

A short introduction to OpenFOAM

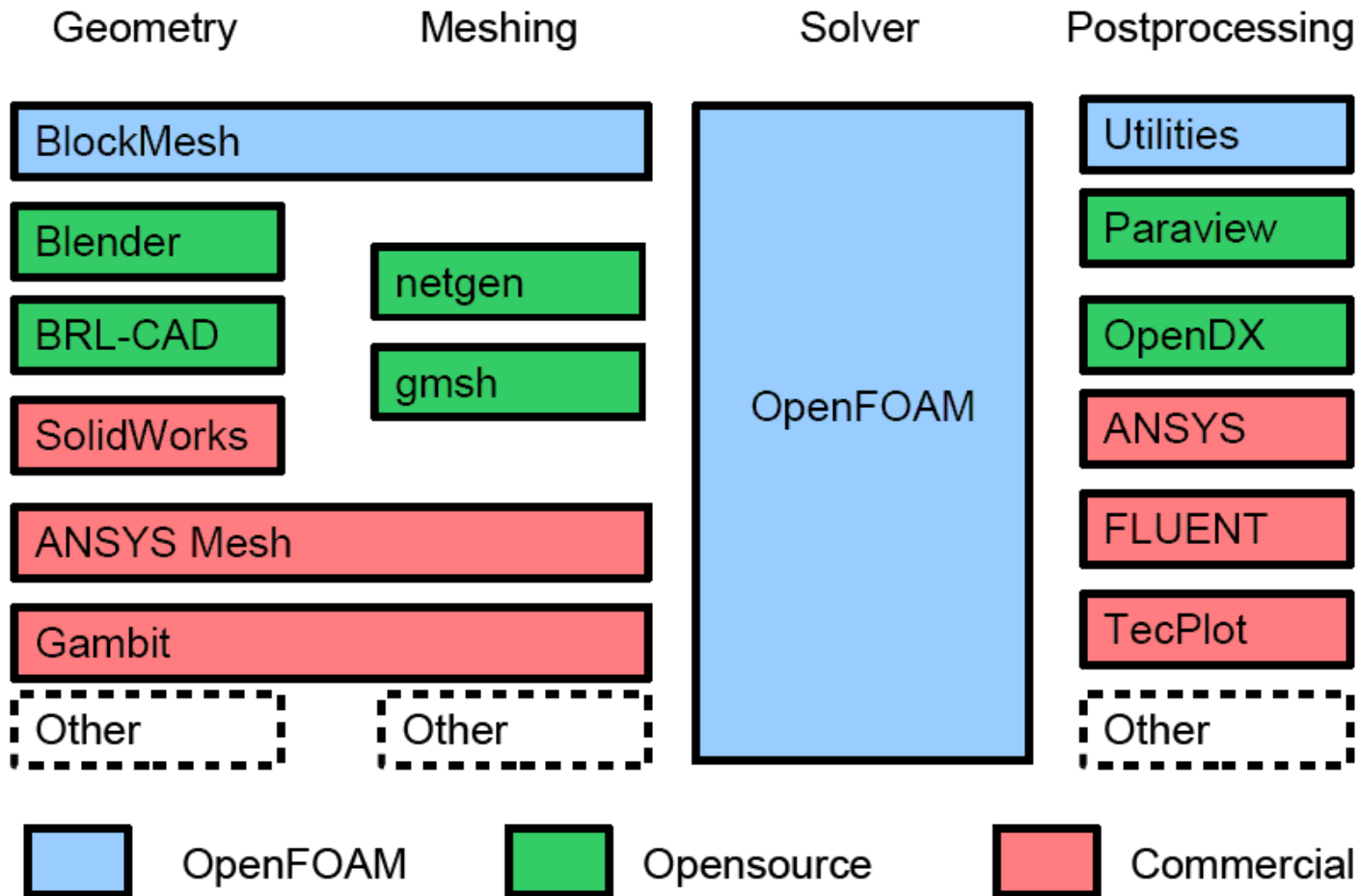
Sergei Strijhak (ISPRAS, Moscow, Russia)

27.06.2016-28.06.2016

Contents

- I part : Introduction to OpenFoam
- 1 Lab (3D cavity)
- II part: FVM method
- 2 Lab (pitzDaily)
- III part : Programming in OpenFoam
- 3 Lab (hotRoom, damBreak)
- IV part: Look in source code
- Literature

Main steps and modules in CFD





OpenFOAM — Open Source Software.

OpenFOAM — written on C++ using object-oriented techniques.

OpenFOAM – A tensorial approach to computational continuum mechanics.

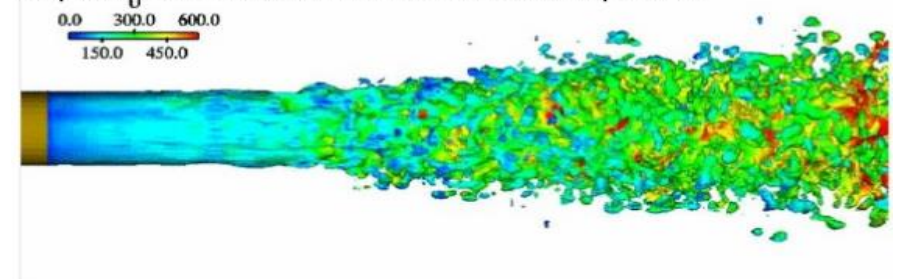
OpenFOAM – Object-orientation techniques enable the creation of data types that closely mimic those of continuum mechanics, and the operator overloading possible in C++ allows normal mathematical symbols to be used for the basic operations.

OpenFOAM – software which is used in industry and education

Salome – CAD/CAE Platform.
Pre-processor.

Paraview – Post-processor.

Впрыск дизельного топлива в цилиндр, LES





OpenFoam. History.



Developed in Imperial College of Science, London, UK. 1991-2003

**Main developers: Ph.D. students (H. Weller and H. Jasak) of Imperial College, London,
Research Supervisor: prof. A.D. Gosman.**

Open Foam became Open Source code in 2004 on GPL license

Literature:

- Weller, H.G.; Tabor G.; Jasak, H. and Fureby, C.: [A Tensorial Approach to CFD using Object Orientated Techniques](#), Computers in Physics, 1998 v. 12 n. 6, pp 620 – 631
- Ferziger J.H., Peric M., Computational Methods for Fluid Dynamics. Springer-Verlag, Berlin et al.: Springer, 2002. – 423p
- B. Stroustrup, The C++ Programming Language, 3rd ed.

Main version: OpenCFD, UK - www.openfoam.org

Open Source Conferences with OpenCFD, Ltd – 2007,2008, 2009

OpenFoam Workshop – 2008-2015

OpenFOAM Summer School in Zagreb for two weeks

Hierarchy of solvers in OpenFOAM

`$FOAM_APP/solvers`

DNS

basic

combustion

compressible

discreteMethods

electromagnetics

financial

heatTransfer

incompressible

Incompressible flows

lagrangian

multiphase

stressAnalysis

Multiphase flows

Hierarchy of means in OpenFOAM

An advantage of OpenFOAM — flexible orientation on the user:

Advanced User — tools for development of new solvers and physical models using C++ and OpenFOAM classes.

Engineers — a scope of ready solvers (~80) and utilities (~170) for computational continuum mechanics

OPEN FOAM

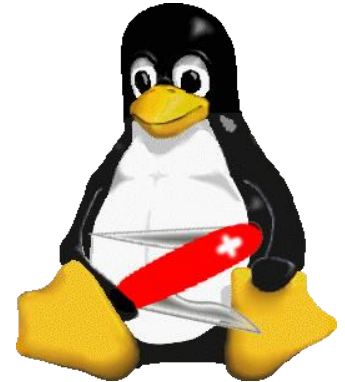
Finite Volume Method

Different Utilities

Different Solvers

Structure of OpenFOAM

- * applications: source codes of solvers
 - Solvers
 - Utilities
 - Bin
 - Test
- * bin: binary files
- * doc: pdf и Doxygen файлы
 - Doxygen
 - Guides-a4
- * lib: libraries
- * src – source codes
- * tutorials – tutorials
- * Wmakes – utility for compilation



Main conservation laws

- Mass, momentum, scalar and volume equations in integral form:

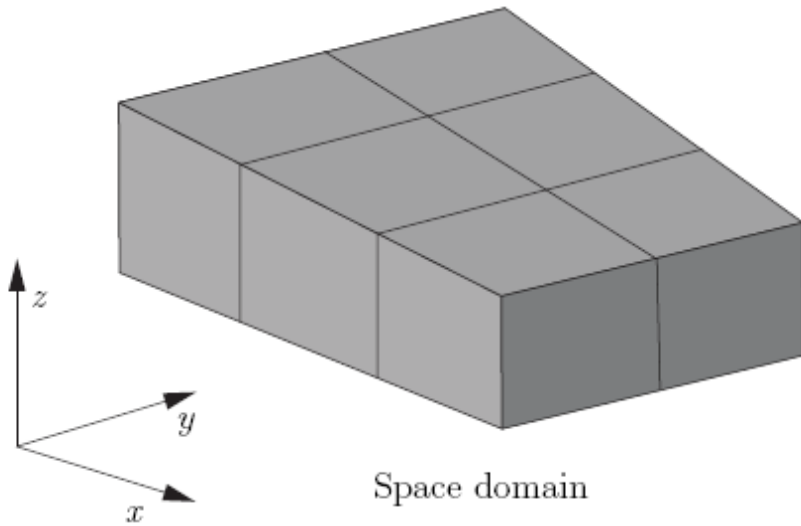
$$\frac{d}{dt} \int_V \rho dV + \int_S \rho(\mathbf{v} - \mathbf{v}_b) \cdot \mathbf{n} dS = 0$$

$$\frac{d}{dt} \int_V \rho \mathbf{v} dV + \int_S \rho \mathbf{v}(\mathbf{v} - \mathbf{v}_b) \cdot \mathbf{n} dS = \int_S (\mathbf{T} - p\mathbf{I}) \cdot \mathbf{n} dS + \int_V \rho \mathbf{b} dV$$

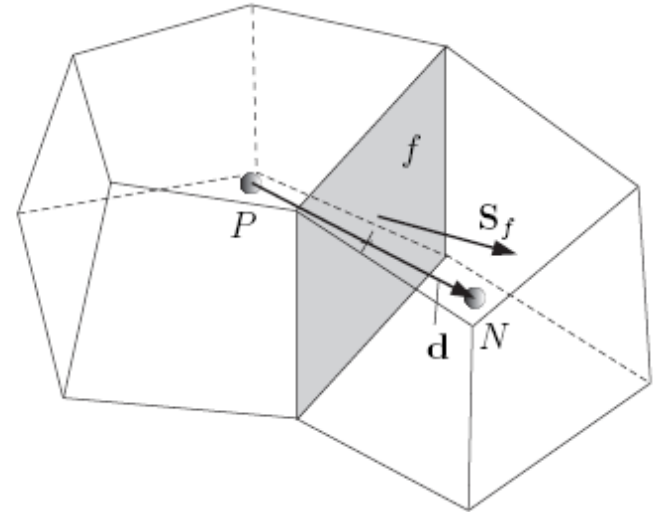
$$\frac{d}{dt} \int_V \rho \phi dV + \int_S \rho \phi(\mathbf{v} - \mathbf{v}_b) \cdot \mathbf{n} dS = \int_S \Gamma \nabla \phi \cdot \mathbf{n} dS + \int_V \rho b_\phi dV$$

$$\frac{d}{dt} \int_V dV - \int_S \mathbf{v}_b \cdot \mathbf{n} dS = 0$$

Finite Volume Method in OpenFOAM



Domain discretization in time and
3 directions



Two cells and main their values (cell,
Centroid , Face, Normal)

Supported cells in OpenFOAM

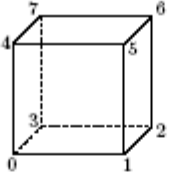
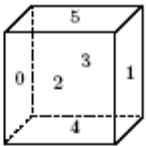
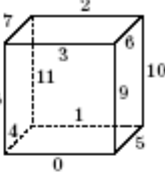
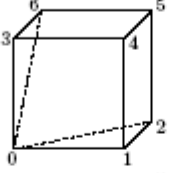
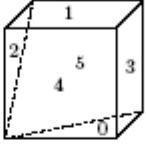
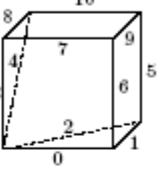
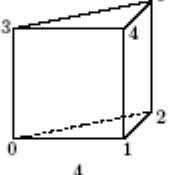
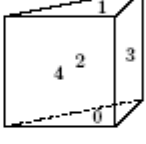
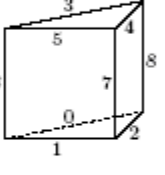
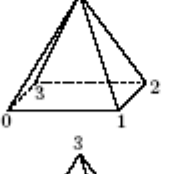
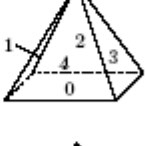
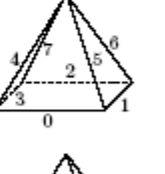
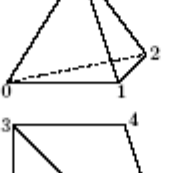
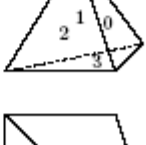
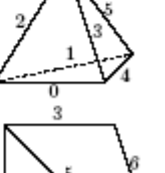
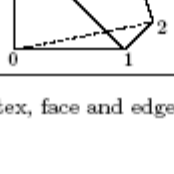
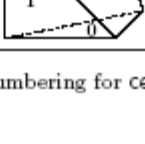
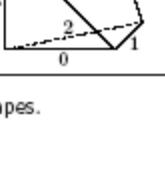
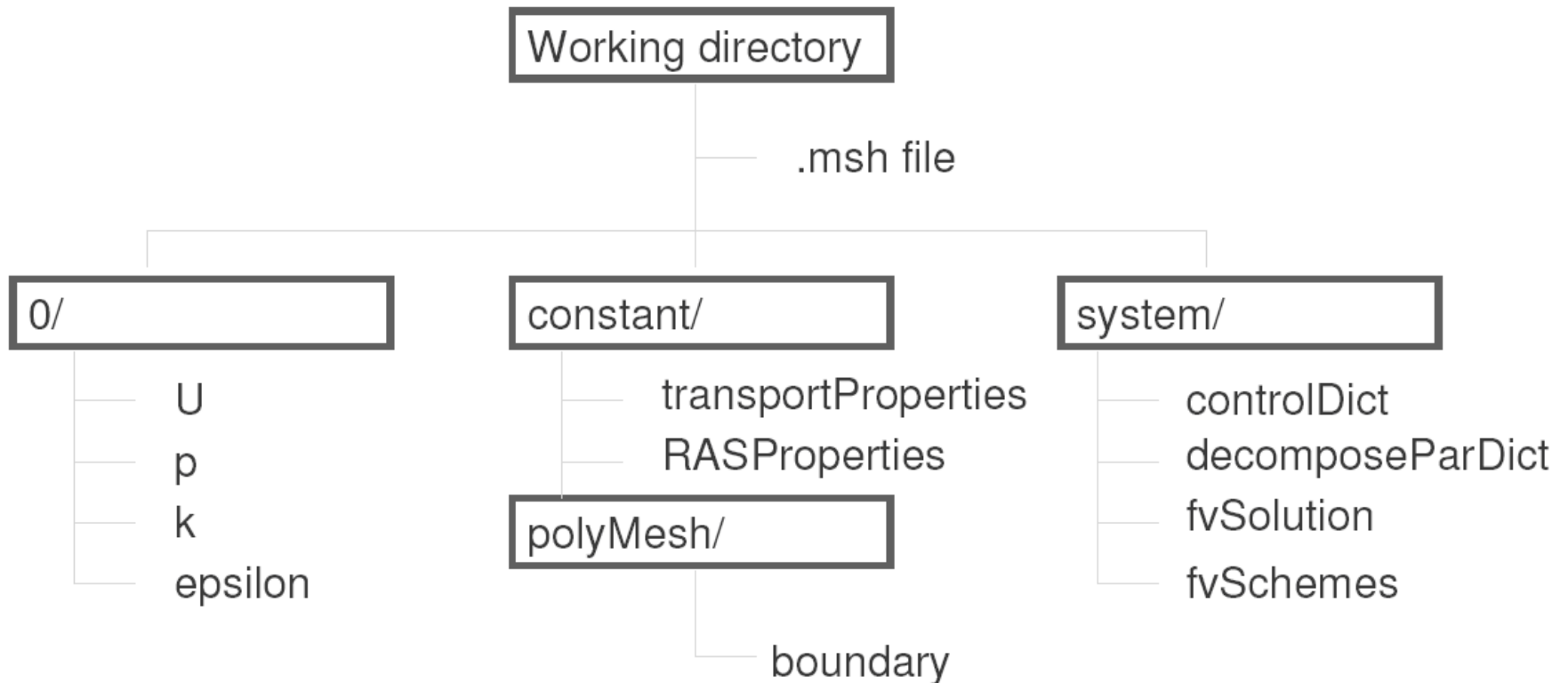
Cell type	Keyword	Vertex numbering	Face numbering	Edge numbering
Hexahedron	hex			
Wedge	wedge			
Prism	prism			
Pyramid	pyr			
Tetrahedron	tet			
Tet-wedge	tetWedge			

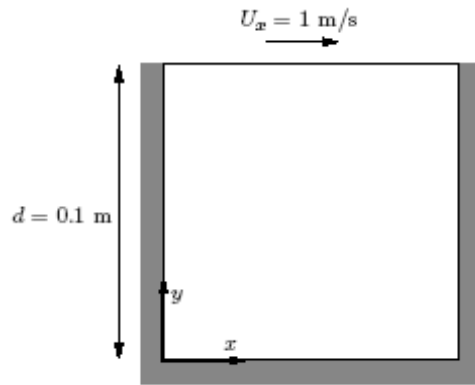
Table 5.1: Vertex, face and edge numbering for cellShapes.

Case directory in OpenFOAM

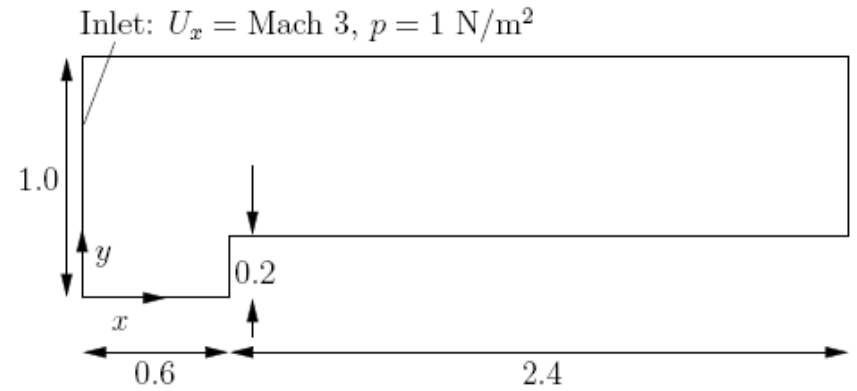
Work Structure, simpleFoam



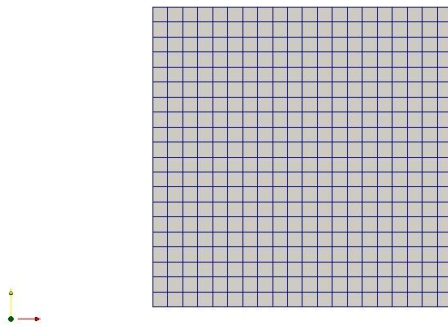
Test cases in OpenFOAM



Cavity



Forward step



Mesh for cavity



Simulation with
rhoCentralFoam

Test cases OpenFOAM

- Mass conservation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0$$

- Ideal gas

$$p = \rho RT$$

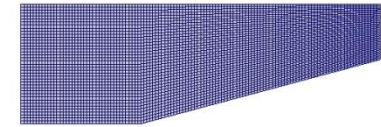
- Momentum conservation

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U} \mathbf{U}) - \nabla \cdot \mu \nabla \mathbf{U} = -\nabla p$$

- Уравнение энергии для жидкости (пренебрегая некоторыми членами вязкости),

$$e = C_v T, \text{ по закону Фурье } q = -k \nabla T$$

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho \mathbf{U} e) - \nabla \cdot \left(\frac{k}{C_v} \right) \nabla e = p \nabla \cdot \mathbf{U}$$

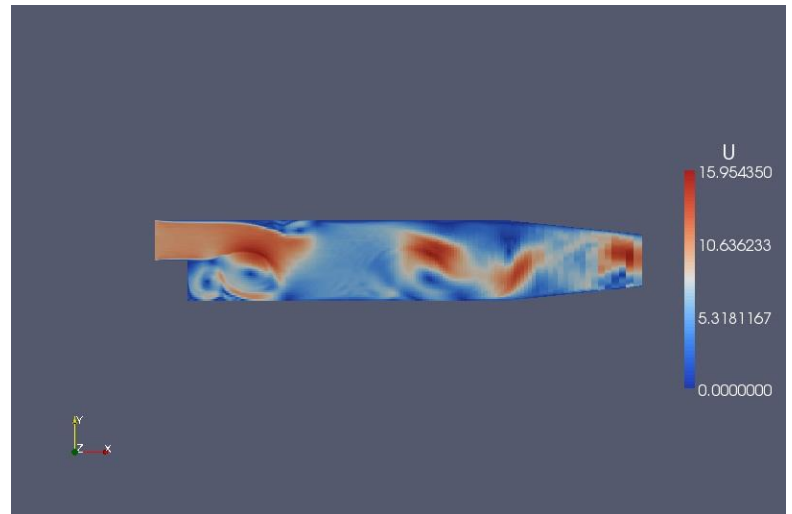
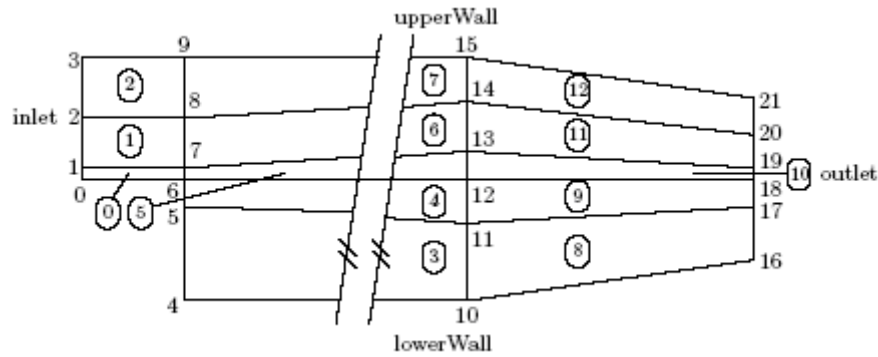
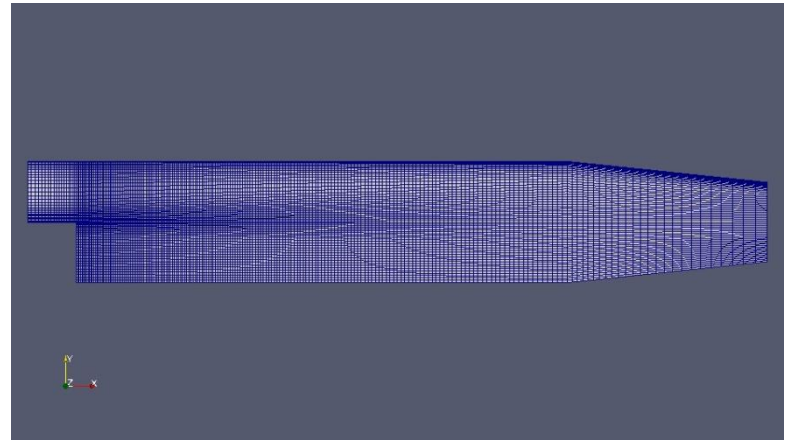
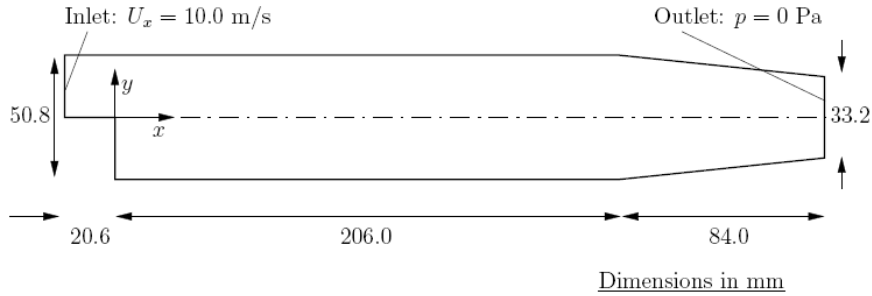


Клин с 15 град. M=5



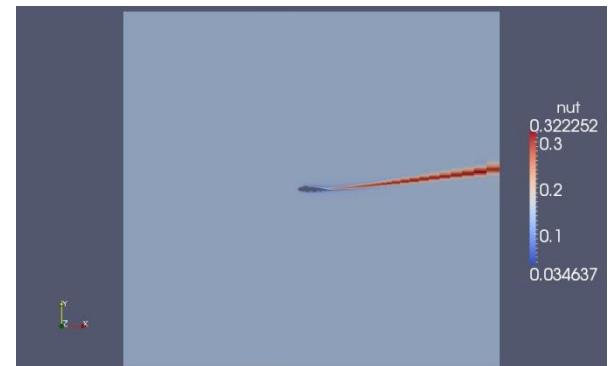
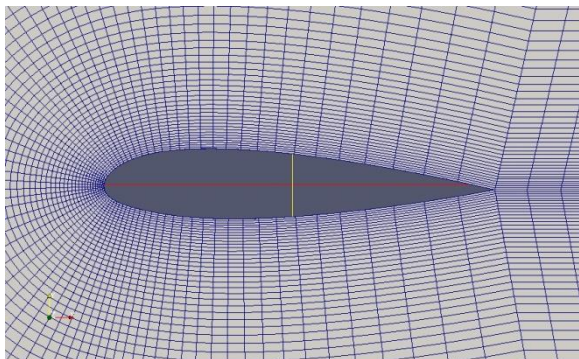
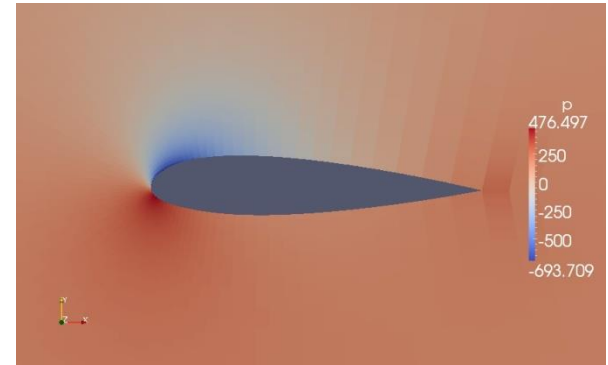
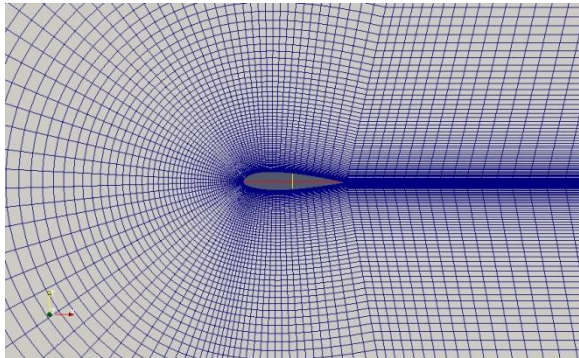
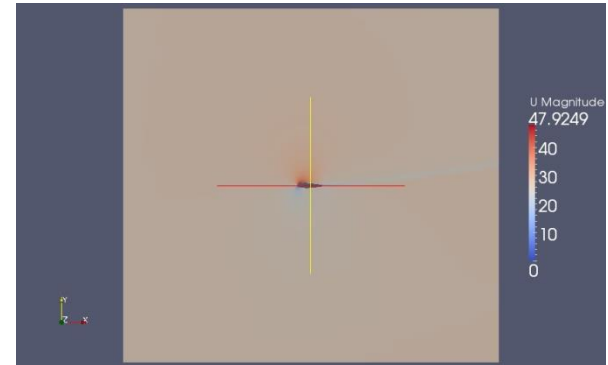
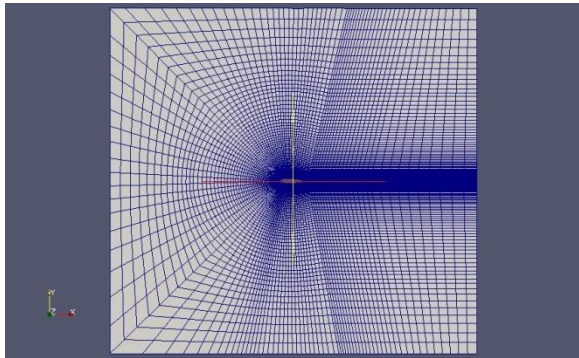
rhoCentralFoam

Test case PitzDaily in OpenFOAM



LES model. 1 equation.

airfoil2D. U= (25.75 3.62 0). S-A turbulence model.



Finite Volume Method in OpenFOAM

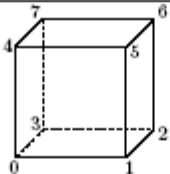
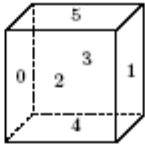
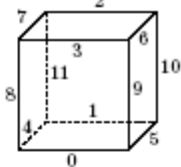
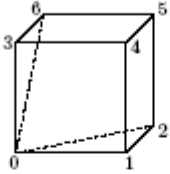
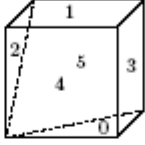
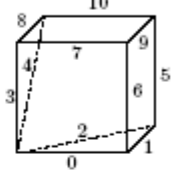
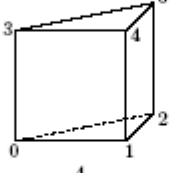
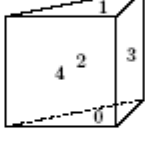
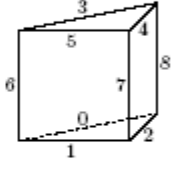
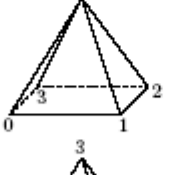
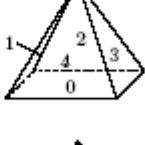
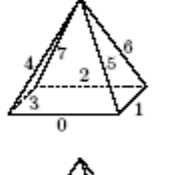
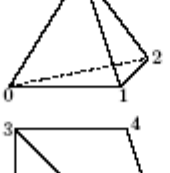
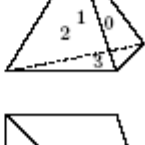
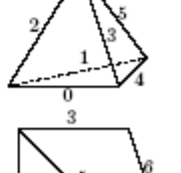
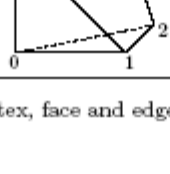
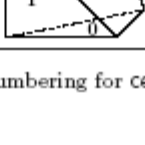
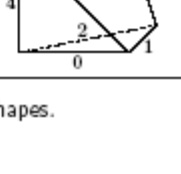
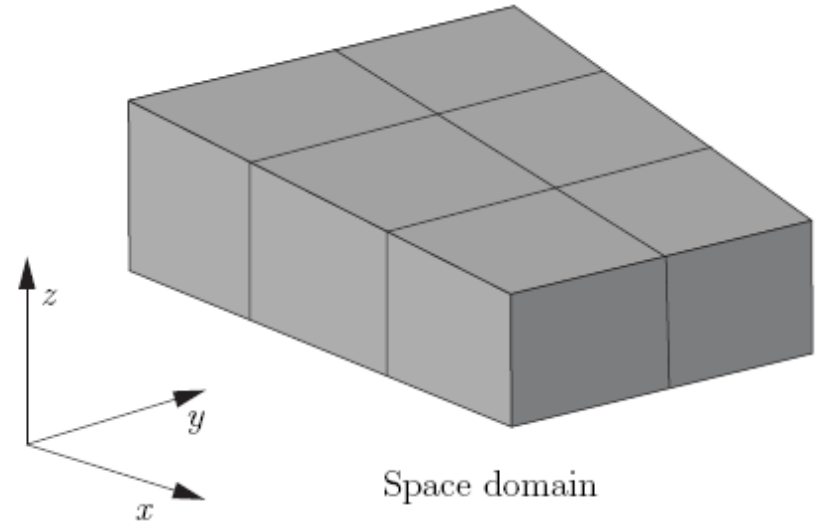
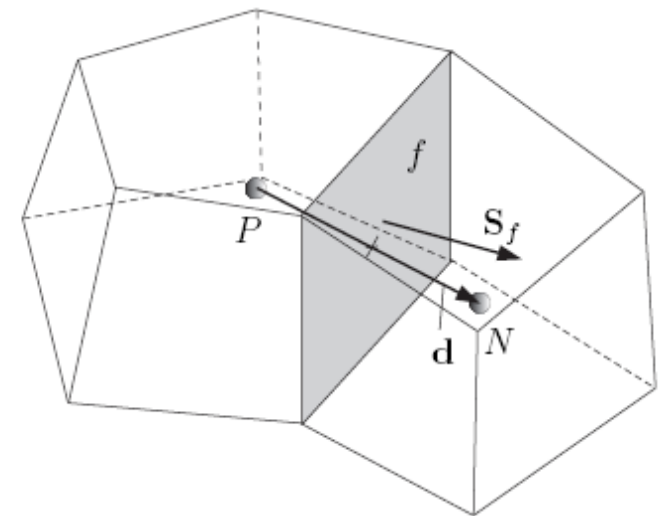
Cell type	Keyword	Vertex numbering	Face numbering	Edge numbering
Hexahedron	hex			
Wedge	wedge			
Prism	prism			
Pyramid	pyr			
Tetrahedron	tet			
Tet-wedge	tetWedge			

Table 5.1: Vertex, face and edge numbering for cellShapes.



Space domain



Neighbor cells

The general transport equation as the starting point to explain the FVM

$$\int_{V_P} \underbrace{\frac{\partial \rho \phi}{\partial t} dV}_{\text{temporal derivative}} + \int_{V_P} \underbrace{\nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{convective term}} - \int_{V_P} \underbrace{\nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{diffusion term}} = \int_{V_P} \underbrace{S_\phi(\phi) dV}_{\text{source term}}$$

1. We want to solve the general transport equation for the transported quantity in a given domain, with given boundary conditions and initial conditions.
2. This is a second order equation. For good accuracy, it is necessary that the order of the discretization is equal or higher than the order of the equation that is being discretized.
3. Hereafter we are going to assume that the discretization practice is at least second order accurate in space and time.

How to vary variables linearly

- Let us use the general transport equation as the starting point to explain the FVM,

$$\int_{V_P} \underbrace{\frac{\partial \rho \phi}{\partial t} dV}_{\text{temporal derivative}} + \int_{V_P} \underbrace{\nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{convective term}} - \int_{V_P} \underbrace{\nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{diffusion term}} = \int_{V_P} \underbrace{S_\phi(\phi) dV}_{\text{source term}}$$

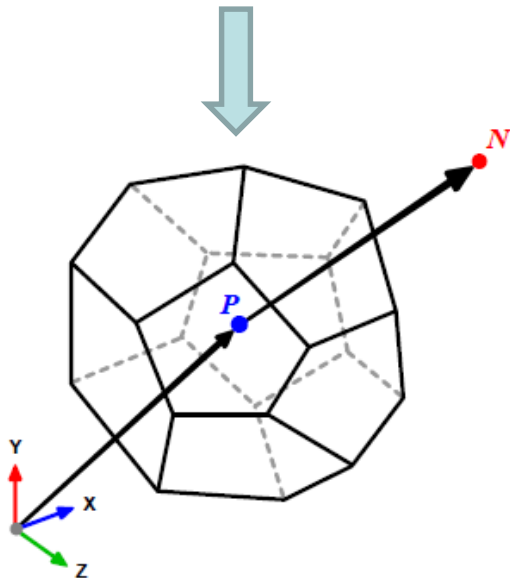
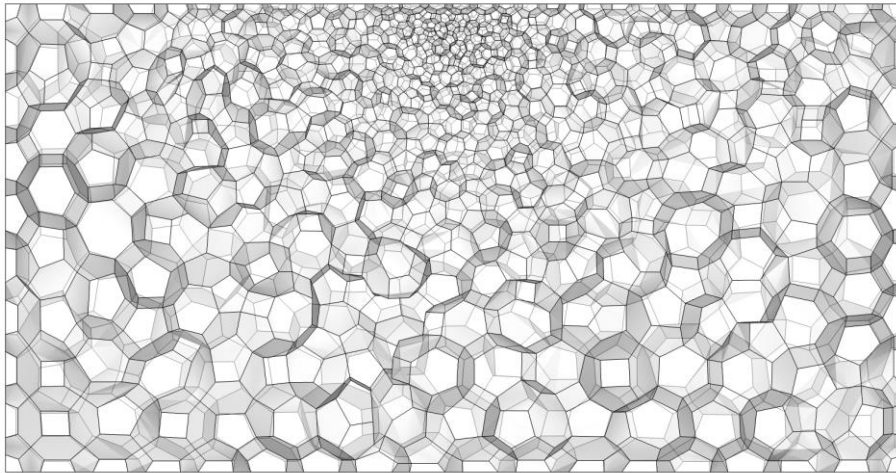
- Hereafter we are going to assume that the discretization practice is at least second order accurate in space and time.

- As consequence of this requirement, all dependent variables are assumed to vary linearly around a point P in space and instant t in time,

$$\phi(\mathbf{x}) = \phi_P + (\mathbf{x} - \mathbf{x}_P) \cdot (\nabla \phi)_P \quad \text{where} \quad \phi_P = \phi(\mathbf{x}_P)$$

$$\phi(t + \delta t) = \phi^t + \delta t \left(\frac{\partial \phi}{\partial t} \right)^t \quad \text{where} \quad \phi^t = \phi(t)$$

Let us divide the solution domain into arbitrary control volumes such as the one illustrated below.

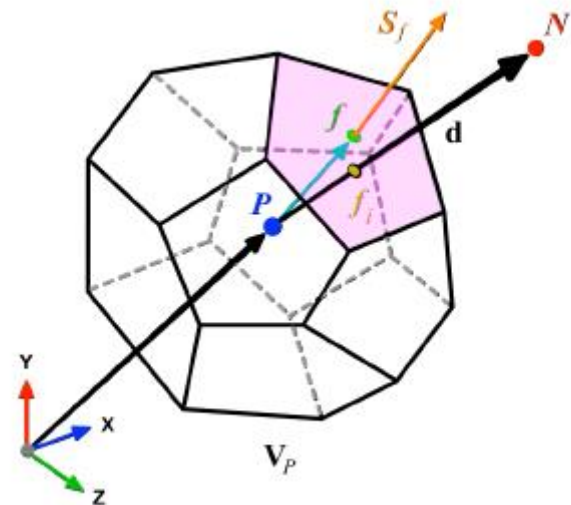
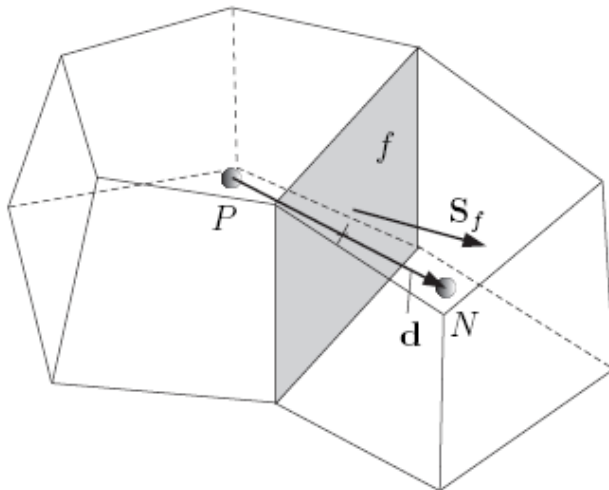


The control volumes can be of any shape (*e.g.*, tetras, hexes, prisms, pyramids, dodecahedrons, and so on).

- The only requirement is that the elements need to be convex and the faces that made up the control volume, need to be planar.
- We know all the connectivity information (P location, neighbors N 's of P , faces connectivity, vertices location and so on).

Main ideas of FVM

1. The control volume has a volume V_p and is constructed around point P , which is the centroid of the control volume. Therefore the notation V_p .
2. The vector from the centroid P of V_p to the centroid N of V_n is named \mathbf{d} .
3. The control volume faces are labeled f , which also denotes the face center.
4. The location where the vector \mathbf{d} intersects a face is f_i .
5. The face area vector point outwards from the control volume, is located at the face centroid, is normal to the face and has a magnitude equal to the area of the face.
6. The vector from the centroid P to the face center f is named Pf .



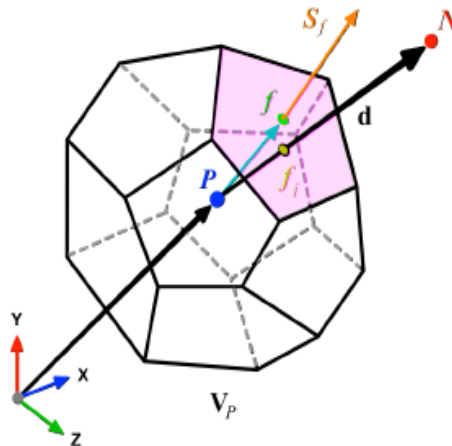
Formula for centroid of control volume

- In the control volume illustrated, the centroid P is given by

$$\int_{V_P} (\mathbf{x} - \mathbf{x}_P) dV = 0$$

- In the same way, the centroid of face f is given by

$$\int_{S_f} (\mathbf{x} - \mathbf{x}_P) dS = 0$$

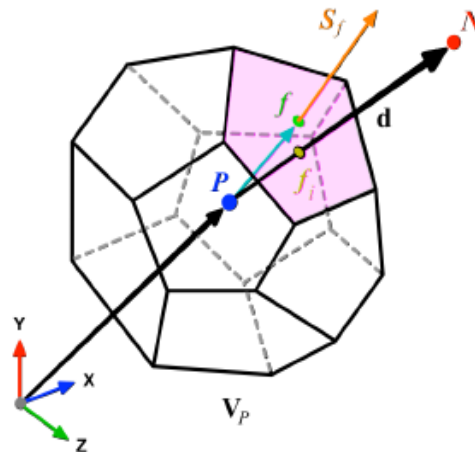


Formula for centroid of control volume

- Finally, we assume that the values of all variables are computed and stored in the centroid of the control volume V_P and that they are represented by a piecewise constant profile (the mean value),

$$\phi_P = \bar{\phi} = \frac{1}{V_P} \int_{V_P} \phi(\mathbf{x}) dV$$

- This is known as the collocated arrangement.
- This is the variable arrangement and mean value assumptions.



Main equation and discretization of terms

$$\int_{V_P} \underbrace{\frac{\partial \rho \phi}{\partial t} dV}_{\text{temporal derivative}} + \int_{V_P} \underbrace{\nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{convective term}} - \int_{V_P} \underbrace{\nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{diffusion term}} = \int_{V_P} \underbrace{S_\phi(\phi) dV}_{\text{source term}}$$

$$\int_V \nabla \cdot \mathbf{a} dV = \oint_{\partial V} d\mathbf{S} \cdot \mathbf{a} \quad \text{Gauss's theorem}$$

Convective term:

$$\int_{V_P} \underbrace{\nabla \cdot (\rho \mathbf{u} \phi) dV}_{\text{convective term}} = \oint_{\partial V_P} \underbrace{d\mathbf{S} \cdot (\rho \mathbf{u} \phi)}_{\text{convective flux}} = \sum_f \int_f d\mathbf{S} \cdot (\rho \mathbf{u} \phi)_f \approx \sum_f \mathbf{S}_f \cdot (\overline{\rho \mathbf{u} \phi})_f = \sum_f \mathbf{S}_f \cdot (\rho \mathbf{u} \phi)_f$$

Diffusive term:

$$\int_{V_P} \underbrace{\nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV}_{\text{diffusion term}} = \oint_{\partial V_P} \underbrace{d\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi)}_{\text{diffusive flux}} = \sum_f \int_f d\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi)_f \approx \sum_f \mathbf{S}_f \cdot (\overline{\rho \Gamma_\phi \nabla \phi})_f = \sum_f \mathbf{S}_f \cdot (\rho \Gamma_\phi \nabla \phi)_f$$

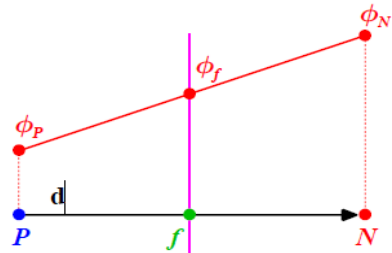
Source term:

$$\int_{V_P} S_\phi(\phi) dV = S_c V_P + S_p V_P \phi_P$$

Gradient term:

$$(\nabla \phi)_P = \frac{1}{V_P} \sum_f (\mathbf{S}_f \phi_f)$$

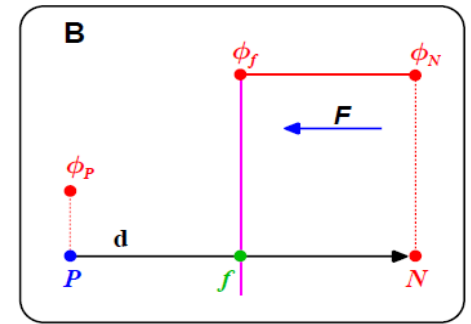
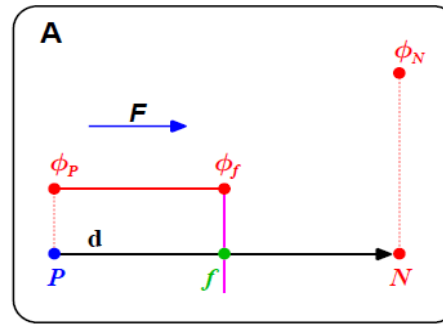
Numerical schemes of 1 and 2 orders



$$\phi_f = f_x \phi_P + (1 - f_x) \phi_N$$

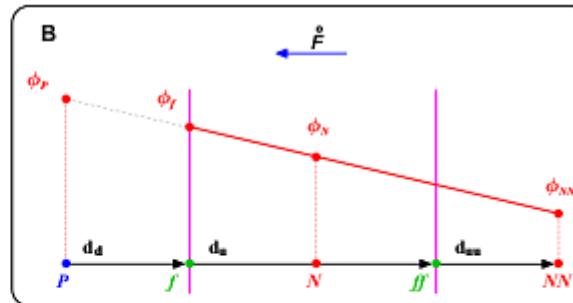
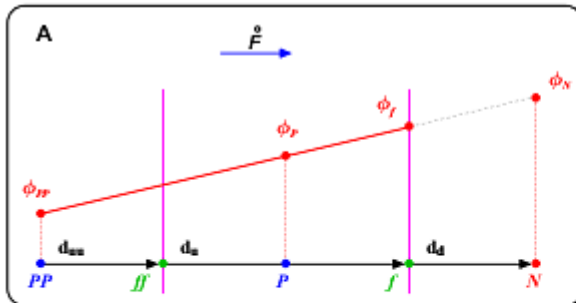
$$f_x = \frac{fN}{PN} = \frac{|\mathbf{x}_f - \mathbf{x}_N|}{|\mathbf{d}|}$$

Central differencing scheme
(Second order accurate)



$$\phi_f = \begin{cases} \phi_f = \phi_P & \text{for } \overset{\circ}{F} \geq 0, \\ \phi_f = \phi_N & \text{for } \overset{\circ}{F} < 0. \end{cases}$$

Upwind differencing scheme (First order accurate)



Second order upwind
(SOU)
Differencing scheme

$$\phi_f = \begin{cases} \phi_P + \frac{1}{2}(\phi_P - \phi_{PP}) = \frac{2}{3}\phi_P - \frac{1}{2}\phi_{PP} & \text{for } \overset{\circ}{F} \geq 0, \\ \phi_N + \frac{1}{2}(\phi_N - \phi_{NN}) = \frac{2}{3}\phi_N - \frac{1}{2}\phi_{NN} & \text{for } \overset{\circ}{F} < 0. \end{cases}$$

SOU scheme and limiter function

- To prevent oscillations in the SOU, we add a gradient or slope limiter function $\psi(r)$.

$$\phi_f = \begin{cases} \phi_P + \frac{1}{2}\psi_P^-(\phi_P - \phi_{PP}) & \text{for } \dot{F} \geq 0, \\ \phi_N - \frac{1}{2}\psi_P^+(\phi_{NN} - \phi_N) & \text{for } \dot{F} < 0. \end{cases}$$

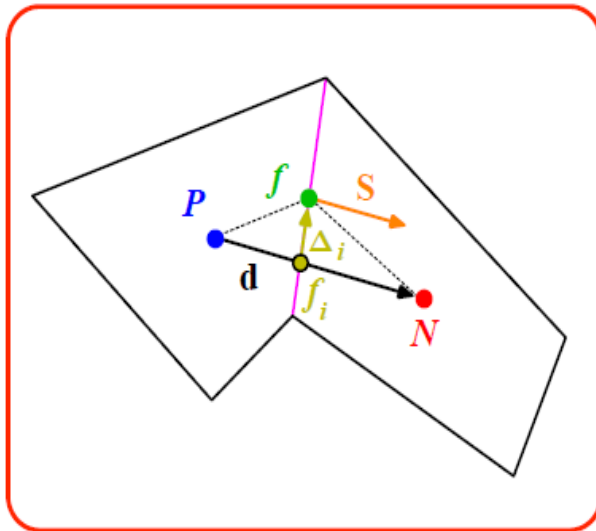
- When the limiter detects strong gradients or changes in slope, it switches to low resolution (upwind).
- The concept of the limiter function $\psi(r)$ is based on monitoring the ratio of successive gradients, e.g.,

$$r_P^- = \frac{\phi_N - \phi_P}{\phi_P - \phi_{PP}} \quad \text{and} \quad r_P^+ = \frac{\phi_P - \phi_N}{\phi_N - \phi_{NN}}$$

- By adding a well designed limiter function $\psi(r)$, we get a high resolution (second order accurate), and bounded scheme. This is a TVD scheme.

Skew mesh

- In the case of a skew mesh (as the one in the figure), we should introduce a correction in order to maintain second order accuracy and avoid unboundedness,



$$1. \phi_f = \phi_{f_i} + \Delta_i \cdot (\nabla\phi)_{f_i}$$

$$2. \phi_{f_i} = f_x \phi_P + (1 - f_x) \phi_N$$

$$\nabla\phi_{f_i} = f_x \nabla\phi_P + (1 - f_x) \nabla\phi_N$$

$$f_x = \frac{f_i N}{PN} = \frac{|\mathbf{x}_{f_i} - \mathbf{x}_N|}{|\mathbf{d}|}$$

$$3. (\nabla\phi)_P = \frac{1}{V_P} \sum_f \mathbf{S}_f \phi_f = \frac{1}{V_P} \sum_f \mathbf{S}_f \left[\phi_{f_i} + \Delta_i \cdot (\nabla\phi)_{f_i} \right]$$

$$\nabla\phi_{f_i} = \frac{(\phi_N - \phi_P)}{|\mathbf{d}|}$$

Initial approximation of the face gradient

Standard solvers in OpenFOAM

- 1) icoFoam – solver for incompressible flow
- 2) rhoCentralFoam - solver for compressible flow with KNT scheme
- 3) simpleFoam - solver for steady incompressible, turbulent flow.
- 4) pisoFoam – solver for unsteady incompressible turbulent flow.
- 5) sonicFoam – solver for unsteady compressible turbulent flow.
- 6) buoyantSimpleFoam – Steady-state solver for buoyant, turbulent flow of compressible fluids
- 7) interFoam – solver for multiphase flow with VOF
- 8) twoPhaseEulerFoam - solver for multiphase flow using Eulerian-Eulerian approach
- 9) dsmcFoam DSMC= Direct Simulation Monte-Carlo solver for rarefied gas dynamics
- 10) engineFoam – solver for flow simulation in internal combustion engine

Numerical schemes in OpenFOAM

- **Convective terms:**

- **Central differencing schemes:**

- Linear – central differencing (CD) (Second order, unbounded)

- Midpoint

- **Wind schemes:**

- Upwind differencing (UD) (First order, bounded)

- LinearUpwind

- skewLinear

- QUICK (First/second order, bounded)

- TVD schemes:**

- LimitedLinear

- vanLeer

- MUSCL

- limitedCubic

- NVD – normalized variable diagram**

- SFCD (self-filtered central differencing)

- (Second order, bounded)

- Gamma & GammaV (Schemes of H.Jasak)

- (First/second order, bounded)

- **Time differencing schemes:**

- Euler (1 and 2 order);

- Crank-Nikolson (2 order);

- Backward;

- Limited backward

- Schemes for diffusive terms:**

- Gauss linear – 2 order

- Gauss limited linear

- leastSquares

- Fourth – 4 order

More than 50 different combination of schemes

Examples of Boundary Conditions

<i>Name of BC</i>	<i>Description</i>
fixedValue	Boundary Condition Type 1. Dirihle condition
fixedGradient	Boundary Condition Type 2.
zeroGradient	Boundary Condition Type 2. Neiman condition
inletOutlet	Works as BC Type 1 as flow goes inside domain and works as BC Type 2 as flow goes out of domain
outletInlet	Opposite to inletOutlet
Mixed, symmetry plane, periodic and cyclic	Mixed condition, symmetry condition, periodic and cyclic conditions
freestream, freestreamPressure	Freestrem BC

Dimension of values in OpenFOAM

No.	Characteristics	Unit of measure	Symbol
1	Mass	Kilogram	Kg
2	Length	Metr	M
3	Time	Second	S
4	Temperature	Temperature	K
5	Amount of substance	Mol	Mol
6	Current	A	A
7	Light intensity	Candella	Candella

Turbulence models for incompressible flow

- | | |
|-----------------------|--|
| 1) kEpsilon | Standard high- Re $k-\epsilon$ model |
| 2) kOmega | Standard high- Re $k-\omega$ model |
| 3) kOmegaSST | $k-\omega$ -SST model |
| 4) RNGkEpsilon | RNG $k-\epsilon$ model |
| 5) NonlinearKEShih | Non-linear Shih $k-\epsilon$ model |
| 6) LienCubicKE | Lien cubic $k-\epsilon$ model |
| 7) qZeta | $q-\zeta$ model |
| 8) LaunderSharmaKE | Launder-Sharma low- Re $k-\epsilon$ model |
| 9) LamBremhorstKE | Lam-Bremhorst low- Re $k-\epsilon$ model |
| 10) LienCubicKELowRe | Lien cubic low- Re $k-\epsilon$ model |
| 11) LienLeschzinerLow | Lien-Leschziner low- Re $k-\epsilon$ model |
| 12) LRR | Launder-Reece-Rodi RSTM |
| 13) LaunderGibsonRSTM | Launder-Gibson RSTM with wall-reflection terms
$k-\epsilon$ |
| 14) realizableKE | Realizable $k-\epsilon$ model |
| 15) SpalartAllmaras | Spalart-Allmaras 1-eqn mixing-length model |

k-omega SST turbulence model

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial}{\partial x_i}(\rho U_i k) = \frac{\partial}{\partial x_i} \left[\left(\mu + \frac{\mu_t}{\sigma_k} \right) \text{grad}(k) \right] + P_k + G_b - \beta^* \rho k \omega$$

где $P_k = \left(2\mu_t \frac{\partial U_i}{\partial x_j} \cdot \frac{\partial U_i}{\partial x_j} - \frac{2}{3} \rho k \frac{\partial U_i}{\partial x_j} \delta_{ij} \right)$

$$\frac{\partial \rho \omega}{\partial t} + \frac{\partial}{\partial x_i}(\rho U_i \omega) = \frac{\partial}{\partial x_i} \left[\left(\mu + \frac{\mu_t}{\sigma_{\omega,1}} \right) \text{grad}(\omega) \right] + \frac{\alpha}{\nu_t} P_k - \beta_2 \rho \omega^2 + (1 - F_1) 2 \frac{\rho \sigma_{\omega,2}}{\omega} \frac{\partial k}{\partial x_k} \frac{\partial \omega}{\partial x_k}$$

$$\tilde{P}_k = \min(P_k, 10\beta^* \rho \omega k) \quad P_k = \left[\mu_t \left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} - \frac{2}{3} \delta_{ij} \frac{\partial U_k}{\partial x_k} \right) - \frac{2}{3} \delta_{ij} \rho k \right] \frac{\partial U_i}{\partial x_j}$$

$$F_1 = \tanh(\arg_1^4); \quad \arg_1 = \min \left[\max \left(\frac{\sqrt{k}}{\beta^* \omega y}; \frac{500\nu}{y^2 \omega} \right); \frac{4\rho \sigma_{\omega,2} k}{CD_{k\omega} y^2} \right];$$

$$CD_{k\omega} = \max \left(2\rho \sigma_{\omega,2} \frac{1}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j}; 10^{-10} \right)$$

$$\mu_t = \text{Min} \left(\frac{\rho k}{\omega}; \frac{a_1 \rho k}{|S_{ij}| F_2} \right),$$

$$|S_{ij}| = \sqrt{2S_{ij}S_{ij}}$$

β^*	β_2	σ_k	$\sigma_{\omega,1}$	$\sigma_{\omega,2}$	γ_2
0.09	0.083	1.0	2.0	1.17	0.44

Turbulence model of Spalart-Allmaras

(<http://turbmodels.larc.nasa.gov/spalart.html>)

$$\frac{\partial(\rho\tilde{\nu})}{\partial t} + \frac{\partial}{\partial x_j}(\rho\tilde{\nu}u_j) = \frac{1}{\sigma_{\tilde{\nu}}} \left\{ \frac{\partial}{\partial x_j} \left[(\mu + \rho\tilde{\nu}) \frac{\partial\tilde{\nu}}{\partial x_j} \right] + C_{b2}\rho \left(\frac{\partial\tilde{\nu}}{\partial x_j} \right)^2 \right\} + G_{\nu} - Y_{\nu};$$

$$\mu_t = \rho\tilde{\nu}f_{\nu1}; \quad f_{\nu1} = \frac{\chi^3}{\chi^3 + C_{\nu1}^3}; \quad \chi \equiv \frac{\tilde{\nu}}{\nu}; \quad G_{\nu} = C_{b1}\rho\tilde{S}\tilde{\nu}; \quad \tilde{S} \equiv \bar{S} + \frac{\tilde{\nu}}{k^2d^2}f_{\nu2};$$

$$f_{\nu2} = 1 - \frac{\chi}{1 + \chi f_{\nu1}}; \quad \bar{S} \equiv |\Omega_{ij}| \equiv \sqrt{2\Omega_{ij}\Omega_{ij}}; \quad \Omega_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right);$$

$$\bar{S} \equiv |\Omega_{ij}| + C_{prod} \min(0, |S_{ij}| - |\Omega_{ij}|); \quad |S_{ij}| = \sqrt{2S_{ij}S_{ij}}; \quad S_{ij} = \frac{1}{2} \left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} \right);$$

$$Y_{\nu} = C_{w1}\rho f_w \left(\frac{\tilde{\nu}}{d} \right)^2; \quad f_w = g \left[\frac{1 + C_{w3}^6}{g^6 + C_{w3}^6} \right]^{\frac{1}{6}}; \quad g = r + C_{w2}(r^6 - r); \quad r \equiv \frac{\tilde{\nu}}{\tilde{S}k^2d^2};$$

$$C_{prod} = 2.0; \quad C_{b2} = 0.622; \quad \sigma_{\tilde{\nu}} = \frac{2}{3}; \quad C_{\nu1} = 7.1;$$

$$C_{w1} = \frac{C_{b1}}{k^2} + \frac{(1 + C_{b2})}{\sigma_{\tilde{\nu}}}; \quad C_{w2} = 0.3; \quad C_{w3} = 2.0; \quad k = 0.4187$$

Ω – The rotation tensor;

S – Stress tensor of velocity

Models of wall functions for turbulence models

The region near wall may be divided on 3 zones:

1) Viscous layer:

$$u^+ = y^+ \quad u^+ \equiv \frac{\bar{u}}{u_\tau} \quad y^+ \equiv \frac{u_\tau y}{\nu} \quad u_\tau = \sqrt{\tau_\omega}$$

2) Buffer layer:

$$u^+ = 5 \ln y^+ + 3.05$$

3) Logarithmic layer :

$$u^+ = (1/\kappa) \ln(Ey^+)$$

κ – const Karman, E – const for wall $E = 8.8$).

OpenFOAM realization
For different values:

- nut: nutWallFunction,
- epsilon:
epsilonWallFunction,
- omega:
omegaWallFunction,
- k, q, R:
kqRWallFunction
- nut –
nutSpalartAllmarasWall
Function.

For temperature:

- alphas:
alphasWallFunction.

BC for k-epsilon и k-omega SST turbulence models

k Kinetic energy of turbulence

$$k = \frac{3}{2} (U I)^2$$

U Flow velocity

I Intensity of turbulence

ϵ Dissipation of energy

$$\epsilon = C_{\mu}^{\frac{3}{4}} \frac{k^{\frac{3}{2}}}{l}$$

where C_{μ} 0.09 Const of turbulence model

k Kinetic energy of turbulence

l turbulence scale

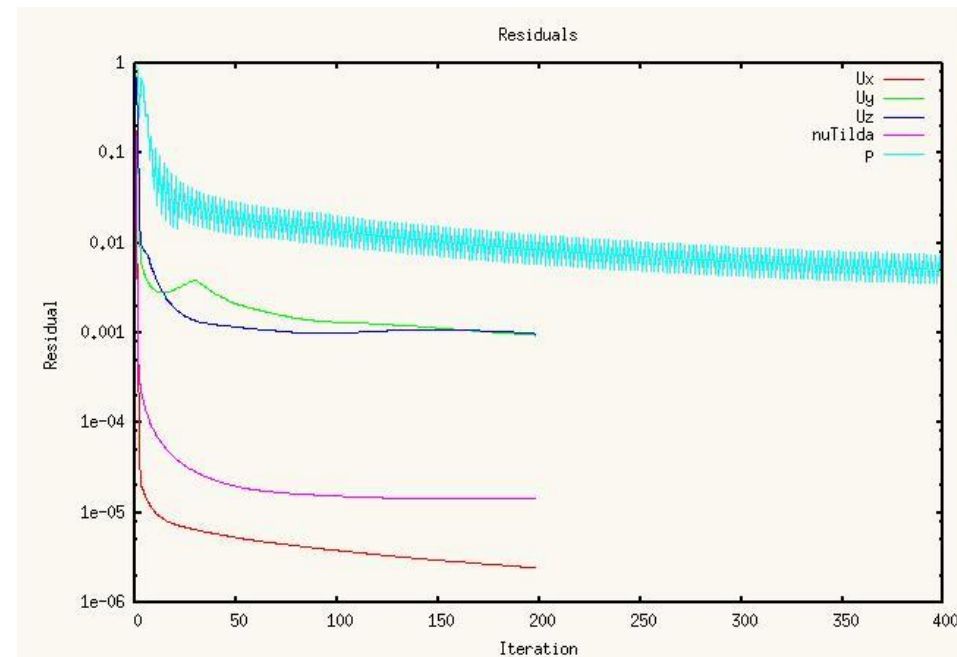
$$\omega = C_{\mu}^{-\frac{1}{4}} \frac{\sqrt{k}}{l} \quad \text{the rate of dissipation of the eddies}$$

Internal Libraries in OpenFOAM



- "libincompressibleRASModels.so" - library for models of turbulence;
- libforces.so – library for aerodynamic forces and moments;
- libLESfilters.so – library for LES filters;
- libODE.so – library for Ord Differential Equations solver;
- libscotch.so – library for parallel algorithm 'scotch';
- libliquids.so – library for liquids with different properties;
- libsolids.so – library for solids with different properties;
- libOpenFoamTurbo.so – library for definition of 1D fixed value profile (radial or vertical) for a typical

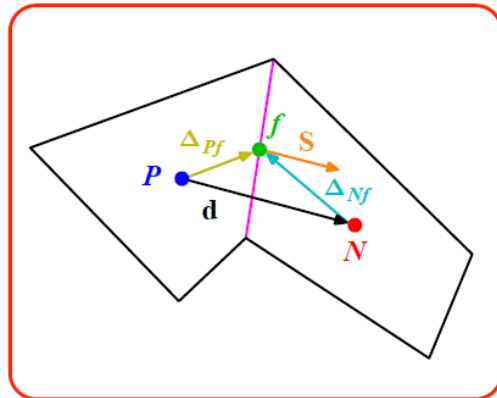
Graphics of residuals using Linux gnuplot and python script



Residuals for Ux,Uy,Uz, p, nuTilda

A skew mesh

- In the case of a skew mesh (as the one in the figure), we should introduce a correction in order to maintain second order accuracy and avoid unboundedness,



$$1. (\nabla\phi)_P = \frac{1}{V_P} \sum_f (\mathbf{S}_f \phi_f)$$

$$2. \phi_f = \frac{(\phi_P + \Delta_{Pf} \cdot \nabla\phi_P) + (\phi_N + \Delta_{Nf} \cdot \nabla\phi_N)}{2}$$

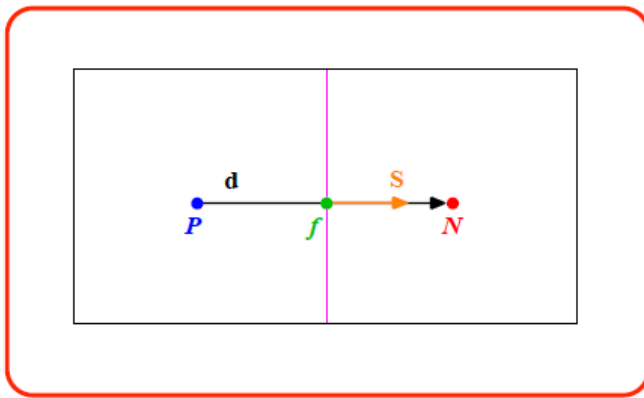
$$3. (\nabla\phi)_f = f_x (\nabla\phi)_P + (1 - f_x) (\nabla\phi)_N$$

$$f_x = \frac{fN}{PN} = \frac{|\mathbf{x}_f - \mathbf{x}_N|}{|\mathbf{d}|}$$

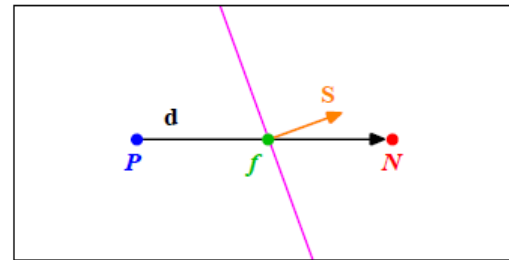
$$\phi_f = \frac{(\phi_P + \phi_N)}{2} \quad \text{and} \quad \nabla\phi_f = \frac{(\nabla\phi_P + \nabla\phi_N)}{2} \quad \text{are the initial approximations}$$

Orthogonal and non-orthogonal

- By looking the figures below, the face values appearing in the diffusive flux in an orthogonal mesh can be computed as follows,



Orthogonal mesh

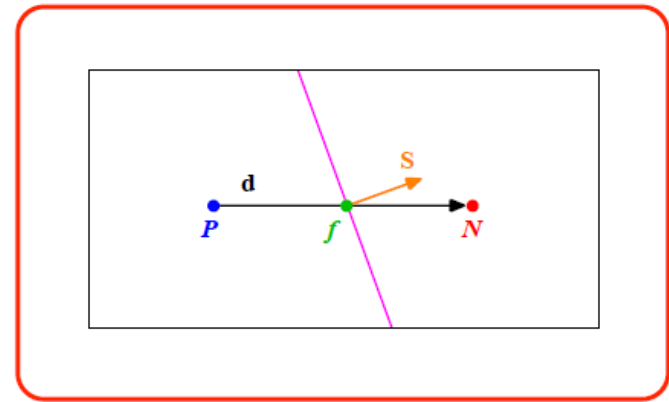
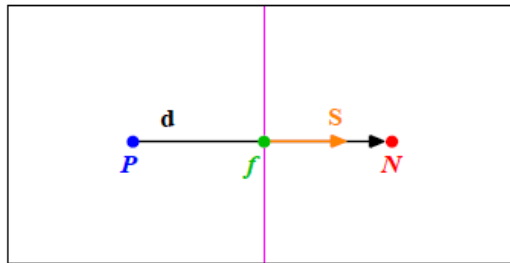


$$\mathbf{S} \cdot (\nabla \phi)_f = |\mathbf{S}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}.$$

- This is a central difference approximation of the first order derivative. This type of approximation is second order accurate.

Orthogonal and non-orthogonal

- By looking the figures below, the face values appearing in the diffusive flux in a non-orthogonal mesh (20°) can be computed as follows,

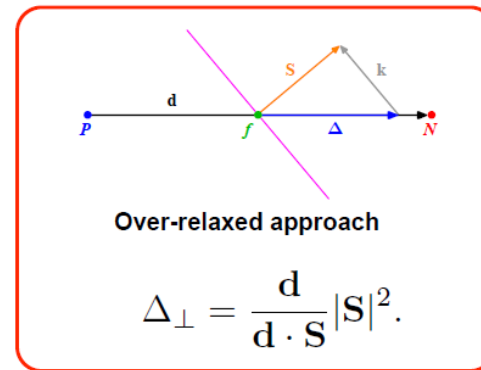
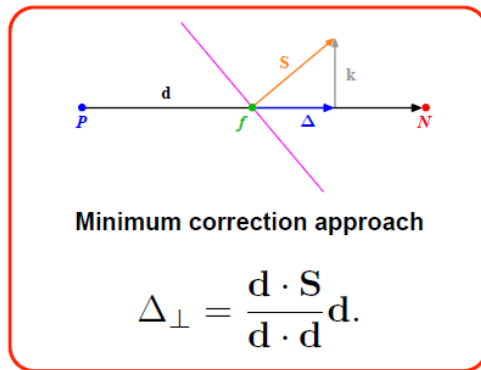


Non-orthogonal mesh

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_\perp| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}} .$$

- This type of approximation is second order accurate but involves a larger truncation error. It also uses a larger numerical stencil, which make it less stable.

- By looking the figures below, the face values appearing in the diffusive flux in a non-orthogonal mesh (40°) can be computed as follows,



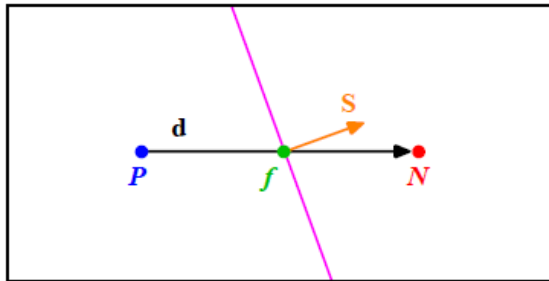
$$\mathbf{S} = \Delta_{\perp} + \mathbf{k}.$$

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}.$$

Face gradient

- The face gradient of the non-orthogonal contribution is computed by using linear interpolation from the gradient of the control volumes centroid, computed using Gauss theorem.

$$\mathbf{S} \cdot (\nabla \phi)_f = \underbrace{|\Delta_{\perp}| \frac{\phi_N - \phi_P}{|\mathbf{d}|}}_{\text{orthogonal contribution}} + \underbrace{\mathbf{k} \cdot (\nabla \phi)_f}_{\text{non-orthogonal contribution}}$$



$$\int_{V_P} \nabla \phi dV = \oint_{\partial V_P} d\mathbf{S} \phi$$

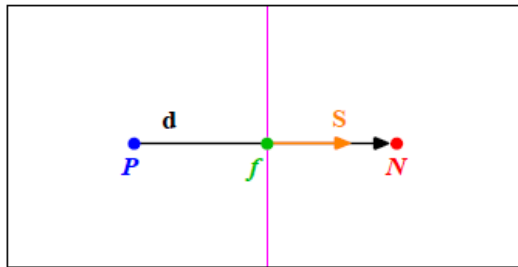
$$(\nabla \phi)_P V_P = \sum_f (\mathbf{S}_f \phi_f)$$

$$(\nabla \phi)_P = \frac{1}{V_P} \sum_f (\mathbf{S}_f \phi_f)$$

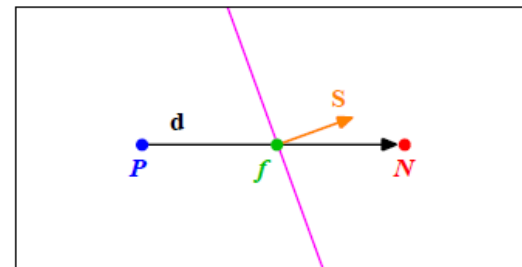
$$(\nabla \phi)_f = f_x (\nabla \phi)_P + (1 - f_x) (\nabla \phi)_N \quad \text{where} \quad f_x = \frac{fN}{PN} = \frac{|\mathbf{x}_f - \mathbf{x}_N|}{|\mathbf{d}|}$$

A mesh induced errors

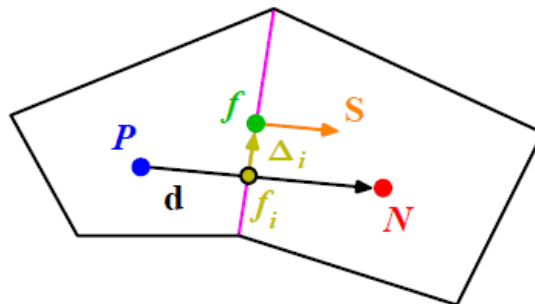
- **Mesh induced errors.** In order to maintain second order accuracy, and to avoid unboundedness, we need to correct non-orthogonality and skewness errors.



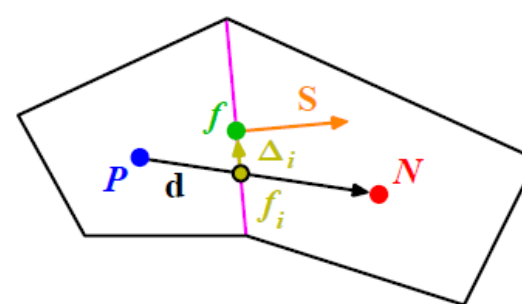
Orthogonal and non skew mesh



Non-orthogonal and non skew mesh



Orthogonal and skew mesh



Non-orthogonal and skew mesh

General transport equation

- Using the previous equations to evaluate the general transport equation over all the control volumes, we obtain the following semi-discrete equation

$$\int_{V_P} \underbrace{\frac{\partial \rho \phi}{\partial t} dV}_{\text{temporal derivative}} + \sum_f \underbrace{\mathbf{S}_f \cdot (\rho \mathbf{u} \phi)_f}_{\text{convective flux}} - \sum_f \underbrace{\mathbf{S}_f \cdot (\rho \Gamma_\phi \nabla \phi)_f}_{\text{diffusive flux}} = \underbrace{(S_c V_P + S_p V_P \phi_P)}_{\text{source term}}$$

where $\mathbf{S} \cdot (\rho \mathbf{u} \phi) = F^C$ is the convective flux and $\mathbf{S} \cdot (\rho \Gamma_\phi \nabla \phi) = F^D$ is the diffusive flux.

- After spatial discretization, we can proceed with the temporal discretization. By proceeding in this way we are using the Method of Lines (MOL).
- The main advantage of the MOL method, is that it allows us to select numerical approximations of different accuracy for the spatial and temporal terms. Each term can be treated differently to yield to different accuracies.

Time discretization

- Now, we evaluate in time the semi-discrete general transport equation

$$\int_t^{t+\Delta t} \left[\left(\frac{\partial \rho \phi}{\partial t} \right)_P V_P + \sum_f \mathbf{S}_f \cdot (\rho \mathbf{u} \phi)_f - \sum_f \mathbf{S}_f \cdot (\rho \Gamma_\phi \nabla \phi)_f \right] dt$$
$$= \int_t^{t+\Delta t} (S_c V_P + S_p V_P \phi_P) dt.$$

- At this stage, we can use any time discretization scheme, e.g., Crank-Nicolson, euler implicit, forward euler, backward differencing, adams-bashforth, adams-moulton.
- It should be noted that the order of the temporal discretization of the transient term does not need to be the same as the order of the discretization of the spatial terms. Each term can be treated differently to yield different accuracies. As long as the individual terms are at least second order accurate, the overall accuracy will also be second order.

Different Operators in OpenFoam

Term description	Implicit/explicit	Mathematical expression	fvm::/fvc:: functions
Laplacian	Implicit/Explicit	$\nabla \cdot \Gamma \nabla \phi$	laplacian(Gamma,phi)
Time derivative	Implicit/Explicit	$\partial \phi / \partial t$	ddt(phi)
Convection	Implicit/Explicit	$\partial \rho \phi / \partial t$	ddt(rho, phi)
		$\nabla \cdot (\psi)$	div(psi, scheme)
Source	Implicit/Explicit	$\nabla \cdot (\psi \phi)$	div(psi, phi, word)
		$\rho \phi$	div(psi, phi)
			Sp(rho, phi)
			SuSp(rho, phi)

ϕ : vol<type>Field, ρ : scalar, volScalarField, ψ : surfaceScalarField

Continuity equation

([OpenFoam/OpenFoam - 1.6/src/finiteVolume/cfdTools/compressible/rhoEqn.h](#))

```
\*-----*\n{\n  solve(fvm::ddt(rho) + fvc::div(phi));\n}\n// ***** //
```

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0$$

where ρ is density and U is velocity.

Momentum equations

([applications/solvers/heatTransfer/buoyantFoam/UEqn.H](#))

$$\frac{\partial(\rho U)}{\partial t} + \nabla \cdot (\rho U \cdot U) - \nabla \cdot [\mu_{eff}(\nabla U + \nabla U^T)] - \frac{2}{3}\mu_{eff}(\nabla \cdot U)I = -\nabla p_d - \nabla \rho g_h \quad (2)$$

where μ_{eff} is the effective viscosity

$$\mu_{eff} = \mu_{laminar} + \mu_{turbulent}$$

$\mu_{laminar}$ is laminar kinematics viscosity,

is turbulent viscosity.

$\mu_{turbulent}$

Pressure correction equation

p_d is dynamic pressure ([applications/solvers/heatTransfer/buoyantFoam/pEqn.H](#)):

$$\frac{\partial \psi p_d}{\partial t} + \frac{\partial \psi}{\partial t} \cdot p_{ref} + \frac{\partial \psi \rho}{\partial t} \cdot + \nabla(\rho U) - \Delta(\rho U) \cdot p_d = 0$$

where $\psi = \frac{1}{RT}$ (s^2/m^2) - compressibility, R is the gas constant.

Energy equation

([applications/solvers/heatTransfer/buoyantFoam/hEqn.H](#))

$$\frac{\partial(\rho h)}{\partial t} + \nabla \cdot (\rho U) - \nabla \cdot (\alpha_{eff} \nabla h) = \frac{\partial p}{\partial t} + U \cdot \nabla p$$

where α_{eff} - thermal diffusivity and is given by

$$\alpha_{eff} = \alpha_{laminar} + \alpha_{turbulent}$$

The total pressure p is :

$$p = p_d + \rho g_h + p_{ref}$$

where p_d is dynamic pressure, ρg_h static pressure, g_h gravity and p_{ref} reference pressure (atmospheric).

Programming in OpenFoam

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot \rho \mathbf{U} \mathbf{U} - \nabla \cdot \mu \nabla \mathbf{U} = -\nabla p$$

```
solve
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::laplacian(mu, U)
  ==
  - fvc::grad(p)
);
```

Five basic classes in foam-extend

Space and time: polyMesh, fvMesh, Time

Field algebra: Field, DimensionedField and GeometricField

Boundary conditions: fvPatchField and derived classes

Sparse matrices: lduMatrix, fvMatrix and linear solvers

Finite Volume discretisation: fvc and fvm namespace

Representation of Time

- Main functions of Time class
 - Follow simulation in terms of time-steps: start and end time, delta t
 - Time is associated with I/O functionality: what and when to write
 - objectRegistry: all IOObjects, including mesh, fields and dictionaries registered with time class
 - Main simulation control dictionary: controlDict
 - Holding paths to <root>, <case> and associated data
- Associated class: regIOobject: database holds a list of objects, with functionality held under virtual functions

Representation of Space

- Computational mesh consists of
 - **List of points.** Point index is determined from its position in the list
 - **List of faces.** A face is an ordered list of points (defines face normal)
 - **List of cells OR owner-neighbour addressing** (defines left and right cell for each face, saving some storage and mesh analysis time)
- List of boundary patches, grouping external faces
- `polyMesh` class holds mesh definition objects
- `primitiveMesh`: some parts of mesh analysis extracted out (topo changes)
- `polyBoundaryMesh` is a list of `polyPatches`

Finite Volume Mesh

- `polyMesh` class provides mesh data in generic manner: it is used by multiple applications and discretisation methods
- For convenience, each discretisation wraps up primitive mesh functionality to suit its needs: mesh metrics, addressing etc.
- `fvMesh`: mesh-related support for the Finite Volume Method

Finite Volume Boundary Conditions

- Implementation of boundary conditions is a perfect example of a virtual class hierarchy
- Consider implementation of a boundary condition
 - Evaluate function: calculate new boundary values depending on **behaviour**: fixed value, zero gradient etc.
 - Enforce boundary type constraint based on matrix coefficients
 - Multiple if-then-else statements throughout the code: asking for trouble
 - **Virtual function interface**: run-time polymorphic dispatch
- Base class: `fvPatchField`
 - Derived from a field container
 - Reference to `fvPatch`: easy data access
 - Reference to internal field
- Types of `fvPatchField`
 - **Basic**: fixed value, zero gradient, mixed, coupled, default
 - **Constraint**: enforced on all fields by the patch: cyclic, empty, processor, symmetry, wedge, GGI
 - **Derived**: wrapping basic type for physics functionality

Sparse Matrix Class

- Some of the oldest parts of OpenFOAM: about to be thrown away for more flexibility
- Class hierarchy
 - Addressing classes: `lduAddressing`, `lduInterface`, `lduMesh`
 - LDU matrix class
 - Solver technology: preconditioner, smoother, solver
 - Discretisation-specific matrix wrapping with handling for boundary conditions, coupling and similar

LDU Matrix

- Square matrix with sparse addressing. **Enforced strong upper triangular ordering in matrix and mesh**

- Matrix stored in 3 parts in **arrow format**
 - Diagonal coefficients
 - Off-diagonal coefficients, upper triangle
 - Off-diagonal coefficients, lower triangle
- Out-of-core multiplication stored as a list of `lduInterface` with coupling functionality: executed eg. on vector matrix multiplication

LDU Matrix: Storage format

- Arbitrary sparse format. Diagonal coefficients typically stored separately
- Coefficients in 2-3 arrays: diagonal, upper and lower triangle
- Diagonal addressing implied
- Off-diagonal addressing in 2 arrays: “owner” (row index) “neighbor” (column index) array. Size of addressing equal to the number of coefficients
- The matrix structure (fill-in) is assumed to be symmetric: presence of a_{ij} implies the presence of a_{ji} . Symmetric matrix easily recognized: efficiency
- If the matrix coefficients are symmetric, only the upper triangle is stored – a symmetric matrix is easily recognized and stored only half of coefficients

```
vectorProduct(b, x) // [b] = [A] [x]
{
for (int n = 0; n < coeffs.size(); n++)
{
int c0 = owner(n);
int c1 = neighbour(n);
b[c0] = upperCoeffs[n]*x[c1];
b[c1] = lowerCoeffs[n]*x[c0];
}}
```

Finite Volume Matrix Support

- Finite Volume matrix class: `fvMatrix`
- Derived from `lduMatrix`, with a reference to the solution field
- Holding dimension set and out-of-core coefficient
- Because of derivation (insufficient base class functionality), all FV matrices are currently always scalar: **segregated** solver for vector and tensor variables
- Some coefficients (diagonal, next-to-boundary) may locally be a higher type, but this is not sufficiently flexible
- Implements standard matrix and field algebra, to allow matrix assembly at equation level: adding and subtracting matrices

- “Non-standard” matrix functionality in `fvMatrix`
 - `fvMatrix::A()` function: return matrix diagonal in FV field form
 - `fvMatrix::H()`: vector-matrix multiply with current `psi()`, using off-diagonal coefficients and rhs
 - `fvMatrix::flux()` function: consistent evaluation of off-diagonal product in “face form”. See derivation of the pressure equation

- New features: coupled matrices (each mesh defines its own addressing space) and matrices with block-coupled coefficients

Finite Volume Discretization

- Finite Volume Method implemented in 3 parts
 - **Surface interpolation**: cell-to-face data transfer
 - **Finite Volume Calculus** (f_{vc}): given a field, create a new field
 - **Finite Volume Method** (f_{vm}): create a matrix representation of an operator, using FV discretization
- In both cases, we have **static functions** with no common data. Thus, f_{vc} and f_{vm} are implemented as **namespaces**
- Discretization involves a number of choices on how to perform identical operations:
eg. gradient operator. In all cases, the signature is common
`volTensorField gradU = fvc::grad(U);`
- Multiple algorithmic choices of gradient calculation operator: Gauss theorem, least square fit, limiters etc. implemented as run-time selection
- Choice of discretization controlled by the user on a per-operator basis:
`system/fvSolution`
- Thus, each operator contains basic data wrapping, selects the appropriate function from run-time selection and calls the function using virtual function dispatch

Application pisoFoam

Description Transient solver for incompressible flow.

Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.

```
\*-----*/
#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulenceModel.H"
// ***** //
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"
    // ***** //
    Info<< "\nStarting time loop\n" << endl;
    while (runTime.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;
        #include "readPISOControls.H"
        #include "CourantNo.H"
        // Pressure-velocity PISO corrector
        {
            // Momentum predictor
            fvVectorMatrix UEqn
            (
                fvm::ddt(U)
                + fvm::div(phi, U)
                + turbulence->divDevReff(U)
            );
            UEqn.relax();
            if (momentumPredictor)
            {
                solve(UEqn == -fvc::grad(p));
            }
        }
    }
}
```

```

// --- PISO loop
for (int corr=0; corr<nCorr; corr++)
{
    volScalarField rUA = 1.0/UEqn.A();
    U = rUA*UEqn.H();
    phi = (fvc::interpolate(U) & mesh.Sf())
        + fvc::ddtPhiCorr(rUA, U, phi);
    adjustPhi(phi, U, p);
    // Non-orthogonal pressure corrector loop
    for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
    {
        // Pressure corrector
        fvScalarMatrix pEqn
        (
            fvm::laplacian(rUA, p) == fvc::div(phi)
        );
        pEqn.setReference(pRefCell, pRefValue);
        if (
            corr == nCorr-1
            && nonOrth == nNonOrthCorr)
        {
            pEqn.solve(mesh.solver("pFinal"));
        }
        else
        {
            pEqn.solve();
        }
        if (nonOrth == nNonOrthCorr)
        {
            phi -= pEqn.flux();
        }
    }
    #include "continuityErrs.H"
    U -= rUA*fvc::grad(p);
    U.correctBoundaryConditions();
}
turbulence->correct();
runTime.write();
Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
<< " ClockTime = " << runTime.elapsedClockTime() << " s"
<< nl << endl;
}
Info<< "End\n" << endl;
return 0; }
// ***** //

```

File .bashrc

- .bashrc is for settings in Linux
- File .bashrc is located in .../home/guest1
- For OpenPBS settings need to add:

```
export
```

```
PATH=/usr/local/bin:/usr/local/maui/bin:/opt/maui/bin:/opt/maui/3.3.1/bin:/opt/torque/4.0.2/bin:/opt/pdsh/2.27/bin:$PATH
```

```
export TORQUE=/opt/torque/4.0.2/bin
```

For OpenFOAM settings:

```
source /unicluster/bl2x220Cluster/opt/OpenFOAM/OpenFOAM-2.3.0/etc/bashrc
```

Then add source strings for another OpenFoam version

Literature

- S. Patankar. Numerical Heat Transfer and Fluid Flow. 1980, Taylor & Francis.
- J. H. Ferziger, M. Peric. Computational Methods for Fluid Dynamics. 2001, Springer.
- H. K. Versteeg, W. Malalasekera. An Introduction to Computational Fluid Dynamics. 2007, Prentice Hall.
- Charles Hirsch. Numerical Computation of Internal and External Flows. 2007.
- D. Wilcox. Turbulence Modeling for CFD 2006, DCW Industries.
- H. Jasak. Error analysis and estimation in the Finite Volume method with applications to fluid flows. PhD Thesis. 1996. Imperial College, London.
- H. Rusche. Computational fluid dynamics of dispersed two-phase flows at high phase fractions. PhD Thesis. 2002. Imperial College, London.