OpenFOAM用户指南

OpenFOAM User Guide - Chinese Edition

Christopher GreenShields 著

李东岳 译

作者 Christopher J. Greenshields

中文工作：

2.3 版译者（2015 年发布）：李东岳、田春来（第 5.5.2 节）、李建治（第 2.3 节）、周后村（第一章）
3.0 版修订（2016 年发布）：李东岳
5.0 版修订（2018 年发布）：李东岳、徐笑笑
7.0 版修订（2019 年发布）：涂灿、李东岳
9.0 版修订（2021 年发布）：李子丰

勘误、增补请前往 http://www.cfd-china.com/topic/1441 或联系 Email: li.dy@dyfluid.com

前言

计算流体力学（CFD）是一门朝气蓬勃的学科。

相对于传统的实验研究，CFD 将数学、流体力学、计算机编程进行糅合，主要的研究内容为采用计算机求解控制流体流动的偏微分方程组，相关的结果可用于指导工业设计。CFD 的特殊性在于只需要计算机即可进行工作。目前大量的行业已经开始使用 CFD 理论来指导实践，如空气动力学外形优化、气动噪声控制、多相流动混合模拟、化工过程装备设计、大气环境预测等。

在国外，将 CFD 真正的用于工业设计已经成为大量公司必须进行的作业流程。巨型的跨国企业有专业的业务部门做 CFD 分析。最近几年，国外也出现一批中小企业提供专业的 CFD 咨询服务。在国内，虽然 CFD 的学术水平与国外平分秋色，但 CFD 的工程应用和国外相比尚具备一定差距。正因如此，国内 CFD 的工程应用具有极大的发展空间。同时，在未来的移动互联时代，CFD 必将如一支利箭刺入形形色色的业务部门。如果以国外的 CFD/CAE 行业为对比标准，国内的 CFD/CAE 行业，将在未来面临势如破竹的发展。

你去了那么多地方，走了那么多的路
你要学会的就是包容和接受这个世界的一切观念

OpenFOAM 为世界最大的 CFD 开源软件。英文版的 OpenFOAM User Guide 为 OpenFOAM 自带的权威教材。中文版的《OpenFOAM 用户指南》及《OpenFOAM 编程指南》（以下简称指南）的翻译工作开始于 2014 年 9 月份，当时我刚到意大利都灵理工大学进行博士联合培养，彼时尚不具备独立科研能力。利用空闲时间将 OpenFOAM User Guide 进行全文翻译后获版权发布。指南的翻译工作我认为是有意义的事情。虽然目前英语的普及水平越来越高，但依然有大量的 CFD 从业者喜欢阅读中文书籍。当然翻译的初心，我只是为了自己能吸纳 OpenFOAM 领域旷世之作的精髓。为达到这个高度，逐字逐句的原著翻译是惟一的途径。这奠定了指南的工作背景。翻译工作进行的时候，业内存在各种不一样的声音。那个时候，我逐一的退出各种 CFD/OpenFOAM 交流群，不受外界干扰，潜心翻译。随后，第一版本的指南于 2015 年 3 月末如期发布。

我曾经跨过山和大海，也穿过人山人海

意大利的学业完成后，我于 2017 年前往德国亥姆霍兹计算流体力学研究所进行博士后工作，后于 2018 年年前回国。在去意大利的时候，每个人都问我为什么去意大利。在回国的时候，每个人都问我为什么不继续在国外做科研。我对第一个问题的答案是：为了追随纯粹的 CFD。为追求 CFD，我放弃全美前 100 高校的博士联培机会（方向略偏离 CFD）而去了意大利。第二个问题的答案是：为了不愧对于我的家庭。执笔之际，我正值 31 周岁。与胖猫恋爱 7 年，我有近 3 年为了提高自己的 CFD 技术独居海外。在随后胖猫备孕的 2018 年，需要丈夫的陪伴。回国前的 15 天，我独自去了一个荒凉的小岛（富埃特文图拉）。我理解要做好一个丈夫，我舍弃的是国外那令人向往的 CFD 科研氛围。在欧洲的近三年，我没去过任何地方游玩，因为我不

想牺牲做 CFD 的时间。相反的，我频繁和 CFD 大牛建立联系。我联系 OpenFOAM 基金会商议 OpenFOAM 中国发展事宜，我联系 ESI 集团讨论 ESI-OpenCFD 的走向，我和帝国理工大学的 Issa 讨论 PISO 算法，我参加世界级 CFD 大会只为了见那些活在文献里的 CFD 人物。工作和生活的难以平衡在我身上是一个鲜活的例子。曾经我为了追随 CFD 穿过人山人海。现在，我需要承担家庭的责任。

裂痕是什么，那是阳光照进来的地方

在北京生存，面临的是最高的房价，需要竞争的是顶尖的人才。2014 年，我创立东岳流体。2015 年，我创立 CFD 界。2016 年，我创立CFD 中文网。2017 年，我创立 CFD 百科并举办OpenFOAM Knowledge Share 课程。这一系列平台广为人知。另一方面，我也时刻警惕，恪守在国内生存的各种规则。特定时期，我的平台不会有我的名字。体制内已退休的家父，以及我在学术界的朋友都建议我关闭我的全部平台保持低调。不是毛头傻小子，我深谙国内自古以来的处事规则。是龙得盘着，是虎得卧着。然而一方面我身背每月 5 位数的房贷，一方面我脱离不开体制内。万幸的是我做的是 CFD，可以一个记事本 + 电脑走天下。

这一段话可能写的你不知所以然。因为有些话，到了嘴边，只能咽下去。未来的某一天，极少一部分人也会面临跟我一样的境遇。这些人，是那些心存向往、内心不安、为生活躁动的人。在这些时代弄潮儿广为人知的一面，存在着别人无法体会的裂痕，但这或许，才是阳光照进来的地方。

感谢我的父母在这 30 年来倾其所有的支持我的学业和事业。感谢我的妻子义无反顾的支持我近 3 年的独自出国留学。

最重要的感谢，献给那些一直以来默默支持和关注我的人们。

我是李东岳，我对我国所有令人钦佩的 CFD 从业者，此致敬礼！

2018.03.25
北京五道口

目 录

第 1 章　导论

本手册是 OpenFOAM 官方出版的 OpenFOAM User Guide 中文翻译版。

本手册详细描述了 OpenFOAM 的基本操作，第二章主要是相关 CFD 教程的练习，后续的章节则对 OpenFOAM 中包含的各类工具组件进行了详细地描述。

OpenFOAM 是一个 C++ 类库（约 100 个），其用于创建可执行文件，如应用程序（*application*）。OpenFOAM 内置的应用程序（约 250 个）有两类：求解器（*solver*）与工具（*utilities*）。求解器是为解决特定的连续介质力学问题而设计的；工具则是为执行数据操作等任务而设计的。OpenFOAM 包含了大量的求解器和工具，牵涉的问题也比较广泛，其将在第三章进行详尽的描述。

OpenFOAM 的一个特点是用户可以通过一些必要的知识（数学、物理和编程技术等）创建新的求解器和工具。

OpenFOAM 需要前处理和后处理环境。OpenFOAM 本身包含前处理和后处理接口，以确保不同环境之间数据传输的一致性。图1-1是 OpenFOAM 的结构示意图。在第四章中，将对前处理和如何运行算例进行详细的阐述，第五章将阐述如何利用 OpenFOAM 自带的网格划分工具来生成网格，以及如何将第三方软件生成的网格进行格式转换。第六章介绍后处理。第七章将介绍热物理模型库。

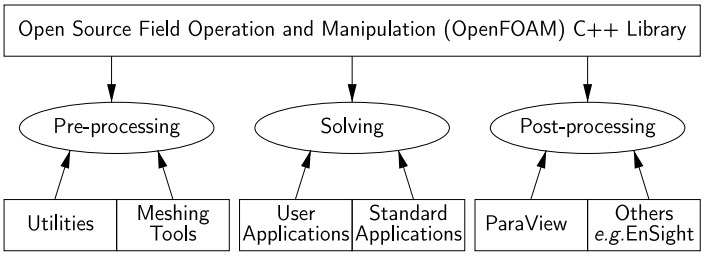


图 1-1: OpenFOAM 总体结构示意图

第 2 章　教程

这一章我们描述如何设置算例、如何进行模拟以及如何对 OpenFOAM 算例进行后处理，主要目的是给用户提供一个如何运行 OpenFOAM 的基本概念。`$FOAM_TUTORIALS` 目录里面有许多算例可以用来展示怎样使用各种求解器，以及如何使用 OpenFOAM 附带的各种工具。在运行算例之前，用户必须首先确保它们正确的安装了 OpenFOAM[1]。

在运行算例之前，用户需要确保 OpenFOAM 安装正确。建议将教程自带的算例拷贝到用户的 `run` 文件夹中，这个文件夹位于 `$HOME/OpenFOAM/<USER>/run` 下，其中 `USER` 为用户具体的名称[2]。`run` 文件夹通过 `$FOAM_RUN` 环境变量来定义，用户可以在终端键入[3]:

```
ls $FOAM_RUN
```

来查看 `run` 文件夹的具体位置。如果终端返回的是空文件夹，则用户可以通过键入：

```
mkdir -p $FOAM_RUN
```

来创立 `run` 文件夹。

第二章将描述如何使用前处理器 `blockMesh`[4]，以及如何用求解器进行求解、如何使用 ParaView 进行后处理等。

OpenFOAM 所有算例都位于 OpenFOAM 的目录 `tutorials` 中。这些例子依据流体类型整理为一系列文件夹，并通过求解器的名字再次分类。例如，`simpleFoam` 求解器的算例都存储在 `incompressible/simpleFoam` 文件夹下，其中 `incompressible` 代表流体类型（不可压缩）。如果用户想要运行一些算例，建议用户把教程下的例子拷贝到上文的 `run` 文件夹下。例如如果用户打算运行 `simpleFoam` 下的 `pitzDaily` 算例，用户可以在终端这样执行[5]:

```
cd $FOAM_RUN
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily .
```

2.1　顶盖驱动流

在这个教程中讲述如何对一个绝热二维方腔的不可压缩流算例进行前处理、运行、以及后处理。几何如图2-1所示，方腔所有的边界都是壁面，顶部壁面以 1m/s 的速度在 x 方向移动，其它壁面均为固定壁面。开始，我们假定流体为层流，并使用 `icoFoam`[6]在一个均匀网格上求解绝热不可压层流。接下来将研究网格非均匀化以及壁面网格非均匀化对计算结果的影响。最后，增加雷诺数，使用另一个求解器 `pisoFoam`[7]来求解绝热不可压湍流。

---

[1] OpenFOAM 中文安装方法请参考本链接

[2] 如：$/HOME/OpenFOAM/dyfluid-7/

[3] 2016 年起，OpenFOAM 间接支持常见的操作系统（如 Mac，Windows 等），但官方的 OpenFOAM 依然需要在终端下键入命令来运行。在 Ubuntu 系统下，终端可以通过 Ctrl+T 打开，且需要注意命令的大小写以及不要遗漏空格

[4] OpenFOAM 自带的网格生成工具，特点为生成快，但功能单一，通常和 OpenFOAM 自带的 snappyHexMesh 搭配使用

[5] 注意下述命令 pitzDaily 后面有一个实心句号

[6] icoFoam 为 OpenFOAM 比较简单的一个 NS 方程求解器 demo，其为一个近直接模拟求解器，有关 icoFoam 求解器解析请参考本链接

[7] 本手册中所有 xxxFoam 均为某个求解器的名字，且可以通过在终端键入具体名称来运行

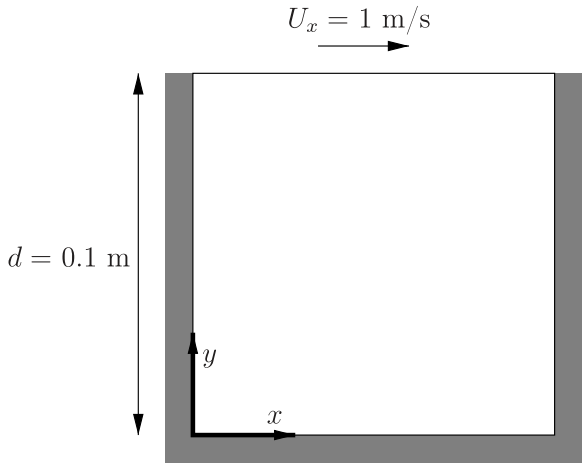


图 2-1: 顶盖驱动流的几何

2.1.1 前处理

用户需要编辑算例文件来下的文件来对 OpenFOAM 算例进行设置。用户需要选择一个文本编辑器来进行编辑，例如 emacs, vi, gedit, kate, nedit 等。OpenFOAM 允许对算例文件进行编辑，这样对一个初学者来说，使用文本文档里面的关键词来输入输出是简单易懂的。

运行算例需要一系列的数据：网格、场、物性、控制参数等。正如4.1节所说，OpenFOAM 中这些数据存储在算例目录下的一系列文件中，而不是像其它 CFD 软件那样把它们存储在单一文件中。例如本算例的所有衍生算例的所有文件则位于 `$FOAM_TUTORIALS/incompressible/icoFoam/cavity` 下，本算例直接简单的命名为 `cavity`。每个算例目录都有一个合适的名字。例如，第一个教程中算例的名字为 `cavity`。在准备编辑算例文件以及运行第一个 `cavity` 算例之前，用户应该将自带的算例拷贝到 `run` 目录下：

```
cd $FOAM_RUN
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity/cavity .
cd cavity
```

2.1.1.1 网格生成

OpenFOAM 采用三维笛卡尔坐标体系，所有算例网格都是三维的[8]。默认情况下，OpenFOAM 在三个维度上求解算例，但是如果指定某些面的边界（例如垂直于第三个方向的平面）条件为 `empty`，那么它就可以用来求解二维算例。

`cavity` 几何由一个 xy 平面上边长为 0.1 米的正方形组成。最开始将使用均一的 20*20 网格。`block` 结构参见图2-2. 我们采用 OpenFOAM 提供的 `blockMesh` 来生成网格，它通过读取指定的字典文件来生成网格，这个字典文件位于算例文件夹下的 `system/ployMesh`[9]文件夹下。`blockMeshDict` 文件信息如下所示：

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2    =========                 |
3    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4     \\    /   O peration     | Website:  https://openfoam.org
```

[8]在 OpenFOAM 中，2D 算例的网格在第三个方向有一定厚度

[9]旧版本 OpenFOAM 的 blockMeshDict 位于 constant/polyMesh 文件夹下

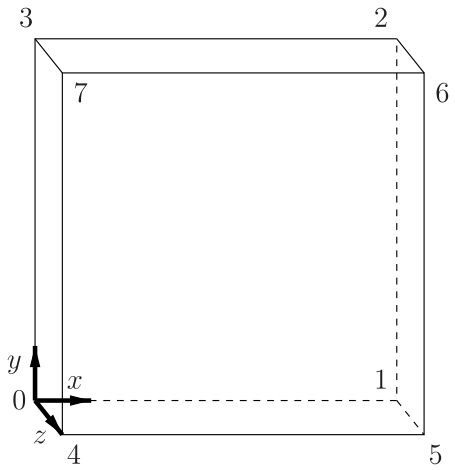


图 2-2: cavity 算例的 block 结构

```
    \\  /    A nd           | Version:  dev
     \\/     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  mergePatchPairs
72  (
73  );
```

文件的第一行到第七行是文件头信息，然后具体的字典信息通过的 FoamFile 后的 (...) 来指定[10]。

这个文件首先指定各个 block 顶点（vertices）的坐标；然后通过顶点编号来定义 block，最后定义边界面。用户可以查阅6.3节来详细了解 blockMeshDict 的具体意义。

网格通过在这个算例目录下运行 blockMesh 命令来生成。在算例目录下，通过在终端简单地键入：

```
blockMesh
```

来完成，blockMesh 命令会把运行的情况输出到终端。blockMeshDict 中的任何错误 blockMesh 都会检测到并在终端输出错误，并告诉用户第几行错了。目前我们运行这个算例的时候，不会有错误。

2.1.1.2 边界和初始条件

一旦网格生成完毕，用户可以查看这个算例的初始场信息。在这个算例中，我们从 0 秒开始计算，因此初始的场信息存储在 cavity 目录下的 0 文件夹下。0 文件夹下又包含两个文件，p 和 U，它们是压力场和速度场，它们的初值和边界条件必须指定。下面我们来看一下 p 文件：

```
17  dimensions [0 2 -2 0 0 0 0];
18
19  internalField uniform 0;
20
21  boundaryField
22  {
23      movingWall
24      {
25          type    zeroGradient;
26      }
27
28      fixedWalls
29      {
30          type    zeroGradient;
31      }
32
33      frontAndBack
34      {
35          type    empty;
36      }
37  }
```

场文件中有 3 个主要信息：

dimensions 指定场的单位，此处为运动压力[11]，单位为 m^2s^{-2}. 用户可查阅4.2.6节以获取更多信息；

internalField 内部场可以 uniform，即均一场；或者每个网格的数据都需要指定的 nonuniform（非均一场，更多信息请查阅4.2.8节）；

boundaryField 指定边界场信息，包括边界条件以及所有边界面所需的信息（请查阅4.2.8节）。

对于这个 cavity 算例，边界场由 walls 组成，并分为两个 patches：(1) fixedWalls，其用来指定固定边界，即 cavity 几何的底部和侧面。(2) movingWall，cavity 几何的上部。作为壁面，压力边界条件均为 zeroGradient，意味着压力的法向梯度为 0。frontAndBack 面即为 2D 算例的前后面，因此必须指定为 empty。

[10]在手册的剩余部分中：为了避免赘述和节省空间，在引用具体的算例字典文件时，文件头信息以及 FoamFile 子字典，都将省略。
[11]即除以密度之后的压力

在这个算例中（以后我们也会经常遇到）初始场我们都设置为 `uniform`。这里的压力为运动压力，作为不可压缩流，它的绝对值是没有意义的。因此我们设置其为 `uniform 0`。这样比较简单[12]。

用户可以采用同样的方法来查看 `0/U` 文件。`dimensions` 是速度的单位。内部场初始化为 `0`，对于速度来说，这个值必须是一个矢量，即：`uniform (0 0 0)`。请查阅4.2.5节获取更多信息。

速度场和压力场的 `frontAndBack` 都为 `empty`, 其它的 `patchs` 均为 `fixedWalls`，且指定为无滑移边界条件，因此为 `fixedValue`，其值为 `value uniform (0 0 0)`。我们假设顶部的壁面以每秒 1 米的速度在 x 方向移动，这是一个固定速度即 `fixedValue`，其具体的值为 `uniform (1 0 0)`。

2.1.1.3 物理特性

算例的物理特性存储在以 `Properties` 为后缀的文件中，对于 `cavity` 算例，唯一需要给定的参数为运动粘度，它存储在 `transportProperties` 字典文件中。用户可以打开 `transportProperties` 文件来查看运动粘度是否设置正确。运动粘度的关键词为 `nu`，类似于希腊字母 ν 的发音。我们的算例雷诺数为 10，雷诺数如下定义:

$$\mathrm{Re} = \frac{d\,|\mathbf{U}|}{\vartheta} \tag{2-1}$$

d 和 $|\mathbf{U}|$ 分别是特征长度和速度，ϑ 为运动粘度。此处 d 为 0.1m，$|\mathbf{U}|$ 为 1m/s，运动粘度为 0.01m^2/s。所以雷诺数为 10。正确地指定运动粘度场应该如下所示[13]：

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

2.1.1.4 控制

在 `controlDict` 字典内可以对时间步、输入输出时间、场数据的读取和写入来进行控制。用户应该查阅这个文件，作为一个算例的控制文件，它位于 `system` 文件夹下。

起始时间、终止时间以及求解时间步必须要进行设定。OpenFOAM 提供了灵活的时间步控制，请查阅4.4节。在这个教程里我们打算在 $t = 0$ 的时候开始进行计算，这意味着 OpenFOAM 需要读取 `0` 文件夹下的场数据，请查阅4.1节来获取更多有关算例文件结构的相关信息。因此我们设定 `startFrom` 关键字信息为 `startTime`，然后指定 `startTime` 为 0。

运算结束的时候，我们希望能达到稳定的状态，即流体在空腔内循环。一个通用准则是，层流下流体应该穿过（循环）计算域 10 次。在这个算例中，我们没有进口和出口，因此流体没有穿过计算域。因此结束时间应该设置为在空腔内循环 10 次的时间。例如 1s。实际上，我们发现 0.5s 就可以了，故我们设置结束时间为 0.5s。我们通过把 `stopAt` 关键字指定为 `endTime` 然后把 `endTime` 指定为 0.5 来完成。

现在我们需要设定时间步，即 `deltaT`。当运行 `icoFoam` 的时候，为了达到数值稳定以及时间计算精度，库郎数[14]应该小于 1。每个网格的库郎数这样定义:

$$\mathrm{Co} = \frac{\delta t\,|\mathbf{U}|}{\delta x} \tag{2-2}$$

[12]本算例压力边界条件全部为 zeroGradient，因此需要设置压力参考点和参考值。此处压力设置 0,10,100 均可。不可压缩流中关心的是相对压力而不是绝对压力

[13]老版本的 OpenFOAM 中，在 nu 的单位制定前还需要进一步的写入 nu 关键词

[14]Courant number，OpenFOAM 中的库郎数定义可参考本链接

δt 为时间步，$|\mathbf{U}|$ 为某个网格单元内的速度矢量的模，δx 为速度方向的网格长度。由于流体域内速度并不均一，为了保证库郎数在各处都小于 1，我们依据最大速度和最小网格尺寸来计算库郎数，在保证其小于 1 的前提下指定时间步。在这个算例中网格大小是固定的，因此在顶盖附近，速度接近 1m/s 的地方库郎数最大，网格大小为：

$$\delta x = \frac{d}{n} = \frac{0.1}{20} = 0.005\text{m} \tag{2-3}$$

为了使库郎数在整个流体域都小于或等于 1。时间步必须小于或等于：

$$\delta t = \frac{\text{Co}\delta x}{|\mathbf{U}|} = \frac{1 * 0.005}{1} = 0.005\text{s} \tag{2-4}$$

随着计算的进行，我们希望在某个固定的时间间隔下写入数据。以便我们可以在后处理中进行查看。对于如何对写入时间步进行设定有很多选择，它可以通过 `writeControl` 关键字来控制。我们决定在 0.1s, 0.2s, ⋯, 0.5s 写入我们的结果。由于计算时间步为 0.005s，因此我们需要每 20 个时间步输出一次结果，这样我们设定 `writeInterval` 为 20。

OpenFOAM 会在每一个时间步内创建一个文件夹，例如 `0.1` 文件夹。正如4.1节所说，每个时间步下都有一系列数据。对于 `icoFoam` 求解器，它把速度 U 和压力 p 写入在每个时间步对应的文件夹中。对于这个算例，`controlDict` 设置如下：

```
18  application         icoFoam;
19
20  startFrom           startTime;
21
22  startTime           0;
23
24  stopAt              endTime;
25
26  endTime             0.5;
27
28  deltaT              0.005;
29
30  writeControl        timeStep;
31
32  writeInterval       20;
33
34  purgeWrite          0;
35
36  writeFormat         ascii;
37
38  writePrecision      6;
39
40  writeCompression    off;
41
42  timeFormat          general;
43
44  timePrecision       6;
45
46  runTimeModifiable   true;
```

2.1.1.5 离散方法和矩阵求解器设置

用户在 `system` 下的 `fvScheme` 文件中指定有限体积法的离散格式。在 `system` 下的 `fvSolution` 文件下指定方程组矩阵求解器、残差以及其它算法控制。用户可以查看这些字典文件，目前我们不讨论这些内容，但是有必要提及 `fvSolution` 里面的 `PISO` 子字典中的 `pRefCell` 以及 `pRefValue`。在一个封闭的不可压体系中，例如这个空腔，压力是相对的。跟绝对压力值来说，压力的相对范围比较重要。在类似的这种例子中，求解器通过 `pRefCell` 以及 `pRefValue` 来设置参考值。这个例子中我们都设置为 0。改变其中任何一个值都会改变绝对压力场。但是不会改变相对压力场和速度场。

2.1.2 查看网格

在这个例子运行之前，我们最好检查一下网格以查看是否有错误。用户可以通过 ParaView 软件来查看网格，ParaView 是 OpenFOAM 推荐采用的后处理软件。针对 OpenFOAM 的数据，OpenFOAM 中的 paraFoam 程序可以自动加载 OpenFOAM 数据并在当前文件位置运行 ParaView[15]。

UNIX/Linux 的任何可执行程序都可以采用两种方式来运行。如果用户以前置的方式来运行程序，则下一个命令需要在前一个命令运行结束后才能运行。如果用户以后台的方式运行程序，则可以同时运行其他程序。因为我们可能需要在运行 ParaView 的同时并使用其他命令，因此我们可以通过后台的方式来运行，用户需切换到算例文件夹下并键入：

```
paraFoam &
```

它也可以在非算例目录下的终端运行，但应该使用-case 命令参数，例如：

```
paraFoam -case $FOAM_RUN/cavity &
```

如图7-1，这会启动 ParaView 窗口，在 Pipeline Browser 下，用户可以看出 ParaView 已经打开了这个算例模块并称之为 cavity.OpenFOAM，在应用 apply 按钮之前，用户需要在 Mesh Parts 里面选择要显示的几何。因为这个算例文件很小，最简单的是在 Mesh Parts 旁的窗口里选择全部几何数据，这会自动的检查所有几何单元。然后用户点击 Apply，网格文件就会被 ParaView 加载。

Display 面板控制所选模块的视觉效果。用户在 Display 面板下用户应该参照图2-3所示来进行操作:

1. 在 Coloring 内选择 Solid Color；
2. 然后点击 Edit 并选择合适的颜色，例如黑色（如果是白色的背景色）；
3. 在 Representation 菜单下，选择 Wireframe。背景色可以通过 Properties 窗口中 Display 面板下的 View Render 面板来设置。

用户第一次运行 ParaView 的时候，建议按照7.1.5节来进行操作。由于这个算例是一个二维算例，建议在 Edit 下的 ViewSetting 中的 General 面板中勾选 Use ParallelProjection。在 Annotation 面板下我们也可以选择打开或者关闭 Orientation Axes。

2.1.3 运行算例

正如其它任何 UNIX/Linux 可执行程序一样，OpenFOAM 程序可以通过前置进程和后台进程的方式来运行我们以前置进程的方法运行 icoFoam，一种方法是直接算例目录下的终端键入：

```
icoFoam
```

来运行，另一种方法是可以通过-case 命令参数来指定算例目录：

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

在终端的窗口里，我们会看到命令的输出信息，它告诉用户当前的时间步以及最大库郎数，所有场的初始残差和最终残差。

[15]用户也可以直接下载 ParaView 软件，并在算例文件夹下创建文件 case.foam，然后使用 ParaView 打开 case.foam 文件加载数据。这种方法也适用于 windows 版本的 OpenFOAM 用户。

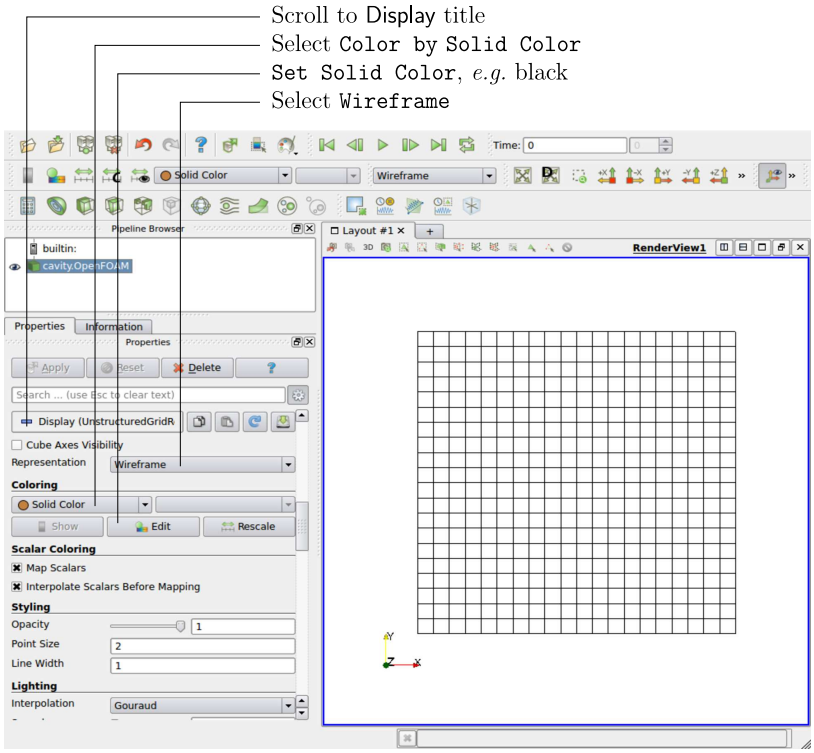


图 2-3: 通过 paraFoam 查看网格

2.1.4 后处理

在把结果写入时间步文件之后，我们可以使用 `paraFoam` 来进行后处理，回到 `paraFoam` 窗口，选择 `cavity.OpenFOAM` 模块。如果当前没有显示算例的网格，请确保 `cavity.OpenFOAM` 处于高亮绿色。并且在左边高亮显示一个眼睛的图标，这样就会显示当前的算例。

为了让 `paraFoam` 展示所需要的数据信息，首先要确保加载 0.5s 时间步的数据。如果运行算例的时候 ParaView 已经打开，新的时间步数据不会自动加载。为了加载新的时间步数据，应该在 `Properties` 中点击 `Refresh Times`。这会自动加载新的时间步数据文件。

为了查看 0.5s 的数据，用户可以使用 `VCR Controls` 或者 `Current Time Controls` 将当前时间步设置为 0.5。他们位于 ParaView 窗口的工具栏，详见图7-4。

2.1.4.1 面场显示

如果用户打算查看压力场，则应该切换到 `Display` 面板，这里控制所选模块的视觉效果。为了简单地做一个压力云图，用户应该按照图2-4来设置。

1. 在 `Representation` 菜单中选择 `Surface`；
2. 在 `Coloring` 中选择前方带圆点的 `p`；
3. 必要的话，单击 `Rescale` 按钮来重新加载数据范围。

这样，压力场就会呈现，正如期望的，空腔的左部顶端为低压区，右部顶端为高压区，详细请查看图2-5。

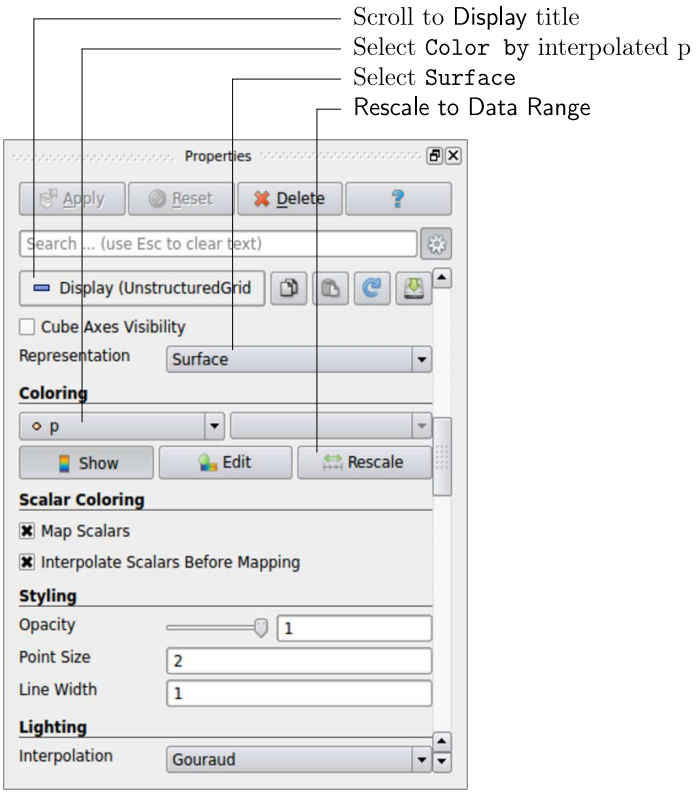


图 2-4: cavity 算例的压力云图设置

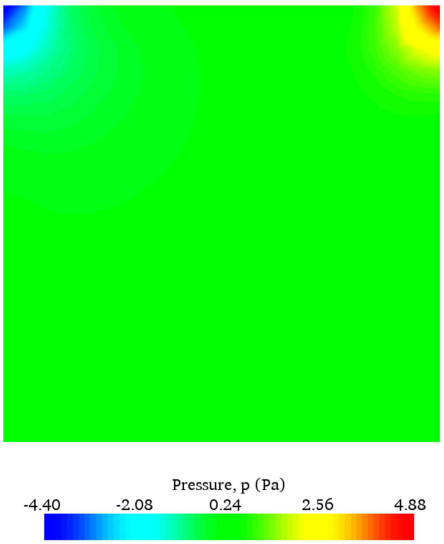


图 2-5: cavity 算例的压力云图

由于我们选择的是前方带圆点的 p，在这种情况下，每个网格的压力场都进行了插值，这样压力场看起来是连续的。如果用户在 Color by 菜单下选择前方带方框的 p，这样会显示每个网格的压力值，并且整个场的每个网格值并不是连续的。

颜色条可以通过点击 Display 面板下 Coloring 下的 Show 按钮中的 Active Variable Controls 工具栏下的 Toggle Color Legend Visibility 按钮来显示，颜色条一般位于窗口的右边，它可以用鼠标来拖拽。点击 ActiveVariable Controls 下或者 Display 面板下 Coloring 窗口下的 Edit 按钮，可以打开 Color Map Editor 窗口，如图2-6所示，用户可以选择自己喜欢的颜色条。ParaView 默认使用蓝-白-红颜色条，而不是常见的蓝-绿-红（彩虹）颜色条。因此当用户第一次打开 ParaView 的时候，他们可能打算改变颜色条的样式。用户可以通过在 Color Scale Editor 中单击 Choose Preset，然后选择 Blue to Red Rainbow。在点击 OK 确认之后，用户可以点击 Save asDefault，这样 ParaView 就会默认采用这种颜色样式。

用户也可以在 ParaView 中编辑其他的显示属性，如字体大小、字体、数字显示方式等，这可以通过单击 Edit Color Legend Properties 来实现，详见图2-6。

2.1.4.2 切面

如果用户旋转视角，可以发现整个三维几何表面都为压力场的颜色。如果想创建一个二维的云图,用户应该首先创建一个面[16],或者通过 slice 滤镜来创建一个 slice。在 Pipeline Brower 面板中高亮 cavity.OpenFOAM，然后在 ParaView 上方的工具栏的 Filters 中选择 Slice 滤镜（位于 Common 子菜单中）。切割平面中心可设置为（0.05,0.05,0.05），法向为（0,0,1）（或者点击 Z Normal 来自动设定）。

2.1.4.3 云图

在生成切面之后，云图可以通过应用 Contour 滤镜来生成。高亮 Slice 模块，然后选择 Contour 滤镜。在 Properties 面板中，用户应该选择 p。在 Isosurfaces 中，用户可以删除默认值并自行添加任意值。云图可以通过 Wireframe 来展现。

2.1.4.4 矢量图

在我们绘制矢量图之前，我们有必要屏蔽其它模块，例如上文生成的 Slice 和 Contour 模块。这有两种选择：要么全部删除，即首先在 Pipeline Browser 中高亮相关模块，然后在相应的 Properties 中点击 Delete[17]；或者关闭模块左边的眼睛标示以隐藏它。

现在我们打算在每个网格中心来生成矢量图。正如7.1.7.1所说，我们首先需要把数据处理到网格中心。首先在 Pipeline Browser 中高亮 cavity.OpenFOAM，然后用户在 Filter->Alphabetical 菜单中选择 Cell Centers 滤镜并应用。

高亮 Pipeline Browser 中的 Centers，然后在 Filter->Alphabetical 中选择 Glyph 滤镜。会出现图2-6所示的面板。在这个面板的 vectors 菜单中，速度场 U 被自动选取，因为它是目前唯一的矢量场。现在我们想查看整个流体域的矢量场，应该选择 off，设置 Set Scale Factor 为 0.005，点击 Apply，矢量场会展现，但可能为单一颜色，例如白色。用户可以通过在 Display 面板中选择 Color by U 来在箭头上显示速度的大小。除此之外，用户也需要在 Edit Color Map 中选择 Show Color Legend。请见图2-7,我们设定大写的 Times Roman 字

[16]Paraview 可以在 source 菜单中选择 plane 来创建切面

[17]直接在模块上右击并 delete 更为方便

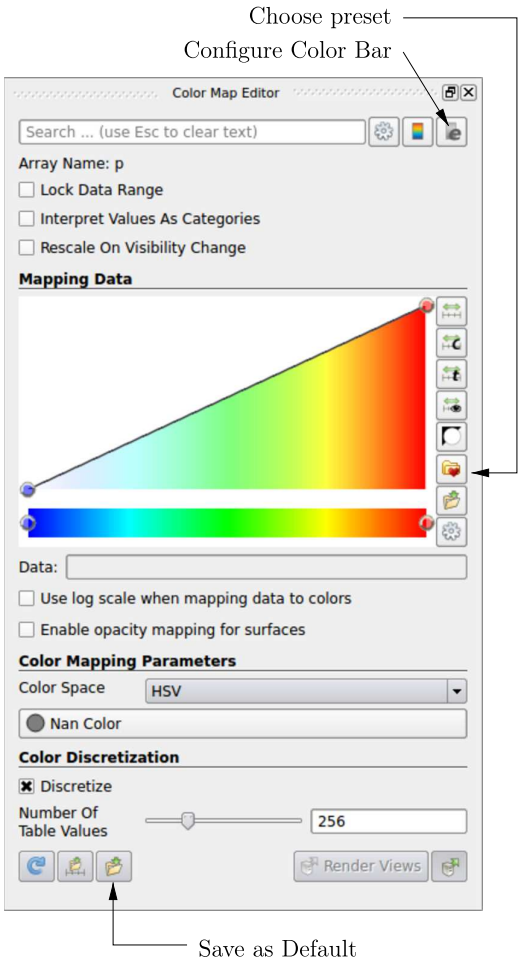


图 2-6: 颜色条设置

体作为 Color Legend,在 Label Format 中键入%-#6.2f,以及取消选择 Automati Label Format 并为其指定为上下 2 个固定值。背景颜色在 View Settings 的 General

面板中设定为白色，详见7.1.5.1。

我们注意到在左右的壁面上，矢量指示其穿透墙壁。然而，在仔细观察之后，我们发现垂直于壁面的矢量大小为 0。这种令人困惑的结果是 ParaView 导致的而不是计算结果的问题，这是因为我们选择了 glyph scaling 为 off，因此 ParaView 会显示所有网格的矢量，即使其大小为 0。

2.1.4.5 绘制流线图

在 ParaView 中进行其他后处理之前，用户应该把之前绘制的矢量图取消显示，现在我们打算创建流线图，更详细的内容请见7.1.8节。首先在 pipelinebrowser 中把 cavity.OpenFOAM 模块高亮，然后在 Filter 中选择 Stream tracer 并 Apply。如图2-9会出现一个 Properties 窗口，其中的 Seed 应该选择 High Resolution LineSource，它垂直于几何中心，例如从 (0.05,0,0.005) 到 (0.05,0.1,0.005)，针对我们显示的这个图，我们指定点的 Resolution 为 21，Maximum Step Length 长度为 0.5，Initial Step Length 为 Cell Length0.01，以及 Integration Direction BOTH，和默认的 Runge-Kutta 4/5Intergrator Type。

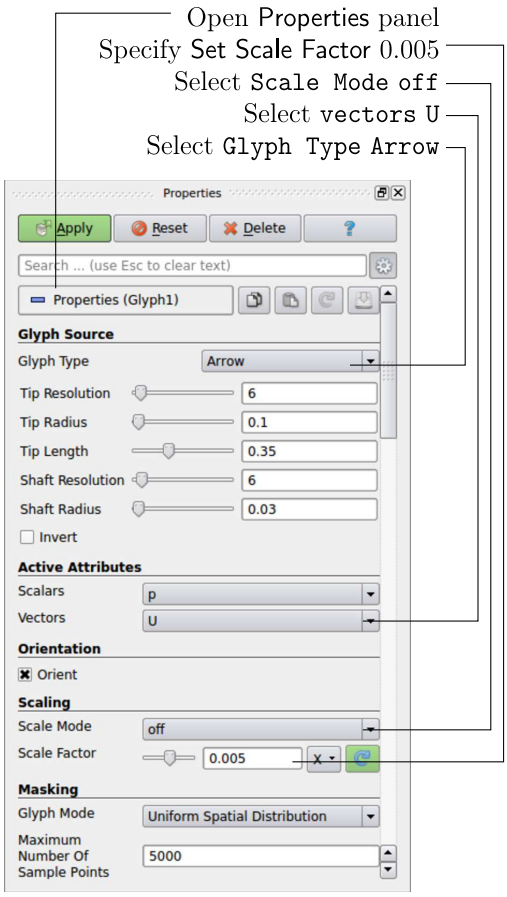


图 2-7: 矢量滤镜设置

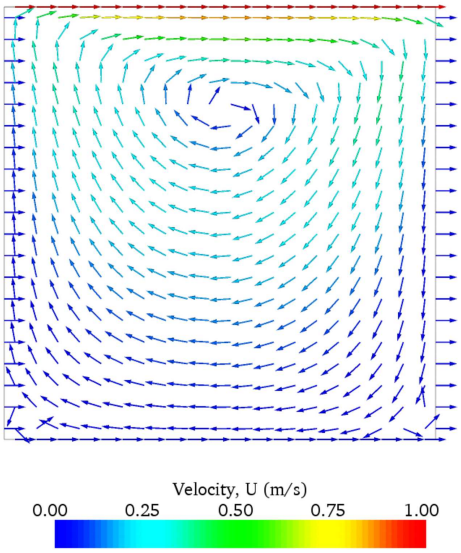


图 2-8: cavity 算例的速度矢量图

再点击 Apply 之后，流线图就会生成了。然后用户应该选择 Filter 中的 Tube 滤镜来创建高清晰度的流线，在图中我们就采用了这个方法。在这个算例中，我们设置 Num.sides 为 6；Radius 为 0.0003；Radius factor 为 10。管线用速度大小来显示，然后点击 Apply，图2-10就会生成。

2.1.5 网格细化

我们把网格数在每个方向上增加 2 倍，并把粗网格的结果映射到细网格作为初始条件。然后对比细网格和粗网格的结果。

2.1.5.1 使用存在的算例创造新算例

现在我们打算在 cavity 算例的基础上创造一个新的算例并称为 cavityFine，用户应该复制 cavity 算例并编辑必要的文件。首先，用户需要在 cavity 同级的目录下创建一个新的算例目录：

```
cd $FOAM_RUN
```

需要注意的是，我们可以直接在终端中键入 run 来直接切换到 $FOAM_RUN 文件夹下。然后需要把 cavity 算例下的文件拷贝到 cavityFine，然后进入 cavityFine 算例目录

```
mkdir cavityFine
cp -r cavity/constant cavityFine
cp -r cavity/system cavityFine
```

然后切换到 cavityFine 目录下：

```
cd cavityFine
```

2.1.5.2 生成细网格

我们打算用 blockMesh 来增加网格数量。用户应该用文本编辑器打开 blockMeshDict 并对其进行编辑。在 blocks 关键词下来指定 blocks。详细的句法请参见6.3.1.3节。在目前的阶段，我们只需知道 hex 后面是 block 的各个顶点，然后是每个方向的网格数量。在 Cavity 算例中，我们设置为 (20 20 1)，在这个算例中我们设置为 (40 40 1) 并保存文件。新的细化网格应该通过运行 blockMesh 来生成。

2.1.5.3 映射算例结果

mapFields 程序把一个给定几何的场信息，映射到另一个几何的对应场。在我们的例子中，场被认为是相同的，因为被映射场和原始场的几何，边界类型和条件都是一样的。因此在执行 mapFields 程序时，可以使用-consistent 附加命令选项。

mapFields 读取某个时间步的场信息，这由被映射场算例 controlDict 中的 startFrom/startTime（例如需要被映射的时间步）来指定，在这个算例中，我们打算把粗网格的最终结果映射到 cavityFine 的细网格中，由于 cavity 算例的最终结果存储在 0.5 文件夹中，因此 controlDict 文件中的 startTime 应该设定为 0.5，startFrom 设定为 startTime。

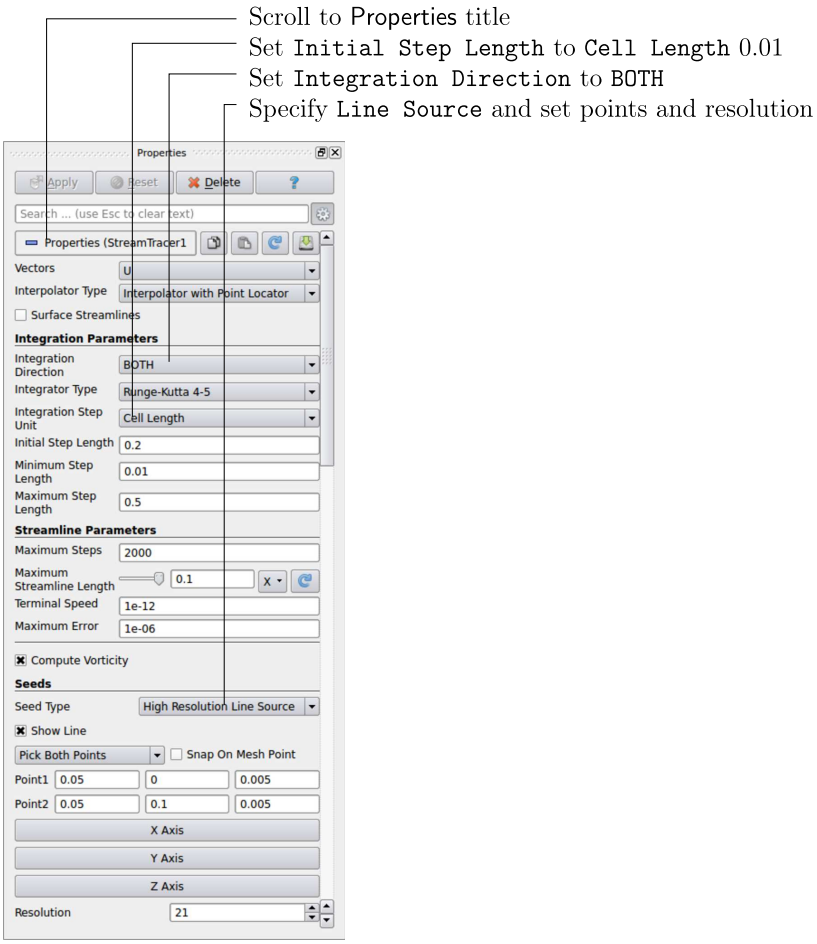


图 2-9: 流线图设置

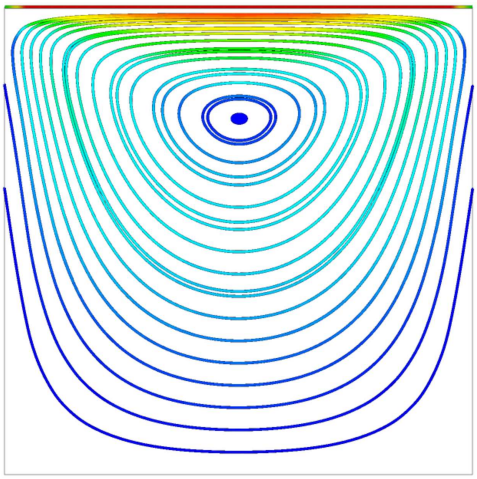


图 2-10: cavity 算例的流线图

现在这个算例就可以运行 `mapFields` 了，键入 `mapFields -help` 可以显示 `mapFields` 的命令参数。可以看出，提供源算例文件的地址是必须的，另外还需要使用`-consistent` 选项，因此在 `cavityFine` 终端里输入：

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

程序会输出以下信息:

```
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 控制参数

正如2.1.1.4所说，在算例求解的时候，最好要保持 Co 小于 1[18]。网格大小减半后时间步长必须减半，故 controlDict 字典中的 deltaT 需要设定为 0.0025。目前场信息设置为每多少个固定时间步写一次。现在我们学习如何在固定的时间间隔来写结果文件[19]。在 controlDict 中的 writeControl 中，之前设置为每个固定 timeStep 输出，我们也可以指定为 runTime 来指定每多少个物理时间间隔写结果文件。在这个算例中我们每 0.1 秒写一次结果。因此我们设定 writeControl 为 runTime，writeInterval 为 0.1。最后，由于算例从一个粗网格的结果来进行计算，因此我们只需要运行短暂的时间就可以达到稳态[20]。我们将 endTime 设置为 0.7 秒，确认所有的设置正确后我们保存文件。

2.1.5.5 后台运行

用户应该知道如何通过后台程序的方式来运行 icoFoam，并把输出信息输入到 log 文件中以便查看。在 cavityFine 目录下，我们输入：

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

2.1.5.6 在细网格上绘制矢量

用户可以在 ParaView 中同时打开多个算例，每个算例只是 Pipeline Browser 下的一个模块。但是在打开一个新的 ParaView 算例的时候会有一个小小的不便之处。即所打开的文件必须具有相应的扩展名。然而，OpenFOAM 算例的特点是每个算例具有多个文件（夹），它们以一定的文件结构存储在一个文件夹下。因此，paraFoam 脚本自动创建一个后缀为.OpenFOAM 的文件，这样 cavity 算例就被称为 cavity.OpenFOAM。

然而，如果用户想在 ParaView 中打开新的算例，他们需要创建这样的一个空文件。用户可以直接手动在算例下创建一个后缀为 OpenFOAM 的空文件，也可以通过 paraFoam 的附加命令

[18]库郎数的问题是 CFD 的基本问题，建议参阅网上相关资料以加深理解
[19]固定时间步间隔为每多少个 deltaT 一写，固定时间间隔为每 0.5 秒一写之类
[20]原文为 steady-state，由于 icoFoam 为瞬态求解器，原文的意思即为稳定的状态

-touch 来操作。例如，如果想加载 cavityFine 算例，可执行以下命令创建后缀为 OpenFOAM 的空文件：

```
paraFoam -touch
```

现在，点击 File 菜单中的 Open，并在导航窗口中选择 cavityFine.OpenFOAM 模块。ParaView 就可以加载 cavityFine 算例了。用户可以在这个细化的网格上用 ParaView 来绘制矢量图。如果想对比数据，可以把两个算例的矢量图同时打开。

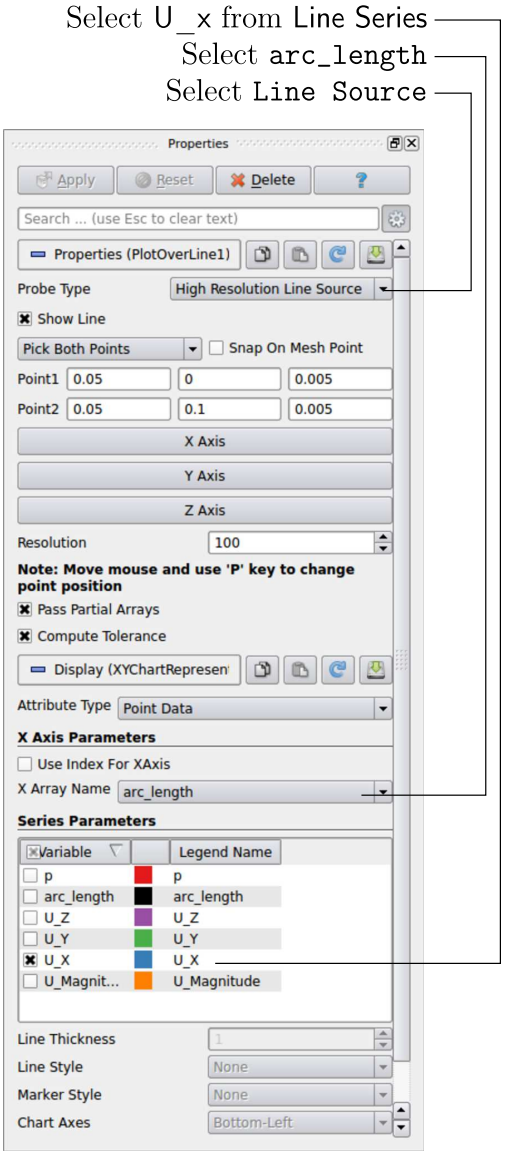


图 2-11: 选择 plot 场

2.1.5.7 绘制数据图

用户可能打算对速度的某个量来进行可视化处理并沿着计算域内的一条线来绘制一个二维数据图。OpenFOAM 中的 postProcess[21]可以调用有大量的函数来进行数据处理操作。postProcess 中包含的函数功能可以通过键入：

```
postProcess -list
```

来查看。其中 components 以及 mag 函数可用来对矢量进行操作来提取矢量分量以及模的大小。当在某个算例下，例如 cavity 下，它会读取每个时间步的速度场文件，并在相应的文件夹下写入 Ux,Uy,Uz 的场信息。

用户可以在 cavity 和 cavityFine 算例中运行 postProcess 后处理程序并附加 components 命令参数。例如，在 cavity 算例中，用户应该切换到 cavity 目录下并这样执行命令：

```
cd $FOAM_RUN/cavity
postProcess -func ''components(U)''
```
[22]

这样，在 ParaView 中就可以绘制速度的分量图了。ParaView 可以快速、方便的对图表进行控制，因此输出的图表是非常标准化的。但用户可能喜欢用其他专业的画图软件，如 gnuplot 和 Grace/xmgr 等对原始数据进行操作来绘制。如果想进行这样的操作，建议使用 sample 函数，详见7.3.2和2.2.3节。

在开始绘图之前，用户需要让 ParaView 加载新生成的 Ux，Uy，Uz 场。用户可以单击 Properties 面板下的 Refresh Times 按钮，这样 ParaView 就会加载新的场并出现在 volume field 列表中。选中需要加载的新场，单击 Apply 按钮。另外我们会发现，如果在 Mesh Parts 中选择了边界区域，那么在边界处数据的插值有错。因此需要在 Mesh Parts 面板中，取消勾选边界面，例如 movingWall，fixedWalls 以及 frontAndBack 并 Apply[23]。

现在，为了在 ParaView 中绘图，用户应该选择当前模块，例如 cavity.OpenFoam，然后在 Filter 选项中选择 Data Analysis 下的 Plot Over Line。这会在现存的 3D 视图旁边打开一个新的 XYPlot 窗口，同时在 Properties 面板下创建一个 PlotOverLine 模块，用户可以设置线的起始端和结束端。在此例子中，用户需创建一个穿过计算域中心的垂直线。例如，在 Point1 中设置起始点为 (0.05,0,0.005)，在 Point2 中设置终点为 (0.05,0.1,0.005)。Resolution 可以设置为 100。

单击 Apply，在 XYPlot 视图中会生成一个图表。在 Display 面板中，用户应该把 Attribute Type 设置为 Point Data。X Axis Data[24]我们选择 Use Data Array 以及 arc_length，这样 x 轴代表了离底距离。

接下来，用户可以在 Display 窗口中的 Line Series[25]面板中选择显示的场，在现有的标量场中，可以看见默认设置就已经有了速度的模以及 3 个分量。例如 U_X[26]，这意味着不需要利用 postProcess 程序来生成 Ux 数据。用户应该仅仅选择 Ux(或者 U_X。在旁边的彩色方框内代表的是线条的颜色。用户可以通过双击鼠标对此进行随意的设置。

[21] 在老版本的 OpenFOAM 中为 foamCalc，foamCalc 和 postProcess 的使用区别较大
[22] 老版本的 OpenFOAM 中需要运行 foamCalc components U
[23] 新版本的 paraview 中默认取消勾选
[24] 新版本的 paraview 为 X Axis Parameters
[25] 新版本的 paraview 为 Series Parameters
[26] 不同版本的 paraview 可能有不同的名称，例如 U(1)，U(x)

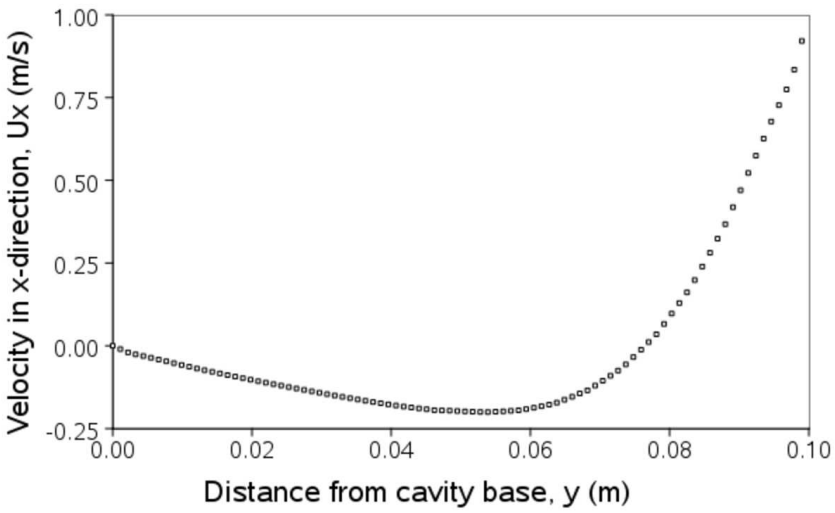


图 2-12: 在 paraFoam 中绘图

如果想对图标进行润饰，可以在 `Line Series` 面板下修改设置，如 `Line color`，`Line Thickness`，`Line Style`，`Marker Style` 以及 `Chart Axes`。

同时，用户注意到曲线图的左上方有一些小工具可供使用。第三个按钮可以用来对视图选项进行设置，例如设置图标的名称、每个轴的单位、字体，颜色以及坐标轴名称等。坐标轴的刻度也可以分为线性和 `log` 坐标轴。在 `XY Plot` 的左上角也有一些功能。例如用户可以点击左上角第三个按钮对 `View Settings` 进行设置，如坐标轴标记。其他的如字体、颜色等也可以采用类似的方法进行设置。

图2-12是使用 ParaView 绘制的图表。图2-11我们使用了下述选项：在 `Notation` 中选择 `Standard`[27]并选取 `Specify Axis Range`，标题为 `Sans Serif 12` 号字。这个图我们没有激活线选项（在 `Display` 面板中选中 `Enable Line Series`），相反的我们用一系列的点来绘制。要注意，如果这个按钮是灰色的，可以通过在 `Line Series` 面板中选择或反选需要绘制的场来激活它。一旦激活 `Enable Line Series`[28]，用户就可以依据他们的喜好随意的选择 `Line Style` 和 `Marker Style`。

2.1.6 网格非均匀分布

在某些区域，真实解的分布型线会和选定的数值格式的形式有很大的不同。如果这种区别很大的话，在这些区域求解的误差会比较大。例如，如果真实解本身为线性形式的话，基于线性的数值格式才会预测准确的解。如果某些区域的真实解本身并不是线性形式，那么在这些区域（比如梯度较大的区域）中使用线性格式来求解的结果会有较大误差。细化网格会使误差减小。

在对任何一个算例进行求解之前，我们都应该对解的形式有一个直觉的概念。这样我们就可以预测哪里误差比较大然后在这一部分细化网格，以使得这些区域的网格较小。在 `cavity` 算例中，速度最大的变动发生在壁面处，因此在我们的教程中，这一部分的网格将会被非均匀化处理。通过这种方法，我们就可以利用同样数量的网格，在不增加计算机资源的情况下获得更准确的解。

接下来我们针对顶盖驱动流算例创建一个 20*20 的网格，壁面处引入网格非均匀分布特性，然

[27]新版本 paraview 中取消了 Standard，并且四个轴可以分别设置

[28]新版本 paraview 中没有 enable line series 选项，只要在 linestyle 里面选择 none 就可不显示线，然后在 marker style 里选 circle 或者 square 能实现图2-12效果，设置的时候注意把调整的三条线都选中

后把2.1.5.2节中的细网格算例的结果映射到非均匀网格中作为初始条件。并把非均匀网格的计算结果和之前的算例结果进行对比。在非均匀的网格算例中，blockMeshDict 字典文件改动非常大，因此我们重新创建一个算例，这个算例可以在 $FOAM_TUTORIALS/incompressible/icoFoam 文件目录的 icoFoam 目录下找到，用户需要拷贝其中的 cavityGrade 算例至 run 文件夹下以便下一步操作。

2.1.6.1 创建非均匀化网格

在这个算例中，这个网格需要 4 个 blocks[29]，上下面和左右面都需要网格非均匀化分布处理。图2-13中即为 block 结构。用户可以在 cavityGrade 的 constant/polyMesh 下查阅

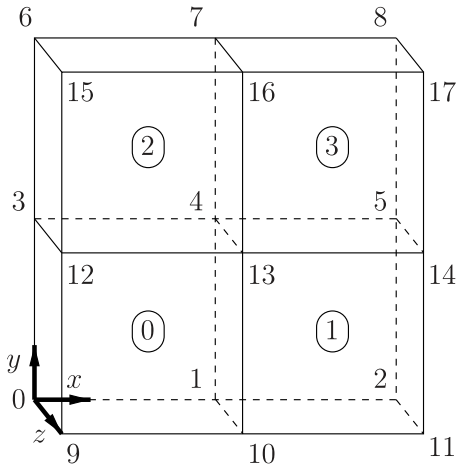


图 2-13: 顶盖驱动流的非均匀 block 结构

blockMeshDict 文件，我们把这个字典文件展示如下，每个 block 在 x,y 方向都有 10 个网格，最大网格和最小网格的大小比为 2：

```
17  convertToMeters 0.1;
18
19  vertices
20  (
21      (0 0 0)
22      (0.5 0 0)
23      (1 0 0)
24      (0 0.5 0)
25      (0.5 0.5 0)
26      (1 0.5 0)
27      (0 1 0)
28      (0.5 1 0)
29      (1 1 0)
30      (0 0 0.1)
31      (0.5 0 0.1)
32      (1 0 0.1)
33      (0 0.5 0.1)
34      (0.5 0.5 0.1)
35      (1 0.5 0.1)
36      (0 1 0.1)
37      (0.5 1 0.1)
38      (1 1 0.1)
39  );
40
41  blocks
42  (
43      hex (0 1 4 3 9 10 13 12) (10 10 1) simpleGrading (2 2 1)
44      hex (1 2 5 4 10 11 14 13) (10 10 1) simpleGrading (0.5 2 1)
45      hex (3 4 7 6 12 13 16 15) (10 10 1) simpleGrading (2 0.5 1)
46      hex (4 5 8 7 13 14 17 16) (10 10 1) simpleGrading (0.5 0.5 1)
47  );
48
49  edges
50  (
51  );
```

[29]新版 OpenFOAM 已经可以处理单 block 的非均匀分布（1 个 block 就可以），此处的处理方法为旧式处理方法

```
52
53  boundary
54  (
55      movingWall
56      {
57          type wall;
58          faces
59              (
60                  (6 15 16 7)
61                  (7 16 17 8)
62              );
63      }
64      fixedWalls
65      {
66          type wall;
67          faces
68              (
69                  (3 12 15 6)
70                  (0 9 12 3)
71                  (0 1 10 9)
72                  (1 2 11 10)
73                  (2 5 14 11)
74                  (5 8 17 14)
75              );
76      }
77      frontAndBack
78      {
79          type empty;
80          faces
81              (
82                  (0 3 4 1)
83                  (1 4 5 2)
84                  (3 6 7 4)
85                  (4 7 8 5)
86                  (9 10 13 12)
87                  (10 11 14 13)
88                  (12 13 16 15)
89                  (13 14 17 16)
90              );
91      }
92  );
93
94  mergePatchPairs
95  (
96  );
```

目前我们应该已经对 blockMeshDict 字典文件很熟悉，用户可以直接执行 blockmesh 命令，非均匀化处理之后的网格可以像之前一样执行 paraFoam 来查看，详见2.1.2节。

2.1.6.2 调整时间步

因为顶盖附近的速度最高，网格最小，因此顶盖附近的库郎数也会变大，原因请参考2.1.1.4节。因此我们首先算一下顶盖附近网格的大小，以此来计算适合本算例的时间步。

当使用一个非均匀网格的时候，blockMesh 会计算网格大小。如果长度为 l，内置 n 个网格单元，最末端和起始端网格比为 R，那么最小的网格大小就为 δx，其大小为：

$$\delta x = l\frac{r-1}{\alpha r-1} \tag{2-5}$$

r 是相邻网格单元的大小比：

$$r = R^{\frac{1}{n-1}} \tag{2-6}$$

且有：

$$\alpha = \begin{cases} R & R>1 \\ 1-r^{-n}r^{-1} & R<1 \end{cases} \tag{2-7}$$

对于 cavityGrade 算例，每个方向网格数为 10，最大网格单元和最小网格单元大小比为 2，block 的高宽为 0.05m。因此，最小的网格单元为 3.45mm。从方程(2-2)可以看出时间步应该小于 3.45 毫秒，才能保证库郎数小于 1。为了方便写入每个时间步的计算结果[30]，deltaT 设置为 2.5 毫秒。writeInterval 设置为 40，因此每隔 0.1 秒写一个结果文件。所有这些设置可以在 cavityGrade/system/controlDict 里面查看。

[30]这样下面就可以每 40 步一写来写入一个整数的时间步

本算例的 startTime 需要设置为 cavityFine 的结束时间，即 0.7 秒。由于 cavity 和 cavityFine 算例都已经收敛的很好了，因此这个算例我们只需要计算 0.1s，结束时间 endTime 设为 0.8。

2.1.6.3 映射场

在2.1.5.3中，我们使用 mapFields 把 cavity 的网格结果映射到 cavityFine 中。现在把 cavityFine 的结果映射到 cavityGrade 中：

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

接下来我们在算例目录下运行 icoFoam 并监控运行的输出信息，并对结果进行后处理，然后和之前的算例结果相对比。如何进行后处理请参阅2.1.5.6和2.1.5.7节。

2.1.7 增加雷诺数

目前的算例雷诺数非常低，仅为 10，其可以快速的达到稳定状态。并且只有一个非常小的涡旋存在于空腔的底角。我们现在把雷诺数增加到 100。这会明显增加收敛时间。首先，我们使用最粗糙的 cavity 网格。用户应该拷贝 cavity 算例，并把它命名为 cavityHighRe。相对于对算例进行直接拷贝操作，新版本 OpenFOAM 可以采用 foamCloneCase 命令进行算例复制。默认情况下，foamCloneCase 拷贝的是 0 文件夹。但是用户可以指定-latestTime 来拷贝最新的文件夹（如 0.5，其可以作为新算例的初始场）。在下面的操作中，我们也演示使用 run 命令快速切换到 run 文件夹下：

```
run
foamCloneCase -latestTime cavity cavityHighRe
cd cavityHighRe
```

2.1.7.1 前处理

进入 cavityHighRe 算例，编辑 transportProperties 字典文件，由于雷诺数需要扩大 10 倍。我们可以通过把粘度降低 10 倍来实现。变为 $1e-3 m^2 s^{-1}$。现在我们可以把 cavity 算例的最终结果作为初始条件来运行算例，这通过把 startTime 改为 latestTime 来实现，这意味着 icoFoam 会从最后一个时间步来开始新的计算，对于当前这个算例是 0.5s。结束时间 endTime 应该是 2s。

2.1.7.2 运行算例

接下来我们应该在算例目录下运行算例并查看输出。如果以后台进程的方式来运行算例的话，会使用到下面的 UNIX 命令：

nohup 当用户退出登陆的时候，这个程序依然会继续运行；

nice 调整进程优先级，-20 对应最高优先进程，19 对应最低优先进程。

有时用户会希望远程运行算例，并且不打算进行监控，且想把进程调为低优先级。这个时候这两个命令会非常有用。当用户退出远程系统的时候，nohup 会让进程继续运行，nice 可以设置为 19

来降低进程优先程度。对于我们的这个算例，可以这样来执行命令：

```
nohup nice -n 19 icoFoam > log &
cat log
```

在之前的算例中，icoFoam 在非常短的时间内就停止了对速度进行迭代[31]，但它一直对压力进行求解直到求解终止。实际上，在 icoFoam 停止求解速度场，并且在压力的初始残差小于 fvSolution 设定的残差值（一般为 10e-6）之后，求解就已经收敛，在求解器把结果场写出来的时候就可以停止了[32]。例如（参见下面的 log 文件，这个是 cavityHighRe 算例的计算输出结果）：速度在 1.395 秒以后已经收敛。并且压力的初始残差非常小；No Iterations 0 表示速度求解已经停止：

```
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 = 2.60898e-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 = 8.53935e-06, Final residual = 8.53935e-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 = 8.44595e-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 高雷诺数流动

接下来我们通过 paraFoam 来查看结果并查看速度矢量。可以发现角落里的第二个涡略微变大。用户可以尝试继续减少粘度来增加雷诺数并重新运行算例。涡的数量会增加，因此就需要在涡的附近增加网格来求解更复杂的流型。另外，高雷诺数也增加了收敛所需要的时间。用户应该监控残差，并延长求解时间以确保收敛。

实际上，在求解湍流域的时候，这种提高网格数量和增加求解时间是不切实际的，求解稳定性也较差。当然许多工程问题的雷诺数都很高，直接对湍流域进行求解带来的计算资源消耗很不经济[33]。另一种方法是，我们可以使用 Reynolds-Average Simulation（RAS）湍流模型来求解流体的平均行为以及统计波动。在这个算例演示中（雷诺数为 10e4 的顶盖驱动流），我们使用附带壁面函数的标准 k - ε 模型来进行求解。这会出现两个新变量：k，湍流动能场；ε，湍流动能耗散率场。所使用的求解器为 pisoFoam。

2.1.8.1 前处理

现在我们切换到 $FOAM_RUN/tutorials/incompressible/pisoFoam/RAS 的 cavity 算例下，复制这个算例并重新命名为 cavityRAS 算例，并切换到算例目录下：

```
run
cp -r $FOAM_TUTORIALS/incompressible/pisoFoam/RAS/cavity cavityRAS
cd cavityRAS
```

[31]通过 No Iterations 来查看

[32]在停止求解器计算之前，务必查看是否已经写入结果。因为当求解器没有写入结果的时候，它也可以被终止并不写入结果，这在大型算例中经常发生，因为大型算例写入场需要消耗的时间较长

[33]这种方法和 DNS 模拟的思路有所接近。OpenFOAM 中的 icoFoam 可以进行 Quasi-DNS 模拟，请参考：本链接

使用 blockMesh 生成网格。当使用附带壁面函数的标准 k -ε 模型的时，没有必要引入非均匀网格。原因为壁面附近的流型已经被模化而不是直接求解。

OpenFOAM 自带的壁面函数边界条件可以分别用于不同边界面[34]。不同的壁面可以使用不同的壁面边界条件。我们可以对湍流粘度场 ν_t 内进行修改以使用不同壁面函数，其位于 0/nut 文件夹内：

```
18  dimensions      [0 2 -1 0 0 0 0];
19
20  internalField   uniform 0;
21
22  boundaryField
23  {
24      movingWall
25      {
26          type            nutkWallFunction;
27          value           uniform 0;
28      }
29      fixedWalls
30      {
31          type            nutkWallFunction;
32          value           uniform 0;
33      }
34      frontAndBack
35      {
36          type            empty;
37      }
38  }
```

在这个算例中，通过把 movingWall 和 fixedWall 指定为 nutWallFunction 类型来使用标准壁面函数模型。其它的壁面边界模型，例如粗糙壁面边界类型，可以通过 nutRoughWallFunction 关键词来指定。

用户可以打开 k 和 ε 文件来查看它们的边界类型。对于壁面边界条件，ε 为 epsilonWallFunction 边界条件，k 为 kqRwallFunction 边界条件。后者是一个普适性边界条件，它可以用于任何的湍动能类场，例如 k、q、以及雷诺应力 R 场。最初的 k 和 ε 值通过对速度波动量，$\mathbf{U}'$，和湍流尺度量，l，进行预估来计算，k 和的 ε 计算公式如下：

$$k = \frac{1}{2}\overline{\mathbf{U}' \cdot \mathbf{U}'} \tag{2-8}$$

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} \tag{2-9}$$

其中，C_μ 是 k - ε 模型的标准参数，其值为 0.09。对于笛卡尔坐标体系，k 的计算公式为：

$$k = \frac{1}{2}\left(U_x'^2 + U_y'^2 + U_z'^2\right) \tag{2-10}$$

其中，$U_x'^2$，$U_y'^2$，$U_z'^2$ 是速度在 x，y，z 方向的脉动速度。假定其为各项同性湍流如 $U_x'^2$=$U_y'^2$=$U_z'^2$，其值为顶盖速度的 5%。湍流尺度为正方体宽度的 20%，0.1m。这样 k 和 ε 可以这样计算：

$$U_x' = U_y' = U_z' = \frac{5}{100}\mathrm{ms}^{-1} \tag{2-11}$$

$$k = \frac{3}{2}\left(\frac{5}{100}\right)^2 \mathrm{m}^2\mathrm{s}^{-2} = 3.75 \times 10^{-3}\mathrm{m}^2\mathrm{s}^{-2} \tag{2-12}$$

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} \approx 7.54 \times 10^{-4}\mathrm{m}^2\mathrm{s}^{-2} \tag{2-13}$$

这样我们就可以计算出 k 和 ε 的初始条件。速度和压力的边界条件跟之前的相同，为 (0,0,0) 以及 0。

[34]有关壁面函数请参考：本链接

目前存在很多的湍流模型，例如 RAS 湍流模型以及 LES 大涡模拟。它们都被植入进了 OpenFOAM。在大多数瞬态求解器中，湍流模型是在运行时进行选择的[35]。在求解器最开始运行的时候，它会读取 tubulenceProperties 字典文件里面的 simulationType 关键词来获取湍流模型的相关信息。用户可以在 constant 文件夹下面打开这个 tubulenceProperties 字典文件会发现以下信息：

```
18  simulationType        RAS;
19
20  RAS
21  {
22      RASModel          kEpsilon;
23
24      turbulence        on;
25
26      printCoeffs       on;
27  }
```

可选的 simulationType 有 laminar，RAS 以及 LES。这个算例中我们选择了 RAS，更进一步的 RAS 湍流模型在 RASProperties 字典文件中指定，位于 constant 文件夹中。8.2.1.1节列举了所有可选的 RAS 湍流模型。kEpsilon 对应的是标准 k -ε 模型；同时，用户应该确保 turbulence 设置为 on 的状态。

每个湍流模型的参数都存储在每个湍流模型的代码中并且具有一个默认值。printCoeffs 开关如果设置为 on 的话，这些参数的值会被输出到终端。它们以子字典的形式被输出，我们会看到每个模型的名字后面都加上了 Coeffs，例如 kEpsilon 模型会输出 kEpsilonCoeffs。这些模型的参数也可以通过在 RAS 子字典文件中自定义来修改。

接下来用户应该设置层流粘度。通过设置动力粘度为 10^{-5}，雷诺数可以达到 10^4，计算方法参考方程(2-1)。最终，用户应该设置 controlDict 里面的 startTime，stopTime，deltaT 以及 writeInverval。为了保证库郎数小于 1，我们把 deltaT 设置为 0.005 秒。结束时间设置为 10 秒。

2.1.8.2 运行

我们在终端键入 pisoFoam 命令来运行 pisoFoam 程序。在这个算例中，粘度非常低，顶盖附近的边界层会非常薄。又因为我们顶盖附近的网格比较糙。因此在这些网格中的速度要比顶盖的速度小得多。实际上，大约 100 个时间步之后，我们可以看出顶盖附近的网格单元速度的上限大约为 0.2m/s。因此最大库郎数不会超过 0.2。我们可以通过调节时间步来增大库郎数，并保持小于 1 即可。因此我们设置 deltaT 为 0.02。在这种情况下，设置 startFrom 为 latestTime。这表示 pisoFoam 会从最新的时间步开始读取并计算，例如在第 10 秒的时候。endTime 应该被设置为 20s，因为湍流算例比层流算例收敛要慢的多。在设置好之后，重新开始运行，监控残差，并查看结果检测算例是否已经达到稳态，或者算例的某个结果场达到了一种循环的状态。在后一种情况下，算例可能永远不会收敛，但这并不意味着计算的不精准。

2.1.9 改变算例几何

用户可能打算改变一下几何模型然后重新计算。在这种情况下，最好保留所有的计算过的文件，并在新算例中调用它们。然而复杂的是，在新的几何模型中，场数据可能不一样了。即便如此我们依然可以使用 mapFields 程序来映射场数据，它可以处理完全一样的几何或者不一样的几何，以及重合边界或者非重合边界。举例：我们在 icoFoam 目录中切换到 cavityClipped 算例，在这个算例中，几何模型和 cavity 算例是一样的，但是在底部的右边，有个边长为 0.04 米

[35]此处涉及到 OpenFOAM 求解器源代码中极为常用的 runTimeMechanism（运行时选择机制）

的正方形被移除。我们通过下面的命令来新增 cavityClipped 算例：

```
run
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity/cavityClipped
cd cavityClipped
```

下面是算例的 blockMeshDict 字典文件：

```
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
(
);
```

我们用 blockMesh 来生成网格，patches 参考之前算例的设定。为了在 mapFields 的过程中便于区分。我们在这个算例中，把原始 cavity 算例中的 movingWall 命名为 lid。

在非连续几何的映射中，不能保证所有场数据都能映射过来。在某些区域，原始场并不包含相

关数据，这些区域的数据还是被映射场的原始数据。因此在映射之前，在相应的时间步内就应该存在相应的场数据。在这个算例中，我们从 0.5 秒开始进行映射，因此在 controlDict 中设定 startTime 为 0.5s。因此，用户需要把 0 秒的场数据拷贝到 0.5s 中。我们使用以下命令：

```
cp -r 0 0.5
```

在进行映射之前，用户可以查看 0.5s 的场数据。

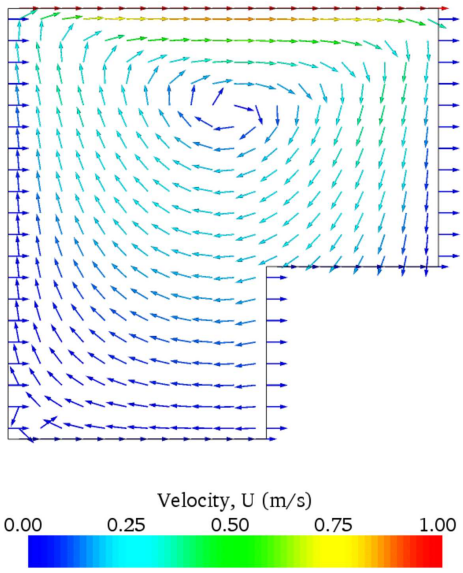


图 2-14: 将 cavity 算例的结果映射到 cavityClipped

现在我们打算把 cavity 的速度和压力场数据映射到 cavityClipped 算例中。因为映射是不连续的，因此需要编辑 system 文件夹下的 mapFieldsDict 文件。这个文件包含两个关键词，一个是 patchMap，另一个是 cuttingPatches。如果用户想把原始场的 patches 条件映射到被映射场的 patches 的时候，我们使用 patchMap。在 cavityClipped 算例中，我们的 lid 边界条件跟 cavity 算例的 movingWall 边界条件是一致的，我们可以这样来设置 patchMap：

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

当用户打算把原始场的内场数据映射到被映射场的边界的时候，我们使用 cuttingPatches。我们通过这个算例来演示这个操作，在这个算例中，因为 fixedWalls 的边界条件为 noSlip，因此我们不需要对 fixedWalls 边界的值进行映射。在这种情况下可以指定 cuttingPatches 为空：

```
cuttingPatches\\
(
);
```

如果用户打算把内部场的值映射到 fixedWalls 边界中，则需要在 fixedWalls 中指定 fixedValue 边界条件，其关键字 value 值在映射的过程中将会被改写。在这种情况下，需要在 cuttingPatches 里面指定 fixedWalls 关键词。现在用户可以在 cavityClipped 目录下执

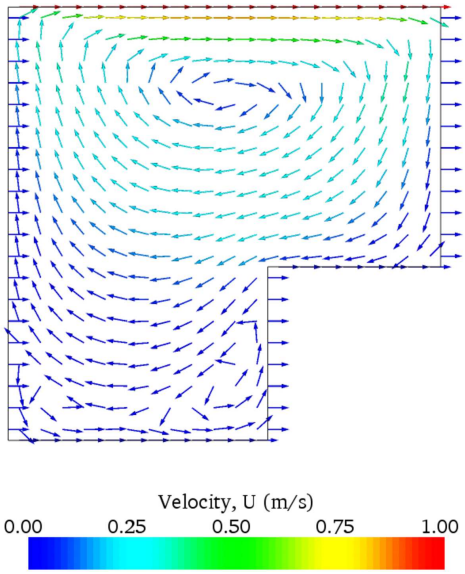


图 2-15: cavityClipped 的速度矢量图

行 `mapFields` 命令：

```
mapFields ../cavity
```

参见图2-14，这即为映射后的场数据。由于我们的 `cuttingPatches` 并没有指定 `fixedWalls` 关键词，因此 `fixedWalls` 的边界场信息并没有被映射，且其边界条件为 `noSlip`。现在用户可以用 `icoFoam` 重新运行这个算例。

2.1.10 后处理

速度场可以按前文所述方法进行后处理，我们可以对比 0.5s 的初始场和 0.6s 的最终场。另外，我们简单说一下对于 2D 算例的特殊处理使其更加漂亮。用户可以在 `Filter` 菜单中选择 `Extract Block` 滤镜，然后在 `Parameter` 面板高亮感兴趣的边界面，如 `lid` 和 `fixedWalls`。点击 `Apply`，然后在 `Display` 面板中选择 `Wireframe`，这些几何就会显示出来。参见图2-15。其中的各个面显示为黑色，我们也可以清楚地看出在这个几何模型下底部存在漩涡。

2.2 带孔盘体应力分析

这个算例我们讲解在一个中心带有圆形空洞的方盘上，如何进行前处理、线弹性稳态应力分析以及如何进行后处理。盘体的几何尺寸为：边长 4 米，中心的空洞半径 R 为 0.5 米。这个方盘左右面分别承受了一个均匀的 10kpa 的载荷，如图2-16所示。这个几何体上可以看出有两个对称平面。因此求解区域只需要覆盖整个几何体的四分之一，即2-16图中的阴影部分面积。

由于负载被附加于盘体的一个平面上，因此这个问题可以看做是二维的。在笛卡尔坐标系下面，关于第三个维度的行为存在两种假定：（1）应力平面条件：适用于在这个 2D 平面外的第三维度的应力是可忽略的情况；（2）应变平面条件：适用于在这个 2D 平面外的第三维度的应变是可忽略的情况。应力平面条件对于那些第三维度方向非常薄的算例是合适的。应变平面条件对于第三维方向比较厚的情况下适用。

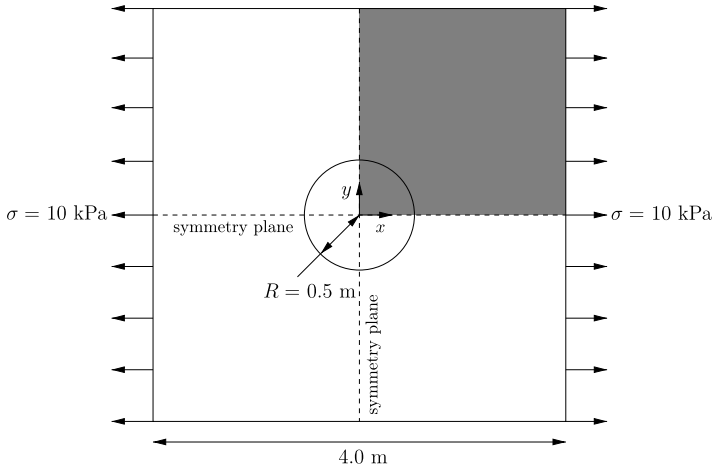


图 2-16: 二维空心盘体几何

对于一个无限大的中心带有圆形空洞的薄盘，它存在解析解。垂直于应力对称平面的解析解为：

$$(\sigma_{xx})_{x=0} = \begin{cases} \sigma\left(1+\frac{R^2}{2y^2}\frac{3R^4}{2y^2}\right) & |y| \geq R \\ 0 & |y| \leq R \end{cases} \tag{2-14}$$

在这个算例中，模拟的结果将和解析解相对比。在这个教程结束后，读者可以对网格无关性进行分析，或者增加盘子大小获得数值解来和解析解相对比。

2.2.1 网格生成

整个区域分为 4 个 block，其中一些具有曲边。xy 平面的 Block 结构如图2-16所示。正如2.1.1.1节所说，即使是一个 2D 算例，所有的几何在 OpenFOAM 里面都是三维的形式。因此 z 方向的厚度应该指定，例如 0.5 米。这个数值的大小并不影响求解。因为应力边界条件以单位面积应力来指定，因此与求解结果横截面的大小无关。

用户从 tutorials/stressAnalysis/solidDisplacemehtFoam 目录中进入 plateHole 算例，并且打开 constant/polyMesh/blockMeshDict 文件，如下所示：

```
17  convertToMeters 1;
18
19  vertices
20  (
21      (0.5 0 0)
22      (1 0 0)
23      (2 0 0)
24      (2 0.707107 0)
25      (0.707107 0.707107 0)
26      (0.353553 0.353553 0)
27      (2 2 0)
28      (0.707107 2 0)
29      (0 2 0)
30      (0 1 0)
31      (0 0.5 0)
32      (0.5 0 0.5)
33      (1 0 0.5)
34      (2 0 0.5)
35      (2 0.707107 0.5)
36      (0.707107 0.707107 0.5)
37      (0.353553 0.353553 0.5)
38      (2 2 0.5)
39      (0.707107 2 0.5)
40      (0 2 0.5)
41      (0 1 0.5)
42      (0 0.5 0.5)
43  );
44
45  blocks
```

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

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

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
            (
                (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
(
);
```

迄今为止，我们在之前的教程中，只制作了直边 `block`，现在我们需要制定曲边 `block`。这可以通过修改关键词 `edges` 下面的相关信息来完成，关键词 `edges` 列举了非直线 `block` 的边。`Edge` 下面的语句要确保每行信息以 `arc`，`simpleSpline`，`polyLine` 等开头，详述请参阅6.3.1节。在

这个例子中，所有的曲边都是圆形的[36]，因此可以用 arc 关键词来指定。后面的信息是弧边的起始点和终止点的标示，以及这个弧边通过的点的坐标。

blockMeshDict 里面的 block 具有方向且各不相同。如图2-17，0 block 的 x2 方向就是 4 block 的-x1 方向。这意味着当定义每个 block 内的网格数和节点分布的时候需要多加小心，以使它们互相匹配。

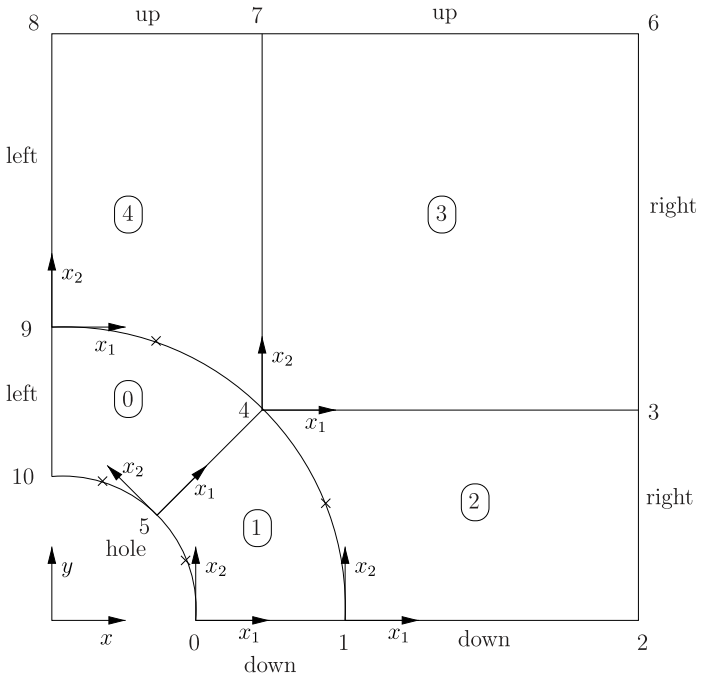


图 2-17: 中心空洞平板的 block 结构

6 个 patch 如下定义：平板的每个边为一个 patch，空洞为一个 patch，前后面各为一个 patch。左边和下面的 patch 是对称面。这个对称面是一个几何的对称面。因此它需要在网格里面进行定义而不是简单的定义一个初始边界条件。用户需要在 blockMeshDict 里面使用 symmetryPlane 关键词来定义。

frontAndBack，这个 patch 代表了这是一个 2D 的算例，在计算中这个 patch 将被忽略。再次提醒一下，这是一个几何方面的限制，因此需要在网格内进行定义。用户需要在 blockMeshDict 中将其定义为 empty。其它更多的边界条件以及几何限定信息，请查阅6.2节。

剩余的就是常规 patch 的设定，网格应该用 blockMesh 程序来生成，用 paraFoam 来查看，详述请查阅2.1.2节。如图2-18所示：

2.2.1.1 边界和初始条件

一旦网格生成后，必须指定边界条件的初始场。对于无热应力的单纯应力分析，只有位移量 D 需要设定。在 0/D 文件下面我们有：

```
17  dimensions      [0 1 0 0 0 0 0];
18
19  internalField   uniform (0 0 0);
20
21  boundaryField
```

[36]即半圆，或者圆的一部分。而非椭圆之类

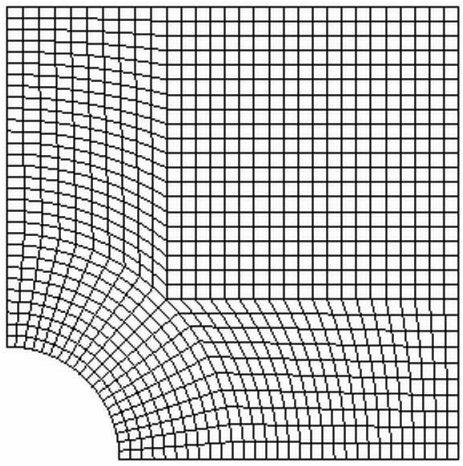


图 2-18: 中心空洞算例的网格

```
{
    left
    {
        type                symmetryPlane;
    }
    right
    {
        type                tractionDisplacement;
        traction            uniform ( 10000 0 0 );
        pressure            uniform 0;
        value               uniform (0 0 0);
    }
    down
    {
        type                symmetryPlane;
    }
    up
    {
        type                tractionDisplacement;
        traction            uniform ( 0 0 0 );
        pressure            uniform 0;
        value               uniform (0 0 0);
    }
    hole
    {
        type                tractionDisplacement;
        traction            uniform ( 0 0 0 );
        pressure            uniform 0;
        value               uniform (0 0 0);
    }
    frontAndBack
    {
        type empty;
    }
}
```

首先，位移量初始条件设定为 (0 0 0) 米。left 以及 down 这两个 patch 必须设定为 symmetryPlane,因为它们在网格中就被设定了,这可以打开 constant/polyMesh/boundary 文件确认。frontAndBack 设定为 empty。

其它 patches 是应力边界条件，应力边界条件通过 traction 字符来指定。它由以下两个关键词组合来构成：（1）在关键词 traction 下指定应力边界矢量和大小。（2）在 pressure 关键词下，如果这个边界面法向应力指向表面之外，就被定义为负值，因为它的 up 和 hole 这两个面牵引力为 0，因此边界牵引力和压力设定为 0。对于 right 边界牵引力应该为 (1e4 0 0)Pa，pressure 应该为 0 Pa。

2.2.1.2 物理特性

这个算例的物理特性放置在 constant 文件夹下的 mechanicalProperties 文件夹下面。我们需要指定钢材料的物理特性，见表2-1。在物理特性字典中，用户需要指定 planeStress 为

yes。

物性	单位	关键词	值
密度	kg/m^3	rho	7854
杨氏模量	Pa	E	2×10^{11}
泊松比	-	nu	0.3

表 2-1: 钢的机械特性

2.2.1.3 热物理特性

在 solidDisplacementFoam 求解器中可以计算温度场 T，因此可以选择求解和动量方程耦合（其通过生成的热应力来耦合）在一起的热物理方程。运行之前，用户可以通过在 thermalProperties 字典下的 thermalStress 开关来指定是否求解热物理方程。这个字典同样设置了这个算例的热物理特性，例如钢的热物理特性，见表2-2。

在此例中我们不想求解热物理方程，因此必须在 thermalProperties 字典中设定 thermalStress 关键词为 no。

2.2.1.4 控制

正如之前所说，求解的控制信息在 controlDict 字典中。对于这个算例，开始时间是 0 秒。由于这是一个稳态问题因此时间步无关紧要[37]；在这个情况下，最好设定时间步长 deltaT 为 1，

物性	单位	关键词	值
热容	J/kgK	C	434
热导	W/mK	K	60.5
热膨胀系数	/K	alpha	1.1×10^{-5}

表 2-2: 钢的热物理特性

在稳态问题中，它表示迭代步。终止时间 endTime 设定为 100，即最高迭代数。写入控制 writeInterval 设定为 20。

controlDict 的信息如下：

```
18  application             solidDisplacementFoam;
19
20  startFrom               startTime;
21
22  startTime               0;
23
24  stopAt                  endTime;
25
26  endTime                 100;
27
28  deltaT                  1;
29
30  writeControl            timeStep;
31
32  writeInterval           20;
33
34  purgeWrite              0;
35
36  writeFormat             ascii;
37
38  writePrecision          6;
```

[37]在 OpenFOAM 中，如果求解器采用的是 SIMPLE 算法（稳态），那么 deltaT 设定为多少是无关紧要的。如果其设定为 1，那么运行 100 次的时候即为迭代 100 次，如果设定为 0.1，运行 100 次的时候即为迭代 1000 次。为了简化，我们在使用 SIMPLE 求解器的 control 字典中，通常设为 1

```
39
40  writeCompression        off;
41
42  timeFormat              general;
43
44  timePrecision           6;
45
46  graphFormat             raw;
47
48  runTimeModifiable       true;
```

2.2.1.5 离散格式和求解器控制

现在，我们来分析一下 fvSchemes 字典文件。首先，我们分析的是一个稳态问题，因此用户应该在 timeScheme 的时间离散里面选择 SteadyState。这就屏蔽掉了时间离散项。在流体力学中，并不是所有的求解器，可以既用稳态又用瞬态来进行计算。但是 solidDisplacementFoam 可以既计算稳态又计算瞬态，因为对于这个求解器，两种类型的算法大体相同。

应力分析[38]的动量方程包含几个含有位移梯度量的显性项。梯度项计算的精准性和光滑性对结果有很大影响。一般情况下，有限体积离散建立于高斯定律之上，高斯定律对于大部分的模拟目的来说是足够精准的。但在这个算例中，我们使用最小二乘法[39]。因此，用户应该打开 system 文件夹下的 fvSchemes 字典，确保 gradSchemes 字典下 grad（U），grad（T）采用 leastSquares 关键词：

```
18  d2dt2Schemes
19  {
20      default             steadyState;
21  }
22
23  ddtSchemes
24  {
25      default             Euler;
26  }
27
28  gradSchemes
29  {
30      default             leastSquares;
31      grad(D)             leastSquares;
32      grad(T)             leastSquares;
33  }
34
35  divSchemes
36  {
37      default             none;
38      div(sigmaD)         Gauss linear;
39  }
40
41  laplacianSchemes
42  {
43      default             none;
44      laplacian(DD,D)     Gauss linear corrected;
45      laplacian(DT,T)     Gauss linear corrected;
46  }
47
48  interpolationSchemes
49  {
50      default             linear;
51  }
52
53  snGradSchemes
54  {
55      default             none;
56  }
```

system 文件下面的 fvSolution 字典文件控制求解线性方程组使用的矩阵求解器。用户应该首先查看 solvers 子字典，会留意到 D 的矩阵求解器为 GAMG[40]。求解器的 tolerance 是 10^{-6}。矩阵求解器的相对误差由 relTol 控制，控制每次迭代的残差减小量。本例中设置一个很小的残差是不经济的，因为方程组中很多项是显性的，并且采用分离迭代求解技术。因此，合理的迭代残差为 0.01，更高也是可以的，比如 0.1，在某些例子中，0.9 也是可以的（比如这个例子）：

[38] OpenFOAM 中采用有限体积法来进行应力分析

[39] 计算梯度的 least squares 对网格较差的算例较好

[40] 有关多重网格矩阵求解策略：本链接

```
18  solvers
19  {
20      "(D|T)"
21      {
22          solver                      GAMG;
23          tolerance                   1e-06;
24          relTol                      0.9;
25          smoother                    GaussSeidel;
26          nCellsInCoarsestLevel       20;
27      }
28  }
29
30  stressAnalysis
31  {
32      compactNormalStress     yes;
33      nCorrectors             1;
34      D                       1e-06;
35  }
```

fvSolution 字典下面包含一个子字典：stressAnalysis。它包含了这个求解器所需的控制参数。首先，nCorrectors 表示整个方程组求解的外循环数，包括每个时间步的拉伸边界条件。由于这是一个稳态问题，我们用时间步代表迭代数。因此我们可以设置 nCorrectors 为 1。

关键词指定外循环的收敛残差，例如：设置一个残差，当达到这个值的时候迭代停止。这个残差，本例为 10^{-6}，应该在运行之前就设定好。

2.2.2 运行

用户应该在后台运行程序，这样就可以在 log 文件中查看收敛信息。运行方式如下：

```
solidDisplacementFoam > log &
```

用户应该在 log 文件中查看收敛信息，它包括了迭代数、每步迭代的初始残差和最终残差。依靠这个残差设定，最终残差应该总是小于最初残差的 0.9 倍。一旦初始残差小于收敛残差 10^{-6} 的时候，程序收敛，用户可以终止程序。

2.2.3 后处理

对于后处理，我们可以参考2.1.4节的方法来进行，OpenFOAM 经常采用数学符号的名字来表示变量，solidDisplacementFoam 求解器以对称张量场 sigma 的形式输出应力场 σ，这和 OpenFOAM 的通常做法是一致的。对于希腊字母，通常按照发音来命名。

如果处理张量的某个分量（张量的分量为一个标量场）：例如：σ_{xx}、σ_{xy} 等，可以由 postProcess 程序来计算，如2.1.5.7节所说，在这个算例中我们采用这个命令：

```
postProcess -func 'components(sigma)'
```

张量分量被命名为 sigmaxx，sigmaxy，并且写在了每个时间步内。下图为在 paraFoam 中的显示，见图2-19：

我们想把方程2-14的解析解和我们的数值解相对比。因此我们必须把计算域中对称面的左边的数据输出出来。用户可以通用 postProcess 并附加 singleGraph 函数来获得。需要注意的是，之前运行的 postProcess 并不需要字典文件，但是本算例需要提供 postProcess 读取的字典文件。在此算例中，system 文件夹下包含了 singleGraph 文件。其中 sets 关键词指定的 sample line 为（0.0, 0.5, 0.25）到（0.0, 2.0, 0.15）之间的线段，提取的场在 fields 关键词内指定：

```
 9  start    (0 0.5 0.25);
10  end      (0 2 0.25);
```

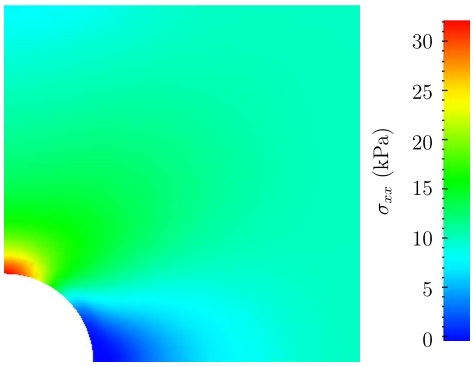


图 2-19: 中心空洞盘体的 σ_{xx} 场

```
11  fields   (sigmaxx);
12
13  #includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"
14
15  setConfig
16  {
17      axis    y;
18  }
19
20  // Must be last entry
21  #includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
```

用户应该执行下述命令：

```
postProcess -func 'singleGraph'
```

其写入格式为 raw，并提取出 2 列的原始数据。数据随后被写入 postProcessing/singleGraph 下的每个时间步文件中。例如：100s 的场数据存储在 singleGraph/100/leftPatch_sigmaxx.xy 中。依靠 GnuPlot[41]，用户可以使用如下命令来把精确解和数值解画在同一个图中：

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

图表请参阅图2-20。

2.2.4 练习

用户可以通过下面的练习来熟悉 solidDisplacementFoam：

2.2.4.1 增加网格数量

在 x、y 方向增加网格数量。参考2.2.3节，使用 mapFields 来把粗网格结果映射到细网格上并作为初始条件。

2.2.4.2 引入网格非均匀化

如果我们使用非均匀网格，会使靠近空洞的网格比远处网格更加细密。我们可以来重新设计网格，使得相邻网格的大小比小于 1.1。这使得相邻块网格之间的网格比例很相近。网格非均匀化

[41]一个软件，需要下载安装后，键入 gnuplot 来进入界面

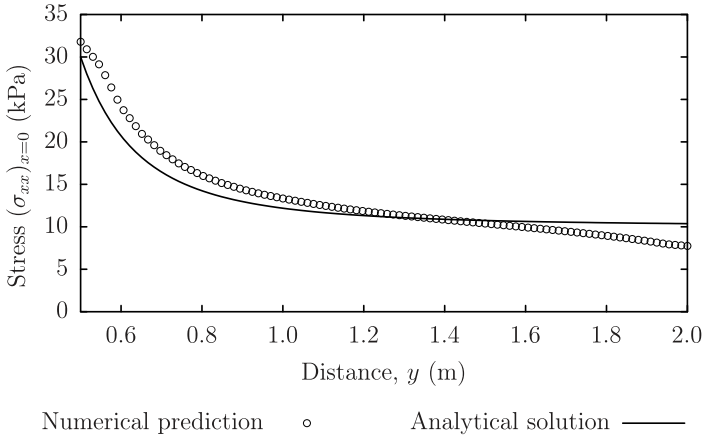


图 2-20: x=0 处的垂直平面法向应力分布

在2.1.6节有提及。我们还可以使用 `mapFields` 把2.2.3节使用的粗网格的结果映射到非均匀化网格的初始场上，对比这些结果，或者和分析解相对比。用非均匀化网格方法计算的结果会比之前的结果好么？

2.2.4.3 改变平板尺寸

解析解是针对一个无限大的带有空洞的二维平板，因此这个对于有限大小的平板来讲不是特别精准。为了估算误差，可以在保持空洞大小不变的情况下增加原盘大小来试试。

2.3 溃坝

在本教程中我们将用 `interFoam` 求解一个简化的二维溃坝问题[42]。该问题为两种液体被一种尖锐的界面（或自由表面）分隔的瞬态流动。`interFoam` 中的两相算法基于流体的体积分数（`VOF`）法，在该方法中，每个网格中的相体积分数（或相分数 `alpha`）通过求解一个组分输运方程确定。物理属性则基于这个相分数通过加权平均计算。`VOF` 方法的本质意味着组分间的界面不是计算出来的，而是作为相分数场的一个属性表现出来[43]。由于相分数可以为 0 和 1 之间的任何值，所以相界面并没有被严格定义，因此相界面表示它应存在的那些网格单元。

算例的物理过程如下：设置为一个静止的水柱，使其位于水箱左侧，在水箱的底部有一个小的障碍，在 $t = 0$s 时刻，让水柱自由流动，然后会产生水柱崩塌。在崩塌的过程中，水撞击水箱底部的一个障碍形成复杂的流场结构，其中包括若干被水包裹的气泡。几何和初始设置如图2-21所示：

2.3.1 生成网格

用户应该把 `$FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak` 文件夹复制到 `run` 文件夹下：

```
run
cp -r $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak .
```

[42]有关 interFoam 求解器请参考本链接

[43]VOF 的界面是由后处理得出来的结果，详细的 VOF 模型请参考其它文档

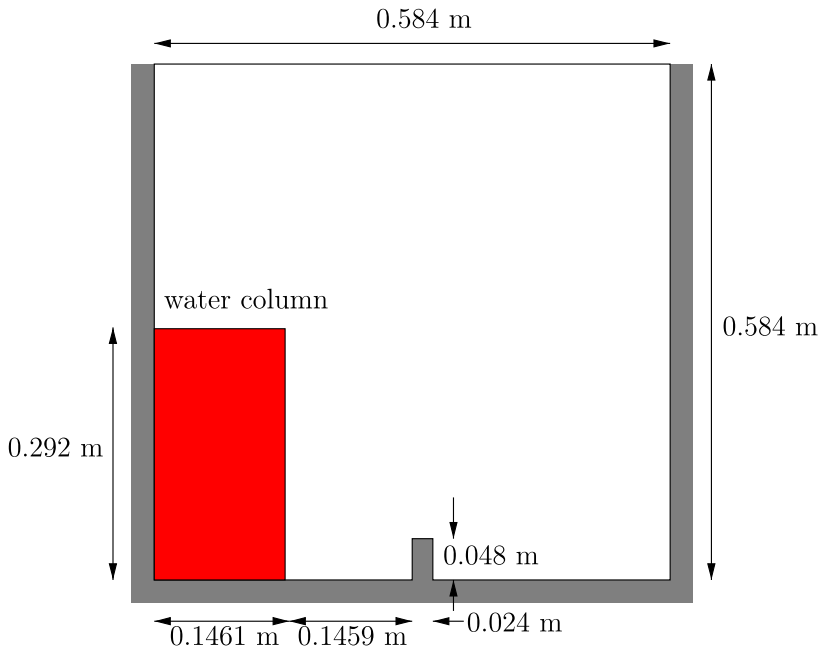


图 2-21: 溃坝的几何场

然后进入 damBreak 算例，根据前面所描述的运行 blockMesh 命令生成网格。damBreak 的网格包含了 5 个 block，其 blockMeshDict 的相关信息如下：

```
17  convertToMeters 0.146;
18
19  vertices
20  (
21      (0 0 0)
22      (2 0 0)
23      (2.16438 0 0)
24      (4 0 0)
25      (0 0.32876 0)
26      (2 0.32876 0)
27      (2.16438 0.32876 0)
28      (4 0.32876 0)
29      (0 4 0)
30      (2 4 0)
31      (2.16438 4 0)
32      (4 4 0)
33      (0 0 0.1)
34      (2 0 0.1)
35      (2.16438 0 0.1)
36      (4 0 0.1)
37      (0 0.32876 0.1)
38      (2 0.32876 0.1)
39      (2.16438 0.32876 0.1)
40      (4 0.32876 0.1)
41      (0 4 0.1)
42      (2 4 0.1)
43      (2.16438 4 0.1)
44      (4 4 0.1)
45  );
46
47  blocks
48  (
49      hex (0 1 5 4 12 13 17 16) (23 8 1) simpleGrading (1 1 1)
50      hex (2 3 7 6 14 15 19 18) (19 8 1) simpleGrading (1 1 1)
51      hex (4 5 9 8 16 17 21 20) (23 42 1) simpleGrading (1 1 1)
52      hex (5 6 10 9 17 18 22 21) (4 42 1) simpleGrading (1 1 1)
53      hex (6 7 11 10 18 19 23 22) (19 42 1) simpleGrading (1 1 1)
54  );
55
56  edges
57  (
58  );
59
60  boundary
61  (
62      leftWall
63      {
64          type wall;
65          faces
66              (
67                  (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 边界条件

用户可以通过查看 constant/polyMesh 文件夹下的 boundary 文件来检查 blockMesh 命令生成的几何边界。该文件包含一个由 5 个 patch 构成的列表：leftWall，rightWall，lowerWall，atmosphere 和 defaultFaces。用户应该注意各个 patch 的 type 关键字。其中 atmosphere 是一个标准的 patch，也就是没有特别的属性，只不过是一个可以在其上面定义边界条件的实体。由于 defaultFaces 的法向垂直于我们的求解平面，因为本算例是个 2D 算例，所以我们对其并不求解，因此 defaultFaces 的类型是 empty。而 leftWall，rightWall 和 lowerWall 各自的类型都是 wall。

像简单的 patch 一样，wall 类型不包含关于网格的任何几何或拓扑信息，它和简单 patch 唯一的不同之处在于它是一个壁面，这意味着可以在其上应用壁面函数模型或者其他有关壁面的模型。例如在 interFoam 求解器中，可以通过壁面依附模型来附加壁面表面张力的影响。该模型通过将 alpha 场的边界条件设置为 constantAlphaContactAngle 来实现。使用这个边界条件，用户必须给出关键词静态接触角 theta0 的值。

在本算例中，我们将会忽略界面和壁面间的表面张力效应。这可以通过设定静态接触角 $\theta = 90$ 度来实现。不过在这里我们可以选择更简单的方法：即设置 alpha 的边界条件为 zeroGradient，而不是用 constantAlphaContactAngle 边界条件。

本算例的上边界 top 与大气环境自由相通，因此其允许流动的出入。我们可以用速度和压力边界条件的组合来实现这个目的，并同时保证数值稳定性。对于压力、速度以及相分数它们分别是：

totalPressure 一种 fixedValue 条件，利用指定的总压 p0 和局部速度 U 计算获得；

pressureInletOutletVelocity 对所有分量应用 zeroGradient 条件，当流动为入流时，对边界切向的分量应用 fixedValue 条件；

inletOutlet 出流时为 zeroGradient 条件，入流时则为 fixedValue 条件。

在所有壁面边界处,压力场采用 fixedFluxPressure 边界条件。在某些包含重力以及表面张力的求解器中，这个边界条件调节压力梯度以使得边界的通量和速度边界条件相匹配。defaultFaces 代表此二维问题的前后面，像往常一样，它们是 empty 类型。

2.3.3 设置初始场

与以往的例子不同，我们现在应为相分数 α_{water} 指定一个非均匀的初始条件，其中

$$\alpha_{water} = \begin{cases} 1 \\ 0 \end{cases} \tag{2-15}$$

这可以通过运行 setFields 工具实现。它需要一个 setFieldsDict 字典，其位于 system 文件夹，本算例的相关信息如下：

```
18  defaultFieldValues
19  (
20      volScalarFieldValue alpha.water 0
21  );
22
23  regions
24  (
25      boxToCell
26      {
27          box (0 0 -1) (0.1461 0.292 1);
28          fieldValues
29              (
30                  volScalarFieldValue alpha.water 1
31              );
32      }
33  );
```

其中 defaultFieldValues 关键字用来设置场的默认值，也就是说这个值将在 regions 子字典中指定的区域以外采用。这个 regions 子字典包含一系列子字典，它包括要在指定区域设定某个场的关键词 fieldValues。我们用一些点、单元或面的集合，通过拓扑约束来建立一个 topoSetSource 区域。在这里，boxToCell 通过定义一个最小和最大的矢量来创建一个盒子区域，在此区域内为水相。该区域内相分数 α_{water} 被设置为 1。

setFields 工具从文件中读取场并重新定义这些场，然后把它们写入文件，原始的文件将会被覆盖，因此我们建议在执行 setFields 前先进行备份。在 damBreak 例子中，场 alpha.water 最初通过存为一个名为 alpha.water.orig 的文件来备份。在 OpenFOAM 中，所有以 orig 为结尾的文件都是备份文件，可用于读取。在执行 setFields 的时候，直接读取 alpha.water.orig 并写为 alpha.water（如果打开“压缩文件”选择，则写为 alpha.water.gz）。在这种情况下，alpha.water.orig 和 alpha.water 同时存在，且 alpha.water.orig 可以被再利用。

用户接下来应在执行其它任何工具之前执行

```
setFields
```

然后利用 paraFoam 检查 alpha.water 的初始场是否符合图2-22所示的分布。

2.3.4 流体特性

让我们看一下 constant 文件夹下的 transportProperties 文件。该字典包含流体的物质属性，分别列在 water 和 air 两个子字典里。各相的传递模型通过关键字 transportModel 进行选择。用户应选择 Newtonian，此时运动粘度为常数且通过关键字 nu 指定。其它模型如 CrossPowerLaw 的粘性参数的指定可以参考第8.3节。密度在关键字 rho 下指定。

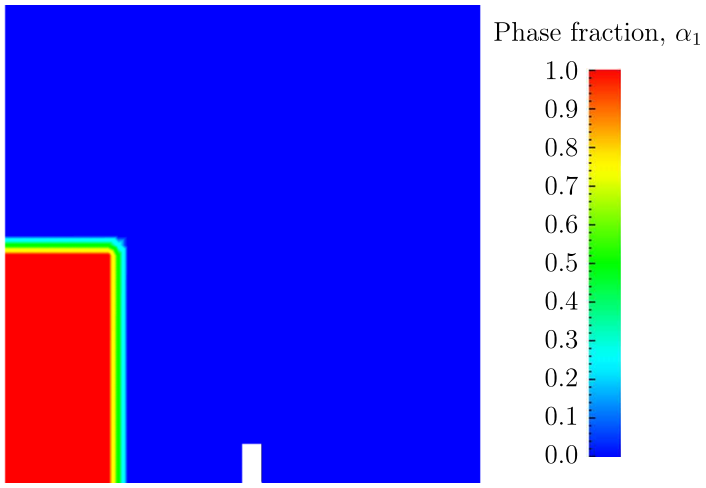


图 2-22: 相分数 alpha.water 的初始场

水的物性			
运动粘度	m^2s^{-1}	nu	1.0×10^{-6}
密度	kg/m^3	rho	1.0×10^{3}

空气的物性			
运动粘度	m^2s^{-1}	nu	1.48×10^{-5}
密度	kg/m^3	rho	1.0

两相特性			
表面张力	N/m	sigma	0.7

表 2-3: 溃坝的流体特性

两相间的表面张力通过关键字 `sigma` 指定。本例中使用的参数列于表2-3中。

重力加速度在整个求解域内统一分布，通过 `constant` 文件夹内名为 `g` 的文件来指定。与普通的场文件 (如 `U` 和 `p`) 不一样，`g` 是 `uniformDimensionedVectorField`（均一的带有量纲的矢量场）类型，因此只是简单的包含一个 `dimensions` 关键字和一个 `value` 关键字，如下所示本例中为 (0,-9.81, 0) m/s^2：

```
18  dimensions      [0 1 -2 0 0 0 0];
19  value           ( 0 -9.81 0 );
```

2.3.5 湍流模型

如 `cavity` 算例一样，湍流模型的选择是在运行时读取字典 `turbulenceProperties` 里的 `simulation-Type` 关键字来进行。在本例中，我们希望运行时不求解湍流模型，于是我们设置为 `laminar`：

```
18  simulationType laminar;
```

2.3.6 时间步长控制

时间步长控制在自由表面追踪问题中是一个重要的问题，因为表面追踪算法比标准流体算法对库郎数更为敏感。若采用显性 MULES 算法，在界面区域库郎数不应超过上限如 0.25。若在 MULESCorr 中指定采用半隐型 MULES 算法，理论上库郎数对相方程求解没有影响，因此库郎数只取决于时间精度。

在某些速度很容易预测的算例中，用户最好指定一个满足 Co 条件的固定时间步长。然而对于更为复杂的算例，这却相当困难。因此 interFoam 求解器在 controlDict 中提供了自动调整时间步长。用户需指定 adjustTimeStep 为 on，并设置最大库郎数 maxCo 为和最大相场的库郎数 maxAlphaCo 为 1.0。时间步长的上限 maxDeltaT 可以设置为一个不会超过的值，比如 1.0。

通过时间步长的自动控制，时间步本身就是一个不确定的时间步，其具有随机性。因此，如果我们要求 OpenFOAM 按照一个固定的时间步间隔保存结果，会发现保存结果的时间步会比较乱。值得一提的是，即使采用自动调整时间步，OpenFOAM 也允许用户在指定的时间步来保存数据，在这种情况下，OpenFOAM 在那一时间点强制调整时间步长，以使运算的时间正好“碰上”所指定的那些输出结果的时间。用户可以通过指定 controlDict 字典里 writeControl 关键字为 adjustableRunTime 来实现。controlDict 字典的相关信息应为：

```
18  application             interFoam;
19
20  startFrom               startTime;
21
22  startTime               0;
23
24  stopAt                  endTime;
25
26  endTime                 1;
27
28  deltaT                  0.001;
29
30  writeControl            adjustableRunTime;
31
32  writeInterval           0.05;
33
34  purgeWrite              0;
35
36  writeFormat             binary;
37
38  writePrecision          6;
39
40  writeCompression        off;
41
42  timeFormat              general;
43
44  timePrecision           6;
45
46  runTimeModifiable       yes;
47
48  adjustTimeStep          yes;
49
50  maxCo                   1;
51  maxAlphaCo              1;
52
53  maxDeltaT               1;
```

2.3.7 离散格式

interFoam 求解器在求解相场的时候采用了 Henry Weller 开发的 MULES 方法[44]，这个方法可以保证在任何数值格式、网格类型的情况下的相场有界。因此在使用 interFoam 求解器的算例中，对流项格式的选择不再局限于那些稳定且有界的格式，如迎风格式。

对流项格式设置在 fvSchemes 字典文件的子字典 divSchemes 中。在本例中，动量方程的对流项为 $\nabla\cdot(\rho\mathbf{UU})$，在字典中体现为关键字 div(rhophi,U)，我们采用 Gauss linearUpwind grad(U) 以实现良好的精度。通常，有界线性格式需要一个系数 ∅。这里，我们选择 ∅=1.0 以实现最好的稳定性。关键字 div(phi,alpha) 代表的 $\nabla\cdot(\mathbf{U}\alpha_l)$ 项采用 vanLeer 格式。因为有界性

[44]MULES（multi-dimensionsal limiter for explicitsolution）为 Henry Weller 基于 Flux-correct transport 理论提出的用于求解相方程时保持相分数有界的限制器

已由 MULES 算法保证，因此关键字 `div(phirb,alpha)` 代表的 $\nabla\cdot(\mathbf{U}_r\alpha_1)$ 项可采用二阶精度的 `linear`（中心）格式。

其它离散项采用常用的格式，因此 `fvSchemes` 字典的相关信息应为：

```
18  ddtSchemes
19  {
20      default         Euler;
21  }
22
23  gradSchemes
24  {
25      default         Gauss linear;
26  }
27
28  divSchemes
29  {
30      div(rhoPhi,U)                   Gauss linearUpwind grad(U);
31      div(phi,alpha)                  Gauss vanLeer;
32      div(phirb,alpha)                Gauss linear;
33      div((muEff*dev(T(grad(U)))))    Gauss linear;
34  }
35
36  laplacianSchemes
37  {
38      default         Gauss linear corrected;
39  }
40
41  interpolationSchemes
42  {
43      default         linear;
44  }
45
46  snGradSchemes
47  {
48      default         corrected;
49  }
```

2.3.8 矩阵求解器控制

在 `fvSolution` 中，`solvers` 中的 `alpha.water` 子字典中包含了 `interFoam` 特定的一些参数，主要是 `nAlphaSubCycles` 和 `cAlpha`。`nAlphaSubCycles` 代表 α 方程中子循环的数目；子循环是在一个给定时间步内对一个方程的附加求解[45]。它可以在不降低时间步长的情况下，保证解的稳定，并使得可求解的时间变的更长。在这里我们指定为 2，这意味着 α 方程在各个实际的时间步内以半个实际时间步长的步长求解了两次。

关键字 `cAlpha` 是一个控制界面压缩的因子，其中 0 相当于无压缩，1 相当于守恒压缩；并且，任何大于 1 的数表示强化界面压缩。我们通常建议取 1，就如本例一样。

2.3.9 运行程序

运行程序已经在前面的教程里详细介绍过。现在尝试下面的命令，采用 `tee` 命令将程序输出写到标准输出和文件：

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

现在程序会交互式地运行[46]，并且备份输出到文件 `log`。

[45] 调整 nAlphaSubCycles 的值会在求解压力之前多次求解相方程，调节 nAlphaCorr 会在当前时间步内对相场和压力场的循环做多次求解，更多信息请查阅 OpenFOAM 有关 interFoam 求解器的介绍

[46] 终端也会显示

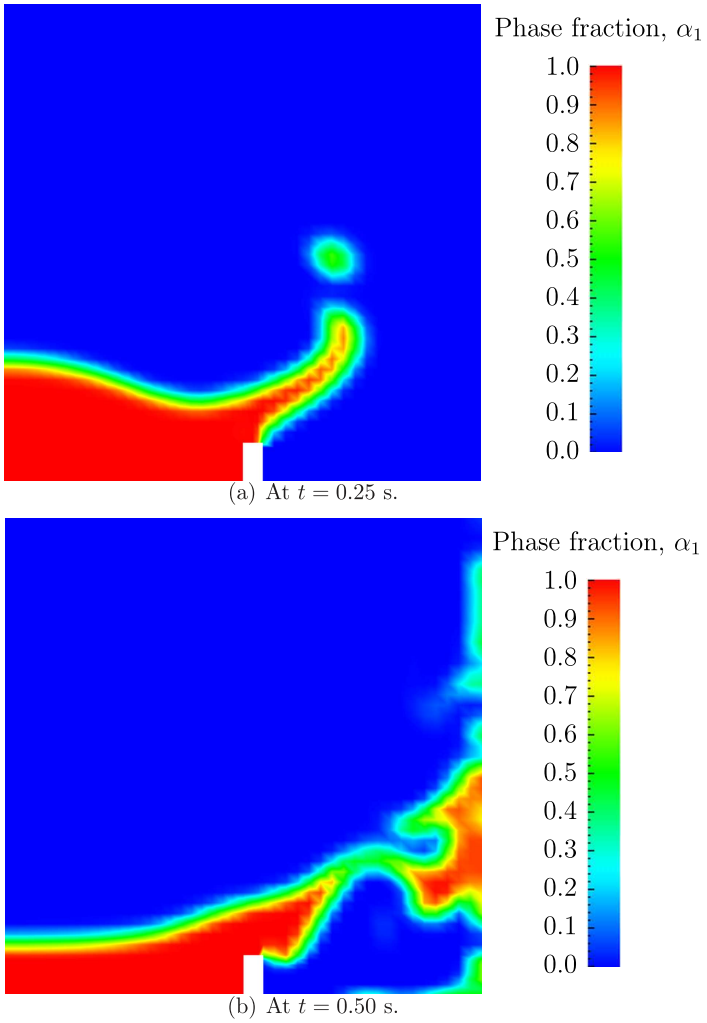


图 2-23: 相分数场 α

2.3.10 后处理

现在可按照之前的方法对结果进行后处理。用户可以监控相分数 `alpha.water` 随时间的发展变化，如图2-23。

2.3.11 并行运行

前面的溃坝算例是使用一个相当粗糙的网格来生成的。我们现在希望增加网格数量并重新运行算例。新的算例如果采用单核计算会运行若干小时。因此，如果用户能够使用多核运算，这样会大大减少运行时间。我们将介绍 OpenFOAM 的并行处理能力。用户首先应复制一份 `damBreak` 算例，执行以下命令：

```
run
foamCloneCase damBreak damBreakFine
```

然后，进入新算例的文件夹并将 `blockMeshDict` 字典中的 `blocks` 修改为：

```
blocks
```

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

在这里，用户必须改变网格密度，比如把其中 3 个方向的的网格数目改为（46 10 1），网格非均匀关键词也应该适当修改例如改为（1 2 1）。一旦这些信息修改正确，运行 blockMesh 命令即可生成网格。

由于现在的网格与之前的 damBreak 算例网格相比有变化，因此用户必须要重新初始化 0 文件夹里的相场 alpha.water，因为在新的网格下，alpha.water 的内部场有很大的变动。我们没有必要改变 U 和 p_rgh，因为它们被指定为 uniform，独立于网格数量。我们希望初始化相分数场使之含有尖锐界面，也就是它的元素只含有 α=1 或 α=0。如果通过 mapFields 映射场可能会在界面处产生插值，即 0<α<1 的情况，所以最好重新运行 setFields 工具。

目前，0 文件夹下的 alpha.water.gz 文件中的网格编号与新生成的网格不相符，这会产生错误。因此，用户需要删除掉这个文件后重新运行 setFields：

```
rm 0/alpha.water.gz
setFields
```

OpenFOAM 所采用的并行计算方法称为区域分解法，该方法中，几何和相关的场被分割为若干部分并被分配到不同的处理器进行求解。因此并行运行一个算例的第一步是利用 decomposePar 工具分解求解域。算例的 system 目录下有一个与 decomposePar 程序相关的字典文件，其名为 decomposeParDict。同其它许多工具一样，它可以在特定工具的源码目录里找到一个默认的字典文件，本例中的字典文件就位于 $FOAM_UTILITIES/parallelProcessing/decomposePar 目录下。

第一项是 numberOfSubdomains，它指定了要将算例分割成为的子区域数量，通常与可用于运算此算例的处理器数一样。

本例中，区域分割的方法我们选择为 simple，其对应的 simpleCoeffs 应按如下条件进行编辑。求解域在 x，y 和 z 方向被分割为子块（子区域），各个方向子区域的数量由矢量 **n** 给定。由于本算例的几何是二维的，第 3 个方向，z 方向，不能被分割，所以必须等于 1。**n** 矢量的 n_x 和 n_y 分量在 x 和 y 方向分割求解域，必须指定来使由 n_x 和 n_y 决定的子区域的数量与由 numberOfSubdomains 指定的相同，也就是 $n_x \times n_y$= numberOfSubdomains。我们最好使每个子区域的网格连接面数量最少[47]。对于这个正方形几何，最好保持 x 和 y 方向的分割数量相等且为偶数。关键字 delta 应设置为 0.001。

举例来说，假设我们希望用 4 个处理器运行程序，应设置 numberOfSubdomains 为 4 且 **n**=(2, 2, 1)。用户直接在终端键入

```
decomposePar
```

即可对网格进行分解，我们可以从屏幕消息中看到求解域被平均分配在各个处理器之中。

关于怎么并行运行一个算例的详细介绍读者应参看3.4节，本例中我们仅仅是举一个并行运算的例子。我们使用 openMPI 作为 MPI 的实现。如果用户想测试这个例子，可以在本地主机上并行运行，通过以下命令实现：

[47]OpenFOAM 提供了不同的分解类型，某些分解方法的目的即为分块后的网格连接面最少，例如 scotch 分解方法，有关并行算法可参考本链接

```
mpirun -np 4 interFoam -parallel > log &
```

用户也可以在网络上的多个节点间运行，如3.4.3节中介绍的那样，通过创建一个文件，列出算例将要在其上运行的主机的主机名等来进行设置，算例应在后台运行，用户可以像往常一样通过监测 `log` 文件跟踪其进度。

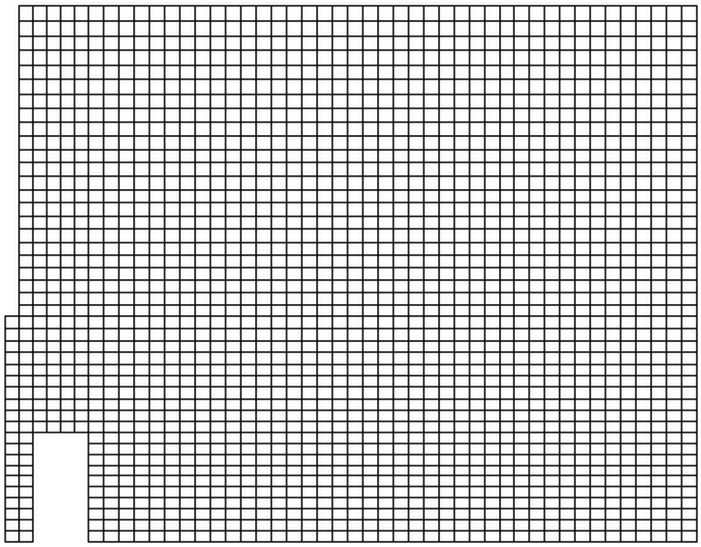


图 2-24: 并行处理算例中处理器 2 的网格

2.3.12 算例的并行后处理

一旦算例运行结束，为了使用其它软件来进行后处理，用户必须使用 `reconstructPar` 工具将分割后的场和网格重新合并。简单地以命令行的方式来运行它就可以。细网格的结果如图2-25所示，用户可以看到界面的分辨率比较粗的网格明显提高了很多。

用户也可以单独地对分割后的求解域的一部分进行后处理，思想即为：简单地将各处理器文件夹看做一个独立的算例。用户可以通过以下的命令来运行 `paraFoam`：

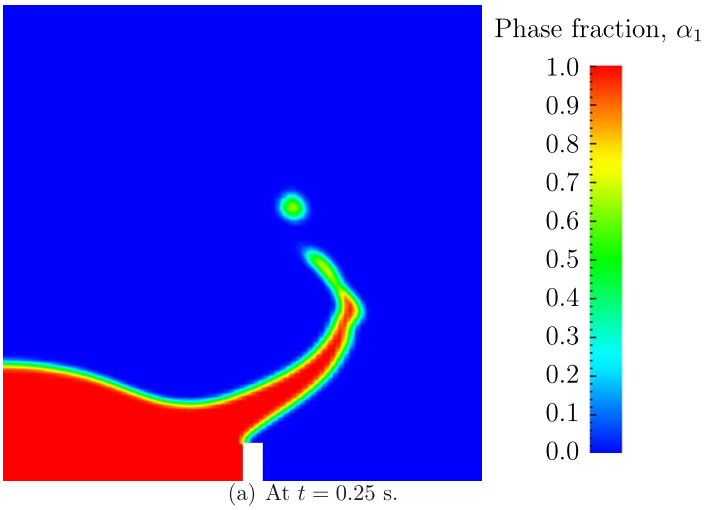
```
paraFoam -case processor1
```

于是 `processor1` 将会以算例模块出现在 ParaView 中。图2-24显示了采用 `simple` 方法分割后的处理器 1 的网格。

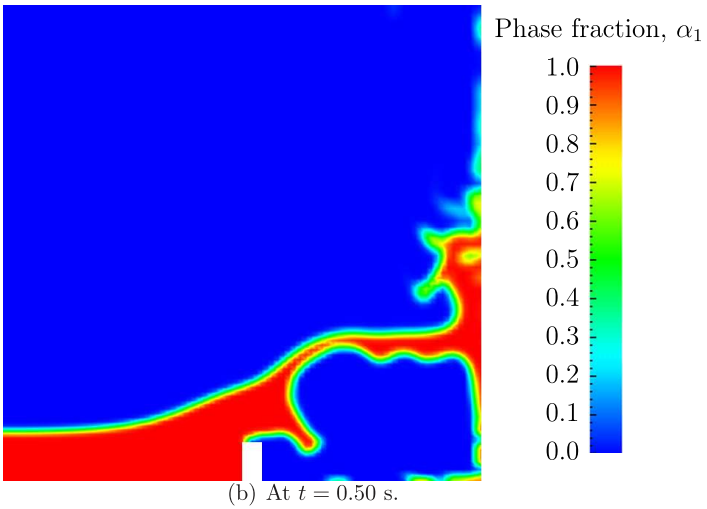
2.4 圆柱绕流

在这个例子当中，我们将利用 `potentialFoam` 来研究圆柱势流（potential flow）。这个例子将会着重介绍 OpenFOAM 如何处理如下问题：

- 非正交网格；
- 对 OpenFOAM 当中的问题生成解析解；



(a) At $t = 0.25$ s.



(b) At $t = 0.50$ s.

图 2-25: 细化网格后的 α 相分数场

2.4.1 问题阐述

我们将这个圆柱绕流问题描述如下：

计算域 计算域是二维的，由一个正方形和一个与其同心的圆组成，如图2-26所示

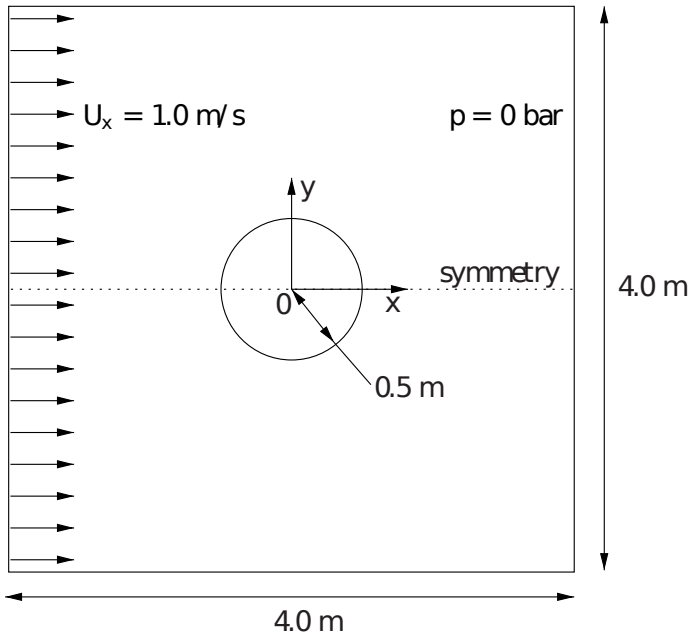


图 2-26: 圆柱绕流几何模型示意图

控制方程

- 不可压缩流体的质量守恒方程：

$$\nabla \cdot \mathbf{U} = 0 \tag{2-16}$$

- 不可压缩缩无旋场的稳态压力方程：

$$\nabla^2 p = 0 \tag{2-17}$$

边界条件

- 入口（左边界）为固定速度入口 U = (1,0,0) m/s；
- 出口（右边界）为固定压力出口 p = 0 pa；
- 下边界为无滑移壁面；
- 上边界为对称面边界；

初始条件 $\mathbf{U} = 0\text{m/}s$ 且 $p = 0\text{pa}$。虽然对于稳态问题来说，并不需要初始条件，但是对于 OpenFOAM 算例来说，无论求解问题是否稳态都必须给定初始条件；

求解器名称 `potentialFoam`，其为求解势流的求解器，假设流场不可压缩、无旋、无黏、稳态、并且忽略重力；

算例名称 `cylinder`，这个算例位于 `$FOAM_TUTORIALS/potentialFoam` 文件夹中；

2.4.2 potentialFoam 求解器的注意事项

potentialFoam 求解器对于 OpenFOAM 正确性的验证有着十分重要的作用，因为势流中的假设使得许多几何模型比较简单的算例都具有解析解。对于这个圆柱绕流这个算例，我们可以用已有的解析解来对照我们的数值解。potentialFoam 还可以对其他问题提供符合质量守恒方程的初始场[48]。当处理类似问题的时候，此种方法可以避免由于初始场不稳定所带来的计算不稳定性。总之，potentialFoam 可以用来从用户提供的不符合质量守恒的初始场生成符合质量守恒的流场。

2.4.3 网格的生成

在《OpenFOAM 用户指南》中，我们已经阐述了如何用 blockMesh 生成网格。在这个算例中，整个计算域由十个 block 构成，如图2-27所示。需要注意的是，在 OpenFOAM 之中，所有网格都是按照三维网格处理的。如果我们想要解决二维问题，我们必须在第三个维度上加上一个网格的厚度（虽然我们并没有这第三个维度上解控制方程）。

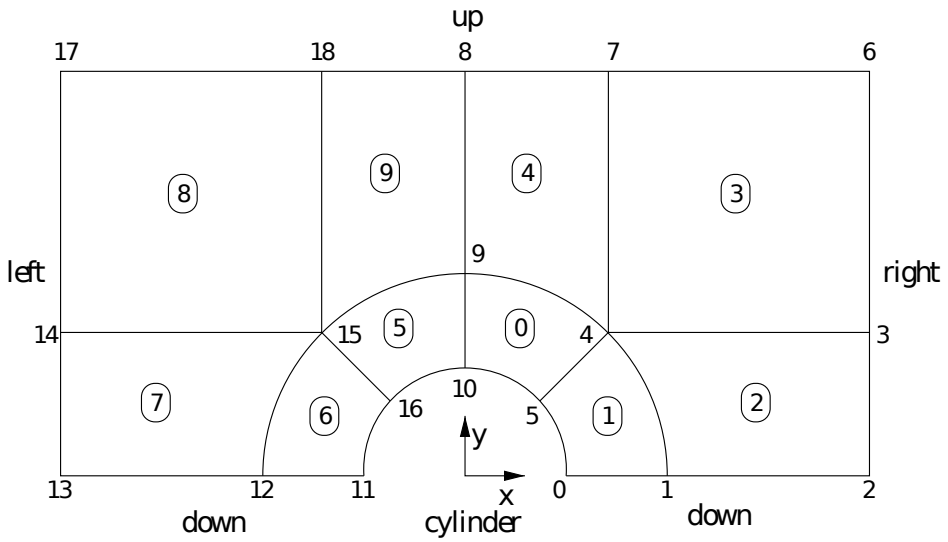


图 2-27: 圆柱绕流中的 block

在图2-27中，我们只显示了几何图形的背面，即 $z=-0.5$ 平面，在这个平面上，只有 0 到 18 号节点，而另外 19 个节点在几何图形的正面，$z=0.5$ 平面。正面 19 个节点的排序和背面相同，网格节点和 block 的信息被储存在 `blockMeshDict` 文件夹当中，如下所示：

```
/*--------------------------------*- C++ -*----------------------------------*\
| = = = = = = = = =                |                                         |
| \ \        /      F ield         | OpenFOAM:     The Open Source CFD Toolbox |
|  \ \      /       O peration     | Version:            2.3.1               |
|   \ \    /        A nd           | Web:          www.OpenFOAM.org            |
|    \ \  /         M anipulation  |                                         |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class               dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices #codeStream
{
```

[48]在算例运行之前首先运行势流求解器可以大大加快收敛且增加求解稳定性

```
    codeInclude
        #{
                #include "pointField.H"
        #};

    code
        #{
        pointField points(19);
        points[0] = point(0.5, 0, -0.5);
        points[1] = point(1, 0, -0.5);
        points[2] = point(2, 0, -0.5);
        points[3] = point(2, 0.707107, -0.5);
        points[4] = point(0.707107, 0.707107, -0.5);
        points[5] = point(0.353553, 0.353553, -0.5);
        points[6] = point(2, 2, -0.5);
        points[7] = point(0.707107, 2, -0.5);
        points[8] = point(0, 2, -0.5);
        points[9] = point(0, 1, -0.5);
        points[10] = point(0, 0.5, -0.5);
        points[11] = point(-0.5, 0, -0.5);
        points[12] = point(-1, 0, -0.5);
        points[13] = point(-2, 0, -0.5);
        points[14] = point(-2, 0.707107, -0.5);
        points[15] = point(-0.707107, 0.707107, -0.5);
        points[16] = point(-0.353553, 0.353553, -0.5);
        points[17] = point(-2, 2, -0.5);
        points[18] = point(-0.707107, 2, -0.5);

        // Duplicate z points
        label sz = points.size();
        points.setSize(2*sz);
        for (label i = 0; i < sz; i++)
        {
            const point& pt = points[i];
            points[i+sz] = point(pt.x(), pt.y(), -pt.z());
        }

        os << points;
        #};
};


blocks
(
    hex (5 4 9 10 24 23 28 29) (10 10 1) simpleGrading (1 1 1)
    hex (0 1 4 5 19 20 23 24) (10 10 1) simpleGrading (1 1 1)
    hex (1 2 3 4 20 21 22 23) (20 10 1) simpleGrading (1 1 1)
    hex (4 3 6 7 23 22 25 26) (20 20 1) simpleGrading (1 1 1)
    hex (9 4 7 8 28 23 26 27) (10 20 1) simpleGrading (1 1 1)
    hex (15 16 10 9 34 35 29 28) (10 10 1) simpleGrading (1 1 1)
    hex (12 11 16 15 31 30 35 34) (10 10 1) simpleGrading (1 1 1)
    hex (13 12 15 14 32 31 34 33) (20 10 1) simpleGrading (1 1 1)
    hex (14 15 18 17 33 34 37 36) (20 20 1) simpleGrading (1 1 1)
    hex (15 9 8 18 34 28 27 37) (10 20 1) simpleGrading (1 1 1)
);

edges
(
    arc 0 5 (0.469846 0.17101 -0.5)
    arc 5 10 (0.17101 0.469846 -0.5)
    arc 1 4 (0.939693 0.34202 -0.5)
    arc 4 9 (0.34202 0.939693 -0.5)
    arc 19 24 (0.469846 0.17101 0.5)
    arc 24 29 (0.17101 0.469846 0.5)
    arc 20 23 (0.939693 0.34202 0.5)
    arc 23 28 (0.34202 0.939693 0.5)
    arc 11 16 (-0.469846 0.17101 -0.5)
    arc 16 10 (-0.17101 0.469846 -0.5)
    arc 12 15 (-0.939693 0.34202 -0.5)
    arc 15 9 (-0.34202 0.939693 -0.5)
    arc 30 35 (-0.469846 0.17101 0.5)
    arc 35 29 (-0.17101 0.469846 0.5)
    arc 31 34 (-0.939693 0.34202 0.5)
    arc 34 28 (-0.34202 0.939693 0.5)
);

boundary
(
    down
    {
        type    symmetryPlane;
        faces
            (
                (0 1 20 19)
                (1 2 21 20)
                (12 11 30 31)
                (13 12 31 32)
            );
    }
    right
    {
        type    patch;
        faces
            (
                (2 3 22 21)
                (3 6 25 22)
            );
    }
    up
    {
```

```
        type    symmetryPlane;
        faces
            (
                (7 8 27 26)
                (6 7 26 25)
                (8 18 37 27)
                (18 17 36 37)
            );
    }
    left
    {
        type    patch;
        faces
            (
                (14 13 32 33)
                (17 14 33 36)
            );
    }
    cylinder
    {
        type    symmetry;
        faces
            (
                (10 5 24 29)
                (5 0 19 24)
                (16 10 29 35)
                (11 16 35 30)
            );
    }
);

mergePatchPairs
(
);

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

2.4.4 边界条件和初始条件

下面我们根据图2-27所示使用 FoamX 或者手动设置边界条件。左边界应该是入口（Inlet），右边界应该是出口（Outlet），下边界和圆柱边界应该设置为 symmetryPlane。当我们选择上边界的时候应当遵循以下规则：所选择的边界条件应当允许我们用理论解与数值解进行比较，因此，我们假设计算域在 y 方向上无穷大。这个假设使得垂直于上边界的速度梯度特别小。因此我们在上边界选择垂直于上边界速度梯度为 0 的边界条件，即 symmetryPlane。这个边界条件使得理论解和数值解之间的比较变得更加合理。

2.4.5 运行算例

在这个算例中，我们不需要设置流体的性质，因为整个流场是不可压缩无旋无黏的。在 system 子文件夹中，controlDict 文件夹记录着算例运行所需要的参数。需要注意的是，因为我们要处理的是稳态问题，所以这个算例只应当运行一个时间步：

```
/*--------------------------------*- C++ -*----------------------------------*\
| = = = = = = = =               |                                            |
| \ \         /  F ield         | OpenFOAM:     The Open Source CFD Toolbox  |
|  \ \       /   O peration     | Version:              2.3.1                |
|   \ \     /    A nd           | Web:          www.OpenFOAM.org             |
|    \ \   /     M anipulation  |                                            |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application             potentialFoam;

startFrom               startTime;

startTime               0;

stopAt                  endTime;

endTime                 1;

```

```
deltaT                  1;

writeControl            timeStep;

writeInterval           1;

purgeWrite              0;

writeFormat             ascii;

writePrecision          6;

writeCompression        off;

timeFormat              general;

timePrecision           6;

runTimeModifiable       true;

functions
{
    difference
    {
        // Load the library containing the 'coded' functionObject
        functionObjectLibs ("libutilityFunctionObjects.so");
        type coded;
        // Name of on-the-fly generated functionObject
        redirectType error;
        code
            #{
            // Lookup U
            Info<< "Looking up field U\n" << endl;
            const volVectorField& U = mesh().lookupObject<volVectorField>("U");

            Info<< "Reading inlet velocity uInfX\n" << endl;

            scalar ULeft = 0.0;
            label leftI = mesh().boundaryMesh().findPatchID("left");
            const fvPatchVectorField& fvp = U.boundaryField()[leftI];
            if (fvp.size())
            {
                ULeft = fvp[0].x();
            }
            reduce(ULeft, maxOp<scalar>());

            dimensionedScalar uInfX
                (
                    "uInfx",
                    dimensionSet(0, 1, -1, 0, 0),
                    ULeft
                );

            Info << "U at inlet = " << uInfX.value() << " m/s" << endl;


            scalar magCylinder = 0.0;
            label cylI = mesh().boundaryMesh().findPatchID("cylinder");
            const fvPatchVectorField& cylFvp = mesh().C().boundaryField()[cylI];
            if (cylFvp.size())
            {
                magCylinder = mag(cylFvp[0]);
            }
            reduce(magCylinder, maxOp<scalar>());

            dimensionedScalar radius
                (
                    "radius",
                    dimensionSet(0, 1, 0, 0, 0),
                    magCylinder
                );

            Info << "Cylinder radius = " << radius.value() << " m" << endl;

            volVectorField UA
                (
                    IOobject
                    (
                        "UA",
                        mesh().time().timeName(),
                        U.mesh(),
                        IOobject::NO_READ,
                        IOobject::AUTO_WRITE
                    ),
                    U
                );

            Info<< "\nEvaluating analytical solution" << endl;

            const volVectorField& centres = UA.mesh().C();
            volScalarField magCentres(mag(centres));
            volScalarField theta(acos((centres & vector(1,0,0))/magCentres));

            volVectorField cs2theta
                (
                    cos(2*theta)*vector(1,0,0)
                    + sin(2*theta)*vector(0,1,0)
                );

            UA = uInfX*(dimensionedVector(vector(1,0,0))
            - pow((radius/magCentres),2)*cs2theta);

```

```
130             // Force writing of UA (since time has not changed)
131             UA.write();
132
133             volScalarField error("error", mag(U-UA)/mag(UA));
134
135             Info<<"Writing relative error in U to " << error.objectPath()
136                 << endl;
137
138             error.write();
139             #};
140     }
141 }
142
143
144 // ************************************************************************* //
```

potentialFoam 通过迭代来求解压力方程，并且可以对压力方程中的拉普拉斯项进行非正交修正（压力方程中的显式项会在正交修正迭代之后进行更新）。对压力方程的非正交修正迭代次数由 controlDict 之中的 nNonOrthogonalCorrectors 来决定。在第一个例子中，我们将 nNonOrthogonalCorrectors 设置为 0，也就是说，在求解过程中没有非正交修正迭代步（压力方程只被求解一次）。与之对应的数值解如图2-28（a）所示（$t=1$，当稳态求解结束之时）。我们期待数值解能够像解析解（如图2-28（c）所示）一样，即穿过整个计算域的流线是光滑的，但是在网格正交性不好的地方我们却能观察到明显的误差（比如在 block 0，1，3 的交界处）。如果我们将 nNonOrthogonalCorrectors 设置为 3，再运行一次算例，其数值解如图2-28（b）所示。这个解的结果与理论解非常相近（流线非常光滑），因此并没有明显的由非正交性所引起的误差。

2.5 稳态后向台阶湍流模拟

在这个算例中，我们将会进行稳态后向台阶湍流模拟。算例的原型来自 Pitz 和 Daily 的实验研究，我们将对算例所得到的数值解与实验值进行比较。这个算例会涉及如下特性：

- 利用 blockMesh 当中的网格非均匀分布功能生成网格；
- 稳态湍流；

2.5.1 问题阐述

稳态后向台阶湍流模拟定义如下：

计算域 计算域是二维的，由一个窄的入口、一个后向台阶、一个在出口处收缩的喷管组成，如图2-29所示：

控制方程

- 不可压缩缩流体的质量守恒方程：

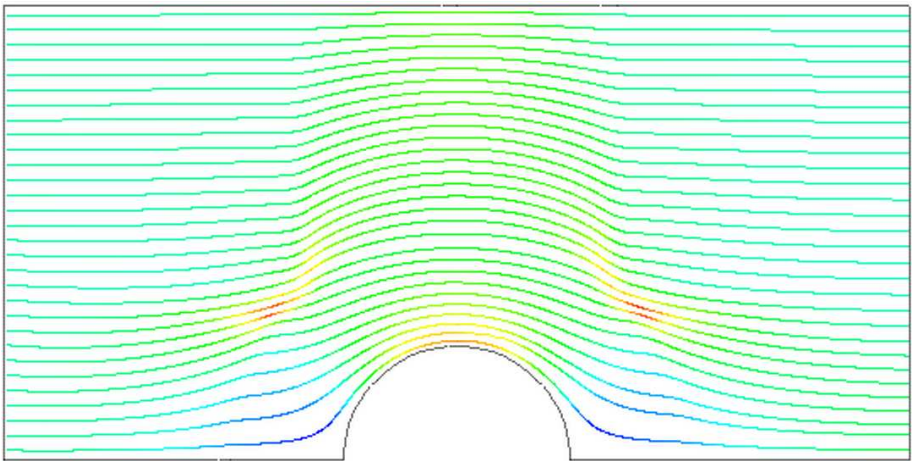
$$\nabla \cdot \mathbf{U} = 0 \tag{2-18}$$

- 稳态动量方程：

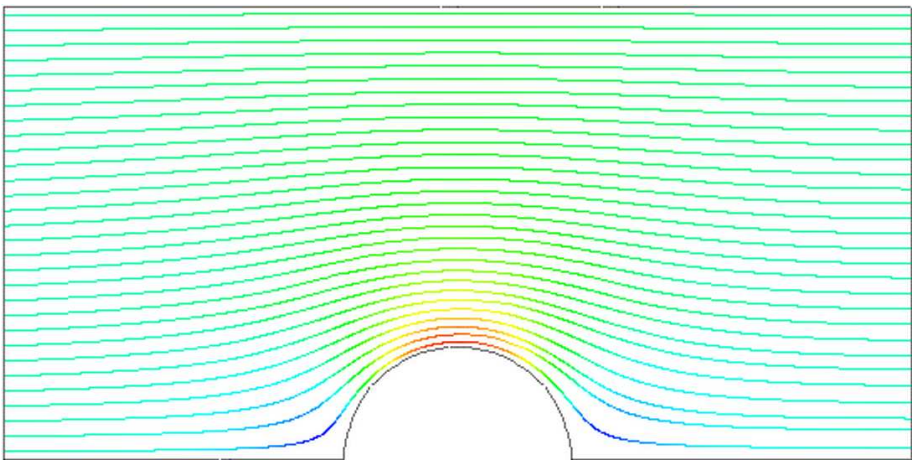
$$\nabla \cdot (\mathbf{UU}) + \nabla \cdot \mathbf{R} = -\nabla p \tag{2-19}$$

这里的 p 是动压，$\mathbf{R} = v_{eff}\nabla \mathbf{U}$ 是带有有效动力粘度系数 eff 的粘性力项。eff 是通过传递模型和湍流模型计算而来；

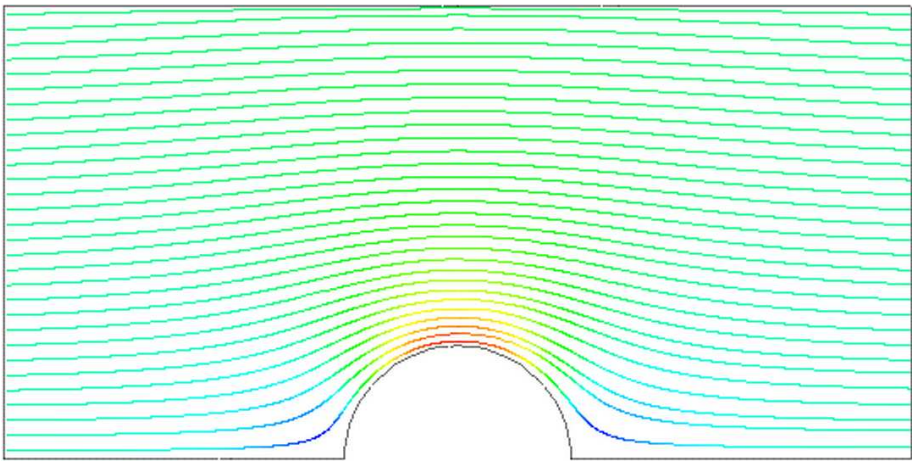
初始条件 $\mathbf{U} = 0\text{m/s}$ 且 $p = 0\text{pa}$。虽然对于稳态问题来说，并不需要初始条件，但是对于 OpenFOAM 算例来说，必须给定初始条件，无论求解问题是否稳态；



(a) With no non-orthogonal correction



(b) With non-orthogonal correction



(c) Analytical solution

图 2-28: 势流中的流线

边界条件

- 进口为固定速度入口，$\mathbf{U} = (10,0,0)\mathrm{m/s}$；

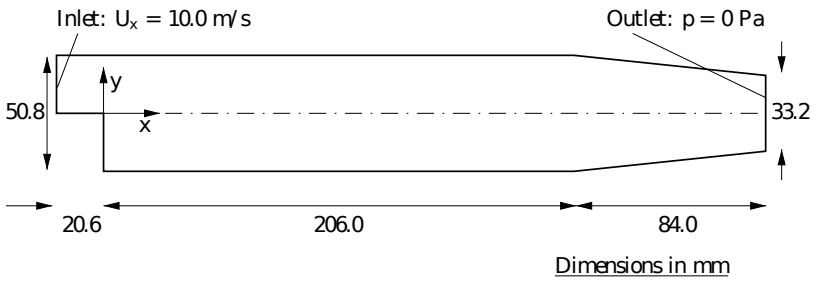


图 2-29: 后向台阶的几何模型

- 出口为固定压力出口 $p = 0$pa；
- 其他边界都为非滑移边界；

物性

- 空气的运动粘度系数 $\nu = \frac{\mu}{\rho} = 18.1 \times 10^{-6}/1.293 = 14.0\mu m^2/s$；

湍流模型

- 标准 k-ε 模型；
- 常数：$C_\mu = 0.09$；$C_1 = 1.44$；$C_2 = 1.92$；$\alpha_k = 1$；$\alpha_\varepsilon = 0.76923$；

求解器名称 simpleFoam，其为稳态不可压缩流体问题的求解器；

算例名称 pitzDaily，位于 `$FOAM_TUTORIALS/simpleFoam` 文件夹中；

我们利用 `simpleFoam` 来求解这个问题。`simpleFoam` 利用 SIMPLE 格式来求解稳态不可压缩流体的问题。这个求解器可以使用 `incompressibleTurbulenceModel` 库中所有的湍流模型以及 `incompressibleTransportModel` 库中的所有牛顿以及非牛顿流体模型。

2.5.2 网格的生成

我么希望这个算例的流场足够的复杂，因此需要利用网格非均匀分布才能得到最优解。一般来说，剪切应力（速度梯度）大的区域网格细化尤其重要。在这种区域的网格需要比其他剪切应力较低的区域更细。在进行求解之前，我们可以提前估计一下在哪些地方容易出现大的剪切应力。在入口处，我们有一个沿着 x 方向的稳态流动，在流体经过台阶时，这个稳态流动和下方静止的流体之间会产生一个剪切应力，并且在计算域的下半部分产生漩涡。所以，在这个问题中，剪切应力比较大的区域应该在计算域的中心线附近和靠近壁面的地方。

计算域被分为 12 个 block，如图2-30所示。

其网格是三维的。图2-30所示的是计算域的背面，$z = -0.5$ 平面。网格节点和 block 的信息储存在 `blockMeshDict` 文件夹中，如下所示：

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM:     The Open Source CFD Toolbox       |
4  |  \\    /   O peration     | Version:              2.3.1                     |
5  |   \\  /    A nd           | Web:          www.OpenFOAM.org                  |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
```

```
        version 2.0;
        format  ascii;
        class   dictionary;
        object  blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001;

vertices
(
        (-20.6 0 -0.5)
        (-20.6 3 -0.5)
        (-20.6 12.7 -0.5)
        (-20.6 25.4 -0.5)
        (0 -25.4 -0.5)
)
```

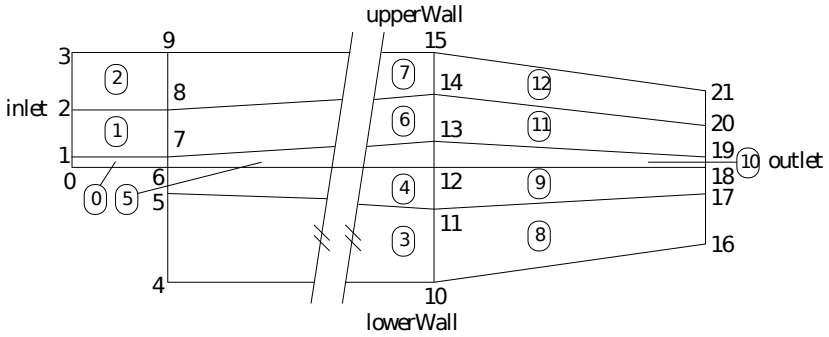


图 2-30: 后向台阶问题中的 block

```
(0 -5 -0.5)
(0 0 -0.5)
(0 3 -0.5)
(0 12.7 -0.5)
(0 25.4 -0.5)
(206 -25.4 -0.5)
(206 -8.5 -0.5)
(206 0 -0.5)
(206 6.5 -0.5)
(206 17 -0.5)
(206 25.4 -0.5)
(290 -16.6 -0.5)
(290 -6.3 -0.5)
(290 0 -0.5)
(290 4.5 -0.5)
(290 11 -0.5)
(290 16.6 -0.5)
(-20.6 0 0.5)
(-20.6 3 0.5)
(-20.6 12.7 0.5)
(-20.6 25.4 0.5)
(0 -25.4 0.5)
(0 -5 0.5)
(0 0 0.5)
(0 3 0.5)
(0 12.7 0.5)
(0 25.4 0.5)
(206 -25.4 0.5)
(206 -8.5 0.5)
(206 0 0.5)
(206 6.5 0.5)
(206 17 0.5)
(206 25.4 0.5)
(290 -16.6 0.5)
(290 -6.3 0.5)
(290 0 0.5)
(290 4.5 0.5)
(290 11 0.5)
(290 16.6 0.5)
);

blocks
(
    hex (0 6 7 1 22 28 29 23) (18 7 1) simpleGrading (0.5 1.8 1)
    hex (1 7 8 2 23 29 30 24) (18 10 1) simpleGrading (0.5 4 1)
    hex (2 8 9 3 24 30 31 25) (18 13 1) simpleGrading (0.5 0.25 1)
    hex (4 10 11 5 26 32 33 27) (180 18 1) simpleGrading (4 1 1)
    hex (5 11 12 6 27 33 34 28) (180 9 1) edgeGrading (4 4 4 4 0.5 1 1 0.5 1 1 1 1)
    hex (6 12 13 7 28 34 35 29) (180 7 1) edgeGrading (4 4 4 4 1.8 1 1 1.8 1 1 1 1)
    hex (7 13 14 8 29 35 36 30) (180 10 1) edgeGrading (4 4 4 4 4 1 1 4 1 1 1 1)
    hex (8 14 15 9 30 36 37 31) (180 13 1) simpleGrading (4 0.25 1)
    hex (10 16 17 11 32 38 39 33) (25 18 1) simpleGrading (2.5 1 1)
    hex (11 17 18 12 33 39 40 34) (25 9 1) simpleGrading (2.5 1 1)
    hex (12 18 19 13 34 40 41 35) (25 7 1) simpleGrading (2.5 1 1)
```

```
    hex (13 19 20 14 35 41 42 36) (25 10 1) simpleGrading (2.5 1 1)
    hex (14 20 21 15 36 42 43 37) (25 13 1) simpleGrading (2.5 0.25 1)
);

edges
(
);

boundary
(
    inlet
    {
        type    patch;
        faces
            (
                (0 22 23 1)
                (1 23 24 2)
                (2 24 25 3)
            );
    }
    outlet
    {
        type    patch;
        faces
            (
                (16 17 39 38)
                (17 18 40 39)
                (18 19 41 40)
                (19 20 42 41)
                (20 21 43 42)
            );
    }
    upperWall
    {
        type    wall;
        faces
            (
                (3 25 31 9)
                (9 31 37 15)
                (15 37 43 21)
            );
    }
    lowerWall
    {
        type    wall;
        faces
            (
                (0 6 28 22)
                (6 5 27 28)
                (5 4 26 27)
                (4 10 32 26)
                (10 16 38 32)
            );
    }
    frontAndBack
    {
        type    empty;
        faces
            (
                (22 28 29 23)
                (23 29 30 24)
                (24 30 31 25)
                (26 32 33 27)
                (27 33 34 28)
                (28 34 35 29)
                (29 35 36 30)
                (30 36 37 31)
                (32 38 39 33)
                (33 39 40 34)
                (34 40 41 35)
                (35 41 42 36)
                (36 42 43 37)
                (0 1 7 6)
                (1 2 8 7)
                (2 3 9 8)
                (4 5 11 10)
                (5 6 12 11)
                (6 7 13 12)
                (7 8 14 13)
                (8 9 15 14)
                (10 11 17 16)
                (11 12 18 17)
                (12 13 19 18)
                (13 14 20 19)
                (14 15 21 20)
            );
    }
);

mergePatchPairs
(
);

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

这个算例的一大特点，是它大量的使用了《OpenFOAM 用户指南》中6.3.1所介绍的 blockMesh 中的网格非均匀功能。用户可以看到，在 block 4、5、6 中，每个边均使用了非均匀分布（每个膨胀

率都对应着 block 当中的一个边）。前四个膨胀率对应与 x1 轴平行的四个边，中间四个膨胀率对应与 x2 轴平行的四个边，后四个膨胀率对应与 x3 轴平行的四个边。在 block 4、5、6 中，膨胀率在 x1 和 x3 方向上都是相同的，但是在 x2 方向（y 方向）却是不同的。如果我们参考《OpenFOAM 用户指南》6.3.1中对 block 的定义，我们可以发现 block4，5，6 中的左边和右边的细化是不一样的。这样做的目的是为了在最关键的地方（台阶的尖角处）生成细网格 0。

网格可以在终端由 bockMesh 生成，也可以在 FoamX 之中生成，查看网格的方法再次不再赘述。

2.5.3 边界条件和初始场

与算例相关的参数文件可以在 `FoamX` 之中浏览或者修改（当然也可以用手动修改）。在这个算例中，我们需要为如下参数设置边界条件和初始条件：速度 **U**，压力 p，湍流动能 k，和耗散率 ε。边界条件的设置可以通过在 `FoamX` 之中设置 patch 的类型来实现: 上边界和下边界被设置为 `wall`，左边界被设置为 `Inlet`，右边界被设置为 `Outlet`。这样的边界条件组合需要我们在入口处设定 **U**、k、ε 的类型为 `fixedValue`。**U** 的具体值已经在问题阐述中给出。k 和 ε 的值应该利用和《OpenFOAM 用户指南》中2.1.8.1节一样的方法计算出来。我们假设入口湍流是各向同性，而且入口处的速度波动为 5% 有：

$$U_x^{'} = U_y^{'} = U_z^{'} = \frac{5}{100}10 = 0.5\text{m/s} \tag{2-20}$$

且

$$k = 1.5(0.5)^2 = 0.375\text{m}^2/\text{s}^2 \tag{2-21}$$

如果我们假定湍流尺度为入口边界长度的十分之一有：

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} = \frac{0.09^{0.75} 0.375^{1.5}}{0.1 \times 25.4 \times 10^{-3}} = 14855\text{m}^2/\text{s}^2 \tag{2-22}$$

在出口处，我们只需要将 p 设置为 0pa。

2.5.4 求解器的控制参数

`fvSchemes` 的设置如下：`timeScheme` 设置为 `SteadyState`；`gradScheme` 和 `laplacianScheme` 设置为默认值 `Gauss`；为了确保有界性，`divScheme` 应该设置为 `UD`。

我们需要特殊注意 `fvTolerances` 的设置。虽然顶层的 `simpleFoam` 代码只包含了压力和速度的方程，但是在湍流模型中，我们还需要求解，和 R 的方程，所以，我们需要为 5 个方程设置残差（tolerance）。除了将压力的 `solverTolerance` 设置为 $10-6$ 且 `solverRelativeTolerance` 设置为 0.01 之外，对于其他变量，我们将这两个参数分别设置为 $10-5$ 和 0.1。由于这个算例是一个稳态问题，所以我们需要设置松弛因子（`relaxationFactor`）。对于压力，为了避免计算的不稳定，松弛因子被设置为 0.3。对于其他参数（U，k，ε，R），松弛因子可以设置为 0.7。

最后，在 `controlDict` 之中，我们将时间步长设置为 1(因为计算的稳态的)，这里的时间步长只用于稳态计算中的迭代次数进行计数。这个算例需要至少一千次迭代才能收敛，所以，我们将 `endTime` 设置为 1000。我们还需要保证存储足够的时间步且不会太多，这样可以设置为 50 步一写，以保证用户的硬盘空间不会在算例运行过程中被数据填满。

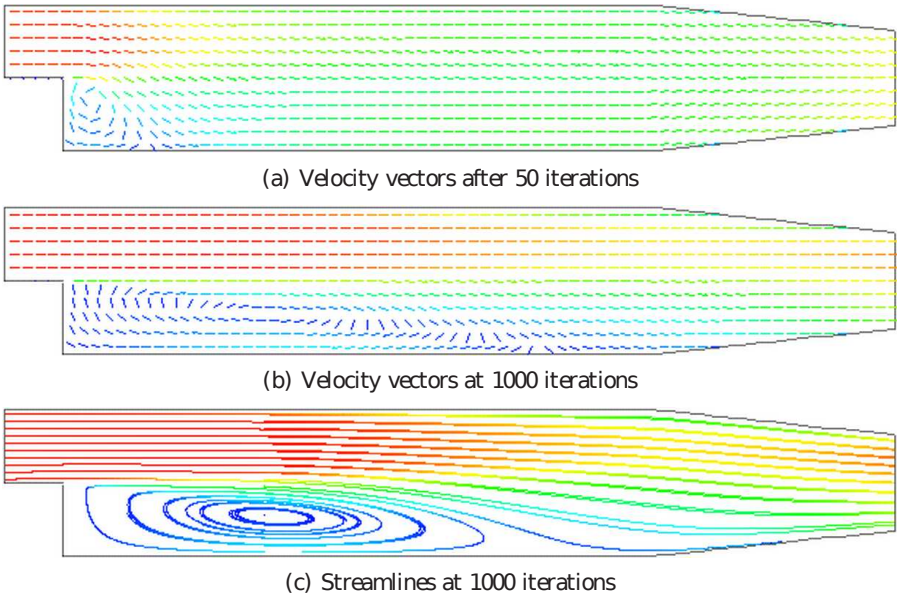


图 2-31: 后面台阶问题中的漩涡发展

2.5.5 运行算例和后处理

接下来我们运行程序并且对运行结果进行后处理。在几步迭代之后，比如 50 步，在台阶的尖角下方会出现一个漩涡，但是漩涡在 x 方向上长度很短，如图2-31（a）所示。在 1000 个时间步之后，漩涡沿着 x 方向从台阶处一直延伸到出口处，此时系统已经达到稳态，并且漩涡已经充分发展，如图2-31（b-c）所示。

2.6 前向台阶的超音速绕流

在这个算例中，我们将会探究前向台阶的超音速绕流。在这个算例中，入口速度马赫数为 3，在入口附近有一个台阶，在台阶附近会产生激波。

这个算例主要涉及 OpenFOAM 的如下特性：

- 超音速问题的求解

2.6.1 问题阐述

计算域 计算域是二维的，并且由一个入口区域和一个前向台阶组成。台阶的高度占入口长度的 20%，如图2-32所示：

控制方程

- 质量守恒方程：

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{2-23}$$

- 理想气体状态方程：

$$p = \rho R T \tag{2-24}$$

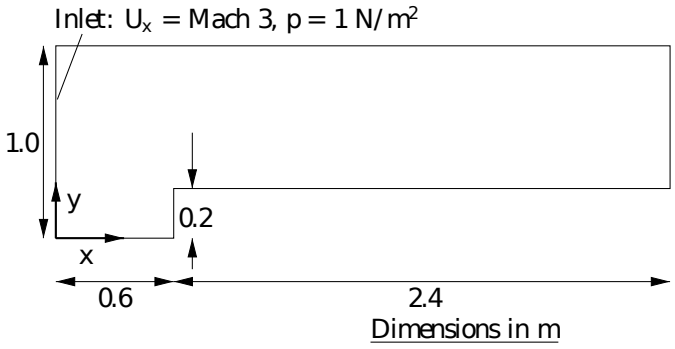


图 2-32: 前向台阶的几何模型

- 牛顿流体的动量守恒方程：

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U}\mathbf{U}) - \nabla \cdot \mu \nabla \mathbf{U} = -\nabla p \tag{2-25}$$

- 能量方程（忽略粘性项），$e = C_v T$，和傅里叶定律，$q = -k\nabla T$：

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho \mathbf{U} e) - \nabla \cdot \frac{k}{c_v} \nabla e = -p \nabla \cdot \mathbf{U} \tag{2-26}$$

初始条件 $U = 1\text{m/s}$，$p = 1\text{pa}$，$T = 1k$；

边界条件

- 入口设置为固定值边界条件 `fixedValue`，$U = 3\text{m/s} = \text{Mach}3$，$p = 1\text{pa}$，$T = 1k$；；
- 出口对速度，压力，温度（U，p，T）设置为法相梯度为零 `zeroGradient`；
- 下边界被设置为无滑移绝热边界；
- 上边界被设置为 `symmetryPlane`；

物性 • 空气的动力粘度系数为 $\mu = 18.1\mu\text{Pa}s$；

热力学性质

- 定压比热容 $C_v = 1.78571$ J/kgK；
- 气体常数 $R = 0.714286$ J/kgK；
- 热导率 $k = 32.3$ μW/mK；

算例名称 forwardStep，其位于 `$FOAM_TUTORIALS/sonicFoam` 文件夹中；

求解器名称 sonicFoam[49]：为可压缩流体的超音速和跨音速问题设计的求解器；

在本算例中，音速被设计为 1m/s($c = \sqrt{\gamma RT} = 1\text{m/s}$)。这样做的结果就是速度的大小直接等于马赫数。音速的大小可以利用如下关系计算。对于理想气体，$C_p - C_v = R$：

$$\gamma = \frac{c_p}{c_v} = \frac{R}{c_v} + 1 \tag{2-27}$$

[49]目前新版本 OpenFOAM 中已经不存在此求解器，可以用 rhoCentralFoam 求解器来替换使用。

2.6.2 网格生成

本算例中所应用的网格比较简单，用大小相同的长方体网格生成即可。这些长方体网格在 x 方向长 0.06m，y 方向长 0.05m。整个计算域可以被分为三个 block，一个在台阶上沿下方，两个在台阶上沿上方（分列于台阶两侧）。关于网格的信息被储存在如下所示的 `blockMeshDict` 文件之中：

```
/*--------------------------------*- C++ -*----------------------------------*\
| = = = = = = = = =              |                                           |
| \ \         /  F ield          | OpenFOAM:     The Open Source CFD Toolbox |
|  \ \       /   O peration      | Version:            2.3.1                 |
|   \ \     /    A nd            | Web:          www.OpenFOAM.org            |
|    \ \   /     M anipulation   |                                           |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    (0 0 -0.05)
    (0.6 0 -0.05)
    (0 0.2 -0.05)
    (0.6 0.2 -0.05)
    (3 0.2 -0.05)
    (0 1 -0.05)
    (0.6 1 -0.05)
    (3 1 -0.05)
    (0 0 0.05)
    (0.6 0 0.05)
    (0 0.2 0.05)
    (0.6 0.2 0.05)
    (3 0.2 0.05)
    (0 1 0.05)
    (0.6 1 0.05)
    (3 1 0.05)
);

blocks
(
    hex (0 1 3 2 8 9 11 10) (25 10 1) simpleGrading (1 1 1)
    hex (2 3 6 5 10 11 14 13) (25 40 1) simpleGrading (1 1 1)
    hex (3 4 7 6 11 12 15 14) (100 40 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    inlet
    {
        type    patch;
        faces
            (
                (0 8 10 2)
                (2 10 13 5)
            );
    }
    outlet
    {
        type    patch;
        faces
            (
                (4 7 15 12)
            );
    }
    bottom
    {
        type    symmetryPlane;
        faces
            (
                (0 1 9 8)
            );
    }
    top
    {
        type    symmetryPlane;
        faces
            (
                (5 13 14 6)
                (6 14 15 7)
            );
    }
    obstacle
    {
        type    patch;
```

```
89          faces
90              (
91                  (1 3 11 9)
92                  (3 4 12 11)
93              );
94      }
95  );
96
97  mergePatchPairs
98  (
99  );
100
101 // ************************************************************************* //
```

2.6.3 运行算例

本算例将会在 5s 之后达到稳态。10 秒钟时的压力分布如图2-33所示。计算结果显示出了压力的明显不连续性，也就是激波。我们可以看到，激波由从台阶下沿的前方开始发展。

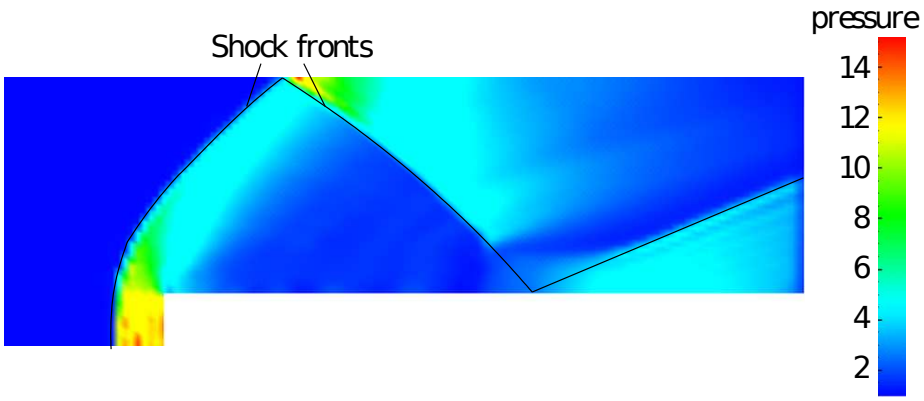


图 2-33: 前向台阶问题中的激波

2.6.4 练习

用户可以检验入口速度的升高对于解的影响。

第 3 章　应用和库

我们反复重申 OpenFOAM 是一个用于创造可执行程序（applications）的 C++ 库。OpenFOAM 在发布时附带了大量可编译的源代码程序。用户也可以随心所欲地创建自己的程序。程序在 OpenFOAM 中分为两个类型：

solvers（**程序**） 依据计算流体力学模型创造的求解器，每个都用来求解一个特定的问题；

utilities（**工具**） 执行简单的前后处理任务，主要负责数据操作和代数几何运算。

OpenFOAM 内部包含了许多编译好了的库，在编译程序和工具时候它们动态地被连接在一起。其中的物理模型库以源代码的方式提供，所以用户可以添加自己的模型在各种库中。这一章我们概览一下这些程序：工具和库。以及如何创建、修改、编译、执行。

3.1　OpenFOAM 编程语言

为了理解 OpenFOAM 库是如何工作的，了解一些 C++（OpenFOAM 使用的语言）的背景知识是非常必要的。这些必要的信息将在这一个章节呈现。在这之前，我们针对计算机语言这个概念来谈一下面向对象编程，以及为什么选择 C++ 作为 OpenFOAM 语言的问题。

3.1.1　普适性编程语言

语言以及数学的成功建立在有效性基础之上，特别是在表达抽象概念的时候。例如，在流体中，我们使用“速度场”来作为文字表达，它没有包含任何有关流体的信息和数据。这个名词将运动的方向、大小以及其它物理量封装起来。在数学中，我们用一个符号来代表速度场：**U**，然后用这个符号进一步表示其它信息。例如“速度场的大小”我们用 |**U**| 表示。数学符号优越于语言符号的根本原因在于它的有效性，数学符号可以简洁的表示复杂的概念。

我们想解决的连续介质问题无法使用计算机可识别的概念（例如 bits，bytes，integers）来表示。因此，连续介质问题首先需要使用文字语言来呈现，然后以时间和空间三个维度的偏微分方程来表示。偏微分方程会包含如下信息：标量、矢量、张量、张量运算、量纲分析等。这些方程的求解涉及到离散、矩阵运算、求解器以及求解算法。

3.1.2　面向对象和 C++

面向对象编程语言，比如 C++，提供了一个有用的工具：类（class），来声明自己的类型和操作，这和工程以及科学中的数学语言是一样的。我们之前用 **U** 来表示的速度场，在编程中也可以用字符 U 来表示，速度场的大小 |**U**| 我们在编程中用 mag(U) 来表示。在面向对象编程中，速度场以 vectorField 这个类来表示。速度场 U 可以说是 vectorField 类的一个对象，这就是面向对象编程。

在编程中使用对象来表示物理对象和抽象实体带来的便捷性不容低估。通过创建这种可以使用代码中所有结构的类分层，能够使得代码管理更容易。新的类可以从其它的类上继承。例如 vectorField 可以从 vector 类以及 Field 类上继承过来。C++ 也提供了模板类机制，例如

Field<Type> 可以表示任何的 <Type> 场，例如 scalar，vector，tensor 等。模板类的特性保留在任何从模板类创造的类型上。模板和继承大大简化了重复代码并且使得整体的代码结构更加清晰。

3.1.3 方程呈现

OpenFOAM 代码设计的中心主题就是使用 OpenFOAM 的类来编写求解器，这些类应该能够很好的表示某个特定的待求解的偏微分方程。例如这个方程：

$$\frac{\partial \rho \mathbf{U}}{\partial t} \nabla \cdot (\mathbf{U}\mathbf{U}) - \nabla(\cdot \mu \nabla \mathbf{U}) = -\nabla p \tag{3-1}$$

它可以使用如下代码来呈现：

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

如果想让 OpenFOAM 使用自己的类表示这个偏微分方程，并且满足一些其它的要求，就要 OpenFOAM 语言具有面向对象的特性，例如继承、模板类、虚函数以及操作符重载。某些计算机语言声称可以实现这些特性（例如 FORTRAN-90），但实际上这些特性在这些语言中是非常有限的。但是 C++ 完全拥有这些特性，同时 C++ 还具有由于其规范标准进而被广泛使用的优势，从而可以使用可靠的编译器来编写高效的可行程序。这就是 OpenFOAM 为什么采用 C++ 的原因。

3.1.4 求解器代码

求解器代码的编写是有步骤可循，因为本身求解器代码就是数学算法和方程组的体现，算法和方程组求解本身，也是有步骤化的。编写一个求解器，用户并不需要纯熟的面向对象的 C++ 编程技能，但是，应该知道面向对象的类型主要规则，以及了解基本的 C++ 句法。相对于编程语言，理解这个方程组、模型和求解步骤更加重要。

用户并不需要每时每刻都研究 OpenFOAM 的代码。面向对象编程的精髓就在于用户不需要这么做。仅仅知道这个类的存在以及它的功能已经足够。每个类及其功能的描述在 doxygen 中都有描述。请查阅本链接

3.2 编译程序和库

编译是程序开发必不可少的过程，由于每一段代码都需要跟某些独立的 OpenFOAM 库来进行对接，因此需要小心对待。在 Unix/Linux 系统中，这些通常需要使用 UNIXmake 程序来传递给编译器进行编译。然而 OpenFOAM 提供了 wmake 程序脚本，它建立在 make 之上，但是 wamke 更加多样性并且简单好用；实际上，不仅仅在 OpenFOAM 的代码上，wmake 也可以在任何代码上使用。为了理解编译过程，我们首先需要讲解一下 C++ 及其文件结构，参见图3-1。类通过类构造函数和类成员函数等一系列代码来构成，这些代码存在于某些文件中。类的文件后缀名为.C。例如 nc 类的文件名为 nc.C。这个文件可以单独编译，编译后为一个共享的库文件可供其它程序调用，其后缀名为.so，比如 nc.so。当编译一个使用这个 nc 类的新程序的时候，比如 newApp.C 的时候，nc.C 不需要重新编译。这即为动态编译。

3.2.1 头文件：*.H*

作为预防错误的发生，编译器必须知道它编译的类以及执行的类函数是一个真实的存在。因此每个类都有一个类声明，它们被放置在后缀为.H头文件中。例如 nc.H。这个文件里面包含了类的名称以及类函数，在任何使用该类的代码中都需要包含这个文件。以.C文件为后缀的程序如果需要这些类必须在文件开始提前声明这些以.H为后缀的类声明。这样，通过递归下降的方式，搜索类层次结构，可以产生一个完整的、以顶层.C文件为最终依赖的头文件列表，这些.h文件称为依赖项。对于一个依赖项列表，编译器可以检查自上次编译后源文件是否有更新，以选择性地只编译那些需要重新编译的文件。

代码中，我们通过 include 语句来包含头文件，例如：

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

会让编译器从当前的文件中暂停读取，去读取指定的文件。任何独立的代码块都可以放进头文件中，并在主函数中的任意位置被包含以提高代码的可读性。例如，在大多数的 OpenFOAM 程序中，创建场和读取场的代码经常被放在 createFields.H 中，这个头文件在程序开始就被调用。在这种方式下，头文件并不是仅仅是类声明了。在 OpenFOAM 中，wmake 可以用于执行以下所列功能中

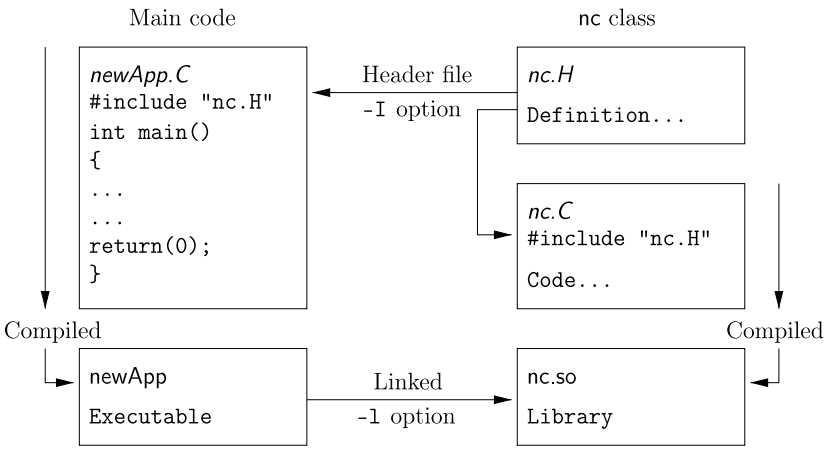


图 3-1: 头文件、源代码、编译和连接

的维护依赖文件列表的任务，并可用于编译源程序：

- 依赖文件列表的自动生成和变更。依赖文件列表就是那些包含在程序中的文件名的列表，程序依靠这些文件的组合来被编译；
- 通过合适的目录结构，多平台编译和链接；
- 多语言编辑和链接，例如 C，C++，JAVA；
- 多种编译和链接选项，例如调试、最优化、并行以及性能分析；
- 支持源代码分析程序，例如 lex、yacc、IDL 以及 MOC；
- 简单的源文件列表语法规则；
- 源文件列表自动生成；
- 静态库或共享库的简单处理；

- 新平台的可扩展性和兼容性；
- 便携性，可以在有 make、sh、ksh 或者 csh、lex、cc 的系统上应用。

3.2.2 使用 wmake 进行编译

OpenFOAM 程序以一种规范的格式来架构，每个程序的源代码放置在以这个程序命名的文件夹中。最顶层的源文件以.C 命名。例如，一个叫做 newApp 的源代码放置在 newApp 的文件夹，顶层文件即为 newApp.C，参见图3-2：这个文件夹必须包含一个叫做 Make 的文件夹，其中有

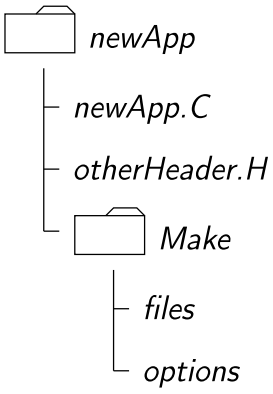


图 3-2: 程序的文件架构

options 以及 files 文件，我们在后面来讨论它们的作用。

3.2.2.1 包含文件头

编译器按照以下顺序在某个文件夹下寻找被包含的头文件，在 wmake 中，它们以-I 标识符来指定：

1. $WM_PROJECT_DIR/src/OpenFOAM/lnInclude 文件夹；
2. 本地 InInclude 文件夹，例如 newApp/InInclude；
3. 本地文件夹，例如 newApp；
4. $WM_PROJECT_DIR/wmake/rules/$WM_ARCH 文件中设定的依赖文件路径，例如/usr/X11/include 目录或者 $(MPICH_ARCH_PATH)/include 目录;
5. 其它在 Make/options 文件中通过-I 指定的其它文件夹。

Make/options 使用按照下面的语句规则来包含文件路径：

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

首先需要注意的是，每个文件夹路径的前面都有一个-I 的标签，在 EXE_INC 之后以及每个文件路径后需要使用

，最后的文件路径后没有
。

3.2.2.2 链接库

编译器在以下的路径中链接库文件，在 wmake 中，它们通过-L 标识符来指定：

1. $FOAM_LIBBIN 目录；
2. $WM_DIR/rules/$WM_ARCH 文件中设定的依赖文件路径，例如./usr/X11/lib 以及 $(MPICH_ARCH_PATH)/lib；
3. Make/options 中指定的其它目录。

链接的库文件必须通过-l 标识符来指定，并且去掉 lib 前缀以及.so 后缀。例如：libnew.so，应该使用-lnew。默认情况下，wmake 调用下面的库文件：

1. $FOAM_LIBBIN 文件夹下面的 libOpenFOAM.so 库；
2. $WM_DIR/rules/$WM_ARCH 文件中设定的依赖库路径，例如./usr/X11/lib 以及 $(MPICH_ARCH_PATH)/lib；
3. Make/options 中指定的其它目录。

```
LIB_LIBS = \\
-L<libraryPath1> \\
-l<library1> \\
-l<library2> \\
... \\
-l<libraryN>
```

值得重申的是，库文件路径使用-L 标识符指定，库名称使用-l 标识符指定。

3.2.2.3 编译源文件

编译器编译的时候需要一个文件列表，除了主程序.C 文件外，需要包含其它没有作为库动态链接的文件并且打算被编译的文件，这个文件列表是必须存在。例如，用户想创建一个新的类或者在一个存在的类上添加一些新的函数。在 Make/files 中必须指定完整的以.C 为后缀的文件列表。许多程序的文件列表只有主程序的文件名。例如，我们之前的例子的文件列表中只有 newApp.C。

Make/files 文件包含了需要进行编译的程序全路径以及名字（用 EXE = 来指定）。标准做法是使用求解器的名字，在例子中，我们使用了 newApp。OpenFOAM 提供了两种可选的路径，一个是标准的路径，编译好的求解器存储在 $FOAM_APPBIN 中；另一个是用户自己的路径存储在 $FOAM_USER_APPBIN 中。

如果用户打算开发他们自己的新程序，我们建议他们在 $WM_PROJECT_USER_DIR 中创建一个子目录，并把他们自己的代码放置于其中。标准程序中，每一个 OpenFOAM 求解器代码都存储在相应的目录。用户自定义的求解器和标准求解器的唯一区别在于 Make/files 文件，它指定用户自定义程序写入 $FOAM_USER_APPBIN 中。Make/files 文件应该是这样的：

```
newApp.C
EXE = $(FOAM_USER_APPBIN)/newApp
```

3.2.2.4 运行 wmake

wmake 可以这样来运行：

```
wmake <optionalDirectory>
```

<optionalDirectory> 是被编译程序的文件路径。一般来说，程序在自己的路径下进行编译，这样一来，<optionalDirectory> 就可以省略。

3.2.2.5 wmake 环境变量设置

下面明确的列举了 wmake 所需设置的环境变量。

$WM_PROJECT_INST_DIR 安装目录路径，例如 $HOME/OpenFOAM；

$WM_PROJECT 编译工程名；

$WM_PROJECT_VERSION 工程版本号，例如 7；

$WM_PROJECT_DIR 可执行文件目录的全路径，例如 $HOME/OpenFOAM/OpenFOAM-9；

$WM_PROJECT_USER_DIR 用户可执行文件目录的全路径，例如 $HOME/OpenFOAM/$USER-9；

$WM_THIRD_PARTY_DIR 用户可执行文件目录的全路径，例如 $HOME/OpenFOAM/ThirdParty-9；

$WM_ARCH 主机架构 linux，linux64，linuxIa64，linuxARM7，linuxPPC64，linuxPPC64le；

$WM_ARCH_OPTION 机器位数 32 位或者 64 位；

$WM_COMPILER 所使用的编译器 Gcc，ICC，Clang；

$WM_COMPILER_OPTION 编译模式 Debug 为调试模式，Opt 为最优模式（默认）；

$WM_COMPILER_TYPE 编译器选择，如 system，ThirdParty；

$WM_DIR wmake 全路径；

$WM_LABEL_SIZE 32 位或者 64 位；

$WM_LABEL_OPTION Int32 或者 Int64 编译；

$WM_LABEL_LANGUAGE 编译器语言；

$WM_MPLIB 并行库 SYSTEMOPENMPI，如 openMPI，SYSTEMMPI，MPICH，HPMPI，MPI 等；

$WM_PRECISION_OPTION 编译二进制文件的精度，SP 为单精度，DP 为双精度。

3.2.3 移除依赖包文件：wclean

在编译的过程中，wmake 会创建一个依赖包文件，扩展名为.deb（在我们的例子中为 newApp.dep）。并在 Make/$WM_OPTIONS 中产生一系列文件。如果用户想要删除这些文件，比如在一些代码被改动的情况下，用户可以运行 wclean 来删除：

```
wclean <optionalDirectory>
```

跟 wmake 相同的是，<optionalDirectory> 是被编译程序的文件路径。一般来说，程序在自己的路径下来运行 wclean，这样，<optionalDirectory> 就可以省略。

3.2.4 编译库

和编译求解器不同，当编译库的时候，Make 文件夹内需要做一些更改：

- files 文件中，EXE = 需要替换为 LIB =，且 $FOAM_APPBIN 需要更改为 $FOAM_LIBBIN（也可以更改为 $FOAM_USER_LIBBIN）；
- options 文件夹中，EXE_LIBS 需要替换为 LIB_LIBS，用来指定编译库的时候需要链接的库。

这样，当执行 wmake 的时候，会自动创建一个 lnInclude 的文件夹包含必要的文件。若用户执行 wclean，这个文件也会被删除[1]。

3.2.5 编译实例：pisoFoam 求解器

pisoFoam 的源代码在 $FOAM_APP/solvers/incompressible/pisoFoam 的文件目录下，最顶层的源代码文件为 pisoFoam.C。如下所示：

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2021 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 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, turbulent flow, using the PISO
    algorithm.

    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 "kinematicMomentumTransportModel.H"
#include "pisoControl.H"
#include "pressureReference.H"
#include "fvModels.H"
#include "fvConstraints.H"

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

int main(int argc, char *argv[])
{
    #include "postProcess.H"
```

[1]老版本的 OpenFOAM 需要执行 wmakelibso 来编译库

```
50
51      #include "setRootCaseLists.H"
52      #include "createTime.H"
53      #include "createMesh.H"
54      #include "createControl.H"
55      #include "createFields.H"
56      #include "initContinuityErrs.H"
57
58      turbulence->validate();
59
60      // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
61
62      Info<< "\nStarting time loop\n" << endl;
63
64      while (runTime.loop())
65      {
66          Info<< "Time = " << runTime.timeName() << nl << endl;
67
68          #include "CourantNo.H"
69
70          // Pressure-velocity PISO corrector
71          {
72              fvModels.correct();
73
74              #include "UEqn.H"
75
76              // --- PISO loop
77              while (piso.correct())
78              {
79                  #include "pEqn.H"
80              }
81          }
82
83          laminarTransport.correct();
84          turbulence->correct();
85
86          runTime.write();
87
88          Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
89              << "  ClockTime = " << runTime.elapsedClockTime() << " s"
90              << nl << endl;
91      }
92
93      Info<< "End\n" << endl;
94
95      return 0;
96  }
97
98
99  // ************************************************************************* //
```

代码最开始是对求解器的一段简单描述。//用于单行注释，多行的注释位于/*…*/之间[2]。在这之后，代码是 include 语句，例如 include ''fvCFD.H''，这时，编译器会暂停读取 pisoFoam.C 文件，转而去读取 fvCFD.H 文件。pisoFoam 会使用 incompressibleRASModels,incompressibleLESModels 和 incompressibleTransportModels 库。因此它们需要一些必要的头文件，它们通过 EXE_INC = -I …来指定，链接的库文件通过 LIB_LIBS = -l …来指定。Make/options 里面信息如下：

```
1   EXE_INC = \\
2       -I$(LIB_SRC)/MomentumTransportModels/momentumTransportModels/lnInclude \\
3       -I$(LIB_SRC)/MomentumTransportModels/incompressible/lnInclude \\
4       -I$(LIB_SRC)/transportModels/lnInclude \\
5       -I$(LIB_SRC)/finiteVolume/lnInclude \\
6       -I$(LIB_SRC)/meshTools/lnInclude \\
7       -I$(LIB_SRC)/sampling/lnInclude
8
9   EXE_LIBS = \\
10
11      -lmomentumTransportModels \\
12      -lincompressibleMomentumTransportModels \\
13      -ltransportModels \\
14      -lfiniteVolume \\
15      -lmeshTools \\
16      -lfvModels \\
17      -lfvConstraints \\
18      -lsampling
```

pisoFoam 仅仅包含 pisoFoam.C 源文件，可执行程序和其它程序一样写入到 $FOAM_APPBIN 文件目录中。Make/Files 包含以下代码：

```
1   pisoFoam.C
2
3   EXE = $(FOAM_APPBIN)/pisoFoam
```

[2]起解释说明的作用，编译的时候跳过

参考第3.2.2.3节的建议，用户可以在自己的文件夹 $FOAM_USER_DIR 下创建自己的 pisoFoam 求解器，可以这样操作来执行：

- 把 pisoFoam 的源代码拷贝到本地文件夹，例如 $FOAM_RUN：

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

- 进入 Make/files 文件中编辑如下：

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

- 执行 wmake：

```
wmake
```

来进行编译。编译过程会产生下列信息：

```
Making dependency list for source file pisoFoam.C
g++ -std=c++0x -m32...
...
-o ... platforms/linuxGccDPInt64Opt/bin/pisoFoam
```

用户可以尝试再次编译这个程序，它会显示下面的信息，表明可执行程序已经是最新版本且没有必要重新编译了：

```
make: ../bin/pisoFoam´ is up to date.
```

用户可以通过：

```
wclean
```

移除依赖文件，从新开始使用 wmake 来进行编译。

3.2.6 调试与优化

OpenFOAM 中存在一个“运行时消息输出机制”，它们有助于对编译 OpenFOAM 算例或者运行算例时出现的问题进行调试。调试开关在 $WM_PROJECT_DIR/etc/controlDict 文件中设置；如果想要改变默认设置，最好把它备份到 $HOME 文件下，例如备份到这个位置：$HOME/.OpenFOAM/9/controlDict（参考4.3节查看更多信息）。默认的调试开关非常多，我们可以通过 foamDebugSwitches 来查看。大多数开关和类以及可执行程序相对应，可以通过在 controlDict 中设置为 1 来打开；例如，OpenFOAM 可以通过把 controlDict 中的 dimensionSet 设置为 1 来检查量纲的一致性。我们列举了一下最高等级的调试开关选项。

调试开关汇总如下：

level OpenFOAM 总体调试等级：共 3 个，即为 0，1，2；

lduMatrix 运行时求解器收敛调试等级：共 3 个，即为 0，1，2。

另外，有些调试开关和某个操作和最优化问题相对应。列出如下所示。其中比较重要的是 fileModificationSkew 选项。OpenFOAM 通过对数据的写入时间确定数据是否被更改。当在计算网络上运行算例的时候，不同计算机的时钟可能不同，某些数据可能会被认为是修改的，这可能会引起混乱，例如这面这种情况：当 OpenFOAM 把所有这些数据当做刚刚修改的新数据并进行读取。fileModificationSkew 关键词是文件更新系统的最大等待时间，其以秒为单位，它用来判断文件是否是刚刚修改的。

优化开关汇总如下：

fileModificationSkew OpenFOAM 运行的计算网络不同计算机的最大延迟，以秒为单位，其值应该高于这个最大延迟；

fileModificationChecking 通过读取 timeStamp 或者使用 inotify 来判断是否在运行时有文件被修改；

commsType 并行通信类型 nonBlocking，scheduled，blocking；

floatTransfer 如果为 1，则在通信之前转换为单精度，默认为 0[3]

nProcsSimpleSum simple 并行分解法的最高值，超于此数值使用 hierarchical 进行分解，默认为 16。

3.2.7 链接自定义库

当用户打算创建一个新的库的时候，比如名称为 new 的库，它希望这个库可以被一系列求解器所调用。例如，用户可能打算创建一个新的边界条件，并编译进入 new 中，并且被求解器、前处理后处理程序以及网格工具等识别。在正常的情况下，用户需要把 new 连接到所有的程序并重新编译。

实际上 OpenFOAM 提供了一种动态链接机制以实现链接一个或多个附加库。我们只需要简单的在 controlDict 中添加一些关键词即可，并在其中键入新的库名称。例如，如果用户在运行的时候打算使用 new1 和 new2 库，我们只需要把下列信息添加到 controlDict 中：

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

3.3 运行程序

对某个算例文件进行读取和写入的程序都可以在终端通过命令行来运行。每个算例的数据文件都存储在这个算例之下，参考4.1节。在这里我们用 <caseDir> 来表示算例的全路径。对于任何可执行程序，都可以在可执行程序后加上-help 来查找命令行的附加命令。例如键入：

```
blockMesh -help
```

会输出以下信息：

```
Usage: blockMesh [OPTIONS]
options:
-blockTopology     write block edges and centres as .obj files
-case <dir>        specify alternate case directory, default is the
```

[3]在通信带宽成为计算速度的瓶颈的时候，可以尝试

```
    cwd
    -dict <file>        specify alternative dictionary for the blockMesh
    description
    -noFunctionObjects do not execute functionObjects
    -region <name>      specify alternative mesh region
    -srcDoc             display source code in browser
    -doc                display application documentation in browser
    -help               print the usage
```

方括号中的参数为可选参数。在某个算例目录下执行某个可执行程序，那么这个程序就会对这个算例目录下的文件进行操作。相反的，如果我们键入-case<caseDir> 附加命令，这个程序就会对系统内任意目录下的数据文件进行操作。

正如任意的 Unix/Linux 可执行程序一样，程序可以以后台的方式来运行。比如，在用户执行其它命令的时候，这个程序不间断的运行下去。如果用户想以后台的方式来执行程序，并把输出流打印到 log 中，他们可以这样键入：

```
blockMesh > log &
```

3.4 并行计算

这一章我们讨论如何使用多个节点来并行计算。OpenFOAM 使用的并行计算方法为计算域分解法。在这个方法中，几何和附属场被拆分为单独的块，每个块用单独的 cpu 来进行计算。并行计算主要涉及到网格和场的分解、并行运行程序、分解场的后处理。我们会在下面的章节详细讨论。在并行计算中我们需要借助第三方 MPI 工具，例如 openMPI。

3.4.1 网格分解与初始场数据

decomposePar 程序用来分解网格和场。想要以最小的资源分解计算域，通过 decomposeParDict（位于算例的 system 文件夹下）字典文件的参数，几何和场文件就能被分解。在 interFoam/damBreak 算例中我们已经有了一个 decomposeParDict 文件，我们可以拷贝它来使用。字典信息如下：

```
18  numberOfSubdomains 4;
19
20  method             simple;
21
22  simpleCoeffs
23  {
24      n               ( 2 2 1 );
25      delta           0.001;
26  }
27
28  hierarchicalCoeffs
29  {
30      n               ( 1 1 1 );
31      delta           0.001;
32      order           xyz;
33  }
34
35  manualCoeffs
36  {
37      dataFile        "";
38  }
39
40  distributed         no;
41
42  roots               ( );
```

用户有 4 个分解方法，它们通过 method 关键词来指定[4]：

simple 简单的几何分解。计算域依据方向被切分，例如 x 方向两块，y 方向一块；

[4]请参考本链接

`hierarchical` 顺序几何分解法和简单分解法差不多，只不过用户指定首先切分哪个方向，例如首先切分 x 方向，然后切分 y 方向；

`scotch`：`scotch` 分解方法不需要用户输入指定参数，并且力求使分割后的每一块的网格边界面最小化[5]。针对不同的计算机处理器，用户可以针对其计算机资源而为每个计算机分配不同的权重，这可以通过 `processorweights` 关键词来指定。另外一个可选关键词是 `strategy`，它可以对 `scotch` 分解法做出一些限定。可以参考:`$FOAM_SRC/parallel/decompose/scotchDecomp/scotchDecomp.C`；

`manual` 手动分解法[6]，用户可以直接把某一片网格区域指定给处理器。

对于每个 `method`[7]都需要指定一些参数，可以在 `decomposeParDict` 字典中的 `<method>-Coeffs` 来指定。表3-1指明了 `decomposeParDict` 所需要的全部关键信息。`decomposePar` 程序可以这样来执行：

```
decomposePar
```

3.4.2 并行文件输入和输出

`decomposePar` 会在算例文件的目录下为每个核产生一个子文件夹。它们的名字为 `processorN`, 此处 `N` 为 0，1…. 其代表着求解器的编号，内部包含了时间步文件、分解场信息以及 `constant/ployMesh` 文件夹（分解后的网格）。

这样，一个分解过的 OpenFOAM 算例就可以并行运行了。并行计算会生成大量的文件。在大型计算模拟中，用户可能会遇到文件打开的瓶颈，其和具体的操作系统有关。

在最新的 OpenFOAM 中引入了 `collated` 文件格式，在这个格式中，分解的场以及网格被存储在主处理器可以读取和写入的单一文件。这些文件位于一个文件夹下，其名称为 `processors`。

必备信息		
numberOfSubdomains	分解域的总数	N
method	分解方法	simple hierarchical scotch metis manual

simpleCoeffs 字典		
n	x, y, z 方向上的分解数量	(n_x, n_y, n_z)
delta	偏斜因子	0.001

hierarchicalCoeffs 字典		
n	x, y, z 方向上的分解数量	(n_x, n_y, n_z)
delta	偏斜因子	0.001
order	分解顺序	xyz/yzx/xzy

scotchCoeffs 字典

[5]请参考本链接，此方法分解后的网格块是不规则形状。
[6]手动分解法的网格可以通过 setFields 来定义
[7]另外一个主要的分解法是 metis，分解后的网格和 scotch 一样不具有规则性，此方法需要下载 metis 库来执行。其他方法略去。

processorWeights（可选）	每个处理器分配的权重，<wt1> 就是第一个处理器分配的权重，值可以随意选取	(<wt1>…<wt2>)
strategy（可选）	分解策略，默认为 b	0.001

manualCoeffs 字典		
dataFile	给各个处理器分配任务的字典文件名称	”<filename>”[8]

数据分布字典（可选）	查阅3.4.3节	
distributed	yes/no	yes/no
root	算例目录路径，例如 <rt1> 就是节点 1 的路径	(<rt1>…<rtN>)

表 3-1: decompositionDict 字典文件

在模拟运行的同时，文件可以被单独的线程写入（详见下述）。因此在使用 `collated` 的时候，不需要调用 `NFS`（网络文件系统）。并且 OpenFOAM 增加了 `materUncollated` 选项支持原本的 `uncollated` 格式，用于设置相关选项的文件位于 `etc/bashrc` 文件下的 `OptimisationSwitches` 关键词中。

```
OptimisationSwitches
{
    ...
    //- Parallel IO file handler
    // uncollated (default), collated or masterUncollated
    fileHandler uncollated;

    //- collated: thread buffer size for queued file writes.
    // If set to 0 or not sufficient for the file size threading is not used.
    // Default: 2e9
    maxThreadFileBufferSize 2e9;

    //- masterUncollated: non-blocking buffer size.
    // If the file exceeds this buffer size scheduled transfer is used.
    // Default: 2e9
    maxMasterFileBufferSize 2e9;
}
```

3.4.2.1 选择 fileHandler

`fileHandler` 可以这样指定：

- 重载 `OptimisationSwtitches fileHandler ...` 设置的默认值；
- 通过在运行时向求解器添加参数：`-fileHandler`；
- 设置 `$FOAM_FILEHANDLER` 环境变量。

3.4.2.2 变更现存文件

`foamFormatConvert` 命令可以转换 `collated` 以及 `uncollated` 文件格式：

[8] 本链接

```
mpirun -np 2 foamFormatConvert -parallel -fileHandler uncollated
```

在 $FOAM_TUTORIALS/IO/fileHandling 文件夹下面有一个算理可用来理解相关操作。

3.4.2.3 多线程支持

Collated 文件格式在多线程下运行更快，尤其是那些大型的算例。但这需要 MPI 的支持，否则的话，模拟将会被“停滞”或者发散。对于 openMPI-2 之前，默认的多线程支持并没有启动。但是在 openMPI-2 之后，默认支持多线程。用户可以运行下面的命令查看 openMPI 多线程是否开启：

```
ompi_info -c | grep -oE "MPI_THREAD_MULTIPLE[^,]*"
```

在使用 collated 文件格式的时候，数据内存挂载在进程中。maxThreadFileBufferSize 指定最大内存大小。如果数据超过这个大小，写入则不会调用超线程。

注意：如果 MPI 没有启用超线程，这个关键词比如在 etc/controlDict 中设置为 0：

```
maxThreadFileBufferSize     0;
```

如果想要使用 masterUncollated 关键词，那么同步非阻塞 MPI 需要主节点具有足够大的内存缓冲层。maxMasterFileBufferSize 设置最大缓冲的大小。如果数据超过这个限制，系统调用排布通讯。

3.4.3 运行并行算例

分解后的 OpenFOAM 算例可以通过 openMPI 来运行。在多核计算机上，使用 openMPI 并行计算非常简单。但是如果通过计算机网络来并行计算，我们必须创建一个包括宿主机器名字的文件。它可以放在任意位置，文件名随意。在下面的讨论中我们用 <machines> 代表这个包含全路径的文件。

在 <machines> 文件中，需要把每个使用 openMPI 并行求解的计算机名单行列出。这个名字应该和机器本身的/etc/hosts 文件中的名字相对应。这个文件必须包含这些名字。当某个节点内有多个核心可供计算的时候，在这个节点的名字后面还必须包含 cpu = n，其中 n 即为 openMPI 可以使用的处理器数量。

例如，如果我们打算在 aaa 的机器上使用 openMPI 在 aaa、bbb（具有两个处理器）、ccc 上并行计算。<machines> 文件应该是这样的：

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

然后使用下面的语句来运行：

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

其中：<nProcs> 是核心数量；<foamExec> 是可执行程序的名称，例如 icoFoam；log 为输出的日志文件。举例，如果使用 icoFoam 在 4 个节点（其由 machines 指定）上计算 cavity 算例（其位于 $FOAM_RUN/tutorials/incompressible/icoFoam 文件夹中），我们应该执行

下面的命令[9]：

```
mpirun --hostfile machines -np 4 icoFoam -parallel > log &
```

3.4.4 多硬盘数据阵列分布

用户有的时候可能打算在不同的硬盘里写入数据，在这种情况下，我们说写入的数据是阵列分布的。在进行分布式写入数据的时候，用户会发现在不同的节点中，写入数据的根目录是不同的。因此，在 decomposePar 字典文件中我们需要指定 distributed 和 roots 关键词，distributed 关键词这样指定：

```
distributed yes;
```

roots 下要包含每个节点的数据路径，例如 <root0>，<root1>

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

其中 <nRoots> 是路径数。

依据 decomposeParDict 字典文件的设置，并行算例中每个处理器的计算路径下都应该包含相应的 processorN 文件夹。除此之外，system 文件夹及 constant 文件夹也应该存在。需要注意：polyMesh 文件夹可以省略，但除了 polyMesh 文件夹其他 constant 下的文件都应该被包含。

3.4.5 并行后处理

当一个算例以并行的方式来计算的时候，用户有下面两种方式来进行后处理：

- 重组网格场和数据以呈现完整的网格和场，这样就可以正常的进行后处理；
- 单独对每个域进行后处理。

3.4.5.1 重组网格和数据

当算例并行计算完毕，就可以重组来进行后处理了。算例中每个处理器下的时间步文件，重组后会被整合到这个算例下的相应的时间步文件夹下。重组程序 reconstructPar 可以这样来执行：

```
reconstructPar
```

当数据分布在几个硬盘中的时候，需要把他们都拷贝在宿主机再进行重组。

3.4.5.2 分解场后处理

正如7.1节所说，用户可以使用 paraFoam 来对每个核下的算例来进行单独后处理。并行计算的算例可以通过重组场的方式来后处理，以及对每个分解场进行单独后处理，这样的话，每个核下

[9]另外一种可以加速并行计算的方式是采用 renumberMesh 来进行预处理，请参考：

面的结果会被当做一个单独的算例来看待。

3.5 标准求解器

OpenFOAM 的求解器源代码位于 `$FOAM_SOLVERS` 文件夹下面，这可以通过 `sol` 命令快速的切换到此处。这个文件夹还依据模型类别的不同来区分为不同的子文件夹，例如不可压缩流体求解器类、燃烧求解器类、应力分析求解器类等。每个求解器的名字都非常具有描述性，例如 `icoFoam` 就是求解 `incompressible flow`（不可压缩流体）的求解器。

3.5.1 基本求解器

`laplacianFoam` 拉普拉斯方程求解器，例如固体中的导热问题。

`potentialFoam` 速度势求解器，可通过求解的通量重组速度场。

`scalarTransportFoam` 瞬态或稳态被动标量传输方程求解器。

3.5.2 不可压缩求解器

`adjointShapeOptimizationFoam` 不可压缩非牛顿流体依据压力损失进行几何优化的求解器（参见：Implementation of a continuous adjoint for topology optimization of ducted flows）。

`boundaryFoam` 稳态不可压缩一维湍流求解器，主要用于为进口生成边界层条件。

`icoFoam` 牛顿流体瞬态不可压缩求解器。

`nonNewtonianIcoFoam` 非牛顿流体瞬态不可压缩求解器。

`pimpleFoam` 采用 PIMPLE 算法的大时间步瞬态不可压缩流求解器，支持动网格。

`SRFPimpleFoam` 支持单旋转参考系的 pimpleFoam。

`pisoFoam` 采用 PISO 算法的瞬态不可压缩流求解器。

`shallowWaterFoam` 瞬态无粘有旋浅水方程求解器。

`simpleFoam` 稳态不可压缩湍流求解器。

`porousSimpleFoam` 支持 MRF 以及多孔介质的 simpleFoam。

`SRFSimpleFoam` 支持单旋转参考系的 simpleFoam。

3.5.3 可压缩求解器

`rhoCentralFoam` 基于 Kurganov&Tadmor 中心迎风格式的密度基可压缩湍流求解器，支持动网格。

`rhoPimpleFoam` 基于 PIMPLE 算法的可压缩流求解器（暖通类应用），支持动网格。

`rhoSimpleFoam` 基于 SIMPLE 的附带 RANS 湍流模型的稳态可压缩求解器。

`rhoPorousSimpleFoam` 附带多孔介质模型的 rhoSimpleFoam。

3.5.4 多相流求解器

`cavitatingFoam` 基于均相平衡模型（用于计算液体/蒸汽的混合可压性）的气蚀求解器，支持动网格。

`compressibleInterFoam` 基于 VOF 模型的可压、绝热、不可溶两相界面捕获求解器，支持动网格。

`compressibleMultiphaseInterFoam` 基于 VOF 模型的可压、绝热、不可溶可压多相界面捕获求解器，支持 MRF。

`driftFluxFoam` 模拟沉降以及类似分离现象的不可压缩两相求解器。

`interFoam` 基于 VOF 模型的不可压、绝热、不可溶两相界面捕获求解器，支持 LTS，MRF，动网格。

`interMixingFoam` 三相不可压缩流体，其中两相互溶，基于 VOF 模型的界面捕获求解器，支持动网格。

`multiphaseEulerFoam` 求解任一数量的可压缩流体相具有共同的压力，但性质不同的系统。可以运行时选择不同的相间反应模型以及动量，热量和质量传递模型。

`multiphaseInterFoam` 基于 VOF 模型的可压、绝热、不可溶不可压多相界面捕获求解器，支持 MRF，支持动网格。

`potentialFreeSurfaceFoam` 包含波高（zeta）的不可压缩 NS 方程求解器，可用于在单相下模拟自由表面的波高，支持动网格。

`twoLiquidMixingFoam` 不可压缩可溶两相混合求解器。

3.5.5 直接模拟求解器

`dnsFoam` 计算域为立方体的各向同性湍流直接模拟求解器。[10]

3.5.6 燃烧求解器

`chemFoam` 单网格化学反应求解器，主要用于和其他软件[11]的求解结果相对比。网格和场从初始条件即时生成。

`coldEngineFoam` 冷态内燃机求解器。

`engineFoam` 附加喷雾粒子云的内燃机求解器。

`PDRFoam` 附带湍流的可压预混燃烧求解器。

`buoyantReactingFoam` 密度基于热力学模型、浮力强化模型的燃烧求解器，附带化学反应模型。

`reactingFoam` 附带化学反应的燃烧求解器。

`XiEngineFoam` 内燃机求解器。

`XiFoam` 附带湍流模型的可压预混/部分预混燃烧求解器。

[10] 使用此求解器需要保证计算域的每个方向网格需要为 2 的 n 次幂，有关采用 OpenFOAM 进行 Quasi-DNS 模拟请参考：本链接

[11] 例如 chemkin

3.5.7 传热求解器

buoyantPimpleFoam 附加湍流的瞬态可压浮力驱动求解器（暖通和传热）。

buoyantSimpleFoam 附加湍流的稳态可压浮力驱动求解器（暖通和传热）。

chtMultiRegionFoam 附加湍流的瞬态/稳态固体-流体多域传热求解器。

thermoFoam 流场固定，单一求解能量方程的求解器。

3.5.8 颗粒跟踪求解器

denseParticleFoam 瞬态粒子云耦合传输求解器，包括粒子体积分数对连续相的影响，支持动网格.

particleFoam 瞬态单颗粒团被动传输的求解器，支持动网格.

rhoParticleFoam 颗粒团被动输运瞬态求解器。

3.5.9 分子动力学模拟

mdEquilibrationFoam 分子动力系统平衡或预处理求解器。

mdFoam 用于流体动力学的分子动力学求解器。

dsmcFoam 顺态多组分流动 DSMC 求解器。

3.5.10 电磁求解器

electrostaticFoam 静电场求解器。

magneticFoam 永磁场的磁场求解器。

mhdFoam 不可压缩，层流，磁力驱动 MHD 求解器。

3.5.11 应力分析求解器

solidDisplacementFoam 刚体微小线弹性应变的有限体积瞬态分离式求解器，可选热扩散和热应力。

solidEquilibriumDisplacementFoam 刚体微小线弹性应变的有限体积稳态分离式求解器，可选热扩散和热应力。

3.5.12 金融分析求解器

financialFoam 期权定价 Black-Scholes 方程求解器。

3.6 标准工具

OpenFOAM 附带的程序位于 `$FOAM_UTILITIES` 目录中，我们可以在终端键入 `util` 快速的切换到文件目录。程序的名字带有描述性，例如 `ideasToFoam` 就是把 `IDEAS` 网格转换为 OpenFOAM 可读取的网格形式。

3.6.1 前处理工具

`applyBoundaryLayer` 依据当前的速度场通过简单的七分之一定律计算边界层。

`boxTurb` 创建符合连续性方程的各项同性湍流场[12]。

`changeDictionary` 用来批处理改变字典信息，例如可以用来修改 polyMesh/boundary 文件内的场的类型。

`createExternalCoupledPatchGeometry` 和 externalCoupled 边界条件一起使用，用来为某个场生成边界几何。

`dsmcInitialise` 通过读取 system/dsmcInitialise 字典文件来为 dsmcFoam 进行初始化。

`engineSwirl` 为发动机计算生成涡流场。

`faceAgglomerate` 通过 pairPatchAgglomeration 算法对边界面进行融合。

`foamSetupCHT` 通过模板文件设置多域算例的文件。

`foamUpgradeCyclics` 对 cyclics 边界面的网格和场进行更新。

`mapFields` 对某个网格上的场进行读取并插值最后映射到其他算例，其中时间步一一对应。对于并行算例，不需要重组算例即可运行。

`mapFieldsPar` 对某个网格上的场进行读取并插值最后映射到其他算例，其中时间步一一对应。

`mdInitialise` 分子模拟初始场。

`setFields` 通过本身的字典文件对网格和边界的场值进行设置。

`setWaves` 应用波模型与水平集对波进行初始化。

`viewFactorsGen` 基于融合（agglomerated）面计算角系数辐射模型的辐射角。

`wallFunctionTable` 读取一个字典文件来计算合适的壁面函数值列表。

3.6.2 网格生成

`blockMesh` 多块网格生成器。

`extrudeMesh` 从现有的 patch 或者几何文件上的 patch 上抽取网格，默认为对法向延伸。

`extrude2DMesh` 在一个 2D 网格上依据厚度建立一个 3D 网格。

`extrudeToRegionMesh` 把网格的 faceZones 抽取为一个单独的网格，可以用来生成液膜域网格。

[12] 各个方向的网格数需要为 2 的幂

`foamyHexMesh` Voronoi 网格自动生成器[13]。

`foamyQuadmesh` Voronoi 二维抽取网格生成器。

`snappyHexMesh` 自动进行表面贴合细化的六面体网格切分器。

3.6.3 网格转换

`ansysToFoam` 把 I-DEAS 制作的 ANSYS 的网格转换为 OpenFOAM 可用格式。

`cfx4ToFoam` 把 CFX 网格转换为 OpenFOAM 可用格式。

`datToFoam` 读取 datToFoam(.dat) 网格文件并输出点文件，一般和 blockMesh 结合使用。

`fluent3DMeshToFoam` 把 fluent 网格转换为 OpenFOAM 可用格式。

`fluentMeshToFoam` 把 fluent2D 网格转换为 OpenFOAM 可用格式。

`foamMeshToFluent` 把 OpenFOAM 网格转化为 fluent 网格格式。

`foamToStarMesh` 把 OpenFOAM 网格转化为 PROSTAR 网格格式。

`foamToSurface` 读取 OpenFOAM 网格并把边界转化为几何。

`gambitToFoam` 把 gambit 网格转换为 OpenFOAM 可用格式。

`gmshToFoam` 把.msh 网格转换为 OpenFOAM 可用格式。

`ideasUnvToFoam` 把.I-Deas 网格转换为 OpenFOAM 可用格式。

`kivaToFoam` 把 KIVA 网格转换为 OpenFOAM 可用格式。

`mshToFoam` 把 Adventurede 生成的.msh 网格转换为 OpenFOAM 可用格式。

`netgenNeutralToFoam` 把 Netgen 4.4 生成的.msh 网格转换为 OpenFOAM 可用格式。

`plot3DToFoam` 把 Plot3d 网格转换为 OpenFOAM 可用格式。

`sammToFoam` 把 STAR-CD(v3) 的 SAMM 网格转换为 OpenFOAM 可用格式。

`star3ToFoam` 把 STAR-CD(v3) 的 PROSTAR 网格转换为 OpenFOAM 可用格式。

`star4ToFoam` 把 STAR-CD(v4) 的 PROSTAR 网格转换为 OpenFOAM 可用格式。

`tetgenToFoam` 把 tetgen 生成的.ele, .node, .face 文件转换为 OpenFOAM 可用格式。

`vtkUnstructuredToFoam` 把 paraview 生成的.vtk 文件转换为 OpenFOAM 可用格式。

`writeMeshObj` 网格 debug 程序，把网格写为三个独立的 OBJ 格式文件并可用 paraview 打开。

[13]有关 Voronoi 多边形：本链接

3.6.4 网格处理

`attachMesh` 将拓扑分离的网格连接。

`autoPatch` 依据特征角把外部网格面切分为 patch。

`checkMesh` 检查网格。

`createBaffles` 把内部面转化为边界面。和 mergeOrSplitBaffles 不同，其不复制点。

`createPatch` 在现有的网格面上（从存在的 patch 上或者 faceSet 面上）创建 patch。

`deformedGeom` 依据速度场通过一个放大因子对网格进行变形。

`flattenMesh` 把一个 2D 网格的前后面平面化。

`insideCells` 选择网格中心处于一个面内的网格，这个面必须是封闭的。

`mergeMeshes` 将两套网格合并。

`mergeOrSplitBaffles` 检测共享点的面（例如挡板），合并他们并复制。

`mirrorMesh` 针对一个平面做网格镜像。

`moveDynamicmesh` 网格拓扑运动运行工具。

`moveEngineMesh` 用于发动机计算的动网格运行工具。

`moveMesh` 动网格运行工具。

`objToVtk` 读取 obj 的线并转化为 vtk 格式。

`orientFaceZone` 修正面域方向。

`polyDualMesh` 创建多面体网格，保留特征面和边界面。

`refineMesh` 在多个方向细化网格。

`renumberMesh` 从所有的时间步进行读取，并对网格单元序列重数以减少带宽。

`rotateMesh` 把网格从 n1 方向旋转到 n2 方向。

`setSet` 对 cell/face/point/set 进行操作。

`setsToZones` 把 pointSets/faceSets/cellSets 转化为 pointZones/faceZones -/cellZones。

`singleCellMesh` 读取场数据并把他们映射到一个移除内部场的网格上（singleCellFvMesh），这个网格被写入为 singleMesh。主要用来为边界层生成网格以及场数据。本工具很容易产生坏网格，因此一般只用于在 paraview 中可视化。

`splitMesh` 使用 attachDetach 把内部面外部化。

`splitMeshRegions` 切分多域网格。

`stitchMesh` 网格缝合。

`subsetMesh` 基于 cellSet 选择网格。

`topoSet` 通过字典文件对 cellSets/faceSets/pointSets 进行划分。

`transformPoints` 对网格点的位置坐标进行操作，例如缩放、旋转等。

`zipUpMesh` 读取网格并缝合网格单元，确保有效的多面体网格单元封闭。

3.6.5 其他网格工具

`autoRefineMesh` 细化表面附近网格。

`collapseEdges` 短边坍塌，把一条线上的边结合。

`combinePatchFaces` 检查网格上的多个面并把他们结合。多余的面可能产生于移除相邻网格单元的过程中（留下宿主网格单元的 4 个面）。

`modifyMesh` 操作网格单元。

`PDRMesh` 用于 PDR 模拟的网格及场操作。

`refineHexMesh` 对六面体进行 2*2*2 细化。

`refinementLevel` 在进行表面贴合之前，对细化等级等指标进行输出。

`refineWallLayer` 对 patch 附近的网格进行细化。

`removeFaces` 移除网格单元的面。

`selectCells` 选择和面相关的网格单元。

`splitCells` 使用平面切割网格单元

3.6.6 后处理

`engineCompRatio` 计算几何压缩比。注意，如果您有阀门和/或额外的体积，它将无法工作，因为它计算 BDC 和 TCD 的体积。

`pdfPlot` 生成概率密度函数图。

`postChannel` 管道流后处理工具。

`temporalInterpolate` 从不同的时间步中进行场插值。

`noise` 通过 noiseFFT 库进行压力场噪音分析。

`postprocess` 通过命令行的方式对时间步下的场执行算例 system/control 文件下的 functionObjects 函数功能。

`particleTracks` 生成颗粒团的 VTK 数据文件，务必要重组结果后才能进行。

`steadyParticleTracks` 生成稳态颗粒团的 VTK 数据文件，务必要重组结果后才能进行

3.6.7 数据后处理

`foamDataToFluent` 转换 OpenFOAM 数据为 Fluent 可用格式。

`foamToEnsight` 转换 OpenFOAM 数据为 EnSight 可用格式。

`foamToEnsightParts` 转换 OpenFOAM 数据为 EnSight 可用格式并为某个网格区域和 patch 创建 Ensight 部件。

`foamToGMV` 转换 OpenFOAM 数据为 GMV 可用格式。

`foamToTetDualMesh` 将 polyMesh 结果转换为 tetDualMesh。

`foamToVTK` VTK 文件生成器。

`smapToFoam` 将 STAR-CD SMAP 程序转换为 OpenFOAM 可用格式

3.6.8 面处理工具

`surfaceAdd` 通过点融合操作来添加两个面，不检查三角形的重叠与否。

`surfaceAutoPatch` 通过特征角对表面进行分片，类似 autoPatch。

`surfaceBooleanFeatures` 在面方向正确的情况下，生成 extendedFeatureEdgeMesh 文件。

`surfaceCheck` 检查面几何和拓扑。

`surfaceClean` 移除挡板，小边坍塌，移除三角形，通过切分边、通过映射点把微小的三角形转化，几何清理。

`surfaceCoarsen` 使用 bunnylod 方法进行表面糙化。

`surfaceConvert` 转换面网格格式。

`surfaceFeatureConvert` 转换 edgeMesh 格式。

`surfaceFeatures` 识别几何的特征并写入文件中。

`surfaceFind` 寻找最近的面和顶点。

`surfaceHookUp` 把最近的开放边缝合到附近的面。

`surfaceInertia` 计算指定几何（刚体或者薄层）的惯性张量、主轴和矩。。

`surfaceLambdaMuSmooth` 使用 lambda/mu 方法来平滑表面，需要设置 lambda 的松弛因子以及将 mu 的松弛因子设为 0。

`surfaceMeshConvert` 转换几何表面格式，可以进行放大或者旋转等操作。

`surfaceMeshConvertTesting` 转换几何表面格式，主要用于测试函数。

`surfaceMeshExport` 将 surfMesh 输出为第三方格式，可以进行放大或者旋转等操作。

`surfaceMeshImport` 将第三方格式输出为 surfMesh，可以进行放大或者旋转等操作。

`surfaceMeshTriangulate` 从 polyMesh 生成由三角形构成的几何面。依据输出格式进行三角形切分。三角形的数量和 polyMesh 中的 patch 数量一致，也可以选择性的生成几何。

`surfaceMeshInfo` 输出面网格相关信息。

`surfaceOrient` 面法向调整（面法向和用户提供的外部点一致，也可以为内部点，需要使用-inside 命令参数）。

`surfacePointMerge` 把面中某距离内的点进行融合，注意其为绝对距离。

`surfaceRedistributePar` 几何重新分布（对已经分解的表面或者没有分解的表面进行操作），以使得三角面和网格重叠。

`surfaceRefineRedGreen` 将三角形的三个边进行切分（red 细化），详见：(A review of a posteriori error estimation and adaptive mesh refinement techniques", Wiley-Teubner, 1996)。

`surfaceSplitByPatch` 依据 patch 名称将几何文件分割。

`surfaceSplitByTopology` 依据挡板将几何文件分割。

`surfaceSplitNonManifolds` 通过复制点来切分边，写入 borderEdge（一个边连接 4 个面）、borderPoint（一个点和两个 borderEdge 相连）、borderLine（borderEdges 的列表）。

`surfaceSubset` 面几何分析工具，依据 subsetMesh 建立感兴趣的一部分子几何。

`surfaceToPatch` 读取面几何并将面域应用到网格，困难的操作使用 boundaryMesh 处理。

`surfaceTransformPoints` 和 transformPoints 一样，用来旋转面几何

3.6.9 并行后处理

`decomposePar` 为 OpenFOAM 并行运行自动分解网格以及场。

`redistributePar` 对当前分解的网格和场通过 decomposeParDict 进行重新分配。

`reconstructParMesh` 对网格进行重组。

`redistributePar` 读取 decomposeParDict 字典文件对当前已经分解的网格进行重新分解

3.6.10 热物理模型程序

`adiabaticFlameT` 计算绝热火焰温度。

`chemkinToFoam` 将 CHEMKIN3 热力反应数据转换为 OpenFOAM 格式。

`equilibriumCO` 计算一氧化碳的平衡等级。

`equilibriumFlameT` 对给定燃料在一系列的未燃温度和等值比下计算平衡火焰温度，需要考虑氧气、水、二氧化碳的分离性。

`mixtureAdiabaticFlameT` 指定温度指定混合率下计算绝热火焰温度

3.6.11 其他程序

`foamDictionary` 操控字典文件。

`foamFormatConvert` 读取 controlDict 文件中的格式设置，将所有输出场的类型进行转换。

`foamInfoExec` 查询和打印信息输出到标准输出。

`patchSummary` 输出算例时间步下计算之后的场信息、边界信息

第4章　OpenFOAM 算例

这一章我们分析 OpenFOAM 算例的文件结构。一般来讲，用户需要对每个算例命名，例如顶盖驱动流我们可以称之为 `cavity`。`cavity` 目录就是这个算例所有文件存储的地方。算例文件放在什么地方都可以，但是我们建议把它们放在 `run` 文件夹内。例如第二章提到的 `$HOME/OpenFOAM/$USER-9` 文件夹内。这样放置算例的一个优点是 `$FOAM_RUN` 环境变量默认设置为 `$HOME/OpenFOAM/$USER-9/run`，因此用户可以快速的使用 `run` 命令切换到这个文件夹。

OpenFOAM 的典件结构可参考其附带的各种算例，他们位于 `$FOAM_TUTORIALS` 文件夹下，可以执行 `tut` 命令快速的切换到此文件夹。用户在阅读这个章节的时候可以顺便看看这些算例的文件夹结构。

4.1　OpenFOAM 文件结构

OpenFOAM 算例的基本文件结构包含了所必须的文件集。如图4-1所示：

`constant` 文件夹下包含了程序所需要的物理特性文件例如 `transportProperties`，`constant` 下的 `polyMesh` 文件夹包含了网格数据文件。

`system` 文件夹主要用于设置求解算法的参数。它至少包含三个文件：`controlDict`：用于控制求解开始/终止时间，时间步，以及输出数据参数；`fvSchemes` 包含各种离散格式；`fvSolution` 包含矩阵求解器设置、残差、以及其它算法控制[1]。

`Time` 一系列文件夹中包含了某些场的计算结果。其中包括必须指定的初始数据以及边界条件，以及 OpenFOAM 求解器的计算结果。值得一提的是即使在稳态问题中（稳态问题只需要边界条件），OpenFOAM 的初始场也必须被初始化。每个时间步的名称由模拟结果决定，这在4.4节有详细说明。由于我们经常在 `0` 时刻进行模拟，因此初始条件通常存储在 `0` 文件夹或者 `0.00000e+00` 中（取决于指定的文件名格式）。例如，`cavity` 算例中，速度场和压力场存储在 `0/U` 和 `0/p` 文件夹中。

4.2　基本输入输出格式

OpenFOAM 需要读取一系列数据，例如字符串、标量、矢量、张量、列表和场。OpenFOAM 输入输出机制很灵活，用户可以对其进行修改。不同于其它软件很难理解的输入输出系统。OpenFOAM 的输入输出机制遵守一些简单的规则，它们是非常容易理解的。下面我们来阐述 OpenFOAM 的文件格式。

4.2.1　通用语法规则

遵循 C++ 的一些基本原则：

[1]老版本 OpenFOAM 中 blockMesh 字典文件必须放在 constant 文件夹中，OpenFOAM-4.0 之后 blockMesh 文件可以放在 system 文件夹中

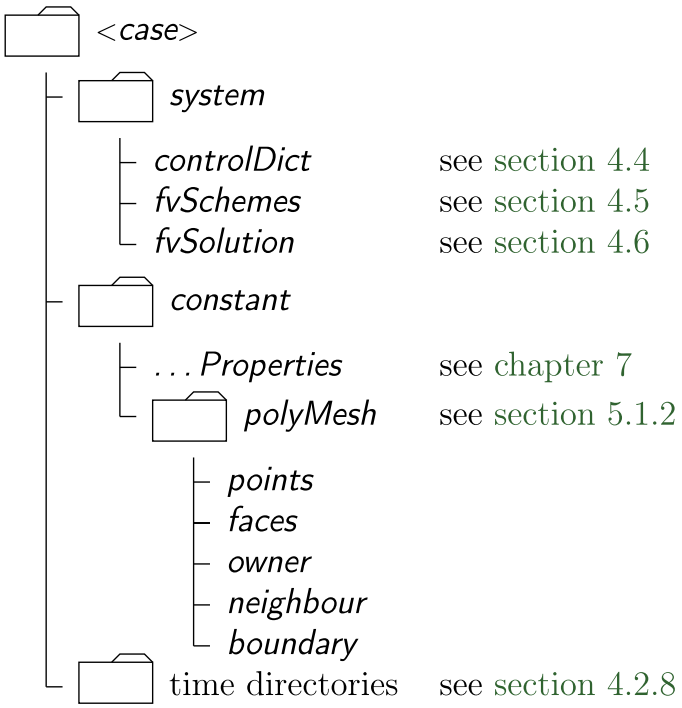


图 4-1: 算例文件结构

- 文件形式任意，书写自由，不需要对列整齐，其会忽略分行信息[2]；
- // 这个符号意味着 OpenFOAM 将会忽略这一行的内容直到行尾；
- 多行的注释由/*……*/来完成。

4.2.2 字典

OpenFOAM 使用字典来输入数据。字典文件可以依靠关键词 `keywords` 来进行输入和输出（`I/O`），关键词遵循如下形式：

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

大多数关键词只有一个数据：

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

某些 OpenFOAM 的字典文件本身又包含了一系列带有关键词的子字典。字典的构建相当具有逻辑性和继承性，即字典本身也可以包含一个或者更多的子字典。字典的形式是在字典名称之后用 把字典内的关键词信息包含进来，例如：

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

[2]单行写入代码和多行写入代码是一样的，参见 107 页4.2.5节一个例子

4.2.3 文件头

所有 OpenFOAM 读写的文件都以 FoamFile 开头，它包含一系列关键词信息，见表4-1。此表对其进行了简单的描述，表中我们提供了简要的可供关键词使用的信息（entry），他们适用于大部分情况。

关键词	描述	信息
version	输入输出格式版本号	2.0
format	数据格式	ascii/binary[3]
location	文件路径，“…”	可选
class	OpenFOAM 文件构建的类	一般为 dictionary 或者场文件例如 volVectorField
object	文件名	例如 controlDict

表 4-1: 文件头信息

class 关键词中的信息就是表明这个文件是构建一个类。如果用户不知道使用代码中的哪一部分来调取这个文件，也不熟悉 OpenFOAM 的类，那他可能不知道这个类名。然而大多数需要指定关键词的数据文件都通过 dictionary 类来表示，因此大部分情况下 class 关键词的信息为 dictionary[4]。

下面这个例子我们来介绍关键词用法,它告诉用户如何用关键词为算例提供数据。fvSolution 字典文件包含两个子字典，solvers 和 PISO。solvers 字典包含了求解器需要的大量数据，以及压力和速度方程需要的残差（他们在 p 和 U 关键词里面指定）。PISO 字典包含了算法控制信息。

```
18  solvers
19  {
20      p
21      {
22          solver                      PCG;
23          preconditioner              DIC;
24          tolerance                   1e-06;
25          relTol                      0;
26      }
27
28      pFinal
29      {
30          $p;
31          relTol                      0;
32      }
33
34      U
35      {
36          solver                      smoothSolver;
37          smoother                    symGaussSeidel;
38          tolerance                   1e-05;
39          relTol                      0;
40      }
41  }
42
43  PISO
44  {
45      nCorrectors                     2;
46      nNonOrthogonalCorrectors        0;
47      pRefCell                        0;
48      pRefValue                       0;
49  }
```

4.2.4 列表

OpenFOAM 的程序包含很多列表。例如描述网格坐标的列表。列表通常在输入输出的文件中以特有的形式存在，通常由 () 包含。在列表的 () 之前也需要遵守一定的格式：

[3]binary 格式体积小且精度高，对于某些需要高精度的算法，binary 更能保证求解稳健

[4]对于数据场通常为 vol<type>Field

- 单一列表：关键词下面尾随着圆括号

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

- 复合列表：关键词后面由 <n> 指定列表数

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

- 占位标识：

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

值得注意的是 `list<scalar>` 里面的 `<scalar>` 不是一个大的类名（例如 `scalar`，`double` 等），而是一个特定的数。

单一列表用起来非常方便。但是其它形式运行起来更快，因为列表的大小在读取数据之前就分配了内存。因此，小列表通常使用单一列表的形式，这样读取时间最小。长列表建议使用其它的复杂形式。

4.2.5 Scalar 标量、Vector 矢量、Tensor 张量

`scalar` 标量就是一个数。`vector` 矢量是一个三维 1 阶的矢量空间 `vectorSpace`。由于维数固定为 3。通常我们使用最简单的列表形式。因此，一个矢量 `(1.0, 1.1, 1.2)` 写成如下形式：

```
(1.0 1.1 1.2)
```

在 OpenFOAM 中，`tensor` 张量是一个三维 2 阶量，因此一个张量有 9 个实数。它通常写成如下形式：

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

下面这个例子表明 OpenFOAM 会自动忽略行信息，所以上面这个张量写成单行形式也是一样的：

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

4.2.6 量纲

在连续介质里，物理量都具有量纲。例如质量：kg, 体积：m^3，压力：Pa。这些物理量的代数几何操作要求量纲一致。对于同样量纲的物理量，只有 +、-、= 是具有物理意义的。为了防止对物理量进行无意义或者错误的运算，OpenFOAM 将量纲依附于场数据以及某个物理量，在对其进行张量操作的时候执行量纲检查。

dimensionSet 的输入输出格式设置为方括号内有限的 7 个标量。例如：

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

编号	名称	SI 单位	USCS 单位
1	质量	kg	lbm
2	米	m	ft
3	时间	s	s
4	开尔文	K	Rankine·R
5	物质的量	kg/mol	lbmol
6	电流	A	A
7	光强	cd	cd

表 4-2: SI 和 USCS 标准量纲

量纲括号 [ ] 内的每个值代表每个物理单位的幂，见表4-2。这个表是采取 SI 单位制以及 USCS 单位制，但 OpenFOAM 可以使用任何量纲单位制。你所需要做的就是确保输入数据准确。另外，OpenFOAM 有时需要一些带有量纲的常数。例如热物理模型中的热力学常数 R。它的单位已经在 OpenFOAM 安装目录（$WM_PROJECT_DIR/etc/controlDict）下 controlDict 的 DimensionedConstant 子字典下指定。它默认使用 SI 单位制。想使用 USCS 单位或者其它单位制的应该修改这个文件进行单位匹配。

4.2.7 单位类型

物理量要依附于量纲。接口以如下形式表示，以 dimensionedScalar 为例：

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

第一个 nu 是关键字，第二个 nu 是存储在 word 类中的 word 名，通常和第一个关键字一样，下一个带中括号的是 dimensionSet，最后一个是 nu 的值。在 OpenFOAM-4.0 之后的版本中，word 名（第二个 nu）可以省略，即：

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

4.2.8 场

大多数 OpenFOAM 输入输出的数据是张量场，如速度、压力，会写入时间步文件。OpenFOAM 采用下列关键词来写入数据，见表4-3：

关键词	描述	举例
量纲	场的量纲	[1 1 -2 0 0 0 0]
内部场	内部场的值	uniform (1 0 0)
边界场	边界场	参见4.2.8节的列表

表 4-3: 场文件内的主要关键词

数据首先以 dimensions 为接口，内部场尾随其后。它有两种类型：

均一场 场内所有网格内被指定了一个值，采用如下形式表示：

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

非均一场 每个网格单元有不同的值，采取如下形式表示：

```
internalField nonuniform <Lists>;
```

边界场 boundaryField 包含了一系列 patch（面），它们和 polyMesh 文件夹下 boundary 文件里的 patch 相对应。每个 patch 是一个字典文件，包含了一系列关键词。type 后面必须要指定这个边界的边界条件类型。在确定边界类型后，可以指定具体的边界信息作为初始条件。6.2.1至6.2.3节列举了 OpenFOAM 可用的边界条件以及这些边界条件必须指定的内容。下面这个速度场 U 的字典可以作为一个例子来说明：

```
17  dimensions      [0 1 -1 0 0 0 0];
18
19  internalField   uniform (0 0 0);
20
21  boundaryField
22  {
23      movingWall
24      {
25          type    fixedValue;
26          value   uniform (1 0 0);
27      }
28
29      fixedWalls
30      {
31          type    noSlip;
32      }
33
34      frontAndBack
35      {
36          type    empty;
37      }
38  }
```

4.2.9 宏

OpenFOAM 在设置算例的过程中，可以使用宏，他们具有很大的灵活性。在使用宏的时候需要在关键词前面使用符号 $。例如如果我们的关键词 a 的值为 10，那么下面的宏将 b 的值也定义为 10：

```
a 10;
b $a;
```

变量也可以存储在不同的子字典内。下面的例子表示在 subdict 子字典中定义 a，b 的值通过 subdict 中的 a 来赋值：

```
subdict
{
  a 10;
}
b $subdict.a;
```

下面是一些衍生的宏语法：

- 如果 b 定义在子字典中，则需要使用 .. 前缀；
- 如果 b 定义在二级子字典中，则需要使用 …前缀；
- 如果要溯源到顶层字典中，可以使用 : 前缀；
- 对于多级别宏替换，需要结合使用 . 和 $。

下面为一个范例：

```
a 10;
b a;
c ${${b}}; // returns 10, since $b returns "a", and $a returns 10
subdict
{
  b $..a; // double-dot takes scope up 1 level, then "a" is available
  subsubdict
  {
    c $:a; // colon takes scope to top level, then "a" is available
  }
}
```

4.2.10 文件包含

在设置 OpenFOAM 算例的时候，可以灵活地进行文件包含。文件包含可以通过 # 号来进行。举例，用户想在边界处为内部压力场初始化。我们可以创立一个文件叫做 initialConditions，包含以下信息：

```
pressure 1e+5;
```

为了让内部场和初始场都使用这个压力值，用户可以简单的把上面的压力场替换到下面的地方：

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

目前 OpenFOAM 可以调用五种文件包含命令：

- include "<path>/<fileName>"，从 <path> 路径中读取 <fileName> 文件；
- includeIfPresent "<path>/<fileName>"，如果文件存在的话，那么就从 <path> 路径中读取 <fileName> 文件；
- includeEtc "<path>/<fileName>"，从 $FOAM_ETC 下指定的 <path> 路径中读取 <fileName> 文件；
- includeFunc <fileName>，如果文件存在的话，则从算例的 system 文件中读取 <fileName> 文件，若文件不存在，从 $FOAM_ETC 下读取 <fileName> 文件；
- remove <keywordEntry> 将关键词信息去掉；可以用一个单词或者一个表达式来表述。

4.2.11 环境变量

OpenFOAM 可以识别输入文件中的环境变量。例如，正如第二章所讲的，run 文件夹下的内容可以通过 $FOAM_RUN 来获取。例如：

```
include ''FOAM_RUN/pitszDaily/0/U''
```

除了类似的 $FOAM_RUN，OpenFOAM 还可以识别一些其他的内置环境变量，如

$FOAM_CASE 当前算例的路径；

$FOAM_CASENAME 当前算例路径的名字；

$FOAM_APPLICATION 求解器的名字。

4.2.12 正则表达式

在运行程序的时候，需要在字典内初始化数据。用户可以对每一个关键词逐一列出并各个进行初始化，或者也可以通过 POSIX 常规表达式来更方便的给定数据。这种方式通常需要 (“…”) 表达式。POSIX 正则表达式具有大量不同的写法，但是在 OpenFOAM 中，我们一般只选择下面几个方案：

"inlet.*" 对任意的以 inlet 为开头的关键词进行匹配，其中的. 代表任意字符，其中的 * 表示重复任意次；

"(inlet|outlet)" 指定 inlet 和 outlet 为同样的数据，因为 () 为分组操作符，| 为“或”表达式。

4.2.13 关键词顺序

当同一个关键词被给定多次的时候，OpenFOAM 调用最后一个关键词的信息。在下面一种情况下，经常会发生这种情况。比如用户针对某个关键词，采用了部分的宏替换，而偏偏这个宏内的某些关键词和未被宏替换的关键词重复，那 OpenFOAM 则会读取最后一个关键词的信息。下面这个例子可以帮助用户更好的理解这个过程：

```
p
{
    solver          PCG;
    preconditioner  DIC;
    tolerance       1e-6;
    relTol          0.05;
}

pFinal
{
    $p;
    relTol          0;
}
```

上述的代码中，pFinal 中的的 $p 将 p 关键词中的所有信息全部复制进来，包括 p 中指定的 relTol 关键词，但是在 pFinal 中随后又指定了 relTol 关键词，在这种情况下，OpenFOAM 默认读取 relTol 关键词的信息。

4.2.14 内嵌代码

OpenFOAM 还提供功能更强大的内嵌代码功能，如 calc 可以用于简单计算，codeStream 可用于复杂计算。

$FOAM_TUTORIALS/incompressible/simpleFoam 中的 pipeCyclic 算例中的 blockMeshDict 字典文件表明了如何通过 #calc 来进行简单计算：

```
//- Half angle of wedge in degrees
halfAngle 45.0;

//- Radius of pipe [m]
radius 0.5;

radHalfAngle    #calc "degToRad($halfAngle)";
y               #calc "$radius*sin($radHalfAngle)";
minY            #calc "-1.0*$y";
z               #calc "$radius*cos($radHalfAngle)";
minZ            #calc "-1.0*$z";
```

上述代码中包含了几个简单的计算可用于从 radius 的值来计算定点的坐标如 z，y 等。在 #calc 中可以调用标准的 C++ 函数如 degToRad，以及 sin 函数等[5]。

codeStream 指令包含的相关字典信息可以用来当做标准的 C++ 代码来编译和执行。它需要指定以下关键词：

code 指定代码部分。在代码中可使用 OStream& os 进行输出或通过 const dicti-onary& dict 调用字典。例如：其可用于调用当前算例的信息；

codeInclude（**可选**） 用 include 包含指定的 C++ 代码；

codeOptions（**可选**） 指定附加头文件路径，参考 Make/options 的 EXE_LNC 中的格式；

codeLibs（**可选**） 指定附加库。参考 Make/options 的 LIB_LIBS 中的格式。

代码可以用... 来包含起来，并可以多行书写。下面是一个 codeStream 的例子，在 controlDict 字典文件里面的这个代码表示它将寻找起始时间和终止时间然后做一个简单的运算并输出。

下面的代码是用来计算一个矩形的惯性矩：

```
momentOfInertia #codeStream
{
  codeInclude
  #{
    #include "diagTensor.H"
    #};
  code
  #{
    scalar sqrLx = sqr($Lx);
    scalar sqrLy = sqr($Ly);
    scalar sqrLz = sqr($Lz);
    os  <<
    $mass
    *diagTensor(sqrLy + sqrLz, sqrLx + sqrLz, sqrLx + sqrLy)/12.0;
    #};
};
```

4.2.15 条件语句

OpenFOAM 中的读取文件同样支持 if else 判断语句，并且支持 calc 进行简单的数学运算，例如：

```
angle 65;

laplacianSchemes
{
  #if calc "${angle} < 75"
  default Gauss linear corrected;
  #else
  default Gauss linear limited corrected 0.5;
  #endif
}
```

ifEq 支持对文字的对比，如：

```
ddtSchemes
{
  #ifeq ${FOAM_APPLICATION} simpleFoam
  default
  steadyState;
  #else
  default
  Euler;
  #endif
}
```

[5]内嵌计算器的时候会在当前算例目录下生成一个较大的文件包

4.3 全局控制

OpenFOAM 的 controlDict 文件中包含了大量的默认参数。这个 controlDict 文件位于 $FOAM_ETC 文件目录下，而不是算例 system 文件下的 controlDict。这个文件中包含了以下主要信息：

Documentation 用于开启 web 浏览器；

InfoSwitches 控制终端输出的信息；

OptimisationSwitches 用于并行计算通信设置；

DebugSwitches 设置 debug 开关，可参考3.2.6节；

DimensionedConstants 一些基本的物理常数，如玻尔兹曼常数；

DimensionSets 一些基本的单位设置。

4.3.1 覆盖全局参数

DimensionedConstants 中设置的值取决于单位系统，例如 SI 单位制或 USCS 单位制。默认采用 SI 单位制，但用户可以设置成 USCS 单位制。用户可以通过下面的语句进行全局的调整或者针对某个算例进行调整：

```
DimensionedConstants
{
  unitSet  SI;
}
```

用户可以在 etc/controlDict 对其进行全局的设置，但最好还是在单独的算例文件中的 controlDict 设置更好。OpenFOAM 提供了一系列的文件位置可用来你对全局设置文件进行包含。用户可以尝试键入下面的命令来查看：

```
foamEtcFile -list
```

其会展现一些列的文件路径，其中包含一个 $HOME/.OpenFOAM 路径，而/etc 文件则位于最低优先级最后展现。

因此，如果用户打算永久的使用 USCS 单位，可以在 $HOME/.OpenFOAM 路径下创立一个 controlDict 文件并包含下面的内容：

```
DimensionedConstants
{
  unitSet  USCS;
}
```

OpenFOAM 会优先读取 $HOME/.OpenFOAM unitSet 关键词。

如果用户只想为某个算例设置 USCS 单位制，那可以在算例的 system 文件夹下单独包含 controlDict 中的相应设置。

4.4 时间与输入输出

在使用 OpenFOAM 进行运行之前，应该首先对数据的输入输出形式进行设定。OpenFOAM 依据这个数据中心的设定来输入输出数据。由于一般在写数据的时候都是在运行的时候，时间

设定是在所难免的。controlDict 字典文件内包含了数据输入输出的关键信息。下面列举了 controlDict 常见的关键词信息。只有时间控制和 writeInterval 部分是必须存在的，默认值后面附带了 + 号。其它可以省略的关键词没有标注默认值。

```
18  application icoFoam;
19
20  startFrom startTime;
21
22  startTime 0;
23
24  stopAt endTime;
25
26  endTime 0.5;
27
28  deltaT 0.005;
29
30  writeControl timeStep;
31
32  writeInterval 20;
33
34  purgeWrite 0;
35
36  writeFormat ascii;
37
38  writePrecision 6;
39
40  writeCompression off;
41
42  timeFormat general;
43
44  timePrecision 6;
45
46  runTimeModifiable true;
```

4.4.1 时间控制

startFrom 控制模拟的起始时间。

- firstTime[6]: 时间步文件夹中的最开始的时间步文件夹。
- startTime: 依据 startTime 关键词信息开始计算。
- latestTime: 从最新的时间步开始计算。

startTime 如果把 startFrom 设置为 startTime，此处指明计算的开始时间步。

stopAt 控制模拟的结束时间。

- endTime: 依据 endTime 关键词信息结束运算。
- writeNow: 在当前的时间步进行完整的一次运算然后写入数据[7]。
- noWriteNow: 在当前的时间步进行完整的一次运算但是不写入数据。
- nextWrite: 依据 writeControl 设置的时间周期，在下一个时间步写入后停止运算。

endTime 如果把 stopAt 设置为 endTime，endTime 需要指定为结束的时间步。

deltaT 计算的时间步长。

4.4.2 数据写入

writeControl 控制输出。

- timeStep: 每 writeInterval 时间步写一次数据。
- runTime: 每 writeInterval 时间间隔（秒）写一次数据。

[6]在设置中请去除“-”号

[7]否则对于大型算例可能发生计算完成但是数据没有写入的情况

- `adjustableRunTime`: 每 writeInterval 时间间隔（秒）写一次数据，必要的时候调整时间步以使得在 writeInterval 的物理时间点写入数据，通常在自动调整时间步的算例中使用。
- `cpuTime`: 每 writeIntervalCPU 时间（秒）写一次数据。
- `clockTime`: 每 writeInterval 运算时间（秒）写一次数据。

`writeInterval` 和 writeControl 结合使用的一个标量。

`purgeWrite` 一种循环写入数据的方式，其值代表循环时间步数。例如如果我们的起始时间为 5s，每个时间步设定为 1s 一写，purgeWrite 设置为 2，那么在计算完 6s，7s 的数据的时候写入，在计算完 8s，9s 的数据后覆盖写入之前的 6s，7s 文件夹。如果不想使用此功能，可以设置为 purgeWrite 0；对于稳态计算，我们可以设置 purgeWrite 为 1。

`writeFormat` 指定数据格式。

- `ascii`: ASCII 格式，由 writePrecision 控制有效数字位数。
- `binary`: 二进制格式。

`writePrecision` 和前文的 writeFormat 联合使用，默认为 6。

`writeCompression`[8] 压缩形式。

- `uncompressed`: 不压缩。
- `compressed`: gzip 压缩格式。

`timeFormat` 时间步文件夹的名称指定。

- `fixed`: ±m.dddddd，其中 d 由 timePrecision 来控制。
- `scientific`: ±m.dddddd±xx，d 由 timePrecision 来控制。
- `general`: 使用 scientific 格式，默认小数点后有 4 位有效位数，也可通过 timePrecision 来调节。

`timePrecision` 和前文的 timeFormat 联合使用，默认为 6。

`graphFormat` 后处理程序写入的数据格式。

- `raw`: ASCII 原始数据。
- `gnuplot`: gnuplot 专用格式。
- `xmgr`: Grace/xmgr 专用格式。
- `jplot`: jPlot 专用格式。

4.4.3 其他设定

`adjustTimeStep` 设定 OpenFOAM 是否开启可调节时间步功能，可设置为 yes 或者 no。

`maxCo` 最大库郎数，例如 0.5。

`runTimeModifiable` 通过设置为 yes 或者 no，让 OpenFOAM 在每个时间步读取或不读取所修改的字典文件，例如 controlDict。

`libs` 运行时需要加载的一系列的附加库（位于 `$LD_LIBR -ARY_PATH` 路径下），例如 (“libUser1.so”“liUser2.so”)。

`functions` 需要加载的函数，例如 probes 函数，参见具体算例其如何设置 `$FOAM_TOTURIALS`。

4.5 离散格式

system 文件里面的 fvSchemes 用来设置离散格式，凡是求解器里面出现的方程，都需要在这里面设置离散格式。这一节主要阐述如何在 fvSchemes 里设置离散格式。

需要在 fvScheme 里面指定的项有很多，从梯度项到点插值都需要指定。OpenFOAM 开源代码的目的是给用户充分自由的选择权。例如，一般情况下线性插值是非常有效的，但 OpenFOAM 也提供了一系列的非线性插值格式供用户选择。OpenFOAM 用户具有充分的自由选择各种格式的权利体现在导数项的离散上。首先，用户有很多离散方法可以选择，通常使用高斯积分定律进行体积分。高斯积分定律建立在把所有的面值加和的基础之上，这就需要从节点值获得面值，这个过程就是插值。用户有充分的自由来选择怎样来进行插值。值得注意的是，某些插值格式是专门为了某些导数而设计，比如是对流项 $\nabla\cdot$。

fvSchemes 中需要指定的信息主要分为下面的小类：

interpolationSchemes 网格体心至面心插值格式。

snGradSchemes 面法向梯度格式。

gradSchemes 梯度项 ∇ 格式。

divSchemes 对流项 $\nabla\cdot$ 格式。

laplacianSchemes 拉普拉斯项 ∇^2 格式。

timeScheme 时间项的一阶 $\frac{\partial}{\partial t}$、二阶 $\frac{\partial^2}{\partial^2 t}$ 导数格式。

wallDist 有些情况下需要定义壁面距离。

进一步的，表中的关键词也是一个子字典，例如 gradSchemes 包含所有梯度项的格式，例如、grad(p)，代表 $\bigtriangledown$p 的格式。更多的例子请参考下面这个从 fvSchemes 字典文件中抽取的片段：

```
18  ddtSchemes
19  {
20      default                 Euler;
21  }
22
23  gradSchemes
24  {
25      default                 Gauss linear;
26  }
27
28  divSchemes
29  {
30      default                 none;
31
32      div(phi,U)              bounded Gauss linearUpwind grad(U);
33      div(phi,k)              bounded Gauss upwind;
34      div(phi,epsilon)        bounded Gauss upwind;
35      div(phi,R)              bounded Gauss upwind;
36      div(R)                  Gauss linear;
37      div(phi,nuTilda)        bounded Gauss upwind;
38
39      div((nuEff*dev2(T(grad(U))))) Gauss linear;
40  }
41
42  laplacianSchemes
43  {
44      default                 Gauss linear corrected;
45  }
46
47  interpolationSchemes
48  {
49      default                 linear;
50  }
51
52  snGradSchemes
53  {
54      default                 corrected;
55  }
```

上述的范例中包含了6个主要格式关键词子字典的指定。如果…Schemes子字典中的 default 关键词已经被指定,那么子字典中所有的项都会应用这个格式。例如,在 gradScheme 中为 default 指定某个格式，那么所有的 ∇p，∇U 都会使用这个格式。当 default 被指定的时候，就没有必要重复指定子字典的其它信息了。例如 ∇p，∇U 等。然而，如果你继续指定它们的格式，那么它会覆盖默认格式并使用自定义格式。

用户也可以指定 default 为 none（比如上面这个范例中的对流项格式 divSchemes）这种情况下，用户必须强制为每个项指定离散格式。把 default 设置为 none 看起来没有必要且繁琐，因为 default 可以被覆盖。但是，指定 none 的话强制用户对方程的每个需要离散的项进行设定，这会提醒用户哪些项在求解器中有所呈现。

OpenFOAM包含了大量的离散格式，然而只有少部分适用于工程应用。用户可以从OpenFOAM提供的算例中学习如何设置不同的离散格式。例如如果打算学习大涡模拟的离散格式设置，用户则需要查看大涡模拟的格式设置。另外，foamSearch 程序可以用来方便的查找自带算例中使用了哪些离散格式。例如，如果我们想输出 $FOAM_TUTORIALS 目录下所有的 ddtScheme 格式的 default 信息，可以输入：

```
foamSearch $FOAM_TUTORIALS fvSchemes ddtSchemes.default
```

这将会输出下面的信息：

```
default         backward;
default         CrankNicolson 0.9;
default         Euler;
default         localEuler;
default         none;
default         steadyState;
```

4.5.1 时间格式

时间项（$\partial/\partial t$）在子字典 ddtSchemes 下制定。可选的离散格式主要有：

steadyState 稳态，忽略时间导数项。

Euler[9] 瞬态，一阶，有界，隐性。

backward 瞬态，二阶，可能越界，隐性。

CrankNicholson 瞬态，二阶，有界，隐性。

localEuler 局部时间步，一阶，隐性。

需要注意的是 Crank-Nicholson 格式中有一个偏移系数，通过这个系数可以设定格式是否和 Euler 格式混合。ϕ=0 意味着纯的 Crank-Nicholson 格式，ϕ=1 意味着纯欧拉隐性格式。大于0小于1的系数可以提高稳定性，因为在有些情况下纯 CrankNicholson 格式并不稳定。实际工程应用中一般可以选择 ϕ=0.9。用户需要注意的是，对于某一个求解器，如果这个求解器本身是瞬态求解器，就不能用于稳态计算。反之亦然。二阶时间偏导项（$\frac{\partial^2}{\partial^2 t}$）在 d2dt2Schemes 子字典中指定，只有 Euler 可以选择。

4.5.2 梯度格式

gradSchemes 子字典包含梯度项，默认的梯度格式主要为：

```
default         Gauss linear;
```

其中的 Gauss 指定使用高斯积分计算梯度，这要求体心到面心的插值。具体的体心到面心的插值格式通过 linear 来指定，意味着使用中心差分进行插值。

在有些算例中，特别是网格质量很差的情况下，某些梯度项需要进行特殊处理来维持稳定性和有界性，通常这些项是速度梯度项：

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

在某些特殊的情况下，湍流变量也需要特殊设定：

```
grad(k)         cellLimited Gauss linear 1;
grad(epsilon)   cellLimited Gauss linear 1;
```

其中的 cellLimited 表示当使用网格体心的值来计算梯度的时候，梯度的大小受限于体心梯度，其进而可以指定一个参数，例如 1 保证完全的限制并达到有界，0 则表示不应用 cellLimited。其他可选的梯度格式还有：

leastSquares 梯度计算的最小二乘法。

Gauss cubit dnsFoam 特定格式。

4.5.3 散度格式

divSchemes 子字典定义散度项的格式，例如类似这种项的格式：$\nabla\cdot$。注意：$\nabla\cdot(\Gamma\nabla...)$ 不算在内。散度项通常包括对流项 $\nabla\cdot(\mathbf{U}k)$，其中 **U** 表示对流通量，以及 $\nabla(\cdot\nu\nabla\mathbf{U})^{\mathrm{T}}$，其中 **U** 表示扩散通量。

散度项由于其特殊性，一般情况下 divScheme 下的 default 关键词默认为 none。对于非对流项一般选用 Gauss linear 格式如：

```
div(U)    Gauss linear;
```

对流项的格式选用在 CFD 数值上一直是个挑战，因此在选用上需要注意，同时 OpenFOAM 也提供了大量的对流项格式。对流项关键词一般开头为：div(phi, …)，其中 phi 在可压缩流中表示质量通量，在不可压缩流中表示体积通量。例如：div(phi,U) 表示速度的对流、div(phi, e) 表示内能的对流、div(phi, k) 表示湍流动能的对流。用户可以通过 foamSearch 命令查看所有自带算例文件中的 div(phi, U) 的设置：

```
foamSearch $FOAM_TUTORIALS fvSchemes "divSchemes.div(phi,U)"
```

对流格式基于高斯积分，因此需要从网格单元体心处向面心处插值，其中 linear,linearUpwind 就是用于插值的格式。用户还可以选择添加 bounded 关键词，这在后续我们将对其进行介绍。

现在首先忽略关键词以 V 结尾的格式以及很少使用的格式如 Gauss cubic，在算例中插值格式主要有：

linear 二阶、无界；

linearUpwind 二阶、基于迎风、无界（但要比 linear 在数值上更好）、需要指定速度梯度；

LUST 75% 的 linear 附加 25% 的 linearUpwind，需要指定速度梯度；

limitedLinear 在 linear 格式的迎风方向进行限制，需要指定一个系数：1 表示最强的限制器；0 表示 linear 格式；

upwind 一阶、有界、精度较差。

具体的如何使用可以参考下面的范例：

```
div(phi,U)    Gauss linear;
div(phi,U)    Gauss linearUpwind grad(U);
div(phi,U)    Gauss LUST grad(U);
div(phi,U)    Gauss LUST unlimitedGrad(U);
div(phi,U)    Gauss limitedLinear 1;
div(phi,U)    Gauss upwind;
```

关键词带 V 的格式主要用来指定矢量场。不同于传统格式（对每个方向应用于不同的限制器），而是对这些不同的方向应用一个相同的限制。这个方向基于梯度变化最大的方向，因此限制的作用会非常强且非常稳定，但同时牺牲了精准度。具体如何使用这个限制器可以参考下面的范例：

```
div(phi,U)    Gauss limitedLinearV 1;
div(phi,U)    Gauss linearUpwindV grad(U);
```

附加 bounded 关键词的格式主要用于物质导数。例如对于不可压缩流场的 e 的物质导数可以表示为空间导数和对流导数的加和：

$$\frac{\mathrm{D}e}{\mathrm{D}t}=\frac{\partial e}{\partial t}+\mathbf{U}\cdot\nabla e=\frac{\partial e}{\partial t}+\nabla\cdot(\mathbf{U}e)-(\nabla\cdot\mathbf{U})e \tag{4-1}$$

对于不可压缩流体，在收敛情况下 $\nabla\cdot\mathbf{U}=0$，也就是说上述公式的第三项为 0。但是在收敛之前，$\nabla\cdot\mathbf{U}\neq 0$。在有些情况下，尤其是在稳态的情况下，建议包含这一项因为它可以提高收敛性和稳定性。附加 bounded 的格式则自动的在对流项中包含了这一项。例如在 simpleFoam 的算例中的 fvScheme 中，可以看到下面的范例：

```
div(phi,U)        bounded Gauss limitedLinearV 1;
div(phi,U)        bounded Gauss linearUpwindV grad(U);
```

标量场的对流格式和速度场的对流格式类似。但是对于标量场，有界性要比精准性要求更高。例如，用户可以这样查看内能的对流格式：

```
foamSearch $FOAM_TUTORIALS fvSchemes "divSchemes.div(phi,e)"

div(phi,e)        bounded Gauss upwind;
div(phi,e)        Gauss limitedLinear 1;
div(phi,e)        Gauss LUST grad(e);
div(phi,e)        Gauss upwind;
div(phi,e)        Gauss vanLeer;
```

如果用户对比速度场的对流格式，就会发现，标量场的对流格式没有任何一个设置为 linear[10]或者 linearUpwind。相反，标量场通常使用 limitedLinear 或者 upwind 格式，有时候还会使用 vanLeer 格式（其为另一个附加限制器的格式，其限制性没有 limitedLinear 强）。

[10]linear 格式在纯对流的情况下是无条件不稳定的

有些情况下有些标量需要保证在 0-1 之间严格有界，例如层流火焰回归变量 b。如果用户查找其对流格式会发现在算例中我们使用了下面的格式：

```
div(phiSt,b)        Gauss limitedLinear01 1;
```

其通过指定 1 调用最强的 limitedLinear 格式，同时添加 01 将变量限制在 01 之间。

OpenFOAM 中还可以通过 multivariateSelection 来对多个项进行指定并应用同样的限制器。其对所有的变量应用其中最强的限制器。例如在求解组分输运的质量传递方程的时候，可以调用这个格式设置。这种方法的设置可以很方便的保证所有组分的格式一致性。在 $FOAM_TUTORIALS/combustion/fireFoam/les 中的 smallPoolFire3D 算例中，用户可以发现下面的设置：

```
div(phi,Yi_h)   Gauss multivariateSelection
{
  O2 limitedLinear01  1;
  CH4 limitedLinear01 1;
  N2 limitedLinear01  1;
  H2O limitedLinear01 1;
  CO2 limitedLinear01 1;
  h limitedLinear     1;
}
```

4.5.4 面法向梯度格式

我们在讨论 laplacianScheme 之前讨论 snGradSchemes 是因为在通过高斯积分计算拉普拉斯项的时候，需要调用面法向梯度计算。面法向梯度项定义在网格单元的面网格上，其为某个量在两个相连的网格中心单元处的值的面梯度分量。

搜索 snGradSchemes 的 default 关键词会输出下面的信息：

```
default         corrected;
default         limited corrected 0.33;
default         limited corrected 0.5;
default         orthogonal;
default         uncorrected;
```

面法向梯度计算的基本法则为将面毗连的网格单元的体心的值进行相减并除以距离。如果连接两个网格单元体心的矢量和面法向是平行的那么这种方法计算的结果为二阶精度的。OpenFOAM 中的 orthogonal 即对应的这种计算方法。

在网格生成中，网格正交性需要保证网格线依附于笛卡尔坐标轴，这在一般情况下是很难达到的。因此，为了提高计算准确性，在计算面法向梯度的时候可以添加一个非正交修正，即 corrected 关键词。网格非正交性越强，对修正的要求越高。

非正交性一般通过连接网格单元体心的矢量和面法向的夹角来表示，如果非正交角度大于 70 度，那么显性正交修正会非常大并引起数值稳定性问题。在 OpenFOAM 中，这个问题可以通过添加一个限定性因子来增加稳定性，其数值应该大于 0 小于 1：

$$\varphi = \begin{cases} 0 \\ 0.333 \\ 0.5 \\ 1 \end{cases} \tag{4-2}$$

一般情况下，φ 的值选择为 0.33 或者 0.5。如果设置为 0.33，则会有较好的稳定性。如果设置为 0.5，则精度比较高。

corrected 格式对隐性离散的正交计算部分以及显性离散的非正交部分应用了亚松弛因子（cosα）增加对角占优特性。uncorrected 格式等同于没有正交修正的 corrected 格式，也类似于附加亚松弛因子（cosα）的 orthogonal 格式。uncorrected 以及 orthogonal 只适用于正交网格或者非正交角度非常低的网格（如小于 5 度）。对于小于 70 度的非正交角度，推荐选用 corrected 格式，也可依据情况选择是否添加 limited 限制器。如果非正交角度大于 80 度，那么收敛性会受到非常大的影响。

4.5.5 拉普拉斯格式

laplacianSchemes 子字典包含了拉普拉斯项，我们来讨论一下流体力学中的这个拉普拉斯项：$\nabla\cdot(\nu\nabla\mathbf{U})$，其为动量方程中的扩散项，对应 laplacianScheme 中的 laplacian(nu,U)。目前只有 Gauss 格式可选，并且需要选择为粘度 ϑ 选择插值格式，以及面法向梯度格式 $\nabla\mathbf{U}$ 的格式，这些项的格式这样指定：

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

用户可以通过下面的命令，在 $FOAM_TUTORIALS 中查找 laplacianSchemes 的 default 关键词，以此来查看 OpenFOAM 设置的默认格式：

```
foamSearch $FOAM_TUTORIALS fvSchemes laplacianSchemes.default
```

它会输出下面的信息：

```
default         Gauss linear corrected;
default         Gauss linear limited corrected 0.33;
default         Gauss linear limited corrected 0.5;
default         Gauss linear orthogonal;
default         Gauss linear uncorrected;
```

在大多数情况下，扩散系数的插值我们选用 linear 格式，对于面法向梯度格式，具体的是否加入非正交修正可以参考4.5.4节。

4.5.6 插值格式

interpolationSchemes 主要用于定义从网格体心到网格面心的插值格式，它大量的用于评估面速度（也即通量）。OpenFOAM 中包含了大量的可用的插值格式，如果用户搜索的话会发现大部分算例都是使用 linear 插值格式。只有一些非常特殊的例子下我们采用 cubic 插值格式，如 DNS、应力分析算例等。

4.6 求解和算法控制

方程求解器、残差、算法在 system 文件夹下的 fvSolution 文件中控制。下面是 icoFoam 求解器所需的一个典型的 fvSolution 设置：

```
18  solvers
19  {
20      p
21      {
22          solver                  PCG;
23          preconditioner          DIC;
24          tolerance               1e-06;
25          relTol                  0;
26      }
27
```

```
28      pFinal
29      {
30          $p;
31          relTol                  0;
32      }
33
34      U
35      {
36          solver                  smoothSolver;
37          smoother                symGaussSeidel;
38          tolerance               1e-05;
39          relTol                  0;
40      }
41  }
42
43  PISO
44  {
45      nCorrectors                 2;
46      nNonOrthogonalCorrectors    0;
47      pRefCell                    0;
48      pRefValue                   0;
49  }
```

fvSolution 包含一系列子字典来为求解器的顺利运行提供必要信息。大部分求解器都是参照这个图中来进行设置。这些子字典中包括 solvers，relaxationFactors，PISO、SIMPLE、PIMPLE 等，在后面的章节中我们都有所描述。

4.6.1 矩阵求解器

这个例子中第一个子字典（所有求解器的第一项均为）是 solvers。它指定求解离散项需要的矩阵求解器。这里矩阵求解器和我们说的流体力学方程求解器是不同的。矩阵求解器是指用来求解大型矩阵的程序。相反的是，流体力学方程求解器是求解一系列偏微分方程组的，包含一系列算法的程序。在之后的讨论中，我们用求解器来表示矩阵求解器，希望读者不会产生混乱。

求解器中出现的 word 关键词代表着方程组中的每一个变量。例如，icoFoam 求解速度和压力场。因此存在 U 和 p。关键词之后跟随的是字典文件，里面有求解器类型，以及求解器使用的参数。在 solvers 关键词下我们有不同的场，在每个场内的 solver 关键词下指定求解器。表 4.12 列举了可使用的求解器。一些参数比如 tolerance, relTol, preconditioner, 将在下面进行描述。

PCG/PBiCG/PBiCGStab（稳定化）预条件（双）共轭求解器，其中，PCG 可用于对称矩阵，PBiCG/PBiCGStab 可用于非对称矩阵。

smoothSolver 光顺求解器。

GAMG 多重网格求解器。

diagonal 对角矩阵求解器。

矩阵求解器分为对称矩阵求解器和非对称矩阵求解器。矩阵的对称性和求解的方程有关，例如时间导数以及拉普拉斯项的离散为对称矩阵，对流项的引入会导致矩阵非对称。用户应该会知道矩阵类型，如果用户使用不合适的矩阵求解器 OpenFOAM 会报错：

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

4.6.1.1 求解残差

在有限体积法中，如果采用分离式解耦求解，其离散后的矩阵为稀疏的，大部分的元素为0。稀疏矩阵求解采用迭代的方式来处理。它们建立在把残差降为最小的基础上。残差是在求解矩阵的时候产生的，这个残差越小，结果越精准。准确来讲，残差就是把当前的解输入到方程中，然后取方程左右两边差的绝对值。同时需要对残差进行统一处理以保证残差规模和待解问题的规模无关。

在求解某个场方程之前，初始残差凭当前时间步的值来评估。在每次迭代之后，残差再次评估，在达到下面的条件之后，求解器停止计算。

- 残差降到求解器设定的误差 `tolerance` 之后；
- 当前残差和初始残差的比降到求解器设定的相对残差比之后，`relTol`；
- 迭代数超过最大值，`maxIter`。

求解器误差表示当残差小于这个值的时候，我们认为计算结果已经足够精准的时候和真实解对比产生的误差。求解器的相对残差比限制了从初始场到最终结果的步进。在瞬态计算中，一般我们设定求解器相对残差比为 0，强制求解器在每个时间步内收敛。`tolerance` 以及 `relTol` 在所有的求解器中必须要指定；`maxIter` 可以不进行设定，其默认值为 1000。

在一个时间步中，方程经常被求解多次。例如当使用 `PISO` 算法的时候，`nCorrectors` 指定压力求解的次数，详见4.5.3节。如果这样的话，矩阵求解器通常需要为最后一次的求解设定不同的参数，其通过 `Final` 参数来指定。例如，对于2.1节中的 `cavity` 算例，压力求解的设置如下：

```
p
{
    solver          PCG;
    preconditioner  DIC;
    tolerance       1e-06;
    relTol          0.05;
}
pFinal
{
    $p;
    relTol          0;
}
```

如果求解器设置每个时间步求解压力 4 次，那么前三次求解的时候则读取 p 的设置，且每次的相对残差为 0.05，这样的话每次求解压力方程的时候所需要的计算资源较低。只有在第 4 次求解方程的时候，它才读取 `Final` 里面的设置，即相对残差（`relTol`）为 0。这种情况下完全满足精度要求，但是所需要的计算资源较高。

4.6.1.2 预条件共轭梯度求解器

在共轭梯度求解器中，可以对矩阵进行预条件，它在求解器字典中通过 `preconditioner` 关键词指定。OpenFOAM 中大量的算例采用了 `DIC/DILU` 预条件。下面是 OpenFOAM 中可选的预条件：

`DIC` 不完全的对角 Cholesky 预条件（对称矩阵）。

`FDIC` 不完全的快速对角 Cholesky 预条件（带缓存的 DIC）。

`DILU` 对角不完全 LU 预条件（非对称）。

`diagonal` 对角预条件。

GAMG 多重网格预条件。

none 不适用预条件。

4.6.1.3 光顺器

使用光顺器的求解器需要指定关键词：smoother。通常我们选用 symGaussSeidel 或 GaussSeidel 光顺器[11]。下面列举了可选的光顺器：

GaussSeidel 高斯赛德尔光顺器。

symGaussSeidel 对称高斯赛德尔光顺器。

DIC/DILU 不完全的对角 Cholesky 光顺（对称矩阵），不完全 LU 光顺器（非对称矩阵）。

DICGaussSeidel 不完全的对角 Cholesky-高斯赛德尔光顺器（对称矩阵）。

其中，在重新计算残差之前需要通过为 nSweeps 关键词来指定 sweep 数。如果不设置的话，其默认值为 1。

4.6.1.4 多重网格求解器

多重网格求解器（GAMG）[12]的概念是：在粗网格上快速生成一个解，然后使用这个解作为初始值并把这个解映射到细网格上，然后在细网格上获得准确解。如果在粗糙网格计算消耗的时间比网格精细化和映射场数据少时，GAMG 比标准方法要快。实际上，GAMG 求解首先需要读取用户指定的网格然后进行糙网格或细网格的过渡进程。用户只需要定义 nCoarsestCells 关键字即可通过糙网格级别来指定最糙的网格大小。

不同网格层的融合（agglomeration）方法通过 agglomeration 关键词来指定。我们建议用户采用 faceAreaPair 方法。但是用户也可以选择 MGridGen 方法，这个方法需要加载一个 MGridGen 库：

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

进一步的设置如下（大部分算例中都是采用的默认设置）：

cacheAgglomeration 指定 agglomeration 策略开关；

nCellsInCoarsestLevel 指定最糙的网格糙化等级；

directSolveCoarset 在最糙的网格上直接求解；

mergeLevel 糙化/细化等级，默认为 1，也最安全。但是对于简单的网格，如果设置 mergeLevels 为 2 的话，求解速度会增加。

有关 smoother 的指定可以通过4.6.1.3节中的方法来设置。sweeps 的数量通过下面的关键词来设置：

nPreSweeps 向前糙化网格的 sweep 数，默认为 0；

[11] 一般情况下 GaussSeidel 是最可靠的选择，但对于有些不好的矩阵，DIC 的收敛性可能更好。
[12] 多信息请参考本链接

preSweepLevelMultiplier 每次向前糙化网格的 sweep 数，默认为 1；

maxPreSweeps 向前糙化网格的最大 sweep 数，默认为 4；

nPostSweeps 向后细化网格的 sweep 数，默认为 2；

postSweepsLevelMultiplier 每次向后细化网格的 sweep 数，默认为 1；

maxPostSweeps 向后细化网格的最大 sweep 数，默认为 4；

nFinestSweeps 最终细化网格的 sweep 数，默认为 2。

4.6.2 亚松弛

fvSolution 中的另一个子字典是 relaxationFactors，这个关键字主要用来控制亚松弛（提高求解稳定性，特别是稳态问题）。亚松弛技术就是通过限制每次变量迭代后改变的大小来实现。其要么通过在迭代之前修改矩阵系数和源项来实现，要么通过直接修改求解后的场来实现。松弛因子 α 小于 1 大于 0，它用来调节松弛量。例如：

- 不指定 α，不进行松弛操作；
- $\alpha = 1$，松弛系数为 1；
- α 降低，松弛增加；
- $\alpha = 0$ 表示完全的松弛，每次的解不变。

最好的是 α：它足够小以确保稳定计算，并且足够大确保迭代迅速有效。例如，如果 $\alpha = 0.9$，在某些情况下就已经可以保证稳定性了。如果再选取比较低的数值，比如 0.2，这可能会导致降低迭代效率。

在 OpenFOAM 中可以对场进行松弛，也可以对方程组进行松弛。如果对场进行松弛，那么相应的场应在 field 下指定，如果对方程进行松弛，那么需要在 equations 下面进行指定。下面是 simpleFoam 下算例的一个设置范例，其为不可压缩稳态流动的典型设置。

```
55  relaxationFactors
56  {
57      fields
58      {
59          p 0.3;
60      }
61      equations
62      {
63          U 0.7;
64          "(k|omega|epsilon).*" 0.7;
65      }
66  }
```

对于 pimpleFoam 下的瞬态不可压缩流求解器，只需要调用亚松弛保证矩阵的对角占优即可，例如：

```
61  relaxationFactors
62  {
63      equations
64      {
65          ".*"                1;
66      }
67  }
```

4.6.3 PISO、SIMPLE 及 PIMPLE

大多数 OpenFOAM 的求解器使用 PISO 或者 SIMPLE 算法，或者二者的结合体 PIMPLE 算法。这些算法采用迭代的方式来求解速度和压力场。PISO 以及 PIMPLE 算法用来处理非稳态问题，SIMPLE 算法用来处理稳态问题。

这些算法的基本特点都是在一个求解时间步或者迭代步内，求解压力方程来保证质量守恒，然后附加现行的速度修正来满足动量方程。可选择的，用户可以在每个时间步内或者迭代步内首先求解一次动量方程，这也即动量预测步。

所有的算法都求解相同的控制方程（有可能是不同的形式），他们的区别主要在于他们如何在方程组之间进行循环的。循环通过下面的参数进行控制，他们位于 fvSolution 字典文件中相对应的关键词（如 SIMPLE，PISO 或者 PIMPLE）下：

nCorrectors 其为 PISO 以及 PIMPLE 调用的关键词，用来设置压力方程以及速度修正方程的迭代步数，一般设置为 2 或者 3；

nNonOrthogonalCorrectors 所有的算法均需要指定这个参数，其用于更新压力方程中显性非正交修正项 $\nabla\cdot((1/A)\nabla \mathrm{p})$ 的求解次数（参见4.5.4节），一般设置为 0（稳态）或 1；

nOuterCorrectors PIMPLE 算法的关键词。其用于指定 PIMPLE 算法中整个外循环的迭代次数，其必须大于等于 1，如果设置为 1，PIMPLE 则变为 PISO 算法；

momentumPredictor 动量预测求解开关，对于多相流以及低雷诺数一般设置为 off。

4.6.4 参考压力

在一个封闭的不可压系统内部，压力是一个相对值。压差（而非绝对压力）才是主要的。在这些例子中，求解器给 pRefCell 设定一个参考值 pRefValue。p 即为压力变量。有些算例中压力是 p_rgh，那么参考压力和参考网格就是 p_rghRefValue 以及 p_rghRefCell。这些信息存储在 PISO/SIMPLE 的子字典中，有些特定的算例需要指定这些信息并供给求解器使用。如果省略了这些信息，求解器将不会运行，但是会提醒用户问题所在。

4.6.5 其它参数

OpenFOAM 标准求解器中 fvSolution 文件包含的信息，本章均已罗列。然而，fvSolution 文件还可以包含任何其它用于控制求解器、算法或其它东西的相关参数。对于一个求解器，用户应该去查看它的源代码看看哪些参数是必须存在的。任何参数或者子字典的未定义，求解过程都将会终止并输出错误警告。用户应该把没有包含的信息添加进去。

4.7 算例管理工具

本节我们讨论可用于用户管理算例的一些工具[13]。

4.7.1 文件管理脚本

下面的脚本有利于管理用户的算例。

[13]对于远程操作、不方便打开 GUI 的情况以及自动化批量运行算例的时候比较有用

foamListTimes 列出算例的所有时间步文件，默认省略 0 文件夹。命令参数-rm 可以删除所有的除了 0 文件夹之外的时间步，这个命令主要用于清理算例：

```
foamListTimes -rm
```

foamCloneCase 创建一个新算例并从现有算例拷贝 0，system 及 constant 文件夹（其中 oldCase 表示旧算例的名称）：

```
foamCloneCase oldCase newCase
```

foamCleanPolyMesh 删除网格文件，在利用 snappyHexMesh 重新生成网格的时候有必要执行此脚本[14]；

4.7.2 foamDictionary 工具

foamDictionary 工具提供了对算例文件进行读写、编辑、添加的功能。命令应该在 OpenFOAM 算例下执行。例如下面的命令对算例的 fvSchemes 进行操作：

```
foamDictionary system/fvSchemes
```

如果不附加任何参数，其会罗列所有的关键词信息，例如下面即为 simpleFoam/pitzDaily 算例的 fvSchemes 信息：

```
{
    FoamFile
    {
        version                 2;
        format                  ascii;
        class                   dictionary;
        location                "system";
        object                  fvSchemes;
    }
    ddtSchemes
    {
        default                 steadyState;
    }
    gradSchemes
    {
        default                 Gauss linear;
    }
    divSchemes
    {
        default                 none;
        div(phi,U)              bounded Gauss linearUpwind grad(U);
        div(phi,k)              bounded Gauss limitedLinear 1;
        div(phi,epsilon)        bounded Gauss limitedLinear 1;
        div(phi,omega)          bounded Gauss limitedLinear 1;
        div(phi,v2)             bounded Gauss limitedLinear 1;
        div((nuEff*dev2(T(grad(U))))) Gauss linear;
        div(nonlinearStress)    Gauss linear;
    }
    laplacianSchemes
    {
        default                 Gauss linear corrected;
    }
    interpolationSchemes
    {
        default                 linear;
    }
    snGradSchemes
    {
        default                 corrected;
    }
    wallDist
    {
        method                  meshWave;
    }
}
```

如果附加命令参数 -entry，用户可以针对某些特别的子字典进行输出（如 divSchemes）：

[14]若不删除，snappyHexMesh 将会报错

```
foamDictionary -entry divSchemes system/fvSchemes
```

下面是输出的信息：

```
divSchemes
{
    default                 none;
    div(phi,U)              bounded Gauss linearUpwind grad(U);
    div(phi,k)              bounded Gauss limitedLinear 1;
    div(phi,epsilon)        bounded Gauss limitedLinear 1;
    div(phi,omega)          bounded Gauss limitedLinear 1;
    div(phi,v2)             bounded Gauss limitedLinear 1;
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
    div(nonlinearStress)    Gauss linear;
}
```

用户也可以使用下面的命令来输出某个子字典下某个特别的关键词：

```
foamDictionary -entry "divSchemes.div(phi,U)" system/fvSchemes
```

这一行命令输出 fvSchemes 文件下 divSchemes 子字典中 div(phi,U) 的关键词信息：

```
div(phi,U)              bounded Gauss linearUpwind grad(U);
```

如果进一步的附加命令参数 -value，则只输出指定的关键词：

```
foamDictionary -entry "divSchemes.div(phi,U)" -value system/fvSchemes
```

其输出：

```
bounded Gauss linearUpwind grad(U);
```

命令参数 -set 可用于设置某关键词的信息。如果用户打算修改 div(phi,U) 为迎风格式，那么可以运行下面的命令：

```
foamDictionary -entry "divSchemes.div(phi,U)" \
-set "bounded Gauss upwind "system/fvSchemes
```

命令参数 -add 可用于添加某些信息。如果用户打算在 divSchemes 下添加 turbulence 关键词并使用迎风格式，那么可以运行下面的命令：

```
foamDictionary -entry "divSchemes.turbulence" \
-add "bounded Gauss upwind "system/fvSchemes
```

在4.5节中，我们演示了如何使用 foamSearch，如果附加命令参数 -c 则会统计每个关键词信息出现的数量，例如用户可以运行下面的命令：

```
foamSearch -c $FOAM_TUTORIALS fvSolution solvers.p.solver
```

其寻找所有自带算例中 fvSolution 文件下 solvers 子字典中的 p 子字典中的关键词 solver 信息：

```
59 solver        GAMG;
3 solver        PBiCG;
18 solver        PCG;
5 solver        smoothSolver;
```

4.7.3 foamGet 工具

foamGet 可以快速的将一些设置拷贝到当前的算例中。用户必须切换到当前的算例文件中，或者附加-case 命令参数。具体的使用可参见下面的范例，在 pitzDaily 算例中，首先进行如下命令切换到算例文件下：

```
run
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily .
cd pitzDaily
```

然后可以通过键入

```
cd pitzDaily
blockMesh
```

来生成网格。参考第7.2节，用户可以通过命令获取已经设置的后处理程序，如：

```
postprocess -list
```

接下来，如果用户打算监控流率，需要通过 flowRatePatch 来进行设置。用户可以通过 foamGet 来实现目的：

```
foamGet flowRatePatch
```

运行命令后，显影的 flowRatePatch 和 flowRatePatch.cfg 文件会自动出现在算例文件下。用户需要将 flowRatePatch 文件拷贝到 system 文件夹下。为了获取 outlet 出口的流量，需要在 flowRatePatch 文件中进行下述设置：

```
name outlet;
```

最后一步，用户需要将 flowRatePatch 设置包含在 controlDict 文件中：

```
functions
{
    #includeFunc streamlines
    #includeFunc flowRatePatch
}
```

4.7.4 foamInfo 脚本

foamInfo 可用于输出一些求解器、程序及边界条件等的信息。例如，下属命令将输出 simpleFoam 求解器的相关信息：

```
foamInfo simpleFoam
```

下面的命令将输出 flowRateInletVelocity 边界条件的信息：

```
foamInfo flowRateInletVelocity
```

输出主要包含源文件的位置，相关描述，使用信息以及一些范例。

参考第4.7.3节，在 pitzDaily 算例中，可以指定体积流量进口。对于均一速度 10m/s 的进口速度，其相当于 $2.54e-4 m^3/s$ 的体积流量，因此在 0 文件夹下的 U 文件中应设置：

```
inlet
{
    type                flowRateInletVelocity;
    volumetricFlowRate  2.54e-4;
    extrapolateProfile  yes;
    value               uniform (0 0 0);
}
```

然后运行 simpleFoam 求解器即可。算例约在 280 左右收敛，并可通过 Paraview 进行后处理。用户可以发现进口表现了一个非均一的速度场，这是因为 extrapolateProfile 关键词被开启。出口的流量被写入在 postProcessing/flowRatePatch/0 文件夹下的 surfaceFieldValue.dat 文件中，出口的流量最终收敛于进口流量。

第 5 章　张量

这一章我们介绍张量和张量运算，以及它们在本手册中是如何进行表达的。然后我们解释张量和张量运算如何在 OpenFOAM 中的实现。

5.1　坐标系统

OpenFOAM 主要用来解决连续介质问题，例如：固体应力分析、液体气体流动问题以及物理材料的变形问题。因此，OpenFOAM 在三维空间和时间中处理用张量来进行描述的物理问题。参见图5-1，OpenFOAM 使用的坐标系统符合右手规则。建立于右手规则的坐标系符合下述特征：三个坐标轴起源于 O 点，他们的名字为 O_x、O_y、O_z，当从 O_z 向上观测的时候（O 点离观测者较近），从 O_x 上某点指向 O_y 上某点的圆弧为顺时针方向。

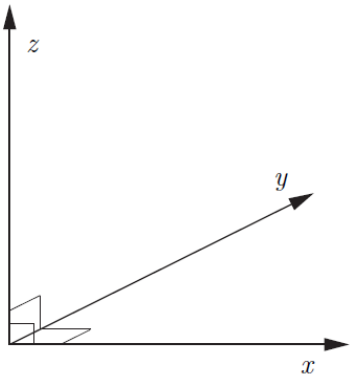


图 5-1: 右手规则坐标轴。

5.2　张量

张量用来描述从属某一空间的实体，其遵循一定的数学法则。简要来说，张量就是和一系列单位矢量相关的一系列的值。每个张量具有如下特征：

维度矢量 d　OpenFOAM 中的维度用 **d** 来表示；

阶数（秩）r　其为一个整数，表示分量的个数，例如我们可以这样表示：$\mathbf{d}^r$；在 OpenFOAM 中默认设置为 3 维的（$\mathbf{d}^r = 3$），并提供 0 到 3 阶张量作为标准张量。对张量进行扩展也非常容易。我们熟知 0 阶张量和 1 阶张量代表标量和矢量，2 阶张量和 3 阶张量有些读者可能会感到有些陌生。为了文章的完整性我们把他们都进行了描述；

0 阶标量　我们用 0 阶张量表示任何一个可以用单个数值表示的物理量，例如：质量 m，体积 V，压力 p，粘度 μ；

1 阶矢量　其用来表示具有大小和方向的物理量。1 阶矢量的形式为：$\mathbf{a} = (a_1, a_2, a_3)$，其和笛卡尔坐

标系的 x,y,z 相对应。有时我们用索引符 i 表示这个张量的分量，例如 a_i，其中 $i=1,2,3$；

2 阶张量 我们用 **T** 来表示，**T** 具有 9 个元素，参见如下：

$$\mathbf{T}=\begin{bmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{bmatrix} \tag{5-1}$$

由于其为 2 阶张量，因此分量 T_{ij} 使用两个索引符来表示。当 $i=j$ 的时候，T_{ij} 为对角元素。那些 $i\neq j$ 的元素为非对角元素。**T** 的转置矩阵如下：

$$\mathbf{T}=\begin{bmatrix} T_{11} & T_{21} & T_{31} \\ T_{12} & T_{22} & T_{32} \\ T_{13} & T_{23} & T_{33} \end{bmatrix} \tag{5-2}$$

需要注意的是，我们通常将二阶张量称为张量，因为再往上的高阶张量非常罕见；

对称 2 阶张量 对称张量指得是非对角元素关于对角元素是对称的。例如 $T_{ij}=T_{ji}$，在这种情况下，只有 6 个独立元素（$T_{12}=T_{21}$，$T_{13}=T_{31}$，$T_{23}=T_{32}$）。在 OpenFOAM 中我们区分对称张量和非对称张量来节省内存。因为对称张量只需要存储 6 个元素，非对称张量需要存储 9 个元素。大多数连续介质问题中的张量为对称张量；

3 阶张量 具有 27 个元素，其分量用 P_{ijk} 来表示，由于其太复杂在这里我们不表示；

对称 3 阶张量 OpenFOAM 将其定义为下述元素相等的 3 阶张量：$P_{ijk}=P_{ikj}=P_{jik}=P_{jki}=P_{kij}=P_{kji}$，因此它具有 10 个独立分量。更确切来说，3 个相同的矢量进行外积即可得到对称 3 阶张量，我们将在第5.3.4节中介绍外积；

5.2.1 张量表示法

本书是一本关于连续介质计算力学的书籍，主要处理 3 维空间和时间的复杂偏微分方程组。在开始后续章节之前我们很有必要了解一下张量的表示方法以防止在后续章节发生混乱。为了很好的理解方程组，我们希望读者接受这样一个概念：张量是一个整体而不是一系列标量的组合。另外，任何张量操作都是在这个整体上进行操作而不是对某个元素进行操作。

在本书下面的章节中，所有的张量（除了标量）我们都是用粗体，**a**。我们使用单一字符来表示张量，这更加的表明张量本身是一个实体，并且这样表示张量非常简洁。这种表达方式潜在的缺点就是张量的阶数并没有有效的表达。很明显，张量的阶数不能为 0。实际上，由于我们很清楚的了解各个张量代表的物理量的实际意义，缺少张量阶数的这种表达方式也不会对我们造成太大的麻烦。例如我们知道速度 **U** 的阶数为 1。

另外一个关于张量操作符的基本概念是张量和坐标系统的无关性。例如矢量 **a** 不管我们从哪个角度来观测它都是矢量 **a**。这意味着张量和坐标系统无关。然而其他符号，例如 a_i，它则和坐标系统有关，因为它表示的是张量的某个分量。这样的结果就是张量由一系列和坐标系统有关的元素值来表示，这些元素值依赖于坐标系统。

正如5.2节所说，本书将使用张量标识法对张量进行标识，它的主要目的是为了将张量运算拓展到其中的每个分量上。当使用张量标识法表示张量的时候，我们定义，如果在一项中某个索引符出现了两次，则这一项表示元素（例如第 1、2、3 元素）的相加。例如：

$$a_i b_i=\sum_{i=1}^{3} a_i b_i=a_1 b_1+a_2 b_2+a_3 b_3 \tag{5-3}$$

5.3 张量运算

在这一章节我们描述 OpenFOAM 可用的张量运算。我们首先来看一下最常见的张量操作：加、减、乘、除。加法和减法只能同阶操作。他们的操作即为将张量对应的元素分别进行加减法。例如，矢量 **a** 和矢量 **b** 的减法即为：

$$\mathbf{a}-\mathbf{b}=a_i-b_i=(a_1-b_1,a_2-b_2,a_3-b_3) \tag{5-4}$$

矢量 **a** 和标量 s 的乘法也是可交换的，即为将张量的每个元素都乘以标量，例如：

$$s\mathbf{a}=sa_i=(sa_1,sa_2,sa_3) \tag{5-5}$$

在做矢量 **a** 和标量 s 的除法的时候，标量 s 必须为分母，例如：

$$\mathbf{a}/s=a_i/s=(a_1/s,a_2/s,a_3/s) \tag{5-6}$$

依据上述基本定义，在接下来章节中，我们讨论一阶张量的更为复杂的乘积操作。

5.3.1 内积

- 矢量 **a** 和矢量 **b** 的内积是可交换的，并且结果为一个标量 s：

$$\mathrm{s}=a_ib_i=a_1b_1+a_2b_2+a_3b_3 \tag{5-7}$$

- 张量 **T** 和矢量 **a** 的内积结果为矢量 **b**，我们用列向量来表示 $\mathbf{b}=\mathbf{T}\cdot\mathbf{a}$：

$$b_i=T_{ij}a_j=\begin{pmatrix}T_{11}a_1+T_{12}a_2+T_{13}a_3\\T_{21}a_1+T_{22}a_2+T_{23}a_3\\T_{31}a_1+T_{32}a_2+T_{33}a_3\end{pmatrix} \tag{5-8}$$

如果 **T** 是非对称张量，那么这个内积操作是非对称的，即 $\mathbf{b}=\mathbf{a}\cdot\mathbf{T}=\mathbf{T}^{\mathrm{T}}\cdot\mathbf{a}$：

$$b_i=a_jT_{ji}=\begin{pmatrix}T_{11}a_1+T_{21}a_2+T_{31}a_3\\T_{12}a_1+T_{22}a_2+T_{32}a_3\\T_{13}a_1+T_{23}a_2+T_{33}a_3\end{pmatrix} \tag{5-9}$$

- 两个张量 **T** 和 **S** 的内积也是一个张量，即 $\mathbf{P}=\mathbf{T}\cdot\mathbf{S}$，**P** 元素的值这样计算：

$$\mathbf{P}_{ij}=\mathbf{T}_{ik}\mathbf{S}_{kj} \tag{5-10}$$

并且这种运算是不可交换的。

- 矢量 **a** 和三阶张量 **P** 的内积结果为一个二阶张量 **T**，$\mathbf{T}=\mathbf{a}\cdot\mathbf{P}$，**T** 的元素为：

$$\mathbf{T}_{ij}=\mathbf{a}_k\mathbf{P}_{kij} \tag{5-11}$$

同样，这个运算也是不可交换的，因此

$$\mathbf{T}_{ij}=\mathbf{P}_{ijk}\mathbf{a}_k\neq\mathbf{a}_k\mathbf{P}_{kij} \tag{5-12}$$

- 二阶张量 **T** 和三阶张量 **P** 的内积为三阶张量 **Q**，$\mathbf{Q}=\mathbf{T}\cdot\mathbf{P}$，**Q** 的元素为：

$$\mathbf{Q}_{ijk}=\mathbf{T}_{ij}\mathbf{P}_{ljk} \tag{5-13}$$

同样，这个运算也是不可交换的，因此

$$\mathbf{Q}_{ijk}=\mathbf{P}_{ijl}\mathbf{T}_{lk}\neq\mathbf{T}_{ij}\mathbf{P}_{ljk} \tag{5-14}$$

5.3.2 双内积

两个二阶张量 **T** 和 **S** 的双内积结果为一个标量 s：

$$\begin{aligned}s=\mathbf{T}:\mathbf{S}=T_{ij}S_{ij}=T_{11}S_{11}+T_{12}S_{12}+T_{13}S_{13}+\\T_{21}S_{21}+T_{22}S_{22}+T_{23}S_{23}\\T_{31}S_{31}+T_{32}S_{32}+T_{33}S_{33}\end{aligned} \tag{5-15}$$

二阶张量 **T** 和三阶张量 **P** 的双内积结果为一个矢量 **a**：

$$a_i=T_{jk}P_{jki} \tag{5-16}$$

这个运算是不可交换的，因此

$$a_i=\mathbf{a}:\mathbf{P}=P_{ijk}T_{jk} \tag{5-17}$$

5.3.3 三内积

两个三阶张量 **P** 和 **Q** 的三内积结果为一个标量 s：

$$s=\mathbf{P}\vdots\mathbf{Q}=P_{ijk}Q_{ijk} \tag{5-18}$$

5.3.4 外积

张量以及矢量的外积这样定义：

- 两个矢量 **a** 和 **b** 的外积是不可交换的，结果为张量 $\mathbf{T}=\mathbf{ab}=(\mathbf{ba})^{\mathrm{T}}$，其这样计算：

$$T_{ij}=a_ib_j=\begin{pmatrix}a_1b_1 & a_1b_2 & a_1b_3\\ a_2b_1 & a_2b_2 & a_2b_3\\ a_3b_1 & a_3b_2 & a_3b_3\end{pmatrix} \tag{5-19}$$

- 一个矢量 **a** 和一个张量 **T** 的外积为一个三阶张量 $\mathbf{P}=\mathbf{aT}$，其这样计算：

$$P_{ijk}=a_iT_{jk} \tag{5-20}$$

同样这个运算为不可交换的，因此 $\mathbf{P}=\mathbf{Ta}$ 的结果为：

$$P_{ijk}=T_{jk}a_i \tag{5-21}$$

5.3.5 矢量的叉乘

叉乘只能用于矢量，对于两个矢量 **a** 和 **b**，他们的叉乘为矢量 **c**，其这样计算：

$$\mathbf{c} = \mathbf{a} \times \mathbf{b} = e_{ijk} a_j b_k = (a_2 b_3 - a_3 b_2, a_3 b_1 - a_1 b_3, a_1 b_2 - a_2 b_1) \tag{5-22}$$

其中 e_{ijk} 这样定义：

$$e_{ijk} = \begin{cases} 0 & \text{when any two indices are equal} \\ +1 & \text{when } i,j,k \text{ are an even permutation of } 1,2,3 \\ -1 & \text{when } i,j,k \text{ are an odd permutation of } 1,2,3 \end{cases} \tag{5-23}$$

其中偶次序为偶数次交换相邻检索符得到 123，231，312，奇次序为奇数次交换交换相邻检索符得到 132，213，321。

5.3.6 其他张量操作

下面我们描述 OpenFOAM 提供的一些不太常用的张量操作：

平方 某个张量和自己本身的外积，例如对于矢量 **a** 的平方为：$\mathbf{a}^2 = \mathbf{aa}$；

***n* 次幂** 某个张量和自己本身的 n 次外积，例如对于矢量 **a** 的 3 次幂为：$\mathbf{a}^3 = \mathbf{aaa}$；

模的平方 某个张量和自己本身的阶次内积，例如对于 2 阶张量 **T** 有 $|\mathbf{T}|^2 = \mathbf{T} : \mathbf{T}$；

模 为模的平方的平方根，例如对于张量 **T** 有 $|\mathbf{T}|^2 = \mathbf{T} : \mathbf{T}$。单位大小的矢量我们称为单位矢量；

最大分量 张量的最大分量（注意正负号），注意：非最大的模；

最小分量 张量的最小分量；

平均分量 张量分量的平均；

缩放 缩放用于使用某个张量的元素对另一个张量的元素进行缩放，这两个张量的阶数需要一致，例如：对 **b** 矢量使用 **a** 矢量进行缩放后为矢量 **c**，计算如下：

$$c_i = \text{scale}(\mathbf{a}, \mathbf{b}) = (a_1 b_1, a_2 b_2, a_3 b_3) \tag{5-24}$$

5.3.7 几何变形和单位张量

2 阶张量可以定义为矢量的线性函数，例如：向量 **a** 和向量 **b** 可以通过二阶张量 **T** 联系起来 $\mathbf{a} = \mathbf{T} \cdot \mathbf{b}$。我们可以通过改变 **T** 中的分量来使 **T** 能够对张量实现不同的坐标变换，例如从 x,y,z 的坐标系统变换为 x^*,y^*,z^* 坐标系统。这种变换称之为张量转换。然而，标量在坐标系变换时不发生变化。矢量可实现下述变换：

$$\mathbf{a}^* = \mathbf{T} \cdot \mathbf{a} \tag{5-25}$$

二阶张量 **S** 可以通过下述公式变换成为 $\mathbf{S}^*$：

$$\mathbf{S}^* = \mathbf{T} \cdot \mathbf{S} \cdot \mathbf{T}^{\mathrm{T}} \tag{5-26}$$

单位张量定义为：它将某个张量变换为自己本身，即：

$$\mathbf{a} = \mathbf{I} \cdot \mathbf{a} \tag{5-27}$$

因此单位张量为：

$$\mathbf{I} = \delta_{ij} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5-28}$$

其中 δ_{ij} 为 Kronecker 标示符。

5.3.8 张量等式

下面我们来列举一下常用的张量等式，我们假定相关的导数都是存在且连续的。其中 s 表示标量，$\mathbf{a}$ 表示矢量：

$$\begin{aligned}
\nabla \cdot (\nabla \times \mathbf{a}) &\equiv 0 \\
\nabla \times (\nabla s) &\equiv 0 \\
\nabla \cdot (s\mathbf{a}) &\equiv s\nabla \cdot \mathbf{a} + \mathbf{a} \cdot \nabla s \\
\nabla \times (s\mathbf{a}) &\equiv s\nabla \times \mathbf{a} + \nabla s \times \mathbf{a} \\
\nabla(\mathbf{a} \cdot \mathbf{b}) &\equiv \mathbf{a} \times (\nabla \times \mathbf{b}) + \mathbf{b} \times (\nabla \times \mathbf{a}) + (\mathbf{a} \cdot \nabla)\mathbf{b} + (\mathbf{b} \cdot \nabla)\mathbf{a} \\
\nabla(\mathbf{a} \cdot \mathbf{b}) &\equiv \mathbf{a} \times (\nabla \times \mathbf{b}) + \mathbf{b} \times (\nabla \times \mathbf{a}) + (\mathbf{a} \cdot \nabla)\mathbf{b} + (\mathbf{b} \cdot \nabla)\mathbf{a} \\
\nabla \times (\mathbf{a} \times \mathbf{b}) &\equiv \mathbf{a}(\nabla \cdot \mathbf{b}) - \mathbf{b}(\nabla \cdot \mathbf{a}) + (\mathbf{b} \cdot \nabla)\mathbf{a} - (\mathbf{a} \cdot \nabla)\mathbf{b} \\
\nabla \times (\nabla \times \mathbf{a}) &\equiv \nabla(\nabla \cdot \mathbf{a}) - \nabla^2 \mathbf{a} \\
(\nabla \times \mathbf{a}) \times \mathbf{a} &\equiv \mathbf{a} \cdot (\nabla \mathbf{a}) - \nabla(\mathbf{a} \cdot \mathbf{a})
\end{aligned} \tag{5-29}$$

了解下面这个 $e-\delta$ 等式有助于用户对张量在标识符下的操作有更好的理解：

$$e_{ijk} e_{irs} = \delta_{jr}\delta_{ks} - \delta_{js}\delta_{kr} \tag{5-30}$$

5.3.9 二阶张量运算

下面我们列举几个 2 阶张量的运算法则：

转置 正如如方程未知 所说，$\mathbf{T} = T_{ij}$，$\mathbf{T}^T = T_{ij}$；

symm、skew 正如5.2节所说，我们有对称张量即关于对角线是对称的：$\mathbf{T} = \mathbf{T}^T$。非对称张量（skew 张量）符合这种特性：$\mathbf{T} = -\mathbf{T}^T$，这意味着 $T_{11} = T_{22} = T_{33} = 0$，每个二阶张量都可以分为 symm 部分和 skew 部分即：

$$\mathbf{T} = 0.5\underbrace{(\mathbf{T} + \mathbf{T}^T)}_{\text{symm}} + 0.5\underbrace{(\mathbf{T} - \mathbf{T}^T)}_{\text{skew}} = \text{symm}\,\mathbf{T} + \text{skew}\,\mathbf{T} \tag{5-31}$$

迹 张量的迹为标量（对角线元素的和）：

$$\text{tr}\,\mathbf{T} = T_{11} + T_{22} + T_{33} \tag{5-32}$$

对角 由张量的对角线元素构成的矢量：

$$\text{diag}\,\mathbf{T} = (T_{11}, T_{22}, T_{33}) \tag{5-33}$$

dev、hyd 每个二阶张量都可分解为 dev 部分（其迹为 0）和 hyd 部分（$\mathbf{T} = s\mathbf{I}$，s 为标量）：

$$\mathbf{T} = \underbrace{\mathbf{T} - \frac{1}{3}(\text{tr}\,\mathbf{T})\mathbf{I}}_{\text{dev}} + \underbrace{\frac{1}{3}(\text{tr}\mathbf{T})\mathbf{I}}_{\text{hyd}} = \text{dev}\,\mathbf{T} + \text{hyd}\,\mathbf{T} \tag{5-34}$$

行列式 二阶张量的 det 这样计算：

$$\det\mathbf{T} = \begin{vmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{vmatrix} = \begin{matrix} T_{11}(T22T_{33} - T23T_{32}) - \\ T_{12}(T21T_{33} - T23T_{31}) + \\ T_{13}(T21T_{32} - T22T_{31}) \end{matrix} = \frac{1}{6}e_{ijk}e_{pqr}T_{ip}T_{jq}T_{kr} \tag{5-35}$$

代数余子式 张量的余子式通常这样计算：对于公式 (5-35) 中的 3×2 行列式，删除元素所在的行和列，剩下元素取 2×2 行列式，例如 T_{12} 的余子式为：

$$\begin{vmatrix} T_{21} & T_{23} \\ T_{31} & T_{33} \end{vmatrix} = T_{21}T_{33} - T_{23}T_{31} \tag{5-36}$$

代数余子式是带符号的余子式，符号这样计算：

$$\begin{cases} +, & i+j\text{为偶数} \\ -, & i+j\text{为奇数} \end{cases} \tag{5-37}$$

T 的代数余子式即为：

$$\text{cof}\,\mathbf{T} = \frac{1}{2}e_{jkr}e_{ist}T_{sk}T_{tr} \tag{5-38}$$

逆 张量的逆这样计算：

$$\text{inv}\,\mathbf{T} = \frac{\text{cof}\,\mathbf{T}^T}{\det\mathbf{T}} \tag{5-39}$$

霍奇对偶 张量的霍奇对偶这样计算：

$$*\mathbf{T} = (T_{23}, -T_{13}, T_{12}) \tag{5-40}$$

5.3.10 标量算术

OpenFOAM 支持大部分的标量算术例如平方根、指数、对数、正弦余弦等，它们在表 1.2 中列出。OpenFOAM 还提供了 3 个针对标量的附加函数即：

Sign

$$\text{sgn}(s) = \begin{cases} 1 & \text{如果}s \geq 0 \\ -1 & \text{如果}s < 0 \end{cases} \tag{5-41}$$

Positive

$$\text{pos}(s) = \begin{cases} 1 & \text{如果}s \geq 0 \\ 0 & \text{如果}s < 0 \end{cases} \tag{5-42}$$

Limit

$$\text{limit}(s,n) = \begin{cases} s & \text{如果} s < 0 \\ 0 & \text{如果} s \geq 0 \end{cases} \tag{5-43}$$

5.4 OpenFOAM 张量类

在 OpenFOAM 中，primitive 类包含了距今为止所有的有关张量算术的计算。表5-1列举了 OpenFOAM 提供的基本的张量类型，表中也提供了访问张量元素的函数：

阶数	名字	基本类	访问函数
0	标量	scalar	
1	矢量	vector	x(), y(), z()
2	张量	tensor	xx(), xy(), xz()

表 5-1: OpenFOAM 张量基本类

在 OpenFOAM 中我们可以把张量：

$$\mathbf{T} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \tag{5-44}$$

这样定义：

$$\text{tensor } \mathbf{T}(1,2,3,4,5,6,7,8,9); \tag{5-45}$$

我们可以利用 xz() 函数来访问 T_{13} 或者 T_{xz}，例如：

```
Info << “Txz = ” << T.xz() << endl;
```

这会输出：

```
Txz = 3
```

5.4.1 OpenFOAM 张量操作

5.3节描述的所有张量操作在 OpenFOAM 中都有特定的语法来实现，并且尽可能的和相对应的算符名称或图形相近。一些函数的名字直接使用操作符的名称，例如 `symm()`，其他的一些可以用操作符直接实现，例如 *，所有的函数参见表5-2：

操作	描述	数学形式	OpenFOAM 中的实现
加法		$\mathbf{a}+\mathbf{b}$	a+b
减法		$\mathbf{a}-\mathbf{b}$	a-b
乘以标量		$\mathbf{sb}$	s*a
除以标量		$\mathbf{a}/\mathbf{s}$	a/s
外积	a 和 b 的阶数 ≥ 1	$\mathbf{ab}$	a * b
内积	a 和 b 的阶数 ≥ 1	$\mathbf{a}\cdot\mathbf{b}$	a&b
双内积	a 和 b 的阶数 ≥ 2	$\mathbf{a}:\mathbf{b}$	a&&b
叉乘	a 和 b 的阶数 $= 1$	$\mathbf{a}\times\mathbf{b}$	a^b
平方		$\mathbf{a}^2$	sqr(a)
模的平方		$\lvert\mathbf{a}\rvert^2$	magSqr(a)

模		$\|\mathbf{a}\|$	mag(a)
幂	n = 0,1,⋯,4	$\mathbf{a}^n$	pow(a,n)
元素均值	i = 0,1,⋯,N	$\overline{a_i}$	cmptAv(a)
元素最大值	i = 0,1,⋯,N	$\max(a_i)$	max(a)
元素最小值	i = 0,1,⋯,N	$\min(a_i)$	min(a)
缩放		$\text{scale}(\mathbf{a},\mathbf{b})$	scale(a,b)
2 阶张量操作			
转置		$\mathbf{T}^T()$	T.T()
对角		$\text{diag}(\mathbf{T})$	diag(T)
迹		$\text{tr}(\mathbf{T})$	tr(T)
Dev		$\text{dev}(\mathbf{T})$	dev(T)
Sym		$\text{symm}(\mathbf{T})$	symm(T)
Skew-sym		$\text{skew}(\mathbf{T})$	skew(T)
行列式		$\det(\mathbf{T})$	det(T)
代数余子式		$\text{cof}(\mathbf{T})$	cof(T)
逆		$\text{inv}(\mathbf{T})$	inv(T)
霍奇对偶		$*\mathbf{T}$	*T
标量算术			
取正负		$sgn(s)$	sign(s)
取正		$s \geq 0$	pos(s)
取负		$s < 0$	neg(s)
极限		$\lim(s,n)$	limit(s,n)
平方根		$\sqrt{s}$	sqrt(s)
e		$\exp s$	exp(s)
对数		$\ln s$	log(s)
10 为基的对数		$\log_{10} s$	log10(s)
sin		$\sin s$	sin(s)
cos		$\cos s$	cos(s)
tan		$\tan s$	tan(s)
arcsin		$\text{asin } s$	asin(s)
arccos		$\text{acos } s$	acos(s)
arctan		$\text{atan } s$	atan(s)
sinh		$\sinh s$	sinh(s)
cosh		$\cosh s$	cosh(s)
tanh		$\tanh s$	tanh(s)
arcsinh		$\text{asinh } s$	asinh(s)
arccosh		$\text{acosh } s$	acosh(s)
arctanh		$\text{atanh } s$	atanh(s)
误差函数		$\text{erf } s$	erf(s)
C 误差函数		$\text{erfc } s$	erfc(s)
Gamma 对数函数		$\ln \Gamma s$	lgamma(s)
0 阶 1 型 Bessel 函数		$\mathbf{J}_0 s$	j0(s)
1 阶 1 型 Bessel 函数		$\mathbf{J}_1 s$	j1(s)
0 阶 2 型 Bessel 函数		$\mathbf{Y}_0 s$	y0(s)

1 阶 2 型 Bessel 函数		$\mathbf{Y}_1 s$	y1(s)
a 和 **b** 为任意阶数的张量（除非进行指定），s 为标量，N 为元素数			

表 5-2: OpenFOAM 中的张量操作

5.5 量纲单位

在连续介质里，物理量都具有量纲。例如质量：kg, 体积：m^3，压力：Pa。这些物理量的代数几何操作要求量纲保持一致。对于某些同样量纲的物理量，只有加、减、等于是具有物理意义的。为了防止对物理量进行无意义或者错误的运算，OpenFOAM 推荐用户将量纲依附于场数据以及某个物理量，这样在对其进行张量操作的时候，OpenFOAM 会执行量纲检查。量纲我们用 `dimensionSet` 类来定义，例如：

```
dimensionSet pressureDims(1, -1 ,-2, 0, 0, 0, 0);
```

编号	名称	SI 单位	USCS 单位
1	质量	kg	lbm
2	米	m	ft
3	时间	s	s
4	温度	K	°R
5	摩尔质量	kgmol	lbmol
6	电流	A	A
7	光强	cd	cd

表 5-3: SI 量纲

表5-3列举了 OpenFOAM 植入的 S.I. 量纲制。我们上个范例中的 `pressureDims` 使用 `dimensionSet` 把单位设置为 $\mathrm{kg\,m^{-1}s^{-2}}$，因为我们 `pressureDims` 序列中第一个值为 1，这对应 k^1，第二个值为 -1，这对应 m^{-1}，以此类推。附带单位的张量使用 `dimensioned<Type>` 模板类来定义，其中 `type` 是 `scalar`，`vector`，`tensor` 等。`dimensioned<Type>` 将存储一个 `word` 类、一种 `<Type>` 类型、和 `dimensionSet`：

```
dimensionedTensor sigma
(
    "sigma",
    dimensionSet                (1, -1, -2, 0, 0, 0, 0),
    tensor                      (1e6,0,0,0,1e6,0,0,0,1e6),
);
```

上述这段代码将会定义这样一个张量（应力单位）：

$$\sigma = \begin{pmatrix} 10^6 & 0 & 0 \\ 0 & 10^6 & 0 \\ 0 & 0 & 10^6 \end{pmatrix} \tag{5-46}$$

第6章　网格生成和转换

这一章我们讨论所有和网格生成有关的话题。6.1节总览一下 OpenFOAM 所用的网格方式；6.2节介绍网格边界类型；6.3节介绍自动生成简单六面体网格的程序 `blockMesh`；6.4节介绍 `snappyHexMesh` 程序，它用来依据几何文件全自动切分六面体网格并生成复杂的多面体网格；6.5节介绍如何用程序把第三方网格转换成 OpenFOAM 可用的格式。

6.1　网格

这一节我们介绍 OpenFOAM 中使用的 C++ 类如何处理网格。网格是数学求解的主要部分，并且必须满足某些标准以确保求解有效和准确。在运行过程中，OpenFOAM 有一套非常严格的检查网格的标准，如果不被满足的话运行将会停止。这样的不利后果是，在修正第三方软件生成的大型网格以确保 OpenFOAM 可以使用的时候，用户可能会感到很困难并且非常沮丧。这非常的不幸。因此确保网格的有效性是非常重要的。否则，在求解开始之前，结果就存在缺陷。

默认情况下，OpenFOAM 可以识别下面这种网格单元形式：即一个使用任意数量的面来定义的一个有界的 3D 多面体。例如，单一的网格单元可以有无数个面，对于每个面，边的数量也没有限制，边的结合方式也没有限制。具有这样结构的网格在 OpenFOAM 中被熟知为 `polyMesh`。在几何非常复杂的时候，或者网格随时间改变的时候，这对网格生成以及操作提供了最大的自由。

6.1.1　网格规范以及限制

在对 OpenFOAM 网格格式 `polyMesh` 文件夹，以及 `cellShape` 工具进行描述之前，我们首先制定了一些网格规则：

6.1.1.1　点

点文件由一个三维空间中的一个位置矢量来定义，单位为米。点文件被编写成了一个列表（`list`），每个点文件由 `label` 标示，从 0 开始，代表在 `list` 中的位置。不能在一个相同的物理位置定义两个不同的点，也不允许有不在任何面内的点存在。

6.1.1.2　面

面是排列有序的点集，点由其序号来表示。每个点通过边来相连，并以此来指定点的顺序。例如：如果你跟随着这个点，你将围着这个面环绕一周。面被汇集在一个列表中，每个面由它的 `label` 标示，代表它在列表中的位置。面法向矢量由右手规则定义。例如，当你看一个面得时候，如果一个面上的点成逆时针方向排序，那么这个面矢量就是面对你的。如图6-1所示：这里存在两种面：

内部面 这些面将两个网格单元连接起来（仅仅是两个网格单元）。每个内部面的面法向矢量指向标识数比较大的网格单元。点由面法向的方向进行排序。例如对于连接网格单元 2 和网格单元 5 的内部面，面法向矢量从网格单元 2 指向网格单元 5；

边界面 它们仅属于一个网格单元，由于和边界重合，因此边界面由一个网格单元和一个 `patch` 来

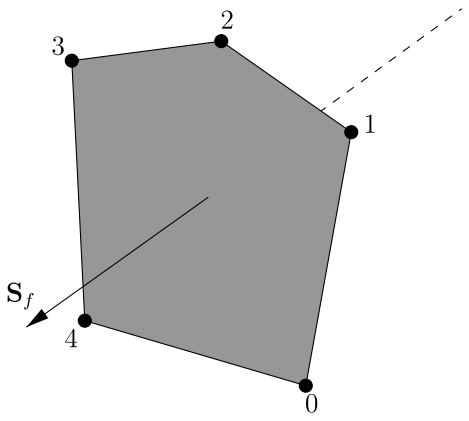


图 6-1: 依据点序来查看面矢量

指定。面法向矢量指向计算域之外，点由面法向矢量的方向来排序。

网格的面应该是凸起的，至少面心应该在面内。面也可以歪斜，并不是所有的面都是平面。

6.1.1.3 网格单元

网格单元是面的汇集，单个网格单元必须具有以下特点：

连续性 所有的网格单元必须完整的覆盖计算域，并且彼此不能重合；

凸起的 每个网格单元必须是凸起的，并且网格单元中心在网格内；

封闭的 不管从几何来讲，还是拓扑结构来说，每个网格单元必须是封闭的；

1. 几何封闭要求当把网格单元的面矢量加和的时候，总和应该等于 0（机械精度）；
2. 拓扑封闭要求一个网格单元内所有的边都由两个面共有。

6.1.1.4 边界

边界是一系列 `patch` 的集合，每个边界和一个边界条件相对应。每个 `patch` 是一系列面的集合，这些面只包含边界面并且不能是内部面。边界必须是封闭的，例如，所有的边界面矢量之和为 0（机器精度）。

6.1.2 polyMesh

`constant` 文件夹下的 `polyMesh` 文件夹包含了对 `polyMesh` 的全部描述。`polyMesh` 建立在我们之前讨论过点、内部面（见上述讨论）以及边界面（通过一系列网格单元的面以及边界 `patch` 来指定）基础之上。每个面被指派了一个 `owner` 和一个 `neighbor`，因此每个连接两个网格的面都具有一个 `owner` 以及 `neighbor`。对于边界，和它连接的网格是 `owner`，`neighbor` 被指定为 `-1`。记住这点之后，网格的输入输出包含以下文件：

points 一系列描述网格顶点的位置矢量，第一个矢量代表点 0，第二个矢量代表定点 1，以此类推；

faces 包含了一系列的面，每个面用点序号组成并标识, 第一行信息代表面 0。以此类推；

owner 包含一系列的 owner 标识，这里面的信息和面相关。第一行的信息表示面 0 的 owner 序号，

第二行信息标示面 1 的 owner 序号，以此类推；

neighbor 一系列的 neighbor 标识，参考上述 owner 的描述；

boundary 包含一系列的 patch（不同的 patch 具有不同的名字），每个

面具有相应的字典信息，例如：

```
movingWall
{
    type patch;
    nFaces 20;
    startFace 760;
}
```

startFace 是这个 patch 中面 1 的标识，nFaces 是这个 patch 中面的总数。

6.1.3 cellShape

OpenFOAM 支持任意的网格单元形状，其他的软件则未必。因此在进行转换的时候，需要定义网格单元形状，例如六面体、四面体等。$FOAM_ETC 目录下的 cellModels 文件内定义的 cellShape 类主要执行的就是这个功能。

表 5.1 中列举了 OpenFOAM 支持的网格形状。每个网格单元类型的点、面、边数量都已经预定义。网格单元的描述由两部分构成，网格单元名称以及排列有序的标识列表。因此，如果我们有以下的点集：

```
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)
)
```

一个六面体网格可以这样描述：

```
(hex 8(0 1 2 3 4 5 6 7))
```

在这里六面体网格形状用 hex 来表述。上面的代码为 blockMesh 的基本代码段，详情请参考6.3节。

6.1.4 一维、二维以及轴对称问题

OpenFOAM 代码主要针对三维问题并且按照三维问题来定义网格。然而，OpenFOAM 同样也可以计算一, 二维以及轴对称问题，你首先需要生成一个三维网格，然后在任何的 patch（那个不需

计算的 patch）上应用一个特定的边界条件即可。在一，二维问题上的第三方向我们使用 empty 边界条件，轴对称问题使用 wedge 边界类型。请参阅6.2.2章节的详细描述，轴对称问题的 wedge 几何生成在6.3.5节有描述。

6.2 边界

这一节我们讨论 OpenFOAM 中的边界条件。在之前的描述中，边界问题我们很少提及，因为它们的角色不仅仅是一个几何实体，相反，它们是边界求解和数值运算的一部分，也可以说是把内部场和边界条件联系起来的纽带。之前我们在讨论网格、场、离散、计算过程的时候讨论边界有点不太合适。因此它被放在这一章是不错的选择。

我们首先需要了解到，边界通常被分割为一系列 patch 的集合，每个 patch 包含了一系列的边界面（封闭的面），边界面不需要是物理相连的。我们按照继承的关系，列举了边界类型。正如6.1.2节所描述的，每个 patch 需要指定类型。下面是 rhoPimpleFoam 的 p 文件边界条件范例（很明显，每个 patch 需要指定 type 类型）：

```
6
(
    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;
        inGroups        1(empty);
        nFaces          10500;
        startFace       10675;
    }
)
```

用户可以扫描自带算例中的字典文件 blockMeshDict（如果使用 blockMesh 生成网格）或 snappyHexMeshDict（如果使用 snappyHexMesh 生成网格）来查看各个类型的 type 使用。例如，用户可以通过下面的命令获取 symmetryPlane 边界条件的使用：

```
foamInfo -a symmetryPlane
```

同理，下面的命令获取所有 snappyHexMeshDict 字典文件中 wall 边界条件的使用：

```
find $FOAM_TUTORIALS -name snappyHexMeshDict | \
xargs grep -El "type[\t ]*wall"
```

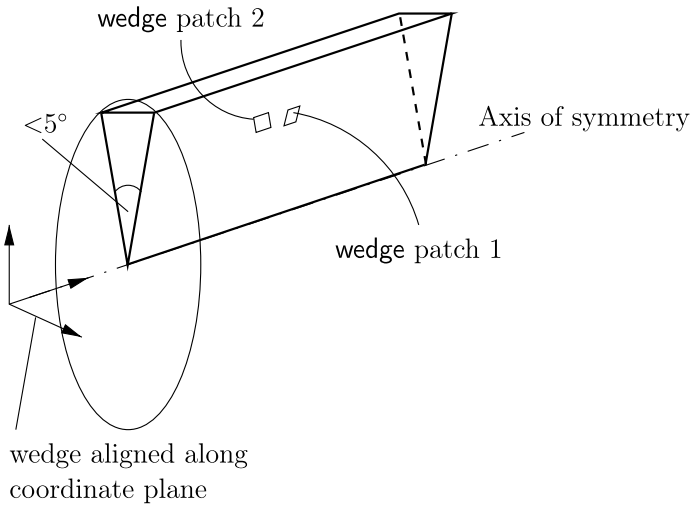


图 6-2: wedge 轴对称几何

6.2.1 几何边界类型

本节概括 OpenFOAM 中主要的几何边界类型。完全的可用边界类型请参考源代码：`finiteVolume/fields/fvPatchFields/constraint`

`patch` 若指定 `patch` 边界条件，那么就说明网格上没有几何拓扑信息（`wall` 除外），例如 `inlet` 和 `outlet`；

`wall` 主要用于可识别的壁面，在这种情况下，需要指定为 `wall`，这样才能使某些特定的模型可选。一个很好的例子就是在湍流壁面函数中，壁面必须指定为 `wall`，这样壁面和网格中心的距离才会被存储；

`symmetryPlane`[1] 严格的对称平面防止累积误差；

`symmetry` 如果面略有不平，可用此边界条件应用对称边界条件；

`empty` OpenFOAM 生成的是三维网格，它可以通过把第三个方向（不在此方向进行求解）上的 `patch` 指定为 `empty` 边界类型来实现一维或者二维运算；

`wedge` 对于二维轴对称算例，例如圆柱。几何可以被定义一个楔形，它的夹角很小，例如 1 度，它的轴要和圆柱的轴平行，如图6-2所示。轴对称的平面必须分别通过不同的 `patch` 来指定为 `wedge` 类型。

`cyclic` 这种边界条件可以使两个不相连的 `patch` “物理上连接起来”。主要用于周期几何，例如散热片。想把一个 `cyclic` 边界和另一个 `cyclic` 边界连接起来，需要在 `boundary` 文件中指定另一个为 `neighbourPatch`。互相匹配面的面积需要相同，至少在设置的误差内，它可以在 `boundary` 文件中的 `matchTolerance` 关键词后设置。这两个面不需要方向相同；

`cyclicAMI` 类似于 `cyclic`，但是不需要面网格是一一对应的，主要用于滑移网格方法；

`processor` 如果需要并行计算，那么就需要分割网格，只有这样每个处理器才能大体计算相同数量的网格。不同核之间处理的网格的边界成为 `processor` 边界。

6.2.2 基本边界类型

如4.2.8节描述，边界条件通过时间步下面的文件（例如 U 或 p）进行定义。下面是 rhoPimpleFoam 下压力场的边界条件定义：

```
17  dimensions                  [1 -1 -2 0 0 0 0];
18
19  internalField               uniform 1;
20
21  boundaryField
22  {
23      inlet
24      {
25          type                fixedValue;
26          value               uniform 1;
27      }
28
29      outlet
30      {
31          type                waveTransmissive;
32          field               p;
33          psi                 thermo:psi;
34          gamma               1.4;
35          fieldInf            1;
36          lInf                3;
37          value               uniform 1;
38      }
39
40      bottom
41      {
42          type                symmetryPlane;
43      }
44
45      top
46      {
47          type                symmetryPlane;
48      }
49
50      obstacle
51      {
52          type                zeroGradient;
53      }
54
55      defaultFaces
56      {
57          type                empty;
58      }
59  }
```

每个 patch 包含 type 关键词用来指定具体的边界条件类型。其中 inlet 简单的指定为 fixedValue 边界条件，outlet 则使用了复杂的 waveTransmissive 边界条件。若 patch 的几何边界定义为 symmetryPlane，对应的，p 文件中的 patch 也应该定义为 symmetryPlane。

下面我们列举 OpenFOAM 主要的边界条件类型。全部得边界条件类型请参考源代码：$FOAM_SRC/finiteVolume/fields/fvPatchFields/basic

fixedValue 指定 φ 的值，value。

fixedGradient 指定 φ 的法向梯度，gradient。

zeroGradient φ 法向梯度为 0。

calculated φ 的值从其他场的值计算而来。

mixed 依据 valueFraction 的值而变的 fixedValue 和 fixedGradient 混合形式。如果 valueFraction 为 1，φ 对应指定的 refValue，如果为 0，则指定梯度 refGradient。

directionMixed valueFraction 具有方向的 mixed 形式，例如，对于法向和切向的混合级数不同的情况。refValue refGradient valueFraction value。

6.2.3 衍生边界类型

在基本边界类型的基础上，OpenFOAM 中有大量在基本边界类型基础上衍生的边界条件。例如，许多的边界条件从基本类型 fixedValue 上进行衍生，其具体的值从函数计算而来或从其他

场计算而来。一些其他的边界条件则从 mixed/directionMixed 中进行互相转换。

用户可以选用不同的方式列举 OpenFOAM 可用的边界条件类型，其中命令参数-listScalarBCs 以及 -listVectorBCs 是最为方便的。这些参数将标量和矢量一一列举。例如下面的命令：

```
simpleFoam -listScalarBCs -listVectorBCs
```

列举 simpleFoam 可用的边界条件类型。如果用户想了解一个特定的边界条件信息，可以在 OpenFOAM 自带算例中寻找相关的例子来帮助理解，或者参考边界条件的源代码中的 Description 部分。例如，对于边界条件 totalPressure，用户可以使用下面的命令获取相关信息：

```
foamInfo  totalPressure
```

源代码文件可以通过下面的命令来定位：

```
find $FOAM_SRC -name totalPressure
```

下面我们来讨论一些比较常见的边界条件类型。

6.2.3.1 inletOutlet

inletOutlet 边界条件从 mixed 边界条件衍生而来，其自动在 zeroGradient 以及 fixedValue 之中进行切换。若流体向外流动，则指定 zeroGradient，反之则指定 fixedValue。对于流入的流体，进入的信息通过 inletValue 来指定[2]。用户可参考 damBreak 这个案例来更好的理解这个边界条件，在这个算例中，其用来指定相分数的 atmosphere 信息。若流出，则表示相分数的边界条件为 zeroGradient，若流入，则固定相分数的 fixedValue 为 0，其对应 100% 的空气。

```
17  dimensions              [0 0 0 0 0 0 0];
18
19  internalField           uniform 0;
20
21  boundaryField
22  {
23      leftWall
24      {
25          type                zeroGradient;
26      }
27
28      rightWall
29      {
30          type                zeroGradient;
31      }
32
33      lowerWall
34      {
35          type                zeroGradient;
36      }
37
38      atmosphere
39      {
40          type                inletOutlet;
41          inletValue          uniform 0;
42          value               uniform 0;
43      }
44
45      defaultFaces
46      {
47          type                empty;
48      }
49  }
```

[2]除了 inletValue 外，还需要指定 value 关键词，在这里只是由于 C++ 的特性所致，value 的内容并没有具体的用处。

6.2.3.2 卷吸边界条件

在某些情况下，回流必然会产生，并且入流的速度不为人知。这种情况下，可以对压力使用 totalPressure 边界条件，对速度使用 pressureInletOutletVelocity 边界条件。在 damBreak 这个算例中，atmosphere 边界即采用的这种边界条件组合。

totalPressure 边界条件的计算公式如下：

$$p = \begin{cases} p_0 \\ p_0 - \frac{1}{2}\left|\mathbf{U}^2\right| \end{cases} \tag{6-1}$$

用户需要通过 p0 给定 p_0 的值。关键词 pressureInletOutletVelocity 一般情况下等同于 zeroGradient，但是进口的切向分量可以给定，其默认为 0。

这种边界条件组合的背后原理主要是对出流应用标准的边界条件组合，但是对入流则可以指定入流速度。但是某些情况下，速度的快速增加会导致不稳定性。如果我们把出口回流的压力适当减小，速度的增加则会减轻，其对应则压力梯度驱动力随着回流进口速度的增加而增加。

下面是 damBreak 算例中的 U 以及 p_rgh 的边界条件：

```
dimensions              [0 1 -1 0 0 0 0];

internalField           uniform (0 0 0);

boundaryField
{
    leftWall
    {
        type                noSlip;
    }
    rightWall
    {
        type                noSlip;
    }
    lowerWall
    {
        type                noSlip;
    }
    atmosphere
    {
        type                pressureInletOutletVelocity;
        value               uniform (0 0 0);
    }
    defaultFaces
    {
        type                empty;
    }
}
```

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

internalField           uniform 0;

boundaryField
{
    leftWall
    {
        type                fixedFluxPressure;
        value               uniform 0;
    }

    rightWall
    {
        type                fixedFluxPressure;
        value               uniform 0;
    }

    lowerWall
    {
        type                fixedFluxPressure;
        value               uniform 0;
    }

    atmosphere
    {
        type                totalPressure;
        p0                  uniform 0;
    }

    defaultFaces
    {
        type                empty;
    }
```

```
51  }
```

6.2.3.3 fixedFluxPressure

用户也可以看到，在上面的所有代码段中，除了 atmosphere 以外，其他的边界条件类型均指定为了 fixedFluxPressure。一般情况下，如果用户想要给定压力的零法向梯度边界条件，但同时求解的方程还存在体积力（如重力、表面张力等），建议使用 fixedFluxPressure，其会自行的调整压力梯度。

6.2.3.4 依时类边界条件

OpenFOAM 中存在一些依时类边界条件，其具体的值和时间有关。在源代码中，这些边界条件可以通过下面的命令来获取：

```
find $FOAM_SRC/finiteVolume/fields/fvPatchFields -type f | \
xargs grep -l Function1 | xargs dirname | sort -u
```

例如，uniformFixedValue 是一个典型的依时类边界条件，在将时间函数（关键词 uniformValue）指定为 constant 的时候，其演变为 fixedValue 边界条件。下面是关键词 uniformValue 可选的类型：

constant 常数值；

table 自定义配对，时间步之间采用线性插值；

tableFile 同上，但其需要读取一个单独的文件；

csvFile 同上，但其需要读取一个单独的 csv 文件；

square 平方波函数；

sine sine 函数；

one 或 zero 1 或者 0 常量；

polynomial 多项式；

scale 通过 scale 关键词对 value 进行缩放；scale 以及 value 均可以为函数；

linearRamp, quadraticRamp, halfCosineRamp, quarterCosineRamp 以及 quarterSineRamp 从 0 至 1 的斜率函数。

下面是几个具体的依时类边界条件范例：

```
inlet
{
    type         uniformFixedValue;
    uniformValue constant 2;
}

inlet
{
    type         uniformFixedValue;
    uniformValue table ((0 0) (10 2));
}

inlet
{
    type         uniformFixedValue;
```

```
    uniformValue polynomial ((1 0) (2 2)); // = 1*t^0 + 2*t^2
}

inlet
{
    type         uniformFixedValue;
    uniformValue
    {
        type             tableFile;
        file             "dataTable.txt";
    }
}

inlet
{
    type         uniformFixedValue;
    uniformValue
    {
        type             csvFile;
        nHeaderLine      4;              // number of header lines
        refColumn        0;              // time column index
        componentColumns (1);            // data column index
        separator        ",";            // optional (defaults to ",")
        mergeSeparators  no;             // merge multiple separators
        file             "dataTable.csv";
    }
}

inlet
{
    type         uniformFixedValue;
    uniformValue
    {
        type             square;
        frequency        10;
        amplitude        1;
        scale            2;  // Scale factor for wave
        level            1;  // Offset
    }
}

inlet
{
    type         uniformFixedValue;
    uniformValue
    {
        type             sine;
        frequency        10;
        amplitude        1;
        scale            2;  // Scale factor for wave
        level            1;  // Offset
    }
}

input // ramp from 0 -> 2, from t = 0 -> 0.4
{
    type
        uniformFixedValue;
    uniformValue
    {
        type             scale;
        scale            linearRamp;
        start            0;
        duration         0.4;
        value            2;
    }
}
```

6.3 blockMesh 网格生成程序

这一章节我们讨论 OpenFOAM 提供的 `blockMesh` 程序，`blockMesh` 程序用来创建均匀或者非均匀分布以及曲边的网格。

网格通过位于 `constant/polyMesh` 字典下的 `blockMeshDict` 来生成，`blockMesh` 读取这个字典，在同样的文件位置下生成网格，输出点、面、网格和边界信息。

`blockMesh` 的原则是把计算域分解成为一个或者多个三维的六面体块。这些块的边可以为直线也可以为曲线。在每个六面体块上，指定每个方向的网格数量。使用这些信息就足够用来生成网格数据了。

每个 `block` 由 8 个顶点来定义，它们位于六面体的每个角上。顶点被写入了列表因此它们可以通过标志号来被计算机识别。OpenFOAM 就是 C++ 程序，第一个标志永远是“0”。表 5.3 就是一个 `block` 的典型例子。每个顶点都进行了排号。连接顶点 1 和顶点 5 的边是曲边，这表明

blockMesh 也可以指定曲边。

blockMesh 也可以生成少于 8 个顶点的 block，这需要将一个或者一对的顶点去除，6.3.5节会有详细的描述。

每个 block 的局部坐标系统 (x1,x2,x3) 都要遵循右手规则。右手规则的定义是：当用户顺着 oz 方向看的时候（o 点为离用户近的点），ox 上的点到 oy 上的点的弧的方向为顺时针。

局部坐标系统的定义是有顺序的，block 的顶点这样来定义：

- 坐标轴顶点是 block 定义的第一条信息，我们的例子中是顶点 0；
- 从顶点 0 到顶点 1 是 x1 轴；
- 从顶点 1 到顶点 2 是 x2 轴；
- 顶点 0，1，2，3 定义 x3=0 的平面；
- 顶点 4 位于将顶点 0 顺着 x3 方向移动的轴上；
- 类似的，5，6，7 顶点就是位于 1，2，3 顶点分别沿着 x3 移动的轴上。

6.3.1 编写 blockMeshDict 文件

blockMeshDict 使用表 5.5 所列的关键字来定义，convertToMeters 指定点矢量缩放因子，所有的顶点坐标矢量都要乘以这个因子。例如：

```
convertToMeters  0.001
```

意味着所有坐标值都乘以 0.001. 这样，blockMeshDict 中的值就是以 mm 为单位。

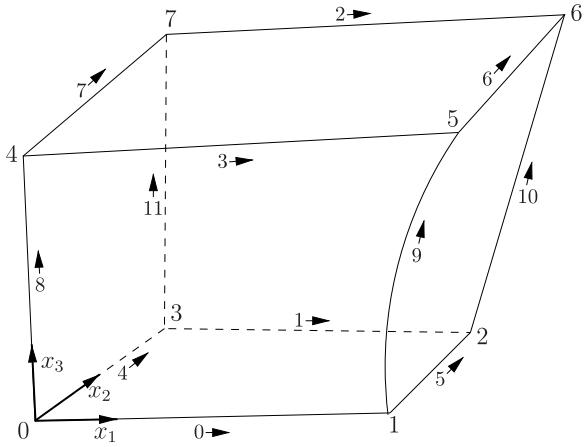


图 6-3: 单一 block 块

关键词	描述	范例
converToMeters	点位置矢量缩放因子	0.001，缩放为 mm
vertices	点位置列表	（0 0 0）
edges	用于指定 arc 以及 spline 边	arc 1 4 (0.939 0.342 -0.5)

block	通过有序的点定义，以及每个方向的网格数量	hex (0 1 2 3 4 5 6 7)(10 10 1)
patches	patches 列表	symmetryPlane base((0 1 2 3))
mergePatchPairs	需要合并的 patches 列表	详见6.3.2节

表 6-1: blockMeshDict 关键词

6.3.1.1 顶点

block 顶点通过一个标准列表 vertices 来指定，例如，表 5.4 中的 block 顶点这样定义：

```
vertices
(
( 0 0 0 )                 //顶点 0
( 1 0 0.1)                //顶点 1
( 1.1 1 0.1)              //顶点 2
( 0 1 0.1)                //顶点 3
(-0.1 -0.1 1 )            //顶点 4
( 1.3 0 1.2)              //顶点 5
( 1.4 1.1 1.3)            //顶点 6
( 0 1 1.1)                //顶点 7
);
```

6.3.1.2 边

连接两个顶点的边默认为是直线，然而很多边都可以通过 edges 来指定为曲线。这个 list 是可以选择的，如果 block 没有曲边，它可以省略。每个曲边通过表6-2列举的关键词来指定：

关键词	描述	附加信息
arc	圆弧	需要一个插值点
simpleSpline	样条曲线	一系列插值点
polyline	线集	一系列插值点
polySpline	样条曲线集	一系列插值点
line	直线	——

表 6-2: blockMeshDict 可用的边类型

如果这个边通过某些插值点，他们也需要被指定。例如，对于一个 arc，只需要定义一个 arc 经过的插值点即可。对于 simpleSpline，polyLine 以及 polySpline，需要定义一系列插值点。line 就是缺省的边，不需要插值点信息。实际上我们并不需要给定 line 的信息，但是我们包含了它以确保完整性。图6-4中的 block 我们指定一个 arc 边来连接顶点 1 和顶点 5，并且它们通过了插值点 (1.1,0.0,0.5)：

```
edges
(
arc    1 5 (1.1 0.0 0.5)
);
```

6.3.1.3 块

block 块通过 blocks 列表来定义。每个 block 需要一系列的信息组合来定义。包括 block

的顶点（它们的顺序在6.3节中有描述）、每个方向上的网格数量、每个方向上的网格单元膨胀率。

其 block 这样定义：

```
blocks
(
hex (0 1 2 3 4 5 6 7)     // 顶点数
(10 10 10)            // 每个方向的网格数
simpleGrading (1 2 3)     // 网格单元膨胀率
);
```

每个 block 的信息如下：

顶点数 在顶点数信息之前定义的是 block 的形状信息，它们在.OpenFOAM-9/cellModels 文件中声明。由于 blocks 经常是六边形，因此 hex 居多。然后它列举了顶点信息，以后文所提的方式来排序；

网格数量 第二行信息定义 x1，x2，x3 方向的网格数量；

网格单元膨胀率 第三行信息定义每个网格方向上的网格单元膨胀率。膨胀率使网格在特定方向上能够非均匀化或细化。参阅图6-4，膨胀率就是一条边上最后一个网格的宽度 δe 除以最起始网格的宽度 δs。下面列举的两个关键词是 blockMesh 可用的非均匀化关键词；

simpleGrading 非均匀化最简单的方式，对 x1，x2，x3 方向设置统一的膨胀率，它可以这样来定义：

```
simpleGrading (1 2 3)
```

edgeGrading 在这种方式中，可对 block 的每个边定义网格单元膨胀率。他们通过图6-4的规则来排序，图中箭头的方向指的是边的起始方向，如果有这样的信息：

```
edgeGrading (1 1 1 1 2 2 2 2 3 3 3 3)
```

这意味着 0-3 边的膨胀率为 1，4-7 边的膨胀率为 2，8-11 边的膨胀率为 3，这和 simpleGrading 的描述类似。

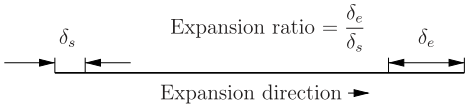


图 6-4: block 边的非均匀处理

6.3.1.4 block 多重非均匀处理

采用上述的方法对 block 定义膨胀率只能对 block 的某个方向上进行一个单一的放大或者缩小处理。其优点为在某些情况下，这使得问题简单化。因为这种方法使得每个 block 在每个方向上存在一个单一的膨胀率。但是在某些情况可能会使得问题复杂化。例如，如果我们定义一个管道的 block，如果我们打算在管道竖直方向上的上下壁面全部调用非均匀分布网格，那么我们需要定义三个 block：上下壁面各一个 block，中间一个 block。

OpenFOAM-2.4 以上的版本对这个功能进行了拓展，即每个 block 可以针对每个方向指定多个节点非均匀分布并且可以应用不同的膨胀率。相对于上一个小节提到的单一非均匀分布（在

block 上某个方向如 1,2,3 方向上指定单一的膨胀率），多重非均匀分布在每个方向上可以进一步的指定多个膨胀率。下面是一个单一非均匀分布的范例：

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

我们打算对其使用多重非均一分布，其特点如下：

- 在 y 方向切分为 3 个方向，每段长为全边长的 20%（1 区间），60%（2 区间），20%（3 区间）；
- 在 1，3 区间分别包含 y 方向总网格数（300）的 30
- 在 1，3 区间设置膨胀率为 1:4，2 区间无膨胀率。

上述的设置参考如下（参考单一非均匀分布，将其中的“2”进行替换）：

```
blocks
(
hex (0 1 2 3 4 5 6 7) (100 300 100)
simpleGrading
(
1                   // x 方向膨胀率
(
(0.2 0.3 4)     // y 边长的 20%, 30% 的网格, 膨胀率为 4
(0.6 0.4 1)     // y 边长的 60%, 40% 的网格, 膨胀率为 1
(0.2 0.3 0.25) // y 边长的 20%, 30% 的网格, 膨胀率为 0.25
)
3                   // z 方向膨胀率
)
);
```

在多重非均匀分布指定的时候，其中的分数被自动处理。其可以为百分数、分数或者绝对值。并且，总和不需要为 100。例如，上面的代码也可以这样写：

```
blocks
(
hex (0 1 2 3 4 5 6 7) (100 300 100)
simpleGrading
(
1
(
(20 30 40) //20\%, 30\%...
(60 40 1)
(20 30 0.25)
)
3
)
);
```

6.3.1.5 边界

网格的边界通过 boundary 来定义。边界被拆分为 patch，每个 patch 都有它们的名字，用户可以进行选择，但是我们建议把 patch 命名为具有描述性的名字，例如 inlet。patch 下的信息以子字典的形式来包含：

type 表述 patch 类型，要么是一个普通类型，要么就是一个特殊的几何条件，详见6.2.1节；

faces 一系列的 block 面，它们可以组成一个 patch，每个 patch 都有它们的名字，用户可以进行选择，但是我们建议把 patch 命名为好识别的名字，例如 inlet。

如果用户没有在 boundary 中指定某些面信息，blockMesh 默认它们为 defaultFaces，类型为 empty。这意味着是一个 2D 的几何，如果用户知道 blockMesh 会收集省略的面信息并把他们指定为 empty，它就可以省略这部分面定义。

下面我们回到图6-3的例子，如果它在左边的面上有一个 inlet，右面的面上有一个 outlet，其它的面都是 walls，那么它可以这样定义：

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

    outlet                  // patch name
    {
        type patch;         // patch type for patch 1
        faces
        (
            (1 2 6 5)
        );
    }

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

每个 block 面通过 4 个顶点来定义，顶点的顺序必须这样定义，从 block 里面来看，它们起始于任何顶点，其它顶点以顺时针方向来排序。

如果要在 blockMesh 里面指定 cyclic 边界条件，需要 neighbourPatch 关键词来指定相关的 cyclic 边界。例如，一对 cyclic 边界需要这样指定：

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

6.3.2 多块网格

网格可以用多个 block 块来创建，在这种条件下，单个 block 块中生成的网格参见前文所述，唯一的问题是 blocks 之间如何相连，这存在两种情况：

面匹配 在某一个 block 中，组成某个 patch 的面由某些顶点来构成，这些顶点又被另一个 block 的 patch 下的面所用；

面融合 某个 block 下的 patch 下的面和另一个 block 下某个 patch 下的面相连，通过这种方式来创造连接两个 block 的内部面。

通过面匹配来连接两个 block 的时候，这个 patch 的信息可以不定义。blockMesh 会识别这些 patch，因为它们不是作为外部边界来存在的，因此 blockMesh 将它们来匹配成为内部面，这个 patch 连接两个 block 的网格。

相对应的，面融合首先要求融合的面在 patch 中必须被定义。然后在 mergePatchPairs

中把需要融合的 patch 信息包含进去，参考如下格式：

```
mergePatchPairs
(
( <masterPatch> <slavePatch> )        // 面融合的第一个 patch 对
( <masterPatch> <slavePatch> )        // 面融合的第二个 patch 对
...
)
```

成对的 patch 可以这样理解：第一个面是主面，第二个面是次面。融合规则如下：

- 主面按照原先的定义，所有的面顶点在它们最原始的坐标上；
- 次面被映射到主面上，即使次面和主面之间是分离的，这也完全可行；
- 在融合进行的过程中，如果有些面的边小于设定的最小值，blockMesh 会调整次面上的顶点来消除这些边；
- 参阅图 5.6，如果两个面有这样的重叠，重叠的部分成为内部面，没有融合的部分依然为外部面，需要指定边界条件；
- 如果 patch 上所有的面都融合了，那么这个 patch 将被完全的移除。

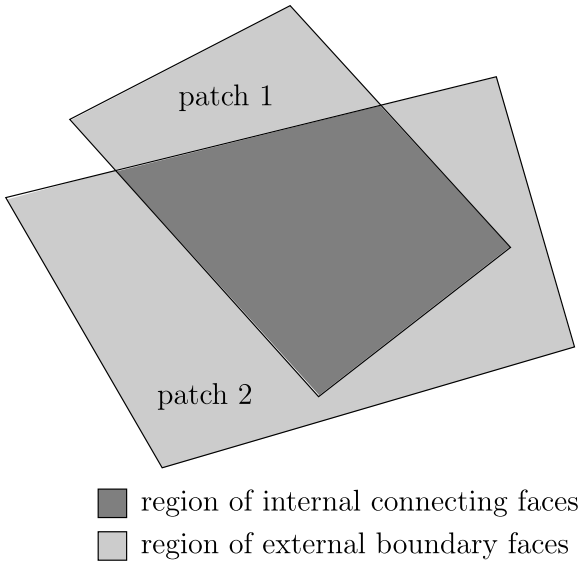


图 6-5: 重叠 patch 的融合

我们进行面融合的一个后果是，次面的原始几何不会被完整的保存。因此，如果某个算例中一个圆柱的 block 要和一个更大的 block 相连，最好指定圆柱为主面，这样圆柱的形状会得以保留。为了融合成功，我们提供了以下几个意见：

- 在 2D 几何中，第三维的网格大小，例如 2D 平面外的那个方向的长度，应该和 2d 平面内网格的长度大体一致；
- 不建议融合一个 patch 两次，例如在 mergePatchPairs 把他们包含两次；
- 当两个 patch 面共享一个边的时候，这两个 patch 都需要被指定为主面。

6.3.3 顶点、边以及面映射

blockMesh 可以将顶点、边、或者面映射至指定几何形状来生成适体网格。这个功能主要可以用来生成管道、球形等网格。用户可以在 blockMeshDict 中指定 geometry 子字典，这个子字典和 snappyHexMesh 指定的子字典信息相同。例如，如果用户打算使用内置的几何形状生成圆柱网格，可以这样给定子字典信息：

```
geometry
{
  cylinder
  {
    type searchableCylinder;
    point1 (0 -4 0);
    point2 (0 4 0);
    radius 0.7;
  }
};
```

然后用户可以对顶点、边或面通过 project 关键词进行映射：

```
vertices
(
project (-1 -0.1 -1) (cylinder)
project ( 1 -0.1 -1) (cylinder)
...
);
edges
(
project 0 1 (cylinder)
...
);
faces
(
project (0 4 7 3) cylinder
...
);
```

用户可以在自带算例中通过下面的命令来更详细的了解如何使用这个功能：

```
find $FOAM_TUTORIALS -name blockMeshDict | xargs grep -l project
```

6.3.4 命名顶点、边、面及 block

blockMeshDict 可以对顶点、边、面以及 block 进行命名，这样对复杂几何的操作更加容易。其可以简单的通过 name 关键词来实现。下面的范例即为对其命名的代码：

```
vertices
(
name    v0 project (-1 -0.1 -1) (cylinder)
name    v1 project ( 1 -0.1 -1) (cylinder)
...
);
edges
(
project v0 v1 (cylinder)
...
);
```

如果给定一个名称，那么就可以用它来代替标识。例如对于上面的范例，在指定边的时候，可以给定 cylinder 关键词，而不是使用 0 和 1 标识。

6.3.5 少于 8 个顶点的 block

可以通过删掉一对或者几对顶点来创造少于 8 个顶点的 block。最常见的例子就是创建针对二维轴对称算例的楔形网格，它使用6.2.2章节提到的 wedge 类型。图6-7是一个非常简单的例子。

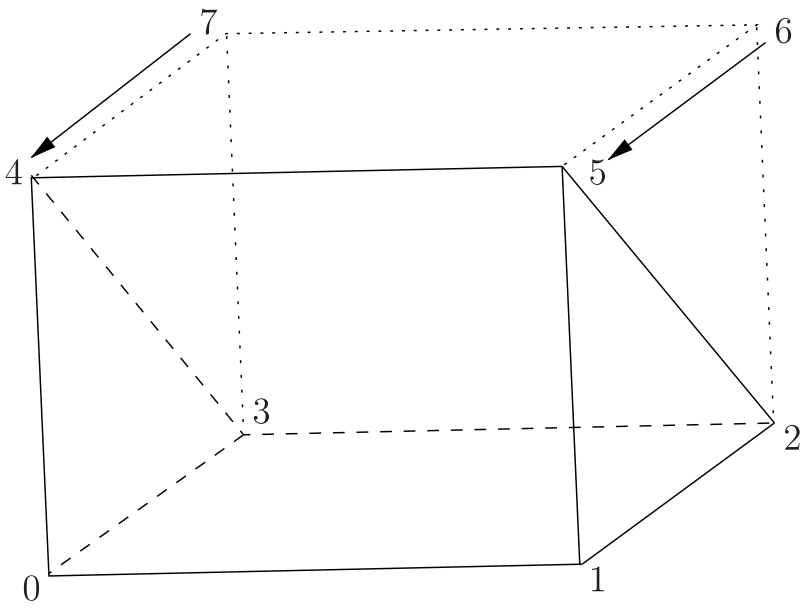


图 6-6: 使用 6 个顶点来创造楔形网格

如果用户想把顶点 7 坍塌到顶点 4，顶点 6 坍塌到顶点 5 来创造一个楔形 block，我们可以简单的把语法中的顶点 7 用顶点 4，顶点 6 用顶点 5 的替换来实现，这样就变成：

```
hex (0 1 2 3 4 5 5 4)
```

语句变更的要点即为修改需要坍塌点的顺序，例如原先六面体的 block 上部的面为 (4 5 6 7)，在楔形 block 的上部面变成了 (4 5 5 4)，实际上这是一个面积为 0 的面，它不含有任何面网格，我们可以通过查阅 ployMesh 字典文件的边界以确认。这个 patch 在 blockMeshDict 中应该被指定为 empty，并且边界条件亦应该为 empty。

6.3.6 运行 blockMesh 程序

正如3.3节我们阐述的，下面的命令行用于执行 blockMesh，它表示读取 <case> 文件夹下的 blockMeshDict 并执行：

```
blockMesh -case <case>
```

在 constant/polyMesh 文件夹中必须有 blockMeshDict

6.4 snappyHexMesh 网格生成工具

这一章节我们描述 OpenFOAM 自带的 snappyHexMesh 这个网格生成工具，snappyHexMesh 可以自动地从 STL，OBJ 文件生成六面体及多面体网格。网格依靠迭代将一个初始网格细化，并将细化后的网格变形以依附于表面。在这个过程之后可以选择是否插入网格边界层。snappyHexMesh 通过一个预先定义的网格质量标准进行控制，这个标准非常灵活，表面处理贴合功能非常强健并且可以并行运算。

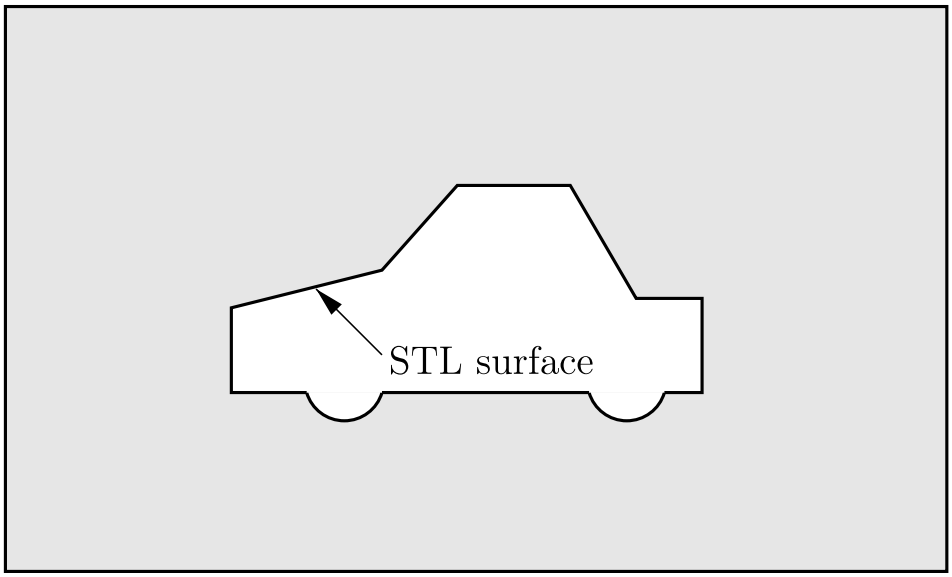


图 6-7: 一个使用 snappyHexMesh 来生成 2D 网格的几何

6.4.1 nappyHexMesh 网格生成

snappyHexMesh 生成网格的步骤可以用图6-8这个简图来表示。在这里我们生成网格的目的是在这个矩形区域网格内（阴影处）生成一个小汽车（STL 文件几何）的网格。这种网格可以用来进行空气动力学模拟。值得注意的是 snappyHexMesh 是一个 3D 网格生成工具，但这是一个二维的简图，这样做的目的是为了让用户理解起来更容易。为了运行 snappyHexMesh，用户需要提供以下内容：

- 提供一个 STL 文件格式的几何文件（binary 或者 ascii 格式）存储在 constant/triSurface 子字典中；
- 一个背景六面体网格，定义计算域范围和背景网格，一般由 blockMesh 生成，详见6.4.2节；
- snappyHexMeshDict 字典，它提供了生成网格的必要信息，位于 system 子文件夹下。

snappyHexMeshDict 文件包含了网格质量控制标准，每个标准都是相应的子字典并具有开关来控制，如下所示：

castellatedMesh 是否切割网格？（true）。

snap 是否进行网格贴合（对齐）?（true）。

addLayers 是否添加网格边界层？（true）。

mergeTolerance 自定义区域融合绝对误差（1e-6）。

geometry 所用几何文件子字典。

castellatedMeshControls 切割网格子字典。

snapControls 贴合网格子字典。

addLayersControls 边界层网格子字典。

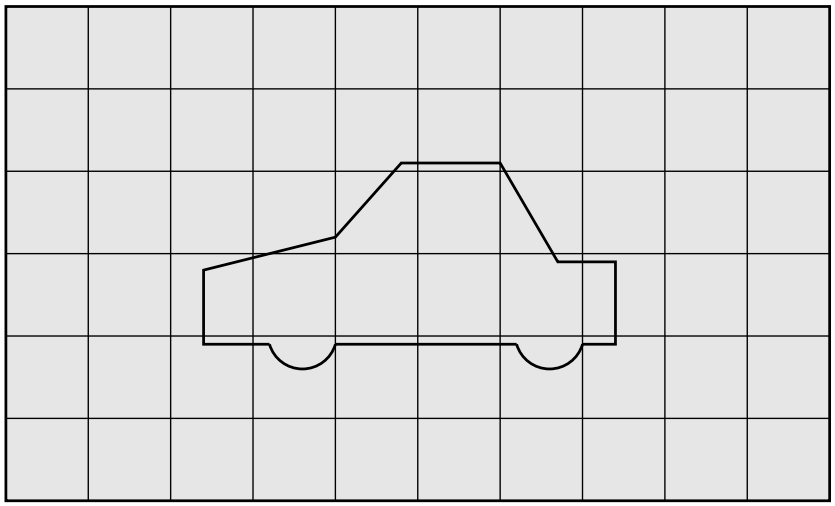


图 6-8: 用于 snappyHexMesh 程序的背景网格

meshQualityControls 网格质量控制子字典。

snappyHexMesh 所用的几何文件通过 snappyHexMeshDict 文件中的 geometry 子字典来指定，它可以是一个 STL 文件或者一个有限边界的几何体。参见以下：

```
geometry
{
    sphere1     // User defined region name
    {
        type    triSurfaceMesh;
        file    "sphere1.obj";  // surface geometry OBJ file
        regions
        {
            secondSolid             // Named region in the OBJ file
            {
                name mySecondPatch; // User-defined patch name
            }                       // otherwise given sphere1_secondSolid
        }
    }
    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;
    }
};
```

6.4.2 创建六面体背景网格

在执行 snappyHexMesh 之前，用户需要创建一个充满全部区域的六面体背景网格，见图6-9。这可以依靠 blockMesh 程序来创建。在创建背景网格的时候，它需要满足以下标准：

- 网格必须为纯六面体；
- 在随后要使用 snap 的表面附近，网格长宽高的比应该大体为 1。否则网格贴合的收敛过程会变慢，有时候还会导致失败；
- 网格单元必须至少有一个边和 STL 文件的面相交叉。例如，如果背景网格是单独的一个大网格单元是不行的。

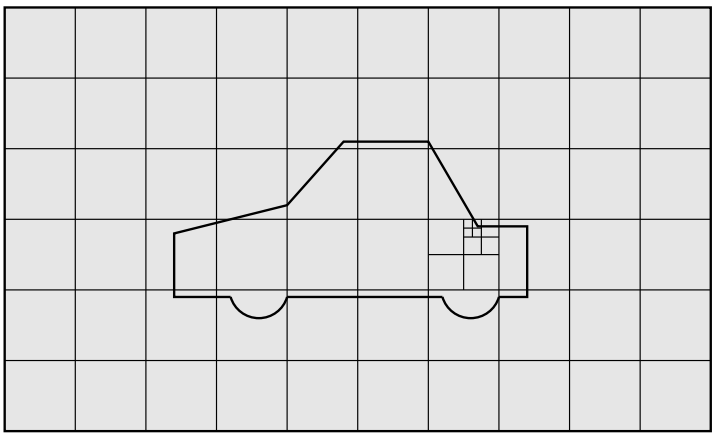


图 6-9: snappyHexMesh 程序的特征边切分过程

6.4.3 特征边和特征面网格切分

网格切分控制通过 snappyHexMesh 中的 castellatedMeshControls 子字典中的相关参数来控制。castellatedMeshControls 中相关信息如下：

locationInMesh 网格保留域（位置矢量），这个点的位置不能和网格单元的面或边重合 ((5 0 0))。

maxLocalCells 细化网格过程中每个处理器处理的最大数量网格数（1e6）。

maxGlobalCells 移除网格之前进行网格细化的最大网格数量（2e6）。

minRefinementCells 如果需要细化的网格数量小于这个值，细化停止（0）。

nCellsBetweenLevels 不同级别细化过程中的缓冲层数量（1）。

resolveFeatureAngle 在面或边的弯曲角度超过这个角度的时候应用最大等级的细化（30）。

features 特征的细化参数。

refinementSurfaces 指定细化表面。

refinementRegions 指定细化区域。

参考图6-9，分割的过程从指定的特征边开始，在 castellatedMeshControls 中的 features 下可以指定 edgeMesh 文件以及细化的程度：level，例如：

```
features
(
    {
        file "features.eMesh"; // file containing edge mesh
        level 2;               // level of refinement
    }
);
```

值得注意的是，其中的 edgeMesh 文件包含了相关的信息，并且可以通过 surfaceFeature 程序从几何文件中提取。surfaceFeature 受控于 surfaceFeatureDict 字典文件，并可以从中读取相关的控制信息。surfaceFeatureExtractDict 字典文件范例可以在 OpenFOAM 安装目录下的 $FOAM_ETC/caseDicts/surface/surfaceFeatureExtract 中找到。用户可以键入下述命令来执行程序：

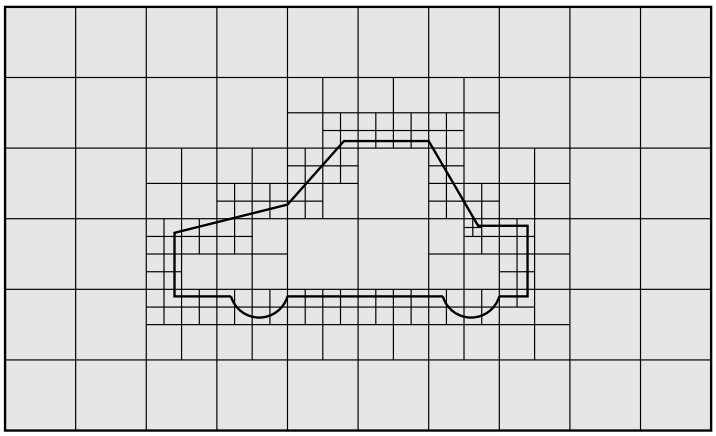


图 6-10: snappyHexMesh 程序进行的网格切分

surfaceFeature

参见图6-11中指定的面，在特征细化的过程中，选取某些网格单元以进行特征边细化并切分，需要注意的是，castellatedMeshControls 中的 refinementSurfaces 字典指定需要细化的 STL 几何面以及它们的细化的最小和最大程度 (<min> <max>)。几何体的表面应用最小的细化等级。当某些几何面或边的弯曲角度超过 resolveFeatureAngle 设定的角度的时候应用最大的细化等级。

细化的程度可以通过对 STL 文件某个区域进一步的指定而被覆盖。区域信息定义在 regions 子字典中。每个区域里的名字就是 STL 文件中定义的名字，细化程度在下一级的子字典中指定。可以参考下面的例子：

```
refinementSurfaces
{
    sphere1
    {
        level (2 2); // default (min max) refinement for whole surface
        regions
        {
            secondSolid
            {
                level (3 3); // optional refinement for secondSolid region
            }
        }
    }
}
```

6.4.4 网格移除

一旦特征边和表面切分完毕，网格移除过程开始。网格移除过程要求指定一些区域，这些区域依靠完全有界的面来封闭。需要保留的区域通过一个位置矢量（点）来指定，关键词是 castellatedMeshControls 里面的 locationInMesh。如果网格单元的 50%（粗略估计）在这个区域内，这些网格单元就会被保留。其余的网格将会被移除，参见图6-12

6.4.5 网格细化

参见图6-12，指定区域内的网格可进一步细化（黑色的阴影部位）。在 castellatedMeshControls 的 refinementRegions 子字典中指定需要细化区域的细化等级，这些区域的具体形状需要在之前的子字典中指定。Mode 是细化模式，它应用于每个区域：

inside 细化区域内的网格；

outside 细化区域外的网格；

distance 通过和表面的距离来细化，通过 levels 关键词可以在不同的距离进行不同的细化水平。

对于 refinementRegions，细化等级通过 levels 指定，格式为：(<distance><level>)。在 inside 和 outside 细化模式下，并不需要 <distance>，因此它被省略。请参考下面的例子：

```
refinementRegions
{
    box1x1x1
    {
        mode inside;
        levels ((1.0 4));
    }
    // refinement level 4 (1.0 entry ignored)
    sphere1
    {
        // refinement level 5 within 1.0 m
        mode distance;
        // refinement level 3 within 2.0 m
        levels ((1.0 5) (2.0 3)); // levels must be ordered nearest first
    }
}
```

6.4.6 表面对齐

下一个步骤就是把某些网格单元的顶点向几何对齐并贴合，以移除锯齿形网格。步骤分为：

1. 把锯齿网格的顶点移动到 STL 表面；
2. 依据新的网格点重新排布内网格点（采用松弛算法迭代处理）；
3. 寻找那些使网格质量降低的顶点；
4. 对于那些被移动的顶点，减少那些顶点的位移并重复进行第二个步骤，以确保网格质量被满足。

snappyHexMesh 里 snapControls 子字典所用的方法在表 5.9 中列出。图6-14有一个例子（这个例子的网格贴合虽然看起来有些不太实际）：

nSmoothPatch 面对应之前的面光顺迭代（3）。

tolerance 网格单元顶点和几何表面的点或特征面的距离/本地最大边长（4.0）。

nSolveIter 网格移动最大迭代数（30）。

nRelaxIter 网格贴合最大迭代数（5）。

6.4.7 网格边界层

在进行网格对齐之后，这种网格可能已经达到了计算的要求。但是它会在边界处产生一些不规则的网格单元。我们可以在边界面上增加一些依附六面体网格的附加边界层网格，参见图6-15中阴影部分的网格单元：

添加网格边界层的过程是将现有的网格单元在边界处向内收缩，然后插入边界层网格，按照如下步骤：

1. 依据一个指定的距离（边界层厚度），网格在几何面法向的方向向后映射；
2. 依据新的网格点重新排布内网格点（采用松弛算法迭代处理）；
3. 检查网格标准是否满足，没有的话减少映射量并返回到第二步。如果任何距离下都不能满足网格标准，则放弃插入边界层网格；
4. 如果网格标准满足，插入网格边界层；
5. 再次检查网格，如果不满足网格标准，去掉边界层并重新进行第二步。

边界层划分通过 snappyHexMesh 中的 addLayersControls 子字典控制（如下表所示）。用户有四种选择可以指定边界层厚度：expansionRatio, finalLayerThickness,firstLayerThickness, thickness，其中用户必须且只能指定两个。如果指定多个，前面指定的会被覆盖。

layers 边界层字典（——）。

relativeSizes 绝对边界层厚度还是相对边界层厚度（true/false）。

expensionRatio 边界层网格膨胀因子（1.0）。

finalLayerThickness 壁面最近的边界层厚度，通过 relativeSizes 设定为相对值或者绝对值（0.3）。

minThickness 边界层网格的最小厚度（相对值或者绝对值，如上）（0.25）。

nGrow 如果没有抽取点则生成指定数量的边界以连接面，这使得特征边附近的边界层添加过程更容易收敛（1）。

featureAngle 边界层自动坍塌特征角（60）。

nRelaxIter 对齐过程的最大迭代数（5）。

nSmoothSurfaceNormals 面法向光顺迭代数（1）。

nSmoothNormals 内部网格移动的光顺迭代数（3）。

nSmoothThickness 面上边界层厚度光顺数（10）。

maxFaceThicknessRatio 停止边界层增长的包裹网格率（0.5）。

maxThicknessToMedialRatio 距离中轴比的最大距离（0.3）。

minMedianAxisAngle 中轴点角度（130）。

nBufferCellsNoExtrude 新的边界层外缓冲层数（0）。

nLayerIter 最大边界层添加迭代数（50）。

nRelaxedIter 最大网格回放迭代数，当此数越界，寻找 meshQuality 中的 relaxed 子字典的相关信息并迭代（20）。

layer 子字典列举了需要添加边界层网格的几何表面，以及边界层数量。在这里我们需要使用 patch 的名字而不是几何表面的信息，因为边界层附加与现有的网格相关。参考下面的例子：

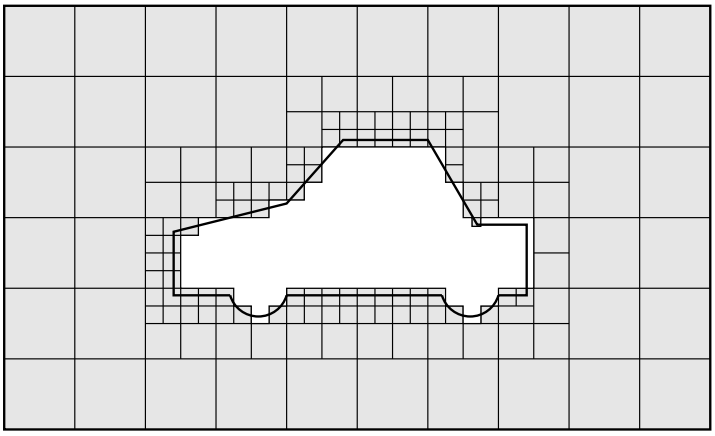


图 6-11: snappyHexMesh 的网格移除过程

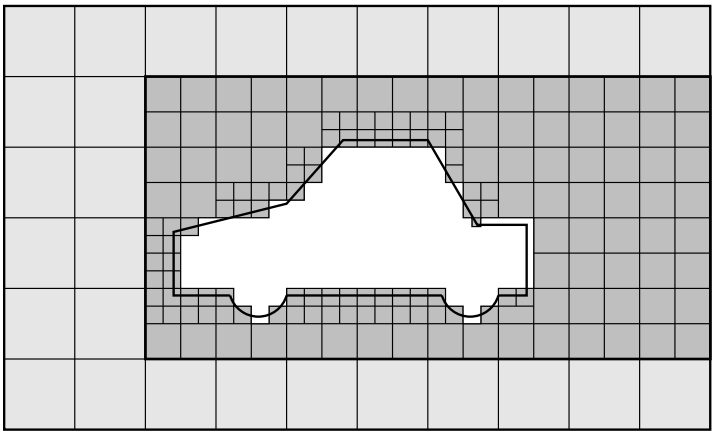


图 6-12: snappyHexMesh 的区域网格切分

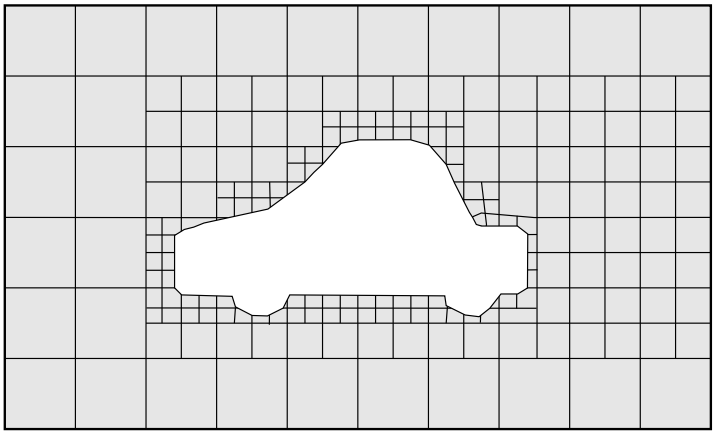


图 6-13: snappyHexMesh 的网格对齐

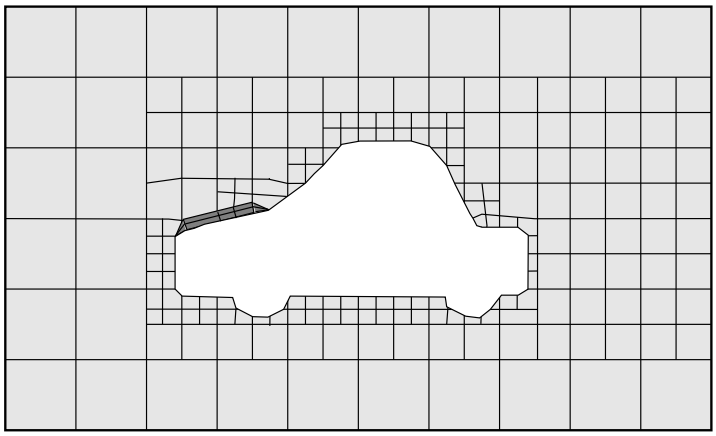


图 6-14: snappyHexMesh 的边界层添加

```
layers
{
    sphere1_firstSolid
    {
        nSurfaceLayers 1;
    }
    maxY
    {
        nSurfaceLayers 1;
    }
}
```

6.4.8 网格质量控制

网格质量控制信息在 `snappyHexMeshDict` 中的 `meshQualityControls` 子字典中控制（如下表所示）。

`maxNorOrtho` 最大非正交角，设置为 180 关闭此标准（65）。

`maxBoundarySkewness` 最大边界网格偏斜度，0 关闭（同上）（20）。

`maxInternalSkewness` 最大内部网格偏斜度，0 关闭（4）。

`maxConcave` 最大凹度，180 关闭（80）。

`minFlatness` 最小映射面积和实际面积的比值，-1 关闭（0.5）。

`minVol` 最小棱锥体体积，-1e30 关闭（1e-13）。

`minArea` 最小网格面，<0 关闭（-1）。

`minTwist` 最小面扭曲，<1 关闭（0.05）。

`minDeterminant` 最小网格行列式值；1 为理想六面体；<=0 为负体积（0.001）。

`minFaceWeight` 0→0.5（0.05）。

`minVolRatio` 0→1.0（0.01）。

`minTriangleTwist` >0 则和 Fluent 匹配（-1）。

`nSmoothScale` 每次回放的光顺数（4）。

errorReduction 网格内某些点的回放量（0.75）。

relaxed 当进行边界层附加的时候，如果 nRelaxedIter 越界，可以通过在本子字典指定其他的网格标准以覆盖（relaxed ….）。

6.5 网格转换

用户可以使用其它软件包生成的网格并转换成为 OpenFOAM 可识别的格式。表 3.6 列举了可用的网格转换程序。这一章节我们讨论如何使用下面这些转换器：

fluentMeshToFoam 读取 fluent.msh 网格文件，它对于 2D，3D 算例都适用[3]。

starToFoam 读取 STAR-CD/PROSTAR 网格文件。

ambitToFoam 读取 GAMBIT.neu 文件。

ideasToFoam 读取 I-DEAS 网格写成 ANSYS.ans 格式。

cfx4ToFoam 读取 CFX 网格并写成.geo 格式。

6.5.1 fluentMeshToFoam

Fluent 的网格采用以.msh 为后缀的单一文件。文件必须写成 ASCII 格式，这并不是 Fluent 的默认设置。在 OpenFOAM 中，也可以成功的转换 Fluent 二维的 singleStream 网格。但是转换后的网格实际上是三维网格。如果我们把第三维度的前后面设置为 empty，那么就可以把这个三维网格当做拟 2D 算例实现。当读取一个二维的 Fluent 网格的时候，转换器会自动在第三维度抽取网格并添加 empty 面，并称为 frontAndBackPlanes。用户也需要了解下面一些特性：

- OpenFOAM 转换器会尽可能的捕获 Fluent 的边界条件；然而，OpenFOAM 和 Fluent 的边界条件之间并没有一个清晰地对应，因此用户在运行算例之前需要仔细检查边界条件；
- 建立一个轴对称的二维网格目前不可行，但是这可以被植入；
- 不支持附加材料属性的网格。如果存在多个流体域，它们会被抓换成一个单一 OpenFOAM 网格文件；如果存在固体域，转换器会直接忽略[4]；
- Fluent 允许用户定义一个网格内的内部面。比如这个实体面从属于两个网格。OpenFOAM 不允许这种操作，如果这样的面存在，OpenFOAM 直接忽略；
- 目前不支持嵌入式界面和求精树。

要转换一个 Fluent.msh 网格文件，首先需要创建一个 OpenFOAM 算例；在 system 文件夹下，需要包含 controlDict 文件。然后用户需要在终端键入以下命令：

```
fluentMeshToFoam <meshFile>
```

meshFile 是网格文件名，包括其相对路径或者绝对路径[5]。

[3]经多次试验发现 fluent3DMeshToFoam 对 3D 网格支持性更好。
[4]在多流体域计算中，比如 MRF 方法，可以通过定义不同的 cellZone 来实现。
[5]如果在算例文件下运行，直接运行 fluentMeshToFoam fluent.msh 即可

6.5.2 starToFoam

这一节描述如何将 STAR-CD 的网格转换为 OpenFOAM 可读取的网格。网格可以使用任何 STAR-CD 的软件包来生成，例如 PROSTAR, SAMM, ProAM 以及它们的其它变种。转换器接受任何 singleStream 网格但是转换器不支持的特性有：

- multiStream 网格；
- 挡板，即流体域内零厚度的面；
- partial 边界；
- 滑移交界面。

对于 multiStream 网格，可以通过将每个单独的 stream 写为单独的网格文件，并将它们在 OpenFOAM 中进行装配。

OpenFOAM 仅仅转换那些严格符合6.1节描述的网格标准的网格。对于无效的网格转换器不会运行。下面的章节我们描述在使用 STAR-CD 软件包生成网格的时候，确保生成 OpenFOAM 可识别网格的具体步骤。为了防止赘述，STAR-CD 提供的网格生成器我们简称为 STAR-CD。

6.5.2.1 转换 STAR-CD 网格的一般建议

在运行 starToFoam 转换器之前，我们强烈建议用户先运行 STAR-CD 网格检查工具。starToFoam 本身也可以显示网格中的不正确信息和警告以方便用户进一步的了解网格存在的问题。在转换之后，我们建议运行 OpenFOAM 的 checkMesh 网格检查工具。对于一个 OpenFOAM 使用的网格，有问题的网格单元应该仔细检查并修复。值得一提的是对于无效的网格 OpenFOAM 不会运行，但是有些时候可以在某些其它的环境下运行，例如当网格检验标准不是很严格的时候[6]。

对于那些由于匹配误差引起的问题，可以通过设定匹配误差来解决。然而，这种方法作用有限，并且，如果需要增加匹配误差这直接表明原始网格是不精准的。

6.5.2.2 消除多余数据

网格生成完毕之后，假定流体网格部分已经创建并且需要移除其他网格单元，我们接下来需要移除不需要的顶点，压缩网格单元边界，对顶点进行编号。这个通过以下 PROSTAR 命令来执行：

```
CEST NEWS FLUID
CEST INVE
```

其中 CSET 应该是空的，如果这个网格不是算例的网格，检查 CSET 中的网格并调整模型。如果网格单元不符合需要的条件，它们可以用 PROSTAR 命令来移除：

```
CDEL CSET
```

同样，顶点需要去除，命令如下：

[6]例如其它的 CFD 求解器

```
CSET NEWS FLUID
VSET NEWS CSET
VSET INVE
```

在去除不需要的顶点的时候，我们先需要把不需要的面收集起来然后再进行清理：

```
CSET NEWS FLUID
VSET NEWS CSET
BSET NEWS VSET ALL
BEST INVE
```

如果 BEST 不是空的，我们可以使用下面的命令移除不必要的边界面：

```
BDEL BSET
```

这样的话，模型应该仅仅包含流体网格单元以及所需的顶点和相关的边界面了。所有的边界面应该和网格的顶点相对应，如果不是这样的话，需要继续清理几何直到所有的几何清理完毕。

6.5.2.3 去掉默认边界条件

默认情况下，STAR-CD 为那些没有明确的和边界区域进行关联的边界施加壁面边界条件。余下的边界面都汇总在 default 边界区域里面，并设定为 0 边界类型。考虑到可能带来的使用错误，OpenFOAM 对这些未定义的边界面有意地不采用默认边界条件类型。这样做的原因是 OpenFOAM 不清楚那些没有关联的面是否应该被指定为默认边界条件。

因此，在网格成功转换完毕后，OpenFOAM 网格中所有的边界必须重新指定。另外，通过下面的步骤可以把 STAR 中的 default 边界转换为真实边界：

1. 使用 Wire Surface 命令参数来绘制几何；
2. 使用与 default 区域 0 相同的参数来定义一个新的边界域 10，并且将全部可见的面添加进来。这可以通过在界面操作中将区域选项开启，然后通过鼠标点选一个矩形来包络屏幕中的全部模型来完成。也可以通过输入以下 PROSTAR 命令实现：

   ```
   RDEF 10 WALL
   BZON 10 ALL
   ```

3. 接下来，我们需要从集合（set）中删除之前定义的全部边界类型：

   ```
   BSET NEWS REGI 1
   BSET NEWS REGI 2
   ...3,4,...
   ```

   然后收集和相关边界相关的顶点，再收集和顶点相关的边界面（这里需要执行两次收集操作，因为有些对象在原始集合里面）：

   ```
   BSET NEWS REGI 1
   VSET NEWS BSET
   BSET NEWS VSET ALL
   ```

```
BSET DELE REGI 1
REPL
```

这样，就给边界区域 10（位于边界区域 1 的顶部）指定了边界面。然后使用 BDEL BSET 删除边界区域。再重复以上操作。

6.5.2.4 模型重新编号

重新对模型编号并检查：

```
CSET NEW FLUID
CCOM CSET

VSET NEWS CSET
VSET INVE (空)
VSET INVE
VCOM VSET

BSET NEWS VSET ALL
BSET INVE (空)
BSET INVE
BCOM BSET

CHECK ALL
GEOM
```

PROSTAR 模型检查使用最后两个命令执行。这样可能会显示一些无法预料的错误。同时，还需要注意模型的缩放因子，因为 PROSTAR 只接受 STAR-CD 模型中的缩放因子，而不接受几何模型中的缩放因子。因此，如果缩放因子不是 1，那么需要在 OpenFOAM 中使用 scalePoints 工具调整缩放因子。

6.5.2.5 写入数据

一旦网格模型完成，替换模型中的全部积分匹配为耦合类型 1。其他全部类型将用于描述任意匹配：

```
CPSET NEWS TYPE INTEGRAL
CPMOD CPSET 1
```

计算域网格必须写入它们自己的文件中。可以通过 PROSTAR 命令实现：

```
BWRITE
```

执行后，在默认的情况下，会将边界写入一个后缀名为.23 的文件（3.0 版之前）或.bnd 文件（3.0 版以后）中。对于网格单元，使用如下命令：

```
CWRITE
```

网格单元信息会输出在后缀名为.14 的文件（3.0 版之前）或.cel 的文件（3.0 版以后）中。对于顶点，使用如下命令：

```
VWRITE
```

执行之后顶点信息会输出在后缀名为.15 的文件（3.0 版之前）或.vrt 的文件（3.0 版以后）中。默认的输出文件采用 ASCII 格式。如果网格中存在耦合，则需要输入一个后缀名为.cpl 的耦合文件，命令如下：

```
CPWRITE
```

以上三个文件输出完成后，退出 PROSTAR 或关闭文件。通过程序面板可以注意到，STAR-CD 中已经定义的的全部子模型、材料、流体属性都需要在 OpenFOAM 中重新进行定义。

PROSTAR 文件转换步骤为：首先创建一个新的 OpenFOAM 算例文件夹，把 PROSTAR 文件存放其中。同时，我们需要修改上文输出的文件扩展名，分别是.23（或 pcs）修改为. bnd，.14（或 pcs）修改为.cel，.15（或 vtx）修改为.vrt。

6.5.2.6 .vrt 文件可能的问题

.vrt 文件的输出格式采用了指定宽度的列数据，而不是自由格式。一个典型的数据行如下，其中给出了一个顶点的坐标：

```
19422   -0.105988957       -0.413711881E-02    0.000000000E+00
```

如果坐标使用科学计数法表示并且是负数，数字之间可能没有空格，例如：

```
19423   -0.953953117E-01  -0.338810333E-02    0.000000000E+00
```

starToFoam 转换程序需要使用空格来读取坐标值，因此，上面的例子无法读取。OpenFOAM 提供了一个简单的脚本程序 foamCorrectVrt 来为坐标值插入空格。通过 foamCorrectVrt 可以把上面的例子转换为下述：

```
19423   -0.953953117E-01  -0.338810333E-02    0.000000000E+00
```

foamCorrectVrt 程序应该在 starToFoam 转换器之前运行，参见如下命令：

```
foamCorrectVrt <file>.vrt
```

6.5.2.7 转换为 OpenFOAM 可用格式

现在可以用 starToFoam 来创建 OpenFOAM 所需的边界、网格以及点文件了：

```
starToFoam meshFilePrefix
```

其中 meshFilePrefix 为网格文件，需要包含全路径或者相对路径。在网格转换完成之后，需要手动更改 OpenFOAM 的 boundary 文件来设置自己的边界类型。

6.5.3 gambitToFoam

GAMBIT 网格文件的扩展名为.neu。转换 GAMBIT 网格的步骤为：首先创建一个 OpenFOAM 算例，然后在终端键入下列命令行：

```
gambitToFoam meshFile
```

其中 meshFile 为网格文件，需要包含全路径或者相对路径。

GAMBIT 网格文件没有包含相应的边界信息例如壁面、对称平面、周期面等。因此所有的边界面都被指定为 patch，用户需要在转换后手动的进行修改。

6.5.4 ideasToFoam

OpenFOAM 可以转换 I-DEAS 生成的网格文件，以 ANSYS 格式输出后网格后缀名为.ans。转换.ans 网格文件的首先需要创建一个 OpenFOAM 的算例，然后在终端键入以下命令：

```
ideasToFoam meshFile
```

其中 meshFile 为网格文件，需要包含全路径或者相对路径。

6.5.5 cfx4ToFoam

CFX 的网格文件为单一文件且后缀为.geo。CFX 的网格形式是结构块形式，即网格被指定为一系列的块，他们具备连接信息以及点位置信息。OpenFOAM 会尽可能的把他们转换并且捕获相应的边界信息。但是有些三维的边界定义，例如多孔介质、固体域等信息会在转换过程中被忽略。CFX 支持默认的 patch，在这种没有定义边界条件的 patch 中被处理为壁面。这些面在转换的过程中会被 OpenFOAM 转换器收集并处理为 defaultFaces，边界类型为 wall；当然，用户需要在转换后手动的进行修改。

跟 CFX 的二维算例相同，OpenFOAM 在转换 2DCFX 网格的时候会在第三方向添加一层厚度。如果用户希望运行 CFX 可用的 2D 算例，需要把转换后网格文件的前后面设置为 empty 类型。目前 OpenFOAM 不能转换 CFX 的二维轴对称几何文件。

要转换 CFX 支持的.geo 网格文件的步骤为：首先创建一个新的 OpenFOAM 算例，然后在终端键入以下命令：

```
cfx4ToFoam meshFile
```

meshFile 是.geo 为后缀的文件，其中要包含文件的全路径或者相对路径。

6.6 不同几何上的映射场

mapFields 工具可以把现存几何上的一个或多个场映射到另外一个几何上的场。在这两个几何之前，完全不需要有任何的类似。然而，对于那些相同的几何，mapFields 可以使用一个附加参数来简化映射过程。

在讨论 mapFields 的过程中，我们需要定义一些词语。首先，我们定义原始场为 source，被映射的场为 target。如果 source 和 target 的几何和边界类型都一样，那么我们认为这两

个场是 consistent 的。mapFields 对 startFrom/startTime 中指定的时间步文件内的场来进行映射。它从原始场指定的时间步内来读取场文件，并把它映射到被映射场相应的时间步中。

6.6.1 连续场映射

连续场映射可以非常简单的执行，在被映射（target）场的目录下，使用-consistent 附加命令来运行：

```
mapFields <source dir>[7] -consistent
```

6.6.2 非连续场映射

当几何不是连续的时候，例如图6-15所示。被映射场的 system 文件夹下需要一个 mapFieldsDict 文件。在映射的过程中，有以下规则：

- mapFields 会尽可能的把所有原始场数据映射到目标场中，在我们的例子中，所有的被映射场数据都是从原始场中映射过来的，除了那些阴影面积的区域[8]；
- 边界的场数据不会改变，除非我们在 mapFields 中进行指定。

mapFieldsDict 文件包含两个列表，它们制定映射的 patch。第一个列表是 patchMap，它指定原始场和被映射场相吻合的 patch，如图6-15所示。在其中要指定原始场和被映射场 patch 对的名字。第二个列表为 cuttingPatches，它指定被映射场中的一些 patch，这些 patch 和原始场的 patch 不吻合，因此只会映射被映射场的 patch 和原始场相交的那一部分。例如在下面这个例子中，底部左边的被映射场的 patch 仅仅和原始场的内场相交。在没有相交的被映射场保持原始数据。mapFieldsDict 字典文件如下：

```
18  patchMap        (lid movingWall);
19
20  cuttingPatches  (fixedWalls);
```

6.6.3 并行映射

原始场和被映射场如果都需要或者某个场需要并行运算，我们需要在执行 mapFields 的时候需要添加附加的命令参数：

-parallelSource 如果原始场使用并行映射[9]；

-parallelTarget 如果被映射场使用并行映射。

[7]此处 <> 在输入的时候不需要添加，如：mapFields /home/test -consistent 即可，参考本链接
[8]此处不存在原始场数据
[9]请使用 decomposePar 对把原始场进行分解，然后使用 mapFields <dir> -consistent -parallelSource 命令

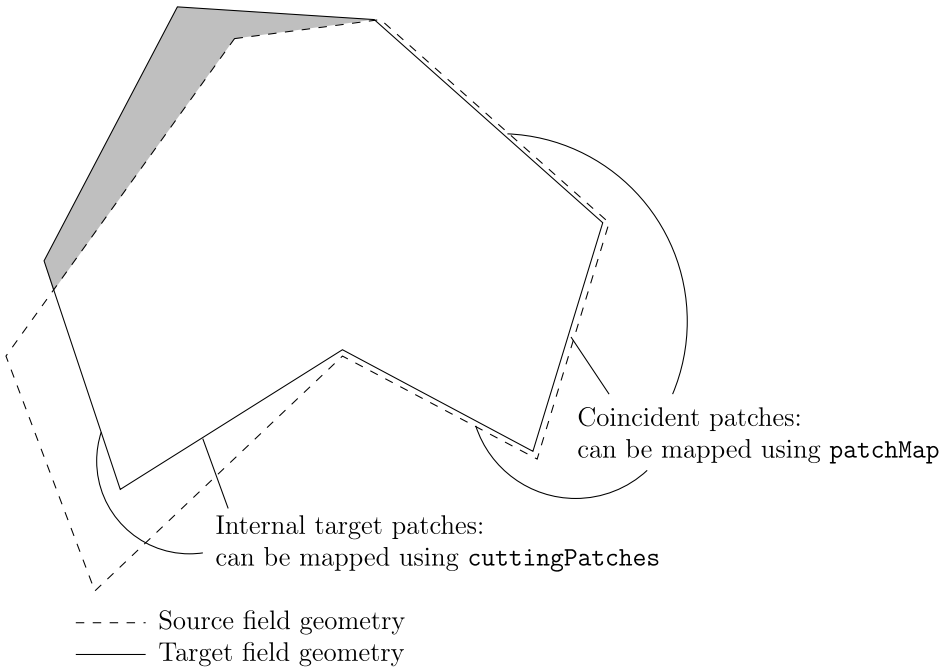

Coincident patches:
can be mapped using patchMap
Internal target patches:
can be mapped using cuttingPatches
Source field geometry
Target field geometry

图 6-15: 映射不连续场

第 7 章　后处理

这一章我们讨论 OpenFOAM 可用的后处理。OpenFOAM 提供了 `paraFoam`（依托 ParaView）来进行后处理。7.1节我们讨论这个开源可视化软件——ParaView。其它后处理方法也有提及，例如 `EnSight`，`Fieldview` 以及 `Fluent` 附带的后处理软件。

7.1　ParaView/paraFoam

OpenFOAM 所提供的后处理软件，是建立在 ParaView 基础之上的插件。插件编译成为下列两个库：`PVFoamReader` 和 `vtkPVFoam`，它们基于 ParaView-5.4.0。虽然最新的 ParaView 包也可以使用，但我们建议使用 ParaView-5.4.0[1]。关于 ParaView 的更多信息请查阅本链接

ParaView 使用 `Visualisation Toolkit`(`VTK`) 来处理数据和渲染工具，它可以读取任何 VTK 数据。OpenFOAM 包含了 `foamToVTK` 程序，它可以把 OpenFOAM 的数据转换成 `VTK` 格式，这意味着任何能读取 `VTK` 的图形程序都可以对 OpenFOAM 算例来进行后处理。因此我们可以使用除了 ParaView 之外的其它后处理程序。

总的来说，我们建议使用 ParaView 作为首选后处理软件。另外，OpenFOAM 数据也可以转换为 `VTK` 格式供 ParaView 或者其它 `VTK` 图形软件使用。

7.1.1　ParaView/paraFoam 概述

`paraFoam` 由 OpenFOAM 提供的用来运行 ParaView 的插件。像任何其它的 OpenFOAM 程序一样，要么通过在算例文件夹的终端下执行单一命令，要么就使用`-case` 命令来对某个算例执行操作：

```
paraFoam -case <caseDir>
```
[2]

如图7-1，ParaView[3]会运行并打开，算例通过左侧的面板来控制：

`Pipeline browser` 列举 ParaView 打开的模块，当前选择的模块被蓝色高亮显示，激活前面的眼睛的标志，可以显示图形；

`Properties` 包含算例需要的一些必要文件和相关设置，例如时间、区域、场信息等；`Display` 面板控制视觉样式，例如颜色条；

其他 可以在 `View` 菜单中选择，比如显示网格几何以及大小等相关信息。

ParaView 以树状结构操作数据，用户可以对顶部的模块应用滤镜并创建子模块。例如：压力的云图可以是一个建立与包含压力场模块下的子模块。ParaView 的强大在于用户可以创建一系列的子模块并展示任何你想展示的图像或者动画。例如，用户可以在压力云图上随意添加一些单纯的几何文件，网

[1] 从经验来看，paraFoam 加载大数量网格算例较慢，Paraview 较快

[2] 同样，注意 <> 不需要输入

[3] OpenFOAM 的用户指南中，paraview 后处理部分并没有更新，但本章节的截图和用户运行的 paraFoam 会有很大出入，但相关操作大体一致。

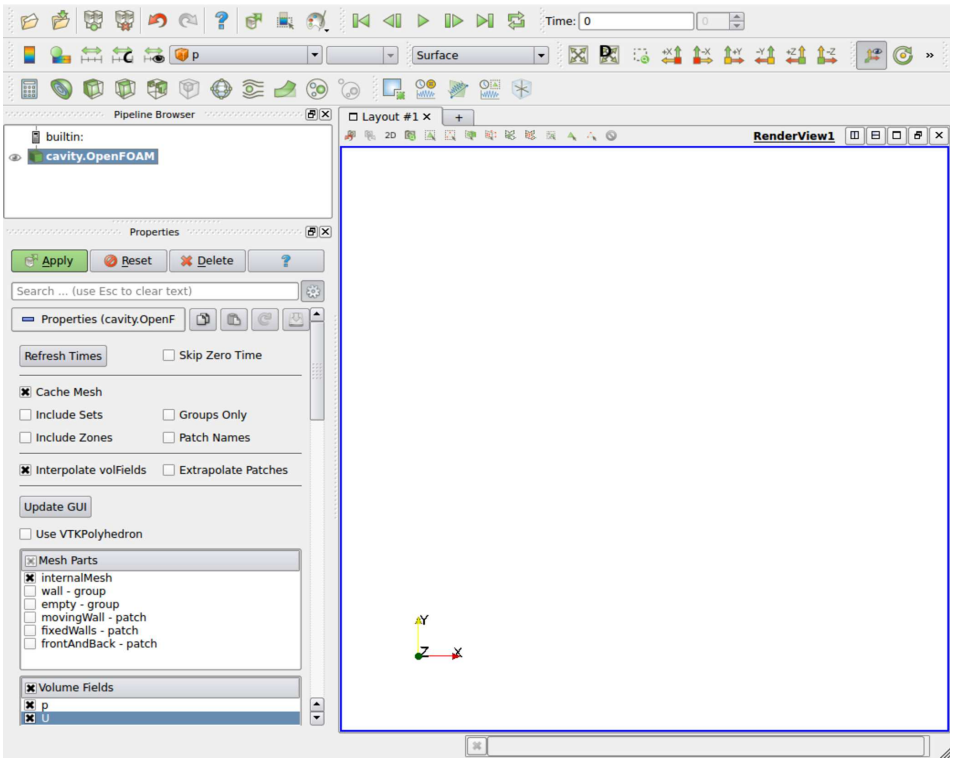


图 7-1: paraFoam 窗口

格或者速度矢量，并且在任何时候随开随关。通常的操作为用户做一些操作然后点击 Properties 面板中的绿的 Apply 按钮。其它按钮是：Reset 开关（在需要的时候可以重启 GUI）；Delete 按钮可以删除模块。

7.1.2 Properties 面板

Properties 面板包含时间步、区域、场设置等相关信息，参见图7-2。值得注意的是在当前的模块中，所有时间步的数据都加载在 ParaView 中。默认情况下，current Time Controls 以及 VCR Controls 工具栏都已经显示，它们展示的时间步数据以及相关控制。详见7.1.4节。

任何 paraFoam 中的操作，做的任何选择或者改变，用户都需要点击 Apply。Apply 按键被高亮为绿色以提醒用户已发生改变但没有应用。这样，用户就可以做一系列操作但不应用它们，这对大型算例是非常好的，可以减少数据处理负担。

有时数据文件会被更改并且 ParaView 需要读取它们，例如一个场数据被写入了新的时间步。用户需要单击 Refresh Times 按钮来读取新的时间步场数据。有的时候算例的数据发生了改变，为了读取这些更改，用户应该点击 Properties 面板顶部的 update GUI[4]按钮并应用。

7.1.3 Display 面板

Properties 面板中包含了 Display 面板，其用于对一个给定的算例进行所需的可视化设置。下面几点是非常重要的：

[4]在 paraview 最新版本中，update GUI 按钮已经缺失，用户如果想让 paraview 读取新的时间步，可以点击 refresh 按键，或者删除目前模块，重新载入。

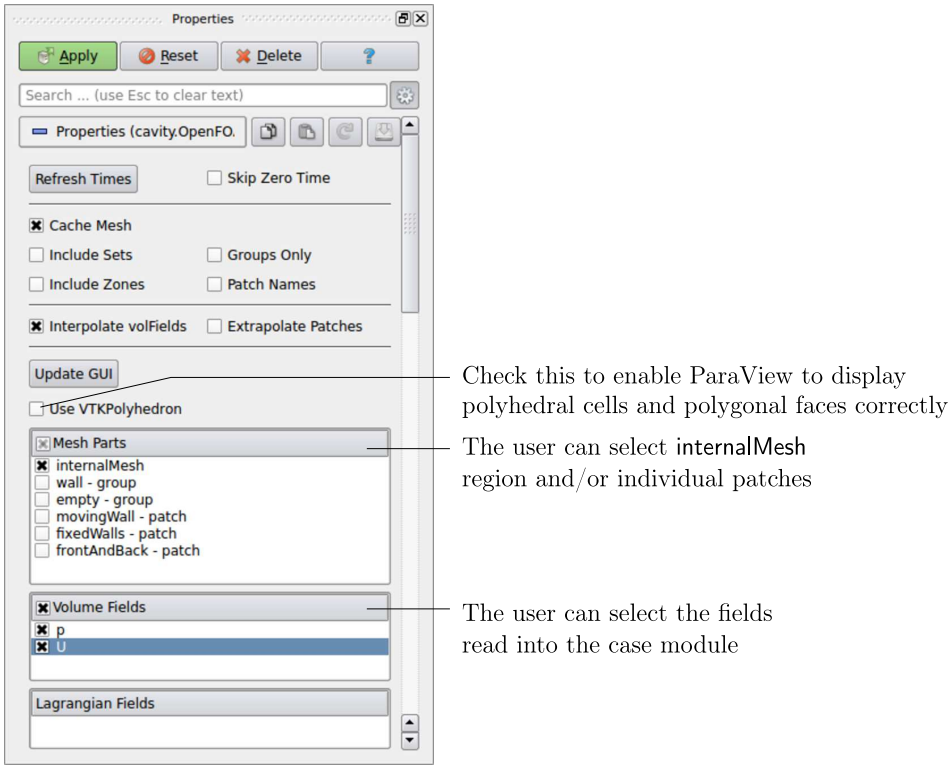


图 7-2: Properties 面板

- 对于数据场的最大值以及最小值，并不会被自动加载，用户应该在合适的情况下，特别是载入初始算例的时候选择 `Rescale` 来更新，尤其是当加载了新数据的时候；

- 点击 `Edit Color Map` 按钮，会开启一个新窗口，包含两个面板：

  1. 色系的选择可以在 `Color Scale` 面板中选择。可以根据如下步骤来选择标准的 `CFD` 红绿色系：点击 `Choose Preset`，然后选择 `Blue to Red Rainbow HSV`；
  2. `Color Legend` 面板有一个开关按钮，它用于设置色带以及色带布局控制，例如字体；

- 网格可以通过 `WireFrame` 来显示，它在 `Style` 面板的 `Representation` 菜单下进行选择；

- 几何部分，例如网格（如果选取的是 `wireframe`），可以单色显示，通过在 `Set Ambient Color` 菜单中的 `Color By` 选项中的 `Solid Color` 来调节；

- 图像可以通过 `Style` 面板中的 `Opacity` 工具来实现透明化处理，0 为全透明，1 为不透明。

7.1.4 工具栏

ParaView 将菜单栏以及面板中的几个功能整合在了一起并放置在工具栏中，位于菜单栏之下。用户可以在 `view` 菜单下的 `Toolbars` 中展示或隐藏这些工具栏。图7-4中是默认的布局设置，它们的每个名称已经标出。它们的功能可以通过图标的样子来分辨，在 `Help` 菜单中，也包含了它们的使用提示，用户可以查阅任何功能按钮的简要描述来获取相关信息。

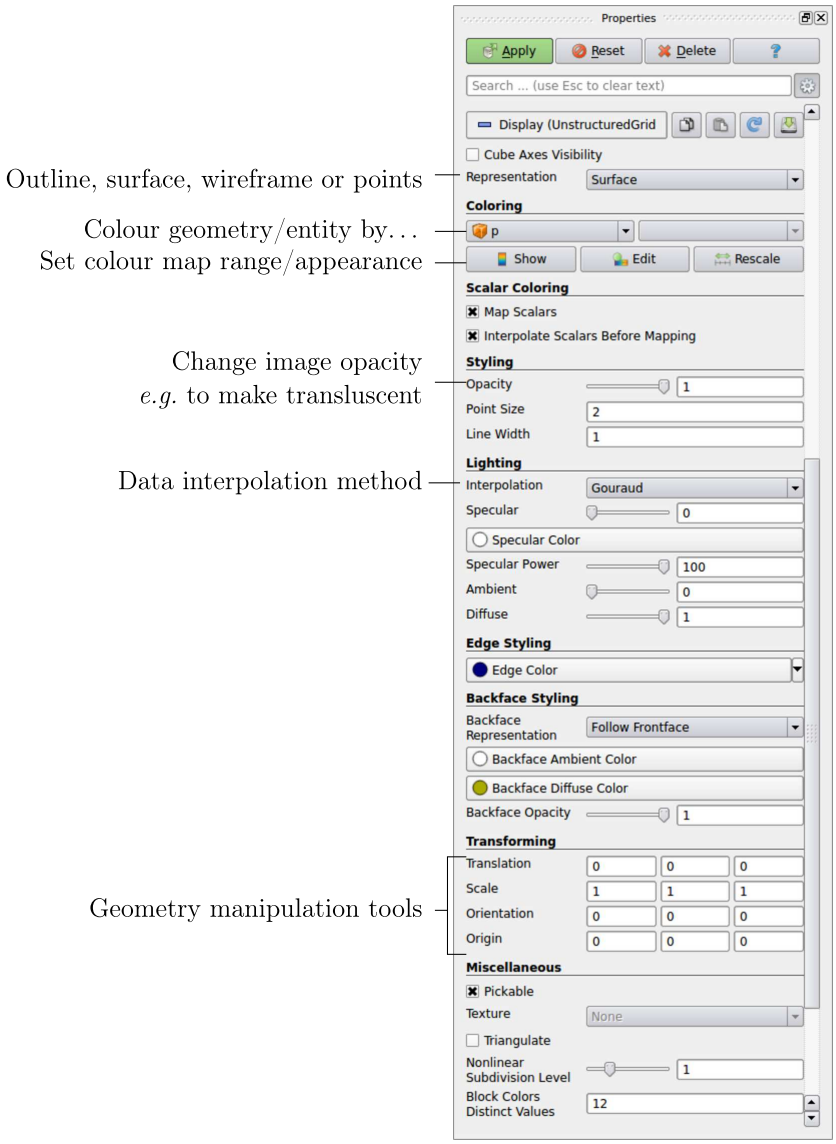


图 7-3: Display 面板

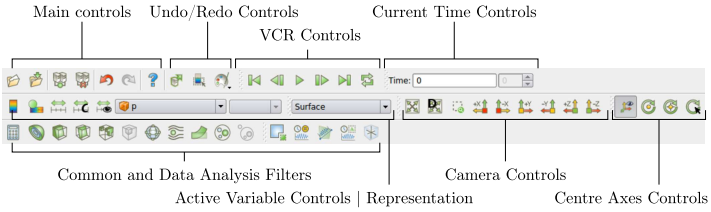


图 7-4: ParaView 的工具栏

7.1.5 效果展示

这一章节描述如何在 paraFoam 中进行设置以及如何对算例进行可视化操作。

7.1.5.1 View 设置

在 Properties 窗口的 Display 面板下的 Render View 面板中可以对 View Settings 进行设置。只有当用户单击 Properties 窗口中搜索栏旁边的齿轮按钮的时候，才会进行更进一步的设置。在这些高级的设置里面，用户可以设置背景色等相关信息。例如如果我们需要打印后处理图片的时候通常我们使用白色作为背景色。

Lights 面板中的 light kit 包含了详细的灯光颜色控制。Headlight 面板控制图像的亮度。通过调节 Headlight 工具条，可以看出设置强度为 1，会使图像更加明亮，这对等值面的展示很有帮助。

CFD 算例，尤其是 2D 算例，通常也勾选 Camera Parallel Projection 选项。其他设置包括 Cube Axes 用来设置是否显示坐标轴以及其方向、单位等。

7.1.5.2 常规设定

在 Edit 菜单栏中，有 Settings 均一设定选项，这包含了一些常规选项，例如：General，Colors，Animations，Charts 以及 Render View 菜单项。

General 面板控制 ParaView 的默认行为。特别地是，这里有一个 Auto Accept 按钮，能使 ParaView 自动的应用，而不需要每次做出操作之后再点击绿色的 Apply 按钮来应用。大部分情况下，我们不建议开启此选项。因为用户并不想在做一系列操作的时候中途每次都应用，用户只想在完成一系列操作之后应用一次。

Render View 面板包含 3 个子项：General，Camera，Server。General 面板包含 level of detail(LOD) 控制，他们用来控制图线操作以及渲染，例如转换、重置大小、旋转、重置图片分辨率等。通过合理的进行设置，用户可以在数量巨大的网格算例中快速渲染。

Camera 面板包含了 2D 以及 3D 的鼠标移动选项。它定义了鼠标左右中键的操作来实现旋转，转换，放大控制。用户可以对其进行自定义操作。

7.1.6 云图绘制

云图可以通过顶部工具栏 Filter 菜单下的 Contour 滤镜来实现。滤镜只能在一个指定的模块中应用，如果是一个 3D 的模块，云图就是一个二维的面，表示一个等值量例如等值面。云图的 Properties 面板可以用来设定一系列等值面的值。在 New range 窗口内可以查看这些值。

在下拉菜单中可以选择需要处理的场[5]。

7.1.6.1 剖面

用户可能想在一个平面创建等值线而不是创建等值面。这样的话，用户应该首先用 Slice 创建一个需要创建云图的剖面。Slice 滤镜允许用户指定一个切割面、盒子、球（指定中心和半径）。用户可以通过鼠标来操作剖面。

用户也可以在剖面上运行 Contour 滤镜来创建云图等值线。

7.1.7 矢量图

矢量图可以通过 Glyph 滤镜来绘制。滤镜对选取的 Vectors 场进行读取，在绘制的过程中，有一系列 Glyph Types（矢量类型）可以选择，Arrow 可以提供一个清晰地矢量图。通过这些矢量图形控制，用户可以实现最好的视觉效果。

Properties 面板中的其它内容主要包括 Scale Mode 菜单，通常我们选择为 Vectors，在这种情况下矢量长度和矢量大小成比例。Off 选项会使所有的矢量具有相同的长度。Set Scale Factor 可以控制矢量箭头的长度。

7.1.7.1 在网格中心绘制

默认情况下，矢量在网格顶点生成，但我们经常想在网格中心生成矢量。首先，我们在选定模块上应用 Cell centers 滤镜，然后应用 Glyph 矢量。

7.1.8 流线图

流线图首先应该通过 Stream Tracer 滤镜来创建示踪轨迹。示踪 Seed 面板指定示踪点，它们分布在一个 Line source 上或者 Point Cloud 上。用户可以查看示踪源，比如一条线。它们通常显示为白色，因此它们需要通过改变背景颜色来让它更清楚。

在 Stream Tracer 面板中，用户可以指定流线覆盖的范围、流线的长度。创建一个自己想要的美观的、数量合适的流线图需要凭借用户自己的经验。使用光顺器来进行光顺会引起误差但是会看起来更加平滑，这也会增加计算时间。

一旦流线创立，用户可以使用 tube 滤镜并把它应用在 tracer 模块上用来创建高质量的流线。这些流管跟流线重合，有的时候它们并不是非常的圆。但是有固定的边数以及一个固定半径。当边被设定之后，比如 10，流管看起来像是圆的，越圆的流管，越需要耗费资源计算。

7.1.9 输出图片

从 ParaView 输出图形最简单的方式就是在 File 菜单下选择 Save Screenshot。选择后，一个窗口会显示出来，用户可以选择输出图形的分辨率。旁边还有一个按钮，点击之后会锁定图形的宽高比。因此如果用户在一个方向上改变了分辨率。那么另一个方向的像素会自动变化。在选择分辨率之后，图像可以保存。为了达到高质量的输出，用户可以把 x 方向的像素分辨率设为 1000 或者更高，这样当图像放大在 A4 或者 US letter 文件，或者 PDF 中的时候，图像会非常锐利。

[5]根据版本的不同，有的 paraFoam 为 Value range，下拉菜单在 contoured by 右侧

7.1.10 动画输出

创建动画[6]，用户可以首先在 File 菜单中选择 Save Animation。会出现一个对话框，用户可以指定一系列例如分辨率等的相关参数。用户需要按需指定分辨率。其它的设定是每个时间步的帧数。一般这个都是 1，但可以设置更大的数值以人工加帧。这个技术在创建慢动画的时候非常有用，因为有些动画播放器的速度控制功能有限，尤其对 mpeg 格式。

点击 Save Animation 之后，另一个窗口会出现，用户可以指定文件名，比如 root，以及图片格式，点击 OK 之后，会生成一系列文件，文件名标准为：“<路径>_<图片编号>.<后缀名>”，例如文件名为 animation 的第三个文件，如果以 jpg 格式保存的话，就是“animation_0002.pdf”。图片起始于编号 0000。

一旦图片生成之后，用户可以使用其他的软件来生成动画。另一个方式是使用 foamCreateVideo 脚本来生成，详细信息请通过命令参数 -help 获取。

7.2 后处理

在 OpenFOAM-4.0 之后提供了无 GUI 的通过命令来运行的后处理工具。后处理主要包含数据再计算、提取（点提取、线提取）、算例控制以及运行时输入输出。后处理主要通过下面两种方式：

- 传统方法：计算后进行后处理；
- 运行时处理：在运行时对结果进行后处理。

两种方法各有优点。在传统方法中，用户可以在计算完成后分析获取的数据。运行时处理由于提供全数据接口的权限因此给用户提供非常大的自由，同时用户也可以在运行时进行监控。这样的话，用户可以在运行中对算例的正确性进行合理的判断。

OpenFOAM 目前提供下面 3 种具体的后处理方式：

- 每个求解器（如 simpleFoam）可以通过设置来进行运行时处理；
- 通过 postProcess 在运行结束进行后处理；
- 通过每个求解器的命令参数 -postProcess 来进行，在这种情况下，只执行后处理命令而不进行算例求解。这种方式主要是提供全数据接口。

7.2.1 postProcess

通过下面的命令参数可以获取 OpenFOAM 现存的后处理命令：

```
simpleFoam -listFunctionObjects
```

这些命令位于 FOAM_ETC/caseDicts/postProcessing 文件夹下，其也可以通过下述命令进行输出：

```
postProcess -list
```

下面我们来讨论后处理主要包含的数值计算。

[6]新版的 paraview 可以直接输出 avi 格式文件

7.2.1.1 场计算

`CourantNo` 输出库郎数。

`Lambda2` 输出 Lambda2。

`MachNo` 计算马赫数。

`Peclect` 输出 Peclet 数。

`Q` 计算速度梯度第二不变量。

`R` 计算雷诺应力张量场。

`XiReactionRate` 写入湍流火焰速度以及反应速率体标量场。

`add` 添加场。

`components` 写入矢量场的分量。

`ddt` 计算场的时间导数。

`div` 计算场的散度。

`enstrophy` 计算场的涡度拟能。

`flowType` 输出速度场类型，-1 表示有旋流；0 表示剪切流；+1 表示其他类型流动。

`grad` 计算场的梯度。

`mag` 计算场的模。

`magSqr` 计算场的模的平方。

`randomize` 对场添加随机分量，扰动量可以指定。

`scale` 对场进行缩放。

`streamFunction` 计算网格点的流函数；计算面通量。

`substract` 从某个场中减去某个场的值。

`turbulenceFields` 计算给定的湍流场。

`turbulenceIntensity` 计算湍流强度场。

`vorticity` 计算涡量场，例如速度的旋度。

`wallShearStress` 计算壁面剪切力，输出体矢量场。

`wallHeatFlux` 计算壁面热通量，输出体矢量场。

`wallHeatTransferCoeff` 计算边界场的壁面换热系数。

`writeCellCenters` 通过体矢量场的方式输出网格体心矢量。

`writeCellVolumes` 通过体标量场的方式输出网格单元体积。

`writeObjects` 输出某些特定场。

`yPlus` 计算湍流的 y+。

7.2.1.2 流率计算

`flowRateFaceZone` 计算 patch 特定面区域的流量。有可能是体积通量，也有可能是质量通量。

`flowRatePatch` 计算 patch 的流量。有可能是体积通量，也有可能是质量通量。

`volFlowRateSurface` 计算三角面的体积通量，速度的计算采用对面表面进行插值并积分。建议三角的面积足够小。

7.2.1.3 力以及力系数

`forceCoeffsCompressible` 对于可压缩求解器，通过对某 patch 上的力进行加和汇总计算升力、压力、矩系数。

`forceCoeffsIncompressible` 对于不可压缩求解器，通过对某 patch 上的力进行加和汇总计算升力、压力、矩系数。

`forceCompressible` 对于可压缩求解器，计算某 patch 上的压力以及粘性力。

`forceIncompressible` 对于不可压缩求解器，计算某 patch 上的压力以及粘性力。

7.2.1.4 提取制图

`singleGraph` 提取某一条线上的数据。

7.2.1.5 拉格朗日数据

`dsmcFields` 从 DSMC 计算结果中提取 UMean、translationalT、internalT 以及 overallT。

7.2.1.6 监控极值

`cellMax` 对于一个或多个场，输出最大的值。

`cellMin` 对于一个或多个场，输出最小的值。

`faceMax` 对于一个或多个场，输出面上最大的值。

`faceMin` 对于一个或多个场，输出面上最小的值。

`minMaxComponents` 通过非标量的形式，输出最大最小值以及位置。

`MinMaxMagnitude` 通过非标量的形式，输出最大最小值的模以及位置。

7.2.1.7 数据格式

`residuals` 对于指定场，输出每个时间步迭代的初始残差。对于矢量场，输出最大的分量。

`Time` 写入运行时间、CPU 时间、钟表时间。

7.2.1.8 压力工具

`pressureDifferencePatch` 计算两个 patch 的平均压力差。

`pressureDifferenceSurface` 将压力插值在三角面并计算平均压力差。

`staticPressure` 通过指定的密度计算动压。

`totalPressureCompressible` 计算可压缩求解器的总压。

`totalPressureIncompressible` 计算不可压缩求解器的动力总压。

7.2.1.9 探针

`boundaryCloud` 输出场在某 patch 上指定位置的值。

`interfaceHeight` 对于一系列的点，输出界面的高度。这个高度是在这个点上，界面和壁面的垂直距离。。

`internalCloud` 输出场在指定位置的值。

`Probes` 输出场在指定位置距离最近的网格单元的值。

7.2.1.10 外挂求解器

`scalarTransport` 求解标量场传输方程。

`icoUncoupledKinematicCloud` 求解拉格朗日粒子云。

7.2.1.11 可视化工具

`streamlines` 通过 VTK 格式输出流线数据。

`surfaces` 通过 VTK 格式输出切割面、等值面等。

7.2.2 运行时数据处理

如果用户打算在运行时处理数据，他们需要对算例进行相应的设置。下面是一个在出口 `outlet` 处监控流率的方法。

首先，用户需要在字典文件的 `controlDict` 中包含 `flowRatePatch` 函数，如：

```
functions
{
    #includeFunc flowRatePatch
    ... other function objects here ...
}
```

这样的话，算例会直接调用 `$FOAM_ETC/caseDicts/postProcessing` 下的相关设置。

`flowRatePatch` 需要指定 `name` 信息。用户可以通过 `foamGet` 将相关的设置拷贝进来，如：

```
foamGet flowRatePatch
```

同时编辑 name 关键词。求解器在运行时，会首先调用算例下的信息，而不是 Dicts/postProcessing 下的信息。相关的流率信息会写入到算例下的一个叫做 postProcessing 的文件夹中。

第二种方法是直接编辑 functions 代码段，如：

```
functions
{
    #includeFunc flowRatePatch(name=outlet)
    ... other function objects here ...
}
```

对于某些计算操作，可能只需要给定一个需要操作的场即可。例如，如果用户打算计算速度场的模，那么，进行下面的设置即可：

```
functions
{
    #includeFunc mag(U)
    ... other function objects here ...
}
```

这是因为该函数的参数是通过场 U 来呈现的，可参考源代码：$FOAM_ETC/caseDicts/postProcessing/fields/mag

一些函数可能要求设置很多的参数，如 forces、forceCoeffs、surfaces 函数等，对于这些函数，最好直接把算例的设置文件拷贝进来进行设置。

7.2.3 postProcess 工具

在计算完成后，用户可以通过 postProcess 工具来进行后处理。现在我们通过 pitzDaily 算例来演示如何使用 postProcess。用户首先通过下面的命令把算例拷贝到对应的文件夹，然后运行 blockMesh 生成网格，并进行 simpleFoam 运行：

```
run
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily .
cd pitzDaily
blockMesh
simpleFoam
```

计算之后，用户可以执行下面的命令，来看一下 postProcess 如何使用：

```
postProcess -help
```

简单的函数，例如求模，可以通过 -func 命令参数来执行。如果要包含文本信息，通常需要把文本包含在 (“…”) 中：

```
postProcess -func "mag(U)"
```

例如上面的命令将计算速度场的模，并写入每个时间步。类似的，如果要使用 flowRatePatch 计算流率，可以使用：

```
postProcess -func "flowRatePatch(name=outlet)"
```

考虑另一个例子。如果想计算不可压缩流动的总压，即 $p = p_0|\mathbf{U}|^2/2$，那么可以使用 totalPressureIncompressible 命令参数：

```
postProcess -func totalPressureIncompressible
```

它会返回下面的信息：

```
--> FOAM Warning : functionObject pressure: Cannot find required field p
```

这个信息表示压力场 p 并没有被挂载。同样的，速度 U 也没有被挂载。因此，用户需要键入下面的命令来挂载相关的场：

```
postProcess -func "totalPressureIncompressible(p,U)"
```

另一方面，用户也可以采用下面的方式通过 -fields 命令参数来挂载：

```
postprocess -fields "(p,U)" -func totalPressureIncompressible
```

7.2.4 求解器后处理

另一个更复杂的范例是 wallShearStress 函数：

```
postprocess -fields "(p,U)" -func wallShearStress
```

即使挂载相关的场，他也会提示下面的错误：

```
--> FOAM FATAL ERROR:
Unable to find turbulence model in the database
```

这个提示表明求解器 simpleFoam 在运行时调用的模型（如湍流模型）没有被挂载到 postProcess 程序上。这种情况表示我们需要通过调用求解器的 postProcess 命令才能挂载相关的模型。因此用户可以运行下面的命令：

```
simpleFoam -postProcess -help
```

这个命令可以列出可选用的命令参数。然后用户可以键入下面的命令：

```
simpleFoam -postProcess -func wallShearStress
```

来计算壁面剪切应力。在这种情况下用户不需要指定 U 或者 p 场，因为 simpleFoam 程序已经读取了这些场。类似的，也可以通过在算例的 controlDict 下添加 functions 外挂并挂载 includeFunc 宏来执行。

7.3 监控数据

OpenFOAM 提供几个后处理函数用来监控数据。一些工具可以生成单一的文件，其包含关于时间步的列表数据。这些数据可以用于绘制 plot。时间序列数据可以通过 foamMonitor 脚本来监控。

7.3.1 探针

在7.2.1.9节中，列举了主要的探针工具。探针工具的基本原理是用户提供一系列的监控点，然后探针工具将输出这些点的数据信息。Cloud 和 probe 工具的区别主要在于：

- probes 监控用户指定点附近的最近的网格单元，并输出网格单元的监控值，相关数据会写入到单一的文件中，并采用时间序列的形式，可用于绘制数据；
- boundaryCloud 和 internalCloud 会对场进行插值到监控点，对于 boundaryCloud 还会进行监控点的贴合。数据会写到单独的时间步中。

一般来说，probes 更适合监控少量监控点的值，cloud 主要用于监控大量位置信息的值。

用户可以采用7.2.3节中的 pitzDaily 算例进行学习。首先运行下面的命令拷贝 probes 的设置：

```
foamGet probes
```

然后用户可以在 probes 文件中修改其想监控的位置点信息：

```
13  #includeEtc "caseDicts/postProcessing/probes/probes.cfg"
14
15  fields (p U);
16  probeLocations
17  (
18      (0.01 0 0)
19  );
```

然后，在算例的 controlDict 文件中添加下面的宏：

```
functions
{
    #includeFunc probes
    ... other function objects here ...
}
```

当用户运行 simpleFoam 的时候，相关的值将会写入在 postProcessing/probes/0 文件夹中的 p 和 U 文件中。

7.3.2 提取数据

singleGraph 函数主要用来提取数据。我们还是使用 pitzDaily 算例来进行演示：

```
foamGet singleGraph
```

接下来需要编辑提取线的起始点和终止点。下面是一个范例，其指定一条垂直的线，长度为 0.05m。

```
14  start (0.01 0.025 0);
15  end (0.01 -0.025 0);
16  fields (U p);
17
18  // Sampling and I/O settings
19  #includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"
20
21  // Override settings here, e.g.
22  // setConfig { type midPoint; }
23
24  // Must be last entry
25  #includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
```

最后，需要在算例的 controlDict 文件中进行宏设置：

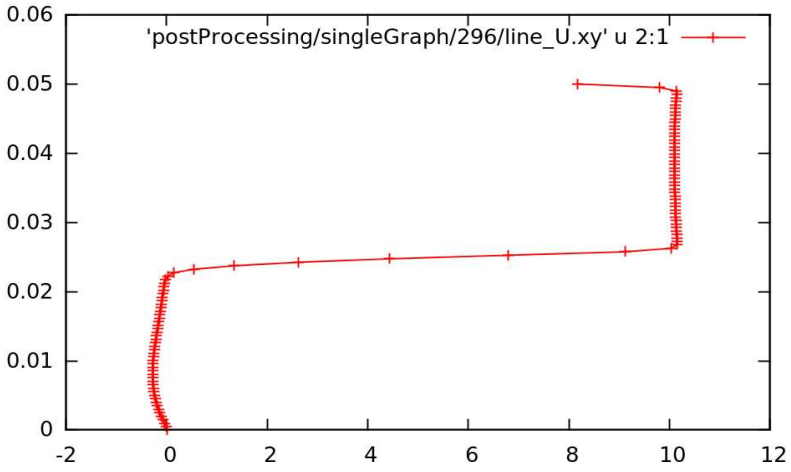


图 7-5: 采用均一提取绘制的 x 方向速度图（x=0.01）

```
functions
{
    #includeFunc singleGraph
    ... other function objects here ...
}
```

然后即可运行 simpleFoam。用户可以尝试通过附加命令参数-postProcess 来运行。运行之后，会生成 postProcessing/singleGraph 文件夹，其中包含时间步文件。用户可以使用 gnuplot 绘制第 296 时间步下的数据：

```
gnuplot
gnuplot> set style data linespoints
gnuplot> plot "postProcessing/singleGraph/296/line_U.xy" u 2:1
```

它会生成图7-5。其中图形的设置文件位于 $FOAM_ETC/caseDicts/postProcessing/graphs 中。下面是 sampleDict.cfg 文件的一些必要信息：

```
 9  interpolationScheme cellPoint;
10
11  setFormat raw;
12
13  setConfig
14  {
15      type lineuniform; // linecell, lineCellFace
16      axis distance; // x, y, z, xyz
17      nPoints 100;
18  }
```

上面的设置表示均一提取 100 个点的信息。有些情况下，用户可能打算提取每个网格单元中点的值，这可以通过下面的方式来指定：

```
14  start (0.01 -0.025 0);
15  end (0.01 0.025 0);
16  fields (U p);
17
18  // Sampling and I/O settings
19  #includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"
20
21  // Override settings here, e.g.
22  setConfig
23  {
24      type lineCell;
25      axis y;
26  }
27
28  // Must be last entry
29  #includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
```

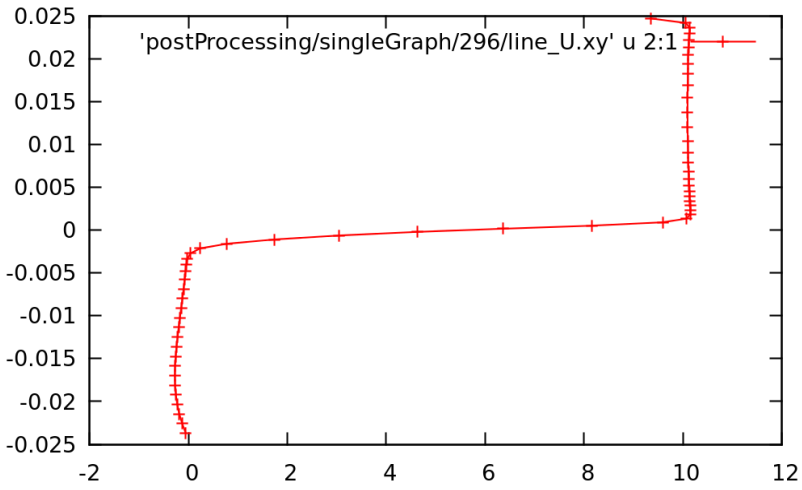


图 7-6: 采用中点提取绘制的 x 方向速度图（x=0.01）

上述文本中的 `setConfig` 部分会覆盖 `sampleDict.cfg` 中的设置。运行 `simpleFoam` 后生成的图片由图7-6所示。

7.3.3 提取可视化数据

`surfaces` 和 `streamlines` 函数可用来生成可视化文件。`pitzDaily` 文件中已经设置好了 `streamlines` 的信息。如果用户想设置 `surfaces` 来提取数据，可以通过下面的命令来拷贝过去：

```
foamGet surfaces
```

这个文件可以用来设置各种的提取面如切割面、等值面、边界面等。我们可以按照下面的设置，来提取 `pitzDaily` 几何 z 方向的一个切割面：

```
17  #includeEtc "caseDicts/postProcessing/visualization/surfaces.cfg"
18
19  fields (p U);
20
21  surfaces
22  (
23      zNormal
24      {
25          \$cuttingPlane;
26          pointAndNormalDict
27          {
28              normalVector \$z;
29          }
30      }
31  );
```

同样的，用户需要按照前述的方法将其通过 `includeFunc` 宏将其包含在 `controlDict` 文件中。另一种方式是，用户通过下面的命令来执行后处理命令：

```
simpleFoam -postProcess -func surfaces
```

上述命令会在 `postProcessing/surfaces` 文件夹下生成一系列时间序列文件，其中包含了对应的 `VTK` 格式结果。用户可以首先打开 paraview，选择 `File` 菜单中的 `Open` 来打开某个时间步下的 `vtk` 文件格式（如图7-7所示）。

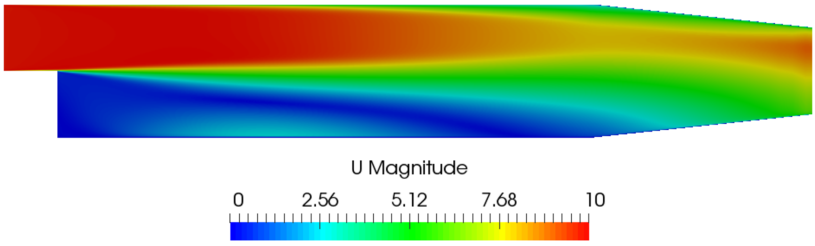


图 7-7: 速度切割面

7.3.4 动态监控数据

上文中提到的 probes 这种工具生成的数据适合绘图。但在有些情况下，用户打算在求解运行时对数据进行监控。在这种情况下，用户可以求助于 foamMonitor 脚本。用户可以通过 -help 命令参数来熟悉 foamMonitor 的作用。

首先，我们在 controlDict 文件中包含 residuals 函数：

```
functions
{
    #includeFunc residuals
    ... other function objects here ...
}
```

默认的残差监控场为 p 和 U。如果用户打算监控其他场的残差，可参考上文讨论的方式拷贝 residuals 文件到 system 目录下，编辑 fields 关键词信息。类似的所有文件均在 $FOAM_ETC/caseDicts 文件夹下。用户也可以通过下面的命令来查找 residuals 文件的位置：

```
foamInfo residuals
```

同时通过下面的命令复制设置文件：

```
foamGet residuals
```

建议用户在运行之前，删除之前留存的 postProcessing 文件夹防止多重信息。可以通过下面的命令来进行（同时后台运行 simpleFoam）：

```
rm -rf postprocessing
simpleFoam > log &
```

然后，用户可以运行 foamMonitor，并附加 -l 命令参数，其将在 y 轴上调用 log 坐标轴：

```
foamMonitor -l postProcessing/residuals/0/residuals.dat
```

如果用户在求解器结束之前键入上面的命令，其会看到图7-8所示的实时残差图。

7.4 第三方后处理

OpenFOAM 内置了下面的应用可以将 OpenFOAM 数据转换为其他软件的格式，并用其他软件来进行后处理。对于 EnSight，OpenFOAM 还附加了一个读取模块，其将在下节进行描述。

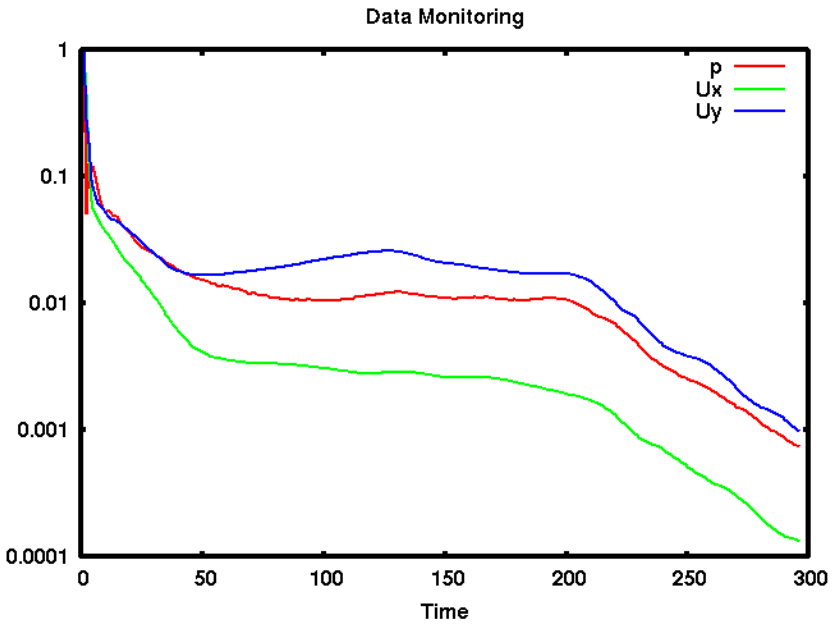


图 7-8: foamMonitor 生成的残差图

`foamDataToFluent` 将 OpenFOAM 结果转换为 `Fluent` 格式。

`foamToEnsight` 将 OpenFOAM 结果转换为 `EnSight` 格式。

`foamToEnsightParts` 将 OpenFOAM 结果转换为 `EnSight` 格式；并为每个区域创建 `Ensight` 数据。

`foamToGMV` 将 OpenFOAM 结果转换为 `GMV` 可读取的格式。

`foamToTecplot360` 将 OpenFOAM 结果转换为 `tecplot` 可读取的格式。

`foamToTetDualMesh` 将 `polyMesh` 转换为 `tetDualMesh`。

`foamToVTK` 将 OpenFOAM 结果转换为 `VTK` 格式。

`smapToFoam` 转换 `STAR-CD SMAP` 格式到 OpenFOAM 可读取的格式。

7.4.1 使用 EnSight 进行后处理

OpenFOAM 算例可以使用 `EnSight` 来进行后处理，它可以采用以下两种方法：

- 使用 `foamToEnsight` 程序把 OpenFOAM 的数据转换为 `EnSight` 格式；
- 使用 `ensight74FoamExec` 插件直接读取 OpenFOAM 数据。

7.4.1.1 转换数据为 EnSight 格式

`foamToEnsight` 把 OpenFOAM 数据转换为 `Ensight` 的文件格式。对于给定的算例，`foamToEnsight` 可以像其它程序一样来执行。`foamToEnsight` 在算例目录下创建 `Ensight_Case` 文件夹并删除之前存在的 `Ensight` 文件。然后转换器读取所有时间步的数据

并写数据[7]。每个数据文件都采用 Ensight_nn.ext 的规则来命名。nn 是从 1 开始的计数指标，第一个时间步为 1，第二个为 2，以此类推。ext 是文件扩展名，依据场信息的不同而命名。例如 T 就是温度场，mesh 就是网格。一旦数据转换完毕，EnSight 可以通过以下步骤读取数据：

1. 从 EnSight 的图形界面读取数据：用户应该在 File 菜单下选择 Data(Reader)；
2. 在 File 窗口读取数据，并高亮显示 EnSight_Case；
3. 依据 EnSight 的默认设置，Format 选择器应该设定为 Case；
4. 然后用户点击 Case 并 Okay。

7.4.1.2 ensightFoamReader 插件

除了读取 EnSight 的标准格式数据，EnSight 还可以使用用户自定义插件来读取其它格式的数据。OpenFOAM 自带供其使用的 ensight74FoamExec 插件，它被编译进入 OpenFOAM 库并成为 libuserd-foam 库。EnSight 可以使用这个库，这意味着它在现存的文件系统下必须能够调用这个库。详见下述。

为了运行 EnSight 插件，需要设置一些环境变量。我们可以这样进行设置：在 $WM_PROJECT DIR/etc/apps/ensightFoam 文件夹中的 bashrc 文件中进行设定。而 EnSight 有关的环境变量则以 $CEI 或者 $ENSIGHT7_ 为前缀，其它的详见表7-1。在标准的设置下，只有 $CEI_HOME 需要手动设置 EnSight 的安装路径。

使用 EnSight 插件来处理数据的主要困难在于 EnSight 希望算例的数据是一个单独的文件，而不是 OpenFOAM 那种各自的目录。因此在下面的叙述中，用户应该特别注意算例的选择，因为 EnSight 不允许选择文件夹。

环境变量	描述和选项
$CEI_HOME	EnSight 的安装路径例如/usr/local/ensight，默认添加。
$CEI_ARCH	平台架构，和 $CEI_HOME/ensight74/machines 中的默认设置（linux_2.4，sgi_6.5_n32）相对应。
$ENSIGHT7_READER	Ensight 寻找自定义 libuserd-foam 库的路径，缺省设置为 $FOAM_LIBBBIN。
$ENSIGHT7_INPUT	默认设置为 dummy。

表 7-1: EnSight 的环境变量和设置

1. 从 EnSight 的图形界面开始，用户应该在 File 菜单下选择 Data(Reader)；
2. 用户在 Format 菜单下选择 OpenFOAM；如果没有的话，就是设置有问题，请重新设置环境变量；
3. 用户应该在 File Selection 窗口下找到它们的算例，并把 Directories 的控件高亮，点击（Set）Geometry；
4. 用户应该在 File Selection 窗口下找到它们的算例，并把 Directories 的控件高亮，点击（Set）Geometry；
5. 用户可以点击 Okay，EnSight 会开始读取数据；

[7]原文此处有重复，故省略

6. 数据读取时，会出现一个 Data Part Loader，问你读取哪些数据。用户应选择 Load all；
7. 在 EnSight 视窗中，当展示网格的时候，用户应关闭 Data Part Loader 窗口，因为有些 EnSight 的特性，在这个窗口打开的时候不会展示。

第 8 章　模型和物理特性

OpenFOAM 包含了很多针对具体问题的求解器。每个求解器所用的方程和算法都不相同。因此用户需要在最初的时候针对不同的例子，对使用哪些模型进行决定。可选择的求解器在3.5节中列出。求解器代码决定了这个算例需要定义的物理参数，其中有一些模型的参数用户可以在运行之后，通过 constant 字典中的字典文件进行更改。这一章节我们讨论常见的模型以及运行之前需要定义的参数。

8.1　热物理模型

热物理模型跟能量、热能等物理特性有关。使用 thermoPhyscial 模型的求解器会读取 thermophy-sicalProperties 字典。OpenFOAM 中，热物理模型通过压力-温度体系而建立，通过这个体系来求解其它物理参数值。一个必须设定的关键词是 thermoType，它指定运行中的热物理模型。OpenFOAM 使用 C++ 模板预定义了大量的热物理模型库。热物理模型从第一层开始，这一层定义了基本的状态方程。然后在这一层的基础上添加附加模型。下面的 thermoType 代码即为这种多层结构的范例：

```
{
  thermoType
  {
    type hePsiThermo;
    mixture pureMixture;
    transport const;
    thermo hConst;
    equationOfState perfectGas;
    specie specie;
    energy sensibleEnthalpy;
  }
}
```

关键词指定了热物理模型，例如上述的关键词中，transport 中的 const 指定常粘度模型（粘度不变，热导率不变），equationOfState 中的 perfectGas 指定理想气体。另外，通过 energy，用户可以指定用于求解的能量形式以及热力学特性。下面的章节我们讨论 thermoType 热物理模型库。

8.1.1　热物理模型库以及混合模型

每个调用热物理模型库的求解器均会构建一个具体的热物理模型类。这些类列举如下：

psiThermo 固定组分、基于可压缩性 $\varphi = (RT)^{-1}$ 的热物理模型库，R 为理想气体常数，T 为温度。调用 psiThermo 的求解器主要为可压缩求解器，例如 rhoCentralFoam、uncoupledKinematicParcelFoam 以及 coldEngineFoam。

rhoThermo 固定组分、基于密度 ρ 的热物理模型库，调用 rhoThermo 的求解器主要为传热类求解器，例如 buoyantSimpleFoam，buoyantPimpleFoam、rhoPorousSimpleFoam、twoPhaseEulerFoam 以及 thermoFoam。

psiReactionThermo 基于可压缩性 φ 的附加反应的热物理模型库。调用 psiReactionThermo 的求解器主要为燃烧求解器，例如 sprayFoam、engineFoam、fireFoam、reactingFoam，以及一些拉格朗日求解器，例如 coalChemistryFoam。

psiuReactionThermo 基于未燃气体可压缩性 φ_u 的反应混合热物理模型库。调用 psiuReactionThermo 的求解器主要为燃烧求解器，这些求解器的物理模型基于层流火焰速度以及回归变量，如 XiFoam、XiEngineFoam 和 PDRFoam。

rhoReactionThermo 基于密度 ρ 的反应混合热物理模型库。调用 rhoReactionThermo 的求解器主要为 chtMultiRegionFoam 以及一些燃烧求解器，如 chemFoam、rhoReactinFoam、rhoReactingBuoyantFoam，以及一些拉格朗日求解器，如 reactingParcelFoam 和 simpleReactingParcelFoam。

multiPhaseMixtureThermo 多相流热物理模型库。调用 multiPhaseMixtureThermo 的求解器主要为可压缩多相界面捕获求解器，如 compressibleInterFoam、compressibleMultiphaseInterFoam。

type 关键词用来指定具体的热物理模型库，可选的有：

- hePsiThermo：调用 fluidThermo,fluidReactionThermo 以及 psiThermo 的求解器需指定；
- heRhoThermo：调用 fluidThermo,fluidReactionThermo 以及 multiphaseMixtureThermo 的求解器需指定；
- heheuPsiThermo：调用 psiuReactionThermo 的求解器需指定。

mixture 关键词指定混合组分。无反应的热物理模型库通常使用 pureMixture，也即固定组分。当指定 pureMixture 的时候，相关的热物理模型系数在 mixture 子字典中指定。

对于变组分且发生化学反应的混合模型，可以使用 reactingMixture。组分和反应需要在化学反应相关文件中被定义，也可以通过使用 foamChemistryFile 来指定相关文件来定义。reactingMixture 模型需要对每个组分进行指定（例如：O_2，N_2）且对每个组分指定相关的热物理模型系数。

对于基于层流火焰速度和回归变量的燃烧，相关组分可能会是一个组分集，例如：fuel、oxidant、burntProducts。这种燃烧模型可选的混合模型为 homogeneousMixture，inhomogeneousMixture 以及 veryInhomogeneousMixture。

其他可变组分的模型还有例如：egrMixture、multiComponentMixture 以及 singleStepReacting-Mixture。

8.1.2 传递模型

传递模型需要计算动力粘度 μ，热导率 κ，扩散率（用于内能方程或者焓方程中）α。当前可选的 transport 模型有：

const 粘度 μ 为常数，普朗特数由公式：$P_r = C_p\mu/\varepsilon$ 来计算，其中需要指定 mu 和 P_r；

sutherland 通过温度 T，sutherland 系数 A_s，sutherland 系数 T_s 的函数来计算粘度 μ。需要指定 As 和 Ts；计算公式为：

$$\mu = \frac{A_s\sqrt{T}}{1+T_s/T} \tag{8-1}$$

polynomial 从一个可以指定任意阶数的函数通过温度 T 来计算粘度 μ 和热导率 ε：

$$\mu = \sum_{i=0}^{N-1} a_i T^i \tag{8-2}$$

logPolynomial 从任意阶数依据 $\ln(T)$ 来计算 $\ln(\mu)$ 和 $\ln(k)$：

$$\ln(\mu) = \sum_{i=0}^{N-1} a_i (\ln T)^i \tag{8-3}$$

icoTabulated 使用粘度和热导率的非均匀列表数据作为温度函数。

WLF 计算 $\ln(\mu)$ 和 $\ln(\kappa)$ 作为 $\ln(T)$ 的函数由任意阶 N 的多项式; 通过指数来计算)μ，κ。

$$\ln(\mu) = \sum_{i=0}^{N-1} a_i [\ln(T)]^i \tag{8-4}$$

8.1.3 热动力学模型

热力模型参数和比热容 C_p 有关，别的相关特性可以从比热容计算而来，当前可选的 thermo 模型如下：

hConst 指定 C_p 以及 H_f 为常量。它们通过两个关键词 Cp 和 Hf 来指定。

eConst 指定 C_v 以及 H_f 为常量。它们通过两个关键词 Cv 和 Hf 来指定。

eIcoTabulated 通过插值 (T, c_p) 键值对的非均匀数据列表计算 C_v，例如：((200 1005) (400 1020));

hIcoTabulated 通过插值 (T, c_p) 键值对的均匀数据列表计算 C_v，例如：((200 1005) (400 1020));

ePolynomial 通过任意 N 阶多项式计算计算 C_v 作为温度的函数：

$$c_v = \sum_{i=0}^{N-1} a_i T^i \tag{8-5}$$

ePolynomial 通过任意 N 阶多项式计算计算 C_p 作为温度的函数：

$$c_p = \sum_{i=0}^{N-1} a_i T^i \tag{8-6}$$

ePower C_v 作为温度的指数来计算：

$$c_v = c_0 \left(\frac{T}{T_{\text{ref}}} \right)^{n_0}. \tag{8-7}$$

hPower C_p 作为温度的指数来计算：

$$c_p = c_0 \left(\frac{T}{T_{\text{ref}}} \right)^{n_0}. \tag{8-8}$$

janaf 从 JANAF 热力学表来选择参数，以温度的函数来计算。参数顺序参见表8-1。此函数在温度限 T1 和 T2 之间有效。他有两套参数需要指定，第一套的参数设定应该在温度 T_c 以上，在

T_h 以下。第二套的参数设定应该在温度 T_l 以上，在 T_e 以下：

$$c_p = R((((a_4T + a_3)T + a_2)T + a_1)T + a_0) \tag{8-9}$$

另外，一些积分常数，例如 a_5 和 a_6（对应最高温度和最低温度值）用来计算 h 和 s。

$$C_p = \sum_{i=0}^{N-1} a_i T^i \tag{8-10}$$

参见下面的这个例子:`$FOAM_TUTORIALS/lagrangian/porousExplicitSourceReactingParcel-Fo`

描述	信息	关键词
温度下限	$T_l\,(K)$	Tlow
温度上限	$T_h\,(K)$	Thigh
温度	$T_c\,(K)$	Tcommon
高温系数	$a_0, \ldots a_4$	highCpCoeffs(a0 a1 a3 a4 a4 a5⋯)
高温焓补偿	a_5	a5⋯
高温熵补偿	a_6	a6⋯
低温系数	$a_0, \ldots a_4$	lowCpCoeffs(a0 a1 a3 a4 a4 a5⋯)
低温焓补偿	a_5	a5⋯
低温熵补偿	a_6	a6⋯

表 8-1: JANAF 热力学模型系数

8.1.4 组分的量

如果要指定组分 `specie` 的量，目前只有一种选择。用户需要定义组分关键词为 `specie`，并键入下面的信息：

- `nMoles`：摩尔数。仅仅在使用反应组分均一混合回归变量燃烧模型的时候起作用，否则其为 1；
- `molWeight`：摩尔质量。

8.1.5 状态方程

下面的状态方程是可选的：

`rhoConst` 密度为常量：

$$\rho = \text{constant} \tag{8-11}$$

`perfectGas` 理想气体：

$$\rho = \frac{1}{RT} p \tag{8-12}$$

`incompressiblePerfectGas` 不可压缩理想气体：

$$\rho = \frac{1}{RT} p_{ref} \tag{8-13}$$

其中 p_{ref} 为参考压力。

`perfectFluid` 理想液体[1]：

$$\rho = \frac{1}{RT}p + \rho_0 \tag{8-14}$$

其中 p_0 为 $T = 0$ 下的密度。

`linear` 线性状态方程：

$$\rho = \varphi p + \rho_0 \tag{8-15}$$

其中 φ 为可压缩性。

`adiabaticPerfectFluid` 绝热理想气体：

$$\rho = \rho_0 \left(\frac{p+B}{p_0 B} \right)^{1/\gamma} \tag{8-16}$$

其中 ρ_0，p_0 为参考密度和压力，B 为模型常数。

`Boussinesq` Boussinesq 近似：

$$\rho = \rho_0 \left(1 - \beta \left(T - T_0\right)\right) \tag{8-17}$$

其中 β 表示体膨张率，ρ_0 表示参考温度 T_0 下的参考密度。

`PengRobinsonGas` Peng & Robinson 状态方程：

$$\rho = \frac{1}{zRT}p \tag{8-18}$$

请查阅 `FOAM_SRC/thermophysicalModels/specie/equationOfState/` 中的 `PengRobinsonGa-sI.H` 文件查看对应的 $z = z(p,T)$ 函数。

`icoPolynomial` 不可压缩多项式状态方程：

$$\rho = \sum_{i=0}^{N-1} a_i T^i \tag{8-19}$$

其中 φ 为可压缩性。

`icoTabulated` 提供不可压缩流体 (T, ρ) 键值对的数据列表，例如：rho ((200 1010) (400 980));

`rhoTabulated` 可压缩流体的统一表格数据，计算 ρ 作为 T 和 p 的函数。

`rPolynomial` 液体或固体的倒数多项式状态方程：

$$\frac{1}{\rho} = C_0 + C_1 T + C_2 T^2 - C_3 p - C_4 pT \tag{8-20}$$

C_i 为系数

8.1.6 能量方程变量选择

用户可以对能量方程求解的变量进行指定，其可为内能 e 也可以为焓 h，并可以选择是否包含热源 $\triangle h_f$。其可以通过指定 `energy` 关键字来实现。

[1]有关理想液体的一个讨论本链接

我们使用绝对能量表示包含热源的情况，显能（sensible energy）表示没有包含热源的情况。例如对于和显焓有关的绝对焓值，我们有公式：

$$h = h_s + \sum_i c_i \triangle h_f^i \tag{8-21}$$

其中，c_i 和 h_f^i 是组分 i 的分子质量和热源。大多数情况下，我们使用更容易解释化学反应带来的能量变化的显能。energy 关键词可以指定为：sensibleEnthalpy、sensibleInternalEnergy 和 absoluteEnthalpy。

8.1.7 热物理模型数据属性

每个组分都需要输入基本的热物理参数。他们必须以组分名作为关键词，例如 O2，H2O，mixture，其后是子字典，包含的参数主要有：

specie 包括组分的摩尔数，nMoles，摩尔分子质量（单位为 g/mol），molWeight；

thermoDynamics 所选热力模型的相关参数；

transport 所选传递模型的相关参数。

下面是一个组分名称为 fuel，使用 sutherland 模型和 janaf 模型的范例：

```
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 -4.84744e-08
                      1.66694e-11 -10246.6 -4.64132);
    }
    transport
    {
        As           1.67212e-06;
        Ts           170.672;
    }
}
```

下面是一个使用 const 和 hConst 模型指定 air 类的一个例子：

```
air
{
    specie
    {
        nMoles           1;
        molWeight        28.96;
    }
    thermodynamics
    {
        Cp               1004.5;
        Hf               2.544e+06;
    }
    transport
    {
        mu               1.8e-05;
        Pr               0.7;
    }
}
```

8.2 湍流模型

任何包含湍流的求解器都会读取 TurbulenceProperties 字典文件。在这个文件中包含一个关键词为 simulationType，它控制使用的湍流模型：

laminar 不使用湍流模型；

RAS 使用雷诺平均模型（RAS）；

LES 使用大涡模拟（LES）。

8.2.1 RAS 模型

如果选用 RAS 模型，那么在 RAS 子字典下需要包含下列信息：

- model：RAS 湍流模型的名称；
- turbulence：开启或关闭湍流模型；
- printCoeffs：输出或不输出湍流模型系数；
- <model>Coeffs：可选参数。指定后可以覆盖默认的湍流模型参数。

求解器可调用的湍流模型可以通过 -listMomentumTransportModels 命令参数来获取，例如：

```
simpleFoam --listMomentumTransportModels
```

输出 simpleFoam 可调用的湍流模型。可压缩湍流模型可以通过 rhoSimpleFoam 来查看。

自带算例文件中调用的 RAS 湍流模型可以通过 foamSearch 命令来查看，如：

```
foamSearch $FOAM_TUTORIALS momentumTransport RAS/model
```

用户也可以寻找某一个特定的湍流模型，如：

```
foamInfo buoyantKEpsilon
```

8.2.1.1 不可压缩 RAS 湍流模型

对于不可压缩流动，RASModel 可以选择下述模型。

LRR Launder Reece and Rodi 雷诺应力湍流模型。

LamBremhorstKE Lam and Bremhorst 低雷诺数 kEpsilon 湍流模型。

LaunderSharmaKE 不可压缩 Launder and Sharma 低雷诺数湍流模型。

LienCubicKE 三次非线性低雷诺数 kEpsilon 湍流模型。

LienLeschziner Lien and Leschziner 低雷诺数 kEpsilon 湍流模型。

RNGkEpsilon 重整化群 kEpsilon 湍流模型。

SSG Speziale Sarkar and Gatski 雷诺应力湍流模型。

ShihQuadraticKE Shih 四次代数雷诺应力 kEpsilon 湍流模型。

SpalartAllmaras Spalart Allmaras 一方程混合长外流湍流模型。

kEpsilon 标准 kEpsilon 湍流模型。

kOmega 标准 kOmega 湍流模型。

kOmega2006 标准 (2006) 高雷诺数 k-omega 湍流模型。

kOmegaSST kOmegaSST 湍流模型。

kOmegaSSTLM 基于 kOmegaSST 的 Langtry Menter 四方程转变区 SST 湍流模型。

kOmegaSSTSAS 基于 kOmegaSST 的自适应 URANS 湍流模型。

kklOmega 低雷诺数 k-kl-Omega 湍流模型。

qZeta Gibson and Dafa Alla 低雷诺数湍流模型。

realizableKE 可实现的 kEpsilon 湍流模型。

v2f Lien and Kalitzin V2F 湍流模型。

8.2.1.2 可压缩 RAS 湍流模型

对于可压缩流动，RASModel 可以选择下述模型。

LRR Launder Reece and Rodi 雷诺应力湍流模型。

LaunderSharmaKE 基于 RDT 理论的 Launder and Sharma 低雷诺数 kEpsilon 燃烧湍流模型。

RNGkEpsilon 重整化群 kEpsilon 湍流模型。

SSG Speziale Sarkar and Gatski 雷诺应力湍流模型。

SpalartAllmaras Spalart Allmaras 一方程混合长外流湍流模型。

buoyantKEpsilon 附带浮力项的标准 kEpsilon 湍流模型。

kEpsilon 标准 kEpsilon 湍流模型。

kOmega 标准 kOmega 湍流模型。

kOmega2006 标准 (2006) 高雷诺数 k-omega 湍流模型。

kOmegaSST kOmegaSST 湍流模型。

kOmegaSSTLM 基于 kOmegaSST 的 Langtry Menter 四方程转变区 SST 湍流模型。

kOmegaSSTSAS 基于 kOmegaSST 的自适应 URANS 湍流模型。

realizableKE 可实现的 kEpsilon 湍流模型。

v2f Lien and Kalitzin V2F 湍流模型。

8.2.2 LES 湍流模型

如果选择 LES，那么在 LES 子字典下需要提供一下关键词信息：

- model：LES 湍流模型名称；
- delta：δ 模型；
- <model>Coeffs：LESModel 模型系数，提供此模型会覆盖原有的参数值；
- <delta>Coeffs：δ 模型参数。

目前 tutorial 中可选的 LES 模型可以通过 foamSearch 命令来获取，如：

```
foamSearch $FOAM_TUTORIALS momentumTransport LES/model
```

8.2.2.1 不可压缩 LES 湍流模型

目前可选的不可压缩 LES 湍流模型如下：

DeardorffDiffStress Deardorff SGS 应力方程模型。

Smagorinsky Smagorinsky SGS 应力方程模型。

SpalartAllmarasDDES SpalartAllmarasDDES 湍流模型。

SpalartAllmarasDES SpalartAllmarasDES 湍流模型。

SpalartAllmarasIDDES SpalartAllmarasIDDES 湍流模型。

WALE 壁面适应大涡粘度 SGS 模型。

dynamicKEqn 动态一方程涡粘模型。

dynamicLagrangian 附加拉格朗日平均的动态一方程涡粘模型。

kEqn 一方程涡粘模型。

kOmegaSSTDES kOmega-SST-DES 模型。

8.2.2.2 可压缩 LES 湍流模型

目前可选的可压缩 LES 湍流模型如下：

DeardorffDiffStress Deardorff SGS 应力方程模型。

Smagorinsky Smagorinsky SGS 应力方程模型。

SpalartAllmarasDDES SpalartAllmarasDDES 湍流模型。

SpalartAllmarasDES SpalartAllmarasDES 湍流模型。

SpalartAllmarasIDDES SpalartAllmarasIDDES 湍流模型。

WALE 壁面适应大涡粘度 SGS 模型。

dynamicKEqn 动态一方程涡粘模型。

dynamicLagrangian 附加拉格朗日平均的动态一方程涡粘模型。

kEqn 一方程涡粘模型。

kOmegaSSTDES kOmega-SST-DES 模型。

8.2.3 模型系数

RAS 湍流模型的各个系数都存在一个默认值，它们在源代码中被指定。如果用户想更改这些默认值，它们可以在 RASProperties 文件中添加一个子字典，关键词就是相关的模型附加上 Coeffs。例如 kEpsilon 模型通过 kEpsilonCoeffs 作为关键词。如果 RASProperties 中的 printCoeffs 开关设置为 on，那么在求解器最开始运行的时候，相关的…Coeffs 字典就会被输出。用户可以简单地把输出的信息拷贝到 RASProperties 中并加以编辑。

8.2.4 壁面函数

OpenFOAM 提供了可用于边界条件的一系列壁面函数模型可供选择。可以在不同的壁面上使用不同的壁面函数。壁面函数可以通过指定 0 文件夹下的 nut 文件（湍流粘度场 ν_t）来实现。需要注意：在 OpenFOAM-3.0.0 之前，壁面函数的指定如下：不可压缩 RAS 的壁面函数通过 0/nut（ν_t）文件指定，可压缩 RAS 的壁面函数通过 0/mut（μ_t）文件指定，不可压缩 LES 的壁面函数通过 0/nuSgs（ν_{sgs}）文件指定，可压缩 LES 的壁面函数通过 0/muSgs（μ_{sgs}）文件指定，例如，一个 0/nut 文件夹：

```
18  dimensions                [0 2 -1 0 0 0 0];
19
20  internalField             uniform 0;
21
22  boundaryField
23  {
24      movingWall
25      {
26          type            nutkWallFunction;
27          value           uniform 0;
28      }
29      fixedWalls
30      {
31          type            nutkWallFunction;
32          value           uniform 0;
33      }
34      frontAndBack
35      {
36          type            empty;
37      }
38  }
```

OpenFOAM 提供了一系列壁面函数，例如：nutWallFunction,nutRoughWallFunction, nutUSpaldingWallFunctionnutkWallFunction, 和 nutkAtmWallFunction。用户可以通过下面的命令来查看壁面函数模型的完整列表：

```
foamInfo wallFunctions
```

在每个壁面函数条件下，用户可以通过指定 E、kappa、以及 Cmu 关键词来重置默认的这些系数。

在 nut 或者 mut 文件中，针对不同的 patch 选择了不同的壁面函数，用户应该在 epsilon 场中的对应的 patch 上选择 epsilonWallFunction，以及 k，q，R 场中对应的 patch 上选择 kqRwallFunction。

8.3 流变模型

在 OpenFOAM 中，求解器不仅可以求解能量/热，还可以求解不同的流变模型来获得不同的粘度 v。流变模型主要和剪切力 γ 有关，他们通过在算例目录下的 `transportProperties` 来指定。目前可选的流变模型请参考下述内容。

8.3.1 牛顿流变模型

牛顿模型假定粘度 v 为常数。粘度通过在 `transportProperties` 文件中对一个 `dimensionedScalar` 类型的 `nu` 赋值来指定，例如：

```
transportModel      Newtonian;
nu                  [ 0 2 -1 0 0 0 0 ] 1.5e-05;
```

注意：动力粘度的单位为 m^2/s。

8.3.2 Bird-Carreau 流变模型

`Bird-Carreau` 流变模型方程为：

$$v = v_\infty + (v_0 - v_\infty)\left[1 + (k\gamma)^a\right]^{(n-1)/a} \tag{8-22}$$

其中 a 的默认值为 2。下面是一个范例：

```
transportModel BirdCarreau;
BirdCarreauCoeffs
{
    nu0      [ 0 2 -1 0 0 0 0] 1e-03;
    nuInf    [ 0 2 -1 0 0 0 0] 1e-05;
    k        [ 0 0  1 0 0 0 0] 1;
    n        [ 0 0  0 0 0 0 0] 0.5;
}
```

8.3.3 Cross 幂率流变模型

`Cross` 幂率流变模型方程为：

$$v = v_\infty + \frac{(v_0 - v_\infty)}{1 + (m\gamma)^n} \tag{8-23}$$

下面是一个范例：

```
transportModel CrossPowerLaw;
CrossPowerLawCoeffs
{
    nu0      [ 0 2 -1 0 0 0 0] 1e-03;
    nuInf    [ 0 2 -1 0 0 0 0] 1e-05;
    m        [ 0 0  1 0 0 0 0] 1;
    n        [ 0 0  0 0 0 0 0] 0.5;
}
```

8.3.4 幂率模型

幂率模型的粘度和剪切力有关，并且被限制在最小值 v_{min} 和最大值 v_{max} 之间。其方程为：

$$v = k(\gamma)^{n-1}, v_{min} < v < v_{max} \tag{8-24}$$

下面是一个范例：

```
transportModel powerLaw;
powerLawCoeffs
{
    nuMax     [0 2 -1 0 0 0 0] 1e-03;
    nuMin     [0 2 -1 0 0 0 0] 1e-05;
    k         [0 2 -1 0 0 0 0] 1e-05;
    n         [0 0  0 0 0 0 0] 1;
}
```

8.3.5 Herschel-Bulkley 流变模型

Herschel-Bulkley 流变模型同时考虑了宾汉流体和幂率流变特性的行为。对于低剪切应力的工况，流体被认为是非常粘的，因此粘度为 v_0。在超出某个阈值，例如超过剪切力 τ_0 之后，粘度通过幂率模型来描述，其方程为：

$$v = \min\left(v_0, \frac{\tau_0}{\gamma} k\gamma^{n-1}\right) \tag{8-25}$$

下面是一个范例：

```
transportModel HerschelBulkley;
HerschelBulkleyCoeffs
{
    nu0     [0 2 -1 0 0 0 0] 1e-03;
    tau0    [0 2 -2 0 0 0 0] 1;
    k       [0 2 -1 0 0 0 0] 1e-05;
    n       [0 0  0 0 0 0 0] 1;
}
```

8.3.6 Casson 模型

Casson 模型主要用于血管流变学中，其需要指定最小值 v_{min} 和最大值 v_{max}。其方程为

$$v = \left(\frac{\tau_0}{\gamma}\sqrt{m}\right)^2 \quad v_{min} < v < v_{max} \tag{8-26}$$

下面是一个范例：

```
transportModel Casson;
CassonCoeffs
{
    m         [0 2 -1 0 0 0 0] 3.934986e-6;
    tau0      [0 2 -2 0 0 0 0] 2.9032e-6;
    nuMax     [0 2 -1 0 0 0 0] 13.3333e-6;
    nuMin     [0 2 -1 0 0 0 0] 3.9047e-6;
}
```

8.3.7 普适性形变率函数

用户可以通过 stranRateFunction 函数来在运行时指定粘度和形变率的关系。例如，在6.2.3.4节中，可以通过依时类边界条件进行自定义。下面是一个使用 polynomial 函数进行指定普适性形变率的范例：

```
transportModel strainRateFunction;
strainRateFunctionCoeffs
{
    function polynomial ((0 0.1) (1 1.3));
}
```