OpenFOAM

The Open Source CFD Toolbox

User Guide

Version 1.4.1
1st August 2007

Typeset in LaTeX.

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on

the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same

name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have

received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Trademarks

ANSYS is a registered trademark of ANSYS Inc.
CFX is a registered trademark of AEA Technology Engineering Software Ltd.
CHEMKIN is a registered trademark of Sandia National Laboratories
CORBA is a registered trademark of Object Management Group Inc.
openDX is a registered trademark of International Business Machines Corporation
EnSight is a registered trademark of Computational Engineering International Ltd.
AVS/Express is a registered trademark of Advanced Visual Systems Inc.
Fluent is a registered trademark of Fluent Inc.
GAMBIT is a registered trademark of Fluent Inc.
Fieldview is a registered trademark of Intelligent Light
Icem-CFD is a registered trademark of ICEM Technologies GmbH
I-DEAS is a registered trademark of Structural Dynamics Research Corporation
JAVA is a registered trademark of Sun Microsystems Inc.
Linux is a registered trademark of Linus Torvalds
MICO is a registered trademark of MICO Inc.
ParaView is a registered trademark of Kitware
STAR-CD is a registered trademark of Computational Dynamics Ltd.
UNIX is a registered trademark of The Open Group

Contents

Copyright Notice **U-2**

GNU Free Documentation Licence **U-3**
1. APPLICABILITY AND DEFINITIONS . . . U-3
2. VERBATIM COPYING . . . U-4
3. COPYING IN QUANTITY . . . U-4
4. MODIFICATIONS . . . U-5
5. COMBINING DOCUMENTS . . . U-6
6. COLLECTIONS OF DOCUMENTS . . . U-7
7. AGGREGATION WITH INDEPENDENT WORKS . . . U-7
8. TRANSLATION . . . U-7
9. TERMINATION . . . U-7
10. FUTURE REVISIONS OF THIS LICENSE . . . U-8

Trademarks **U-9**

Contents **U-11**

1 Introduction **U-17**

2 Tutorials **U-19**
2.1 Lid-driven cavity flow . . . U-19
2.1.1 Pre-processing . . . U-20
2.1.1.1 Mesh generation . . . U-21
2.1.1.2 Boundary and initial conditions . . . U-23
2.1.1.3 Physical properties . . . U-24
2.1.1.4 Control . . . U-24
2.1.1.5 Discretisation and linear-solver settings . . . U-25
2.1.1.6 Saving data to file . . . U-26
2.1.2 Viewing the mesh . . . U-26
2.1.3 Running an application . . . U-28
2.1.4 Post-processing . . . U-28
2.1.4.1 Contour plots . . . U-28
2.1.4.2 Vector plots . . . U-30
2.1.4.3 Streamline plots . . . U-30
2.1.5 Increasing the mesh resolution . . . U-33
2.1.5.1 Creating a new case using an existing case . . . U-33
2.1.5.2 Creating the finer mesh . . . U-33

2.1.5.3 Mapping the coarse mesh results onto the fine mesh . . . U-34
2.1.5.4 Control adjustments . . . U-35
2.1.5.5 Running the code as a background process . . . U-35
2.1.5.6 Vector plot with the refined mesh . . . U-35
2.1.5.7 Plotting graphs . . . U-35
2.1.6 Introducing mesh grading . . . U-37
2.1.6.1 Creating the graded mesh . . . U-37
2.1.6.2 Changing time and time step . . . U-39
2.1.6.3 Mapping fields . . . U-40
2.1.7 Increasing the Reynolds number . . . U-40
2.1.7.1 Pre-processing . . . U-41
2.1.7.2 Running the code . . . U-41
2.1.8 High Reynolds number flow . . . U-42
2.1.8.1 Pre-processing . . . U-42
2.1.8.2 Running the code . . . U-43
2.1.9 Changing the case geometry . . . U-43
2.1.10 Post-processing the modified geometry . . . U-45
2.2 Stress analysis of a plate with a hole . . . U-45
2.2.1 Mesh generation . . . U-48
2.2.1.1 Boundary and initial conditions . . . U-50
2.2.1.2 Mechanical properties . . . U-51
2.2.1.3 Thermal properties . . . U-51
2.2.1.4 Control . . . U-52
2.2.1.5 Discretisation schemes and linear-solver control . . . U-53
2.2.2 Running the code . . . U-55
2.2.3 Post-processing . . . U-55
2.2.4 Exercises . . . U-56
2.2.4.1 Increasing mesh resolution . . . U-56
2.2.4.2 Introducing mesh grading . . . U-57
2.2.4.3 Changing the plate size . . . U-57
2.3 Breaking of a dam . . . U-57
2.3.1 Mesh generation . . . U-57
2.3.2 Boundary conditions . . . U-59
2.3.3 Setting initial field . . . U-59
2.3.4 Fluid properties . . . U-60
2.3.5 Time step control . . . U-61
2.3.6 Discretisation schemes . . . U-62
2.3.7 Linear-solver control . . . U-63
2.3.8 Running the code . . . U-63
2.3.9 Post-processing . . . U-64
2.3.10 Running in parallel . . . U-64
2.3.11 Post-processing a case run in parallel . . . U-68

3 Applications and libraries **U-69**
3.1 The programming language of OpenFOAM . . . U-69
3.1.1 Language in general . . . U-69
3.1.2 Object-orientation and C++ . . . U-70
3.1.3 Equation representation . . . U-70

3.1.4 Solver codes . . . U-71
3.2 Compiling applications and libraries . . . U-71
3.2.1 Header .H files . . . U-71
3.2.2 Compiling with wmake . . . U-73
3.2.2.1 Including headers . . . U-73
3.2.2.2 Linking to libraries . . . U-74
3.2.2.3 Source files to be compiled . . . U-75
3.2.2.4 Running wmake . . . U-75
3.2.2.5 wmake environment variables . . . U-76
3.2.3 Removing dependency lists: wclean and rmdepall . . . U-76
3.2.4 Compilation example: the turbFoam application . . . U-77
3.2.5 Debug messaging and optimisation switches . . . U-80
3.2.6 Linking new user-defined libraries to existing applications . . . U-80
3.3 Running applications . . . U-81
3.4 Running applications in parallel . . . U-82
3.4.1 Decomposition of mesh and initial field data . . . U-82
3.4.2 Running a decomposed case . . . U-83
3.4.3 Distributing data across several disks . . . U-85
3.4.4 Post-processing parallel processed cases . . . U-85
3.4.4.1 Reconstructing mesh and data . . . U-85
3.4.4.2 Post-processing decomposed cases . . . U-86
3.5 Standard solvers . . . U-86
3.6 Standard utilities . . . U-88
3.7 Standard libraries . . . U-93

4 OpenFOAM cases U-99
4.1 File structure of OpenFOAM cases . . . U-99
4.2 Basic input/output file format . . . U-100
4.2.1 General syntax rules . . . U-100
4.2.2 Dictionaries . . . U-101
4.2.3 The data file header . . . U-101
4.2.4 Lists . . . U-102
4.2.5 Scalars, vectors and tensors . . . U-103
4.2.6 Dimensioned types . . . U-103
4.2.7 Fields . . . U-104
4.3 Time and data input/output control . . . U-105
4.4 Numerical schemes . . . U-107
4.4.1 Interpolation schemes . . . U-109
4.4.1.1 Schemes for strictly bounded scalar fields . . . U-110
4.4.1.2 Schemes for vector fields . . . U-110
4.4.2 Surface normal gradient schemes . . . U-111
4.4.3 Gradient schemes . . . U-112
4.4.4 Laplacian schemes . . . U-112
4.4.5 Divergence schemes . . . U-113
4.4.6 Time schemes . . . U-114
4.4.7 Flux calculation . . . U-114
4.5 Solution and algorithm control . . . U-115
4.5.1 Linear solver control . . . U-115

4.5.1.1 Solution tolerances . . . U-116
4.5.1.2 Preconditioned conjugate gradient solvers . . . U-116
4.5.1.3 Smooth solvers . . . U-117
4.5.1.4 Geometric-algebraic multi-grid solvers . . . U-117
4.5.2 Solution under-relaxation . . . U-118
4.5.3 PISO and SIMPLE algorithms . . . U-119
4.5.3.1 Pressure referencing . . . U-119
4.5.4 Other parameters . . . U-120

5 The FoamX case manager U-121
5.1 The name server and host browser . . . U-122
5.1.1 Notes for running the name server . . . U-123
5.2 The JAVA GUI . . . U-123
5.3 The case browser . . . U-125
5.3.1 Opening a root directory . . . U-127
5.3.2 Creating a new case . . . U-127
5.3.3 Opening an existing case . . . U-128
5.3.4 Deleting an existing case . . . U-128
5.3.5 Cloning an existing case . . . U-128
5.3.6 Unlocking an existing case . . . U-129
5.3.7 The process editor . . . U-129
5.3.8 Running OpenFOAM utilities . . . U-131
5.4 The case server . . . U-131
5.4.1 Importing an existing mesh . . . U-131
5.4.2 Reading a mesh . . . U-132
5.4.3 Setting boundary patches . . . U-132
5.4.4 Setting the fields . . . U-132
5.4.5 Editing the dictionaries . . . U-133
5.4.6 Saving data . . . U-134
5.4.7 Running solvers . . . U-134
5.4.8 Running utilities . . . U-135
5.4.9 Closing the case server . . . U-135
5.5 Configuration to run FoamX . . . U-136
5.5.1 JAVA . . . U-137
5.5.2 Paths to case files . . . U-137

6 Mesh generation and conversion U-139
6.1 Mesh description . . . U-139
6.1.1 Mesh specification and validity constraints . . . U-139
6.1.1.1 Points . . . U-140
6.1.1.2 Faces . . . U-140
6.1.1.3 Cells . . . U-141
6.1.1.4 Boundary . . . U-141
6.1.2 The polyMesh description . . . U-141
6.1.3 The cellShape tools . . . U-142
6.1.4 1- and 2-dimensional and axi-symmetric problems . . . U-143
6.2 Boundaries . . . U-143
6.2.1 Specification of patch types in OpenFOAM . . . U-146

6.2.2 Base types . . . U-148
6.2.3 Primitive types . . . U-149
6.2.4 Derived types . . . U-149
6.3 Mesh generation with the blockMesh utility . . . U-149
6.3.1 Writing a *blockMeshDict* file . . . U-153
6.3.1.1 The `vertices` . . . U-153
6.3.1.2 The `edges` . . . U-153
6.3.1.3 The `blocks` . . . U-154
6.3.1.4 The `patches` . . . U-155
6.3.2 Multiple blocks . . . U-156
6.3.3 Creating blocks with fewer than 8 vertices . . . U-158
6.3.4 Running blockMesh . . . U-159
6.4 Mesh conversion . . . U-159
6.4.1 fluentMeshToFoam . . . U-159
6.4.2 starToFoam . . . U-160
6.4.2.1 General advice on conversion . . . U-160
6.4.2.2 Eliminating extraneous data . . . U-161
6.4.2.3 Removing default boundary conditions . . . U-161
6.4.2.4 Renumbering the model . . . U-162
6.4.2.5 Writing out the mesh data . . . U-163
6.4.2.6 Problems with the *.vrt* file . . . U-164
6.4.2.7 Converting the mesh to OpenFOAM format . . . U-164
6.4.3 gambitToFoam . . . U-164
6.4.4 ideasToFoam . . . U-165
6.4.5 cfxToFoam . . . U-165
6.5 Mapping fields between different geometries . . . U-165
6.5.1 Mapping consistent fields . . . U-166
6.5.2 Mapping inconsistent fields . . . U-166
6.5.3 Mapping parallel cases . . . U-167

7 Post-processing **U-169**
7.1 paraFoam . . . U-169
7.1.1 Overview of paraFoam . . . U-169
7.1.2 The Parameters panel . . . U-171
7.1.3 The Display panel . . . U-172
7.1.4 Manipulating the view . . . U-172
7.1.4.1 3D view properties . . . U-172
7.1.4.2 Rotation, translation and magnification . . . U-173
7.1.5 Contour plots . . . U-173
7.1.5.1 Introducing a cutting plane . . . U-173
7.1.6 Vector plots . . . U-173
7.1.6.1 Plotting at cell centres . . . U-173
7.1.7 Streamlines . . . U-175
7.2 Post-processing with Fluent . . . U-175
7.3 Post-processing with Fieldview . . . U-177
7.4 Post-processing with EnSight . . . U-177
7.4.1 Converting data to EnSight format . . . U-177
7.4.2 The ensight74FoamExec reader module . . . U-178

7.4.2.1 Configuration of EnSight for the reader module . . . U-178
7.4.2.2 Using the reader module . . . U-179
7.5 Sampling data for plotting graphs . . . U-179
7.6 Monitoring and managing jobs . . . U-182
7.6.1 The foamJob script for running jobs . . . U-182
7.6.2 The foamLog script for monitoring jobs . . . U-183

8 Models and physical properties **U-185**
8.1 Thermophysical models . . . U-185
8.1.1 Thermophysical property data . . . U-186
8.2 Turbulence models . . . U-188

A Reference information **U-191**
A.1 Running a decomposed case in parallel using MPICH . . . U-191
A.1.1 Same executable pathname on all nodes . . . U-191
A.1.2 Different executable pathname on different nodes . . . U-192

Index **U-193**

Chapter 1

Introduction

This guide accompanies the release of version 1.4.1 of the Open Source Field Operation and Manipulation (OpenFOAM) C++ libraries. It provides a description of the basic operation of OpenFOAM, first through a set of tutorial exercises in chapter 2 and later by a more detailed description of the individual components that make up OpenFOAM.

OpenFOAM is first and foremost a *C++ library*, used primarily to create executables, known as *applications*. The applications fall into two categories: *solvers*, that are each designed to solve a specific problem in continuum mechanics; and *utilities*, that are designed to perform tasks that involve data manipulation. The OpenFOAM distribution contains numerous solvers and utilities covering a wide range of problems, as described in chapter 3.

One of the strengths of OpenFOAM is that new solvers and utilities can be created by its users with some pre-requisite knowledge of the underlying method, physics and programming techniques involved. Information relating to these subjects is placed in the Programmer's Guide.

OpenFOAM is supplied with pre- and post-processing environments. The interface to the pre- and post-processing are themselves OpenFOAM utilities, thereby ensuring consistent data handling across all environments. The overall structure of OpenFOAM is shown in Figure 1.1. The pre-processing and running of OpenFOAM cases is described in chapter 4

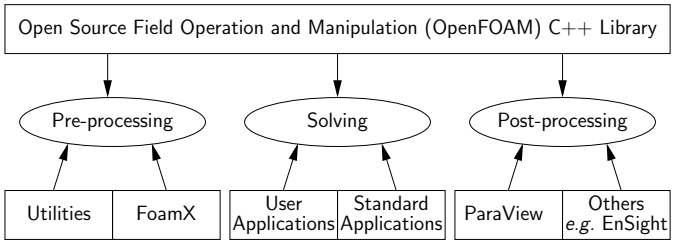


Figure 1.1: Overview of OpenFOAM structure.

and chapter 5. In chapter 6, we cover both the generation of meshes using the mesh generator supplied with OpenFOAM and conversion of mesh data generated by third-party products. Post-processing is described in chapter 7.

Chapter 2

Tutorials

In this chapter we shall describe in detail the process of setup, simulation and post-processing for some OpenFOAM test cases, with the principal aim of introducing a user to the basic procedures of running OpenFOAM. The *$FOAM_TUTORIALS* directory contains many more cases that demonstrate the use of all the solvers and many utilities supplied with OpenFOAM. Before attempting to run the tutorials, the user must first make sure that they have installed OpenFOAM correctly.

The tutorial cases describe the use of the FoamX and blockMesh pre-processing tools, running OpenFOAM solvers and post-processing using paraFoam. Those users with access to third-party post-processing tools supported in OpenFOAM have an option: either they can follow the tutorials using paraFoam; or refer to the description of the use of the third-party product in chapter 7 when post-processing is required.

Copies of all tutorials are available from the *tutorials* directory of the OpenFOAM installation. The tutorials are organised into a set of subdirectories by solver, *e.g.* all the icoFoam cases are stored within a subdirectory *icoFoam*. It is strongly recommended that the user copy the *tutorials* directory into their local *run* directory. If not, they can be easily copied by typing:

```
mkdir -p $FOAM_RUN
cp -r $FOAM_TUTORIALS $FOAM_RUN
```

FoamX locates cases by the `caseRoots` path settings in the *$WM_PROJECT_DIR/.OpenFOAM-1.4.1/controlDict* file. If the user has copied the tutorials into their account as described above, the paths to the tutorial cases will be set correctly by default; otherwise the user must make a local copy of this file in *$HOME/.OpenFOAM-1.4.1/controlDict* to edit the paths accordingly.

2.1 Lid-driven cavity flow

This tutorial will describe how to pre-process, run and post-process a case involving isothermal, incompressible flow in a two-dimensional square domain. The geometry is shown in Figure 2.1 in which all the boundaries of the square are walls. The top wall moves in the x-direction at a speed of 1 m/s while the other 3 are stationary. Initially, the flow will be assumed laminar and will be solved on a uniform mesh using the icoFoam solver for laminar, isothermal, incompressible flow. During the course of the tutorial, the effect of increased

mesh resolution and mesh grading towards the walls will be investigated. Finally, the flow Reynolds number will be increased and the turbFoam solver will be used for turbulent, isothermal, incompressible flow.

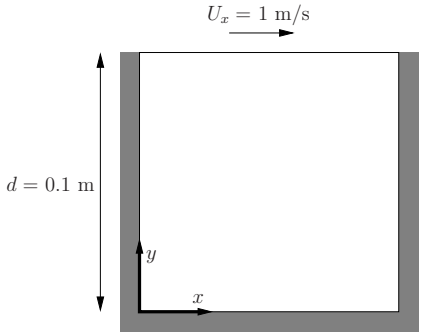


Figure 2.1: Geometry of the lid driven cavity.

2.1.1 Pre-processing

The pre-processing in OpenFOAM can be performed using **either** FoamX, a JAVA GUI tool for managing OpenFOAM cases, **or** by editing files by hand. Most OpenFOAM users choose to edit files by hand because the I/O uses a dictionary format with keywords that convey sufficient meaning to be understood by even the least experienced users. FoamX is really a layer that interprets the entries and presents them in a GUI. In these tutorials we first present the use of FoamX while aiming also to explain the structure of relevant files. The use of FoamX is described in more detail in chapter 5, but the command descriptions given in the text below should be sufficient to guide the user through the tutorial.

First the user must start up FoamX. Here, we are assuming that the FoamX host browser is being run on the local machine; otherwise, to run on a remote machine the user should consult chapter 5. Simply type

```
FoamX
```

at a command prompt to start up the script which should run the nameserver, host browser and the JAVA GUI. The FoamX JAVA GUI should now appear as in Figure 2.2. The left panel of the window contains the name of the host machine. The user should open the host by a double click on the host icon which produces a tree list of directory paths to the tutorial cases that have been copied into the user's *run* directory, as described on page U-19.

The user should now open the *$FOAM_RUN/tutorials/icoFoam* directory by a double click on the root directory icon. This opens the directory revealing all the case directories located at the *$FOAM_RUN/tutorials/icoFoam* path. The user should select the cavity case, again by a double click on the case icon.

The case opens in the same panel of the FoamX window and presents the user with a directory tree containing the description of the case: Dictionaries of input data and control parameters; a list of the Fields required in the problem *e.g.* velocity; and, the Mesh.

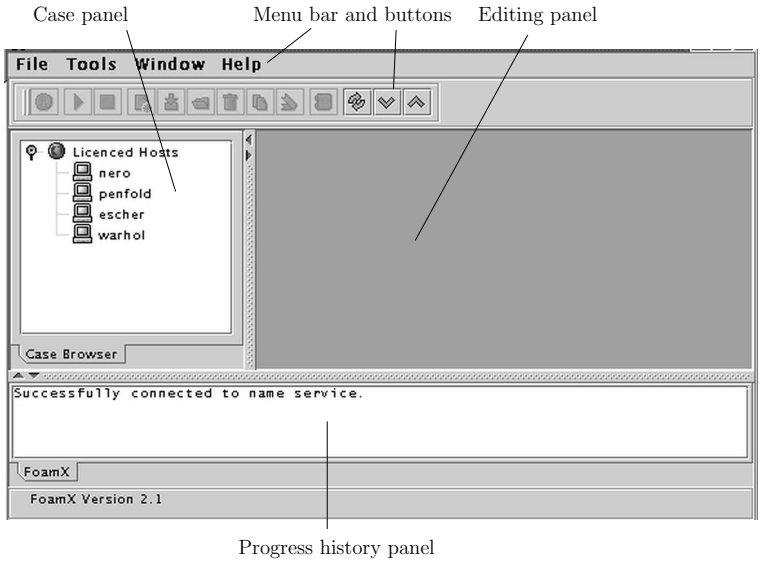


Figure 2.2: FoamX main browser window.

2.1.1.1 Mesh generation

OpenFOAM always operates in a 3 dimensional Cartesian coordinate system and all geometries are generated in 3 dimensions. OpenFOAM solves the case in 3 dimensions by default but can be instructed to solve in 2 dimensions by specifying a 'special' empty boundary condition on boundaries normal to the (3rd) dimension for which no solution is required.

The cavity domain consists of a square of side length $d = 0.1$ m in the x-y plane. A uniform mesh of 20 by 20 cells will be used initially. The block structure is shown in Figure 2.3. The mesh generator supplied with OpenFOAM, blockMesh, generates meshes from a description specified in an input dictionary, *blockMeshDict* located in the *constant/polyMesh* directory for a given case. The *blockMeshDict* entries for this case are as follows:

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25
26  convertToMeters 0.1;
27
28  vertices
29  (
30      (0 0 0)
31      (1 0 0)
32      (1 1 0)
33      (0 1 0)
34      (0 0 0.1)
35      (1 0 0.1)
36      (1 1 0.1)
37      (0 1 0.1)
38  );
39
40  blocks
41  (
42      hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
```

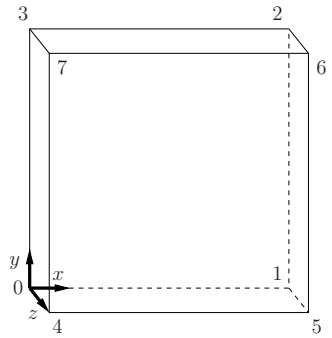



Figure 2.3: Block structure of the mesh for the cavity.

```
43  );
44
45  edges
46  (
47  );
48
49  patches
50  (
51      wall movingWall
52      (
53          (3 7 6 2)
54      )
55      wall fixedWalls
56      (
57          (0 4 7 3)
58          (2 6 5 1)
59          (1 5 4 0)
60      )
61      empty frontAndBack
62      (
63          (0 3 2 1)
64          (4 5 6 7)
65      )
66  );
67
68  mergePatchPairs
69  (
70  );
71
72
73  // ************************************************************************* //
```

The file first specifies coordinates of the block `vertices`; it then defines the `blocks` (here, only 1) from the vertex labels and the number of cells within it; and finally, it defines the boundary patches. The user is encouraged to consult section 6.3 to understand the meaning of the entries in the *blockMeshDict* file.

The mesh is generated by running blockMesh on this *blockMeshDict* file from within FoamX. This is done by clicking the right mouse button with the cursor over the case name cavity at the top of the directory tree. A menu opens to allow the user to select `blockMesh` from the `mesh -> generation` sub-menu of the `Foam Utilities` menu as shown in Figure 2.4. A blockMesh window appears. The user can view a table containing the components of the *blockMeshDict* by pressing the Edit Dictionary button. From this table the entries can be edited by clicking on cells in the right hand column. The user should make no changes to the dictionary, but simply close it and execute blockMesh by pressing

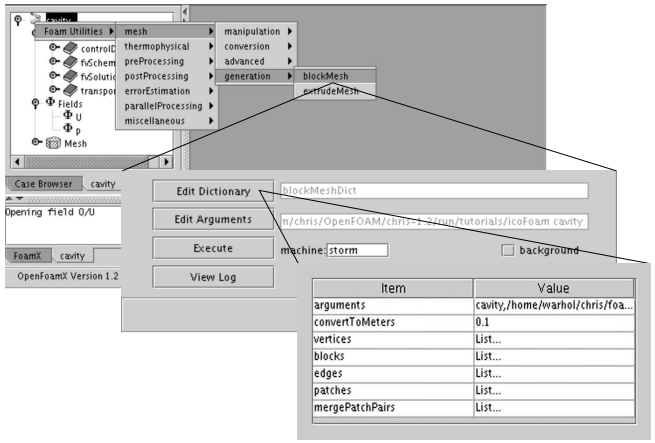


Figure 2.4: Running blockMesh.

the Execute button. The running status of blockMesh is reported in the terminal window in which FoamX was started. Any mistakes in the *blockMeshDict* file are picked up by blockMesh and the resulting error message directs the user to the line in the file where the problem occurred. There should be no error messages at this stage.

2.1.1.2 Boundary and initial conditions

Once the mesh generation is complete, click the right mouse button on Mesh in the case directory tree and select the `Read Mesh&Fields` function to load the mesh into FoamX, as shown in Figure 5.16. When the Mesh tree is opened, the names of the patches will appear in a folder named Patches. The user must click on each patch in turn to specify the physical boundary conditions as shown in Figure 5.17. Clicking on a patch brings up a window requesting the physical boundary types. For the cavity, the wall type should be selected for the `fixedWalls` and `movingWall` patches. The `frontAndBack` patch represents the front and back planes of the 2D case and therefore must be set as empty. Notice that as each boundary type is selected, the window displays the patch conditions for the primitive solution variables for each physical type, *e.g.* fixedValue for `U` and zeroGradient for `p` for the wall type.

Next select the Fields to set the internal and boundary fields, *e.g.* clicking on `U` brings up a window requesting internal field and fields on the wall boundaries as shown in Figure 2.5. Internal and boundary fields can be: `uniform`, described by a single value; or `nonuniform`, where all the values of the field must be specified. In most examples we may encounter, the initial field is set to be uniform, *e.g.* uniform internal velocity of $(0, 0, 0)$, uniform velocity of the lid (`movingWall`) boundary. If the initial field is nonuniform, it is not usually specified by entering each value by hand, but generated by a previous calculation from an application.

To enter an entry with multiple values, *e.g.* a vector, simply click on the entry and a button will appear on the right hand side of the that window; clicking on that button brings

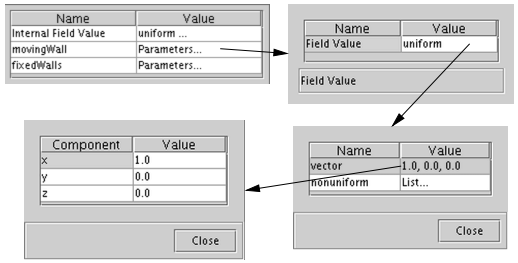


Figure 2.5: Editing the velocity field.

up a table containing the individual values of the entry that can be edited as normal as shown in Figure 2.5.

Clicking on `p` brings up a window requesting internal field which should be set to 0. Clicking on `U` brings up a window requesting internal field and values on the wall boundaries. The internal and reference fields should both be set to $(0, 0, 0)$. The velocity should be set to $(0, 0, 0)$ on the `fixedWalls` and $(1, 0, 0)$ on the `movingWall`.

2.1.1.3 Physical properties

The physical properties for the case are stored in dictionaries whose names are given the suffix ... *Properties*, located in the `Dictionaries` directory tree. For an icoFoam case, the only property that must be specified is the kinematic viscosity which is stored from the *transportProperties* dictionary. The user must ensure that the kinematic viscosity is set correctly by double clicking on the *transportProperties* dictionary to view/edit its entries. The keyword for kinematic viscosity is `nu`, the phonetic label for the Greek symbol ν by which it is represented in equations. Initially this case will be run with a Reynolds number of 10, where the Reynolds number is defined as:

$$Re = \frac{d|\mathbf{U}|}{\nu} \tag{2.1}$$

where d and $|\mathbf{U}|$ are the characteristic length and velocity respectively and ν is the kinematic viscosity. Here $d = 0.1$ m, $|\mathbf{U}| = 1\,\mathrm{m\,s^{-1}}$, so that for $Re = 10$, $\nu = 0.01\,\mathrm{m^2\,s^{-1}}$. The correct dimensions for kinematic viscosity are specified with its value, in SI units. The dimension is described in terms of powers of SI base units of measurement [kg m s K A mol cd], in this case $\mathrm{m^2\,s^{-1}}$, or

```
[0 2 -1 0 0 0 0]
```

Further information on the use of dimensional units in OpenFOAM is available in section 1.5 of the Programmer's Guide. Edit the kinematic viscosity as appropriate and close the *transportProperties* window.

2.1.1.4 Control

Input data relating to the control of time and reading and writing of the solution data are read in from the *controlDict* dictionary. Firstly, the user must set the start/stop times and

the time step for the run. OpenFOAM offers great flexibility with time control which is described in full in section 4.3. In this tutorial we wish to start the run at time $t = 0$ which means that OpenFOAM needs to read field data from a directory named *0* — see section 4.1 for more information of the case file structure. Therefore we set the `startFrom` keyword to `startTime` and then specify the `startTime` keyword to be 0.

For the end time, we wish to reach the steady state solution where the flow is circulating around the cavity. As a general rule, the fluid must pass through the domain 10 times to reach steady state in laminar flow. In this case the flow does not pass through this domain as there is no inlet or outlet, so instead the end time can be set to the time taken for the lid to travel ten times across the cavity, *i.e.* 1 s; in fact, with hindsight, we discover that 0.5 s is sufficient so we shall adopt this value. To specify this end time, we must specify the `stopAt` keyword as `endTime` and then set the `endTime` keyword to 0.5.

Now we need to set the time step, represented by the keyword `deltaT`. To achieve temporal accuracy and numerical stability when running icoFoam, a Courant number of less than 1 is required. The Courant number is defined for one cell as:

$$Co = \frac{\delta t |\mathbf{U}|}{\delta x} \tag{2.2}$$

where δt is the time step, $|\mathbf{U}|$ is the magnitude of the velocity through that cell and δx is the cell size in the direction of the velocity. The flow velocity varies across the domain and we must ensure $Co < 1$ everywhere. We therefore choose δt based on the worst case: the *maximum* Co corresponding to the combined effect of a large flow velocity and small cell size. Here, the cell size is fixed across the domain so the maximum Co will occur next to the lid where the velocity approaches 1 $\mathrm{m\,s^{-1}}$. The cell size is:

$$\delta x = \frac{d}{n} = \frac{0.1}{20} = 0.005 \text{ m} \tag{2.3}$$

Therefore to achieve a Courant number less than or equal to 1 throughout the domain the time step `deltaT` must be set to less than or equal to:

$$\delta t = \frac{Co\ \delta x}{|\mathbf{U}|} = \frac{1 \times 0.005}{1} = 0.005 \text{ s} \tag{2.4}$$

As the simulation progresses we wish to write results at certain intervals of time that we can later view with a post-processing package. The `writeControl` keyword presents several options for setting the time at which the results are written; here we select the `timeStep` option which specifies that results are written every nth time step where the value n is specified under the `writeInterval` keyword. Let us decide that we wish to write our results at times 0.1, 0.2,..., 0.5 s. With a time step of 0.005 s, we therefore need to output results at every 20th time time step and so we set `writeInterval` to 20.

OpenFOAM creates a new directory *named after the current time*, *e.g.* 0.1 s, on each occasion that it writes a set of data, as discussed in full in section 4.1. In the icoFoam solver, it writes out the results for each field, `U` and `p`, into the time directories. For this case, the remaining entries in the *controlDict* are shown in Figure 2.6.

2.1.1.5 Discretisation and linear-solver settings

The user specifies the choice of finite volume discretisation schemes in the *fvSchemes* dictionary. The specification of the linear equation solvers and tolerances and other algorithm

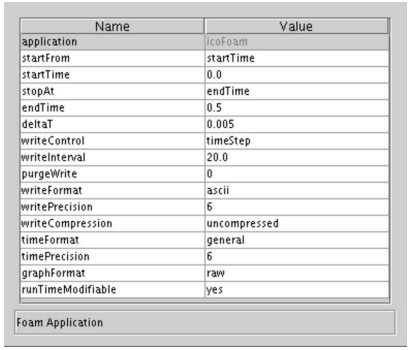


Figure 2.6: Initial *controlDict* settings for cavity.

controls is made in the *fvSolution* dictionary. The user is free to view these dictionaries but we do not need to discuss all their entries at this stage except for `pRefCell` and `pRefValue` in the *PISO* subdictionary of the *fvSolution* dictionary. In a closed incompressible system such as the cavity, pressure is relative: it is the pressure range that matters not the absolute values. In cases such as this, the solver sets a reference level by `pRefValue` in cell `pRefCell`. In this example both are set to 0. Changing either of these values will change the absolute pressure field, but not, of course, the relative pressure field or velocity field.

2.1.1.6 Saving data to file

The mesh has now been generated, the boundary conditions and fields have been initialised and the control parameters and material properties have been set. This data must be **saved** to the case files by clicking the menu button with the floppy disk icon at the top of the FoamX window.

For the remainder of the manual:

> There are menu buttons at the top of the FoamX window. To check which function a button performs, hold the cursor over the button for 1 s and a text description will appear.

2.1.2 Viewing the mesh

Before the case is run it is a good idea to view the mesh to check for any errors. The mesh is viewed in paraFoam, the post-processing tool supplied with OpenFOAM. The paraFoam post-processing is started by typing at a command prompt

```
paraFoam <root> <case>
```

i.e. making the appropriate substitutions for `<root>` path and `<case>`, paraFoam is executed on cavity by typing

```
paraFoam $FOAM_RUN/tutorials/icoFoam cavity
```

This launches the ParaView window as shown in Figure 7.1. In the Selection Window, the user can see that ParaView has opened `cavity.foam`, the module for the cavity case. The user should immediately click the Accept button which will bring up an image of the case geometry in the image display window. The user should then open the Display panel that controls the visual representation of the selected module. Within the Display panel the user should select, as shown in Figure 2.7: `Color by Property`; a suitable `Actor Color`, *e.g.* white; `Wireframe of Surface` representation.

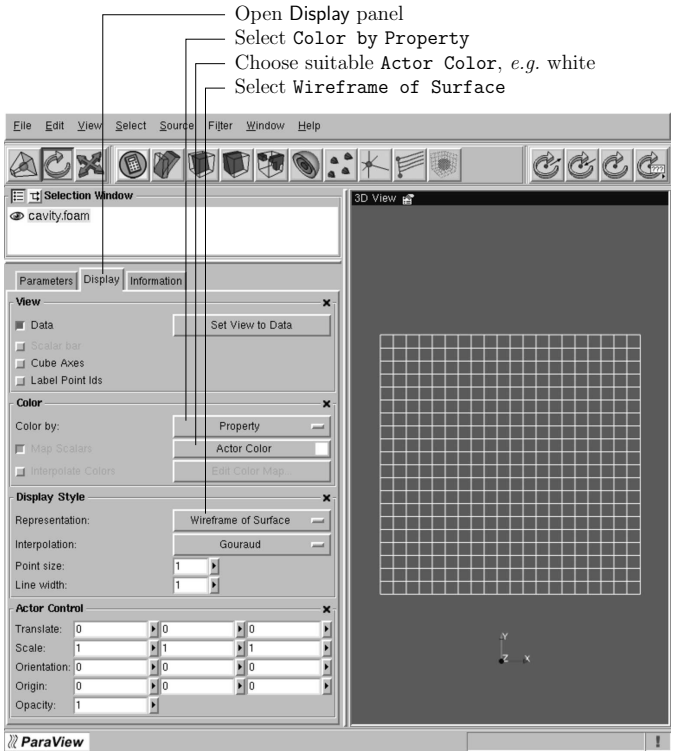


Figure 2.7: Viewing the mesh in paraFoam.

The user can try manipulating the view as described in section 7.1.4. In particular, since this is a 2D case, it is recommended that Use parallel projection is selected in the `3D view Properties` window, described in section 7.1.4.1. The Orientation Axes can be toggled on and off in the Annotate window or moved by drag and drop with the mouse.

2.1.3 Running an application

Like any UNIX/Linux executable, OpenFOAM applications can be run in two ways: as a foreground process, *i.e.* one in which the shell waits until the command has finished before giving a command prompt; as a background process, one which does not have to be completed before the shell accepts additional commands.

On this occasion, we will run icoFoam in the foreground. There are two ways that this can be done: either by clicking the Start Calculation Now button ( ) in FoamX; or by typing at a command prompt:

```
icoFoam $FOAM_RUN/tutorials/icoFoam cavity
```

The progress of the job is written to the terminal window. It tells the user the current time, maximum Courant number, initial and final residuals for all fields. For more detail about running OpenFOAM solvers, see section 5.4.7.

2.1.4 Post-processing

As soon as results are written to time directories, they can be viewed using paraFoam. Return to the paraFoam window and select the Parameters panel for the `cavity.foam` case module. If the correct window panels for the case module do not seem to be present at any time, please ensure that: `cavity.foam` is highlighted in yellow; eye button alongside it is switched on to show the graphics are enabled; `Source` is selected in the from the `View` menu.

To prepare paraFoam to display the data of interest, we must first load the data at the required run time of 0.5 s. To do so, the user must click the 0.5 button in the Time window of the Parameters panel, and then Accept to confirm. All the geometric and field data are loaded since, by default, all items are selected in the Region and Fields, respectively.

2.1.4.1 Contour plots

To view pressure, the user should return to the Display panel since it that controls the visual representation of the selected module. To make a simple plot of pressure, the user should select as shown in Figure 2.8: `Color by volPointInterpolate(p)`; `Edit Color Map` and **select** Reset Range; `Surface` representation.

The pressure field solution at $t = 0.5$ s has, as expected, a region of low pressure at the top left of the cavity and one of high pressure at the top right of the cavity as shown in Figure 2.9.

The pressure field is interpolated across each cell to give a continuous appearance. Instead if the user selects `Color by cell(p)` in the Display panel, a single value for pressure will be attributed to each cell so that each cell will be denoted by a single colour with no grading.

A colour bar can be included by clicking the visibility button in the Scalar bar window of the `Edit Color Map` menu. The user can set a range of attributes of the colour bar, such as text size, font selection and numbering format for the scale. The colour bar can be located in the image window by drag and drop with the mouse.

If the user rotates the image, they can see that we have now coloured the complete geometry surface by the pressure in this case. In order to produce a genuine contour plot

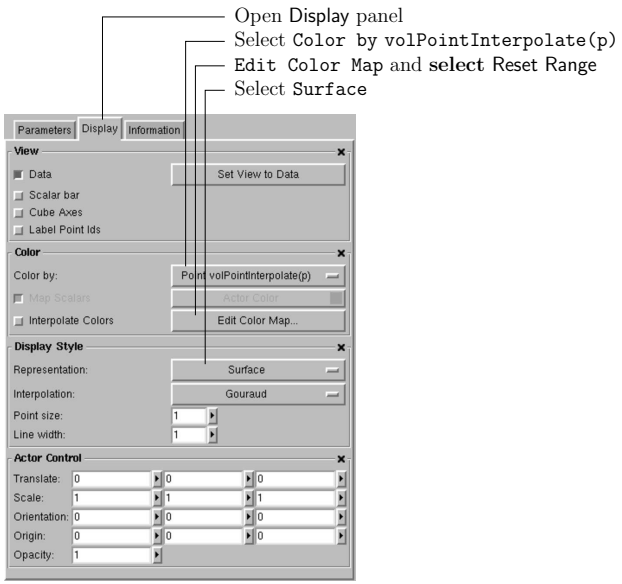


Figure 2.8: Displaying pressure contours for the cavity case.

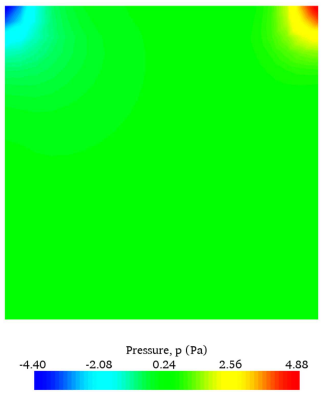


Figure 2.9: Pressures in the cavity case.

the user should first create a cutting plane through the geometry using the Cut filter as described in section 7.1.5.1. The cutting plane should be centred at $(0.05, 0.05, 0.005)$ and its normal should be set to $(0, 0, 1)$. Having generated the cutting plane, the contours can be created using by the Contour filter described in section 7.1.5.

2.1.4.2 Vector plots

Before we start to plot the vectors of the flow velocity, it may be useful to remove other modules that have been created, *e.g.* using the Cut and Contour filters described above. These can: either be deleted entirely, by highlighting the relevant module in the Selection Window and clicking Delete in their respective Parameters panel; or, be disabled by toggling the eye button for the relevant module in the Selection Window.

We now wish to generate a vector glyph for velocity at the centre of each cell. We first need to filter the data to cell centres as described in section 7.1.6.1. With the cavity.foam module highlighted in the Selection Window, the user should select Cell Centers from the Filter menu and then click Accept.

With these Centers highlighted in the Selection Window, the user should then select Glyph from the Filter menu. The Parameters window panel should appear as shown in Figure 2.10. In the resulting Parameters panel, the velocity field, U, is automatically selected in the vectors menu, since it is the only vector field present. By default the Scale Mode for the glyphs will be Vector Magnitude of velocity but, since the we may wish to view the velocities throughout the domain, the user should instead select Data Scaling Off and use a Scale Factor of 0.005. On clicking Accept, the glyphs appear but, probably as a single colour, *e.g.* white. The user should colour the glyphs by velocity magnitude which, as usual, is controlled by setting Color by Point U(3) (or Point Glyph Vector(3)) in the Display panel. The user should also create a Scalar Bar in Edit Color Map. The output is shown in Figure 2.11, in which uppercase Times Roman fonts are selected for the Scalar Bar headings and the labels are specified to 2 fixed significant figures by entering %-#6.2f in the Labels text box. The background colour is set to white in the General panel of 3D view Properties as described in section 7.1.4.1.

2.1.4.3 Streamline plots

Again, before the user continues to post-process in ParaView, they should disable modules such as those for the vector plot described above. We now wish to plot a streamlines of velocity as described in section 7.1.7. With the cavity.foam module highlighted in the Selection Window, the user **must first extract the internal mesh** since data probing in ParaView does not work on surface geometry.

Therefore the user should select Extract Parts from the Filter menu, select **only** Internal Mesh and then click Accept. With the ExtractParts module highlighted in the Selection Window, the user should then select Stream Tracer from the Filter menu and then click Accept. The Parameters window panel should appear as shown in Figure 2.12. The Seed points should be specified along a Line running vertically through the centre of the geometry, *i.e.* from $(0.05, 0, 0.005)$ to $(0.05, 0.1, 0.005)$. For the image in this guide we used: a point resolution of 21; Max Propagation by Length 0.5; Init. Step Len. by Cell Length 0.01; and, Integration Direction BOTH. The Runge-Kutta 2 IntegratorType was used with default parameters.

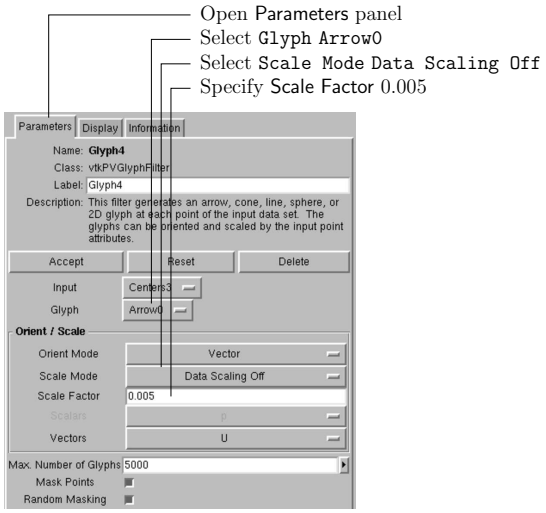


Figure 2.10: Parameters panel for the `Glyph` filter.

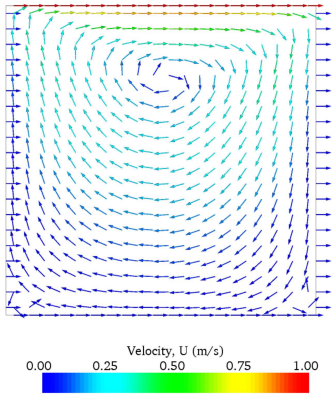


Figure 2.11: Velocities in the cavity case.

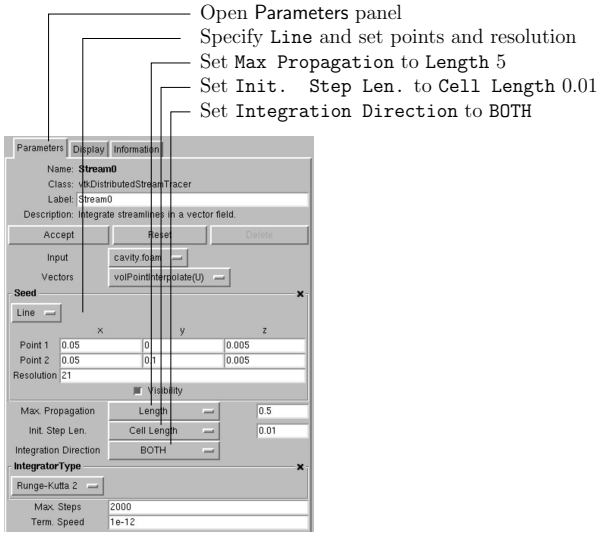


Figure 2.12: Parameters panel for the `Stream Tracer` filter.

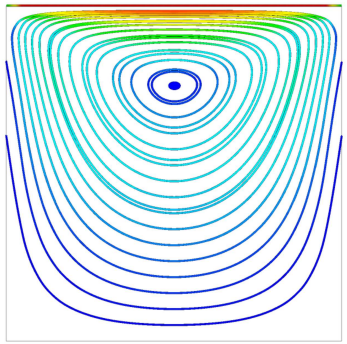


Figure 2.13: Streamlines in the cavity case.

On clicking Accept the tracer is generated. The user should then select Tube from the Filter menu to produce high quality streamline images. For the image in this report, we used: Num. sides 20; Radius 0.0003; and, Radius factor 10. On clicking Accept the image in Figure 2.13 should be produced.

2.1.5 Increasing the mesh resolution

The mesh resolution will now be increased by a factor of two in each direction. The results from the coarser mesh will be mapped onto the finer mesh to use as initial conditions for the problem. The solution from the finer mesh will then be compared with those from the coarser mesh.

2.1.5.1 Creating a new case using an existing case

Close the cavity case in FoamX by clicking the Close Case button ( ). We now wish to create a new case named cavityFine that is created from cavity. The user should therefore clone the cavity case and edit the necessary files. In the FoamX case browser window, simply place the cursor over the cavity case and click the right mouse button to bring up a menu from which Clone Case can be selected. The user should enter the root path, the name of the new case, cavityFine, and the icoFoam application class. The times option specifies which time directories are copied into the cloned case. In this example, we require no time directory from cavity, otherwise after refining the mesh, the size of the copied fields will be inconsistent with the size of the mesh. Therefore we select noTime. By clicking Close and Yes, the new case is created and presented in the case browser window. The user should open the cavityFine case.

2.1.5.2 Creating the finer mesh

We now wish to increase the number of cells in the mesh by using blockMesh. As before, select blockMesh from the mesh -> generation sub-menu of the Foam Utilities menu by clicking the right mouse button with the cursor over the case name at the top of the directory tree. The blockMesh window appears in which the user should press the Edit Dictionary button. The user should then select the value cell of the the blocks item and in the blocks window, select (the only) block 0. This produces a new window with a table of entries for the block. The first entry, hex, describes the type of block, a hexahedron — in fact the only option in blockMesh— which is specified by an ordered list of vertex labels. The second entry, cellDensity, is the mesh density: 20 cells in the x-direction; 20 in the y-direction; and 1 in the z-direction, since it is a 2 dimensional problem. The third entry, expansionRatio specifies the mesh grading which is discussed in section 2.1.6.

In this case we wish to increase the mesh density to 41 cells in the x and y directions. Select the cellDensity entry and edit the elements, replacing 20 20 1 by 41 41 1. Then close the dictionary editing window and the dictionary will save automatically. Press the Execute button and the mesh is generated with the running status of blockMesh reported in the terminal window as before. To view the mesh, use paraFoam as described in section 2.1.2. The user may be prompted to read the new mesh into FoamX. The user **must** do so if he/she wishes, because the number of faces in the the boundary patch definitions have changed.

2.1.5.3 Mapping the coarse mesh results onto the fine mesh

The mapFields utility maps one or more fields relating to a given geometry onto the corresponding fields for another geometry. In our example, the fields are deemed 'consistent' because the geometry and the boundary types, or conditions, of both source and target fields are identical. We use the -consistent command line option when executing mapFields in this example.

The field data that mapFields maps is read from the time directory specified by startFrom/startTime in the *controlDict* of the target case, *i.e.* those **into which** the results are being mapped. In this example, we wish to map the final results of the coarser mesh from case cavity onto the finer mesh of case cavityFine. Therefore, since these results are stored in the *0.5* directory of cavity, the startTime should be set to 0.5 s in the *controlDict* dictionary and startFrom should be set to startTime. The user should **save** these changes.

The case is ready to run mapFields. Click the right mouse button with the cursor over the case name cavityFine at the top of the directory tree. A menu opens to allow the user to select mapFields from the preProcessing sub-menu of the Foam Utilities menu. A window containing arguments to the mapFields utility opens as shown in Figure 2.14. The name of the target case, with full root path, into which the results are being mapped are set correctly by default in <rootAndCase>. The user must enter the root path and name of the source case in <sourceRootAndCase>, *e.g.* as the example shows in Figure 2.14.

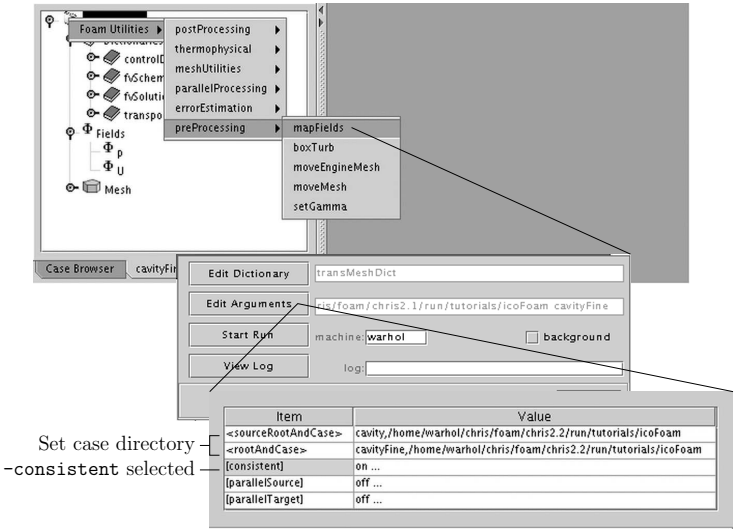


Figure 2.14: mapFields arguments.

The user must select the -consistent option from the arguments by selecting on in the [consistent] value box. The mapFields utility will run when the user clicks Execute. The progress messages in the terminal window should tell the user that the data has been interpolated from the *cavity* case to the *cavityFine* case.

2.1.5.4 Control adjustments

To maintain a Courant number of less that 1, as discussed in section 2.1.1.4, the time step must now be halved since the size of all cells has halved. Therefore `deltaT` should be set to to 0.0025 s in the *controlDict* dictionary. Field data is currently written out at an interval of a fixed number of time steps. Here we demonstrate how to specify data output at fixed intervals of time. Under the `writeControl` keyword in *controlDict*, instead of requesting output by a fixed number of time steps with the `timeStep` entry, a fixed amount of run time can be specified between the writing of results using the `runTime` entry. In this case the user should specify output every 0.1 s and therefore should set `writeInterval` to `0.1` and `writeControl` to `runTime`. Finally, since the case is starting with a the solution obtained on the coarse mesh we only need to run it for a short period to achieve reasonable convergence to steady-state. Therefore the `endTime` should be set to 0.7 s. Make sure these settings are correct and then **save** the case.

2.1.5.5 Running the code as a background process

The user should experience running icoFoam as a background process. Press the Start Calculation button ( ) and a window appears. With the background button checked, click Execute. The case runs in the background and is complete when a line of text appears in the terminal window beginning with `Finished doing...`. The user can then view the log file by clicking View Log.

2.1.5.6 Vector plot with the refined mesh

The user can open multiple cases simulateously in ParaView; essentially because each new case is simply another module that appears in the Selection Window. There is one minor inconvenience when opening a new case in ParaView because there is a prerequisite that the selected data is a file with a name that has an extension. However, in OpenFOAM, each case is stored in a multitude of files with no extensions within a specific directory structure. The solution, that the paraFoam script performs automatically, is to create a dummy file with the extension *.foam* — hence, the cavity case module is called `cavity.foam`.

However, if the user wishes to open another case directly from within ParaView, they need to create such a dummy file. For example, to load the cavityFine case the file would be created by typing at the command prompt:

```
cd $FOAM_RUN/tutorials/icoFoam
touch cavityFine/cavityFine.foam
```

Now the cavityFine case can be loaded into ParaView by selecting `Open Data` from the `File` menu, and having navigated the directory tree, selecting `cavityFine.foam`. The user can now make a vector plot of the results from the refined mesh in ParaView. The plot can be compared with the cavity case by enabling glyph images for both case simultaneously.

2.1.5.7 Plotting graphs

The user may wish to visualise the results by extracting some scalar measure of velocity and plotting 2-dimensional graphs along lines through the domain. OpenFOAM is well equipped for this kind of data manipulation and is released with some standard utilities

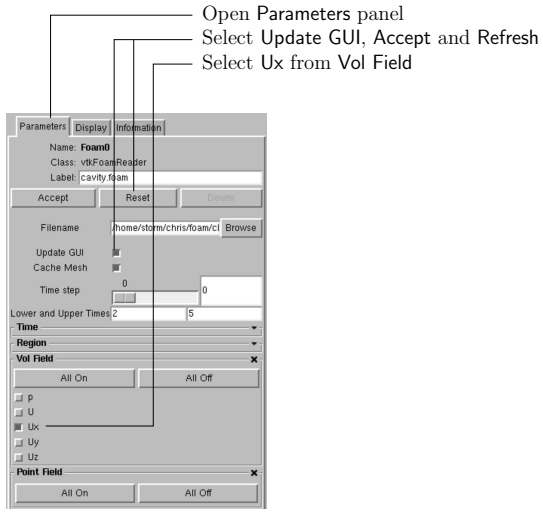


Figure 2.15: Selecting fields for graph plotting.

for this purpose, namely Ucomponents and magU. When Ucomponents is run on a case, say *cavity*, it reads in the velocity vector field from each time directory and, in the corresponding time directories, writes scalar fields `Ux`, `Uy` and `Uz` representing the x, y and z components of velocity.

The user can run the Ucomponents utility on both cavity and cavityFine cases. Like all utilities, Ucomponents can be executed from the menus opened by clicking the right mouse button with the cursor over the case name in FoamX. Ucomponents is located in the `postProcessing -> velocityField` sub-menu; execute it on cavityFine. For cavity the user may wish to execute Ucomponents from the command line by the following command:

```
Ucomponents $FOAM_RUN/tutorials/icoFoam cavityFine
```

The individual components can be plotted as a graph in ParaView, although we would recommend using the sample utility, described in section 7.5 and section 2.2.3 if users wish to produce graphs for publication.

To plot a graph in ParaView, the users **must** first extract the internal mesh using the `Extract Parts` filter as described for streamlines in section 2.1.4.3.

The user can then plot a graph by selecting `Probe` from the `Filter` menu. In the Probe Object window, the user should select `Line` and position the line vertically up the centre of the domain, *i.e.* from $(0.05, 0, 0.005)$ to $(0.05, 0.1, 0.005)$, with a resolution of, say, 50. On clicking Accept, a graph is generated as shown in Figure 2.16. The fields that are plotted are selected in the Point scalars window of the Probe panel.

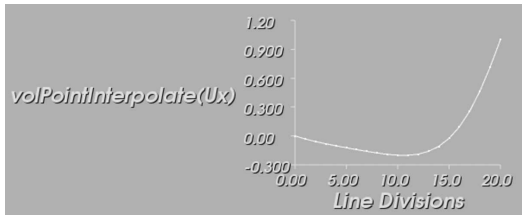


Figure 2.16: Plotting graphs in paraFoam.

There is no control over the graph image display, other than general positioning, so while it may be useful as a general viewing tool, we recommend the sample utility is used to produce graphs for publication.

2.1.6 Introducing mesh grading

The error in any solution will be more pronounced in regions where the form of the true solution differ widely from the form assumed in the chosen numerical schemes. For example a numerical scheme based on linear variations of variables over cells can only generate an exact solution if the true solution is itself linear in form. The error is largest in regions where the true solution deviates greatest from linear form, *i.e.* where the change in gradient is largest. Error decreases with cell size.

It is useful to have an intuitive appreciation of the form of the solution before setting up any problem. It is then possible to anticipate where the errors will be largest and to grade the mesh so that the smallest cells are in these regions. In the cavity case the large variations in velocity can be expected near a wall and so in this part of the tutorial the mesh will be graded to be smaller in this region. By using the same number of cells, greater accuracy can be achieved without a significant increase in computational cost.

A mesh of 20×20 cells with grading towards the walls will be created for the lid-driven cavity problem and the results from the finer mesh of section 2.1.5.2 will then be mapped onto the graded mesh to use as an initial condition. The results from the graded mesh will be compared with those from the previous meshes. Since the changes to the *blockMeshDict* dictionary are fairly substantial, the case used for this part of the tutorial, cavityGrade, is supplied in the *$FOAM_RUN/tutorials/icoFoam* directory.

2.1.6.1 Creating the graded mesh

The mesh now needs 4 blocks as different mesh grading is needed on the left and right and top and bottom of the domain. The block structure for this mesh is shown in Figure 2.17. Rather than reiterating how to pre-process and run this case in FoamX, the following steps describe the alternative method of executing applications from the terminal command line. First of all, the user can view the *blockMeshDict* file in the *constant/polyMesh* subdirectory of cavityGrade using a text editor of their choice; for completeness the key elements of the *blockMeshDict* file are also reproduced below. Each block now has 10 cells in the x and y directions and the ratio between largest and smallest cells is 2.

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
```

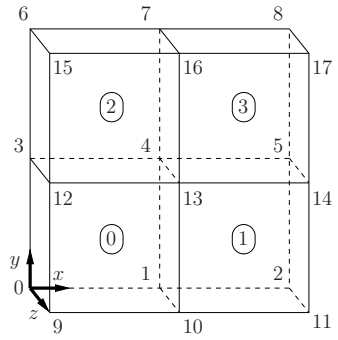



Figure 2.17: Block structure of the graded mesh for the cavity (block numbers encircled).

```
25  convertToMeters 0.1;
26
27  vertices
28  (
29      (0 0 0)
30      (0.5 0 0)
31      (1 0 0)
32      (0 0.5 0)
33      (0.5 0.5 0)
34      (1 0.5 0)
35      (0 1 0)
36      (0.5 1 0)
37      (1 1 0)
38      (0 0 0.1)
39      (0.5 0 0.1)
40      (1 0 0.1)
41      (0 0.5 0.1)
42      (0.5 0.5 0.1)
43      (1 0.5 0.1)
44      (0 1 0.1)
45      (0.5 1 0.1)
46      (1 1 0.1)
47  );
48
49  blocks
50  (
51      hex (0 1 4 3 9 10 13 12) (10 10 1) simpleGrading (2 2 1)
52      hex (1 2 5 4 10 11 14 13) (10 10 1) simpleGrading (0.5 2 1)
53      hex (3 4 7 6 12 13 16 15) (10 10 1) simpleGrading (2 0.5 1)
54      hex (4 5 8 7 13 14 17 16) (10 10 1) simpleGrading (0.5 0.5 1)
55  );
56
57  edges
58  (
59  );
60
61  patches
62  (
63      wall movingWall
64      (
65          (6 15 16 7)
66          (7 16 17 8)
67      )
68      wall fixedWalls
69      (
70          (3 12 15 6)
71          (0 9 12 3)
72          (0 1 10 9)
73          (1 2 11 10)
74          (2 5 14 11)
75          (5 8 17 14)
76      )
```

```
77      empty frontAndBack
78      (
79          (0 3 4 1)
80          (1 4 5 2)
81          (3 6 7 4)
82          (4 7 8 5)
83          (9 10 13 12)
84          (10 11 14 13)
85          (12 13 16 15)
86          (13 14 17 16)
87      )
88  );
89
90  mergePatchPairs
91  (
92  );
93
94  // ************************************************************************* //
```

Once familiar with the *blockMeshDict* file for this case, the user can execute blockMesh from the command line using a command of the form:

```
blockMesh <root> <case>
```

i.e. making the appropriate substitutions for <root> path and <case>, blockMesh is executed on cavityGrade by typing

```
blockMesh $FOAM_RUN/tutorials/icoFoam cavityGrade
```

For the remainder of the manual:
The form of the command line entry for any application can be found by simply entering the application name at the command line, *e.g.* typing `blockMesh` returns information including

```
Usage: blockMesh <root> <case> [-blockTopology]
```

the parameters in square brackets being optional flags.

The graded mesh can be viewed as before using paraFoam as described in section 2.1.2.

2.1.6.2 Changing time and time step

The highest velocities and smallest cells are next to the lid, therefore the highest Courant number will be generated next to the lid, for reasons given in section 2.1.1.4. It is therefore useful to estimate the size of the cells next to the lid to calculate an appropriate time step for this case.

When a nonuniform mesh grading is used, blockMesh calculates the cell sizes using a geometric progression. Along a length l, if n cells are requested with a ratio of R between the first and last cells, the size of the smallest cell, δx_s, is given by:

$$\delta x_s = l\frac{r-1}{\alpha r - 1} \tag{2.5}$$

where r is the ratio between one cell size and the next which is given by:

$$r = R^{\frac{1}{n-1}} \tag{2.6}$$

and

$$\alpha = \begin{cases} R^n & \text{for } R > 1, \\ R^{1/n} & \text{for } R < 1. \end{cases} \tag{2.7}$$

For the cavityGrade case the number of cells in each direction in a block is 10, the ratio between largest and smallest cells is 2 and the block height and width is 0.05 m. Therefore the smallest cell length is 3.45 mm. From Equation 2.2, the time step should be less than 3.45 ms to maintain a Courant of less than 1. To ensure that results are written out at convenient time intervals, the time step `deltaT` should be reduced to 2.5 ms and the `writeInterval` set to 40 so that results are written out every 0.1 s.

On this occasion we shall demonstrate the fact that any changes can be made to the case dictionaries by simply editing the relevant file. Here we wish to edit the time and control information which is stored in the *cavityGrade/system/controlDict* file. Open this file in an editor of your choice. As stipulated in the previous paragraph ensure that the time step `deltaT` is set to `2.5e-3` and the `writeInterval` is set to `40`.

The `startTime` needs to be set to that of the final conditions of the case cavityFine, *i.e.* `0.7`. Since cavity and cavityFine converged well within the prescribed run time, we can set the run time for case cavityGrade to 0.1 s, *i.e.* the `endTime` should be `0.8`.

2.1.6.3 Mapping fields

As in section 2.1.5.3, use mapFields to map the final results from case cavityFine onto the mesh for case cavityGrade. The mapFields utility is executed by a line of the form

```
mapFields <sourceRoot> <sourceCase> <root> <case> -consistent
```

so that we can execute it for our case by typing the following in a terminal window:

```
cd $FOAM_RUN/tutorials/icoFoam
mapFields . cavityFine . cavityGrade -consistent
```

Now run icoFoam from the case directory and monitor the run time information:

```
icoFoam . cavityGrade
```

View the converged results for this case and compare with other results using post-processing tools described previously in section 2.1.5.6 and section 2.1.5.7.

2.1.7 Increasing the Reynolds number

The cases solved so far have had a Reynolds number of 10. This is very low and leads to a stable solution quickly with only small secondary vortices at the bottom corners of the cavity. We will now increase the Reynolds number to 50, at which point the solution takes a noticably longer time to converge. The coarsest mesh in case cavity will be used initially. The user should make a copy of the cavity case and name it cavityHighRe. The user can use the `Clone Case` function in FoamX as described in section 2.1.5.1 or simply copy the cavity case directory by typing:

```
cd $FOAM_RUN/tutorials/icoFoam
cp -r cavity cavityHighRe
```

2.1.7.1 Pre-processing

Let us return to use FoamX for managing the cavityHighRe case. If the user has created the cavityHighRe case by making a copy of cavity as described above, the case will probably not appear in the case directory tree panel of the case browser window. To make it appear, the user must select `Refresh Case Browser` by either: pressing the right mouse button with the cursor over the `Licenced Hosts` icon at the top of the case directory tree; or, by selecting the Refresh Case Browser button from the menu buttons.

From the case directory tree, open the cavityHighRe case and edit the *transportProperties* dictionary. Since the Reynolds number is required to be increased by a factor of 10, decrease the kinematic viscosity by a factor of 10, *i.e.* to 1×10^{-3} m^2 s^{-1}. We can now run this case by restarting from the solution at the end of the cavity case run. To do this we can use the option of setting the `startFrom` keyword to `latestTime` so that icoFoam takes as its initial data the values stored in the directory corresponding to the most recent time, *i.e. 0.5*. The `endTime` should be set to 2 s. **Save** the case.

2.1.7.2 Running the code

Run icoFoam for this case from the case directory and view the run time information.

```
cd $FOAM_RUN/tutorials/icoFoam
nohup nice -n 19 icoFoam . cavityHighRe  > log &
cat log
```

In previous runs you may have noticed that icoFoam stops solving for velocity `U` quite quickly but continues solving for pressure `p` for a lot longer or until the end of the run. In practice, once icoFoam stops solving for `U` and the initial residual of `p` is less than the tolerance set in the *fvSolution* dictionary (typically 10^{-6}), the run has effectively converged and can be stopped once the field data has been written out to a time directory. For example, at convergence a sample of the *log* file from the run on the cavityHighRe case appears as follows in which the velocity has already converged after 1.62 s and initial pressure residuals are small; `No Iterations 0` indicates that the solution of `U` has stopped:

```
1
2   Time = 1.63
3
4   Courant Number mean: 0.108642 max: 0.818175
5   DILUPBiCG:  Solving for Ux, Initial residual = 7.86044e-06, Final residual = 7.86044e-06,
6   No Iterations 0
7   DILUPBiCG:  Solving for Uy, Initial residual = 9.4171e-06, Final residual = 9.4171e-06,
8   No Iterations 0
9   DICPCG:  Solving for p, Initial residual = 3.54721e-06, Final residual = 7.13506e-07,
10  No Iterations 4
11  time step continuity errors : sum local = 6.46788e-09, global = -9.44516e-19,
12  cumulative = 1.04595e-17
13  DICPCG:  Solving for p, Initial residual = 2.15824e-06, Final residual = 9.95068e-07,
14  No Iterations 3
15  time step continuity errors : sum local = 8.67501e-09, global = 7.54182e-19,
16  cumulative = 1.12136e-17
17  ExecutionTime = 1.02 s  ClockTime = 1 s
18
19  Time = 1.635
20
21  Courant Number mean: 0.108643 max: 0.818176
22  DILUPBiCG:  Solving for Ux, Initial residual = 7.6728e-06, Final residual = 7.6728e-06,
23  No Iterations 0
24  DILUPBiCG:  Solving for Uy, Initial residual = 9.19442e-06, Final residual = 9.19442e-06,
25  No Iterations 0
26  DICPCG:  Solving for p, Initial residual = 3.13107e-06, Final residual = 8.60504e-07,
27  No Iterations 4
28  time step continuity errors : sum local = 8.15435e-09, global = -5.84817e-20,
29  cumulative = 1.11552e-17
30  DICPCG:  Solving for p, Initial residual = 2.16689e-06, Final residual = 5.27197e-07,
31  No Iterations 14
32  time step continuity errors : sum local = 3.45666e-09, global = -5.62297e-19,
33  cumulative = 1.05929e-17
34  ExecutionTime = 1.02 s  ClockTime = 1 s
```


2.1.8 High Reynolds number flow

View the results in paraFoam and display the velocity vectors. The secondary vortices in the corners have increased in size somewhat. The user can then increase the Reynolds number further by decreasing the viscosity and then rerun the case. The number of vortices increases so the mesh resolution around them will need to increase in order to resolve the more complicated flow patterns. In addition, as the Reynolds number increases the time to convergence increases. The user should monitor residuals and extend the `endTime` accordingly to ensure convergence.

The need to increase spatial and temporal resolution then becomes impractical as the flow moves into the turbulent regime, where problems of solution stability may also occur. Of course, many engineering problems have very high Reynolds numbers and it is infeasible to bear the huge cost of solving the turbulent behaviour directly. Instead turbulence models are used to solve for the mean flow behaviour and calculate the statistics of the fluctuations. The standard $k-\varepsilon$ model with wall functions will be used in this tutorial to solve the lid-driven cavity case with a Reynolds number of 10^4. Two extra variables are solved for: k, the turbulent kinetic energy; and, ε, the turbulent dissipation rate. The additional equations and models for turbulent flow are implemented into a OpenFOAM solver called turbFoam.

2.1.8.1 Pre-processing

In FoamX, open the cavity case in the *$HOME_RUN/tutorials/turbFoam* directory (N.B: the *$FOAM_RUN/tutorials/***turbFoam** directory). Generate the mesh by running blockMesh from within FoamX as before. Mesh grading towards the wall is not necessary when using the standard $k-\varepsilon$ model with wall functions since the flow in the near wall cell is modelled, rather than having to be resolved.

Set the boundary conditions using FoamX as described in section 2.1.1.2. The selection of the wall type boundary condition assigns a zeroGradient boundary condition to ε and a fixedValue 0 boundary condition to k. To set the initial conditions, select the fields as described previously. The initial conditions for $\mathbf{U}$ and p are $(0,0,0)$ and 0 respectively as before. However, positive values for k and ε must be given to avoid division by 0 in the solution algorithm. We can specify reasonable initial conditions for k and ε in terms of an estimated fluctuating component of velocity $\mathbf{U}'$ and a turbulent length scale, l. k and ε are defined in terms of these parameters as follows:

$$k = \frac{1}{2}\overline{\mathbf{U}' \bullet \mathbf{U}'} \tag{2.8}$$

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} \tag{2.9}$$

where C_μ is a constant of the $k-\varepsilon$ model equal to 0.09. For a Cartesian coordinate system, k is given by:

$$k = \frac{1}{2}({U'_x}^2 + {U'_y}^2 + {U'_z}^2) \tag{2.10}$$

where ${U'_x}^2$, ${U'_y}^2$ and ${U'_z}^2$ are the fluctuating components of velocity in the x, y and z directions respectively. Let us assume the initial turbulence is isotropic, *i.e.* ${U'_x}^2 = {U'_y}^2 = {U'_z}^2$, and equal to 5% of the lid velocity and that l, is equal to 20% of the box width, 0.1

m, then k and ε are given by:

$$U'_x = U'_y = U'_z = \frac{5}{100} 1 \text{ m s}^{-1} \tag{2.11}$$

$$\Rightarrow k = \frac{3}{2}\left(\frac{5}{100}\right)^2 \text{ m}^2\text{ s}^{-2} = 3.75 \times 10^{-3} \text{ m}^2\text{ s}^{-2} \tag{2.12}$$

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} \approx 7.65 \times 10^{-4} \text{ m}^2\text{s}^{-3} \tag{2.13}$$

Set these initial conditions for k and ε.

Next set the laminar kinematic viscosity in the *transportProperties* dictionary. To achieve a Reynolds number of 10^4, a kinematic viscosity of 10^{-5} m is required based on the Reynolds number definition given in Equation 2.1.

To select the turbulence model open the *turbulenceProperties* dictionary. The turbulence model is selected by the `turbulenceModel` entry. It gives a long list of available models that are listed in Table 3.9. The user should select the `kEpsilon` which is is the standard $k-\varepsilon$ model; the user should also ensure that `turbulence` calculation is switched `on`. The coefficients relating to the model are stored in a standard dictionary under `kEpsilonCoeffs`; the model also uses the `wallFunctionCoeffs`.

Next set the `startTime`, `stopTime`, `deltaT` and the `writeInterval` in the *controlDict*. Set `deltaT` to 0.005 s to satisfy the Courant number restriction and the `endTime` to 10 s.

2.1.8.2 Running the code

Execute turbFoam using any of the methods described earlier in this tutorial. In this case, where the viscosity is low, the boundary layer next to the moving lid is very thin and the cells next to the lid are comparatively large so the velocity at their centres are much less than the lid velocity. In fact, after $\approx$ 100 time steps it becomes apparent that the velocity in the cells adjacent to the lid reaches an upper limit of around 0.2 m s^{-1} hence the maximum Courant number does not rise much above 0.2. It is sensible to increase the solution time by increasing the time step to a level where the Courant number is much closer to 1. Therefore reset `deltaT` to 0.02 s and, on this occasion, set `startFrom` to `latestTime`. This instructs turbFoam to read the start data from the latest time directory, *i.e. 10.0*. The `endTime` should be set to 20 s since the run converges a lot slower than the laminar case. Restart the run as before and monitor the convergence of the solution.

2.1.9 Changing the case geometry

A user may wish to make changes to the geometry of a case and perform a new simulation. It may be useful to retain some or all of the original solution as the starting conditions for the new simulation. This is a little complex because the fields of the original solution are not consistent with the fields of the new case. However the mapFields utility can map fields that are inconsistent, either in terms of geometry or boundary types or both.

As an example, let us open in FoamX the case cavityClipped in the *icoFoam* directory which consists of the standard cavity geometry but with a square of length 0.04 m removed from the bottom right of the cavity, according to the *blockMeshDict* below:

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  convertToMeters 0.1;
```



```
26
27  vertices
28  (
29      (0 0 0)
30      (0.6 0 0)
31      (0 0.4 0)
32      (0.6 0.4 0)
33      (1 0.4 0)
34      (0 1 0)
35      (0.6 1 0)
36      (1 1 0)
37
38      (0 0 0.1)
39      (0.6 0 0.1)
40      (0 0.4 0.1)
41      (0.6 0.4 0.1)
42      (1 0.4 0.1)
43      (0 1 0.1)
44      (0.6 1 0.1)
45      (1 1 0.1)
46
47  );
48
49  blocks
50  (
51      hex (0 1 3 2 8 9 11 10) (12 8 1) simpleGrading (1 1 1)
52      hex (2 3 6 5 10 11 14 13) (12 12 1) simpleGrading (1 1 1)
53      hex (3 4 7 6 11 12 15 14) (8 12 1) simpleGrading (1 1 1)
54  );
55
56  edges
57  (
58  );
59
60  patches
61  (
62      wall lid
63      (
64          (5 13 14 6)
65          (6 14 15 7)
66      )
67      wall fixedWalls
68      (
69          (0 8 10 2)
70          (2 10 13 5)
71          (7 15 12 4)
72          (4 12 11 3)
73          (3 11 9 1)
74          (1 9 8 0)
75      )
76      empty frontAndBack
77      (
78          (0 2 3 1)
79          (2 5 6 3)
80          (3 6 7 4)
81          (8 9 11 10)
82          (10 11 14 13)
83          (11 12 15 14)
84      )
85  );
86
87  mergePatchPairs
88  (
89  );
90
91  // ************************************************************************* //
```

Generate the mesh with blockMesh and read the mesh into the case browser as described previously. Ensure that the patch types are set correctly: the `fixedWalls` and `lid` patches should be set to wall, the `lid` patch being the `movingWall` patch of the cavity case renamed for sake of clarity. Now **save the case**, which saves the field data into time directory *0.5* since the `startTime` is set to 0.5 in the *controlDict*. View the geometry and fields at 0.5 s.

Now we wish to map the velocity and pressure fields from cavity onto the new fields of cavityClipped. Simply open the mapFields utility as before in FoamX and edit the arguments: set the root and case of the source and target but make sure the `-consistent` option is

switched `off`.

Return to the mapFields utility window and select Edit Dictionary. A window opens containing the arguments followed by 2 entries: `patchMap` and `cuttingPatches`. The `patchMap` list contains a mapping of patches from the source fields to the target fields. It is used if the user wishes a patch in the target field to inherit values from a corresponding patch in the source field. In cavityClipped, we wish to inherit the boundary values on the `lid` patch from `movingWall` in cavity so we must set the `patchMap` as:

```
patchMap
(
    lid movingWall
);
```

The `cuttingPatches` list contains names of target patches whose values are to be mapped from the source internal field through which the target patch cuts. In this case we will include the `fixedWalls` to demonstrate the interpolation process.

```
cuttingPatches
(
    fixedWalls
);
```

Now the user should run mapFields, either from FoamX or from the command line:

```
cd $FOAM_RUN/tutorials/icoFoam
mapFields . cavity . cavityClipped
```

The user should now select the `Read Mesh&Fields` function to load the new fields into FoamX. The user can view the mapped field as shown in Figure 2.18. The boundary patches have inherited values from the source case as we expected. Having demonstrated this, however, we actually wish to reset the velocity on the `fixedWalls` patch to $(0, 0, 0)$. Open the `U` field in FoamX, select the `fixedWalls` patch and change the field from `nonuniform` to `uniform` $(0, 0, 0)$. Now run the case with icoFoam.

2.1.10 Post-processing the modified geometry

Velocity glyphs can be generated for the case as normal, first at time 0.5 s and later at time 0.6 s, to compare the initial and final solutions. In addition, we provide an outline of the geometry which requires some care to generate for a 2D case. The user should select `Extract Parts` from the `Filter` menu and, in the Parameter panel, highlight the patches of interest, namely the lid and fixedWalls. On clicking Accept, these items of geometry can be displayed by selecting `Wireframe Of Surface` in the Display panel. Figure 2.19 displays the patches in black and shows vortices forming in the bottom corners of the modified geometry.

2.2 Stress analysis of a plate with a hole

This tutorial describes how to pre-process, run and post-process a case involving linear-elastic, steady-state stress analysis on a square plate with a circular hole at its centre. The

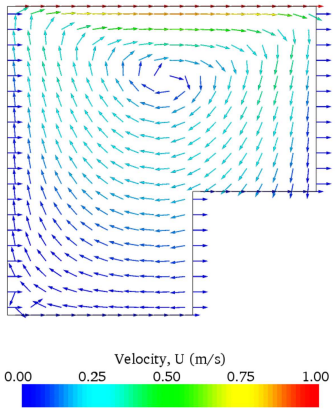


Figure 2.18: cavity solution velocity field mapped onto cavityClipped.

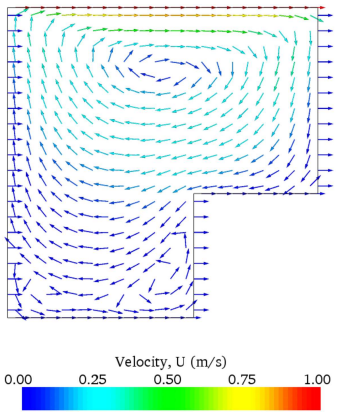


Figure 2.19: cavityClipped solution for velocity field.

plate dimensions are: side length 4 m and radius $R = 0.5$ m. It is loaded with a uniform traction of $\sigma = 10$ kPa over its left and right faces as shown in Figure 2.20. Two symmetry planes can be identified for this geometry and therefore the solution domain need only cover a quarter of the geometry, shown by the shaded area in Figure 2.20.

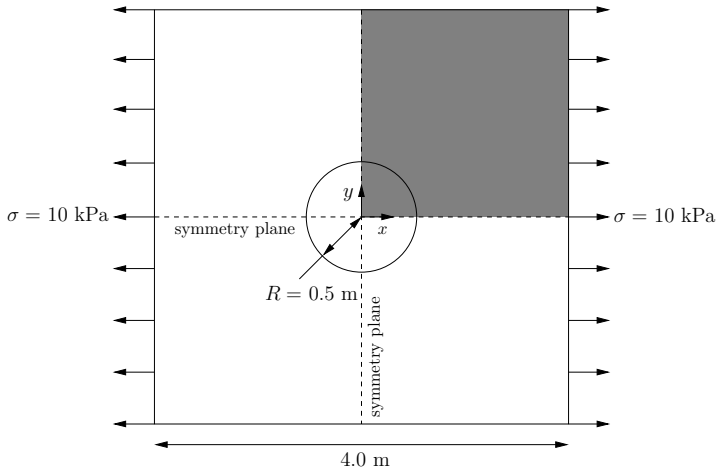


Figure 2.20: Geometry of the plate with a hole.

The problem can be approximated as 2-dimensional since the load is applied in the plane of the plate. In a Cartesian coordinate system there are two possible assumptions to take in regard to the behaviour of the structure in the third dimension: (1) the plane stress condition, in which the stress components acting out of the 2D plane are assumed to be negligible; (2) the plane strain condition, in which the strain components out of the 2D plane are assumed negligible. The plane stress condition is appropriate for solids whose third dimension is thin as in this case; the plane strain condition is applicable for solids where the third dimension is thick.

An analytical solution exists for loading of an infinitely large, thin plate with a circular hole. The solution for the stress normal to the vertical plane of symmetry is

$$(\sigma_{xx})_{x=0} = \begin{cases} \sigma\left(1 + \frac{R^2}{2y^2} + \frac{3R^4}{2y^4}\right) & \text{for } |y| \geq R \\ 0 & \text{for } |y| < R \end{cases} \quad (2.14)$$

Results from the simulation will be compared with this solution. At the end of the tutorial, the user can: investigate the sensitivity of the solution to mesh resolution and mesh grading; and, increase the size of the plate in comparison to the hole to try to estimate the error in comparing the analytical solution for an infinite plate to the solution of this problem of a finite plate.

2.2.1 Mesh generation

The domain consists of four blocks, some of which have arc-shaped edges. The block structure for the part of the mesh in the $x-y$ plane is shown in Figure 2.21. As already mentioned in section 2.1.1.1, all geometries are generated in 3 dimensions in OpenFOAM even if the case is to be as a 2 dimensional problem. Therefore a dimension of the block in the z direction has to be chosen; here, 0.5 m is selected. It does not affect the solution since the traction boundary condition is specified as a stress rather than a force, thereby making the solution independent of the cross-sectional area.

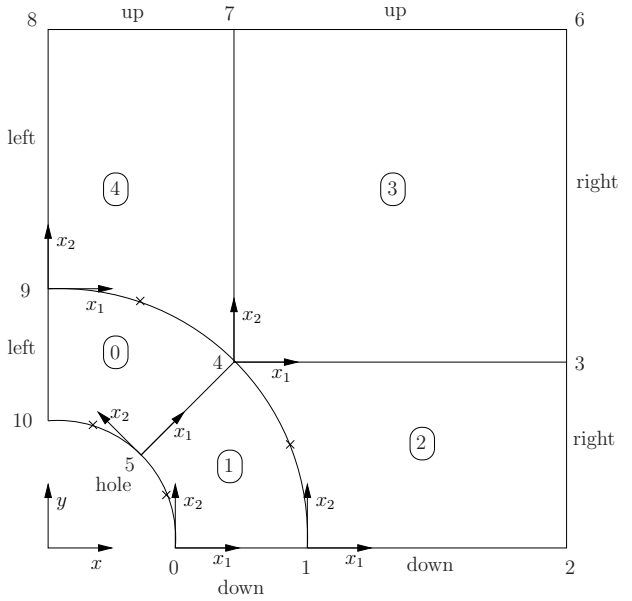


Figure 2.21: Block structure of the mesh for the plate with a hole.

The user should start up FoamX as normal and open the plateHole case in their own *$FOAM_RUN/tutorials/solidDisplacementFoam* directory. Open the blockMesh utility as described previously in section 2.1.1.1 and press the Edit Dictionary button. Until now, we have only specified straight edges in the geometries of previous tutorials but here we need to specify curved edges. The user should select `edges` to view how they are specified as shown in Figure 2.22. All 8 curved edges in this example are listed in a table; each can be edited which opens a selection window offering different types of curve, including `arc`, `simpleSpline`, `polyLine` *etc.*, described further in section 6.3.1. In this example, all the edges are circular and so can be specified by the `arc` keyword entry. Editing the `arc` selection opens a window in which the user specifies the labels of the start and end vertices of the arc and a point vector through which the circular arc passes.

The blocks in this *blockMeshDict* do not all have the same orientation. As can be seen in

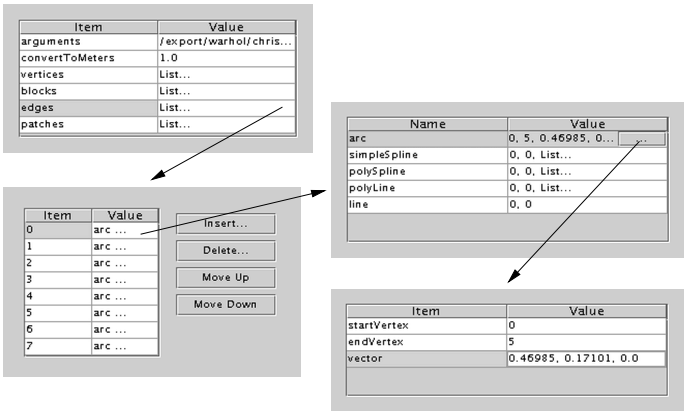


Figure 2.22: Specifying curved edges in blockMesh.

Figure 2.21 the x_2 direction of block 0 is equivalent to the $-x_1$ direction for block 4. This means care must be taken when defining the number and distribution of cells in each block so that the cells match up at the block faces.

5 patches are defined, one for each side of the plate and one for the hole. The *blockMeshDict* dictionary entries are given below.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    (0.5 0 0)
    (1 0 0)
    (2 0 0)
    (2 0.707107 0)
    (0.707107 0.707107 0)
    (0.353553 0.353553 0)
    (2 2 0)
    (0.707107 2 0)
    (0 2 0)
    (0 1 0)
    (0 0.5 0)
    (0.5 0 0.5)
    (1 0 0.5)
    (2 0 0.5)
    (2 0.707107 0.5)
    (0.707107 0.707107 0.5)
    (0.353553 0.353553 0.5)
    (2 2 0.5)
    (0.707107 2 0.5)
    (0 2 0.5)
    (0 1 0.5)
    (0 0.5 0.5)
);

blocks
(
    hex (5 4 9 10 16 15 20 21) (10 10 1) simpleGrading (1 1 1)
    hex (0 1 4 5 11 12 15 16) (10 10 1) simpleGrading (1 1 1)
    hex (1 2 3 4 12 13 14 15) (20 10 1) simpleGrading (1 1 1)
    hex (4 3 6 7 15 14 17 18) (20 20 1) simpleGrading (1 1 1)
    hex (9 4 7 8 20 15 18 19) (10 20 1) simpleGrading (1 1 1)
);
```



```

edges
(
    arc 0 5 (0.469846 0.17101 0)
    arc 5 10 (0.17101 0.469846 0)
    arc 1 4 (0.939693 0.34202 0)
    arc 4 9 (0.34202 0.939693 0)
    arc 11 16 (0.469846 0.17101 0.5)
    arc 16 21 (0.17101 0.469846 0.5)
    arc 12 15 (0.939693 0.34202 0.5)
    arc 15 20 (0.34202 0.939693 0.5)
);

patches
(
    symmetryPlane left
    (
        (8 9 20 19)
        (9 10 21 20)
    )
    patch right
    (
        (2 3 14 13)
        (3 6 17 14)
    )
    symmetryPlane down
    (
        (0 1 12 11)
        (1 2 13 12)
    )
    patch up
    (
        (7 8 19 18)
        (6 7 18 17)
    )
    patch hole
    (
        (10 5 16 21)
        (5 0 11 16)
    )
    empty frontAndBack
    (
        (10 9 4 5)
        (5 4 1 0)
        (1 4 3 2)
        (4 7 6 3)
        (4 9 8 7)
        (21 16 15 20)
        (16 11 12 15)
        (12 13 14 15)
        (15 14 17 18)
        (15 18 19 20)
    )
);

mergePatchPairs
(
);

// ************************************************************************* //
```

The mesh should be generated using blockMesh and can be viewed in paraFoam as described in section 2.1.2. It should appear as in Figure 2.23.

2.2.1.1 Boundary and initial conditions

Once the mesh generation is complete, load the mesh into FoamX: remember this is done by clicking the right mouse button on `Mesh` in the case directory tree and selecting the `Read Mesh&Fields` function. The names of the patches will appear in a dictionary named Patches which should be set as follows: `left` and `down` patches are both symmetryPlane; the `undefinedFaces` are the front and back planes of the 2D geometry and should therefore be declared empty; the other patches are traction boundary conditions, set by traction boundary

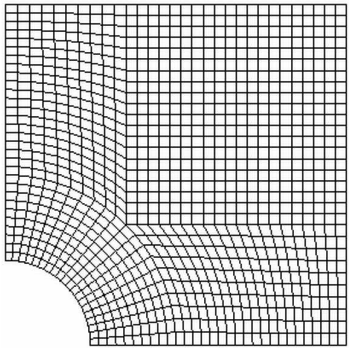


Figure 2.23: Mesh of the hole in a plate problem.

type.

The Fields must be set as before; here, the displacement `D` and temperature `T`. The traction boundary conditions are specified by a linear combination of: (1) a boundary traction vector; (2) a pressure that produces a traction normal to the boundary surface that is defined as negative when pointing out of the surface. The `up` and `hole` patches are zero traction so the boundary traction and pressure are set to zero. For the `right` patch the traction should be $(1e4, 0, 0)$ Pa and the pressure should be 0 Pa as shown in Figure 2.24. All the displacement initial conditions should be set to $(0, 0, 0)$ m.

2.2.1.2 Mechanical properties

The physical properties for the case are set in the *mechanicalProperties* dictionary. For this problem, we need to specify the mechanical properties of steel given in Table 2.1. In the mechanical properties dictionary, the user must also set `planeStress` to `yes`.

Property	Units	Keyword	Value
Density	$\mathrm{kg\,m^{-3}}$	`rho`	7854
Young's modulus	Pa	`E`	2×10^{11}
Poisson's ratio	—	`nu`	0.3

Table 2.1: Mechanical properties for steel

2.2.1.3 Thermal properties

The temperature field variable `T` is present in the solidDisplacementFoam solver since the user may opt to solve a thermal equation that is coupled with the momentum equation through the thermal stresses that are generated. The user specifies at run time whether OpenFOAM should solve the thermal equation by the `thermalStress` switch in the *thermalProperties* dictionary. This dictionary also sets the thermal properties for the case, *e.g.* for steel as listed in Table 2.2.

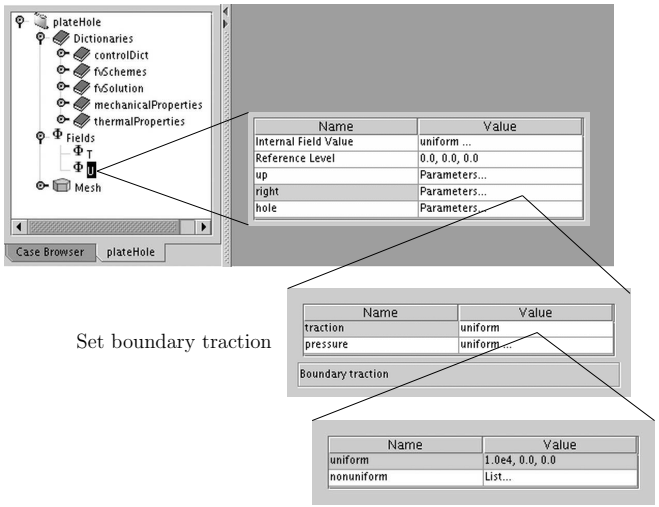


Figure 2.24: Setting the traction boundary condition.

Property	Units	Keyword	Value
Specific heat capacity	$\mathrm{Jkg^{-1}K^{-1}}$	`C`	434
Thermal conductivity	$\mathrm{Wm^{-1}K^{-1}}$	`k`	60.5
Thermal expansion coeff.	$\mathrm{K^{-1}}$	`alpha`	1.1×10^{-5}

Table 2.2: Thermal properties for steel

In this case we do not want to solve for the thermal equation. Therefore we must set the `thermalStress` keyword entry to `no` in the *thermalProperties* dictionary.

2.2.1.4 Control

As before, the information relating to the control of the solution procedure are read in from the *controlDict* dictionary. For this case, the `startTime` is 0 s. The time step is not important since this is a steady state case; in this situation it is best to set the time step `deltaT` to 1 so it simply acts as an iteration counter for the steady-state case. The `endTime`, set to 100, then acts as a limit on the number of iterations. The `writeInterval` can be set to 20.

The *controlDict* entries should be saved as follows:

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  application     solidDisplacementFoam;
26
27  startFrom       startTime;
28
29  startTime       0;
30
31  stopAt          endTime;
```

```
endTime         100;

deltaT          1;

writeControl    timeStep;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

graphFormat     raw;

runTimeModifiable yes;


// ************************************************************************* //
```

2.2.1.5 Discretisation schemes and linear-solver control

Let us turn our attention to the *fvSchemes* dictionary. Firstly, the problem we are analysing is steady-state so the user should select `SteadyState` for the time derivatives in `timeScheme`. This essentially switches off the time derivative terms. Not all solvers work for both steady-state and transient problems but solidDisplacementFoam does work, since the base algorithm is the same for both types of simulation.

The momentum equation in linear-elastic stress analysis includes several explicit terms containing the gradient of displacement. The calculations benefit from accurate and smooth evaluation of the gradient. Normally, in the finite volume method the discretisation is based on Gauss's theorem, as discussed in section 4.4.3 and section 2.4.6 of the Programmer's Guide. The Gauss method is sufficiently accurate for most purposes but, in this case, the least squares method will be used. The user should therefore open the `fvSchemes` dictionary and select `leastSquares` for the `grad(U)` gradient discretisation scheme.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

d2dt2Schemes
{
    default     steadyState;
}

gradSchemes
{
    default         leastSquares;
    grad(D)         leastSquares;
    grad(T)         leastSquares;
}

divSchemes
{
    default         none;
    div(sigmaD)     Gauss linear;
}

laplacianSchemes
{
    default         none;
    laplacian(DD,D) Gauss linear corrected;
    laplacian(DT,T) Gauss linear corrected;
}
```



```

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         none;
}

fluxRequired
{
    default         no;
    D               yes;
    T               no;
}


// ************************************************************************* //
```

The *fvSolution* dictionary controls the linear equation solvers and algorithms used in the solution. The user should first look at the *solvers* subdictionary and select either the `GAMG` solver with entries listed below or the simpler `PCG` solver for `D`. The solver `tolerance` should be set to 10^{-6} for this problem. The solver relative tolerance, denoted by `relTol`, sets the required reduction in the residuals within each iteration. It is uneconomical to set a high relative tolerance within each iteration since a lot of terms are explicit and are updated as part of the segregated iterative procedure. Therefore a reasonable value for the relative tolerance is 0.01, or possibly even higher, say 0.1.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    D GAMG
    {
        tolerance       1e-06;
        relTol          0.9;

        smoother        GaussSeidel;

        cacheAgglomeration true;

        nCellsInCoarsestLevel 20;

        agglomerator    faceAreaPair;
        mergeLevels     1;
    };

    T GAMG
    {
        tolerance       1e-06;
        relTol          0.9;

        smoother        GaussSeidel;

        cacheAgglomeration true;

        nCellsInCoarsestLevel 20;

        agglomerator    faceAreaPair;
        mergeLevels     1;
    };
}

stressAnalysis
{
    compactNormalStress yes;
    nCorrectors     1;
    D               1e-06;
}


// ************************************************************************* //
```

The *fvSolution* dictionary contains a subdictionary, *stressAnalysis* that contains some control parameters specific to the application solver. Firstly there is `nCorrectors` which specifies the number of outer loops around the complete system of equations, including traction boundary conditions *within each time step*. Since this problem is steady-state, we are performing a set of iterations towards a converged solution with the 'time step' acting as an iteration counter. We can therefore set `nCorrectors` to 1.

The `D` keyword specifies a convergence tolerance for the outer iteration loop, *i.e.* sets a level of initial residual below which solving will cease. It should be set to the desired solver tolerance specified earlier, 10^{-6} for this problem.

2.2.2 Running the code

The user should run the code here in the background, either from FoamX or from the command line as specified below, so he/she can look at convergence information in the log file afterwards.

```
cd $FOAM_RUN/tutorials/solidDisplacementFoam
nohup nice -n 19 solidDisplacementFoam . plateHole > log &
```

The user should check the convergence information by viewing the generated *log* file which shows the number of iterations and the initial and final residuals of the displacement in each direction being solved. The final residual should always be less than 0.1 times the initial residual as this iteration tolerance set. Once both initial residuals have dropped below the convergence tolerance of 10^{-6} the run has converged and can be stopped by killing the batch job.

2.2.3 Post-processing

Post processing can be performed as in section 2.1.4. The solidDisplacementFoam solver outputs the stress field σ as a symmetric tensor field `sigma`. This is consistent with the way variables are usually represented in OpenFOAM solvers by the mathematical symbol by which they are represented; in the case of Greek symbols, the variable is named phonetically.

For postprocessing individual scalar field components, σ_{xx}, σ_{xy} *etc.*, can be generated by running the sigmaComponents utility on the case. Components named `sigmaxx`, `sigmaxy` *etc.* are written to time directories of the case. The σ_{xx} stresses can be viewed in paraFoam as shown in Figure 2.25.

We would like to compare the analytical solution of Equation 2.14 to our solution. We therefore must output a set of data of σ_{xx} along the left edge symmetry plane of our domain. The user may generate the required graph data using the sample utility. In the FoamX case server, the user should select select `sample` from the `postProcessing -> miscellaneous` sub-menu of the from the `Foam Utilities` menu. The user should select Edit Dictionary which opens a *sampleDict* dictionary, pre-specified in the tutorial case. The dictionary contains the entries as follows and summarised in Table 7.3. The sample line specified in `sampleSets` is set between $(0.0, 0.5, 0.25)$ and $(0.0, 2.0, 0.25)$. Clicking Execute will execute the sample utility.

The `writeFormat` is `raw` 2 column format. In an application such as GnuPlot, one could type the following at the command prompt would be sufficient to plot both the numerical data and analytical solution:

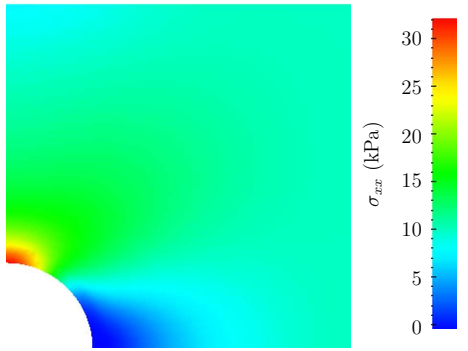


Figure 2.25: σ_{xx} stress field in the plate with hole.

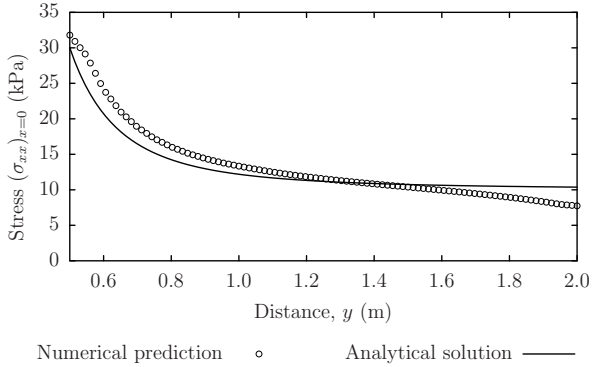


Figure 2.26: Normal stress along the vertical symmetry $(\sigma_{xx})_{x=0}$

```
plot [0.5:2] '<datafile>', 1e4*(1+(0.125/(x**2))+(0.09375/(x**4)))
```

An example plot is shown in Figure 2.26.

2.2.4 Exercises

The user may wish to experiment with solidDisplacementFoam by trying the following exercises:

2.2.4.1 Increasing mesh resolution

Increase the mesh resolution in each of the x and y directions. Use mapFields to map the final coarse mesh results from section 2.2.3 to the initial conditions for the fine mesh.

2.2.4.2 Introducing mesh grading

Grade the mesh so that the cells near the hole are finer than those away from the hole. Design the mesh so that the ratio of sizes between adjacent cells is no more than 1.1 and so that the ratio of cell sizes between blocks is similar to the ratios within blocks. Mesh grading is described in section 2.1.6. Again use mapFields to map the final coarse mesh results from section 2.2.3 to the initial conditions for the graded mesh. Compare the results with those from the analytical solution and previous calculations. Can this solution be improved upon using the same number of cells with a different solution?

2.2.4.3 Changing the plate size

The analytical solution is for an infinitely large plate with a finite sized hole in it. Therefore this solution is not completely accurate for a finite sized plate. To estimate the error, increase the plate size while maintaining the hole size at the same value.

2.3 Breaking of a dam

In this tutorial we shall solve a problem of simplified dam break in 2 dimensions using the interFoam.The feature of the problem is a transient flow of two fluids separated by a sharp interface, or free surface. The two-phase algorithm in interFoam is based on the volume of fluid (VOF) method in which a specie transport equation is used to determine the relative volume fraction of the two phases, or phase fraction γ, in each computational cell. Physical properties are calculated as weighted averages based on this fraction. The nature of the VOF method means that an interface between the species is not explicitly computed, but rather emerges as a property of the phase fraction field. Since the phase fraction can have any value between 0 and 1, the interface is never sharply defined, but occupies a volume around the region where a sharp interface should exist.

The test setup consists of a column of water at rest located behind a membrane on the left side of a tank. At time $t = 0$ s, the membrane is removed and the column of water collapses. During the collapse, the water impacts an obstacle at the bottom of the tank and creates a complicated flow structure, including several captured pockets of air. The geometry and the initial setup is shown in Figure 2.27.

2.3.1 Mesh generation

The user should start up FoamX as normal and open the *damBreak* case in their own *$FOAM_RUN/tutorials/interFoam* directory. Generate the mesh running blockMesh as described previously. The damBreak mesh consist of 5 blocks; the *blockMeshDict* entries are given below.

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  convertToMeters 0.146;
26
27  vertices
28  (
29      (0 0 0)
30      (2 0 0)
31      (2.16438 0 0)
32      (4 0 0)
33      (0 0.32876 0)
34      (2 0.32876 0)
```

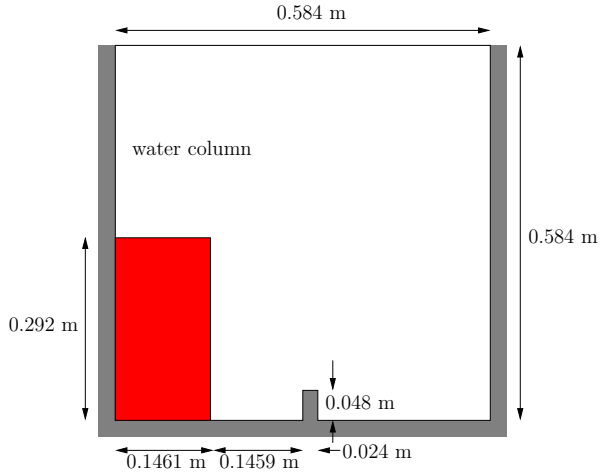



Figure 2.27: Geometry of the dam break.

```
35      (2.16438 0.32876 0)
36      (4 0.32876 0)
37      (0 4 0)
38      (2 4 0)
39      (2.16438 4 0)
40      (4 4 0)
41      (0 0 0.1)
42      (2 0 0.1)
43      (2.16438 0 0.1)
44      (4 0 0.1)
45      (0 0.32876 0.1)
46      (2 0.32876 0.1)
47      (2.16438 0.32876 0.1)
48      (4 0.32876 0.1)
49      (0 4 0.1)
50      (2 4 0.1)
51      (2.16438 4 0.1)
52      (4 4 0.1)
53  );
54
55  blocks
56  (
57      hex (0 1 5 4 12 13 17 16) (23 8 1) simpleGrading (1 1 1)
58      hex (2 3 7 6 14 15 19 18) (19 8 1) simpleGrading (1 1 1)
59      hex (4 5 9 8 16 17 21 20) (23 42 1) simpleGrading (1 1 1)
60      hex (5 6 10 9 17 18 22 21) (4 42 1) simpleGrading (1 1 1)
61      hex (6 7 11 10 18 19 23 22) (19 42 1) simpleGrading (1 1 1)
62  );
63
64  edges
65  (
66  );
67
68  patches
69  (
70      wall leftWall
71      (
72          (0 12 16 4)
73          (4 16 20 8)
74      )
75      wall rightWall
76      (
77          (7 19 15 3)
```

```
            (11 23 19 7)
        )
        wall lowerWall
        (
            (0 1 13 12)
            (1 5 17 13)
            (5 6 18 17)
            (2 14 18 6)
            (2 3 15 14)
        )
        patch atmosphere
        (
            (8 20 21 9)
            (9 21 22 10)
            (10 22 23 11)
        )
    );

    mergePatchPairs
    (
    );

    // ************************************************************************* //
```

2.3.2 Boundary conditions

Set the boundary conditions using the same procedure as described before. This should correspond to the description in Figure 2.27 although there is an important issue relating to the walls: the interFoam solver includes modelling of surface tension at the contact point between the interface and wall surface. If the user selects the wallContactAngle boundary type in FoamX, they will notice that the gamma (γ) field is assigned a gammaContactAngle boundary condition. The user must then specify the following: a static contact angle, `theta0` θ_0; leading and trailing edge dynamic contact angles, `thetaA` θ_A and `thetaR` θ_R respectively; and a velocity scaling function for dynamic contact angle, `uTheta`.

In this tutorial we would like to ignore surface tension effects between the wall and interface. We can do this by setting the static contact angle, $\theta_0 = 90°$ and the velocity scaling function to 0. However, there is also an option which we shall choose here to select a basic wall boundary type for the walls. Rather than use the gammaContactAngle boundary condition for gamma, this specifies a zeroGradient type instead.

The `top` boundary is free to the atmosphere and so is given an atmosphere boundary type; the `defaultFaces` representing the front and back planes of the 2D problem, is, as usual, an empty type.

2.3.3 Setting initial field

Unlike the previous cases, we shall now specify a non-uniform initial condition for the phase fraction γ where

$$\gamma = \begin{cases} 1 & \text{for the liquid phase} \\ 0 & \text{for the gas phase} \end{cases} \tag{2.15}$$

This will be done by running the setFields utility that can be accessed in FoamX from the `parallelProcessing` menu of `Foam Utilities`. The *setFieldsDict* dictionary entries for this case are shown below.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
```



```
    volScalarFieldValue gamma 0
    volVectorFieldValue U (0 0 0)
);

regions
(
    boxToCell
    {
        box (0 0 -1) (0.1461 0.292 1);

        fieldValues
        (
            volScalarFieldValue gamma 1
        );
    }
);


// ************************************************************************* //
```

The `defaultFieldValues` sets the default value of the fields, *i.e.* the value the field takes unless specified otherwise in the `regions` subdictionary. It contains a list of subdictionaries containing `fieldValues` that override the defaults in a specified region. The region is expressed in terms of a `topoSetSource` that creates a set of points, cells or faces based on some topological constraint. Here, `boxToCell` creates a bounding box within a vector minimum and maximum to define the set of cells of the liquid region. The phase fraction γ is defined as 1 in this region.

The user should execute setFields as any other utility is executed. Following that, the user **must** ensure the changes to the fields are read into FoamX by clickling `Read Mesh&-Fields`. Check the gamma field in FoamX to see that it is now `nonuniform`. Using paraFoam, check that the initial gamma field corresponds to the desired distribution as in Figure 2.28.

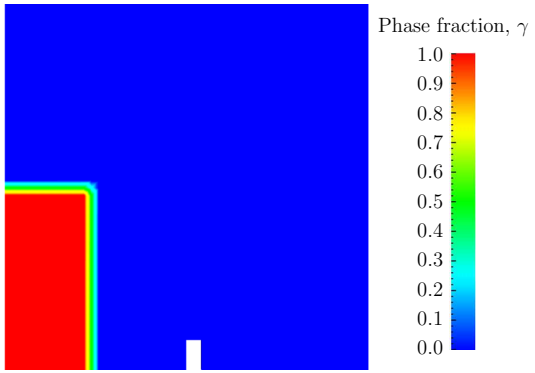


Figure 2.28: Initial conditions for phase fraction gamma.

2.3.4 Fluid properties

Let us examine the case setup from within FoamX. The *transportProperties* dictionary contains the material properties for each fluid, separated into two subdictionaries *phase1* and *phase2*. The transport model for each phase is selected by the `transportModel` keyword.

The user should select `Newtonian` in which case the kinematic viscosity is single valued and specified under the keyword `nu`. The viscosity parameters for the other models, *e.g.* `CrossPowerLaw`, are specified within subdictionaries with the generic name <*model*>*Coeffs*, *i.e.* *CrossPowerLawCoeffs* in this example. The density is specified under the keyword `rho`.

The surface tension between the two phases is specified under the keyword `sigma`. The values used in this tutorial are listed in Table 2.3.

`phase1` properties			
Kinematic viscosity	$m^2\,s^{-1}$	`nu`	1.0×10^{-6}
Density	$kg\,m^{-3}$	`rho`	1.0×10^{3}
`phase2` properties			
Kinematic viscosity	$m^2\,s^{-1}$	`nu`	1.48×10^{-5}
Density	$kg\,m^{-3}$	`rho`	1.0
Properties of both phases			
Surface tension	$N\,m^{-1}$	`sigma`	0.07

Table 2.3: Fluid properties for the damBreak tutorial

The *environmentalProperties* dictionary specifies the gravity acceleration vector which should be set to $(0, 9.81, 0)\ m\,s^{-2}$ for this tutorial.

2.3.5 Time step control

Time step control is an important issue in free surface tracking since the surface-tracking algorithm is considerably more sensitive to the Courant number Co than in standard fluid flow calculations. Ideally, we should not exceed an upper limit $Co \approx 0.2$ in the region of the interface. In some cases, where the propagation velocity is easy to predict, the user should specify a fixed time-step to satisfy the Co criterion. For more complex cases, this is considerably more difficult. interFoam therefore offers automatic adjustment of the time step as standard in the *controlDict*. The user should specify `adjustTimeStep` to be `on` and the the maximum Co, `maxCo` to be 0.2. The upper limit on time step `maxDeltaT` can be set to a value that will not be exceeded in this simulation, *e.g.* 1.0.

By using automatic time step control, the steps themselves are never rounded to a convenient value. Consequently if we request that OpenFOAM saves results at a fixed number of time step intervals, the times at which results are saved are somewhat arbitrary. However even with automatic time step adjustment, OpenFOAM allows the user to specify that results are written at fixed times; in this case OpenFOAM forces the automatic time stepping procedure to adjust time steps so that it 'hits' on the exact times specified for write output. The user selects this with the `adjustableRunTime` option for `writeControl` in the *controlDict* dictionary. The *controlDict* dictionary entries should be:

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  application interFoam;
26
27  startFrom       startTime;
28
29  startTime       0;
30
```



```
31  stopAt          endTime;
32
33  endTime         1;
34
35  deltaT          0.001;
36
37  writeControl    adjustableRunTime;
38
39  writeInterval   0.05;
40
41  purgeWrite      0;
42
43  writeFormat     ascii;
44
45  writePrecision  6;
46
47  writeCompression uncompressed;
48
49  timeFormat      general;
50
51  timePrecision   6;
52
53  runTimeModifiable yes;
54
55  adjustTimeStep  yes;
56
57  maxCo           0.5;
58
59  maxDeltaT       1;
60
61
62  // ************************************************************************* //
```

2.3.6 Discretisation schemes

The free surface treatment in OpenFOAM does not account for the effects of turbulence. This is a consequence of the fact that the Reynolds averaged approach to turbulence modelling does not match the notion of an infinitesimally thin interface between air and water. As a consequence, all free surface simulations can be viewed as a direct numerical simulation (DNS) of fluid flow. DNS is associated with certain requirements on the mesh size, far beyond the mesh resolution of our test case.

This solver uses the multidimensional universal limiter for explicit solution (MULES) method, created by OpenCFD, to maintain boundedness of the phase fraction independent of underlying numerical scheme, mesh structure, *etc.*. The choice of schemes for convection are therfore not restricted to those that are strongly stable or bounded, *e.g.* upwind differencing.

The convection schemes settings are made in the *divSchemes* subdictionary of the *fvSchemes* dictionary. In this example, the convection term in the momentum equation ($\nabla \bullet (\rho\phi\mathbf{U})$), denoted by the `div(rho*phi,U)` keyword, uses `Gauss limitedLinearV 1.0` to produce good accuracy. The limited linear schemes require a coefficient ϕ as described in section 4.4.1. Here, we have opted for best stability with $\phi = 1.0$. The $\nabla \bullet (\phi\gamma)$ term, represented by the `div(phi,gamma)` keyword uses the `vanLeer` scheme. The $\nabla \bullet (\phi_{rb}\gamma)$ term, represented by the `div(phirb,gamma)` keyword, can similarly use the `vanLeer` scheme, but generally produces smoother interfaces using the specialised `interfaceCompression` scheme.

The other discretised terms use commonly employed schemes so that the *fvSchemes* dictionary entries should therefore be:

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  ddtSchemes
26  {
27      default Euler;
28  }
29
30  gradSchemes
31  {
```

```
    default         Gauss linear;
    grad(U)         Gauss linear;
    grad(gamma)     Gauss linear;
}

divSchemes
{
    div(rho*phi,U)  Gauss limitedLinearV 1;
    div(phi,gamma)  Gauss vanLeer;
    div(phirb,gamma) Gauss interfaceCompression;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    pd;
    pcorr;
    gamma;
}


// ************************************************************************* //
```

2.3.7 Linear-solver control

In the *fvSolution*, the *PISO* subdictionary contains elements that are specific to interFoam. There are the usual correctors to the momentum equation but also correctors to a PISO loop around the γ phase equation. Of particular interest are the `nGammaSubCycles` and `cGamma` keywords. `nGammaSubCycles` represents the number of sub-cycles within the γ equation; sub-cycles are additional solutions to an equation within a given time step. It is used to enable the solution to be stable without reducing the time step and vastly increasing the solution time. Here we specify 4 sub-cycles, which means that the γ equation is solved in $4\times$ quarter length time steps within each actual time step.

The `cGamma` keyword is a factor that controls the compression of the interface where: 0 corresponds to no compression; 1 corresponds to conservative compression; and, anything larger than 1, relates to enhanced compression of the interface. We generally recommend a value of 1.0 which is employed in this example.

2.3.8 Running the code

Running of the code has been described in detail in previous tutorials. The job can be launched from FoamX or manually. Try the following:

```
cd $FOAM_RUN/tutorials/interFoam
interFoam . damBreak | tee log
```

The code will now be run interactively, with a copy of output stored in the *log* file.

2.3.9 Post-processing

Post-processing of the results can now be done in the usual way. The user can monitor the development of the phase fraction `gamma` in time; Figure 2.29. If a comparison with experimental data is required, a series of pictures for this case can be found in [**?**].

2.3.10 Running in parallel

The results from the previous example are generated using a fairly coarse mesh. We now wish to increase the mesh resolution and re-run the case. The new case will typically take a few hours to run with a single processor so, should the user have access to multiple processors, we can demonstrate the parallel processing capability of OpenFOAM.

The user should first make a copy of the damBreak case, *e.g.* using the `Clone Case` function in the FoamX case browser. The new case should be named damBreakFine. Open the new case and change the `blocks` description in the *blockMeshDict* dictionary to

```
blocks
(
    hex (0 1 5 4 12 13 17 16) (46 10 1) simpleGrading (1 1 1)
    hex (2 3 7 6 14 15 19 18) (40 10 1) simpleGrading (1 1 1)
    hex (4 5 9 8 16 17 21 20) (46 76 1) simpleGrading (1 2 1)
    hex (5 6 10 9 17 18 22 21) (4 76 1) simpleGrading (1 2 1)
    hex (6 7 11 10 18 19 23 22) (40 76 1) simpleGrading (1 2 1)
);
```

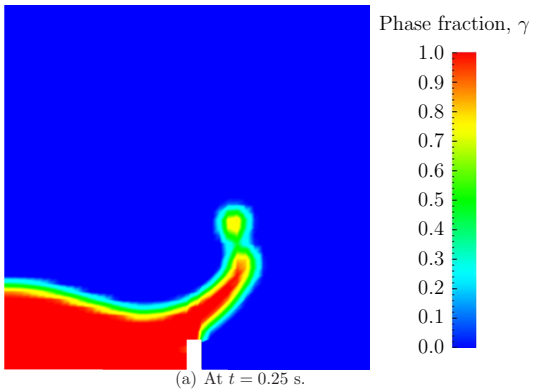
Here, the entry is presented as printed from the *blockMeshDict* file; in short the user must change the mesh densities, *e.g.* the `46 10 1` entry, and some of the mesh grading entries to `1 2 1`. Once the dictionary is correct, generate the mesh.

As the mesh has now changed from the damBreak example, the user must re-initialise the phase field `gamma` in the *0* time directory since it contains a number of elements that is inconsistent with the new mesh. Note that there is no need to change the `U` and `p` fields since they are specified as `uniform` which is independent of the number of elements in the field. We wish to initialise the field with a sharp interface, *i.e.* it elements would have $\gamma = 1$ or $\gamma = 0$. Updating the field with mapFields may produce interpolated values $0 < \gamma < 1$ at the interface, so it is better to rerun the setFields utility by:

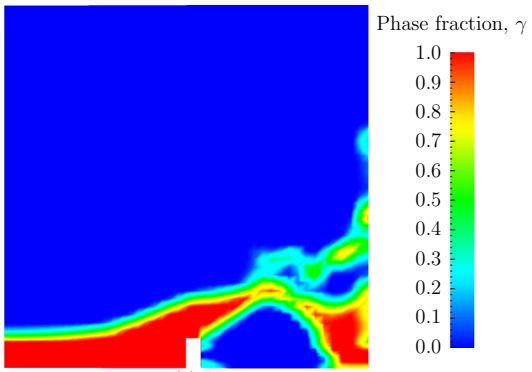
```
setFields $FOAM_RUN/tutorials/interFoam damBreakFine
```

As in the damBreak case, if the user is using FoamX, they should select `Read Mesh&Fields` which updates the case server to the changes to `gamma` caused by running the setFields externally of FoamX.

The method of parallel computing used by OpenFOAM is known as domain decomposition, in which the geometry and associated fields are broken into pieces and allocated to separate processors for solution. The first step required to run a parallel case is therefore to decompose the domain using the decomposePar utility which can be selected from the `parallelProcessing` menu of `Foam Utilities` in FoamX. The user must edit the dictionary associated with decomposePar named *decomposeParDict* which is located in the *system* directory of the case. The first entry is `numberOfSubdomains` which specifies the number of



(a) At $t = 0.25$ s.



(b) At $t = 0.50$ s.

Figure 2.29: Snapshots of phase γ.

subdomains into which the case will be decomposed, usually corresponding to the number of processors available for the case.

In this tutorial, the `method` of decomposition should be `simple` and the corresponding `simpleCoeffs` should be edited according to the following criteria. The domain is split into pieces, or subdomains, in the x, y and z directions, the number of subdomains in each direction being given by the vector $\mathbf{n}$. As this geometry is 2 dimensional, the 3rd direction, z, cannot be split, hence n_z must equal 1. The n_x and n_y components of $\mathbf{n}$ split the domain in the x and y directions and must be specified so that the number of subdomains specified by n_x and n_y equals the specified `numberOfSubdomains`, *i.e.* $n_x n_y =$ `numberOfSubdomains`. It is beneficial to keep the number of cell faces adjoining the subdomains to a minimum so, for a square geometry, it is best to keep the split between the x and y directions should be fairly even. The `delta` keyword should be set to 0.001.

For example, let us assume we wish to run on 4 processors. We would set `numberOfSubdomains` to 4 and $\mathbf{n} = (2, 2, 1)$. We close the *decomposeParDict* and run decomposePar. The screen messages of decomposePar can be monitored and show that the decomposition is distributed fairly even between the processors.

Parallel runs can currently only be executed from the command line. The user should consult section 3.4 for details of how to run a case in parallel; in this tutorial we merely present an example of running in parallel. We use the openMPI implementation of the standard message-passing interface (MPI). As a test here, the user can run in parallel on a single node, the local host only, by typing:

```
mpirun -np 4 interFoam $FOAM_RUN/tutorials/interFoam damBreakFine
    -parallel > log &
```

The user may run on more nodes over a network by creating a file that lists the host names of the machines on which the case is to be run as described in section 3.4.2. The case should run in the background and the user can follow its progress by monitoring the *log* file as usual.

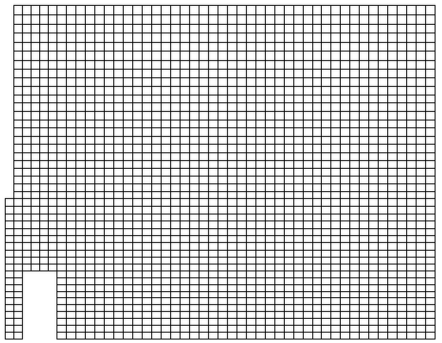
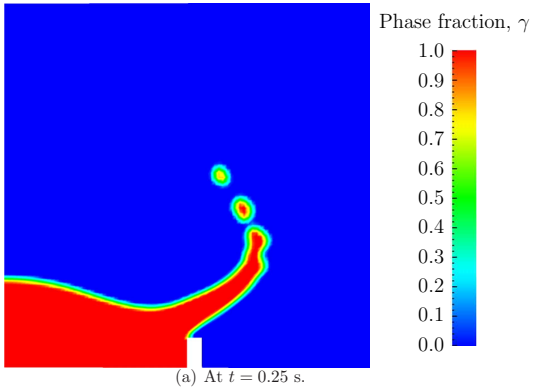
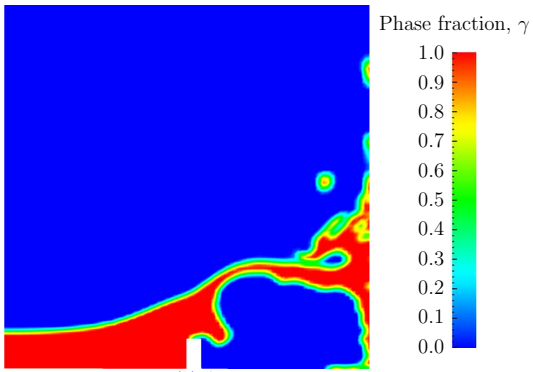


Figure 2.30: Mesh of processor 2 in parallel processed case.



(a) At $t = 0.25$ s.



(b) At $t = 0.50$ s.

Figure 2.31: Snapshots of phase γ with refined mesh.

2.3.11 Post-processing a case run in parallel

Once the case has completed running, the decomposed fields and mesh must be reassembled for post-processing using the reconstructPar utility. Simply select the utility from the `parallelProcessing` menu of `Foam Utilities` in FoamX or run it from the command line. The results from the fine mesh are shown in Figure 2.31. The user can see that the resolution of interface has improved significantly compared to the coarse mesh.

The user may also post-process a segment of the decomposed domain individually by simply treating the individual processor directory as a case in its own right. For example if the user starts paraFoam by

```
paraFoam $FOAM_RUN/tutorials/interFoam/damBreakFine processor2
```

then processor2 will appear as a case module in ParaView. Figure 2.30 shows the mesh from processor 2 following the decomposition of the domain using the `simple` method.

Chapter 3

Applications and libraries

We should reiterate from the outset that OpenFOAM is a C++ library used primarily to create executables, known as *applications*. OpenFOAM is distributed with a large set of precompiled applications but users also have the freedom to create their own or modify existing ones. Applications are split into two main categories:

solvers that are each designed to solve a specific problem in computational continuum mechanics;

utilities that perform simple pre-and post-processing tasks, mainly involving data manipulation and algebraic calculations.

OpenFOAM is divided into a set of precompiled libraries that are dynamically linked during compilation of the solvers and utilities. Libraries such as those for physical models are supplied as source code so that users may conveniently add their own models to the libraries.

This chapter gives an overview of solvers, utilities and libraries, their creation, modification, compilation and execution. The actual writing of code for solvers and utilities is not described here but is within the Programmer's Guide. The Programmer's Guide is currently under development so, if users have any queries, further information may also available at the OpenFOAM discussion group and the OpenFOAM web site.

3.1 The programming language of OpenFOAM

In order to understand the way in which the OpenFOAM library works, some background knowledge of C++, the base language of OpenFOAM, is required; the necessary information will be presented in this chapter. Before doing so, it is worthwhile addressing the concept of language in general terms to explain some of the ideas behind object-oriented programming and our choice of C++ as the main programming language of OpenFOAM.

3.1.1 Language in general

The success of verbal language and mathematics is based on efficiency, especially in expressing abstract concepts. For example, in fluid flow, we use the term "velocity field", which has meaning without any reference to the nature of the flow or any specific velocity data. The term encapsulates the idea of movement with direction and magnitude and relates to other physical properties. In mathematics, we can represent velocity field by a single symbol, *e.g.*

U, and express certain concepts using symbols, *e.g.* "the field of velocity magnitude" by $|\mathbf{U}|$. The advantage of mathematics over verbal language is its greater efficiency, making it possible to express complex concepts with extreme clarity.

The problems that we wish to solve in continuum mechanics are not presented in terms of intrinsic entities, or types, known to a computer, *e.g.* bits, bytes, integers. They are usually presented first in verbal language, then as partial differential equations in 3 dimensions of space and time. The equations contain the following concepts: scalars, vectors, tensors, and fields thereof; tensor algebra; tensor calculus; dimensional units. The solution to these equations involves discretisation procedures, matrices, solvers, and solution algorithms. The topics of tensor mathematics and numerics are the subjects of chapter 1 and chapter 2 of the Programmer's Guide.

3.1.2 Object-orientation and C++

Progamming languages that are object-oriented, such as C++, provide the mechanism — *classes* — to declare types and associated operations that are part of the verbal and mathematical languages used in science and engineering. Our velocity field introduced earlier can be represented in programming code by the symbol `U` and "the field of velocity magnitude" can be `mag(U)`. The velocity is a vector field for which there should exist, in an object-oriented code, a vectorField class. The velocity field `U` would then be an instance, or *object*, of the vectorField class; hence the term object-oriented.

The clarity of having objects in programming that represent physical objects and abstract entities should not be underestimated. The class structure concentrates code development to contained regions of the code, *i.e.* the classes themselves, thereby making the code easier to manage. New classes can be derived or inherit properties from other classes, *e.g.* the vectorField can be derived from a vector class and a Field class. C++ provides the mechanism of *template classes* such that the template class Field<Type> can represent a field of any <Type>, *e.g.* scalar, vector, tensor. The general features of the template class are passed on to any class created from the template. Templating and inheritance reduce duplication of code and create class hierarchies that impose an overall structure on the code.

3.1.3 Equation representation

A central theme of the OpenFOAM design is that the solver applications, written using the OpenFOAM classes, have a syntax that closely resembles the partial differential equations being solved. For example the equation

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \bullet \phi \mathbf{U} - \nabla \bullet \mu \nabla \mathbf{U} = -\nabla p$$

is represented by the code

```
solve
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::laplacian(mu, U)
    ==
  - fvc::grad(p)
```

```
);
```

This and other requirements demand that the principal programming language of OpenFOAM has object-oriented features such as inheritance, template classes, virtual functions and operator overloading. These features are not available in many languages that purport to be object-orientated but actually have very limited object-orientated capability, such as FORTRAN-90. C++, however, possesses all these features while having the additional advantage that it is widely used with a standard specification so that reliable compilers are available that produce efficient executables. It is therefore the primary language of OpenFOAM.

3.1.4 Solver codes

Solver codes are largely procedural since they are a close representation of solution algorithms and equations, which are themselves procedural in nature. Users do not need a deep knowledge of object-orientation and C++ programming to write a solver but should know the principles behind object-orientation and classes, and to have a basic knowledge of some C++ code syntax. An understanding of the underlying equations, models and solution method and algorithms is far more important.

There is often little need for a user to immerse themselves in the code of any of the OpenFOAM classes. The essence of object-orientation is that the user should not have to; merely the knowledge of the class' existence and its functionality are sufficient to use the class. A description of each class, its functions *etc.* is supplied with the OpenFOAM distribution in HTML documentation generated with Doxygen at *$WM_PROJECT_DIR/doc/Doxygen/html/index.html*.

3.2 Compiling applications and libraries

Compilation is an integral part of application development that requires careful management since every piece of code requires its own set instructions to access dependent components of the OpenFOAM library. In UNIX/Linux systems these instructions are often organised and delivered to the compiler using the standard UNIX make utility. OpenFOAM, however, is supplied with the wmake compilation script that is based on make but is considerably more versatile and easier to use; wmake can, in fact, be used on any code, not simply the OpenFOAM library. To understand the compilation process, we first need to explain certain aspects of C++ and its file structure, shown schematically in Figure 3.1. A class is defined through a set of instructions such as object construction, data storage and class member functions. The file containing the class *definition* takes a *.C* extension, *e.g.* a class nc would be written in the file *nc.C*. This file can be compiled independently of other code into a binary executable library file known as a shared object library with the *.so* file extension, *i.e.* *nc.so*. When compiling a piece of code, say *newApp.C*, that uses the nc class, *nc.C* need not be recompiled, rather *newApp.C* calls *nc.so* at runtime. This is known as *dynamic linking*.

3.2.1 Header *.H* files

As a means of checking errors, the piece of code being compiled must know that the classes it uses and the operations they perform actually exist. Therefore each class requires a class

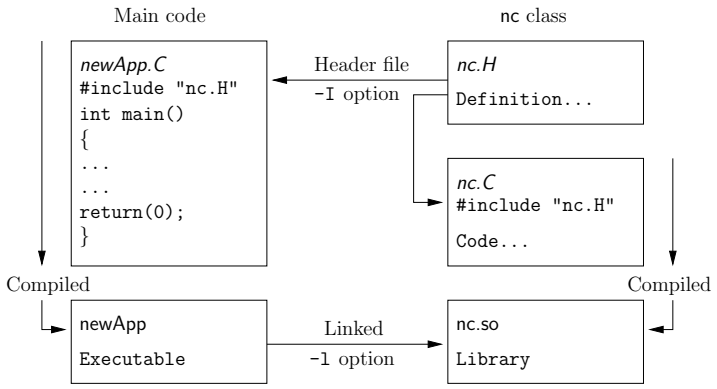


Figure 3.1: Header files, source files, compilation and linking.

declaration, contained in a header file with a *.H* file extension, *e.g.* *nc.H*, that includes the names of the class and its functions. This file is included at the beginning of any piece of code using the class, including the class declaration code itself. Any piece of *.C* code can resource any number of classes and must begin with all the *.H* files required to declare these classes. The classes in turn can resource other classes and begin with the relevant *.H* files. By searching recursively down the class hierarchy we can produce a complete list of header files for all the classes on which the top level *.C* code ultimately depends; these *.H* files are known as the *dependencies*. With a dependency list, a compiler can check whether the source files have been updated since their last compilation and selectively compile only those that need to be.

Header files are included in the code using `# include` statements, *e.g.*

```
# include "otherHeader.H";
```

causes the compiler to suspend reading from the current file to read the file specified. Any self-contained piece of code can be put into a header file and included at the relevant location in the main code in order to improve code readability. For example, in most OpenFOAM applications the code for creating fields and reading field input data is included in a file *createFields.H* which is called at the beginning of the code. In this way, header files are not solely used as class declarations. It is wmake that performs the task of maintaining file dependency lists amongst other functions listed below.

- Automatic generation and maintenance of file dependency lists, *i.e.* lists of files which are included in the source files and hence on which they depend.
- Multi-platform compilation and linkage, handled through appropriate directory structure.
- Multi-language compilation and linkage, *e.g.* C, C++, Java.
- Multi-option compilation and linkage, *e.g.* debug, optimised, parallel and profiling.

- Support for source code generation programs, *e.g.* lex, yacc, IDL, MOC.
- Simple syntax for source file lists.
- Automatic creation of source file lists for new codes.
- Simple handling of multiple shared or static libraries.
- Extensible to new machine types.
- Extremely portable, works on any machine with: `make`; `sh`, `ksh` or `csh`; `lex`, `cc`.
- Has been tested on Apollo, SUN, SGI, HP (HPUX), Compaq (DEC), IBM (AIX), Cray, Ardent, Stardent, PC Linux, PPC Linux, NEC, SX4, Fujitsu VP1000.

3.2.2 Compiling with wmake

OpenFOAM applications are organised using a standard convention that the source code of each application is placed in a directory whose name is that of the application. The top level source file takes the application name with the *.C* extension. For example, the source code for an application called newApp would reside is a directory *newApp* and the top level file would be *newApp.C* as shown in Figure 3.2. The directory must also contain a *Make*

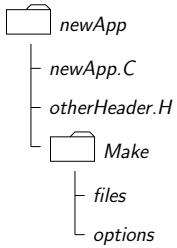


Figure 3.2: Directory structure for an application

subdirectory containing 2 files, *options* and *files*, that are described in the following sections.

3.2.2.1 Including headers

The compiler searches for the included header files in the following order, specified with the `-I` option in wmake:

1. the *$WM_PROJECT_DIR/src/OpenFOAM/lnInclude* directory;
2. a local *lnInclude* directory, *i.e.* *newApp/lnInclude*;
3. the local directory, *i.e.* *newApp*;
4. platform dependent paths set in files in the *$WM_PROJECT_DIR/wmake/rules/$WM_ARCH/* directory, *e.g.* */usr/X11/include* and *$(MPICH_ARCH_PATH)/include*;
5. other directories specified explicitly in the *Make/options* file with the `-I` option.

The *Make/options* file contains the full directory paths to locate header files using the syntax:

```
EXE_INC = \
    -I<directoryPath1> \
    -I<directoryPath2> \
    ...                \
    -I<directoryPathN>
```

Notice first that the directory names are preceeded by the `-I` flag and that the syntax uses the \ to continue the `EXE_INC` across several lines, with no \ after the final entry.

3.2.2.2 Linking to libraries

The compiler links to shared object library files in the following directory **paths**, specified with the `-L` option in wmake:

1. the *$FOAM_LIBBIN* directory;
2. platform dependent paths set in files in the *$WM_DIR/rules/$WM_ARCH/* directory, *e.g.* */usr/X11/lib* and *$(MPICH_ARCH_PATH)/lib*;
3. other directories specified in the *Make/options* file.

The actual library **files** to be linked must be specified using the `-l` option and removing the `lib` prefix and `.so` extension from the library file name, *e.g.* libnew.so is included with the flag `-lnew`. By default, wmake loads the following libraries:

1. the libOpenFOAM.so library from the *$FOAM_LIBBIN* directory;
2. platform dependent libraries specified in set in files in the *$WM_DIR/rules/$WM_ARCH/* directory, *e.g.* libm.so from */usr/X11/lib* and liblam.so from *$(LAM_ARCH_PATH)/lib*;
3. other libraries specified in the *Make/options* file.

The *Make/options* file contains the full directory paths and library names using the syntax:

```
EXE_LIBS = \
    -L<libraryPath1> \
    -L<libraryPath2> \
    ...              \
    -L<libraryPathN> \
    -l<library1>     \
    -l<library2>     \
    ...              \
    -l<libraryN>
```

Let us reiterate that the directory paths are preceeded by the `-L` flag, the library names are preceeded by the `-l` flag.

3.2.2.3 Source files to be compiled

The compiler requires a list of *.C* source files that must be compiled. The list must contain the main *.C* file but also any other source files that are created for the specific application but are not included in a class library. For example, users may create a new class or some new functionality to an existing class for a particular application. The full list of *.C* source files must be included in the *Make/files* file. As might be expected, for many applications the list only includes the name of the main *.C* file, *e.g.* *newApp.C* in the case of our earlier example.

The *Make/files* file also includes a full path and name of the compiled executable, specified by the `EXE =` syntax. Standard convention stipulates the name is that of the application, *i.e.* newApp in our example. The OpenFOAM release offers two useful choices for path: standard release applications are stored in *$FOAM_APPBIN*; applications developed by the user are stored in *$FOAM_USER_APPBIN*.

If the user is developing their own applications, we recommend they create an applications subdirectory in their *$WM_PROJECT_USER_DIR* directory containing the source code for personal OpenFOAM applications. As with standard applications, the source code for each OpenFOAM application should be stored within its own directory. The only difference between a user application and one from the standard release is that the *Make/files* file should specify that the user's executables are written into their *$FOAM_USER_APPBIN* directory. The *Make/files* file for our example would appear as follows:

```
newApp.C

EXE = $(FOAM_USER_APPBIN)/newApp
```

3.2.2.4 Running wmake

The wmake script is executed by typing:

```
wmake <optionalArguments> <optionalDirectory>
```

The <optionalDirectory> is the directory path of the application that is being compiled. Typically, wmake is executed from within the directory of the application being compiled, in which case <optionalDirectory> can be omitted.

If a user wishes to build an application executable, then no <optionalArguments> are required. However <optionalArguments> may be specified for building libraries *etc.* as described in Table 3.1.

Argument	Type of compilation
`lib`	Build a statically-linked library
`libso`	Build a dynamically-linked library
`libo`	Build a statically-linked object file library
`jar`	Build a JAVA archive
`exe`	Build an application independent of the specified project library.

Table 3.1: Optional compilation arguments to wmake.

3.2.2.5 wmake environment variables

For information, the environment variable settings used by wmake are listed in Table 3.2.

Main paths	
$WM_PROJECT_INST_DIR	Full path to installation directory, *e.g.* *$HOME/OpenFOAM*
$WM_PROJECT	Name of the project being compiled: `OpenFOAM`
$WM_PROJECT_VERSION	Version of the project being compiled: `1.4.1`
$WM_PROJECT_DIR	Full path to locate binary executables of OpenFOAM release, *e.g.* *$HOME/OpenFOAM/OpenFOAM-1.4.1*
$WM_PROJECT_USER_DIR	Full path to locate binary executables of the user *e.g.* *$HOME/OpenFOAM/${USER}-1.4.1*
Other paths/settings	
$WM_ARCH	Machine architecture: `cray` `decAlpha` `dec` `ibm` `linux` `linuxPPC` `sgi3` `sgi32` `sgi64` `sgiN32` `solaris` `sx4` `t3d`
$WM_COMPILER	Compiler being used: `Gcc3` - gcc 4.2.1, `KAI` - KAI
$WM_COMPILER_DIR	Compiler installation directory
$WM_COMPILER_BIN	Compiler installation binaries *$WM_COMPILER_BIN/bin*
$WM_COMPILER_LIB	Compiler installation libraries *$WM_COMPILER_BIN/lib*
$WM_COMPILE_OPTION	Compilation option: `Debug` - debugging, `Opt` optimisation.
$WM_DIR	Full path of the *wmake* directory
$WM_JAVAC_OPTION	Compilation option for JAVA: `Debug` - debugging, `Opt` optimisation.
$WM_LINK_LANGUAGE	Compiler used to link libraries and executables. In multi-language projects a $WM_LINK_LANGUAGE is set to the primary language.
$WM_MPLIB	Parallel communications library: `LAM`, `MPI`, `MPICH`, `PVM`
$WM_OPTIONS	= WM_ARCHWM_COMPILER... ...$WM_COMPILE_OPTION$WM_MPLIB *e.g.* `linuxGcc3OptMPICH`
$WM_PROJECT_LANGUAGE	Programming language of project, *e.g.* `c++`
$WM_SHELL	Shell used for the wmake scripts `bash`, `csh`, `ksh`, `tcsh`

Table 3.2: Environment variable settings for wmake.

3.2.3 Removing dependency lists: wclean and rmdepall

On execution, wmake builds a dependency list file with a *.dep* file extension, *e.g.* *newApp.dep* in our example, and a list of files in a *Make/$WM_OPTIONS* directory. If the user wishes to remove these files, perhaps after making code changes, the user can run the wclean script by typing:

```
wclean <optionalArguments> <optionalDirectory>
```

Again, the <optionalDirectory> is a path to the directory of the application that is being compiled. Typically, wclean is executed from within the directory of the application, in which case the path can be omitted.

If a user wishes to remove the dependency files and files from the *Make* directory, then no <optionalArguments> are required. However if lib is specified in <optionalArguments> a local *lnInclude* directory will be deleted also.

An additional script, rmdepall removes all dependency *.dep* files recursively down the directory tree from the point at which it is executed. This can be useful when updating OpenFOAM libraries.

3.2.4 Compilation example: the turbFoam application

The source code for application turbFoam is in the *$FOAM_APP/solvers/turbFoam* directory and the top level source file is named *turbFoam.C*. The *turbFoam.C* source code is:

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Application
    turbFoam

Description
    Transient solver for incompressible, turbulent flow.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "incompressible/singlePhaseTransportModel/singlePhaseTransportModel.H"
#include "incompressible/turbulenceModel/turbulenceModel.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{

#   include "setRootCase.H"

#   include "createTime.H"
#   include "createMesh.H"
#   include "createFields.H"
#   include "initContinuityErrs.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

```



```
#       include "readPISOControls.H"
#       include "CourantNo.H"

        // Pressure-velocity PISO corrector
        {
            // Momentum predictor

            fvVectorMatrix UEqn
            (
                fvm::ddt(U)
              + fvm::div(phi, U)
              + turbulence->divR(U)
            );

            if (momentumPredictor)
            {
                solve(UEqn == -fvc::grad(p));
            }

            // --- PISO loop

            for (int corr=0; corr<nCorr; corr++)
            {
                volScalarField rUA = 1.0/UEqn.A();

                U = rUA*UEqn.H();
                phi = (fvc::interpolate(U) & mesh.Sf())
                    + fvc::ddtPhiCorr(rUA, U, phi);

                adjustPhi(phi, U, p);

                // Non-orthogonal pressure corrector loop
                for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
                {
                    // Pressure corrector

                    fvScalarMatrix pEqn
                    (
                        fvm::laplacian(rUA, p) == fvc::div(phi)
                    );

                    pEqn.setReference(pRefCell, pRefValue);
                    pEqn.solve();

                    if (nonOrth == nNonOrthCorr)
                    {
                        phi -= pEqn.flux();
                    }
                }

#               include "continuityErrs.H"

                U -= rUA*fvc::grad(p);
                U.correctBoundaryConditions();
            }
        }

        turbulence->correct();

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return(0);
}


// ************************************************************************* //
```

The code begins with a brief description of the application contained within comments over 1 line (//) and multiple lines (/*...*/). Following that, the code contains several # include statements, *e.g.* # include "fvCFD.H", which causes the compiler to suspend reading from

the current file, *turbFoam.C* to read the *fvCFD.H*.

turbFoam resources the cfdTools, turbulenceModels and transportModels libraries and therefore requires the necessary header files, specified by the `EXE_INC = -I...` option, and links to the libraries with the `EXE_LIBS = -l...` option. The *Make/options* therefore contains the following:

```
EXE_INC = \
    -I$(LIB_SRC)/turbulenceModels \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/finiteVolume/lnInclude

EXE_LIBS = \
    -lincompressibleTurbulenceModels \
    -lincompressibleTransportModels \
    -lfiniteVolume \
    -lmeshTools
```

turbFoam contains only the *turbFoam.C* source and the executable is written to the *$FOAM_APPBIN* directory as all standard applications are. The *Make/files* therefore contains:

```
turbFoam.C

EXE = $(FOAM_APPBIN)/turbFoam
```

The user can compile turbFoam by going to the *$FOAM_CFD/turbFoam* directory and typing:

```
wmake
```

The code should compile and produce a message similar to the following

```
Making dependency list for source file turbFoam.C

SOURCE_DIR=.
SOURCE=turbFoam.C ;
g++ -DFOAM_EXCEPTION -Dlinux -DlinuxOptMPICH
-DscalarMachine -DoptSolvers -DPARALLEL -DUSEMPI -Wall -O2 -DNoRepository
-ftemplate-depth-17 -I/export/warhol/chris/OpenFOAM/OpenFOAM-1.4.1/src/OpenFOAM/lnInclude
-IlnInclude
-I.
......
-lmpich -L/usr/X11/lib -lm
-o /export/warhol/chris/OpenFOAM/OpenFOAM-1.4.1/applications/bin/linuxOptMPICH/turbFoam
```

The user can now try recompiling and will receive a message similar to the following to say that the executable is up to date and compiling is not necessary:

```
make: Nothing to be done for 'allFiles'.
make: 'Make/linuxOptMPICH/dependencies' is up to date.

make: '/export/warhol/chris/OpenFOAM/OpenFOAM-1.4.1/applications/bin/linuxOptMPICH/turbFoam'
is up to date.
```

The user can compile the application from scratch by removing the dependency list with

```
wclean
```

and running wmake.

3.2.5 Debug messaging and optimisation switches

OpenFOAM provides a system of messaging that is written during runtime, most of which are to help debugging problems encountered during running of a OpenFOAM case. The switches are listed in the *$WM_PROJECT_DIR/.OpenFOAM-1.4.1/controlDict* file; should the user wish to change the settings they should make a copy to their *$HOME* directory, *i.e.* *$HOME/.OpenFOAM-1.4.1/controlDict* file. The list of possible switches is extensive and can be viewed by running the foamDebugSwitches application. Most of the switches correspond to a class or range of functionality and can be switched on by their inclusion in the *controlDict* file, and by being set to `1`. For example, OpenFOAM can perform the checking of dimensional units in all calculations by setting the `dimensionSet` switch to `1`. There are some switches that control messaging at a higher level than most, listed in Table 3.3.

In addition, there are some switches that control certain operational and optimisation issues. These switches are also listed in Table 3.3. Of particular importance is `fileModificationSkew`. OpenFOAM scans the write time of data files to check for modification. When running over a NFS with some disparity in the clock settings on different machines, field data files appear to be modified ahead of time. This can cause a problem if OpenFOAM views the files as newly modified and attempting to re-read this data. The `fileModificationSkew` keyword is the time in seconds that OpenFOAM will subtract from the file write time when assessing whether the file has been newly modified.

High level debugging switches - subdictionary *DebugSwitches*	
`level`	Overall level of debugging messaging for OpenFOAM- - 3 levels `0`, `1`, `2`
`FoamX`	Debugging information messaging for FoamX
`lduMatrix`	Messaging for solver convergence during a run - 3 levels `0`, `1`, `2`

Optimisation switches - subdictionary *OptimisationSwitches*	
`fileModificationSkew`	A time in seconds that should be set higher than the maximum delay in NFS updates and clock difference for running OpenFOAM over a NFS.
`nProcsSimpleSum`	Optimises global sum for parallel processing; sets number of processors above which hierarchical sum is performed rather than a linear sum (default `16`)

Table 3.3: Runtime message switches.

3.2.6 Linking new user-defined libraries to existing applications

The situation may arise that a user creates a new library, say new, and wishes the features within that library to be available across a range of applications. For example, the user may create a new boundary condition, compiled into new, that would need to be recognised by a range of solver applications, pre- and post-processing utilities, mesh tools, *etc.* Under normal circumstances, the user would need to recompile every application with the new linked to it.

Instead, OpenFOAM uses a special library called foamUser to eliminate the need to recompile. It works by first having the foamUser library compiled into each application by default. The foamUser library is compiled from code located in *$FOAM_SRC/foamUser* directory. The user simply needs to add the new library to the linked libraries in the *Make/options* file of foamUser and recompile foamUser.

Taking the example already given, the user should therefore make a local copy of the *foamUser* directory, and move to that directory, *e.g.*:

```
cp -r $WM_PROJECT_DIR/src/foamUser $WM_PROJECT_USER_DIR/applications
cd $WM_PROJECT_USER_DIR/applications/foamUser
```

It is recommended to edit the *Make/files* file so that the foamUser library is compiled locally into $FOAM_USER_LIBBIN as follows:

```
libfoamUser.C
LIB = $(FOAM_USER_LIBBIN)/libfoamUser
```

The new library should be added to the LIB_LIBS in *Make/options*

```
LIB_LIBS = \
    -l...  \
    -lnew
```

Finally, the library should be recompiled with:

```
wmake libso
```

3.3 Running applications

Each application is designed to be executed from a terminal command line, typically reading and writing a set of data files associated with a particular case. The data files for a case are typically stored in a directory named after the case as described in section 4.1, and here given the generic name <*case*>; the root directory path to the <*case*> directory is given the generic name <*root*>.

For any application, the form of the command line entry for any can be found by simply entering the application name at the command line, *e.g.* typing `blockMesh` returns information including

```
Usage: blockMesh <root> <case> [-blockTopology]
```

The arguments in angled brackets, < >, *i.e.* <root> and <case>, are the compulsory arguments — the blockMesh utility requires a case on which to run. The arguments in square brackets, [ ], are optional flags.

Like any UNIX/Linux executable, applications can be run as as a background process, *i.e.* one which does not have to be completed before the user can give the shell additional commands. If the user wished to run the blockMesh example as a background process and output the case progress to a *log* file, they could enter:

```
nohup nice -n 19 blockMesh <root> <case> > log &
```


3.4 Running applications in parallel

This section describes how to run OpenFOAM in parallel on distributed processors. The method of parallel computing used by OpenFOAM is known as domain decomposition, in which the geometry and associated fields are broken into pieces and allocated to separate processors for solution. The process of parallel computation involves: decomposition of mesh and fields; running the application in parallel; and, post-processing the decomposed case as described in the following sections. The parallel running uses the public domain openMPI implementation of the standard message passing interface (MPI). OpenFOAM can also be run using the MPICH implementation of MPI which is described in section A.1.

3.4.1 Decomposition of mesh and initial field data

The mesh and fields are decomposed using the decomposePar utility. The underlying aim is to break up the domain with minimal effort but in such a way to guarantee a fairly economic solution. The geometry and fields are broken up according to a set of parameters specified in a dictionary named *decomposeParDict* that must be located in the *system* directory of the case of interest. An example *decomposeParDict* dictionary can be copied from the interFoam/damBreak tutorial if the user requires one; the dictionary entries within it are reproduced below:

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25
26  numberOfSubdomains 4;
27
28  method          simple;
29
30  simpleCoeffs
31  {
32      n               (2 2 1);
33      delta           0.001;
34  }
35
36  hierarchicalCoeffs
37  {
38      n               (1 1 1);
39      delta           0.001;
40      order           xyz;
41  }
42
43  metisCoeffs
44  {
45      processorWeights
46      (
47          1
48          1
49          1
50          1
51      );
52  }
53
54  manualCoeffs
55  {
56      dataFile        "";
57  }
58
59  distributed     no;
60
61  roots
62  (
63  );
64
65
66  // ************************************************************************* //
```

The user has a choice of four methods of decomposition, specified by the `method` keyword as described below.

`simple` Simple geometric decomposition in which the domain is split into pieces by direction, *e.g.* 2 pieces in the x direction, 1 in y *etc.*

`hierarchical` Hierarchical geometric decomposition which is the same as `simple` except the user specifies the order in which the directional split is done, *e.g.* first in the y-direction, then the x-direction *etc.*

`metis` METIS decomposition which requires no geometric input from the user and attempts to minimise the number of processor boundaries. The user can specify a weighting for the decomposition between processors which can be useful on machines with differing performance between processors.

`manual` Manual decomposition, where the user directly specifies the allocation of each cell to a particular processor.

For each `method` there are a set of coefficients specified in a subdictionary of *decompositionDict*, named <*method*>*Coeffs* as shown in the dictionary listing. The full set of keyword entries in the *decomposeParDict* dictionary are explained in Table 3.4.

The decomposePar utility is executed in the normal manner by typing

```
decomposePar <root> <case>
```

On completion, a set of subdirectories will have been created, one for each processor, in the case directory. The directories are named *processor*N where $N = 0, 1, \ldots$ represents a processor number and contains a time directory, containing the decomposed field descriptions, and a *constant/polyMesh* directory containing the decomposed mesh description.

3.4.2 Running a decomposed case

A decomposed OpenFOAM case is run in parallel using the openMPI implementation of MPI (openMPI).

openMPI can be run on a local multiprocessor machine very simply but when running on machines across a network, a file must be created that contains the host names of the machines. The file can be given any name and located at any path. In the following description we shall refer to such a file by the generic name, including full path, <*machines*>.

The <*machines*> file contains the names of the machines listed one machine per line. The names must correspond to a fully resolved hostname in the */etc/hosts* file of the machine on which the openMPI is run. The list must contain the name of the machine running the openMPI. Where a machine node contains more than one processor, the node name may be followed by the entry `cpu=`n where n is the number of processors openMPI should run on that node.

For example, let us imagine a user wishes to run openMPI from machine `aaa` on the following machines: `aaa`; `bbb`, which has 2 processors; and `ccc`. The <*machines*> would contain:

Compulsory entries		
`numberOfSubdomains`	Total number of subdomains	N
`method`	Method of decomposition	`simple/` `hierarchical/` `metis/` `manual/`

`simpleCoeffs` **entries**		
`n`	Number of subdomains in x, y, z	(n_x n_y n_z)
`delta`	Cell skew factor	Typically, 10^{-3}

`hierarchicalCoeffs` **entries**		
`n`	Number of subdomains in x, y, z	(n_x n_y n_z)
`delta`	Cell skew factor	Typically, 10^{-3}
`order`	Order of decomposition	`xyz`/`xzy`/`yxz`...

`metisCoeffs` **entries**		
`processorWeights`	List of weighting factors for allocation of cells to processors; <`wt1`> is the weighting factor for processor 1, *etc.*; weights are normalised so can take any range of values.	(<`wt1`>...<`wtN`>)

`manualCoeffs` **entries**		
`dataFile`	Name of file containing data of allocation of cells to processors	"<`fileName`>"

Distributed data entries (optional) — see section 3.4.3		
`distributed`	Is the data distributed across several disks?	`yes/no`
`roots`	Root paths to case directories; <`rt1`> is the root path for node 1, *etc.*	(<`rt1`>...<`rtN`>)

Table 3.4: Keywords in *decompositionDict* dictionary.

```
aaa
bbb cpu=2
ccc
```

An application is run in parallel using mpirun.

```
mpirun --hostfile <machines> -np <nProcs>
    <foamExec> <root> <case> <otherArgs> -parallel > log
&
```

where: <`nProcs`> is the number of processors; <`foamExec`> is the executable, *e.g.* icoFoam; and, the output is redirected to a file named `log`. For example, if icoFoam is run on 4 nodes,

specified in a file named *machines*, on the cavity tutorial in the $FOAM_RUN/*tutorials/icoFoam* directory, then the following command should be executed:

```
mpirun --hostfile machines -np 4 icoFoam
    $FOAM_RUN/tutorials/icoFoam cavity -parallel > log &
```

3.4.3 Distributing data across several disks

Data files may need to be distributed if, for example, if only local disks are used in order to improve performance. In this case, the user may find that the root path to the case directory may differ between machines. The paths must then be specified in the *decomposeParDict* dictionary using `distributed` and `roots` keywords. The `distributed` entry should read

```
distributed  yes;
```

and the `roots` entry is a list of root paths, <root0>, <root1>, ..., for each node

```
roots
<nRoots>
(
   "<root0>"
   "<root1>"
   ...
);
```

where <nRoots> is the number of roots.

Each of the *processorN* directories should be placed in the case directory at each of the root paths specified in the *decomposeParDict* dictionary. The *system* directory and *files* within the *constant* directory must also be present in each case directory. Note: the files in the *constant* directory are needed, but the *polyMesh* directory is not.

3.4.4 Post-processing parallel processed cases

When post-processing cases that have been run in parallel the user has two options:

- reconstruction of the mesh and field data to recreate the complete domain and fields, which can be post-processed as normal;
- post-processing each segment of decomposed domain individually.

3.4.4.1 Reconstructing mesh and data

After a case has been run in parallel, it can be reconstructed for post-processing. The case is reconstructed by merging the sets of time directories from each *processorN* directory into a single set of time directories. The reconstructPar utility performs such a reconstruction by executing the command:

```
reconstructPar <root> <case>
```

When the data is distributed across several disks, it must be first copied to the local case directory for reconstruction.

3.4.4.2 Post-processing decomposed cases

The user may post-process decomposed cases using the paraFoam post-processor, described in section 7.1. The whole simulation can be post-processed by reconstructing the case or alternatively it is possible to post-process a segment of the decomposed domain individually by simply treating the individual processor directory as a case in its own right.

3.5 Standard solvers

The solvers with the OpenFOAM distribution are in the *$FOAM_APP/solvers* directory, reached quickly by typing app at the command line. This directory is further subdivided into several directories by category of continuum mechanics, *e.g.* incompressible flow, combustion and solid body stress analysis. Each solver is given a name that is reasonably descriptive, *e.g.* icoFoam solves incompressible, laminar flow, turbFoam solves incompressible, turbulent flow. The current list of solvers distributed with OpenFOAM is given in Table 3.5.

'Basic' CFD codes	
laplacianFoam	Solves a simple Laplace equation, e.g. for thermal diffusion in a solid
potentialFoam	Simple potential flow solver which can be used to generate starting fields for full Navier-Stokes codes
scalarTransportFoam	Solves a transport equation for a passive scalar
Incompressible flow	
boundaryFoam	Steady-state solver for 1D turbulent flow, typically to generate boundary layer conditions at an inlet, for use in a simulation.
icoDyMFoam	Transient solver for incompressible, laminar flow of Newtonian fluids with dynamic mesh
icoFoam	Transient solver for incompressible, laminar flow of Newtonian fluids
nonNewtonianIcoFoam	Transient solver for incompressible, laminar flow of non-Newtonian fluids
simpleFoam	Steady-state solver for incompressible, turbulent flow of non-Newtonian fluids
turbFoam	Transient solver for incompressible, turbulent flow
Compressible flow	
rhopSonicFoam	Pressure-density-based compressible flow solver
rhoSimpleFoam	Steady-state solver for turbulent flow of compressible fluids for ventilation and heat-transfer
rhoSonicFoam	Density-based compressible flow solver
rhoTurbFoam	Transient solver for compressible, turbulent flow
sonicFoam	Transient solver for trans-sonic/supersonic, laminar flow of a compressible gas.
sonicFoamAutoMotion	Transient solver for trans-sonic/supersonic, laminar flow of a compressible gas with mesh motion

Continued on next page

Continued from previous page

sonicLiquidFoam	Transient solver for trans-sonic/supersonic, laminar flow of a compressible liquid
sonicTurbFoam	Transient solver for trans-sonic/supersonic, turbulent flow of a compressible gas

Multiphase flow

bubbleFoam	Solver for a system of 2 incompressible fluid phases with one phase dispersed, e.g. gas bubbles in a liquid
cavitatingFoam	Solver for compressible liquid flow including cavitation modelled by a barotropic equations of state
interFoam	Solver for 2 incompressible fluids, which captures the interface using a VOF method
lesInterFoam	Solver for 2 incompressible fluids capturing the interface. Turbulence is modelled using a runtime selectable incompressible LES model
multiphaseInterFoam	Solver for an arbitrary number of incompressible immiscible fluids, capturing the multiple interfaces using a VOF method
rasInterFoam	Solver for 2 incompressible fluids capturing the interface. Turbulence is modelled using a runtime selectable incompressible RAS model
settlingFoam	Solver for 2 incompressible fluids for simulating the settling of the dispersed phase
twoLiquidMixingFoam	Solver for mixing 2 incompressible fluids
twoPhaseEulerFoam	Solver for a system of 2 incompressible fluid phases with one phase dispersed, e.g. gas bubbles in a liquid

Direct numerical simulation (DNS) and large eddy simulation (LES)

channelOodles	Incompressible LES solver for flow in a channel
coodles	(Currently no description)
dnsFoam	Direct numerical simulation solver for boxes of isotropic turbulence
oodles	Incompressible LES solver

Combustion

coldEngineFoam	Solver for cold-flow in internal combustion engines
dieselEngineFoam	Diesel engine spray and combustion code
dieselFoam	Diesel spray and combustion code
engineFoam	Solver for internal combustion engines
reactingFoam	Chemical reaction code
XiFoam	Compressible premixed/partially-premixed combustion solver with turbulence modelling
Xoodles	Compressible premixed/partially-premixed combustion solver with large-eddy simulation (LES) turbulence modelling

Heat transfer

Continued on next page

Continued from previous page

buoyantFoam	Transient Solver for buoyant, turbulent flow of compressible fluids for ventilation and heat-transfer
buoyantSimpleFoam	Steady-state solver for buoyant, turbulent flow of compressible fluids for ventilation and heat-transfer

Electromagnetics

electrostaticFoam	Solver for electrostatics
mhdFoam	Solver for magnetohydrodynamics (MHD): incompressible, laminar flow of a conducting fluid under the influence of a magnetic field

Stress analysis of solids

solidDisplacementFoam	Transient segregated finite-volume solver of linear-elastic, small-strain deformation of a solid body, with optional thermal diffusion and thermal stresses
solidEquilibriumDisplacementFoam	Steady-state segregated finite-volume solver of linear-elastic, small-strain deformation of a solid body

Finance

financialFoam	Solves the Black-Scholes equation to price commodities

Table 3.5: Standard library solvers.

3.6 Standard utilities

The utilities with the OpenFOAM distribution are in the *$FOAM_APP/utilities* directory, reached quickly by typing `util` at the command line. Again the names are reasonably descriptive, *e.g.* magU calculates the magnitude of velocity from velocity field data, ideasToFoam converts mesh data from the format written by I-DEAS to the OpenFOAM format. The current list of utilities distributed with OpenFOAM is given in Table 3.6.

Pre-processing — see chapter 5

boxTurb	Makes a box of turbulence which conforms to a given energy spectrum and is divergence free
engineSwirl	Generates a swirling flow for engine calulations
FoamX	(Description not found)
mapFields	Maps volume fields from one mesh to another, reading and interpolating all fields present in the time directory of both cases. Parallel and non-parallel cases are handled without the need to reconstruct them first
setFields	Selects a cell set through a dictionary

Mesh generation — see section 6.3

Continued on next page

Continued from previous page

blockMesh	Mesh generator: blockOffsets_(createBlockOffsets()), mergeList_(createMergeList()), points_(createPoints()), cells_(createCells()), patches_(createPatches())
extrudeMesh	Extrude mesh from existing patch or from patch read from file

Mesh conversion — see section 6.4

ansysToFoam	Converts an ANSYS input mesh file, exported from I-DEAS, to OpenFOAM format
ccm26ToFoam	CCM mesh converter using CCM version 2.6 library
cfxToFoam	Converts a CFX mesh to OpenFOAM format
fluentMeshToFoam	Converts a Fluent mesh to OpenFOAM format including multiple region and region boundary handling
foamMeshToFluent	Writes out the OpenFOAM mesh in Fluent mesh format
gambitToFoam	Converts a GAMBIT mesh to OpenFOAM format
gmshToFoam	Reads .msh file as written by Gmsh
ideasUnvToFoam	Converts meshes from I-DEAS `.unv` format to OpenFOAM format
kivaToFoam	Converts a KIVA3v grid to OpenFOAM format
mshToFoam	Reads .msh format generated by the Adventure system
netgenNeutralToFoam	read Neutral file format as written by Netgen4.4
plot3dToFoam	Plot3d mesh (ascii format) converter
polyDualMesh	(Currently no description)
sammToFoam	Converts a STAR-CD SAMM mesh to OpenFOAM format
starToFoam	Converts a STAR-CD PROSTAR mesh into OpenFOAM format
tetgenToFoam	Reads .ele and .node and .face files as written by tetgen
writeMeshObj	For mesh debugging: writes mesh as three separate OBJ files which can be viewed with e.g. javaview

Mesh manipulation

attachMesh	Attach topologically detached mesh using prescribed mesh modifiers
autoPatch	Divides external faces into patches based on (user supplied) feature angle
cellSet	Selects a cell set through a dictionary
checkMesh	Checks validity of a mesh
couplePatches	Utility to reorder cyclic and processor patches
createPatch	Utility to create patches out of selected boundary faces. Faces come either from existing patches or from a faceSet
deformedGeom	Deforms a polyMesh using a displacement field `U` and a scaling factor supplied as an argument
faceSet	Selects a face set through a dictionary
flattenMesh	Flatten the front and back planes of a 2D cartesian mesh
insideCells	Pick up cells with cell centre 'inside' of surface. Requires surface to be closed and singly connected

Continued on next page

Continued from previous page

mergeMeshes	Merge two meshes
mirrorMesh	(Currently no description)
moveDynamicMesh	Mesh motion and topological mesh changes utility
moveEngineMesh	Solver for moving meshes for engine calculations.
moveMesh	Solver for moving meshes
objToVTK	Read *obj* line (not surface!) file and convert into vtk
patchTool	(Description not found)
pointSet	Selects a point set through a dictionary
refineMesh	Utility to refine cells in multiple directions. Either supply -all option to refine all cells (3D refinement for 3D cases; 2D for 2D cases) or reads a *refineMeshDict* with - cellSet to refine - directions to refine
renumberMesh	Renumbers the cell list in order to reduce the bandwidth, reading and renumbering all fields from all the time directories
rotateMesh	Rotates the mesh and fields from the direcion n1 to the direction n2
splitMesh	Splits mesh by making internal faces external. Uses attachDetach
splitMeshRegions	Splits mesh into multiple regions and writes them to consecutive time directories. Each region is defined as a domain whose cells can all be reached by cell-face-cell walking. Uses meshWave. Could work in parallel but never tested
stitchMesh	'Stitches' a mesh
subsetMesh	Selects a section of mesh based on a cellSet
tetDecomposition	Takes a mesh and decomposes it into tetrahedra using a face-cell centre decomposition
transformPoints	Transforms the mesh points in the polyMesh directory according to the options:
zipUpMesh	Reads in a mesh with hanging vertices and zips up the cells to guarantee that all polyhedral cells of valid shape are closed

Post-processing graphics — see chapter 7

ensight76FoamExec	Module for EnSight 7.6 to read OpenFOAM data directly without translation
paraFoam	(Description not found)

Post-processing data converters — see chapter 7

foamDataToFluent	Translates OpenFOAM data to Fluent format
foamToEnsight	Translates OpenFOAM data to EnSight format
foamToFieldview9	Write out the OpenFOAM mesh in Version 3.0 Fieldview-UNS format (binary). See Fieldview Release 9 Reference Manual - Appendix D (Unstructured Data Format) Borrows various from uns/write_binary_uns.c from FieldView dist
foamToGMV	Translates foam output to GMV readable files. A free post-processor with available binaries from http://www-xdiv.lanl.gov/XCM/gmv/

Continued on next page

Continued from previous page

foamToVTK	legacy VTK file format writer. - handles volScalar, volVector, pointScalar, pointVector, surfaceScalar fields. - mesh topo changes. - both ascii and binary. - single time step writing. - write subset only. - automatic decomposition of cells; polygons on boundary undecomposed since handled by vtk
smapToFoam	Translates a STAR-CD SMAP data file into OpenFOAM field format

Post-processing velocity fields

Co	Configurable graph drawing program
divU	Calculates and writes the divergence of velocity field `U` at each time
enstrophy	Calculates and writes the enstrophy of velocity field `U` at each time
flowType	Calculates and writes the flowType of velocity field `U` at each time
Lambda2	Calculates and writes the second largest eigenvalue of the sum of the square of the symmetrical and anti-symmetrical parts of the velocity gradient tensor, for each time
Mach	Calculates and writes the local Mach number from the velocity field `U` at each time
magGradU	Calculates and writes the scalar magnitude of velocity field `U` at each time
magU	Calculates and writes the scalar magnitude of the gradient of the velocity field `U` for each time
Pe	Calculates and writes the Pe number as a surfaceScalarField obtained from field phi for each time
Q	Calculates and writes the second invariant of the velocity gradient tensor for each time
streamFunction	Calculates and writes the stream function of velocity field `U` at each time
Ucomponents	Writes the three scalar fields, `Ux`, `Uy` and `Uz`, for each component of the velocity field `U` for each time
uprime	Calculates and writes the scalar field of `uprime` ($\sqrt{\frac{2}{3}k}$) at each time
vorticity	Calculates and writes the vorticity of velocity field `U` at each time

Post-processing stress fields

R	Calculates and writes the Reynolds stress `R` for the current time step
Rcomponents	Calculates and writes the scalar fields of the six components of the Reynolds stress `R` for each time
stressComponents	Calculates and writes the scalar fields of the six components of the stress tensor `sigma` for each time

Continued on next page

Continued from previous page

Post-processing at walls

checkYPlus	Calculates and reports `yPlus` for all wall patches, for each time in a database
wallGradU	Calculates and writes the gradient of `U` at the wall
wallHeatFlux	Calculates and writes the heat flux for all patches as the boundary field of a volScalarField and also prints the integrated flux for all wall patches
wallShearStress	Calculates and writes the wall shear stress for the current time step
yPlusLES	Calculates the `yPlus` of the near-wall cells for an LES

Post-processing at patches

patchAverage	Calculate average of fields over all patches
patchIntegrate	Integrates fields over all patches

Miscellaneous post-processing

engineCompRatio	Calculate the geometric compression ratio. Note that if you have valves and/or extra volumes it will not work, since it calculates the volume at BDC and TCD
postChannel	Post-processes data from channel flow calculations
ptot	For each time: calculate the total pressure
sample	Sample field data with a choice of interpolation schemes, sampling options and write formats
sampleSurface	Surface sampling. Runs in parallel (but does not merge points)
wdot	Calculates and writes wdot for each time
writeCellCentres	Write the three components of the cell centres as volScalarFields so they can be used in postprocessing in thresholding

Parallel processing — see section 3.4

decomposePar	Automatically decomposes a mesh and fields of a case for parallel execution of OpenFOAM
reconstructPar	Reconstructs a mesh and fields of a case that is decomposed for parallel execution of OpenFOAM
reconstructParMesh	Reconstructs a mesh using geometric information only. Writes point/face/cell procAddressing so afterwards reconstructPar can be used to reconstruct fields

Thermophysical-related utilities

adiabaticFlameT	Calculates the adiabatic flame temperature for a given fuel over a range of unburnt temperatures and equivalence ratios
chemkinToFoam	Converts CHEMKIN 3 thermodynamics and reaction data files into OpenFOAM format
equilibriumCO	Calculates the equilibrium level of carbon monoxide

Continued on next page

Continued from previous page

equilibriumFlameT	Calculates the equilibrium flame temperature for a given fuel and pressure for a range of unburnt gas temperatures and equivalence ratios; the effects of dissociation on O_2, H_2O and CO_2 are included
mixtureAdiabaticFlameT	Calculates the adiabatic flame temperature for a given mixture at a given temperature

Error estimation

estimateScalarError	Estimates the error in the solution for a scalar transport equation in the standard form
icoErrorEstimate	Estimates error for the incompressible laminar CFD application icoFoam
icoMomentError	Estimates error for the incompressible laminar CFD application icoFoam
momentScalarError	Estimates the error in the solution for a scalar transport equation in the standard form

Miscellaneous utilities

foamDebugSwitches	Write out all library debug switches
foamInfoExec	Interrogates a case and prints information to screen

Table 3.6: Standard library utilities.

3.7 Standard libraries

The libraries with the OpenFOAM distribution are in the *$FOAM_LIB/$WM_OPTIONS* directory, reached quickly by typing `lib` at the command line. Again, the names are prefixed by `lib` and reasonably descriptive, *e.g.* incompressibleTransportModels contains the library of incompressible transport models. For ease of presentation, the libraries are separated into two types:

General libraries those that provide general classes and associated functions listed in Table 3.7;

Model libraries those that specify models used in computational continuum mechanics, listed in Table 3.8, Table 3.9 and Table 3.10.

Library of basic OpenFOAM tools — OpenFOAM

algorithms	Algorithms
containers	Container classes
db	Database classes
dimensionSet	dimensionSet class
dimensionedTypes	dimensioned<Type> class and derivatives
fields	Field classes

Continued on next page

Continued from previous page

finiteVolume	Finite volume discretisation classes
global	Global settings
interpolations	Interpolation schemes
matrices	Matrix classes
meshes	Mesh classes
primitives	Primitive classes

Library of CFD tools — cfdTools

adjustPhi	Adjusts boundary fluxes
bound	Bounds scalar fields
compressible	Compressible flow CFD tools
incompressible	Incompressible flow CFD tools
wallDist	Calculations relating to wall boundaries

Post-processing libraries

incompressiblePostProcessing	Tools for post-processing incompressible flow data
sampling	Tools for sampling field data at prescribed locations in a domain

Solution and mesh manipulation libraries

cellDecompFiniteElement	Cell decomposed finite element scheme
dynamicMesh	For solving systems with moving meshes
edgeMesh	For handling edge-based mesh descriptions
errorEstimation	Error estimation tools
faceDecompFiniteElement	Face decomposed finite element scheme
ODE	Solvers for ordinary differential equations
shapeMeshTools	Tools for handling a mesh whose cells are defined by a set of standard shapes
meshTools	Tools for handling a OpenFOAM mesh
triSurface	For handling standard triangulated surface-based mesh descriptions

Lagrangian particle tracking libraries

dieselSpray	Diesel spray tracking solution scheme
lagrangian	Basic Lagrangian, or particle-tracking, solution scheme

Public domain libraries

mico-2.3.13	Implementation of the Common Object Request Broker Architecture (CORBA)
mpich-1.2.4	Portable message-passing interface for parallel processing
openmpi-1.2.3	Portable message-passing interface for parallel processing
zlib-1.2.1	General purpose data compression

Miscellaneous libraries

engine	Tools for engine calculations
Gstream	2D graphics stream

Continued on next page

Continued from previous page

randomProcesses	Tools for analysing and generating random processes

Table 3.7: Shared object libraries for general use.

Basic thermophysical models — basicThermophysicalModels

hThermo	General thermophysical model calculation based on enthalpy h
pureMixture	General thermophysical model calculation for passive gas mixtures

Combustion models — combustionThermophysicalModels

hMixtureThermo	Calculates enthalpy for combustion mixture
hhuMixtureThermo	Calculates enthalpy for unburnt gas and combustion mixture
homogeneousMixture	Combustion mixture based on normalised fuel mass fraction b
inhomogeneousMixture	Combustion mixture based on b and total fuel mass fraction f_t
veryInhomogeneousMixture	Combustion mixture based on b, f_t and unburnt fuel mass fraction f_u
dieselMixture	Combustion mixture based on f_t and f_u
multiComponentMixture	Combustion mixture based on multiple components [**]
chemkinMixture	Combustion mixture using CHEMKIN thermodynamics and reaction schemes database files

Laminar flame speed models — laminarFlameSpeedModels

constLaminarFlameSpeed	Constant laminar flame speed
guldersLaminarFlameSpeed	Gülder's laminar flame speed model

Thermophysical properties of liquids — liquids

nHeptane	Thermophysical properties of nHeptane
nOctane	Thermophysical properties of nOctane
nDecane	Thermophysical properties of nDecane
nDodecane	Thermophysical properties of nDodecane
isoOctane	Thermophysical properties of isoOctane
diMethylEther	Thermophysical properties of diMethylEther
diEthylEther	Thermophysical properties of diEthylEther
water	Thermophysical properties of water

Thermophysical properties of gaseous species — specie

Continued on next page

Continued from previous page

perfectGas	Perfect gas equation of state
hConstThermo	Constant specific heat c_p model with evaluation of enthalpy h and entropy s
janafThermo	c_p evaluated by a function with coefficients from JANAF thermodynamic tables, from which h, s are evaluated
specieThermo	Thermophysical properties of species, derived from c_p, h and/or s
constTransport	Constant transport properties
sutherlandTransport	Sutherland's formula for temperature-dependent transport properties

Functions/tables of thermophysical properties — thermophysicalFunctions

NSRDSfunctions	National Standard Reference Data System (NSRDS) - American Institute of Chemical Engineers (AICHE) data compilation tables
APIfunctions	American Petroleum Institute (API) function for vapour mass diffusivity

Probability density functions — pdf

RosinRammler	Rosin-Rammler distribution
normal	Normal distribution
uniform	Uniform distribution
exponential	Exponential distribution
general	General distribution

Chemistry model — chemistryModel

chemistryModel	Chemical reaction model
chemistrySolver	Chemical reaction solver

Table 3.8: Libraries of thermophysical models.

Turbulence models for incompressible fluids — incompressibleTurbulenceModels

laminar	Dummy turbulence model for laminar flow
kEpsilon	Standard $k-\varepsilon$ model with wall functions
RNGkEpsilon	RNG $k-\varepsilon$ model with wall functions
NonlinearKEShih	Non-linear Shih $k-\varepsilon$ model with wall functions
LienCubicKE	Lien cubic $k-\varepsilon$ model with wall functions
QZeta	$q-\zeta$ model
LaunderSharmaKE	Launder-Sharma low-Re $k-\varepsilon$ model
LamBremhorstKE	Lam-Bremhorst low-Re $k-\varepsilon$ model
LienCubicKELowRE	Lien cubic low-Re $k-\varepsilon$ model
LienLeschzinerLowRE	Lien-Leschziner low-Re $k-\varepsilon$ model
LRR	Launder-Reece-Rodi RSTM with wall functions
LaunderGibsonRSTM	Launder-Gibson RSTM with wall-reflection terms and wall functions

Continued on next page

Continued from previous page

SpalartAllmaras	Spalart-Allmaras 1-eqn mixing-length model for external flows

Turbulence models for compressible fluids — compressibleTurbulenceModels

laminar	Dummy turbulence model for laminar flow
kEpsilon	Standard $k-\varepsilon$ model with wall functions
RNGkEpsilon	RNG $k-\varepsilon$ model with wall functions
LaunderSharmaKE	Launder-Sharma low-Re $k-\varepsilon$ model
LRR	Launder-Reece-Rodi RSTM with wall functions
LaunderGibsonRSTM	Launder-Gibson RSTM with wall-reflection terms and wall functions

Large-eddy simulation (LES) filters — LESfilters

laplaceFilter	Laplace filters
simpleFilter	Simple filter
anisotropicFilter	Anisotropic filter

Large-eddy simulation deltas — LESdeltas

PrandtlDelta	Prandtl delta
cubeRootVolDelta	Cube root of cell volume delta
smoothDelta	Smoothing of delta

Incompressible LES models — incompressibleLESmodels

Smagorinsky	Smagorinsky model
Smagorinsky2	Smagorinsky model with 3-D filter
dynSmagorinsky	Dynamic Smagorinsky
scaleSimilarity	Scale similarity model
mixedSmagorinsky	Mixed Smagorinsky/scale similarity model
dynMixedSmagorinsky	Dynamic mixed Smagorinsky/scale similarity model
oneEqEddy	k-equation eddy-viscosity model
dynOneEqEddy	Dynamic k-equation eddy-viscosity model
locDynOneEqEddy	Localised dynamic k-equation eddy-viscosity model
spectEddyVisc	Spectral eddy viscosity model
LRDDiffStress	LRR differential stress model
DeardorffDiffStress	Deardorff differential stress model
SpalartAllmaras	Spalart-Allmaras model

Compressible LES models — compressibleLESmodels

Smagorinsky	Smagorinsky model
oneEqEddy	k-equation eddy-viscosity model
dynOneEqEddy	Dynamic k-equation eddy-viscosity model
lowReOneEqEddy	Low-Re k-equation eddy-viscosity model
DeardorffDiffStress	Deardorff differential stress model

Table 3.9: Libraries of turbulence and LES models.

Transport models for incompressible fluids — incompressibleTransportModels

Newtonian	Linear viscous fluid model
CrossPowerLaw	Cross Power law nonlinear viscous model
BirdCarreau	Bird-Carreau nonlinear viscous model

Table 3.10: Shared object libraries of transport models.

Chapter 4

OpenFOAM cases

This chapter deals with the file structure and organisation of OpenFOAM cases. Normally, a user would assign a name to a case, *e.g.* the tutorial case of flow in a cavity is simply named cavity. This name becomes the name of a directory in which all the case files and subdirectories are stored. The case directories themselves can be located anywhere but we recommend they are within a *run* subdirectory of the user's project directory, *i.e.* *$HOME/OpenFOAM/${USER}-1.4.1* as described at the beginning of chapter 2. One advantage of this is that the $FOAM_RUN environment variable is set to *$HOME/OpenFOAM/${USER}-1.4.1/run* by default; the user can quickly move to that directory by executing a preset alias, `run`, at the command line.

The tutorial cases that accompany the OpenFOAM distribution provide useful examples of the case directory structures. The tutorials are located in the *$FOAM_TUTORIALS* directory, reached quickly by executing the `tut` alias at the command line. Users can view tutorial examples at their leisure while reading this chapter.

4.1 File structure of OpenFOAM cases

The basic directory structure for a OpenFOAM case, that contains the minimum set of files required to run an application, is shown in Figure 4.1 and described as follows:

A *constant* directory that contains a full description of the case mesh in a subdirectory *polyMesh* and files specifying physical properties for the application concerned, *e.g.* *transportProperties*.

A *system* directory for setting parameters associated with the solution procedure itself. It contains *at least* the following 3 files: *controlDict* where run control parameters are set including start/end time, time step and parameters for data output; *fvSchemes* where discretisation schemes used in the solution may be selected at run-time; and, *fvSolution* where the equation solvers, tolerances and other algorithm controls are set for the run.

The 'time' directories containing individual files of data for particular fields. The data can be: either, initial values and boundary conditions that the user must specify to define the problem; or, results written to file by OpenFOAM. Note that the OpenFOAM fields must always be initialised, even when the solution does not strictly require it, as in steady-state problems. The name of each time directory is based on the simulated

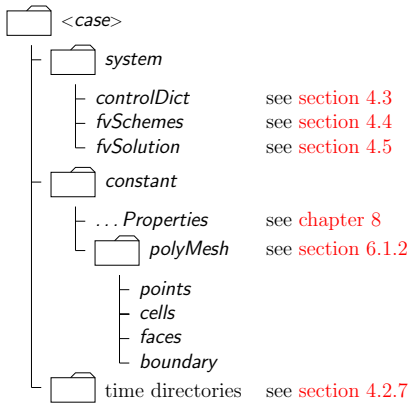


Figure 4.1: Case directory structure

time at which the data is written and is described fully in section 4.3. It is sufficient to say now that since we usually start our simulations at time $t = 0$, the initial conditions are usually stored in a directory named *0* or *0.000000e+00*, depending on the name format specified. For example, in the cavity tutorial, the velocity field **U** and pressure field p are initialised from files *0/U* and *0/p* respectively.

4.2 Basic input/output file format

OpenFOAM needs to read a range of data structures such as strings, scalars, vectors, tensors, lists and fields. The input/output (I/O) format of files is designed to be extremely flexible to enable the user to modify the I/O in OpenFOAM applications as easily as possible. The I/O follows a simple set of rules that make the files extremely easy to understand, in contrast to many software packages whose file format may not only be difficult to understand intuitively but also not be published anywhere. The description of the OpenFOAM file format is described in the following sections.

4.2.1 General syntax rules

The format follows the following some general principles of C++ source code.

- Files have free form, with no particular meaning assigned to any column and no need to indicate continuation across lines.
- Lines have no particular meaning except to a `//` comment delimiter which makes OpenFOAM ignore any text that follows it until the end of line.
- A comment over multiple lines is done by enclosing the text between `/*` and `*/` delimiters.

4.2.2 Dictionaries

OpenFOAM uses *dictionaries* as the most common means of specifying data. A dictionary is an entity that contains as set data entries that can be retrieved by the I/O by means of *keywords*. The keyword entries follow the general format

```
<keyword>  <dataEntry1> ... <dataEntryN>;
```

Most entries are single data entries of the form:

```
<keyword>  <dataEntry>;
```

Most OpenFOAM data files are themselves dictionaries containing a set of keyword entries. Dictionaries provide the means for organising entries into logical categories and can be specified hierarchically so that any dictionary can itself contain one or more dictionary entries. The format for a dictionary is to specify the dictionary name followed the the entries enclosed in curly braces {} as follows

```
<dictionaryName>
{
    ... keyword entries ...
}
```

4.2.3 The data file header

All data files that are read and written by OpenFOAM begin with a dictionary named `FoamFile` containing a standard set of keyword entries, listed in Table 4.1. The table provides brief descriptions of each entry, which is probably sufficient for most entries with the notable exception of `class`. The `class` entry is the name of the C++ class in the OpenFOAM library that will be constructed from the data in the file. Without knowledge of the underlying code which calls the file to be read, and knowledge of the OpenFOAM classes,

Keyword	Description	Entry
`version`	I/O format version	`1.4.1`
`format`	Data format	`ascii` / `binary`
`root`	Root path to case directory, in `"..."`	*e.g.* `"~/OpenFOAM/chris1.4.1/run"`
`case`	Case directory name, in `"..."`	*e.g.* `"cavity"`
`instance`	Subdirectory within case, in `"..."`	`"<timeDirectory>"` / `"system"` / `"constant"`
`local`	Any subdirectory within `instance`, in `"..."` (optional entry)	*e.g.* `"polyMesh"`
`class`	OpenFOAM class constructed from the data file concerned	typically `dictionary` or a field, *e.g.* `volVectorField`
`object`	Filename	*e.g.* `controlDict`

Table 4.1: Header keywords entries for data files.

the user will probably be unable to surmise the `class` entry correctly. However, most data files with simple keyword entries are read into an internal dictionary class and therefore the `class` entry is `dictionary` in those cases.

The following example shows the use of keywords to provide data for a case using the types of entry described so far. The extract, from an *fvSolution* dictionary file, contains 2 dictionaries, *solvers* and *PISO*. The *solvers* dictionary contains multiple data entries for solver and tolerances for each of the pressure and velocity equations, represented by the `p` and `U` keywords respectively; the *PISO* dictionary contains algorithm controls.

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  solvers
26  {
27      p PCG
28      {
29          preconditioner   DIC;
30          tolerance        1e-06;
31          relTol           0;
32      };
33
34      U PBiCG
35      {
36          preconditioner   DILU;
37          tolerance        1e-05;
38          relTol           0;
39      };
40  }
41
42  PISO
43  {
44      nCorrectors     2;
45      nNonOrthogonalCorrectors 0;
46      pRefCell        0;
47      pRefValue       0;
48  }
49
50
51  // ************************************************************************* //
```

4.2.4 Lists

OpenFOAM applications contain lists, *e.g.* a list of vertex coordinates for a mesh description. Lists are commonly found in I/O and have a format of their own in which the entries are contained within round braces `( )`. There is also a choice of format preceeding the round braces:

simple the keyword is followed immediately by round braces

```
<listName>
(
    ... entries ...
);
```

numbered the keyword is followed by the number of elements `<n>` in the list

```
<listName>
<n>
(
    ... entries ...
);
```

token identifier the keyword is followed by a class name identifier Label<Type> where <Type> states what the list contains, *e.g.* for a list of scalar elements is

```
<listName>
List<scalar>
<n>        // optional
(
    ... entries ...
);
```

Note that <scalar> in List<scalar> is not a generic name but the actual text that should be entered.

The simple format is a convenient way of writing a list. The other formats allow the code to read the data faster since the size of the list can be allocated to memory in advance of reading the data. The simple format is therefore preferred for short lists, where read time is minimal, and the other formats are preferred for long lists.

4.2.5 Scalars, vectors and tensors

A scalar is a single number represented as such in a data file. A vector is a VectorSpace of rank 1 and dimension 3, and since the number of elements is always fixed to 3, the simple List format is used. Therefore a vector $(1.0, 1.1, 1.2)$ is written:

```
(1.0 1.1 1.2)
```

In OpenFOAM, a tensor is a VectorSpace of rank 2 and dimension 3 and therefore the data entries are always fixed to 9 real numbers. Therefore the identity tensor, described in section 1.3.7 of the Programmer's Guide, can be written:

```
(
    1 0 0
    0 1 0
    0 0 1
)
```

This example demonstrates the way in which OpenFOAM ignores the line return is so that the entry can be written over multiple lines. It is treated no differently to listing the numbers on a single line:

```
( 1 0 0 0 1 0 0 0 1 )
```

4.2.6 Dimensioned types

Physical properties are typically specified with their associated dimensions, to be created by the dimensioned<Type> class in OpenFOAM as described in section 1.5 of the Programmer's Guide. These entries have the format that the following example of a dimensionedScalar demonstrates:


```
nu              nu  [0 2 -1 0 0 0 0]  1;
```

The first nu is the keyword; the second nu is the word name stored in class word, usually chosen to be the same as the keyword; the next entry is the dimensionSet and the final entry is the scalar value.

4.2.7 Fields

Much of the I/O data in OpenFOAM are tensor fields, *e.g.* velocity, pressure data, that are read from and written into the time directories. More precisely, the fields are objects of the geometricField<Type> class, as described in section 2.3.2 of the Programmer's Guide. OpenFOAM writes geometricField<Type> data using keyword entries as described in Table 4.2.

Keyword	Description	Example
dimensions	Dimensions of field	[1 1 -2 0 0 0 0]
referenceLevel	Reference level for the internal field	(0 0 0)
internalField	Value of internal field	uniform (1 0 0)
boundaryField	Boundary field	see file listing in section 4.2.7

Table 4.2: Main keywords used in field dictionaries.

The data begins with an entry for its dimensions. It is followed by a referenceLevel value; the field variables are stored as values relative to the reference level entry, which is usually set to zero but can be set to other values to improve solution accuracy. Following that, is the internalField, described in one of the following ways.

Uniform field a single value is assigned to all elements within the field, taking the form:

```
internalField uniform <entry>;
```

Nonuniform field each field element is assigned a unique value from a list, taking the following form where the token identifier form of list is recommended:

```
internalField nonuniform <List>;
```

The boundaryField is a dictionary containing a set of entries whose names correspond to each of the names of the boundary patches listed in the *boundary* file in the *polyMesh* directory. Each patch entry is itself a dictionary containing a list of keyword entries. The compulsory entry, type, describes the patch field condition specified for the field. The remaining entries correspond to the type of patch field condition selected and can typically include field data specifying initial conditions on patch faces. The patch field conditions available in OpenFOAM are listed in Table 6.3 and Table 6.4 with a description and the data that must be specified with it. Example field dictionary entries for velocity U are shown below:

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
    }
}


// ************************************************************************* //
```

4.3 Time and data input/output control

The OpenFOAM solvers begin all runs by setting up a database. The database controls I/O and, since output of data is usually requested at intervals of time during the run, time is an inextricable part of the database. The *controlDict* dictionary sets input parameters *essential* for the creation of the database. The keyword entries in *controlDict* are listed in Table 4.3. Only the time control and `writeInterval` entries are truly compulsory, with the database taking default values indicated by † in Table 4.3 for any of the optional entries that are omitted.

Keywords required by FoamX	
`applicationClass`	Name of application class which is used when opening this case in FoamX, *e.g.* icoFoam.

Time control	
`startFrom`	Controls the start time of the simulation.
- `firstTime`	Earliest time step from the set of time directories.
- `startTime`	Time specified by the `startTime` keyword entry.
- `latestTime`	Most recent time step from the set of time directories.
`startTime`	Start time for the simulation with `startFrom startTime;`
`stopAt`	Controls the end time of the simulation.
- `endTime`	Time specified by the `endTime` keyword entry.
- `writeNow`	Stops simulation on completion of current time step and writes data.

Continued on next page

Continued from previous page

- `noWriteNow`	Stops simulation on completion of current time step and does not write out data.
- `nextWrite`	Stops simulation on completion of next scheduled write time, specified by `writeControl`.
`endTime`	End time for the simulation when `stopAt endTime;` is specified.
`deltaT`	Time step of the simulation.

Data writing	
`writeControl`	Controls the timing of write output to file.
- `timeStep`†	Writes data every `writeInterval` time steps.
- `runTime`	Writes data every `writeInterval` seconds of simulated time.
- `adjustableRunTime`	Writes data every `writeInterval` seconds of simulated time, adjusting the time steps to coincide with the `writeInterval` if necessary — used in cases with automatic time step adjustment.
- `cpuTime`	Writes data every `writeInterval` seconds of CPU time.
- `clockTime`	Writes data out every `writeInterval` seconds of real time.
`writeInterval`	Scalar used in conjunction with `writeControl` described above.
`purgeWrite`	Integer representing a limit on the number of time directories that are stored by overwriting time directories on a cyclic basis. Example of $t_0 = 5$s, $\Delta t = 1$s and `purgeWrite 2;`: data written into 2 directories, *6* and *7*, before returning to write the data at 8 s in *6*, data at 9 s into *7*, *etc.* *To disable the time directory limit, specify* `purgeWrite 0;`† For steady-state solutions, results from previous iterations can be continuously overwritten by specifying `purgeWrite 1;`
`writeFormat`	Specifies the format of the data files.
- `ascii`†	ASCII format, written to `writePrecision` significant figures.
- `binary`	Binary format.
`writePrecision`	Integer used in conjunction with `writeFormat` described above, 6† by default
`writeCompression`	Specifies the compression of the data files.
- `uncompressed`	No compression.†
- `compressed`	gzip compression.
`timeFormat`	Choice of format of the naming of the time directories.
- `fixed`	$\pm m.dddddd$ where the number of ds is set by `timePrecision`.
- `scientific`	$\pm m.dddddde \pm xx$ where the number of ds is set by `timePrecision`.
- `general`†	Specifies `scientific` format if the exponent is less than -4 or greater than or equal to that specified by `timePrecision`.

Continued on next page

Continued from previous page

`timePrecision`	Integer used in conjunction with `timeFormat` described above, 6† by default
`graphFormat`	Format for graph data written by an application.
- `raw`†	Raw ASCII format in columns.
- `gnuplot`	Data in gnuplot format.
- `xmgr`	Data in Grace/xmgr format.
- `jplot`	Data in jPlot format.
Data reading	
`runTimeModifiable`	`yes`†/`no` switch for whether dictionaries, *e.g.* *controlDict*, are re-read by OpenFOAM at the beginning of each time step.

† denotes default entry if associated keyword is omitted.

Table 4.3: Keyword entries in the *controlDict* dictionary.

Example entries from a *controlDict* dictionary are given below:

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  application icoFoam;
26
27  startFrom       startTime;
28
29  startTime       0;
30
31  stopAt          endTime;
32
33  endTime         0.5;
34
35  deltaT          0.005;
36
37  writeControl    timeStep;
38
39  writeInterval   20;
40
41  purgeWrite      0;
42
43  writeFormat     ascii;
44
45  writePrecision  6;
46
47  writeCompression uncompressed;
48
49  timeFormat      general;
50
51  timePrecision   6;
52
53  runTimeModifiable yes;
54
55
56  // ************************************************************************* //
```

4.4 Numerical schemes

The *fvSchemes* dictionary in the *system* directory sets the numerical schemes for terms, such as derivatives in equations, that appear in applications being run. This section describes

how to specify the schemes in the *fvSchemes* dictionary; a description of the numerics of the schemes is in section 2.4 of the Programmer's Guide.

The terms that must typically be assigned a numerical scheme in *fvSchemes* range from derivatives, *e.g.* gradient ∇, and interpolations of values from one set of points to another. The aim in OpenFOAM is to offer an unrestricted choice to the user. For example, while linear interpolation is effective in many cases, OpenFOAM offers complete freedom to choose from a wide selection of interpolation schemes for all interpolation terms.

The derivative terms further exemplify this freedom of choice. The user first has a choice of discretisation practice where standard Gaussian finite volume integration is the common choice. Gaussian integration is based on summing values on cell faces, which must be interpolated from cell centres. The user again has a completely free choice of interpolation scheme, with certain schemes being specifically designed for particular derivative terms, especially the convection divergence $\nabla \bullet$ terms.

The set of terms, for which numerical schemes must be specified, are subdivided within the *fvSchemes* dictionary into the categories listed in Table 4.4. Each keyword in Table 4.4 is the name of a subdictionary which contains terms of a particular type, *e.g.* `gradSchemes` contains all the gradient derivative terms such as `grad(p)` (which represents ∇p). Further examples can be seen in the extract from an *fvSchemes* dictionary below:

Keyword	Category of mathematical terms
`interpolationSchemes`	Point-to-point interpolations of values
`snGradSchemes`	Component of gradient normal to a cell face
`gradSchemes`	Gradient ∇
`divSchemes`	Divergence $\nabla \bullet$
`laplacianSchemes`	Laplacian ∇^2
`timeScheme`	First and second time derivatives $\partial/\partial t, \partial^2/\partial^2 t$
`fluxRequired`	Fields which require the generation of a flux

Table 4.4: Main keywords used in *fvSchemes*.

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  ddtSchemes
26  {
27      default         Euler;
28  }
29
30  gradSchemes
31  {
32      default         Gauss linear;
33      grad(p)         Gauss linear;
34  }
35
36  divSchemes
37  {
38      default         none;
39      div(phi,U)      Gauss linear;
40  }
41
42  laplacianSchemes
43  {
44      default         none;
45      laplacian(nu,U) Gauss linear corrected;
46      laplacian((1|A(U)),p) Gauss linear corrected;
47  }
48
49  interpolationSchemes
50  {
```

```
51      default         linear;
52      interpolate(HbyA) linear;
53  }
54
55  snGradSchemes
56  {
57      default         corrected;
58  }
59
60  fluxRequired
61  {
62      default         no;
63      p;
64  }
65
66
67  // ************************************************************************* //
```

The example shows that the *fvSchemes* dictionary contains the following:

- 6 . . . *Schemes* subdictionaries containing keyword entries for each term specified within including: a `default` entry; other entries whose names correspond to a word identifier for the particular term specified, *e.g.* `grad(p)` for ∇p
- a *fluxRequired* subdictionary containing fields for which the flux is generated in the application, *e.g.* `p` in the example.

If a `default` scheme is specified in a particular . . . *Schemes* subdictionary, it is assigned to all of the terms to which the subdictionary refers, *e.g.* specifying a `default` in *gradSchemes* sets the scheme for all gradient terms in the application, *e.g.* ∇p, $\nabla \mathbf{U}$. When a `default` is specified, it is not necessary to specify each specific term itself in that subdictionary, *i.e.* the entries for `grad(p)`, `grad(U)` in this example. However, if any of these terms are included, the specified scheme overrides the `default` scheme for that term.

Alternatively the user may insist on no `default` scheme by the `none` entry. In this instance the user is obliged to specify all terms in that subdictionary individually. Setting `default` to `none` may appear superfluous since `default` can be overridden. However, specifying `none` forces the user to specify all terms individually which can be useful to remind the user which terms are actually present in the application.

The following sections describe the choice of schemes for each of the categories of terms in Table 4.4.

4.4.1 Interpolation schemes

The *interpolationSchemes* subdictionary contains terms that are interpolations of values typically from cell centres to face centres. A *selection* of interpolation schemes in OpenFOAM are listed in Table 4.5, being divided into 4 categories: 1 category of general schemes; and, 3 categories of schemes used primarily in conjunction with Gaussian discretisation of convection (divergence) terms in fluid flow, described in section 4.4.5. It is *highly unlikely* that the user would adopt any of the convection-specific schemes for general field interpolations in the *interpolationSchemes* subdictionary, but, as valid interpolation schemes, they are described here rather than in section 4.4.5. Note that additional schemes such as `UMIST` are available in OpenFOAM but only those schemes that are generally recommended are listed in Table 4.5.

A general scheme is simply specified by quoting the keyword and entry, *e.g.* a `linear` scheme is specified as `default` by:


```
default linear;
```

The convection-specific schemes calculate the interpolation based on the flux of the flow velocity. The specification of these schemes requires the name of the flux field on which the interpolation is based; in most OpenFOAM applications this is `phi`, the name commonly adopted for the surfaceScalarField velocity flux ϕ. The 3 categories of convection-specific schemes are referred to in this text as: general convection; normalised variable (NV); and, total variation diminishing (TVD). With the exception of the `blended` scheme, the general convection and TVD schemes are specified by the scheme and flux, *e.g.* an `upwind` scheme based on a flux `phi` is specified as `default` by:

```
default upwind phi;
```

Some TVD/NVD schemes require a coefficient $\psi, 0 \leq \psi \leq 1$ where $\psi = 1$ corresponds to TVD conformance, usually giving best convergence and $\psi = 0$ corresponds to best accuracy. Running with $\psi = 1$ is generally recommended. A `limitedLinear` scheme based on a flux `phi` with $\psi = 1.0$ is specified as `default` by:

```
default limitedLinear 1.0 phi;
```

4.4.1.1 Schemes for strictly bounded scalar fields

There are enhanced versions of some of the limited schemes for scalars that need to be strictly bounded. To bound between user-specified limits, the scheme name should be preprended by the word `limited` and followed by the lower and upper limits respectively. For example, to bound the `vanLeer` scheme strictly between -2 and 3, the user would specify:

```
default limitedVanLeer -2.0 3.0;
```

There are specialised versions of these schemes for scalar fields that are commonly bounded between 0 and 1. These are selected by adding `01` to the name of the scheme. For example, to bound the `vanLeer` scheme strictly between 0 and 1, the user would specify:

```
default vanLeer01;
```

Strictly bounded versions are available for the following schemes: `limitedLinear`, `vanLeer`, `Gamma`, `limitedCubic`, `MUSCL` and `SuperBee`.

4.4.1.2 Schemes for vector fields

There are improved versions of some of the limited schemes for vector fields in which the limited is formulated to take into account the direction of the field. These schemes are selected by adding `V` to the name of the general scheme, *e.g.* `limitedLinearV` for `limitedLinear`. 'V' versions are available for the following schemes: `limitedLinearV`, `vanLeerV`, `GammaV`, `limitedCubicV` and `SFCDV`.

Centred schemes	
`linear`	Linear interpolation (central differencing)
`cubicCorrection`	Cubic scheme
`midPoint`	Linear interpolation with symmetric weighting
Upwinded convection schemes	
`upwind`	Upwind differencing
`linearUpwind`	Linear upwind differencing
`skewLinear`	Linear with skewness correction
`QUICK`	Quadratic upwind differencing
TVD schemes	
`limitedLinear`	limited linear differencing
`vanLeer`	van Leer limiter
`MUSCL`	MUSCL limiter
`limitedCubic`	Cubic limiter
NVD schemes	
`SFCD`	Self-filtered central differencing
`Gamma` ψ	Gamma differencing

Table 4.5: Interpolation schemes.

4.4.2 Surface normal gradient schemes

The *snGradSchemes* subdictionary contains surface normal gradient terms. A surface normal gradient is evaluated at a cell face; it is the component, normal to the face, of the gradient of values at the centres of the 2 cells that the face connects. A surface normal gradient may be specified in its own right and is also required to evaluate a Laplacian term using Gaussian integration.

The available schemes are listed in Table 4.6 and are specified by simply quoting the keyword and entry, with the exception of `limited` which requires a coefficient $\psi, 0 \leq \psi \leq 1$ where

$$\psi = \begin{cases} 0 & \text{corresponds to } \texttt{uncorrected}, \\ 0.333 & \text{non-orthogonal correction} \leq 0.5 \times \text{orthogonal part}, \\ 0.5 & \text{non-orthogonal correction} \leq \text{orthogonal part}, \\ 1 & \text{corresponds to } \texttt{corrected}. \end{cases} \tag{4.1}$$

A `limited` scheme with $\psi = 0.5$ is therefore specified as `default` by:

```
default limited 0.5;
```


Scheme	Description
`corrected`	Explicit non-orthogonal correction
`uncorrected`	No non-orthogonal correction
`limited` ψ	Limited non-orthogonal correction
`bounded`	Bounded correction for positive scalars
`fourth`	Fourth order

Table 4.6: Surface normal gradient schemes.

4.4.3 Gradient schemes

The *gradSchemes* subdictionary contains gradient terms. The discretisation scheme for each term can be selected from those listed in Table 4.7.

Discretisation scheme	Description
`Gauss <interpolationScheme>`	Second order, Gaussian integration
`leastSquares`	Second order, least squares
`fourth`	Fourth order, least squares
`limited <gradScheme>`	Limited version of one of the above schemes

Table 4.7: Discretisation schemes available in *gradSchemes*.

The discretisation scheme is sufficient to specify the scheme completely in the cases of `leastSquares` and `fourth`, *e.g.*

```
grad(p) leastSquares;
```

The `Gauss` keyword specifies the standard finite volume discretisation of Gaussian integration which requires the interpolation of values from cell centres to face centres. Therefore, the `Gauss` entry must be followed by the choice of interpolation scheme from Table 4.5. It would be extremely unusual to select anything other than general interpolation schemes and in most cases the `linear` scheme is an effective choice, *e.g.*

```
grad(p) Gauss linear;
```

Limited versions of any of the 3 base gradient schemes — `Gauss`, `leastSquares` and `fourth` — can be selected by preceding the discretisation scheme by `limited`, *e.g.* a limited Gauss scheme

```
grad(p) limited Gauss linear;
```

4.4.4 Laplacian schemes

The *laplacianSchemes* subdictionary contains Laplacian terms. Let us discuss the syntax of the entry in reference to a typical Laplacian term found in fluid dynamics, $\nabla \bullet (\nu \nabla \mathbf{U})$, given the word identifier `laplacian(nu,U)`. The `Gauss` scheme is the only choice of discretisation and requires a selection of both an interpolation scheme for the diffusion coefficient, *i.e.* ν in our example, and a surface normal gradient scheme, *i.e.* $\nabla \mathbf{U}$. To summarise, the entries required are:

```
Gauss <interpolationScheme> <snGradScheme>
```

The interpolation scheme is selected from Table 4.5, the typical choices being from the general schemes and, in most cases, `linear`. The surface normal gradient scheme is selected from Table 4.6; the choice of scheme determines numerical behaviour as described in Table 4.8. A typical entry for our example Laplacian term would be:

```
laplacian(nu,U) Gauss linear corrected;
```

Scheme	Numerical behaviour
`corrected`	Unbounded, second order, conservative
`uncorrected`	Bounded, first order, non-conservative
`limited` ψ	Blend of `corrected` and `uncorrected`
`bounded`	First order for bounded scalars
`fourth`	Unbounded, fourth order, conservative

Table 4.8: Behaviour of surface normal schemes used in *laplacianSchemes*.

4.4.5 Divergence schemes

The *divSchemes* subdictionary contains divergence terms. Let us discuss the syntax of the entry in reference to a typical convection term found in fluid dynamics $\nabla \bullet (\rho \mathbf{U}\mathbf{U})$, which in OpenFOAM applications is commonly given the identifier `div(phi,U)`, where `phi` refers to the flux $\phi = \rho \mathbf{U}$.

The `Gauss` scheme is only choice of discretisation and requires a selection of the interpolation scheme for the dependent field, *i.e.* **U** in our example. To summarise, the entries required are:

```
Gauss <interpolationScheme>
```

The interpolation scheme is selected from the full range of schemes in Table 4.5, both general and convection-specific. The choice critically determines numerical behaviour as described in Table 4.9. The syntax here for specifying convection-specific interpolation schemes *does not include the flux* as it is already known for the particular term, *i.e.* for `div(phi,U)`, we know the flux is `phi` so specifying it in the interpolation scheme would only invite an inconsistency. Specification of upwind interpolation in our example would therefore be:

```
div(phi,U) Gauss upwind;
```


Scheme	Numerical behaviour
`linear`	Second order, unbounded
`skewLinear`	Second order, (more) unbounded, skewness correction
`cubicCorrected`	Fourth order, unbounded
`upwind`	First order, bounded
`linearUpwind`	First/second order, bounded
`QUICK`	First/second order, bounded
TVD schemes	First/second order, bounded
`SFCD`	Second order, bounded
NVD schemes	First/second order, bounded

Table 4.9: Behaviour of interpolation schemes used in *divSchemes*.

4.4.6 Time schemes

The first time derivative $(\partial/\partial t)$ terms are specified in the *ddtSchemes* subdictionary. The discretisation scheme for each term can be selected from those listed in Table 4.10.

There is an off-centering coefficient ψ with the `CrankNicholson` scheme that blends it with the `Euler` scheme. A coefficient of $\psi = 1$ corresponds to pure `CrankNicholson` and and $\psi = 0$ corresponds to pure `Euler`. The blending coefficient can help to improve stability in cases where pure `CrankNicholson` are unstable.

Scheme	Description
`Euler`	First order, bounded, implicit
`CrankNicholson` ψ	Second order, bounded, implicit
`backward`	Second order, implicit
`steadyState`	Does not solve for time derivatives

Table 4.10: Discretisation schemes available in *ddtSchemes*.

When specifying a time scheme it must be noted that an application designed for transient problems will not necessarily run as steady-state and visa versa. For example the solution will not converge if `steadyState` is specified when running icoFoam, the transient, laminar incompressible flow code; rather, simpleFoam should be used for steady-state, incompressible flow.

Any second time derivative $(\partial^2/\partial t^2)$ terms are specified in the *d2dt2Schemes* subdictionary. Only the `Euler` scheme is available for *d2dt2Schemes*.

4.4.7 Flux calculation

The *fluxRequired* subdictionary lists the fields for which the flux is generated in the application. For example, in many fluid dynamics applications the flux is generated after solving a pressure equation, in which case the *fluxRequired* subdictionary would simply be entered as follows, `p` being the word identifier for pressure:

```
fluxRequired
{
    p;
```

```
}
```

4.5 Solution and algorithm control

The equation solvers, tolerances and algorithms are controlled from the *fvSolution* dictionary in the *system* directory. Below is an example set of entries from the *fvSolution* dictionary required for the icoFoam solver.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p PCG
    {
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
    };

    U PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0;
    };
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}


// ************************************************************************* //
```

fvSolution contains a set of subdictionaries that are specific to the solver being run. However, there is a small set of standard subdictionaries that cover most of those used by the standard solvers. These subdictionaries include *solvers*, *relaxationFactors*, *PISO* and *SIMPLE* which are described in the remainder of this section.

4.5.1 Linear solver control

The first subdictionary in our example, and one that appears in all solver applications, is `solvers`. It specifies each linear-solver that is used for each discretised equation; it is emphasised that the term *linear*-solver refers to the method of number-crunching to solve the set of linear equations, as opposed to *application* solver which describes the set of equations and algorithms to solve a particular problem. The term 'linear-solver' is abbreviated to 'solver' in much of the following discussion; we hope the context of the term avoids any ambiguity.

The syntax for each entry within *solvers* begins with a keyword that is the word relating to the variable being solved in the particular equation. For example, icoFoam solves equations for velocity **U** and pressure p, hence the entries for `U` and `p`. The variable name is followed by the solver name and a dictionary containing the parameters that the solver uses. The solvers available in OpenFOAM are listed in Table 4.11. The parameters, including `tolerance`, `relTol`, `preconditioner`, *etc.* are described in following sections.

The solvers distinguish between symmetric matrices and asymmetric matrices. The symmetry of the matrix depends on the structure of the equation being solved and, while

Solver	Keyword
Preconditioned (bi-)conjugate gradient	`PCG`/`PBiCG`†
Solver using a smoother	`smoothSolver`
Generalised geometric-algebraic multi-grid	`GAMG`

†`PCG` for symmetric matrices, `PBiCG` for asymmetric

Table 4.11: Linear solvers.

the user may be able to determine this, it is not essential since OpenFOAM will produce an error message to advise the user if an inappropriate solver has been selected, *e.g.*

```
--> FOAM FATAL IO ERROR : Unknown asymmetric matrix solver PCG
Valid asymmetric matrix solvers are :
3
(
PBiCG
smoothSolver
GAMG
)
```

4.5.1.1 Solution tolerances

The sparse matrix solvers are iterative, *i.e.* they are based on reducing the equation residual over a succession of solutions. The residual is ostensibly a measure of the error in the solution so that the smaller it is, the more accurate the solution. More precisely, the residual is evaluated by substituting the current solution into the equation and taking the magnitude of the difference between the left and right hand sides; it is also normalised in to make it independent of the scale of problem being analysed.

Before solving an equation for a particular field, the initial residual is evaluated based on the current values of the field. After each solver iteration the residual is re-evaluated. The solver stops if *either* of the following conditions are reached:

- the residual falls below the *solver tolerance*, `tolerance`;
- the ratio of current to initial residuals falls below the *solver relative tolerance*, `relTol`;

The solver tolerance should represents the level at which the residual is small enough that the solution can be deemed sufficiently accurate. The solver relative tolerance limits the relative improvement from initial to final solution. It is quite common to set the solver relative tolerance to 0 to force the solution to converge to the solver tolerance. The tolerances, `tolerance` and `relTol` must be specified in the dictionaries for all solvers.

4.5.1.2 Preconditioned conjugate gradient solvers

There are a range of options for preconditioning of matrices in the conjugate gradient solvers, represented by the `preconditioner` keyword in the solver dictionary. The preconditioners are listed in Table 4.12.

Preconditioner	Keyword
Diagonal incomplete-Cholesky (symmetric)	DIC
Faster diagonal incomplete-Cholesky (DIC with caching)	FDIC
Diagonal incomplete-LU (asymmetric)	DILU
Diagonal	diagonal
Geometric-algebraic multi-grid	GAMG
No preconditioning	none

Table 4.12: Preconditioner options.

4.5.1.3 Smooth solvers

The solvers that use a smoother require the smoother to be specified. The smoother options are listed in Table 4.13. Generally GaussSeidel is the most reliable option, but for bad matrices DIC can offer better convergence. In some cases, additional post-smoothing using GaussSeidel is further beneficial, *i.e.* the method denoted as DICGaussSeidel

Smoother	Keyword
Gauss-Seidel	GaussSeidel
Diagonal incomplete-Cholesky (symmetric)	DIC
Diagonal incomplete-Cholesky with Gauss-Seidel (symmetric)	DICGaussSeidel

Table 4.13: Smoother options.

The user must also pecify the number of sweeps, by the nSweeps keyword, before the residual is recalculated, following the tolerance parameters.

4.5.1.4 Geometric-algebraic multi-grid solvers

The generalised method of geometric-algebraic multi-grid (GAMG) uses the principle of: generating a quick solution on a mesh with a small number of cells; mapping this solution onto a finer mesh; using it as an initial guess to obtain an accurate solution on the fine mesh. GAMG is faster than standard methods when the increase in speed by solving first on coarser meshes outweighs the additional costs of mesh refinement and mapping of field data. In practice, GAMG starts with the mesh specified by the user and coarsens/refines the mesh in stages. The user is only required to specify an approximate mesh size at the most coarse level in terms of the number of cells nCoarsestCells.

The agglomeration of cells is performed by the algorithm specified by the agglomerator keyword. Presently we recommend the faceAreaPair method. It is worth noting there is an MGridGen option that requires an additional entry specifying the shared object library for MGridGen:

```
geometricGamgAgglomerationLibs ("libMGridGenGamgAgglomeration.so");
```

In the experience of OpenCFD, the MGridGen method offers no obvious benefit over the faceAreaPair method. For all methods, agglomeration can be optionally cached by the cacheAgglomeration switch.

Smoothing is specified by the smoother as described in section 4.5.1.3. The number of sweeps used by the smoother at different levels of mesh density are specified by the nPreSweeps, nPostSweeps and nFinestSweeps keywords. The nPreSweeps entry is used as the algorithm is coarsening the mesh, nPostSweeps is used as the algorithm is refining, and nFinestSweeps is used when the solution is at its finest level.

The mergeLevels keyword controls the speed at which coarsening or refinement levels is performed. It is often best to do so only at one level at a time, *i.e.* set mergeLevels 1. In some cases, particularly for simple meshes, the solution can be safely speeded up by coarsening/refining two levels at a time, *i.e.* setting mergeLevels 2.

4.5.2 Solution under-relaxation

A second subdictionary of *fvSolution* that is often used in OpenFOAM is *relaxationFactors* which controls under-relaxation, a technique used for improving stability of a computation, particularly in solving steady-state problems. Under-relaxation works by limiting the amount which a variable changes from one iteration to the next, either by modifying the solution matrix and source prior to solving for a field or by modifying the field directly. An under-relaxation factor $\alpha, 0 < \alpha \leq 1$ specifies the amount of under-relaxation, ranging from none at all for $\alpha = 1$ and increasing in strength as $\alpha \to 0$. The limiting case where $\alpha = 0$ represents a solution which does not change at all with successive iterations. An optimum choice of α is one that is small enough to ensure stable computation but large enough to move the iterative process forward quickly; values of α as high as 0.9 can ensure stability in some cases and anything much below, say, 0.2 are prohibitively restrictive in slowing the iterative process.

The user can specify the relaxation factor for a particular field by specifying first the word associated with the field, then the factor. The user can view the relaxation factors used in a tutorial example of simpleFoam for incompressible, laminar, steady-state flows.

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  solvers
26  {
27      p PCG
28      {
29          preconditioner   DIC;
30          tolerance        1e-06;
31          relTol           0.01;
32      };
33      U PBiCG
34      {
35          preconditioner   DILU;
36          tolerance        1e-05;
37          relTol           0.1;
38      };
39      k PBiCG
40      {
41          preconditioner   DILU;
42          tolerance        1e-05;
43          relTol           0.1;
44      };
45      epsilon PBiCG
46      {
47          preconditioner   DILU;
48          tolerance        1e-05;
49          relTol           0.1;
50      };
51      R PBiCG
52      {
53          preconditioner   DILU;
54          tolerance        1e-05;
55          relTol           0.1;
56      };
```

```
    nuTilda PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0.1;
    };
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
    p               0.3;
    U               0.7;
    k               0.7;
    epsilon         0.7;
    R               0.7;
    nuTilda         0.7;
}


// ************************************************************************* //
```

4.5.3 PISO and SIMPLE algorithms

Most fluid dynamics solver applications in OpenFOAM use the pressure-implicit split-operator (PISO) or semi-implicit method for pressure-linked equations (SIMPLE) algorithms. These algorithms are iterative procedures for solving equations for velocity and pressure, PISO being used for transient problems and SIMPLE for steady-state.

Both algorithms are based on evaluating some initial solutions and then correcting them. SIMPLE only makes 1 correction whereas PISO requires more than 1, but typically not more than 4. The user must therefore specify the number of correctors in the PISO dictionary by the `nCorrectors` keyword as shown in the example on page U-115.

An additional correction to account for mesh non-orthogonality is available in both SIMPLE and PISO in the standard OpenFOAM solver applications. A mesh is orthogonal if, for each face within it, the face normal is parallel to the vector between the centres of the cells that the face connects, *e.g.* a mesh of hexahedral cells whose faces are aligned with a Cartesian coordinate system. The number of non-orthogonal correctors is specified by the `nNonOrthogonalCorrectors` keyword as shown in the examples above and on page U-115. The number of non-orthogonal correctors should correspond to the mesh for the case being solved, *i.e.* 0 for an orthogonal mesh and increasing with the degree of non-orthogonality up to, say, 20 for the most non-orthogonal meshes.

4.5.3.1 Pressure referencing

In a closed incompressible system, pressure is relative: it is the pressure range that matters not the absolute values. In these cases, the solver sets a reference level of `pRefValue` in cell `pRefCell` where `p` is the name of the pressure solution variable. Where the pressure is `pd`, the names are `pdRefValue` and `pdRefCell` respectively. These entries are generally stored in the *PISO*/*SIMPLE* subdictionary and are used by those solvers that require them when the case demands it. If ommitted, the solver will not run, but give a message to alert the user to the problem.

4.5.4 Other parameters

The *fvSolutions* dictionaries in the majority of standard OpenFOAM solver applications contain no other entries than those described so far in this section. However, in general the *fvSolution* dictionary may contain any parameters to control the solvers, algorithms, or in fact anything. For a given solver, the user can look at the source code to find the parameters required. Ultimately, if any parameter or subdictionary is missing when an solver is run, it will terminate, printing a detailed error message. The user can then add missing parameters accordingly.

Chapter 5

The FoamX case manager

OpenFOAM is distributed with the FoamX utility to manage the running of cases. FoamX is a GUI that can manage cases over a distributed network, *e.g.* the Internet, although most often it is used to manage cases on a local machine.

This chapter contains mainly reference material for FoamX, and while section 5.3 and section 5.4 provide useful advice on the general use of FoamX, new users are first directed to the tutorials (chapter 2) to learn how to use FoamX.

The mechanism for running cases over a network is to have a host machine providing services that can be called from a JAVA GUI on another machine. The interface between the JAVA GUI and these services — a host browser, case browser and case server, written in C++ — is MICO, an implementation the Common Object Request Broker Architecture (CORBA). If the user simply wishes to manage cases on their local machine, the host browser and JAVA GUI can both be launched from that machine. We shall refer to this as **normal mode** in the following sections. Let us summarise the options below:

host browser run locally (normal mode) in this case the user can launch **both the host browser and GUI** by executing runFoamX

```
runFoamX
```

host browser run remotely (remote mode) in this case the host browser is first launched on the host machine by runFoamXHB

```
runFoamXHB
```

and the GUI is launched locally by executing runFoamX which connects to the running host browser

```
runFoamX
```

The processes involved in both these options are shown in Figure 5.1. When runFoamX is executed, it searches for a running host browser. If one is running, *i.e.* previously launched with runFoamXHB, it will connect to it; otherwise it starts a host browser itself. In section 5.1, section 5.2 and section 5.3 the general operation of FoamX is described with particular emphasis on how it can be operated over a network. Following that, the running of OpenFOAM cases through the case server is described in section 5.4. Configuration issues relating to FoamX are described in section 5.5.

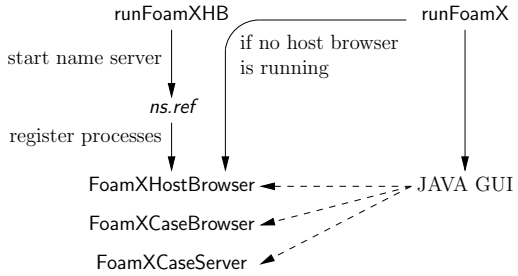
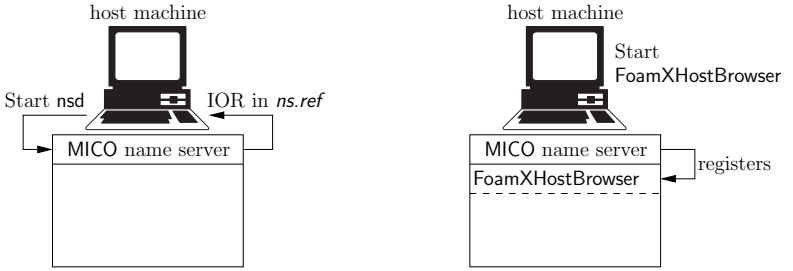


Figure 5.1: Options for running FoamX.



(a) Starting the nsd name server.

(b) Starting the FoamXHostBrowser.

Figure 5.2: Running runFoamXHB.

5.1 The name server and host browser

To start the FoamX host browser on the host machine, the user should either run the runFoamXHB script, or, in the case that the host browser is run locally (normal mode), run runFoamX which itself launches runFoamXHB. runFoamXHB performs two functions as shown in Figure 5.2.

- The MICO name server — a process called nsd — is started by the host machine. It uses the host name and a default port address that can be set manually by the `org.omg.CORBA.ORBInitialHost=` and `org.omg.CORBA.ORBInitialPort=` entries in the *FoamXClient.cfg* file of the *.OpenFOAM-1.4.1/apps/FoamX* directory. The name server writes the host/port address in IOR form in the *ns.ref* file in the same directory.
- The FoamXHostBrowser process is started on the host/port address where nsd was started and registers itself under the name `FoamXHostBrowser`.

Therefore the execution of runFoamXHB, by typing at the command prompt

```
runFoamXHB
```

launches the name server and host browser which outputs to screen the following:

```
Starting NameServer with inet:<host>:<port>...
Starting FoamX Host Browser with inet:<host>:<port>...
```

where <host>:<port> are set by default or are those specified in the *FoamXClient.cfg* file. The FoamXHostBrowser prints the OpenFOAM logo strip to screen and details about its execution status to indicate it is running correctly.

5.1.1 Notes for running the name server

- The contents of the *ns.ref* file can be 'translated' and viewed by typing

  ```
  iordump < $FOAMX_USER_CONFIG/ns.ref
  ```

- An administration tool for MICO can be started by typing

  ```
  nsadmin  -ORBNamingAddr inet:<host>:<port>
  ```

  where the `inet:<host>:<port>` entry can be found by viewing the *ns.ref* file. The user should type `help` to view the options within the tool, which include `ls` to list the registered services.

5.2 The JAVA GUI

Any remote machine, or the host machine itself, can connect to the name server using a copy of the *ns.ref* file generated previously to provide the IOR. The remote machine also needs the `org.omg.CORBA.ORBInitialHost=` entry to be set to the name of the host machine in the *FoamXClient.cfg* file, with a corresponding entry in its */etc/hosts* file as described in section 5.1.1.

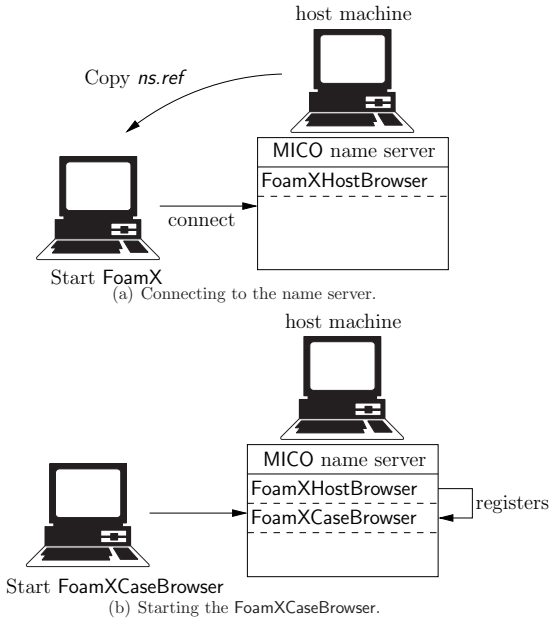
To start the FoamX JAVA GUI on a remote machine as shown in Figure 5.3 a), the user should run the runFoamX script which should locate the name server already launched by runFoamXHB. The user will be prompted on the command line to acknowledge that they wish to connect to this server:

```
Found server reference $FOAMX_USER_CONFIG/ns.ref
Do you want to connect to this server ? (n)
```

A new name server will be created locally if the user decides not to connect to the existing name server or if no name server exists, as in the case where runFoamXHB has not been executed. This is why when running both host browser and GUI locally it is sufficient to execute runFoamX without running runFoamXHB. Typing, at a command prompt

```
runFoamX
```

opens the JAVA browser window, as shown in Figure 5.4. The browser is split into the following regions:



(a) Connecting to the name server.

(b) Starting the FoamXCaseBrowser.

Figure 5.3: Running runFoamX.

Menu bar and buttons (top) containing the operations used in creation, construction and running of a case;

Case panel (left) consisting of the case directory tree in the case browser and the contents of the OpenFOAM cases in the case server;

Editing panel (right, blue) in which the editing of case entries is done;

Progress history panel (bottom) a dialogue box which informs on certain actions that have been performed.

By default the case panel will display the host machine on which the name server is run. **If the user wishes to access cases on other remote machines**, they should list the machines in `hosts` in the *.OpenFOAM-1.4.1/controlDict* file. The FoamX window can be resized in the normal manner; the individual windows within it can also be resized by clicking on the speckled bars separating the windows and dragging the cursor across the screen.

There are three ways to pass commands to the browser:

- selecting an item and double-clicking, typically to open its contents;
- selecting an item and clicking the right mouse button brings up a menu of operations which can be performed on that item;

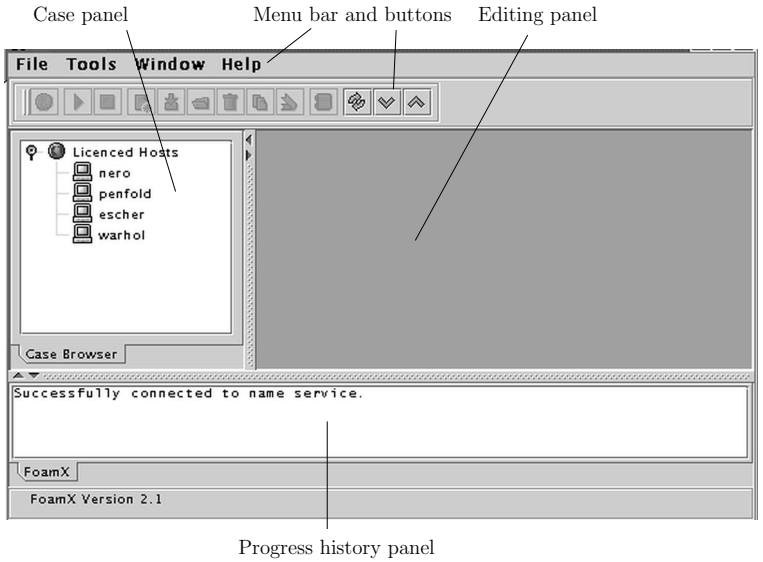


Figure 5.4: FoamX main browser window

- selecting an item from the menu bar and buttons can perform other operations.

Note that if the cursor is held over any menu button for one second a short description of the button's use appears in a small dialogue box below the base of the cursor.

5.3 The case browser

From the JAVA GUI, a case browser may be opened for a machine listed in the case panel by: either double-clicking on the host icon; or, highlighting the host with a single-click and selecting Open Case Browser ( ) from a the menu buttons or right mouse button. This operation makes a call to the FoamXHostBrowser to open a FoamXCaseBrowser as shown in Figure 5.3 b). The FoamXCaseBrowser reads the *ns.ref* file to get a reference to the name server and registers itself. The JAVA GUI can then look up the FoamXCaseBrowser and make calls to it, *e.g.* to start up a FoamXCaseServer to start working on a case. The FoamXCaseServer registers itself on the name server, and so the process continues of registering services and making calls to them.

Note that a case browser may be opened automatically at launch of the JAVA GUI by executing runFoamX with the host as an argument

```
runFoamX [host]
```

Starting the case browser on a host machine produces a directory tree list of root path directories in which OpenFOAM cases are stored as shown in Figure 5.5. The case roots

specified in the user's *.OpenFOAM-1.4.1/controlDict* file; for information on adding or removing case roots, please refer to section 5.5.2.

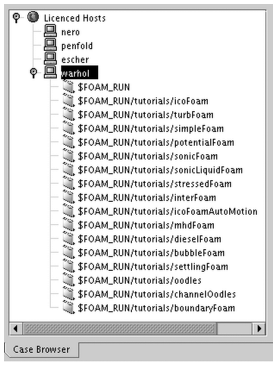
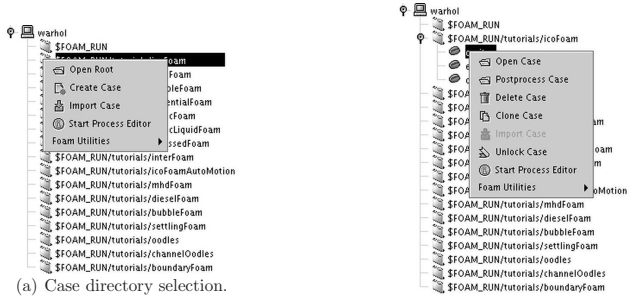


Figure 5.5: Case root directory tree.

For the remainder of the manual:
It will be assumed that any operation in FoamX, described in the text, is selected either from the menu bar or button, or by a right button click on the mouse unless otherwise stated.



(a) Case directory selection.

(b) Case name selection.

Figure 5.6: Case browser functions.

The case browser offers a range of functions as shown in Figure 5.6. By selecting a root directory icon, the user may open the directory, create a new case, import a case or run some utilities; by highlighting a case name icon, the user may open, delete, clone or unlock that case or run OpenFOAM utilities on the case.

5.3.1 Opening a root directory

The current set of cases within a case root directory can be viewed by selecting the the `Open Root` function by placing the cursor over the root directory and clicking the right mouse button to reveal the menu as shown in Figure 5.6 a), or by a double-click on the root directory icon. The directory opens to reveal a case tree for that root directory as shown in Figure 5.7.

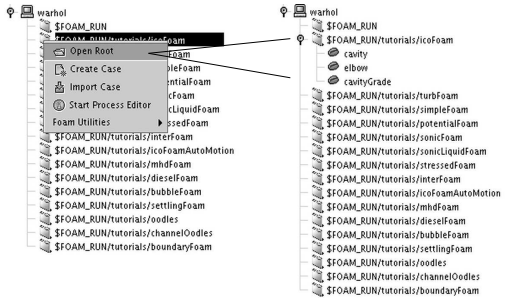


Figure 5.7: Opening a case root.

5.3.2 Creating a new case

A new case is created by selecting the `Create Case` function ( ) either from the menu buttons or by placing the cursor over the host icon or a case directory and clicking the right mouse button as shown in Figure 5.8. A small window appears with data entry boxes for the `Class`, Case Name and Case Root as shown in Figure 5.8. The `Class` provides a scroll menu

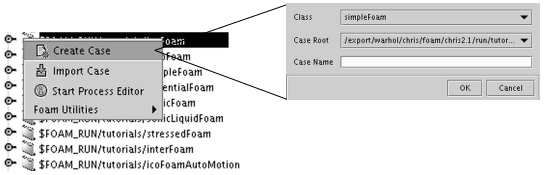


Figure 5.8: Creating a new case.

containing OpenFOAM solver names, such as icoFoam and turbFoam. FoamX generates the necessary data entries in the case files required by the selected solver; hence, it is essential to choose the correct solver. `Case Name` and `Case Root` are the directory path and directory name respectively, in which the case data is stored according to the file structure described in section 4.1. Once the correct entries have been made, click OK. A case server for the new case is opened allowing the user to edit case files, run solvers and utilities, *etc.* as described in section 5.4.

5.3.3 Opening an existing case

The `Open Case` function ( ) opens an existing case in a case server as shown in Figure 5.9. The case server allows the user to edit case files, run solvers and utilities, *etc.* as described in section 5.4.

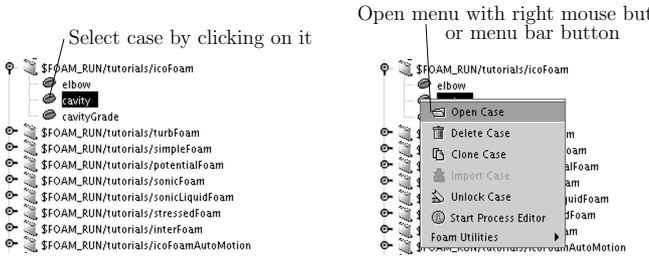


Figure 5.9: Opening an existing case.

5.3.4 Deleting an existing case

The user may highlight a case and select the `Delete Case` function ( ) to delete the case directory from the hard disk. As shown in Figure 5.10, the function prompts the user with a window asking whether he/she wishes to delete the case which the user may accept by clicking the Yes button or decline with the No button.

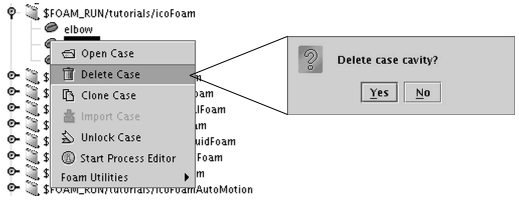


Figure 5.10: Deleting an existing case.

5.3.5 Cloning an existing case

The `Clone Case` function ( ) creates a new case into which existing files from a selected case are copied. As shown in Figure 5.11, the user must first highlight the case that is to be cloned and select the `Clone Case` function. This opens a table in which the new `case` name must be specified and the `root` path and the `applicationClass` may be changed to something different to those of the case being cloned. Finally the `times` entry allows the user to choose the time directories that are copied during the clone operation. The options are listed in Table 5.1.

Option	Description
`firstTime`	Copies the earliest time directory
`latestTime`	Copies the most recent time directory
`allTime`	Copies all time directories
`noTime`	Copies no time directories

Table 5.1: Options for copying time directories in a `Clone Case` operation.

On entering the correct information and clicking the Close button, the user is prompted to complete the clone operation. The new case can then be opened as described in section 5.3.3.

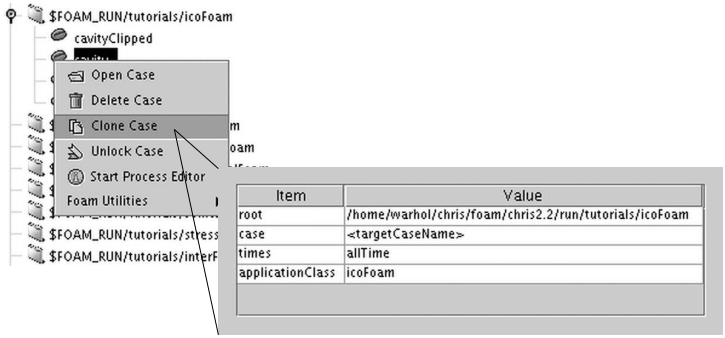


Figure 5.11: Cloning an existing case.

5.3.6 Unlocking an existing case

When a case is created or opened, a lock file is created to prevent the case being opened in a separate server. When the case is closed, the lock file is removed to allow it to be opened once more. In a few circumstances the lock file may not be deleted even though the case is no longer being processed in a case server, *e.g.* if the host browser is killed while the case is open in the case server. The `Unlock Case` function ( ) therefore provides the option of deleting the lock file. As shown in Figure 5.12, it presents a window warning the user that the case may be being processed by another user. It is then the user's responsibility to ensure that it is not being processed elsewhere before accepting to delete the lock file.

5.3.7 The process editor

The `Start Process Editor` function ( ) opens an editor in which the user can monitor all the OpenFOAM jobs that are finished and currently running. The editor is simply a GUI that reads the files in the *runningJobs* and *finishedJobs* directories, located in the *$FOAM_LIC_DIR* directory of the installation. It consists of a window as shown in Figure 5.13. Tags allow the user to move between a runningJobs table and a finishedJobs

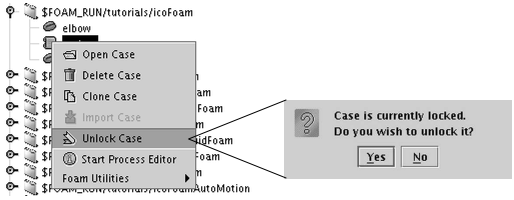
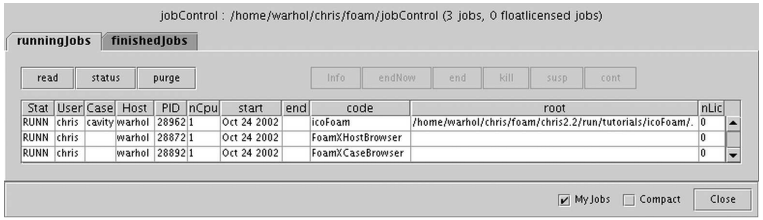


Figure 5.12: Unlocking an existing case.



(a) Running jobs table.

(b) Finished jobs table.

Figure 5.13: The process editor.

table. The tables contain the details of jobs which are fairly self-explanatory. There are buttons above and to the left of the runningJobs table that perform the tasks listed in Table 5.2. The user may select a job by clicking on it in the runningJobs table, which activates the buttons above right of the table. These buttons allow the user to control jobs as listed in Table 5.2.

The finishedJobs table is an archive of information of jobs that were running in OpenFOAM but were terminated for one reason or another. The user is free to store the entries they find useful and delete those that are not. There are 2 buttons for deleting entries in the table: the purge button deletes finished jobs that are older than 7 days; the remove button simply removes a selected entry from the table.

There are two check boxes at the bottom of the process editor window that govern which jobs are listed in the runningJobs and finishedJobs tables as listed in Table 5.2.

Main buttons	
read	Re-reads the jobs in the *runningJobs* and *finishedJobs* directories
status	Contacts host machines to update the status of jobs
purge	Removes jobs that are no longer running

Running jobs buttons	
Info	Displays an information panel about the job
endNow	Forces the job to stop at the end of the next time step
end	Forces the job to stop next time step the job outputs field data to file
kill	Terminates the job immediately
suspend	Suspends the job immediately
cont	Restarts a suspended job

Check boxes	
My Jobs	Only shows the jobs of the current user
Compact	Removes the jobs relating to FoamX from the list

Table 5.2: Process editor buttons.

5.3.8 Running OpenFOAM utilities

The `Foam Utilities` function allows the user to run OpenFOAM utilities. This function is also offered in the case server and is more commonly used there; it is therefore described in section 5.4.

5.4 The case server

When a case is opened from the case browser, a case server starts up. A directory tree appears in the case window as shown in Figure 5.14. The user can move between the new case and case browser windows using the tags at the base of the case window. The directory tree contains 3 entries at the top level:

Dictionaries Contains the dictionaries for controlling the case and setting physical properties.

Fields Sets the initial and boundary values for the fields.

Mesh Reads/imports a mesh and sets the boundary conditions for the patches of the mesh.

5.4.1 Importing an existing mesh

The case requires a mesh, either created using the blockMesh utility described in section 6.3 or using third-party software combined with the OpenFOAM mesh converters. A OpenFOAM mesh is stored in the *constant/polyMesh* directory of the case as: either the files that constitute a OpenFOAM mesh — *boundary*, *cells* *etc.*; or, as a *blockMeshDict* file that blockMesh uses to create a OpenFOAM mesh; or, both. The user may import all these files from an existing *constant/polyMesh* directory into their case using the `Import Mesh` function as shown in Figure 5.15.

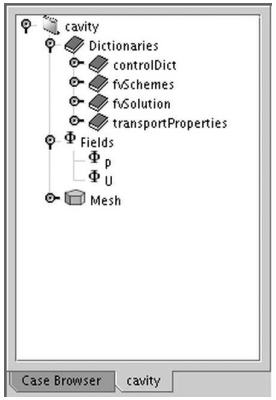


Figure 5.14: Case server window

5.4.2 Reading a mesh

Once the mesh files exist in the *constant/polyMesh* directory, whether imported directly or generated by blockMesh or one of the mesh converter utilities, they can be read into the case server using the `Read Mesh&Fields` function. Should the reader wish to test this function, they can open one of the tutorial examples and generate a mesh with the blockMesh utility as described in section 5.4.8.

5.4.3 Setting boundary patches

As shown in Figure 5.16, once the `Read Mesh&Fields` function executed, the directory tree displays a list of the boundary patches for the mesh. The user can then impose physical boundary conditions onto a patch by highlighting the patch and selecting the `Define Boundary Type` function. This brings up a patch description window inside the editing panel. As Figure 5.17 illustrates, the physical boundary type can be selected by clicking on the ... button to the right of the `Boundary Type` descriptor. This opens a new window listing the physical boundary types available to the specific solver. The user make a selection from the list and click OK, which closes the window and returns the user to the patch description window. Beneath the physical boundary type descriptor is a table listing the primitive variables that are present in the solver and their numerical patch types, or boundary conditions, used in the solution. The user should select the physical boundary types for all the patches noting that in 2D cases the front and back patches, aligned in the 2D solution plane, should be assigned the empty type.

5.4.4 Setting the fields

Once all the physical patch types are specified, the Fields can be edited using the `Edit Field` function, selected as usual by highlighting the field and clicking the right mouse button or by double-clicking on the field icon. The `Edit Field` function brings up a field window in

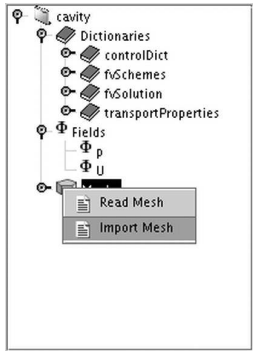


Figure 5.15: Importing a OpenFOAM mesh

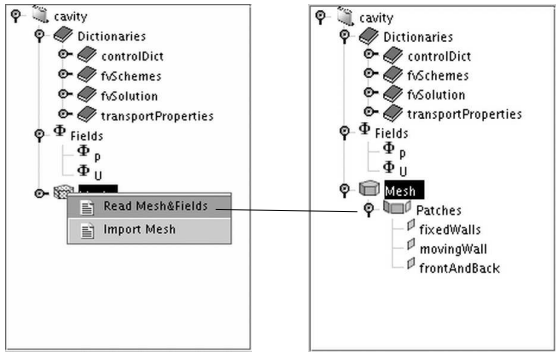


Figure 5.16: Reading a OpenFOAM mesh

the editing panel as shown in Figure 5.18. The table lists a series of data values required for each field as outlined in section 4.2.7: `internalField`, `referenceLevel` and any values corresponding to one or more patches required from the physical type specification. Note that the patch list is updated to accommodate any changes to the specification of a physical patch type. The user can click on entries in the Value column to change values. In Figure 5.18 we demonstrate the setting of a uniform velocity of $(1, 0, 0)$ m/s on the patch named `movingWall`.

5.4.5 Editing the dictionaries

The user can edit the data in the Dictionaries. The dictionaries include *controlDict*, shown in Figure 5.19, *fvSchemes*, *fvSolution*, described in section 4.3, section 4.4 and section 4.5 respectively, and those for material properties. The dictionaries present the entry in tabular form with the data entry in the right column. Clicking on the entry will allow the user to edit

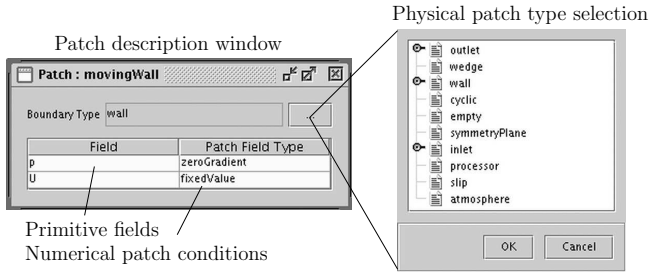


Figure 5.17: Selecting the physical boundary types

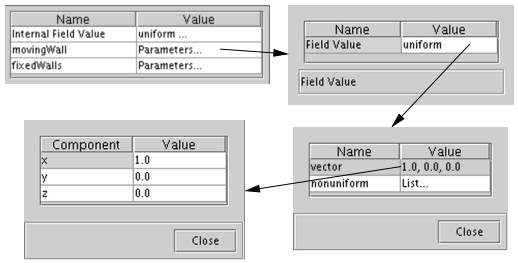


Figure 5.18: Editing a field and setting patch conditions

the value directly or open a sub-dictionary whose values can be edited in the same manner. Note that entries that are printed in grey, *e.g.* the `applicationClass` in Figure 5.19 are non-editable. Also note that some entries are selected from a `Selection Editor`; in this case the selected entry is that which is highlighted in green.

5.4.6 Saving data

The user can save any changes to the case by selecting the `Save Case` function ( ) from the button bar. The dictionary, fields and mesh data will be saved.

5.4.7 Running solvers

The user can run the solver for which the case is written in one of two ways. To run immediately in the foreground, the user should select the `Start Calculation Now` function ( ) from the button bar. The OpenFOAM solver is immediately launched without prompting the user for more information.

Alternatively, the user can select the `Start Calculation` function ( ) from the button bar. This brings up a Run Application window as shown in Figure 5.20. The user may select to run the case in the background by clicking the background button, before pressing the Start Run button. For a case run in the background, the progress history is written to a log file specified in the log text box, which can be viewed by pressing the View Log button.

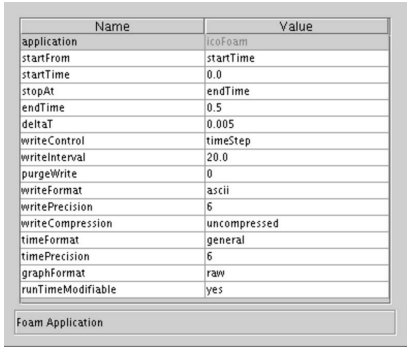


Figure 5.19: Example dictionary window: *controlDict*

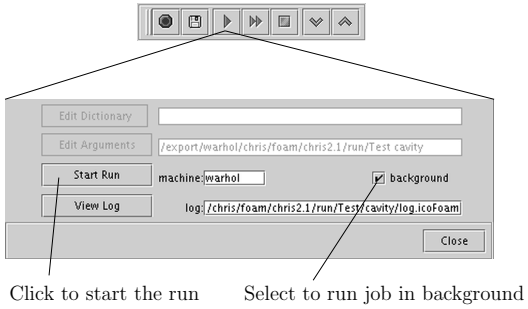


Figure 5.20: Running a solver using the `Start Calculation` function

5.4.8 Running utilities

There are numerous utilities supplied with OpenFOAM that can be executed by highlighting the case name icon in the case server window and clicking the right mouse button which opens a hierarchy of menus containing the utilities, as shown in Figure 5.21. Selecting a utility, blockMesh in our example in Figure 5.22, opens up a window in which the user can edit the dictionary associated with the utility, if one exists. The mandatory command line arguments are set by default for the case that is being edited. The user can select optional arguments accordingly from the table.

5.4.9 Closing the case server

The user should click the `Close Case` button () to close the case server window and return the user to the case browser.

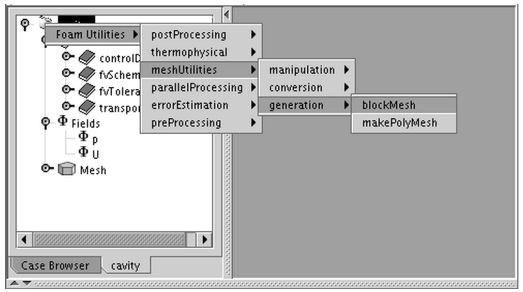


Figure 5.21: Running a utility

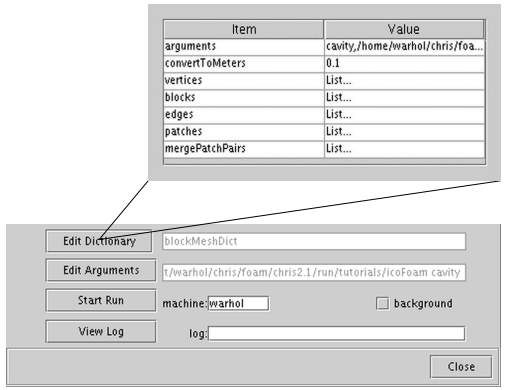


Figure 5.22: Opening the utility dictionary

5.5 Configuration to run FoamX

The FoamX user configuration files are located in the user *.OpenFOAM-1.4.1/apps/FoamX* directory, that may be copied to the user's *$HOME*, maintaining the directory structure. The files that can be configured, if the user so wishes, are:

FoamXClient.cfg contains settings for the networking and appearance of FoamX. In particular, the user may wish to set:

- the host/port address given by the `org.omg.CORBA.ORBInitialHost=` and `org.omg.CORBA.ORB` entries.
- the default browser, by editing the `FoamX.Browser=` entry to `netscape`, `mozilla`, `konqueror` or any other browser or executable that can be passed a URL;
- the default editor, by commenting out (`#`) the relevant entries for `FoamX.Editor=` to leave the editor of choice from *e.g.* `internal`, `nedit`, `xemacs`.

FoamX.cfg contains settings for `processControl` that can be edited. In particular the user should set the `remoteShell` to `rsh` or `ssh`, depending on whether they are running remote or secure shell. The file also contains settings for timings associated with the connection timing out and retries of commands which can be increased if the user experiences problems.

The environment variables associated with FoamX compilation are prefixed by $FOAMX_ and listed in Table 5.3.

Environment variable	Description and options
$FOAMX_PATH	Path to FoamX installation, *$FOAM_UTIL/FoamX*
$FOAMX_SYSTEM_CONFIG	Path to FoamX system configuration files, *$FOAMX_PATH/config*
$FOAMX_USER_CONFIG	Path to FoamX user configuration files, *$HOME/$FOAM_DOT_DIR/apps/FoamX*

Table 5.3: Environment variable settings for FoamX.

5.5.1 JAVA

The FoamX case browser uses JAVA 1.4.2 which may be installed as standard on the machine, although perhaps not the required version. It is therefore supplied with the OpenFOAM release and the $JAVA_HOME environment variable is specified by default in *$WM_PROJECT_DIR/.bashrc* (or *.cshrc*) to the top level directory of the supplied JAVA release. The system administrator may choose to install JAVA 1.4.2 in an alternative location setting $JAVA_HOME accordingly.

5.5.2 Paths to case files

FoamX finds paths to the user's case files from the `caseRoots` entries in the *.OpenFOAM-1.4.1/controlDict* file. By default they are set as:

```
caseRoots
(
    "."
    "$FOAM_RUN/tutorials/icoFoam"
    "$FOAM_RUN/tutorials/turbFoam"
    ...
);
```

where $FOAM_RUN points by default to the directory *$HOME/OpenFOAM/${USER}-1.4.1/run*. This means that by default the user can open cases in the tutorial directory copied to their *run* directory and cases within the directory from which FoamX is launched. If the user wished to set their own paths, they should do so in a local copy of *controlDict* file in the *$HOME/.OpenFOAM-1.4.1* directory.

Chapter 6

Mesh generation and conversion

This chapter describes all topics relating to the creation of meshes in OpenFOAM: section 6.1 gives an overview of the ways a mesh may be described in OpenFOAM; section 6.3 covers the blockMesh utility for generating meshes; section 6.4 describes the options available for conversion of a mesh that has been generated by a third-party product into a format that OpenFOAM can read.

6.1 Mesh description

This section provides a specification of the way the OpenFOAM C++ classes handle a mesh. The mesh is an integral part of the numerical solution and must satisfy certain criteria to ensure a valid, and hence accurate, solution. During any run, OpenFOAM checks that the mesh satisfies a fairly stringent set of validity constraints and will cease running if the constraints are not satisfied. The consequence is that a user may experience some frustration in 'correcting' a large mesh generated by third-party mesh generators before OpenFOAM will run using it. This is unfortunate but we make no apology for OpenFOAM simply adopting good practice to ensure the mesh is valid; otherwise, the solution is flawed before the run has even begun.

By default OpenFOAM defines a mesh of arbitrary polyhedral cells in 3-D, bounded by arbitrary polygonal faces, *i.e.* the cells can have an unlimited number of faces where, for each face, there is no limit on the number of edges nor any restriction on its alignment. A mesh with this general structure is known in OpenFOAM as a polyMesh. It is described in further detail in section 2.3 of the Programmer's Guide, but it is sufficient to mention here that this type of mesh offers great freedom in mesh generation and manipulation in particular when the geometry of the domain is complex or changes over time. The price of absolute mesh generality is, however, that it can be difficult to convert meshes generated using conventional tools. The OpenFOAM library therefore provides cellShape tools to manage conventional mesh formats based on sets of pre-defined cell shapes.

6.1.1 Mesh specification and validity constraints

Before describing the OpenFOAM mesh format, polyMesh, and the cellShape tools, we will first set out the validity constraints used in OpenFOAM. The conditions that a mesh must satisfy are:

6.1.1.1 Points

A point is a location in 3-D space, defined by a vector in units of metres (m). The points are compiled into a list and each point is referred to by a label, which represents its position in the list, starting from zero. *The point list cannot contain two different points at an exactly identical position nor any point that is not part at least one face.*

6.1.1.2 Faces

A face is an ordered list of points, where a point is referred to by its label. The ordering of point labels in a face is such that each two neighbouring points are connected by an edge, *i.e.* you follow points as you travel around the circumference of the face. Faces are compiled into a list and each face is referred to by its label, representing its position in the list. The direction of the face normal vector is defined by the right-hand rule, *i.e.* looking towards a face, if the numbering of the points follows an anti-clockwise path, the normal vector points towards you, as shown in Figure 6.1.

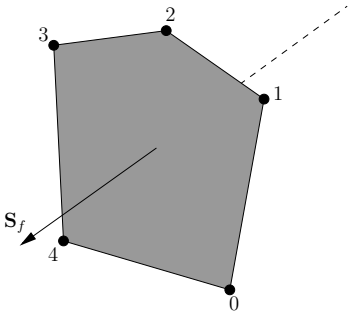


Figure 6.1: Face area vector from point numbering on the face

There are two types of face:

Internal faces Those faces that connect two cells (and it can never be more than two). For each internal face, the ordering of the point labels is such that the face normal points into the cell with the larger label, *i.e.* for cells 2 and 5, the normal points into 5;

Boundary faces Those belonging to one cell since they coincide with the boundary of the domain. A boundary face is therefore addressed by one cell(only) and a boundary patch. The ordering of the point labels is such that the face normal points outside of the computational domain.

Faces are generally expected to be convex; at the very least the face centre needs to be inside the face. Faces are allowed to be warped, i.e. not all points of the face need to be coplanar.

6.1.1.3 Cells

A cell is a list of faces in arbitrary order. Cells must have the properties listed below.

Contiguous The cells must completely cover the computational domain and are must not overlap one another.

Convex Every cell must be convex and its cell centre inside the cell.

Closed Every cell must be *closed*, both geometrically and topologically where:

- geometrical closedness requires that when all face area vectors are oriented to point outwards of the cell, their sum should equal the zero vector to machine accuracy;
- topological closedness requires that all the edges in a cell are used by exactly two faces of the cell in question.

Orthogonality For all internal faces of the mesh, we define the centre-to-centre vector as that connecting the centres of the 2 cells that it adjoins oriented from the the centre of the cell with smaller label to the centre of the cell with larger label. The orthogonality constraint requires that for each internal face, the angle between the face area vector, oriented as described above, and the centre-to-centre vector must always be less than 90°.

6.1.1.4 Boundary

A boundary is a list of patches, each of which is associated with a boundary condition. A patch is a list of face labels which clearly must contain only boundary faces and no internal faces. The boundary is required to be closed, *i.e.* the sum all boundary face area vectors equates to zero to machine tolerance.

6.1.2 The polyMesh description

The *constant* directory contains a full description of the case polyMesh in a subdirectory *polyMesh*. The polyMesh description is based around faces and, as already discussed, internal cells connect 2 cells and boundary faces address a cell and a boundary patch. Each face is therefore assigned an 'owner' cell and 'neighbour' cell so that the connectivity across a given face can simply be described by the owner and neighbour cell labels. In the case of boundaries, the connected cell is the owner and the neighbour is assigned the label '-1'. With this in mind, the I/O specification consists of the following files:

points a list of vectors describing the cell vertices, where the first vector in the list represents vertex 0, the second vector represents vertex 1, *etc.*;

faces a list of faces, each face being a list of indices to vertices in the points list, where again, the first entry in the list represents face 0, *etc.*;

owner a list of owner cell labels, the index of entry relating directly to the index of the face, so that the first entry in the list is the owner label for face 0, the second entry is the owner label for face 1, etc;

neighbour a list of neighbour cell labels;

boundary a list of patches, containing a dictionary entry for each patch, declared using the patch name, *e.g.*

```
movingWall
{
    type patch;
    physicalType wall; // (optional entry)
    nFaces 20;
    startFace 760;
}
```

The `startFace` is the index into the face list of the first face in the patch, and `nFaces` is the number of faces in the patch. The `physicalType` describes the physical type of boundary as described in section 6.2, and is used only by FoamX.

Note that if the user wishes to know how many cells are in their domain, there is a `note` *in the* `FoamFile` *header of the* owner *file that contains an entry for* `nCells`.

6.1.3 The cellShape tools

We shall describe the alternative cellShape tools that may be used particularly when converting some standard (simpler) mesh formats for the use with OpenFOAM library.

The vast majority of mesh generators and post-processing systems support only a fraction of the possible polyhedral cell shapes in existence. They define a mesh in terms of a limited set of 3D cell geometries, referred to as *cell shapes*. The OpenFOAM library contains definitions of these standard shapes, to enable a conversion of such a mesh into the polyMesh format described in the previous section.

The cellShape models supported by OpenFOAM are shown in Table 6.1. The shape is defined by the ordering of point labels in accordance with the numbering scheme contained in the shape model. The ordering schemes for points, faces and edges are shown in Table 6.1. The numbering of the points must not be such that the shape becomes twisted or degenerate into other geometries, *i.e.* the same point label cannot be used more that once is a single shape. Moreover it is unnecessary to use duplicate points in OpenFOAM since the available shapes in OpenFOAM cover the full set of degenerate hexahedra.

The cell description consists of two parts: the name of a cell model and the ordered list of labels. Thus, using the following list of points

```
8
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.5)
    (1 0 0.5)
    (1 1 0.5)
    (0 1 0.5)
)
```

A hexahedral cell would be written as:

```
(hex 8(0 1 2 3 4 5 6 7))
```

Here the hexahedral cell shape is declared using the keyword `hex`. Other shapes are described by the keywords listed in Table 6.1.

6.1.4 1- and 2-dimensional and axi-symmetric problems

OpenFOAM is designed as a code for 3-dimensional space and defines all meshes as such. However, 1- and 2- dimensional and axi-symmetric problems can be simulated in OpenFOAM by generating a mesh in 3 dimensions and applying special boundary conditions on any patch in the plane(s) normal to the direction(s) of interest. More specifically, 1- and 2-dimensional problems use the empty patch type and axi-symmetric problems use the wedge type. The use of both are described in section 6.2.2 and the generation of wedge geometries for axi-symmetric problems is discussed in section 6.3.3.

6.2 Boundaries

In this section we discuss the way in which boundaries are treated in OpenFOAM. The subject of boundaries is a little involved because their role in modelling is not simply that of a geometric entity but an integral part of the solution and numerics through boundary conditions or inter-boundary 'connections'. A discussion of boundaries sits uncomfortably between a discussion on meshes, fields, FoamX, discretisation, computational processing *etc.* Its placement in this Chapter on meshes is a choice of convenience.

We first need to consider that, for the purpose of applying boundary conditions, a boundary is generally broken up into a set of *patches*. One patch may include one or more enclosed areas of the boundary surface which do not necessarily need to be physically connected.

There are four attributes associated with a patch that are described below in their natural hierarchy and Figure 6.2 shows the names of different patch types introduced at each level of the hierarchy. The hierarchy described below is very similar, but not identical, to the class hierarchy used in the OpenFOAM library.

Base type The type of patch described purely in terms of geometry or a data 'communication link'.

Primitive type The base numerical patch condition assigned to a field variable on the patch.

Derived type A complex patch condition, derived from the primitive type, assigned to a field variable on the patch.

Physical type A type describing the boundary conditions in the physical world which may include specification of derived or primitive types on one or more fields, *e.g.* an inlet in fluid flow may be a fixedValue condition on **U** and fixedGradient condition on p.

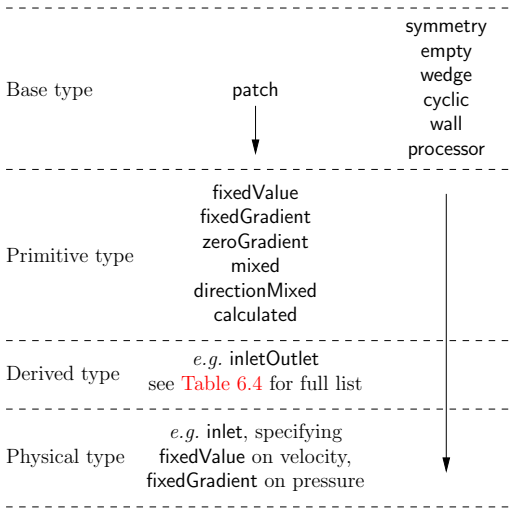


Figure 6.2: Patch attributes

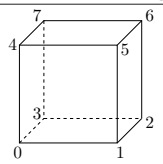
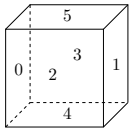
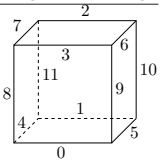
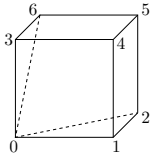
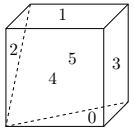
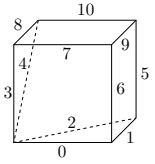
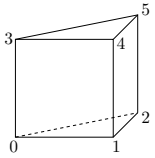
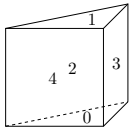
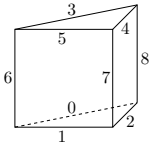
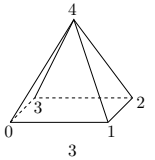
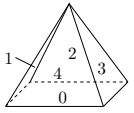
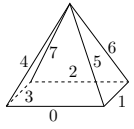
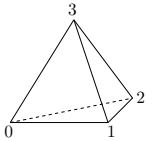
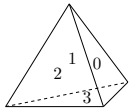
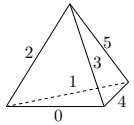
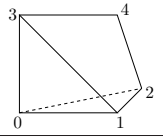
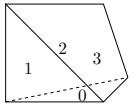
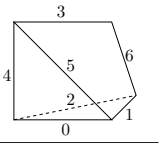
Cell type	Keyword	Vertex numbering	Face numbering	Edge numbering
Hexahedron	`hex`			
Wedge	`wedge`			
Prism	`prism`			
Pyramid	`pyr`			
Tetrahedron	`tet`			
Tet-wedge	`tetWedge`			

Table 6.1: Vertex, face and edge numbering for cellShapes.

6.2.1 Specification of patch types in OpenFOAM

The patch types are specified in the mesh and field files of a OpenFOAM case. More precisely:

- the base type is specified under the `type` keyword for each patch in the *boundary* file, located in the *constant/polyMesh* directory;
- the numerical patch type, be it a primitive or derived type, is specified under the `type` keyword for each patch in a field file.

The base type and numerical type are sufficient for the OpenFOAM case to run. However, there is one further *optional* entry:

- the physical type can be specified under the `physicalType` keyword for each patch in the *boundary* file.

This entry is generated by FoamX and corresponds to the configuration of physical patch types for individual OpenFOAM solvers. FoamX reads the entry when a case is opened but if it does not exist, it will need to be specified by the user.

An example *boundary* file is shown below for a sonicFoam case, followed by a pressure field file, *p*, for the same case:

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25  6
26  (
27  inlet
28  {
29      type patch;
30      physicalType supersonicInlet;
31      nFaces 50;
32      startFace 10325;
33  }
34
35  outlet
36  {
37      type patch;
38      physicalType pressureTransmissiveOutlet;
39      nFaces 40;
40      startFace 10375;
41  }
42
43  bottom
44  {
45      type symmetryPlane;
46      physicalType symmetryPlane;
47      nFaces 25;
48      startFace 10415;
49  }
50
51  top
52  {
53      type symmetryPlane;
54      physicalType symmetryPlane;
55      nFaces 125;
56      startFace 10440;
57  }
58
59  obstacle
60  {
61      type patch;
62      physicalType adiabaticWall;
63      nFaces 110;
64      startFace 10565;
65  }
66
67  defaultFaces
68  {
69      type empty;
```

```
70      nFaces 10500;
71      startFace 10675;
72  }
73  )
74
75  // ************************************************************************* //


23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25
26  dimensions      [1 -1 -2 0 0 0 0];
27
28  internalField   uniform 1;
29
30  boundaryField
31  {
32      inlet
33      {
34          type            fixedValue;
35          value           uniform 1;
36      }
37
38      outlet
39      {
40          type            waveTransmissive;
41          field           p;
42          phi             phi;
43          rho             rho;
44          psi             psi;
45          gamma           1.4;
46          fieldInf        1;
47          lInf            3;
48          value           uniform 1;
49      }
50
51      bottom
52      {
53          type            symmetryPlane;
54      }
55
56      top
57      {
58          type            symmetryPlane;
59      }
60
61      obstacle
62      {
63          type            zeroGradient;
64      }
65
66      defaultFaces
67      {
68          type            empty;
69      }
70  }
71
72
73  // ************************************************************************* //
```

The `type` in the boundary file is `patch` for all patches except those that patches that have some geometrical constraint applied to them, *i.e.* the symmetryPlane and empty patches. The *p* file includes primitive types applied to the `inlet` and `bottom` faces, and a more complex derived type applied to the `outlet`. Comparison of the two files shows that the base and numerical types are consistent where the base type is not a simple patch, *i.e.* for the symmetryPlane and empty patches.

The `physicalType` for each patch in the *boundary* file correspond to a named type in the FoamX configuration file for sonicFoam. For example, the `inlet` is inletFixedTemp which is configured to be a standard supersonic inlet condition with fixedValue on all fields: pressure `p`; velocity `U`; and temperature `T`. The type applied to `inlet` in the *p* file corresponds accordingly.

6.2.2 Base types

The base and geometric types are described below; the keywords used for specifying these types in OpenFOAM are summarised in Table 6.2.

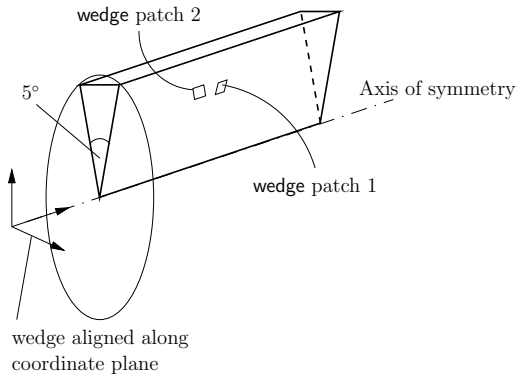


Figure 6.3: Axi-symmetric geometry using the wedge patch type.

Selection Key	Description
`patch`	generic patch
`symmetryPlane`	plane of symmetry
`empty`	front and back planes of 2D geometry
`wedge`	wedge front and back
`cyclic`	cyclic plane
`wall`	wall (used for wall functions in turbulent flows)
`processor`	inter-processor boundary

Table 6.2: Basic patch types.

patch The basic patch type for a patch condition that contains no geometric or topological information about the mesh (with the exception of wall), *e.g.* an inlet or an outlet.

wall For cases which require wall turbulence modelling, a wall must be specified with a wall patch type, so that the distance from the wall of the cell centres next to the wall are stored as part of the patch.

symmetryPlane For a symmetry plane.

empty While OpenFOAM always generates geometries in 3 dimensions, it can be instructed to solve in 2 (or 1) dimensions by specifying a special empty condition on each patch whose plane is normal to the 3rd (and 2nd) dimension for which no solution is required.

wedge For 2 dimensional axi-symmetric cases, *e.g.* a cylinder, the geometry is specified as a wedge of 5° angle and 1 cell thick running along the plane of symmetry, straddling

one of the coordinate planes, as shown in Figure 6.3. The axi-symmetric wedge planes must be specified as separate patches of wedge type. The details of generating wedge-shaped geometries using blockMesh are described in section 6.3.3.

cyclic Enables two patches to be treated as if they are physically connected; used for repeated geometries, *e.g.* heat exchanger tube bundles. A single cyclic patch splits the faces in its faceList into two, and links the two sets of faces as shown in Figure 6.4. Each face-face pair must be of the same area but the faces do not need to be of the same orientation.

processor If a code is being run in parallel, on a number of processors, then the mesh must be divided up so that each processor computes on roughly the same number of cells. The boundaries between the different parts of the mesh are called processor boundaries.

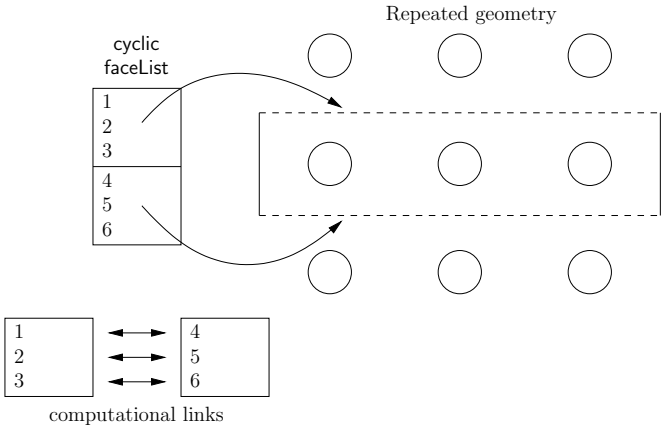


Figure 6.4: Repeated geometry using the cyclic patch type.

6.2.3 Primitive types

The primitive types are listed in Table 6.3.

6.2.4 Derived types

The derived types are listed in Table 6.4.

6.3 Mesh generation with the blockMesh utility

This section describes the mesh generation utility, blockMesh, supplied with OpenFOAM. The blockMesh utility creates parametric meshes with grading and curved edges.

Type	Description of condition for patch field ϕ	Data to specify
fixedValue	Value of ϕ is specified	`value`
fixedGradient	Normal gradient of ϕ is specified	`gradient`
zeroGradient	Normal gradient of ϕ is zero	—
calculated	Boundary field ϕ derived from other fields	—
mixed	Mixed fixedValue/ fixedGradient condition depending on the value in `valueFraction`	`refValue`, `refGradient`, `valueFraction`, `value`
directionMixed	A mixed condition normal to the patch with a fixedGradient condition tangential to the patch	`refValue`, `refGradient`, `valueFraction`, `value`

Table 6.3: Primitive patch field types.

Types derived from fixedValue		Data to specify
movingWallVelocity	Replaces the normal of the patch `value` so the flux across the patch is zero	`value`
pressureInletVelocity	When p is known at inlet, $\mathbf{U}$ is evaluated from the flux, normal to the patch	`value`
pressureDirectedInletVelocity	When p is known at inlet, $\mathbf{U}$ is calculated from the flux in the `inletDirection`	`value`, `inletDirection`
surfaceNormalFixedValue	Specifies a vector boundary condition, normal to the patch, by its magnitude; +ve for vectors pointing out of the domain	`value`
totalPressure	Total pressure $p_0 = p + \frac{1}{2}\rho\|\mathbf{U}\|^2$ is fixed; when $\mathbf{U}$ changes, p is adjusted accordingly	`p0`
turbulentInlet	Calculates a fluctuating variable based on a scale of a mean value	`referenceField`, `fluctuationScale`
Types derived from fixedGradient/zeroGradient		
fluxCorrectedVelocity	Calculates normal component of $\mathbf{U}$ at inlet from flux	`value`
wallBuoyantPressure	Sets fixedGradient pressure based on the atmospheric pressure gradient	—
Types derived from mixed		
inletOutlet	Switches $\mathbf{U}$ and p between fixedValue and zeroGradient depending on direction of $\mathbf{U}$	`inletValue`, `value`
outletInlet	Switches $\mathbf{U}$ and p between fixedValue and zeroGradient depending on direction of $\mathbf{U}$	`outletValue`, `value`
pressureInletOutletVelocity	Combination of pressureInletVelocity and inletOutlet	`value`
pressureDirected-InletOutletVelocity	Combination of pressureDirectedInletVelocity and inletOutlet	`value`, `inletDirection`
pressureTransmissive	Transmits supersonic pressure waves to surrounding pressure p_∞	`pInf`
supersonicFreeStream	Transmits oblique shocks to surroundings at p_∞, T_∞, $\mathbf{U}_\infty$	`pInf`, `TInf`, `UInf`
Other types		
slip	zeroGradient if ϕ is a scalar; if ϕ is a vector, normal component is fixedValue zero, tangential components are zeroGradient	—
partialSlip	Mixed zeroGradient/ slip condition depending on the `valueFraction`; = 1 for slip	`valueFraction`

Note: p is pressure, $\mathbf{U}$ is velocity

Table 6.4: Derived patch field types.

The mesh is generated from a dictionary file named *blockMeshDict* located in the *constant/polyMesh* directory of a case. blockMesh reads this dictionary, generates the mesh and writes out the mesh data to *points* and *faces*, *cells* and *boundary* files in the same directory.

The principle behind blockMesh is to decompose the domain geometry into a set of 1 or more three dimensional, hexahedral blocks. Edges of the blocks can be straight lines, arcs or splines. The mesh is ostensibly specified as a number of cells in each direction of the block, sufficient information for blockMesh to generate the mesh data.

Each block of the geometry is defined by 8 vertices, one at each corner of a hexahedron. The vertices are written in a list so that each vertex can be accessed using its label, remembering that OpenFOAM always uses the C++ convention that the first element of the list has label '0'. An example block is shown in Figure 6.5 with each vertex numbered according to the list. The edge connecting vertices 1 and 5 is curved to remind the reader that curved edges can be specified in blockMesh.

It is possible to generate blocks with less than 8 vertices by collapsing one or more pairs of vertices on top of each other, as described in section 6.3.3.

Each block has a local coordinate system (x_1, x_2, x_3) that must be right-handed, as defined in section 1.1 of the Programmer's Guide. The local coordinate system is defined by the order in which the vertices are presented in the block definition according to:

- the axis origin is the first entry in the block definition, vertex 0 in our example;
- the x_1 direction is described by moving from vertex 0 to vertex 1;
- the x_2 direction is described by moving from vertex 1 to vertex 2;
- vertices 0, 1, 2, 3 define the plane $x_3 = 0$;
- vertex 4 is found by moving from vertex 0 in the x_3 direction;
- vertices 5,6 and 7 are similarly found by moving in the x_3 direction from vertices 1,2 and 3 respectively.

Figure 6.5: A single block

Keyword	Description	Example/selection
convertToMeters	Scaling factor for the vertex coordinates	0.001 scales to mm
vertices	List of vertex coordinates	(0 0 0)
edges	Used to describe arc or spline edges	arc 1 4 (0.939 0.342 -0.5)
block	Ordered list of vertex labels and mesh size	hex (0 1 2 3 4 5 6 7) (10 10 1) simpleGrading (1.0 1.0 1.0)
patches	List of patches	symmetryPlane base ((0 1 2 3))

Table 6.5: Keywords used in *blockMeshDict*.

6.3.1 Writing a *blockMeshDict* file

The *blockMeshDict* file is a dictionary using keywords described in Table 6.5. The convertToMeters keyword specifies a scaling factor by which all vertex coordinates in the mesh description are multiplied. For example,

```
convertToMeters     0.001;
```

means that all coordinates are multiplied by 0.001, *i.e.* the values quoted in the *blockMeshDict* file are in mm.

6.3.1.1 The vertices

The vertices of the blocks of the mesh are given next as a standard list named vertices, *e.g.* for our example block in Figure 6.5, the vertices are:

```
vertices
(
    ( 0    0    0  )    // vertex number 0
    ( 1    0    0.1)    // vertex number 1
    ( 1.1  1    0.1)    // vertex number 2
    ( 0    1    0.1)    // vertex number 3
    (-0.1 -0.1  1  )    // vertex number 4
    ( 1.3  0    1.2)    // vertex number 5
    ( 1.4  1.1  1.3)    // vertex number 6
    ( 0    1    1.1)    // vertex number 7
);
```

6.3.1.2 The edges

Each edge joining 2 vertex points is assumed to be straight by default. However any edge may be specified to be curved by entries in a list named edges. The list is optional; if the geometry contains no curved edges, it may be omitted.

Keyword selection	Description	Additional entries
arc	Circular arc	Single interpolation point
simpleSpline	Spline curve	List of interpolation points
polyLine	Set of lines	List of interpolation points
polySpline	Set of splines	List of interpolation points
line	Straight line	—

Table 6.6: Edge types available in the *blockMeshDict* dictionary.

Each entry for a curved edge begins with a keyword specifying the type of curve from those listed in Table 6.6.

The keyword is then followed by the labels of the 2 vertices that the edge connects. Following that, interpolation points must be specified through which the edge passes. For a arc, a single interpolation point is required, which the circular arc will intersect. For simpleSpline, polyLine and polySpline, a list of interpolation points is required. The line edge is directly equivalent to the option executed by default, and requires no interpolation points. Note that there is no need to use the line edge but it is included for completeness. For our example block in Figure 6.5 we specify an arc edge connecting vertices 1 and 5 as follows through the interpolation point $(1.1, 0.0, 0.5)$:

```
edges
(
    arc 1 5 (1.1 0.0 0.5)
);
```

6.3.1.3 The blocks

The block definitions are contained in a list named blocks. Each block definition is a compound entry consisting of a list of vertex labels whose order is described in section 6.3, a vector giving the number of cells required in each direction, the type and list of cell expansion ratio in each direction.

Then the blocks are defined as follows:

```
blocks
(
    hex (0 1 2 3 4 5 6 7)    // vertex numbers
    (10 10 10)               // numbers of cells in each direction
    simpleGrading (1 2 3)    // cell expansion ratios
);
```

The definition of each block is as follows:

Vertex numbering The first entry is the is the shape identifier of the block, as defined in the *.OpenFOAM-1.4.1/cellModels* file. The shape is always hex since the blocks are always hexahedra. There follows a list of vertex numbers, ordered in the manner described on page U-152.

Number of cells The second entry gives the number of cells in each of the x_1 x_2 and x_3 directions for that block.

Cell expansion ratios The third entry gives the cell expansion ratios for each direction in the block. The expansion ratio enables the mesh to be graded, or refined, in specified directions. The ratio is that of the width of the end cell δ_e along one edge of a block to the width of the start cell δ_s along that edge, as shown in Figure 6.6. Each of the following keywords specify one of two types of grading specification available in blockMesh.

`simpleGrading` The simple description specifies uniform expansions in the local x_1, x_2 and x_3 directions respectively with only 3 expansion ratios, *e.g.*

```
simpleGrading (1 2 3)
```

`edgeGrading` The full cell expansion description gives a ratio for each edge of the block, numbered according to the scheme shown in Figure 6.5 with the arrows representing the direction 'from first cell...to last cell' *e.g.* something like

```
edgeGrading (1 1 1 1 2 2 2 2 3 3 3 3)
```

This means the ratio of cell widths along edges 0-3 is 1, along edges 4-7 is 2 and along 8-11 is 3 and is directly equivalent to the `simpleGrading` example given above.

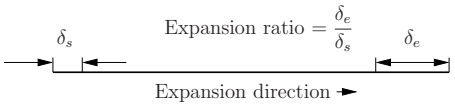


Figure 6.6: Mesh grading along a block edge

6.3.1.4 The `patches`

The patches of the mesh are given in a list named `patches`. Each patch in the list is a compound entry containing:

- the patch type, either a generic `patch` on which some boundary conditions are applied or a particular geometric condition, as listed in Table 6.2 and described in section 6.2.2;
- a list of block faces that make up the patch and whose name is the choice of the the user, although we recommend something that conveniently identifies the patch, *e.g.* quoteTextinlet; the name is used as an identifier for for for setting boundary conditions in the field data files.

blockMesh collects faces from any boundary patch that is omitted from the `patches` list and assigns them to a default patch named `defaultFaces` of type `empty`. This means that for a 2 dimensional geometry, the user has the option to omit block faces lying in the 2D plane, knowing that they will be collected into an `empty` patch as required.

Returning to the example block in Figure 6.5, if it has an inlet on the left face, an output on the right face and the four other faces are walls then the patches could be defined as follows:


```
patches                 // keyword
(
    patch               // patch type for patch 0
    inlet               // patch name
    (
        (0 4 7 3)       // block face in this patch
    )                   // end of 0th patch definition

    patch               // patch type for patch 1
    outlet              // arbitrary patch name
    (
        (1 2 6 5)
    )

    wall
    walls
    (
        (0 1 5 4)
        (0 3 2 1)
        (3 7 6 2)
        (4 5 6 7)
    )
);
```

Each block face is defined by a list of 4 vertex numbers. The order in which the vertices are given **must** be such that, looking from inside the block and starting with any vertex, the face must be traversed in a clockwise direction to define the other vertices.

6.3.2 Multiple blocks

A mesh can be created using more than 1 block. In such circumstances, the mesh is created as has been described in the preceeding text; the only additional issue is the connection between blocks, in which there are two distinct possibilities:

face matching the set of faces that comprise a patch from one block are exactly collocated with a set of faces patch that comprise a patch from another block;

face merging a group of faces from a patch from one block are connected to another group of faces from a patch from another block, to create a new set of internal faces connecting the two blocks.

To connect two blocks with **face matching**, the two patches that form the connection should simply be ignored from the `patches` list. blockMesh then identifies that the faces do not form an external boundary and combines each collocated pair into a single internal faces that connects cells from the two blocks.

The alternative, **face merging**, requires that the block patches to be merged are first defined in the `patches` list. Each pair of patches whose faces are to be merged must then

be included in an optional list named `mergePatchPairs`. The format of `mergePatchPairs` is:

```
mergePatchPairs
(
    ( <masterPatch> <slavePatch> ) // merge patch pair 0
    ( <masterPatch> <slavePatch> ) // merge patch pair 1
    ...
)
```

The pairs of patches are interpreted such that the first patch becomes the *master* and the second becomes the *slave*. The rules for merging are as follows:

- the faces of the master patch remain as originally defined, with all vertices in their original location;
- the faces of the slave patch are projected onto the master patch where there is some separation between slave and master patch;
- the location of any vertex of a slave face might be adjusted by blockMesh to eliminate any face edge that is shorter than a minimum tolerance;
- if patches overlap as shown in Figure 6.7, each face that does not merge remains as an external face of the original patch, on which boundary conditions must then be applied;
- if all the faces of a patch are merged, then the patch itself will contain no faces and is removed.

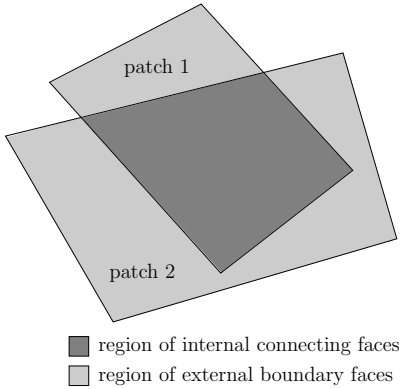


Figure 6.7: Merging overlapping patches

The consequence is that the original geometry of the slave patch will not necessarily be completely preserved during merging. Therefore in a case, say, where a cylindrical block

is being connected to a larger block, it would be wise to the assign the master patch to the cylinder, so that its cylindrical shape is correctly preserved. There are some additional recommendations to ensure successful merge procedures:

- in 2 dimensional geometries, the size of the cells in the third dimension, *i.e.* out of the 2D plane, should be similar to the width/height of cells in the 2D plane;
- it is inadvisable to merge a patch twice, *i.e.* include it twice in `mergePatchPairs`;
- where a patch to be merged shares a common edge with another patch to be merged, both should be declared as a master patch.

6.3.3 Creating blocks with fewer than 8 vertices

It is possible to collapse one or more pair(s) of vertices onto each other in order to create a block with fewer than 8 vertices. The most common example of collapsing vertices is when creating a 6-sided wedge shaped block for 2-dimensional axi-symmetric cases that use the wedge patch type described in section 6.2.2. The process is best illustrated by using a simplified version of our example block shown in Figure 6.8. Let us say we wished to create a wedge shaped block by collapsing vertex 7 onto 4 and 6 onto 5. This is simply done by exchanging the vertex number 7 by 4 and 6 by 5 respectively so that the block numbering would become:

```
hex (0 1 2 3 4 5 5 4)
```

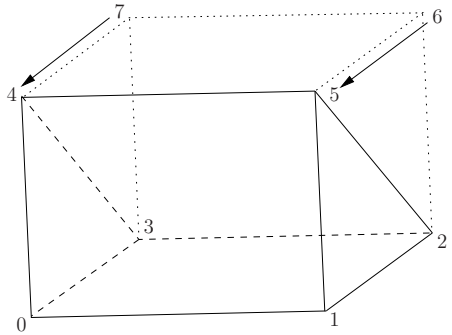


Figure 6.8: Creating a wedge shaped block with 6 vertices

The same applies to the patches with the main consideration that the block face containing the collapsed vertices, previously `(4 5 6 7)` now becomes `(4 5 5 4)`. This is a block face of zero area which creates a patch with no faces in the polyMesh, as the user can see in a *boundary* file for such a case. The patch should be specified as empty in the *blockMeshDict* and the boundary condition for any fields should consequently be empty also.

6.3.4 Running blockMesh

As described in section 3.3, the following can be executed at the command line to run blockMesh for a case in the <*case*> directory located at the path <*path*>:

```
blockMesh <path> <case>
```

The *blockMeshDict* file must exist in subdirectory *constant/polyMesh*.

6.4 Mesh conversion

The user can generate meshes using other packages and convert them into the format that OpenFOAM uses. The mesh conversion codes have the naming convention available mesh converters are:

fluentMeshToFoam reads a Fluent `.msh` mesh file, working for both 2-D and 3-D cases;

starToFoam reads STAR-CD PROSTAR mesh files.

gambitToFoam reads a GAMBIT `.neu` neutral file;

ideasToFoam reads an I-DEAS mesh written in ANSYS `.ans` format;

cfxToFoam reads a CFX mesh written in `.geo` format;

6.4.1 fluentMeshToFoam

Fluent writes mesh data to a single file with a *.msh* extension. The file must be written in ASCII format, which is not the default option in Fluent. It is possible to convert single-stream Fluent meshes, including the 2 dimensional geometries. In OpenFOAM, 2 dimensional geometries are currently treated by defining a mesh in 3 dimensions, where the front and back plane are defined as the empty boundary patch type. When reading a 2 dimensional Fluent mesh, the converter automatically extrudes the mesh in the third direction and adds the empty patch, naming it `frontAndBackPlanes`.

The following features should also be observed.

- The OpenFOAM converter will attempt to capture the Fluent boundary condition definition as much as possible; however, since there is no clear, direct correspondence between the OpenFOAM and Fluent boundary conditions, the user should check the boundary conditions, *e.g.* in FoamX, before running a case.
- Creation of axi-symmetric meshes from a 2 dimensional mesh is currently not supported but can be implemented on request.
- Multiple material meshes are not permitted. If multiple fluid materials exist, they will be converted into a single OpenFOAM mesh; if a solid region is detected, the converter will attempt to filter it out.
- Fluent allows the user to define a patch which is internal to the mesh, *i.e.* consists of the faces with cells on both sides. Such patches are not allowed in OpenFOAM and the converter will attempt to filter them out.

- There is currently no support for embedded interfaces and refinement trees.

The procedure of converting a Fluent *.msh* file is first to create a new OpenFOAM case, either from FoamX, as described in section 5.3.2, or by creating the necessary directories/files: the case directory containing a *controlDict* file in a *system* subdirectory. Then at a command prompt, or from within FoamX, the user should execute:

```
fluentMeshToFoam <root> <caseName> <meshFile>
```

where <`meshFile`> is the name of the *.msh* file, including the full or relative path.

6.4.2 starToFoam

This section describes how to convert a mesh generated on the STAR-CD code into a form that can be read by OpenFOAM mesh classes. The mesh can be generated by any of the packages supplied with STAR-CD, *i.e.* PROSTAR, SAMM, ProAM and their derivatives. The converter accepts any single-stream mesh including integral and arbitrary couple matching and all cell types are supported. The features that the converter does not support are:

- multi-stream mesh specification;
- baffles, *i.e.* zero-thickness walls inserted into the domain;
- partial boundaries, where an uncovered part of a couple match is considered to be a boundary face;
- sliding interfaces.

For multi-stream meshes, mesh conversion can be achieved by writing each individual stream as a separate mesh and reassemble them in OpenFOAM.

OpenFOAM adopts a policy of only accepting input meshes that conform to the fairly stringent validity criteria specified in section 6.1. It will simply not run using invalid meshes and cannot convert a mesh that is itself invalid. The following sections describe steps that must be taken when generating a mesh using a mesh generating package supplied with STAR-CD to ensure that it can be converted to OpenFOAM format. To avoid repetition in the remainder of the section, the mesh generation tools supplied with STAR-CD will be referred to by the collective name STAR-CD.

6.4.2.1 General advice on conversion

We strongly recommend that the user run the STAR-CD mesh checking tools before attempting a starToFoam conversion and, after conversion, the checkMesh utility should be run on the newly converted mesh. Alternatively, starToFoam may itself issue warnings containing PROSTAR commands that will enable the user to take a closer look at cells with problems. Problematic cells and matches should be checked and fixed before attempting to use the mesh with OpenFOAM. Remember that an invalid mesh will not run with OpenFOAM, but it may run in another environment that does not impose the validity criteria.

Some problems of tolerance matching can be overcome by the use of a matching tolerance in the converter. However, there is a limit to its effectiveness and an apparent need to increase the matching tolerance from its default level indicates that the original mesh suffers from inaccuracies.

6.4.2.2 Eliminating extraneous data

When mesh generation in is completed, remove any extraneous vertices and compress the cells boundary and vertex numbering, assuming that fluid cells have been created and all other cells are discarded. This is done with the following PROSTAR commands:

```
CSET NEWS FLUID
CSET INVE
```

The `CSET` should be empty. If this is not the case, examine the cells in `CSET` and adjust the model. If the cells are genuinely not desired, they can be removed using the PROSTAR command:

```
CDEL CSET
```

Similarly, vertices will need to be discarded as well:

```
CSET NEWS FLUID
VSET NEWS CSET
VSET INVE
```

Before discarding these unwanted vertices, the unwanted boundary faces have to be collected before purging:

```
CSET NEWS FLUID
VSET NEWS CSET
BSET NEWS VSET ALL
BSET INVE
```

If the `BSET` is not empty, the unwanted boundary faces can be deleted using:

```
BDEL BSET
```

At this time, the model should contain only the fluid cells and the supporting vertices, as well as the defined boundary faces. All boundary faces should be fully supported by the vertices of the cells, if this is not the case, carry on cleaning the geometry until everything is clean.

6.4.2.3 Removing default boundary conditions

By default, STAR-CD assigns wall boundaries to any boundary faces not explicitly associated with a boundary region. The remaining boundary faces are collected into a `default` boundary region, with the assigned boundary type `0`. OpenFOAM deliberately does not have a concept of a `default` boundary condition for undefined boundary faces since it invites human error, *e.g.* there is no means of checking that we meant to give all the unassociated faces the default condition.

Therefore **all** boundaries for each OpenFOAM mesh must be specified for a mesh to be successfully converted. The `default` boundary needs to be transformed into a real one using the procedure described below:

1. Plot the geometry with `Wire Surface` option.

2. Define an extra boundary region with the same parameters as the `default` region 0 and add all visible faces into the new region, say 10, by selecting a zone option in the boundary tool and drawing a polygon around the entire screen draw of the model. This can be done by issuing the following commands in PROSTAR:

   ```
   RDEF 10 WALL
   BZON 10 ALL
   ```

3. We shall remove all previously defined boundary types from the set. Go through the boundary regions:

   ```
   BSET NEWS REGI 1
   BSET NEWS REGI 2
   ... 3, 4, ...
   ```

   Collect the vertices associated with the boundary set and then the boundary faces associated with the vertices (there will be twice as many of them as in the original set).

   ```
   BSET NEWS REGI 1
   VSET NEWS BSET
   BSET NEWS VSET ALL
   BSET DELE REGI 1
   REPL
   ```

   This should give the faces of boundary Region 10 which have been defined on top of boundary Region 1. Delete them with `BDEL BSET`. Repeat these for all regions.

6.4.2.4 Renumbering the model

Renumber and check the model using the commands:

```
CSET NEW FLUID
CCOM CSET

VSET NEWS CSET
VSET INVE (Should be empty!)
VSET INVE
VCOM VSET

BSET NEWS VSET ALL
BSET INVE (Should be empty also!)
BSET INVE
BCOM BSET

CHECK ALL
GEOM
```

Internal PROSTAR checking is performed by the last two commands, which may reveal some other unforeseeable error(s). Also, take note of the scaling factor because PROSTAR only applies the factor for STAR-CD and not the geometry. If the factor is not 1, use the scalePoints utility in OpenFOAM.

6.4.2.5 Writing out the mesh data

Once the mesh is completed, place all the integral matches of the model into the couple type 1. All other types will be used to indicate arbitrary matches.

```
CPSET NEWS TYPE INTEGRAL
CPMOD CPSET 1
```

The components of the computational grid must then be written to their own files. This is done using PROSTAR for boundaries by issuing the command

```
BWRITE
```

by default, this writes to a *.23* file (versions prior to 3.0) or a *.bnd* file (versions 3.0 and higher). For cells, the command

```
CWRITE
```

outputs the cells to a *.14* or *.cel* file and for vertices, the command

```
VWRITE
```

outputs to file a *.15* or *.vrt* file. The current default setting writes the files in ASCII format. If couples are present, an additional couple file with the extension *.cpl* needs to be written out by typing:

```
CPWRITE
```

After outputting to the three files, exit PROSTAR or close the files. Look through the panels and take note of all STAR-CD sub-models, material and fluid properties used – the material properties and mathematical model will need to be set up using FoamX or by editing OpenFOAM dictionary files.

The procedure of converting the PROSTAR files is first to create a new OpenFOAM case by creating the necessary directories or from within FoamX. The PROSTAR files must be stored within the same directory and the user must change the file extensions: from *.23*, *.14* and *.15* (below STAR-CD version 3.0), or *.pcs*, *.cls* and *.vtx* (STAR-CD version 3.0 and above); to *.bnd*, *.cel* and *.vrt* respectively.

6.4.2.6 Problems with the *.vrt* file

The *.vrt* file is written in columns of data of specified width, rather than free format. A typical line of data might be as follows, giving a vertex number followed by the coordinates:

```
19422       -0.105988957     -0.413711881E-02 0.000000000E+00
```

If the ordinates are written in scientific notation and are negative, there may be no space between values, *e.g.*:

```
19423       -0.953953117E-01-0.338810333E-02 0.000000000E+00
```

The starToFoam converter reads the data using spaces to delimit the ordinate values and will therefore object when reading the previous example. Therefore, OpenFOAM includes a simple script, foamCorrectVrt to insert a space between values where necessary, *i.e.* it would convert the previous example to:

```
19423       -0.953953117E-01 -0.338810333E-02 0.000000000E+00
```

The foamCorrectVrt script should therefore be executed if necessary before running the starToFoam converter, by typing:

```
foamCorrectVrt <file>.vrt
```

6.4.2.7 Converting the mesh to OpenFOAM format

The translator utility starToFoam can now be run to create the boundaries, cells and points files necessary for a OpenFOAM run:

```
starToFoam <root> <caseName> <meshFilePrefix>
```

where <meshFilePrefix> is the name of the the prefix of the mesh files, including the full or relative path. After the utility has finished running, OpenFOAM boundary types should be specified in the usual way with FoamX or editing by hand.

6.4.3 gambitToFoam

GAMBIT writes mesh data to a single file with a *.neu* extension. The procedure of converting a GAMBIT *.neu* file is first to create a new OpenFOAM case, then at a command prompt, or from within FoamX, the user should execute:

```
gambitToFoam <root> <caseName> <meshFile>
```

where <root> and <caseName> are the root path and case name of the case and <meshFile> is the name of the *.neu* file, including the full or relative path.

The GAMBIT file format does not provide information about type of the boundary patch, *e.g.* wall, symmetry plane, cyclic. Therefore all the patches have been created as type patch. Please reset after mesh conversion as necessary.

6.4.4 ideasToFoam

OpenFOAM can convert a mesh generated by I-DEAS but written out in ANSYS format as a *.ans* file. The procedure of converting the *.ans* file is first to create a new OpenFOAM case, then at a command prompt, or from within FoamX, the user should execute:

```
ideasToFoam <root> <caseName> <meshFile>
```

where <root> and <caseName> are the root path and case name of the case and <meshFile> is the name of the *.ans* file, including the full or relative path.

6.4.5 cfxToFoam

CFX writes mesh data to a single file with a *.geo* extension. The mesh format in CFX is block-structured, *i.e.* the mesh is specified as a set of blocks with glueing information and the vertex locations. OpenFOAM will convert the mesh and capture the CFX boundary condition as best as possible. The 3 dimensional 'patch' definition in CFX, containing information about the porous, solid regions *etc.* is ignored with all regions being converted into a single OpenFOAM mesh. CFX supports the concept of a 'default' patch, where each external face without a defined boundary condition is treated as a wall. These faces are collected by the converter and put into a `defaultFaces` patch in the OpenFOAM mesh and given the type wall; of course, the patch type can be subsequently changed.

Like, OpenFOAM 2 dimensional geometries in CFX are created as 3 dimensional meshes of 1 cell thickness [**]. If a user wishes to run a 2 dimensional case on a mesh created by CFX, the boundary condition on the front and back planes should be set to empty; the user should ensure that the boundary conditions on all other faces in the plane of the calculation are set correctly. Currently there is no facility for creating an axi-symmetric geometry from a 2 dimensional CFX mesh.

The procedure of converting a CFX *.geo* file is first to create a new OpenFOAM case, then at a command prompt, or from within FoamX, the user should execute:

```
cfxToFoam <root> <caseName> <meshFile>
```

where <root> and <caseName> are the root path and case name of the case and <meshFile> is the name of the *.geo* file, including the full or relative path.

6.5 Mapping fields between different geometries

The mapFields utility maps one or more fields relating to a given geometry onto the corresponding fields for another geometry. It is completely generalised in so much as there does not need to be any similarity between the geometries to which the fields relate. However, for cases where the geometries are consistent, mapFields can be executed with a special option that simplifies the mapping process.

For our discussion of mapFields we need to define a few terms. First, we say that the data is mapped from the *source* to the *target*. The fields are deemed *consistent* if the geometry *and* boundary types, or conditions, of both source and target fields are identical. The field data that mapFields maps are those fields within the time directory specified by `startFrom`/`startTime` in the *controlDict* of the target case. The data is read from the

equivalent time directory of the source case and mapped onto the equivalent time directory of the target case.

6.5.1 Mapping consistent fields

A mapping of consistent fields is simply performed by executing mapFields on the (target) case using the `-consistent` command line option as follows:

```
mapFields <sourceRoot> <sourceCase> <root> <case> -consistent
```

6.5.2 Mapping inconsistent fields

When the fields are not consistent, as shown in Figure 6.9, mapFields requires a *mapFieldsDict* dictionary in the *system* directory of the target case. The following rules apply to the mapping:

- the field data is mapped from source to target wherever possible, *i.e.* in our example all the field data within the target geometry is mapped from the source, except those in the shaded region which remain unaltered;
- the patch field data is left unaltered unless specified otherwise in the *mapFieldsDict* dictionary.

The *mapFieldsDict* dictionary contain two lists that specify mapping of patch data. The first list is `patchMap` that specifies mapping of data between pairs of source and target patches that are geometrically coincident, as shown in Figure 6.9. The list contains each pair of names of source and target patch. The second list is `cuttingPatches` that contains names of target patches whose values are to be mapped from the source internal field through which the target patch cuts. In the situation where the target patch only cuts through part of the source internal field, *e.g.* bottom left target patch in our example, those values within the internal field are mapped and those outside remain unchanged. An example *mapFieldsDict* dictionary is shown below:

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25
26  patchMap
27  (
28      lid movingWall
29  );
30
31  cuttingPatches
32  (
33      fixedWalls
34  );
35
36
37  // ************************************************************************* //
```

```
mapFields <sourceRoot> <sourceCase> <root> <case>
```

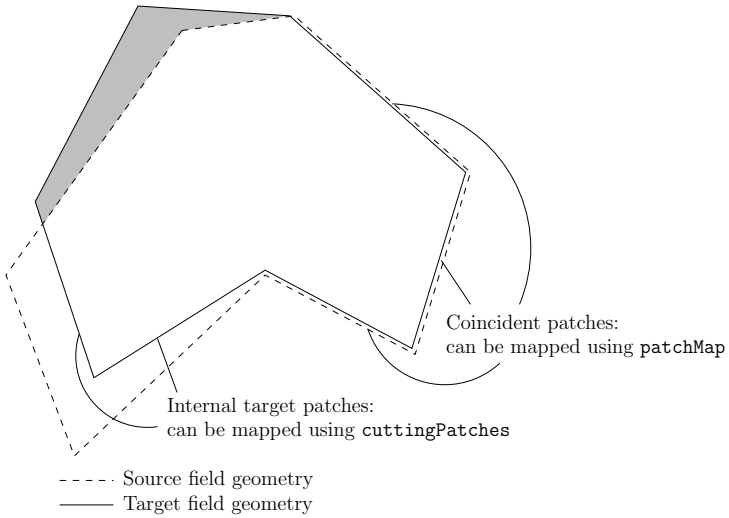


Figure 6.9: Mapping inconsistent fields

6.5.3 Mapping parallel cases

If either or both of the source and target cases are decomposed for running in parallel, additional options must be supplied when executing mapFields:

`-parallelSource` if the source case is decomposed for parallel running;

`-parallelTarget` if the target case is decomposed for parallel running.

Chapter 7

Post-processing

This chapter describes options for post-processing with OpenFOAM. OpenFOAM is supplied with a post-processing utility paraFoam that uses ParaView, an open source visualisation application described in section 7.1.

Other methods of post-processing using third party products are offered, including EnSight, AVS/Express and the post-processing supplied with Fluent.

7.1 paraFoam

The main post-processing tool provided with OpenFOAM is the a reader module to run with ParaView, an open-source, visualization application. The module is compiled into 2 libraries, PVFoamReader and vtkFoam, using version 2.4.4 of ParaView supplied with the OpenFOAM release. It is recommended that this version of ParaView is used, although it is possible that the latest binary release of the software will run adequately. Further details about ParaView can be found at http://www.paraview.org and further documentation is available at http://www.kitware.com/products/paraviewguide.html.

ParaView uses the Visualisation Toolkit (VTK) as its data processing and rendering engine and can therefore read any data in VTK format. OpenFOAM includes the foamToVTK utility to convert data from its native format to VTK format, which means that any VTK-based graphics tools can be used to post-process OpenFOAM cases. This provides an alternative means for using ParaView with OpenFOAM. For users who wish to experiment with advanced, parallel visualisation, we can recommend the free VisIt software, available at http://www.llnl.gov/visit.

In summary, we recommend the reader module for ParaView as the primary post-processing tool for OpenFOAM. Alternatively OpenFOAM data can be converted into VTK format to be read by ParaView or any other VTK -based graphics tools.

7.1.1 Overview of paraFoam

paraFoam is strictly a script that launches ParaView using the reader module supplied with OpenFOAM. It is executed like any of the OpenFOAM utilities with the root directory path and the case directory name as arguments:

```
paraFoam <root> <case>
```

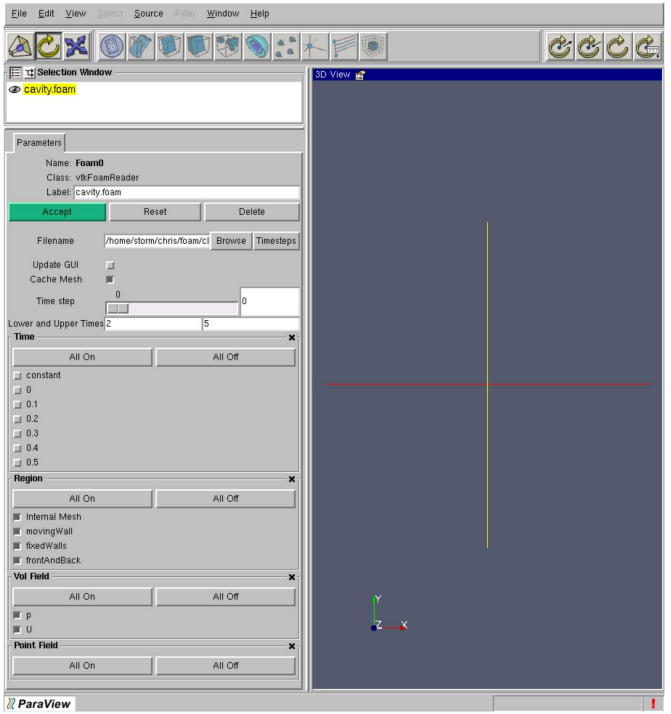



Figure 7.1: The paraFoam window

ParaView is launched and opens the window shown in Figure 7.1. The case is controlled from the left panel, which contains the following:

Selection Window lists the *modules* opened in ParaView, where the selected modules are highlighted in yellow and the graphics for the given module can be enabled/disabled by clicking the eye button alongside;

Parameters **panel** contains the input selections for the case, such as times, regions and fields;

Display **panel** controls the visual representation of the selected module, *e.g.* colours;

Information **panel** gives case statistics such as mesh geometry and size.

ParaView operates a tree-based structure in which data can be filtered from the top-level case module to create sets of sub-modules. For example, a contour plot of, say, pressure could be a sub-module of the case module which contains all the pressure data. The strength of ParaView is that the user can create a number of sub-modules and display whichever ones

they feel to create the desired image or animation. For example, they may add some solid geometry, mesh and velocity vectors, to a contour plot of pressure, switching any of the items on and off as necessary.

The general operation of the system is based on the user making a selection and then clicking the Accept button in the left panel. The additional buttons are: the Reset button which used to reset the GUI if necessary; and, the Delete button that will delete the active module.

7.1.2 The Parameters panel

The Parameters panel for the case module contains the settings for time step, regions and fields. The controls are described in Figure 7.2. As with any operation in paraFoam, the

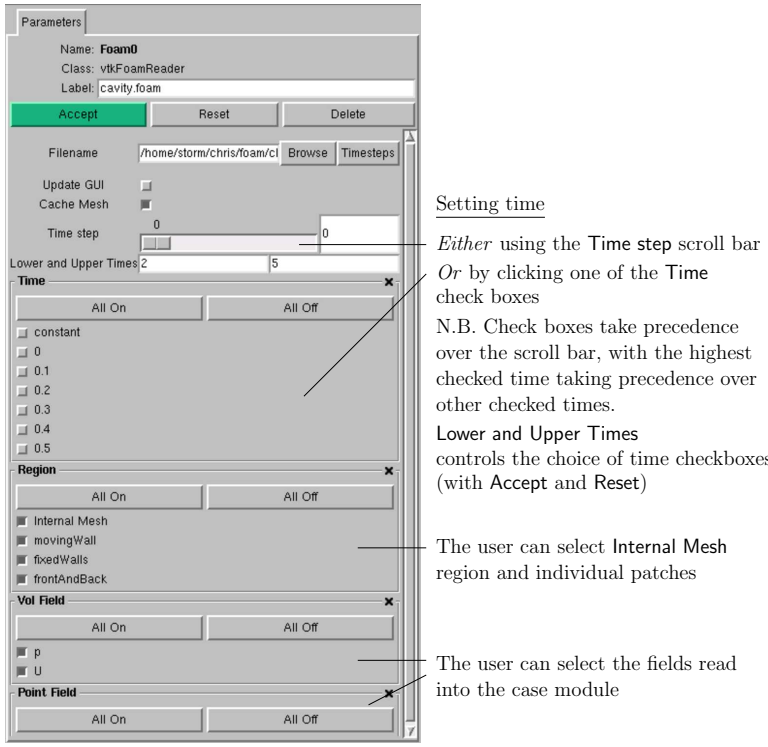


Figure 7.2: The Parameters panel for the case module

user must click Accept after making any changes to any selections. The Accept button is highlighted in green to alert the user if changes have been made but not accepted. This method of operation has the advantage of allowing the user to make a number of selections

before accepting them, which is particularly useful in large cases where data processing is best kept to a minimum.

7.1.3 The Display panel

The Display panel contains the settings for visualising the data for a given case module. The following points are particularly important:

- the data range is not automatically updated to the max/min limits of a field, so the user should take care **to select** Edit Color Map **and** Reset range at appropriate intervals, in particular after loading the initial case module;
- the style of legend, *e.g.* font, for the colour bar is controlled by in the Scalar Bar panel of the Edit Color Map window;
- the underlying mesh can be represented by selecting `Wireframe` in the `Representation` menu;
- the geometry, *e.g.* a mesh (if `Wireframe` is selected), can be visualised as a single colour by selecting `Property` from the `Color by` menu and specifying the Actor color;
- the image can be made translucent by editing the value in Opacity (1 = solid, 0 = invisible).

7.1.4 Manipulating the view

This section describes operations for setting and manipulating the view of objects in paraFoam.

7.1.4.1 3D view properties

There are first some settings controlling the `3D view Properties`, selected from the `View` menu, or alternatively by clicking the small 3D View toggle button at the top left of the image display window. This changes the left hand window to contain 3 panels, described below.

The General panel includes the following items which are **often worth setting at startup**:

- the background colour, where white is often a preferred choice for printed material;
- Use parallel projection which is the usual choice for CFD, especially for 2D cases;
- the level of detail (LOD) which controls the rendering of the image while it is being manipulated, *e.g.* translated, resized, rotated; lowering the levels set by the sliders, allows cases with large numbers of cells to be re-rendered quickly during manipulation;

The Annotate panel includes options for including annotations in the image. The Display Orientation Axes feature is particularly useful.

The Camera panel includes buttons of Standard Views, settings for Camera Controls and the options for detailed Camera Orientation. **Of great use** are the Stored Camera Position buttons. The user can store up to 6 camera positions by clicking on a given button using the right mouse button, for subsequent retrieval using the left mouse button.

To exit the 3D view Properties window, click the small 3D View toggle button at the top left of the image display window, or make an alternate selection, *e.g.* Source, from the View menu.

7.1.4.2 Rotation, translation and magnification

There are 3 grouped buttons at the top left of ParaView below the menu bar, that control image manipulation as shown in Figure 7.4. The user can select one of these buttons to set the overall mouse button configuration that control rotation, translation and magnification. For example, clicking the button that sets 3D motion properties, will cause the mouse buttons to operate as specified by 3D view Properties, discused in section 7.1.4.1. By default the 3D motion properties for the following mouse buttons selections are: left = rotate; middle = translate; right = zoom.

7.1.5 Contour plots

A contour plot is created by selecting Contour from the Filter menu at the top menu bar. The filter acts on a given module so that, if the module is the 3D case module itself, the contours will be a set of 2D surfaces that represent a constant value, *i.e.* isosurfaces. The Parameters panel for contours contains the list of constant values, that the user can edit, most conveniently by the Generate range of values window. The Input and Scalars menus present the module and fields, respectively, that are read by the filter.

7.1.5.1 Introducing a cutting plane

Very often a user will wish to create a contour plot across a plane rather than producing isosurfaces. To do so, the user must first use the Cut filter to create the cutting plane, on which the contours can be plotted. The Cut filter allows the user to specify a cutting Plane or Sphere in the Cut Function menu by a center and normal/radius respectively. The user can manipulate the cutting plane like any other using the mouse.

The user can then run the Contour filter on the cut plane to generate contour lines.

7.1.6 Vector plots

Vector plots are created using the Glyph filter. The filter reads an Input and offers a range of Glyphs for which the Arrow0 provide a clear vector plot images. In the Orient/Scale window, the most common options for Scale Mode are: Vector Magnitude, where the glyph length is proportional to the vector magnitude; and, Data Scaling Off where each glyph is the same length. An additional Scale Factor parameter controls the base length of the glyphs.

7.1.6.1 Plotting at cell centres

Vectors are by default plotted on cell vertices but, very often, we wish to plot data at cell centres. This is done by first applying the Cell Centers filter to the case module, and then applying the Glyph filter to the resulting cell centre data.

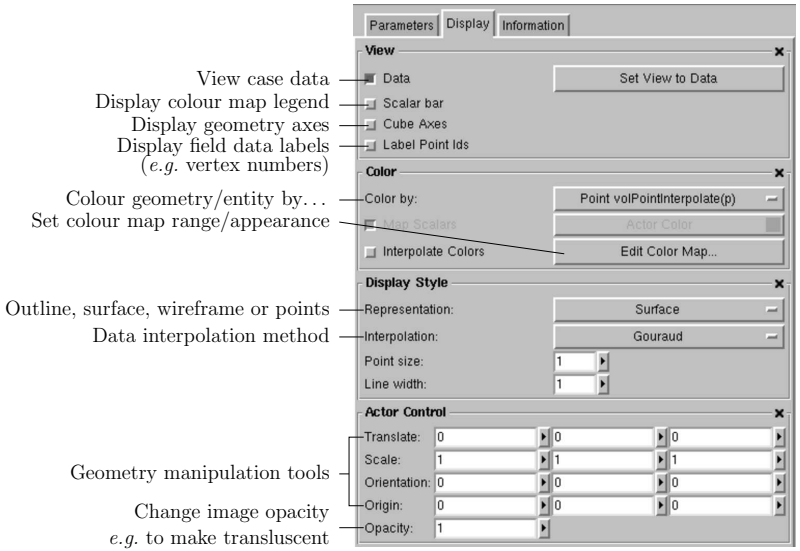


Figure 7.3: The Display panel

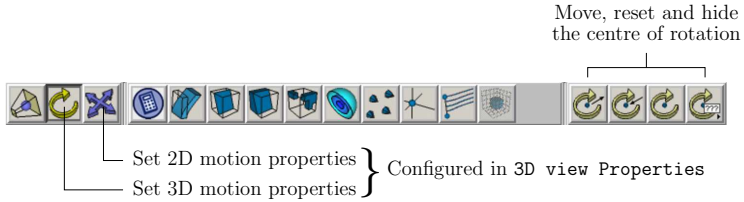


Figure 7.4: Buttons for image manipulation

7.1.7 Streamlines

Streamlines are created by first creating tracer lines using the `Stream Tracer` filter. The tracer Seed window specifies a distribution of tracer points over a `Line` or `Point Cloud`. The distance the tracer travels and the length of steps the tracer takes are specified in the text boxes below. The process of achieving desired tracer lines is largely one of trial and error in which the tracer lines obviously appear smoother as the step length is reduced but with the penalty of a longer calculation time.

Once the tracer lines have been created, the `Tubes` filter can be applied to produce high quality images. The tubes follow each tracer line and are not stricly cylidrical but have a fixed number of sides and given radius. When the number of sides is set above, say, 10, the tubes do however appear cylindrical, but again this adds a computational cost.

7.2 Post-processing with Fluent

It is possible to use Fluent as a post-processor for the cases run in OpenFOAM. Two converters are supplied for the purpose: foamMeshToFluent which converts the OpenFOAM mesh into Fluent format and writes it out as a *.msh* file; and, foamDataToFluent converts the OpenFOAM results data into a *.dat* file readable by Fluent. foamMeshToFluent is used by typing the following:

```
foamMeshToFluent <root> <caseName>
```

The resulting mesh is written out in a *fluentInterface* subdirectory of the case directory, *i.e.* *<caseName>/fluentInterface/<caseName>.msh*

foamDataToFluent converts the OpenFOAM data results into the Fluent format. The conversion is controlled by two files. First, the *controlDict* dictionary specifies `startTime`, giving the set of results to be converted. If you want to convert the latest result, `startFrom` can be set to `latestTime`. The second file which specifies the translation is the *foamDataToFluentDict* dictionary, located in the *constant* directory. An example *foamDataToFluentDict* dictionary is given below:

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.4                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "";
    instance        "";
    local           "";

    class           dictionary;
    object          foamDataToFluentDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


p               1;

U               2;

T               3;

h               4;

k               5;

epsilon         6;

gamma           150;


// ************************************************************************* //
```

The dictionary contains entries of the form

```
<fieldName> <fluentUnitNumber>
```

The `<fluentUnitNumber>` is a label used by the Fluent post-processor that only recognises a fixed set of fields. The basic set of `<fluentUnitNumber>` numbers are quoted in Table 7.1. The dictionary must contain all the entries the user requires to post-process, *e.g.* in our example we have entries for pressure `p` and velocity `U`. If the user runs foamDataToFluent through FoamX, he/she will be provided with a dictionary with a list of default entries described in Table 7.1. To run foamDataToFluent from the command line the user should type

Fluent name	Unit number	Common OpenFOAM name
PRESSURE	1	p
MOMENTUM	2	U
TEMPERATURE	3	T
ENTHALPY	4	h
TKE	5	k
TED	6	epsilon
SPECIES	7	—
G	8	—
XF_RF_DATA_VOF	150	gamma
TOTAL_PRESSURE	192	—
TOTAL_TEMPERATURE	193	—

Table 7.1: Fluent unit numbers for post-processing.

```
foamDataToFluent <root> <caseName>
```

To view the results using Fluent, go to the *fluentInterface* subdirectory of the case directory and start a 3 dimensional version of Fluent with

```
fluent 3d
```

The mesh and data files can be loaded in and the results visualised. The mesh is read by selecting `Read Case` from the `File` menu. Support items should be selected to read certain data types, *e.g.* to read turbulence data for `k` and `epsilon`, the user would select `k-epsilon`

from the `Define->Models->Viscous` menu. The data can then be read by selecting `Read Data` from the `File` menu.

A note of caution: users MUST NOT try to use an original Fluent mesh file that has been converted to OpenFOAM format in conjunction with the OpenFOAM solution that has been converted to Fluent format since the alignment of zone numbering cannot be guaranteed.

7.3 Post-processing with Fieldview

OpenFOAM offers the capability for post-processing OpenFOAM cases with Fieldview. The method involves running a post-processing utility foamToFieldview to convert case data from OpenFOAM to Fieldview *.uns* file format. For a given case, foamToFieldview is executed like any normal application, either from FoamX, or from a terminal window by a command of the form:

```
foamToFieldview <root> <case>
```

foamToFieldview creates a directory named *Fieldview* in the case directory, *deleting any existing Fieldview directory in the process*. By default the converter reads the data in all time directories and writes into a set of files of the form <*case*>*_nn.uns*, where *nn* is an incremental counter starting from 1 for the first time directory, 2 for the second and so on. The user may specify the conversion of a single time directory with the option `-time <time>`, where `<time>` is a time in general, scientific or fixed format.

Fieldview provides certain functions that require information about boundary conditions, *e.g.* drawing streamlines that uses information about wall boundaries. The converter tries, wherever possible, to include this information in the converted files by default. The user can disable the inclusion of this information by using the `-noWall` option in the execution command.

The data files for Fieldview have the *.uns* extension as mentioned already. If the original OpenFOAM case includes a dot '.', Fieldview may have problems interpreting a set of data files as a single case with multiple time steps.

7.4 Post-processing with EnSight

OpenFOAM offers the capability for post-processing OpenFOAM cases with EnSight, with a choice of 2 options:

- converting the OpenFOAM data to EnSight format with the foamToEnsight utility;
- reading the OpenFOAM data directly into EnSight using the ensight74FoamExec module.

7.4.1 Converting data to EnSight format

The foamToEnsight utility converts data from OpenFOAM to EnSight file format. For a given case, foamToEnsight is executed like any normal application, either from FoamX, or from a terminal window by a command of the form:


```
foamToEnsight <root> <case>
```

foamToEnsight creates a directory named *Ensight* in the case directory, *deleting any existing Ensight directory in the process*. The converter reads the data in all time directories and writes into a case file and a set of data files. The case file is named *EnSight_Case* and contains details of the data file names. Each data file has a name of the form *EnSight_nn.ext*, where *nn* is an incremental counter starting from 1 for the first time directory, 2 for the second and so on and *ext* is a file extension of the name of the field that the data refers to, as described in the case file, *e.g.* `T` for temperature, `mesh` for the mesh. Once converted, the data can be read into EnSight by the normal means:

1. from the EnSight GUI, the user should select `Data (Reader)` from the `File` menu;
2. the appropriate *EnSight_Case* file should be highlighted in the Files box;
3. the `Format` selector should be set to `Case`, the EnSight default setting;
4. the user should click (Set) Case and Okay.

7.4.2 The ensight74FoamExec reader module

EnSight provides the capability of using a user-defined module to read data from a format other than the standard EnSight format. OpenFOAM includes its own reader module ensight74FoamExec that is compiled into a library named libuserd-foam. It is this library that EnSight needs to use which means that it must be able to locate it on the filing system as described in the following section.

7.4.2.1 Configuration of EnSight for the reader module

In order to run the EnSight reader, it is necessary to set some environment variables correctly. The settings are made in the *bashrc* (or *cshrc*) file in the user's *$HOME/.OpenFOAM-1.4.1/apps/ensightFoam* directory. The environment variables associated with EnSight are prefixed by $CEI_ or $ENSIGHT7_ and listed in Table 7.2. With a standard user setup, only $CEI_HOME may need to be set manually, to the path of the EnSight installation.

Environment variable	Description and options
$CEI_HOME	Path where EnSight is installed, eg */usr/local/ensight*, added to the system path by default
$CEI_ARCH	Machine architecture, from a choice of names corresponding to the machine directory names in *$CEI_HOME/ensight74/machines*; default settings include `linux_2.4` and `sgi_6.5_n32`
$ENSIGHT7_READER	Path that EnSight searches for the user defined libuserd-foam reader library, set by default to `$FOAM_LIBBIN`
$ENSIGHT7_INPUT	Set by default to `dummy`

Table 7.2: Environment variable settings for EnSight.

7.4.2.2 Using the reader module

The principal difficulty in using the EnSight reader lies in the fact that EnSight expects that a case to be defined by the contents of a particular file, rather than a directory as it is in OpenFOAM. Therefore in following the instructions for the using the reader below, the user should pay particular attention to the details of case selection, since EnSight does not permit selection of a directory name.

1. from the EnSight GUI, the user should select `Data (Reader)` from the `File` menu;
2. The user should now be able to select the `OpenFOAM` from the `Format` menu; if not, there is a problem with the configuration described above.
3. The user should find their case directory from the File Selection window, highlight one of top 2 entries in the Directories box ending in `/.` or `/..` and click (Set) Geometry.
4. The path field should now contain an entry of the form <root>/<case>, where <root> and <case> represent the root path and case name, using the standard OpenFOAM terminology. The (Set) Geometry text box should contain a '/'.
5. The user may now click Okay and EnSight will begin reading the data.
6. When the data is read, a new Data Part Loader window will appear, asking which part(s) are to be read. The user should select Load all.
7. When the mesh is displayed in the EnSight window the user should close the Data Part Loader window, since some features of EnSight will not work with this window open.

7.5 Sampling data for plotting graphs

OpenFOAM provides the sample utility to sample field data for plotting on graphs. The sampling locations are specified for a case through a *sampleDict* dictionary in the case *system* directory. The data can be written in a range of formats including well-known graphing packages such as Grace/xmgr, gnuplot and jPlot.

The *sampleDict* dictionary can be generated with the standard set of keyword entries using FoamX, or by copying an example *sampleDict*, shown below, from the plateHole tutorial case in the *$FOAM_TUTORIALS/solidDisplacementFoam* directory. The dictionary contains the entries as follows and summarised in Table 7.3.

```
23  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
24
25
26  interpolationScheme cellPoint;
27
28  writeFormat     raw;
29
30  sampleSets
31  (
32      uniform
33      {
34          name            leftPatch;
35          axis            y;
36          start           (0 0.5 0.25);
37          end             (0 2 0.25);
38          nPoints         100;
39      }
40  );
41
```



```
42  fields
43  (
44      sigmaxx
45  );
46
47
48  // ************************************************************************* //
```

Keyword	Options	Description
`interpolation-`	`cell`	Cell-centre value assumed constant over cell
`Scheme`	`cellPoint`	Linear weighted interpolation using cell values
	`cellPointFace`	Mixed linear weighted / cell-face interpolation
`outputFormat`	`raw`	Raw ASCII data in columns
	`gnuplot`	Data in gnuplot format
	`xmgr`	Data in Grace/xmgr format
	`jplot`	Data in jPlot format
`fields`	List of fields to be sampled, *e.g.* for velocity `U`:	
	`U`	Writes all components of **U**
	`U.component(0)`	Writes component 0, *i.e.* $\mathbf{U}_x$
	`U.component(1)`	Writes component 1, *i.e.* $\mathbf{U}_y$
	`mag(U)`	Writes magnitude, *i.e.* $\|\mathbf{U}\|$
`sampleSets`	List of sample set subdictionaries — see Table 7.4	

Table 7.3: keyword entries for *sampleDict*.

`interpolationScheme` the scheme of data interpolation;

`writeFormat` the format of data output;

`fields` the fields to be sampled;

`sampleSets` the locations within the domain that the fields are sampled.

The `interpolationScheme` includes `cellPoint` and `cellPointFace` options in which each polyhedral cell is decomposed into tetrahedra and the sample values are interpolated from values at the tetrahedra vertices. With `cellPoint`, the tetrahedra vertices include the polyhedron cell centre and 3 face vertices. The vertex coincident with the cell centre inherits the cell centre field value and the other vertices take values interpolated from cell centres. With `cellPointFace`, one of the tetrahedra vertices is also coincident with a face centre, which inherits field values by conventional interpolation schemes using values at the centres of cells that the face intersects.

The `writeFormat` includes a raw data format and formats for gnuplot, Grace/xmgr and jPlot graph drawing packages. The data are written into a *samples* directory within the case directory. The directory is split into a set of time directories and the data files are contained therein. Each data file is given a name containing the field name, the sample set name, and an extension relating to the output format, including *.xy* for raw data, *.agr* for Grace/xmgr and *.dat* for jPlot. The gnuplot format has the data in raw form with an

additional commands file, with *.gplt* extension, for generating the graph. *Note that any existing samples directory is deleted when* sample *is run.*

The `fields` list contains the fields that the user wishes to sample. The sample utility can parse the following restricted set of functions to enable the user to manipulate vector and tensor fields, *e.g.* for `U`:

`U.component(`n`)` writes the nth component of the vector/tensor, $n = 0, 1 \ldots$;

`mag(U)` writes the magnitude of the vector/tensor.

The `sampleSets` list contains subdictionaries of locations where the data is to be sampled. The subdictionary is named according to the type of sampling as listed in Table 7.4. The dictionary contains a set of entries, also listed in Table 7.4, that describes the locations where the data is to be sampled. For example, a `uniform` sampling provides a uniform distribution of `nPoints` sample locations along a line specified by a `start` and `end` point. All sample sets are also given: a `name`; and, means of specifying the length ordinate on a graph by the `axis` keyword.

Known bug: At present, the user may experience problems if a sample line coincides with a set of cell vertices, edges or faces; the user should set the sample line accordingly.

		Required entries					
Sampling type	Sample locations	`name`	`axis`	`start`	`end`	`nPoints`	`points`
`uniform`	Uniformly distributed points on a line	•	•	•	•	•	
`face`	Intersection of specified line and cell faces	•	•	•	•		
`midPoint`	Midpoint between line-face intersections	•	•	•	•		
`midPointAndFace`	Combination of `midPoint` and `face`	•	•	•	•		
`curve`	Specified points, tracked along a curve	•	•				•
`cloud`	Specified points	•	•				•

Entries	Description	Options	
`name`	Name of the sample set	*e.g.* `inletProfile`	
`axis`	Output of sample location	`x`	x ordinate
		`y`	y ordinate
		`z`	z ordinate
		`xyz`	xyz coordinates
		`distance`	distance from point 0
`start`	Start point of sample line	*e.g.* `(0.0 0.0 0.0)`	
`end`	End point of sample line	*e.g.* `(0.0 2.0 0.0)`	
`nPoints`	Number of sampling points	*e.g.* 200	
`points`	List of sampling points		

Table 7.4: Entries within `sampleSets` subdictionaries.

7.6 Monitoring and managing jobs

This section is concerned primarily with successful running of OpenFOAM jobs and extends on the basic execution of solvers described in section 3.3. When a solver is executed, it reports the status of equation solution to standard output, *i.e.* the screen, if the `level` debug switch is set to 1 or 2 (default) in *DebugSwitches* in the *$HOME/.OpenFOAM-1.4.1/controlDict* file. An example from the beginning of the solution of the cavity tutorial is shown below where it can be seen that, for each equation that is solved, a report line is written with the solver name, the variable that is solved, its initial and final residuals and number of iterations.

```
    Starting time loop

Time = 0.005

Max Courant Number = 0
BICCG:  Solving for Ux, Initial residual = 1, Final residual = 2.96338e-06, No Iterations 8
ICCG:  Solving for p, Initial residual = 1, Final residual = 4.9336e-07, No Iterations 35
time step continuity errors : sum local = 3.29376e-09, global = -6.41065e-20, cumulative = -6.41065e-20
ICCG:  Solving for p, Initial residual = 0.47484, Final residual = 5.41068e-07, No Iterations 34
time step continuity errors : sum local = 6.60947e-09, global = -6.22619e-19, cumulative = -6.86725e-19
ExecutionTime = 0.14 s


Time = 0.01

Max Courant Number = 0.585722
BICCG:  Solving for Ux, Initial residual = 0.148584, Final residual = 7.15711e-06, No Iterations 6
BICCG:  Solving for Uy, Initial residual = 0.256618, Final residual = 8.94127e-06, No Iterations 6
ICCG:  Solving for p, Initial residual = 0.37146, Final residual = 6.67464e-07, No Iterations 33
time step continuity errors : sum local = 6.34431e-09, global = 1.20603e-19, cumulative = -5.66122e-19
ICCG:  Solving for p, Initial residual = 0.271556, Final residual = 3.69316e-07, No Iterations 33
time step continuity errors : sum local = 3.96176e-09, global = 6.9814e-20, cumulative = -4.96308e-19
ExecutionTime = 0.16 s


Time = 0.015

Max Courant Number = 0.758267
BICCG:  Solving for Ux, Initial residual = 0.0448679, Final residual = 2.42301e-06, No Iterations 6
BICCG:  Solving for Uy, Initial residual = 0.0782042, Final residual = 1.47009e-06, No Iterations 7
ICCG:  Solving for p, Initial residual = 0.107474, Final residual = 4.8362e-07, No Iterations 32
time step continuity errors : sum local = 3.99028e-09, global = -5.69762e-19, cumulative = -1.06607e-18
ICCG:  Solving for p, Initial residual = 0.0806771, Final residual = 9.47171e-07, No Iterations 31
time step continuity errors : sum local = 7.92176e-09, global = 1.07533e-19, cumulative = -9.58537e-19
ExecutionTime = 0.19 s
```

7.6.1 The foamJob script for running jobs

The user may be happy to monitor the residuals, iterations, Courant number *etc.* as report data passes across the screen. Alternatively, the user can redirect the report to a log file which will improve the speed of the computation. The foamJob script provides useful options for this purpose with the following executing the specified <`solver`> as a background process and redirecting the output to a file named <*root*>/<*case*>/*log*:

```
foamJob <solver> <root> <case>
```

For further options the user should execute `foamJob -h`. The user may monitor the *log* file whenever they wish, using the UNIX tail command, typically with the `-f` 'follow' option which appends the new data as the *log* file grows:

```
tail -f <root>/<case>/log
```

7.6.2 The foamLog script for monitoring jobs

There are limitations to monitoring a job by reading the log file, in particular it is difficult to extract trends over a long period of time. The foamLog script is therefore provided to extract data of residuals, iterations, Courant number *etc.* from a log file and present it in a set of files that can be plotted graphically. The script is executed by:

```
foamLog <root> <case> <logFile>
```

The files are stored in a subdirectory of the case directory named *logs*. Each file has the name <*var*>_<*subIter*> where <*var*> is the name of the variable specified in the log file and <*subIter*> is the iteration number within the time step. Those variables that are solved for, the initial residual takes the variable name <*var*> and final residual takes <*var*>*FinalRes*. By default, the files are presented in two-column format of time and the extracted values.

For example, in the cavity tutorial we may wish to observe the initial residual of the Ux equation to see whether the solution is converging to a steady-state. In that case, we would plot the data from the *logs/Ux_0* file as shown in Figure 7.5. It can be seen here that the residual falls monotonically until it reaches the convergence tolerance of 10^{-5}.

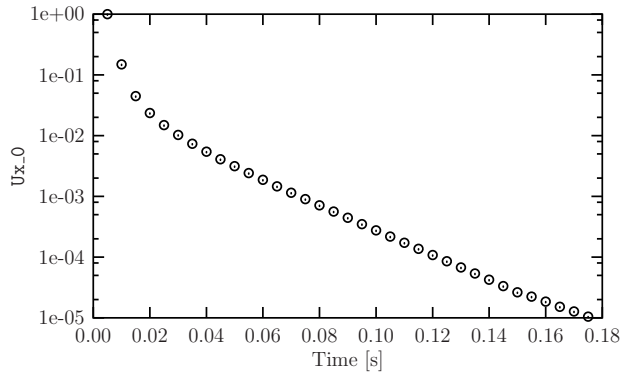


Figure 7.5: Initial residual of Ux in the cavity tutorial

foamLog generates files for everything it feasibly can from the *log* file. In the cavity tutorial example, this includes:

- the Courant number, Courant_0;
- Ux equation initial and final residuals, Ux_0 and UxFinalRes_0, and iterations, UxIters_0 (and equivalent Uy data);

- cumulative, global and local continuity errors after each of the 2 p equations, contCumulative_0, contGlobal_0, contLocal_0 and contCumulative_1, contGlobal_1, contLocal_1;
- residuals and iterations from the the 2 p equations p_0, pFinalRes_0, pIters_0 and p_1, pFinalRes_1, pIters_1;
- and execution time, executionTime.

Chapter 8

Models and physical properties

OpenFOAM includes a large range of solvers each designed for a specific class of problem. The equations and algorithms differ from one solver to another so that the selection of a solver involves the user making some initial choices on the modelling for their particular case. The choice of solver typically involves scanning through their descriptions in Table 3.5 to find the one suitable for the case. It ultimately determines many of the parameters and physical properties required to define the case but leaves the user with some modelling options that can be specified at runtime through the entries in dictionary files in the *constant* directory of a case. This chapter deals with many of the more common models and associated properties that may be specified at runtime.

8.1 Thermophysical models

Thermophysical models are concerned with the energy, heat and physical properties.

The *thermophysicalProperties* dictionary is read by any solver that uses the thermophysical model library. A thermophysical model is constructed in OpenFOAM as a pressure-temperature $p-T$ system from which other properties are computed. There is one compulsory dictionary entry called `thermoType` which specifies the complete thermophysical model that is used in the simulation. The thermophysical modelling starts with a layer that defines the basic equation of state and then adds more layers of modelling that derive properties from the previous layer(s). The naming of the `thermoType` reflects these multiple layers of modelling as listed in Table 8.1.

Equation of State — equationOfState	
perfectGas	Perfect gas equation of state
Basic thermophysical properties — thermo	
hConstThermo	Constant specific heat c_p model with evaluation of enthalpy h and entropy s
janafThermo	c_p evaluated by a function with coefficients from JANAF thermodynamic tables, from which h, s are evaluated
Derived thermophysical properties — specieThermo	
specieThermo	Thermophysical properties of species, derived from c_p, h and/or s

Continued on next page

Continued from previous page

Transport properties — transport	
constTransport	Constant transport properties
sutherlandTransport	Sutherland's formula for temperature-dependent transport properties
Mixture properties — mixture	
pureMixture	General thermophysical model calculation for passive gas mixtures
homogeneousMixture	Combustion mixture based on normalised fuel mass fraction b
inhomogeneousMixture	Combustion mixture based on b and total fuel mass fraction f_t
veryInhomogeneousMixture	Combustion mixture based on b, f_t and unburnt fuel mass fraction f_u
dieselMixture	Combustion mixture based on f_t and f_u
multiComponentMixture	Combustion mixture based on multiple components [**]
chemkinMixture	Combustion mixture using CHEMKIN thermodynamics and reaction schemes database files
Thermophysical model — thermoModel	
hThermo	General thermophysical model calculation based on enthalpy h
hMixtureThermo	Calculates enthalpy for combustion mixture
hhuMixtureThermo	Calculates enthalpy for unburnt gas and combustion mixture

Table 8.1: Layers of thermophysical modelling.

The `thermoType` entry takes the form:

```
thermoModel<mixture<transport<specieThermo<thermo<equationOfState>>>>>
```

so that the following is an example entry for `thermoType`:

```
hThermo<pureMixture<constTransport<specieThermo<hConstThermo<perfectGas>>>>>
```

8.1.1 Thermophysical property data

The basic thermophysical properties are specified for each species from input data. The data is specified using a compound entry with the following format for a specie accessed through the keyword `mixture`:

```
mixture <specieCoeffs> <thermoCoeffs> <transportCoeffs>
```

Description	Entry
String name	*e.g.* `mixture`
Number of moles of this specie	n_{moles}
Molecular weight	W (kg/kmol)

Table 8.2: Specie coefficients.

The specie coefficients <`specieCoeffs`> contains the entries listed in Table 8.2 in the order that they are specified in input.

The thermodynamic coefficients <`thermoCoeffs`> are ostensibly concerned with evaluating the specific heat c_p from which other properties are derived. The current thermo models are described as follows:

hConstThermo assumes a constant c_p and a heat of fusion H_f which is simply specified by a two values c_p H_f following the <`specieCoeffs`>.

janafThermo calculates c_p as a function of temperature T from a set of coefficients taken from JANAF tables of thermodynamics. The ordered list of coefficients is given in Table 8.3. The function is valid between a lower and upper limit in temperature T_l and T_h respectively. Two sets of coefficients are specified, the first set for temperatures above a common temperature T_c (and below T_h, the second for temperatures below T_c (and above T_l). The function relating c_p to temperature is:

$$c_p = R((((a_4T + a_3)T + a_2)T + a_1)T + a_0) \tag{8.1}$$

In addition, there are constants of integration, a_5 and a_6, both at high and low temperature, used to evaluating h and s respectively.

Description	Entry
Lower temperature limit	T_l (K)
Upper temperature limit	T_h (K)
Common temperature	T_c (K)
High temperature coefficients	$a_0 \dots a_4$
High temperature enthalpy offset	a_5
High temperature entropy offset	a_6
Low temperature coefficients	$a_0 \dots a_4$
Low temperature enthalpy offset	a_5
Low temperature entropy offset	a_6

Table 8.3: JANAF thermodynamics coefficients.

The transport coefficients <`transportCoeffs`> are used to to evaluate dynamic viscosity μ, thermal conductivity κ and laminar thermal conductivity (for enthalpy equation) α. The current transport models are described as follows:

constTransport assumes a constant μ and Prandtl number $Pr = c_p\mu/\kappa$ which is simply specified by a two values μ Pr following the <`thermoCoeffs`>.

sutherlandTransport calculates μ as a function of temperature T from a Sutherland coefficient A_s and Sutherland temperature T_s, specified by values following the <`thermoCoeffs`>; μ is calculated according to:

$$\mu = \frac{A_s\sqrt{T}}{1 + T_s/T} \tag{8.2}$$

The following is an example entry for a specie named `fuel` modelled using sutherlandTransport and janafThermo, with comments to explain the entries:

```
fuel                                            // keyword
fuel 1 44.0962                                  // specie
200 5000 1000                                   // -- janafThermo --
7.53414 0.0188722 -6.27185e-06 9.14756e-10 -4.78381e-14
-16467.5 -17.8923
0.933554 0.0264246 6.10597e-06 -2.19775e-08 9.51493e-12
-13958.5 19.2017                                // ------------------
1.67212e-06 170.672;                            // sutherlandTransport
```

The following is an example entry for a specie named `air` modelled using constTransport and hConstThermo, with comments to explain the entries:

```
mixture                                         // keyword
air 1 28.9                                      // specie
1000 2.544e+06                                  // hConstThermo
1.8e-05 0.7;                                    // constTransport
```

8.2 Turbulence models

The *turbulenceProperties* dictionary is read by any solver that uses models for turbulence or large-eddy simulation (LES). The entries required in the *turbulenceProperties* dictionary differ depending on whether the solver models turbulence or LES. The entries are listed for both turbulence and LES in Table 8.4.

The incompressible and compressible turbulence models, isochoric and anisochoric LES models and delta models are all named and described in Table 3.9. The user may consult *turbulenceProperties* dictionary from a relevant example case to get a full list of coefficients required for each model and their default values. The required coefficients may differ depending on whether the turbulence models are incompressible or compressible and whether the LES models are isochoric or anisochoric. For reference, these different categories of turbulence/LES models are represented in the *turbulenceProperties* dictionaries of the following example cases in the *$FOAM_TUTORIALS* directory:

turbFoam/cavity Incompressible turbulent models;

sonicTurbFoam/prism Compressible turbulent models;

oodles/pitzDaily Isochoric LES models;

Xoodles/pitzDaily Anisochoric LES models;

Entries for turbulence modelling	
`turbulenceModel`	Name of turbulence model
`turbulence`	Switch to turn turbulence modelling on/off
<`turbulenceModel`>`Coeffs`	Dictionary of coefficients for the respective `turbulenceModel`
`wallFunctionCoeffs`	Dictionary of wall function coefficients

Entries for LES	
`LESmodel`	Name of LES model
`delta`	Name of delta δ model
<`LESmodel`>`Coeffs`	Dictionary of coefficients for the respective `LESmodel`
<`delta`>`Coeffs`	Dictionary of coefficients for each `delta` model
`kappa`	von Kármán's constant κ
`wallFunctionCoeffs`	Dictionary of wall function coefficients

Table 8.4: Keyword entries in the *turbulenceProperties* dictionary.

Appendix A

Reference information

This chapter is currently a repository of information that: we do not consider worthy of inclusion in the main part of the User Guide because, for example, it is contains unnecessary detail or is outdated; we consider may be useful to a user in certain circumstances.

A.1 Running a decomposed case in parallel using MPICH

This section describes how to run OpenFOAM cases in parallel using MPI/MPICH rather than openMPI, as described in section 3.4.2.

The invocation of MPI/MPICH differs whether or not the application that is being executed has the same pathname on all processor nodes. The pathname to the executable can differ if:

- the processors do not all belong to the same UNIX/Linux architectures;
- there is no networked file system (NFS) access to the executable from all nodes and therefore it is installed in a different place on different nodes.

A.1.1 Same executable pathname on all nodes

On a single machine in which the processor nodes are all local to the user, the following command should be executed[1], noting that ` is a backwards quotation character, typically found at the top left of the keyboard (not a '):

```
mpirun -np <nProcs> `which <foamExec>` <root> <case>
   <otherArgs> -parallel < /dev/null >& log &
```

where: <nProcs> is the number of processors; <foamExec> is the executable, *e.g.* icoFoam; and, the output is redirected to a file named log. For example, if icoFoam is run on 3 nodes on the cavity tutorial in the $FOAM_RUN/*tutorials/icoFoam* directory, then the following command should be executed:

```
mpirun -np 3 `which icoFoam` $FOAM_RUN/tutorials/icoFoam cavity
    -parallel < /dev/null >& log &
```

[1]This command executes mpirun in the background which does not work for SGI mpirun; SGI users must therefore omit the final & in this and later mpirun commands.

When the processors that the user wishes to access are distributed across a cluster of machines, the user should execute the command:

```
mpirun  -machinefile <machinesFile> -np <nProcs> `which <foamExec>`
<root> <case> <otherArgs> -parallel < /dev/null >& log &
```

This is the same as before except that there is the <*machinesFile*> file that contains the names of the nodes, one per line, with the first one being the machine that the user is currently logged onto. A <*machinesFile*> is a file read by MPICH and therefore requires no header, only the names of the machines and number of processors to be used on each machine. For example, to run one process on machine arp and two on noddy, the file would be:

```
arp:1
noddy:2
```

Note: optimisation of the performance on a cluster of machines with shared memory may require recompiling of the MPICH library. See the MPICH documentation on how to do this.

A.1.2 Different executable pathname on different nodes

To run an executable with a different pathname on different nodes requires the same version of OpenFOAM to be installed on all nodes and the ability to run using rsh. The latter can be tested by trying to execute an application, *e.g.* icoFoam, on all nodes:

```
rsh <machineName> icoFoam <root> <case>
```

Different pathnames of executables can be specified through a <*p4pgFile*> file containing the names of nodes and the respective pathname to the executable. For example to run icoFoam on machine arp, a Linux machine, and on noddy, a Solaris machine, the <*p4pgFile*> would contain the following entries:

```
arp  0 /usr/local/OpenFOAM/OpenFOAM-1.4.1/applications/bin/linuxOptMPICH/icoFoam
noddy 1 /usr/local/OpenFOAM/OpenFOAM-1.4.1/applications/bin/solarisOptMPICH/icoFoam
```

The second entry per line, here 0 and 1, are the number of *additional* processes per machine. Since the MPI run is started from arp the master process runs on it and no additional processes should be started on it. The job is run by executing

```
mpirun  -p4pg <p4pgFile> `which <foamExec>` <root>
   <case> <otherArgs> -parallel < /dev/null >& log &
```

Index

Symbols Numbers A B C D E F G H I J K L M N O P Q R S T U V W X Z

Symbols

* tensor member function, P-25
+ tensor member function, P-25
- tensor member function, P-25
/ tensor member function, P-25
/*...*/ C++ syntax, U-78
// C++ syntax, U-78
OpenFOAM file syntax, U-100
include C++ syntax, U-72, U-78
& tensor member function, P-25
&& tensor member function, P-25
^ tensor member function, P-25
<LESmodel>Coeffs keyword, U-189
<delta>Coeffs keyword, U-189
<turbulenceModel>Coeffs keyword, U-189
0.000000e+00 directory, U-100
1-dimensional mesh, U-143
1D mesh, U-143
2-dimensional mesh, U-143
2D mesh, U-143
3D View button, U-172, U-173
3D view Properties menu entry, U-27, U-172–U-174

Numbers

0 directory, U-100

A

Accept button, U-171
access functions, P-23
Actor color button, U-172
adiabaticFlameT utility, U-92
adjustableRunTime keyword entry, U-61, U-106
adjustPhi tools, U-94
adjustTimeStep keyword, U-61
agglomerator keyword, U-117
algorithms tools, U-93
allTime menu entry, U-129
analytical solution, P-45
anisotropicFilter model, U-97
Annotate window panel, U-27, U-172
ansysToFoam utility, U-89
APIfunctions model, U-96
applicationClass keyword, U-105
applications, U-69
arbitrarily unstructured, P-31
arc keyword entry, U-48, U-154
arc keyword, U-153
ascii keyword entry, U-106
attachMesh utility, U-89
autoPatch utility, U-89
axes right-handed, U-152
right-handed rectangular Cartesian, P-15, U-21
axi-symmetric cases, U-148, U-158
axi-symmetric mesh, U-143

B

background process, U-28, U-81
backward keyword entry, U-114
Backward differencing, P-39

basicThermophysicalModels library, U-95
binary keyword entry, U-106
BirdCarreau model, U-98
blended differencing, P-38
block expansion ratio, U-155
block keyword, U-153
blockMesh solver, P-47
blockMesh utility, U-39, U-89, U-149
blockMesh menu entry, U-22, U-33
blockMesh executable vertex numbering, U-154
blockMeshDict dictionary, U-21, U-22, U-37, U-48, U-49, U-152, U-159
blocks keyword, U-22, U-154
bound tools, U-94
boundaries, U-143
boundary, U-143
boundary dictionary, U-142, U-152
boundary condition calculated, U-150
cyclic, U-149
directionMixed, U-150
empty, P-65, P-71, U-21, U-143, U-148
fixedGradient, U-150
fixedValue, U-150
fluxCorrectedVelocity, U-151
gammaContactAngle, U-59
inlet, P-71
inletOutlet, U-151
mixed, U-150
movingWallVelocity, U-151
outlet, P-71
outletInlet, U-151
partialSlip, U-151
patch, U-148
pressureDirectedInletVelocity, U-151
pressureInletVelocity, U-151
pressureOutlet, P-65
pressureTransmissive, U-151
processor, U-149
setup, U-23
slip, U-151
supersonicFreeStream, U-151
surfaceNormalFixedValue, U-151
symmetryPlane, P-65, U-148
totalPressure, U-151
turbulentInlet, U-151
wall, U-42
wall, P-65, P-71, U-148
wallBuoyantPressure, U-151
wedge, U-143, U-148, U-149, U-158
zeroGradient, U-150
boundary conditions, P-43
Dirichlet, P-43
inlet, P-44
Neumann, P-43
no-slip impermeable wall, P-44
outlet, P-44
physical, P-44
symmetry plane, P-44
boundary type empty, U-132
wall, U-42
boundaryField keyword, U-104
boundaryFoam solver, U-86
bounded keyword entry, U-112, U-113
boxToCell keyword, U-60
boxTurb utility, U-88
breaking of a dam, U-57
bubbleFoam solver, U-87
buoyantFoam solver, U-88
buoyantSimpleFoam solver, U-88
button 3D View, U-172, U-173
Accept, U-171
Actor color, U-172
Close Case, U-33
Compact, U-131
Delete, U-171
Display Orientation Axes, U-172
Info, U-131
My Jobs, U-131
Orientation Axes, U-27
Refresh Case Browser, U-41
Reset Range, U-28
Reset, U-171
Start Calculation Now, U-28
Start Calculation, U-35
Use parallel projection, U-27, U-172
cont, U-131
endNow, U-131
end, U-131
kill, U-131

purge, U-131
read, U-131
status, U-131
suspend, U-131

C

C++ syntax
/*...*/, U-78
//, U-78
include, U-72, U-78
cacheAgglomeration keyword, U-117
calculated
boundary condition, U-150
Camera window panel, U-172
Camera Controls window panel, U-172
Camera Orientation window panel, U-172
case
browser, U-125
server, U-131
case keyword, U-101
case manager
FoamX, U-121
Case Name text box, U-127
Case Root text box, U-127
caseRoots keyword, U-19
cases, U-99
cavitatingFoam solver, U-87
cavity flow, U-19
ccm26ToFoam utility, U-89
CEI_ARCH
environment variable, U-178
CEI_HOME
environment variable, U-178
cell
expansion ratio, U-155
cell class, P-31
cell
keyword entry, U-180
cellDecompFiniteElement
library, U-94
cellPoint
keyword entry, U-180
cellPointFace
keyword entry, U-180
cells
dictionary, U-152
cellSet utility, U-89
central differencing, P-38
cfdTools
library, U-94
cfxToFoam utility, U-89, U-159
cGamma keyword, U-63
channelOodles solver, U-87
checkMesh utility, U-89, U-160
checkYPlus utility, U-92
chemistryModel
library, U-96
chemistryModel model, U-96
chemistrySolver model, U-96
chemkinMixture model, U-95, U-186
chemkinToFoam utility, U-92
Class menu, U-127
class
cell, P-31
dimensionSet, P-26, P-32, P-33
face, P-31
finiteVolumeCalculus, P-36
finiteVolumeMethod, P-36
fvMesh, P-31
fvSchemes, P-38
fvc, P-36
fvm, P-36
pointField, P-31
polyBoundaryMesh, P-31
polyMesh, P-31, U-139, U-141
polyPatchList, P-31
polyPatch, P-31
scalarField, P-29
scalar, P-24
slice, P-31
symmTensorField, P-29
symmTensorThirdField, P-29
tensorField, P-29
tensorThirdField, P-29
tensor, P-24
vectorField, P-29
vector, P-24, U-103
word, P-26, P-31
class keyword, U-101
clockTime
keyword entry, U-106
Close Case button, U-33
cloud keyword, U-181
cmptAv
tensor member function, P-25
Co utility, U-91
cofactors
tensor member function, P-25
coldEngineFoam solver, U-87
Color by menu, U-172
combustionThermophysicalModels

library, U-95
comments, U-78
Compact button, U-131
compressed
keyword entry, U-106
compressible tools, U-94
compressibleLESmodels
library, U-97
compressibleTurbulenceModels
library, U-97
constant directory, U-99, U-185
constLaminarFlameSpeed model, U-95
constTransport model, U-96, U-186
cont button, U-131
containers tools, U-93
continuum
mechanics, P-15
control
of time, U-105
controlDict
dictionary, P-67, U-24, U-34, U-43, U-52, U-61, U-99, U-165
controlDict file, P-49
convection, *see* divergence, P-38
convergence, U-41
convertToMeters keyword, U-153
coodles solver, U-87
coordinate
system, P-15
coordinate system, U-21
CORBA, U-94, U-121
corrected
keyword entry, U-112, U-113
couplePatches utility, U-89
Courant number, P-42, U-25
cpuTime
keyword entry, U-106
Crank Nicholson
temporal discretisation, P-43
CrankNicholson
keyword entry, U-114
createPatch utility, U-89
cross product, *see* tensor, vector cross product
CrossPowerLaw
keyword entry, U-61
CrossPowerLaw model, U-98
cubeRootVolDelta model, U-97
cubicCorrected
keyword entry, U-114
cubicCorrection
keyword entry, U-111
curl, P-37
curl
fvc member function, P-37
curve keyword, U-181
cyclic
boundary condition, U-149
cyclic
keyword entry, U-148
cylinder
flow around a, P-45

D

d2dt2
fvc member function, P-37
fvm member function, P-37
dam
breaking of a, U-57
db tools, U-93
ddt
fvc member function, P-37
fvm member function, P-37
DeardorffDiffStress model, U-97
decomposePar utility, U-82, U-83, U-92
decomposeParDict
dictionary, U-82
decomposition
of field, U-82
of mesh, U-82
decompression of a tank, P-63
defaultFieldValues keyword, U-60
deformedGeom utility, U-89
Delete button, U-171
delta keyword, U-84, U-189
deltaT keyword, U-106
dependencies, U-72
dependency lists, U-72
det
tensor member function, P-25
determinant, *see* tensor, determinant
dev
tensor member function, P-25
diag
tensor member function, P-25
diagonal
keyword entry, U-117
DIC
keyword entry, U-117
DICGaussSeidel
keyword entry, U-117
Dictionaries dictionary tree, U-133

dictionary
PISO, U-26
blockMeshDict, U-21, U-22, U-37, U-48, U-49, U-152, U-159
boundary, U-142, U-152
cells, U-152
controlDict, P-67, U-24, U-34, U-43, U-52, U-61, U-99, U-165
decomposeParDict, U-82
faces, U-141, U-152
fvSchemes, U-62, U-99, U-107, U-108
fvSolution, U-99, U-115
mechanicalProperties, U-51
neighbour, U-142
owner, U-141
points, U-141, U-152
thermalProperties, U-51, U-52
thermophysicalProperties, U-185
transportProperties, U-24, U-41, U-43
turbulenceProperties, U-43, U-188
dictionary tree
Dictionaries, U-133
Fields, U-23, U-132
Mesh, U-23
Patches, U-23
dieselEngineFoam solver, U-87
dieselFoam solver, U-87
dieselMixture model, U-95, U-186
dieselSpray
library, U-94
diEthylEther model, U-95
differencing
Backward, P-39
blended, P-38
central, P-38
Euler implicit, P-39
Gamma, P-38
MINMOD, P-38
SUPERBEE, P-38
upwind, P-38
van Leer, P-38
DILU
keyword entry, U-117
dimension
checking in OpenFOAM, P-26
dimensioned<Type> template class, P-26
dimensionedTypes tools, U-93
dimensions keyword, U-104
dimensionSet class, P-26, P-32, P-33
dimensionSet tools, U-93
diMethylEther model, U-95
direct numerical simulation, U-62
directionMixed
boundary condition, U-150
directory
0.000000e+00, U-100
0, U-100
Make, U-73
constant, U-99, U-185
fluentInterface, U-175
polyMesh, U-99, U-141
processorN, U-83
run, U-99
system, P-49, U-99
tutorials, P-45, U-19
discretisation
equation, P-33
Display window panel, U-27, U-28, U-170, U-172
Display Orientation Axes button, U-172
distance
keyword entry, U-181
distributed keyword, U-84, U-85
div
fvc member function, P-37
fvm member function, P-37
divergence, P-37, P-39
divSchemes keyword, U-108
divU utility, U-91
dnsFoam solver, U-87
double inner product, *see* tensor,double inner product
dynamicMesh
library, U-94
dynMixedSmagorinsky model, U-97
dynOneEqEddy model, U-97
dynSmagorinsky model, U-97

E

edgeGrading keyword, U-155
edgeMesh
library, U-94
edges keyword, U-153
electrostaticFoam solver, U-88
empty
boundary condition, P-65, P-71, U-21, U-143, U-148
empty boundary type, U-132
empty
keyword entry, U-148
end button, U-131
endNow button, U-131

endTime keyword, U-25, U-105, U-106
engine
library, U-94
engineCompRatio utility, U-92
engineFoam solver, U-87
engineSwirl utility, U-88
ensight74FoamExec utility, U-178
ensight76FoamExec utility, U-90
ENSIGHT7_INPUT
environment variable, U-178
ENSIGHT7_READER
environment variable, U-178
enstrophy utility, U-91
environment variable
CEI_ARCH, U-178
CEI_HOME, U-178
ENSIGHT7_INPUT, U-178
ENSIGHT7_READER, U-178
FOAMX_PATH, U-137
FOAMX_SYSTEM_CONFIG, U-137
FOAMX_USER_CONFIG, U-137
FOAM_RUN, U-99, U-137
JAVA_HOME, U-137
WM_ARCH, U-76
WM_COMPILER_BIN, U-76
WM_COMPILER_DIR, U-76
WM_COMPILER_LIB, U-76
WM_COMPILER, U-76
WM_COMPILE_OPTION, U-76
WM_DIR, U-76
WM_JAVAC_OPTION, U-76
WM_LINK_LANGUAGE, U-76
WM_MPLIB, U-76
WM_OPTIONS, U-76
WM_PROJECT_DIR, U-76
WM_PROJECT_INST_DIR, U-76
WM_PROJECT_LANGUAGE, U-76
WM_PROJECT_USER_DIR, U-76
WM_PROJECT_VERSION, U-76
WM_PROJECT, U-76
WM_SHELL, U-76
wmake, U-76
environmentalProperties file, U-61
equilibriumCO utility, U-92
equilibriumFlameT utility, U-93
errorEstimation
library, U-94
estimateScalarError utility, U-93
Euler
keyword entry, U-114
Euler implicit
differencing, P-39
temporal discretisation, P-42
examples
decompression of a tank, P-63
flow around a cylinder, P-45
flow over backward step, P-54
Hartmann problem, P-69
supersonic flow over forward step, P-59
explicit
temporal discretisation, P-42
exponential model, U-96
extrudeMesh utility, U-89

F

face class, P-31
face keyword, U-181
faceAreaPair
keyword entry, U-117
faceDecompFiniteElement
library, U-94
faces
dictionary, U-141, U-152
faceSet utility, U-89
FDIC
keyword entry, U-117
field
U, U-25
p, U-25
decomposition, U-82
FieldField<Type> template class, P-32
Fields dictionary tree, U-23, U-132
Fields window, U-28
fields, P-29
mapping, U-165
fields tools, U-93
fields keyword, U-180
Field<Type> template class, P-29
fieldValues keyword, U-60
file
FoamX.cfg, U-137
FoamXClient.cfg, U-122, U-136
Make/files, U-75
controlDict, P-49
environmentalProperties, U-61
files, U-73
options, U-73
transportProperties, U-60
file format, U-100
files file, U-73
financialFoam solver, U-88

finite volume
discretisation, P-27
mesh, P-31
finiteVolume tools, U-94
finiteVolumeCalculus class, P-36
finiteVolumeMethod class, P-36
firstTime
menu entry, U-129
firstTime keyword, U-105
fixed
keyword entry, U-106
fixedGradient
boundary condition, U-150
fixedValue
boundary condition, U-150
flattenMesh utility, U-89
flow
free surface, U-57
laminar, U-19
steady, turbulent, P-54
supersonic, P-59
turbulent, U-20
flow around a cylinder, P-45
flow over backward step, P-54
flowType utility, U-91
fluentInterface directory, U-175
fluentMeshToFoam utility, U-89, U-159
fluxCorrectedVelocity
boundary condition, U-151
fluxRequired keyword, U-108
OpenFOAM
cases, U-99
FOAM_RUN
environment variable, U-99, U-137
Foam Utilities menu, U-22, U-33, U-34
foamCorrectVrt script/alias, U-164
foamDataToFluent utility, U-90, U-175
foamDebugSwitches utility, U-93
FoamFile keyword, U-101
foamInfoExec utility, U-93
foamJob script/alias, U-182
foamLog script/alias, U-183
foamMeshToFluent utility, U-89, U-175
foamToEnsight utility, U-90
foamToFieldview9 utility, U-90
foamToGMV utility, U-90
foamToVTK utility, U-91
foamUser
library, U-81
FoamX
case browser, U-125
case manager, U-121
case server, U-131
OpenFOAM case manager, U-121
host browser, U-122
JAVA GUI, U-123
name server, U-122
FoamX utility, U-88
FoamX.cfg file, U-137
FOAMX_PATH
environment variable, U-137
FOAMX_SYSTEM_CONFIG
environment variable, U-137
FOAMX_USER_CONFIG
environment variable, U-137
FoamXClient.cfg file, U-122, U-136
foreground
process, U-28
format keyword, U-101
fourth
keyword entry, U-112, U-113
fvc class, P-36
fvc member function
curl, P-37
d2dt2, P-37
ddt, P-37
div, P-37
gGrad, P-37
grad, P-37
laplacian, P-37
lsGrad, P-37
snGrad, P-37
snGradCorrection, P-37
sqrGradGrad, P-37
fvm class, P-36
fvm member function
d2dt2, P-37
ddt, P-37
div, P-37
laplacian, P-37
Su, P-37
SuSp, P-37
fvMatrix template class, P-36
fvMesh class, P-31
fvSchemes
dictionary, U-62, U-99, U-107, U-108
fvSchemes class, P-38
fvSchemes
menu entry, U-53
fvSolution

dictionary, U-99, U-115

G

gambitToFoam utility, U-89, U-159
GAMG
keyword entry, U-116, U-117
Gamma
keyword entry, U-111
Gamma differencing, P-38
gammaContactAngle
boundary condition, U-59
Gauss
keyword entry, U-112
Gauss's theorem, P-36
GaussSeidel
keyword entry, U-117
General window panel, U-172
general model, U-96
general
keyword entry, U-106
geometric-algebraic multi-grid, U-117
GeometricBoundaryField template class, P-32
geometricField<Type> template class, P-32
gGrad
fvc member function, P-37
global tools, U-94
gmshToFoam utility, U-89
gnuplot
keyword entry, U-107, U-180
grad
fvc member function, P-37
(Grad Grad) squared, P-37
gradient, P-37, P-40
Gauss scheme, P-40
Gauss's theorem, U-53
least square fit, U-53
least squares method, P-40, U-53
surface normal, P-40
gradSchemes keyword, U-108
graphFormat keyword, U-107
Gstream
library, U-94
guldersLaminarFlameSpeed model, U-95

H

hConstThermo model, U-96, U-185
hhuMixtureThermo model, U-95, U-186
hierarchical
keyword entry, U-83, U-84
hMixtureThermo model, U-95, U-186
homogeneousMixture model, U-95, U-186
host, U-20
browser, U-122
hThermo model, U-95, U-186

I

I
tensor member function, P-25
icoDyMFoam solver, U-86
icoErrorEstimate utility, U-93
icoFoam solver, U-19, U-24, U-25, U-28, U-86
icoMomentError utility, U-93
ideasToFoam utility, U-159
ideasUnvToFoam utility, U-89
identities, *see* tensor, identities
identity, *see* tensor, identity
incompressible tools, U-94
incompressibleLESmodels
library, U-97
incompressiblePostProcessing
library, U-94
incompressibleTransportModels
library, P-55, U-98
incompressibleTurbulenceModels
library, P-55, U-96
index
notation, P-16, P-17
Info button, U-131
Information window panel, U-170
inhomogeneousMixture model, U-95, U-186
inlet
boundary condition, P-71
inletOutlet
boundary condition, U-151
inner product, *see* tensor, inner product
insideCells utility, U-89
instance keyword, U-101
interFoam solver, U-87
internalField keyword, U-104, U-133
interpolationScheme keyword, U-180
interpolations tools, U-94
interpolationSchemes keyword, U-108
inv
tensor member function, P-25
isoOctane model, U-95

J

janafThermo model, U-96, U-185
JAVA_HOME
environment variable, U-137
jplot
keyword entry, U-107, U-180

K

kappa keyword, U-189
kEpsilon model, U-96, U-97
keyword
FoamFile, U-101
LESmodel, U-189
adjustTimeStep, U-61
agglomerator, U-117
applicationClass, U-105
arc, U-153
blocks, U-22, U-154
block, U-153
boundaryField, U-104
boxToCell, U-60
cGamma, U-63
cacheAgglomeration, U-117
caseRoots, U-19
case, U-101
class, U-101
cloud, U-181
convertToMeters, U-153
curve, U-181
defaultFieldValues, U-60
deltaT, U-106
delta, U-84, U-189
dimensions, U-104
distributed, U-84, U-85
divSchemes, U-108
edgeGrading, U-155
edges, U-153
endTime, U-25, U-105, U-106
face, U-181
fieldValues, U-60
fields, U-180
firstTime, U-105
fluxRequired, U-108
format, U-101
gradSchemes, U-108
graphFormat, U-107
instance, U-101
internalField, U-104, U-133
interpolationSchemes, U-108
interpolationScheme, U-180
kappa, U-189
laplacianSchemes, U-108
latestTime, U-41
leastSquares, U-53
local, U-101
manualCoeffs, U-84
maxCo, U-61
maxDeltaT, U-61
mergeLevels, U-118
method, U-84
metisCoeffs, U-84
midPointAndFace, U-181
midPoint, U-181
nFaces, U-142
nFinestSweeps, U-118
nGammaSubCycles, U-63
nPostSweeps, U-118
nPreSweeps, U-118
numberOfSubdomains, U-84
n, U-84
object, U-101
order, U-84
outputFormat, U-180
pRefCell, U-26, U-119
pRefValue, U-26, U-119
patchMap, U-166
patches, U-153, U-155
pdRefCell, U-119
pdRefValue, U-119
physicalType, U-142, U-146
preconditioner, U-115, U-116
processorWeights, U-84
purgeWrite, U-106
refGradient, U-150
referenceLevel, U-104, U-133
regions, U-60
relTol, U-54, U-115, U-116
roots, U-84, U-85
root, U-101
runTimeModifiable, U-107
sampleSets, U-180
simpleGrading, U-155
smoother, U-118
snGradSchemes, U-108
solvers, U-115
spline, U-153
startFace, U-142
startFrom, U-25, U-105
startTime, U-25, U-105
stopAt, U-105
thermoType, U-185
timeFormat, U-106
timePrecision, U-107
timeScheme, U-108
tolerance, U-54, U-115, U-116
topoSetSource, U-60
turbulenceModel, U-189

turbulence, U-189
type, U-146
uniform, U-181
valueFraction, U-150
value, U-150
version, U-101
vertices, U-22, U-153
wallFunctionCoeffs, U-189
writeCompression, U-106
writeControl, U-25, U-61, U-106
writeFormat, U-55, U-106
writeInterval, U-25, U-35, U-106
writePrecision, U-106
<LESmodel>Coeffs, U-189
<delta>Coeffs, U-189
<turbulenceModel>Coeffs, U-189
keyword entry
CrankNicholson, U-114
CrossPowerLaw, U-61
DICGaussSeidel, U-117
DIC, U-117
DILU, U-117
Euler, U-114
FDIC, U-117
GAMG, U-116, U-117
Gamma, U-111
GaussSeidel, U-117
Gauss, U-112
MGridGen, U-117
MUSCL, U-111
Newtonian, U-61
PBiCG, U-116
PCG, U-116
QUICK, U-111, U-114
SFCD, U-111, U-114
UMIST, U-109
adjustableRunTime, U-61, U-106
arc, U-48, U-154
ascii, U-106
backward, U-114
binary, U-106
bounded, U-112, U-113
cellPointFace, U-180
cellPoint, U-180
cell, U-180
clockTime, U-106
compressed, U-106
corrected, U-112, U-113
cpuTime, U-106
cubicCorrected, U-114
cubicCorrection, U-111
cyclic, U-148
diagonal, U-117
distance, U-181
empty, U-148
faceAreaPair, U-117
fixed, U-106
fourth, U-112, U-113
general, U-106
gnuplot, U-107, U-180
hierarchical, U-83, U-84
jplot, U-107, U-180
latestTime, U-105
leastSquares, U-112
limitedCubic, U-111
limitedLinear, U-111
limited, U-112, U-113
linearUpwind, U-111, U-114
linear, U-111, U-114
line, U-154
manual, U-83, U-84
metis, U-83, U-84
midPoint, U-111
nextWrite, U-106
noWriteNow, U-106
none, U-109, U-117
patch, U-148
polyLine, U-154
polySpline, U-154
processor, U-148
raw, U-107, U-180
runTime, U-35, U-106
scientific, U-106
simpleSpline, U-154
simple, U-83, U-84
skewLinear, U-111, U-114
smoothSolver, U-116
startTime, U-25, U-105
steadyState, U-114
symmetryPlane, U-148
timeStep, U-25, U-35, U-106
uncompressed, U-106
uncorrected, U-112, U-113
upwind, U-111, U-114
vanLeer, U-111
wall, U-148
wedge, U-148
writeControl, U-106
writeNow, U-105
xmgr, U-107, U-180

xyz, U-181
x, U-181
y, U-181
z, U-181
kill button, U-131
kivaToFoam utility, U-89
Kronecker delta, P-21

L

lagrangian
library, U-94
Lambda2 utility, U-91
LamBremhorstKE model, U-96
laminar model, U-96, U-97
laminarFlameSpeedModels
library, U-95
laplaceFilter model, U-97
Laplacian, P-38
laplacian, P-37
laplacian
fvc member function, P-37
fvm member function, P-37
laplacianFoam solver, U-86
laplacianSchemes keyword, U-108
latestTime
keyword entry, U-105
menu entry, U-129
latestTime keyword, U-41
LaunderGibsonRSTM model, U-96, U-97
LaunderSharmaKE model, U-96, U-97
leastSquares
keyword entry, U-112
leastSquares keyword, U-53
LESdeltas
library, U-97
LESfilters
library, U-97
lesInterFoam solver, U-87
LESmodel keyword, U-189
libraries, U-69
library
Gstream, U-94
LESdeltas, U-97
LESfilters, U-97
ODE, U-94
OpenFOAM, U-93
PVFoamReader, U-169
basicThermophysicalModels, U-95
cellDecompFiniteElement, U-94
cfdTools, U-94
chemistryModel, U-96
combustionThermophysicalModels, U-95
compressibleLESmodels, U-97
compressibleTurbulenceModels, U-97
dieselSpray, U-94
dynamicMesh, U-94
edgeMesh, U-94
engine, U-94
errorEstimation, U-94
faceDecompFiniteElement, U-94
foamUser, U-81
incompressibleLESmodels, U-97
incompressiblePostProcessing, U-94
incompressibleTransportModels, P-55, U-98
incompressibleTurbulenceModels, P-55, U-96
lagrangian, U-94
laminarFlameSpeedModels, U-95
liquids, U-95
meshTools, U-94
mico-2.3.13, U-94
mpich-1.2.4, U-94
openmpi-1.2.3, U-94
pdf, U-96
primitive, P-23
randomProcesses, U-95
sampling, U-94
shapeMeshTools, U-94
specie, U-95
thermophysicalFunctions, U-96
thermophysical, U-185
triSurface, U-94
vtkFoam, U-169
zlib-1.2.1, U-94
lid-driven cavity flow, U-19
LienCubicKE model, U-96
LienCubicKELowRE model, U-96
LienLeschzinerLowRE model, U-96
limited
keyword entry, U-112, U-113
limitedCubic
keyword entry, U-111
limitedLinear
keyword entry, U-111
line
keyword entry, U-154
linear
keyword entry, U-111, U-114
linearUpwind
keyword entry, U-111, U-114
liquid
electrically-conducting, P-69

liquids
library, U-95
lists, P-29
List<Type> template class, P-29
local keyword, U-101
locDynOneEqEddy model, U-97
Lower and Upper Times text box, U-171
lowReOneEqEddy model, U-97
LRDDiffStress model, U-97
LRR model, U-96, U-97
lsGrad
fvc member function, P-37

M

Mach utility, U-91
mag
tensor member function, P-25
magGradU utility, U-91
magnetohydrodynamics, P-69
magSqr
tensor member function, P-25
magU utility, U-36, U-91
Make directory, U-73
make script/alias, U-71
Make/files file, U-75
manual
keyword entry, U-83, U-84
manualCoeffs keyword, U-84
mapFields utility, U-34, U-40, U-43, U-56, U-88, U-165
mapFields
menu entry, U-34
mapping
fields, U-165
matrices tools, U-94
max
tensor member function, P-25
maxCo keyword, U-61
maxDeltaT keyword, U-61
mechanicalProperties
dictionary, U-51
menu
Class, U-127
Color by, U-172
Foam Utilities, U-22, U-33, U-34
Mesh, U-50
View, U-28, U-172, U-173
menu entry
3D view Properties, U-27, U-172–U-174
Property, U-172
Read Mesh&Fields, U-23, U-45, U-50
Refresh Case Browser, U-41
Source, U-28, U-173
Wireframe, U-172
allTime, U-129
blockMesh, U-22, U-33
firstTime, U-129
fvSchemes, U-53
latestTime, U-129
mapFields, U-34
noTime, U-129
preProcessing, U-34
sample, U-55
mergeLevels keyword, U-118
mergeMeshes utility, U-90
Mesh dictionary tree, U-23
Mesh menu, U-50
mesh
1-dimensional, U-143
1D, U-143
2-dimensional, U-143
2D, U-143
axi-symmetric, U-143
basic, P-31
block structured, U-149
decomposition, U-82
description, U-139
finite volume, P-31
generation, U-149
grading, U-149, U-155
grading, example of, P-54
non-orthogonal, P-45
refinement, P-63
resolution, U-33
specification, U-139
validity constraints, U-139
meshes tools, U-94
meshTools
library, U-94
message passing interface
MPICH, U-191
openMPI, U-83
method keyword, U-84
metis
keyword entry, U-83, U-84
metisCoeffs keyword, U-84
MGridGen
keyword entry, U-117
mhdFoam solver, P-71, U-88
mico-2.3.13
library, U-94

midPoint
keyword entry, U-111
midPoint keyword, U-181
midPointAndFace keyword, U-181
min
tensor member function, P-25
MINMOD differencing, P-38
mirrorMesh utility, U-90
mixed
boundary condition, U-150
mixedSmagorinsky model, U-97
mixtureAdiabaticFlameT utility, U-93
model
APIfunctions, U-96
BirdCarreau, U-98
CrossPowerLaw, U-98
DeardorffDiffStress, U-97
LRDDiffStress, U-97
LRR, U-96, U-97
LamBremhorstKE, U-96
LaunderGibsonRSTM, U-96, U-97
LaunderSharmaKE, U-96, U-97
LienCubicKELowRE, U-96
LienCubicKE, U-96
LienLeschzinerLowRE, U-96
NSRDSfunctions, U-96
Newtonian, U-98
NonlinearKEShih, U-96
PrandtlDelta, U-97
QZeta, U-96
RNGkEpsilon, U-96, U-97
RosinRammler, U-96
Smagorinsky2, U-97
Smagorinsky, U-97
SpalartAllmaras, U-97
anisotropicFilter, U-97
chemistryModel, U-96
chemistrySolver, U-96
chemkinMixture, U-95, U-186
constLaminarFlameSpeed, U-95
constTransport, U-96, U-186
cubeRootVolDelta, U-97
diEthylEther, U-95
diMethylEther, U-95
dieselMixture, U-95, U-186
dynMixedSmagorinsky, U-97
dynOneEqEddy, U-97
dynSmagorinsky, U-97
exponential, U-96
general, U-96
guldersLaminarFlameSpeed, U-95
hConstThermo, U-96, U-185
hMixtureThermo, U-95, U-186
hThermo, U-95, U-186
hhuMixtureThermo, U-95, U-186
homogeneousMixture, U-95, U-186
inhomogeneousMixture, U-95, U-186
isoOctane, U-95
janafThermo, U-96, U-185
kEpsilon, U-96, U-97
laminar, U-96, U-97
laplaceFilter, U-97
locDynOneEqEddy, U-97
lowReOneEqEddy, U-97
mixedSmagorinsky, U-97
multiComponentMixture, U-95, U-186
nDecane, U-95
nDodecane, U-95
nHeptane, U-95
nOctane, U-95
normal, U-96
oneEqEddy, U-97
perfectGas, U-96, U-185
pureMixture, U-95, U-186
scaleSimilarity, U-97
simpleFilter, U-97
smoothDelta, U-97
specieThermo, U-96, U-185
spectEddyVisc, U-97
sutherlandTransport, U-96, U-186
uniform, U-96
veryInhomogeneousMixture, U-95, U-186
water, U-95
momentScalarError utility, U-93
moveDynamicMesh utility, U-90
moveEngineMesh utility, U-90
moveMesh utility, U-90
movingWallVelocity
boundary condition, U-151
MPI
MPICH, U-191
openMPI, U-83
MPICH
message passing interface, U-191
MPI, U-191
mpich-1.2.4
library, U-94
mshToFoam utility, U-89
multiComponentMixture model, U-95, U-186
multigrid

geometric-algebraic, U-117
multiphaseInterFoam solver, U-87
MUSCL
keyword entry, U-111
My Jobs button, U-131

N

n keyword, U-84
nabla
operator, P-27
name
server, U-122
nDecane model, U-95
nDodecane model, U-95
neighbour
dictionary, U-142
netgenNeutralToFoam utility, U-89
Newtonian
keyword entry, U-61
Newtonian model, U-98
nextWrite
keyword entry, U-106
nFaces keyword, U-142
nFinestSweeps keyword, U-118
nGammaSubCycles keyword, U-63
nHeptane model, U-95
nOctane model, U-95
non-orthogonal mesh, P-45
none
keyword entry, U-109, U-117
NonlinearKEShih model, U-96
nonNewtonianIcoFoam solver, U-86
normal model, U-96
noTime
menu entry, U-129
noWriteNow
keyword entry, U-106
nPostSweeps keyword, U-118
nPreSweeps keyword, U-118
NSRDSfunctions model, U-96
numberOfSubdomains keyword, U-84

O

object keyword, U-101
objToVTK utility, U-90
ODE
library, U-94
oneEqEddy model, U-97
oodles solver, U-87
Opacity text box, U-172
OpenFOAM
applications, U-69
file format, U-100
libraries, U-69
OpenFOAM
library, U-93
OpenFOAM file syntax
//, U-100
openMPI
message passing interface, U-83
MPI, U-83
openmpi-1.2.3
library, U-94
operator
scalar, P-28
vector, P-27
options file, U-73
order keyword, U-84
Orientation Axes button, U-27
outer product, *see* tensor, outer product
outlet
boundary condition, P-71
outletInlet
boundary condition, U-151
outputFormat keyword, U-180
owner
dictionary, U-141

P

p field, U-25
paraFoam, U-26, U-169
paraFoam utility, U-90
parallel
running, U-82
Parameters window panel, U-28, U-170, U-171
partialSlip
boundary condition, U-151
patch
boundary condition, U-148
patch
keyword entry, U-148
patchAverage utility, U-92
Patches dictionary tree, U-23
patches keyword, U-153, U-155
patchIntegrate utility, U-92
patchMap keyword, U-166
patchTool utility, U-90
PBiCG
keyword entry, U-116
PCG
keyword entry, U-116
pdf

library, U-96
pdRefCell keyword, U-119
pdRefValue keyword, U-119
Pe utility, U-91
perfectGas model, U-96, U-185
permutation symbol, P-20
physicalType keyword, U-142, U-146
PISO
dictionary, U-26
plot3dToFoam utility, U-89
pointField class, P-31
pointField<Type> template class, P-33
points
dictionary, U-141, U-152
pointSet utility, U-90
polyBoundaryMesh class, P-31
polyDualMesh utility, U-89
polyLine
keyword entry, U-154
polyMesh directory, U-99, U-141
polyMesh class, P-31, U-139, U-141
polyPatch class, P-31
polyPatchList class, P-31
polySpline
keyword entry, U-154
post-processing, U-169
post-processing
paraFoam, U-169
postChannel utility, U-92
potentialFoam solver, P-46, U-86
pow
tensor member function, P-25
PrandtlDelta model, U-97
preconditioner keyword, U-115, U-116
pRefCell keyword, U-26, U-119
pRefValue keyword, U-26, U-119
preProcessing
menu entry, U-34
pressure waves
in liquids, P-64
pressureDirectedInletVelocity
boundary condition, U-151
pressureInletVelocity
boundary condition, U-151
pressureOutlet
boundary condition, P-65
pressureTransmissive
boundary condition, U-151
primitive
library, P-23
primitives tools, U-94
process
background, U-28, U-81
foreground, U-28
processor
boundary condition, U-149
processor
keyword entry, U-148
processorN directory, U-83
processorWeights keyword, U-84
Property
menu entry, U-172
ptot utility, U-92
pureMixture model, U-95, U-186
purge button, U-131
purgeWrite keyword, U-106
PVFoamReader
library, U-169

Q

Q utility, U-91
QUICK
keyword entry, U-111, U-114
QZeta model, U-96

R

R utility, U-91
randomProcesses
library, U-95
rasInterFoam solver, U-87
raw
keyword entry, U-107, U-180
Rcomponents utility, U-91
reactingFoam solver, U-87
read button, U-131
Read Mesh&Fields
menu entry, U-23, U-45, U-50
reconstructPar utility, U-85, U-92
reconstructParMesh utility, U-92
referenceLevel keyword, U-104, U-133
refGradient keyword, U-150
refineMesh utility, U-90
Refresh Case Browser button, U-41
Refresh Case Browser
menu entry, U-41
Region window, U-28
regions keyword, U-60
relative tolerance, U-116
relTol keyword, U-54, U-115, U-116
renumberMesh utility, U-90
Reset button, U-171

Reset Range button, U-28
restart, U-41
Reynolds number, U-20, U-24
rhopSonicFoam solver, U-86
rhoSimpleFoam solver, U-86
rhoSonicFoam solver, U-86
rhoTurbFoam solver, U-86
rmdepall script/alias, U-77
RNGkEpsilon model, U-96, U-97
root keyword, U-101
roots keyword, U-84, U-85
RosinRammler model, U-96
rotateMesh utility, U-90
run
parallel, U-82
run directory, U-99
runFoamX script/alias, U-121–U-123
runFoamXHB script/alias, U-121, U-122
runTime
keyword entry, U-35, U-106
runTimeModifiable keyword, U-107

S

sammToFoam utility, U-89
sample utility, U-92, U-179
sample
menu entry, U-55
sampleSets keyword, U-180
sampleSurface utility, U-92
sampling
library, U-94
scalar, P-16
operator, P-28
scalar class, P-24
scalarField class, P-29
scalarTransportFoam solver, U-86
scale
tensor member function, P-25
scalePoints utility, U-163
scaleSimilarity model, U-97
scientific
keyword entry, U-106
script/alias
foamCorrectVrt, U-164
foamJob, U-182
foamLog, U-183
make, U-71
rmdepall, U-77
runFoamXHB, U-121, U-122
runFoamX, U-121–U-123
wclean, U-76
wmake, U-71
second time derivative, P-37
Seed window, U-175
Selection Window window, U-27, U-170
setFields utility, U-59, U-60, U-88
settlingFoam solver, U-87
SFCD
keyword entry, U-111, U-114
shape, U-154
shapeMeshTools
library, U-94
simple
keyword entry, U-83, U-84
simpleFilter model, U-97
simpleFoam solver, P-55, U-86
simpleGrading keyword, U-155
simpleSpline
keyword entry, U-154
skew
tensor member function, P-25
skewLinear
keyword entry, U-111, U-114
slice class, P-31
slip
boundary condition, U-151
Smagorinsky model, U-97
Smagorinsky2 model, U-97
smapToFoam utility, U-91
smoothDelta model, U-97
smoother keyword, U-118
smoothSolver
keyword entry, U-116
snGrad
fvc member function, P-37
snGradCorrection
fvc member function, P-37
snGradSchemes keyword, U-108
solidDisplacementFoam solver, U-51, U-88
solidEquilibriumDisplacementFoam solver, U-88
solver
XiFoam, U-87
Xoodles, U-87
blockMesh, P-47
boundaryFoam, U-86
bubbleFoam, U-87
buoyantFoam, U-88
buoyantSimpleFoam, U-88
cavitatingFoam, U-87
channelOodles, U-87
coldEngineFoam, U-87

coodles, U-87
dieselEngineFoam, U-87
dieselFoam, U-87
dnsFoam, U-87
electrostaticFoam, U-88
engineFoam, U-87
financialFoam, U-88
icoDyMFoam, U-86
icoFoam, U-19, U-24, U-25, U-28, U-86
interFoam, U-87
laplacianFoam, U-86
lesInterFoam, U-87
mhdFoam, P-71, U-88
multiphaseInterFoam, U-87
nonNewtonianIcoFoam, U-86
oodles, U-87
potentialFoam, P-46, U-86
rasInterFoam, U-87
reactingFoam, U-87
rhoSimpleFoam, U-86
rhoSonicFoam, U-86
rhoTurbFoam, U-86
rhopSonicFoam, U-86
scalarTransportFoam, U-86
settlingFoam, U-87
simpleFoam, P-55, U-86
solidDisplacementFoam, U-51, U-88
solidEquilibriumDisplacementFoam, U-88
sonicFoamAutoMotion, U-86
sonicFoam, P-61, U-86
sonicLiquidFoam, P-65, U-87
sonicTurbFoam, U-87
turbFoam, U-20, U-86
twoLiquidMixingFoam, U-87
twoPhaseEulerFoam, U-87
solver relative tolerance, U-116
solver tolerance, U-116
solvers keyword, U-115
sonicFoam solver, P-61, U-86
sonicFoamAutoMotion solver, U-86
sonicLiquidFoam solver, P-65, U-87
sonicTurbFoam solver, U-87
Source
menu entry, U-28, U-173
source, P-37
SpalartAllmaras model, U-97
specie
library, U-95
specieThermo model, U-96, U-185
spectEddyVisc model, U-97
spline keyword, U-153
splitMesh utility, U-90
splitMeshRegions utility, U-90
sqr
tensor member function, P-25
sqrGradGrad
fvc member function, P-37
Standard Views window panel, U-172
Start Calculation button, U-35
Start Calculation Now button, U-28
startFace keyword, U-142
startFrom keyword, U-25, U-105
starToFoam utility, U-89, U-159
startTime
keyword entry, U-25, U-105
startTime keyword, U-25, U-105
status button, U-131
steady flow
turbulent, P-54
steadyState
keyword entry, U-114
stitchMesh utility, U-90
stopAt keyword, U-105
Stored Camera Position window panel, U-172
streamFunction utility, U-91
stress analysis of plate with hole, U-45
stressComponents utility, U-91
Su
fvm member function, P-37
subsetMesh utility, U-90
summation convention, P-17
SUPERBEE differencing, P-38
supersonic flow, P-59
supersonic flow over forward step, P-59
supersonicFreeStream
boundary condition, U-151
surfaceField<Type> template class, P-33
surfaceNormalFixedValue
boundary condition, U-151
SuSp
fvm member function, P-37
suspend button, U-131
sutherlandTransport model, U-96, U-186
symm
tensor member function, P-25
symmetryPlane
boundary condition, P-65, U-148
symmetryPlane
keyword entry, U-148
symmTensorField class, P-29

symmTensorThirdField class, P-29
system directory, P-49, U-99

T

T()
tensor member function, P-25
template class
GeometricBoundaryField, P-32
fvMatrix, P-36
dimensioned<Type>, P-26
FieldField<Type>, P-32
Field<Type>, P-29
geometricField<Type>, P-32
List<Type>, P-29
pointField<Type>, P-33
surfaceField<Type>, P-33
volField<Type>, P-33
temporal discretisation, P-42
Crank Nicholson, P-43
Euler implicit, P-42
explicit, P-42
in OpenFOAM, P-43
tensor, P-15
addition, P-18
algebraic operations, P-18
algebraic operations in OpenFOAM, P-24
antisymmetric, *see* tensor, skew
calculus, P-27
classes in OpenFOAM, P-23
cofactors, P-22
component average, P-20
component maximum, P-20
component minimum, P-20
determinant, P-22
deviatoric, P-22
diagonal, P-22
dimension, P-16
double inner product, P-19
geometric transformation, P-21
Hodge dual, P-23
hydrostatic, P-22
identities, P-21
identity, P-21
inner product, P-18
inverse, P-23
magnitude, P-20
magnitude squared, P-20
mathematics, P-15
notation, P-17
*n*th power, P-20
outer product, P-19
rank, P-16
rank 3, P-17
scalar division, P-18
scalar multiplication, P-18
scale function, P-20
second rank, P-16
skew, P-22
square of, P-20
subtraction, P-18
symmetric, P-22
symmetric rank 2, P-16
symmetric rank 3, P-17
trace, P-22
transformation, P-21
transpose, P-16, P-22
triple inner product, P-19
vector cross product, P-20
tensor class, P-24
tensor member function
*, P-25
+, P-25
-, P-25
/, P-25
&, P-25
&&, P-25
^, P-25
cmptAv, P-25
cofactors, P-25
det, P-25
dev, P-25
diag, P-25
I, P-25
inv, P-25
mag, P-25
magSqr, P-25
max, P-25
min, P-25
pow, P-25
scale, P-25
skew, P-25
sqr, P-25
symm, P-25
T(), P-25
tr, P-25
transform, P-25
tensorField class, P-29
tensorThirdField class, P-29
tetDecomposition utility, U-90
tetgenToFoam utility, U-89
text box

Case Name, U-127
Case Root, U-127
Lower and Upper Times, U-171
Opacity, U-172
Time step, U-171
times, U-33
thermalProperties
dictionary, U-51, U-52
thermophysical
library, U-185
thermophysicalFunctions
library, U-96
thermophysicalProperties
dictionary, U-185
`thermoType` keyword, U-185
Time window, U-28
time
control, U-105
time derivative, P-37
first, P-39
second, P-37, P-39
Time step text box, U-171
time step, U-25
`timeFormat` keyword, U-106
`timePrecision` keyword, U-107
times text box, U-33
`timeScheme` keyword, U-108
`timeStep`
keyword entry, U-25, U-35, U-106
tolerance
solver, U-116
solver relative, U-116
`tolerance` keyword, U-54, U-115, U-116
tools
adjustPhi, U-94
algorithms, U-93
bound, U-94
compressible, U-94
containers, U-93
db, U-93
dimensionSet, U-93
dimensionedTypes, U-93
fields, U-93
finiteVolume, U-94
global, U-94
incompressible, U-94
interpolations, U-94
matrices, U-94
meshes, U-94
primitives, U-94
wallDist, U-94
`topoSetSource` keyword, U-60
totalPressure
boundary condition, U-151
`tr`
tensor member function, P-25
trace, *see* tensor, trace
`transform`
tensor member function, P-25
transformPoints utility, U-90
transportProperties
dictionary, U-24, U-41, U-43
transportProperties file, U-60
triple inner product, P-19
triSurface
library, U-94
turbFoam solver, U-20, U-86
turbulence
dissipation, U-42
kinetic energy, U-42
length scale, U-42
model, U-43
`turbulence` keyword, U-189
turbulence model, U-42
`turbulenceModel` keyword, U-189
turbulenceProperties
dictionary, U-43, U-188
turbulent flow
steady, P-54
turbulentInlet
boundary condition, U-151
tutorials
breaking of a dam, U-57
lid-driven cavity flow, U-19
stress analysis of plate with hole, U-45
tutorials directory, P-45, U-19
twoLiquidMixingFoam solver, U-87
twoPhaseEulerFoam solver, U-87
`type` keyword, U-146

U

`U` field, U-25
Ucomponents utility, P-72, U-36, U-91
`UMIST`
keyword entry, U-109
`uncompressed`
keyword entry, U-106
`uncorrected`
keyword entry, U-112, U-113
uniform model, U-96
`uniform` keyword, U-181

units
of measurement, P-26
S.I. base, P-26
uprime utility, U-91
`upwind`
keyword entry, U-111, U-114
upwind differencing, P-38, U-62
Use parallel projection button, U-27, U-172
utility
Co, U-91
FoamX, U-88
Lambda2, U-91
Mach, U-91
Pe, U-91
Q, U-91
Rcomponents, U-91
R, U-91
Ucomponents, P-72, U-36, U-91
adiabaticFlameT, U-92
ansysToFoam, U-89
attachMesh, U-89
autoPatch, U-89
blockMesh, U-39, U-89, U-149
boxTurb, U-88
ccm26ToFoam, U-89
cellSet, U-89
cfxToFoam, U-89, U-159
checkMesh, U-89, U-160
checkYPlus, U-92
chemkinToFoam, U-92
couplePatches, U-89
createPatch, U-89
decomposePar, U-82, U-83, U-92
deformedGeom, U-89
divU, U-91
engineCompRatio, U-92
engineSwirl, U-88
ensight74FoamExec, U-178
ensight76FoamExec, U-90
enstrophy, U-91
equilibriumCO, U-92
equilibriumFlameT, U-93
estimateScalarError, U-93
extrudeMesh, U-89
faceSet, U-89
flattenMesh, U-89
flowType, U-91
fluentMeshToFoam, U-89, U-159
foamDataToFluent, U-90, U-175
foamDebugSwitches, U-93
foamInfoExec, U-93
foamMeshToFluent, U-89, U-175
foamToEnsight, U-90
foamToFieldview9, U-90
foamToGMV, U-90
foamToVTK, U-91
gambitToFoam, U-89, U-159
gmshToFoam, U-89
icoErrorEstimate, U-93
icoMomentError, U-93
ideasToFoam, U-159
ideasUnvToFoam, U-89
insideCells, U-89
kivaToFoam, U-89
magGradU, U-91
magU, U-36, U-91
mapFields, U-34, U-40, U-43, U-56, U-88, U-165
mergeMeshes, U-90
mirrorMesh, U-90
mixtureAdiabaticFlameT, U-93
momentScalarError, U-93
moveDynamicMesh, U-90
moveEngineMesh, U-90
moveMesh, U-90
mshToFoam, U-89
netgenNeutralToFoam, U-89
objToVTK, U-90
paraFoam, U-90
patchAverage, U-92
patchIntegrate, U-92
patchTool, U-90
plot3dToFoam, U-89
pointSet, U-90
polyDualMesh, U-89
postChannel, U-92
ptot, U-92
reconstructParMesh, U-92
reconstructPar, U-85, U-92
refineMesh, U-90
renumberMesh, U-90
rotateMesh, U-90
sammToFoam, U-89
sampleSurface, U-92
sample, U-92, U-179
scalePoints, U-163
setFields, U-59, U-60, U-88
smapToFoam, U-91
splitMeshRegions, U-90
splitMesh, U-90

starToFoam, U-89, U-159
stitchMesh, U-90
streamFunction, U-91
stressComponents, U-91
subsetMesh, U-90
tetDecomposition, U-90
tetgenToFoam, U-89
transformPoints, U-90
uprime, U-91
vorticity, U-91
wallGradU, U-92
wallHeatFlux, U-92
wallShearStress, U-92
wdot, U-92
writeCellCentres, U-92
writeMeshObj, U-89
yPlusLES, U-92
zipUpMesh, U-90

V

value keyword, U-150
valueFraction keyword, U-150
van Leer differencing, P-38
vanLeer
keyword entry, U-111
vector, P-16
operator, P-27
unit, P-20
vector class, P-24, U-103
vector product, *see* tensor, vector cross product
vectorField class, P-29
version keyword, U-101
vertices keyword, U-22, U-153
veryInhomogeneousMixture model, U-95, U-186
View menu, U-28, U-172, U-173
viscosity
kinematic, U-24, U-43
volField<Type> template class, P-33
vorticity utility, U-91
vtkFoam
library, U-169

W

wall
boundary condition, P-65, P-71, U-148
wall boundary type, U-42
wall
keyword entry, U-148
wall function, U-96, U-97
wallBuoyantPressure
boundary condition, U-151
wallDist tools, U-94
wallFunctionCoeffs keyword, U-189
wallGradU utility, U-92
wallHeatFlux utility, U-92
wallShearStress utility, U-92
water model, U-95
wclean script/alias, U-76
wdot utility, U-92
wedge
boundary condition, U-143, U-148, U-149, U-158
wedge
keyword entry, U-148
window
Fields, U-28
Region, U-28
Seed, U-175
Selection Window, U-27, U-170
Time, U-28
window panel
Annotate, U-27, U-172
Camera Controls, U-172
Camera Orientation, U-172
Camera, U-172
Display, U-27, U-28, U-170, U-172
General, U-172
Information, U-170
Parameters, U-28, U-170, U-171
Standard Views, U-172
Stored Camera Position, U-172
Wireframe
menu entry, U-172
WM_ARCH
environment variable, U-76
WM_COMPILE_OPTION
environment variable, U-76
WM_COMPILER
environment variable, U-76
WM_COMPILER_BIN
environment variable, U-76
WM_COMPILER_DIR
environment variable, U-76
WM_COMPILER_LIB
environment variable, U-76
WM_DIR
environment variable, U-76
WM_JAVAC_OPTION
environment variable, U-76
WM_LINK_LANGUAGE
environment variable, U-76

WM_MPLIB
environment variable, U-76
WM_OPTIONS
environment variable, U-76
WM_PROJECT
environment variable, U-76
WM_PROJECT_DIR
environment variable, U-76
WM_PROJECT_INST_DIR
environment variable, U-76
WM_PROJECT_LANGUAGE
environment variable, U-76
WM_PROJECT_USER_DIR
environment variable, U-76
WM_PROJECT_VERSION
environment variable, U-76
WM_SHELL
environment variable, U-76
wmake
platforms, U-73
wmake script/alias, U-71
word class, P-26, P-31
writeCellCentres utility, U-92
writeCompression keyword, U-106
writeControl
keyword entry, U-106
writeControl keyword, U-25, U-61, U-106
writeFormat keyword, U-55, U-106
writeInterval keyword, U-25, U-35, U-106
writeMeshObj utility, U-89
writeNow
keyword entry, U-105
writePrecision keyword, U-106

X

x
keyword entry, U-181
XiFoam solver, U-87
xmgr
keyword entry, U-107, U-180
Xoodles solver, U-87
xyz
keyword entry, U-181

Y

y
keyword entry, U-181
yPlusLES utility, U-92

Z

z
keyword entry, U-181
zeroGradient
boundary condition, U-150
zipUpMesh utility, U-90
zlib-1.2.1
library, U-94