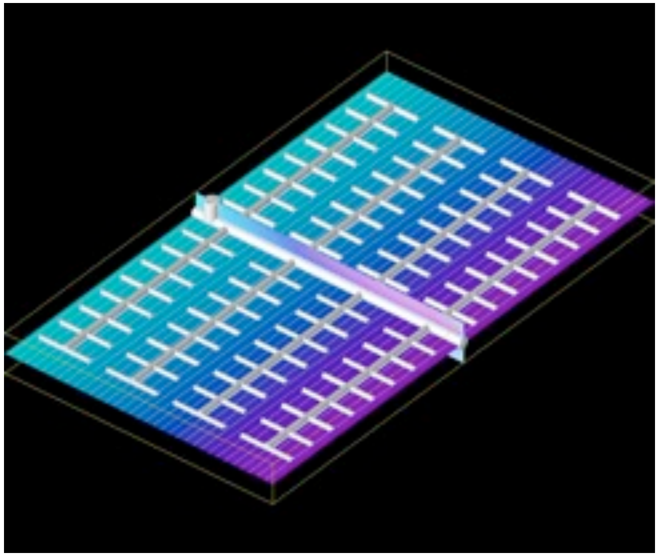
# Introduction to freefem++

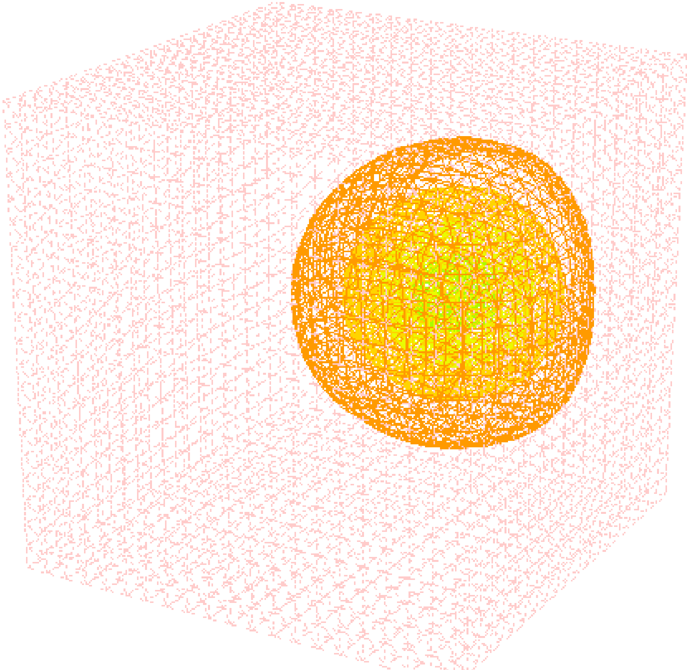
Olivier Pironneau

LJLL-UPMC

# History

- 1985: MacFEM – PCFEM
- 1990: syntax analyzer (+ D. Bernardi) freefem
- 1995: freefem+ (+ Hecht)
- 2000: freefem++ (Hecht alone)
- 2000: freefem3D (DelPino, Havé , Pironneau)
- 2003: an integrated environment + web (Lerouzic)
- 2005: a new documentation (+ Ohtsuka)
- 2009: freefem++3D
- A web site [www.freefem.org](http://www.freefem.org)
- do not mix **freefem++3xx** and ff3D:
  - Fictitious domain & mesh gen by marching cube
  - Parallel iterative solver with  multigrid





# Leading ideas

- Follow the math => variational formulation
- Algorithms are the user's responsability
- Blocks: An elliptic + upwinding operator
- Use Finite Element Methods
- Automatic mesh generation with adaptivity
- Follow the research front (if it is FEM it can be done with freefem++)

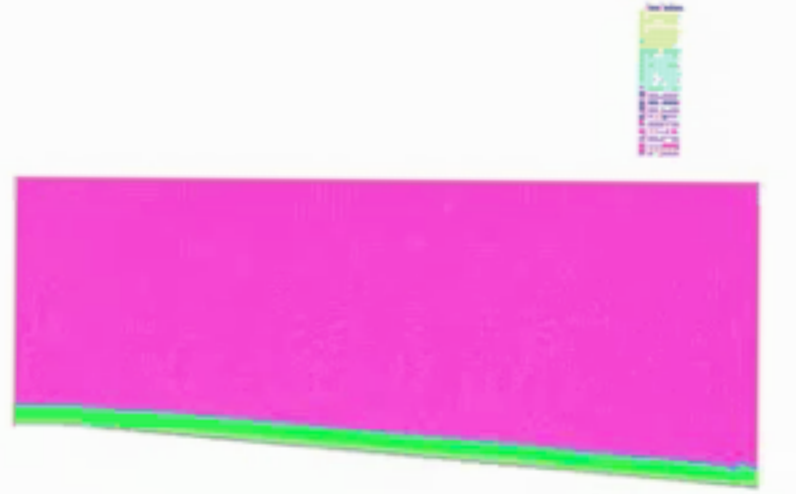
$$\partial_t u + a \cdot \nabla u - \nu \Delta u = f, \quad u|_{\partial\Omega} = 0$$

$$\frac{u^{m+1}(x) - u^m(x - a^m(x)\delta t)}{\delta t} - \nu \Delta u(x) = f^m(x)$$

$$\int_\Omega (uw + \delta t \nu \nabla u \nabla w) = \int_\Omega (u^m o X w + \delta t f w) \;\; \forall w \in H_0^1(\Omega)$$

```
solve A(u,w) =  int2d(th)(u*w+ nu*dt*grad(u)*grad(w))
               - int2d(th)(convect(v,[a1,a2],-dt)+ f*w*dt )
               + on(bdy, u=0);
```

# Leading ideas



- Follow the math => variational formulation
- Algorithms are the user's responsability
- Blocks: An elliptic + upwinding operator
- Use Finite Element Methods
- Automatic mesh generation with adaptivity
- Follow the research front (if it is FEM it can be done with freefem++)

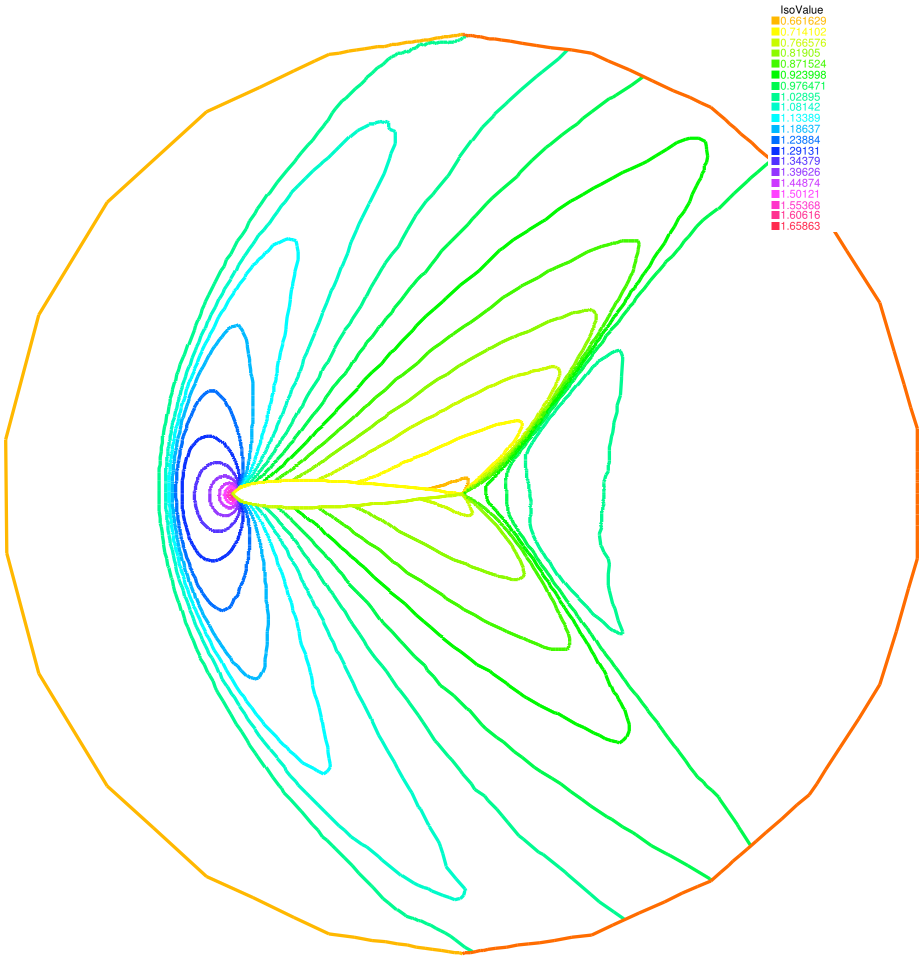
$$\partial_t u + a \cdot \nabla u - \nu \Delta u = f, \quad u|_{\partial\Omega} = 0$$

$$\frac{u^{m+1}(x) - u^m(x - a^m(x)\delta t)}{\delta t} - \nu \Delta u(x) = f^m(x)$$

$$\int_\Omega (uw + \delta t \nu \nabla u \nabla w) = \int_\Omega (u^m \text{o} X w + \delta t f w) \ \ \forall w \in H_0^1(\Omega)$$

```
solve A(u,w) =  int2d(th)(u*w+ nu*dt*grad(u)*grad(w))
              - int2d(th)(convect(v,[a1,a2],-dt)+ f*w*dt )
              + on(bdy, u=0);
```

# freefem++ documentation

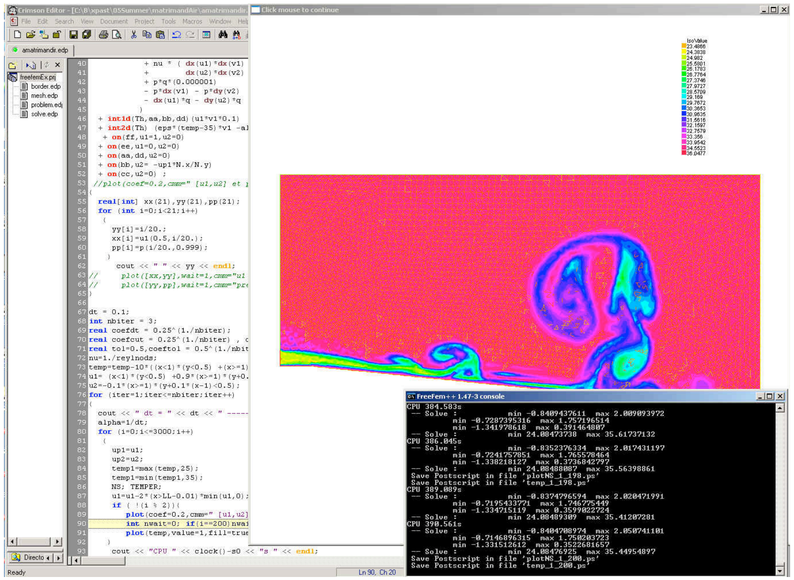



Freefem++

Third Edition, Version 3.4-2

http://www.freefem.org/ff++

F. Hecht

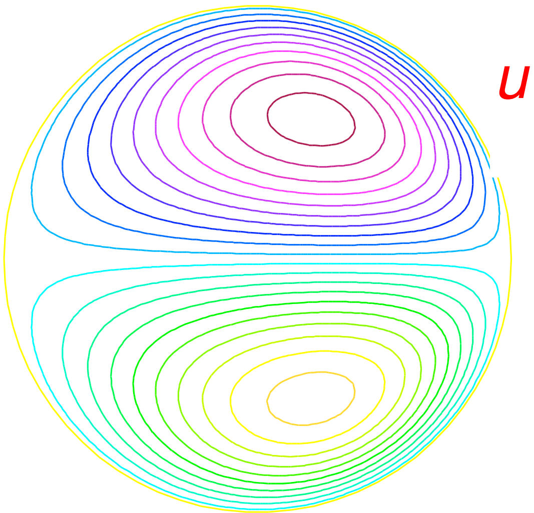


Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie, Paris

# A Dirichlet Problem

$$-\Delta u = f, \quad u|_{\partial\Omega} = 0$$

Variational formulation



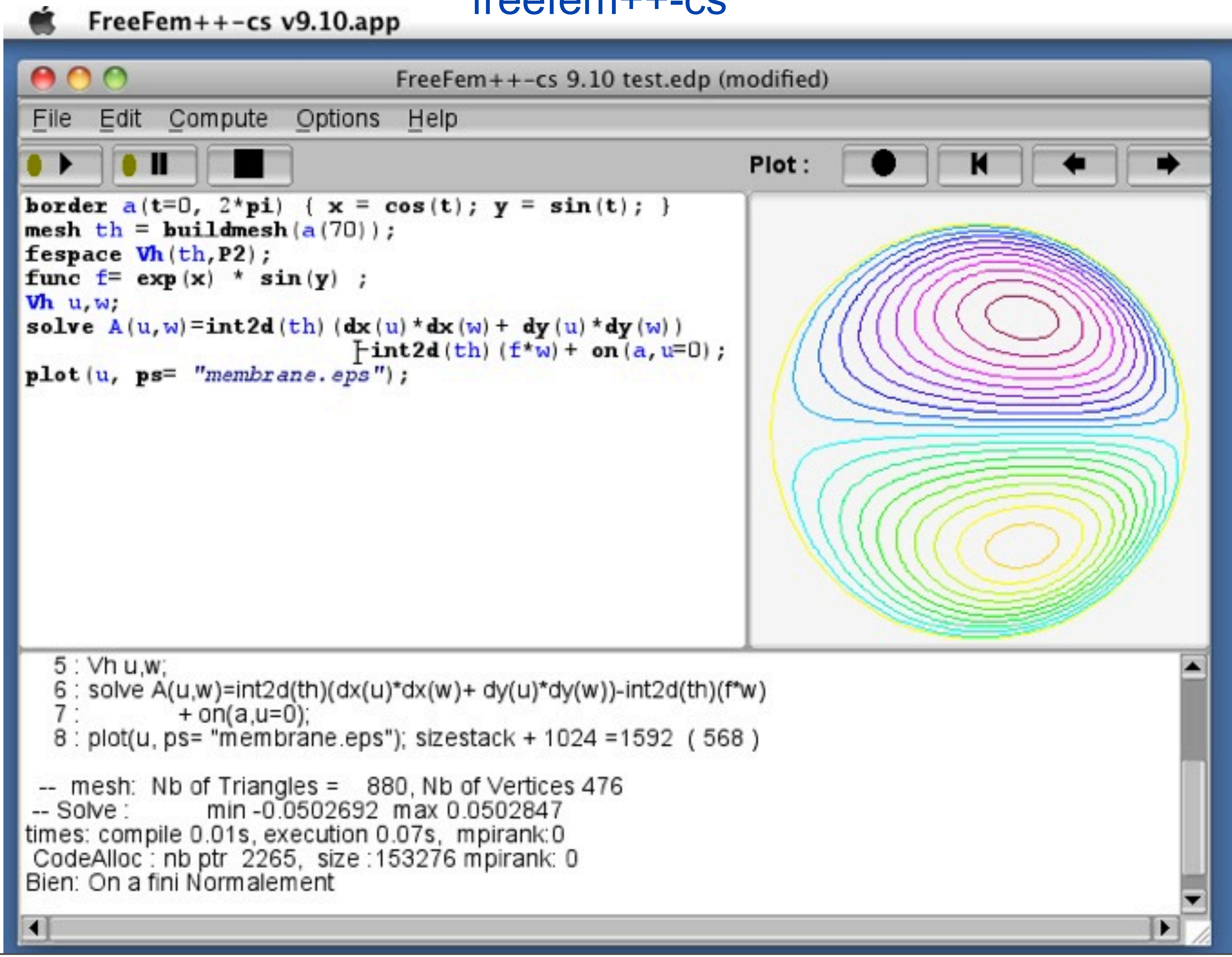
$$u \in H^1_0(\Omega) \; ? \; : \quad \int_\Omega \nabla u \nabla w) = \int_\Omega f w \;\; \forall w \in H^1_0(\Omega)$$

Approximation

$$\int_\Omega \nabla u_h \nabla w_h) = \int_\Omega f w_h \;\; \forall w \in V_0$$

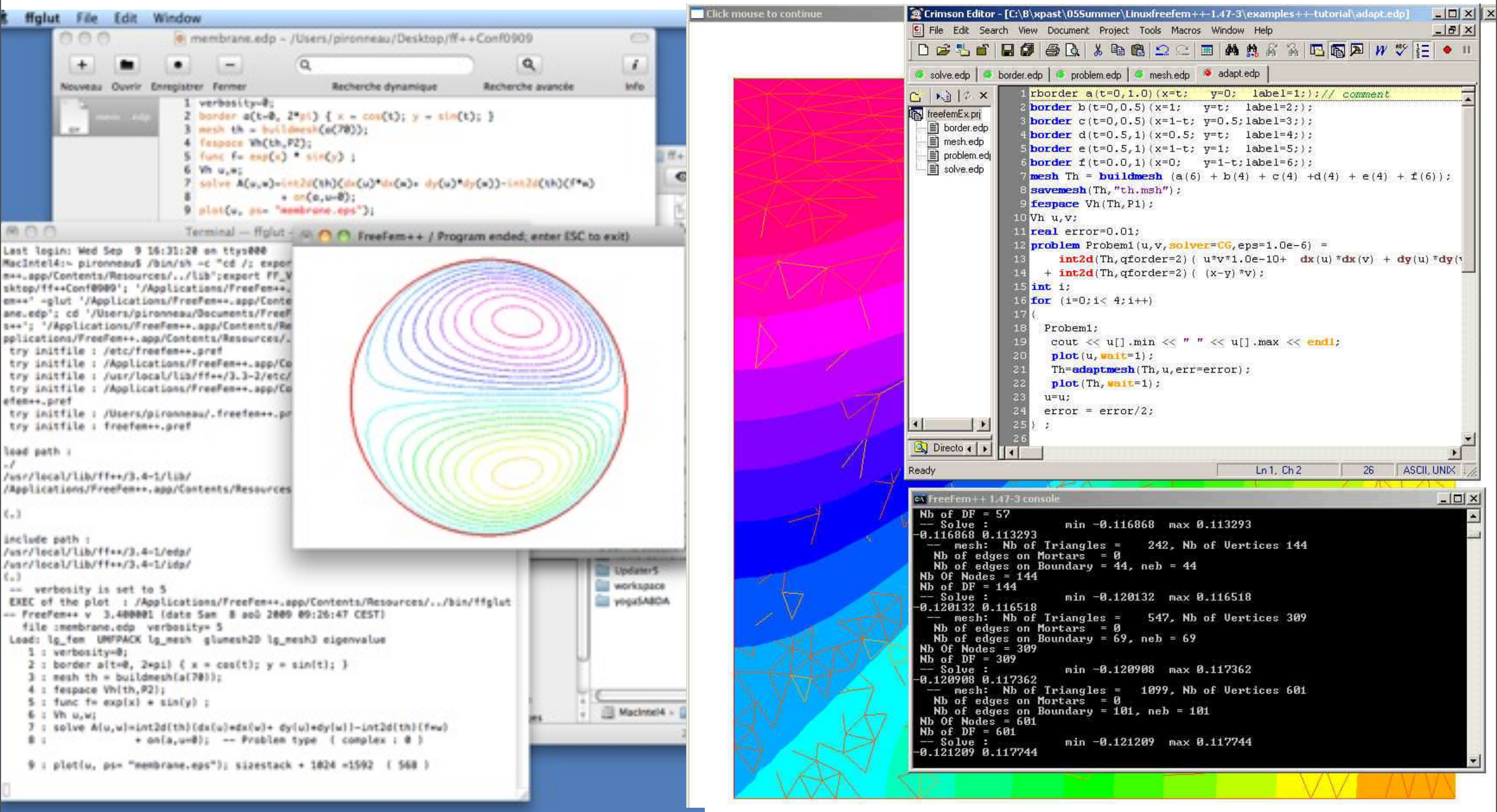
```
border a(t=0, 2*pi) { x = cos(t); y = sin(t); }
mesh th = buildmesh(a(70));
fespace Vh(th,P2);
func f= exp(x) * sin(y) ;
Vh u,w;
solve A(u,w)=int2d(th)(dx(u)*dx(w)+ dy(u)*dy(w))-int2d(th)(f*w)
              + on(a,u=0);
plot(u, ps= “membrane.eps“);
```

# freefem++-cs

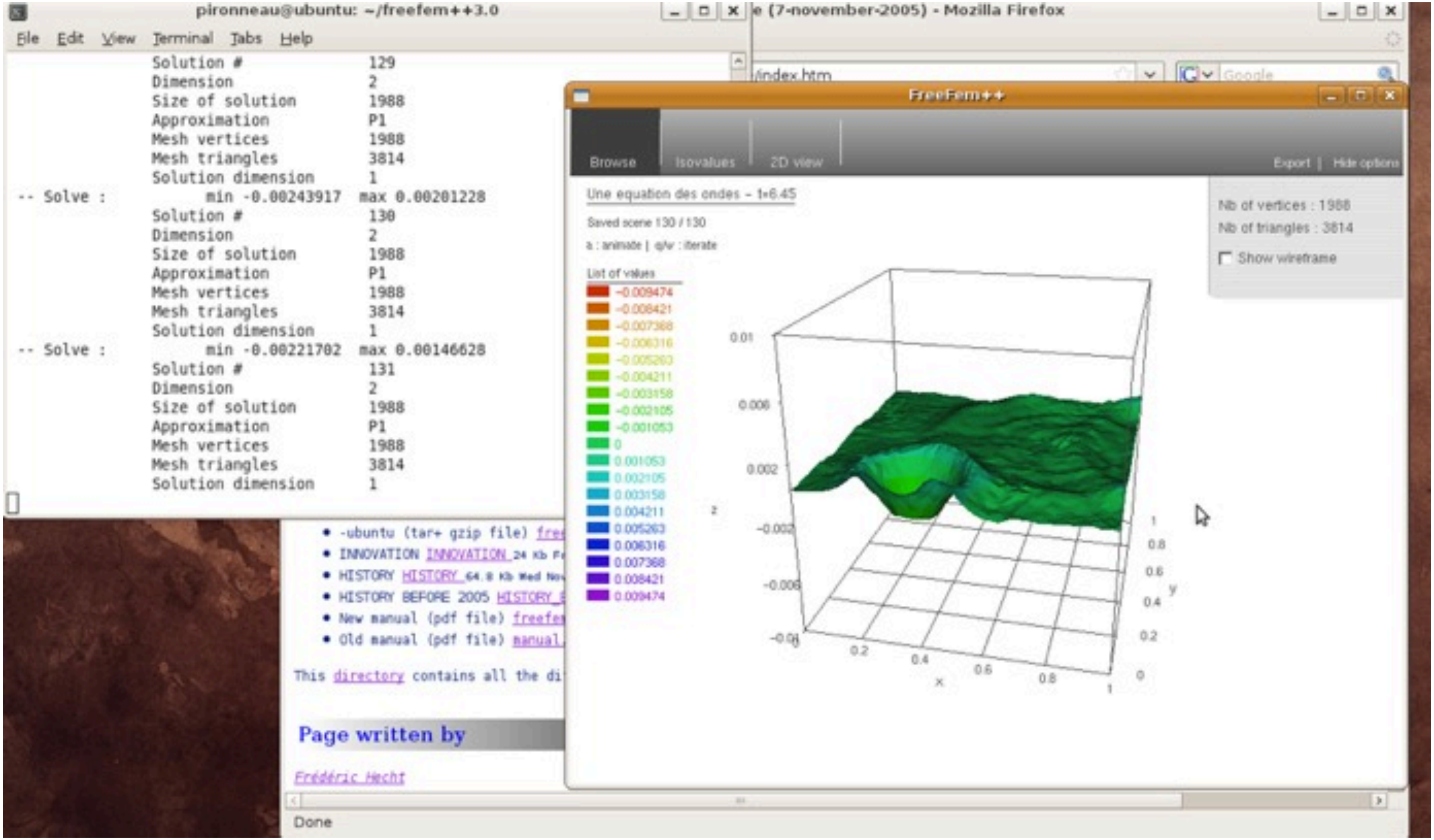

FreeFem++-cs v9.10.app
FreeFem++-cs 9.10 test.edp (modified)
File Edit Compute Options Help
Plot :
border a(t=0, 2*pi) { x = cos(t); y = sin(t); }
mesh th = buildmesh(a(70));
fespace Vh(th,P2);
func f= exp(x) * sin(y) ;
Vh u,w;
solve A(u,w)=int2d(th)(dx(u)*dx(w)+ dy(u)*dy(w))
-int2d(th)(f*w)+ on(a,u=0);
plot(u, ps= "membrane.eps");
5 : Vh u,w;
6 : solve A(u,w)=int2d(th)(dx(u)*dx(w)+ dy(u)*dy(w))-int2d(th)(f*w)
7 : + on(a,u=0);
8 : plot(u, ps= "membrane.eps"); sizestack + 1024 =1592 ( 568 )
-- mesh: Nb of Triangles = 880, Nb of Vertices 476
-- Solve : min -0.0502692 max 0.0502847
times: compile 0.01s, execution 0.07s, mpirank:0
CodeAlloc : nb ptr 2265, size :153276 mpirank: 0
Bien: On a fini Normalement

# Integrated Development Environment

- Edit/compile/debug = freefem++-CS (by A. Leyaric)
- Your favorite editor + terminal window
- Smultron (Mac) Crimpson(Windows) + scripts

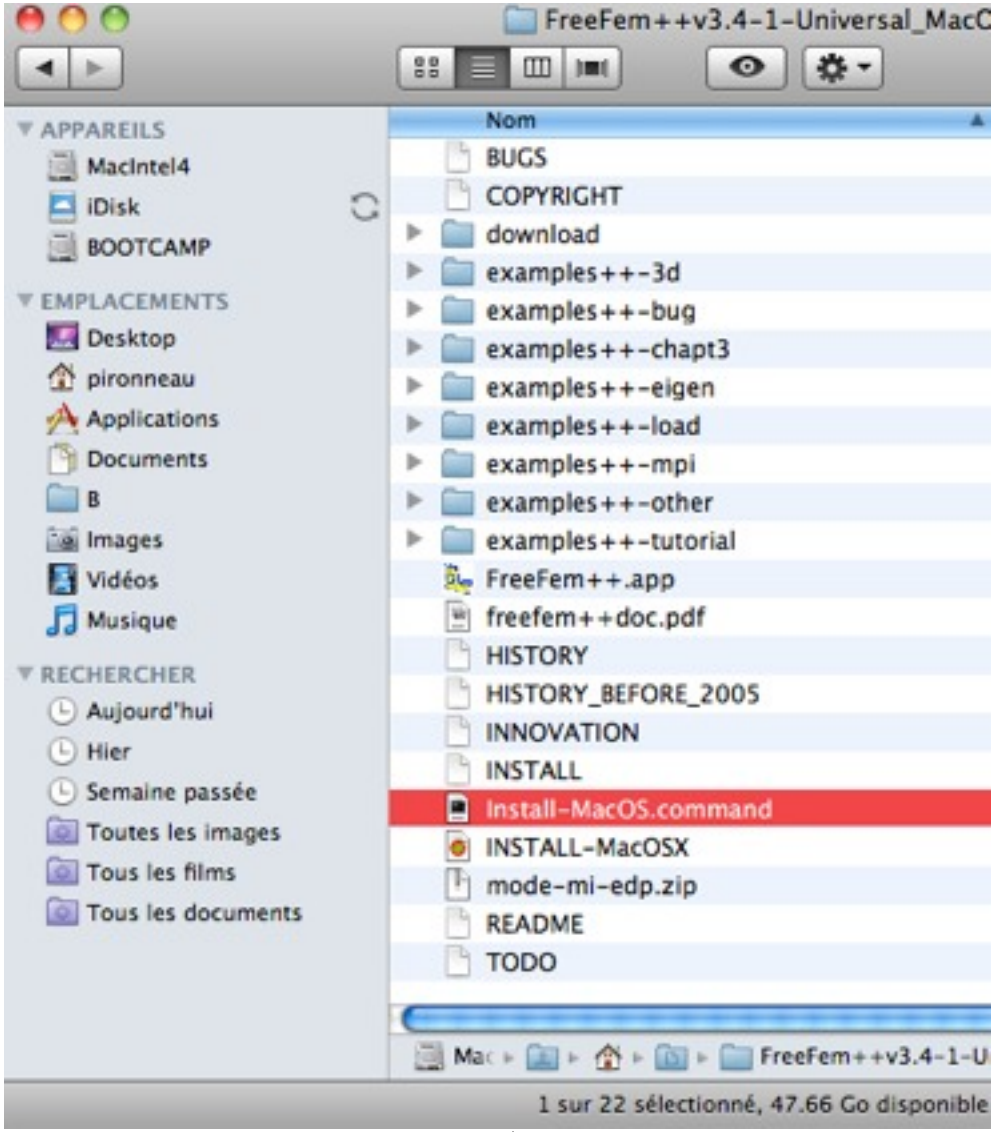


# ubuntu-64




with openGL graphics by L. Dumont
(to know which ubunto you have do: )

# Install of freefem++




Mac OSX: Download + expand+
Download freefem++-cs

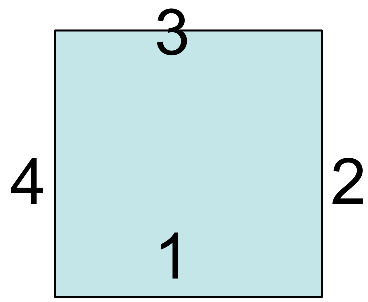
Windows:
Download the archive+exe
Download freefem++-cs

Linux: download+unzip+compile
ubuntu-32: download the
[freefem++-v3.5-ubuntu.tar.gz]
[freefem++-v3.2-usr-lib.tar.gz]

```
sudo tar zxvf freefem++-v3.5-ubuntu.tar.gz -C /
sudo tar zxvf freefem++-v3.2-usr-lib.tar.gz -C / -k
```

# 2D Mesh Generation

```
mesh th = square(5,5); //unit square: bdy 1 is (0,1)x(0)
// bdy 2 is (1)x(0,1)... bdy 4 is (0)x(1,0)
mesh Th = square(5,10,[x-0.5, 10*y]);//(-0.5,0.5)x(0,10)
border a(t=0,2*pi){ x = cos(t); y = sin(t);label=2;}
border b(t=0,2*pi){ x =0.5+0.3*cos(-t); y =0.2*sin(-t);}
mesh th1 = buildmesh( a(20) + b(10));
mesh th2 = movemesh(th1,[x+1,y+2]);
mesh th3 = readmesh("mymesh.msh");
func f = sin(x+1);
```




**Rule 1**: The domain is on the left of its oriented boundary

**Rule 2**: Borders are defined piecewise analytically but must make continuous and closed curves.

**Rule 3**: borders may not overlap nor cross each other.

**Rule 4**: Each border is assigned a number but can be refered by names also. Unless overwritten the number is the order of appearance of the key word «border».

# Finite Element Spaces

P0, P1, P2, P3, P1nc, P1dc,P2dc, P1b,RT0
P03d, P13d, P23d, RT03d, Edge03d, P1b3d

```
fespace Vh(th,P1dc);  Vh v,vh;
varf  A(v,vh) = int2d(th)(v*vh/dt/2);
varf  B(vh,w) =intalledges(th)(vh*mean(w)*(N.x*u1+N.y*u2))
                   -int2d(th)( w*(u1*dx(vh)+u2*dy(vh)));
```

[N.x,N.y]=vecteur normal

Mean(w)=(v+ + v-)/2

$$\partial_t u + a\nabla u = 0$$

$$\frac{1}{2\delta t}\int_\Omega (u^{m+1} - u^{m-1})w$$
$$= \int_\Omega u^m(a\nabla w) - \sum \int_{\partial T} \bar{u}^m(a\cdot n)w$$

# Finite Element Spaces

P0, P1, P2, P3, P1nc, P1dc,P2dc, P1b,RT0
P03d, P13d, P23d, RT03d, Edge03d, P1b3d

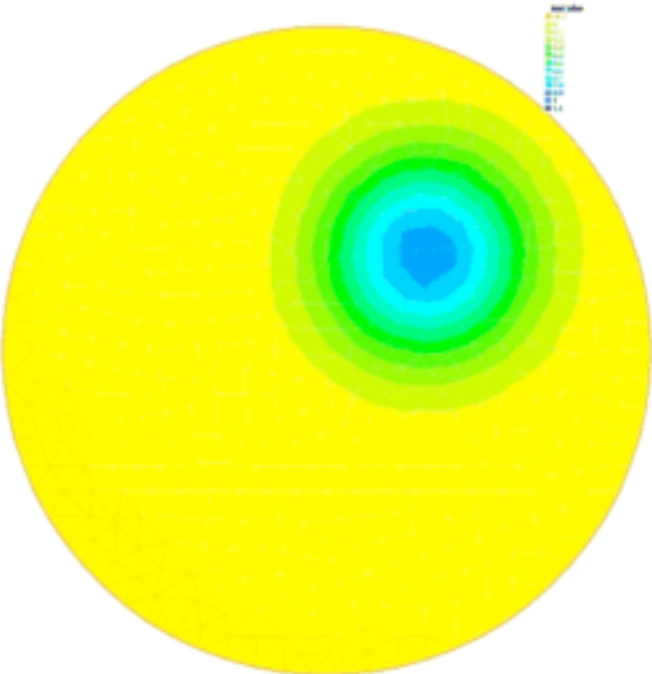
```
fespace Vh(th,P1dc);  Vh v,vh;
varf  A(v,vh) = int2d(th)(v*vh/dt/2);
varf  B(vh,w) =intalledges(th)(vh*mean(w)*(N.x*u1+N.y*u2))
                    -int2d(th)( w*(u1*dx(vh)+u2*dy(vh)));
```

[N.x,N.y]=vecteur normal

Mean(w)=(v+ + v-)/2

$$\partial_t u + a\nabla u = 0$$

$$\frac{1}{2\delta t}\int_\Omega (u^{m+1} - u^{m-1})w$$
$$= \int_\Omega u^m(a\nabla w) - \sum \int_{\partial T} \bar{u}^m(a\cdot n)w$$



# Boundary Conditions

- Dirichlet cond by using `on(thebdylabel,u=z)`
- Neumann cond are in the variational formulation: `int1d(th,2)(nu*g*w)`

$$u - \nu \Delta u = 0, \quad \frac{\partial u}{\partial n} = g \text{ on } \Gamma_2, \quad u|_{\Gamma_1} = z$$

$$\int_\Omega (uw + \nu \nabla u \nabla w) = \int_{\Gamma_2} \nu g w \quad \forall w|_{\Gamma_1} = 0$$

Periodic conditions are within the space definition

```
mesh Th=square(10,15);
fespace Vh(Th,P1,periodic= [2,y],[4,y]);
```

Conditions on RT0 elements can be tricky to formulate:

# Operators

- `fespace Vh(th,P2);`
- **`Vh`** `u;`
- `dx(u), dy(u), dxx(u), dyy(u), dxy(u)`
- `convect(u,[a_1,a_2],dt), mean(u), jump(u)`

- You can make your own

- `macro div(u,v) ( dx(u)+dy(v) )  //`
- 

- `sin(u), exp(u), ...`
- `int2d(u), u[].max, ...`

- Rule: these are evaluated pointwise when needed . `Example:`
- `real I = intalledge(th)(sin(dx(u))^2);`
- `is computed as the sum of the values of the integrand at the quadrature points of the edges in a loop over all triangles.`

# Quadrature formulae and Solvers

```
mesh th = square(15,15);
fespace Vh(th,P1);
Vh u, w, g = x+y;
solve a(u,w,solver=UMFPACK)
    = int2d(th)(grad(u)'*grad(w))
       +  int1d(th,qfe=qf1pE) (1.e10* u*w)
         -int1d (th, qfe=qf1pE)(1.e10*g*w) ;
```

Because the quadrature points are the vertices, it is the same as

```
solve a(u,w) = int2d(th)(grad(u)*grad(w))
        - int2d(f,w) + on(th,1,2,3,4,u=0);
```

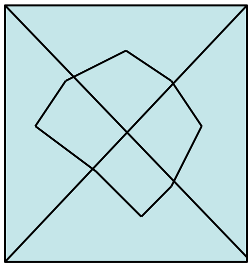
```
solver
= LU, CG, Crout, Cholesky, GMRES, sparsesolver, UMFPACK
```

# Syntax: an incomplete extension of C++

```
mesh Th = square(5,5);
fespace Vh(Th,P1);  Vh u=0;
Vh<complex> uc = x+1.i*y ; //complex FE function or array

int i = 0 ;
real a=2/5 ; // quiz? value of a?
bool b=(a<2) ;
real[int] aa(10) ; // a real array of 10 value
cout<<u(.5,0.6)<<endl ; //value of FE function u at (.5,.6)
if(u<1.0) a=2; else a=1;  // wrong
Vh au = (u<1.0) ? 2.0 : 1.0;
ofstream ff("myfile.txt");
for(i=0;i<Th.nv;i++) // also while, break, continue
for(int j=0;j<3;j++)
        cout<<Th[i][j].x<<"\t"<<Th[i][j].y<<"\t"<<u[]
[Vh(i,j)]<<endl;

for (int i=0 ;i<u[].n ;++i) { u[][i]=1 ;
       plot(u,wait=1,dim=3,fill=1,cmm=" v"+i) ; u[][i]=0;}
```

# Finite Volumes / Finite Elements

- A volume $\sigma$ is associated to each vertex



$$\partial_t v + \nabla . F(v) = 0 \Rightarrow \int_{\sigma} \partial_t v + \int_{\partial\sigma} F(v).n = 0$$

triangle/triangle assembly is possible

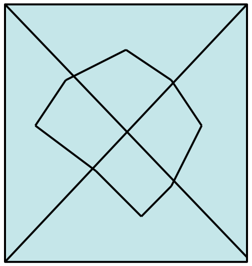
-The first intégral is 1/3 of the same on triangles

-One needs to write a load module for the boundary terms

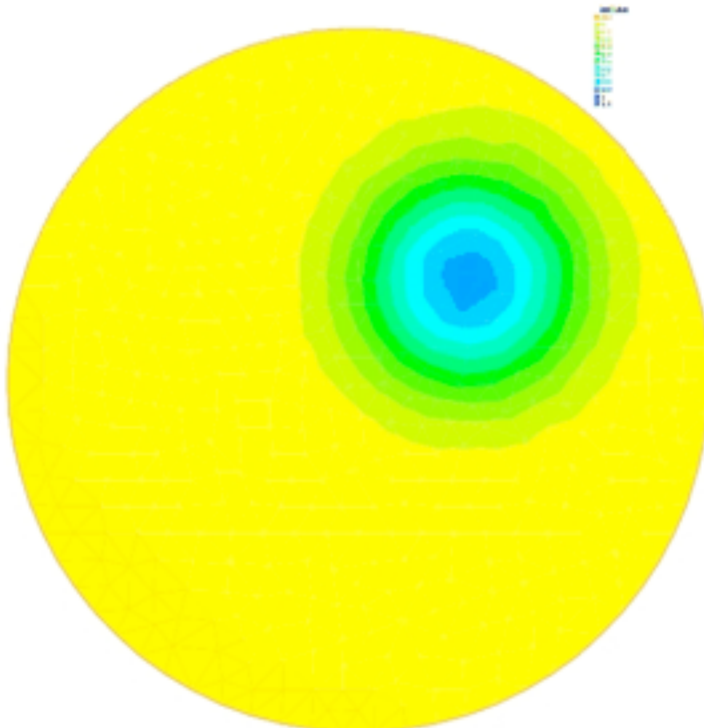
Max=0.43 (very diffusive

# Finite Volumes / Finite Elements

- A volume $\sigma$ is associated to each vertex



$$\partial_t v + \nabla . F(v) = 0 \Rightarrow \int_\sigma \partial_t v + \int_{\partial\sigma} F(v).n = 0$$



triangle/triangle assembly is possible

-The first intégral is 1/3 of the same on triangles

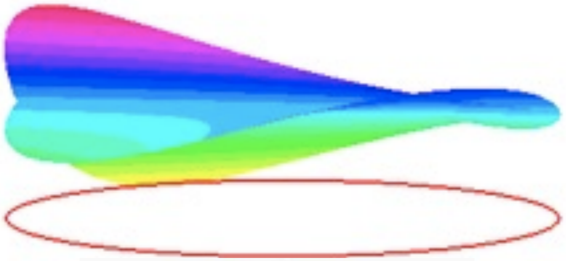
-One needs to write a load module for the boundary terms

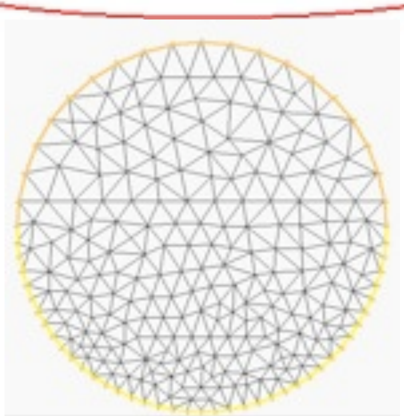
Max=0.43 (very diffusive

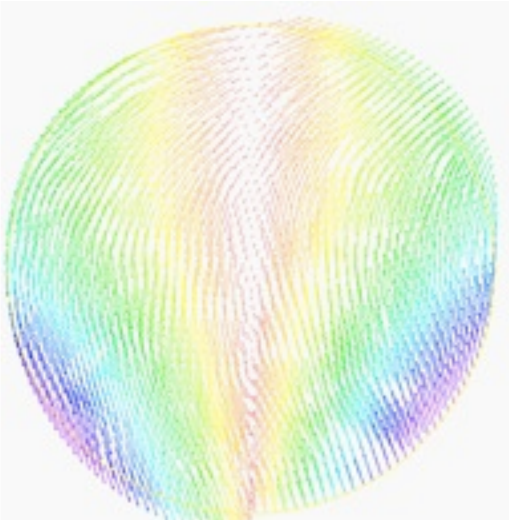
# Plots

```
• border a(t=0,pi){ x=cos(t); y=sin(t);}
• border b(t=pi,2*pi){ x=cos(t); y=sin(t);}
• plot(a(20)+b(40),wait=true);
• mesh th=buildmesh(a(20)+b(40));
• plot(th, wait=1,ps="th.eps");
• fespace Vh(th,P2);    Vh u=sin(x*y), v=x*exp(-y);
• plot(u,v,wait=1, value=1,fill=1,dim=3);
```





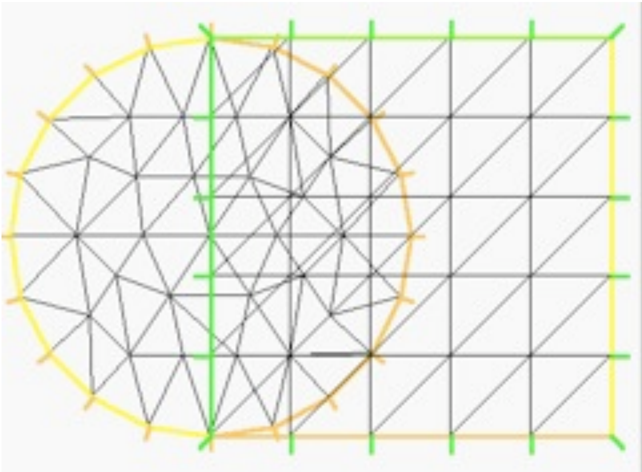


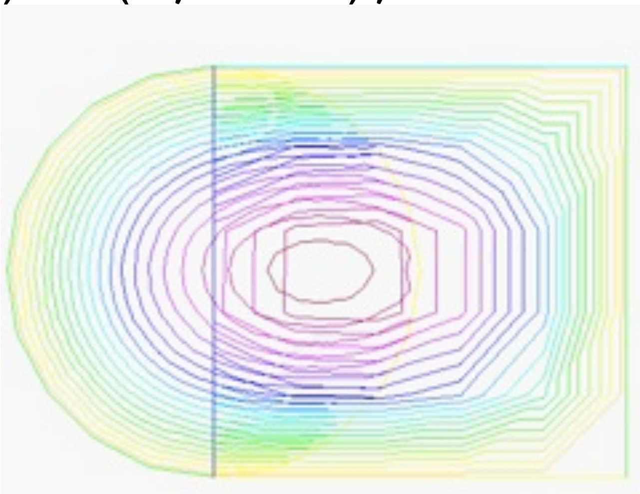


More: cut, link with gnuplot and medit

# Interpolation

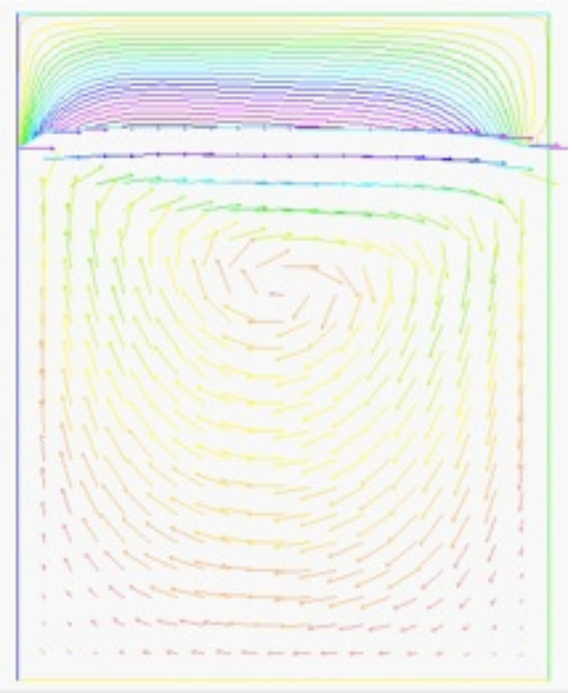
```
border a(t=-pi/2,pi/2){ x=cos(t); y=sin(t);}
border b(t=pi/2,3*pi/2){ x=cos(t); y=sin(t);}
mesh th1=buildmesh(a(10)+b(10));
mesh th2=square(5,5,[2*x,2*y-1]);
plot(th1,th2, wait=1);
fespace Vh1(th1,P2);    Vh1 w1,u1=0;
fespace Vh2(th2,P1);    Vh2 w2,u2=0;

macro Grad(u) [dx(u),dy(u)]  //
func f=1;
int i; // to avoid refactorization LU
problem L1(u1,w1,solver=LU, init=i)
      = int2d(th1)(Grad(u1)'*Grad(w1))
         -int2d(th1)(f*w1) + on(b,u1=0)+on(a,u1=u2);
problem L2(u2,w2,solver=LU, init=i)
      = int2d(th2)(Grad(u2)'*Grad(w2))
         -int2d(th2)(f*w2)
+ on(4,u2=u1) +on(1,2,3,u2=0);

for(i=0;i<5;i++){
     L2; L1; plot(u1,u2,wait=1);}
```





**Rule**: pointwise evaluation when needed

# Multi-Physics



```
    mesh th=square(20,10,[x,y/4+1]);
fespace Vh(th,P2); Vh u,v,uu,vv;

mesh Th=square(20,20);
fespace Uh(Th,P1b); Uh uf,vf,uuf,vvf;
fespace Ph(Th,P1); Ph p,pp;

solve stokes([uf,vf,p],[uuf,vvf,pp]) =
   int2d(Th)(dx(uf)*dx(uuf)+dy(uf)*dy(uuf)
         + dx(vf)*dx(vvf)+ dy(vf)*dy(vvf)
         + dx(p)*uuf + dy(p)*vvf + pp*(dx(uf)+dy(vf)))
         + on(1,2,4,uf=0,vf=0) + on(3,uf=1,vf=0);

real s2=sqrt(2.0);
macro epsilon(u1,u2) [dx(u1),dy(u2),(dy(u1)+dx(u2))/s2] // EOM
macro div(u,v) ( dx(u)+dy(v) ) // EOM
real E=21e5, nu=0.28, mu=E/(2*(1+nu)), lambda=E*nu/((1+nu)*(1-2*nu)), f=-1;

solve lame([u,v],[uu,vv])= int2d(th)( lambda*div(u,v)*div(uu,vv)
                  +2*mu*( epsilon(u,v)'*epsilon(uu,vv) ) ) - int2d(th)(f*vv)
                  +int1d(th,1)(50*p*vv) + on(2,3,4,u=0,v=0);

th = movemesh(th,[x,y+400*v]);
Th = movemesh(Th, [x, y+400*y*v(x,1.0)]);

 u=u; v=v; uf=uf;vf=vf;
plot(v,[uf,vf],wait=0);
```

# Non-linear problem

$$-\nabla\cdot((1+|u|^p)\nabla u) = f, \quad u|_{\partial\Omega} = 0$$

Try the fixed point scheme:

$$u^{m+1} \in H^1_0(\Omega) \ : \ \int_\Omega ((1+|u^m|^p)\nabla u^{m+1}\cdot\nabla w) = \int_\Omega fw, \quad \forall w \in H^1_0(\Omega)$$

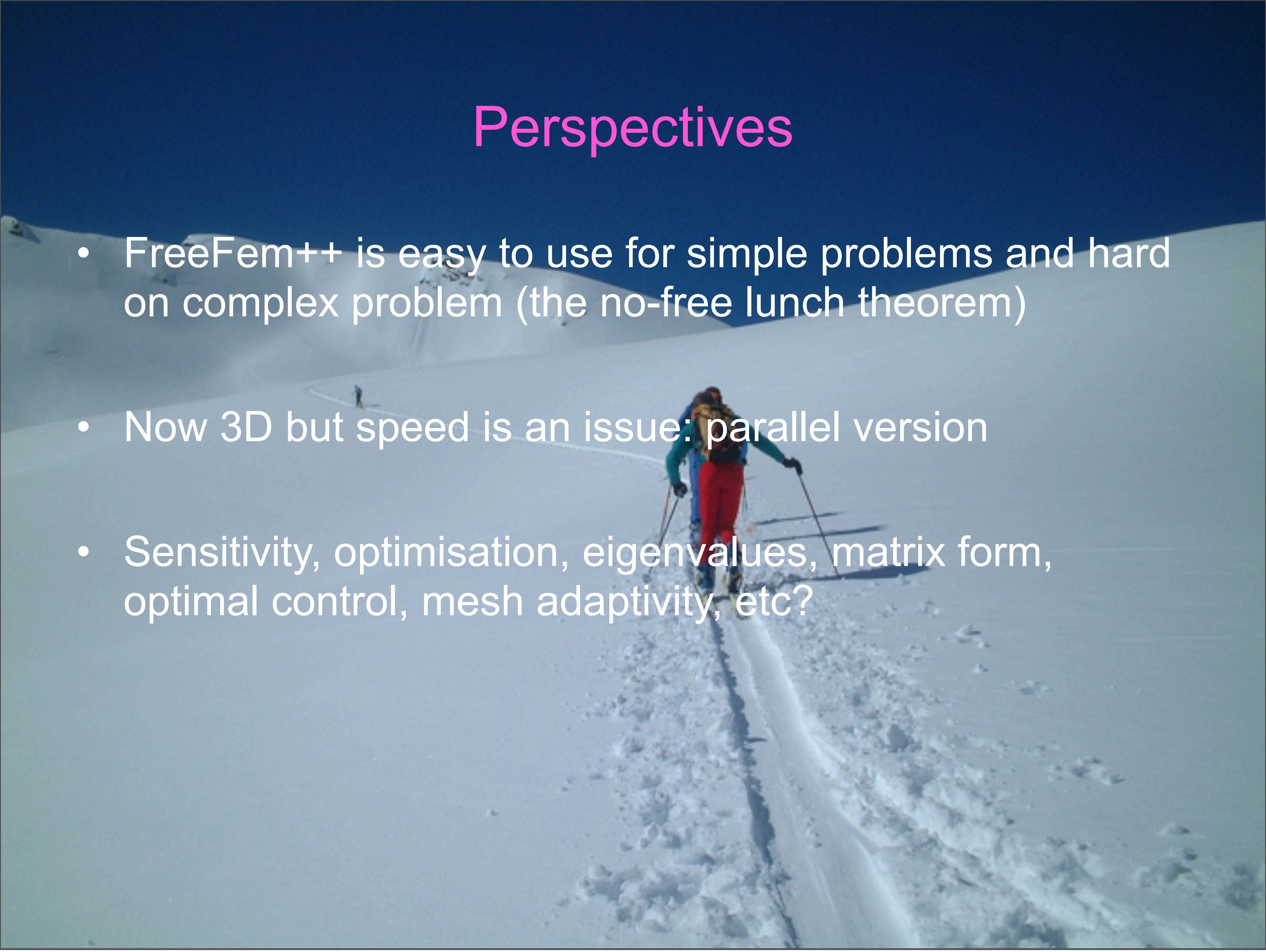
```
mesh th = square(10,10);
fespace Vh(th,P1);
func f= exp(x) * sin(y) ;
Vh u,w, uold=x*(x-1)*y*(y-1);

problem A(u,w,solver=LU)
        =int2d(th)((1+uold^3) *(dx(u)*dx(w)+ dy(u)*dy(w)))
- int2d(th)(f*w) + on(1,2,3,4,u=0);

for(int m=0;m<5;m++){
    A; w=u-uold;
    plot(w,wait=true,value=true);
    uold=u;
}
```

# Optimization e.g. $-\nabla\cdot((1+|\nabla u|^2)^p\nabla u)=f,\quad u|_{\partial\Omega}=0$

A better method is to solve, with q=p+1

$$\min_{u\in H^1_0(\Omega)}\int_\Omega(1+|\nabla u|^2)^q-2q\int_\Omega fu$$

```
mesh th = square(10,10);
fespace Vh(th,P1);
fespace Ph(th,P0);
func f=1;
func real F(real v){return (1+v^2)^4; } //v will be |grad(u)|
func real dF(real v){return 8*(1+v^2)^3;}
func real J(real[int] & u) {
    Vh w; w[]=u; // copy array u in the FEM function w
    return int2d(th)(F( dx(w)^2 + dy(w)^2 ) - 8*f*w) ;
}
func real[int] dJ(real[int] & u) {
    Vh w;w[]=u;
    Ph rho=dF( dx(w)^2 + dy(w)^2);
    varf au(uh,vh)=int2d(th)(rho*(dx(w)*dx(vh)+dy(w)*dy(vh))
                    -8*f*vh) + on(1,2,3,4,uh=0);
u= au(0,Vh); //above with vh replaced by the ith hat function
return u;
}
real[int] u(th.nv);
for(int j=0;j<u.n;j++) u[j]=0;
BFGS(J,dJ,u,eps=1.e-6,nbiter=10,nbiterline=10);
Vh w;  w[]=u;  plot(w);
```



# Perspectives

- FreeFem++ is easy to use for simple problems and hard on complex problem (the no-free lunch theorem)

- Now 3D but speed is an issue: parallel version

- Sensitivity, optimisation, eigenvalues, matrix form, optimal control, mesh adaptivity, etc?