Explanation of `dynamicRefineFvMesh` for adaptive mesh refinement with an extension for independent bulk and interface mesh refinement for two phase simulations.

Yatin Darbar

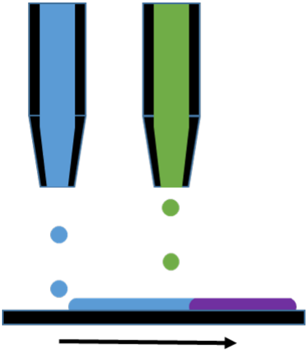
Centre for Doctoral Trainning in Fluid Dynamics,
Universtiy of Leeds,
United Kingdom

University Of Leeds

1. Background
2. Using AMR
3. AMR Code
4. Two–Field AMR
5. Tutorial Case

Reactive Inkjet Printing

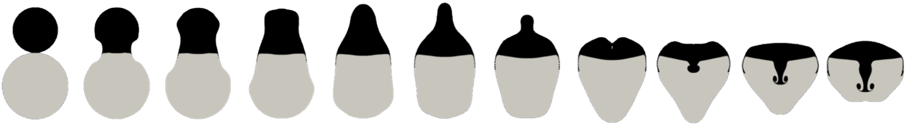
- Additive manufacture (AM) has revolutionised how products can be made.
- However some materials are not fit for AM methods.
- Chemical reactions can be harnessed to created materials on the printing surface.
- The mixing of the chemically reactive droplets is not well understood.



RIJ Example

My Research

- **Project Title**: Mixing dynamics during coalescence of complex fluids.
- Experimentally it is difficult to assess the mixing of droplets.
- Therefore my research is mainly numerical.
- Need an efficient way to capture coalescence and mixing.



Example Simulation of Advective Mixing

Adaptive Mesh Refinement

- Adaptive Mesh Refinement (AMR) is a method of locally adapting the structure of a CFD mesh.
- This can help increase the accuracy of a solution with less significant computational expense.
- Currently OpenFOAM has the functionality to refine in one evolving region, but not two (or more).

My Project

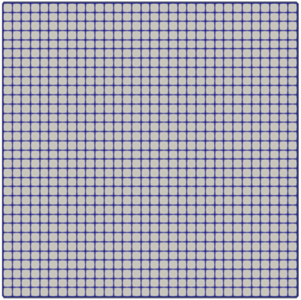
1. Provide a thorough description of the AMR source code.
2. Create a new class capable of refining the mesh for two evolving fields.
3. Extend the unrefinement procedure to add functionality.

Running a simulation with AMR

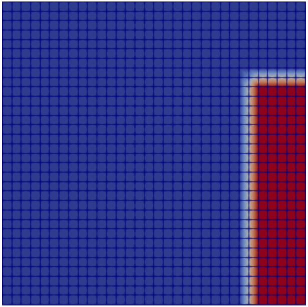
Using the `damBreakWithObstacle` tutorial case we can examine a simulation that uses AMR.

```
cd $FOAM_RUN
cp -r $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreakWithObstacle .
cd damBreakWithObstacle
./Allrun
```

Initial Conditions

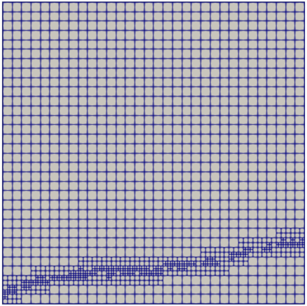


Mesh
uncoloured

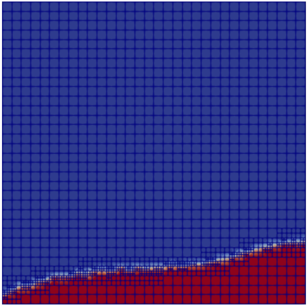


Mesh
coloured by phase fraction

Example Results at Time 0.4 s



Mesh
uncoloured



Mesh
coloured by phase fraction

Case Files

The `damBreakWithObstacle` tutorial contains the following direcotries and files.

```
0
├── U
├── alpha.water
└── p_rgh

constant
├── dynamicMeshDict
├── g
├── momentumTransport
├── polyMesh
└── transportProperties

system
├── blockMeshDict
├── controlDict
├── decomposeParDict
├── fvSchemes
├── fvSolutions
├── setFieldsDict
└── topoSetDict
```

It is the `dynamicMeshDict` that controls the AMR.

The dynamicMeshDict

The dynamicMeshDict contains settings which prescribe and control the mesh refinements.

The dictionary contains the following entries:

- dynamicFvMesh
- refineInterval
- field
- lowerRefineLevel
- upperRefineLevel
- unrefineLevel
- nBufferLayers
- maxRefinemnet
- maxCells
- correctFluxes
- dumpLevel

The dynamicMeshDict

```
dynamicFvMesh   dynamicRefineFvMesh;

// How often to refine
refineInterval  1;

// Field to be refinement on
field           alpha.water;

// Refine field in between lower..upper
lowerRefineLevel 0.001;
upperRefineLevel 0.999;

// If value < unrefineLevel unrefine
unrefineLevel   10;

// Have slower than 2:1 refinement
nBufferLayers   1;

// Refine cells only up to maxRefinement levels
maxRefinement   2;
```

The dynamicMeshDict

```
// Stop refinement if maxCells reached
maxCells        200000;

// Flux field and corresponding velocity field. Fluxes on changed
// faces get recalculated by interpolating the velocity. Use 'none'
// on surfaceScalarFields that do not need to be reinterpolated.
correctFluxes
(
    (phi none)
    (nHatf none)
    (rhoPhi none)
    (alphaPhi0.water none)
    (ghf none)
);

// Write the refinement level as a volScalarField
dumpLevel       true;
```

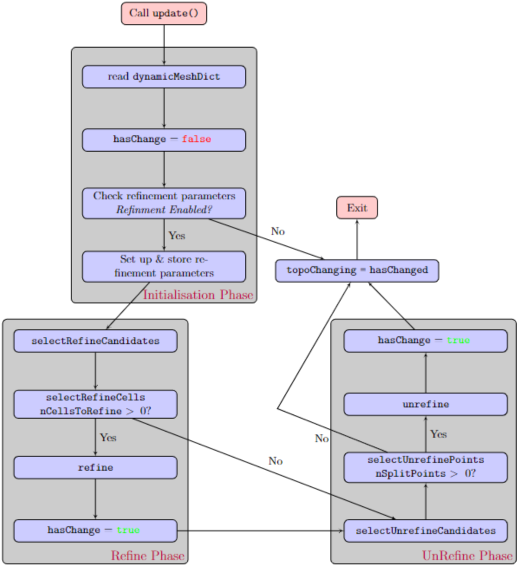
AMR Source Code

- When running the `damBreakWithObstacle` case we only run the `interFoam` solver.
- Within the `interFoam` solver (`interFoam.C`) an object called `mesh` of the `dynamicRefineFvMesh` class is created.
- The CFD mesh is refined by calling the `update` function of the `mesh` object (`mesh.update()` – line 125 of `interFoam.C`).
- The source code for the `dynamicRefineFvMesh` class is found in `$FOAM_SRC/dynamicFvMesh/dynamicRefineFvMesh`.

The update function

There are **three** main sections to the `update` function:

- Initialisation Phase
- Refinement Phase
- Unrefinement Phase



Initialisation Phase

- Mostly concerns creating variables to be used in the latter phases.
- The `dynamicMeshDict` is read and stored as a local dictionary
- The `refineInterval`, `maxCells` and `nBufferLayers` parameters are extracted and checked to ensure they are properly defined.

Initialisation Phase

The dynamicMeshDict is re–read each time–step into the refineDict dictionary.

```
const dictionary refineDict
(
    dynamicMeshDict().optionalSubDict(typeName + "Coeffs")
);
```

Initialisation Phase

For example the `refineInterval` parameter is created and checked by:

```
label refineInterval = refineDict.lookup<label>("refineInterval");

bool hasChanged = false;

if (refineInterval == 0)
{
    topoChanging(hasChanged);

    return false;
}
else if (refineInterval < 0)
{
    FatalErrorInFunction
        << "Illegal refineInterval " << refineInterval << nl
        << "The refineInterval setting in the dynamicMeshDict should"
        << " be >= 1." << nl
        << exit(FatalError);
}
```

In a similar way the `maxCells` and `nBufferLayers` are created and checked.

Refinement Phase

- Cells are selected as candidates to be refined
  - If no cells are to be refined the code moves to the unrefinement phase
- The maximum number of cells that could be refined is calculated
- A list of actual cells to be refined is created.

Refinement Phase

Cells are marked as candidates to be refined using the `selectRefineCandidates` function.

This uses a local `error` function to determine whether to refine a cell

```
scalar err = min(fld[celli] - minLevel, maxLevel - fld[celli]);
```

Notice, cells with a non–zero error value are such that

$$\texttt{minLevel} < \texttt{field} < \texttt{maxLevel}.$$

Refinement Phase

The error value and the `cellLevel` are then used to set the cells that are candidates for refinement:

```
if
(
cellLevel[celli] < maxRefinement
&& cellError[celli] > 0
)
{
candidateCells.set(celli, 1);
```

Refinement Phase

- The number of cells that can be refined without breaking the the `maxCells` limit is calculated assuming each refined cell will cause seven more cells to be created.
- If the number of candidates is larger than `nTotToRefine` then the list of candidates is truncated.
- Else refinement takes place for all the cells.

Unrefinement Phase

- Points are selected as candidates to be removed.
  - If no points are to be removed the code moves to the return statement.
- The points that are marked to be removed are checked to ensure they are not part of any protected areas.
- A list of actual points to be removed is created.

Unrefinement Phase

The unrefinement phase uses the `selectUnrefineCandidates` function to identify points that are to be removed from the mesh. It does this by considering the cells around a point in the mesh.

```
forAll(pointCells(), pointi)
{
    const labelList& pCells = pointCells()[pointi];
    scalar maxVal = -great;
    forAll(pCells, i)
    {
        maxVal = max(maxVal, vFld[pCells[i]]);
    }

    unrefineCandidates[pointi] =
            unrefineCandidates[pointi] && maxVal < unrefineLevel;
}
```

Unrefinement Phase

- After using the `selectUnrefineCandidates` function the candidate points for removal are checked using the `selectUnrefinePoints`.
- It is made sure they do not form part of the cells that have just been refined, or the intermediate layer between refined and unrefined regions of the mesh.
- After this check, if the number of points to be removed from the mesh (`splitPoints`) is non–zero then these points are removed and the mesh is unrefined.

Project Aim

The aim of the adaptations to the source code is to:

- Create a new class capable of refining the mesh for two evolving fields.
- Extend the unrefinement procedure to add functionality.

Method

- Create a new `dynamicDualRefineFvMesh` class by copying the `dynamicRefineFvMesh` class.
- Duplicate the refinement and unrefinement phases so that two evolving regions could be refined on independently.
- Ammend the initialisation phase to ensure that all parameters a checked
- Use the current unrefinement functions as a basis for added unrefinement functionality.

Creation of the `dynamicDualRefineFvMesh` class

A new `dynamicFvMesh` subclass can be created in the following way:

```
cd $WM_PROJECT_USER_DIR
mkdir src/dynamicFvMesh/dynamicDualRefineFvMesh
cd src/dynamicFvMesh/dynamicDualRefineFvMesh
cp -r $FOAM_SRC/dynamicFvMesh/dynamicRefineFvMesh/dynamicRefineFvMesh* .
mv dynamicRefineFvMesh.H dynamicDualRefineFvMesh.H
mv dynamicRefineFvMesh.C dynamicDualRefineFvMesh.C
sed -i 's/dynamicRefineFvMesh/dynamicDualRefineFvMesh/g' dynamicDualRefineFvMesh
    .*
cp -r $FOAM_SRC/dynamicFvMesh/Make .
```

The new class is called `dynamicDualRefineFvMesh` since it will be capable of refining two fields.

Make/files

The Make/files file should read:

```
dynamicDualRefineFvMesh.C

LIB = $(FOAM_USER_LIBBIN)/libdynamicDualRefineFvMesh
```

Make/options

The `Make/options` file should read:

```
EXE_INC = \
 -I$(LIB_SRC)/triSurface/lnInclude \
 -I$(LIB_SRC)/meshTools/lnInclude \
 -I$(LIB_SRC)/dynamicMesh/lnInclude \
 -I$(LIB_SRC)/finiteVolume/lnInclude \
 -I$(LIB_SRC)/dynamicFvMesh/lnInclude

LIB_LIBS = \
 -ltriSurface \
 -lmeshTools \
 -ldynamicMesh \
 -lfiniteVolume \
 -ldynamicFvMesh
```

Method

- Since the `update` function controls the mesh refinement, only this needs to be adapted.
- The reading and refinement for two fields will be added by duplicating the existing code.
- The two refinement fields will be `field1` and `field2` respectively.
- Changes made for one field are similar for the other.
- Adaptations made for one field will be detailed for brevity.

Adaptations to the Initialisation Phase

Recall the reading and checking of the `refineInterval` parameter.

```
label refineInterval = refineDict.lookup<label>("refineInterval");

bool hasChanged = false;

if (refineInterval == 0)
{
    topoChanging(hasChanged);

    return false;
}
else if (refineInterval < 0)
{
    FatalErrorInFunction
        << "Illegal refineInterval " << refineInterval << nl
        << "The refineInterval setting in the dynamicMeshDict should"
        << " be >= 1." << nl
        << exit(FatalError);
}
```

Adaptations to the Initialisation Phase

This is adapted to:

```
label refineInterval1 = refineDict.lookup<label>("refineInterval1");
label refineInterval2 = refineDict.lookup<label>("refineInterval2");

bool hasChanged = false;

if (refineInterval1 == 0 && refineInterval2 == 0)
{
    topoChanging(hasChanged);

    return false;
}
else if (refineInterval1 < 0  || refineInterval2 < 0)
{
    FatalErrorInFunction
        << "Illegal refineInterval " << refineInterval1
        << " | " << refineInterval2 << nl
        << "The refineInterval setting in the dynamicMeshDict should"
        << " be >= 1." << nl
        << exit(FatalError);
}
```

Adaptations to the Refinement Phase

Within the refinement phase (lines 1374–1465 of `dyanmicRefineFvMesh.C`) of the update function, the following variables need to be changed

Original Name	New Name	Type
`refineCells`	`refineCells1`	`PackedBoolList`
`maxRefinement`	`maxRefinement1`	`label`
`selectRefineCandidates`	`selectRefineCandidates1`	`function`
`maxCells`	`maxCells1`	`label`
`nCellsToRefine`	`nCellsToRefine1`	`label`
`cellsToRefine`	`cellsToRefine1`	`labelList`
`nBufferLayers`	`nBufferLayers1`	`label`

Adaptations to the Refinement Phase

Since we have created a new function `selectRefineCandidates1`, this function needs to be declared and defined.

The declaration of the `selectRefineCandidates1` can be added to the `dynamicDualRefineFvMesh.H` file

```
virtual scalar selectRefineCandidates1
(
    PackedBoolList& candidateCell,
    const dictionary& refineDict
) const;
```

Adaptations to the Refinement Phase

The `selectRefineCandidates` function reads some parameters from the `dyanmicMeshDict`:

```
const word fieldName(refineDict.lookup("field"));

 const volScalarField& vFld = lookupObject<volScalarField>(fieldName);

 const scalar lowerRefineLevel =
     refineDict.lookup<scalar>("lowerRefineLevel");
 const scalar upperRefineLevel =
     refineDict.lookup<scalar>("upperRefineLevel");
```

Recall that the `dynamicMeshDict` is locally stored as the `refineDict`

Adaptations to the Refinement Phase

This needs to be amended due to the re–naming of the parameters

```
  const word fieldName(refineDict.lookup("field1"));

   const volScalarField& vFld = lookupObject<volScalarField>(fieldName);

   const scalar lowerRefineLevel =
       refineDict.lookup<scalar>("lowerRefineLevel1");
   const scalar upperRefineLevel =
       refineDict.lookup<scalar>("upperRefineLevel1");
```

Adaptations to the Unrefinement Phase

Within the unrefinement phase (lines 1467–1518 of `dyanmicRefineFvMesh.C`) of the update function, the following variables need to be changed:

Old Name	New Name	Type
`unrefineCandidates`	`unrefineCandidates1`	`boolList`
`selectUnrefineCandidates`	`selectUnrefineCandidates1`	`function`
`pointsToUnrefine`	`pointsToUnrefine1`	`labelList`
`nSplitPoints`	`nSplitPoints1`	`label`
`refineCells`	`refineCells1`	`PackedBoolList`
`unrefineLevel`	`lowerUnrefineLevel`	`label`

Adaptations to the Unrefinement Phase

Again, since we create a new function selectUnrefineCandidates1, this function needs to be declared and defined.

The declaration of the selectUnrefineCandidates1 can be added to the dynamicDualRefineFvMesh.H file

```
void selectUnrefineCandidates1
(
    boolList& unrefineCandidates,
    const dictionary& refineDict
) const;
```

Adaptations to the Unrefinement Phase

- Introduce the `upperRefineLevel` parameter.
- Need to find the minimum value in the cells around a point.
- Then check if this value is larger then `upperRefineLevel`.

Adaptations to the Unrefinement Phase

This process is achieved by the following code:

```
forAll(pointCells(), pointi)
    {
        const labelList& pCells = pointCells()[pointi];
        scalar minVal = great;
        forAll(pCells, i)
        {
            minVal = min(minVal, vFld[pCells[i]]);
        }

        unrefineCandidates[pointi] =
 unrefineCandidates[pointi] && minVal > upperUnrefineLevel
    }
```

Adaptations to the Unrefinement Phase

This addition is added into the existing unrefinement procedure by use of if–statements.

- If both `lowerUnrefineLevel` and `upperUnrefineLevel` are found in the `dynamicMeshDict` then the mesh is unrefined in places in which

$$\texttt{field} < \texttt{lowerUnrefineLevel} \text{ or } \texttt{upperUnrefineLevel} < \texttt{field}$$

- If both only `lowerUnrefineLevel` is found in the `dynamicMeshDict` then the mesh is unrefined in places in which

$$\texttt{field} < \texttt{lowerUnrefineLevel}$$

- If `upperUnrefineLevel` is found in the `dynamicMeshDict` then the mesh is unrefined in places in which

$$\texttt{upperUnrefineLevel} < \texttt{field}$$

damBreak Case

- The same `damBreakWithObstacle` case files will be used to demonstrate the two field refinement capabilites of the `dynamicDualRefineFvMesh`.
- Amendments need to be made to the case files to enable access to the `dynamicDualRefineFvMesh` class.
- Addition parameters need to be specified in the `dynamicMeshDict`.

Case Amendments

Copy a clean version of the `damBreakWithObstacle` Case and allow it to access the `dynamicDualRefineFvMesh` class by:

```
cd $FOAM_RUN
cp -r $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreakWithObstacle ./
    damBreak
cd damBreak
sed -i 's/dynamicRefineFvMesh/dynamicDualRefineFvMesh/g' constant/
    dynamicMeshDict
echo 'libs ("libdynamicDualRefineFvMesh.so");' >> system/controlDict
```

New dynamicMeshDict

- The `dynamicMeshDict` needs to be changed to account for the two field refinement
- In this tutorial we will prescribe
  - `cellLevel` = 1 refinement in the bulk of the water phase
  - `cellLevel` = 2 refinement on the air–water interface

Interface Refinement Settings

The settings to refine the air–water interface in the `dynamicMeshDict` are given by:

```
// --- Field 1 -- Interface --- //
// How often to refine
refineInterval1 1;
// Field to be refinement on
field1 alpha.water;
// Refine field in between lower..upper
lowerRefineLevel1 0.001;
upperRefineLevel1 0.999;
// If value < unrefineLevel unrefine
lowerUnrefineLevel1 0;
upperUnrefineLevel1 0.9;
// Have slower than 2:1 refinement
nBufferLayers1 2;
// Refine cells only up to maxRefinement levels
maxRefinement1 2;
// Stop refinement if maxCells reached
maxCells1 200000;
```

Bulk Water Refinement Settings

The settings to refine the bulk of the water phase in the `dynamicMeshDict` are given by:

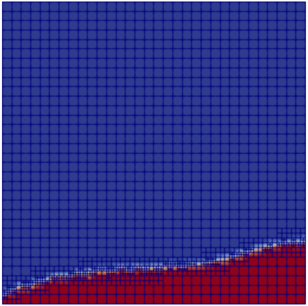
```
// --- Field 2 -- Bulk Water --- //
// How often to refine
refineInterval2 1;
// Field to be refinement on
field2 alpha.water;
// Refine field in between lower..upper
lowerRefineLevel2 0.9;
upperRefineLevel2 1.1;
// If value < unrefineLevel unrefine
lowerUnrefineLevel2 0.001;
// Have slower than 2:1 refinement
nBufferLayers2 1;
// Refine cells only up to maxRefinement levels
maxRefinement2 1;
// Stop refinement if maxCells reached
maxCells2 200000;
```

Running the Case

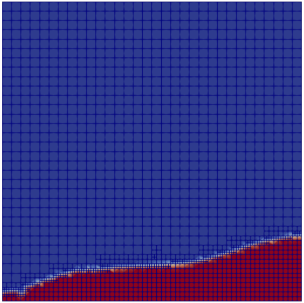
The case can be run by executing:

```
blockMesh
setFields
interFoam
```

Results – 0.4 s



`dynamicRefineFvMesh`



`dynamicDualRefineFvMesh`