New Border Meshes and FEM (Surface, Line) in FreeFEM

Axel Fourmont - Inria Paris
work with Prof Frédéric Hecht

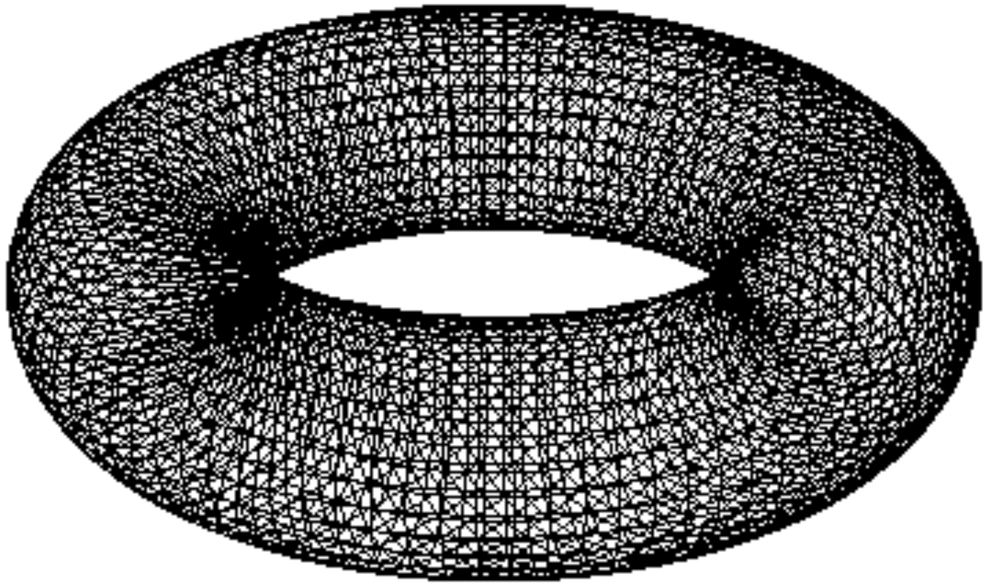
Summary

I

New surface 3d mesh and FEM

bêta version (V4.0) - release version (V4.2)

Example of problem



Problem:
Find a function u defined on Γ such as

$$-\Delta_\Gamma u + u = f \quad \text{+ C.L.}$$

with:

$\Omega \subset \mathbb{R}^3$: A volume domain

$\Gamma = \partial\Omega$: The surface of Ω

Δ_Γ : The Laplace-Beltrami operator

Strong form:

$$\begin{cases} -\Delta_\Gamma u + u = f \\ u = 0 \text{ on } \partial\Gamma \end{cases}$$

Weak form:

$$\int_\Gamma uv - \nabla_\Gamma u \nabla_\Gamma v = \int_\Gamma fv \qquad , \forall v \in H^1(\Gamma)$$

Solve the problem with FreeFEM

Weak form: $$\int_\Gamma uv - \nabla_\Gamma u \nabla_\Gamma v = \int_\Gamma fv \qquad , \forall v \in H^1(\Gamma)$$

HeatTorus.edp

```
load "msh3"

real R = 3, r=1, h = 0.2;
int nx = R*2*pi/h, ny = r*2*pi/h;
func torex= (R+r*cos(y*pi*2))*cos(x*pi*2);
func torey= (R+r*cos(y*pi*2))*sin(x*pi*2);
func torez= r*sin(y*pi*2);

meshS ThS=square3(nx,ny,[torex,torey,torez],removeduplicate=true) ;

fespace VhS(ThS,P1);
VhS u,v;
macro grad3(u) [dx(u),dy(u),dz(u)]  // EOM

problem Lap(u,v) = int2d(ThS)( u*v+grad3(u)'*grad3(v)) -int2d(ThS)
((x+y)*v);
Lap;
plot(u,wait=1,nbiso=20,fill=1);
```

Steps of development to solve this problem with FreeFEM

➡ Define a new type of surface 3d mesh `meshS`

➡ Define a `fespace`, a finite element space on a surface 3D

➡ Define a `problem` or a `varf` to write the weak form of a surface 3D PDE
3.1/ Bilinear form
3.2/ Linear form
3.3/ Apply boundary conditions

➡ Solve the variational problem

➡ PostProcess: `plot`, export the mesh and/or the solution

Solve the problem with FreeFEM

Weak form: $$\int_\Gamma uv - \nabla_\Gamma u \nabla_\Gamma v = \int_\Gamma fv \qquad , \forall v \in H^1(\Gamma)$$

HeatTorus.edp

load "msh3"

```
real R = 3, r=1, h = 0.2;
int nx = R*2*pi/h, ny = r*2*pi/h;
func torex= (R+r*cos(y*pi*2))*cos(x*pi*2);
func torey= (R+r*cos(y*pi*2))*sin(x*pi*2);
func torez= r*sin(y*pi*2);
```

meshS ThS=square3(nx,ny,[torex,torey,torez],removeduplicate=true) ;

```
fespace VhS(ThS,P1);
VhS u,v;
macro grad3(u) [dx(u),dy(u),dz(u)]  // EOM

problem Lap(u,v) = int2d(ThS)( u*v+grad3(u)'*grad3(v)) -int2d(ThS)
((x+y)*v);
Lap;
plot(u,wait=1,nbiso=20,fill=1);
```

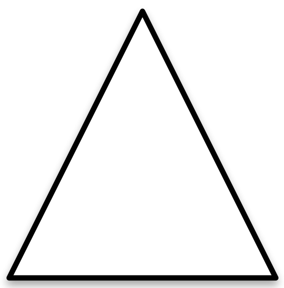
1/ Define a new type of surface 3d mesh

```
class MeshS : public GenericMesh<TriangleS,BoundaryEdgeS,Vertex3>
```

nv vertices 3D

$$\begin{pmatrix} (p_i)_x \\ (p_i)_y \\ (p_i)_z \end{pmatrix}$$

nt triangles



(it_0, it_1, it_2)

nbe edges

(ie_0, ie_1)

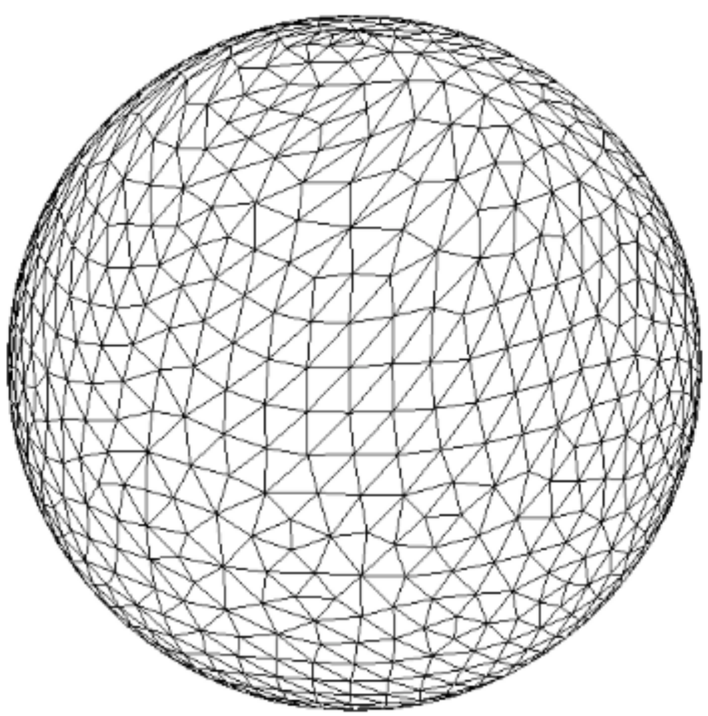
Remark: If border elements (edges) are no given,
FreeFEM automatically builds the real boundary of the mesh,
(algo based on the adjacency of the elements)

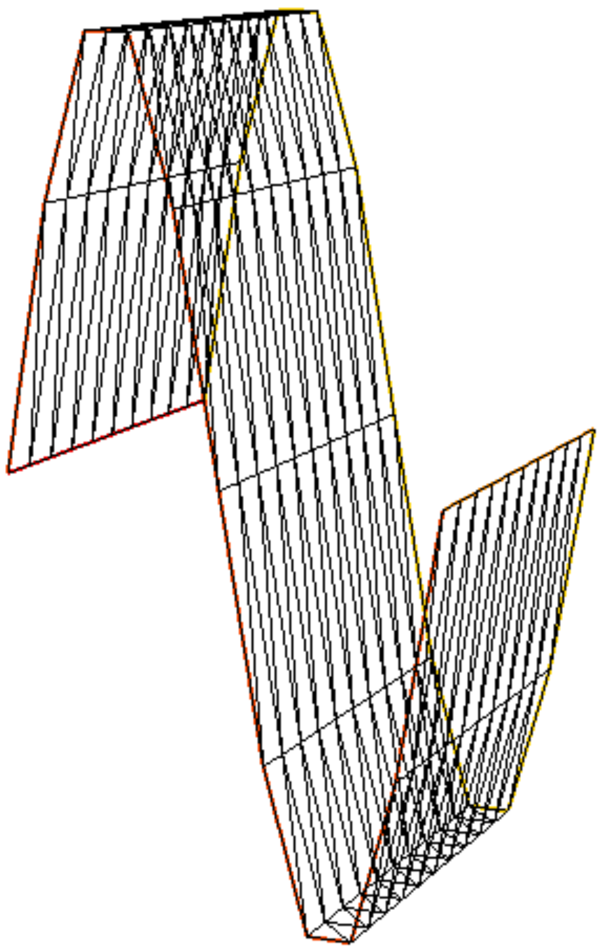
Example format Medit input

```
MeshVersionFormatted 2
Dimension 3

Vertices
NbVertices
(v0)x (v0)y (v0)z
...
(vn)x (vn)y (vn)z

Triangles
NbTriangles
Vertex1 Vertex2 Vertex3 Label
...
Vertex1 Vertex2 Vertex3 Label

Edges
NbEdges
Vertex1 Vertex2 Label
...
Vertex1 Vertex2 Label

End
```

2 possibilities to define a `meshS` :

- the considered domain is a 3D surface, so naturally the FreeFEM type is a `meshS`
- let Th3 be a volume mesh (`mesh3`) and its border Γ
  defined a surface $ThS \subset Th3$ [`mesh3` $\subset$ `meshS`]
  by default in FreeFEM $ThS = \Gamma$





The FreeFEM type: `meshS ThS`

From external types (ff-format, medit, vtk, gmsh)

✓ *medit format*

```
load "msh3"
meshS ThS=readmeshS("ThS.mesh");
```

✓ *ff format*

```
load "msh3"
meshS Th3ff = readmeshS("ThS.msh");
```

✓ *vtk format*

```
load "iovtk"
meshS ThS=vtkloadS("ThS.vtk");
```

✓ *gmsh format*

```
load "gmsh"
meshS ThS=gmshloadS("ThS.msh");
```

available too for mesh3

Optional argument

```
meshS ThS = <name input function> ( <filename> , cleanmesh= false/true,
        removeduplicate=false/true, precisvertice=1e-6/double, orientation=1/-1),
        ridgeangledetection=8.*atan(1.)/9./double )
```

How to generate it?

Use FreeFEM predefined :

✓ build with predefined forms

```
square3,( SurfaceHex, Sphere, Ellipsoide )
```

```
load "msh3"
      meshS ThS = square3(n, m, [Tx, Ty, Tz], <optional arg>);
```

- n,m generates a n×m grid in the unit square
- [Tx, Ty, Tz] is the geometric transformation from $\mathbb{R}^2$ to $\mathbb{R}^3$

```
include "MeshSurface.idp"  ( since v4.2.1 the type is a meshS )
      meshS ThS = SurfaceHex(N, B, L, orient);
```

- int[int] N = [nx,ny,nz] ; the number of seg in the 3 directions
- real [int,int] B = [[xmin,xmax],[ymin,ymax],[zmin,zmax]] ; the bounding box
- int [int,int] L=[[1,2],[3,4],[5,6]]; // the label of the 6 face left,right, front, back, down, righ
- orient designs a int the give the sens of the reference numbering

```
      meshS ThS = Ellipsoide (RX, RY, RZ, h, L, orient);
```

- RX, RY, RZ are real numbers given by the parametric equations of the ellipsoid
- h is the mesh size
- L is the label
- orient designs a int the give the sens of the reference numbering

```
      meshS ThS = Sphere(R, h, L, orient);
                [= Ellipsoide (R, R, R, h, L, orient) ]
```

How to generate it?

Use FreeFEM operators:

✓ build from the border of a mesh3 (Th3)

```
Th3 = buildBdMesh(Th3)
```

and extract from the border of a mesh3 mesh

```
meshS ThS = Th3.Gamma;
```

✓ extract a labeled part of a mesh3

```
mesh3 Th = cube(nn, nn, nn, label=labs);
int[int] llabs = [1, 2];
meshS ThS = extract(Th,label=llabs);
```

✓ create a 3D plane from a 2D mesh (*a projection of a 2d plane in* $\mathbb{R}^3$)

```
mesh Th = square(10,10);
real theta = pi/2;
meshS ThS = movemesh23(Th, transfo=[x, cos(theta)*y-sin(theta)*z,
                                    sin(theta)*y+cos(theta)*z]);
```

Remarks:

- ✓ `tetg` the function for the tetrahelization of 3D volume

```
meshS ThS= […];
meshS Th3=tetg(ThS,switch="paAAQYY");
```

- ✓ If a mesh3 contains a meshS, all call of meshing operator applied on `mesh3` is implicitly apply on its `meshS`

Warning:

Since the release 4.2.1, the surface `mesh3` object (list of vertices and border elements, without tetahedra elements) is replaced by `meshS` type.

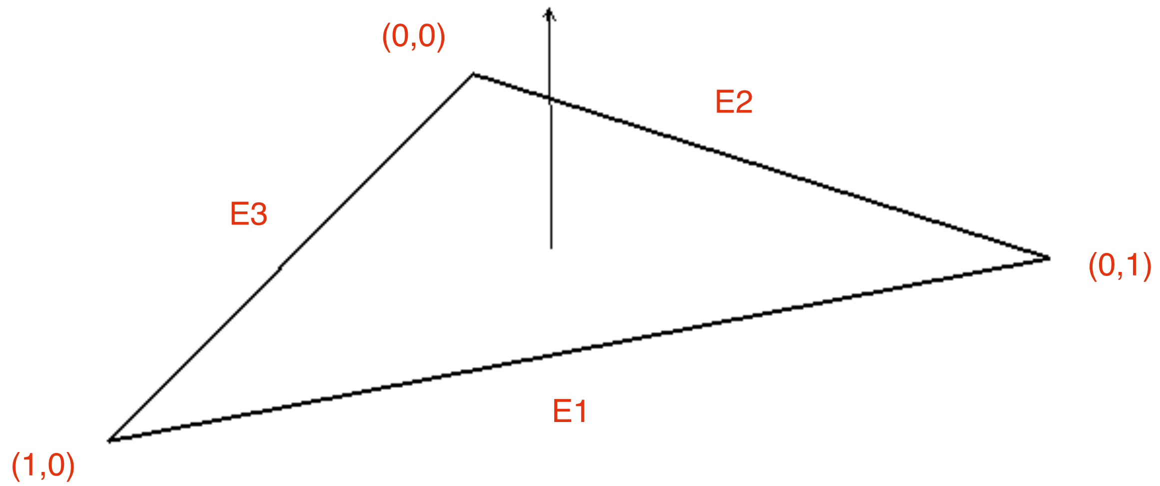
➡ For a FreeFEM V3 script working with surface meshes, try to change `mesh3` by `meshS`

2 types of Normal:
exterior to the surface N and normal to the surface Nt

Normal at each triangle element denote Nt

By definition $\vec{N}_t = \vec{E}_2 \wedge \vec{E}_1$

Remark: Th sens of Normal Nt is implicitly given by the element orientation



Remark:
Needed to apply an external form in the variational formulation

For example $$\int_\Omega f.N_t \partial X$$

Principal operators for `meshS` *type*

- `ThS = movemesh(ThS, [Tx,Ty,Tz], region=…,label=…,orientation=…,)`
- `meshS ThS = trunc(TS1, boolean function, split = …, label = …)`
- `ThS = change(ThS, reftri=…, refedge=…, flabel=…, fregion=…)`
- gluing of `meshS` with the operator + `ThS = ThS1 + Th2 + Th3…`
- `ThS = extract( Th3, refface=…, label=…)`
- `Th3 = buildBdMesh(Th3);`
- `Th = checkmesh(Th, precisvertice=…,removeduplicate=…)`
- clean mesh during the input generation (remove duplicate vertices and/or elements / border elements

Solve the problem with FreeFEM

Weak form: $$\int_\Gamma uv - \nabla_\Gamma u \nabla_\Gamma v = \int_\Gamma fv$$

HeatTorus.edp

```
load "msh3"

real R = 3, r=1, h = 0.2;
int nx = R*2*pi/h, ny = r*2*pi/h;
func torex= (R+r*cos(y*pi*2))*cos(x*pi*2);
func torey= (R+r*cos(y*pi*2))*sin(x*pi*2);
func torez= r*sin(y*pi*2);

meshS ThS=square3(nx,ny,
[torex,torey,torez],removeduplicate=true) ;
fespace VhS(ThS,P1);
VhS u,v;
macro grad3(u) [dx(u),dy(u),dz(u)]  // EOM

problem Lap(u,v) = int2d(ThS)( u*v+grad3(u)'*grad3(v))
-int2d(ThS)((x+y)*v);
Lap;
plot(u,wait=1,nbiso=20,fill=1);
```

Allow to build a FESpace coupling a meshS and a surface 3d FE

Defining finite element space:

```
fespace name(mymesh, typeFE);
```

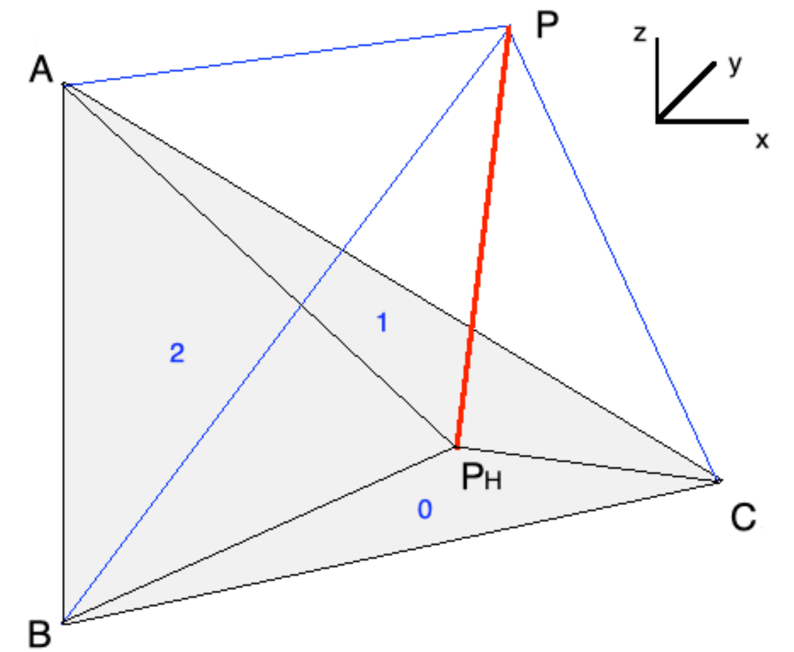
```
fespace VhS(ThS,P1);
```

`meshS`

Finite Element surface 3D: `P0` `P1`, `P2`, `P1b` Lagrange finite elements

➡ defines VhS as the space of Lagrange P1 elements on the meshS ThS

3D surface Pk Lagrange



Definition of P1-Lagrange basis functions

Properties in $\mathbb{R}^3$

Let $\vec{N}$ be the normal to the tangent plane generated by $(A_0, \overrightarrow{A_0A_1}, \overrightarrow{A_0A_2})$

$\vec{N} = \overrightarrow{A_0A_1} \wedge \overrightarrow{A_0A_2}$ with $\wedge$ the usual vectorial in $\mathbb{R}^3$

$$\mathcal{A}(ABC) = \frac{1}{2} | < \vec{N}, \vec{N} > | \qquad (2D)$$

$$\mathcal{A}^S(ABC) = \frac{1}{2} < \vec{N}, \vec{N} > \qquad \mathcal{A}^S(PBC) = \frac{1}{2} < \vec{N}_0, \vec{N} >$$

➡ Barycentric coordinates are equivalent to the ratio of vectorial area in the triangle K

2D generalization
Basis functions

$$\lambda_0(P) = \mathcal{A}^S(PBC)/\mathcal{A}^S(ABC)$$
$$\lambda_1(P) = \mathcal{A}^S(APC)/\mathcal{A}^S(ABC)$$
$$\lambda_2(P) = \mathcal{A}^S(ABP)/\mathcal{A}^S(ABC)$$

$$\boxed{\lambda_i(P) = \frac{< \vec{N}_i(P), \vec{N} >}{< \vec{N}, \vec{N} >}}$$

and the 1st derivative

$$\boxed{\nabla \lambda_i = \frac{< \vec{N}, \vec{E}_i >}{< \vec{N}, \vec{N} >}}$$

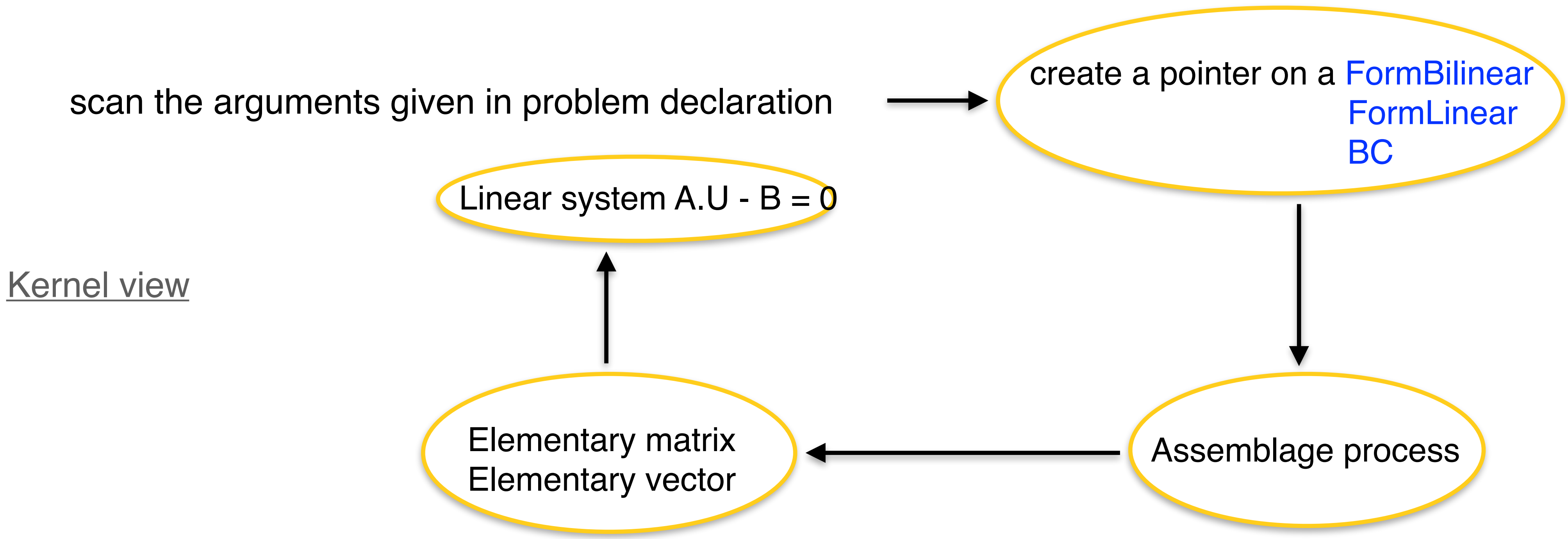
Solve the problem with FreeFEM

<u>Weak form:</u> $$\int_\Gamma uv - \nabla_\Gamma u \nabla_\Gamma v = \int_\Gamma fv$$

HeatTorus.edp

```
load "msh3"

real R = 3, r=1, h = 0.2;
int nx = R*2*pi/h, ny = r*2*pi/h;
func torex= (R+r*cos(y*pi*2))*cos(x*pi*2);
func torey= (R+r*cos(y*pi*2))*sin(x*pi*2);
func torez= r*sin(y*pi*2);

meshS ThS=square3(nx,ny,
[torex,torey,torez],removeduplicate=true) ;

fespace VhS(ThS,P1);
VhS u,v;
macro grad3(u) [dx(u),dy(u),dz(u)] // EOM
solve Lap(u,v) = int2d(ThS)( u*v+grad3(u)'*grad3(v))
                 -int2d(ThS)((x+y)*v);
plot(u,wait=1,nbiso=20,fill=1);
```

Build a FreeFEM linear system



Possible to define the variational form with varf to manipulate matrices and vectors

```
varf a(uVar,vVar)
matrix matrix AVar =a(Vh,Vh);
real[int] bVar = a(0,Vh);
real[int] solVar = AVar^-1*bVar;
```

Keywords:

```
varf    problem
int2d   int1d
on
```
TODO `intalledges`

Solve the problem with FreeFEM

Weak form: $$\int_\Gamma uv - \nabla_\Gamma u \nabla_\Gamma v = \int_\Gamma fv$$

HeatTorus.edp

```
load "msh3"

real R = 3, r=1, h = 0.2;
int nx = R*2*pi/h, ny = r*2*pi/h;
func torex= (R+r*cos(y*pi*2))*cos(x*pi*2);
func torey= (R+r*cos(y*pi*2))*sin(x*pi*2);
func torez= r*sin(y*pi*2);

meshS ThS=square3(nx,ny,
[torex,torey,torez],removeduplicate=true) ;

fespace VhS(ThS,P1);
VhS u,v;
macro grad3(u) [dx(u),dy(u),dz(u)]  // EOM

problem Lap(u,v) = int2d(ThS)( u*v+grad3(u)'*grad3(v))
-int2d(ThS)((x+y)*v);
```

```
Lap;
plot(u,wait=1,nbiso=20,fill=1)
```

Save/export a type `meshS`

✓ *medit format*

```
load "msh3"
savemesh(Th3, "surf.mesh");
```

From a `meshS`,

```
savesurfacemesh(Th3, "surf.mesh");
```

✓ *FreeFEM format*

```
load "msh3"
savemesh(Th3, "surf.msh");
```

✓ *vtk format*

```
load "iovtk"
savevtk("ThS.vtk",ThS);
```

✓ *gmsh format*

```
load "gmsh"
savegmsh(ThS, "ThS");
```

Save/export a solution

✓ *FreeFEM format*

extension .sol and .solb

✓ *vtk format*

extension .vtk, .vtu

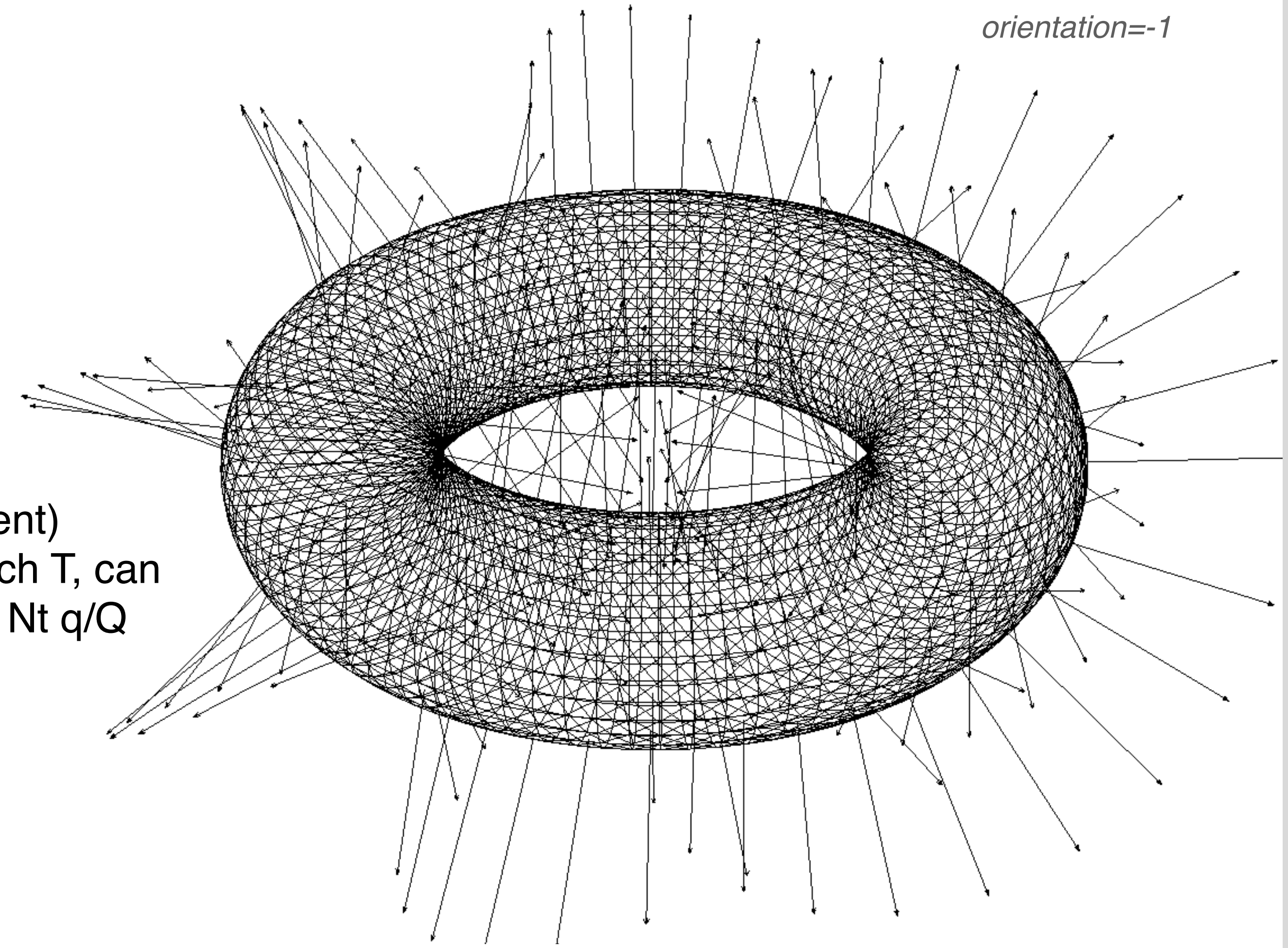
Minimal example:

```
meshS ThS=square3(10,10);

fespace Uh(ThS,P1);
Uh u = x;

load "iovtk"
int[int] fforder=[1];
savevtk("sol.vtk",ThS,u,dataname="u",order=fforder);
```

Remark:

All `savevtk` function uses binary format by default.

Visualisation with ff-glut v4.1



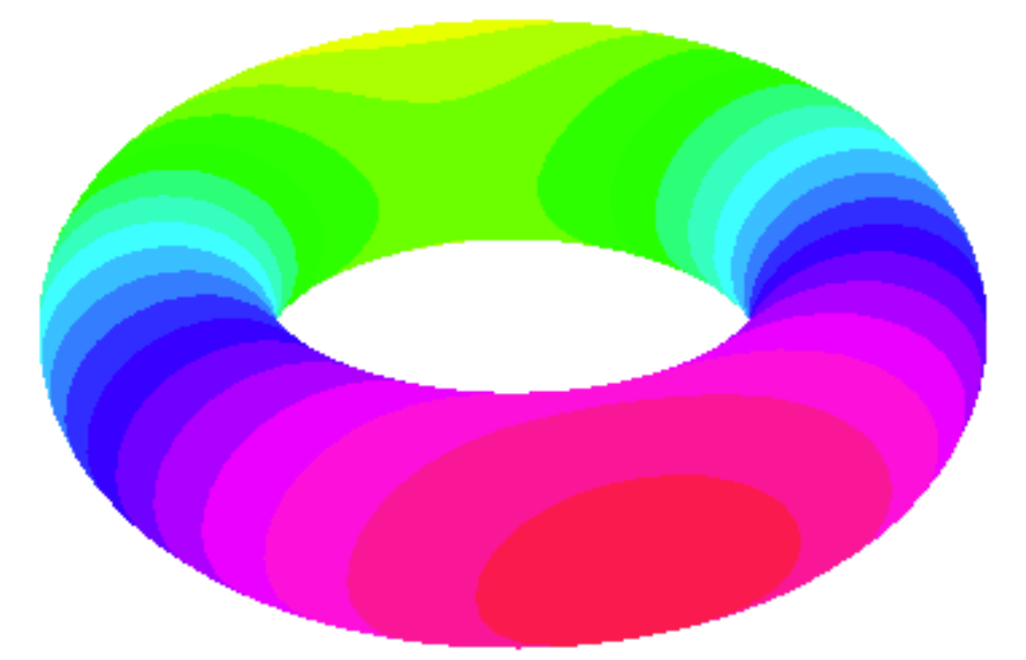
Same functionalities that a mesh3/mesh

- plot a `meshS`
- plot a surface scalar solution
- *TODO* a vectorial solution
- plot surface normal (normal at each triangle element)
  - the plotting of Nt is available with the touch T, can
  - decrease/increase the number of plotted Nt q/Q
  - decrease/increase the arrow size a/A

Note:

medit soft can be use

```
load "medit"
medit("u",Th,u);
```



II

New line 3d mesh and FEM

bêta version (V4.4-3)

Define a new type of curve 3d mesh

```
class MeshL : public GenericMesh<EdgeL,PointL,Vertex3>
```

nt edges	nbe border points	nv vertices 3D
(ie_0, ie_1)		$\begin{pmatrix}(p_i)_x\\(p_i)_y\\(p_i)_z\end{pmatrix}$

Example format Medit input

```
MeshVersionFormatted 2
Dimension 3

Vertices
NbVertices
(v0)x (v0)y (v0)z
...
(vn)x (vn)y (vn)z

Edges
NbEdges
Vertex1 Vertex2 Label
...
Vertex1 Vertex2 Label

End
```

Warning:
Input formats don't have " border point type "

➡ FreeFEM detects them when generating a meshL type

Define a Building a meshL

From external types (ff-format, medit, vtk)

✓ *medit format*

```
load "msh3"
meshL ThL=readmeshL("ThL.mesh");
```

✓ *ff format*

```
load "msh3"
meshL ThL = readmeshL("ThL.msh");
```

✓ *vtk format*

```
load "iovtk"
meshL ThL=vtkloadL("ThL.vtk");
```

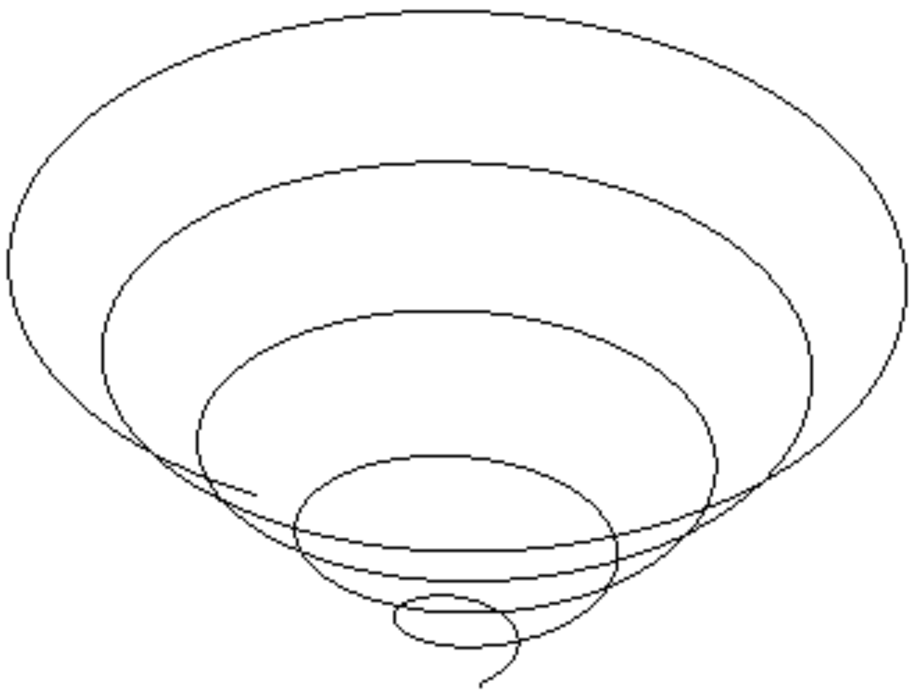
Define a Building a meshL

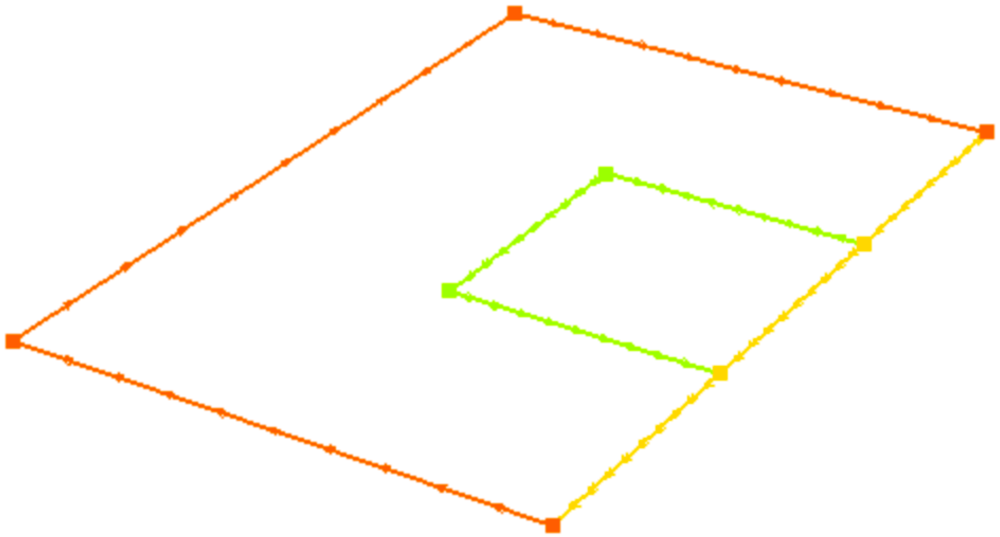
With `border` type

```
load "msh3"

int upper = 1, others = 2, inner = 3, n = 10;
border D01(t=0, 1){x=0; y=-1+t; z=t; label=upper;}
border D02(t=0, 1){x=1.5-1.5*t; y=-1; z=3;label=upper;}
border D03(t=0, 1){x=1.5; y=-t; z=3;label=upper;}
border D04(t=0, 1){x=1+0.5*t; y=0; z=3;label=others;}
border D05(t=0, 1){x=0.5+0.5*t; y=0; z=3;label=others;}
border D06(t=0, 1){x=0.5*t; y=0; z=3;label=others;}
border D11(t=0, 1){x=0.5; y=-0.5*t; z=3;label=inner;}
border D12(t=0, 1){x=0.5+0.5*t; y=-0.5; z=3;label=inner;}
border D13(t=0, 1){x=1; y=-0.5+0.5*t; z=3;label=inner;}

plot(D01(-n) + D02(-n) + D03(-n) + D04(-n) + D05(-n)
   + D06(-n) + D11(n) + D12(n) + D13(n), wait=true);

meshL ThL=buildmeshL(D01(-n) + D02(-n) + D03(-n) + D04(-n) + D05(-n)
    + D06(-n) + D11(n) + D12(n) + D13(n));

savemesh(ThL,"myTh.mesh");
```

Define a Building a meshL

With* `border` *type





previous building

```
border E1(t=0, 10.*pi){x=(1.)*t*cos(t);
                       y=-(1.)*t*sin(t);
                       z=t;}
meshL ThL=buildmeshL(E1(1000));
plot(ThL2);
```

Define a Building a meshL

From a meshS

- ✓ build from the border of a meshS (ThS)

```
ThS = buildBdMesh(ThS)
```

  and extract from the border of a meshS mesh

```
meshL ThL = ThS.Gamma;
```

- ✓ extract a labeled part of a meshS

```
meshS Th = square3(nn, nn);
int[int] llabs = [1, 2];
meshL ThL = extract(Th,label=llabs);
```

With `Sline` builder

```
meshL Th = Sline(nn, [Tx, Ty, Tz]);
```

Principal operators for `meshL` type

- `ThL = movemesh(ThL, [Tx,Ty,Tz], region=…,label=…,orientation=…,)`
- `meshL ThL = trunc(TL1, boolean function, split = …, label = …)`
- gluing of `meshL` with the operator + `ThL = Th1 + Th2 + Th3…`
- `ThL = extract( ThS, refedge=…, label=…)`
- `ThS = buildBdMesh(ThS);`
- `Th = checkmesh(Th, precisvertice=…,removeduplicate=…)`
- clean mesh during the input generation (remove duplicate vertices and/or elements / border elements

Remark: 3D mesh operators are standard, use the same syntax

Solve a elastic wave problem by FEM

$$
\left|
\begin{array}{l}
\rho\epsilon \frac{\partial^2}{\partial^2 t} u + (1+\beta)\lambda_1 \Delta u - \lambda_0 \nabla u + \alpha\rho\epsilon \frac{\partial}{\partial t} u = f_t \\
u|_{x=0} = 0 \\
u(x, t_0) = 0
\end{array}
\right.
$$

with:

ρ	the density solid
ϵ	the thickness
λ_0, λ_1	the Lame coefficient
α, β	the Rayleigh coefficient

Applying a Backward Euler time scheme, the variational problem is

Find $u \in H^1(\Omega)$ such as

$$
\left(\frac{\rho\epsilon}{\Delta t^2} + \lambda_0 + \frac{\alpha\rho\epsilon}{\Delta t}\right) \int_\Omega u^n v + \left(\lambda_1 + \frac{\beta\lambda_1}{\Delta t}\right) \int_\Omega \Delta u^n \Delta v
$$

$$
= \frac{\rho\epsilon}{\Delta t} \int_\Omega \dot{u}^{n-1} v + \frac{\rho\epsilon}{\Delta t^2}(1 + \Delta t\alpha) \int_\Omega u^{n-1} v + \frac{\beta\lambda_1}{\Delta t} \int_\Omega \Delta u^{n-1} \Delta v + \int_\Omega f_t v \quad , \forall v \in H^1(\Omega)/(v|_{x=0} = 0)
$$

- ➡ Finite element line 3D : P0, P1 and P2 Lagrange
- ➡ Keywords:

```
varf     problem
int1d    on
```

Resolution of the elastic wave is available at *example/3dCurve/elasticstring.edp*

main steps:

```
meshL Th=Sline(100,[2.*x,y,z]);  // line 3D mesh
// FE space
fespace Uh(Th,P1);
[…]
varf m(u,v) = int1d(Th) (u*v);    // mass matrix
matrix mass=m(Uh,Uh);
varf k(u,v) = int1d(Th) (Grad(u)'*Grad(v));    // stiffness matrix
matrix stiffness=k(Uh,Uh);
[…]
varf cl(u,v) = on(1,u=0);
```

Resolution of the elastic wave is available at *example/3dCurve/elasticstring.edp*

main steps:

```
// loop time
for (int i=0; i<=iMax; i++) {
       […]
       real coeff=100; // coeff for the visu
       Thmv=movemesh(Th,[x,y,u*coeff]);
       plot(Th, Thmv, wait=0, cmm=" d = "+t+" iter = "+i,prev=1);


    }
```

Save/export a solution

✓ *FreeFEM format*

extension .sol and .solb (`savesol`)

✓ *vtk format*

extension .vtk, .vtu

- ➡ Application to BEM cf Pierre-Henri Tournier's session
- ➡ Add functionalities for line 3D FEM
- ➡ Some examples for border FEM
- ➡ parallel `meshL` type
- ➡ I/O line 3D
- ➡ Add vectorial plot
- ➡ …