Introducing a hybrid rebound and sticking particle-wall interaction model

Johannes Hansson

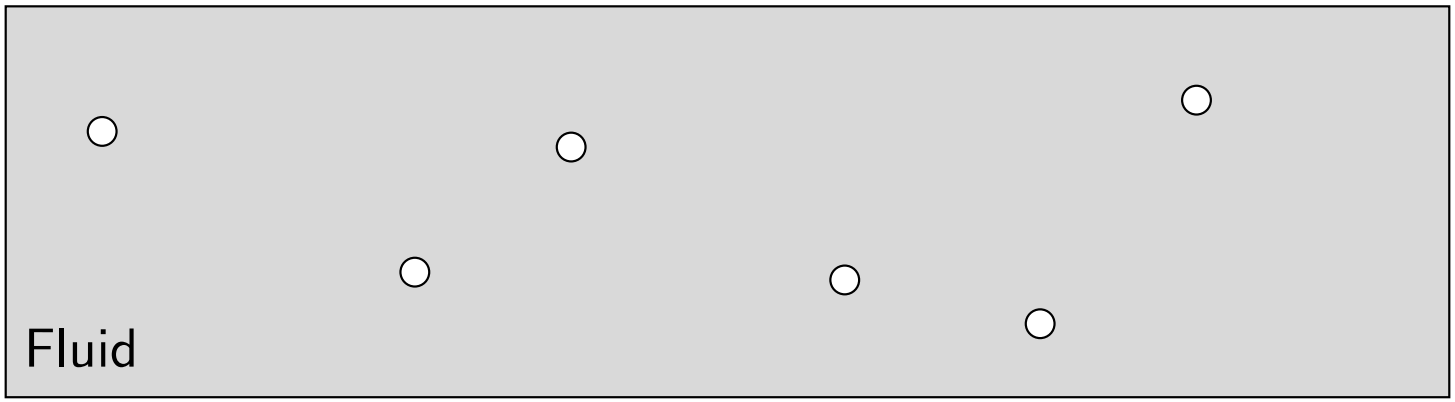
Mechanics and Maritime Sciences/Fluid Dynamics,
Chalmers University of Technology,
Gothenburg, Sweden

2024-01-15

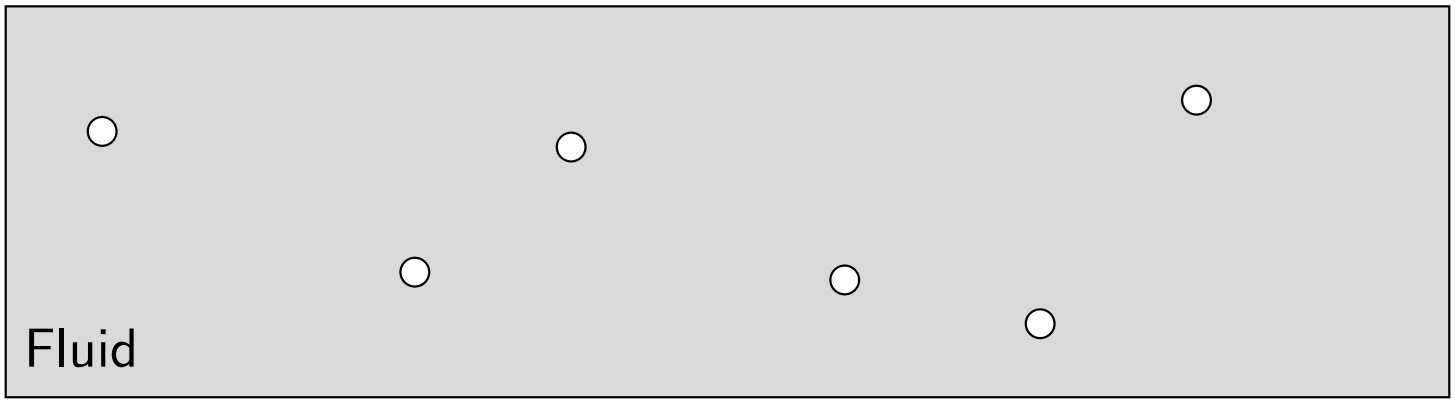
Aim

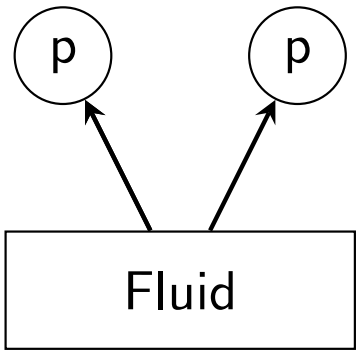
- Outline a method for modifying particle-wall interaction models
- Implement a hybrid particle-wall interaction model based on existing theory

Small particles in carrier fluid

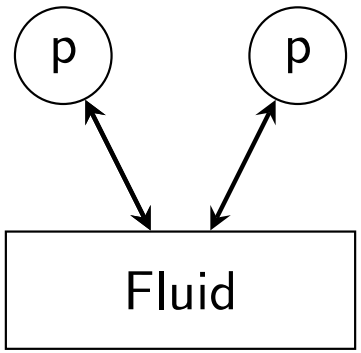


Small particles in carrier fluid

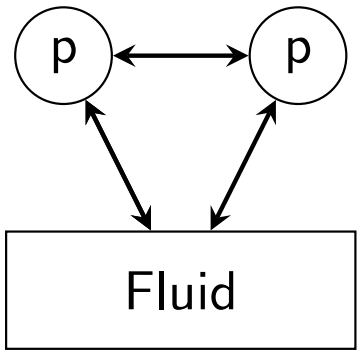




One-way coupling



Two-way coupling



Four-way coupling

Elastic and inelastic collisions

- Elastic collisions:
  - No kinetic energy is lost a collision
  - $|v_{\text{after}}| = |v_{\text{before}}|$
- Inelastic collisions
  - Some kinetic energy is lost in a collision
  - $|v_{\text{after}}| = e|v_{\text{before}}|$
  - e is the coefficient of restitution

Image: MichaelMaggs, Edit by Richard Bartz. CC BY-SA 3.0
https://en.wikipedia.org/wiki/File:Bouncing_ball_strobe_edit.jpg

Available particle-wall interaction models in OpenFOAM

- Stick
  - Particles stick no matter what
- Rebound
  - Particles rebound no matter what, with constant coefficient of restitution
- Escape
  - Particles leave the domain

Problems with currently implemented models

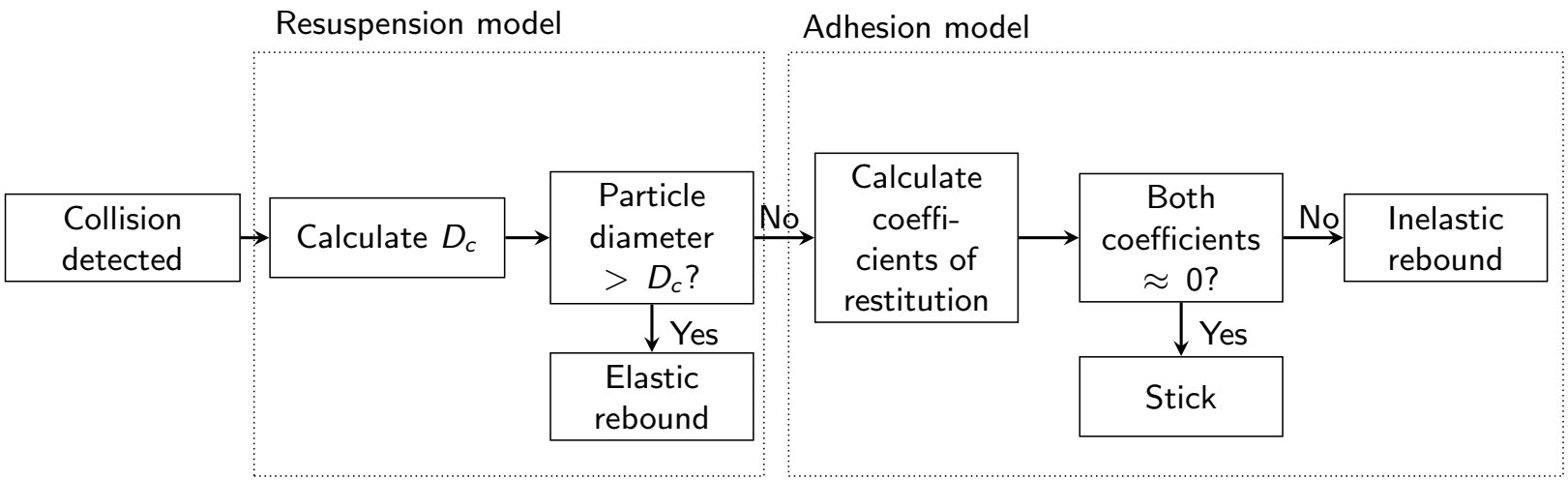
Particle-wall collision models in OpenFOAM have a few issues.

- Implemented models are too idealized, not representative of real-world behavior of particles like snow or ash
- Not all collisions for a certain material are the same

There are many more models that are more advanced.

- Snow: Eidevåg et al. 2022 [1]
- Ash in boilers: Srinivasachar et al. 1991 [2]

Eidevåg et al. model



Eidevåg et al. model

$$D_\mathrm{c} = a u_\tau^b$$

$$e_\mathrm{n} = e_\mathrm{qe} \sqrt{1 - \left(\frac{V_\mathrm{c,n}}{\max\left(V_\mathrm{i}, V_\mathrm{c,n}\right)} \right)^2}$$

$$e_\mathrm{t} = 1 - \mu = e_\mathrm{qe} \sqrt{1 - \left(\frac{V_\mathrm{c,t}}{\max\left(V_\mathrm{i}, V_\mathrm{c,t}\right)} \right)^2}$$

$$V_\mathrm{c,n} = \sqrt{\frac{2E_\mathrm{s}}{m_\mathrm{p}}}$$

$$V_\mathrm{c,t} \approx 0.23 \left(\frac{\Delta\gamma}{\gamma} \right) V_\mathrm{c,n}$$

Particle tracking in OpenFOAM

Particle tracking is already available in OpenFOAM.

To activate:

1. Declare the function object in the `system/controlDict` file
2. Declare particle tracking settings in the `constant/kinematicCloudProperties` file
3. Declare settings for gravity in the `constant/g` file

Particle tracking in OpenFOAM

system/controlDict

```
//controlDict contents...

functions
{
    tracks
    {
        type    icoUncoupledKinematicCloud;
        libs    (lagrangianFunctionObjects);
    }
}
```

Particle tracking in OpenFOAM

constant/kinematicCloudProperties

```
//...

solution
{
    active          true;
    coupled         false;
    transient       yes;
    cellValueSourceCorrection off;
    //...

}

//...

subModels
{
    //...

    patchInteractionModel localInteraction;

    localInteractionCoeffs
    {
        patches
        (
        //...
        "(cylinder)"
        {
            type stick;
        }
        );
    }
    //...
}
```

Particle tracking in OpenFOAM

constant/g

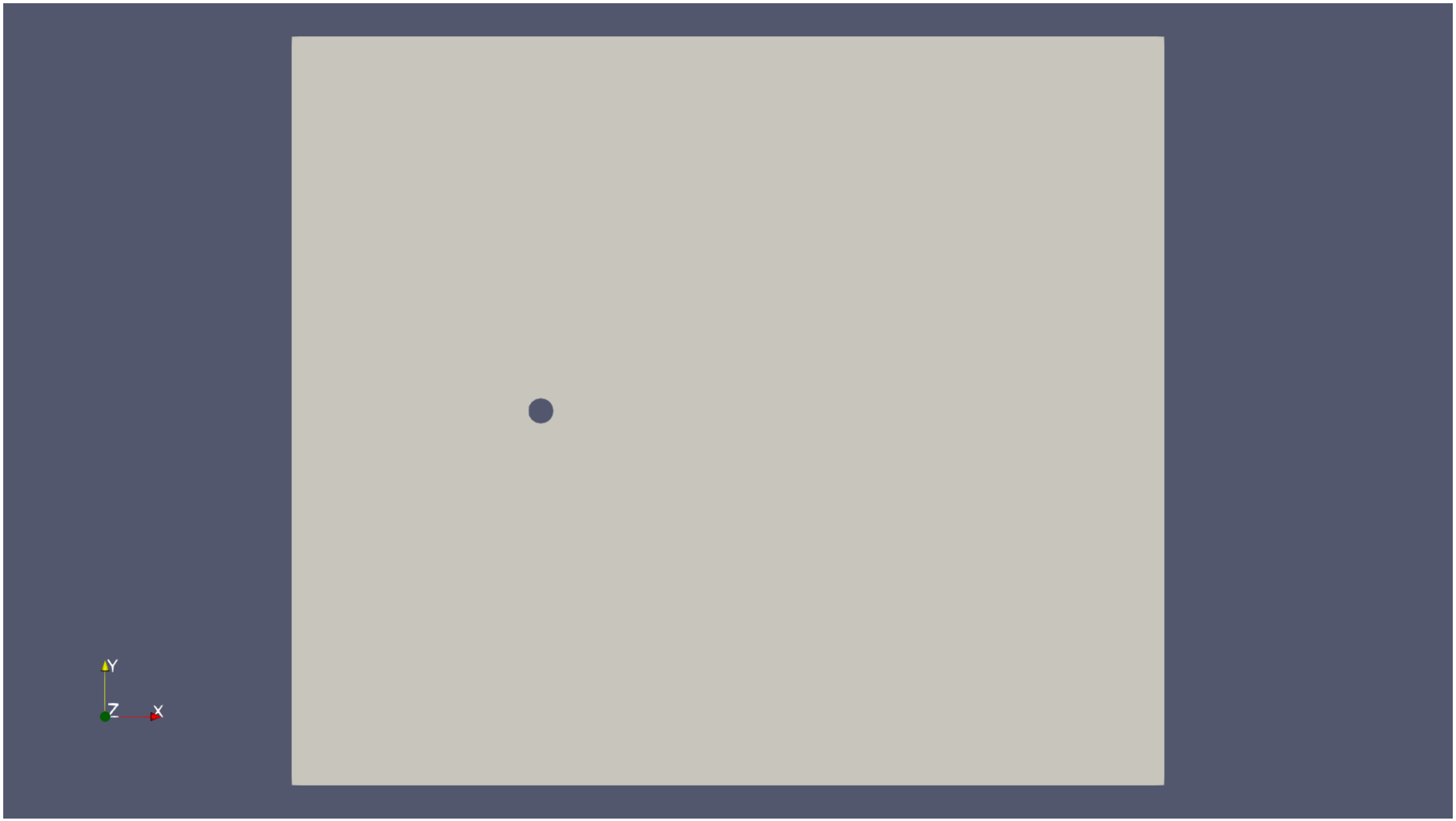
```
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0 0 -9.81);


// ************************************************************************* //
```

Particle tracking in OpenFOAM

- Can now run case as usual, with particle tracking enabled
- Works with most transient solvers in OpenFOAM
- Function object structure very flexible

Particle tracking in OpenFOAM

Example case in this work: flow around a cylinder at $\mathrm{Re} = 200\,000$



Particle tracking in OpenFOAM

Running a case and looking at the log file output

```
Time = 0.01

Courant Number mean: 0.0202404 max: 0.809068
smoothSolver:  Solving for Ux, Initial residual = 1, Final residual = 4.54416e-08, No Iterations 11
smoothSolver:  Solving for Uy, Initial residual = 1, Final residual = 3.83003e-08, No Iterations 10
GAMG:  Solving for p, Initial residual = 1, Final residual = 0.0341452, No Iterations 4
time step continuity errors : sum local = 4.59568e-07, global = -6.63446e-08, cumulative = -6.63446e-08
GAMG:  Solving for p, Initial residual = 0.134128, Final residual = 0.00416861, No Iterations 2
time step continuity errors : sum local = 1.49621e-07, global = -3.34946e-08, cumulative = -9.98391e-08
GAMG:  Solving for p, Initial residual = 0.0280719, Final residual = 6.94414e-07, No Iterations 12
time step continuity errors : sum local = 2.80701e-11, global = -4.81325e-12, cumulative = -9.9844e-08
smoothSolver:  Solving for omega, Initial residual = 0.00259155, Final residual = 3.70316e-08, No Iterations 4
smoothSolver:  Solving for k, Initial residual = 1, Final residual = 4.6953e-08, No Iterations 6
bounding k, min: 0 max: 1.18753e-15 average: 1.00278e-15
ExecutionTime = 0.06 s  ClockTime = 0 s

#...
```

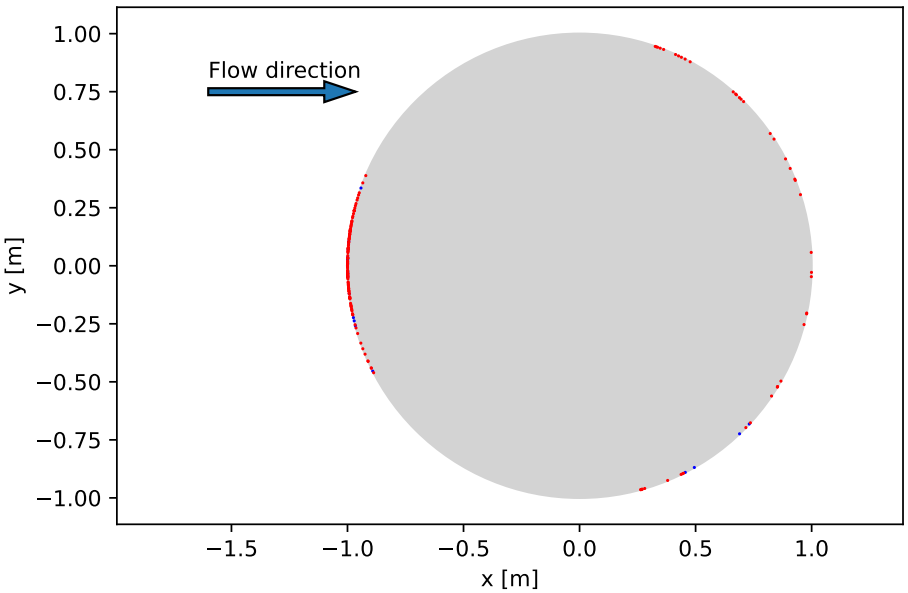
Particle tracking in OpenFOAM

```
#...

Solving2-D cloud kinematicCloud

Cloud: kinematicCloud injector: injection
Added 10 new parcels

Cloud: kinematicCloud
Current number of parcels       = 10
Current mass in system          = 5.17243e-09
Linear momentum                 = (5.1724e-09 -3.37811e-14 0)
|Linear momentum|                = 5.1724e-09
Linear kinetic energy           = 2.58619e-09
Average particle per parcel     = 1
Injector injection:
- parcels added               = 10
- mass introduced             = 5.17243e-09
Parcel fate: system (number, mass)
- escape                      = 0, 0
Parcel fate: patch (sides) (number, mass)
- escape                      = 0, 0
- stick                       = 0, 0
Parcel fate: patch (inlet|outlet|injectionPatch) (number, mass)
- escape                      = 0, 0
- stick                       = 0, 0
Parcel fate: patch (cylinder) (number, mass)
- escape                      = 0, 0
- stick                       = 0, 0
Rotational kinetic energy       = 0
```

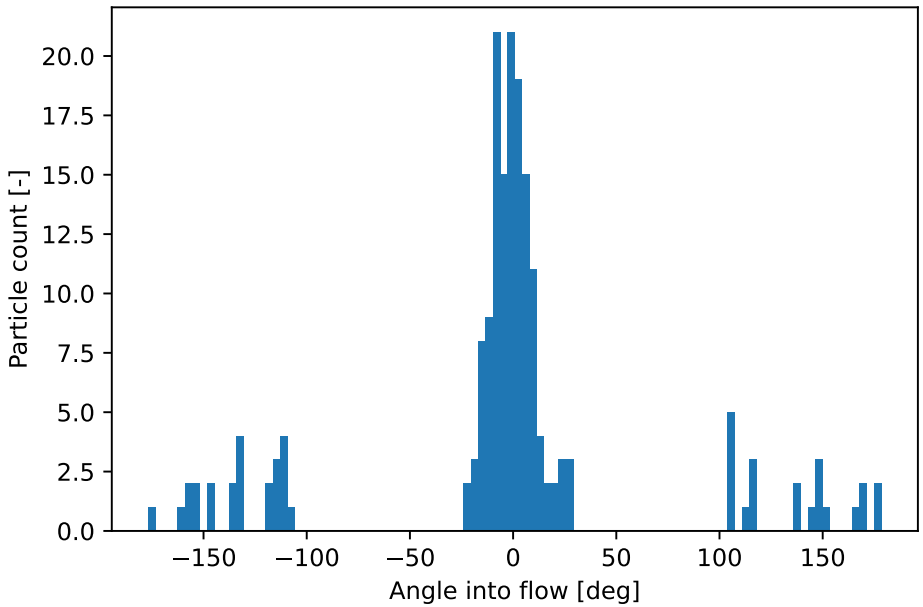
Particle tracking in OpenFOAM

`stick` collision model

Particle positions



Particle deposition density



Red: deposited
Blue: free-floating

Particle tracking: existing implementation

Collision behavior controlled by switch statement in the `correct` function in the `LocalInteraction.C` file of the `intermediate` library.

`LocalInteraction.C`

```
//...

switch (it)
{
    case PatchInteractionModel<CloudType>::itNone:
    {
        //...
    }
    case PatchInteractionModel<CloudType>::itEscape:
    {
        //...
    }
    case PatchInteractionModel<CloudType>::itStick:
    {
        //...
    }
    case PatchInteractionModel<CloudType>::itRebound:
    {
        //...
    }
    default:
    {
        //...
    }
}
```

Particle tracking: existing implementation

Enums are defined in `PatchInteractionModel.H`

`PatchInteractionModel.H`

```
63 template<class CloudType>
64 class PatchInteractionModel
65 :
66     public CloudSubModelBase<CloudType>,
67     public functionObjects::writeFile
68 {
69 public:
70
71     // Public enumerations
72
73         // Interaction types
74         enum interactionType
75         {
76             itNone,
77             itRebound,
78             itStick,
79             itEscape,
80             itOther
81         };
82
83         static wordList interactionTypeNames_;
```

Particle tracking: escape condition

LocalInteraction.C

```
46 case PatchInteractionModel<CloudType>::itEscape:
47 {
48     keepParticle = false;
49     p.active(false);
50     U = Zero;
51
52     const scalar dm = p.mass()*p.nParticle();
53
54     nEscape_[patchi][idx]++;
55     massEscape_[patchi][idx] += dm;
56
57     if (writeFields_)
58     {
59         const label pI = pp.index();
60         const label fI = pp.whichFace(p.face());
61         massEscape().boundaryFieldRef()[pI][fI] += dm;
62     }
63     break;
64 }
```

Particle tracking: stick condition

LocalInteraction.C

```
65 case PatchInteractionModel<CloudType>::itStick:
66 {
67     keepParticle = true;
68     p.active(false);
69     U = Zero;
70
71     const scalar dm = p.mass()*p.nParticle();
72
73     nStick_[patchi][idx]++;
74     massStick_[patchi][idx] += dm;
75
76     if (writeFields_)
77     {
78         const label pI = pp.index();
79         const label fI = pp.whichFace(p.face());
80         massStick().boundaryFieldRef()[pI][fI] += dm;
81     }
82     break;
83 }
```

Particle tracking: rebound condition

LocalInteraction.C

```
84  case PatchInteractionModel<CloudType>::itRebound:
85  {
86      keepParticle = true;
87      p.active(true);
88
89      vector nw;
90      vector Up;
91
92      this->owner().patchData(p, pp, nw, Up);
93
94      // Calculate motion relative to patch velocity
95      U -= Up;
96
97      if (mag(Up) > 0 && mag(U) < this->Urmax())
98      {
99          WarningInFunction
00          << "Particle U the same as patch "
01          << "    The particle has been removed" << nl << endl;
02
03          keepParticle = false;
04          p.active(false);
05          U = Zero;
06          break;
07      }
08
09      scalar Un = U & nw;
10      vector Ut = U - Un*nw;
11
12      if (Un > 0)
13      {
14          U -= (1.0 + patchData_[patchi].e())*Un*nw;
15      }
16
17      U -= patchData_[patchi].mu()*Ut;
18
19      // Return velocity to global space
20      U += Up;
21
22      break;
23  }
```

Particle-wall collision types: special cases

LocalInteraction.C

```
case PatchInteractionModel<CloudType>::itNone:
{
    return false;
}

//...

default:
{
    FatalErrorInFunction
    << "Unknown interaction type "
    << patchData_[patchi].interactionTypeName()
    << "(" << it << ") for patch "
    << patchData_[patchi].patchName()
    << ". Valid selections are:" << this->interactionTypeNames_
    << endl << abort(FatalError);
}
```

Creating a new collision model

1. Copy the `intermediate` and Lagrangian function objects libraries to user path
2. Create a new model based on the `LocalInteraction` type
3. Add new files to compile
4. Compile the libraries

Creating a new collision type: copy files to user path and compile

```
cd $WM_PROJECT_USER_DIR
cp -r $FOAM_SRC/lagrangian/intermediate .
```

```
cd $WM_PROJECT_USER_DIR/intermediate/Make
sed -i 's/FOAM_LIBBIN/FOAM_USER_LIBBIN/' files
```

```
cd $WM_PROJECT_USER_DIR/intermediate
wmake      #wmake -j 8    for 8-thread parallel compilation
```

We now have a user-local copy of the `intermediate` library.

Creating a new collision type: copy files to user path and compile

```
cd $WM_PROJECT_USER_DIR
cp -r $FOAM_SRC/functionObjects/lagrangian .
cd $WM_PROJECT_USER_DIR/lagrangian/Make
sed -i 's/FOAM_LIBBIN/FOAM_USER_LIBBIN/' files
sed -i 's|-I$(LIB_SRC)/lagrangian/intermediate/lnInclude|-I$(
    WM_PROJECT_USER_DIR)/intermediate/lnInclude|' options
cd $WM_PROJECT_USER_DIR/lagrangian
wmake
```

We now have a user-local copy of the Lagrangian function objects library that links to our new `intermediate` library.
If we want we can try using the user-local version of the library to check that everything works.

Creating a new collision type: create new model based on the LocalInteraction type

```
cd $WM_PROJECT_USER_DIR/intermediate/submodels/Kinematic/PatchInteractionModel
mkdir LocalInteractionMix
cp LocalInteraction/LocalInteraction.H LocalInteractionMix/LocalInteractionMix.
    H
cp LocalInteraction/LocalInteraction.C LocalInteractionMix/LocalInteractionMix.
    C
```

```
cd $WM_PROJECT_USER_DIR/\
intermediate/submodels/Kinematic/PatchInteractionModel/LocalInteractionMix
sed -i 's/LocalInteraction/LocalInteractionMix/g' LocalInteractionMix.H
sed -i 's/localInteraction/localInteractionMix/g' LocalInteractionMix.H
sed -i 's/LocalInteraction/LocalInteractionMix/g' LocalInteractionMix.C
```

Creating a new collision type: create new model based on the LocalInteraction type

```
cd $WM_PROJECT_USER_DIR/intermediate/parcels/include/
sed -i '/"LocalInteraction.H"/{p;s|"LocalInteraction.H"|"LocalInteractionMix.H
    "|;}'\
makeParcelPatchInteractionModels.H
sed -i '/LocalInteraction,/{p;s|LocalInteraction,|LocalInteractionMix,|;}'\
makeParcelPatchInteractionModels.H
```

```
cd $WM_PROJECT_USER_DIR/intermediate/
wmake
```

Testing the new collision model

It's a good idea to test the changes so far. Can run the same case as before, but with two lines changed.

`kinematicCloudProperties`

```
patchInteractionModel localInteractionMix; //Change 1: added "...Mix"

localInteractionMixCoeffs //Change 2: added "...Mix..."
{
    patches
    (

        //...

    );
}
```

If we run the case we see that we get exactly the same simulation result as before, as expected.

Creating a new collision type

PatchInteractionModel.H

```
63 template<class CloudType>
64 class PatchInteractionModel
65 :
66     public CloudSubModelBase<CloudType>,
67     public functionObjects::writeFile
68 {
69 public:
70
71     // Public enumerations
72
73         // Interaction types
74         enum interactionType
75         {
76             itNone,
77             itRebound,
78             itStick,
79             itReboundStickMix,
80             itEscape,
81             itOther
82         };
```

PatchInteractionModel.C

```
36 template<class CloudType>
37 Foam::wordList Foam::PatchInteractionModel<CloudType>::interactionTypeNames_
38 {
39     "rebound", "stick", "reboundStickMix", "escape"
40 };
```

Creating a new collision type

PatchInteractionModel.C

```
59  template<class CloudType>
60  Foam::word Foam::PatchInteractionModel<CloudType>::interactionTypeToWord
61  (
62      const interactionType& itEnum
63  )
64  {
65      word it = "other";
66
67      switch (itEnum)
68      {
69          case itNone:
70          {
71              it = "none";
72              break;
73          }
74          case itRebound:
75          {
76              it = "rebound";
77              break;
78          }
79          case itStick:
80          {
81              it = "stick";
82              break;
83          }
84          case itReboundStickMix:
85          {
86              it = "reboundStickMix";
87              break;
88          }
89          case itEscape:
90          {
91              it = "escape";
92              break;
93          }
94          default:
95          {
96          }
97      }
98
99      return it;
100 }
```

Creating a new collision type

PatchInteractionModel.C

```
03 template<class CloudType>
04 typename Foam::PatchInteractionModel<CloudType>::interactionType
05 Foam::PatchInteractionModel<CloudType>::wordToInteractionType
06 (
07     const word& itWord
08 )
09 {
10     if (itWord == "none")
11     {
12         return itNone;
13     }
14     if (itWord == "rebound")
15     {
16         return itRebound;
17     }
18     else if (itWord == "stick")
19     {
20         return itStick;
21     }
22     else if (itWord == "reboundStickMix")
23     {
24         return itReboundStickMix;
25     }
26     else if (itWord == "escape")
27     {
28         return itEscape;
29     }
30     else
31     {
32         return itOther;
33     }
34 }
```

Creating a new collision type

LocalInteractionMix.C

```
switch (it)
{
    case PatchInteractionModel<CloudType>::itNone:
    {
        //...
    }
    case PatchInteractionModel<CloudType>::itEscape:
    {
        //...
    }
    case PatchInteractionModel<CloudType>::itStick:
    {
        //...
    }
    case PatchInteractionModel<CloudType>::itRebound:
    {
        //...
    }
    case PatchInteractionModel<CloudType>::itReboundStickMix:
    {

    }
    default:
    {
        //...
    }
}
```

Creating a new collision type

`kinematicCloudProperties`

```
"(cylinder)"
{
    type reboundStickMix;
}
```

We have now defined all the infrastructure needed for our particle-wall collision model.
However, still no code in the actual model.

An ad-hoc model for testing the implementation

To test our code structure we can implement an ad-hoc model that gives very clear indications ig we are doing things correctly.

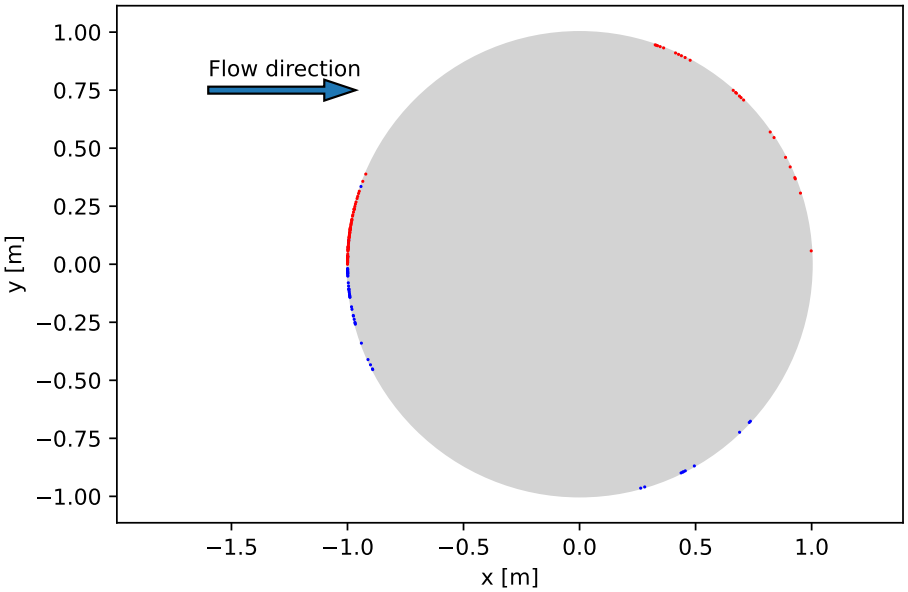
$$\begin{cases} \text{stick} & \text{if } y_\text{p} \geq 0 \\ \text{rebound} & \text{if } y_\text{p} < 0 \end{cases}$$

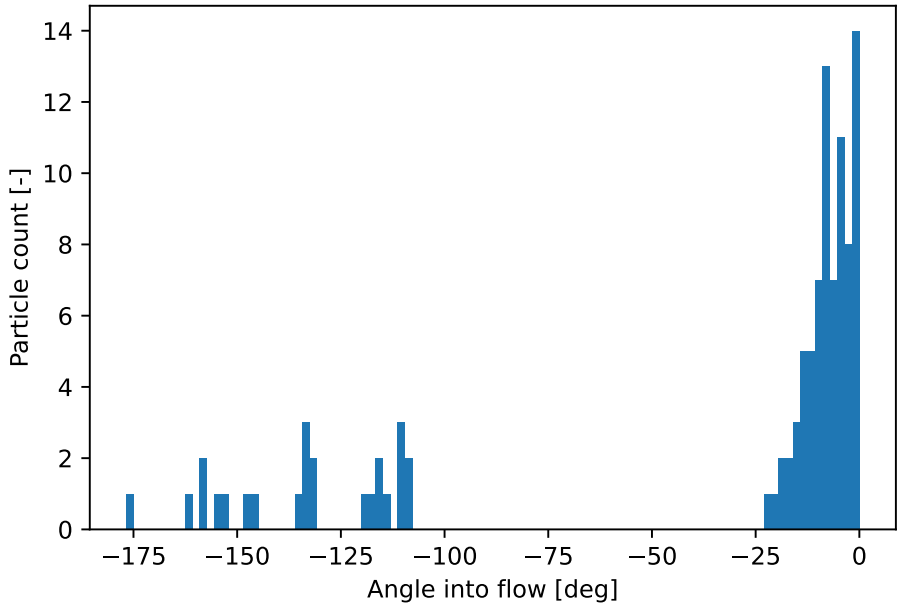
An ad-hoc model for testing the implementation

LocalInteractionMix.C

```
case PatchInteractionModel<CloudType>::itReboundStickMix:
{
    if (p.position()[1]<0)//proof-of-concept, rebound if particle y position is negative, stick otherwise
    //very ad hoc, used just to show the concept
    {
        //rebound code ...
    }
    else
    {
        //stick code ...
    }
```

Testing the ad-hoc model





Red: deposited
Blue: free-floating

Implementing the snow collision model

We now know that can modify the collision model and make it do what we want. We will now properly implement the chosen snow collision model by Eidevåg et al.

LocalInteractionMix.C

```
//Resuspension criterion.
//If particle diameter is >= D_c, then we always get rebound.
//Else, follow adhesion model
if (p.d() >= D_c)
{
    //Particle will resuspend
    //Always rebound with e_n=e_t=1
    e_n = 1;
    e_t = 1;
    rebound(p,pp,keepParticle,patchi,U,idx,e_n,e_t);
}
else
{
    //check adhesion model
    if (e_n < SMALL && e_t < SMALL)
    {
        //stick
        stick(p, pp, keepParticle, patchi, U, idx);
    }
    else
    {
        //rebound, variable e_n, e_t
        rebound(p,pp,keepParticle,patchi,U,idx,e_n,e_t);
    }
}
```

Resuspension model: calculating the critical diameter D_c

LocalInteractionMix.C

```
const volVectorField& wallShearStressMeanField = mesh_.lookupObject<volVectorField>("wallShearStressMean");
const volSymmTensorField& wallShearStressPrime2MeanField = mesh_.lookupObject<volSymmTensorField>("wallShearStressPrime2Mean");

autoPtr<interpolation<vector>> interpolatorMean = interpolation<vector>::New("cellPoint", wallShearStressMeanField);
auto wallShearStressMeanPointValue = interpolatorMean->interpolate(p.position(), p.cell());

autoPtr<interpolation<symmTensor>> interpolatorPrime2Mean = interpolation<symmTensor>::New("cellPoint",
       wallShearStressPrime2MeanField);
auto wallShearStressPrime2MeanPointValue = interpolatorPrime2Mean->interpolate(p.position(), p.cell());

scalar D_c = a*pow(sqrt(mag(wallShearStressMeanPointValue) +
    3*sqrt(wallShearStressPrime2MeanPointValue.xx() + wallShearStressPrime2MeanPointValue.yy() +
       wallShearStressPrime2MeanPointValue.zz())), b);
```

Adhesion model: calculating the coefficients of restitution

LocalInteractionMix.C

```
//adhesion model
scalar W = 0.218; //J/m^2, work of adhesion
scalar E_star = 5.4e9; //Pa, effective Young's modulus
scalar Dgamma_gamma = 1; //[-], adhestion hysteresis of rolling
scalar e_qe = sqrt(1-0.15);
scalar R_star = p.d()/2; //m, effective radius of contact, i.e. particle radius when colliding with wall, see Eidevåg 2020

scalar K_1 = 0.9355; //integration constant, see Eidevåg 2020
scalar m_p = p.mass()*p.nParticle(); //kg, mass of particle
scalar a_0 = cbrt(9*mathematical::pi*W*pow(R_star,2)/(2*E_star));
scalar E_s = 3*K_1*mathematical::pi*pow(a_0,2)*W/(4*cbrt(6.0));
scalar V_s = sqrt(2*E_s/m_p);

scalar V_cn = V_s; //Critical velocity in normal direction
scalar V_ct = 0.23*Dgamma_gamma*V_cn; //Critical velocity in tangential direction

vector nw; //wall normal vector
vector Up; //particle velocity
this->owner().patchData(p, pp, nw, Up);
vector Ur = U - Up; //particle velocity relative to wall
vector Un = (Ur & nw)*nw; //velocity component along normal direction
vector Ut = Ur - Un; //velocity component along tangential direction
scalar V_in = mag(Un);
scalar V_it = mag(Ut);

//wall-normal and tangential coefficients of restitution
scalar e_n = e_qe * sqrt(1 - pow((V_cn/(max(V_in, V_cn))),2));
scalar e_t = e_qe * sqrt(1 - pow((V_ct/(max(V_it, V_ct))),2));
```

Using function objects to generate fields for calculation

Need to define function objects for calculating wall shear stress and averages.

controlDict

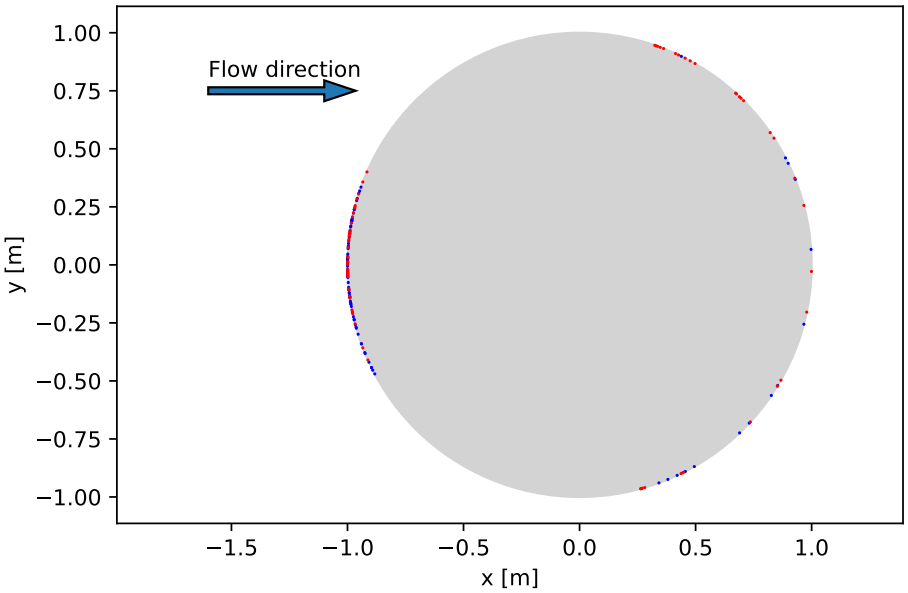
```
wallShearStress1
{
    // Mandatory entries (unmodifiable)
    type            wallShearStress;
    libs            (fieldFunctionObjects);

    // Optional entries (runtime modifiable)
    //patches         (cylinder); // (wall1 "(wall2|wall3)");

    // Optional (inherited) entries
    executeControl  timeStep;
    executeInterval 1;
    writeControl    writeTime;
}
```

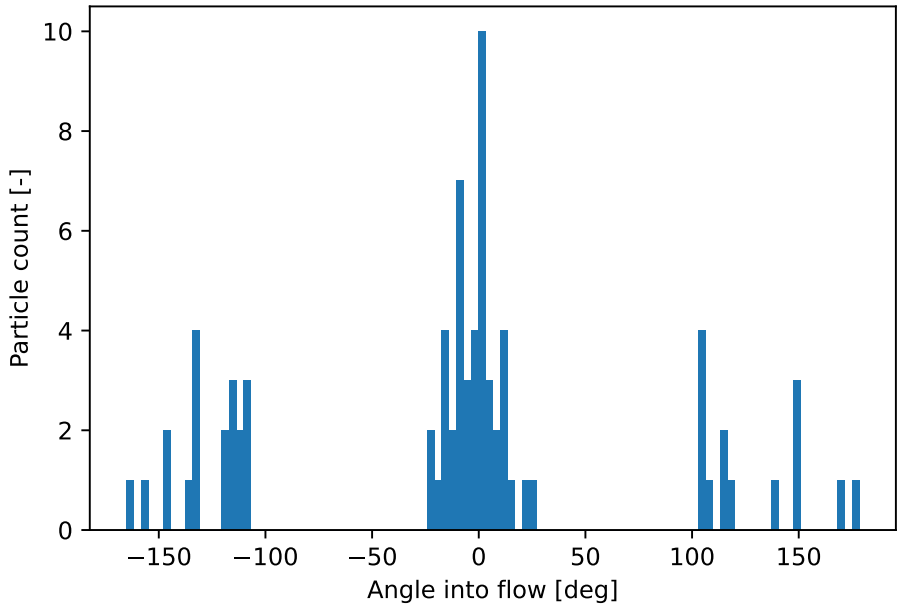
Using function objects to generate fields for calculation

controlDict

```
fieldAverage1
{
    // Mandatory entries (unmodifiable)
    type            fieldAverage;
    libs            (fieldFunctionObjects);

    // Mandatory entries (runtime modifiable)
    fields
    (
    wallShearStress
    {
        mean        yes;
        prime2Mean  yes;
        base        time;
        window          10000;
        windowName      w1;
        windowType      none;
    }
    );
    executeControl  timeStep;
    executeInterval 1;
    writeControl    writeTime;
}
```

Testing the new collision model





Red: deposited
Blue: free-floating
New model gave $\approx 60\,\%$ reduction in particle deposition rates compared to `stick`.

Use cases for the hybrid collision model

- Car industry
- Icing of aircraft
- Ash build-up in boilers

Conclusions

- This work presents a methodology for creating new particle-wall collision models in OpenFOAM.
- One collision model from theory was implemented and shown to behave differently from the already implemented models.

Outlook and future work

- Validate the model against other implementations
- Validate the model against experimental results
- Implement and test other models

References

[1] Eidevåg, T.; Eng, M.; Kallin, D.; Casselgren, J.; Bharadhwaj, Y.; Bangalore Narahari, T. S.; Rasmuson, A. Snow Contamination of Simplified Automotive Bluff Bodies: A Comparison Between Wind Tunnel Experiments and Numerical Modeling; 2022; pp 2022-01–0901. https://doi.org/10.4271/2022-01-0901.

[2] Srinivasachar, S.; Helble, J. J.; Boni, A. A. An Experimental Study of the Inertial Deposition of Ash under Coal Combustion Conditions. Symposium (International) on Combustion 1991, 23 (1), 1305–1312. https://doi.org/10.1016/S0082-0784(06)80394-2.