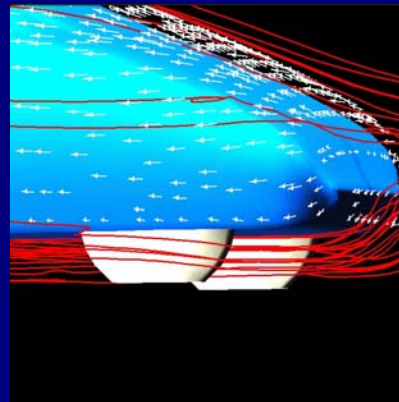


Fastflo Fluids ToolBox

Fastflo



Flexible finite element software for
the numerical solution of PDEs

Outline of presentation

- *Fastflo* - overview, strengths, features
- *Fastflo* and CFD
- two simple CFD algorithms
- overview of Fluids ToolBox
- examples based on Fluids ToolBox
- glimpse of advanced applications

Overview of *Fastflo*

- based on the finite element method, 2D and 3D
- range of element types (linear, quadratic; triangles, quadrilaterals, tetrahedra, hexahedra)
- internal mesh generator for 2D problems
- interface to commercial pre- and post-processors
- includes a high level macro command language to specify and solve PDEs
- graphical user interface

Overview of *Fastflo* (continued)

- selection of sparse matrix solvers (direct and iterative)
- Tutorial Guide, on-line Reference Manual
- many well-documented applications
- incorporates feedback from dozens of licensees
- Fluids ToolBox released with *Fastflo V3*
- available in PC and UNIX versions, both written in C. The PC GUI is built using Borland C++ and makes use of Windows facilities. The UNIX GUI is built using Motif.

Strengths of *Fastflo*

- can solve multiple PDEs
- flexible (in terms of geometry, equations, algorithms)
- (almost) any PDE can be solved
- is available and supported worldwide
- self-contained (mesh generation, graphics)
- programming environment that empowers users
- macro code provides open system for team use
- macro code works on all platforms
- able to specify and solve problems on boundaries
- very useful for rapid prototyping

Strengths of *Fastflo* (continued)

- moving meshes and free surfaces are possible
- inexpensive
- able to specify and solve problems in multiple regions

In addition, *Fastflo*

- has been road-tested through 40 person years of development
- has an emerging network of users, worldwide
- is CFD capable (especially for niche CFD)
- and is the best in the world in its class

Design features of *Fastflo*

- users present problems to *Fastflo* via two files:
 - *.msh which contains geometrical information;
 - *.prb which contains equations, boundary conditions, the algorithm, and commands to view the results.
- think of *Fastflo* as a workbench, with a number of powerful tools to specify and solve PDEs; the workbench offers graphics, editing and printing facilities.

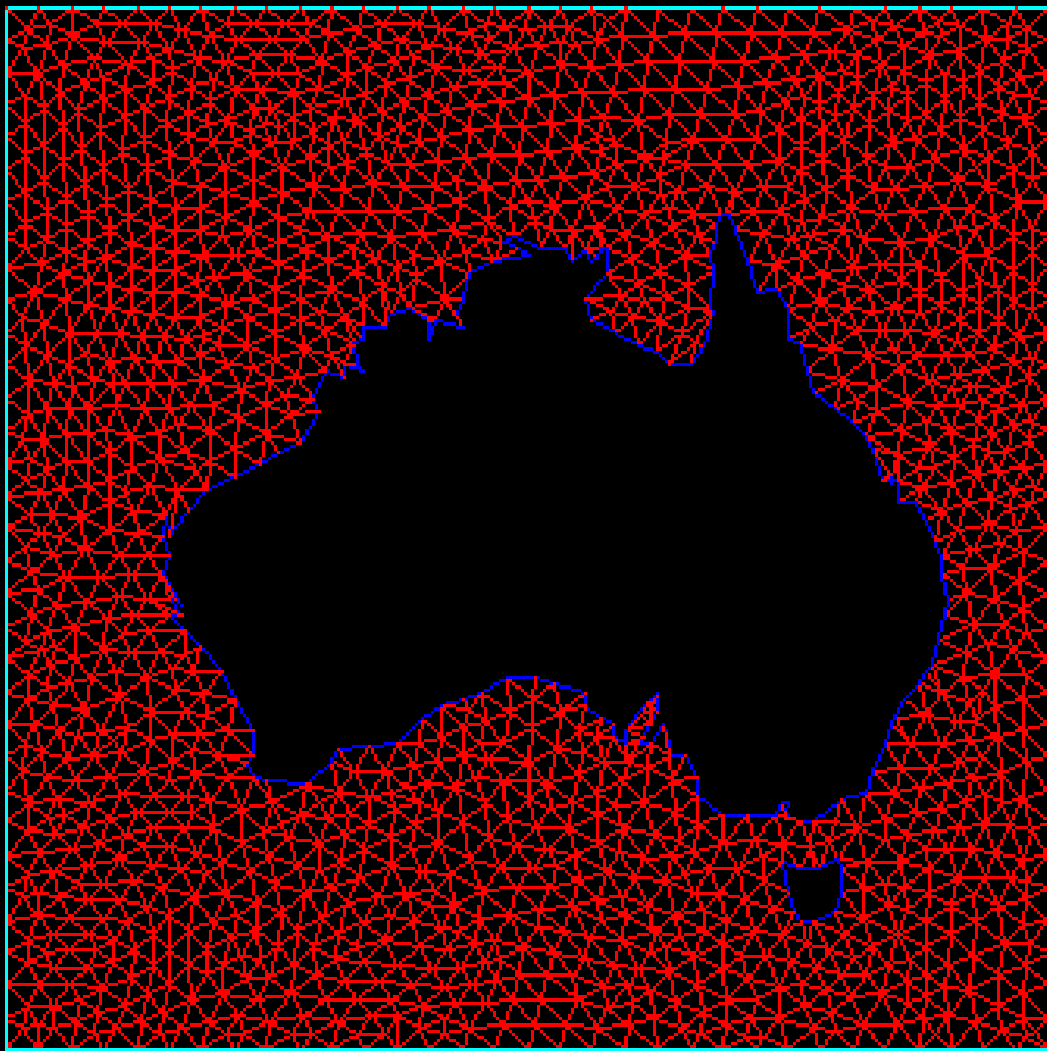
Design features of *Fastflo* (continued)

- *Fastflo* macro code is open and portable; there is no need for time-consuming low level programming
- users are free
 - # to specify what equation to solve
 - # to design the algorithm used for the solution
 - # to control the computations intelligently
- substantial help is available from an extensive list of examples

Design features of *Fastflo* (continued)

- it is possible to interface with third-party mesh generation and post-processing packages
- data is stored and manipulated in *Fastflo* on a vector stack
- *Fastflo* is designed to solve systems of PDEs; for time-dependent and nonlinear problems, the user is free to design the algorithm that will be applied
- the on-line Reference Manual provides the user with immediate help, including search facilities

Mesh generation



- * triangular mesh generator
- * linear and quadratic approx
- * 2D: triangles, quadrilaterals
- * 3D: tetrahedra, hexahedra
- * interface to third-party software
- * isoparametric elements
- * deformable boundaries
- * block mesh generator
- * axisymmetry

Derivative expressions

38 expressions hard-wired into the package

$$D_j A D_j U1 \quad - \nabla \cdot (a \nabla u)$$

$$A_j D_j U1_i \quad a \cdot \nabla u$$

$$D_i A D_j U1_j \quad - \nabla (a \nabla \cdot u)$$

1	$D_j A D_j U1$	$-\nabla \cdot (a \nabla u)$
2	$A U1$	au
3	$A_j D_j U1$	$a \cdot \nabla u$
4	$D_j A_j U1$	$-\nabla \cdot (au)$
5	$D_j A_{jk} D_k U1$	$-\nabla \cdot (A \nabla u)$
6	$D_j A U1_j$	$-\text{div}(au)$
7	$A D_j U1_j$	$a \text{ div } u$
8	$A_j U1_j$	$a \cdot u$
9	$D_j A_k D_k U1_j$	$-\text{div}(a \cdot \nabla u)$
10	$D_j A_j D_k U1_k$	$-\text{div}(a \text{ div } u)$
11	$D_j A_{jk} U1_k$	$-\text{div}(A u)$
12	$A_{jk} D_j U1_k$	$\text{div}(A u)$
13	$D_i A U1$	$-\nabla (au)$
14	$A D_i U1$	$a \nabla u$
15	$A_i U1$	au
16	$D_i A_j D_j U1$	$-\nabla (a \cdot \nabla u)$
17	$D_j A_j D_i U1$	$-\mathbf{a} \cdot \nabla (\nabla u) - (\nabla u) \cdot \nabla \mathbf{a}$
18	$D_j A_{ji} U1$	$-\nabla \cdot (A u)$
19	$A_{ij} D_j U1$	$A \nabla u$
20	$A U1_i$	au
21	$A_j D_j U1_i$	$a \cdot \nabla u$
22	$D_j A_j U1_i$	$-\mathbf{a} \cdot \nabla u - u \text{ div } \mathbf{a}$
23	$D_j A D_j U1_i$	$-\nabla \cdot (a \nabla u)$
24	$D_j A_{jk} D_k U1_i$	$-\nabla \cdot (A \nabla u)$
25	$D_i A D_j U1_j$	$-\nabla (a \nabla \cdot u)$
26	$D_i A_j U1_j$	$-\nabla (a \cdot u)$
27	$D_j A D_i U1_j$	$(\nabla a) \cdot \nabla (\text{div } u) - \nabla (\text{div } au)$
28	$A_j D_i U1_j$	$a \cdot (\nabla u)$
29	$D_j A_i U1_j$	$-\mathbf{a} (\nabla \cdot u) - u \cdot \nabla \mathbf{a}$
30	$A_i D_j U1_j$	$\mathbf{a} (\nabla \cdot u)$
31	$A_{ij} U1_j$	$A u$
32	$D_i A_{jk} D_j U1_k$	$-\nabla \cdot (A \nabla u)$
33	$D_j A_{jk} D_i U1_k$	
34	$D_j A_{ik} D_k U1_j$	
35	$D_j A_{ij} D_k U1_k$	
36	$D_j A_{ik} D_j U1_k$	
37	$D_j A_k D_j U1_k$	$-\text{div } a \nabla u$
38	$D_j A_i D_j U1$	$-\text{div } a \nabla u$

Fastflo and Computational Fluid Dynamics

- successful implementations of the following formulations: penalty, augmented Lagrangian, streamfunction - vorticity, artificial compressibility
- extensive experience with an operator-splitting algorithm (Glowinski θ -scheme)
- successful investigations using *Fastflo*: two-equation turbulence models, free surface computations, non-Newtonian flow, weakly compressible flow, two-phase flow, porous media flow, plastic deformations, buoyancy

Two simple CFD algorithms applied to the driven cavity problem

These examples demonstrate:

- two simple CFD algorithms (penalty, augmented Lagrangian)
- nonlinearity in the Navier-Stokes equation
- convergence using Picard iteration
- simple mesh generation and coding

2D Driven Cavity Problem (continued)

```
400 0 4
-2 0 0 0 0
3 4
```

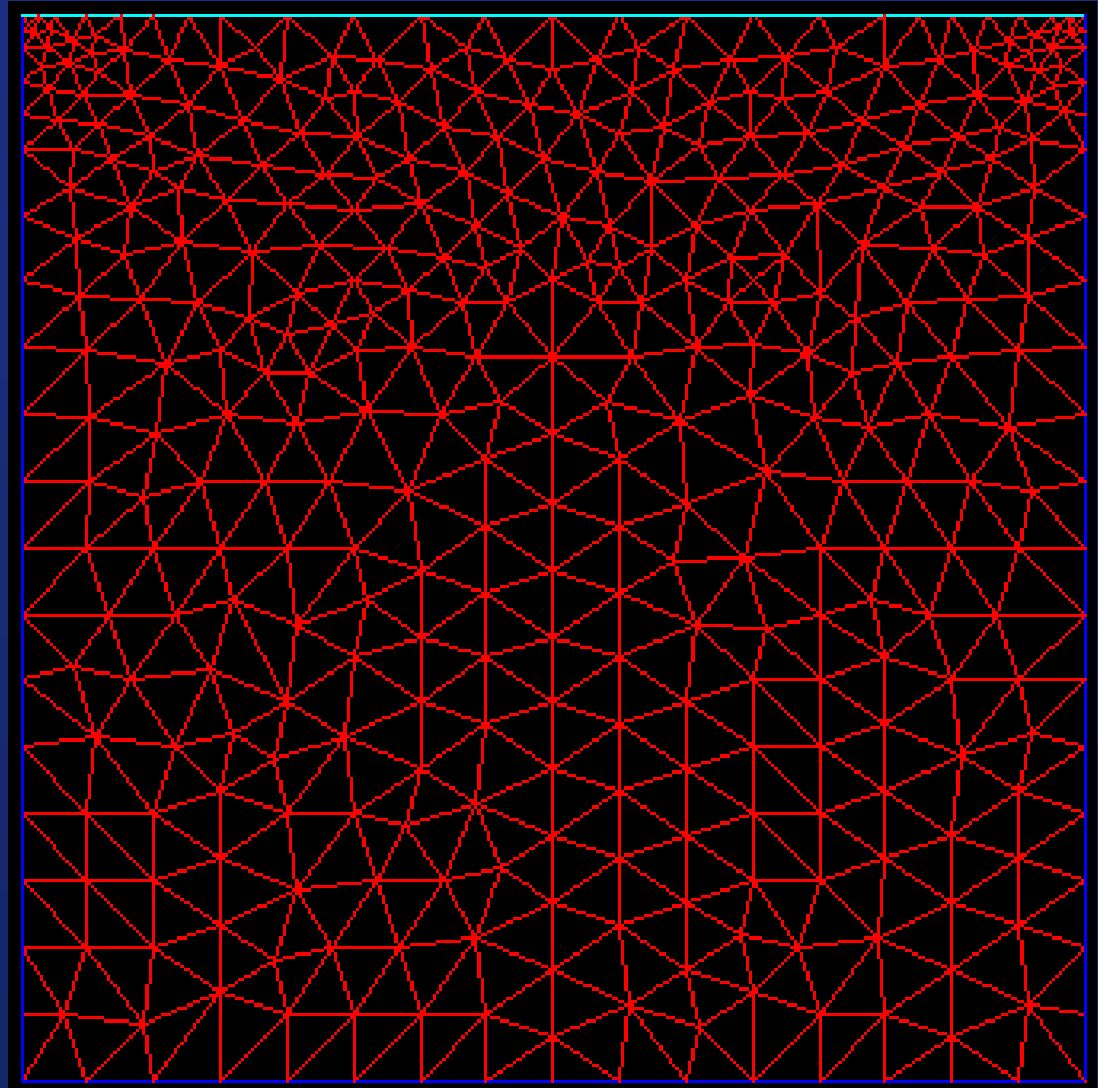
List of vertices

```
0.0 0.0
1.0 0.0
1.0 1.0
0.0 1.0
```

List of boundary tags

```
1 1
2 1
3 2
4 1
```

End of boundary list

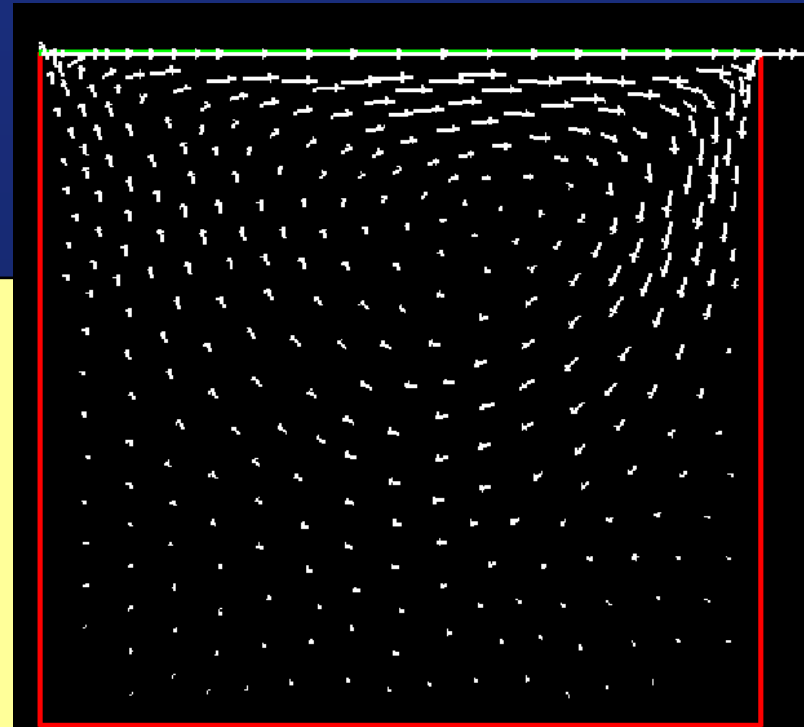


Penalty formulation

$$-\text{Pen} \nabla(\nabla \cdot \mathbf{v}_n) - \nabla^2 \mathbf{v}_n + \text{Re} \mathbf{v}_{n-1} \cdot \nabla \mathbf{v}_n = 0$$

```
% cav_pen.prb
P Re 100.0
P Pen 3000.0
A pen
e D_i { Pen} D_j U1_j + D_j D_j U1_i \
= {V100 * Re}_j D_j U1_i
b 1 U1 = {0.0, 0.0}
b 2 U1 = {1.0, 0.0}
< pic
test=100.
while test>1.0
  pen
  solve
```

```
V400=V200-V100
test=L1 V400
show test
black
arrow
endwhile
```



Augmented Lagrangian formulation

$$-\text{Pen} \nabla(\nabla \cdot \mathbf{v}_n) + \nabla p_{n-1} - \text{Re}^{-1} \nabla^2 \mathbf{v}_n + \mathbf{v}_{n-1} \cdot \nabla \mathbf{v}_n = 0$$

$$\delta p = p_n - p_{n-1} = \text{Pen} \nabla \cdot \mathbf{v}_n$$

```
% cav_aug.prb
P Re 200.
P Pen 20.
P iter 0
P nstep 5
```

```
A augment
e D_i{Pen}D_jU1_j + D_j{1.0/Re}D_jU1_i \
= {V100}_jD_jU1_i + D_i{V201}
b 1 U1={0.0,0.0}
b 2 U1={1.0,0.0}
A [reduced] delp
e U1={Pen}D_j{-V100}_j
< run
  while iter<nstep
    augment
    solve
```

```
V500 = V200-V100
test=L1 V500
show test
delp
solve
expand 1
V401=V401+V101
popp
V200=V100
black
arrow
popp
iter=iter+1
endwhile
```

>

Features of these algorithms

NOTE: neither of these solvers is recommended for large-scale problems

- penalty method does not solve the actual problem
- if it converges, the augmented Lagrangian method does solve the correct problem
- a lot of memory required for big problems
- both methods can be adapted to timestepping
- augmented Lagrangian method calculates the pressure; the penalty method does not

Scope of the *Fastflo* Fluids ToolBox

- incompressible, viscous flows in 2D and 3D
- laminar flow at appropriate Reynolds numbers
- turbulence models (linear k - ε , nonlinear k - ε , RNG)
- three boundary treatments (no slip, mixing length effective viscosity, logarithmic wall functions)
- time-dependent flows, with options of pseudo-transient convergence to the steady state
- user-defined body forces (e.g. convection)

What you ought to know to make a start

- theory of fluid mechanics for incompressible flow
- theory of PDEs and algorithms for their solution
- a working knowledge of the finite element method
- use of *Fastflo* to specify problems and develop algorithms to solve them
- post-processing in *Fastflo*
- use of the *Fastflo* graphical user interface

User interface to the Fluids ToolBox

Fastflo Laminar/Turbulence Flow LAMINAR/TURBULENCE FLOW TOOLBOX

Open Save Exit Help

Current File:

Options

Geometry	Solver	Time Stepping
<input type="text" value="Plane TwoD"/>	<input type="text" value="Direct"/>	<input type="text" value="Fixed dt"/>
Job	Laminar/Turbulence	Wall Function
<input type="text" value="New Job"/>	<input type="text" value="Laminar flow"/>	<input type="text" value="No-Wall-Func"/>

Boundary Conditions

Tag: Type: Value: Accept

Parameters

Viscosity:	<input type="text" value="0.01"/>	Time step dt:	<input type="text" value="0.01"/>
Num. of steps	<input type="text" value="20"/>	Maximum dt:	<input type="text" value="0.1"/>
Initial vel.	<input type="text" value="0.,0.,0."/>	Save Interval	<input type="text" value="12"/>
Pic. Interval	<input type="text" value="1"/>	Print Out:	<input type="text" value="0"/>
Steady Test	<input type="text" value="0.001"/>		

Turbulent flow over a backward facing step (2D)

- linear k - ε method
- van Driest wall condition
- converges to the steady state in 237 timesteps
- this example uses a structured mesh
- boundary conditions
 - tag 1 (walls): no slip
 - tag 2 (inlet): $(u,v,0)$, k , ε , ψ
 - tag 3 (outlet): zero stress
- all settings assigned within user interface

Backward facing step (2D) - settings

```
#LaminarTurbulenceToolBoxFile bfst.box
```

```
=====Options=====
```

```
Geometry: Plane TwoD
```

```
Solver: Direct
```

```
Time Stepping: Variable dt
```

```
Job : New Job
```

```
Turbulence Model: Linear k-e
```

```
Wall Function: van Driest
```

```
=====Boundary Conditions=====
```

```
Tag 1 has No-Slip BC with Value= N/A
```

```
Tag 1 has Stream Function BC with Value=  $2*(X2>1.5)$ 
```

```
Tag 2 has Velocity BC with Value=  $(X2<2.999)*(X2>1.001),0.$ 
```

```
Tag 2 has Stream Function BC with Value=  $X2-1$ 
```

```
Tag 2 has K-Inlet BC with Value= 0.003
```

```
Tag 2 has Epsilon-Inlet BC with Value= 0.00364
```

```
Tag 3 has Outlet BC with Value= N/A
```

```
=====Control Parameters=====
```

```
Viscosity: 0.000022
```

```
Time step dt: 0.05
```

```
Num. of steps: 250
```

```
Maximum dt: 0.4
```

```
Initial Vel. 0.,0.,0.
```

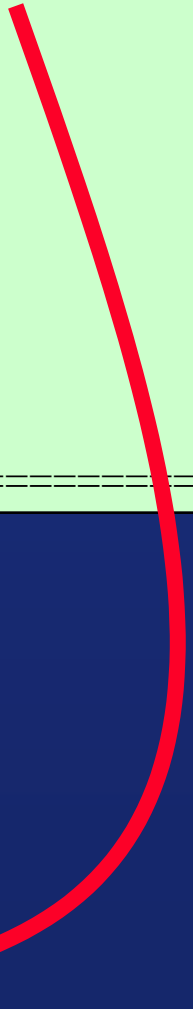
```
Save interval: 100
```

```
Pic. interval: 1
```

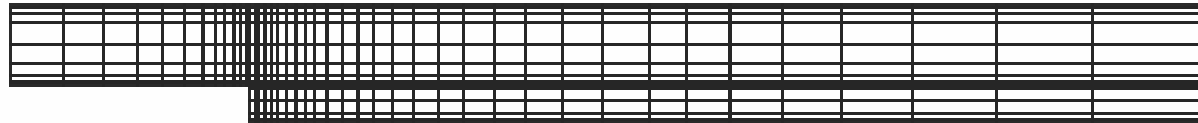
```
Print out: 0
```

```
Steady Test: 0.001
```

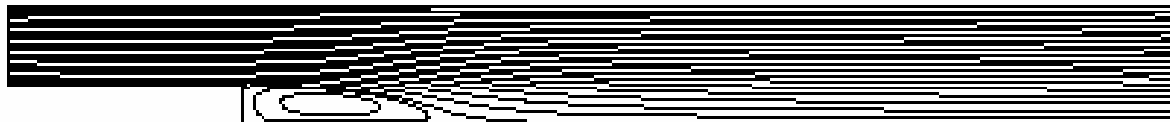
```
=====End of File=====
```



Backward facing step (2D) - results



mesh



ψ



k



ϵ



v_t

Flow of salty liquid down an inclined plane into a stratified fluid

- laminar flow, time dependent, 2D
- buoyancy force given by Boussinesq approximation
- needs an auxiliary equation in each timestep for salt concentration
- this example uses an unstructured mesh
- boundary conditions
 - tag 1 (walls): no slip, $\nabla C \cdot \mathbf{n} = 0$
- all settings assigned within user interface
- need auxiliary programming for body force

Dimensionless equations

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\nabla p + (\sigma / \text{Ra})^{0.5} \nabla^2 \mathbf{v} - (1 + \alpha C) \hat{\mathbf{e}}_2$$

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

$$\frac{\partial C}{\partial t} + \mathbf{v} \cdot \nabla C = 1 / (\sigma \cdot \text{Ra})^{0.5} \nabla^2 C$$

Ra = Rayleigh number, σ = Schmidt number

Backward Euler formulation for C

$$\frac{C^{n+1} - C^n}{\Delta t} + \mathbf{v} \cdot \nabla C^n = 1/(\sigma \text{Ra})^{0.5} \nabla^2 C^{n+1}$$

```
P BodyForce 1
P Schmidt 13600
P Rayleigh 6.06E14
P ksalt 1/sqrt(Rayleigh*Schmidt)
P Viscosity sqrt(Schmidt/Rayleigh)
A salt
e {1.0/dt}U1 - D_j{ksalt}D_j U1 + {V200}_jD_j U1 \
= {1.0/dt}{V301}
```

< UserFunc

```
salt 1 200 V200 301 V2001
solve
V2001=V101
black
contour
```

>

```
< bodyforce
V401=0.0
V402=-V2001
>
< init
V2001=(X2<.25)*(0.0025*(X1<.321)+0.005*(X1>.32))
>
```

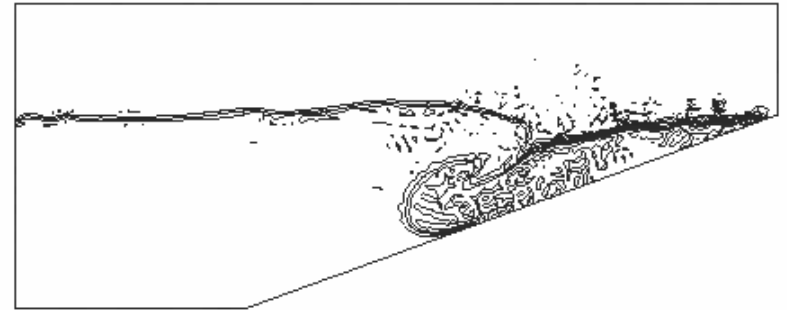
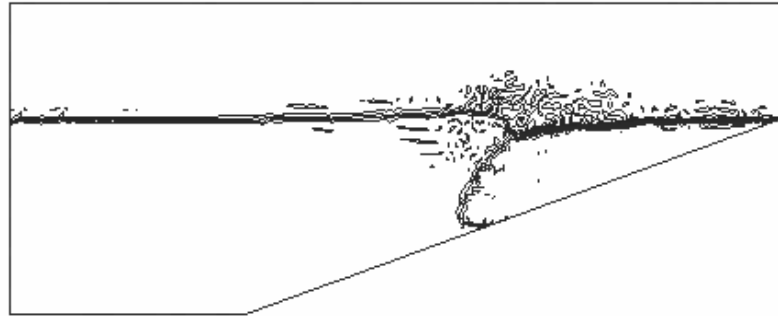
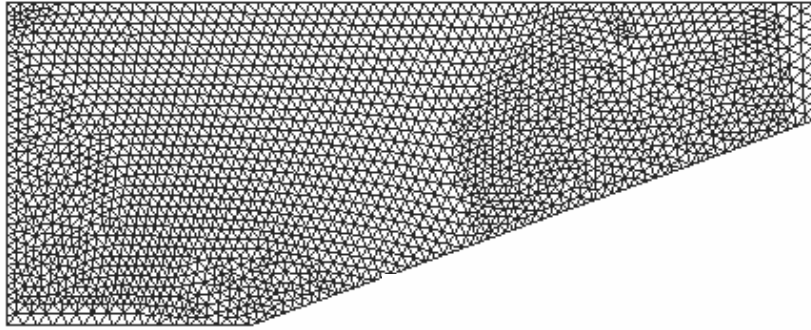
for initial
conditions

Flow of salty liquid down an inclined plane into a stratified fluid - settings

```
#LaminarTurbulenceToolBoxFile ramp.box
=====Options=====
Geometry: Plane TwoD
Solver: Direct
Time Stepping: Fixed dt
Job : New Job
Turbulence Model: Laminar flow
Wall Function: No-Wall-Func
=====Boundary Conditions=====
Tag 1 has No-Slip BC with Value= N/A
```

```
=====Control Parameters=====
Viscosity: 1
Time step dt: 0.3
Num. of steps: 80
Maximum dt: 0.3
Initial Vel. 0.,0.,0.
Save interval: 12
Pic. interval: 1
Print out: 0
Steady Test: 0.001
=====End of File=====
```

Flow of dense salty liquid - results



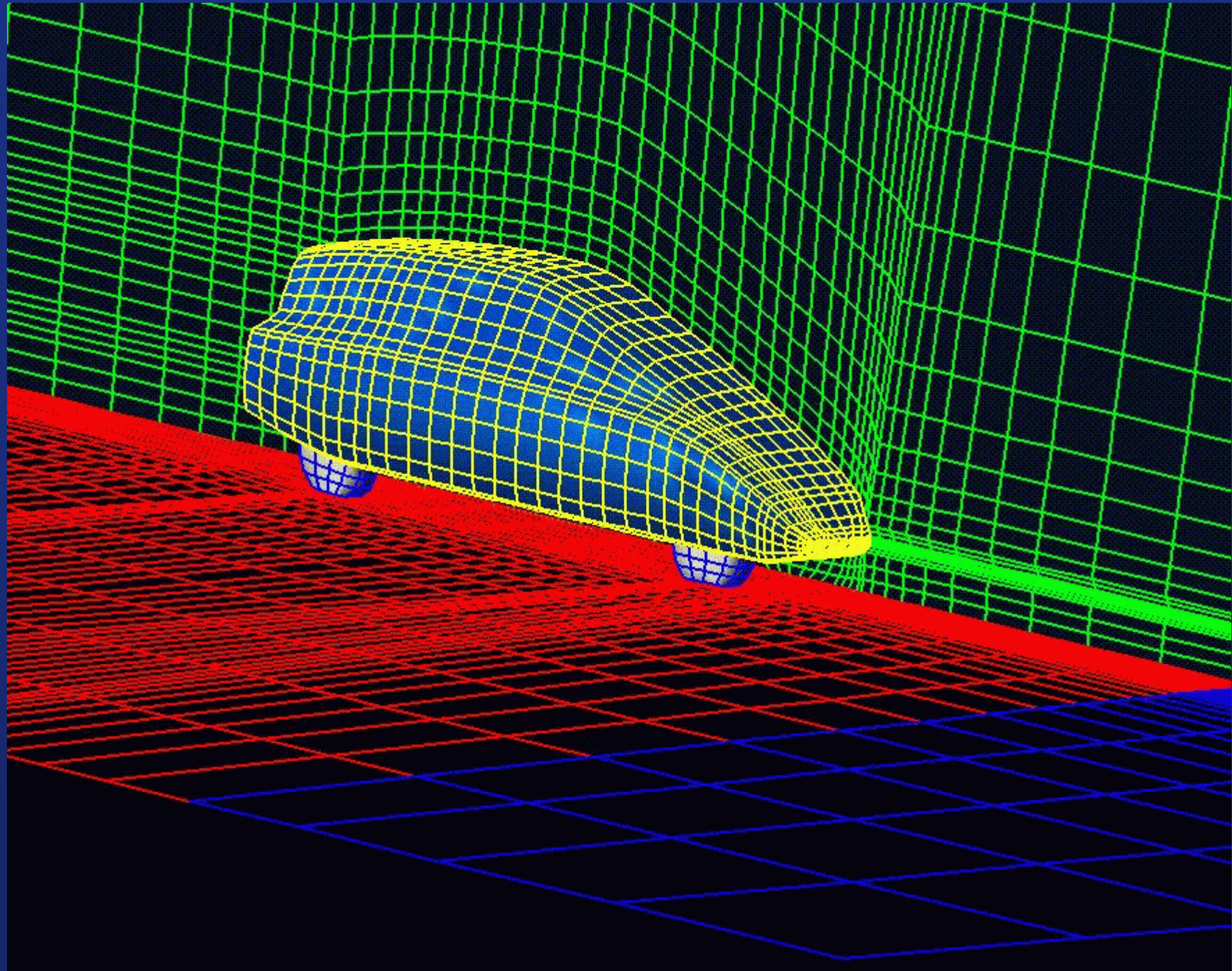
Contour lines for C for
 $t = 12$ and 24 (40 and
80 timesteps)

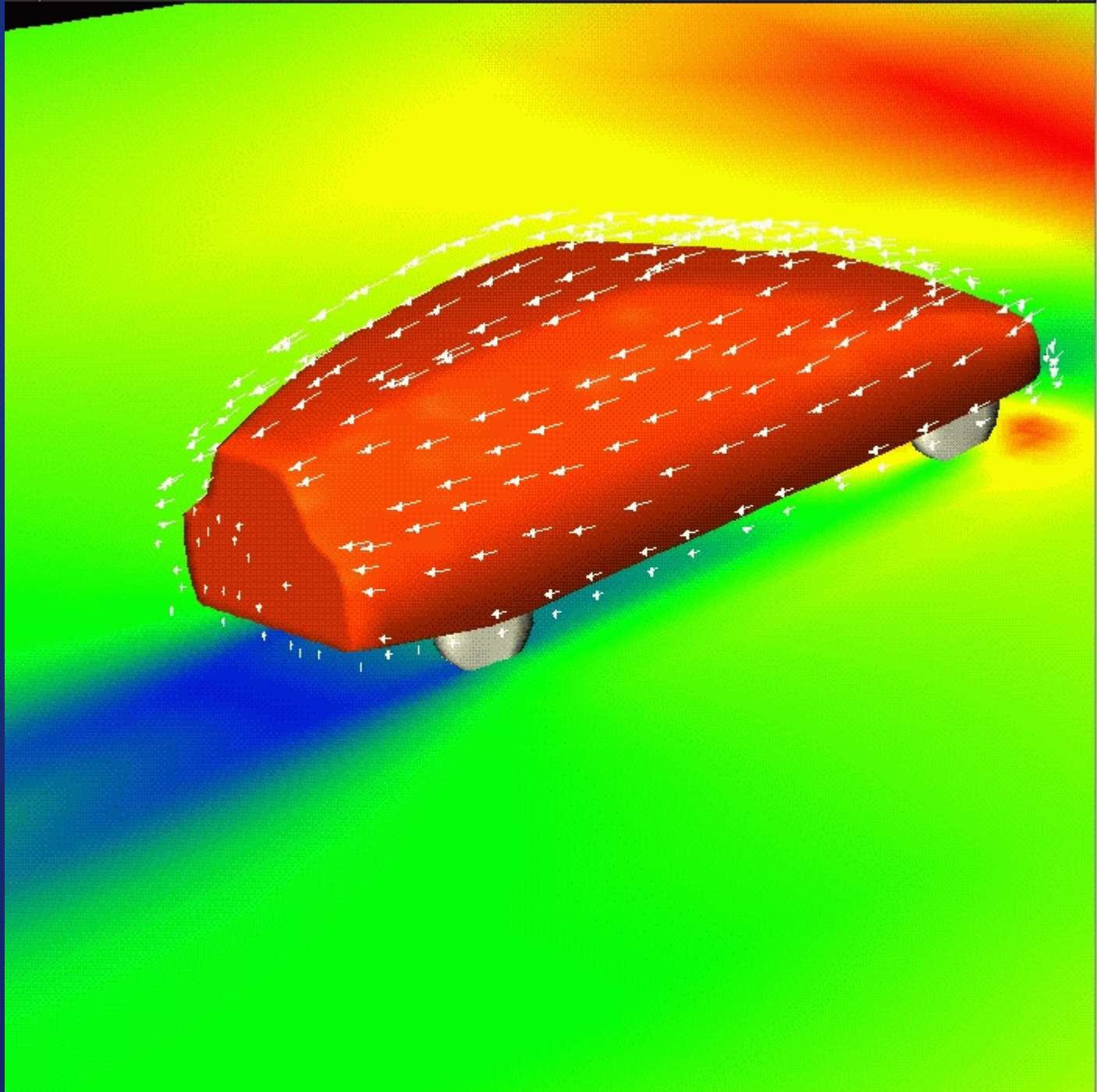
Fastflo case studies for CFD

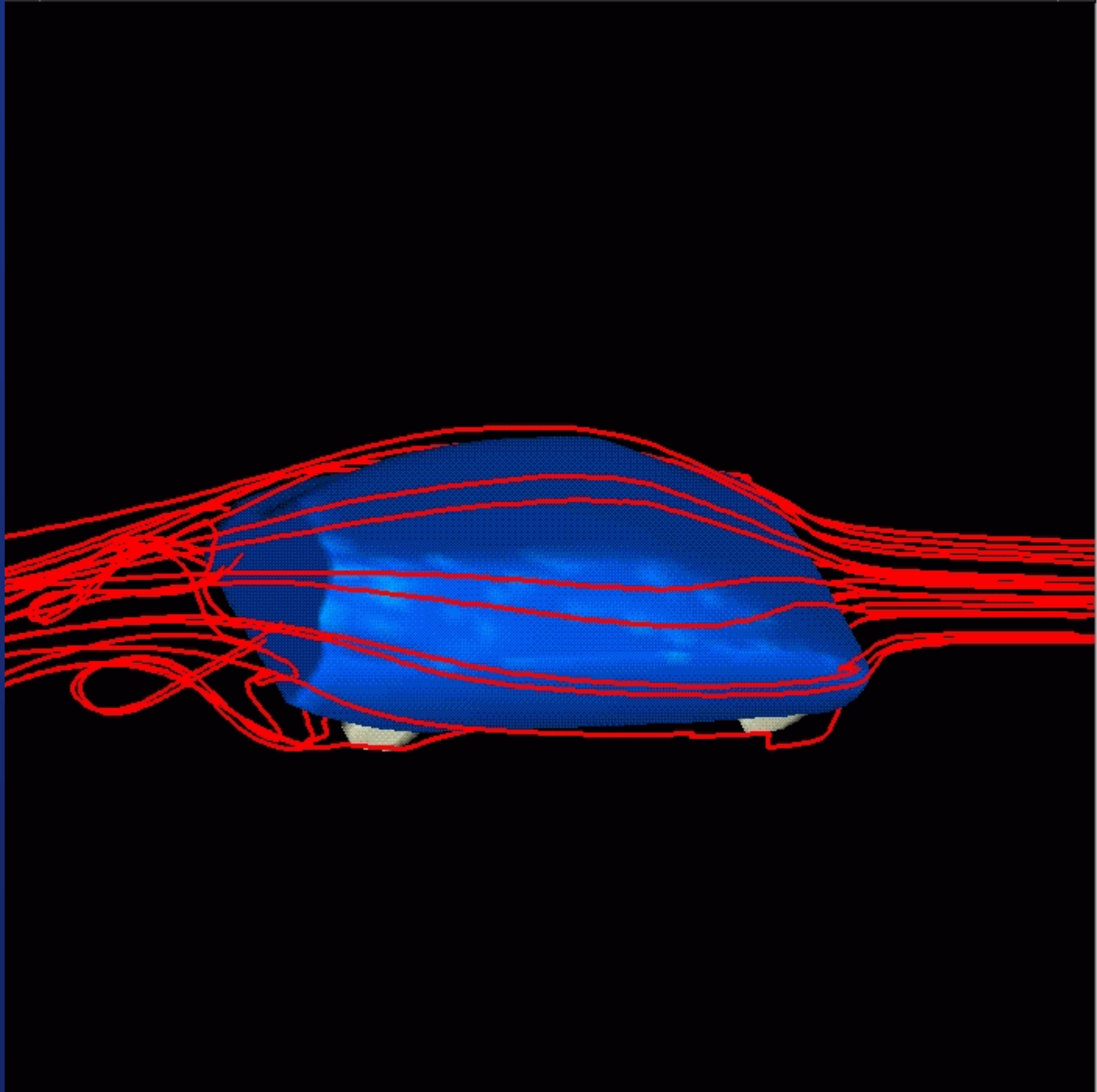
- viscoelastic flow around a sphere in a tube
- convective cooling of heated plates
- turbulent flow in axisymmetric U-bend
- 3D turbulent flow of molten steel
- swirling turbulent combustion in a burner
- free surface flow in double roll coater
- turbulent air flow around a car
- two-phase flow in a stirred tank

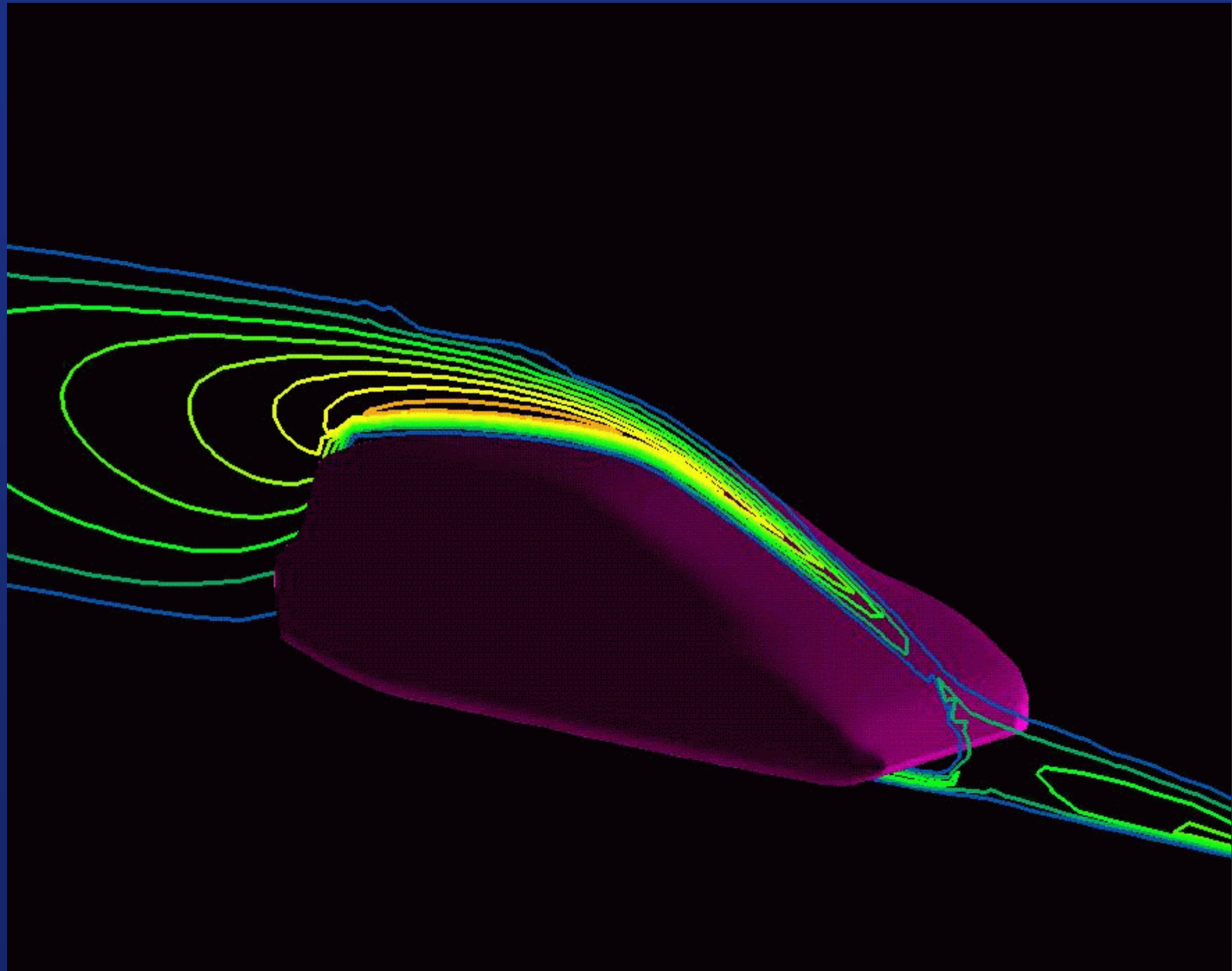
Turbulent Flow around a Model Automobile

- 3D steady turbulent airflow around model car
- 1996 benchmark by WUA-CFD and Daimler-Benz
- meshed by in-house mesh generator
- k-epsilon turbulence model; treatment of boundary elements by von Driest model
- solved using operator-splitting algorithm
- 85,542 nodes , converged in 30 timesteps
(66 hours, 175 MHzDEC workstation)
- excellent results (top surface, drag coefficient)









Summary

- *Fastflo* - features and strengths
- overview of *Fastflo* Version 3
- two simple CFD algorithms in *Fastflo*
- capabilities of *Fastflo* Fluids ToolBox
- two examples of use of the ToolBox
- industrial strength CFD - case studies

Any questions?