Laser Convection Boundary Condition User Guide

- by Priyanshu Asthana

⭐ This work is based on Tobias Holzmann's boundary condition, which works until OpenFOAM-6. The current code is developed for newer versions of OpenFOAM with new features by Priyanshu Asthana as a part of his Summer Research Internship at Materials Modelling Group, Indian Institute of Science, Bangalore, India.

⚠️ Recommended way of using OpenFOAM is with OpenFOAM-in-Box which works on any Linux OS.This code works with OpenFOAM-in-Box-18.10 and OpenFOAM-in-Box-20.09.
The code has been successfully tested on following versions of OpenFOAM on Ubuntu 18.04 and CentOS 6 (Cluster):
1. OpenFOAM-in-Box-18.10v1
2. OpenFOAM-in-Box-20.09v2
3. OpenFOAM v6.x, v7.x, v8.x, v9.x

The code does not work with any newer or older versions of OpenFOAM and OpenFOAM-in-Box than mentioned above.

Table of Contents

1. Introduction

This boundary condition provides a laser source boundary based on the Gaussian distribution function with additional convective heat transfer. The boundary also works as a convective boundary condition, if no source is added. This boundary condition uses OpenFOAM for calculating the temperature distribution in the material. It has been optimized to work with newer versions of OpenFOAM (v6.x, v7.x, v8.x ,v9.x) and OpenFOAM-in-Box(v18.10, v20.09).

Salient features of the boundary condition includes:

- Creating multiple Gaussian laser profiles with different settings within one patch
- Heat transfer coefficients can be added for convection
- Two different laser motion modes - linear and circular
    - Circular motion means to move the Gaussian spot around a center point with a given angular velocity
    - Linear motion means to move the Gaussian spot linear within a set of given points with a defined linear velocity
- Using temperature depended thermal conductivity field
- Adjusting and reducing the laser power based on temperature or stresses or some user defined function

⚠️ The laser source can only be applied **perpendicular** to the x-y plane, as no coordinate transformation has been implemented.

---

2. Download and Install OpenFOAM-in-Box

2.1 Downloading

Download OpenFOAM-in-Box using the following link:

Openfoam In Box - CFD SUPPORT

OpenFOAM for Linux independent on system libraries

https://www.cfdsupport.com/openfoam-in-box.html

Note: On the official website only the latest version is available (v20.09) at the time of writing this document. To download older versions (v18.10) please visit: https://drive.google.com/file/d/1gzK8ipaC-be_dvljckus6uFizazkO1eU/view

2.2 Installing

1. Copy the installation package in your installation path. You can do that either manually or use following Linux commands:

```
mkdir -p ~/OpenFOAM/OpenFOAM-in-Box
# Go to location where openfoam-in-box installation file is located and then execute the following:
cp OpenFOAM-in-Box-XX.XXvX-rXXX-installer.sh ~/OpenFOAM/OpenFOAM-in-Box/
```

2. Install the package using:

```
cd ~/OpenFOAM/OpenFOAM-in-Box/
bash OpenFOAM-in-Box-XX.XXvX-rXXX-installer.sh -install
```

When opening a new terminal window the OpenFOAM environment (system) variables need to be loaded:

```
source ~/OpenFOAM/OpenFOAM-in-Box/OpenFOAM-in-Box-XX.XXvX/OpenFOAM-dev/etc/bashrc
```

For easier access, you can create an alias in your system's `.bashrc` file by using the following command:

```
echo "alias ofboxXX='source ~/OpenFOAM/OpenFOAM-in-Box/OpenFOAM-in-Box-XX.XXvX/OpenFOAM-dev/etc/bashrc" >> ~/.bashrc
source ~/.bashrc
```

This will create and alias named `ofboxXX` in `.bashrc` file. Next time when you want to source OpenFOAM environment variables just run `ofboxXX` in a new terminal.

⚠️ **XX** here refers to the OpenFOAM-in-Box version. Please check your corresponding version, eg. ofbox18 for OpenFOAM-in-Box-18.

Alias is created instead of directly adding the source line in `.bashrc` because we might have different versions of OpenFOAM on our system and each might have its own set of home directory, environment variables and commands. This functionality of sourcing every time helps us choose the appropriate OpenFOAM version on the go.

---

3. Using Laser Convection Boundary Condition

💡 **You should be familiar with the basic operation of OpenFOAM which includes being able to edit OpenFOAM cases, compile solvers, run cases and view results in Paraview to use the boundary condition. We have provided explanation wherever possible.**

3.1 Compiling the Boundary Condition

Feel free to compile it where ever you want, but normally its nice to have a fixed folder for *user compiled modules.*

1. Open a new terminal window and run:

```
ofboxXX
```

2. Create a new folder and switch to this new folder:

```
mkdir -p $FOAM_RUN/../OpenFOAM_extensions
cd $FOAM_RUN/../OpenFOAM_extensions
```

3. Clone the repository to the new folder: (If you don't have GIT installed on your computer, you can install git from here.)

```
git clone <link to github repo>
```

1. Go to `laserConvectionBC` directory and run:

```
cd laserconvectionbc
git pull
wmake libso
```

Using the above commands, the boundary condition should compile without any error for the first time or else it would mean that the installation is not done correctly.

3.2 Preparing the Test Case

1. Go to your `OpenFoam case` directory which contains files like `0, constant, system` and few other files.
2. If you compiled the library (as explained above), add the `libs` command to your `system/controlDict`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
libs ( "liblaserConvectionBC.so" );    //include the compiled boundary condition here
application     laplacianFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         6e-07;
deltaT          5e-09;
writeControl    runTime;
writeInterval   2e-08;
purgeWrite      0;
writeFormat     ascii;
```

```
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
// ************************************************************************* //
```

3. In `0/T` file, enter the boundary condition in the required patch. A description how to set-up the boundary condition is given in the later sections.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 1 0 0 0];
internalField   uniform 300;
boundaryField
{
    floor
    {
        type            zeroGradient;
    }

    ceiling
    {
         type            laserConvection;
         value           uniform 300;
         // for detailed description, please check the Examples section.
    }

    fixedWalls
    {
        type            zeroGradient;
    }

}

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

4. Run your test case as per the solver manual or your solver. For demonstration purpose**,** use the `test_case` added along with boundary condition.

3.3 Important Points to Remember

1. The boundary condition should only be mentioned in the specified format and with the specified units.
2. Whenever you open a new terminal, source the OpenFOAM environment variables from the version that we want to use. For OpenFOAM-in-Box use:

```
ofboxXX
```

3. To unload the environment variable, use:

```
wmUNSET
```

Thus, you don't have to close the window to start another version of OpenFOAM.

4. Examples

4.1 Boundary Condition set-up for convection without sources

```
{
        type            laserConvection;           // Specifying the Laser Bounday Condition
        value           uniform 300;               // Set the initial temperature of the system

        HTCheating      23;                        // Heat Transfer Coefficient used until the time the laser is on [W/m^2/K]
        HTCquenching    15000;                     // Heat Transfer Coefficient used after the laser is turned off [W/m^2/K]
/* Note:
> In HTCheating, use the heat transfer coefficient of air/gas surrounding the object.
> If the object is quenched after the laser is turned off then use the heat transfer coeffient corresponding to that
  of the quenching medium in in HTCquenching
> If the object is not quenched after the laser is turned off then keep the value of HTCquenching same as HTCheating.
*/
        TfH             300;                       // Temperature of surrounding fluid used until the time the laser is on [K]
        TfQ             290;                       // Temperature of fluid after the laser is turned off [K]

        heatingTime     0.013;                     // Time for which laser source is active [s]

        powerReduceName   none;                    // Name of field (scalar) that reduce the LASER power
        kName             k;                       // In case of temperature depended thermal conductivity field, a function can be speci

        kValue          250;                       // Use this line only if you have constant thermal conductivity [W/mK]
    }
```

4.2 Boundary Condition set-up for convection with LASER sources

5. Programming Description

> 💡 **For the following guide on boundary condition, it is assumed that you are well conversant with a good knowledge of C++ as the source code is majorly developed in C++. We have provided self explanatory description of the code wherever possible.**

In the boundary condition, we have used `Foam::laserConvectionFvPatchField` class which is derived from the `Foam::mixedFvPatchField` class. The general schematic of code is given below:

Firstly all the variables are declared to be used throughout the code.

```
template<class Type>
Foam::laserConvectionFvPatchField<Type>::laserConvectionFvPatchField
(
    const laserConvectionFvPatchField<Type>& ptf,
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF,
    const fvPatchFieldMapper& mapper
)
:
    mixedFvPatchField<Type>(ptf, p, iF, mapper),
    nSources_(ptf.nSources_),
    reducedCoeff_(ptf.reducedCoeff_),
    reducedCoeffName_(ptf.reducedCoeffName_),
    coeff_(ptf.coeff_),
    kField_(ptf.kField_),
    kFieldName_(ptf.kFieldName_),
    kValue_(ptf.kValue_),
    sourceCenters_(ptf.sourceCenters_),
    normals_(ptf.normals_),
    sigmaX_(ptf.sigmaX_),
    sigmaY_(ptf.sigmaY_),
    power_(ptf.power_),
    correlation_(ptf.correlation_),
    HTCHeating_(ptf.HTCHeating_),
    HTCCooling_(ptf.HTCCooling_),
    TfluidHeating_(ptf.TfluidHeating_),
```

```
    TfluidQuench_(ptf.TfluidQuench_),
    heatingTime_(ptf.heatingTime_),
    heatingTimeSource_(ptf.heatingTimeSource_),
    motion_(ptf.motion_),
    motionMode_(ptf.motionMode_),
    startMotion_(ptf.startMotion_),
    centerUpdated_(ptf.centerUpdated_),
    actualTime_(ptf.actualTime_),
    motionCenters_(ptf.motionCenters_),
    startAngle_(ptf.startAngle_),
    omega_(ptf.omega_),
    nCycles_(ptf.nCycles_),
    lMPoints_(ptf.lMPoints_),
    timeAcc_(ptf.timeAcc_),
    endPoint_(ptf.endPoint_),
    gaussDistribution_(ptf.gaussDistribution_),
    heatFluxDistribution_(ptf.heatFluxDistribution_),
    valueFraction_(ptf.valueFraction_),
    refValue_(ptf.refValue_),
    debug_(ptf.debug_)
{
    if (debug_)
    {
        Info<< "Constructor 2\n" << endl;
    }
}
```

Once declared, all variables are read from the `0/T` file and initialized with initial values.

```
template<class Type>
Foam::laserConvectionFvPatchField<Type>::laserConvectionFvPatchField
(
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF,
    const dictionary& dict
)
:
    mixedFvPatchField<Type>(p, iF),
    nSources_(0),
    reducedCoeff_(false),
    reducedCoeffName_("none"),
    coeff_(scalar(1)),
    kField_(false),
    kFieldName_("none"),
    kValue_(scalar(0)),
    sourceCenters_(0,point(0,0,0)),
    normals_(0,vector(0,0,0)),
    sigmaX_(0,scalar(0)),
    sigmaY_(0,scalar(0)),
    power_(0,scalar(0)),
    correlation_(0,scalar(0)),
    HTCHeating_(readScalar(dict.lookup("HTCheating"))),
    HTCCooling_(readScalar(dict.lookup("HTCquenching"))),
    TfluidHeating_(readScalar(dict.lookup("TfH"))),
    TfluidQuench_(readScalar(dict.lookup("TfQ"))),
    heatingTime_(readScalar(dict.lookup("heatingTime"))),
    heatingTimeSource_(0,scalar(0)),
    motion_(0,false),
    motionMode_(0,word("None")),
    startMotion_(0,scalar(0)),
    centerUpdated_(0,false),
    actualTime_(scalar(0.)),
    motionCenters_(0,point(0,0,0)),
    startAngle_(0,scalar(0)),
    omega_(0,scalar(0)),
    nCycles_(0,scalar(0)),
    lMPoints_(0,pointField(0,point(0,0,0))),
    lMSpeed_(0,scalar(0)),
    timeAcc_(0,scalarList(0,scalar(0))),
    endPoint_(0,false),
    gaussDistribution_(0,scalarField(0,scalar(0))),
    heatFluxDistribution_(p.size(), scalar(0)),
    valueFraction_(p.size(), scalar(1)),
    refValue_(TfluidHeating_),
    debug_(false)
```

The `patchname` contains the name of the patch where boundary condition is required.

```
const word patchName = this->patch().name();
```

All laser sources are read and stored in `contentOfTable` . `sourceDictFound` is initialized as false which turns true if laser source is specified in boundary condition. Once the list is made, all the items are iterated through to register the laser sources.

```
    // Creating a dictionary "contentOfTable" with list of content
    const wordList& contentOfTable = dict.toc();
    bool sourceDictFound = false;
    forAll(contentOfTable, c)
    {
        .
        .
    }
```

For each patch we can create `n` arbitrary LASER sources with `subdicts`. If a new sub-dictionary is found, the code recognizes it as a new laser source.

```
if(dict.isDict(contentOfTable[c])
{
  //- New source found
  nSources_++;
  .
  .
}
```

If `active` is set to `TRUE` and we set the LASER `source time` to this value; otherwise the laser source will be active till the `heating time` is passed.

```
if (found)
            {
                const bool active = sourceDict.lookup("active");

                if (active)
                {
                    heatingTimeSource_.append
                    (
                        readScalar(sourceDict.lookup("active"))
                    );
                }
            }
            else
            {
                heatingTimeSource_.append(heatingTime_);
            }
```

If we find a `subdictionary` for the motion, we read motion diction for source. This sub dictionary contains all the information regarding the movement of laser.

```
            if (motion_[nSources_-1])
            {
                Info<< "\n ++ Read motion dict for source "
                    << sourceDictName_[nSources_-1] << endl;
                .
                .
            }
```

Once all the input parameters are read and calculations are done for a particular `timestep`, we run the following functions to update the system for next `timestep`:

Function	Description

Function	Description
`updateSpotCenter()`	Changes the spot center while the laser source is moving
`updateValueFraction()`	Depending on the heating or quenching method, this function calculates the value fraction
`updateGaussDistribution()`	Recalculates Gauss Distribution if motion is active
`updateHeatFluxDistribution()`	Calculates the heat flux distribution while the laser source is moving
`updateRefTemperature()`	Reference Temperature depending on heating or quenching method

Finally we write all the required parameters to the next `timestep`.

```
template<class Type>
void Foam::laserConvectionFvPatchField<Type>::write(Ostream& os) const
{
    fvPatchField<Type>::write(os);

    os.writeKeyword("HTCheating");
    os<< HTCHeating_ << ";\n";

    os.writeKeyword("HTCquenching");
    os<< HTCCooling_ << ";\n";

    os.writeKeyword("TfH");
    os<< TfluidHeating_ << ";\n";

    os.writeKeyword("TfQ");
    os<< TfluidQuench_ << ";\n";

    os.writeKeyword("heatingTime");
    os<< heatingTime_ << ";\n";

    //- Writing source dicts to next timestep
    forAll(sourceDicts_, dict)
    {
        os<< "\n";
        os.writeKeyword(sourceDictName_[dict]);
        os<< sourceDicts_[dict];
    }

    os.writeKeyword("powerReduceName");
    if (reducedCoeff_)
    {
        os<< reducedCoeffName_ << ";\n";
    }
    else
    {
        os<< "none;\n";
    }

    if (kField_)
    {
        os.writeKeyword("kName");
        os<< kFieldName_ << ";\n";
    }
    else
    {
        os.writeKeyword("kName none;\n");
        os.writeKeyword("kValue");
        os<< kValue_ << ";\n";
    }

    //- Writing face values
    this->writeEntry("value", os); // OpenFOAM older versions (v=6.x)
//    writeEntry(os, "value", *this);    // OpenFOAM older versions (v>=7.x)

}
```

For in-depth understanding of mathematical concepts used in the code, please refer to the documentation by Tobias Holzmann.