Spatial extension of the advection step of the geometric Volume Of Fluid algorithm isoAdvector

Konstantinos Missios

Department of Science and Environment,
Roskilde University,
Roskilde, Denmark

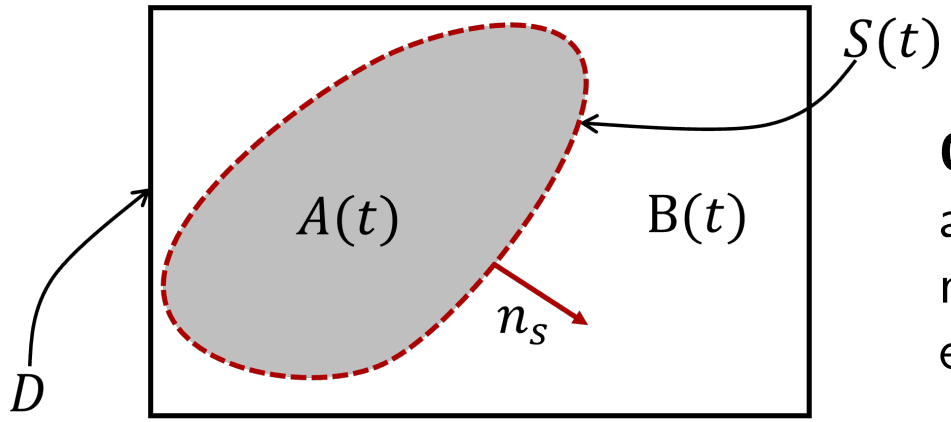
January 20, 2022

Outline

1. Background theory
   - Interfacial flows (a glimpse)
   - Volume of Fluid (VoF) method
   - isoAdvector
2. Motivation
3. Implementation
   - isoAdvector
   - extension
4. Hands-on modification
5. Test-case
6. Conclusion

Interfacial flows

Interfacial flows is a **sub-category** of multiphase flow and play a significant role in various applications and processes. Examples of such flows include **breaking waves**, **spray dryers**, **injection systems**, **oil extraction**, etc.



Interfacial flow sketch

One-fluid approach: The two phases are handled as one fluid with variable material properties (density, viscosity, etc.).

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \qquad \nabla \cdot \mathbf{u} = 0$$

$$\rho \left[\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{u}\mathbf{u} \right] = -\nabla p + \rho \mathbf{g} + \nu \nabla \cdot (\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}) + \mathbf{f_s}$$

Volume of Fluid (VoF) Method

Heaviside indication function

$$H(\mathbf{x},t) = \frac{\rho(\mathbf{x},t) - \rho_b}{\rho_a - \rho_b} = \begin{cases} 1, & \text{if } \mathbf{x} \in A(t) \\ 0, & \text{if } \mathbf{x} \in B(t) \end{cases}.$$

Volume fraction is acquired through integrating over a control volume V_p

$$\alpha_P = \frac{1}{V_P} \int_{V_P} H(\mathbf{x},t) dV.$$

Given the incompressibility condition $H(\mathbf{x},t)$ can be consider a passive scalar that follows the advection equation.

$$\frac{\partial H}{\partial t} + \nabla \cdot (H\mathbf{u}) = 0 \xrightarrow{\int} \frac{d\alpha_P}{dt} + \frac{1}{V_P} \sum_f \int_{S_f} H(\mathbf{x},t)\, \mathbf{u}(\mathbf{x},t) \cdot d\mathbf{S} = 0.$$

The semi-descrite form of the volume fraction evolution is obtained after integrating over a control volume V_p and applying the divergence theorem.

isoAdvector 1/5

After temporal integration

$$\alpha_P(t+\Delta t) = \alpha_P(t) - \frac{1}{V_P}\sum_f \int_t^{t+\Delta t} \int_{S_f} H(\mathbf{x},\tau)\,\mathbf{u}(\mathbf{x},\tau)\cdot d\mathbf{S}d\tau$$

$$\alpha_P(t+\Delta t) = \alpha_P(t) - \frac{1}{V_P}\sum_f \Delta V_f(t,\Delta t)$$

ΔV_f represents the volume of the fluid A that is transported through face f during the time-step Δt. And its calculation is the **ultimate goal** of the advection step of isoAdvector.

isoAdvector 2/5

Under the assumption that the velocity field remains temporally constant during the time-step dt, $\mathbf{u}(\mathbf{x},\tau) = \mathbf{u}(\mathbf{x},t)$

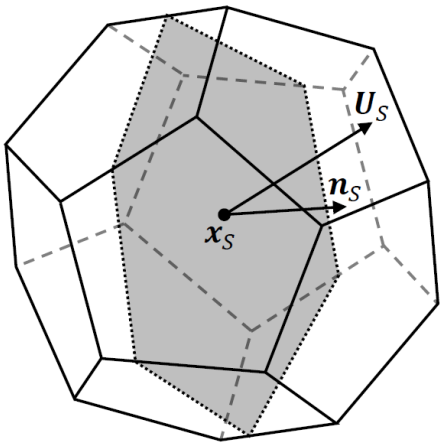
$$\Delta V_f(t,\Delta t) \approx \int_t^{t+\Delta t} \int_{S_f} H(\mathbf{x},\tau) \frac{\overline{\phi}_f(t)}{A_f} dA d\tau = \overline{\phi}_f(t) \int_t^{t+\Delta t} \underbrace{\frac{1}{A_f} \int_{S_f} H(\mathbf{x},\tau)\, dA}_{\alpha_f^+(\tau)} d\tau$$

$$\text{"Area of Fluid": } \alpha_f^+(t) = \frac{1}{A_f} \int_{S_f} H(\mathbf{x},t)\, dA$$

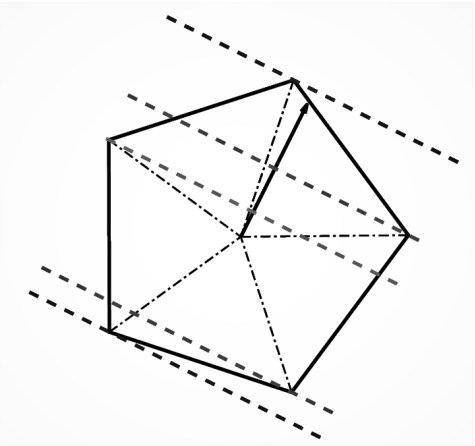
$$\text{Total face area: } A_f = \int_f dA$$

In other words, given a face f that is cut by the interface, the ratio of the face area submerged in the heavy fluid to the total face area A_f is equal to α_f^+.

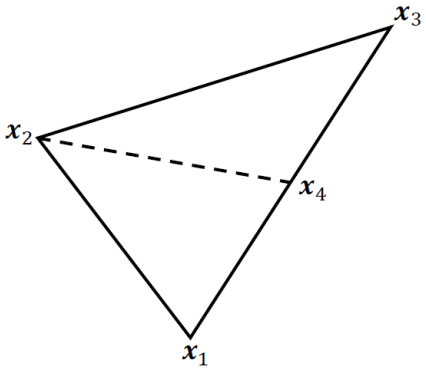
isoAdvector 3/5



Cell cut by planar interface



face-interface intersection line



subface cut by interface

Central to isoAdvector is the evolution of the **face-interface intersection line**

$$\text{Arrival time: } t_v = t + \frac{(\mathbf{x}_v - \mathbf{x}_s) \cdot \hat{\mathbf{n}}_s}{\mathbf{u}_s \cdot \hat{\mathbf{n}}_s}$$

isoAdvector 4/5

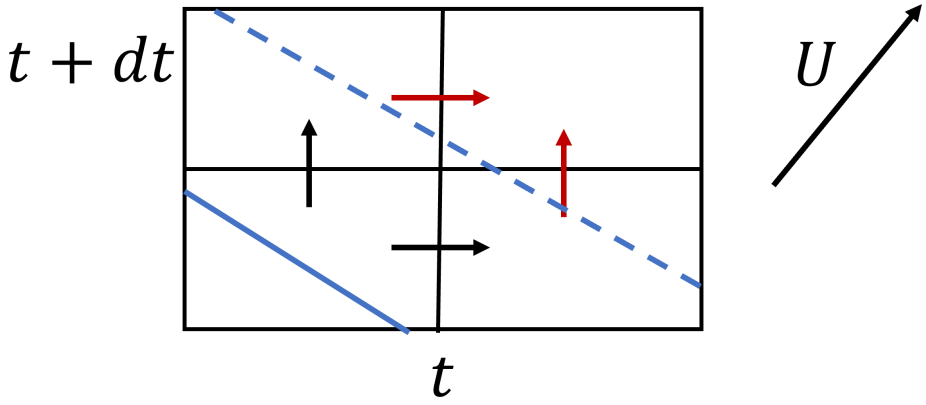
$$\int_{t}^{t+\Delta t} \alpha_f^+(\tau) d\tau \approx \sum_{i=1}^{N} \sum_{j=1}^{2} \frac{1}{3} a_{i,j} \left(t_{i,j+1}^3 - t_{i,j}^3 \right) + \frac{1}{2} b_{i,j} \left(t_{i,j+1}^2 - t_{i,j}^2 \right) + c_{i,j} \left(t_{i,j+1} - t_{i,j} \right), j = \begin{cases} 1, & \text{if } \tau \in [t_1,\ t_2] \\ 2, & \text{if } \tau \in (t_2,\ t_3] \end{cases} \text{ and } i = 1, \ldots, N$$

where $t_{i,1}$, $t_{i,2}$ and $t_{i,3}$ are the arrival times of the i'th triangular sub-face. The coefficients are determined based on the position of subface's vertices.

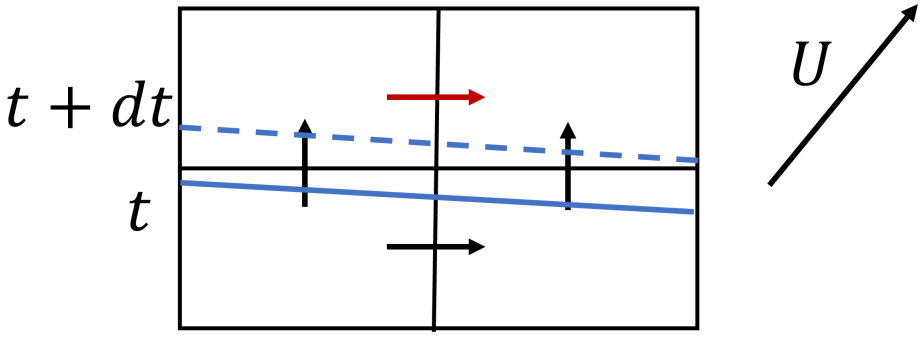
isoAdvector 5/5

1. In every **downwind** face f of a cell C_P the ΔV_f is initialized as $\Delta V_f = \alpha_P \phi_f \Delta t$.
2. All the **interface** cells are identified through the condition $\varepsilon < \alpha(t) < 1 - \varepsilon$, where ε is a tolerance specified by the user.
3. Do for each interface cell:
   1. Determine the isoface based on the reconstruction procedure.
   2. Estimate the isoface motion within the timestep Δt according to the aforementioned advection step.
4. For each cell update the volume fraction.
5. For every cell C_P with $\alpha_P(t) \notin [0, 1]$ apply the bounding procedure.

Motivation (the issue)



Omitted cell

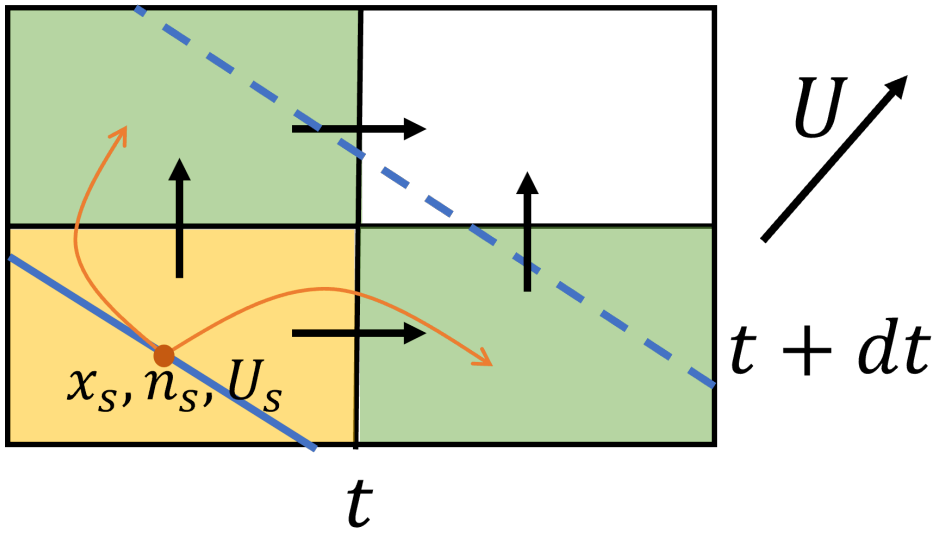


Omitted face

Downwind neighbour cells that are not face-neighbours to the interface cells are not treated correctly by isoAdvector.

Motivation (a solution)

One way to overcome this issue could be to extend the advection stencil so that more *downwind-to-interface* cells are considered in the isoAdvection process. To identify such cells, the flux direction can be used



Extension sketch

the green cells are downwind to the interface cell which is shown in yellow. For isoAdvector to be applied on downwind-to-interface cells, it is required to inherit an interface centre, an interface normal, and interface advection velocity from their upwind neighbours. The upwind neighbours are referred as *donors*.

Motivation (a solution)

1. In every downwind face f of a cell C_P the ΔV_f is initialized as $\Delta V_f = \alpha_P \phi_f \Delta t$.
2. All the interface cells are identified through the condition $\varepsilon < \alpha(t) < 1 - \varepsilon$, where ε is a tolerance specified by the user.
3. Do for each surface cell:
   1. Determine the isoface based on the based on the reconstruction procedure.
   2. Estimate the isoface motion within the timestep Δt according to the aforementioned advection step.
4. Do for each downwind-to-interface cell:
   1. Inherit $(\mathbf{x}_s, \hat{\mathbf{n}}_s, \mathbf{U}_s)$ from a chosen upwind *donor*.
   2. Estimate the isoface motion within the timestep Δt according to the advection step.
5. For each cell update the volume fraction.
6. For every cell C_P with $\alpha_P(t) \notin [0, 1]$ apply the bounding procedure.

Implementation

isoAdvector's functionality is placed in `$FOAM_SRC/transportModels/geometricVoF` The files that we are interested in are

- `isoAdvection.H`
- `isoAdvectionTemplates.C`
- `isoAdvection.C`

isoAdvection::advect

This is the backbone function of the isoAdvector.

```
template<class SpType, class SuType>
void Foam::isoAdvection::advect(const SpType& Sp, const SuType& Su)
{
    DebugInFunction << endl;

    scalar advectionStartTime = mesh_.time().elapsedCpuTime();

    scalar rDeltaT = 1/mesh_.time().deltaTValue();

    // reconstruct the interface
    surf_->reconstruct();

    // Initialising dVf with upwind values
    // i.e. phi[facei]*alpha1[upwindCell[facei]]*dt
    dVf_ = upwind<scalar>(mesh_, phi_).flux(alpha1_)*mesh_.time().deltaT();

    // Do the isoAdvection on surface cells
    timeIntegratedFlux();
```

Listing: Definition of `isoAdvection::advect()`.

isoAdvection::advect

```
    // Advect the free surface
    alpha1_.primitiveFieldRef() =
    (
        alpha1_.oldTime().primitiveField()*rDeltaT
      + Su.field()
      - fvc::surfaceIntegrate(dVf_)().primitiveField()*rDeltaT
    )/(rDeltaT - Sp.field());
```

Listing: Definition of `isoAdvection::advect()`.

$$\alpha_P\left(t+\Delta t\right)=\alpha_P\left(t\right)-\frac{1}{V_P}\sum_f \Delta V_f\left(t,\Delta t\right)$$

isoAdvection::timeIntegratedFlux

```
void Foam::isoAdvection::timeIntegratedFlux()
{
    // Get time step
    const scalar dt = mesh_.time().deltaTValue();

    // Create object for interpolating velocity to isoface centres
    interpolationCellPoint<vector> UInterp(U_);

    // For each downwind face of each surface cell we "isoadvect" to find dVf
    label nSurfaceCells = 0;

    // Clear out the data for re-use and reset list containing information
    // whether cells could possibly need bounding
    clearIsoFaceData();

    // Get necessary references
    const scalarField& phiIn = phi_.primitiveField();
    const scalarField& magSfIn = mesh_.magSf().primitiveField();
    scalarField& dVfIn = dVf_.primitiveFieldRef();
```

Listing: Definition of `isoAdvection::timeIntegratedFlux`

isoAdvection::timeIntegratedFlux

```
    const DynamicField<label>& interfaceLabels = surf_->interfaceLabels();

    // Loop through cells
    forAll(interfaceLabels, i)
    {
        const label celli = interfaceLabels[i];
        if (mag(surf_->normal()[celli]) != 0)
        {

            // This is a surface cell, increment counter, append and mark cell
            nSurfaceCells++;
            surfCells_.append(celli);

            DebugInfo
                << "\n------------ Cell " << celli << " with alpha1 = "
                << alpha1In_[celli] << " and 1-alpha1 = "
                << 1.0 - alpha1In_[celli] << " ------------"
                << endl;
```

Listing: Definition of `isoAdvection::timeIntegratedFlux`

isoAdvection::timeIntegratedFlux

```
            // Cell is cut
            const point x0 = surf_->centre()[celli];
            vector n0 = -surf_->normal()[celli];
            n0 /= (mag(n0));

            // Get the speed of the isoface by interpolating velocity and
            // dotting it with isoface unit normal
            const scalar Un0 = UInterp.interpolate(x0, celli) & n0;
```

Listing: Definition of `isoAdvection::timeIntegratedFlux`

isoAdvection::timeIntegratedFlux

```
            const cell& celliFaces = cellFaces[celli];
            forAll(celliFaces, fi)
            {
                const label facei = celliFaces[fi];

                if (mesh_.isInternalFace(facei))
                {
                    bool isDownwindFace = false;
```

Listing: Definition of `isoAdvection::timeIntegratedFlux`

isoAdvection::timeIntegratedFlux

Determine which interfacial faces are downwind based on the flux direction.

```
if (celli == own[facei])
{
    if (phiIn[facei] >= 0)
    {
        isDownwindFace = true;
    }
}
else
{
    if (phiIn[facei] < 0)
    {
        isDownwindFace = true;
    }
}
```

Listing: Definition of `isoAdvection::timeIntegratedFlux`

isoAdvection::timeIntegratedFlux

Calculate the ΔV_f of a downwind interfacial face.

```
if (isDownwindFace)
{
    dVfIn[facei] = advectFace_.timeIntegratedFaceFlux
    (
        facei,
        x0,
        n0,
        Un0,
        dt,
        phiIn[facei],
        magSfIn[facei]
    );
}
```

Listing: Definition of `isoAdvection::timeIntegratedFlux`

$$\Delta V_f(t, \Delta t) \approx \int_t^{t+\Delta t} \int_{S_f} H(\mathbf{x}, \tau) \frac{\overline{\phi}_f(t)}{A_f} dA d\tau = \overline{\phi}_f(t) \int_t^{t+\Delta t} \underbrace{\frac{1}{A_f} \int_{S_f} H(\mathbf{x}, \tau) dA}_{\alpha_f^+(\tau)} d\tau$$

Extension of isoAdvection::timeIntegratedFlux

Initialize the new lists

```
    // List of cells that are not interface cells but are downwind to an
        interface cell
    DynamicList<label> cellsDownwindToInterface;
    DynamicList<bool> isInterfaceCell;
    // For each cell j in cellsDownwindToInterface donorCells is a list of
        interface cells upwind to cell j.
    DynamicList<DynamicList<label>> donorCells;
    // Populating cellsDownwindToInterface and donorCells
    const cellList& cells = mesh_.cells();
```

Listing: Extension of `isoAdvection::timeIntegratedFlux`

Extension of isoAdvection::timeIntegratedFlux

```
    // Populating isInterfaceCell
    forAll(cells, cellI)
    {

        bool boolValue = false;
        forAll(interfaceLabels, i)
        {
            if(cellI == interfaceLabels[i])
            {
                boolValue = true;
                break;
            }
        }

        isInterfaceCell.append(boolValue);
    }
```

Listing: Extension of `isoAdvection::timeIntegratedFlux`

Extension of isoAdvection::timeIntegratedFlux

For every cell store all its upwind neighbours.

```
    forAll(cells, cellI)
    {
        if (isInterfaceCell[cellI])
        {
            continue;
        }

        DynamicList<label> upwindCells ;
        const cell& celliFaces = cellFaces[cellI];
```

Listing: Extension of `isoAdvection::timeIntegratedFlux`

This list is going to help us afterwards to determine which cells have upwind neighbours belong to the interface.

Extension of isoAdvection::timeIntegratedFlux

Populate the upwindCells based on the face flux orientation.

```
forAll(celliFaces, fi)
{
    label facei = celliFaces[fi];
    if (mesh_.isInternalFace(facei))
    {
        if (cellI == own[facei])
        {
            if (phiIn[facei] >= 0)
            {
                upwindCells.append(mesh_.faceNeighbour()[facei]);
            }
        }
        else
        {
            if (phiIn[facei] < 0)
            {
                upwindCells.append(mesh_.faceNeighbour()[facei]);
            }
        }
```

Listing: Extension of `isoAdvection::timeIntegratedFlux`

Extension of isoAdvection::timeIntegratedFlux

```
        DynamicList<label> upwindInterfaceCells;
        forAll(upwindCells, j)
        {
            const label uCellI = upwindCells[j];

            if(isInterfaceCell[uCellI])
            {
                upwindInterfaceCells.append(uCellI);
                continue;
            }
        }

        if(upwindInterfaceCells.size() > 0)
        {
            cellsDownwindToInterface.append(cellI);
            donorCells.append(upwindInterfaceCells);
        }
    }
```

Listing: Extension of `isoAdvection::timeIntegratedFlux`

Extension of isoAdvection::timeIntegratedFlux

```
    forAll(cellsDownwindToInterface, i)
    {
        label cellI = cellsDownwindToInterface[i];
        DynamicList<label> donors = donorCells[i];

        //At the moment, we apply a very basic implementation where only 1 donor is required
```

Listing: Extension of `isoAdvection::timeIntegratedFlux`

```
        // Now we can use this data to set (x0, n0, Un0) in cellI.
        // A very simple take (probably not final solution) would be to simply let cellI
        // inherrit (x0, n0, Un0) from donor[0]:
        const label cellJ = donors[0];
        const point x0 = surf_->centre()[cellJ];
        vector n0 = -surf_->normal()[cellJ];
        n0 /= (mag(n0));
        const scalar Un0 = UInterp.interpolate(x0, cellJ) & n0;
```

Listing: Extension of `isoAdvection::timeIntegratedFlux`

Extension of isoAdvection::timeIntegratedFlux

```
        const cell& celliFaces = cellFaces[cellI];
        forAll(celliFaces, fi)
        {
            const label facei = celliFaces[fi];
            if (mesh_.isInternalFace(facei))
            {
                bool isDownwindFace = false;
                if (cellI == own[facei])
                {
                    if (phiIn[facei] >= 0)
                    {
                        isDownwindFace = true;
                    }
                }
                else
                {
                    if (phiIn[facei] < 0)
                    {
                        isDownwindFace = true;
                    }
                }
```

Listing: Extension of `isoAdvection::timeIntegratedFlux`

Extension of isoAdvection::timeIntegratedFlux

```
if (isDownwindFace)
{
    dVfIn[facei] = advectFace_.timeIntegratedFaceFlux
    (
        facei,
        x0,
        n0,
        Un0,
        dt,
        phiIn[facei],
        magSfIn[facei]
    );
```

Listing: Extension of `isoAdvection::timeIntegratedFlux`

Hands-on modification

- change to the $WM_PROJECT_USER_DIR directory
  cd $WM_PROJECT_USER_DIR
- Create a new directory inside the $WM_PROJECT_USER_DIR where you can store all the relevant code for this extension.
  mkdir spatialExtensionCode
- Copy the relevant code inside the spatialExtensionCode directory. There are two places that we copy from. The first one is the isAdvector's functionality, geometricVoF, found under the $FOAM_SRC/transportModels/geometricVoF directory.
- The second one is the interIsoFoam solver found under the $FOAM_SOLVERS/multiphase/interIsoFoam directory.
  cp -r $FOAM_SOLVERS/multiphase/interIsoFoam spatialExtensionCode /interIsoExtFoam

Hands-on modification (geometricVoFExt)

- To apply the modification of the `geometricVoFOne` has to change directory first.
  `cd spatialExtensionCode/geometricVoFExt`
- Then it is required to copy the code described in the previous subsections and paste it inside `isoAdvection.C`. The code should be pasted right after line 265.
- In `Make/files` change the name of the compiled library to `LIB = $(FOAM_USER_LIBBIN)/ libgeometricVoFExt`. `Make/options` remain intact.
- Compile `geometricVoFExT`
  `wmake`

Hands-on modification (interIsoExtFoam) 1/2

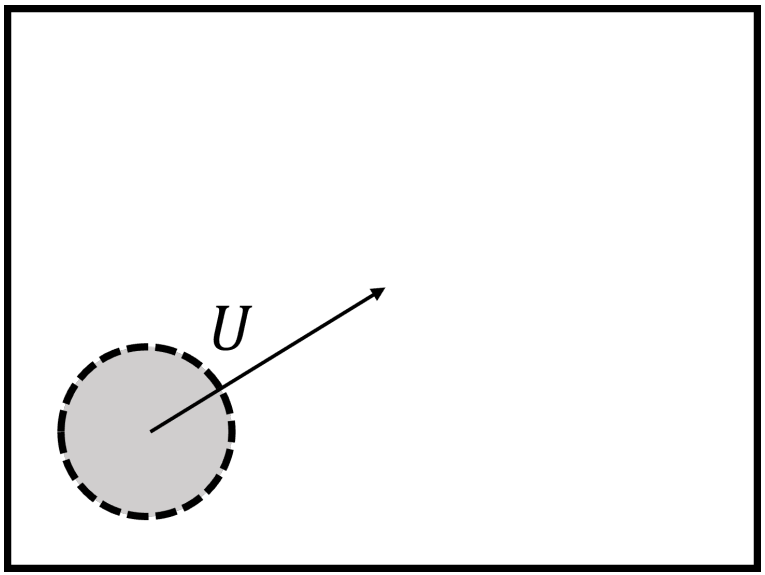
- To modify the `interIsoExtFoam` change directory `cd ../interIsoExtFoam/`
- Rename the source file `mv interIsoFoam.C interIsoExtFoam.C`
- The solver is not modified, as a result the only thing that is required is to couple the new solver to the `geometricVoFExT` instead of the `geometricVoF` library. This will be done inside the `Make` folder.
- In the `Make/options` file and under the `EXE_INC` paths the first two paths should be replaced by their global versions so: `-I../VoF \` is replaced by `-I$(FOAM_SOLVERS)/multiphase/VoF \` and `-I../interFoam\` by `-I$(FOAM_SOLVERS)/multiphase/interFoam\`
- The last path which is related to the geometricVoF has to be replaced as well. `-I$(LIB_SRC)/transportModels/geometricVoF/lnInclude` is replaced by `-I$(WM_PROJECT_USER_DIR)/src/spatialExtensionCode/geometricVoFExt`
  `wmake`

Hands-on modification (interIsoExtFoam)2/2

- The last change that takes place in the `Make/options` file is under the included library binaries, where `-lgeometricVoF` is replaced by `-libgeometricVoFExt` and the `-L$(FOAM_USER_LIBBIN)` that contains the libraries compiled by the user path is added as well.
- The last file that needs to be modifies is the `Make/files` where `interIsoFoam.C` is changed to `interIsoExtFoam.C` and `EXE = $(FOAM_APPBIN)/interIsoFoam` to `EXE = $(FOAM_USER_APPBIN)/interIsoExtFoam`
- Finally we are able compile the new `interIsoExtFoam` solver `wmake`

Taste-Case description

- Two-dimensional pure advection
- Disc with `origin` $(x, y) = (0.5, 0.5)[m]$, `radious` $r = 0.25[m]$
- Uniform velocity field $U = (1, 0.5)[ms^{-1}]$
- Interfacial Courant number $Co_I = 0.9$

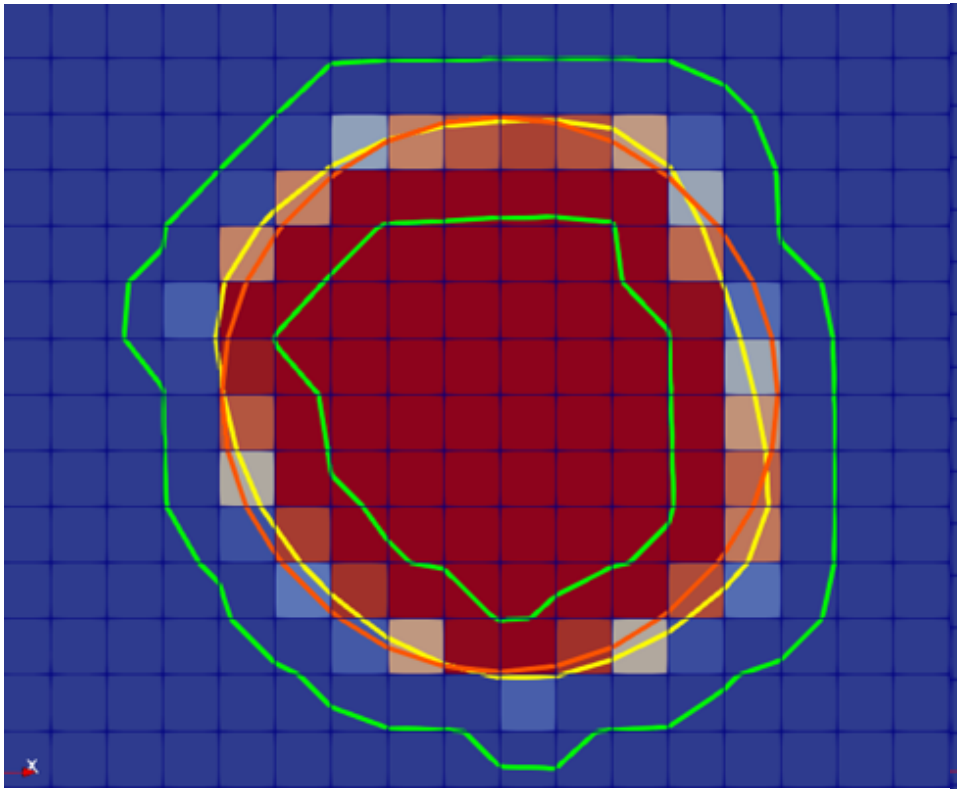


disc simulation sketch

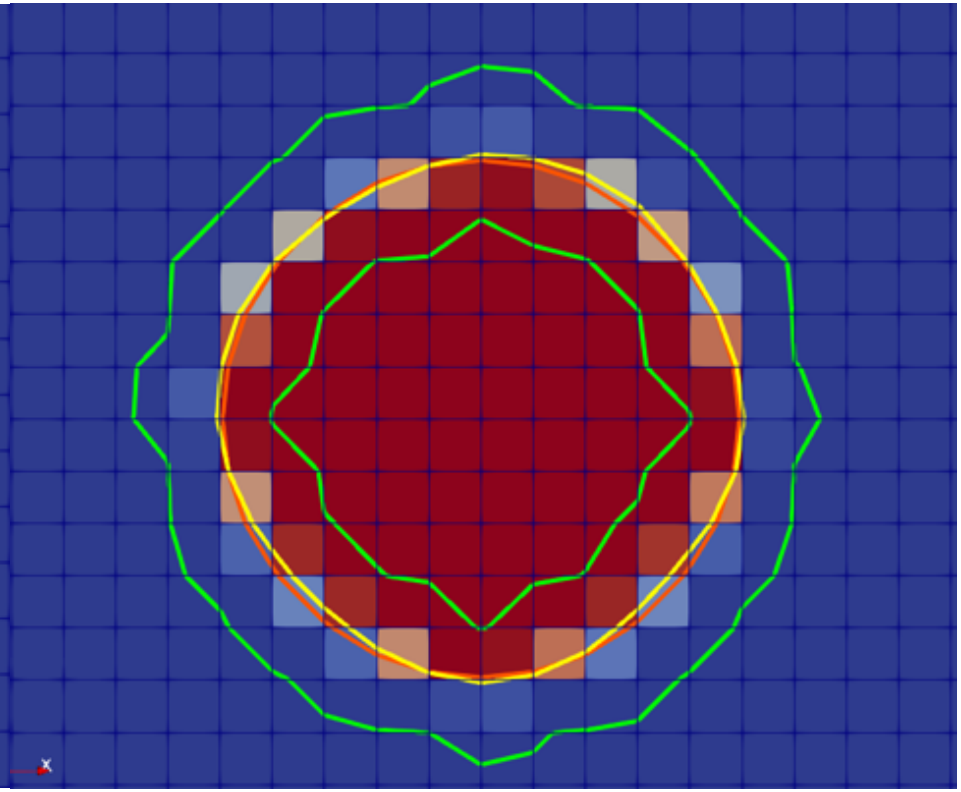
Taste-Case and How to run it

- Restore 0 folder.
- Run blockMesh utility to generate the mesh.
- Run setAlphaFields utility to set the volume fraction in the domain.
- Run interIsoFoam or interIsoExtFoam solver to simulate the case.
- The figures presented below can be generated by a python script and paraView snapshots.

Taste-Case results 1/2

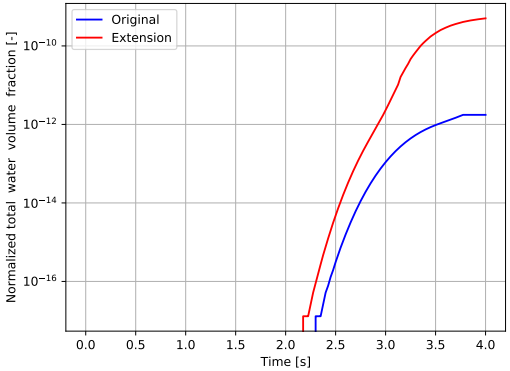


Original

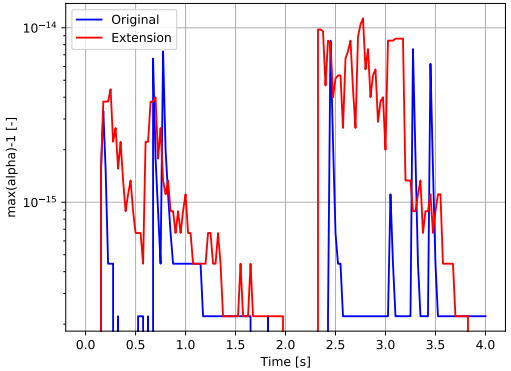


Extension

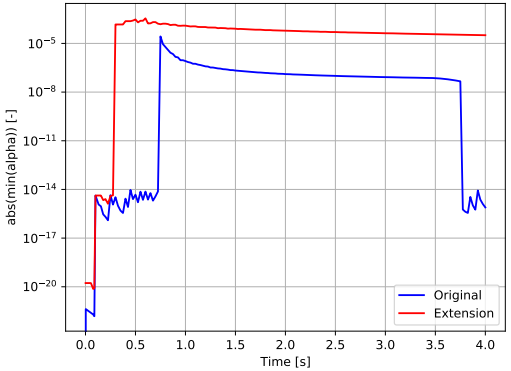
Taste-Case results 2/2



Conservation



Upper-bounding



Lower-bounding

Conclusion

- Brief description of the Interfacial flows and VoF theory
- Analysis of the current implementation of isoAdvector
- Extension of the `isoAdvection::timeIntegratedFlux` to accommodate for cells downwind to the interface. This allows for higher interfacial Courant numbers and enhances shape preservation accuracy.

Further work

- There is plenty of room for code optimization.
- Fix the bounding issue.
- Research how mesh types affect the performance of this extension.
- Simulate more cases (especially 3D).

Q&A

Thank you for your attention!
Questions?