Reynolds Documentation

Release 0.1

Deepak Surti

Sep 16, 2022

User Documentation

1 FAQ **3**

2 Using reynolds **5**
- 2.1 Requirements . . . 5
- 2.2 Pre-requisites . . . 5
- 2.3 Installation . . . 5
- 2.4 Starting OpenFoam . . . 6
- 2.5 Generating blockMeshDict . . . 6
- 2.6 Running a solver . . . 7
- 2.7 Running with Blender using Docker . . . 8

3 Installation **9**
- 3.1 OpenFoam Dependency . . . 9

4 Package Design **11**
- 4.1 foam . . . 11
- 4.2 dict . . . 11
- 4.3 tests . . . 12

5 Contributing **13**

6 reynolds package **15**
- 6.1 Subpackages . . . 15
- 6.2 Module contents . . . 16

7 Indices and tables **17**

Python Module Index **19**

Index **21**

Reynolds is a full featured, scriptable python API of components for the preprocessing and solver environments of OpenFoam. These components can be easily combined to build a GUI using any 3D graphics package such as Blender.

The main documentation for this library is split into:

- *User Documentation*
- *Developer Documentation*
- *API Docs*

CHAPTER 1

FAQ

1. Why this library?

A. The intention behind developing this library is to have a set of python components that can execute the preprocessing and solver stages of OpenFoam, such that they can be easily integrated with a 3D graphics package to develop a GUI for these stages. Thus one of the important design constraint is that library be GUI agnostic!

2. Does this currently work with any 3D graphics package?

A. Yes, currently there is a reference implementation for integrating with a 3D graphics package using Blender. There exists a Blender add-on, that uses this library to start openfoam, generate a blockMesh and run a solver.

   You can checkout this add on here. You can also run this Blender add-on with this Docker image on Ubuntu.

3. What is the current state of this library?

A. This library right now supports blockMesh and executing any solver which works with a mesh generated by blockMesh.

4. What mesh generators does it support?

A. Currently, blockMesh is supported. More, including snappyHexMesh will be added in upcoming versions.

5. Is this cross-platform?

A. Currently, it supports macOS and linux platforms. However the Docker image runs only on Linux (tested on Ubuntu) and can work with macOS if you can share the mac host X11. The Windows platform has not been tested.

   The final version will support all of Linux, macOS and Windows.

6. Do I need to install openfoam4?

A. Yes, you need to install openfoam4 for this library to work, it is a dependency. See *Pre-requisites* for installing openfoam4.

CHAPTER 2

Using reynolds

2.1 Requirements

- Python 3.6 or later
- OpenFoam version 4.x
- macOS 10.10 or later
- Ubuntu 14.x or later

2.2 Pre-requisites

You need to install OpenFoam version 4.x on macOS or Ubuntu.

Please note that reynolds depends on an OpenFoam sparsbundle on macOS.

The following are recommended guides for installing openfoam:

- Installing on Mac
- Installing on Ubuntu

2.3 Installation

1. Clone the repository from Github:

```
git clone git@github.com:dmsurti/reynolds.git
```

2. Install the requirements:

```
pip install -r requirements.txt
```

3. Run setup.py:

```
python setup.py install
```

2.4 Starting OpenFoam

To use OpenFoam in your python environment, you can use the FoamRunner class which will source the openfoam environment variables. See the code listing below:

```
from reynolds.foam.start import FoamRunner

foam_runner = FoamRunner()
foam_runner.start()
```

This loads your environment with various openFoam utilities such as blockMesh and various solvers so they can be executed with a python process using *POpen*.

2.5 Generating blockMeshDict

To generate a blockMeshDict for the cavity tutorial, you could do:

```
from reynolds.dict.parser import ReynoldsFoamDict

block_mesh_dict = ReynoldsFoamDict('blockMeshDict.foam')
self.assertIsNotNone(block_mesh_dict)

# add vertices
vertices = []
vertices.append([0, 0, 0])
vertices.append([1, 0, 0])
vertices.append([1, 1, 0])
vertices.append([0, 1, 0])
vertices.append([0, 0, 0.1])
vertices.append([1, 0, 0.1])
vertices.append([1, 1, 0.1])
vertices.append([0, 1, 0.1])
block_mesh_dict['vertices'] = vertices

# add blocks
blocks = []
blocks.append('hex')
blocks.append([0, 1, 2, 3, 4, 5, 6, 7])
blocks.append([20, 20, 1])
blocks.append('simpleGrading')
blocks.append([1, 1, 1])
block_mesh_dict['blocks'] = blocks
self.assertEqual(block_mesh_dict['blocks'],
                 ['hex', [0, 1, 2, 3, 4, 5, 6, 7], '(20 20 1)',
                 'simpleGrading', '(1 1 1)'])

# add edges
self.assertEqual(block_mesh_dict['edges'], [])
edges = []
```

(continues on next page)

(continued from previous page)

```
edges.append('arc')
edges.append(1)
edges.append(5)
edges.append([1.1, 0.0, 0.5])
block_mesh_dict['edges'] = edges

boundary = []
# add moving wall
boundary.append('movingWall')
moving_wall = {}
moving_wall['faces'] = [[3, 7, 6, 2]]
moving_wall['type'] = 'wall'
boundary.append(moving_wall)
# add fixed walls
boundary.append('fixedWalls')
fixed_walls = {}
fixed_walls['faces'] = [[0, 4, 7, 3], [2, 6, 5, 1], [1, 5, 4, 0]]
fixed_walls['type'] = 'wall'
boundary.append(fixed_walls)
# add front and back
boundary.append('frontAndBack')
front_and_back = {}
front_and_back['faces'] = [[0, 3, 2, 1], [4, 5, 6, 7]]
front_and_back['type'] = 'empty'
boundary.append(front_and_back)
block_mesh_dict['boundary'] = boundary

# add mergePatchPairs
mergePatchPairs = []
mergePatchPairs.append(['inlet1', 'outlet1'])
mergePatchPairs.append(['inlet2', 'outlet2'])
block_mesh_dict['mergePatchPairs'] = mergePatchPairs

print(block_mesh_dict)
```

The above generates an in memory blockMeshDict. To write this to a file on disk, you can do:

```
# case_dir is the absolute path to your case directory on disk
file_path = os.path.join(case_dir, 'system', 'blockMeshDict')

with open(file_path, 'w') as f:
    f.write(str(block_mesh_dict))
```

2.6 Running a solver

You can run any openfoam solver available in the openfoam environment which has been sourced, see *Installation* instructions. For example, to run the icoFoam solver used in the cavity tutorial, you can do:

```
from reynolds.foam.cmd_runner import FoamCmdRunner

# case_dir is the absolute path to your case directory on disk
solver_runner = FoamCmdRunner(cmd_name='icoFoam', case_dir=cavity_case_dir)
for info in solver_runner.run():
    pass # client can stream this info live
```

(continues on next page)

(continued from previous page)

```
if solver_runner.run_status: # All is well
    print("Success")
else:
    print("Failure")
```

On exactly the same lines, you can run any other OpenFoam command such as *blockMesh* using the *FoamCmdRunner*.

2.7 Running with Blender using Docker

You can use Blender with an add-on that invokes this reynolds API to start openfoam, generate a blockMeshDict and run a solver.

The simplest way to run Blender with this addon is to use this Docker file, which can be installed on Ubuntu, and runs the Blender GUI with this add-on.

You can refer to the docker image repository homepage for instructional videos.

CHAPTER 3

Installation

You can install with the following simple 2 step process:

1. Fork this repository.
2. Clone your forked repository.

3.1 OpenFoam Dependency

This library depends on OpenFoam 4.x installed on your Mac or Linux machine. You can use the following guides to install openfoam4.

- Installing on Mac
- Installing on Ubuntu

CHAPTER 4

Package Design

This project is divided into the following packages:

4.1 foam

The foam package is responsible to start openfoam with the environment sourced into the target python's os.environ, so that openfoam utilities such as blockMesh and solvers etc can be executed using a python subprocess with POpen.

This package contains a single FoamRunner class which does the above.

This package also provides a class FoamCmdRunner which is used to execute any OpenFoam command that is sourced after starting OpenFoam. The command runner class requires the command name and the case directory in which to execute the command.

Both the starter and the runner class emit the progress which can be *yield'ed in the client code that uses these. For a sample, see the 'tests/test_foam_cmd_runner.py*.

4.2 dict

The dict package is responsible for reading/writing any OpenFoam dict file.

The class *ReynoldsFoamDict* uses *PyFoam* provided *ParsedParameterFile* to read and write foam dicts. You need to initialize a *ReynoldsFoamDict* with a template for the foam dict you want to read/write. The templates are availabel in *dict/templates* directory.

The requirement of a template dict file is because ParsedParameterFile cannot be intialized without a foam dict file and so we use the template to start with an initial, empty foam dict.

See: *blockMeshDict.foam* template under *dict/templates*.

4.3 tests

The above classes in various packages are tested and reading the test code and API docs can be a good starting point to delve deeper into the code.

CHAPTER 5

Contributing

To contribute to reynolds:

- Please open an issue describing the bug, enhancement or any other improvement.
- If possible, please supply the case directory that can help demonstrate the issue.
- If the design involves a larger refactor, please open a issue to dicuss the refactor.
- After discussion on the issue, you can submit a Pull Request by forking this project.
- Please accompany your Pull Request with updates to test code.

CHAPTER 6

reynolds package

6.1 Subpackages

6.1.1 reynolds.dict package

Submodules

reynolds.dict.parser

class reynolds.dict.parser.**ReynoldsFoamDict**(*dict_template_filename*, *solver_name=None*)

Bases: dict

Read and write any foam dictionary, using python dictionary.

__init__(*dict_template_filename*, *solver_name=None*)

Creates an inital empty python dictionary for a given foam dict.

Parameters **dict_template_filname** – The foam dict template filename

Module contents

6.1.2 reynolds.foam package

Submodules

reynolds.foam.cmd_runner module

reynolds.foam.start module

class reynolds.foam.start.**FoamRunner**

Bases: object

A class to start the OpenFoam runtime environment.

This works by loading the openfoam env variables into the python os.environ thereby allowing openfoam tools such as blockMesh and various solvers to be executed using a python process.

This is a platform dependent class that supports linux and macOS only as the process of loading openfoam env variables is different on the two supported platforms.

__init__()
Initializes the runner with:

1. Foam path: which is path to the openfoam installation
2. Source path: which is the path to the openfoam bash script to source openfoam env variables.

The foam path and source path are setup as per the os platform.

shell_source(*script*)
Updates the environment with variables sourced from openfoam bash script.

Parameters **script** – The path to the openform etc/bashrc script to be sourced

Returns True if env sourced succesfully. False otherwise.

start()
Starts the openfoam environment as such on the following os platforms:

1. macOS: Loads the openfoam sparsebundle and sources the openfoam env variables.
2. linux: Sources the openfoam env variables.

Returns True if the process was successful, else False on macOs and linux AssertionError for any other non supported os platform.

Module contents

6.2 Module contents

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

reynolds, 16
reynolds.dict, 15
reynolds.dict.parser, 15
reynolds.foam, 16
reynolds.foam.start, 15

Index

Symbols

__init__() (*reynolds.dict.parser.ReynoldsFoamDict method*), 15
__init__() (*reynolds.foam.start.FoamRunner method*), 16

F

FoamRunner (*class in reynolds.foam.start*), 15

R

reynolds (*module*), 16
reynolds.dict (*module*), 15
reynolds.dict.parser (*module*), 15
reynolds.foam (*module*), 16
reynolds.foam.start (*module*), 15
ReynoldsFoamDict (*class in reynolds.dict.parser*), 15

S

shell_source() (*reynolds.foam.start.FoamRunner method*), 16
start() (*reynolds.foam.start.FoamRunner method*), 16