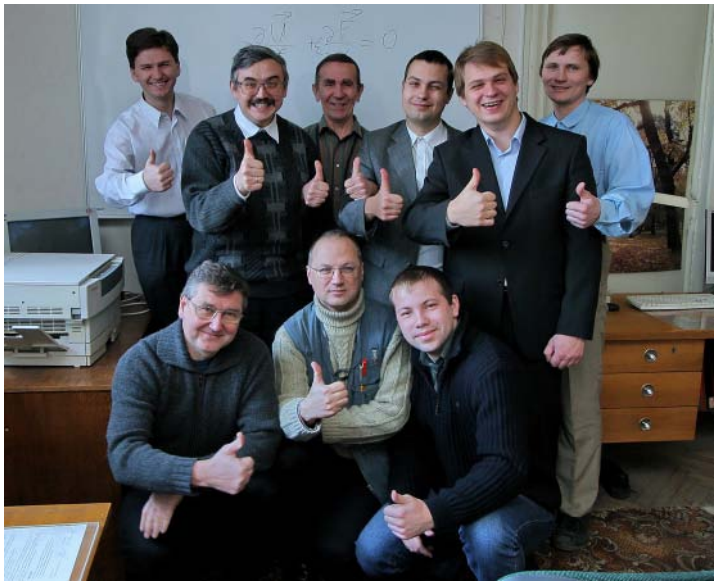
Пакет EWT-ЦАГИ

Михайлов С.В.

Кто мы, где мы

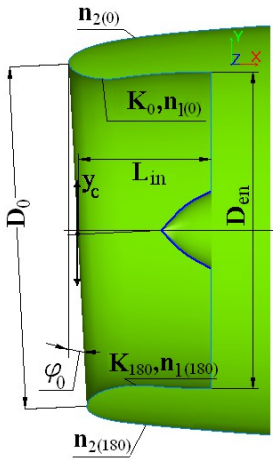
ЦАГИ
Отделение Силовых Установок
Отдел CFD (Босняков С.М.)

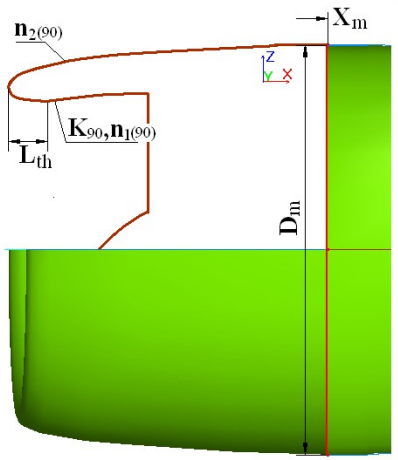
МФТИ, ФАЛТ
Кафедра КМ (Босняков С.М.)
Кафедра информатики

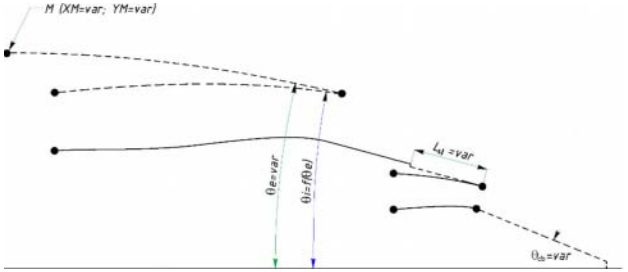


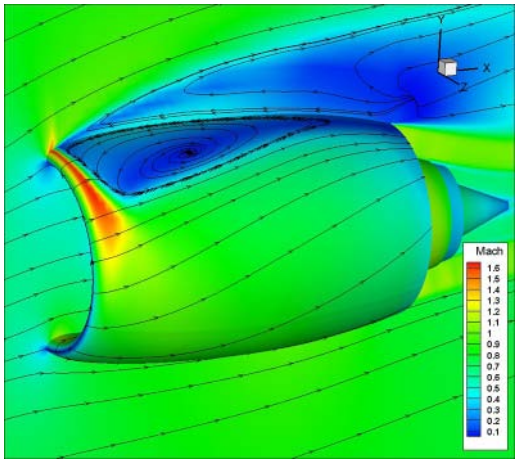
Зленко Н.А.,	1971
Босняков С.М.,	1975
Михайлов С.В.,	1980
Матяш С.В.,	1988
Морозов А.Н.,	1988
Власенко В.В.,	1990
Енгулатова М.Ф.,	1991
Лысенков А.В.,	2001
Курсаков И.А.,	2006
Кажан Е.В.,	2006
Савельев А.В.,	2009
Ширяева А.В.,	2009
Анисимов К.В.,	2010
Подаруев В.Ю.,	2010
Трошин А.И.,	2011
Павлик С.В.,	2013
Матяш Е.С.,	2014
Молев С.С.,	2014

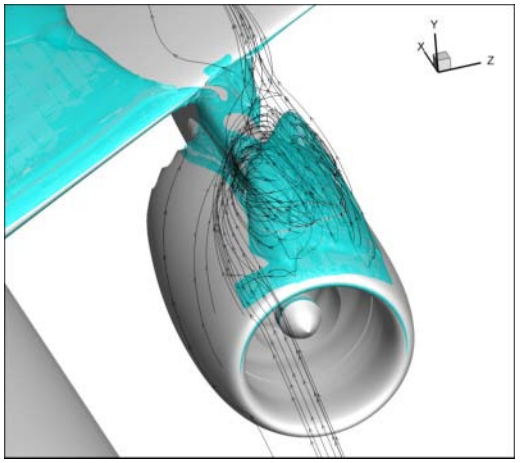
Оптимизация мотогондолы двигателя

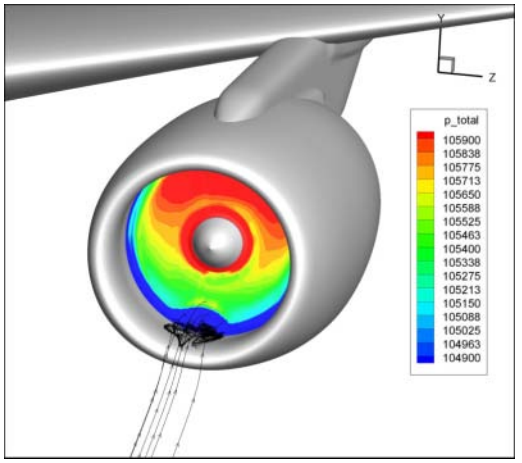




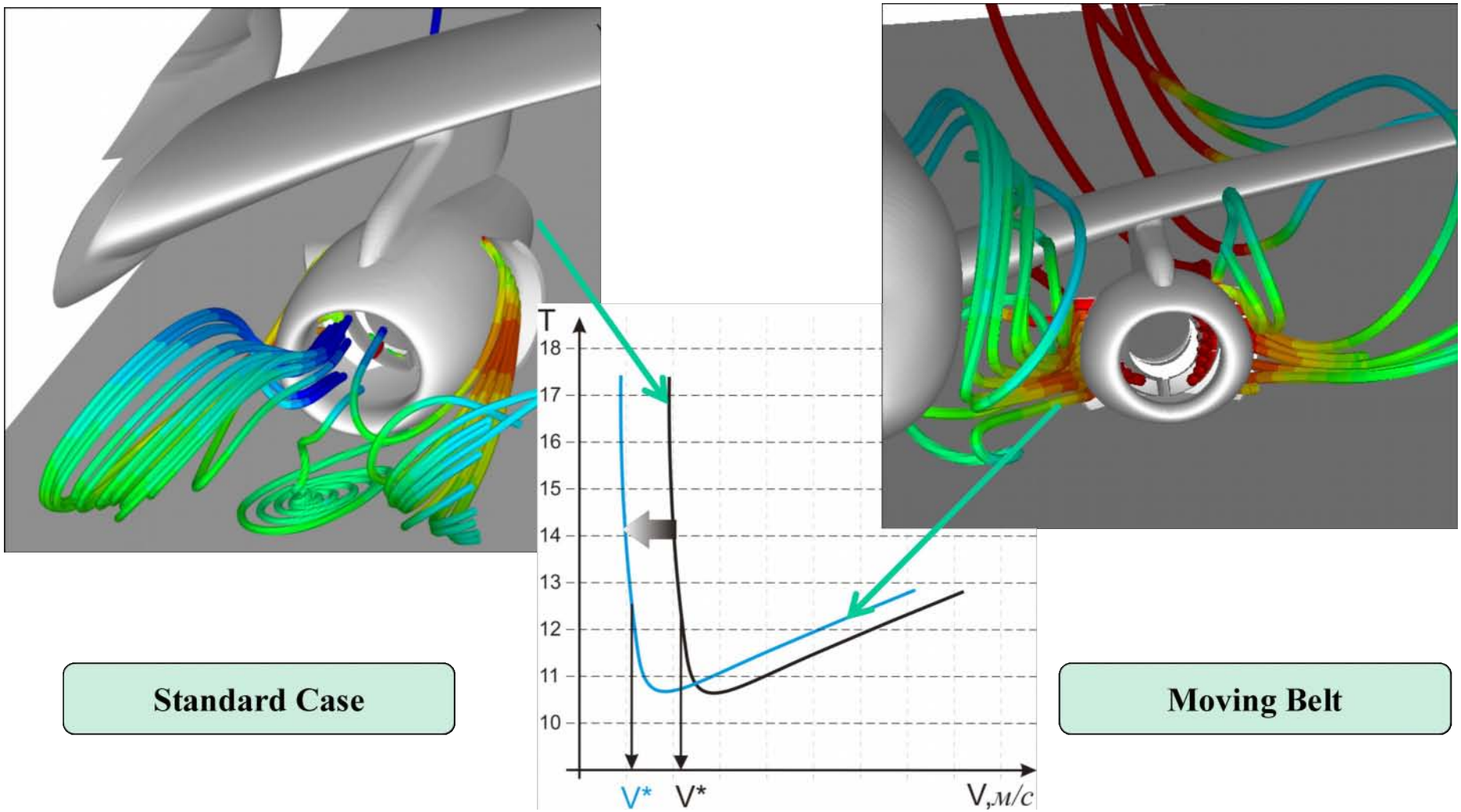




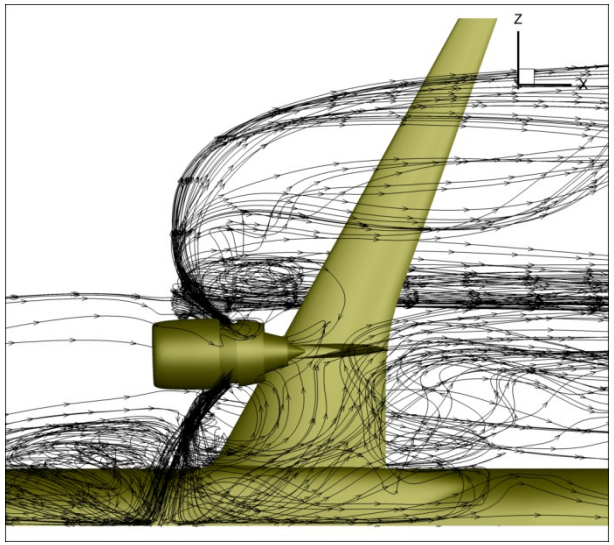


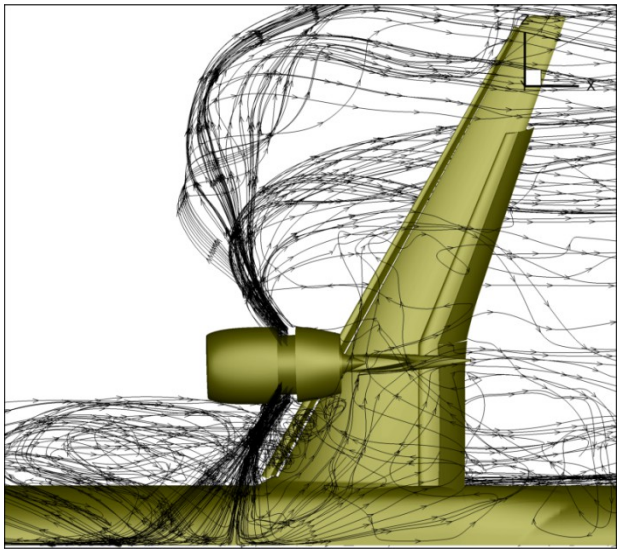


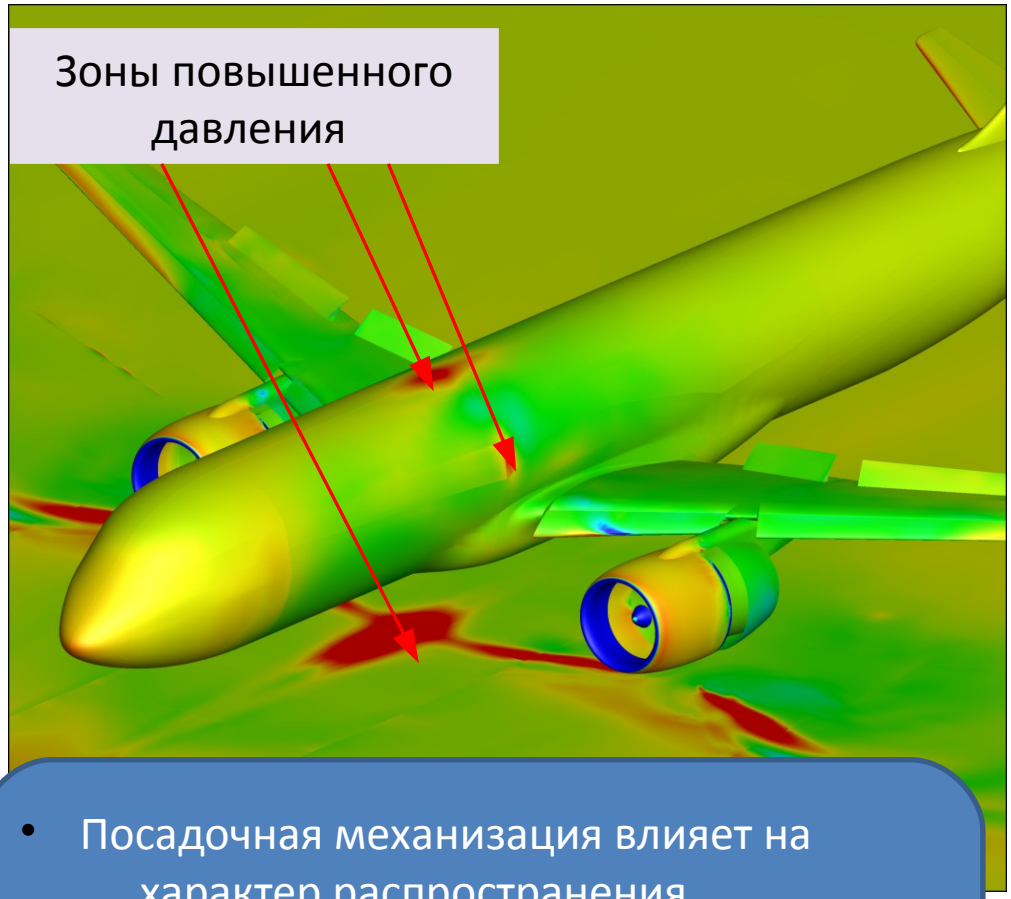
Влияние бегущей дорожки на скорость отсечки при реверсе



Влияние механизации крыла на реверс

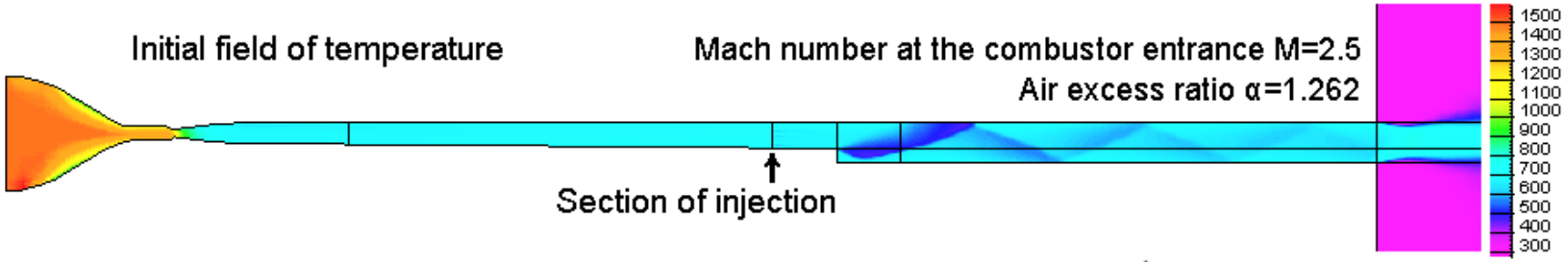


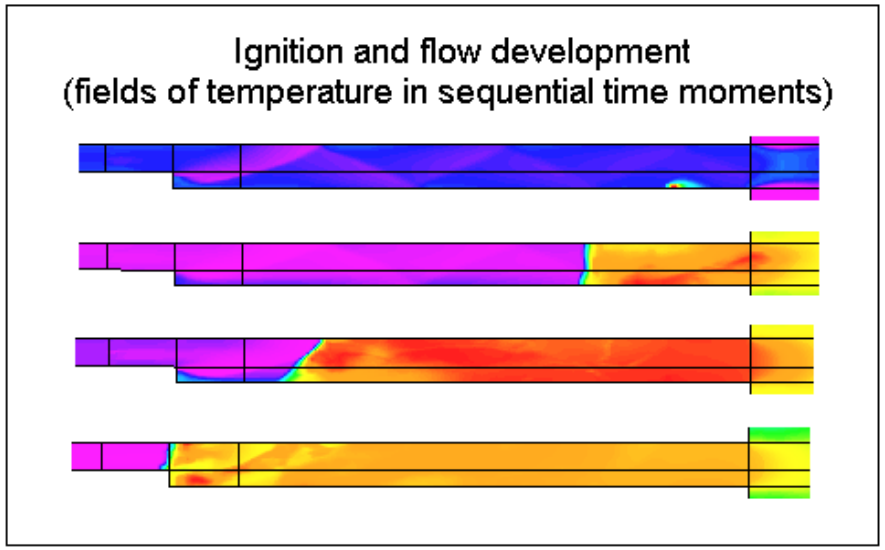


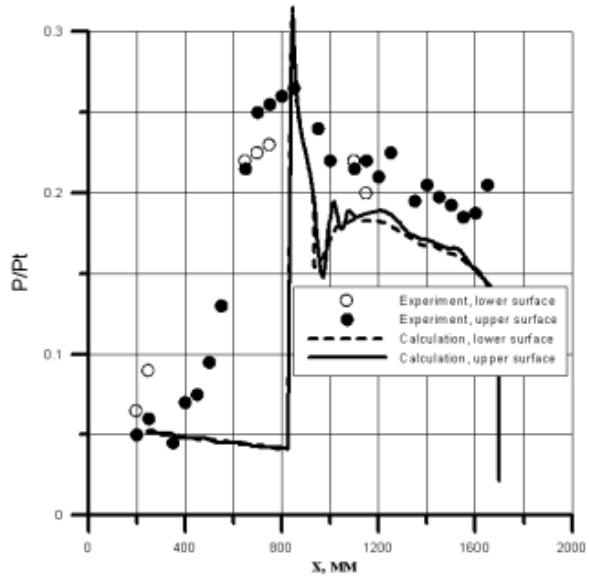


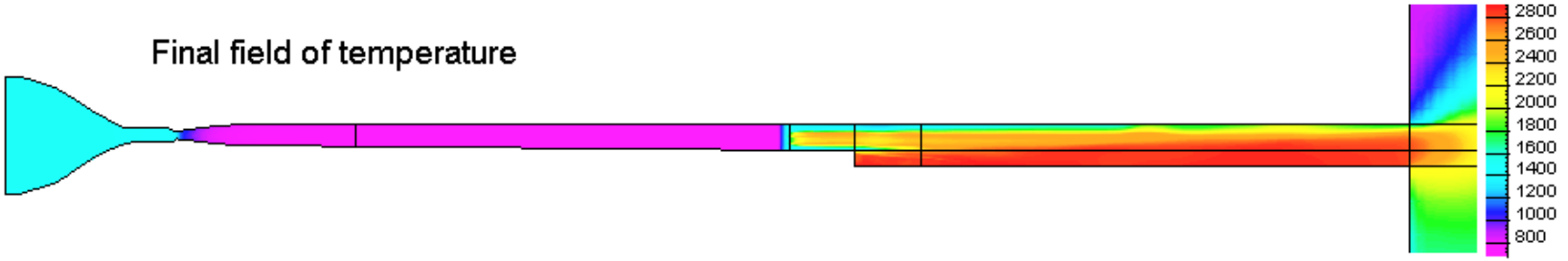
- Посадочная механизация влияет на характер распространения реверсивных струй
- Расчеты с учетом механизации позволяют определить зоны повышенных нагрузок

Численное моделирование рабочего процесса в модели в камере сгорания









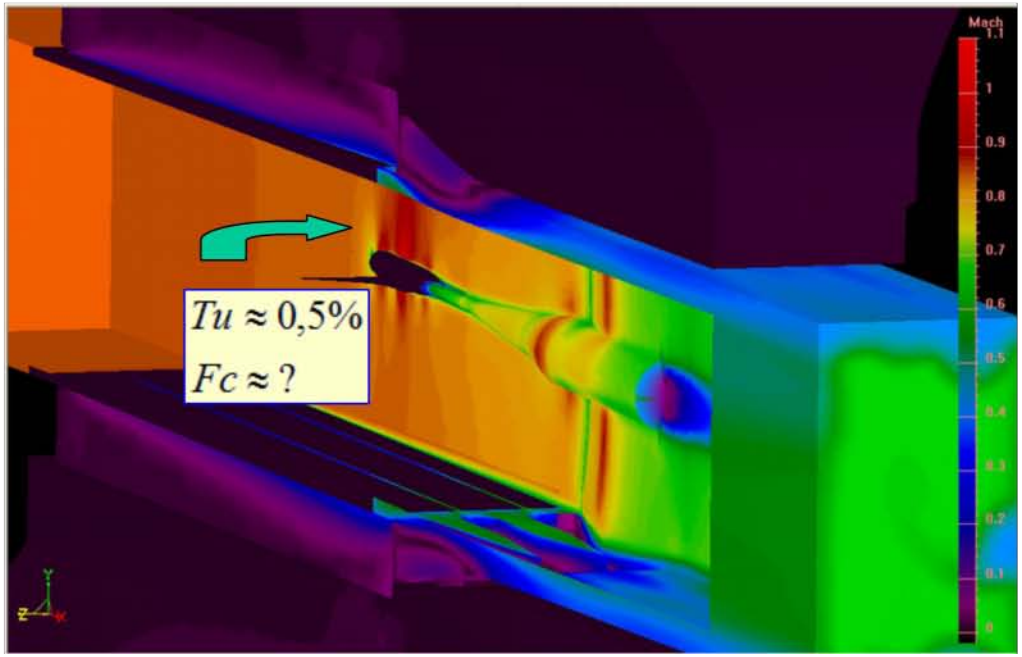
Моделирование АДТ

T-128

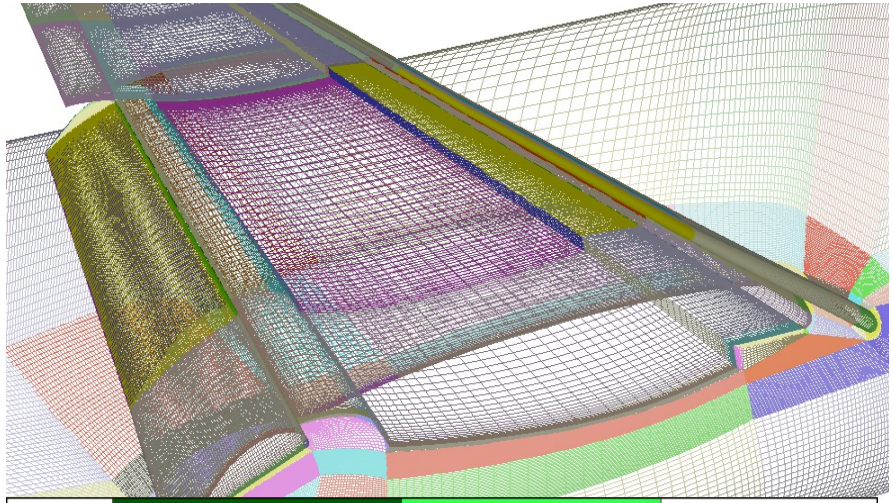
Стенки АДТ и градиенты порождают турбулентность

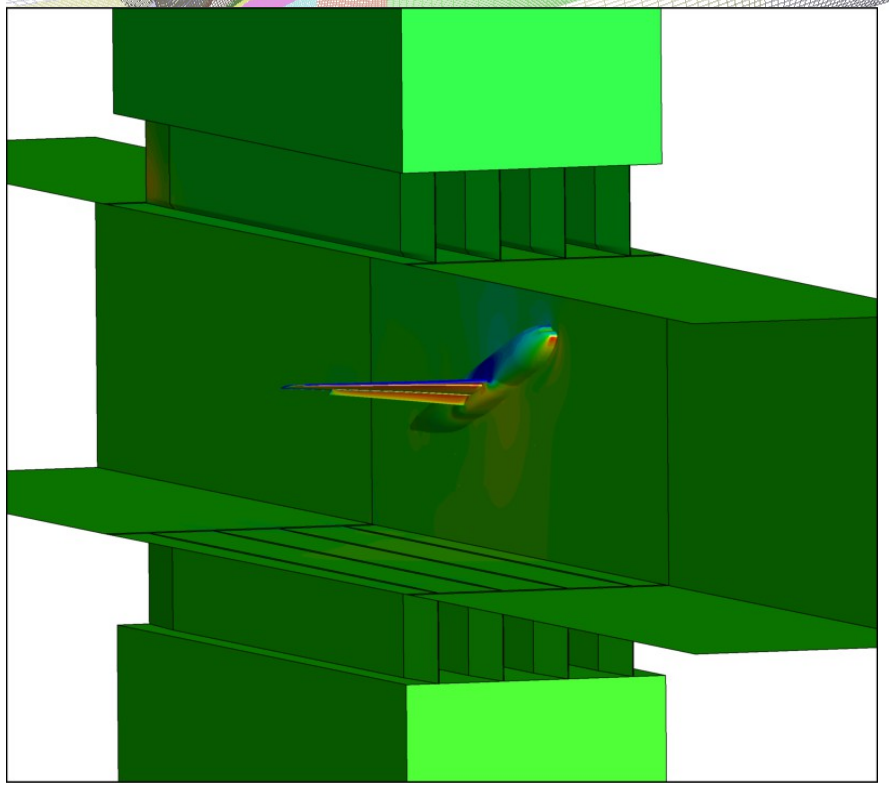
Параметры турбулентности необходимо задавать в месте установки модели

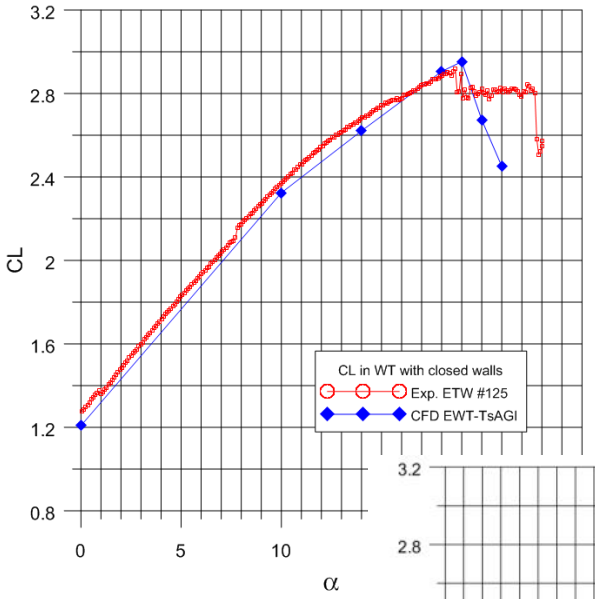
ETW

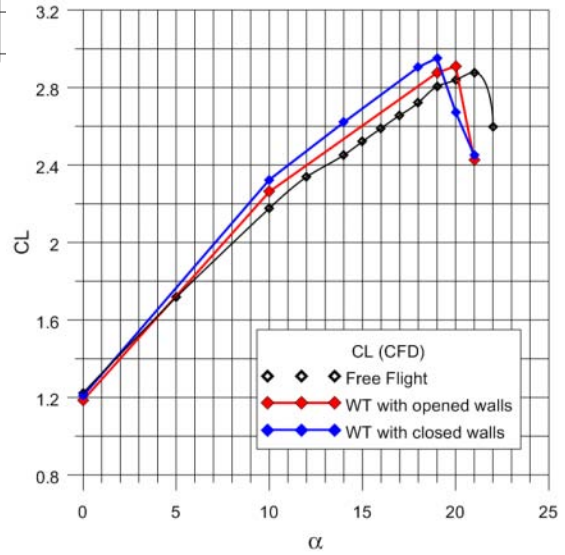


Трехзвенное крыло



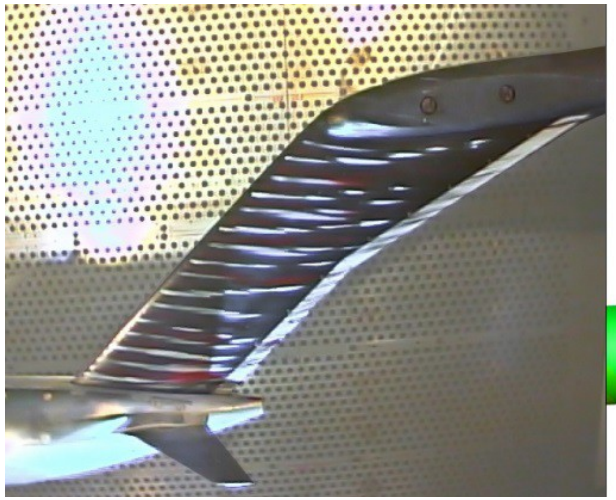


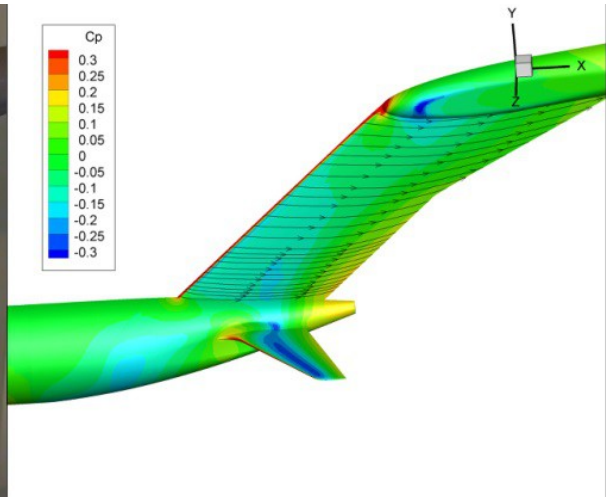




Получены принципиально новые количественные оценки поправок влияния стенок АДТ на Cy_max

Поправка на наличие протока между килевой державкой и имитатором хвостовой

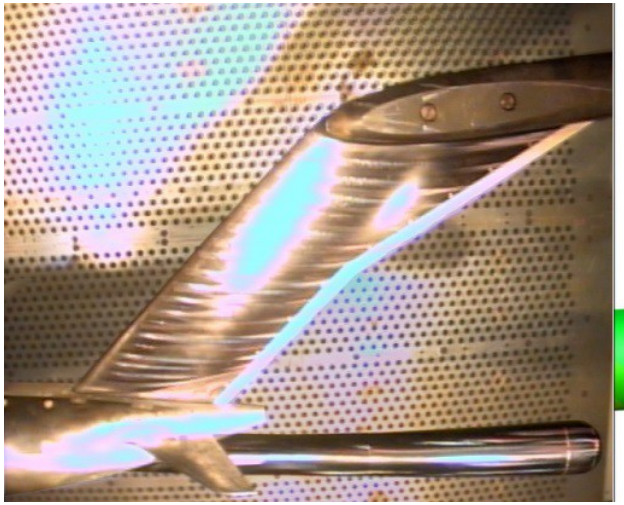


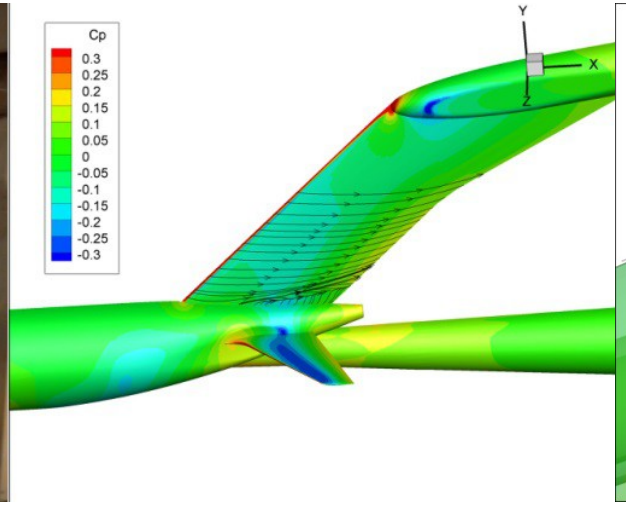


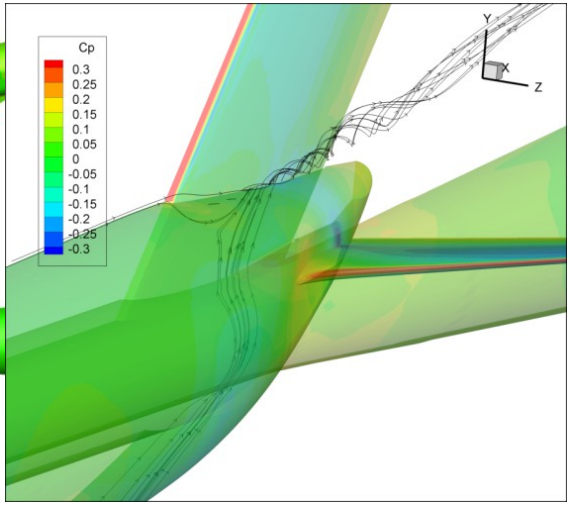
$\Delta_2 Cya \sim 0.0006$

$\Delta_2 Cxa \sim -0.007$

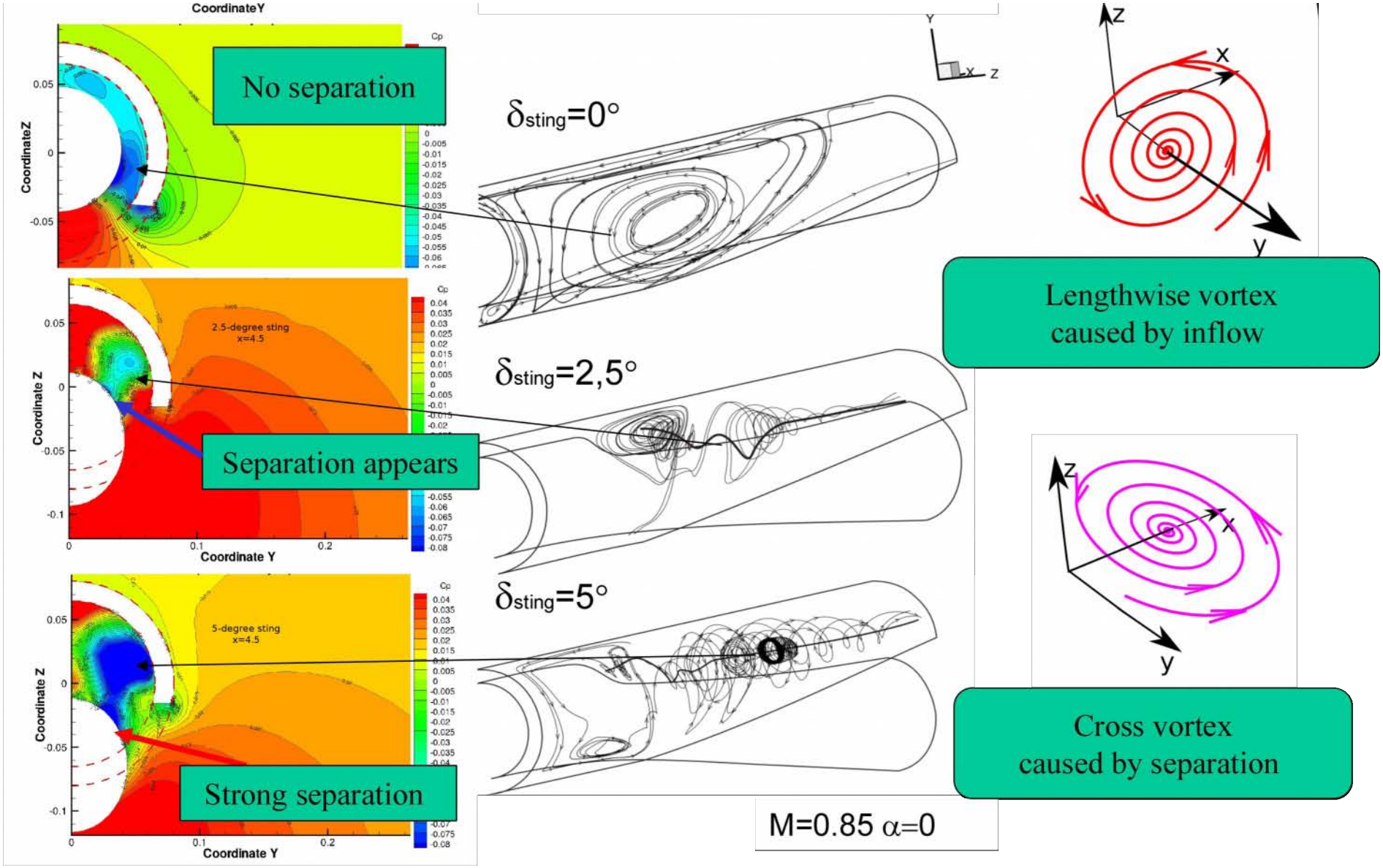
$\Delta_2 mza \sim 0.03$



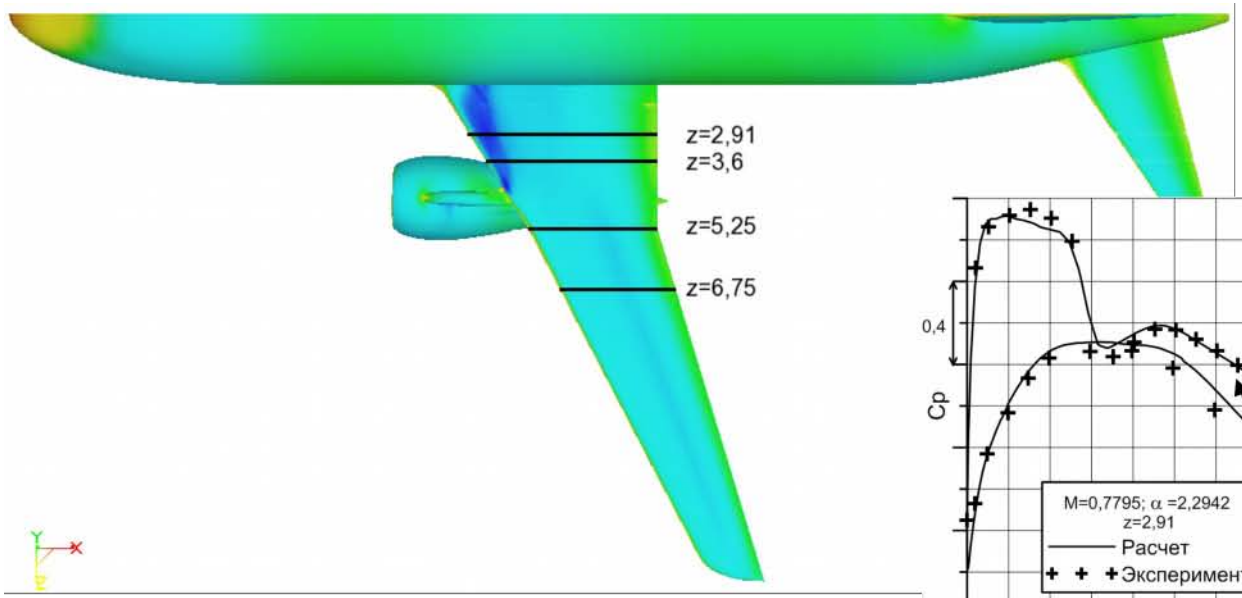




Flow structure inside the cavern

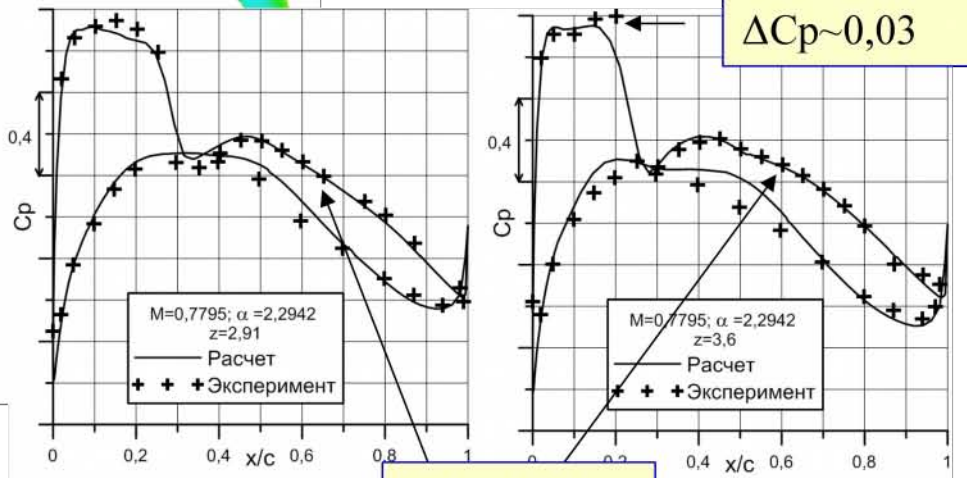


Euro Test Case Cruiser Configuration

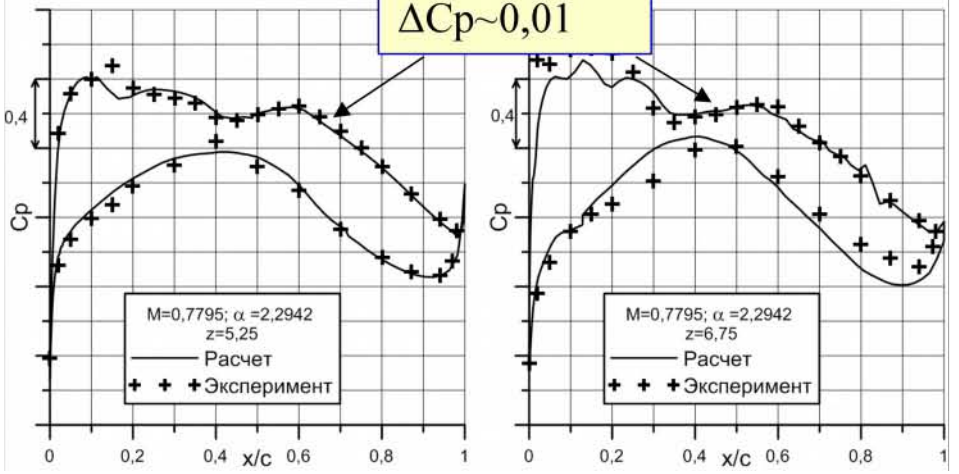


M=0,7795 α=2,2942°

ΔCp~0,03



ΔCp~0,01



Quality of results are inside of Technical Requirements

Method

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}_i}{\partial x_i} = \vec{W}$$

$$\vec{U}^{n+1}_{i,j,k} = \vec{U}^{n}_{i,j,k} - \frac{\tau^n}{V_{i,j,k}} \cdot \left[\left(\vec{F}_{i+1/2} - \vec{F}_{i-1/2} \right) + \left(\vec{F}_{j+1/2} - \vec{F}_{j+1/2} \right) + \left(\vec{F}_{k+1/2} - \vec{F}_{k+1/2} \right) \right] + \tau^n \vec{W}_{i,j,k} .$$

Finite Volume Method;

Hexahedral cells;

Godunov-type scheme for the approximation of **convective fluxes;**

Central-difference approximation for the **diffusion fluxes**;

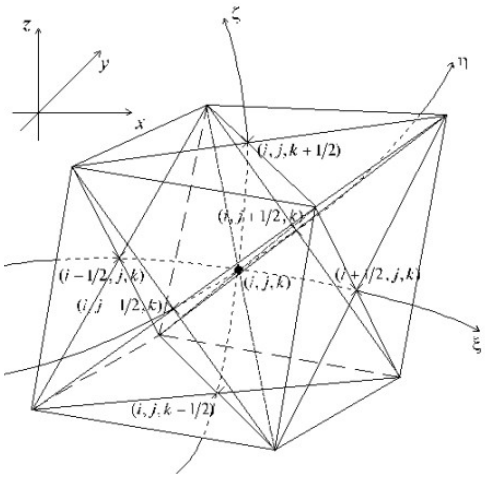
Semi-implicit approximation **for source terms** in equations for turbulence parameters;

Godunov or Modified Roe solution for Riemann problem

Kolgan, MUSCL or Roe TVD reconstruction

Vector ‘minmod’ for velosity

WENO5, WENO7, WENO9 reconstruction



Computational cell and map of indexes

Method

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}_i}{\partial x_i} = \vec{W}$$

$$\vec{U}^{n+1}_{i,j,k} = \vec{U}^{n}_{i,j,k} - \frac{\tau^n}{V_{i,j,k}} \cdot \left[\left(\vec{F}_{i+1/2} - \vec{F}_{i-1/2} \right) + \left(\vec{F}_{j+1/2} - \vec{F}_{j+1/2} \right) + \left(\vec{F}_{k+1/2} - \vec{F}_{k+1/2} \right) \right] + \tau^n \vec{W}_{i,j,k}.$$

- Годунов С.К., Забродин А.В., Иванов М.Я., Крайко А.Н., Прокопов Г.П. Численное решение многомерных задач газовой динамики. — М., Наука, 1976.
- Колган В.П. Применение принципа минимальных значений производной к построению конечно-разностных схем для расчета разрывных решений газовой динамики. // Ученые Записки ЦАГИ, 1972, т.3, №6.
- Leer van B. Towards the ultimate conservative difference scheme. Part V; A second order sequel to Godunov's method // J. of Computational Physics. 1979. V. 32, N 1.
- Куликовский А.Г., Погорелов Н.В., Семенов А.Ю. Математичечкие вопросы численного решения гиперболических систем уравнений. М., "Физмалит", 2001.
- Родионов А.В. Монотонная схема второго порядка аппроксимации для маршевых расчетов неравновесных потоков. // ЖВМ и МФ. 1987, т.27, №4.
- Власенко В.В. О математическом подходе и принципах построения численных методологий для пакета прикладных программ EWT-ЦАГИ. // Труды ЦАГИ, выпуск 2671, с.20-85, 2007.

Method

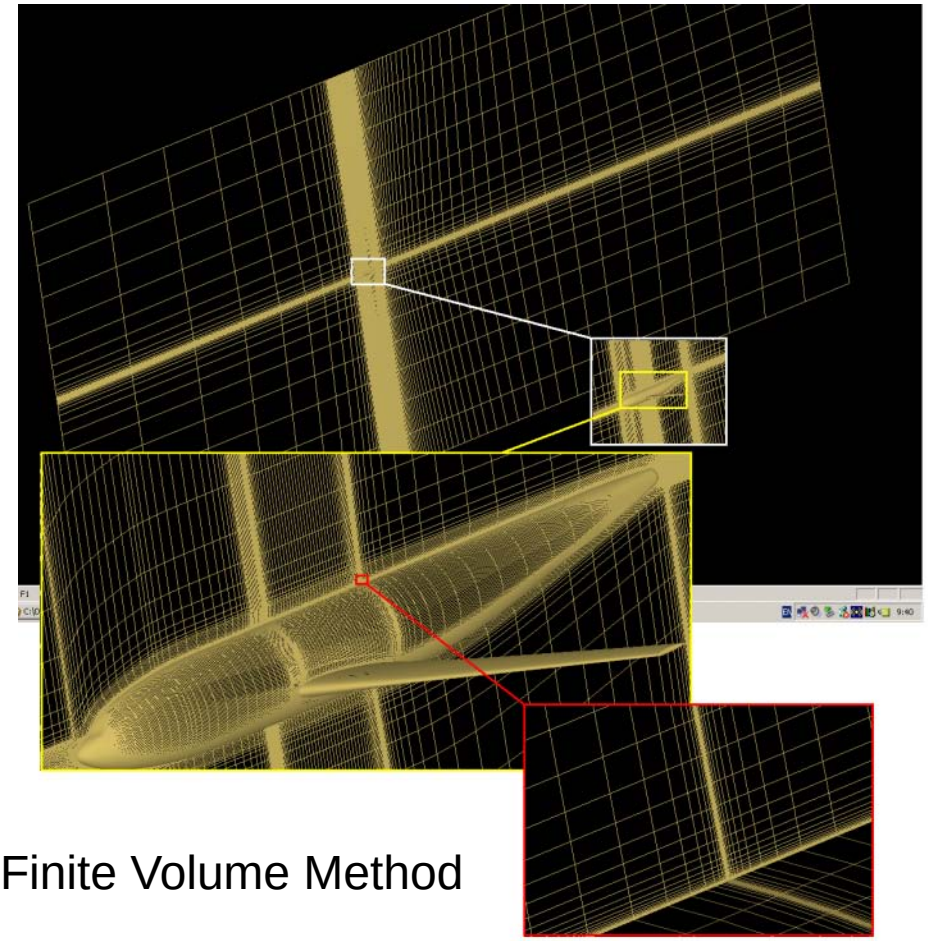
$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}_i}{\partial x_i} = \vec{W}$$

$$\vec{U}^{n+1}_{i,j,k} = \vec{U}^{n}_{i,j,k} - \frac{\tau^n}{V_{i,j,k}} \cdot \left[\left(\vec{F}_{i+1/2} - \vec{F}_{i-1/2} \right) + \left(\vec{F}_{j+1/2} - \vec{F}_{j+1/2} \right) + \left(\vec{F}_{k+1/2} - \vec{F}_{k+1/2} \right) \right] + \tau^n \vec{W}_{i,j,k}$$

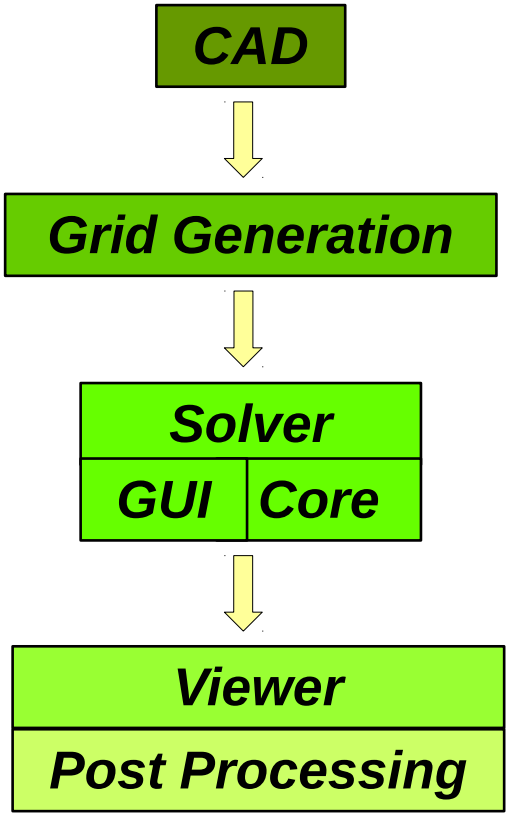
- Coakley T.J. Turbulence modeling methods for the compressible Navier-Stokes equations // AIAA-83-1693. 1983.
- Coakley T.J., Hsieh T. Comparison between implicit and hybrid methods for the calculation of steady and unsteady inlet flows // AIAA-85-1125. 1985.
- Vuong S.T., Coakley T.J. Modeling of turbulence for hypersonic flows with and without separation // AIAA-87-0286. 1987.
- Menter, F.R., Langtry, R.B., Likki, S.R., Suzen, Y.B., Huang, P.G., and Völker, S., “A Correlation based Transition Model using Local Variables Part 1, Part 2- Model Formulation” ASME-GT2004-53452, ASME-GT2004-53454 ASME TURBO EXPO 2004, Vienna, Austria.;
- Spalart P.R., Allmaras S.R. A one-equation turbulence model for aerodynamic flows // AIAA Paper 92-439, Reno, NV. 1992..
- Власенко В.В. О математическом подходе и принципах построения численных методологий для пакета прикладных программ EWT-ЦАГИ. // Труды ЦАГИ, выпуск 2671, с.20-85, 2007.

CFD cycle

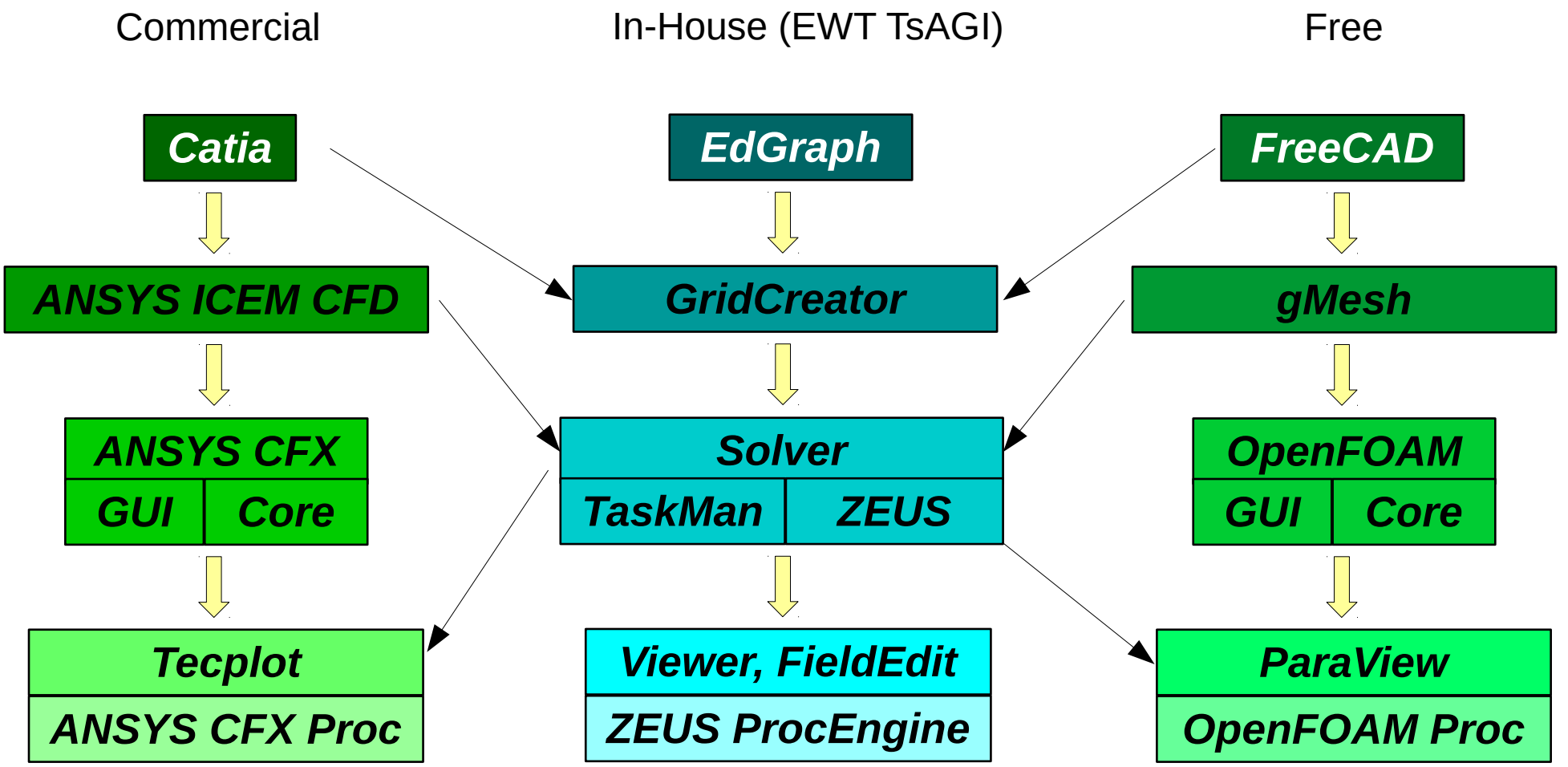
$$\frac{\partial \vec{U}}{\partial t} + \sum \frac{\partial \vec{F}_k}{\partial x_k} = W$$



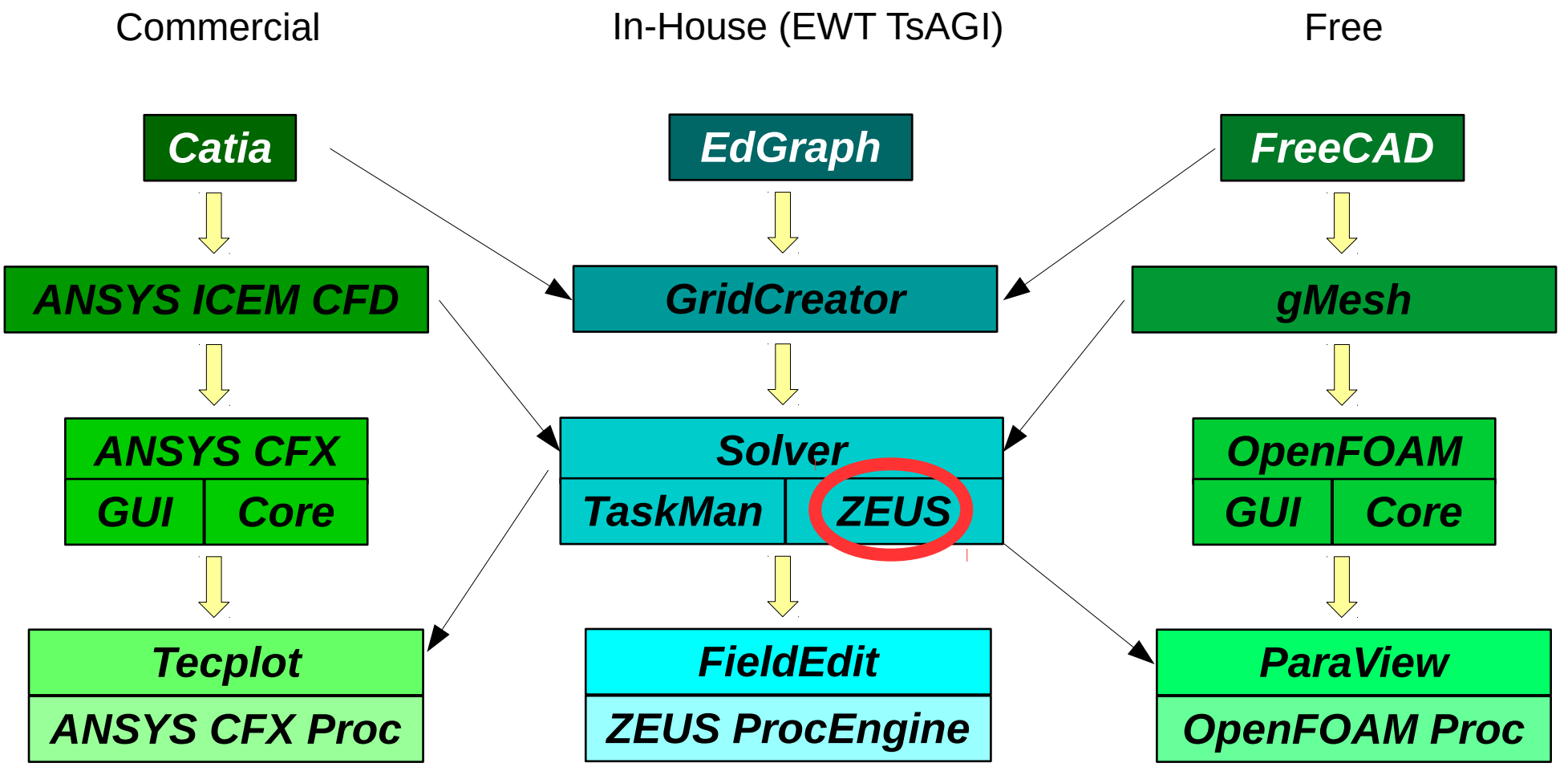
Finite Volume Method



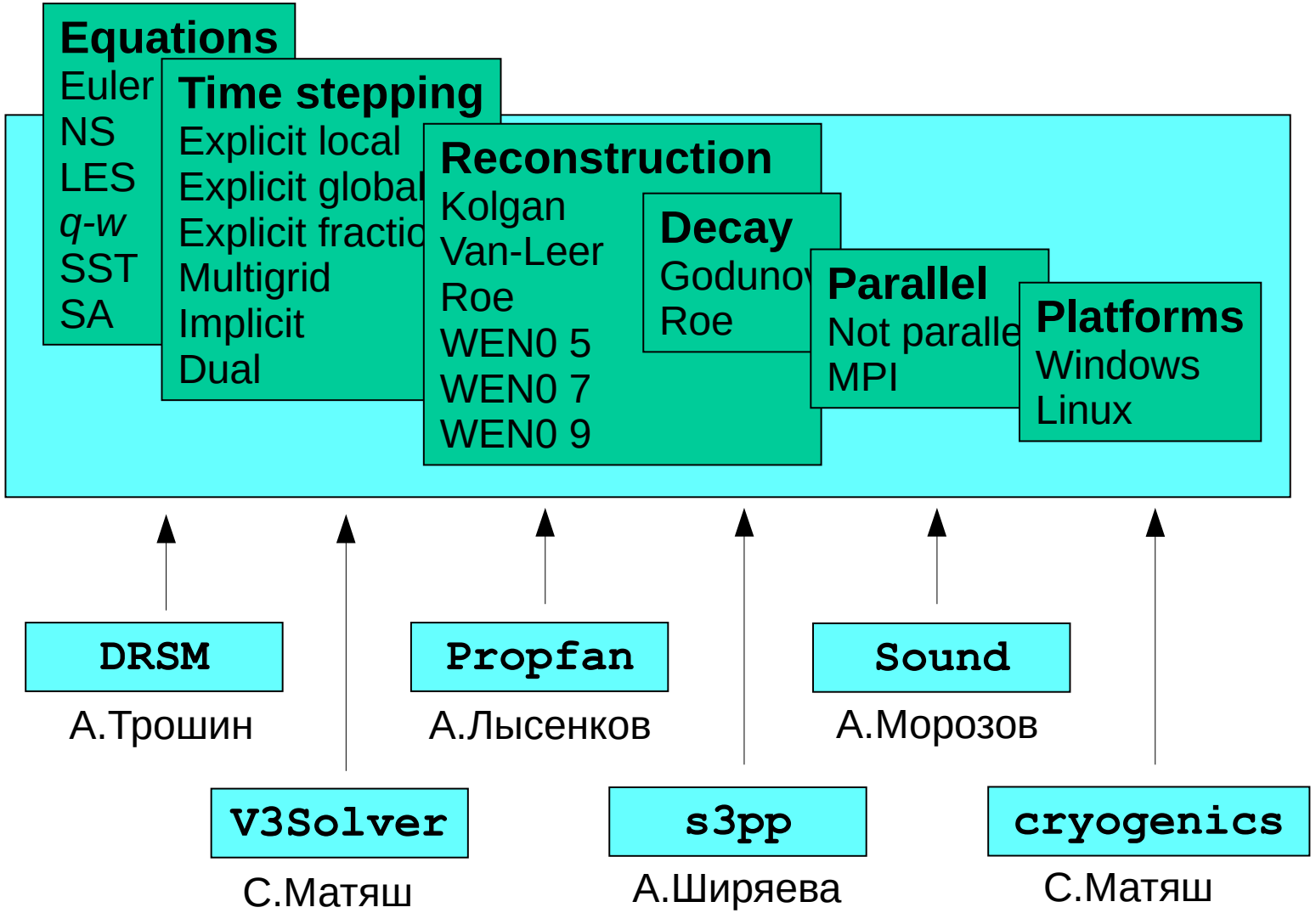
CFD cycle


Commercial
In-House (EWT TsAGI)
Free
Catia
EdGraph
FreeCAD
ANSYS ICEM CFD
GridCreator
gMesh
ANSYS CFX
GUI
Core
Solver
TaskMan
ZEUS
OpenFOAM
GUI
Core
Tecplot
ANSYS CFX Proc
Viewer, FieldEdit
ZEUS ProcEngine
ParaView
OpenFOAM Proc

CFD cycle


Commercial
In-House (EWT TsAGI)
Free
Catia
EdGraph
FreeCAD
ANSYS ICEM CFD
GridCreator
gMesh
ANSYS CFX
GUI
Core
Solver
TaskMan
ZEUS
OpenFOAM
GUI
Core
Tecplot
ANSYS CFX Proc
FieldEdit
ZEUS ProcEngine
ParaView
OpenFOAM Proc

ZEUS



С.Михайлов

В.Власенко

С.Матяш

Е.Кажан

И.Курсаков

А.Савельев

А.Трошин

Е.Матяш

С.Молев

В.Подаруев

Equations

From coder point of view

From Euler to Navier-Stokes Equations

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}_i}{\partial x_i} = \vec{W}$$

Euler Equations

$$\vec{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix}, \vec{F} = \begin{bmatrix} Q \\ Qu + ps_x \\ Qv + ps_y \\ Qw + ps_z \\ Q \cdot (E + p/\rho) \end{bmatrix}, \vec{W} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$Q = \rho\left(us_x + vs_y + ws_z\right)$$

$$p = \rho RT$$

$$E = \frac{u^2 + v^2 + w^2}{2} + \frac{RT}{\gamma - 1}$$

Navier-Stokes Equations

$$\vec{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix}, \vec{F} = \begin{bmatrix} Q \\ Qu + ps_x + \tau_{xn} \\ Qv + ps_y + \tau_{yn} \\ Qw + ps_z + \tau_{zn} \\ Q \cdot (E + p/\rho) + (\tau_{xn}u + \tau_{yn}v + \tau_{zn}w) + q_n \end{bmatrix}, \vec{W} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

From Navier-Stokes Equations to RANS

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}_i}{\partial x_i} = \vec{W}$$

Navier-Stokes Equations

$$\vec{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix}, \quad \vec{F} = \begin{bmatrix} Q \\ Qu + ps_x + \tau_{xn} \\ Qv + ps_y + \tau_{yn} \\ Qw + ps_z + \tau_{zn} \\ Q \cdot (E + p/\rho) + (\tau_{xn} u + \tau_{yn} v + \tau_{zn} w) + q_n \end{bmatrix}, \quad \vec{W} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$Q = \rho \left(u s_x + v s_y + w s_z \right)$$

$$p = \rho R T$$

$$E = \frac{u^2 + v^2 + w^2}{2} + \frac{RT}{\gamma - 1}$$

RANS, *q-w*

$$\vec{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \\ \rho q \\ \rho \omega \end{bmatrix}, \quad \vec{F} = \begin{bmatrix} Q \\ Qu + ps_x + I_{xn} \\ Qv + ps_y + I_{yn} \\ Qw + ps_z + I_{zn} \\ Q \cdot (E + p/\rho) + (I_{xn} u + I_{yn} v + I_{zn} w) + \theta_n + T_n^k \\ Qq + T_n^q \\ Q\omega + T_n^\omega \end{bmatrix}, \quad \vec{W} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ S(q) \\ S(\omega) \end{bmatrix}$$

$$E = \frac{u^2 + v^2 + w^2}{2} + \frac{RT}{\gamma - 1} + q^2$$

$$I_{in} = \tau_{in} + \rho R_{in}$$

$$\theta_n = q_n + \rho \sigma_n$$

From RANS to Chemistry

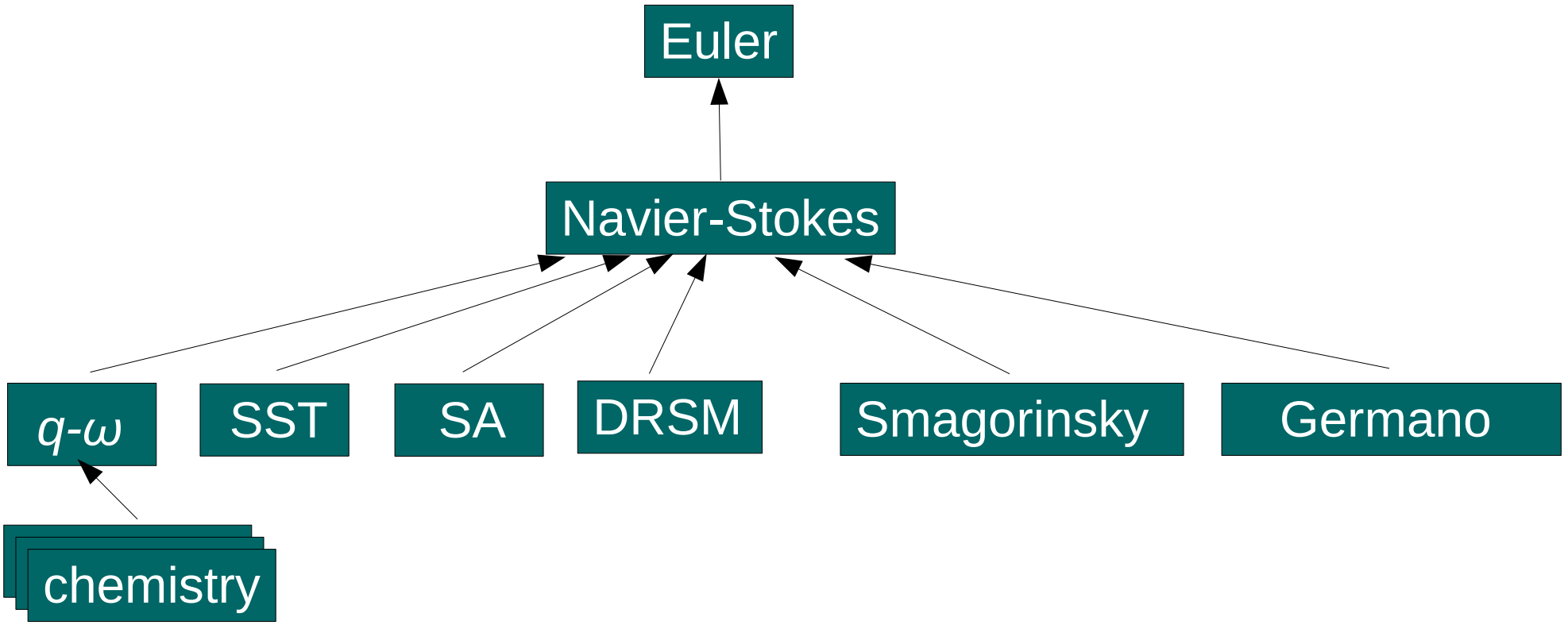
RANS, *q-w*

$$\vec{U}=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ \rho E\\ \rho q\\ \rho \omega\end{bmatrix},\ \vec{F}=\begin{bmatrix}Q\\ Qu+ps_x+I_{xn}\\ Qv+ps_y+I_{yn}\\ Qw+ps_z+I_{zn}\\ Q\cdot(E+p/\rho)+(I_{xn}u+I_{yn}v+I_{zn}w)+\theta_n+T_n^k\\ Qq+T_n^q\\ Q\omega+T_n^\omega\end{bmatrix},\ \vec{W}=\begin{bmatrix}0\\ 0\\ 0\\ 0\\ 0\\ S(q)\\ S(\omega)\end{bmatrix}$$

Chemistry

$$\vec{U}=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ \rho E\\ \rho q\\ \rho \omega\\ \rho Y_k\end{bmatrix},\ \vec{F}=\begin{bmatrix}Q\\ Qu+ps_x+I_{xn}\\ Qv+ps_y+I_{yn}\\ Qw+ps_z+I_{zn}\\ Q\cdot(E+p/\rho)+(I_{xn}u+I_{yn}v+I_{zn}w)+\theta_n+T_n^k\\ Qq+T_n^q\\ Q\omega+T_n^\omega\\ \rho Y_k u_i+J_i(Y_k)\end{bmatrix},\ \vec{W}=\begin{bmatrix}0\\ 0\\ 0\\ 0\\ 0\\ S(q)\\ S(\omega)\\ S_k^{chem}\end{bmatrix}$$

Equations inheritance



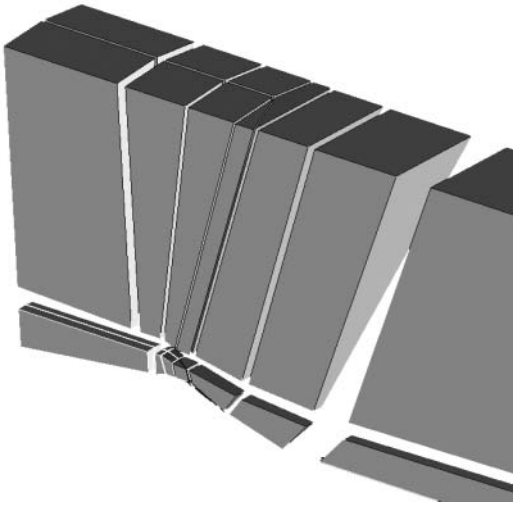
We can draw inheritance diagram for gasdynamics equations

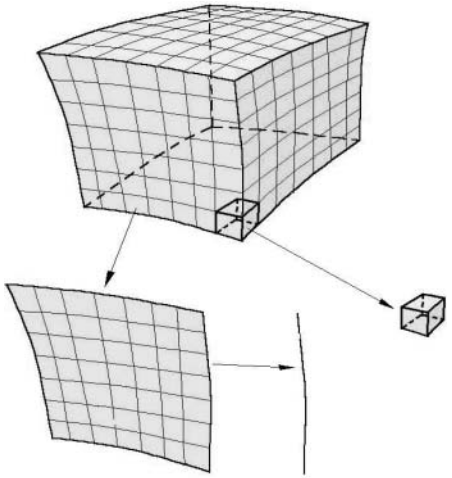
Scheme

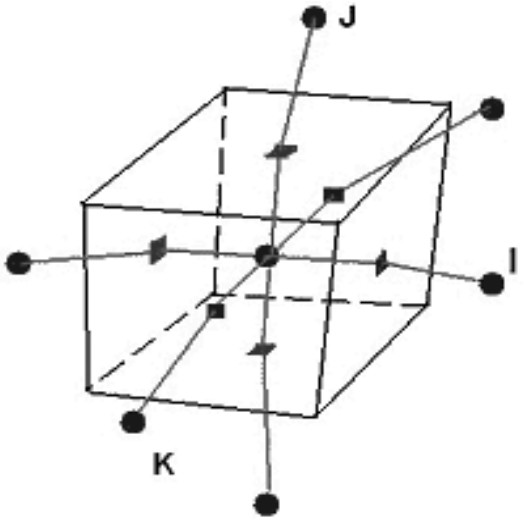
Finite Volume Method
Topology, Scheme

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}_i}{\partial x_i} = \vec{W}$$

$$\vec{U}_{i,j,k}^{n+1} = \vec{U}_{i,j,k}^{n} - \frac{\tau^n}{V_{i,j,k}} \cdot \left[\left(\vec{F}_{i+1/2} - \vec{F}_{i-1/2} \right) + \left(\vec{F}_{j+1/2} - \vec{F}_{j+1/2} \right) + \left(\vec{F}_{k+1/2} - \vec{F}_{k+1/2} \right) \right] + \tau^n \vec{W}_{i,j,k}.$$





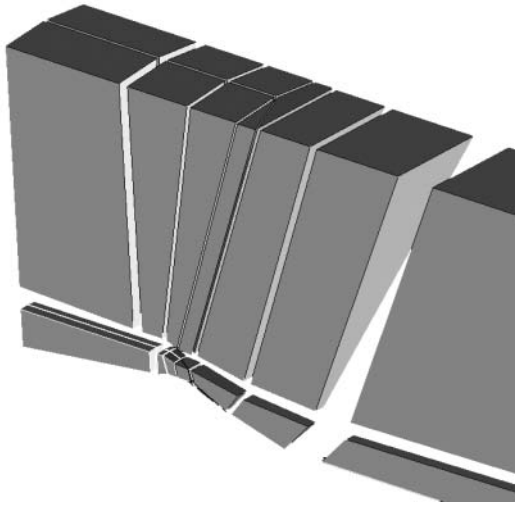


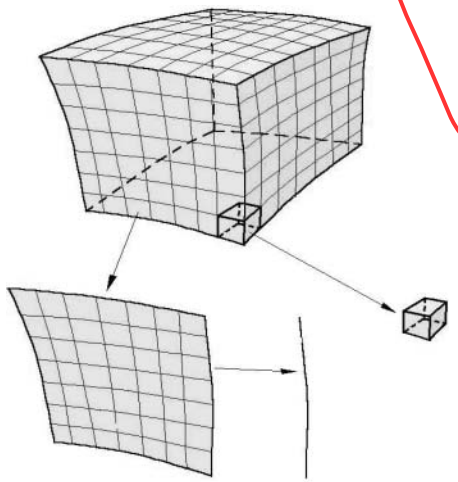
Finite Volume Method
Topology, Scheme

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}_i}{\partial x_i} = \vec{W}$$

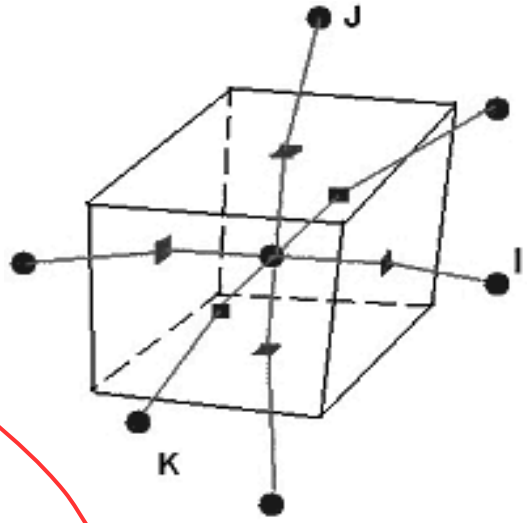
$$\vec{U}_{i,j,k}^{n+1} = \vec{U}_{i,j,k}^{n} - \frac{\tau^n}{V_{i,j,k}} \cdot \left[\left(\vec{F}_{i+1/2} - \vec{F}_{i-1/2} \right) + \left(\vec{F}_{j+1/2} - \vec{F}_{j+1/2} \right) + \left(\vec{F}_{k+1/2} - \vec{F}_{k+1/2} \right) \right] + \tau^n \vec{W}_{i,j,k}.$$

Topology is the same for different schemes





Scheme is written locally for separate cell and its neighbors



Finite Volume Method
Topology, Scheme

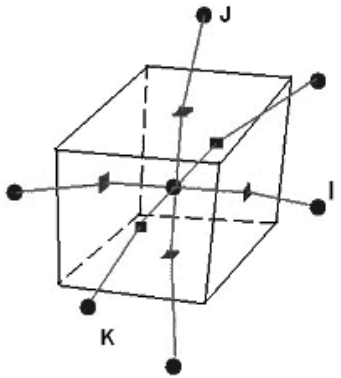
Cell for topology

zCell

Pointers to the adjacent cells

Pointers to the adjacent sides

Pointers to the coordinates



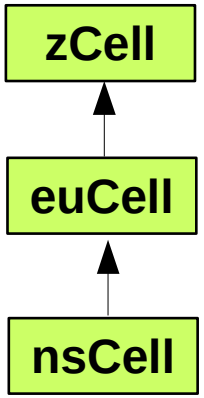
Cell for **xx** Scheme

xxCell

Geometric parameters

Gasdynamic parameters

Gradients



Side for topology

zSide

Pointers to the adjacent cells

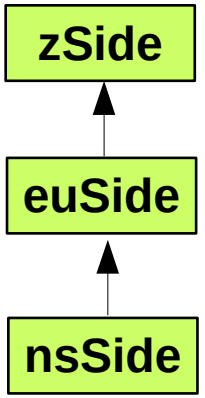
Pointers to the coordinates

Side for **xx** Scheme

xxSide

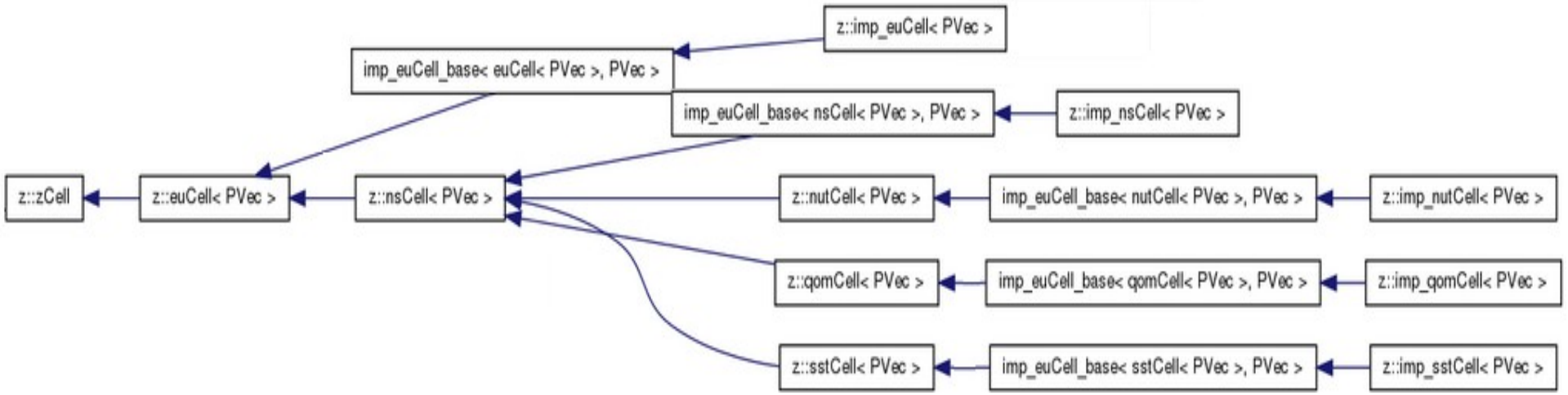
Geometric parameters

Gasdynamic parameters



Cells inheritance diagram

Imp Euler Imp NS



Topology Euler NS RANS Imp RANS

Entities

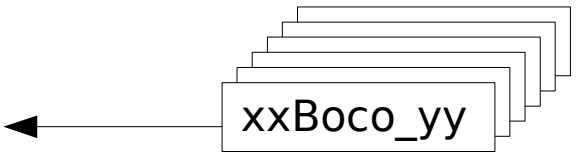
zSolver

Topology

- **zCell** Topology (pointers of adjacent cells and sides)
- **zSide** Topology (pointers of adjacent cells)
- zMesh (array of nodes)
- zCellsArray (array of cells type zCell)
- zSidesArray (array of sides type zSidel)
- zPatch (array of pointers to the boundary cells)
- zBlock (include zMesh, zCellsArray, zSideArray, zPatch)
- zRank (include set of zBlock for parallel computations)
- zTransferCells (cells for blocks joining)

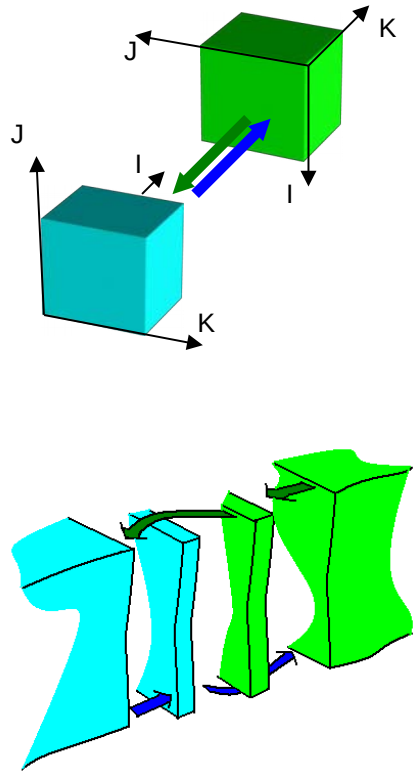
Scheme

- xxData (coordinates,P,U)
- **xxCell** (volume, size, gasdynamics parameters, gradients…)
- **xxSide** (area, fluxes,…)
- xxEquations
- xxScheme (scheme functions)
- xxBoco (boundary condition functions)

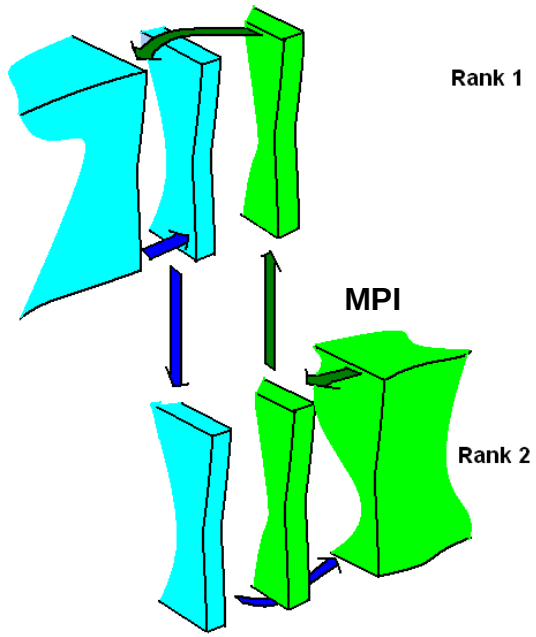


Using shared and distributed memory

Shared memory



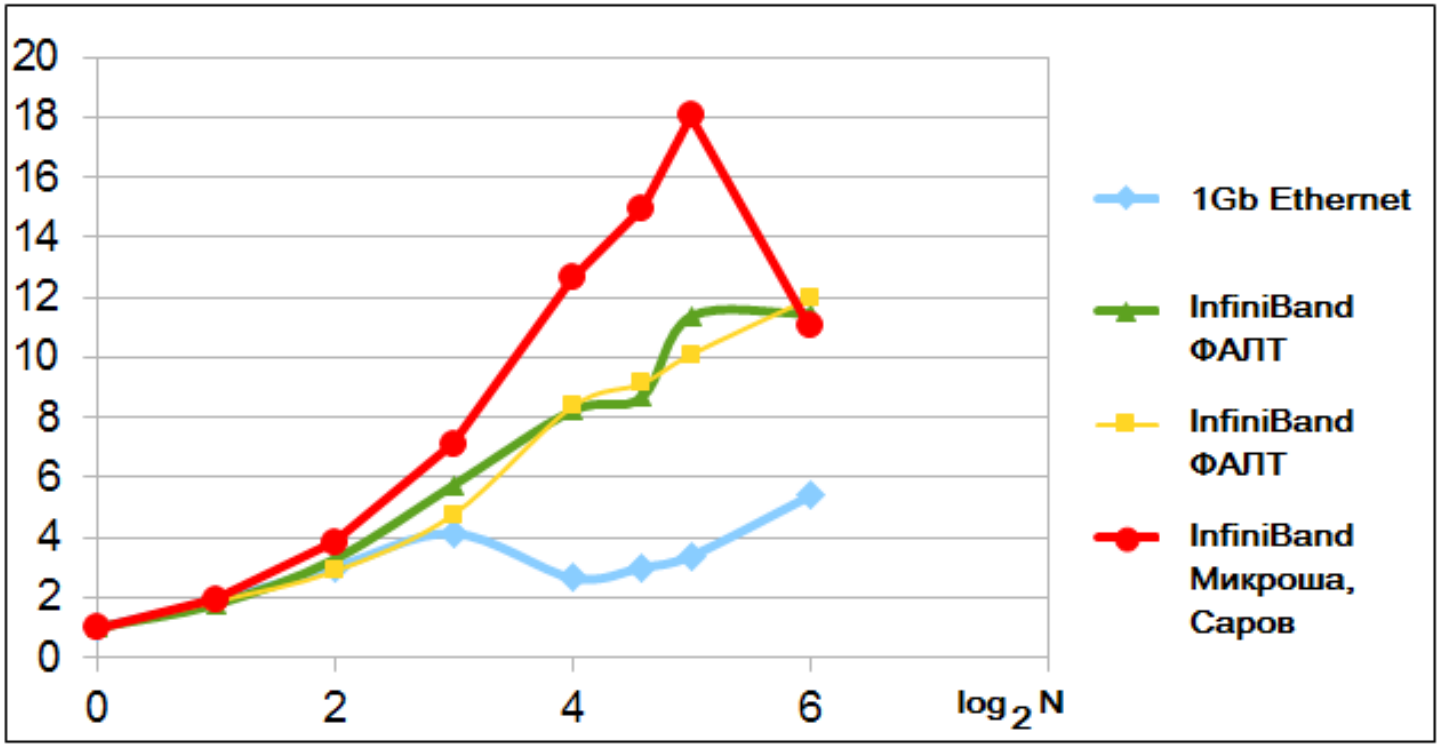
Distributed memory



Exchange table

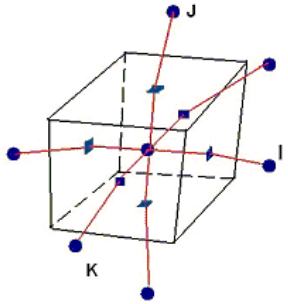
rank	block	side	size	rank	block	side	size
0	0	0	…	1	0	1	…
0	0	1	…	1	0	0	…
2	2	4	…	4	2	1	…
…	…	…	…	…	…	…	…
1	0	1	…	0	0	0	…
1	0	0	…	0	0	1	…

Using shared and distributed memory

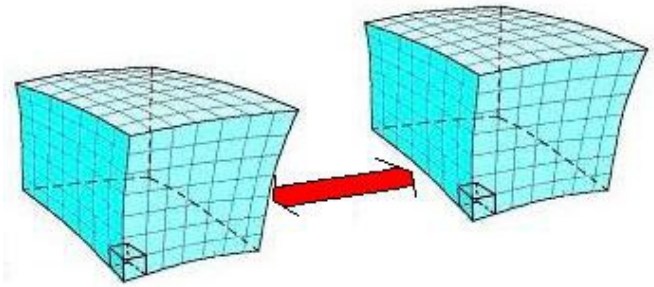


Computational blocks and cells

xxCell (Scheme)



Blocks (Topology)



Scheme function

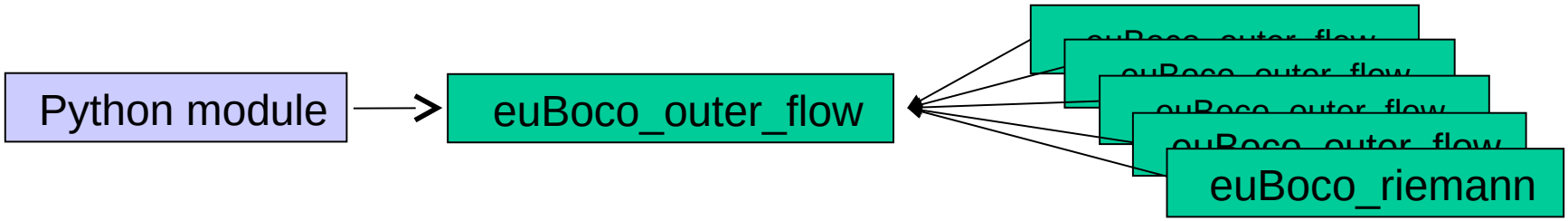
```
void corrector (const Cell* cell)
{
   ...
}
```

zSolver function

```
void step()
{
  join_blocks();
  for_each_cell (before_step);
  for_each_cell (grades);
  for_each_bound_cell (boco_grades);
  for_each_cell (predictor);
  join_blocks();
  for_each_cell (gradients);
  for_each_bound_cell (boco_gradients);
  join_blocks();
  for_each_home_side (side_gradients);
  for_each_bound_cell (boco_side_gradients);
  for_each_home_side (fluxes);
  for_each_bound_cell (boco_fluxes);
  for_each_cell (corrector);
}
```

Boundary Conditions inheritance

Unsteady Boundary Conditions




```
def p (x,y,z,time,iter) :
  if y>0.1 or y<0.065:
      return 1.0
  y1 = ((95.0 - 86.4)*33.972/46.5 + 65.241)/1000.0;
  y2 = ((111.0 - 86.4)*33.972/46.5 + 65.241)/1000.0;
  if y<y1:
      d = 1.47666 - 6.37986*y - 2.15226*y*y;
  if y >= y1 and y < y2:
      d = 2.03816 - 25.2697*y + 152.194*y*y;
  if y >= y2:
      d = 1.02288 - .404225*y;
  return d
```

Short Vectors

Coordinates

$$\vec{X} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \xi_i}{\partial x} & \frac{\partial \xi_i}{\partial y} & \frac{\partial \xi_i}{\partial z} \\ \frac{\partial \xi_j}{\partial y} & \frac{\partial \xi_j}{\partial y} & \frac{\partial \xi_j}{\partial y} \\ \frac{\partial \xi_k}{\partial z} & \frac{\partial \xi_k}{\partial z} & \frac{\partial \xi_k}{\partial z} \end{bmatrix}$$

$$\mathbf{J}^{-1} = \begin{bmatrix} \frac{\partial \xi_i}{\partial x} & \frac{\partial \xi_j}{\partial x} & \frac{\partial x}{\partial \xi_k} \\ \frac{\partial y}{\partial \xi_i} & \frac{\partial y}{\partial \xi_j} & \frac{\partial y}{\partial \xi_k} \\ \frac{\partial z}{\partial \xi_i} & \frac{\partial z}{\partial \xi_j} & \frac{\partial z}{\partial \xi_k} \end{bmatrix}$$

Euler equations

$$\vec{P} = \begin{bmatrix} T \\ u \\ v \\ w \\ p \end{bmatrix}$$

$$\vec{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix}$$

$$\vec{F} = \begin{bmatrix} Q \\ Qu + ps_x \\ Qv + ps_y \\ Qw + ps_z \\ Q \cdot (E + p/\rho) \end{bmatrix}$$

1. Vectors are short
2. We know length of vectors
3. We know names of components

RANS (*q*-*w* turbulence model)

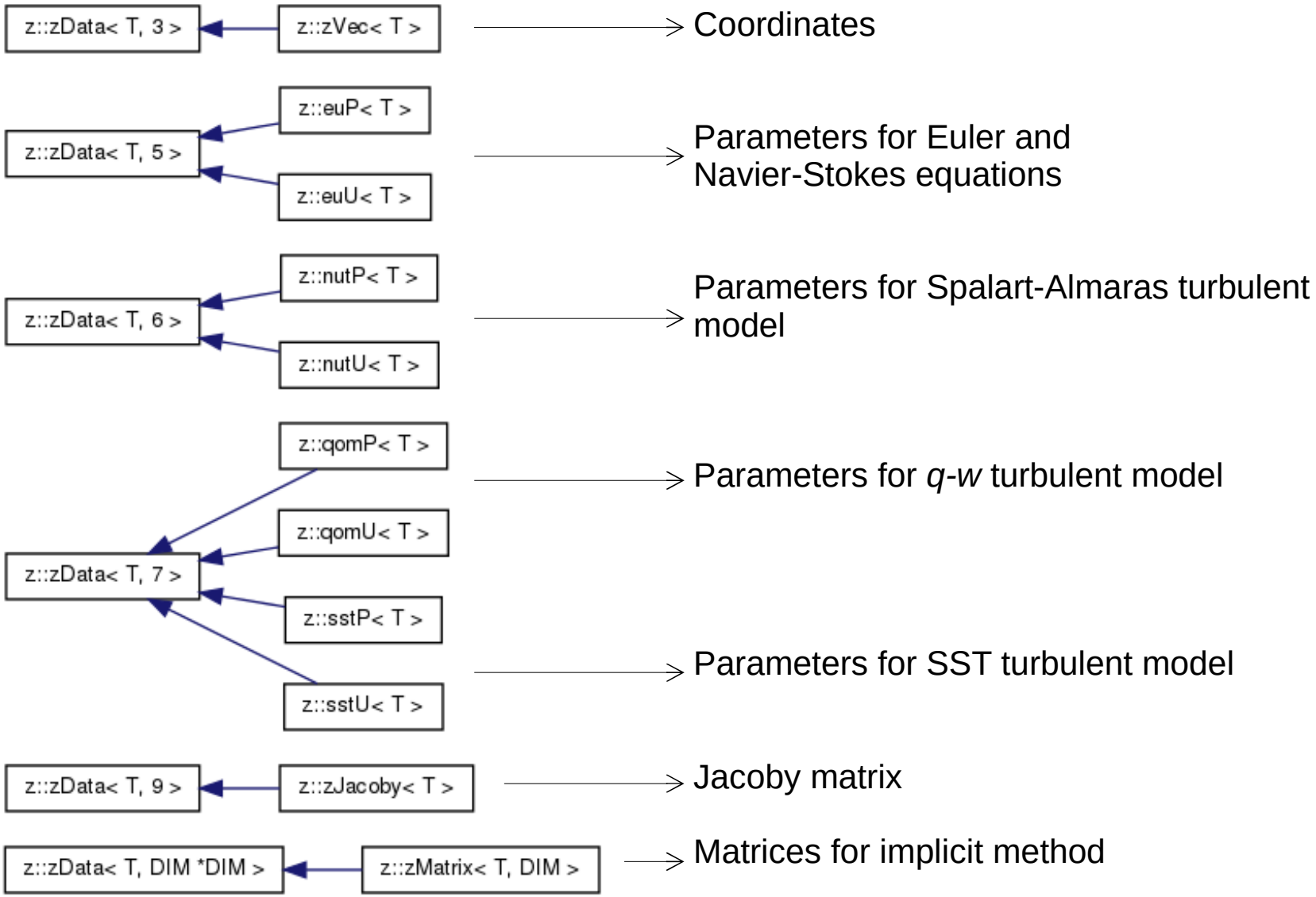
$$\vec{P} = \begin{bmatrix} T \\ u \\ v \\ w \\ p \\ q \\ \omega \end{bmatrix}$$

$$\vec{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \\ \rho q \\ \rho \omega \end{bmatrix}$$

$$\vec{W} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ S(q) \\ S(\omega) \end{bmatrix}$$

$$\vec{F} = \begin{bmatrix} Q \\ Qu + ps_x + I_{xn} \\ Qv + ps_y + I_{yn} \\ Qw + ps_z + I_{zn} \\ Q \cdot (E + p/\rho) + (I_{xn}u + I_{yn}v + I_{zn}w) + \theta_n + T_n^k \\ Qq + T_n^q \\ Q\omega + T_n^\omega \end{bmatrix}$$

Short Vectors. zData



zData. Realisation

```
template<typename T, size_t SIZE>
class zData
{
protected:
  T arr[SIZE];  // base array
```

```
  T mod ()
  {
    double sum = 0.0;
    for (size_t i = 0; i < SIZE; ++i)
      sum += arr[i] * arr[i];

    return static_cast<T>(sqrt (sum));
  }
```

```
  T& operator [] (size_t i)
  {
#ifdef Z_INDEX_CHECK
    if (i >= size ())
      throw zIndexErr (__FILE__, __LINE__);
#endif
    return arr[i];
  }
```

```
  const T& operator [] (size_t i) const
  {
#ifdef Z_INDEX_CHECK
    if (i >= size ())
      throw zIndexErr (__FILE__, __LINE__);
#endif
    return arr[i];
  }
```

```
// binary arithmetic operations between two arrays
zData<T, SIZE> operator + (const zData<T, SIZE>& A) const
{
  T X[SIZE];
  for (size_t i = 0; i < SIZE; ++i)
    X[i] = arr[i] + A.arr[i];

  return zData<T, SIZE> (X);
}
```

```
zData<T, SIZE> operator - (const zData<T, SIZE>& A) const
{
  T X[SIZE];
  for (size_t i = 0; i < SIZE; ++i)
    X[i] = arr[i] - A.arr[i];

  return zData<T, SIZE> (X);
}
```

```
zData<T, SIZE> operator * (const zData<T, SIZE>& A) const
{
  T X[SIZE];
  for (size_t i = 0; i < SIZE; ++i)
    X[i] = arr[i] * A.arr[i];

  return zData<T, SIZE> (X);
}
```

```
zData<T, SIZE> operator / (const zData<T, SIZE>& A) const
{
  T X[SIZE];
  for (size_t i = 0; i < SIZE; ++i)
    X[i] = arr[i] / A.arr[i];

  return zData<T, SIZE> (X);
}
```

zVec. Realization

$$\vec{X} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

```
//! 3D point or 3D vector
template<typename T>
class zVec
  : public zData<T, 3>
{
  using zData<T, 3>::arr;

public:
  --------
  //! Return elements
  T& x ()
  {
    return arr[0];
  }

  const T& x () const
  {
    return arr[0];
  }
  --------
};  // class zVec
```

```
//! Dot product
//! \param V1,V2 - vectors-arguments of product
//! \return result of dot product type T
template<typename T, typename S>
T dot (const zData<T, 3> &V1, const zData<S, 3>& V2)
{
  return V1[0] * V2[0] + V1[1] * V2[1] + V1[2] * V2[2];
}
```

```
//! Vector product
//! \param V1,V2 - vectors-arguments of product
//! \return result of vector product type zVec
template<typename T, typename S>
zVec<T> cross (const zData<T, 3>& V1, const zData<S, 3>& V2)
{
  return zVec<T> (V1[1] * V2[2] - V1[2] * V2[1],
                  V1[2] * V2[0] - V1[0] * V2[2],
                  V1[0] * V2[1] - V1[1] * V2[0]);
}
```

```
//! Normalization of the vector
//! \param V - vector - to normalize
//! \return result of normalization type zVec
template<typename T>
zVec<T> norm (const zData<T, 3>& V)
{
  T mod_v_inv = 1.0 / mod(V);
  if (!std::isfinite (mod_v_inv))
    throw zDivideNull (__FILE__, __LINE__);

  zVec<T> N = V * mod_v_inv;

  return N;
}
```

zJacoby. Realization

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \xi_i}{\partial x} & \frac{\partial \xi_i}{\partial y} & \frac{\partial \xi_i}{\partial z} \\ \frac{\partial \xi_j}{\partial y} & \frac{\partial \xi_j}{\partial y} & \frac{\partial \xi_j}{\partial y} \\ \frac{\partial \xi_k}{\partial z} & \frac{\partial \xi_k}{\partial z} & \frac{\partial \xi_k}{\partial z} \end{bmatrix}$$

```
//! Jacoby matrix of 3x3 size
template<typename T>
class zJacoby : public zData<T, 9>
{
  using zData<T, 9>::arr;

public:
 --------

  //! Return elements
  T& xx ()             { return arr[0]; }
  const T& xx () const { return arr[0]; }

 --------

  T& at (size_t i, size_t j)
  {
    return arr[i + 3 * j];
  }

  const T& at (size_t i, size_t j) const
  {
    return arr[i + 3 * j];
  }
 --------
```

```
template<typename T>
zJacoby<T> operator * (const zJacoby<T>& M1, const zJacoby<T>& M2)
{
  zJacoby<T> J;
  J.set_column (0, M1*M2.column(0));
  J.set_column (1, M1*M2.column(1));
  J.set_column (2, M1*M2.column(2));
  return J;
}
-----------------------------------------------------------------------------
template<typename T, typename S>
zVec<S> operator * (const zJacoby<T>& _J, const zVec<S>& _V)
{
  zVec<S> V;
  V.x() = _J.xx()*_V.x() + _J.xy()*_V.y() + _J.xz()*_V.z();
  V.y() = _J.yx()*_V.x() + _J.yy()*_V.y() + _J.yz()*_V.z();
  V.z() = _J.zx()*_V.x() + _J.zy()*_V.y() + _J.zz()*_V.z();
  return V;
}
-----------------------------------------------------------------------------
template<typename T, typename S>
zVec<S> operator * (const zVec<S>& _V, const zJacoby<T>& _J)
{
  zVec<S> V;
  V.x() = _J.xx()*_V.x() + _J.yx()*_V.y() + _J.zx()*_V.z();
  V.y() = _J.xy()*_V.x() + _J.yy()*_V.y() + _J.zy()*_V.z();
  V.z() = _J.xz()*_V.x() + _J.yz()*_V.y() + _J.zz()*_V.z();
  return V;
}
```

```
//! Roe matrix on home sides
virtual void imp_matrix (Side& side,
                         const PVec& Pl, const PVec& Pr, const PVec& Psi,
                         const PVec& Gx, const PVec& Gy, const PVec& Gz,
                         zMatrix<double, PVec::SIZE>& Rcl,
                         zMatrix<double, PVec::SIZE>& Rcr,
                         zMatrix<double, PVec::SIZE>& Rsl,
                         zMatrix<double, PVec::SIZE>& Rsr)
{
  //roe
  zVec<double>& S = side.S;
  zMatrix<double, PVec::SIZE> A, mod_A, I;
  matrix_roe_eu (Pl, Pr, S, A, mod_A, I);

  double mod_S = mod (S);
  Rcl = 0.5*mod_S*(A + mod_A);
  Rcr = 0.5*mod_S*(A - mod_A);

  //diffusion (momentum terms)
  zMatrix<double, PVec::SIZE> dFdU;
  matrix_dFdU_ns (Psi, Gx, Gy, Gz, S, dFdU);
  zMatrix<double, PVec::SIZE> E;
  E.identity ();
  zMatrix<double, PVec::SIZE> Zp = 0.5*dFdU*(E + I);
  zMatrix<double, PVec::SIZE> Zm = 0.5*dFdU*(E - I);
  Rcl = Rcl + Zp;
  Rcr = Rcr + Zm;

  //diffusion (other terms)
  double d_inv = 1.0 / (d_side (side, 0) + d_side (side, 1));
  zMatrix<double, PVec::SIZE> Aq;
  matrix_Aq_ns (Psi, S, Aq);
  zMatrix<double, PVec::SIZE> Xl;
  matrix_X_ns (Pl, Xl);
  zMatrix<double, PVec::SIZE> Xr;
  matrix_X_ns (Pr, Xr);
  Rcl = Rcl - d_inv*Aq*Xl;
  Rcr = Rcr + d_inv*Aq*Xr;
}
```

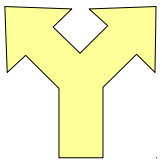
Method. Time Stepping

Explicit Steady
- local
- multigrid

Implicit Steady
- local
- global

Explicit Unsteady
- global
- fractional

Implicit Unsteady
- dual & local explicit
- dual & global implicit

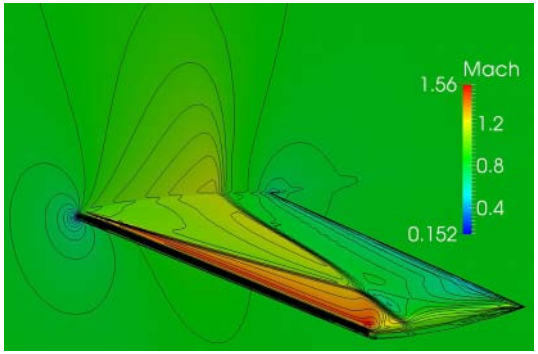


Zonal approach

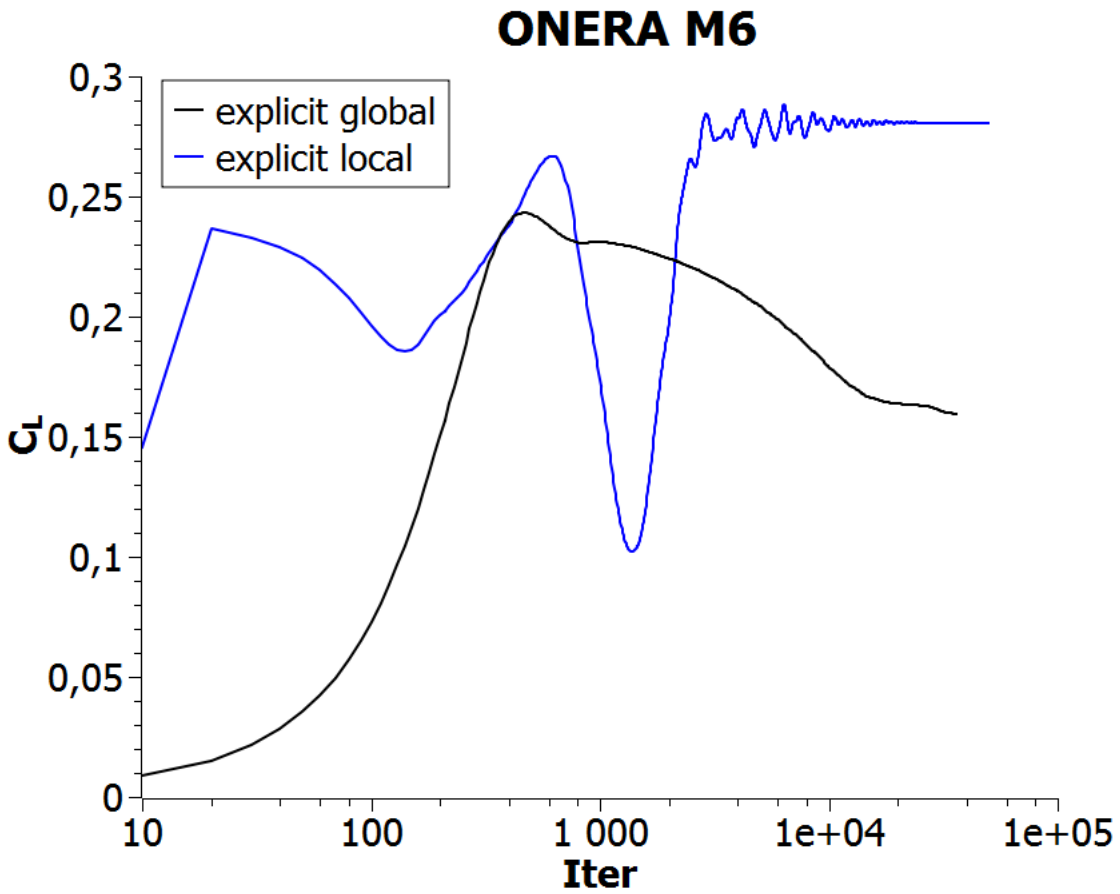
Explicit Scheme

- Власенко В.В. О математическом подходе и принципах построения численных методологий для пакета прикладных программ EWT-ЦАГИ. // Труды ЦАГИ, выпуск 2671, с.20-85, 2007

Explicit scheme



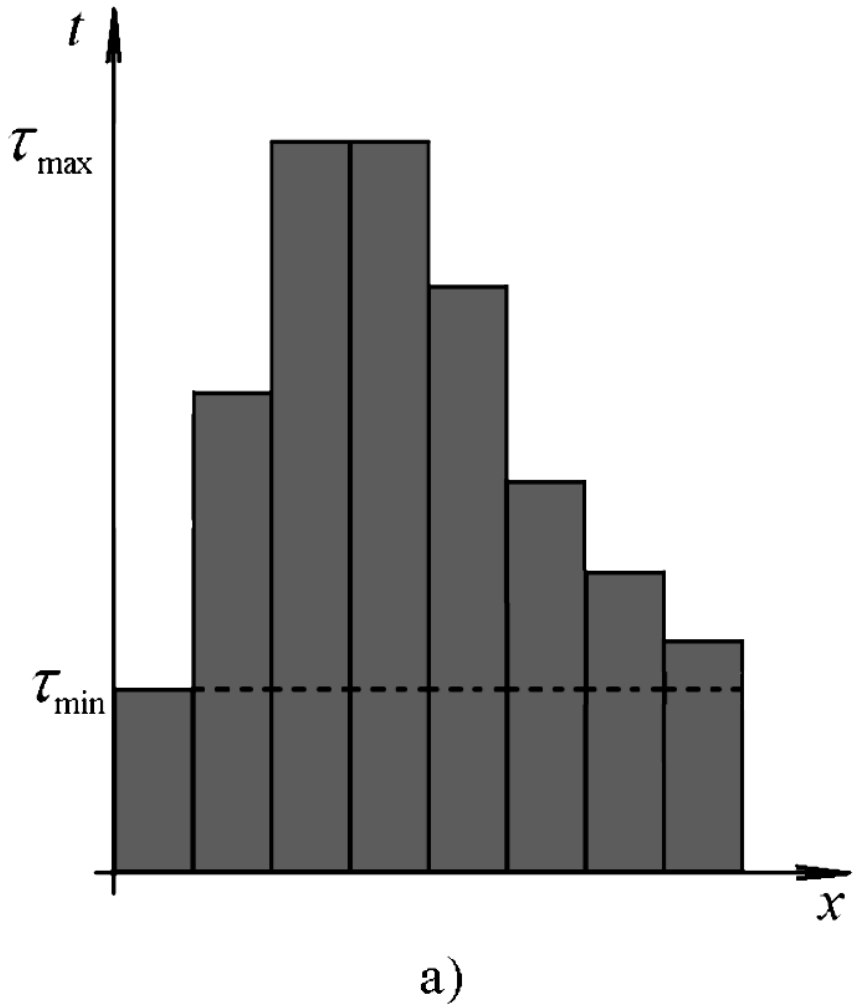
M=0.84
AOA = 3.0



Fractional Time Stepping

- Pervaiz, M.M., Baron, J.R. 1989. Spatiotemporal adaptation algorithm for two-dimensional reacting flows. AIAA Journal, Vol.27, No.10.
- Власенко В.В. О математическом подходе и принципах построения численных методологий для пакета прикладных программ EWT-ЦАГИ. // Труды ЦАГИ, выпуск 2671, с.20-85, 2007
- Власенко В.В., Михайлов С.В., Морозов Н.А. Разработка и верификация численного метода для исследования в рамках EWT-ЦАГИ тяговых и акустических характеристик сопл сложной пространственной конфигурации. // Труды ЦАГИ, выпуск 2671, 2007.

Global Time Stepping



a)

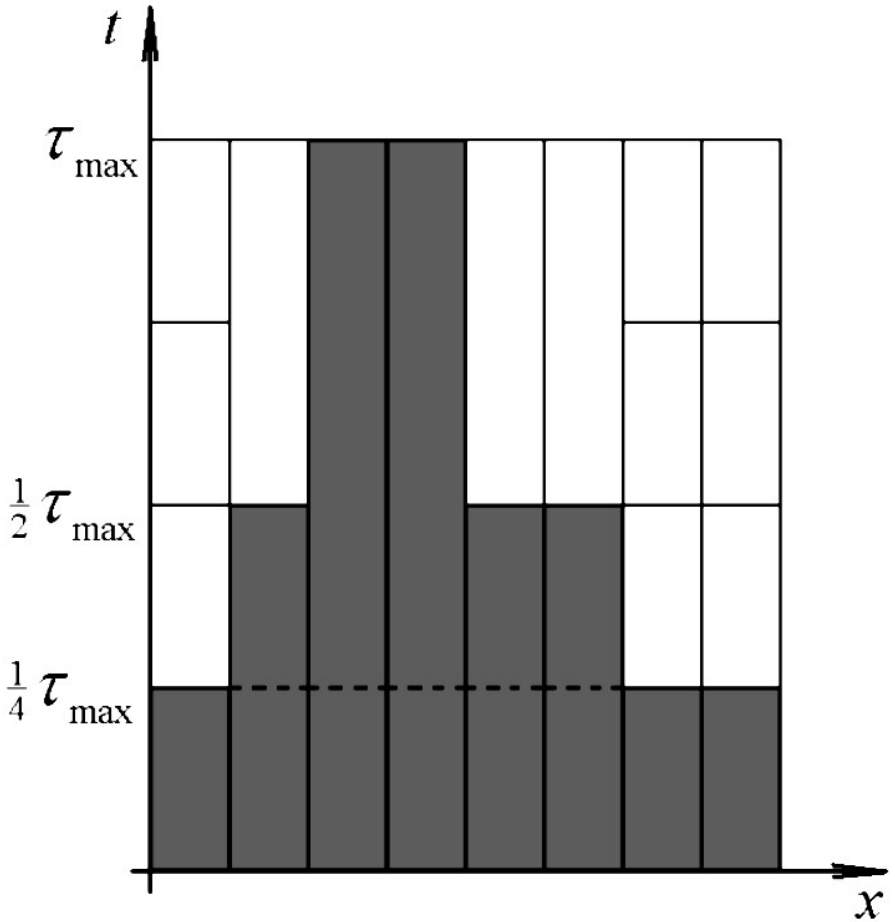
Global Time Stepping:

$\tau < \tau_{max}$

As a result:

- Computations are long
- CFL << 1 – great dissipation errors

Fractional Time Stepping

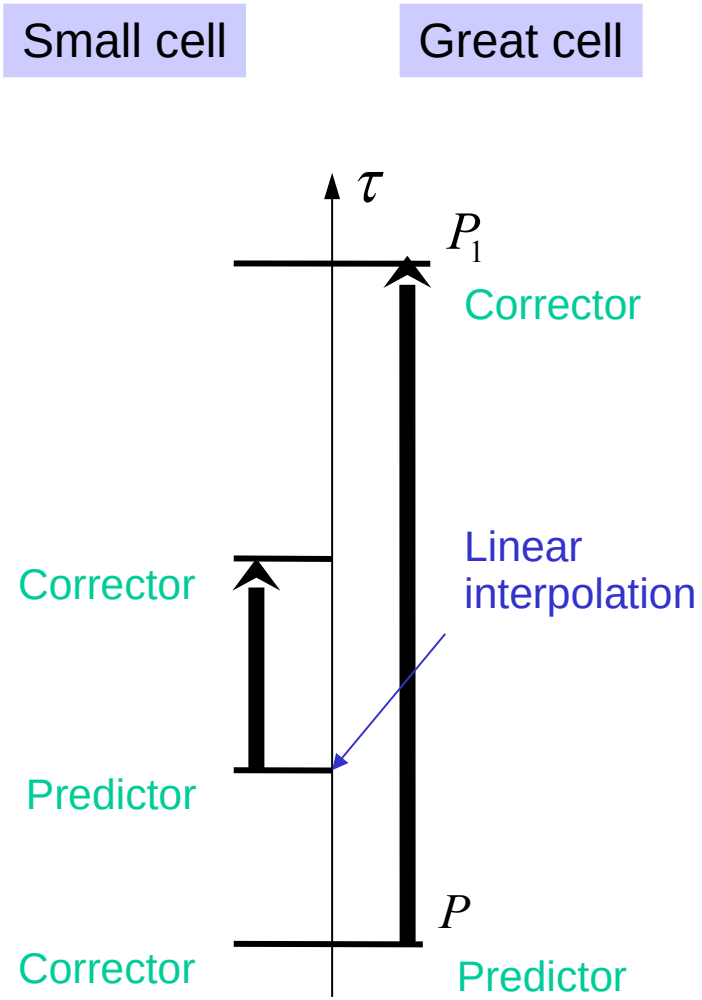


б)

Advantages:

- Correct simulation of unsteady processes
- A few steps in big cells – computations are more quicker
- 0.5 < CFL < 1 – little dissipation errors

Fractional Time Stepping



$\tau_{\min}$ - Minimal step time

τ_{cell} - Step time in cell

$n_{cell} \geq \frac{\tau_{cell}}{\tau_{\min}}$ - Number of fractional steps in cell

$n_{\max} = \max_{i}(n_{cell})$

$n_{iter} = \min_{1}^{6}(\frac{n_{cell}}{2})$ - Interpolation step

$(n + n_{cell})\%(2 \cdot n_{cell}) == 0$ - Cell is in the predictor

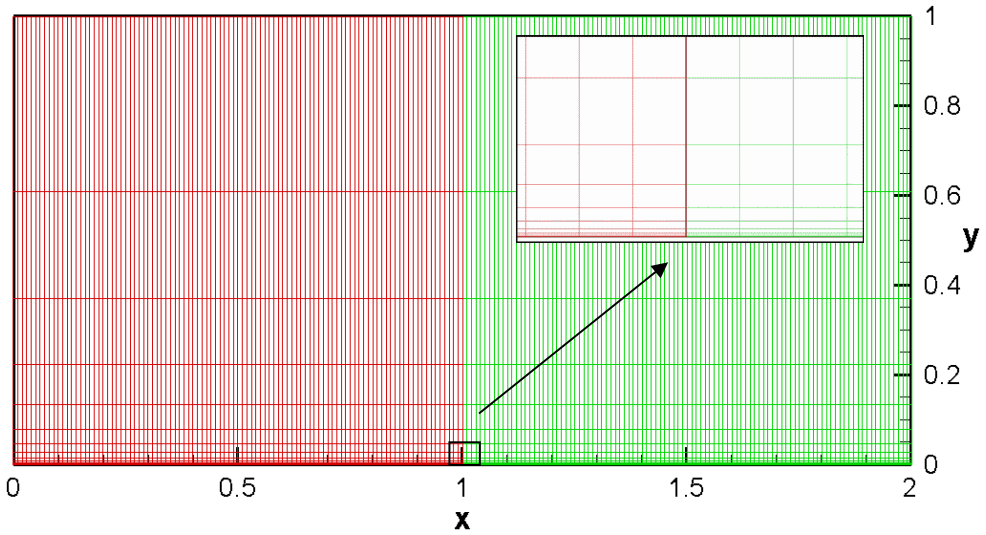
$n\%(2 \cdot n_{cell}) == 0$ - Cell is in the corrector

$n\%n_{inter} == 0$

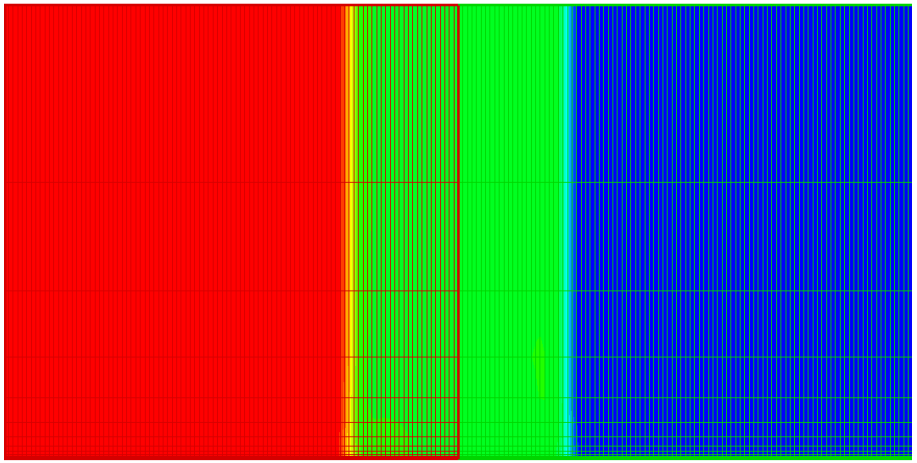
$dn = \frac{n_{\max} - n}{n_{inter}}$ - Condition of the interpolation

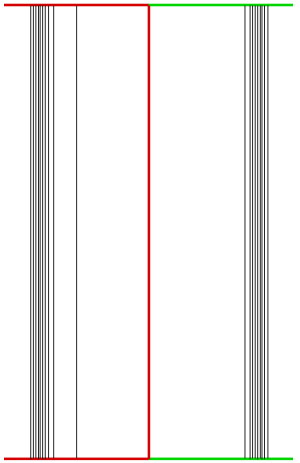
$P = \frac{P_1 + P \cdot dn}{dn + 1}$ - Interpolated value

Fractional Time Stepping. Decay

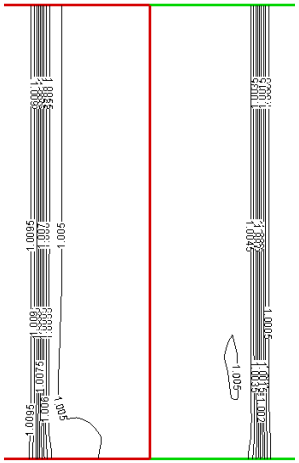


	acoustic	collision	fly away	mach3
ρ	1.01	1.0	1.0	3.875
u	0.0	1.7748239	-1.7748239	0.92
p	1.01	1.0	1.0	10.333
T	0.0034835	0.0034835	0.0034835	0.0093325
ρ	1.0	1.0	1.0	1.0
u	0.0	-1.7748239	1.7748239	3.55
p	1.0	1.0	1.0	1.0
T	0.0034835	0.0034835	0.0034835	0.0034835
t	0.2	0.1	0.1	0.09



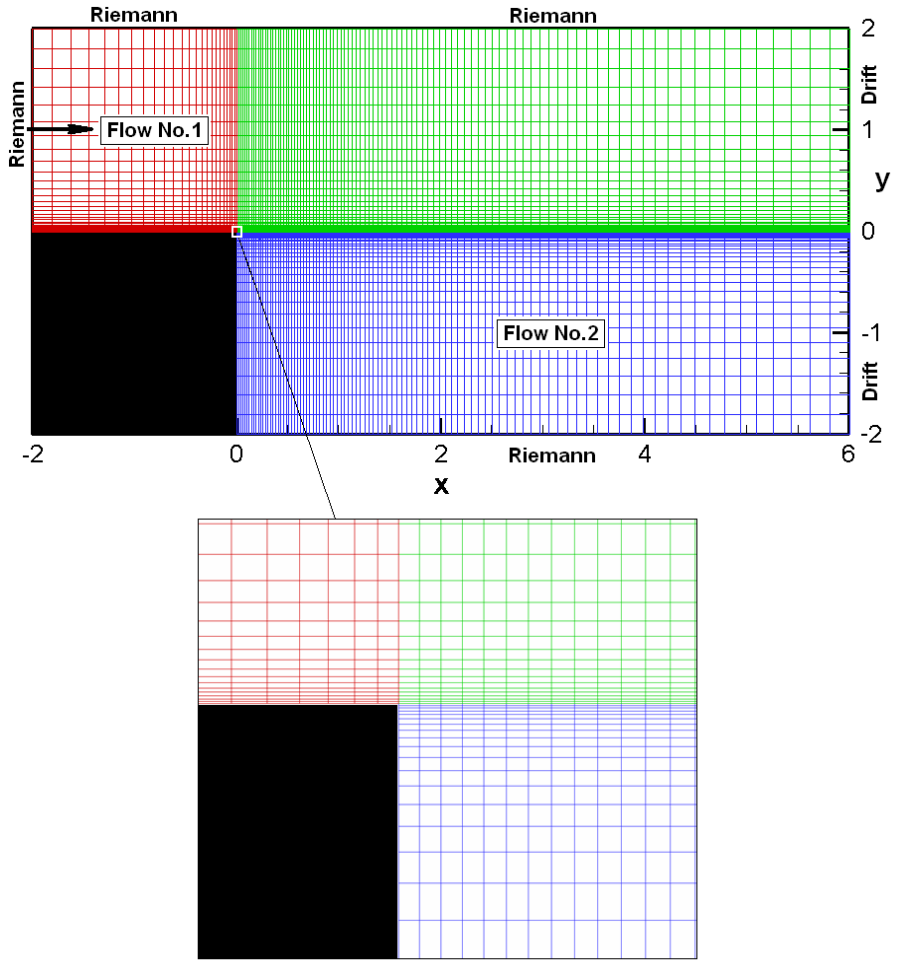


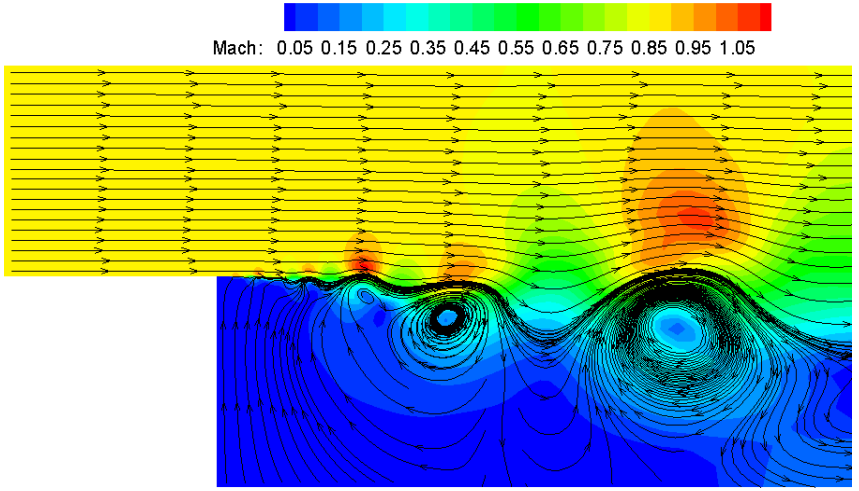
global

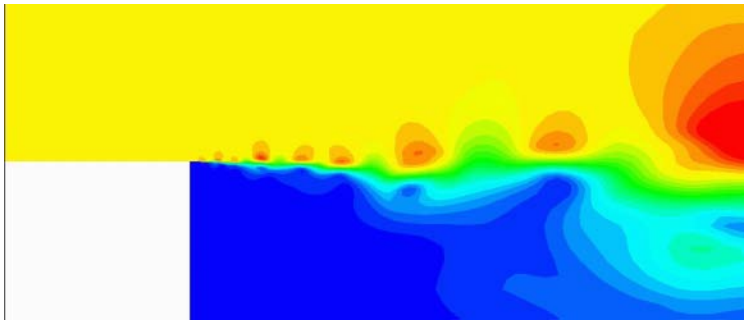


fractional

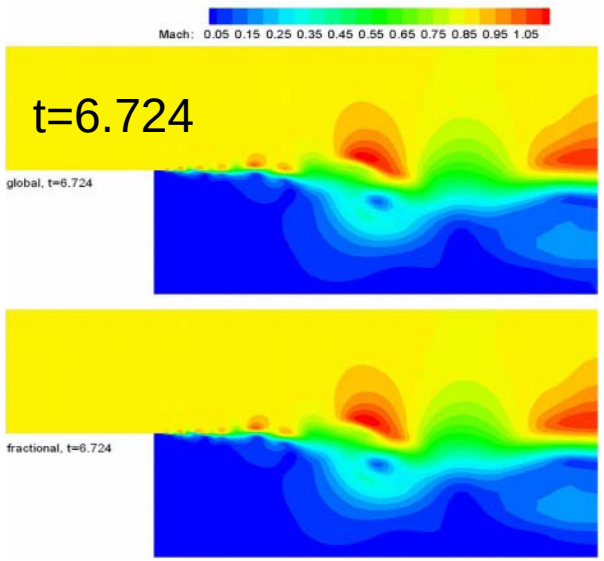
Fractional Time Stepping. Back-step. M=0.85

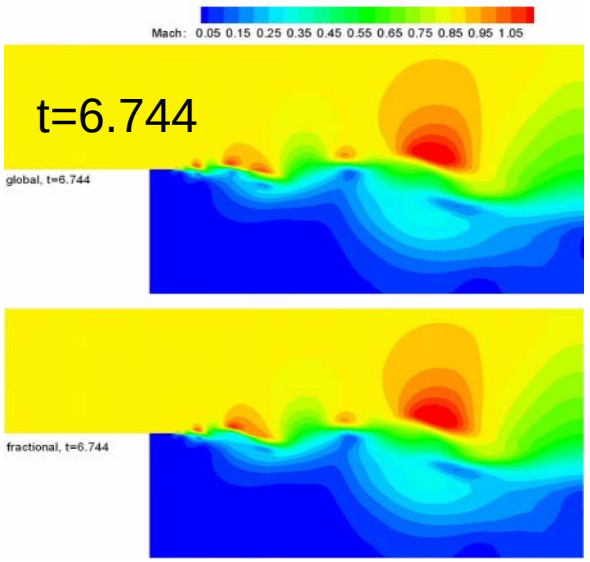


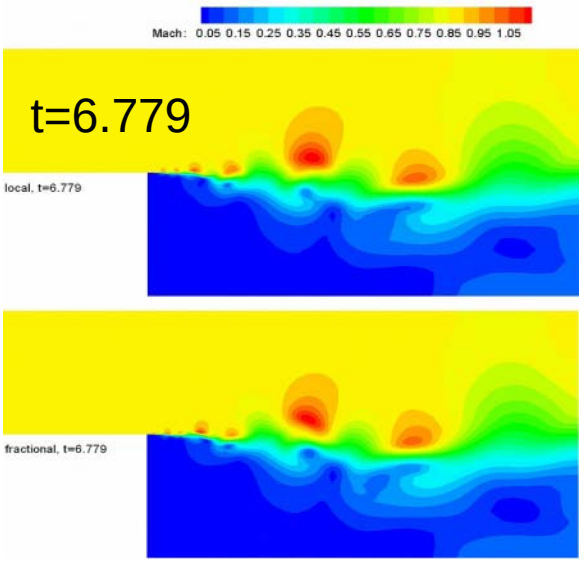


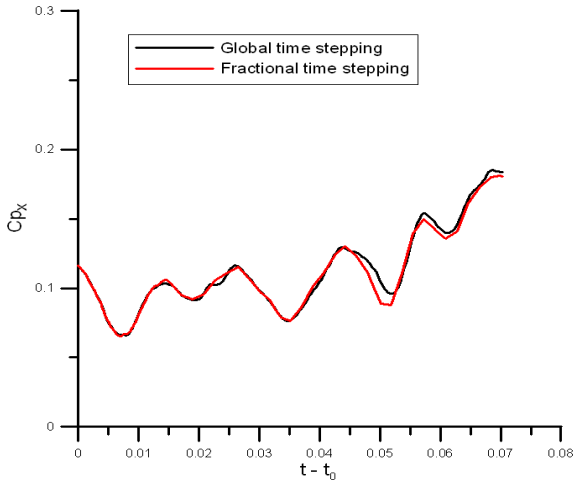


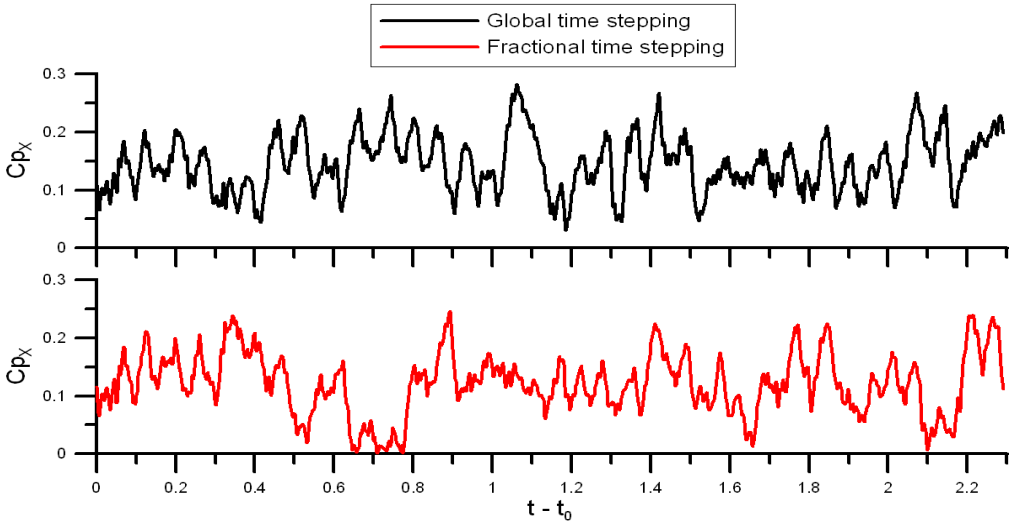
Fractional Time Stepping. Back-step. M=0.85











Multigrid

- Blazek J,Computational Fluid Dynamics: principles and applications
- Toro E.F. Riemann Solvers and Numerical Methods for Fluid Dynamic
- Jameson A. Acceleration of Transonic Potential Flow Calculations on Arbitrary Meshes by the MultiGrid Method // AIAA-79-1458-CP (1979)

Multigrid

1. Fine mesh → Coarse mesh

$$\vec{W}_{2h}^{n+1} \xrightarrow[\text{interpolation}]{\hat{I}_{2h}^{4h}} \vec{W}_{4h}^{(0)} = \hat{I}_{2h}^{4h}\vec{W}_{2h}^{n+1}$$

$$\vec{R}_{2h}^{n+1} + \vec{Q}_F \xrightarrow[\text{restriction}]{I_{2h}^{4h}} I_{2h}^{4h}\left[\vec{R}_{2h}^{n+1} + \left(\vec{Q}_F\right)_{2h}\right]$$

2. Computation at the coarse mesh

3. Coarse mesh → Fine mesh

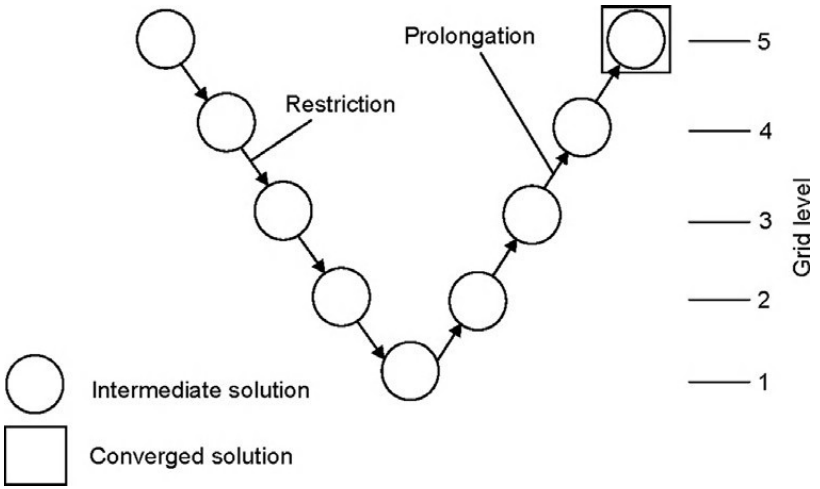
$$\delta\vec{W}_{2h} = \vec{W}_{2h}^{n+1} - \vec{W}_{2h}^{(0)} \xrightarrow[\text{prolongation}]{I_{2h}^{h}} I_{2h}^{h}\delta\vec{W}_{2h}$$

4. Computation at the fine mesh

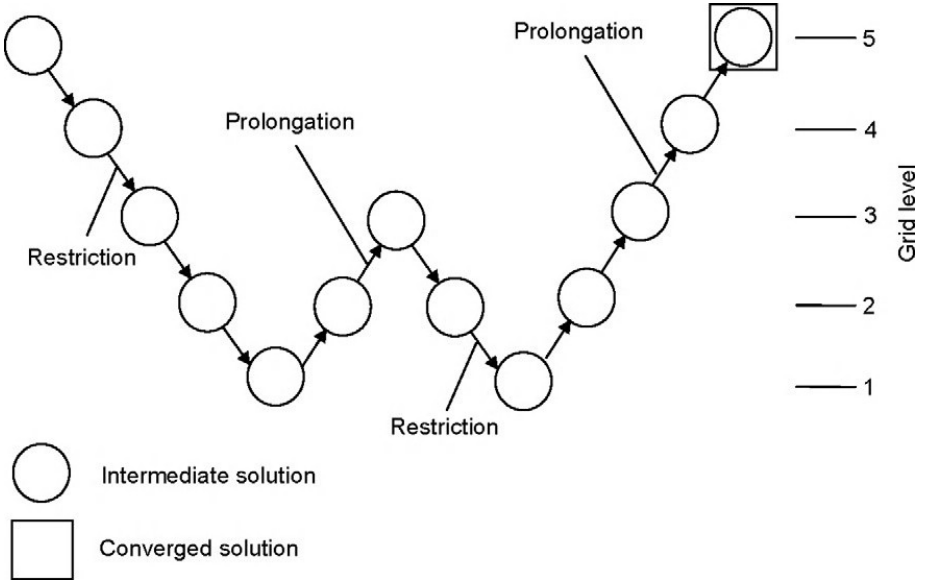
$$\vec{W}_h^{+} = \vec{W}_h^{n+1} + I_{2h}^{h}\delta\vec{W}_{2h}$$

Multigrid

V - cycle



W - cycle



1. How to set number of levels?
2. How to set cycle?

Multigrid

```
  Solver solver;

  …

  size_t multigrid_levels_number;   // read number of levels from project


// init solvers
  std::vector<Solver*> solv;

  solv.push_back(&solver);

  for (size_t level=1; level<multigrid_levels_number; level++)
    init_coarse_solver(solv[level]);
```

level	indexes	cells number
0	80 x 48 x 16	61440
1	40 x 24 x 8	7680
2	20 x 12 x 4	960
3	10 x 6 x 2	120
4	5 x 3 x 1	15

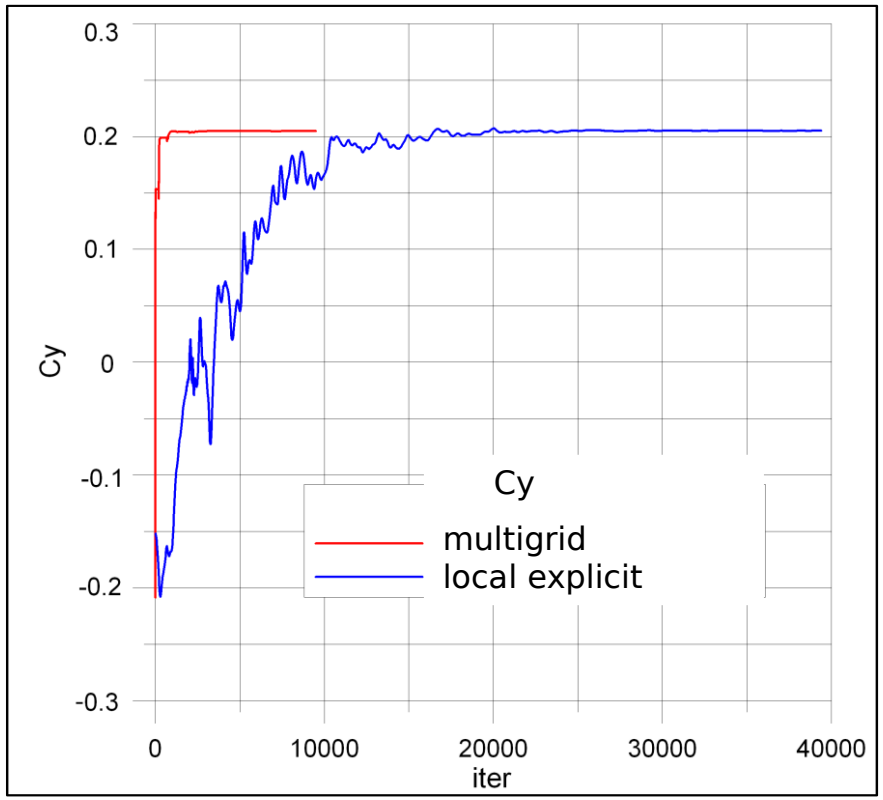
Multigrid

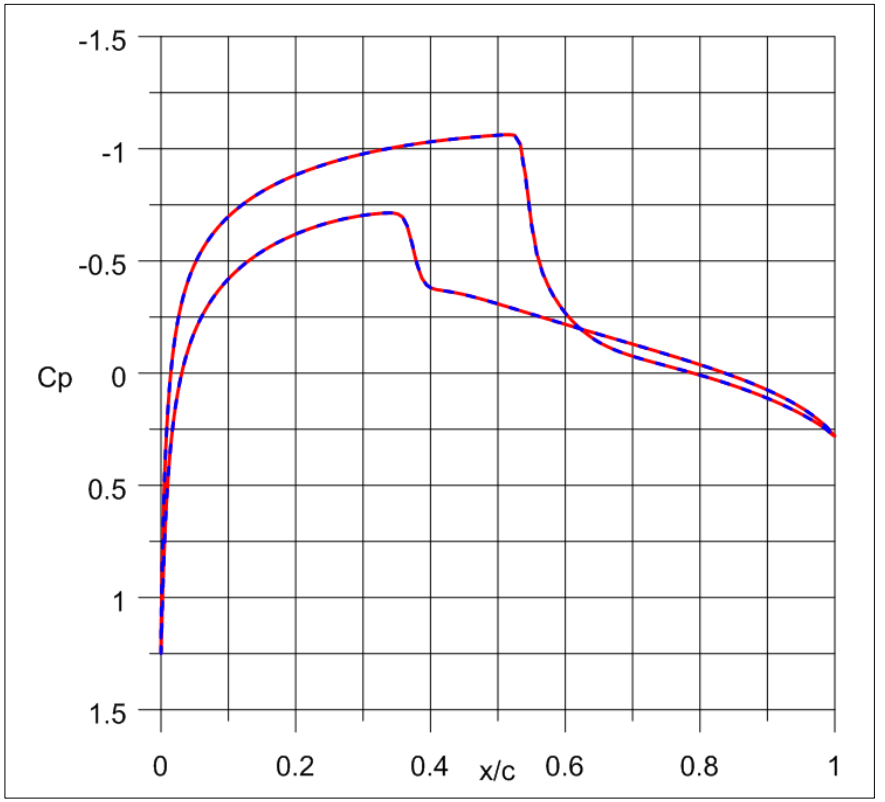
```
// init strategy
  std::vector<int> multigrid_step;

  if (multigrid_levels_number>0) {
    multigrid_step.push_back(RESTRICTION);
    multigrid_step.push_back(COARSE_RESIDUAL);
    if (multigrid_levels_number>1) {
      multigrid_step.push_back(RESTRICTION);
      multigrid_step.push_back(COARSE_RESIDUAL);
      if (multigrid_levels_number>2) {
        multigrid_step.push_back(RESTRICTION);
        multigrid_step.push_back(COARSE_RESIDUAL);
        multigrid_step.push_back(COARSE_SMOOTHING);
        multigrid_step.push_back(PROLONGATION);
      }
      multigrid_step.push_back(COARSE_SMOOTHING);
      multigrid_step.push_back(PROLONGATION);
    }
    multigrid_step.push_back(COARSE_SMOOTHING);
    multigrid_step.push_back(PROLONGATION);
  }

// init solvers
  Solver solv[multigrid_levels_number];
  for (int level=1; level<=multigrid_levels_number; level++)
    init_coarse_solver(solv[level]);
```

Multigrid

```
// step residual
        solv[0].multigrid_step_residual_2nd_order();

// multigrid steps begin
        size_t level = 0;
        for (size_t step=0; step<multigrid_step.size(); step++)
        {
          switch(multigrid_step[step])
          {
            case (RESTRICTION):
              transfer_to_coarse(*(solv[level]),solv[level+1]);
              Level++;
              break;
            case (COARSE_RESIDUAL):
              solv[level]->multigrid_step_coarse(true);
              break;
            case (PROLONGATION):
              transfer_to_fine(*(solv[level-1]),solv[level]);
              level--;
              break;
          }
        }

// step smoothing
        solv[0].multigrid_step_2nd_order();
```

Multigrid. NACA0012 M=0.8, AOA=1.0





Acceleration up to 5 – 50 times

Implicit Scheme

- Иванов М.Я., Нигматулин Р.З. Неявная схема С.К.Годунова повышенной точности для численного интегрирования уравнений Эйлера. // ЖВМ и МФ. 1987, т.27, №11, стр 1725-1735.
- Кажан Е.В. Неявная схема в проектах EWT и ZEUS. // Материалы XX школы-семинара "Аэродинамика летательных аппаратов", п. Володарского, 26-27 февраля 2009 г, стр. 62-63

Implicit Scheme

$$\frac{\vec{u}_{i,j,k}^{n+1} - \vec{u}_{i,j,k}^{n}}{\tau^n} + \frac{1}{V_{i,j,k}} \cdot \left[\sum_{i,j,k} \left(\vec{F}_{i+1/2} - \vec{F}_{i-1/2} \right) \right]^{n+1} - \vec{W}_{i,j,k} = 0$$

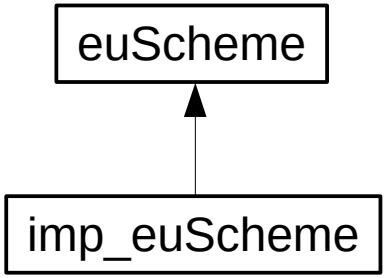
$$u^{n+1} = u^n - \frac{\tau^n}{V_i} \left(\vec{F}^n_{i+\frac{1}{2}} - \vec{F}^n_{i-\frac{1}{2}} \right) + \left(- RM^n_{i-1} \Delta\vec{u}_{i-1} + RM^n_i \Delta\vec{u}_i + RM^n_{i+1} \Delta\vec{u}_{i+1} \right) - \vec{W}_{i,j,k}$$

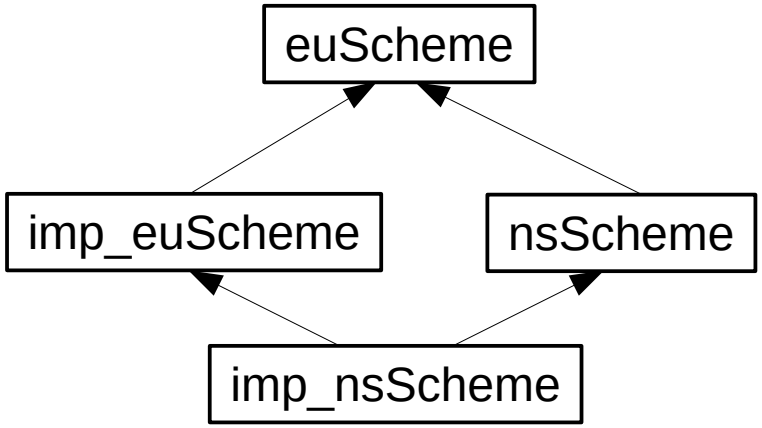
$$- RM^n_{i-1} \Delta\vec{u}_{i-1} + \left[1 + \frac{\tau^n}{V_i} RM^n_i \right] \Delta\vec{u}_i + RM^n_{i+1} \Delta\vec{u}_{i+1} = - \frac{\tau^n}{V_i} \left(\vec{F}^n_{i+\frac{1}{2}} - \vec{F}^n_{i-\frac{1}{2}} \right) - \vec{W}$$

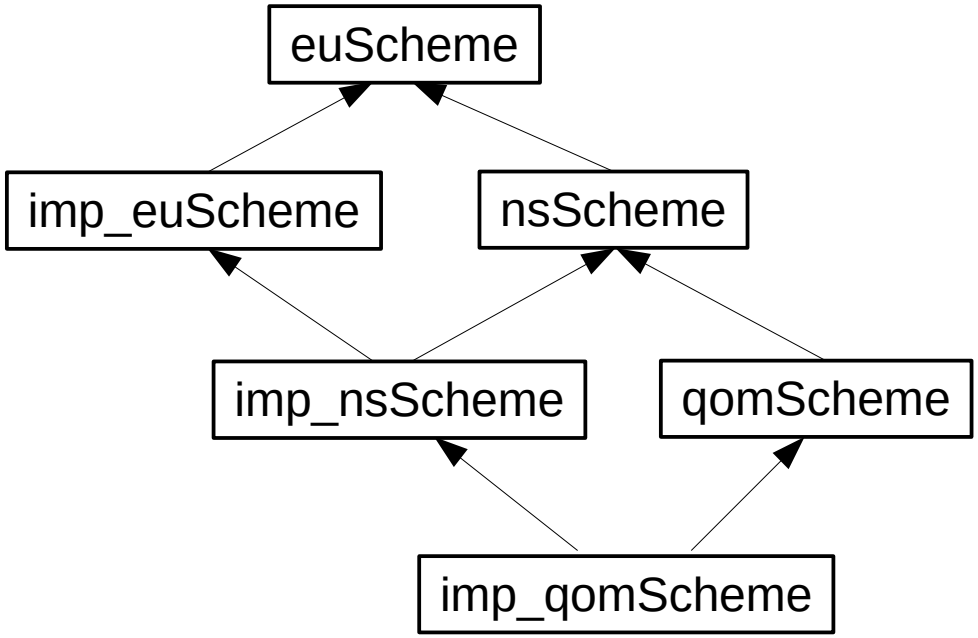
$$\vec{u}_i^{\,n+1} = \vec{u}_i^{\,n} + \Delta\vec{u}_i$$

RAM (implicit) = ~3*RAM (explicit)

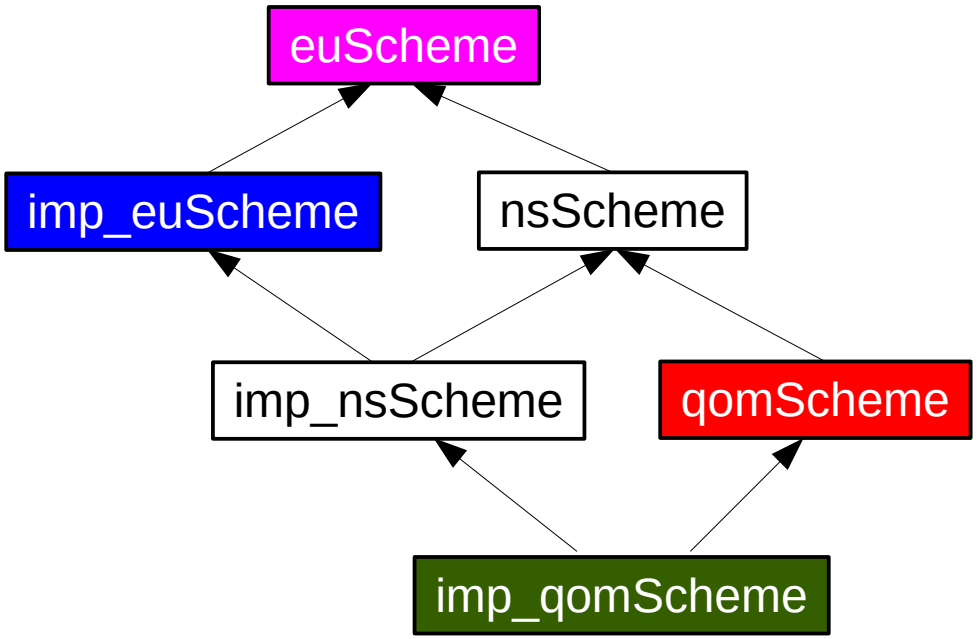
Implicit Scheme







Implicit Scheme




```
imp_qomScheme::foooo()
{
    return E();
}
```

Error: Abiguous call

```
imp_euScheme::foo()
{
    return E();
}

imp_qomScheme::foooo()
{
    return foo();
}
```

Result – euE – error!!!

```
virtual euScheme::E()
{
    return euE;
}
```

```
virtual qomScheme::E()
{
    return euE + qomE;
}
```

Implicit Scheme

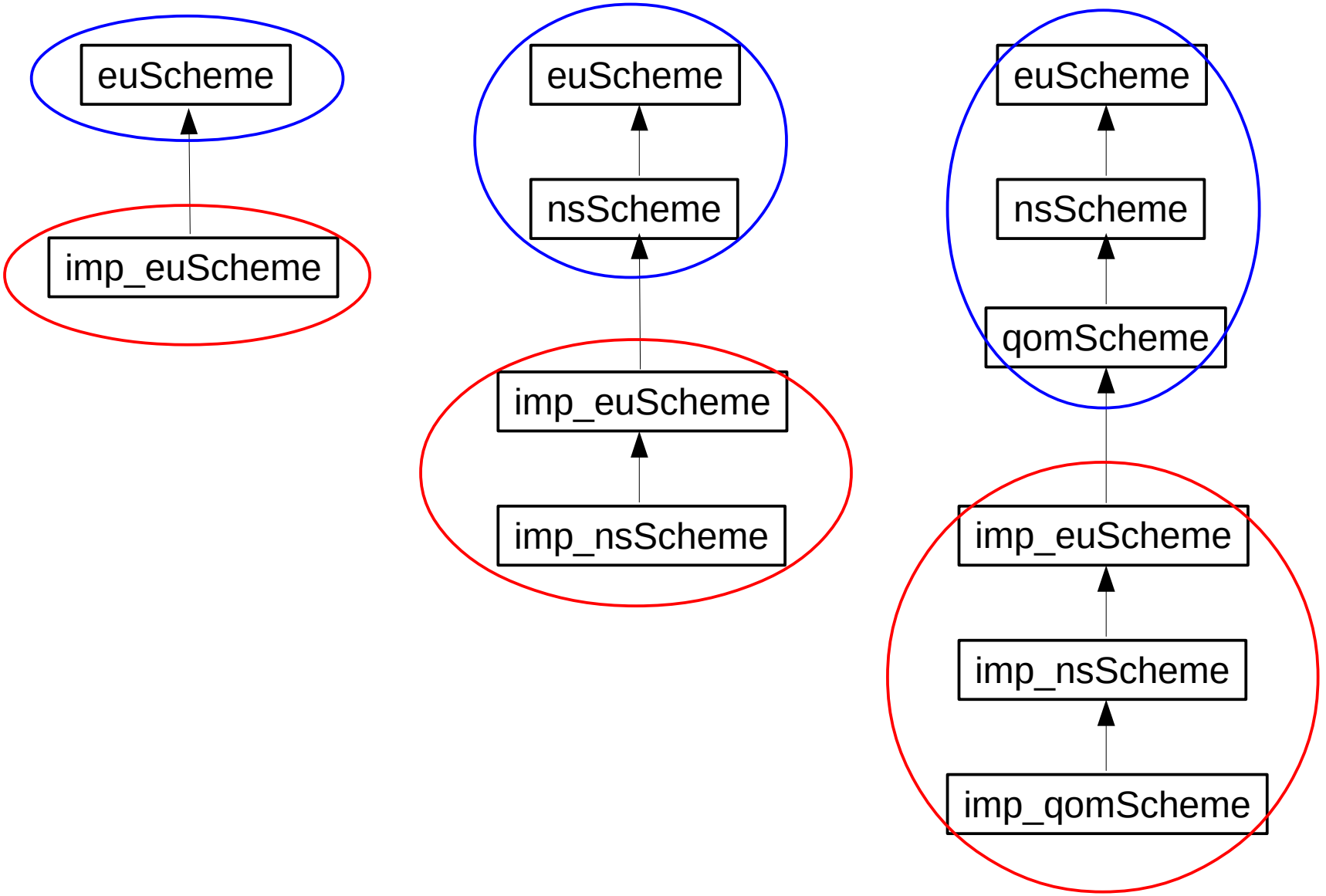
Inheritance for Explicit Scheme

```
template<class PVec,class UVec,class Cell,class Scheme,class Boco>
class nsSolver : public euSolver<PVec,UVec,Cell,Scheme,Boco>
{};
```

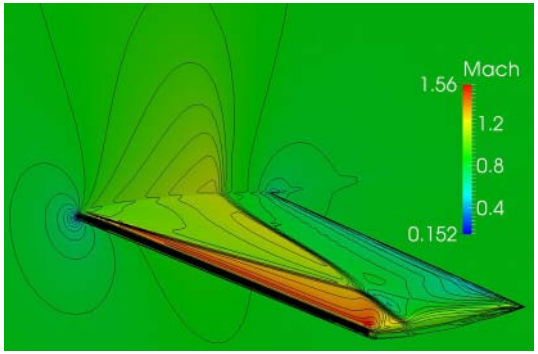
Inheritance for the Implicit Scheme

```
template<class Solver,class PVec,class UVec,class Cell,class Scheme,class Boco>
class imp_euSolver_base : public Solver
{};

template<class Solver,class PVec,class UVec,class Cell,class Scheme,class Boco>
class imp_nsSolver_base : public imp_euSolver_base<Solver,PVec,UVec,Cell,Scheme,Boco>
{};

template<class PVec,class UVec,class Cell,class Scheme,class Boco>
class imp_nsSolver : public imp_nsSolver_base<nsSolver<PVec,UVec,Cell,Scheme,Boco>
                                              ,PVec,UVec,Matrix,Cell,Scheme,Boco>
{};
```

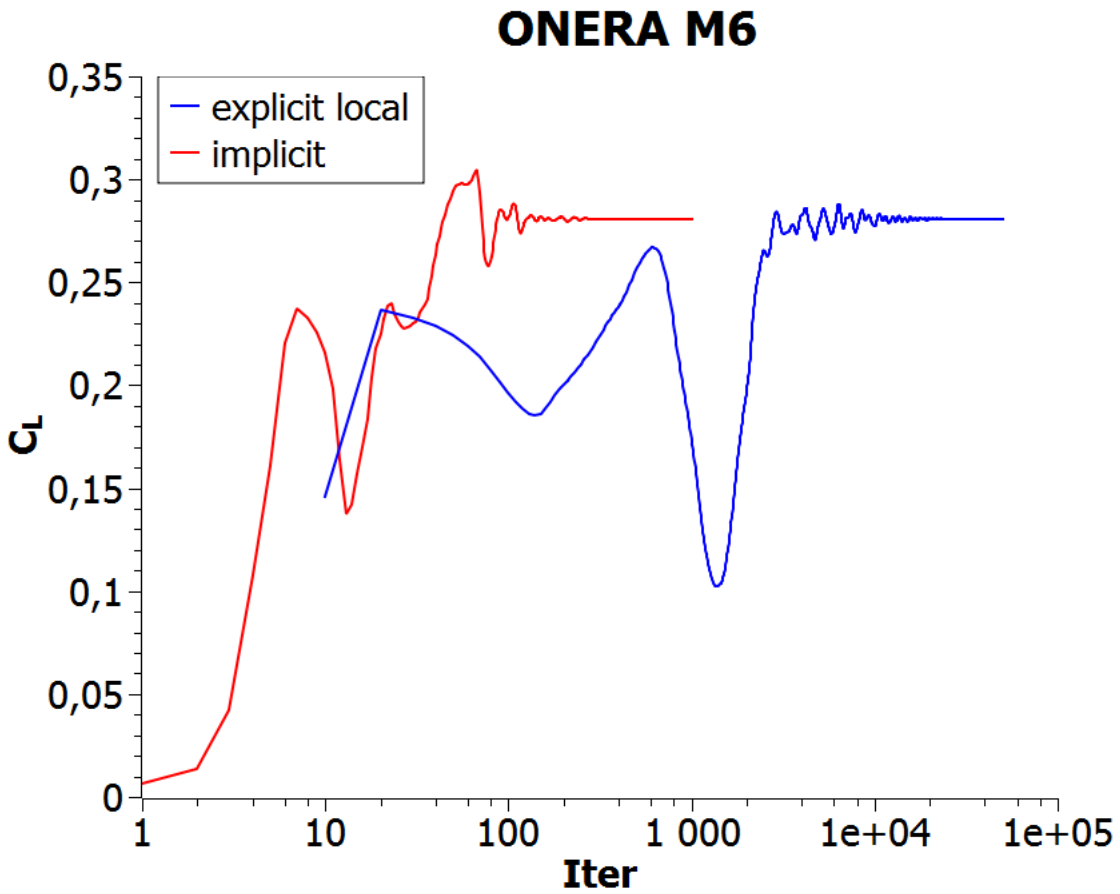
Implicit Scheme



Implicit scheme



M=0.84
AOA = 3.0



Dual Time Stepping

- Jameson A. A perspective on computational algorithms for aerodynamic analysis and design // Progress in Aerospace Sciences. 2001. V. 37 N2
- NUMECA
- ZEN CIRA

Dual time stepping

$$\frac{\partial u}{\partial t} + \frac{\partial F(u)}{\partial x} = 0$$

Implicit Scheme of 2-nd order of accuracy in time:

$$\frac{3}{2}\frac{u_i^{n+1} - u_i^n}{\tau} - \frac{1}{2}\frac{u_i^n - u_i^{n-1}}{\tau} + \frac{F_{i+1/2}(u^{n+1}) - F_{i-1/2}(u^{n+1})}{h} = 0$$

Equation:

$$\frac{\partial u}{\partial \xi} + \frac{3}{2}\frac{u - u_i^n}{\tau} - \frac{1}{2}\frac{u_i^n - u_i^{n-1}}{\tau} + \frac{F_{i+1/2}(u) - F_{i-1/2}(u)}{h} = 0$$

Explicit Scheme with local time stepping in time:

$$\begin{cases} u_i^{(0)} = u_i^n, \\ \dfrac{u_i^{(k+1)} - u_i^{(k)}}{\Delta\xi} + \dfrac{3}{2}\dfrac{u_i^{(k)} - u_i^n}{\tau} - \dfrac{1}{2}\dfrac{u_i^n - u_i^{n-1}}{\tau} + \dfrac{F_{i+1/2}(u^{(k)}) - F_{i-1/2}(u^{(k)})}{h} = 0, \quad k = 1, \ldots, M, \\ u_i^{n+1} = u_i^{(M)} \end{cases}$$

Zonal approach

S.Deck. Zonal-Detached-Eddy Simulation of the Flow Around a High-Lift Configuration. // AIAA JOURNAL Vol. 43, No. 11, November 2005

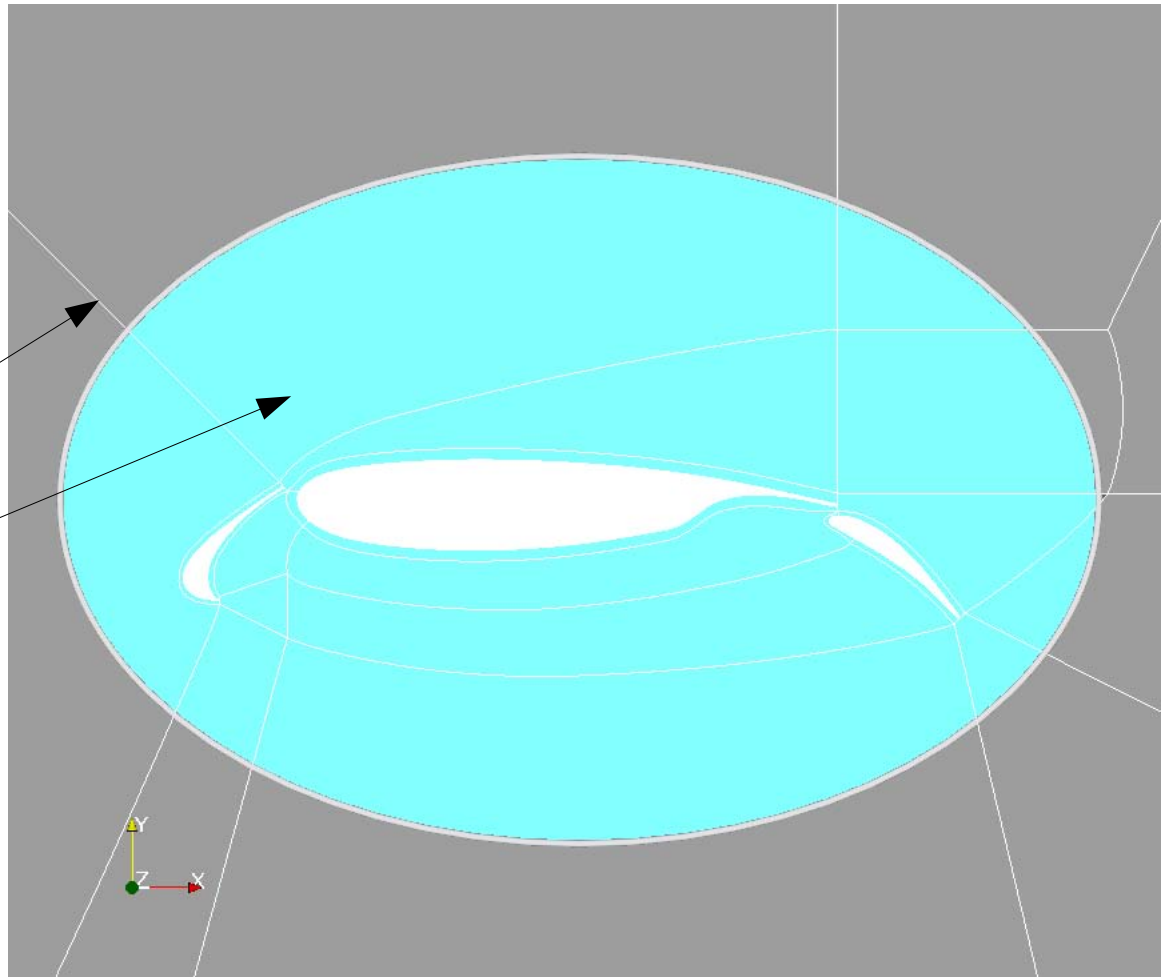
Zonal approach

Efficiency

d_{min}/d_{max}	10	10^3	10^6
N_{max}	8	512	524288
N_{ave}	1.5	10	1000

Buffer (CFL<<1)

ROI (CFL~1)



Zonal approach

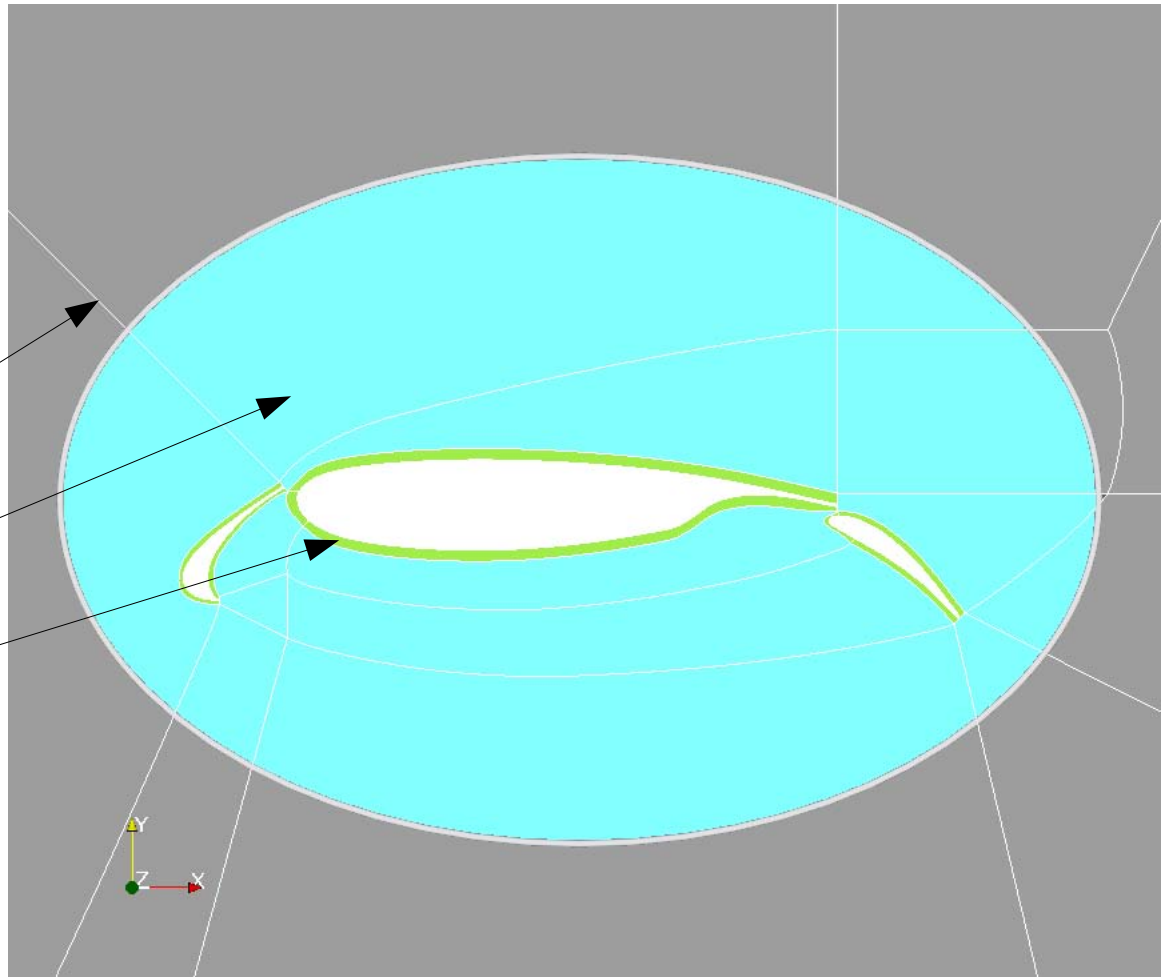
Efficiency

d_{min}/d_{max}	10	10^3	10^6
N_{max}	8	512	524288
N_{ave}	1.5	10	1000

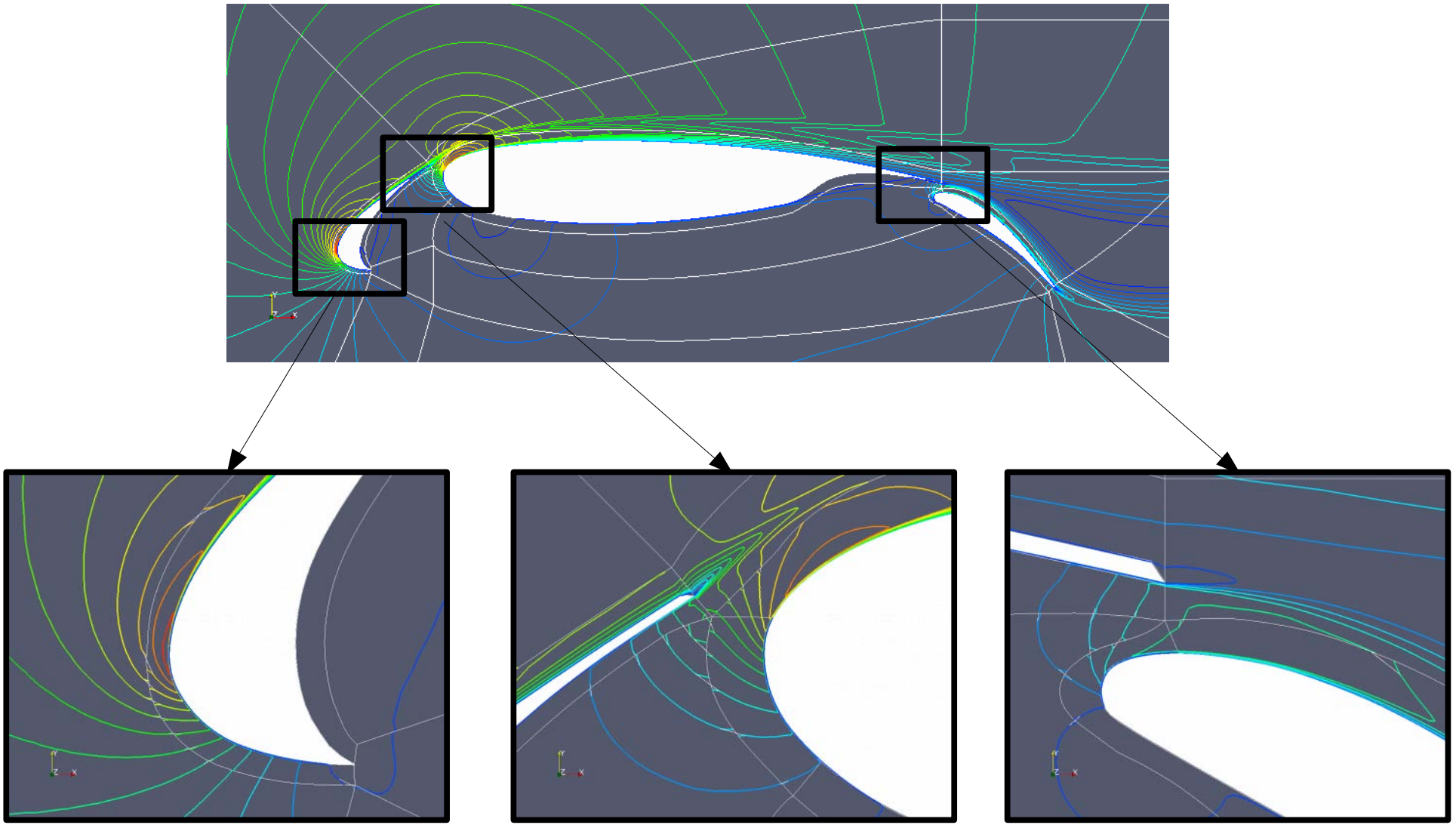
Buffer (CFL<<1)

ROI (CFL~1)

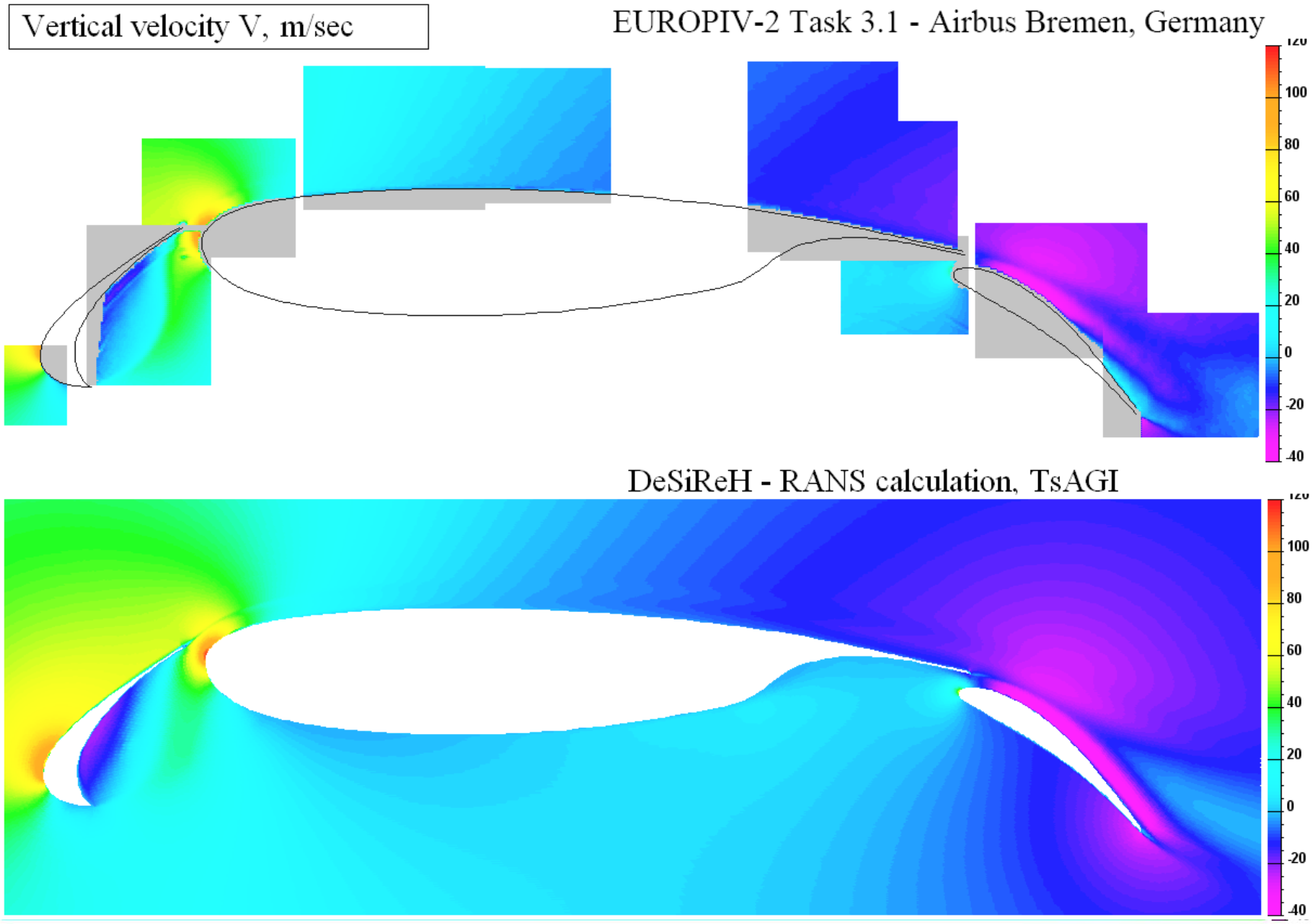
ROI (CFL>>1)



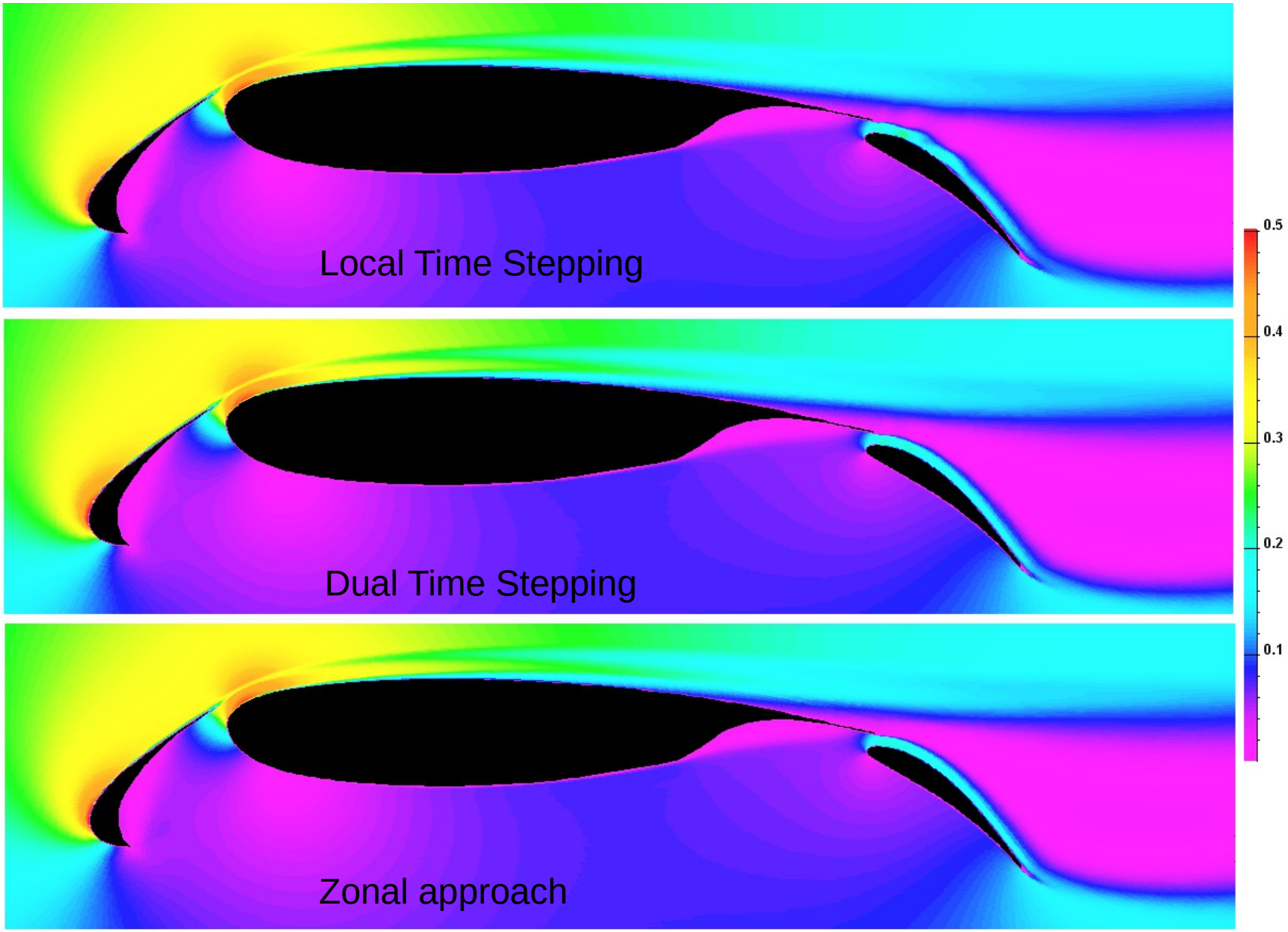
Zonal approach



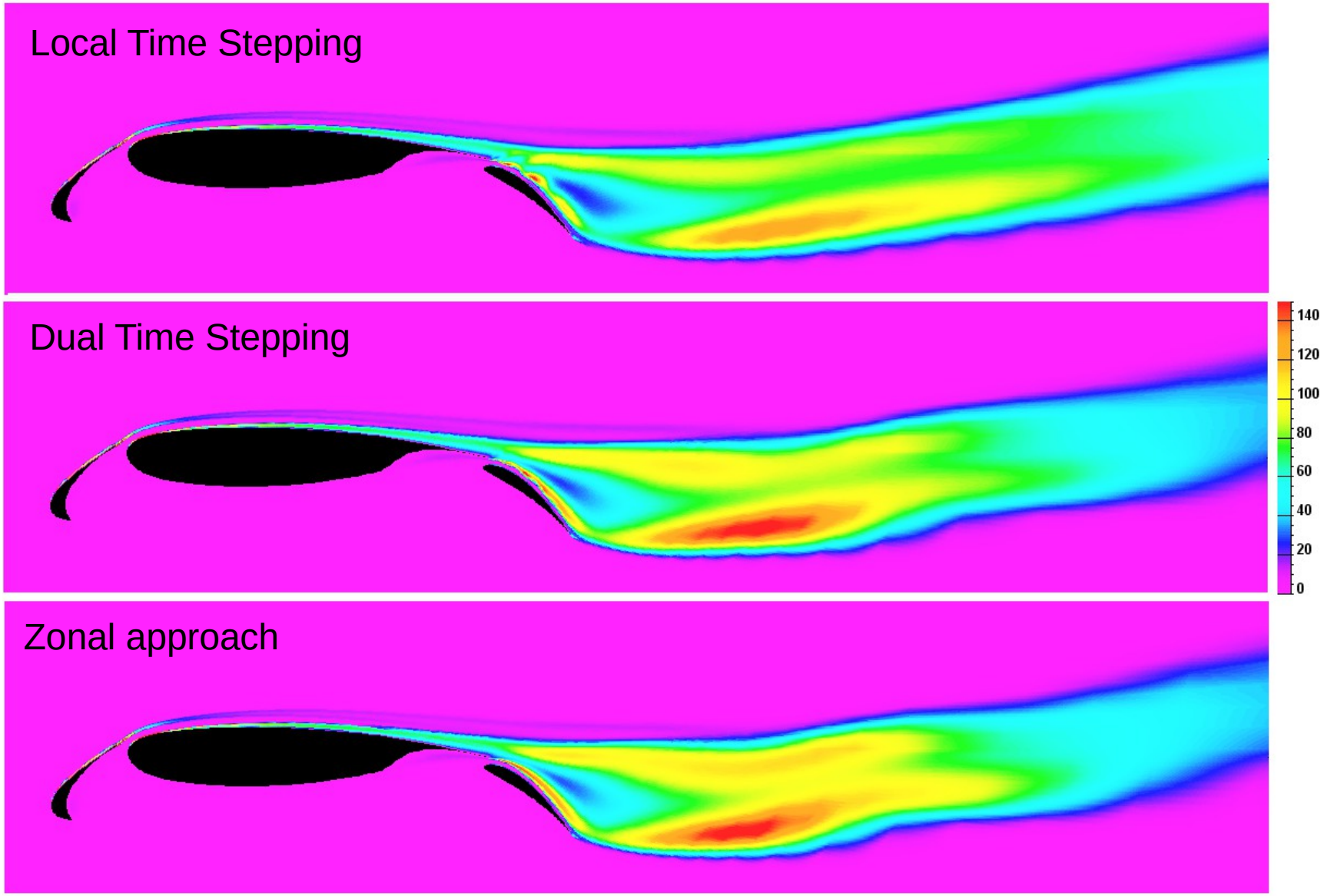
Vertical velocity (V)



Zonal approach

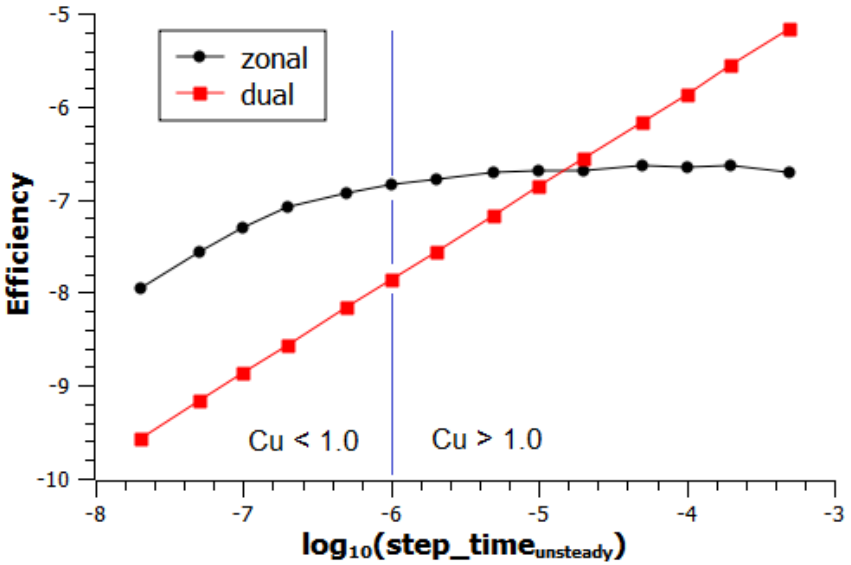


Zonal approach

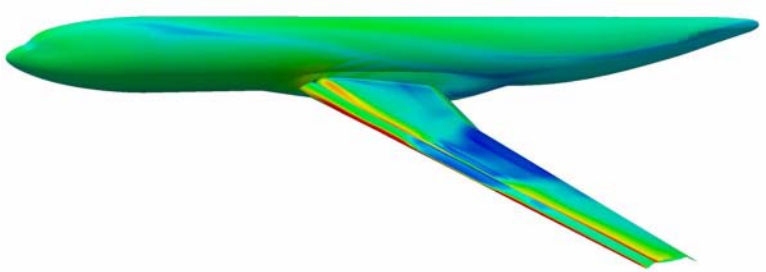


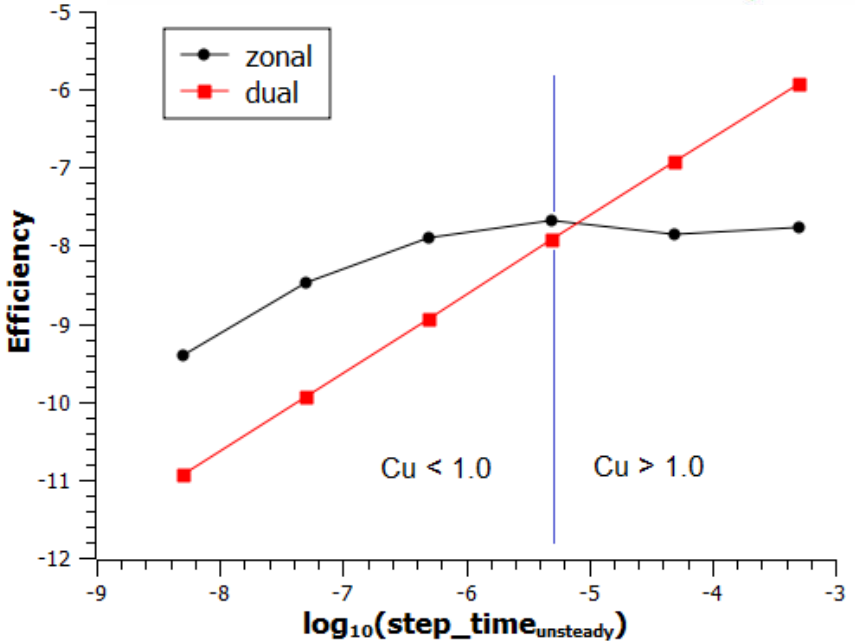
Computational efficiency of zonal approach

Airfoil EUROPIV2



Wing-Airframe TC217





The End