Free convection in a chamber with heating from bottom

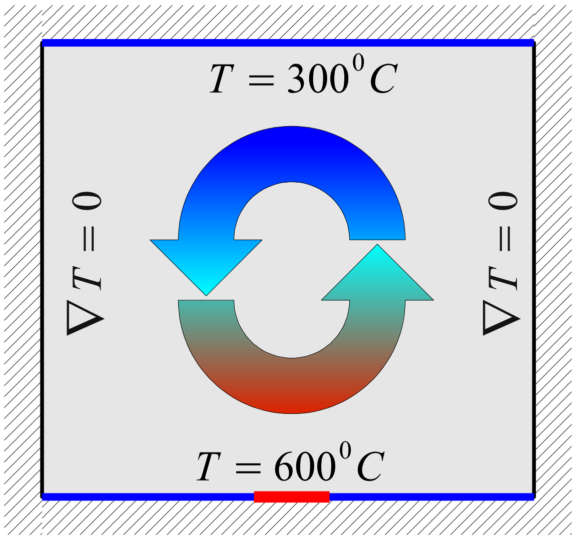
Sergei Strijhak (ISP RAS, Moscow, Russia)

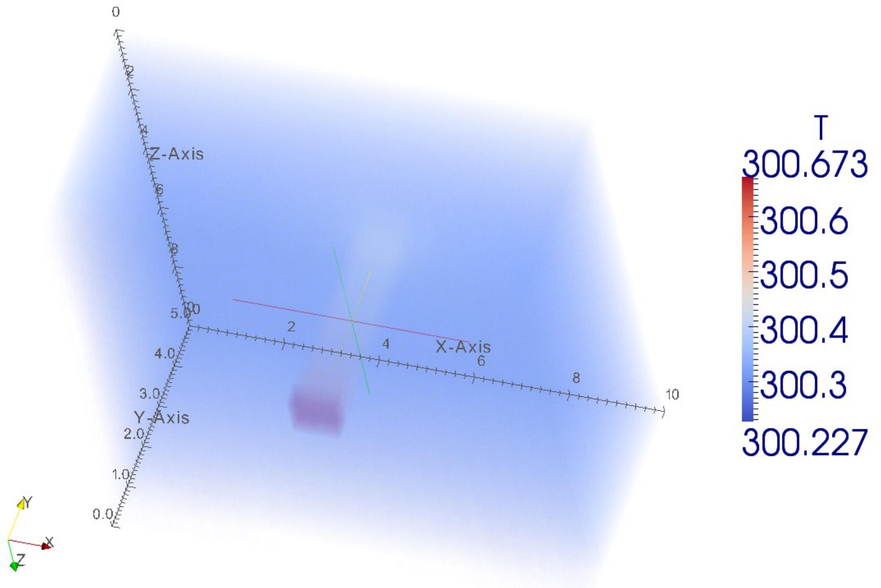
*27.06.*2016

Free convection in a chamber with heating from bottom

A flow of compressible liquid (air) with subsonic velocity under the action of the buoyant force (according to the Archimedes' principle) in a cubic closed volume is examined.

The buoyant force appears as a result of medium heating in some area of the lower wall.





FREE CONVECTION — GOALS AND OBJECTIVES

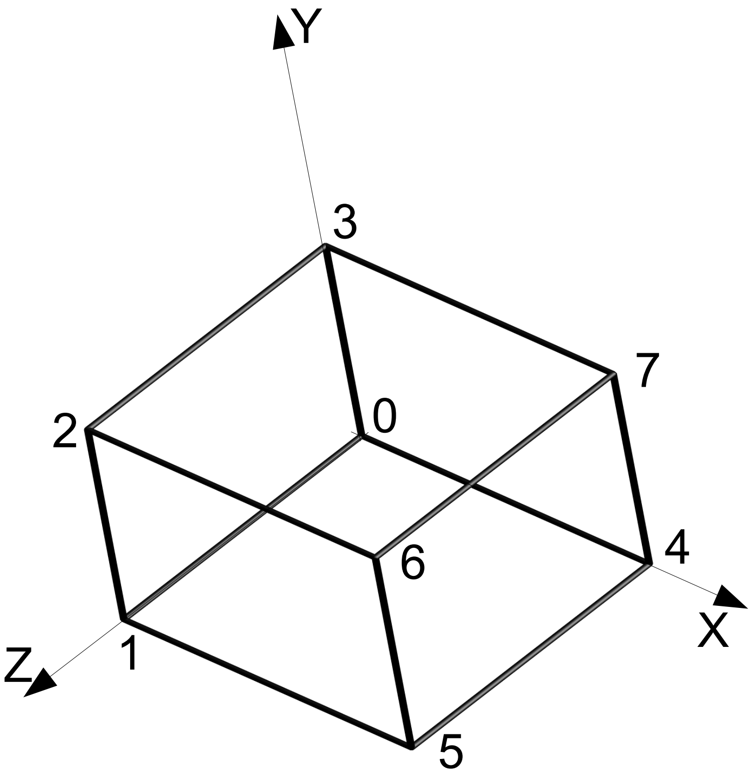
In this example we'll see:

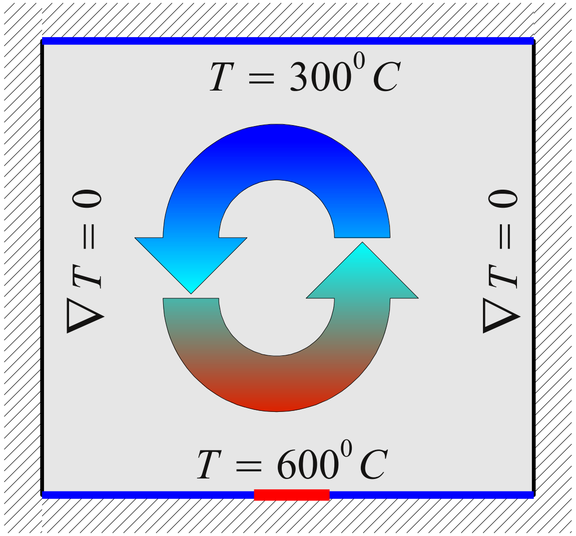
- *How to set up the computational model for compressible problem solving, what input data are necessary for this;*
- *How to realize the computation with heat transfer and what parameters of the computational scheme to use;*
- *How to execute the steady state calculations (SIMPLE method);*
- *How to set up a non-uniform distribution of value over the space of boundaries with the help of user OpenFOAM utilities*

FREE CONVECTION — MESH CONSTRUCTION

Computational domain — hexahedron of dimensions 10x5x10 (XYZ). The lower plane is heated from bottom, the upper one cools the chamber, the other walls are adiabatic.

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (20 10 20) simpleGrading (1 1 1)
);
```






```
convertToMeters 1;

vertices
(
    (0 0 0)
    (10 0 0)
    (10 5 0)
    (0 5 0)
    (0 0 10)
    (10 0 10)
    (10 5 10)
    (0 5 10)
);
```

FREE CONVECTION — BOUNDARIES

Computational domain — hexahedron of dimensions 10x5x10 (XYZ). The lower plane is heated from bottom, the upper one cools the chamber, the other walls are adiabatic.

```
patches
(
    wall floor
    (
        (1 5 4 0)
    )
    wall ceiling
    (
        (3 7 6 2)
    )
    wall fixedWalls
    (
        (0 4 7 3)
        (2 6 5 1)
        (0 3 2 1)
        (4 5 6 7)
    )
);
```

Lower wall (with heating from the center). Temperature assignment

Upper wall (cooling). Temperature assignment

Other walls are adiabatic. Assignment of zero temperature gradient

FREE CONVECTION — BOUNDARY CONDITIONS (1)

*1. Velocity **U**. As the liquid doesn't enter the computational domain and doesn't leave it, the slip condition — equality to zero of the velocity vector — is assigned on all the walls.*

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    floor
    {
        type                fixedValue;
        value               uniform (0 0 0);
    }

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

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

FREE CONVECTION — BOUNDARY CONDITION (2)

2. Pressure p. As the liquid doesn't enter the computational domain and doesn't leave it, the slip condition — equality to zero of the velocity vector — is assigned on all the walls.

In OpenFoam 1.7.1 for buoyancy problem solving there are two pressures: hydrostatic (p), and the second surplus, devoid of the product $\rho g h$

*For the first pressure the BC is **calculated**, for the second one the setting is **buoyantPressure***

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

internalField   uniform 1e5;

boundaryField
{
    floor
    {
        type            calculated;
        value           $internalField;
    }

    ceiling
    {
        type            calculated;
        value           $internalField;
    }

    fixedWalls
    {
        type            calculated;
        value           $internalField;
    }
}
```

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

internalField   uniform 1e5;

boundaryField
{
    floor
    {
        type            buoyantPressure;
        value           uniform 1e5;
    }

    ceiling
    {
        type            buoyantPressure;
        value           uniform 1e5;
    }

    fixedWalls
    {
        type            buoyantPressure;
        value           uniform 1e5;
    }
}
```

FREE CONVECTION — BOUNDARY CONDITIONS (3)

3. Turbilent model's fields — k (turbulence kinetic energy), epsilon (dissipation of turbulence kinetic energy), alphat and mut — turbulent diffusion and turbulent dynamic viscosity coefficients respectively. For all the four values the wall-functions are applied, hence the BC can be written the next way:

```
type            compressible::kqRWallFunction;
value           uniform 0.1;
```

```
type            compressible::epsilonWallFunction;
value           uniform 0.01;
```

```
type            mutWallFunction;
value           uniform 0;
```

```
type            alphatWallFunction;
value           uniform 0;
```

*Before the type definition of k and epsilon (or of an other value) you need to put **compressible::** to destinguish them from the uncompressible wall-functions. For mut and alphat it isn't requiered*

FREE CONVECTION — BOUNDARY CONDITIONS (4)

3. Temperature T. In this problem there will be two temperature fields — T.org (original) and T, that will be used in calculations. The last differs from the first one by non-uniform temperature distribution on the lower wall (with the maximum in the center).

Dimensions — K (Kelvins),
Initial condition in the volume - 300K

```
dimensions      [0 0 0 1 0 0 0];

internalField   uniform 300;
```

```
boundaryField
{
```

Lower wall — uniform 300K (T.org) on the whole surface, afterwards - 600K on the center, 300K on the other cells

```
    floor
    {
        type            fixedValue;
        value           uniform 300;
    }
```

Upper wall — uniform 300K on the whole surface

```
    ceiling
    {
        type            fixedValue;
        value           uniform 300;
    }
```

Adiabatic side walls — zero gradient

```
    fixedWalls
    {
        type            zeroGradient;
    }
}
```

FREE CONVECTION — BOUNDARY CONDITIONS (4)

3. Temperature T. In this problem there will be two temperature fields — T.org (original) and T, that will be used in calculations. The last differs from the first one by non-uniform temperature distribution on the lower wall (with the maximum in the center).

Dimensions — К (Kelvins),
Initial condition in the volume - 300K

Lower wall — uniform 300K (T.org) on the whole surface, afterwards - 600K on the center, 300K on the other cells

Upper wall — uniform 300K on the whole surface

Adiabatic side walls — zero gradient

```
dimensions      [0 0 0 1 0 0 0];

internalField   uniform 300;

boundaryField
{
    floor
    {
        type            fixedValue;
        value           uniform 300;
    }

    ceiling
    {
        type            fixedValue;
        value           uniform 300;
    }

    fixedWalls
    {
        type            zeroGradient;
    }
}
```

FREE CONVECTION — BOUNDARY CONDITIONS (4)

3. Temperature T. In this problem there will be two temperature fields — T.org (original) and T, that will be used in calculations. The last differs from the first one by non-uniform temperature distribution on the lower wall (with the maximum in the center).

Dimensions — K (Kelvins),
Initial condition in the volume - 300K

```
dimensions      [0 0 0 1 0 0 0];

internalField   uniform 300;
```

Lower wall — uniform 300K (T.org) on the whole surface, afterwards - 600K on the center, 300K on the other cells

Upper wall — uniform 300K on the whole surface

Adiabatic side walls — zero gradient

```
boundaryField
{
    floor
    {
        type            fixedValue;
        value           uniform 300;
    }

    ceiling
    {
        type            fixedValue;
        value           uniform 300;
    }

    fixedWalls
    {
        type            zeroGradient;
    }
}
```

FREE CONVECTION — BOUNDARY CONDITIONS (5)

To construct a non-uniform tempreture field on the lower wall we'll use the setHotRoom utility, its initial code is located in the example's folder.

The initial code of every OpenFOAM application necessarily contains the next files:

- *Make catalogue — files controlling the assembly of the package by means of the wmake utility.*
- *Make/options — compilation and assembly options, that are communicated to the wmake utility*
- *Make/files — list of compiled files and name of the executed module*
- *<Programme_name>.C — at the least one of the initial files must be mentioned in Make/files*

Make/files

Name of the compiled file

```
setHotRoom.C
```

Location of the exe-file

```
EXE = $(FOAM_USER_APPBIN)/setHotRoom
```

Make/options

Compilation options

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

Assembling options

```
EXE_LIBS = \
    -lfiniteVolume
```

FREE CONVECTION — BOUNDARY CONDITIONS (6)

The initial code of the application setHotRoom.C is typical for C++ programmes, first of all we link up the heading files:

```
#include "fvCFD.H"
#include "OSspecific.H"
#include "fixedValueFvPatchFields.H"
                                ......
```

Main procedure (enter point)

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

Mandatory stages of the initialization:

```
#   include "setRootCase.H"
```

Set-up of the file system parameters

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

construction of the time counter (physical)
mesh construction (loading to the memory)
construction (reading) of the essential values' fields

FREE CONVECTION — BOUNDARY CONDITIONS (7)

More in detail about createFields.H and its content:

```
Info<< "Reading field T\n" << endl;
  volScalarField T
  (
      IOobject
      (
          "T",
          runTime.timeName(),
          mesh,
          IOobject::MUST_READ,
          IOobject::AUTO_WRITE
      ),
      mesh
  );
```

FREE CONVECTION — BOUNDARY CONDITIONS (8)

In the body of the main(...) function setHotRoom.C performs the procedure of initialization of the local temperature field values on the surface «floor».

```
// List of all the outer surfaces of the model
volScalarField::GeometricBoundaryField& Tpatches = T.boundaryField();

// FORloop for all surfaces
forAll(Tpatches, patchI)
{
// If the the surface name is «floor»
if
    (
        isA<fixedValueFvPatchScalarField>(Tpatches[patchI])
     && mesh.boundaryMesh()[patchI].name() == "floor"
    )
    {

//Get the list of face centers of this surface
       fixedValueFvPatchScalarField& Tpatch =
            refCast<fixedValueFvPatchScalarField>(Tpatches[patchI]);

        const vectorField& faceCentres =
            mesh.Cf().boundaryField()[patchI];
```

FREE CONVECTION — BOUNDARY CONDITIONS (9)

For all the faces with the center corresponding to 4.5<Xc<5.5 and 4.5<Zc<5.5 we set the local temperature 600K

```
        forAll(faceCentres, facei)
        {
            if
            (
                (faceCentres[facei].x() > 4.5) &&
                (faceCentres[facei].x() < 5.5) &&
                (faceCentres[facei].z() > 4.5) &&
                (faceCentres[facei].z() < 5.5)
            )
            {
                Tpatch[facei] = 600;
            }
            else
            {
                Tpatch[facei] = 300;
            }
        }
    };
```

FREE CONVECTION — BOUNDARY CONDITIONS (10)

Finally, we proceed writing of the temperature fields to the file and return to the operating system

```
Info<< "Writing modified field T\n" << endl;
T.write();

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

return 0;
```

To compile the programme it is necessary to move to the folder with the initial code in the command line and execute ***wmake***

To initialize a non-uniform temperature field you need to do the next:

- *Move the content of the file T.org in T:* ***cat T.org > T***
- *Run* ***setHotRoom*** *utility*
- *Not forget to control the mesh —* ***checkMesh****!!!*

FREE CONVECTION — CONSTANT ENVIRONMENT SET-UP(1)

During heat transfer problem solving you need to regulate the equation of state. OpenFOAM uses only the Clapeyron-Mendeleev equation p/V=nRT

*All other properties depend on this above dependence. Thermophysical properties are assigned in **constant/thermophysicalProperties***

```
thermoType
hRhoThermo<pureMixture<constTransport<specieThermo<hConstThermo<perfectGas>>>>>;

mixture         air 1 28.9 1000 0 1.8e-05 0.7;

pRef            100000;
```

Entry thermoType can be interpreted as:

hrhoTermo — properties depend on enthalpy, density (rho) is a function of T and p
pureMixture — specificator by default (there is only one liquid type)
constTransport — constant viscosity(1.8e-5)
specieThermo<hConstThermo<...> - constant basic enthalpy, h=h0+dT(dh/dT)*

1 mol of a substance with the molar weight 28.9, isobaric heat capacity 1000, initial enthalpy 0, viscosity 1.8e-5 and Prt=0.7

FREE CONVECTION — CONSTANT ENVIRONMENT SET-UP (2)

On the next stage the method of turbulence modelling is defined. As far as the problem is steady only the RAS (Reynolds Averaged Stresses) method is available. File — constant/turbulenceProperties.

```
simulationType  RASModel;
```

After defining the turbulence model's class we define its type (in this example — k-e), file constant/RASProperties

```
RASModel        kEpsilon;

turbulence      on;

printCoeffs     on;
```

RASModel — model type (laminar, kEpsilon, kOmegaSST, kOmega, realizableKE)

turbulence — will we use or not the RAS model to calculate the stress tensor

printCoeffs — do we need to print the model's coefficients?

FREE CONVECTION — CONSTANT ENVIRONMENT SETTINGS (3)

Finally, we define the free fall acceleration vector's direction (file constant/g)

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.7.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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


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

FREE CONVECTION: SETTINGS FOR NUMERICAL SCHEMES (1)

*Finally, we need to adjust the numerical shemes. As in the previous examples it is implemented in system/fvSchemes. For divergent items the **upwind** scheme is chosen, for the diffusion — the scheme of central differences **linear**.*

*An important difference is that the Euler time differentiation scheme (ddtSchemes) is implemented, though for the steady state we can choose the option **steadyState** — the time derivative is equal to 0*

Then, as before, we define the method to solve the SLE in the file system/fvSolution. There is no necessity in having a strict solution on each step, that's why the relative precision relTol can take values of order 0.01 — 0.001

```
p_rgh
{
    solver          PCG;
    preconditioner  DIC;
    tolerance       1e-8;
    relTol          0.01;
}
```

FREE CONVECTION: SETTINGS FOR NUMERICAL SCHEMES(2)

In conclusion, we'll set the output and integration parameters (system/controlDict)

```
application     buoyantPimpleFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         2000;

deltaT          2;

writeControl    timeStep;

writeInterval   100;
```

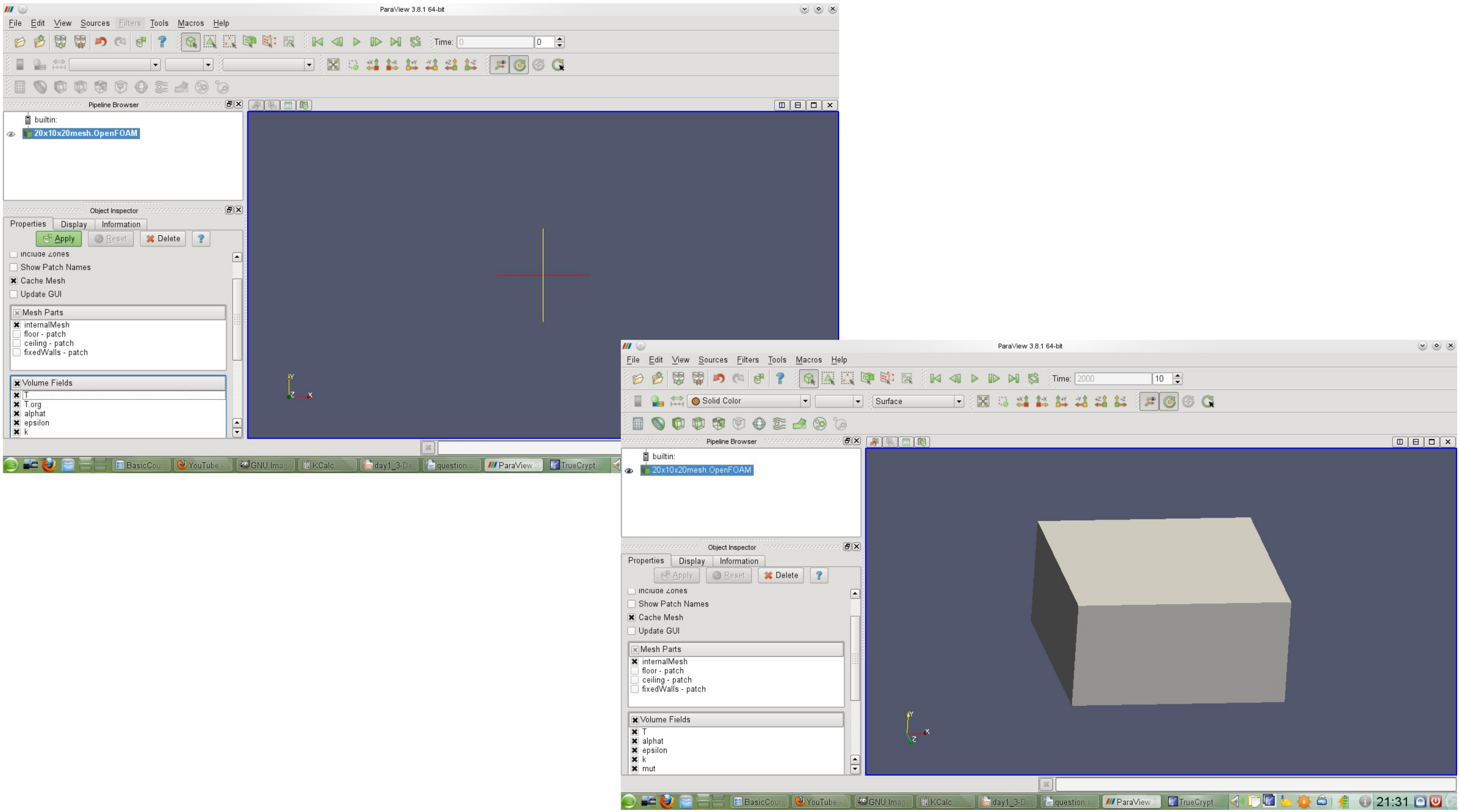
```
purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression
uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  no;

maxCo           0.5;
```

FREE CONVECTION: RUN & MONITOR

Let's run the programme:

rm -rf run.log; buoyantPimpleFoam | tee -a run.log

FREE CONVECTION: VISUALIZATION (1)



FREE CONVECTION: VISUALIZATION (2)