Lifting-a-sessile-drop

This repository contains the codes used for simulating the cases discussed in the manuscript: Lifting a sessile oil drop from a superamphiphobic surface with an impacting one (link).

You can use the following article citation:

```
@article{ramirez2020lifting,
  title={Lifting a sessile oil drop from a superamphiphobic surface with an impacting one},
  author={Ram{\'\i}rez-Soto, O. and Sanjay, V. and Lohse, D. and Pham, J. T. and Vollmer, D.},
  journal={Science advances},
  volume={6},
  number={34},
  pages={eaba4330},
  year={2020},
  publisher={American Association for the Advancement of Science}
}
```

Note Use the raw source file available as dropOnDropImpact.c, present in all the folders in this repository.

Introduction

We investigate the dynamics of an oil drop impacting an identical sessile drop sitting on a superamphiphobic surface. On this page, I am presenting the code that we used to simulate the process shown in the above video. For a more detailed documentation on the code included in the manuscript, please visit my Basilisk sandbox.

Some test cases

Here is a typical experiment:

- Experimental Video
- Simulation Video: The process
- Simulation Video: Velocity Field

Numerical code

Id 1 is for the sessile drop, and Id 2 is mobile/impacting drop.

```
1 #include "grid/octree.h"
2 #include "navier-stokes/centered.h"
3 #define FILTERED // Smear density and viscosity jumps
```

Modelling non-coalescence of drops

To model non-coalescing drops, we use two different Volume of Fluid tracers (f1 and f2). For this, we use a modified version of two-phase.h. A proof-of-concept

example is here.

```
#include "two-phaseDOD.h"
#include "tension.h"
#include "distance.h"
```

We use a modified adapt-wavelet algorithm available (here). It is written by *César Pairetti* (Thanks :)).

```
#include "adapt_wavelet_limited.h"
```

Some global parameters

```
// maximum level
int MAXlevel = 12;
// minimum level
#define MINlevel 5
// timestep used to save simulation snapshot
#define tsnap (0.01)
// Error tolerances
// VOF
#define fErr (5e-4)
// curvature of sessile drop
#define K1Err (1e-4)
// curvature of impacting drop
#define K2Err (1e-4)
// velocity
#define VelErr (5e-3)

// viscosity and density ratios between the gas and the liquid
#define Mu21 (6e-3)
#define Rho21 (1./770)
// height of impacting drop from the substrate
#define Zdist (3.)
```

Offset parameter

In the manuscript, offset parameter χ is defined as:

$$\chi = \frac{d}{2R}$$

Here, d is the distance between the axes of two drops, and R is the equivalent radius of the drop. In the simulations, we input 2χ (because the length scale is R).

```
#define Xoffset (0.50)
// Equation for the impacting drop
#define R2Drop(x,y,z) (sq(x-Xoffset) + sq(y) + sq(z-Zdist))
// Dimension of the domain
#define Ldomain 16
```

Boundary Conditions

Back Wall is superamphiphobic and has the no-slip condition for velocity.

```
u.t[back] = dirichlet(0.);
u.r[back] = dirichlet(0.);
f1[back] = dirichlet(0.);
f2[back] = dirichlet(0.);
double tmax, We, Oh, Bo;
```

Weber number is based on the impact velocity, U_0.

$$We = \frac{\rho_l U_0^2 R}{\gamma}$$

Note that the impact Weber number we input is slightly less than the actual Weber number that we need. As the impacting drop falls, it also gains some kinetic energy. So, while analyzing the results, the Weber number should be taken at the instant, which one decides to choose as $t = 0$. These calculations get tricky as one goes to a higher χ. However, the results do not change much as long as $We \sim \mathcal{O}(1)$.

Navier Stokes equation for this case:

$$\partial_t U_i + \nabla \cdot (U_i U_j) = \frac{1}{\hat{\rho}} \left(-\nabla p + Oh \nabla \cdot (2\hat{\mu} D_{ij}) + \kappa \delta_s n_i\right) + Bo g_i$$

The $\hat{\rho}$ and $\hat{\mu}$ are the VoF equations to calculate properties, given by:

$$\hat{A} = (f_1 + f_2) + (1 - f_1 - f_2)\frac{A_g}{A_l}$$

Ohnesorge number Oh is a measure of viscous forces as compared to inertia and surface tension forces.

$$Oh = \frac{\mu_l}{\sqrt{\rho_l \gamma R}}$$

Bond number Bo is a measure between Gravity and surface tension.

$$Bo = \frac{\rho_l g R^2}{\gamma}$$

Note: The subscript l denotes liquid. Also, the radius used in the dimensionless numbers is the equivalent radius of the drops $\left(R = (3\pi V_l/4)^{1/3}\right)$. V_l is the volume of the two drops.

Velocity scale as the intertial-capillary velocity,

$$U_\gamma = \sqrt{\frac{\gamma}{\rho_l R}}$$

```
int main() {
  tmax = 7.5;
  We = 1.375; // We is 1 for 0.1801875 m/s
  // <770*0.1801875^2*0.001/0.025>

  init_grid (1 << MINlevel);
  L0=Ldomain;
  Oh = 0.0216; // <0.003/sqrt(770*0.025*0.001)>
  Bo = 0.308; // <770*10*0.001^2/0.025>
  fprintf(ferr, "tmax = %g. We = %g\n",tmax, We);
  rho1 = 1.0; mu1 = Oh;
  rho2 = Rho21; mu2 = Mu21*Oh;
  f1.sigma = 1.0; f2.sigma = 1.0;
  run();
}
/**
This event is specific to César's adapt_wavelet_limited.
Near the substrate, we refine the grid one level higher
than the rest of the domain.
*/
int refRegion(double x, double y, double z){
    return (z < 0.128 ? MAXlevel+1 : MAXlevel);
}
```

Initial Condition

For Bond numbers > 0.1, assuming the sessile drop to be spherical is inaccurate. One can also see this in the experimental video above. So, we used the code (here) to get the initial shape of the sessile drop. We import an STL file.

```
event init(t = 0){
  if(!restore (file = "dump")){
    char filename[60];
    sprintf(filename,"Sessile-Bo0.3080.stl");
    FILE * fp = fopen (filename, "r");
    if (fp == NULL){
      fprintf(ferr, "There is no file named %s\n", filename);
```

```
      return 1;
    }
    coord * p = input_stl (fp);
    fclose (fp);
    coord min, max;
    bounding_box(p, &min, &max);
    fprintf(ferr, "xmin %g xmax %g\nymin %g ymax %g\nzmin %g
↪   zmax %g\n", min.x, max.x, min.y, max.y, min.z, max.z);

    origin((min.x+max.x)/2. - L0/2, (min.y+max.y)/2. - L0/2,
↪   0.); // We choose (X,Y) of origin as the center of the
↪   sessile drop. And substrate at Z = 0.

    refine(R2Drop(x,y,z) < sq(1.+1./16) && level < MAXlevel);
    fraction(f2, 1. - R2Drop(x,y,z));
    scalar d[];
    distance (d, p);
    while (adapt_wavelet_limited ((scalar *){f2, d},
    ↪   (double[]){1e-6, 1e-6*L0}, refRegion).nf);
    vertex scalar phi[];
    foreach_vertex(){
      phi[] = (d[] + d[-1] + d[0,-1] + d[-1,-1] +
         d[0,0,-1] + d[-1,0,-1] + d[0,-1,-1] + d[-1,-1,-1])/8.;
    }
    boundary ((scalar *){phi});
    fractions (phi, f1);
    foreach () {
      u.z[] = -sqrt(We)*f2[];
      u.y[] = 0.0;
      u.x[] = 0.0;
    }
    boundary((scalar *){f1, f2, u.x, u.y});
    dump (file = "dump");
    /**
    **Note:** I think
↪   [distance.h](http://basilisk.fr/src/distance.h) is not
↪   compatible with mpi. So, I ran the file to import .stl file
↪   and generate the dump file at t = 0 locally. For this,
↪   OpenMP multi-threading can be used.
    */
    return 1;
  }
}
```

Gravity

Gravity is added as a body forces. It would be nice to use something like reduced.h. But, I could not figure out how to do it with two different VoF tracers.

```
event acceleration(i++) {
  face vector av = a;
  foreach_face(z){
    av.z[] -= Bo;
  }
}
```

Adaptive Mesh Refinement

```
event adapt(i++) {
  /**
  We refine based on curvatures of the two drops along with the generally used VoF and velocity fields. This ensures that the refinement level along the interface is MAXlevel.
  */
  scalar KAPPA1[], KAPPA2[];
  curvature(f1, KAPPA1);
  curvature(f2, KAPPA2);
  adapt_wavelet_limited ((scalar *){f1, f2, KAPPA1, KAPPA2, u.x, u.y, u.z},
    (double[]){fErr, fErr, K1Err, K2Err, VelErr, VelErr, VelErr},
      refRegion, MINlevel);
}
```

Dumping snapshots

```
event writingFiles (t = 0; t += tsnap; t <= tmax) {
  dump (file = "dump");
  char nameOut[80];
  sprintf (nameOut, "intermediate/snapshot-%5.4f", t);
  dump (file = nameOut);
}
```

Log writing

```
event logWriting (i++) {
  double ke = 0.;
```

```
  foreach (reduction(+:ke)){
    ke += 0.5*(sq(u.x[]) + sq(u.y[]) +
↪   sq(u.z[]))*rho(f1[]+f2[])*cube(Delta);
  }
  static FILE * fp;
  if (i == 0) {
    fprintf (ferr, "i dt t ke\n");
    fp = fopen ("log", "w");
    fprintf (fp, "i dt t ke\n");
    fprintf (fp, "%d %g %g %g\n", i, dt, t, ke);
    fclose(fp);
  } else {
    fp = fopen ("log", "a");
    fprintf (fp, "%d %g %g %g\n", i, dt, t, ke);
    fclose(fp);
  }
  fprintf (ferr, "%d %g %g %g\n", i, dt, t, ke);
}
```

Running the code

Use the following procedure:

Step 1: Importing the stl file and generating the first dump file

```
#!/bin/bash
mkdir intermediate
qcc -fopenmp -O2 -Wall dropOnDropImpact.c -o dropOnDropImpact
↪   -lm
export OMP_NUM_THREADS=8
./dropOnDropImpact
```

Step 2: Follow the method described (here). Do not forget to use the dump file generated in the previous step.

Output and Results

The post-processing codes and simulation data are available at: PostProcess