



fc hypermesh Octave package, User's Guide *

François Cuvelier[†] Gilles Scarella[‡]

June 29, 2018

Abstract

This object-oriented Octave package allows to generate conforming meshes of hypercubes, hyperrectangles or of any d -orthotopes by simplices or orthotopes with their m -faces. It was created to show the implementation of the algorithms of [?]. The `fc hypermesh` package uses Octave objects and is provided with meshes visualisation tools for dimension less than or equal to 3.

0 Contents

| | | |
|----------|--|----------|
| 1 | Introduction | 2 |
| 2 | Installation | 2 |
| 3 | Installation via pkg command | 3 |
| 4 | All-in-one installation | 3 |
| 5 | Using the fc_hypermesh package | 4 |
| 5.1 | Class object OrthMesh | 4 |
| 5.2 | 2d-orthotope meshing by simplices | 6 |
| 5.3 | 3d-orthotope meshing by simplices | 7 |
| 5.4 | 2d-orthotope meshing by orthotopes | 8 |

*Compiled with Octave 4.4.0, packages `fc_hypermesh-0.0.8` and `fc_tools-0.0.23` under `cosmos` computer

[†]Université Paris 13, Sorbonne Paris Cité, LAGA, CNRS UMR 7539, 99 Avenue J-B Clément, F-93430 Villetaneuse, France, cuvelier@math.univ-paris13.fr

[‡]Université Côte d'Azur, CNRS, LJAD, F-06108 Nice, France, gilles.scarella@unice.fr.

This work was partially supported by the ANR project DEDALES under grant ANR-14-CE23-0005.

| | | |
|----------|--|-----------|
| 5.5 | 3d-orthotope meshing by orthotopes | 9 |
| 5.6 | Mapping of a 2d-orthotope meshing by simplices | 10 |
| 5.7 | 3d-orthotope meshing by orthotopes | 11 |
| 6 | Benchmarking | 12 |
| 6.1 | fc_bench.bench01 function | 12 |
| 6.2 | Examples | 13 |
| 7 | Mesh refinement | 13 |
| 7.1 | Non-conforming mesh refinement | 13 |

1 Introduction

The [@hypermesh](#) package contains a simple class object `OrthMesh` which permits, in any dimension $d \geq 1$, to obtain a simplicial mesh or orthotope mesh with all their m -faces, $0 \leq m < d$. It is also possible with the method function `plotmesh` of the class object `OrthMesh` to represent a mesh or its m -faces for $d \leq 3$.

This package was tested under

Windows 10.0.16299: with Octave 4.2.0, 4.2.1 and 4.2.2

macOS High Sierra 10.13.4: with Octave 4.2.1 (installed with homebrew)

Ubuntu 16.04.3 LTS: with Octave 4.2.0, 4.2.1 and 4.2.2 (all compiled from source)

Ubuntu 17.10: with Octave 4.2.0 and 4.2.1 and 4.2.2 (all compiled from source)

centOS 7.4: with Octave 4.2.0, 4.2.1 and 4.2.2 (all compiled from source)

Fedora 27: with Octave 4.2.0, 4.2.1 and 4.2.2 (all compiled from source)

OpenSUSE Leap 42.3: with Octave 4.2.0, 4.2.1 and 4.2.2 (all compiled from source)

It is not compatible with Octave 4.0.x and previous.

In the following section, the class object `OrthMesh` is presented. Thereafter some warning statements on the memory used by these objects in high dimension are given. Finally computation times for orthotope meshes and simplicial meshes are given in dimension $d \in \llbracket 1, 5 \rrbracket$.

2 Installation

Here are two methods of installations. The first uses the Octave `pkg` command and the second a provided Octave script.

3 Installation via pkg command

- Download the packages. For example, in a terminal:

```
# wget http://www.math.univ-paris13.fr/~cuvelier/software/codes/  
  octave/fc-tools/0.0.23/fc-tools-0.0.23.tar.gz  
# wget http://www.math.univ-paris13.fr/~cuvelier/software/codes/  
  octave/fc-hypermesh/0.0.8/fc-hypermesh-0.0.8.tar.gz
```

- Under Octave :

```
>> pkg install fc-tools-0.0.23.tar.gz  
>> pkg install fc-hypermesh-0.0.8.tar.gz
```

- Now to use `fc-hypermesh` in any Octave session, it is necessary to load the package:

```
>> pkg load fc-hypermesh
```

- To try the package, one can launch a demo:

```
>> fc_hypermesh_demo01
```

For uninstalling the package, just do in an Octave session:

```
>> pkg uninstall fc-hypermesh  
>> pkg uninstall fc-tools
```

4 All-in-one installation

For this method, one just has to get/download the install file

`ofc_hypermesh_install.m`

or get it on the dedicated web page. Thereafter, it should be run under Octave. This command downloads, extracts and configures the `fc-hypermesh` and the required `fc-tools` packages in the current directory.

For example, to install this package in directory `~/Octave/packages`, in a terminal one can do:

```
# mkdir -p ~/Octave/packages  
# cd ~/Octave/packages  
# wget http://www.math.univ-paris13.fr/~cuvelier/software/codes/Octave/fc-  
  hypermesh/0.0.8/ofc_hypermesh_install.m
```

Then in a Octave terminal run the following commands

```
>> cd ~/Octave/packages  
>> ofc_hypermesh_install
```

This is the output of the `ofc_hypermesh_install` command:

```

Parts of the GNU Octave <fc-hypermesh> package.
Copyright (C) 2017-2018 Francois Cuvelier <cuvelier@math.univ-paris13.fr>

1- Downloading and extracting the packages
2- Setting the <fc-hypermesh> package
Write in ~/Octave/packages//fc_hypermesh-0.0.8/configure_loc.m ...
3- Using toolboxes :
->          fc-tools : 0.0.23
->          fc-hypermesh : 0.0.8
*** Using instructions
To use the <fc-hypermesh> toolbox:
addpath('~/Octave/packages//fc_hypermesh-0.0.8')
fc_hypermesh.init()

See ~/Octave/packages/ofc_hypermesh_set.m

```

The complete package (i.e. with all the other needed packages) is stored in the directory `~/Octave/packages/fc-hypermesh-full` and, for each Octave session, one has to set the package by:

```

>> addpath('~/Octave/packages/fc-hypermesh-full/ofc-hypermesh-0.0.8')
>> fc_hypermesh.init()

```

To **uninstall**, one just has to delete directory `~/Octave/packages/fc-hypermesh-full`

5 Using the hypermesh package

First of all, the main class object `OrthMesh` is presented. Thereafter some usage samples are given.

5.1 Class object `OrthMesh`

The aim of the class object `OrthMesh` is to efficiently create an object which contains a mesh of a d -orthotope and all its m -face meshes. An elementary mesh class object `EltMesh` is used to store only one mesh, the main mesh as well as any of the m -face meshes. This class `EltMesh` also simplifies (for me) the codes writing and its fields are the following:

- `d`, space dimension
- `m`, kind of mesh ($m = d$ for the main mesh)
- `type`, 0 for simplicial mesh or 1 for orthotope mesh
- `nq`, number of vertices
- `q`, vertices array of dimension d -by-`nq`
- `nme`, number of mesh elements
- `me`, connectivity array of dimension $(d + 1)$ -by-`nme` for simplices elements or 2^d -by-`nme` for orthotopes elements
- `toGlobal`, index array linking local array `q` to the one of the main mesh

- `label`, name/number of this elementary mesh
- `color`, color of this elementary mesh (for plotting purpose)

Let the d -orthotope defined by $[a_1, b_1] \times \dots \times [a_d, b_d]$. The class object `OrthMesh` corresponding to this d -orthotope contains the main mesh and all its m -face meshes, $0 \leq m < d$. Its Fields are the following

- `d`: space dimension
- `type`: string 'simplicial' or 'orthotope' mesh
- `Mesh`: main mesh as an `EltMesh` object
- `Faces`: list of arrays of `EltMesh` objects such that `Faces(1)` is an array of all the $(d - 1)$ -face meshes, `Faces(2)` is an array of all the $(d - 2)$ -face meshes, and so on
- `box`: a d -by-2 array such that `box(i,1) = ai` and `box(i,2) = bi`.

5.1.1 Constructor

```
Oh = OrthMesh(d,N)
Oh = OrthMesh(d,N, key,value, ...)
```

Description

```
Oh = OrthMesh(d,N)
```

Generates the `OrthMesh` object `Oh` which contains which contains a simplicial mesh of the unit d -orthotope and all its m -face meshes.

```
Oh = OrthMesh(d,N, key,value, ...)
```

Some optional `key/value` pairs arguments are available with `key`:

- `'type'` : used to select the kind of elements used for meshing. The default value is 'simplicial' and otherwise 'orthotope' can be used.

```
Oh = OrthMesh(3,10, 'type','orthotope')
```

- `'box'` : used to specify the d -orthotope $[a_1, b_1] \times \dots \times [a_d, b_d]$ by setting `value` as an d -by-2 array such that `ai = value(i,1)` and `bi = value(i,2)`.

```
Oh = OrthMesh(3,10, 'box',[-1 1;-2 2;0 3])
```

- `'m_min'` : used to only mesh the m -Faces for m in $[[m, d]]$. Default value is 0.

```
Oh = OrthMesh(3,10, 'm_min',2)
```

- `'mapping'` : used to apply on the mesh a mapping function given by a function handle.

```
Oh = OrthMesh(3,10, 'mapping',@(q) [q(1,:)+sin(q(2,:));q(2,:);q(3,:)])
```

5.1.2 plotmesh method

The `plotmesh()` member function can be used to represent the mesh given by an `OrthMesh` object if the space dimension is less than or equal to 3.

Syntaxe

```
obj.plotmesh()  
obj.plotmesh(key, value, ...)
```

Description

```
obj.plotmesh()
```

```
obj.plotmesh(key, value, ...)
```

Some optional key/value pairs arguments are available with key:

- 'legend' : if value is True, a legend is displayed. Default is False.
- 'm' : plots all the m-faces of the mesh. Default $m = d$ i.e. the main mesh. ($0 \leq m \leq d$)
- 'color' : use to specify the color of the mesh.
- ...

Other key/value pairs arguments can be used depending of `obj.d` and `obj.m` values and they are those of the plotting function used:

- with `obj.d=3` and `obj.m=3`, `patch` function is used;
- with `obj.d=3` and `obj.m=2`, `trimesh` function is used for simplicial mesh and `patch` function is used for orthotope mesh;
- with `obj.d=3` and `obj.m=1`, `line` function is used;
- with `obj.d=3` and `obj.m=0`, `scatter3` function is used;
- with `obj.d=2` and `obj.m=2`, `triplot` function is used for simplicial mesh and `patch` function is used for orthotope mesh;
- with `obj.d=2` and `obj.m=1`, `line` function is used;
- with `obj.d=2` and `obj.m=0`, `scatter` function is used;
- with `obj.d=1` and `obj.m=1`, `line` function is used;
- with `obj.d=1` and `obj.m=0`, `scatter` function is used;

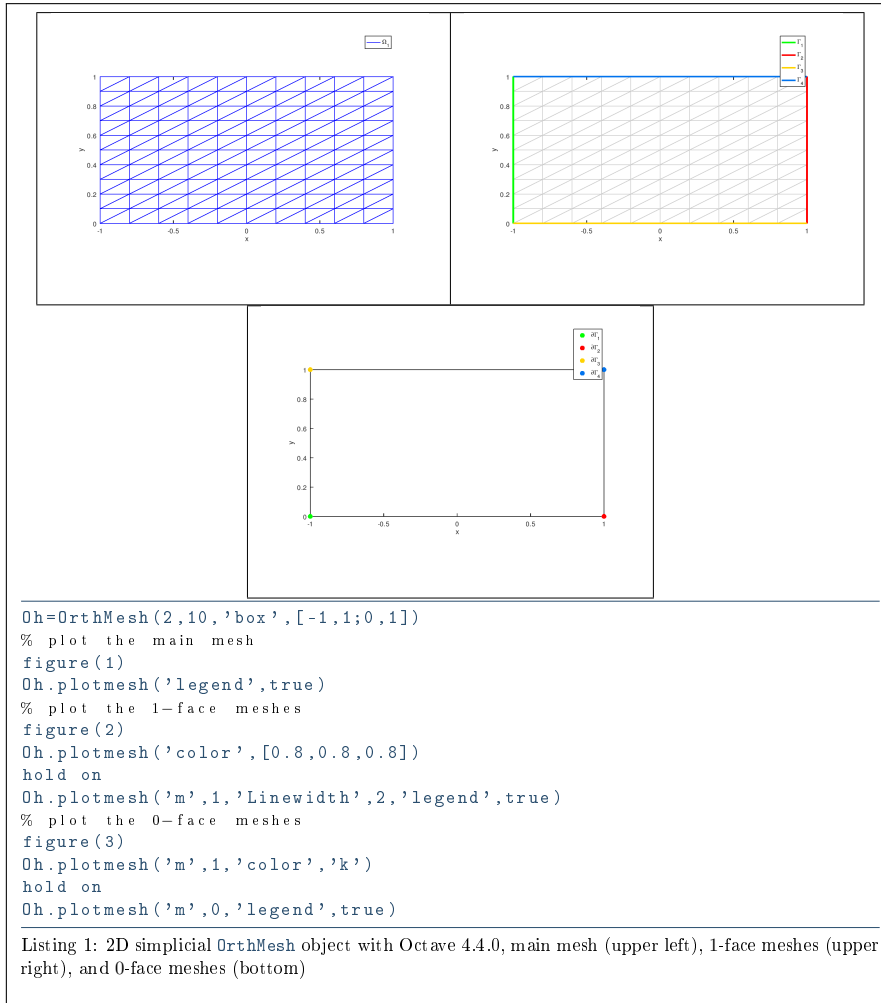
5.2

2d-orthotope meshing by simplices

In Listing 1, an `OrthMesh` object is built under Octave by using command

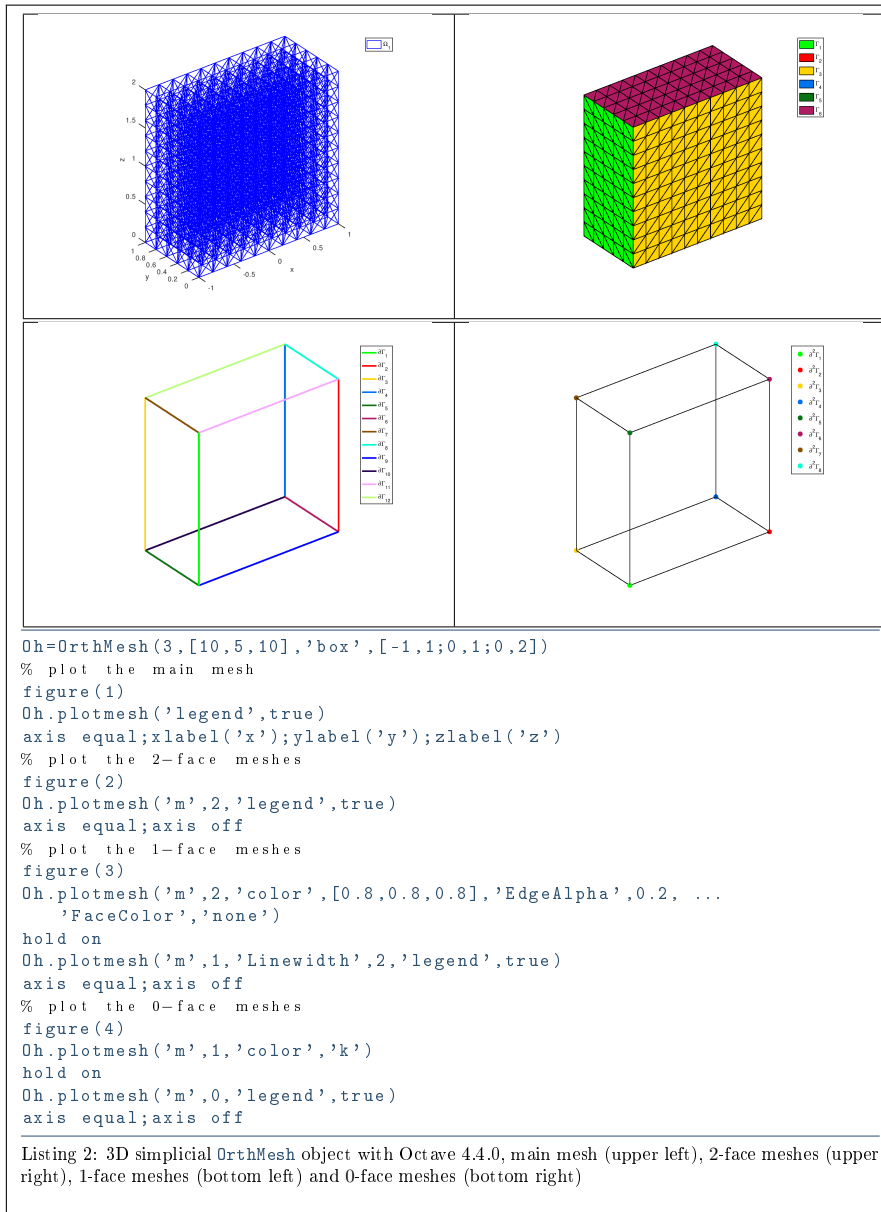
```
Oh=OrthMesh(2,10,'box',[-1,1;0,1])
```

So the `Oh` object is the tessellations of the orthotope $[-1, 1] \times [0, 1]$ with simplicial elements. In each direction $10 + 1 (= 11!)$ points are taken. So we have 11^2 vertices in this mesh. The main mesh and all the `m`-face meshes of the resulting object are plotted by using `plotmesh` method.



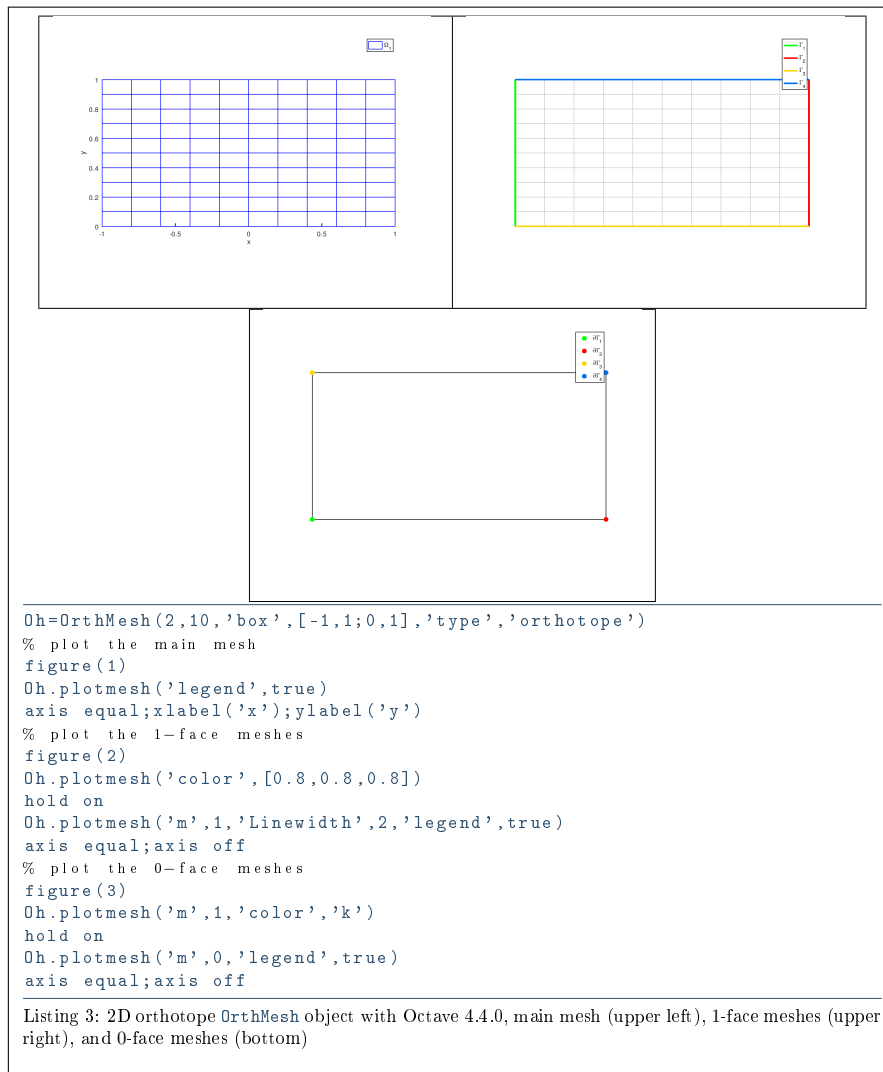
5.3 3d-orthotope meshing by simplices

In Listing 1, an `OrthMesh` object is built under Octave for the orthotope $[-1, 1] \times [0, 1] \times [0, 2]$ with simplicial elements and $N=[10, 5, 10]$. The main mesh and all the `m`-face meshes of the resulting object are plotted.



5.4 2d-orthotope meshing by orthotopes

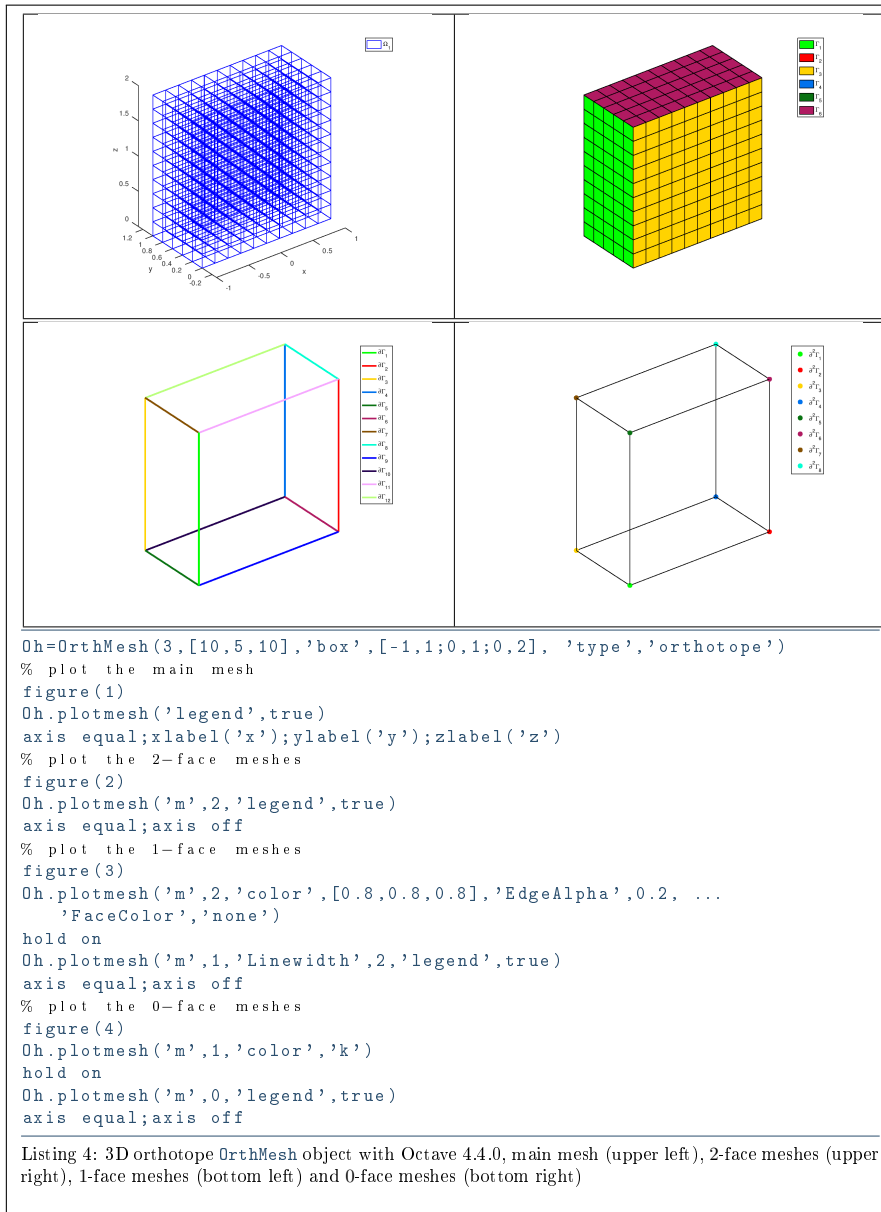
In Listing 1, an `OrthMesh` object is built under Octave for the orthotope $[-1, 1] \times [0, 1] \times [0, 2]$ with orthotope elements and $N=[10, 5, 10]$. The main mesh and all the m -face meshes of the resulting object are plotted.



5.5

3d-orthotope meshing by orthotopes

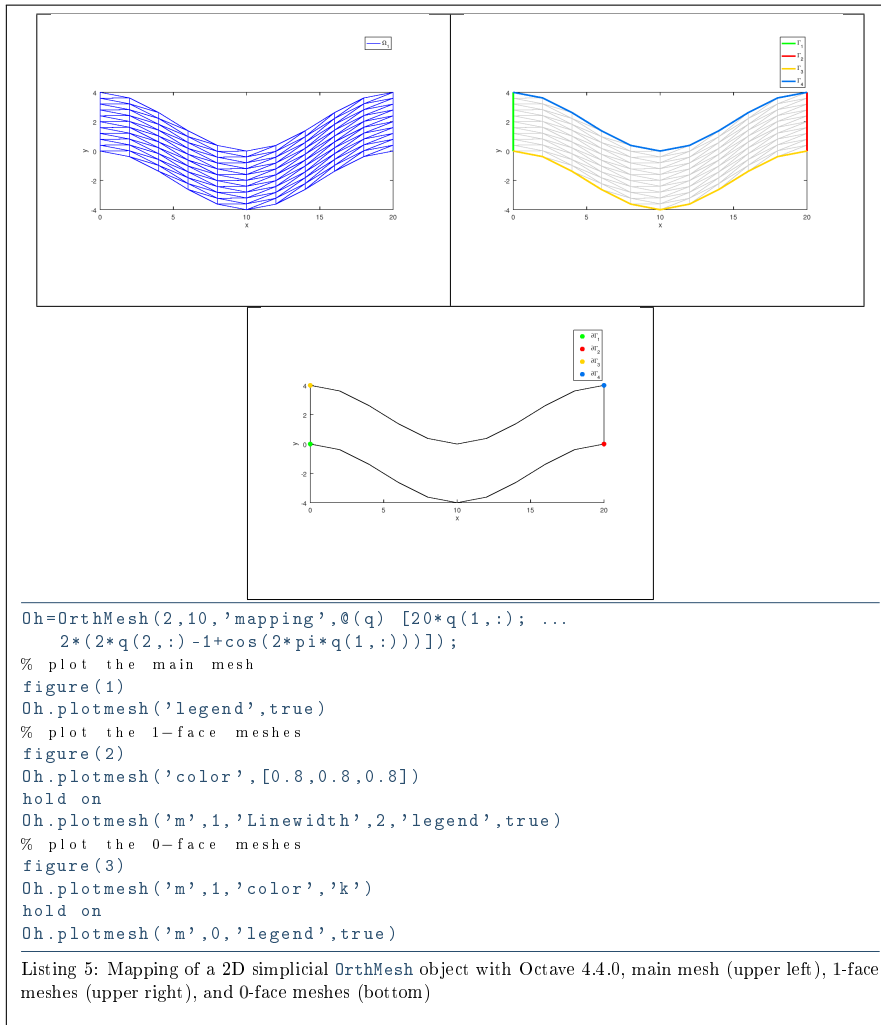
In Listing 1, an `OrthMesh` object is built under Octave for the orthotope $[-1, 1] \times [0, 1] \times [0, 2]$ with orthotope elements and $N=[10, 5, 10]$. The main mesh and all the `m`-face meshes of the resulting object are plotted.



5.6 Mapping of a 2d-orthotope meshing by simplices

For example, the following 2D geometrical transformation allows to deform the reference unit hypercube.

$$\begin{aligned}
 [0, 1] \times [0, 1] &\longrightarrow \mathbb{R}^2 \\
 \begin{pmatrix} x \\ y \end{pmatrix} &\longrightarrow F(x, y) = \begin{pmatrix} 20x \\ 2(2y - 1 + \cos(2\pi x)) \end{pmatrix}
 \end{aligned}$$

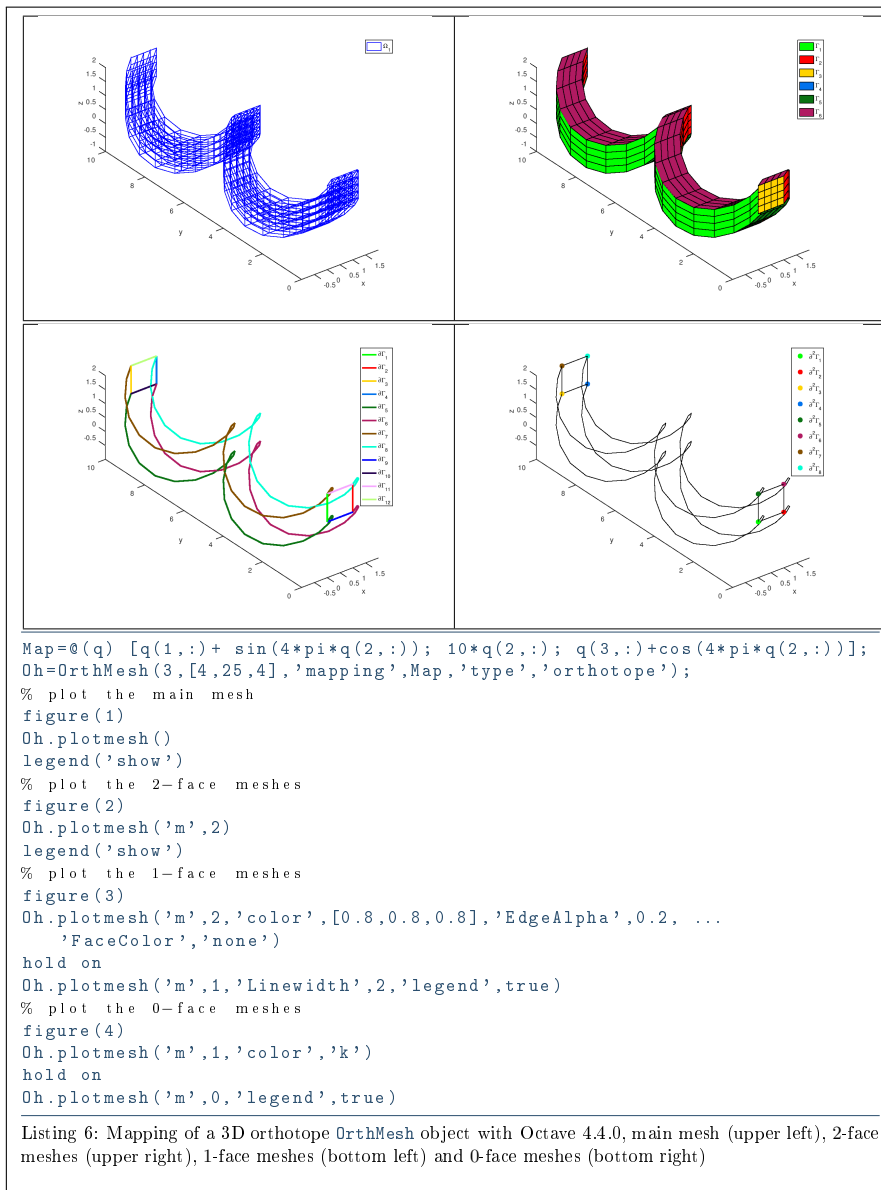


5.7 3d-orthotope meshing by orthotopes

For example, the following 3D geometrical transformation allows to deform the reference unit hypercube.

$$[0, 1] \times [0, 1] \times [0, 1] \longrightarrow \mathbb{R}^2$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \longrightarrow F(x, y, y) = \begin{pmatrix} x + \sin(4\pi y) \\ 10y \\ z + \cos(4\pi y) \end{pmatrix}$$



6 Benchmarking

6.1 fc_bench.bench01 function

The `fc_bench.bench01` function can be used to obtain computational times of the `OrthMesh` constructor.

Syntaxe

```
fc_bench.bench01(d, ctype, Box, LN)
```

Description

```
fc_bench.bench01(d, ctype, Box, LN)
```

displays computational times of the `OrthMesh` constructor as follows

```
ts=tic();Oh=OrthMesh(d,N,'box',Box,'type',ctype);tcpu=toc(ts);
```

for each `N` in `LN`.

6.2 Examples

Listing 7: : Computational times of `OrthMesh` constructor in dimension `d=3` (simplicial mesh)

```
fc_hypermesh.bench01(3,'simplicial',[-1 1;-1 1;-1 1],25:25:175)
```

Output

```
# BENCH in dimension 3 with simplicial mesh
#d: 3
#type: simplicial
#box: [-1 1; -1 1; -1 1]
#desc: N      nq      nme  time(s)
      25    17576    93750   0.282
      50   132651   750000   0.336
      75   438976  2531250   0.539
     100  1030301  6000000   0.863
     125  2000376 11718750   1.407
     150  3442951 20250000   2.271
     175  5451776 32156250   4.076
```

Listing 8: : Computational times of `OrthMesh` constructor in dimension `d=5` (orthotope mesh)

```
fc_hypermesh.bench01(5,'orthotope',[-1 1;-1 1;-1 1;-1 1;-1 ...
1], [5:5:25,27])
```

Output

```
# BENCH in dimension 5 with orthotope mesh
#d: 5
#type: orthotope
#box: [-1 1; -1 1; -1 1; -1 1; -1 1]
#desc: N      nq      nme  time(s)
      5     7776     3125   1.578
     10   161051   100000   1.682
     15  1048576   759375   2.168
     20  4084101  3200000   3.691
     25 11881376  9765625   8.416
     27 17210368 14348907  11.419
```

7 Mesh refinement

7.1 Non-conforming mesh refinement

In this part we propose a preliminary refinement of a regular d -grid. We want to generate a refinement of some cells of this regular grid with d -simplices and/or d -orthotopes: the mesh obtained is therefore non-conforming.

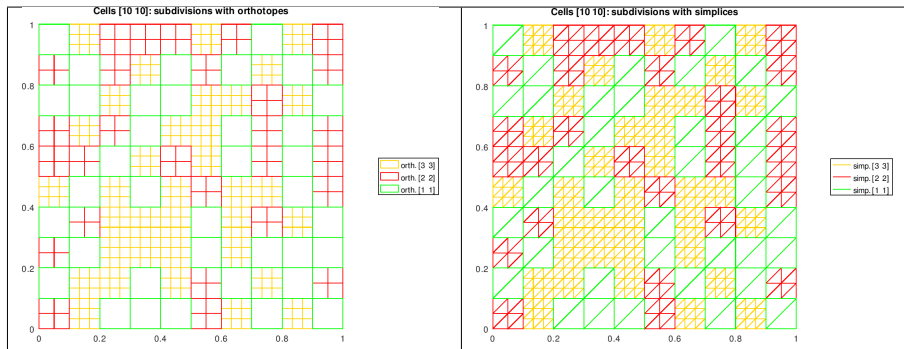


Figure 1: regular refinement of a 2D-grid with orthotopes (left) and simplices (right)

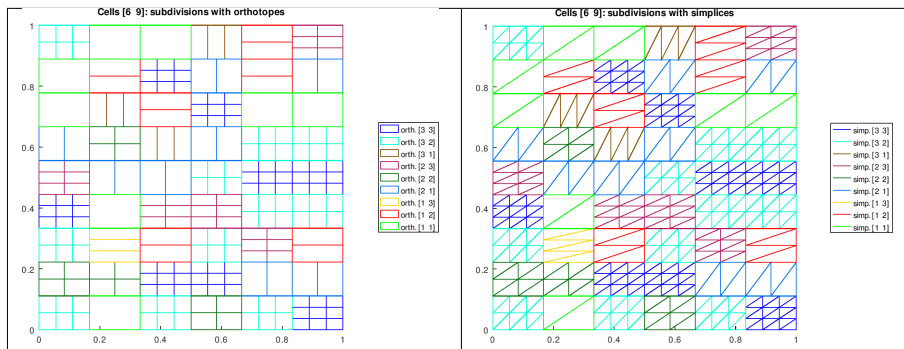


Figure 2: no regular refinement of a 2D-grid with orthotopes (left) and simplices (right)

In Figure 1, a 2D-grid with 10-by-10 cells is refined with regular subcells (i.e. same number of discretisation in each directory) made of simplices or orthotopes. For example a cell subdivided in 3-by-3 orthotopes is denoted by `orth[3 3]` and this same cell subdivided in 3-by-3 simplices (in fact $3 \times 3 \times 2$ simplices) is denoted by `simp[3 3]`. In Figure 2, a 2D-grid with 10-by-10 cells (left) and with 6-by-9 cells (right) is refined with subcells (not necessarily with same number of discretisation in each directory). In Figure 3, a 2D-grid with 10-by-10 cells (left) is refined with regular subcells made of orthotopes or simplices. On the right, a 2D-grid with 6-by-9 cells is refined with not necessarily regular subcells made of orthotopes or simplices refined with subcells respectively.

7.1.1 Refinement function

The refinement of an `OrthMesh` object whose elements are orthotopes is obtained by using the `fc_hypermesh.refinement.refine` Octavefunction.

```
scs = refine(Oh, ndiscells)
scs = refine(Oh, ndiscells, type)
```

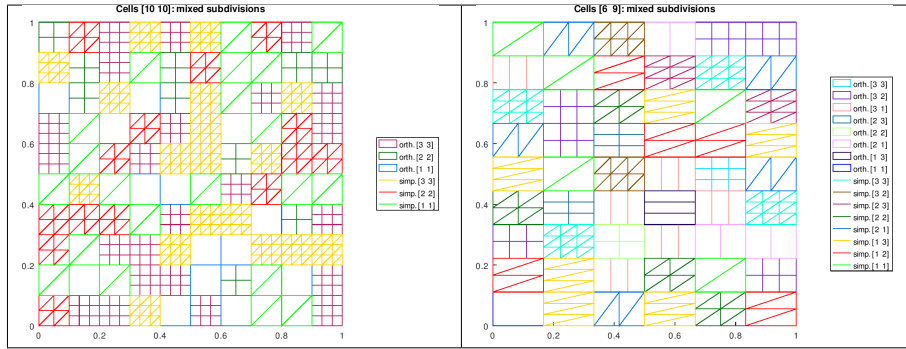


Figure 3: Regular (left) and not regular (right) refinement of a 2D-grid with orthotopes and simplices

```
scs = refine(Oh, ndiscells, types)
```

This function returns cells array of structures. Each array entry contains all cells refined with a `type` of element (simplicial or orthotope) and selected numbers of discretisation in each axis direction. Each structure (called *subcells structure*) has the fields

- `d` : space dimension
- `type` : type of element: 'simplicial' or 'orthotope'
- `n` : 1-by-`d` array corresponding to selected numbers of discretisation in each axis direction
- `nq` : number of vertices
- `nme` : number of mesh elements
- `q` : `d`-by-`nq` vertices array
- `me` : `p`-by-`nme` connectivity array ($p = d + 1$ for simplices and $p = 2^d$ for orthotopes)
- `ncell` : contains all the number of the cells grid which are refined.

Description

```
scs = fc_hypermesh.refinement.refine(Oh, ndiscells)
```

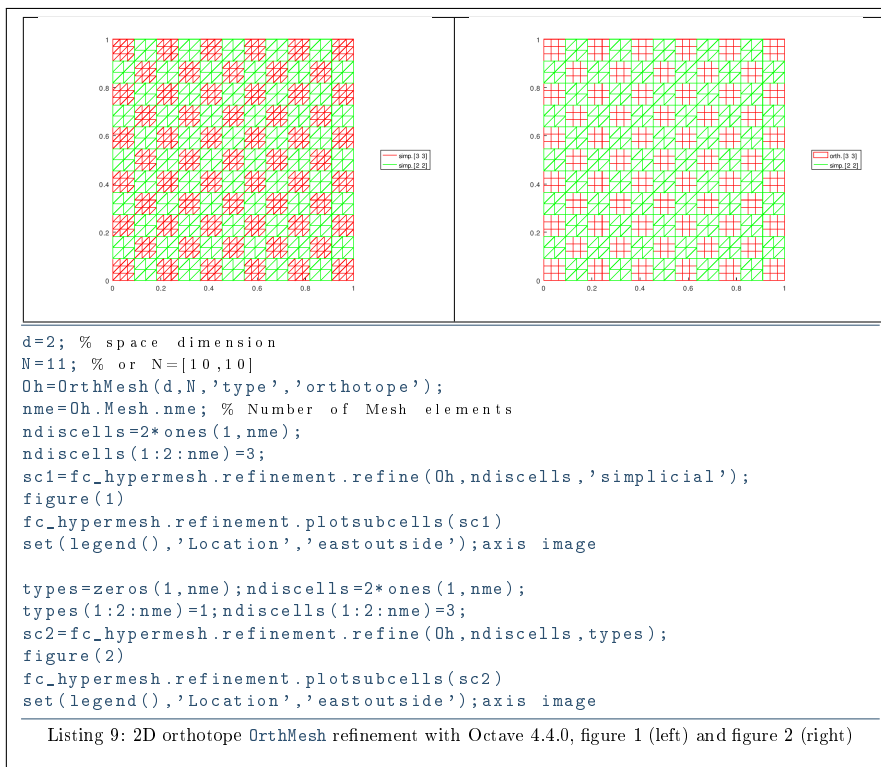
The first input `Oh` is an `OrthMesh` object whose elements are orthotopes. `ndiscells` is an 1-by-`Oh.Mesh.nme` array (for regular refinement) or a `d`-by-`Oh.Mesh.nme` array. `ndiscells(:,k)` are the numbers of discretisation in each axis direction. The output is a cells array of *subcells structure* where a all subcells `type` are 'orthotope'

```
scs = fc_hypermesh.refinement.refine(Oh,ndiscells,type)
```

Same as previous one if `type` is 'orthotope'. Otherwise `type` is 'simplicial' and all subcells `type` of the output are 'simplicial'

```
[scsimp,scorth] = fc_hypermesh.refinement.refine(Oh,ndiscells,types)
```

The input `types` is an 1-by-`Oh.Mesh.nme` array: if `types(k)` is equal to 0 then the k -th mesh element (cell) of the `OrthMesh` is refined with simplices otherwise with orthotopes.



More examples are provided by scripts:

- `fc_hypermesh.refinement.demo01`
- ...
- `fc_hypermesh.refinement.demo10`