height.h 说明文档

Haochen Huang
西安交通大学 MFM 课题组

版本：1.02
日期：2023 年 7 月 3 日

摘　　要

本文档为 height.h 说明文档的测试初稿版本，需要继续的工作包括对树状网格函数部署的相关代码解析，整体算法的高度总结，对代码中各种数值加密的详细解析等

1.02 更新，根据反馈更改了大量注释，对图例进行了阐释

目录

1 height.h 整体头文件目的 1
2 heights 函数整体循环框架与具体实现 1
2.1 height 整体循环框架 1
2.2 heights 及 half column 源代码及注释 4
3 column propagation 函数 12
3.1 函数目的及原理 12
3.2 column propagation 函数源代码及注释 13
参考文献 13

1. height.h 整体头文件目的

本头文件是构造多相流边界几何的必要工具头文件之一，其与 parabola.h 共同为 curvature.h 服务，计算实现边界面的曲率。相关算法的具体细节请参考 [1]

2. heights 函数整体循环框架与具体实现

2.1 height 整体循环框架

由于 height.h 文件框架相当特殊，两个主要函数相互耦合才能完全展示整个循环，是故不分别撰写 half column 与 heights 函数。

heights 函数的目的是遍历该单元所有方向（二维 2 个方向，三维 3 个方向）上的正负 4 个单元，并求出该单元的高度函数值，存储在 vector 型数据中。两函数在此过程中的作用分别是：

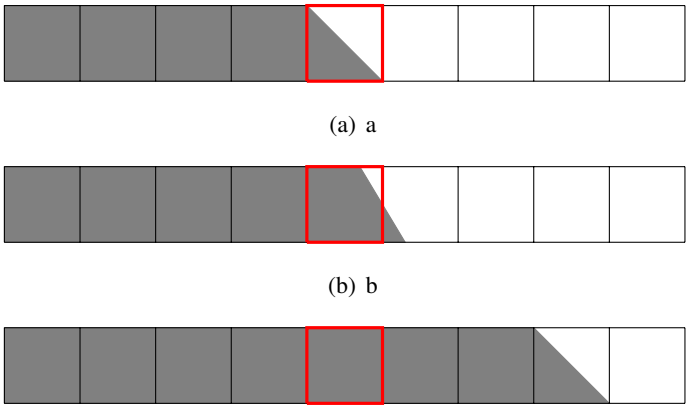
1. half column 函数通过传递的参数，求解所有维度上单一方向的 height 函数值，并在 heights 函数对方向进行循环时更新完整的 height 函数值并返回。

2. heights 函数通过调用 half column 并通过对参数 j 的循环达到正负方向的遍历。

由于各个方向的高度相似性，下文中我们均以二维问题中 x 方向的情况作为例子。

接下来我们对高度函数值进行解析定义。

一个单元的高度函数值得重要条件是空间上的循环**完整**地通过一个界面，即在界面前后都找到 $c = 0, c = 1$ 的单元格，如下图情况所示（本文中图例仅作参考，实际运用过程中会有所不同，请读者按照代码仔细甄别，在 curvature.h 中会具体分析）



(a) a

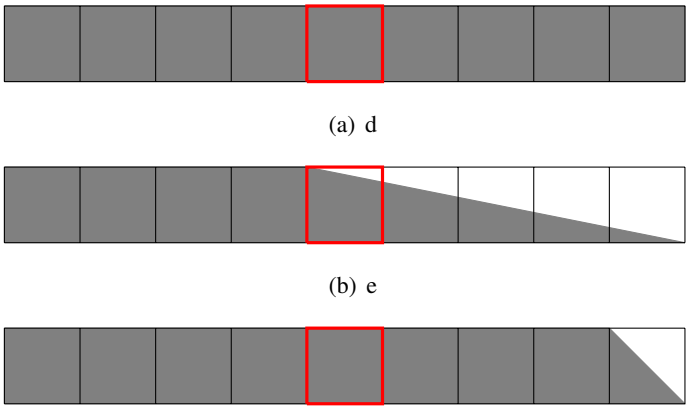
(b) b

(c) c

图 1: 高度函数值可定义情况示例

其中灰色部分代表界面内部，白色代表界面外侧，红色方框代表当前单元。在这种情况下高度函数有定义。

如不满足上述条件的网格情况，高度函数在该单元上没有定义，算法会对单元赋值 $nodata$（一个极大的数值，在 common.h 中定义），如下图2



(a) d

(b) e

(c) f

图 2: 高度函数不可定义情况示例

高度函数的本质是单元界面内体积分数的加和，各个单元表示量以单元中心线为基准，零标准面处于**界面开始与界面终结阶段的数值点**，如图：

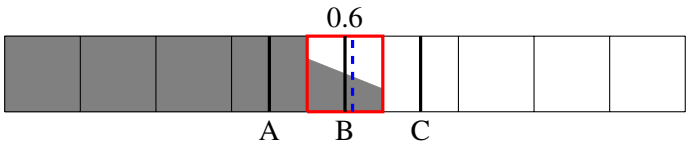


图 3: 高度函数值示例图

图中界面部分体积分数为 0.6，ABC 三个单元即为 “界面的开始与界面的终结” 故其零标准面在距离 A 左端 1.6 长度处（$1+0.6$）为图中蓝色点划线，而各个单元中存储的数据为位于单元中心的黑色直线代表的函数高度，计算方法为从零界面开始，坐标正方向递减，负方向增加。ABC 所储存的函数值分别为 $1.1, 0.1, -0.9$。

规定由界面内部指向界面外为一个界面的方向（由 $1 \geq c > 0$ 指向 $c = 0$），得到的相关结果如图4

图中 E 处为界面位置，可以看到该函数利用数值非常巧妙的表达了界面方向，即当坐标正方向与界面方向一致时，高度函数值为 $(-5, 5)$，当相互反向时，高度函数值为 $(15, 25)$。

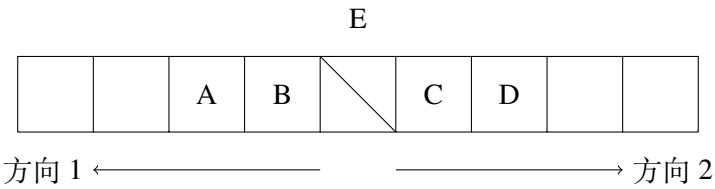


图 4: 具体数值结果示例

方向 \ 位置序号	A	B	E	C	D
方向 1	22	21	20	19	18
方向 2	2	1	0	-1	-2

表 1: 不同界面方向下各个位置的高度函数值，为了简化流程，界面为平分单元界面

接下来介绍在代码实现中的算法。

Algorithm 1: Height 主题循环框架

Input: 全场各个单元的体积分数 c，总维度 DIMENSION

Output: 全场各个单元的高度函数值 h，vector 型数据

for $dimension = 1; dimension <= DIMENSION; dimension++$ **do**
  设定当前维度的高度函数值 h.dimension[];
  for $j = -1; j <= 1; j+ = 2$ **do**
    for $i = 0; i <= 4; i++$ **do**
      $h.dimension[]+ = c[i \times j]$;
      if $(c[] == 0 \&\& c[i \times j] == 1) || (c[] == 1 \&\& c[i \times j] == 1)$ **then**
        这代表此时循环已经完全穿透整个界面，根据相关方程计算当前单元的高度函数值
      if $0 < c[] < 1$ **then**
        代表本身处于边界上，需要双方向循环计算积累，判断该单元的函数值
      if 并未发现边界或在循环过程中没有穿透边界 **then**
        需要双方向循环判断是否能后返回高度函数值
return h;

更多的实现细节需要完整的推演以及仔细阅读相应的代码注释，语言文字并不能完整的描述整个精妙的循环过程，读者在阅读推演代码时需要注意几个问题：

1. HSHIFT 的使用方式：代码定义 HSHIFT 为全环境静态值 20，目的是区分界面方向，代码中使用该数值对高度函数值进行加密
2. 多次循环的数据清零问题：在具体代码中方向控制参数与真正计算高度函数的 half column 是相互分离的，在每次方向循环时，half column 中自定义的 S H 会被清零
3. 数据的格式以及 100 加密：函数在表达当前的找寻状态时启用并定义了参数 S，它在第二次正向循环时会被 int 型数据 s 赋值；由于 s 会忽视小数，这给了代码使用 100 对数据进行加密再解析的条件

2.2 heights 及 half column 源代码及注释

```
/**
# Height-Functions

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. This distance is
estimated using the "column integral" of the volume fraction in the
corresponding direction. This integral is not always defined (for
example because the interface is too far i.e. farther than 5.5 cells
in our implementation) in which case the value of the field is set to
```

```
*nodata*. See e.g. [Popinet, 2009](references.bib#popinet2009) for
more details on height functions.

We also store the "orientation" of the height function together with
its value by adding *HSHIFT* if the volume fraction is unity on the
"top" end. The function below applied to the value will return the
corresponding height and orientation.

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

~~~c
(x, y + Delta*height(h.y[])) or (x + Delta*height(h.x[]), y)
~~~
*/

#define HSHIFT 20.//区分正方向

static inline double height (double H) {//内联函数，通过文本替换的方式减少函数
↪  内存的调用
  return H > HSHIFT/2. ? H - HSHIFT : H < -HSHIFT/2. ? H + HSHIFT : H;//先行判
  ↪  断 H 是否大于 HSHIFT/2. 再以此类推
}//本函数的作用是规范高度值，将其完美限定在 [-10,10] 之间

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

/**
We make sure that two layers of ghost cells are defined on the
boundaries (the default is one layer). */

#define BGHOSTS 2

/**
## Half-column integration
```

```
/* This helper function performs the integration on half a column, either
"downward" (*j = -1*) or "upward" (*j = 1*). */
//计算从本网格往一个方向延伸 4 个网格范围内的高度函数值，初始界面网格中高度函数参考值为 20，延伸两侧分别递增和递减，本函数为一个工具函数

static void half_column (Point point, scalar c, vector h, vector cs, int j)//h 是指针，需要填充操作
{

  /**
  The 'state' of the height function can be: *complete* if both
  ends were found, zero or one if one end was found and between zero
  and one if only the interface was found. */

  const int complete = -1;

  foreach_dimension() {//注意其计算了各个方向的高度函数

    /**
     *S* is the state and *H* the (partial) value of the height
     function. If we are on the (first) downward integration (*j =
     -1*) we initialise *S* and *H* with the volume fraction in
     the current cell. */

    double S = c[], H = S, ci, a;
    //需要提醒的是在每一次 j 的数值变化后，S，H 将会被归零
    /**
     On the upward integration (*j = 1*), we recover the state of the
     downward integration. Both the state and the (possibly partial)
     height value are encoded in a single number using a base 100
     shift for the state. */

    typedef struct { int s; double h; } HState;
    HState state = {0, 0};
    if (j == 1) {//在调用本函数时，j 的初始值被设定为-1，所以阅读时请从-1 看起
    /* 在第二次循环（j==1）时会进入本判断语句，本段代码的主要作用就是设置表示当前循环状态的 S 值。
```

```
    当上一个循环完整通过界面后，h.x[] 会被赋值为小于 25 的数值（这取决于其方向），是故在接下来的判断操作中，int s 会取为 0，S 则被设定为-1，代表已经完成循环。
    当上一个循环没有找到界面或没有通过界面一端；
    即向负方向循环遍历后并没有
    1. 初始位置 1 ≥ c > 0，然而在负方向循环上没有出现 c[] = 1 或 c[] = 0
    2. 初始位置 c[] = 1 或 c[] = 0 在负方向循环上没有出现 c[] = 0 或 c = 1
    S 会保持为 c[] 向另一个方向进行遍历，以求得通过界面获得定义，而在此之前 state.s 会被提前设置为-1，state,h 会被提前设置为 nodata。
    而当初始单元处于界面上即 1 ≥ c > 0 在负方向循环中找到单元 c[] = 1 或 c[] = 0 此时 S 则会被赋值为 1 或 0，表示已经完整一半，并记录方向 */
      /**
      Check whether this is an inconsistent height. */

      if (h.x[] == 300.)//inconsistent
^^I       state.s = complete, state.h = nodata;//nodata 就意味着没有找到边界，或者无法定义

      /**
      Otherwise, this is either a complete or a partial height. */

      else {
            int s = (h.x[] + HSHIFT/2.)/100.;//注意这里是 int 值，小数归零
            state.h = h.x[] - 100.*s;
            state.s = s - 1;
      }

      /**
      If this is a complete height, we start a fresh upward
      integration. */

      if (state.s != complete)
            S = state.s, H = state.h;
    }

    /**
     We consider the four neighboring cells of the half column, the
     corresponding volume fraction *ci* is recovered either from the
```

```
113     standard volume fraction field *c* (first two cells) or from the
114     shifted field *cs* (last two cells). The construction of *cs* is
115     explained in the next section. */
116
117   for (int i = 1; i <= 4; i++) {//即开始对某一个方向上的单元进行循环，依次判定
      ↪  是否经过包含边界的单元，以及是否是填充/全空单元
118     ci = i <= 2 ? c[i*j] : cs.x[(i - 2)*j];//cs 即为下文 heights 函数中的 s,
        ↪  cs[] 为 c[2*j] 即向 j 方向移动两单元格后的体积分数值
119     H += ci;//对高度函数进行累加
120
121     /**
122      We then check whether the partial height is complete or not. */
123
124     if (S > 0. && S < 1.) {//即此时起始单元所处位置恰好在边界上
125           S = ci;
126           if (ci <= 0. || ci >= 1.) {//若不能进入该循环，说明下一循环单元还是
              ↪  包含界面
127
128     /**
129      We just left an interfacial cell (*S*) and found a full or
130      empty cell (*ci*): this is a partial height and we can stop
131      the integration. If the cell is full (*ci = 1*) we shift
132      the origin of the height. */
133
134               H -= i*ci;//将初始坐标定在界面内距离界面一格的地方
135               break;
136           }
137   }
138
139     /**
140      If *S* is empty or full and *ci* is full or empty, we went
141      right through he interface i.e. the height is complete and
142      we can stop the integration. The origin is shifted
143      appropriately and the orientation is encoded using the "HSHIFT
144      trick". */
145
146     else if (S >= 1. && ci <= 0.) {//下两种用于判断起始单元位置在界面内/外，
        ↪  循环单元已变为完全界面外/内单元
```

```
147             H = (H - 0.5)*j + (j == -1)*HSHIFT;//HSHIFT 就是用来辨别处理方向的，
                ↪ 当该判别启动时最终的 H 数值会增长畸变大于 10
148             S = complete;
149             break;
150       }
151       else if (S <= 0. && ci >= 1.) {//此二者全部将状态设置为 complete 原因是成
          ↪ 功通过了边界
152             H = (i + 0.5 - H)*j + (j == 1)*HSHIFT;
153             S = complete;
154             break;
155       }
156
157       /**
158        If *ci* is identical to *S* (which is empty or full), we
159        check that *H* is an integer (i.e. its fractional value is
160        zero), otherwise we are in the case where we found an
161        interface but didn't go through it: this is an
162        inconsistent height and we stop the integration. */
163
164       else if (S == ci && modf(H, &a))//modf() 的功能是提取 double 的整数与小
          ↪ 数部分，因为如果经过的单元都是界面内或外，其 H 必定为整数，modf() 必定
          ↪ 为 0，否则就是并没有完全穿透一个界面
165             break;
166    }
167
168    /**
169     We update the global state using the state *S* of the
170     half-integration. */
171
172    if (j == -1) {
173
174       /**
175        For the downward integration, we check that the partial heights
176        (*S != complete*) are consistent: if the first cell is full
177        or empty or if the last cell is interfacial, the partial
178        height is marked as inconsistent. */
179
```

```
180       if (S != complete && ((c[] <= 0. || c[] >= 1.) ||
181             (S > 0. && S < 1.)))
182             /* 当当前状态并非完成且（原始单元并非全满/全空或最终单元在界面上），根
                ↪ 据上一循环，进入该判断条件的应该为
183             1. 单元负方向循环过程中找到界面，但最终没有完整通过界面
184             2. 在整个循环中单元全部保持 c[] = 1 或 c[] = 0*/
185             h.x[] = 300.; // inconsistent，一旦被赋此值，直接会认定为 nodata
186       else if (S == complete)
187             h.x[] = H;
188       else//根据上一循环，有一种情况会进入该判断 1. 起始单元位于边界面上，循环单
          ↪ 元完全离开边界面，此时 S 被赋值为循环单元值
189
190     /**
191     This is a partial height: we encode the state using a base 100
192     shift. */
193
194             h.x[] = H + 100.*(1. + (S >= 1.));//100 加密操作，用于判断在该情况
                ↪ 下 S>=1 是否成立，用于判断界面外向方向，若 S>=1 则说明其脱离界面
                ↪ 进入界面内，界面方向与正方向相同，若不成立，则说明界面方向与正方
                ↪ 向相反
195     }
196     else { // j = 1
197
198       /**
199        For the upward integration, we update the current *state*
200        using the state of the half-integration *S* only if the
201        first downward integration returned a partial height, or if
202        the upward integration returned a complete height with a
203        smaller value than the (complete) height of the downward
204        integration. */
205
206       if (state.s != complete ||
207       (S == complete && fabs(height(H)) < fabs(height(state.h))))//对于之前没
          ↪ 有完成的情况，进行数据更新，其中第二个满足条件为在向负方向循环并没有找
          ↪ 到界面，正方向循环后寻找到完整的界面
208             state.s = S, state.h = H;
209
```

```
      /**
       Finally, we set the vector field *h* using the state and
       height. */

      if (state.s != complete)//即在正负方向上都没有找到两个端点
            h.x[] = nodata;
      else
            h.x[] = (state.h > 1e10 ? nodata : state.h);
    }
  }
}

/**
## Multigrid implementation

The *heights()* function takes a volume fraction field *c* and returns
the height function vector field *h*. */
//计算网格 4 个网格范围内各个方向的高度函数

#if !TREE
trace
void heights (scalar c, vector h)
{

  /**
  We need a 9-points-high stencil (rather than the default
  5-points). To do this we store in *s* the volume fraction field *c*
  shifted by 2 grid points in the respective directions. We make sure
  that this field uses the same boundary conditions as *c*. */

  vector s[];
  foreach_dimension()
    for (int i = 0; i < nboundary; i++)
      s.x.boundary[i] = c.boundary[i];

  /**
  To compute the height function, we sum the volume fractions in a
```

```
  (half-)column starting at the current cell. We start by integrating
  downward (*j = -1*) and then integrate upward (*j = 1*). */

  for (int j = -1; j <= 1; j += 2) {

    /**
    We first build the shifted (by ±2) volume fraction field in each
    direction. */

    foreach()
      foreach_dimension()
        s.x[] = c[2*j];

    /**
    We sum the half-column, downward or upward. We (exceptionally)
    need to allow for stencil overflow. */

    foreach (overflow)
      half_column (point, c, h, s, j);
  }

  /**
  Finally we "propagate" values along columns. */

  column_propagation (h);
}
```

3. column propagation 函数

3.1 函数目的及原理

本函数的目的为：对靠近边界（5.5 个单元长度内）但由于算法被赋值 nodata 的单元进行高度函数赋值。

如下图：

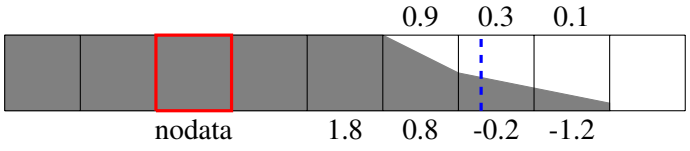


图 5: column propagation 函数值示例图

图中红色方框中的单元正负方向 9 个单元长度内都没有完整的通过边界，是故在 half column 中，其高度函数 h 被赋值为 nodata，经过 column propagation 之后被赋值为 3.8。

3.2 column propagation 函数源代码及注释

```
/**
## Column propagation

Once columns are computed on a local 9-cells-high stencil, we will
need to "propagate" these values upward or downward so that they are
accessible at distances of up to 5.5 cells from the interface. This is
important in 3D in particular where marginal (~45 degrees) cases may
require such high stencils to compute consistent HF curvatures. We do
this by selecting the smallest height in a 5-cells neighborhood along
each direction. */

static void column_propagation (vector h)
{
  foreach (serial) // not compatible with OpenMP
    for (int i = -2; i <= 2; i++)
      foreach_dimension()
          if (fabs(height(h.x[i])) <= 3.5 &&
          fabs(height(h.x[i]) + i) < fabs(height(h.x[])))//该函数目的是保证远
          ↪ 端靠近界面的 nodata 型被赋值
            h.x[] = h.x[i] + i;
}
```

参考文献

[1] Stéphane Popinet. “An accurate adaptive solver for surface-tension-driven interfacial flows”. In: *Journal of Computational Physics* 228.16 (2009), pp. 5838–5866.