The open source CFD toolbox



## User Guide

Contents

# 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.

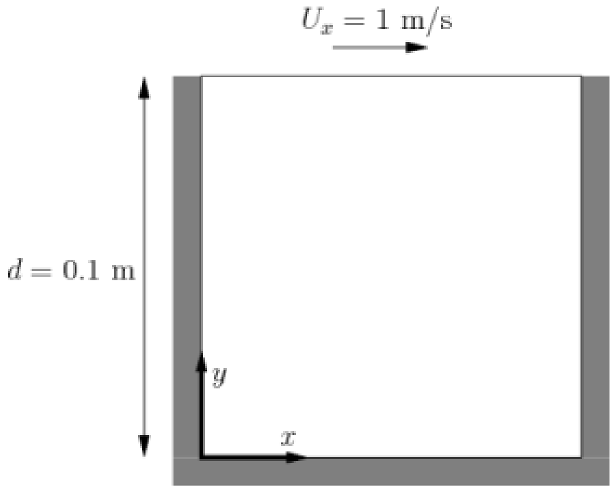



Figure 2.1: Geometry of the lid driven cavity.

## 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.

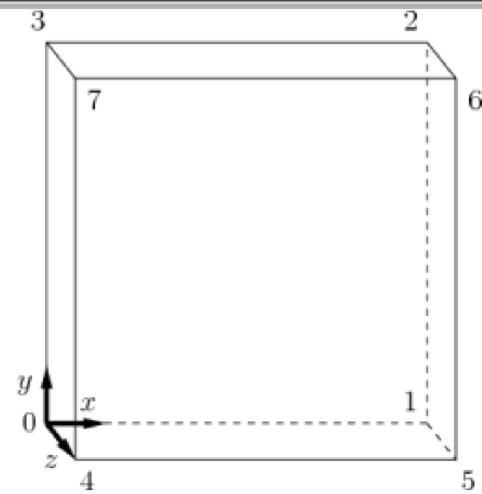



Figure 2.2: Block structure of the mesh for the cavity.

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

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.0.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

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)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

```
71
72 mergePatchPairs
73 (
74 );
75
76 // ************************************************************************* //
```

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 ($\mathbf{U}$) 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;
    }

```

```
    frontAndBack
    {
        type            empty;
    }
}

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

There are 3 principal entries in field data files:

`dimensions`
: specifies the dimensions of the field, here *kinematic* pressure, *i.e.* $\mathrm{m}^2\mathrm{s}^{-2}$ (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 *…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 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:

$$Re = \frac{d|\mathbf{U}|}{\nu} \qquad (2.1)$$

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

```
nu              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:

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

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 $\mathrm{ms}^{-1}$. The cell size is:

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

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

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

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

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

```

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;


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

## 2.1.1.5 Discretisation and linear-solver settings

The user specifies the choice of finite volume discretisation schemes in the *fvSchemes* dictionary in the *system* directory. The specification of the linear equation solvers and tolerances and other algorithm controls is made in the *fvSolution* dictionary, similarly in the *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 `pRefCell` and `pRefValue` in the *PISO* sub-dictionary of the *fvSolution* dictionary. In a closed incompressible system such as the cavity, pressure is relative: it is the pressure range that matters not the absolute values. In cases such as this, the solver sets a reference level by `pRefValue` in cell `pRefCell`. In this example both are set to 0. Changing either of these values will change the absolute pressure field, but not, of course, the relative 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 *paraFoam*, the post-processing tool supplied with OpenFOAM. The *paraFoam* post-processing is started by typing in the terminal from within the case directory

```
paraFoam
```

Alternatively, it can be launched from another directory location with an optional `-case` argument giving the case directory, *e.g.*

```
paraFoam -case $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
```

This launches the *ParaView* window as shown in Figure 6.1. In the *Pipeline Browser*, the user can see that *ParaView* has opened `cavity.OpenFOAM`, the module for the *cavity* case. **Before clicking the *Apply* button**, the user needs to select some geometry from the *Mesh Parts* panel. Because the case is small, it is easiest to select all the data by checking the box adjacent to the *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 *ParaView*. There are some general settings are applied as described in section 6.1.5.1. **Please consult this section about these settings**.

The user should then open the *Display* panel that controls the visual representation of the selected module. Within the *Display* panel the user should do the following as shown in Figure 2.3: (1) set `Color By Solid Color`; (2) click *Set Ambient Color* and select an appropriate colour *e.g.* black (for a white background); (3) in the *Style* panel, select `Wireframe` from the `Representation` menu. The background colour can be set by selecting `View Settings...` from `Edit` in the top menu panel.

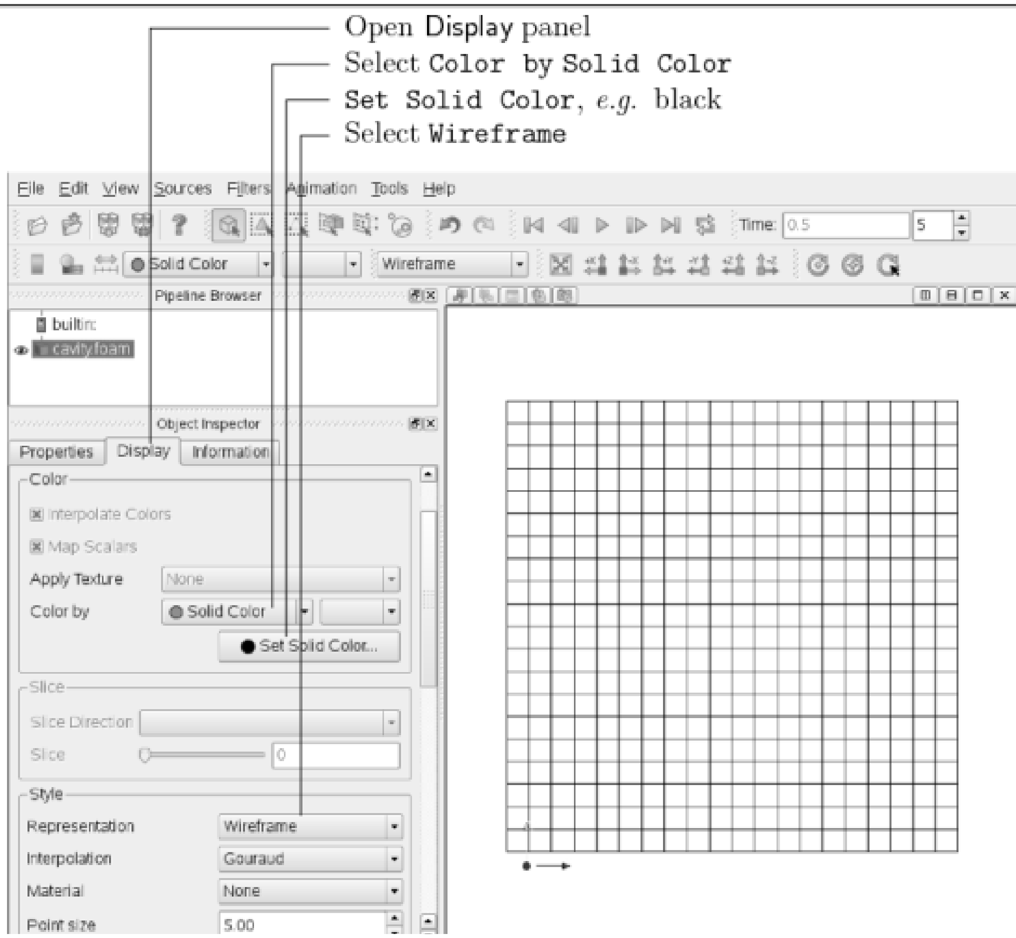


Figure 2.3: Viewing the mesh in *paraFoam*.

Especially the first time the user starts *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 *Use Parallel Projection* is selected in the *General* panel of `View Settings` window selected from the `Edit` menu. The *Orientation Axes* can be toggled on and off in the *Annotation* window or moved by drag and drop with the mouse.

## 2.1.3 Running an application

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

On this occasion, we will run *icoFoam* in the foreground. The *icoFoam* solver is executed either by entering the case directory and typing

```
icoFoam
```

at the command prompt, or with the optional `-case` argument giving the case directory, *e.g.*

```
icoFoam -case $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
```

The progress of the job is written to the terminal window. It tells the user the current time, maximum Courant number, initial and final residuals for all fields.

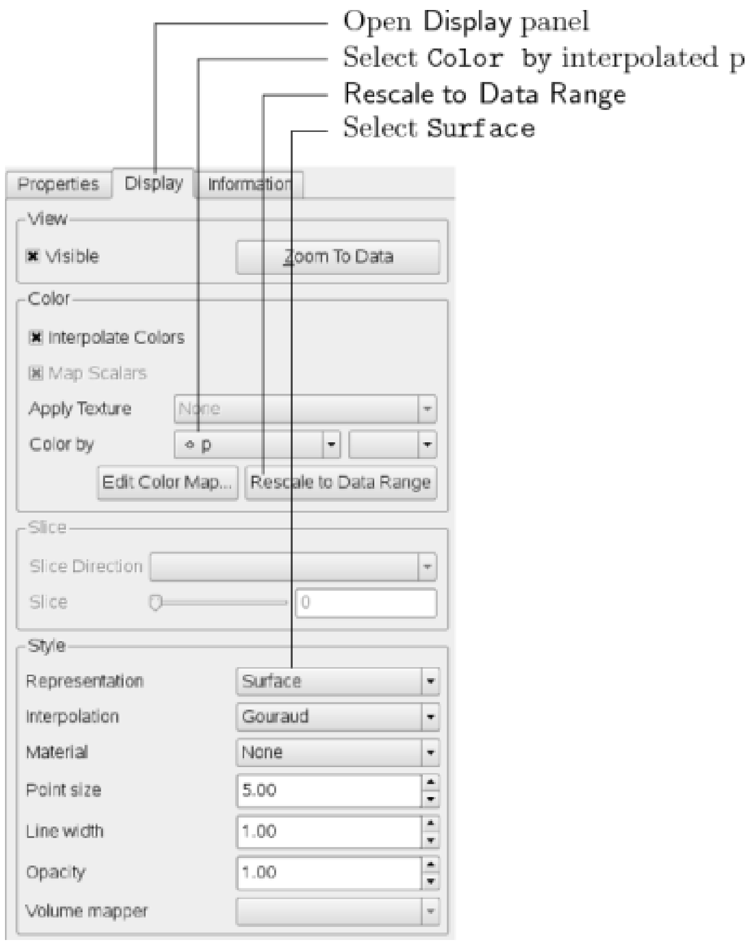



Figure 2.4: Displaying pressure contours for the cavity case.

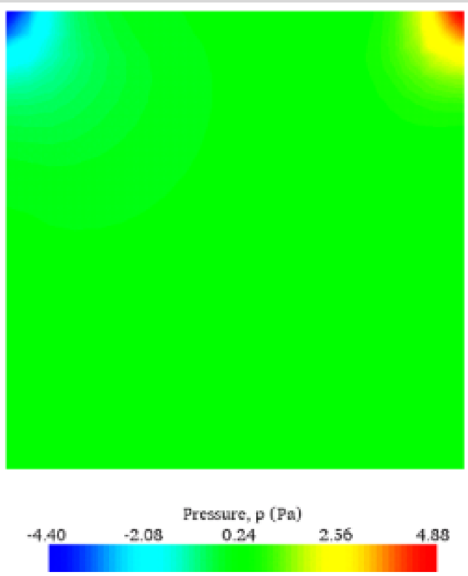



Figure 2.5: Pressures in the *cavity* case.

## 2.1.4 Post-processing

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

To prepare *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 *ParaView* was open, the output data in time directories will not be automatically loaded within *ParaView*. To load the data the user should click *Refresh Times* in the *Properties* window. The time data will be loaded into *ParaView*.

### 2.1.4.1 Isosurface and contour plots

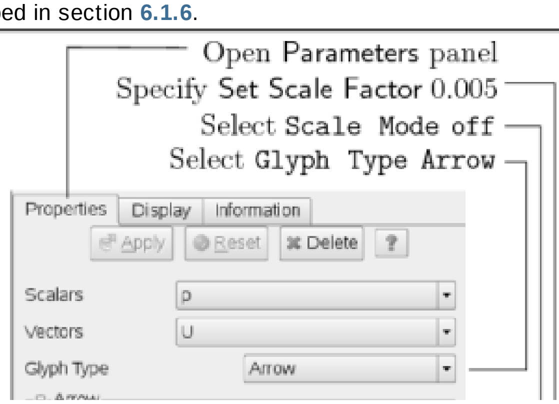
To view pressure, the user should open the *Display* panel since it that controls the visual representation of the selected module. To make a simple plot of pressure, the user should select the following, as described in detail in Figure 2.4: in the *Style* panel, select `Surface` from the `Representation` menu; in the *Color* panel, select `Color by` and *Rescale to Data Range*. Now in order to view the solution at $t = 0.5$ s, the user can use the `VCR Controls` or `Current Time Controls` to change the 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 by selecting `Show Color Legend` from the `View` menu. Clicking the *Edit Color Map* button, either in the `Active Variable Controls` toolbar or in the *Color* panel of the *Display* window, 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.

New versions of *ParaView* default 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 *Choose Preset* 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 $(0, 0, 1)$ (clcik 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.



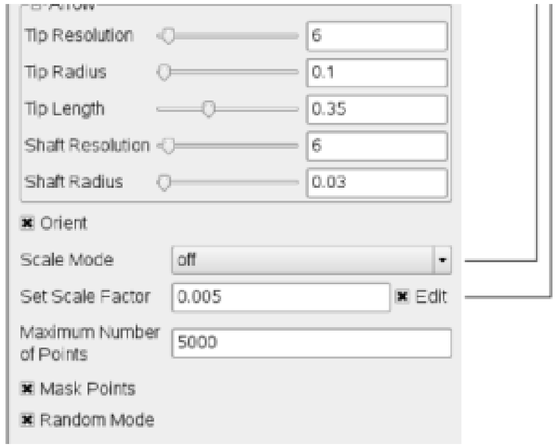



Figure 2.6: Properties panel for the `Glyph` filter.

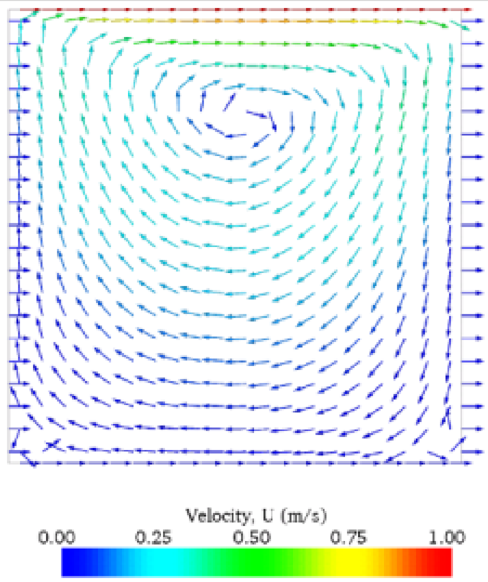



Figure 2.7: Velocities in the *cavity* case.

## 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->Alphabetical` menu. The *Properties* window panel should appear as shown in Figure 2.6. 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 *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.

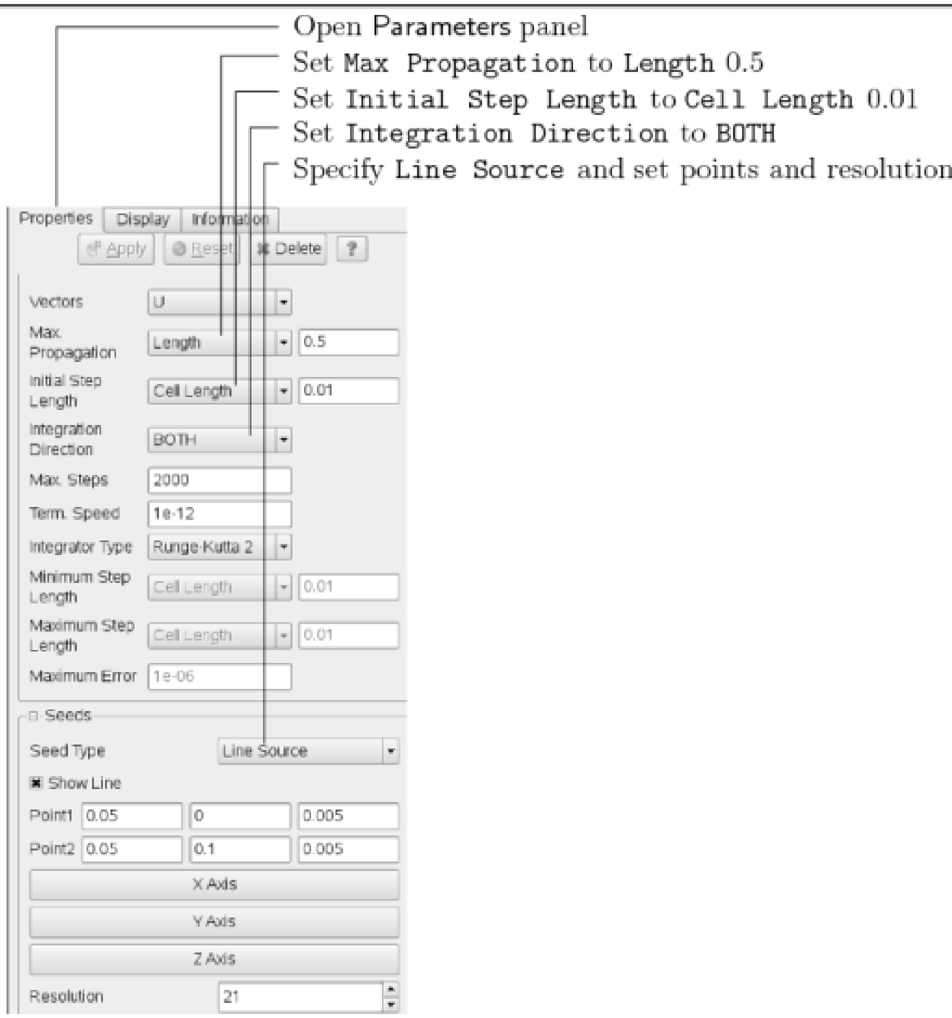
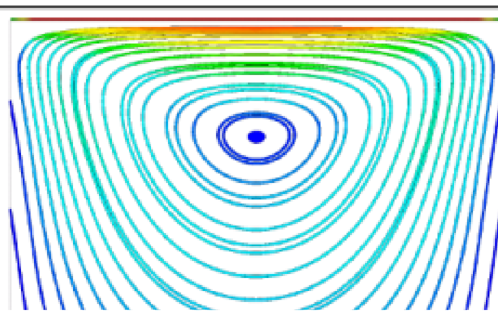



Figure 2.8: Properties panel for the `Stream Tracer` filter.



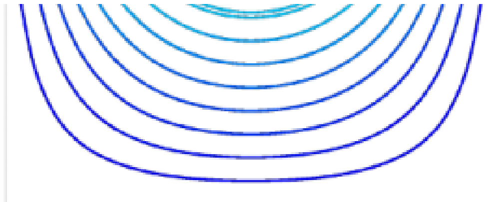


Figure 2.9: Streamlines in the *cavity* case.

### 2.1.4.3 Streamline plots

Again, before the user continues to post-process in *ParaView*, they should disable modules such as those for the vector plot described above. We now wish to plot a streamlines of velocity as described in section **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 `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; *Max Propagation* by `Length` 0.5; *Initial Step Length* by `Cell Length` 0.01; and, `Integration Direction BOTH`. The `Runge-Kutta 2` *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.*

```
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.

```
cp -r cavity/constant cavityFine
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`/`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 withing the *cavityFine* directory by

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

The utility should run with output to the terminal including:

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

Create databases as time

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

    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 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
```

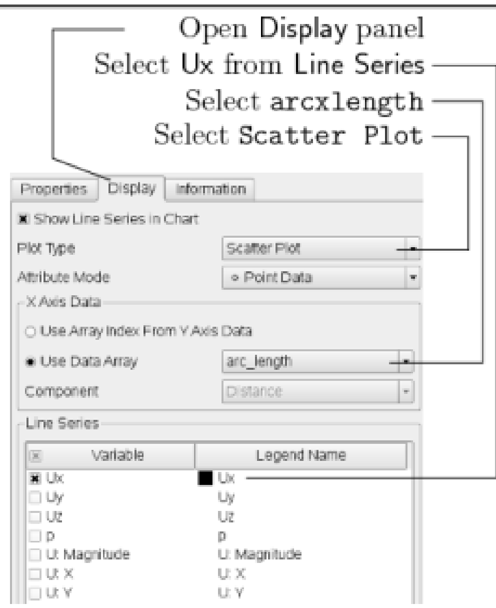
## 2.1.5.6 Vector plot with the refined mesh

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.



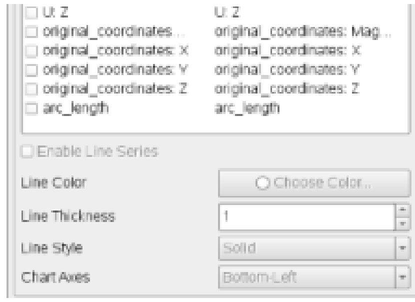



Figure 2.10: Selecting fields for graph plotting.

## 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 *foamCalc*. As a utility, it is unique in that it is executed by

```
foamCalc <calcType> <fieldName1 … fieldNameN>
```

The calculator operation is specified in `<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 `<calcType>` by deliberately calling one that does not exist, so that *foamCalc* throws up an error message and lists the types available, *e.g.*

```
>> foamCalc xxxx
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* `calcType`s 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 **6.5** 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.

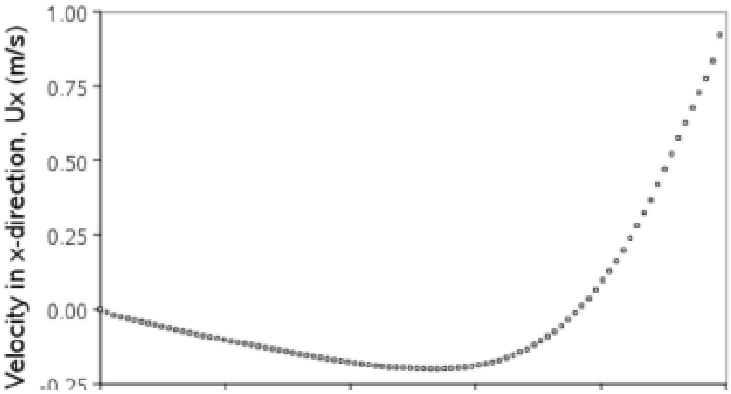
On clicking *Apply*, a graph is generated in the *XY Plot* window. In the *Display* panel, the user should set `Attribute Mode` to `Point Data`. The `Use Data Array` option can be selected for the *X Axis Data*, taking the `arc_length` option so that the x-axis of the graph represents distance from the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

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 `Ux` using *foamCalc*. Nevertheless, the user should deselect all series except `Ux` (or `U:x`). 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.



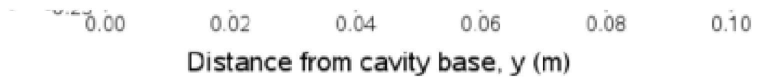



Figure 2.11: Plotting graphs in *paraFoam*.

## 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 $20 \times 20$ 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**.

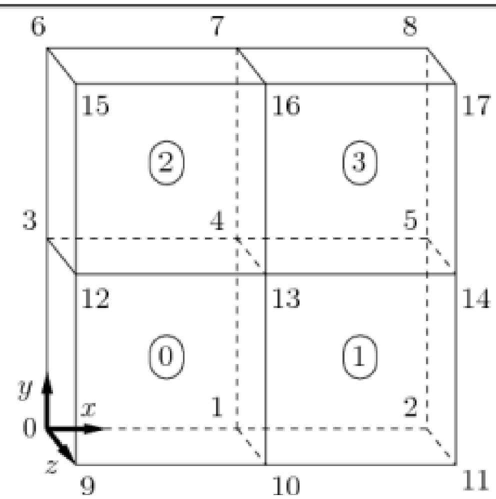



Figure 2.12: Block structure of the graded mesh for the cavity (block numbers encircled).

The user can view the *blockMeshDict* file in the *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)
    (0 1 0.1)
    (0.5 1 0.1)
    (1 1 0.1)
);

blocks
(
    hex (0 1 4 3 9 10 13 12) (10 10 1) simpleGrading (2 2 1)
    hex (1 2 5 4 10 11 14 13) (10 10 1) simpleGrading (0.5 2 1)
    hex (3 4 7 6 12 13 16 15) (10 10 1) simpleGrading (2 0.5 1)
    hex (4 5 8 7 13 14 17 16) (10 10 1) simpleGrading (0.5 0.5 1)
);

edges
(
);

boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (6 15 16 7)
            (7 16 17 8)
        );
    }
```

```
    fixedWalls
    {
        type wall;
        faces
        (
            (3 12 15 6)
            (0 9 12 3)
            (0 1 10 9)
            (1 2 11 10)
            (2 5 14 11)
            (5 8 17 14)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 4 1)
            (1 4 5 2)
            (3 6 7 4)
            (4 7 8 5)
            (9 10 13 12)
            (10 11 14 13)
            (12 13 16 15)
            (13 14 17 16)
        );
    }
);

mergePatchPairs
(
);

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

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:

$$\delta x_s = l \frac{r-1}{\alpha r - 1} \qquad (2.5)$$

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

$$r = R^{\frac{1}{n-1}} \qquad (2.6)$$

and

$$\alpha = \begin{cases} R & \text{for } R > 1, \\ 1 - r^{-n} + r^{-1} & \text{for } R < 1. \end{cases} \qquad (2.7)$$

For the *cavityGrade* case the number of cells in each direction in a block is 10, the ratio between largest and smallest cells is $2$ and the block height and width is 0.05 $\mathrm{m}$. Therefore the smallest cell length is 3.45 $\mathrm{mm}$. From Equation **2.2**, the time step should be less than 3.45 $\mathrm{ms}$ 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 $\mathrm{ms}$ 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
```

### 2.1.7.1 Pre-processing

Enter the 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 $1 \times 10^{-3}\ \mathrm{m^2 s^{-1}}$. 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:

```
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 1.62 s and initial pressure residuals are small; `No Iterations 0` indicates that the solution of `U` has stopped:

```

Time = 1.63

Courant Number mean: 0.108642 max: 0.818175
DILUPBiCG:  Solving for Ux, Initial residual = 7.86044e-06, Final residual = 7.86044e-06,
No Iterations 0
DILUPBiCG:  Solving for Uy, Initial residual = 9.4171e-06, Final residual = 9.4171e-06,
No Iterations 0
DICPCG:  Solving for p, Initial residual = 3.54721e-06, Final residual = 7.13506e-07,
 No Iterations 4
 time step continuity errors : sum local = 6.46788e-09, global = -9.44516e-19,
 cumulative = 1.04595e-17
 DICPCG:  Solving for p, Initial residual = 2.15824e-06, Final residual = 9.95068e-07,
 No Iterations 3
 time step continuity errors : sum local = 8.67501e-09, global = 7.54182e-19,
 cumulative = 1.12136e-17
 ExecutionTime = 1.02 s  ClockTime = 1 s

```

```
Time = 1.635

Courant Number mean: 0.108643 max: 0.818176
DILUPBiCG:  Solving for Ux, Initial residual = 7.6728e-06, Final residual = 7.6728e-06,
No Iterations 0
DILUPBiCG:  Solving for Uy, Initial residual = 9.19442e-06, Final residual = 9.19442e-06,
No Iterations 0
DICPCG:  Solving for p, Initial residual = 3.13107e-06, Final residual = 8.60504e-07,
No Iterations 4
time step continuity errors : sum local = 8.15435e-09, global = -5.84817e-20,
cumulative = 1.11552e-17
DICPCG:  Solving for p, Initial residual = 2.16689e-06, Final residual = 5.27197e-07,
No Iterations 14
time step continuity errors : sum local = 3.45666e-09, global = -5.62297e-19,
cumulative = 1.05929e-17
ExecutionTime = 1.02 s  ClockTime = 1 s
```

## 2.1.8 High Reynolds number flow

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

The need to increase spatial and temporal resolution then becomes impractical as the flow moves into the turbulent regime, where problems of solution stability may also occur. Of course, many engineering problems have very high Reynolds numbers and it is infeasible to bear the huge cost of solving the turbulent behaviour directly. Instead 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:

```

dimensions      [0 2 -1 0 0 0 0];

```

```
internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            nutkWallFunction;
        value           uniform 0;
    }
    fixedWalls
    {
        type            nutkWallFunction;
        value           uniform 0;
    }
    frontAndBack
    {
        type            empty;
    }
}


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

This case uses standard wall functions, specified by the `nutWallFunction` keyword entry one the `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/epsilon*) 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 $\mathbf{U}'$ and a turbulent length scale, $l$. $k$ and $\varepsilon$ are defined in terms of these parameters as follows:

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

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

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

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

where $U_x'^{\,2}$, $U_y'^{\,2}$ and $U_z'^{\,2}$ are the fluctuating components of velocity in the $x$, $y$ and $z$ directions respectively. Let us assume the initial turbulence is isotropic, *i.e.* $U_x'^{\,2} = U_y'^{\,2} = U_z'^{\,2}$, and equal to 5% of the lid velocity and that $l$, is equal to 20% of the box width, 0.1 $\mathrm{m}$, then $k$ and $\varepsilon$ are given by:

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

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

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

These form the initial conditions for $k$ and $\varepsilon$. The initial conditions for $\mathbf{U}$ 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. In most transient solvers, 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:

```

simulationType  RASModel;


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

The options for `simulationType` are `laminar`, `RASModel` and `LESModel`. With `RASModel` selected in this case, the choice of RAS modelling is specified in a *RASProperties* file, also in the *constant* directory. The turbulence model is selected by the `RASModel` entry from a long list of available models that are listed in Table **3.9**. The `kEpsilon` model should be selected which is 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 `printCoeffs` to `on` will make the default values be printed to standard output, *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 `Coeffs` appended, *e.g.* `kEpsilonCoeffs` in the case of the `kEpsilon` model. The coefficients of the model, *e.g.* `kEpsilon`, can be modified by optionally including (copying and pasting) that sub-dictionary within the RASProperties dictionary and adjusting values accordingly.

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

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

## 2.1.8.2 Running the code

Execute *pisoFoam* by entering the case directory and typing “`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 $\text{ms}^{-1}$ hence the maximum Courant number does not rise much above 0.2. It is sensible to increase the solution time by increasing the time step to a level where the Courant number is much closer to 1. Therefore reset `deltaT` to 0.02 s and, on this occasion, set `startFrom` to `latestTime`. This instructs *pisoFoam* to read the start data from the latest time directory, *i.e.10.0*. The `endTime` should be set to 20 s since the run converges a lot slower than the laminar case. Restart the run as before and monitor the convergence of the solution. 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 *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 *cavityClipped* case in the *icoFoam* directory which consists of the standard *cavity* geometry but with a square of length $0.04\ \mathrm{m}$ removed from the bottom right of the cavity, according to the *blockMeshDict* below:

```
convertToMeters 0.1;

vertices
(
    (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
(
    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)
    hex (3 4 7 6 11 12 15 14) (8 12 1) simpleGrading (1 1 1)
);

edges
(
);

```

```
boundary
(
    lid
    {
        type wall;
        faces
        (
            (5 13 14 6)
            (6 14 15 7)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (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
    {
        type empty;
        faces
        (
            (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
(
);

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

Generate the mesh with *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 `lid`, previously the `movingWall` patch of the original *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 *cavityClipped* case the mapping is set to occur at time 0.5 s, since the `startTime` is set to 0.5 s in the *controlDict*. Therefore the user needs to copy initial field data to that directory, *e.g.* from time 0:

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavityClipped
cp -r 0 0.5
```

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 *cavity* onto the new fields of *cavityClipped*. Since the mapping is inconsistent, we need to edit the *mapFieldsDict* dictionary, located in the *system* directory. The dictionary contains 2 keyword entries: `patchMap` and `cuttingPatches`. The `patchMap` list contains a mapping of patches from the source fields to the target fields. It is used if the user wishes a patch in the target field to inherit values from a corresponding patch in the source field. In *cavityClipped*, we wish to inherit the boundary values on the `lid` patch from `movingWall` in *cavity* so we must set the `patchMap` as:

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

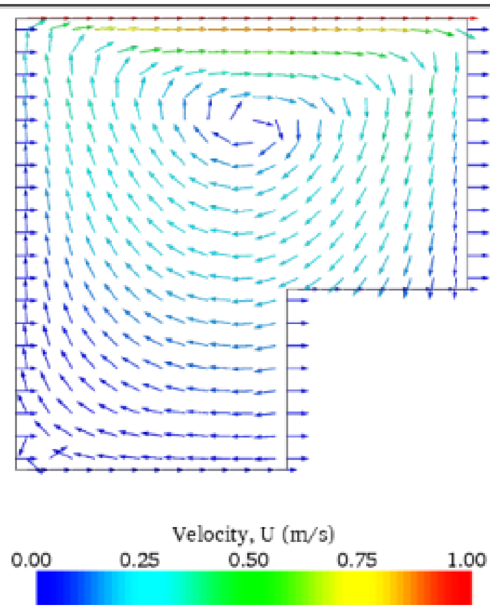
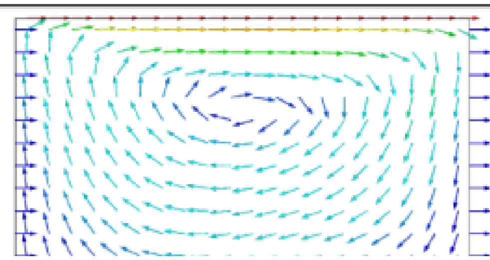



Figure 2.13: *cavity* solution velocity field mapped onto *cavityClipped*.



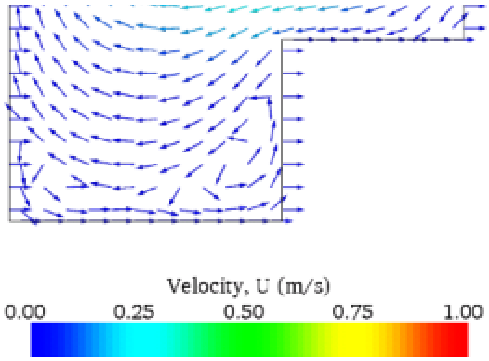



Figure 2.14: *cavityClipped* solution for velocity field.

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

```
cuttingPatches
(
    fixedWalls
);
```

Now the user should run *mapFields*, from within the *cavityClipped* directory:

```
mapFields ../cavity
```

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 `fixedWalls` patch to $(0,0,0)$. Edit the `U` field, go to the `fixedWalls` patch and change the field from `nonuniform` to `uniform` $(0,0,0)$. The `nonuniform` field is a list of values that requires deleting in its entirety. Now run the case with *icoFoam*.

## 2.1.10 Post-processing the modified geometry

Velocity glyphs can be generated for the case as normal, first at time 0.5 s and later at time 0.6 s, to compare the initial and final solutions. In addition, we provide an outline of the geometry which requires some care to generate for a 2D case. The user should select `Extract 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.