Conjugate Heat Transfer of Cooling Channels using COOLFluiD 3

Sebastian Scholl

von Karman Institute for Fluid Dynamics

Doctoral Seminar, June 06, 2012

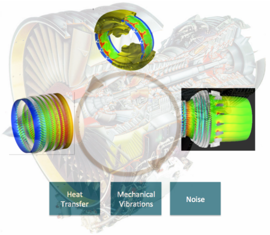
Section Outline

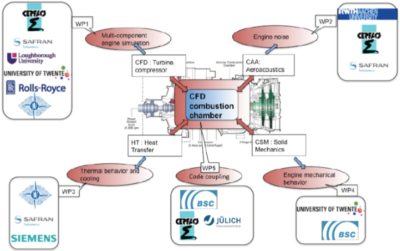
1. Introduction
2. COOLFluiD 3
3. Programming techniques
4. Application
5. Conclusions and future steps

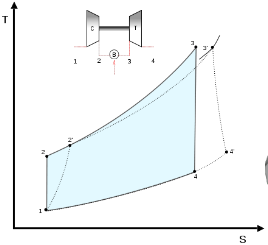
COPA-GT

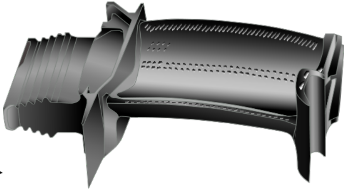
What is COPA-GT ?

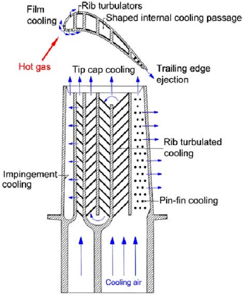
- COupled PArallel simulations of Gas Turbines
- Collaborative research project











Goal of the project

Simulation of a cooling channel in a turbine blade

Conjugate Heat Transfer

Modes of heat transfer

- Conduction
- Convection
- Radiation

How to deal with CHT ?

- Uncoupled
- Conjugate
- Coupled

Coupling methods

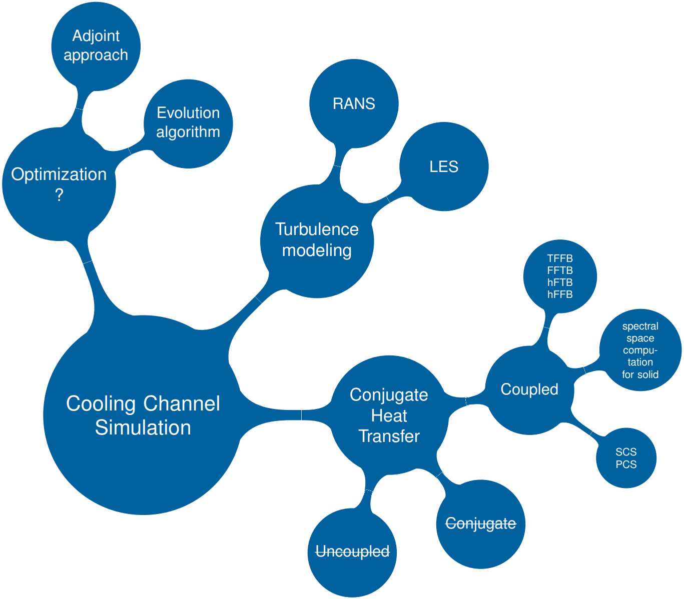
$$\boxed{fluid} \underset{T}{\overset{q_{wall}}{\rightleftarrows}} \boxed{solid}$$

$$\boxed{fluid} \underset{T}{\overset{h,\, T_{fluid}}{\rightleftarrows}} \boxed{solid}$$

$$\boxed{fluid} \underset{q_{wall}}{\overset{T}{\rightleftarrows}} \boxed{solid}$$

$$\boxed{fluid} \underset{q_{wall}}{\overset{h,\, T_{fluid}}{\rightleftarrows}} \boxed{solid}$$

Objectives


Adjoint approach
Evolution algorithm
Optimization ?
RANS
LES
Turbulence modeling
TFFB
FFTB
hFTB
hFFB
spectral space compu-tation for solid
Coupled
SCS
PCS
Cooling Channel Simulation
Conjugate Heat Transfer
~~Conjugate~~
~~Uncoupled~~

Section Outline

1. Introduction
2. COOLFluiD 3
3. Programming techniques
4. Application
5. Conclusions and future steps

Current State

Kernel

- Simulation environment focused on complex multi physics
- Conceptually more mature than earlier version
- Component-based architecture, object oriented, generic, event driven
- Parallel, unique Pre- and Postprocessing features
- Control via python, CF-script and GUI
- https://coolfluid.github.com (LGPLv3 license)

Discretization and models

- UFEM, RDM, Spectral-FDM
- Compr. Euler and NS, incompr. NS, Conduction

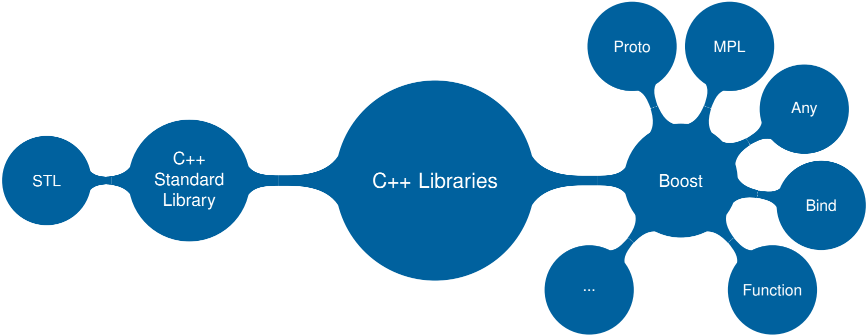
UFEM

- A plugin among others in COOLFluiD 3
- Discretization of PDE with Finite Elements
- Further techniques used: metaprogramming, expression templates to build a Domain Specific Language (DSL)
- This DSL was developed by using Boost::Proto

Boost::Proto

What are Boost and Proto ?

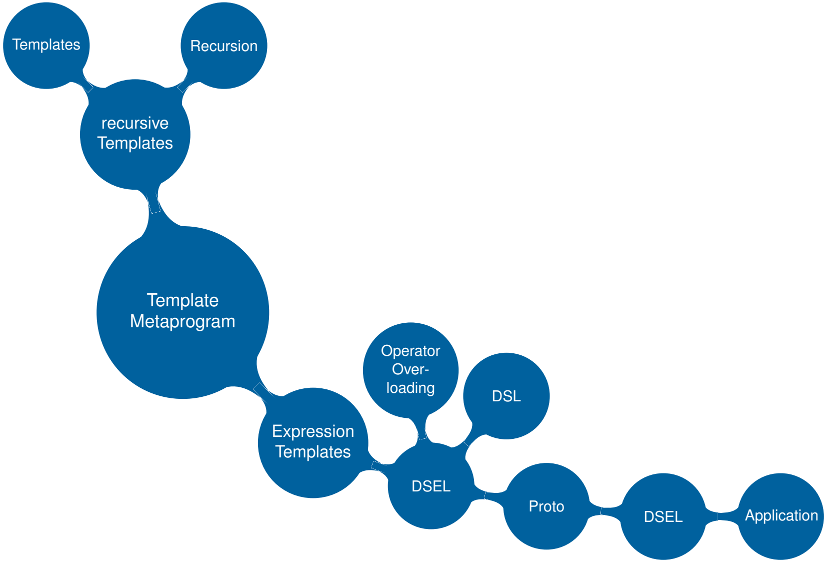
- Boost is a set of about 80 libraries
- MPL for Metaprogramming
- Proto for Domain Specific Languages



Section Outline

1. Introduction
2. COOLFluiD 3
3. Programming techniques
4. Application
5. Conclusions and future steps

Overview



Templates
Recursion
recursive Templates
Template Metaprogram
Expression Templates
Operator Over-loading
DSL
DSEL
Proto
DSEL
Application

What is Template Metaprogramming ?

What is a Metaprogram ?

- Code that produces other code
- Code conducting computations at compile time

What is a Template Metaprogram ?

- Metaprogram using Templates to compute something at compile time
- relies on other techniques: recursive templates, traits, type functions

Review on recursion

What is recursion ?

- Solving a problem by reducing it to smaller versions of itself

What is a recursive function ?

- A function that calls itself
- Different to iterative functions which use control structures to repeat a set of statements

Review on recursion

An easy example - computaion of the factorial n!

$$
\begin{aligned}
0! &= 1 && \text{(base case)} \\
n! &= n * (n-1)! && \text{(general case)}
\end{aligned}
$$

$$
\begin{aligned}
3! &= 3 * 2! \\
2! &= 2 * 1! \\
1! &= 1 * 0!
\end{aligned}
$$

Factorial - recursive

```
int factorial(int n)
{
  if (n > 0)
  return n*factorial(n-1);
  else return 1;
}
```

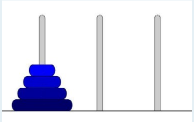
Factorial - iterative

```
int factorial(int n)
{
  int result = 1;
  for(int i=1;i<=n;i++) result=i*result;
  return result;
}
```

Review on recursion

Another example - The tower of Hanoi

- set of disks on a spindle
- move from source to destination
- move only one disk at a time
- only place disk on top of larger disk



Recursive solution for the tower of Hanoi problem

```
void moveDisk(int count, int n1, int n3, int n2)
{
  if (count > 0)
  {
    moveDisk(count-1, n1, n2, n3);
    cout << " Move disk " << count << " from " << n1 << " to " << n3 <<
     endl;
    moveDisk(count-1, n2, n3, n1);
  }
}
```

A recursive Template as a Template Metaprogram

Header file for factorial

```
template <int N>
struct Factorial
{
   enum{value =
     N*Factorial<N-1>::value};
};

template <>
struct Factorial<0>
{
   enum{value = 1};
};
```

Source code, factorial

```
// example use
#include <iostream>
#include "Metaprogram.h"

using namespace std;

int main()
{
     const int fact =
      Factorial<4>::value;
     cout << fact << endl;
     return 0;
}
```

Is that useful ?

- not very much yet
- the computation is done at compile time and the value can be used as a constant at run time

Example: The dot product

Function to compute the dot product

```
double dot(const vector<double>& u, const vector<double>& v)
{
 double dotprod = 0.;
 int n = u.size();
 for (int i = 0; i<n; i++) dotprod += u[i]*v[i];
 return dotprod;
}
```

Alternative definition to calculate the dot product

```
inline double dot3(const vector<double>& u, const vector<double>& v)
{
  return u[0]*v[0] + u[1]*v[1] + u[2]*v[2];
}
```

The dot product computation as Metaprogram

Header file for dot product

```
template <int DIM, typename T>
class DotProduct
{
  public:
    static T result (T* a, T* b){
      return *a * *b +
     DotProduct<DIM-1,T>::result(a+1,b+1);
    }
};
template <typename T>
class DotProduct<1,T>{
  public:
    static T result (T* a, T* b){
      return *a * *b;
    }
};
template <int DIM, typename T>
inline T dot_product (T* a, T* b)
{
  return DotProduct<DIM,T>::result(a,b);
}
```

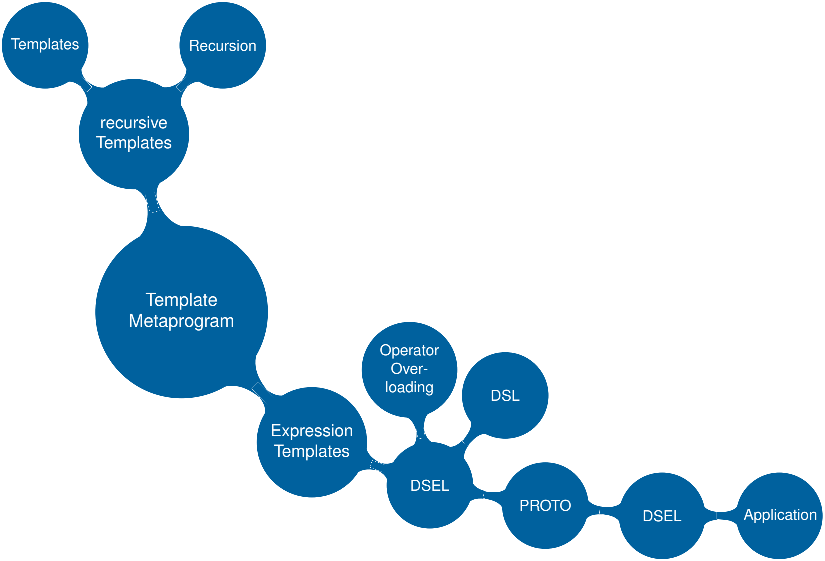
Unrolling loop

```
dot_product<>(a,b)
= DotProduct<3>::result(a,b)
= a[2]*b[2] +
     DotProduct<2>::result(a,b)
= a[2]*b[2] + a[1]*b[1] +
     DotProduct<1>::result(a,b)
= a[2]*b[2] + a[1]*b[1] +
     a[0]*b[0]
     DotProduct<1>::result(a,b)
```

Source code, dot product

```
int main(){
  int a[3] = {2, 1, 3};
  int b[3] = {1, 4, 1};
  cout << " dot_product<3>(a,b) = "
       << dot_product<3>(a,b) <<
    endl;}
```

Overview


Templates
Recursion
recursive Templates
Template Metaprogram
Operator Over-loading
DSL
Expression Templates
DSEL
PROTO
DSEL
Application

What are Expression Templates ?

- Templates which also use recursive instantiations
- Relative of the template metapgrogram
- Similar to operator overloading
- Expressions are not immediately evaluated (call by need)
- Are used to build Domain Specific Languages (DSL)

What is a Domain Specific Language ?

- Language closer to the demand of the developer
- A Domain Specific Embedded Language (DSEL) is embedded into another programming language (e.g. C++)

What is PROTO ?

- Proto is a Library to build DSEL (embedded into C++)
- Proto itself is a DSEL (embedded into C++)

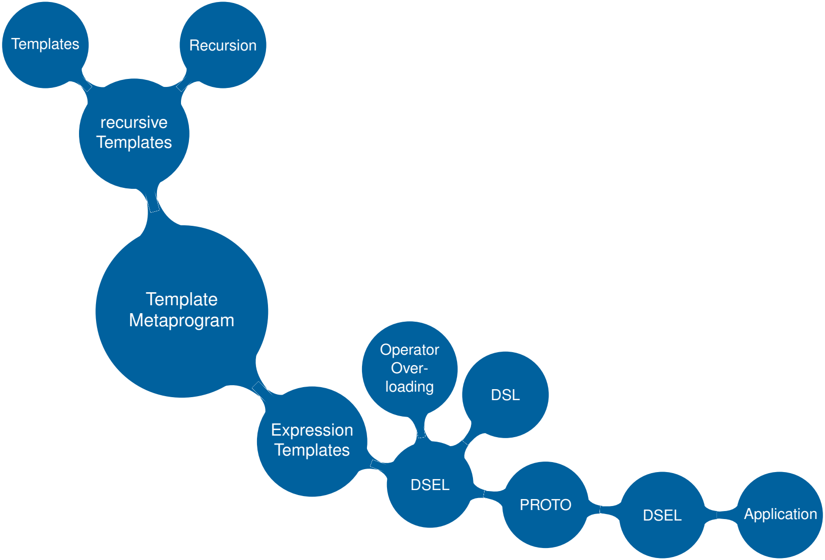
How to use PROTO ?

- Defining a so-called terminal as a placeholder
- Build expressions with the terminals
- Specify, how the expression will be evaluated. Define an evaluation context, else PROTO does default evaluation.
- PROTO will do all the expression templates and overloading on its own.
- Using the expression.

Section Outline

1. Introduction
2. COOLFluiD 3
3. Programming techniques
4. Application
5. Conclusions and future steps

Overview


Templates
Recursion
recursive Templates
Template Metaprogram
Expression Templates
Operator Over-loading
DSL
DSEL
PROTO
DSEL
Application

The continuity equation

Differential form

$$\nabla \cdot \mathbf{u} = 0$$

Stabilized weak form

$$\int_\Omega \omega \nabla \cdot \mathbf{u}^{n+1} \mathrm{d}\Omega + \int_\Omega \tau_{pspg} \nabla \omega \cdot \left(\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \left(\mathbf{u}^{n+1} \cdot \nabla \right) \mathbf{u}^{n+1} + \frac{\nabla p^{n+1}}{\rho} \right) \mathrm{d}\Omega = 0$$

Differential form

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\, \mathbf{u} + \frac{\nabla p}{\rho} - \nu \nabla^2 \mathbf{u} = 0$$

Stabilized weak form

$$\int_\Omega \left(\omega + \tau_{supg} \nabla \omega \cdot \mathbf{u}^{n+1} \right) \left(\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \left[(\mathbf{u} \cdot \nabla)\, \mathbf{u} + \frac{\nabla p}{\rho} - \nu \nabla^2 \mathbf{u} \right]^{n+1} \right) \mathrm{d}\Omega + \int_\Omega \tau_{bulk} \nabla \omega \left(\nabla \cdot \mathbf{u}^{n+1} \right) \mathrm{d}\Omega = 0$$

Element Matrix

Matrix form for each element

$$\frac{1}{\Delta t} T_e \left(\mathbf{x}_e^{n+1} - \mathbf{x}_e^{n} \right) + A_e \mathbf{x}_e^{n+1} = 0$$

Split sub-matrices

$$A_e = \begin{bmatrix} A_{pp} & A_{pu_0} & A_{pu_1} & A_{pu_2} \\ A_{u_0 p} & A_{u_0 u_0} & A_{u_0 u_1} & A_{u_0 u_2} \\ A_{u_1 p} & A_{u_1 u_0} & A_{u_1 u_1} & A_{u_1 u_2} \\ A_{u_2 p} & A_{u_2 u_0} & A_{u_2 u_1} & A_{u_2 u_2} \end{bmatrix}$$

Discretization procedure for the continuity equation

Stabilized weak form

$$\int_\Omega \omega \nabla \cdot \mathbf{u}^{n+1} \mathrm{d}\Omega + \int_\Omega \tau_{pspg} \nabla \omega \cdot \left(\frac{\mathbf{u}^{n+1} - \mathbf{u}^{n}}{\Delta t} + \left(\mathbf{u}^{n+1} \cdot \nabla \right) \mathbf{u}^{n+1} + \frac{\nabla p^{n+1}}{\rho} \right) \mathrm{d}\Omega = 0$$

Contribution of the pressure term to the first equation

$$A_{pp} = \int_{\Omega_k} \tau_{pspg} \frac{1}{\rho} \nabla \mathbf{N}^T \nabla \mathbf{N} \mathrm{d}\Omega_k$$

Discretization procedure for the continuity equation

Stabilized weak form

$$\int_\Omega \omega \nabla \cdot \mathbf{u}^{n+1} \mathrm{d}\Omega + \int_\Omega \tau_{pspg} \nabla \omega \cdot \left(\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \left(\mathbf{u}^{n+1} \cdot \nabla\right) \mathbf{u}^{n+1} + \frac{\nabla p^{n+1}}{\rho} \right) \mathrm{d}\Omega = 0$$

Contribution of the velocity term to the first equation

$$A_{pu_i} = \int_{\Omega_k} \left(\mathbf{N}^T \left(\nabla \mathbf{N}\right)_i + \tau_{pspg} \left(\nabla \mathbf{N}\right)_i^T \mathbf{u}^{n+1} \nabla \mathbf{N} \right) \mathrm{d}\Omega_k$$

Discretization procedure for the continuity equation

Stabilized weak form

$$\int_\Omega \omega \nabla \cdot \mathbf{u}^{n+1} \mathrm{d}\Omega + \int_\Omega \tau_{pspg} \nabla \omega \cdot \left(\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \left(\mathbf{u}^{n+1} \cdot \nabla \right) \mathbf{u}^{n+1} + \frac{\nabla p^{n+1}}{\rho} \right) \mathrm{d}\Omega = 0$$

Contribution of the time dependent term to the first equation

$$T_{pu_i} = \int_{\Omega_k} \tau_{pspg} \left(\nabla \mathbf{N} \right)_i^T \mathbf{N} \mathrm{d}\Omega_k$$

The language in UFEM

MeshTerm

- `MeshTerm` class provides access to quantities defined at each node or element
- Defines a Boost.Proto “terminal” (i.e. placeholder)
- Use as function: `u(xi)`: Evaluation at mapped coordinates `xi`.

MeshTerm

```
FieldVariable<0, VectorField> u("Velocity", "solution");
FieldVariable<1, ScalarField> p("Pressure", "solution");
FieldVariable<2, VectorField> u_adv("AdvVelocity", "advection");
```

More Notation

- A_e and T_e: `_A` and `_T`
- A_{pp}: `_A(p, p)`
- A_{pu_i}: `_A(p, u[_i])`
- $\mathbf{N} \equiv \mathbf{N}_u(\xi)$ for brevity
- Represented by the terminal `N`, used as a function: `N(u, xi)` $\equiv \mathbf{N}_u(\xi)$
- Gradient matrix $\nabla \mathbf{N}_u(\xi)$ omits $\mathbf{N}$ for brevity: `nabla(u, xi)`
- Index can be applied to gradient, i.e. get the *i*-th row: `nabla(u, xi)[_i]`
- `element_quadrature` as a terminal for the integration over the elements

User terminals

Add custom functions

```
inline Real min(const Real a, const Real b)
{
  return a < b ? a : b;
}
```

Create a terminal to use the function for evaluation

```
static boost::proto::terminal< double(*)(double, double) >::type const
    _min = {&min};
```

User terminals

Add custom functions

```
struct ComputeTau
{
  typedef void result_type;

  template<typename UT>
  void operator()(const UT& u, SUPGCoeffs& c) const
  {
    // Actual calculation omitted for brevity
  }
};
```

Create a terminal to use the struct for evaluation

```
static MakeSFOp<ComputeTau>::type const compute_tau = {};
```

Continuity equation in COOLFluiD UFEM

$$A_{pp} = \int_{\Omega_k} \tau_{pspg} \frac{1}{\rho} \nabla \mathbf{N}^T \nabla \mathbf{N} \mathrm{d}\Omega_k$$

```
_A(p, p) += c.tau_ps * transpose(nabla(p)) * nabla(p) * c.one_over_rho
```

$$A_{pu_i} = \int_{\Omega_k} \left(\mathbf{N}^T (\nabla \mathbf{N})_i + \tau_{pspg} (\nabla \mathbf{N})_i^T \mathbf{u}^{n+1} \nabla \mathbf{N} \right) \mathrm{d}\Omega_k$$

```
_A(p, u[_i]) += transpose(N(p)) * nabla(u)[_i] + c.tau_ps *
     transpose(nabla(p)     [_i]) * u_adv*nabla(u)
```

Continuity equation in COOLFluiD UFEM

```
elements_expression
(
 boost::mpl::vector2<LagrangeP1::Quad2D, LagrangeP1::Hexa3D>(),
 group
 (
  _A = _0, _T = _0,
  compute_tau(u, c),
  element_quadrature
  (
   _A(p, u[_i]) += transpose(N(p)) * nabla(u)[_i] + c.tau_ps *
     transpose(nabla(p)[_i]) * u_adv*nabla(u),
   _A(p, p) += c.tau_ps * transpose(nabla(p)) * nabla(p) *
     c.one_over_rho,
   _T(p , u[_i]) += c.tau_ps * transpose(nabla(p)[_i]) * N(u)
  ),
  system_matrix += invdt() * _T + _A,
  system_rhs += -_A * _x
 )
);
```

The Spalart-Allmaras turbulence model

Why to choose the Spalart-Allmaras turbulence model ?

- Still widely used
- Good properties
- As one equation model easy to implement
- Computationally cheap

Reattachment length ratio			
Re	Exp	k-ϵ	SA
3600	6.45	4.5	6.2
3000	7.6	5.3	7.23
2400	9.2	6.1	8.94

The transport equation for the turbulent viscosity

$$\frac{\partial \hat{\nu}}{\partial t} + u_j \frac{\partial \hat{\nu}}{\partial x_j} = c_{b1}(1 - f_{t2})\hat{S}\hat{\nu} - \left[c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2} \right] \left(\frac{\hat{\nu}}{d} \right)^2 + \frac{1}{\sigma} \left[\frac{\partial}{\partial x_j} \left((\nu + \hat{\nu}) \frac{\partial \nu}{\partial x_j} \right) + c_{b2} \frac{\partial \hat{\nu}}{\partial x_i} \right]$$

Spalart-Allmaras in COOLFluiD UFEM

```
_A(NU)+=transpose(N(NU))*u_adv*nabla(NU)+m_coeffs.tau_su*transpose(u_adv*nabla(NU))
*u_adv*nabla(NU)+cb1*transpose(N(NU))*N(NU)*((_norm(nabla(u) *nodal_values(u)-
transpose(nabla(u)*nodal_values(u))))+(NU/(kappa*kappa*d*d))*cb1*S_hat*NU_hat(1-
((NU/m_coeffs.mu)/(1+(NU/m_coeffs.mu)*((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*
(NU/m_coeffs.mu))/(cv1+((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*(NU/m_coeffs.mu)))))))
+cw1*((transpose(N(NU))*N(NU)*NU)/(d*d))*((_min(10((NU)/(kappa*kappa*d*d*((_norm
(nabla(u)*nodal_values(u)-transpose(nabla(u)*nodal_values(u))))+(NU/(kappa*kappa*d*d))
*(1-((NU/m_coeffs.mu)/(1+(NU/m_coeffs.mu)*((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*
(NU/m_coeffs.mu))/(cv1+((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*(NU/m_coeffs.mu)))))))
))))+cw2*(_pow((_min(10((NU)/(kappa*kappa*d*d*((_norm(nabla(u)*nodal_values(u) -
transpose(nabla(u)*nodal_values(u))))+(NU/(kappa*kappa*d*d))*(1((NU/m_coeffs.mu)/
(1+(NU/m_coeffs.mu)*((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*(NU/m_coeffs.mu))/(cv1+
((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*(NU/m_coeffs.mu)))))))))),6)-(_min(10((NU)/(kappa
*kappa*d*d*((_norm(nabla(u)*nodal_values(u)-transpose(nabla(u)*nodal_values(u))))
+(NU/(kappa*kappa*d*d))*(1-((NU/m_coeffs.mu)/(1+(NU/m_coeffs.mu)*((NU/m_coeffs.mu)*
(NU/m_coeffs.mu)*(NU/m_coeffs.mu))/(cv1+((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*
(NU/m_coeffs.mu))))))))))))*_pow(((1+_pow(cw3,6))/(_pow(((_min(10,((NU)/(kappa*kappa*
d*d*((_norm(nabla(u)*nodal_values(u)-transpose(nabla(u)*nodal_values(u))))+(NU/(kappa*
kappa*d*d))*(1-((NU/m_coeffs.mu)/(1+(NU/m_coeffs.mu)*((NU/m_coeffs.mu)*(NU/m_coeffs.mu)
*(NU/m_coeffs.mu))/(cv1+((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*(NU/m_coeffs.mu))))))))))
+cw2*(_pow((_min(10,((NU)/(kappa*kappa*d*d*((_norm(nabla(u)*nodal_values(u)-transpose
(nabla(u)*nodal_values(u))))
+(NU/(kappa*kappa*d*d))*(1-((NU/m_coeffs.mu)/(1+(NU/m_coeffs.mu)*((NU/m_coeffs.mu)*
(NU/m_coeffs.mu)*(NU/m_coeffs.mu))/(cv1+((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*(NU/
m_coeffs.mu))))))))))),6)-(_min(10((NU)/(kappa*kappa*d*d*((_norm(nabla(u)*
nodal_values(u)-transpose(nabla(u)*nodal_values(u))))+(NU/(kappa*kappa*d*d))(1
-((NU/m_coeffs.mu)/(1+(NU/m_coeffs.mu)*((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*(NU/
m_coeffs.mu))/(cv1+((NU/m_coeffs.mu)*(NU/m_coeffs.mu)*(NU/m_coeffs.mu))))))))))))
,6)+_pow(cw3,6))),1/6)-(1/sigma)*((NU+m_coeffs.mu)*transpose(nabla(NU))*nabla(NU))-
(1/sigma)*(cb2)*transpose(N(NU))*transpose(nabla(NU)*nodal_values(NU))*nabla(NU),
_T(NU,NU) +=  transpose(N(NU) + m_coeffs.tau_su * u_adv * nabla(NU)) * N(NU)
```

Section Outline

1 Introduction

2 COOLFluiD 3

3 Programming techniques

4 Application

5 Conclusions and future steps

Conclusions

+/- of DSEL from a beginner's point of view

- \+ more functionality with less code -> short code
- \+ less development time for a new functionality
- \+ eases implementation and leads to less mistakes
- \+ performance
- \- long time of development to create a DSEL
- \- debugging within the DSEL is costly

Future steps

What are the next steps ?

- making the implementation of Spalart-Allmaras work
- implement a stable coupling procedure
- perform simulations first with RANS, later with LES

References

- T. Banyai et. al. *A Fast Fully-Coupled Solution Algorithm For The Unsteady Incompressible Navier-Stokes Equations.* CMFF, 2006
- B. Janssens et. al. *Discretization of the Incompressible Navier-Stokes Equations using a Domain Specific Embedded Language.* 9th National Congress on Theoretical and Applied Mechanics, 2012
- D. Vandevoorde and N.M. Josuttis. *C++ Templates - The Complete Guide.* Addison-Wesley, 2003
- T. Verstraete. *Multidisciplinary Turbomachinery Component Optimization Considering Performance, Stress, and Internal Heat Transfer.* Universiteit Gent, PhD Thesis, 2008

Conjugate Heat Transfer of Cooling Channels using COOLFluiD 3

Sebastian Scholl

von Karman Institute for Fluid Dynamics

Doctoral Seminar, June 06, 2012