Inlets, outlets, and post-processing for modelling open-channel flow with the volume of fluid method

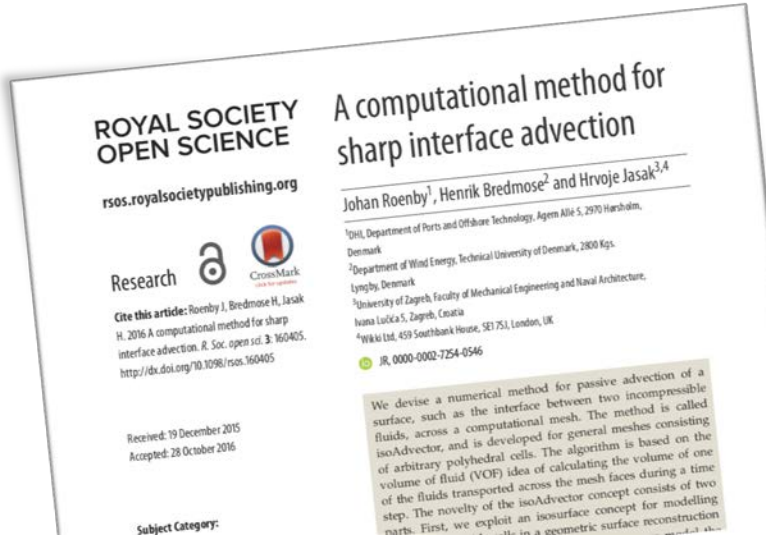
Shannon Leakey
School of Engineering
Newcastle University

Traditional hydraulic modelling (p.4)

- Shallow water equations (1D/2D)
- Depth-averaged velocity
- Subcritical & supercritical
- Upstream & downstream BCs

Volume of fluid method (pp.5-6)

- Phase fraction
- interFoam
- interIsoFoam



ROYAL SOCIETY OPEN SCIENCE

rsos.royalsocietypublishing.org

Research

Cite this article: Roenby J, Bredmose H, Jasak H. 2016 A computational method for sharp interface advection. *R. Soc. open sci.* **3**: 160405. http://dx.doi.org/10.1098/rsos.160405

Received: 19 December 2015
Accepted: 28 October 2016

Subject Category:

A computational method for sharp interface advection

Johan Roenby[1], Henrik Bredmose[2] and Hrvoje Jasak[3,4]

[1]DHI, Department of Ports and Offshore Technology, Agern Allé 5, 2970 Hørsholm, Denmark
[2]Department of Wind Energy, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark
[3]University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture, Ivana Lučića 5, Zagreb, Croatia
[4]Wikki Ltd, 459 Southbank House, SE1 7SJ, London, UK

JR, 0000-0002-7254-0546

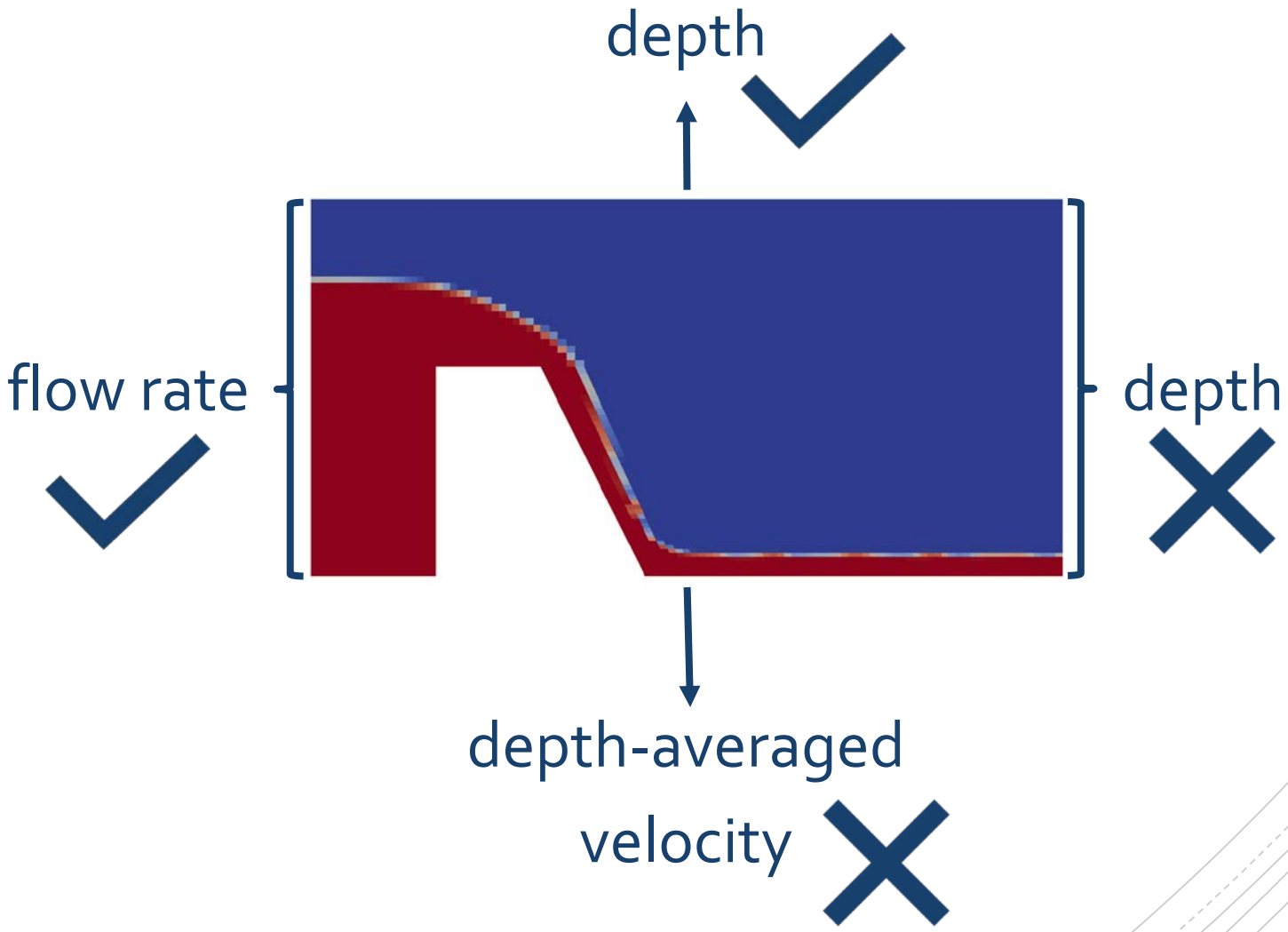
We devise a numerical method for passive advection of a surface, such as the interface between two incompressible fluids, across a computational mesh. The method is called isoAdvector, and is developed for general meshes consisting of arbitrary polyhedral cells. The algorithm is based on the volume of fluid (VOF) idea of calculating the volume of one of the fluids transported across the mesh faces during a time step. The novelty of the isoAdvector concept consists of two parts. First, we exploit an isosurface concept for modelling

The problem
(p.4)

What if …

- you want to compare VOF results with hydraulic modelling results?
- you want your VOF BCs to match up with your hydraulic BCs?

The problem
(pp.4,6-12)

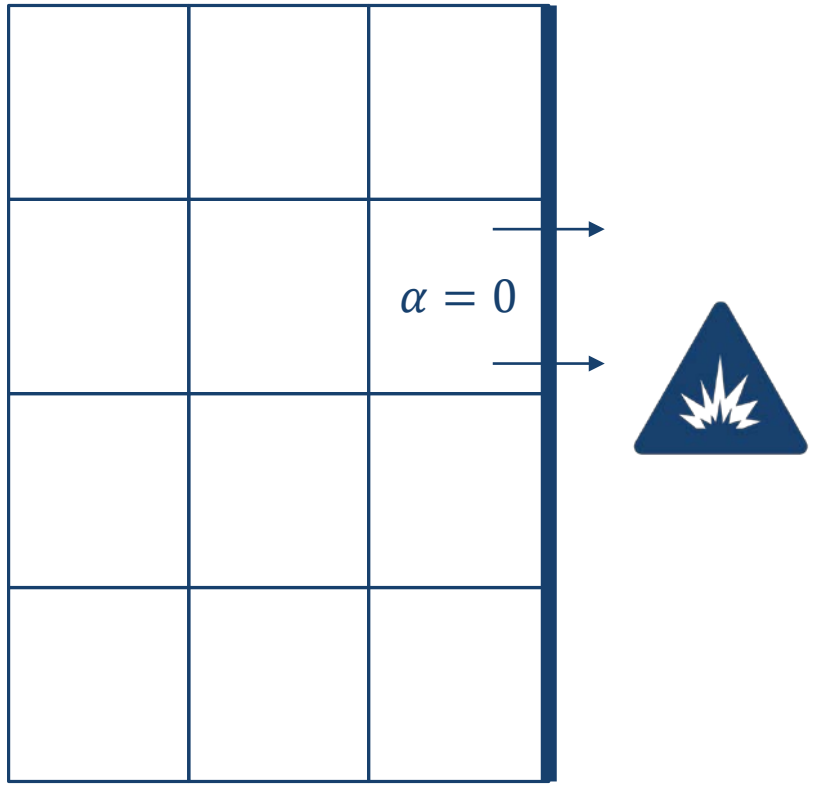


The general idea for depthOutlet

Creating
depthOutlet

Which field?

- α ✗
- $\vec{u}$ ✓



Creating depthOutlet**Velocity**

- Calculate flow rate Q from α and $\vec{u}$
- Calculate cross-sectional area A from α
- Calculate average velocity $V = Q/A$

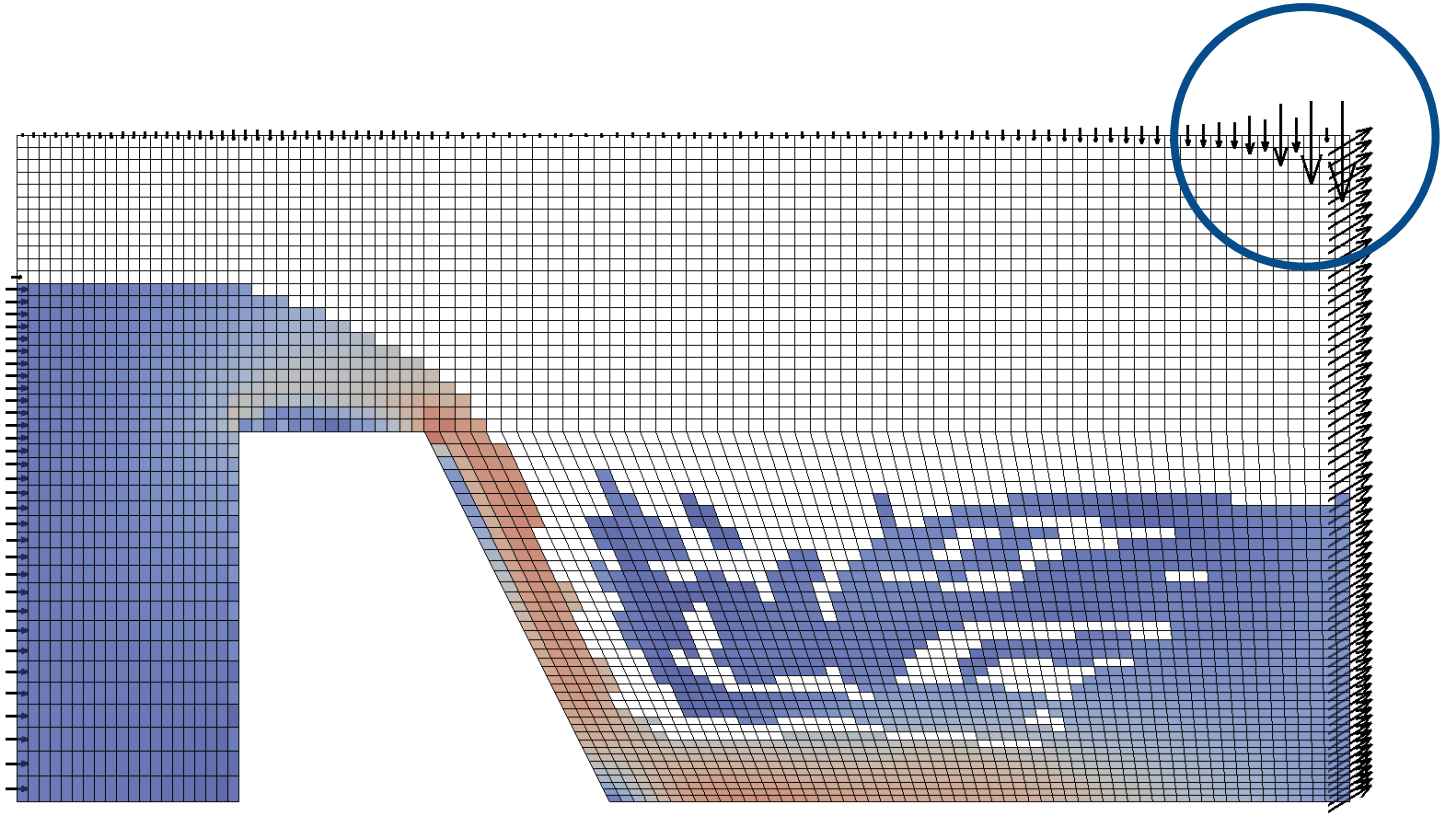
Creating depthOutlet**Velocity**

Which cells to set this average velocity…

- All of them?
- Those below depth?
- Something else?

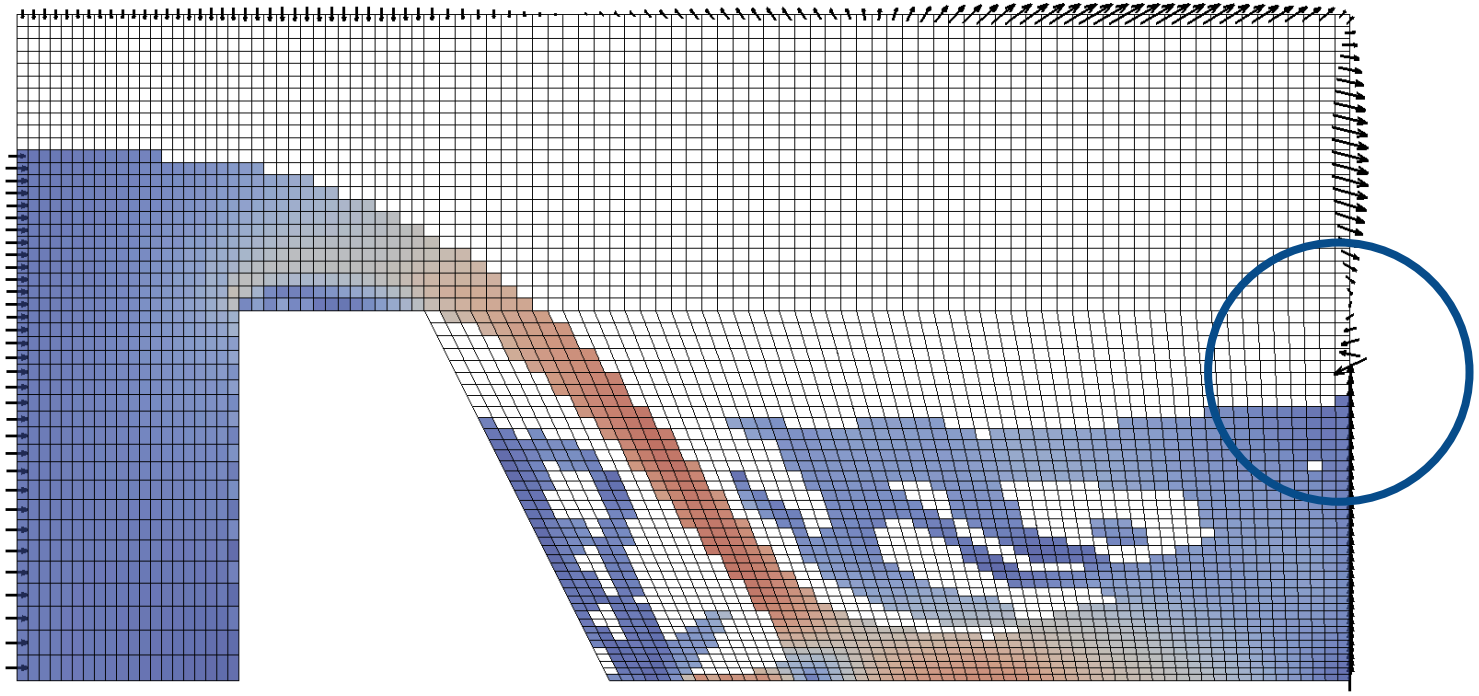
Applied across whole patch

air too fast

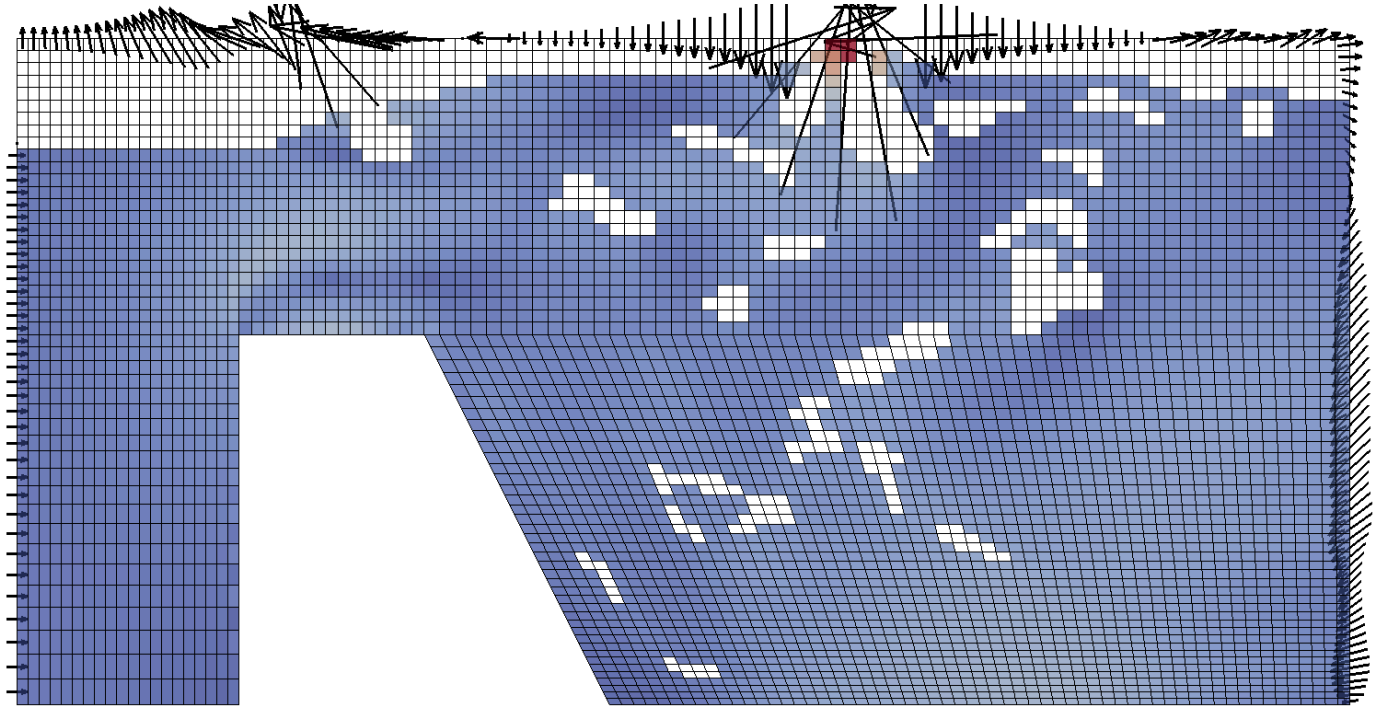


Applied below depth

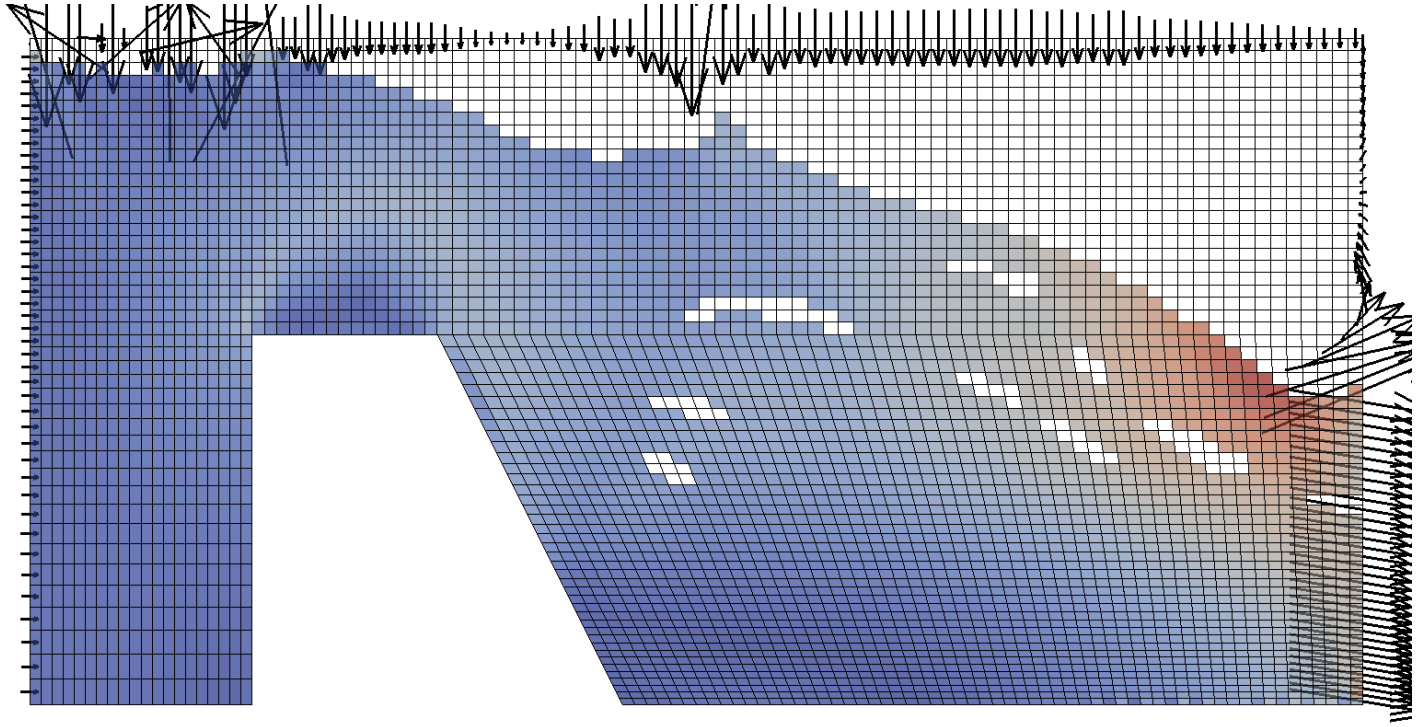
seems to work



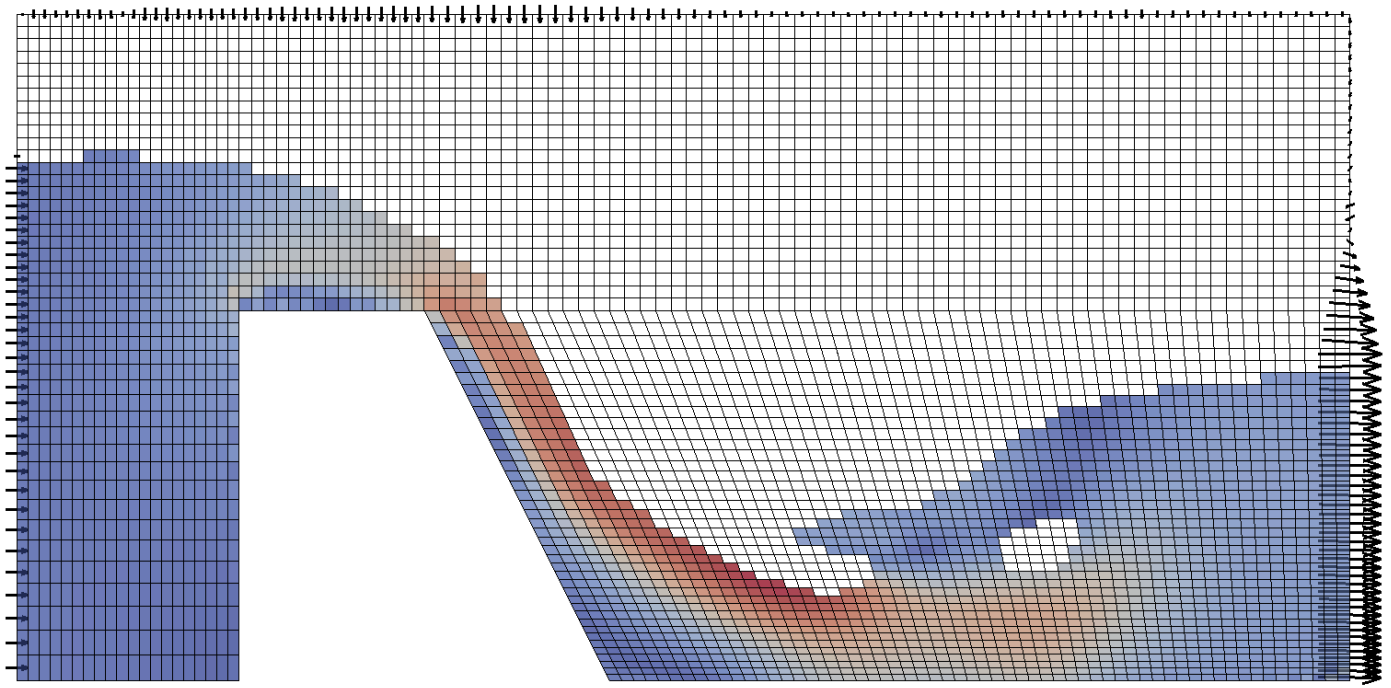
Applied below depth
then fills up



Applied below depth
then empties

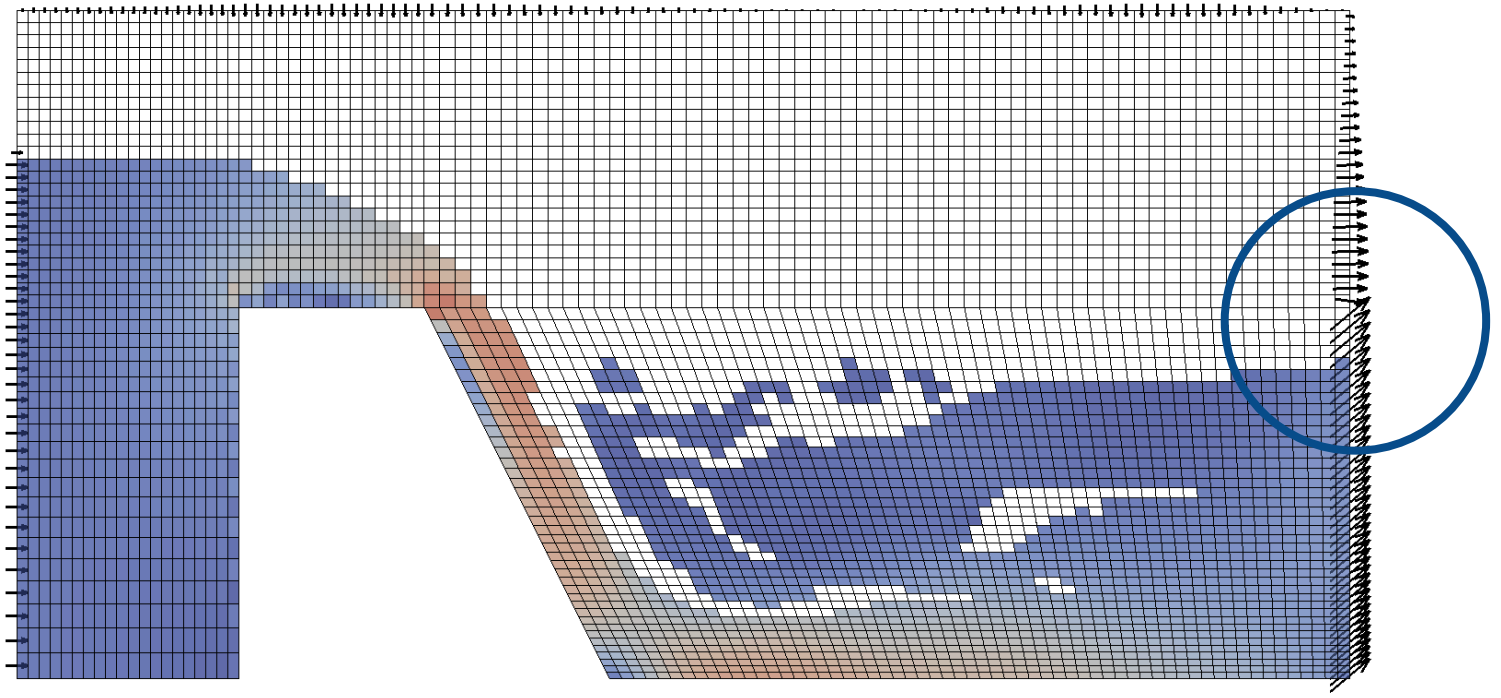


Applied below depth
then settles out



Applied below depth + buffer

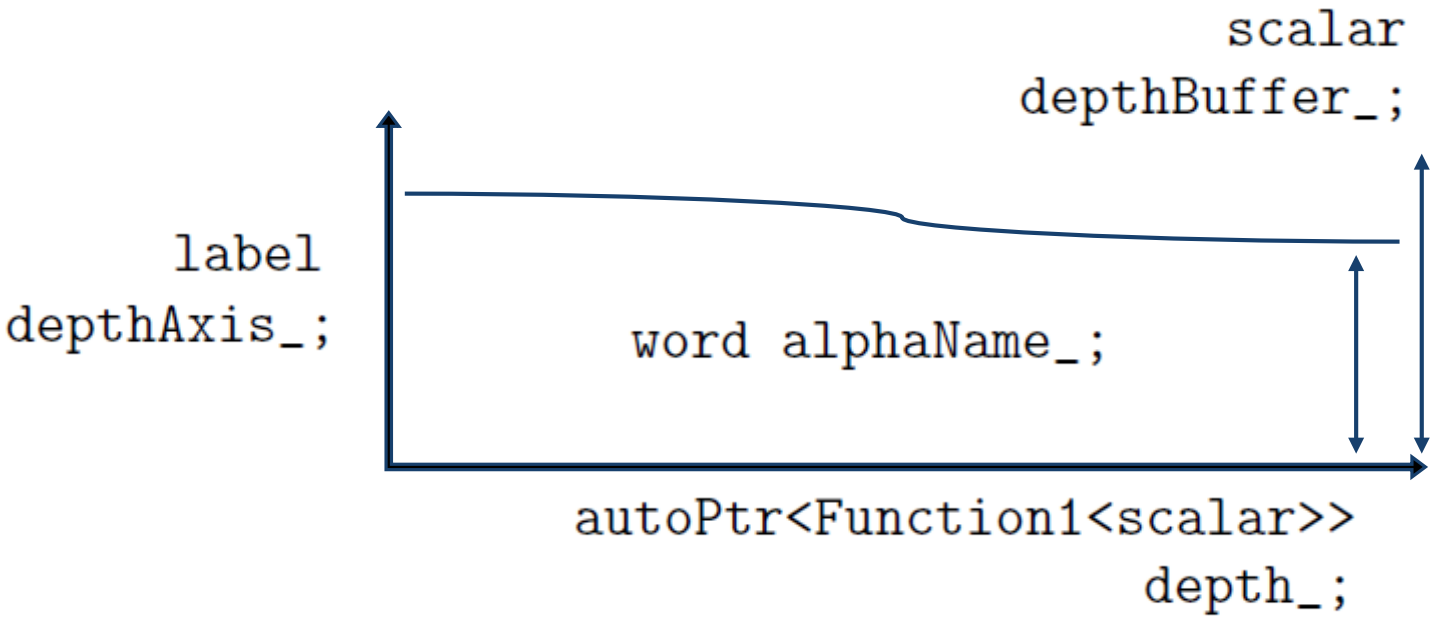
works from start



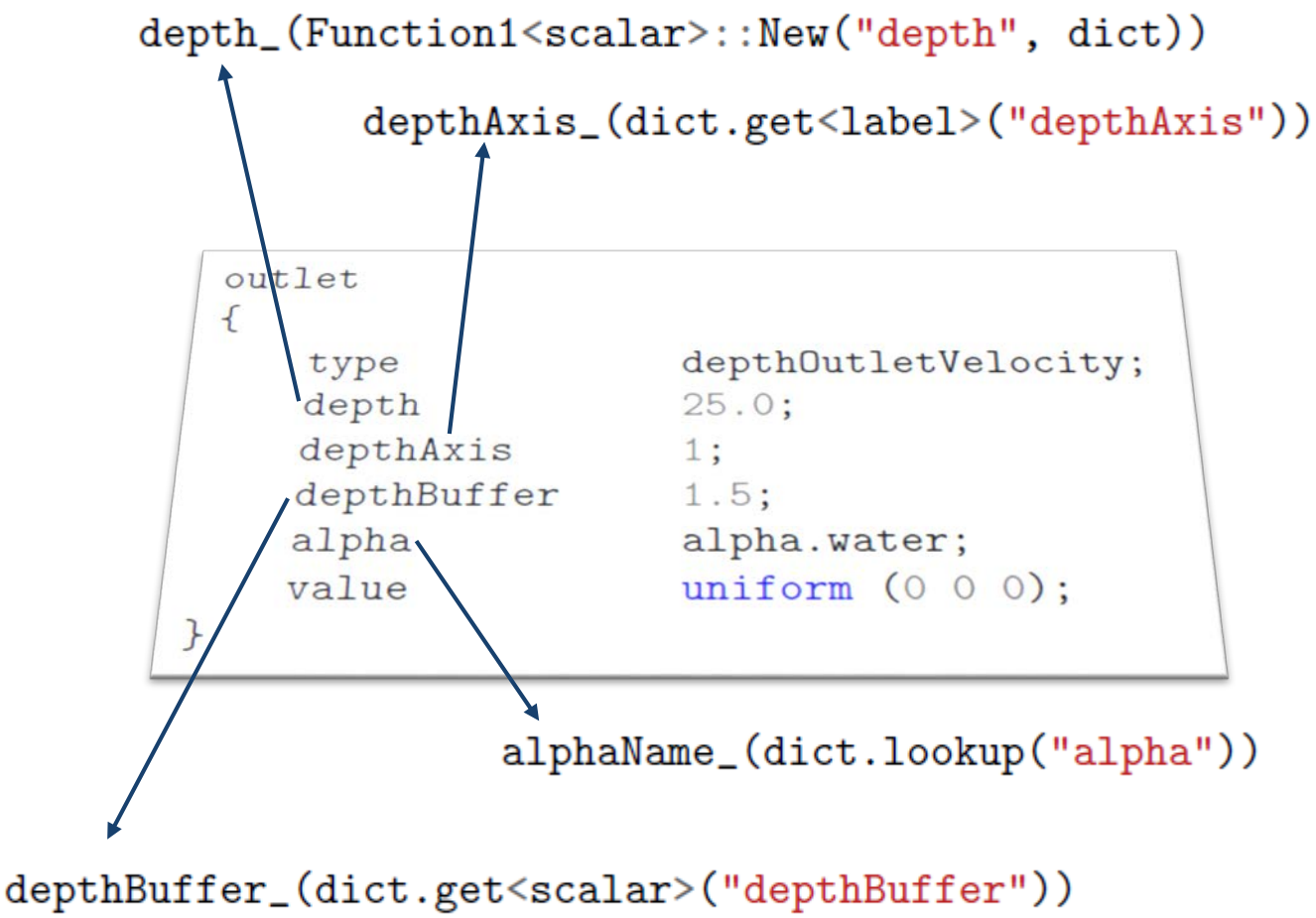
The code in pictures for depthOutletVelocity

(pp.13-18)

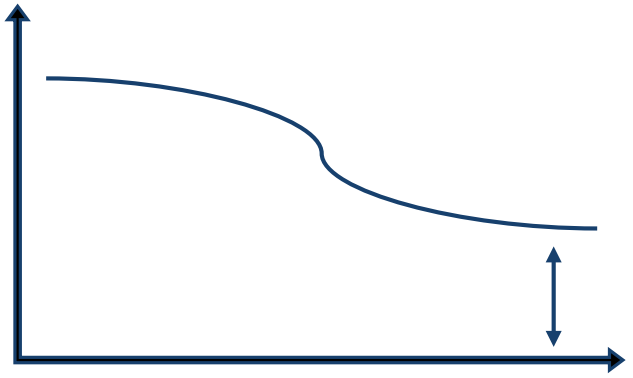
Declaration
.H

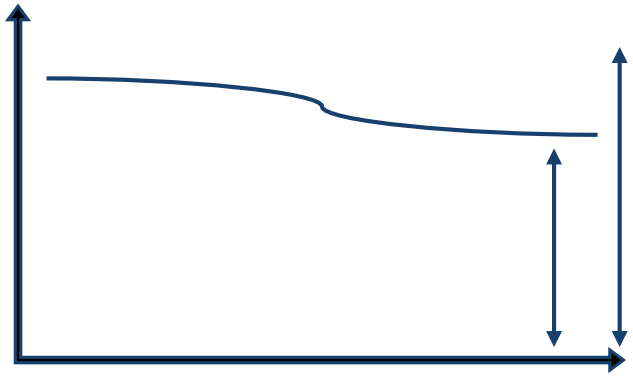


Definition
.C



Fields
.C





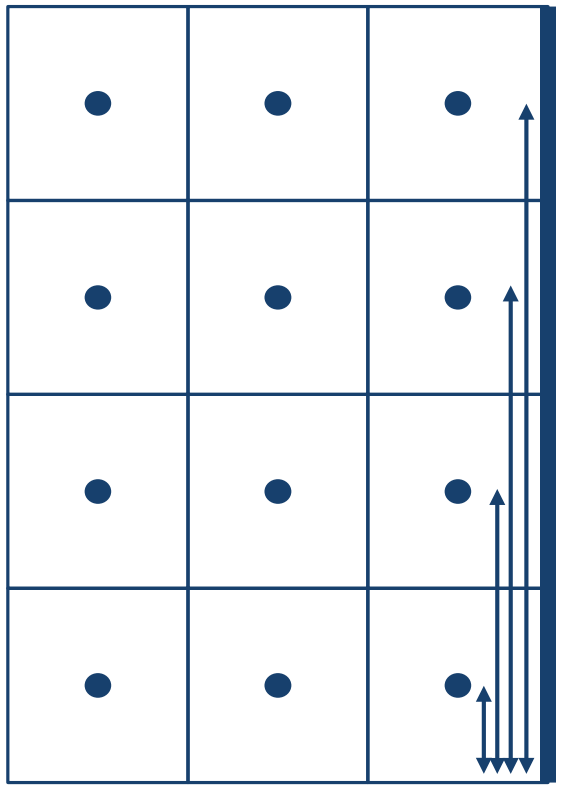
```
scalarField alphaOld =
    patch().lookupPatchField
    <volScalarField, scalar>(alphaName_);

alphaOld = max(alphaOld, scalar(0));
alphaOld = min(alphaOld, scalar(1));

vectorField velocityOld(this->patchInternalField());
```

```
scalarField alphaNew(patch().size(),0);
scalarField alphaBuffer(patch().size(),0);
```

Geometry .C

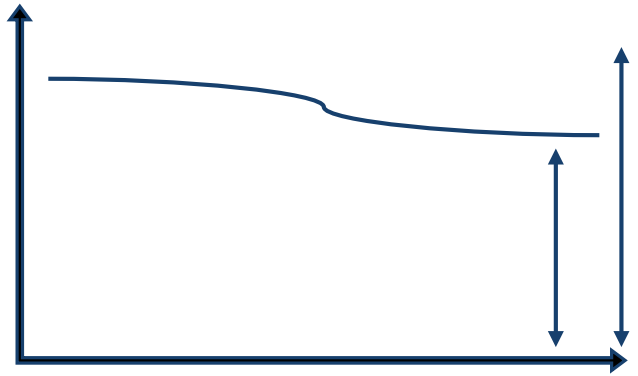


```
const scalarField centresAxis =
      centres.component(depthAxis_);
```

```
boundBox bb(patch().patch().localPoints(), true);
scalar datum = bb.min().component(depthAxis_);
```

Marking cells .C

```
forAll(centresAxis, i)
{
    if ((centresAxis[i] - datum) < depth)
    {
        alphaNew[i] = 1.0;
        alphaBuffer[i] = 1.0;
    }
    else if ((centresAxis[i] - datum) < depthBuffer_*depth)
    {
        alphaNew[i] = 0.0;
        alphaBuffer[i] = 1.0;
    }
    else
    {
        alphaNew[i] = 0.0;
        alphaBuffer[i] = 0.0;
    }
}
```



Applying the velocity .C

```
scalar areaNew = gSum(alphaNew*areas);

vector discharge = gSum(alphaOld*velocityOld*areas);
vector velocityNew = discharge/areaNew;

if (gSum(discharge & normals) <= 0)
{
    operator==(velocityOld);
}
else
{
    operator==
    (
        velocityNew*alphaBuffer +
        velocityOld*(1-alphaBuffer)

    );
}
```

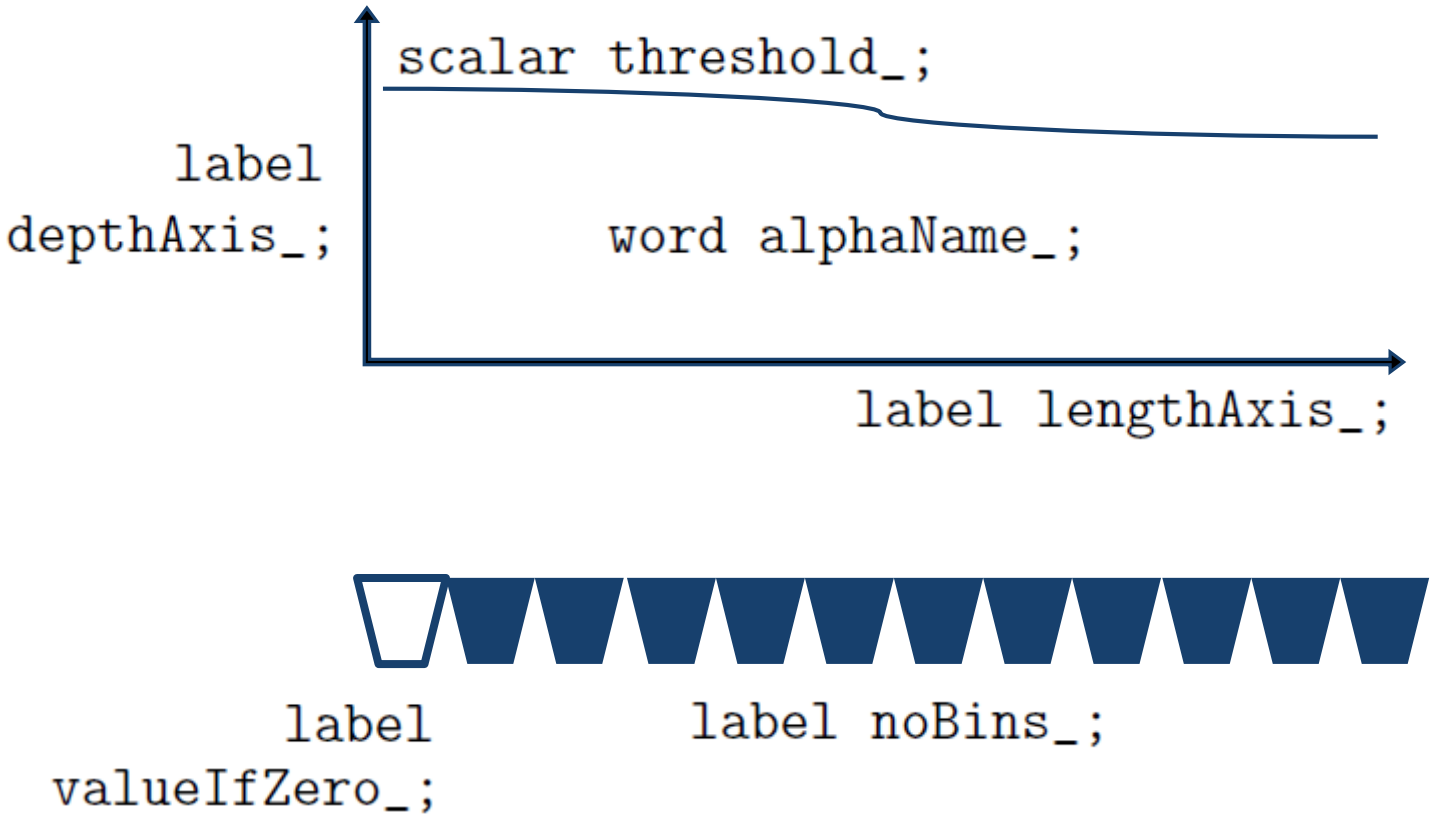
Does it work in general cases?
sometimes

- Tested how general it was by translating and rotating the mesh first
- Works if the mesh aligns (or nearly aligns) with Cartesian coordinates
- Could be more general but there were problems if the velocity was applied only normal to the patch

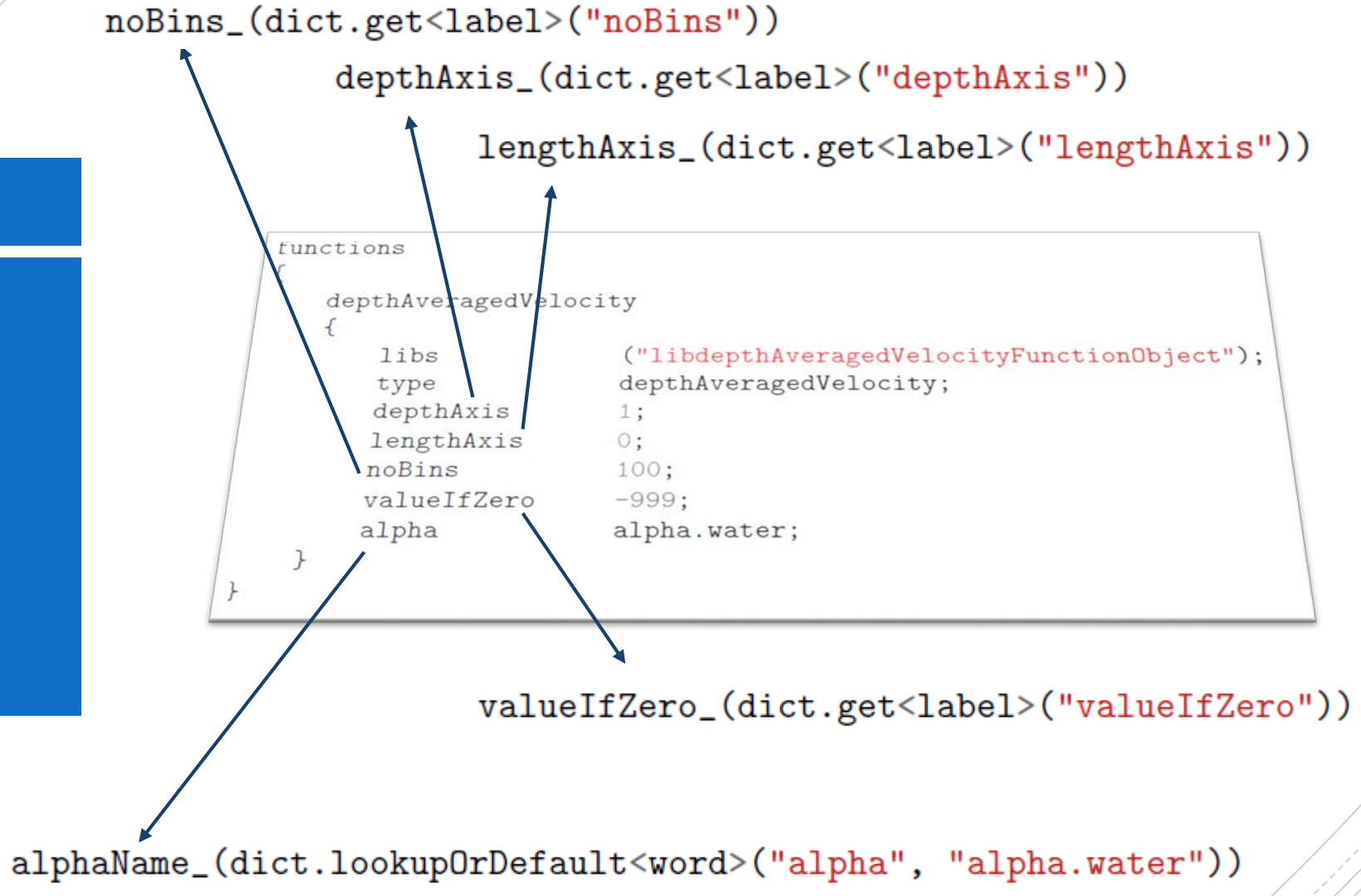
Also a function object for depth-averaged velocity

(pp.19-22)

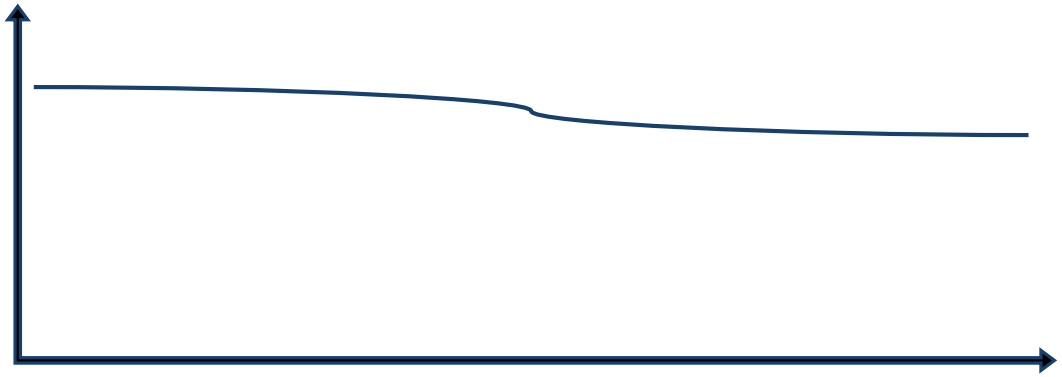
Declaration
.H



Definition
.C



Geometry .C



```
scalar longMin =
meshBounds.min().
component(lengthAxis_);
```

```
scalar longMax =
meshBounds.max().
component(lengthAxis_);
```

```
scalar length = longMax - longMin;
```

```
scalar dx = length/noBins_;
```

Looping over the bins .C

```
for (int n = 0; n < noBins_; n++)
{
    scalar lowerBound = longMin + dx*n;
    scalar midpoint = lowerBound + dx/2;
    scalar upperBound = longMin + dx*(n+1);

}
```

Looping over the bins .C

```
for (int n = 0; n < noBins_; n++)
{
        scalar sum = 0;
        scalar totVol = 0;

        forAll (centres, i)
        {
            scalar centresAxis = centres[i].component(lengthAxis_);

            bool centreInInterval =
                    centresAxis >= lowerBound &&
                    centresAxis < upperBound;

            if (centreInInterval && alpha[i] > threshold_)
            {
                scalar UAxis = U[i].component(lengthAxis_);
                scalar vol = mesh_.V()[i];

                sum += UAxis*vol;
                totVol += vol;
            }
        }

        reduce(sum, sumOp<scalar>());
        reduce(totVol, sumOp<scalar>());

}
```

Looping over the bins .C

```
for (int n = 0; n < noBins_; n++)
{


        scalar avg;

        if (totVol < 1e-12)
        {
            avg = valueIfZero_;
        }
        else
        {
            avg = sum/totVol;
        }


}
```

Questions?

Shannon Leakey
School of Engineering
Newcastle University