



Hydrothermal Foam Manual

Release 1.0

Zhikui Guo, Lars Rüpke, Chunhui Tao

May 12, 2020

CONTENTS

1	Introduction	3
1.1	Features	3
1.2	Information	4
1.3	Dependent package	4
2	Installation	5
2.1	Docker image	5
2.2	Build from source	7
2.2.1	Install OpenFOAM	7
2.2.2	Build HydrothermalFoam	7
3	Model description	9
3.1	Solver and equations	9
3.2	Boundary conditions	9
3.2.1	Basic boundary conditions	10
3.2.2	Customized boundary conditions for pressure	12
3.2.3	Customized boundary conditions for temperature	13
3.2.4	Coded boundary conditions	15
3.3	Properties	15
3.3.1	Transport properties	16
3.3.2	Thermophysical properties	17
4	Tutorial	19
4.1	Mesh generation	19
4.1.1	Simple mesh generation: <code>blockMesh</code>	19
4.1.2	Convert <code>gmsH</code> to OpenFOAM: <code>gmsHToFoam</code>	22
4.2	Case setup	27
4.2.1	Initial state	28
4.2.2	Constant	29
4.2.3	System	29
4.3	Set fields	34
4.4	Run Case	35
4.5	Post-processing and visualization	37
5	Cookbooks	39

5.1	Convection in a box	39
5.1.1	Hello World: 2D box	39
5.1.2	Nonuniform fixed temperature BC	46
5.1.3	Time-dependent permeability	47
5.1.4	Gmsh	49
5.1.5	3D box	53
5.1.6	Parallel computing	53
5.2	Pipe model	56
5.2.1	Two-dimensional pipe	56
5.2.2	Three-dimensional pipe	58
5.3	Single pass model	58
5.3.1	Two-dimensional single pass model	58

Bibliography		65
---------------------	--	-----------

Welcome to the [HydrothermalFoam](#) Manual! Here you will find resources for using HydrothermalFoam and examples of what it can do.

INTRODUCTION

HydrothermalFoam —combination of **hydrothermal** and **OpenFOAM** —a three dimensional hydro-thermo-transport model designed to resolve fluid flow within submarine hydrothermal circulation systems. HydrothermalFoam has been developed on the OpenFOAM platform, which is a Finite Volume based C++ toolbox for fluid-dynamic simulations and for developing customized numerical solvers that provides access to state-of-the-art parallelized solvers and to a wide range of pre- and post-processing tools. We have implemented a porous media Darcy-flow model with associated boundary conditions designed to facilitate numerical simulations of submarine hydrothermal systems. The current implementation is valid for single-phase fluid states and uses a pure water equation-of-state (IAPWS-97). We here present the model formulation, OpenFOAM implementation details, and a sequence of 1-D, 2-D and 3-D benchmark tests. The source code repository further includes a number of tutorials that can be used as starting points for building specialized hydrothermal flow models.

1.1 Features

Features of the [HydrothermalFoam](#) are summarized as following:

- **Original characteristics of OpenFOAM:** [HydrothermalFoam](#) keeps all the original characteristics of OpenFoam, for example, file structure of case, syntax of all the input files and output files, mesh, utilities, even part of variable names in the source code are kept the same. This principle has two advantages, one is that it is easy to understand and to use [HydrothermalFoam](#) if you are a OpenFoam user. The other is that it is easy to compare and understand [HydrothermalFoam](#) solver and the other standard solvers in OpenFoam.
- **Mesh:** HydrothermalFoam supports both structured regular mesh and unstructured mesh. The internal structured regular mesh tool, `blockMesh`, is recommended for new users. While `Gmsh` is also an excellent open source unstructured mesh generator, and there is a utility named `gmshToFoam` can transform `gmsh` to OpenFoam mesh.
- **Boundary conditions:** even though OpenFoam has a lot of build-in boundary conditions, we also developed some specific boundary conditions, e.g. `noFlux`, `HydrothermalHeatFlux` for the specific problem —Hydrothermal system.

1.2 Information

- License: GNU GeneralPublic License v3.0
- Software doi: [10.5281/zenodo.3755648](https://doi.org/10.5281/zenodo.3755648)
- GitLab repository: <https://gitlab.com/gmdpapers/hydrothermalfoam>
- Docker Hub repository: [zguo/hydrothermalfoam](https://hub.docker.com/r/zguo/hydrothermalfoam)

1.3 Dependent package

- OpenFOAM
- freesteam
- Gmsh

INSTALLATION

Tip: We provide a 5-minutes quick start video (on Mac OS) that can be accessed here <https://youtu.be/6czcxC90gp0>.

There are two ways to install `HydrothermalFoam`. The easiest way is installing via Docker image, see [Section 2.1](#). Another way is building from source code if user has experience of OpenFOAM installation, see section of *Build from source*.

2.1 Docker image

In order to quick start to use `HydrothermalFoam` for non-Ubuntu users, we provided a pre-compiled docker image which can be found on [Docker Hub](#) repository, named `zguo/hydrothermalfoam`. It's pretty simple to install `HydrothermalFoam` via Docker, all the steps are summarized below,

1. **Install Docker desktop** and keep it running.
2. **Pull** the docker image of `HydrothermalFoam` by running command of `docker pull zguo/hydrothermalfoam`.
3. **Install a container** from the docker image by running shell script which can be found in source code directory of `docker` (see also [Listing 2.1](#) and [Listing 2.2](#)). The directory named `HydrothermalFoam_runs` directory is a shared folder between the container and host machine.

Listing 2.1: Script for Mac OS (`installMacHydrothermalFoam.sh`)

```
1 homeInHost="${1:-$HOME}"
2 dirInContainer="/home/openfoam/HydrothermalFoam_runs"
3 dirInHost="${homeInHost}/HydrothermalFoam_runs"
4 imageName="zguo/hydrothermalfoam"
5 containerName="hydrothermalfoam"
6 # List container in docker environment
7 echo "*****"
8 echo "List of Container in docker environment:"
9 echo "*****"
```

(continues on next page)

(continued from previous page)

```

10 docker ps -a
11 echo "***** "
12 echo " "
13 echo "Creating Docker OpenFOAM container ${containerName}"
14 # Create docker container with OpenFOAM environment
15 docker run -it -d --name ${containerName} --workdir="/home/openfoam" -v=${
↵{dirInHost}:${dirInContainer} -e DISPLAY=host.docker.internal:0 ${imageName}
16 echo "Container ${containerName} was created."
17 echo "*****"
18 echo "Run the ./startMacOpenFoam script to launch container"
19 echo "*****"

```

Listing 2.2: Script for Windows (installWindowsHydrothermalFoam.ps1)

```

1 $dirInContainer="/home/openfoam/HydrothermalFoam_runs"
2 $dirInHost="$home/HydrothermalFoam_runs"
3 $imageName="zguo/hydrothermalfoam"
4 $containerName="hydrothermalfoam"
5
6 # List container in docker environment
7 echo "*****"
8 echo "List of Container in docker environment:"
9 echo "*****"
10 docker ps -a
11 echo "***** "
12 echo " "
13 echo "Creating Docker OpenFOAM container ${containerName}"
14 # Create docker container with OpenFOAM environment
15 docker run -it -d --name $containerName --workdir="/home/openfoam" -v="$dirInHost
↵":$dirInContainer $imageName
16 echo "Container ${containerName} was created."
17 echo "*****"
18 echo "Run the startWindowsOpenFOAM.ps1 script to launch container"
19 echo "*****"

```

4. **Start the container** by running command of `docker start hydrothermalfoam`.

5. **Attach the container** by running command of `docker attach hydrothermalfoam`.

The user now in a Ubuntu linux environment with precompiled HydrothermalFoam tools which located at directory of `~/HydrothermalFoam`. We recommend user run HydrothermalFoam cases in the directory of `HydrothermalFoam_runs` in the container, and then the results are synchronized in the shared directory in the host, and thus can be visualized by `ParaView`, `Tecplot` or other CFD post-processing software.

2.2 Build from source

2.2.1 Install OpenFOAM

The HydrothermalFoam v1.0 is developed based on OpenFOAM-7, which can be installed according to the installation instructions (<https://openfoam.org/download/>) given by the development team for Ubuntu Linux, Other Linux, macOS and Windows platform, respectively.

2.2.2 Build HydrothermalFoam

Once OpenFOAM is built successfully, the source code of HydrothermalFoam be downloaded from [Zenodo](#) or from [GitLab repository](#). The directory structure and components of HydrothermalFoam are shown in Fig. 2.1 and the components can be built follow three steps below,

Note: The following steps are only proper for Mac OS and Linux systems, we do not yet build HydrothermalFoam on Windows system directly. If users using ubuntu sub-system on Windows 10, the following steps could work in the sub-system.

1. **Build freesteam-2.1 library.** The freesteam project is constructed by `scons`, which is a open source software construction tool dependent on `python 2`, and based on `GSL` (GNU-ScientificLibrary). Therefore `python 2`, `scons` and `GSL` have to be installed firstly, then change directory to `freesteam-2.1` in HydrothermalFoam source code and type command of `scons INSTALL_PREFIX=$FOAM_USER_LIBBIN install` to compile freesteam library named `libfreesteam.so`. See home page of [freesteam-2.1](#) project for more details.
2. **Build libraries of customized boundary conditions and thermo-physical model.** Change directory to `libraries` and type command of `./Allmake` to compile the libraries named `libHydroThermoPhysicalModels.so`, `libHydrothermalBoundaryConditions.so`.
3. **Build solver of HydrothermalSinglePhaseDarcyFoam.** Change directory to `HydrothermalSinglePhaseDarcyFoam` and type command of `wmake` to compile the solver named `HydrothermalSinglePhaseDarcyFoam`.

Note: All the library files and executable application (solver) file will be generated in directories defined by OpenFOAM's path variables of `FOAM_USER_LIBBIN` and `FOAM_USER_APPBIN`, respectively. If build HydrothermalFoam in Mac OS, the extension of the library files is `.dylib`, please make a symbolic links. For example, see following command for `libHydroThermoPhysicalModels.dylib`

```
ln -s $FOAM_USER_LIBBIN/libHydroThermoPhysicalModels.dylib $FOAM_USER_LIBBIN/
↳ libHydroThermoPhysicalModels.so
```

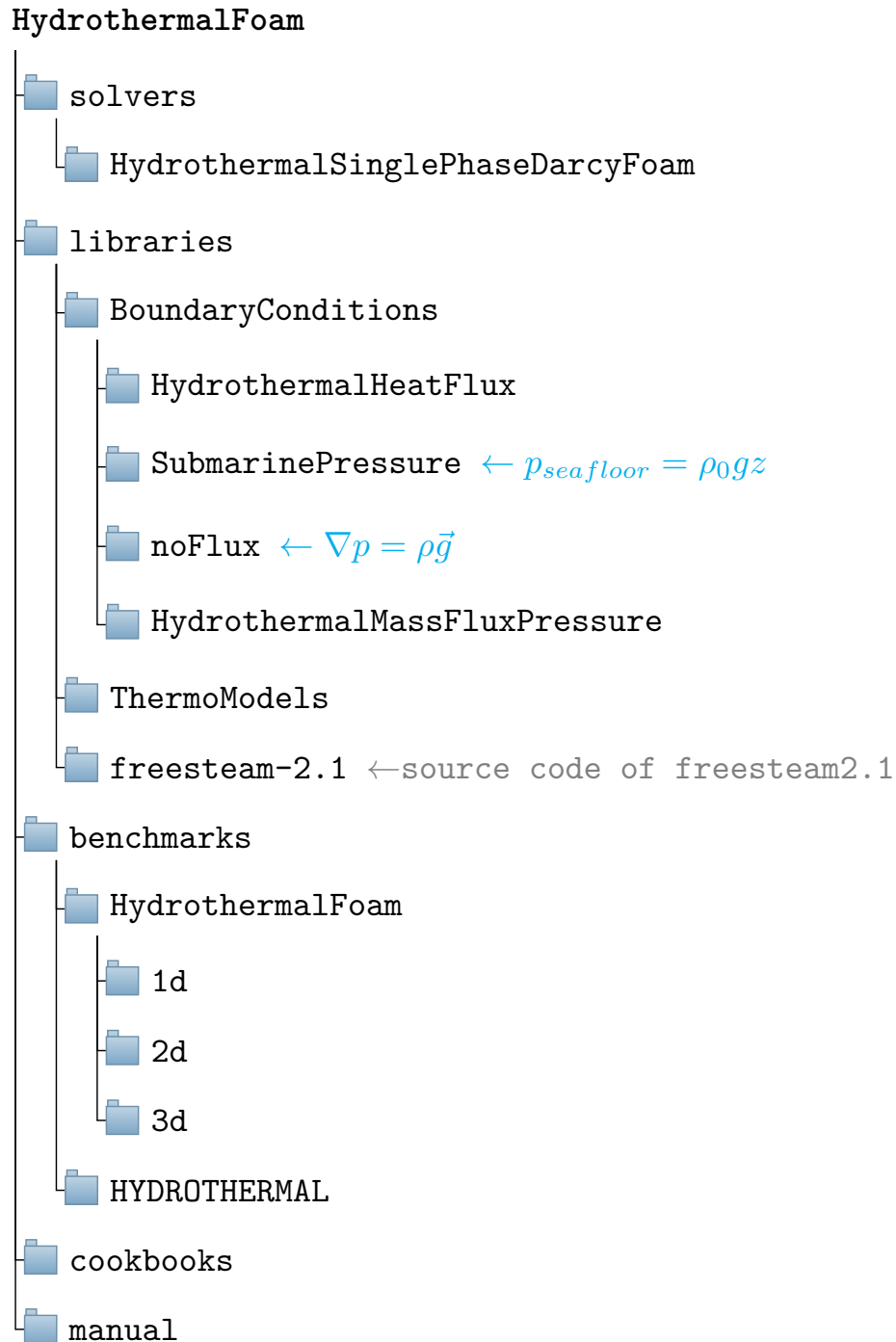


Fig. 2.1: Structure and components of the HydrothermalFoam toolbox.

MODEL DESCRIPTION

3.1 Solver and equations

The solver is named `HydrothermalSinglePhaseDarcyFoam`, which is designed to resolve fluid flow within submarine hydrothermal circulation systems. The hydrothermal fluid flow is governed by Darcy's law (Eqn. (3.1)), mass continuity (Eqn. (3.2)) and energy conservation (Eqn. (3.4)) equations shown below,

$$\vec{U} = -\frac{k}{\mu_f}(\nabla p - \rho_f \vec{g}) \quad (3.1)$$

$$\varepsilon \frac{\partial \rho_f}{\partial t} + \nabla \cdot (\vec{U} \rho_f) \quad (3.2)$$

$$\varepsilon \rho_f \left(\beta_f \frac{\partial p}{\partial t} - \alpha_f \frac{\partial T}{\partial t} \right) = \nabla \cdot \left(\rho_f \frac{k}{\mu_f} (\nabla p - \rho_f \vec{g}) \right) \quad (3.3)$$

$$(\varepsilon \rho_f C_{pf} + (1 - \varepsilon) \rho_r C_{pr}) \frac{\partial T}{\partial t} = \nabla \cdot (\lambda_r \nabla T) - \rho_f C_{pf} \vec{U} \cdot \nabla T + \frac{\mu_f}{k} \|\vec{U}\|^2 - \left(\frac{\partial \ln \rho_f}{\partial \ln T} \right)_p \frac{Dp}{Dt} \quad (3.4)$$

where the pressure equation (3.3) is derived from continuity equation (3.2) and Darcy's law (3.1) (see [Hasenclever et al., 2014]).

Note: see the manuscript (Guo, Ruepke & Tao, 2020) for details of symbols and mathematical model description.

3.2 Boundary conditions

The available boundary conditions (BCs) of temperature and pressure for `HydrothermalSinglePhaseDarcyFoam` are presented below.

Tip: Syntax of all the input files of OpenFOAM, thus of `HydrothermalFoam` is C++ style. The basic form of a boundary condition can be written as following dictionary structure,

```
patchName
{
    type    boundaryConditionType; //compulsive
    value   flotNumber; //compulsive
    option1 valueOfOption1; //optional
    option2 valueOfOption2; //optional
}
```

where `type` and `value` are always compulsive keys, and sometime the `value` key is just a placeholder but have to be there. `option*` represents some optional parameters for a special boundary condition, e.g. `qmax` for `HydrothermalHeatFlux` shown blow.

3.2.1 Basic boundary conditions

Empty BC: `empty`

The empty BC is used and only applied on non-computed patches for one-dimensional and two-dimensional models. See [Listing 3.1](#) for example.

Listing 3.1: Example of empty BC for a 2D model

```
frontAndBack
{
    type          empty;
}
```

Fixed value BC: `fixedValue`

This is the basic and commonly used Dirichlet BC in OpenFOAM, $f = f_0$ on $\partial\Omega$, where f denotes some field and Ω represents a boundary patch (*same meaning as belows*). See [Listing 3.2](#) for example.

Listing 3.2: Example of fixedValue BC

```
bottom
{
    type          fixedValue;
    value         uniform 473.15; //473.15 K = 200 C
}
```

Fixed gradient: `fixedGradient`

Just like its name, `fixedGradient` specify a Neumann boundary condition on a patch. $\nabla f = g_0$ on $\partial\Omega$. See [Listing 3.3](#) for example.

Listing 3.3: Example of fixedGradient BC.

```
bottom
{
  type          fixedGradient;
  gradient      0.005; //required
}
```

Zero gradient: zeroGradient

It is a special case of `fixedGradient`. $\nabla f = 0$ on $\partial\Omega$. See [Listing 3.4](#) for example.

Listing 3.4: Example of zeroGradient BC for temperature.

```
right
{
  type          zeroGradient;
}
```

Tip: `zeroGradient` is always applied for permeability. Because permeability is not a primary variable and thus don't need to solve it. But we have to regard it as a field variable just like temperature to initialize the field value, because permeability in our model is not always uniform distribution. Therefore we have to specify a boundary condition for permeability.

Fixed flux pressure: fixedFluxPressure

This boundary condition sets the pressure gradient to the provided value such that the flux on the boundary is that specified by the velocity boundary condition. This `fixedFluxPressure` BC for pressure `p` is commonly combined with `fixedValue` BC for velocity `U`, and of course the velocity field `U` have to be set in `0` folder even though `U` is not a primary variable. see [Listing 3.5](#) for example.

Tip: We highly recommend the new defined `noFlux` or `hydrothermalMassFluxPressure` (see [Section 3.2.2](#), [Section 3.2.2](#)) for Neumann boundary condition of pressure.

Listing 3.5: Example of fixedFluxPressure BC

```
right
{
  type          fixedFluxPressure;
}
```

Inlet and outlet BC: `inletOutlet`

This is a generic outflow boundary condition and is commonly applied to temperature T at seafloor boundary. It is a mixed boundary condition that using `zeroGradient` BC when fluid flow out of the boundary and using a `fixedValue` BC when fluid flow into the boundary. See [Listing 3.6](#) for example and options.

Listing 3.6: Example of `inletOutlet` BC

```
top
{
    type            inletOutlet;
    phi             phi; //optional
    inletValue      uniform 278.15; //required, fixed value for inflow
    value           uniform 278.15; //required, recommend the same value as
    ↪inletValue
}
```

3.2.2 Customized boundary conditions for pressure

The following customized boundary conditions of pressure p are designed for seafloor hydrothermal models.

Zero mass flux BC: `noFlux`

This boundary condition is always applied on impermeable insulating boundary, e.g. side wall. See [Listing 3.7](#) for example and options.

Listing 3.7: Example of `noFlux` BC

```
left
{
    type            noFlux;
}
right
{
    type            noFlux;
}
```

Seafloor BC: `submarinePressure`

This is a spatial coordinate dependent Dirichlet BC and is derived from `fixedValue` BC. The pressure boundary value is the hydrostatic pressure on the boundary patch, which is calculated from coordinate y of the patch. This is designed for seafloor boundary. See [Listing 3.8](#) for example.

Listing 3.8: Example of submarinePressure BC

```
top
{
  type          submarinePressure;
}
```

Mass flux BC: hydrothermalMassFluxPressure

According Darcy' s law (3.1), gradient of pressure can be expressed by velocity, thus expressed by mass flux. See Listing 3.9 for example.

Listing 3.9: Example of hydrothermalMassFluxPressure BC

```
bottom
{
  type          hydrothermalMassFluxPressure;
  q             uniform -0.015; // inflow
}
```

where q in the dictionary denotes mass flux value with unit of $kg/m^2/s$. If mass flux represents inflow through the boundary, the value is negative, otherwist is positive.

3.2.3 Customized boundary conditions for temperature**Heat flux BC: hydrothermalHeatFlux**

A commonly used Neumann boundary condition of temperature T in hydrothermal modeling is heat flux (W/m^2) BC. For example, it is applied on bottom boundary representing heat source. A basic example is shown in Listing 3.10, heat flux on the boundary patch (named `heatsource`) is a constant equal to $5 W/m^2$.

Listing 3.10: Example of hydrothermalHeatFlux BC

```
heatsource
{
  type          HydrothermalHeatFlux;
  shape         fixed; //Optional, default is fixed.
  q             uniform 5; //W/m^2
  value         uniform 0; //Placeholder
}
```

The `hydrothermalHeatFlux` also support Gaussian shape heat flux, see equation (3.5) and Fig. 3.1.

$$q_h(x, z) = q_{min} + (q_{max} - q_{min})e^{-\frac{(x-x_0)^2+(y-z_0)^2}{2c^2}} \quad (3.5)$$

See Listing 3.11 and Listing 3.12 for 2D and 3D example, respectively.

Listing 3.11: Example of Gaussian shape heat flux BC for 2D model (see also Fig. 3.1)

```
bottom
{
  type      hydrothermalHeatFlux;
  q         uniform 0.05; //placeholder
  value     uniform 0; //placeholder
  shape     gaussian2d;
  x0        0;
  qmax      5;
  qmin      0.05;
  c         500;
}
```

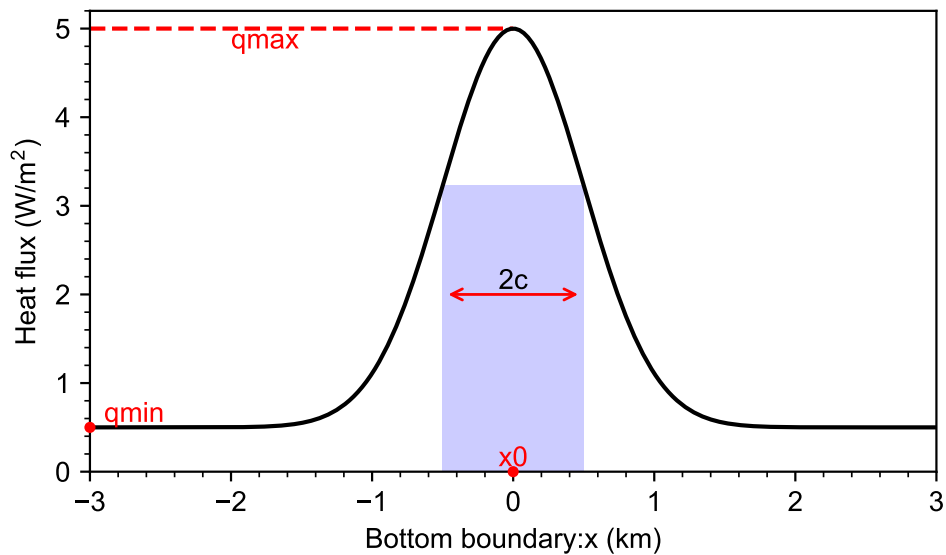


Fig. 3.1: Gaussian shape heat flux curve corresponding to Listing 3.11.

Listing 3.12: Example of Gaussian shape heat flux BC for 3D model

```
bottom
{
  type      hydrothermalHeatFlux;
  q         uniform 0.05; //placeholder
  value     uniform 0; //placeholder
  shape     gaussian2d;
  x0        0;
  z0        0;
  qmax      5;
  qmin      0.5;
}
```

(continues on next page)

(continued from previous page)

```
c      500;
}
```

3.2.4 Coded boundary conditions

The `fixedValue` BC mentioned above basically specifies a constant boundary condition value. OpenFOAM provided a so-called dynamic compiling mechanism which allows user to define a customized fixed value boundary condition, e.g. a coordinate dependent `fixedValue` BC.

- **Coded fixed value BC:** `codedFixedValue`

Warning: Some programming experience of OpenFOAM, at least C++ is required to use this boundary condition.

An example of a Gaussian shape fixed temperature boundary condition is shown in [Listing 3.13](#).

Listing 3.13: Example of Gaussian shape fixed temperature (T) BC of a 2D model.

```
bottom
{
  type      codedFixedValue;
  value     uniform 873.15; //placeholder
  name      gaussShapeT;
  code      #{
              scalarField x(this->patch().Cf().component(0));
              double wGauss=200;
              double x0=1000;
              double Tmin=573;
              double Tmax=873.15;
              scalarField T(Tmin+(Tmax-Tmin)*exp(-(x-x0)*(x-x0)/
↪ (2*wGauss*wGauss)));
              operator==(T);
            #};
}
```

The code shown in [Listing 3.13](#) implements a Dirichlet boundary condition for temperature `T` which varies along **bottom** boundary with coordinate `x`, the temperature distribution curve is shown in [Fig. 3.2](#). In addition, there is also a `codedMixed` BC available for dynamic compiled mixed boundary condition, see [OpenFOAM documentation](#) for more details.

3.3 Properties

The parameters in the governing equations can be classified as transport and thermophysical properties.

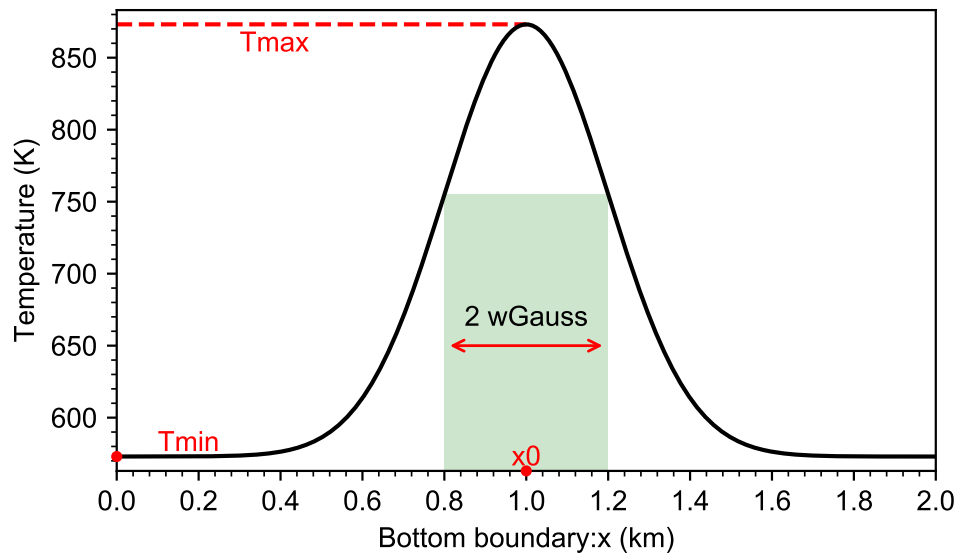


Fig. 3.2: Gaussian shape temperature curve corresponding to Listing 3.13.

3.3.1 Transport properties

The transport properties in HydrothermalFoam are porosity, thermal conductivity k_r of rock, specific heat capacity cp_rock of rock, density ρ_{rock} of rock. All the transport properties are stored in `constant/transportProperties` file, see Listing 3.14 for example.

Listing 3.14: Example of transport properties.

```

1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        dictionary;
6     location     "constant";
7     object       transportProperties;
8 }
9
10 porosity porosity [0 0 0 0 0 0 0] 0.1;
11 kr kr [1 1 -3 -1 0 0 0] 2;
12 cp_rock cp_rock [0 2 -2 -1 0 0 0] 880;
13 rho_rock rho_rock [1 -3 0 0 0 0 0] 2700;

```

Note: The head (line 1-8 in Listing 3.14) of the `transportProperties` file always keep the same, users just need to modify the transport properties (line 10-12 in Listing 3.14) for a specific hydrothermal system.

3.3.2 Thermophysical properties

The thermophysical properties describe the equation of state (EOS) and thermodynamic properties of a specific fluid, e.g. water. For single phase hydrothermal circulation modeling, we developed a [OpenFOAM thermophysical model](#) (named `htHydroThermo`) of water based on [IAPWS-IF97](#) and [freesteam-2.1](#) project. Similar as other [OpenFOAM thermophysical model](#) for other specific solvers, the usage of water `htHydroThermo` is shown in [Listing 3.15](#).

Listing 3.15: Usage of thermophysical model of `htHydroThermo`.

```

1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        dictionary;
6     location     "constant";
7     object       thermophysicalProperties;
8 }
9
10 thermoType
11 {
12     type         htHydroThermo;
13     mixture      pureMixture;
14     transport    IAPWS;
15     thermo       IAPWS;
16     equationOfState IAPWS; //Boussinesq
17     specie       specie;
18     energy       temperature; //sensibleEnthalpy
19 }
20
21 mixture
22 {
23     specie
24     {
25         molWeight  18;
26     }
27 }

```

Note: `htHydroThermo` is the unique support thermophysical model for solver of `HydrothermalSinglePhaseDarcyFoam` so far. Therefore, **please always copy Listing 3.15 and save it in file of `constant/thermophysicalProperties`.**

Warning: The temperature and pressure limitation of `htHydroThermo` are $[273.15, 1073.15]$ K and $[10^5, 10^8]$ Pa, respectively. Please make sure the temperature and pressure field value are in the valid range.

Usage of `HydrothermalFoam` tool is similar to any other solver of OpenFOAM, see OpenFOAM `user guide` for more details. The basic usage of `HydrothermalFoam` are shown as below.

4.1 Mesh generation

There are several ways to generate a mesh, including OpenFOAM's built-in `blockMesh` application for generating meshes of simple geometries, `snappyHexMesh` application for meshing complex geometries and applications that convert meshes from well known formats into the OpenFOAM format, e.g. `fluentMeshToFoam`, `gmshToFoam`. see OpenFOAM `user guide-mesh generation and conversion` <<https://cfd.direct/openfoam/user-guide/v7-mesh/>> for details.

In this section we simply present how to use `blockMesh` and `gmshToFoam` for generating a mesh and defining boundary patches.

4.1.1 Simple mesh generation: `blockMesh`

The mesh is generated from a dictionary file named `blockMeshDict` located in the `system` directory of a case. `blockMesh` reads this dictionary, generates the mesh and writes out the mesh data to `points` and `faces`, `cells` and `boundary` files in the `constant/polyMesh` directory (see Fig. 4.5).

`blockMesh` decomposes the domain geometry into a set of 1 or more three dimensional, hexahedral blocks. Each block of the geometry is defined by 8 vertices, one at each corner of a hexahedron. A simple example block and vertex numbering is shown in Fig. 4.1. The vertices are written in a list so that each vertex can be accessed using its label, remembering that OpenFOAM always uses the C++ convention that the first element of the list has label '0'.

An example of `blockMeshDict` for describing a 2D box is shown in Listing 4.1. The key entries are `vertices`, `blocks` and `boundary`.

- **vertices** contains all three-dimensional coordinates of each vertex (line 15-25), e.g. vertex 0 is $(0 \ \$ymin \ 0)$ and the vertex 5 is $(\$Lx \ \$ymin \ \$Lz)$, the coordinate system is shown in Fig. 4.1.
- **blocks** contains vertex connection of a hexahedron, numbers of cells in each direction and cell expansion ratios. The order of vertex connection is $(0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7)$. $(100 \ 50 \ 1)$ means the number of cells in x , y and z direction will be 100, 50 and 1 (for 2D case), respectively.

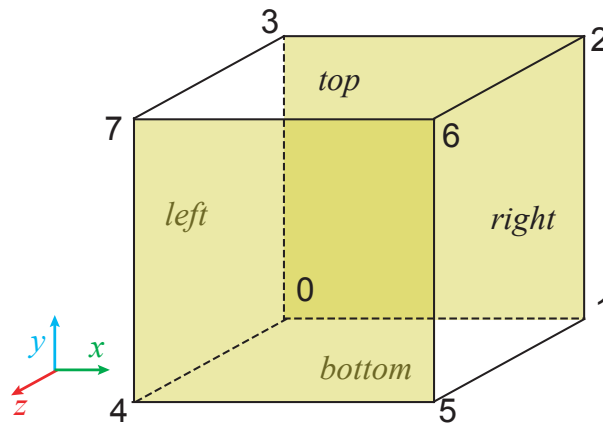


Fig. 4.1: The coordinate system and vertices numbering of single block for `blockMesh`.

`simpleGrading` is typically set to `(1 1 1)`, see OpenFOAM user guide-`blockMesh` for more details.

- **boundary** contains sub-dictionaries for defining boundary patches. The name (e.g. `right` or whatever the user like) of the sub-dictionary is the boundary patch name which will be used in filed data (see Section 3.2). There are two key entries, `type` and `faces` in each sub-dictionary. The `type` is typically set to `patch` or `empty` for a non-computing boundary patch of a non-three-dimensional case. `faces` is a list of block faces that make up the patch with a user defined name (see lines 43-46 for `right` patch or lines 67-71 for `frontAndBack` patch shown as transparent surface in Fig. 4.1)

Listing 4.1: Example of `blockMeshDict` for a 2D box.

```

1 FoamFile
2 {
3     version     2.0;
4     format      ascii;
5     class       dictionary;
6     object      blockMeshDict;
7 }
8 // * * * * *
9
10 convertToMeters 1;
11 Lx 2000; //variable definition
12 ymin -3000;
13 ymax -2000;
14 Lz 1;
15 vertices //vertices definition
16 (
17     (0 $ymin 0) //coordinate of vertex 0
18     ($Lx $ymin 0) //coordinate of vertex 1
19     ($Lx $ymax 0) //coordinate of vertex 2
20     (0 $ymax 0) //coordinate of vertex 3
21     (0 $ymin $Lz) //coordinate of vertex 4

```

(continues on next page)

(continued from previous page)

```

22     ($Lx    $ymin    $Lz)//coordinate of vertex 5
23     ($Lx    $ymax    $Lz)//coordinate of vertex 6
24     (0      $ymax    $Lz)//coordinate of vertex 7
25 );
26 blocks
27 (
28     hex (0 1 2 3 4 5 6 7) (100 50 1) simpleGrading (1 1 1)
29 );
30 boundary
31 (
32     left    //patch name
33     {
34         type patch ;
35         faces //face list
36         (
37             (0 4 7 3)
38         );
39     }
40     right
41     {
42         type patch;
43         faces
44         (
45             (2 6 5 1)
46         );
47     }
48     top
49     {
50         type patch;
51         faces
52         (
53             (3 7 6 2)
54         );
55     }
56     bottom
57     {
58         type patch;
59         faces
60         (
61             (1 5 4 0)
62         );
63     }
64     frontAndBack //patch name
65     {
66         type empty;
67         faces //face list
68         (

```

(continues on next page)

(continued from previous page)

```

69         (0 3 2 1) //back face
70         (4 5 6 7) //front face
71     );
72 }
73 );
74 // ***** //

```

Tip: We can also use variable in `blockMeshDict`. For example `ymin -3000;`, it should be noted that there is no `=` between variable name and value, and `;` is required. In addition, the `convertToMeters` keyword specifies a scaling factor by which all vertex coordinates in the mesh description are multiplied, e.g. `convertToMeters 0.01;` scales to mm.

Then just type `blockMesh` command in the terminal in the root directory of the case after finishing `blockMeshDict`.

4.1.2 Convert gmsh to OpenFOAM: gmshToFoam

The OpenFOAM built-in meshing utility, `blockMesh`, can generate some simple mesh, but it is difficulty to describe complex geometry in `blockMeshDict` dictionary file. In this section, we introduce how to use `gmsh` to generate mesh and then use `gmshToFoam` convert the mesh to OpenFOAM format.

Create geometry and define boundary patches

Here we present a example for creating 2D and 3D box geometry, and defining boundary patches. See [gmsh manual \(pdf\)](#) for more details about geometry definition and mesh generation.

- **Geometry description**

The geometry descriptions are stored in a gmsh geometry script file with extension `.geo`. Gmsh script files support both C and C++ style comments and the syntax is similar to C/C++ as well. The user can use gmsh GUI to create a geometry script file, and also can write the geometry script file directly in your favorite text editor, e.g. [Visual Studio Code](#) with [gmsh extension](#). An gmsh geometry script file of a 3D box is shown in [Listing 4.2](#) (`box.geo`).

Listing 4.2: Example of gmsh geometry script of a 3D box.

```

1 // 0. define some variables
2 xmin=0;
3 xmax=600;
4 ymin=-200;
5 ymax=0;
6 zmin=0;
7 zmax=100;
8 lc=10;

```

(continues on next page)

(continued from previous page)

```

9 // 1. define points
10 Point(1) = {xmin, ymax, zmin, lc};
11 Point(2) = {xmax, ymax, zmin, lc};
12 Point(3) = {xmax, ymin, zmin, lc};
13 Point(4) = {xmin, ymin, zmin, lc};
14 // 2. define lines
15 Line(1) = {1, 2};
16 Line(2) = {2, 3};
17 Line(3) = {3, 4};
18 Line(4) = {4, 1};
19 // 3. define line loop and surface
20 Line Loop(6) = {4, 1, 2, 3};
21 Plane Surface(6) = {6};
22 // 4. extrude 2D surface to a 3D volume
23 Extrude {0, 0, zmax} {
24 Surface{6};
25 Layers{1}; //set layer number to 1 for 2D model
26 Recombine;
27 }

```

The geometry script file can be opened and visualized by gmsh GUI, see Fig. 4.2. Gmsh supports three-dimensional interactive operation and there are mouse tooltips of number and other informations for each point, line, surface and volume. Therefor the user can easily get the number of each boundary patch and volume (cell region).

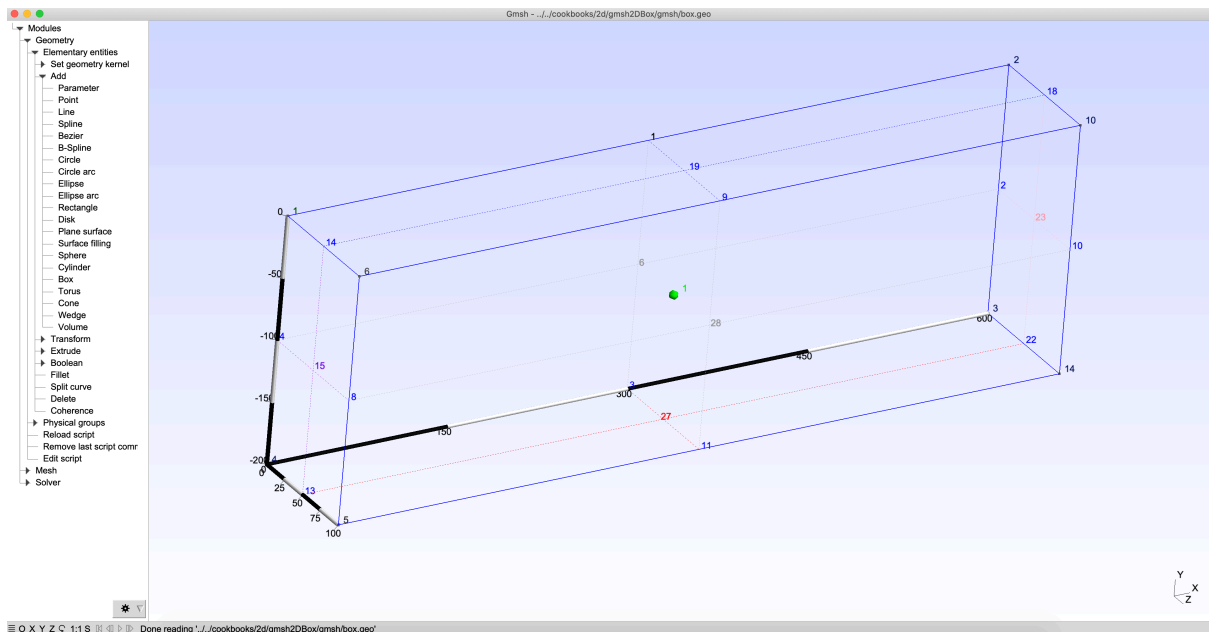


Fig. 4.2: Gmsh GUI display a 2D box geometry.

- **Boundary patches and cell regions definition**

The user can specify a specific name, e.g. `bottom`, for each `Surface` (gmsh keyword) or a name for boundary patches group, e.g. `frontAndBack`. These boundary patches name will be used to specify

boundary conditions in field data file(see [Section 3.2](#)). In addition, the user can also specify specific name for each Volume (gmsh keyword). This volume name will be used to set field distribution, e.g. permeability (see [Section 4.3](#)). The specification of boundary patch name and cell region name can be done by Physical keyword, see

Listing 4.3: Example of boundary patch definition in gmsh geometry script.

```
1 // 5. define boundary patches via Physical keyword
2 Physical Surface("frontAndBack") = {28,6};
3 Physical Surface("bottom") = {27};
4 Physical Surface("left") = {15};
5 Physical Surface("top") = {19};
6 Physical Surface("right") = {23};
7 // 6. specify a name for cell region which is used for 'setFields'
8 Physical Volume("internal") = {1};
```

Tip: Gmsh will specify some default colors for every surfaces and volumes. The user can also set specific color for each patch and volume (see below) to check boundary mesh.

```
1 // 7. specify different color for different boundary patches
2 Color Gray{Surface{28, 6};}
3 Color Red{Surface{27};}
4 Color Purple{Surface{15};}
5 Color Pink{Surface{23};}
6 Color Blue{Surface{19};}
7 Color Green{Volume{1};}
```

Generate mesh

Mesh generation process is pretty easy, the user can do it using gmsh GUI *Mesh -> 3D*. The mesh result is shown in [Fig. 4.3](#). Because gmshToFoam in OpenFOAM-7 can only read gmsh .msh file in format of version 2, the user have to export (*File -> Export -> Gmsh MESH*) the mesh file to **Version 2 ASCII** format (see [Fig. 4.4](#)).

Alternately, the mesh generation and save file can be done by a single line of command (see [Listing 4.4](#))

Listing 4.4: Meshing and saving command of gmsh.

```
gmsh gmsh/mesh.geo -3 -o gmsh/mesh.msh -format msh22`.
```

Convert mesh

If the mesh file is generated successfully, the user can convert the mesh to OpenFOAM format by running command of gmshToFoam mymesh.msh in the root directory of a case. Then a directory named

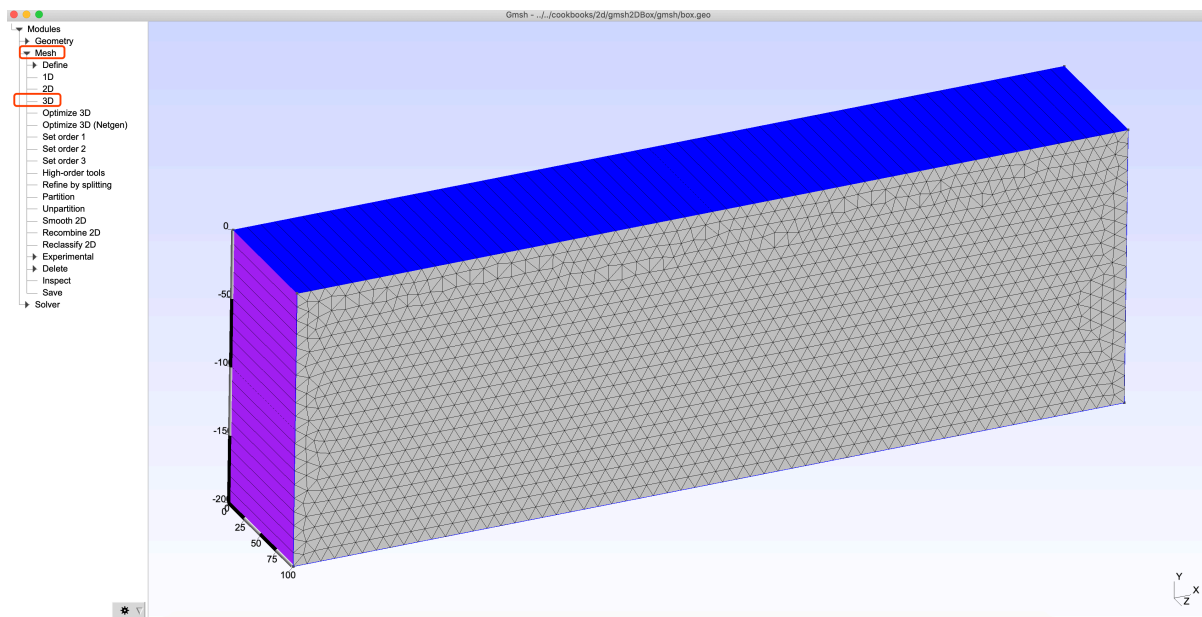


Fig. 4.3: Gmsh GUI display a 2D box mesh.

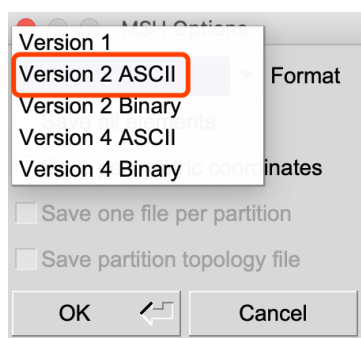


Fig. 4.4: Export msh to Version 2.

polyMesh will be generated in constant directory. All the defined boundary patches are converted to polyMesh/boundary file (see Listing 4.5). If the case is three-dimensional, now the mesh generation process is done. But for 2D case, the user have to set empty boundary pathches by modifying the boundary file directly.

Listing 4.5: polyMesh/boundary converted by gmshToFoam.

```
5
(
  frontAndBack
  {
    type          patch;
    physicalType  patch;
    nFaces        6240;
    startFace     4600;
  }
  left
  {
    type          patch;
    physicalType  patch;
    nFaces        20;
    startFace     10840;
  }
  top
  {
    type          patch;
    physicalType  patch;
    nFaces        60;
    startFace     10860;
  }
  right
  {
    type          patch;
    physicalType  patch;
    nFaces        20;
    startFace     10920;
  }
  bottom
  {
    type          patch;
    physicalType  patch;
    nFaces        60;
    startFace     10940;
  }
)
```

Tip: We provide a python script (setEmptyPatch.py) to modify a boundary patch to empty by patch name. For example, set frontAndBack patches to empty by running the following command at case

root directory.

```
python setEmptyPatch.py frontAndBack
```

4.2 Case setup

A model case of `HydrothermalFoam` basically consists of **time directory** (e.g. 0 for initial state), **constant** and **system** directory (see Fig. 4.5). The sub-directory of **polyMesh** in **constant** folder consists mesh files, which are generated by meshing utility, e.g. `blockMesh`, `gmshToFoam`, `snappyHexMesh`, ..., see Section 4.1.

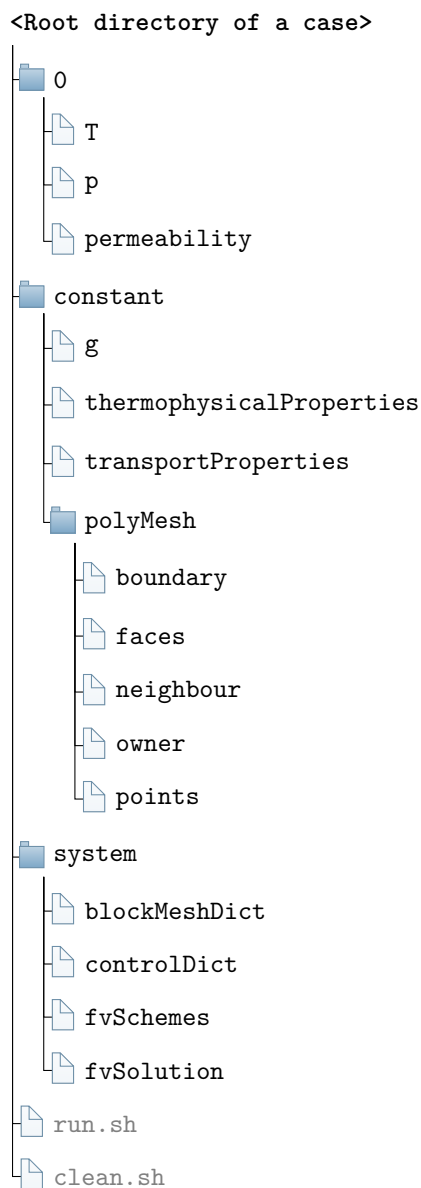


Fig. 4.5: Directory structure of a case of `HydrothermalFoam`.

4.2.1 Initial state

The field data files of initial state are commonly stored in 0 time directory. Of course it can be another time directory, e.g. 1000, which is specified by key of `startFrom` in `controlDict` in `system` directory (see Section 4.2.3). In the initial state directory, the field file of primary variable `T`, `p` and permeability are compulsive, and `U` is also required if `fixedFluxPressure` is applied on a boundary patch for pressure (see Section 3.2.1). An example field data file of `T` is shown in Listing 4.6, which is a basic structure of dictionary file of a field data. A field data file basically contains its **variable type** (line 5), **object name** (line 6), **dimension** (line 9), **internal field** value (line 10) and **boundary field** value (boundary conditions, line 11-42).

Listing 4.6: Example field data of temperature.

```

1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        volScalarField;
6     object       T;
7 }
8
9 dimensions      [0 0 0 1 0 0 0];
10 internalField   uniform 278.15;    //5 C
11 boundaryField
12 {
13     left
14     {
15         type      zeroGradient;
16     }
17     right
18     {
19         type      zeroGradient;
20     }
21     top
22     {
23         type      fixedValue;
24         value     uniform 278.15;
25     }
26     bottom
27     {
28         type      hydrothermalHeatFlux;
29         q         uniform 0.05;
30         value     uniform 0;
31     }
32     heatsource
33     {
34         type      hydrothermalHeatFlux;
35         q         uniform 5;

```

(continues on next page)

(continued from previous page)

```

36     value          uniform 0;
37   }
38   frontAndBack
39   {
40     type          empty;
41   }
42 }

```

- **Variable type** (`class`) and **object name** (`object`). The See the manuscript(Guo, Ruepke & Tao, 2020) for variable type and object name index.
- **Internal field** (`internalField`) can be set as uniform (just like line 10 in Listing 4.6) or non-uniform (see Section 4.3).
- **Boundary conditions** (`boundaryField`). The boundary patch name, e.g. `bottom`, is defined in mesh file of `constant/polyMesh/boundary` (see Fig. 4.5 and Section 4.1). The available boundary conditions and their usage can be found in Section 3.2.

Tip: To avoid making mistakes, user should copy the field data files form any case in [cookbooks](#) or [benchmarks](#) and then make some changes.

4.2.2 Constant

For `HydrothermalFoam`, the `constant` directory always contains three files named `g`, `thermophysicalProperties`, `transportProperties` respectively, and one folder named `polyMesh` (see Fig. 4.5).

- **g** contains gravitational acceleration constant. It is same for basically all cases of OpenFOAM-based solvers. Therefor, please just copy this file from any existed case to a new case.
- **thermophysicalProperties** contains thermophysical model. It is the same for all cases of `HydrothermalSinglePhaseDarcyFoam` solver, see Section 3.3.2.
- **transportProperties** contains constant parameters about transport, see Section 3.3.1.
- **polyMesh** directory contains necessary mesh files which are automatically generated by meshing utility, e.g. `blockMesh`, `gmshToFoam` (see Section 4.1).

4.2.3 System

As shown in Fig. 4.5, the `system` directory contains three compulsive directory files of `controlDict`, `fvSchemes` and `fvSolution`, and optional dictionary file, e.g. `blockMeshDict` for OpenFOAM built-in meshing utility `blockMesh`.

Time and data input/output control

The `controlDict` dictionary sets time and data input/output control, the commonly used entries for `HydrothermalSinglePhaseDarcyFoam` solver are shown in Listing 4.7. See OpenFOAM user guide-`controlDict` for more details of each entry. A few entries that need to be emphasized are highlighted in the Listing 4.7.

- **application** specifies solver name, which should be `HydrothermalSinglePhaseDarcyFoam`.
- **libs** contains some shared libraries using in the case. These three libraries listed in **lines 46-48** must be included in **libs** sub-dictionary for customized thermophysical model and boundary conditions.

Listing 4.7: Example entries from a `controlDict` dictionary.

```

1 FoamFile
2 {
3     version    2.0;
4     format     ascii;
5     class      dictionary;
6     object     controlDict;
7 }
8
9 application HydrothermalSinglePhaseDarcyFoam;
10 startFrom latestTime;
11 startTime 0;
12 stopAt endTime;
13 endTime 6912000000; //8640000000
14 deltaT 864000;
15 adjustTimeStep yes;
16 maxCo 0.8;
17 maxDeltaT 86400000;
18 writeControl adjustableRunTime;
19 writeInterval 86400000;
20 purgeWrite 0;
21 writeFormat ascii;
22 writePrecision 6;
23 writeCompression off;
24 timeFormat general;
25 timePrecision 14;
26 runtimeModifiable true;
27 libs
28 (
29     "libfreesteam.so"
30     "libHydrothermalBoundaryConditions.so"
31     "libHydroThermoPhysicalModels.so"
32 );

```

Note: There are three options (`firstTime`, `startTime`, `latestTime`) available for the

`startFrom` keyword entry. `startTime` specifies start time for the simulation when `startFrom` `startTime`;

Numerical schemes

The `fvSchemes` dictionary in the `system` directory sets the numerical schemes for terms, such as derivatives in equations, that are calculated during a simulation. The commonly used entries of `fvSchemes` for `HydrothermalSinglePhaseDarcyFoam` solver are shown in [Listing 4.8](#). See [OpenFOAM user guide-fvSchemes](#) for more available scheme options of each terms.

As shown in [Listing 4.8](#), user have to specify numerical scheme for transient (lines 11-14), gradient of pressure p (line 19) and temperature T (line 20), laplacian terms (line 33-34), surface interpolation (lines 37-40), surface gradient (lines 42-45). In addition, the `fluxRequired` sub-dictionary have to be specified for reconstructing Darcy velocity from flux after solving pressure. See **Model development** section in the manuscript (Guo, Ruepke & Tao, 2020) or [source code the solver](#) for more detail implementation of each terms.

Listing 4.8: Example entries from a `fvSchemes` dictionary.

```

1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        dictionary;
6     location     "system";
7     object       fvSchemes;
8 }
9
10 ddtSchemes
11 {
12     default      Euler;
13 }
14 gradSchemes
15 {
16     default      none;
17     grad(p)      Gauss linear;
18     grad(T)      Gauss linear;
19 }
20 divSchemes
21 {
22     default      none;
23     div(phi,T)   Gauss upwind;
24     div((phi*interpolate(Cp)),T) Gauss vanLeer;
25 }
26 laplacianSchemes
27 {
28     default      none;

```

(continues on next page)

(continued from previous page)

```

29     laplacian(kr,T) Gauss linear corrected;
30     laplacian(rhorAUf,p) Gauss linear corrected;
31 }
32 interpolationSchemes
33 {
34     default          linear;
35 }
36 snGradSchemes
37 {
38     default          corrected;
39 }
40 fluxRequired
41 {
42     default no;
43     p;
44 }

```

Tip: The basic numerical schemes of `HydrothermalSinglePhaseDarcyFoam` solver are shown in [Listing 4.8](#), user can other available schemes of each term (see `OpenFOAM-fvSchemes`), e.g. `Gauss vanLeer` for `div(phi, T)`.

Solution and algorithm control

The equation solvers, tolerances and algorithms are controlled from the `fvSolution` dictionary in the `system` directory. An example set of entries from the `fvSolution` dictionary required for the `HydrothermalSinglePhaseDarcyFoam` solver is shown in [Listing 4.9](#). See `OpenFOAM` user guide-[fvSolution](#) for more details of each entry.

As shown in [Listing 4.9](#), user have to specify `linear equations solver`, `preconditioner`, `tolerance` and `relTol` for pressure and temperature in sub-dictionary of `p` and `T`, respectively. In addition, a sub-dictionary named in format of `*Final` is required as well (see lines 20-24 and 32-36). The prefix of `*` denotes primary variable name, e.g. `p` for pressure. The key entry `relTol` is typically set to 0. For `HydrothermalSinglePhaseDarcyFoam` solver, key entry `PIMPLE` is also required besides solvers, because we adopt `pimple.correctNonOrthogonal()` for non-orthogonal correction. The key entry `nNonOrthogonalCorrectors` in `PIMPLE` dictionary specifies repeated solutions of the pressure equation, used to update the explicit non-orthogonal correction of Laplacian term (see `OpenFOAM` user guide-[Surface normal gradient schemes](#) for more details). `nNonOrthogonalCorrectors` is typically set to 0 or 1.

Listing 4.9: Example entries from a `fvSolution` dictionary.

```

1 FoamFile
2 {
3     version      2.0;

```

(continues on next page)

(continued from previous page)

```
4     format      ascii;
5     class       dictionary;
6     location    "system";
7     object      fvSolution;
8 }
9
10 solvers
11 {
12     p
13     {
14         solver      PCG;
15         preconditioner DIC;
16         tolerance   1e-12;
17         relTol      0;
18     }
19     pFinal
20     {
21         $p;
22         relTol      0;
23     }
24     T
25     {
26         solver      PBiCG;
27         preconditioner DILU;
28         tolerance   1e-06;
29         relTol      0;
30     }
31     TFinal
32     {
33         $T;
34         relTol      0;
35     }
36 }
37 PIMPLE
38 {
39     nNonOrthogonalCorrectors 0;
40 }
```

Warning: The entries and values shown in Listing 4.9 are the recommended options, it is unnecessary to modify them unless the user understands the source code of the solver and OpenFOAM and then wants to test other available parameters of `fvSolution`.

4.3 Set fields

The user can set a specific value for a field (e.g. permeability) in a specific cell region using utility `setFields`, which reads dictionary file `setFieldsDict` in `system` directory. An example of `setFieldsDict` is shown in Listing 4.10 (see Section 5.2).

Listing 4.10: Example of `setFieldsDict` file.

```
1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        dictionary;
6     location     "system";
7     object       setFieldsDict;
8 }
9 defaultFieldValues
10 (
11     volScalarFieldValue permeability 1e-15
12 );
13 regions
14 (
15     zoneToCell
16     {
17         name "layer2A";
18         fieldValues
19         (
20             volScalarFieldValue permeability 4e-14
21         );
22     }
23     zoneToCell
24     {
25         name "layer2B";
26         fieldValues
27         (
28             volScalarFieldValue permeability 1e-15
29         );
30     }
31 );
```

The option (e.g. `layer2A` in line 17) of key entry `name` in `zoneToCell` sub-dictionary in Listing 4.10 is defined as `Physical Volume("xxx")={...}`; in `gmsh geometry (.geo)` file (see Section 4.1.2). Another commonly used key entry is `boxToCell` which sets field value in a box defined by the two ends of the diagonal, see Listing 4.11.

Listing 4.11: Example of `boxToCell` in `setFieldsDict` file.

```

1 regions
2 (
3   boxToCell
4   {
5     box (0 0 0) (10 10 10);
6     fieldValues
7     (
8       volScalarFieldValue permeability 4e-14
9     );
10  }
11 )

```

4.4 Run Case

The user can run a case just by running command of `HydrothermalSinglePhaseDarcyFoam` in the root directory of a case. But several processes mentioned above, e.g. mesh generation and/or mesh conversion, empty boundary type modification, case setup and initial field setting, have to be done before running a case. All the pre-processing steps can be assembled into a bash file, e.g. `run.sh` shown in Listing 4.12.

Listing 4.12: Example of command `set(run.sh)` to run a case.

```

1 #!/bin/sh
2 cd ${0%/*} || exit 1 # Run from this directory
3
4 # 1. Source tutorial run functions
5 . $WM_PROJECT_DIR/bin/tools/RunFunctions
6 # 2. get solver name
7 application=`getApplication`
8 # 3. clean case if necessary
9 ./clean.sh
10 # 4. generate mesh file using gmsh command
11 gmsh gmsh/mesh.geo -3 -o gmsh/mesh.msh -format msh22
12 # 5. convert gmsh to OpenFOAM format
13 gmshToFoam gmsh/mesh.msh
14 # 6. set empty patches for 2D case
15 python setEmptyPatch.py frontAndBack
16 # 7. set fields
17 runApplication setFields
18
19 # 8.1 run a case using single thread
20 runApplication $application
21

```

(continues on next page)

(continued from previous page)

```

22 # 8.2 or run a case in parallel
23 # runApplication decomposePar
24 # runParallel $application
25 # runApplication reconstructPar

```

Listing 4.13: Example of command set(clean.sh) to clean a case.

```

1 #!/bin/sh
2 cd ${0%/*} || exit 1 # run from this directory
3
4 # Source tutorial run functions
5 . $WM_PROJECT_DIR/bin/tools/CleanFunctions
6
7 cleanCase

```

Note: If the user want to run a case in parallel, just need to comment line 20 in Listing 4.12 and uncomment lines 23-25 in Listing 4.12. In addition, the user have to setup the decomposeParDict dictionary file in system directory (see Listing 4.14 for example). See OpenFOAM user guide for more details about parallel computing.

Listing 4.14: Example of decomposeParDict dictionary file.

```

1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        dictionary;
6     location     "system";
7     object       decomposeParDict;
8 }
9 // * * * * *
10
11 numberOfSubdomains 4;
12
13 method            scotch;
14
15 simpleCoeffs
16 {
17     n              (2 2 1);
18     delta          0.001;
19 }
20
21 hierarchicalCoeffs
22 {

```

(continues on next page)

(continued from previous page)

```
23     n                (1 1 1);
24     delta            0.001;
25     order            xyz;
26 }
27
28 manualCoeffs
29 {
30     dataFile         "";
31 }
32
33 distributed         no;
34
35 roots               ( );
36
37
38 // ***** //
```

Note: It is of course possible to set up input files for cases completely from scratch. However, in practice, it is often simpler to go through the list of [cookbooks](#) already provided and find one that comes close to what you want to do. You would then modify this cookbook until it does what you want to do. The advantage is that you can start with something you already know works, and you can inspect how each change you make – changing the details of the geometry, changing the boundary conditions, or changing initial field distribution – affects what you get.

4.5 Post-processing and visualization

The commonly used post-processing tool is [ParaView](#). There is a built-in utility `paraFoam`, which is based on ParaView, can read and render generic results of OpenFOAM case. The user can run the `paraFoam` command in the root directory of case direction, or run command of `paraFoam -case <caseDir>` in any other directory.

Tip: The new version (e.g. 5.5.0) of [ParaView](#) has OpenFOAM case reader which will read a file with extension of `.foam` in the root directory of a case. Therefore, if the user install OpenFOAM or HydrothermalFoam tool via Docker (see [Section 2.1](#) and [video](#)), the results can be visualized in host through the shared folder. The results will saved in the shared folder when running a case in the container, and then create a empty file with extension of `.foam`, e.g. `results.foam`, in the shared folder. The user can open file `results.foam` in the shared folder in host by ParaView to display the results.

5.1 Convection in a box

In this first example, let us consider a simple situation: a 2d box that is heated from below, insulated at the side walls, and cooled from the top. We will start from a 2D box example and then make some changes to show its features.

5.1.1 Hello World: 2D box

All the input files can be found in [cookbooks/helloworld](#) directory.

This is the first simplest example to show the basic steps to run a case. The geometry and boundary conditions are shown in figure Fig. 5.1. The following steps present how to setup a case from scratch, see [helloworld](#) case.

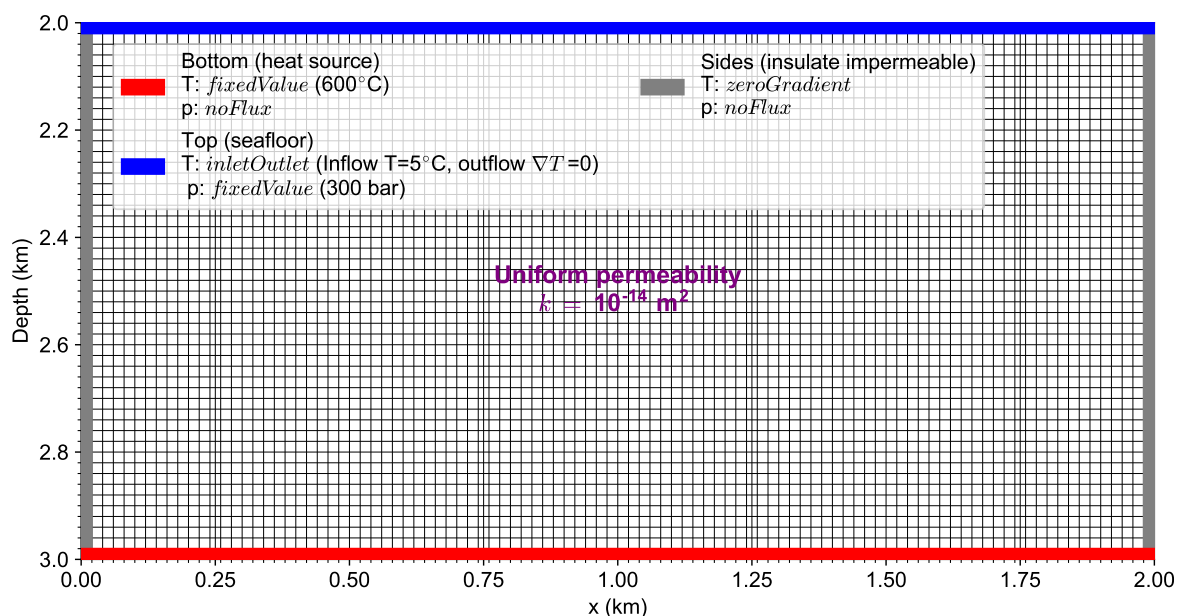


Fig. 5.1: The geometry and boundary conditions of the 2D box model.

Step 1: create controlDict file

First thing we have to do is create a case directory, e.g. named `helloworld`, and sub-directory `system` (see [Listing 5.1](#)).

Listing 5.1: Create case directory.

```
1 mkdir helloworld
2 cd helloworld
3 mkdir system
```

The `controlDict` file is the first and most important file, which is located at `system` directory. Create `controlDict` file in `system` directory with script shown in [Listing 5.2](#).

Listing 5.2: controlDict.

```
1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        dictionary;
6     object       controlDict;
7 }
8
9 application HydrothermalSinglePhaseDarcyFoam;
10 startFrom latestTime;
11 startTime 0;
12 stopAt endTime;
13 endTime 2592000000;
14 deltaT 8640000;
15 writeControl adjustableRunTime;
16 writeInterval 8640000;
17 purgeWrite 0;
18 writeFormat ascii;
19 writePrecision 6;
20 writeCompression off;
21 timeFormat general;
22 timePrecision 14;
23 runTimeModifiable true;
24 libs
25 (
26     "libfreesteam.so"
27     "libHydrothermalBoundaryConditions.so"
28     "libHydroThermoPhysicalModels.so"
29 );
```

Note: The key entry `application` has to be set to `HydrothermalSinglePhaseDarcyFoam` which

is the solver name.

Step 2: mesh generation

In this step we create dictionary `blockMeshDict` (just copy Listing 4.1) in `system` to setup geometry of the 2D box. Then we just run command `blockMesh` to generate mesh (see Fig. 5.1). The directory `constant` will be created automatically and the mesh files will be generated in `polyMesh` directory.

Note: After step 2, the mesh is basically generated and we can display using `paraFoam` utility default or using ParaView for Docker user. The mesh result is shown in Fig. 5.2.

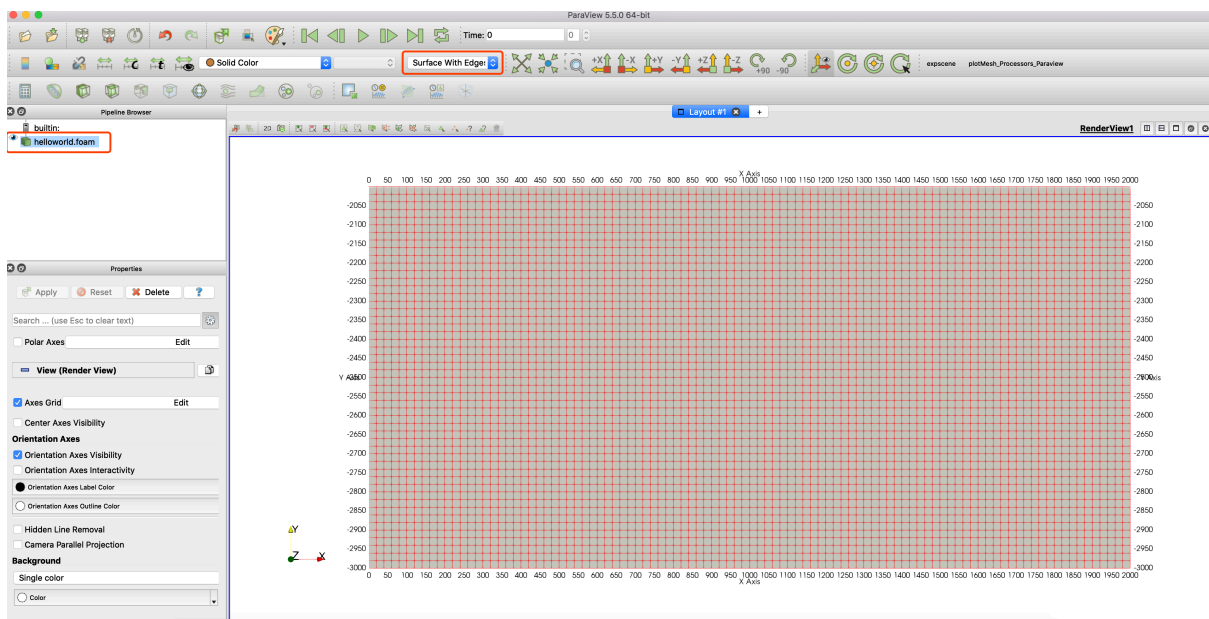


Fig. 5.2: The screenshot of 2D box mesh in ParaView.

Step 3: field data in 0 folder

In this step, we need to create I/O data files of field `p`, `T` and permeability in `0` time directory. This three files are shown in `lst:helloworld:p`, `lst:helloworld:T` and `lst:helloworld:permeability`, respectively.

Listing 5.3: Data field `p`.

```

1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        volScalarField;
6     object       p;

```

(continues on next page)

(continued from previous page)

```

7  }
8  // * * * * *
9
10 dimensions      [1 -1 -2 0 0 0 0];
11
12 internalField    uniform 300e5; //300e5 Pa = 300 bar
13
14 boundaryField
15 {
16     left
17     {
18         type      noFlux;
19     }
20     right
21     {
22         type      noFlux;
23     }
24     top
25     {
26         type      fixedValue;
27         value     uniform 300e5;
28     }
29     bottom
30     {
31         type      noFlux;
32     }
33     frontAndBack
34     {
35         type      empty;
36     }
37 }
38
39 // * * * * *

```

Listing 5.4: Data field T.

```

1  FoamFile
2  {
3     version      2.0;
4     format       ascii;
5     class        volScalarField;
6     object       T;
7  }
8  // * * * * *
9
10 dimensions      [0 0 0 1 0 0 0];
11

```

(continues on next page)

(continued from previous page)

```

12 internalField    uniform 278.15;    //278.15 K = 5 C
13
14 boundaryField
15 {
16     left
17     {
18         type      zeroGradient;
19     }
20     right
21     {
22         type      zeroGradient;
23     }
24     top
25     {
26         type      inletOutlet;
27         phi      phi;
28         inletValue  uniform 278.15;
29     }
30     bottom
31     {
32         type      fixedValue;
33         value     uniform 873.15;
34     }
35     frontAndBack
36     {
37         type      empty;
38     }
39 }
40
41 // ***** //

```

Listing 5.5: Data field permeability.

```

1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        volScalarField;
6     location     "0";
7     object       permeability;
8 }
9 // ***** //
10
11 dimensions      [0 2 0 0 0 0 0];
12
13 internalField    uniform 1e-14;
14

```

(continues on next page)

(continued from previous page)

```

15 boundaryField
16 {
17     left
18     {
19         type          zeroGradient;
20     }
21     right
22     {
23         type          zeroGradient;
24     }
25     top
26     {
27         type          zeroGradient;
28     }
29     bottom
30     {
31         type          zeroGradient;
32     }
33     frontAndBack
34     {
35         type          empty;
36     }
37 }
38
39
40 // ***** //

```

Tip: Now we can display the initial filed of `p`, `T` and `permeability` using ParaView. Screenshot of field `T` is shown in Fig. 5.3. In order to view the initial filed, the user have to uncheck the checkbox of `Skip Zero Time` (see red rectangle in the figure)

Step 4: constant property files

In this step we need to copy three constant files `g` (shown in Listing 5.6), `thermophysicalProperties` (Listing 3.15) and `transportProperties` (Listing 3.14) into the `constant` directory.

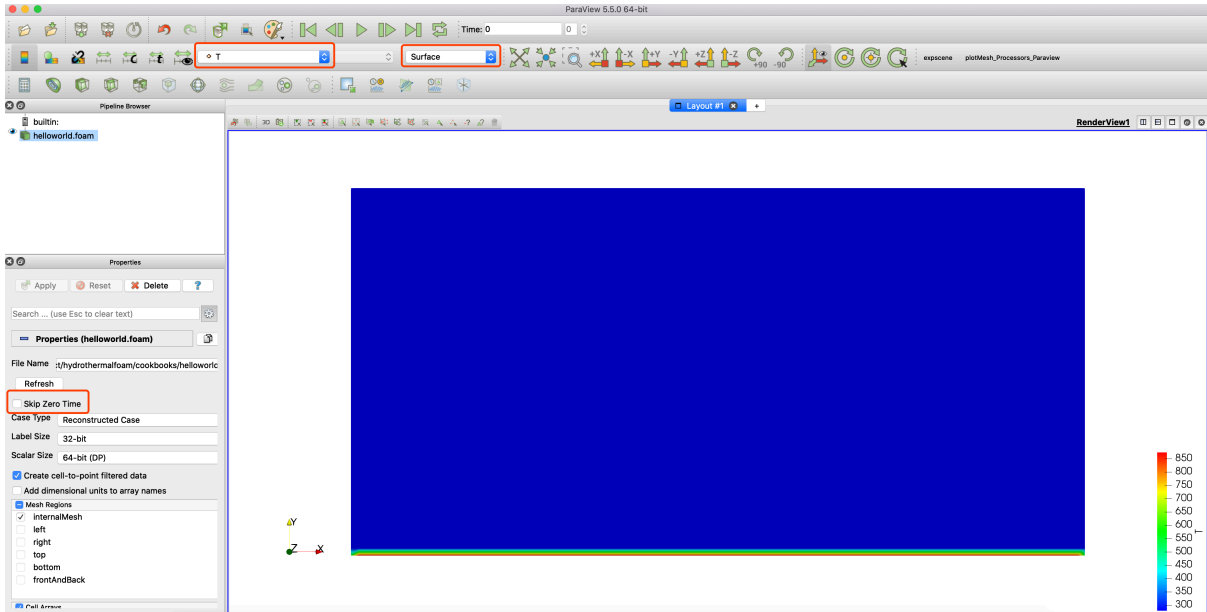
Listing 5.6: Constant property file `g`.

```

1 FoamFile
2 {
3     version      2.0;
4     format       ascii;
5     class        uniformDimensionedVectorField;
6     location     "constant";

```

(continues on next page)

Fig. 5.3: The screenshot of field T in ParaView.

(continued from previous page)

```

7   object      g;
8   }
9   // * * * * *
10  dimensions  [0 1 -2 0 0 0 0];
11  value       (0 -9.81 0);
12  // * * * * *

```

Tip: The user can change some propertie values in `transportProperties`, e.g. `porosity`, according to a specific modeling problem.

Step 4: setup numerical schemes and solution control

In this step we will create numerical schemes and solution control dictionary file `fvSchemes` and `fvSolution` in `system` directory. These two files can be found in [Listing 4.8](#) and [Listing 4.9](#) in [Section 4.2.3](#).

Step 5: run the case

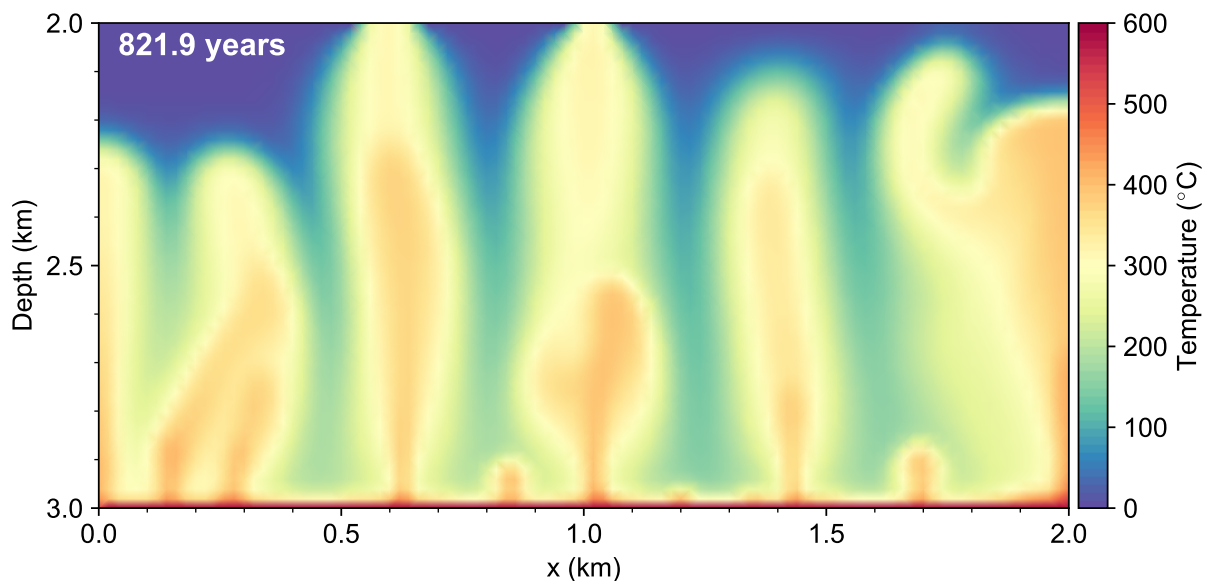
Now the simple case setup or pre-processing has been completed. The we just run the following simple command ([Listing 5.7](#)) to run the case.

Listing 5.7: Command of running the 2D box case.

```
1 blockMesh # mesh generation
2 HydrothermalSinglePhaseDarcyFoam # execute the solver application
```

Step 6: display results

We can use `paraFoam` or ParaView (just like step 3) to display results which saved in time directories. The time directories name is dependent on key entries `writeInterval`, `purgeWrite` and `timePrecision`. The temperature is shown in figure Fig. 5.4.

Fig. 5.4: Temperature result of the **hello world** model.

Tip: If users want to run a case for a long time, there is an excellent tool named `tmux`, which is a terminal multiplexer. It lets you switch easily between several programs in one terminal, detach them (they keep running in the background) and reattach them to a different terminal.

5.1.2 Nonuniform fixed temperature BC

All the input files can be found in `cookbooks/nonUniformFixedValueBC` directory.

This example, based on *Hello World: 2D box*, presents how to set a nonuniform fixed boundary condition by using `codedFixedValue` BC type (see Listing 5.8)

Listing 5.8: Example of nonuniform boundary condition:
codedFixedValue.

```

1  bottom
2  {
3      type          codedFixedValue;
4      value         uniform 873.15; //placeholder
5      name          gaussShapeT;
6      code          #{
7                  scalarField x(this->patch().Cf().component(0));
8                  double wGauss=200;
9                  double x0=1000;
10                 double Tmin=573;
11                 double Tmax=873.15;
12                 scalarField T(Tmin+(Tmax-Tmin)*exp(-(x-x0)*(x-x0)/
↪ (2*wGauss*wGauss)));
13                 operator==(T);
14             #};
15     }

```

The model geometry and boundary conditions are shown in Fig. 5.5.

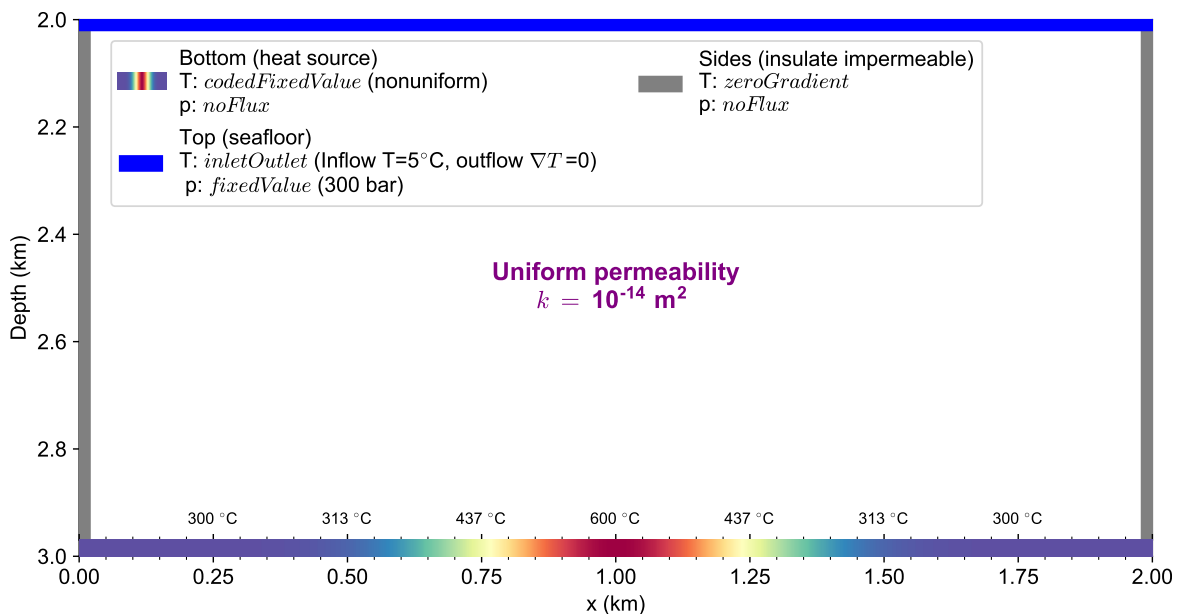


Fig. 5.5: The geometry and boundary conditions of the *Nonuniform fixed temperature BC* model.

The temperature result is shown in Fig. 5.6

5.1.3 Time-dependent permeability

All the input files can be found in [cookbooks/timeDependentPerm](#) directory.

The change of permeability over time, e.g. due to mineral precipitation, is an important process real hydrothermal systems. This example, based on *Hello World: 2D box*, presents how to set time-dependent

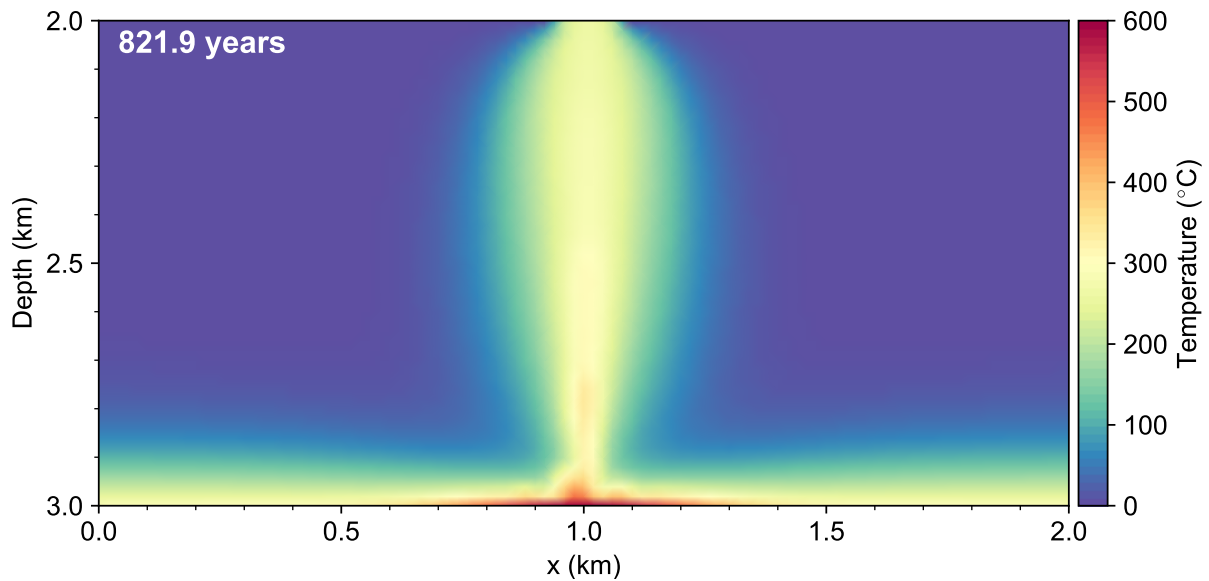


Fig. 5.6: Temperature result of the *Nonuniform fixed temperature BC* model.

permeability at run time in HydrothermalFoam tools. This can be reached using function sub-dictionary in `controlDict` file (see Listing 5.9).

Listing 5.9: Change permeability at run time: `controlDict`.

```

1 functions
2 {
3     changePermeability
4     {
5         libs          ("libutilityFunctionObjects.so");
6         type          coded;
7         enabled       true;
8         writeControl  runtime;
9         writeInterval 86400000;
10        name          changePermeability;
11        codeWrite
12        #{
13            //1. Get simulation time
14            double time=mesh().time().value();
15            //2. Get modifiable pointer of a field variable by name
16            volScalarField& perm_ = const_cast<volScalarField&>(mesh().lookupObject
17            ↪<volScalarField>("permeability"));
18            //3. Increase permeability from 1e-14 to 1e-13 after 500 years in
19            ↪region of y>-2400 m,
20            double year2sec=86400*365;
21            double t0=500*year2sec; // 500 years
22            double wGauss=100*year2sec; // 300 years
23            double kmax=14, kmin=13;
24            double y0=-2400;
25            if(time>t0)

```

(continues on next page)

(continued from previous page)

```

24     {
25         double k = kmin+(kmax-kmin)*exp(-(time-t0)*(time-t0)/
↪ (2*wGauss*wGauss));
26         forAll(perm_, i)
27         {
28             double y = mesh().C()[i].component(1);
29             if(y>y0)
30             {
31                 perm_[i]= pow(10, -k);
32             }
33         }
34     }
35     #};
36 }
37 }

```

The model geometry and boundary conditions are shown in Fig. 5.7, the permeability in shallow region (depth < 2.4 km) will be increase after 500 years (the inset curve).

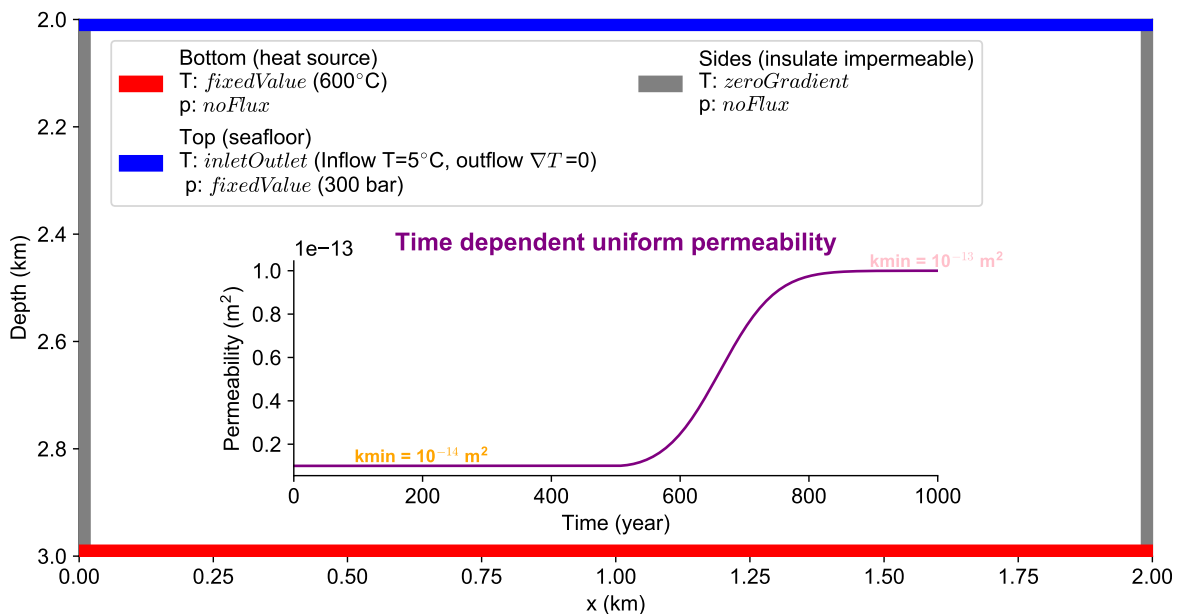


Fig. 5.7: The geometry, BCs and permeability of the *Time-dependent permeability* model.

The temperature and permeability results are shown in Fig. 5.8.

5.1.4 Gmsh

All the input files can be found in [cookbooks/gmsh](#) directory.

This example is based on *Hello World: 2D box*, the only difference is the mesh generation. We use [Gmsh](#) to generate an unstructured triangular mesh. The gmsh script `box.geo` is shown in [Listing 5.10](#).

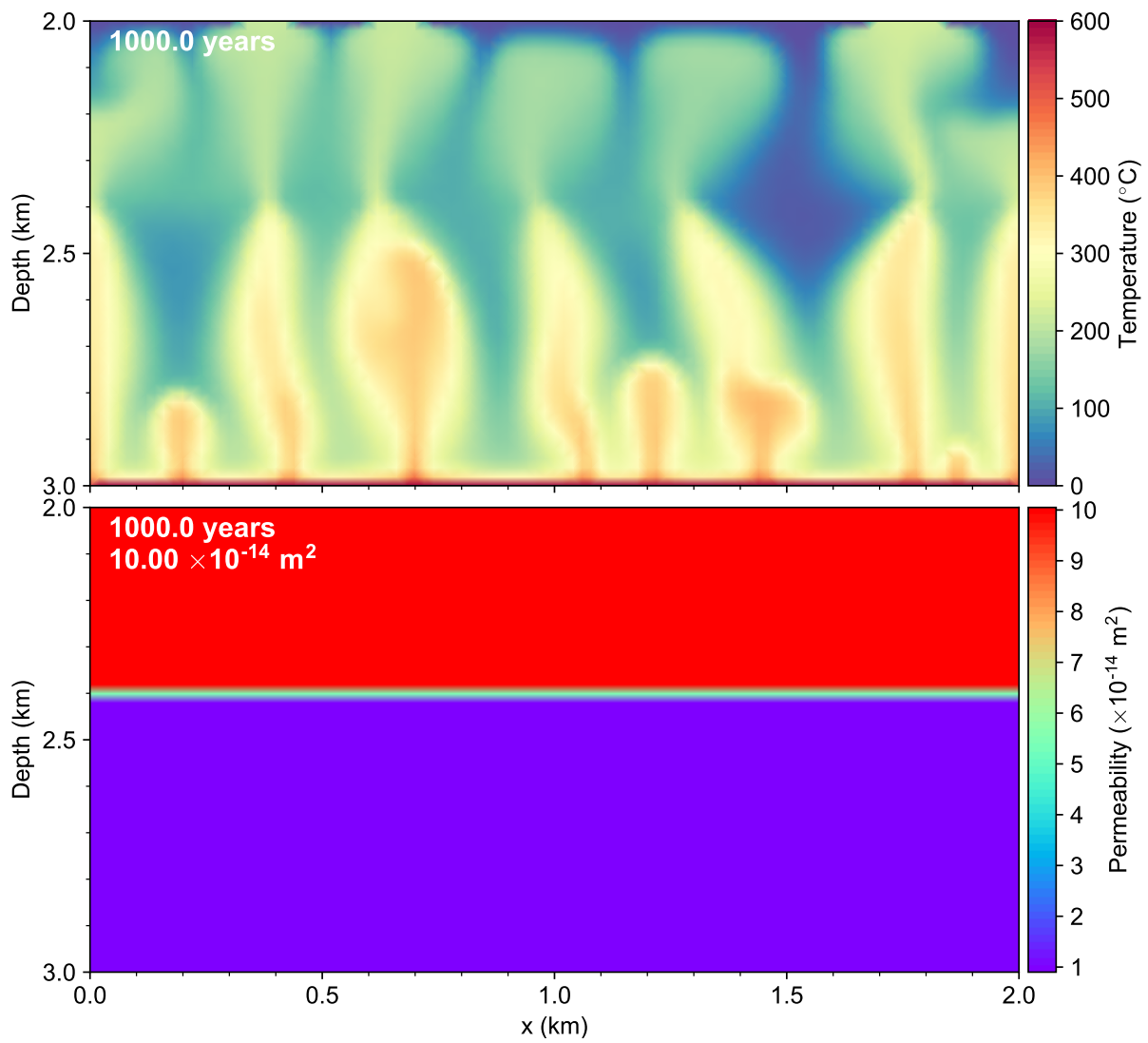


Fig. 5.8: Temperature and permeability result of the *Time-dependent permeability* model.

Listing 5.10: Gmsh geometry script of the *Gmsh* model.

```

1 // 0. define some variables
2 xmin=0;
3 xmax=2000;
4 ymin=-3000;
5 ymax=-2000;
6 zmin=0;
7 zmax=10;
8 lc=20;
9 // 1. define points
10 Point(1) = {xmin, ymax, zmin, lc};
11 Point(2) = {xmax, ymax, zmin, lc};
12 Point(3) = {xmax, ymin, zmin, lc};
13 Point(4) = {xmin, ymin, zmin, lc};
14 // 2. define lines
15 Line(1) = {1, 2};
16 Line(2) = {2, 3};
17 Line(3) = {3, 4};
18 Line(4) = {4, 1};
19 // 3. define line loop and surface
20 Line Loop(6) = {4, 1, 2, 3};
21 Plane Surface(6) = {6};
22 // // 3.1 make regular mesh
23 // Transfinite Surface {6};
24 // Recombine Surface {6};
25 // 4. extrude 2D surface to a 3D volume
26 Extrude {0, 0, zmax} {
27 Surface{6};
28 Layers{1}; //set layer number to 1 for 2D model
29 Recombine;
30 }
31 // 5. define boundary patches via Physical keyword
32 Physical Surface("frontAndBack") = {28,6};
33 Physical Surface("bottom") = {27};
34 Physical Surface("left") = {15};
35 Physical Surface("top") = {19};
36 Physical Surface("right") = {23};
37 // 6. specify a name for cell region which is used for 'setFields'
38 Physical Volume("internal") = {1};

```

Tip: The Gmsh can generate regular mesh as well, see lines 23-24 in Listing 5.10.

The model geometry, boundary conditions and mesh structure are shown in Fig. 5.9.

The temperature result is shown in Fig. 5.10

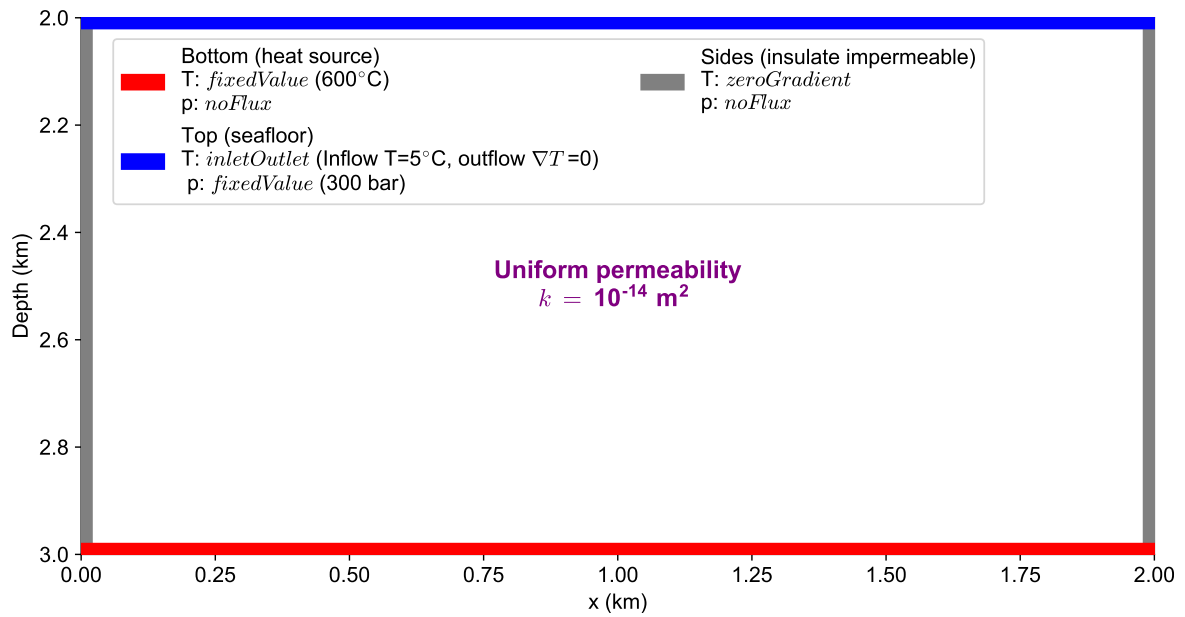


Fig. 5.9: The geometry, boundary conditions and mesh structure of the *Gmsh* model.

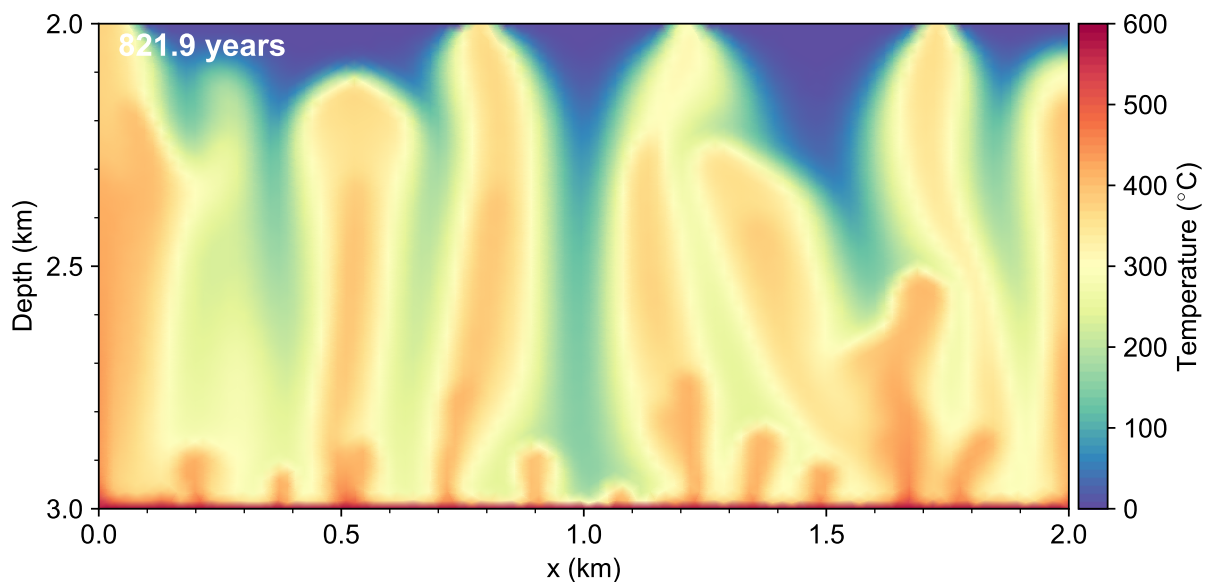


Fig. 5.10: Temperature result of the *Gmsh* model.

5.1.5 3D box

All the input files can be found in `cookbooks/3Dbox` directory.

This example is based on models in [Section 5.1.2](#) and [Section 5.1.4](#). We just to make the following three changes to make a 3D box model,

1. For mesh generation, we just need to change `zmax=xmax`; and change the extrude layers to a number greater than 1, e.g. 40, see line 7 and line 28 in [Listing 5.10](#).
2. For boundary conditions, we just need to change `type` of `frontAndBack` boundary to be consistent with the `left` and `right` boundary, rather than `empty`. And then change `bottom` boundary condition of `T` similar to example [Section 5.1.2](#), see [Listing 5.11](#).
3. Unlike 2D model(e.g.), we don't need to set `frontAndBack` to `empty` patch in `polyMesh/boundary` file. See [Section 4.1.2](#) for empty boundary of 2D model.

Listing 5.11: Bottom boundary condition of `T` of the 3D box model.

```

1 bottom
2 {
3     type          codedFixedValue;
4     value         uniform 873.15; //placeholder
5     name          gaussShapeT;
6     code          #{
7                 scalarField x(this->patch().Cf().component(0));
8                 scalarField z(this->patch().Cf().component(2));
9                 double wGauss=200;
10                double x0=1000;
11                double z0=1000;
12                double Tmin=573;
13                double Tmax=873.15;
14                scalarField T(Tmin+(Tmax-Tmin)*exp(-(x-x0)*(x-x0)+(z-
15                ↪ z0)*(z-z0))/(2*wGauss*wGauss));
16                operator==(T);
17                #};
18 }

```

The mesh is shown in [Fig. 5.11](#) and the bottom boundary condition of temperature is shown in [Fig. 5.12](#).

The isothermal surface of 300 °C , flow arrows and stream lines are shown below.

5.1.6 Parallel computing

All the input files can be found in `cookbooks/3Dbox_par` directory.

Parallel computing is one of features of OpenFOAM, and the `HydrothermalFoam` fully inherits this feature. There are two steps to run a case in Parallel,

1. Add `decomposeParDict` dictionary file into `system` directory (see [Listing 4.14](#)).
2. Decomposition of mesh and initial field data.

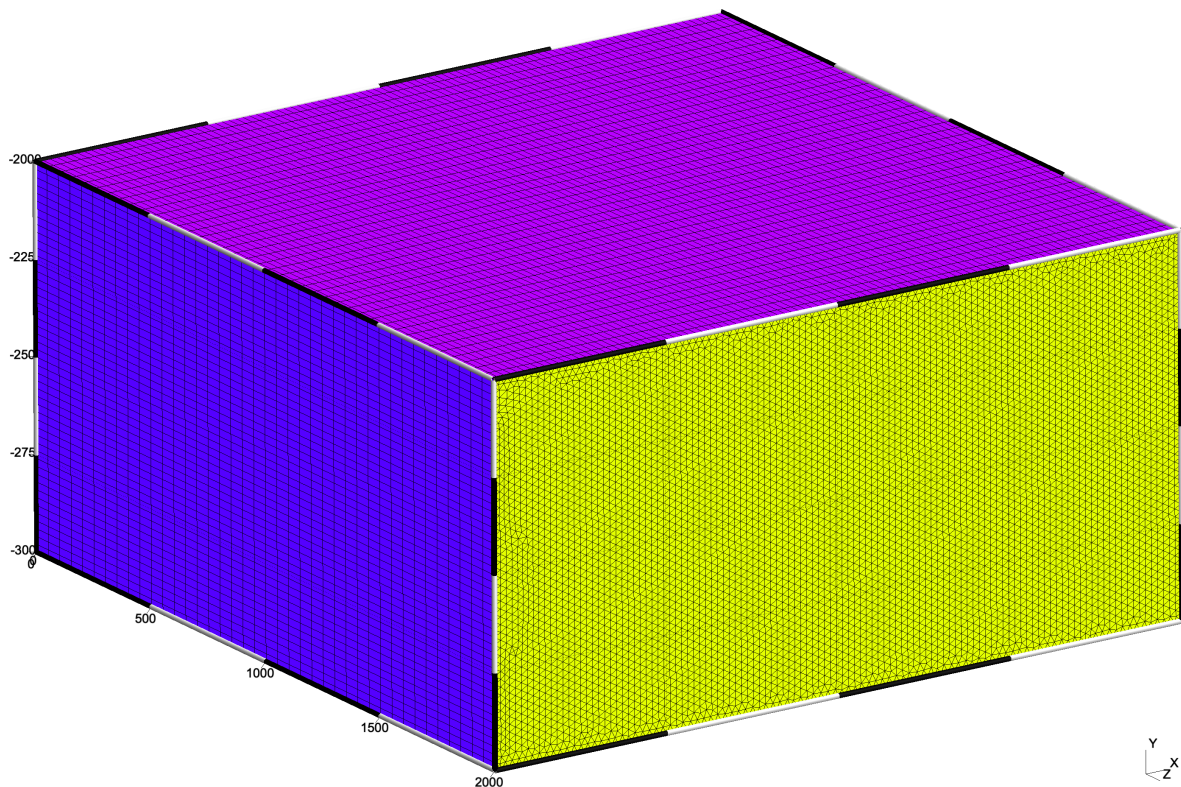


Fig. 5.11: The screenshot of mesh of 3D box model.

3. Replace `runApplication $application` with `runParallel $application`
4. Reconstructing mesh and data.

The last three steps can be assembled in `run.sh` which is highlighted in Listing 5.12

Listing 5.12: Command set of running a case in parallel.

```

1  #!/bin/sh
2  cd ${0%/*} || exit 1    # Run from this directory
3
4  # Source tutorial run functions
5  . $WM_PROJECT_DIR/bin/tools/RunFunctions
6
7  application=`getApplication`
8
9  . clean.sh
10
11 # generate mesh using gmsh
12 gmsh gmsh/box.geo -3 -o gmsh/box.msh -format msh22
13 # convert gmsh to OpenFOAM format
14 gmshToFoam gmsh/box.msh
15 # run solver in parallel
16 runApplication decomposePar
17 runParallel $application

```

(continues on next page)

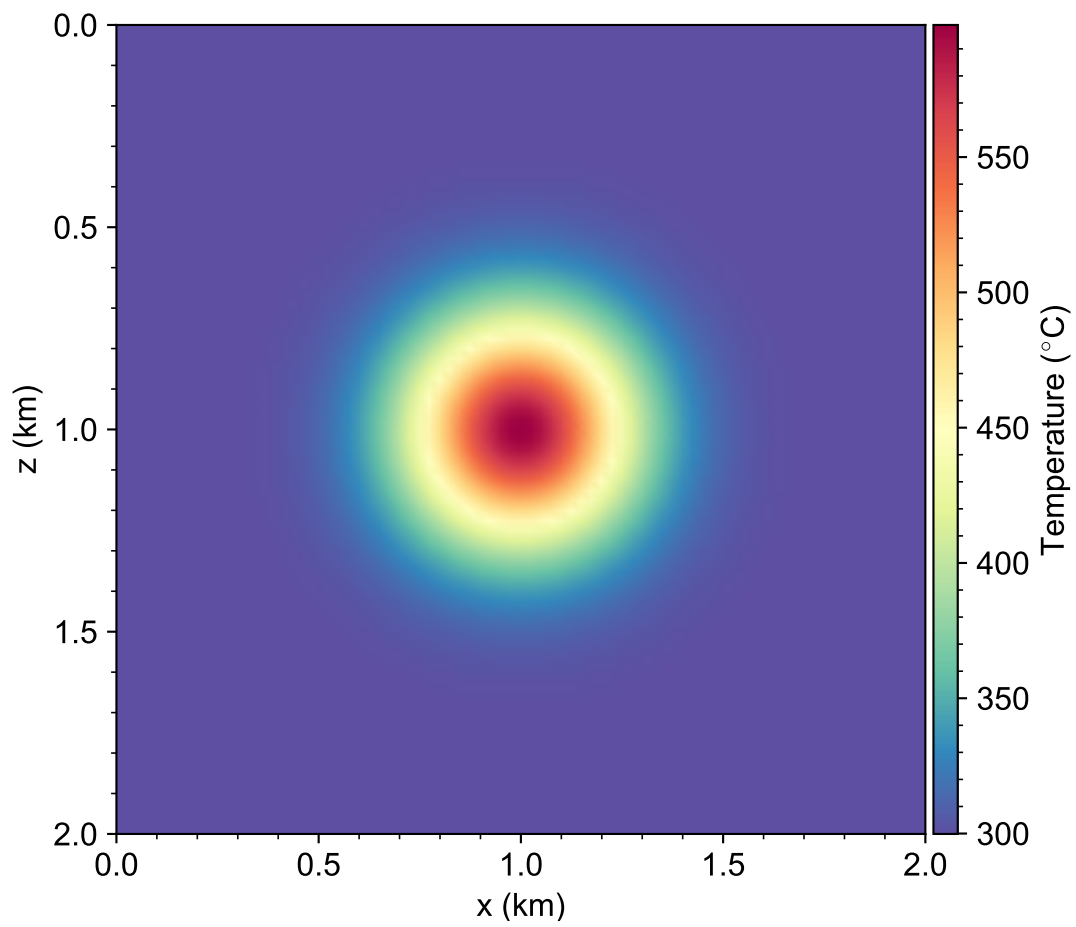


Fig. 5.12: The bottom boundary condition of temperature of the 3D box model.

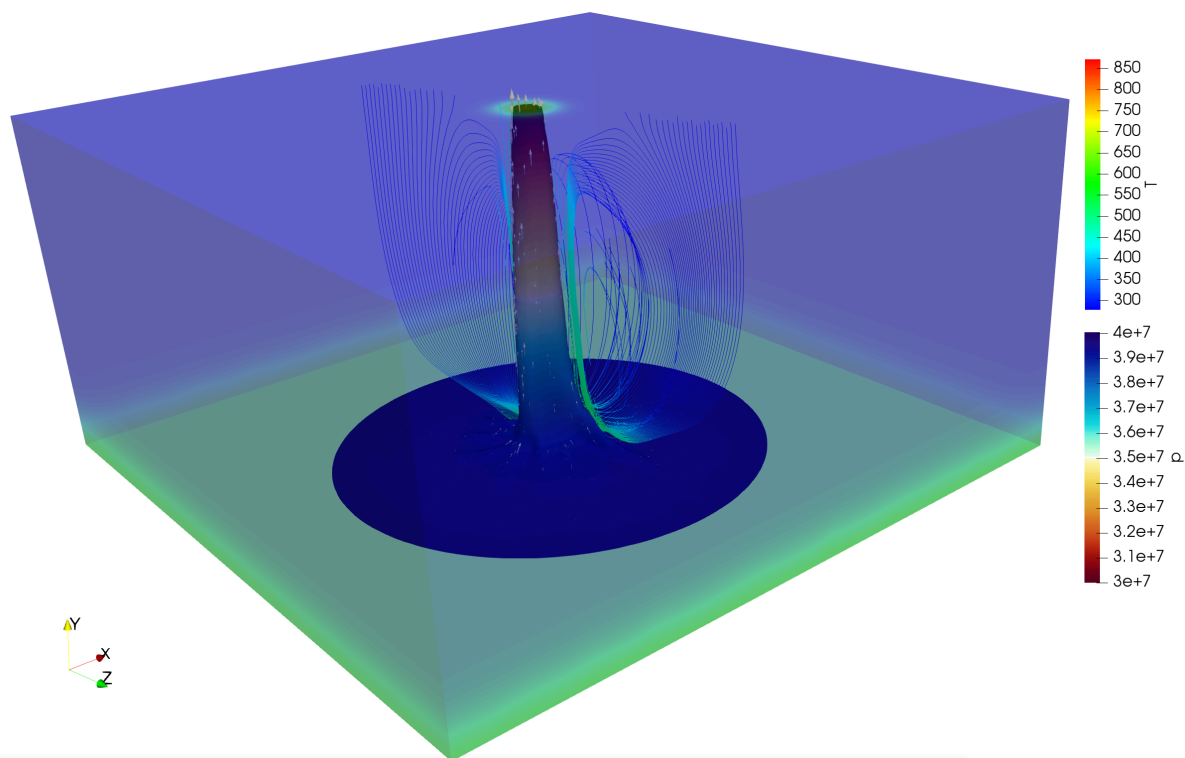


Fig. 5.13: The isothermal surface of 300 °C = 573.15 K, flow arrows and stream lines of the 3D box simulation at 265 year. The unit of temperature in this figure is K.

(continued from previous page)

```
runApplication reconstructPar
```

The mesh will be decomposed into multiple connected regions, the decomposed region number is defined in `decomposeParDict` file. The mesh of this example, which is based on 3D box model in Section 5.1.5, is decomposed into 4 regions by using `decomposePar` command in step 2.

The decomposed mesh of the 3D box model is shown in Fig. 5.14.

5.2 Pipe model

5.2.1 Two-dimensional pipe

All the input files can be found in `cookbooks/pipe` directory.

The pipe model, illustrated in Fig. 5.15, could mimic a simplified scenario of hydrothermal circulation in oceanic crust.

There are five highlights in the pipe model.

1. To simulate a focused upflow zone in the deeper crust (e.g. layer 2B) where permeability is sufficiently low ($k = 10^{-15} \text{ m}^2$) to allow for high-temperature fluid flow, a `hydrothermalMassFluxPressure` ($\phi = 1 \text{ g/m}^2/\text{s}$) boundary condition for ppressure and

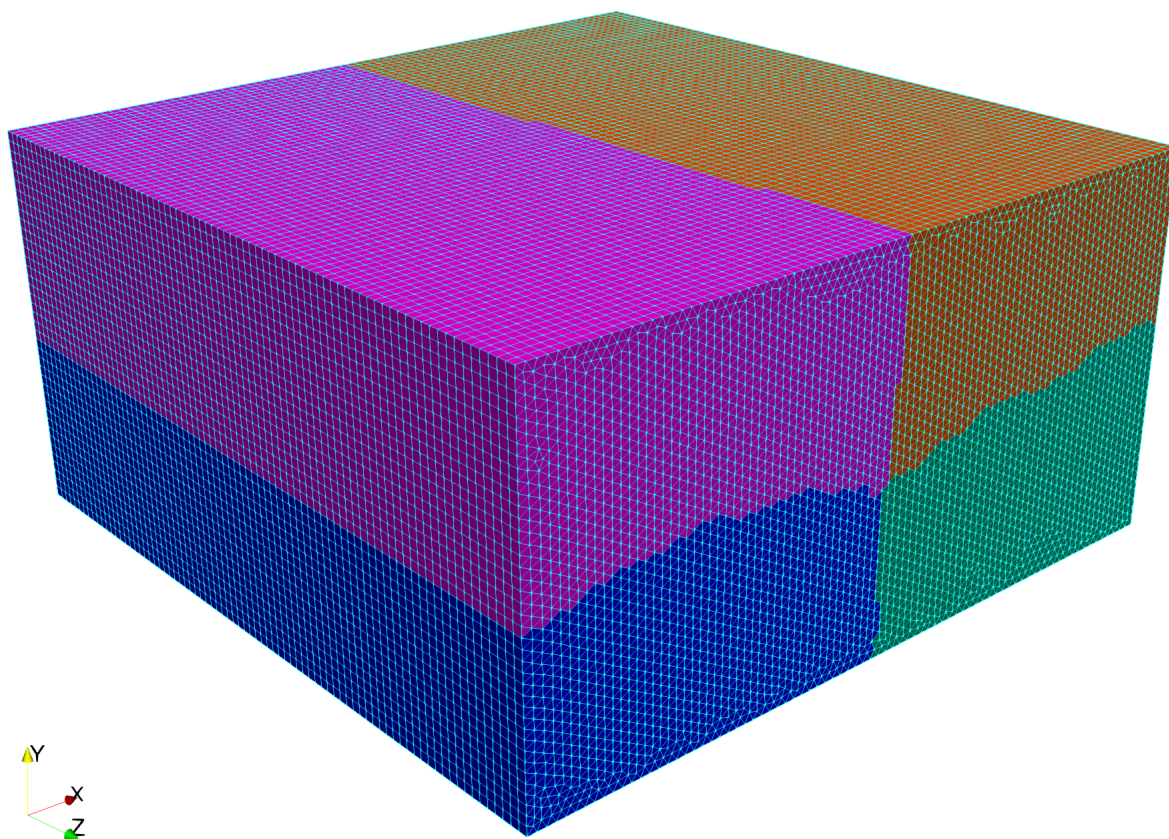


Fig. 5.14: The decomposed mesh of the 3D box.

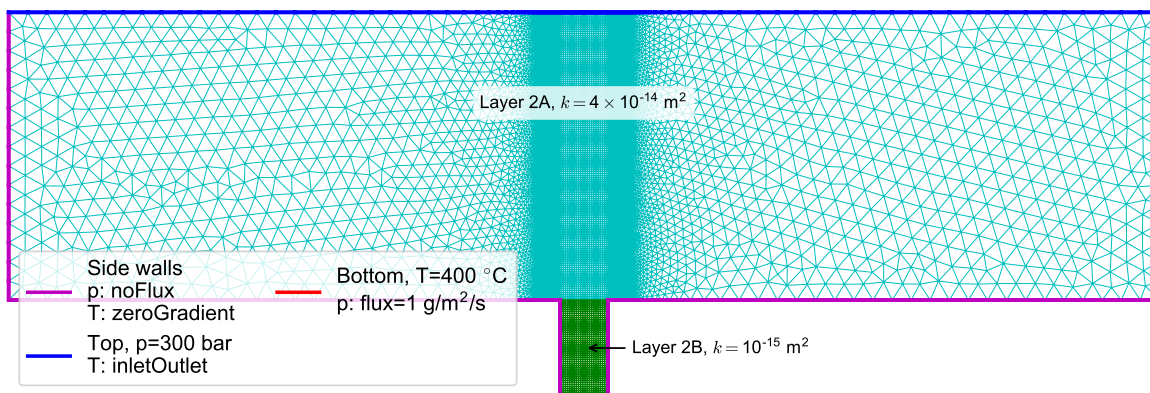


Fig. 5.15: Mesh and boundary condions of the pipe model.

`fixedValue` ($T_{bot} = 400$ °C) boundary condition for temperature are applied on the pipe bottom.

2. Hybrid mesh, triangular mesh and regular mesh, is generated by `gmsh` . The mesh in the central zone is regular.
3. Different resolution of mesh is set by `gmsh` .
4. Permeability field is specified with different value for layer 2A and layer 2B by `zoneToCell` keyword in `setFieldDict` file.
5. To simulate hydrothermal flow discharge out of seafloor, a `inletOutlet` boundary condition of temperature is applied on the pipe top.

The temperature result is shown in Fig. 5.16.

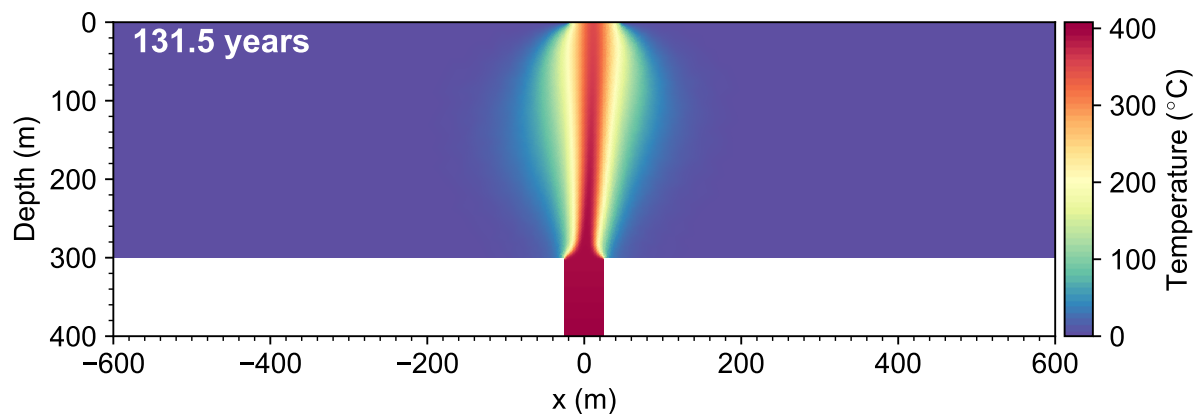


Fig. 5.16: Temperature result of the *Two-dimensional pipe* model.

5.2.2 Three-dimensional pipe

All the input files can be found in `cookbooks/pipe_3D` directory.

The three-dimensional pipe model, illustrated in Fig. 5.17, is the result of a two-dimensional model(see Section 5.2.1) rotating around the central axis. Of course one can extrude mesh of *Two-dimensional pipe* model to get a three-dimensional pipe model(see Fig. 5.18).

Note: The boundary conditions setup is similar to the *Two-dimensional pipe* model. One can also run this example in parallel, see Section 5.1.6. The result is shown below.

The isothermal surface of 200 °C , flow arrows and stream lines are shown below.

5.3 Single pass model

5.3.1 Two-dimensional single pass model

All the input files can be found in `cookbooks/singlepass` directory.

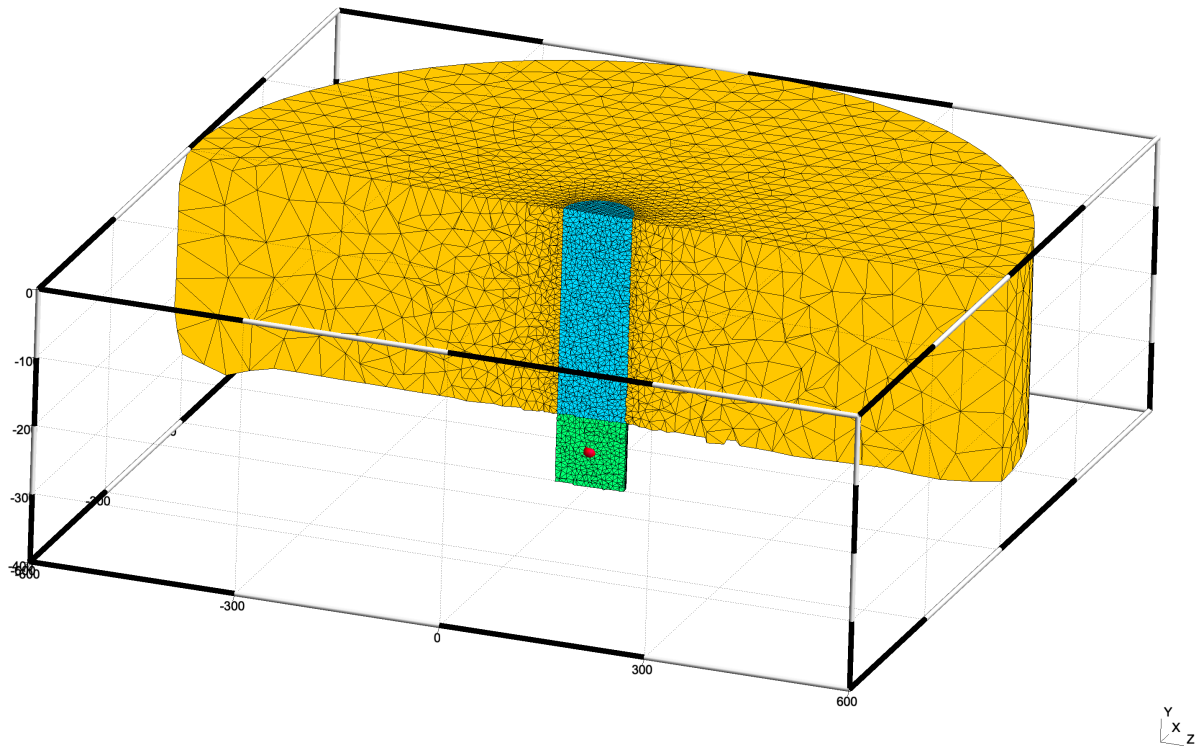


Fig. 5.17: Mesh of the 3D pipe model.

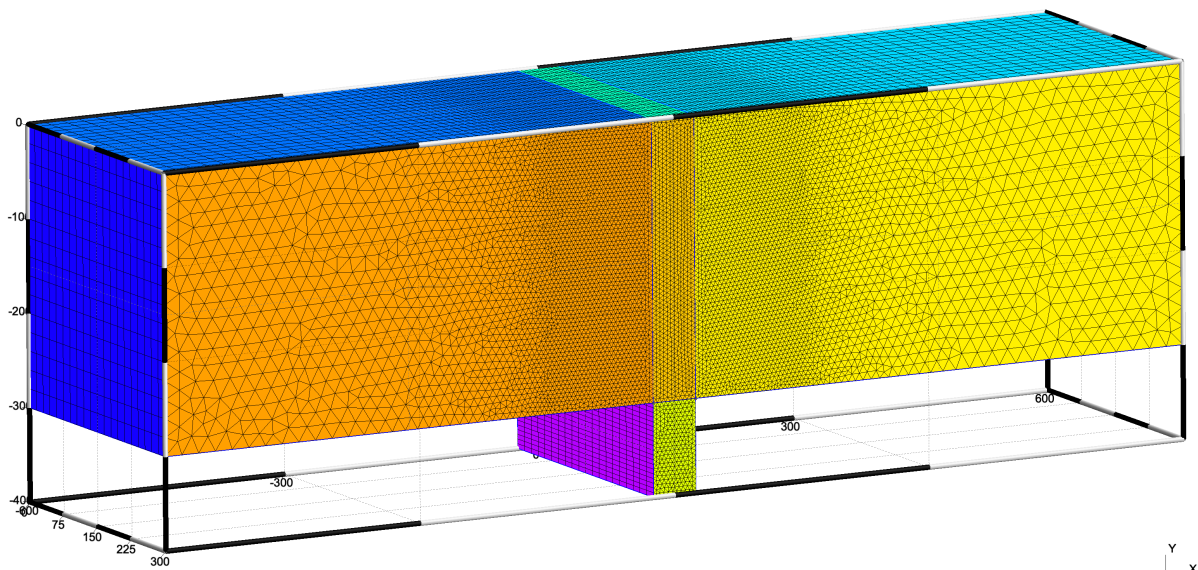


Fig. 5.18: Alternative mesh of the 3D pipe model.

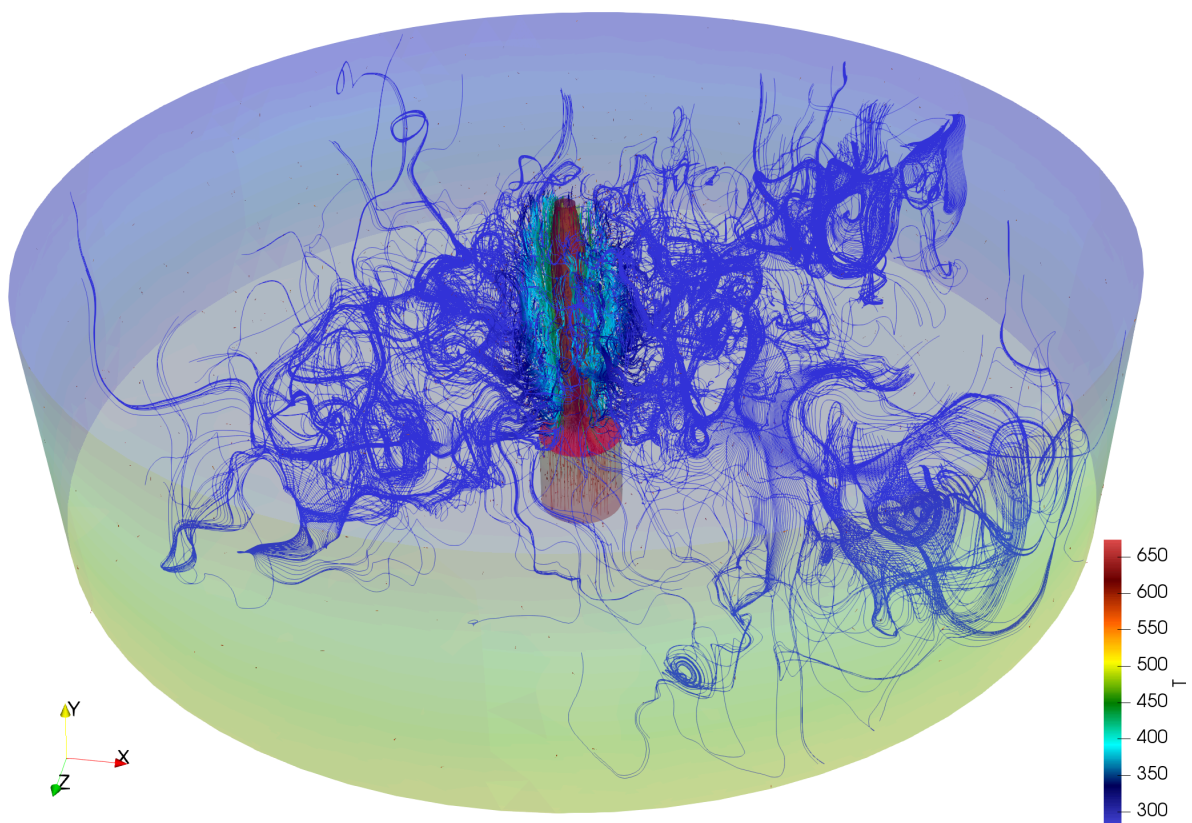


Fig. 5.19: The isothermal surface (red) of $200\text{ }^{\circ}\text{C} = 473.15\text{ K}$, flow arrows and stream lines of the 3D pipe simulation at 80 year. The unit of temperature in this figure is K.

One- and two-limb single-pass models are usually used to determine vent field characteristics such as mass flow rate Q , bulk permeability in the discharge zone k_d , thickness of the conductive boundary layer at the base of the system d , magma replenishment rate, and residence time in the discharge zone [Lowell et al., 2013].

One-limb classical single pass model

A schematic of the one- and two-limb single-pass model are shown in Fig. 5.20.

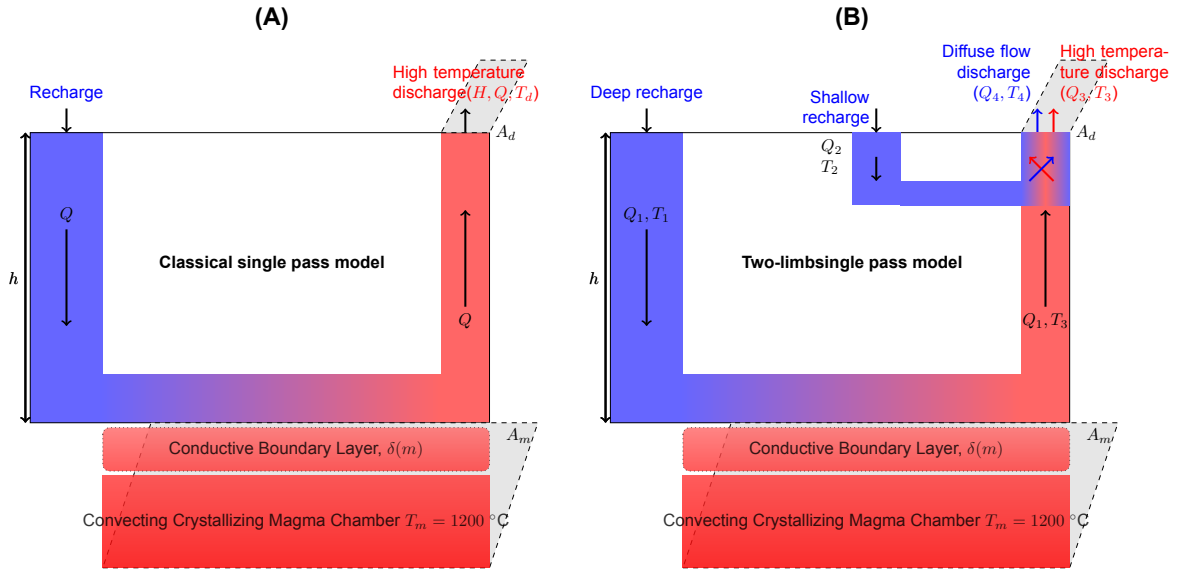


Fig. 5.20: Schematic of the one-limb (A) and two-limb (B) single-pass model (reproduced from [Lowell et al., 2013]).

Here we present the one-limb single pass mode simulation using HydrothermalFoam, the model geometry, mesh and boundary conditions are shown in Fig. 5.21.

The model setup is similar to [Lowell et al., 2007]. The temperature evolution and streamlines of the *Two-dimensional single pass model* are in Fig. 5.22.

If we consider the recharge zone on the right side in single pass model shown in Fig. 5.21, the hydrothermal circulation pattern would be different, a schematic of the full single pass model is shown in

Here we present the full single pass mode simulation using HydrothermalFoam, the model geometry, mesh and boundary conditions are shown in Fig. 5.24.

The temperature evolution and streamlines of the full *Two-dimensional single pass model* are in Fig. 5.22.

Two-limb single pass model

All the input files can be found in cookbooks/singlepass_twolimb directory. The model geometry, mesh and boundary conditions are shown in Fig. 5.26

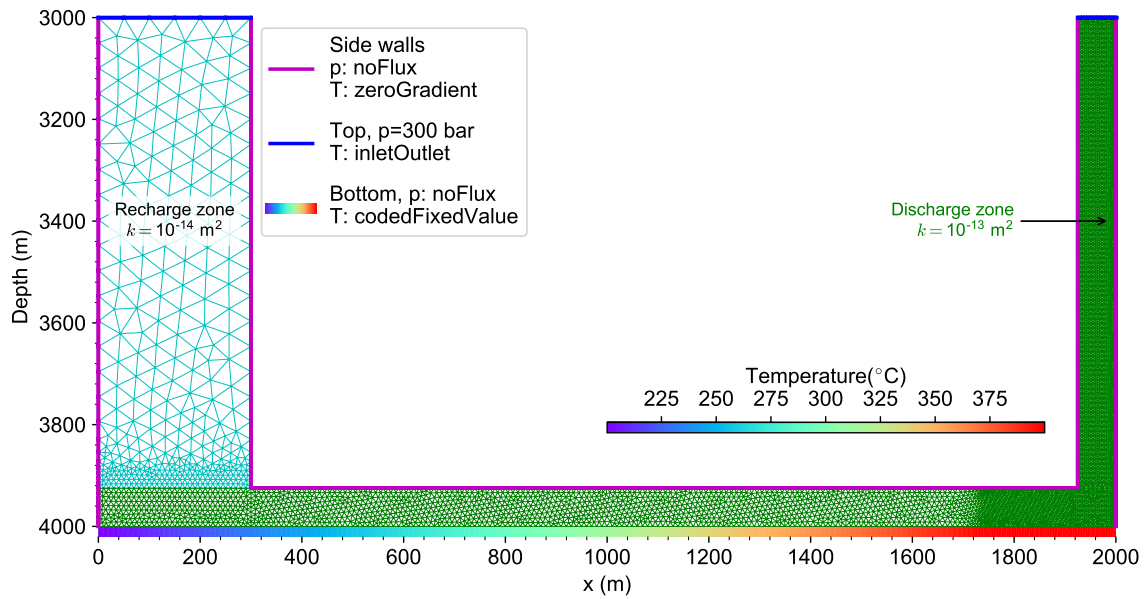


Fig. 5.21: Model geometry, mesh and boundary conditions of the 2D single pass model.

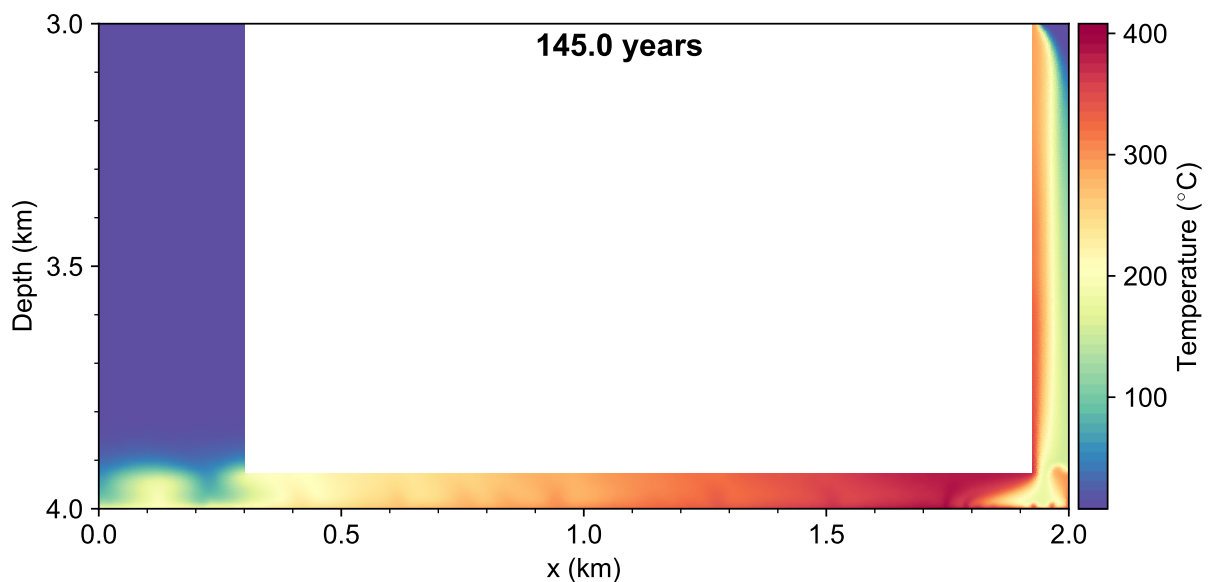


Fig. 5.22: Temperature result of the *Two-dimensional single pass model*.

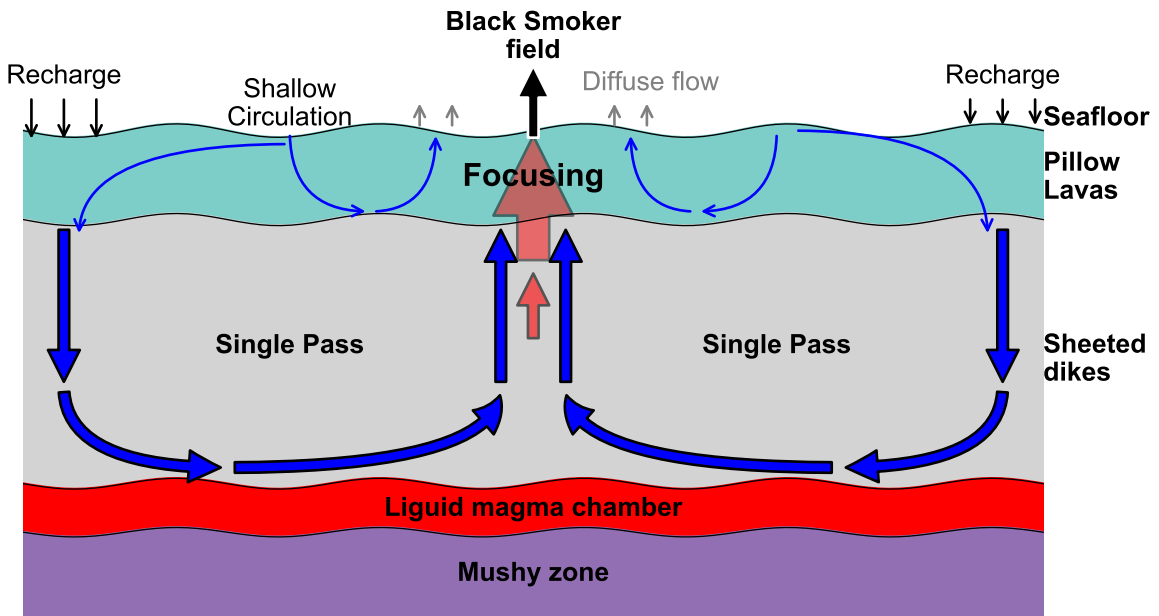


Fig. 5.23: Schematic of the full single-pass model (reproduced from [Lowell et al., 2014]).

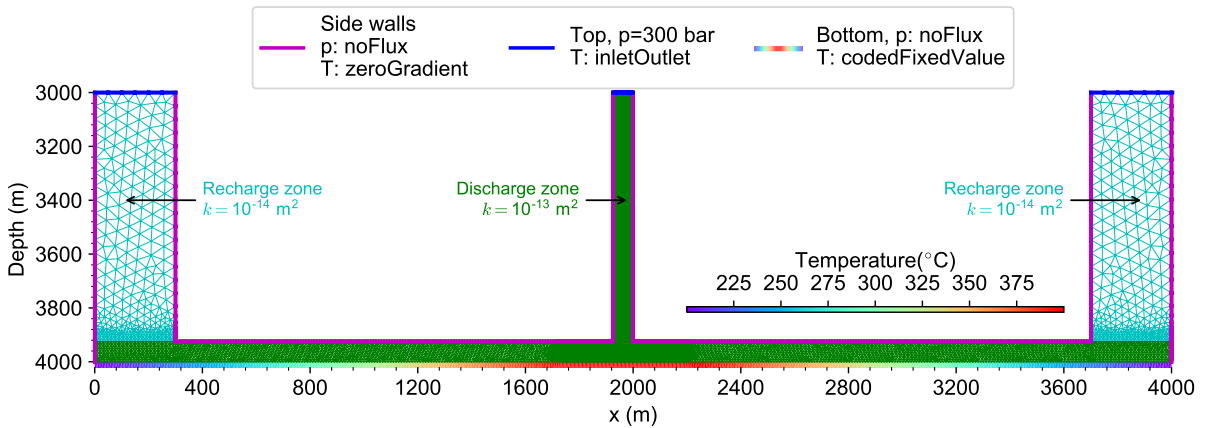


Fig. 5.24: Model geometry, mesh and boundary conditions of the improved 2D single pass model.

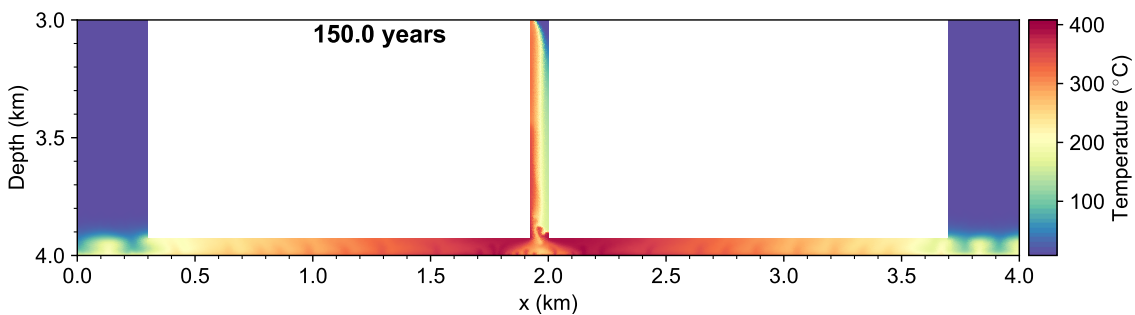


Fig. 5.25: Temperature result of the *Two-dimensional single pass model*.

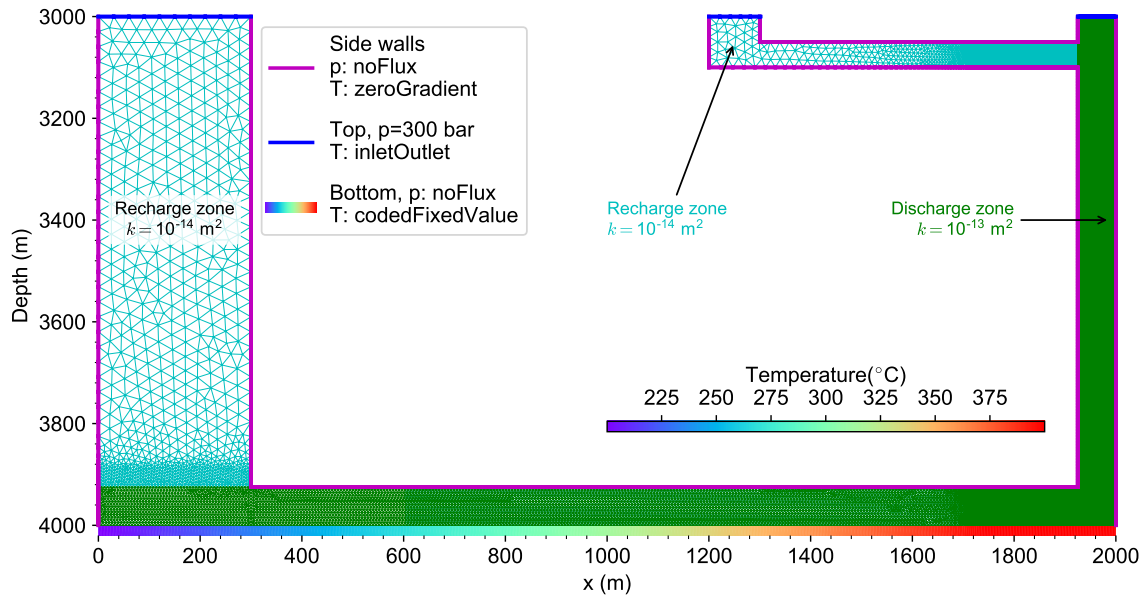


Fig. 5.26: Model geometry, mesh and boundary conditions of the two-dimensional two-limb single pass model.

The temperature evolution and streamlines of the two-limb single pass model are in Fig. 5.27.

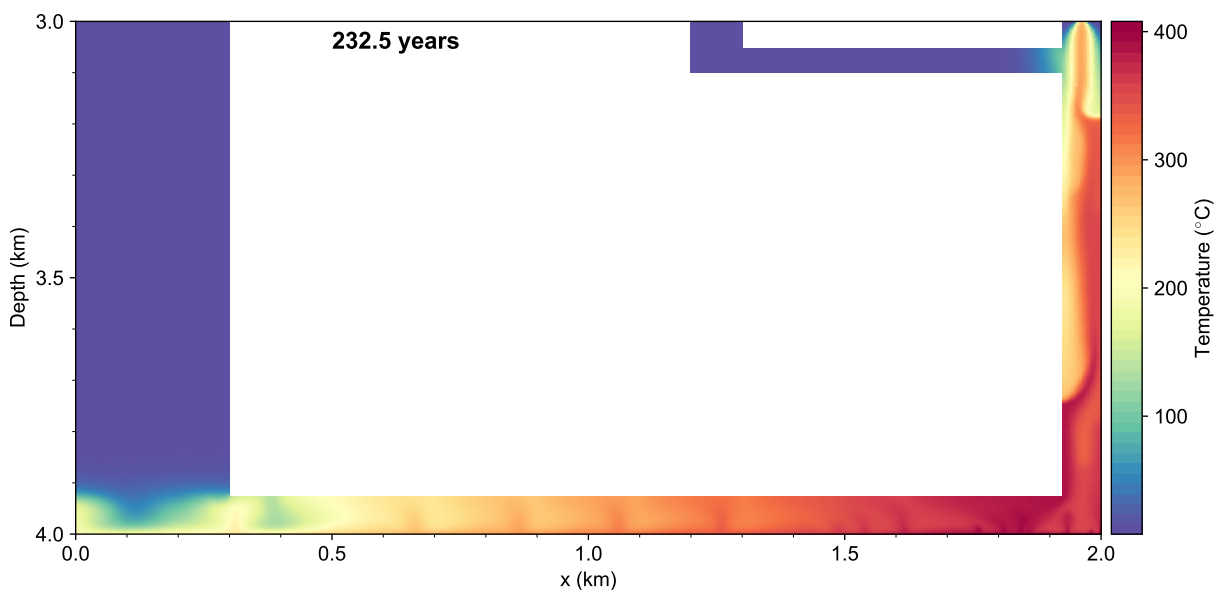


Fig. 5.27: Temperature result of the *Two-dimensional single pass model*.

BIBLIOGRAPHY

- [Lowell et al., 2013] Robert P. Lowell, Aida Farough, Joshua Hoover, and Kylin Cummings. Characteristics of magma-driven hydrothermal systems at oceanic spreading centers. *Geochemistry, Geophysics, Geosystems*, 14(6):1756–1770, jun 2013. doi:[10.1002/ggge.20109](https://doi.org/10.1002/ggge.20109).
- [Lowell et al., 2007] Robert P Lowell, Sawyer Gosnell, and Yang Yang. Numerical simulations of single-pass hydrothermal convection at mid-ocean ridges: Effects of the extrusive layer and temperature-dependent permeability. *Geochemistry, Geophysics, Geosystems*, oct 2007. doi:[10.1029/2007GC001653](https://doi.org/10.1029/2007GC001653).
- [Lowell et al., 2014] R.P. Lowell, K. Kolandaivelu, and P.A. Rona. Hydrothermal activity. In *Reference Module in Earth Systems and Environmental Sciences*. Elsevier, 2014. doi:<https://doi.org/10.1016/B978-0-12-409548-9.09132-6>.
- [Hasenclever et al., 2014] Jörg Hasenclever, Sonja Theissen-Krah, Lars H Rüpke, Jason P Morgan, Karthik Iyer, Sven Petersen, and Colin W Devey. Hybrid shallow on-axis and deep off-axis hydrothermal circulation at fast-spreading ridges. *Nature*, 508(7497):508–512, 2014. doi:[10.1038/nature13174](https://doi.org/10.1038/nature13174).