Dual Degree Project

Generated by Doxygen 1.8.11

Contents

Chapter 1

My Personal Index Page

1.1 Introduction

This is the C++ code to solve the high speed fluid flow. Currently, Euler flow is being solved but this code has been designed in moulder way so to solve the viscus flow additional "viscus flux" class can be added very easily. This code has been written to fulfill the requirement of the Dual Degree Project(DDP). All the major input has been given from the input file. For post processing either MATLAB or Python could be used.

1.2 Installation & Use

To use the solver. Follow these simple steps.

- Download form here : `https://github.com/singh-kuldeep/DDP2` or `click here`
- Go to the DDP2 folder and compile and run the file MainSolver.cpp (ex. g++ MainSolver.cpp && ./a.out)
- Nozzle has been set up as a default geometry but it can be changed from "inputfile" by uncommenting the appropriate geometry

1.3 Input file

All the major inputs are taken through the input file. For example

1. CFL (Courant–Friedrichs–Lewy)

   ```
   CFL = 0.5
   ```

2. Total number of iterations (TotalIteration)

   ```
   TotalIteration = 1500000
   ```

3. There are two options available for scheme

   ```
   Scheme = Roe
   Scheme = AUSM
   ```

4. There are two options available for gamma

```
gamma = Constant
gamma = Gamma(T)
```

5. If SpecificHeatRatio is constant, then define the value

```
SpecificHeatRatio = 1.4
```

6. Boundary condition. Currently, There are 5 Options for BC.

   (a) SuperSonicInlet (T0,p0 and M needs to be specified)

   (b) SuperSonicExit

   (c) SubSonicInlet (T0, p0 needs to be specified)

   (d) SubSonicExit (Exit pressure needs to be specified)

   (e) Wall

7. One has to specify boundary condition (only the above mentioned) at all the faces

```
BoundaryConditionati0 = SubSonicInlet
BoundaryConditionatj0 = Wall
BoundaryConditionatk0 = Wall
BoundaryConditionatiNi = SuperSonicExit
BoundaryConditionatjNj = Wall
BoundaryConditionatkNk = Wall
```

8. Total Quantities

```
InletTotalTemperature = 1800
InletTotalPressure = 5.2909e+07
InletMach = 3.0(Only when supersonic inlet)
```

9. Initial Condition options

```
InitialCondition = ZeroVelocity
InitialCondition = FreeStreamParameterAndZeroVelocity
InitialCondition = FreeStreamParameterEverywhare
InitialCondition = NozzleBasedOn1DCalculation
(Uncomment only in case of nozzle)
InitialCondition = StartFromPreviousSolution
```

10. Geometry options, Currently there are three different geometry options are available

```
GeometryOption = StraightDuct
GeometryOption = BumpInsidetheStraightSuct
GeometryOption = IdelNozzleDesignedUsingMOC
```

11. Time steeping

```
TimeSteping = Local
TimeSteping = Global
```

1.4 Brief about the solver

- Written in C++
- Structured grid
- Roe and AUSM scheme
- Euler flow with both variable and constant gamma
- 3D Cartesian (x,y,z)
- Detailed theory can be found in the report here.

1.5 Output files.

Here are the list of files which will come as the output of the solver for post processing of the results. All these file are automatically stored in the **"./Results/outputfiles"** folder.

- CellCenter_ij.csv : Cell centers of XY plane
- ConservedQuantity.csv : All the conserved quantities of the current iteration
- ds.csv : Minimum distance for "delta t" calculation
- Residual_Nozzle.csv : All the residuals(Mass, Momentum, Energy)

1.6 Post processing (Results or Plots)

The same older contains the MATLAB script **"PostProcessing.m"** and **"PostProcessing.py"**. Any one of the file(Script/Code) can be used for the post processing. Once the simulation has started and the output files are generated, one can simply run these scripts and can see the plots which are listed below.

- Density Residual
- X Momentum Residual
- Y Momentum Residual
- Z Momentum Residual
- Energy Residual
- Mach Number
- Density
- Velocity
- Temperature
- Pressure
- Total pressure
- Total temperature
- Geometry 2D cross section

Chapter 2

Bug List

Member grid (vector< vector< vector< vector< double > > > > &Coordinate, vector< vector< vector< vector< double > > > > &iFaceAreaVector, vector< vector< vector< vector< double > > > > &jFaceAreaVector, vector< vector< vector< vector< double > > > > &kFaceAreaVector, vector< vector< vector< double > > > &CellVolume, vector< vector< vector< double > > > &ds, int &Ni, int &Nj, int &Nk, string GeometryOption)

Randomly extra zeros at the end so to remove them pop is used

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 eulerfluxAUSM Class Reference

This class calculates the Euler flux vectors(Ee,Fe,Ge) at the interface for the AUSM scheme.

`#include <ConvectiveFluxAUSM.h>`

Public Member Functions

- eulerfluxAUSM (vector< double > ConservedVariable, vector< double > AreaVector, string gamma, double SpecificHeatRatio)

Public Attributes

- double Flux [5]
- double MachPlus
- double MachMinus
- double PressurePlus
- double PressureMinus
- double Mach

5.1.1 Detailed Description

This class calculates the Euler flux vectors(Ee,Fe,Ge) at the interface for the AUSM scheme.

Date

18-May-2017

5.1.2 Constructor & Destructor Documentation

5.1.2.1 eulerfluxAUSM::eulerfluxAUSM (vector< double > *ConservedVariable,* vector< double > *AreaVector,* string *gamma,* double *SpecificHeatRatio*) `[inline]`

A constructor to calculate the convective flux and initiates the parameter which are required in the flux calculation.

Parameters

ConservedVariable	All the conserved variable in the cell
AreaVector	Face area vector
gamma	String tells whether to consider specific heat ratio is constant or varying with temperature
SpecificHeatRatio	Specific heat ratio in case of constant gamma

Parameters

AreaVectorNormal	Interface unit area vector
AreaVectorMagnitude	Magnitude of area vector
VelocityNormal	Magnitude of velocity vector normal to the interface, or **contravarient velocity**

5.1.3 Member Data Documentation

5.1.3.1 double eulerfluxAUSM::Flux[5]

Parameters

Flux	Convective flux vector in the cell

5.1.3.2 double eulerfluxAUSM::Mach

Parameters

Mach	Mach number at the cell

5.1.3.3 double eulerfluxAUSM::MachMinus

Parameters

MachMinus	MachMinus needed in AUSM scheme, at a the cell

5.1.3.4 double eulerfluxAUSM::MachPlus

Parameters

MachPlus	MachPlus needed in AUSM scheme, at a the cell

5.1.3.5 double eulerfluxAUSM::PressureMinus

Parameters

PressureMinus	PressureMinus needed in AUSM scheme, at a the cell

5.1.3.6 double eulerfluxAUSM::PressurePlus

Parameters

PressurePlus	PressurePlus needed in AUSM scheme, at a the cell

The documentation for this class was generated from the following file:

- ConvectiveFluxAUSM.h

5.2 netfluxAUSM Class Reference

Calculates the net AUSM flux at the interface.

```
#include <NetFluxAUSM.h>
```

Public Member Functions

- netfluxAUSM (vector< double > LeftConservedVariable, vector< double > RightConservedVariable, vector< double > AreaVector, string gamma, double SpecificHeatRatio)

Public Attributes

- double NetFlux [5]

5.2.1 Detailed Description

Calculates the net AUSM flux at the interface.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 netfluxAUSM::netfluxAUSM (vector< double > *LeftConservedVariable*, vector< double > *RightConservedVariable*, vector< double > *AreaVector*, string *gamma*, double *SpecificHeatRatio*) [inline]

A constructor to calculate the net AUSM flux and to initiates the parameter which are required in the flux calculation.

Parameters

LeftConservedVariable	Conserved variable of the left cell
RightConservedVariable	Conserved variable of the right cell
AreaVector	Face area vector
gamma	String tells whether to consider specific heat ratio is constant or varying with temperature
SpecificHeatRatio	Specific heat ratio in case of constant gamma

5.2.3 Member Data Documentation

5.2.3.1 double netfluxAUSM::NetFlux[5]

Parameters

NeTFlux[5]	Net AUSM flux vector at the cell interface

The documentation for this class was generated from the following file:

- NetFluxAUSM.h

Chapter 6

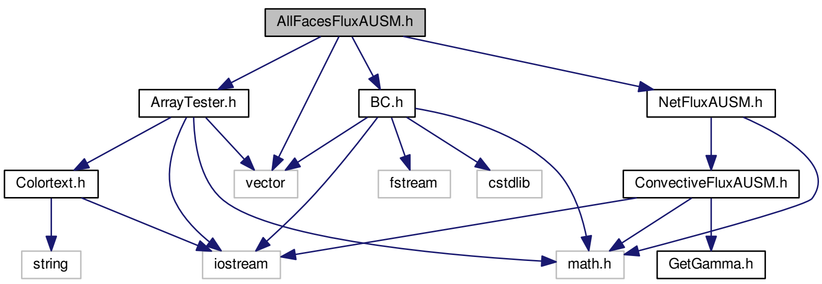
File Documentation

6.1 AllFacesFluxAUSM.h File Reference

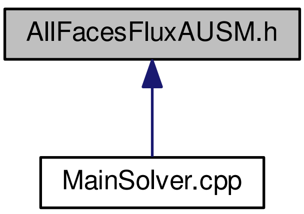
Calculates the flux at all the cell faces in the domain using the AUSM scheme.

```
#include "vector"
#include "ArrayTester.h"
#include "NetFluxAUSM.h"
#include "BC.h"
```

Include dependency graph for AllFacesFluxAUSM.h:



This graph shows which files directly or indirectly include this file:



Functions

- void flux (vector< vector< vector< vector< double > > > > &iFacesFlux, vector< vector< vector< vector< double > > > > &jFacesFlux, vector< vector< vector< vector< double > > > > &kFacesFlux, vector< vector< vector< vector< double > > > > iFaceAreaVector, vector< vector< vector< vector< double > > > > jFaceAreaVector, vector< vector< vector< vector< double > > > > kFaceAreaVector, vector< vector< vector< vector< double > > > > ConservedVariables, int Ni, int Nj, int Nk, string gamma, double SpecificHeatRatio)

  function flux() calculates the flux at every face in the domain

6.1.1 Detailed Description

Calculates the flux at all the cell faces in the domain using the AUSM scheme.

Date

18-May-2017

6.1.2 Function Documentation

6.1.2.1 void flux (vector< vector< vector< vector< double > > > > & *iFacesFlux*, vector< vector< vector< vector< double > > > > & *jFacesFlux*, vector< vector< vector< vector< double > > > > & *kFacesFlux*, vector< vector< vector< vector< double > > > > *iFaceAreaVector*, vector< vector< vector< vector< double > > > > *jFaceAreaVector*, vector< vector< vector< vector< double > > > > *kFaceAreaVector*, vector< vector< vector< vector< double > > > > *ConservedVariables*, int *Ni*, int *Nj*, int *Nk*, string *gamma*, double *SpecificHeatRatio*)

function flux() calculates the flux at every face in the domain

Parameters

`in,out`	*&iFacesFlux*	Pointer to an array which stores the flux at "i" interfaces
`in,out`	*&jFacesFlux*	Pointer to an array which stores the flux at "j" interfaces
`in,out`	*&kFacesFlux*	Pointer to an array which stores the flux at "k" interfaces
`in`	*&iFacesFlux*	An array which has area vectors of the "i" interfaces
`in`	*&jFacesFlux*	An array which has area vectors of the "j" interfaces
`in`	*&kFacesFlux*	An array which has area vectors of the "k" interfaces

Parameters

LeftConservedVariables	Temporary vector which store the "left" cell conserved variables
RightConservedVariables	Temporary vector which store the "right" cell conserved variables
i0GhostConservedVariable	Ghost cell Conserved variables array at i = 0
j0GhostConservedVariable	Ghost cell Conserved variables array at j = 0
k0GhostConservedVariable	Ghost cell Conserved variables array at k = 0
iNiGhostConservedVariable	Ghost cell Conserved variables array at i = Ni
jNjGhostConservedVariable	Ghost cell Conserved variables array at j = Nj
kNkGhostConservedVariable	Ghost cell Conserved variables array at k = Nk

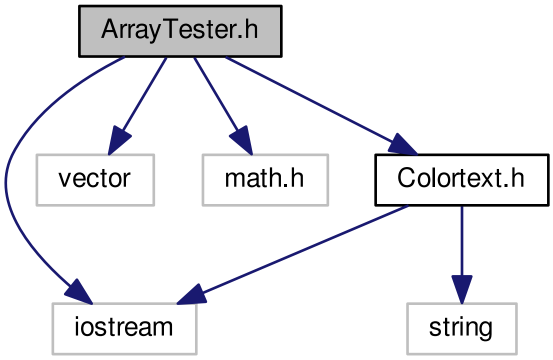
Before every time step we need to have proper value in all the ghost cells So, function BC() takes care of this

6.2 ArrayTester.h File Reference

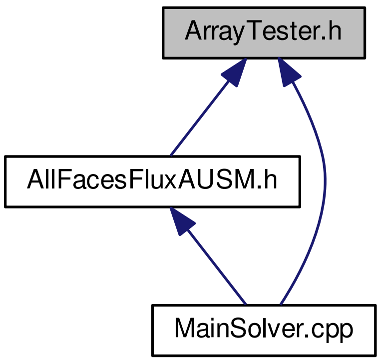
Contains the function which checks the NaN/-NaN in array.

```
#include "iostream"
#include <vector>
#include "math.h"
#include "Colortext.h"
```

Include dependency graph for ArrayTester.h:



This graph shows which files directly or indirectly include this file:



Functions

- int test3DArray (string arrayname, vector< vector< vector< double > > > a, int Ni, int Nj, int Nk)

    Checks the NaN in a 3D array.
- int test4DArray (string arrayname, vector< vector< vector< vector< double > > > > a, int Ni, int Nj, int Nk, int Nl)

    Checks the NaN in a 3D array.
- int testConservedVariables (string arrayname, vector< vector< vector< vector< double > > > > ConservedVariables, int Ni, int Nj, int Nk, int Nl)

    Checks whether density, pressure, energy are becoming NaN, at any location.

6.2.1 Detailed Description

Contains the function which checks the NaN/-NaN in array.

6.2.2 Function Documentation

6.2.2.1 int test3DArray (string *arrayname*, vector< vector< vector< double > > > *a*, int *Ni*, int *Nj*, int *Nk*)

Checks the NaN in a 3D array.

Parameters

in	*arrayname*	Name of the array.
in	*a*	Pointer to the array
in	*Ni*	Array's first dimension
in	*Nj*	Array's second dimension
in	*Nk*	Array's third dimension

Returns

"0" if NaN and "1" if Not NaN

6.2.2.2 int test4DArray (string *arrayname*, vector< vector< vector< vector< double > > > > *a*, int *Ni*, int *Nj*, int *Nk*, int *Nl*)

Checks the NaN in a 3D array.

Parameters

in	*arrayname*	Name of the array.
in	*a*	Pointer to the array
in	*Ni*	Array's first dimension
in	*Nj*	Array's second dimension
in	*Nk*	Array's third dimension
in	*Nl*	Array's fourth dimension

Returns

"0" if NaN and "1" if Not NaN

6.2.2.3 int testConservedVariables (string *arrayname*, vector< vector< vector< vector< double > > > > *ConservedVariables*, int *Ni*, int *Nj*, int *Nk*, int *Nl*)

Checks whether density, pressure, energy are becoming NaN, at any location.

Parameters

in	*arrayname*	Name of the array.
in	*ConservedVariables*	Pointer to the array
in	*Ni*	Array's first dimension
in	*Nj*	Array's second dimension
in	*Nk*	Array's third dimension
in	*Nl*	Array's fourth dimension

Returns

"0" if NaN and "1" if Not NaN

Parameters

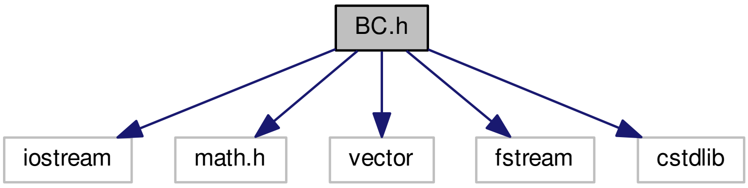
Density	Density at any location
XVelocity	X direction velocity at any location
YVelocity	Y direction velocity at any location
ZVelocity	Z direction velocity at any location
pressure	Pressure at any location
energy	Total energy at any location

6.3 BC.h File Reference

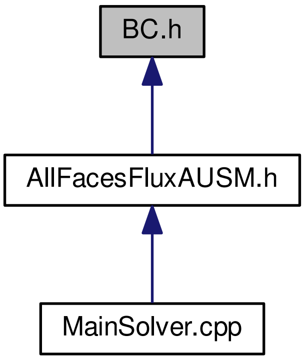
Implements the all three boundary conditions.

```
#include "iostream"
#include "math.h"
#include <vector>
#include <fstream>
#include <cstdlib>
```

Include dependency graph for BC.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **RealGasConstant** 287.17

Functions

- void getNormal (vector< double > &UnitNormal, vector< double > areaVector)

    Give the unit area vector, using the area vector of the face.
- double getMachfromPressureRatio (double Pressure, double TotalPressure, double SpecificHeatRatio)

    Calculates the Mach number using the total and static pressure.
- void WallBC (vector< double > &GhostCellConservedVariables, vector< double > LiveCellConservedVariables, vector< double > AreaVectors, double SpecificHeatRatio)

    This function implements the wall boundary condition.
- void SubSonicInletBC (vector< double > &GhostCellConservedVariables, vector< double > LiveCellConservedVariables, double InletTotalPressure, double InletTotalTemperature, double SpecificHeatRatio)

    Implements the subsonic inlet boundary condition.

- void SubSonicExitBC (vector< double > &GhostCellConservedVariables, vector< double > LiveCellConservedVariables, double ExitPressure, double SpecificHeatRatio)

    Implements the subsonic exit boundary condition.

- void SuperSonicExitBC (vector< double > &GhostCellConservedVariables, vector< double > LiveCellConservedVariables)

    Implements the supersonic exit boundary condition, by simple extrapolation.

- void SuperSonicInletBC (vector< double > &GhostCellConservedVariables, double InletTotalPressure, double InletTotalTemperature, double InletMach, double SpecificHeatRatio)

    Implements the subsonic exit boundary condition.

- void BC (vector< vector< vector< vector< double > > > > ConservedVariables, vector< vector< vector< vector< double > > > > iFaceAreaVector, vector< vector< vector< vector< double > > > > jFaceAreaVector, vector< vector< vector< vector< double > > > > kFaceAreaVector, vector< vector< vector< vector< double > > > > &i0GhostConservedVariable, vector< vector< vector< vector< double > > > > &j0GhostConservedVariable, vector< vector< vector< vector< double > > > > &k0GhostConservedVariable, vector< vector< vector< vector< double > > > > &iNiGhostConservedVariable, vector< vector< vector< vector< double > > > > &jNjGhostConservedVariable, vector< vector< vector< vector< double > > > > &kNkGhostConservedVariable, int Ni, int Nj, int Nk, double SpecificHeatRatio)

    Function BC() implements the boundary condition. Here ghost cell are used to implement the boundary condition. In simple words this function calculates the conserved variables for all ghost cells.

6.3.1 Detailed Description

Implements the all three boundary conditions.

- Inlet
- Exit and
- Wall boundary

Date

18-May-2017

6.3.2 Function Documentation

6.3.2.1 void BC (vector< vector< vector< vector< double > > > > *ConservedVariables*, vector< vector< vector< vector< double > > > > *iFaceAreaVector*, vector< vector< vector< vector< double > > > > *jFaceAreaVector*, vector< vector< vector< vector< double > > > > *kFaceAreaVector*, vector< vector< vector< vector< double > > > > & *i0GhostConservedVariable*, vector< vector< vector< vector< double > > > > & *j0GhostConservedVariable*, vector< vector< vector< vector< double > > > > & *k0GhostConservedVariable*, vector< vector< vector< vector< double > > > > & *iNiGhostConservedVariable*, vector< vector< vector< vector< double > > > > & *jNjGhostConservedVariable*, vector< vector< vector< vector< double > > > > & *kNkGhostConservedVariable*, int *Ni*, int *Nj*, int *Nk*, double *SpecificHeatRatio*)

Function BC() implements the boundary condition. Here ghost cell are used to implement the boundary condition. In simple words this function calculates the conserved variables for all ghost cells.

Parameters

in	*ConservedVariables*	Pointer to the 4D vector where all the conserved variables of previous time step are stored.

Parameters

in	*iFaceAreaVector*	Pointer to the 4D vector which has the area vector of all faces which are in "i" direction.
in	*jFaceAreaVector*	Pointer to the 4D vector which has the area vector of all faces which are in "j" direction.
in	*kFaceAreaVector*	Pointer to the 4D vector which has the area vector of all faces which are in "k" direction.
in,out	*i0GhostConservedVariable*	Conserved variables in the ghost cells at i=0
in,out	*j0GhostConservedVariable*	Conserved variables in the ghost cells at j=0
in,out	*k0GhostConservedVariable*	Conserved variables in the ghost cells at k=0
in,out	*iNiGhostConservedVariable*	Conserved variables in the ghost cells at i=Ni
in,out	*jNjGhostConservedVariable*	Conserved variables in the ghost cells at j=Nj
in,out	*kNkGhostConservedVariable*	Conserved variables in the ghost cells at k=Nk
in	*Ni*	Number of cells in in "i" direction.
in	*Nj*	Number of cells in in "j" direction.
in	*Nk*	Number of cells in in "k" direction.
in	*SpecificHeatRatio*	Specific heat ratio

primitive variables at the inlet

Total quantities are given at the inlet

```
\warning Inlet Mach will be given/used,
```

only in case of supersonic inlet

6.3.2.2 double getMachfromPressureRatio (double *Pressure*, double *TotalPressure*, double *SpecificHeatRatio*)

Calculates the Mach number using the total and static pressure.

Parameters

in	*Pressure*	Static pressure
in	*TotalPressure*	Total pressure Changes the input vector into the unit normal vector.

Returns

Mach number

6.3.2.3 void getNormal (vector< double > & *UnitNormal*, vector< double > *areaVector*)

Give the unit area vector, using the area vector of the face.

Parameters

in	*&UnintNormal*	Pinter of the normal vector
in	*areaVector*	Area vector of the face Changes the input vector into the unit normal vector.

6.3.2.4 void SubSonicExitBC (vector< double > & *GhostCellConservedVariables*, vector< double > *LiveCellConservedVariables*, double *ExitPressure*, double *SpecificHeatRatio*)

Implements the subsonic exit boundary condition.

Parameters

`in`	*LiveCellConservedVariables*	Conserved variables array for the live cell.
`in,out`	*GhostCellConservedVariables*	Conserved variables array for the ghost cell.
`in`	*ExitPressure*	Pressure at exit.
`in`	*SpecificHeatRatio*	Specific heat ratio

6.3.2.5 void SubSonicInletBC (vector< double > & *GhostCellConservedVariables*, vector< double > *LiveCellConservedVariables*, double *InletTotalPressure*, double *InletTotalTemperature*, double *SpecificHeatRatio*)

Implements the subsonic inlet boundary condition.

Parameters

`in`	*LiveCellConservedVariables*	Conserved variables array for the live cell.
`in,out`	*GhostCellConservedVariables*	Conserved variables array for the ghost cell.
`in`	*InletTotalPressure*	Total pressure at inlet.
`in`	*InletTotalTemperature*	Total temperature at inlet.
`in`	*SpecificHeatRatio*	Specific heat ratio

6.3.2.6 void SuperSonicExitBC (vector< double > & *GhostCellConservedVariables*, vector< double > *LiveCellConservedVariables*)

Implements the supersonic exit boundary condition, by simple extrapolation.

Parameters

`in`	*LiveCellConservedVariables*	Conserved variables array for the live cell.
`in,out`	*GhostCellConservedVariables*	Conserved variables array for the ghost cell.

6.3.2.7 void SuperSonicInletBC (vector< double > & *GhostCellConservedVariables*, double *InletTotalPressure*, double *InletTotalTemperature*, double *InletMach*, double *SpecificHeatRatio*)

Implements the subsonic exit boundary condition.

Parameters

`in`	*LiveCellConservedVariables*	Conserved variables array for the live cell.
`in,out`	*GhostCellConservedVariables*	Conserved variables array for the ghost cell.
`in`	*InletMach*	Mach number at the inlet.
`in`	*SpecificHeatRatio*	Specific heat ratio

6.3.2.8 void WallBC (vector< double > & *GhostCellConservedVariables*, vector< double > *LiveCellConservedVariables*, vector< double > *AreaVectors*, double *SpecificHeatRatio*)

This function implements the wall boundary condition.

Parameters

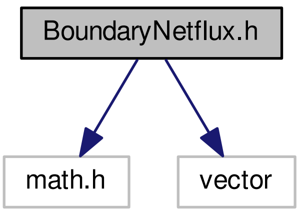
`in`	*AreaVectors*	Face area vectors.
`in`	*LiveCellConservedVariables*	Conserved variables array for the live cell.
`in,out`	*GhostCellConservedVariables*	Conserved variables array for the ghost cell.

6.4 BoundaryNetflux.h File Reference

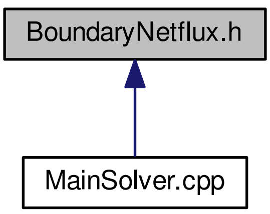
Contains the function boundaryNetflux() to obtain the all the fluxes at all the boundaries computational domain.

```
#include <math.h>
#include "vector"
```

Include dependency graph for BoundaryNetflux.h:



This graph shows which files directly or indirectly include this file:



Functions

- double Magnitude (vector< double > v)

    finds the magnitude of the vector
- void boundaryNetflux (vector< double > &iNetFlux, vector< double > &iNiNetFlux, vector< double > &j←NetFlux, vector< double > &jNjNetFlux, vector< double > &kNetFlux, vector< double > &kNkNetFlux, vector< vector< vector< vector< double > > > > &iFacesFlux, vector< vector< vector< vector< double > > > > &jFacesFlux, vector< vector< vector< vector< double > > > > &kFacesFlux, vector< vector< vector< vector< double > > > > iFaceAreaVector, vector< vector< vector< vector< double > > > > jFaceAreaVector, vector< vector< vector< vector< double > > > > kFaceAreaVector, int Ni, int Nj, int Nk)

    finds the net flux vector at the domain boundaries

6.4.1 Detailed Description

Contains the function boundaryNetflux() to obtain the all the fluxes at all the boundaries computational domain.

Date

18-May-2017

6.4.2 Function Documentation

6.4.2.1 void boundaryNetflux (vector< double > & *iNetFlux*, vector< double > & *iNiNetFlux*, vector< double > & *jNetFlux*, vector< double > & *jNjNetFlux*, vector< double > & *kNetFlux*, vector< double > & *kNkNetFlux*, vector< vector< vector< vector< double > > > > & *iFacesFlux*, vector< vector< vector< vector< double > > > > & *jFacesFlux*, vector< vector< vector< vector< double > > > > & *kFacesFlux*, vector< vector< vector< vector< double > > > > *iFaceAreaVector*, vector< vector< vector< vector< double > > > > *jFaceAreaVector*, vector< vector< vector< vector< double > > > > *kFaceAreaVector*, int *Ni*, int *Nj*, int *Nk*)

finds the net flux vector at the domain boundaries

Parameters

in	*iNetFlux*	Net flux at "i=0" boundary
in	*jNetFlux*	Net flux at "j=0" boundary
in	*kNetFlux*	Net flux at "k=0" boundary
in	*iNiNetFlux*	Net flux at "i=Ni" boundary
in	*jNjNetFlux*	Net flux at "j=Nj" boundary
in	*kNkNetFlux*	Net flux at "k=Nk" boundary
in	*&iFacesFlux*	Pointer to the "i" faces(whole domain) flux vector
in	*&jFacesFlux*	Pointer to the "j" faces(whole domain) flux vector
in	*&kFacesFlux*	Pointer to the "k" faces(whole domain) flux vector
in	*iFaceAreaVector*	Area vectors of the "i" faces(whole domain)
in	*jFaceAreaVector*	Area vectors of the "j" faces(whole domain)
in	*kFaceAreaVector*	Area vectors of the "k" faces(whole domain)
in	*Ni*	Number of cells in in "i" direction.
in	*Nj*	Number of cells in in "j" direction.
in	*Nk*	Number of cells in in "k" direction.

6.4.2.2 double Magnitude (vector< double > *v*)

finds the magnitude of the vector

Parameters

in	*v*	3D vector

Returns

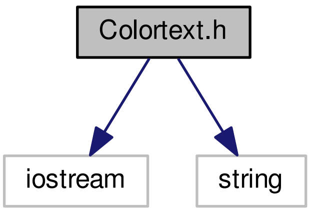
Magnitude of a 3D vector

6.5 Colortext.h File Reference

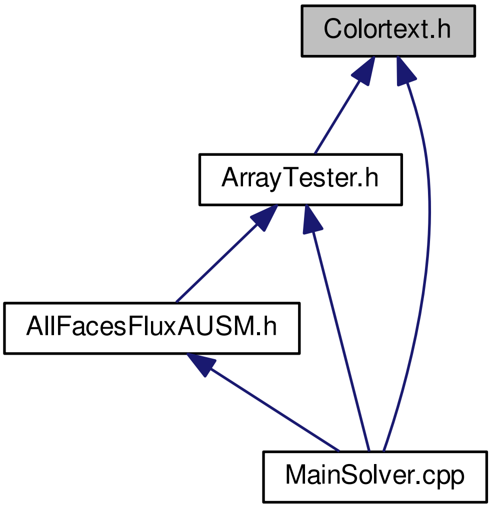
Contains the function which prints the colorful texts.

```
#include "iostream"
#include <string>
```

Include dependency graph for Colortext.h:



This graph shows which files directly or indirectly include this file:



Functions

- string fail ()

    Print the warning in red color if some test case fails.
- string pass ()

    Print the success in green color if some test case gets pass.
- string blue (string inputstring)

    Return the string after appending the syntax of blue color.
- string red (string inputstring)

    Return the string after appending the syntax of red color.
- string green (string inputstring)

    Return the string after appending the syntax of green color.

6.5.1 Detailed Description

Contains the function which prints the colorful texts.

Date

18-May-2017

6.5.2 Function Documentation

6.5.2.1 string blue (string *inputstring*)

Return the string after appending the syntax of blue color.

Parameters

[IN]	inputstring Input string

Returns

string

6.5.2.2 string fail ( )

Print the warning in red color if some test case fails.

Returns

string

6.5.2.3 string green (string *inputstring*)

Return the string after appending the syntax of green color.

Parameters

[IN]	inputstring Input string

Returns

string

6.5.2.4 string pass ( )

Print the success in green color if some test case gets pass.

Returns

string

6.5.2.5 string red (string *inputstring*)

Return the string after appending the syntax of red color.

Parameters

[IN]	inputstring Input string

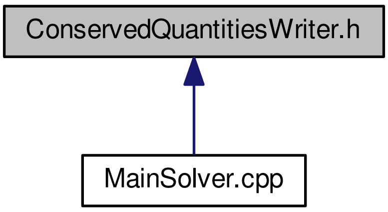
Returns

string

6.6 ConservedQuantitiesWriter.h File Reference

Writes all the conserved quantities of the domain in the file.

This graph shows which files directly or indirectly include this file:



Functions

- void WriteConserveredQuantities (vector< vector< vector< vector< double > > > > ConservedVariables, int Ni, int Nj, int Nk)

  Writes all the conserved quantities in the file "./Results/outputfiles/ConservedQuantity.csv".

6.6.1 Detailed Description

Writes all the conserved quantities of the domain in the file.

6.6.2 Function Documentation

6.6.2.1 void WriteConserveredQuantities (vector< vector< vector< vector< double > > > > *ConservedVariables,* int *Ni,* int *Nj,* int *Nk*)

Writes all the conserved quantities in the file "./Results/outputfiles/ConservedQuantity.csv".

Parameters

[IN]	ConservedVariables 4D Array where all the conserved quantities are stored
[IN]	Ni Number of cells in in "i" direction.
[IN]	Nj Number of cells in in "j" direction.
[IN]	Nk Number of cells in in "k" direction.

Warning

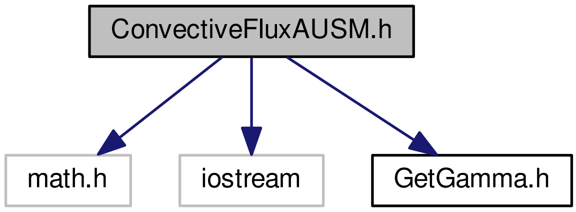
Before uncommenting this, uncomment the code in the Grid.h file which puts the extra points on the boundaries, otherwise there will be a conflict in the post processing

6.7 ConvectiveFluxAUSM.h File Reference

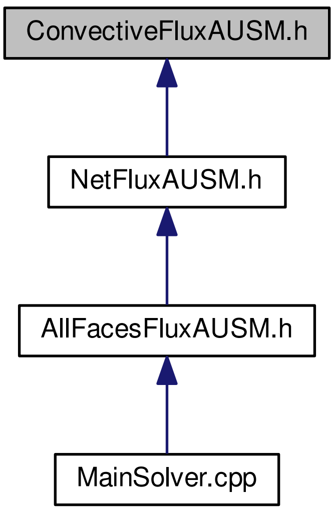
Calculates the convective flux of the AUSM scheme.

```
#include "math.h"
#include "iostream"
#include "GetGamma.h"
```

Include dependency graph for ConvectiveFluxAUSM.h:



This graph shows which files directly or indirectly include this file:



Classes

- class eulerfluxAUSM

    This class calculates the Euler flux vectors(Ee,Fe,Ge) at the interface for the AUSM scheme.

6.7.1 Detailed Description

Calculates the convective flux of the AUSM scheme.

Date

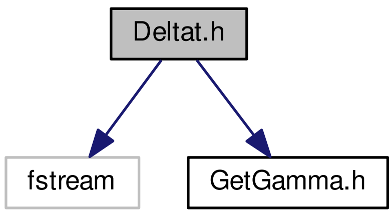
18-May-2017

6.8 Deltat.h File Reference

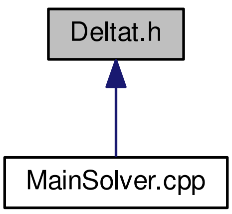
Contains the functions getLocalDeltaT() and getGlobalDeltaT(), which calculates the Local and global time step values respectively.

```
#include <fstream>
#include "GetGamma.h"
```

Include dependency graph for Deltat.h:



This graph shows which files directly or indirectly include this file:



Functions

- double getLocalDeltaT (vector< double > ConservedVariables, double MinimumDistance, double CFL, string gamma, double SpecificHeatRatio)

  Calculates the local time step value for a given cell.
- double getGlobalDeltaT (vector< vector< vector< vector< double > > > > ConservedVariables, vector< vector< vector< double > > > MinimumDistance, double CFL, int Ni, int Nj, int Nk, string gamma, double SpecificHeatRatio)

  Calculates the global time step value at every time iteration.

6.8.1 Detailed Description

Contains the functions getLocalDeltaT() and getGlobalDeltaT(), which calculates the Local and global time step values respectively.

Date

18-May-2017

6.8.2 Function Documentation

6.8.2.1 double getGlobalDeltaT (vector< vector< vector< vector< double > > > > *ConservedVariables,* vector< vector< vector< double > > > *MinimumDistance,* double *CFL,* int *Ni,* int *Nj,* int *Nk,* string *gamma,* double *SpecificHeatRatio*)

Calculates the global time step value at every time iteration.

Parameters

[IN]	ConservedVariables Conserved variables at cell center of a particular cell
[IN]	MinimumDistance Minimum distance of a given cell
[IN]	CFL
[IN]	Ni Number of cells in in "i" direction.
[IN]	Nj Number of cells in in "j" direction.
[IN]	Nk Number of cells in in "k" direction.
[IN]	gamma String which tells that specific heat ratio is constant or function of temperature
SpecificHeatRatio	Specific heat ratio

6.8.2.2 double getLocalDeltaT (vector< double > *ConservedVariables,* double *MinimumDistance,* double *CFL,* string *gamma,* double *SpecificHeatRatio*)

Calculates the local time step value for a given cell.

Parameters

[IN]	ConservedVariables Conserved variables at cell center of a particular cell
[IN]	MinimumDistance Minimum distance of a given cell
[IN]	CFL

Parameters

[IN]	gamma String which tells that specific heat ratio is constant or function of temperature
SpecificHeatRatio	Specific heat ratio

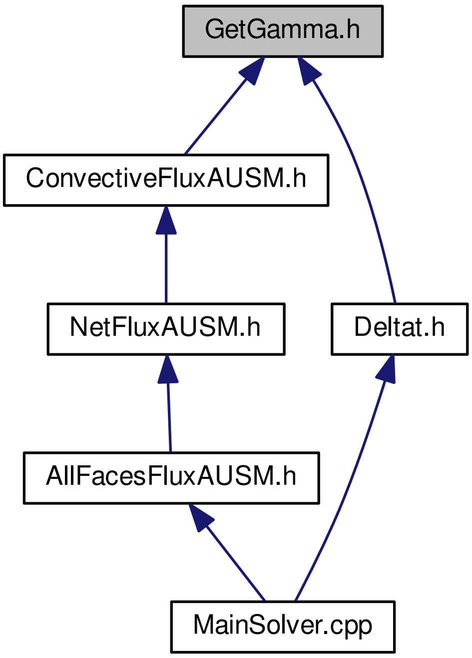
Parameters

deltat	time step
Density	Density in the cell
Pressure	Pressure in the cell
Velocity	Velocity in the cell
VelocitySound	Sound velocity in the cell

6.9 GetGamma.h File Reference

Contains the getgamma() function which calculates the specific heats ratio for given conserved quantities.

This graph shows which files directly or indirectly include this file:



Functions

- double getgamma (std::vector< double > U)

  Function getgamma() calculates the gamma(T) for given conserved quantities in the cell.

6.9.1 Detailed Description

Contains the getgamma() function which calculates the specific heats ratio for given conserved quantities.

6.9.2 Function Documentation

6.9.2.1 double getgamma (std::vector< double > *U*)

Function getgamma() calculates the gamma(T) for given conserved quantities in the cell.

Parameters

in	*U*	Conserved quantities in the cell

Parameters

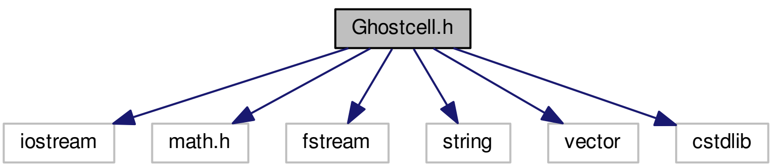
gamma_p	Reference specific heat ratio
T_theta	Reference temperature for gamma calculation
gamma_old	Lower limit of specific heat ratio in the iteration
gamma_new	Upper limit of specific heat ratio in the iteration
gammaTolerance	Tolerance value for specific heat ratio in the iteration
Temperature	Temperature where gamma needs to be calculated

6.10 Ghostcell.h File Reference

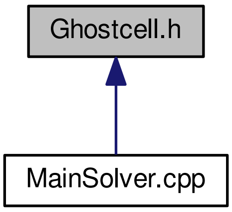
File contains the functions ghostcell(), which defines the ghost cells area vectors and ghost cells volume.

```
#include <iostream>
#include "math.h"
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>
```

Include dependency graph for Ghostcell.h:



This graph shows which files directly or indirectly include this file:



Functions

- void ghostcell (vector< vector< vector< vector< double > > > > Coordinates, vector< vector< vector< vector< double > > > > iFaceAreaVector, vector< vector< vector< vector< double > > > > jFace↵AreaVector, vector< vector< vector< vector< double > > > > kFaceAreaVector, vector< vector< vector< double > > > CellVolume, vector< vector< vector< double > > > &i0GhostCellVolume, vector< vector< vector< double > > > &j0GhostCellVolume, vector< vector< vector< double > > > &k0GhostCellVolume, vector< vector< vector< double > > > &iNiGhostCellVolume, vector< vector< vector< double > > > &jNjGhostCellVolume, vector< vector< vector< double > > > &kNkGhostCellVolume, int Ni, int Nj, int Nk)

  defines the ghost cells area vectors and ghost cells volume.

6.10.1 Detailed Description

File contains the functions ghostcell(), which defines the ghost cells area vectors and ghost cells volume.

Date

18-May-2017

6.10.2 Function Documentation

6.10.2.1 void ghostcell (vector< vector< vector< vector< double > > > > *Coordinates*, vector< vector< vector< vector< double > > > > *iFaceAreaVector*, vector< vector< vector< vector< double > > > > *jFaceAreaVector*, vector< vector< vector< vector< double > > > > *kFaceAreaVector*, vector< vector< vector< double > > > *CellVolume*, vector< vector< vector< double > > > & *i0GhostCellVolume*, vector< vector< vector< double > > > & *j0GhostCellVolume*, vector< vector< vector< double > > > & *k0GhostCellVolume*, vector< vector< vector< double > > > & *iNiGhostCellVolume*, vector< vector< vector< double > > > & *jNjGhostCellVolume*, vector< vector< vector< double > > > & *kNkGhostCellVolume*, int *Ni*, int *Nj*, int *Nk*)

defines the ghost cells area vectors and ghost cells volume.

Parameters

i0GhostCellVolume	Ghost cell volume array at i = 0
j0GhostCellVolume	Ghost cell volume array at j = 0

Parameters

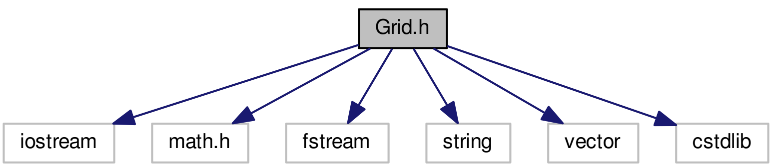
k0GhostCellVolume	Ghost cell volume array at k = 0
iNiGhostCellVolume	Ghost cell volume array at i = Ni
jNjGhostCellVolume	Ghost cell volume array at j = Nj
kNkGhostCellVolume	Ghost cell volume array at k = Nk
Coordinates	4D vector which has the coordinates of all vertices's
iFaceAreaVector	Area vector of all faces the in "i" direction.
jFaceAreaVector	Area vector of all faces the in "j" direction.
kFaceAreaVector	Area vector of all faces the in "k" direction.
CellVolume	Pointer to cell volumes in the live domain
Ni	Number of live cells in in "i" direction
Nj	Number of live cells in in "j" direction
Nk	Number of live cells in in "k" direction

6.11 Grid.h File Reference

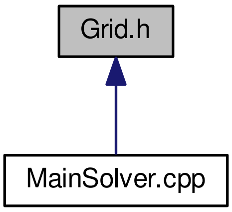
File contains functions, which find the live cell vertices's, live area vectors and live cell volumes for the multiple geometries based on the input given in the input file.

```
#include <iostream>
#include "math.h"
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>
```

Include dependency graph for Grid.h:



This graph shows which files directly or indirectly include this file:



Functions

- double finddeltaz (std::vector< std::vector< double > > DownCoordinates)

    Find the cell side in z direction by taking average of all dx .
- double find_y (double x, std::vector< std::vector< double > > UpperCoordinates)

    finds appropriate "y" for a given x location in the nozzle
- void grid (vector< vector< vector< vector< double > > > > &Coordinate, vector< vector< vector< vector< double > > > > &iFaceAreaVector, vector< vector< vector< vector< double > > > > &jFaceAreaVector, vector< vector< vector< vector< double > > > > &kFaceAreaVector, vector< vector< vector< double > > > &CellVolume, vector< vector< vector< double > > > &ds, int &Ni, int &Nj, int &Nk, string Geometry↩ Option)

    Function defines the area vector and cell volumes of live cells.

6.11.1 Detailed Description

File contains functions, which find the live cell vertices's, live area vectors and live cell volumes for the multiple geometries based on the input given in the input file.

Date

18-May-2017

Warning

For different geometries, change the option in the input file.

6.11.2 Function Documentation

6.11.2.1 double find_y (double *x*, std::vector< std::vector< double > > *UpperCoordinates*)

finds appropriate "y" for a given x location in the nozzle

Parameters

x	Axial location in the nozzle
UpperCoordinates	Upper wall coordinates (x,y) of the nozzle geometry

Returns

"double y"

6.11.2.2 double finddeltaz (std::vector< std::vector< double > > *DownCoordinates*)

Find the cell side in z direction by taking average of all dx .

Parameters

`in`	*DownCoordinates*	(x,y)coordinates of the down wall of the nozzle.

Returns

double deltaz

6.11.2.3 void grid (vector< vector< vector< vector< double > > > > & *Coordinate,* vector< vector< vector< vector< double > > > > & *iFaceAreaVector,* vector< vector< vector< vector< double > > > > & *jFaceAreaVector,* vector< vector< vector< vector< double > > > > & *kFaceAreaVector,* vector< vector< vector< double > > > & *CellVolume,* vector< vector< vector< double > > > & *ds,* int & *Ni,* int & *Nj,* int & *Nk,* string *GeometryOption*)

Function defines the area vector and cell volumes of live cells.

Parameters

`in`	*Coordinate*	Input pointer to the 4D array which stores the coordinates of all live cells
`in`	*iFaceAreaVector*	Input pointer to "i" faces area vector
`in`	*jFaceAreaVector*	Input pointer to "j" faces area vector
`in`	*kFaceAreaVector*	Input pointer to "k" faces area vector
`in`	*CellVolume*	Input pointer to cell volumes
`in`	*ds*	Input pointer to minimum distance
`in`	*Ni*	Number of live cells in "i" direction.
`in`	*Nj*	Number of live cells in "j" direction.
`in`	*Nk*	Number of live cells in "k" direction.
`in`	*GeometryOption*	For which geometry grid should be defined (ie. Nozzle, Duct etc.)

Parameters

N	Total cells in all direction somewhat depends on N

Structure of grid output file ("./Results/outputfiles/CellCenter_ij.csv")

- First line : Number of cell center in i direction and j direction
- then all the Coordinates of the cell centers of ij plane

Warning

To increase the grid density, change the "N"
Do not change the Ni and Nj otherwise you will have to change the code for grid as well, written inside the for loop below

Parameters

UpperCoordinates	Upper wall coordinates (x,y) of the nozzle
DownCoordinates	Down wall coordinates (x,y) of the nozzle

Bug Randomly extra zeros at the end so to remove them pop is used

Warning

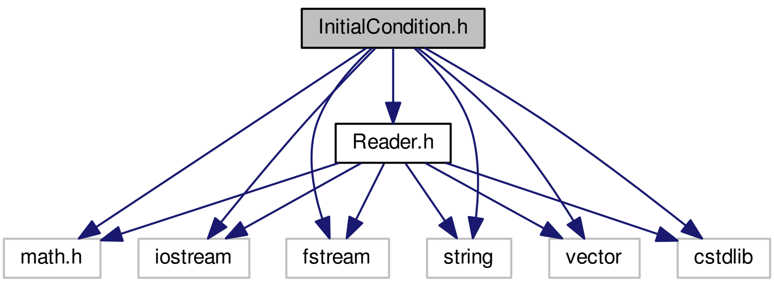
To increase the number of cell in the nozzle(to make the grids fine) increase this **"N"**

6.12 InitialCondition.h File Reference

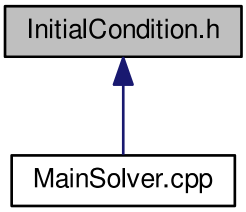
Declares the conserved variables in the live cells according the option given in the input file.

```
#include "math.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>
#include "Reader.h"
```

Include dependency graph for InitialCondition.h:



This graph shows which files directly or indirectly include this file:



Functions

- double find_throat (std::vector< std::vector< double > > UpperWallCoordinates, int &throat_location)

  Function find_throat() Finds the location of the nozzle throat and the area of the throat.
- double getPressure (double InletPressure, double InletMach, double Mach, double SpecificHeatRatio)

  Gives the pressure for a given Mach number using Inlet Mach and pressure.
- double getDensity (double InletDensity, double InletMach, double Mach, double SpecificHeatRatio)

  Gives the density for a given Mach number using Inlet Mach and density.
- double getMachDivergingDuct (double areaRatio)

  getMachDivergingDuct() finds the local Mach number for a given area ratio in the diverging section
- double getMachConvergingDuct (double areaRatio)

  getMachConvergingDuct() finds the local Mach number for a given area ratio in the diverging section
- double getAreaRatio (double Mach)

  give the area ratio for a given Mach number
- void initial_condition (vector< vector< vector< vector< double > > > > &ConservedVariables, vector< vector< vector< vector< double > > > > &ConservedVariablesNew, int Ni, int Nj, int Nk, double SpecificHeatRatio)

  Function initial_condition() defines the initial values of the conserved quantities in the live cells.

6.12.1 Detailed Description

Declares the conserved variables in the live cells according the option given in the input file.

Date

18-May-2017

6.12.2 Function Documentation

6.12.2.1 double find_throat (std::vector< std::vector< double > > *UpperWallCoordinates*, int & *throat_location*)

Function find_throat() Finds the location of the nozzle throat and the area of the throat.

Parameters

UpperWallCoordinates	Coordinates of the upper wall of the nozzle
throat_location	Location of the throat

Returns

Area of the throat

Parameters

throat_area	Area of the throat

6.12.2.2 double getAreaRatio (double *Mach*)

give the area ratio for a given Mach number

Returns

Area ratio

6.12.2.3 double getDensity (double *InletDensity,* double *InletMach,* double *Mach,* double *SpecificHeatRatio*)

Gives the density for a given Mach number using Inlet Mach and density.

Parameters

InletDensity	Density at the inlet
InletMach	Mach at the inlet
Mach	Mach number at the location where density is needed
SpecificHeatRatio	Specific heat capacity ratio

Returns

Density

6.12.2.4 double getMachConvergingDuct (double *areaRatio*)

getMachConvergingDuct() finds the local Mach number for a given area ratio in the diverging section

Parameters

areaRatio	Area ratio of the location

Returns

Mach number in the converging section

Parameters

MachConverging	This will always be below 1 so initially let's keep it 100

6.12.2.5 double getMachDivergingDuct (double *areaRatio*)

getMachDivergingDuct() finds the local Mach number for a given area ratio in the diverging section

Parameters

areaRatio	Area ratio of the location

Returns

Mach number in the diverging section

Parameters

MachDiverging	This will always be above 1 so initially let's keep it 100

6.12.2.6 double getPressure (double *InletPressure,* double *InletMach,* double *Mach,* double *SpecificHeatRatio*)

Gives the pressure for a given Mach number using Inlet Mach and pressure.

Parameters

InletPressure	Pressure at the inlet
InletMach	Mach at the inlet
Mach	Mach number at the location where pressure is needed
SpecificHeatRatio	Specific heat capacity ratio

Returns

Pressure

6.12.2.7 void initial_condition (vector< vector< vector< vector< double > > > > & *ConservedVariables,* vector< vector< vector< vector< double > > > > & *ConservedVariablesNew,* int *Ni,* int *Nj,* int *Nk,* double *SpecificHeatRatio*)

Function initial_condition() defines the initial values of the conserved quantities in the live cells.

There are "5" options are available for initial condition. Which are

1. InitialCondition = ZeroVelocity Zero velocity inside the domain and total quantities are given at the inlet
2. InitialCondition = FreeStreamParameterEverywhare Free stream parameters are filled inside the domain
3. InitialCondition = FreeStreamParameterAndZeroVelocity
4. InitialCondition = StartFromPreviousSolution
5. InitialCondition = NozzleBasedOn1DCalculation

Parameters are varying inside the nozzle domain based on the 1D is-entropic relation.

Warning

"InitialCondition = NozzleBasedOn1DCalculation" Is only valid for nozzle simulation. DO NOT use it for any other simulation

Parameters

in	*ConservedVariables*	This is the pointer to the 4D vector where all the conserved variables of previous time step are stored.
in	*ConservedVariablesNew*	This is the pointer to the 4D vector where all the conserved variables of next/new time step are stored.
in	*Ni*	Number of cells in in "i" direction.
in	*Nj*	Number of cells in in "j" direction.
in	*Nk*	Number of cells in in "k" direction.
in	*SpecificHeatRatio*	Specific heat ratio

Returns

void

Inlet conditions are user given data. one has to mention the free stream parameters at inlet (ex. static pressure (P_inf), temperature(T))

throat_location Location of the throat

Parameters

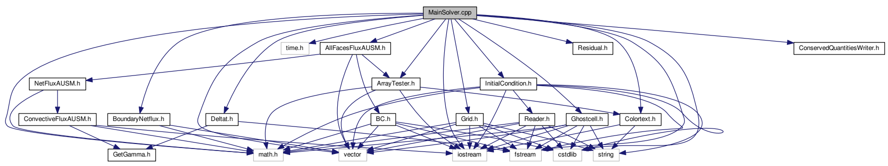
throat_area	Area at the throat
Density	Density at local point
Pressure	Pressure at local point
Temperature	Temperature at local point
VelocityMagnitude	Velocity magnitude at local point
Mach	Mach number at a location
AreaRatio	Area ratio at a location with throat area

6.13 MainSolver.cpp File Reference

This is the Main file which runs the simulation.

```
#include "iostream"
#include <vector>
#include <fstream>
#include "math.h"
#include "time.h"
#include <cstdlib>
#include "AllFacesFluxAUSM.h"
#include "BoundaryNetflux.h"
#include "Deltat.h"
#include "Residual.h"
#include "InitialCondition.h"
#include "Grid.h"
#include "Ghostcell.h"
#include "ArrayTester.h"
#include "ConservedQuantitiesWriter.h"
#include "Colortext.h"
```

Include dependency graph for MainSolver.cpp:



Functions

- int main ()

6.13.1 Detailed Description

This is the Main file which runs the simulation.

Author

Kuldeep Singh

Date

May, 2017

6.13.2 Function Documentation

6.13.2.1 int main ( )

Parameters

StartTime	Simulation starting time
EndTime	Simulation ending time
TotalIteration	Total iterations

Parameters

Scheme	"AUSM" or "Roe"
TimeSteping	Whether to use local "delta t" or global "delta t"
GeometryOption	Using this option grids (area vector and the cell volumes) will be defined appropriately
gamma	Option, whether constant Specific heat ratio is constant or variable
CFL	
SpecificHeatRatio	Specific heat ratio in case of constant gamma
deltat	Time step

Reading the input file

Reading the input file over

Parameters

Ni	Number of live cells in in "i" direction.
Nj	Number of live cells in in "j" direction.
Nk	Number of live cells in in "k" direction.

Warning

Final value of the Ni,Nj,Nk has been decided inside the grid() function. So, do not use these parameters until the grid function is called

Parameters

Coordinates	This is a 4D vector which has the coordinated of all vertices's
iFaceAreaVector	This is a 4D vector which has the area vector of all faces which are in "i" direction.
jFaceAreaVector	This is a 4D vector which has the area vector of all faces which are in "j" direction.
kFaceAreaVector	This is a 4D vector which has the area vector of all faces which are in "k" direction.
CellVolume	Input pointer to cell volumes
delta_s	Minimum distance

After calling grid function all the live cell quantities will be decided

Testing the array declared using grid function

Parameters

i0GhostCellVolume	Ghost cell volume array at i = 0
j0GhostCellVolume	Ghost cell volume array at j = 0
k0GhostCellVolume	Ghost cell volume array at k = 0
iNiGhostCellVolume	Ghost cell volume array at i = Ni
jNjGhostCellVolume	Ghost cell volume array at j = Nj
kNkGhostCellVolume	Ghost cell volume array at k = Nk
ConservedVariables	This is the pointer to the 4D vector where all the conserved variables ([Density , x-momentum, y-momentum, z-momentum, Energy]) of previous time step are stored.
ConservedVariablesNew	This is the pointer to the 4D vector where all the conserved variables ([Density , x-momentum, y-momentum, z-momentum, Energy]) of current/new time step are stored.

Initializing the domain

Parameters

iFacesFlux	This is a 4D vector which has the fluxes of all faces which are in "i" direction.
jFacesFlux	This is a 4D vector which has the fluxes of all faces which are in "j" direction.
kFacesFlux	This is a 4D vector which has the fluxes of all faces which are in "k" direction.

Opening the "Residual.csv" file to store the all the residuals

Iterations starts here

calculating the global time step after every time iteration

Calculating the local "delta t" here

Parameters

NetFlux	Normal component of the flux across the interface

Updating the previous time step conserved quantities

Net Fluxes integration at the boundaries

Parameters

iNetFlux	Net flux per unit time at "i=0"
iNetFlux	Net flux per unit time at "i=Ni or imax"
jNetFlux	Net flux per unit time at "j=0"
jNetFlux	Net flux per unit time at "Nj=0 or jmax"
kNetFlux	Net flux per unit time at "k=0"
kNetFlux	Net flux per unit time at "Nk=0"

this function "boundaryNetflux()" calculates the flux at all the boundary

See also

boundaryNetflux()

here writing the boundary flux in the file

before going to the new time step updating the old conserved variables by new ones

Writing the Conserved quantities in the output file

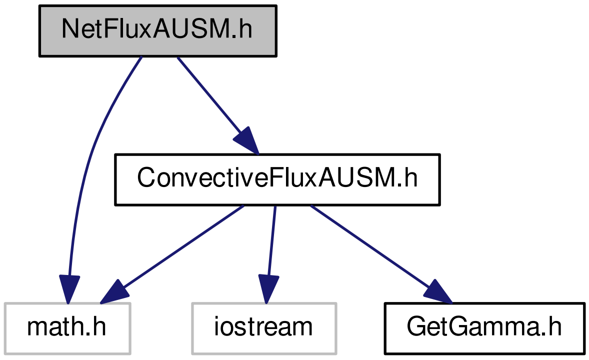
Simulation ends here

6.14 NetFluxAUSM.h File Reference

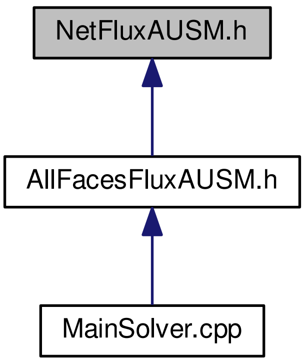
Calculates the AUSM net flux(convective and pressure) at a cell interface.

```
#include "math.h"
#include "ConvectiveFluxAUSM.h"
```

Include dependency graph for NetFluxAUSM.h:



This graph shows which files directly or indirectly include this file:



Classes

- class netfluxAUSM

  Calculates the net AUSM flux at the interface.

6.14.1 Detailed Description

Calculates the AUSM net flux(convective and pressure) at a cell interface.

Date

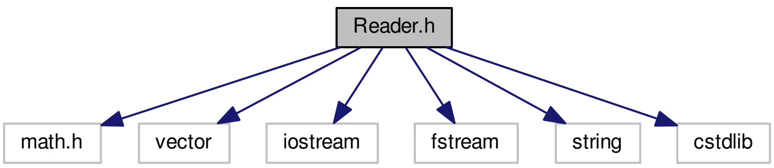
18-May-2017

6.15 Reader.h File Reference

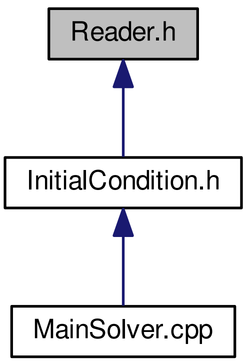
Contains the function ReadConservedVariables(), which reads the conserved variables from the file.

```
#include <math.h>
#include "vector"
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
```

Include dependency graph for Reader.h:



This graph shows which files directly or indirectly include this file:



Functions

- void ReadConservedVariables (vector< vector< vector< vector< double > > > > &Uin, vector< vector< vector< vector< double > > > > &UinNew, int Ni, int Nj, int Nk)

    Reads the conserved variables from the file.

6.15.1 Detailed Description

Contains the function ReadConservedVariables(), which reads the conserved variables from the file.

Date

18-May-2017

6.15.2 Function Documentation

6.15.2.1 void ReadConservedVariables (vector< vector< vector< vector< double > > > > & *Uin,* vector< vector< vector< vector< double > > > > & *UinNew,* int *Ni,* int *Nj,* int *Nk*)

Reads the conserved variables from the file.

Parameters

in	*Uin*	pointer to the conserved variable matrix of current time step
in	*UinNew*	pointer to the conserved variable matrix of the new time step
in	*Ni*	Number of cells in in "i" direction.
in	*Nj*	Number of cells in in "j" direction.
in	*Nk*	Number of cells in in "k" direction.

Returns

Magnitude of a 3D vector

Parameters

	Residual	Vector which stores all the residuals
in	*iteration*	Current time iteration
in	*ConservedVariables*	Conserved variable matrix of current previous time step
in	*ConservedVariablesNew*	Conserved variable matrix of the next time step
in	*Ni*	Number of cells in in "i" direction.
in	*Nj*	Number of cells in in "j" direction.
in	*Nk*	Number of cells in in "k" direction.

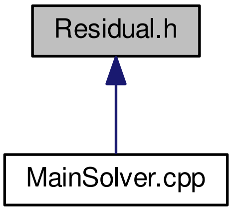
Returns

Magnitude of a 3D vector

6.16 Residual.h File Reference

Contains the function residual(), which prints the all the residuals.

This graph shows which files directly or indirectly include this file:



Functions

- void residual (vector< double > &Residual, int iteration, vector< vector< vector< vector< double > > > > ConservedVariables, vector< vector< vector< vector< double > > > > ConservedVariablesNew, int Ni, int Nj, int Nk)

6.16.1 Detailed Description

Contains the function residual(), which prints the all the residuals.

Date

18-May-2017

6.16.2 Function Documentation

6.16.2.1 void residual (vector< double > & *Residual,* int *iteration,* vector< vector< vector< vector< double > > > > *ConservedVariables,* vector< vector< vector< vector< double > > > > *ConservedVariablesNew,* int *Ni,* int *Nj,* int *Nk*)

Parameters

DensityResidual	Density residual
xMomentumResidual	x Momentum residual
yMomentumResidual	y Momentum residual
zMomentumResidual	z Momentum residual
Energy	residual

Index