

FAST3D USER AND PROGRAMMER REFERENCE MANUAL

FAST3D_BETA_1.1

27 January 1998

prepared by

Alexandra Landsberg, Almadena Chtchelkanova, Charles Lind,
Jay Boris, and Theodore Young

Introduction and Overview

- I.1. About the FAST3D CFD Modeling System
FAST3D Manual Style Guide
- I.2. Getting Started
- I.3. FAST3D Capabilities
- I.4. What Defines a CFD Problem for FAST3D?
- I.5. Six Steps in Using the FAST3D System
- I.6. FAST3D Directory Structure and Contents
- I.7. General Data Input for FAST3D
- I.8. Running FAST3D Programs

Step 1. How to Define a CFD Simulation for FAST3D

What Constitutes a Properly Posed CFD Simulation?

- 1.1. A Problem Template and Naming Conventions for FAST3D
- 1.2. Choosing a FAST3D Problem for Execution
- 1.3. Multiple FAST3D Problems Active Simultaneously

Step 2. How to Specify and Check a Geometry for FAST3D

The FAST3D Approach to Complex Geometry

- 2.1. Coding the User-Supplied Geometry Subroutine 'geometry.f'
- 2.2. Compiling and Linking with FAST3D Programs
- 2.3. Specifying Input Data to 'geometry.f'
- 2.4. Running VOYEUR to Check the Geometry

Step 3. Running GRIDVCE to Generate a FAST3D Grid

The Role of the Generated Grid in FAST3D

- 3.1. Specifying Input Data to GRIDVCE
- 3.2. Running GRIDVCE
- 3.3. Monitoring and Checking the Resulting FAST3D Grid
- 3.4. Transferring the FAST3D Grid File

Step 4. Conducting CFD Simulations Using FAST3D

Operation of the flow solver FAST3D

- 1. Specifying Input Data to FAST3D
- 2. Running the FAST3D Flow Solver
- 3. Monitoring FAST3D Runs During Execution
- 4. Killing the FAST3D Flow Solver
- 5. Restarting FAST3D for Continuation Runs

Step 5. Using VOYEUR to Visualize the Computed Flow

General Description of the VOYEUR Program

- 5.1. Specifying Input Data to VOYEUR
- 5.2. Running the VOYEUR Graphics Package On-line
- 5.3. Modifying and Adjusting VOYEUR Plots in 'Real Time'

5.4. Killing the VOYEUR Package

Step 6. Post-Processing FAST3D Simulations

General Discussion

- 6.1. The FAST3D Parallel Dump/Restart Files
- 6.2. Using VOYEUR Cross Section Files for Post-processing
- 6.3. Specifying Station History Files for Post-processing
- 6.4. Reading/Interpreting the Station History Files

Section 7. FAST3D Test Problems

General Discussion

- 7.1. Blast Problems with Null Geometry
- 7.2. Blunt Body and Jet Problems
- 7.3. Muzzle Blast Problems
- 7.4. Wedge/Cone Problems
- 7.5. Circular Arc Airfoil Problems
- 7.6. Multiple Ellipse Problems

Section 8. Additional Information About the FAST3D System

- 8.1. Flux-Corrected Transport and Virtual Cell Embedding Algorithms
- 8.2. Tradeoffs Made in Constructing the FAST3D System
- 8.3. The VCE Data Structure in FAST3D
- 8.4. Portable/Heterogeneous Implementation of FAST3D
- 8.5. The FAST3D Model Limitations

Section 9. Advanced Uses and Planned Capabilities for FAST3D

- 9.1. Advanced Uses and Tricks for FAST3D
- 9.2. Parallel Data Input for User-Generated Routines
- 9.3. Making geometry.f More Efficient
- 9.4. Planned Capabilities for FAST3D (updates and future versions)
- 9.5. Problem Reports for FAST3D

References and Figures

Appendix A: Definition and Initialization of *runname.gi* Variables

Appendix B: Definition and Initialization of *runname.ni* Variables

Appendix C: Definition and Initialization of *runname.vi* Variables

Appendix D: Test Problem Data Sets and Selected Results

1. Blast Problems with Null Geometry
2. Blunt Body and Jet Problems
3. Muzzle Blast Problems
4. Wedge/Cone Problems
5. Circular Arc Airfoil Problems
6. Multiple Ellipse Problems

Appendix E: FAST3D File Contents and Subroutine Descriptions

Appendix F: Scripts used for running FAST3D

Introduction and Overview

I.1. About the FAST3D CFD Modeling System

The FAST3D modeling system is built around a Computational Fluid Dynamics (CFD) model for solving three-dimensional, time-dependent, compressible, reactive flow problems in geometrically complex configurations. This document describes the three major software components of the system: the grid generator GRIDVCE, the flow solver FAST3D, and an associated asynchronous graphics package, VOYEUR. The name, FAST3D, refers to the entire software system as well as the flow solver. A distinction will be made when necessary.

The underlying fluid dynamics algorithm used in FAST3D is the high-resolution, direction-split, Flux-Corrected Transport (FCT) algorithm called LCPFCT. The subroutine packages associated with LCPFCT are documented in NRL Memorandum Report 6410-93-7192 (Boris, *et al.*, 1992). The complex geometry capabilities are provided by an efficient parallel implementation of the Virtual Cell Embedding (VCE) algorithms including the direction-coupling surface divergence transfer terms. These underlying algorithms are described in Landsberg, *et al.* (1993a, 1993b, 1994). The parallel implementation is discussed in Section 8.4 and is also described in Young, *et al.* (1993).

The purpose of this report is to give users a step-by-step reference manual for using the FAST3D programs. Several test problems will be provided with the FAST3D system. These problems can be run as specified or used as the basis for new problem specifications. In addition, examples of the data input files, lists of variable names, their descriptions, and program options will be provided. FAST3D is designed to be run as a 'white box' - a suite of programs into which a moderately skilled user can insert their own code or make their own modifications for special features needed for their particular problem.

I.2. Getting Started

The FAST3D package is made up of a User Reference Manual (this manual), the FAST3D cheat sheet (Appendix A), and the FAST3D suite of code.

The FAST3D software package is distributed as a single, compressed, tarred file: `fast3d.tar.Z`. Since it is recommended that the FAST3D directory structure be a subdirectory in the users \$HOME space the `fast3d.tar.Z` file should be placed in there. The actual installation of the code is fairly straight forward. First the file must be uncompressed using the Unix `uncompress` command:

```
prompt> uncompress fast3d.tar.Z
```

The `uncompress` command will rename the `fast3d.tar.Z` file to `fast3d.tar`. Next, the Unix `tar` command is used to extract the directory structure, along with the associated files:

```
prompt> tar -xvf fast3d.tar
```

These two steps may be combined as follows:

```
prompt> zcat fast3d.tar.Z | tar -xvf -
```

After extracting the files the user should verify the integrity of the directory structure, as explained in Section I.6. FAST3D Directory Structure and Contents. Note that once uncompressed and untarred, the name of the primary directory will reflect the current version of the code and the suffix of the subdirectories `fast3d`, `gridvce`, and `voyeur`, will reflect that applications current version. The main `fast3d` directory, with its associated version number, should be renamed to `fast3d` as follows:

```
prompt> mv fast3d_version_number fast3d
```

The actual process by which one generates the grid using GRIDVCE, runs the code using FAST3D and views the results using VOYEUR, is explained in Section I.5. Six Steps in Using the FAST3D System.

To simplify reading and understanding, several standard printing conventions have been adopted in this manual:

Bold Face Font (Times)	is used for section headings, and for figure and table labels, and to emphasize new terms and concepts in the text.
Constant Width Font (Monaco)	is used for file names and directory names when they appear in the body of a paragraph and to show contents of files or the output of commands. For example, <code>fast3d/gri dvce_0. 1/gri dvce. f</code>
Italic Constant Width Font (Monaco)	is used to show variables for which a context-specific substitution should be made. For example, in <code>flow_fi el d. f_ probname</code> , the extension <code>probname</code> would be replaced by a specific problem name such as <code>fl ow_ fi el d. f_ bl ast</code> .
Constant Width Bold Font (Monaco)	represents commands that should be typed literally as they appear in the text or example. For example, <code>buil d_ fast3d -f bl ast</code> indicates that one should type this command at the command line.
ALL CAPITALS	in the text indicates the FAST3D system or one of the generic program names within the system as distinct from the file or routine name. For example, GRIDVCE, FAST3D, VOYEUR.

I.3. FAST3D Capabilities

The FAST3D capabilities can be divided into two categories: (1) capabilities and features of the FAST3D system and (2) capabilities and features of the FAST3D flow solver.

The FAST3D system is implemented to obtain computational efficiency with programming simplicity by using a single, globally structured grid extending throughout the domain. In the parallel implementation of FAST3D, domain decomposition is used to distribute the grid cells across the processors while a running transpose is used for the interprocessor communications. The VCE data structure implemented in FAST3D is also partitioned for efficient parallel processing and communication. Communication time is minimized when executing on a parallel processing system, i.e. FAST3D is truly fast. Another important feature of FAST3D is each cell requires only a few words of memory per cell (typically 7 - 10). Thus, problems necessitating large computational grids are best suited for FAST3D since the memory requirements per cell are low.

The FAST3D flow solver can handle a wide range of unsteady, time-accurate, compressible flow problems. The FCT algorithm, upon which the flow solver is based, is a high-order, high resolution algorithm. The VCE algorithms handle extremely complex geometries. Physical flow features such as vortex shedding, shocks, detonations and other reactive flow phenomena have been captured accurately and compared to experimental data and/or other computational data. A hydrogen-oxygen induction parameter (reduced chemistry) model is included in FAST3D. This model allows multiple chemical species sharing a single velocity field. Interspecies diffusion, thermal diffusion, radiation transport and complex chemical reaction sets with stiff equations are not treated. The initial implementation has a constant gamma equation of state although future releases will generalize this limitation. Another feature included in FAST3D is a simple gravitational acceleration model which can be used for large-scale problems; however, a hydrostatic atmospheric equilibrium is not guaranteed.

A more extensive discussion of FCT and VCE is given in Section 8.1 and limitations of the FAST3D flow solver are discussed in Section 8.5.

I.4. What Defines a CFD Problem for FAST3D?

Before outlining the six steps a user needs to follow to run a FAST3D simulation, it is necessary to explain in general terms what input is needed to specify a CFD problem. Of course, we are assuming that the user has already decided that the physical/fluid dynamic questions being tackled are within the scope of the mathematical and physical models in FAST3D.

A FAST3D CFD problem definition has six components:

a - A geometry for the flow field being studied (e.g. the shape of the wing, the dimensions and configuration of the combustion chamber, or the shape of a blast shield). This may include bodies piercing or even defining the boundaries of the computational domain.

b - Information sufficient to specify the initial conditions of the flow variables at every point in the computational domain. This includes how many chemical species there are, their physical

properties, and the flow variables of the fluid on and about the solid bodies embedded in the domain. In addition, the boundary conditions to be applied at the edges of the domain must be specified.

c - A procedure to modify the flow field during the simulation. Typical examples include adding localized sources to model a blast at a specific location and time, injecting fuel or gases at a certain rate and position, or extracting material at an interface or surface. A special case built directly into the FAST3D model is the ability to integrate chemical rate equations or simplified 'induction parameter' energy release equations between the chemical species as the multidimensional flow field evolves.

d - The specification of the spatial grid defining the computational domain on which the simulation is to be performed. These data are used by the grid generator program GRIDVCE or can be used directly by the flow solver for self-initialization of the grid.

e - The specification of the variables controlling the execution of the flow solver FAST3D. This includes number of time steps (run duration), the check point dump/restart frequency, the active file and problem names, paths to various directories, etc.

f - Information about the frequency, spatial locations and physical variables of the output required (both numerical and graphical). For example, every specific cross-section desired for interactive plotting must be specified, including the particular variable to be plotted, the contour levels required for each of these plots, and the extent and magnification factors for the plots. FAST3D also permits station history files to be defined and written for use with other user-selected data analyses and graphics packages. These station history files save the flow variables at regular time intervals and at specified points, along particular lines, on selected planes or subplane cross-sections of the domain, or on specified subcubes of the domain.

Each of these six components of a problem definition will be described in detail in relation to the specification and modification of input and subsequently controlling the execution of GRIDVCE, FAST3D, and/or VOYEUR.

I.5. Six Steps in Using the FAST3D System

To switch from one set of initial conditions to another, or to a different variation of the basic geometry, or to a generate a wholly different CFD problem, a number of specific changes have to be made in the various data files used by GRIDVCE, FAST3D, and VOYEUR. Sufficiently complex modifications may even require programming user-supplied subroutines to be compiled and linked with the FAST3D system. The next six sections of this manual provide details and examples of these changes and organize the necessary information relating to them into a logical sequence of six steps. Thus these six sections are titled 'Step 1 ...' through 'Step 6 ...' for clarity rather than Section 1 through Section 6. Each of these steps deals with one or two of the aspects of defining a CFD problem listed above. These steps (manual sections) are now described briefly to give the reader an overview of the manual's contents and the road map for conducting a FAST3D simulation.

Three level of user expertise will be defined here. The “novice” user typically runs the code with select parameter modifications, primarily changing input parameters in the supplied input files. The “intermediate” user can change any input parameters in the supplied input files, create new input files and modify the user-supplied subroutines for geometry and flow field specification. The “advanced” user has the freedom to modify any file associated with the FAST3D package; however, this is at the user’s own risk. We will assist user’s with code modifications when appropriate; however, we are not responsible for externally-created coding errors.

Section 1 entitled, 'Step 1. How to Define a CFD Simulation for FAST3D', describes the preparatory actions a user must take to set-up a FAST3D problem. The first three of the problem-defining components in the list from Section I.4 above, a, b, and c, are generally embodied in subroutines because of the variation and generality of the wide-range of problems FAST3D is intended to solve. Default (null) routines are provided and described in Section 7, entitled 'FAST3D Test Problems'. These defaults are sufficient for many cases and can also serve as templates for a user graduating from the novice to the intermediate level and wanting to do something more complex.

The problem-defining components d, e and f listed above are contained in three data input files. For simple problems or for a set of closely related problems (subproblems) based on a generic geometry, changing the input numbers in these data files is all that is required to initialize and control the execution of the FAST3D programs. 'Step 1' defines the naming conventions used for problems and subproblems in FAST3D, what directories and files will have to be written, copied, or modified to construct a new problem template, how to 'select' a FAST3D problem for execution, and an approach for manipulating several FAST3D problems simultaneously.

The second section is 'Step 2. How to Specify and Check a Geometry for FAST3D'. An arbitrary geometry containing multiple bodies is defined for FAST3D by a single subroutine named `geometry.f`. A complex geometry can be built up as the union of a number of simple bodies. Several standard test problem examples of `geometry.f` are given in Section 7. The graphics package VOYEUR can then be used to look at selected cross-sections of this user-specified geometry before the FAST3D grid is generated or the simulation run.

The third section is 'Step 3. Running GRIDVCE to Generate a FAST3D Grid'. For geometries with curved bodies or straight walls that do not align with the planar FAST3D cell interfaces, the VCE method requires auxiliary data structures to be initialized. These data describe the interface areas and cell volumes of partially obstructed cells and temporary storage for boundary 'source' terms to take account of fluid dynamic fluxes impinging on the body. 'Generating' this grid information is the job of a program called GRIDVCE. The third section tells how to specify the input data to GRIDVCE, how to run and stop ('kill') GRIDVCE, and how to monitor and check the resulting FAST3D grid. Input data about the grid and the geometry is specified in a file `runs/probname/runname.gi`. One of the files GRIDVCE produces is named `longrunname.geo0`. This problem-specific geometry output file is subsequently used by the flow solver FAST3D as input. This file `longrunname.geo0` may need to be transferred to a parallel, high speed I/O storage system for execution with FAST3D. If this transfer is required, a highly system-dependent issue, there are naming conventions for the corresponding directory structure and path names to be set in the input data.

The fourth section below, 'Step 4. Conducting CFD Simulations Using FAST3D', describes the preparation of data for and operation of the FAST3D flow solver. Input data to FAST3D is specified in a file `runs/probname/runname.ni`. Once the user has submitted FAST3D to the scalable computer and has it running, keeping track of the execution becomes an issue. Useful techniques to monitor FAST3D runs during execution are presented, both for interactive runs and batch runs. This monitor information is designed to tell the user whether FAST3D is running, how well, and how efficiently. The procedures for stopping ('killing') the FAST3D flow solver gracefully are described. The simple actions needed to restart FAST3D for a continuation run from where it left off are presented. Two methods are provided to run the FAST3D flow solver (as well as the other FAST3D programs), and these are described generically in Section I.7 as well as in each of the steps involving the specific programs.

The fifth section, 'Step 5. Using VOYEUR to Visualize the Computed Flow', includes a general description of the VOYEUR interactive graphics program and how to run it. VOYEUR is rather limited in its graphical capabilities (generating two-dimensional planes) but its execution is asynchronous from the flow-solver allowing for automatic updates and real-time visualization *while* FAST3D is running - a big advantage for real-time analysis and initial run debugging. The user must specify and modify input data to VOYEUR 'on-the-fly' through the third FAST3D data file, `runs/probname/runname.vi`. The fifth section describes how to run the VOYEUR package and how to modify and adjust VOYEUR color pixel cross-section plots in 'real time'.

VOYEUR is supplied with the FAST3D package and it is our intention to keep it compatible and current with both FAST3D and GRIDVCE. Complete support for it and resolution of all portability issues, however, cannot be guaranteed at this time. Since VOYEUR typically executes asynchronously on a (local) system different from the scalable system running FAST3D, the generic portability issues, are compounded significantly. Supporting this composite portability requires interactive, 'on-demand' access to the scalable platform running FAST3D for each implementation. Efforts are underway to guarantee this on the new scalable platforms at NRL. Running VOYEUR at other DoD sites, the Major Shared Resource Centers (MSRCs) and geographically distributed over the DREN, is probably best tackled on a case-by-case basis. We expect to involve the PET efforts at the MSRCs and interested users throughout the community in this auxiliary effort.

The sixth section is entitled 'Step 6. Post-Processing FAST3D Simulations'. In FAST3D, as in most large CFD models, the range of intended applications is sufficiently wide that a significant level of 'post-processing' of the data is usually required. In 'Step 6 ...' the three facilities included in the FAST3D system to make this post-processing easy are described: the FAST3D parallel dump/restart files, using VOYEUR cross section files for post-processing, and specifying detailed 'station history' files for FAST3D to write. These station history files represent a flexible, user-controlled data structure for which an auxiliary program is provided to re-read and interpret the resulting files.

Section 7 describes the suite of six 'FAST3D Test Problems'. These are generic problem classes for which geometries and several specific data sets for FAST3D subproblems are provided. The 'null' or 'default' versions of the three user-supplied subroutines `geometry.f`,

`flow_field.f`, and `flow_mods.f` are printed and explained. Each of these problem sets is described in its own subsection:

- 7.1. Blast Problems with Null Geometry,
- 7.2. Blunt Body and Jet Problems
- 7.3. Muzzle Blast Problems
- 7.4. Wedge/Cone Problems
- 7.5. Circular Arc Airfoil Problems'
- 7.6. Multiple Ellipse Problems'

Each was chosen to illustrate, test, and demonstrate different features of FAST3D.

Section 8, 'Additional Information About the FAST3D System', presents useful information that is not part of the step-by-step FAST3D procedure per se but that will eventually be important as users progress in their level of sophistication in using FAST3D. The underlying Flux-Corrected Transport and Virtual Cell Embedding algorithms and the resulting VCE data structures are described. FAST3D's portable/heterogeneous system implementation is described and related to the tradeoffs that had to be made in constructing the software to balance simplicity, flexibility, generality, robustness, accuracy, efficiency, and portability. Some of the major FAST3D model limitations are described.

The final section, 'Section 9. Advanced Uses and Planned Capabilities for FAST3D', briefly suggests some non-obvious uses of FAST3D and tricks that can be used to accomplish the same ends as would be possible with more complicated software. In very complex geometries, making `geometry.f` more efficient can also be quite important. Reading parallel data into the three user-supplied routines `geometry.f`, `flow_field.f`, and `flow_mods.f`, in an architecture-independent way, has been a difficult issue. A user-friendly approach has been developed and implemented and is also described in this section. Section 9 also lays out some of the planned future capabilities for the FAST3D CFD modeling system. This manual concludes with procedures for making problem reports and tells the user how to prepare for subsequent versions and updates to FAST3D.

Detailed discussions of the specific control and data variables in the input data files to GRIDVCE, FAST3D and VOYEUR are deferred to Appendices B, C and D, respectively. Appendix E contains the actual test problem data sets whose descriptions and explanations are found in Section 7. This appendix also contains some selected results so the user can be assured of knowing that the program 'port' was correctly accomplished. Appendix F contains a list of FAST3D file contents, directory by directory, supported by a brief list of subroutine descriptions.

1.6. FAST3D Directory Structure and Contents

The directory structure for the FAST3D programs has been arranged based on previous experience running and developing these programs. The structure used here should be straight-forward for new users, facilitate starting new CFD problems, and simplify performing computations on several CFD problems simultaneously. Though other conventions could be implemented, possibly more tuned to a particular user's taste, following the procedures laid out here, and the associated file and

directory naming conventions, will allow for easier debugging of any problems encountered. It will also ensure that the various scripts and global makefile provided can find the routines and files that are needed.

An overall outer directory is established for the entire suite of FAST3D codes at some level in the user's file structure. We called it `fast3d/` (the system - not to be confused with the flow solver). This outer directory contains a number of subdirectories within it.

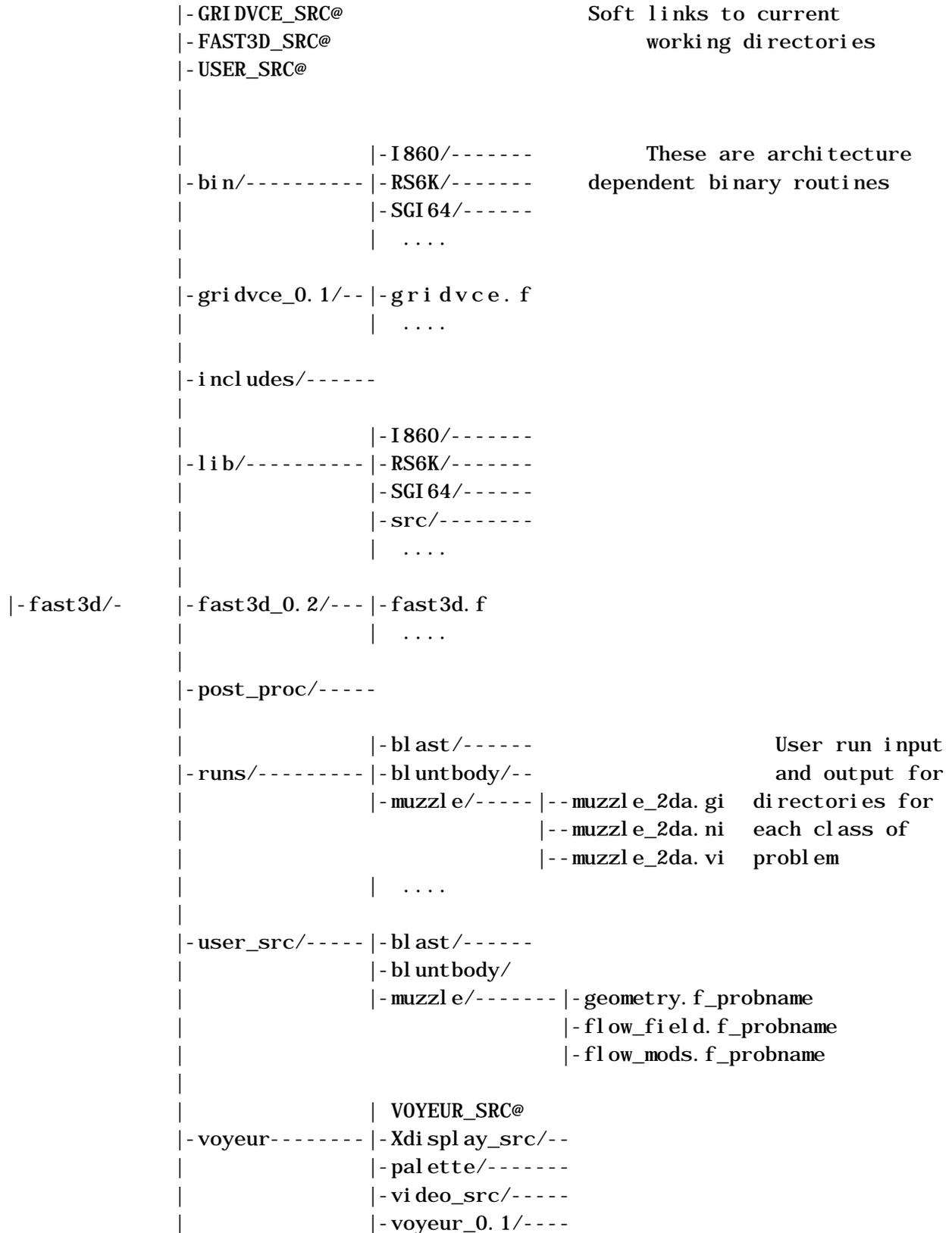


Figure. 1 . FAST3D Directory Structure

A general description of each directory follows. A detailed description of these files and the subroutines in them can be found in Appendix F.

Directory `bin/` and the files within it will generally never be touched by the user. The sub-directories under `bin/` contain architecture-specific executables. The corresponding files are utilities, such as script files, that facilitate all aspects of using the FAST3D suite of programs. Currently architecture-specific differences have been resolved for the Convex SPP2000, the SGI Origin 2000, the SGI Power Challenge Array, the IBM SP and for the Intel i860 and Paragon computers. The CFD benchmark program derived from FAST3D has also been run on the Cray C-90 and IBM SP-2 systems.

Directory `gridvce_vno/` contains the source files for the FAST3D VCE grid generator. The extension here, `vno`, indicates the current version number. The files within this directory do not need to be modified when changing between FAST3D problems or subproblems. The user-supplied subroutine `geometry.f` resides in the directory `user_src/`.

Directory `lib/` contains source files (in the subdirectory `src/`) and object files organized as a library for use by all of the programs of the FAST3D software suite. Routines appear in this library for two reasons: either they are so generic or abstruse that the general user would never be expected to need to study or modify them (e.g. data formatting, buffer construction, file header interpreters, etc.) or because they are used by two or more of the FAST3D programs and thus belong in no single one of the program source directories. More routines may from time to time be added to `lib/` as the functionality of FAST3D is expanded. Subroutines may also occasionally be removed when entirely different versions of what is currently a single routine have to be developed. As an example, perhaps VOYEUR and FAST3D need to implement separate versions of a single routine for some new capability.

Directory `fast3d_vno/` contains all the source files needed to run the parallel version of the FAST3D flow solver. FAST3D can also be run on a single workstation with these same subroutines. These subroutines are generally written in Fortran and are portable across a number of architectures and manufacturers. Specific issues of program portability are addressed later in Section 8.4. The specific subroutines in directory `fast3d_vno/` are described further in Sections 4.1 through 4.4.

Directory `post_proc/` is a directory reserved for various auxiliary diagnostic and post-processing programs to be used with the output files (dump/restart, cross-section, and station history files) generated by FAST3D. These output files may reside on one's local machine or on a disk connected to your parallel processing system. We have developed a few useful utilities for our problems and individual users will surely develop many others. The FAST3D documentation, however, does not go into detail on the use of these utilities at this time. Nevertheless, the contents of this directory are described further in Section 6.

Directory `runs/` is a directory containing the input data sets, object files, dump/restart files, executable files, and output files actually required to perform various FAST3D simulations, perhaps simultaneously. FAST3D uses dynamic memory allocation and is written so that it does

not have to be recompiled for different runs. Therefore this directory can, in principle, hold all of the run files. As a practical matter, a user should set up sub-directories for each project or different flow problem class identified by a unique probname. In Figure 1, a subdirectory is shown for each of the six test problem geometries being supplied with the released version of FAST3D. The contents of directory runs/are described further in Section 4.2.

Directory user_src/ contains subdirectories of each of the test problems where the source files for the three subroutines that a user may want to modify to specify a particular CFD problem. The three user-supplied files are: geometry.f_probname, flow_field.f_probname and flow_mods.f_probname. A version number is not appended to this source code directory.

Directory voyeur/ contains all of the necessary source files and libraries for the VOYEUR graphical diagnostics utility program. The program source files are within this directory in voyeur_vno/. The use of VOYEUR is described briefly below in Step 2.3. Its use is further expanded upon in Step 5.

I.7. General Data Input for FAST3D Programs

Each of the three main programs in the FAST3D suite requires somewhat different input data, although there is also significant overlap. The three problem-specific data files are to be located within a directory runs/probname/. GRIDVCE takes as an input the file probname.gi, VOYEUR reads probname.vi, and FAST3D reads probname.ni. These files are all formatted identically, as a set of distinct Fortran NAMELIST data input fields. The explanation in any good Fortran manual plus the examples given in this manual should be adequate to explain the use of NAMELIST. The Fortran NAMELIST input format is used in each of these data files because it provides a very readable and convenient free format to input and change data for a run. Any value from the list of names that is not specified automatically is given its default value. If a variable name is specified two or more times, the final value is used. Thus a record can be kept of a series of tests by leaving the earlier values of the input variables in the input data file. All of the input variables for the FAST3D system are collected in Appendices A, B and C for ready reference, organized by the NAMELIST in which they are read. FAST3D's specific use of NAMELIST is further discussed in Appendix A.

Each of the three input data files contains non-overlapping data input, although the data may be used by more than one program. For example, when FAST3D is being asked to self-initialize a simple problem geometry without first running GRIDVCE, information such as the cell spacing and cell interface locations in the X-, Y- and Z-directions is not available from an initial restart/dump file. The input data is in probname.gi; therefore, this file and data must be read by FAST3D and passed into the mesh generating subroutine mesh_set to generate a computational grid. Circumstances similar to these require replicating functionality in different programs and will be described below and in the appendices.

The set of NAMELISTs in each of the three data files that the user must construct to run the FAST3D suite of programs are now listed for each of the three main programs separately.

GRIDVCE - Grid Generator Input NAMELISTS:

GADMN_DAT	specifies the administrative information and data necessary to run GRIDVCE.
SIZES_DAT	conveys the information that determines the size of the FAST3D grid, e.g. the number of cells and the number and types of the convected variables.
GRID_DAT	conveys the information that defines the FAST3D computational domain, i.e. the coordinate system location and stretching, and the underlying structured grid specifications.
GEOM_DAT	conveys the data specifying the geometry of the bodies (if any) in the computational domain.

FAST3D - Flow Solver Input NAMELISTS:

FADMIN_DAT	administrative information and data necessary to run FAST3D.
BC_DAT	conveys all the boundary condition information needed by the FCT routines in the flow solver.
INIT_DAT	conveys material specifications and initial conditions for the flow field.
CHEM_DAT	conveys the chemistry species and reaction mechanism data.
FFIM_DAT	conveys flow field Init/Mods input record initializations. These data are user-supplied, problem-specific input needed to establish spatially dependent initial conditions and time and space dependent boundary conditions such as local source terms.
STD_DAT	conveys the set of station data locations, and other information needed to control the on-line collection of this data.

FAST3D also reads three of the namelists defined in probname. gi , SIZES_DAT, GRID_DAT, and GEOM_DAT.

VOYEUR - On-line Graphics Package Input NAMELISTS:

VADMIN_DAT	specifies the administrative information and data necessary to run VOYEUR.
XSEC_DAT	conveys the information that defines the cross-sections that FAST3D must output.

GLOBAL_DAT conveys the default data for all plots such as the number of pixels in the order, the overall magnification factors, etc. Each plot can separately override these values but this gives a convenient way to initialize a number of variables at once.

PLOT_DAT conveys the information that determines the physical variable and layout of each of the up to 100 plots permitted by VOYEUR.

CNTR_DAT conveys the data determining how each physical variable will be plotted, the contour levels to be used, the geometry of the bodies (if any) in the computational domain.

VOYEUR also reads three of the namelists defined in `probname.gi`, `SIZES_DAT`, `GRID_DAT`, and `GEOM_DAT`.

I.8. Running FAST3D Programs

The three codes associated with the FAST3D system, FAST3D, VOYEUR, and GRIDVCE, are run using scripts named `run_fast3d`, `run_voyeur`, `run_gridgen`, respectively. Each of these scripts lists the available options. `run_voyeur` typically takes an additional argument indicated by `-h xs_src_rmt` which is the remote file storage where the VOYEUR cross-sections are being or have been written by FAST3D.

The naming convention for the FAST3D suite of programs is that each executable program, i.e. `fast3d`, `gridvce`, and `voyeur` has an underscore, "_", and the problem name, `probname`, appended to the executable name. For example, if your particular `probname` is `blast`, then the FAST3D executable is `fast3d_blast`, the GRIDVCE executable is `gridvce_blast`, and the VOYEUR executable is `voyeur_blast`. Each of these programs must have been compiled for the specific computer on which it will be executed. In most instances the FAST3D flow solver will run on a different computer from GRIDVCE and VOYEUR, hence each executable is placed within a subdirectory of `$FAST3D/bin`. This subdirectory has the path `$FAST3D/bin/$target_arch`. In this subdirectory path, `$FAST3D` represents the path to the FAST3D system directory and `$target_arch` represents the machine architecture. Currently, the following computing architectures are supported: Intel iPSC/860 (I860), IBM with RISC Processor (RS6K), Silicon Graphics (SGI and SGI64) and the Sun 4 (SUN4). While the executable can be run directly, it is easiest to use one of the scripts below to execute the programs.

In order to facilitate the compilation, renaming, and execution of the FAST3D suite of programs, a set of Bourne shell scripts were written. These scripts are: `build_fast3d`, `make_new_run`, `run_fast3d`, `run_gridvce` and `run_voyeur`. The `build_fast3d` script is used to compile FAST3D, VOYEUR, or GRIDVCE, creating the correct executable with the appropriate `probname` extension and placing this executable into the appropriate `$FAST3D/bin/$target_arch` directory. The `make_new_run` script creates a new `probname` directory with new `probname.ni`, `probname.gi`, and `probname.vi` files or creates a set of new `probname_ext` data input files for the new subproblem, identified by the extension `_ext`, within an already existing `probname` directory. The `run_fast3d` script submits the FAST3D flow

solver executable onto the specified System Resource Manager, SRM, in the iPSC860 computer chosen. The `run_gridvce` script runs the appropriately compiled GRIDVCE code, based on the `probname_ext` argument, and optionally saves the run into a log file. The `run_voyeur` script runs the appropriately compiled VOYEUR code, based on the `probname_ext` argument, and optionally saves the run into a log file. Each of these scripts will be discussed in more detail in later sections. A brief description of these scripts can be obtained on-line by typing the script name at the command prompt, since no arguments are provided the default is the help facility .

Step 1. How to Define a CFD Simulation for FAST3D

What constitutes a properly posed CFD simulation? A full discussion of this question is found in Oran and Boris, *Numerical Simulation of Reactive Flow*, Chapter 3, (1987) and shall not be address in this manual

1.1. A Problem Template and Naming Conventions in FAST3D

In Section I.3 above, we identified six components of a complete CFD problem definition for FAST3D, the first three of the problem-defining components in the list, a, b and c are generally embodied in subroutines because of the variation and generality of the wide-range of problems FAST3D is intended to solve. Default (null) routines are provided and are usually sufficient for many simple cases. These can also serve as templates for a user graduating from the novice to the intermediate level and wanting to do something more complex. However, one could start from scratch to build these data sets for each new problem.

mknewprob

To simplify the creation of new run subdirectories and/or data files for new runs the script creates three new files (i.e. <>.ni, <>.vi <>.gi) based on the files from another run or directory. The command `mknewprob` can be used to either generate a set of data files in the current directory to make a new subproblem or to generate a new directory with the corresponding set of data file to make a new problem.

Help for `mknewprob` is obtained by typing the command with any options. For example,

```
% mknewprob
```

```
Usage: mknewprob newname ol dname
```

In particular, the script copies the files corresponding to the `oldname` problem directory or run and renames them to the `newname` directory or run and changes all references with the `*.i` files to the `newname`.

For example, if you are in directory `$FAST3D/runs/bluntbody` and you wish to duplicate the subproblem `bluntbody_2d`, renaming it to `bluntbody_3d`, simply type:

```
prompt> mknewprob bluntbody_3d bluntbody_2d
```

```
Creating bluntbody_3d.gi
Creating bluntbody_3d.ni
Creating bluntbody_3d.vi
** DONE **
```

Note that the directory name always corresponds to the part of the problem name preceding the `"_"`. Hence, although the full problem names are `bluntbody_2d`, `bluntbody_3d`, etc., they all

reside in the directory named \$FAST3D/runs/bluntbody. Also, the file paths defined in the *.ni and *.vi files, file_path and src_path respectively, reflect the name of the problem subdirectory to runs/, not the particular run name.

Similarly, one can use the script makefastdir to create new directories:

```
prompt> makefastdir
Usage: makefastdir [oldname] [newname] ['all']
```

1.2. Choosing a FAST3D Problem for Execution

Once the /runs/problem_name directory exists and the corresponding geometry.f_probname, flow_field.f_probname and flow_mods.f_probname have been specified in the /user_src directory, choosing a problem is as simple as changing the directory to the corresponding runs directory. For example:

```
dax% cd ~/fast3d/runs/problem_name
dax% run_fast3d -p 4 problem_name
```

1.3. Multiple FAST3D Problems Active Simultaneously

It is often desirable to have several problems, or even different runs of the same problem, active simultaneously. This usually is done to speed up results, to facilitate on-line comparisons of two cases, or to take advantage of available computer resources. You do not want to have two instances of the same run submitted at the same time since files will overwrite one another. In order to have two similar subproblems running at the same time, you want to give them a distinct runname, this runname identifies the run such as

```
blast_1da
blast_1db
blast_1d50
```

Each of these runs has uses the same geometry.f_blast, flow_field.f_blast and flow_mods.f_blast files but differs in the data input in the *.gi and/or *.ni files. These three files could be running on the same system or different systems. Since each run has a unique name, one can easily access the log files that are generated or use VOYEUR to view the simulation.

In theory one could run the same subproblem, i.e. blast_1da, on two different systems as long as the output files are stored on separate file systems. However, this tends to cause confusion since both subproblems have the same name.

Step 2. How to Specify and Check a Geometry for FAST3D

The scalable FAST3D suite of CFD programs achieves its complex geometry capability using a Virtual Cell Embedding grid constructed on a underlying globally structured rectilinear mesh. The only cells in the domain that are specially treated and require additional storage are those adjacent to a body. We have adopted the designation 'feature' cells for these cells. The number of these cells is generally proportional to the surface area of the bodies being represented and is therefore much smaller than the total number of cells in most three-dimensional domains. Only the 'feature' cells require additional storage for boundary flux terms, for actual interface areas, and for reduced cell volumes. The actual VCE data structure implemented in FAST3D is described in Section 8.3.

FAST3D, GRIDVCE, and VOYEUR require only the simplest information about any geometry to function properly. An accurate treatment of curved and non-grid aligned geometries of arbitrary complexity can be constructed automatically provided the user can answer the one simple question: "Is a given point x, y, z inside one of the solid bodies or is it in the fluid?" for every point of the computational domain. This is a different approach from other grid generators and lends itself to very compact geometry definitions in most cases. Needing to know the intersection curve of two complex curved surfaces, for example, is not necessary. Complex geometries can be built up easily as the union of a number of simple, overlapping shapes. This approach also permits implementation of proprietary geometries by third parties. A subroutine answering the single question above can be compiled and supplied in object form. No formulae, curves, or CAD descriptions need be supplied.

In this release of FAST3D, it is assumed that all bodies and walls bounding the computational flow domain are rigid with no thermal conduction or porosity and are either an ideal wall (`solid_bc = 1`) or extract some amount of tangential momentum from the first cell next to the wall (`solid_bc = 2`). This information is supplied in `NAMELIST /GRID_DAT/` as described in Section 3.1 and Appendix A.

2.1. Coding the User-Supplied Geometry Subroutine `geometry.f`

Each new type of geometry must have its associated subroutine `geometry.f_probname`. This subroutine's sole function is the answer the question "Is the given point xt, yt, zt (the argument list variables) inside one of the solid bodies or is it in the fluid?" for the particular class of geometries associated with `probname`. By repeated calls to this subroutine GRIDVCE builds a high resolution VCE data base for use by FAST3D in its output file `probname.geo0`. In particularly simple circumstances, such as when all boundaries are straight and grid aligned or all curved bodies can have a rough "staircased" surface, FAST3D can self initialize without requiring a grid generation step at all. `geometry.f_probname` is also used by VOYEUR to construct masks showing the location of all bodies in the chosen cross-section for plotting. The resolution for these three different uses of `geometry.f_probname` can all be different, the subroutine does not need to be changed to generate different grids or to run many different subproblems.

The following example listing of directory `/user_src` shows the three subroutines that define a problem class.

```

fozzi e% pwd
/u/crfdslandsberg/fast3d/user_src
fozzi e% ls -l *
-rw-r--r--  1 landsber      1205 Sep 17 14:09 Dependenci es. mk

bluntbody:
total 7
-rw-r--r--  1 landsber      2329 Sep 17 14:09 flow_fi el d. f_bluntbody
-rw-r--r--  1 landsber      1374 Nov 13 16:01 geometry. f_bluntbody

muzzle:
total 7
-rw-r--r--  1 landsber      2344 Oct  9 14:28 flow_fi el d. f_muzzl e
-rw-r--r--  1 landsber      1465 Nov  8 09:54 geometry. f_muzzl e

null:
total 4
-rw-r--r--  1 landsber        265 Sep 17 15:32 f3d_user. h_null
-rw-r--r--  1 landsber        321 Sep 17 15:32 flow_mods. f_null
-rw-r--r--  1 landsber      1309 Sep 17 15:32 geometry. f_null

```

The extension appended via an underscore at the end of each `geometry.f` name is the user-prescribed problem name as described above. It defines a particular generic problem geometry, e.g. `bluntbody` or `muzzle`.

Each of these geometry routines is written by the user in the format supplied with the given input argument list. Since the geometry of a problem is your choice, we cannot specify it for you. The programming needed to write this geometry routine has been made as simple as possible, however, consistent with being able to handle the general case of very complex geometry. Further, the example `geometry.f_null` routine described in this documentation is a ‘skeleton’ or ‘template’ program that FAST3D users can take as a starting point. It contains all the generic variable declarations, and provides the simplest structure we have found for a wide class of geometry specifications.

The calling program, GRIDVCE, FAST3D, or VOYEUR, must be able to determine from the coordinates of a point in three-dimensional space whether that point lies in the fluid being integrated (for which `geometry.f` returns a value of 0 or -1 for `Ftr_geom`) or whether it lies in one of the solid bodies bounding the domain (`Ftr_geom = 1` returned).

Blunt Body Geometry Subroutine Example: Here our version of `geometry.f_bluntbody` is used as an example. Following immediately is a working geometry subroutine that implements a particularly simple blunt body geometry, one of the test problem geometries distributed with the FAST3D program suite. This subroutine establishes the geometry for a 3D rectangular blunt body in the flow defined by $X_{wall_1} \leq X \leq X_{wall_2}$, $Y_{wall_1} \leq Y \leq Y_{wall_2}$, and $Z_{wall_1} \leq Z \leq Z_{wall_2}$. Into this solid, grid-direction aligned, six-sided ‘rectangle’ is cut a corresponding ‘rectangular’ notch defined by $X_{notch_1} \leq X \leq X_{notch_2}$, $Y_{notch_1} \leq Y \leq Y_{notch_2}$, and $Z_{notch_1} \leq Z \leq Z_{notch_2}$. The volume of the notch is set back to being in the fluid. The notch can be anywhere in, around, or adjacent to the solid body. It can even be an

enclosed hole in the interior or a rectangular hole right through the body. Any one of these possibilities could be dealt with easily by most grid generators but different grid structures would have to be specified and initialized for the distinct cases.

2.2. Compiling and Linking with FAST3D Programs: `build_fast3d`

The three main FAST3D programs and their support subroutines use dynamic memory allocation for variable length arrays so they generally do not need to be recompiled and linked. The user-provided subroutines for geometry, flow initialization, and flow modification will have to be compiled and linked whenever changes are made to these subroutines. For this purpose, and whenever one of the programs has to be re-made, a script called `build_fast3d` has been written as an simple user interface to the appropriate makefile.

Help for `build_fast3d` is obtained by typing the command with no arguments or by using the "-H" option. For example,

```
% build_fast3d
```

```
Usage: /u/landsber/fast3d_beta_1.1/bin/build_fast3d [-fF] [-gG] [-vV] [-cC]
[-uU] [-ot] prob_name
```

```
-f/F: Update[f]/Rebuild[F] FAST3D object library
-g/G: Update[g]/Rebuild[G] GRIDVCE object library
-v/V: Update[v]/Rebuild[V] VOYEUR object library

-c/C: Update[c]/Rebuild[C] COMMON object library
-u/U: Update[u]/Rebuild[U] USER prob_name dependent library

-o: Site/Organization name? Default: [CEWES]
-t: Target Architecture? Default: [RS6K]
```

To compile and link the grid generator GRIDVCE with the new or modified geometry subroutine type:

```
% build_fast3d -g bluntbody
```

To compile and link FAST3D with the new or modified geometry subroutine type:

```
% build_fast3d -f bluntbody
```

To compile and link VOYEUR with the new or modified geometry subroutine type:

```
% build_fast3d -v bluntbody
```

We will return to these commands in Steps 3, 4, 5, giving copies of the output and explaining what is being done in each case.

`Buid_fast3d` will build the corresponding executable for the specific architecture. For example, if `GRIDVCE` and `VOYEUR` are built on `henson` (SGI64) for the `bluntbody` problem then the executables will reside and be named as follows:

```
~/fast3d/bin/SGI64/gridvce_bluntbody
~/fast3d/bin/SGI64/voyeur_bluntbody
```

Similarly, if these same problems are built on `oscar` (RS6K) they will reside and be named as follows:

```
~/fast3d/bin/RS6K/gridvce_bluntbody
~/fast3d/bin/RS6K/voyeur_bluntbody
```

Finally, FAST3D can be built on `fizzie`, to be run on the Intels (I860), or built on `henson` (SGI64), corresponding to the following file paths and names:

```
~/fast3d/bin/I860/fast3d_bluntbody
~/fast3d/bin/SGI64/fast3d_bluntbody
```

2.3. Specifying Input Data to `geometry.f`

The generic `bluntbody` test problems defined above will also be used, later in the documentation, to illustrate the use of two other FAST3D features, the user-supplied, flow field initializing subroutine `flowfield.f_bluntbody` and the `flow_mods.f_bluntbody` subroutine that modifies the flow field during a simulation. In the version of these routines documented with this FAST3D release, the problem-specific data are compiled directly into the subroutines, giving very simple example programs. In the FAST3D input file `bluntbody_3d.gi`, reproduced immediately below, a namelist `GEOM_DAT` has been inserted containing the same information as hardwired into the `bluntbody` geometry subroutine above. This namelist read in overrides the data in the subroutine. Being able to read this information removes the necessity of recompiling every time parameters of the geometry are changed.

```
fizzie% cat bluntbody_3d.gi
$GADMN_DAT
    Run_name = 'bluntbody_3d',          GADMN_echo = T,
    . . .
$END
$SIZES_DAT
    Run_name = 'bluntbody_3d',          SIZES_echo = T,
    . . .
$END
$DOMAIN_DAT
    Run_name = 'bluntbody_3d',          DOMAIN_echo = T,
    . . .
$END
```

```

SGRID_DAT
    Run_name = 'bluntbody_3d',
    . . .
SEND
SGEOM_DAT
    runname = 'bluntbody_3d',
    GEOM_echo = .TRUE.,
    Geom_rec(1) = 'Xwall_1 = -16.0, Xwall_2 = 0.0',
    Geom_rec(2) = 'Ywall_1 = -8.0, Ywall_2 = 8.0',
    Geom_rec(3) = 'Zwall_1 = -24.0, Zwall_2 = 24.0',
    Geom_rec(4) = 'Xnotch_1 = -4.0, Xnotch_2 = 1.0',
    Geom_rec(5) = 'Ynotch_1 = -9.0, Ynotch_2 = 9.0',
    Geom_rec(6) = 'Znotch_1 = -4.0, Znotch_2 = 4.0',
SEND

```

Note that for the data files defining a problem, as opposed to the subroutines associated with a class of problems, an extension (suffix) has been added to the problem name, in this case `_3d`, to distinguish this particular subproblem. Another of the test cases is `bluntbody_2d`. There is clearly an advantage to reading the data into `geometry.f`, allowing one compiled subroutine to service a number of active subproblems simultaneously. This capability must deal with the difficulty of getting the data down to each of the processors; however, this has been addressed in the current version of FAST3D.

2.4. Running VOYEUR to Check the Geometry

VOYEUR is designed to be run asynchronously in support of FAST3D and is currently not a scalable program. It should be run on a moderate to powerful workstation where the user has sufficient interactive access to coordinate VOYEUR execution with the execution of FAST3D on the large, scalable processor. VOYEUR reads files containing selected flow cross-sections that are written periodically by FAST3D and plots them independently of the ongoing simulation. VOYEUR currently runs with 8-bit color X-windows.

The VOYEUR graphics package is designed as a simple graphical adjunct to the scalable FAST3D program. It allows interpolated, color-scale contour plots of geometry and flow field cross-sections extracted from the FAST3D program as it is executing. It recognizes the `geometry.f` routines and the VCE mesh and grid data structures generated by `GRIDVCE` and used by FAST3D. Here only the extremely simple use of VOYEUR to inspect specified slices through the newly determined geometry is described. Detailed reading of Section 5 (Step 5) will give a much broader knowledge of the range of features in VOYEUR. Modification of the VOYEUR programs by the user is seldom needed or desired.

Once the abbreviated data files are ready for viewing the geometry, running VOYEUR is generally done from a high performance.

To check the geometry only (no flow field), set `L_justgeo = .True.` in `probname.vi` and then run `voyeur: dax% run_voyeur -i probname.`

Step 3. Running GRIDVCE to Generate a FAST3D Grid

The Role of the Generated Grid in FAST3D

GRIDVCE is a Fortran program to generate the VCE complex geometry grid for executing FAST3D. The flow solver FAST3D can generate a Cartesian mesh with complex geometry; however, the resolution around the body will be staircased, i.e. only the cell center will be checked whether it is inside or outside of the body. To generate a grid with VCE or feature cells requires running GRIDVCE, typically performed on a high-performance workstation. GRIDVCE not only produces a Cartesian mesh, but computes and writes out the reduced areas and volumes of the feature cells, marks feature cells that are exceeding small to prevent numerical instabilities, and generates a list of the starting and stopping indices of the integration line segments. If the flow solver is being run on a remote parallel system, the binary output geometry file (*.geo0) from GRIDVCE should then be transferred to the parallel computer's disk. The *.geo0 file is read on initialization, at subsequent restarts from a dump file (*.dmp1 or *.dmp2) all of the grid information from the *.geo0 file is stored in the dump files.

3.1. Specifying Input Data to GRIDVCE

The GRIDVCE Input Data File 'runname.gi'

Reproduced directly below is a GRIDVCE input data file bluntbody_3d.gi as it would appear to generate the VCE grid for the test problem of the 3-D blunt body. This example continues the development of the particular case used in the previous section.

```
&GADMIN_DAT
  runname = 'bluntbody_3d',
  GADMIN_echo = T,
  geostep = 1,           gf_bytswp = F,
  arch = 'MOST',
  arch = 'RS6K',
/
&SIZES_DAT
  runname = 'bluntbody_3d',
  SIZES_echo = .TRUE.,
  Nci = 256,             Ncj = 96,           Nck = 48,
  Npv = 5,              Ncv = 0,           Nsv = 1,
  Nfv = 2,              N_vn = 64,        Mfdrs = 0,
/
&GEOM_DAT
  runname = 'bluntbody_3d',
  GEOM_echo = .TRUE.,
  Geom_rec(1) = 'Xwall_1 = -16.0, Xwall_2 = 0.0',
  Geom_rec(2) = 'Ywall_1 = -8.0, Ywall_2 = 8.0',
  Geom_rec(3) = 'Zwall_1 = -24.0, Zwall_2 = 24.0',
  Geom_rec(4) = 'Xnotch_1 = -4.0, Xnotch_2 = 1.0',
  Geom_rec(5) = 'Ynotch_1 = -9.0, Ynotch_2 = 9.0',
  Geom_rec(6) = 'Znotch_1 = -4.0, Znotch_2 = 4.0',
/
&GRID_DAT
```

```

runname = 'bluntbody_3d',
GRID_echo = .TRUE.,
alpha =      1,          1,          1,
deltah =     1.00,      1.00,      1.00,
centrh =     0.0,       0.0,       0.0,
uniform_g =  .TRUE.,   .TRUE.,   .TRUE.,
P_bc1 =      4,         1,         1,
P_bcN =      7,         1,         1,
Lsleft =    10000,     10000,     10000,
Lscent =     16,        2,         2,
Lscnc =     256,       96,         48,
Lsri gh =    16,        4,         8,
hsl eft =    0.10,     0.00,     0.10,
hsri gh =    0.05,     0.08,     0.10,
nsubv = 16,          nsuba = 32,          solid_bc = 1,

```

Note: These namelists are in Fortran 90 format for the IBM SP.

There are four active namelists in this file that are needed to launch the GRIDVCE program and its main grid generator subroutine `initalg.f`. The data included in each of these namelists serve a different function to the grid generator program as described briefly below and in expanded form in Appendix A. Definition and Initialization of `runname.gi` Variables.

The first of the namelists read is `GADMN_DAT`. It sets the architecture dependent variables.

The second of the namelists read is `SIZES_DAT`. It sets all the pertinent sizes for running the grid generator including the number of computational cells in each direction, gives the number of physical variables, chemistry species, special variables, and geometric feature variables. These values will subsequently be read and used in the flow solver.

The third of the namelists read is `GEOM_DAT`. In this namelist, the variables, their definitions, uses, defaults, etc. are defined by the user. In this example, the bounds of the rectangular wall and notch are being specified.

The fourth of the namelists read is `GRID_DAT`. It sets the type of grid (Cartesian, cylindrical, or spherical) in each direction, the grid spacing, whether the spacing is uniform, the boundary conditions on the computational volume, grid stretching information and the number of subdivisions on the face areas and volumes.

3.2. Running GRIDVCE

`run_gridvce` is a script that runs the FAST3D grid generation program, GRIDVCE, either interactively or in the background. The executable program actually submitted is `gridvce_probname` where `probname` is the generic problem name which determines the specific user-supplied routine `geometry.f`. The `build_fast3d` script, described above, generates this executable program, which once compiled, resides in the `SFAST3D/bin/Target_arch` directory. The last argument to the script is the specific run or subproblem name which determines the names

of the input files to be used in the directory \$FAST3D/runs/probname. Help for the run_gridvce script is obtained typing the command with no arguments or by using the "-H" option:

```
osprey4% run_gridvce
```

Usage: run_gridvce [-acei] run_name

- a: Append the log file to a previous log file
- c: DO NOT cd to the appropriate runs directory based on run_name
- c: cd to the appropriate directory based on run_name
- e: Send standard error to the log file
- i: Run GRIDVCE interactively (i.e. not in the background)

The only required option to this script is the run_name. If the command is run in the background, then output from the run_gridvce script and output of the GRIDVCE program is put into the file runname.glog. This file does not actually contain the grid data, but contains information such as input data and number of feature cells detected. This file may be viewed by using the Unix tail command or any editor. If the command is run interactively, then this information is written to the screen.

3.3. Monitoring and Checking the Resulting FAST3D Grid

The file runname.glog or output to the screen provides information about input data to the grid generator and the number of feature cells detected in a block. GRIDVCE also outputs a text format file bluntbody_3d.grid that contains the physical locations of the indices. If the grid is uniform, for each integration direction the extent of the boundaries is shown, followed by the extent of the cell centers, followed by the grid spacing. If the grid is non-uniform, then for each integration direction, the file will show the indices left boundary, cell center and cell spacing. Currently, there is no way to visualize the grid "lines", one can visualize the extent of the boundaries with geometry using VOYEUR as described in Section 2.4.

3.4. Transferring the FAST3D Grid File

As mentioned above, GRIDVCE typically is run on a high-performance workstation. There will be occasion when the *.geo0 file needs to be transferred to another computer system, i.e. the system on which the flow solver will be run. The *.geo0 file is a binary file, when using ftp, remember to turn the file transfer mode to binary. If binary mode is not set, the file will still be transferred but errors will occur when this file is read by the flow solver. The FAST3D grid file can be transferred to any machine by simply ftp-ing to the desired machine. On many of the HPC platforms, the *.geo0 file will be stored on a scratch or temp disk along with the dump files generated by the flow solver. In this case, the file can simply be moved to the correct disk and directory.

Step 4. Conducting CFD Simulations Using FAST3D

Operation of the flow solver FAST3D

The FAST3D flow solver is data driven by NAMELISTS in the *.gi and *.ni files. In addition, user-supplied subroutines specify the geometry, flow field and flow modifications. The 6 test cases provide the user-supplied subroutines from which the intermediate and advanced user can develop new problems. If the user changes these files, it is the user's responsibility to modify these files correctly, i.e. compile, execute and produce accurate results. However, for the novice and intermediate user the set of test cases and demonstration problems are an excellent starting point for understanding how the FAST3D flow solver executes.

1. Specifying Input Data to FAST3D

All of the input data to the FAST3D flow solver is specified in initialization files. The bluntbody_3d test case will be used for demonstration purposes. The bluntbody_3d.ni file is shown here:

```
$FADMIN_DAT
  runname = 'bluntbody_3d' ,
  FADMIN_echo = .TRUE. ,
  arch = 'SGI64' ,
  file_pth = '/tmp2/young/fast3d/bluntbody/' ,
  file_prfx = 'bluntbody_3d' ,
  dump_freq = 200, 200, 1000, 1000, 0,
  hist_freq = 2, 50, 400, 20001, 30001,
  spot_freq = 0, 0, 0, 1, 0,
  pix_freq = 10, 0, 0, 0, 0,
  Opt_cfix = 0, 0, 0, 0, 0,
  note_freq = 50,
  Minstep = 7201,
  deltat = 5.0e-6,
  CFL_user = .250,
  L_upstr = .false. ,
  L_chem = .false. ,
  sim_desc = 'Flow over Blunt Wall: 256x96x48 ' ,
  L_osfn = .FALSE. ,
  Dmp0_flg = .FALSE. ,
  L_hist = .FALSE. ,
  L_spot = .FALSE. ,
  L_video = .FALSE. ,
  cnsvr_freq = 500,
  Maxstep = 15001,
  dtmax = 5.0e-6,
  dtmin = 5.0e-6,
  FCT_adiff = 0.998,
  L_antidif = .false. ,
  L_ffim = .False. ,
  L_geom = .True. ,
$SEND
$INIT_DAT
  runname = 'bluntbody_3d' ,
  INIT_echo = .TRUE. ,
  gamma0 = 1.4,
  vinit = 10000.0, 0.0, 0.0,
  vamb = 10000.0, 0.0, 0.0,
  grav = 0.0, 0.0, 0.0,
  rhoinit = 1.226-3, preinit = 1.01325e+6,
  rhoamb = 1.226-3, preamb = 1.01325e+6,
```

```

        rhomin = 1. 226- 4,      premi n = 1. 01325e+5,
        rhomax = 1. 226- 2,      premax  = 1. 01325e+7,
$SEND
$SBC_DAT
    runname = 'bluntbody_3d' ,
    BC_echo = . TRUE. ,
    rhobc = 1. 0,      1. 0,      1. 0,      1. 226e- 3,      1. 226e- 3,      1. 226e- 3,
    prebc = 1. 0,      1. 0,      1. 0,      1. 01325e+6,  1. 01325e+6,  1. 01325e+6,
    velbc = 0. 0,      0. 0,      0. 0,      10000. 0,      0. 0,      0. 0,
            0. 0,      0. 0,      0. 0,      0. 0,      0. 0,      0. 0,
            0. 0,      0. 0,      0. 0,      0. 0,      0. 0,      0. 0,
    svbc = 0. 0,      0. 0,      0. 0,      1. 0,      0. 0,      0. 0,
    epsbc = 0. 0,      0. 0,      0. 0,      0. 0,      0. 0,      0. 0,
$SEND
$STD_dat
    runname = 'bluntbody_3d' ,
    STD_echo = . FALSE. ,
    xs_pool = 1014, 1064, 1150,
            2048,
            3024,
$SEND

```

These NAMELISTS completely specify the flow conditions for the problem. In addition, the grid information from the `bluntbody_3d.gi` file is also needed in the case of self-initialization. To self-initialize the grid (this does not use VCE), one sets `Mi nstep = 0` while to read the GRIDVCE output file one sets `Mi nstep = 1`. The novice and intermediate user can modify the flow conditions or changes boundary conditions solely through the `*.ni` file without the need for recompilation. The user is now ready to run the FAST3D flow solver.

2. Running the FAST3D Flow Solver

The `run_fast3d` script is provided to the user to submit jobs on parallel systems without having to type in complex MPI commands. The novice user can easily use this script to run the flow solver given that the software has been installed on the parallel platform correctly. The following is a comprehensive description of the `run_fast3d` script.

run_fast3d

The `run_fast3d` script runs the appropriate flow solver for a specific problem, `fast3d_probname`. The `build_fast` script, described above, generates this executable program, which once compiled, resides in the `$FAST3D/bin/$target_arch` directory. The last argument to the script is the specific run or subproblem name which determines the names of the input files to be used in the directory `$FAST3D/runs/probname`. By typing `run_fast3d` at the prompt, with no arguments, one gets the usage:

```
%> run_fast3d
```

Usage: `run_fast3d [-acehijpnsx] run_name`

- a: Append the log file to a previous log file
- c: DO NOT cd to the appropriate run_name directory
- e: Send standard error to the log file
- h: System name (Default: jim-henson)
- i: Run FAST3D interactively (i.e. not in the background)
- j: Add run information to the journal (\$FAST3D/runs/run_name.jnl)
- p: Number of Processors (Default: [1])
- n: npri value, if required
- s: Submit when processors are available
- x: Use the following executable name, fast3d_[name]

The only required options to this script are -p and run_name. If, however, the job is being submitted to the intels (iPSC/860) from the frontend (lcp), then the -h option must be used. This option is used to determine which SRM to run the job on. The other two options indicate how many processors to run the job on and run_name of the job you wish to run. For example, if you wish to run the blast_2d casewith 16 processors on the iPSC/860 with SRM name wonka you would type:

```
% run_fast3d -h cerberus -p 16 blast_2d
```

This command would then submit the FAST3D executive code `fast3d_blast` to the iPSC/860 with SRM name wonka on 16 nodes and use the input files `blast_2d.gi`, `blast_2d.ni` and `blast_2d.vi`.

Occasionally, it may be desirable to run an executable which is different than the *probnam*e. For example, if you are in *blast* directory and wish to submit a job using the executable for the *blntbdy* problem simply the "-x" option for run_fast3d . For this example one simply types

```
%> run_fast3d -h cerberus -p 16 -x blntbdy blast_2d
```

This command would then submit the FAST3D executable code `fast3d_blntbdy` to the iPSC/860 with SRM name wonka on 16 nodes and use the input files `blast_2d.ni` and `blast_2d.vi`.

All output of the **run_fast3d** script, which includes output of the FAST3D program, is put into the file *runname.flog*. This file may be viewed by using the unix tail command or any editor.

The "-j" option will add job information (time of job submission, job name, run name, etc.) into a journal file with name *runname.jnl* within the current *probnam*e directory. This file is simply of journal of what jobs have been submitted and may be useful since it summarizes the runs completed.

3. Monitoring FAST3D Runs During Execution

CFD, particularly when used as a research tool, is very much an empirical science. Just as state-of-the-art experiments with brand new technology should be monitored in the laboratory, runs of **fast3d**, particularly with new physics and/or geometry, should be monitored. There are two mechanisms in place to monitor a FAST3D run: the *.flog file and VOYEUR. As the run is progressing, diagnostics such as timestep, time, CFL number, and conservation of the state variables are written to the output file, *.flog (flow solver log file), negative densities, negative pressures, etc. On systems that flush the buffer immediately, this file is updated every few seconds (the user can determine the rate of the output) so that one can easily determine if the solution has gone unstable or is progressing as expected. The *.flog file is not intended to contain an excessive amount of information about the flow solution, just the minimal information necessary to determine how the code is running.

A more quantitative method to monitor the advancing FAST3D flow solution is to use the graphics package VOYEUR, supplied with the FAST3D package. VOYEUR is discussed in Step 5. Essentially, one can view the primary variables and derived variables using two-dimensional cross-sections with VOYEUR. From these plots, the user is able to determine if the flow solution is advancing as expected.

4. Killing the FAST3D Flow Solver (KOFAST3D)

There are two ‘obvious’ methods to kill the FAST3D flow solver. The first one is the brute force method. Depending on the system and whether a job is running interactively or in batch mode, the user would kill the FAST3D flow solver using ‘^C’ (Unix interrupt) in interactive mode and qdel jobnumber on a batch system (or similar command). These would instantaneously halt execution of the flow solver. A more graceful way to exit the program is to wait for the next dump/restart file to be written and then exit. Therefore, run time is not lost by killing the job in the middle of a dump or just shortly before a dump file is written. The command to exit after the dump file has been written is KOFAST3D.

5. Restarting FAST3D for Continuation Runs

Obviously, the user would like to restart runs to further advance the flow solution. One simply looks in the *.flog file for the last completed dump and then specifies that `MINSTEP = dump#` in the *.ni file. It should be noted that `MINSTEP = dump#` usually ends with a 1, such as `MINSTEP = 1001`.

Step 5. Using VOYEUR to Visualize the Computed Flow

An important feature of the FAST3D program suite is the ability to watch selected flow visualization cross-sections produced from the simulation while the calculation is underway on a large, parallel processing computer. A program called VOYEUR has been written to visualize the flow variables from FAST3D on several selected cross-sections while the simulation is proceeding.

Since this useful 'peeping-Tom' capability is quite architecture- and network-dependent, its smooth attachment to FAST3D cannot presently be guaranteed under all circumstances.

1. Specifying Input Data to VOYEUR

B. The Full 'voyeur' Input Data File 'Run_Name.vi'

```
SVADMIN_DAT
  runname = 'bluntbody_3d',
  VADMIN_echo = .true.,
  arch = 'SGI64',
  L_justgeo = .True.,      L_outline = .true.,
  L_justgeo = .false.,    L_outline = .true.,
  sleep_time = 10,        L_osfn = .false.,      keep_tfs = .false.,
  xsec_fn = 'bluntbody_3d',
  src_path = '/cfs/ccpd/young/fast3d/bluntbody/',
  dst_path = '/tmp1/young/fast3d/',
  dst_path = '/tmp2/young/fast3d/',
  mn_frm = 1,             mx_frm = 50000,          mx_try = 1000,
  L_video = .false.,

SEND
$VIDEO_DAT
  runname = 'bluntbody_2d',
  VIDEO_echo = .true.,
  L_video = .false.,     vid_freq = 1,
  vid_fram = 3,          vid_time = '00180000',
  L_psync = .false.,     Lpn = 0,              Np = 1,

SEND
$GLBL_DAT
!cp% cat bluntbody_3d.vi
SVADMIN_DAT
  runname = 'bluntbody_3d',
  VADMIN_echo = .true.,
  arch = 'SGI64',
  L_justgeo = .True.,      L_outline = .true.,
  L_justgeo = .false.,    L_outline = .true.,
  sleep_time = 10,        L_osfn = .false.,      keep_tfs = .false.,
  xsec_fn = 'bluntbody_3d',
  src_path = '/cfs/ccpd/young/fast3d/bluntbody/,'
```

```

dst_path = '/tmp1/young/fast3d/',
dst_path = '/tmp2/young/fast3d/',
mn_frm = 1,          mx_frm = 50000,          mx_try = 1000,
L_vide o = .false.,
$SEND
$VIDEO_DAT
  runname = 'bluntbody_2d',
  VIDEO_echo = .true.,
  L_vide o = .false.,          vid_freq = 1,
  vid_fram = 3,          vid_time = '00180000',
  L_psync = .false.,          Lpn = 0,          Np = 1,
$SEND
$GLBL_DAT
  runname = 'bluntbody_3d',
  GLBL_echo = .false,
  Macropx = 2, 2, 2,
  Macropx = 1, 1, 1,
  NMppGrid = 0,          NPixBorder = 1,
  cbr_wid th = 20,          cbr_leng th = 151,
  Ti c_wid th = 2,          Ti c_leng th = 6,
  pxmx = 248,
$SEND
$XSEC_DAT
  runname = 'bluntbody_3d',
  XSEC_echo = .false,
  i j k_pp = 30*0,
  i j k_pp = 1014, 1064, 1150,
          2048, 3024,
  xyz_pp = 30*0.0,
$SEND
          2016, 2032, 2048,
          3016, 3032, 3048,
$PLOT_DAT
  runname = 'bluntbody_3d',
  PLOT_echo = false.,
  pl t_i rec =
  ' Ptyp, Pf, Xsec, Sp, Zh, Zv, Ph1, PhN, Pv1, PvN, Wl h, Wl v',
  ' Vy, 1, 1014, F, 2, 0, 0, 0, 0, 0, 300, 5',
  ' Vz, 1, 1014, F, 2, 0, 0, 0, 0, 0, 300, 140',
  ' Vy, 1, 1064, F, 2, 0, 0, 0, 0, 0, 300, 275',
  ' Vz, 1, 1064, F, 2, 0, 0, 0, 0, 0, 300, 415',
  ' Vz, 0, 1150, F, 0, 0, 0, 0, 0, 0, 300, 575',
  ' Tv01, 1, 3024, F, 0, 0, 0, 0, 0, 0, 5, 5',
  ' nVx, 1, 3024, F, 0, 0, 0, 0, 0, 0, 5, 140',
  ' Vy, 1, 3024, F, 0, 0, 0, 0, 0, 0, 5, 275',
  ' Tv01, 1, 2048, F, 0, 2, 0, 0, 0, 0, 5, 415',

```

```

' Vz,      1,  2048,  F,  0,  2,  0,  0,  0,  0,  5,  555',
' Tv01,   0,  3013,  F,  0,  0,  0,  0,  0,  0,  5,  255',
' Tv01,   0,  2024,  F,  0,  0,  0,  0,  0,  0,  5,  680',

```

SEND

\$CNTR_DAT

```

runname = 'bluntbody_3d',
CNTR_echo = .False,
ctr_i rec =
' Ptyp,      Cmin,      Ccent,      Cmax,  Prcnt,  Mde,  Cbi,  Clh,  Clv',
' Vx,      -1.0E+4,  0.0E+1,  1.0E+4,  4.0,   1,   0, 1000, 1625',
' nVx,     -1.5E+4,  0.0E+1,  6.0E+3,  2.0,   1,   0, 1000, 1625',
' Vy,      -5.0E+3,  0.0E+1,  5.0E+3,  2.0,   1,   0, 1000, 1625',
' Nrho,     0.85,    1.000,    1.02,   5.0,   1,   0, 1000, 1625',
' Vz,      -5.0E+3,  0.0E+0,  5.0E+3,  2.0,   1,   0, 1000, 1625',
' rho,     1.1e-3,  1.226E-3,  1.25E-3,  1.0,   1,   0, 1000, 1625',
' erg,     1.0E+5,  1.0E+6,  5.1E+6,  1.0,   2,   0, 100,  400',
' Pre,     0.92E+6,  1.013E+6,  1.05E+6,  2.0,   1,   0, 1000, 1625',
' Npre,     0.5,    1.0,     1.03,   2.0,   1,   0, 1000, 1625',
' Tem,     1.0E+2,  1.0E+3,  3.0E+3,  2.0,   1,   0, 100,  400',
' Cs,      0.5E+4,  3.4E+4,  1.0E+5,  1.0,   1,   0, 100,  400',
' Mach,    0.0E+0,  0.15E+0,  0.3E+0,  2.0,   1,   0, 1000, 1625',
' Tv01,   -1.0E+0,  0.0E-0,  1.0E+0,  5.0,   1,   0, 1000, 1625',
' obj,     7.5E-1,  1.2E+0,  2.0E+0,  2.0,   1,   0, 1000, 1625',

```

SEND

\$PLOT_DAT

```

Run_name = 'shp01'=00,  PLOT_echo = T,
plt_freq =  0,  10,  0,  0,  0,  0,  0,  0,
plt_name = 'rho', 'Trc', 'Vx', 'Vx', 'Vy', 'Vx', 'Tshp', 'Tshp',
xsec_ind =  31,  31,  31,  20,  10,  10,  31,  31,
colorbar =  0,  1,  2,  3,  4,  0,  1,  2,
FixGrid =  F,  F,  F,  T,  F,  F,  F,  F,

zoom_hor =  2,  3,  2,  2,  4,  2,  7,  7,
pan_hor1 =  0,  0,  0, 107, 10,  0,  70,  70,
pan_horN =  0,  0,  0, 180, 39,  0, 120, 120,
zoom_ver =  2,  3,  2,  2,  4,  2,  7,  7,
pan_ver1 =  0,  0,  0,  7,  3,  1,  16,  40,
pan_verN =  0,  0,  0, 41, 15, 35,  40,  16,
plt_loch = 10, 25, 30, 10, 65, 45, 32, 34,
plt_locv = 40, 40, 25, 40, 20, 40, 32, 34,

MacropxX = 2,      MacropxY = 2,      MacropxZ = 2,
LMppGrid = F,      LPixBorder = T,      keep_wndos = 1,

```

SEND

SCNTR_DAT

```
Run_name = ' shp01+00' , CNTR_echo = T,
ctr__Vx_ = -0. 8e+3,      0. 0e+3,      2. 2e+3,      0. 04,      1. 0,
ctr__Vy_ = -0. 3e+3,      0. 0,        0. 3e+3,      0. 02,      1. 0,
ctr__Vz_ = -0. 3e+3,      0. 0,        0. 3e+3,      0. 02,      1. 0,
ctr__rho =  0. 4e- 3,      1. 1373e- 3,  1. 5e- 3,      0. 05,      1. 0,
ctr__erg =  1. 0e+6,      2. 50e+6,      5. 0e+6,      0. 01,      2. 0,
ctr__Pre =  3. 75e+4,      4. 052e+4,      4. 25e+4,      0. 02,      1. 0,
ctr__Tem = -100. 0,       100. 0,        155. 0,       0. 03,      1. 0,
ctr__Cs_ =  0. 5e+4,      3. 4e+4,        1. 0e+5,       0. 01,      1. 0,
ctr__Mach = -2. 0,        0. 0,          2. 0,         0. 02,      1. 0,
ctr__Ptot = 0. 5e+6,      1. 01e+6,      2. 0e+6,       0. 01,      1. 0,
ctr__cv01 =  0. 001,      0. 01,         1. 0,         0. 02,      1. 0,
ctr__cv02 =  0. 001,      0. 01,         1. 0,         0. 02,      1. 0,
ctr__cv03 =  0. 001,      0. 01,         1. 0,         0. 02,      1. 0,
ctr__cv04 =  0. 001,      0. 01,         1. 0,         0. 02,      1. 0,
ctr__cv05 =  0. 001,      0. 1,          1. 0,         0. 02,      1. 0,
ctr__trc_ =  0. 001,      0. 01,         1. 0,         0. 02,      1. 0,
ctr__Tshp = -100. 0,       100. 0,        300. 0,       0. 03,      1. 0,
ctr__obj_ =  1. 0,        1. 0,          1. 0,         0. 00,      1. 0,
ctr__dum1 = -1. 0,        0. 0,          1. 0,         0. 10,      1. 0,
pxmx = 248,
```

\$END

2. Running the VOYEUR Graphics Package Online

Of necessity this data set is used differently in the different FAST3D program versions. In the parallel version of FAST3D, only the cross-section indices and/or locations are used to extract from none, to 10 planes from each of the three abscissa coordinate directions from the block-structured 3D grid. These are written out to a data file to be read back in as the variable buffer 'var_buf' by VOYEUR. Therefore any planes needed should be defined in FAST3D. Some may not be used subsequently in any particular run while others may be used repeatedly. VOYEUR usually runs simultaneously with FAST3D to monitor the ongoing simulations or else after the fact to post process them. The grid indices and/or locations are ignored by VOYEUR except to generate labels for the plots since the data planes have already been selected in the executing (completed) FAST3D run. The cross-section file written out by FAST3D contains the grid indices and/or locations used to define the cross-sections initially to avoid error.

In order to facilitate running the VOYEUR package, a script has been provided. The run_voyeur script is detailed here.

run_voyeur

The `run_voyeur` script runs the FAST3D graphics program, VOYEUR, either interactively or in the background. The executable program submitted is `voyeur_probname` where `probname` is the generic problem name which determines the specific user-supplied routine geometry.f. The `build_fast` script, described above, generates this executable program, which once compiled, resides in the `$FAST3D/bin/$target_arch` directory. The last argument to the script is the specific run or subproblem name which determines the names of the input files to be used in the directory `$FAST3D/runs/probname`. By typing `run_voyeur` at the prompt, with no arguments, one gets the usage:

```
%> run_voyeur
```

```
Usage: run_voyeur [-acehi] run_name
```

- a: Append the log file to a previous log file
- c: DO NOT cd to the appropriate runs directory based on run_name
- e: Send standard error to the log file
- h: Voyeur's source files remote name (Default: LOCAL)
- i: Run Voyeur interactively (i.e. not in the background)

All output of the `run_voyeur` script, which includes output of the VOYEUR program, is put into the file `runname.vlog`. This file may be viewed by using the unix `tail` command or any editor.

3. Modifying and Adjusting VOYEUR Plots in 'Real Time'

As VOYEUR is running, one can modify the `*.vi` file, save the changes and VOYEUR will regularly re-read the `*.vi` file and plot the changes. Therefore, if one wishes to change the contour levels in a specific `*.vi` file, they can modify, save and replot within a matter of seconds without having to exit VOYEUR. This utility is useful when trying to identify physical features in the flow field.

4. Killing the VOYEUR Package

In general, VOYEUR is run interactively so a '^C' kills VOYEUR. However, one can run VOYEUR in batch mode in which case a `qdel j obnumber` would kill VOYEUR.

Step 6. Post-Processing FAST3D Simulations

The range of possible needs for post-processing data from FAST3D simulations is so varied and problem-dependent that three different facilities were incorporated: 1) The dump/restart (and geometry) files, 2) selected VOYEUR cross-section files, and 3) station history files. In the FAST3D system, the desired flow field or geometry information, which can be specified by the user during the simulation, are output to a file or files. The user then reads the selected information back in, perhaps on another computer or workstation, and can compute with this data and/or reformat it according to their specific needs. As development of the FAST3D reactive flow/CFD system expands, we plan to have a number of commonly-used formats available, e.g. AVS, SPYGLASS, or PLOT3D.

6.1. The FAST3D Parallel Dump/Restart Files

The *probname_ext.geo0* grid geometry data file generated by GRIDVCE and the *probname_ext.dmp** dump/restart files generated by FAST3D can be used for archival and diagnostic purposes as well as their primary function to initialize and restart complex geometry simulations. These are generally very large data sets so writing them to permanent disk may be quite slow. Therefore, it is usually not efficient in terms of time and space to save them very often. The frequency of saving these files will depend strongly on the reliability of the computer system; the more reliable the system the less often these files need to be written out. Rather than providing programs to read these files directly, the flow solver FAST3D is started from the particular **.dmp** file desired (written at given intervals in the simulation) and the VOYEUR cross-section and/or the station history file outputs can be used to capture the desired data while the simulation is proceeding. VOYEUR cross-section files will contain a series of two-dimensional slices of data. Station history files can contain line data, plane data or volume data as specified by the user at any given interval.

This approach seems indirect but is actually the best for three reasons. First, it is often not obvious what information is actually needed at any given location or time until the simulation is partially analyzed. Second, the FAST3D data storage scheme is dynamic due to the distributed memory paradigm used to achieve high computational efficiency. Any desired output information has to be "re-assembled" due to data being spread across multiple processors. This is a relatively complex process; however, it is already in place for the dump/restart files, the VOYEUR cross-sections, and the station history files. Third, access to and turn-around on the large scalable processors for which FAST3D is designed may be quite limited. Therefore debugging one-time diagnostic routines for use directly in FAST3D will generally be slow and relatively unproductive. Instead, the recommended approach is to use a post-processing program on a workstation to read in the FAST3D data for reformatting. Because of the 'virtual node' implementation of FAST3D and its portability, this approach puts much more control in the hands of the user.

Parallel I/O for the new scalable HPC multiprocessors is still very system and hardware dependent. Therefore, the particular actions necessary to implement efficient access to the FAST3D dump/restart files is not yet very portable. Therefore, this section of the documentation and the associated portability library are still being developed. Here we give the actions necessary

to deal with FAST3D's main developmental systems, the Intel iPSC/860, and will expand these facilities in collaboration with the main sites for the various other scalable architectures.

System Dependent Documentation (Intel iPSC/860 Example):

When running FAST3D on LCP&FD's Intel systems, you may want to copy the *.dmp* files off to an archiver or move these files to another system and restart the run. In addition, LCP&FD's Intel disks are not backed-up, it is the user's responsibility to back-up files. Retrieving files off of the CFS is very similar to moving files to the CFS. An example of retrieving a *.dmp* file is shown below. The file bluntbody_3d_256x096x048_064.dmp2 will be copied to the directory where the ftp command was initiated.

```
% ftp gorgs
Connected to gorgs.nrl.navy.mil.
220 gorgs.nrl.navy.mil CMC FTP server (Version 2.5.3 Fri Feb 9 13:19:29 PST
1990) ready.
Name (gorgs:landsber): landsber
331 Password required for landsber.
Password:
230 User landsber logged in.
ftp> cd fast3d/bluntbody
250 CWD command successful.
ftp> bin
200 Type set to I.
ftp> ls
200 PORT command successful.
150 Opening data connection for /usr/ipsc/bin/ls (132.250.114.6, 1173) (0
bytes).
bluntbody_3d_256x096x048_064.geo0
bluntbody_3d_256x096x048_064.dmp1
bluntbody_3d_256x096x048_064.dmp2
226 Transfer complete.
ftp> get bluntbody_3d_256x096x048_064.dmp2
200 PORT command successful.
150 Opening data connection for bluntbody_3d_256x096x048_064.dmp2
(132.250.114.6, 1174) (25199500 bytes).
226 Transfer complete.
25199500 bytes received in 169.5 seconds (145.2 Kbytes/s)
ftp>
```

6.2. Using VOYEUR Cross Section Files for Post-processing

The graphics data cross-sections generated by FAST3D and written periodically to disk for subsequent use by VOYEUR can also be used in post-processing. These files are constructed in a rather complex format because of the dynamic multiblock data structure in FAST3D. Therefore the easiest way to extract the desired information is to use VOYEUR to read the data, but instead of (or in addition to) plotting it, one of the options is to write the data in the selected cross-section(s) to a user-readable disk file. This procedure can be used while FAST3D is running to get a time sequence of cross-section information. It can also be used as a staged process, automatically extracting and capturing subsets of the data from previously computed dump/restart files.

6.3. Specifying Station History Files for Post-processing

Fluid dynamics experiments usually make measurements at only a few points in the flow but record the data almost continuously in time. These 'station history' measurements require a kind of output data qualitatively different from the 2D and 3D snapshots of the simulated flow obtained from the FAST3D dump/restart or VOYEUR cross-section files for comparison. Typically, to reproduce essentially continuous variation of the fluid properties at a point requires data at least every five or ten timesteps. When shocks and blast waves are involved, the governing flow variables can change appreciably in one or two timesteps where the flow discontinuities occur.

To save data even every 10 steps from a single run, which may last 10,000 timesteps, requires reducing the number of cells at which values are recorded by at least as large a corresponding factor. If the resulting data files are to be considerably smaller than the dump/restart files, an even greater reduction in the data stored from each timestep has to be made. Suppose the full dump file is 280 megabytes in size (7 flow variables in 32-bit precision on a 200 x 200 x 250 grid). When this full dump/restart file is saved every 1000 steps, a 3D station history file for the same integration interval, perhaps 10% of the interesting portion of a run, will require 28 megabytes when every tenth point in each direction is saved, a 20 x 20 x 25 grid. While the original 280 megabyte dump file is not prohibitive for a large parallel processor, manipulating multiple 28 megabyte files for a desktop Fourier transform program can be a major undertaking. Moving using ftp or rcp, for instance, such files around can be time-consuming and disk-spacing consuming. E-mailing one of these files is even worse.

To turn station data collection on, in the file *probname.ni* set the flag `L_hist = .TRUE.`. The frequency of storing the selected data is in an array `hist_freq(5)`. The information in `hist_freq(5)` is:

```
hist_freq(1) = record the data every n timesteps (must be a multiple of 2)
hist_freq(2) = write the buffer every m timesteps
hist_freq(3) = open a new file every k timesteps
hist_freq(4) = minimum step for data collection (starting value)
hist_freq(5) = maximum step for data collection (final value)
```

For example, the *probname.ni* file may contain the following:

```
hist_freq = 2, 50, 400, 20001, 30001,          L_hist = .TRUE. ,
```

Since `L_hist` is set equal to `.true.`, this means record the data every other timestep, write the buffer every 50 timesteps, and write a new station data file every 400 timesteps. Station data will only be collected from timesteps 20001 to 30001.

Another option exists to record average values. This is achieved by setting `hist_freq(1)` negative. Now:

`hist_freq(1) = -n` samples data every `n` timesteps and acts as an accumulator
`hist_freq(2) =` writes the buffer (value) every `m` timesteps (one average value will be written)

This input is strictly related to the frequency of station data collection. It does not indicate the locations of the station data points. FAST3D allows station data points to be specified in three ways. In the file *probname.ni*, the switch that determines this is `N_st(3)`, which is an array of length three. The user can specify any or all of the following:

`N_st(1) =` number of individual station data points
`N_st(2) =` number of lines (unimplemented)
`N_st(3) =` number of volume boxes

Once this is specified the actual points, lines or volumes are specified. `st_ijk` is the variable that contains this information. If, for instance, one specifies:

```

N_st(1) = 3, N_st(2) = 0, N_st(3) = 2
st_ijk = 9, 11, 13,
         11, 9, 13,
         13, 11, 9,
         12, 48, 4, 12, 24, 2, 8, 36, 4,
         60, 90, 3, 24, 36, 3, 36, 72, 6
  
```

This indicates that three station data point locations are desired in (i,j,k) space of (9,11,13), (11,9,13), and (13,11,9). The next two lines (18 data values) represent `imin,imax,istep,jmin,jmax,jstep,zmin,zmax,zstep` of the bounding volume box. Station data collection is now fully specified.

6.4. Reading/Interpreting the Station History Files

A serial program called `STD_DAT` has been written to read the station history files generated by FAST3D and reformat them for subsequent analyses. This is not currently a scalable program, the required work load being so much less than to perform the underlying CFD simulation. Clearly, the station data history files must reside in a location where `STD_DAT` can read the data, this may require these files to be ftp'ed off of a parallel system back on to a workstation.

Section 7. FAST3D Test Problems

A large number of tests were performed during the development of FAST3D. A few of these have been documented in the past and/or have led to published research papers. Others are primarily tests of interest to potential users as a benchmark of the model's accuracy, flexibility, etc. or as a convenient starting point for closely related research problems. Six classes of test problems are identified here, each class characterized by a distinct, geometry-defining subroutine *geometry.f_probname*. In this section we reproduce these six geometry routines as referenced in the preceding sections of this manual, (Steps 1 - 6). We briefly discuss the specific test problems chosen from within that class. The corresponding input data sets are given in Appendix D. The first three of these test problem sets, Sections 7.1, 7.2, and 7.3, have been purposely constructed to run satisfactorily with FAST3D without needing to use Virtual Cell Embedding. The last three test problem sets require VCE, slated for later release with the VCE grid generator, GRIDVCE. Their documentation is therefore incomplete, here and in the appendices, and the geometry routines will be provided subsequently. Data that is problem-specific, such as blast radius, wall height, etc., is now data-driven by the *.gi and *.ni files. Thus, re-compilation of the code is not necessary when changing these parameters.

7.1. Blast Problems with Null Geometry

The FCT algorithms in FAST3D were originally designed to accurately capture the sharp gradients and flow discontinuities associated with shock and blast problems. The problem name is *blast*. This problem uses the *null* geometry subroutine since the problem complexity arises strictly from the non-uniform flow field initialization of the fluid density and pressure. This initialization is performed by *flow_field.f_blast* when it is called on the first timestep by FAST3D. Four test problems can be run by using the files *geometry.f_null*, *flow_field.f_blast* and *flow_mods.f_null*. One does not need to copy the file *flow_mods.f_null* to *flow_mods.f_blast*, if the file does not exist the default extension is *null*. This also applies to the geometry subroutine although one may later wish to add some geometry to the problem. The four test problem names and simulation descriptions are given here.

<i>blast_1d</i>	1D Cartesian Diaphragm Problem
<i>blast_2d</i>	2D Cylindrical Blast
<i>blast_axi</i>	R-Z axisymmetric simulation of spherical blast
<i>blast_3d</i>	3D Cartesian simulation of spherical blast

These problems will be described in more detail below.

The subroutine Geometry within the file *geometry.f_null* is shown below. This null geometry routine is also the default template supplied as the minimal starting point for a user to construct a *geometry.f_probname* routine. It automatically labels every point in the computational domain as being in the fluid, i.e. no body or bodies are present in the computational domain. Therefore, it is directly useful for many scientific fluid dynamics and reactive flow problems.

c- *-fortran- *-

c Last Edited: Mon Aug 19 16:12:47 1996 by Jay Boris

```

SUBROUTINE Geometry ( Ldir, xt, yt, zt, Ftr_geom )
C -----
C *****
C NOTE: this routine establishes the null (open domain) geometry.
C *****
C xt, yt, zt = Location (in cm) of the point in space being tested.
C Ldir       The current search direction (not needed here).
C Ftr_geom = 0 if xt,yt,zt lies in the fluid (there are no bodies).
C *****
IMPLICIT NONE
REAL      xt, yt, zt      ! Standard argument declarations
INTEGER   Ldir, Ftr_geom  ! Standard argument declarations

Ftr_geom = 0

RETURN
END
C *****

```

The flow modification routine `flow_mods.f_null` is also provided for completeness. This subroutine is typically used for localized physics and will be almost always be problem specific. The main use is to insert or remove physics, such as blasts, at a timestep other than zero or to include problem specific physics such as a helicopter downwash model. The subroutine shown below, if called, would simply return. This subroutine is called from `ijk_fct.f` for an entire line segment in the current integration direction.

```

c- *-fortran- *-
C Last edited: Wed Mar 20 16:16:44 1996 by Ted Young

Subroutine ff_mods (Dat_buf, Ldir, Lsg1, Lsgn, Nxyz)
C -----

Implicit NONE          ! A good bug catching habit.

C Local Declarations . . .
Integer   Ldir, Lsg1, Lsgn, Nxyz

Real      Dat_buf(1)

Return
End

```

The above set of blast problems is controlled by the subroutine `flow_field.f_blast`, reproduced here. Each of the test problems will be described first so that the user can understand this subroutine.

blast_1d - 1D Cartesian Diaphragm Problem:

Physically this is a 1D Cartesian diaphragm problem which is run using a 2D grid. Set `iprob=1` in `flow_field.f_blast`. This will set the density and pressure equal to `rhoinit` and `preinit` for

all x values below $x=rad$, where rad is specified in the namelist, FFIM_DAT, in the file blast_1d.ni.

blast_2d - 2D Cylindrical Blast:

This problem is initialized identical to blast_1d. The difference is that in blast_2d.gi the value of $\alpha=2,1,1$. The first element of α indicates that this direction, in this case the x-direction, is a radial term indicating axisymmetric coordinates. x can now be thought of as r and y will be z in RZ coordinates. This will set the density and pressure equal to ρ_{init} and p_{init} for all r values below $x=rad$, where rad is specified in the namelist, FFIM_DAT, in the file blast_2d.ni. Although unnecessary, you can set $iprob=2$ in flow_field.f_blast to keep track of 1D, 2D and 3D.

blast_axi -R-Z axisymmetric simulation of spherical blast

This problem initializes a spherical blast. Since this is an axisymmetric simulation, the problem is set up so that $r=0$ is the axis of symmetry. The blast is set at a height (y_0) along this axis of symmetry. The blast radius is specified in a data statement within flow_field.f_blast. To improve the representation of a spherical blast on a Cartesian grid, cell subdivision is performed along the sphere's surface to better approximate the density and pressure in these cells. Set $iprob=3$ in flow_field.f_blast since this physically is a 3D sphere, although solved with a 2D axisymmetric grid.

blast_3d - 3D Cartesian simulation of spherical blast

This problem is initialized identical to blast_axi. Rather than using an axis of symmetry to model a spherical blast, a full 3D Cartesian grid is used. Clearly this solution is computationally more expensive than the axisymmetric solution.

The file flow_field.f_blast is shown here. The Entry In_Flow simply returns since no special boundary conditions are required for this problem. This option will be used in other test problems.

c- *-fortran- *-

C Last edited: Wed Aug 7 14:45:30 1996 by Ted Young

```
      Subroutine Flow_fie ld (i_pos, j_pos, k_pos, NxPr_mrk)
```

```
C      -----
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

C Problem specific subroutine to initialize FAST3D flow field variables

C as a function of position (i_pos, j_pos, k_pos).

```
      Impl icit NONE ! A good bug catching habit!
```

```
      Incl ude ' f3d_i ncl ds/f3d_si ze. h'
      Incl ude ' f3d_i ncl ds/f3d_admn. h'
      Incl ude ' f3d_i ncl ds/f3d_gri ds. h'
      Incl ude ' f3d_i ncl ds/f3d_i nput. h'
```

```

Include 'f3d_user.h' ! User's responsibility

C Local declarations.
Real trcamb
Parameter (trcamb = 0.0)

Integer nsub
Real rsub, rsubcube
Parameter (nsub = 10, rsub = 1.0/nsub, rsubcube = rsub**2)

Logical First_call

Integer Lv, Mode, i_pos, j_pos, k_pos, in, jn, iprob
Integer NxPr_mrk(*)

Real rho0, pre0, vx0, vy0, vz0, trc0
Real Vars(*)

Real fracamb, fracinit
Real rad, rad_max, rad_sq
Real x0, y0, xpt, ypt, dx, dy, xt, yt, dvol

Save First_call, Mode

Data First_call /. true. /, Mode /0/
Data iprob / 3 /
Data x0, y0, rad / 0.0, 32.0, 8.0 /

RETURN

Entry In_Flow (i_pos, j_pos, k_pos, Vars)
c -----

RETURN

Entry Init_Flow (i_pos, j_pos, k_pos, Vars)
c -----

c Calculate some constants / properties
If (First_call) Then
  First_call = .false.
End If !First_call

xt = 0.5*(cbn(i_pos, 1) + cbn(i_pos+1, 1))
yt = 0.5*(cbn(j_pos, 2) + cbn(j_pos+1, 2))

rho0 = rhoamb
pre0 = preamb
vx0 = vxinit
vy0 = vyinit
vz0 = vzinit

If (iprob .eq. 1 .or. iprob .eq. 2) Then

```

```

    If (xt .lt. rad) Then
        rho0 = rhoinit
        pre0 = preinit
    Endif

Else

    dvol = 0.0
    rad_max = 1.5*rad
    rad_sq = (xt - x0)**2 + (yt - y0)**2

c Do cell subdivision to get more accurate density and pressure.

    If (rad_sq .le. rad_max**2) Then
        dx = (cbn(i_pos+1, 1) - cbn(i_pos, 1))
        dy = (cbn(j_pos+1, 2) - cbn(j_pos, 2))
        Do jn = 1, nsub
            ypt = cbn(j_pos, 2) + 0.5*rsub*dy + Float (jn - 1)*rsub*dy
            Do in = 1, nsub
                xpt = cbn(i_pos, 1) + 0.5*rsub*dx + Float (in - 1)*rsub*dx
                rad_sq = (xpt-x0)**2 + (ypt-y0)**2
                If (rad_sq .le. rad**2) dvol = dvol + rsubcube
            End Do
        End Do
    End If

    fracinit = dvol
    fracamb = 1.0 - dvol

C Initialize Rho, E, RVx, RVy, & RVz for ambient . . .
    rho0 = fracamb*rhoamb + fracinit*rhoinit
    pre0 = fracamb*preamb + fracinit*preinit

End If

Vars (1) = rho0
Vars (2) = pre0/gammam + 0.5*rho0*(vx0**2 + vy0**2 + vz0**2)
Vars (3) = vx0*rho0
Vars (4) = vy0*rho0
Vars (5) = vz0*rho0

C Intialize any special variables (i.e. Tracers, Water, etc.)
    trc0 = trcamb
    Do Lv = 1, Nsv
        Vars (Lv+Npv) = trc0
    End Do

C Intialize any chemical variables (i.e. Species, etc.)
    If (Ncv .gt. 0) Then
        Call Ichem_vars (fracamb, fracinit, Vars(Npv+Nsv+1))
    End If

RETURN
END

```

7.2. Blunt Body and Jet Problems

The subsonic flow past a bluff body (rectangular body) has previously been computed using the FCT algorithm and compared with experiment [Grinstein]. The results show agreement with experiment by comparing Strouhal number and pressure drag on the body. One of the flow conditions ($M = 0.3$) presented in [Grinstein] was chosen as the basis for the second test problem for FAST3D. This allows the user to compare to previous published computational and experimental results. The problem name is *bluntbody*. The two sub-problems can be run by using the files *geometry.f_bluntbody*, *flow_field.f_blunt_body* and *flow_mods.f_null*. The problem name and simulation description are listed below.

<i>bluntbody_2d</i>	Rectangular body trailing edge, X-Y Cartesian
<i>bluntbody_3d</i>	Rectangular blunt body with trailing edge notch

The file *geometry.f_bluntbody* is shown here. Additional discussion is given below.

```

c- *-fortran- *-
C Last Edited: Thu Nov 21 15:58:12 1996 by Ted Young

      Subroutine Geometry (Ldir, xt, yt, zt, Ftr_geom)
C      -----

c *****
c NOTE: this routine determines if the given point is inside the so
c       called speed bump
c *****
c xt, yt, zt = Location (in cm) of point in space being tested to see
c              if it lies within one of the bodies in the grid
c
c Ldir          Indicates the general direction the search is taking
c              i. e. Ldir = 1, 2, 3 indicates x, y, z directions
c              respectively.
c
c Ftr_geom =   1 if xt,yt,zt lies inside a body,
c              0 if xt,yt,zt lies outside a body,
c              -1 if it can be determined that all values along a given
c              'Ldir' direction lie outside a body
c *****
      Implicit NONE

      Include    '../FAST3D_SRC/f3d_input.h'

C Local declarations . . .
      Logical   First_call
      Real      xt, yt, zt
      Integer   Ldir, Ftr_geom

      Real      Xwall_1, Xwall_2, Ywall_1, Ywall_2, Zwall_1, Zwall_2
      Real      Xnotch_1, Xnotch_2

```

```

Real      Ynotch_1, Ynotch_2
Real      Znotch_1, Znotch_2

Save      Xwall_1, Xwall_2, Ywall_1, Ywall_2, Zwall_1, Zwall_2
Save      Xnotch_1, Xnotch_2
Save      Ynotch_1, Ynotch_2
Save      Znotch_1, Znotch_2

Data      First_call /. True. /

If (First_call) Then
  First_call = .False.
  Read (Geom_rec(1), *) Xwall_1, Xwall_2
  Read (Geom_rec(2), *) Ywall_1, Ywall_2
  Read (Geom_rec(3), *) Zwall_1, Zwall_2

  Read (Geom_rec(4), *) Xnotch_1, Xnotch_2
  Read (Geom_rec(5), *) Ynotch_1, Ynotch_2
  Read (Geom_rec(6), *) Znotch_1, Znotch_2
  If (GEOM_echo) Then
    Write (*, '(/A)') ' Input echoed from GEOMETRY. F_BLUNTBODY'
    Write (*, *) ' Xwall_1, Xwall_2 =', Xwall_1, Xwall_2
    Write (*, *) ' Ywall_1, Ywall_2 =', Ywall_1, Ywall_2
    Write (*, *) ' Zwall_1, Zwall_2 =', Zwall_1, Zwall_2

    Write (*, *) ' Xnotch_1, Xnotch_2 =', Xnotch_1, Xnotch_2
    Write (*, *) ' Ynotch_1, Ynotch_2 =', Ynotch_1, Ynotch_2
    Write (*, *) ' Znotch_1, Znotch_2 =', Znotch_1, Znotch_2

  End If

End If

End If

Ftr_geom = 0
If (Ldir .le. 0) Return

If (xt .ge. Xwall_1 .and. xt .le. Xwall_2) Then
  If (yt .ge. Ywall_1 .and. yt .le. Ywall_2) Then
    If (zt .ge. Zwall_1 .and. zt .le. Zwall_2) Then
      Ftr_geom = 1

      If (xt .gt. Xnotch_1 .and. xt .lt. Xnotch_2) Then
        If (yt .gt. Ynotch_1 .and. yt .lt. Ynotch_2) Then
          If (zt .gt. Znotch_1 .and. zt .lt. Znotch_2) Then
            Ftr_geom = 0

          End If
        End If
      End If

    End If
  End If

End If

End If
End If

```

Return
End

c *****

Some Geometry routines have large computational domains where the body or bodies reside in a small section of the full domain, for such cases, one can optimize the geometry routine, this is discussed in detail in Section 9.3. The trade-off between simplicity and efficiency can be important and is also discussed in Section 9.3. In this simple blunt body example, analytic formulas are used to determine the extent of the body and the notch. We have, at various times, used table look-ups, interpolations, postscript pixel files of text, and even black-box subroutines to evaluate the single simple question: is the point (xt, yt, zt) in the fluid being integrated or in one of the solid bodies determining the geometry of the fluid flow?

bluntbody_2d - Rectangular body trailing edge, X-Y Cartesian

The domain for the two-dimensional blunt body is specified in bluntbody_2d.gi and has the following values:

x-direction: -16 to +752 (axial)
y-direction: -96 to +96 (vertical)

The solid body is specified as data in bluntbody_2d.gi as:

x-direction: -16 to 0 (axial)
y-direction: -8 to +8 (vertical)

In two-dimensions, this problem is initialized as a uniform flow over a rectangular body. Vortex shedding will eventually start due to round-off error and feedback from the boundaries after several thousand time steps. The shedding frequency matches that shown in [Grinstein].

bluntbody_3d Rectangular blunt body with trailing edge notch

The domain for the three-dimensional blunt body is specified in bluntbody_3d.gi and has the following values:

x-direction: -16 to +240 (axial)
y-direction: -48 to +48 (vertical)
z-direction: -24 to +24 (transverse)

The solid body is specified as data in bluntbody_3d.gi as:

x-direction: -16 to 0 (axial)
y-direction: -8 to +8 (vertical)
z-direction: -24 to +24 (transverse)

The open notch is specified as data in `bluntbody_3d.gi` as:

x-direction: -4 to 0 (axial)
y-direction: -8 to +8 (vertical)
z-direction: -4 to +4 (transverse)

We could have generated a simple rectangular blunt body in three-dimensions without a notch; the flow solution in each z-plane would then be identical to the two-dimensional solution. This can be done by changing data values for the notch so that no notch is created. However, the effect of the notch is to generate a true three-dimensional flow.

The `flow_field.f_bluntbody` is shown below.

c- *-fortran- *-

C Last edited: Wed Sep 11 10:44:08 1996 by Ted Young

```
C      Subroutine Flow_field (i_pos, j_pos, k_pos, NxPr_mrk)
C      -----
```

C Problem specific subroutine to initialize FAST3D flow field variables
C as a function of position (i_pos, j_pos, k_pos).

```
      Implicit NONE                                ! A good bug catching habit!
```

```
      Include  '.. /FAST3D_SRC/f3d_size.h'
      Include  '.. /FAST3D_SRC/f3d_admn.h'
      Include  '.. /FAST3D_SRC/f3d_bcs.h'
      Include  '.. /FAST3D_SRC/f3d_gri ds.h'
      Include  '.. /FAST3D_SRC/f3d_i nput.h'
      Include  '.. /FAST3D_SRC/f3d_l ocal.h'
```

```
      Include  'f3d_user.h'
```

C Local declarations.

```
      Real      trcamb
      Parameter (trcamb = 0.0)
```

```
      Logical   First_call
```

```
      Integer   Lv, Mode, i_pos, j_pos, k_pos, Ldir, In_body
      Integer   NxPr_mrk(*)
```

```
      Real      rho0, pre0, vi0, vj0, vk0, trc0
      Real      c_sound, rc_sound, ycn
      Real      Vars(*)
```

```
      Save      First_call, Mode, rc_sound
```

```
      Data      First_call /.true./, Mode /0/
```

```
      Return
```

```

C      Entry In_Flow (i_pos, j_pos, k_pos, Vars)
C      -----

      Mode = 1

C Initialize inflow and outflow boundary conditions . . .

      If (First_call) Then
        First_call = .false.
        rc_sound = Sqrt (rhoamb/(gamma0*preamb))
        rhobc(4)   = rhoamb
        prebc(4)   = preamb
        Do Ldir = 1, 3
          velbc (4, Ldir) = vamb(Ldir)
          machbc(4, Ldir) = Abs (velbc(4, Ldir)) *rc_sound
        End Do
      End If

      ycn = 0.5*(cbn(j_pos+1, 2) + cbn(j_pos, 2) )

      If (ycn .lt. 0.0) Then
        svbc(4, 1) = -1.0
      Else
        svbc(4, 1) = +1.0
      Endif

      Return

C      Entry Init_flow (i_pos, j_pos, k_pos, Vars)
C      -----

      ycn = 0.5*(cbn(j_pos+1, 2) + cbn(j_pos, 2) )

C Initialize Rho, E, RVi, RVj, & RVk . . .
      rho0 = rhoamb
      pre0 = preamb
      vi0  = vamb(1)
      vj0  = vamb(2)
      vk0  = vamb(3)

      Vars (1) = rho0
      Vars (2) = pre0/gammam + 0.5*rho0*(vi0**2 + vj0**2 + vk0**2)
      Vars (3) = vi0*rho0
      Vars (4) = vj0*rho0
      Vars (5) = vk0*rho0

C Intialize any special variables (i.e. Tracers, species, etc.)
      If (ycn .lt. 0.0) then
        trc0 = -1.0
      Else
        trc0 = 1.0
      Endif
      Do Lv = 1, Nsv
        Vars (Lv+Npv) = trc0

```

```

End Do

Mode = 0

Return
End

```

The flow field subroutine initializes a uniform flow from the ambient conditions specified in bluntbody_2d.ni or bluntbody_3d.ni. In this subroutine a "special variable" (nsv =1) is used to represent a tracer to show mixing. The tracer is initialized to have a value of -1.0 below $y = 0.0$ and a value of +1.0 above $y = 0.0$. This in turn requires that the inflow boundary conditions also represent these values. The implementation of this is shown in Entry In_Flow where the special variable boundary condition is set to match the flow field initialization. No other flow modifications are performed; therefore, the subroutine flow_mods.f_null can be used.

7.3. Muzzle Blast Problems

The three test problem names and simulation descriptions are given here.

muzzle_2d	Idealized Cartesian blast into quiescent air
muzzle_axi	Idealized axisymmetric blast out of a muzzle
muzzle_3d	Idealized muzzle blast problem in 3D Cartesian

muzzle_2d - Idealized Cartesian blast into quiescent air

Similar to the blast problem, this test problem has a high pressure/high density region, however, this problem has geometry. A wall is specified as data in muzzle_2d.gi by Rad_1 and Rad_2 and B_len where Rad_1 and Rad_2 are the inner and outer radius of the solid wall and B_len is the height of the wall (this assumes the wall starts at the lower boundary). The high pressure/high density region are specified as data in muzzle_2d.ni and use the rhoinit and preinit values, the rhoamb and preamb specify the density and pressure everywhere else in the domain. This high pressure/high density region exists from the left boundary to Rad_init and from the lower boundary to Z_init. These values are specified as data in muzzle_2d.ni. Typically the value of Rad_init equals Rad_1 and Z_init is some value lower than B_len.

muzzle_axi - Idealized axisymmetric blast out of a muzzle

This problem is identical to muzzle_2d with the exception that the muzzle is axisymmetric. To make the problem axisymmetric one sets $\alpha = 2, 1, 1$ in the file muzzle_axi.gi. This will create an axis of symmetry at the left boundary. The geometry specification is identical for muzzle_2d and muzzle_axi; however, the grid generator and flow solver use the correct metrics for axisymmetric flow.

muzzle_3d - Idealized muzzle blast problem in 3D Cartesian

This problem is identical to muzzle_axi in terms of the resulting physics; however, in three-dimensions the muzzle will require VCE gridding. In general one would not compute a full three-

dimensional solution unless the physics was truly three-dimensional. However, by introducing variations in the flow field initialization or geometry one could easily introduce three-dimensional physics.

The geometry.f_muzzle file for muzzle_2d and muzzle_axi is shown here. Notice that the parameters describing the wall are data-driven from the muzzle_2d.gi and muzzle_axi.gi files.

```

c- *-fortran- *-
C Last Edited: Wed Nov 20 08:57:58 1996 by Ted Young

      Subroutine Geometry (Ldir, xt, yt, zt, Ftr_geom)
C      -----

c *****
c NOTE: this routine determines if the given point is inside the so
c       called speed bump
c *****
c xt, yt, zt = Location (in cm) of point in space being tested to see
c              if it lies within one of the bodies in the grid
c
c Ldir          Indicates the general direction the search is taking
c              i.e. Ldir = 1, 2, 3 indicates x, y, z directions
c              respectively.
c
c Ftr_geom =   1 if xt,yt,zt lies inside a body,
c              0 if xt,yt,zt lies outside a body,
c              -1 if it can be determined that all values along a given
c              'Ldir' direction lie outside a body
c *****
      Implicit NONE

      Include  '.. /FAST3D_SRC/f3d_input.h'

C Local declarations . . .
      Logical  First_call

      Integer  Ldir, Ftr_geom

      Real     xt, yt, zt

      Real     Rad_1, Rad_2, B_len

      Save     First_call, Rad_1, Rad_2, B_len

      Data     First_call /. True. /
Ctry         Data Rad_1, Rad_2, B_len /1.6, 2.6, 2.0/
c
      Ftr_geom = 0

      If (First_call) THEN
         First_call = .FALSE.
         Read (Geom_rec(1), *) Rad_1, Rad_2, B_len

```

```

      If (GEOM_echo) Then
        Write (*, '(/A)') ' Input echoed from GEOMETRY.F_MUZZLE'
        Write (*, *) 'Rad_1, Rad_2, B_len =', Rad_1, Rad_2, B_len
      End If

      End If

      If (Ldir .le. 0) Return
c
      If (xt .ge. Rad_1 .and. xt .le. rad_2 .and. yt .le. B_len) Then
        Ftr_geom = 1
      End If
c
      Return
      Endc

```

The flow_field.f_muzzle file for muzzle_2d and muzzle_axi is shown here. Notice that the parameters describing the high pressure/high density are data-driven from the muzzle_2d.ni and muzzle_axi.ni files. The blast in this subroutine is aligned to the grid; however, this subroutine does use cell-subdivision at the interface of the blast for cases when the blast is not aligned to the grid such as with spherical shocks.

c- *-fortran- *-

C Last edited: Tue Nov 26 10:55:03 1996 by Ted Young

```

      Subroutine Flow_fie ld (i_pos, j_pos, k_pos, NxPr_mrk)
c  -----

```

C Problem specific subroutine to initialize FAST3D flow field variables
C as a function of position (i_pos, j_pos, k_pos).

```

      Implicit NONE                                ! A good bug catching habit!

      Include  '.. /FAST3D_SRC/f3d_si ze. h'
      Include  '.. /FAST3D_SRC/f3d_adm n. h'
      Include  '.. /FAST3D_SRC/f3d_gri ds. h'
      Include  '.. /FAST3D_SRC/f3d_i nput. h'
c  try      Include  '.. /FAST3D_SRC/f3d_l ocal. h'

```

C Local declarations.

```

      Real      trcamb
      Parameter (trcamb = 0.0)

```

```

      Logical   First_call

```

```

      Integer   Lv, Mode, i_pos, j_pos, k_pos, Ldir, In_body
      Integer   NxPr_mrk(*)

```

```

      Real      rho0, pre0, vi0, vj0, vk0, trc0
      Real      Vars(*)

```

```

Real      fracamb, fracinit
Real      rt, zt

Real      Rad_init, Z_init

Save      First_call, Mode, Rad_init, Z_init

Data      First_call /.true./, Mode /0/
Ctry      Data Rad_init, Z_init /1.6, 1.0/

Return

Entry In_Flow (i_pos, j_pos, k_pos, Vars)
C
-----

mode = 1

Return

Entry Init_flow (i_pos, j_pos, k_pos, Vars)
C
-----

c Calculate some constants / properties
  If (First_call) Then
    First_call = .false.
    Read (FFim_rec(1), *) Rad_init, Z_init
    If (Lgn .eq. iecho_Lgn) Then
      Write (*, *) 'Rad_init, Z_init =', Rad_init, Z_init
    End If

  End If

                                !First_call

  rt = 0.5*(cbn(i_pos, 1) + cbn(i_pos+1, 1))
  zt = 0.5*(cbn(j_pos, 2) + cbn(j_pos+1, 2))

C D0 cell subdivision to get more accurate density and pressure.

  fracinit = 0.0
  If (rt .le. Rad_init .and. zt .le. Z_init) Then
    fracinit = 1.0
  End If
  fracamb = 1.0 - fracinit

C Initialize Rho, E, RVx, RVy, & RVz for ambient . . .
  rho0 = fracamb*rhoamb + fracinit*rhoinit
  pre0 = fracamb*preamb + fracinit*preinit
  vi0 = fracamb*vamb(1) + fracinit*vinit(1)
  vj0 = fracamb*vamb(2) + fracinit*vinit(2)
  vk0 = fracamb*vamb(3) + fracinit*vinit(3)

  Vars (1) = rho0
  Vars (2) = pre0/gammam + 0.5*rho0*(vi0**2 + vj0**2 + vk0**2)
  Vars (3) = vi0*rho0
  Vars (4) = vj0*rho0

```

```
Vars (5) = vk0*rho0
```

```
C Initialize any special variables (i.e. Tracers, species, etc.)
```

```
trc0 = trcamb  
Do Lv = 1, Nsv  
  Vars (Lv+Npv) = trc0  
End Do
```

```
Mode = 0
```

```
Return  
End
```

There are no flow field modifications, therefore the flow_mods.f_null file is used.

7.4. Wedge/Cone Problems

The test case for a wedge is defined as a Mach = 2.0 flow past a 15 degree wedge at standard atmospheric conditions. The wedge angle is data driven in GEOM_DAT. The flow conditions are also data-driven as specified in wedge_2d.ni.

```
geometry.f_wedge  
  wedge_2d      Cartesian Shock-on-Wedge face on (X-Y)  
  wedge_3d      Cartesian Shock-on-Wedge at a skew angle  
  wedge_axi     Conical wedge in axisymmetric R-Z coordinates  
  wedge_3dcone  Conical wedge in 3D
```

The flow field uses the default flow field and flow mods. Only the geometry is unique to this wedge case. These cases can be compared to analytic solutions for flow over a 2-D wedge and cone flow. The geometry.f_wedge file is shown here.

```
c *-fortran*-
```

```
C Last Edited: Mon Jul 15 13:22:51 1996 by Ted Young
```

```
Subroutine Geometry (Ldir, xt, yt, zt, Ftr_geom)
```

```
C
```

```
C *****
```

```
C NOTE: this geometrical space has no features imbedded in it
```

```
C *****
```

```
C xt, yt, zt = Location (in cm) of point in space being tested to see  
C             if it lies within one of the bodies in the grid
```

```
C Ldir          Indicates the general direction the search is taking  
C              i.e. Ldir = 1, 2, 3 indicates x, y, z directions  
C              respectively.
```

```
C Ftr_geom = 1 if xt,yt,zt lies inside a body,  
C           0 if xt,yt,zt lies outside a body,  
C           -1 if it can be determined that all values along a given
```

```

c          'Ldir' direction lie outside a body
c *****
Implicit NONE

Include   '.. /FAST3D_SRC/f3d_input.h'
c *****
Integer   Ftr_geom, Ldir, x_body, y_body, z_body

Logical   First_call

Data      First_call /. True. /

Real      Ca, x0, xt, yt, zt, Pi, Angle
Parameter (Pi = 3.141592654)

Ftr_geom = 0

If (First_call) THEN
  First_call = .FALSE.
  Read (Geom_rec(1), *) x0, Angle
  If (GEOM_echo) Then
    Write (*, '(/A)') ' Input echoed from GEOMETRY.F_WEDGE'
    Write (*, *) 'X0, Angle =', x0, Angle
  End If

End If

Ca = Angle*Pi/180.0
If (xt .ge. x0 .and. yt .le. (xt - x0)*tan(Ca)) Ftr_geom = 1

Return
End

```

7.5. Circular Arc Airfoil Problems

The circular arc airfoil is a standard aero test case. This problem has a 4% circular arcfoil in a channel with Mach = 1.4 flow. The geometry.f_arcfoil is the only unique user-supplied subroutine in this test case.

```

geometry.f_arcfoil
  arcfoil_2d      4 degree circular arc airfoil on flat wall
  arcfoil_3d      6 degree circular 'discus' on flat wall

```

The geomtery subroutine is provided here.

C Last Edited: Wed May 31 16:33:18 1995 by Ted Young

```

C Subroutine Geometry (Ldir, xt, yt, zt, Ftr_geom)
C -----

```

```

c *****
c NOTE: this routine determines if the given point is inside the so
c      called speed bump
c
c *****
c xt, yt, zt = Location (in cm) of point in space being tested to see
c              if it lies within one of the bodies in the grid
c
c Ldir          Indicates the general direction the search is taking
c              i.e. Ldir = 1, 2, 3 indicates x, y, z directions
c              respectively.
c
c Ftr_geom =  1 if xt,yt,zt lies inside a body,
c             0 if xt,yt,zt lies outside a body,
c            -1 if it can be determined that all values along a given
c              'Ldir' direction lie outside a body
c *****
c      Implicit NONE
c
c      Integer   Ldir, Ftr_geom
c
c      Real      xt, yt, zt, yc, tau, rsqrd, yheight
c
c      tau = 0.04
c      Ftr_geom = 0
c
c      yc = (tau**2 - 0.25)/(2.*tau)
c      rsqrd = 0.25 + yc**2
c      if (xt .ge. 0.0 .and. xt .le. 1.0) then
c          yheight = yc + sqrt(rsqrd - (xt - 0.5)**2)
c          if ( yt .lt. yheight) Ftr_geom = 1
c      endif
c
c      Return
c      End
c *****

```

7.6. Multiple Ellipse Problems

```

geometry.f_ellipse
    ellipse_2d      Mach 0.3 flow over a circular cylinder
    ellipse_3d      Mach 0.3 over four intersecting ellipses

```

To be included in next release.

Section 8. Additional Information About the FAST3D System

8.1. Flux-Corrected Transport and Virtual Cell Embedding Algorithms

The underlying fluid dynamics algorithm is the high-resolution, direction-split Flux-Corrected Transport (FCT) algorithm called **LCPFCT**. This subroutine packages is documented in NRL Memorandum Report 93-7192.

The complex geometry capabilities are provided by an efficient parallel implementation of the Virtual Cell Embedding (VCE) algorithms including the direction-coupling surface divergence transfer terms. These underlying algorithms are well described in Landsberg, et.al. The parallel implementation, explained briefly below, is also described in Young, et.al. For a comprehensive listing of FAST3D and FCT publications go to the following URL: <http://www.lcp.nrl.navy.mil/cfd-cta/CFD1/CFD1.html>

2. Tradeoffs Made in Constructing the FAST3D System

Numerous tradeoffs had to be made to develop the fast3d system in its present form. These involve performance efficiency/optimization, program flexibility, solution accuracy, and portability across computing systems. Having exposed you to the details of specifying, running and diagnosing simulations with '**fast3d**', a few paragraphs about these tradeoffs and the underlying philosophy in making them might help in rounding out your understanding of these programs.

The most important tradeoff made in performing detailed CFD simulations for science and engineering is the contention between program flexibility and computational performance. In **fast3d** the operational philosophy has been to maximize flexibility for a wide range of CFD problems using robust algorithms that combine high accuracy with reasonably high efficiency. Thus **fast3d** should address the general needs of many DoD users who wish to simulate compressible, transient, time-dependent fluid dynamics with reactive flow. This does not mean that **fast3d** will be completely optimal for every particular part or aspect

The flexibility and performance of **fast3d** have been enhanced simultaneously by using a sensible functional decomposition of the problem. The computational representation of the problem geometry and grid, the mathematical algorithms used to solve the physical equations, and the flow and chemical reaction physics have all been separated into distinct routines to enhance portability and readability.

3. The VCE Data Structure in FAST3D

For a comprehensive listing of FAST3D, FCT, GRIDVCE, and parallel implementation publications go to the following URL: <http://www.lcp.nrl.navy.mil/cfd-cta/CFD1/CFD1.html>

4. Portable/Heterogeneous Implementation of FAST3D

For a comprehensive listing of FAST3D, FCT, GRIDVCE, and parallel implementation publications go to the following URL: <http://www.lcp.nrl.navy.mil/cfd-cta/CFD1/CFD1.html>

5. The FAST3D Model Limitations

The user needs to understand the limitations of the present, scalable version of the FAST3D flow-solver being documented here. This version IS NOT a multiphase flow code with distinct phases (e.g. air and water) or multiple, interpenetrating velocity fields. It is not a classical Navier-Stokes solver (though a simple boundary condition is included to allow momentum extraction from the flow adjacent to a solid boundary). It is also not really an ideal Euler solver (to the extent that such a time-dependent, mathematical abstraction might be considered to exist). The model approximates the high Reynolds-Number limit solutions of the Navier-Stokes Equations within the resolution limits allowed by the fundamental grid spacing chosen by the user. This capacity results from the particular form of the weak numerical diffusion in the Flux-Corrected Transport algorithm and acts in concert with the finite-resolution (truncation-error) representation of the fundamental fluid dynamics (Euler) equations. This means that the cell-Reynolds-number limit imposed on Detailed and Spectral Numerical simulation models can be violated by a large but finite factor without most adverse effects such as violation of monotonicity. The physically unstable potential flow and Euler solutions admitted mathematically in some limits are difficult to recover using FAST3D. They become unstable and reduce to vortex shedding calculations as if triggered by an unresolved boundary layer.

The diffusion and cross-derivative terms needed to make a detailed, moderate Reynolds-number, Navier-Stokes solver can be incorporated rather directly using the FCT Source utility routines, as has been done by a number of authors, [references to Weber, Grinstein, Vuillermoz, etc]. In the current version, stable, monotone solutions are obtained although the details of small-scale, laminar boundary layers cannot be resolved. Therefore the distribution of vorticity shed into the flow will not be correctly determined in cases without sharp corners or well defined separation points. The model generally sheds the correct amount of circulation into the flow, however, to the extent that adverse pressure gradients occur as a result of the large scale flow behavior.

Solution adaptive gridding is not part of this initial distribution version of FAST3D but will be added, along with dynamic regridding for moving bodies, when the scalable implementations achieve the simplicity, efficiency, and robustness of the FCT algorithms and VCE gridding now in use. This version also does not contain explicit turbulence models nor should they be necessary since the basic flow algorithm has an effective Large Eddy Simulation capability built in [References ???]. The FAST3D system is structured so that such models, or indeed other problem-specific phenomenologies should be easy to add.

The time integration implemented in the current version is explicit and second-order accurate so flows with peak Mach number below about 0.1 can be handled at best inefficiently. Efforts are underway to develop an algorithm to alleviate this restriction using an iterative implicit technique well matched to the data structures and communication algorithm used. When successful, these additions will be incorporated in future releases.

Section 9. Advanced Uses and Planned Capabilities for FAST3D

A user, progressing in understanding the capabilities and usage of the FAST3D system, will quickly think up additional things he "needs" to do. These desired capabilities divide in three categories: 'advanced uses' permitted by the model as it currently is but not expected to be of immediate interest to a new user, capabilities that can be supplied by 'tricks' with very little, if any, additional programming, and 'future capabilities' that may require significant modification to FAST3D. A discussion of a few of these follows.

9.1. Advanced Uses and Tricks for FAST3D

Dignosing Flow Variables on a Surface: Users often want to plot flow variables on a complex surface in the flow, for example calculating the lift on an airplane by integrating the pressure. This can clearly be done by analysing the dump/restart files off line but this approach is cumbersome and impractical if it needs to be done more than a few times with a full 3D data set.

Time-Dependent Boundary Conditions:

In the next release of the code, details of how to implement time-dependent boundary conditions will be provided.

9.2. Parallel Data Input for User-Generated Routines

A simple, free-format data input capability has been developed for FAST3D that is compatible with the standard Fortran I/O statements. This allows data to be transferred efficiently down to each of the parallel processors in a scalable system while still permitting the users to choose variables names, types, and order to suit their specific problem.

3. Making geometry.f More Efficient

There are some constants, parameters, and computations for most problems that need only be initialized or performed once. The bunker geometry routine example being used here has been written so that local variables are saved and these initializing computations are only performed the first time the subroutine is called, i.e. when `first_call = .true.` Since subroutine **geometry** can be called many millions of times in setting up the initial finite volume grids for **fast3d** and again to determine the geometry-specific masks for the graphics in **voyeur**, each user can improve performance, often significantly, by paying attention to what does and does not need to be recomputed each time the test location (xt, yt, zt) is changed.

The two characters 'UI' after a line of Fortran in the example above indicate 'user initialization,' problem-dependent program (as opposed to declaration) that initializes parameters used by the geometry that may be saved and reused from point to point without recomputation. In the example here, `A_hill` and `B_hill` are coefficients arrays (dimensioned 2) used to specify the shape of the two hills according to the formula $y(x) = a_hill x^2 / (1 + B_hill x^2)$. These coefficients are precomputed from `H_hill`, the height of each hill, and from `X_hill`, the distance to

the inflection point from the base of the hill. Also precomputed are the sines and the cosines for the rotation of the alignment of the hills (see Figures) in the X-Z plane. The repeated computation of these trigonometric functions can also be avoided in this example. Further, in composite geometries, great speed ups can often be obtained by paying attention to the order in which tests are performed. For example, in the bunker geometry above, most of the grid is above ground so this is checked relatively early to limit complex checks for whether or not the test point is actually inside a bunker or a doorway to points that are generally below ground.

We want to stress that the flow solver's performance is not affected by these optimization efforts so you won't save a lot of supercomputer time. Nevertheless, the user is usually working interactively while the grid is being generated, the geometry is being checked, and/or the graphics are being initialized. These activities, furthermore, are often done repeatedly and in a hurry to get the problem geometry, **voyeur** plots, or **fast3d** grid just right. Therefore these optimizations are well worth your effort - particularly in the long run.

To further reduce the cost of geometry evaluations in a generic way for complex geometry, a set of rectangular 'body_check' boxes known to contain all of the bodies and features in the domain can be specified by the user. The idea is that for many truly complicated (expensive) shapes a rectangular box, or perhaps several boxes, can be prespecified that totally encloses and closely bounds the body. A given test point (xt, yt, zt) is automatically rejected, that is $Ftr_geom = 0$ is returned, if the point does not lie within one of the boxes. An airplane, for example, can be pretty well enclosed by five rectangular boxes. In the example above, all the cells and VCE subcells in the air above the hills and tower never need to be checked.

Though programmed as boxes containing body parts, **gridvce** uses this additional 'body-check' information in a much more efficient way than simply reducing the cost of geometric evaluation at each point. A quick check is done using this information to determine whether an entire row of points along direction $Ldir$ is guaranteed to be free of body points. When this occurs, as determined in subroutine **Body_Chk**, $Ftr_geom = -1$ is returned. In addition to short circuiting the remainder of the current call to **geometry**, as is evident in the example above, this information is used to ensure that **geometry** is never called again for another point along the same line.

On the first call to **geometry**, the minimum and maximum coordinates of the body-bounding boxes in all three directions are passed to the checking utility through a sequence of calls to **Body_Box** (xmin, ymin, zmin, xmax, ymax, zmax). The utility records the bounding coordinates if each box and counts the number of boxes initialized (up to 10). The user does not need to worry about these details. In the bunker geometry example, one rectangular box from $xmin = ymin = -200m$ to $zmin = zmin = +200m$ and from $ymin = 0m$ to the $ymax =$ maximum hill height is specified. The second box initialized in the example encloses the rectangular building with the conical tower on top. This second box is included to treat the cases where the building and/or tower actually extend above the top of the highest hill. The subsequent call to **Body_Chk** ($Ldir, xt, yt, zt, Ftr_geom$) checks both of these test boxes for the given test point, returning $Ftr_geom = -1$ if the line along direction $Ldir$ does not pass through any (either) of the body-check boxes.

In the **geometry.f_bunker** example above, the designation 'OUI' stands for 'optional user initialization.' These lines of program, to be provided by the user, aid the efficiency of the geometry by initializing the **Body_Chk** procedure. The use of **Body_Box** and the subsequent calls to **Body_Chk**, indicated by 'OP' for 'optional programming' just below the 'OUI' lines of code, can be deleted entirely, of course, but this is only recommended for extremely simple geometries for which the cost of checking the geometry is comparable to or less than.

4. Planned Capabilities for FAST3D (updates and future versions)

The new features added to the FAST3D package will be documented and released to the user community.

5. Problem Reports for FAST3D

The users of the FAST3D system, both inside and outside of NRL, will have problems to report. A standard problem-tracking system, GNATS, has been implemented for FAST3D. In addition, RCS is being used for version control.

References and Figures

Boris, J.P., Landsberg, A.M., Oran, E.S., and Gardner, J.H., *LCPFCT -- A Flux-Corrected Transport Algorithm for Solving Generalized Continuity Equations*, U.S. Naval Research Laboratory Memorandum Report 6410-93-7192, December 1992.

Landsberg, A.M., Boris, J.P., Young, T.R., and Scott, R.J., *Computing Complex Shocked Flows Through the Euler Equations*, 19th International Symposium on Shock Waves, University of Provence, Marseille, 26-30 July 1993.

Landsberg, A.M., Boris, J.P., Sandberg, W.C., and Young, T.R., *Naval Ship Superstructure Design: Complex Three-Dimensional Flows Using an Efficient, Parallel Method*, in **High Performance Computing 1993 - Grand Challenges in Computer Simulation**, Adrian Tentner (ed), 1993 SCS Simulation Multiconference, Arlington VA, 29 March - 1 April 1993, (SCS, San Diego, 1993).

Landsberg, A.M., Young, T.R., and Boris, J.P., *An Efficient, Parallel Method for Solving Complex Three-Dimensional Flows*, AIAA Paper 94-0413, AIAA 32rd Aerospace Sciences Meeting, Reno NV, 10 - 13 January 1994.

Oran, Elaine S. and Boris, Jay P., *Numerical Simulation of Reactive Flow (Chapter 3)*, (Elsevier Science Publishing Co, New York, 1987).

Young, T., Landsberg, A.M., and Boris, J.P., *Implementation of the Full 3D FAST3D (FCT) Code Including Complex Geometry on the Intel iPSC/860 Parallel Computer*, in **High Performance Computing 1993 - Grand Challenges in Computer Simulation**, Adrian Tentner (ed), 1993 SCS Simulation Multiconference, Arlington VA, 29 March - 1 April 1993, (SCS, San Diego, 1993).

Appendix A: Definition and Initialization of *runname.gi* Variables

The FAST3D grid generator program GRIDVCE requires that many variable values be specified by the user. These input data values are all input via a single data file supplied by the user, *runname.gi*. As explained previously, *runname* is the user-determined problem name with the subproblem extension appended following an underscore, i.e. *probname_spx*. Some of the variables initialized by this file are local variables for only one or two subroutines; others are 'Fortran common block' variables. The variables in common blocks are declared in the 'include' file *gvce_blk.h* located in subdirectory *gridvce_vno*. Common block and local variables describing similar properties are combined into lists of variable names using the NAMELIST statement in the *gridvce.f*, *inital.f*, *voyeur.f*, *graph_io.f*, and *geometry.f_probname* programs. There are four of these namelists for GRIDVCE:

GADMIN_DAT namelist variables specifying the administrative information and data necessary to run GRIDVCE. This namelist is read in file *gridvce.f*, and the variable default values are specified in file *gdat_init.f*.

SIZES_DAT namelist variables convey the information that determines the size of the FAST3D grid, e.g. the number of cells and the number and types of the convected variables. This namelist is read in files *gridvce.f*, *inital.f*, and *voyeur.f* and the variable default values are specified in file *gdat_init.f*.

GRID_DAT namelist variables convey the information that defines the FAST3D computational domain, i.e. the coordinate system location and stretching, and the underlying structured grid grid specifications. This namelist is read in files *gridvce.f*, *inital.f*, and *voyeur.f* and the variable default values are specified in file *gdat_init.f*.

GEOM_DAT namelist variables convey the data specifying the geomery of the bodies (if any) in the computational domain. This namelist is read by *gridvce.f*, *?????voyeur.f*, and *geometry.f_probname* and the variable default values are specified in file *fdat_init.f*.

The Fortran namelist facility allows the required data to be read in a self-explanatory free format. The data files themselves are easy to read and only data changed from the default values specified below actually need be provided by the user. As explained above, Variable *runname* doesn't have a default value. It is specified in the command line by user when he initiates execution of any FAST3D program. This name is used for two purposes: first, to identify the problem for which GRIDVCE is to generate a grid, and second, for checking the consistency of the input. Each NAMELIST statement includes *runname*, and it must be separately specified for each NAMELIST in each of the three FAST3D input data files.

Description of the variables contained in each of the four namelists in the input data file *runname.gi* and the default values for these variables follow. The comma is included after each default value in these three appendices to remind the reader that this comma is the variable separator for the NAMELIST statement.

GADMIN_DAT Input Variables

Namelist /GADMN_DAT/ runname, GADMIN_echo, arch, geostep, gf_bytswp, Time

```
runname = '*****' ???????
GADMIN_echo = .True. All GADMIN_DAT input variables will be echoed in log.
arch = 'MOST' ???????
geostep = 1 ???????
gf_bytswp = .False. ???????
Time = 0.0 Input required Physical problem time
```

SIZES_DAT Input Variables

Namelist /SIZES_DAT/ runname, SIZES_echo, Npv, Ncv, Nsv, Nfv,
& N_vn, Nci, Ncj, Nck, Mfdrs

```
runname = '*****' ???????
SIZES_echo = .True., All SIZES_DAT input variables will be echoed in log.
Npv = 5, # of physical (flux) variables (rho, rvx, rvy, rvz, erg)
Ncv = 0, No. Chemistry Variables (e.g. O2, H2, H2O, etc)
Nsv = 0, Number Special Variables (e.g. induction time, etc)
      (typically advected but not compressed)
Nfv = 2, Number Feature Variables ( currently fixed at 2 )
N_vn = 64, Number Virtual Nodes for this fast3d run. The number
      of physical nodes may vary from run to run.
Nci = 64, Number Cells in i-direction. These 3 numbers must be
Ncj = Nci, Number Cells in j-direction. multiples of the square
Nck = Nci, Number Cells in k-direction. root of N_vn.
Mfdrs = 200, Maximum number of feature cells in a block
```

GRID_DAT Input Variables

Namelist /GRID_DAT/ runname, GRID_echo, nsubv, nsuba, solid_bc, uniform_g,
& P_bc1, P_bcN, sleft, Lscent, Lscnc, Lsrigh, alpha,
& hsleft, Hsrigh, deltah, centrh

```
runname = '*****' ???????
GRID_echo = .True., All GRID_DAT input variables will be echoed in log.
nsubv = 8, Cell volumes divided into nsubv^3 subcells
nsuba = 16, Interface areas divided into nsuba^2 subcells
solid_bc = 1, Boundary condition on solid bodies (1 = ideal)
```

each of the following MESH specification variables is an array of three elements, i takes values from 1 to 3

```

uniform_g(i) = .True.,   Uniform (equally space) grid in direction i
P_bc1(i) = 1,           Perimeter boundary condition  ?????
P_bcN(i) = 1,           Boundary condition at Nbh interface (last cell)
Lsleft(i) = 10000,      1/isleft = fraction of cells stretched on left
Lscent(i) = 100         1/iscent = location of 0.0 in cells.
Lscnc(i) = 64,          Standard system size for scaling
Lsrigh(i) = 10000,      1/isrigh = fraction of cells stretched on right
alpha(i) = 1,           1 = Cartesian, 2 = cylindrical, 3 = spherical systems
                        of coordinates
hsleft = ????,         ??????????
Hsrigh = ????,         ??????????
deltah(i) = 1.0,       [cm] ??????????
centrh(i) = 0.0,       [cm] ??????????

```

GEOM_DAT Input Variables

NameList /GEOM_DAT/ runname, GEOM_echo, Geom_doc, Geom_rec

```

runname = '*****' ???????
GEOM_echo = .True.,   All GEOM_DAT input variables will be echoed in log.
Geom_rec(1) = ''      Record strings of free format data to
... Geom_rec(Nrec) = '' read in the user-programmed variables
Geom_doc(1) = ''      Record strings of variable names to
... Geom_doc(Nrec) = '' remind users of the data value order
                        in the parallel input lists.

```

The lists of namelist variables, their default values, and their definitions given above, and also below in appendices B and C for files *runname.ni* and *runname.vi*, are very useful reminders to have available when data is being modified to change a run. To do this we suggest incorporating the information in the tables above directly in the data files by constructing a bogus namelist, for example GRID_DOC, that contains the definitions of the GRID_DAT variables, their default values and even the physical units. These _DOC namelists are never actually read by GRIDVCE because the usual fortran namelist read statements skips down through the data file looking for a specified name with the _DAT extension). You may want to keep these documentation namelists right in the data file to cue you as to the possible choices and their meaning. The example of this for GRID_DOC is given here.

```

$GRID_DOC
runname = '*****' ???????
GRID_echo = .True., All GRID_DAT input variables will be echoed in log.
nsubv = 8,          Cell volumes divided into nsubv^3 subcells
nsuba = 16,         Interface areas divided into nsuba^2 subcells

```

solid_bc = 1, Boundary condition on solid bodies (1 = ideal)

each of the following MESH specification variables is an array of three elements, i takes values from 1 to 3

uniform_g(i) = .True., Uniform (equally space) grid in direction i
P_bc1(i) = 1, Perimeter boundary condition ?????
P_bcN(i) = 1, Boundary condition at Nbh interface (last cell)
Lsleft(i) = 10000, 1/isleft = fraction of cells stretched on left
Lscent(i) = 100 1/iscent = location of 0.0 in cells.
Lscnc(i) = 64, Standard system size for scaling
Lsrigh(i) = 10000, 1/isrigh = fraction of cells stretched on right
alpha(i) = 1, 1 = Cartesian, 2 = cylindrical, 3 = spherical
 of coordinates
hsleft = ????, ??????????
Hsrigh = ????, ??????????
deltah(i) = 1.0, [cm] ??????????
centrh(i) = 0.0, [cm] ??????????
\$END

Appendix B: Definition and Initialization of *runname.ni* Variables

The FAST3D flow solver program FAST3D requires that many variable values be specified by the user. These input data values are all input via a single data file supplied by the user, *runname.ni*. Some of the variables initialized by this file are local variables for only one or two subroutines; others are 'Fortran common block' variables. The variables in common blocks used by FAST3D are declared in include file *gvce_blk.h* located in the directory *gridvce_vno*, and *f3d_admn.h*, *f3d_bcs.h*, *f3d_cgeom.h*, *f3d_grids.h*, *f3d_hdr.h*, *f3d_input.h*, *f3d_local.h*, *f3d_lpchem.h*, *f3d_size.h*, and *f3d_std.h* in subdirectory *fast3d_vno*. The NAMELIST statements are read in the routines (files) *gridvce.f*, *inital.f*, *voyeur.f*, *graph_io.f*, and *geometry.f_probname*. There are six of these namelists used exclusively by FAST3D in addition to the three namelists DOMAIN_DAT, SIZES_DAT, GRID_DAT, and GEOM_DAT read from *runname.gi* because the grid and geometry information is also needed by FAST3D.

FADMIN_DAT namelist variables specifying the administrative information and data necessary to run FAST3D. This namelist is read in file *inital.f*, and the variable default values are specified in file *fdat_init.f*.

BC_DAT namelist variables convey all the boundary condition information needed by the FCT routines in the flow solver. This namelist is read in file *inital.f*, and the variable default values are specified in file *fdat_init.f*.

INIT_DAT namelist variables convey material specifications and initial conditions for the flow field. This namelist is read in file *inital.f*, and the variable default values are specified in file *fdat_init.f*.

CHEM_DAT namelist variables convey the chemistry species and reaction mechanism data. This namelist is read in file *inital.f*, and the variable default values are specified in file *fdat_init.f*.

FFIM_DAT namelist variables convey flow field Init/Mods input record initializations. These data are user-supplied, problem-specific input needed to establish spatially dependent initial conditions and time and space dependent boundary conditions such as local source terms. These data are read by the subroutines *flow_field* and *flow_mods* in file *flow_field.f*. This namelist is read in file *inital.f*, and the variable default values are specified in file *fdat_init.f*.

STD_DAT namelist variables convey the set of station data locations, and other information needed to control the online collection of this data. This namelist is read in file *inital.f*, and the variable default values are specified in file *fdat_init.f*.

As with the data in file *runname.gi*, the Fortran namelist utility is used to read this data into FAST3D. Description of the variables contained in each of the six namelists in the input data file *runname.ni* and the default values for these variables follow. The comma is included after each default value in these three appendices to remind the reader that this comma is the variable separator for the NAMELIST statement.

FADMIN_DAT Input Variables

NAMELIST /FADMIN_DAT/ runname, FADMIN_echo, sim_desc, file_prfx, file_pth,

& arch, dump_freq, pix_freq, hist_freq, spot_freq, note_freq,
& Opt_i, Opt_j, Opt_k, Mx_dat_bfs, r_wdl, i_wdl, cnsvr_freq,
& Minstep, Maxstep, geostep, iecho_Lgn, xs_bfs, sd_bfs,
& Rst_bfs, Dmp_bfs, xs_bytswp, sd_bytswp, Rst_bytswp,
& Dmp_bytswp, L_hist, L_pix, L_chem, L_ffim, L_geom, Dmp0_flg,
& im_buf, L_upstr, L_antidif, L_coefix, L_video, L_osfn,
& L_spot, Deltat, dtmin, dtmax, CFL_user, FCT_adiff

runname = '*****' ????????

FADMIN_echo = .True., All FADMIN_DAT input variables will be echoed in log.

sd_bfs = 256 Number of station data buffers ??????????

Rst_bfs = 256 Number of restart buffers ??????????

Dmp_bfs = 256 Number of dump buffers ??????????

Opt_i = 0 ??????????

Opt_j = 0 ??????????

Opt_k = 0 ??????????

N_vn = 64 Number of virtual nodes

note_freq = 50 Frequency at which information is written to screen.

Mfdrs = 0 Maximum number of feature data cells ??????????

iecho_Lgn = 0 Logical unit to echo data from ??????????

L_hist = .False. Turn on/off station data

L_video = .False. Turn on/off video

L_pix = .False. Turn on/off ??????????

L_upstr = .False. Turn on/off antidiffusion

L_antidif = .False. Turn on/off antidiffusion

L_coefix = .False. Turn on/off the ability to modify FCT coefficients

L_spot = .False. Turn on/off ability to write selected station data

L_osfn = .False. Turn on/off basic file naming convention

L_chem = .False. Turn on/off chemistry

L_ffim = .False. Turn on/off flow field modification

L_geom = .False. Turn on/off self initialization ??????????

xs_bytswp = .True. Turn on/off ??????????

sd_bytswp = .True. Turn on/off ??????????

Rst_bytswp = .True. Turn on/off ??????????

Dmp_bytswp = .True. Turn on/off ??????????

im_buf = .True. Turn on/off ??????????

Dmp0_flg = .True. Turn on/off ??????????

sim_desc = 'FAST3D' Simulation description

file_pth = './' Path to .geo0 and where dump and station is written arch = 'MOST'

Architecture of machine running FAST3D

Deltat = 2.0e-5 Time step, if Deltat = Dtmin = Dtmax then Dt is fixed

Dtmin= Deltat Minimum time step if CFL_user is used
Dtmax= Deltat Maximum ime step if CFL_user is used
FCT_adiff = 0.9980 Antidiffusion coefficient
CFL_user = .25 User specified CFL number

BC_DAT Input Variables

NameList /BC_DAT/ runname, BC_echo, rhobc, prebc, velbc, epsbc, svbc, cvbc
Real rhobc(6), prebc(6), velbc(6), epsbc(6), svbc(6,????), cvbc(6,????)

runname = '*****' ???????
BC_echo = .True., All BC_DAT input variables will be echoed in log.
rhobc(j) = 1.0
 rhobc(j+3) = rhoamb,
 prebc(j) = 1.0
 prebc(j+3) = preamb
 velbc(i,j) = 0.0
 epsbc(i) = 0.0
 svbc(i,j) = 0.0
 cvbc(i,j) = 0.0

All values set to look like a namelist, not fortran code

INIT_DAT Input Variables

NameList /INIT_DAT/ runname, INIT_echo, gamma0, rhoamb, rhoinit, rhomin,
& rhomax, preamb, preinit, premin, premax,
& vamb, vinit, vmin, vmax, grav

runname = ***** Run name entered on the command line
INIT_echo = .True. All INIT_DAT input variables will be echoed in log.
gamma0 = -1.0 Input required
 Assumed constant adiabatic gas constant

rhoamb = 1.2260e-3 Ambient mass density - STP air (gm/cc)
rhoinit = 5.3000e-4 Uniform initial density (gm/cc)
rhomin = 1.2260e-5 Minimum possible mass density (gm/cc)
rhomax = 1.2260e-4 Maximum allowable mass density (gm/cc), not used
preamb = 1.01325e+6 Ambient pressure - STP Air (erg/cc = dyne/cm^2)
preinit = 2.7000e+6 Uniform initial pressure (erg/cc = dyne/cm^2)
premin = 1.01325e+5 Minimum possible pressure (erg/cc = dyne/cm^2)
premax = 1.01325e+6 Maximum allowable pressure, not used
vamb(i) = 0.0 Ambient (background/far field) velocity (cm/s)
vinit(i) = 0. Uniform initial velocity (cm/s)
grav(i) = 0.0 Gravity (external acceleration)

vmin = 0.0 Minimal physical velocity
vmax = 3.0e+09 Maximal velocity - Speed of light

CHEM_DAT Input Variables

NameList /CHEM_dat/ Chem_echo, Run_name, Chem_doc, Chem_rec

runname = ***** Run name entered on the command line
CHEM_echo = .True. All CHEM_DAT input variables will be echoed in log.
Chem_rec(i) = ''

All values set to look like a namelist, not fortran code

FFIM_DAT Input Variables

NameList /FFim_DAT/ runname, FFIM_echo, FFIM_doc, FFIM_rec

runname = ***** Run name entered on the command line
FFIM_echo = .True. All FFIM_DAT input variables will be echoed in log.
FFIM_doc,
FFIM_rec

All values set to look like a namelist, not fortran code

STD_DAT Input Variables

NameList /STD_DAT/ runname, STD_echo, xs_pool, N_xs, st_ijk, N_st, N_stp,
& N_sts, N_stc, cp_ijk, N_cp, cv_ijk, N_cv, sp_ijk, N_sp

runname = ***** Run name entered on the command line
STD_echo = .True. All STD_DAT input variables will be echoed in log.
N_xs(i) = 0 ??????????????
N_st(i) = 0 ??????????????
N_stp = 1 ??????????????
N_sts = 1 ??????????????
N_stc = 1 ??????????????
N_sp = 0 ??????????????
N_cp = 0 ??????????????
N_cv = 0 Number of control volumes

xs_ijk(i,k) = -1 ??????????????
st_ijk(k,i) = 0 ??????????????
sp_ijk(k,i) = 0 ??????????????
cv_ijk(j,k,i) = 0 ??????????????
cp_ijk(j,k,i) = 0 ??????????????

Appendix C: Definition and Initialization of *runname.vi* Variables

The FAST3D visualization program VOYEUR requires many input values to be specified by the user. These input data values are all given in file *runname.vi*. As with FAST3D, some of the information in data file *runname.gi*, namely the grid information, is also needed. Some of the variables in *runname.vi* are local variables for only one or two subroutines; others are 'Fortran common block' variables. The variables in common blocks used by VOYEUR are declared in include file *gvce_blk.h* located in the directory *gridvce_vno*, and files *voy_admn.h*, *voy_grid.h*, *voy_pixels.h*, *voy_pvadr.h*, *voy_size.h* located in the directory *voyeur_vno*. Common block and local variables describing similar properties are combined into lists of variable names using the NAMELIST statement in the ????? *gridvce.f*, *voyeur.f*, *geometry.f_probname* programs. There are six of these namelists used exclusively by FAST3D in addition to the three namelists DOMAIN_DAT, SIZES_DAT, GRID_DAT, and GEOM_DAT read from *runname.gi* because the grid and geometry information is also needed by FAST3D.

There are six of namelists used exclusively by VOYEUR; four other are described in the Appendix A (SIZES_DAT, GRID_DAT, GEOM_DAT):

VADMIN_DAT namelist variables specifying the administrative information and data necessary to run VOYEUR. This namelist is read in file *gridvce.f*, and the variable default values are specified in file *gdat_init.f*.

XSEC_DAT namelist variables convey the information that defines the cross-sections that FAST3D must output, i.e. the coordinate system location and stretching, and the underlying structured grid grid specifications. This namelist is read in file ??????.f and the variable default values are specified in file ??????.f.

GLOBAL_DAT namelist variables convey the default data for all plots such as the number of pixels in the obrder, the overall magnification factors, etc. Each plot can separately override these values but this gives a convenient way to initialize a number of variables at once. This namelist is read in file ??????.f and the variable default values are specified in file ??????.f.

PLOT_DAT namelist variables convey the information that determines the physical variable and layout of each of the up to ????? plots permitted by VOYEUR. This namelist is read in file ??????.f and the variable default values are specified in file ??????.f.

CONTOUR_DAT namelist variables convey the data determining how each physical variable will be plotted, the contour levels to be used, the geomery of the bodies (if any) in the computational domain. This namelist is read in file ??????.f and the variable default values are specified in file ??????.f.

VIDEO_DAT namelist variables convey the data specifying the frequencies, delays, repeat frame counts, etc. to make a video from FAST3D through VOYEUR while the simulation is progressing. This obviates the need to store off and post process a

lot of information. This namelist is read in file `?????.f` and the variable default values are specified in file `?????.f`.

As with the data in files `runname.gi`, and `runname.ni`, the Fortran namelist utility is used to read this data into VOYEUR. Description of the variables contained in each of the six namelists in the input data file `runname.ni` and the default values for these variables follow. The comma is included after each default value in these three appendices to remind the reader that this comma is the variable separator for the NAMELIST statement.

VADMIN_DAT Input Variables

```
NAMELIST /VADMIN_DAT/ runname, VADMIN_echo, arch, src_remote, src_path,
&      dst_path, xsec_fn, Sleep_time, keep_tfs, mn_frm, mx_frm,
&      Mx_try, L_xsh, L_video, vid_time, vid_fram, vid_freq,
&      xs_bytswp
```

```
runname = ***** Run name entered on the command line
VADMIN_echo = .True. All VADMIN_DAT input variables will be echoed in log.
Sleep_time = 5      Sleep time between reading of .xsh file
mn_frm = 1          Minimum frame number
mx_frm = 5000       Maximum frame number
Mx_try = 500        Maximum number of times to look for .xsh file
src_remote = ''     Name of the remote machine where the dump file resides
src_path = ''       Path to the dump file on remote machine
dst_path = '        Destination path on local machine to put dump file
vid_time = '00180000' Video Timing
L_video = .false.   Logical for video recording
vid_freq = 1        ??????????
vid_fram = 3        ??????????
L_xsh = .true.      ??????????
L_psync = .False.   ??????????
Lpn = -1            ??????????
Np = 1              ??????????
```

XSEC_DAT Input Variables

The VOYEUR input data set below defines the cross sections (planes) from the set of FAST3D programs for subsequent online plotting. These planes are stored in the variable buffer 'var_buf' in sequence. The defining cell indices and coordinate locations are put into 'ijk_val' and 'xyz_val' respectively. In 'var_buf' are the YZ planes with indices i1, i2, ..., i10 at x1, x2, ... x10. The XZ planes follow with indices j1, j2, ..., j10 at y1, y2, ... y10. The XY planes follow with indices k1, k2, ..., k10 at z1, z2, ... z10. When 'ijk_val(csec_ind) = 0' on input, the cell interface arrays 'xbn', 'ybn', or 'zbn' are used to find the cell containing 'xyz_val(csec_ind)'. Up to ten planes may be specified for each cross-section direction.

xsec_ind = 10, 11, ..., ni, 20, 21, ..., nj, 30, 31, ..., nk
planes at i1, i2, ..., in, j1, j2, ..., jn, k1, k2, ..., kn

Namelist /XSEC_DAT/ runname, XSEC_echo, ijk_pp, xyz_pp

Data ijk_pp /MPlns*0/, xyz_pp /MPlns* 0.0/, New_pln /Mplns*0/

runname = ***** Run name entered on the command line
XSEC_echo = .True. All XSEC_DAT input variables will be echoed in log.
ijk_val = 20, 25, 30, 0, 0, 0, 34, 36, 0,
xyz_val = 0.0, 0.0, 0.0, -10.0, 0.0, 10.0, 0.0, 0.0, 5.0,

GLOBAL_DAT Input Variables

Namelist /GLBL_DAT/ runname, GLOBAL_echo, Macropx, NMppGrid, NPixBorder,
& pxmx, D_flpH, D_flpV, keep_wndos, Cbr_Length, Cbr_width,
& Tic_length, Tic_width

runname = ***** Run name entered on the command line
GLOBAL_echo = .True. All GLOBAL_DAT input variables will be echoed in log.

Macropx = ,
NMppGrid = ,
NPixBorder = ,
pxmx = ,
D_flpH = ,
D_flpV = ,
keep_wndos = ,
Cbr_Length = ,
Cbr_width = ,
Tic_length = ,
Tic_width = ,

PLOT_DAT Input Variables This needs work to put it in some sort of format like the rest

Namelist /PLOT_DAT/ runname, PLOT_echo, plt_irec, ????????

runname = ***** Run name entered on the command line
PLOT_echo = .True. All PLOT_DAT input variables will be echoed in log.
plt_xsec = 1001 . . . 100i, 2001 . . . 200j, 3001 . . . 300k

Data plt_xsec / Mw*0 /

```

Data plt_freq / Mw*0 /, Strchpix / Mw*.False. /
Data zoom_hor / Mw*0 /, pan_hor1 / Mw*0 /, pan_horN / Mw*0 /
Data zoom_ver / Mw*0 /, pan_ver1 / Mw*0 /, pan_verN / Mw*0 /
Data plt_loch / Mw*0 /, plt_locv / Mw*0 /, plt_name / Mw*' ' /
Data Macropx / 2, 2, 2 /, keep_wndos / 1 /
Data NMppGrid, NPixBorder / 0, 1 /
Data D_flpH / .False. /, D_flpV / .True. /
Data CNTR_echo, GLBL_echo, PLOT_echo, XSEC_echo /4*.true./
Data pxmx /248/
Data Cbr_length, Cbr_width /225, 20/
Data Tic_length, Tic_width / 8, 4/

```

\$PLOT_DAT

```

Run_name = 'shp01'=00, PLOT_echo = T,
plt_freq = 0, 10, 0, 0, 0, 0, 0, 0,
plt_name = 'rho', 'Trc', 'Vx', 'Vy', 'Vx', 'Tshp', 'Tshp',
xsec_ind = 31, 31, 31, 20, 10, 10, 31, 31,
colorbar = 0, 1, 2, 3, 4, 0, 1, 2,
FixGrid = F, F, F, T, F, F, F, F,
zoom_hor = 2, 3, 2, 2, 4, 2, 7, 7,
pan_hor1 = 0, 0, 0, 107, 10, 0, 70, 70,
pan_horN = 0, 0, 0, 180, 39, 0, 120, 120,
zoom_ver = 2, 3, 2, 2, 4, 2, 7, 7,
pan_ver1 = 0, 0, 0, 7, 3, 1, 16, 40,
pan_verN = 0, 0, 0, 41, 15, 35, 40, 16,
plt_loch = 10, 25, 30, 10, 65, 45, 32, 34,
plt_locv = 40, 40, 25, 40, 20, 40, 32, 34,

```

```

MacropxX = 2, MacropxY = 2, MacropxZ = 2,
LMppGrid = F, LPixBorder = T, keep_wndos = 1,

```

\$END

```

c plt_freq = The timestep frequency of the selected plot in FAST3D
c      = Used as a toggle switch (do 'plt_name' if plt_freq.gt.0)
c plt_name = eg. 'abcd' Four character name selects 1 of 20 CNTR_DAT
c   = 'Vx', 'Vy', 'Vz', 'rho', 'erg', 'Pre', 'Tem', 'Tshp', 'Cs',
c     = 'Mach', 'Ptot', 'cv[i]#', 'Trc', 'NPf[i]', 'obj', 'dum1',
c xsec_ind = 10, 11, ..., 39 Selects data cross section from XSEC_DAT
c plt_loch = 0-99 10*plt_loch locates the window upper left corner X
c plt_loch = 0-99 1 0*plt_locv locates the window upper left corner Y
c colorbar = 0 no color bar plotted   = 1 color bar at picture bottom
c      = 2 (color bar at right) = 3 (at top)   = 4 (at left)
c FixGrid = T (fix stretched grid) = F (treat as if uniform grid)
c zoom_hor = Horizontal macropixel size, if 0 => NmcropxX or NmcropxZ
c pan_hor1 = 1-st index in horizontal window direction, if 0 => 1
c pan_horN = Last index in horizontal window direction, if 0 => NX?
c zoom_ver = Vertical macropixel size, if 0 => NmcropxY or NmcropxZ

```

c pan_ver1 = 1-st index in vertical window direction, if 0 => 1
 c pan_verN = Last index in vertical window direction, if 0 => NY?
 c MacropxX = 2 Default macropixel size for X coordinate scaling
 c MacropxY = 2 Default macropixel size for Y coordinate scaling
 c MacropxZ = 2 Default macropixel size for Z coordinate scaling
 c keep_wndos = 0 Closes the plot windows at the end of a run

CONTOUR_DAT Input Variables

Namelist /CONTOUR_DAT/ runname, CONTOUR_echo, ctr_irec

runname = '*****' ???????

CONTOUR_echo = .True., All CONTOUR_DAT input variables will be echoed in log.

??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written

VIDEO_DAT Input Variables

Namelist /VIDEO_DAT/ runname, VIDEO_echo, ????????????

runname = '*****' ???????

VIDEO_echo = .True., All VIDEO_DAT input variables will be echoed in log.

??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written
 ??????? = ??????? This comment needs to be written

Appendix D: Test Problem Data Sets and Selected Results

- 1. Blast Problems with Null Geometry**
- 2. Blunt Body and Jet Problems**
- 3. Muzzle Blast Problems**
- 4. Wedge/Cone Problems**
- 5. Circular Arc Airfoil Problems**
- 6. Multiple Ellipse Problems**

Reproduced directly below is an abbreviated (preliminary) input data file **elp_a1.vi** as it would appear to initialize VOYEUR with the test problem of four, overlapping, 3D ellipsoids suspended in space away from the boundaries of the computational domain. This example continues the development of the particular case used in the previous section

Appendix E: FAST3D File Contents and Subroutine Descriptions

The following lists the subroutine source files currently in lib/ and describes their function in a few words. This list a descriptive information is included for completeness and can be skipped by a first time reader of the manual.

The lib/src/ Directory Contents.

Dependencies.mk

Luns_Admin.f : Subroutine Luns_Admin
administers all logical unit numbers for FORTRAN programs if used exclusively to obtain Logical Unit values. To avoid conflict with implicit FORTRAN Logical Unit values such as standard in and standard out, only numbers in the range from 10 to 99 are returned.

Mem_Admin.f : Subroutine Mem_Admin
used administer dynamic memory allocation for FORTRAN programs. This subbboutine uses vendor specific means to allocateand release memory buffers.

cmnd_arg.f : Subroutine Cmnd_arg
employs non-standard unix utilities to obtain runtime arguments supplied on the command line.

geo_box_chk.f : Subroutine Body_Box
Initializes the time-saving check of 'boxes' around the bodies in complex geometries. Each call to Body_Box sets up one box but no more than 10 can be used

mesh_set.f : Subroutine mesh_set
initializes the FAST3D grid for the given Logical direction Ldir.

rd_wrt_rec.f : Subroutine Rd_Wrt_Rec

rd_wrt_rec.f : Subroutine Rd_buf
Read a sequential string of fixed length records from a file.

rd_wrt_rec.f : Subroutine Wrt_buf
Write sequential string of fixed length records to a file.

rnm_chk.f : Function RNM_chk
This subroutine checks the namelist record input *runname* against that submitted with the job (i.e. for GRIDVCE, VOYEUR, or FAST3D) and sets the value RNM_chk to non zero if they are inconsistent.

upr_case.f : Function UPR_case
utility to convert alphabetic characters to uppercase.

The fast3d_vno/ Directory Contents.

bndcgasd.f : Subroutine bndcgasd

boundcon sets the boundary conditions for the planes version of gasdyn is used in the fast3d code. There is space for up to 16 different boundary conditions, all selected by the two integers bc1 and bcN fed into gasdyn defining what is to be done at the ends of each line segment integrated.

cell_va.f : Subroutine CELL_VA

This routine scales the FCT cell volumes & face areas factors by the actual size of the cells.

cfl_cn.f : Subroutine CFL_cn

This subroutine evaluates the current Courant, Friedrichs, & Lewy (CFL) condition number for the active flow field. This routine provides a means for maintaining stability with an adaptive time-step and a diagnostic under constant time-step conditions.

cnsrv_chk.f : Subroutine cnsrv_chk

This routine performs a complete flow field volume - variable integral as runing diagnostic. Such items as MASS & ENERGY should be conserved to algorithm accuracy. This routine provides a method for checking this as needed.

cont_vol.f : Subroutine Cont_Vol

This subroutine will calculate the mass, momentum, and energy fluxes through the planes which form the control volume (CV) faces. The planes are located at the given i, j, k indices. The momentum flux net force) is positive in the positive grid direction. The areas used are between given cell and the lower index cell.

date_times.f : Subroutine XDate (Date_buf)

date_times.f : Subroutine XTime (Time_buf)

date_times.f : Subroutine Wall_Time

dmp_sd.f : Subroutine Std_files

writes selected station data from the full 3D data to disk at any stage of the running transpose

dmp_xs.f : Subroutine Dmp_xs

writes 2D cross-sections from the full 3D data to disk at the {I} stage of the running transpose.

fdat_init.f : Subroutine fdat_init

This subroutine is used to initialize optional input data values that reside in various common blocks. Normally this would be done with a Block Data program. But Block Data programs never get

referenced from the main code so that a loader building the executable from an object library may never load it without vendor specific commands being present in the build procedure.

flow_field.f : Subroutine Flow_field
Problem specific subroutine to initialize FAST3D flow field variables as a function of position

flow_mods.f : Subroutine ff_mods

gasdyn.f : Subroutine gasdyn
This version for the planes code has chemistry and shocks, entering through the conserved energy density with the pressure found from ideal gas equation of state:

$$pre = (\text{gamma} - 1)(\text{erg} - 0.5 * \text{rho} * (\text{vx} ** 2 + \text{vy} ** 2 + \text{vz} ** 2))$$

'gasdyn' integrates along the line or segment of a line that has been determined from the geometry in xyzfct.

gsd_ext.f : Subroutine Coefix
Check if velocities are too large, or the density is too low or negative, or the pressure is too low via EPSH, rhomin, premin, etc and modify the FCT coefficients accordingly

gsd_ext.f : Subroutine Upstream

gsd_ext.f : Subroutine Antidiff

idp_chem.f : Subroutine Ichem_parms
null routine, use to satisfy fast3d built in external reference

idp_chem.f : Subroutine chemdyn
null routine, use to satisfy fast3d built in external reference

ijk_fct.f : Subroutine IJK_fct
This routine selects 1-D arrays from the parallel sub-blocked 3-D data storage Dat_buf for time advancement. Selection is based on values of Ldir, (i.e. values of 1, 2, & 3, specify I, J, & K, directions respectively). Subroutines get_ijk_cols & put_ijk_cols are used to retrieve and store column data in the Dat_buf storage area. Subroutine gasdyn is used to call the lcpfct routines to integrate in each of the three axis directions. Lsg1 and LsgN are determined from the embedded values of next and prev and specify the range of integration for each of the identified integration line segments. The values and bc1 and bcN (each from 1 to 16) specify the respective boundary conditions. The I, J and K momenta are separately identified in gasdyn to resolve confusion as to parallel and perpendicular components to the walls.

ijk_fct.f : Subroutine Get_Put_ijk
Entry get_ijk_cols extracts data from the 3-d arrays according to the specified direction of the integration and stores the data in 1D scratch arrays for integration. Entry put_ijk_cols replaces data into the 3-d arrays according to the specified direction of the integration.

init_geom.f : Subroutine Init_geom
Problem specific subroutine to initialize FAST3D flow field geometry as a function of position (i_pos, j_pos, k_pos). A simple staircase implimentation based on the returned value of IN_BODY from the problem specific subroutine GEOMETRY is all that is currently available.

inital.f : Subroutine Inital
This routine initializes the ship hull turbulent flow problem with the physical parameters set in /init3/ and /init4/ in blokdata.f.

int_vol.f : Subroutine int_vol
This subroutine calculates the total quantity of the varialbe within the volume bounded by the i, j, k indices

lcpfct.f : Subroutine LCPFCT
This routine solves generalized continuity equations of the form
$$dRHO/dt = -div (RHO*V) + SOURCES$$
in the user's choice of Cartesian, cylindrical, or spherical coordinate systems. A facility is included to allow definition of other coordinates. The grid can be Eulerian, sliding rezone, or Lagrangian and can be arbitrarily spaced. The algorithm is a low-phase-error FCT algorithm, vectorized and optimized for a combination of speed and flexibility. A complete description appears in the NRL Memorandum Report (1992), "LCPFCT - A Flux-Corrected Transport Algorithm For Solving Generalized Continuity Equations".

lcpfct.f : Subroutine MADFCT (remove completely from FAST3D)

lcpfct.f : Subroutine MAKEGRID
This Subroutine initializes geometry variables and coefficients. It should be called first to initialize the grid. The grid must be defined for all of the grid interfaces from I1 to INP. Subsequent calls to VELOCITY and LCPFCT can work on only portions of the grid, however, to perform restricted integrations on separate line segments.

lcpfct.f : Subroutine VELOCITY
This subroutine calculates all velocity-dependentcoefficients for the LCPFCT and CNVFCT routines. This routine must be called before either LCPFCT or CNVFCT is called. MAKEGRID must be called earlier to set grid and geometry data used here.

lcpfct.f : Subroutine SOURCES
This Subroutine accumulates different source terms.

lcpfct.f : Subroutine CNVFCT
This routine solves an advective continuity equation of the form
$$dRHO/dt = -V*grad(RHO) + SOURCES$$
in the user's choice of Cartesian, cylindrical, or spherical coordinate systems. A facility is included to allow definition of other coordinates. The grid can be Eulerian, sliding rezone, or Lagrangian and can be arbitrarily spaced. The algorithm is a low-phase-error FCT algorithm, vectorized and optimized for a combination of speed and flexibility. A complete description

appears in the NRL Memorandum Report (1992), "LCPFCT - A Flux-Corrected Transport Algorithm For Solving Generalized Continuity Equations".

lcpfct.f : Subroutine CONSERVE

This routine computes the ostensibly conserved sum. Beware your boundary conditions and note that only one continuity equation is summed for each call to this subroutine.

lcpfct.f : Subroutine COPYGRID

This Subroutine makes a complete copy of the grid variables defined by the most recent call to MAKEGRID from cell I1 to IN including the boundary values at interface IN+1 when the argument MODE = 1. When MODE = 2, these grid variables are reset from common block OLD_GRID. This routine is used where the same grid is needed repeatedly after some of the values have been over-written, for example, by a grid which moves between the halfstep and the whole step.

lcpfct.f : Subroutine NEW_GRID

This Subroutine initializes geometry variables and coefficients when the most recent call to MAKEGRID used the same set of values RADHO and only the new interface locations RADHN are different. NEW_GRID is computationally more efficient than the complete grid procedure in MAKEGRID because several formulae do not need to be recomputed. The grid should generally be defined for the entire number of grid interfaces from 1 to INP, however subsets of the entire grid may be reinitialized with care.

lcpfct.f : Subroutine RESIDIFF

Allows the user to give FCT some residual numerical diffusion by making the anti-diffusion coefficient smaller.

lcpfct.f : Subroutine SET_GRID

This subroutine includes the radial factor in the cell volume for polar coordinates. It must be preceded by a call to MAKE_GRID with ALPHA = 1 to establish the angular dependence of the cell volumes and areas and a call to COPY_GRID to save this angular dependence. The angular coordinate is measured in radians (0 to 2 pi) in cylindrical coordinates and cos theta (1 to -1) in spherical coordinates. SET_GRID is called inside the loop over radius in a multidimensional model to append the appropriate radial factors when integrating in the angular direction.

lcpfct.f : Subroutine ZERODIFF

This Subroutine sets the FCT diffusion and anti-diffusion parameters to zero at the specified cell interface to inhibit unwanted diffusion across the interface. This routine is used for inflow and outflow boundary conditions. If argument IND is positive, the coefficients at that particular interface are reset. If IND is negative, the list of NIND distinct INDICES are used to reset coefficients at NIND different interfaces.

lcpfct.f : Subroutine ZEROFLUX

This Subroutine sets all the velocity dependent FCT parameters to zero at the specified cell interface to inhibit transport fluxes AND diffusion of material across the interface. This routine is needed in

solid wall boundary conditions. If IND is positive, the coefficients at that particular interface are reset. If IND is negative, the list of NIND distinct INDICES are used to reset coefficients at NIND different interfaces.

lp_chem.f : Subroutine Ichem_parms

lp_chem.f : Subroutine Ichem_vars

lp_chem.f : Subroutine Tau_scl

This routine calculates the Chemical Model dependent scaling factor for TAU the local induction time.

lp_chem.f : Subroutine chemdyn

This routine performs any chemistry reactions needed.

lp_chem.f : Subroutine Indpm

This routine performs a global chemistry approximation to the case:

npr(1)*Fuel + npr(2)*Oxidant + Diluent -->

Npr(3..)*Prods(1..) + Diluent

cv(i,1) --> fuel

cv(i,2) --> oxidizer (If any)

cv(i,3...) --> products

cv(i,Ncv-1) --> diluent

cv(i,Ncv) --> amount of induction time that has expired.

lp_chem.f : Subroutine Taupt_RDX

lp_chem.f : Subroutine Taupt_CH4O2

lp_chem.f : Subroutine Taupt_H2O2

pdat_utils.f : Subroutine Blk_tp

This subroutine is used to perform the 3D sub-block swapping at any stage of the Running Transpose (RT). There are 6 stages of the RT which take two time steps to complete. Each level or stage is reach-able only from the previous or adjacent of the RT. Even though each stage is reversible, this capability is not used in the current version of FAST3D. The logic in this subroutine works most efficiently between pairs of nodes on the Intel hypercube. Although probably not optimum, it does work well on other machines such as the CM5 and the Intel Delta and Paragon. The value of TP_stage determines the current stage of the RT.

pdat_utils.f : Global_indices

Initialize Global Index offsets (ig_ofs, jg_ofs, kg_ofs) as a function of the virtual node number and the current stage of the running transpose.

pdat_utils.f : Subroutine Get_Pv_xs

Get selected cross-sectional Primary and Chemical species data at the current stage (Ndiim) of the transpose.

pdat_utils.f : Subroutine GS_Pvs
Gather-Scatter primary-variable column data for each stage of transpose.

pdat_utils.f : Subroutine GS_Gvs
Gather-Scatter Geometrical feature indicator (i.e. Lgeom, Next, Prev) column data for each stage of transpose.

pdat_utils.f : Subroutine GS_Gfdr
Get or store Geometrical feature data record as indicated by Lgm for each stage of transpose as indicated by Ldir.

periodic_bcs.f: Subroutine Per_setup
This subroutine performs the wrapping required for the periodic boundary condition when another boundary due to an embedded object is present between the to periodiends. All the active data from $D(\text{per_bc1})$ through $D(\text{per_bc1} + \text{Nsg1})$ is moved to $D(\text{per_bcN} + 1)$ through $D(\text{per_bcN} + 1 + \text{Nsg1})$. This allows the normal lcpfct pencil integration package to advance the periodicconditions correctly.

periodic_bcs.f: Subroutine Per_reset
This subroutine performs the unwrapping required for the periodic boundary condition when another boundary due to an embedded object is present between the to periodiends. All the advanced data from $D(\text{per_bcN} + 1)$ through $D(\text{per_bcN} + 1 + \text{Nsg1})$ is returned to $D(\text{per_bc1})$ through $D(\text{per_bc1} + \text{Nsg1})$. This allows the normal lcpfct pencil integration package to advance the periodic conditions correctly.

pln_flx_pre.f : Subroutine pln_flx_pre
This subroutine calculates the flux through a plane or planar section. The flux is positive in the positive grid direction. The areas used are between given cell and the lower index cell.

rstrt_dmp.f : Subroutine Dumper
iPSC/860 specificcode. Fully parallel and scalable. Subroutine to write to disk all the necessary data so that the simulation can be restarted. Dumping occurs only at the X contiguous stage of the running transpose, at multiples of dump_freq steps (+1), except on the 'minstep'. Restarting from the physical variable dump occurs whenever istep = minstep.

rstrt_dmp.f : Subroutine Restart
iPSC/860 specificcode. Fully parallel and scalable. Subroutine to read from disk all the necessary data so that the simulation can be restarted. Dumping occurs only at the X contiguous stage of the running transpose, at multiples of dump_freq steps (+1), except on the 'minstep'. Restarting from the physical variable dump occurs whenever istep = minstep.

spot_chk.f : Subroutine Spot_chk

iPSC/860 specific code. Fully parallel and scalable. Subroutine to write selected station data from the full 3D data to disk at the initial stage of the running transpose.

vce.f : Subroutine FEATURES

This routine modifies the cell area and volume data for feature cell geometry and copies the boundary source information into geosrc.

vce.f : Subroutine SML_CELL

This routine averages the FCT transported diffused densities so adjacent cells will be lumped together when adjacent cells differ greatly in size

vce.f : Subroutine bsregasd

This version for the fastflow fast3d code has chemistry and shocks.

The gridvce_vno/ Directory Contents

GRIDVCE is the FAST3D grid generator program. It is required when more complicated geometries with curved, smooth surfaces must be simulated. Documentation for the subroutines in this source directory will be provided with the next release of the FAST3D suite.

gdat_init.f : Subroutine grd_dat_int

geometry.f : Subroutine

initalg.f : Subroutine initalg

initalg.f_sav : Subroutine initalg

The voyeur_vno/ Directory Contents

VOYEUR is the FAST3D online graphics program. It provides a fast, moderately general capability to diagnose ongoing FAST3D simulations graphically. Documentation for the subroutines in this source directory will be provided with the next release of the FAST3D suite.

f_write.f: Subroutine f_write

geometry.f: Subroutine Geometry

get_xsec.f: Subroutine GET_XSEC

graph_io.f: Subroutine graph_io

graph_io.f: Subroutine Cntr_set

graph_io.f: Subroutine PAN_RANGE

header.f_old: Subroutine Header

maskout.f: Subroutine maskout

maskout.f: Subroutine STRGRID

phys2ind.f: Subroutine phys_to_index

pixelate.f: Subroutine PIXELATE

pixelate.f: Subroutine INTERPOLATE

pixelate.f: Subroutine STRETCH

pixelate.f: Subroutine UNIFORM

pre_plot.f: Subroutine pre_plot

pre_plot.f: Subroutine Cbar_

usr_plot.f: Subroutine usr_plot (

usr_plot.f: Subroutine CREEPY

usr_plot.f: Subroutine Nxt_Priv

vdat_init.f: Subroutine vdat_init

voyeur.f: Subroutine VP_sync

wndo_dat.f: Subroutine wndo_dat

wndo_dat.f: Subroutine FLIP_PIX (

wndo_drw.f: Subroutine wndo_drw

wndo_lbl.f: Subroutine wndo_lbl

Appendix F: Scripts used for running FAST3D

build_fast3d

The FAST3D suite of programs, *fast3d*, *gridvce*, and *voyeur*, and their support subroutines use dynamic memory allocation for variable length arrays so they generally do not need to be recompiled and linked when the problem size changes. However, user-provided subroutines for geometry, flow initialization, and flow modification will have to be compiled and linked whenever changes are made to these subroutines. In order to facilitate the process of building or updating the object libraries, as well as the executable files, a script called *build_fast3d* has been written as a simple user interface to the appropriate makefile.

Help for *build_fast3d* is obtained by typing the command with no arguments. For example,

```
% build_fast3d
```

returns the following to the screen:

```
Usage: build_fast3d [-fF] [-gG] [-vV] [-cC] [-uU] [-ot] prob_name
```

```
-f/F: Update/Rebuild (f)ast3d object library  
-g/G: Update/Rebuild (g)ridgen object library  
-v/V: Update/Rebuild (v)oyeur object library  
  
-c/C: Update/Rebuild (c)ommon object library  
-u/U: Create/Rebuild (u)ser prob_name dependent library  
  
-o: Site/Organization name? Default: [LCP]  
-t: Target Architecture? Default: [SGI64]
```

Using the script is straightforward: Simply choose your option(s) and *prob_name*. For example, to compile and link the grid generator GRIDVCE with the new or modified geometry subroutine, or to compile the entire GRIDVCE executable if it is being compiled for the first time, type at the prompt:

```
% build_fast3d -g bluntbody
```

at the prompt. The script determines which files it needs based on the user supplied *prob_name*, determine if any of these files have been updated, and recompile the executable code(s). The executable produced with the above argument is *gridvce_bluntbody*. Of the *LCP&FD machines*, *henson* (SGI64) or *oscar*(RS6K) are the recommended machines to compile the *gridvce* on.

As another example, to compile and link FAST3D with the new or modified geometry subroutine associated with the *bluntbody* problem, type:

```
% build_fast3d -f bluntbody
```

at the prompt.

To compile and link VOYEUR with the new or modified geometry subroutine type at the prompt:

```
% build_fast3d -v bluntbody
```

The executable produced with the above argument is `voyer_bluntbody`. Of the *LCP&FD machines*, henson (SGI64) or oscar(RS6K) are the recommended machines to compile voyeur on.

Any combination of the above commands may also be used. For example, if you wish to compile and link VOYEUR and FAST3D with the new or modified geometry subroutine, type:

```
%> build_fast3d -fv bluntbody
```

The executables produced with the above argument are `fast3d_bluntbody` and `voyer_bluntbody`.

Once the appropriate executables have been created, running GRIDVCE, FAST3D, or VOYEUR is accomplished by using the scripts **run_gridvce**, **run_fast3d**, or **run_voyeur**, respectively.

run_gridvce

The `run_gridvce` script runs the appropriate grid generator for a specific problem, `gridvce_probname`. The `build_fast` script, described above, generates this executable program, which once compiled, resides in the `$FAST3D/bin/$target_arch` directory. The last argument to the script is the specific run or subproblem name which determines the names of the input files to be used in the directory `$FAST3D/runs/probname`. By typing `run_gridvce` at the prompt, with no arguments, one gets the usage message:

```
% run_gridvce
```

Usage: `run_gridvce [-acei] run_name`

- a: Append the log file to a previous log file
- c: DO NOT cd to the appropriate runs directory based on `run_name`
- e: Send standard error to the log file
- i: Run GRIDVCE interactively (i.e. not in the background)

The only required option to this script is the `run_name`. If the command is run in the background (the default), then output from the `run_gridvce` script and output of the GRIDVCE program is put into the file `run_name.glog`. This file does not actually contain the grid data, but contains information such as input data and number of feature cells detected. This file may be viewed by using the Unix `tail` command or any editor. If the command is run interactively, then this information is written to the screen.

run_fast3d

The `run_fast3d` script runs the appropriate flow solver for a specific problem, `fast3d_probname`. The `build_fast` script, described above, generates this executable program, which once compiled, resides in the `$FAST3D/bin/$target_arch` directory. The last argument to the script is the specific run or subproblem name which determines the names of the input files to be used in the directory `$FAST3D/runs/probname`. By typing `run_fast3d` at the prompt, with no arguments, one gets the usage:

```
%> run_fast3d
```

Usage: `run_fast3d [-acehijpnxs] run_name`

- a: Append the log file to a previous log file
- c: DO NOT cd to the appropriate run_name directory
- e: Send standard error to the log file
- h: System name (Default: jim-henson)
- i: Run FAST3D interactively (i.e. not in the background)
- j: Add run information to the journal (`$FAST3D/runs/run_name.jnl`)
- p: Number of Processors (Default: [1])
- n: npri value, if required
- s: Submit when processors are available
- x: Use the following executable name, `fast3d_[name]`

The only required options to this script are `-p` and `run_name`. If, however, the job is being submitted to the intel (iPSC/860) from the frontend (lcp), then the `-h` option must be used. This option is used to determine which SRM to run the job on. The other two options indicate how many processors to run the job on and `run_name` of the job you wish to run. For example, if you wish to run the `blast_2d` casewith 16 processors on the iPSC/860 with SRM name `wonka` you would type:

```
% run_fast3d -h cerberus -p 16 blast_2d
```

This command would then submit the FAST3D executive code `fast3d_blast` to the iPSC/860 with SRM name `wonka` on 16 nodes and use the input files `blast_2d.gi`, `blast_2d.ni` and `blast_2d.vi`.

Occasionally, it may be desirable to run an executable which is different than the `probname`. For example, if you are in `blast` directory and wish to submit a job using the executable for the `blntbdy` problem simply the `"-x"` option for `run_fast3d`. For this example one simply types

```
%> run_fast3d -h cerberus -p 16 -x blntbdy blast_2d
```

This command would then submit the FAST3D executable code `fast3d_blntbdy` to the iPSC/860 with SRM name `wonka` on 16 nodes and use the input files `blast_2d.ni` and `blast_2d.vi`.

All output of the **run_fast3d** script, which includes output of the FAST3D program, is put into the file *runname.flog*. This file may be viewed by using the unix tail command or any editor.

The "-j" option will add job information (time of job submission, job name, run name, etc.) into a journal file with name *runname.jnl* within the current *probname* directory. This file is simply of journal of what jobs have been submitted and may be useful since it summarizes the runs completed.

run_voyeur

The *run_voyeur* script runs the FAST3D graphics program, VOYEUR, either interactively or in the background. The executable program submitted is *voyeur_probname* where *probname* is the generic problem name which determines the specific user-supplied routine geometry.f. The *build_fast* script, described above, generates this executable program, which once compiled, resides in the *\$FAST3D/bin/\$target_arch* directory. The last argument to the script is the specific run or subproblem name which determines the names of the input files to be used in the directory *\$FAST3D/runs/probname*. By typing *run_voyeur* at the prompt, with no arguments, one gets the usage:

```
%> run_voyeur
```

```
Usage: run_voyeur [-acehi] run_name
```

- a: Append the log file to a previous log file
- c: DO NOT cd to the appropriate runs directory based on run_name
- e: Send standard error to the log file
- h: Voyeur's source files remote name (Default: LOCAL)
- i: Run Voyeur interactively (i.e. not in the background)

All output of the **run_voyeur** script, which includes output of the VOYEUR program, is put into the file *runname.vlog*. This file may be viewed by using the unix tail command or any editor.

mknewprob

The script *mknewrun* simplifies the creation of new run and/or user subdirectories, as well as the associated input files and/or source code. The script takes as arguments the *run_name* you wish to create, *NEW_run_name*, and the *run_name* that you wish to base it on, *OLD_run_name*. It then makes the new problem directories in *\$FAST3D/runs* and *\$FAST3D/user_src*, based on the *problem_name*, and then copies the old files, modifying the names of the files and, if necessary, the text within the files, to the appropriate directories. If the option argument "-r" is specified, then only new run files (*.i) files are created in the appropriate directory.

Help for *mknewrun* is obtained by typing the command at the prompt:

% **mknewprob**

Usage: mknewprob [-r] NEW_run_name OLD_run_name

-r: Make a new run only (i.e. create only *.i files)

As an example, if you wish to create a new problem cone_r1 and base it on the old problem bluntbody_2d, type:

```
%> mknewprob cone_r1 bluntbody_2d
```

```
New Problem: cone
```

```
New Run: cone_r1
```

```
Old Problem: bluntbody
```

```
Old Run: bluntbody_2d
```

```
Making directories $FAST3D/runs/cone $FAST3D/user_src/cone
```

```
Creating cone.*i files in /u/guests/lind/fast3d/runs/cone
```

```
- cone_r1.gi
```

```
- cone_r1.ni
```

```
- cone_r1.vi
```

```
Copying flow_field.f_bluntbody to flow_field.f_cone
```

```
Copying geometry.f_bluntbody to geometry.f_cone
```

Done

Note that the directory name always corresponds to the part of the problem name preceding the "_". Hence, although the full problem names are bluntbody_2d, bluntbody_3d, etc., they all reside in the directory named \$FAST3D/runs/bluntbody. Also, the file paths defined in the *.ni and *.vi files, file_path and src_path respectively, reflect the name of the problem subdirectory to runs/, not the particular run name.

As another example, consider the scenario in which you wish to create another run bluntbody_r1 and base it on the run bluntbody_3d. This is done by simply using the “-r” option of the mknewrun command, as follows:

```
%> mknewprob -r bluntbody_r1 bluntbody_3d
```

```
New Problem: bluntbody
```

```
New Run: bluntbody_r1
```

```
Old Problem: bluntbody
```

```
Old Run: bluntbody_3d
```

```
Creating bluntbody.*i files in /u/guests/lind/fast3d/runs/bluntbody
```

```
- bluntbody_r1.gi
```

```
- bluntbody_r1.ni
```

- bluntbody_r1.vi

Done