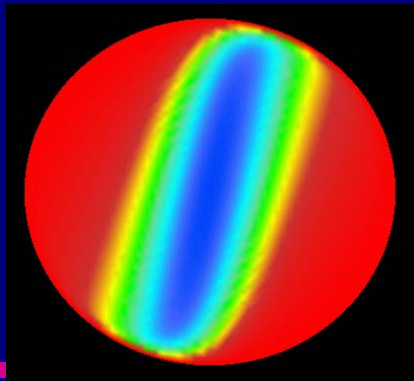


# Solution of PDEs using *Fastflo*

CSIRO Mathematical and Information Sciences  
Clayton, Australia

*Fastflo*



Flexible finite element software for  
the numerical solution of PDEs

# Outline of presentation

- *Fastflo* – summary of main features
- Key steps in setting up and solving PDEs  
(mesh, model, algorithm, coding, ...)
- two simple PDE problems  
acoustic scattering, thermoelasticity
- Where/how to find out more
- Questions

# 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 more than 40 person years of development
- has an emerging network of users, worldwide

# 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.

# 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
- data is stored on a vector stack (user-accessible)
- we think of *Fastflo* as a workbench, with tools to specify and solve PDEs; the workbench offers graphics, editing and printing facilities.

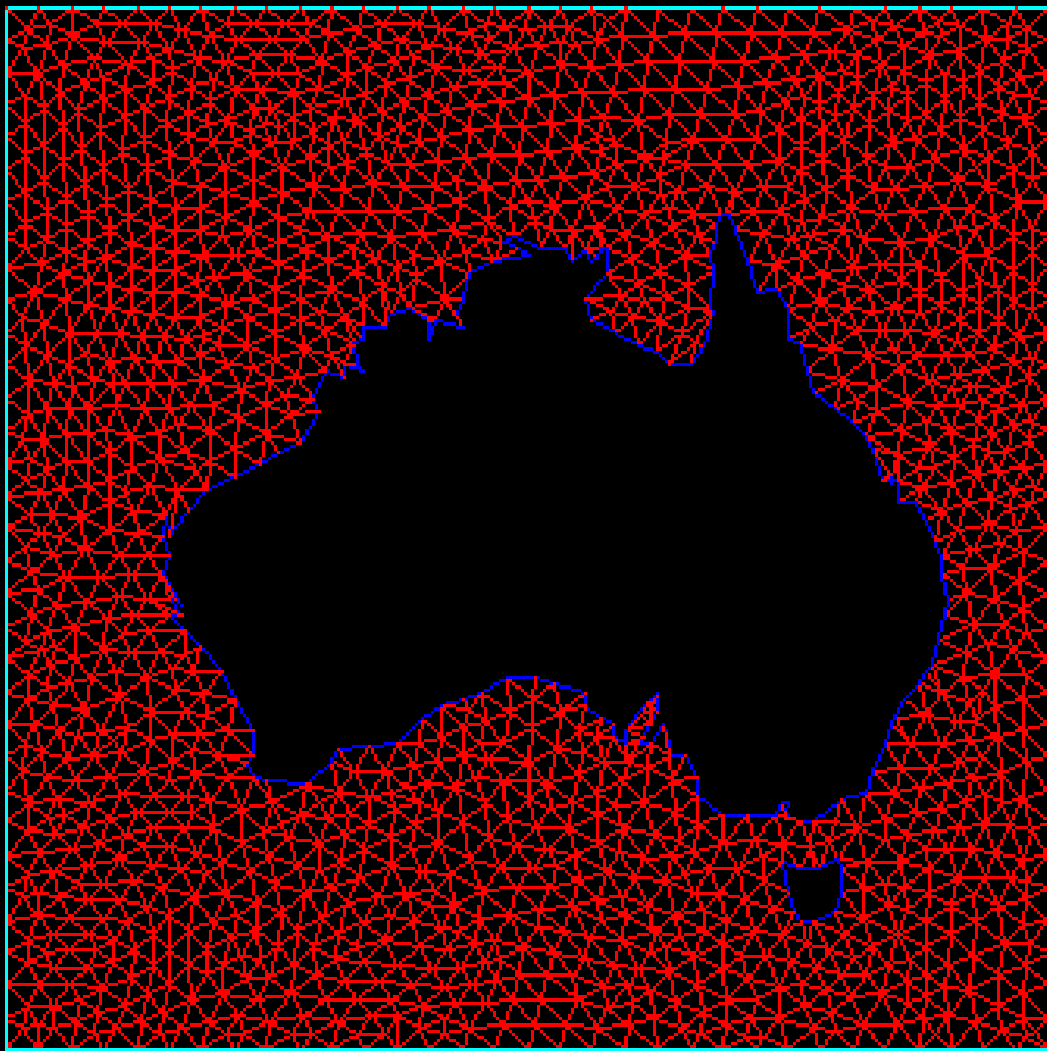
## Design features of *Fastflo* (continued)

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

# Key steps in setting up and solving PDE problems

- mesh generation
- model equations
- algorithm (especially timestepping, iteration, boundary conditions)
- coding of the algorithm
- running the code
- postprocessing
- mesh refinement, tuning of the algorithm

# Mesh generation



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

# Model equations – user to provide!

- most applications are to elliptic or parabolic PDEs of second order, or to systems of such equations (e.g. thermoelasticity, Navier-Stokes equations)
- boundary conditions must also be provided by user; generally best to express the model in the form of a conservation law
- multiple domains can be handled; moving and deformable boundaries can be handled (expert users can set up ODEs or lower dimensional PDEs on boundaries)

# Algorithm

- always reduce the problem to a set of linear equations, which the FE representation reduces to a large sparse system
- for nonlinear problems – introduce iterative scheme, typically Picard or Newton
- for time-dependent problems – introduce suitable timestepping scheme; implicit schemes (e.g. Crank-Nicolson, Backward Euler) are most commonly used

# Boundary conditions

- need to understand principles of FE method, which involves integration by parts
- example – the heat equation

$$\text{equation: } \rho c \frac{\partial \theta}{\partial t} = \nabla \cdot (k \nabla \theta)$$

$$\text{backward Euler scheme: } \left( \frac{\rho c}{dt} \right) (\theta^n - \theta^{n-1}) = \nabla \cdot (k \nabla \theta^n)$$

$$\left( \frac{\rho c}{dt} \right) \theta^n - \nabla \cdot (k \nabla \theta^n) = \left( \frac{\rho c}{dt} \right) \theta^{n-1}$$

natural boundary conditions are applied to  $k \nabla \theta$   
(which results from integration by parts)

# Boundary conditions (continued)

The principal options are:

- do nothing – equivalent to natural boundary expression is zero (e.g. zero heat flux)
- supply alternative value/function for natural boundary expression (e.g. non-zero heat flux)
- apply Dirichlet condition (e.g. temperature)

$$U1 = \{ \text{expression} \}$$

# Coding the algorithm

The principal steps usually are ...

- declare parameters
- define “problems” = equations + BC’s
- type the name of the problem to assemble the FE system; type solve to solve the sparse matrix
- generally, manage the computations (e.g. assembly solving, timestepping, iteration, error control, graphics, file management, ...) within “macros”

# 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	$D_j A_i D_j U1$	$-\text{div } a \nabla u$

# Trivial example – the heat equation

```
P rho 8900 % kg/m^3
P c 400 % J/(kg.K)
P k 403 % W/(m.K)
P dt 0.01 % s
P tend 1.0 % s
P rcondt rho*c/dt
A heateq
e {rcondt}U1 - D_j{k}D_j U1 = {rcondt}{V101}
b 1 U1 = {400}
b 2 curvature = {0.2071} % implicitly, no heat flux at 2
< run
    t = dt
    prim
    V101 = 0
    while t < tend
        heateq
        solve
        show V101
        black
        contour
        t = t + dt
    endwhile
>
```

# *Fastflo* examples



These simple examples are presented to illustrate the following strengths of *Fastflo*:

- can solve multiple PDEs
- flexible (in terms of geometry, equations, algorithms)
- (almost) any PDE can be solved
- self-contained (mesh generation, graphics)
- can solve problems in multiple domains
- programming environment that empowers users
- macro code provides open system for team use and works on all platforms

# Scattering of Acoustic Waves by a Cylinder

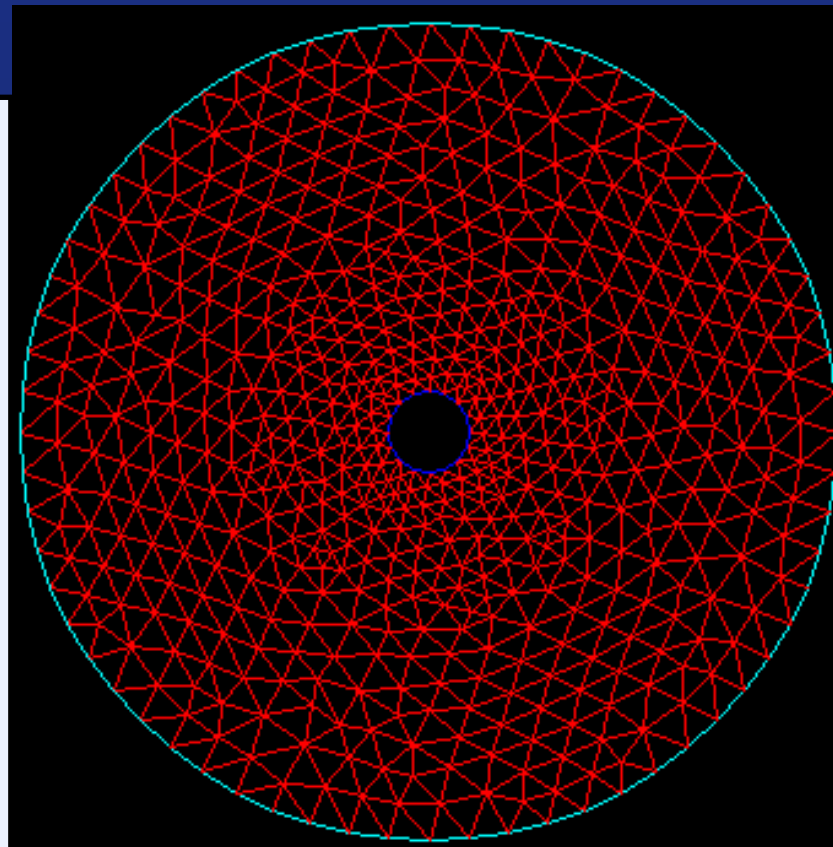
This example involves:

- two (Helmholtz) equations coupled by the Sommerfeld radiation condition at the far boundary
- no acoustic velocity at the cylinder
- straightforward code, easily extendable to more complex geometries and 3D

# Scattering of Acoustic Waves by a Cylinder (continued)


```
600 0 4
1 0 0 0
0.0 0.0
List of vertices
-10.0 -10.0
10.0 -10.0
10.0 10.0
-10.0 10.0
-1.0 -1.0
1.0 -1.0
1.0 1.0
-1.0 1.0
```

```
List of boundary tags
1 2 % outer circle
2 2
3 2
4 0
8 1 % inner circle
7 1
6 1
5 1
8 0
4 2
End of boundary list
```



# Acoustic Scattering (cont'd)

```
% scatter1.prb
P k 1.0
P alpha 3.1416/4.
P k1 cos(alpha)
P k2 sin(alpha)
P Rin 1.414
A scat
e D_jD_jU1_i+{k*k}U1_i=0
b 1 V301={k1*X1+(k2*X2)}
b 1 {(V301/Rin)*sin(V301), \
(-V301/Rin)*cos(V301)}_i
b 2 -{0.,k,-k,0.}_ijU1_j
b 1 curvature={-0.2071}
b 2 curvature={0.2071}
```



$$\nabla^2 u + k^2 u = 0$$

$$u = u_R + i u_I$$

$$\mathbf{n} \cdot \nabla (u^i + u^s) = 0$$

$$\sqrt{r} \left\{ \frac{\partial u^s}{\partial r} - i k u^s \right\} \rightarrow 0$$

```
< run
scat
solve
V401=sqrt(V101*V101+(V102*V102))
black
contour 401
shade 401
```

```
>
```



## Input

```
Fastflo>
run
Fastflo>
cont 401
Fastflo>
```

## Output

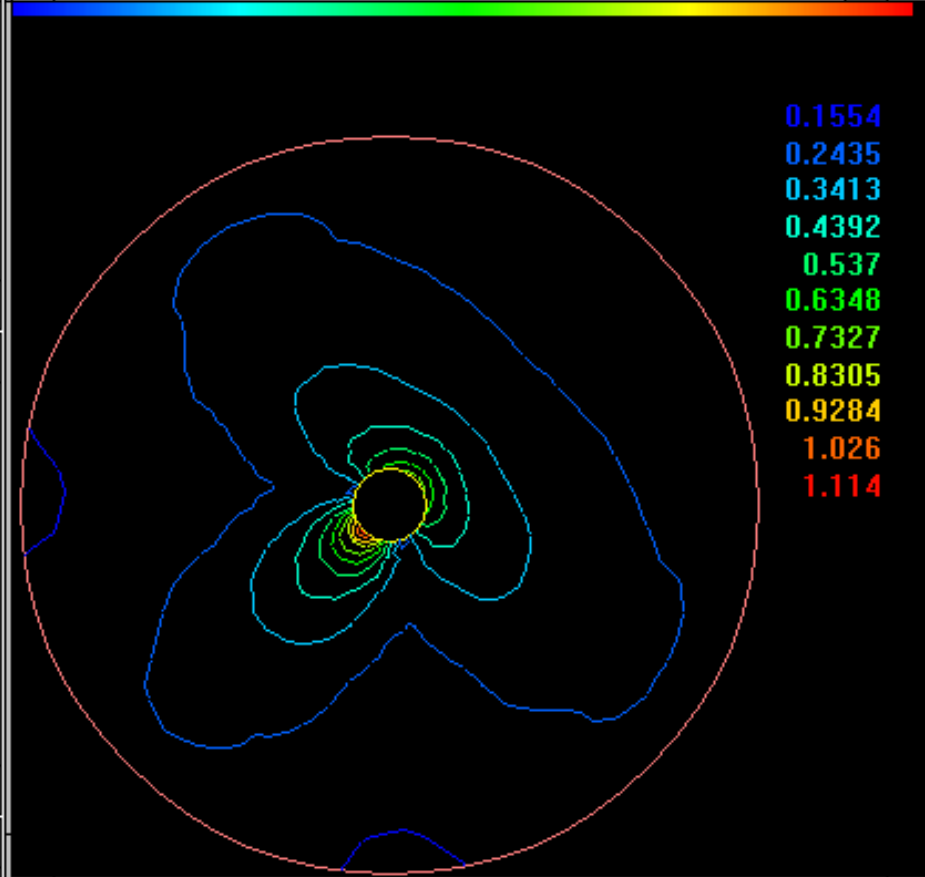
```
elements. Integration uses 7 quadrature
For this problem and memory allocation,
you should not exceed 9142 nodes.

Program now running
prim
black
```

## Message

To select vector: place cursor there and press <e>

## Graphics



# Deformation of a Bimetallic Strip

This example involves:

- time dependent heat diffusion (parabolic PDE)  
solved by implicit timestepping
- two further coupled PDEs for the displacements
- two computational domains, joined along a  
common boundary
- moving meshes and boundaries

# Deformation of a Bimetallic Strip

1000 0 3

0 0 0 0 0

List of vertices

0.0 0.0

0.02 0.0

0.02 0.0002

0.0 0.0002

0.02 0.0004

0.0 0.0004

List of boundary tags

1 2

2 2

3 1

4 3

0 0

3 2

5 2

6 3

4 1

0 0

End of boundary list

These lines demarcate the two regions; region 1 will be Cu, region 2 will be Fe

undeformed bimetallic strip



## Features of this computation

(for which colours indicate temperature and the macro code is shown on the next slide)

- time-dependent thermoelasticity
- in each region, we solve for displacements (two components) and temperature
- multiple regions with continuous heat flux
- moving meshes and boundaries

37.55  
52.19  
70.5  
88.81  
107.1  
125.4  
143.7  
162  
180.3  
198.6  
216.9  
235.2  
253.6  
271.9  
290.2  
308.5  
326.8  
345.1  
363.4  
381.7  
396.3



hot end

cold end

# Bimetallic Strip (cont'd)

```
% bimet.al.prb
P 1 nu 0.3 % parameters for Cu
P 1 E 110000
P 1 Expan 1.65e-5
P 1 mu E_1/(2+2*nu_1)
P 1 lambda mu_1*nu_1*2/(1-nu_1)
P 1 k 393
P 1 rhocp 396*8930
P 2 nu 0.33 % parameters for Fe
P 2 E 209000
P 2 Expan 1.1e-5
P 2 mu E_2/(2+2*nu_2)
P 2 lambda mu_2*nu_2*2/(1-nu_2)
P 2 k 80
P 2 rhocp 450*7860
P dt 0.1 % other parameters
P nstep 5
```

```
A displace
e D_j{mu}D_j U1_i + D_j{mu}D_i U1_j + \
  D_i{lambda}D_jU1_j=D_i{V201*E*Expan/(1-nu)}
b 3 U1={0.,0.}
A tempcalc
e {rhocp}U1-D_j{k*dt}D_jU1={rhocp*V202}
b 3 U1=400.
```

```
< run
  iter=1
  V202=25
  while iter<nstep
    step
    iter=iter+1
  endwhile
>
< step
  heat
  stretch
  V400=V400+V100
  show V400
>
```

```
< heat
  tempcalc
  solve
  V301=V101-V302
  V302=V101
  black
  contour 302
  shade 302
  show V300
  popp
>
```

```
< stretch
  displace
  solve
  V200=V100
  mapm
  V100=V100+V200
  mapm
  popp
  show V100
>
```

# Some examples for investigation

- 3 algorithms for heat diffusion

2Ddiff.msh

2Ddiff1.prb, 2Ddiff2.prb, 2Ddiff3.prb

crucible.msh, crucible.prb

- more on thermoelasticity

elas\_th2.msh, elas\_th2.prb

- Black-Scholes equation

blasch1.msh, blasch1.prb

# Black-Scholes equation for options pricing

This example involves:

- a famous equation in finance
- time regarded as a space-like variable
- simple mesh generation and coding

## Black-Scholes equation (continued)

$$-\frac{\partial V}{\partial t} + \frac{\partial}{\partial S} \left( \frac{1}{2} \sigma^2 S^2 \frac{\partial V}{\partial S} \right) - (\sigma^2 - r) S \frac{\partial V}{\partial S} - rV = 0$$

$V(S,t)$  is the value of an option to be exercised at  $t^*=T$

$t = T - t^*$

$S$  is value of underlying portfolio (over which option is held)

$\sigma$  is the volatility

$r$  is the interest rate

# Black-Scholes equation (continued)

600 0 4

1 0 0 0 0

0.4 0.

List of vertices

0.0 0.0

1.0 0.0

1.0 1.0

0.0 1.0

List of boundary tags

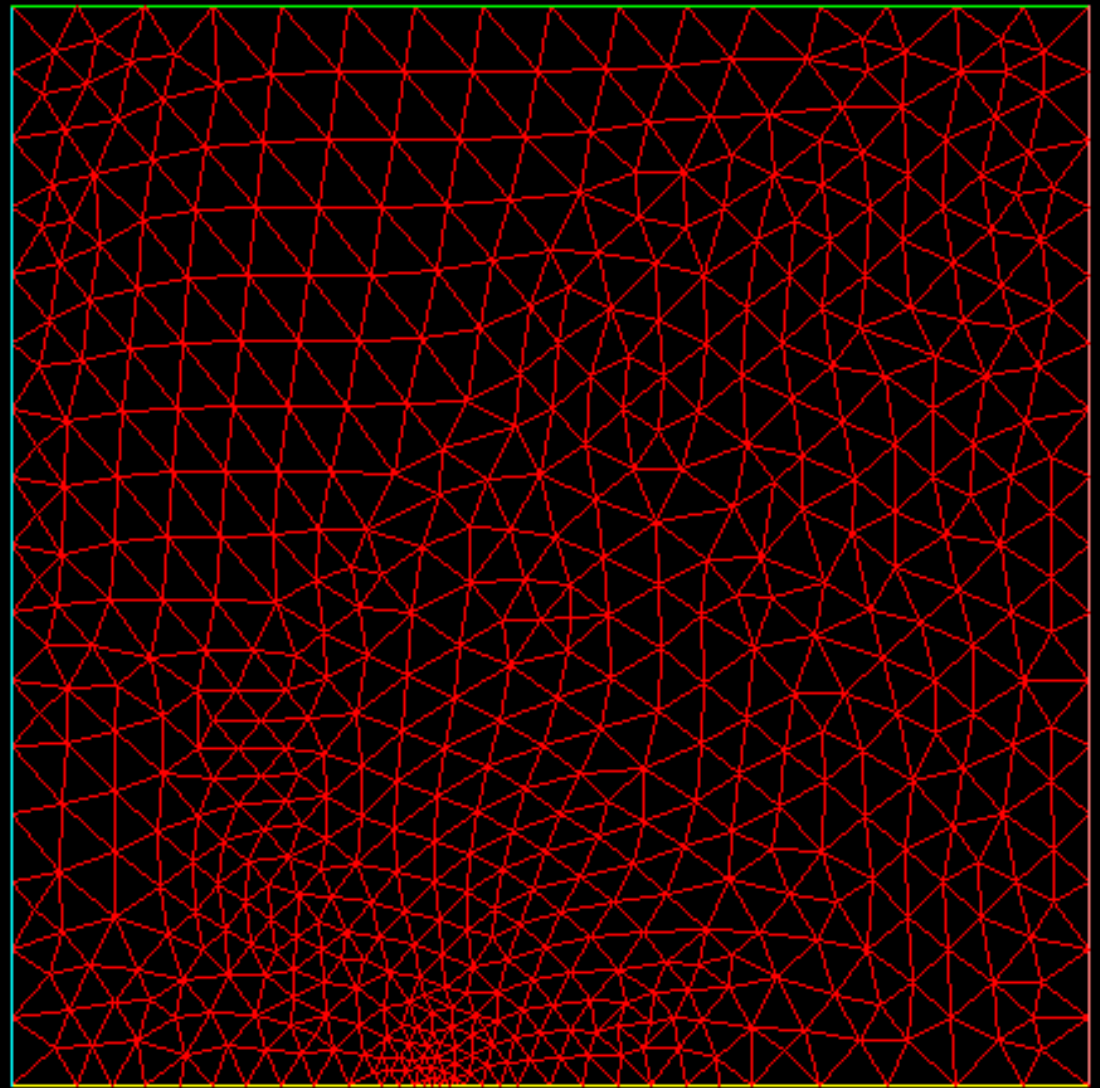
1 1

2 2

3 3

4 4

End of boundary list



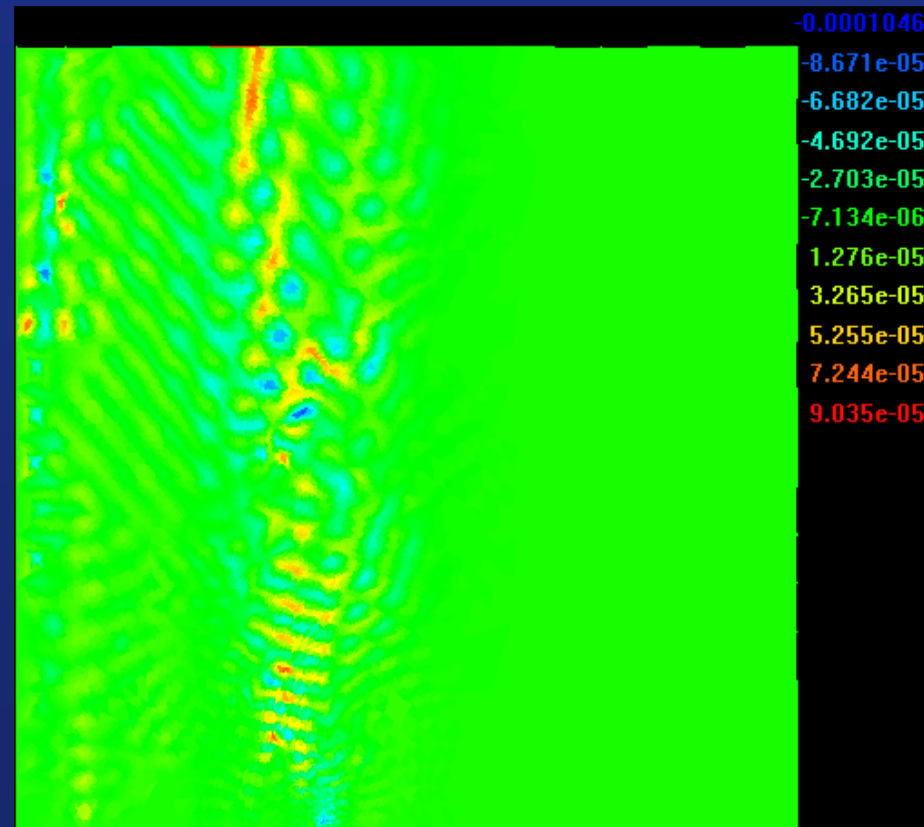
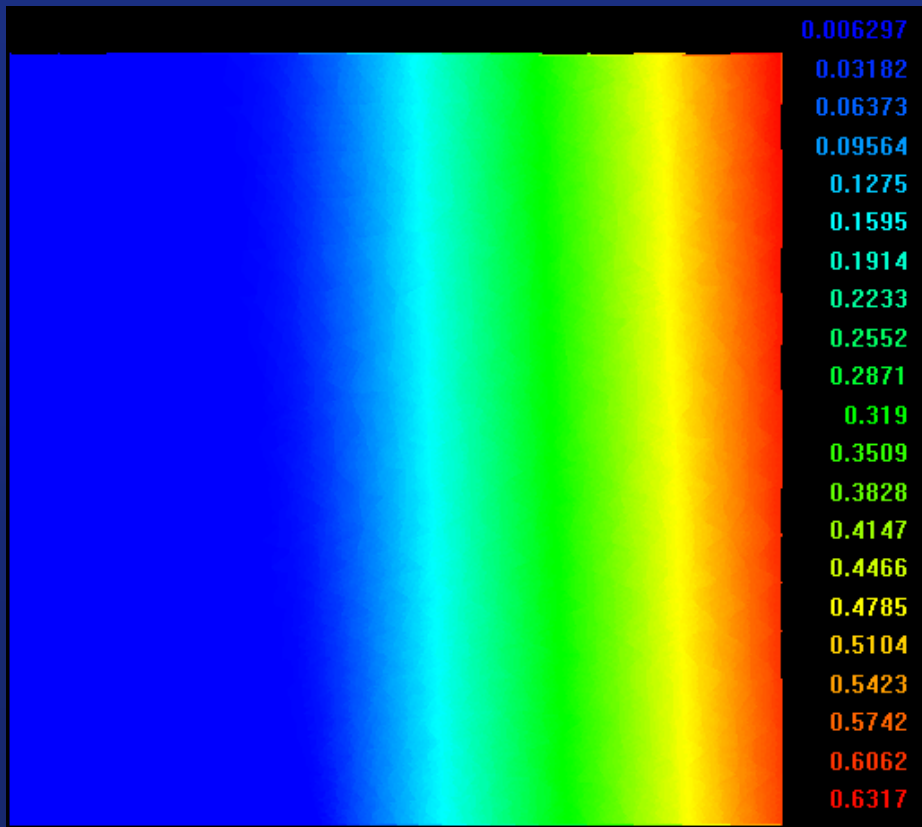
# Black-Scholes equation (continued)

```
% blasch1.prb
P sigma 0.2
P s2haf sigma^2/2.
P r 0.1
P strike 0.4
A blsch
e D_j{s2haf*X1*X1,0.0,0.0,0.0}_jk D_k U1 \
    -{s2m1*X1,1.0}_jD_jU1-{r}U1={0}
b 2 U1={1-strike*exp(-r*X2)}
b 4 U1={0.0}
b 1 U1={cut(X1-strike)}
```

```
< run
    s2m1=s2haf*2-r
    blsch
    solve
    contour
    approx
    V301=V201-V101
    show V301
```

```
>
```

# Black-Scholes equation (continued)

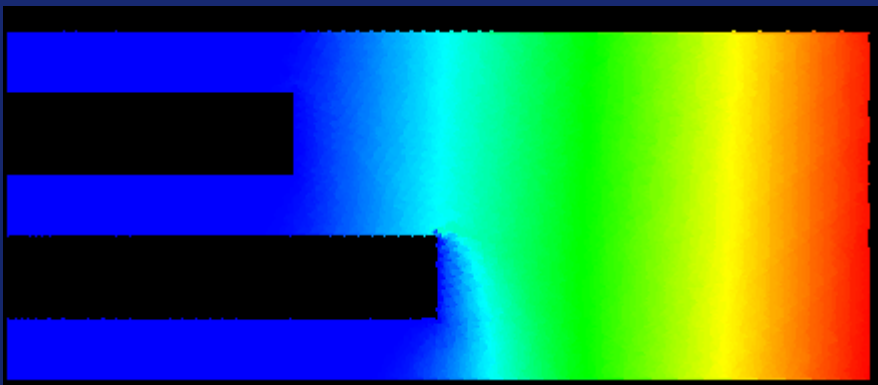
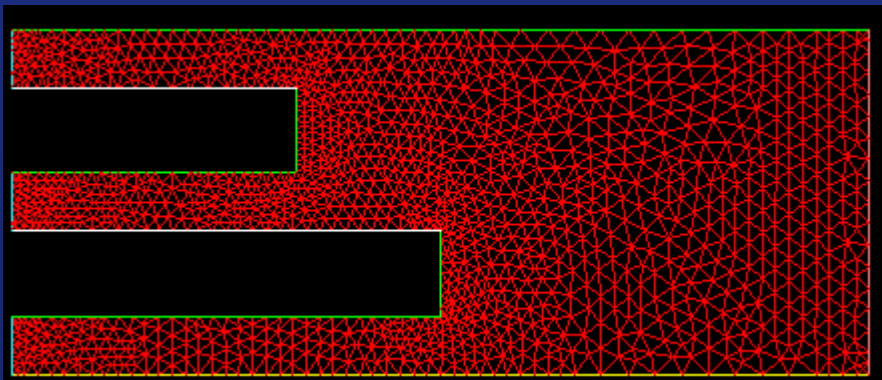


Solution for  $V(\sigma, t)$

Error in the solution

# Black-Scholes equation (continued)

Why is a finite element method useful? ... to handle cases with “barriers”, which are equivalent to geometrical complexity.



Such cases are commonly encountered in “exotic” options

## Where to find out more

- [www.cmis.csiro.au/Fastflo](http://www.cmis.csiro.au/Fastflo)
- [www.compumod.com.au](http://www.compumod.com.au)
- [www.nag.co.uk](http://www.nag.co.uk)
- *Fastflo* Tutorial Guide, Version 3
- *Fastflo* Fluids ToolBox

# Summary

- *Fastflo* - features and strengths
- overview of *Fastflo* Version 3
- how to formulate/solve PDEs
- examples – scattering, heat diffusion,  
Black-Scholes equation
- where to find further information