Notes Basilisk CFD

Wenge Huang

February 3, 2023

1 Codes

1.1 heights.h

The "height-function" is a vector field which gives the distance, along each coordinate axis, from the center of the cell to the closest interface defined by a volume fraction field.

That means if the vector is (x, y, z), ~~the results of the high function will be $(height(h.x[]), h.y, h.z)$~~. In this case h itself is a vector and we have the three component of h: $(h.x, h.y, h.z)$

The distance is normalised with the cell size so that the coordinates of the interface are given by

(x, y + Delta*height(h.y[])) or (x + Delta*height(h.x[]), y) Here heigh is a function that takes a scalar h.y[] or h.x[] as an input.

Here Delta is initially defined as the size of the cell

```
#define HSHIFT 20.

static inline double height (double H) {
  return H > HSHIFT/2. ? H - HSHIFT : H < -HSHIFT/2. ? H + HSHIFT : H
    ;
}

static inline int orientation (double H) {
  return fabs(H) > HSHIFT/2.;
}
```

There are some other codes to calculate the distance and finally we can get the H which is distance. The we'll use the *height* function in other place to call and get a good H (which is a scalar).

1.2 curvature.h

To compute the curvature, we estimate the derivatives of the height functions in a given direction (x, y or z). We first check that all the heights are defined and that their orientations are the same. We then compute the curvature as:

$$\kappa = \frac{h_{xx}}{(1+h_x^2)^{3/2}}$$

$$\kappa = \frac{h_{xx}(1+h_y^2) + h_{yy}(1+h_x^2) - 2h_{xy}h_xh_y}{(1+h_x^2+h_y^2)^{3/2}}$$

```
#include "heights.h"

#if dimension == 2
foreach_dimension()
static double kappa_y (Point point, vector h)
{
  int ori = orientation(h.y[]);
  for (int i = -1; i <= 1; i++)
    if (h.y[i] == nodata || orientation(h.y[i]) != ori)
      return nodata;
  double hx = (h.y[1] - h.y[-1])/2.;
  double hxx = (h.y[1] + h.y[-1] - 2.*h.y[])/Delta;
  return hxx/pow(1. + sq(hx), 3/2.);
}

foreach_dimension()
static coord normal_y (Point point, vector h)
{
  coord n = {nodata, nodata, nodata};
  if (h.y[] == nodata)
    return n;
  int ori = orientation(h.y[]);
  if (h.y[-1] != nodata && orientation(h.y[-1]) == ori) {
    if (h.y[1] != nodata && orientation(h.y[1]) == ori)
      n.x = (h.y[-1] - h.y[1])/2.;
    else
      n.x = h.y[-1] - h.y[];
  }
  else if (h.y[1] != nodata && orientation(h.y[1]) == ori)
    n.x = h.y[] - h.y[1];
  else
    return n;
  double nn = (ori ? -1. : 1.)*sqrt(1. + sq(n.x));
  n.x /= nn;
  n.y = 1./nn;
  return n;
}
```

We first check that all the heights are defined and that their orientations are the same. we use the command foreach_dimension(). with head file 'heights.h' we somehow calculate the height function h. That means we have h.x, h.y and h.z and also the corresponding orientations.

We now need to choose one of the x, y or z height functions to compute the curvature. This is done by the function below which returns the HF curvature given a volume fraction field *c* and a height function field *h*.

1.3 fractions.h

We need to use the fraction function to maintain or define volume and suface fractions either from an initial geometric definition or from an existing volume fraction field.

```
fraction (f, sq(radius) - sq(y) - sq(x));
```

Now we're going to discuss how this function works.

The convenience **macros** below can be used to define volume and surface fraction fields directly from a function.

```
#define fraction(f,func) do {      \
    vertex scalar phi[];       \
    foreach_vertex()         \
      phi[] = func;        \
    fractions (phi, f);        \
  } while(0)

#define solid(cs,fs,func) do {       \
    vertex scalar phi[];       \
    foreach_vertex()         \
      phi[] = func;        \
    fractions (phi, cs, fs);       \
  } while(0)
```

In this fraction function, it called the vertex variable phi[] and also another function fractions.

```
struct Fractions {
  vertex scalar Phi; // compulsory
  scalar c;          // compulsory
  face vector s;     // optional
  double val;        // optional (default zero)
};

trace
void fractions (struct Fractions a)
{
  vertex scalar Phi = a.Phi;
```

```
  scalar c = a.c;
  face vector s = automatic (a.s);
  double val = a.val;
```

First, it defines a structure Fraction with members Phi (capital P) and c and others. Then the structure is renamed as a. So we have a.Phi, a.c and so on.

1.4 Example Height functions

We need to use fraction function to get the volume fraction of the computation domain:

```
scalar c[];
fraction (c, - (0.2 - sqrt(sq(x+0.2) + sq(y+0.2))));
```

With the volume fraction c known, we can use the height function to calculate the height:

```
 vector h[];
 heights (c, h);
```

The heights() function takes a volume fraction field c and returns the height function vector field h.

Also, we can use the height function to calculate the calvature:

```
  scalar kappa[];
  curvature (c, kappa);
```