My Project

Generated by Doxygen 1.9.3

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 solver::FWave< T > Class Template Reference

Public Types

- enum CellState { **Dry** , **Wet** , **WetDry** , **DryWet** }

  This enum stores the state of the cells: Dry: both cells are dry Wet: both cells are wet WetDry: left cell is wet and the right one is dry DryWet: right cell is dry and the right one is wet.

Public Member Functions

- void determineState (CellState &state)

  This function examines the current state according to the water heights if the water height is below the tolerance the cell is considered to be dry.
- void computeNetUpdates (const T &i_hl, const T &i_hr, const T &i_hul, const T &i_hur, const T &i_bl, const T &i_br, T &o_hUpdateL, T &o_hUpdateR, T &o_huUpdateL, T &o_huUpdateR, T &o_maxWaveSpeed)

  This function computes the netupdates (A-deltaQ, A+deltaQ)

Public Attributes

- T **hl**
- T **hr**
- T **hul**
- T **hur**
- T **bl**
- T **br**

3.1.1 Member Function Documentation

3.1.1.1 computeNetUpdates()

```
template<typename T >
void solver::FWave< T >::computeNetUpdates (
            const T & i_hl,
            const T & i_hr,
            const T & i_hul,
            const T & i_hur,
            const T & i_bl,
            const T & i_br,
            T & o_hUpdateL,
            T & o_hUpdateR,
            T & o_huUpdateL,
            T & o_huUpdateR,
            T & o_maxWaveSpeed )  [inline]
```

This function computes the netupdates (A-deltaQ, A+deltaQ)

Template Parameters

T	(type parameter could be double for example)

Parameters

i_hl	water height of the left cell
i_hr	water height of the right cell
i_hul	momentum of the left cell
i_hur	momentum of the right cell
i_bl	bathymetry of the left cell
i_br	bathymetry of the right cell
o_hUpdateL	output: left-going update of h
o_hUpdateR	output: right-going update of h
o_huUpdateL	output: left-going update of hu
o_huUpdateR	output: right-going update of hu
o_maxWaveSpeed	output: wavespeed(eigenvalue) with the biggest absolute value

3.1.1.2 determineState()

```
template<typename T >
void solver::FWave< T >::determineState (
            CellState & state )  [inline]
```

This function examines the current state according to the water heights if the water height is below the tolerance the cell is considered to be dry.

Parameters

state	This is the output

The documentation for this class was generated from the following file:

- fwaveSolver.hpp

Chapter 4

File Documentation

4.1 fwaveSolver.hpp

```
1 #include<iostream>
2 #include<math.h>
3 #include <cassert>
4 using namespace std;
5
6 namespace solver {
7   template <typename T> class FWave;
8 }
9
10 template <typename T> class solver::FWave{
11   public:
12     T hl;
13     T hr;
14     T hul;
15     T hur;
16     T bl;
17     T br;
18
26     enum CellState{
27       Dry,
28       Wet,
29       WetDry,
30       DryWet
31     };
32
39     void determineState(CellState &state){
40       //tolerance
41       double tol= 0.01;
42
43       if(hl<tol&&hr<tol){
44         state = Dry;
45       }else if(hr<tol){
46         state = WetDry;
47         hr = hl;
48         //wall reflects wave
49         hur = -hul;
50         br = bl;
51
52       }else if(hl<tol){
53         state = DryWet;
54         hl = hr,
55         hul = -hur;
56         bl = br;
57       }else{
58         state = Wet;
59       }
60     }
61
78     void computeNetUpdates (const T &i_hl,  const T &i_hr,
79                             const T &i_hul, const T &i_hur,
80                             const T &i_bl,  const T &i_br,
81                             T &o_hUpdateL,
82                             T &o_hUpdateR,
83                             T &o_huUpdateL,
84                             T &o_huUpdateR,
85                             T &o_maxWaveSpeed )
86     {
87       //the second case is the one for the dry cells
```

```
88        assert(i_hl > 0||(i_hl==0&&i_hul==0&&i_bl>=0));
89        assert(i_hr > 0||(i_hr==0&&i_hur==0&&i_br>=0));
90
91        hl = i_hl;
92        hr = i_hr;
93        hul = i_hul;
94        hur = i_hur;
95        bl = i_bl;
96        br = i_br;
97
98        o_hUpdateL = 0;
99        o_hUpdateR = 0;
100        o_huUpdateL = 0;
101        o_huUpdateR = 0;
102
103
104        CellState state;
105        //Get the current state
106        determineState(state);
107        if(state==Dry){
108          //both sides dry => no update
109          return;
110        }
111
112        int t = 0;
113
114        T ul = hul/hl;
115        T ur = hur/hr;
116        T g = 9.81;
117
118        T hlSqrt = sqrt(hl);
119        T hrSqrt = sqrt(hr);
120
121        T h_roe = 0.5*(hl+hr);
122        T u_roe = (ul*hlSqrt+ur*hrSqrt)/(hlSqrt+hrSqrt);
123
124        T lambda1 = u_roe-sqrt(g*h_roe);
125        T lambda2 = u_roe+sqrt(g*h_roe);
126
127        //determine the wavespeed(eigenvalue) with the biggest absolute value
128        o_maxWaveSpeed = std::max(std::abs(lambda1),std::abs(lambda2));
129
130        //effect of bathymetry
131        T deltaXPsi [2] = {0, -g*(br-bl)*(hl+hr)*0.5} ;
132
133        T fqr[2] = {hur, hr*ur*ur+0.5*g*hr*hr};
134        T fql[2] = {hul, hl*ul*ul+0.5*g*hl*hl};
135
136        //jump in the flux
137        T deltaF[2] = {fqr[0]-fql[0],fqr[1]-fql[1]};
138
139        //due bathymetry
140        deltaF[1] -= deltaXPsi[1];
141
142        T r_inv[2][2];
143        T det = (lambda1 - lambda2);
144
145        r_inv[0][0] = -lambda2 / det;
146        r_inv[0][1] =        1 / det;
147        r_inv[1][0] =  lambda1 / det;
148        r_inv[1][1] = -      1 / det;
149
150        T alpha[2] = {r_inv[0][0] * deltaF[0] + r_inv[0][1] * deltaF[1],
151          r_inv[1][0] * deltaF[0] + r_inv[1][1] * deltaF[1]};
152
153
154        //compute the actual netupdates (A-deltaQ, A+deltaQ)
155        //first iteration(p=1)
156        if(lambda1>0){
157            o_hUpdateR=alpha[0];
158            o_huUpdateR=alpha[0]*lambda1;
159        }else{
160            o_hUpdateL = alpha[0];
161            o_huUpdateL = alpha[0]*lambda1;
162        }
163
164        //second iteration (p=2)
165        if(lambda2>0){
166            //calculates Z2
167            o_hUpdateR+=alpha[1];
168            o_huUpdateR+=alpha[1]*lambda2;
169        }else{
170            o_hUpdateL+=alpha[1];
171            o_huUpdateL+=alpha[1]*lambda2;
172        }
173
174        //Dry wall on the right => no update
```

```
175       if(state==WetDry){
176         o_hUpdateR = 0;
177         o_huUpdateR = 0;
178       }else if(state==DryWet){
179         o_hUpdateL = 0;
180         o_hUpdateR = 0;
181       }
182 
183     // std::cout«"i_hl: "«i_hl«", i_hr: "«i_hr«", i_hul: "«i_hul«", i_hur: "«i_hur«", i_bl: "«i_bl«",
        i_br: "«i_br«", o_hUpdateL: "«o_hUpdateL«", o_hUpdateR: "«o_hUpdateR«", o_huUdateL: "«o_huUpdateL«",
        o_huUpdateR: "«o_huUpdateR«"\n";
184     }
185 };
```

Index