}}\right) \cdot \sigma \, dV \qquad (3.3)$$

where $\boldsymbol{n}$ defines a unit outward normal to the deformed surface S, and V denotes the deformed volume of the body.

The increment of strain that advances the solution from load step n to n+1 is given by

$$\Delta\boldsymbol{\epsilon} = \overline{\boldsymbol{B}}(\boldsymbol{x}_{n+1/2})\Delta\boldsymbol{u} \qquad (3.4)$$

where the $\overline{\boldsymbol{B}}$ form the linear–strain displacement matrix is evaluated at the mid–step deformed configuration. This corresponds to a finite increment of the rate of deformation tensor, $\boldsymbol{D}_{n+1/2} \cdot \Delta t$, over the step. Converged increments of $\Delta\epsilon$ are summed over k load steps to define a measure of strain for output as

$$\epsilon = \sum_{n=1}^{n=k} \Delta\epsilon \tag{3.5}$$

where the shear strains follow the usual engineering definition, i.e., $\Delta\gamma_{xy} = 2 \times \Delta\epsilon_{xy}$.

The increment of strain $\Delta\epsilon$ is identified as the rate of logarithmic strain with respect to the current (deformed) configuration.

3.1.7 The $\overline{\boldsymbol{B}}$ Formulation

Many methods to alleviate locking which occurs in fully integrated elements have been proposed in the literature. The so–called $\overline{\boldsymbol{B}}$ method (Hughes [40]) implemented for the *l3disop* element in WARP3D is outlined below.

The strains are divided into deviatoric and dilatational parts in the following manner.

$$\epsilon_{ij} = \epsilon_{ij}^{dev} + \epsilon_{ij}^{dil} \qquad \epsilon_{ij}^{dil} = \tfrac{1}{3}\delta_{ij}\sum_{k=1}^{3}\epsilon_{kk} \qquad \epsilon_{ij}^{dev} = \epsilon_{ij} - \epsilon_{ij}^{dil} \tag{3.6}$$

The strain–displacement matrix, $\boldsymbol{B}$, is divided divided into a dilatational and deviatoric parts in the same manner as

$$\boldsymbol{B} = [\boldsymbol{B}_1 \boldsymbol{B}_2 \dots \boldsymbol{B}_n] \tag{3.7}$$

where

$$\boldsymbol{B}_i = \begin{bmatrix} B_1 & 0 & 0 \\ 0 & B_2 & 0 \\ 0 & 0 & B_3 \\ B_2 & B_1 & 0 \\ 0 & B_3 & B_2 \\ B_3 & 0 & B_1 \end{bmatrix} \qquad \boldsymbol{B}_i^{dil} = \frac{1}{3}\begin{bmatrix} B_1 & B_2 & B_3 \\ B_1 & B_2 & B_3 \\ B_1 & B_2 & B_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{B}_i^{dev} = \boldsymbol{B}_i - \boldsymbol{B}_i^{dil} \tag{3.8}$$

with the subscript i is omitted for clarity on each term inside the [] and

$$B_{i1} = \frac{\partial N_i}{\partial x} \qquad B_{i2} = \frac{\partial N_i}{\partial y} \qquad B_{i3} = \frac{\partial N_i}{\partial z} \tag{3.9}$$

The dilatational contribution to the stiffness causes locking for near incompressible conditions and is replaced the dilatational part of the strain-displacement matrix with a modified dilatational part, $\overline{\boldsymbol{B}}^{dil}$. The strain-displacement matrix is replaced by $\overline{\boldsymbol{B}}$ defined as:

$$\overline{\boldsymbol{B}}_i = \boldsymbol{B}_i^{dev} + \overline{\boldsymbol{B}}_i^{dil} \qquad \text{where } \overline{\boldsymbol{B}}_i^{dil} = \frac{1}{3}\begin{bmatrix} \overline{B}_1 & \overline{B}_2 & \overline{B}_3 \\ \overline{B}_1 & \overline{B}_2 & \overline{B}_3 \\ \overline{B}_1 & \overline{B}_2 & \overline{B}_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.10}$$

where again the subscript i on each term in the [] has been omitted. The $\overline{\boldsymbol{B}}$ matrix can then be written out explicitly in the following form (with subscript i omitted inside [])

$$\bar{\boldsymbol{B}}_i = \frac{1}{3}\begin{bmatrix} 2B_1 + \bar{B}_1 & \bar{B}_2 - B_2 & \bar{B}_3 - B_3 \\ \bar{B}_1 - B_1 & 2B_2 + \bar{B}_2 & \bar{B}_3 - B_3 \\ \bar{B}_1 - B_1 & \bar{B}_2 - B_2 & 2B_3 + \bar{B}_3 \\ 3B_2 & 3B_1 & 0 \\ 0 & 3B_3 & 3B_2 \\ 3B_3 & 0 & 3B_1 \end{bmatrix} \tag{3.11}$$

Mean Dilatation

Several options for defining $\bar{\boldsymbol{B}}_i^{dil}$ have been proposed in the literature. Here we use the "mean dilatation" approach suggested by Nagtegaal, et al. [63]. A volume averaged (mean) $\tilde{\boldsymbol{B}}_i$ matrix is computed over the element as

$$\tilde{\boldsymbol{B}}_i = \frac{1}{V_e}\int_{V_e} \boldsymbol{B}_i dV_e \tag{3.12}$$

with $\bar{\boldsymbol{B}}_i^{dil}$ at each Gauss point taken as the dilatational component of $\tilde{\boldsymbol{B}}_i$ as in Eq. (3.10). To save computations, only the three terms needed from $\tilde{\boldsymbol{B}}_i$ to compute $\bar{\boldsymbol{B}}_i^{dil}$ are actually evaluated

$$\bar{B}_{i1} = \frac{\partial \bar{N}_i}{\partial x} = \frac{1}{V_e}\int_{V_e} \frac{\partial N_i}{\partial x} dV_e \tag{3.13}$$

$$\bar{B}_{i2} = \frac{\partial \bar{N}_i}{\partial y} = \frac{1}{V_e}\int_{V_e} \frac{\partial N_i}{\partial y} dV_e \tag{3.14}$$

$$\bar{B}_{i3} = \frac{\partial \bar{N}_i}{\partial z} = \frac{1}{V_e}\int_{V_e} \frac{\partial N_i}{\partial z} dV_e \tag{3.15}$$

using the standard $2 \times 2 \times 2$ Gauss quadrature.

This formulation provides an element with the same dilatational strain and mean stress at each of the $2 \times 2 \times 2$ Gauss points. When plane strain constraints are imposed on the $\bar{\boldsymbol{B}}$ element, the ϵ_z is not restricted to 0 at each Gauss point, but is only restricted to 0 over the element as a whole, i.e., for *center_output* the ϵ_z value is zero.

Large Displacement Form

When large displacement effects are present, the current coordinates of the element nodes are adopted to form $\bar{\boldsymbol{B}}$ to compute virtual strains for internal force computation as in

$$\boldsymbol{IF}_e = \int_{V_e} \bar{\boldsymbol{B}}^T(\boldsymbol{x}_{n+1})\sigma_{n+1} dV_e \tag{3.16}$$

where σ denotes the Cauchy stresses and V_e the current (deformed) element volume at n+1.

Hourglass Stabilization

The $\bar{\boldsymbol{B}}$ modification which enforces constant pressure throughout the element can introduce spurious hourglass modes. A classic example which illustrates this element behavior involves finite compression of a plane-strain block, where the ends are restrained from lateral expansion. The differences between deformed shapes with and without the $\overline{\boldsymbol{B}}$ modification are quite surprising.

A simple stabilization procedured suggested by Nakamura et al. [67] often helps to suppress this behavior. A specified fraction of the usual $\boldsymbol{B}_i^{dil}$ replaces a similar fraction of $\overline{\boldsymbol{B}}_i^{dil}$ as

$$\overline{\boldsymbol{B}}_i = \boldsymbol{B}_i^{dev} + \overline{\boldsymbol{B}}_i^{dil} \quad + \quad \epsilon\left[\boldsymbol{B}_i^{dil} - \overline{\boldsymbol{B}}_i^{dil}\right] . \qquad (3.17)$$

When $\varepsilon = 0.0$, the full $\overline{\boldsymbol{B}}$ form of the element is obtained; when $\varepsilon = 1.0$ the conventional $\boldsymbol{B}$ matrix for the 8-node element results. No extra computational costs incurr for $\varepsilon > 0$.

Users specify the value of ε in the *nonlinear solution* parameters for the analysis with the command (see Section 2.9.11) *bbar stabilization factor <numr>*. The default value of ε is 0.0.

3.1.8 Example

The following example illustrates the specification of elements in a model.

```
structure cct
c
material a533b
  properties ....
c
number of nodes 25642 22092
c
elements
  14000-22092 type l3disop  nonlinear material a533b order 2x2x2,
                    long bbar center_output
  2000-4000 type q3disop  linear material a533b order 2x2x2,
                    long nodpts
c
```

3.2 Material Model Type: *deformation*

The flow or incremental theory of plasticity with a Mises yield surface has been extensively employed in elastic–plastic analyses. Alternatively, plasticity can be described by a deformation theory which assumes that the *strain path* at each material point remains linear (proportional) over the full range of loading. Deformation plasticity is essentially a nonlinear elasticity theory. For a proportional strain path, deformation and incremental plasticity theories provide identical results. Deformation plasticity *does not* correctly model the path–dependent behavior of materials for radical departures from proportional loading.

Deformation plasticity offers significant savings of computational effort compared to flow theory plasticity. Much larger load steps may be imposed on the model and only a few Newton iterations are needed for convergence at each load step. The number of computations performed in the material model is greatly reduced compared to a general incremental theory model since there is no explicit yield surface to complicate matters. Solutions tend to be very stable compared to those for flow theory especially when a region of the model contains large strain gradients, e.g., at a crack.

This model employs a representation of the uniaxial (tensile) stress–strain curve consisting of three parts: an initial, linear response followed by a small circular transition to a pure, power–law model.

The model supports only a small–strain formulation and rate–independent reponse. The assumptions of purely proportional loading in the model are questionable at best when finite strains and large rotations of material elements occur.

3.2.1 Formulation and Computational Procedures

The uniaxial stress–strain curve for the material is represented by the following relations (refer to Fig. 3.4):

$$\frac{\epsilon}{\epsilon_0} = \frac{\sigma}{\sigma_0} \qquad \text{for} \quad \frac{\sigma}{\sigma_0} \le K_1 \tag{3.18}$$

$$\frac{\epsilon}{\epsilon_0} = \epsilon_{Nc} - \sqrt{r_{Nc}^2 - \left(\frac{\sigma}{\sigma_0} - \sigma_{Nc}\right)^2} \qquad \text{for} \quad K_1 \le \frac{\sigma}{\sigma_0} \le K_2 \tag{3.19}$$

$$\frac{\epsilon}{\epsilon_0} = \left(\frac{\sigma}{\sigma_0}\right)^n \qquad \text{for} \quad \frac{\sigma}{\sigma_0} > K_2 \tag{3.20}$$

where,

ϵ_0	reference stress (yield stress)
σ_0	reference stress (yield stress)
n	hardening exponent for power–law region
K_1, K_2	lower, upper stress limits for transition
$\epsilon_{Nc}, \sigma_{Nc}$	center of circular transition arc
r_{Nc}	radius of transition arc

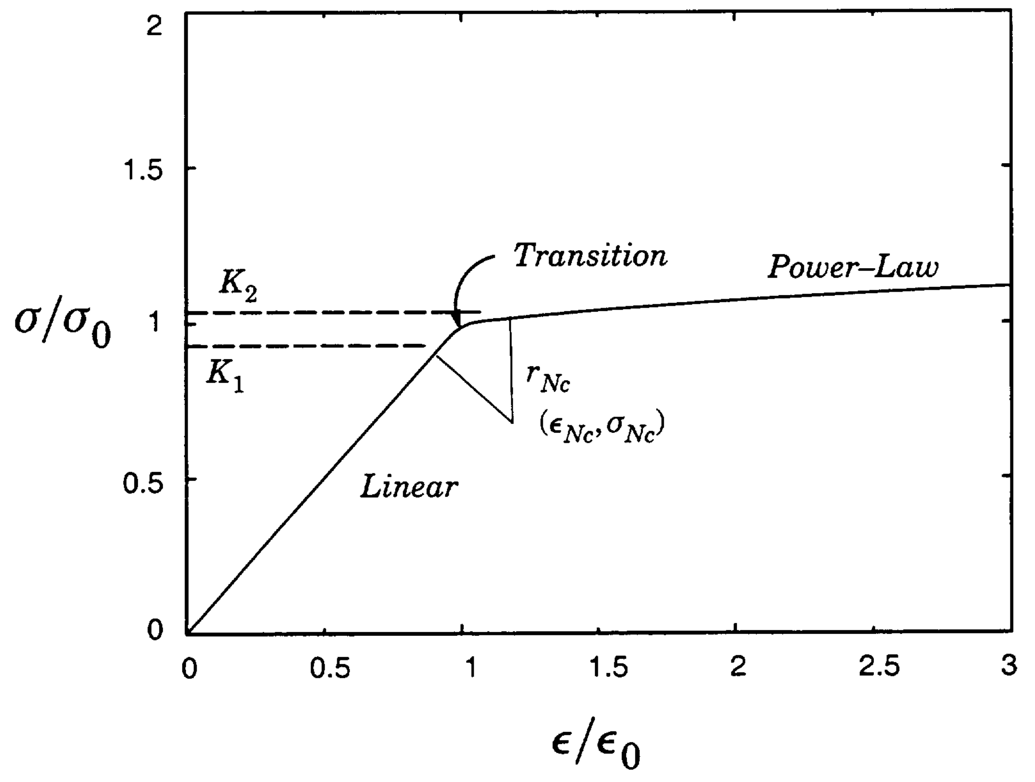


FIG. 3.4—*Uniaxial (tensile) stress–strain curve for the "deformation" plasticity material model.*

Given the linear limit, K_1, the model is able to compute the upper limit for the transition, K_2, based on the hardening exponent as well as the center of the small transitional arc and the corresponding radius. K_1 has the value 0.95.

Using an effective stress defined from the von Mises yield function and an effective strain defined from the Prandlt–Reuss relations, the total stress components in terms of the total strain components are given by:

$$\frac{\sigma_{ij}}{\sigma_0} = \frac{1}{3(1-2\nu)}\frac{\epsilon_{kk}}{\epsilon_0}\delta_{ij} + \frac{2}{3}\frac{\sigma_e/\sigma_0}{e_e/\epsilon_0}\frac{e_{ij}}{\epsilon_0} \tag{3.21}$$

where the effective stress and strain are defined by:

$$\sigma_e^2 = \frac{1}{2}\left[(\sigma_{11}-\sigma_{22})^2 + (\sigma_{22}-\sigma_{33})^2 + (\sigma_{33}-\sigma_{11})^2 + 6(\sigma_{12}^2 + \sigma_{23}^2 + \sigma_{13}^2)\right] \tag{3.22}$$

$$e_e^2 = \frac{2}{9}\left[(\epsilon_{11}-\epsilon_{22})^2 + (\epsilon_{22}-\epsilon_{33})^2 + (\epsilon_{33}-\epsilon_{11})^2 + \frac{3}{2}(\gamma_{12}^2 + \gamma_{23}^2 + \gamma_{31}^2)\right] \tag{3.23}$$

Full details of the formulation may be found in the Appendix of the thesis by Wang [89].

3.2.2 Model Properties

The properties defined for material model *deformation* are listed in Table 3.2. *Temperature loadings are not supported by this material model.*

Model Property	Keyword	Mode	Default Value
Young's modulus	*e*	Number	0.0
Poisson's ratio	*nu*	Real	0.0
Mass density	*rho*	Real	0.0
Reference yield stress (σ_0)	*yld_pt*	Number	0.0
Power law exponent (n)	*n_power*	Number	0.0

Table 3.2 Properties for *deformation* Material Model

3.2.3 Model Output

By default, the material model prints no messages during computations unless the numerical algorithms fail to converge. If requested, the material model prints the element number and strain point number whenever the effective stress *first* exceeds the specified yield stress. This option is requested with the nonlinear solution parameter *material messages on* (refer to Section 2.9.8)

The model makes available the exactly integrated strain energy density, U_0, to the element routines for subsequent output.

3.2.4 Computational Efficiency

The computational routines for this model process elements in blocks of a size matched to the vector length of the computer (i.e., Crays) or to the cache size of the workstation. All model computations are written in vectorized code except for the local Newton loop to update the scalar stress σ_e using the uniaxial stress-strain curve in Fig. 3.4. Compared to the general rate-dependent Mises model discussed later, this model is much faster. It is, however, slower than the fully vectorized Mises model with constant hardening described in the following section.

3.2.5 Example

The following example defines the properties for an A533B material frequently used in fracture models and assigns the material to some elements.

```
structure cct
c
material a533b
  properties deformation e 30000 nu 0.3 n_power 10 yld_pt 60.0,
      rho  7.3e-07
```

```
c
number of nodes 25642 22092
c
elements
  14000-22092 type q3disop  linear material a533b order 9pt_rule,
                    long
c
```

3.3 Material Model Type: *bilinear* (mises)

This material model extends the small-strain Mises plasticity theory to include the effects of finite strains and rotations. Rate-independent, incremental theory of plasticity with (constant) isotropic and kinematic hardening is employed with the von Mises yield surface expressed in terms of the Cauchy (true) stress. This model is formulated for the analysis of ductile metals which undergo large plastic strains but small elastic strains. By this we mean that the unloaded configuration obtained after significant plastic deformation is negligibly different from the deformed configuration. This assumption simplifies considerably the treatment of material elasticity and permits additive decomposition into elastic, plastic and thermal components of strain increments defined with respect to the deformed configuration.

The constitutive framework for WARP3D outlined in Chapter 1 neutralizes finite rotation effects during stress-update and computation of the consistent tangent moduli. The small-strain, stress-updating procedures follow a single-step, elastic-predictor radial-return algorithm. The algorithm is unconditionally stable for large strain increments and provides superb accuracy in the updated stresses (for a single step method). Inelastic unloading-reloading events are processed without difficulty. The last section provides an overview of the radial-return procedures implemented for this model.

This model employs a representation of the uniaxial (tensile) stress-strain curve consisting of an initial, linear response followed by linear hardening. Purely isotropic, purely kinematic and mixed isotropic-kinematic hardening are offered as options.

The model offers two types of properties for response to temperature changes imposed in the analysis: isotropic and anisotropic. Isotropic refers to the conventional description which uses the same thermal expansion coefficient (*alpha*) for each normal strain component with zero thermal strains generated for the shear components. The anisotropic model enables definition of a unique thermal expansion coefficient for each of the 6 strain components; it is intended to support modeling of various initial strain-stress fields, for example, residual stresses imposed through an eigenstrain approach. The isotropic model for thermal loading can be used in small displacement, large displacement and finite strain analyses. The anisotropic model for thermal loading should not be used in large displacement-finite strain analyses.

The *bilinear* model provides a very computationally efficient alternative to the general Mises model described in the next section when the rate-independent, constant hardening assumptions apply in the analysis. All computational steps of stress-updating and consistent tangent generation are vectorized. This model has the best computational performance.

The following sections describe needed parameters to utilize this material model. Full details of the numerical implementation are provided in the final section.

3.3.1 Stress-Strain Curve and Hardening Options

The uniaxial stress-strain curve for the material is represented by the linear hardening model shown in Fig. 3.5.

For a small-strain analysis (*linear* kinematic formulation), specify *engineering* values for the strain(ϵ_E) and stress (σ_E). For a finite-strain analysis (*nonlinear* kinematic formulation), specify the uniaxial stress-strain curve using the logarithmic strain, ϵ, and the true (Cauchy) stress, σ. *For a finite-strain analysis, the user should convert conventional engineering strain, ϵ_E, and engineering (nominal) stress, σ_E, values for input using the relations*:

$$\sigma = \sigma_E(1 + \epsilon_E) \tag{3.24}$$

$$\epsilon = \ln(1 + \epsilon_E) \tag{3.25}$$

The above conversions assume incompressible, homogeneous deformation. The true stress-true strain curve discussed here assumes homogeneous, uniaxial deformation of the material, i.e., prior to necking. Once necking occurs, the above expressions are no longer applicable. More elaborate corrections, for example those developed by Bridgeman, are required.

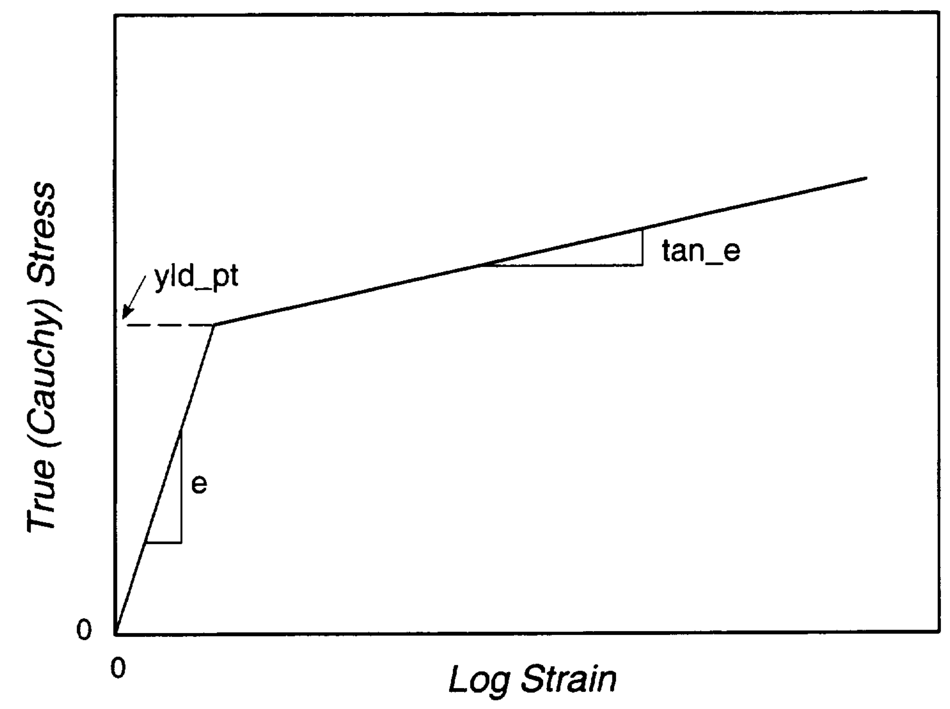


FIG. 3.5—*Uniaxial (tensile) stress-strain curve for the "bilinear" plasticity material model. For finite-strain analysis, input the Cauchy stress and log strain description.*

Once yielding begins, three strain hardening options are available. The strain hardening option is selected with the β (*beta*) model property. The rate of strain hardening is controlled by the user-specified tangent modulus, E_T, and the value of β. The strain hardening options are:

1. *Isotropic hardening* ($\beta = 1.0$). The radius of the yield surface increases in proportion to the plastic modulus, $H' = EE_T/(E - E_T)$. *This is the default hardening option.*

2. *Kinematic hardening* ($\beta = 0$). The radius of the yield surface remains constant at the initial yield value. The yield surface translates in the direction normal to the surface at the current stress contact point. The rate of translation is governed by the plastic modulus, $H' = EE_T/(E - E_T)$, of the uniaxial stress-strain curve.

3. *Mixed hardening* ($0 < \beta < 1.0$). Part of the strain hardening is isotropic and part is kinematic. The value of β controls the proportion assigned to each hardening model, e.g., $\beta = 0.25$ requests that 25% of the hardening be processed as kinematic and 75% as isotropic.

3.3.2 Model Properties

The properties defined for material model *bilinear* are listed in Table 3.3. When the value of *alpha* is specified, that value will be used for *alphax*, *alphay* and *alphaz*.

3.3.3 Model Output

By default, the material model prints no messages during computations. If requested, the material model prints the element number and strain point number whenever the effective stress *first* exceeds the specified yield stress. This option is requested with the nonlinear solution parameter *material messages on* (refer to Section 2.9.8)

The model makes available the strain energy density, U_0, to the element routines for subsequent output. U_0 at step n+1 is evaluated using the trapezoidal rule

$$U_0^{n+1} = U_0^n + \frac{1}{2}\left(\boldsymbol{t}^{n+1} + \boldsymbol{t}^n\right) : \Delta \boldsymbol{d} \tag{3.26}$$

where the unrotated Cauchy stresses and unrotated strain increments are adopted for the finite-strain formulation. Thermal contributions to $\Delta\boldsymbol{d}$ are subtracted prior to the above computation.

The element stress output contains up to three values for the material model "state" variables. These values for the *mises* material are:

mat_val1: accumulated plastic strain, $\bar{\varepsilon}^p = \sqrt{(2/3)}\sum \lambda \Delta t$
mat_val2: equivalent stress, $\bar{\sigma} = \sqrt{(3/2)\xi_{ij}'\xi_{ij}'}$
mat_val3: not used

Model Property	Keyword	Mode	Default Value
Young's modulus	*e*	Number	0.0
Poisson's ratio	*nu*	Real	0.0
Mass density	*rho*	Real	0.0
Yield stress	*yld_pt*	Number	0.0
Hardening modulus (E_T)	*tan_e*	Number	0.0
Hardening mixity (β)	*beta*	Number	1.0
Thermal expansion coefficient for isotropic response	*alpha*	Number	0.0
Thermal expansion coefficients for anisotropic response	*alphax, alphay, alphaz, alphaxy, alphaxz, alphayz*	Number	0.0

Table 3.3 Properties for *bilinear* Material Model

3.3.4 Computational Efficiency

The computational routines for this model process elements in blocks of a size matched to the vector length of the computer (i.e., Crays) or to the cache size of the workstation. All model computations are written in vectorized code. Compared to the general rate-dependent Mises model discussed later, this model is significantly faster.

3.3.5 Example

The following example defines the properties for a mild steel material frequently used in fracture models and assigns the material to some elements.

```
structure cct
c
material a516
  properties bilinear e 30000 nu 0.3 yld_pt 60.0 tan_e 300.0,
      rho  7.3e-07 alphax 1.2e-06 alphay 3.2e-06 alphaz 5.3e-05
c
number of nodes 25642 22092
c
elements
  14000-22092 type l3disop  linear material a516 order 2x2x2,
                        long bbar
c
```

3.3.6 Plasticity Algorithms

During a time step from state n to state n+1, global equilibrium iterations, designated by i, are performed at a constant external load level to reduce the residual sufficiently close to zero. Each iteration allows a new estimate of the strain rate to be determined at the state n+1 which is associated with the iteration. With this estimate, the stress at the ith update of state n+1 is computed. This process is termed the stress recovery and is the principal focus of a material model. For stress updating, the ith estimate of the strain increment over the step is used, $\Delta\epsilon = \epsilon^i_{n+1} - \epsilon_n$, which defines the so-called 'path independent' strategy. The current implementation of this model does not use subincrementation schemes which subdivide the strain increment.

Also necessary at each global iteration is a constitutive tangent operator that relates stress rate to strain rate, or changes in stress to changes in strain, so that increments of displacement from n+1 at i-1 to n+1 at i may be computed and strain rates estimated. This task is also the responsibility of the material model.

The small-strain plasticity model is based on rate independent, isotropic J_2 flow theory considering both isotropic and kinematic hardening and utilizing a bilinear uniaxial material response. The stress recovery during plastic flow is performed using an elastic predictor-radial return numerical integration scheme (see Key [49], Kreig and Key [54], Dodds [22], Keppel and Dodds [48] for additional details). A consistent rather than a continuum tangent operator is computed for use in the calculation of the element tangent stiffness matrix in order to maintain quadratic convergence in the global nonlinear solution (see Simo and Taylor [82]). The complete algorithm for the stress recovery and the evaluation of the consistent tangent operator at a given material point is developed and outlined in the following discussion.

Stress Recovery

Let t_{ij}, d_{ij}, and a_{ij} be the stress, strain rate (minus thermal strain rate), and back stress respectively. Deviator values and norms associated with these tensors are defined by

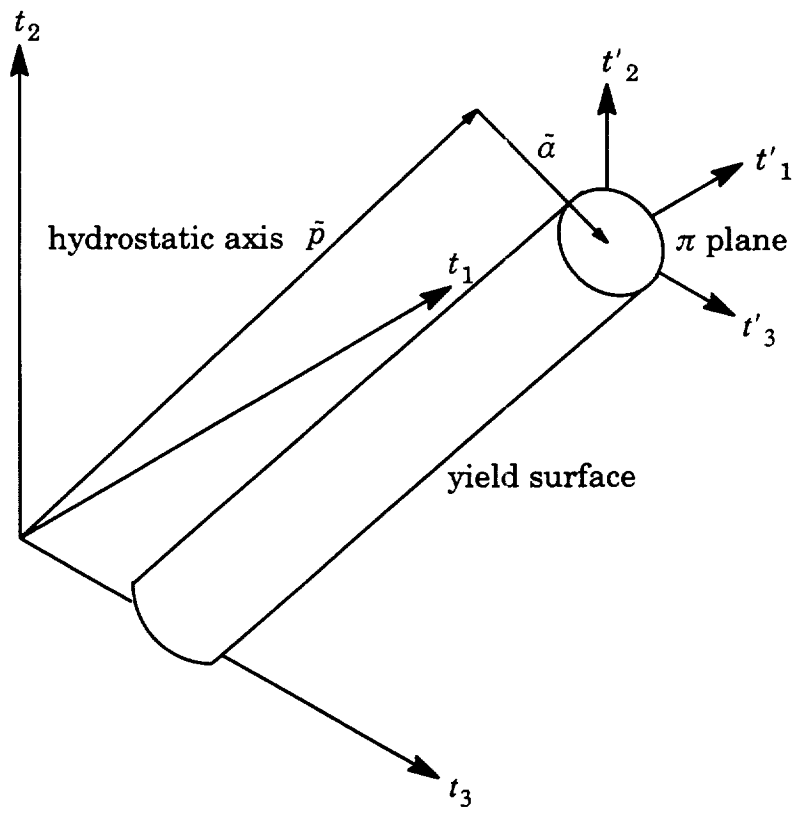


FIG. 3.6—*Mises yield surface in principal stress space*

$$(\,)'_{ij} = (\,)_{ij} - \frac{(\,)_{kk}}{3}\delta_{ij}\,; \qquad \| (\,)_{ij} \| = \sqrt{(\,)_{ij}(\,)_{ij}} \tag{3.27}$$

Because the vector corresponding to a_{ij} in principal stress space lies in the π plane of the yield surface, a_{kk} is zero and the deviator of the back stress is the back stress. Accordingly, the deviator relative stress is given as

$$\xi'_{ij} = t'_{ij} - a_{ij} \tag{3.28}$$

allowing the Mises yield surface (Fig. 3.6) to be described by the equation

$$\frac{\xi'_{ij}\xi'_{ij}}{2} - k^2 = 0 \tag{3.29}$$

where k is proportional to the radius of the yield surface in the π plane.

The strain rate is decomposed into elastic and plastic components by the equation

$$d_{ij} = d^e_{ij} + d^p_{ij} \tag{3.30}$$

The unit normal tensor is defined as

$$n_{ij} = \frac{\xi'_{ij}}{\| \xi'_{ij} \|} \tag{3.31}$$

so that plastic strain rate can be described by the equation

$$d^p_{ij} = \lambda n_{ij} \tag{3.32}$$

Increments of the plastic strain will thus be normal to the yield surface in stress space. As a consequence of J_2 flow theory, d^p_{kk}, the change in plastic volume with time, is zero; the deviator plastic strain rate is therefore equal to the plastic strain rate.

The effective plastic strain rate and the effective stress are defined as

$$\dot{\bar{e}}^p = \sqrt{\frac{2}{3} d^p_{ij} d^p_{ij}} \tag{3.33}$$

and

$$q = \sqrt{3J_2'}\ ; \qquad J_2' = \frac{1}{2} t'_{ij} t'_{ij} \tag{3.34}$$

The derivative of the effective stress with respect to the effective strain is the plastic modulus H'. For a Mises yield surface with a bilinear uniaxial stress-strain diagram, H' is given as

$$H' = \frac{EE_T}{E - E_T} \tag{3.35}$$

where E and E_T are Young's modulus and the tangent modulus, respectively. Note that for a bilinear material, E_T and H' are constants.

Along with Eq. (3.32), the evolution equations for the material are given by

$$\dot{\alpha}_{ij} = \frac{2}{3}(1 - \beta) H' d^p_{ij} = \frac{2}{3}(1 - \beta) H' \lambda n_{ij} \tag{3.36}$$

$$\dot{k} = \frac{\sqrt{2}}{3} \beta H' \parallel d^p_{ij} \parallel = \frac{\sqrt{2}}{3} \beta H' \lambda \tag{3.37}$$

$$\dot{t}'_{ij} = 2G(d'_{ij} - d'^{(p)}_{ij}) \tag{3.38}$$

$$\dot{p} = \frac{\dot{t}_{kk}}{3} = K d_{kk} \tag{3.39}$$

The parameter β controls the type of hardening used in the analysis. It measures the proportion of the hardening which is isotropic, ranging in value between zero and one. Values of $\beta = 0.0$, 1.0, and 0.25 indicate pure kinematic hardening , pure isotropic hardening, and 25% isotropic hardening - 75% kinematic hardening. The parameters K and G are the bulk and shear moduli of the material.

The material point is assumed to be strained at a constant rate during the time step. Rate tensors are evaluated at state n+1/2 when integrated to produce an increment over the step. Consequently, the hydrostatic stress p of Eq. (3.39) and the elastic predictor trial deviator stress at state n+1 are computed as

$${}^{n+1}p = {}^{n}p + K\Delta t\ {}^{n+1/2}d'_{kk} \tag{3.40}$$

$${}^{n+1}t'^{t}_{ij} = {}^{n}t'_{ij} + 2G\Delta t\ {}^{n+1/2}d'_{ij} \tag{3.41}$$

The trial deviator relative stress is defined in terms of the trial deviator stress and the back stress at state n:

$$ {}^{n+1}\xi'^{t}_{ij} = {}^{n+1}t'^{t}_{ij} - {}^{n}\alpha_{ij} \tag{3.42}$$

At this stage, if the material point is elastic, the stress recovery is essentially complete. It remains only to re-combine the hydrostatic stress and the trial deviator stress. If the material point is in the state of plastic flow, using Eq. (3.38) the trial deviator stress is modified by a stress increment corresponding to a radial return to the yield surface in order to calculate the updated deviator stress at state n+1:

$$ {}^{n+1}t'_{ij} = {}^{n+1}t'^{t}_{ij} - 2G\Delta t\, {}^{n+1/2}d'^{(p)}_{ij} = {}^{n+1}t'^{t}_{ij} - 2G\lambda\Delta t n_{ij} \tag{3.43}$$

For simplicity of notation, in Eq. (3.43) and later λ is taken as evaluated at state n+1/2 and n_{ij} at state n+1. The updated back stress at state n+1 follows from Eq. (3.36):

$$ {}^{n+1}\alpha_{ij} = {}^{n}\alpha_{ij} + \tfrac{2}{3}(1-\beta)H'\lambda\Delta t n_{ij} \tag{3.44}$$

Combining Eqs. (3.43) and (3.44), the deviator relative stress at state n+1 is expressed as

$$ {}^{n+1}\xi'_{ij} = {}^{n+1}\xi'^{t}_{ij} - \lambda\Delta t\left[\, 2G + \tfrac{2}{3}(1-\beta)H' \,\right] n_{ij} \tag{3.45}$$

Specifying the unit normal tensor at state n+1 to be

$$ n_{ij} = \frac{{}^{n+1}\xi'^{t}_{ij}}{\| {}^{n+1}\xi'^{t}_{ij} \|} \tag{3.46}$$

and substituting into Eq. (3.45) leads to the relationship

$$ \| {}^{n+1}\xi'_{ij} \| = \| {}^{n+1}\xi'^{t}_{ij} \| - \lambda\Delta t\left[\, 2G + \tfrac{2}{3}(1-\beta)H' \,\right] \tag{3.47}$$

Recasting Eq. (3.29) as

$$ \| {}^{n+1}\xi'_{ij} \| - \sqrt{2}\, {}^{n+1}k = 0 \tag{3.48}$$

and noting from Eq. (3.37) that

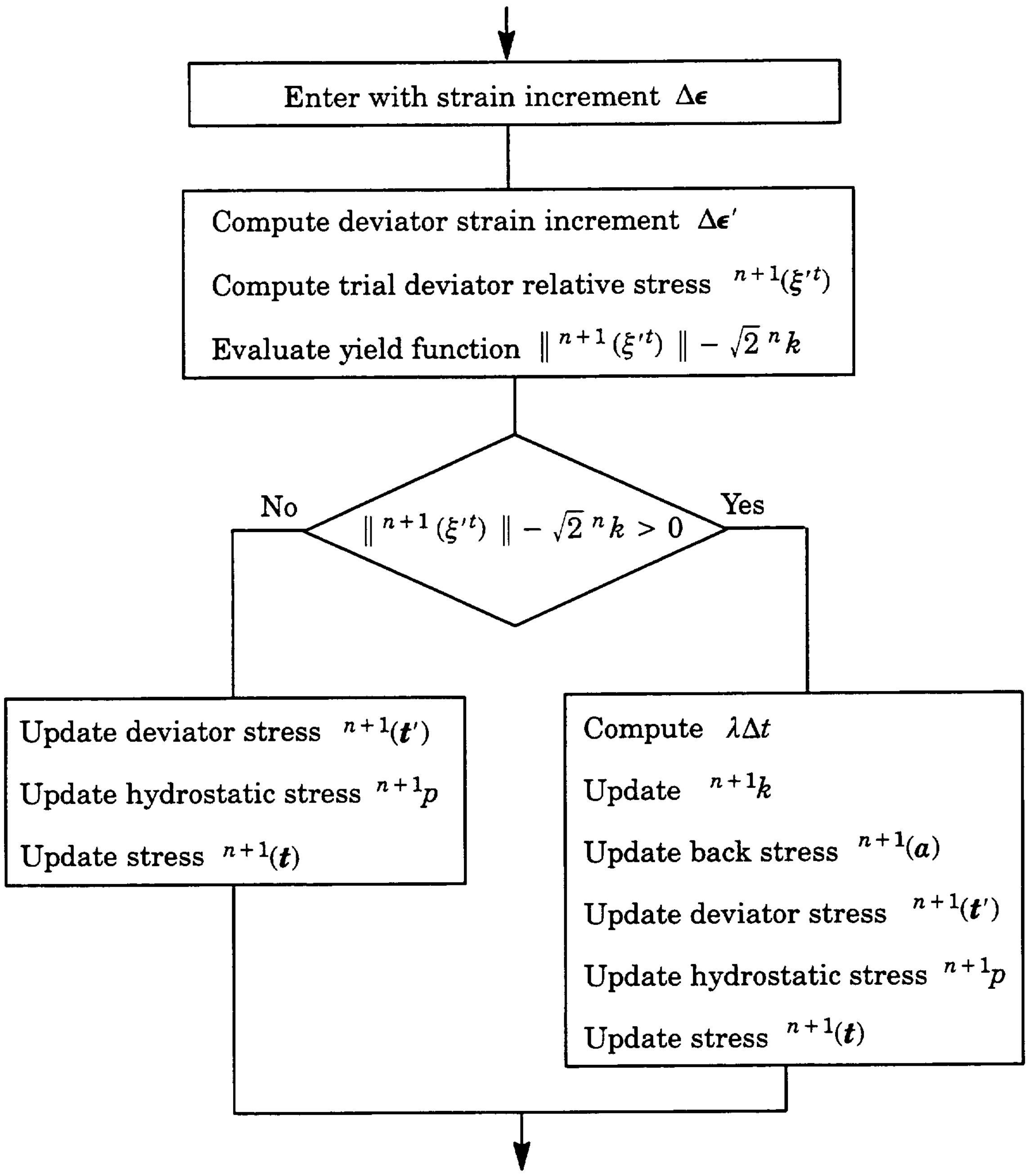
$$ {}^{n+1}k = {}^{n}k + \frac{\sqrt{2}}{3}\beta H'\lambda\Delta t \tag{3.49}$$

allows Eq. (3.47) to be manipulated, yielding

$$ \lambda\Delta t = \frac{\| {}^{n+1}\xi'^{t}_{ij} \| - \sqrt{2}\, {}^{n}k}{2G\left(1 + \frac{H'}{3G}\right)} \tag{3.50}$$

$\lambda\Delta t$ is backsubstituted into the preceding equations to resolve all stresses and state variables. It is possible to directly compute $\lambda\Delta t$ because H' is a constant signifying that the effective stress is a linear function of the effective plastic strain. If this function is not linear, then it would be necessary to iterate to determine $\lambda\Delta t$.

A flow chart illustrating the steps required for the recovery of stresses is displayed in Fig. 3.7. The algorithm above is implemented in a vector form. The six components of stress tensors are arrayed in the order { 11 22 33 12 23 13 }. Strain tensors correspond to vectors with identical ordering but with diagonal terms doubled to form engineering strains.



Enter with strain increment $\Delta\epsilon$
Compute deviator strain increment $\Delta\epsilon'$
Compute trial deviator relative stress ${}^{n+1}(\xi'^t)$
Evaluate yield function $\| {}^{n+1}(\xi'^t) \| - \sqrt{2}\,{}^n k$
No
Yes
$\| {}^{n+1}(\xi'^t) \| - \sqrt{2}\,{}^n k > 0$
Update deviator stress ${}^{n+1}(\boldsymbol{t}')$
Update hydrostatic stress ${}^{n+1}p$
Update stress ${}^{n+1}(\boldsymbol{t})$
Compute $\lambda\Delta t$
Update ${}^{n+1}k$
Update back stress ${}^{n+1}(a)$
Update deviator stress ${}^{n+1}(\boldsymbol{t}')$
Update hydrostatic stress ${}^{n+1}p$
Update stress ${}^{n+1}(\boldsymbol{t})$

FIG. 3.7—*Stress recovery procedure for bilinear (Mises) material model*

Consistent Tangent Operator

The tangent operator required for the calculation of element tangent stiffness matrices satisfies the following relationship between stress rate and the total strain rate:

$$\dot{t}_{ij} = C_{ijkl}\, d_{kl} \tag{3.51}$$

For a material point in the elastic state, the isotropic tangent operator is given by

$$C^{E}_{ijkl} = K\delta_{ij}\delta_{kl} + 2G\left[\tfrac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \tfrac{1}{3}\delta_{ij}\delta_{kl}\right] \qquad (3.52)$$

Once the material point experiences plastic flow, the tangent operator is given by

$$C^{EP}_{ijkl} = K\delta_{ij}\delta_{kl} + 2G\left[\tfrac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \tfrac{1}{3}\delta_{ij}\delta_{kl}\right] - 2G\gamma n_{ij} n_{kl} \qquad (3.53)$$

$$\gamma = \frac{1}{\left[1 + \dfrac{H'}{3G}\right]} \qquad (3.54)$$

The operator of Eq. (3.53) is termed the continuum tangent operator. Its use is compatible with an exact integration of the evolution equations, which are continuum in nature. However, the elastic predictor-radial return procedure for stress updating does not represent an exact integration; it is in essence a secant approach. Not surprisingly, use of the continuum tangent operator leads to a degradation in the quadratic convergence characteristic of the global Newton iterations. Simo and Taylor [82] established a tangent operator compatible with the elastic predictor-radial return algorithm which preserves the quadratic convergence. It is often termed the *consistent tangent* operator, and it is the tangent operator employed in WARP. The consistent tangent operator is given by the following equations:

$$C^{EP}_{ijkl} = K\delta_{ij}\delta_{kl} + 2GB\left[\tfrac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \tfrac{1}{3}\delta_{ij}\delta_{kl}\right] - 2G\bar{\gamma} n_{ij} n_{kl} \qquad (3.55)$$

$$B = \frac{\left[\sqrt{2}\;{}^{n+1}k + \tfrac{2}{3}(1-\beta)H'\lambda\Delta t\right]}{\|\;{}^{n+1}\xi'^{t}_{ij}\;\|}\;;\quad \bar{\gamma} = \frac{1}{\left[1 + \dfrac{H'}{3G}\right]} - (1 - B) \qquad (3.56)$$

In the code, the above tangent operator is applied in a 6×6 matrix form that relates a stress vector to an engineering strain vector.

3.4 Material Model Type: *mises*

This material model extends the capabilities offered by the *bilinear* (mises) model to include more general descriptions of the uniaxial stress-strain response and viscoplastic effects. The response to temperature changes imposed in the analysis follows that described previously for the *bilinear* model.

The *mises* model provides three options for the inviscid uniaxial (tensile) stress-strain curve: (1) constant linear hardening after yield, (2) pure power-law hardening after an initially linear response prior to yield, and (3) general piecewise-linear description. This generalized form of mises plasticity supports only isotropic hardening.

To introduce a rate dependence into the model, we adopt a power-law viscoplastic relationship suitable for ductile metals undergoing large amounts of plastic straining. The viscoplastic strain rate is given by

$$\dot{\epsilon}^{vp} = D\left[\left(\frac{q}{\sigma_e}\right)^m - 1\right] \tag{3.57}$$

where D and m are user-specified material constants, q denotes the rate-dependent (uniaxial) tensile stress and σ_e the inviscid (uniaxial) tensile stress. The viscosity term is often written in the form $D = 1/\eta$. For a moderately rate-sensitive material, such as an A533B pressure vessel steel at 100° C, typical values of D and m are 1.0 (*in./in./sec*) and 35, respectively. In the simplest case, σ_e is specified to remain constant at the yield stress, σ_0 (the linear hardening model with $E_T = 0$). More generally, σ_e is a linear or power-law function of ϵ^{vp}.

The following sections describe needed parameters to utilize the *mises* material model. Additional details for rate-dependent features of the model are then provided. All other aspects of the formulation follow those of the *bilinear* model described in the previous section.

3.4.1 Stress-Strain Curves and Hardening

The inviscid uniaxial stress-strain curve for the material is represented by: (1) the linear hardening model shown in Fig. 3.5, (2) a linear, power-law model shown in Fig. 3.8 or (3) by a general piecewise-linear curve of the type shown in Fig. 2.1. To maintain continuous values of E_T between the initially linear and power-law regions, a small (cubic) transition curve is inserted automatically in the description of the stress-strain curve.

For a small-strain analysis (*linear* kinematic formulation), specify *engineering* values for the strain(ϵ_E) and stress (σ_E). For a finite-strain analysis (*nonlinear* kinematic formulation), specify the uniaxial stress-strain curve using the logarithmic strain, ϵ, and the true (Cauchy) stress, σ. *For a finite-strain analysis, the user should convert conventional engineering strain, ϵ_E, and engineering (nominal) stress, σ_E, values for input using the relations:*

$$\sigma = \sigma_E(1 + \epsilon_E) \tag{3.58}$$

$$\epsilon = \ln(1 + \epsilon_E) \tag{3.59}$$

The above conversions assume incompressible, homogeneous deformation. The true stress-true strain curve discussed here assumes homogeneous, uniaxial deformation of the material, i.e., prior to necking. Once necking occurs, the above expressions are no longer applicable. More elaborate corrections, for example those developed by Bridgeman, are required.

Once yielding begins, the inviscid hardening follows the isotropic model.

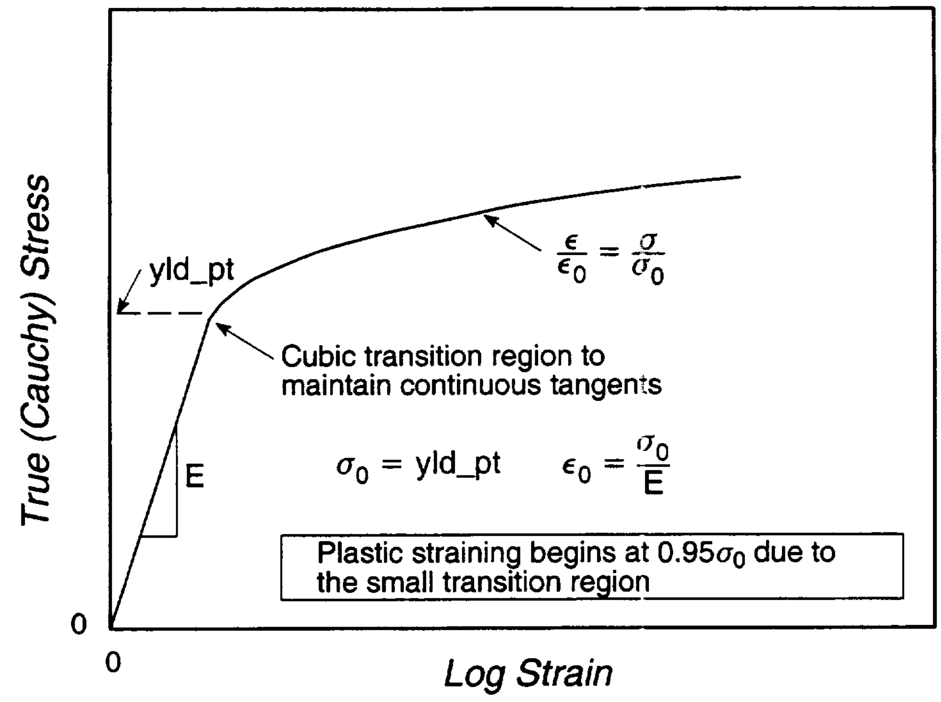


FIG. 3.8—*Power-law form of the inviscid uniaxial (tensile) stress-strain curve for the "mises" plasticity material model. For finite-strain analysis, input the Cauchy stress and log strain description.*

3.4.2 Model Properties

The properties defined for material model *mises* are listed in Table 3.4. When the *curve* option is invoked to indicate a separately defined piecewise-linear stress-strain curve, Young's modulus must still be specified. Thermal expansion coefficients are identical to those listed for the *bilinear* model.

3.4.3 Model Output

By default, the material model prints no messages during computations. If requested, the material model prints the element number and strain point number whenever the effective stress *first* exceeds the specified yield stress. This option is requested with the nonlinear solution parameter *material messages on* (refer to Section 2.9.8)

The model makes available the strain energy density, U_0, at each Gauss point to the element routines for subsequent output. U_0 at step n+1 is evaluated using the trapezoidal rule

$$U_0^{n+1} = U_0^n + \tfrac{1}{2}\left(\boldsymbol{t}^{n+1} + \boldsymbol{t}^n\right) : \Delta\boldsymbol{d} \qquad (3.60)$$

where the unrotated Cauchy stresses and unrotated strain increments are adopted for the finite-strain formulation. Thermal contributions to $\Delta\boldsymbol{d}$ are subtracted prior to the above computation.

The element stress output contains up to three values for the material model "state" variables. These values for the *mises* material are:

The following sections describe the techniques used to solve Eq. (3.61) first for the inviscid case with power-law hardening and then for the general viscoplastic case. Once $\Delta\epsilon^p$ and $\sigma_e(\epsilon^p_{n+1})$ are determined from these calculations, the correction of the trial deviatoric stress to the yield surface and the inclusion of hydrostatic stress terms proceeds exactly as for the bilinear model. Also discussed here is the proper definition of H' for use in the consistent tangent operator to model the power-law hardening and viscoplastic cases.

Rate-Independent Consistency Equation

When the uniaxial tensile response has other than linear (constant) hardening, Eq. (3.61) is nonlinear in the plastic strain increment, $\Delta\epsilon^p$, and requires an iterative solution. Re-write Eq. (3.61) in the form

$$R = \sigma^T - 3G\Delta\epsilon^p - \sigma_e(\epsilon^p_n + \Delta\epsilon^p) \equiv 0 \tag{3.62}$$

where σ^T, the shear modulus G and the scalar plastic strain at the beginning of the step (ϵ^p_n) remain fixed during the solution of this equation to make $R \rightarrow 0$. The user-supplied form of $\sigma_e(\bar{\epsilon})$ can be quite complex (power-law hardening with an initial cubic transition region or piecewise-linear). Power-law hardening, preceded by a small cubic transition curve, defines a very smooth decay of the tangent modulus (E_T) with increasing strain and leads to a very stable solution of Eq. (3.62) via a local Newton iteration. However, the piecewise-linear form often causes such a local Newton procedure to fail: the stress-strain curve may be defined to "stiffen" after an initially low hardening region, for example, due to Luder's strains and/or the discontinuous E_T values at break points on the curve can mislead a pure Newton scheme. After much experimentation, we have adopted a variant of the false position approach, known as Ridders' method, to iteratively solve Eq. (3.62) in all cases. The method has superlinear convergence and readily eliminates the difficulties of a pure Newton scheme. The procedure insures that during iterations the cureent estimate of $\Delta\epsilon^p$ never strays outside the lower-bound value of 0.0 and the upper-bound value found by assuming that $\sigma_e(\bar{\epsilon}_{n+1}) = \sigma_e(\bar{\epsilon}_n)$, i.e. no further hardening. The procedure converges to very tight tolerances on $\Delta\epsilon^p$ and $\sigma_e(\bar{\epsilon}_{n+1})$ in 2-3 cycles and has proven very robust.

When the uniaxial stress-strain curve follows the power-law model, the evaluation of $\sigma_e(\epsilon^p_n + \Delta\epsilon^p)$ for each trial value of $\Delta\epsilon^p$ (during Ridders' method) itself requires a local Newton iteration as described below. The power-law model is defined by (neglecting yet more complication with the initial cubic transition region and dropping the implied e subscript)

$$\frac{\epsilon}{\epsilon_0} = \frac{\sigma}{\sigma_0}, \qquad \epsilon \le \epsilon_0 \tag{3.63a}$$

$$\frac{\epsilon}{\epsilon_0} = \left(\frac{\sigma}{\sigma_0}\right)^N, \quad \epsilon > \epsilon_0 \tag{3.63b}$$

where σ_0 and ϵ_0 are reference yield stress and strain levels that also define E. To evaluate Eq. (3.63b) given an estimate of $\Delta\epsilon^p$, write

$$\epsilon_{n+1} = \frac{\sigma_{n+1}}{E} + \epsilon^p_n + \Delta\epsilon^p \tag{3.64}$$

This is a simple, nonlinear equation solved readily for ϵ_{n+1} and thus σ_{n+1} using a local Newton scheme. Define the residual, $\overline{R}$, of Eq. (3.64) by

$$\overline{R} = \epsilon_{n+1} - \frac{\sigma_{n+1}}{E} - \epsilon^p_{n+1}\,. \tag{3.65}$$

For the (i) estimate of ϵ_{n+1}, find the change in $\overline{R}$ such that $\overline{R} + d\overline{R} = 0$ where

$$d\overline{R} = \frac{\partial \overline{R}}{\partial \epsilon_{n+1}} d\epsilon_{n+1} . \tag{3.66}$$

The required derivative is found to be

$$\frac{\partial \overline{R}}{\partial \epsilon_{n+1}} = 1 - \frac{1}{N}\left(\frac{\epsilon_{n+1}}{\epsilon_0}\right)^{\frac{1}{N}-1} . \tag{3.67}$$

Successive improvements to the value of ϵ_{n+1} are thus

$$\epsilon^{(i+1)}_{n+1} = \epsilon^{(i)}_{n+1} + d\epsilon^{(i)}_{n+1} = \epsilon^{(i)}_{n+1} - \frac{\overline{R}^{(i)}}{\dfrac{\partial \overline{R}}{\partial \epsilon_{n+1}}} \tag{3.68}$$

Iterations continue until convergence on ϵ_{n+1} and σ_{n+1} is achieved. Suitable convergence tests are

$$\left|\sigma^{(i+1)}_{n+1} - \sigma^{(i)}_{n+1}\right| \le tol\, \sigma^{(i+1)}_{n+1} \tag{3.69}$$

$$\left|\epsilon^{(i+1)}_{n+1} - \epsilon^{(i)}_{n+1}\right| \le tol\, \epsilon^{(i+1)}_{n+1} \tag{3.70}$$

where we specify 10^{-6} for *tol*. The starting estimate $\epsilon^{(1)}_{n+1}$ is given by

$$\epsilon^{(1)}_{n+1} = \epsilon_0 + \Delta\epsilon^p \tag{3.71}$$

We find convergence is achieved in at most 3 iterations over Eq. (3.65) - (3.70). The instantaneous plastic modulus, needed for the consistent tangent, is given by

$$H'_{n+1} = \frac{EE_{T,n+1}}{E - E_{T,n+1}} \tag{3.72}$$

where

$$E_{T,n+1} = \frac{E}{N}\left(\frac{\sigma_{n+1}}{\sigma_0}\right)^{(1-N)} . \tag{3.73}$$

With a converged value for $\Delta\epsilon^p$ given by the solution of Eq. (3.61), the updated stress state is computed by the usual radial return to the updated yield surface (isotropic hardening)

$$\{\sigma\}_{n+1} = \{\sigma^T\}_{n+1} - \frac{3G\Delta\epsilon^p}{\sigma^T}\{S^T\}_{n+1} , \quad \{\,\}\ \textit{implies a 6x1 vector} \tag{3.74}$$

where $\{S^T\}_{n+1}$ is the deviatoric portion of the trial elastic stress state $\{\sigma^T\}_{n+1}$.

Rate-Dependent Consistency Equation

To introduce a rate dependence into the model, re-write the consistency equation of Eq. (3.61) in the form

$$\sigma^T - 3G\Delta\epsilon^{vp} - q(\epsilon^{vp}_{n+1}, \sigma_{e,n+1}, \Delta t) \equiv 0 \tag{3.75}$$

where q denotes the rate-dependent equivalent stress, $\sigma_{e,n+1}$ becomes the inviscid equivalent stress (which may be a nonlinear function of ϵ^{vp}) and $\Delta t = t_{n+1} - t_n$. We adopt a power-law viscoplastic relationship suitable for ductile metals undergoing large amounts of plastic straining. The viscoplastic strain rate is given by *mat_val1:* matrix plastic strain, $\bar{\epsilon}^p = \sqrt{(2/3)\epsilon^p_{ij}\epsilon^p_{ij}}$

mat_val2: matrix equivalent stress, $\bar{\sigma} = \sqrt{(3/2)\sigma_{ij}'\sigma_{ij}'}$

mat_val3: not used

3.4.4 Computational Efficiency

The computational routines for this model process elements in blocks of a size matched to the vector length of the computer (i.e., Crays) or to the cache size of the workstation. The majority of model computations are written in vectorized code. The local Newton loop to solve the scalar consistency equation executes in scalar mode. In terms of efficiency, the constant (linear) hardening form of the stress-strain curve incurs the least computational effort for inviscid analyses (no local Newton loop is needed to solve the consistency equation). The piecewise-linear model provides the next most efficiency followed by the power-law model. The large number of exponentiations required with the power-law model significantly increases the computational effort (often 25% total job time) in most cases. We strongly recommend description of stress-strain properties with the piecewise-linear model even when the material description follows the power-law description—just to reduce the computational effort.

Our testing indicates the piecewise-linear model combined with the viscoplastic option can reduce the convergence rate of global Newton iterations. No such degradation is experienced with simple linear hardening or power-law hardening combined with viscoplasticity.

This model is computationally less efficient than the simple *bilinear* model of the previous sections.

Model Property	**Keyword**	**Mode**	**Default Value**
Young's modulus	*e*	Number	0.0
Poisson's ratio	*nu*	Real	0.0
Mass density	*rho*	Number	0.0
Yield stress	*yld_pt*	Number	0.0
Hardening modulus (E_T)	*tan_e*	Number	0.0
Power law exponent (n)	*n_power*	Number	0.0
Reference strain rate (D)	*ref_eps*	Number	0.0
Viscous exponent (m)	*m_power*	Number	0.0
Stress-strain curve	*curve*	Number	0

Thermal expansion coefficient for isotropic response	*alpha*	Number	0.0
Thermal expansion coefficients for anisotropic response	*alphax, alphay, alphaz, alphaxy, alphaxz, alphayz*	Number	0.0

Table 3.4 Properties for *mises* Material Model

3.4.5 Example

The following example defines the properties for two mild steels material frequently used in fracture models and assigns the material to some elements.

```
structure cct
c
stress-strain curve 3
  0.0012 36 0.01 36,  0.05 50,
  0.10 55, 0.30 60
c
material a533b
  properties mises e 30000 nu 0.3 yld_pt 60.0 n_power 10,
      rho  7.3e-07 ref_eps 40 m_power 20
c
material a36
  properties mises e 30000 nu 0.3 curve 3 rho  7.3e-07
c
number of nodes 25642 22092
c
elements
  14000-22092 type l3disop linear material a533b order 2x2x2,
                     long bbar
c
```

3.4.6 Plasticity Algorithms

The formulation and implementation of the general, rate-dependent *mises* model differs from the *bilinear* model in the complexity of computing the term $\lambda\Delta t$. Eqs. (3.45) and (3.46) define the deviatoric terms of the updated stress state as a return to the new yield surface along the direction of trial elastic deviator (which for Mises is normal to the updated yield surface). Eq. (3.47) then represents the scalar product of each side of Eq. (3.45) and defines the so-called *scalar consistency equation* for determination $\lambda\Delta t$. Using the relationships of Eqs. (3.33) and (3.34), the consistency equation may be written in the simpler form

$$\sigma^T - 3G\Delta\epsilon^p - \sigma_e(\epsilon^p_{n+1}) \equiv 0 \tag{3.61}$$

where σ^T is the equivalent uniaxial stress computed from the elastic trial stress at the step (n+1) [see Eq. (3.41)], G is the elastic shear modulus, $\Delta\epsilon^p$ is the unknown increment of plastic strain over the step, $\Delta\epsilon^p = \epsilon^p_{n+1} - \epsilon^p_n$, and $\sigma_e(\epsilon^p_{n+1})$ is the equivalent (Mises) stress corresponding to the plastic strain at the end of the step. The analyst provides the functional relationship, $\sigma_e(\epsilon^p)$, through the uniaxial stress-strain curves described previously [to make this simpler, the analyst actually specifies $\sigma_e(\bar{\epsilon})$ rather than $\sigma_e(\epsilon^p)$].

$$\dot{\epsilon}^{vp} = \frac{1}{\eta}\left[\left(\frac{q}{\sigma_e}\right)^m - 1\right] \tag{3.76}$$

where η and m are material constants. The viscosity term is often written in the form $D = 1/\eta$. In the simplest case, σ_e is specified to remain constant at the yield stress, σ_0. More generally, σ_e is a nonlinear function of ϵ^{vp}.

The integration of Eq. (3.76) with a backward Euler procedure yields

$$\Delta\epsilon^{vp} = \frac{\Delta t}{\eta}\left[\left(\frac{q_{n+1}}{\sigma_{e,n+1}}\right)^m - 1\right] \tag{3.77}$$

which is solved for q_{n+1}

$$q_{n+1} = \sigma_{e,n+1}\left[\left(\frac{\eta\Delta\epsilon^{vp}}{\Delta t}\right) + 1\right]^{1/m} . \tag{3.78}$$

We observe in Eq. (3.78) that as $\eta/\Delta t \rightarrow 0$ the inviscid solution is recovered. The rate-dependent consistency equation, Eq. (3.75), is also solved using Ridders' method for $\Delta\epsilon^{vp}$ with q_{n+1} defined as in Eq. (3.78). Convergence is achieved in a few iterations. With $\Delta\epsilon^{vp}$ known, the updated stress state at n+1 is given by the usual radial-return to the yield surface

$$\{\sigma\}_{n+1} = \{\sigma^T\}_{n+1} - \frac{3G\Delta\epsilon^{vp}}{\sigma^T}\{S^T\}_{n+1} . \tag{3.79}$$

To form the consistent tangent, the instantaneous plastic modulus for the rate-dependant equivalent stress is required

$$H'_{q,n+1} = \left.\frac{dq}{d\epsilon^{vp}}\right|_{n+1} \tag{3.80}$$

We must obtain $H_{q,n+1}$ by differentiating the *algorithm* that defines the evolution of q. From Eq. (3.77) we obtain

$$d\dot{\epsilon}^{vp}_{n+1} = d(\Delta\epsilon^{vp}) = \frac{m\Delta t}{\eta}\left(\frac{q_{n+1}}{\sigma_{e,n+1}}\right)^{m-1}\left(\frac{\sigma_{e,n+1}dq_{n+1} - q_{n+1}d\sigma_{e,n+1}}{\sigma^2_{e,n+1}}\right) . \tag{3.81}$$

By substituting for $d\sigma_{e,n+1}$ in terms of the plastic modulus for the inviscid response Eq. (3.72)

$$d\sigma_{e,n+1} = H'_{n+1}d\epsilon^{vp}_{n+1} \tag{3.82}$$

Eq. (3.81) is solved for $H'_{q,n+1}$ as

$$H'_{q,n+1} = \left.\frac{dq}{d\epsilon^{vp}_{n+1}}\right|_{n+1} = \frac{\eta\sigma_{e,n+1}}{m\Delta t}\left(\frac{q_{n+1}}{\sigma_{e,n+1}}\right)^{1-m} + \left(\frac{q_{n+1}}{\sigma_{e,n+1}}\right)H'_{n+1} \tag{3.83}$$

The plastic modulus $H'_{q,n+1}$ provides the value of H' that appears in the consistent tangent operator, Eq. (3.55). Note that as $\eta/\Delta t \rightarrow 0$ in Eq. (3.83), $q_{n+1}/\sigma_{e,n+1}$ must also $\rightarrow$1 and the inviscid H'_{n+1} is recovered.

3.5 Material Model Type: *gurson*

This material model implements the Gurson–Tvergaard (GT) plastic potential to predict the response of an elastic–plastic solid containing voids (Gurson [29], see Tvergaard [88] for a comprehensive review). The basis for the model derives from analyses of a single cell containing a centered spherical void of initial volume fraction f_0. The void volume fraction, f, increases under loading and eventually leads to a gradual loss of stress carrying capacity for a macroscopic material element. With this model, a material element effectively contains a void of volume fraction f and (solid) matrix material of volume fraction $(1–f)$. The matrix response follows the material's uniaxial (tensile) stress–strain properties which can be represented in one of several ways and can also include viscoplastic effects.

The GT yield condition is given by

$$g(\sigma_e, \sigma_m, \bar{\sigma}, f) = \left(\frac{\sigma_e}{\bar{\sigma}}\right)^2 + 2q_1 f \cosh\left(\frac{3q_2\sigma_m}{2\bar{\sigma}}\right) - \left(1 + q_3 f^2\right) = 0 \qquad (3.84)$$

where σ_e denotes the (Mises) equivalent (macroscopic) stress, σ_m is the mean (macroscopic) stress, $\bar{\sigma}$ is the (Mises) equivalent stress of the matrix and f is the current void fraction. Factors q_1, q_2, and q_3 introduced by Tvergaard improve the model predictions for periodic arrays of cylindrical and spherical voids. When $f = 0$ the yield condition reduces to conventional J_2 plasticity. The computations for this model should be carried out with the finite strain formulation (*nonlinear* element property) so that Cauchy stresses are used in the evaluation of Eq. (3.84).

The current implementation employs a backward Euler technique developed by Aravas [2] to integrate the plasticity rate equations. This procedure is unconditionally stable thereby permitting the use of larger load increments than is possible with traditional forward Euler and semi-explicit procedures. However, the use of large load increments can lead to non-convergence of Newton loops within the model to resolve updated state variables.

This model offers three forms for the uniaxial (tensile) response of the matrix material, an option to include viscoplasticity in the response and a strain-controlled nucleation model to initiate new voids at severe levels of plastic deformation. The response to temperature changes imposed in the analysis follows that described previously for the *bilinear* model.

3.5.1 Stress–Strain Curves

The inviscid uniaxial stress-strain curve for the material is represented by: (1) the linear hardening model shown in Fig. 3.5, (2) a linear, power–law model shown in Fig. 3.8 or (3) by a general piecewise linear curve of the type shown in Fig. 2.1. To maintain continuous values of E_T between the linear and power-law regions, a small (cubic) transition curve is inserted automatically in the description of the stress-strain curve. For the piecewise linear curve, the jumps in E_T across segments can adversely affect convergence of local Newton iterations used to solve the consistency equation (fewer segments and a non-stiffening curve work best).

For a small–strain analysis (*linear* kinematic formulation), specify *engineering* values for the strain(ϵ_E) and stress (σ_E). For a finite–strain analysis (*nonlinear* kinematic formulation), specify the uniaxial stress–strain curve using the logarithmic strain, ϵ, and the true (Cauchy) stress, σ. *For a finite–strain analysis, the user should convert conventional engineering strain, ϵ_E, and engineering (nominal) stress, σ_E, values for input using the relations:*

$$\sigma = \sigma_E\left(1 + \epsilon_E\right) \qquad (3.85)$$

$$\epsilon = \ln(1 + \epsilon_E) \tag{3.86}$$

The above conversions assume incompressible, homogeneous deformation. The true stress–true strain curve discussed here assumes homogeneous, uniaxial deformation of the material, i.e., prior to necking. Once necking occurs, the above expressions are no longer applicable. More elaborate corrections, for example those developed by Bridgeman, are required.

3.5.2 Viscoplasticity

To introduce a rate dependence into the matrix response, we adopt a power–law viscoplastic relationship suitable for ductile metals undergoing large amounts of plastic straining. The viscoplastic strain rate is given by

$$\dot{\epsilon}^{vp} = D\left[\left(\frac{q}{\sigma_e}\right)^m - 1\right] \tag{3.87}$$

where D and m are user–specified material constants, q denotes the rate–dependent (uniaxial) tensile stress and σ_e the inviscid (uniaxial) tensile stress. The viscosity term is often written in the form D=1/η. For a moderately rate–sensitive material, such as an A533B pressure vessel steel at 100° C, typical values of D and m are 1.0 (*in./in./sec*) and 35, respectively. In the simplest case, σ_e is specified to remain constant at the yield stress, σ_0 (the linear hardening model with $E_T = 0$). More generally, σ_e is a linear or power-law function of ϵ^{vp}.

We recommend that the piecewise-linear description of the tensile stress–strain curve not be used with the viscoplastic option at this time. Convergence of the global Newton iterations is sometimes reduced; no such problems occur for the linear or power-law hardening options.

3.5.3 Nucleation Model

The volume fraction of voids increases over an increment of load due to continued growth of existing voids and due to the formation of new voids caused by interfacial decohesion of inclusions or second phase particles. Thus,

$$df = df_{growth} + df_{nucleation} \ . \tag{3.88}$$

The growth component is defined by the current volume fraction of voids and the macroscopic change in void fraction is (the matrix material satisfies plastic incompressibility)

$$df_{growth} = (1 - f)d\boldsymbol{\epsilon}^p : \mathbf{I} = (1 - f)d\epsilon_p \ . \tag{3.89}$$

We adopt an evolution model for nucleation based on current plastic strain in the matrix

$$df_{nucleation} = \mathcal{A}(\bar{\epsilon}^p)d\bar{\epsilon}^p \ . \tag{3.90}$$

Chu and Needleman suggest a form for $\mathcal{A}$ as

$$\mathcal{A} = \frac{f_N}{s_N\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{\bar{\epsilon}^p - \epsilon_N}{s_N}\right)^2\right] \tag{3.91}$$

where the nucleation strain follows a normal distribution with a mean value ϵ_N and a standard deviation s_N with the volume fraction of void nucleating particles given by f_N. A simpler form of Gurson's model which neglects nucleation is derived by setting $\mathcal{A} \equiv 0$ (three fewer material parameters are then required).

Model Property	Keyword	Mode	Default Value
Young's modulus	*e*	Number	0.0
Poisson's ratio	*nu*	Real	0.0
Mass density	*rho*	Number	0.0
Yield stress	*yld_pt*	Number	0.0
Hardening modulus (E_T)	*tan_e*	Number	0.0
Power law exponent (n)	*n_power*	Number	0.0
Reference strain rate (D)	*ref_eps*	Number	0.0
Viscous exponent (m)	*m_power*	Number	0.0
Initial porosity (f_0)	*f_0*	Number	0.0
Yield function parameter q_1	*q1*	Number	1.5
Yield function parameter q_2	*q2*	Number	1.0
Yield function parameter q_3	*q3*	Number	2.25
Include nucleation of new voids	*nucleation*	Logical	.False.
Nucleation parameter f_N	*f_n*	Number	0.04
Nucleation parameter s_N	*s_n*	Number	0.10
Nucleation parameter ε_N	*e_n*	Number	0.30
Put element in ***killable*** list	*killable*	Logical	.False.
Suppress step size cutbacks	*no_cutback*	Logical	.False.
Stress–strain curve	*curve*	Number	0

Table 3.5 Properties for *gurson* Material Model

3.5.4 Element Extinction

Under increasing deformation, the void volume fraction reaches a level at which the element capacity to resist stress decreases precipitously. This final stage of deformation just prior to material separation is not realistically predicted with the GT model (even though the numerical computations remain stable to very high levels of f, approaching 0.5).

Chapter 5 describes an extinction procedure which removes elements from the model and slowly reduces the remaining tractions to zero. This occurs when the void fraction f reaches a user–specified level, denoted f_E. The crack growth procedures in Chapter 5 apply only to elements which have an associated *gurson* material with the material logical property *killable* specified.

When the *killable* property is not specified, the stress updating process continues with f increasing. Eventually, the model routines may request load step reductions to stabilize the state update process.

3.5.5 Adaptive Step Sizes

By default, the *gurson* model routines request a global load step *cutback* when the state update process fails to converge. If the nonlinear solution parameter *adaptive on* is in effect (see Section 2.9.4), the global load step reduction occurs and subsequent *gurson* computations nearly always converge. If the nonlinear solution parameters have *adaptive off*, the *gurson* routines print an message describing the convergence problem and terminate the analysis.

Users may disable the automatic cutback requests in the material model through the model property *no_cutback*. If the state update process fails to converge, the model immediately terminates execution of the program.

3.5.6 Model Properties

The properties defined for material model *gurson* are listed in Table 3.5. When the *curve* option is invoked to indicate a separately defined piecewise–linear stress–strain curve, Young's modulus must still be specified. The thermal expansion coefficients are specified as described previously for the *bilinear* and *mises* models.

3.5.7 Model Output

By default, the material model prints no messages during computations. If requested, the material model prints the element number and strain point number whenever the effective stress *first* exceeds the specified yield stress. This option is requested with the nonlinear solution parameter *material messages on* (refer to Section 2.9.8). Messages about requests for global load step reductions are always printed.

The model makes available the strain energy density, U_0, to the element routines for subsequent output. U_0 at step n+1 is evaluated using the trapezoidal rule

$$U_0^{n+1} = U_0^n + \tfrac{1}{2}\left(\boldsymbol{t}^{n+1} + \boldsymbol{t}^n\right) : \Delta\boldsymbol{d} \tag{3.92}$$

where the unrotated Cauchy stresses and unrotated strain increments are adopted for the finite–strain formulation.Thermal contributions to $\Delta\boldsymbol{d}$ are subtracted prior to the above computation.

The element stress output contains up to three values for the material model "state" variables. These values for the *gurson* material are:

mat_val1:	matrix plastic strain, $\bar{\epsilon}^p$
mat_val2:	matrix equivalent stress, $\bar{\sigma}$
mat_val3:	current void fraction, f

3.5.8 Computational Efficiency

The computational routines for this model process elements in blocks of a size matched to the vector length of the computer (i.e., Crays) or to the cache size of the workstation. The majority of model computations are written in vectorized code. The local Newton loops to solve the scalar consistency equations execute in scalar mode. Our experience with these algorithms indicate that the linear (constant) hardening model requires by far the least computational effort and provides the most robustness in terms of handling large step sizes. The power-law representation of the stress-strain curve is equally robust but is much more expensive due to the sub-iterations needed to compute the uniaxial stress given an estimate for the increment of plastic strain, combined with the large number of exponential operations (the power-law model can increase total execution time by 25%). The stress-strain curve defined by piecewise linear segments provides the least robustness of the three models. Convergence often fails during iterations to resolve the consistency equation; however, the algorithms are sensitive to the number of segments (fewer is better).

Our testing also indicates the piecewise-linear model combined with the viscoplastic option can reduce the convergence rate of global Newton iterations. No such degradation is experienced with purely linear hardening or power-law hardening combined with viscoplasticity.

This model is computationally less efficient than the mises model of the previous section.

3.5.9 Example

The following example defines the properties for a mild steel material frequently used in fracture models and assigns the material to some elements.

```
structure cct
c
material a533b
  properties gurson e 30000 nu 0.3 yld_pt 60.0 n_power 10,
      rho  7.3e-07 ref_eps 40 m_power 20 f_0 0.005 killable
c
number of nodes 25642 22092
c
elements
  14000-22092 type l3disop  linear material a533b order 2x2x2,
                        long bbar
c
```

3.5.10 Plasticity Algorithms

Material Elasticity and Yield Criterion

The material is elastically isotropic and for a specified increment of total (macroscopic) strain,

$$\Delta\boldsymbol{\epsilon} = \boldsymbol{\epsilon}_{n+1} - \boldsymbol{\epsilon}_n \tag{3.93}$$

the trial (T) elastic stress state is defined by

$$\sigma^T_{n+1} = \sigma_n + \boldsymbol{D}^e : \Delta\boldsymbol{\epsilon} \ . \tag{3.94}$$

We use bold italics to denote a second–order tensor, bold roman indicates a symmetric fourth–order tensor, and : denotes the operator consistent for the order of tensors involved, e.g., $(\mathbf{C} : \boldsymbol{B})_{ij} = C_{ijkl}B_{kl}$. *Italic* symbols denote scalar variables. All tensor components are given with respect to a fixed, Cartesian system.

We define $\boldsymbol{S}^T_{n+1}$ as the deviatoric component of σ^T_{n+1} from which the equivalent (macroscopic) stress is given by

$$q^T_{n+1} = \left(\tfrac{3}{2}S^T_{ij}S^T_{ij}\right)^{1/2}_{n+1} \ . \tag{3.95}$$

Similarly, the trial hydrostatic stress is given by

$$p^T_{n+1} = -\tfrac{1}{3}\sigma^T_{n+1} : \boldsymbol{I} = -\tfrac{1}{3}(\sigma_{11} + \sigma_{22} + \sigma_{33})^T_{n+1} \ . \tag{3.96}$$

Gurson's yield function is given by

$$g = \left(\frac{q}{\bar{\sigma}}\right)^2 + 2q_1 f \cosh\left(-\frac{3}{2}\frac{q_2 p}{\bar{\sigma}}\right) - (1 + q_3 f^2) = 0 \tag{3.97}$$

where q_1, q_2, q_3 are material constants, f is the current void fraction and $\bar{\sigma}$ is the current (Mises) equivalent stress of the matrix. Most often, $q_1 = 1.5$, $q_2 = 1$ and $q_3 = q_1^2$ to match the response of discrete hole growth models under pure shear and pure hydrostatic loading. We evaluate the yield criterion for the trial elastic state using current values of the (scalar) state variables

$$g^T_{n+1} = g(q^T_{n+1}, p^T_{n+1}, f_n, \bar{\sigma}_n) \ . \tag{3.98}$$

The material loading is defined by

$$\begin{array}{ll} g^T_{n+1} < 0 & \textit{linear - elastic} \\ g^T_{n+1} \geq 0 & \textit{plastic loading} \end{array} \ . \tag{3.99}$$

Unloading from a previously plastic state is treated inelastically such that

$$\sigma_{n+1} = \sigma^T_{n+1} \tag{3.100}$$

with the internal state variables retaining their values at n.

Plasticity Rate Equations

When the material is loading plastically as indicated by Eq. (3.99), the macroscopic continuum flow rule is expressed as

$$d\boldsymbol{\epsilon}^p = d\Lambda\frac{\partial g}{\partial \sigma} \tag{3.101}$$

where $d\Lambda$ is the (positive) plastic multiplier. Integration of the plastic strain rate over the step using the backward Euler procedure yields

$$\Delta\boldsymbol{\epsilon}^p = \Delta\Lambda\left.\frac{\partial g}{\partial \sigma}\right|_{n+1} \ . \tag{3.102}$$

The derivative of the yield function in Eq. (3.102) is written in the terms of the hydrostatic and deviatoric contributions to provide

$$\Delta\boldsymbol{\epsilon}^p = \Delta\Lambda\left.\left(-\frac{1}{3}\frac{\partial g}{\partial p}\boldsymbol{I} + \frac{\partial g}{\partial q}\boldsymbol{n}\right)\right|_{n+1} \tag{3.103}$$

where the unit normal $\boldsymbol{n}$ is defined by

$$\boldsymbol{n}_{n+1} = \frac{3}{2q_{n+1}}\boldsymbol{S}_{n+1} \ . \tag{3.104}$$

To simplify subsequent expressions, we introduce definitions for the (scalar) volumetric and deviatoric plastic strain as

$$\Delta\epsilon_p = -\Delta\Lambda\left(\frac{\partial g}{\partial p}\right)\bigg|_{n+1} \tag{3.105}$$

$$\Delta\epsilon_q = \Delta\Lambda\left(\frac{\partial g}{\partial q}\right)\bigg|_{n+1} \tag{3.106}$$

and substitute into Eq. (3.103) to give

$$\Delta\boldsymbol{\epsilon}^p = \tfrac{1}{3}\Delta\epsilon_p\boldsymbol{I} + \Delta\epsilon_q\boldsymbol{n}_{n+1} \ . \tag{3.107}$$

If the updated stress state at $n+1$ is written in terms of the usual volumetric and deviatoric components

$$\boldsymbol{\sigma}_{n+1} = -p_{n+1}\boldsymbol{I} + \boldsymbol{S}_{n+1} \tag{3.108}$$

then with the notation defined by Eq. (3.104), the updated stress state also may be written in the form

$$\boldsymbol{\sigma}_{n+1} = -p_{n+1}\boldsymbol{I} + \boldsymbol{S}_{n+1} = -p_{n+1}\boldsymbol{I} + \tfrac{2}{3}q_{n+1}\boldsymbol{n}_{n+1} \tag{3.109}$$

In terms of the trial elastic stress state, the updated stress state may be constructed as follows:

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}_n + \mathbf{D}^e : (\Delta\boldsymbol{\epsilon} - \Delta\boldsymbol{\epsilon}^p) \tag{3.110}$$

where the first two terms on the right side combine to define the trial elastic state such that

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^T_{n+1} - \mathbf{D}^e : \Delta\boldsymbol{\epsilon}^p \ . \tag{3.111}$$

The negative term on the right side defines a plastic stress correction for the trial elastic stress. Using Eq. (3.107), this term may be expressed in the form

$$\mathbf{D}^e : \Delta\boldsymbol{\epsilon}^p = K\Delta\epsilon_p\boldsymbol{I} + 2G\Delta\epsilon_q\boldsymbol{n}_{n+1} \tag{3.112}$$

where K and G are the elastic bulk and shear modulus, respectively. Substitution of Eq. (3.112) into Eq. (3.111) provides a convenient form of the updated stress as

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^T_{n+1} - K\Delta\epsilon_p\boldsymbol{I} - 2G\Delta\epsilon_q\boldsymbol{n}_{n+1} \ . \tag{3.113}$$

In the above equation, the trial elastic stress state is corrected (i.e. *returned*) to the updated yield surface. In deviatoric space, the return direction is along the normal defined by $\boldsymbol{n}_{n+1}$. Using the material elasticity, we also have the following relations for the updated hydrostatic and equivalent stress

$$p_{n+1} = p^T_{n+1} + K\Delta\epsilon_p \tag{3.114}$$

$$q_{n+1} = q^T_{n+1} - 3G\Delta\epsilon_q \tag{3.115}$$

which prove very useful in the numerical processes described below.

The key step in the backward Euler scheme defines the return normal direction as the deviatoric direction of the trial elastic state as, using Eq. (3.95) and Eq. (3.104),

$$\boldsymbol{n}_{n+1} = \frac{3}{2q^T_{n+1}}\boldsymbol{S}^T_{n+1} \tag{3.116}$$

which yields finally

$$\sigma_{n+1} = \sigma^T_{n+1} - K\Delta\epsilon_p \boldsymbol{I} - \frac{3G\Delta\epsilon_q}{q^T_{n+1}}\boldsymbol{S}^T_{n+1} \ . \tag{3.117}$$

This choice for the return direction simplifies greatly numerical solution for the updated stress state; in 3–D the number of unknowns is reduced by 6, the number of unique terms in $\boldsymbol{n}_{n+1}$. A more detailed discussion of similar return mapping algorithms is given by Simo.

From Eq. (3.113), a knowledge of $\Delta\epsilon_p$, $\Delta\epsilon_q$ fully defines the updated stress state. The numerical solution must determine values for these scalar parameters so that $\Delta\boldsymbol{\epsilon}^p$ satisfies the flow rule over the step and the updated stresses satisfy the yield criterion. In the process of computing $\Delta\epsilon_p$, $\Delta\epsilon_q$, the internal state variables are updated as well.

Internal State Variables

Gurson's model includes a set of state variables which partition the macroscopic stress–strain into the matrix material and the "smeared" voids. These state variables define the microscopic plastic strain in the matrix and the current volume fraction of voids.

Plastic Strain in the Matrix

Plastic work in the matrix is taken to be a relative fraction, 1–*f*, of macroscopic plastic work such that

$$(1 - f)\bar{\sigma} d\bar{\epsilon}^p = \sigma : d\boldsymbol{\epsilon}^p \tag{3.118}$$

where $\bar{\epsilon}^p$ denotes the matrix plastic strain. This rate equation is integrated over the step using backward Euler and solved for the increment of plastic strain in the matrix

$$\Delta\bar{\epsilon}^p = \frac{\sigma_{n+1} : \Delta\boldsymbol{\epsilon}^p}{(1 - f_{n+1})\bar{\sigma}_{n+1}} \tag{3.119}$$

where the numerator simplifies considerably to provide

$$\Delta\bar{\epsilon}^p = \frac{-p_{n+1}\Delta\epsilon_p + q_{n+1}\Delta\epsilon_q}{(1 - f_{n+1})\bar{\sigma}_{n+1}} \ . \tag{3.120}$$

A variety of models for the evolution of $\bar{\sigma}$, the equivalent matrix stress, with increasing plastic strain in the matrix may be defined. Both inviscid and power–law viscoplastic models are discussed in a subsequent section.

Evolution of Void Fraction

The volume fraction of voids increases over an increment due to continued growth of existing voids and due to the formation of new voids caused by interfacial decohesion of inclusions or second phase particles. Thus,

$$df = df_{growth} + df_{nucleation} \ . \tag{3.121}$$

The growth component is defined by the current volume fraction of voids and the macroscopic change in void fraction is (the matrix material satisfies plastic incompressibility)

$$df_{growth} = (1-f)d\boldsymbol{\epsilon}^p : \underline{\boldsymbol{I}} = (1-f)d\epsilon_p \ . \tag{3.122}$$

We adopt an evolution model for nucleation based on current plastic strain in the matrix

$$df_{nucleation} = \mathcal{A}(\bar{\epsilon}^p)d\bar{\epsilon}^p \ . \tag{3.123}$$

Chu and Needleman suggest a form for $\mathcal{A}$ as

$$\mathcal{A} = \frac{f_N}{s_N\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{\bar{\epsilon}^p - \epsilon_N}{s_N}\right)^2\right] \tag{3.124}$$

where the nucleation strain follows a normal distribution with a mean value ϵ_N and a standard deviation s_N with the volume fraction of void nucleating particles given by f_N. A simpler form of Gurson's model which neglects nucleation is derived by setting $\mathcal{A} \equiv 0$ (three fewer material parameters are then required).

Equation (3.122) and its component terms are integrated using backward Euler to obtain

$$\Delta f = (1 - f_{n+1})\Delta\epsilon_p + \mathcal{A}(\bar{\epsilon}^p_{n+1})\Delta\bar{\epsilon}^p \ . \tag{3.125}$$

Equations (3.120) and (3.125) comprise a pair of coupled, nonlinear algebraic equations to update the microscopic state variables f, $\bar{\epsilon}^p$ for specified values of the macroscopic plastic strains $\Delta\epsilon_p$, $\Delta\epsilon_q$.

Response of the Matrix Material

A variety of models for the evolution of $\bar{\sigma}$, the equivalent matrix stress, may be defined. Here we consider two inviscid models, the first of which is

$$\bar{\sigma} = \sigma_0 + H'\bar{\epsilon}^p \tag{3.126}$$

where H' is the specified (constant) plastic hardening modulus (H' may be zero) and σ_0 is the specified uniaxial yield stress. The second inviscid model is a simple power–law with initially linear response

$$\frac{\bar{\epsilon}}{\epsilon_0} = \frac{\bar{\sigma}}{\sigma_0}, \qquad \bar{\epsilon} \le \epsilon_0 \tag{3.127}$$

$$\frac{\bar{\epsilon}}{\epsilon_0} = \left(\frac{\bar{\sigma}}{\sigma_0}\right)^N, \qquad \bar{\epsilon} > \epsilon_0 \tag{3.128}$$

where the total equivalent strain in the matrix, $\bar{\epsilon}$, is simply $\bar{\epsilon} = \bar{\sigma}/E + \bar{\epsilon}^p$ and $E = \sigma_0/\epsilon_0$. Eq. (3.128) is solved iteratively for $\bar{\sigma}$ with a local Newton loop for a given value of plastic strain in the matrix, $\bar{\epsilon}^p$. The plastic modulus, H', is then found by

$$H' = \frac{EE_T}{E - E_T} \tag{3.129}$$

where the tangent modulus is defined from Eq. (3.128) by

$$E_T = \frac{E}{N}\left(\frac{\bar{\sigma}}{\sigma_0}\right)^{(1-N)} \ . \tag{3.130}$$

To model a viscoplastic matrix material, we adopt a power–law model of the form

$$\dot{\bar{\epsilon}}^p = \frac{1}{\eta}\left[\left(\frac{\bar{\sigma}}{\sigma_e}\right)^m - 1\right] \tag{3.131}$$

where η and m are material constants and σ_e is the inviscid equivalent stress for the matrix. The viscosity term is often written in the form $D=1/\eta$. In the simplest case, σ_e is specified to remain constant at the yield stress, σ_0. More generally, σ_e is a nonlinear function of $\bar{\epsilon}^p$ along the lines of Eq. (3.128).

The integration of Eq. (3.131) with a backward Euler procedure yields

$$\Delta\bar{\epsilon}^p = \frac{\Delta t}{\eta}\left[\left(\frac{\bar{\sigma}_{n+1}}{\sigma_{i,n+1}}\right)^m - 1\right] \tag{3.132}$$

where subscript i denotes the inviscid response at the same plastic strain in the matrix. This expression is solved directly for $\bar{\sigma}_{n+1}$

$$\bar{\sigma}_{n+1} = \sigma_{i,n+1}\left[\left(\frac{\eta\Delta\bar{\epsilon}^p}{\Delta t}\right) + 1\right]^{1/m} . \tag{3.133}$$

We observe in Eq. (3.133) that as $\eta/\Delta t \rightarrow 0$ the inviscid solution is recovered. Each of the above models for $\bar{\sigma}_{n+1}$ are functions of the plastic strain in the matrix and can thus be resolved during the solution for $\Delta\epsilon_p$, $\Delta\epsilon_q$. The plastic modulus is given by

$$H' = \left.\frac{d\bar{\sigma}}{d\bar{\epsilon}^p}\right|_{n+1} = \frac{\eta\sigma_i}{m\Delta t}\left(\frac{\bar{\sigma}}{\sigma_i}\right)^{1-m} + \left(\frac{\bar{\sigma}}{\sigma_i}\right)H_i' \tag{3.134}$$

where all terms on the RHS of (3.134) are evaluated at n+1.

Summary of Updating Process

The stress updating process requires computation of a set of stresses defined by Eq. (3.117) for which the flow conditions given in Eq. (3.105) and Eq. (3.106) are satisfied consistent with updated values of the internal state variables. The proportionality factor $\Delta\Lambda$ is eliminated by dividing Eq. (3.105) by Eq. (3.106) to define the relationship between the increments of volumetric and deviatoric plastic strain as

$$\Delta\epsilon_p\left(\frac{\partial g}{\partial q}\right) + \Delta\epsilon_q\left(\frac{\partial g}{\partial p}\right) = 0 . \tag{3.135}$$

This relationship together with satisfaction of the yield criterion at n+1 using stresses of Eq. (3.117)

$$g_{n+1} = g(q_{n+1}, \bar{\sigma}_{n+1}, p_{n+1}, f_{n+1}) = 0 \tag{3.136}$$

defines a pair of nonlinear algebraic equations for numerical solution. The primary unknown variables in these two equations are the macroscopic plastic strains $\Delta\epsilon_p$, $\Delta\epsilon_q$.

These equations are solved iteratively using Newton's method. Given estimates for $\Delta\epsilon_p$ and $\Delta\epsilon_q$, the updated stress state, p_{n+1} and q_{n+1}, are given by Eq. (3.114) and Eq. (3.115). The internal state variables, $\bar{\epsilon}^p$, $\bar{\sigma}$ and f, are updated to n+1 by solving these three equations simultaneously, Eqs. (3.120) and (3.125) are repeated for clarity)

$$\Delta\bar{\epsilon}^p = \bar{\epsilon}^p_{n+1} - \bar{\epsilon}^p_n = \frac{-p_{n+1}\Delta\epsilon_p + q_{n+1}\Delta\epsilon_q}{(1-f_{n+1})\bar{\sigma}_{n+1}} . \tag{3.137}$$

$$\Delta f = f_{n+1} - f_n = (1-f_{n+1})\Delta\epsilon_p + \mathcal{A}(\bar{\epsilon}^p_{n+1})\Delta\bar{\epsilon}^p . \tag{3.138}$$

$$\bar{\sigma}_{n+1} = \bar{\sigma}(\bar{\epsilon}^p_{n+1}) \tag{3.139}$$

The numerical complexity in updating $\bar{\varepsilon}^p$ and f depends on the form adopted for $\bar{\sigma}(\bar{\varepsilon}^p)$ and whether or not the nucleation component of f is included. If $\bar{\sigma}(\bar{\varepsilon}^p)$ of the form defined by Eq. (3.126) is adopted and $\mathcal{A} \equiv 0$, the above three equation reduce to a single linear equation for $\Delta\bar{\varepsilon}^p$ after which Δf is found directly as well. In other cases, another level of Newton's iterations is required to resolve $\Delta\bar{\varepsilon}^p$ and Δf consistent with $\bar{\sigma}$.

Chapter 4

Domain Integrals

4.1 Introduction

Finite element methods are especially powerful for computing linear and nonlinear fracture mechanics parameters. For linear analyses, the stress-intensity factors, K_I, are readily determined from the energy release-rate interpretation of the J-integral (Rice [75], Knowles and Sternberg [53], Budiansky and Rice [10]). For nonlinear analyses, the intensity of deformation along the crack front is generally characterized by the Crack Tip Opening Displacement (CTOD) and/or a pointwise value of the J-integral. In two-dimensions, the J-integral sets the amplitude of the singular field near a sharp crack tip, as given by the HRR solutions (Rice and Rosengren [74], Hutchinson [45]), under certain limiting conditions involving material constitutive behavior and the extent of plastic deformation relative to the uncracked ligament size. In three-dimensions, the situation is not nearly so clear; the nature of near-tip fields in 3-D remains a focus of current research. Remote from traction free surfaces, the crack front fields may closely resemble those of plane-strain; near free surfaces the fields exhibit strong 3-D effects. However, purely mechanical arguments concerning the energy flux show that the J-integral provides a local energy release rate independent of the exact singular form of the near tip fields. Under these conditions, J characterizes the *crack driving force*.

This chapter describes the Domain Integral (DI) capabilities implemented in WARP to compute J-integral values in 3-D (Mode I) following the solution for a load step (Li, et al. [55], Moran and Shih [60][61], Shih, et al. [81]). The DI procedures are more general and simpler for the analyst to specify than the earlier Virtual Crack Extension (VCE) technique (Parks [72], Helen [34]). The analyst defines nodal values of a *weight function* which may be interpreted as the motion of material near the crack front due to a *virtual* crack extension. The numerical computations then require evaluation a volume integral over elements in 3-D which includes the energy density, the stress field, the displacement, velocity, acceleration fields and the weight functions. Weight functions over elements are constructed from the specified nodal values using conventional isoparametric procedures. This quickly becomes an onerous task; however, capabilities are included for automatic generation of the weight function values which greatly simplify J computations in 3-D crack configurations. An option for the user to specify directly the weight function values on a node-by-node basis remains available.

The procedures described in this chapter may be invoked following a linear or nonlinear solution for a load step (static/dynamic). The user provides input commands to define a "domain" for evaluation of J followed by a *compute domain integral* command. The specification of a single "automatic" domain by the user typically causes J evaluations over many separate domains of increasing distance from the crack front. The computed J-value for each domain and the variations J-values between the domains are printed (minimum J, maximum J and average J for assessment of path independence).

The DI procedures currently implemented have these features/limitations:

- the material response is considered nonlinear elastic when the material model employs an incremental plasticity theory (this is a very common assumption and avoids unnecessary complications that arise from the explicit partial derivative of the stress work density)

- the kinetic energy and accelerations of crack region material in dynamic loading are included in J
- the effects of finite strains and finite rotations at material points are included in J
- the effects of *rapid* crack growth are not included in J ("slow" crack growth under dynamic loading is supported)
- the effects of user specified loads applied to the crack faces are included in J using an approximate technique (these terms maintain path independence for domains remote from the front). The J processor cannot properly distinguish between simultaneously applied crack face loads and temperature loads (for crack face elements)
- initial strains caused by imposed thermal loading are included in J
- body forces, other than caused accelerations, are ignored during J computations

The next section of this chapter provides a summary of the theoretical basis for the DI method. Other sections describe the numerical algorithms to evaluate the volume integrals and input commands. Sample output from a computation illustrates the various information available.

4.2 Background

4.2.1 Local Energy Release Rates

A *local* value of the mechanical energy release rate, denoted $J(s)$, at each point s on a planar, non-growing crack front under general dynamic loading is given by

$$J(s) = \lim_{\Gamma \to 0} \int_{\Gamma} \left[(W + T)n_1 - P_{ji}\frac{\partial u_i}{\partial X_1} n_j \right] d\Gamma \tag{4.1}$$

where W and T are the stress-work density and the kinetic energy density per unit volume at $t = 0$; Γ is a vanishingly small contour which lies in the principal normal plane at s, and n is the unit vector normal to Γ (see Fig. 4.1). P_{ji} denotes the non-symmetric 1st Piola-Kirchhoff (1st PK) stress tensor which is work conjugate to the displacement gradient expressed on the $t = 0$ configuration, $\partial u_i / \partial X_j$, i.e., the stress-work rate is simply $P_{ij} \partial u_i / \partial X_j$ per unit volume at $t = 0$. All field quantities are expressed in the local orthogonal coordinate system, $X_1 - X_2 - X_3$, at location s on the crack front.

This important result was first derived by Eshelby [23] and independently by Cherepanov [13], and later by others considering only mechanical energy balance for a local translation of the crack front in the X_1 direction (Mode I). Any form of loading (including crack face tractions) and arbitrary material behavior is permitted when $\Gamma \to 0$. All proposed forms of path independent integrals (contour, area, volume) for application in fracture mechanics derive from Eq. (4.1) by specialization of the loading and material behavior (see for example, Amestoy et al. [1], Bakker [5], Carpenter et al. [12], de Lorenzi [19] and Kishimoto et al. [50]).

Moran and Shih [60][61] have proven the *local* path independence of J on the actual shape of Γ in the limit as $\Gamma \to 0^+$. To have both path independence and a non-vanishing, finite value, the integrand of Eq. (4.1) must have order $1/r$. The quantity J defined by Eq. (4.1) has no direct relationship to the form of the near-tip strain-stress fields, except for very limited circumstances. For plane-stress and plane-strain conditions, with nonlinear elastic material response and small-strain theory, J of Eq. (4.1) simplifies to the well-known J-integral due to Rice [75] that exhibits *global* path independence. Under the additional limitation of small-scale yielding (SSY), J sets the amplitude of the HRR singular fields. The role of J as a single parameter which characterizes the near tip strain-stress fields for arbitrary loading (static, thermal, dynamic) and 3-D configurations is a topic of much current research.

The stress-work density (W) per unit initial volume may be defined in terms of the mechanical strains as

$$W = |\boldsymbol{F}| \int_0^t \boldsymbol{t} : \left(\boldsymbol{d} - \boldsymbol{d}^{th} \right) dt \tag{4.2}$$

where $|\boldsymbol{F}|$ denotes the determinant of the deformation gradient $\boldsymbol{F} = \partial \boldsymbol{x} / \partial \boldsymbol{X}$, $\boldsymbol{t}$ denotes the unrotated Cauchy stress and $\boldsymbol{d}$ is the unrotated rate of deformation tensor computed from the displacement gradients. $\boldsymbol{d}^{th}$ denotes the contribution arising from specified thermal strains. The kinetic energy density is given directly by

$$T = \frac{1}{2}\rho \left(\frac{\partial u_i}{\partial t} \right)^2 \tag{4.3}$$

where ρ is the material mass density (sum on i) in the initial configuration at $t = 0$.

The direct evaluation of Eq. (4.1) is cumbersome in a finite element model due to the geometric difficulties encountered in defining a contour that passes through the integration points. Such a contour is desired since the most accurate stress and strain quantities are available at the integration points. Moreover, the limiting definition of the contour requires extensive mesh refinement near the crack tip to obtain meaningful numerical results. The next section develops the Domain Integral equivalent of Eq. (4.1) which is naturally suited for finite element models.

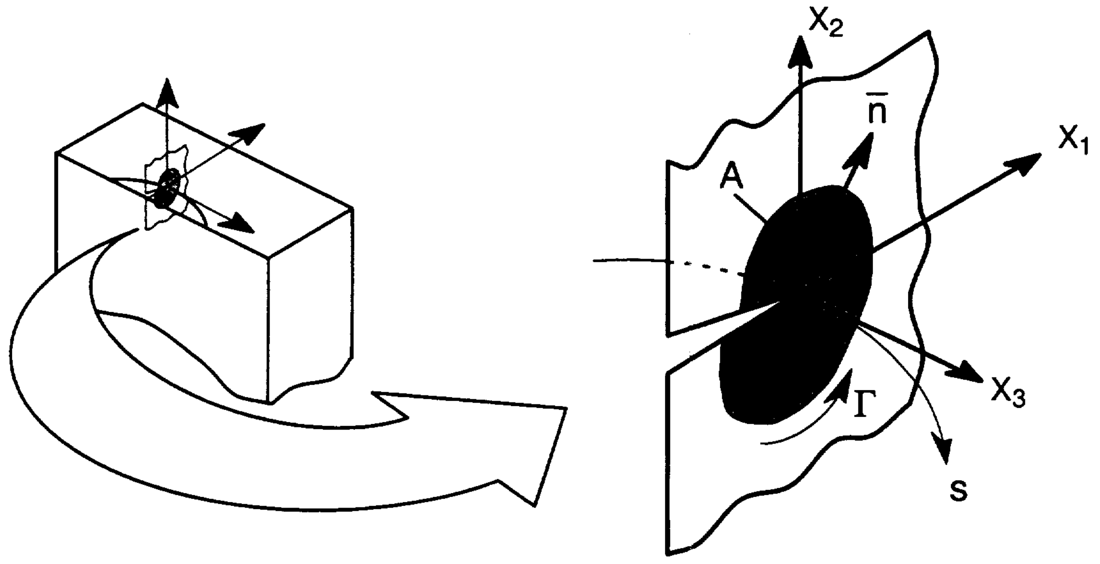


FIG. 4.1—*Local J-integral in 3-D.*

4.2.2 Domain Integral Formulation

By using a weight function, which may be interpreted as a virtual displacement field, the contour integral of Eq. (4.1) is converted into an area integral for two dimensions and into a volume integral for three dimensions (Li, et al. [55], Nikishkov and Atluri [70]). The resulting expressions are (see Fig. 4.2):

$$\bar{J}_{a-c} = \int_{s_a}^{s_c} [\, J(s)\, q_t(s)\,]\, ds = \bar{J}_1 + \bar{J}_2 + \bar{J}_3 \tag{4.4}$$

where each integral is defined by

$$\bar{J}_1 = \int_{V_0} \left(P_{ji} \frac{\partial u_i}{\partial X_k} \frac{\partial q_k}{\partial X_j} - W \frac{\partial q_k}{\partial X_k} \right) dV_0 \tag{4.5}$$

$$\bar{J}_2 = -\int_{V_0} \left(\frac{\partial W}{\partial X_k} - P_{ji} \frac{\partial^2 u_i}{\partial X_j \partial X_k} \right) q_k \, dV_0 \tag{4.6}$$

$$\bar{J}_3 = -\int_{V_0} \left(T \frac{\partial q_k}{\partial X_k} - \rho \frac{\partial^2 u_i}{\partial t^2} \frac{\partial u_i}{\partial X_k} q_k + \rho \frac{\partial u_i}{\partial t} \frac{\partial^2 u_i}{\partial t \partial X_k} q_k \right) dV_0 \tag{4.7}$$

q_k denotes a component of the vector weight function in the k coordinate direction, $q_t(s)$ represents the resultant value of the weight function at point s on the crack front, V_0 represents the volume of the domain surrounding the crack tip in the (undeformed) configuration at $t = 0$, and s denotes positions along the crack front segment.

The vector function q is directed parallel to the direction of crack extension. When all field quantities of the finite element solution are transformed to the local crack front coordinate system at point s, and Mode I extension is considered, only the q_1 term of the weight function is non-zero. In subsequent discussions, this transformation to the (local) crack front coordinate system is assumed to hold; the k subscript on q terms is thus dropped with q alone implying the q_1 term.

Body forces (other than inertial loading) are assumed to be zero for simplicity. The treatment of crack face tractions involves an additional integral discussed subsequently. $J(s)$ is the local energy release rate that corresponds to the perturbation at s, $q_t(s)$. Figure 4.2 shows a typical domain volume defined for an internal segment along a three-dimensional surface crack.

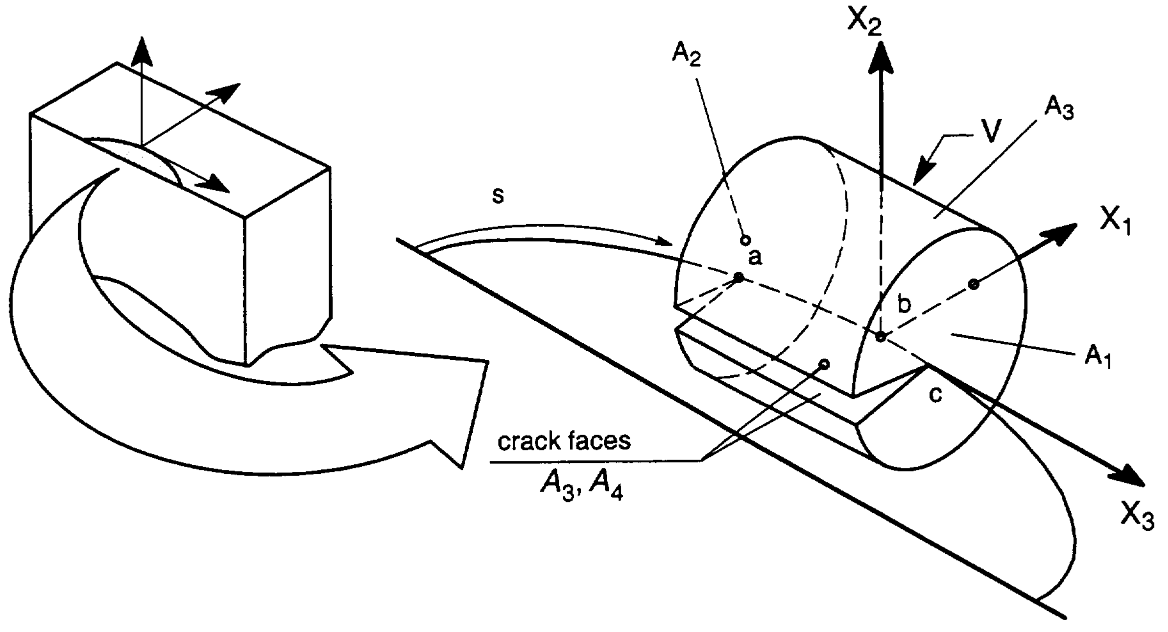


FIG. 4.2—*Finite volume for use in Domain Integral formulation*

The q-function must vanish on the surfaces A_1, A_2 and A_3 in Fig. 4.2 for the development of Eqs. (4.5) through (4.7) from (4.4). This requirement makes area integrals (line integrals in two dimensions) defined on these surfaces vanish. Fig. 4.3 shows the variation in amplitude of a valid q-function for the domain shown in Fig. 4.2. All material over which the q-function and its first derivative are non-zero must be included in the volume integrals. The value of q at each point in the volume, V_0, is readily interpreted as the virtual displacement of a material point due to the virtual extension of the crack front, $q_t(s)$.

An approximate value of $J(s_b)$ is obtained by applying the mean-value theorem over the interval $s_a < s < s_c$. The pointwise value of the J-integral at s_b is given by (see Fig. 4.3):

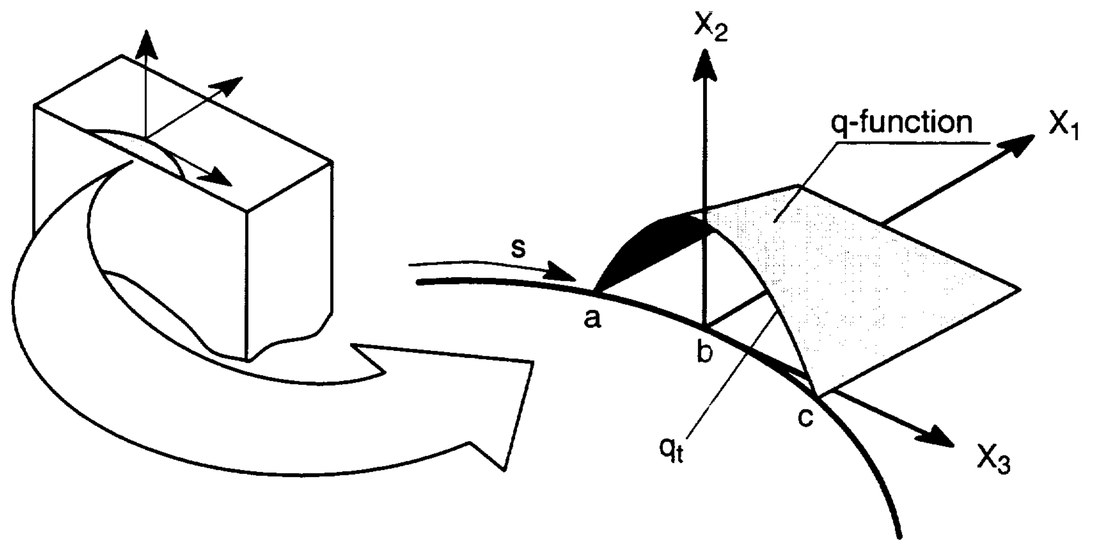


FIG. 4.3—*Variation of weight function, q, over volume at crack front*

$$J(s=b) \approx \frac{\int_{s_a}^{s_c} J(s)\, q_t(s)\, ds}{\int_{s_a}^{s_c} q_t(s)\, ds} = \frac{\bar{J}}{A_q} \tag{4.8}$$

where $\bar{J}$ is the energy released due to the crack-tip perturbation, $q_t(s)$. The increase in crack-area corresponding to this perturbation, A_q, is simply the integral of $q_t(s)$ along the crack front from s_a to s_c.

For common through crack test specimens, e.g. SE(B), C(T), the crack front is generally straight or only slightly curved. For such crack geometries, the average J for the entire crack front value is obtained by the application of a uniform $q_t(s)$ across the full crack front.

The above volume integrals are evaluated by Gauss quadrature. Derivatives of the q-function over each finite element in V are computed by standard isoparametric techniques from specified values of q at element nodes. The higher order derivatives are computed by either: 1) extrapolating Gauss point values to the element nodes and applying standard isoparametric techniques or, 2) interpolating the Gauss point values to a lower order integration within the element.

4.2.3 Domain Form of the J-Integral: Discussion

Thermal Loading

The $\bar{J}_2$ integral vanishes for an elastic material (linear or nonlinear) in the absence of thermal strains as shown in the following manner (using small displacement gradient theory for simplicity). Begin by replacing the 1st PK stresses with the conventional (symmetric) stress tensor applicable when strains and displacement gradients are small. Then

$$-P_{ji}\frac{\partial^2 u_i}{\partial X_j \partial X_1} \approx -\sigma_{ij}\frac{\partial^2 u_i}{\partial X_j \partial X_1} \ . \tag{4.9}$$

After exchanging the order of differentiation, inserting the (symmetric) small-strain tensor and using symmetry of σ_{ij}, the second term in Eq. (4.6) is rewritten as:

$$-\sigma_{ij}\frac{\partial^2 u_i}{\partial X_j \partial X_1} = -\sigma_{ij}\frac{\partial}{\partial X_1}\left(\frac{\partial u_i}{\partial X_j}\right) = -\sigma_{ij}\frac{\partial \epsilon_{ij}}{\partial X_1} . \tag{4.10}$$

The chain rule is now evoked to expand the first term in Eq. (4.6), again assuming small-displacement gradients. The derivative of strain energy density with respect to strain is the stress tensor for *elastic* materials. The result is:

$$\frac{\partial W}{\partial X_1} = \frac{\partial W}{\partial \epsilon_{ij}}\frac{\partial \epsilon_{ij}}{\partial X_1} = \sigma_{ij}\frac{\partial \epsilon_{ij}}{\partial X_1} . \tag{4.11}$$

The two terms defining the integrand of $\bar{J}_2$ thus sum to zero for elastic materials when thermal strains are absent.

Now consider the influence of initial strains caused imposed thermal loading, again using small strain theory for simplicity. Equation (4.10) remains unchanged; however, Eq. (4.11) must be re-written more explicitly as

$$\frac{\partial W}{\partial X_1} = \frac{\partial W}{\partial \epsilon^e_{ij}}\frac{\partial \epsilon^e_{ij}}{\partial X_1} = \sigma_{ij}\frac{\partial \epsilon^e_{ij}}{\partial X_1} = \sigma_{ij}\frac{\partial}{\partial X_1}\left(\epsilon_{ij} - \epsilon^{th}_{ij}\right) \tag{4.12}$$

where the *total* strain is now given by elastic (including nonlinear) and thermal components such that $\epsilon_{ij} = \epsilon^e_{ij} + \epsilon^{th}_{ij}$. Upon combining Eqs. (4.10) and (4.12), we have

$$\bar{J}_2 = \int_{V_0} \sigma_{ij}\frac{\partial \epsilon^{th}_{ij}}{\partial X_1} q \, dV_0 . \tag{4.13}$$

To simplify numerial implementation, re-write the thermal strains as $\epsilon^{th}_{ij} = \alpha_{ij}\Theta$, where Θ denotes the temperature change. The above expression becomes

$$\bar{J}_2 = \int_{V_0} \sigma_{ij}\left[\alpha_{ij}\frac{\partial \Theta}{\partial X_1} + \frac{\partial \alpha_{ij}}{\partial X_1}\Theta\right] q \, dV_0 . \tag{4.14}$$

which somewhat simplifies implementation in a finite element context since temperatures are known at element nodes. Computation of the Cartesian temperature derivative follows standard finite element procedures. Derivatives of the thermal expansion coefficients with respect to the crack direction (X_1) must be obtained and this does complicate the computations. Also, the symmetric tensor of thermal expansion coefficients, α_{ij}, require transformation into crack front coordinates. For materials with constant thermal expansion coefficients within the domain of integration (V_0), the above (kernel) expression simplifies further to just $q\alpha\sigma_{ij}\partial\Theta/\partial X_1$.

Dynamic Effects

Dynamic loading effects appear in the $\bar{J}_3$ term of the domain integral representation of the J-integral. The first term in $\bar{J}_3$ provides the flux of the kinetic energy in the direction of the crack propagation. The second and third terms arise from the explicit partial derivative, $(\partial[\,]/\partial X_1)$, of the kinetic energy density. The second term contains material accelerations and the third term is identified with the spatial gradient of the velocities. The second term, containing the material accelerations, has been found to make significant contributions to the total $\bar{J}$-integral for non-propagating cracks. This term is similar in form to domain integrals that accommodate ordinary body forces.

Incremental Plasticity Effects

For an elastic structure under static loading (without any thermal strains), $\bar{J}_2$ and $\bar{J}_3$ are identically zero. For incremental (load path dependent) plasticity, the deviation of $\bar{J}_2$ from zero indicates the degree of non-proportional loading experienced over the domain of integration.

For many practical cases, the loading produces nearly proportional material histories within the domain of integration; in such cases the very small contribution of $\bar{J}_2$ is neglected. Shih, Moran and Nakamura [81] neglected $\bar{J}_2$ for J-integral calculations. Vargas and Dodds show that up to 15% of the J-integral in a 2-dimensional static case can be due to $\bar{J}_2$ for incremental plasticity models when the plastic strains and the elastic strains within the domain are similar in magnitude. For larger plastic strains, however, this difference diminishes to less than 0.1%, which justifies the use of $\bar{J}_2 + \bar{J}_3$ as an approximation to Eq. (4.4) for large amounts of plastic deformation. However, the contribution of $\bar{J}_2$ in the presence of thermal strain gradients within the integration domain can be essential to maintain domain independence of computed J-values.

The derivation of Eqs. (4.4) through (4.7) is mathematically rigorous. Provided sufficient resolution of the crack-tip stress-strain fields exists for accurate numerical integration, the calculated $\bar{J}$-integral equals the weighted $J(s)$, where $J(s)$ is the contour definition in the limit as the contour shrinks onto the crack tip. For a given $q_t(s)$, i.e., the crack front variation of the weighting function, many combinations of domain volume and distribution of the q-function are possible. Thus, similar to *path independence* arguments for the contour J-integral, *domain independence* arguments apply for the domain J-integral. In practice, several domains defined concentrically about the crack tip are evaluated to insure domain independence of the computed J-integral. In the general case of thermal loading and inelastic material response all three components of the J-integral are required for the calculated value to be domain independent.

Summary

Numerical evaluation of the $\bar{J}_1$ integral requires only straightforward application of isoparametric element techniques once the computed field quantities are transformed from the global $X-Y-Z$ coordinate system to the $X_1 - X_2 - X_3$ (local crackfront) system at point s on the front. In this simplified form, Eq. (4.5) becomes

$$\bar{J}_1 = \int_{V_0} \left(P_{ji} \frac{\partial u_i}{\partial X_1} \frac{\partial q}{\partial X_j} - W \frac{\partial q}{\partial X_1} \right) dV_0 \,. \tag{4.15}$$

$\bar{J}_2$ makes a non-zero contribution in the WARP3D implementation only in the presence of thermal strains (finite strain form with symmetric thermal expansion coefficients)

$$\bar{J}_2 = \int_{V_0} \sigma_{ij} \left[\alpha_{ij} \frac{\partial \Theta}{\partial X_1} + \frac{\partial \alpha_{ij}}{\partial X_1} \Theta \right] q \, dV_0 \,. \tag{4.16}$$

The kinetic energy and inertial loading terms from Eq. (4.7) become

$$\bar{J}_3 = - \int_{V_0} \left(T \frac{\partial q}{\partial X_1} - \rho \frac{\partial^2 u_i}{\partial t^2} \frac{\partial u_i}{\partial X_1} q + \rho \frac{\partial u_i}{\partial t} \frac{\partial^2 u_i}{\partial t \partial X_1} q \right) dV_0 \tag{4.17}$$

WARP3D domain integral processors evaluate only the first two terms of this integral. The third term (velocity) is vanishing small unless high speed crack propagation takes place.

A_q may be interpreted as area of crack extension represented by a virtual crack extension q. The value of A_q is defined by

$$A_q = \int_{s=a}^{s=c} q(s)\, ds \tag{4.26}$$

which is numerically evaluated using Gauss quadrature as

$$A_q = \sum_p \sum_I N_I(s_p)\, q_I \left[\sqrt{dX_1^2 + dX_2^2}\,\right]_p w_p \tag{4.27}$$

where the functional form of q over the segment of crack front under consideration, $a \le s \le c$, is specified by the user to vary in a piecewise linear, parabolic or cubic manner. Lagrangian interpolating functions, $N_I(s)$, are used to construct the piecewise functions for q along the crack front. The length of crack front over $a \le s \le c$ is computed with the expression

$$L = \sum_p \left[\sqrt{dX_1^2 + dX_2^2}\,\right]_p w_p \tag{4.28}$$

and is displayed for checking purposes.

4.3.6 Output From Computations

The printed output displayed during Domain Integral computations is organized in a hierarchial manner at the load step for the user specified domains. By default, only the results for each complete domain are printed; an option to print contributions for each element is available. The values printed for each domain (or element in a domain) are labeled *DM1* through *DM6* and correspond to the terms in Eqs. (4.15) through (4.18) as follows

$$DM_1 = -\int_{V_{e(0)}} W \frac{\partial q}{\partial X_1}\, dV_0 \tag{4.29}$$

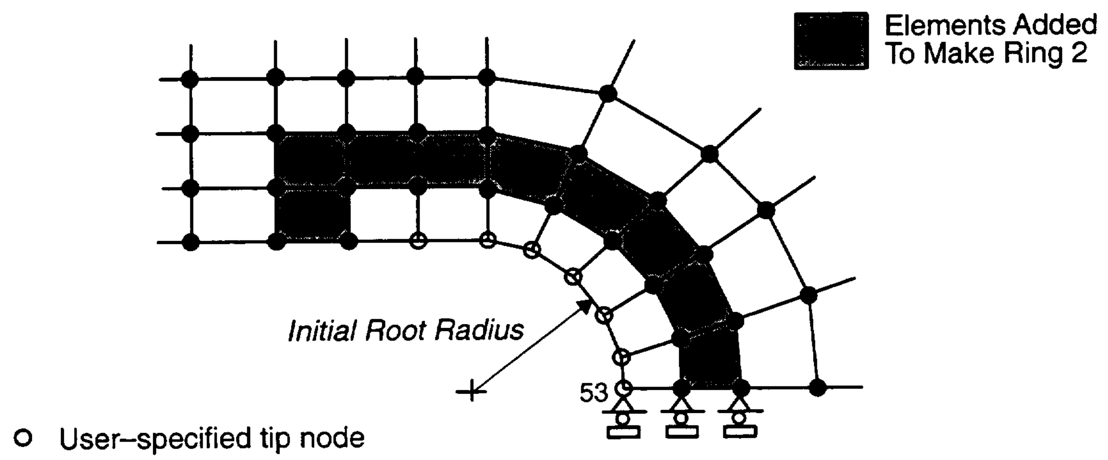


FIG. 4.4—*Typical blunt-tip model employed in finite-strain analyses*

$$DM_2 = \int_{V_{e(0)}} P_{ji} \frac{\partial u_i}{\partial X_1} \frac{\partial q}{\partial X_j} \, dV_0 \tag{4.30}$$

$$DM_3 = - \int_{V_{e(0)}} T \frac{\partial q}{\partial X_1} \, dV_0 \tag{4.31}$$

$$DM_4 = \int_{V_{e(0)}} \rho \frac{\partial^2 u_i}{\partial t^2} \frac{\partial u_i}{\partial X_1} q \, dV_0 \tag{4.32}$$

$$DM_5 = - \int_{A_3 + A_4} t_i \frac{\partial u_i}{\partial X_1} q \, dA_0 \tag{4.33}$$

$$DM_6 = \int_{V_{e(0)}} \sigma_{ij} \left[\alpha_{ij} \frac{\partial \Theta}{\partial X_1} + \frac{\partial \alpha_{ij}}{\partial X_1} \Theta \right] q \, dV_0 \ . \tag{4.34}$$

The sum of these integrals over all elements of the domain is displayed followed by A_q, the area under the q-function along the crack front. The J-integral value is printed as the sum of the integrals divided by A_q. The units of J are $F\text{-}L/L^2$.

The average, maximum, and minimum J values are summarized in tabular form. Separate sums are also printed for static and dynamic contributions.

4.4 Commands for Domain Integrals

4.4.1 Outline of Process

Once the analysis completes for the list of load steps appearing in the current *compute displacements* command, WARP command processors read the next data line. This can be an *output* command, another *compute* command or a *domain* command (as well as a number of other valid commands).

The *domain* command initiates the input sequence to specify information about a domain for computation of the J-integral. Following specification of a valid domain, the input command *compute domain integral* invokes the domain integral processors to perform the computations using analysis results for the most recent (current) load step analyzed.

To evaluate J over different domains using results for the current load step, simply repeat the *domain ... compute domain integral* sequence as often as desired. WARP stores only the definition of the most recently defined domain. When J is evaluated using the same domain definitions at many load steps, the **input from file* command proves convenient to eliminate repetition. The *domain* definitions and *compute domain integral* commands are defined in a separate input file and simply referenced with the **input from file* feature of WARP.

At completion of domain integral computations, other commands may be given to compute displacements for additional steps, request other output, alter solution parameters, etc.

4.4.2 Input Error Correction

The processor of domain integral commands recovers easily from most syntax errors. Messages indicating the error are displayed and a new input line read; simply re-enter the corrected form of the command. The new information overwrites previous values.

The input processor performs immediate checks for obvious errors in the specified data. More extensive consistency checking of the domain definition occurs during the actual numerical computations.

4.4.3 Components of a Domain Definition

Each domain for J computation consists of the following information:

1. The alphanumeric name (id) of the domain as specified in a *domain* command.
2. Components of a unit vector normal to the crack plane.
3. A symmetry flag, if applicable. J-values are then doubled prior to printing.
4. A list of nodes defining a portion of the crack front under consideration. Note that all elements along the crack front must be of the same type: *l3disop* or *q3disop*.
5. q-values at nodes along the portion of the crack front under consideration and over the desired volume of domain integration. Two methods to specify nodal values of q are available: user-defined and automatic.
    a. User defined—users specify actual nodal values for q and the list of elements over which the domain integration is desired. A single J-value is printed.
    b. Automatic—users specify the number of concentric *rings* of elements enclosing the tip over which J is evaluated at the crack front position. The q-values and lists of elements are generated automatically by WARP domain processors. A J-value is printed for each *ring* of elements requested.

6. Printing options. By default the total DM_i values are printed for the domain (each ring if automatic); individual element contributions are not printed. Users may request printing of individual element values as well.
7. Order of Gauss quadrature for element volume integrals. The default integration order is that used for element stiffness computation. A one-point rule is an optional order.
8. Crack face loadings option. By default, contributions to J from elements with detectable crack face loading are included. An option is available to neglect crack face loading contributions. This option is needed for crack growth analyses in which the crack closing forces are slowly relaxed to zero behind the extending front. These forces are interpreted by the domain processors as equivalent loads for crack face loading.
9. Debug output options. Two levels of debugging information may be requested.
10. Verification of domain input. A "dump" option prints the definition of domain parameters from internal storage.

4.4.4 Initiating a Domain Definition

The command to initiate a new domain has the form

```
domain    < name: label >
```

where the domain name appears as a descriptor in printed output.

4.4.5 Crack Plane Orientation

The orientation of the local crack front system, X_1-X_2-X_3, shown in Fig. 4.3 must be specified. The user defines components of a unit vector normal to the crack plane (X_1-X_3) aligned in the positive direction of X_2. WARP then determines the direction of X_3 using the list of crack front nodes (the positive direction of X_3 is in the direction from the first node to the second node in the list). The direction X_1 is found from the cross product $X_2 \otimes X_3$.

The command to define crack plane normals has the form

$$\underline{\text{normal}}\ (\underline{\text{plane}}) \left[\begin{Bmatrix} \underline{\text{nx}} \\ \underline{\text{ny}} \\ \underline{\text{nz}} \end{Bmatrix} < \text{direction cosine: number} > \right]$$

where nx, for example, defines the projection of the crack plane (unit) normal onto the global X axis. If the global Z axis is normal to the crack plane, for example, use the command

```
normal plane nz 1.0
```

The direction cosines provided in the command must define a vector of unit length ($nx^2 + ny^2 + nz^2 \equiv 1$).

When the J-values are negative but have the correct absolute value, reverse the sense of the crack plane normal vector.

This combined procedure in which the user specifies the X_2 direction and the domain processors use the front node list to compute directions for X_1-X_3 naturally fits the pointwise computation of J along a curved crack front. Similarly, a thickness-average J-value for a slightly curved or straight crack front in a through crack configuration is easily obtained with the automatic method of q specification. Note, however, that the X_3 direction for the domain is defined by the first two nodes given in the front node list.

4.4.6 Symmetric Option

The *symmetric* option is provided as a convenience since many finite element models are defined for symmetric geometries, loading and constraints When this keyword is specified, all J-values are double prior to printing. An output message signals when J-values are doubled as well.

The command to request doubling of J-values for symmetry has the form

symmetric

4.4.7 Crack Front Nodes

Fronts Defined by Collapsed Elements

The command to define nodes on the crack front for the domain has the form

$$\underline{\text{front}}\ \text{(nodes)} < \text{integerlist} > \left\{ \begin{array}{c} \underline{\text{l3disop}} \\ \underline{\text{q3disop}} \end{array} \right\} \quad (\underline{\text{verify}})$$

where the ordering of front nodes in the list must follow increasing X_3. The type of elements along the crack front (*l3disop* or *q3disop*) must be specified to support error checking. When *q3disop* elements are used along the crack front, the number of front nodes listed must always be an odd number (3, 5, 7, ...).

When the crack front is modeled with collapsed elements, there are multiple coincident nodes at locations along the front. Only one of the coincident nodes should be specified at these locations in this command. The remaining coincident nodes are located automatically and included in subsequent processing. A list of the other nodes coincident with each front node specified in this command is printed if the keyword *verify* appears as the last item of the command.

To illustrate the use of this command, consider the curved crack front sketched in Fig. 4.5. Crack front elements are linear isoparametrics (*l3disop*). Let node 10 lie on a symmetry plane; node 22 lies on the outside (free) surface. To compute J at node 10 on the front, the crack front segment in the domain includes nodes 10 and 14. The input command is

```
front nodes 10 14 l3disop verify
```

To compute J at node 14, the crack front segment in the domain includes nodes 10, 14 and 18. The input command is

```
front nodes 10 14 18 l3disop verify
```

Here, q varies linearly (piecewise) along the front between nodes 10, 14 and 18 (q will be zero at 10 and 18 and > 0 at 14).

Fronts Defined With Initial Root Radius

The user first defines lists of "tip" nodes for each crack front position needed in the domain using commands of the form

$$\underline{\text{node}}\ \underline{\text{set}} <\text{set id:integer}> \quad < \text{integerlist} >$$

where the *set id* is simply a convenient identifier in the range of 1-30 for later reference. If the mesh has 9 elements defined along the crack front, for example, 10 such lists of tip nodes are usually defined. The first node appearing in the < integerlist > should be the actual front node on the symmetry plane, i.e., node 53 in Fig. 4.4. The correspondence between the *set id* and a specific crack front position is made explicit in the modified *front nodes* command.

The command to define nodes on the crack front for the domain has the form

$$\underline{\text{front}}\ \text{(node)}\ \underline{\text{se}}\text{ts}\ <\text{integerlist}>\ \left\{ \begin{array}{c} \underline{\text{l3disop}} \\ \underline{\text{q3disop}} \end{array} \right\}$$

where the ordering of front node sets in the list must follow increasing X_3. The type of elements along the crack front (*l3disop* or *q3disop*) must be specified to support error checking. When *q3disop* elements are used along the crack front, the number of front sets listed must always be an odd number (3, 5, 7, ...).

To illustrate the use of this command, consider again the curved crack front sketched in Fig. 4.5. Crack front elements are linear isoparametrics (*l3disop*). Let node set 1 lie on a symmetry plane; node set 4 lies on the outside (free) surface. Let nodes 10, 14, 18 and 22 in the figure now denote the symmetry plane node at the blunt notch tip at each front position (the same as node 53 in Fig. 4.4). To compute J at the front position identified by node set 1, the crack front segment in the domain includes node sets 1 and 2. The input commands are (define the needed sets of nodes first followed by the *front* command)

```
node set 1   10  3 42 64 ...
node set 2   14 43 29 31 ...
node set 3   18 21 24 83 ...
node set 4   22 41 39 44 ...

front node sets 1 2 l3disop
```

The definition of all front node sets is included above for illustration even though only sets 1 and 2 are referenced in the *front node sets* command.

To compute J at the front position 14, the crack front segment in the domain includes node sets 1, 2 and 3. The input command is

```
front node sets 1 2 3 l3disop
```

Here, q will vary linearly (piecewise) along the front between node sets 1, 2 and 3 (q will be zero for nodes in set 1 and 3 and 1.0 for nodes in set 2).

4.4.8 Specification of q-Values

Two methods for defining the q-values are available: automatic and fully user-specified. Each method is described in a section below. The automatic method will suffice for must applications.

Automatic q Definition

The automatic method supports J computation for the following situations:

1. Pointwise evaluation at a crack front location on a symmetry plane or on a free surface (there are elements only to one side of the crack front location).
2. Pointwise evaluation at an interior crack front location corresponding to a corner node (elements exists on both sides of the crack front location).
3. Average J-value for the complete crack front (straight or slightly curved fronts).

At a crack front location, the automatic method constructs one or more domains for investigation of domain independence of computed J-values. The concept of a *ring* of elements is adopted to describe the domains generated at a crack front location (see Fig. 4.6). *Ring 1* contains those elements incident on the nodes defined in the list of front nodes or in the referenced *node sets*. Figures 4.4, 4.5 illustrate the *Ring 1* elements for initially blunt crack

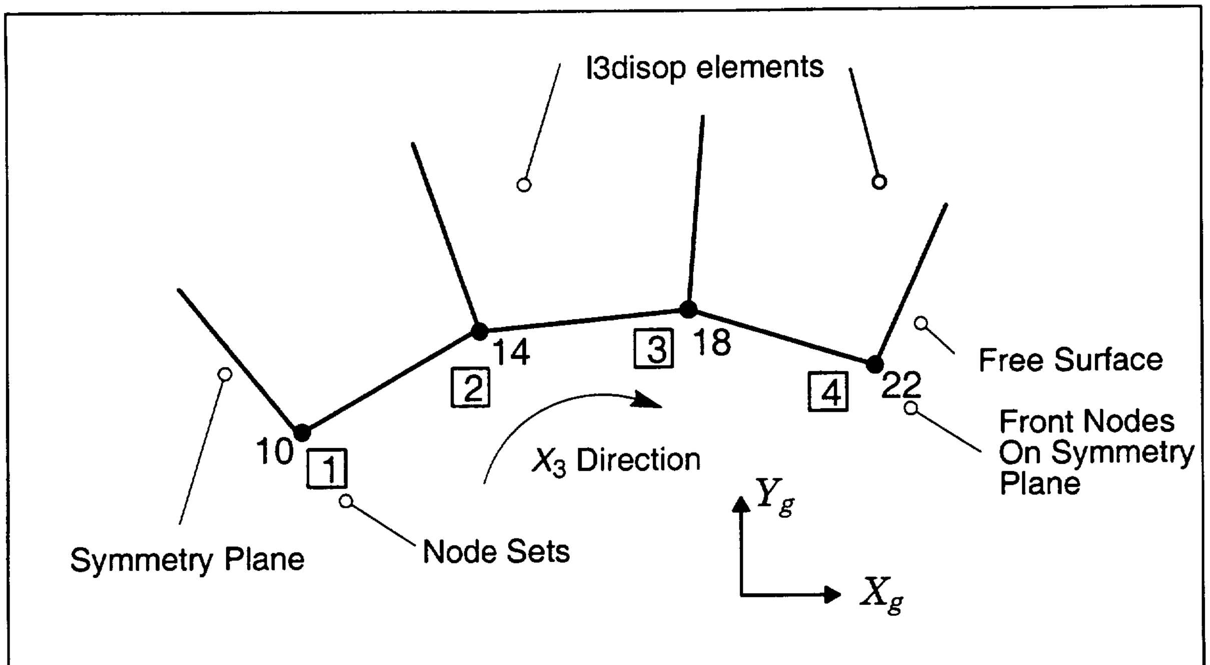


FIG. 4.5—*Example crack front to illustrate front nodes specification.*

tips and collapsed crack tips. Additional rings are constructed by examination of element connectivities. *Ring 2* contains the front elements plus the next ring of elements enclosing the tip. Again, Figs. 4.4, 4.5 illustrate the additional elements for the two types of crack tips. For the initially blunt-tip model, the user exercises full control over the elements included in the rings by selection of the "seed" nodes specified in the *node sets*.

J-values for the first few rings usually have the greatest error (especially for the blunt-tip models) and should be avoided if possible. J-values for *rings 4, 5*, ... should be reasonably similar. For a nonlinear elastic (deformation plasticity) model, the values in *rings 4, 5*, ... often show less than 1% variation.

The command to specify automatic generation of q-values has the form

q(-values) automatic (rings) < integerlist >

and must be followed by the command

$$\text{function (type)} \begin{Bmatrix} \text{a} \\ \text{b} \\ \text{c} \\ \text{d} \end{Bmatrix}$$

where a-d denotes the variation (function type) of q along the crack front. The four function types are illustrated in Fig. 4.7. Types a and c are used to evaluate J at end points of a crack front, e.g., at nodes 10 and 22 in Fig. 4.5. Type b is used to evaluate J at an interior node, e.g., nodes 14 and 18 in Fig. 4.5. Function type d is used to compute a "through-thickness" average J for a straight or slightly curved crack front.

When *q3disop* elements are used along the crack front, the automatic method supports J computation only at the element corner nodes. In this case, the automatic procedure to construct q–values sets the mid-side node value to the average value of the adjacent two corner node values (as illustrated in Fig. 4.7).

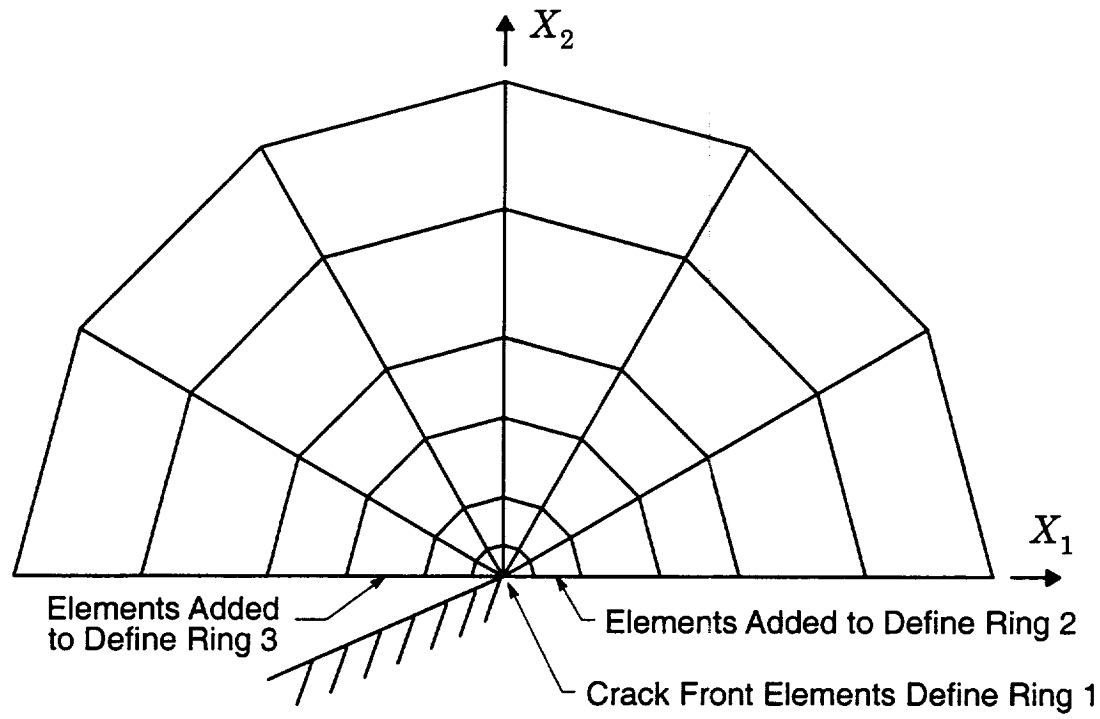

X_2
X_1
Elements Added to Define Ring 3
Elements Added to Define Ring 2
Crack Front Elements Define Ring 1

FIG. 4.6—*Concept of rings used in automatic domain generation.*

For function types a-c, the automatic algorithms construct nodal values for q which vary linearly in the X_3 direction. For function type d, q maintains a constant value in the X_3 direction along the front. Nodal values for q are generated automatically such that the following conditions hold:

Ring 1: q derivatives: $\partial q/\partial X_j$ = constant.

Ring i: for elements appearing in rings 1, 2, 3, ... i-1, the q derivatives: $\partial q/\partial X_1 = 0$, $\partial q/\partial X_2 = 0$ and $\partial q/\partial X_3 \neq 0$. For elements added to ring i-1 to define ring i, the q derivatives are $\partial q/\partial X_j$ = constant.

As a consequence of these q-derivative properties, element rings 1, 2, 3, ... i-1 have $DM_1 = DM_3 = 0$. These elements make a small contribution to DM_2 since the variation of u_3 with X_1 is non-singular. The acceleration forces which define DM_4 and the thermal strains which define DM_6 make significant contributions in near front rings since q, rather than q-derivatives, appear in the integral. For function type d, the terms DM_1, DM_2 and $DM_3 = 0$ for elements in rings 1, 2, 3, ... i-1.

The automatic generation process creates one additional domain for each ring requested by the user. The J-value for each of these domains is printed and included in the average, minimum, maximum statistics. If the element printing option is also *on*, the contribution for each element to each domain is printed. The list of rings specified in the automatic domain method can be of the form *rings* 2 4 6 10 15 .... While the domains for all rings (through the maximum ring listed) are created internally, J is computed and printed only for the ring numbers in the list. In this way, for example, the user may request computation and output for a few rings far from the crack front, e.g., rings 10-15. The domain processors

When crack face tractions are present, an additional contribution to the J-integral is computed using

$$\bar{J}_4 = -\int_{A_3+A_4} t_i \frac{\partial u_i}{\partial X_1} q \, dA_0 \; . \tag{4.18}$$

where t_i denotes the face traction expressed in the front system and $A_3 + A_4$ denotes the upper and lower portion of the loaded faces (refer to Fig. 4.2).

Eqs. (4.15) through (4.18) are implemented to support finite-strains and finite-rotations as indicated. The present formulation applies most realistically to models in which the displacement field leads to large (rigid) rotations on the domain but in which finite strains are confined to the usual blunting zone ahead of the crack tip. An example is a pin loaded, single-edge notch tension specimen, SE(T), containing a deep notch, i.e., $a/W > 0.5$. Under increased loading, the specimen may undergo relatively large rotations as the line of action of the axial load re-aligns with the center point of the remaining ligament. Finite strains are confined to the near tip region. The present formulation includes the effects of such large (rigid) rotations of the specimen on J-values.

4.3 Numerical Procedures

This section describes the numerical procedures implemented to evaluate the Domain Integrals described previously. An understanding of these procedures is necessary for the correct use of the commands described subsequently.

4.3.1 Definition of the q-Function

Consistent with the isoparametric formulation, the q-function within an element has the form

$$q(\xi,\eta,\zeta) = \sum_{i=1} N_i(\xi,\eta,\zeta) q_i \tag{4.19}$$

where q_i are the specified values of the q-function at the element nodes. The user defines: (1) a list of nodes along the crack front included in the computations to evaluate A_q, (2) elements over which integrations are to be performed, (3) q_i at nodes over the volume, V, and (4) orientation of the crack front coordinate axes at the point s under consideration.

When collapsed elements are defined along the crack front producing coincident nodes, only one of the coincident nodes at each location is specified; the computational routines locate the remaining coincident nodes and assign them the same value of q.

When the crack front has a small, initial radius, the user specifies a list of nodes at each crack front position to be treated as tip nodes. The computational routines then assign all listed nodes at each front position the same value of q. The specified lists of front nodes also play a key role in the generation of automatic domains for initially blunt crack fronts.

To define the orientation of the crack front axes relative to the global axes, users specify the components of a unit vector normal to the crack plane.

The specification of nodal q-values becomes exceedingly tedious for 3-D analyses. An "automatic" procedure is available as an option for generation of q-values. This procedure requires that the user specify: front nodes along the crack front, the number of domains required for checking path independence and components of the unit vector normal to the crack plane. The domain processors create domains of increasing distance from the crack tip using the mesh topology.

4.3.2 Volume Integrals

The volume integrals are numerically evaluated using the same Gaussian quadrature procedures adopted for element stiffness generation. The integral in Eq. (4.15) presents no difficulties as both W and the stresses are available at the Gauss point locations and the q-function derivative is readily computed from specified nodal values and Eq. (4.19). Gauss quadrature applied to Eq. (4.15) yields the expression for numerical computations as

$$\bar{J}_1 = -\sum_p \left[W\frac{\partial q}{\partial X_1} - P_{ji}\frac{\partial u_i}{\partial X_1}\frac{\partial q}{\partial X_j} \right]_p \det\left[\frac{\partial X_m}{\partial \eta_m}\right]_t w_p \tag{4.20}$$

where the summation extends over all Gauss quadrature points (p) and w_p denotes the Gauss weight values. The 1st PK stresses are computed from the unrotated Cauchy stressess using the two step transformation

$$\boldsymbol{\sigma} = \boldsymbol{R} \cdot \boldsymbol{t} \cdot \boldsymbol{R}^T \tag{4.21}$$

$$\boldsymbol{P} = |\boldsymbol{F}|\boldsymbol{\sigma}\cdot\boldsymbol{F}^{-T} \tag{4.22}$$

Cartesian derivatives of q and the displacements are obtained in the usual manner using the chain rule

$$\frac{\partial q}{\partial X_1} = \sum_{I}^{n}\sum_{m}^{3}\frac{\partial N_I}{\partial\eta_m}\frac{\partial\eta_m}{\partial X_1}q_I \tag{4.23}$$

and,

$$\frac{\partial u_i}{\partial X_1} = \sum_{I}^{n}\sum_{m}^{3}\frac{\partial N_I}{\partial\eta_m}\frac{\partial\eta_m}{\partial X_1}u_{iI} \tag{4.24}$$

where N is the number of element nodes. Similar procedures are followed for evaluation of the first two terms of Eq. (4.17); the third term in this equation is neglected.

To evaluate the $\bar{J}_2$ integral, standard isoparametric procedures readily support computation of the first term which involves $\partial\Theta/\partial X_1$, since temperatures are known at the nodes of elements. The second term requires evaluation of spatial derivatives of the thermal expansion coefficients, $\partial\alpha_{ij}/\partial X_1$. Thermal expansion coefficients are specified for materials and materials are associated with lists of finite elements in the WARP3D input. To compute the required derivatives, nodal values for α_{ij} are constructed by averaging values from elements incident on the nodes. Only elements with non-zero expansion coefficients are included in the averaging process. Standard isoparametric techniques then yield $\partial\alpha_{ij}/\partial X_1$ at Gauss integration points within elements. The $\partial\alpha_{ij}/\partial X_1$ term maintains domain independence of the J-values when the thermal expansion coefficient(s) are not constant within the integration domain.

4.3.3 Crack Face Traction Integral

The crack face traction integral, Eq. (4.18), is evaluated using nodal forces applied to crack face nodes. These include the "equivalent" nodal forces computed by the code from user-specified face pressures *and* any forces applied directly by the user to crack face nodes. The crack face integral is thus evaluated numerically using the expression

$$\int_{A_3+A_4} t_i\frac{\partial u_i}{\partial X_1}\,q\,dA_0 = \sum_{k}\sum_{l} q_l\left\{\partial u_i/\partial X_1\right\}_l^T\left\{P_i\right\}_l \tag{4.25}$$

where k is taken over elements with non-zero crack face tractions; l is taken over all element nodes on the loaded face; $\{P\}$ is the vector of *total* nodal forces at acting element node l (forces derived from applied face pressures *and* all user applied nodal forces on the face nodes). Displacement derivatives at the element nodes needed in Eq. (4.25) are obtained by extrapolating derivatives computed at Gauss point locations. Lagrangian polynomials are again adopted for the extrapolation. Not only is this technique more accurate than evaluating derivatives directly at the element nodes, the difficulty in computing derivatives at nodes on the crack front due to the singularity is avoided (extrapolated derivatives are not singular). Numerical tests demonstrate that the approximate expression given in Eq. (4.25) works very well.

The computational routines that evaluate Eq. (4.25) determine which element faces are loaded by examining the nodal forces for the complete element. If an element force vector indicates that more than one face is loaded, the lowest numbered element face is processed and a warning message is issued to the user. Because this procedure was adopted (thereby eliminating the need to respecify crack face loads during J computation), crack face loads and thermal loads should not be specified in the same loading condition—the computational routines will mistake the equivalent nodal forces due to the thermal loading for crack face loading.

If all nodes of an element have non-zero applied forces, a body force load is assumed to exist and no domain integral contributions are computed.

For user specified domains (not the *automatic* domains), the list of elements to process must include all elements with crack face loading if any node on the face has a non-zero q value. The automatic domain procedure performs this task.

When the effects of crack face loading are specified by the user through directly applied nodal forces (rather than using the built-in face pressure loading), nodal forces must be specified on all nodes of an element face, including those that have constraints in the direction of the face loading. If omitted, the logic to determine which element face is loaded does not function properly.

4.3.4 Crack Front Nodes

Fronts Defined by Collapsed Elements

The use of degenerated brick-type elements generally leads to meshes with multiple, coincident nodes along the crack front. To simplify specification of the q-function over the domain volume, the q-value for only *one* of the coincident nodes at such crack front positions is required. The remaining coincident nodes at corresponding crack front positions are located and assigned the same value for q. The procedure followed to locate coincident nodes is outlined below.

For each user specified node along the crack front, the numerical procedure constructs coordinates for a cubical prism centered at the node, then locates all other nodes of the model that lie within the prism. Such nodes are treated as coincident with the specified node and are assigned the same q-value. Dimensions for the cubical prism are defined as follows: for 2 or more nodes specified along the crack front (3-D models), the prism extends $\pm\ R \times tol$ about the node, where R is the distance between the first two listed nodes on the crack front.

The value 0.001 is currently specified for *tol*. While this value has proven adequate for most crack front meshes, models with exceptionally large element lengths along the front may require a smaller value for *tol* (at present this requires a change in the source code).

Fronts Defined With Initial Root Radius

For analyses that require a formulation including finite-strain effects, crack fronts are generally modeled with a small, initial radius as illustrated in Fig. 4.4. For these models, the automatic procedure described above to define a set of "tip" nodes becomes inadequate. Users are required to specify the appropriate list of "tip" nodes at each front position, e.g., the nodes indicated by open symbols in the figure. Each node is then assigned the same q-value by the computational routines.

Only the node actually on the symmetry plane (53 in the figure) is required unless the automatic domain option is invoked. The list of user-specifed tip nodes provides the "seed" to start the automatic procedure for domain generation, i.e., elements in the first domain are those incident on the listed "tip" nodes.

4.3.5 Computation of A_q

The area under the q-function along the crack front, denoted A_q, is required to normalize J for arbitrary magnitudes of the specified q-function in Eq. (4.8), see also Fig.4.3. Thus,

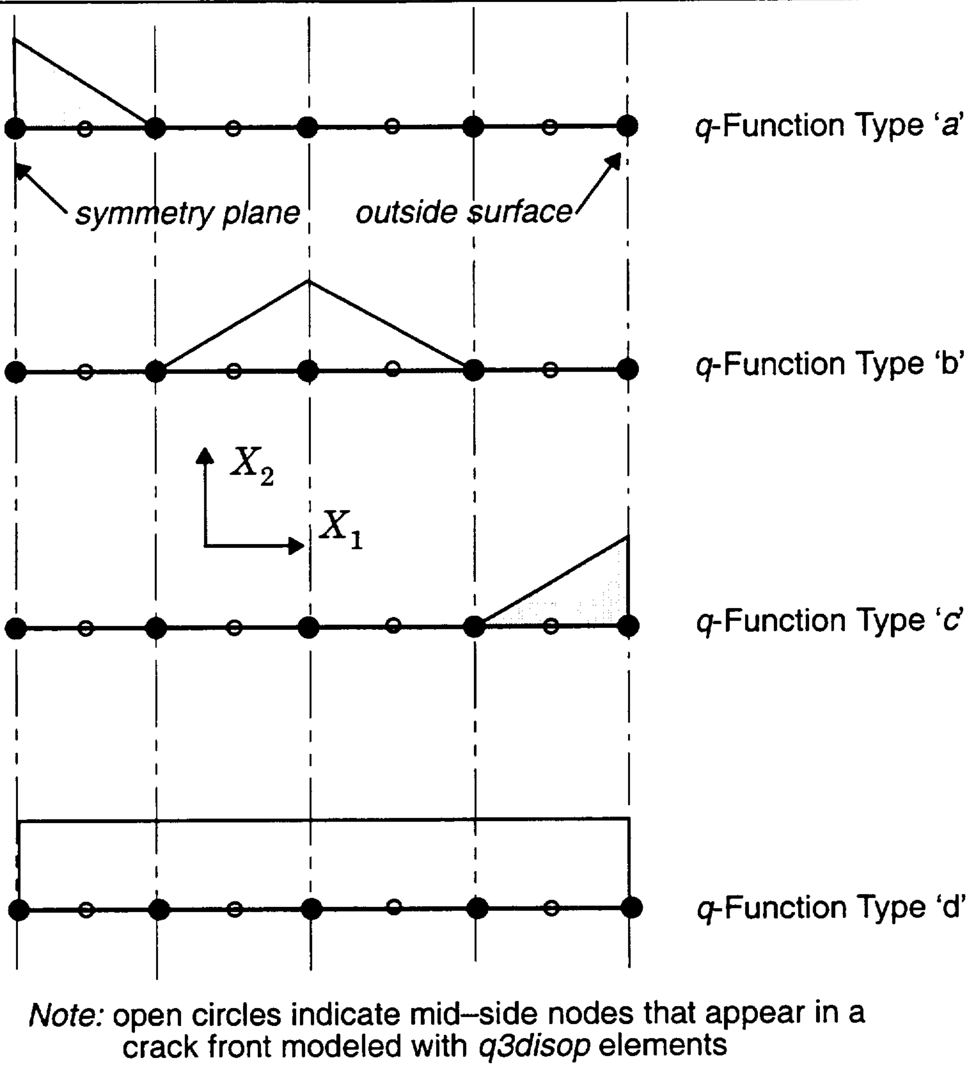


FIG. 4.7—*Types of q-functions available for automatic domain generation.*

include the contributions of all elements in rings nearer the tip as required for each term of J, e.g., crack face loading and inertia terms which involve q and not q-derivatives.

Consider the following example of automatic domain generation (refer to Fig. 4.5). Let the crack plane be normal to the global Z-axis.

```
domain symm_corner
  normal plane nz 1.0
  front nodes 10 14 linear verify
  q-values automatic rings 2-4
  function type a
  .
  .
  .
compute domain integral
```

Function type a is specified since node 10 is on the symmetry plane. Automatic domains are constructed for rings 1-4 but J is computed and printed only for rings 2-4 to omit ring 1 which usually has the most error.

To compute J at the front location of node 14 and 18, the following automatic domains and compute commands are used

```
domain front_14
  normal plane nz 1.0
  front nodes 10 14 18 l3disop verify
  q-values automatic rings 2-4
  function type b
  .
  .
  .
compute domain integral

domain front_18
  normal plane nz 1.0
  front nodes 14 18 22 l3disop verify
  q-values automatic rings 2-4
  function type b
  .
  .
  .
compute domain integral
```

At the intersection of the crack front with the outside free surface (at node 22), the following domain is specified

```
domain outside_22
  normal plane nz 1.0
  front nodes 18 22 l3disop verify
  q-values automatic rings 2-4
  function type c
  .
  .
  .
compute domain integral
```

For a crack with the front curvature indicated in Fig. 4.5, a thickness-average J using function type d would seem to be of questionable value.

User Specified q-Values and Elements

All nodal values of q are zero by default. Non-zero nodal values of q over the domain are defined with the command

q(-values) < node list > < q: real >

where the nodal q-values must be of class <real> to be distinguished from the list of node numbers. This command may be repeated as needed to define all nodal values for q in the domain. q-values must be specified for all element corner nodes in the domain and for all nodes along the crack front segment under consideration. Computational routines for quadratic elements employ a linear variation of q between corner nodes (they override the specified mid–side node values including those along the crack front).

The list of all elements to be included in the computations is defined with the command

elements <integerlist>

Elements that should be included are: (1) those over which q is not constant, (2) those with loaded crack faces and non-zero q-values, (3) those with inertia forces and non-zero q-values, (3) those with thermal loading and non-zero q-values.

The following example illustrates the definition of a domain to compute J at node 14 for the crack front illustrated in Fig. 4.5.

```
domain outside
  normal plane nz 1.0
  front nodes 10 14 18 l3disop verify
  q-values 10 18 0.0
  q-values 14 1.0
  elements 10-14
  .
  .
compute domain integral
```

In this example, only the crack front elements incident on node 18 make contributions to J (this is *not* recommended!). q-values at nodes 10 and 18 default to 0.0 and can be omitted from the above commands (they are included for readability). The normal plane and front node specifications are identical to automatic domains. Only elements appearing in the specified list are evaluated during J computations.

4.4.9 Printing Options

By default, the total contributions (DM_1, DM_2 ...) and the sum of DM_1, DM_2 ... are printed for the domain (each ring of an automatic domain). The domain values are followed by the minimum J, maximum J and average J for the domains. When inertia effects are present,$DM_3, DM_4 \neq 0$, separate totals for static and dynamic terms are provided to make obvious the relative importance of these terms in the total J-value.

To explicitly request this level of output, use the command

print total<u>s</u>

More detailed output listing the contribution from each element is requested with the command

print elem<u>ent</u> (valu<u>es</u>)

This option also provides the information of the *print totals* default.

4.4.10 Integration Order

The volume integrals contributing to J are evaluated using the same order of Gauss integration as is used for stiffness computation. For *l3disop* elements, J-values with a greater level of domain independence are often obtained by using one-point Gauss integration. This option is requested with the command

use 1 (point rule)

4.4.11 Face Loading

By default, crack face loadings if present are included in the domain integral computations. The crack face loadings may be omitted with the command ignore (crack) (face) loading

As noted previously, this option should be invoked when crack growth is modeled by releasing the closing forces to zero over a number of load steps. Such forces are mistakenly interpreted as crack face tractions by the domain integral processors.

4.4.12 Domain Verification

The definition of a domain as stored in internal tables may be printed with the command *dump*. This command may be given at any time during the domain definition and as many times as desired.

4.4.13 Debugging Domain Computations

The actual domain computations may be traced with printed output detailing each step of the computations. This may prove convenient to more closely examine J-values. To trace the primary domain integral processor (but not element integration routines), use the command

debug driver

To debug element integration routines, use the command

debug elements

Both commands may be specified in the domain definition.

4.4.14 Complete Examples

The following are two complete examples illustrating all commands for domain definition using automatic procedures.

```
domain symm_corner
   symmetry
   normal plane nz 1.0
   front nodes 10 14 l3disop verify
   q-values automatic rings 2-10
   function type a
   print totals
   print element values
   use 1 point rule
   ignore crack face loading
   debug driver
   debug elements
   dump
compute domain integral

domain free_edge
   symmetry
   normal plane nz 1.0
   node set 1  32 54 90 31 63
   node set 2  87 43 21 76 34
   front node sets 1 2 l3disop
   q-values automatic rings 2-10
   function type c
   print totals
```

```
      print element values
      use 1 point rule
      ignore crack face loading
      debug driver
      debug elements
      dump
   compute domain integral
```

Chapter 5

Crack Growth Procedures

5.1 Introduction

Two procedures are provided to include the effects of discrete crack extension in WARP3D. In the first type of crack growth, termed *element_extinction*, complete elements in the model are deleted when a critical condition (damage) is reached under increased loading. The element stiffness is set to zero and the forces exerted by the element on adjacent nodes are relaxed to zero over a user–specified number of load steps or using a traction-separation model. In this procedure the element is not topologically deleted from the model but it no longer contributes any resistance to loading. In other codes, this technique of element extinction is often referred to as an element "death" option.

In the second type of crack growth, termed *node_release*, an increment of crack extension on a symmetry plane is achieved by the traditional node release procedure. When conditions for growth are achieved, the displacement constraint holding the crack closed at that point on the front is replaced by the corresponding reaction force, which is then relaxed to zero. The force release process occurs over a user-specified number of steps or using a traction-separation model. The element remains in the model and most often undergoes inelastic unloading and then re-yielding as the crack tip continues to extend. The node release procedures support growth along multiple crack fronts on the symmetry plane and readily model non-uniform growth along the front, e.g. tunneling. Currently, the conditions for growth are specified by a critical crack-tip opening angle (CTOA). The growth processor examines each possible CTOA value using element edges incident on an active front node and grows the crack when any of those angles exceeds the specified critical value. The user can also request crack extensions during the analysis irrespective of the crack growth criterion.

At the present time, only the *l3disop* elements are supported for crack growth using the node release procedures. Other element types may be used to construct the finite element model but only *l3disop* elements may be involved in the crack growth processing.

All available element types can be processed in crack growth analyses using the element extinction procedures.

This chapter describes commands to invoke each of the two crack growth procedures and additional details of their implementation in WARP3D.

5.2 Crack Growth by Element Extinction

Elements are effectively deleted from the solution when a user-specified level of damage develops under increased loading. During subsequent load steps, the element stiffness is taken as zero and the nodal forces exerted by the element on adjacent nodes are relaxed to zero over: 1) a user-specified number of load steps or 2) a simple linear, traction-separation model. The presently available measures of damage include: (1) attainment of a critical void fraction, f, in elements which have the Gurson-Tvergaard dilatant plasticity material model (type *gurson*), and (2) attainment of a critical plastic strain defined by the stress-modified critical model in elements which have the mises (J_2) plasticity material model (type *mises*).

The user actions required to invoke the element extinction option during an analysis are:

- specify the logical property *killable* in the definition of a material that invokes the *gurson* or *mises* material model (in the same analysis, there can be other materials using the *gurson* and *mises* model that do not have the *killable* property).
- following the procedures for other nonlinear analyses, define the finite element model, loading, constraints and nonlinear solution parameters.
- use the commands described subsequently in this section to define parameters controlling the crack growth procedures (critical porosity, critical plastic strain, number of release steps, printing options, etc.). These parameters are specified in a manner analogous to specification of the nonlinear solution parameters; some crack growth parameters may be altered during the analysis as noted in the command descriptions that follow.
- use various combinations of *compute* and *output* commands to control the nonlinear solution over load steps. The crack growth procedures are automatically invoked by solution management routines in WARP3D.
- the analysis restart features of WARP3D fully support crack growth modeling. Restart files contain the values of growth parameters and the solution state required to continue an analysis with crack growth.

5.2.1 Damage Criteria

Gurson-Tvergaard Model (GT)

For this damage model, element extinction takes place when the current void fraction, f, reaches a user-specified critical value, f_c. The present implementation of the GT model limits the "failure" condition to a triaxiality independent, critical void fraction. Typical values of f_c for structural and pressure vessel steels are 0.1-0.25, compared to initial void fractions, f_0, of 0.0005-0.005. The current value of f for this comparison is obtained from the simple average of the element Gauss point values.

Stress-Modified Critical Strain (SMCS)

For this damage model, element extinction takes place when the equivalent plastic strain, $\bar{\epsilon}_p$, reaches a critical value computed with the SMCS criterion, i.e.,

$$\bar{\epsilon}_c^p = \alpha \exp\left(-\beta\frac{\sigma_m}{\sigma_e}\right) \tag{5.1}$$

where α and β are user-specified, material dependent constants; σ_m denotes the mean stress and σ_e denotes the Mises equivalent stress. Most often β is taken equal to 1.5 in accord with the continuum hole growth model of Rice and Tracey [76]. Mackenzie, et al. [56] and Hancock and Cowling [32] proposed the above form of the SMCS model based on experimental studies of notched tensile specimens (steel). Panontin and Shepard [71] describe a complete study of the calibration process to estimate α and β from notched-tensile data for an A516 pressure vessel steel and an HY 80 steel. Their work focuses on applying the SMCS model to estimate geometry effects on J_{Ic}. For their A516 material, they found $\alpha = 1.996$ and $\beta = 1.5$, and for HY 80 they found $\alpha = 3.865$ and $\beta = 1.5$.

The values of σ_m, σ_e, and $\bar{\epsilon}_p$ used to evaluate Eq. (5.1) are obtained from the simple average of the element Gauss point values.

5.2.2 General Input Commands

The sequence of commands to initiate the definition of crack growth parameters is

$$\text{\underline{crack} (\underline{grow}th) (\underline{para}meters)}$$

$$\text{\underline{type} (\underline{of}) (\underline{cra}ck) (\underline{grow}th)} \left\{ \begin{array}{c} \text{\underline{none}} \\ \text{(\underline{elem}ent\_extinction) \underline{gurson}} \\ \text{(\underline{elem}ent\_extinction) \underline{smcs}} \end{array} \right\}$$

where *none* turns off subsequent element extinction during the analysis. Once elements have been made extinct in an analysis and the option *none* is given, further crack growth cannot then be re-invoked. To temporarily suppress further growth, the simplest (and recommended) procedure is to modify critical values of the damage criteria.

The keyword *gurson* invokes element extinction based on attainment of a critical porosity, f, in ***killable*** elements associated with the GT material model. The keyword *smcs* invokes element extinction based on attainment of a critical plastic strain, $\bar{\epsilon}_p$, in ***killable*** elements associated with the *mises* material model. Note: only *one* type of damage criterion may be specified in an analysis and it cannot be changed to another criterion during the analysis.

When the element damage first exceeds the specified limit, the element "internal" forces are imposed on adjacent nodes in the model as nodal forces. The values of these forces decrease linearly to zero over 1) a number of sequential load steps or 2) a linear traction-separation model. The element stiffness is immediately set to zero and remains zero for all subsequent load steps. Input commands to describe the force release models are described in a subsection below.

The element extinction procedures provide a convenient *printing* option to simplify interpretation of the growth process. The command has the form

$$\text{\underline{print} (\underline{sta}tus)} \left\{ \begin{array}{c} \text{\underline{off}} \\ \text{\underline{on}} \end{array} \right\} \text{(\underline{order} (\underline{elem}ents)< element list: integerlist >)}$$

where the keyword *on* or *off* is required. An optional list of elements previously marked ***killable*** may be specified for processing. If no list is given, all elements having a *gurson* or *mises* material model with the ***killable*** property are included in the list (in ascending numerical order). When the optional list is given, information is printed for elements in the order specified in the list.

At the beginning of each load step when this printing option is *on*, a tabular summary of the current status is printed for each element in the list. For the *gurson* damage criterion, the following values are provided: initial porosity (f_0), current (average) porosity (f), average plastic strain in the matrix material ($\bar{\epsilon}^p$) and average (Mises) equivalent stress in the matrix material ($\bar{\sigma}$). For the *smcs* damage criterion, the following values are provided: the average plastic strain in the element ($\bar{\epsilon}^p$), the current value of critical plastic strain defined by the *smcs* criterion ($\bar{\epsilon}^p_c$), average mean stress in the element (σ_m) and the average (Mises) equivalent stress in the element (σ_e). Additionally, if automatic load reduction is enabled (see section 5.2.4), the table includes the increase in the current growth parameter over the last step; if the *gurson* damage criterion is used, the increase in average porosity is printed, while the increase in average plastic strain in the element is output if the *smcs* criterion is specified.

To prevent excessive amounts of output, information is printed only for those elements with $f > f_0$ (*gurson* model) or $\bar{\epsilon}_p > 0$ (*mises* model).

By default, every element eligible to be made extinct is processed without regard to any specific topological order. In some cases, it may be desirable to force extinction of elements in prescribed topological order. To specify this feature, use the command

sequential (extinction) $\left\{\begin{array}{c}\text{off}\\ \text{on}\end{array}\right\}$ (order < element list: integerlist >)

where the use of this feature is invoked/suppressed with the required *on/off* keyword. The optional list provides the topological sequencing of elements to be made extinct. For example, if the second element in the list reaches the critical damage parameter prior to the first element in the list, then both the first and second elements in the list are made extinct simultaneously. When the list is omitted, the topological ordering is taken to be ascending numerical sequence by element number for all elements in the model with the *killable* material property.

5.2.3 Damage Criteria Commands

For the *gurson* damage criterion, the porosity value at which element extinction occurs, f_c, is specified by the command

critical (porosity) < porosity limit: value >

The average porosity at the Gauss points for each *killable* element with a *gurson* material model is compared with the specified critical value at the beginning of each load step. When the average value first exceeds the porosity limit, the element extinction process begins for that element. The default value for critical porosity is 0.20.

For the *smcs* damage criterion, the user specifies values for the material dependent parameters, α and β, with commands of the form

alpha < value >

beta < value >

The current plastic strain is compared to the computed critical plastic strain for each *killable* element with a *mises* material model at the beginning of each load step. The comparison is made using single point values obtained from the average of Gauss point values. When the average value first exceeds the critical strain, the element extinction process begins for that element. Default values are: $\alpha = 1.0$ and $\beta = 1.5$.

5.2.4 Automatic Load Reduction

When load steps are too large, element extinction may occur too rapidly, which allows the force release process to affect adversely the stress-strain history of material ahead of the crack front. The history effects may cause difficulties in Newton convergence of the global solution, or may cause J-Δa curves to be too high, too low, or to oscillate. To alleviate this problem, WARP3D provides a feature to reduce automatically the load step size based on the change within a load step of the appropriate crack growth parameter. The reduction algorithm operates as follows: consider a killable element on the crack plane. If the parameter used as the growth criterion increases more than a user-specified amount in a single load step, the growth processor sets a permanent 50% reduction on all future load step sizes. This 50% reduction repeats as needed in subsequent steps until the maximum increment of growth parameter is not exceeded within a load step.

For the *gurson* damage criterion, the reduction mechanism triggers when the porosity growth during a step, Δf, is larger than a specified percentage of the critical porosity ($\Delta f > af_c$). For the *smcs* model, a user-specified increment of plastic strain within a step provides the criterion for load reduction ($\Delta\bar{\varepsilon}^p > \Delta\bar{\varepsilon}^p_{crit}$). Use of a load reduction parameter which is too stringent forces the load steps to become unnecessarily small, while the use of a lenient reduction parameter may not completely eliminate history effects; trial and error is required to determine proper values. To assist in choosing reduction values, WARP3D prints the change in the growth parameter after each step for all killable elements if the print status flag is set to *on*. Note that appropriate reduction values may be dissimilar between analyses with different geometries, loadings, or material characteristics.

The command to request automatic load reduction for the *gurson* criterion is:

$$\underline{\text{auto}}\text{matic } (\underline{\text{load}})\ (\underline{\text{reduc}}\text{tion}) \begin{Bmatrix} \underline{\text{off}} \\ \underline{\text{on}} \end{Bmatrix} \text{ [} (\underline{\text{max}}\text{imum})\ \underline{\text{poro}}\text{sity } (\underline{\text{change}})\ \text{<real>]}$$

where *<real>* denotes the maximum allowable change in porosity between load steps expressed as a percentage of the critical porosity. By default, the maximum porosity change is 10% (0.1 as the input value above) and the load reduction algorithm is off; typical values are 0.05 – 0.1.

The command to request automatic load reduction for the *smcs* criterion is:

$$\underline{\text{auto}}\text{matic } (\underline{\text{load}})\ (\underline{\text{reduc}}\text{tion}) \begin{Bmatrix} \underline{\text{off}} \\ \underline{\text{on}} \end{Bmatrix} \text{ [} (\underline{\text{max}}\text{imum})\ (\underline{\text{plas}}\text{tic})\ \underline{\text{st}}\text{rain } (\underline{\text{change}})\ \text{<real>]}$$

where *<real>* denotes the maximum allowable increase of the average plastic strain in the element ($\bar{\varepsilon}^p$) within a load step. *Note*: this is a specific strain increment, not a percentage as for the *gurson* criterion. By default, the maximum increase in plastic strain is 0.01 and the load reduction algorithm is off.

5.2.5 Extinction Algorithm

At the beginning of each load step n (n>1), and each adaptive sub-step, the average value of the damage parameter is computed for each *killable* element in the model. When the element conditions are such to require extinction (achieved the critical value of the damage parameter or the sequential ordering feature dictates extinction even when the critical damage value is not yet attained), the following actions are taken:

- Young's modulus and Poisson's ratio for the element are set to zero. The element history data is deleted (porosity, plastic strain, stresses, etc.).

- Element contributions to the global internal force vector are applied as nodal forces. All subsequent contributions of the element to global equilibrium are zero. The element internal force vector when extinction begins is gradually decreased in a linear fashion over subsequent load steps. Because the element forces are converted into nodal forces and treated thereafter as ordinary (user-specified) forces, the adaptive step algorithm is unaffected by crack growth and often proves essential for obtaining converged solutions following a growth increment.
- All subsequent computations for the element stiffness (linear or tangent) resolve to a zero matrix.
- When all elements connected to a node are made extinct, the node has no stiffness and introduces a singularity into subsequent equation solving efforts. To prevent this, the element extinction procedures track the number of elements attached to model nodes at any time and automatically supply new constraints on "free" nodes to eliminate the singularity.
- The blocking requirements dictate that all elements in a block must be *killable*. When a new element is made extinct in a block, checks are made to determine if all elements in the block have been made extinct; computations on such blocks may be completely skipped in subsequent load step solutions.
- The crack growth processor modifies the nonlinear solution parameters as follows: (a) a linear stiffness matrix is requested for the first iteration of the upcoming load step, (b) the displacement extrapolation flag is turned off permanently. Such use of the *linear* stiffness matrix for the structure properly accounts for elastic unloading that generally occurs immediately following a node release. These modifications in the nonlinear solution procedure are found necessary to maintain high rates of convergence.

5.2.6 Release Models for Element Forces

The simplest procedure to relax internal nodal forces for a newly extinct element employs a fixed number of load steps. The commands to specify this option are

force (release) (type) steps

release (steps) < integer >

This force release model is the default option and can be used to render extinct any element in the model, whether or not it lies on the crack plane. The default value is 5 steps. The number of release steps *cannot* be altered once any elements have been made extinct. A complete example of crack growth input using this force release procedure is:

```
crack growth parameters
  type of growth element_extinction gurson
  force release type steps
  release steps 10
  critical porosity 0.25
  print status on order 20-80 by 2
  sequential extinction on order 20-80 by 2
```

While the above procedure is simple, computed solutions often exhibit an undue dependence on the number of load steps employed in the analysis. When large load steps are defined, for example, the above procedure (with release steps > 1) artificially restrains opening of the crack faces. In analyses with very small load steps, the element internal forces may be reduced to zero too quickly.

To place the force release process on a more physical basis, a linear traction-separation model is provided. Figure 5.1 illustrates this model using a 2-D schematic. In Fig. 5.1 (a), let D denote the undeformed height normal to the crack plane of a typical "cell" element (such elements are often square or nearly so). When this element reaches the critical poros-

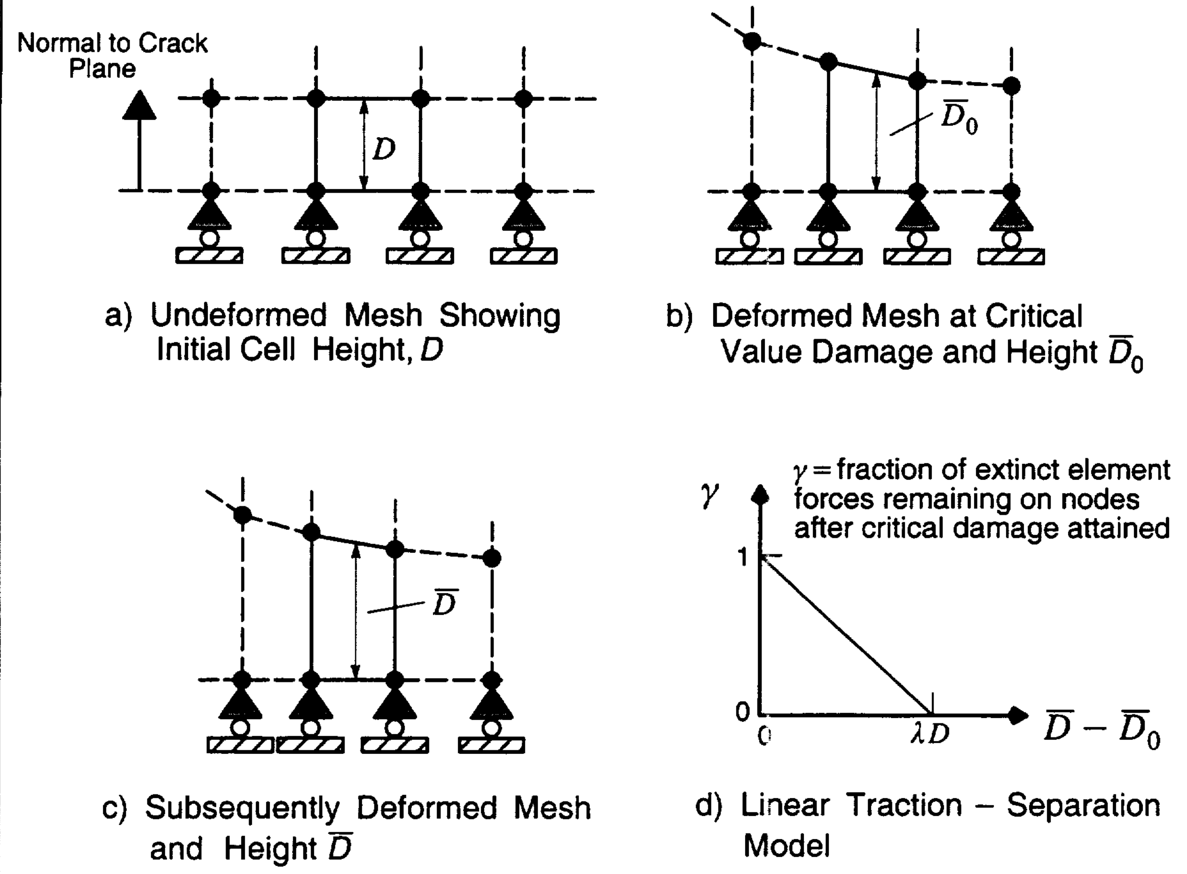

Normal to Crack Plane
D
a) Undeformed Mesh Showing Initial Cell Height, D
$\bar{D}_0$
b) Deformed Mesh at Critical Value Damage and Height $\bar{D}_0$
$\bar{D}$
c) Subsequently Deformed Mesh and Height $\bar{D}$
γ
γ = fraction of extinct element forces remaining on nodes after critical damage attained
1
0
0
λD
$\bar{D} - \bar{D}_0$
d) Linear Traction – Separation Model

FIG. 5.1—*Traction-Separation Model for Release of Extinct Element Forces*

ity, the average deformed height normal to the crack plane is denoted $\bar{D}_0$, as indicated in Fig. 5.1 (b). This value is computed using the *average* displacement normal to the crack plane of the four nodes on the "top" face of the element. During subsequent load steps, the newly extinct element continues to elongate normal to the crack plane, with the average deformed height denoted $\bar{D}$, as shown in Fig. 5.1 (c). The internal forces present in the element at extinction are reduced to zero in a linear fashion with subsequent increases in $\bar{D} > \bar{D}_0$. At any load step after attaining the critical damage state, the remaining fraction of internal forces applied to nodes of the extinct element, γ, is given by

$$\gamma = 1.0 - \frac{\bar{D}-\bar{D}_0}{\lambda D} \quad (0 \le \gamma \le 1) \tag{5.2}$$

where a typical value for the release factor, λ, is 0.1.

Input commands to specify the traction-separation model are thus:

force (release) (type) traction(-separation)

release (fraction) < λ factor:number>

crack (plane) normal $\left\{ \begin{matrix} x \\ y \\ z \end{matrix} \right\}$ coordinate <number>

cell height < D dimension:number>

To support general element meshing, the normal direction to the crack plane may be any one of the coordinate axes and the position of the crack plane may be non-zero. *Default* values are *not* supplied for the crack plane normal direction or cell height; users must explicitly define values for these parameters. The default value of λ is 0.1. A complete example of crack growth input using the traction-separation model is:

```
crack growth parameters
  type of growth element_extinction smcs
  release type traction-separation
  cell height 0.004
  crack plane normal y coordinate 0.0
  release fraction 0.2
  alpha 0.95
  beta 1.5
  print status on order 20-80 by 2
  sequential extinction on order 20-80 by 2
```

5.2.7 Meshing Restrictions

The automatic procedures implemented to render elements extinct impose restrictions on the element mesh layout along the initial crack front and along the roots of side-grooves. Figure 5.2 illustrates the recommended mesh design in a plane normal to a general 3-D crack front. Crack extension occurs in the X_c direction on this figure; Z_c is tangent to the crack front. It is essential that no other elements connect to the "front nodes" except the those along the front indicated by the letter **A**. A traditional "focused" mesh at the initial crack tip can be used only if a small keyhole remains so that the topological requirement shown in the figure remains satisfied. If needed for convenience in mesh construction, elements on the initial crack face ($Y_c = 0$) below **B** shown on the figure can be included in the model. Such an arrangement defines a one-element, square keyhole at the initial tip.

A similar topological requirement must be satisfied at the roots of side grooves. The figure shows a "square" shape for the root of the side-groove with the "radius" of the root given by the Y_c dimension of elements **A**. No other elements except those indicated by **A** can be connected to the indicated nodes along the side-groove root.

Both the crack front and side-groove root restrictions on mesh topology arise from the connectivity counting used in the extinction algorithm. Consider the crack front elements **A** shown in the top figure. The crack growth processor maintains a count of the number of elements connected to each node of the model. As the **A** elements are made extinct, the number of elements connected to the nodes are decremented. When the "count" for a node reaches zero, the crack growth processor inserts new displacement constraints into the database which fully constrain subsequent movement of the node; when the count reaches zero, there are no elements with remaining stiffness attached to the node and a singular solution would otherwise result. If regular elements (no damage allowed) are defined to the left of **A** in the top figure, the count for the front nodes never reaches zero, and those elements suppress opening of the tip.

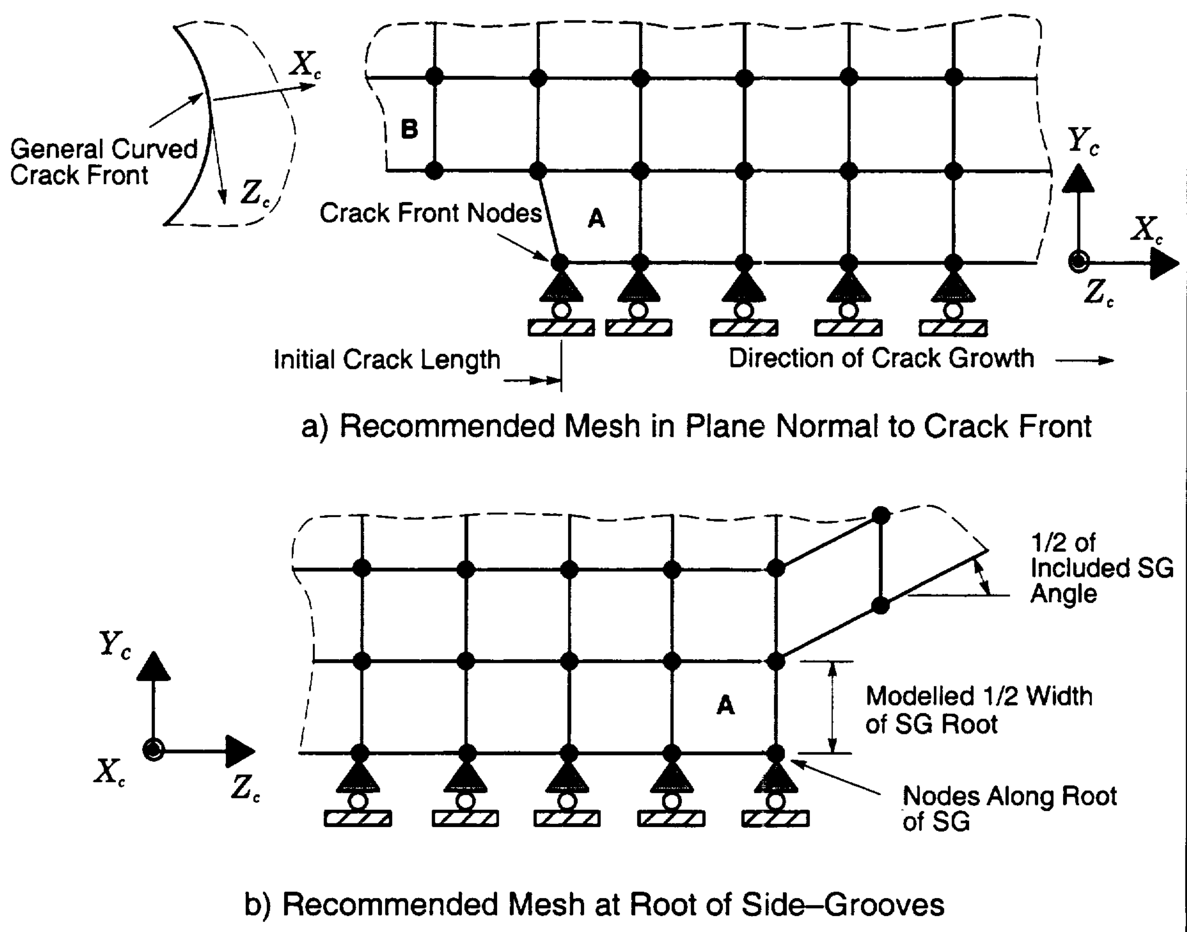

X_c
General Curved Crack Front
Z_c
B
Crack Front Nodes
A
Y_c
X_c
Z_c
Initial Crack Length
Direction of Crack Growth
a) Recommended Mesh in Plane Normal to Crack Front
1/2 of Included SG Angle
Y_c
Modelled 1/2 Width of SG Root
A
X_c
Z_c
Nodes Along Root of SG
b) Recommended Mesh at Root of Side–Grooves

FIG. 5.2—*Topological Restrictions on Meshes for Element Extinction*

5.3 Crack Growth by Node Release

In crack growth by node release, WARP3D releases constraints applied to nodes on the crack front after a user-specified level of deformation, thereby creating new, traction-free crack surfaces. The presently available measure of deformation is the opening angle of the crack at each crack front node (CTOA). During subsequent, user-specified load steps, the forces previously exerted by the constraints normal to the crack plane (reactions) relax to zero over a user-specified number of load steps, or through simple, linear traction-separation law.

Strategies for node release generally follow one of two approaches: (1) the user-specified external loads and displacement boundary conditions remain fixed while additional load steps relax the reaction forces to zero on newly unconstrained nodes, or (2) the external loads and displacement boundary conditions continue to change according to the user-specified values during subsequent load steps concurrent with the relaxation of reaction forces on newly unconstrained crack front nodes. WARP3D adopts the second strategy for two reasons: (1) it more closely models the physical process of crack growth thereby minimizing the effects of artificial elastic unloading of the material, and (2) the computation time is dramatically reduced by eliminating the additional "release" steps. Numerical studies reveal negligible differences in solutions with the two strategies.

The node release procedures permit very general, non-uniform (Mode I) growth along initially straight or curved crack fronts located in plates, pipes, pressure vessels, etc. Multiple, initial crack fronts may exist and are detected automatically by the crack growth processors. Crack fronts may grow to coalesce thereby merging two or more smaller cracks into a larger crack.

The crack growth processors also provide the option to enforce uniform growth along a crack front. The user specifies a "master" node on each initial front; the CTOA at the initial and subsequent master nodes governs when that entire front advances. Again, multiple initial crack fronts may exist and they may grow to coalesce thereby merging smaller cracks into fewer, larger cracks. This capability enables strong 3-D effects to influence the evolution of CTOA with loading while maintaining a simpler model for crack extension, e.g., to model crack extension in very thin materials. In this enforced uniform growth approach, the CTOA is computed at a user-specified, fixed distance behind each current crack front. The distance may or may not coincide with a node location. The crack front advances in increments of this same distance, which may represent multiple elements ahead of the current front location. Numerical experiments demonstrate this approach produces very well behaved solutions and affords the option to investigate convergence of crack growth solutions with mesh refinement. With the introduction of a fixed distance behind the front at which to compute the CTOA, the mechanics of growth become divorced from the specifics of the element sizes. The user can specify a different distance for the initiation of growth and for continued growth (this enables initial crack growth to occur at a specified CTOD).

Finally, with both the general 3-D and enforced uniform growth capability, the user can explicitly force crack growth at any time by artificially adjusting the critical CTOA value between load steps.

Crack growth by node release requires these user actions:

- following the procedures for other nonlinear analyses, define the finite element model, loading, constraints and nonlinear solution parameters. The model must follow some specific geometry guidelines, which are given below.
- use the commands described subsequently in this section to define parameters controlling the crack growth procedures (critical angle for release, number of release steps, etc.). The

specification of these parameters is analogous to the nonlinear solution parameters; some crack growth parameters may be altered during the analysis as noted in the command descriptions that follow.

- use combinations of *compute* and *output* commands to control the nonlinear solution over load steps. The solution management routines in WARP3D automatically invoke crack growth procedures.
- the analysis restart features of WARP3D fully support crack growth modeling. Restart files contain the values of growth parameters and the solution state required to continue an analysis with crack growth.

To assist the user in performing fracture analyses, additional features are provided by the crack growth processors. These address the proper specification of load increments and sizes during crack growth. Because the applied load is discretized into finite size increments (load steps), it may be difficult to estimate *a priori* load step sizes so as not to "overshoot" the critical CTOA value. Similarly, the crack growth process introduces the possibility of strong history effects on stress-strain fields ahead of the crack front. If crack growth occurs over too few load steps, the solution may show an effect due to an insufficient number of load steps. The automatic procedures implemented in the WARP3D crack growth processors resolve these two problems.

5.3.1 Geometry Requirements

The procedures for automatic node release impose several restrictions on the geometry of the model. The crack exists initially and propagates within a plane which must be normal to one of the coordinate axes. Moreover, the crack plane is also a symmetry plane. Consider all the nodes on the crack plane. Nodes initially unconstrained in the normal direction define the initial crack shape. Constrained nodes which share an element edge with one or more unconstrained nodes define the crack front. At least one node on the crack plane must be left unconstrained, otherwise no crack growth can occur. Multiple crack fronts can exist on the crack plane. Currently, the node release algorithm does not support focused meshes which use collapsed elements at the crack tip.

At present, only *l3disop* elements (8-node bricks) may be defined over the portion of the crack plane involved in the node release procedures. The model may contain 20-node ele-

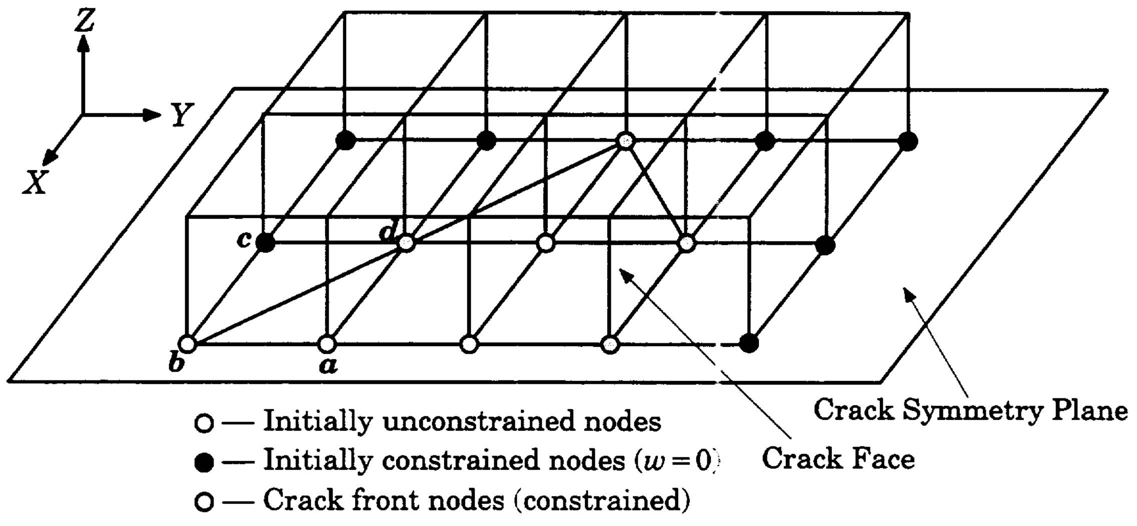

Z
Y
X
c
d
b
a
Crack Symmetry Plane
Crack Face
○ — Initially unconstrained nodes
● — Initially constrained nodes ($w = 0$)
○ — Crack front nodes (constrained)

FIG. 5.3—*Example Crack for General Node Release Crack Growth.*

ments and transition elements elsewhere in the mesh, but not over that portion of the crack plane involved in the crack growth process.

Figure 5.3 illustrates a simple arrangement for crack growth by node release in which the initial crack front geometry varies strongly. This also illustrates a possible configuration after some amount of non-uniform growth from an initially straight crack front. Here, the *z* direction coincides with the crack plane normal; filled and unfilled circles indicate crack plane nodes. The three unconstrained nodes on the crack plane (unfilled circles) define the initial crack (shaded area) with the corresponding five crack front nodes (shaded circles). The unconstrained node ***a***, for example, makes nodes ***b*** and ***d*** lie on the crack front. Node ***c*** is not on the crack front since it does not share an element edge with any unconstrained node on the crack plane.

This crack growth process as implemented in WARP3D is quite general. For example, it easily models non-uniform crack growth along an initial semi-elliptical surface flaw in a flat plate, pipe, pressure vessel, etc. Moreover, multiple crack fronts are detected automatically by the growth processors and the cracks may grow to coalesce during the loading process.

A less general capability is also implemented for those situations requiring uniform growth along a front, e.g., to model growth in thin materials where very local 3–D effects at the front can influence the CTOA but where growth is satisfactorily represented as uniform over the thickness. In such cases, the number of elements defined over the crack front in the model must remain fixed from one increment of growth to the next. Consider, for example, a thin plate modeled with eight, variable thickness layers of 8-node elements over the initial crack front. After each increment of growth, there must always be eight similar layers of elements. The element size in the direction of crack growth at each front should be maintained at a constant value in the mesh. At locations on the crack plane beyond the growth region, the mesh can be transitioned to other configurations and types of elements over the thickness. With this restricted growth model, multiple initial cracks may be defined and those cracks may also grow to coalesce.

5.3.2 Input Commands

The following commands initiate the definition of parameters for crack growth by node release:

```
crack (growth) (parameters)

                             {    none      }
type (of) (crack) (growth)   { node release }
```

where *none* turns off subsequent node release during the analysis. Unlike the element extinction algorithm, crack growth by node release can be turned on and off at any time. Crack growth by node release and crack growth by element extinction cannot both be used in a single analysis.

The node release algorithm requires specification of the crack plane. The command to describe the crack plane is

where *x*, *y*, or *z* denotes the direction of the normal to the plane, and *<number>* sets the position of the plane relative to the origin (e.g. the command *crack plane normal z coordinate 5.0* describes the plane $z = 5.0$). This command has no default; without a crack plane definition, program execution terminates at the next *compute* command.

```
crack (plane) normal  { x }  coordinate <number>
                      { y }
                      { z }
```

Critical Angles for Growth — Non-Uniform Growth Along a Front

The processors release a crack front node when the crack tip opening angle (CTOA) at the node is within 1% of a critical value. WARP3D requires two critical angles; one for initiation of crack growth (*initiation angle*) and one for continued growth (*release angle*). The *initiation angle* only applies to nodes on the initial crack front, while the *release angle* applies to nodes added to the crack front as it evolves; this simulates the difference in the deformation required for crack initiation and continued growth. The commands for specifying these critical angles are:

```
constant (front) growth off

angle (for) release < critical angle: value >

angle (for) initiation < critical angle: value >
```

where *<critical angle>* has units of degrees. Note that the values describe the full angle at the crack tip, and thus are twice the angle from the model to the crack plane. The critical angles have no default value, and cause WARP3D to stop at the next *compute* command if not set.

The node release processors compare the appropriate critical angle with the current opening angle for each crack front node. If a crack front node has several opening angles (i.e., shares element edges with multiple unconstrained nodes), the processors compare each angle with the critical value. Computation of the current opening angle proceeds as follows (see Figure 5.4): Consider crack front node ***a*** and the corresponding unconstrained node ***b***. Construct a line through nodes ***a*** and ***b***. Find the angle Θ_1 between the constructed line (***a-b***) and the projection of the constructed line in the crack plane (***a-d***); this angle is half of the current opening angle. Repeat this process for nodes ***b-c***. If $2\Theta_1 > (0.99 \times$ critical CTOA), the processors release the constraint on node ***a***. If $2\Theta_2 > (0.99 \times$ critical CTOA), the processors also release the constraint on node ***c***.

When any of the opening angles at a crack front node exceed 0.99 × the specified critical value, the crack growth processors release the constraint on that node in the normal direction of the crack plane; other constraints, if any, remain unchanged. After releasing the node, the processors convert the nodal reaction force in the normal direction into a nodal applied force, which decreases linearly to zero either over a number of sequential load steps or using a simple traction-separation law. Multiple nodes can be in various stages of release at any time. A subsequent section gives the commands that describe the force release models.

Critical Angles for Growth — Enforced Uniform Growth Along a Front

In some modeling situations, the enforcement of uniform growth along a crack front becomes desirable, e.g., growth in very thin materials. In such cases, the user specifies a single "master" node for each initial crack front. The crack growth processors then locate all adjacent nodes on each of the crack fronts. The CTOA at a user-specified distance behind the front of each master node is monitored to govern the crack growth process. When the CTOA for a master node reaches the critical value, all nodes on that front are released si-

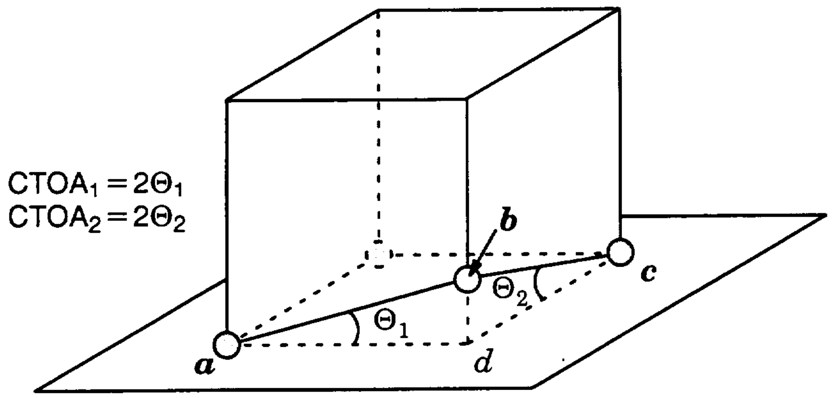


FIG. 5.4—*Computation of Crack Tip Opening Angles.*

multaneously. The corresponding master node on the new crack front is located automatically by the growth processors. This process repeats (immediately) until the crack front advances a distance equal to the same distance behind the front at which the CTOA is computed (to the nearest whole element size). To enable such automatic processing, the number of nodes along the new crack front must be identical to the number before growth. WARP3D requires that the user specify as input the number of crack front nodes to support error checking and automatic updating of master node lists. The commands to invoke uniform growth are:

constant (front) growth on

master (node) list   <master nodes: integer list>

number (of) nodes (along) front   <integer>

where this capability is *off* by default. The *constant growth* command must precede the remaining two commands. If invoked, both the master node list and the number of front nodes must be specified. One master node is required per initial crack front. The ordering of master nodes in the <list> is immaterial. The overshoot control features key only on the master nodes when uniform growth is enforced.

The commands to request uniform growth *must follow* specification of the crack plane normal. This enables significant, immediate error checking and validation.

For enforced uniform growth, WARP3D computes the CTOA for each crack front master node at a user-specified distance behind the front. The specified distance does not need to correspond to a node location. The uniform front extension makes possible this very desirable modeling capability. The *angle for release* commands are modified as follows to support enforced uniform extension:

angle (for) release < critical angle: value >   distance <value>

angle (for) initiation < critical angle: value >   distance <value>

where the distances specified for initiation and growth may be different. The different value for initiation proves convenient to set the criterion for initial growth using a critical value of the CTOD, for example, based on the usual 90-degree intercept definition of CTOD. To define this situation, set the distance as 0.5 × the critical CTOD and the initiation angle as 90-degrees.

In the above discussion, let L_c denote the distance behind the crack front at which the CTOA is computed (L_c is the *distance* value specified in the above two commands). Whenever crack growth occurs, the front is advanced this same distance forward in a *single* node release process. Generally, the distance L_c corresponds to a multiple of the element dimension, denoted L_e, on the crack plane in the direction of crack advance. This is not required but it is most often the case so that the CTOA is computed at a node location. To enable WARP3D to allocate necessary data structures and to perform consistency checks, the code needs the value for L_e. This information is specified with the command

```
characteristic (length) < length:number>
```

where $L_e \leq L_c$. When L_c is a simple multiple of L_e, the crack advances by L_c/L_e whole elements in the direction of growth during each release process. When L_c is not a simple multiple of L_e, the crack advances by the nearest whole number of elements. If $L_c/L_e = 1.4$, for example, the crack front advances one element forward; if $L_c/L_e = 1.6$ the crack advances two elements forward, etc.

Critical Angles for Growth — Crack Growth on Demand

In some modeling situations, users may desire to force crack growth immediately before the next load step of the analysis irrespective of the crack growth criterion. We expect this capability to be used from the start of crack growth in an analysis. The recommended set-up of the crack growth parameters is:

- Set the critical angles for initiation and continued growth to have very large values.
- Use the step release method to relax reaction forces on newly released nodes (use of the traction separation method for forced growth is complex due to the changing angles)
- The overshoot and load-reduction features are not applicable for this type of analysis. They should not be included in crack growth parameters.
- Suppose the user wants the next increment of crack growth to occur at the beginning of load step n. Then, after step $n-1$ has completed, and before the solution for step n begins, re-define the critical angles to have very small values. The crack growth processors then trigger an increment of growth (for general 3-D at all current front nodes or at all crack fronts for enforced uniform growth).
- Right after the solution for step n, reset the critical angles back to very large values.
- Repeat the above sequence each time an increment of crack growth is needed during the analysis.

Overshoot Control

Analyses conducted using large load step sizes may experience significant "overshoots" of the critical angle before the crack growth processors detect the event at the beginning of the next load step. For example, if the critical angle is 10°, large step sizes may cause node release at some angle larger than 10°, due to the discretization of the loading path into finite size load steps. The crack growth processor includes a mechanism to reduce this overshoot "error" of the CTOA at release. The mechanism predicts the change in CTOA for each crack front node over the next step by extrapolating the previous step value, based upon the user-specified load increment (the numeric "multiplier" in the step definition). If the predicted angle exceeds a specified percentage of the critical angle, the growth processors reduce the loading multiplier by the amount required to eliminate the overshoot. However, the nonlinearity of the solution makes this only an approximate process. The procedure performs this computation at all crack front nodes and uses the largest reduction found. A limit on the load reduction may be specified to avoid excessively small load steps. This process executes prior to the actual solution for a load step and simply scales the incremental loads (and non-zero constraints). Consequently, the adaptive load step algorithm used to enhance convergence of the Newton iterations remains fully available if needed by the process that directs the solution for a load step.

The commands to request overshoot control are:

$$\text{\underline{over}shoot (\underline{cont}rol)} \left\{ \begin{matrix} \text{\underline{off}} \\ \text{\underline{on}} \end{matrix} \right\} \left[\left\{ \begin{matrix} \text{\underline{per}cent (\underline{over}shoot) <overshoot\_limit: value>} \\ \text{\underline{min}imum (\underline{red}uction) <reduction limit: value>} \end{matrix} \right\} \right]$$

where *<overshoot limit>* specifies the maximum allowable overshoot as a percentage of the critical angle, and *<reduction limit>* specifies the smallest allowable load factor expressed as a percentage of the original load step size. By default, the overshoot control mechanism is disabled. When activated with the above command, the default value for maximum overshoot is 2.0 (i.e., 2.0%), and the minimum load reduction is 10.0 (i.e., 10.0%).

Numerical testing shows that overshoot control is highly effective in plane-strain models and in 3-D models with enforced uniform growth along the crack fronts. However, it can be less effective in general 3-D crack growth when nodal force release follows the traction-separation model coupled with a small release fraction (β). This follows because multiple nodes can be in various stage of release along each crack front with strong, nonlinear interactions between them as the remaining force decreases to zero. The use of β-values in the 0.3-0.5 range, or smaller load steps, improves the ability of the mechanism to reduce the overshoot error in 3-D.

Control of Simultaneous Node Releases

When load steps are too large, only one or two steps may occur between consecutive node releases (with or without overshoot control). This may allow the force release process to affect adversely the stress-strain history of material ahead of the crack front. To alleviate this problem, WARP3D provides a feature to reduce automatically the load step size based on the number of load steps between consecutive node releases. The reduction algorithm operates as follows for general 3-D growth. Consider a current crack front node with several neighbors having opening angles larger than the critical value for release. If the number of load steps since each of these neighbors were released is less than a user-specified value, the growth processor sets a permanent 50% reduction on all future load step sizes. This 50% reduction may occur any number of additional times if the load steps remain too large.

The command to request automatic load reduction is:

$$\text{\underline{auto}matic (\underline{load}) (\underline{reduc}tion)} \left\{ \begin{matrix} \text{\underline{off}} \\ \text{\underline{on}} \end{matrix} \right\} \text{[(\underline{min}imum) \underline{step}s <steps: integer>]}$$

where <*steps*> denotes the minimum number of steps allowable between node releases. By default, the minimum number of steps between node releases is 6 and the load reduction algorithm is off. The procedure works in the same manner for enforced uniform growth but only the master node at each crack front enters into the decision process.

Status Printing

The node release procedures provide a convenient *printing* option to simplify interpretation of the growth process. The command has the form

print (status) { off | on }

where the keyword *on* or *off* is required. At the beginning of each load step, this printing option prints the current crack front nodes and the corresponding crack tip opening angles.

The *print* command also provides options to list the initial crack front nodes and the crack plane nodes, both located by the automatic search procedures built into crack growth processors. The command syntax is:

print crack front nodes

print crack plane nodes

5.3.3 Release Models for Reaction Forces

Release Over a Fixed Number of Load Steps

The simplest procedure to relax the reaction force for a released node employs a fixed number of load steps. The commands to specify this option are

force (release) (type) steps

release (steps) < integer >

This force release model is the default option, with a default value of 5 steps. The user *cannot* alter the number of release steps once the processors for crack growth release nodes. A complete example for general, 3-D crack growth input using this force release procedure is:

```
crack growth parameters
  type of growth node_release
  crack plane normal z coordinate 0.0
  constant front growth off
  angle for release 20.0
  angle for initiation 100.0
  force release type steps
  release steps 10
  overshoot control on percent overshoot 10.0 minimum reduction 20.0
  automatic load reduction on minimum steps 8
  print status on
```

While the above procedure is simple, computed solutions may exhibit a dependence on the number of load steps employed in the analysis. If the load steps are large, for example, the above procedure (with release steps > 1) artificially restricts opening of the crack faces. In analyses with very small load steps, the reaction forces may reduce to zero too quickly.

Traction-Separation Procedure

The linear traction-separation model improves upon the fixed number of steps approach by placing the force release process on a more physical basis. In this model, 20% of the reaction force is released immediately; the remaining 80% decreases linearly with the increased opening displacement of the node, reaching zero at a specified opening displacement. The immediate release of 20% proves removes the possibility of spurious crack closing. A fraction, β, of the critical CTOA for continued crack growth (release angle) provides a convenient means to define the opening displacement. Consider Figure 5.5a, where node $\boldsymbol{a}$ represents a released crack plane node and node $\boldsymbol{b}$ represents a constrained crack plane node. At release, the reaction force on node $\boldsymbol{a}$ changes to an applied nodal force. After further deformation, node $\boldsymbol{a}$ opens to position $\boldsymbol{a}'$. In this position, the angle between the crack plane and the element edge ($\boldsymbol{a}'$-$\boldsymbol{b}$) equals the specified fraction of one-half of the critical CTOA ($\beta \times critical\_angle/2$). This is the release height from the node to the crack plane (line segment $\boldsymbol{a}'$-$\boldsymbol{a}$) at which the nodal force is fully reduced to zero. When node $\boldsymbol{a}$ lies between positions $\boldsymbol{a}'$ and $\boldsymbol{a}$, the reaction force decreases linearly, as described by:

$$\gamma = 0.8 - \frac{Current\ \ Distance}{Release\ \ Height} \qquad (0 \le \gamma \le 0.8) \tag{5.3}$$

where γ is the fraction of the reaction force on the node (see Figure 5.5b). Note that there is an immediate 20% reduction of the nodal force to prevent spurious re-closing of the crack face.

Use with general 3-D growth

Since there are no restrictions on the lengths of the element edges on the crack plane, edges of different length may connect to a crack plane node. Therefore, using element edges in the calculation of the release height may result in multiple height values. To resolve this ambiguity, the traction-separation model employs a user-defined, characteristic length instead of the length of individual element edges. The characteristic length describes the generic edge length on the crack plane; it could be the average of the edge lengths, a median edge length, or based on some other criterion. Using the characteristic length, the release height, D, becomes:

$$D = L\ \tan(\beta\theta_c/2) \tag{5.4}$$

where L is the characteristic length, β is the release fraction, and θ_c is the critical CTOA for continued crack growth. The choice of characteristic length, L, directly affects operation of the node release procedures. If the edges connected to a crack plane node are significantly smaller than the characteristic length, all the neighbors of the node may be released before the reaction force reduces to zero. Also, if the characteristic length is too small, the reaction forces may dissipate too quickly. In Figure 5.5a, the characteristic length corresponds exactly with the length of the element edge on the crack plane.

Input commands to specify the traction-separation model are:

force (release) (type) traction(–separation)

release (fraction) < β factor:number>

characteristic (length) < length:number>

The node release processors require the input of a characteristic length; there is no default value. β is the fraction of the critical CTOA for continued crack growth (release angle), with

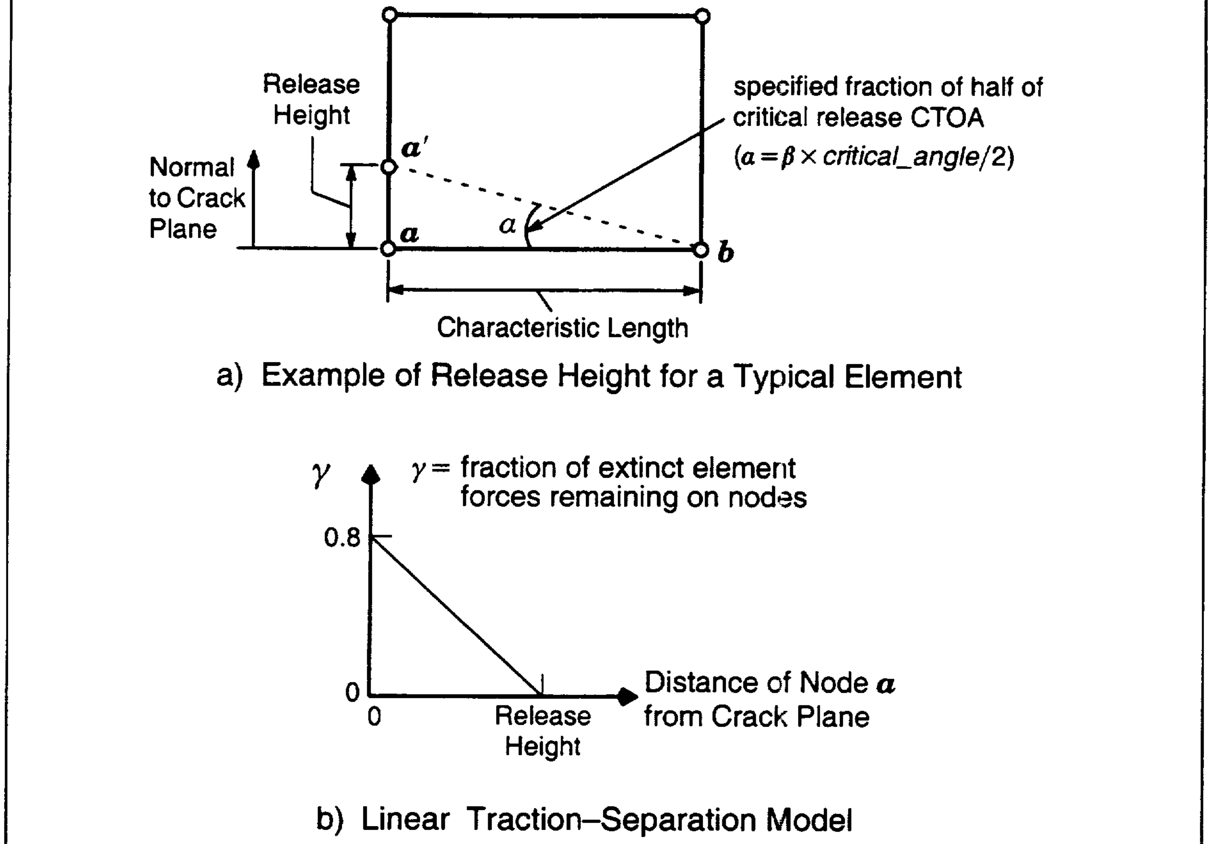

Release Height
specified fraction of half of critical release CTOA
(α = β × critical_angle/2)
Normal to Crack Plane
a'
a
α
b
Characteristic Length
a) Example of Release Height for a Typical Element
γ
γ = fraction of extinct element forces remaining on nodes
0.8
0
0
Release Height
Distance of Node a from Crack Plane
b) Linear Traction–Separation Model

FIG. 5.5—*Traction-Separation Model for Release of Nodal Reaction Forces*

a default value of 0.1. Neither of these values may be changed after nodes have been released. A complete example of crack growth input using the traction-separation model is:

```
crack growth parameters
  type of growth node_release
  crack plane normal z coordinate 0.0
  constant front growth off
  angle for release 20.0
  angle for initiation 100.0
  release type traction-separation
  release fraction 0.2
  characteristic length .01
  automatic load reduction on minimum steps 8
  overshoot control on percent 1.0 minimum reduction 1.0
  print status on
```

Use with enforced, uniform growth

Since there are no restrictions on the lengths of the element edges on the crack plane, edges of different length may connect to a crack plane node. Therefore, using element edges in the calculation of the release height may result in multiple height values. To resolve this ambiguity, the traction-separation model employs the distance, L_c, for continued growth (defined previously with the critical angle) instead of the length of individual element edges. Using L_c, the release height, D, becomes:

$$D = L_c \tan(\beta\theta_c/2) \tag{5.5}$$

where β is the release fraction, and θ_c is the critical CTOA for continued crack growth.

Input commands to specify the traction-separation model are:

force (release) (type) traction(–separation)

release (fraction) < β factor:number>

where β is the fraction of the critical CTOA for continued crack growth (release angle), with a default value of 0.1. This value cannot changed after any nodes have been released. A complete example of crack growth input using the traction-separation model with enforced uniform growth is:

```
crack growth parameters
  type of growth node_release
  crack plane normal z coordinate 0.0
  constant front growth on
  master node list 3489 2032
  number of nodes along front 9
  angle for release 5.1 distance 0.040
  angle for initiation 10.2 distance 0.060
  characteristic length .010
  release type traction-separation
  release fraction 0.2
  automatic load reduction on minimum steps 8
  overshoot control on percent 1.0 minimum reduction 1.0
  print status on
```

5.3.4 Node Release Algorithm

During the initialization phase, the node release processors find all of the nodes on the crack plane within a tolerance based on the maximum model dimension in the direction normal to the crack plane. From this information, the processors identify the crack front nodes as all the nodes constrained in the normal direction on the crack plane which share an element edge with an unconstrained node on the crack plane. The program terminates at the next *compute* command if it cannot find any crack plane nodes.

At the beginning of each load step n (n>1), or adaptive sub-step, the processors calculate the crack tip opening angle (CTOA) for each node on the crack front (general 3-D growth) or each master node (enforced uniform growth). When the angle reaches the critical value, the following actions occur:

- The processors release the constraint normal to the crack plane .
- The reaction force normal to the crack plane changes into a nodal force which decreases in a linear fashion over subsequent load steps. Because the reaction force changes into a nodal force and thereafter behaves as an ordinary (user-specified) force, crack growth does not effect operation of the (global) adaptive step algorithm. Such global adaptivity of the load step sizes often proves essential for obtaining converged solutions following a growth increment.
- The crack front expands to include the constrained nodes in the crack plane which share an element edge with the released node.
- The crack growth processor modifies the nonlinear solution parameters as follows: a linear stiffness matrix is requested for the first iteration of the upcoming load step. Such use of the *linear* stiffness matrix for the structure properly accounts for elastic unloading that generally occurs immediately following a node release. This modification in the nonlinear solution procedure is found necessary to maintain high rates of convergence.

Eventually, only a few constrained nodes may remain on the crack plane. This state may cause the solution to fail to converge and/or give anomalous displacement values. The analysis should be halted before this situation occurs.

Effect of User Constraint Changes

When changing the global constraints between steps, the user may inadvertently re-constrain a previously released node. To alleviate this problem, the processors check all the previously killed nodes at the beginning of each load step and remove all new constraints on these nodes in the direction normal to the crack plane.

5.3.5 Analysis Guidelines

Crack growth analyses using node release place severe demands on the nonlinear solution procedures implemented in the code. Experience derived from a number of crack growth analyses suggests the following guidelines as starting points for consideration.

Number of Load Steps

Controls the overall solution convergence and resolution of the deformation process. Crack growth analyses generally reveal strong history effects; the computed R-curves and near-tip fields are sensitive to the load step sizes and to details of the reaction force release at crack extension. As an example, consider an SE(B) model with $a/W = 0.6$ and $W = 50$ mm. The material is a mild structural steel with moderate strain hardening. Element sizes along the crack plane are 0.1 mm. A typical CTOA for initiation of growth is 80-90° and 15° for continued growth. Successful analyses using 100 equal size load steps to initiate growth and 400 additional steps to extend the crack 4 mm have been generated. These solutions employ displacement control loading; small geometry change and large geometry change solutions require essentially the same number of iterations for convergence at each step. The computed J-Δa curve reveals very minor differences for a larger number of load steps or even a slightly smaller number of steps. In this example analysis, the crack extends by 1 element every 10 load steps. Analyses with many fewer load steps also obtain converged solutions but reveal dependencies on the load step sizes.

Solution Procedures

WARP3D provides the user full control of the incremental-iterative solution process. Current recommendations for crack growth analyses are as follows:

solution technique:	platform specific sparse solver for all 2-D type models and moderate size 3-D models. PCG for all large 3-D models (try the *diagonal* preconditioner first as it is more efficient). For "shell" type models, always use the sparse, direct solvers.
adaptive solution:	use the *on* option, especially for preliminary analyses. In parametric studies with a good knowledge of the loading steps required in hand, adaptive solutions should not be used, i.e., the code will simply "re-learn" the correct load step sizes required for convergence. However, we have found cases in which the solution appears to "stick" and will not converge. All such cases observed thus far have been successfully solved by forcing the solution to continue on to the next step, which invariably converges. For such situations, set *adaptive off* and *nonconvergent solutions continue*.
extrapolation:	displacement extrapolation at the start of a new load step is *on* by default. This procedure generally accelerates the convergence of solutions including geometric nonlinearity, but *may* cause convergence problems during crack growth. The current recommendation is to set *extrapolation off* once crack growth begins.

linear stiffness:	the option to use a linear stiffness to resolve iteration 1 of a step helps in cases with large regions of linear-elastic unloading. Normally, this option should *not* be used (slows convergence) and it is *off* by default. The crack growth processor forces its use for the next step whenever a new node is released (see previous notes on crack growth algorithms).

Convergence Tests and Tolerances

This is perhaps the most difficult decision in specifying nonlinear solution parameters. Many problems with non-convergent solutions during crack growth have been traced to tolerances set *too large*. Experience suggests using a combination of two convergence tests as illustrated by the command:

```
convergence test norm res tol ???? maximum residual tol ????
```

where ???? are replaced by actual values appropriate to the actual analysis. Suggested starting values for crack growth analyses are 0.01 and 0.001 for the first and second tests, respectively. The *norm res* test provides good control of the overall convergence of the solution but often does not indicate proper convergence in small elements ahead of the crack tip. The second test offers an indirect means of controlling residuals of these elements. Please refer to the manual sections defining these tests.

Good convergence rates and accurate solutions require small residual forces on nodes in the crack tip region. The *output internal forces <node list>* command prints the residual forces on the specified nodes following convergence (or non-convergence) of the solution. These forces should be very small relative to forces exerted by these elements on their nodes due to the internal stresses. A simple estimate for the forces is given by the following procedure: multiply the material flow stress by the element volume/20. Then each component of the residual force at nodes should be 2-3 orders of magnitude smaller than the estimated internal force. The internal forces output also include actual reactions for constrained dof and should not be included in these comparisons.

Chapter 6

Contact Procedures

6.1 Introduction

Many of the most difficult problems in solid mechanics involve the contact interaction between deformable bodies. Key examples include metal forming processes (rolling and die-casting) and crash-impact problems. In fracture mechanics, problems of crack closure and the proper modelling of experimental conditions often necessitates treatment of contact. Regions of a finite element model which undergo contact have boundary conditions that vary with the amount of deformation. In contact, implicit analyses are especially challenging as the boundary conditions may change abruptly during a load increment. This introduces severe nonlinearity into analyses, with corresponding changes in model behavior and solution convergence rates.

The large volume of research on contact algorithms offers a range of complexity in available approaches to treat contact [92]. The literature typically focuses on two aspects of contact algorithms—contact detection and contact enforcement. Detection of contact in the most general form, where any part of a modeled body can interact with any other body and/or itself, often consumes most of the computation time for an analysis and introduces significant complexity in the code. Finite element codes often employ a number of techniques to simplify contact detection by explicitly identifying regions which may contact each other, or by limiting contact to simple rigid surfaces. Enforcement of contact proceeds along one of two approaches. Lagrange multiplier techniques eliminate penetration of contact surfaces by including additional constraint equations directly in the solution of the problem. The Lagrange multipliers, included as additional unknown scalar variables, become the forces required to eliminate penetration. This technique is very popular in implicit codes; however, the additional equations may cause the global structural stiffness matrix to become positive semi-definite, thus limiting the techniques available for solution of the corresponding linear system. The penalty method provides an alternative approach by introducing very stiff springs which move penetrating nodes back to the contact surface. This method allows some limited penetration between bodies. Increasing the penalty parameter (spring stiffness) reduces the amount of penetration. Although the penalty method retains a positive definite stiffness matrix, very high penalty parameters can make the stiffness matrix ill-conditioned, causing convergence problems and significant round-off error in the final results. Efficient analysis requires care in choosing an appropriate penalty parameter. Other approaches to enforce contact conditions involve hybrids between these methods ([35],[92]).

The contact algorithms in WARP3D implement frictionless, rigid-body contact using a standard penalty method. Contact between deformable bodies and self-contact are not supported in this implementation. Contact occurs between nodes of a finite element mesh and a user-defined set of rigid contact surfaces. Currently, WARP3D supplies three geometries of surfaces which can be arbitrarily oriented in space: finite-sized rectangular planes, cylinders, and spheres. Additional surfaces may be added as needs warrant. The contact processors allow the assignment of velocities to the contact surfaces to simplify simulation of moving boundaries. Each contact surface requires a stiffness (penalty parameter) which limits the penetration between the model and the surface. Specification of an appropriate stiffness can be difficult; Section 6.4 provides some guidance on this topic. The algorithms also address issues of multiple contact, where a node penetrates more than one contact surface. This enables the appropriate treatment of intrinsic and extrinsic corners in contact analyses. The implementation of contact is completely compatible with all other parts of WARP3D, including crack growth, finite deformation, all material models, restart facilities, and parallel execution.

This chapter continues with a description of the contact algorithms used in WARP3D, including an overview of the penalty method, a summary of the contact detection techniques, a description of the algorithms which handle intersecting contact surfaces, and key details of the parallel implementation of contact. A section on contact input describes the necessary commands for contact specification, as well as some restrictions on contact models. Section 6.4 provides advice on performing analyses with contact. The chapter concludes with a section of examples which illustrate three WARP3D contact analyses: rolling of a steel bar, crushing of a pipe, and crack closure on a pin-loaded C(T) specimen.

6.2 Numerical Procedures

6.2.1 Overview of the Penalty Method

The penalty method defines a simple approach to enforce displacement constraints in the solution of a finite element model. It has a variety of applications, including the imposition of multi-point constraints, incompressible material models, mesh locking problems, and contact enforcement [16]. The penalty method follows from the minimum potential energy formulation of the finite element method. The potential energy for a finite element model is

$$\Pi = \frac{1}{2}\boldsymbol{U}^T\boldsymbol{K}\boldsymbol{U} - \boldsymbol{U}^T\boldsymbol{F} \tag{6.1}$$

where $\boldsymbol{U}$ is the vector of nodal displacements, $\boldsymbol{K}$ is the global stiffness matrix, $\boldsymbol{F}$ is the corresponding nodal force vector, and Π is the potential energy. An equilibrium configuration (deformed shape) for the structure makes the potential energy take on a local minimum. The minimum potential energy occurs when $\partial\Pi/\partial\boldsymbol{U} = 0$, thus:

$$\frac{\partial\Pi}{\partial\boldsymbol{U}} = \boldsymbol{K}\boldsymbol{U} - \boldsymbol{F} = 0 \tag{6.2}$$

which results in the standard equilibrium equations $\boldsymbol{K}\boldsymbol{U} = \boldsymbol{F}$.

The penalty method as applied to contact adds an additional term to Eq. 6.1:

$$\Pi = \frac{1}{2}\boldsymbol{U}^T\boldsymbol{K}\boldsymbol{U} - \boldsymbol{U}^T\boldsymbol{F} + \frac{1}{2}\boldsymbol{P}^T\boldsymbol{\alpha}\boldsymbol{P} \tag{6.3}$$

where $\boldsymbol{P}$ corresponds to the penetration displacement of nodes into contact surfaces, and $\boldsymbol{\alpha}$ corresponds to the "penalty parameters", constants which determine the relative importance of forcing the penetration to zero. An increase in the magnitude of the penalty parameter causes the penetration to have a stronger effect on the total potential energy, thus enforcement of $\boldsymbol{P} = 0$ becomes proportionally more strict. The terms in $\boldsymbol{\alpha}$ have units of stiffness so that $1/2\boldsymbol{P}^T\boldsymbol{\alpha}\boldsymbol{P}$ has units for energy. Solution of $\partial\Pi/\partial\boldsymbol{U} = 0$ for Eq. 6.3 transforms the equilibrium equations to $\boldsymbol{K}'\boldsymbol{U} = \boldsymbol{F}'$, where $\boldsymbol{K}'$ is the effective structural stiffness matrix and $\boldsymbol{F}'$ is the effective force vector including the effects of contact.

An equivalent approach to implement the penalty method for contacting bodies creates springs at the contact points (see Figure 6.1). The springs, placed between each penetrating node and the closest point on the penetrated surface, have a very high stiffness which reduces the penetration nearly to zero. The spring stiffness corresponds to the penalty parameter, while the amount of remaining penetration corresponds to the error in the enforcement of the constraint. A larger spring stiffness decreases the magnitude of penetration after introduction of the spring. However, too large a spring stiffness can cause numerical difficulties. Addition of such a spring affects two parts of the finite element calculations: inclusion of the contact force into the residual force vector, and addition of the spring stiffness in to the global stiffness matrix. Experience indicates that increasing the stiffness of the spring slightly when including it in the stiffness matrix eliminates oscillation problems caused by the over-compensation of penetration. Consequently, WARP3D uses a penalty stiffness 0.1% higher than the user-specified value for the stiffness matrix calculations. Furthermore, WARP3D adds each spring stiffness into the corresponding element stiffness matrices instead of directly into the global stiffness matrix. Thus, for example, if 6 elements connect to a contacting node, then each element stiffness receives 1/6 of the total spring stiffness introduced at that node. This approach allows full use of the element-by-element architecture inherent inside WARP3D, as well as the linear preconditioned conjugate gradient solver (LNPCG). Contact between two deformable bodies requires application of the contact force to the penetrating node and the penetrated element. However, rigid body contact eliminates the need to compute forces on penetrated elements; the contact springs only affect penetrating nodes. This greatly simplifies the calculation of contact forces and the additions to element stiffness matrices [21].

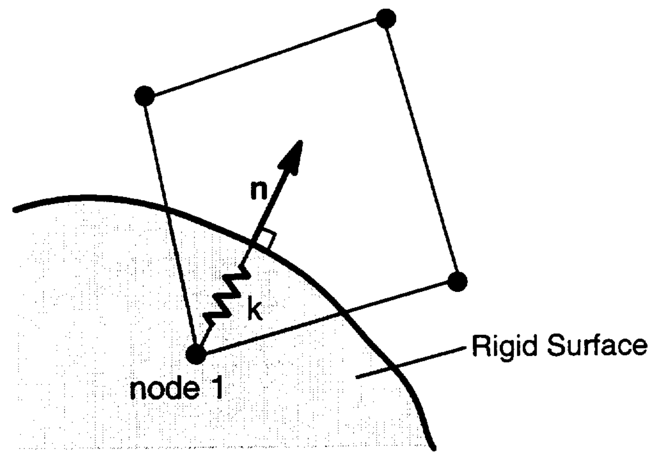


FIG. 6.1—*Illustration of penalty method*

Addition of the spring stiffness into the element stiffness matrix seriously degrades the convergence of the LNPCG solver. A large spring stiffness increases the spread of eigenvalues for the system, thereby increasing the number of iterations required for convergence, if convergence remains possible at all. For this reason, the specified value for the penalty parameter requires additional care when employing the LNPCG solver; see Section 6.4 for details on choosing the contact stiffness. Furthermore, diagonal dominance of the stiffness matrix is crucial for effective LNPCG convergence. If the contact spring is orthogonal to one of the global coordinate directions, then the spring stiffness adds solely to a corresponding diagonal term in the element stiffness matrices. However, if it lies skewed to the global axes, then part of the spring stiffness adds to off-diagonal terms, reducing diagonal dominance of the element stiffness matrix. To alleviate this problem, the contact processors construct a local coordinate system at the penetrating node which is orthogonal to the spring force. All global data values at the node are rotated into the new coordinate system. As a result, the spring stiffness adds directly into the diagonal of the element stiffness matrix. If the node has no explicit constraints, then formation of the nodal coordinate transformation is straightforward. If the node has one constraint, then formation of the transformation is only possible if the spring force and the direction of the constraint are orthogonal. In cases where formulation of a nodal coordinate transformation is not compatible with specified constraints, the penalty stiffness terms add to the element stiffness in global coordinates. If a node undergoing contact utilizes a (local) nodal coordinate transformation defined previously through user input, then calculation of contact becomes difficult; currently, the contact processors print an error message and stop execution of the code when this occurs.

6.2.2 Contact Detection/Calculation

WARP3D determines contact between nodes of the finite element mesh and a set of rigid contact surfaces at the beginning of each global Newton iteration to solve the equilibrium equations. During the contact detection phase, contact processors compare all nodes in the structure with all defined contact surfaces. The implementation currently provides three geometries of rigid contact surfaces; finite-sized rectangular planes, cylinders, and spheres. When a node penetrates one or more of the contact surfaces, the contact algorithms compute the amount and direction of the penetration. This section describes the contact and penetration algorithms for each of the contact surface geometries. Please see Section 6.3.3 for additional description of the contact surfaces and the corresponding input.

Finite-Sized Rectangular Planes

The geometric description of the rectangular plane includes a base point corresponding to one of the corners of the rectangle, two vectors which extend along the edges of the rectangle from the base point to the two adjoining corners, and a normal vector. Figure 6.2 shows the geometric description and outlines the contact detection algorithm.

1. Compute a position vector between current location of the node $\boldsymbol{p}$ and the base point of the rectangle

   $$\boldsymbol{v}' = \boldsymbol{p} - \boldsymbol{p}_R$$

2. Compute the dot product between this vector and the normal to the plane; this is the negative of the penetration. If penetration is negative or zero, no contact.

   $$d = -(\boldsymbol{v}' \cdot \boldsymbol{n}); \quad d \le 0 \Rightarrow no\ contact$$

3. Compute dot product between $\boldsymbol{v}'$ and the two edge vectors

   $$a = (\boldsymbol{v}' \cdot \boldsymbol{v}_1)$$
   $$b = (\boldsymbol{v}' \cdot \boldsymbol{v}_2)$$

4. If both dot products are between 0 and 1, then the node has penetrated the rigid surface

   $$0 < \{a, b\} < 1 \Rightarrow contact$$

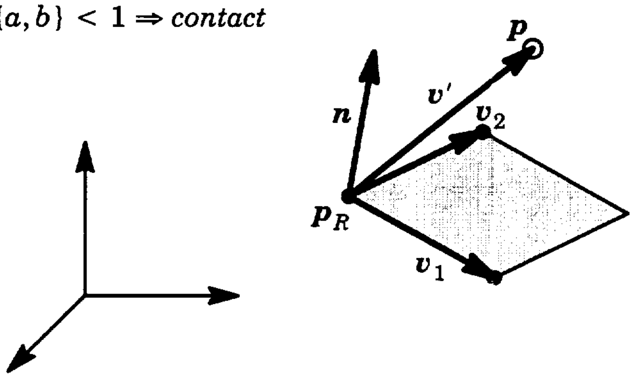


FIG. 6.2—*Contact Detection for Rectangular Plane*

Cylinder

The spatial orientation of the contact cylinder uses a base point, a vector pointing in the direction of the center line, the length of the cylinder, and the radius. Figure 6.3 outlines the contact detection algorithm.

1. Compute a position vector between the current location of the node $\boldsymbol{p}$ and the base point of the cylinder
   $\boldsymbol{v}' = \boldsymbol{p} - \boldsymbol{p}_C$
2. Compute the angle between the vector and the center line of the cylinder. If the angle is greater than 90 degrees, no contact is possible, so return.
   $$\theta = \arccos\left(\frac{\boldsymbol{v}' \cdot \boldsymbol{n}}{\|\boldsymbol{v}'\|}\right); \quad \theta > \frac{\pi}{2} \Rightarrow \textit{no contact}$$
3. Calculate penetration: radius of cylinder minus distance from node to closest point on center line. If there is no penetration, return.
   $$d = R - \|\boldsymbol{v}'\|\sin\theta; \quad d < 0 \Rightarrow \textit{no contact}$$
4. Compute distance between base point and projection of node on to center line. If distance is greater than the cylinder length, return.
   $$h = \|\boldsymbol{v}'\|\cos\theta; \quad h > L \Rightarrow \textit{no contact}$$

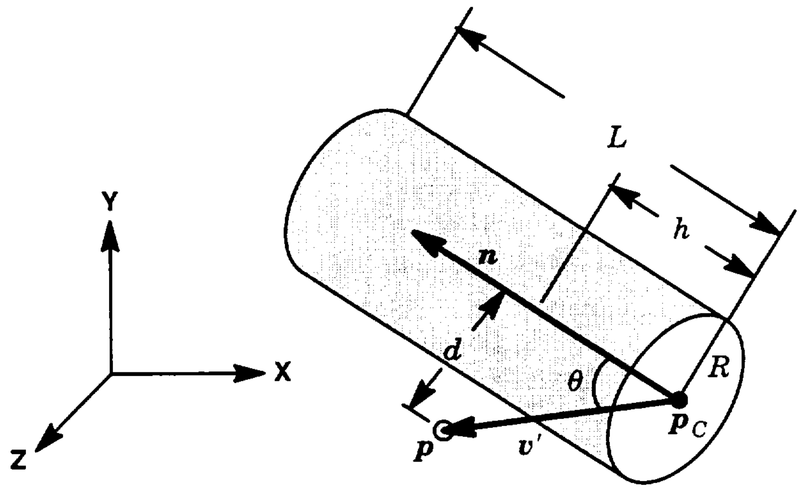


FIG. 6.3—*Contact Detection for Cylinder*

Sphere

The sphere requires only a base point and a radius to completely describe its orientation in space, making detection of contact very simple. The processors compute the vector between the center point of the sphere and the node. If the length of the vector is less than the radius of the sphere, contact occurs. The spring force acts in the direction of the calculated vector.

6.2.3 Penetration of Multiple Contact Surfaces

If a node penetrates several contact surfaces, the contact algorithms must return the node to the correct location. However, the choice of which set of surfaces should be considered is not always clear. For instance, Figure 6.4 shows an element with three of its nodes penetrating a set of three contact planes. Node *a* penetrates surface 3, so a single spring returns it to the correct location. Node *b* penetrates both surfaces 1 and 2, but the node should return only to surface 2. All three planes influence node *c*, but the correct return point is to the intersection of surfaces 2 and 3.

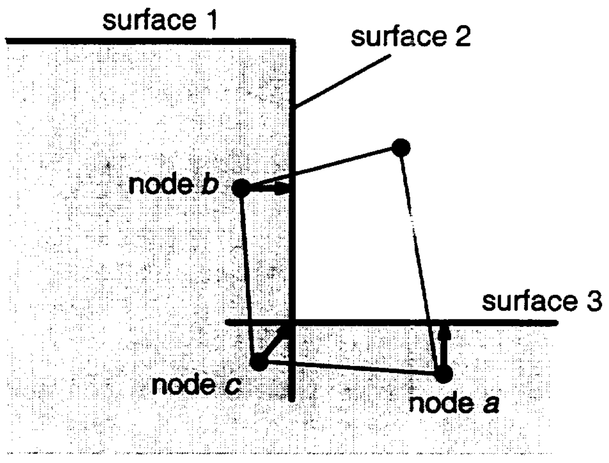


FIG. 6.4—*Nodes penetrating multiple contact surfaces*

To handle these conditions, the contact processors in WARP3D compare each of the penetrated contact surfaces by temporarily returning the node to a contact surface, and evaluating if the other shapes are still penetrated given the new location. By looping over the contact planes, this process eliminates all the superfluous contact surfaces, leaving only the set which must be simultaneously satisfied. The processors also calculate the location to which the node returns following the imposition of each of the valid contact shapes. Figure 6.5 provides additional details on the algorithm.

A separate algorithm constructs the new return location given a new contact surface, as shown in steps 3.c and 3.d of Figure 6.5. The processors compute the nearest point on the intersection of the previous return location and the new contact surface. The algorithm assumes during this step that all contact surfaces are planes. This causes some error for curved surfaces (cylinders, spheres), but if the load steps are sufficiently small, this error is negligible. Also, the algorithm may require additional Newton iterations for global convergence in problems with intersecting curved contact surfaces.

This algorithm appears to handle correctly cases where nodes penetrate multiple contact surfaces. Highly complicated constructions of contact surfaces or large load steps may cause this algorithm to fail; the use of relatively few intersecting contact surfaces and small load steps is advised.

6.2.4 Parallel Implementation

The parallel implementation of WARP3D uses a message passing approach (MPI) with a single "master" processor and many "slave" processors. To support contact during parallel execution, the root processor sends all slave processors the data for every defined contact

1. Compare the node location, $\boldsymbol{p}$, against the set of all contact surfaces, $\mathcal{C}$, assessing penetration P. Construct $\mathcal{C}^n$, a list of the k penetrated surfaces ordered from the smallest to largest penetration.

$$\mathcal{C}^n = \{\mathcal{C} : P(\mathcal{C}_i, \boldsymbol{p}) > 0\}$$

2. Consider the contact surface which the node penetrates the least. Set the current return location, $\boldsymbol{r}$, to the location which returns the node to this surface along its normal, $\boldsymbol{n}$.

$$\boldsymbol{L}(\mathcal{C}_i, \boldsymbol{p}) = \boldsymbol{p} - P(\mathcal{C}_i, \boldsymbol{p})\boldsymbol{n}(\mathcal{C}_i, \boldsymbol{p})$$
$$\boldsymbol{r} = \boldsymbol{L}(\mathcal{C}_1^n, \boldsymbol{p})$$

3. Loop over the remaining sorted contact surfaces:

$$\mathcal{C}_i^n : i = 2, 3, \dots, k$$

   a. Determine the penetration of the new contact surface by the current return location. If the new surface is not penetrated, remove it from the list of penetrated surfaces and proceed to the next contact surface.

$$P(\mathcal{C}_i^n, \boldsymbol{r}) < 0 \quad \Rightarrow \quad \mathcal{C}^n = \mathcal{C}^n - \mathcal{C}_i^n; \quad i = i + 1$$

   b. Find the location $\boldsymbol{r}_i'$ which returns the node to the current contact surface, and zero the current return location.

$$\boldsymbol{r}_i' = \boldsymbol{L}(\mathcal{C}_i, \boldsymbol{p}), \quad \boldsymbol{r} = 0$$

   c. Determine if new contact surface supercedes any previously verified surfaces. Loop over previously evaluated contact surfaces; if the location $\boldsymbol{r}_i'$ does not cause penetration of $\mathcal{C}_j$, remove $\mathcal{C}_j$ from $\mathcal{C}^n$. Otherwise, include contact plane into new return location.

$$j = 1, 2, 3, \dots, i - 1$$
$$P(\mathcal{C}_j^n, \boldsymbol{r}_i')\begin{cases} < 0 & \Rightarrow \quad \mathcal{C}^n = \mathcal{C}^n - \mathcal{C}_j^n \\ > 0 & \Rightarrow \quad \boldsymbol{r} = \boldsymbol{r} \cap \boldsymbol{L}(\mathcal{C}_j^n, \boldsymbol{p}) \end{cases}$$

   d. Include new contact surface in return location.

$$\boldsymbol{r} = \boldsymbol{r} \cap \boldsymbol{L}(\mathcal{C}_i^n, \boldsymbol{p})$$

4. Continue looping over all contact surfaces until all initially penetrated surfaces have been processed.

FIG. 6.5—*Algorithm for treating nodes which penetrate multiple contact surfaces*

surface. During the contact detection phase, processors assess contact for all nodes connected to elements which they own. Processors also compute the contact force for the nodes which they own. After all processors complete evaluation of contact for the appropriate nodes, the slave processors send their contact force contributions to the root processor, which reduces the contributions into the global contact force vector. The slave processors also send the computed nodal coordinate transformation matrices for all owned nodes; these are used in subsequent force calculations and result output.

6.3 Commands for Contact

6.3.1 Outline of Process

Contact in WARP3D takes place between a deformable mesh and a set of rigid contact surfaces. Specification of rigid contact surfaces may occur at any point in the input file. Input involves a block of commands, beginning with the *contact surfaces* command, followed by specifications for each contact surface. The description of a contact surface includes information about the type of surface, the geometry, the location in space, and the basic parameters (stiffness, surface velocity, etc.). Errors encountered in the input for a contact surface cause the contact processors to ignore the surface. Re-definition or removal of one or more contact surfaces may occur between any computational step. Adequate convergence of the analysis may require a change of the penalty parameters at different times in the analysis (see the tips section for additional information). Currently, WARP3D allows up to 20 defined contact surfaces in an analysis.

6.3.2 Initiating Contact Definition

The command to initiate the definition of contact surfaces has the form

```
contact (surfaces)
```

The contact input processors assume the commands following this statement pertain to contact. Contact input stops once the processors encounter a command they do not recognize.

6.3.3 Description of Contact Surfaces

WARP3D currently supports three geometries of contact surfaces: rectangular surfaces, cylinders, and spheres. Description of a contact surface requires the specification of the type of surface, the geometry, the orientation in space, and the basic parameters (stiffness, velocity, etc.).

The command to initiate the definition of a contact surface is:

```
                                       (  plane    )
surface  < surface number: integer >  {   cylinder  }
                                       (  sphere   )
```

where *surface number* is a number between 1 and 20. Only one surface may be assigned to a specific surface number; defining a surface with a specific surface number supercedes any previous surface definitions assigned to that number. Sequential numbering of contact surfaces is not required.

Information Required for all Contact Surfaces

All contact surfaces require a stiffness (penalty parameter). The command

```
(contact) stiffness  < stiffness: number >
```

specifies the stiffness for the contact surface currently under definition. The stiffness value must be a number greater than zero. Efficient analyses may mandate altering the stiffness of a surface during solution (see the tips section for additional information). There is no default for this value.

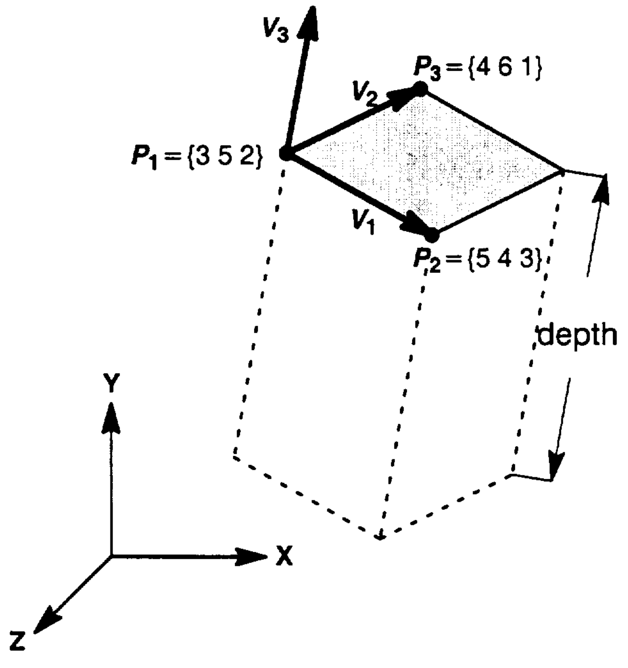
Contact surfaces may also move through space over time. The following command gives the velocity of the surface:

```
velocity  < dx: number > < dy: number > < dz: number >
```

where the velocity has units of distance per unit time. The default value for the velocity is zero in all directions. Note that this command requires appropriate setting of the time step size (see Section 2.9 on solution parameters). Currently, the contact processors translate, but do not rotate, contact surfaces. Rotation of a contact surface occurs only through user re-definition of the contact surface after each load step.

Rectangular Surface

The rectangular contact surface is a flat surface located in space with a given normal. The normal defines the positive (outward) side of the surface. The rectangular surface defines a right rectangular prism extending in the negative normal direction a depth as specified in the contact input. All nodes falling within the volume of the rectangular prism are penetrating nodes, with penetration defined as the distance to the rectangular surface. See Figure 6.6 for a typical rectangular contact surface. Section 6.2.3 describes the algorithms which handle intersections between multiple rectangular surfaces and/or other contact surfaces.



Normal Calculation:

$V_1 = P_2 - P_1 = [2\ {-1}\ 1]$
$V_2 = P_3 - P_1 = [1\ 1\ {-1}]$
$V_3 = V_1 \times V_2 = [.707\ .707\ 0]$

Sample Input:

```
contact surface
  surface 1 plane
     point 3 5 2
     point 5 4 3
     point 4 6 1
     depth 4
     stiffness 1.0e8
```

FIG. 6.6—*Definition of rectangular contact surface*

Designation of the geometry of the rectangular surface requires the specification of three points in space (use the following command three times):

```
point  < x: number > < y: number > < z: number >
```

These three points define the location in global coordinates of three corners for the rectangle. The first point serves as the base corner for the rectangle, while the second and third points are the corners adjacent to the base point; see Figure 6.6. The normal of the contact plane follows from the three points as follows. Denote the three points as $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, and $\boldsymbol{p}_3$. Define two vectors, $\boldsymbol{v}_1 = \boldsymbol{p}_2 - \boldsymbol{p}_1$ and $\boldsymbol{v}_2 = \boldsymbol{p}_3 - \boldsymbol{p}_1$. The normal is $\boldsymbol{v}_1 \times \boldsymbol{v}_2$. The normal vector defines the positive (outward) side of the contact plane; all nodes found on the negative side of the plane are penetrating nodes. This places several restrictions on the specification of the three points. The vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ must define a 90 degree included angle. Furthermore, the order in which the points are input determines the normal vector; flipping the definition of points 2 and 3 flips the direction of the normal. Users should verify that the normal vector has the correct direction.

The command to specify the depth of the rectangular contact plane is:

depth < depth value: number >

where *depth value* is a number greater than zero. The default value is 10^{10}.

A typical set of commands to define a rectangular contact surface is:

```
contact surface
  surface 1 plane
     point 3 5 2
     point 5 4 3
     point 4 6 1
     depth 4
     stiffness 1.0e8
     velocity 0.1 0.1 0.1
```

Cylindrical Surface

This is a right circular cylindrical contact surface is a cylinder with a finite length. Contact occurs on the curved surface — nodes penetrating the flat circular ends are moved to the nearest point on the cylindrical surface. Required geometrical input includes a base point, the direction of the center line measured from the base point, the length of the cylinder, and the radius. See Figure 6.7 for an sample contact cylinder.

To input the base point and direction vector, use the commands:

point < x: number > < y: number > < z: number >

direction < dx: number > < dy: number > < dz: number >

where the specified point and direction correspond to $\boldsymbol{P}$ and $\boldsymbol{V}$ in Figure 6.7. The direction does not need to be a unit vector; the contact processors automatically normalize the direction.

To input the length and radius, use the commands:

radius < R: number >

length < L: number >

where the specified radius and length and correspond to R and L in Figure 6.7. The values must be greater than zero, and have no defaults.

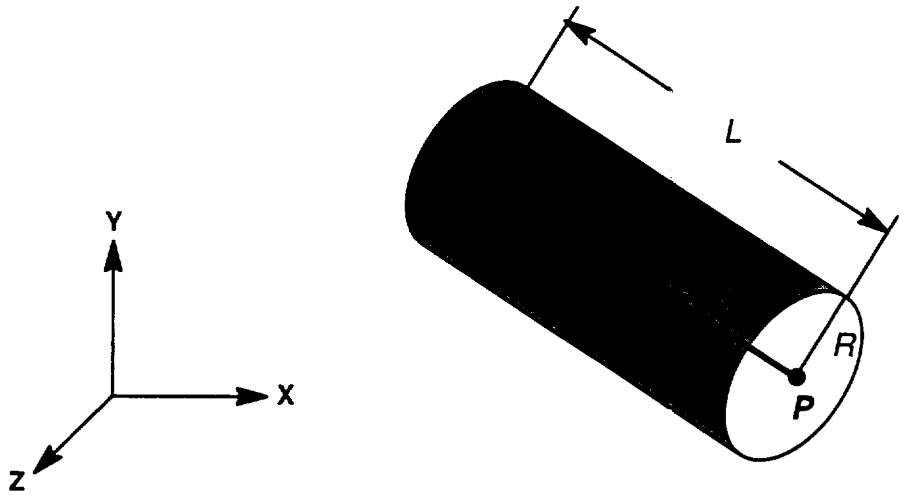


FIG. 6.7—*Example of cylindrical contact surface*

A typical set of commands to define a cylindrical contact surface is:

```
contact surface
   surface 1 cylinder
       point 5 2 3
       direction -3 1 -1
       radius 1.5
       length 4
       stiffness 1.0e8
       velocity 0.1 0.1 0.1
```

Spherical Surface

The spherical contact surface is a full sphere in space, and requires only the center point and the radius as input. The commands for the sphere are:

point < x: number > < y: number > < z: number >

radius < R: number >

where R is a number greater than zero. There are no defaults for these values.

A typical set of commands to define a spherical contact surfaces is:

```
contact surface
   surface 1 sphere
       point 5 2 3
       radius 1.5
       stiffness 1.0e8
       velocity 0.0 -0.1 0.0
```

6.3.4 Utility Options

The *clear* command provides an easy means to delete all contact surfaces. The syntax is:

clear

Additionally, the *dump* command prints the pertinent information about all of the current contact surfaces, including the current geometry and parameter values. The printed location of the base point for each contact surface is the current location after considering the surface velocity and elapsed analysis time. The syntax is:

dump

6.3.5 Notes on Multiple Contacting Surfaces

The implementation of frictionless, rigid body contact in WARP3D includes procedures to address cases involving multiple intersecting contact surfaces; Section 6.2.3 details the interaction algorithms. This allows the creation and appropriate handling of corners and other composite rigid surfaces. However, the algorithms impose some restrictions and requirements on contact surface definitions.

Overlapping of Contact Surfaces for Corners

The proper treatment of corners requires that the contact surfaces which form the corner overlap slightly. Consider Figure 6.8.a, where two rectangular contact surfaces form a corner, but do not overlap. For the contact processors to return the penetrating node to the corner, they must consider both planes. However, the node is actually only penetrating contact surface 1. After the the evaluation of contact, the node moves close to the surface of contact surface 1, but not far enough to move into contact surface 2. The node does not return to the corner in this case. With an overlap, as in Figure 6.8.b,the contact node violates both contact surfaces, thus the contact processors return the node to the corner. The amount of overlap depends on the stiffness of the contact planes; the overlap should be greater than the remaining penetration after enforcement of contact .

Avoid Acute Angles in Corners

The algorithms which manage intersecting contact surfaces operate best with corners formed at obtuse angles. Avoid internal or external corners with acute angles. See Figure 6.9 for examples of acceptable and unacceptable corners.

Potential Errors in Intersecting Rectangles with Cylinders and Spheres

The algorithms which resolve penetration of intersecting contact surfaces assume the contact surfaces are planes. Large penetrations into intersecting contact surfaces which include curved surfaces, particularly curved surfaces with small radii, may cause errors in the return location of the node. The use of small load steps, and avoidance of excess intersections between rectangles and curved surfaces, can help alleviate this problem.

6.3.6 Complete Examples

The following example is a valid contact definition which includes an example of each of the types of contact surfaces.

```
contact surface
   clear
   surface 1 plane
```

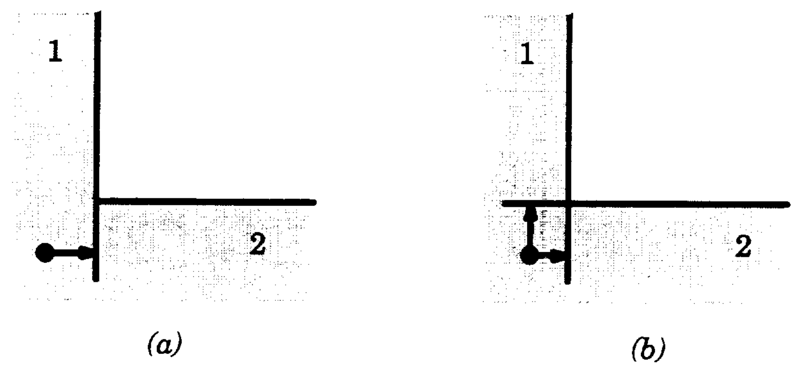


FIG. 6.8—*Overlapping of corners; a) with no overlap, node only returns to surface 1; b) overlap allows return of node to corner.*

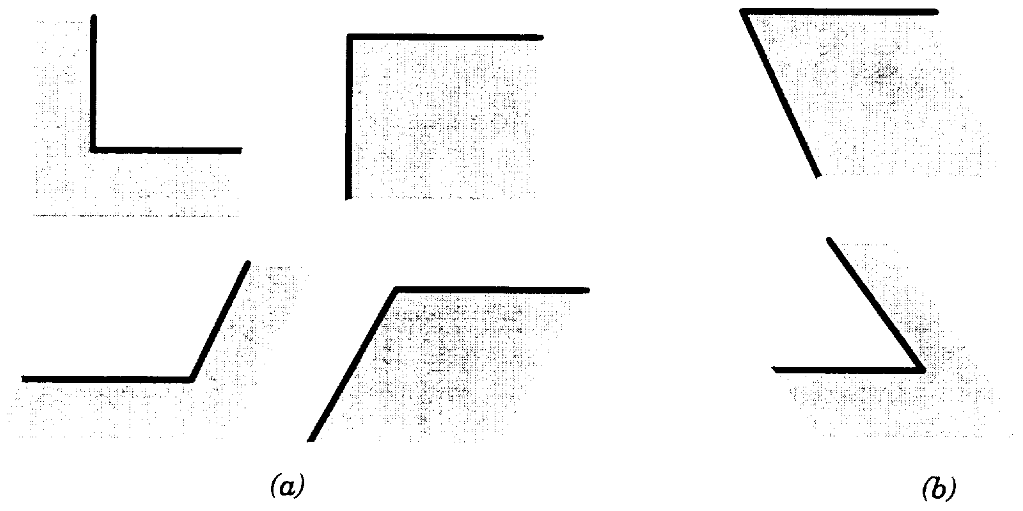


FIG. 6.9—*Acceptable and unacceptable corner definitions; a) corners have obtuse angles, and are permissible; b) corners have acute angles, which may cause problems.*

```
      point 3 5 2
      point 5 4 3
      point 4 6 1
      depth 4
      stiffness 1.0e8
      velocity 0.1 0.1 0.1
   surface 2 cylinder
      point 5 2 3
      direction -3 1 -1
      radius 1.5
      length 4
      stiffness 1.0e8
      velocity 0.05 0.0 0.05
   surface 3 sphere
      point 5 2 3
      radius 1.5
      stiffness 1.0e8
      velocity 0.0 -0.1 0.0
   dump
```

6.4 Tips for Analyses Using Contact

Contact analyses can be very troublesome; convergence is by no means guaranteed, and may be difficult to achieve. Also, improperly defined contact surfaces cause severe problems and may prove formidable to find. However, a variety of solution strategies and modelling techniques can significantly improve the likelihood of a successful analysis. This section presents some tips and suggestions for improving the performance of contact analyses in WARP3D.

Choosing a penalty parameter

Choosing an appropriate penalty parameter (contact stiffness) is one of the most important factors in the success of a contact analysis. A contact stiffness which is too small allows too great a penetration, while a parameter which is too large causes significant convergence problems (particularly with the preconditioned conjugate gradient solver), and degrades the accuracy of the solution. Also, the choice of a penalty parameter depends on the material model, the element type, mesh size, the type of loading, and even the contact surface. A few guidelines:

- The contact stiffness should be several orders of magnitude greater than the local "stiffness" of the structure at the point of contact. To evaluate the structural stiffness at a node, run an analysis with a unit force at the node normal to the potential contact surface. The local stiffness of the structure at the node is simply the force divided by the resulting displacement. A contact stiffness many magnitudes more than this value may cause ill-conditioning and large errors in the results.
- To assess the adequacy of an appropriate penalty parameter in a new analysis, try a small value first, then re-run the analysis several times, each time increasing the penalty parameter. A good choice is a penalty parameter which maintains strong convergence properties, but which would cause convergence problems if increased somewhat. Once a successful parameter is found, analyses with similar properties (loading, material, etc.) can use a similar value.
- Experience indicates that curved contact surfaces, such as cylinders and spheres, need a lower contact stiffness than rectangular contact surfaces.

Convergence of the first step

Achieving convergence of contact analyses on the first step in which contact takes place can be challenging. Oscillations may occur where contact springs force penetrating nodes completely out of contact. Without any contact to restrain them, the nodes penetrate on the next Newton iteration, repeating the cycle and impeding convergence. To avoid this problem:

- Reduce the size of the first step in which contact occurs; this is particularly necessary for moving contact surfaces. An effective step size for the first step with contact may need to be one-thousandth of the step size used for the remainder of the analysis. After the first step converges and contact initiates successfully, the step size can increase significantly. A reduced loading for a step or two may be necessary each time the contact surface experiences a significant shift in direction. Re-definition of the time increment per load step provides an effective means of reducing the movement rate for a contact surface; see Section 2.9 for information on the *time step* parameter. Make sure that any explicitly defined, non-zero constraints also reflect the change in movement rate.
- Decrease the contact surface stiffness by several orders of magnitude for the first load step with contact. Then redefine the contact surface with a much higher stiffness after a step or two.

- If the analysis contains a contact surface with a non-zero velocity, insure that the time increment per load step is reasonable. Many convergence problems arise from inaccurately specified time increments. See Section 2.9 for information on the *time step* parameter.

Improving general convergence

A number of techniques can improve general convergence of contact analyses. A few are listed below:

- Take smaller computational load steps.
- Use a smaller contact stiffness.
- Analyses which contain symmetry planes are significantly more robust than similar analyses which model the entire structure (and thus have more rigid body motion).
- Make sure that all rigid body motions are prevented using explicit constraints. Contact enforcement using the penalty method is equivalent to supplying force boundary conditions, which are typically less robust than explicit constraint boundary conditions.
- If elimination of rigid body motions is not possible, try including mass in the analysis with small time steps. The addition of inertia can help stabilize the analysis.
- If the analysis involves moving contact surfaces, then application of explicit constraints on nodes which move along with the contact surface can improve convergence significantly. See the pin-loaded C(T) specimen in the examples section for an example.
- Incorrect specification of contact surfaces can cause hidden difficulties; see Section 6.3 for more details on contact input and possible problems.
- Turn adaptive load reduction off (*adaptive off* in the *nonlinear solution parameters* command).

6.5 Example Analyses Using Contact

To illustrate contact in WARP3D, the following three examples present representative analyses which use contact surfaces. The examples include the rolling of a metal bar, crushing of a pipe, and crack closure in a pin-loaded C(T) specimen.

6.5.1 Rolling of a Metal Bar

This problem simulates the crushing of a metal bar using a rigid roller. The roller, with a 3" radius, comes into contact with the 2" × 2" × 10" bar 2 inches from the end, moves downward 0.5 inches, then moves along the bar at a constant height until it passes through the other end. Figure 6.10 shows the mesh and boundary conditions for the problem, as well as the path of the roller. The mesh contains 320 8-noded brick elements and 525 nodes, and uses the *mises* material model, with $E = 30000$ ksi, $\nu = 0.3$, $\sigma_0 = 60.0$ ksi, and $n = 10$. The solution uses large deformation theory, and a total of 200 load steps, with 50 load steps for the initial crushing and 150 load steps for the rolling. The penalty stiffness of the rigid cylinder is 10^6.

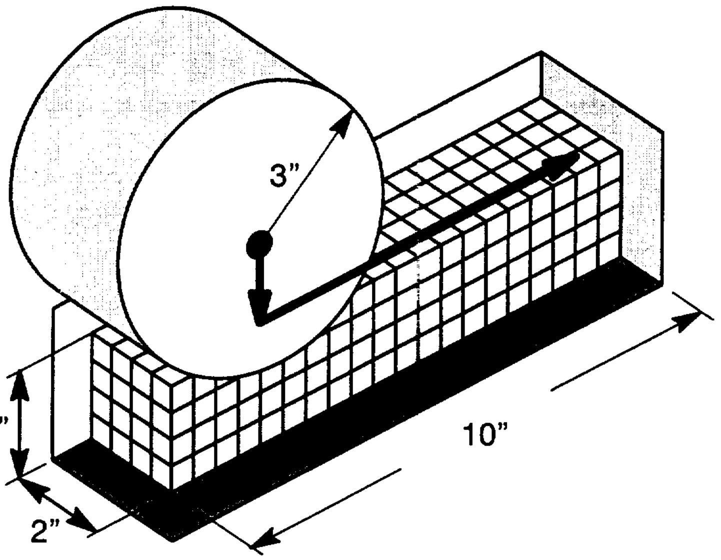
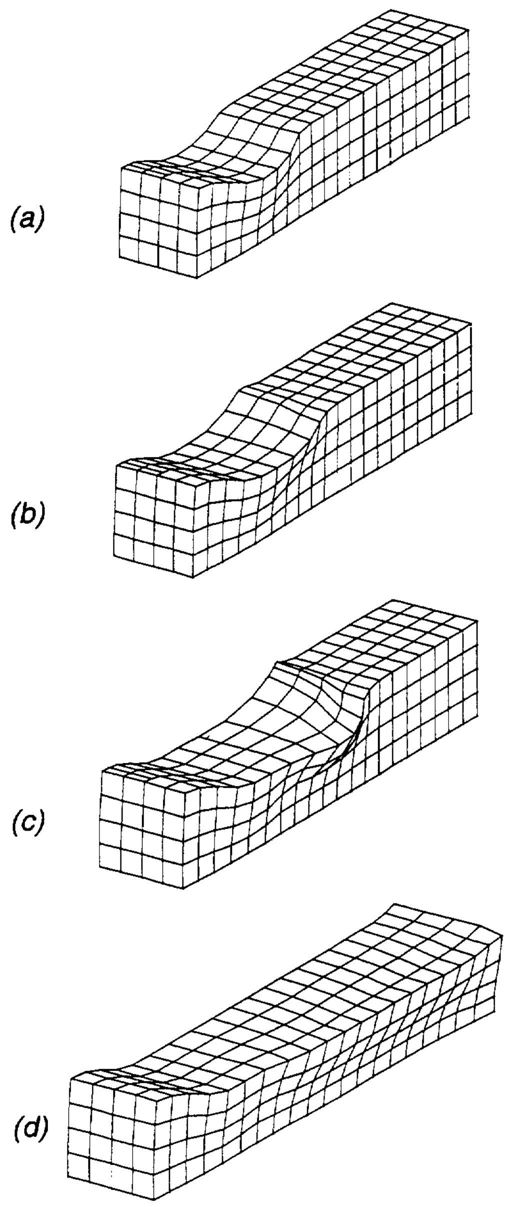


FIG. 6.10—*Mesh for rolling of a metal bar. Shaded planes indicate planes of nodes which are constrained. Arrows indicate path of cylinder during analysis.*

In order to start the contact smoothly and avoid initial convergence problems, the roller moves down 10^{-5} inches per step over the first two steps, then 10^{-2} inches per step until step 50. After 50 steps, the total downward displacement of the roller is 0.48002 inches. The roller stops for a load step, then moves across the bar, starting at a rate of 10^{-3} inches per step for steps 52 and 53, and at 0.05 inches per step for steps 54 to 200. Figure 6.11 shows deformation plots at several points during the analysis, and portions of the analysis input. Three load steps require adaptive step reduction to achieve convergence of the Newtons iterations.

6.5.2 Crushing of a Pipe

This example is a large deformation analysis of a pipe crushed inside a rigid box. The plane strain model uses four contact planes; a horizontal contact plane descends from above the



```
c
c ========== cylinder moves downwards =======
c
contact planes
 surface 1 cylinder
    point 2 5.000 -10
    direction 0 0 1
    length 20
    radius 3.0
    stiffness 10.0e5
    rate 0.0 -.01 0.0
nonlinear analysis parameters
    time step .001
    extrapolate off
compute displacements for loading test step 1-2
nonlinear analysis parameters
    time step 1.0
compute displacements for loading test step 3-50
c
c  ========== cylinder stops=======
c
contact planes
 clear
 surface 1 cylinder
    point 2 4.51998 -10
    direction 0 0 1
    length 20
    radius 3.0
    stiffness 10.0e5
    rate 0.0 0.0 0.0
compute displacements for loading test step 51
c
c ============ cylinder moves horizontally
c
contact planes
 clear
 surface 1 cylinder
    point 2 4.51998-10
    direction 0 0 1
    length 20
    radius 3.0
    stiffness 10.0e5
    rate 0.01 0.0 0.0
nonlinear analysis parameters
    time step .1
compute displacements for loading test step 52 - 53
nonlinear analysis parameters
    time step 5.0
compute displacements for loading test step 54 - 200
```

(e)

FIG. 6.11—*Deformed shapes from rolling of a metal bar. Deformed shapes shown at (a) step 50, (b) step 70, (c) step 150, and (d) step 200. Part (e) shows a portion of the input file for this analysis.*

pipe, while the side and bottom planes remain stationary. As the planes crush the pipe, the initially circular pipe transforms to a rectangular shape. The outside radius of the pipe is 5 inches, and the wall is 1 inch thick. The model, constructed with 186 8-noded elements and 496 nodes, uses a *mises* material model, with $E = 30000$ ksi, $\nu = 0.3$, $\sigma_0 = 60.0$ ksi, and $n = 10$. The top plane translates down 10^{-6} inches the first step, then 2.5×10^{-3} inches per step for the remainder of the analysis, providing a 5 inch height reduction after 2000 load steps. Fixed constraints on the topmost nodes travelling with the moving contact plane help

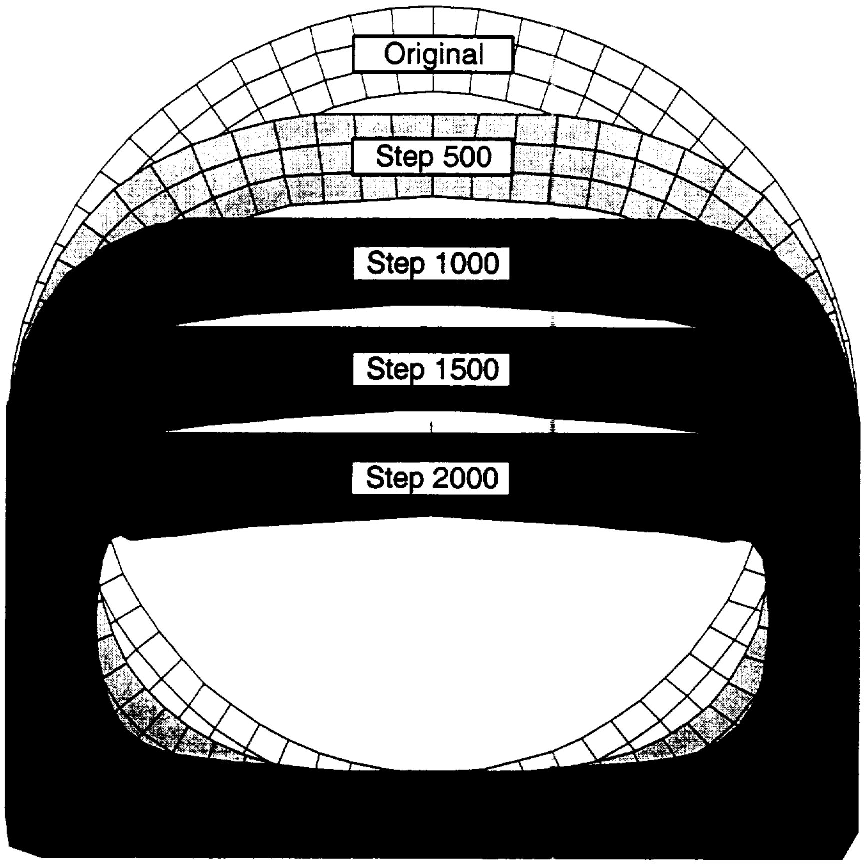


FIG. 6.12—*Deformed shapes from crushing of a pipe. Deformed shapes shown at 500 step increments between 0 and 2000 load steps.*

stabilize the model. Figure 6.12 displays a series of deformed shapes at various stages of the analysis.

6.5.3 Crack Closure in a Pin-Loaded C(T) Specimen

Finite element analyses of fracture specimens typically do not include the actual boundary conditions as applied in experiments. For instance, experimental procedures dictate that loading of a compact tension, C(T), specimen use pins, while finite element analyses of the specimen usually load a single line of nodes or fill in the pin hole with linear elastic elements. While in general the discrepancies between experimental and modelled boundary conditions cause only minor differences in the crack front results, there are cases in which the true boundary conditions can cause significant differences.

To demonstrate the use of contact surfaces provided in WARP3D in the analysis of fracture specimens, this section describes the large deformation, plane strain analysis of a pin-loaded C(T) specimen, where a contact cylinder explicitly models the pin loading. Furthermore, the analysis includes both cyclic tensile and compressive loading, causing crack closure. To model crack closure, a rectangular contact surface congruent with the symmetry plane prevents penetration of the crack face. The model has a width W of 1.9685 inches, and a crack length to width (a/W) ratio of 0.6. An initially blunt notch with a radius of 7.6×10^{-4} inches allows significant blunting at the crack tip as deformation increases. The pin has a radius of 0.17 inches, while the radius of the hole in the C(T) specimen is 0.18 inches. The model contains 435 8-noded elements and 1000 nodes, and uses a *mises* material model with $E = 30000$ ksi, $\nu = 0.3$, $\sigma_0 = 60.0$ ksi, and $n = 10$. Figure 6.13 illustrates the mesh, and Figure 6.14 shows the deformation and stress contours of the model at several points during the analysis.

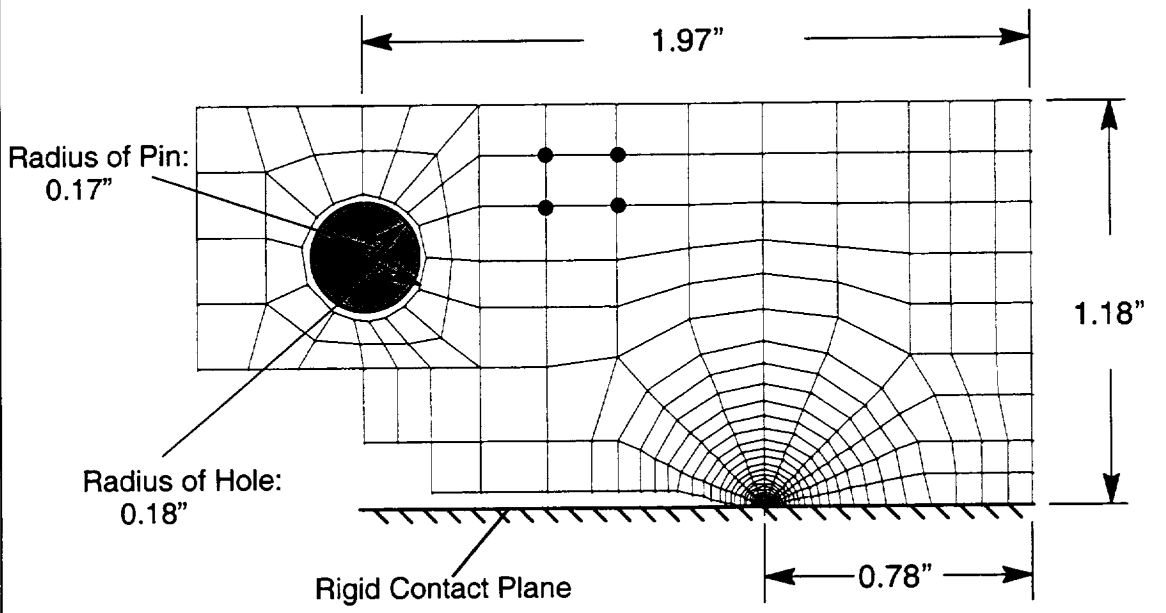


FIG. 6.13—*Mesh for pin-loaded C(T) analysis.*

To initiate contact, the pin moves only 5.0×10^{-8} inches upwards during the first step. On step 2, the rate increases to 5.0×10^{-4} inches per load step. This continues until the specimen experiences a total upward displacement of 0.0138 inches at step 30, after which the pin moves downwards. The downward movement begins at 5.0×10^{-8} inches for step 31, then increases to 1.0×10^{-4} inches per step. To assist computational stability during the initial loading and unloading of the specimen, explicit constraints on the nodes at the top of the loading hole ensure model displacement commensurate with the pin displacement. This is crucial in the unloading section between steps 31 and 61, where an analysis which does not include the explicit constraints requires a loading rate several orders of magnitude smaller. After step 61, the analysis moves into compressive loading, at which point the explicit constraints are removed. Once again, to ensure smooth convergence with the initial contact, the first step uses a displacement of 5.0×10^{-8} inches, while movement of 5.0×10^{-4} inches per step ensues afterwards. The loading continues until step 130, achieving a total downwards displacement of 0.032 inches. Crack closure initiates on step 75, and by step 130, the crack closes almost completely except for a small region near the crack tip.

A selected portion of the input file follows:

```
constraints
*input from file 'constraints'
   33 34     v 5.0e–8
nonlinear analysis parameters
   time step 0.0001
   extrapolate off
c
```

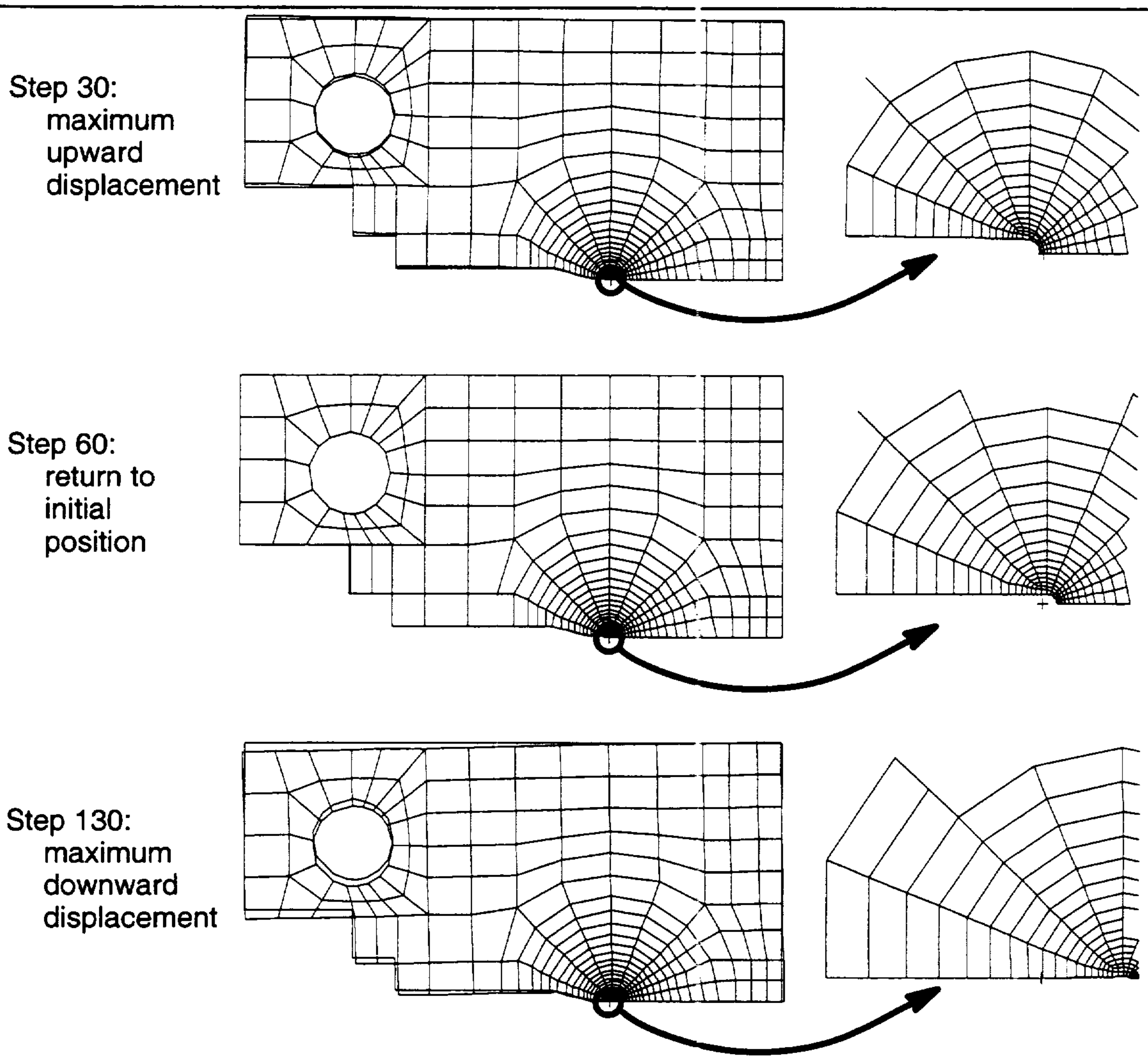


FIG. 6.14—*Deformed shapes for pin-loaded C(T) analysis. Stress contours range from 0 to 120 ksi in 20 ksi increments.*

```
contact surface
   surface 1 cylinder
      point –1.181 .75322844 –1
      direction 0 0 1
      length 2
      radius .17
      stiffness 10.0e5
      rate 0.0 0.0005 0.0
c
compute displacements for loading test step 1
c
nonlinear analysis parameters
   time step 1.0
constraints
*input from file 'constraints'
   33 34      v .0005
```

```
c
 compute displacements for loading test step 2–30
c
constraints
*input from file 'constraints'
   33 34      v –5.0e–8
contact surface
   surface 1 cylinder
      point –1.181 .76772849 –1
      direction 0 0 1
      length 2
      radius .17
      stiffness 10.0e5
      rate 0.0 –0.0005 0.0
nonlinear analysis parameters
   time step 0.0001
c
 compute displacements for loading test step 31
c
constraints
*input from file 'constraints'
   33 34      v –.0005
nonlinear analysis parameters
   time step 1.0
c
 compute displacements for loading test step 32–60
c
constraints
*input from file 'constraints'
contact surface
   surface 1 cylinder
      point –1.181 .7245 –1
      direction 0 0 1
      length 2
      radius .17
      stiffness 10.0e5
      rate 0.0 –0.0005 0.0
   surface 2 plane
      point –10 0 –10
      point –10 0  10
      point  10 0 –10
      stiffness 10.0e5
 nonlinear analysis parameters
   time step 0.0001
 c
 compute displacements for loading test step 61
 c
 nonlinear analysis parameters
   time step 1.0
 c
 compute displacements for loading test step 62–130
```

Patran Results File Formats

Figures in this Appendix provide skeletal Fortran programs to read Patran nodal results files. They provide a starting point for development of more advanced programs.

Fig. A.1	read a binary file of nodal strain/stress results
Fig. A.2	read a binary file of nodal displacements, velocities, accelerations, internal forces
Fig. A.3	read an Ascii file of nodal strain/stress results
Fig. A.4	read an Ascii file of nodal displacements, velocities, accelerations, internal forces

```
c *********************************************************************
c *                                                                   *
c *   read stress/strain binary patran file                           *
c *                                                                   *
c *********************************************************************
c
c
      implicit integer (a-z)
      parameter( maxnod=20000, maxcols=50 )
      double precision nodval(maxcols,maxnod)
      real rtemp, pvals(maxcols)
      dimension title(80)
      character * 80 binnam
c
      termin = 5
      termot = 6
      binfil = 10
c
      write(termot,*) ' '
      write(termot,*) ' '
      write(termot,*) '>> binary strain/stress processing program'
      write(termot,*) ' '
      write(termot,9400) ' > name of results file? '
      read(termin,9500) binnam
      open(unit=binfil,file=binnam,status='old',recl=3000,form='unformatted')
      write(termot,*) ' > file opened ok'
c
c             read the binary results file of nodal strains/stresses.
c             patran results are single precision. read as single and
c             store as double.
c
      read(binfil) title,nnode,ii,rtemp,ii,nvals
      read(binfil) title
      read(binfil) title
      write(termot,*) '>> number of nodes: ',nnode
      do node = 1, nnode
         if ( mod(node,200) .eq. 0 ) then
            write(termot,*) '    > processing node: ',node
         end if
         read(binfil) ii, (pvals(jj),jj=1,nvals)
         nodval(1:nvals,node) = pvals(1:nvals)
      end do
      close(unit=binfil)
c
c            call a routine to do something with the nodal values
c            of stress/strain.
c
      call process( nodval, nnode, maxcols, termot )
c
      write(termot,*) '>> processing completed'
      write(termot,*) '>> normal termination'
c
      call exit
c
 9400 format(a,$)
 9500 format(a80)
      end
c
      subroutine process ( values, nnode, nrow, termot )
      implicit integer (a-z)
      double precision values(nrow,nnode)
      return
      end
```

FIG. A.1—*Fortran program to read Patran binary file of nodal strain or stress results.*

```
c *****************************************************************************
c *                                                                           *
c *   read displ, vel, accel, inter. forces binary patran file               *
c *                                                                           *
c *****************************************************************************
c
c
      implicit integer (a-z)
      parameter( maxnod=20000 )
      double precision x(maxnod), y(maxnod), z(maxnod)
      real xval, yval, zval
      dimension title(80)
      character * 80 binnam
c
      termin = 5
      termot = 6
      binfil = 10
c
      write(termot,*) ' '
      write(termot,*) ' '
      write(termot,*) '>> binary node value pocessing program'
      write(termot,*) ' '
      write(termot,9400) ' > name of results file? '
      read(termin,9500) binnam
      open(unit=binfil,file=binnam,status='old',recl=3000,form='unformatted')
      write(termot,*) ' > file opened ok'
c
c             read the binary results file of nodal values.
c             read x, y, z components. patran results are single
c             precision. read as single and store as double.
c
      read(binfil) title,nnode,ii,rtemp,ii,nvals
      read(binfil) title
      read(binfil) title
      write(termot,*) '>> number of nodes: ',nnode
      do node = 1, nnode
         read(binfil) ii, xval, yval, zval
         x(node) = xval
         y(node) = yval
         z(node) = zval
      end do
      close(unit=binfil)
c
c            call a routine to do something with the nodal values
c
      call process( x, y, z, nnode, termot )
c
      write(termot,*) '>> processing completed'
      write(termot,*) '>> normal termination'
c
      call exit
c
 9400 format(a,$)
 9500 format(a80)
      end
c
      subroutine process ( x, y, z, nnode, termot )
      implicit integer (a-z)
      double precision x(*), y(*), z(*)
      return
      end
```

FIG. A.2—*Fortran program to read Patran binary file of nodal displacements, velocities, accelerations or internal forces.*

```
c ***********************************************************************
c *                                                                     *
c *   read ascii stress/strain patran file                              *
c *                                                                     *
c ***********************************************************************
c
c
      implicit integer (a-z)
      parameter( maxnod=20000, maxcols=50 )
      double precision nodval(maxcols,maxnod), val3
      character * 80 asciinam, line
c
c
      termin = 5
      termot = 6
      asciifil = 10
c
      write(termot,*) ' '
      write(termot,*) ' '
      write(termot,*) '>> ascii strain/stress processing program'
      write(termot,*) ' '
      write(termot,9400) ' > name of results file? '
      read(termin,9500) asciinam
      open(unit=asciifil,file=asciinam,status='old')
      write(termot,*) ' > file opened ok'
c
c             skip past the header lines of neutral file.
c             get number of nodes and number of result values
c             for each node.
c
      read(asciifil,9500) line
      read(asciifil,9600) nnode, ival2, val3, ival4, nvals
      read(asciifil,8900) line
      read(asciifil,8900) line
c
c             read values for each node into a double array.
c
      write(termot,*) ' > reading nodal results file..'
      do node = 1, nnode
        read(asciifil,9000) ii, (nodval(jj,node),jj=1,nvals)
      end do
      close(unit=asciifil)
      write(termot,*) '>> nodal results file read'
c
c            call a routine to do something with the nodal values
c            of stress/strain.
c
      call process( nodval, nnode, maxcols, termot )
c
      write(termot,*) '>> processing completed'
      write(termot,*) '>> normal termination'
c
      call exit
c
 8900 format(a1)
 9000 format(i8,(5e13.7))
 9400 format(a,$)
 9500 format(a80)
 9600 format(2i5,e15.6,2i6)
      end
c
      subroutine process ( values, nnode, nrow, termot )
      implicit integer (a-z)
      double precision values(nrow,nnode)
      return
      end
```

FIG. A.3—*Fortran program to read Patran ASCII file of nodal strain or stress results.*

```
c *******************************************************************
c *                                                                 *
c *   read displ, vel, accel, inter. forces ascii patran file       *
c *                                                                 *
c *******************************************************************
c
c
      implicit integer (a-z)
      parameter( maxnod=20000 )
      double precision x(maxnod), y(maxnod), z(maxnod)
      character * 80 asciinam, line
c
c
      termin = 5
      termot = 6
      asciifil = 10
c
      write(termot,*) ' '
      write(termot,*) ' '
      write(termot,*) '>> ascii node value pocessing program'
      write(termot,*) ' '
      write(termot,9400) ' > name of results file? '
      read(termin,9500) asciinam
      open(unit=asciifil,file=asciinam,status='old')
      write(termot,*) ' > file opened ok'
c
c             skip past the header lines of neutral file.
c             get number of nodes and number of result values
c             for each node.
c
      read(asciifil,9500) line
      read(asciifil,9600) nnode, ival2, val3, ival4, nvals
      read(asciifil,8900) line
      read(asciifil,8900) line
c
c             read values for each node into a double array.
c
      write(termot,*) ' > reading nodal results file..'
      do node = 1, nnode
        read(asciifil,9000) ii, x(ii), y(ii), z(ii)
      end do
      close(unit=asciifil)
      write(termot,*) '>> nodal results file read'
c
c            call a routine to do something with the nodal values
c
      call process( x, y, z, nnode, termot )
c
      write(termot,*) '>> processing completed'
      write(termot,*) '>> normal termination'
c
      call exit
c
 8900 format(a1)
 9000 format(i8,(5e13.7))
 9400 format(a,$)
 9500 format(a80)
 9600 format(2i5,e15.6,2i6)
      end
c
c
      subroutine process ( x, y, z, nnode, termot )
      implicit integer (a-z)
      double precision x(*), y(*), z(*)
      return
      end
```

FIG. A.4—*Fortran program to read Patran ASCII file of nodal displacements, velocities, accelerations, or internal forces.*

Appendix B

References

[1] Amestoy, H.D., Bui,, and Labbens, "On the Definition of Local Path Independent Integrals in *3–D* Crack Problems," *Mechanics Research Communications,* Vol. 8, 1981, pp. 231–236.†

[2] Aravas, N., "On the Numerical Integration of a Class of Pressure–Dependent Plasticity Models," *International Journal for Numerical Methods in Engineering,* Vol. 24, 1987, pp. 1395–1416.†

[3] Asaro, R. J., "Crystal Plasticity," *Journal of Applied Mechanics,* Vol. 50, 1984, pp. 1–12.†

[4] Atluri, S. N. "On Constitutive Relations at Finite Strain: Hypo–Elasticity and Elasto–Plasticity with Isotropic and Kinematic Hardening" *Computer Methods in Applied Mechanics and Engineering,* Vol. 43, 1984, pp. 137–171.†

[5] Bakker, A., "The Three–Dimensional *J*–Integral: An Investigation into Its Use for Post–Yield Fracture Assessment," WTHD No. 167, Laboratory for Thermal Power Engineering, Delft University of Technology, Mekelweg 2, 2628 CD, Delft., 1984.§

[6] Bathe, K. J. *Finite Element Procedures in Engineering Analysis.* Prentice–Hall, Inc. Englewood–Cliffs, N.J., 1982.†

[7] Biffle J.H., "Indirect Solution of Static Problems Using Concurrent Vector Processing Computers," *Parallel Computations and their Impact on Mechanics*, ed. by A.K. Noor, American Society of Mechanical Engineers, New York, 1987, pp. 317–330.†

[8] Biffle, J. H., and M. L. Blanford, "JAC2D– A Two–Dimensional Finite Element Computer Program for the Nonlinear Quasi–Static Response of Solids with the Conjugate Gradient Method," *SAND93-1891, Sandia National Laboratories,* Albuquerque, NM., 1994.§

[9] Biffle, J. H., and M. L. Blanford, "JAC3D– A Three–Dimensional Finite Element Computer Program for the Nonlinear Quasi–Static Response of Solids with the Conjugate Gradient Method," *SAND87-1305, Sandia National Laboratories,* Albuquerque, NM., 1993.§

[10] Budiansky, B., and Rice, J., "Conservation Laws and Energy Release Rates," *Journal of Applied Mechanics*, Vol. 40, 1973, pp. 201–203.†

[11] Carey G.F., and Jiang B., "Element–By–Element Linear and Nonlinear Solution Schemes," *Communications in Applied Numerical Methods*, Vol. 2, No. 2, March–April 1986, pp. 145–153.†

[12] Carpenter, W.C., Read, D.T., and Dodds, R.H., "Comparison of Several Path Independent Integrals Including Plasticity Effects," *International Journal of Fracture,* Vol. 31, 1986, pp. 303–323.†

[13] Cherepanov, G.P., "The Propagation of Cracks in a Continuous Medium," *Journal of Applied Mathematics and Mechanics,* Vol. 31, 1967, pp 503–512.†

[14] Chu, C. C. and Needleman, A., "Void Nucleation Effects in Biaxially Stretched Sheets," *Journal of Engineering Materials and Technology,* Vol. 102, 1980, pp. 249–256.†

[15] Concus P., Golub G.H., and O'Leary D.P., "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equation," *Sparse Matrix Computations,*ed. J.R. Bunch and D.J.Rose, Academic Press, New York, 1965, pp. 307–322.†

[16] Cook R. D., Malkus, D. S., and Plesha, M. E., "Concepts and Applications of Finite Element Analysis," Third Ed., John Wiley & Sons, Inc., New York NY, 1989.†

[17] Crisfield, M.A., *Nonlinear Finite Element Analysis of Solids and Structures - Volume 1:Essentials*, John Wiley & Sons Ltd., 1991.†

[18] Cuitino, A. and Ortiz, M., "A Material–Independent Method for Extending Stress Update Algorithms from Small–Strain Plasticity to Finite Plasticity with Multiplicative Kinematics," *Engineering Computations,* Vol. 9, 1992, pp. 437–451.†

[19] de Lorenzi, H.G., "On the Energy Release Rate and the *J*–integral for *3–D," International Journal of Fracture,* Vol. 19, 1982, pp. 183–193.†

[20] Dienes, J. K., "On the Analysis of Rotation and Stress Rate in Deforming Bodies," *Acta Mechanica*, Vol. 32, 1979, pp. 217–232.†

[21] Doghri, I., Muller, A., and Taylor, R. L. "A General Three–Dimensional Contact Procedure for Implicit Finite Element Codes," Engineering Computations, Vol. 15 no. 2, 1998, pp.233–259.

[22] Dodds, R., "Numerical Techniques for Plasticity Computations in Finite Element Analysis," *Computers and Structures,* Vol. 26, No. 5, 1987, pp. 767–779.†

[23] Eshelby, J.D., "Energy Relations and the Energy Momentum Tensor in Continuum Mechanics," in *Inelastic Behavior of Solids*, M.F. Kanninen, et al. (eds), Mc Graw–Hill, NY, 1970.†

[24] Flanagan, D. P. and Taylor, L. M., "An Accurate Numerical Algorithm for Stress Integration with Finite Rotations," *Computer Methods in Applied Mechanics and Engineering,* Vol. 62, 1987, pp. 305–320.†

[25] Flanagan D.P., and Taylor L.M., "Structuring Data for Concurrent Vectorized Processing in a Transient Dynamics Finite Element Program," *Parallel Computations and their Impact on Mechanics*, ed. by A.K. Noor, American Society of Mechanical Engineers, New York, 1987, pp. 291–299.†

[26] Golub G.H., and Van Loan C.F., "Matrix Computations," The Johns Hopkins University Press, Baltimore Maryland, 1983.§

[27] Goudreau, G. L. and Hallquist, J. O., "Recent Developments in Large–Scale Lagrangian Hydrocode Technology," *Computer Methods in Applied Mechanics and Engineering,* Vol. 33, 1982, pp. 725–757.†

[28] Green, A. E. and Naghdi, P. M., "A General Theory of an Elastic–plastic Continuum. *Archives of Rational Mechanics Analysis,* Vol. 18, 1965, pp. 251–281.†

[29] Gurson, A. L., "Continuum Theory of Ductile Rupture by Void Nucleation and Growth: Part I– Yield Criteria and Flow Rules for Porous Ductile Media," *Journal of Engineering Materials and Technology,* Vol. 99, 1977, pp. 2–15.†

[30] Hallquist, J. O., "NIKE 2–D – A Vectorized, Implicit, Finite Deformation, Finite–Element Code for Analyzing the Static and Dynamic Response of 2–D Solids," *Lawrence Livermore Laboratory Report UCRL–52678*, 1984.§

[31] Hallquist, J. O., "NIKE 3–D – A Vectorized, Implicit, Finite Deformation, Finite–Element Code for Analyzing the Static and Dynamic Response of 3–D Solids." *Lawrence Livermore Laboratory Report UCID–18822*, 1984.§

[32] Hancock, J., and Cowling, M., "Role of Sate of Stress in Crack-Tip Failure Processes," *Metal Science,* Aug.-Sept., 1980, pp. 292–304.†

[33] Healy, B., Pecknold, D. A. and Dodds, R., "Applications of Parallel and Vector Algorithms in Nonlinear Structural Dynamics Using the Finite Element Method," *Civil Engineering Studies,* SRS No. 571, UILU–ENG–92–2011, University of Illinois, Urbana, Illinois, 1992.§

[34] Hellen, T.K., "On the Method of Virtual Crack Extension," *International Journal for Numerical Methods in Engineering,* Vol. 9, 1975, pp. 187–207.†

[35] Hibbitt, Karlsson & Sorensen, Inc., *ABAQUS* Theory Manual, Version 5.3, Providence R.I., 1993.§

[36] Hibbitt, Karlsson & Sorensen, Inc., *ABAQUS–Explicit* Theory Manual, Version 5.2, Providence R.I., 1992.§

[37] Hinton, E, Rock, T., and Zienkiewicz, O. C., "A Note on Mass Lumping and Related Processes in the Finite Element Method," *Earthquake Engineering and Structural Dynamics,* Vol. 4, No. 3, 1976, pp 245–249.†

[38] Hoger, A. and Carlson, D. E., "Determination of the Stretch and Rotation in the Polar Decomposition of the Deformation Gradient," *Quarterly of Applied mathematics,* Vol. 42, 1984, pp. 113–117.†

[39] Hughes, T.J.R., "Stability, Convergence, and Growth and Decay of Energy of the Average Acceleration Method in Nonlinear Structural Dynamics," *Computers and Structures,* Vol. 6, 1976, pp. 313–324.†

[40] Hughes, T. J. "Generalization of Selective Integration Procedures to Anisotropic and Nonlinear Media," *International Journal for Numerical Methods in Engineering*, Vol. 15, 1980, pp. 1413–1418.†

[41] Hughes, T. J. and Winget, J., "Finite Rotation Effects in Numerical Integration of Rate Constitutive Equations Arising in Large–Deformation Analysis," *International Journal for Numerical Methods in Engineering,* Vol. 15, 1980, pp. 1862–1867.†

[42] Hughes, T. J., Levit, I., and Winget, J. M., "An Element–By–Element Solution Algorithm for Problems of Structural and Solid Mechanics," *Computer Methods in Applied Mechanics and Engineering,* Vol. 36, 1983, pp. 241–254.†

[43] Hughes T.J.R., Ferencz R.M., and Hallquist J.O., "Large–scale Vectorized Implicit Calculations in Solid, Mechanics on a Cray X–MP/48 Utilizing EBE Preconditioned Conjugate Gradients," *Computer Methods in Applied Mechanics and Engineering,* Vol. 61, 1987, pp. 215–248.†

[44] Hughes T.J.R., *The Finite Element Method.* Prentice–Hall, Englewood–Cliffs, New Jersey, 1987.†

[45] Hutchinson, J., "Singular Behavior at the End of a Tensile Crack in a Hardening Material," *Journal of the Mechanics and Physics of Solids,* Vol. 16, 1968, pp. 13–31.†

[46] Jaumann, G. "Geschlossenes System Physikalischer Und Chemisher Differentialgesefze," *Sitz Zer. Akad. Wiss. Wein,* (IIa) 120, 1911, pp. 385.†

[47] Johnson, G. C. and Bammann, D. J., "A Discussion of Stress Rates in Finite Deformation Problems," *International Journal for Solids and Structures*, Vol. 20, 1984, pp. 725–737.†

[48] Keppel, M. and Dodds, R. H., "Improved Numerical Techniques for Plasticity Computations in Finite–Element Analysis," *Computers and Structures*, Vol. 36, No. 1, 1990, pp. 183–185.†

[49] Key, S. W. and Krieg, R. D., "On the Numerical Implementation of Inelastic Time Dependent and Time Independent, Finite Strain Constitutive Equations in Structural Mechanics," *Computer Methods in Applied Mechanics and Engineering,* Vol. 33, 1982, pp. 439–452.†

[50] Kishimoto, K., Aoki, S., and Sakata, M., "On the Path Independent $\hat{J}$–Integral," *Engineering Fracture Mechanics,* Vol. 13, 1980, pp. 841–850.†

[51] Kojic, M. and Bathe, K. J., "Studies of Finite–element Procedures: Stress Solution of a Closed Elastic Strain Path with Stretching and Shearing Using the Updated Lagrangian Jaumann Formulation," *Computers and Structures*, Vol. 26, 1987, pp. 175–179.†

[52] Lee, E. H., "Elastic–Plastic Deformation at Finite Strain," *Journal of Applied Mechanics*, Vol. 36, 1969, pp. 1–6.†

[53] Knowles, J., and Sternberg, E., "On a Class of Conservation Integrals," *Archives of Rational Mechanics Analysis,* Vol. 44, 1972, pp. 187–211.†

[54] Krieg, R. D. and Key, S. W., "Implementation of a Time Independent Plasticity Theory into Structural Computer Programs," In *Constitutive Equations in Viscoplasticity: Computational and Engineering Aspects, AMD–20* (Edited by J. A. Stricklin and K. J. Saczalski), ASME, New York, 1976, pp. 125–138.†

[55] Li, F.Z., Shih, C.F., and Needleman, A., "A Comparison of Methods for Calculating Energy Release Rates," *Engineering Fracture Mechanics,* Vol. 21, (1985), 405–421.†

[56] Mackenzie, A., Hancock, J., and Brown, D., "On the Influence of State of Stress on Ductile Fracture Initiation in High Strength Steels," *Engineering Fracture Mechanics,* Vol. 9, 1977, pp. 167–188.†

[57] Malvern, L., *An Introduction to the Mechanics of Continuous Media.* Prentice–Hall, Englewood–Cliffs, New Jersey, 1969.†

[58] Marsden, J. E.. and Hughes, T. J. R., *Mathematical Foundations of Elasticity.* Prentice–Hall, Englewood–Cliffs, New Jersey, 1983.†

[59] McCalpin, J. D., "Memory Bandwidth and Machine Balance in Current High Performance Computers," Invited for submission tc IEEE Technical Committee on Computer Architecture Newsletter. To appear December 1995.

[60] Moran, B., and Shih, C.F., "A General Treatment of Crack Tip Contour Integrals," *International Journal of Fracture,* Vol. 35, 1987, pp. 295–310.†

[61] Moran, B., and Shih, C.F., "Crack Tip and Associated Domain Integrals from Momentum and Energy Balance," *Engineering Fracture Mechanics,* Vol. 27, 1987, pp. 615–642.†

[62] Moran, B., Ortiz, M., and Shih, C. F., "Formulation of Implicit Finite Element Methods for Multiplicative Finite Deformation Plasticity," *International Journal for Numerical Methods in Engineering,* Vol. 29, 1990, pp. 483–514.†

[63] Nagtegaal, J. C. Parks, D. M., and Rice, J. R., "On Numerically Accurate Finite Element Solutions in the Fully Plastic Range," *Computer Methods in Applied Mechanics and Engineering,* Vol. 4, 1974, pp. 153–178.†

[64] Nagtegaal, J. C. and de Jong, J. E., "Some Computational Aspects of Elastic–Plastic, Large Strain Analysis," *International Journal for Numerical Methods in Engineering,* Vol. 12, 1981, pp. 15–41.†

[65] Nagtegaal, J. C., "On the Implementation of Inelastic Constitutive Equations with Special Reference to Large Deformations," *Computer Methods in Applied Mechanics and Engineering,* Vol. 33, 1982, pp. 221–245.†

[66] Nagtegaal, J. C. and Veldpaus, F. E., "On the Implementation of Finite Strain Plasticity Equations in a Numerical Model," In *Numerical Analysis of Forming Processes* (edited by J.F. Pittman, O. C. Ziekiewicz, R. D. Wood and J. M. Alexander), p. 351. John Wiley and Sons, New York, 1984.†

[67] Nakamura, T., Shih, C., and Freund, L., "Analysis of Dynamicaly Loaded SE(B) Ductile Fracture Specimen," *Engineering Fracture Mechanics,* Vol. 25, 1986, pp. 323–339.†

[68] Needleman, A. and Tvergaard, V., "An Analysis of Ductile Rupture Modes at a Crack Tip," *Journal of Mechanics and Physics of Solids,* Vol. 35, 1987, pp. 151–183.†

[69] Newmark N.M., "A Method of Computation for Structural Dynamics," *Journal of the Engineering Mechanics Division*, ASCE, Vol. 32, No. EM3, 1959, pp. 67–94.†

[70] Nikishkov, G.P. and Atluri, S.N., "Calculation of Fracture Mechanics Parameters for an Arbitrary Three–Dimensional Crack, by the 'Equivalent Domain Integral' Method," *International Journal for Numerical Methods in Engineering,* Vol. 24, 1987, pp. 1801–1827.†

[71] Panontin, T., and Sheppard, S., "The Relationship Between Constraint and Ductile Fracture Initiation as Defined by Micromechanical Analyses," *Fracture Mechanics: 26th Volume, ASTM STP1256,* W. Reuter, J. Underwood, J. Newman, Eds., American Society for Testing and Materials, Philadelphia, 1995·

[72] Parks, D.M., "The Virtual Crack Extension Method for Nonlinear Material Behavior," *Computer Methods in Applied Mechanics and Engineering,* Vol. 12, 1977, pp. 353–364.†

[73] Pinsky, P. M., Ortiz, M. and Pister, K. S. "Numerical Integration of Rate Constitutive Equations in Finite Deformation Analysis," *Computer Methods in Applied Mechanics and Engineering*, Vol. 40, 1983, pp. 137–158.†

[74] Rice, J., and Rosengren, G.F., "Plane Strain Deformation Near a Crack Tip in a Power Law Hardening Material," *Journal of Mechanics and Physics of Solids,* Vol. 16, 1968, pp. 1–12.†

[75] Rice, J. "A Path Independent Integral and the Approximate Analysis of Strain Concentration by Notches and Cracks," *Journal of Applied Mechanics,* Vol. 35, 1968, pp. 379–386. †

[76] Rice, J., and Tracey, D., 'On the Ductile Enlargement of Voids in Triaxial Stress Fields," *Journal of Mechanics and Physics of Solids,* Vol. 17, 1969, pp. 201–217.†

[77] Roy, S., Fossum, A. F., and Dexter, R. J., "On the Use of Polar Decomposition in the Integration of Hypo–Elastic Constitutive Laws," *International Journal of Engineering Science,* Vol. 30, 1992, pp. 119–133.†

[78] Rubinstein, R. and Atluri, S. N., "Objectivity of Incremental Constitutive Equations Over Finite Time Steps in Computational Finite Deformation Analysis," *Computer Methods in Applied Mechanics and Engineering,* Vol. 36, 1983, pp. 277–290.†

[79] Schoeberle, D.F., and Belytschko, T., "On the Unconditional Stability of an Implicit Algorithm for Nonlinear Structural Dynamics," *Journal of Applied Mechanics,* Vol. 42, 1975, pp. 865–869.†

[80] Schreyer H.L., Kulak R.F., and Kramer J.M., "Accurate Numerical Solutions for Elastic–Plastic Models," *Journal of Pressure Vessel Technology,* Vol. 101, 1979, pp. 226–234.†

[81] Shih, C.F., Moran, B., and Nakamura, T. "Energy Release Rate Along a Three–Dimensional Crack Front in a Thermally Stressed Body," *International Journal of Fracture,* Vol. 30, 1986, pp. 79–102.†

[82] Simo, J.C. and Taylor, R.L., "Consistent Tangent Operators for Rate Independent Elastoplasticity," *Computer Methods in Applied Mechanics and Engineering,* Vol. 35, 1985, pp. 101–118.†

[83] Simo, J. C. and Ortiz, M., "A Unified Approach to Finite Deformation Elasto–Plastic Analysis Based on the Use of Hyper–elastic Constitutive Equations," *Computer Methods in Applied Mechanics and Engineering,* Vol. 49, 1985, pp. 221–245.†

[84] Simo, J. C. and Hughes, T. J. R. *Elastoplasticity and Viscoplasticity: Computational Aspects.* Stanford University, 1988.§

[85] Taylor, L. M. and Flanagan, D. P., "PRONTO 2D, A Two–Dimensional Transient Solid Dynamics Program," *SAND86–0594, Sandia National Laboratories,* Albuquerque, NM., 1987.§

[86] Taylor, L. M. and Flanagan, D. P., "PRONTO 3D, A Three–Dimensional Transient Solid Dynamics Program," *SAND87–1912, Sandia National Laboratories,* Albuquerque, NM., 1989.§

[87] Tvergaard, V., "Influence of Void Nucleation on Ductile Shear Fracture at a Free Surface," *Journal of Mechanics and Physics of Solids,* Vol. 30, 1982, pp. 399–425.†

[88] Tvergaard, V., "Material Failure by Void Growth to Coalescence," *Advances in Applied Mechanics,* Vol. 27, 1990, pp. 83–151.†

[89] Wang, Y., "A Two–Parameter Characterization of Elastic–Plastic Crack Tip Fields and Applications to Cleavage Fracture," Ph. D. Dissertation, Dept. of Mechanical Engineering, MIT, 1991.§

[90] Zienkiewicz, O. C. and Taylor, R. L. *The Finite Element Method*. Fourth Edition, Volume 1, *Basic Formulation and Linear Problems*. McGraw–Hill, London, 1990.†

[91] Zienkiewicz, O. C. and Taylor, R. L. *The Finite Element Method*. Fourth Edition, Volume 2, *Solid and Fluid Mechanics, Dynam,ics and Nonlinearity.* McGraw–Hill, London, 1991.†

[92] Zhong, Z., *Finite Element Procedures fot Contact-Impact Problems*, Oxford University Press, New York, 1993

* Available for purchase from national Technical Information Service, Springfield, VA 22161.

† Available from public technical libraries.

‡ Copies are available from U.S. Government Printing Office, Washington, D.C. 20402. ATTN: Regulatory Guide Account.

§ Available for purchase from vendor.

Patran-to-WARP3D Translators (*patwarp*)

C.1 Introduction

This appendix describes the procedures to communicate data between the Patran modeling/ post-processing code and the WARP3D analysis code. We assume that the reader is familiar with the use of both Patran and WARP3D.

Figure C.1 illustrates the general flow of data between the two codes. Patran is used to create interactively the geometric model and the finite element model. One form of output from Patran is denoted the "neutral file" (have Patran produce a 2.5 version neutral file for the model). This is a sequential (ASCII) file of line images that describes essential features of the finite element model in a manner independent of any specific analysis system. After building the finite element model, the user requests that Patran create the neutral file. The user initiates the program denoted *patwarp* in Figure C.1. This interactive program is a "forward translator." It reads the neutral file and produces an input file for WARP3D. The input file for WARP3D generally requires minor changes and additions by the user to include additional information, e.g. material properties, solution parameters might need to be added. Any text editor available on the computer system may be employed to make the required changes to the input file. The modified file should then be suitable for input to WARP3D to perform the analysis.

PATRAN has many features for post-processing of the analysis results. Analysis results from WARP3D (displacements, strains, stresses, etc.) are written to Patran compatible files

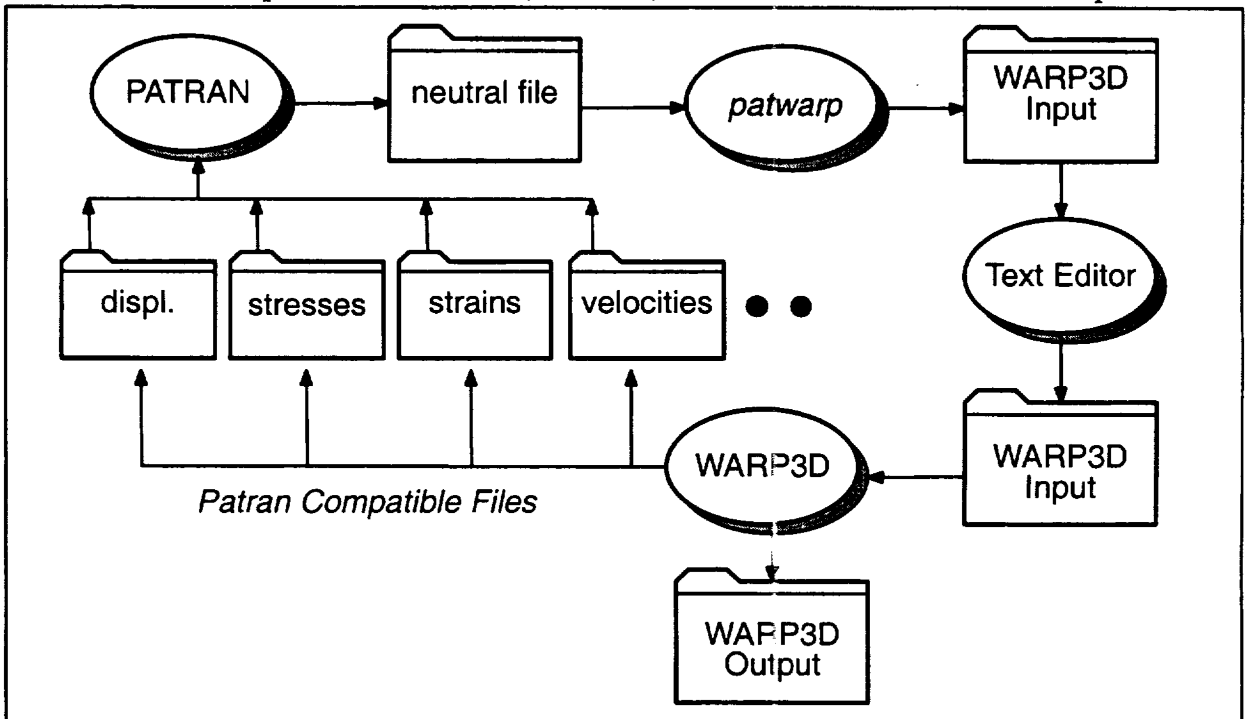


Figure C.1

at the user's request. Each of these results files contains the strains, stresses, displacements, etc. for a single load step. The file format can be either formatted or unformatted (binary) as directed in the *output* request given to WARP3D (refer to Section 2.11.2). A file written with the formatted option can be examined with a text editor but the structure of the file is not readily discerned by the user. A binary file is a sequential, unformatted data file. It cannot be examined with a text editor. Binary files are considerably smaller than formatted files but are not generally transportable between different computer architectures. WARP3D offers the option to write Patran *nodal* results files or *element* results files (see Section 2.11.2).

After analysis by WARP3D and output of the results files, continue Patran execution and the post-processing operations to read and process the results files. Because WARP3D directly generates Patran compatible results files, there is no need for a "reverse translator" program. The results files are also available to all users of WARP3D for input to other special-purpose programs.

Section 2.11 describes the ordering of results for model nodes in the Patran compatible results files. Appendix A provides a description of the data formatting in these files and small program fragments to read these files.

C.2 PATRAN-to-WARP3D Translation

The Patran–to–WARP3D translator program handles the most frequently used modeling features of WARP3D. In some cases, Patran has modeling capabilities for which there is no corresponding analysis capability in WARP3D, e.g. alternate coordinate systems. Despite these conceptual differences, the use of Patran dramatically reduces the effort for model generation and results post-processing (in nearly all cases).

The following model data in a neutral file (Patran 2.5 format) are supported by the Patran–to–WARP3D translator program:

- Structure name (a default name is generated by the translator program).
- Structure size (number of nodes/elements).
- Nodal coordinates.
- Element incidences.
- Element types.
- Nodal constraints (absolute).
- Nodal loads (forces and temperatures).
- Element loads (uniform face pressures on 8 and 20-node elements)
- Request for computation for linear models.
- Request for output of the analysis results in Patran compatible format.

The forward translator ignores other types of data contained in the Patran generated neutral file, e.g. material properties.

The above data constitute a large majority of the input for most finite element analyses. A list of modeling data not currently supported by the translator includes:

- Groups other than the default group.
- Material properties for elements.
- Physical properties for elements (*the Config Id is recognized by patwarp*)
- Alternate coordinate systems as defined in PATRAN.
- Relative constraints.
- Solution parameters which control a dynamic or a nonlinear analysis.

The processing of material properties and physical properties will be implemented in later versions of the translator program.

C.3 Executing the Translator Program

Neutral file translation is performed on all platforms by the program *patwarp* (supplied in the WARP3D distribution). The program prints an identifying message followed by a prompt for the name of the PATRAN neutral file. The names of neutral files are assigned by the user when prompted by Patran. *patwarp* verifies that the neutral file exists and that it can be processed. If the file cannot be accessed by the translator program, the prompt is re-issued. After the neutral file name has been defined, the name of the desired WARP3D input file is requested. The file may have any name and need not already exist.

patwarp performs the translation process in two steps, although these steps are transparent to the program user. In step one, the neutral file is read and the data stored internally to *patwarp*. In step two, the actual WARP3D input file is produced. A complete log of the processing is displayed for the user. After *patwarp* has terminated, the user should edit the generated WARP3D input file to supply the element properties, *output* requests, etc.

C.4 Element Mapping

Within Patran, elements are generated using the MESH menu form under the **Finite Elements** radio button on the main (top) menu). Elements have generic types such as Hex8, Hex20, etc. For example, the Hex8 element implies a 3-D solid element that has 12 edges, 6 faces and 8 nodes.

patwarp supports only elements that conform to the Hex8 and Hex20 type in Patran. The 9, 12 and 15 node transition elements available in WARP3D are created automatically by *patwarp* from the 8-node elements that share common faces/edges with 20-node elements (see Section C.11 for details on processing models with transition elements). To distinguish between different "groups" of elements (e.g. those with common material properties), the user can invoke the numeric "configuration" code provided by Patran. The configuration code assigned to each element is passed through in the neutral file for use by *patwarp*. The configuration code numbers are assigned to elements in Patran using the **Properties** menu button on the main form. This brings up the **Element Properties** menu in Patran. Click on the **Input Properties** button to bring up the menu form. The first listed item is **Config Id**.

All elements of the same type (*l3disop, ts15isop,* etc.) with the same configuration code are grouped in clearly identified lists in the generated WARP3D file. This feature simplifies considerably the assignment of options and material properties to elements following execution of *patwarp*.

C.5 Element Re-Ordering in Blocks

On vector processor computers (e.g., Crays) or for use of the Hughes-Winget preconditioner algorithm on scalar computers, WARP3D requires that elements be numbered in non-conflicting blocks. That is, no elements in a "block" may have a common node. Moreover, all elements in a block must have the same type, the same type of nonlinearity and the same material model (but not necessarily the same property values). Users may indicate common elements through the "configuration" code for the elements. Only elements with the same configuration code can be assigned to the same block.

patwarp provides the facilities to re-number elements in this blocked manner prior to generating the WARP3D input file. *patwarp* prompts the user for the maximum number of elements per block and whether the elements are to be numbered in *scalar* mode or *vector* mode. Using a simple red-black algorithm, *patwarp* then assigns elements to blocks in an order which satisfies the criteria. Elements are renumbered in sequential order within and across blocks, with a table printed in the WARP3D input file which specifies the block number, the number of elements in the block and the first element in the block.

The maximum block size varies with the computer architecture. For the Crays, this value is 128 which matches the vector lengths of the hardware. For workstations, the maximum block size is most often 64 to reduce cache memory misses.

Finally, *patwarp* offers to print a correspondence table between the original (PATRAN) element numbers and the blocked element numbers. It also offers to generate a new neutral file to reflect the new element numbering.

C.6 Constraint Processing

Nodal constraints on the three translations may be specified in PATRAN under the **BCs/Loads** radio button on the main (top) menu). The translator program builds the corresponding CONSTRAINTS data for WARP3D. Only absolute nodal constraints are handled.

In Patran, constraints may be imposed within a loading set (condition) or external to all loadings. The *patwarp* translator combines all constraint sets in the neutral file into a single block of nodal constraints.

C.7 Loads Processing

Nodal and element loads may be specified in PATRAN under the **BCs/Loads** radio button on the main (top) menu). In Patran, the user groups these cases together (along with imposed displacements) to define loading cases, e.g., the "default_case". The *patwarp* translator processes applied nodal forces, applied nodal temperatures and pressure loads applied to element faces.

When PATRAN writes the neutral file for the model, the loading cases are converted to simple loading "sets" with assigned numbers (1, 2, 3, ...). The *patwarp* translator processes applied nodal forces, temperatures and element face pressures for any number of loading sets. The translator builds the loading condition name for WARP3D as *set_n*, where '*n*' refers to the PATRAN loading set number.

The *patwarp* translator now recognizes pressure loads applied to the faces of elements. They are converted to element load commands in the WARP3D input file. When *patwarp* automatically converts an 8-node element into one of the transition elements, the user-defined pressure loads are carried forward onto the transition element as well.

C.8 Solution and Output Commands

patwarp writes the following commands in the input file for analysis and output. These commands request a solution for load step 1 of the model and output of PATRAN binary results files. These commands should be modified by the user as neeeded for specific analyses.

```
c
c
 nonlinear analysis parameters
   solution technique lnpcg
c   solution technique direct sparse
   preconditioner type diagonal
   lnr_pcg conv test res tol 0.01
```

```
    maximum linear iterations 20000
    maximum iterations 5
    minimum iterations 1
    convergence test norm res tol 0.01
    nonconvergent solutions stop
    adaptive on
    linear stiffness for iteration one off
    batch messages off
    cpu time limit off
    material messages off
    bbar stabilization factor 0.0
    consistent q-matrix on
    time step 1.0e06
    trace solution on lpcg_solution off
    extrapolate on
c
c
 compute displacements for loading test step 1
 output displacements 1-8
 output stresses 1
 output strains 1
c
 output patran binary displacements
 output patran binary strains
 output patran binary stresses
c
stop
```

C.9 Example

The following example illustrates the process of translating a PATRAN model into a WARP3D input. User supplied input values to the *patwarp* program is indicated in ***italics***.

```
% patwarp

 ***************************************************
 *                                                 *
 *     PATRAN to WARP3D neutral file translator    *
 *                                                 *
 *            Patran Version 3 and later           *
 *        (60,000 nodes - 60,000 elements)         *
 *               build date 8-28-97                *
 *    (supports 8, 9, 12, 15, 20-node elements)    *
 *                                                 *
 ***************************************************


 >> patran neutral file name
   (default: patran.out.1) ? test_29_patran.out

 >> warp3d input file name
   (default: warp3d_input) ?

 >> user title as read from neutral file is:

 P3/PATRAN Neutral File from: test_temper_2/bo
```

```
>> neutral file created on: 13-Jul-96    time: 13:19:26

>> patran version:

>>>> model size parameters:

         number of nodes .................. 1572
         number of elements ...............  717
         number of materials ..............    0
         number of physical properties   ...    1

>> begin processing nodal data
         >> processing data for node:        500
         >> processing data for node:       1000
         >> processing data for node:       1500

>> begin processing element data
         >> processing data for element:     200
         >> processing data for element:     400
         >> processing data for element:     600

>> begin processing nodal displacement data
         >> processing displacements for node:         500
         >> processing displacements for node:        1000
         >> processing displacements for node:        1500

>> begin processing nodal temperatures
  > processing nodal temperatures for node:         500 load set:      1
  > processing nodal temperatures for node:        1000 load set:      1
  > processing nodal temperatures for node:        1500 load set:      1

>> begin element reordering

>> input desired block size. use negative number to
   activate scalar mode. (default 128):128

   *******************************************
   * Warning:                                *
   *  Vectorized blocking invoked. Elements  *
   *  are renumbered.                        *
   *******************************************

>> print the new->old element listing? (y/n, default=n):n

>> begin warp3d input file generation

         >> model title and sizes written
         >> nodal coordinates written
         >> element types written
         >> element incidences written
         >> blocking command written
         >> nodal loads written
         >> element loads written
         >> constraints written
         >> warp3d input file completed

>> make an updated patran neutral file(y,n, default=n)?n

>> analysis file generation completed.
>> job terminated normally.
```

C.10 Limitations and Advice

The WARP3D code requires that elements and nodes be numbered sequentially. Patran performs this task through the **Renumber** option of the main **Finite Elements** menu. After the model is generated, the user should always perform a node and element compaction within Patran.

If the (old) direct solver option is used in WARP3D, the node numbering scheme should always be optimized before generating the neutral file. Use the node-id optimization command and select the RMS wavefront to be minimized. This produces a node numbering scheme with minimum active triangle.

C.11 Processing of Models Containing Transition Elements

The 9, 12 and 15 node elements (*ts9isop, ts12isop, ts15isop*) enable development of models containing 8 and 20 node elements which maintain complete displacement compatibility. However, Patran does not directly support such transition elements. Models containing these elements can be constructed and post-processed using Patran with support of the *patwarp* program.

1. Create the Patran model using Hex/8 and Hex/20 type elements. At this stage there will be mismatches in the number of nodes on common faces/edges shared between the 8 and 20-node elements. Apply nodal constraints and element pressure loadings as necessary to the model. Configuration ids to signify different materials should be included at this point.
2. Create the Patran neutral file for the model.
3. Run the *patwarp* program. Once the neutral file has been read into memory, *patwarp* will ask the user if the creation of transition elements is desired. If yes, *patwarp* searches the user-defined 8 and 20-node elements to find all the shared faces/edges. It then redefines the 8-node elements as one of the transtion elements (9, 12, 15 node elements) needed to maintain full displacement compatibility in the model. In this process, *no new nodes or elements are added to the model*. The 8-node elements are simply refined as transition elements by appending existing nodes (shared with 20-node elements) to their incidence list and then re-ordering the incidences to conform with the ordering requirements for nodes in WARP3D.
4. *patwarp* then executes the normal blocking strategy to renumber elements into blocks of common element types and writes the WARP3D input file.
5. Patran compatible node and element result files are thus unaffected by the conversion of some 8-node elements into transition elements. Patran believes it is processing a mesh of only 8 and 20-node elements. When the user requests that *patwarp* write a new neutral file for the model to reflect the blocked element re-ordering, the transition elements are written as standard 8-node elements.

Tips for Modeling Fracture Specimens

D.1 Use Large Elements at the Loading Point

The imposition of loads through localized nodal forces or nodal displacements creates severe stress–strain concentrations which may cause the nonlinear solution to converge very slowly or not at all. The simplest remedy relies on the use of large elements at the load point as shown in Fig. D.1. The large elements smooth the solution by effectively spreading the load over a large region. For problems that can be loaded through applied forces, rather than through imposed displacements, spread the loading region over a number of elements.

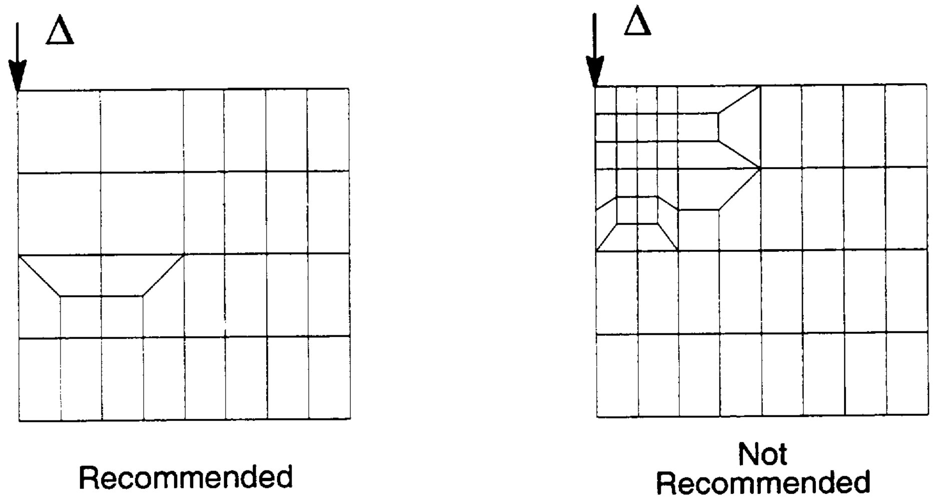


FIG. D.1—*Mesh Refinement Near Load Points*

D.2 Displacement Extrapolation

Our experience indicates that the displacement *extrapolation* feature of the nonlinear solution procedure accelerates convergence, especially for large displacement-finite strain solutions. In high-rate analyses, the lack of a self-similar deformation pattern leads to a large Euclidean norm of the residual load vector after the first iteration. However, subsequent iterations within a step typically converge more rapidly following displacement extrapolation in the first iteration.

D.3 Displacement Loading During High-Rate Analyses

For high-rate loading, non-zero displacements should be applied on a single line of nodes over the entire thickness of the specimen. Displacement loading techniques which do not follow this recommendation often produce non-convergent solutions. Figure D.2 shows three methods for applying non-zero displacements (only the first method is recommended for high-rate analyses). The two methods not recommended for high-rate loading are found to introduce convergence problems in the analysis. Applying displacements to multiple nodes through the depth under high-rate loading requires a rigid body acceleration for this section of the model (shaded region of figure). If displacements are not applied over the entire thickness of the model, a localized, high-rate "punching" deformation pattern may cause the residual to become large at the edge of the applied displacement field.

For static analyses, we find that imposition of the nodal displacements over multiple sets of through-depth nodes, as in Fig. D.2 (b), provides more readily convergent solutions.

D.4 Newton Tolerances

Four convergence tests to control the termination of global Newton iterations exist currently in WARP. Test number two, for example, compares the ratio of the Euclidean norm of the residual force vector to the Euclidean norm of the total applied force vector, with a user specified value of an acceptable tolerance. This ratio of Euclidean norms should decrease towards zero by roughly one order of magnitude per Newton iteration. If a slower convergence rate persists over several iterations, the model should be re-examined relative to the load step size and the modeling guidelines discussed here.

To provide a specific example of tolerances for convergence values, consider the 3–D analysis of Charpy fracture specimen (10 mm square cross-section, $a/W = 0.5$). Elements along the crack front have a nominal edge length of 10 μm. The analysis uses the small-strain kinematic formulation with a rate dependent, linear/power law material model ($E/\sigma_0 = 500, n = 10, m = 35, D = 1$). The high-rate analysis covers 200μs in 400 loading steps to push the response well into the fully yielded regime with a constant time increment of 0.5μs. The solution converges in 3-4 iterations for a typical step using a Newton tolerance of 0.005 (convergence test number two).

D.5 Maximum Residual Force

At the completion of each Newton iteration, WARP outputs the value for the maximum residual force and node at which this residual occurs. If a step fails to converge rapidly, the location of the maximum residual force may indicate the region of the model responsible for poor convergence performance.

D.6 Adaptive Load Step Sizes

The adaptive solution strategy provided in WARP often proves essential in exploratory analyses for which satisfactory load step sizes are unknown. When the specified limit on the number of Newton iterations is exceeded, the current load step is first sub–divided into four equal sub–steps; each of these sub–steps can be again divided into four more equal steps if required. Even with this process, we occasionally experience problems for which the adaptive solution strategy fails to produce a convergent solution. The difficulty is most often traced to insufficiently tight convergence tolerances which allowed previous steps to accumulate too much error.

D.7 Non-Convergent Solutions

Nonlinear analyses occasionally exhibit load steps which converge slowly despite seemingly rapid convergence rates in previous and subsequent steps. We refer to these as "sticky"

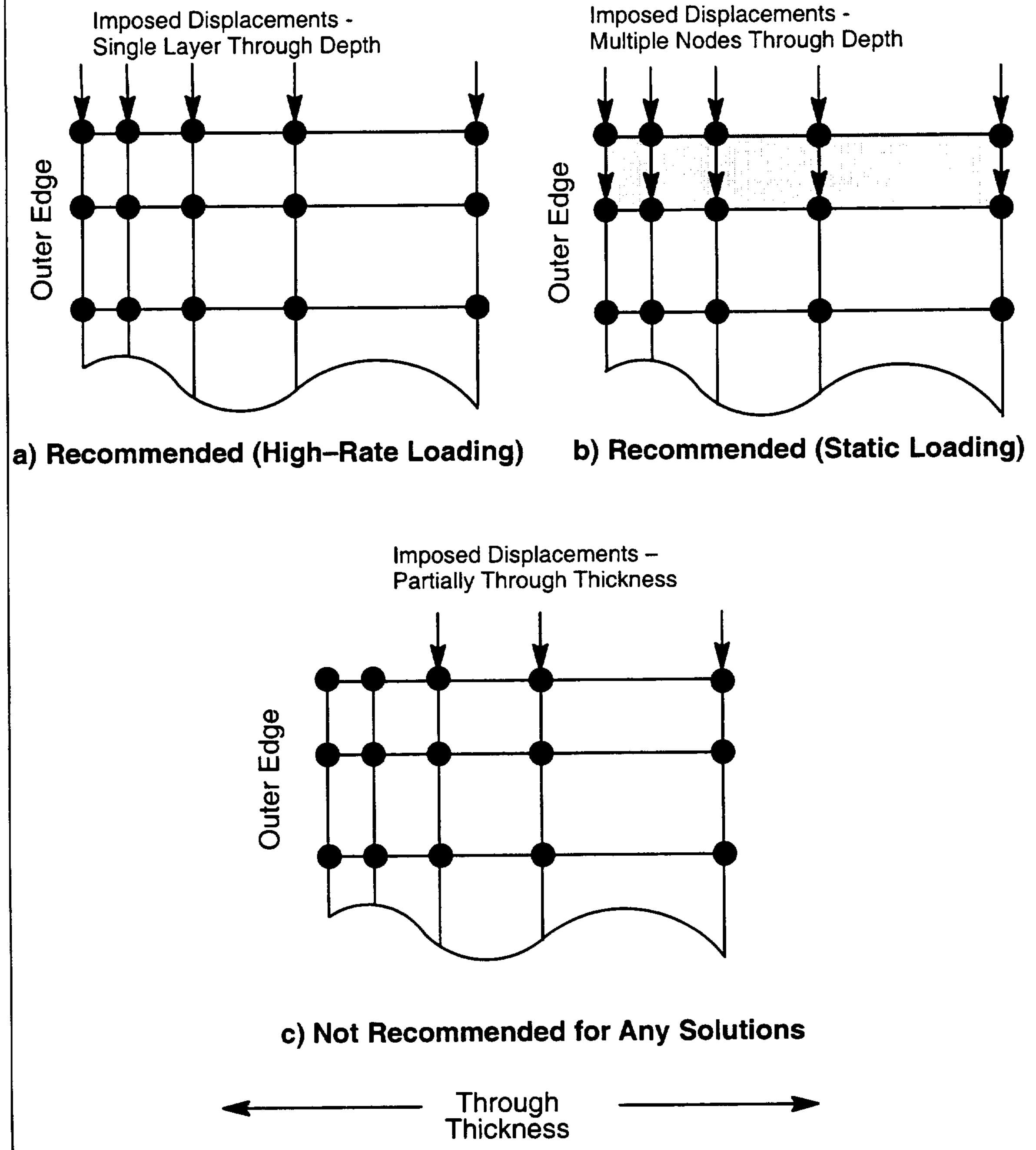

Imposed Displacements - Single Layer Through Depth
Outer Edge
a) Recommended (High–Rate Loading)
Imposed Displacements - Multiple Nodes Through Depth
Outer Edge
b) Recommended (Static Loading)
Imposed Displacements – Partially Through Thickness
Outer Edge
c) Not Recommended for Any Solutions
Through Thickness

FIG. D.2—*Procedures to Impose Loading by Nodal Displacements*

steps; they can be caused, for example, by extension of the plastic zone across elements that vary widely in size. WARP provides the solution parameter "nonconvergent solution continue" which forces the analysis to continue beyond a non-convergent step. If subsequent steps show a return to rapid convergence rates, there is no accumulation of error in the solution (WARP employs a *total* equilibrium formulation to compute residual nodal forces). Our experience suggests this parameter may be helpful when other options have been exhausted.

D.8 Mesh Refinement for High-Rate Loading

High-rate loading may deform a specimen in a substantially different manner than static loading. The analysis of high-rate loading may require additional mesh refinement in regions experiencing localized deformation. For impact loading of the Charpy specimen considered in the previous example, we find it necessary to extend detailed mesh refinement to the quarter-span position. The additional mesh refinement is necessary to resolve the severe bending induced gradients that occur at the quarter-span location. Without the mesh refinement, resolution of the plastic zone of large elements with steep gradients created convergence problems.

D.9 Blunt Notch Tip for Finite Strain Analyses

The use of "focused" element meshes to model a sharp crack tip works very well in linear elastic and small-strain plasticity analyses and no difficulties surface in the Newton computations. When such analyses must include the effects of finite strains along the crack front, focused meshes often introduce convergence difficulties due to the exceedingly large strain increments in elements incident on the front. In such cases, the crack tip should be modeled as a "blunt" tip having a small, semi-circular shape (see figure). The undeformed opening, b_0, should be defined in the model as approximately 1/3 of the deformed opening, b, at which results in the crack tip region are needed. Once the deformed opening exceeds $\approx 3 \times b_0$, parameter studies reveal a minimal affect of the b_0.

The use of a ***B***-bar modification to suppress volumetric locking in the 8-node solid element occasionally introduces spurious deformation modes in elements on the free surface of the blunt notch tip. This phenomena develops when the strains in these elements become exceedingly large. The stabilization procedure described in Sect. 2.9.11 may or may not prevent the spurious modes in crack tip elements. Our experience reveals that completely eliminating the ***B***-bar formulation resolves this problem, thereby allowing the achievement of larger b/b_0 ratios.

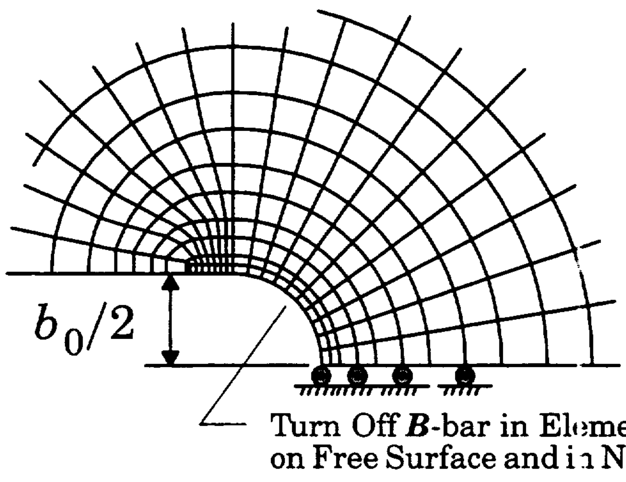

$b_0/2$
Turn Off ***B***-bar in Elements on Free Surface and in Next Ring

FIG. D.3—*Recommended Model of Crack Tip for Finite Strain Analyses*

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. SRS 607	2.	3. Recipient's Accession No.

4. Title and Subtitle WARP3D: Dynamic Nonlinear Analysis of Solids Using a Preconditioned Conjugate Gradient Software Architecture	5. Report Date September 1998
	6.
7. Author(s) K.C. Koppenhoefer, A.C. Gullerud, C. Ruggieri, R.H. Dodds, B.E. Healy	8. Performing Organization Report No. UILU-ENG-95-2012
9. Performing Organization Name and Address University of Illinois at Urbana-Champaign, Department of Civil Engineering, 205 N. Mathews Avenue, Urbana, Illinois 61801	10. Project/Task/Work Unit No.
	11. Contract(C) or Grant(G) No.
12. Sponsoring Organization Name and Address U.S. Nuclear Regulatory Commission, Office Of Nuclear Regulatory Research, Division Of Engineering, Washington, D.C.; NASA-Ames Research Center, Moffett Field, California	13. Type of Report & Period Covered
	14.

15. Supplementary Notes

16. Abstract (Limit: 200 words)

This report describes theoretical background material and commands necessary to use the WARP3D finite element code. WARP3D is under continuing development as a research code for the solution of very large-scale, 3-D solid models subjected to static and dynamic loads. Specific features in the code oriented toward the investigation of ductile fracture in metals include a robust finite strain formulation, a general J-integral computation facility (with inertia, face loading), an element extinction facility to model crack growth, nonlinear material models including viscoplastic effects, and the Gurson-Tvergaard dilatant plasticity model for void growth. The nonlinear, dynamic equilibrium equations are solved using an incremental-iterative, implicit formulation with full Newton iterations to eliminate residual nodal forces. Time history integration of the nonlinear equations of motion is accomplished with Newmark's β method. A central feature of WARP3D involves the use of a linear-preconditioned conjugate gradient (LPCG) solver implemented in an element-by-element format to replace a conventional direct linear equation solver. This software architecture dramatically reduces both the memory requirements and CPU time for very large, nonlinear solid models since formation of the assembled (dynamic) stiffness matrix is avoided. Analyses thus exhibit the numerical stability for large time (load) steps provided by the implicit formulation coupled with the low memory requirements characteristic of an explicit code. In addition to the much lower memory requirements of the LPCG solver, the CPU time required for solution of the linear equations during each Newton iteration is generally one-half or less of the CPU time required for a traditional direct solver. All other computational aspects of the code (element stiffnesses, element strains, stress updating, element internal forces) are implemented in the element-by-element, blocked architecture. This greatly improves vectorization of the code on uni-processor hardware and enables straightforward parallel-vector processing of element blocks on multi-processor hardware.

17. Document Analysis a. Descriptors

Finite elements, conjugate gradient, finite-strains, plasticity, Newton, supercomputers

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement Release Unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 120
	20. Security Class (This Page) UNCLASSIFIED	22. Price

(See ANSI-Z39.18)

OPTIONAL FORM 272 (4-77)
Department of Commerce