11th OpenFOAM Workshop, Guimarães, Portugal, 26-30th June, 2016

A Combined Euler-Euler Euler-Lagrange Slurry Model

Alasdair Mackenzie*, MT Stickland, WM Dempster

Weir Advanced Research Centre, University of Strathclyde, Glasgow, Scotland

Background

University of Strathclyde Glasgow

- Weir group produce equipment for the mining and oil and gas industries

- Erosion is a large problem

- Use CFD modelling to predict erosion = better designs

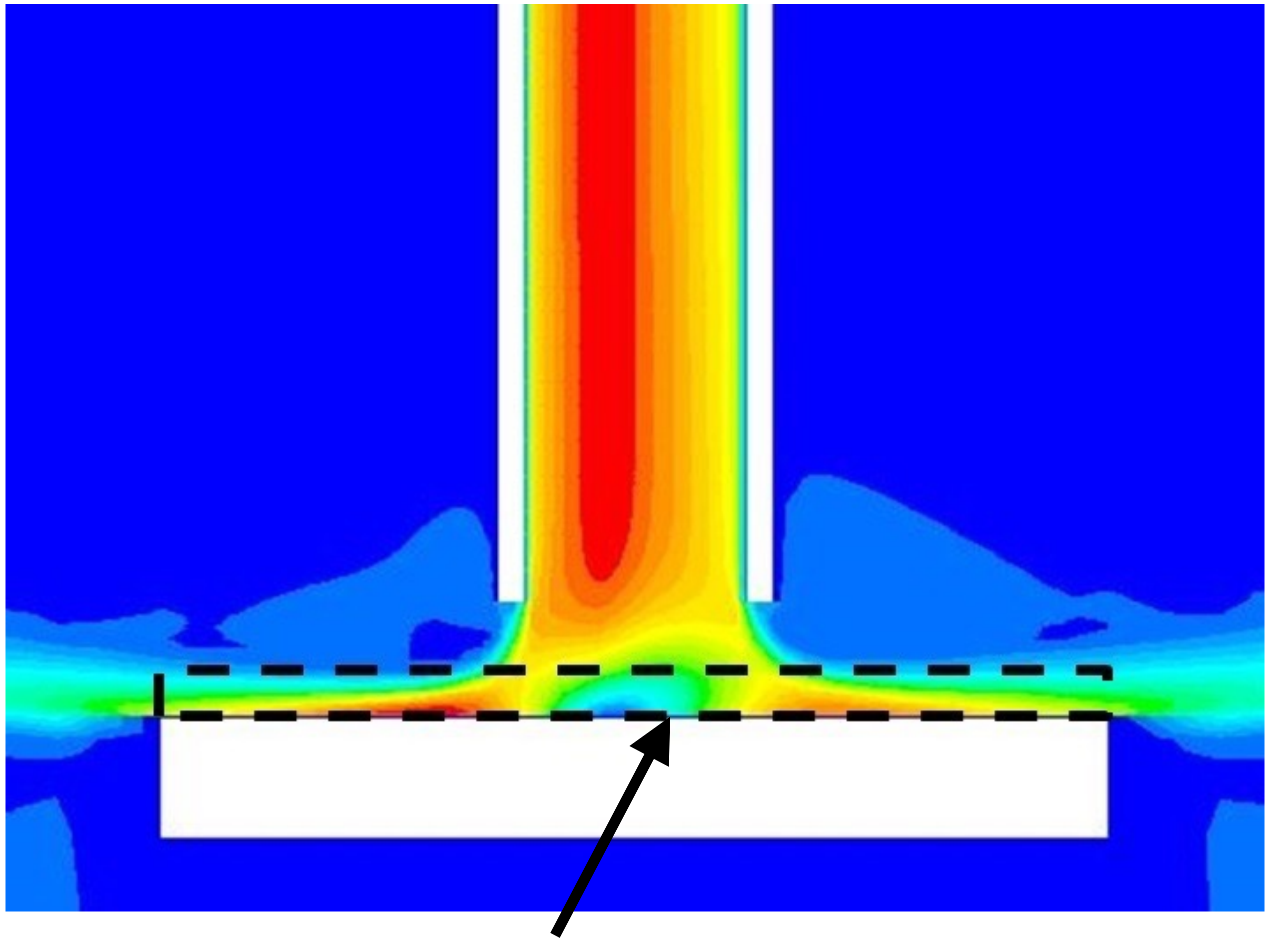
- Longer pump life, better for customer

WEIR Advanced Research Centre

Problem/Motivation

- Need particle impact data at the wall for erosion modelling
- Fluid/particulate flow simulation is computationally expensive: especially for dense slurries
- Solution: Combine with two-fluid model

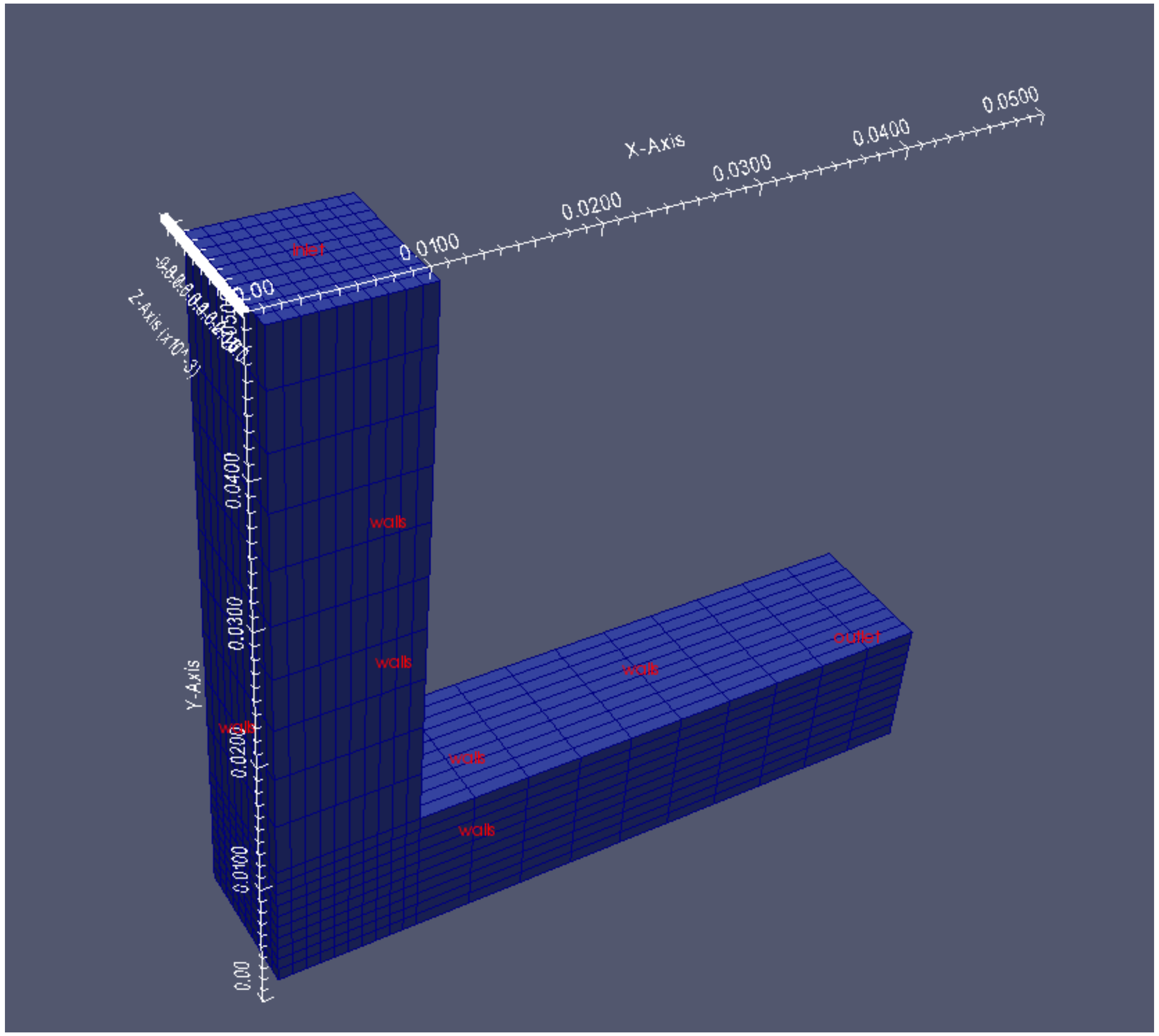
Velocity contours of submerged jet impingement test



Dotted region where particles are necessary for impact data

Geometry and Solvers

- A simple geometry was chosen for solver development
- reactingTwoPhaseEulerFoam for Euler-Euler
- DPMFoam for Euler-Lagrange
- OpenFOAM 3.0.x was used
- Started course in Chalmers University:



http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/

Geometry shown with sizes in metres

Description of Solvers

<u>reactingTwoPhaseEulerFoam</u>

Euler-Euler

Two fluid model

Both phases treated as continuum.

Incompressible model: setting in dictionary

Fast to solve

<u>DPMFoam</u>

Euler-Lagrange

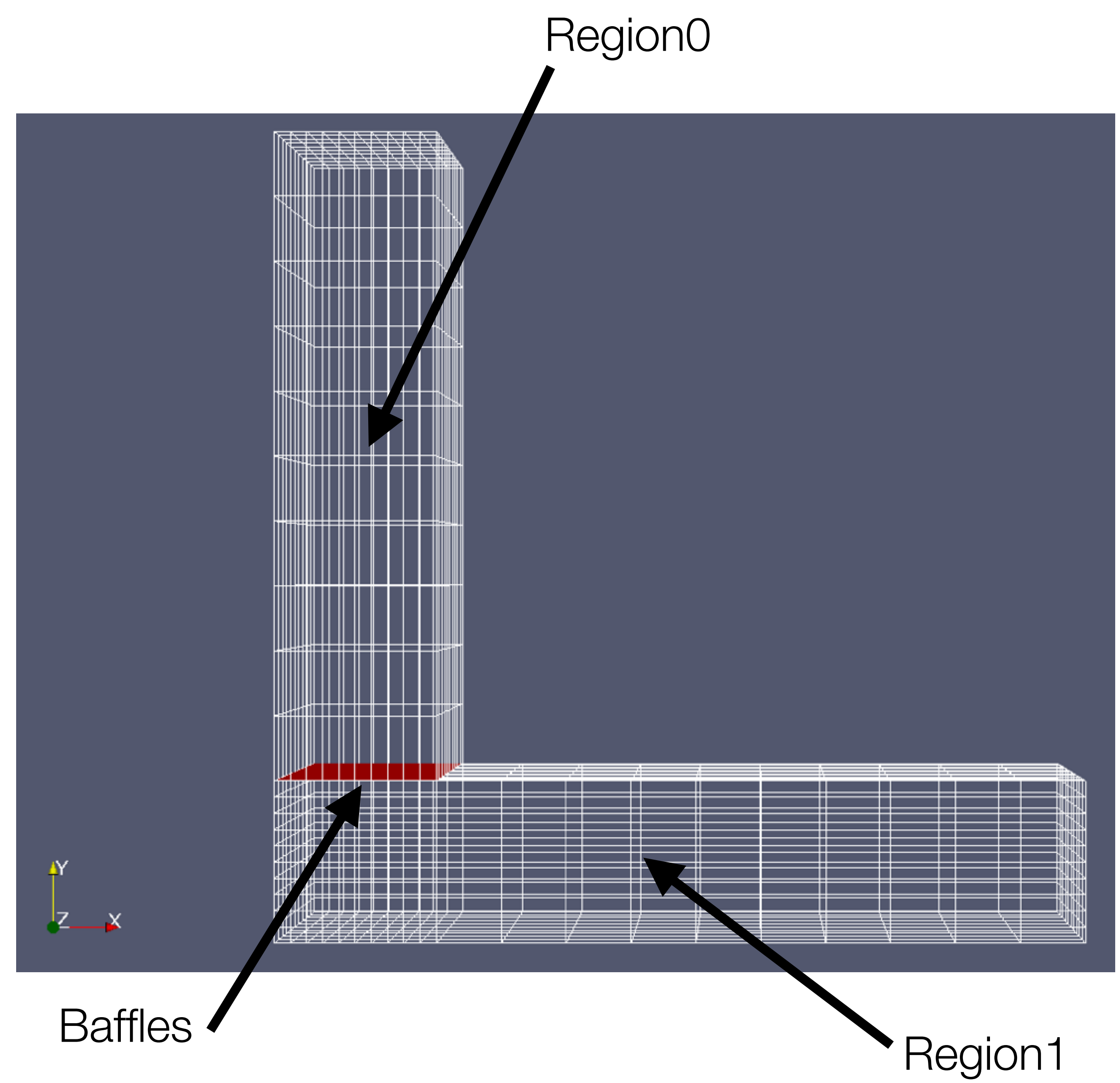
Fluid/particle model

Transient solver for coupled transport of kinematic particle clouds

Includes the effect of volume fraction of the particles on the continuous phase

Baffles + Regions

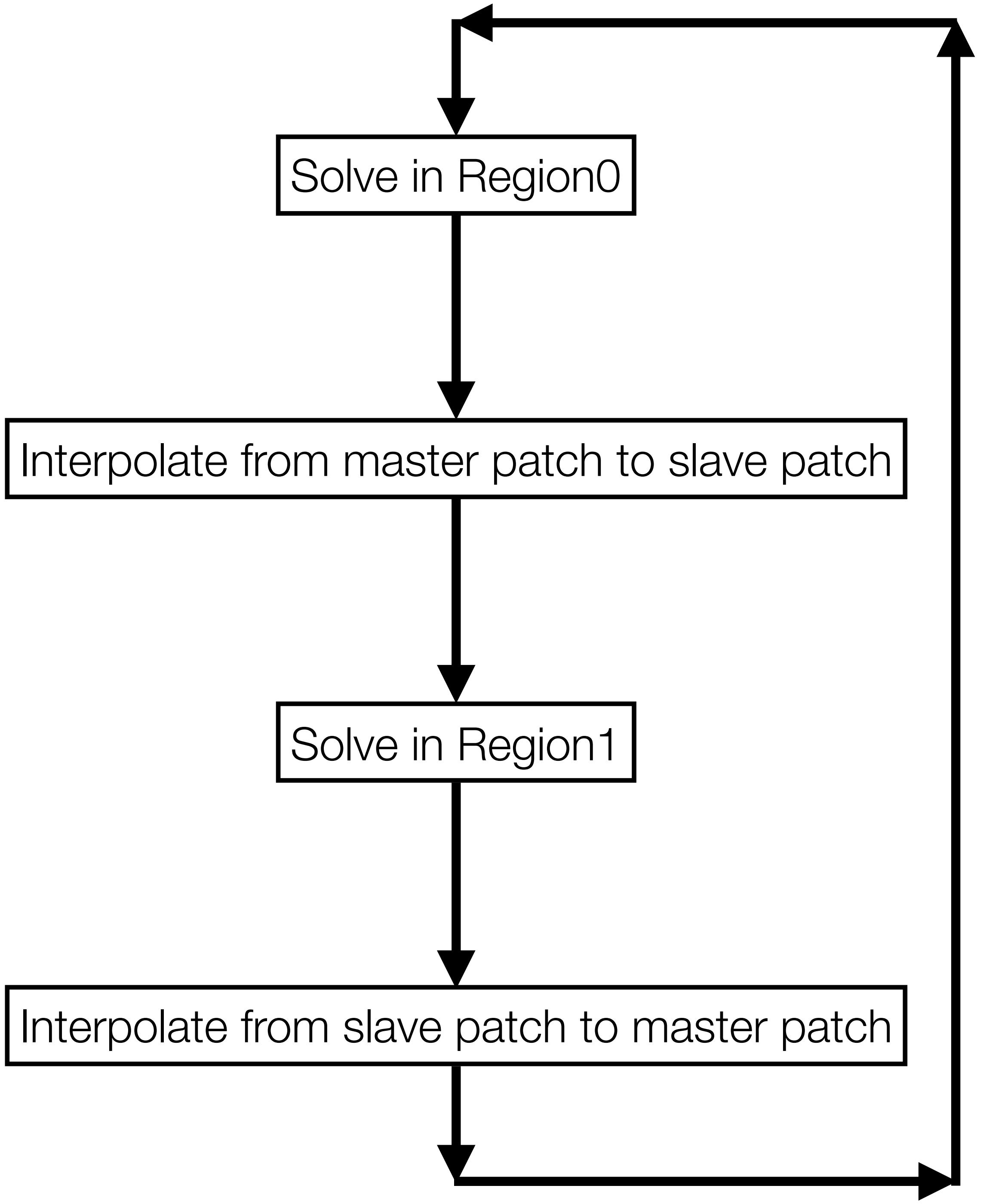
- Surface created where transition to take place
- createBaffles: makes internal surface into boundary face
- ‘master’ and ‘slave’ patch created
- splitMeshRegions: Splits mesh into 2 separate regions
- BC’s can now be applied to surface



Interpolation

- chtMultiRegionFoam: Inspiration for solving regions sequentially
- patchToPatchInterpolation: transfers data between two patches
- Variables interpolated are: U1, U2, p, p_rgh, alpha1, alpha2, k, epsilon, nut, and theta
- After this is implemented, the domain runs as if it was one region, not two: the surface doesn't affect the flow
- 'back pressures' are taken into account by interpolating upstream

Iterative loop



DPMFoam added

- Code from DPMFoam was added to new solver.
- Particles injected from slave patch after back interpolation (slave to master)
- Particles are only in region1 (near wall)
- Injection values based on phase 2 from region0 by using a lookup table

DPMFoam injection

```
/* (x y z) (u v w) d rho mDot numParcels
     where:
   x, y, z = global cartesian co-ordinates [m]
   u, v, w = global cartesian velocity components [m/s]
   d       = diameter [m]
   rho     = density [kg/m3]
   mDot    = mass flow rate [kg/m3]
   numParcels = number of Parcels
   Dictionary for the KinematicLookupTableInjection */
(
(0.0005 0.01 -0.0005) (0.01417 0.01831 -0.001718) 5.5e-05 2750 0.005 -2
(0.0015 0.01 -0.0005) (0.06206 -0.1608 -0.001616) 5.5e-05 2750 0.005 10
(0.0025 0.01 -0.0005) (0.1088 -0.3422 -0.0005019) 5.5e-05 2750 0.005 19
(0.0035 0.01 -0.0005) (0.1497 -0.4695 -0.001312) 5.5e-05 2750 0.005 24
```

- Modified kinematicLookupTableInjection used to inject particles
- Lookup table is updated every time step (but not read every time step!)
- 1 line = 1 cell (100 cells in this case)
- Values for particle injection are based on new updated values so solver can deal with geometry changes etc. See Lopez' presentation for more details:

https://sourceforge.net/projects/openfoam-extend/files/OpenFOAM_Workshops/OFW10_2015_AnnArbor/Presentations/Lopez-present-OFW10-16.pdf/download

DPMFoam injection

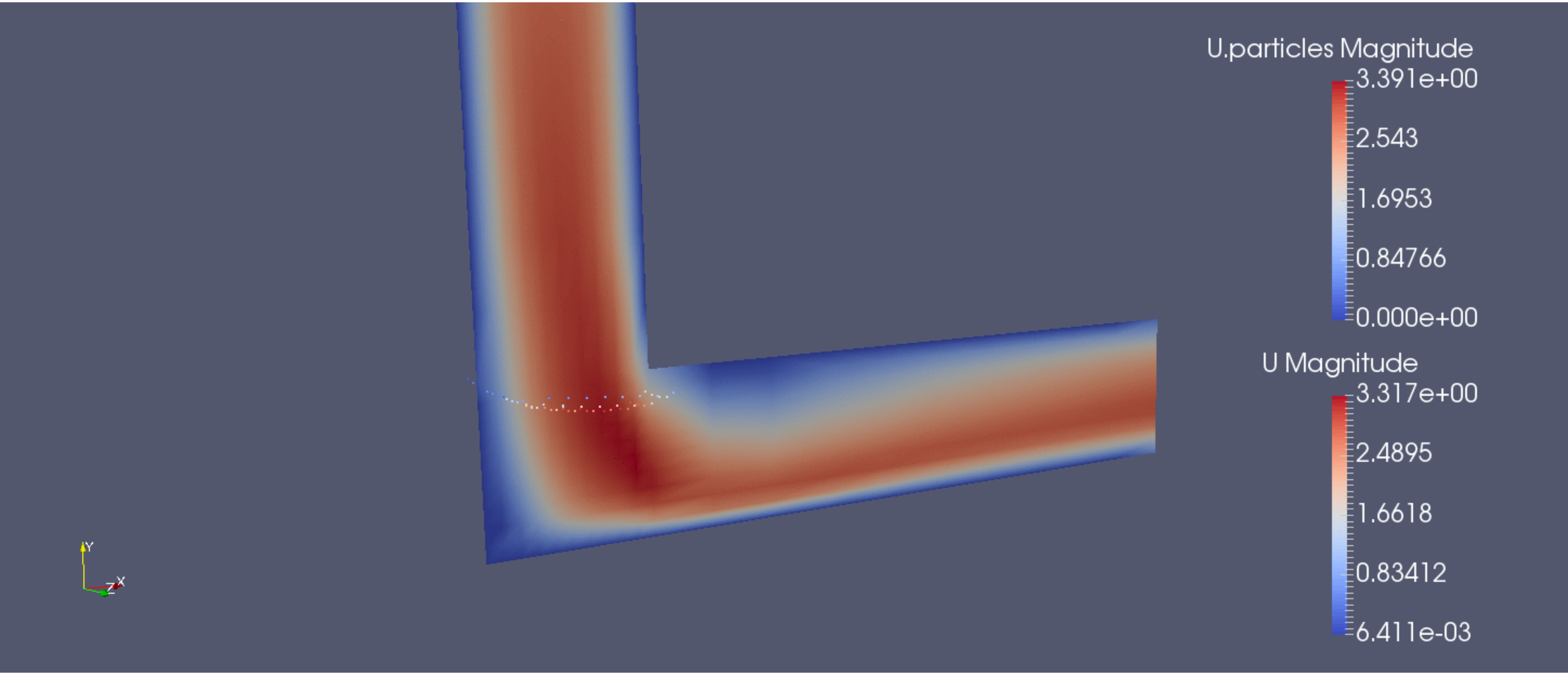
```
    os <<"/* (x y z) (u v w) d rho mDot numParcels  \n";
    os <<"     where: \n";
    os <<"   x, y, z = global cartesian co-ordinates [m] \n";
    os <<"   u, v, w = global cartesian velocity components [m/s] \n";
    os <<"   d        = diameter [m] \n";
    os <<"   rho      = density [kg/m3] \n";
    os <<"   mDot     = mass flow rate [kg/m3] \n";
    os <<"   numParcels = number of Parcels  \n";

    os <<"   Dictionary for the KinematicLookupTableInjection */ \n";
    os << "(  " << endl;
       forAll(interpolatedInletU1, i)
           {
               os << centres[i] << " " << interpolatedInletU1[i] << " " << 55e-6 << " " << 2750 << " " << 0.005 << " " << floor ((alpha1[i]*(mag(normalSlaveVector[i]))*uNormal[i])/((8.71e-14)*3*(-1)*5000)) << endl;
           }
    os << "//The end"<< endl;
    os << ");"<< endl;
```

- Number of parcels to be injected is calculated from volume flow rate, number of particles/parcel and alpha distribution.

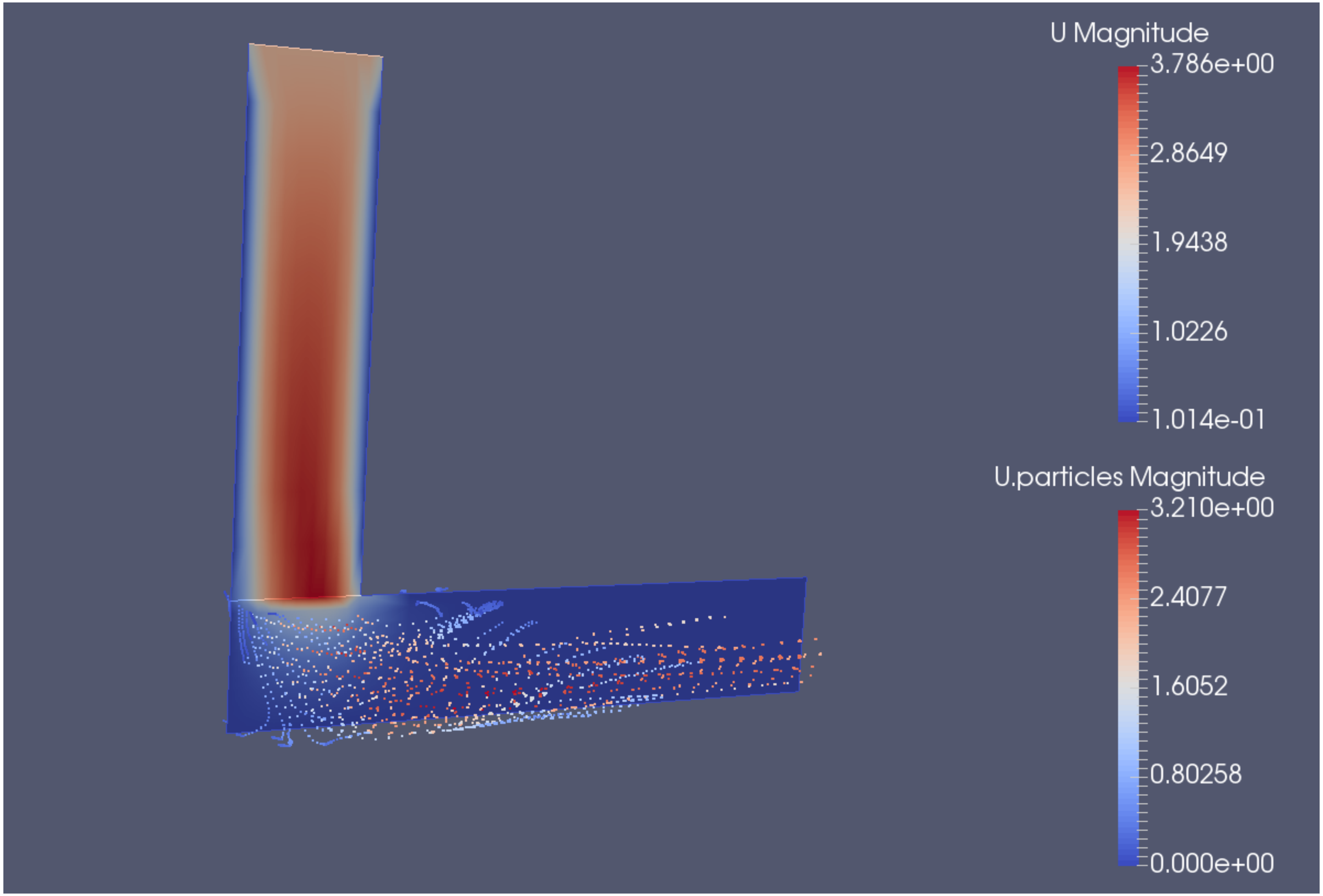
- Number of parcels/cell = (alpha particles * area of cell * normal velocity component to cell boundary face) / (volume of particle * number of particles/parcel * number of time-steps/second)

Velocity contours



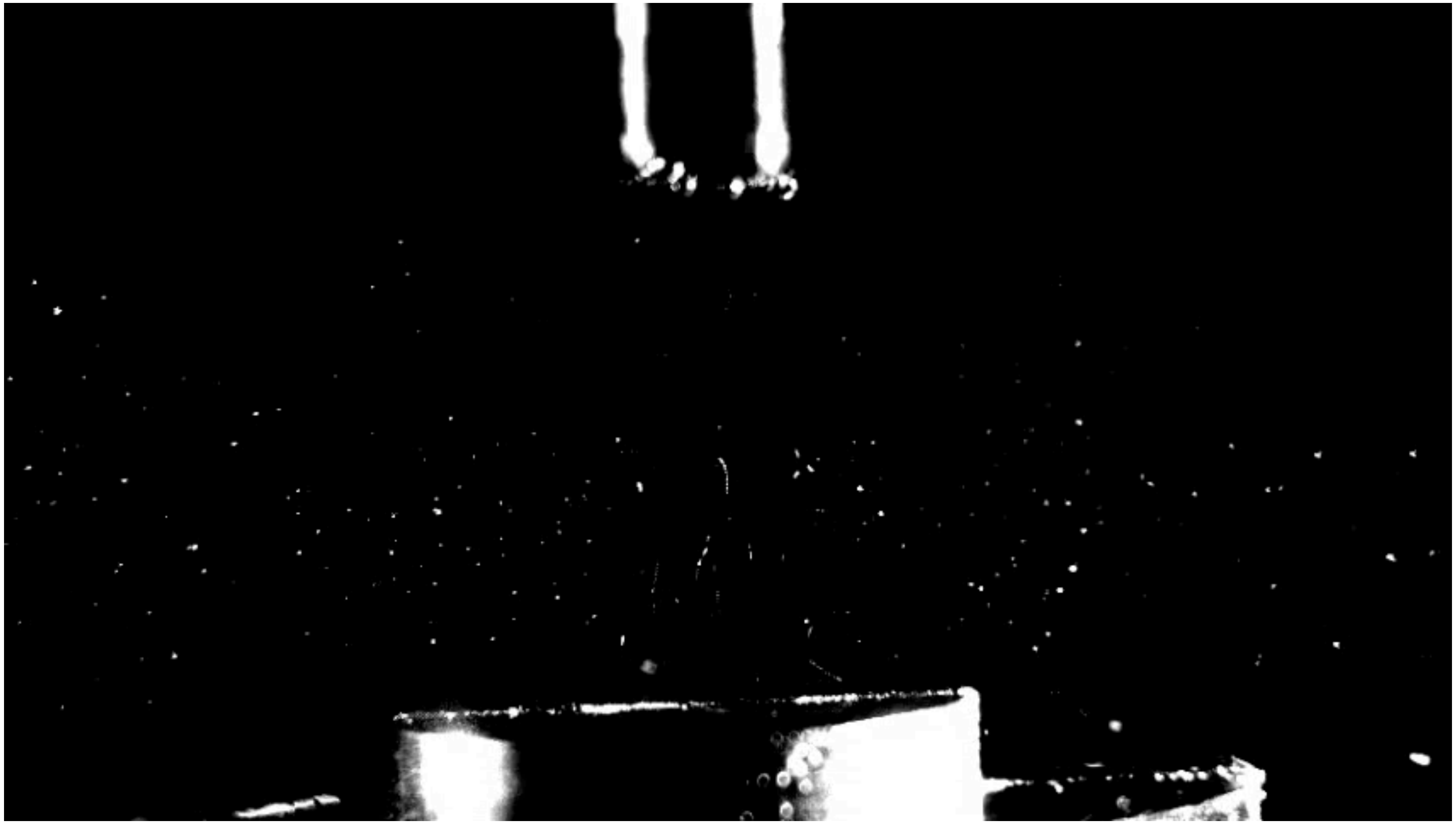
- 2D slice through Z normal. Particles injected from slave patch

Velocity contours



Future work

- Validation of hybrid model: CFD and experimental (PIV)
- Particles to fluid, for after region of interest…
- Make solver re-read the lookupTable (suggestions welcome!)



Conclusion

University of Strathclyde Glasgow

- Solver should dramatically reduce computational time
- Particle data should still be present near walls, where required
- Enable better design of mining equipment

Worn impeller of slurry pump

Thank you. Questions?

alasdair.mackenzie.100@strath.ac.uk

Weir Advanced Research Centre, University of Strathclyde, Glasgow, Scotland