#include "[navier-stokes/centered.h](http://basilisk.fr/src/navier-stokes/centered.h)"

1. Values at each grid point are computed by considering the values at neighbouring grid points symmetrically around the point of interest.
2. Derivative at a point can be approximated by shape between the values at adjacent grid points.
3. It includes:
   1. Time-integration
   2. Linearization – iterative approach like central-difference method
   3. Boundary conditions
   4. Adaptivity
   5. Projection
4. Governing equations:
   1. Continuity Equation
   2. Navier Stokes Equation

#include "[two-phase.h](http://basilisk.fr/src/two-phase.h)"

1. Related to simulating two-phase flows. It refers to flow of two immiscible fluids (like gas and liquid) within a computational domain.
2. It includes:
   1. Interface tracking - accurate modelling of fluid interface and capturing its behaviour.
   2. Surface tension
   3. Phase properties – density, viscosity, interfacial tension

#include "tension.h"

1. It takes into account the surface tension or the function and macros related to interfacial tension modelling.
2. In multiphase flows, surface tension plays role.
3. It includes:
   1. Surface tension forces – these forces arise due to curvature of interface and interface tension between two phases
   2. Interfacial curvature – it determines the behaviour of fluid interfaces and influences surface tension forces
   3. Surface tension models – like Cahn-Hilliard or Allen Cahn equations. It captures the dynamics of fluid interfaces with surface tension effects
   4. Timestep stability while using Surface tension
4. Interfacial force density = Φ∇f
   * 1. f- volume fraction describing interface
     2. Φ = σκ (potential)
     3. σ = surface tension
     4. κ = interface mean curvature

#include "[adapt\_wavelet.h](http://basilisk.fr/sandbox/pairetti/bag_mode/adapt_wavelet_limited.h)"

1. Related to Adaptive Mesh Refinement (AMR). It provides higher resolution in regions of interest and reducing computational cost in less important region.
2. AMR based on conditions like gradients, error estimates.
3. It includes:
   1. Defining Refinement criteria - criteria for determining where the mesh should be refined based on specific conditions, such as gradients, error estimates.
   2. Adaptive Mesh Refinement - actual refinement of the mesh based on the criteria
   3. Solution projection – projects the solution onto adaptive mesh ensuring consistency and accuracy.

#include "[adapt\_wavelet\_limited.h](http://basilisk.fr/sandbox/pairetti/bag_mode/adapt_wavelet_limited.h)"

Modified version of adapt\_wavelet where it supports the maximum level of refinement limited by certain regions.

It contains an input structure which has a function MLFun that defines maxlevel for a domain.

## A correction I made in the s.coarsen part, because no\_coarsen is not defined anywhere in Basilisk.

bool limitedAdaptation = 0;

Here the limitedAdaptation is given a ‘false’ value.

## however later in the code it is set to ‘true’. Thus, we get the knowledge that there are two cases-

1. Using the adapt\_wavelet\_limited header file
2. Not using it.

double xmin, xmax, ymin, ymax;

It sets the maximum and minimum x and y coordinates of the bounding box for mesh refinement. It is set as global variable.

void approx\_bbox (scalar c) {

A function that defines the bounding box region with respect to the scaler field c. Bounding box is the region where mixed cells are present.

[assert](http://man7.org/linux/man-pages/man3/assert.3.html) (dimension == 2);

It ensures that we are working in 2D.

xmin = L0;

xmax = -L0;

ymin = L0;

ymax = -L0;

[foreach](http://basilisk.fr/Basilisk%20C#iterators)()

**if** (c[] > 1e-6 && c[] < 1. - 1e-6) {

**if** (xmin > x) xmin = x;

**if** ymin > y) ymin = y;

**if** (xmax < x) xmax = x;

**if** (ymax < y) ymax = y;

}

}

Iterates over each cell and updates the bbox coordinates based on the given range of c.

x and y are the coordinates of the bottom-left corner of each cell.

## However, it is not being used anywhere else in the program.

#define RHOL 785.

#define MUL 16.1e-3

#define SIGMA 48.3e-3

#define RHOG 1.

#define MUG 1.81e-5

#define DIAMETER 2.7e-3

#define USTREAM 15.

#define MAXTIME 30e-3

density and viscosity of air and liquid are given along with the value of surface tension. Drop diameter and stream velocity is also given. Maximum simulation time is given as 30ms. Thus, we cannot go beyond this time.

## MAXTIME is defined but is not being used anywhere further. It could be used in event\_end

int minlevel = 6;

int MAXLEVEL = 10;

double maxruntime = HUGE;

Minimum and maximum level of refinement is given.

## HUGE is not defined anywhere in the program. To solve this, we can include <float.h> header file. Also it is being defined again here.

u.n[left] = dirichlet (USTREAM);

u.t[left] = dirichlet (0);

u.n[right] = neumann (0);

p[left] = neumann (0);

p[right] = dirichlet (0)

Standard way of stating the boundary conditions in the inlet and outlet. Dirichlet assigns the specific value given and neumann assigns the gradient of the field. Also, we assume that the flow becomes fully developed at the outlet.

int main (int argc, char \* argv []) {

**if** (argc > 1)

MAXLEVEL = [atoi](http://man7.org/linux/man-pages/man3/atoi.3.html) (argv [1]);

The main function will run based on the command line argument given. If at least one command line is given it will set maxlevel of refinement to that level. Otherwise, it will be 10 as specified earlier. It is useful for setting the maxlevel only at execution time rather than modifying the code.

For passing maxlevel we need to use ‘. / droplet 8’ in the terminal.

L0 = 5\*DIAMETER;

size (L0);

origin (-L0/4, -L0/2);

init\_grid (pow (2.0, minlevel));

L0 is just a variable given a value based on diameter of droplet. It is used in size () function to determine the dimensions of the simulation domain. Origin is also set according to L0. Finally, we initialise a grid with resolution set to 2^6 (this many grid cells are generated).

double minDelta = L0/pow (2.0, MAXLEVEL); // length of each cell

double sigmaTSLimit = sqrt (min (RHOL, RHOG) \* pow(minDelta,3.0)/(SIGMA));

double muTSLimit = min (MUL, MUG) \* minDelta/SIGMA;

DT = min (sigmaTSLimit, muTSLimit);

CFL = 0.4;

Next, we limit the timesteps based on the value of surface tension and viscosity to ensure stability in fluid flow and know the proper behaviour of fluid system. Large timestep cannot capture the accurate effects of both. Thus, we set it to the min. of the sigmaTSLimit (surface tension) and muTSLimit (viscosity).

Both CFL and DT is present in navier-stokes/centered header file.

minDelta represents the size of each cell.

## I don’t know why CFL number is again defined here. Its significance becomes the same thing as DT. Maybe it’s just for reference. Only DT is used.

f. sigma = SIGMA;

rho1 = RHOL;

rho2 = RHOG;

mu1 = MUL;

mu2 = MUG;

f [] is a scaler field defined in two-phase.h and f. sigma is defined in tension.h header file. f defines the volume fraction field of liquid in gas. It will determine the interfacial forces and behaviour of liquid and gas phase.

TOLERANCE = 1e-4;

It is the error upto which iterations to be done while solving poisons equation. It is a must as it solves for the pressure in Navier-Stokes. The values are stored in the variables ‘mgp’ and ‘mgpf’ (face centered) included in the navier-stokes/centered.h

[run](http://basilisk.fr/src/run.h#run)();

limitedAdaptation = 1;

[run](http://basilisk.fr/src/run.h#run)();

}

## Here two run() functions are used. One before limitedAdaption = true and one after. Is it possible and if so when is they first run() ending?

[event](http://basilisk.fr/Basilisk%20C#events) init (t = 0) {

**if** (![restore](http://basilisk.fr/src/output.h#restore) (file = "dump")) {

refine (sq(x) + sq(y) – sq (0.6\*DIAMETER) < 0 && sq(x) + sq(y) – sq (0.4\*DIAMETER) > 0 && level < MAXLEVEL);

fraction (f, sq (0.5\*DIAMETER) - (sq(x) + sq(y)));

}

}

The program first checks if there is a ‘dump’ file with previous-states defined present or not.

If true (value = 1) then it will keep the same values of f, u, p and skip the whole init event.

If false (value =0) then it will proceed to the if statement. Here a ring is taken between two circles of radius 0.6\*DIAMETER and 0.4\*DIAMETER, and the mesh is refined only in this region.

Volume fraction of liquid (f) is set in the droplet defined in the circle of radius 0.5\*DIAMETER

Refine () is present in adapt\_wavelet\_limited and fraction () is present in navier-stokes/centered header.

## I can include approx.\_bbox (f) just after fraction ()

[event](http://basilisk.fr/Basilisk%20C#events) end (i = 10) {

[printf](http://man7.org/linux/man-pages/man3/printf.3.html) ("i = %d t = %g\n", i, t);

}

Triggered only at i = 10. Prints the simulation time at 10th iteration.

int refRegion (double x, double y, double z) {

int lev;

**if**(sq(x)+sq(y) < sq (DIAMETER\*0.7))

lev = MAXLEVEL;

else

lev = MAXLEVEL-4;

**return** lev;

}

RefRegion returns the level of refinement taking arguments x, y and z coordinates of the cell. The level of refinement is done only in a region slightly larger than droplet thus ensuring that whole droplet is covered and any effect near its vicinity is also covered. In other regions the refinement is scaled down to MAXLEVEL-4.

## We have previously asserted that dimension is 2. So, I don’t think there is need of z argument.

[event](http://basilisk.fr/Basilisk%20C#events) adapt (i++) {

double uemax = 5e-2;

**if**(limitedAdaptation)

[adapt\_wavelet\_limited](http://basilisk.fr/sandbox/pairetti/bag_mode/adapt_wavelet_limited.h#adapt_wavelet_limited) ({f, u}, (double []) {1e-3, uemax, uemax, uemax}, [refRegion](http://basilisk.fr/sandbox/pairetti/bag_mode/bag_mode.c#refRegion), minlevel);

else

[adapt\_wavelet](http://basilisk.fr/src/grid/tree-common.h#adapt_wavelet) ({f, u}, (double []) {1e-3, uemax, uemax, uemax}, MAXLEVEL, minlevel);

}

Maximum error in velocity field is defined. In this event the main AMR is done. If limitedAdaptation is true then the user-modified adapt\_wavelet function will work. It takes argument a struct with keys:

1. List of scalers
2. Tolerance for each scaler
3. User defined function that returns (int) maxlevel of refinement
4. Minlevel of refinement (default is 1)
5. List of fields to update (default is all)

Only values of values of f and u are to be taken while refining. While considering u, we will take separately u.x, u.y and u.z

If limitedAdaptation is false then standard adapt\_wavelet will work. Same arguments except the local function.

## We are working in a 2D. So, we should not take u.z into consideration. Also, I can’t understand the value of limitedAdaptation. First, we assign it false. So, the standard adapt\_wavelet will work. Then when will it change to true? When is the first run () completed?

[event](http://basilisk.fr/Basilisk%20C#events) logfile (i ++, first) {

**if** (i == 0) {

[fprintf](http://man7.org/linux/man-pages/man3/fprintf.3.html) (ferr,

"t dt mgp.i mgpf.i mgu.i perf.t perf.speed grid->tn\n");

}

[fprintf](http://man7.org/linux/man-pages/man3/fprintf.3.html) (ferr,

"%g %g %d %d %d"

"%.2e %.2e %ld \n",

t, dt, mgp.i, mgpf.i, mgu.i,

perf.t, perf.speed, grid->tn);

[fflush](http://man7.org/linux/man-pages/man3/fflush.3.html) (ferr);

}

‘First’ as an argument will make sure that logile is updated first before any other event. It will make a file with the values of

t = simulation time

dt = timestep (already defined earlier)

mgp.i = multigrid iterations for pressure solver

mgpf.i = multigrid iterations for phase fraction solver

mgu.i = multigrid iterations for velocity solver

perf.t = CPU time taken

perf.speed = grid point per sec

grid->tn = total number of cells in grid

All of these are defined in navier-stokes/centered header file

[event](http://basilisk.fr/Basilisk%20C#events) snapshot (i=0; i++; i <= 10)

{

scalar omegaz [];

[foreach](http://basilisk.fr/Basilisk%20C#iterators)()

omegaz [] = (u.y[1] - u.y[-1] - u.x[0,1] + u.x[0,-1])/ (2. \*Delta);

[boundary](http://basilisk.fr/Basilisk%20C#boundary-conditions) ({omegaz});

char name [80];

**if**(limitedAdaptation)

[sprintf](http://man7.org/linux/man-pages/man3/sprintf.3.html) (name, "snapshot-limited-%i\_ts.gfs", i);

else

[sprintf](http://man7.org/linux/man-pages/man3/sprintf.3.html) (name, "snapshot-standard-%i\_ts.gfs", i);

[output\_gfs](http://basilisk.fr/src/output.h#output_gfs) (file = name, t = t, list = {f, u, p, omegaz});

}

Vorticity ω = ∇×V

In 2D it reduces to ωz = ω · ˆk = ∂ v / ∂ x − ∂u / ∂ y, where v = u.y and u = u.x

Central difference approximation method is used for calculating vorticity component. Boundary condition on vorticity is imposed to ensure its effect on the boundary edges.

As stated earlier, two kinds of adapt wavelet function are being used here. Hence two kinds of gfs will be generated based on standard function and limited function. The gfs file will store 4 kinds of field for evaluation (f, u, p and omegaz).

## I cannot open GFS file even after using gfsviewer in my WSL. So, I changed the output to ppm files and changed the name likewise. Also, instead of manually defining omegaz I could have used vorticity () built-in function. For that embed.h is required.

References:

[1] A. Salih, Streamfunction-Vorticity Formulation

[2] Cédric Galusinski & Paul Vigneaux, On stability condition for bifluid flows with surface tension: Application to microfluidics, Volume 227, Issue 12, Journal of Computational Physics

[3] L. Opfer, I. V. Roisman, J. Venzmer, M. Klostermann, and C. Tropea, Droplet-air collision dynamics: Evolution of the film thickness, 10.1103, PhysRevE.89.013023

[4] Kurian J. Vachaparambil and Kristian Etienne Einarsrud, Comparison of Surface Tension Models for the Volume of Fluid Method,

[5] [Basilisk - Basilisk](http://basilisk.fr/Front%20Page)