C++ Teil 13

Sven Groß

25. Jan 2016

Themen der letzten Vorlesung

- Wdh. Klassen: `Array`-Klasse schreiben
- Konstruktoren: Initialisierungsliste
- Friends
- Statische Attribute+Methoden

Heutige Themen

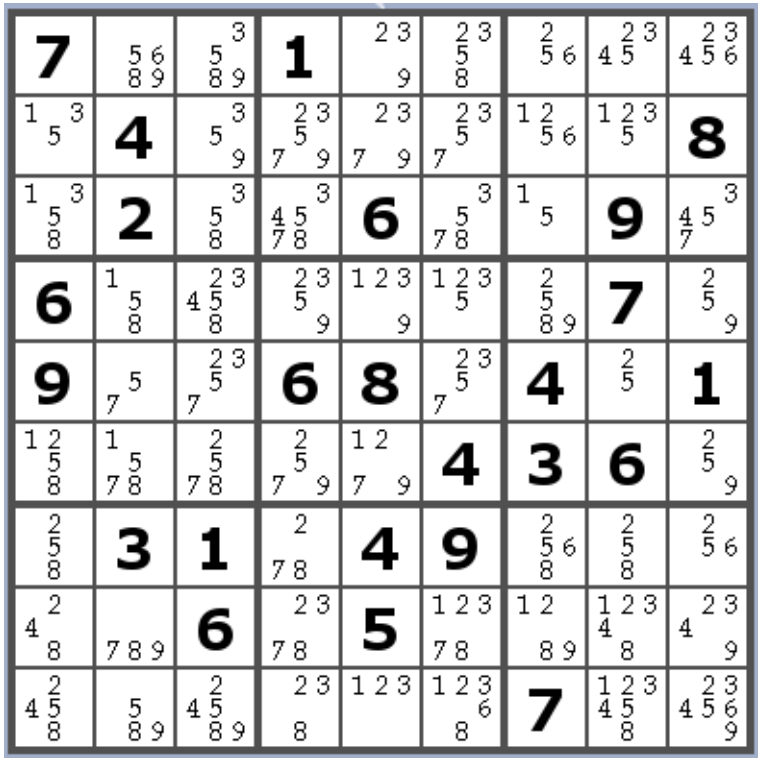
1. Besprechung der Sudoku-Aufgabe
2. Projekte mit mehreren Dateien
3. Klassen-Design von `SudokuSolver`
4. Werbung: Mathe + Programmieren = Numerik

Sudoku-Klasse in eigene Datei auslagern

- Bisher: 1 Programm = 1 Datei, z.B. `main.cpp`
- wird für größere Projekte unübersichtlich, deswegen aufteilen in mehrere Dateien
  - **Module**, die bestimmte Aufgaben übernehmen: Klassen, Funktionen, z.B. alles zum Thema Sudoku
  - pro Modul
    - 1 **Header-Datei** (z.B. `sudoku.h`, enthält ***Schnittstellen***)
    - 1 **Source-Datei** (z.B. `sudoku.cpp`, enthält ***Implementierung***).
  - Hauptprogramm, das die benötigten Module **einbindet** (z.B. `#include "sudoku.h"` am Anfang von `main.cpp`)
- Übersetzen aller Module und Hauptprogramme in Objektdateien (z.B. `sudoku.o`, `main.o`)
- Linken aller Objektdateien zu fertigem Programm
- In `Code::Blocks`: Hinzufügen der Header-/Source-Dateien zum Projekt, Compilieren/Linken funktioniert dann automatisch

Sudokus lösen: Hilfszahlen

7			1					
	4							8
	2			6			9	
6							7	
9			6	8		4		1
					4	3	6	
	3	1		4	9			
		6		5				
						7		



Hilfszahlen kennzeichnen, welche Ziffern möglich sind:

kommen nicht in der zugehörigen Zeile vor,

und kommen nicht in der zugehörigen Spalte vor,

und kommen nicht in dem zugehörigen Quadrat vor.

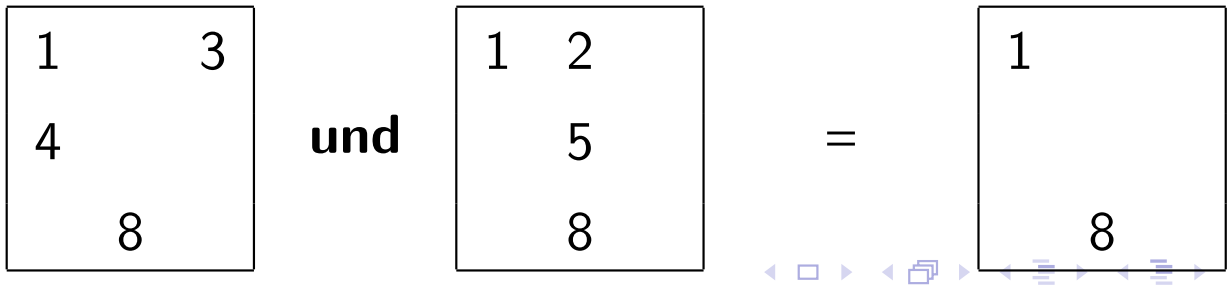
Datentyp für Hilfszahlen

```
class possibleDigits {
  private:
    vector<bool> possible;  // 9 bool-Eintraege
  public:
    possibleDigits( bool allPossible= true);
                                // Konstruktor mit Init.wert

    bool isPossible( int digit) const; // Ziffer abfragen
    void enable( int digit);           // Ziffer setzen
    void disable( int digit);          // Ziffer loeschen
};
```

Außerdem: `operator&&` für logisches Und:

```
possibleDigits operator&&( const possibleDigits& a,
                           const possibleDigits& b);
```

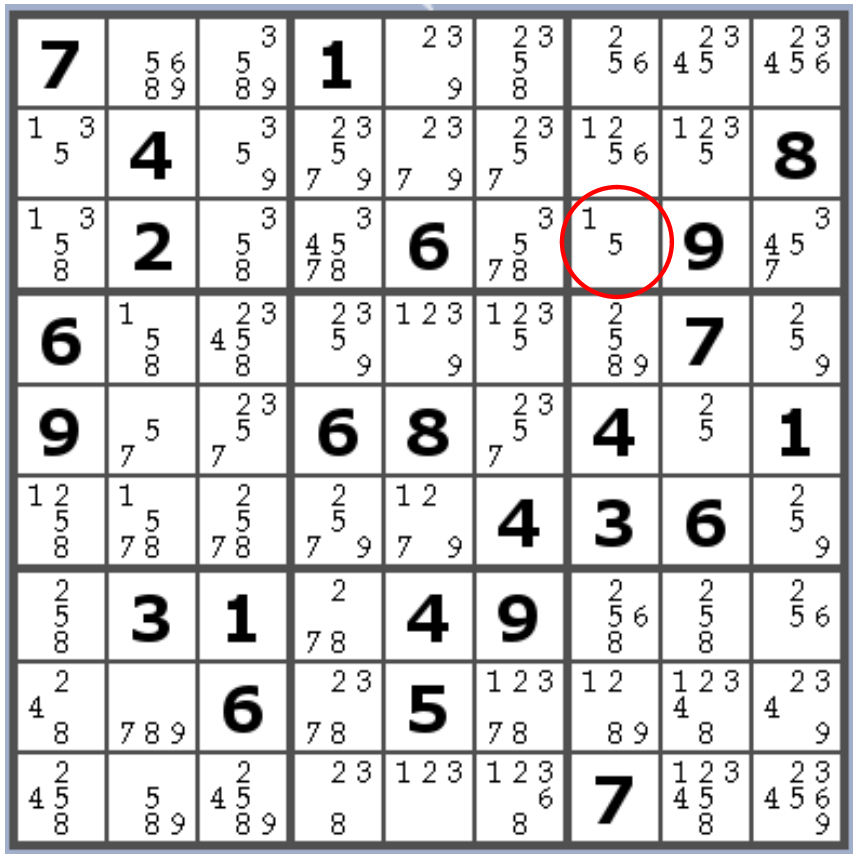
Beispiel:



Lösungsstrategie – Beispiel

Cleveres Durchprobieren aller Möglichkeiten:

1. Wir wählen ein leeres Feld mit *möglichst wenigen* Hilfszahlen, damit wir weniger durchprobieren müssen.



Lösungsstrategie – Beispiel

Cleveres Durchprobieren aller Möglichkeiten:

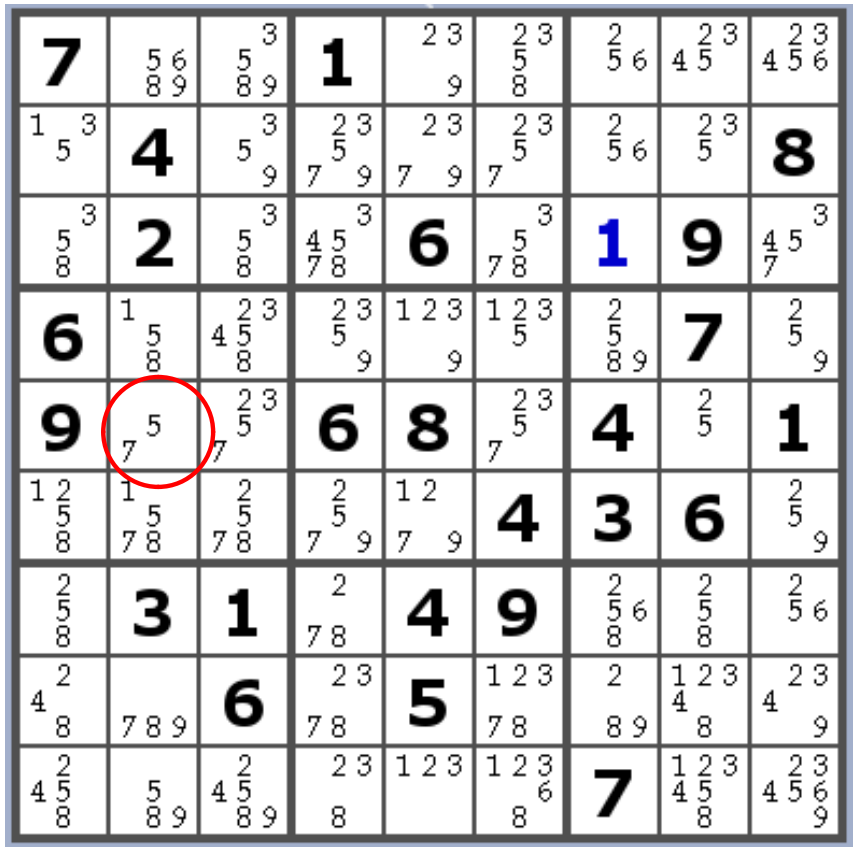
1. Wir wählen ein leeres Feld mit *möglichst wenigen* Hilfszahlen, damit wir weniger durchprobieren müssen.
2. Wir probieren der Reihe nach die möglichen Ziffern durch. Starte mit der **1**.

7	5 6 8 9	3 5 8 9	**1**	2 3 9	2 3 5 8	2 5 6	2 3 4 5	2 3 4 5 6
1 3 5	**4**	3 5 9	2 3 5 7 9	2 3 7 9	2 3 5 7	2 5 6	2 3 5	**8**
3 5 8	**2**	3 5 8	3 4 5 7 8	**6**	3 5 7 8	**1**	**9**	3 4 5 7
6	1 5 8	2 3 4 5 8	2 3 5 9	1 2 3 9	1 2 3 5	2 5 8 9	**7**	2 5 9
9	5 7	2 3 5 7	**6**	**8**	2 3 5 7	**4**	2 5	**1**
1 2 5 8	1 5 7 8	2 5 7 8	2 5 7 9	1 2 7 9	**4**	**3**	**6**	2 5 9
2 5 8	**3**	**1**	2 7 8	**4**	**9**	2 5 6 8	2 5 8	2 5 6
2 4 8	7 8 9	**6**	2 3 7 8	**5**	1 2 3 7 8	2 8 9	1 2 3 4 8	2 3 4 9
2 4 5 8	5 8 9	2 4 5 8 9	2 3 8	1 2 3	1 2 3 6 8	**7**	1 2 3 4 5 8	2 3 4 5 6 9

Lösungsstrategie – Beispiel

Cleveres Durchprobieren aller Möglichkeiten:

1. Wir wählen ein leeres Feld mit *möglichst wenigen* Hilfszahlen, damit wir weniger durchprobieren müssen.
2. Wir probieren der Reihe nach die möglichen Ziffern durch. Starte mit der **1**.
3. Auf das modifizierte Sudoku wenden wir wieder rekursiv unsere Strategie an.

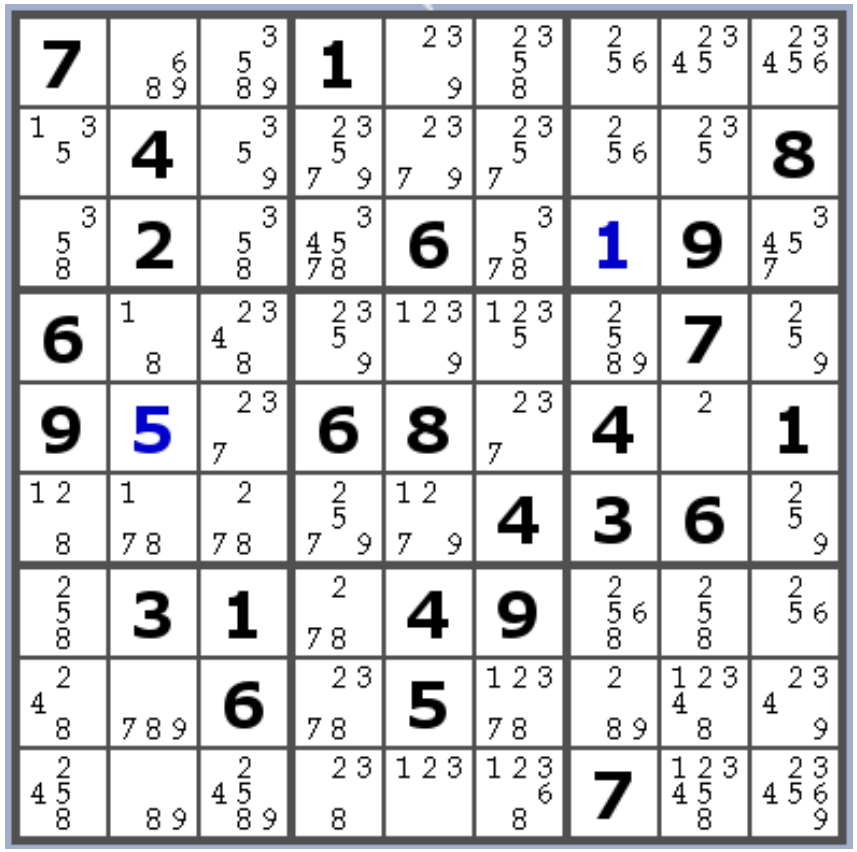


Lösungsstrategie – Beispiel

Cleveres Durchprobieren aller Möglichkeiten:

1. Wir wählen ein leeres Feld mit *möglichst wenigen* Hilfszahlen, damit wir weniger durchprobieren müssen.
2. Wir probieren der Reihe nach die möglichen Ziffern durch. Starte mit der **1**.
3. Auf das modifizierte Sudoku wenden wir wieder rekursiv unsere Strategie an.

   usw. . .



Lösungsstrategie

Cleveres Durchprobieren aller Möglichkeiten, etwas formaler:

Strategie `solve`

1. Wähle ein freies Feld (r, c) mit möglichst wenigen Hilfszahlen.
2. Für alle möglichen Ziffern d in Feld (r, c):
   a) Setze Ziffer d in Feld (r, c) und passe Hilfszahlen an.
   b) Wende (rekursiv) die Strategie `solve` an.
   Wenn das erfolgreich war, sind wir **fertig** und melden den Erfolg zurück.
   c) Mache Schritt a) rückgängig.
3. Wenn wir hier landen, waren wir nicht erfolgreich.
   Melde den Misserfolg zurück.

- **Rekursionstiefe:** Anzahl der freien Felder (`numEmpty`)
- **Verzweigungsgrad:** Anzahl der möglichen Ziffern in der jeweiligen Stufe

Lösungsstrategie (2)

Strategie `bool solve (int numEmpty)`

0. Falls `numEmpty==0`, melden wir Erfolg/Misserfolg zurück.
1. Wähle ein freies Feld (r, c) mit möglichst wenigen Hilfszahlen.
   `getNextCell( r, c);`
2. Für alle möglichen Ziffern d in Feld (r, c):
   a) Setze Ziffer d in Feld (r, c) und passe Hilfszahlen an.
      `setDigit( r, c, d);`
   b) Wende (rekursiv) die Strategie `solve` an.
      `bool success= solve( numEmpty-1);`
      Wenn das erfolgreich war, sind wir **fertig** und melden den Erfolg zurück.
      `if (success) return true;`
   c) Mache Schritt a) rückgängig.
      `unsetDigit( r, c);`
3. Wenn wir hier landen, waren wir nicht erfolgreich.
   Melde den Misserfolg zurück. `return false;`

Klasse für den Sudokulöser

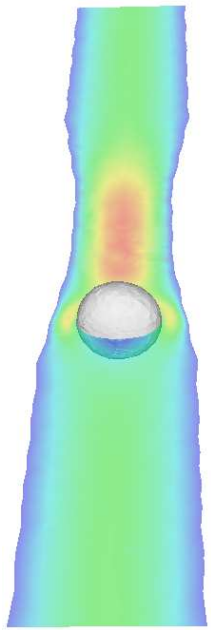
```
class SudokuSolver {
  private:
    Sudoku&                    Sudo;
    vector<possibleDigits>     pdRow, pdCol, pdSqr;
                               // jeweils 9 Eintraege
  public:
    SudokuSolver( Sudoku& S);

    void getNextCell (int& r, int& c) const;

    possibleDigits getPossible( int r, int c) const;

    void   setDigit (int r, int c, int digit);
    void unsetDigit (int r, int c);

    bool solve (int numEmpty);
};
```

Numerik und Simulation

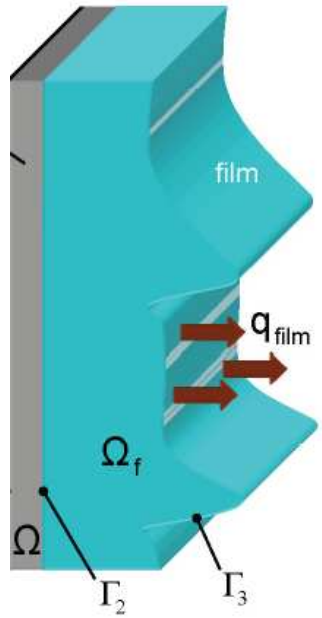
- **Simulation** von Strömungen kennt jeder: Wetterbericht
- andere Anwendungen: z.B. virtuelle Crashtests in der Automobilindustrie, Umströmung von Flugzeugen (virtueller Windkanal)

- Mein Forschungsgebiet: Simulation von **Zweiphasenströmungen**
  - Entwicklung numerischer Methoden
  - Analyse (Satz - Beweis)
  - Implementierung
    → Programmpaket DROPS

www.igpm.rwth-aachen.de/DROPS/

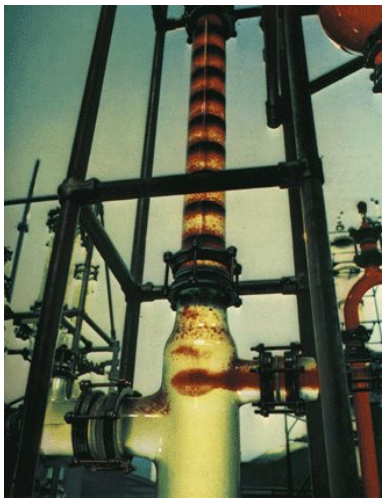


Levitierter Tropfen



Fallfilm

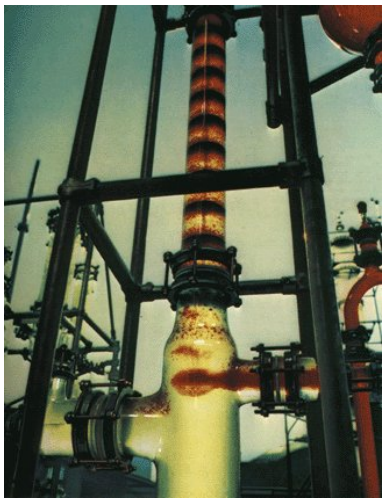
Zweiphasenströmungen in der Verfahrenstechnik



Extraktionskolonne (flüssig-flüssig)

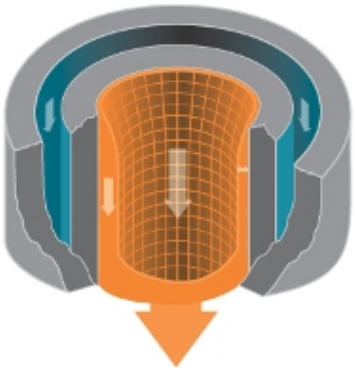
- Stofftransport: aufsteigender Tropfenschwarm ↔ umgebende Flüssigkeit
- *Anwendung:* thermisch schonende Stofftrennung,
  z.B. Vitaminherstellung

Zweiphasenströmungen in der Verfahrenstechnik



Extraktionskolonne (flüssig-flüssig)

- Stofftransport: aufsteigender Tropfenschwarm ↔ umgebende Flüssigkeit
- *Anwendung:* thermisch schonende Stofftrennung, z.B. Vitaminherstellung



Fallfilm (flüssig-gasförmig)

- dünner Film an Rohrinnenwand, Heizung/Kühlung von außen
- *Anwendung:* Fruchtsaftkonzentration, ESL-Milch, alkoholfreies Bier

SFB 540 — computational engineering science

Interdisciplinary research project **SFB 540** with chemical engineers
Topic: Modeling of interfacial phenomena in multi-phase flows.

$\Rightarrow$ better design of chemical engineering plants

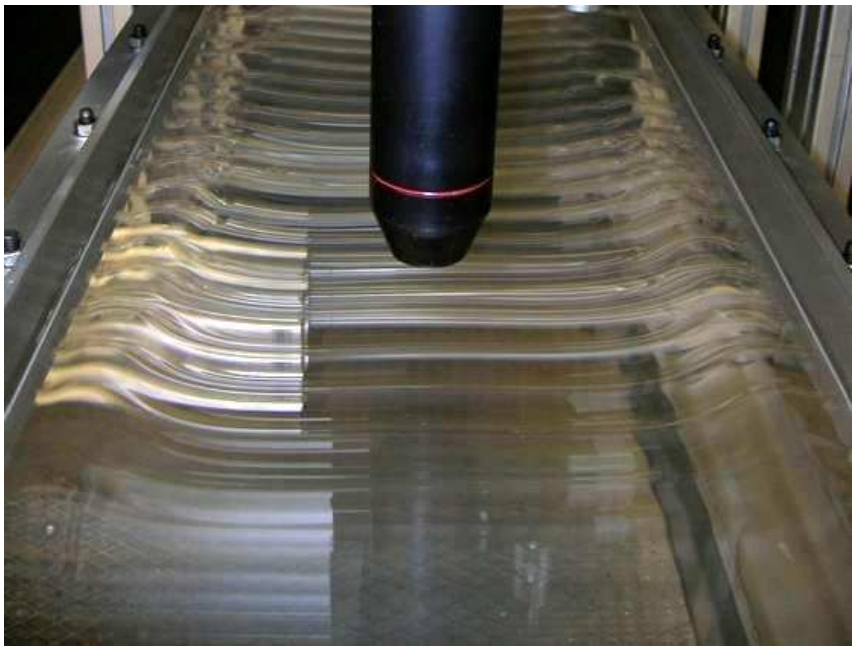
Measurements $\longleftrightarrow$ Inverse Problems $\longleftrightarrow$ Numerical Simulation

SFB 540 — computational engineering science

Interdisciplinary research project **SFB 540** with chemical engineers
Topic: Modeling of interfacial phenomena in multi-phase flows.
⇒ better design of chemical engineering plants

Measurements ⟷ Inverse Problems ⟷ Numerical Simulation

levitated droplet, NMR measurements of velocities

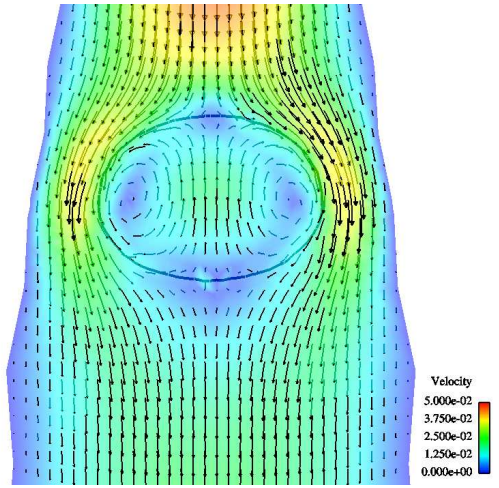


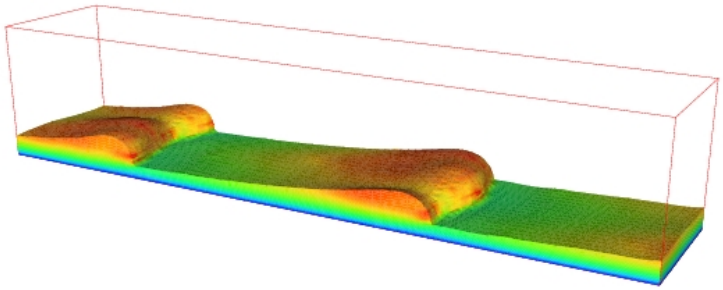
falling film, measurements of film thickness (CCI), velocities (PIV), ...

Numerical simulation of droplet and film systems

DROPS package, under development:

- 3D instationary Navier-Stokes (incompressible), Low Reynolds numbers
- FE discretization (Hood-Taylor) on tetrahedral grids
- adaptive multilevel refinement
- interface capturing: level set technique
- enhanced numerical treatment of surface tension: pressure XFEM, Laplace-Beltrami
- velocity XFEM for large viscosity jumps (liquid/gas)





www.igpm.rwth-aachen.de/DROPS/

Strategy

FE simulation in 3D:

- typical problem size n: $10^5 - 10^8$
- uniform refinement: problem size n increased by factor 8
- solution of large sparse linear systems of equations

$$\mathbf{A}\vec{x} = \vec{b}, \qquad \mathbf{A} \in \mathbb{R}^{n \times n}, \quad \vec{x}, \vec{b} \in \mathbb{R}^n$$

- big computational and memory effort

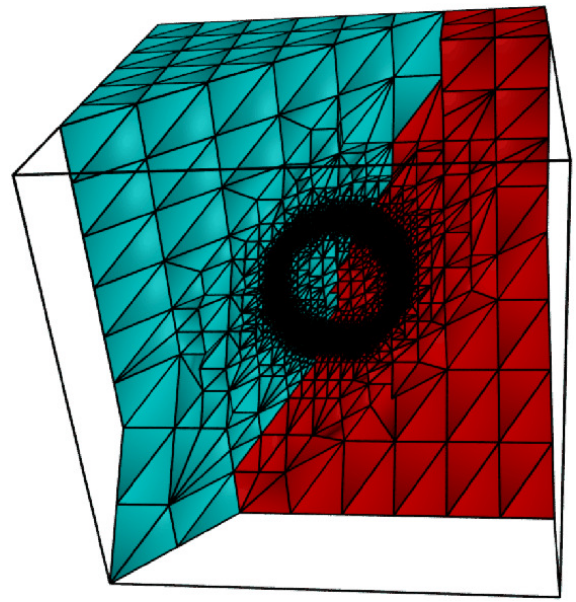
$\rightsquigarrow$ strategies:

a) reduce computational effort

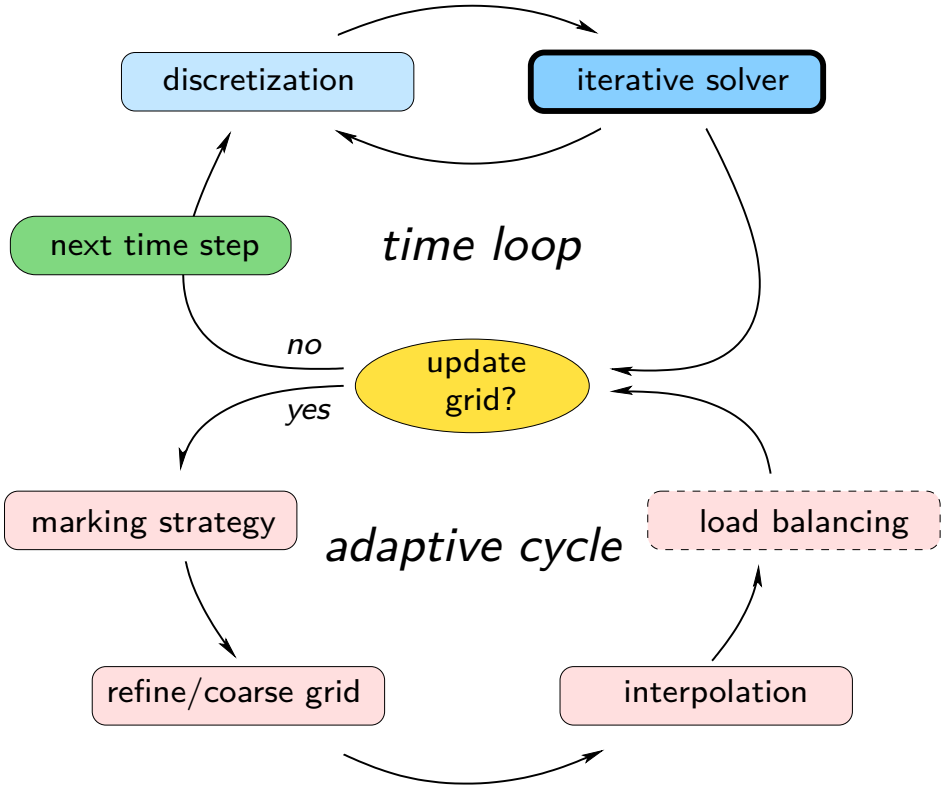
- **adaptivity**: reduce problem size n
- efficient methods, e.g. **multigrid** solvers: $\mathcal{O}(n)$

b) distribute computational load

- **parallelization**: MPI (distributed mem.), OMP (shared mem.)



Adaptive cycle



per time step

- most time consuming part: iterative solver
- coupling of flow and interface tracking: frequent discretization updates

Adaptivity

- refinement zone around interface
- update grid, if interface has moved too far

DROPS software structure