Free Surface Flows

M. Farahani, E. Wolter, A. Hahn

03.02.2014

Finde eine stückweise zwei mal stetig differenzierbare Bahnkurve der Hinterachse $\Phi \in C^1([0, t^*], \mathbb{R}^2)$, sodass

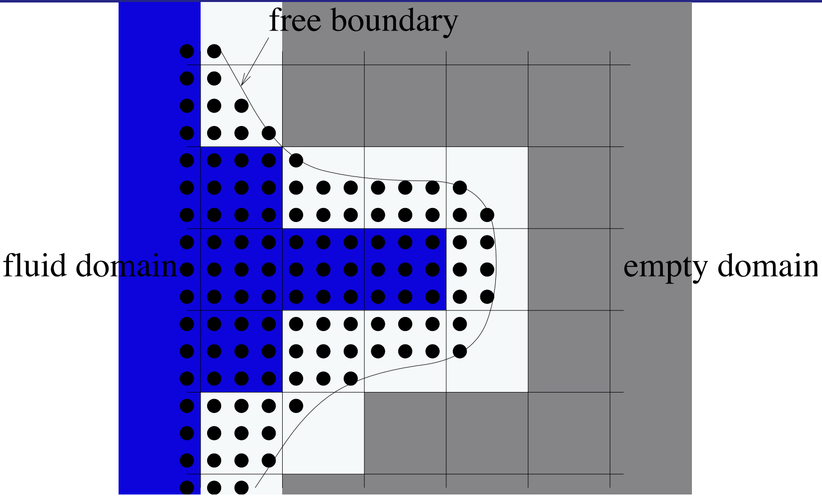
(I) das Auto zu jedem Zeitpunkt t in einem Gebiet G ist

(II) die Randbedingungen für $\Phi(0), [\Phi(t^*)]_2$ erfüllt sind und
$$\frac{\Phi'(0)}{\|\Phi'(0)\|} = \frac{\Phi'(t^*)}{\|\Phi'(t^*)\|} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

(III) für $-\frac{\Phi'(t)}{\|\Phi'(t)\|_2} = \begin{pmatrix} \cos\beta(t) \\ \sin\beta(t) \end{pmatrix}$
$\beta(t) \in \left[0, \frac{\pi}{2}\right[$ für alle $t \in [0, t^*]$ erfüllt ist

(IV) (Wendekreisbeschränkung erfüllt)

[c]0.45


free boundary
fluid domain
empty domain

One empty neighbor

Images/one.pdf

Two empty neighbor-common corner

Images/two1.pdf

Two empty neighbor-opposite side

Images/two2.pdf

Three empty neighbor

Images/three.pdf

Four empty neighbor

Images/four.pdf

Particle and ParticleTracer

- **Particle(real x, real y, int type)**
  Has some functions which can detect its position on the grid
- **ParticleTracer(StaggeredGrid *grid)**
  Has a vector of particles
  - **void markCells()**
  - **void fillCell(int i, int j, int numParticles, int type)**
  - **void addRectangle(real x1, real y1, real x2, real y2, int type)**
  - **void addCircle(real x, real y, real r, int type)**
  - **void advanceParticles(real const dt)**

Types and StaggeredGrid

- **Types.hh**:
  - **flag EMPTY**
- **StaggeredGrid.cc**:
  - **int ppc_**
  - **bool isEmpty(const int x, const int y)**
  - **void setCellToEmpty(int x, int y)**
  - **void refreshEmpty()**

FluidSimulator

- **FluidSimulator.cc**:
  - **real rectX1_particle_, rectX2_particle_ , ...**
  - **real circR_particle_, circX_particle_, ...**
  - **void set_UVP_surface(int i, int j , const real &dt, bool compP)**
  - **void one_empty_neighbour(int i , int j , const real &dt, bool compP)**
  - ...
  - **four_empty_neighbour(int i , int j , const real &dt, bool compP)**
  - **void refreshEmpty()**

Main while-loop

```
while (n <= nrOfTimeSteps)
{
    ...
    determineNextDT(safetyfac_);
    particle_tracer_.markCells();
    set_UVP_surface(dt_, true);
    computeFG();
    composeRHS();
    solv().solve(grid_);
    updateVelocities();
    refreshBoundaries();
    set_UVP_surface(dt_, false);
    particle_tracer_.advanceParticles(dt_);
    ...
}
```

Breaking dam with outflow at the east wall

$$
\Phi(t) = \begin{cases}
\Phi(0) - \begin{pmatrix} 1 \\ 0 \end{pmatrix} t & \text{für } t \in [0, t_0] \\
M^1 + r_1 \begin{pmatrix} -\sin\left(\frac{t-t_0}{r_1}\right) \\ \cos\left(\frac{t-t_0}{r_1}\right) \end{pmatrix} & \text{für } t \in [t_0, t_1] \\
\Phi(t_1) + \begin{pmatrix} -\cos\left(\frac{t_1-t_0}{r_1}\right) \\ -\sin\left(\frac{t_1-t_0}{r_1}\right) \end{pmatrix} (t - t_1) & \text{für } t \in [t_1, t_2] \\
M^2 + r_2 \begin{pmatrix} \sin\left(\frac{t^*-t}{r_2}\right) \\ -\cos\left(\frac{t^*-t}{r_2}\right) \end{pmatrix} & \text{für } t \in [t_2, t^*]
\end{cases}
$$

Breaking dam with free-slip at the east wall

Falling drop

- Umschreibung der Bedingungen in die Variablen t_0, t_1, r_1 und t_2.
- Lösung des entstehenden Optimierungsproblems