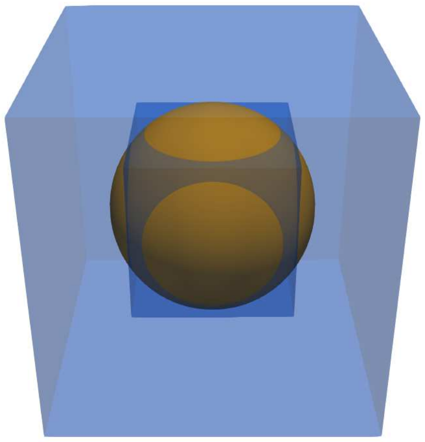
VOFTools



A package of routines with analytical and geometrical tools for 2D/3D VOF methods in general grids

User Manual

(Version 5, January 2020)

JOAQUÍN LÓPEZ

Universidad Politécnica de Cartagena

JULIO HERNÁNDEZ

Universidad Nacional de Educación a Distancia

Contents

List of Figures

List of Tables

1 | Introduction

VOFTools is a package of routines with analytical and geometrical tools for 2D/3D volume of fluid (VOF) methods in general grids. The VOFTools library includes efficient analytical and geometrical tools for area/volume computation, truncation operations that typically arise in VOF methods, area/volume conservation enforcement (VCE) to locate the interface in piecewise linear interface calculation (PLIC) reconstruction, liquid area/volume fraction initialization and computation of the distance from a given point to the reconstructed interface. Earlier versions of this library were presented in [5–8, 10]. The present version incorporates the capability to perform volume truncation, initialization and conservation enforcement operations over non-convex geometries without the need of costly convex decomposition techniques. The implementation details and additional performance analysis can be found in [9, 11]. More details can be found in [14].

The VOFTools routines are implemented in FORTRAN. To enable the routines to be used with C programs, the declarations in C of all the implemented routines are also included in the distributed software. The relevant arrays are pre-allocated for a user-specified number of faces (ns) and vertices (nv). By default, all the variables whose name starts with a letter between 'a' and 'h' or between 'o' and 'z' are considered as double-precision floating-point data and the rest are considered as regular integers. The files included in the supplied package of routines are the following:

- voftools.f: source code of the tools included in the VOFTools library.
- uservoftools.f: source code of user-defined routines and functions.
- mesh.f: routines with definitions of different cell geometries.
- dim.h: array dimensions for FORTRAN codes.
- dimc.h: array dimensions for C codes (the values for the array dimensions must be the same than that specified in the dim.h file).
- cvoftools.h: declaration of the VOFTools library routines to be used in C programs.
- cuservoftools.h: declaration of the user-defined routines and functions included in the uservoftools.f file to be used in C programs.
- cmesh.h: declaration of the routines included in mesh.f file to be used in C programs.
- test2d.f: 2D test program in FORTRAN.

- `test2d.c`: 2D test program in `C`.
- `test3d.f`: 3D test program in `FORTRAN`.
- `test3d.c`: 3D test program in `C`.
- `vofvardef`: text file with input data for the test programs.
- `Makefile.linux` and `Makefile.mac`: script examples in different platforms for the set of tasks to construct the `VOFTools` library and make executable files for test programs in `C` and `FORTRAN`.
- `user-manual-voftools-5.pdf`: this user manual in `pdf` format.
- `readme.txt`: information about the contents of the supplied package.
- `change.log`: record of all notable changes made to the routines.
- `COPYING`: copy of the GNU General Public License, Version 3.

2 | Installation

To install the VOFTools library and execute the test programs supplied in FORTRAN and C, perform the following steps:

1. Decompress the downloaded package in the working directory. For example, type

   ```
   tar -zxvf voftools-5.tgz
   ```

2. Go to the VOFTools directory. For example, type

   ```
   cd voftools-5
   ```

3. Depending on the available platform, edit one of the script files, Makefile.linux or Makefile.mac (in the following, it will be considered that the selected script file is Makefile.linux), and choose the compiler (for example, in the Makefile.linux file, the user can set COMPILER = 1, 2 or 3 for GNU, Intel or PGI compilers, respectively). Windows users could use the GNU make utility and adapt one of the supplied Makefile scripts, or use Microsoft Visual Studio or any other similar IDE to compile the source code. Table 2.1 shows some tested compilers and platforms. For other situations, the users can construct their own script by adapting one of the Makefile scripts provided with the supplied package. The user can introduce other compilers by setting variables CC, F77 and LIBS in the script file.

   Table 2.1: Some tested compilers and platforms.

Operating system	Compiler		
Linux	GNU	Intel	PGI
macOS	GNU	Intel	-
Windows	GNU	Intel	PGI

4. Build the VOFTools library (libvoftools.a):

   ```
   make -f Makefile.linux
   ```

5. To compile the test programs, type

```
make c_2d -f Makefile.linux
```

for the 2D version (`test2d_c`) in `C`,

```
make c_3d -f Makefile.linux
```

for the 3D version (`test3d_c`) in `C`,

```
make fortran_2d -f Makefile.linux
```

for the 2D version (`test2d_f`) in `FORTRAN`, or

```
make fortran_3d -f Makefile.linux
```

for the 3D version (`test3d_f`) in `FORTRAN`.

Alternatively, to build the `VOFTools` library and compile all the test programs, type

```
make all -f Makefile.linux
```

Optionally, the built `VOFTools` library can be moved to a search path (for example, `/usr/lib/`)

```
sudo cp libvoftools.a /usr/lib
sudo chmod 777 /usr/lib/libvoftools.a
```

and any test program (for example, `testprogram.f`) can be compiled as

```
ifort -o testprogram testprogram.f -lvoftools
```

6. Edit the `vofvardef` file and input the appropriate values for the parameters corresponding to the considered test case. Information about the values of the different input parameters is included in the source files of the test programs (`test2d.c`, `test2d.f`, `test3d.c` and `test3d.f`) and in Section 5. For example, the `vofvardef` file listed below corresponds to a 3D case in which a half-space with unit-length vector (`xnc` $= 0$; `ync` $= -1$; `znc` $= 0$) normal to the interface intersects a cubic cell (`icelltype` $= 11$) with a truncated volume fraction `f` $= 0.5$, the distance from a point P (`xp` $= 0$; `yp` $= 0$; `zp` $= 0$) to the reconstructed PLIC interface is computed (note that, at this moment, the current version of the package does not include the extension to non-convex geometries of the distance computation routines) and the volume of a spherical material body (`ishape` $= 11$) contained in the cell is estimated using the proposed initialization procedure with `nc`=10 subdivisions along each coordinate direction and a tolerance `tol`=10:

```
Cell geometry, ICELLTYPE:
11
Material body shape, ISHAPE:
11
Material volume/area fraction, F:
0.5
X coordinate of the unit-length normal vector of the interface
   plane, XNC:
0.0
Y coordinate of the unit-length normal vector of the interface
   plane, YNC:
```

```
-1.0
Z coordinate of the unit-length normal vector of the interface
   plane, ZNC:
0.0            !Ignore for 2D test programs
X coordinate of point P from which the distance is calculated, XP:
0.0
Y coordinate of point P from which the distance is calculated, YP:
0.0
Z coordinate of point P from which the distance is calculated, ZP:
0.0            !Ignore for 2D test programs
Subdivision number in the volume fraction cell initialization, NC:
10
Tolerance for the initialization procedure, TOL:
10.0
```

7. Execute the test program/s. E.g., for the 3D FORTRAN test program, type

```
 ./test3d_f
```

If the user needs to modify the values of parameters `ns` or `nv` in the `dim.h` file for the FORTRAN version of the code or in the `dimc.h`[1] file for the C version of the code, for example in order to use cells with a number of faces or vertices higher than that initially specified in the above files, the `VOFTools` library must be recompiled. To recompile the `VOFTools` library in the same directory, type

```
make clean -f Makefile.linux
```

first and then proceed as indicated above.

[1]As mentioned above, caution must be taken by using the same values for the `ns` and `nv` parameters in the `dimc.h` file than those used in the `dim.h` file during the compilation of the `VOFTools` library.

3 | Routines description and usage

In the routines described below, the structure of a given polyhedron is arranged using the following parameters:

`ipv`:	array of dimensions `(ns,nv)`, which stores the numbers of the polyhedron vertices
`nipv`:	array of dimension `ns`, which stores the total number of vertices of each face boundary
`ntp`:	last vertex number
`nts`:	total number of face boundaries
`ntv`:	total number of vertices (note that, if the polyhedron has not previously been truncated, then `ntp=ntv`)
`vertp`:	array of dimensions (`nv`,3), which stores the x, y, z-coordinates of the polyhedron vertices
`xns, yns, zns`:	arrays of dimension `ns`, which store the x, y, z-components of the unit-length vectors normal to the faces pointing out the polyhedron

In 2D, parameters `ipv(nv)`, `ntp`, `ntv` and `vertp`(`nv`,2) define the structure of a given polygon in a similar way.

In the supplied `mesh.f` file, different convex and non-convex polyhedra and polygons (Table **3.1** summarizes the geometries considered in the test programs of Section 5) can be set by calling the indicated routines, which return the parameters that define the structure of polyhedra or polygons, as described above. Figs. **3.1** and **3.2** depict the corresponding cells with help of the `polout2d` (Section 3.20) and `polout3d` (Section 3.10) routines and `Gnuplot` [12] and `ParaView` [3] programs, respectively (the name of the routine used to define the geometry of each cell is also included). As an example, the calling conventions for a cubic cell in `FORTRAN` and `C` are, respectively,

```
      call cubicmesh(ipv,nipv,ntp,nts,ntv,vertp,xns,yns,zns)
```

```
cubicmesh_(ipv,nipv,&ntp,&nts,&ntv,vertp,xns,yns,zns);
```

and, for a square cell,

```
      call squaremesh(ipv,ntp,ntv,vertp)
```

Table 3.1: Cell geometries considered in the test programs of Section 5 and included in the `mesh.f` file.

Routine name	**Cell geometry**
	3D
`cubicmesh`	Cube
`hexahemesh`	Irregular hexahedron
`tetramesh`	Tetrahedron
`dodecamesh`	Dodecahedron
`icosamesh`	Icosahedron
`complexmesh`	Complex polyhedron with 32 vertices and 18 faces
`ncpentapyramid`	Non-convex pentagonal pyramid
`nccubicpyramid`	Non-convex cell obtained by substracting a pyramid to the cubic cell
`ncscubicmesh`	Stellated cube
`nchexahemesh`	Non-convex hexahedron
`ncdodecamesh`	Stellated dodecahedron
`ncicosamesh`	Stellated icosahedron
`nchollowedcube`	Hollowed cube
`drilledcube`	Drilled cube
`zigzagmesh`	Zig zag prism
`voftoolslogo`	VOFTools logo
	2D
`squaremesh`	Square
`hexagomesh`	Regular hexagon
`trianglemesh`	Irregular triangle
`quadranglemesh`	Irregular quadrangle
`pentagonmesh`	Irregular pentagon
`hexagonmesh`	Irregular hexagon
`ncquadranglemesh`	Non-convex quadrangle
`ncpentagonmesh`	Non-convex pentagon
`nchexagonmesh`	Non-convex hexagon
`ncshexagonmesh`	Stellated hexagon
`nchollowedsquare`	Hollowed square
`ncmultisquare`	Non-convex multi-square cell

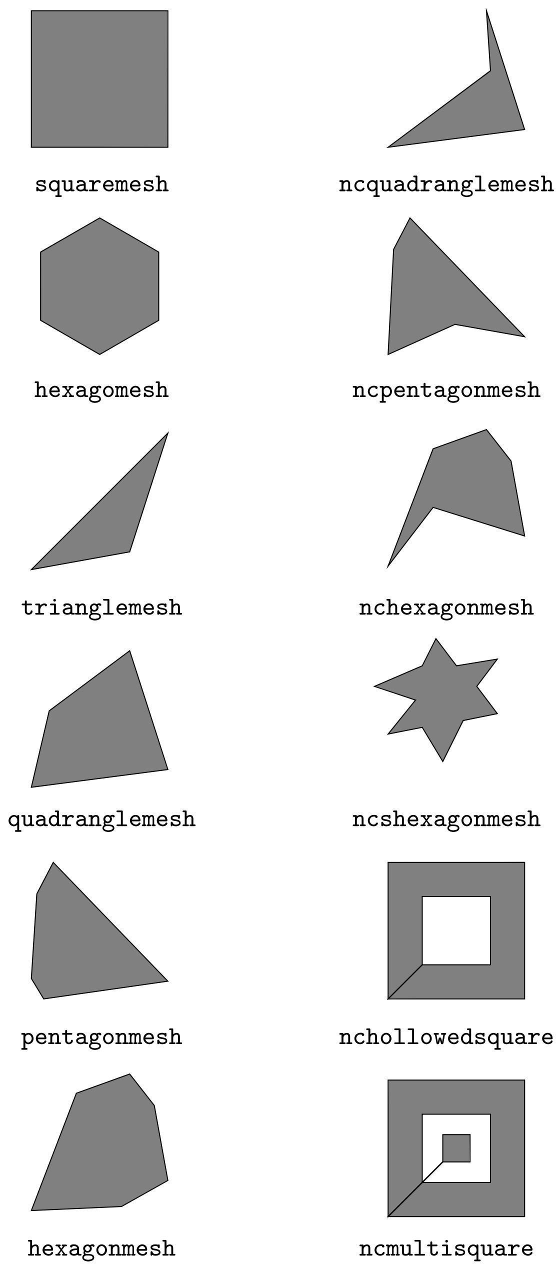


Figure 3.1: Cell geometries for the 2D cases of Table 5.1.

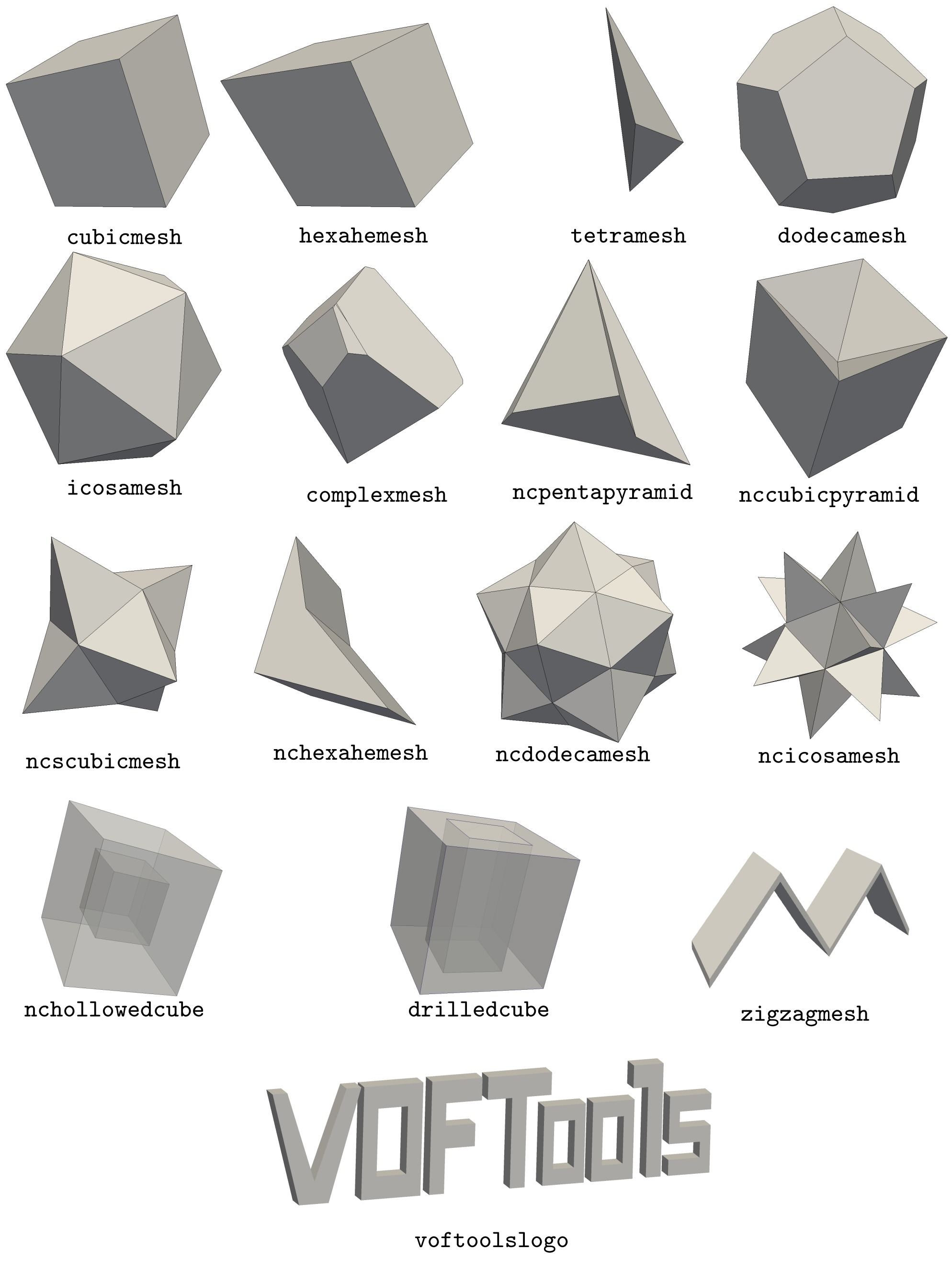


Figure 3.2: Cell geometries for the 3D cases of Table 5.1.

```
squaremesh_(ipv,&ntp,&ntv,vertp);
```

In the following, the input and output arguments and the calling convention of each routine implemented in the VOFTools library (voftools.f) are described in detail. In some of the routines, an additional numerical character has been added to the name of some of the parameters defined at the beginning of this section (for example, ipv1) to denote a value previous to or obtained from a certain operation.

3.1 enforv3d

Solves the VCE problem in 3D to locate the PLIC interface in order to cut off a certain liquid volume from the polyhedrical cell, and is invoked in FORTRAN and C as follows:

```
      call enforv3d(c,ipv,nipv,ntp,nts,ntv,v,vt,vertp,xnc,
     -                xns,ync,yns,znc,zns)
```

```
enforv3d_(&c,ipv,nipv,&ntp,&nts,&ntv,&v,&vt,vertp,&xnc,xns,
&ync,yns,&znc,zns);
```

where the arguments are

- On entry:

ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns:	parameters that define the structure of the polyhedron
v:	liquid volume
vt:	total volume of the polyhedron
xnc, ync, znc:	components of the unit-length vector normal to the interfacial plane pointing to the liquid

- On return:

c:	solution of the problem

3.2 enforv3dsz

Solves the local volume conservation enforcement problem for rectangular parallelepipedic cells, such as that defined in the cubicmesh routine, using the more efficient analytical method of Scardovelli and Zaleski [13], which was proposed for use specifically with this type of cell:

```
      call enforv3dsz(c,dx,dy,dz,v,vertp,xnc,ync,znc)
```

```
enforv3dsz_(&c,&dx,&dy,&dz,&v,vertp,&xnc,&ync,&znc);
```

Arguments:

- On entry:

`dx, dy, dz`:	cell dimensions along the coordinate axis x, y, z
`vertp`:	array of vertex coordinates
`v`:	liquid volume
`xnc, ync, znc`:	components of the unit-length vector normal to the interfacial plane pointing to the liquid

- On return:

`c`:	solution of the problem

3.3 newpol3d

Rearranges the vertices of a truncated polyhedron:

```
      call newpol3d(ia,ipia0,ipia1,ipv,iscut,nipv,ntp,nts,
     -              ntv,xnc,xns,ync,yns,znc,zns)
```

```
newpol3d_(ia,ipia0,ipia1,ipv,iscut,nipv,&ntp,&nts,&ntv,
&xnc,xns,&ync,yns,&znc,zns);
```

Arguments:

- On entry:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters that define the structure of the original polyhedron
`xnc, ync, znc`:	components of the unit-length vector normal to the interface plane
`ia`:	array of dimension `nv` that, for each original polyhedron vertex, stores a value of 0 if the normal to the interface plane points out from the vertex and 1 otherwise

- On return:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters that define the structure of the truncated polyhedron (note that the parameters of the original polyhedron are replaced by those of the truncated polyhedron, as also occurs in the routine `inte3d` described below)
`ipia0`, `ipia1`:	arrays of dimension `nv` that store the numbers of the original polyhedron vertices in which `ia`=0 and `ia`=1, respectively, and which are located on the edge containing the intersection point
`iscut`:	array of dimension `ns`, whose elements are equal to 1 if the face boundary is truncated and 0 otherwise

3.4 `inte3d`

Performs the intersection between a generic polyhedron and a plane:

```
      call inte3d(c,icontn,icontp,ipv,nipv,ntp,nts,ntv,vertp,
     - xnc,xns,ync,yns,znc,zns)
```

```
inte3d_(&c,&icontn,&icontp,ipv,nipv,&ntp,&nts,&ntv,vertp,
&xnc,xns,&ync,yns,&znc,zns);
```

Arguments:

- On entry:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the original polyhedron
`xnc, ync, znc`:	components of the unit-length vector normal to the interface plane
`c`:	constant of the truncating plane

- On return:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the truncated polyhedron
`icontn`:	total number of vertices of the original polyhedron outside the truncated region
`icontp`:	total number of vertices of the original polyhedron that remain in the truncated region

3.5 toolv3d

Computes the volume of a polyhedron:

```
      call toolv3d(ipv,nipv,nts,vertp,vol,xns,yns,zns)
```

```
toolv3d_(ipv,nipv,&nts,vertp,&vol,xns,yns,zns);
```

Arguments:

- On entry:

`ipv, nipv, nts, vertp, xns, yns, zns`:	polyhedron parameters

- On return:

`vol`:	polyhedron volume

3.6 cppol3d

Makes a copy of the structure of a polyhedron:

```
      call cppol3d(cs,cs0,ipv,ipv0,nipv,nipv0,ntp,ntp0,nts,
     - nts0,ntv,ntv0,vertp,vertp0,xns,xns0,yns,yns0,zns,zns0)
```

```
cppol3d_(cs,cs0,ipv,ipv0,nipv,nipv0,&ntp,&ntp0,&nts,&nts0,
&ntv,&ntv0,vertp,vertp0,xns,xns0,yns,yns0,zns,zns0);
```

Arguments:

- On entry:

`ipv0, nipv0, ntp0, nts0, ntv0, vertp0, xns0, yns0, zns0`:	parameters of the original polyhedron
`cs0`:	array of dimension `ns` that stores the constants of the planes containing the faces of the original polyhedron

- On return:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the copied polyhedron
`cs`:	copy of `cs0`

3.7 restore3d

Restores the structure of a polyhedron by renumbering consecutively the faces and values of the vertex number of the polyhedron:

```
      call restore3d(cs,ipv,nipv,ntp,nts,ntv,vertp,xns,yns,
     - zns)
```

```
restore3d_(cs,ipv,nipv,&ntp,&nts,&ntv,vertp,xns,yns,zns);
```

Arguments:

- On entry:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the original polyhedron
`cs`:	array of dimension `ns` that stores the constants of the planes containing the faces of the original polyhedron

- On return:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the restored polyhedron
`cs`:	array of dimension `ns` that stores the constants of the planes containing the faces of the restored polyhedron

Note that the parameters of the original polyhedron are replaced by those of the restored polyhedron.

It should be mentioned that if a polyhedron is previously intersected by a plane, this routine must be called before solving the VCE problem over this previously-truncated polyhedron (the routine `enforv3d` requires that all the polyhedral vertices be consecutively numbered). Although this situation is not common in the context of a PLIC-VOF method, the `restore3d` routine is supplied for any other situations that might require it.

3.8 dist3d

Computes the distance from a point P to a general convex polygon in 3D:

```
      call dist3d(d,n,x,y,z,xp,yp,zp)
```

```
dist3d_(&d,&n,x,y,z,&xp,&yp,&zp);
```

Arguments:

- On entry:

`n`:	number of vertices of the polygon
`x, y, z`:	arrays of dimension `nv` that store the coordinates of the polygon vertices
`xp, yp, zp`:	coordinates of point P

- On return:

`d`:	distance from point P to the polygon

3.9 initf3d

Computes the fraction of the material volume contained in a cell:

```
      call initf3d(func3d,ipv,nc,nipv,ntp,nts,ntv,tol,vertp,vf,
     -  xns,yns,zns)
```

```
initf3d_(func3d_,ipv,&nc,nipv,&ntp,&nts,&ntv,&tol,vertp,&vf,
xns,yns,zns);
```

Arguments:

- On entry:

`func3d`:	user-defined implicit function, included in the `uservoftools.f` file, that defines the material body shape
`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the original polyhedron
`nc`:	number of divisions along each coordinate axis of the superimposed cell box
`tol`:	prescribed positive tolerance for the distance to the interface of the material body

- On return:

`vf`:	material volume fraction

3.10 polout3d

Exports the geometry of the polyhedron to a file in VTK (visualization toolkit [4]) format which can be plotted by using, for example, the ParaView program [3][1]:

[1]For a polyhedron with non-convex polygonal faces, we recommend to use the "Triangulate" filter available in the ParaView program to triangulate the polygonal faces and avoid rendering issues of non-convex polygons.

```
        call polout3d(ifile,ipv,nipv,ntp,nts,vertp)
```

```
polout3d_(&ifile,ipv,nipv,&ntp,&nts,vertp);
```

Arguments:

- On entry:

`ifile`:	number # used to name the external VTK format file
`ipv, nipv, ntp, nts, vertp`:	parameters of the polyhedron to be plotted

- On return:

`pol#.vtk`:	VTK-format file

3.11 enforv2d

Solves the VCE problem in 2D to locate the PLIC interface in order to cut off a certain liquid area from the polygonal cell:

```
       call enforv2d(c,ipv,ntp,ntv,v,vt,vertp,xnc,ync)
```

```
enforv2d_(&c,ipv,&ntp,&ntv,&v,&vt,vertp,&xnc,&ync);
```

Arguments:

- On entry:

`ipv, ntp, ntv, vertp`:	parameters of the polygonal cell
`v`:	liquid area
`vt`:	total area of the polygonal cell
`xnc, ync`:	components of the unit-length vector normal to the interfacial line pointing to the liquid

- On return:

`c`:	solution of the problem

3.12 enforv2dsz

Solves the VCE problem for rectangular cells, such as that defined in the `squaremesh` routine, using the more efficient analytical method of Scardovelli and Zaleski [13]:

```
       call enforv2dsz(c,dx,dy,v,vertp,xnc,ync)
```

```
enforv2dsz_(&c,&dx,&dy,&v,vertp,&xnc,&ync);
```

Arguments:

- On entry:

`dx, dy`:	cell dimensions
`vertp`:	array of vertex coordinates of the cell
`v`:	liquid volume
`xnc, ync`:	components of the unit-length vector normal to the interfacial line pointing to the liquid

- On return:

`c`:	solution of the problem

3.13 newpol2d

Rearranges the vertices of a truncated polygon:

```
      call newpol2d(ia,ipia0,ipia1,ipv,ntp,ntv,vertp,xncut,
     - yncut)
```

```
newpol2d_(ia,ipia0,ipia1,ipv,&ntp,&ntv,vertp,xncut,yncut);
```

Arguments:

- On entry:

`ipv, ntp, ntv, vertp`:	parameters of the original polygon
`xnc, ync`:	components of the unit-length vector normal to the interfacial line
`ia`:	array of dimensions `nv` that stores a value of 0 if the normal to the interface plane points out from the vertex and 1 otherwise

- On return:

`ipv, ntp, ntv, vertp`:	parameters of the truncated polygon (note that the parameters of the original polygon are replaced by those of the truncated polygon, as also occurs in the routine `inte2d` described below)
`xncut, yncut`:	two-element arrays that store the components of the unit-length vectors normal to the two edges cut by the interface line
`ipia0`, `ipia1`:	arrays of dimension `nv` that store the numbers of the original polygon vertices for which `ia`=0 and `ia`=1, respectively, and which are located on the edge containing the intersection point

3.14 inte2d

Performs the intersection between a generic polygon and a line:

```
      call inte2d(c,icontn,icontp,ipv,ntp,ntv,vertp,xnc,ync)
```

```
inte2d_(&c,&icontn,&icontp,ipv,&ntp,&ntv,vertp,&xnc,&ync);
```

Arguments:

- On entry:

`ipv, ntp, ntv, vertp`:	parameters of the original polygon
`xnc, ync`:	components of the unit-length vector normal to the interfacial line
`c`:	constant of the truncating line

- On return:

`ipv, ntp, ntv, vertp`:	parameters corresponding of the truncated polygon
`icontn`, `icontp`:	total number of vertices of the original polygon that are outside and inside the truncated region, respectively

3.15 toolv2d

Computes the area of a generic polygon:

```
      call toolv2d(ipv,ntv,vertp,vol)
```

```
toolv2d_(ipv,&ntv,vertp,&vol);
```

Arguments:

- On entry:

`ipv, ntv, vertp`:	parameters of the polygon

- On return:

`vol`:	area of the polygon

3.16 cppol2d

Copies the structure of a polygon:

```
      call cppol2d(ipv0,ipv,ntp0,ntp,ntv0,ntv,vertp0,vertp)
```

```
cppol2d_(ipv0,ipv,&ntp0,&ntp,&ntv0,&ntv,vertp0,vertp);
```

Arguments:

- On entry:

`ipv0, ntp0, ntv0, vertp0`:	parameters of the original polygon

- On return:

`ipv, ntp, ntv, vertp`:	parameters of the copied polygon

3.17 restore2d

Restores the structure of a polygon by renumbering consecutively the values of the vertex number of the polygon:

```
      call restore2d(ipv,ntp,ntv,vertp)
```

```
restore2d_(ipv,&ntp,&ntv,vertp);
```

Arguments:

- On entry:

`ipv, ntp, ntv, vertp`:	parameters of the original polygon

- On return:

`ipv, ntp, ntv, vertp`:	parameters of the restored polygon

Note that the parameters of the original polygon are replaced by those of the restored polygon.

A comment similar to that mentioned at the end of Section 3.7 about the usefulness of the `restore3d` routine can also be applied for this 2D routine.

3.18 dist2d

Computes the distance from point P to a segment in 2D:

```
      call dist2d(d,x,y,xp,yp)
```

```
dist2d_(&d,x,y,&xp,&yp);
```

Arguments:

- On entry:

`x, y`:	two-element arrays that store the coordinates of the segment vertices
`xp, yp`:	coordinates of point P

- On return:

d:	distance from point P to the segment

3.19 initf2d

Computes the fraction of the material area contained in a cell:

```
      call initf2d(func2d,ipv,nc,ntp,ntv,tol,vertp,vf)
```

```
initf2d_(func2d_,ipv,&nc,&ntp,&ntv,&tol,vertp,&vf);
```

Arguments:

- On entry:

`func2d`:	user-defined implicit function, included in the `uservoftools.f` file, that defines the material body shape
`ipv, ntp, ntv, vertp`:	parameters of the original polygon
`nc`:	number of divisions along each coordinate axis of the superimposed cell box
`tol`:	prescribed positive tolerance for the distance to the interface of the material body

- On return:

`vf`:	material area fraction

3.20 polout2d

Exports the geometry of the polygon to a file in a two column format which can be plotted using, for example, the Gnuplot program [12]:

```
       call polout2d(ifile,ipv,ntv,vertp)
```

```
polout2d_(&ifile,ipv,&ntv,vertp);
```

Arguments:

- On entry:

ifile:	number # used to name the external file
ipv, ntv, vertp:	parameters of the polygon to be plotted

- On return:

pol#.out:	two column format file

4 | Performance analysis

An analysis of the performance of the different routines is presented in this section. The codes were compiled using the `GNU FORTRAN` (`gfortran`) compiler with the `-O0` option to avoid automatic vectorization and run on a Linux platform with a 2.9 GHz Intel Xeon E3-1535Mv5 processor. Further assessments can be found in References [5–11].

4.1 Volume truncation

Firstly, the non-convex pentagonal pyramid shown in Fig. 3.2, which is defined by the supplied routine `ncpentapyramid` (the position coordinates of its vertices are shown in Fig. 4.1), is considered. Table 4.1 presents the arrangement of the vertices of this polyhedron. Fig. 4.2

Table 4.1: Array `ipv0` of vertex number, i_p, assigned to every vertex index i of face boundary j of the polyhedron of Fig. 4.1.

Vertex index	Face boundary j					
i	1	2	3	4	5	6
1	1	1	2	3	4	5
2	5	2	3	4	5	1
3	4	6	6	6	6	6
4	3	–	–	–	–	–
5	2	–	–	–	–	–

shows the truncation operation, performed by calling the routine `inte3d`, over the considered polyhedron and the plane $\mathscr{P}$ with interface orientation $\boldsymbol{n}$ given by `xnc` $= 0$, `ync` $= -1$ and `znc` $= 0$, and position given by `c` $= 0.18$. The vertices with `ia` array values equal to 1 and 0 are represented in Fig. 4.2 with black and white circles, respectively. As described below, the resulting truncated region Ω_T (thick lines in Fig. 4.2) will be defined by the vertices where the `ia` value is equal to 1 and the points of intersection between $\mathscr{P}$ and the edges of Ω ($\times$ symbols in Fig. 4.2). For example, one of these edges is defined by the two adjacent vertices, $\boldsymbol{x}_5$ and $\boldsymbol{x}_4$, and the position vector of the intersection point results as $\boldsymbol{x}_7 = \boldsymbol{x}_4 - \frac{\phi_4}{\phi_5 - \phi_4}(\boldsymbol{x}_5 - \boldsymbol{x}_4)$. The truncation procedure begins by discarding the polyhedron face boundaries in which all the vertices have null value for the array `ia` (discarded face boundary 6 in Fig. 4.3) by setting the corresponding value of array `nipv1` to 0. Next, the new array `ipv1` of ordered vertex indices for

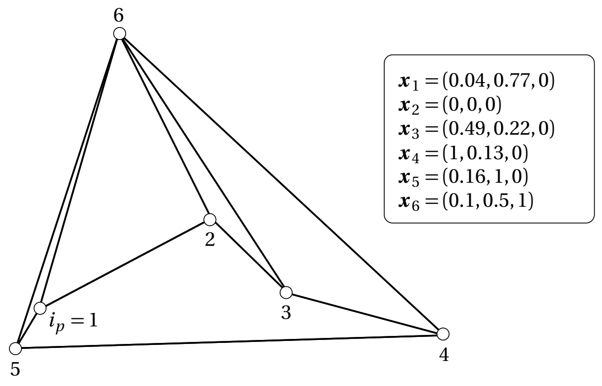

6
2
3
4
5
$i_p = 1$
$\boldsymbol{x}_1 = (0.04, 0.77, 0)$
$\boldsymbol{x}_2 = (0, 0, 0)$
$\boldsymbol{x}_3 = (0.49, 0.22, 0)$
$\boldsymbol{x}_4 = (1, 0.13, 0)$
$\boldsymbol{x}_5 = (0.16, 1, 0)$
$\boldsymbol{x}_6 = (0.1, 0.5, 1)$

Figure 4.1: Position coordinates of the vertices of the non-convex pentagonal pyramid shown in Fig. 3.2.

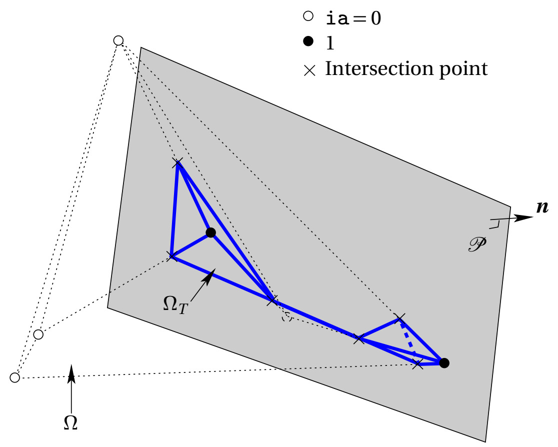


Figure 4.2: Truncation of the polyhedron Ω of Fig. 4.1 (dotted lines) by a plane $\mathscr{P}$ defined by `xnc` $= 0$, `ync` $= -1$, `znc` $= 0$ and `c` $= 0.18$ (highlighted in gray). The truncated polyhedron Ω_T is represented with blue thick lines.

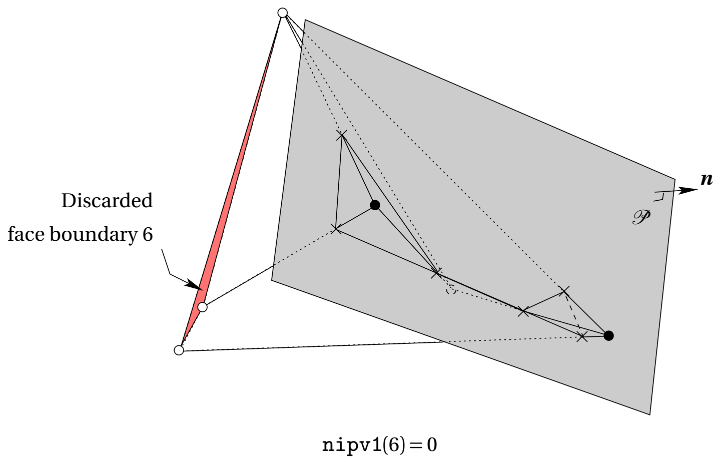


Figure 4.3: Face discarding.

Table 4.2: Array `ipv1` of the index number, i_p, assigned to every vertex index i of face boundary j of the polyhedron, Ω_T, resulting from the truncation operation of Fig. 4.2.

Vertex index	Face boundary j						
i	1	2	3	4	5	6	7
1	7	10	2	8	4	–	7
2	4	2	9	4	7	–	10
3	8	11	11	12	12	–	11
4	9	–	–	–	–	–	9
5	2	–	–	–	–	–	8
6	10	–	–	–	–	–	12

every truncated face boundary (face boundaries 1, 2, 3, 4 and 5 in Fig. 4.4) and every new on-$\mathscr{P}$ face boundary of the truncated polyhedron (face boundary 7 in Fig. 4.5) is constructed following, respectively, the clipping and capping procedures described in [9]. Fig. 4.5 illustrates the sequence of the on-$\mathscr{P}$ face boundary construction. Intersection vertices are represented with $\times$, truncated faces are gray filled and the polygonal boundary of the constructed on-$\mathscr{P}$ face is highlighted at the end of the sequence in blue thick lines. The edge of a truncated face boundary containing a "key vertex" [9] and the edge of that face boundary through which the edges of the on-$\mathscr{P}$ face boundaries are sequentially constructed are also highlighted in each picture as $\circ\!\!\longrightarrow\!\!\bullet$ and $\bullet\!\!\longrightarrow\!\!\circ$, respectively. In this example, the resulting on-$\mathscr{P}$ face boundary is a weakly-simple polygon with two overlapping edges (7-10 and 9-8). Note that in this way, two disjoint co-planar regions (highlighted in blue in Fig. 4.6) are represented with a single polygonal chain as $\texttt{ipv1}(7,i) = (7,10,11,9,8,12)$ for $i = 1$ to $\texttt{nipv1}(7) = 6$ (Fig. 4.6 shows this arrangement). Table 4.2 shows the vertex arrangement of the polyhedron, Ω_T, resulting from the truncation operation of Fig. 4.2.

A case in which the application of the capping procedure produces multiple new on-$\mathscr{P}$

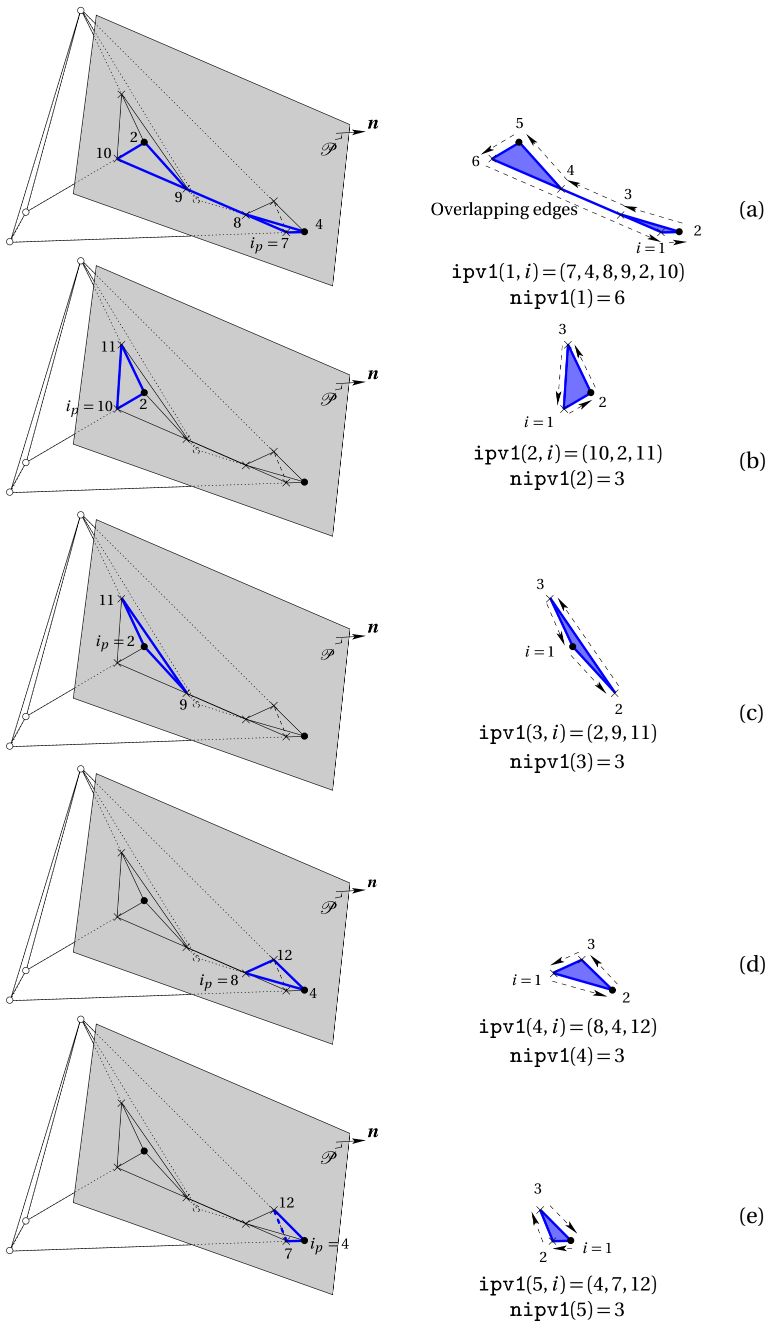


Figure 4.4: Clipping procedure for the face boundaries (a) 1, (b) 2,(c) 3, (d) 4 and (e) 5 of Fig. 4.2. Polygon truncation on the left and truncated polygons, filled with blue (arrows indicate vertex order), on the right.

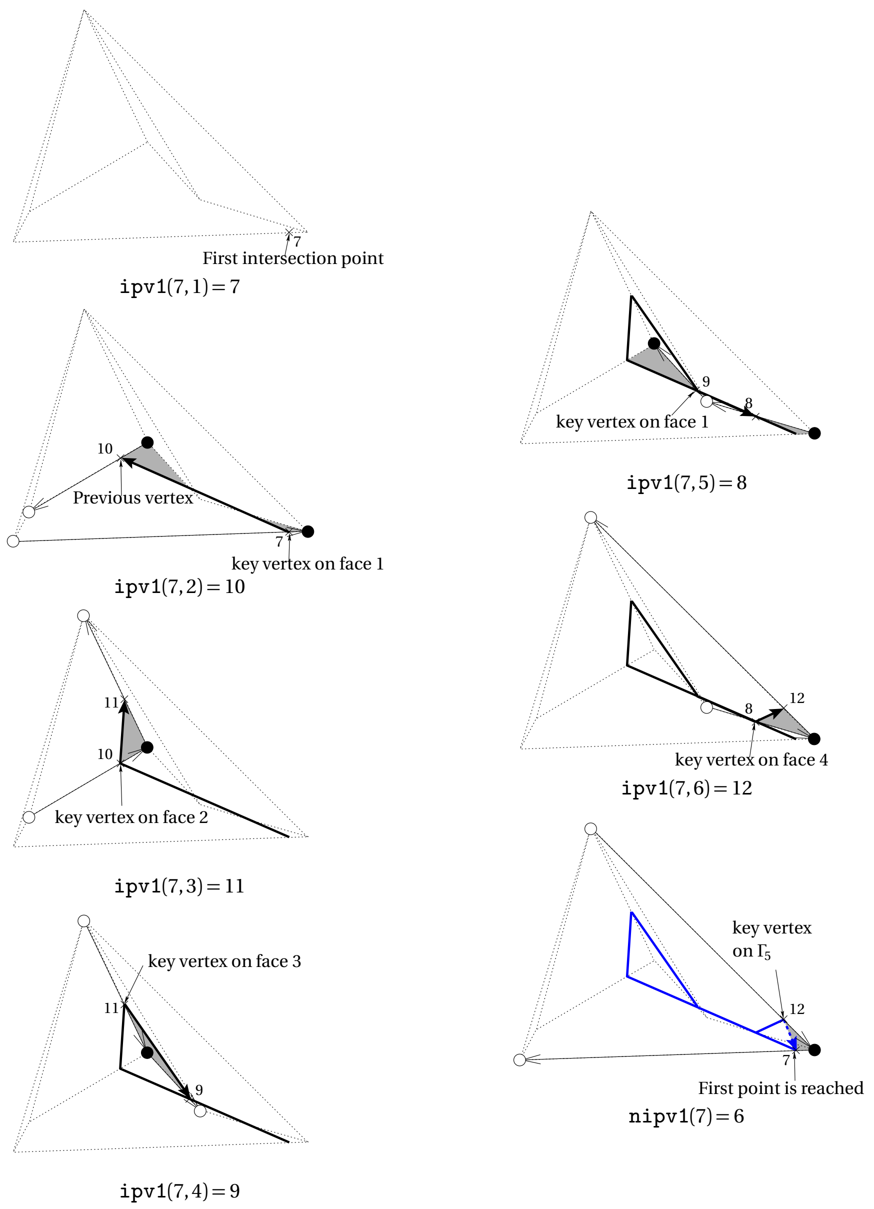


Figure 4.5: Sequence (from top to bottom and from left to right) of the capping procedure applied for the construction of the on-$\mathscr{P}$ face boundary (thick lines) in the example of Fig. 4.2.

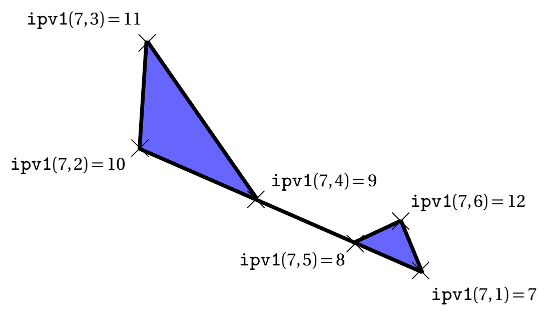


Figure 4.6: Vertex arrangement of the on-$\mathscr{P}$ face boundary in the example of Fig. 4.2.

face boundaries (four in total) can be seen in Fig. 4.7.

To assess the computational efficiency of the new version of the `VOFTools` package to perform truncation operations over non-convex cells, Table 4.3 shows the speedup achieves with respect to the previous `VOFTools` 3.2 version [10] coupled with the convex decomposition technique proposed in [1]. Note that the average computational efficiency is improved by almost five times, which represents a substantial reduction in the CPU time required by operations that must be executed repeatedly in, for example, geometric VOF methods.

4.2 Volume conservation enforcement in PLIC reconstruction

Figs. 4.8 and 4.9 show the sequence of application of the VCE operation, performed by calling the routine `enforv3d`, over the cell of Fig. 4.1 for a case with PLIC interface orientation $\boldsymbol{n}$ given by $\texttt{xnc} = 0$, $\texttt{ync} = -1$ and $\texttt{znc} = 0$, and ratio between the reference material and cell volumes (volume fraction `f`) $\texttt{v}/\texttt{vt} = 0.55$ (a sketch of the flow diagram of the VCE operation is also included in the figures to highlight the applied steps). The sequence for this case occurs as follows.[1]

1. A first tentative value, `ct1`, for the `c` solution is obtained by using the linear interpolation shown in Fig. 4.8(a), resulting
$$\texttt{ct1} = \frac{\texttt{v}}{\texttt{vt}} = 0.55.$$

2. The section of the function `v(c)` that satisfies the condition $\texttt{cmin} \le \texttt{ct1} < \texttt{cmax}$ is searched (Fig. 4.8(b)). The identified section is delimited by the tentative bracket values $\texttt{cmin} = 0.5$ and $\texttt{cmax} = 0.77$ (these values correspond to the constant of the planar interfaces passing through vertices 6 (thus, $\texttt{cmin} = -\texttt{xnc} \times \texttt{vertp(6,1)} - \texttt{ync} \times \texttt{vertp(6,2)} - \texttt{znc} \times \texttt{vertp(6,3)}$) and 1 (thus, $\texttt{cmax} = -\texttt{xnc} \times \texttt{vertp(1,1)} - \texttt{ync} \times \texttt{vertp(1,2)} -$

[1]For clarity in the explanation of this example, it has been omitted the activation, when appropriate to achieve a better computational efficiency, of the solution of
$$\texttt{v(c)} - (\texttt{vt} - \texttt{v}) = 0,$$
with interface normal $-\boldsymbol{n}$, instead of the solution of $\texttt{v(c)} - \texttt{v} = 0$ with interface normal $\boldsymbol{n}$ (see Reference [7] for the details).

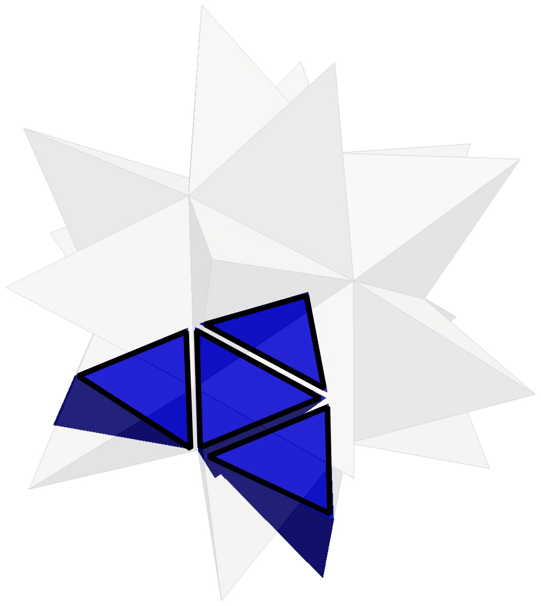
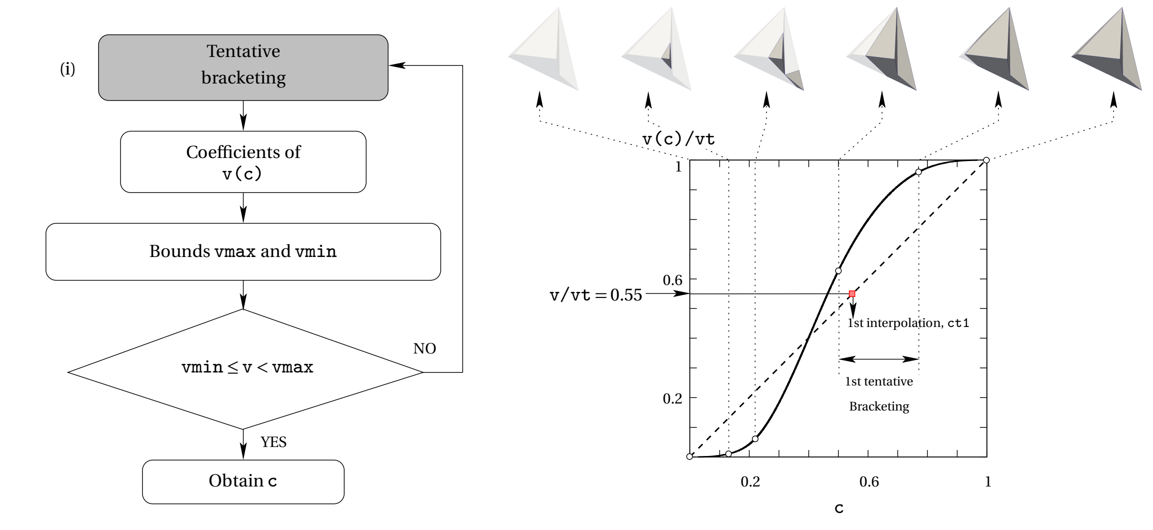


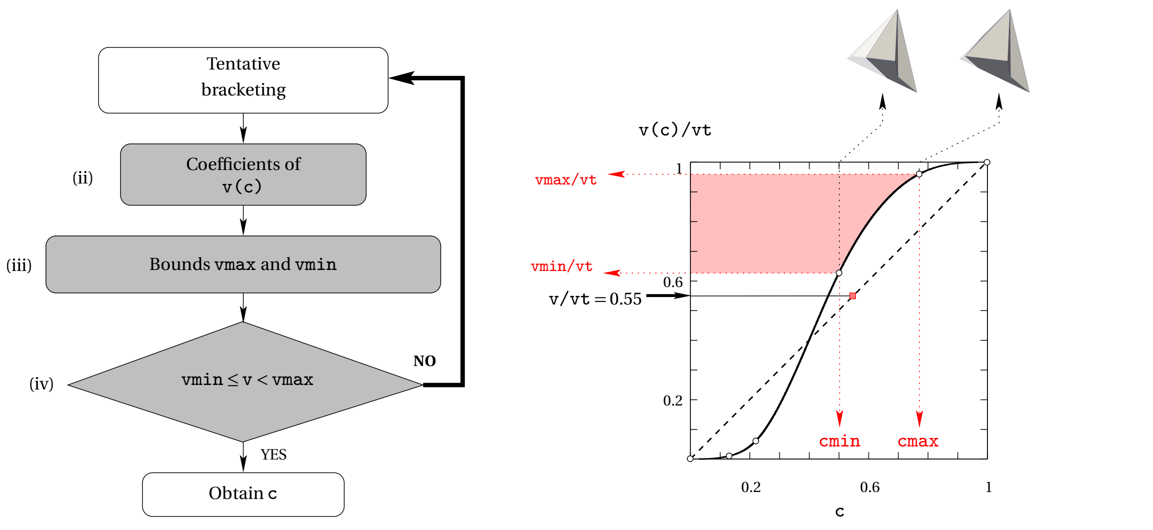
Figure 4.7: Example of the truncation between the stellated icosahedron of Fig. 3.2 and a half-space that produces four new on-$\mathscr{P}$ face boundaries (black thick lines). The truncated regions are highlighted in blue.

Table 4.3: Speedup achieved by the present `VOFTools` version with respect to the previous one [10] coupled with the convex decomposition technique proposed in [1] for truncation operations performed over several non-convex cells.

Non-convex cell geometry	Speedup
2D geometries	
Quadrangle	2.5
Pentagon	3.3
Hexagon	4.3
Stellated hexagon	6.8
Hollowed square	3.5
3D geometries	
Pentagonal pyramid	5.2
Stellated cube	4.4
Stellated dodecahedron	5.2
Stellated icosahedron	5.2
Hollowed cube	4.6

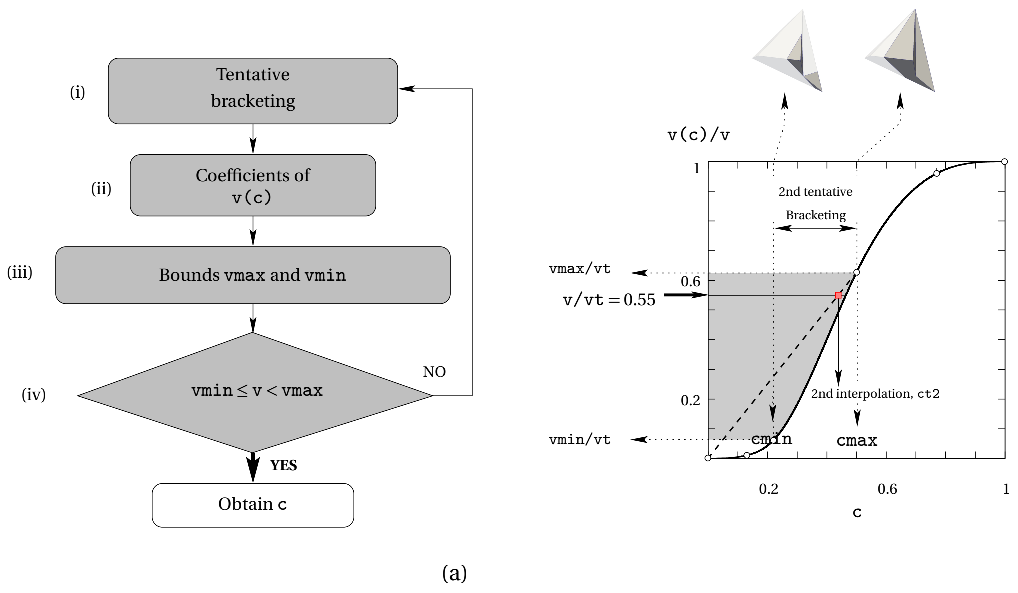


(a)



(b)

Figure 4.8: Sequence of application of the VCE operation in a PLIC reconstruction within the cell of Fig. 4.1 for a case with xnc $= 0$, ync $= -1$, znc $= 0$ and f $=$ v/vt $= 0.55$. (a) First tentative value of the solution and identification of the bracketing interval. (b) Computation of the coefficients of the analytical function v(c) valid inside the tentative bracketing interval, from which the bounds vmax and vmin are obtained, and check if the condition vmin $\leq$ v $<$ vmax is or not satisfied.



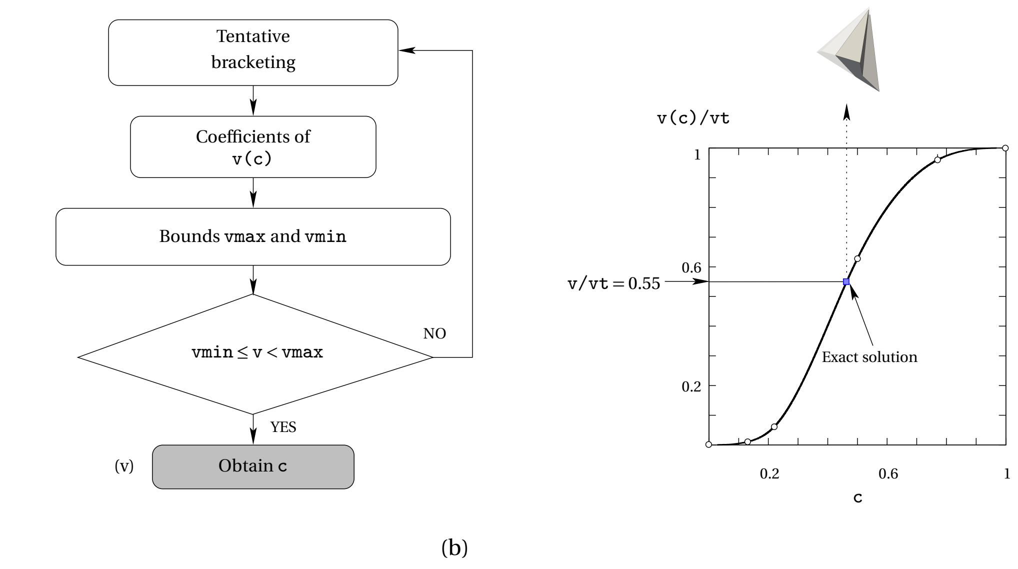


Figure 4.9: Continuation of the sequence of Fig. 4.8. (a) Steps (i) to (iii) are repeated and the volume condition is checked again. (b) The volume condition is satisfied and the final solution is analytically obtained.

$\mathtt{znc} \times \mathtt{vertp(1,3)}$)) and the corresponding coefficients of the function $\mathtt{v(c)/vt}$ are obtained from the analytical relations given in [9], resulting for this section as (the coefficients have been truncated by the fifth decimal)

$$\frac{\mathtt{v(c)}}{\mathtt{vt}} = 1.36248\mathtt{c}^3 - 5.41244\mathtt{c}^2 + 6.43269\mathtt{c} - 1.40609.$$

From the above expression, the bounds of the first tentative bracket are

$$\frac{\mathtt{vmax}}{\mathtt{vt}} = 0.96006$$

and

$$\frac{\mathtt{vmin}}{\mathtt{vt}} = 0.62745.$$

Note that the tentative bracket determined by these bounds does not contain the value $\mathtt{v/vt} = 0.55$ and a new tentative interpolation bracketing must be carried out.

3. A second tentative value, $\mathtt{ct2}$, is obtained by using the linear interpolation shown in Fig. 4.9(a) for which the upper limit of the interpolation line has been moved to the point on the curve $\mathtt{v(c)/vt}$ corresponding to $\mathtt{c} = 0.5$. In this way, the new interpolated tentative value results as

$$\mathtt{ct2} = 0.43828.$$

The section of the function $\mathtt{v(c)}$ that satisfies the condition $\mathtt{cmin} \le \mathtt{ct2} < \mathtt{cmax}$ is now delimited by the planar interfaces passing through vertices 6 (thus, $\mathtt{cmax} = -\mathtt{xnc} \times \mathtt{vertp(6,1)} - \mathtt{ync} \times \mathtt{vertp(6,2)} - \mathtt{znc} \times \mathtt{vertp(6,3)} = 0.5$) and 3 (thus, $\mathtt{cmin} = -\mathtt{xnc} \times \mathtt{vertp(3,1)} - \mathtt{ync} \times \mathtt{vertp(3,2)} - \mathtt{znc} \times \mathtt{vertp(3,3)} = 0.22$) and the corresponding analytical expression of $\mathtt{v(c)/vt}$ for this section results from [9] as

$$\frac{\mathtt{v(c)}}{\mathtt{vt}} = -12.29024\mathtt{c}^3 + 15.0666\mathtt{c}^2 - 3.80685\mathtt{c} + 0.30050.$$

The bounds of the second tentative bracket are

$$\frac{\mathtt{vmax}}{\mathtt{vt}} = 0.62745$$

and

$$\frac{\mathtt{vmin}}{\mathtt{vt}} = 0.06135.$$

This tentative bracket includes the value $\mathtt{v/vt} = 0.55$.

4. The final solution is analytically computed (Fig. 4.9(b)) as indicated in [5], resulting

$$\mathtt{c} = 0.46394.$$

Table 4.4: CPU times, relative to those obtained with the new `VOFTools` version, consumed by the previous package version (`VOFTools` v3.2 [10]) for volume truncation and conservation enforcement operations, and different convex cell geometries.

Cell geometry	Relative CPU-time (a) VCE	(b) Truncation
2D geometries		
Square	0.99	1.00
Regular hexagon	0.99	1.02
Irregular triangle	0.99	1.01
Irregular quadrangle	0.99	1.00
Irregular pentagon	0.98	1.00
Irregular hexagon	0.98	1.03
3D geometries		
Cube	1.00	1.08
Irregular hexahedron	1.23	1.15
Tetrahedron	1.20	1.10
Dodecahedron	1.06	1.15
Icosahedron	1.14	1.18
Complex polyhedron (32 vertices and 18 faces)	1.26	1.09

4.3 Computational efficiency in cases with convex geometries

In order to show that the new version of the `VOFTools` package maintains, or even improves, the computational efficiency over convex cells, Table 4.4 presents the CPU times consumed by the previous package version (`VOFTools` v3.2 [10]), relative to those consumed by the new version, for volume truncation and enforcement conservation operations over different convex cell geometries (a comparison for different non-convex cells was presented in Table 4.3). Note that the relative CPU-times maintain very close to unity, despite the profound change in the algorithms to extend the applicability of the new `VOFTools` package to more complex geometries. The achieved improvement, especially remarkable for 3D cases, is mainly due to the differences of the algorithms used to rearrange the vertices of the truncated polyhedra and the use of a more efficient computation of the analytic coefficients involved in the VCE operation.

4.4 Operations over a non-convex polyhedron with disjoint regions

This version of the `VOFTools` library can also be applied to non-convex polyhedra with disjoint regions, for which `v(c)` may be a discontinuous function. Figs. 4.10 and 4.11 show the liquid volume regions (blue color) obtained from the intersection between the `VOFTools` logo

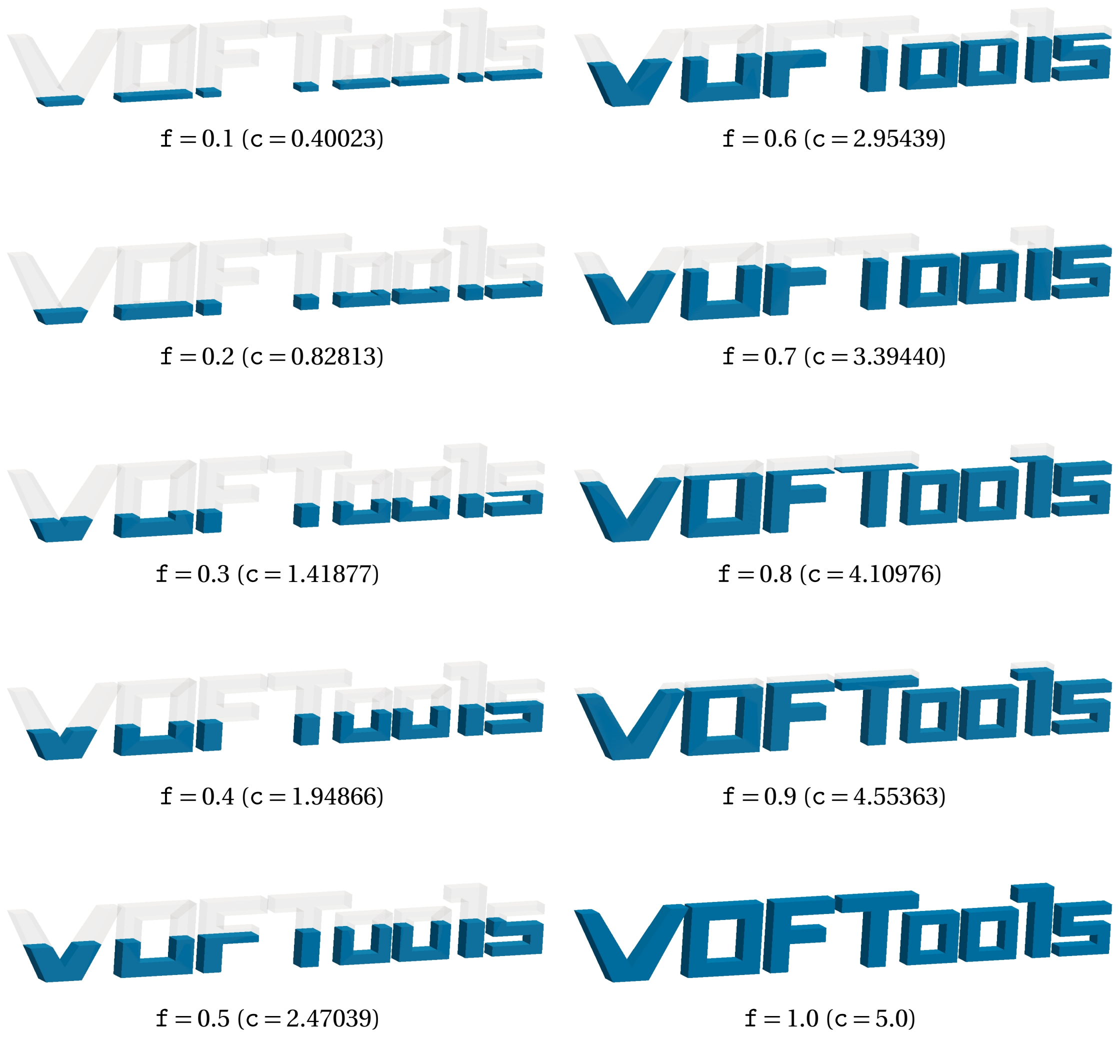
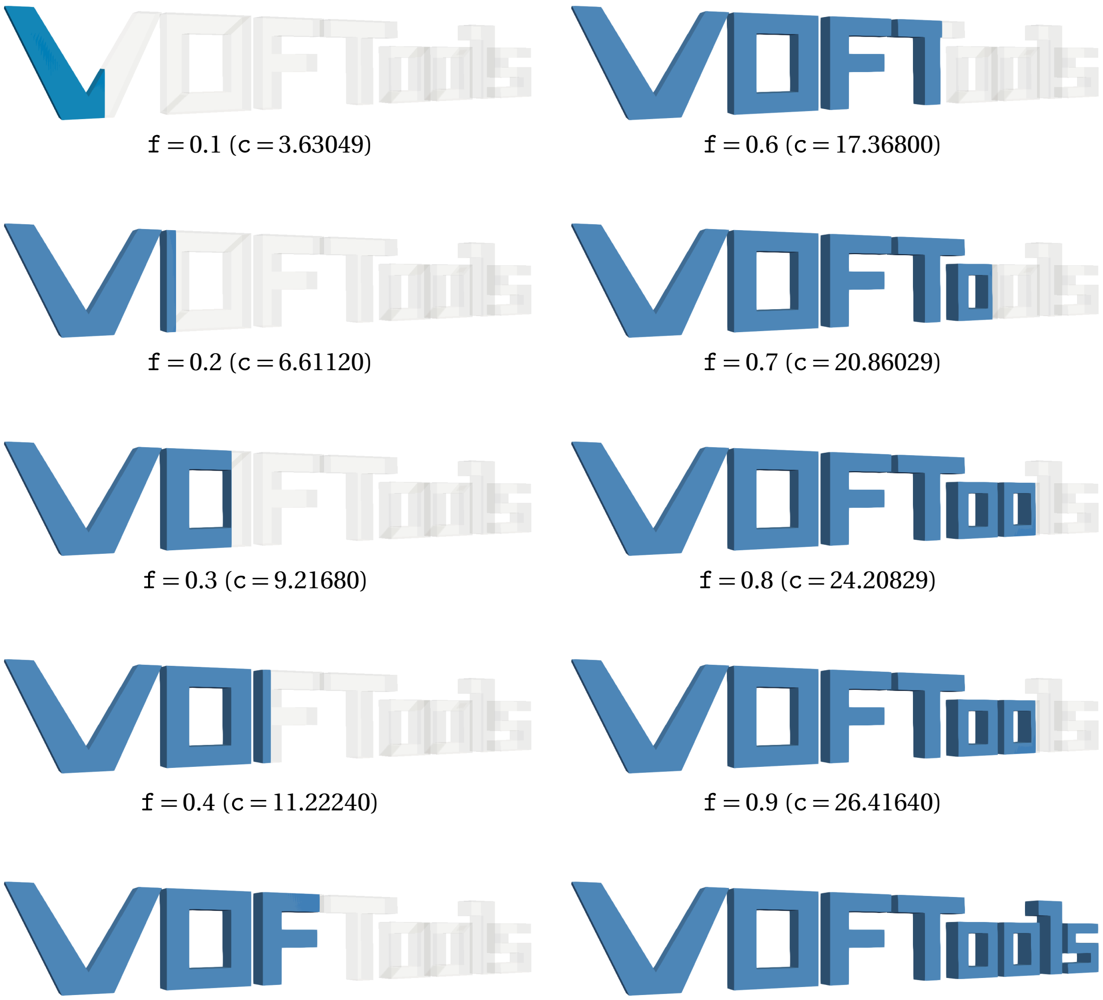

$\mathtt{f} = 0.1$ ($\mathtt{c} = 0.40023$)
$\mathtt{f} = 0.6$ ($\mathtt{c} = 2.95439$)
$\mathtt{f} = 0.2$ ($\mathtt{c} = 0.82813$)
$\mathtt{f} = 0.7$ ($\mathtt{c} = 3.39440$)
$\mathtt{f} = 0.3$ ($\mathtt{c} = 1.41877$)
$\mathtt{f} = 0.8$ ($\mathtt{c} = 4.10976$)
$\mathtt{f} = 0.4$ ($\mathtt{c} = 1.94866$)
$\mathtt{f} = 0.9$ ($\mathtt{c} = 4.55363$)
$\mathtt{f} = 0.5$ ($\mathtt{c} = 2.47039$)
$\mathtt{f} = 1.0$ ($\mathtt{c} = 5.0$)

Figure 4.10: Liquid volume regions (blue color) obtained from the intersection between the `VOFTools` logo cell (routine `voftoolslogo` implemented in the file `mesh.f`) and half-spaces defined by planes that contain the PLIC interfaces with normal vector $\boldsymbol{n} = (0, -1, 0)$ and different liquid volume fractions `f` (the solutions of the VCE problem are included in parenthesis). The original cell is depicted on each picture using partial transparent gray.



f = 0.1 (c = 3.63049)

f = 0.2 (c = 6.61120)

f = 0.3 (c = 9.21680)

f = 0.4 (c = 11.22240)

f = 0.5 (c = 13.94000)

f = 0.6 (c = 17.36800)

f = 0.7 (c = 20.86029)

f = 0.8 (c = 24.20829)

f = 0.9 (c = 26.41640)

f = 1.0 (c = 29.75)

Figure 4.11: Same as in Fig. 4.10 but with $\boldsymbol{n} = (-1, 0, 0)$.

cell (routine `voftoolslogo` in file `mesh.f`), where each letter is represented by a disjoint polyhedral region, and half-spaces defined by planes given by $\boldsymbol{n}\cdot\boldsymbol{x}+\mathtt{c}=0$ with, respectively, $\boldsymbol{n}=(\mathtt{xnc}=0,\mathtt{ync}=-1,\mathtt{znc}=0)$ and $\boldsymbol{n}=(\mathtt{xnc}=-1,\mathtt{ync}=0,\mathtt{znc}=0)$, and the constant `c` obtained by solving the VCE problem for different liquid volume fractions `f` (the original cell is depicted on each picture using partial transparent gray). Note that in these examples `c` corresponds to the minimum distance from the cell bottom to the multi-polygonal reconstructed PLIC interface, and for the example of Fig. 4.11 the function `v(c)` is discontinuous (see Fig. 4.12).

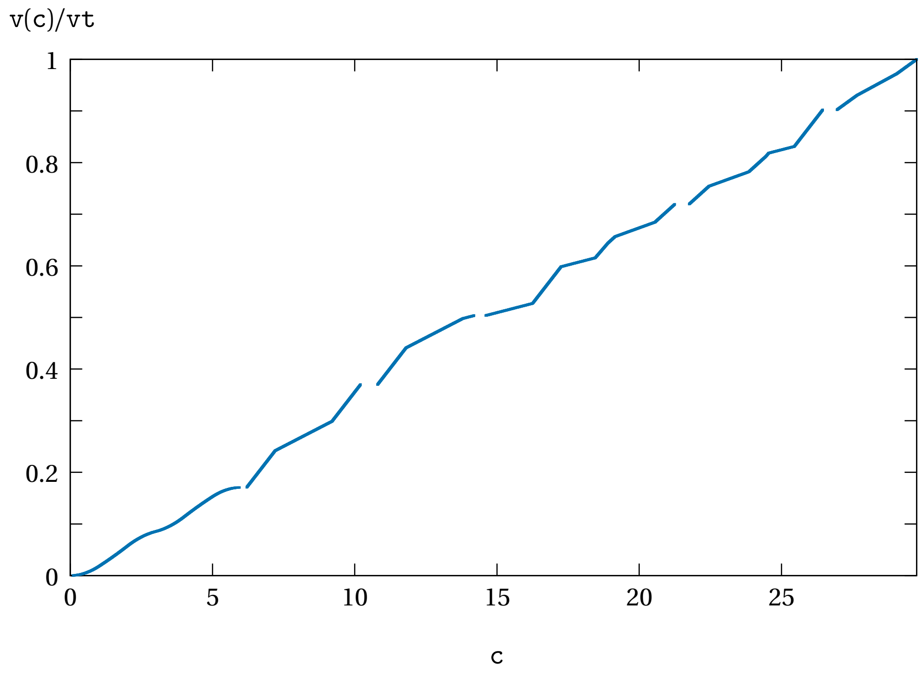


Figure 4.12: Liquid volume fraction `v(c)/vt` as a function of `c` for the `VOFTools` logo cell and an interface orientation given by $\boldsymbol{n}=(-1,0,0)$.

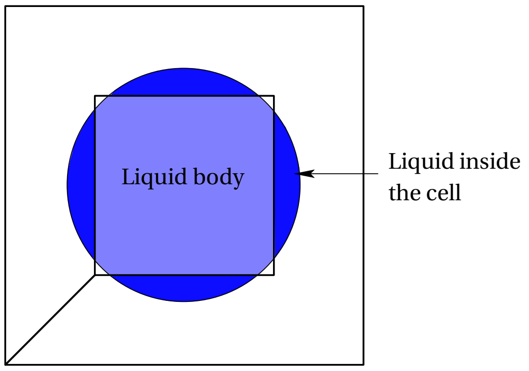
4.5 Volume fraction initialization

The accuracy of several initialization operations, performed by calling the routines `initf2d` for 2D and `initf3d` for 3D cases, is also assessed. Firstly, the following two cases with non-convex cells (see Fig. 4.13) are considered:

1. Initialization of a circular liquid body with radius r_l in an unit-length square with a half-length square hole located at its center (top picture in Fig. 4.13). The exact area fraction of liquid inside the cell can be obtained as

$$\mathtt{vf}_{\mathrm{exact}}=\frac{2^3}{3}r_l^2(\theta-\sin\theta),$$

where $\theta=2\arccos\left[1/(2^2 r_l)\right]$ and $2^{-2}\le r_l\le 2^{-3/2}$.



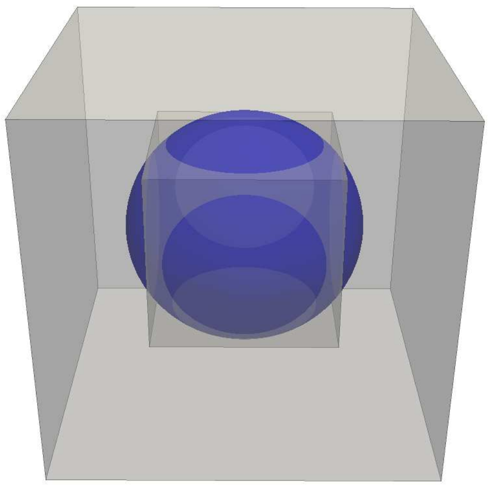


Figure 4.13: Liquid volume initialization of a circular and spherical liquid body (blue color) with radius $r_l = 0.325$ in the non-convex cell of Fig. 3.1 defined by the routine `nchollowedsquare` (top picture) and the non-convex cell of Fig. 3.2 defined by the routine `nchollowedcube` (bottom picture).

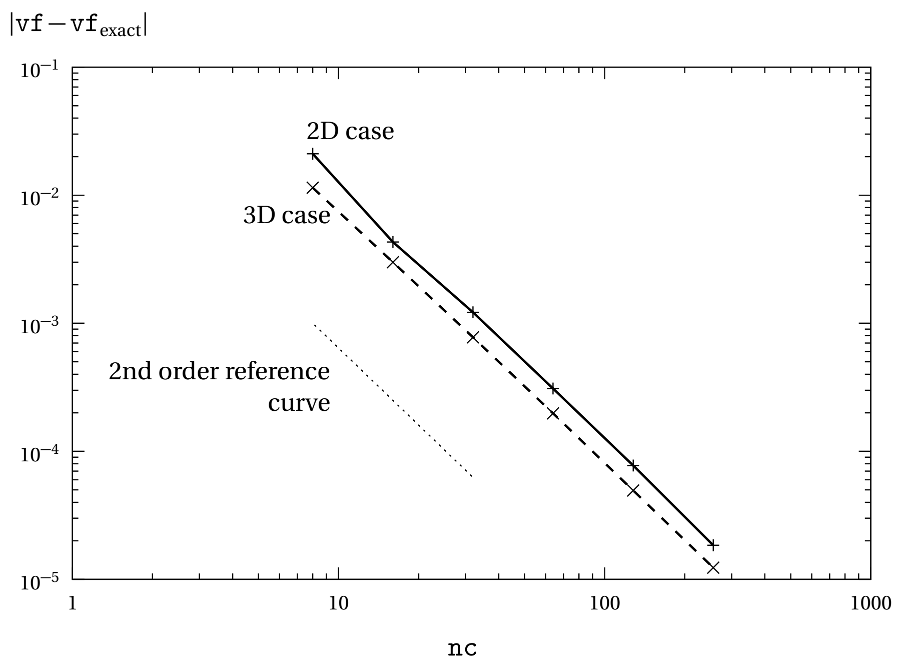


Figure 4.14: Initialization errors as a function of `nc` for the 2D and 3D cases of Fig. 4.13.

2. Initialization of a spherical liquid body with radius r_l in an unit-length cube with a half-length cubic hole located at its center (bottom picture in Fig. 4.13). The exact volume fraction of liquid inside the cell can be obtained as

$$\texttt{vf}_{\text{exact}} = \frac{2^4}{7}\pi\left(r_l - 2^{-2}\right)^2\left(2r_l + 2^{-2}\right),$$

with $2^{-2} \leq r_l \leq 2^{-3/2}$.

Fig. 4.14 shows the initialization error

$$|\texttt{vf} - \texttt{vf}_{\text{exact}}|$$

as a function of the number of divisions `nc` for these 2D and 3D cases with $r_l = 0.325$ (both liquid bodies are defined by calling the functions `func2d1` and `func3d1` implemented in the file `uservoftools.f`). For these and the rest of cases presented below, a high value of the tolerance `tol` is used to force the cell to be tagged as interfacial (see Reference [9] for a more detailed discussion about this tolerance). It can be observed that the results tend to reach the exact solution with a second-order convergence rate.

The accuracy of the initialization procedure for more complex cases, for which obtaining the exact solution analytically is not an easy task, has been obtained using the Richardson extrapolation method [2]. Two material bodies with elliptical and toroidal interfaces defined by calling the functions `func2d2` and `func3d2` implemented in the `uservoftools.f` file are given as a function of x, y, z-coordinates by, respectively,

$$f(x,y) = 1 - \left[\left(\frac{x-0.5}{0.5}\right)^2 + \left(\frac{y-0.5}{0.2}\right)^2\right] \tag{4.1}$$

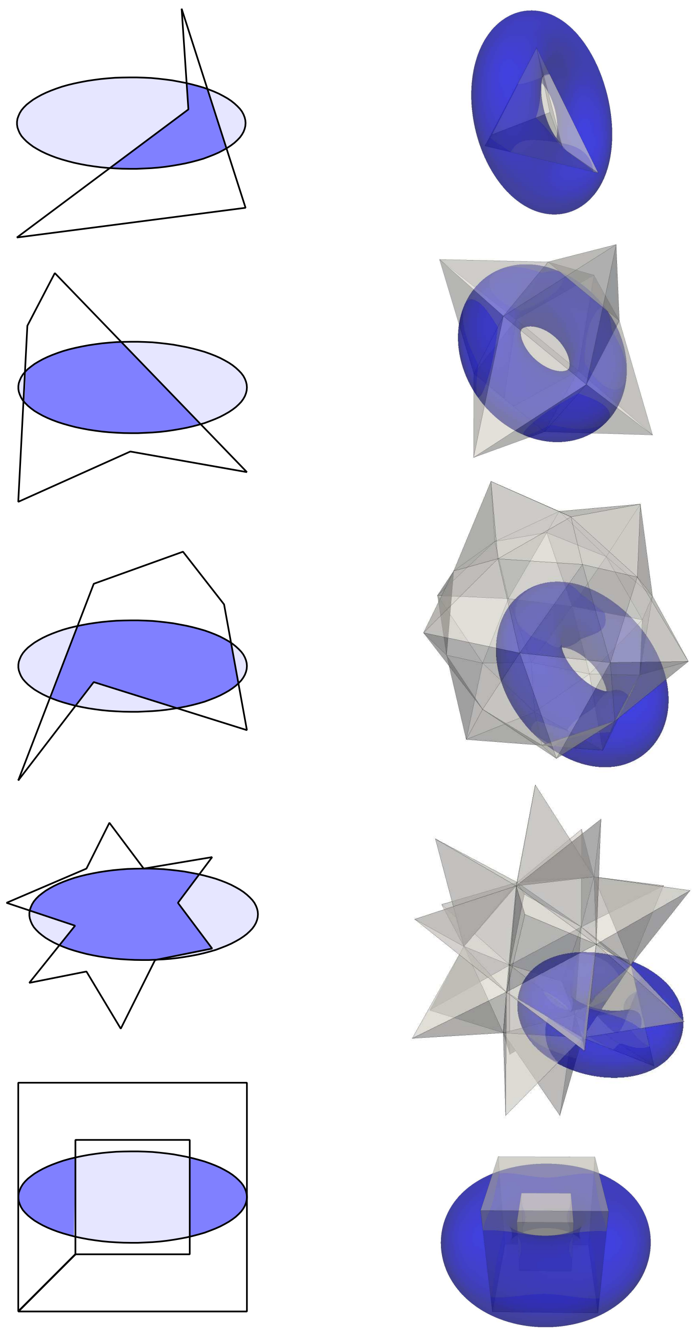


Figure 4.15: Examples for the initialization of the material bodies of Eq. (4.1) (left column) and Eq. (4.2) (right column) in several cells of Fig. 3.1 and Fig. 3.2, respectively.

and

$$f(x,y,z)=\left(\frac{1}{3}\right)^2-\left\{\frac{2}{3}-\left[(x-0.5)^2+(z-0.5)^2\right]^{0.5}\right\}^2-(y-0.5)^2. \tag{4.2}$$

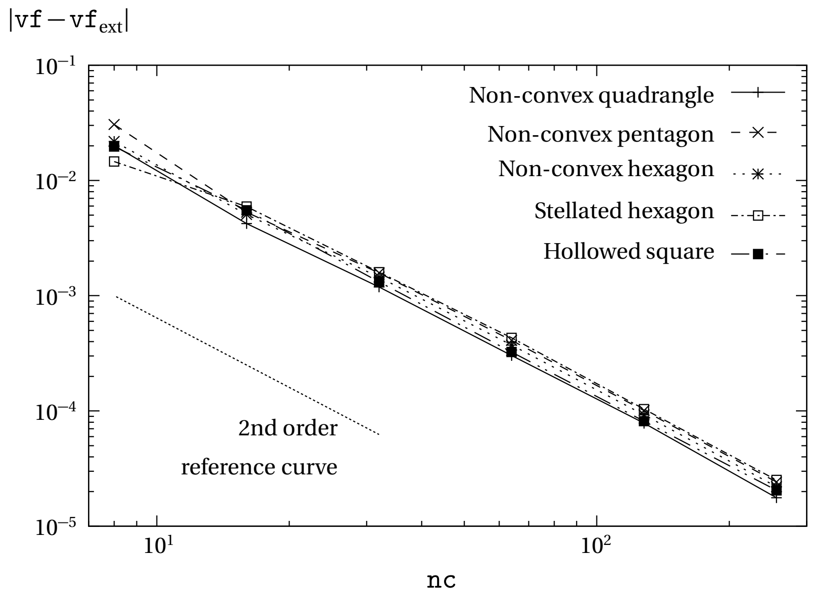
Fig. 4.15 shows the above material bodies along with several cells of Figs. 3.1 (left column) and 3.2 (right column). The accuracy is quantified through the error defined as

$$|\mathtt{vf}-\mathtt{vf}_{\text{ext}}|,$$

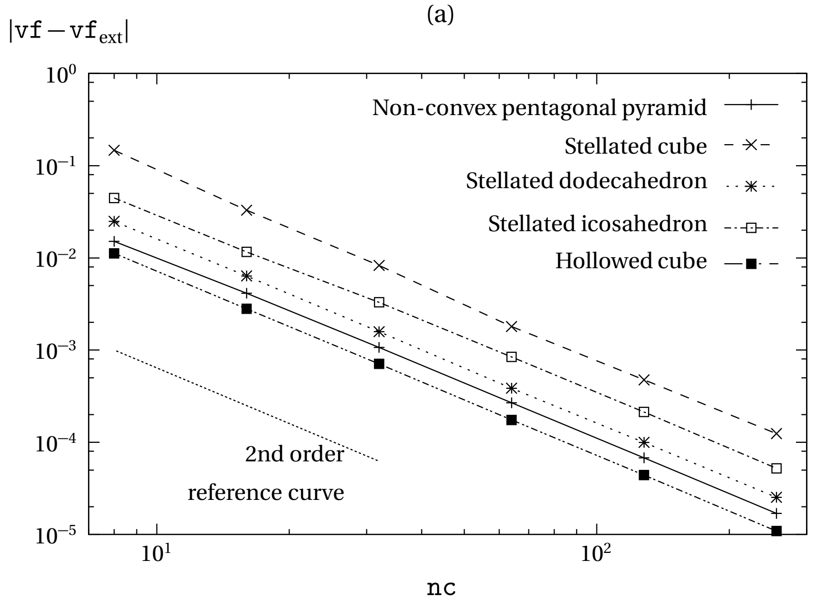
where $\mathtt{vf}_{\text{ext}}$ is the extrapolated numerical solution [2] obtained as

$$\mathtt{vf}_{\text{ext}}=\frac{4}{3}\mathtt{vf}_{1024}-\frac{1}{3}\mathtt{vf}_{512}$$

where $\mathtt{vf}_{1024}$ and $\mathtt{vf}_{512}$ are very accurate numerical results obtained with, respectively, $\mathtt{nc}=1024$ and 512. Fig. 4.16 shows the corresponding initialization errors as a function of nc. As it is expected, second-order convergence is also obtained for these more complex cases.



(a)



(b)

Figure 4.16: Initialization errors as a function of `nc` for several non-convex cells of Figs. 3.1 and 3.2 and the material bodies with (a) elliptical shape of Eq. (4.1) and (b) toroidal shape of Eq. (4.2), respectively.

5 | Test programs

The `VOFTools` package includes 2D and 3D test programs which are implemented in `FORTRAN` (`test2d.f` and `test3d.f`) and `C` (`test2d.c` and `test3d.c`). In these programs, all the arrays are first dimensioned, and the input data, which include the following parameters

- the cell geometry,
- the shape of the material body whose volume contained in the cell is computed by the initialization procedure,
- the material volume fraction defined by the reconstructed PLIC interface in the cell,
- the unit-length vector normal to the reconstructed PLIC interface in the cell,
- location of the point from which the distance to the reconstructed PLIC interface is calculated, and
- sub-grid and tolerance used in the initialization procedure,

are provided to the test program through the file `vofvardef` included in the supplied package (more details about this file can be found in Chapter 2). These input data are read by calling the routine `vofvardef` included in the file `uservoftools.f`. This routine is invoked in `FORTRAN` and `C` as follows:

```
      call vofvardef(f,icelltype,ishape,nc,tol,xnc,xp,ync,yp,znc,
     -               zp)
```

```
vofvardef_(&f,&icelltype,&ishape,&nc,&tol,&xnc,&xp,&ync,&yp,&znc,
&zp);
```

where the arguments are

`f`:	material volume/area fraction defined by the reconstructed PLIC interface in the cell
`icelltype`:	cell geometry index (Table 5.1 shows the value of the index corresponding to each cell geometry considered in the test programs)

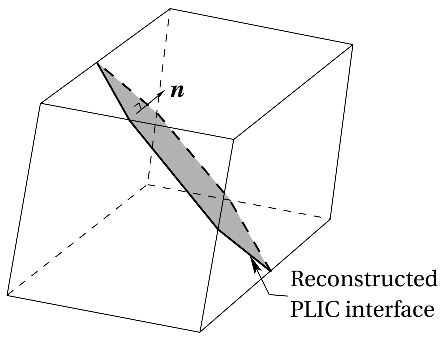


Figure 5.1: Positioning the interface by solving the VCE problem on a hexahedral cell.

`ishape`:	material body shape index (Table 5.2 shows the value of the index corresponding to each body shape considered in the test programs and defined through the user-defined implicit functions included in the file `uservoftools.f`)
`nc`:	number of divisions along each coordinate axis of the box superimposed to the cell
`tol`:	prescribed positive tolerance for the distance to the interface of the material body
`xnc, ync, znc`:	components of the unit-length vector normal to the interfacial plane
`xp, yp, zp`:	coordinates of point P from which the distance to the reconstructed PLIC interface is calculated

Then, the following operations are performed:

- Area/volume computation of the selected cell. The routine `toolv2d` in 2D or `toolv3d` in 3D is called, which gives the area or volume of the cell, `vt`.
- Area/volume conservation enforcement (Fig. 5.1). A linear/planar interface with a given orientation is positioned in the cell to cut off a given liquid area/volume fraction from a cell. The position, defined by parameter `c`, is obtained by calling the routine `enforv2d` in 2D or `enforv3d` in 3D.
- Area/volume truncation (Fig. 5.2). The intersection between the original cell and the half-space determined by the line/plane that contains the reconstructed PLIC interface is performed by calling the routine `inte2d` in 2D or `inte3d` in 3D.
- Distance computation[1] (Fig. 5.3). The distance between a point P and the new face

[1]At this moment, the current version of the package does not include the extension to non-convex geometries of `dist2d` and `dist3d` routines.

Table 5.1: Values of the index used to denote the cell geometry routines considered in the test programs.

Routine name	icelltype index
3D geometries	
cubicmesh	11
hexahemesh	12
tetramesh	13
dodecamesh	14
icosamesh	15
complexmesh	16
ncpentapyramid	111
nccubicpyramid	112
ncscubicmesh	113
nchexahemesh	114
ncdodecamesh	115
ncicosamesh	116
nchollowedcube	117
drilledcube	118
zigzagmesh	119
voftoolslogo	120
2D geometries	
squaremesh	1
hexagomesh	2
trianglemesh	3
quadranglemesh	4
pentagonmesh	5
hexagonmesh	6
ncquadranglemesh	101
ncpentagonmesh	102
nchexagonmesh	103
ncshexagonmesh	104
nchollowedsquare	105
ncmultisquare	106

(edge in 2D) resulting from the truncation operation for a convex cell is obtained by calling the routine `dist2d` in 2D or `dist3d` in 3D.

- Area/volume initialization (Fig. 5.4 shows an example for a non-convex cell). The fraction of the area/volume of a material body, defined by an user-defined implicit function included in the `uservoftools.f` file, contained inside the cell is computed by calling the routine `initf2d` in 2D or `initf3d` in 3D.

Table 5.2: Values of the `ishape` index used to denote the body shapes considered in the test programs.

`ishape` **index**	**Body shape**
	3D geometries
11	Sphere with radius 0.325 defined by the implicit function $f(x,y,z)=0.325^2-(x-0.5)^2-(y-0.5)^2-(z-0.5)^2$
12	Torus with major radius 2/3 and minor radius 1/3 defined by the implicit function $f(x,y,z) = \left(\frac{1}{3}\right)^2 - \left\{\frac{2}{3}-\left[(x-0.5)^2+(z-0.5)^2\right]^{1/2}\right\}^2-(y-0.5)^2$
	2D geometries
1	Circle with radius 0.325 defined by the implicit function $f(x,y)=0.325^2-(x-0.5)^2-(y-0.5)^2$
2	Ellipse with semi-major axis 0.5 and semi-minor axis 0.2 defined by the implicit function $f(x,y)=1-\left(\frac{x-0.5}{0.5}\right)^2-\left(\frac{y-0.5}{0.2}\right)^2$

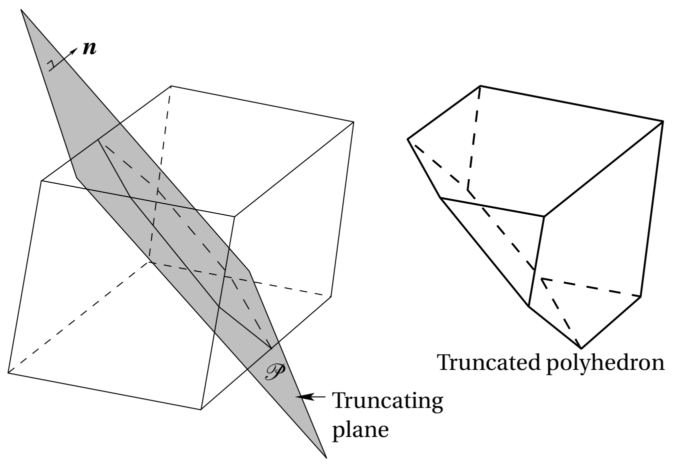


Figure 5.2: Truncation of the hexahedral cell by a plane containing the PLIC interface.

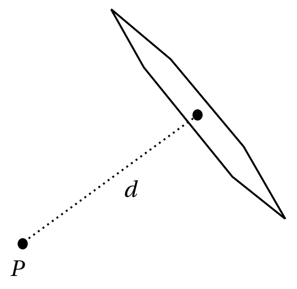


Figure 5.3: Computation of the distance from point P to the reconstructed PLIC interface.

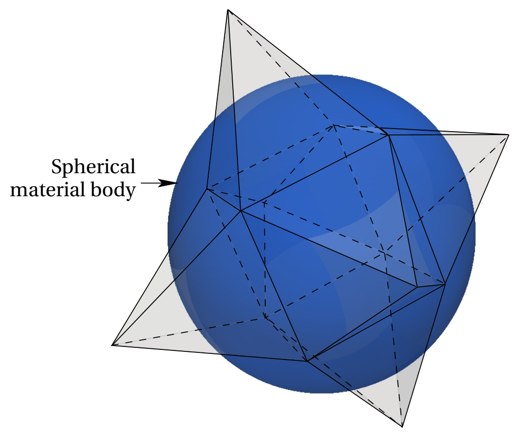


Figure 5.4: Initialization of the volume of a spherical material body contained inside a non-convex cell (example corresponding to `icelltype` = 113 and `ishape` = 11).

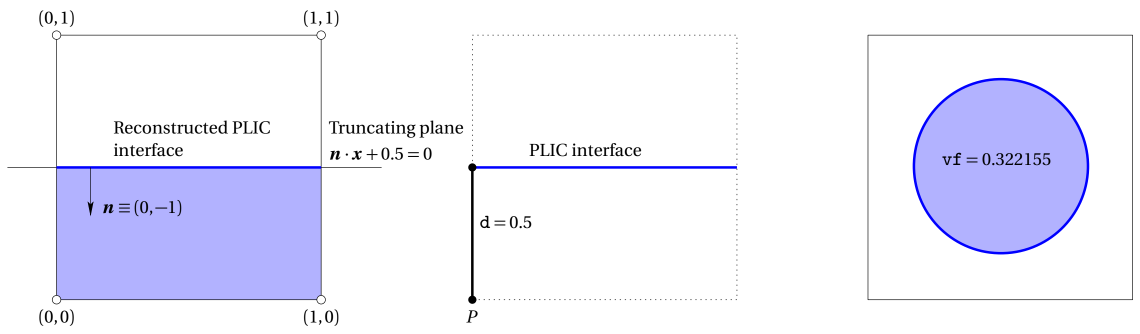

(0, 1)
(1, 1)
Reconstructed PLIC interface
Truncating plane
$\boldsymbol{n}\cdot\boldsymbol{x}+0.5=0$
PLIC interface
$\boldsymbol{n}\equiv(0,-1)$
d = 0.5
(0, 0)
(1, 0)
P
vf = 0.322155

Figure 5.5: Illustration of the results produced by the 2D test program for the input data corresponding to those of Table 5.3.

5.1 Examples

In the following, several examples of the execution of the 2D and 3D test programs are presented.

Table 5.3: Input data for example 1.

`icelltype`:	1
`ishape`:	1
`f`:	0.5
`xnc, ync`:	0.0, −1.0
`xp, yp`:	0.0, 0.0
`nc`:	10
`tol`:	10.0

Example 1. Table 5.3 shows the input data considered for this 2D example. The execution of the 2D test program (`test2d_f` or `test2d_c`) produces the following results:

Area of the cell, `vt`:	1.0
Solution of the VCE problem, `c`:	0.5
Distance from $\boldsymbol{x}_P=(\mathtt{xp},\mathtt{yp})$ to the interfacial segment, `d`:	0.5
Material area fraction in the cell, `vf`:	0.322155

Fig. 5.5 illustrates the results provided by the test program. By changing the position of point P to $(-1,0)$, $(0.5,0)$ and $(2,0)$ the test program produces, respectively, the results shown in Figs. 5.6(a), 5.6(b) and 5.6(c). Note that the exact area fraction of the circular material body contained in the cell is

$$\mathtt{vf}_{\text{exact}}=\pi 0.325^2.$$

Thus, the initialization error obtained using a division number $\mathtt{nc}=10$ is 9.7×10^{-3}. Increasing the division number to $\mathtt{nc}=20$, the material area fraction in the cell results as $\mathtt{vf}=0.329106$

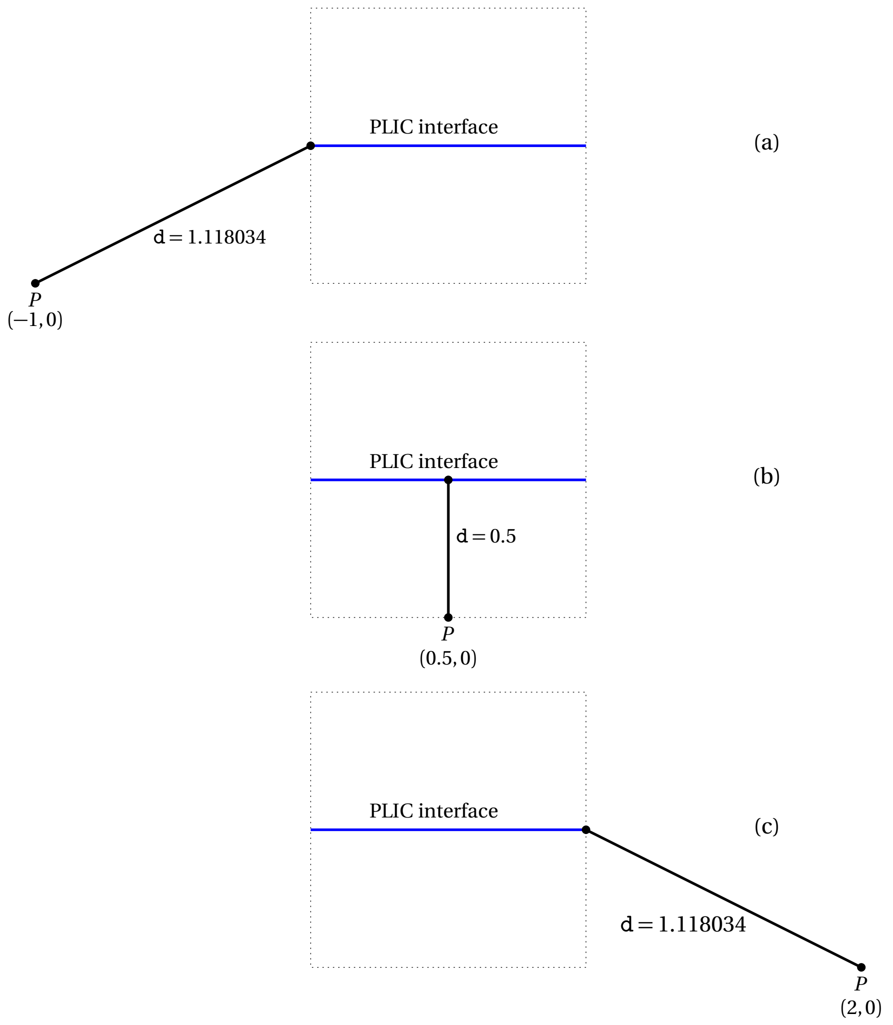


Figure 5.6: Distance `d` produced by the test program for the input data corresponding to those of Table 5.3 but changing the position of the point P to (a) $(-1,0)$, (b) $(0.5,0)$ and (c) $(2,0)$.

producing an initialization error equal to 2.7×10^{-3}, which shows a nearly second-order convergence accuracy.

Table 5.4: Input data for example 2.

`icelltype`:	106
`ishape`:	1
`f`:	0.5
`xnc, ync`:	$1/\sqrt{2}, 1/\sqrt{2}$
`xp, yp`:	-
`nc`:	10
`tol`:	10.0

Example 2. Table 5.4 shows the input data considered for this 2D example (note that the distance computation operation is ignored for examples like this with non-convex cells). The execution of the 2D test program (`test2d_f` or `test2d_c`) produces the following results:

Area of the cell, `vt`:	0.79
Solution of the VCE problem, `c`:	$-1/\sqrt{2}$
Material area fraction in the cell, `vf`:	0.148487

Fig. 5.7 illustrates the results provided by the test program. The exact area fraction of the cir-

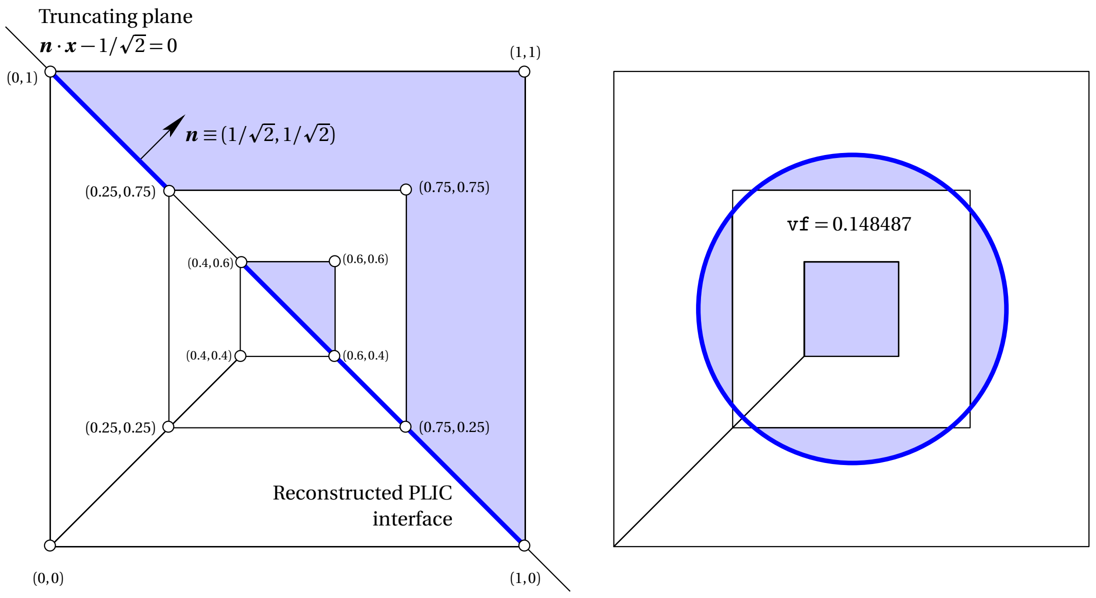


Figure 5.7: Illustration of the results produced by the 2D test program for the input data corresponding to those of Table 5.4.

cular material body contained in the cell is, for this example,

$$\mathtt{vf}_{\text{exact}} = \frac{2\times 0.325^2(\theta - \sin\theta) + \frac{1}{5^2}}{1 - \frac{1}{2^2} + \frac{1}{5^2}},$$

where $\theta = 2\arccos\left[1/\left(2^2 \times 0.325\right)\right]$. Thus, the initialization error obtained using a division number `nc` $= 10$ is 1.0×10^{-2}. Increasing the division number to `nc` $= 20$, the material area fraction in the cell results as `vf` $= 0.155460$ producing an initialization error equal to 3.0×10^{-3}, showing again a nearly second-order convergence accuracy.

Table 5.5: Input data for example 3.

`icelltype`:	11
`ishape`:	11
`f`:	0.5
`xnc, ync, znc`:	$0, -1, 0$
`xp, yp, zp`:	$0, 0, 0$
`nc`:	10
`tol`:	10.0

Example 3. Table 5.5 shows the input data considered for this 3D example. The execution of the 3D test program (`test3d_f` or `test3d_c`) produces the following results:

Area of the cell, `vt`:	1.0
Solution of the VCE problem, `c`:	0.5
Distance from $\boldsymbol{x}_P = (\mathtt{xp}, \mathtt{yp}, \mathtt{zp})$ to the interfacial segment, `d`:	0.5
Material area fraction in the cell, `vf`:	0.133985

Fig. 5.8 illustrates the results provided by the test program. By changing the position of point P to $(0.5, 0.5, 0.5)$, $(-0.5, 0, 1.5)$ and $(1.5, 0, 0.5)$ the test program produces, respectively, the results shown in Figs. 5.9(a), 5.9(b) and 5.9(c). The exact volume fraction of the spherical material body contained in the cell is

$$\mathtt{vf}_{\text{exact}} = \frac{4}{3}\pi 0.325^3.$$

Therefore, the initialization error obtained for this example is 9.8×10^{-3}. Increasing the division number to twice, the material volume fraction in the cell results as `vf` $= 0.141256$ producing an initialization error equal to 2.5×10^{-3}, showing a second-order convergence accuracy.

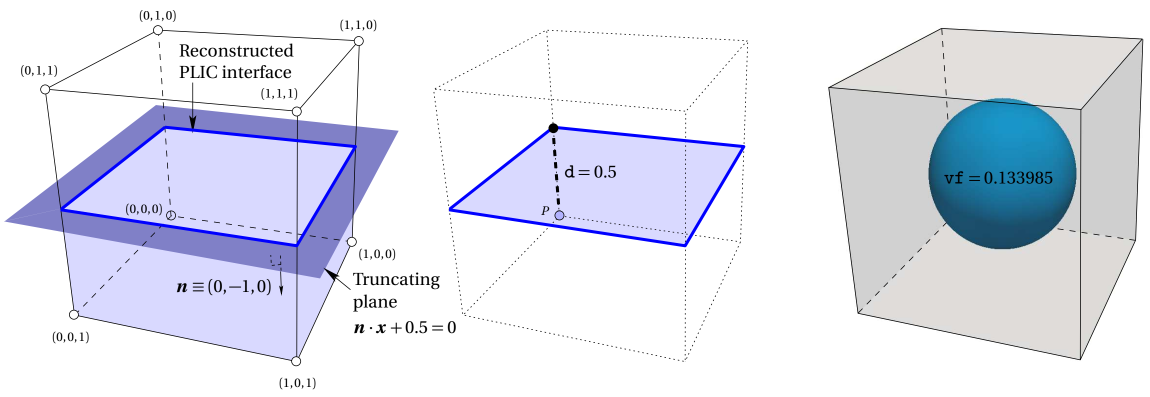


Figure 5.8: Illustration of the results produced by the 3D test program for the input data corresponding to those of Table 5.5.

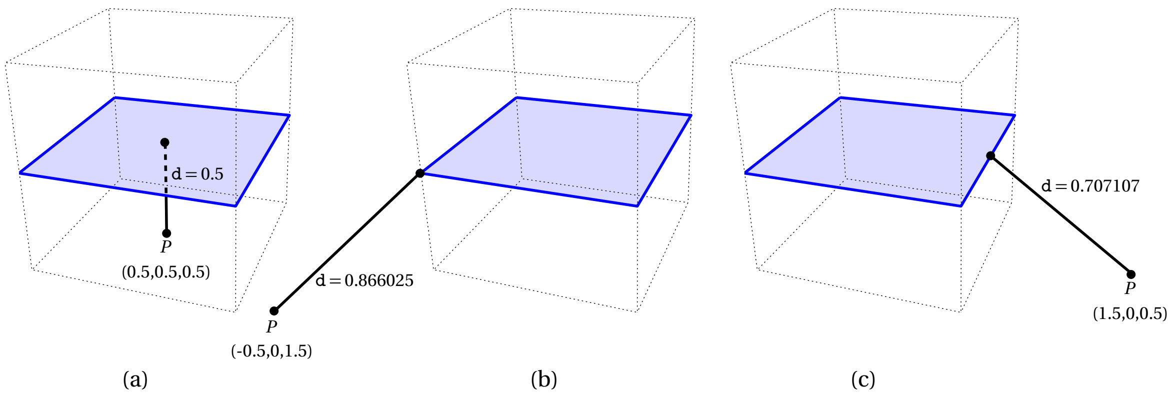


Figure 5.9: Distance d produced by the test program for the input data corresponding to those of Table 5.5 but changing the position of the point P to (a) $(0.5, 0.5, 0.5)$, (b) $(-0.5, 0, 1.5)$ and (c) $(1.5, 0, 0.5)$.

Example 4. Table 5.6 shows the input data considered for this 3D example. The execution of the 3D test program (`test3d_f` or `test3d_c`) produces the following results:

Area of the cell, `vt`:	3.0
Solution of the VCE problem, `c`:	0.5
Material volume fraction in the cell, `vf`:	0.321329

Fig. 5.10 illustrates the results provided by the test program. Due to the higher geometric complexity of this example, the volume initialization accuracy is quatified through the error defined as $|\texttt{vf} - \texttt{vf}_{\text{ext}}|$, where $\texttt{vf}_{\text{ext}}$ is the extrapolated numerical solutions obtained as

$$\texttt{vf}_{\text{ext}} = \frac{4}{3}\texttt{vf}_{1024} - \frac{1}{3}\texttt{vf}_{512},$$

where $\texttt{vf}_{1024}$ and $\texttt{vf}_{512}$ are very accurate numerical solutions obtained with, respectively, `nc` values of 1024 and 512, resulting $\texttt{vf}_{\text{ext}} = 0.3907925$. Therefore, the initialization error esti-

Table 5.6: Input data for example 4.

`icelltype:`	113
`ishape:`	12
`f:`	0.5
`xnc, ync, znc:`	$0, -1, 0$
`xp, yp, zp:`	-
`nc:`	10
`tol:`	10.0

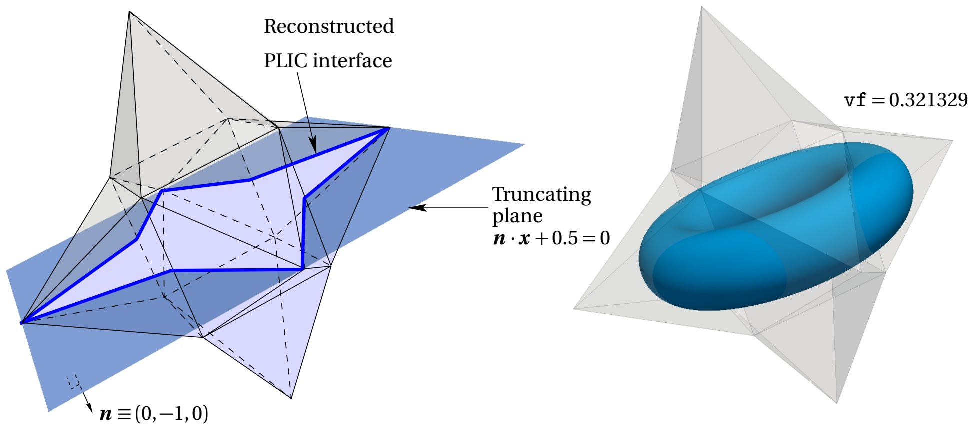

Reconstructed
PLIC interface
Truncating plane
$\boldsymbol{n} \cdot \boldsymbol{x} + 0.5 = 0$
$\boldsymbol{n} \equiv (0, -1, 0)$
vf = 0.321329

Figure 5.10: Illustration of the results produced by the test program for the input data corresponding to those of Table 5.6.

mated in this way results 6.9×10^{-2} for `nc` = 10. Increasing the division number `nc` twice, `vf` = 0.372632 and the initialization error results 1.8×10^{-2}, showing second-order convergence accuracy.

Bibliography

[1] H.T. Ahn, M. Shashkov, Multi-material interface reconstruction on generalized polyhedral meshes, *J. Comput. Phys.* 226 (2007) 2096-2132.

[2] I. Celik, W. Zhang, Calculation of numerical uncertainty using Richardson extrapolation: Application to some simple turbulent flow calculations, *ASME J. Fluids Eng.* 117 (1995) 439-445.

[3] A. Henderson, J. Ahrens, C. Law, The `ParaView` guide (2004).

[4] Kitware, `https://www.vtk.org`

[5] J. López, J. Hernández, Analytical and geometrical tools for 3D volume of fluid methods in general grids, *J. Comput. Phys.* 227 (2008) 5939-5948.

[6] J. López, P. Gómez, J. Hernández, F. Faura, A two-grid adaptive volume of fluid approach for dendritic solidification, *Comput. Fluids* 86 (2013) 326-342. Open access,

[7] J. López, J. Hernández, P. Gómez, F. Faura, A new volume conservation enforcement method for PLIC reconstruction in general convex grids, *J. Comput. Phys.* 316 (2016) 338-359. Open access,

[8] J. López, J. Hernández, P. Gómez, F. Faura, `VOFTools` — A software package of calculation tools for volume of fluid methods using general convex grids, *Comput. Phys. Commun.* 223 (2018) 45-54. Open access,

[9] J. López, J. Hernández, P. Gómez, F. Faura, Non-convex analytical and geometrical tools for volume truncation, initialization and conservation enforcement in VOF methods, *J. Comput. Phys.* 392 (2019) 666-693. Open access,

[10] J. López, J. Hernández, P. Gómez, C. Zanzi, R. Zamora, `VOFTools` 3.2: Added VOF functionality to initialize the liquid volume fraction in general convex cells, *Comput. Phys. Commun.* 245 (2019) 106859.

[11] J. López, J. Hernández, P. Gómez, C. Zanzi, R. Zamora, `VOFTools` 5: An extension to non-convex geometries of calculation tools for volume of fluid methods, submitted to *Comput. Phys. Commun.*

[12] E.A. Merritt, et al., `Gnuplot`. An interactive plotting program. `http://www.gnuplot.info/`

[13] R. Scardovelli, S. Zaleski, Analytical relations connecting linear interfaces and volume fractions in rectangular grids, *J. Comput. Phys.* 164 (2000) 228-237.

[14] http://www.dimf.upct.es/personal/lrj/voftools.html.