License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE (“CCPL” OR “LICENSE”). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

   a. “Adaptation” means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image (“synching”) will be considered an Adaptation for the purpose of this License.

   b. “Collection” means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

   c. “Distribute” means to make available to the public the original and copies of the Work through sale or other transfer of ownership.

   d. “Licensor” means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

   e. “Original Author” means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the
publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

f. “Work” means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

g. “You” means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. “Publicly Perform” means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. “Reproduce” means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights.

Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant.

Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;
b. and, to Distribute and Publicly Perform the Work including as incorporated in Collections.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).

4. Restrictions.

The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.

b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution (“Attribution Parties”) in Licensor’s copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or
Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. For the avoidance of doubt:

   i. **Non-waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

   ii. **Waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

   iii. **Voluntary License Schemes.** The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b).

e. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author’s honor or reputation.

5. **Representations, Warranties and Disclaimer**

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. **Limitation on Liability.**

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. **Termination**

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections
from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You.

e. This License may not be modified without the mutual written agreement of the Licensor and You. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.
Trademarks

ANSYS is a registered trademark of ANSYS Inc.
CFX is a registered trademark of Ansys Inc.
CHEMKIN is a registered trademark of Reaction Design Corporation
EnSight is a registered trademark of Computational Engineering International Ltd.
Fieldview is a registered trademark of Intelligent Light
Fluent is a registered trademark of Ansys Inc.
GAMBIT is a registered trademark of Ansys Inc.
Icem-CFD is a registered trademark of Ansys Inc.
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
OpenFOAM is a registered trademark of ESI Group
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
1. Definitions ........................................ U-2
2. Fair Dealing Rights ................................ U-3
3. License Grant ...................................... U-3
4. Restrictions ........................................ U-4
5. Representations, Warranties and Disclaimer .... U-5
7. Termination ......................................... U-5
8. Miscellaneous ...................................... U-6

### Trademarks

### Contents

1  Introduction ....................................... U-15

2  Tutorials .......................................... U-17

2.1  Lid-driven cavity flow ............................ U-17

2.1.1  Pre-processing ................................. U-18

2.1.1.1  Mesh generation ............................. U-18
2.1.1.2  Boundary and initial conditions ........ U-20
2.1.1.3  Physical properties ....................... U-21
2.1.1.4  Control ..................................... U-22
2.1.1.5  Discretisation and linear-solver settings . U-23

2.1.2  Viewing the mesh .............................. U-23

2.1.3  Running an application ....................... U-24

2.1.4  Post-processing ............................... U-25

2.1.4.1  Isosurface and contour plots ............. U-25
2.1.4.2  Vector plots ................................ U-27
2.1.4.3  Streamline plots ........................... U-29

2.1.5  Increasing the mesh resolution ............... U-29

2.1.5.1  Creating a new case using an existing case U-29
2.1.5.2  Creating the finer mesh .................... U-31
2.1.5.3  Mapping the coarse mesh results onto the fine mesh . U-31
2.1.5.4  Control adjustments ......................... U-32
2.1.5.5  Running the code as a background process U-32
2.1.5.6  Vector plot with the refined mesh .......... U-32
2.1.5.7  Plotting graphs ............................. U-33
2.1.6 Introducing mesh grading .............................................. U-35
  2.1.6.1 Creating the graded mesh ...................................... U-36
  2.1.6.2 Changing time and time step .................................. U-37
  2.1.6.3 Mapping fields .................................................. U-38
2.1.7 Increasing the Reynolds number ...................................... U-38
  2.1.7.1 Pre-processing .................................................... U-39
  2.1.7.2 Running the code ................................................. U-39
2.1.8 High Reynolds number flow ......................................... U-40
  2.1.8.1 Pre-processing .................................................... U-40
  2.1.8.2 Running the code ................................................. U-42
2.1.9 Changing the case geometry ........................................ U-42
2.1.10 Post-processing the modified geometry .......................... U-46
2.2 Stress analysis of a plate with a hole ............................... U-46
  2.2.1 Mesh generation .................................................... U-47
    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-51
    2.2.1.5 Discretisation schemes and linear-solver control ....... U-52
  2.2.2 Running the code .................................................... U-54
  2.2.3 Post-processing .................................................... U-54
  2.2.4 Exercises ........................................................... U-55
    2.2.4.1 Increasing mesh resolution ................................. U-56
    2.2.4.2 Introducing mesh grading .................................... U-56
    2.2.4.3 Changing the plate size ...................................... U-56
2.3 Breaking of a dam ....................................................... U-56
  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 Turbulence modelling .............................................. U-61
  2.3.6 Time step control .................................................. U-62
  2.3.7 Discretisation schemes ............................................ U-62
  2.3.8 Linear-solver control ............................................. U-63
  2.3.9 Running the code .................................................... U-64
  2.3.10 Post-processing ..................................................... U-64
  2.3.11 Running in parallel ............................................... U-64
  2.3.12 Post-processing a case run in parallel ....................... U-67

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
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.2.1 Including headers</td>
<td>U-73</td>
</tr>
<tr>
<td>3.2.2.2 Linking to libraries</td>
<td>U-74</td>
</tr>
<tr>
<td>3.2.2.3 Source files to be compiled</td>
<td>U-75</td>
</tr>
<tr>
<td>3.2.2.4 Running <code>wmake</code></td>
<td>U-75</td>
</tr>
<tr>
<td>3.2.2.5 <code>wmake</code> environment variables</td>
<td>U-76</td>
</tr>
<tr>
<td>3.2.3 Removing dependency lists: <code>wclean</code> and <code>rmdepall</code></td>
<td>U-76</td>
</tr>
<tr>
<td>3.2.4 Compilation example: the <code>pisoFoam</code> application</td>
<td>U-77</td>
</tr>
<tr>
<td>3.2.5 Debug messaging and optimisation switches</td>
<td>U-79</td>
</tr>
<tr>
<td>3.2.6 Linking new user-defined libraries to existing applications</td>
<td>U-80</td>
</tr>
<tr>
<td>3.3 Running applications</td>
<td>U-81</td>
</tr>
<tr>
<td>3.4 Running applications in parallel</td>
<td>U-81</td>
</tr>
<tr>
<td>3.4.1 Decomposition of mesh and initial field data</td>
<td>U-82</td>
</tr>
<tr>
<td>3.4.2 Running a decomposed case</td>
<td>U-84</td>
</tr>
<tr>
<td>3.4.3 Distributing data across several disks</td>
<td>U-84</td>
</tr>
<tr>
<td>3.4.4 Post-processing parallel processed cases</td>
<td>U-85</td>
</tr>
<tr>
<td>3.4.4.1 Reconstructing mesh and data</td>
<td>U-85</td>
</tr>
<tr>
<td>3.4.4.2 Post-processing decomposed cases</td>
<td>U-85</td>
</tr>
<tr>
<td>3.5 Standard solvers</td>
<td>U-85</td>
</tr>
<tr>
<td>3.6 Standard utilities</td>
<td>U-90</td>
</tr>
<tr>
<td>3.7 Standard libraries</td>
<td>U-97</td>
</tr>
</tbody>
</table>

4 OpenFOAM cases

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 File structure of OpenFOAM cases</td>
<td>U-105</td>
</tr>
<tr>
<td>4.2 Basic input/output file format</td>
<td>U-106</td>
</tr>
<tr>
<td>4.2.1 General syntax rules</td>
<td>U-106</td>
</tr>
<tr>
<td>4.2.2 Dictionaries</td>
<td>U-107</td>
</tr>
<tr>
<td>4.2.3 The data file header</td>
<td>U-107</td>
</tr>
<tr>
<td>4.2.4 Lists</td>
<td>U-108</td>
</tr>
<tr>
<td>4.2.5 Scalars, vectors and tensors</td>
<td>U-109</td>
</tr>
<tr>
<td>4.2.6 Dimensional units</td>
<td>U-109</td>
</tr>
<tr>
<td>4.2.7 Dimensioned types</td>
<td>U-110</td>
</tr>
<tr>
<td>4.2.8 Fields</td>
<td>U-110</td>
</tr>
<tr>
<td>4.2.9 Directives and macro substitutions</td>
<td>U-111</td>
</tr>
<tr>
<td>4.2.10 The <code>#include</code> and <code>#inputMode</code> directives</td>
<td>U-112</td>
</tr>
<tr>
<td>4.2.11 The <code>#codeStream</code> directive</td>
<td>U-112</td>
</tr>
<tr>
<td>4.3 Time and data input/output control</td>
<td>U-113</td>
</tr>
<tr>
<td>4.4 Numerical schemes</td>
<td>U-116</td>
</tr>
<tr>
<td>4.4.1 Interpolation schemes</td>
<td>U-117</td>
</tr>
<tr>
<td>4.4.1.1 Schemes for strictly bounded scalar fields</td>
<td>U-118</td>
</tr>
<tr>
<td>4.4.1.2 Schemes for vector fields</td>
<td>U-118</td>
</tr>
<tr>
<td>4.4.2 Surface normal gradient schemes</td>
<td>U-119</td>
</tr>
<tr>
<td>4.4.3 Gradient schemes</td>
<td>U-120</td>
</tr>
<tr>
<td>4.4.4 Laplacian schemes</td>
<td>U-120</td>
</tr>
<tr>
<td>4.4.5 Divergence schemes</td>
<td>U-121</td>
</tr>
<tr>
<td>4.4.6 Time schemes</td>
<td>U-122</td>
</tr>
<tr>
<td>4.4.7 Flux calculation</td>
<td>U-122</td>
</tr>
<tr>
<td>4.5 Solution and algorithm control</td>
<td>U-123</td>
</tr>
<tr>
<td>4.5.1 Linear solver control</td>
<td>U-123</td>
</tr>
</tbody>
</table>
4.5.1.1 Solution tolerances ........................................... U-124
4.5.1.2 Preconditioned conjugate gradient solvers ................. U-125
4.5.1.3 Smooth solvers ............................................... U-125
4.5.1.4 Geometric-algebraic multi-grid solvers ....................... U-125
4.5.2 Solution under-relaxation ..................................... U-126
4.5.3 PISO and SIMPLE algorithms .................................. U-127
4.5.3.1 Pressure referencing ........................................ U-127
4.5.4 Other parameters .............................................. U-128

5 Mesh generation and conversion .................................. U-129
5.1 Mesh description .................................................. U-129
5.1.1 Mesh specification and validity constraints ..................... U-129
5.1.1.1 Points .......................................................... U-130
5.1.1.2 Faces ........................................................... U-130
5.1.1.3 Cells ............................................................. U-131
5.1.1.4 Boundary ....................................................... U-131
5.1.2 The polyMesh description ...................................... U-131
5.1.3 The cellShape tools ............................................. U-132
5.1.4 1- and 2-dimensional and axi-symmetric problems ............ U-133
5.2 Boundaries ......................................................... U-133
5.2.1 Specification of patch types in OpenFOAM ...................... U-135
5.2.2 Base types ........................................................ U-136
5.2.3 Primitive types ................................................ U-138
5.2.4 Derived types .................................................... U-138
5.3 Mesh generation with the blockMesh utility ....................... U-140
5.3.1 Writing a blockMeshDict file ................................ U-140
5.3.1.1 The vertices .................................................. U-141
5.3.1.2 The edges ..................................................... U-142
5.3.1.3 The blocks .................................................... U-142
5.3.1.4 Multi-grading of a block .................................. U-143
5.3.1.5 The boundary ................................................ U-145
5.3.2 Multiple blocks .................................................. U-146
5.3.3 Creating blocks with fewer than 8 vertices ..................... U-148
5.3.4 Running blockMesh ............................................. U-149
5.4 Mesh generation with the snappyHexMesh utility ................. U-149
5.4.1 The mesh generation process of snappyHexMesh ............... U-149
5.4.2 Creating the background hex mesh ................................ U-151
5.4.3 Cell splitting at feature edges and surfaces .................... U-152
5.4.4 Cell removal ..................................................... U-154
5.4.5 Cell splitting in specified regions ............................. U-154
5.4.6 Snapping to surfaces ........................................... U-155
5.4.7 Mesh layers ..................................................... U-155
5.4.8 Mesh quality controls ......................................... U-158
5.5 Mesh conversion ..................................................... U-158
5.5.1 fluentMeshToFoam ............................................. U-159
5.5.2 starToFoam ....................................................... U-160
5.5.2.1 General advice on conversion ............................... U-160
5.5.2.2 Eliminating extraneous data ................................ U-160
5.5.2.3 Removing default boundary conditions ................................ U-161
5.5.2.4 Renumbering the model ....................................................... U-162
5.5.2.5 Writing out the mesh data .................................................... U-163
5.5.2.6 Problems with the .vrt file ................................................... U-163
5.5.2.7 Converting the mesh to OpenFOAM format ............................. U-164
5.5.3 gambitToFoam ................................................................. U-164
5.5.4 ideasToFoam ................................................................. U-164
5.5.5 cfx4ToFoam ................................................................. U-165
5.6 Mapping fields between different geometries ................................. U-165
5.6.1 Mapping consistent fields ....................................................... U-165
5.6.2 Mapping inconsistent fields .................................................... U-166
5.6.3 Mapping parallel cases ......................................................... U-167

6 Post-processing ......................................................................... U-169
6.1 paraFoam ............................................................................. U-169
6.1.1 Overview of paraFoam ......................................................... U-169
6.1.2 The Parameters panel ............................................................ U-171
6.1.3 The Display panel ................................................................. U-172
6.1.4 The button toolbars ............................................................... U-173
6.1.5 Manipulating the view ......................................................... U-173
6.1.5.1 View settings ................................................................. U-173
6.1.5.2 General settings ............................................................... U-174
6.1.6 Contour plots ................................................................. U-174
6.1.6.1 Introducing a cutting plane ................................................. U-174
6.1.7 Vector plots ................................................................. U-174
6.1.7.1 Plotting at cell centres .................................................... U-175
6.1.8 Streamlines ................................................................. U-175
6.1.9 Image output ................................................................. U-175
6.1.10 Animation output .............................................................. U-175
6.2 Function Objects .................................................................... U-176
6.2.1 Using function objects ......................................................... U-177
6.2.2 Packaged function objects .................................................... U-179
6.3 Post-processing with Fluent .................................................. U-181
6.4 Post-processing with Fieldview ............................................... U-182
6.5 Post-processing with EnSight .................................................. U-183
6.5.1 Converting data to EnSight format ........................................ U-183
6.5.2 The ensight74FoamExec reader module ................................. U-183
6.5.2.1 Configuration of EnSight for the reader module .................... U-183
6.5.2.2 Using the reader module .................................................. U-184
6.6 Sampling data ......................................................................... U-184
6.7 Monitoring and managing jobs ................................................. U-186
6.7.1 The foamJob script for running jobs ...................................... U-188
6.7.2 The foamLog script for monitoring jobs .................................. U-188

7 Models and physical properties .................................................. U-191
7.1 Thermophysical models .......................................................... U-191
7.1.1 Thermophysical and mixture models ...................................... U-192
7.1.2 Transport model ............................................................... U-193
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1.3</td>
<td>Thermodynamic models</td>
<td>U-193</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Composition of each constituent</td>
<td>U-194</td>
</tr>
<tr>
<td>7.1.5</td>
<td>Equation of state</td>
<td>U-195</td>
</tr>
<tr>
<td>7.1.6</td>
<td>Selection of energy variable</td>
<td>U-196</td>
</tr>
<tr>
<td>7.1.7</td>
<td>Thermophysical property data</td>
<td>U-196</td>
</tr>
<tr>
<td>7.2</td>
<td>Turbulence models</td>
<td>U-197</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Model coefficients</td>
<td>U-198</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Wall functions</td>
<td>U-198</td>
</tr>
<tr>
<td>7.3</td>
<td>Transport/rheology models</td>
<td>U-199</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Newtonian model</td>
<td>U-199</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Bird-Carreau model</td>
<td>U-200</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Cross Power Law model</td>
<td>U-200</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Power Law model</td>
<td>U-200</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Herschel-Bulkley model</td>
<td>U-201</td>
</tr>
</tbody>
</table>

Index                  | U-203 |
Chapter 1

Introduction

This guide accompanies the release of version 3.0.1 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.

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.](image)

In chapter 5, 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 6.
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 blockMesh pre-processing tool, case setup and 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 6 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 directories according to the type of flow and then subdirectories according to solver. For example, all the icoFoam cases are stored within a subdirectory incompressible/icoFoam, where incompressible indicates the type of flow. If the user wishes to run a range of example cases, it is recommended that the user copy the tutorials directory into their local run directory. They can be easily copied by typing:

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

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 pisoFoam solver will be used for turbulent, isothermal, incompressible flow.
2.1.1 Pre-processing

Cases are setup in OpenFOAM by editing case files. Users should select an xeditor of choice with which to do this, such as emacs, vi, gedit, kate, nedit, etc. Editing files is possible in OpenFOAM because the I/O uses a dictionary format with keywords that convey sufficient meaning to be understood by even the least experienced users.

A case being simulated involves data for mesh, fields, properties, control parameters, etc. As described in section 4.1, in OpenFOAM this data is stored in a set of files within a case directory rather than in a single case file, as in many other CFD packages. The case directory is given a suitably descriptive name, e.g. the first example case for this tutorial is simply named cavity. In preparation of editing case files and running the first cavity case, the user should change to the case directory

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
```

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 2.2. The mesh generator supplied with OpenFOAM, blockMesh, generates meshes from a description specified in an input dictionary, blockMeshDict located in the system (or constant/polyMesh) directory for a given case. The blockMeshDict entries for this case are as follows:

```plaintext
/*--------------------------------*- C++ -*----------------------------------*
 | ========= | |
 | \ / F ield | OpenFOAM: The Open Source CFD Toolbox |
 | \ / O peration | Version: 3.0.1 |
 | \ / A nd | Web: www.OpenFOAM.org |
 | \/ M anipulation |

OpenFOAM-3.0.1
```
Figure 2.2: Block structure of the mesh for the cavity.

```
FoamFile
{
  version     2.0;
  format      ascii;
  class       dictionary;
  object      blockMeshDict;
}

// ***************************************************************************
vertices
(
  (0 0 0)
  (1 0 0)
  (1 1 0)
  (0 1 0)
  (0 0 0.1)
  (1 0 0.1)
  (1 1 0.1)
  (0 1 0.1)
);

blocks
(
  hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
  movingWall
  {
    type      wall;
    faces
    (
      (3 7 6 2)
    );
  }
  fixedWalls
  {
    type      wall;
    faces
    (
      (0 4 7 3)
      (2 6 5 1)
      (1 5 4 0)
    );
  }
)
```
The file first contains header information in the form of a banner (lines 1-7), then file information contained in a FoamFile sub-dictionary, delimited by curly braces ({ ... }).

For the remainder of the manual:

For the sake of clarity and to save space, file headers, including the banner and FoamFile sub-dictionary, will be removed from verbatim quoting of case files.

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 5.3 to understand the meaning of the entries in the blockMeshDict file.

The mesh is generated by running blockMesh on this blockMeshDict file. From within the case directory, this is done, simply by typing in the terminal:

   blockMesh

The running status of blockMesh is reported in the terminal window. 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, the user can look at this initial fields set up for this case. The case is set up to start at time $t = 0$ s, so the initial field data is stored in a 0 sub-directory of the cavity directory. The 0 sub-directory contains 2 files, $p$ and $U$, one for each of the pressure ($p$) and velocity ($U$) fields whose initial values and boundary conditions must be set. Let us examine file $p$:

   dimensions [0 2 -2 0 0 0 0];
   internalField uniform 0;
   boundaryField
      movingWall
      { type zeroGradient; }
      fixedWalls
      { type zeroGradient; }

// ************************************************************************* //
There are 3 principal entries in field data files:

dimensions specifies the dimensions of the field, here kinematic pressure, \( i.e. \) \( \text{m}^2\text{s}^{-2} \) (see section 4.2.6 for more information);

internalField the internal field data which can be uniform, described by a single value; or nonuniform, where all the values of the field must be specified (see section 4.2.8 for more information);

boundaryField the boundary field data that includes boundary conditions and data for all the boundary patches (see section 4.2.8 for more information).

For this case cavity, the boundary consists of walls only, split into 2 patches named: (1) fixedWalls for the fixed sides and base of the cavity; (2) movingWall for the moving top of the cavity. As walls, both are given a zeroGradient boundary condition for \( p \), meaning “the normal gradient of pressure is zero”. The frontAndBack patch represents the front and back planes of the 2D case and therefore must be set as empty.

In this case, as in most we encounter, the initial fields are set to be uniform. Here the pressure is kinematic, and as an incompressible case, its absolute value is not relevant, so is set to uniform 0 for convenience.

The user can similarly examine the velocity field in the \( 0/U \) file. The dimensions are those expected for velocity, the internal field is initialised as uniform zero, which in the case of velocity must be expressed by 3 vector components, \( i.e. \) uniform \((0 0 0)\) (see section 4.2.5 for more information).

The boundary field for velocity requires the same boundary condition for the frontAndBack patch. The other patches are walls: a no-slip condition is assumed on the fixedWalls, hence a fixedValue condition with a value of uniform \((0 0 0)\). The top surface moves at a speed of 1 m/s in the \( x \)-direction so requires a fixedValue condition also but with uniform \((1 0 0)\).

2.1.1.3 Physical properties

The physical properties for the case are stored in dictionaries whose names are given the suffix \( \ldots \text{Properties} \), 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 can check that the kinematic viscosity is set correctly by opening the transportProperties dictionary to view/edit its entries. The keyword for kinematic viscosity is \( \text{nu} \), the phonetic label for the Greek symbol \( \nu \) 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:

\[
Re = \frac{d|U|}{\nu}
\]  

(2.1)
where $d$ and $|\mathbf{U}|$ are the characteristic length and velocity respectively and $\nu$ is the kinematic viscosity. Here $d = 0.1$ m, $|\mathbf{U}| = 1$ m s$^{-1}$, so that for $Re = 10$, $\nu = 0.01$ m$^2$ s$^{-1}$. The correct file entry for kinematic viscosity is thus specified below:

```bash
nu [0 2 -1 0 0 0 0] 0.01;
```

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. The user should view this file; as a case control file, it is located in the `system` directory.

The start/stop times and the time step for the run must be set. 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 should 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:

$$Co = \frac{\delta t |\mathbf{U}|}{\delta x} \quad (2.2)$$

where $\delta t$ is the time step, $|\mathbf{U}|$ is the magnitude of the velocity through that cell and $\delta x$ 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 $\delta t$ 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 m s$^{-1}$. The cell size is:

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

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:

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

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 \texttt{timeStep} option which specifies that results are written every $n$th time step where the value $n$ is specified under the \texttt{writeInterval} keyword. Let us decide that we wish to write our results at times 0.1, 0.2, \ldots, 0.5 s. With a time step of 0.005 s, we therefore need to output results at every 20th time step and so we set \texttt{writeInterval} to 20.

OpenFOAM creates a new directory \textit{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 \texttt{icoFoam} solver, it writes out the results for each field, $U$ and $p$, into the time directories. For this case, the entries in the \texttt{controlDict} are shown below:

\begin{verbatim}
application icoFoam;
startFrom startTime;
startTime 0;
stopAt endTime;
endTime 0.5;
deltaT 0.005;
writeControl timeStep;
writeInterval 20;
purgeWrite 0;
writeFormat ascii;
writePrecision 6;
writeCompression off;
timeFormat general;
timePrecision 6;
runTimeModifiable true;

// ************************************************************************* //
\end{verbatim}

2.1.1.5 Discretisation and linear-solver settings

The user specifies the choice of finite volume discretisation schemes in the \texttt{fvSchemes} dictionary in the \texttt{system} directory. The specification of the linear equation solvers and tolerances and other algorithm controls is made in the \texttt{fvSolution} dictionary, similarly in the \texttt{system} directory. The user is free to view these dictionaries but we do not need to discuss all their entries at this stage except for \texttt{pRefCell} and \texttt{pRefValue} in the \texttt{PISO} sub-dictionary of the \texttt{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 \texttt{pRefValue} in cell \texttt{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 pressures or velocity field.

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 \texttt{paraFoam}, the post-processing tool supplied with OpenFOAM. The \texttt{paraFoam} post-processing is started by typing in the terminal from within the case directory
Alternatively, it can be launched from another directory location with an optional \texttt{-case} argument giving the case directory, \textit{e.g.}

\begin{verbatim}
paraFoam -case $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
\end{verbatim}

This launches the \textit{ParaView} window as shown in Figure 6.1. In the Pipeline Browser, the user can see that \textit{ParaView} has opened \texttt{cavity.OpenFOAM}, the module for the \texttt{cavity} case. \textbf{Before clicking the Apply button}, the user needs to select some geometry from the \texttt{Mesh Parts} panel. Because the case is small, it is easiest to select all the data by checking the box adjacent to the \texttt{Mesh Parts} panel title, which automatically checks all individual components within the respective panel. The user should then click the Apply button to load the geometry into \textit{ParaView}.

The user should then scroll down to the \texttt{Display} panel that controls the visual representation of the selected module. Within the \texttt{Display} panel the user should do the following as shown in Figure 2.3: (1) set \texttt{Coloring Solid Color}; (2) click \texttt{Set Ambient Color} and select an appropriate colour \textit{e.g.} black (for a white background); (3) select \texttt{Wireframe} from the \texttt{Representation} menu. The background colour can be set in the \texttt{View Render} panel below the \texttt{Display} panel in the \texttt{Properties} window.

Especially the first time the user starts \textit{ParaView}, it is recommended that they manipulate the view as described in section 6.1.5. In particular, since this is a 2D case, it is recommended that \texttt{Use Parallel Projection} is selected near the bottom of the \texttt{View Render} panel, available only with the \texttt{Advanced Properties} gearwheel button pressed at the top of the \texttt{Properties} window, next to the search box. \texttt{View Settings} window selected from the \texttt{Edit} menu. The \texttt{Orientation Axes} can be toggled on and off in the \texttt{Annotation} window or moved by drag and drop with the mouse.

\subsection{Running an application}

Like any UNIX/Linux executable, OpenFOAM applications can be run in two ways: as a foreground process, \textit{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 \texttt{icoFoam} in the foreground. The \texttt{icoFoam} solver is executed either by entering the case directory and typing

\begin{verbatim}
icoFoam
\end{verbatim}

at the command prompt, or with the optional \texttt{-case} argument giving the case directory, \textit{e.g.}

\begin{verbatim}
icoFoam -case $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
\end{verbatim}

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.
2.1 Lid-driven cavity flow

2.1.4 Post-processing

As soon as results are written to time directories, they can be viewed using \textit{paraFoam}. Return to the \textit{paraFoam} window and select the \textit{Properties} panel for the \textit{cavity}.\textit{OpenFOAM} case module. If the correct window panels for the case module do not seem to be present at any time, please ensure that: \textit{cavity}.\textit{OpenFOAM} is highlighted in blue; \textit{eye} button alongside it is switched on to show the graphics are enabled;

To prepare \textit{paraFoam} to display the data of interest, we must first load the data at the required run time of 0.5 s. If the case was run while \textit{ParaView} was open, the output data in time directories will not be automatically loaded within \textit{ParaView}. To load the data the user should click \textit{Refresh Times} in the \textit{Properties} window. The time data will be loaded into \textit{ParaView}.

2.1.4.1 Isosurface and contour plots

To view pressure, the user should go to the \textit{Display} panel since it controls the visual representation of the selected module. To make a simple plot of pressure, the user should select the following, as described in detail in Figure 2.4: select \textit{Surface} from the \textit{Representation} menu; select \textit{Solid Color} in \textit{Coloring} and \textit{Rescale to Data Range}. Now in order to view the solution at $t = 0.5$ s, the user can use the \textit{VCR Controls} or \textit{Current Time Controls} to change the
Figure 2.4: Displaying pressure contours for the cavity case.

Figure 2.5: Pressures in the cavity case.
current time to 0.5. These are located in the toolbars below the menus at the top of the 
ParaView window, as shown in Figure 6.4. The pressure field solution 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 2.5.

With the point icon ( ) the pressure field is interpolated across each cell to give a 
continuous appearance. Instead if the user selects the cell icon, , from the Color by 
menu, 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 either by clicking the Toggle Color Legend Visibility button 
in the Active Variable Controls toolbar or the Coloring section of the Display panel. 
Clicking the Edit Color Map button, either in the Active Variable Controls toolbar or in 
the Coloring panel of the Display panel, 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.

ParaView defaults to using a colour scale of blue to white to red rather than the more 
common blue to green to red (rainbow). Therefore the first time that the user executes 
ParaView, they may wish to change the colour scale. This can be done by selecting the 
Choose Preset button (with the heart icon) in the Color Scale Editor and selecting Blue to 
Red Rainbow. After clicking the OK confirmation button, the user can click the Make Default 
button so that ParaView will always adopt this type of colour bar.

If the user rotates the image, they can see that they have now coloured the complete 
geometry surface by the pressure. In order to produce a genuine contour plot the user 
should first create a cutting plane, or ‘slice’, through the geometry using the Slice filter as 
described in section 6.1.6.1. The cutting plane should be centred at (0.05, 0.05, 0.005) and its 
normal should be set to (0, 0, 1) (click the Z Normal button). Having generated the cutting 
plane, the contours can be created using by the Contour filter described in section 6.1.6.

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 Slice and Contour filters described above. 
These can: either be deleted entirely, by highlighting the relevant module in the Pipeline 
Browser and clicking Delete in their respective Properties panel; or, be disabled by toggling 
the eye button for the relevant module in the Pipeline Browser.

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 6.1.7.1. With the cavity.OpenFOAM 
module highlighted in the Pipeline Browser, the user should select Cell Centers from the 
Filter->Alphabetical menu and then click Apply.

With these Centers highlighted in the Pipeline Browser, the user should then select Glyph 
from the Filter->Common menu. The Properties window panel should appear as shown in 
Figure 2.6. Note that newly selected filters are moved to the Filter->Recent menu and are unavailable in the menus from where they were originally selected. In the resulting 
Properties 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 off and Set Scale Factor to 0.005. On clicking Apply, 
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 U in the
Open Properties panel
Specify Set Scale Factor 0.005
Select Scale Mode off
Select Glyph Type Arrow

Figure 2.6: Properties panel for the Glyph filter.

Figure 2.7: Velocities in the cavity case.
2.1 Lid-driven cavity flow

Display panel. The user should also select Show Color Legend in Edit Color Map. The output is shown in Figure 2.7, in which uppercase Times Roman fonts are selected for the Color Legend headings and the labels are specified to 2 fixed significant figures by deselecting Automatic Label Format and entering %#6.2f in the Label Format text box. The background colour is set to white in the General panel of View Settings as described in section 6.1.5.1.

Note that at the left and right walls, glyphs appear to indicate flow through the walls. On closer examination, however, the user can see that while the flow direction is normal to the wall, its magnitude is 0. This slightly confusing situation is caused by ParaView choosing to orientate the glyphs in the $x$-direction when the glyph scaling off and the velocity magnitude is 0.

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 streamlines of velocity as described in section 6.1.8.

With the cavity.OpenFOAM module highlighted in the Pipeline Browser, the user should then select Stream Tracer from the Filter menu and then click Apply. The Properties window panel should appear as shown in Figure 2.8. The Seed points should be specified along a High Resolution Line Source 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; Maximum Step Length of 0.5; Initial Step Length of 0.2; and, Integration Direction BOTH. The Runge-Kutta 4/5 IntegratorType was used with default parameters.

On clicking Apply 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 6; Radius 0.0003; and, Radius factor 10. The streamtubes are coloured by velocity magnitude. On clicking Apply the image in Figure 2.9 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

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. First the user should create a new case directory at the same directory level as the cavity case, e.g.

```bash
    cd $FOAM_RUN/tutorials/incompressible/icoFoam
    mkdir cavityFine
```

The user should then copy the base directories from the cavity case into cavityFine, and then enter the cavityFine case.

```bash
    cp -r cavity/constant cavityFine
```
Scroll to Properties title
Set Initial Step Length to Cell Length 0.01
Set Integration Direction to BOTH
Specify Line Source and set points and resolution

Figure 2.8: Properties panel for the Stream Tracer filter.

Figure 2.9: Streamlines in the cavity case.
2.1 Lid-driven cavity flow

```bash
cp -r cavity/system cavityFine
cd cavityFine
```

2.1.5.2 Creating the finer mesh

We now wish to increase the number of cells in the mesh by using `blockMesh`. The user should open the `blockMeshDict` file in an editor and edit the block specification. The blocks are specified in a list under the `blocks` keyword. The syntax of the block definitions is described fully in section 5.3.1.3; at this stage it is sufficient to know that following `hex` is first the list of vertices in the block, then a list (or vector) of numbers of cells in each direction. This was originally set to `(20 20 1)` for the `cavity` case. The user should now change this to `(40 40 1)` and save the file. The new refined mesh should then be created by running `blockMesh` as before.

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` and `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 case is ready to run `mapFields`. Typing `mapFields -help` quickly shows that `mapFields` requires the source case directory as an argument. We are using the `-consistent` option, so the utility is executed from within the `cavityFine` directory by

```bash
mapFields ../cavity -consistent
```

The utility should run with output to the terminal including:

```
Source: ".." "cavity"
Target: "." "cavityFine"

Create databases as time
Case : ../cavity
nProcs : 1

Source time: 0.5
Target time: 0.5

Create meshes
Source mesh size: 400  Target mesh size: 1600

Consistently creating and mapping fields for time 0.5

Creating mesh-to-mesh addressing ...
  Overlap volume: 0.0001
Creating AMI between source patch movingWall and target patch movingWall ...
```
interpolating p
interpolating U

End

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 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 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 file.

2.1.5.5 Running the code as a background process

The user should experience running icoFoam as a background process, redirecting the terminal output to a log file that can be viewed later. From the cavityFine directory, the user should execute:

```
icoFoam > log &
cat log
```

2.1.5.6 Vector plot with the refined mesh

The user can open multiple cases simultaneously in ParaView; essentially because each new case is simply another module that appears in the Pipeline Browser. 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 .OpenFOAM — hence, the cavity case module is called cavity.OpenFOAM.

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/incompressible/icoFoam
    touch cavityFine/cavityFine.OpenFOAM
```

Now the cavityFine case can be loaded into ParaView by selecting Open from the File menu, and having navigated the directory tree, selecting cavityFine.OpenFOAM. 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. There are numerous utilities that do specialised data manipulations, and some, simpler calculations are incorporated into a single utility \texttt{foamCalc}. As a utility, it is unique in that it is executed by

\begin{verbatim}
foamCalc <calcType> <fieldName1 ... fieldNameN>
\end{verbatim}

The calculator operation is specified in \texttt{<calcType>}; at the time of writing, the following operations are implemented: addSubtract; randomise; div; components; mag; magGrad; magSqr; interpolate. The user can obtain the list of \texttt{<calcType>} by deliberately calling one that does not exist, so that \texttt{foamCalc} throws up an error message and lists the types available, \textit{e.g.}
Selecting calcType xxxx  
unknown calcType type xxxx, constructor not in hash table
Valid calcType selections are:

8
  (randomise
  magSqr
  magGrad
  addSubtract
  div
  mag
  interpolate
  components
)

The components and mag calcTypes provide useful scalar measures of velocity. When “foamCalc components U” 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. Similarly “foamCalc mag U” writes a scalar field magU to each time directory representing the magnitude of velocity.

The user can run foamCalc with the components calcType on both cavity and cavityFine cases. For example, for the cavity case the user should do into the cavity directory and execute foamCalc as follows:

cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
foamCalc components U

The individual components can be plotted as a graph in ParaView. It is quick, convenient and has reasonably good control over labelling and formatting, so the printed output is a fairly good standard. However, to produce graphs for publication, users may prefer to write raw data and plot it with a dedicated graphing tool, such as gnuplot or Grace/xmgr. To do this, we recommend using the sample utility, described in section 6.6 and section 2.2.3.

Before commencing plotting, the user needs to load the newly generated Ux, Uy and Uz fields into ParaView. To do this, the user should click the Refresh Times at the top of the Properties panel for the cavity.OpenFOAM module which will cause the new fields to be loaded into ParaView and appear in the Volume Fields window. Ensure the new fields are selected and the changes are applied, i.e. click Apply again if necessary. Also, data is interpolated incorrectly at boundaries if the boundary regions are selected in the Mesh Parts panel. Therefore the user should deselect the patches in the Mesh Parts panel, i.e. movingWall, fixedWall and frontAndBack, and apply the changes.

Now, in order to display a graph in ParaView the user should select the module of interest, e.g. cavity.OpenFOAM and apply the Plot Over Line filter from the Filter->Data Analysis menu. This opens up a new XY Plot window below or beside the existing 3D View window. A PlotOverLine module is created in which the user can specify the end points of the line in the Properties panel. In this example, the user should 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), in the Point1 and Point2 text boxes. The Resolution can be set to 100.

On clicking Apply, a graph is generated in the XY Plot window. In the Display panel, the user should set Attribute Mode to Point Data. The Use Data Array option can be selected for the X Axis Data, taking the arc_length option so that the x-axis of the graph represents distance from the base of the cavity.

OpenFOAM-3.0.1
2.1 Lid-driven cavity flow

The user can choose the fields to be displayed in the Line Series panel of the Display window. From the list of scalar fields to be displayed, it can be seen that the magnitude and components of vector fields are available by default, e.g. displayed as $U_x$, so that it was not necessary to create $U_x$ using foamCalc. Nevertheless, the user should deselect all series except $U_x$ (or $U_{xx}$). A square colour box in the adjacent column to the selected series indicates the line colour. The user can edit this most easily by a double click of the mouse over that selection.

In order to format the graph, the user should modify the settings below the Line Series panel, namely Line Color, Line Thickness, Line Style, Marker Style and Chart Axes.

Also the user can click one of the buttons above the top left corner of the XY Plot. The third button, for example, allows the user to control View Settings in which the user can set title and legend for each axis, for example. Also, the user can set font, colour and alignment of the axes titles, and has several options for axis range and labels in linear or logarithmic scales.

Figure 2.11 is a graph produced using ParaView. The user can produce a graph however he/she wishes. For information, the graph in Figure 2.11 was produced with the options for axes of: Standard type of Notation; Specify Axis Range selected; titles in Sans Serif 12 font. The graph is displayed as a set of points rather than a line by activating the Enable Line Series button in the Display window. Note: if this button appears to be inactive by being “greyed out”, it can be made active by selecting and deselecting the sets of variables in the Line Series panel. Once the Enable Line Series button is selected, the Line Style and Marker Style can be adjusted to the user’s preference.

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/incompressible/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 2.12. The

![Block structure of the graded mesh for the cavity (block numbers encircled).](image)

user can view the blockMeshDict file in the system (or constant/polyMesh) subdirectory of
cavityGrade; 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.

```
convertToMeters 0.1;
vertices
{
(0 0 0)
(0.5 0 0)
(1 0 0)
(0 0.5 0)
(0.5 0.5 0)
(1 0.5 0)
(0 1 0)
(0.5 1 0)
(1 1 0)
(0 0 0.1)
(0.5 0 0.1)
(1 0 0.1)
(0 0.5 0.1)
(0.5 0.5 0.1)
(1 0.5 0.1)
```

OpenFOAM-3.0.1
2.1 Lid-driven cavity flow

Once familiar with the `blockMeshDict` file for this case, the user can execute `blockMesh` from the command line. 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 last and first cells, the size of the smallest cell, $\delta x_s$, is given by:

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

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

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

and

$$
\alpha = \begin{cases} 
R & \text{for } R > 1, \\
1 - r^{-n} + r^{-1} & \text{for } R < 1.
\end{cases}
$$

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. These settings can be viewed in the *cavityGrade/system/controlDict* file.

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*. Enter the *cavityGrade* directory and execute *mapFields* by:

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

Now run *icoFoam* from the case directory and monitor the run time information. 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 100, at which point the solution takes a noticeably 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* by typing:

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

OpenFOAM-3.0.1
2.1 Lid-driven cavity flow

2.1.7.1 Pre-processing

Enter 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 $1 \times 10^{-3}$ 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.

2.1.7.2 Running the code

Run icoFoam for this case from the case directory and view the run time information. When running a job in the background, the following UNIX commands can be useful:

- `nohup` enables a command to keep running after the user who issues the command has logged out;
- `nice` changes the priority of the job in the kernel’s scheduler; a niceness of -20 is the highest priority and 19 is the lowest priority.

This is useful, for example, if a user wishes to set a case running on a remote machine and does not wish to monitor it heavily, in which case they may wish to give it low priority on the machine. In that case the nohup command allows the user to log out of a remote machine he/she is running on and the job continues running, while nice can set the priority to 19. For our case of interest, we can execute the command in this manner as follows:

```bash
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavityHighRe
nohup nice -n 19 icoFoam > 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 1.395 s and initial pressure residuals are small; No Iterations 0 indicates that the solution of U has stopped:

```
Time = 1.43
Courant Number mean: 0.221921 max: 0.839902
smoothSolver: Solving for Ux, Initial residual = 8.73381e-06, Final residual = 8.73381e-06, No Iterations 0
smoothSolver: Solving for Uy, Initial residual = 9.89679e-06, Final residual = 9.89679e-06, No Iterations 0
DICPCG: Solving for p, Initial residual = 3.67506e-06, Final residual = 8.62986e-07, No Iterations 4
time step continuity errors : sum local = 6.57947e-09, global = -6.6679e-19, cumulative = -6.2539e-18
DICPCG: Solving for p, Initial residual = 2.60986e-06, Final residual = 7.92532e-07, No Iterations 3
time step continuity errors : sum local = 6.26199e-09, global = -1.02984e-18, cumulative = -7.28374e-18
ExecutionTime = 0.37 s ClockTime = 0 s
```

```
Time = 1.435
Courant Number mean: 0.221923 max: 0.839903
smoothSolver: Solving for Ux, Initial residual = 8.59381e-06, Final residual = 8.59381e-06, No Iterations 0
smoothSolver: Solving for Uy, Initial residual = 9.71405e-06, Final residual = 9.71405e-06, No Iterations 0
DICPCG: Solving for p, Initial residual = 4.0223e-06, Final residual = 9.89693e-07, No Iterations 3
time step continuity errors : sum local = 8.15199e-09, global = 5.33614e-19, cumulative = -6.75012e-18
DICPCG: Solving for p, Initial residual = 2.38807e-06, Final residual = 8.44596e-07, No Iterations 3
time step continuity errors : sum local = 7.48751e-09, global = -4.42707e-19, cumulative = -7.19283e-18
ExecutionTime = 0.37 s ClockTime = 0 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 Reynolds-averaged simulation (RAS) 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, $\varepsilon$, the turbulent dissipation rate. The additional equations and models for turbulent flow are implemented into a OpenFOAM solver called pisoFoam.

2.1.8.1 Pre-processing

Change directory to the cavity case in the $FOAM\_RUN/tutorials/incompressible/pisoFoam/-ras directory (N.B: the pisoFoam/ras directory). Generate the mesh by running blockMesh 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.

A range of wall function models is available in OpenFOAM that are applied as boundary conditions on individual patches. This enables different wall function models to be applied to different wall regions. The choice of wall function models are specified through the turbulent viscosity field, $\nu_t$ in the 0/nut file:

```plaintext
17 dimensions [0 2 -1 0 0 0];
18 internalField uniform 0;
19 boundaryField
20 {
21 movingWall
22 {
23 type nutkWallFunction;
24 value uniform 0;
25 }
26 fixedWalls
27 {
28 type nutkWallFunction;
29 value uniform 0;
30 }
31 frontAndBack
32 {
33 type empty;
34 }
35 }
36
38 // ************************************************************************* //
39
40 // ***************************************************
41```

This case uses standard wall functions, specified by the nutWallFunction type on the OpenFOAM-3.0.1
movingWall and fixedWalls patches. Other wall function models include the rough wall functions, specified though the nutRoughWallFunction keyword.

The user should now open the field files for $k$ and $\varepsilon$ ($0/k$ and $0/\varepsilon$) and examine their boundary conditions. For a wall boundary condition, $\varepsilon$ is assigned a epsilonWallFunction boundary condition and a $kqRwallFunction$ boundary condition is assigned to $k$. The latter is a generic boundary condition that can be applied to any field that are of a turbulent kinetic energy type, e.g. $k$, $q$ or Reynolds Stress $R$. The initial values for $k$ and $\varepsilon$ are set using an estimated fluctuating component of velocity $U'$ and a turbulent length scale, $l$. $k$ and $\varepsilon$ are defined in terms of these parameters as follows:

$$k = \frac{1}{2} U' \cdot U' \quad (2.8)$$
$$\varepsilon = C_\mu^{0.75} k^{1.5} \frac{l}{l} \quad (2.9)$$

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

$$k = \frac{1}{2} (U_x'^2 + U_y'^2 + U_z'^2) \quad (2.10)$$

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 5% of the box width, 0.1 m, then $k$ and $\varepsilon$ are given by:

$$U_x' = U_y' = U_z' = \frac{5}{100} \text{ m s}^{-1} \quad (2.11)$$
$$\Rightarrow k = \frac{3}{2} \left( \frac{5}{100} \right)^2 \text{ m}^2 \text{s}^{-2} = 3.75 \times 10^{-3} \text{ m}^2 \text{s}^{-2} \quad (2.12)$$
$$\varepsilon = C_\mu^{0.75} k^{1.5} \frac{l}{l} \approx 7.54 \times 10^{-3} \text{ m}^2 \text{s}^{-3} \quad (2.13)$$

These form the initial conditions for $k$ and $\varepsilon$. The initial conditions for $U$ and $p$ are (0,0,0) and 0 respectively as before.

Turbulence modelling includes a range of methods, e.g. RAS or large-eddy simulation (LES), that are provided in OpenFOAM. The choice of turbulence modelling method is selectable at run-time through the simulationType keyword in turbulenceProperties dictionary. The user can view this file in the constant directory:

```
17  simulationType RAS;
18  RAS
19    {       RASModel kOmega;
20      turbulence on;
21      printCoeffs on;
22    }
23  // *************************************************************************/
```

The options for simulationType are laminar, RAS and LES (RASModel and LESModel in OpenFOAM versions prior to v3.0.0). With RAS selected in this case, the choice of RAS
modelling is specified in a RAS subdictionary (or separate RASProperties file in OpenFOAM versions prior to v3.0.0). The turbulence model is selected by the RASModel entry from a long list of available models that are listed in Table 3.9. The \textit{kEpsilon} model should be selected which is the standard \( k - \varepsilon \) model; the user should also ensure that turbulence calculation is switched on.

The coefficients for each turbulence model are stored within the respective code with a set of default values. Setting the optional switch called \texttt{printCoeffs} to on will make the default values be printed to standard output, \textit{i.e.} the terminal, when the model is called at run time. The coefficients are printed out as a sub-dictionary whose name is that of the model name with the word \texttt{Coeffs} appended, \textit{e.g.} \texttt{kEpsilonCoeffs} in the case of the \texttt{kEpsilon} model. The coefficients of the model, \textit{e.g.} \texttt{kEpsilon}, can be modified by optionally including (copying and pasting) that sub-dictionary within the RAS sub-dictionary and adjusting values accordingly.

The user should next set the laminar kinematic viscosity in the \textit{transportProperties} dictionary. To achieve a Reynolds number of \( 10^4 \), a kinematic viscosity of \( 10^{-5} \text{ m} \) is required based on the Reynolds number definition given in Equation 2.1.

Finally the user should set the \texttt{startTime}, \texttt{stopTime}, \texttt{deltaT} and the \texttt{writeInterval} in the \texttt{controlDict}. Set \texttt{deltaT} to 0.005 s to satisfy the Courant number restriction and the \texttt{endTime} to 10 s.

### 2.1.8.2 Running the code

Execute \texttt{pisoFoam} by entering the case directory and typing “\texttt{pisoFoam}” in a terminal. 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 \( 0.2 \text{ m s}^{-1} \) 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 \texttt{deltaT} to 0.02 s and, on this occasion, set \texttt{startFrom} to \texttt{latestTime}. This instructs \texttt{pisoFoam} to read the start data from the latest time directory, \textit{i.e.} \texttt{10.0}. The \texttt{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. View the results at consecutive time steps as the solution progresses to see if the solution converges to a steady-state or perhaps reaches some periodically oscillating state. In the latter case, convergence may never occur but this does not mean the results are inaccurate.

### 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 \texttt{mapFields} utility can map fields that are inconsistent, either in terms of geometry or boundary types or both.

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

\begin{verbatim}
OpenFOAM-3.0.1
\end{verbatim}
2.1 Lid-driven cavity flow

convertToMeters 0.1;
vertices
do{
  (0 0 0)
  (0.6 0 0)
  (0 0.4 0)
  (0.6 0.4 0)
  (1 0.4 0)
  (0 1 0)
  (0.6 1 0)
  (1 1 0)
  (0 0 0.1)
  (0.6 0 0.1)
  (0 0.4 0.1)
  (0.6 0.4 0.1)
  (1 0.4 0.1)
  (0 1 0.1)
  (0.6 1 0.1)
  (1 1 0.1)
};
blocks
do{
  hex (0 1 3 2 8 9 11 10) (12 8 1) simpleGrading (1 1 1)
  hex (2 3 6 5 10 11 14 13) (12 12 1) simpleGrading (1 1 1)
};
edges
do{
};
boundary
do{
  lid
do{
    type wall;
    faces
do{
      (5 13 14 6)
      (6 14 15 7)
    );
  }
  fixedWalls
do{
    type wall;
    faces
do{
      (0 8 10 2)
      (2 10 13 5)
      (7 15 12 4)
      (4 12 11 3)
      (3 11 9 1)
      (1 9 8 0)
    );
  }
  frontAndBack
do{
    type empty;
    faces
do{
      (0 2 3 1)
      (2 5 6 3)
      (3 6 7 4)
      (8 9 11 10)
      (10 11 14 13)
      (11 12 15 14)
    );
  }
};
mergePatchPairs
do{

// ************************************************************************* //
Generate the mesh with \texttt{blockMesh}. The patches are set accordingly as in previous cavity cases. For the sake of clarity in describing the field mapping process, the upper wall patch is renamed \textit{lid}, previously the \textit{movingWall} patch of the original \textit{cavity}.

In an inconsistent mapping, there is no guarantee that all the field data can be mapped from the source case. The remaining data must come from field files in the target case itself. Therefore field data must exist in the time directory of the target case before mapping takes place. In the \texttt{cavityClipped} case the mapping is set to occur at time 0.5 s, since the \texttt{startTime} is set to 0.5 s in the \texttt{controlDict}. Therefore the user needs to copy initial field data to that directory, \textit{e.g.} from time 0:

\begin{verbatim}
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavityClipped
cp -r 0 0.5
\end{verbatim}

Before mapping the data, the user should view the geometry and fields at 0.5 s.

Now we wish to map the velocity and pressure fields from \texttt{cavity} onto the new fields of \texttt{cavityClipped}. Since the mapping is inconsistent, we need to edit the \texttt{mapFieldsDict} dictionary, located in the \texttt{system} directory. The dictionary contains 2 keyword entries: \texttt{patchMap} and \texttt{cuttingPatches}. The \texttt{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 \texttt{cavityClipped}, we wish to inherit the boundary values on the \textit{lid} patch from \texttt{movingWall} in \texttt{cavity} so we must set the \texttt{patchMap} as:

\begin{verbatim}
patchMap
(
    lid movingWall
);
\end{verbatim}

The \texttt{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 \texttt{fixedWalls} to demonstrate the interpolation process.

\begin{verbatim}
cuttingPatches
(
    fixedWalls
);
\end{verbatim}

Now the user should run \texttt{mapFields}, from within the \texttt{cavityClipped} directory:

\begin{verbatim}
mapFields ../cavity
\end{verbatim}

The user can view the mapped field as shown in Figure 2.13. 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 \texttt{fixedWalls} patch to (0,0,0). Edit the \texttt{U} field, go to the \texttt{fixedWalls} patch and change the field from \texttt{nonuniform} to \texttt{uniform} (0,0,0). The \texttt{nonuniform} field is a list of values that requires deleting in its entirety. Now run the case with \texttt{icoFoam}. 
2.1 Lid-driven cavity flow

Figure 2.13: cavity solution velocity field mapped onto cavityClipped.

Figure 2.14: cavityClipped solution for velocity field.
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 Block from the Filter menu and, in the Parameter panel, highlight the patches of interest, namely the lid and fixedWalls. On clicking Apply, these items of geometry can be displayed by selecting Wireframe in the Display panel. Figure 2.14 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 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 2.15. 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 2.15.

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.
2.2 Stress analysis of a plate with a hole

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

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

(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 2.16. 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.

The user should change into the `plateHole` case in the `$FOAM_RUN/tutorials/stress-Analysis/solidDisplacementFoam` directory and open the `blockMeshDict` file in an editor, as listed below

```plaintext
17    convertToMeters 1;
18
19    vertices
20    (
21        (0.5 0.0)
22        (1.0 0.0)
23        (2.0 0.0)
24        (2.0 0.707107 0.0)
25        (0.707107 0.707107 0.0)
26        (0.353553 0.353553 0.0)
27        (2.0 2.0)
28        (0.707107 2.0 0.0)
29        (0.2 2.0)
30        (0.1 0.0)
31        (0.5 0.5)
32        (0.5 0.0 0.5)
33        (1.0 0.5)
34        (2.0 0.5)
35        (2.0 0.707107 0.5)
36        (0.707107 0.707107 0.5)
37        (0.353553 0.353553 0.5)
38        (2.0 2.0 0.5)
39        (0.707107 2.0 0.5)
40        (0.2 2.0 0.5)
41        (0.1 0.5)
42        (0.5 0.5 0.5)
43    );
44
45    blocks
46    (
47        hex (5 4 9 10 16 15 20 21) (10 10 1) simpleGrading (1 1 1)
48        hex (0 1 4 5 11 12 15 16) (10 10 1) simpleGrading (1 1 1)
49        hex (1 2 3 4 12 13 14 15) (20 10 1) simpleGrading (1 1 1)
50        hex (4 3 7 8 20 15 18 19) (20 20 1) simpleGrading (1 1 1)
51        hex (9 4 7 8 20 15 18 19) (10 20 1) simpleGrading (1 1 1)
52    );
53
54    edges
```
Figure 2.16: Block structure of the mesh for the plate with a hole.

```
arc 0 5 (0.469846 0.17101 0)
arc 5 10 (0.17101 0.469846 0)
arc 1 4 (0.939693 0.34202 0)
ar 4 9 (0.34202 0.939693 0)
ar 11 16 (0.469846 0.17101 0.5)
ar 16 21 (0.17101 0.469846 0.5)
ar 12 15 (0.939693 0.34202 0.5)
ar 15 20 (0.34202 0.939693 0.5)
);
boundary
{
  left
  {  
    type symmetryPlane;
    faces
    
      (8 9 20 19)
      (9 10 21 20)
    );
  }
  right
  {  
    type patch;
    faces
    
      (2 3 14 13)
      (3 6 17 14)
    );
  }
  down
  {  
    type symmetryPlane;
    faces

```
2.2 Stress analysis of a plate with a hole

(0 1 12 11)
(1 2 13 12)

up
{
  type patch;
  faces
  {
    (7 8 19 18)
    (6 7 18 17)
  }
}
hole
{
  type patch;
  faces
  {
    (10 5 16 21)
    (5 0 11 16)
  }
}
frontAndBack
{
  type empty;
  faces
  {
    (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
{
};

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

Until now, we have only specified straight edges in the geometries of previous tutorials but here we need to specify curved edges. These are specified under the edges keyword entry which is a list of non-straight edges. The syntax of each list entry begins with the type of curve, including arc, simpleSpline, polyLine etc., described further in section 5.3.1. In this example, all the edges are circular and so can be specified by the arc keyword entry. The following entries are 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 2.16 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.

6 patches are defined: one for each side of the plate, one for the hole and one for the front and back planes. The left and down patches are both a symmetry plane. Since this is a geometric constraint, it is included in the definition of the mesh, rather than being purely a specification on the boundary condition of the fields. Therefore they are defined as such using a special symmetryPlane type as shown in the blockMeshDict.

The frontAndBack patch represents the plane which is ignored in a 2D case. Again this is a geometric constraint so is defined within the mesh, using the empty type as shown in the blockMeshDict. For further details of boundary types and geometric constraints, the user
should refer to section 5.2.1.

The remaining patches are of the regular patch type. 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 2.17.

![Figure 2.17: Mesh of the hole in a plate problem.](image)

### 2.2.1.1 Boundary and initial conditions

Once the mesh generation is complete, the initial field with boundary conditions must be set. For a stress analysis case without thermal stresses, only displacement $D$ needs to be set. The $0/D$ is as follows:

```plaintext
17 dimensions [0 1 0 0 0 0];
18 internalField uniform (0 0 0);
19 boundaryField
20 {
21   left
22   {
23     type symmetryPlane;
24   }
25   right
26   {
27     type tractionDisplacement;
28     traction uniform (10000 0 0);
29     pressure uniform 0;
30     value uniform (0 0 0);
31   }
32   down
33   {
34     type symmetryPlane;
35   }
36   up
37   {
38     type tractionDisplacement;
39     traction uniform (0 0 0);
40     pressure uniform 0;
41     value uniform (0 0 0);
42   }
43   hole
44   {
45     type tractionDisplacement;
46     traction uniform (0 0 0);
47   }
```

OpenFOAM-3.0.1
2.2 Stress analysis of a plate with a hole

```plaintext
pressure uniform 0;
value uniform (0 0 0);
}
frontAndBack
{
  type empty;
}

// ************************************************************************* //</
```

Firstly, it can be seen that the displacement initial conditions are set to \((0,0,0)\) m. The **left** and **down** patches **must** be both of **symmetryPlane** type since they are specified as such in the mesh description in the `constant/polyMesh/boundary` file. Similarly the **frontAndBack** patch is declared **empty**.

The other patches are traction boundary conditions, set by a specialist **traction** boundary type. The traction boundary conditions are specified by a linear combination of: (1) a boundary traction vector under keyword **traction**; (2) a pressure that produces a traction normal to the boundary surface that is defined as negative when pointing out of the surface, under keyword **pressure**. 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.

### 2.2.1.2 Mechanical properties

The physical properties for the case are set in the **mechanicalProperties** dictionary in the `constant` directory. 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Units</th>
<th>Keyword</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density</td>
<td>kg m(^{-3})</td>
<td>rho</td>
<td>7854</td>
</tr>
<tr>
<td>Young’s modulus</td>
<td>Pa</td>
<td>E</td>
<td>(2 \times 10^{11})</td>
</tr>
<tr>
<td>Poisson’s ratio</td>
<td>—</td>
<td>nu</td>
<td>0.3</td>
</tr>
</tbody>
</table>

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.

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
<table>
<thead>
<tr>
<th>Property</th>
<th>Units</th>
<th>Keyword</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specific heat capacity</td>
<td>Jkg⁻¹K⁻¹</td>
<td>C</td>
<td>434</td>
</tr>
<tr>
<td>Thermal conductivity</td>
<td>Wm⁻¹K⁻¹</td>
<td>k</td>
<td>60.5</td>
</tr>
<tr>
<td>Thermal expansion coeff.</td>
<td>K⁻¹</td>
<td>alpha</td>
<td>1.1 × 10⁻⁵</td>
</tr>
</tbody>
</table>

Table 2.2: Thermal properties for steel

important since this is a steady state case; in this situation it is best to set the time step \( \Delta T \) to 1 so it simply acts as an iteration counter for the steady-state case. The \texttt{endTime} set to 100, then acts as a limit on the number of iterations. The \texttt{writeInterval} can be set to 20.

The \texttt{controlDict} entries are as follows:

```
17 application solidDisplacementFoam;
18 startFrom startTime;
19 startTime 0;
20 stopAt endTime;
21 endTime 100;
22 \Delta T 1;
23 writeControl timeStep;
24 writeInterval 20;
25 purgeWrite 0;
26 writeFormat ascii;
27 writePrecision 6;
28 writeCompression off;
29 timeFormat general;
30 timePrecision 6;
31 graphFormat raw;
32 runTimeModifiable true;
33 // ************************************************************************* //
```

2.2.1.5 Discretisation schemes and linear-solver control

Let us turn our attention to the \texttt{fvSchemes} dictionary. Firstly, the problem we are analysing is steady-state so the user should select \texttt{SteadyState} for the time derivatives in \texttt{timeScheme}. This essentially switches off the time derivative terms. Not all solvers, especially in fluid dynamics, work for both steady-state and transient problems but \texttt{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 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 \texttt{fvSchemes} dictionary in the \texttt{system} directory and ensure the \texttt{leastSquares} method is selected for the \texttt{grad(U)} gradient discretisation scheme in the \texttt{gradSchemes} sub-dictionary.
2.2 Stress analysis of a plate with a hole

The `fvSolution` dictionary in the `system` directory controls the linear equation solvers and algorithms used in the solution. The user should first look at the `solvers` sub-dictionary and notice that the choice of `solver` for `D` is `GAMG`. 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 tight (low) relative tolerance within each iteration since a lot of terms in each equation 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, or in some cases even 0.9 (as in this case).

```plaintext
solvers
{
    
    "(<D|T>)"
    {

        solver GAMG;
        tolerance 1e-06;

        relTol 0.9;
        smoother GaussSeidel;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 20;
        agglomerator faceAreaPair;
        mergeLevels 1;
    }
}

stressAnalysis
{

    compactNormalStress yes;

    nCorrectors 1;
}
```

OpenFOAM-3.0.1
The `fvSolution` dictionary contains a sub-dictionary, `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 from the command line as specified below, so he/she can look at convergence information in the log file afterwards.

```bash
cd $FOAM_RUN/tutorials/stressAnalysis/solidDisplacementFoam/plateHole
solidDisplacementFoam > 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 0.9 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 $\sigma$ 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 post-processing individual scalar field components, $\sigma_{xx}$, $\sigma_{xy}$ etc., can be generated by running the `foamCalc` utility as before in section 2.1.5.7, this time on `sigma`:

```bash
foamCalc components sigma
```

Components named `sigmaxx`, `sigmayy` etc. are written to time directories of the case. The $\sigma_{xx}$ stresses can be viewed in `paraFoam` as shown in Figure 2.18.

We would like to compare the analytical solution of Equation 2.14 to our solution. We therefore must output a set of data of $\sigma_{xx}$ along the left edge symmetry plane of our domain. The user may generate the required graph data using the `sample` utility. The utility uses a `sampleDict` dictionary located in the `system` directory, whose entries are summarised in Table 6.8. The sample line specified in `sets` is set between $(0.0, 0.5, 0.25)$ and $(0.0, 2.0, 0.25)$, and the fields are specified in the `fields` list.
2.2 Stress analysis of a plate with a hole

![Figure 2.18: σxx stress field in the plate with hole.](image)

```
interpolationScheme cellPoint;
setFormat raw;
sets (
  leftPatch {
    type uniform;
    axis y;
    start (0 0.5 0.25);
    end (0 2 0.25);
    nPoints 100;
  }
);
fields (sigmaEq);
```

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

The user should execute sample as normal. The writeFormat is raw 2 column format. The data is written into files within time subdirectories of a postProcessing/sets directory, e.g. the data at \( t = 100 \) s is found within the file sets/100/leftPatch_sigmmax.xy. 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:

```
plot [0.5:2] [0:] 'postProcessing/sets/100/leftPatch sigmaxx.xy',
1e4*(1+(0.125/(x**2))+(0.09375/(x**4)))
```

An example plot is shown in Figure 2.19.

## 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 $\alpha$, 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 2.20.

![Figure 2.20: Geometry of the dam break.](image)

### 2.3.1 Mesh generation

The user should go to the `damBreak` case in their `$FOAM_RUN/tutorials/multiphase/interFoam/laminar` directory. Generate the mesh running `blockMesh` as described previously. The `damBreak` mesh consist of 5 blocks; the `blockMeshDict` entries are given below.

```plaintext
convertToMeters 0.146;
vertices
(
(0 0 0)
(2 0 0)
(2.16438 0 0)
(4 0 0)
(0 0.32876 0)
(2 0.32876 0)
(2.16438 0.32876 0)
(4 0.32876 0)
(0 4 0)
(2 4 0)
(2.16438 4 0)
(4 4 0)
(0 0 0.1)
(2 0 0.1)
(2.16438 0 0.1)
```

OpenFOAM-3.0.1
(4 0 0.1)
(0 0.32876 0.1)
(2 0.32876 0.1)
(2.16438 0.32876 0.1)
(4 0.32876 0.1)
(0 4 0.1)
(2 4 0.1)
(2.16438 4 0.1)
(4 4 0.1)
);
blocks
{
    hex (0 1 5 4 12 13 17 16) (23 8 1) simpleGrading (1 1 1)
    hex (2 3 7 6 14 15 19 18) (19 8 1) simpleGrading (1 1 1)
    hex (4 5 9 8 16 17 21 20) (23 42 1) simpleGrading (1 1 1)
    hex (5 6 10 9 17 18 22 21) (4 42 1) simpleGrading (1 1 1)
    hex (6 7 11 10 18 19 23 22) (19 42 1) simpleGrading (1 1 1)
};
edges
{
};
boundary
{
    leftWall
    {
        type wall;
        faces
        {
            (0 12 16 4)
            (4 16 20 8)
        }
    }
    rightWall
    {
        type wall;
        faces
        {
            (7 19 15 3)
            (11 23 19 7)
        }
    }
    lowerWall
    {
        type wall;
        faces
        {
            (0 1 13 12)
            (1 5 17 13)
            (5 6 18 17)
            (2 14 18 6)
            (2 3 15 14)
        }
    }
    atmosphere
    {
        type patch;
        faces
        {
            (8 20 21 9)
            (9 21 22 10)
            (10 22 23 11)
        }
    }
};
mergePatchPairs
{
};
// ******************************************************************************** //
2.3.2 Boundary conditions

The user can examine the boundary geometry generated by blockMesh by viewing the boundary file in the constant/polyMesh directory. The file contains a list of 5 boundary patches: leftWall, rightWall, lowerWall, atmosphere and defaultFaces. The user should notice the type of the patches. The atmosphere is a standard patch, i.e. has no special attributes, merely an entity on which boundary conditions can be specified. The defaultFaces patch is empty since the patch normal is in the direction we will not solve in this 2D case. The leftWall, rightWall and lowerWall patches are each a wall. Like the plain patch, the wall type contains no geometric or topological information about the mesh and only differs from the plain patch in that it identifies the patch as a wall, should an application need to know, e.g. to apply special wall surface modelling.

A good example is that the interFoam solver includes modelling of surface tension at the contact point between the interface and wall surface. The models are applied by specifying the alphaContactAngle boundary condition on the alpha (\(\alpha\)) field. With it, the user must specify the following: a static contact angle, \(\theta_0\); leading and trailing edge dynamic contact angles, \(\theta_A\) and \(\theta_R\) respectively; and a velocity scaling function for dynamic contact angle, \(u\theta\).

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, \(\theta_0 = 90^\circ\) and the velocity scaling function to 0. However, the simpler option which we shall choose here is to specify a zeroGradient type on alpha, rather than use the alphaContactAngle boundary condition.

The top boundary is free to the atmosphere so needs to permit both outflow and inflow according to the internal flow. We therefore use a combination of boundary conditions for pressure and velocity that does this while maintaining stability. They are:

- totalPressure which is a fixedValue condition calculated from specified total pressure \(p_0\) and local velocity \(U\);
- pressureInletOutletVelocity, which applies zeroGradient on all components, except where there is inflow, in which case a fixedValue condition is applied to the tangential component;
- inletOutlet, which is a zeroGradient condition when flow outwards, fixedValue when flow is inwards.

At all wall boundaries, the fixedFluxPressure boundary condition is applied to the pressure field, which adjusts the pressure gradient so that the boundary flux matches the velocity boundary condition.

The defaultFaces patch 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 \(\alpha_{\text{water}}\) where

\[
\alpha_{\text{water}} = \begin{cases} 
1 & \text{for the water phase} \\
0 & \text{for the air phase} 
\end{cases}
\] (2.15)
This will be done by running the \texttt{setFields} utility. It requires a \texttt{setFieldsDict} dictionary, located in the \texttt{system} directory, whose entries for this case are shown below.

```
defaultFieldValues
  (  
    volScalarFieldValue alpha.water 0 )
regions
  (  
    boxToCell
      {  
        box (0 0 -1) (0.1461 0.292 1);
        fieldValues
          (  
            volScalarFieldValue alpha.water 1  
          )
      } );
// **************************************************************************
```

The \texttt{defaultFieldValues} sets the default value of the fields, \textit{i.e.} the value the field takes unless specified otherwise in the \texttt{regions} sub-dictionary. That sub-dictionary contains a list of subdictionaries containing \texttt{fieldValues} that override the defaults in a specified region. The region is expressed in terms of a \texttt{topoSetSource} that creates a set of points, cells or faces based on some topological constraint. Here, \texttt{boxToCell} creates a bounding box within a vector minimum and maximum to define the set of cells of the water region. The phase fraction \( \alpha_{\text{water}} \) is defined as 1 in this region.

The \texttt{setFields} utility reads fields from file and, after re-calculating those fields, will write them back to file. Because the files are then overridden, it is recommended that a backup is made before \texttt{setFields} is executed. In the \texttt{damBreak} tutorial, the \texttt{alpha.water} field is initially stored as a backup \textit{only}, named \texttt{alpha.water.org}. Before running \texttt{setFields}, the user first needs to copy \texttt{alpha.water.org} to \texttt{alpha.water}, \textit{e.g.} by typing:

```
cp 0/alpha.water.org 0/alpha.water
```

The user should then execute \texttt{setFields} as any other utility is executed. Using \texttt{paraFoam}, check that the initial \texttt{alpha.water} field corresponds to the desired distribution as in Figure 2.21.

### 2.3.4 Fluid properties

Let us examine the \texttt{transportProperties} file in the \texttt{constant} directory. The dictionary contains the material properties for each fluid, separated into two dictionaries \texttt{water} and \texttt{air}. The transport model for each phase is selected by the \texttt{transportModel} keyword. The user should select \texttt{Newtonian} in which case the kinematic viscosity is single valued and specified under the keyword \texttt{nu}. The viscosity parameters for the other models, \textit{e.g.} \texttt{CrossPowerLaw}, are specified within subdictionaries with the generic name \texttt{<model>Coeffs}, \textit{i.e.} \texttt{CrossPowerLawCoeffs} in this example. The density is specified under the keyword \texttt{rho}.

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

Gravitational acceleration is uniform across the domain and is specified in a file named \texttt{g} in the \texttt{constant} directory. Unlike a normal field file, \textit{e.g.} \texttt{U} and \texttt{p}, \texttt{g} is a \texttt{uniformDimensionedVectorField} and so simply contains a set of \texttt{dimensions} and a \texttt{value} that represents \((0, 9.81, 0)\) m s\(^{-2}\) for this tutorial.: 

OpenFOAM-3.0.1
2.3 Breaking of a dam

Figure 2.21: Initial conditions for phase fraction alpha.water.

<table>
<thead>
<tr>
<th>water properties</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kinematic viscosity</td>
<td>m²s⁻¹</td>
<td>nu</td>
<td>1.0 × 10⁻⁶</td>
</tr>
<tr>
<td>Density</td>
<td>kg m⁻³</td>
<td>rho</td>
<td>1.0 × 10³</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>air properties</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kinematic viscosity</td>
<td>m²s⁻¹</td>
<td>nu</td>
<td>1.48 × 10⁻⁵</td>
</tr>
<tr>
<td>Density</td>
<td>kg m⁻³</td>
<td>rho</td>
<td>1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties of both phases</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface tension</td>
<td>N m⁻¹</td>
<td>sigma</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 2.3: Fluid properties for the damBreak tutorial

17 dimensions [0 1 -2 0 0 0 0];
18 value (0 -9.81 0);
20 // ************************************************************************* //

2.3.5 Turbulence modelling

As in the cavity example, the choice of turbulence modelling method is selectable at run-time through the simulationType keyword in turbulenceProperties dictionary. In this example, we wish to run without turbulence modelling so we set laminar:

17 simulationType laminar;
18 // ************************************************************************* //
2.3.6 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 $Co \approx 0.5$ 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 $Co$ for the phase fields, maxAlphaCo, and other fields, maxCo, to be 1.0. 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:

```plaintext
application interFoam;
startFrom startTime;
startTime 0;
endTime 1;
deltaT 0.001;
writeControl adjustableRunTime;
writeInterval 0.05;
purgeWrite 0;
writeFormat ascii;
writePrecision 6;
writeCompression uncompressed;
timeFormat general;
timePrecision 6;
runTimeModifiable yes;
adjustTimeStep yes;
maxCo 1;
maxAlphaCo 1;
maxDeltaT 1;
```

2.3.7 Discretisation schemes

The interFoam 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 therefore not restricted to those that are strongly stable or bounded, e.g. upwind differencing.

The convection schemes settings are made in the *divSchemes* sub-dictionary of the *fvSchemes* dictionary. In this example, the convection term in the momentum equation \( \nabla \cdot (\rho \mathbf{U}) \), denoted by the *div(rho*phi*,U)* keyword, uses Gauss linearUpwind *grad(U)* to produce good accuracy. The limited linear schemes require a coefficient \( \phi \) as described in section 4.4.1. Here, we have opted for best stability with \( \phi = 1.0 \). The \( \nabla \cdot (\mathbf{U}_\alpha) \) term, represented by the *div(phi,alpha)* keyword uses the vanLeer scheme. The \( \nabla \cdot (\mathbf{U}_\alpha \mathbf{U}_\alpha) \) term, represented by the *div(phirb,alpha)* keyword, can use second order linear (central) differencing as boundedness is assured by the MULES algorithm.

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

```plaintext

// ************************************************************************* //
2.3.8 Linear-solver control

In the *fvSolution*, the *PIMPLE* sub-dictionary 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 \( \alpha \) phase equation. Of particular interest are the *nAlphaSubCycles* and *cAlpha* keywords. *nAlphaSubCycles* represents the number of sub-cycles within the \( \alpha \) 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 2 sub-cycles, which means that the \( \alpha \) equation is solved in \( 2 \times \) half length time steps within each actual time step.

The *cAlpha* 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.

2.3.9 Running the code

Running of the code has been described in detail in previous tutorials. Try the following, that uses tee, a command that enables output to be written to both standard output and files:

```
cd $FOAM_RUN/tutorials/multiphase/interFoam/laminar/damBreak
interFoam | tee log
```

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

2.3.10 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 alpha.water in time, e.g. see Figure 2.22.

2.3.11 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. by

```
cd $FOAM_RUN/tutorials/multiphase/interFoam/laminar
mkdir damBreakFine
cp -r damBreak/0 damBreakFine
cp -r damBreak/system damBreakFine
cp -r damBreak/constant damBreakFine
```

Enter the new case directory 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) (46 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.
2.3 Breaking of a dam

Figure 2.22: Snapshots of phase $\alpha$.
As the mesh has now changed from the damBreak example, the user must re-initialise the phase field \texttt{alpha.water} 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 \texttt{U} and \texttt{p_rgh} fields since they are specified as \texttt{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 $\alpha = 1$ or $\alpha = 0$. Updating the field with \texttt{mapFields} may produce interpolated values $0 < \alpha < 1$ at the interface, so it is better to rerun the \texttt{setFields} utility. There is a backup copy of the initial uniform $\alpha$ field named \texttt{0/alpha.water.org} that the user should copy to \texttt{0/alpha.water} before running \texttt{setFields}:

\begin{verbatim}
cd $FOAM_RUN/tutorials/multiphase/interFoam/laminar/damBreakFine
cp -r 0/alpha.water.org 0/alpha.water
setFields
\end{verbatim}

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 \texttt{decomposePar} utility. There is a dictionary associated with \texttt{decomposePar} named \texttt{decomposeParDict} which is located in the \texttt{system} directory of the tutorial case; also, like with many utilities, a default dictionary can be found in the directory of the source code of the specific utility, i.e. in \texttt{$FOAM$UTILITIES/parallelProcessing/decomposePar} for this case.

The first entry is \texttt{numberOfSubdomains} which specifies the number of 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 \texttt{simple} and the corresponding \texttt{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 \texttt{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 \texttt{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 \texttt{numberOfSubdomains}, i.e. $n_x n_y = \texttt{numberOfSubdomains}$. 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 \texttt{delta} keyword should be set to 0.001.

For example, let us assume we wish to run on 4 processors. We would set \texttt{numberOfSubdomains} to 4 and \texttt{n} = (2, 2, 1). When running \texttt{decomposePar}, we can see from the screen messages that the decomposition is distributed fairly evenly between the processors.

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:

\begin{verbatim}
mpirun -np 4 interFoam -parallel > log &
\end{verbatim}

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 \texttt{log} file as usual.

OpenFOAM-3.0.1
2.3.12 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 execute it from the command line. The results from the fine mesh are shown in Figure 2.24. 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 -case processor1
```

then `processor1` will appear as a case module in `ParaView`. Figure 2.23 shows the mesh from processor 1 following the decomposition of the domain using the `simple` method.
Figure 2.24: Snapshots of phase $\alpha$ with refined mesh.
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.

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 $|U|$. 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.

### 3.1.2 Object-orientation and C++

Programming 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 $\text{mag}(U)$. 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

$$\frac{\partial pU}{\partial t} + \nabla \cdot \phi U - \nabla \cdot \mu \nabla U = -\nabla p$$

is represented by the code

```cpp
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

OpenFOAM-3.0.1
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 UNIXmake 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 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.

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

does 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.
3.2 Compiling applications and libraries

- 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 in 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 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/InInclude directory;
2. a local InInclude directory, i.e. newApp/InInclude;
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:

Figure 3.2: Directory structure for an application
Applications and libraries

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

Notice first that the directory names are preceded 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> \n   -l<library2> \n   ... \n   -l<libraryN>

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

OpenFOAM-3.0.1
3.2 Compiling applications and libraries

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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type of compilation</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib</td>
<td>Build a statically-linked library</td>
</tr>
<tr>
<td>libso</td>
<td>Build a dynamically-linked library</td>
</tr>
<tr>
<td>libo</td>
<td>Build a statically-linked object file library</td>
</tr>
<tr>
<td>jar</td>
<td>Build a JAVA archive</td>
</tr>
<tr>
<td>exe</td>
<td>Build an application independent of the specified project</td>
</tr>
</tbody>
</table>

Table 3.1: Optional compilation arguments to `wmake`.  

---

OpenFOAM-3.0.1
3.2.2.5 wmake environment variables

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

<table>
<thead>
<tr>
<th>Main paths</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$WM_PROJECT_INST_DIR</td>
<td>Full path to installation directory, e.g. $HOME/OpenFOAM</td>
</tr>
<tr>
<td>$WM_PROJECT</td>
<td>Name of the project being compiled: OpenFOAM</td>
</tr>
<tr>
<td>$WM_PROJECT_VERSION</td>
<td>Version of the project being compiled: 3.0.1</td>
</tr>
<tr>
<td>$WM_PROJECT_DIR</td>
<td>Full path to locate binary executables of OpenFOAM release, e.g. $HOME/OpenFOAM/OpenFOAM-3.0.1</td>
</tr>
<tr>
<td>$WM_PROJECT_USER_DIR</td>
<td>Full path to locate binary executables of the user e.g. $HOME/OpenFOAM/$USER-3.0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other paths/settings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$WM_ARCH</td>
<td>Machine architecture: Linux, SunOS</td>
</tr>
<tr>
<td>$WM_ARCH_OPTION</td>
<td>32 or 64 bit architecture</td>
</tr>
<tr>
<td>$WM_COMPILER</td>
<td>Compiler being used: Gcc43 - gcc 4.5+, ICC - Intel</td>
</tr>
<tr>
<td>$WM_COMPILER_DIR</td>
<td>Compiler installation directory</td>
</tr>
<tr>
<td>$WM_COMPILER_BIN</td>
<td>Compiler installation binaries $WM_COMPILER_BIN/bin</td>
</tr>
<tr>
<td>$WM_COMPILER_LIB</td>
<td>Compiler installation libraries $WM_COMPILER_BIN/lib</td>
</tr>
<tr>
<td>$WM_COMPILE_OPTION</td>
<td>Compilation option: Debug - debugging, Opt optimisation</td>
</tr>
<tr>
<td>$WM_DIR</td>
<td>Full path of the wmake directory</td>
</tr>
<tr>
<td>$WM_MPLIB</td>
<td>Parallel communications library: LAM, MPI, MPICH, PVM</td>
</tr>
<tr>
<td>$WM_OPTIONS</td>
<td>$WM_ARCH$WM_COMPILER...</td>
</tr>
<tr>
<td></td>
<td>...$WM_COMPILE_OPTION$WM_MPLIB</td>
</tr>
<tr>
<td></td>
<td>e.g. linuxGcc3OptMPICH</td>
</tr>
<tr>
<td>$WM_PRECISION_OPTION</td>
<td>Precision of the compiled binaries, SP, single precision or DP, double precision</td>
</tr>
</tbody>
</table>

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 *Include* 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 pisoFoam application

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

```cpp
/*---------------------------------------------------------------------------*/
// Field | OpenFOAM: The Open Source CFD Toolbox
// And | Copyright (C) 2011-2015 OpenFOAM Foundation
\\// Manipulation |
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 3 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, see <http://www.gnu.org/licenses/>.

Application
pisoFoam
Description
Transient solver for incompressible flow.

Sub-models include:
- turbulence modelling, i.e. laminar, RAS or LES
- run-time selectable MRF and finite volume options, e.g. explicit porosity

#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulentTransportModel.H"
#include "pisoControl.H"
#include "fvIOoptionList.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"

    pisoControl piso(mesh);

    #include "createFields.H"
    #include "createMRF.H"
    #include "createFvOptions.H"
    #include "initContinuityErrs.H"

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

    while (runTime.loop())
```

OpenFOAM-3.0.1
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, pisoFoam.C to read the fvCFD.H.

pisoFoam resources the incompressibleRASModels, incompressibleLESModels and incompressibleTransportModels 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 = 
-$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude 
-$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude 
-$(LIB_SRC)/transportModels 
-$(LIB_SRC)/transportModels/incompressible/singlePhaseTransportModel 
-$(LIB_SRC)/finiteVolume/lnInclude 
-$(LIB_SRC)/meshTools/lnInclude 
-$(LIB_SRC)/fvOptions/lnInclude 
-$(LIB_SRC)/sampling/lnInclude
```

```
EXE_LIBS = 
-lturbulenceModels 
-lincompressibleTurbulenceModels 
-lincompressibleTransportModels 
-lfiniteVolume 
-lmeshTools 
-lfvOptions 
-lsampling
```

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

```
pisoFoam.C
EXE = $(FOAM_APPBIN)/pisoFoam
```

Following the recommendations of section 3.2.2.3, the user can compile a separate version of pisoFoam into their local $FOAM_USER_DIR directory by the following:
3.2 Compiling applications and libraries

- copying the pisoFoam source code to a local directory, e.g. `$FOAM_RUN`;

```bash
cd $FOAM_RUN
cp -r $FOAM_SOLVERS/incompressible/pisoFoam .
cd pisoFoam
```

- editing the `Make/files` file as follows;

```bash
1  pisoFoam.C
2  EXE = $(FOAM_USER_APPBIN)/pisoFoam
```

- executing `wmake`.

```
wmake
```

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

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

SOURCE_DIR=.
SOURCE=pisoFoam.C;
g++ -DFOAM_EXCEPT -DLinux -DLinuxGccDPOpt
-DscalarMachine -DoptSolvers -DPARALLEL -DUSEMPI -Wall -O2 -DNpRepository
-ftemplate-depth-17 -I../OpenFOAM/OpenFOAM-3.0.1/src/OpenFOAM/lnInclude
-IlnInclude
-I.
......
-lmpich -L/usr/X11/lib -lm
-o ... platforms/bin/linuxGccDPOpt/pisoFoam
```

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/linuxGccDPOpt/dependencies' is up to date.
make: `... platforms/linuxGccDPOpt/bin/pisoFoam'
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 `$WM_PROJECT_DIR/etc/controlDict` file; should the user wish to change the settings they should make a copy to their `$HOME` directory, i.e. `$HOME/...OpenFOAM/3.0.1/controlDict` file. The list of possible switches is extensive and can be
viewed by running the \texttt{foamDebugSwitches} application. Most of the switches correspond to a class or range of functionality and can be switched on by their inclusion in the \texttt{controlDict} file, and by being set to 1. For example, OpenFOAM can perform the checking of dimensional units in all calculations by setting the \texttt{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 \texttt{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 \texttt{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.

<table>
<thead>
<tr>
<th>High level debugging switches - sub-dictionary \texttt{DebugSwitches}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{level}</td>
</tr>
<tr>
<td>\texttt{lduMatrix}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimisation switches - sub-dictionary \texttt{OptimisationSwitches}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{fileModificationSkew}</td>
</tr>
<tr>
<td>\texttt{fileModificationChecking}</td>
</tr>
<tr>
<td>\texttt{commsType}</td>
</tr>
<tr>
<td>\texttt{floatTransfer}</td>
</tr>
<tr>
<td>\texttt{nProcsSimpleSum}</td>
</tr>
</tbody>
</table>

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 \texttt{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 \texttt{new}, that would need to be recognised by a range of solver applications, pre- and post-processing utilities, mesh tools, \texttt{etc}. Under normal circumstances, the user would need to recompile every application with the \texttt{new} linked to it.

Instead there is a simple mechanism to link one or more shared object libraries dynamically at run-time in OpenFOAM. Simply add the optional keyword entry \texttt{libs} to the
controlDict file for a case and enter the full names of the libraries within a list (as quoted string entries). For example, if a user wished to link the libraries new1 and new2 at run-time, they would simply need to add the following to the case controlDict file:

```plaintext
libs
(
    "libnew1.so"
    "libnew2.so"
);
```

### 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 stored in a directory named after the case as described in section 4.1; the directory name with full path is here given the generic name `<caseDir>`.

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 with the `-help` option, e.g. typing

```plaintext
blockMesh -help
```

returns the usage

```
```

The arguments in square brackets, [ ], are optional flags. If the application is executed from within a case directory, it will operate on that case. Alternatively, the `-case <caseDir>` option allows the case to be specified directly so that the application can be executed from anywhere in the filing system.

Like any UNIX/Linux executable, applications can be run 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:

```plaintext
blockMesh > 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).
### 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:

```plaintext
numberOfSubdomains 4;
method simple;

simpleCoeffs
{
    n (2 2 1);
    delta 0.001;
}

hierarchicalCoeffs
{
    n (1 1 1);
    delta 0.001;
    order xyz;
}

manualCoeffs
{
    dataFile "";
}

distributed no;
roots ( );

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

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.

- **scotch** Scotch 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, through an optional `processorWeights` keyword which can be useful on machines with differing performance between processors. There is also an optional keyword entry `strategy` that controls the decomposition strategy through a complex string supplied to Scotch. For more information, see the source code file: `$FOAM_SRC/parallel/decomposesotchDecomp/scotchDecomp.C`

- **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 sub-dictionary 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.
### Compulsory entries

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>numberOfSubdomains</td>
<td>Total number of subdomains</td>
<td>$N$</td>
</tr>
<tr>
<td>method</td>
<td>Method of decomposition</td>
<td>simple/ hierarchical/ scotch/metis/manual/</td>
</tr>
</tbody>
</table>

### simpleCoeffs entries

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Number of subdomains in $x$, $y$, $z$</td>
<td>$(n_x \ n_y \ n_z)$</td>
</tr>
<tr>
<td>delta</td>
<td>Cell skew factor</td>
<td>Typically, $10^{-3}$</td>
</tr>
</tbody>
</table>

### hierarchicalCoeffs entries

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Number of subdomains in $x$, $y$, $z$</td>
<td>$(n_x \ n_y \ n_z)$</td>
</tr>
<tr>
<td>delta</td>
<td>Cell skew factor</td>
<td>Typically, $10^{-3}$</td>
</tr>
<tr>
<td>order</td>
<td>Order of decomposition</td>
<td>$xyz$ or $xzy$ or $yxz$...</td>
</tr>
</tbody>
</table>

### scotchCoeffs entries

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>processorWeights</td>
<td>List of weighting factors for allocation of cells to processors; $&lt;wt_1&gt;$ is the weighting factor for processor 1, etc.; weights are normalised so can take any range of values.</td>
<td>$(&lt;wt_1&gt;...&lt;wt_N&gt;)$</td>
</tr>
<tr>
<td>strategy</td>
<td>Decomposition strategy: optional and complex</td>
<td></td>
</tr>
</tbody>
</table>

### manualCoeffs entries

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataFile</td>
<td>Name of file containing data of allocation of cells to processors</td>
<td>&quot;&lt;fileName&gt;&quot;</td>
</tr>
</tbody>
</table>

### Distributed data entries (optional) — see section 3.4.3

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>distributed</td>
<td>Is the data distributed across several disks?</td>
<td>yes/no</td>
</tr>
<tr>
<td>roots</td>
<td>Root paths to case directories; $&lt;rt_1&gt;$ is the root path for node 1, etc.</td>
<td>$(&lt;rt_1&gt;...&lt;rt_N&gt;)$</td>
</tr>
</tbody>
</table>

Table 3.4: Keywords in `decompositionDict` dictionary.

The `decomposePar` utility is executed in the normal manner by typing

```
decomposePar
```

On completion, a set of subdirectories will have been created, one for each processor, in the case directory. The directories are named `processorN` 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 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:

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

An application is run in parallel using mpirun.

```
mpirun --hostfile <machines> -np <nProcs> <foamExec> <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/-incompressible/icoFoam directory, then the following command should be executed:

```
mpirun --hostfile machines -np 4 icoFoam -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>
```

OpenFOAM-3.0.1
where \(<nRoots>\) is the number of roots.

Each of the \(\text{processorN}\) directories should be placed in the case directory at each of the root paths specified in the \(\text{decomposeParDict}\) dictionary. The \text{system} directory and \text{files} within the \text{constant} directory must also be present in each case directory. Note: the files in the \text{constant} directory are needed, but the \text{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 \(\text{processorN}\) directory into a single set of time directories. The \text{reconstructPar} utility performs such a reconstruction by executing the command:

```
reconstructPar
```

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 \text{paraFoam} post-processor, described in section 6.1. 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 \$\text{FOAM\_SOLVERS}\) directory, reached quickly by typing \text{sol} at the command line. This directory is further subdivided into several directories by category of continuum mechanics, \emph{e.g.} incompressible flow, combustion and solid body stress analysis. Each solver is given a name that is reasonably descriptive, \emph{e.g.} \text{icoFoam} solves incompressible, laminar flow. The current list of solvers distributed with OpenFOAM is given in Table 3.5.
### ‘Basic’ CFD codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>laplacianFoam</td>
<td>Solves a simple Laplace equation, e.g. for thermal diffusion in a solid</td>
</tr>
<tr>
<td>potentialFoam</td>
<td>Potential flow solver which solves for the velocity potential from which the flux-field is obtained and velocity field by reconstructing the flux</td>
</tr>
<tr>
<td>scalarTransportFoam</td>
<td>Solves a transport equation for a passive scalar</td>
</tr>
</tbody>
</table>

### Incompressible flow

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjointShapeOptimizationFoam</td>
<td>Steady-state solver for incompressible, turbulent flow of non-Newtonian fluids with optimisation of duct shape by applying ”blockage” in regions causing pressure loss as estimated using an adjoint formulation</td>
</tr>
<tr>
<td>boundaryFoam</td>
<td>Steady-state solver for incompressible, 1D turbulent flow, typically to generate boundary layer conditions at an inlet, for use in a simulation</td>
</tr>
<tr>
<td>icoFoam</td>
<td>Transient solver for incompressible, laminar flow of Newtonian fluids</td>
</tr>
<tr>
<td>nonNewtonianIcoFoam</td>
<td>Transient solver for incompressible, laminar flow of non-Newtonian fluids</td>
</tr>
<tr>
<td>pimpleFoam</td>
<td>Large time-step transient solver for incompressible, flow using the PIMPLE (merged PISO-SIMPLE) algorithm</td>
</tr>
<tr>
<td>pisoFoam</td>
<td>Transient solver for incompressible flow</td>
</tr>
<tr>
<td>shallowWaterFoam</td>
<td>Transient solver for inviscid shallow-water equations with rotation</td>
</tr>
<tr>
<td>simpleFoam</td>
<td>Steady-state solver for incompressible, turbulent flow</td>
</tr>
</tbody>
</table>

### Compressible flow

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rhoCentralDyMFoam</td>
<td>Density-based compressible flow solver based on central-upwind schemes of Kurganov and Tadmor with moving mesh capability and turbulence modelling</td>
</tr>
<tr>
<td>rhoCentralFoam</td>
<td>Density-based compressible flow solver based on central-upwind schemes of Kurganov and Tadmor</td>
</tr>
<tr>
<td>rhoPimpleFoam</td>
<td>Transient solver for laminar or turbulent flow of compressible fluids for HVAC and similar applications</td>
</tr>
<tr>
<td>rhoPorousSimpleFoam</td>
<td>Steady-state solver for turbulent flow of compressible fluids with RANS turbulence modelling, implicit or explicit porosity treatment and run-time selectable finite volume sources</td>
</tr>
<tr>
<td>rhoSimplecFoam</td>
<td>Steady-state SIMPLEC solver for laminar or turbulent RANS flow of compressible fluids</td>
</tr>
<tr>
<td>rhoSimpleFoam</td>
<td>Steady-state SIMPLE solver for laminar or turbulent RANS flow of compressible fluids</td>
</tr>
<tr>
<td>sonicDyMFoam</td>
<td>Transient solver for trans-sonic/supersonic, laminar or turbulent flow of a compressible gas with mesh motion</td>
</tr>
</tbody>
</table>

*Continued on next page*
3.5 Standard solvers

Continued from previous page

**sonicFoam**
Transient solver for trans-sonic/supersonic, laminar or turbulent flow of a compressible gas

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

### Multiphase flow

**cavitatingDyMFoam**
Transient cavitation code based on the homogeneous equilibrium model from which the compressibility of the liquid/vapour “mixture” is obtained, with optional mesh motion and mesh topology changes including adaptive re-meshing

**cavitatingFoam**
Transient cavitation code based on the homogeneous equilibrium model from which the compressibility of the liquid/vapour “mixture” is obtained

**compressibleInterDyMFoam**
Solver for 2 compressible, non-isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach, with optional mesh motion and mesh topology changes including adaptive re-meshing

**compressibleInterFoam**
Solver for 2 compressible, non-isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach

**compressibleMultiphaseInterFoam**
Solver for $n$ compressible, non-isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach

**driftFluxFoam**
Solver for 2 incompressible fluids using the mixture approach with the drift-flux approximation for relative motion of the phases

**interFoam**
Solver for 2 incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach

**interMixingFoam**
Solver for 3 incompressible fluids, two of which are miscible, using a VOF method to capture the interface

**interPhaseChangeFoam**
Solver for 2 incompressible, isothermal immiscible fluids with phase-change (e.g. cavitation). Uses a VOF (volume of fluid) phase-fraction based interface capturing approach

**interPhaseChangeDyMFoam**
Solver for 2 incompressible, isothermal immiscible fluids with phase-change (e.g. cavitation). Uses a VOF (volume of fluid) phase-fraction based interface capturing approach, with optional mesh motion and mesh topology changes including adaptive re-meshing

**multiphaseEulerFoam**
Solver for a system of many compressible fluid phases including heat-transfer

**multiphaseInterFoam**
Solver for $n$ incompressible fluids which captures the interfaces and includes surface-tension and contact-angle effects for each phase

**potentialFreeSurfaceFoam**
Incompressible Navier-Stokes solver with inclusion of a wave height field to enable single-phase free-surface approximations

Continued on next page
**Applications and libraries**

*Continued from previous page*

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reactingEulerFoam</td>
<td>(Description not found)</td>
</tr>
<tr>
<td>twoLiquidMixingFoam</td>
<td>Solver for mixing 2 incompressible fluids</td>
</tr>
<tr>
<td>twoPhaseEulerFoam</td>
<td>Solver for a system of 2 compressible fluid phases with one phase dispersed, e.g. gas bubbles in a liquid including heat-transfer</td>
</tr>
</tbody>
</table>

**Direct numerical simulation (DNS)**

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dnsFoam</td>
<td>Direct numerical simulation solver for boxes of isotropic turbulence</td>
</tr>
</tbody>
</table>

**Combustion**

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chemFoam</td>
<td>Solver for chemistry problems - designed for use on single cell cases to provide comparison against other chemistry solvers - single cell mesh created on-the-fly - fields created on the fly from the initial conditions</td>
</tr>
<tr>
<td>coldEngineFoam</td>
<td>Solver for cold-flow in internal combustion engines</td>
</tr>
<tr>
<td>engineFoam</td>
<td>Solver for internal combustion engines</td>
</tr>
<tr>
<td>fireFoam</td>
<td>Transient Solver for Fires and turbulent diffusion flames</td>
</tr>
<tr>
<td>PDRFoam</td>
<td>Solver for compressible premixed/partially-premixed combustion with turbulence modelling</td>
</tr>
<tr>
<td>reactingFoam</td>
<td>Solver for combustion with chemical reactions</td>
</tr>
<tr>
<td>rhoReactingBuoyantFoam</td>
<td>Solver for combustion with chemical reactions using density based thermodynamics package, using enhanced buoyancy treatment</td>
</tr>
<tr>
<td>rhoReactingFoam</td>
<td>Solver for combustion with chemical reactions using density based thermodynamics package</td>
</tr>
<tr>
<td>XiFoam</td>
<td>Solver for compressible premixed/partially-premixed combustion with turbulence modelling</td>
</tr>
</tbody>
</table>

**Heat transfer and buoyancy-driven flows**

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buoyantBoussinesqPimpleFoam</td>
<td>Transient solver for buoyant, turbulent flow of incompressible fluids</td>
</tr>
<tr>
<td>buoyantBoussinesqSimpleFoam</td>
<td>Steady-state solver for buoyant, turbulent flow of incompressible fluids</td>
</tr>
<tr>
<td>buoyantPimpleFoam</td>
<td>Transient solver for buoyant, turbulent flow of compressible fluids for ventilation and heat-transfer</td>
</tr>
<tr>
<td>buoyantSimpleFoam</td>
<td>Steady-state solver for buoyant, turbulent flow of compressible fluids, including radiation, for ventilation and heat-transfer</td>
</tr>
<tr>
<td>chtMultiRegionFoam</td>
<td>Combination of heatConductionFoam and buoyantFoam for conjugate heat transfer between a solid region and fluid region</td>
</tr>
<tr>
<td>chtMultiRegionSimpleFoam</td>
<td>Steady-state version of chtMultiRegionFoam</td>
</tr>
<tr>
<td>thermoFoam</td>
<td>Evolves the thermodynamics on a frozen flow field</td>
</tr>
</tbody>
</table>

*Continued on next page*
3.5 Standard solvers

**Particle-tracking flows**

<table>
<thead>
<tr>
<th>Solver</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coalChemistryFoam</td>
<td>Transient solver for: - compressible, - turbulent flow, with - coal and limestone parcel injections, - energy source, and - combustion</td>
</tr>
<tr>
<td>DPMFoam</td>
<td>Transient solver for the coupled transport of a single kinematic particle cloud including the effect of the volume fraction of particles on the continuous phase</td>
</tr>
<tr>
<td>icoUncoupledKinematicParcelFoam</td>
<td>Transient solver for the passive transport of a single kinematic particle cloud</td>
</tr>
<tr>
<td>reactingParcelFilmFoam</td>
<td>Transient PIMPLE solver for compressible, laminar or turbulent flow with reacting Lagrangian parcels, and surface film modelling</td>
</tr>
<tr>
<td>reactingParcelFoam</td>
<td>Transient PIMPLE solver for compressible, laminar or turbulent flow with reacting multiphase Lagrangian parcels, including run-time selectable finite volume options, e.g. sources, constraints</td>
</tr>
<tr>
<td>sprayFoam</td>
<td>Transient PIMPLE solver for compressible, laminar or turbulent flow with spray parcels</td>
</tr>
<tr>
<td>uncoupledKinematicParcelFoam</td>
<td>Transient solver for the passive transport of a single kinematic particle cloud</td>
</tr>
</tbody>
</table>

**Molecular dynamics methods**

<table>
<thead>
<tr>
<th>Solver</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdEquilibrationFoam</td>
<td>Equilibrates and/or preconditions molecular dynamics systems</td>
</tr>
<tr>
<td>mdFoam</td>
<td>Molecular dynamics solver for fluid dynamics</td>
</tr>
</tbody>
</table>

**Direct simulation Monte Carlo methods**

<table>
<thead>
<tr>
<th>Solver</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsmcFoam</td>
<td>Direct simulation Monte Carlo (DSMC) solver for 3D, transient, multi-species flows</td>
</tr>
</tbody>
</table>

**Electromagnetics**

<table>
<thead>
<tr>
<th>Solver</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>electrostaticFoam</td>
<td>Solver for electrostatics</td>
</tr>
<tr>
<td>magneticFoam</td>
<td>Solver for the magnetic field generated by permanent magnets</td>
</tr>
<tr>
<td>mhdFoam</td>
<td>Solver for magnetohydrodynamics (MHD); incompressible, laminar flow of a conducting fluid under the influence of a magnetic field</td>
</tr>
</tbody>
</table>

**Stress analysis of solids**

<table>
<thead>
<tr>
<th>Solver</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>solidDisplacementFoam</td>
<td>Transient segregated finite-volume solver of linear-elastic, small-strain deformation of a solid body, with optional thermal diffusion and thermal stresses</td>
</tr>
<tr>
<td>solidEquilibriumDisplacementFoam</td>
<td>Steady-state segregated finite-volume solver of linear-elastic, small-strain deformation of a solid body, with optional thermal diffusion and thermal stresses</td>
</tr>
</tbody>
</table>

*Continued on next page*
Applications and libraries

Continued from previous page

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 $FOAM_UTILITIES$ directory, reached quickly by typing util at the command line. Again the names are reasonably descriptive, e.g. 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.

<table>
<thead>
<tr>
<th>Pre-processing</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>applyBoundaryLayer</td>
<td>Apply a simplified boundary-layer model to the velocity and turbulence fields based on the 1/7th power-law</td>
</tr>
<tr>
<td>applyWallFunction-BoundaryConditions</td>
<td>Updates OpenFOAM RAS cases to use the new (v1.6) wall function framework</td>
</tr>
<tr>
<td>boxTurb</td>
<td>Makes a box of turbulence which conforms to a given energy spectrum and is divergence free</td>
</tr>
<tr>
<td>changeDictionary</td>
<td>Utility to change dictionary entries, e.g. can be used to change the patch type in the field and polyMesh/boundary files</td>
</tr>
<tr>
<td>createExternalCoupled-PatchGeometry</td>
<td>Application to generate the patch geometry (points and faces) for use with the externalCoupled boundary condition</td>
</tr>
<tr>
<td>dsmcInitialise</td>
<td>Initialise a case for dsmcFoam by reading the initialisation dictionary system/dsmcInitialise</td>
</tr>
<tr>
<td>engineSwirl</td>
<td>Generates a swirling flow for engine calulations</td>
</tr>
<tr>
<td>faceAgglomerate</td>
<td>Agglomerate boundary faces for use with the view factor radiation model. Writes a map from the fine to the coarse grid.</td>
</tr>
<tr>
<td>foamUpgradeCyclics</td>
<td>Tool to upgrade mesh and fields for split cyclics</td>
</tr>
<tr>
<td>foamUpgradeFvSolution</td>
<td>Simple tool to upgrade the syntax of system/fvSolution::solvers</td>
</tr>
<tr>
<td>mapFields</td>
<td>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</td>
</tr>
<tr>
<td>mapFieldsPar</td>
<td>Maps volume fields from one mesh to another, reading and interpolating all fields present in the time directory of both cases</td>
</tr>
<tr>
<td>mdInitialise</td>
<td>Initialises fields for a molecular dynamics (MD) simulation</td>
</tr>
<tr>
<td>setFields</td>
<td>Set values on a selected set of cells/patchfaces through a dictionary</td>
</tr>
<tr>
<td>viewFactorsGen</td>
<td>Calculates view factors based on agglomerated faces (faceAgglomerat) for view factor radiation model.</td>
</tr>
<tr>
<td>wallFunctionTable</td>
<td>Generates a table suitable for use by tabulated wall functions</td>
</tr>
</tbody>
</table>

Continued on next page
### Mesh generation

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blockMesh</td>
<td>A multi-block mesh generator</td>
</tr>
<tr>
<td>extrudeMesh</td>
<td>Extrude mesh from existing patch (by default outwards facing normals; optional flips faces) or from patch read from file.</td>
</tr>
<tr>
<td>extrude2DMesh</td>
<td>Takes 2D mesh (all faces 2 points only, no front and back faces) and creates a 3D mesh by extruding with specified thickness</td>
</tr>
<tr>
<td>extrudeToRegionMesh</td>
<td>Extrude faceZones into separate mesh (as a different region), e.g. for creating liquid film regions</td>
</tr>
<tr>
<td>foamyHexMesh</td>
<td>Conformal Voronoi automatic mesh generator</td>
</tr>
<tr>
<td>foamyHexMeshBackgroundMesh</td>
<td>Writes out background mesh as constructed by foamyHexMesh and constructs distanceSurface</td>
</tr>
<tr>
<td>foamyHexMeshSurfaceSimplify</td>
<td>Simplifies surfaces by resampling</td>
</tr>
<tr>
<td>foamyQuadMesh</td>
<td>Conformal-Voronoi 2D extruding automatic mesher</td>
</tr>
<tr>
<td>snappyHexMesh</td>
<td>Automatic split hex mesher. Refines and snaps to surface</td>
</tr>
</tbody>
</table>

### Mesh conversion

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ansysToFoam</td>
<td>Converts an ANSYS input mesh file, exported from I-DEAS, to OpenFOAM format</td>
</tr>
<tr>
<td>cfx4ToFoam</td>
<td>Converts a CFX 4 mesh to OpenFOAM format</td>
</tr>
<tr>
<td>datToFoam</td>
<td>Reads in a datToFoam mesh file and outputs a points file. Used in conjunction with blockMesh</td>
</tr>
<tr>
<td>fluent3DMeshToFoam</td>
<td>Converts a Fluent mesh to OpenFOAM format</td>
</tr>
<tr>
<td>fluentMeshToFoam</td>
<td>Converts a Fluent mesh to OpenFOAM format including multiple region and region boundary handling</td>
</tr>
<tr>
<td>foamMeshToFluent</td>
<td>Writes out the OpenFOAM mesh in Fluent mesh format</td>
</tr>
<tr>
<td>foamToStarMesh</td>
<td>Reads an OpenFOAM mesh and writes a PROSTAR (v4) bnd/cel/vrt format</td>
</tr>
<tr>
<td>foamToSurface</td>
<td>Reads an OpenFOAM mesh and writes the boundaries in a surface format</td>
</tr>
<tr>
<td>gambitToFoam</td>
<td>Converts a GAMBIT mesh to OpenFOAM format</td>
</tr>
<tr>
<td>gmshToFoam</td>
<td>Reads .msh file as written by Gmsh</td>
</tr>
<tr>
<td>ideasUnvToFoam</td>
<td>I-Deas unv format mesh conversion</td>
</tr>
<tr>
<td>kivaToFoam</td>
<td>Converts a KIVA grid to OpenFOAM format</td>
</tr>
<tr>
<td>mshToFoam</td>
<td>Converts .msh file generated by the Adventure system</td>
</tr>
<tr>
<td>netgenNeutralToFoam</td>
<td>Converts neutral file format as written by Netgen v4.4</td>
</tr>
<tr>
<td>plot3dToFoam</td>
<td>Plot3d mesh (ascii/formatted format) converter</td>
</tr>
<tr>
<td>sammToFoam</td>
<td>Converts a STAR-CD (v3) SAMM mesh to OpenFOAM format</td>
</tr>
<tr>
<td>star3ToFoam</td>
<td>Converts a STAR-CD (v3) PROSTAR mesh into OpenFOAM format</td>
</tr>
<tr>
<td>star4ToFoam</td>
<td>Converts a STAR-CD (v4) PROSTAR mesh into OpenFOAM format</td>
</tr>
<tr>
<td>tetgenToFoam</td>
<td>Converts .ele and .node and .face files, written by tetgen</td>
</tr>
</tbody>
</table>

*Continued on next page*
**Applications and libraries**

Continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>vtkUnstructuredToFoam</strong></td>
<td>Converts ascii .vtk (legacy format) file generated by vtk/paraview</td>
</tr>
<tr>
<td><strong>writeMeshObj</strong></td>
<td>For mesh debugging: writes mesh as three separate OBJ files which can be viewed with e.g. javaview</td>
</tr>
</tbody>
</table>

**Mesh manipulation**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attachMesh</strong></td>
<td>Attach topologically detached mesh using prescribed mesh modifiers</td>
</tr>
<tr>
<td><strong>autoPatch</strong></td>
<td>Divides external faces into patches based on (user supplied) feature angle</td>
</tr>
<tr>
<td><strong>checkMesh</strong></td>
<td>Checks validity of a mesh</td>
</tr>
<tr>
<td><strong>createBaffles</strong></td>
<td>Makes internal faces into boundary faces. Does not duplicate points, unlike <strong>mergeOrSplitBaffles</strong></td>
</tr>
<tr>
<td><strong>createPatch</strong></td>
<td>Utility to create patches out of selected boundary faces. Faces come either from existing patches or from a <strong>faceSet</strong></td>
</tr>
<tr>
<td><strong>deformedGeom</strong></td>
<td>Deforms a polyMesh using a displacement field ( U ) and a scaling factor supplied as an argument</td>
</tr>
<tr>
<td><strong>flattenMesh</strong></td>
<td>Flattens the front and back planes of a 2D cartesian mesh</td>
</tr>
<tr>
<td><strong>insideCells</strong></td>
<td>Picks up cells with cell centre 'inside' of surface. Requires surface to be closed and singly connected</td>
</tr>
<tr>
<td><strong>mergeMeshes</strong></td>
<td>Merges two meshes</td>
</tr>
<tr>
<td><strong>mergeOrSplitBaffles</strong></td>
<td>Detects faces that share points (baffles). Either merge them or duplicate the points</td>
</tr>
<tr>
<td><strong>mirrorMesh</strong></td>
<td>Mirrors a mesh around a given plane</td>
</tr>
<tr>
<td><strong>moveDynamicMesh</strong></td>
<td>Mesh motion and topological mesh changes utility</td>
</tr>
<tr>
<td><strong>moveEngineMesh</strong></td>
<td>Solver for moving meshes for engine calculations</td>
</tr>
<tr>
<td><strong>moveMesh</strong></td>
<td>Solver for moving meshes</td>
</tr>
<tr>
<td><strong>objToVTK</strong></td>
<td>Read obj line (not surface!) file and convert into vtk</td>
</tr>
<tr>
<td><strong>orientFaceZone</strong></td>
<td>Corrects orientation of <strong>faceZone</strong></td>
</tr>
<tr>
<td><strong>polyDualMesh</strong></td>
<td>Calculates the dual of a polyMesh. Adheres to all the feature and patch edges</td>
</tr>
<tr>
<td><strong>refineMesh</strong></td>
<td>Utility to refine cells in multiple directions</td>
</tr>
<tr>
<td><strong>renumberMesh</strong></td>
<td>Renumbers the cell list in order to reduce the bandwidth, reading and renumbering all fields from all the time directories</td>
</tr>
<tr>
<td><strong>rotateMesh</strong></td>
<td>Rotates the mesh and fields from the direction ( n_1 ) to direction ( n_2 )</td>
</tr>
<tr>
<td><strong>setSet</strong></td>
<td>Manipulate a cell/face/point/ set or zone interactively</td>
</tr>
<tr>
<td><strong>setsToZones</strong></td>
<td>Add <strong>pointZones/</strong> <strong>faceZones/</strong> <strong>cellZones</strong> to the mesh from similar named <strong>pointSets/</strong> <strong>faceSets/</strong> <strong>cellSets</strong></td>
</tr>
<tr>
<td><strong>singleCellMesh</strong></td>
<td>Reads all fields and maps them to a mesh with all internal faces removed (singleCellFvMesh) which gets written to region ’singleCell’</td>
</tr>
<tr>
<td><strong>splitMesh</strong></td>
<td>Splits mesh by making internal faces external. Uses <strong>attachDetach</strong></td>
</tr>
<tr>
<td><strong>splitMeshRegions</strong></td>
<td>Splits mesh into multiple regions</td>
</tr>
<tr>
<td><strong>stitchMesh</strong></td>
<td>'Stitches' a mesh</td>
</tr>
</tbody>
</table>

Continued on next page
3.6 Standard utilities

Continued from previous page

subsetMesh
- Selects a section of mesh based on a cellSet

topoSet
- Operates on cellSets/facesets/pointSets through a dictionary

transformPoints
- Transforms the mesh points in the polyMesh directory according to the translate, rotate and scale 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

Other mesh tools

autoRefineMesh
- Utility to refine cells near to a surface

collapseEdges
- Collapses short edges and combines edges that are in line

combinePatchFaces
- Checks for multiple patch faces on same cell and combines them. Multiple patch faces can result from e.g. removal of refined neighbouring cells, leaving 4 exposed faces with same owner

modifyMesh
- Manipulates mesh elements

PDRMesh
- Mesh and field preparation utility for PDR type simulations

refineHexMesh
- Refines a hex mesh by 2x2x2 cell splitting

refinementLevel
- Tries to figure out what the refinement level is on refined cartesian meshes. Run before snapping

refineWallLayer
- Utility to refine cells next to patches

removeFaces
- Utility to remove faces (combines cells on both sides)

selectCells
- Select cells in relation to surface

splitCells
- Utility to split cells with flat faces

Post-processing graphics

ensightFoamReader
- EnSight library module to read OpenFOAM data directly without translation

Post-processing data converters

foamDataToFluent
- Translates OpenFOAM data to Fluent format

foamToEnsight
- Translates OpenFOAM data to EnSight format

foamToEnsightParts
- Translates OpenFOAM data to Ensight format. An Ensight part is created for each cellZone and patch

foamToGMV
- Translates foam output to GMV readable files

foamToTecplot360
- Tecplot binary file format writer

foamToTetDualMesh
- Converts polyMesh results to tetDualMesh

foamToVTX
- Legacy VTK file format writer

smapToFoam
- Translates a STAR-CD SMAP data file into OpenFOAM field format

Post-processing velocity fields

Co
- Calculates and writes the Courant number obtained from field phi as a volScalarField.

enstrophy
- Calculates and writes the enstrophy of the velocity field U

flowType
- Calculates and writes the flowType of velocity field U

Continued on next page
Continued from previous page

<table>
<thead>
<tr>
<th>Applications and libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lambda2</strong></td>
</tr>
<tr>
<td><strong>Mach</strong></td>
</tr>
<tr>
<td><strong>Pe</strong></td>
</tr>
<tr>
<td><strong>Q</strong></td>
</tr>
<tr>
<td><strong>streamFunction</strong></td>
</tr>
<tr>
<td><strong>uPrime</strong></td>
</tr>
<tr>
<td><strong>vorticity</strong></td>
</tr>
</tbody>
</table>

**Post-processing stress fields**

| stressComponents          | Calculates and writes the scalar fields of the six components of the stress tensor $\sigma$ for each time |

**Post-processing scalar fields**

| pPrime2                   | Calculates and writes the scalar field of `$pPrime2 ([p - \overline{p}]^2)$` at each time |

**Post-processing at walls**

| 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 reports the turbulent wall shear stress for all patches, for the specified times |
| yPlus                     | Calculates and reports $yPlus$ for the near-wall cells of all wall patches, for the specified times for laminar, LES and RAS |

**Post-processing turbulence**

| createTurbulenceFields    | Creates a full set of turbulence fields |
| R                         | Calculates and writes the Reynolds stress R for the current time step |

**Post-processing patch data**

| patchAverage              | Calculates the average of the specified field over the specified patch |
| patchIntegrate            | Calculates the integral of the specified field over the specified patch |

**Post-processing Lagrangian simulation**

Continued on next page
### 3.6 Standard utilities

*Continued from previous page*

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>particleTracks</strong></td>
<td>Generates a VTK file of particle tracks for cases that were computed using a tracked-parcel-type cloud</td>
</tr>
<tr>
<td><strong>steadyParticleTracks</strong></td>
<td>Generates a VTK file of particle tracks for cases that were computed using a steady-state cloud NOTE: case must be re-constructed (if running in parallel) before use</td>
</tr>
</tbody>
</table>

#### Sampling post-processing

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>probeLocations</strong></td>
<td>Probe locations</td>
</tr>
<tr>
<td><strong>sample</strong></td>
<td>Sample field data with a choice of interpolation schemes, sampling options and write formats</td>
</tr>
</tbody>
</table>

#### Miscellaneous post-processing

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dsmcFieldsCalc</strong></td>
<td>Calculate intensive fields (U and T) from averaged extensive fields from a DSMC calculation</td>
</tr>
<tr>
<td><strong>engineCompRatio</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>execFlowFunctionObjects</strong></td>
<td>Execute the set of functionObjects specified in the selected dictionary (which defaults to <code>system/controlDict</code>) for the selected set of times. Alternative dictionaries should be placed in the <code>system/</code> directory</td>
</tr>
<tr>
<td><strong>foamListTimes</strong></td>
<td>List times using timeSelector</td>
</tr>
<tr>
<td><strong>pdfPlot</strong></td>
<td>Generates a graph of a probability distribution function</td>
</tr>
<tr>
<td><strong>postChannel</strong></td>
<td>Post-processes data from channel flow calculations</td>
</tr>
<tr>
<td><strong>ptot</strong></td>
<td>For each time: calculate the total pressure</td>
</tr>
<tr>
<td><strong>temporalInterpolate</strong></td>
<td>Interpolate fields between time-steps e.g. for animation</td>
</tr>
<tr>
<td><strong>wdot</strong></td>
<td>Calculates and writes wdot for each time</td>
</tr>
<tr>
<td><strong>writeCellCentres</strong></td>
<td>Write the three components of the cell centres as <code>volScalarFields</code> so they can be used in postprocessing in thresholding</td>
</tr>
</tbody>
</table>

#### Surface mesh (e.g. STL) tools

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>surfaceAdd</strong></td>
<td>Add two surfaces. Does not check for overlapping/intersecting triangles</td>
</tr>
<tr>
<td><strong>surfaceAutoPatch</strong></td>
<td>Patches surface according to feature angle. Like autoPatch</td>
</tr>
<tr>
<td><strong>surfaceBooleanFeatures</strong></td>
<td>Generates the extendedFeatureEdgeMesh for the interface between a boolean operation on two surfaces</td>
</tr>
<tr>
<td><strong>surfaceCheck</strong></td>
<td>Checking geometric and topological quality of a surface</td>
</tr>
<tr>
<td><strong>surfaceClean</strong></td>
<td>- removes baffles - collapses small edges, removing triangles.</td>
</tr>
<tr>
<td><strong>surfaceCoarsen</strong></td>
<td>Surface coarsening using ‘bunnylod’.</td>
</tr>
<tr>
<td><strong>surfaceConvert</strong></td>
<td>Converts from one surface mesh format to another</td>
</tr>
<tr>
<td><strong>surfaceFeatureConvert</strong></td>
<td>Convert between edgeMesh formats</td>
</tr>
<tr>
<td><strong>surfaceFeatureExtract</strong></td>
<td>Extracts and writes surface features to file</td>
</tr>
<tr>
<td><strong>surfaceFind</strong></td>
<td>Finds nearest face and vertex</td>
</tr>
<tr>
<td><strong>surfaceHookUp</strong></td>
<td>Find close open edges and stitches the surface along them</td>
</tr>
</tbody>
</table>

*Continued on next page*
surfaceInertia
Calculates the inertia tensor, principal axes and moments of a command line specified triSurface. Inertia can either be of the solid body or of a thin shell.

surfaceLambdaMuSmooth
Smoothes a surface using lambda/mu smoothing. To get laplacian smoothing (previous surfaceSmooth behavior), set lambda to the relaxation factor and mu to zero.

surfaceMeshConvert
Converts between surface formats with optional scaling or transformations (rotate/translate) on a coordinateSystem.

surfaceMeshConvertTesting
Converts from one surface mesh format to another, but primarily used for testing functionality.

surfaceMeshExport
Exports from surfMesh to various third-party surface formats with optional scaling or transformations (rotate/translate) on a coordinateSystem.

surfaceMeshImport
Imports from various third-party surface formats into surfMesh with optional scaling or transformations (rotate/translate) on a coordinateSystem.

surfaceMeshInfo
Miscellaneous information about surface meshes.

surfaceMeshTriangulate
Extracts triSurface from a polyMesh. Depending on output surface format triangulates faces. Region numbers on triangles are the patch numbers of the polyMesh. Optionally only triangulates named patches.

surfaceOrient
Sets normal consistent with respect to a user provided ‘outside’ point. If the -inside is used the point is considered inside.

surfacePointMerge
Merges points on surface if they are within absolute distance. Since absolute distance use with care!

surfaceRedistributePar
(Re)distribution of triSurface. Either takes an undecomposed surface or an already decomposed surface and redistributes it so that each processor has all triangles that overlap its mesh.

surfaceRefineRedGreen
Refine by splitting all three edges of triangle (‘red’ refinement). Neighbouring triangles (which are not marked for refinement get split in half (‘green’ refinement). (R. Verfuerth, "A review of a posteriori error estimation and adaptive mesh refinement techniques", Wiley-Teubner, 1996)

surfaceSplitByPatch
Writes regions of triSurface to separate files.

surfaceSplitByTopology
Strips any baffle parts of a surface.

surfaceSplitNonManifolds
Takes multiply connected surface and tries to split surface at multiply connected edges by duplicating points. Introduces concept of - borderEdge. Edge with 4 faces connected to it. - borderPoint. Point connected to exactly 2 borderEdges. - borderLine. Connected list of borderEdges.

surfaceSubset
A surface analysis tool which sub-sets the triSurface to choose only a part of interest. Based on subsetMesh.

surfaceToPatch
Reads surface and applies surface regioning to a mesh. Uses boundaryMesh to do the hard work.

surfaceTransformPoints
Transform (scale/rotate) a surface. Like transformPoints but for surfaces.
3.7 Standard libraries

Continued from previous page

Parallel processing

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decomposePar</td>
<td>Automatically decomposes a mesh and fields of a case for parallel execution of OpenFOAM</td>
</tr>
<tr>
<td>reconstructPar</td>
<td>Reconstructs fields of a case that is decomposed for parallel execution of OpenFOAM</td>
</tr>
<tr>
<td>reconstructParMesh</td>
<td>Reconstructs a mesh using geometric information only</td>
</tr>
<tr>
<td>redistributePar</td>
<td>Redistributes existing decomposed mesh and fields according to the current settings in the decomposeParDict file</td>
</tr>
</tbody>
</table>

Thermophysical-related utilities

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adiabaticFlameT</td>
<td>Calculates the adiabatic flame temperature for a given fuel over a range of unburnt temperatures and equivalence ratios</td>
</tr>
<tr>
<td>chemkinToFoam</td>
<td>Converts CHEMKIN 3 thermodynamics and reaction data files into OpenFOAM format</td>
</tr>
<tr>
<td>equilibriumCO</td>
<td>Calculates the equilibrium level of carbon monoxide</td>
</tr>
<tr>
<td>equilibriumFlameT</td>
<td>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$_2$O and CO$_2$ are included</td>
</tr>
<tr>
<td>mixtureAdiabaticFlameT</td>
<td>Calculates the adiabatic flame temperature for a given mixture at a given temperature</td>
</tr>
</tbody>
</table>

Miscellaneous utilities

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expandDictionary</td>
<td>Read the dictionary provided as an argument, expand the macros etc. and write the resulting dictionary to standard output</td>
</tr>
<tr>
<td>foamDebugSwitches</td>
<td>Write out all library debug switches</td>
</tr>
<tr>
<td>foamFormatConvert</td>
<td>Converts all IOobjects associated with a case into the format specified in the controlDict</td>
</tr>
<tr>
<td>foamHelp</td>
<td>Top level wrapper utility around foam help utilities</td>
</tr>
<tr>
<td>foamInfoExec</td>
<td>Interrogates a case and prints information to stdout</td>
</tr>
<tr>
<td>patchSummary</td>
<td>Writes fields and boundary condition info for each patch at each requested time instance</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithms</td>
<td>Algorithms</td>
</tr>
<tr>
<td>containers</td>
<td>Container classes</td>
</tr>
<tr>
<td>db</td>
<td>Database classes</td>
</tr>
<tr>
<td>dimensionedTypes</td>
<td>dimensioned&lt;Type&gt; class and derivatives</td>
</tr>
<tr>
<td>dimensionSet</td>
<td>dimensionSet class</td>
</tr>
<tr>
<td>fields</td>
<td>Field classes</td>
</tr>
<tr>
<td>global</td>
<td>Global settings</td>
</tr>
<tr>
<td>graph</td>
<td>graph class</td>
</tr>
<tr>
<td>interpolations</td>
<td>Interpolation schemes</td>
</tr>
<tr>
<td>matrices</td>
<td>Matrix classes</td>
</tr>
<tr>
<td>memory</td>
<td>Memory management tools</td>
</tr>
<tr>
<td>meshes</td>
<td>Mesh classes</td>
</tr>
<tr>
<td>primitives</td>
<td>Primitive classes</td>
</tr>
</tbody>
</table>

### Finite volume method library — finiteVolume

<table>
<thead>
<tr>
<th>Category</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfdTools</td>
<td>CFD tools</td>
</tr>
<tr>
<td>fields</td>
<td>Volume, surface and patch field classes; includes boundary conditions</td>
</tr>
<tr>
<td>finiteVolume</td>
<td>Finite volume discretisation</td>
</tr>
<tr>
<td>fvMatrices</td>
<td>Matrices for finite volume solution</td>
</tr>
<tr>
<td>fvMesh</td>
<td>Meshes for finite volume discretisation</td>
</tr>
<tr>
<td>interpolation</td>
<td>Field interpolation and mapping</td>
</tr>
<tr>
<td>surfaceMesh</td>
<td>Mesh surface data for finite volume discretisation</td>
</tr>
<tr>
<td>volMesh</td>
<td>Mesh volume (cell) data for finite volume discretisation</td>
</tr>
</tbody>
</table>

### Post-processing libraries

<table>
<thead>
<tr>
<th>Category</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloudFunctionObjects</td>
<td>Function object outputs Lagrangian cloud information to a file</td>
</tr>
<tr>
<td>fieldFunctionObjects</td>
<td>Field function objects including field averaging, min/max, etc.</td>
</tr>
<tr>
<td>foamCalcFunctions</td>
<td>Functions for the foamCalc utility</td>
</tr>
<tr>
<td>forces</td>
<td>Tools for post-processing force/lift/drag data with function objects</td>
</tr>
<tr>
<td>FVFunctionObjects</td>
<td>Tools for calculating fvcDiv, fvcGrad etc with a function object</td>
</tr>
<tr>
<td>jobControl</td>
<td>Tools for controlling job running with a function object</td>
</tr>
<tr>
<td>postCalc</td>
<td>For using functionality of a function object as a post-processing activity</td>
</tr>
<tr>
<td>sampling</td>
<td>Tools for sampling field data at prescribed locations in a domain</td>
</tr>
</tbody>
</table>

*Continued on next page*
Continued from previous page

**systemCall**  General function object for making system calls while running a case

**utilityFunctionObjects**  Utility function objects

### Solution and mesh manipulation libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>autoMesh</strong></td>
<td>Library of functionality for the <em>snappyHexMesh</em> utility</td>
</tr>
<tr>
<td><strong>blockMesh</strong></td>
<td>Library of functionality for the <em>blockMesh</em> utility</td>
</tr>
<tr>
<td><strong>dynamicMesh</strong></td>
<td>For solving systems with moving meshes</td>
</tr>
<tr>
<td><strong>dynamicFvMesh</strong></td>
<td>Library for a finite volume mesh that can move and undergo topological changes</td>
</tr>
<tr>
<td><strong>edgeMesh</strong></td>
<td>For handling edge-based mesh descriptions</td>
</tr>
<tr>
<td><strong>fvMotionSolvers</strong></td>
<td>Finite volume mesh motion solvers</td>
</tr>
<tr>
<td><strong>ODE</strong></td>
<td>Solvers for ordinary differential equations</td>
</tr>
<tr>
<td><strong>meshTools</strong></td>
<td>Tools for handling a OpenFOAM mesh</td>
</tr>
<tr>
<td><strong>surfMesh</strong></td>
<td>Library for handling surface meshes of different formats</td>
</tr>
<tr>
<td><strong>triSurface</strong></td>
<td>For handling standard triangulated surface-based mesh descriptions</td>
</tr>
<tr>
<td><strong>topoChangerFvMesh</strong></td>
<td>Topological changes functionality (largely redundant)</td>
</tr>
</tbody>
</table>

### Lagrangian particle tracking libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>coalCombustion</strong></td>
<td>Coal dust combustion modelling</td>
</tr>
<tr>
<td><strong>distributionModels</strong></td>
<td>Particle distribution function modelling</td>
</tr>
<tr>
<td><strong>dsmc</strong></td>
<td>Direct simulation Monte Carlo method modelling</td>
</tr>
<tr>
<td><strong>lagrangian</strong></td>
<td>Basic Lagrangian, or particle-tracking, solution scheme</td>
</tr>
<tr>
<td><strong>lagrangianIntermediate</strong></td>
<td>Particle-tracking kinematics, thermodynamics, multispecies reactions, particle forces, etc.</td>
</tr>
<tr>
<td><strong>potential</strong></td>
<td>Intermolecular potentials for molecular dynamics</td>
</tr>
<tr>
<td><strong>molecule</strong></td>
<td>Molecule classes for molecular dynamics</td>
</tr>
<tr>
<td><strong>molecularMeasurements</strong></td>
<td>For making measurements in molecular dynamics</td>
</tr>
<tr>
<td><strong>solidParticle</strong></td>
<td>Solid particle implementation</td>
</tr>
<tr>
<td><strong>spray</strong></td>
<td>Spray and injection modelling</td>
</tr>
<tr>
<td><strong>turbulence</strong></td>
<td>Particle dispersion and Brownian motion based on turbulence</td>
</tr>
</tbody>
</table>

### Miscellaneous libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>conversion</strong></td>
<td>Tools for mesh and data conversions</td>
</tr>
<tr>
<td><strong>decompositionMethods</strong></td>
<td>Tools for domain decomposition</td>
</tr>
<tr>
<td><strong>engine</strong></td>
<td>Tools for engine calculations</td>
</tr>
<tr>
<td><strong>fileFormats</strong></td>
<td>Core routines for reading/writing data in some third-party formats</td>
</tr>
<tr>
<td><strong>genericFvPatchField</strong></td>
<td>A generic patch field</td>
</tr>
<tr>
<td><strong>MGridGenGAMG-Agglomera-</strong></td>
<td>Library for cell agglomeration using the <em>MGridGen</em> algorithm</td>
</tr>
<tr>
<td><strong>pairPatchAgglomeration</strong></td>
<td>Primitive pair patch agglomeration method</td>
</tr>
<tr>
<td><strong>OSspecific</strong></td>
<td>Operating system specific functions</td>
</tr>
<tr>
<td><strong>randomProcesses</strong></td>
<td>Tools for analysing and generating random processes</td>
</tr>
</tbody>
</table>

Continued on next page
Parallel libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decompose</td>
<td>General mesh/field decomposition library</td>
</tr>
<tr>
<td>distributed</td>
<td>Tools for searching and IO on distributed surfaces</td>
</tr>
<tr>
<td>metisDecomp</td>
<td>Metis domain decomposition library</td>
</tr>
<tr>
<td>reconstruct</td>
<td>Mesh/field reconstruction library</td>
</tr>
<tr>
<td>scotchDecomp</td>
<td>Scotch domain decomposition library</td>
</tr>
<tr>
<td>ptsotchDecomp</td>
<td>PTScotch domain decomposition library</td>
</tr>
</tbody>
</table>

Table 3.7: Shared object libraries for general use.

Basic thermophysical models — basicThermophysicalModels

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hePsiThermo</td>
<td>General thermophysical model calculation based on compressibility $\psi$</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>General thermophysical model calculation based on density $\rho$</td>
</tr>
<tr>
<td>pureMixture</td>
<td>General thermophysical model calculation for passive gas mixtures</td>
</tr>
</tbody>
</table>

Reaction models — reactionThermophysicalModels

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psiReactionThermo</td>
<td>Calculates enthalpy for combustion mixture based on $\psi$</td>
</tr>
<tr>
<td>psiuReactionThermo</td>
<td>Calculates enthalpy for combustion mixture based on $\psi_u$</td>
</tr>
<tr>
<td>rhoReactionThermo</td>
<td>Calculates enthalpy for combustion mixture based on $\rho$</td>
</tr>
<tr>
<td>heheuPsiThermo</td>
<td>Calculates enthalpy for unburnt gas and combustion mixture</td>
</tr>
<tr>
<td>homogeneousMixture</td>
<td>Combustion mixture based on normalised fuel mass fraction $b$</td>
</tr>
<tr>
<td>inhomogeneousMixture</td>
<td>Combustion mixture based on $b$ and total fuel mass fraction $f_t$</td>
</tr>
<tr>
<td>veryInhomogeneousMixture</td>
<td>Combustion mixture based on $b$, $f_t$, and unburnt fuel mass fraction $f_u$</td>
</tr>
<tr>
<td>basicMultiComponentMixture</td>
<td>Basic mixture based on multiple components</td>
</tr>
<tr>
<td>multiComponentMixture</td>
<td>Derived mixture based on multiple components</td>
</tr>
<tr>
<td>reactingMixture</td>
<td>Combustion mixture using thermodynamics and reaction schemes</td>
</tr>
<tr>
<td>egrMixture</td>
<td>Exhaust gas recirculation mixture</td>
</tr>
<tr>
<td>singleStepReactingMixture</td>
<td>Single step reacting mixture</td>
</tr>
</tbody>
</table>

Radiation models — radiationModels

Continued on next page
3.7 Standard libraries

Continued from previous page

P1 P1 model
fvDOM Finite volume discrete ordinate method
opaqueSolid Radiation for solid opaque solids; does nothing to energy equation source terms (returns zeros) but creates absorptionEmissionModel and scatterModel
viewFactor View factor radiation model

Laminar flame speed models — laminarFlameSpeedModels

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>Constant laminar flame speed</td>
</tr>
<tr>
<td>GuldersLaminarFlameSpeed</td>
<td>Gulder’s laminar flame speed model</td>
</tr>
<tr>
<td>GuldersEGRLaminarFlameSpeed</td>
<td>Gulder’s laminar flame speed model</td>
</tr>
<tr>
<td>RaviPetersen</td>
<td>Laminar flame speed obtained from Ravi and Petersen’s correlation</td>
</tr>
</tbody>
</table>

Barotropic compressibility models — barotropicCompressibilityModels

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>Linear compressibility model</td>
</tr>
<tr>
<td>Chung</td>
<td>Chung compressibility model</td>
</tr>
<tr>
<td>Wallis</td>
<td>Wallis compressibility model</td>
</tr>
</tbody>
</table>

Thermophysical properties of gaseous species — specie

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adiabaticPerfectFluid</td>
<td>Adiabatic perfect gas equation of state</td>
</tr>
<tr>
<td>icoPolynomial</td>
<td>Incompressible polynomial equation of state, e.g. for liquids</td>
</tr>
<tr>
<td>perfectFluid</td>
<td>Perfect gas equation of state</td>
</tr>
<tr>
<td>incompressiblePerfectGas</td>
<td>Incompressible gas equation of state using a constant reference pressure. Density only varies with temperature and composition</td>
</tr>
<tr>
<td>rhoConst</td>
<td>Constant density equation of state</td>
</tr>
<tr>
<td>eConstThermo</td>
<td>Constant specific heat (c_p) model with evaluation of internal energy (e) and entropy (s)</td>
</tr>
<tr>
<td>hConstThermo</td>
<td>Constant specific heat (c_p) model with evaluation of enthalpy (h) and entropy (s)</td>
</tr>
<tr>
<td>hPolynomialThermo</td>
<td>(c_p) evaluated by a function with coefficients from polynomials, from which (h), (s) are evaluated</td>
</tr>
<tr>
<td>janafThermo</td>
<td>(c_p) evaluated by a function with coefficients from JANAF thermodynamic tables, from which (h), (s) are evaluated</td>
</tr>
<tr>
<td>specieThermo</td>
<td>Thermophysical properties of species, derived from (c_p), (h) and/or (s)</td>
</tr>
<tr>
<td>constTransport</td>
<td>Constant transport properties</td>
</tr>
<tr>
<td>polynomialTransport</td>
<td>Polynomial based temperature-dependent transport properties</td>
</tr>
<tr>
<td>sutherlandTransport</td>
<td>Sutherland’s formula for temperature-dependent transport properties</td>
</tr>
</tbody>
</table>

Functions/tables of thermophysical properties — thermophysicalFunctions

Continued on next page
Continued from previous page

<table>
<thead>
<tr>
<th>NSRDSfunctions</th>
<th>National Standard Reference Data System (NSRDS) - American Institute of Chemical Engineers (AICHE) data compilation tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIfunctions</td>
<td>American Petroleum Institute (API) function for vapour mass diffusivity</td>
</tr>
</tbody>
</table>

Chemistry model — chemistryModel

<table>
<thead>
<tr>
<th>chemistryModel</th>
<th>Chemical reaction model</th>
</tr>
</thead>
<tbody>
<tr>
<td>chemistrySolver</td>
<td>Chemical reaction solver</td>
</tr>
</tbody>
</table>

Other libraries

| liquidProperties | Thermophysical properties of liquids |
| liquidMixtureProperties | Thermophysical properties of liquid mixtures |
| basicSolidThermo | Thermophysical models of solids |
| hExponentialThermo | Exponential properties thermodynamics package templated into the equationOfState |
| SLGThermo | Thermodynamic package for solids, liquids and gases |
| solidChemistryModel | Thermodynamic model of solid chemistry including pyrolysis |
| solidProperties | Thermophysical properties of solids |
| solidMixtureProperties | Thermophysical properties of solid mixtures |
| solidSpecie | Solid reaction rates and transport models |
| solidThermo | Solid energy modelling |

Table 3.8: Libraries of thermophysical models.

RAS turbulence models for incompressible fluids — incompressibleRASModels

| laminar | Dummy turbulence model for laminar flow |
| kEpsilon | Standard high-Re $k - \varepsilon$ model |
| kOmega | Standard high-Re $k - \omega$ model |
| kOmegaSST | $k - \omega$-SST model |
| RNGkEpsilon | RNG $k - \varepsilon$ model |
| NonlinearKEShih | Non-linear Shih $k - \varepsilon$ model |
| LienCubicKE | Lien cubic $k - \varepsilon$ model |
| qZeta | $q - \zeta$ model |
| kkLOmega | Low Reynolds-number k-kl-omega turbulence model for incompressible flows |
| 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 |
| LaunderGibsonRSTM | Launder-Gibson RSTM with wall-reflection terms |
| realizableKE | Realizable $k - \varepsilon$ model |
| SpalartAllmaras | Spalart-Allmaras 1-eqn mixing-length model |

Continued on next page
3.7 Standard libraries

Continued from previous page

v2f  Lien and Kalitzin’s v2-f turbulence model for incompressible flows

**RAS turbulence models for compressible fluids — compressibleRASModels**

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>laminar</td>
<td>Dummy turbulence model for laminar flow</td>
</tr>
<tr>
<td>kEpsilon</td>
<td>Standard $k - \varepsilon$ model</td>
</tr>
<tr>
<td>kOmegaSST</td>
<td>$k - \omega - SST$ model</td>
</tr>
<tr>
<td>RNGkEpsilon</td>
<td>RNG $k - \varepsilon$ model</td>
</tr>
<tr>
<td>LaunderSharmaKE</td>
<td>Launder-Sharma low-Re $k - \varepsilon$ model</td>
</tr>
<tr>
<td>LRR</td>
<td>Launder-Reece-Rodi RSTM</td>
</tr>
<tr>
<td>LaunderGibsonRSTM</td>
<td>Launder-Gibson RSTM</td>
</tr>
<tr>
<td>realizableKE</td>
<td>Realizable $k - \varepsilon$ model</td>
</tr>
<tr>
<td>SpalartAllmaras</td>
<td>Spalart-Allmaras 1-eqn mixing-length model</td>
</tr>
<tr>
<td>v2f</td>
<td>Lien and Kalitzin’s v2-f turbulence model for incompressible flows</td>
</tr>
</tbody>
</table>

**Large-eddy simulation (LES) filters — LESfilters**

<table>
<thead>
<tr>
<th>Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>laplaceFilter</td>
<td>Laplace filters</td>
</tr>
<tr>
<td>simpleFilter</td>
<td>Simple filter</td>
</tr>
<tr>
<td>anisotropicFilter</td>
<td>Anisotropic filter</td>
</tr>
</tbody>
</table>

**Large-eddy simulation deltas — LESdeltas**

<table>
<thead>
<tr>
<th>Delta</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrandtlDelta</td>
<td>Prandtl delta</td>
</tr>
<tr>
<td>cubeRootVolDelta</td>
<td>Cube root of cell volume delta</td>
</tr>
<tr>
<td>maxDeltaxyz</td>
<td>Maximum of x, y and z; for structured hex cells only</td>
</tr>
<tr>
<td>smoothDelta</td>
<td>Smoothing of delta</td>
</tr>
</tbody>
</table>

**Incompressible LES turbulence models — incompressibleLESModels**

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smagorinsky</td>
<td>Smagorinsky model</td>
</tr>
<tr>
<td>Smagorinsky2</td>
<td>Smagorinsky model with 3-D filter</td>
</tr>
<tr>
<td>homogenousDynSmagorinsky</td>
<td>Homogeneous dynamic Smagorinsky model</td>
</tr>
<tr>
<td>dynLagrangian</td>
<td>Lagrangian two equation eddy-viscosity model</td>
</tr>
<tr>
<td>scaleSimilarity</td>
<td>Scale similarity model</td>
</tr>
<tr>
<td>mixedSmagorinsky</td>
<td>Mixed Smagorinsky/scale similarity model</td>
</tr>
<tr>
<td>homogenousDynOneEqEddy</td>
<td>One Equation Eddy Viscosity Model for incompressible flows</td>
</tr>
<tr>
<td>laminar</td>
<td>Simply returns laminar properties</td>
</tr>
<tr>
<td>kOmegaSST SAS</td>
<td>$k - \omega$-SST scale adaptive simulation (SAS) model</td>
</tr>
<tr>
<td>oneEqEddy</td>
<td>$k$-equation eddy-viscosity model</td>
</tr>
<tr>
<td>dynOneEqEddy</td>
<td>Dynamic $k$-equation eddy-viscosity model</td>
</tr>
<tr>
<td>spectEddyVisc</td>
<td>Spectral eddy viscosity model</td>
</tr>
<tr>
<td>LRDStress</td>
<td>LRR differential stress model</td>
</tr>
<tr>
<td>DeardorffStress</td>
<td>Deardorff differential stress model</td>
</tr>
<tr>
<td>SpalartAllmaras</td>
<td>Spalart-Allmaras model</td>
</tr>
</tbody>
</table>

Continued on next page
Continued from previous page

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpalartAllmarasDDES</td>
<td>Spalart-Allmaras delayed detached eddy simulation (DDES) model</td>
</tr>
<tr>
<td>SpalartAllmarasIDDES</td>
<td>Spalart-Allmaras improved DDES (IDDES) model</td>
</tr>
<tr>
<td>vanDriestDelta</td>
<td>Simple cube-root of cell volume delta used in incompressible LES models</td>
</tr>
</tbody>
</table>

**Compressible LES turbulence models** — compressibleLESModels

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smagorinsky</td>
<td>Smagorinsky model</td>
</tr>
<tr>
<td>oneEqEddy</td>
<td>$k$-equation eddy-viscosity model</td>
</tr>
<tr>
<td>lowReOneEqEddy</td>
<td>Low-$Re$ $k$-equation eddy-viscosity model</td>
</tr>
<tr>
<td>homogenousDynOneEqEddy</td>
<td>One Equation Eddy Viscosity Model for incompressible flows</td>
</tr>
<tr>
<td>DeardorffDiffStress</td>
<td>Deardorff differential stress model</td>
</tr>
<tr>
<td>SpalartAllmaras</td>
<td>Spalart-Allmaras 1-eqn mixing-length model</td>
</tr>
<tr>
<td>vanDriestDelta</td>
<td>Simple cube-root of cell volume delta used in incompressible LES models</td>
</tr>
</tbody>
</table>

Table 3.9: Libraries of RAS and LES turbulence models.

**Transport models for incompressible fluids** — incompressibleTransportModels

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newtonian</td>
<td>Linear viscous fluid model</td>
</tr>
<tr>
<td>CrossPowerLaw</td>
<td>Cross Power law nonlinear viscous model</td>
</tr>
<tr>
<td>BirdCarreau</td>
<td>Bird-Carreau nonlinear viscous model</td>
</tr>
<tr>
<td>HerschelBulkley</td>
<td>Herschel-Bulkley nonlinear viscous model</td>
</tr>
<tr>
<td>powerLaw</td>
<td>Power-law nonlinear viscous model</td>
</tr>
<tr>
<td>interfaceProperties</td>
<td>Models for the interface, <em>e.g.</em> contact angle, in multiphase simulations</td>
</tr>
</tbody>
</table>

Miscellaneous transport modelling libraries

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interfaceProperties</td>
<td>Calculation of interface properties</td>
</tr>
<tr>
<td>twoPhaseProperties</td>
<td>Two phase properties models, including boundary conditions</td>
</tr>
<tr>
<td>surfaceFilmModels</td>
<td>Surface film models</td>
</tr>
</tbody>
</table>

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-3.0.1 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-3.0.1/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
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 \( \mathbf{U} \) and pressure field \( p \) are initialised from files 0/\( \mathbf{U} \) 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 OpenFOAM file format is described in the following sections.

4.2.1 General syntax rules

The format follows 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 Basic input/output file format

4.2.2 Dictionaries

OpenFOAM uses *dictionaries* as the most common means of specifying data. A dictionary is an entity that contains 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 by keyword 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, 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 \( \text{p} \) and \( \text{U} \) keywords respectively; the \textit{PISO} dictionary contains algorithm controls.

```plaintext
solvers
{
    p
    {
        solver PCG;
        preconditioner DIC;
        tolerance 1e-06;
        relTol 0;
    }
    U
    {
        solver smoothSolver;
        smoother symGaussSeidel;
        tolerance 1e-05;
        relTol 0;
    }
    PISO
    {
        nCorrectors 2;
        nNonOrthogonalCorrectors 0;
        pRefCell 0;
        pRefValue 0;
    }
}

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

4.2.4 Lists

OpenFOAM applications contain lists, \textit{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 preceding the round braces:

\textbf{simple} the keyword is followed immediately by round braces

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

\textbf{numbered} the keyword is followed by the number of elements \(<n>\) in the list

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

\textbf{token identifier} the keyword is followed by a class name identifier \texttt{Label\textless Type\textgreater} where \texttt{<Type>} states what the list contains, \textit{e.g.} for a list of \texttt{scalar} elements is

```plaintext
<listName>
List<scalar>
```
4.2 Basic input/output file format

<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 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 Dimensional units

In continuum mechanics, properties are represented in some chosen units, e.g. mass in kilograms (kg), volume in cubic metres (m³), pressure in Pascals (kg m⁻¹ s⁻²). Algebraic operations must be performed on these properties using consistent units of measurement; in particular, addition, subtraction and equality are only physically meaningful for properties of the same dimensional units. As a safeguard against implementing a meaningless operation, OpenFOAM attaches dimensions to field data and physical properties and performs dimension checking on any tensor operation.

The I/O format for a dimensionSet is 7 scalars delimited by square brackets, e.g.

[0 2 -1 0 0 0 0]
where each of the values corresponds to the power of each of the base units of measurement listed in Table 4.2. The table gives the base units for the Système International (SI) and the United States Customary System (USCS) but OpenFOAM can be used with any system of units. All that is required is that the input data is correct for the chosen set of units. It is particularly important to recognise that OpenFOAM requires some dimensioned physical constants, e.g. the Universal Gas Constant $R$, for certain calculations, e.g. thermophysical modelling. These dimensioned constants are specified in a `DimensionedConstant` sub-dictionary of main `controlDict` file of the OpenFOAM installation (`$WM_PROJECT_DIR/etc/controlDict`). By default these constants are set in SI units. Those wishing to use the USCS or any other system of units should modify these constants to their chosen set of units accordingly.

### 4.2.7 Dimensioned types

Physical properties are typically specified with their associated dimensions. These entries have the format that the following example of a `dimensionedScalar` demonstrates:

$$
\text{nu} \quad \text{nu} \quad [0 \ 2 \ -1 \ 0 \ 0 \ 0 \ 0] \quad 1;
$$

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.

### 4.2.8 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. OpenFOAM writes field data using keyword entries as described in Table 4.3.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimensions</td>
<td>Dimensions of field</td>
<td><code>[1 1 -2 0 0 0 0]</code></td>
</tr>
<tr>
<td>internalField</td>
<td>Value of internal field</td>
<td><code>uniform (1 0 0)</code></td>
</tr>
<tr>
<td>boundaryField</td>
<td>Boundary field</td>
<td>see file listing in section 4.2.8</td>
</tr>
</tbody>
</table>

Table 4.3: Main keywords used in field dictionaries.

The data begins with an entry for its `dimensions`. Following that, is the `internalField`, described in one of the following ways.

---

OpenFOAM-3.0.1
4.2 Basic input/output file format

**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. A selection of patch field conditions available in OpenFOAM are listed in Table 5.3 and Table 5.4 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 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;
  }
}
```

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

4.2.9 Directives and macro substitutions

There is additional file syntax that offers great flexibility for the setting up of OpenFOAM case files, namely directives and macro substitutions. Directives are commands that can be contained within case files that begin with the hash (#) symbol. Macro substitutions begin with the dollar ($) symbol.

At present there are 4 directive commands available in OpenFOAM:

- `#include "<fileName>"` (or `#includeIfPresent "<fileName>"`) reads the file of name `<fileName>`;

- `#inputMode` has two options: `merge`, which merges keyword entries in successive dictionaries, so that a keyword entry specified in one place will be overridden by a later specification of the same keyword entry; `overwrite`, which overwrites the contents of an entire dictionary; generally, use `merge`;
#remove <keywordEntry> removes any included keyword entry; can take a word or regular expression;

#codeStream followed by verbatim C++ code, compiles, loads and executes the code on-the-fly to generate the entry.

### 4.2.10 The #include and #inputMode directives

For example, let us say a user wishes to set an initial value of pressure once to be used as the internal field and initial value at a boundary. We could create a file, *e.g.* named `initialConditions`, which contains the following entries:

```plaintext
pressure 1e+05;
#inputMode merge
```

In order to use this pressure for both the internal and initial boundary fields, the user would simply include the following macro substitutions in the pressure field file `p`:

```plaintext
#include "initialConditions"
internalField uniform $pressure;
boundaryField
{
    patch1
    {
        type fixedValue;
        value $internalField;
    }
}
```

This is a fairly trivial example that simply demonstrates how this functionality works. However, the functionality can be used in many, more powerful ways particularly as a means of generalising case data to suit the user’s needs. For example, if a user has a set of cases that require the same turbulence model settings, a single file can be created with those settings which is simply included in the `turbulenceProperties` file of each case. Macro substitutions can extend well beyond a single value so that, for example, sets of boundary conditions can be predefined and called by a single macro. The extent to which such functionality can be used is almost endless.

### 4.2.11 The #codeStream directive

The #codeStream directive takes C++ code which is compiled and executed to deliver the dictionary entry. The code and compilation instructions are specified through the following keywords.

- **code**: specifies the code, called with arguments `OStream& os` and `const dictionary& dict` which the user can use in the code, *e.g.* to lookup keyword entries from within the current case dictionary (file).
4.3 Time and data input/output control

- **codeInclude** (optional): specifies additional C++ `#include` statements to include OpenFOAM files.

- **codeOptions** (optional): specifies any extra compilation flags to be added to `EXE_INC` in `Make/options`.

- **codeLibs** (optional): specifies any extra compilation flags to be added to `LIB_LIBS` in `Make/options`.

Code, like any string, can be written across multiple lines by enclosing it within hash-bracket delimiters, i.e. `#{}...#`. Anything in between these two delimiters becomes a string with all newlines, quotes, etc. preserved.

An example of `#codeStream` is given below. The code in the `controlDict` file looks up dictionary entries and does a simple calculation for the write interval:

```cpp
startTime 0;
endTime 100;
...
writeInterval #codeStream
{
  code
  #{
    scalar start = readScalar(dict.lookup("startTime"));
    scalar end = readScalar(dict.lookup("endTime"));
    label nDumps = 5;
    os << ((end - start)/nDumps);
  #};
};
```

### 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 4.4. Only the time control and `writeInterval` entries are truly compulsory, with the database taking default values indicated by † in Table 4.4 for any of the optional entries that are omitted.

<table>
<thead>
<tr>
<th>Time control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>startFrom</strong> Controls the start time of the simulation.</td>
</tr>
<tr>
<td>- <strong>firstTime</strong> Earliest time step from the set of time directories.</td>
</tr>
<tr>
<td>- <strong>startTime</strong> Time specified by the <code>startTime</code> keyword entry.</td>
</tr>
<tr>
<td>- <strong>latestTime</strong> Most recent time step from the set of time directories.</td>
</tr>
<tr>
<td><strong>startTime</strong> Start time for the simulation with <code>startFrom startTime</code>;</td>
</tr>
<tr>
<td><strong>stopAt</strong> Controls the end time of the simulation.</td>
</tr>
<tr>
<td>- <strong>endTime</strong> Time specified by the <code>endTime</code> keyword entry.</td>
</tr>
<tr>
<td>- <strong>writeNow</strong> Stops simulation on completion of current time step and writes data.</td>
</tr>
</tbody>
</table>

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 $t_0 = 5s$, $\Delta t = 1s$ 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** $\pm m.ddddde \pm xx$ 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
4.3 Time and data input/output control

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.

Adjustable time step

adjustTimeStep yes†/no switch for OpenFOAM to adjust the time step during the simulation, usually according to...

maxCo Maximum Courant number, e.g. 0.5

Data reading

runTimeModifiable yes†/no switch for whether dictionaries, e.g. controlDict, are re-read by OpenFOAM at the beginning of each time step.

Run-time loadable functionality

libs List of additional libraries (on $LD_LIBRARY_PATH) to be loaded at run-time, e.g. ( "libUser1.so" "libUser2.so" )

functions List of functions, e.g. probes to be loaded at run-time; see examples in $FOAM_TUTORIALS

† denotes default entry if associated keyword is omitted.

Table 4.4: Keyword entries in the controlDict dictionary.

Example entries from a controlDict dictionary are given below:

17 application icoFoam;
18 startFrom startTime;
19 startTime 0;
20 stopAt endTime;
21 endTime 0.5;
22 deltaT 0.005;
23 writeControl timeStep;
24 writeInterval 20;
25 purgeWrite 0;
26 writeFormat ascii;
27 writePrecision 6;
28 writeCompression off;
29 timeFormat general;
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.

The terms that must typically be assigned a numerical scheme in `fvSchemes` range from derivatives, e.g. gradient $\nabla$, 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 $\nabla \cdot$ terms.

The set of terms, for which numerical schemes must be specified, are subdivided within the `fvSchemes` dictionary into the categories listed in Table 4.5. Each keyword in Table 4.5 is the name of a sub-dictionary which contains terms of a particular type, e.g. `gradSchemes` contains all the gradient derivative terms such as $\text{grad}(p)$ (which represents $\nabla p$). Further examples can be seen in the extract from an `fvSchemes` dictionary below:

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

Table 4.5: Main keywords used in `fvSchemes`.

```
<table>
<thead>
<tr>
<th>17</th>
<th>ddtSchemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>{</td>
</tr>
<tr>
<td>19</td>
<td>default</td>
</tr>
<tr>
<td>20</td>
<td>Euler;</td>
</tr>
<tr>
<td>21</td>
<td>}</td>
</tr>
<tr>
<td>22</td>
<td>gradSchemes</td>
</tr>
<tr>
<td>23</td>
<td>{</td>
</tr>
<tr>
<td>24</td>
<td>default</td>
</tr>
<tr>
<td>25</td>
<td>Gauss linear;</td>
</tr>
<tr>
<td>26</td>
<td>grad(p)</td>
</tr>
<tr>
<td>27</td>
<td>Gauss linear;</td>
</tr>
<tr>
<td>28</td>
<td>}</td>
</tr>
<tr>
<td>29</td>
<td>divSchemes</td>
</tr>
</tbody>
</table>
```

OpenFOAM-3.0.1
4.4 Numerical schemes

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 sub-dictionary 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 sub-dictionary, it is assigned to all of the terms to which the sub-dictionary refers, e.g. specifying a default in gradSchemes sets the scheme for all gradient terms in the application, e.g. ∇p, ∇U. When a default is specified, it is not necessary to specify each specific term itself in that sub-dictionary, 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 sub-dictionary 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 4.5.

### 4.4.1 Interpolation schemes

The interpolationSchemes sub-dictionary 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 4.6, 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 sub-dictionary, 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 4.6.
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 \( \phi \). 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 phi 1.0;
```

### 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 preceded 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;
```

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 limiter 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.
### 4.4 Numerical schemes

#### 4.4.1 Centred schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>Linear interpolation (central differencing)</td>
</tr>
<tr>
<td>cubicCorrection</td>
<td>Cubic scheme</td>
</tr>
<tr>
<td>midPoint</td>
<td>Linear interpolation with symmetric weighting</td>
</tr>
</tbody>
</table>

#### 4.4.2 Upwined convection schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>upwind</td>
<td>Upwind differencing</td>
</tr>
<tr>
<td>linearUpwind</td>
<td>Linear upwind differencing</td>
</tr>
<tr>
<td>skewLinear</td>
<td>Linear with skewness correction</td>
</tr>
<tr>
<td>filteredLinear2</td>
<td>Linear with filtering for high-frequency ringing</td>
</tr>
</tbody>
</table>

#### 4.4.3 TVD schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>limitedLinear</td>
<td>limited linear differencing</td>
</tr>
<tr>
<td>vanLeer</td>
<td>van Leer limiter</td>
</tr>
<tr>
<td>MUSCL</td>
<td>MUSCL limiter</td>
</tr>
<tr>
<td>limitedCubic</td>
<td>Cubic limiter</td>
</tr>
</tbody>
</table>

#### 4.4.4 NVD schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFCD</td>
<td>Self-filtered central differencing</td>
</tr>
<tr>
<td>Gamma ψ</td>
<td>Gamma differencing</td>
</tr>
</tbody>
</table>

Table 4.6: Interpolation schemes.

### 4.4.2 Surface normal gradient schemes

The `snGradSchemes` sub-dictionary 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 4.7 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

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

Equation (4.1)

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

```plaintext
default limited 0.5;
```
### 4.4.3 Gradient schemes

The `gradSchemes` sub-dictionary contains gradient terms. The discretisation scheme for each term can be selected from those listed in Table 4.8.

<table>
<thead>
<tr>
<th>Discretisation scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss <code>&lt;interpolationScheme&gt;</code></td>
<td>Second order, Gaussian integration</td>
</tr>
<tr>
<td>leastSquares</td>
<td>Second order, least squares</td>
</tr>
<tr>
<td>fourth</td>
<td>Fourth order, least squares</td>
</tr>
<tr>
<td>cellLimited <code>&lt;gradScheme&gt;</code></td>
<td>Cell limited version of one of the above schemes</td>
</tr>
<tr>
<td>faceLimited <code>&lt;gradScheme&gt;</code></td>
<td>Face limited version of one of the above schemes</td>
</tr>
</tbody>
</table>

Table 4.8: 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 4.6. 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 `cellLimited` (or `faceLimited`), *e.g.* a cell limited Gauss scheme

```
grad(p) cellLimited Gauss linear 1;
```

### 4.4.4 Laplacian schemes

The `laplacianSchemes` sub-dictionary contains Laplacian terms. Let us discuss the syntax of the entry in reference to a typical Laplacian term found in fluid dynamics, $\nabla \cdot (\nu \nabla \mathbf{U})$, 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.* \( \nu \) in our example, and a surface normal gradient scheme, *i.e.* \( \nabla \mathbf{U} \). To summarise, the entries required are:

\[
\text{Gauss} \langle \text{interpolationScheme} \rangle \langle \text{snGradScheme} \rangle
\]

The interpolation scheme is selected from Table 4.6, the typical choices being from the general schemes and, in most cases, \text{linear}. The surface normal gradient scheme is selected from Table 4.7; the choice of scheme determines numerical behaviour as described in Table 4.9. A typical entry for our example Laplacian term would be:

\[
laplacian(\nu, \mathbf{U}) \text{ Gauss linear corrected};
\]

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Numerical behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>corrected</td>
<td>Unbounded, second order, conservative</td>
</tr>
<tr>
<td>uncorrected</td>
<td>Bounded, first order, non-conservative</td>
</tr>
<tr>
<td>limited ( \psi )</td>
<td>Blend of corrected and uncorrected</td>
</tr>
<tr>
<td>bounded</td>
<td>First order for bounded scalars</td>
</tr>
<tr>
<td>fourth</td>
<td>Unbounded, fourth order, conservative</td>
</tr>
</tbody>
</table>

Table 4.9: Behaviour of surface normal schemes used in \text{laplacianSchemes}.

### 4.4.5 Divergence schemes

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

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

\[
\text{Gauss} \langle \text{interpolationScheme} \rangle
\]

The interpolation scheme is selected from the full range of schemes in Table 4.6, both general and convection-specific. The choice critically determines numerical behaviour as described in Table 4.10. 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 \text{div}(\phi, \mathbf{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:

\[
\text{div}(\phi, \mathbf{U}) \text{ Gauss upwind};
\]
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Numerical behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>Second order, unbounded</td>
</tr>
<tr>
<td>skewLinear</td>
<td>Second order, (more) unbounded, skewness correction</td>
</tr>
<tr>
<td>cubicCorrected</td>
<td>Fourth order, unbounded</td>
</tr>
<tr>
<td>upwind</td>
<td>First order, bounded</td>
</tr>
<tr>
<td>linearUpwind</td>
<td>First/second order, bounded</td>
</tr>
<tr>
<td>QUICK</td>
<td>First/second order, bounded</td>
</tr>
<tr>
<td>TVD schemes</td>
<td>First/second order, bounded</td>
</tr>
<tr>
<td>SFCD</td>
<td>Second order, bounded</td>
</tr>
<tr>
<td>NVD schemes</td>
<td>First/second order, bounded</td>
</tr>
</tbody>
</table>

Table 4.10: 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 sub-dictionary. The discretisation scheme for each term can be selected from those listed in Table 4.11.

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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euler</td>
<td>First order, bounded, implicit</td>
</tr>
<tr>
<td>localEuler</td>
<td>Local-time step, first order, bounded, implicit</td>
</tr>
<tr>
<td>CrankNicolson (\psi)</td>
<td>Second order, bounded, implicit</td>
</tr>
<tr>
<td>backward</td>
<td>Second order, implicit</td>
</tr>
<tr>
<td>steadyState</td>
<td>Does not solve for time derivatives</td>
</tr>
</tbody>
</table>

Table 4.11: 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 sub-dictionary. Only the Euler scheme is available for d2dt2Schemes.

4.4.7 Flux calculation

The fluxRequired sub-dictionary 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 sub-dictionary would simply be entered as follows, \(p\) being the word identifier for pressure:

```plaintext
fluxRequired
{

```

OpenFOAM-3.0.1
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.

```plaintext
solvers
{
  p
  {
    solver PCG;
    preconditioner DIC;
    tolerance 1e-06;
    relTol 0;
  }
  U
  {
    solver smoothSolver;
    smoother symGaussSeidel;
    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 sub-dictionary 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* uses 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 keyword is followed by a dictionary containing the type of solver and the parameters that the solver uses. The solver is selected through the *solver* keyword from the choice in OpenFOAM, listed
in Table 4.12. The parameters, including tolerance, relTol, preconditioner, etc. are described in following sections.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditioned (bi-)conjugate gradient</td>
<td>PCG/PBiCG†</td>
</tr>
<tr>
<td>Solver using a smoother</td>
<td>smoothSolver</td>
</tr>
<tr>
<td>Generalised geometric-algebraic multi-grid</td>
<td>GAMG</td>
</tr>
<tr>
<td>Diagonal solver for explicit systems</td>
<td>diagonal</td>
</tr>
</tbody>
</table>

†PCG for symmetric matrices, PBiCG for asymmetric

Table 4.12: Linear solvers.

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 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 ID 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 to make it independent of the scale of the 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 number of iterations exceeds a maximum number of iterations, maxIter;

The solver tolerance should represent 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. In transient simulations, it is usual to set the solver relative tolerance to 0 to force the solution to converge to the solver tolerance in each time step. The tolerances, tolerance and relTol must be specified in the dictionaries for all solvers; maxIter is optional.
4.5 Solution and algorithm control

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 4.13.

<table>
<thead>
<tr>
<th>Preconditioner</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal incomplete-Cholesky (symmetric)</td>
<td>DIC</td>
</tr>
<tr>
<td>Faster diagonal incomplete-Cholesky (DIC with caching)</td>
<td>FDIC</td>
</tr>
<tr>
<td>Diagonal incomplete-LU (asymmetric)</td>
<td>DILU</td>
</tr>
<tr>
<td>Diagonal</td>
<td>diagonal</td>
</tr>
<tr>
<td>Geometric-algebraic multi-grid</td>
<td>GAMG</td>
</tr>
<tr>
<td>No preconditioning</td>
<td>none</td>
</tr>
</tbody>
</table>

Table 4.13: 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 4.14. 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.

<table>
<thead>
<tr>
<th>Smoother</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss-Seidel</td>
<td>GaussSeidel</td>
</tr>
<tr>
<td>Diagonal incomplete-Cholesky (symmetric)</td>
<td>DIC</td>
</tr>
<tr>
<td>Diagonal incomplete-Cholesky with Gauss-Seidel (symmetric)</td>
<td>DICGaussSeidel</td>
</tr>
</tbody>
</table>

Table 4.14: Smoother options.

The user must also specify 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 sub-dictionary 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, as described below.

- No specified $\alpha$: no under-relaxation.
- $\alpha = 1$: guaranteed matrix diagonal equality/dominance.
- $\alpha$ decreases, under-relaxation increases.
- $\alpha = 0$: solution does not change with successive iterations.

An optimum choice of $\alpha$ is one that is small enough to ensure stable computation but large enough to move the iterative process forward quickly; values of $\alpha$ 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.

```plaintext
solvers
{
    p
    {
        solver GAMG;
        tolerance 1e-06;
        relTol 0.1;
        smoother GaussSeidel;
        nPreSweeps 0;
        nPostSweeps 2;
        cacheAgglomeration on;
        agglomerator faceAreaPair;
    }
```

OpenFOAM-3.0.1
4.5 Solution and algorithm control

```csharp
nCellsInCoarsestLevel 10;
mergeLevels 1;

"(U|k|epsilon|omega|f|v2)"
{
  solver smoothSolver;
  smoother symGaussSeidel;
  tolerance 1e-05;
  relTol 0.1;
}

SIMPLE
{
  nNonOrthogonalCorrectors 0;
  consistent yes;

  residualControl
  {
    P 1e-2;
    U 1e-3;
    "(k|epsilon|omega|f|v2)" 1e-3;
  }

  relaxationFactors
  {
    equations
    {
      U 0.9; // 0.9 is more stable but 0.95 more convergent
      ".*" 0.9; // 0.9 is more stable but 0.95 more convergent
    }
  }
}

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

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-123.

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-123. 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 \( p_{\text{rgh}} \), the names are \( p_{\text{rghRefValue}} \) and \( p_{\text{rghRefCell}} \) respectively. These entries are generally stored in the \textit{PISO/SIMPLE} sub-dictionary and are used by those solvers that require them when the case demands it. If omitted, the solver will not run, but give a message to alert the user to the problem.

### 4.5.4 Other parameters

The \textit{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 \textit{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 sub-dictionary 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

Mesh generation and conversion

This chapter describes all topics relating to the creation of meshes in OpenFOAM: section 5.1 gives an overview of the ways a mesh may be described in OpenFOAM; section 5.3 covers the blockMesh utility for generating simple meshes of blocks of hexahedral cells; section 5.4 covers the snappyHexMesh utility for generating complex meshes of hexahedral and split-hexahedral cells automatically from triangulated surface geometries; section 5.5 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.

5.1 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. 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.

5.1.1 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:
5.1.1.1 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.*

5.1.1.2 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 5.1.

![Figure 5.1: Face area vector from point numbering on the face](image)

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.
5.1.1.3 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 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 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 $90^\circ$.

5.1.1.4 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.

5.1.2 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 faces 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;
        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.

    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.

5.1.3 The cellShape tools

We shall describe the alternative cellShape tools that may be used particularly when con-
verting 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 5.1. 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 5.1.
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 than 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:

```
OpenFOAM-3.0.1
```
Here the hexahedral cell shape is declared using the keyword `hex`. Other shapes are described by the keywords listed in Table 5.1.

### 5.1.4 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 5.2.2 and the generation of wedge geometries for axi-symmetric problems is discussed in section 5.3.3.

### 5.2 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, 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.
<table>
<thead>
<tr>
<th>Cell type</th>
<th>Keyword</th>
<th>Vertex numbering</th>
<th>Face numbering</th>
<th>Edge numbering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hexahedron</td>
<td>hex</td>
<td><img src="image" alt="Hexahedron" /></td>
<td><img src="image" alt="Hexahedron" /></td>
<td><img src="image" alt="Hexahedron" /></td>
</tr>
<tr>
<td>Wedge</td>
<td>wedge</td>
<td><img src="image" alt="Wedge" /></td>
<td><img src="image" alt="Wedge" /></td>
<td><img src="image" alt="Wedge" /></td>
</tr>
<tr>
<td>Prism</td>
<td>prism</td>
<td><img src="image" alt="Prism" /></td>
<td><img src="image" alt="Prism" /></td>
<td><img src="image" alt="Prism" /></td>
</tr>
<tr>
<td>Pyramid</td>
<td>pyr</td>
<td><img src="image" alt="Pyramid" /></td>
<td><img src="image" alt="Pyramid" /></td>
<td><img src="image" alt="Pyramid" /></td>
</tr>
<tr>
<td>Tetrahedron</td>
<td>tet</td>
<td><img src="image" alt="Tetrahedron" /></td>
<td><img src="image" alt="Tetrahedron" /></td>
<td><img src="image" alt="Tetrahedron" /></td>
</tr>
<tr>
<td>Tet-wedge</td>
<td>tetWedge</td>
<td><img src="image" alt="Tet-wedge" /></td>
<td><img src="image" alt="Tet-wedge" /></td>
<td><img src="image" alt="Tet-wedge" /></td>
</tr>
</tbody>
</table>

Table 5.1: Vertex, face and edge numbering for cellShapes.
There are three attributes associated with a patch that are described below in their natural hierarchy and Figure 5.2 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.

### 5.2.1 Specification of patch types in OpenFOAM

The patch types are specified in the mesh and field files of an 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.

An example `boundary` file is shown below for a sonicFoam case, followed by a pressure field file, \( p \), for the same case:

```plaintext
...{inlet

  type patch;
  nFaces 50;
  startFace 10325;
}

outlet

  type patch;
  nFaces 40;
  startFace 10375;
}

bottom

  type symmetryPlane;
  inGroups 1(symmetryPlane);
  nFaces 25;
  startFace 10415;
}

top

  type symmetryPlane;
  inGroups 1(symmetryPlane);
  nFaces 125;
  startFace 10440;
}

obstacle

  type patch;
  nFaces 110;
  startFace 10565;
}

defaultFaces

  type empty;
...
The type in the boundary file is `patch` for all patches except those 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.

### 5.2.2 Base types

The base and geometric types are described below; the keywords used for specifying these types in OpenFOAM are summarised in Table 5.2.

**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.
5.2 Boundaries

![Diagram of a wedge patch aligned along coordinate plane](Figure 5.3: Axi-symmetric geometry using the wedge patch type.)

<table>
<thead>
<tr>
<th>Selection Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patch</td>
<td>generic patch</td>
</tr>
<tr>
<td>symmetryPlane</td>
<td>plane of symmetry</td>
</tr>
<tr>
<td>empty</td>
<td>front and back planes of a 2D geometry</td>
</tr>
<tr>
<td>wedge</td>
<td>wedge front and back for an axi-symmetric geometry</td>
</tr>
<tr>
<td>cyclic</td>
<td>cyclic plane</td>
</tr>
<tr>
<td>wall</td>
<td>wall — used for wall functions in turbulent flows</td>
</tr>
<tr>
<td>processor</td>
<td>inter-processor boundary</td>
</tr>
</tbody>
</table>

Table 5.2: Basic patch types.

**wall** There are instances where a patch that coincides with a wall needs to be identifiable as such, particularly where specialist modelling is applied at wall boundaries. A good example is wall turbulence modelling where a wall must be specified with a wall patch type, so that the distance from the wall to 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 small angle (e.g. < 5°) and 1 cell thick running along the plane of symmetry, straddling one of the coordinate planes, as shown in Figure 5.3. 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 5.3.3.

**cyclic** Enables two patches to be treated as if they are physically connected; used for repeated geometries, e.g. heat exchanger tube bundles. One cyclic patch is linked to another.
through a `neighbourPatch` keyword in the `boundary` file. Each pair of connecting faces must have similar area to within a tolerance given by the `matchTolerance` keyword in the `boundary` file. 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.

### 5.2.3 Primitive types

The primitive types are listed in Table 5.3.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description of condition for patch field $\phi$</th>
<th>Data to specify</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fixedValue</code></td>
<td>Value of $\phi$ is specified</td>
<td><code>value</code></td>
</tr>
<tr>
<td><code>fixedGradient</code></td>
<td>Normal gradient of $\phi$ is specified</td>
<td><code>gradient</code></td>
</tr>
<tr>
<td><code>zeroGradient</code></td>
<td>Normal gradient of $\phi$ is zero</td>
<td>—</td>
</tr>
<tr>
<td><code>calculated</code></td>
<td>Boundary field $\phi$ derived from other fields</td>
<td>—</td>
</tr>
<tr>
<td><code>mixed</code></td>
<td>Mixed <code>fixedValue</code>/ <code>fixedGradient</code> condition depending on the value in <code>valueFraction</code></td>
<td><code>refValue, refGradient, value, valueFraction</code></td>
</tr>
</tbody>
</table>

**directionMixed** A mixed condition with tensorial `valueFraction`, e.g. for different levels of mixing in normal and tangential directions

<table>
<thead>
<tr>
<th>Type</th>
<th>Description of condition for patch field $\phi$</th>
<th>Data to specify</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>directionMixed</code></td>
<td>A mixed condition with tensorial <code>valueFraction</code>, e.g. for different levels of mixing in normal and tangential directions</td>
<td><code>refValue, refGradient, valueFraction, value</code></td>
</tr>
</tbody>
</table>

Table 5.3: Primitive patch field types.

### 5.2.4 Derived types

There are numerous derived types of boundary conditions in OpenFOAM, too many to list here. Instead a small selection is listed in Table 5.4. If the user wishes to obtain a list of all available models, they should consult the OpenFOAM source code. Derived boundary condition source code can be found at the following locations:

- in `$FOAM_SRC/finiteVolume/fields/fvPatchFields/derived`
- within certain model libraries, that can be located by typing the following command in a terminal window
  
  ```sh
  find $FOAM_SRC -name "*derivedFvPatch*"
  ```
  
  within certain solvers, that can be located by typing the following command in a terminal window
  
  ```sh
  find $FOAM_SOLVERS -name "*fvPatch*"
  ```
### Mesh generation with the blockMesh utility

**Types derived from fixedValue**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Data to specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>movingWallVelocity</td>
<td>Replaces the normal of the patch value so the flux across the patch is zero</td>
<td>value</td>
</tr>
<tr>
<td>pressureInletVelocity</td>
<td>When ( p ) is known at inlet, ( \mathbf{U} ) is evaluated from the flux, normal to the patch</td>
<td>value</td>
</tr>
<tr>
<td>pressureDirectedInletVelocity</td>
<td>When ( p ) is known at inlet, ( \mathbf{U} ) is calculated from the flux in the ( \text{inletDirection} ) value</td>
<td>inletDirection</td>
</tr>
<tr>
<td>surfaceNormalFixedValue</td>
<td>Specifies a vector boundary condition, normal to the patch, by its magnitude; +ve for vectors pointing out of the domain</td>
<td>value</td>
</tr>
<tr>
<td>totalPressure</td>
<td>Total pressure ( p_0 = p + \frac{1}{2}\rho</td>
<td>\mathbf{U}</td>
</tr>
<tr>
<td>turbulentInlet</td>
<td>Calculates a fluctuating variable based on a scale of a mean value</td>
<td>referenceField, fluctuationScale</td>
</tr>
</tbody>
</table>

**Types derived from fixedGradient/zeroGradient**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Data to specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>fluxCorrectedVelocity</td>
<td>Calculates normal component of ( \mathbf{U} ) at inlet from flux</td>
<td>value</td>
</tr>
<tr>
<td>buoyantPressure</td>
<td>Sets fixedGradient pressure based on the atmospheric pressure gradient</td>
<td>—</td>
</tr>
</tbody>
</table>

**Types derived from mixed**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Data to specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>inletOutlet</td>
<td>Switches ( \mathbf{U} ) and ( p ) between fixedValue and zeroGradient depending on direction of ( \mathbf{U} ) value</td>
<td>inletValue, value</td>
</tr>
<tr>
<td>outletInlet</td>
<td>Switches ( \mathbf{U} ) and ( p ) between fixedValue and zeroGradient depending on direction of ( \mathbf{U} ) value</td>
<td>outletValue, value</td>
</tr>
<tr>
<td>pressureInletOutletVelocity</td>
<td>Combination of pressureInletVelocity and inletOutlet value</td>
<td>value</td>
</tr>
<tr>
<td>pressureDirectedInletOutletVelocity</td>
<td>Combination of pressureDirectedInletVelocity and inletOutlet value</td>
<td>value, inletDirection</td>
</tr>
<tr>
<td>pressureTransmissive</td>
<td>Transmits supersonic pressure waves to surrounding pressure ( p_\infty ) ( p_\infty )</td>
<td>pInf value Fraction</td>
</tr>
<tr>
<td>supersonicFreeStream</td>
<td>Transmits oblique shocks to surroundings at ( p_\infty, T_\infty, \mathbf{U}<em>\infty ) ( p</em>\infty )</td>
<td>pInf, TInf, UInf</td>
</tr>
</tbody>
</table>

**Other types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Data to specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>slip</td>
<td>zeroGradient if ( \phi ) is a scalar; if ( \phi ) is a vector, normal component is fixedValue zero, tangential components are zeroGradient</td>
<td>—</td>
</tr>
<tr>
<td>partialSlip</td>
<td>Mixed zeroGradient/ slip condition depending on the valueFraction; = 0 for slip</td>
<td>valueFraction</td>
</tr>
</tbody>
</table>

Note: \( p \) is pressure, \( \mathbf{U} \) is velocity

Table 5.4: Derived patch field types.
5.3 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.

The mesh is generated from a dictionary file named blockMeshDict located in the system (or 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 5.4 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 5.3.3.

Each block has a local coordinate system \((x_1, x_2, x_3)\) that must be right-handed. A right-handed set of axes is defined such that to an observer looking down the \(Oz\) axis, with \(O\) nearest them, the arc from a point on the \(Ox\) axis to a point on the \(Oy\) axis is in a clockwise sense.

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.

5.3.1 Writing a blockMeshDict file

The blockMeshDict file is a dictionary using keywords described in Table 5.5. The convertToMeters keyword specifies a scaling factor by which all vertex coordinates in the mesh description are multiplied. For example,

\[
\text{convertToMeters} \quad 0.001;
\]

means that all coordinates are multiplied by 0.001, \(i.e.\) the values quoted in the blockMeshDict file are in mm.
5.3 Mesh generation with the blockMesh utility

5.3.1.1 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 5.4, the vertices are:

```plaintext
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
};
```

Table 5.5: Keywords used in blockMeshDict.
5.3.1.2 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.

Each entry for a curved edge begins with a keyword specifying the type of curve from those listed in Table 5.6.

<table>
<thead>
<tr>
<th>Keyword selection</th>
<th>Description</th>
<th>Additional entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>arc</td>
<td>Circular arc</td>
<td>Single interpolation point</td>
</tr>
<tr>
<td>spline</td>
<td>Spline curve</td>
<td>List of interpolation points</td>
</tr>
<tr>
<td>polyLine</td>
<td>Set of lines</td>
<td>List of interpolation points</td>
</tr>
<tr>
<td>BSpline</td>
<td>B-spline curve</td>
<td>List of interpolation points</td>
</tr>
<tr>
<td>line</td>
<td>Straight line</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 5.6: Edge types available in the blockMeshDict dictionary.

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 spline, polyLine and BSpline, 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 5.4 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)
);
```

5.3.1.3 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 5.3, 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
);
```
5.3 Mesh generation with the blockMesh utility

The definition of each block is as follows:

**Vertex numbering** The first entry is the shape identifier of the block, as defined in the `.OpenFOAM-3.0.1/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-140.

**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 $\delta_e$ along one edge of a block to the width of the start cell $\delta_s$ along that edge, as shown in Figure 5.5. 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 5.4 with the arrows representing the direction ‘from first cell... to last cell’ *e.g.* something like
  
  `edgeGrading (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.

\[ \frac{\delta_e}{\delta_s} \]

Expansion direction

Figure 5.5: Mesh grading along a block edge

5.3.1.4 Multi-grading of a block

Using a single expansion ratio to describe mesh grading permits only “one-way” grading within a mesh block. In some cases, it reduces complexity and effort to be able to control grading within separate divisions of a single block, rather than have to define several blocks with one grading per block. For example, to mesh a channel with two opposing walls and grade the mesh towards the walls requires three regions: two with grading to the wall with one in the middle without grading.

OpenFOAM v2.4+ includes multi-grading functionality that can divide a block in an given direction and apply different grading within each division. This multi-grading is specified by replacing any single value expansion ratio in the grading specification of the block, *e.g.* “1”, “2”, “3” in

---

OpenFOAM-3.0.1
We will present multi-grading for the following example:

- split the block into 3 divisions in the $y$-direction, representing 20%, 60% and 20% of the block length;
- include 30% of the total cells in the $y$-direction (300) in each divisions 1 and 3 and the remaining 40% in division 2;
- apply 1:4 expansion in divisions 1 and 3, and zero expansion in division 2.

We can specify this by replacing the $y$-direction expansion ratio “2” in the example above with the following:

```plaintext
blocks
(
    hex (0 1 2 3 4 5 6 7) (100 300 100)
    simpleGrading (1 2 3);
);
```

Both the fraction of the block and the fraction of the cells are normalized automatically. They can be specified as percentages, fractions, absolute lengths, etc. and do not need to sum to 100, 1, etc. The example above can be specified using percentages, e.g.

```plaintext
blocks
(
    hex (0 1 2 3 4 5 6 7) (100 300 100)
    simpleGrading
    ( 1 // x-direction expansion ratio
        (0.2 0.3 4) // 20% y-dir, 30% cells, expansion = 4
        (0.6 0.4 1) // 60% y-dir, 40% cells, expansion = 1
        (0.2 0.3 0.25) // 20% y-dir, 30% cells, expansion = 0.25 (1/4)
    )
    3 // z-direction expansion ratio
);
```
5.3 Mesh generation with the blockMesh utility

5.3.1.5 The boundary

The boundary of the mesh is given in a list named **boundary**. The boundary is broken into patches (regions), where each patch in the list has its name as the keyword, which is the choice of the user, although we recommend something that conveniently identifies the patch, *e.g.* inlet; the name is used as an identifier for setting boundary conditions in the field data files. The patch information is then contained in sub-dictionary with:

- **type**: the patch type, either a generic **patch** on which some boundary conditions are applied or a particular geometric condition, as listed in Table 5.2 and described in section 5.2.2;

- **faces**: a list of block faces that make up the patch and whose name is the choice of the user, although we recommend something that conveniently identifies the patch, *e.g.* inlet; the name is used as an identifier for setting boundary conditions in the field data files.

blockMesh collects faces from any boundary patch that is omitted from the **boundary** 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 5.4, 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:

```plaintext
boundary  // keyword
{
  inlet  // patch name
  {
    type patch;  // patch type for patch 0
    faces
      (
        (0 4 7 3) // block face in this patch
      );
    }  // end of 0th patch definition
  outlet // patch name
  {
    type patch;  // patch type for patch 1
    faces
      (
        (1 2 6 5)
      );
  }
}
```

OpenFOAM-3.0.1
walls
{
    type wall;
    faces
    {
        (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.

When specifying a cyclic patch in blockMesh, the user must specify the name of the related cyclic patch through the neighbourPatch keyword. For example, a pair of cyclic patches might be specified as follows:

left
{
    type cyclic;
    neighbourPatch right;
    faces ((0 4 7 3));
}
right
{
    type cyclic;
    neighbourPatch left;
    faces ((1 5 6 2));
}

5.3.2 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 formed from the same set of vertices as 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:

```plaintext
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 5.6, 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.

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 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.
5.3.3 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 \texttt{wedge} patch type described in section 5.2.2. The process is best illustrated by using a simplified version of our example block shown in Figure 5.7. 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:

\begin{verbatim}
hex (0 1 2 3 4 5 5 4)
\end{verbatim}

Figure 5.7: 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.

5.3.4 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:

    blockMesh -case <case>

The blockMeshDict file must exist in the system (or constant/polyMesh) directory.

5.4 Mesh generation with the snappyHexMesh utility

This section describes the mesh generation utility, snappyHexMesh, supplied with OpenFOAM. The snappyHexMesh utility generates 3-dimensional meshes containing hexahedra (hex) and split-hexahedra (split-hex) automatically from triangulated surface geometries, or tri-surfaces, in Stereolithography (STL) or Wavefront Object (OBJ) format. The mesh approximately conforms to the surface by iteratively refining a starting mesh and morphing the resulting split-hex mesh to the surface. An optional phase will shrink back the resulting mesh and insert cell layers. The specification of mesh refinement level is very flexible and the surface handling is robust with a pre-specified final mesh quality. It runs in parallel with a load balancing step every iteration.

![Figure 5.8: Schematic 2D meshing problem for snappyHexMesh](image)

5.4.1 The mesh generation process of snappyHexMesh

The process of generating a mesh using snappyHexMesh will be described using the schematic in Figure 5.8. The objective is to mesh a rectangular shaped region (shaded grey in the
Mesh generation and conversion

figure) surrounding an object described by a tri-surface, e.g. typical for an external aerodynamics simulation. Note that the schematic is 2-dimensional to make it easier to understand, even though the snappyHexMesh is a 3D meshing tool.

In order to run snappyHexMesh, the user requires the following:

- one or more tri-surface files located in a constant/triSurface sub-directory of the case directory;
- a background hex mesh which defines the extent of the computational domain and a base level mesh density; typically generated using blockMesh, discussed in section 5.4.2.
- a snappyHexMeshDict dictionary, with appropriate entries, located in the system sub-directory of the case.

The snappyHexMeshDict dictionary includes: switches at the top level that control the various stages of the meshing process; and, individual sub-directories for each process. The entries are listed in Table 5.7.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>castellatedMesh</td>
<td>Create the castellated mesh?</td>
<td>true</td>
</tr>
<tr>
<td>snap</td>
<td>Do the surface snapping stage?</td>
<td>true</td>
</tr>
<tr>
<td>doLayers</td>
<td>Add surface layers?</td>
<td>true</td>
</tr>
<tr>
<td>mergeTolerance</td>
<td>Merge tolerance as fraction of bounding box of initial mesh</td>
<td>1e-06</td>
</tr>
<tr>
<td>debug</td>
<td>Controls writing of intermediate meshes and screen printing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Write final mesh only</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>— Write intermediate meshes</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>— Write volScalarField with cellLevel for post-processing</td>
<td>2</td>
</tr>
<tr>
<td>geometry</td>
<td>Sub-dictionary of all surface geometry used</td>
<td></td>
</tr>
<tr>
<td>castellatedMeshControls</td>
<td>Sub-dictionary of controls for castellated mesh</td>
<td></td>
</tr>
<tr>
<td>snapControls</td>
<td>Sub-dictionary of controls for surface snapping</td>
<td></td>
</tr>
<tr>
<td>addLayersControls</td>
<td>Sub-dictionary of controls for layer addition</td>
<td></td>
</tr>
<tr>
<td>meshQualityControls</td>
<td>Sub-dictionary of controls for mesh quality</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.7: Keywords at the top level of snappyHexMeshDict.

All the geometry used by snappyHexMesh is specified in a geometry sub-directory in the snappyHexMeshDict dictionary. The geometry can be specified through a tri-surface or bounding geometry entities in OpenFOAM. An example is given below:

```plaintext
gamey

{  
  sphere.stl // STL filename
  |
    type triSurfaceMesh;
  |
  regions
  |
    { secondSolid // Named region in the STL file
  }
```

OpenFOAM-3.0.1
5.4 Mesh generation with the \textit{snappyHexMesh} utility

```c

name mySecondPatch; // User-defined patch name

box1x1x1 // User defined region name
{
    type searchableBox; // region defined by bounding box
    min (1.5 1 -0.5);
    max (3.5 2 0.5);
}

sphere2 // User defined region name
{
    type searchableSphere; // region defined by bounding sphere
    centre (1.5 1.5 1.5);
    radius 1.03;
}

```

5.4.2 Creating the background hex mesh

Before \textit{snappyHexMesh} is executed the user must create a background mesh of hexahedral cells that fills the entire region within by the external boundary as shown in Figure 5.9. This can be done simply using \textit{blockMesh}. The following criteria must be observed when creating the background mesh:

- the mesh must consist purely of hexes;

- the cell aspect ratio should be approximately 1, at least near surfaces at which the subsequent snapping procedure is applied, otherwise the convergence of the snapping procedure is slow, possibly to the point of failure;

- there must be at least one intersection of a cell edge with the tri-surface, \textit{i.e.} a mesh of one cell will not work.

![Figure 5.9: Initial mesh generation in \textit{snappyHexMesh} meshing process](image)

OpenFOAM-3.0.1
Mesh generation and conversion

5.4.3 Cell splitting at feature edges and surfaces

Cell splitting is performed according to the specification supplied by the user in the `castellatedMeshControls` sub-dictionary in the `snappyHexMeshDict`. The entries for `castellatedMeshControls` are presented in Table 5.8.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>locationInMesh</td>
<td>Location vector inside the region to be meshed</td>
<td>(5 0 0)</td>
</tr>
<tr>
<td>maxLocalCells</td>
<td>Max number of cells per processor during refinement</td>
<td>1e+06</td>
</tr>
<tr>
<td>maxGlobalCells</td>
<td>Overall cell limit during refinement (i.e. before removal)</td>
<td>2e+06</td>
</tr>
<tr>
<td>minRefinementCells</td>
<td>If ≥ number of cells to be refined, surface refinement stops</td>
<td>0</td>
</tr>
<tr>
<td>nCellsBetweenLevels</td>
<td>Number of buffer layers of cells between different levels of refinement</td>
<td>1</td>
</tr>
<tr>
<td>resolveFeatureAngle</td>
<td>Applies maximum level of refinement to cells that can see intersections whose angle exceeds this</td>
<td>30</td>
</tr>
<tr>
<td>features</td>
<td>List of features for refinement</td>
<td></td>
</tr>
<tr>
<td>refinementSurfaces</td>
<td>Dictionary of surfaces for refinement</td>
<td></td>
</tr>
<tr>
<td>refinementRegions</td>
<td>Dictionary of regions for refinement</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.8: Keywords in the `castellatedMeshControls` sub-dictionary of `snappyHexMeshDict`.

The splitting process begins with cells being selected according to specified edge features first within the domain as illustrated in Figure 5.10. The `features` list in the `castellatedMeshControls` sub-dictionary permits dictionary entries containing a name of an `edgeMesh` file and the `level` of refinement, *e.g.*:

```
features
(

OpenFOAM-3.0.1
```
5.4 Mesh generation with the snappyHexMesh utility

The edgeMesh containing the features can be extracted from the tri-surface file using surfaceFeatureExtract which specifies the tri-surface and controls such as included angle through a surfaceFeatureExtractDict configuration file, examples of which can be found in several tutorials and the $FOAM_UTILITIES/surface/surfaceFeatureExtract directory in the OpenFOAM installation. The utility is simply run by executing the following in a terminal

```
  surfaceFeatureExtract
```

Following feature refinement, cells are selected for splitting in the locality of specified surfaces as illustrated in Figure 5.11. The refinementSurfaces dictionary in castellatedMesh-Controls requires dictionary entries for each STL surface and a default level specification of the minimum and maximum refinement in the form (<min> <max>). The minimum level is applied generally across the surface; the maximum level is applied to cells that can see intersections that form an angle in excess of that specified by resolveFeatureAngle.

The refinement can optionally be overridden on one or more specific region of an STL surface. The region entries are collected in a regions sub-dictionary. The keyword for each region entry is the name of the region itself and the refinement level is contained within a further sub-dictionary. An example is given below:

```
refinementSurfaces
{
  sphere.stl
  {
    level (2 2); // default (min max) refinement for whole surface
    regions
      {
        secondSolid
          {
            level (3 3); // optional refinement for secondSolid region
          }
      }
  }
}
```

![Figure 5.11: Cell splitting by surface in snappyHexMesh meshing process](image-url)
5.4.4 Cell removal

Once the feature and surface splitting is complete a process of cell removal begins. Cell removal requires one or more regions enclosed entirely by a bounding surface within the domain. The region in which cells are retained are simply identified by a location vector within that region, specified by the `locationInMesh` keyword in `castellatedMeshControls`. Cells are retained if, approximately speaking, 50% or more of their volume lies within the region. The remaining cells are removed accordingly as illustrated in Figure 5.12.

![Cell removal in snappyHexMesh meshing process](image)

**Figure 5.12**: Cell removal in `snappyHexMesh` meshing process

5.4.5 Cell splitting in specified regions

Those cells that lie within one or more specified volume regions can be further split as illustrated in Figure 5.13 by a rectangular region shown by dark shading. The `refinementRegions` sub-dictionary in `castellatedMeshControls` contains entries for refinement of the volume regions specified in the `geometry` sub-dictionary. A refinement mode is applied to each region which can be:

- **inside** refines inside the volume region;
- **outside** refines outside the volume region
- **distance** refines according to distance to the surface; and can accommodate different levels at multiple distances with the `levels` keyword.

For the `refinementRegions`, the refinement level is specified by the `levels` list of entries with the format `<distance> <level>`. In the case of `inside` and `outside` refinement, the `<distance>` is not required so is ignored (but it must be specified). Examples are shown below:

```plaintext
refinementRegions
{
  box1x1x1
  {
    mode inside;
    levels ((1.0 4)); // refinement level 4 (1.0 entry ignored)
  }
}
```

OpenFOAM-3.0.1
5.4 Mesh generation with the \texttt{snappyHexMesh} utility

```c
sphere.stl
{
  mode distance; // refinement level 5 within 1.0 m
  levels ((1.0 5) (2.0 3)); // levels must be ordered nearest first
}
```

5.4.6 Snapping to surfaces

The next stage of the meshing process involves moving cell vertex points onto surface geometry to remove the jagged castellated surface from the mesh. The process is:

1. displace the vertices in the castellated boundary onto the STL surface;
2. solve for relaxation of the internal mesh with the latest displaced boundary vertices;
3. find the vertices that cause mesh quality parameters to be violated;
4. reduce the displacement of those vertices from their initial value (at 1) and repeat from 2 until mesh quality is satisfied.

The method uses the settings in the \texttt{snapControls} sub-dictionary in \texttt{snappyHexMeshDict}, listed in Table 5.9. An example is illustrated in the schematic in Figure 5.14 (albeit with mesh motion that looks slightly unrealistic).

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSmoothPatch</td>
<td>Number of patch smoothing iterations before finding correspondence to surface</td>
<td>3</td>
</tr>
<tr>
<td>tolerance</td>
<td>Ratio of distance for points to be attracted by surface feature point or edge, to local maximum edge length</td>
<td>4.0</td>
</tr>
<tr>
<td>nSolveIter</td>
<td>Number of mesh displacement relaxation iterations</td>
<td>30</td>
</tr>
<tr>
<td>nRelaxIter</td>
<td>Maximum number of snapping relaxation iterations</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5.9: Keywords in the \texttt{snapControls} dictionary of \texttt{snappyHexMeshDict}.

5.4.7 Mesh layers

The mesh output from the snapping stage may be suitable for the purpose, although it can produce some irregular cells along boundary surfaces. There is an optional stage of the meshing process which introduces additional layers of hexahedral cells aligned to the boundary surface as illustrated by the dark shaded cells in Figure 5.15.

The process of mesh layer addition involves shrinking the existing mesh from the boundary and inserting layers of cells, broadly as follows:

1. the mesh is projected back from the surface by a specified thickness in the direction normal to the surface;
2. solve for relaxation of the internal mesh with the latest projected boundary vertices;
Figure 5.13: Cell splitting by region in *snappyHexMesh* meshing process

Figure 5.14: Surface snapping in *snappyHexMesh* meshing process

Figure 5.15: Layer addition in *snappyHexMesh* meshing process
3. check if validation criteria are satisfied otherwise reduce the projected thickness and return to 2; if validation cannot be satisfied for any thickness, do not insert layers;

4. if the validation criteria can be satisfied, insert mesh layers;

5. the mesh is checked again; if the checks fail, layers are removed and we return to 2.

The layer addition procedure uses the settings in the `addLayersControls` sub-dictionary in `snappyHexMeshDict`; entries are listed in Table 5.10. The `layers` sub-dictionary contains

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>layers</td>
<td>Dictionary of layers</td>
<td></td>
</tr>
<tr>
<td>relativeSizes</td>
<td>Are layer thicknesses relative to undistorted cell size outside layer or absolute?</td>
<td>true/false</td>
</tr>
<tr>
<td>expansionRatio</td>
<td>Expansion factor for layer mesh</td>
<td>1.0</td>
</tr>
<tr>
<td>finalLayerThickness</td>
<td>Thickness of layer furthest from the wall, either relative or absolute according to the <code>relativeSizes</code> entry</td>
<td>0.3</td>
</tr>
<tr>
<td>minThickness</td>
<td>Minimum thickness of cell layer, either relative or absolute (as above)</td>
<td>0.25</td>
</tr>
<tr>
<td>nGrow</td>
<td>Number of layers of connected faces that are not grown if points get not extruded; helps convergence of layer addition close to features</td>
<td>1</td>
</tr>
<tr>
<td>featureAngle</td>
<td>Angle above which surface is not extruded</td>
<td>60</td>
</tr>
<tr>
<td>nRelaxIter</td>
<td>Maximum number of snapping relaxation iterations</td>
<td>5</td>
</tr>
<tr>
<td>nSmoothSurfaceNormals</td>
<td>Number of smoothing iterations of surface normals</td>
<td>1</td>
</tr>
<tr>
<td>nSmoothNormals</td>
<td>Number of smoothing iterations of interior mesh movement direction</td>
<td>3</td>
</tr>
<tr>
<td>nSmoothThickness</td>
<td>Smooth layer thickness over surface patches</td>
<td>10</td>
</tr>
<tr>
<td>maxFaceThicknessRatio</td>
<td>Stop layer growth on highly warped cells</td>
<td>0.5</td>
</tr>
<tr>
<td>maxThicknessToMedialRatio</td>
<td>Reduce layer growth where ratio thickness to medial distance is large</td>
<td>0.3</td>
</tr>
<tr>
<td>minMedianAxisAngle</td>
<td>Angle used to pick up medial axis points</td>
<td>130</td>
</tr>
<tr>
<td>nBufferCellsNoExtrude</td>
<td>Create buffer region for new layer terminations</td>
<td>0</td>
</tr>
<tr>
<td>nLayerIter</td>
<td>Overall max number of layer addition iterations</td>
<td>50</td>
</tr>
<tr>
<td>nRelaxedIter</td>
<td>Max number of iterations after which the controls in the <code>relaxed</code> sub dictionary of <code>meshQuality</code> are used</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 5.10: Keywords in the `addLayersControls` sub-dictionary of `snappyHexMeshDict`.

entries for each `patch` on which the layers are to be applied and the number of surface layers required. The patch name is used because the layers addition relates to the existing mesh, not the surface geometry; hence applied to a patch, not a surface region. An example `layers` entry is as follows:
layers
{
    sphere.stl_firstSolid
    {
        nSurfaceLayers 1;
    }
    maxY
    {
        nSurfaceLayers 1;
    }
}

Keyword | Description | Example
-------|-------------|--------
maxNonOrtho | Maximum non-orthogonality allowed; 180 disables | 65
maxBoundarySkewness | Max boundary face skewness allowed; <0 disables | 20
maxInternalSkewness | Max internal face skewness allowed; <0 disables | 4
maxConcave | Max concaveness allowed; 180 disables | 80
minFlatness | Ratio of minimum projected area to actual area; -1 disables | 0.5
minVol | Minimum pyramid volume; large negative number, e.g. -1e30 disables | 1e-13
minArea | Minimum face area; <0 disables | -1
minTwist | Minimum face twist; <-1 disables | 0.05
minDeterminant | Minimum normalised cell determinant; 1 = hex; ≤ 0 illegal cell | 0.001
minFaceWeight | 0→0.5 | 0.05
minVolRatio | 0→1.0 | 0.01
minTriangleTwist | >0 for Fluent compatibility | -1
nSmoothScale | Number of error distribution iterations | 4
time | Amount to scale back displacement at error points | 0.75
relaxed | Sub-dictionary that can include modified values for the above keyword entries to be used when nRelaxedIter is exceeded in the layer addition process | relaxed

Table 5.11: Keywords in the meshQualityControls sub-dictionary of snappyHexMeshDict.

5.4.8 Mesh quality controls

The mesh quality is controlled by the entries in the meshQualityControls sub-dictionary in snappyHexMeshDict; entries are listed in Table 5.11.

5.5 Mesh conversion

The user can generate meshes using other packages and convert them into the format that OpenFOAM uses. There are numerous mesh conversion utilities listed in Table 3.6. Some of
the more popular mesh converters are listed below and their use is presented in this section.

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;

cfx4ToFoam reads a CFX mesh written in .geo format;

5.5.1 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 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 by creating the necessary directories/files: the case directory containing a controlDict file in a system subdirectory. Then at a command prompt the user should execute:

    fluentMeshToFoam <meshFile>

where <meshFile> is the name of the .msh file, including the full or relative path.
5.5.2 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 5.1. 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.

5.5.2.1 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.

5.5.2.2 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:

\[ \text{CDEL CSET} \]

Similarly, vertices will need to be discarded as well:

\[ \text{CSET NEWS FLUID} \\ \text{VSET NEWS CSET} \\ \text{VSET INVE} \]

Before discarding these unwanted vertices, the unwanted boundary faces have to be collected before purging:

\[ \text{CSET NEWS FLUID} \\ \text{VSET NEWS CSET} \\ \text{BSET NEWS VSET ALL} \\ \text{BSET INVE} \]

If the BSET is not empty, the unwanted boundary faces can be deleted using:

\[ \text{BDEL BSET} \]

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.

### 5.5.2.3 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.

5.5.2.4 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.

OpenFOAM-3.0.1
5.5.2.5 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 by creating and editing OpenFOAM dictionary files.

The procedure of converting the PROSTAR files is first to create a new OpenFOAM case by creating the necessary directories. 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.

5.5.2.6 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
```

### 5.5.2.7 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 <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 by editing the `boundary` file by hand.

### 5.5.3 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, the user should execute:

```
gambitToFoam <meshFile>
```

where `<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.

### 5.5.4 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, the user should execute:

```
ideasToFoam <meshFile>
```

where `<meshFile>` is the name of the `.ans` file, including the full or relative path.
5.5.5 cfx4ToFoam

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, the user should execute:

```
cfx4ToFoam <meshFile>
```

where <meshFile> is the name of the .geo file, including the full or relative path.

5.6 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.

5.6.1 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 <source dir> -consistent
```
5.6.2 Mapping inconsistent fields

When the fields are not consistent, as shown in Figure 5.16, 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 5.16. 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:

```
17    patchMap ( lid movingWall );
18    cuttingPatches ( fixedWalls );
21
22    // ************************************************************************* //
```

Figure 5.16: Mapping inconsistent fields
5.6 Mapping fields between different geometries

mapFields <source dir>

5.6.3 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 6
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 6.1.

Other methods of post-processing using third party products are offered, including EnSight, Fieldview and the post-processing supplied with Fluent.

6.1 paraFoam

The main post-processing tool provided with OpenFOAM is a reader module to run with ParaView, an open-source, visualization application. The module is compiled into 2 libraries, PV4FoamReader and vtkPV4Foam using version 4.4.0 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.

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 foam-ToVTK 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.

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.

6.1.1 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 either by the single command from within the case directory or with the -case option with the case path as an argument, e.g.:

    paraFoam -case <caseDir>

ParaView is launched and opens the window shown in Figure 6.1. The case is controlled from the left panel, which contains the following:
Pipeline Browser lists the modules opened in ParaView, where the selected modules are highlighted in blue and the graphics for the given module can be enabled/disabled by clicking the eye button alongside;

Properties panel contains the input selections for the case, such as times, regions and fields; it includes the Display panel that controls the visual representation of the selected module, e.g. colours;

Other panels can be selected from the View menu, including the Information panel which 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 green Apply button in the Properties panel. The additional buttons are: the Reset button which can be used to reset the GUI if necessary; and, the Delete button that will delete the active module.
## 6.1.2 The Parameters panel

The Properties window for the case module includes the Parameters panel that contains the settings for mesh, fields and global controls. The controls are described in Figure 6.2. The user can select mesh and field data which is loaded for all time directories into ParaView. The buttons in the Current Time Controls and VCR Controls toolbars then select the time data to be displayed, as shown is section 6.1.4.

As with any operation in paraFoam, the user must click Apply after making any changes to any selections. The Apply 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.

If new data is written to time directories while the user is running ParaView, the user must load the additional time directories by checking the Refresh Times button. Where there are occasions when the case data changes on file and ParaView needs to load the changes, the user can also check the Update GUI button in the Parameters panel and apply the changes.
Outline, surface, wireframe or points

Colour geometry/entity by...
Set colour map range/appearance

Change image opacity
e.g. to make translucent

Data interpolation method

Geometry manipulation tools

Figure 6.3: The Display panel

6.1.3 The Display panel

The Properties window contains the Display panel that includes the settings for visualising the data for a given case module. The following points are particularly important:

- the data range may not be automatically updated to the max/min limits of a field, so the user should take care to select Rescale at appropriate intervals, in particular after loading the initial case module;

- clicking the Edit Color Map button, brings up a window in which there are two panels:
6.1 paraFoam

1. The Color Scale panel in which the colours within the scale can be chosen. The standard blue to red colour scale for CFD can be selected by clicking Choose Preset and selecting Blue to Red Rainbow HSV.

2. The Color Legend panel has a toggle switch for a colour bar legend and contains settings for the layout of the legend, e.g. font.

- the underlying mesh can be represented by selecting Wireframe in the Representation menu of the Style panel;
- the geometry, e.g. a mesh (if Wireframe is selected), can be visualised as a single colour by selecting Solid Color from the Color By menu and specifying the colour in the Set Ambient Color window;
- the image can be made translucent by editing the value in the Opacity text box (1 = solid, 0 = invisible) in the Style panel.

6.1.4 The button toolbars

ParaView duplicates functionality from pull-down menus at the top of the main window and the major panels, within the toolbars below the main pull-down menus. The displayed toolbars can be selected from Toolbars in the main View menu. The default layout with all toolbars is shown in Figure 6.4 with each toolbar labelled. The function of many of the buttons is clear from their icon and, with tooltips enabled in the Help menu, the user is given a concise description of the function of any button.

![Figure 6.4: Toolbars in ParaView](image)

6.1.5 Manipulating the view

This section describes operations for setting and manipulating the view of objects in paraFoam.

6.1.5.1 View settings

The View Settings are available in the Render View panel below the Display panel in the Properties window. Settings that are generally important only appear when the user checks the gearwheel button at the top of the Properties window, next to the search bar. These advanced properties include setting the background colour, where white is often a preferred choice for creating images for printed and website material.
The Lights button opens detailed lighting controls within the Light Kit panel. A separate Headlight panel controls the direct lighting of the image. Checking the Headlight button with white light colour of strength 1 seems to help produce images with strong bright colours, e.g. with an isosurface.

The Camera Parallel Projection is the usual choice for CFD, especially for 2D cases, and so should generally be checked. Other settings include Cube Axes which displays axes on the selected object to show its orientation and geometric dimensions.

6.1.5.2 General settings

The general Settings are selected from the Edit menu, which opens a general Options window with General, Colors, Animations, Charts and Render View menu items.

The General panel controls some default behaviour of ParaView. In particular, there is an Auto Accept button that enables ParaView to accept changes automatically without clicking the green Apply button in the Properties window. For larger cases, this option is generally not recommended: the user does not generally want the image to be re-rendered between each of a number of changes he/she selects, but be able to apply a number of changes to be re-rendered in their entirety once.

The Render View panel contains 3 sub-items: General, Camera and Server. The General panel includes 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 Camera panel includes control settings for 3D and 2D movements. This presents the user with a map of rotation, translate and zoom controls using the mouse in combination with Shift- and Control-keys. The map can be edited to suit by the user.

6.1.6 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 Properties panel for contours contains an Isosurfaces list that the user can edit, most conveniently by the New Range window. The chosen scalar field is selected from a pull down menu.

6.1.6.1 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 Slice filter to create the cutting plane, on which the contours can be plotted. The Slice filter allows the user to specify a cutting Plane, Box or Sphere in the Slice Type 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.

6.1.7 Vector plots

Vector plots are created using the Glyph filter. The filter reads the field selected in Vectors and offers a range of Glyph Types for which the Arrow provides a clear vector plot images.
Each glyph has a selection of graphical controls in a panel which the user can manipulate to best effect.

The remainder of the Properties panel contains mainly the Scale Mode menu for the glyphs. The most common options are Scale Mode are: Vector, where the glyph length is proportional to the vector magnitude; and, Off where each glyph is the same length. The Set Scale Factor parameter controls the base length of the glyphs.

### 6.1.7.1 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.

### 6.1.8 Streamlines

Streamlines are created by first creating tracer lines using the Stream Tracer filter. The tracer Seed panel specifies a distribution of tracer points over a Line Source or Point Cloud. The user can view the tracer source, e.g. the line, but it is displayed in white, so they may need to change the background colour in order to see it.

The distance the tracer travels and the length of steps the tracer takes are specified in the text boxes in the main Stream Tracer panel. 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 the Tracer module to produce high quality images. The tubes follow each tracer line and are not strictly cylindrical 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.

### 6.1.9 Image output

The simplest way to output an image to file from ParaView is to select Save Screenshot from the File menu. On selection, a window appears in which the user can select the resolution for the image to save. There is a button that, when clicked, locks the aspect ratio, so if the user changes the resolution in one direction, the resolution is adjusted in the other direction automatically. After selecting the pixel resolution, the image can be saved.

To achieve high quality output, the user might try setting the pixel resolution to 1000 or more in the x-direction so that when the image is scaled to a typical size of a figure in an A4 or US letter document, perhaps in a PDF document, the resolution is sharp.

### 6.1.10 Animation output

To create an animation, the user should first select Save Animation from the File menu. A dialogue window appears in which the user can specify a number of things including the image resolution. The user should specify the resolution as required. The other noteworthy setting is number of frames per timestep. While this would intuitively be set to 1, it can be set to a larger number in order to introduce more frames into the animation artificially.
This technique can be particularly useful to produce a slower animation because some movie players have limited speed control, particularly over mpeg movies.

On clicking the Save Animation button, another window appears in which the user specifies a file name root and file format for a set of images. On clicking OK, the set of files will be saved according to the naming convention “<fileRoot>_<imageNo>.<fileExt>”, e.g. the third image of a series with the file root “animation”, saved in jpg format would be named “animation_0002.jpg” (<imageNo> starts at 0000).

Once the set of images are saved the user can convert them into a movie using their software of choice. One option is to use the built in foamCreateVideo script from the command line whose usage is shown with the -help option.

### 6.2 Function Objects

OpenFOAM provides functionality that can be executed during a simulation by the user at run-time through a configuration in the controlDict file. For example, a user may wish to run a steady-state, incompressible, turbulent flow simulation of aerodynamics around a vehicle and from that simulation they wish to calculate the drag coefficient. While the simulation is performed by the simpleFoam solver, the additional force calculation for drag coefficient is included in a tool, called a function object, that can be requested by the user to be executed at certain times during the simulation.

<table>
<thead>
<tr>
<th>Function object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cellSource</td>
<td>performs operations on cell values, e.g. sums, averages and integrations</td>
</tr>
<tr>
<td>faceSource</td>
<td>performs operations on face values, e.g. sums, averages and integrations</td>
</tr>
<tr>
<td>fieldMinMax</td>
<td>writes min/max values of fields</td>
</tr>
<tr>
<td>fieldValue</td>
<td>averaging/integration across sets of faces/cells, e.g. for flux across a plane</td>
</tr>
<tr>
<td>fieldValueDelta</td>
<td>Provides differencing between two fieldValue function objects, e.g. to calculate pressure drop</td>
</tr>
<tr>
<td>forces</td>
<td>calculates pressure/viscous forces and moments</td>
</tr>
<tr>
<td>forceCoeffs</td>
<td>calculates lift, drag and moment coefficients</td>
</tr>
<tr>
<td>regionSizeDistrib-</td>
<td>creates a size distribution via interrogating a continuous phase fraction</td>
</tr>
<tr>
<td>ution</td>
<td></td>
</tr>
<tr>
<td>sampledSet</td>
<td>data sampling along lines, e.g. for graph plotting</td>
</tr>
<tr>
<td>probes</td>
<td>data probing at point locations</td>
</tr>
<tr>
<td>residuals</td>
<td>writes initial residuals for selected fields</td>
</tr>
</tbody>
</table>

Table 6.1: Function objects writing time-value data for monitoring/plotting

A large number of function objects exist in OpenFOAM that perform mainly a range of post-processing calculations but also some job control activities. Function objects can be broken down into the following categories.

- Table 6.1: write out tabulated data, typically containing time-values, usually at regular time intervals, for plotting graphs and/or monitoring, e.g. force coefficients.
### 6.2 Function Objects

<table>
<thead>
<tr>
<th>Function object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fieldAverage</strong></td>
<td>temporal averaging of fields</td>
</tr>
<tr>
<td><strong>writeRegistered-Object</strong></td>
<td>writes fields that are not scheduled to be written</td>
</tr>
<tr>
<td><strong>fieldCoordinate-SystemTransform</strong></td>
<td>transforms fields between global to local co-ordinate system</td>
</tr>
<tr>
<td><strong>turbulenceFields</strong></td>
<td>stores turbulence fields on the mesh database</td>
</tr>
<tr>
<td><strong>calcFvcDiv</strong></td>
<td>calculates the divergence of a field</td>
</tr>
<tr>
<td><strong>calcFvcGrad</strong></td>
<td>calculates the gradient of a field</td>
</tr>
<tr>
<td><strong>calcMag</strong></td>
<td>calculates the magnitude of a field</td>
</tr>
<tr>
<td><strong>CourantNo</strong></td>
<td>outputs the Courant number field</td>
</tr>
<tr>
<td><strong>Lambda2</strong></td>
<td>outputs Lambda2</td>
</tr>
<tr>
<td><strong>Peclet</strong></td>
<td>outputs the Peclet number field</td>
</tr>
<tr>
<td><strong>pressureTools</strong></td>
<td>calculate pressure in different forms, static, total, etc.</td>
</tr>
<tr>
<td><strong>Q</strong></td>
<td>second invariant of the velocity gradient</td>
</tr>
<tr>
<td><strong>vorticity</strong></td>
<td>calculates vorticity field</td>
</tr>
<tr>
<td><strong>processorField</strong></td>
<td>writes a field of the local processor ID</td>
</tr>
<tr>
<td><strong>partialWrite</strong></td>
<td>allows registered objects to be written at specified times</td>
</tr>
<tr>
<td><strong>readFields</strong></td>
<td>reads fields from the time directories and adds to database</td>
</tr>
<tr>
<td><strong>blendingFactor</strong></td>
<td>outputs the blending factor used by the convection schemes</td>
</tr>
<tr>
<td><strong>DESModelRegions</strong></td>
<td>writes out an indicator field for DES turbulence</td>
</tr>
</tbody>
</table>

Table 6.2: Function objects for writing/reading fields, typically writing to time directories

<table>
<thead>
<tr>
<th>Function object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nearWallFields</strong></td>
<td>writes fields in cells adjacent to patches</td>
</tr>
<tr>
<td><strong>wallShearStress</strong></td>
<td>evaluates and outputs the shear stress at wall patches</td>
</tr>
<tr>
<td><strong>yPlusLES</strong></td>
<td>outputs turbulence y+ for LES models</td>
</tr>
<tr>
<td><strong>yPlusRAS</strong></td>
<td>outputs turbulence y+ for RAS models</td>
</tr>
</tbody>
</table>

Table 6.3: Function objects for near wall fields

- Table 6.2: write out field data, usually at the normal time interval for writing fields into time directories.
- Table 6.3: write out field data on boundary patches, e.g. wall shear stress, usually at the time interval for writing fields into time directories.
- Table 6.4: write out image files, e.g. iso-surface for visualization.
- Table 6.5: function objects that manage job control, supplementary simulation, etc.

### 6.2.1 Using function objects

Function objects are specified in the `controlDict` file of a case through a dictionary named `functions`. One or more function objects can be specified, each within its own sub-dictionary. An example of specification of 2 function objects is shown below:
Function object | Description
--- | ---
**streamline** | streamline data from sampled fields
**surfaces** | iso-surfaces, cutting planes, patch surfaces, with field data
**wallBoundedStreamline** | streamlines constrained to a boundary patch

Table 6.4: Function objects for creating post-processing images

Function object | Description
--- | ---
**timeActivatedFile-Update** | modifies case settings at specified times in a simulation
**abortCalculation** | aborts simulation when named file appears in the case directory
**removeRegistered-Object** | removes specified registered objects in the database
**setTimeStepFunctionObject** | enables manual over-ride of the time step
**codedFunctionObject** | function object coded with **#codeStream**
**cloudInfo** | outputs Lagrangian cloud information
**scalarTransport** | solves a passive scalar transport equation
**systemCall** | makes any call to the system, e.g. sends an email

Table 6.5: Miscellaneous function objects

```plaintext
functions
{
  pressureProbes
  {
    type probes;
    functionObjectLibs ("libsampling.so");
    outputControl timeStep;
    outputInterval 1;
    probeLocations
    {
      ( 1 0 0 )
      ( 2 0 0 )
    );
    fields
    {
      P
    };
  }

  meanVelocity
  {
    type fieldAverage;
    functionObjectLibs ( "libfieldFunctionObjects.so" );
  }
}
```

OpenFOAM-3.0.1
outputControl outputTime;
fields
{
    U
    {
        mean on;
        prime2Mean off;
        base time;
    }
};
}

For each function object, there are some mandatory keyword entries.

name   Every function object requires a unique name, here pressureProbes and meanVelocity are used. The names can be used for naming output files and/or directories.

type   The type of function object, e.g. probes.

functionObjectLibs A list of additional libraries that may need to be dynamically linked at run-time to access the relevant functionality. For example the forceCoeffs function object is compiled into the libforces.so library, so force coefficients cannot be calculated without linking that library.

outputControl Specifies when data should be calculated and output. Options are: timeStep, when data is output each writeInterval time steps; and, outputTime when data is written at scheduled times, i.e. when fields are written to time directories.

The remaining entries in the example above are specific to the particular function object. For example probeLocations describes the locations where pressure values are probed. For any other function object, how can the user find out the specific keyword entries required? One option is to access the code C++ documentation at either http://openfoam.org/-docs/cpp or http://openfoam.github.io/Documentation-dev/html and click the post-processing link. This takes the user to a set of lists of function objects, the class description of each function object providing documentation on its use.

Alternatively the user can scan the cases in $FOAM_TUTORIALS directory for examples of function objects in use. The find and grep command can help to locate relevant examples, e.g.

    find $FOAM_TUTORIALS -name controlDict | xargs grep -l functions

6.2.2 Packaged function objects

From OpenFOAM v2.4, commonly used function objects are “packaged” in the distribution in $FOAM_ETC/caseDicts/postProcessing. The tools range from quite generic, e.g. minMax to monitor min and max values of a field, to some more application-oriented, e.g. flowRate to monitor flow rate.
The configuration of function objects includes both required input data and control parameters for the function object. This creates a lot of input that can be confusing to users. The packaged function objects separate the user input from control parameters. Control parameters are pre-configured in files with .cfg extension. For each tool, required user input is all in one file, for the users to copy into their case and set accordingly.

The tools can be used as follows, using an example of monitoring flow rate at an outlet patch named outlet.

1. Locate the flowRatePatch tool in the flowRate directory:
   
   
   $FOAM_ETC/caseDicts/postProcessing/flowRate

2. Copy the flowRatePatch file into the case system directory (not flowRatePatch.cfg)

3. Edit system/flowRatePatch to set the patch name, replacing “patch <patchName>;” with “patch outlet;”

4. Activate the function object by including the flowRatePatch file in functions sub-dictionary in the case controlDict file, e.g.

   functions
   {
     #include "flowRatePatch"
     ... other function objects here ...
   }

Current packaged function objects are found in the following directories:

- **fields** calculate specific fields, e.g. \( Q \)
- **flowRate**: tools to calculate flow rate
- **forces**: forces and force coefficients for incompressible/compressible flows
- **graphs**: simple sampling for graph plotting, e.g. singleGraph
- **minMax**: range of minimum and maximum field monitoring, e.g. cellMax
- **numerical**: outputs information relating to numerics, e.g. residuals
- **pressure**: calculates different forms of pressure, pressure drop, etc
- **probes**: options for probing data
- **scalarTransport**: for plugin scalar transport calculations
- **visualization**: post-processing VTK files for cutting planes, streamlines, etc.
- **faceSource**: configuration for some of the tools above
6.3 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 executed in the usual manner. 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:

```cpp
/*----------------------------- C++ -----------------------------*/

FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    location "system";
    object foamDataToFluentDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

p 1;
U 2;
T 3;
h 4;
k 5;
epsilon 6;
alpha1 150;

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

The dictionary contains entries of the form

```cpp
<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 6.6. 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. The list of default entries described in Table 6.6. The user can run foamDataToFluent like any utility.

To view the results using Fluent, go to the fluentInterface subdirectory of the case directory and start a 3 dimensional version of Fluent with

```cpp
fluent 3d
```
U-182

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 6.6: Fluent unit numbers for post-processing.

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.

### 6.4 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. `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.
6.5 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.

6.5.1 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. 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.

6.5.2 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.

6.5.2.1 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 $WM_PROJECT_DIR/etc/apps/-ensightFoam directory. The environment variables associated with EnSight are prefixed by $CEI_ or $ENSIGHT7_ and listed in Table 6.7. 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 6.7: Environment variable settings for EnSight.

6.5.2.2 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 for the case. 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.

6.6 Sampling data

OpenFOAM provides the sample utility to sample field data, either through a 1D line for plotting on graphs or a 2D plane for displaying as isosurface images. 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 by copying an example sampleDict from the sample source code directory at $FOAM\_UTILITIES/postProcessing/sampling/sample. The
plateHole tutorial case in the $FOAM_TUTORIALS/solidDisplacementFoam$ directory also contains an example for 1D line sampling:

```cpp
interpolationScheme cellPoint;
setFormat raw;

sets
{
  leftPatch
  {
    type uniform;
    axis y;
    start (0 0.5 0.25);
    end (0 2 0.25);
    nPoints 100;
  }
};

fields (sigmaEq);

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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interpolationScheme</td>
<td>cell</td>
<td>Cell-centre value assumed constant over cell</td>
</tr>
<tr>
<td>Scheme</td>
<td>cellPoint</td>
<td>Linear weighted interpolation using cell values</td>
</tr>
<tr>
<td></td>
<td>cellPointFace</td>
<td>Mixed linear weighted / cell-face interpolation</td>
</tr>
<tr>
<td>setFormat</td>
<td>raw</td>
<td>Raw ASCII data in columns</td>
</tr>
<tr>
<td></td>
<td>gnuplot</td>
<td>Data in gnuplot format</td>
</tr>
<tr>
<td></td>
<td>xmgr</td>
<td>Data in Grace/xmgr format</td>
</tr>
<tr>
<td></td>
<td>jplot</td>
<td>Data in jPlot format</td>
</tr>
<tr>
<td>surfaceFormat</td>
<td>null</td>
<td>Suppresses output</td>
</tr>
<tr>
<td>foamFile</td>
<td>points, faces, values</td>
<td>file</td>
</tr>
<tr>
<td>dx</td>
<td>DX scalar or vector format</td>
<td></td>
</tr>
<tr>
<td>vtk</td>
<td>VTK ASCII format</td>
<td></td>
</tr>
<tr>
<td>raw</td>
<td>xyz values for use with e.g. gnuplotsplot</td>
<td></td>
</tr>
<tr>
<td>stl</td>
<td>ASCII STL; just surface, no values</td>
<td></td>
</tr>
</tbody>
</table>

| fields               | List of fields to be sampled, e.g. for velocity U:                         |
|                      | U | Writes all components of U                                                 |

| sets                 | List of 1D sets subdictionaries — see Table 6.9                             |
| surfaces             | List of 2D surfaces subdictionaries — see Table 6.10 and Table 6.11        |

Table 6.8: keyword entries for sampleDict.

The dictionary contains the following entries:

- **interpolationScheme** the scheme of data interpolation;
- **sets** the locations within the domain that the fields are line-sampled (1D);
- **surfaces** the locations within the domain that the fields are surface-sampled (2D);
- **setFormat** the format of line data output;
**surfaceFormat** the format of surface data output;

**fields** the fields to be 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 **setFormat** entry for line sampling includes a raw data format and formats for **gnuplot**, Grace/xmgr and jPlot graph drawing packages. The data are written into a **sets** 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 sets directory is deleted when sample is run.*

The **surfaceFormat** entry for surface sampling includes a raw data format and formats for **gnuplot**, Grace/xmgr and jPlot graph drawing packages. The data are written into a **surfaces** directory within the case directory. The directory is split into time directories and files are written much as with line sampling.

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 **sets** list contains sub-dictionaries of locations where the data is to be sampled. The sub-dictionary is named according to the name of the set and contains a set of entries, also listed in Table 6.9, 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 **type**; and, means of specifying the length ordinate on a graph by the **axis** keyword.

The **surfaces** list contains sub-dictionaries of locations where the data is to be sampled. The sub-dictionary is named according to the name of the surface and contains a set of entries beginning with the **type**: either a **plane**, defined by point and normal direction, with additional sub-dictionary entries specified in Table 6.10; or, a **patch**, coinciding with an existing boundary patch, with additional sub-dictionary entries a specified in Table 6.11.

### 6.7 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**
### Monitoring and managing jobs

**Required entries**

<table>
<thead>
<tr>
<th>Sampling type</th>
<th>Sample locations</th>
<th>name</th>
<th>axis</th>
<th>start</th>
<th>end</th>
<th>nPoints</th>
<th>points</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform</td>
<td>Uniformly distributed points on a line</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>face</td>
<td>Intersection of specified line and cell faces</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>midPoint</td>
<td>Midpoint between line-face intersections</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>midPointAndFace</td>
<td>Combination of midPoint and face</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>curve</td>
<td>Specified points, tracked along a curve</td>
<td></td>
<td>.</td>
<td>.</td>
<td></td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>cloud</td>
<td>Specified points</td>
<td></td>
<td>.</td>
<td></td>
<td>.</td>
<td></td>
<td>.</td>
</tr>
</tbody>
</table>

**Entries**

<table>
<thead>
<tr>
<th>Entries</th>
<th>Description</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Sampling type</td>
<td>see list above</td>
</tr>
<tr>
<td>axis</td>
<td>Output of sample location</td>
<td>x, y, z ordinates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xyz coordinates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>distance</td>
</tr>
<tr>
<td>start</td>
<td>Start point of sample line</td>
<td>e.g. (0.0 0.0 0.0)</td>
</tr>
<tr>
<td>end</td>
<td>End point of sample line</td>
<td>e.g. (0.0 2.0 0.0)</td>
</tr>
<tr>
<td>nPoints</td>
<td>Number of sampling points</td>
<td>e.g. 200</td>
</tr>
<tr>
<td>points</td>
<td>List of sampling points</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.9: Entries within sets sub-dictionaries.

**Keyword**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>basePoint</td>
<td>Point on plane</td>
<td>e.g. (0 0 0)</td>
</tr>
<tr>
<td>normalVector</td>
<td>Normal vector to plane</td>
<td>e.g. (1 0 0)</td>
</tr>
<tr>
<td>interpolate</td>
<td>Interpolate data?</td>
<td>true/false</td>
</tr>
<tr>
<td>triangulate</td>
<td>Triangulate surface? (optional)</td>
<td>true/false</td>
</tr>
</tbody>
</table>

Table 6.10: Entries for a plane in surfaces sub-dictionaries.

The debug switch is set to 1 or 2 (default) in DebugSwitches in the $WM_PROJECT_DIR/etc/-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**

```
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 = -6.41057e-20, cumulative = -6.41057e-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
```
Table 6.11: Entries for a patch in surfaces sub-dictionaries.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>patchName</td>
<td>Name of patch</td>
<td>e.g. movingWall</td>
</tr>
<tr>
<td>interpolate</td>
<td>Interpolate data?</td>
<td>true/false</td>
</tr>
<tr>
<td>triangulate</td>
<td>Triangulate surface? (optional)</td>
<td>true/false</td>
</tr>
</tbody>
</table>

6.7.1 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 log:

    foamJob <solver>

For further options the user should execute foamJob -help. 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 log

6.7.2 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 <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 6.5. It can be seen here that the residual falls monotonically until it reaches the convergence tolerance of 10^{-5}.

![Figure 6.5: Initial residual of Ux in the cavity tutorial](image)

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 7
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 must be specified at runtime.

7.1 Thermophysical models

Thermophysical models are concerned with 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 package of thermophysical modelling that is used in the simulation. OpenFOAM includes a large set of pre-compiled combinations of modelling, built within the code using C++ templates. This coding approach assembles thermophysical modelling packages beginning with the equation of state and then adding more layers of thermophysical modelling that derive properties from the previous layer(s). The keyword entries in thermoType reflects the multiple layers of modelling and the underlying framework in which they combined. Below is an example entry for thermoType:

```plaintext
thermoType
{
    type hePsiThermo;
mixture pureMixture;
transport const;
thermo hConst;
equationOfState perfectGas;
specie specie;
energy sensibleEnthalpy;
}
```
The keyword entries specify the choice of thermophysical models, e.g. constant transport (constant viscosity, thermal diffusion), Perfect Gas equationOfState, etc. In addition there is a keyword entry named energy that allows the user to specify the form of energy to be used in the solution and thermodynamics. The following sections explains the entries and options in the thermoType package.

7.1.1 Thermophysical and mixture models

Each solver that uses thermophysical modelling constructs an object of a particular thermophysical model class. The model classes are listed below.

psiThermo Thermophysical model for fixed composition, based on compressibility \( \psi = (RT)^{-1} \), where \( R \) is Gas Constant and \( T \) is temperature. The solvers that construct psiThermo include the compressible family of solvers (sonicFoam, simpleFoam, etc., excluding rhoPorousSimpleFoam) and uncoupledKinematicParcelFoam and coldEngineFoam.

rhoThermo Thermophysical model for fixed composition, based on density \( \rho \). The solvers that construct rhoThermo include the heatTransfer family of solvers (buoyantSimpleFoam, CHT solvers, etc., excluding Boussinesq solvers) and rhoPorousSimpleFoam, twoPhaseEulerFoam and thermoFoam.

psiReactionThermo Thermophysical model for reacting mixture, based on \( \psi \). The solvers that construct psiReactionThermo include many of the combustion solvers, e.g. sprayFoam, chemFoam, fireFoam and reactingFoam, and some lagrangian solvers, e.g. coalChemistryFoam and reactingParcelFilmFoam.

psiuReactionThermo Thermophysical model for combustion, based on compressibility of unburnt gas \( \psi_u \). The solvers that construct rhoReactionThermo include some combustion solvers, e.g. rhoReactingFoam, rhoReactingBuoyantFoam, and some lagrangian solvers, e.g. reactingParcelFoam and simpleReactingParcelFoam.

rhoReactionThermo Thermophysical model for reacting mixture, based on \( \rho \). The solvers that construct psiuReactionThermo include combustion solvers that model combustion based on laminar flame speed and regress variable, e.g. XiFoam, PDRFoam and engineFoam.

multiphaseMixtureThermo Thermophysical models for multiple phases. The solvers that construct multiphaseMixtureThermo include compressible multiphase interface-capturing solvers, e.g. compressibleInterFoam, and compressibleMultiphaseInterFoam.

The type keyword specifies the underlying thermophysical model. Options are listed below.

- hePsiThermo: for solvers that construct psiThermo and psiReactionThermo.
- heRhoThermo: for solvers that construct rhoThermo, rhoReactionThermo and multiphaseMixtureThermo.
- heheuPsiThermo: for solvers that construct psiuReactionThermo.
The mixture specifies the mixture composition. The option typically used for thermophysical models without reactions is pureMixture, which represents a mixture with fixed composition. When pureMixture is specified, the thermophysical models coefficients are specified within a sub-dictionary called mixture.

For mixtures with variable composition, required by thermophysical models with reactions, the reactingMixture option is used. Species and reactions are listed in a chemistry file, specified by the foamChemistryFile keyword. The reactingMixture model then requires the thermophysical models coefficients to be specified for each specie within sub-dictionaries named after each specie, e.g. O2, N2.

For combustion based on laminar flame speed and regress variables, constituents are a set of mixtures, such as fuel, oxidant and burntProducts. The available mixture models for this combustion modelling are homogeneousMixture, inhomogeneousMixture and veryInhomogeneousMixture.

Other models for variable composition are egrMixture, multiComponentMixture and singleStepReactingMixture.

7.1.2 Transport model

The transport modelling concerns evaluating dynamic viscosity $\mu$, thermal conductivity $\kappa$ and thermal diffusivity $\alpha$ (for internal energy and enthalpy equations). The current transport models are as follows:

const assumes a constant $\mu$ and Prandtl number $Pr = c_p\mu/\kappa$ which is simply specified by a two keywords, mu and Pr, respectively.

sutherland calculates $\mu$ as a function of temperature $T$ from a Sutherland coefficient $A_s$ and Sutherland temperature $T_s$, specified by keywords As and Ts; $\mu$ is calculated according to:

$$\mu = \frac{A_s\sqrt{T}}{1 + T_s/T}.$$  \hspace{1cm} (7.1)

polynomial calculates $\mu$ and $\kappa$ as a function of temperature $T$ from a polynomial of any order $N$, e.g.:

$$\mu = \sum_{i=0}^{N-1} a_i T^i.$$  \hspace{1cm} (7.2)

7.1.3 Thermodynamic models

The thermodynamic models are concerned with evaluating the specific heat $c_p$ from which other properties are derived. The current thermo models are as follows:

hConst assumes a constant $c_p$ and a heat of fusion $H_f$ which is simply specified by a two values $c_p$ and $H_f$, given by keywords Cp and Hf.

eConst assumes a constant $c_v$ and a heat of fusion $H_f$ which is simply specified by a two values $c_v$ and $H_f$, given by keywords Cv and Hf.


\textbf{Janaf} calculates \( c_p \) as a function of temperature \( T \) from a set of coefficients taken from \textit{JANAF} tables of thermodynamics. The ordered list of coefficients is given in Table 7.1. 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:

\[
    c_p = R(((a_4 T + a_3)T + a_2)T + a_1)T + a_0). \tag{7.3}
\]

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.

\textbf{hPolynomial} calculates \( c_p \) as a function of temperature by a polynomial of any order \( N \):

\[
    c_p = \sum_{i=0}^{N-1} a_i T^i. \tag{7.4}
\]

The following case provides an example of its use: \texttt{$FOAM\_TUTORIALS/lagrangian/-porousExplicitSourceReactingParcelFoam/filter}

<table>
<thead>
<tr>
<th>Description</th>
<th>Entry</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower temperature limit</td>
<td>( T_l ) (K)</td>
<td>\texttt{Tlow}</td>
</tr>
<tr>
<td>Upper temperature limit</td>
<td>( T_h ) (K)</td>
<td>\texttt{Thigh}</td>
</tr>
<tr>
<td>Common temperature</td>
<td>( T_c ) (K)</td>
<td>\texttt{Tcommon}</td>
</tr>
<tr>
<td>High temperature coefficients</td>
<td>( a_0 ) . . . ( a_4 )</td>
<td>\texttt{highCpCoeffs (a0 a1 a2 a3 a4...}</td>
</tr>
<tr>
<td>High temperature enthalpy offset</td>
<td>( a_5 )</td>
<td>\texttt{a5...}</td>
</tr>
<tr>
<td>High temperature entropy offset</td>
<td>( a_6 )</td>
<td>\texttt{a6)</td>
</tr>
<tr>
<td>Low temperature coefficients</td>
<td>( a_0 ) . . . ( a_4 )</td>
<td>\texttt{lowCpCoeffs (a0 a1 a2 a3 a4...}</td>
</tr>
<tr>
<td>Low temperature enthalpy offset</td>
<td>( a_5 )</td>
<td>\texttt{a5...}</td>
</tr>
<tr>
<td>Low temperature entropy offset</td>
<td>( a_6 )</td>
<td>\texttt{a6)</td>
</tr>
</tbody>
</table>

Table 7.1: \textit{JANAF} thermodynamics coefficients.

\section*{7.1.4 Composition of each constituent}

There is currently only one option for the \texttt{specie} model which specifies the composition of each constituent. That model is itself named \texttt{specie}, which is specified by the following entries.

- \texttt{nMoles}: number of moles of component. This entry is only used for combustion modelling based on regress variable with a homogeneous mixture of reactants; otherwise it is set to 1.

- \texttt{molWeight} in grams per mole of specie.
7.1 Thermophysical models

7.1.5 Equation of state

The following equations of state are available in the thermophysical modelling library.

rhoConst Constant density:

\[ \rho = \text{constant}. \]  \hspace{1cm} (7.5)

perfectGas Perfect gas:

\[ \rho = \frac{1}{RT} p. \]  \hspace{1cm} (7.6)

incompressiblePerfectGas Perfect gas for an incompressible fluid:

\[ \rho = \frac{1}{RT} p_{\text{ref}}, \]  \hspace{1cm} (7.7)

where \( p_{\text{ref}} \) is a reference pressure.

perfectFluid Perfect fluid:

\[ \rho = \frac{1}{RT} p + \rho_0, \]  \hspace{1cm} (7.8)

where \( \rho_0 \) is the density at \( T = 0 \).

linear Linear equation of state:

\[ \rho = \psi p + \rho_0, \]  \hspace{1cm} (7.9)

where \( \psi \) is compressibility (not necessarily \( (RT)^{-1} \)).

adiabaticPerfectFluid Adiabatic perfect fluid:

\[ \rho = \rho_0 \left( \frac{p + B p_0}{p_0 + B} \right)^{1/\gamma}, \]  \hspace{1cm} (7.10)

where \( \rho_0, p_0 \) are reference density and pressure respectively, and \( B \) is a model constant.

PengRobinsonGas Peng Robinson equation of state:

\[ \rho = \frac{1}{zRT} p, \]  \hspace{1cm} (7.11)

where the complex function \( z = z(p, T) \) can be referenced in the source code in \texttt{PengRobinsonGasI.H}, in the \$FOAM\_SRC/thermophysicalModels/specie/equationOfState/ directory.

icoPolynomial Incompressible, polynomial equation of state:

\[ \rho = \sum_{i=0}^{N-1} a_i T^i, \]  \hspace{1cm} (7.12)

where \( a_i \) are polynomial coefficients of any order \( N \).
7.1.6 Selection of energy variable

The user must specify the form of energy to be used in the solution, either internal energy $e$ and enthalpy $h$, and in forms that include the heat of formation $\Delta h_f$ or not. This choice is specified through the `energy` keyword.

We refer to absolute energy where heat of formation is included, and sensible energy where it is not. For example absolute enthalpy $h$ is related to sensible enthalpy $h_s$ by

$$ h = h_s + \sum_i c_i \Delta h_f^i $$

where $c_i$ and $h_f^i$ are the molar fraction and heat of formation, respectively, of specie $i$. In most cases, we use the sensible form of energy, for which it is easier to account for energy change due to reactions. Keyword entries for `energy` therefore include e.g. `sensibleEnthalpy`, `sensibleInternalEnergy` and `absoluteEnthalpy`.

7.1.7 Thermophysical property data

The basic thermophysical properties are specified for each species from input data. Data entries must contain the name of the species as the keyword, e.g. `O2`, `H2O`, `mixture`, followed by sub-dictionaries of coefficients, including:

- `specie` containing *i.e.* number of moles, `nMoles`, of the species, and molecular weight, `molWeight` in units of g/mol;
- `thermodynamics` containing coefficients for the chosen thermodynamic model (see below);
- `transport` containing coefficients for the chosen transport model (see below).

The following is an example entry for a species named `fuel` modelled using `sutherland` transport and `janaf` thermodynamics:

```
fuel
{
    specie
    {
        nMoles 1;
        molWeight 16.0428;
    }
    thermodynamics
    {
        Tlow 200;
        Thigh 6000;
        Tcommon 1000;
        highCpCoeffs (1.63543 0.0100844 -3.36924e-06 5.34973e-10
                      -3.15528e-14 -10005.6 9.9937);
        lowCpCoeffs (5.14988 -0.013671 4.91801e-05 -4.85744e-08
                      1.66694e-11 -10246.6 -4.64132);
    }
    transport
```

OpenFOAM-3.0.1
The following is an example entry for a specie named \texttt{air} modelled using \texttt{const transport} and \texttt{hConst} thermodynamics:

\begin{verbatim}
air
{
    specie
    {
        nMoles 1;
        molWeight 28.96;
    }
    thermodynamics
    {
        Cp 1004.5;
        Hf 2.544e+06;
    }
    transport
    {
        mu 1.8e-05;
        Pr 0.7;
    }
}
\end{verbatim}

\section{7.2 Turbulence models}

The \texttt{turbulenceProperties} dictionary is read by any solver that includes turbulence modelling. Within that file is the \texttt{simulationType} keyword that controls the type of turbulence modelling to be used, either:

- \texttt{laminar} uses no turbulence models;
- \texttt{RAS} uses Reynolds-averaged stress (RAS) modelling;
- \texttt{LES} uses large-eddy simulation (LES) modelling.

\textbf{Note for OpenFOAM versions prior to v3.0.0}: the keyword options are instead \texttt{RASModel} and \texttt{LESModel} respectively.

If \texttt{RAS} is selected, the choice of RAS modelling is specified in a \texttt{RAS} sub-dictionary, also in the \texttt{constant} directory. The RAS turbulence model is selected by the \texttt{RASModel} entry from a long list of available models that are listed in Table 3.9. Similarly, if \texttt{LES} is selected, the choice of LES modelling is specified in a \texttt{LES} dictionary and the LES turbulence model is selected by the \texttt{LESModel} entry.
Note for OpenFOAM versions prior to v3.0.0: the RAS modelling is specified in a separate RASProperties file rather than in a RAS sub-dictionary of turbulenceProperties; similarly, LES modelling is in a separate LESProperties file.

The entries required in the RAS sub-dictionary are listed in Table 7.2 and those for LES sub-dictionary dictionaries are listed in Table 7.3. The incompressible and compressible RAS turbulence models, isochoric and anisochoric LES models and delta models are all named and described in Table 3.9. Examples of their use can be found in the $FOAM_TUTORIALS.

### 7.2.1 Model coefficients

The coefficients for the RAS turbulence models are given default values in their respective source code. If the user wishes to override these default values, then they can do so by adding a sub-dictionary entry to the RAS sub-dictionary file, whose keyword name is that of the model with Coeffs appended, e.g. kEpsilonCoeffs for the kEpsilon model. If the printCoeffs switch is on in the RAS sub-dictionary, an example of the relevant ...Coeffs dictionary is printed to standard output when the model is created at the beginning of a run. The user can simply copy this into the RAS sub-dictionary file and edit the entries as required.

### 7.2.2 Wall functions

A range of wall function models is available in OpenFOAM that are applied as boundary conditions on individual patches. This enables different wall function models to be applied to different wall regions. The choice of wall function model is specified through the turbulent viscosity field $\nu_t$ in the 0/nut file. Note for OpenFOAM versions prior to v3.0.0: wall functions for compressible RAS are specified through the $\mu_t$ field in the 0/mut file, through $\nu_{sgs}$ in the 0/nuSgs file for incompressible LES and $\mu_{sgs}$ in the 0/muSgs file for compressible LES. For example, a 0/nut file:

```
 dimensions [0 2 -1 0 0 0 0];
```
7.3 Transport/rheology models

There are a number of wall function models available in the release, e.g. nutWallFunction, nutRoughWallFunction, nutUSpaldingWallFunction, nutkWallFunction and nutkAtmWallFunction. The user can consult the relevant directories for a full list of wall function models:

```
find $FOAM_SRC/TurbulenceModels -name wallFunctions
```

Within each wall function boundary condition the user can over-ride default settings for \( E \), \( \kappa \) and \( C_\mu \) through optional \( E \), kappa and Cmu keyword entries.

Having selected the particular wall functions on various patches in the nut/mut file, the user should select epsilonWallFunction on corresponding patches in the epsilon field and kqRwallFunction on corresponding patches in the turbulent fields \( k \), \( q \) and \( R \).

### 7.3 Transport/rheology models

In OpenFOAM, solvers that do not include energy/heat, include a library of models for viscosity \( \nu \). The models typically relate viscosity to strain rate \( \dot{\gamma} \) and are specified by the user in the transportProperties dictionary. The available models are listed in the following sections.

#### 7.3.1 Newtonian model

The Newtonian model assumes \( \nu \) is constant. Viscosity is specified by a dimensionedScalar nu in transportProperties, e.g.

```
transportModel Newtonian;

nu nu [ 0 2 -1 0 0 0 0 ] 1.5e-05;
```

Note the units for kinematic viscosity are \( \text{L}^2/\text{T} \).
7.3.2 Bird-Carreau model

The Bird-Carreau model is:

\[
\nu = \nu_\infty + (\nu_0 - \nu_\infty) \left[ 1 + (k\dot{\gamma})^a \right]^{(n-1)/a} \tag{7.14}
\]

where the coefficient \(a\) has a default value of 2. An example specification of the model in \textit{transportProperties} is:

```plaintext
transportModel BirdCarreau;
BirdCarreauCoeffs
{
    nu0 nu0 [ 0 2 -1 0 0 0 0 ] 1e-03;
    nuInf nuInf [ 0 2 -1 0 0 0 0 ] 1e-05;
    k k [ 0 0 1 0 0 0 0 ] 1;
    n n [ 0 0 0 0 0 0 0 ] 0.5;
}
```

7.3.3 Cross Power Law model

The Cross Power Law model is:

\[
\nu = \nu_\infty + \frac{\nu_0 - \nu_\infty}{1 + (m\dot{\gamma})^n} \tag{7.15}
\]

An example specification of the model in \textit{transportProperties} is:

```plaintext
transportModel CrossPowerLaw;
CrossPowerLawCoeffs
{
    nu0 nu0 [ 0 2 -1 0 0 0 0 ] 1e-03;
    nuInf nuInf [ 0 2 -1 0 0 0 0 ] 1e-05;
    m m [ 0 0 1 0 0 0 0 ] 1;
    n n [ 0 0 0 0 0 0 0 ] 0.5;
}
```

7.3.4 Power Law model

The Power Law model provides a function for viscosity, limited by minimum and maximum values, \(\nu_{\text{min}}\) and \(\nu_{\text{max}}\) respectively. The function is:

\[
\nu = k\dot{\gamma}^{n-1}, \quad \nu_{\text{min}} \leq \nu \leq \nu_{\text{max}} \tag{7.16}
\]

An example specification of the model in \textit{transportProperties} is:

```plaintext
transportModel powerLaw;
powerLawCoeffs
{
    nuMax nuMax [ 0 2 -1 0 0 0 0 ] 1e-03;
}
```

OpenFOAM-3.0.1
7.3 Transport/rheology models

nuMin
nuMin  [ 0 2 -1 0 0 0 0 ] 1e-05;
k
k      [ 0 2 -1 0 0 0 0 ] 1e-05;
n
n      [ 0 0 0 0 0 0 0 ] 1;

7.3.5 Herschel-Bulkley model

The Herschel-Bulkley model combines the effects of Bingham plastic and power-law behavior in a fluid. For low strain rates, the material is modelled as a very viscous fluid with viscosity $\nu_0$. Beyond a threshold in strain-rate corresponding to threshold stress $\tau_0$, the viscosity is described by a power law. The model is:

$$\nu = \min (\nu_0, \frac{\tau_0}{\dot{\gamma}} + k\dot{\gamma}^{n-1})$$  \hspace{1cm} (7.17)

An example specification of the model in transportProperties is:

transportModel HerschelBulkley;
HerschelBulkleyCoeffs
{
    nu0   nu0  [ 0 2 -1 0 0 0 0 ] 1e-03;
    tau0  tau0 [ 0 2 -2 0 0 0 0 ] 1;
    k     k     [ 0 2 -1 0 0 0 0 ] 1e-05;
    n     n     [ 0 0 0 0 0 0 0 ] 1;
}
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-23
+
  tensor member function, P-23
-
  tensor member function, P-23
/
  tensor member function, P-23
/*...*/
  C++ syntax, U-78
//
  C++ syntax, U-78
OpenFOAM file syntax, U-106
# include
  C++ syntax, U-72, U-78
&
  tensor member function, P-23
&&
  tensor member function, P-23
^
  tensor member function, P-23

Numbers
0 directory, U-106

A
access functions, P-21
addLayersControls keyword, U-150
adiabatic FlameT utility, U-97
adiabatic PerfectFluid model, U-101, U-195
adjointShapeOptimizationFoam solver, U-86
adjustableRunTime
  keyword entry, U-62, U-114
adjustTimeStep keyword, U-62, U-115
agglomerator keyword, U-125
algorithms tools, U-98
alphaContactAngle
  boundary condition, U-59
analytical solution, P-43
Animations window panel, U-174
anisotropicFilter model, U-103
Annotation window panel, U-24
ansysToFoam utility, U-91
APIfunctions model, U-102
applications, U-69
Apply button, U-170, U-174
applyBoundaryLayer utility, U-90
applyWallFunctionBoundaryConditions utility, U-90
arbitrarily unstructured, P-29
arc
  keyword entry, U-142
arc keyword, U-141
As keyword, U-193
ascii
  keyword entry, U-114
attachMesh utility, U-92
Auto Accept button, U-174
autoMesh
  library, U-99
autoPatch utility, U-92
autoRefineMesh utility, U-93
axes
  right-handed, U-140
  right-handed rectangular Cartesian, P-13, U-18
axi-symmetric cases, U-137, U-148
axi-symmetric mesh, U-133
**B**

background
- process, U-24, U-81

backward
- keyword entry, U-122
Backward differencing, P-37

barotropicCompressibilityModels
- library, U-101

basicMultiComponentMixture model, U-100

basicSolidThermo
- library, U-102

basicThermophysicalModels
- library, U-100

binary
- keyword entry, U-114
BirdCarreau model, U-104

blended differencing, P-36
block
- expansion ratio, U-143
block keyword, U-141
blocking
- keyword entry, U-80

blockMesh
- library, U-99
blockMesh solver, P-45
blockMesh utility, U-38, U-91, U-140
blockMesh executable
- vertex numbering, U-143

*blockMeshDict*
- dictionary, U-18, U-20, U-36, U-49, U-140, U-149
blocks keyword, U-20, U-31, U-142
boundaries, U-133
boundary, U-133

**boundary**
- dictionary, U-132, U-140
boundary keyword, U-145
boundary condition

- alphaContactAngle, U-59
- buoyantPressure, U-139
- calculated, U-138
- cyclic, U-137, U-146
- directionMixed, U-138
- empty, P-63, P-69, U-18, U-133, U-137
- fixedGradient, U-138
- fixedValue, U-138
- fluxCorrectedVelocity, U-139
- inlet, P-69
- inletOutlet, U-139
- mixed, U-138

movingWallVelocity, U-139
outlet, P-69
outletInlet, U-139
partialSlip, U-139
patch, U-136
pressureDirectedInletVelocity, U-139
pressureInletVelocity, U-139
pressureOutlet, P-63
pressureTransmissive, U-139
processor, U-138
setup, U-20
slip, U-139
supersonicFreeStream, U-139
surfaceNormalFixedValue, U-139
symmetryPlane, P-63, U-137
totalPressure, U-139
turbulentInlet, U-139
wall, U-41
wall, P-63, P-69, U-59, U-137
wedge, U-133, U-137, U-148
zeroGradient, U-138
boundary conditions, P-41
Dirichlet, P-41
inlet, P-42
Neumann, P-41
no-slip impermeable wall, P-42
outlet, P-42
physical, P-42
symmetry plane, P-42

*boundaryField*
- keyword, U-21, U-110

boundaryFoam solver, U-86
bounded
- keyword entry, U-120, U-121
boxToCell keyword, U-60
boxTurb utility, U-90
breaking of a dam, U-56
BSpline
- keyword entry, U-142
buoyantBoussinesqPimpleFoam solver, U-88
buoyantBoussinesqSimpleFoam solver, U-88
buoyantPimpleFoam solver, U-88
buoyantPressure
- boundary condition, U-139
buoyantSimpleFoam solver, U-88
burntProducts keyword, U-193
button
- Apply, U-170, U-174
- Auto Accept, U-174
Camera Parallel Projection, U-174
Choose Preset, U-173
Index

U-205

Delete, U-170
Edit Color Map, U-172
Enable Line Series, U-35
Lights, U-174
Orientation Axes, U-24
Refresh Times, U-25, U-171
Rescale to Data Range, U-25
Reset, U-170
Set Ambient Color, U-173
Update GUI, U-171
Use Parallel Projection, U-24

C

C++ syntax
/*...*/
/, U-78
#/include, U-72, U-78
CachedAgglomeration keyword, U-126
calculated boundary condition, U-138
cAlpha keyword, U-63
camera Parallel Projection button, U-174
cases, U-105
castellatedMesh keyword, U-150
castellatedMeshControls
dictionary, U-152–U-154
castellatedMeshControls keyword, U-150
cavitatingDyMFoam solver, U-87
cavitatingFoam solver, U-87
cavity flow, U-17
CEI
ARCH environment variable, U-184
CEI
HOME environment variable, U-184
cell expansion ratio, U-143
cell class, P-29
cell keyword entry, U-185
cellLimited keyword entry, U-120

cellPoint keyword entry, U-185
cellPointFace
keyword entry, U-185
cells
dictionary, U-140
central differencing, P-36
cfdTools tools, U-98
cfx4ToFoam utility, U-91, U-159
changeDictionary utility, U-90
Charts window panel, U-174
checkMesh utility, U-92, U-160
chemFoam solver, U-88
chemistryModel
library, U-102
chemistryModel model, U-102
chemistrySolver model, U-102
cemkinToFoam utility, U-97
Choose Preset button, U-173
chtMultiRegionSimpleFoam solver, U-88
chtMultiRegionFoam solver, U-88
Chung
library, U-101
class
cell, P-29
dimensionSet, P-24, P-30, P-31
face, P-29
finiteVolumeCalculi, P-34
finiteVolumeMethod, P-34
fvMesh, P-29
fvSchemes, P-36
fvC, P-34
fvm, P-34
pointField, P-29
polyBoundaryMesh, P-29
polyMesh, P-29, U-129, U-131
polyPatchList, P-29
polyPatch, P-29
scalarField, P-27
scalar, P-22
slice, P-29
symmTensorField, P-27
symmTensorThirdField, P-27
tensorField, P-27
tensorThirdField, P-27
tensor, P-22
vectorField, P-27
vector, P-22, U-109
word, P-24, P-29
class keyword, U-107
clockTime
keyword entry, U-114
cloud keyword, U-187
cloudFunctionObjects
library, U-98
cmpAv
tensor member function, P-23
Co utility, U-93
coalChemistryFoam solver, U-89
coalCombustion
Index

library, U-99
cofactors
tensor member function, P-23
coldEngineFoam solver, U-88
collapseEdges utility, U-93
Color By menu, U-173
Color Legend window, U-29
Color Legend window panel, U-173
Color Scale window panel, U-173
Colors window panel, U-174
compressibleInterDyMFoam solver, U-87
compressibleInterFoam solver, U-87
compressibleMultiphaseInterFoam solver, U-87
combinePatchFaces utility, U-93
comments, U-78
commsType keyword, U-80
compressed
keyword entry, U-114
compressibleLESModels
library, U-104
compressibleRASModels
library, U-103
constant directory, U-105, U-191
constant model, U-101
constTransport model, U-101
containers tools, U-98
continuum
mechanics, P-13
control
of time, U-113
controlDict
controlDict file, P-48
convection, see divergence, P-36
convergence, U-39
conversion
library, U-99
convertToMeters keyword, U-140
convertToMeters keyword, U-141
coordinate
system, P-13
coordinate system, U-18
corrected
keyword entry, U-120, U-121
Courant number, P-40, U-22
Cp keyword, U-193
cpuTime
keyword entry, U-114
Crank Nicolson

temporal discretisation, P-41
CrankNicolson
keyword entry, U-122
createExternalCoupledPatchGeometry utility, U-90
createBaffles utility, U-92
createPatch utility, U-92
createTurbulenceFields utility, U-94
cross product, see tensor, vector cross product
CrossPowerLaw
keyword entry, U-60
CrossPowerLaw model, U-104
cubeRootVolDelta model, U-103
cubicCorrected
keyword entry, U-122
cubicCorrection
keyword entry, U-119
curl, P-35
curl
fvc member function, P-35
Current Time Controls menu, U-25, U-171
curve keyword, U-187
Cv keyword, U-193
cyclic
boundary condition, U-137, U-146
cyclic
keyword entry, U-137
cyldiner
flow around a, P-43

d
D2dt2
fvc member function, P-35
fvm member function, P-35
dam
breaking of a, U-56
datToFoam utility, U-91
db tools, U-98
ddt
fvc member function, P-35
fvm member function, P-35
DeardorffDiffStress model, U-103, U-104
debug keyword, U-150
decompose model, U-100
decomposePar utility, U-82, U-83, U-97
decomposeParDict
dictionary, U-82
decomposition
of field, U-82
of mesh, U-82
decompositionMethods

OpenFOAM-3.0.1
library, U-99
decompression of a tank, P-61
defaultFieldValues keyword, U-60
deformedGeom utility, U-92
Delete button, U-170
delta keyword, U-83, U-198
deltaT keyword, U-114
dependencies, U-72
dependency lists, U-72
det
tensor member function, P-23
determinant, see tensor, determinant
dev
tensor member function, P-23
diag
tensor member function, P-23
diagonal
keyword entry, U-124, U-125
DIC
keyword entry, U-125
DICGaussSeidel
keyword entry, U-125
dictionary
PISO, U-23
blockMeshDict, U-18, U-20, U-36, U-49,
U-140, U-149
boundary, U-132, U-140
castellatedMeshControls, U-152–U-154
cells, U-140
controlDict, P-65, U-22, U-31, U-42, U-51,
U-62, U-105, U-165
decomposeParDict, U-82
faces, U-131, U-140
tfScSchemes, U-63, U-105, U-116
tfSolution, U-105, U-123
mechanicalProperties, U-51
neighbour, U-132
owner, U-131
points, U-131, U-140
thermalProperties, U-51
thermophysicalProperties, U-191
transportProperties, U-21, U-39, U-42, U-199
turbulenceProperties, U-41, U-61, U-197
differencing
Backward, P-37
blended, P-36
central, P-36
Gamma, P-36
MINMOD, P-36
SUPERBEE, P-36
upwind, P-36
van Leer, P-36
DILU
keyword entry, U-125
dimension
checking in OpenFOAM, P-24, U-109
dimensional units, U-109
dimensioned<Type> template class, P-24
dimensionedTypes tools, U-98
dimensions keyword, U-21, U-110
dimensionSet class, P-24, P-30, P-31
dimensionSet tools, U-98
directionMixed
boundary condition, U-138
directory
0.000000e+00, U-106
0, U-106
Make, U-73
constant, U-105, U-191
fluuentInterface, U-181
polyMesh, U-105, U-131
processorN, U-83
run, U-105
system, P-48, U-105
tutorials, P-43, U-17
discretisation
equation, P-31
Display window panel, U-24, U-25, U-170, U-172
distance
keyword entry, U-154, U-187
distributed model, U-100
distributed keyword, U-83, U-84
distributionModels
library, U-99
div
fvc member function, P-35
fvm member function, P-35
divergence, P-35, P-37
divSchemes keyword, U-116
dnsFoam solver, U-88
doLayers keyword, U-150
double inner product, see tensor,double inner
product
DPMFoam solver, U-89
driftFluxFoam solver, U-87
dsmc
library, U-99
dsmcFieldsCalc utility, U-95
dsmcFoam solver, U-89
dsmcInitialise utility, U-90
dx
  keyword entry, U-185
dynamicFvMesh
  library, U-99
dynamicMesh
  library, U-99
dynLagrangian model, U-103
dynOneEqEddy model, U-103

e
edgeGradingThermo model, U-101
dynamicFvMesh
  library, U-99
dynamicMesh
  library, U-99
dynLagrangian model, U-103
dynOneEqEddy model, U-103

e
edgeGradingThermo model, U-101
dynamicFvMesh
  library, U-99
dynamicMesh
  library, U-99
dynLagrangian model, U-103
dynOneEqEddy model, U-103

e
edgeGradingThermo model, U-101
dynamicFvMesh
  library, U-99
dynamicMesh
  library, U-99
dynLagrangian model, U-103
dynOneEqEddy model, U-103

E
econstThermo model, U-101
depedgedGrading keyword, U-143
dynamicFvMesh
  library, U-99
dynamicMesh
  library, U-99
dynLagrangian model, U-103
dynOneEqEddy model, U-103

e
empty
  boundary condition, P-63, P-69, U-18,
    U-133, U-137

E
e
  keyword entry, U-137
Enable Line Series button, U-35
time keyword, U-22, U-113, U-114
definition keyword, U-192, U-196
engine
  library, U-99
definitionRatio model, U-95
definitionFoam solver, U-88
definitionSwirl utility, U-90
definition7FoamExec utility, U-183
definition7_INPUT
  environment variable, U-184
definition7_READER
  environment variable, U-184
definitionFoamReader utility, U-93
definitions utility, U-93
environment variable
  CELARCH, U-184
  CELHOME, U-184
definition7_INPUT, U-184
definition7_READER, U-184
  FOAM_RUN, U-105
  WM_ARCH_OPTION, U-76
  WM_ARCH, U-76
  WM_COMPILER_BIN, U-76
  WM_COMPILER_DIR, U-76
  WM_COMPILER, U-76
  WM_COMPILE_OPTION, U-76
  WM_DIR, U-76
  WM_MPLIB, U-76
  WM_OPTIONS, U-76
  WM_PRECISION_OPTION, U-76
  WM_PROJECT_DIR, U-76
  WM_PROJECT_INST_DIR, U-76
  WM_PROJECT_USER_DIR, U-76
  WM_PROJECT_VERSION, U-76
  WM_PROJECT, U-76
wmake, U-76
e
  keyword entry, U-122
Euler implicit
differencing, P-37
temporal discretisation, P-40
examples
decompression of a tank, P-61
flow around a cylinder, P-43
flow over backward step, P-50
Hartmann problem, P-67
supersonic flow over forward step, P-58
execFlowFunctionObjects utility, U-95
declensionDictionary utility, U-97
definitionRatio keyword, U-157
explicit
temporal discretisation, P-40
e
extrude2DMesh utility, U-91
extrudeMesh utility, U-91
e
extrudeToRegionMesh utility, U-91
  F

class, P-29

class keyword, U-187
classAgglomerate utility, U-90
classAreaPair
  keyword entry, U-125
classLimited
  keyword entry, U-120
class
  dictionary, U-131, U-140
FDIC
  keyword entry, U-125
featureAngle keyword, U-157
features keyword, U-152
field
  U, U-23
  p, U-23
decomposition, U-82
FieldField<Type> template class, P-30
fieldFunctionObjects
  library, U-98
fields, P-27
  mapping, U-165
fields tools, U-98
fields keyword, U-185
Field<Type> template class, P-27
fieldValues keyword, U-60
file
  Make/files, U-75
  controlDict, P-48
  files, U-73
g, U-60
  options, U-73
  snappyHexMeshDict, U-150
  transportProperties, U-60
file format, U-106
fileFormats
  library, U-99
fileModificationChecking keyword, U-80
fileModificationSkew keyword, U-80
files file, U-73
filteredLinear2
  keyword entry, U-119
finalLayerThickness keyword, U-157
financialFoam solver, U-90
find script/alias, U-179
finite volume
discretisation, P-25
  mesh, P-29
finiteVolume
  library, U-98
finiteVolume tools, U-98
finiteVolumeCalculus class, P-34
finiteVolumeMethod class, P-34
fireFoam solver, U-88
firstTime keyword, U-113
fixed
  keyword entry, U-114
fixedGradient
  boundary condition, U-138
fixedValue
  boundary condition, U-138
flattenMesh utility, U-92
floatTransfer keyword, U-80
flow
  free surface, U-56
  laminar, U-17
  steady, turbulent, P-50
  supersonic, P-58
  turbulent, U-17
flow around a cylinder, P-43
flow over backward step, P-50
flowType utility, U-93
fluent3DMeshToFoam utility, U-91
fluentInterface directory, U-181
fluentMeshToFoam utility, U-91, U-159
fluxCorrectedVelocity
  boundary condition, U-139
fluxRequired keyword, U-116
OpenFOAM
cases, U-105
FOAM_RUN
  environment variable, U-105
foamCalc utility, U-33
foamCalcFunctions
  library, U-98
foamChemistryFile keyword, U-193
foamCorrectVrt script/alias, U-164
foamDataToFluent utility, U-93, U-181
foamDebugSwitches utility, U-97
FoamFile keyword, U-107
foamFile
  keyword entry, U-185
foamFormatConvert utility, U-97
foamHelp utility, U-97
foamInfoExec utility, U-97
foamJob script/alias, U-188
foamListTimes utility, U-95
foamLog script/alias, U-188
foamMeshToFoam utility, U-91, U-181
foamToEnsight utility, U-93
foamToEnsightParts utility, U-93
foamToGMV utility, U-93
foamToStarMesh utility, U-91
foamToSurface utility, U-91
foamToTecplot360 utility, U-93
foamToTetDualMesh utility, U-93
foamToVTK utility, U-93
foamUpgradeCyclics utility, U-90
foamUpgradeFvSolution utility, U-90
foamyHexMeshBackgroundMesh utility, U-91
foamyHexMeshSurfaceSimplify utility, U-91
foamyHexMesh utility, U-91
foamyQuadMesh utility, U-91
forces library, U-98
foreground process, U-24
format keyword, U-107
fourth keyword entry, U-120, U-121
fuel keyword, U-193
functionObjectLibs keyword, U-179
functions keyword, U-115, U-177
fvc class, P-34
fvc member function
curl, P-35
d2dt2, P-35
ddt, P-35
div, P-35
gGrad, P-35
grad, P-35
laplacian, P-35
lsGrad, P-35
snGrad, P-35
snGradCorrection, P-35
sqrGradGrad, P-35
fvDOM library, U-101
FVFunctionObjects library, U-98
fvm class, P-34
fvm member function
d2dt2, P-35
ddt, P-35
div, P-35
laplacian, P-35
Su, P-35
SuSp, P-35
fvMatrices tools, U-98
fvMatrix template class, P-34
fvMesh class, P-29
fvMesh tools, U-98
fvMotionSolvers library, U-99
fvSchemes dictionary, U-63, U-105, U-116
fvSchemes class, P-36
default menu entry, U-52
default dictionary, U-105, U-123

g file, U-60
gambitToFoam utility, U-91, U-159
GAMG keyword entry, U-53, U-124, U-125
Gamma keyword entry, U-119
Gamma differencing, P-36
Gauss keyword entry, U-120
Gauss’s theorem, P-34
GaussSeidel keyword entry, U-125
General window panel, U-174
genericFvPatchField library, U-99
geometric-algebraic multi-grid, U-125
GeometricBoundaryField template class, P-30
genericField<type> template class, P-30
gamma keyword entry, U-150
gGrad fvc member function, P-35
global tools, U-98
gmshToFoam utility, U-91
gnuplot keyword entry, U-115, U-185
grad fvc member function, P-35
(Grad Grad) squared, P-35
gradient, P-35, P-38
Gauss scheme, P-38
Gauss’s theorem, U-52
least square fit, U-52
least squares method, P-38, U-52
surface normal, P-38
gGradSchemes keyword, U-116
graph tools, U-98
graphFormat keyword, U-115
GuldersEGRLaminarFlameSpeed model, U-101
GuldersLaminarFlameSpeed model, U-101
H
hConstThermo model, U-101
heheuPsiThermo model, U-100
heheuPsiThermo keyword entry, U-192
Help menu, U-173
hePsiThermo model, U-100
hePsiThermo keyword entry, U-192
heRhoThermo model, U-100
heRhoThermo
  keyword entry, U-192
HerschelBulkley model, U-104
hExponentialThermo
  library, U-102
Hf keyword, U-193
hierarchical
  keyword entry, U-82, U-83
highCpCoeffs keyword, U-194
homogenousDynOneEqEddy model, U-103, U-104
homogenousDynSmagorinsky model, U-103
homogeneousMixture model, U-100
homogeneousMixture keyword, U-193
hPolynomialThermo model, U-101
interPhaseChangeDyMFoam solver, U-87
interPhaseChangeFoam solver, U-87
interfaceProperties
  library, U-104
interfaceProperties model, U-104
interFoam solver, U-87
interMixingFoam solver, U-87
internalField keyword, U-21, U-110
interpolation tools, U-98
interpolationScheme keyword, U-185
interpolations tools, U-98
interpolationSchemes keyword, U-116
inv
  tensor member function, P-23
iterations
  maximum, U-124
J
janafThermo model, U-101
jobControl
  library, U-98
jplot
    keyword entry, U-115, U-185
K
kEpsilon model, U-102, U-103
keyword
  As, U-193
  Cp, U-193
  Cv, U-193
  FoamFile, U-107
  Hf, U-193
  LESModel, U-198
  N2, U-193
  O2, U-193
  Pr, U-193
  RASModel, U-198
  Tcommon, U-194
  Thigh, U-194
  Tlow, U-194
  Ts, U-193
addLayersControls, U-150
adjustTimeStep, U-62, U-115
agglomerator, U-125
arc, U-141
blocks, U-20, U-31, U-142
block, U-141
boundaryField, U-21, U-110
boundary, U-145
boxToCell, U-60
burntProducts, U-193
cAlpha, U-63
cacheAgglomeration, U-126
castellatedMeshControls, U-150
castellatedMesh, U-150
class, U-107
cloud, U-187
commsType, U-80
convertToMeters, U-141
convertToMeters, U-140
curve, U-187
debug, U-150
defaultFieldValues, U-60
deltaT, U-114
delta, U-83, U-198
dimensions, U-21, U-110
distributed, U-83, U-84
divSchemes, U-116
doLayers, U-150
edgeGrading, U-143
edges, U-141
gmmixture, U-193
dendTime, U-22, U-113, U-114
denergy, U-192, U-196
equationOfState, U-192
errorReduction, U-158
expansionRatio, U-157
face, U-187
featureAngle, U-157
features, U-152
fieldValues, U-60
fields, U-185
fileModificationChecking, U-80
fileModificationSkew, U-80
finalLayerThickness, U-157
firstTime, U-113
floatTransfer, U-80
fluxRequired, U-116
foamChemistryFile, U-193
format, U-107
fuel, U-193
functionObjectLibs, U-179
functions, U-115, U-177
geometry, U-150
gradSchemes, U-116
graphFormat, U-115
highCpCoeffs, U-194
homogeneousMixture, U-193
inhomogeneousMixture, U-193
internalField, U-21, U-110
interpolationSchemes, U-116
interpolationScheme, U-185
laplacianSchemes, U-116
latestTime, U-39
layers, U-157
leastSquares, U-52
levels, U-154
libs, U-80, U-115
locationInMesh, U-152, U-154
location, U-107
lowCpCoeffs, U-194
manualCoeffs, U-83
maxDeltaCo, U-62
maxBoundarySkewness, U-158
maxConcave, U-158
maxCo, U-62, U-115
maxDeltaT, U-62
maxFaceThicknessRatio, U-157
maxGlobalCells, U-152
maxInternalSkewness, U-158
maxIter, U-124
maxLocalCells, U-152
maxNonOrtho, U-158
maxThicknessToMedialRatio, U-157
mergeLevels, U-126
mergePatchPairs, U-141
mergeTolerance, U-150
meshQualityControls, U-150
method, U-83
midPointAndFace, U-187
midPoint, U-187
minArea, U-158
minDeterminant, U-158
minFaceWeight, U-158
minFlatness, U-158
minMedianAxisAngle, U-157
minRefinementCells, U-152
minThickness, U-157
minTriangleTwist, U-158
minTwist, U-158
minVolRatio, U-158
minVol, U-158
mixture, U-193
mode, U-154
molWeight, U-196
multiComponentMixture, U-193
mu, U-193
nAlphaSubCycles, U-63
nBufferCellsNoExtrude, U-157
nCellsBetweenLevels, U-152
nFaces, U-132
Index

nFinestSweeps, U-126
nGrow, U-157
nLayerIter, U-157
nMoles, U-196
nPostSweeps, U-126
nPreSweeps, U-126
nRelaxIter, U-155, U-157
nRelaxedIter, U-157
nSmoothNormals, U-157
nSmoothPatch, U-155
nSmoothScale, U-158
nSmoothSurfaceNormals, U-157
nSmoothThickness, U-157
nSolveIter, U-155
neighbourPatch, U-146
numberOfSubdomains, U-83
nu, U-199
n, U-83
object, U-107
order, U-83
outputControl, U-179
oxidant, U-193
pRefCell, U-23, U-128
pRefValue, U-23, U-127
p_rhgRefCell, U-128
p_rhgRefValue, U-128
patchMap, U-166
patches, U-141
preconditioner, U-124, U-125
pressure, U-51
printCoeffs, U-42, U-198
processorWeights, U-82
processorWeights, U-83
purgeWrite, U-114
refGradient, U-138
refinementRegions, U-152, U-154
refinementSurfaces, U-152, U-153
refinementRegions, U-154
regions, U-60
relTol, U-53, U-124
relativeSizes, U-157
relaxed, U-158
resolveFeatureAngle, U-152, U-153
roots, U-83, U-84
runTimeModifiable, U-115
scotchCoeffs, U-83
setFormat, U-185
sets, U-185
simpleGrading, U-143
simulationType, U-41, U-61, U-197

singleStepReactingMixture, U-193
smoother, U-126
snGradSchemes, U-116
snapControls, U-150
snap, U-150
solvers, U-123
solver, U-53, U-123
specie, U-196
spline, U-141
startFace, U-132
startTime, U-22, U-113
stopAt, U-113
strategy, U-82, U-83
surfaceFormat, U-185
surfaces, U-185
thermoType, U-191
thermodynamics, U-196
timeFormat, U-114
timePrecision, U-115
timeScheme, U-116
tolerance, U-53, U-124, U-155
topoSetSource, U-60
traction, U-51
transport, U-192, U-196
turbulence, U-198
type, U-135, U-192
uniform, U-187
valueFraction, U-138
value, U-21, U-138
version, U-107
vertices, U-20, U-141
veryInhomogeneousMixture, U-193
writeCompression, U-114
writeControl, U-22, U-62, U-114
writeFormat, U-55, U-114
writeInterval, U-23, U-32, U-114
writePrecision, U-114
<LESModel>Coeffs, U-198
<RASModel>Coeffs, U-198
<delta>Coeffs, U-198

keyword entry
BSpline, U-142
CrankNicolson, U-122
CrossPowerLaw, U-60
DICGaussSeidel, U-125
DIC, U-125
DILU, U-125
Euler, U-122
FDIC, U-125
GAMG, U-53, U-124, U-125
Gamma, U-119
GaussSeidel, U-125
Gauss, U-120
LESModel, U-41
LES, U-41, U-197
MGridGen, U-125
MUSCL, U-119
Newtonian, U-60
PBiCG, U-124
PCG, U-124
QUICK, U-122
RASModel, U-41
RAS, U-41, U-197
SFCD, U-119, U-122
UMIST, U-117
adjustableRunTime, U-62, U-114
arc, U-142
ascii, U-114
backward, U-122
binary, U-114
blocking, U-80
bounded, U-120, U-121
cellLimited, U-120
cellPointFace, U-185
cellPoint, U-185
cell, U-185
clockTime, U-114
compressed, U-114
corrected, U-120, U-121
cpuTime, U-114
cubicCorrected, U-122
cubicCorrection, U-119
cyclic, U-137
diagonal, U-124, U-125
distance, U-154, U-187
dx, U-185
empty, U-137
faceAreaPair, U-125
faceLimited, U-120
filteredLinear2, U-119
fixed, U-114
foamFile, U-185
fourth, U-120, U-121
genral, U-114
gnuplot, U-115, U-185
hePsiThermo, U-192
heRhoThermo, U-192
heheuPsiThermo, U-192
hierarchical, U-82, U-83
inotifyMaster, U-80
inotify, U-80
inside, U-154
jplot, U-115, U-185
laminar, U-41, U-197
latestTime, U-113
leastSquares, U-120
limitedCubic, U-119
limitedLinear, U-119
limited, U-120, U-121
linearUpwind, U-119, U-122
linear, U-119, U-122
line, U-142
localEuler, U-122
manual, U-82, U-83
metis, U-83
midPoint, U-119
nextWrite, U-114
noWriteNow, U-114
nonBlocking, U-80
none, U-117, U-125
null, U-185
outputTime, U-179
outside, U-154
patch, U-137, U-186
polyLine, U-142
processor, U-137
pureMixture, U-193
raw, U-115, U-185
reactingMixture, U-193
runTime, U-32, U-114
scheduled, U-80
scientific, U-114
scotch, U-82, U-83
simple, U-82, U-83
skewLinear, U-119, U-122
smoothSolver, U-124
spline, U-142
startTime, U-22, U-113
steadyState, U-122
stl, U-185
symmetryPlane, U-137
timeStampMaster, U-80
timeStamp, U-80
timeStep, U-23, U-32, U-114, U-179
uncompressed, U-114
uncorrected, U-120, U-121
upwind, U-119, U-122
vanLeer, U-119
vtk, U-185
Index

wall, U-137
wedge, U-137
writeControl, U-114
writeInterval, U-179
writeNow, U-113
xmgr, U-115, U-185
xyz, U-187
x, U-187
y, U-187
z, U-187
kivaToFoam utility, U-91
kkLOmega model, U-102
kOmega model, U-102
kOmegaSST model, U-102, U-103
kOmegaSSTSAS model, U-103
Kronecker delta, P-19

L

lagrangian
  library, U-99
lagrangianIntermediate
    library, U-99
Lambda2 utility, U-94
LamBremhorstKE model, U-102
laminar model, U-102, U-103
laminar
  keyword entry, U-41, U-197
laminarFlameSpeedModels
  library, U-101
laplaceFilter model, U-103
Laplacian, P-36
laplacian
laplacian
  fvc member function, P-35
  fvm member function, P-35
laplacianFoam solver, U-86
laplacianSchemes keyword, U-116
latestTime
  keyword entry, U-113
latestTime keyword, U-39
LaunderGibsonRSTM model, U-102, U-103
LaunderSharmaKE model, U-102, U-103
layers keyword, U-157
leastSquares
  keyword entry, U-120
leastSquares keyword, U-52
LES
  keyword entry, U-41, U-197
LESDeltas
    library, U-103
LESFilters

library, U-103
LESModel
  keyword entry, U-41
LESModel keyword, U-198
levels keyword, U-154
libraries, U-69

library
  Chung, U-101
  FVFunctionObjects, U-98
  LESdeltas, U-103
  LESfilters, U-103
  MGridGenGAMAGglomeration, U-99
  ODE, U-99
  OSspecific, U-99
  OpenFOAM, U-98
  P1, U-101
  PV4FoamReader, U-169
  SLGThermo, U-102
  Wallis, U-101
  autoMesh, U-99
  barotropicCompressibilityModels, U-101
  basicSolidThermo, U-102
  basicThermophysicalModels, U-100
  blockMesh, U-99
  chemistryModel, U-102
  cloudFunctionObjects, U-98
  coalCombustion, U-99
  compressibleLESModels, U-104
  compressibleRASModels, U-103
  conversion, U-99
  decompositionMethods, U-99
  distributionModels, U-99
  dsmc, U-99
  dynamicFvMesh, U-99
  dynamicMesh, U-99
  edgeMesh, U-99
  engine, U-99
  fieldFunctionObjects, U-98
  fileFormats, U-99
  finiteVolume, U-98
  foamCalcFunctions, U-98
  forces, U-98
  fvDOM, U-101
  fvMotionSolvers, U-99
  genericFvPatchField, U-99
  hExponentialThermo, U-102
  incompressibleLESModels, U-103
  incompressibleRASModels, U-102
  incompressibleTransportModels, P-53, U-104
  incompressibleTurbulenceModels, P-53
interfaceProperties, U-104
jobControl, U-98
lagrangianIntermediate, U-99
lagrangian, U-99
laminarFlameSpeedModels, U-101
linear, U-101
liquidMixtureProperties, U-102
liquidProperties, U-102
meshTools, U-99
molecularMeasurements, U-99
molecule, U-99
opaqueSolid, U-101
pairPatchAgglomeration, U-99
postCalc, U-98
potential, U-99
primitive, P-21
radiationModels, U-100
randomProcesses, U-99
reactionThermophysicalModels, U-100
sampling, U-98
solidChemistryModel, U-102
solidMixtureProperties, U-102
solidParticle, U-99
solidProperties, U-102
solidSpecie, U-102
solidThermo, U-102
specie, U-101
spray, U-99
surfMesh, U-99
surfaceFilmModels, U-104
systemCall, U-99
thermophysicalFunctions, U-101
thermophysical, U-191
topoChangerFvMesh, U-99
triSurface, U-99
turbulence, U-99
twoPhaseProperties, U-104
utilityFunctionObjects, U-99
viewFactor, U-101
vtkPV4Foam, U-169
libs keyword, U-80, U-115
lid-driven cavity flow, U-17
LienCubicKE model, U-102
LienCubicKELowRe model, U-102
LienLeschzinerLowRe model, U-102
Lights button, U-174
limited
limitLinear
limitedLinear
keyword entry, U-119
line
keyword entry, U-142
Line Style menu, U-35
linear
library, U-101
linear model, U-195
linear
keyword entry, U-119, U-122
linearUpwind
keyword entry, U-119, U-122
liquid
electrically-conducting, P-67
liquidMixtureProperties
library, U-102
liquidProperties
library, U-102
lists, P-27
List<
Type>
template class, P-27
localEuler
keyword entry, U-122
location
keyword, U-107
locationInMesh keyword, U-152, U-154
lowCpCoeffs keyword, U-194
lowReOneEqEddy model, U-104
LRDDiffStress model, U-103
LRR model, U-102, U-103
lsGrad
fvc member function, P-35

M
Mach utility, U-94
mag
tensor member function, P-23
magneticFoam solver, U-89
magnetohydrodynamics, P-67
magSqr
tensor member function, P-23
Make
directory, U-73
make script/alias, U-71
Make/files file, U-75
manual
keyword entry, U-82, U-83
manualCoeffs keyword, U-83
mapFields utility, U-31, U-38, U-42, U-56, U-90, U-165
mapFieldsPar utility, U-90
mapping
fields, U-165
Marker Style menu, U-35
matrices tools, U-98
max
  tensor member function, P-23
maxAlphaCo keyword, U-62
maxBoundarySkewness keyword, U-158
maxCo keyword, U-62, U-115
maxConcave keyword, U-158
maxDeltaT keyword, U-62
maxDeltaxyz model, U-103
maxFaceThicknessRatio keyword, U-157
maxGlobalCells keyword, U-152
maximum iterations, U-124
maxInternalSkewness keyword, U-158
maxIter keyword, U-124
maxLocalCells keyword, U-152
maxNonOrtho keyword, U-158
maxThicknessToMedialRatio keyword, U-157
mdEquilibrationFoam solver, U-89
mdFoam solver, U-89
mdInitialise utility, U-90
mechanicalProperties
dictionary, U-51
memory tools, U-98
menu
  Color By, U-173
  Current Time Controls, U-25, U-171
  Edit, U-174
  Help, U-173
  Line Style, U-35
  Marker Style, U-35
  VCR Controls, U-25, U-171
  View, U-170, U-173
menu entry
  Plot Over Line, U-34
  Save Animation, U-175
  Save Screenshot, U-175
  Settings, U-174
  Solid Color, U-173
  Toolbars, U-173
  View Settings, U-24, U-173
  Wireframe, U-173
  fvSchemes, U-52
mergeLevels keyword, U-126
mergeMeshes utility, U-92
mergeOrSplitBaffles utility, U-92
mergePatchPairs keyword, U-141
mergeTolerance keyword, U-150
mesh
  1-dimensional, U-133
  2D, U-133
  axi-symmetric, U-133
  basic, P-29
  block structured, U-140
  decomposition, U-82
  description, U-129
  finite volume, P-29
  generation, U-140, U-149
  grading, U-140, U-143
  grading, example of, P-50
  non-orthogonal, P-43
  refinement, P-61
  resolution, U-29
  specification, U-129
  split-hex, U-149
  Stereolithography (STL), U-149
  surface, U-149
  validity constraints, U-129
Mesh Parts window panel, U-24
meshes tools, U-98
meshQualityControls keyword, U-150
meshTools
  library, U-99
message passing interface
  openMPI, U-84
method keyword, U-83
metis
  keyword entry, U-83
  metisDecomp model, U-100
MGridGenGAMGAgglomeration
  library, U-99
MGridGen
  keyword entry, U-125
mhdFoam solver, P-69, U-89
midPoint
  keyword entry, U-119
midPoint keyword, U-187
midPointAndFace keyword, U-187
min
  tensor member function, P-23
minArea keyword, U-158
minDeterminant keyword, U-158
minFaceWeight keyword, U-158
minFlatness keyword, U-158
minMedianAxisAngle keyword, U-157
MINMOD differencing, P-36
minRefinementCells keyword, U-152
minThickness keyword, U-157
minTriangleTwist keyword, U-158
Index

minTwist keyword, U-158
minVol keyword, U-158
minVolRatio keyword, U-158
mirrorMesh utility, U-92
mixed
  boundary condition, U-138
mixedSmagorinsky model, U-103
mixture keyword, U-193
mixtureAdiabaticFlameT utility, U-97
mode keyword, U-154
model
  APIfunctions, U-102
  BirdCarreau, U-104
  CrossPowerLaw, U-104
  DeardorffDiffStress, U-103, U-104
  GuldersEGRLaminarFlameSpeed, U-101
  GuldersLaminarFlameSpeed, U-101
  HerschelBulkley, U-104
  LRDiffStress, U-103
  LRR, U-102, U-103
  LamBremhorstKE, U-102
  LaunderGibsonRSTM, U-102, U-103
  LaunderSharmaKE, U-102, U-103
  LienCubicKELowRe, U-102
  LienCubicKE, U-102
  LienLeschzinerLowRe, U-102
  NSRDSfunctions, U-102
  Newtonian, U-104
  NonlinearKEShih, U-102
  PengRobinsonGas, U-195
  PrandtlDelta, U-103
  RNGkEpsilon, U-102, U-103
  RaviPetersen, U-101
  Smagorinsky2, U-103
  Smagorinsky, U-103, U-104
  SpalartAllmarasDDES, U-104
  SpalartAllmarasDDDES, U-104
  SpalartAllmaras, U-102-U-104
  adiabaticPerfectFluid, U-101, U-195
  anisotropicFilter, U-103
  basicMultiComponentMixture, U-100
  chemistryModel, U-102
  chemistrySolver, U-102
  constTransport, U-101
  constant, U-101
  cubeRootVolDelta, U-103
  decompose, U-100
  distributed, U-100
  dynLagrangian, U-103
  dynOneEqEddy, U-103
eConstThermo, U-101
egrMixture, U-100
hConstThermo, U-101
hPolynomialThermo, U-101
hePsiThermo, U-100
heRhoThermo, U-100
heheuPsiThermo, U-100
homogenousDynOneEqEddy, U-103, U-104
homogenousDynSmagorinsky, U-103
homogeneousMixture, U-100
icoPolynomial, U-101, U-195
incompressiblePerfectGas, U-101, U-195
inhomogeneousMixture, U-100
interfaceProperties, U-104
janafThermo, U-101
kEpsilon, U-102, U-103
kOmegaSSTSAS, U-103
kOmegaSST, U-102, U-103
kOmega, U-102
kkLOmega, U-102
laminar, U-102, U-103
laplaceFilter, U-103
linear, U-195
lowReOneEqEddy, U-104
maxDeltaxyz, U-103
metisDecomp, U-100
mixedSmagorinsky, U-103
multiComponentMixture, U-100
multiphaseMixtureThermo, U-192
oneEqEddy, U-103, U-104
perfectFluid, U-101, U-195
perfectGas, U-195
polynomialTransport, U-101
powerLaw, U-104
psiReactionThermo, U-100, U-192
psiThermo, U-192
psiuReactionThermo, U-100, U-192
ptsotchDecomp, U-100
pureMixture, U-100
qZeta, U-102
reactingMixture, U-100
realizableKE, U-102, U-103
reconstruct, U-100
rhoConst, U-101, U-195
rhoReactionThermo, U-100, U-192
rhoThermo, U-192
scaleSimilarity, U-103
scotchDecomp, U-100
simpleFilter, U-103
singleStepReactingMixture, U-100

OpenFOAM-3.0.1
Index

smoothDelta, U-103

cpecieThermo, U-101

spectEddyVisc, U-103

sutherlandTransport, U-101

v2f, U-103

vanDriestDelta, U-104

veryInhomogeneousMixture, U-100

modifyMesh utility, U-93

molecularMeasurements library, U-99

molecule library, U-99

molWeight keyword, U-196

moveDynamicMesh utility, U-92

moveEngineMesh utility, U-92

moveMesh utility, U-92

movingWallVelocity boundary condition, U-139

MPI

openMPI, U-84

mshToFoam utility, U-91

mu keyword, U-193

multiComponentMixture model, U-100

multiComponentMixture keyword, U-193

multigrid geometric-algebraic, U-125

multiphaseEulerFoam solver, U-87

multiphaseInterFoam solver, U-87

multiphaseMixtureThermo model, U-192

MUSCL keyword entry, U-119

n keyword, U-83

N2 keyword, U-193

nabla operator, P-25

nAlphaSubCycles keyword, U-63

nBufferCellsNoExtrude keyword, U-157

nCellsBetweenLevels keyword, U-152

neighbour dictionary, U-132

neighbourPatch keyword, U-146

netgenNeutralToFoam utility, U-91

Newtonian keyword entry, U-60

Newtonian model, U-104

nextWrite keyword entry, U-114

nFaces keyword, U-132

nFinestSweeps keyword, U-126

gGrow keyword, U-157

nLayerIter keyword, U-157

nMoles keyword, U-196

non-orthogonal mesh, P-43

nonBlocking keyword entry, U-80

none keyword entry, U-117, U-125

NonlinearKEShih model, U-102

nonNewtonianIcoFoam solver, U-86

noWriteNow keyword entry, U-114

nPostSweeps keyword, U-126

nPreSweeps keyword, U-126

nRelaxedIter keyword, U-157

nRelaxIter keyword, U-155, U-157

nSmoothNormals keyword, U-157

nSmoothPatch keyword, U-155

nSmoothScale keyword, U-158

nSmoothSurfaceNormals keyword, U-157

nSmoothThickness keyword, U-157

nSolveIter keyword, U-155

NSRDSfunctions model, U-102

nu keyword, U-199

null keyword entry, U-185

numberOfSubdomains keyword, U-83

O

O2 keyword, U-193

object keyword, U-107

objToVTK utility, U-92

ODE

library, U-99

oneEqEddy model, U-103, U-104

Opacity text box, U-173

opaqueSolid library, U-101

OpenFOAM applications, U-69

file format, U-106

libraries, U-69

OpenFOAM library, U-98

OpenFOAM file syntax //, U-106

openMPI message passing interface, U-84

MPI, U-84

operator scalar, P-26
vector, P-25
Options window, U-174
options file, U-73
order keyword, U-83
Orientation Axes button, U-24
orientFaceZone utility, U-92
OSspecific library, U-99
outer product, see tensor, outer product
outlet boundary condition, P-69
outletInlet boundary condition, U-139
outputControl keyword, U-179
outputTime keyword entry, U-179
outside keyword entry, U-154
owner dictionary, U-131
oxidant keyword, U-193

P

p field, U-23
P1
library, U-101
p_rhgRefCell keyword, U-128
p_rhgRefValue keyword, U-128
pairPatchAgglomeration library, U-99
paraFoam, U-23, U-169
parallel running, U-81
Paramters window panel, U-171
partialSlip boundary condition, U-139
particleTracks utility, U-95
patch boundary condition, U-136
patch keyword entry, U-137, U-186
patchAverage utility, U-94
patches keyword, U-141
patchIntegrate utility, U-94
patchMap keyword, U-166
patchSummary utility, U-97
PBICG keyword entry, U-124
PCG keyword entry, U-124
pdfPlot utility, U-95
PDRFoam solver, U-88
PDRMesh utility, U-93
Pe utility, U-94
PengRobinsonGas model, U-195
perfectFluid model, U-101, U-195
perfectGas model, U-195
permutation symbol, P-18
pimpleFoam solver, U-86
Pipeline Browser window, U-24, U-170
PISO
dictionary, U-23
pisoFoam solver, U-17, U-86
Plot Over Line menu entry, U-34
plot3dToFoam utility, U-91
pointField class, P-29
pointField<Type> template class, P-31
points
dictionary, U-131, U-140
polyBoundaryMesh class, P-29
copolyDualMesh utility, U-92
cpolyLine
keyword entry, U-142
cpolyMesh directory, U-105, U-131
cpolyMesh class, P-29, U-129, U-131
cpolynomialTransport model, U-101
cpolyPatch class, P-29
cpolyPatchList class, P-29
cpost-processing, U-169
cpost-processing
paraFoam, U-169
cpostCalc
library, U-98
cpostChannel utility, U-95
cpotential
library, U-99
cpotentialFreeSurfaceFoam solver, U-87
cpotentialFoam solver, P-44, U-86
cpow
tensor member function, P-23
cpowerLaw model, U-104
cpPrime2 utility, U-94
cPr keyword, U-193
cPrandtlDelta model, U-103
cpreconditioner keyword, U-124, U-125
cpRefCell keyword, U-23, U-128
cpRefValue keyword, U-23, U-127
cpressure keyword, U-51
cpressure waves
in liquids, P-62
pressureDirectedInletVelocity
  boundary condition, U-139
pressureInletVelocity
  boundary condition, U-139
pressureOutlet
  boundary condition, P-63
pressureTransmissive
  boundary condition, U-139
primitive
  library, P-21
primitives tools, U-98
printCoeffs keyword, U-42, U-198
processorWeights keyword, U-82
probeLocations utility, U-95
process
  background, U-24, U-81
  foreground, U-24
processor
  boundary condition, U-138
processorN directory, U-83
processorWeights keyword, U-83
Properties window, U-171, U-172
Properties window panel, U-25, U-170
psiReactionThermo model, U-100, U-192
psiThermo model, U-192
psiuReactionThermo model, U-100, U-192
ptot utility, U-95
ptsotchDecomp model, U-100
pureMixture model, U-100
pureMixture
  keyword entry, U-193
purgeWrite keyword, U-114
PV4FoamReader
  library, U-169
Q utility, U-94
QUICK
  keyword entry, U-122
qZeta model, U-102
R utility, U-94
radiationModels
  library, U-100
randomProcesses
  library, U-99
RAS
  keyword entry, U-41, U-197
RASModel
  keyword entry, U-41
RASModel keyword, U-198
RaviPetersen model, U-101
raw
  keyword entry, U-115, U-185
reactingEulerFoam solver, U-88
reactingFoam solver, U-88
reactingMixture model, U-100
reactingMixture
  keyword entry, U-193
reactingParcelFilmFoam solver, U-89
reactingParcelFoam solver, U-89
reactionThermophysicalModels library, U-100
realizableKE model, U-102, U-103
reconstruct model, U-100
reconstructPar utility, U-85, U-97
reconstructParMesh utility, U-97
redistributePar utility, U-97
refGradient keyword, U-138
refineHexMesh utility, U-93
refinementRegions keyword, U-154
refinementLevel utility, U-93
refinementRegions keyword, U-152, U-154
refinementSurfaces keyword, U-152, U-153
refineMesh utility, U-92
refineWallLayer utility, U-93
Refresh Times button, U-25, U-171
regions keyword, U-60
relative tolerance, U-124
relativeSizes keyword, U-157
relaxed keyword, U-158
relTol keyword, U-53, U-124
removeFaces utility, U-93
Render View window, U-174
Render View window panel, U-173, U-174
renumberMesh utility, U-92
Rescale to Data Range button, U-25
Reset button, U-170
resolveFeatureAngle keyword, U-152, U-153
restart, U-39
Reynolds number, U-17, U-21
rhoPorousSimpleFoam solver, U-86
rhoReactingBuoyantFoam solver, U-88
rhoCentralDyMFoam solver, U-86
rhoCentralFoam solver, U-86
rhoConst model, U-101, U-195
rhoPimpleFoam solver, U-86
rhoReactingFoam solver, U-88

OpenFOAM-3.0.1
rhoReactionThermo model, U-100, U-192
rhoSimpleFoam solver, U-86
rhoSimplecFoam solver, U-86
rhoThermo model, U-192
rmdepall script/alias, U-77
RNGkEpsilon model, U-102, U-103
roots keyword, U-83, U-84
rotateMesh utility, U-92
run
parallel, U-81
run directory, U-105
runTime
keyword entry, U-32, U-114
runTimeModifiable keyword, U-115

S
sammToFoam utility, U-91
sample utility, U-95, U-184
sampling
library, U-98
Save Animation
menu entry, U-175
Save Screenshot
menu entry, U-175
scalar, P-14
operator, P-26
scalar class, P-22
scalarField class, P-27
scalarTransportFoam solver, U-86
scale
tensor member function, P-23
scalePoints utility, U-162
scaleSimilarity model, U-103
scheduled
keyword entry, U-80
scientific
keyword entry, U-114
scotch
keyword entry, U-82, U-83
scotchCoeffs keyword, U-83
scotchDecomp model, U-100
script/alias
find, U-179
foamCorrectVrt, U-164
foamJob, U-188
foamLog, U-188
make, U-71
rmdepall, U-77
wclean, U-76
wmake, U-71
second time derivative, P-35
Seed window, U-175
selectCells utility, U-93
Set Ambient Color button, U-173
setFields utility, U-60, U-90
setFormat keyword, U-185
sets keyword, U-185
setSet utility, U-92
setsToZones utility, U-92
Settings
menu entry, U-174
SFCD
keyword entry, U-119, U-122
shallowWaterFoam solver, U-86
shape, U-143
SI units, U-110
simple
keyword entry, U-82, U-83
simpleFilter model, U-103
simpleFoam solver, P-53, U-86
simpleGrading keyword, U-143
simulationType keyword, U-41, U-61, U-197
singleCellMesh utility, U-92
singleStepReactingMixture model, U-100
singleStepReactingMixture keyword, U-193
skew
tensor member function, P-23
skewLinear
keyword entry, U-119, U-122
SLGThermo
library, U-102
slice class, P-29
slip
boundary condition, U-139
Smagorinsky model, U-103, U-104
Smagorinsky2 model, U-103
smapToFoam utility, U-93
smoothDelta model, U-103
smoother keyword, U-126
smoothSolver
keyword entry, U-124
snap keyword, U-150
snapControls keyword, U-150
snappyHexMesh utility
background mesh, U-151
cell removal, U-154
cell splitting, U-152
mesh layers, U-155
meshing process, U-149
snapping to surfaces, U-155
snappyHexMesh utility, U-91, U-149
\texttt{snappyHexMeshDict} file, U-150
\texttt{snGrad}
  \texttt{fvc} member function, P-35
\texttt{snGradCorrection}
  \texttt{fvc} member function, P-35
\texttt{snGradSchemes} keyword, U-116
\texttt{Solid Color}
  menu entry, U-173
\texttt{solidChemistryModel}
  library, U-102
\texttt{solidDisplacementFoam} solver, U-89
\texttt{solidDisplacementFoam} solver, U-51
\texttt{solidEquilibriumDisplacementFoam} solver, U-89
\texttt{solidMixtureProperties}
  library, U-102
\texttt{solidParticle}
  library, U-99
\texttt{solidProperties}
  library, U-102
\texttt{solidSpecie}
  library, U-102
\texttt{solidThermo}
  library, U-102
\texttt{solver}
  \texttt{DPMFoam}, U-89
  \texttt{PDRFoam}, U-88
  \texttt{XiFoam}, U-88
  \texttt{adjointShapeOptimizationFoam}, U-86
  \texttt{blockMesh}, P-45
  \texttt{boundaryFoam}, U-86
  \texttt{buoyantBoussinesqPimpleFoam}, U-88
  \texttt{buoyantBoussinesqSimpleFoam}, U-88
  \texttt{buoyantPimpleFoam}, U-88
  \texttt{buoyantSimpleFoam}, U-88
  \texttt{cavitatingDyMFoam}, U-87
  \texttt{cavitatingFoam}, U-87
  \texttt{chemFoam}, U-88
  \texttt{chtMultiRegionFoam}, U-88
  \texttt{chtMultiRegionSimpleFoam}, U-88
  \texttt{coalChemistryFoam}, U-89
  \texttt{coldEngineFoam}, U-88
  \texttt{compressibleInterDyMFoam}, U-87
  \texttt{compressibleInterFoam}, U-87
  \texttt{compressibleMultiphaseInterFoam}, U-87
  \texttt{dnsFoam}, U-88
  \texttt{driftFluxFoam}, U-87
  \texttt{dsmcFoam}, U-89
  \texttt{electrostaticFoam}, U-89
  \texttt{engineFoam}, U-88
  \texttt{financialFoam}, U-90
  \texttt{fireFoam}, U-88
  \texttt{icoFoam}, U-17, U-21, U-22, U-24, U-86
  \texttt{icoUncoupledKinematicParcelFoam}, U-89
  \texttt{interFoam}, U-87
  \texttt{interMixingFoam}, U-87
  \texttt{interPhaseChangeDyMFoam}, U-87
  \texttt{interPhaseChangeFoam}, U-87
  \texttt{laplacianFoam}, U-86
  \texttt{magneticFoam}, U-89
  \texttt{mdEquilibrationFoam}, U-89
  \texttt{mdFoam}, U-89
  \texttt{mhdFoam}, P-69, U-89
  \texttt{multiphaseEulerFoam}, U-87
  \texttt{multiphaselnterFoam}, U-87
  \texttt{nonNewtonianIcoFoam}, U-86
  \texttt{pimpleFoam}, U-86
  \texttt{pisoFoam}, U-17, U-86
  \texttt{potentialFoam}, P-44, U-86
  \texttt{potentialFreeSurfaceFoam}, U-87
  \texttt{reactingEulerFoam}, U-88
  \texttt{reactingFoam}, U-88
  \texttt{reactingParcelFilmFoam}, U-89
  \texttt{reactingParcelFoam}, U-89
  \texttt{rhoCentralDyMFoam}, U-86
  \texttt{rhoCentralFoam}, U-86
  \texttt{rhoPimpleFoam}, U-86
  \texttt{rhoReactingFoam}, U-88
  \texttt{rhoSimpleFoam}, U-86
  \texttt{rhoSimplecFoam}, U-86
  \texttt{rhoPorousSimpleFoam}, U-86
  \texttt{rhoReactingBuoyantFoam}, U-88
  \texttt{scalarTransportFoam}, U-86
  \texttt{shallowWaterFoam}, U-86
  \texttt{simpleFoam}, P-53, U-86
  \texttt{solidDisplacementFoam}, U-89
  \texttt{solidDisplacementFoam}, U-51
  \texttt{solidEquilibriumDisplacementFoam}, U-89
  \texttt{solidEulerFoam}, U-89
  \texttt{sonicDyMFoam}, U-86
  \texttt{sonicFoam}, P-59, U-87
  \texttt{sonicLiquidFoam}, P-63, U-87
  \texttt{sprayFoam}, U-89
  \texttt{thermoFoam}, U-88
  \texttt{twoLiquidMixingFoam}, U-88
  \texttt{twoPhaseEulerFoam}, U-88
  \texttt{uncoupledKinematicParcelFoam}, U-89
  \texttt{solver} keyword, U-53, U-123
  \texttt{solver relative tolerance}, U-124
  \texttt{solver tolerance}, U-124
  \texttt{solvers} keyword, U-123
  \texttt{sonicDyMFoam} solver, U-86

OpenFOAM-3.0.1
sonicFoam solver, P-59, U-87
sonicLiquidFoam solver, P-63, U-87
source, P-35
SpalartAllmaras model, U-102–U-104
SpalartAllmarasDDES model, U-104
SpalartAllmarasIDDES model, U-104
specie library, U-101
specie keyword, U-196
specieThermo model, U-101
spectEddyVisc model, U-103
spline
  keyword entry, U-142
spline keyword, U-141
splitCells utility, U-93
splitMesh utility, U-92
splitMeshRegions utility, U-92
spray
  library, U-99
sprayFoam solver, U-89
sqr
  tensor member function, P-23
sqrGradGrad
    fvc member function, P-35
start3ToFoam utility, U-91
start4ToFoam utility, U-91
startFace keyword, U-132
startFrom keyword, U-22, U-113
startToFoam utility, U-159
startTime
  keyword entry, U-22, U-113
startTime keyword, U-22, U-113
steady flow
turbulent, P-50
steadyParticleTracks utility, U-95
steadyState
  keyword entry, U-122
Stereolithography (STL), U-149
stitchMesh utility, U-92
stl
  keyword entry, U-185
stopAt keyword, U-113
strategy keyword, U-82, U-83
streamFunction utility, U-94
stress analysis of plate with hole, U-46
stressComponents utility, U-94
Style window panel, U-173
Su
  fvm member function, P-35
subsetMesh utility, U-93
summation convention, P-15
SUPERBEE differencing, P-36
supersonic flow, P-58
supersonic flow over forward step, P-58
supersonicFreeStream boundary condition, U-139
surfaceLambdaMuSmooth utility, U-96
surface mesh, U-149
surfaceAdd utility, U-95
surfaceAutoPatch utility, U-95
surfaceBooleanFeatures utility, U-95
surfaceCheck utility, U-95
surfaceClean utility, U-95
surfaceCoarsen utility, U-95
surfaceConvert utility, U-95
surfaceFeatureConvert utility, U-95
surfaceFeatureExtract utility, U-95, U-153
surfaceField<Type> template class, P-31
surfaceFilmModels
  library, U-104
surfaceFind utility, U-95
surfaceFormat keyword, U-185
surfaceHookUp utility, U-95
surfaceInertia utility, U-96
surfaceMesh tools, U-98
surfaceMeshConvert utility, U-96
surfaceMeshConvertTesting utility, U-96
surfaceMeshExport utility, U-96
surfaceMeshImport utility, U-96
surfaceMeshInfo utility, U-96
surfaceMeshTriangulate utility, U-96
surfaceNormalFixedValue boundary condition, U-139
surfaceOrient utility, U-96
surfacePointMerge utility, U-96
surfaceRedistributePar utility, U-96
surfaceRefineRedGreen utility, U-96
surfaces keyword, U-185
surfaceSplitByPatch utility, U-96
surfaceSplitByTopology utility, U-96
surfaceSplitNonManifolds utility, U-96
surfaceSubset utility, U-96
surfaceToPatch utility, U-96
surfaceTransformPoints utility, U-96
surfMesh
  library, U-99
SuSp
  fvm member function, P-35
sutherlandTransport model, U-101
symm

OpenFOAM-3.0.1
tensor member function, P-23

symmetryPlane
  boundary condition, P-63, U-137

symmetryPlane
  keyword entry, U-137

symmTensorField class, P-27

symmTensorThirdField class, P-27

system directory, P-48, U-105

systemCall
  library, U-99

T

tensor member function, P-23

Tcommon keyword, U-194

template class
  GeometricBoundaryField, P-30
  fvMatrix, P-34
  dimensioned<Type>, P-24
  FieldField<Type>, P-30
  Field<Type>, P-27
  geometricField<Type>, P-30
  List<Type>, P-27
  pointField<Type>, P-31
  surfaceField<Type>, P-31
  volField<Type>, P-31

temporal discretisation, P-40
  Crank Nicolson, P-41
  Euler implicit, P-40
  explicit, P-40
  in OpenFOAM, P-41

temporalInterpolate utility, U-95

tensor, P-13
  addition, P-16
  algebraic operations, P-16
  algebraic operations in OpenFOAM, P-22
  antisymmetric, see tensor, skew calculus, P-25
  classes in OpenFOAM, P-21
  cofactors, P-20
  component average, P-18
  component maximum, P-18
  component minimum, P-18
  determinants, P-20
  deviatoric, P-20
  diagonal, P-20
  dimension, P-14
  double inner product, P-17
  geometric transformation, P-19
  Hodge dual, P-21
  hydrostatic, P-20
identities, P-19
  identity, P-19
  inner product, P-16
  inverse, P-21
  magnitude, P-18
  magnitude squared, P-18
  mathematics, P-13
  notation, P-15
  nth power, P-18
  outer product, P-17
  rank, P-14
  rank 3, P-15
  scalar division, P-16
  scalar multiplication, P-16
  scale function, P-18
  second rank, P-14
  skew, P-20
  square of, P-18
  subtraction, P-16
  symmetric, P-20
  symmetric rank 2, P-14
  symmetric rank 3, P-15
  trace, P-20
  transformation, P-19
  transpose, P-14, P-20
  triple inner product, P-17
  vector cross product, P-18

tensor class, P-22

tensor member function
  *, P-23
  +, P-23
  -, P-23
  /, P-23
  &, P-23
  &&, P-23
  ^, P-23
  cmptAv, P-23
  cofactors, P-23
  det, P-23
  dev, P-23
  diag, P-23
  I, P-23
  inv, P-23
  mag, P-23
  magSqr, P-23
  max, P-23
  min, P-23
  pow, P-23
  scale, P-23
  skew, P-23

OpenFOAM-3.0.1
boundary condition, U-139

tutorials
  breaking of a dam, U-56
  lid-driven cavity flow, U-17
  stress analysis of plate with hole, U-46

tutorials directory, P-43, U-17
twoLiquidMixingFoam solver, U-88
twoPhaseEulerFoam solver, U-88
twoPhaseProperties library, U-104

type keyword, U-135, U-192

U field, U-23

Ucomponents utility, P-70

UNIST
  keyword entry, U-117

uncompressed
  keyword entry, U-114

uncorrected
  keyword entry, U-120, U-121

uncoupledKinematicParcelFoam solver, U-89

uniform keyword, U-187

units
  base, U-110
  of measurement, P-24, U-109
  S.I. base, P-24
  SI, U-110
  Syst`ème International, U-110
  United States Customary System, U-110
  USCS, U-110

Update GUI button, U-171

uprime utility, U-94

upwind
  keyword entry, U-119, U-122

upwind differencing, P-36, U-63

USCS units, U-110

Use Parallel Projection button, U-24

utility
  Co, U-93
  Lambda2, U-94
  Mach, U-94
  PDRMesh, U-93
  Pe, U-94
  Q, U-94
  R, U-94
  Ucomponents, P-70
  adiabaticFlameT, U-97
  ansysToFoam, U-91
  applyBoundaryLayer, U-90
  applyWallFunctionBoundaryConditions, U-90

attachMesh, U-92
autoPatch, U-92
autoRefineMesh, U-93
blockMesh, U-38, U-91, U-140
boxTurb, U-90
cfx4ToFoam, U-91, U-159
changeDictionary, U-90
checkMesh, U-92, U-160
chemkinToFoam, U-97
collapseEdges, U-93
combinePatchFaces, U-93
createBaffles, U-92
createPatch, U-92
createTurbulenceFields, U-94
createExternalCoupledPatchGeometry, U-90
datToFoam, U-91
decomposePar, U-82, U-83, U-97
deformedGeom, U-92
dsmcFieldsCalc, U-95
dsmcInitialise, U-90
gineCompRatio, U-95
gineSwirl, U-90
enSight74FoamExec, U-183
enSightFoamReader, U-93
enstrophy, U-93
equilibriumCO, U-97
equilibriumFlameT, U-97
execFlowFunctionObjects, U-95
expandDictionary, U-97
extrude2DMesh, U-91
extrudeMesh, U-91
extrudeToRegionMesh, U-91
faceAgglomerate, U-90
flattenMesh, U-92
flowType, U-93
fluent3DMeshToFoam, U-91
fluentMeshToFoam, U-91, U-159
foamCalc, U-33
foamDataToFluent, U-93, U-181
foamDebugSwitches, U-97
foamFormatConvert, U-97
foamHelp, U-97
foamInfoExec, U-97
foamListTimes, U-95
foamMeshToFoam, U-91, U-159
foamToFluent, U-93, U-181
foamToEnsightParts, U-93
foamToEnsight, U-93
foamToGMV, U-93
foamToStarMesh, U-91
foamToSurface, U-91
foamToTecplot360, U-93
foamToTetDualMesh, U-93
foamToVTK, U-93
foamUpgradeCyclics, U-90
foamUpgradeFvSolution, U-90
foamyHexMesh, U-91
foamyQuadMesh, U-91
foamyHexMeshBackgroundMesh, U-91
foamyHexMeshSurfaceSimplify, U-91
gambitToFoam, U-91, U-159
gmshToFoam, U-91
ideasToFoam, U-159
ideasUnvToFoam, U-91
insideCells, U-92
kivaToFoam, U-91
mapFieldsPar, U-90
mapFields, U-31, U-38, U-42, U-56, U-90, U-165
mdInitialise, U-90
mergeMeshes, U-92
mergeOrSplitBaffles, U-92
mirrorMesh, U-92
mixtureAdiabaticFlameT, U-97
modifyMesh, U-93
moveDynamicMesh, U-92
moveEngineMesh, U-92
moveMesh, U-92
mshToFoam, U-91
netgenNeutralToFoam, U-91
objToVTK, U-92
orientFaceZone, U-92
pPrime2, U-94
particleTracks, U-95
patchAverage, U-94
patchIntegrate, U-94
patchSummary, U-97
pdfPlot, U-95
plot3dToFoam, U-91
polyDualMesh, U-92
postChannel, U-95
probeLocations, U-95
ptot, U-95
reconstructParMesh, U-97
reconstructPar, U-85, U-97
redistributePar, U-97
refineHexMesh, U-93
refineMesh, U-92
refineWallLayer, U-93
refinementLevel, U-93
removeFaces, U-93
renumberMesh, U-92
rotateMesh, U-92
sammToFoam, U-91
sample, U-95, U-184
scalePoints, U-162
selectCells, U-93
setFields, U-60, U-90
setSet, U-92
setsToZones, U-92
singleCellMesh, U-92
smapToFoam, U-93
snappyHexMesh, U-91, U-149
splitCells, U-93
splitMeshRegions, U-92
splitMesh, U-92
star3ToFoam, U-91
star4ToFoam, U-91
starToFoam, U-159
steadyParticleTracks, U-95
stitchMesh, U-92
streamFunction, U-94
stressComponents, U-94
subsetMesh, U-93
surfaceLambdaMuSmooth, U-96
surfaceAdd, U-95
surfaceAutoPatch, U-95
surfaceBooleanFeatures, U-95
surfaceCheck, U-95
surfaceClean, U-95
surfaceCoarsen, U-95
surfaceConvert, U-95
surfaceFeatureConvert, U-95
surfaceFeatureExtract, U-95, U-153
surfaceFind, U-95
surfaceHookUp, U-95
surfaceInertia, U-96
surfaceMeshConvertTesting, U-96
surfaceMeshConvert, U-96
surfaceMeshExport, U-96
surfaceMeshImport, U-96
surfaceMeshInfo, U-96
surfaceMeshTriangulate, U-96
surfaceOrient, U-96
surfacePointMerge, U-96
surfaceRedistributeMerge, U-96
surfaceRedistributePar, U-96
surfaceRefineRedGreen, U-96
surfaceSplitByPatch, U-96
surfaceSplitByTopology, U-96
surfaceSplitNonManifolds, U-96
surfaceSubset, U-96
surfaceToPatch, U-96
surfaceTransformPoints, U-96
temporalInterpolate, U-95
tetgenToFoam, U-91
topoSet, U-93
transformPoints, U-93
uprime, U-94
viewFactorsGen, U-90
vorticity, U-94
vtkUnstructuredToFoam utility, U-92
wallFunctionTable, U-90
wallGradU, U-94
wallHeatFlux utility, U-94
wallShearStress utility, U-94
Wallis library, U-101
wallShearStress utility, U-94
wdot utility, U-95
wedge boundary condition, U-133, U-137, U-148
wedge keyword entry, U-137
window
  Color Legend, U-29
  Options, U-174
  Pipeline Browser, U-24, U-170
  Properties, U-171, U-172
  Render View, U-174
  Seed, U-175
window panel
  Animations, U-174
  Annotation, U-24
  Charts, U-174
  Color Legend, U-173
  Color Scale, U-173
  Colors, U-174
  Display, U-24, U-25, U-170, U-172
  General, U-174
  Information, U-170
  Mesh Parts, U-24
  Parameters, U-171
  Properties, U-25, U-170
  Render View, U-173, U-174
  Style, U-173
  View Render, U-24
Wireframe menu entry, U-173
WM_ARCH
environment variable, U-76
WM_ARCH_OPTION
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_MPLIB
environment variable, U-76
WM_OPTIONS
environment variable, U-76
WM_PRECISION_OPTION
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_USER_DIR
environment variable, U-76
WM_PROJECT_VERSION
environment variable, U-76
wmake
   platforms, U-73
wmake script/alias, U-71

word class, P-24, P-29
writeCellCentres utility, U-95
writeCompression keyword, U-114
writeControl
   keyword entry, U-114
writeControl keyword, U-22, U-62, U-114
writeFormat keyword, U-55, U-114
writeInterval
   keyword entry, U-179
writeInterval keyword, U-23, U-32, U-114
writeMeshObj utility, U-92
writeNow
   keyword entry, U-113
writePrecision keyword, U-114

X
x
   keyword entry, U-187
XiFoam solver, U-88
xmgr
   keyword entry, U-115, U-185
xyz
   keyword entry, U-187

Y
y
   keyword entry, U-187
yPlus utility, U-94

Z
z
   keyword entry, U-187
zeroGradient
   boundary condition, U-138
zipUpMesh utility, U-93