Implementation of the Dynamically Thickened Flame LES Model for Premixed and Non-Premixed Turbulent Combustion in OpenFOAM

Hafiz Ali Haider Sehole

Aalto University
School of Engineering

Aalto University

22 January 2025

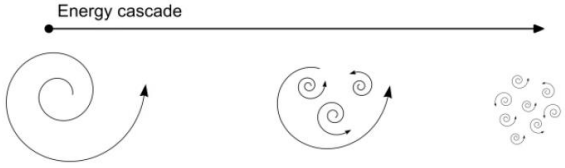
Contents

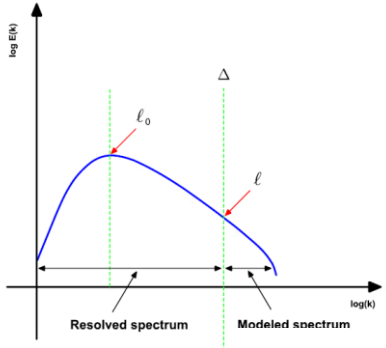
Introduction

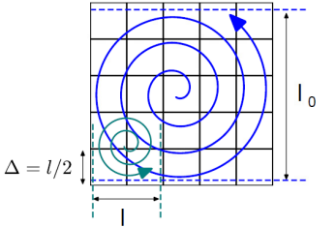
Turbulent Combustion

Aalto University
School of Engineering

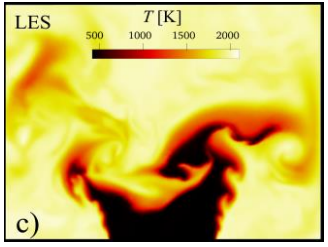
Introduction | Turbulent Combustion







- **Turbulence**
- **Chemical Kinetics**
- **Solution**
  - Sub-grid Turbulence Models
    - WALE, K-Equation
  - Sub-grid Combustion Models
    - PaSR, EDC
- **OpenFOAM Implementations**



TCI Model	Non-Premixed	Partially-Premixed	Premixed
Laminar	✓		
EDC	✓	✓	
PaSR		✓	✓
DTF	✓	✓	✓

Aalto University
School of Engineering

*images related to energy cascade and spectrum are taken from wolfdynamcis presentation

Theory

Governing Equation

- Diffusion
- Reaction Rate's

Governing Equation

- **Species Transport Equation**

$$\frac{\partial}{\partial t}(\rho Y_i) + \nabla \cdot (\rho \vec{v} Y_i) = -\nabla \cdot \vec{j_i} + R_i,$$

- **Diffusive Flux**

$$\vec{j_i} = -\rho D_{i,m} \nabla Y_i - D_{T,i} \frac{\nabla T}{T},$$

Mass diffusion

Thermal (Soret) diffusion

Unity Lewis Eddy Diffusivity

Thermal diffusion

$$\vec{j_i} = -\rho D_i \nabla Y_i \quad , \; D_i \propto D, \; D = \frac{\lambda}{\rho c_p}$$

Mass diffusion

$$L_{e_i} = \frac{\lambda}{\rho c_p D_i} = \frac{D}{D_i}$$

λ = Conductivity
ρ = density
c_p = Heat Capacity

- **Reaction Rate's**

R_i is reaction rate source term

Laminar (no TCI)

- **Assumption**
  - Perfectly mixed reactor
  - no sub-grid turbulence-chemistry
- **Direct use of Reaction Rates**

$$\overline{R_i} = R_i$$

Partially Stirred Reactor (PaSR)

$$R_i = \kappa \times R_i(Y_i),$$

κ is reacting volume fraction

Where,

$$\kappa = \frac{\tau_c}{\tau_c + \tau_k},$$

τ_c is chemical time scale
τ_k is mixing time scale

τ_k is defined as

$$\tau_k = C_{mix}\sqrt{\frac{\nu_{\text{eff}}}{\epsilon}},$$

τ_c is defined as

$$\tau_c = \frac{c_{\text{tot}}}{\sum_{j=1}^{n_R}\sum_{i=1}^{N_{s,\text{RHS}}} \nu_{i,j} k_{f,j}},$$

c_{tot} is total sum of species concentration
n_R is number of reactions
N_s is number of species
$v_{i,j}$ is the stoichiometric co-efficient
$k_{f,j}$is the forwards reaction rate

$$\kappa = \begin{cases} 1.0 & \text{if } \tau_k < \text{small}, \\ \frac{\tau_c}{\tau_c + \tau_k} & \text{otherwise}, \end{cases}$$

If the value of κ is very small, almost zero, then it will take 1, otherwise, it calculate the κ.

Dynamic Thickened Flame (DTF)

$$\frac{\partial(\rho Y_i)}{\partial t} + \nabla \cdot (\rho \vec{v} Y_i) = \nabla \cdot (\rho D_{iF} \nabla Y_i) - R_{iF}$$

$$D_{iF} \rightarrow DF, \quad R_{iF} \rightarrow \frac{1}{F} R_i$$

$$D_{iEF} \rightarrow EFD, \quad R_{iEF} \rightarrow \frac{E}{F} R_i,$$

1. Flame Sensor Ω *(S)*
   - Track the flame
2. Thickening Factor *(F)*
   - The thickening factor makes flame thicker
3. Efficiency Function *(E)*
   - Adjustment to the thickening Factor, to consider unresolved flame on the grid

Dynamic Thickened Flame (DTF)

$$\frac{\partial(\rho Y_i)}{\partial t} + \nabla \cdot (\rho \vec{v} Y_i) = \nabla \cdot (\rho D_{iF} \nabla Y_i) - R_{iF}$$

1. Flame Sensor

$$D_{iF} \rightarrow DF, \quad R_{iF} \rightarrow \frac{1}{F} R_i$$

$$F = 1 + (min(F_{\max}, F_s) - 1)\Omega,$$

Heat Release

$$\Omega = tanh(\beta \frac{h}{h_{max}}),$$

Hyperbolic Tangent

Maximum Heat Release

Dynamic Thickened Flame (DTF)

$$\frac{\partial(\rho Y_i)}{\partial t} + \nabla \cdot (\rho \vec{v} Y_i) = \nabla \cdot (\rho D_{iF} \nabla Y_i) - R_{iF}$$

2. Thickening Factor

$$D_{iF} \rightarrow DF, \quad R_{iF} \rightarrow \frac{1}{F} R_i$$

$$F = 1 + (min(F_{\max}, F_s) - 1)\Omega,$$

Thermal Diffusivity

$$D = \frac{\lambda}{\rho c_p}$$

λ = Conductivity
ρ = density
c_p = Heat Capacity

Premixed Laminar Flame Speed

$$S_l^0 = \sqrt{DR}$$

D = Thermal Diffusivity
R = Reaction Rate

Laminar Flame Thickness

$$\delta_l = \frac{D}{S_l^0}$$

Thickening Factor (F)

$$F_s = \frac{\Delta N}{\delta_l}$$

$$\Delta = V^{\frac{1}{N_d}}$$

Dynamic Thickened Flame (DTF)

$$\frac{\partial(\rho Y_i)}{\partial t} + \nabla \cdot (\rho \vec{v} Y_i) = \nabla \cdot (\rho D_{iF} \nabla Y_i) - R_{iF}$$

3. Efficiency Function

$$D_{iF} \rightarrow DF, \quad R_{iF} \rightarrow \frac{1}{F} R_i$$

$$D_{iEF} \rightarrow EFD, \quad R_{iEF} \rightarrow \frac{E}{F} R_i,$$

$$E = \frac{\Xi|_{\delta_l = \delta_l^0}}{\Xi|_{\delta_l = \delta_l^1}} \geq 1.$$

$$S_l^0 = \sqrt{DR} \qquad \delta_l = \frac{D}{S_l^0}$$

Turbulent Flame Thickness

$$\delta_l^1 = F \delta_l^0$$

Wrinkling Factor

$$\Xi = 1 + \alpha \frac{u'_{\Delta e}}{S_l^0} \Gamma\left(\frac{\Delta_e}{\delta_l}, \frac{u'_{\Delta e}}{S_l^0}\right),$$

Stretch Function

$$\Gamma\left(\frac{\Delta_e}{\delta_l}, \frac{u'_{\Delta e}}{S_l^0}\right) \approx 0.75 \exp\left[-1.2 \left(\frac{u'_{\Delta e}}{S_l^0}\right)^{-0.3}\right] \left(\frac{\Delta_e}{\delta_l}\right)^{2/3}$$

PaSR Implementation

OpenFOAM

Partially Stirred Reactor (PaSR)

PaSR:correct()

Laminar::correct()

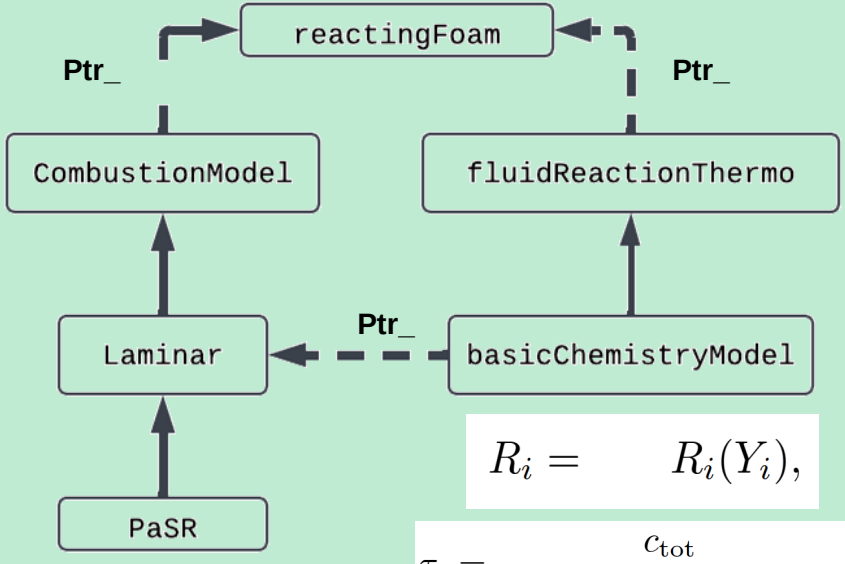
$$\kappa = \frac{\tau_c}{\tau_c + \tau_k},$$

$$\tau_k = C_{mix}\sqrt{\frac{\nu_{\text{eff}}}{\epsilon}},$$

PaSR::R(volScalarField& Y)

$$R_i = \kappa \times R_i(Y_i),$$

PaSR::Qdot()



Partially Stirred Reactor (PaSR)

Listing 3.3: PaSR.C correct()

```
1  void Foam::combustionModels::PaSR::correct()
2  {
3      laminar::correct();
4
5      tmp<volScalarField> tepsilon(this->turbulence().epsilon());
6      const scalarField& epsilon = tepsilon();
7
8      tmp<volScalarField> tnuEff(this->turbulence().nuEff());
9      const scalarField& nuEff = tnuEff();
10
11     tmp<volScalarField> ttc(this->chemistryPtr_->tc());
12     const scalarField& tc = ttc();
13
14     forAll(epsilon, i)
15     {
16         const scalar tk =
17             Cmix_*sqrt(max(nuEff[i]/(epsilon[i] + small), 0));
18
19         if (tk > small)
20         {
21             kappa_[i] = tc[i]/(tc[i] + tk);
22         }
23         else
24         {
25             kappa_[i] = 1.0;
26         }
27     }
28 }
```

Laminar Correct

$$\tau_c = \frac{c_{\text{tot}}}{\sum_{j=1}^{n_R} \sum_{i=1}^{N_{s,\text{RHS}}} \nu_{i,j} k_{f,j}},$$

$$\tau_k = C_{mix} \sqrt{\frac{\nu_{\text{eff}}}{\epsilon}},$$

$$\kappa = \frac{\tau_c}{\tau_c + \tau_k},$$

$$\kappa = \begin{cases} 1.0 & \text{if } \tau_k < \text{small}, \\ \frac{\tau_c}{\tau_c + \tau_k} & \text{otherwise}, \end{cases}$$

Aalto Scho

Partially Stirred Reactor (PaSR)

Listing 3.4: PaSR R(volScalarField& Y)

```
1 Foam::tmp<Foam::fvScalarMatrix>
2 Foam::combustionModels::PaSR::R(volScalarField& Y) const
3 {
4     return kappa_*laminar::R(Y);
5 }
```

$$R_i = \kappa \times R_i(Y_i),$$

DTF Implementation

Dynamic Thickened Flame (DTF)

$$D_{iEF} \rightarrow EFD, \quad R_{iEF} \rightarrow \frac{E}{F} R_i,$$

DTF:correct()

Laminar::correct()
flameSpeed() (SL deltaL)

$$\Omega = tanh(\beta \frac{h}{h_{max}}),$$

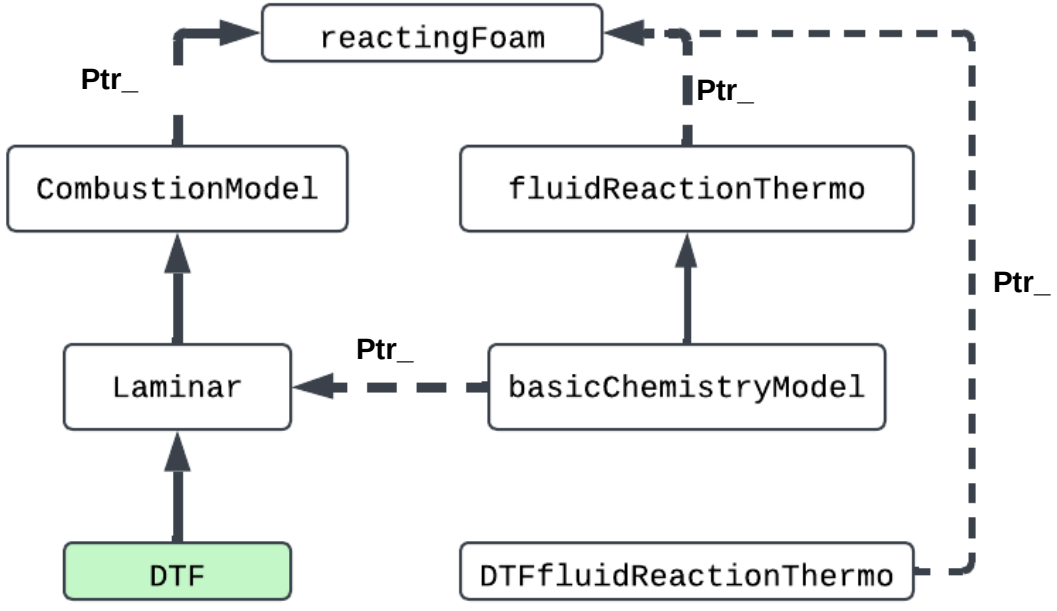
$$F = 1 + (min(F_{\max}, F_s) - 1)\Omega,$$

$$E = \frac{\Xi|_{\delta_l = \delta_l^0}}{\Xi|_{\delta_l = \delta_l^1}} \geq 1$$

DTF::R(volScalarField& Y)

$$R_{iEF} \rightarrow \frac{E}{F} R_i,$$

DTF::Qdot()



Dynamic Thickened Flame (DTF)

Parameters	TFM	DTF
Mesh	Uniform	Non-uniform (Tetra, Poly, Hexa)
Physics	Laminar/DNS	Laminar/DNS/LES
Laminar Flame Speed (S_l^0)	User-defined (Fixed Value)	Computed (update/iteration)
Laminar Flame Thickness (δ_l)	User-defined (Fixed Value)	Computed (update/iteration)
Thickening Factor (F)	User-defined (Fixed Value)	Computed (update/iteration)
Efficiency Factor (E)	Power-law	Colin
Flame Sensor (S)	Based on heat release	Optimized `TFM` sensor*
ThermophysicalTransportModels	Modified Library	Same implementation as `TFM`
Flame Speed (S_l^0 & δ_l)	Not implemented	Implemented

* *Optimized `TFM` sensor: The same sensor as in `TFM` is used, but filters are applied to smooth the values. The magnitude of the sensor is also taken to ensure positive values.*

DTF: Flame Sensor

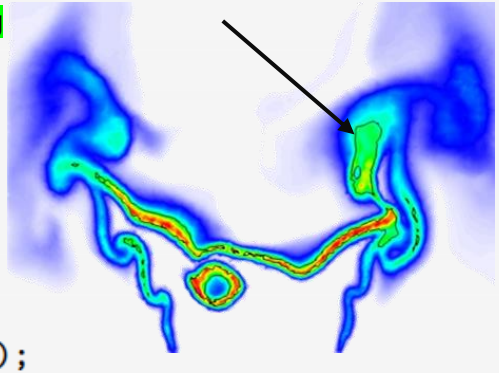
$$F = 1 + (min(F_{\max}, F_s) - 1)\Omega,$$

```
tmp<volScalarField> tqdot = mesh_.lookupObject<combustionModel>(modelName).Qdot();
////////////////////////////////simpleFilter////////////////////////////////
    for(int i = 0; i < n_filters_; i++) {
      tqdot = sFilter_(tqdot);
     }
volScalarField qdot = Foam::mag(tqdot.ref());
Info << "qdot abs " << endl;

forAll(qdot, cellI)
{
    qdot[cellI] = max(qdot[cellI], 100);
}

dimensionedScalar maxValue(qdot.dimensions(),gMax(qdot));
Info << "qdot maximum: "<<maxValue << "Minimum : "<< gMin(qdot)<< endl;


S_ = tanh(beta_*qdot/maxValue);
```

Filter

Foam::mag

Flame Sensor , value = 0.99



$$\Omega = tanh(\beta \frac{h}{h_{max}}),$$

DTF: flameSpeed() (1 of 5)

Reaction Rate

$$S_l^0 = \sqrt{DR}$$

```
// Reset reaction rate for all cells to zero
forAll(tRR_.internalField(), celli)
{
    tRR_[celli] = 0.0;
}

// Calculate the reaction rate for all reactions and species
for (label ri = 0; ri < nReaction; ++ri)  Loop over number of reactions
{
    for (label si = 0; si < nSpecie; ++si) Loop over number of species
    {
        volScalarField::Internal RR = Foam::mag(chemistryPtr_->calculateRR(ri, si));
        forAll(RR, celli)
        {                                                    Pointer to , calculateRR
            if (!std::isnan(F_[celli]) && F_[celli] > SMALL)
            {
                tRR_[celli] += RR[celli];
            }
        }
    }
}
```

DTF: flameSpeed() (2 of 5)

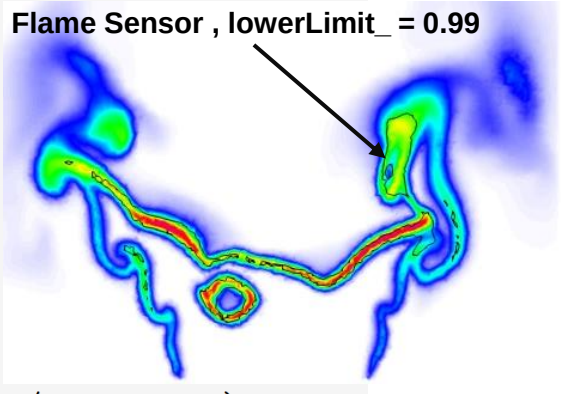
Reaction Rate

flameSensor

$$S_l^0 = \sqrt{DR}$$

```
volScalarField q_sensor = flameSensor_->S();
scalar sumReactionRate = 0.0;
scalar count = 0.0;
Info << "Internal field size: " << q_sensor.internalField().size() << endl;

// Extract reaction rates between the set values of the flame sensor
forAll(q_sensor.internalField(), celli)
{
    if (q_sensor[celli] >= lowerLimit_ && q_sensor[celli] <= upperLimit_)
    {
        sumReactionRate += tRR_[celli];
        count++;
    }
}

// Parallel reduction for sum and count
scalar globalSumReactionRate = sumReactionRate;
scalar globalCount = count;
reduce(globalSumReactionRate, sumOp<scalar>());
reduce(globalCount, sumOp<scalar>());

// Compute average reaction rate across all processors
scalar avgReactionRate = (globalCount > 0) ? (globalSumReactionRate / globalCount) : 0.0;

Info << "Average reaction rate (based on sensor): " << avgReactionRate << endl;
```

User inputs



DTF: flameSpeed() (3 of 5)

Thermal Diffusivity

$$D = \frac{\lambda}{\rho c_p}$$

```
const volScalarField& rhoField = Foam::combustionModel::thermo_.rho();
const volScalarField& kappaField = Foam::combustionModel::thermo_.kappa();
const volScalarField& CpField = Foam::combustionModel::thermo_.Cp();

forAll(this->mesh().V(), celli)
{
    scalar rho = rhoField[celli];
    scalar lambda = kappaField[celli];
    scalar cp = CpField[celli];

    if (rho > SMALL && cp > SMALL)
    {
        D_[celli] = lambda / (rho * cp);
    }
    else
    {
        D_[celli] = 0;
    }
}
Info << "Maximum Diffusion, D_: " << Foam::gMax(D_) << endl;
Info << "Minimum Diffusion, D_: " << Foam::gMin(D_) << endl;

// Smoothing filter
volScalarField smoothed_D = D_;
fvc::smooth(smoothed_D, filters_);
D_ = smoothed_D;
Info << "Smoothed Diffusion: Maximum: " << Foam::gMax(D_) << ", Minimum: " << Foam::gMin(D_) <<
 endl;
```

A?

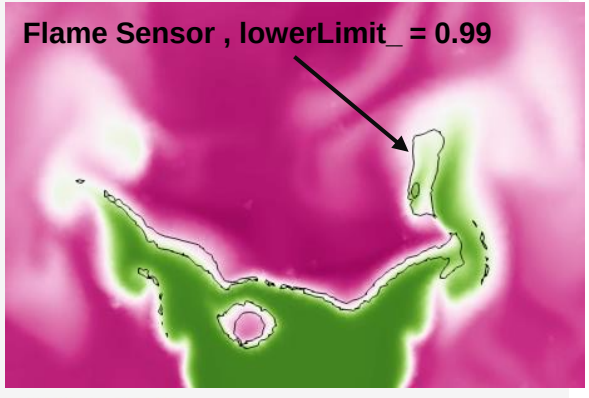
DTF: flameSpeed() (4 of 5)

Thermal Diffusivity

$$D = \frac{\lambda}{\rho c_p}$$

```
scalar sumD = 0.0;
scalar countD = 0.0;
//Extracting Values of diffusion between the set values of flame sensor
forAll(q_sensor.internalField(), celli)
{
    if (q_sensor[celli] >= lowerLimit_ && q_sensor[celli] <= upperLimit_)
    {
        sumD += D_[celli];
        countD++;
    }
}

// Parallel reduction for sumD and countD
scalar globalSumD = sumD;
scalar globalCountD = countD;
reduce(globalSumD, sumOp<scalar>());
reduce(globalCountD, sumOp<scalar>());

// Compute average D across all processors
scalar avgD = (globalCountD > 0) ? (globalSumD / globalCountD) : 0.0;

Info << "Average Diffusion (based on sensor): " << avgD << endl;
```

User inputs



DTF: flameSpeed() (5 of 5)

SL and DeltaL

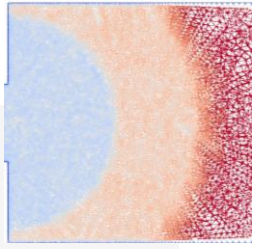
```
    ////////////////////// SL_ and deltaL_ Calculation ////////////////////////
    Info << "Starting Lamianr flame speed (SL_) and Laminar flame thickness (deltaL_) calculation
..." << endl;

        if ( avgReactionRate > SMALL && avgD > SMALL )
        {
            SL_ = sqrt(avgReactionRate * avgD);
            Info << "Laminar Flame Speed: " << SL_ << endl;

            deltaL_ = avgD /SL_;

            Info << "Laminar Flame thickness: " << SL_ << endl;

            if (SL_ > 50)
            {
                Info <<  " SL_ value " << SL_ << " exceeds 50, clamping to 50." << endl;
                SL_ = 50;
            }

            if (deltaL_ > 10)
            {
                Info << "deltaL_ value " << deltaL_ << " exceeds 10, clamping to 10." << endl;
                deltaL_ = 10;
            }
        }
        else
        {
            SL_ = this->coeffs().template lookup<scalar>("SL");
            deltaL_ = this->coeffs().template lookup<scalar>("deltaL");
            }
        Info << "Completed Lamianr flame speed  (SL_) and Laminar flame thickness (deltaL_)
    calculation." << endl;
}
```

$S_l^0 = \sqrt{DR}$

$\delta_l = \frac{D}{S_l^0}$

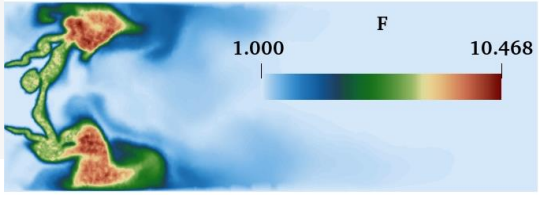
DTF: Flame Thickness (F)

```
forAll(F_, celli)
{
    scalar delta_g = pow(mesh_.V()[celli], 1.0 / static_cast<scalar>(mesh_.nGeometricD()));
    if (deltaL_ > SMALL)
    {
        scalar Fvalue = (delta_g * this->coeffs().template lookup<scalar>("N")) / deltaL_;
        Fs_[celli] = min(Fvalue, Fmax_);
        scalar sensorValue = flameSensor_->S()[celli];
        F_[celli] = 1 + (Fs_[celli] - 1) * sensorValue;

        if (F_[celli] < 1) F_[celli] = 1;
    }
    else
    {
        F_[celli] = 1.0;
    }
}
```



$$\Delta = V^{\frac{1}{N_d}}$$

$$F = 1 + (min(F_{\max}, F_s) - 1)\Omega$$



DTF: Efficiency Function (E)

$$\Xi = 1 + \alpha \frac{u'_{\Delta e}}{S_l^0} \Gamma\left(\frac{\Delta_e}{\delta_l}, \frac{u'_{\Delta e}}{S_l^0}\right),$$

$$\Gamma\left(\frac{\Delta_e}{\delta_l}, \frac{u'_{\Delta e}}{S_l^0}\right) \approx 0.75 \exp\left[-1.2 \left(\frac{u'_{\Delta e}}{S_l^0}\right)^{-0.3}\right] \left(\frac{\Delta_e}{\delta_l}\right)^{2/3}$$

```
  scalar numerator = 1.0 + alpha_ * 0.75 * exp(-1.2 / pow(up_SL_r, 0.3)) * pow(delta_r, 2.0 /
.0) * up_SL_r;
  scalar denominator = 1.0 + alpha_ * 0.75 * exp(-1.2 / pow(up_SL_r, 0.3)) * pow(delta_R, 2.0 /
.0) * up_SL_r;

  if (denominator <= 0) {
      WarningInFunction << "Invalid denominator at cell " << celli << endl;
      E_[celli] = 1.0; // Assign minimum efficiency to avoid division by zero
      continue;
  }

  scalar efficiency = numerator / denominator;

  // Enforce the condition E >= 1
  E_[celli] = max(efficiency, 1.0);
```

$$E = \frac{\Xi|_{\delta_l = \delta_l^0}}{\Xi|_{\delta_l = \delta_l^1}} \geq 1$$

DTF: thermoPhysicalTransport

```
    virtual tmp<volScalarField> DEff(const volScalarField& Yi) const
    {
    const volScalarField& EF = mesh_.lookupObject<volScalarField>("EF"); // look for EF value
        return volScalarField::New
        (
            "DEff",
            this->thermo().kappa()/this->thermo().Cp()*EF + this->alphat()  //EF multiplied to
scale the diffusivity term
        );
    }
```

DTF Tutorial

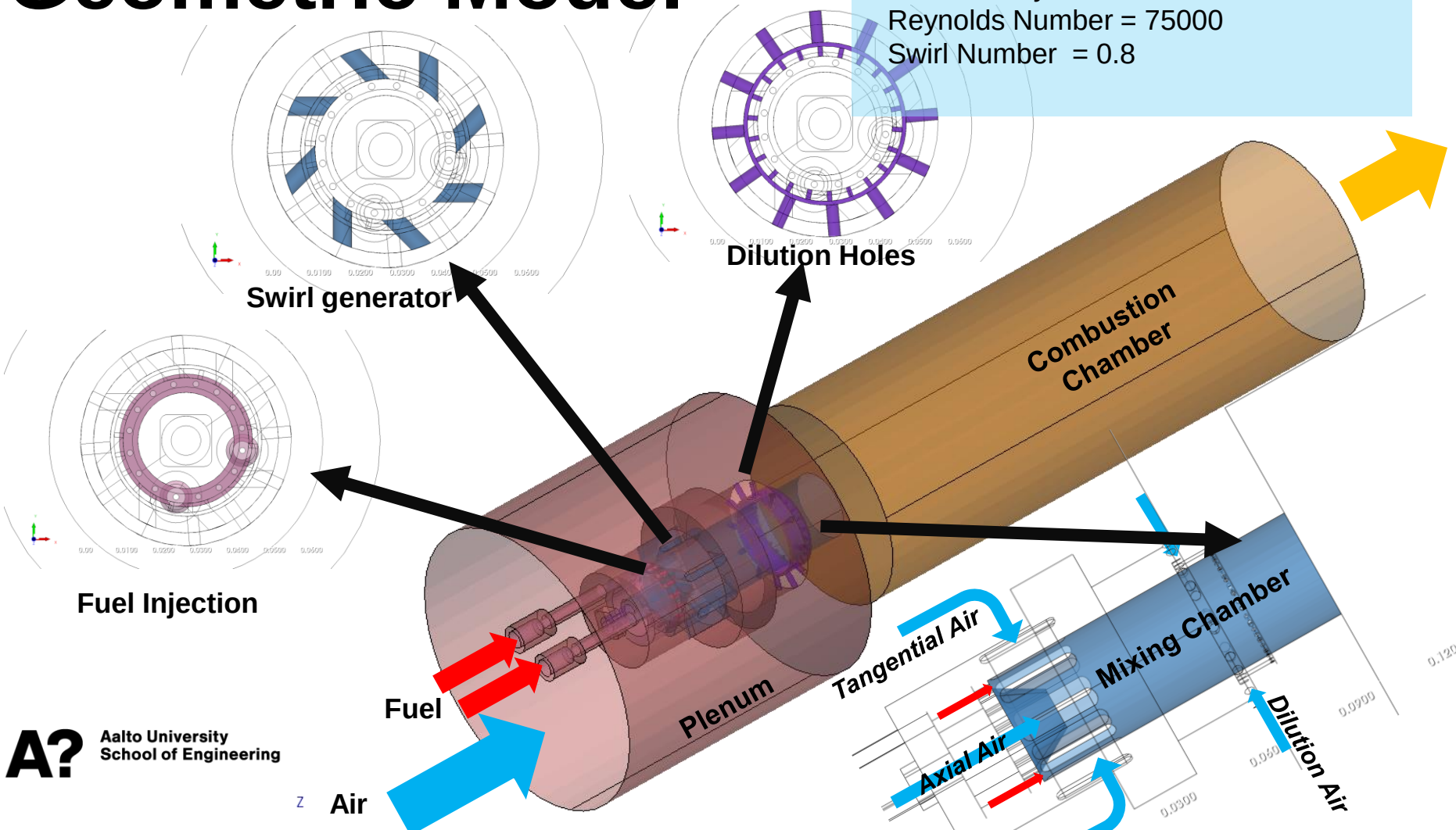
Aalto University
School of Engineering

Tutorial

- **Case 01: AHEAD Premixed Burner**
  - Chapter 05; Results → Validation
- **Case 02: Test Case**
  - Appendix B; Contain all the files require to run a test case

Geometric Model

Flow Conditions;
Bulk Velocity = 75 m/s
Reynolds Number = 75000
Swirl Number = 0.8



Case Setup

- **PaSR**

constant/combustionProperties

```
combustionModel PaSR;

PaSRCoeffs
{
    Cmix 1.0;
}
```

- **DTF**

constant/combustionProperties

```
combustionModel  DTF;

DTFCoeffs
{
    //efficiencyFunction none;
    efficiencyFunction colin;

    colinCoeffs
    {
    alpha 0.02;
    n_filters 10;  // Number of filtering operations
    }
    Fmax          15;           // Maximum thickening factor
    N             3;            // Control parameter for flame thickening
    Prt       0.85;
    FlameSpeed    true;
    lowerLimit    0.9999; //flame sensor Upper value
    upperLimit    1; //flame sensor lower value
    filters       2;
    //flameSensor none; //tanh; //
    flameSensor tanh;
    tanhCoeffs {
    beta 10;
    n_filters 10;
    }
}
```

Case Setup

- **PaSR**

  constant/thermophysicalTransport

- **DTF**

  constant/thermophysicalTransport

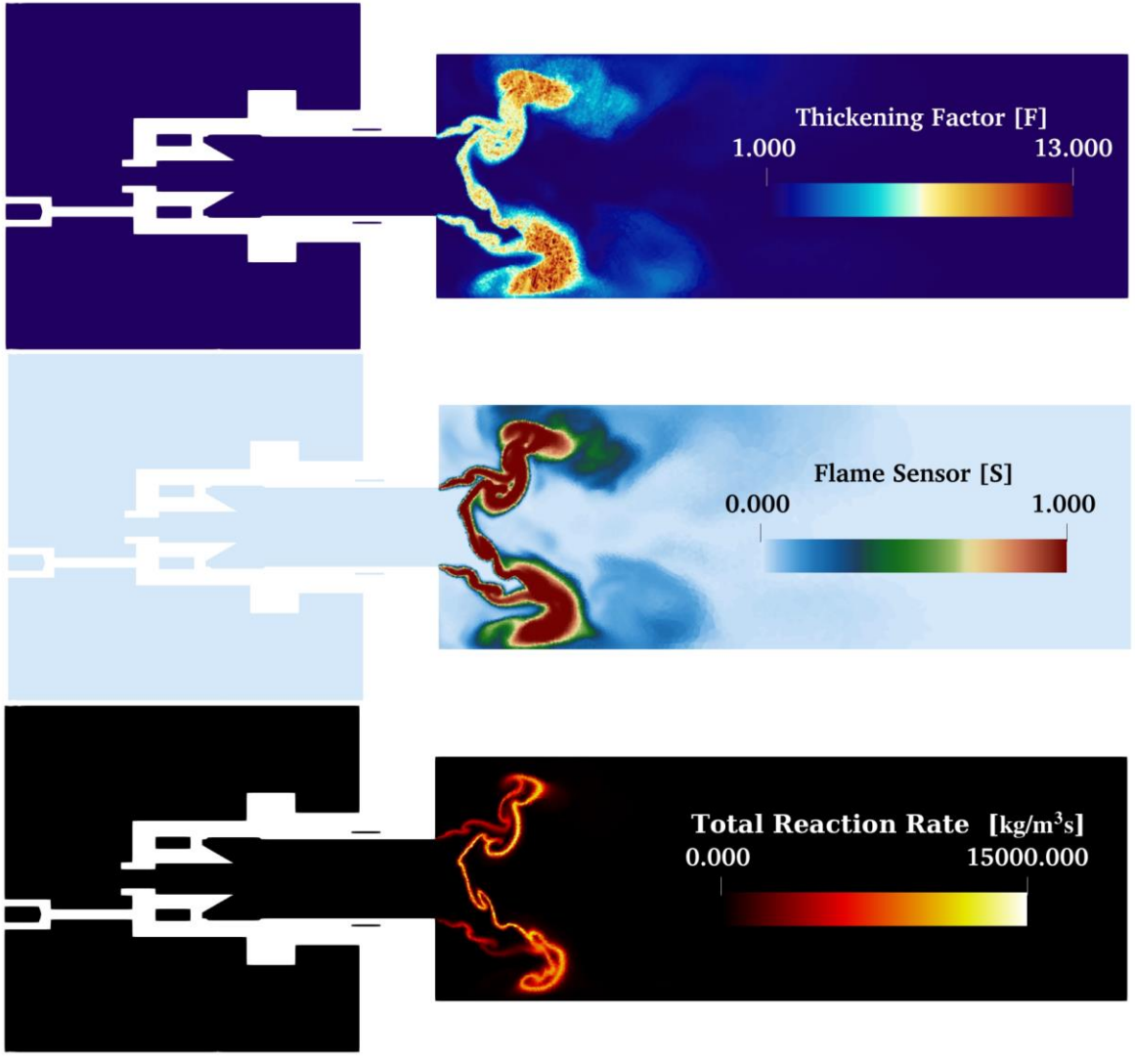
```
   LES
   {
       model           DTFunityLewisEddyDiffusivity;
       Prt             0.85;
   }
```

system/controlDict

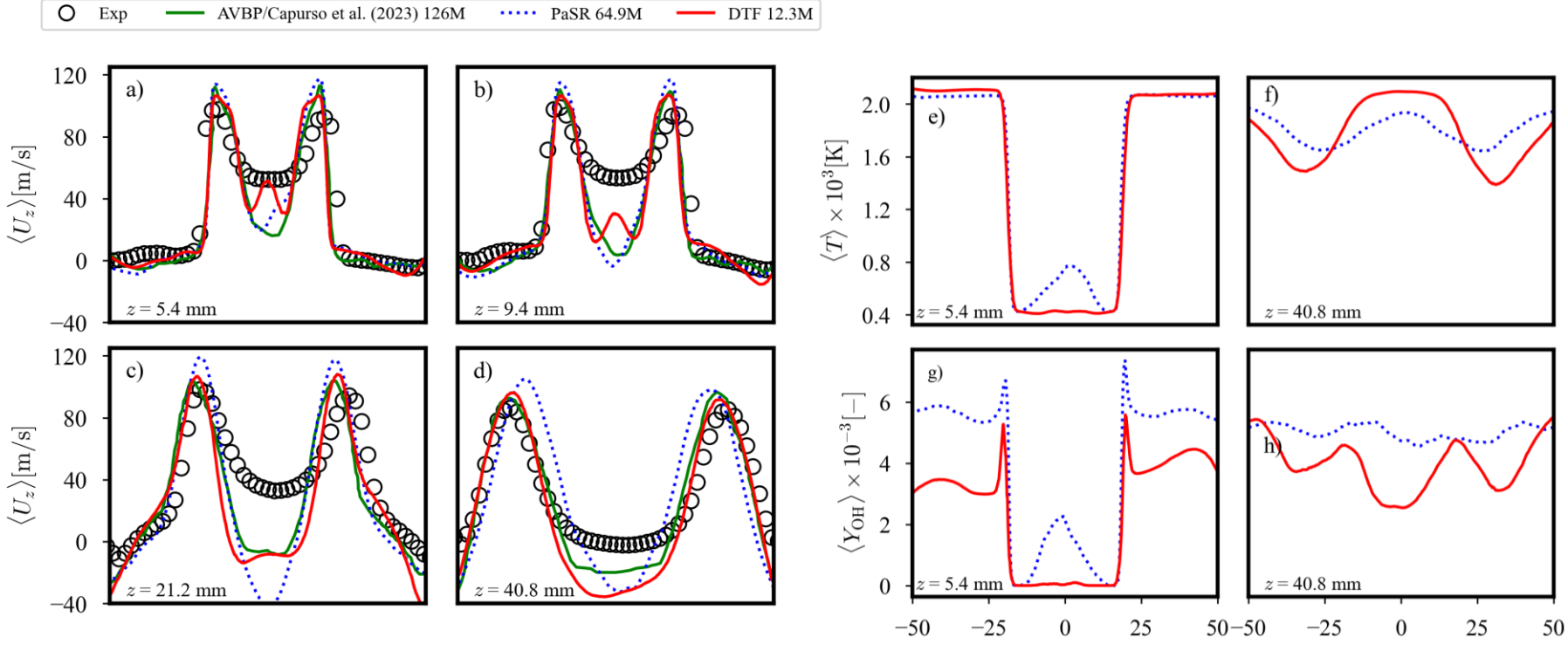
```
libs
(
    "libDTFcombustion.so"
    "libDTFtransport.so"
);
```

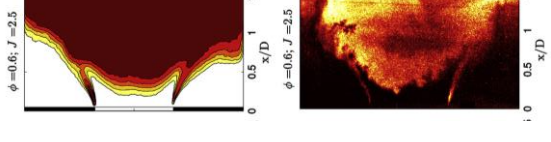
Results

Animation



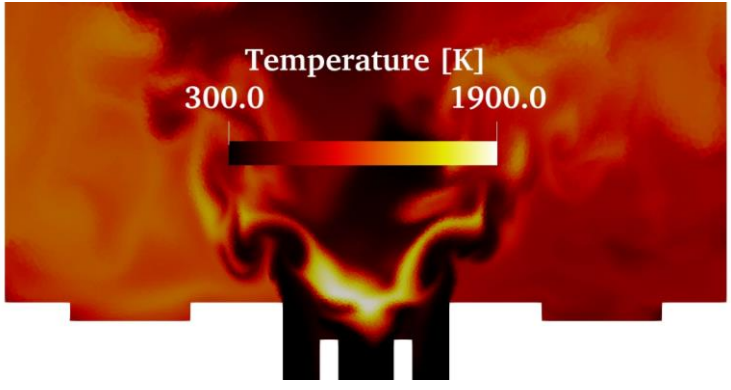
Results



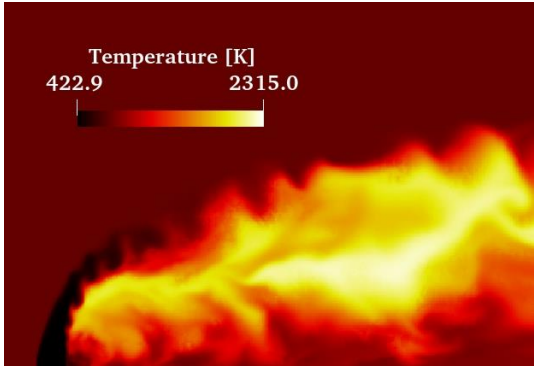


Aalto University
School of Engineering

Work in-progress



Flame L
HYLON



Jet-in-Cross Flow

Thank you