DET FRIE FORSKNINGSRÅD
DANISH COUNCIL FOR
INDEPENDENT RESEARCH

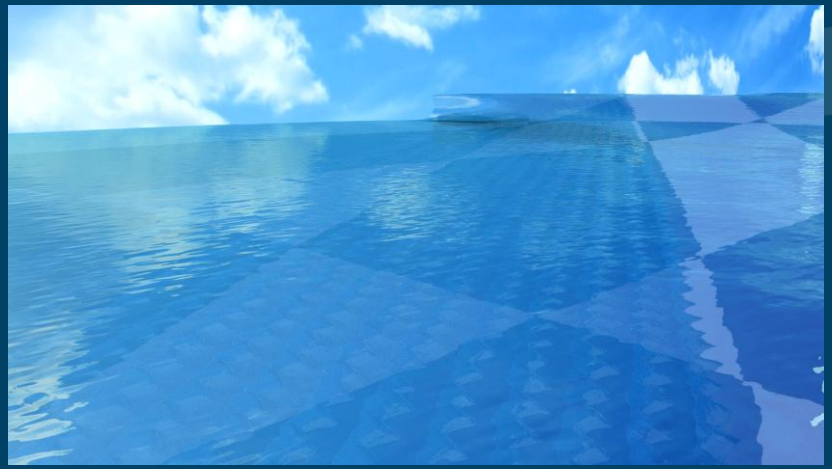
IsoAdvector

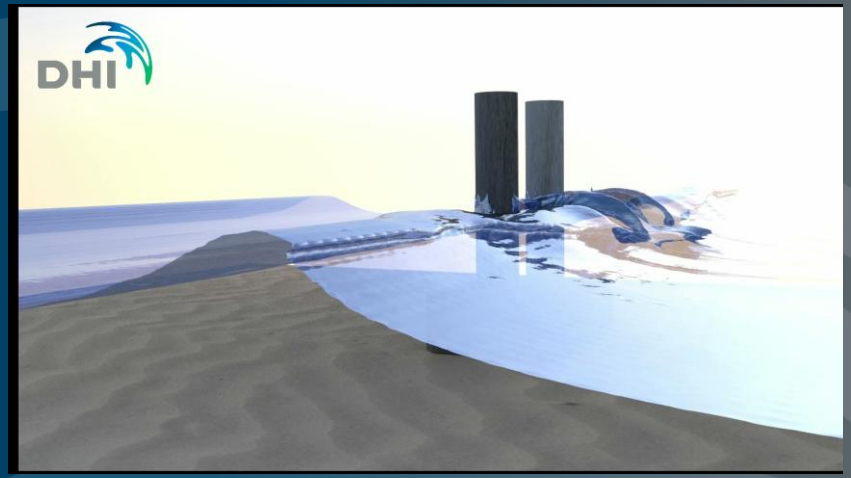
A new geometric VOF method for arbitrary meshes

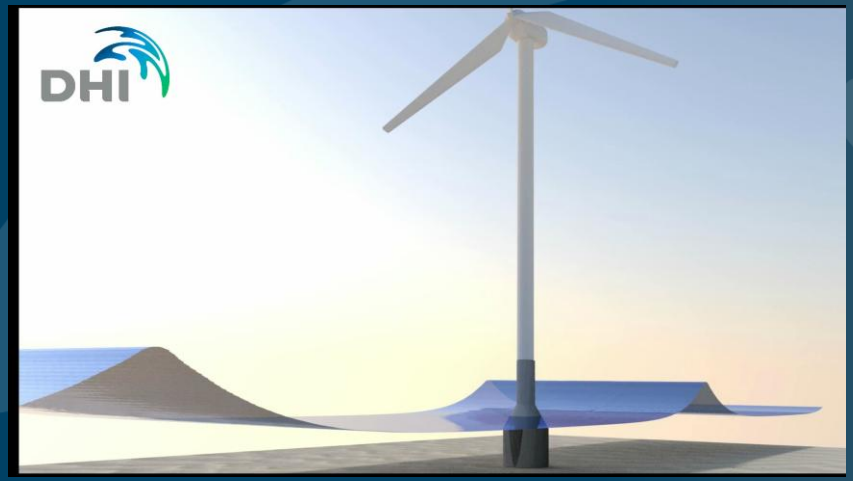
Johan Rønby, Ports & Offshore Technology, DHI
Henrik Bredmose, DTU Wind Energy, Technical University of Denmark
Hrvoje Jasak, FSB, University of Zagreb

OFW11, June 26-30, Guimarães, Portugal

DHI



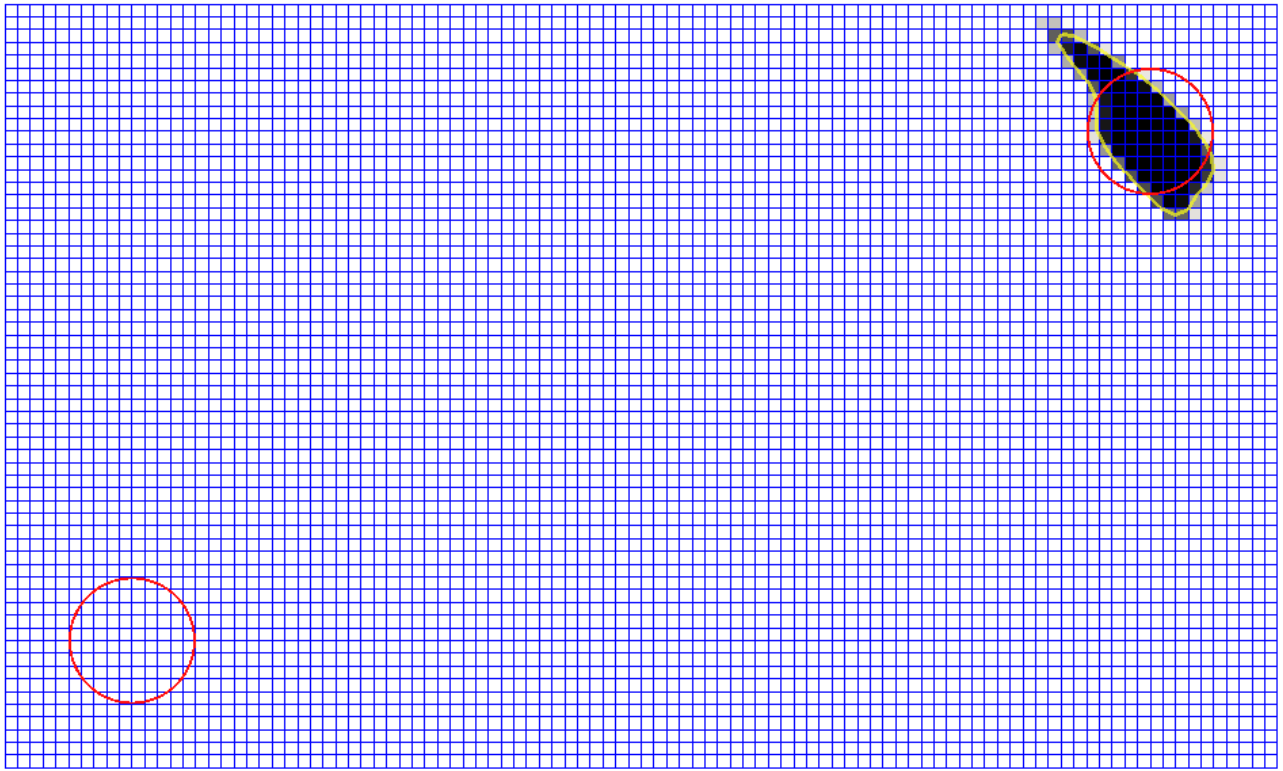




Our experience is that interFoam works well for waves but…

- Tends to give too large and too early wave breaking
- Produces spurious velocities at interface
- Requires very small time steps (CFL~0.1)
- Only converges slowly with mesh refinement

A simple VOF test



Disk advected in velocity field
U = (1, 0.5)

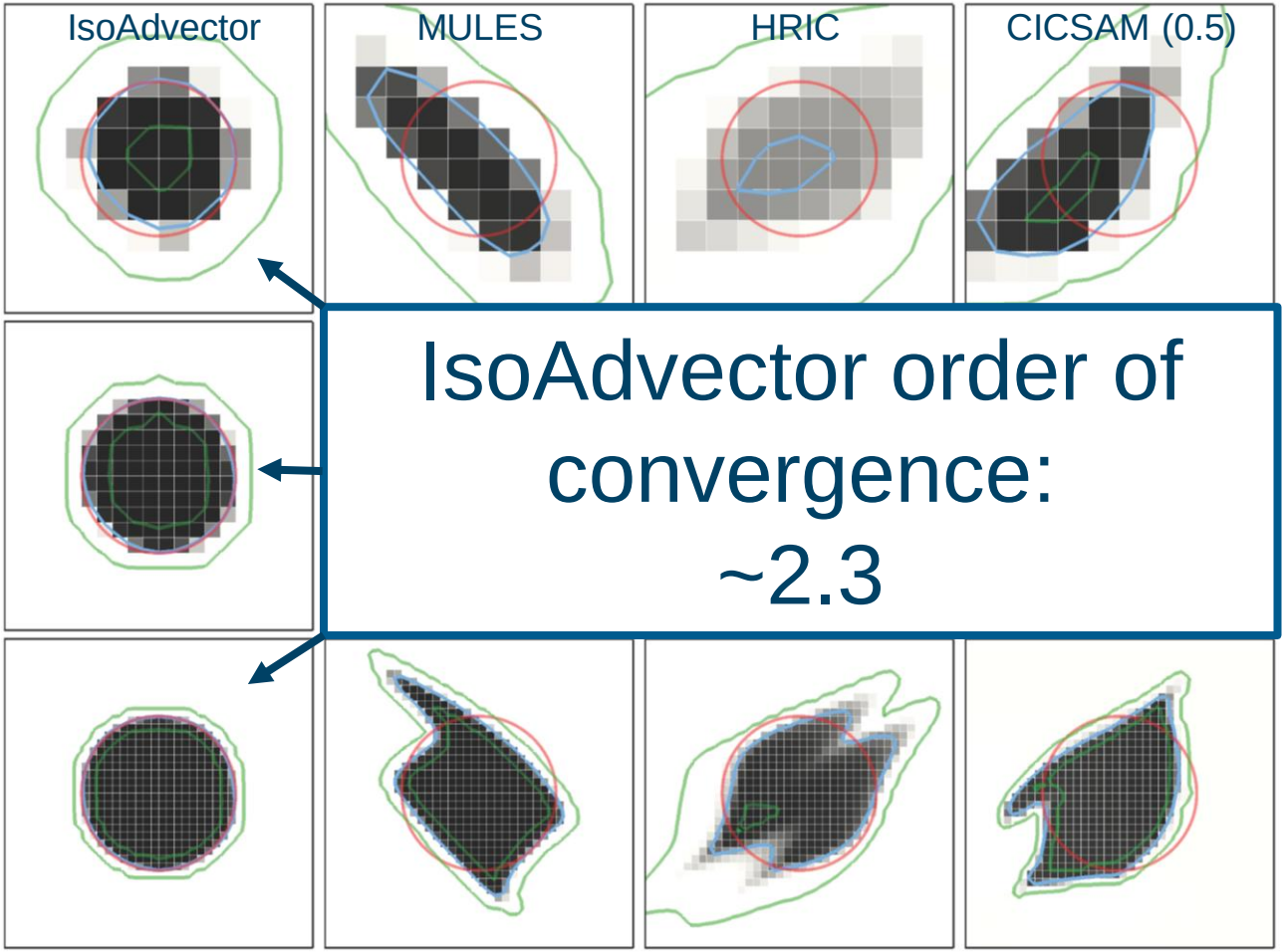
With MULES (interFoam)

CFL = 0.5

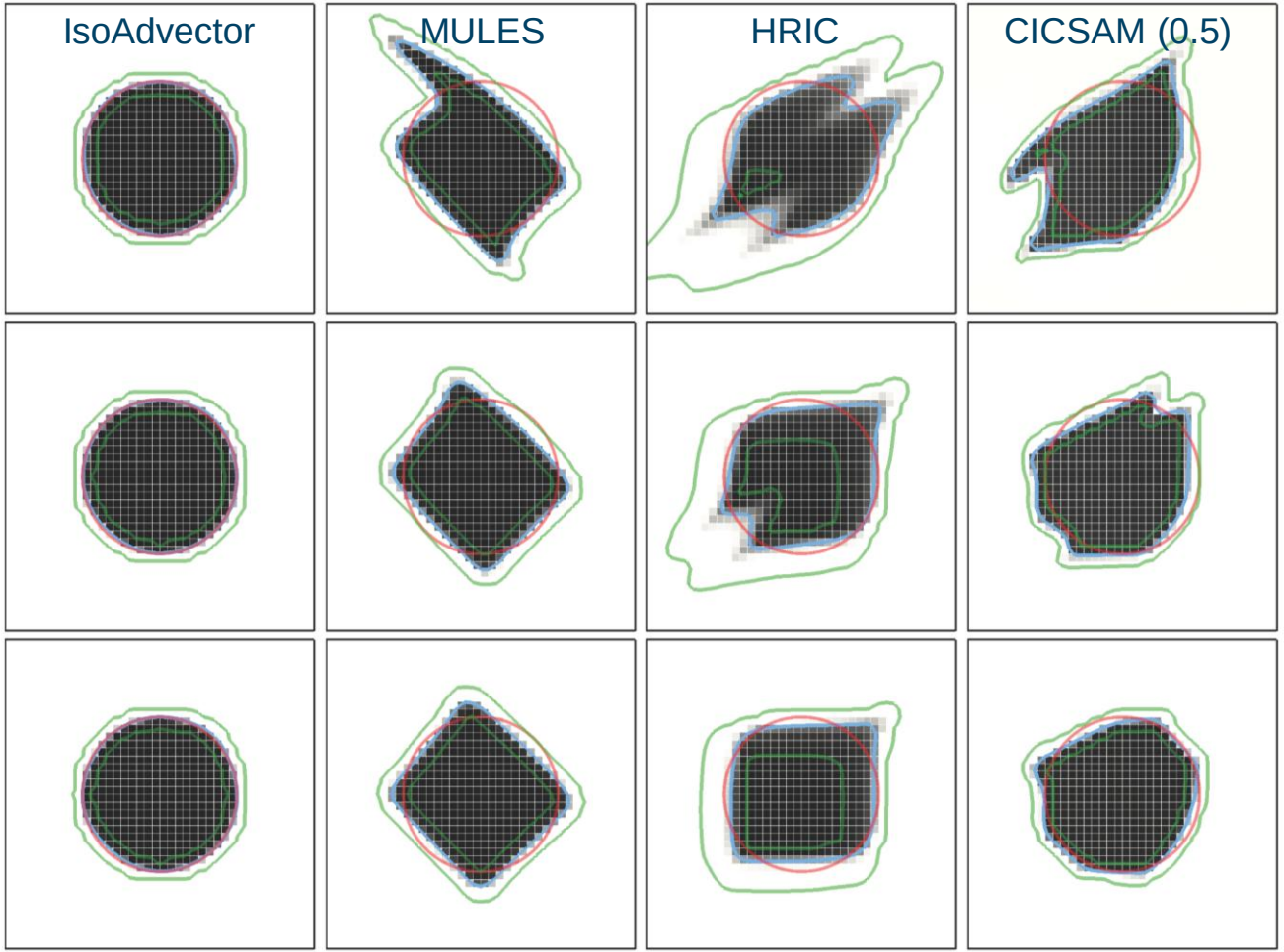
…

Very poor shape preservation

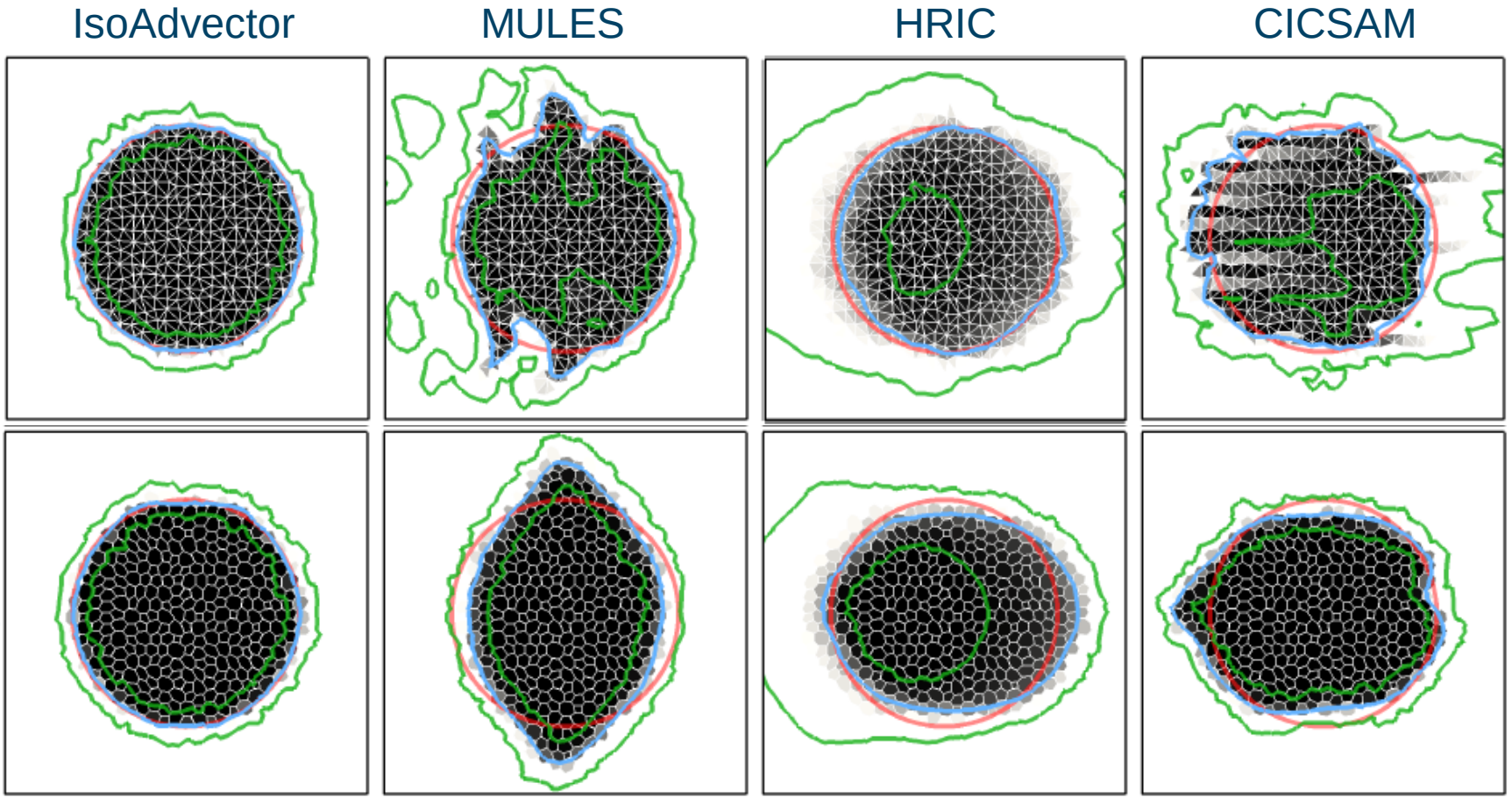
Disc in uniform flow on hex mesh – refining mesh…



Disc in uniform flow on hex mesh – smaller time steps…



Unstructured meshes

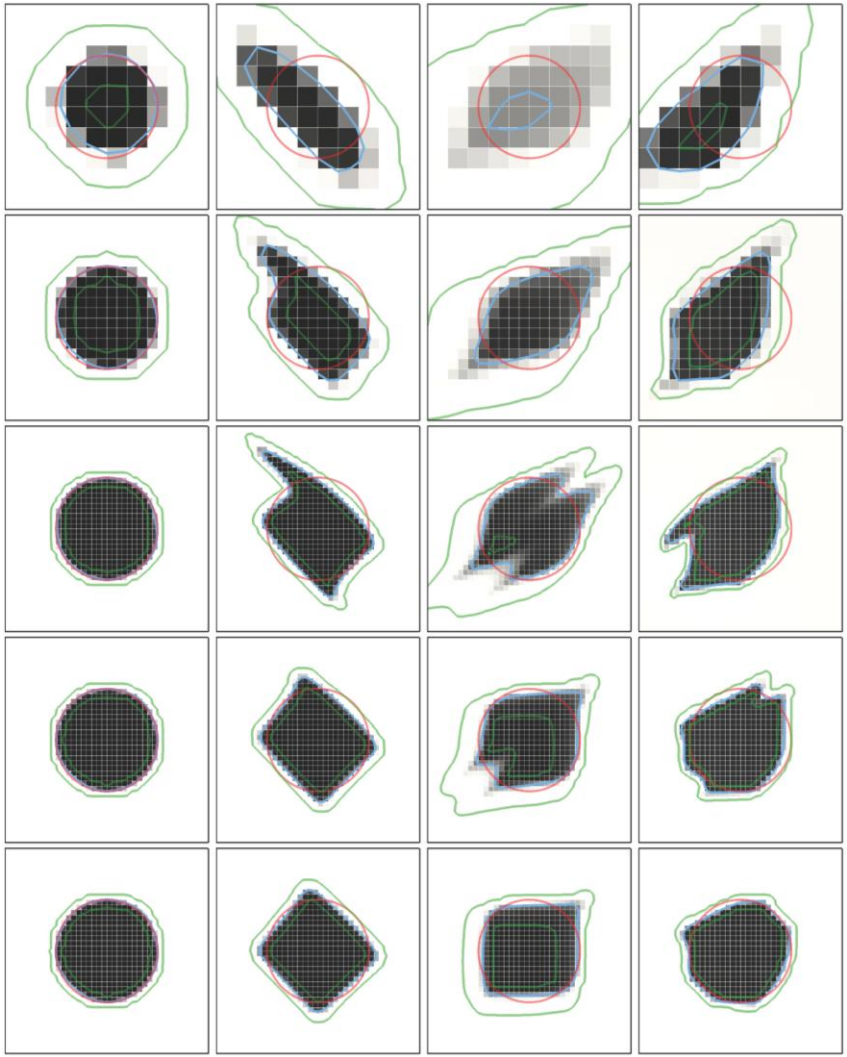


Efficiency

Calculation times in seconds (1 core)

(Nx,Co)	isoAdvector	MULES	HRIC	CICSAM
(30,0.5)	0.23	0.52	0.22	0.19
(60,0.5)	0.86	2.16	0.94	0.87
(120,0.5)	4.03	12.42	5.28	4.49
(120,0.2)	7.22	28.28	9.46	8.4
(120,0.1)	12.53	55.13	16.82	14.61

IsoAdvector is slightly faster than HRIC and CICSAM, and 2-4 times faster than MULES

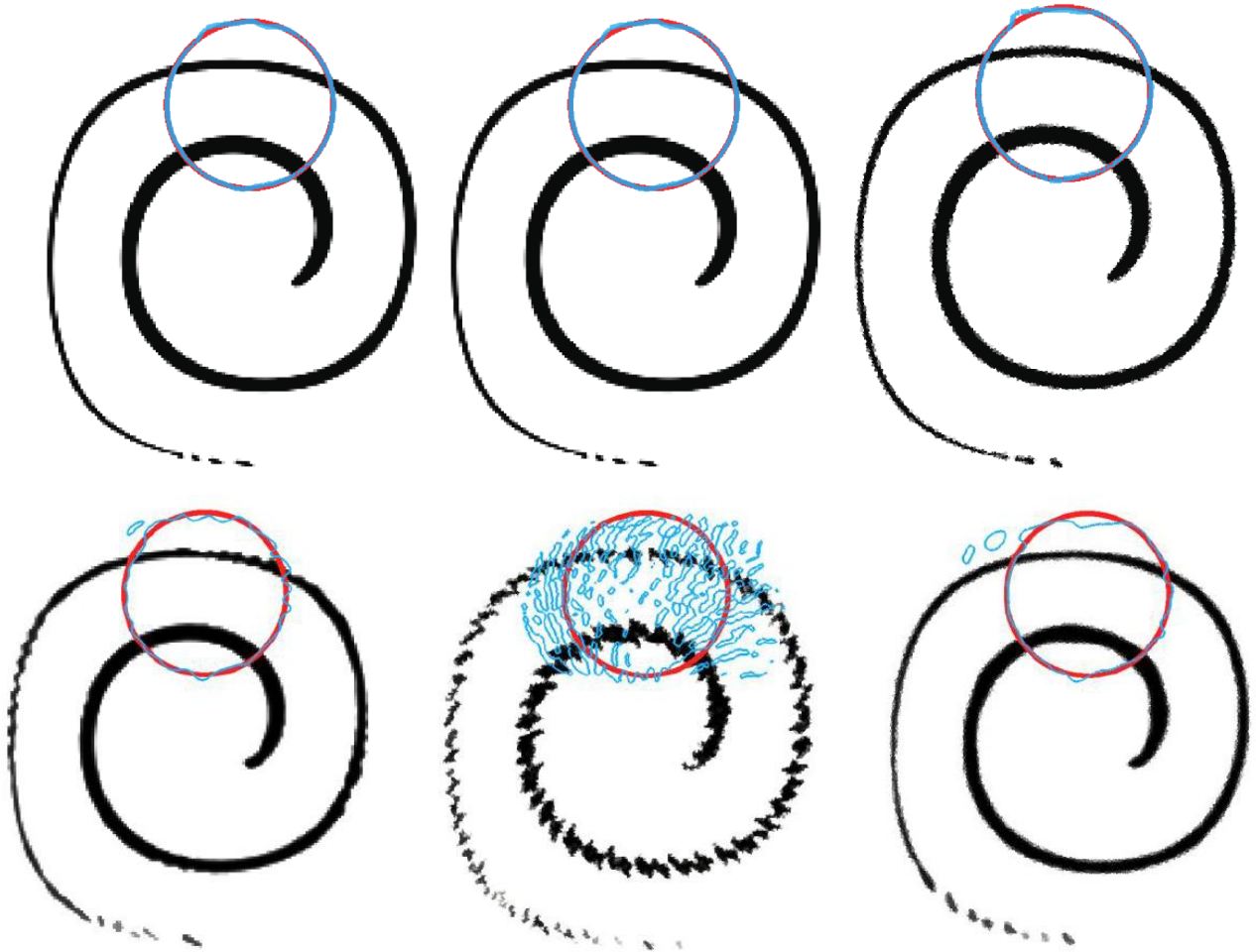


Spiralling Disk

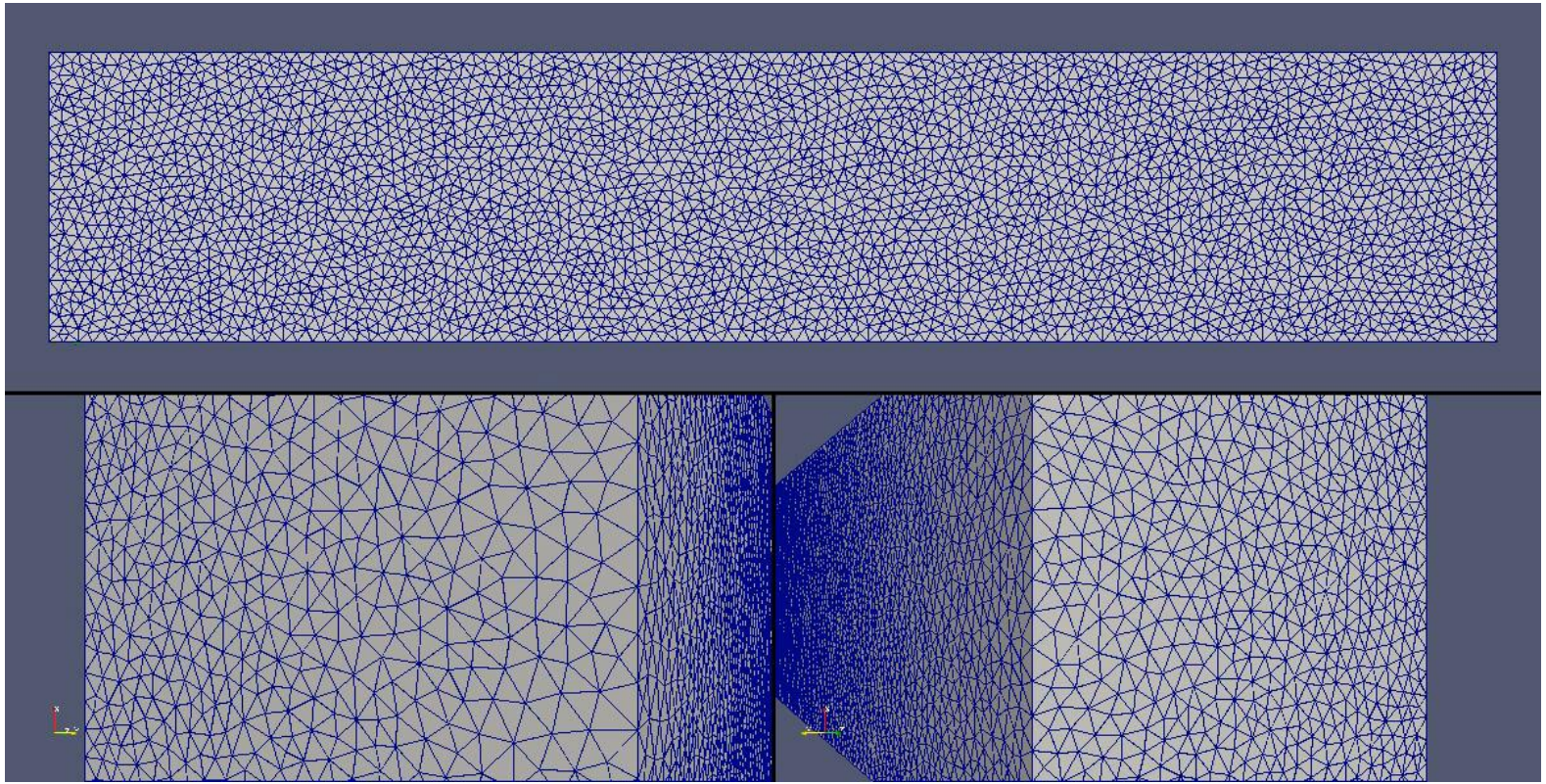
Square cells | Triangular cells | Polygonal cells

IsoAdvector, CFL = 0.5

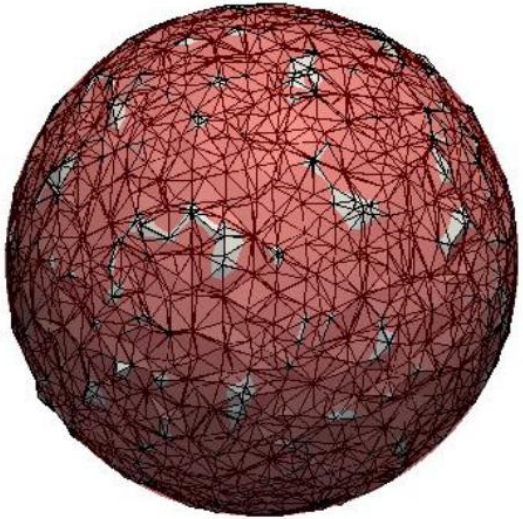
MULES, CFL = 0.1



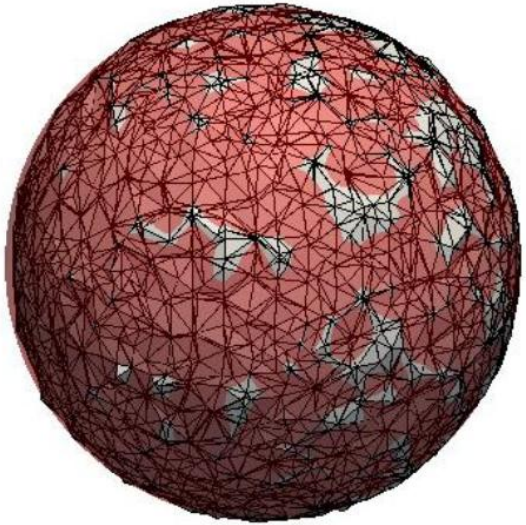
Sphere in uniform flow on random tet mesh



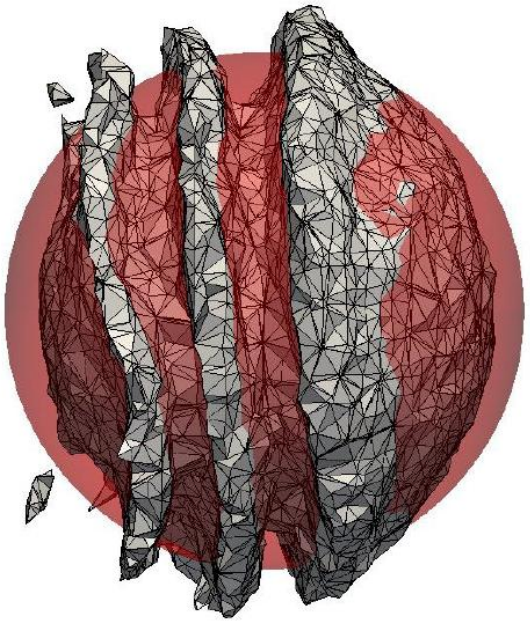
Sphere in uniform flow on random tet mesh



Exact

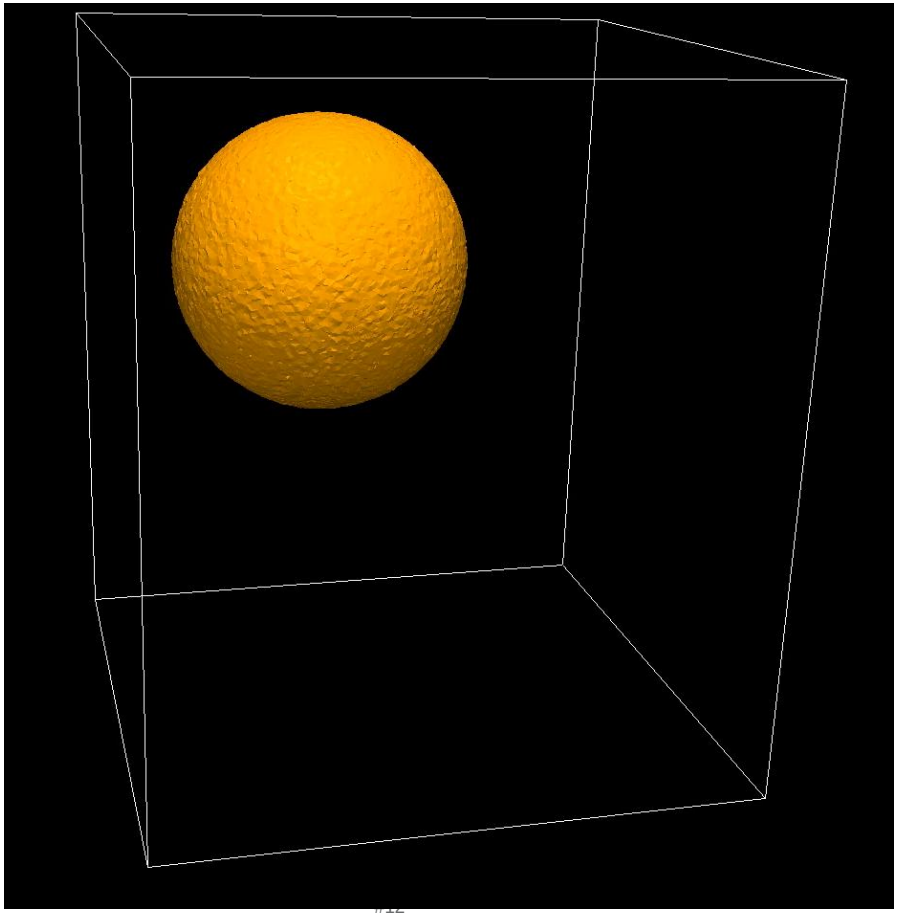


IsoAdvector

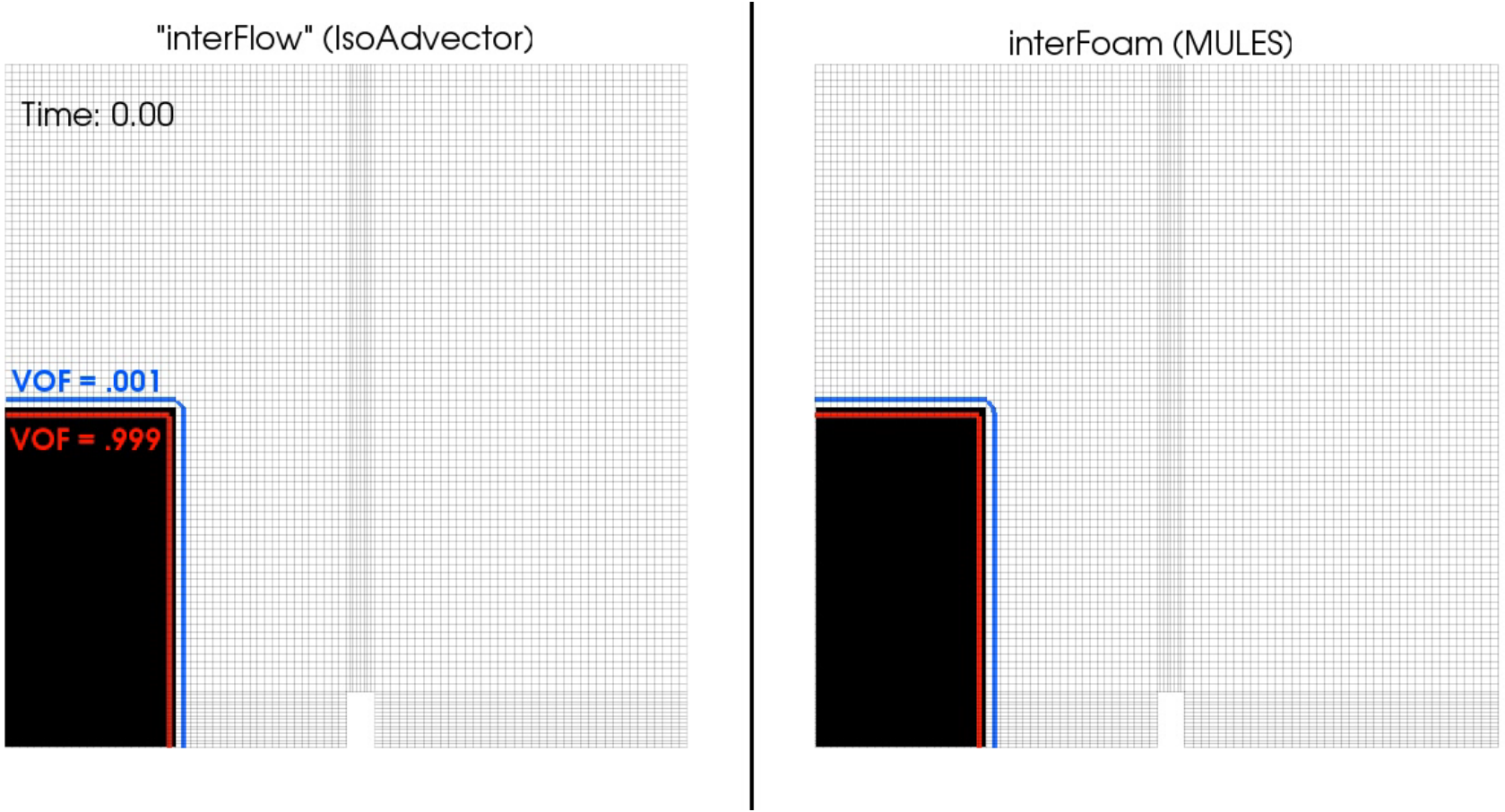


MULES

Sphere in non-uniform 3D flow on random tet mesh

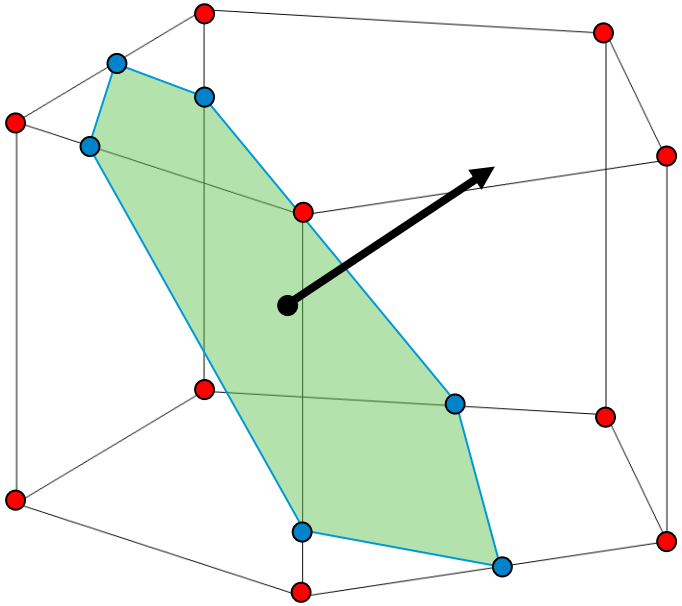


damBreak – IsoAdvector versus MULES

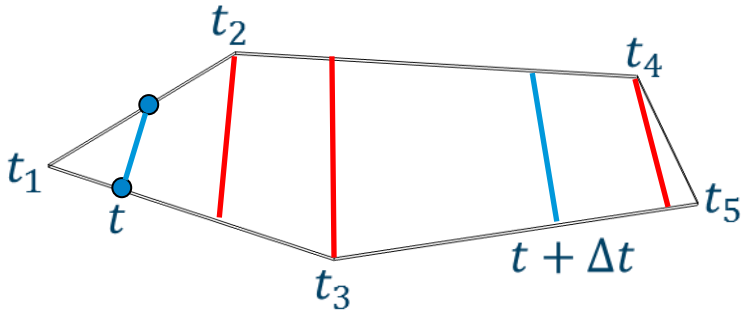


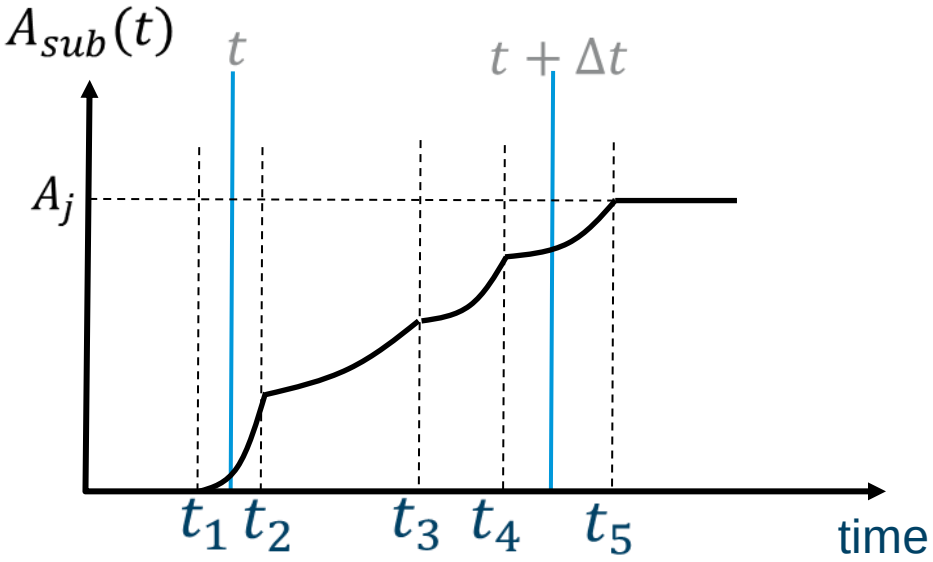
The IsoAdvector concept

Idea 1: The *isoface*



Idea 2: The *face-interface intersection line*





IsoAdvector summary

- Explicit geometric VOF scheme for interface advection on arbitrary meshes
- Beats MULES and other algebraic VOF schemes both on accuracy and speed
- Code will be available on github.com soon
- Implementation in your own interFoam type solver is a one-liner (well, almost)
- Google ”IsoAdvector” to find RSOS preprint on arxiv.org and youtube channel

Thank you

Zalesak's (notched) disk

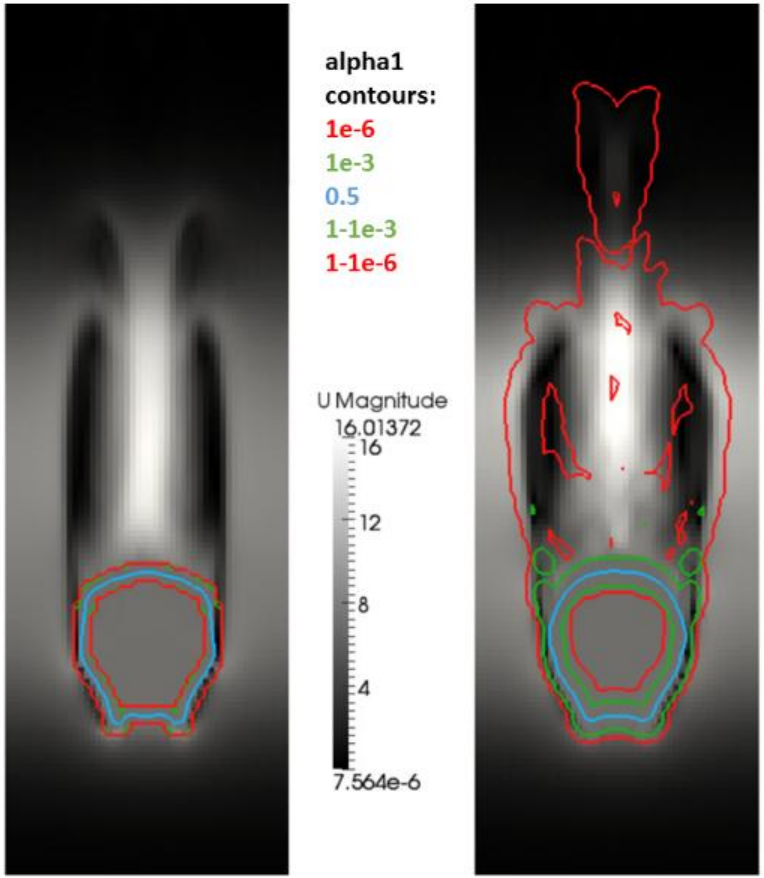
CFL = 0.1 | CFL = 0.5 | CFL = 1.0 | CFL = 1.5

dV = -7.7e-11%	dV = -2.2e-08%	dV = 2.7e-06%	dV = 0.00012%
E1 = 3.5%	E1 = 2.9%	E1 = 3.5%	E1 = 6.5%
Unbound = 8e-17	Unbound= 1.1e-16	Unbound = 3.9e-08	Unbound = 5.7e-09

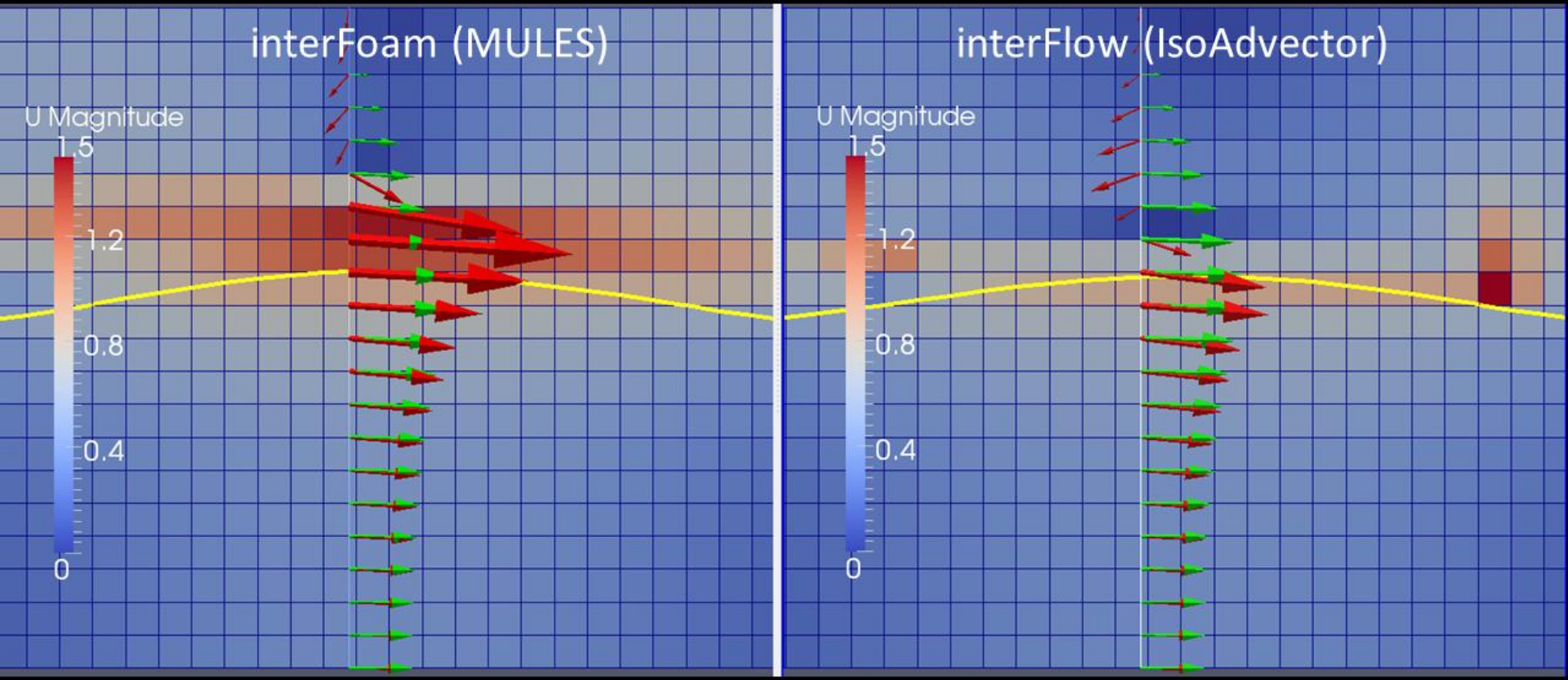
Water droplet falling in air

IsoAdvector

MULES



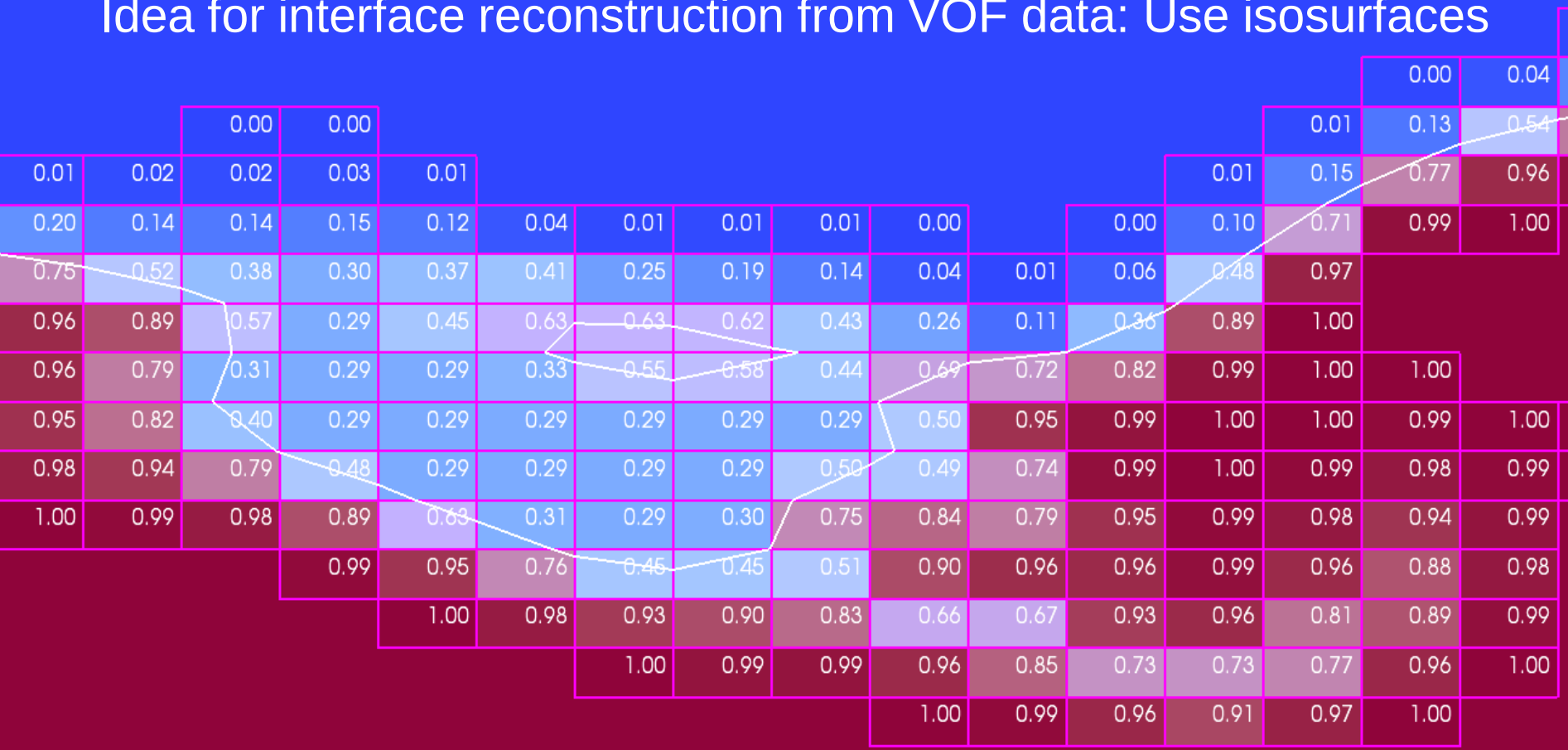
Velocity profile for stream function wave



Green arrows: Theoretical profile. Red arrows: Actual profile. Yellow contour: Free surface.
Wave input: D = 1m, H = 0.3m, T = 3s

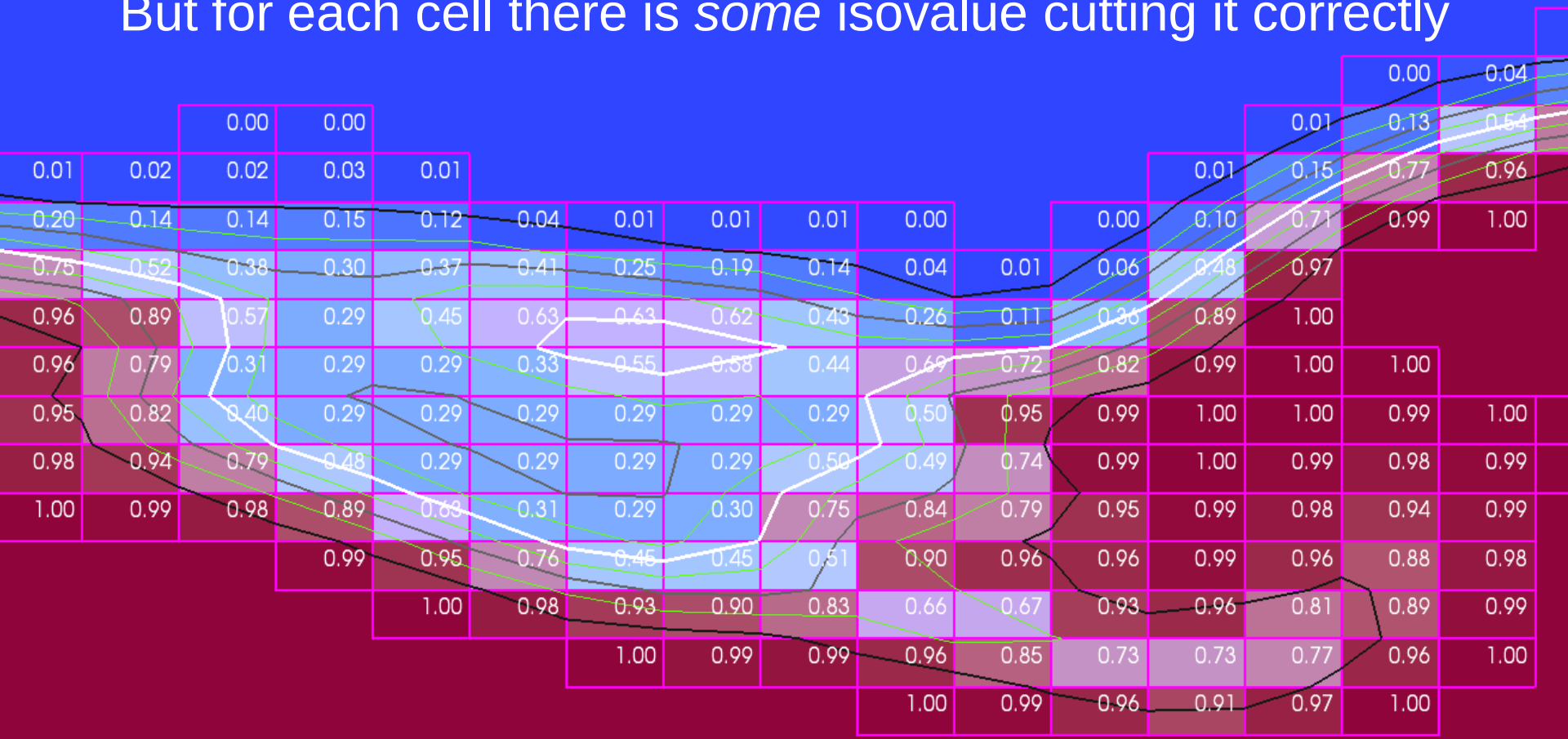
The IsoAdvector concept

Idea for interface reconstruction from VOF data: Use isosurfaces



$\alpha = 0.5$ isosurface generally cuts cells incorrectly

But for each cell there is *some* isovalue cutting it correctly

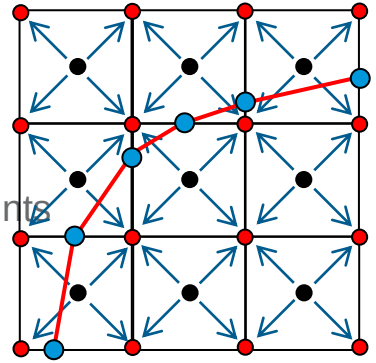


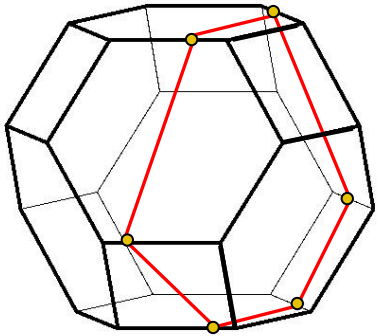
How to calculate an isosurface from volume fraction data?

Steps:

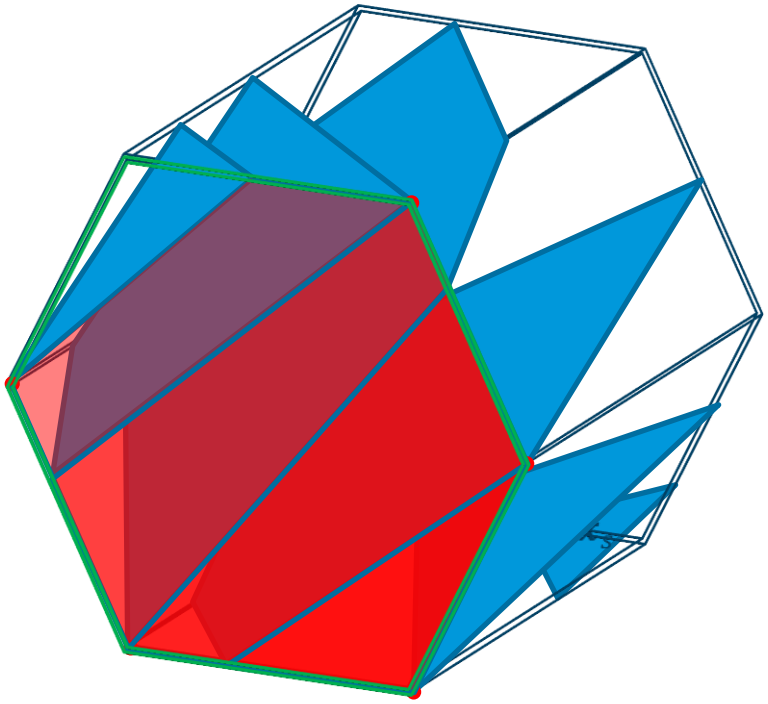
1. Interpolate VOF cell data to *vertices*
2. For each *edge* find cutting point by linear interpolation using *isovalue*
3. For each cut cell form isosurface by connecting edge cutting points

Gives surface position and orientation inside cells on *arbitrary unstructured* 3D meshes





Estimating face-interface intersection line during time step



At time t cell i is a *surface cell* if $\epsilon < \alpha_i(t) < 1 - \epsilon$ (typically we use $\epsilon = 10^{-8}$).

We calculate its *isoface*, that is, the isosurface confined to the cell volume and with isovalue consistent with $\alpha_i(t)$.

This cell specific isovalue is currently found by binary search.

The isoface has a well-defined face centre, $\boldsymbol{x}_s$, and unit face normal, $\boldsymbol{n}_s$, together defining a plane.

We interpolate the velocity data, $\boldsymbol{U}$, to the isoface centre, $\boldsymbol{x}_s$, to get its velocity, $\boldsymbol{U}_s(t)$.

We dot $\boldsymbol{U}_s(t)$ with the isoface unit normal, $\boldsymbol{n}_s$, to get the isoface velocity normal to itself, $U_n = \boldsymbol{U}_s \cdot \boldsymbol{n}_s$.

Now for every vertex, $\boldsymbol{x}_v$, of every a face we can estimate the interface time of arrival as $t_v = (\boldsymbol{x}_v - \boldsymbol{x}_s) \cdot \boldsymbol{n}_s / U_n$.

The submerged face area

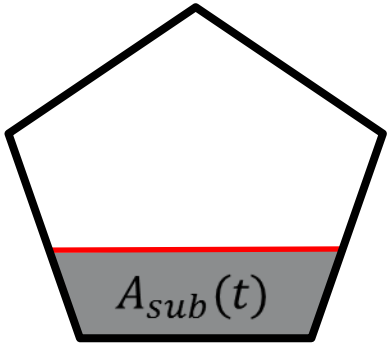
$$\Delta V_j(t,\Delta t) \equiv \int_t^{t+\Delta t} \int_{F_j} I(\boldsymbol{x},\tau)\boldsymbol{U}(\boldsymbol{x},\tau)\cdot d\boldsymbol{S}\, d\tau \approx \frac{\phi_j(t)}{|\boldsymbol{S}_j|} \int_t^{t+\Delta t} A_{sub}(\tau) d\tau$$

Volumetric face flux:

$$\phi_j(t) \equiv \int_{F_j} \boldsymbol{U}(\boldsymbol{x},t)\cdot d\boldsymbol{S}$$

Submerged face area

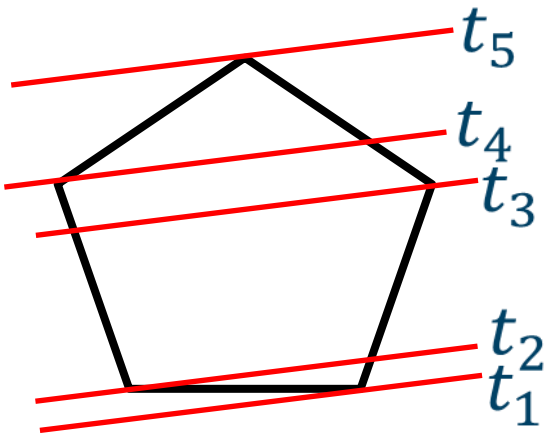
$$A_{sub}(t) \equiv \int_{F_j} I(\boldsymbol{x},t)\, dA$$

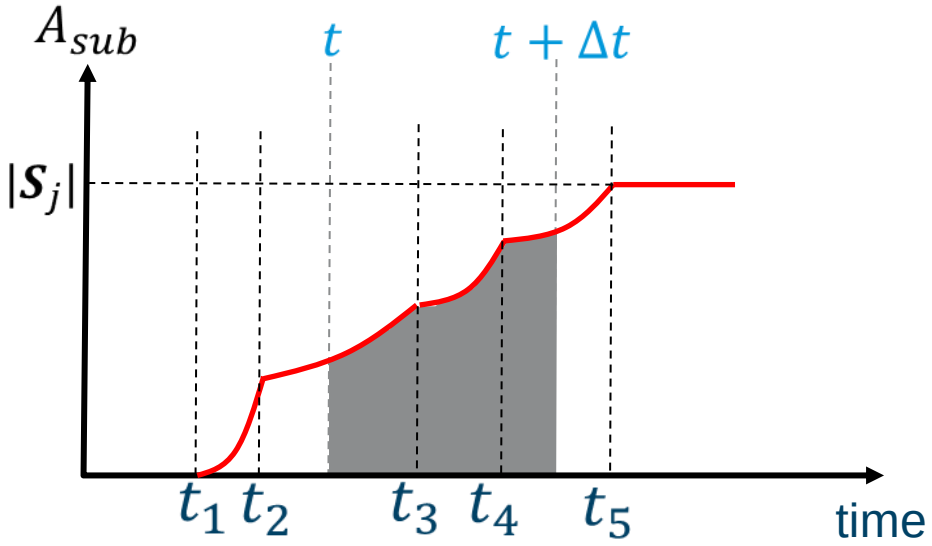


Face F_j

Time integrated submerged face area

$$\Delta V_j(t, \Delta t) \approx \frac{\phi_j(t)}{|\boldsymbol{S}_j|} \int_t^{t+\Delta t} A_{sub}(\tau) d\tau$$





We estimated $F_j \cap S(t)$ lines at vertex intersection times $t_1, t_2, t_3, \ldots$

Assume steady line motion in between

Then in sub time intervals

$$A_{sub}(\tau) = a\tau^2 + b\tau + c$$

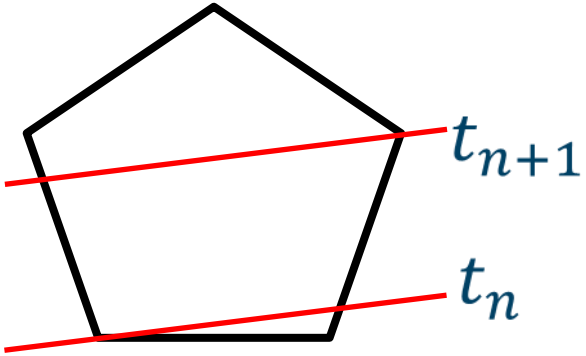
with $a,\ b,\ c$ given by vertices.

Analytical time integral!

IsoAdvector concept summary

$$\alpha_i(t+\Delta t) = \alpha_i(t) - \frac{1}{V_i}\sum_{F_j \in \partial C_i} \Delta V_j(t,\Delta t)$$

$$\begin{aligned}\Delta V_j(t,\Delta t) &\equiv \int_t^{t+\Delta t}\int_{F_j} I(\boldsymbol{x},\tau)\boldsymbol{U}(\boldsymbol{x},\tau)\cdot d\boldsymbol{S}\, d\tau \\ &\approx \frac{\phi_j(t)}{|\boldsymbol{S}_j|}\int_t^{t+\Delta t} A_{sub}(\tau)d\tau \\ &\approx \frac{\phi_j(t)}{|\boldsymbol{S}_j|}\sum_{n=1}^{M-1}\left[\frac{1}{3}a_n\tau^3 + \frac{1}{2}b_n\tau^2 + c_n\tau\right]_{\tau=t_n}^{t_{n+1}}\end{aligned}$$



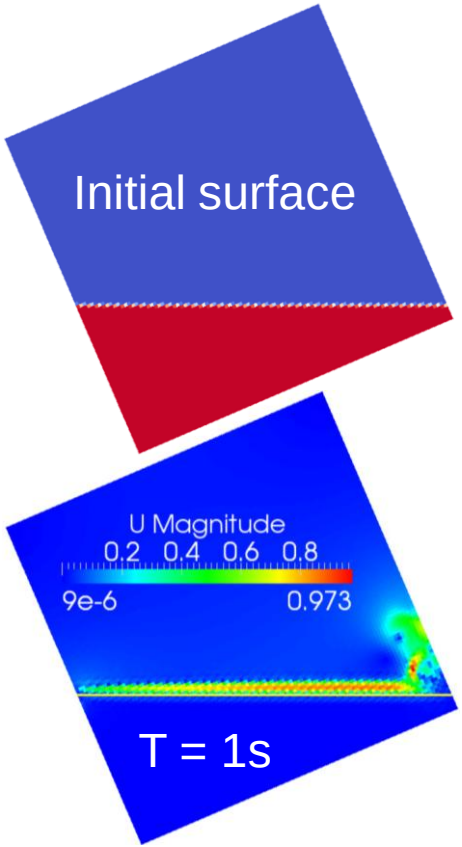
a_n, b_n and c_n given by quadrilateral corners of two subsequent vertex-interface encounters.

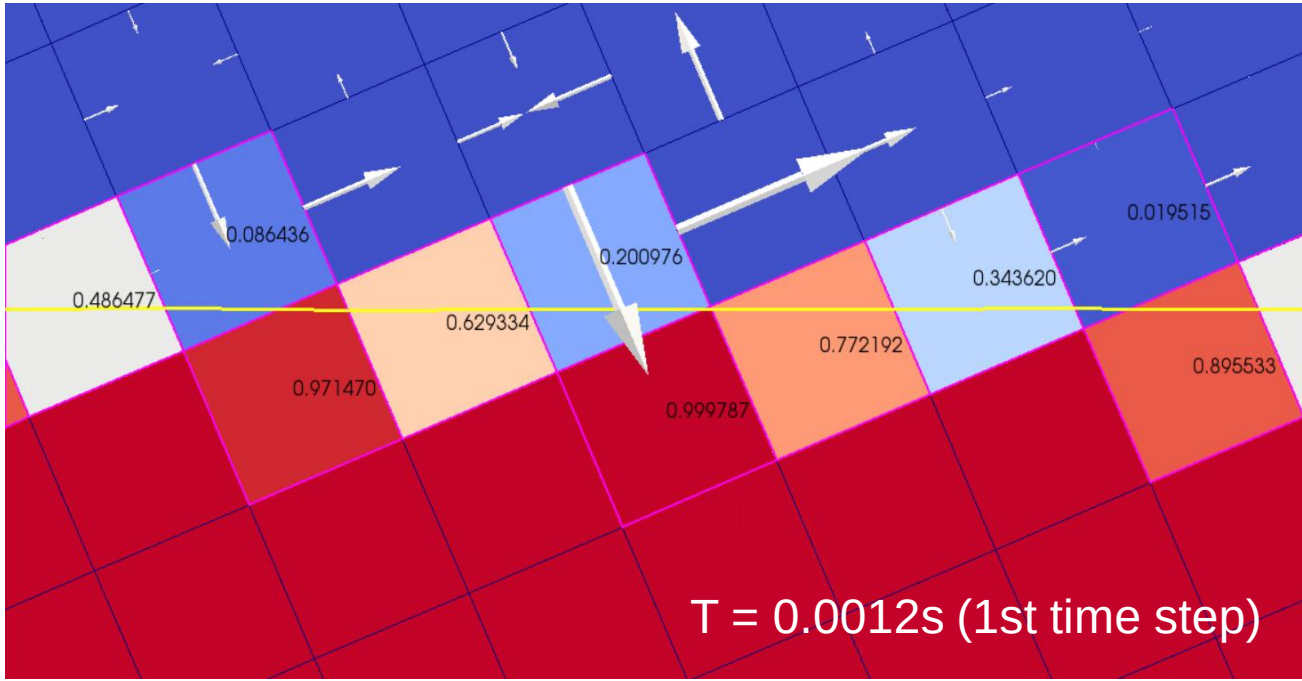
face-interface intersection lines and times calculated using isosurfaces

Optional bounding procedure

- Sum up all face dV_j's for cell i to get change in α_i during time step
- Check if more water is added than there is room for (overfilling) or more water is removed than is available in the cell (over-emptying)
- If this is the case adjust dV_j for all cell faces cut by isoface so that cell is exactly filled/emptied. This is currently done by using the dV_j's of these faces as weights of total added/removed water rather than as absolute water volumes.
- Because it is the dV_j's that are modified, this step does not compromise volume conservation.
- For practical reasons it is convenient to have the option to snap α_i's to 0 or 1

Inconsistency in interFoam pressure handling





Tilted 1m x 1m box with initial flat water

Pressure equation in interFoam (pEqn.H)

```
    volScalarField rAU("rAU", 1.0/UEqn.A());
    surfaceScalarField rAUf("Dp", fvc::interpolate(rAU));

    volVectorField HbyA("HbyA", U);
    HbyA = rAU*UEqn.H();

    surfaceScalarField phiHbyA
    (
        "phiHbyA",
        (fvc::interpolate(HbyA) & mesh.Sf())
      + fvc::ddtPhiCorr(rAU, rho, U, phi)
    );

    adjustPhi(phiHbyA, U, p_rgh);
    phi = phiHbyA;

    surfaceScalarField phig
    (
        (
            fvc::interpolate(interface.sigmaK())*fvc::snGrad(alpha1)
          - ghf*fvc::snGrad(rho)
        )*rAUf*mesh.magSf()
    );

    phiHbyA += phig;

    while (pimple.correctNonOrthogonal())
    {
        fvScalarMatrix p_rghEqn
        (
            fvm::laplacian(rAUf, p_rgh) == fvc::div(phiHbyA)
        );
```

To numerically take the gradient of a jump function is asking for trouble

$$\nabla \cdot \int_{C_i} (\mathbf{g} \cdot \boldsymbol{x}) \nabla \rho \, dV =$$

$$\sum_{face\ j} \int_{F_j} (\boldsymbol{g} \cdot \boldsymbol{x}) \nabla \rho \cdot d\boldsymbol{S}$$

$$\nabla \rho = (\rho_2 - \rho_1) \hat{\boldsymbol{n}}_S \delta(\boldsymbol{x} - \boldsymbol{x}_S)$$

$$\int_{F_j} (\boldsymbol{g} \cdot \boldsymbol{x}) \nabla \rho \cdot d\boldsymbol{S} =$$

$$(\rho_2 - \rho_1) \int_{F_j \cap S} (\boldsymbol{g} \cdot \boldsymbol{x}) \hat{\boldsymbol{n}}_S \cdot d\boldsymbol{S}$$

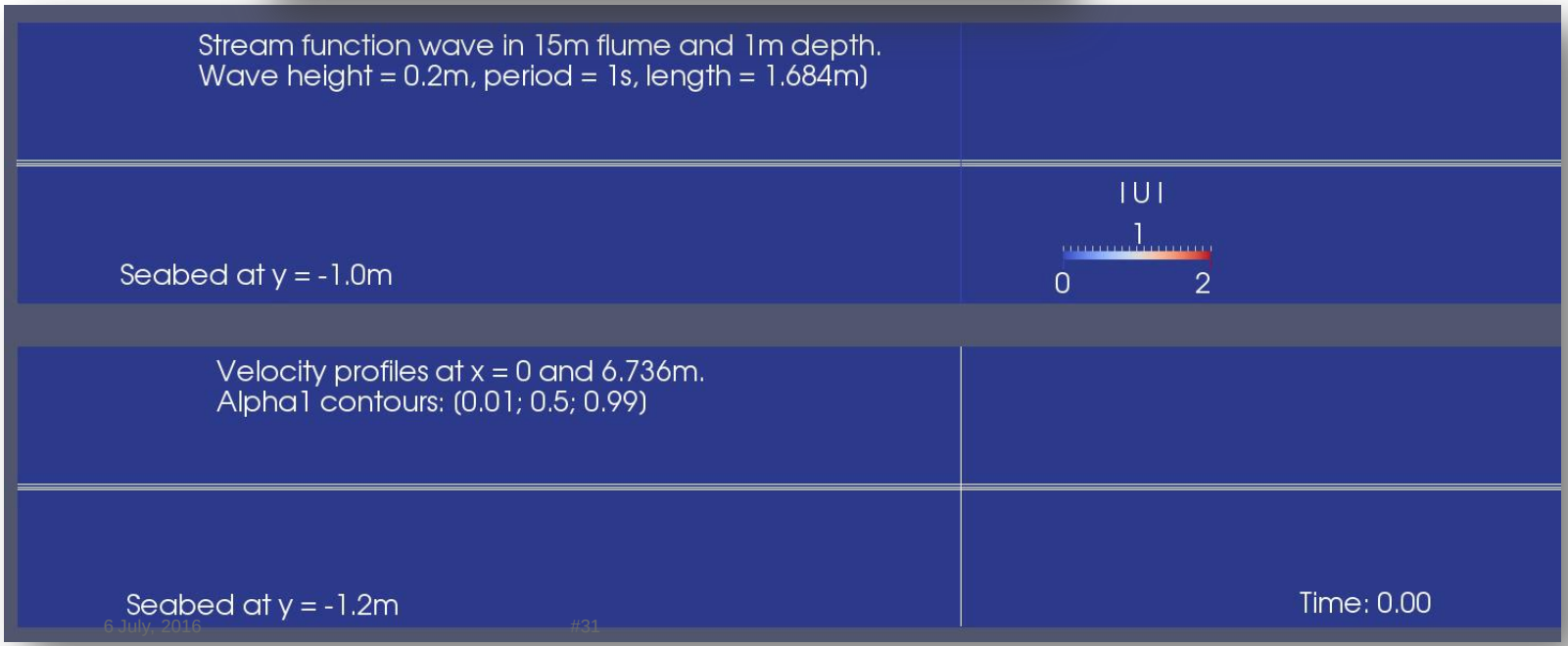
Solution depends on choice of z = 0

interFoam/pEqn.H:

```
surfaceScalarField phig
(
    (
        fvc::interpolate(interface.sigmaK())*fvc::snGrad(alpha1)
      - ghf*fvc::snGrad(rho)
    )*rAUf*mesh.magSf()
);

phiHbyA += phig;
```

ExecutionTime = 4921.02 s



ExecutionTime = 4072.95 s