XML GUIDE
FOR DUALSPHYSICS

mDBC: MODIFIED DYNAMIC BOUNDARY CONDITION

July 2020

DualSPHysics team

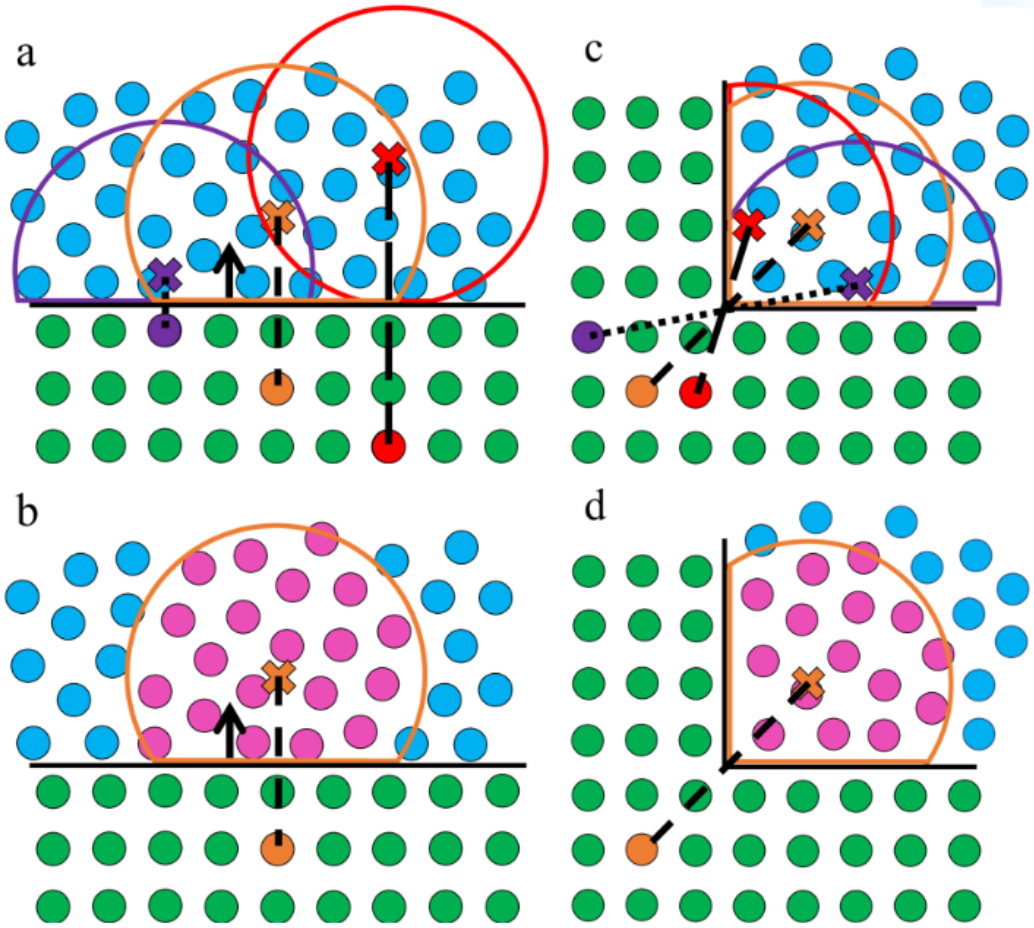
http://dual.sphysics.org

mDBC: modified Dynamic Boundary Condition

mDBC is the new accurate boundary condition available in DualSPHysics v5.

This boundary condition presents several improvements regarding to DBC:

- Physical density/pressure values in boundary particles.
- Avoids the separation distance between boundary and fluid particles (GAP).



Mirroring of ghost nodes (**crosses**) and the kernel radius around the ghost nodes for boundary particles in a flat surface (a) and a corner (c).

Fluid particles (**pink**) included in the kernel sum around ghost nodes for boundary particles in a flat surface (b) and a corner (d).

mDBC: modified Dynamic Boundary Condition

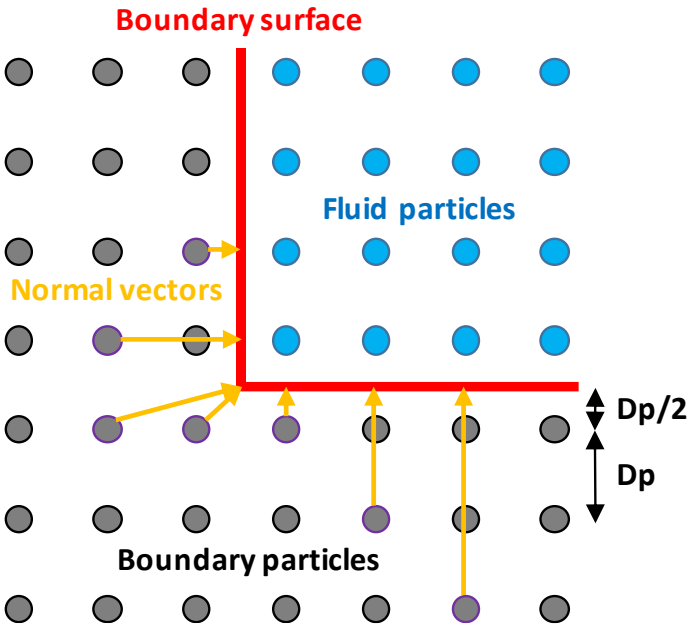
mDBC is the new accurate boundary condition available in DualSPHysics v5.

This boundary condition presents several improvements regarding to DBC:

- Physical density/pressure values in boundary particles.
- Avoids the separation distance between boundary and fluid particles (GAP).

However, creating the initial condition for mDBC is more complicated since:

- Vectors from boundary particles to actual boundary limit are necessary (**normals**).
- Distance between boundary limit and boundary particles should be half the initial inter-particle distance (**Dp/2**).
- Several boundary particles layers are necessary (**three layers or more**). It depends on the SPH smoothing length (h). Three layers are enough when $2h \leq 3Dp$.

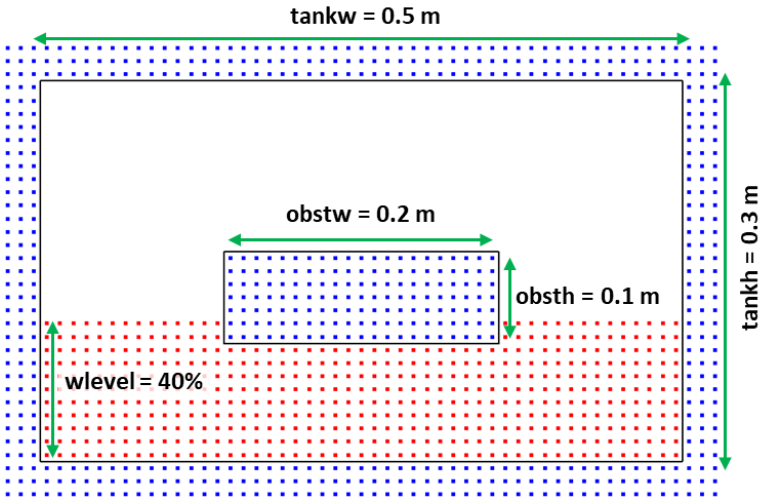


Two methods to generate normal vectors for mDBC

There are two main methods to create the normal vectors for mDBC:

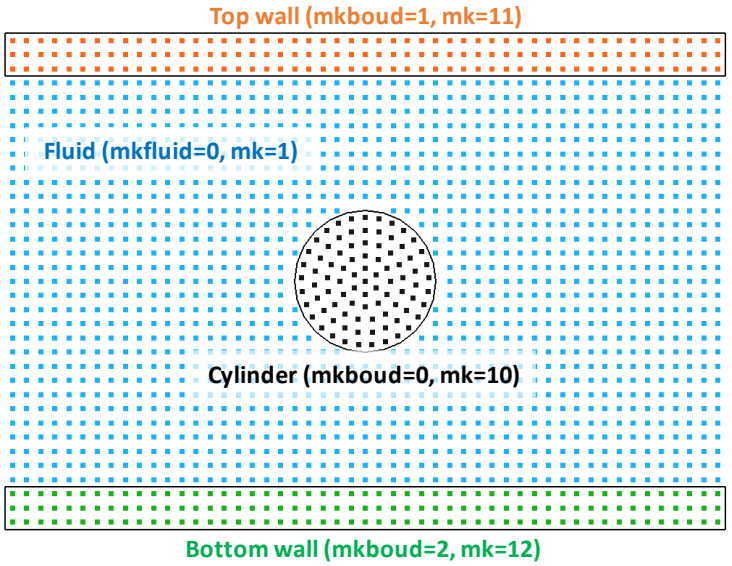
Method 1: Normal vectors of boundary particles are automatically computed starting from the actual geometry (the triangles that describe the surface of the objects). This actual geometry can be created by GenCase as VTK files.

Example: ***Sloshing tank with obstacle***



Method 2: Normal vectors of boundary particles are explicitly defined in the DualSPHysics configuration.

Example: ***Flow past a circular cylinder***



Both methods can be also combined to define the normal vectors.

Method 1: Automatic normal vectors from actual geometry

```
<mainlist>
    <!-- User-defined variables  -->
    <!-- ======================  -->
    <newvarcte tankw="0.5" tankh="0.3" _rem="Tank dimensions (width and height)" />
    <newvarcte obstw="0.20" obsth="0.06" _rem="Obstacle dimensions (width and height)"/>
    <newvarcte obstz="0.1" _rem="Distance from obstacle ot the bottom of the tank" />
    <newvarcte wlevel="40" _rem="Water level in percentage of tank hight" />
    <!-- Actual geometry at dp/2 -->
    <!-- ======================  -->
    <setshapemode>actual | bound</setshapemode>
    <!-- Actual tank -->
    <setmkbound mk="0" />
    <setnormalinvert invert="true" />
    <drawbox>
        <boxfill>bottom|top|left|right</boxfill>
        <point x="#-tankw/2" y="-0.1" z="0" />
        <size x="#tankw" y="0.2" z="#tankh" />
        <layers vdp="-0.5" />
    </drawbox>
    <!-- Actual obstacle -->
    <setnormalinvert invert="false" />
    <drawbox>
        <boxfill>bottom|top|left|right</boxfill>
        <point x="#-obstw/2" y="-0.1" z="#obstz" />
        <size x="#obstw" y="0.2" z="#obsth" />
        <layers vdp="0.5" />
    </drawbox>
    <!-- Save geometry file and remove particles -->
    <shapeout file="hdp" />
    <setmkvoid />
    <redraw />
    <!-- Particle generation -->
    <!-- ===================  -->
    <setshapemode>actual | bound</setshapemode>
    <setdrawmode mode="full" />
    <!-- Tank particles -->
    <setmkbound mk="0" />
    <drawbox>
        <boxfill>bottom|top|left|right</boxfill>
        <point x="#-tankw/2" y="-0.1" z="0" />
        <size x="#tankw" y="0.2" z="#tankh" />
        <layers vdp="0,1,2" />
    </drawbox>
    <!-- Obstacle particles -->
    <drawbox>
        <boxfill>solid</boxfill>
        <point x="#-obstw/2" y="-0.1" z="#obstz" />
        <size x="#obstw" y="0.2" z="#obsth" />
    </drawbox>
    <!-- Fluid particles-->
    <setmkfluid mk="0" />
    <fillbox x="0" y="0" z="#Dp*2">
        <modefill>void</modefill>
        <point x="#-tankw/2" y="-0.1" z="0" />
        <size x="#tankw" y="0.2" z="#tankh*wlevel/100" />
    </fillbox>
</mainlist>
```

Defines all dimensions and magnitudes necessary to design the case.

Creates the actual geometry of the case where the boundary limits are at Dp/2 from boundary particles in a VTK file.

[CaseName]_hdp_Actual.vtk

This VTK file is used to compute the normal vectors automatically. Thus this part is not necessary if one VTK with actual geometry is already available.

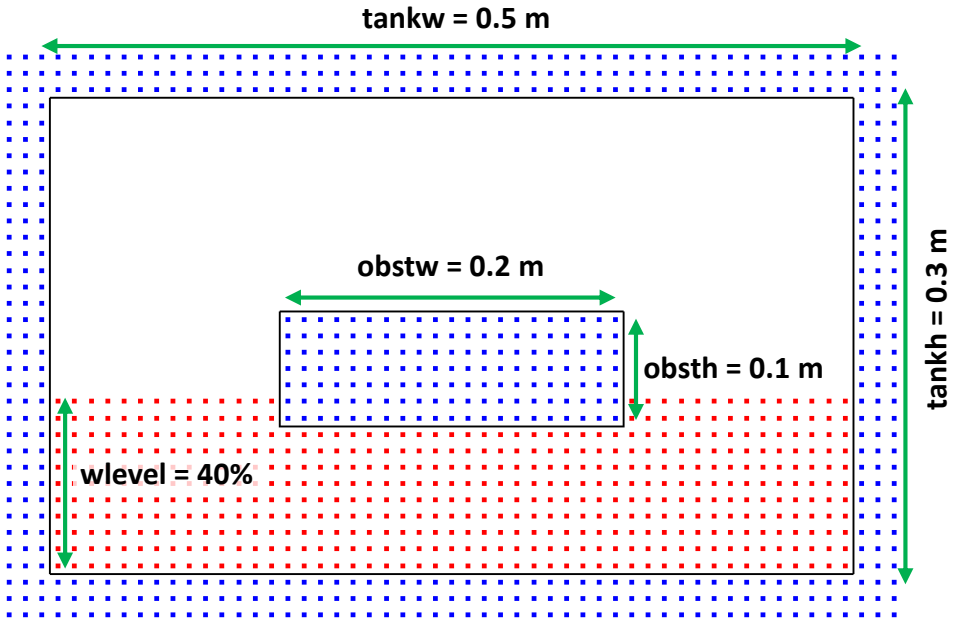
Creates the initial condition of the simulation (boundary and fluid particles).

Method 1: Automatic normal vectors from actual geometry

GenCase program allows to define variables in the XML and numerical expressions with these variables can be solved by GenCase and DualSPHysics programs (see XML_GUIDE_v5.0.pdf). This feature is very useful for mDBC cases, since some values are used in several places in the XML file.

First part of section <mainlist> defines all dimensions and magnitudes necessary to design a **sloshing tank with one obstacle**.

```
<mainlist>
    <!-- User-defined variables  -->
    <!-- ======================  -->
    <newvarcte tankw="0.5" tankh="0.3" _rem="Tank dimensions (width and height)"/>
    <newvarcte obstw="0.20" obsth="0.06" _rem="Obstacle dimensions (width and height)"/>
    <newvarcte obstz="0.1" _rem="Distance from obstacle ot the bottom of the tank"/>
    <newvarcte wlevel="40" _rem="Water level in percentage of tank hight"/>
```



Blue points are the boundary particles. Note that 3 boundary layers are used.

Red points are the fluid particles.

Black lines are the actual boundary limits at Dp/2 from boundary particles.

Method 1: Automatic normal vectors from actual geometry

Second part of XML generates a VTK file with the actual geometry at Dp/2 from boundary particles. The user-defined variables explained above are used to create the tank and the obstacle inside the tank.

```
<!-- Actual geometry at dp/2 -->
<!-- ======================  -->
<setshapemode>actual | bound</setshapemode>
<!-- Actual tank -->
<setmkbound mk="0" />
<setnormalinvert invert="true" />
<drawbox>
    <boxfill>bottom|top|left|right</boxfill>
    <point x="#-tankw/2" y="-0.1" z="0" />
    <size x="#tankw" y="0.2" z="#tankh" />
    <layers vdp="-0.5" />
</drawbox>
<!-- Actual obstacle -->
<setnormalinvert invert="false" />
<drawbox>
    <boxfill>bottom|top|left|right</boxfill>
    <point x="#-obstw/2" y="-0.1" z="#obstz" />
    <size x="#obstw" y="0.2" z="#obsth" />
    <layers vdp="0.5" />
</drawbox>
<!-- Save geometry file, remove particles and reset draw configuation -->
<shapeout file="hdp" />
<resetdraw />
```

The new feature of GenCase to define objects with several **layers** according to Dp is used to define the actual boundary at Dp/2 from boundary particles.

The tank is created Dp/2 smaller and the obstacle Dp/2 bigger.

The normal vectors of geometry must point to the fluid. Default direction of normal vectors is fine for solid bodies, but they must be reversed for container bodies (**tank**).

In this case we invert normal direction before creating the tank and we deactivate the option after it.

← Saves geometry in the VTK file "***[CaseName]_hdp_Actual.vtk***".

← Removes current particles and reset draw configuration since this part is only used to create the file [CaseName]_hdp_Actual.vtk.

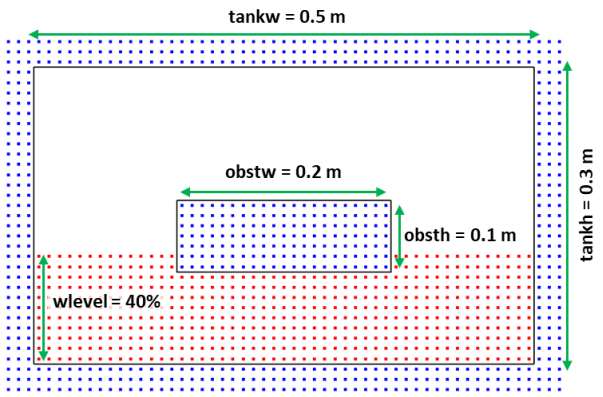
Method 1: Automatic normal vectors from actual geometry

Third part of XML generates the boundary (tank and obstacle) and fluid particles. Note that several boundary layers are necessary and the use of the “layers” feature of GenCase makes it easy.

```
    <!-- Particle generation -->
    <!-- ==================  -->
    <setshapemode>actual | bound</setshapemode>
    <setdrawmode mode="full" />
    <!-- Tank particles -->
    <setmkbound mk="0" />
    <drawbox>
        <boxfill>bottom|top|left|right</boxfill>
        <point x="#-tankw/2" y="-0.1" z="0" />
        <size x="#tankw" y="0.2" z="#tankh" />
        <layers vdp="0,1,2" />
    </drawbox>
    <!-- Obstacle particles -->
    <drawbox>
        <boxfill>solid</boxfill>
        <point x="#-obstw/2" y="-0.1" z="#obstz" />
        <size x="#obstw" y="0.2" z="#obsth" />
    </drawbox>
    <!-- Fluid particles-->
    <setmkfluid mk="0" />
    <fillbox x="0" y="0" z="#Dp*2">
        <modefill>void</modefill>
        <point x="#-tankw/2" y="-0.1" z="0" />
        <size x="#tankw" y="0.2" z="#tankh*wlevel/100"/>
    </fillbox>
</mainlist>
```

Creates the fluid tank using **3 boundary layers** at 0xDp, 1xDp and 2xDp outwards.

Creates the obstacle as **solid** so “layers” parameter is no necessary.

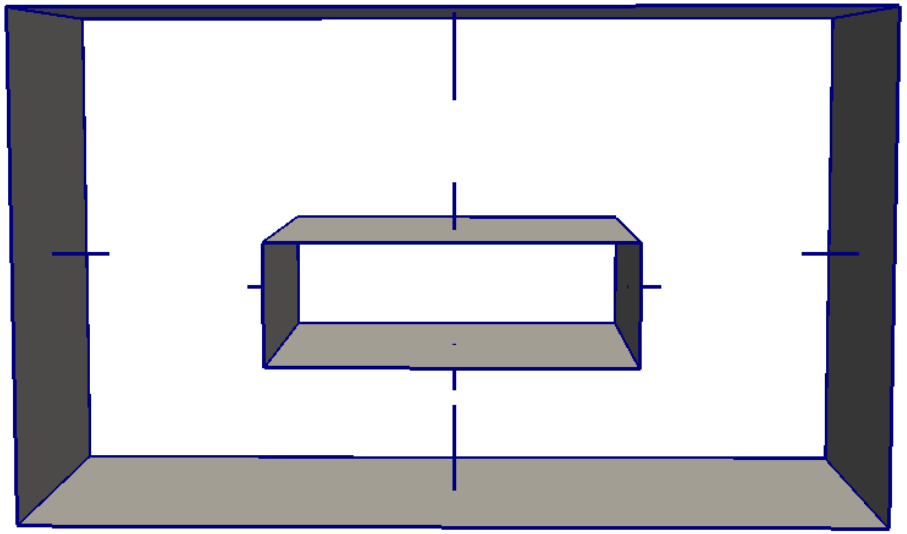


Method 1: Automatic normal vectors from actual geometry

Fourth part of XML configures the **automatic calculation of normal vectors** for each boundary particle starting from a geometry VTK file.

```
        </geometry>
        <normals>
            <distanceh value="3.0" comment="(default=2)"/>
            <geometryfile file="[CaseName]_hdp_Actual.vtk"/>
            <svshapes value="1" comment="(default=0)"/>
        </normals>
    </casedef>
```

distanceh: Only creates the normal vectors with size smaller than value*h (SPH smoothing length).
geometryfile: File name of geometry file in VTK format (“NONE” when an external file is not used).
svshapes: Saves VTK with geometry in triangles and quads with its normal vectors to debug.



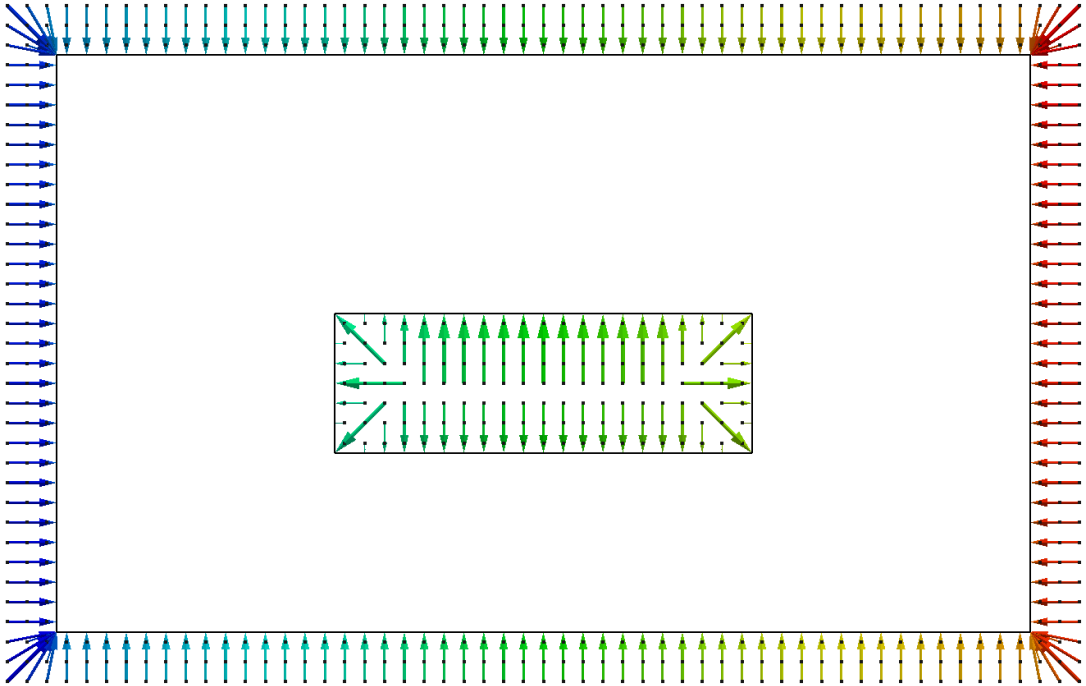
The option “svsphapes” creates the file ***SloshingObstacle_GeometryShapes.vtk***, where the normal vectors are the **orthogonal blue lines**. This file is useful to check the direction of normal vectors.

Method 1: Automatic normal vectors from actual geometry

Finally, the **use of mDBC** boundary condition is configured in execution parameters section. Simulation with mDBC requires the use of some Density Diffusion Term (DDT) to avoid fluid instabilities and **DensityDT=3 is strongly recommended**.

```
<execution>
    <parameters>
        <parameter key="Boundary" value="2" comment="Boundary method 1:DBC, 2:mDBC"/>
        <parameter key="DensityDT" value="3" comment="Density Diffusion Term 0:None,
            1:Molteni, 2:Fourtakas, 3:Fourtakas(full) (default=0)" />
```

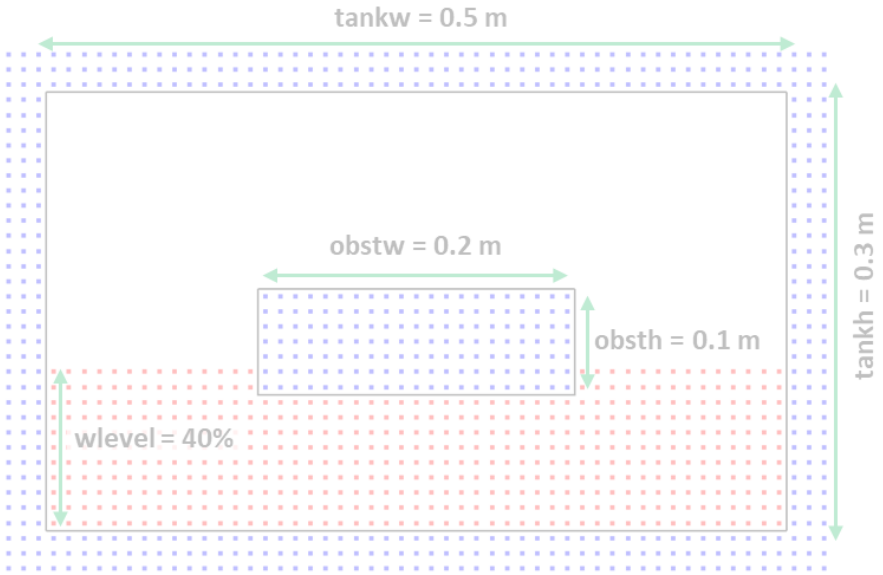
The **final normal vectors** of the boundary particles can be seen in the file ***CfgInit_Normals.vtk***. This VTK file is created by DualSPHysics when the simulation starts.



Two methods to generate normal vectors for mDBC

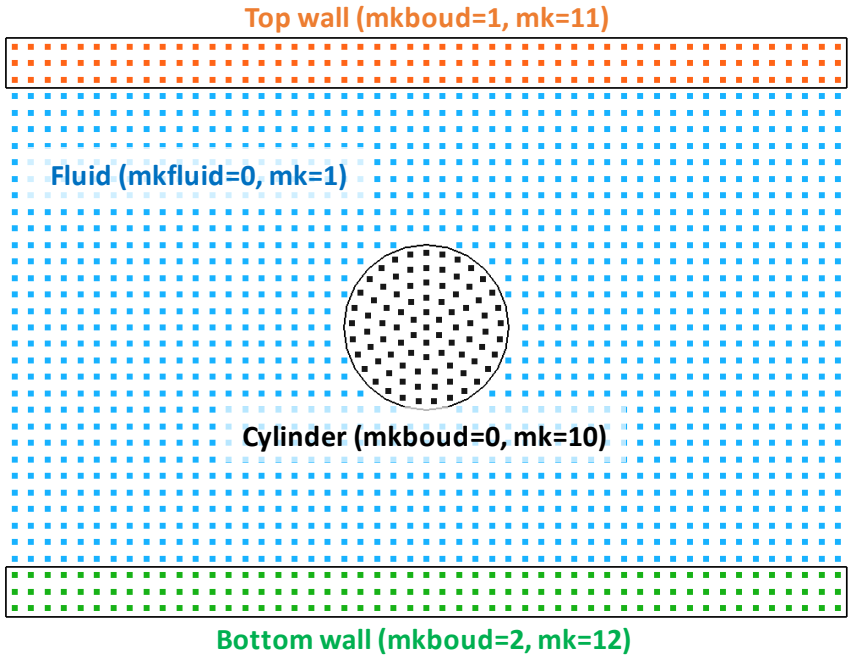
Method 1: Automatic normal vectors from actual geometry

Example: Sloshing tank with obstacle



Method 2: Explicit normal vectors definition in DualSPHysics configuration

Example: Flow past a circular cylinder



Method 2: Explicit normal vectors definition in DualSPHysics configuration

```
<casedef>
    <mkconfig boundcount="240" fluidcount="9" />
    <geometry>
        <definition dp="0.02">
            <pointmin x="-1.2" y="0" z="-2" />
            <pointmax x="2.2" y="0" z="2" />
        </definition>
        <commands>
            <mainlist>
                <!-- SIMULATION DIMENSIONS AND OTHER CONFIGURATIONS -->
                <newvarcte xmin="-0.5" xmax="0.5" _rem="Fluid domain in X" />
                <newvarcte zmin="-0.3" zmax="0.3" _rem="Fluid domain in Z" />
                <newvarcte cylx="0" cylz="0" cylradius="0.1" _rem="Cylinder values"/>
                <newvarcte fluidvel="1.0" _rem="Fluid velocity" />
                <exportvar vars="zmin,zmax,cyl*,fluidvel" />
                <setshapemode>actual | bound</setshapemode>
                <setdrawmode mode="full" />
                <!-- FLUID DOMAIN -->
                <setmkfluid mk="0" />
                <drawbox>
                    <boxfill>solid</boxfill>
                    <point x="#xmin" y="-0.1" z="#zmin" />
                    <endpoint x="#xmax" y="0.2" z="#zmax" />
                </drawbox>
                ...
                <!-- CYLINDER -->
                <setmkbound mk="0" />
                <setfrdrawmode auto="true" />
                <drawcylinder radius="#cylradius-Dp/2">
                    <point x="#cylx" y="0" z="#cylz" />
                    <point x="#cylx" y="0.1" z="#cylz" />
                </drawcylinder>
                <setfrdrawmode auto="false" />
                <!-- SOLID BOUNDARY: BOTTOM AND TOP -->
                <setmkbound mk="1" />
                <drawbox>
                    <boxfill>solid</boxfill>
                    <point x="#xmin" y="-0.1" z="#zmin-Dp*2" />
                    <endpoint x="#xmax" y="0.2" z="#zmin" />
                </drawbox>
                <setmkbound mk="2" />
                <drawbox>
                    <boxfill>solid</boxfill>
                    <point x="#xmin" y="-0.1" z="#zmax" />
                    <endpoint x="#xmax" y="0.2" z="#zmax+Dp*2" />
                </drawbox>
                <!-- END -->
                <shapeout file="" />
            </mainlist>
        </commands>
    </geometry>
    <initials>
        <velocity mkfluid="0" x="#fluidvel" y="0" z="0" units_comment="m/s" />
    </initials>
</casedef>
```

```
<execution>
    <special>
        <initialize>
            <boundnormal_cylinder mkbound="0">
                <centerl x="#cylx" y="-1" z="#cylz" />
                <center2 x="#cylx" y="1" z="#cylz" />
                <radius v="#cylradius" />
                <inside v="true" comment="Boundary particles inside the cylinder"/>
                <maxdisth v="2.0" comment="Maximum distance to boundary limit."/>
            </boundnormal_cylinder>
            <boundnormal_plane mkbound="1">
                <point x="0" y="0" z="#zmin+Dp/2" />
                <normal x="0" y="0" z="1" />
                <maxdisth v="2.0" comment="Maximum distance to boundary limit."/>
            </boundnormal_plane>
            <boundnormal_plane mkbound="2">
                <point auto="true" comment="Point is calculated automatically." />
                <normal x="0" y="0" z="-1" />
                <maxdisth v="2.0" comment="Maximum distance to boundary limit."/>
            </boundnormal_plane>
        </initialize>
    </special>
    <parameters>
        <parameter key="Boundary" value="2"
            comment="Boundary method 1:DBC, 2:mDBC (default=1)" />
        <parameter key="DensityDT" value="3"
            comment="Density Diffusion Term 0:None, 1:Molteni,
            2:Fourtakas, 3:Fourtakas(full) (default=0)" />
```

GenCase:

- Defines all dimensions and magnitudes necessary to design the case
- Creates boundary and fluid particles
- Initialize fluid velocity

DualSPHysics:

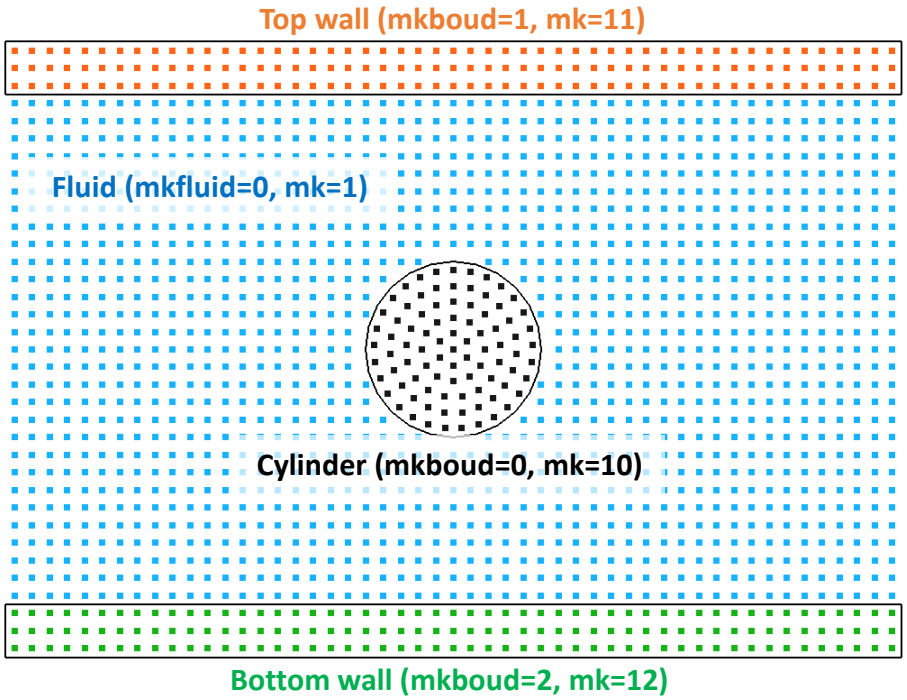
- Defines normal vectors for walls and cylinder
- Configures the use of mDBC and DDT=3

**Inlet/Outlet configuration was omitted here*

Method 2: Explicit normal vectors definition in DualSPHysics configuration

This method defines the normal vectors of boundary particles according to its Mk value in the **special section <initialize>**. This configuration is applied by DualSPHysics before starting the simulation and users can create new options for <initialize> since it is open-source.

```
<mainlist>
    <!-- SIMULATION DIMENSIONS AND OTHER CONFIGURATIONS -->
    <newvarcte xmin="-0.5" xmax="0.5" _rem="Fluid domain in X" />
    <newvarcte zmin="-0.3" zmax="0.3" _rem="Fluid domain in Z" />
    <newvarcte cylx="0" cylz="0" cylradius="0.1" _rem="Cylinder values"/>
    <newvarcte fluidvel="1.0" _rem="Fluid velocity" />
    <exportvar vars="zmin,zmax,cyl*,fluidvel" />
```



In the **FlowCylinder example** the initial particles are created in <mainlist> using *drawbox* and *drawcylinder* commands and normal vectors are defined in <execution> <initialize>.

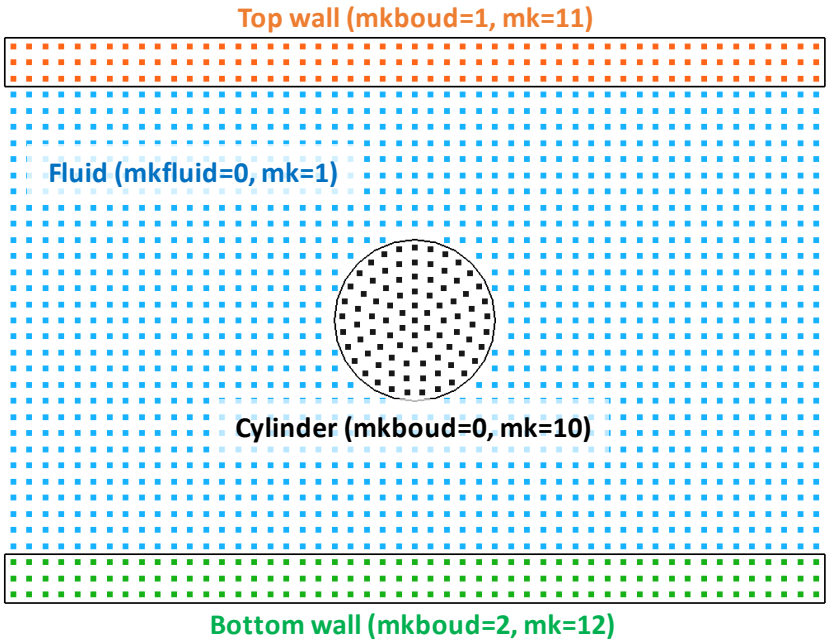
User-defined variables are used to define dimensions and magnitudes to design this case.

- Domain and walls: xmin, xmax, zmin, zmax.
- Cylinder: cylx, cylz, cylradius.

Method 2: Explicit normal vectors definition in DualSPHysics configuration

Drawing commands are used in <mainlist> to create fluid and boundary particles. The normal vectors for boundary particles are defined according to Mk. Thus each wall and body should be defined using a different Mk value.

```
<!-- BASIC CONFIGURATION -->
<setshapemode>actual | bound</setshapemode>
<setdrawmode mode="full" />
<!-- FLUID DOMAIN -->
<setmkfluid mk="0" />
<drawbox>
    <boxfill>solid</boxfill>
    <point x="#xmin" y="-0.1" z="#zmin" />
    <endpoint x="#xmax" y="0.2" z="#zmax" />
</drawbox>
...
<!-- CYLINDER -->
<setmkbound mk="0" />
<setfrdrawmode auto="true" />
<drawcylinder radius="#cylradius-Dp/2">
    <point x="#cylx" y="0" z="#cylz" />
    <point x="#cylx" y="0.1" z="#cylz" />
</drawcylinder>
<setfrdrawmode auto="false" />
<!-- SOLID BOUNDARY: BOTTOM AND TOP -->
<setmkbound mk="1" />
<drawbox>
    <boxfill>solid</boxfill>
    <point x="#xmin" y="-0.1" z="#zmin-Dp*2" />
    <endpoint x="#xmax" y="0.2" z="#zmin" />
</drawbox>
<setmkbound mk="2" />
<drawbox>
    <boxfill>solid</boxfill>
    <point x="#xmin" y="-0.1" z="#zmax" />
    <endpoint x="#xmax" y="0.2" z="#zmax+Dp*2" />
</drawbox>
```



Each set of particles using different mk value:

- Fluid particles with mkfluid=0 (mk=1)
- Cylinder as boundary particles with mkbound=0 (mk=10)
- Bottom wall as boundary particles with mkbound=1 (mk=11)
- Top wall as boundary particles with mkbound=2 (mk=12)

Method 2: Explicit normal vectors definition in DualSPHysics configuration

Final normal vector of each boundary particle is defined in the **special section <initialize>**. Each set of particles is configured according to its mk value. This normal vectors are calculated by DualSPHysics at the beginning of the simulation. Only normal vectors of particles configured in <initialize> are changed. The unmodified particles maintain their original normals, so **this normal definition method can be combined with the previous one**.

```
<execution>
    <special>
        <initialize>
            <!-- Defines cylinder normal vectors -->
            <boundnormal_cylinder mkbound="0">
                <center1 x="#cylx" y="-1" z="#cylz"/>
                <center2 x="#cylx" y="1" z="#cylz"/>
                <radius v="#cylradius"/>
                <inside v="true"/>
                <maxdisth v="2.0"/>
            </boundnormal_cylinder>

            <!-- Defines bottom wall normal vectors -->
            <boundnormal_plane mkbound="1">
                <point x="0" y="0" z="#zmin+Dp/2"/>
                <normal x="0" y="0" z="1"/>
                <maxdisth v="2.0"/>
            </boundnormal_plane>

            <!-- Defines top wall normal vectors -->
            <boundnormal_plane mkbound="2">
                <point auto="true" />
                <normal x="0" y="0" z="-1"/>
                <maxdisth v="2.0"/>
            </boundnormal_plane>

        </initialize>
    </special>
```

Cylinder normal vectors are defined according to central points and radius (***center1***, ***center2***, ***radius***).

Option ***inside*** allows normal for a solid cylinder (true) or a container cylinder (false).

Values ***maxdist*** define maximum distance from each boundary particle to boundary limit allowed to compute the normal.

Walls normal vectors are defined by one ***point*** in the boundary limit and one ***normal*** vector. All particles use the same normal direction, but size of normal vector depends on the distance between each particle and the boundary limit.

Normal vectors of bottom wall are **calculated automatically** according to the normal configuration and initial particles.

Method 2: Explicit normal vectors definition in DualSPHysics configuration

The **special section <initialize>** allows to define normal vectors using other options that were not used in this FlowCylinder example, which are the following:

- **Explicit normal vector** is assigned to all particles with the mkbound indicated. All boundary particles use the same normal vector (same direction and same size).

```
<special>
    <initialize>
        <boundnormal_set mkbound="1">
            <normal x="1" y="0" z="0" />
        </boundnormal_set>
```

- **Normal vectors for spheres.** ***Center*** and ***radius*** of sphere are required. Option ***inside*** allows normal for a solid sphere (true) or a container sphere (false). Value ***maxdist*** defines maximum distance from each boundary particle to boundary limit allowed to compute the normal vector.

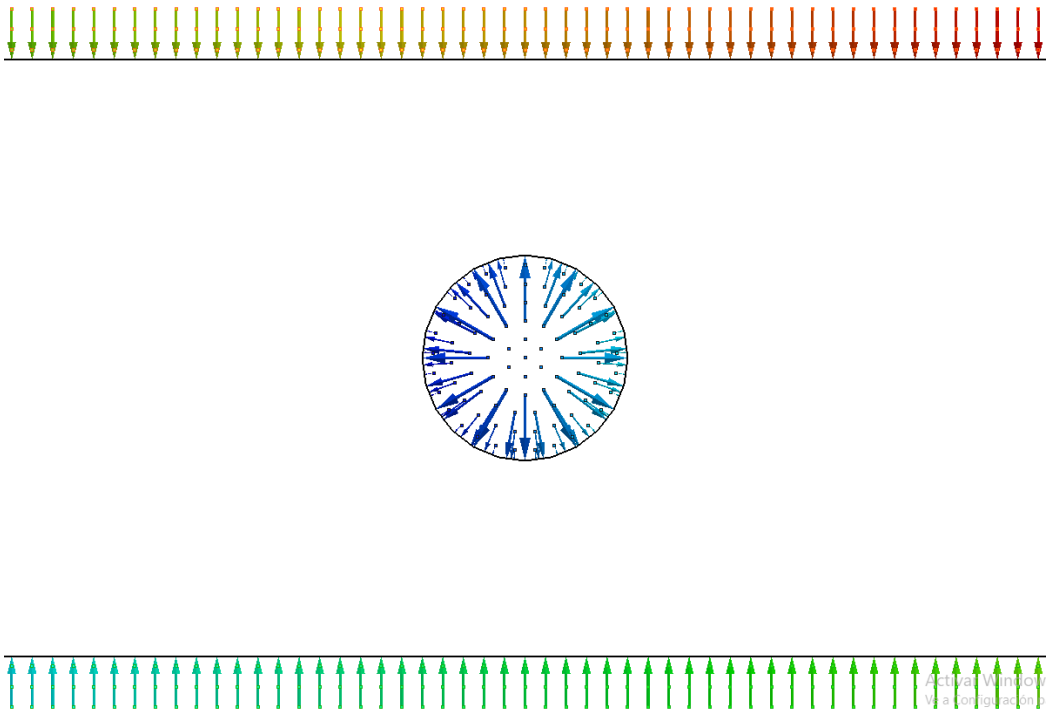
```
        <boundnormal_sphere mkbound="3">
            <center x="1" y="0" z="0" />
            <radius v="1" />
            <inside v="true" comment="Boundary particles inside the sphere" />
            <maxdisth v="2.0" comment="Maximum distance to boundary limit as H*maxdisth (default=2)"/>
        </boundnormal_sphere>
    </initialize>
</special>
```

Method 2: Explicit normal vectors definition in DualSPHysics configuration

As explained for the Method 1, the **use of mDBC** boundary condition is configured in execution parameters section. Simulation with mDBC requires the use of some Density Diffusion Term (DDT) and **DensityDT=3 is strongly recommended**.

```
<execution>
    <parameters>
        <parameter key="Boundary" value="2" comment="Boundary method 1:DBC, 2:mDBC"/>
        <parameter key="DensityDT" value="3" comment="Density Diffusion Term 0:None,
            1:Molteni, 2:Fourtakas, 3:Fourtakas(full) (default=0)" />
```

The **final normal vectors** of the boundary particles can be seen in the file ***CfgInit_Normals.vtk***. This VTK file is created by DualSPHysics when the simulation starts.



Slip modes with mDBC

Three different slip modes are being implemented:

- Zero velocity imposed to boundary particles
- No-slip condition
- Free slip condition

The label ***SlipMode*** will allow us to choose one of those options:

```
<parameter key="SlipMode" value="1" comment="Slip mode for mDBC 1:DBC vel=0, 2:No-slip, 3:Free slip" />
```

Let us assume a boundary particle b, with a ghost node g and the wall is moving with velocity u_w

SlipMode=1: DBC v=0

Only **SlipMode=1** is available in **v5**

This option is the same we already implemented with DBC so we impose v=0 to boundary particles

$$u_b = u_w$$

SlipMode=2: **No-slip**

We impose to boundary particles the interpolated velocity at the mirroring point inside the fluid field but with opposite value

$$u_b = 2*u_w - u_g$$

SlipMode=3: Free-slip

We impose to boundary particles the interpolated velocity at the mirroring point inside the fluid field but with same value

$$u_b = u_g$$