Axel Fourmont
Research engineer
Inria Paris

FreeFem++ days

Adding a new FE and the 3D Surface FEM for FreeFem++

Work with Prof Frédéric Hecht

1/ Add a new high order Finite Element in FreeFem++

2/ 3D Surface Finite Elements in FreeFem++

1/ Add a new high order Finite Element in FreeFem++

- Theory on RT1 3d element
- FreeFem++: a constructor of FE
- Development for FreeFem++

Theory about a Finite element: What do we need?

The interpolator operator: $\boxed{\Pi_h \boldsymbol{f} = \sum_{k=0}^{\mathtt{kPi}-1} \alpha_k \boldsymbol{f}_{j_k}(P_{p_k}) \boldsymbol{\omega}_{i_k}^K}$

Let be f a function, taking value in $\mathbb{R}^n,\ (n = 1, 2, 3)$

- NbDOF the num of DOF on the current element K
- $\boldsymbol{\omega}_i^K$ the i-th basis function (if vectorial function $\boldsymbol{\omega}_{ij}^K$)
- α_k the k-th coefficient depending of the integration FE type (kPi coeffs)
- P_p is a set of $npPi$ interpolation points
- $\mathrm{kPi} = \mathrm{npPi} \times \mathrm{n}$

- Read the FreeFem++ documentation, **chapter 13 « Addition of a new finite element »** and the **Appendix D**

Here, it finds:

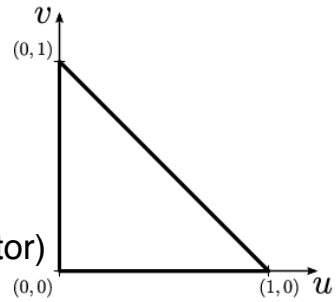
— the definition of a FE according to FreeFem++'s FE implementation
— the technic to add a new FE with a dynamic library
— exemples to understand the inherence of your FE class

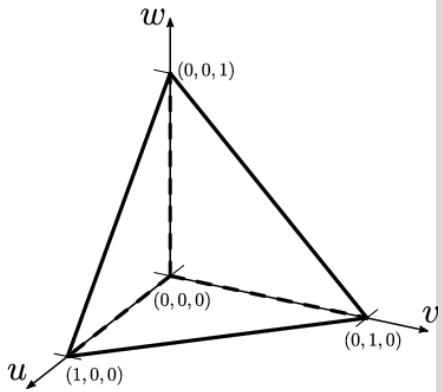
- In **example++-load**, examples of use of new finite elements can be find

Add a new Finite Element in FreeFem++

Theory about a Finite element

- $\hat{K}$ is a reference geometric shape
- Shape functions and its derivatives
- Degree of freedom at mesh node (the interpolation operator)
- Barycentric coordinates





Link with the FE's FreeFem++

2d case

Base class TypeOfFE
The constructor TypeOfFE(——)
void FB(——)
void Pi_h_alpha(——)

3d case

Base class GTypeOfFE<Mesh3>
The constructor GTypeOfFE<Mesh3>(——)
void FB(——)
void set(——)

Barycentric coordinates. $(\lambda_1, \lambda_2, \lambda_3)$ / $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$
Basis functions ϕ_i
Interpolation op Π_h defines the DOFs
The support DOFs Data[ ] / dfon [ ]
Interpolation points / coefficients α_{ij}
Quadrature rules GQuadratureFormular< >

inventeurs du monde numérique

Add a new Finite Element in FreeFem++

A dual use of FreeFem++ to add a new FE

1/ Calculate the basic functions and validate the unisolvence
- Matricial calculus
- Use previous defined Finite element
- Work with FE functions
- Validate on the reference and deformed element
  - Barycentric coordinates
  - Geometric transformation (DOFs permutation)

findMyFE.edp script

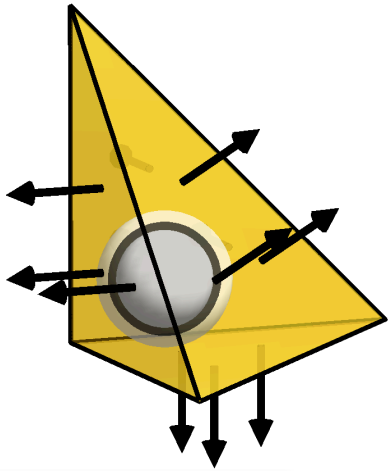
2/ Add the finite element in FreeFem++ to use it for solving PDEs

- write a C++ plugin that defines a C++ class by inherence
- Define the basis functions in the local geometry shape
- Define the interpolator operator

myFE.cpp and generate a library
with ff-c++
and use in .edp with load " myFE "

Add the RT1 3d FE

The 1st order Raviart Thomas FE 3d *(H(div) FE)*



Notations

d dimension	f_i face
K current d-simplex	n_i exterior normal to the face f_i
Q_i vertice	λ_i barycentric coordinate of $\hat{K}$

$\hat{K}$, the reference tetrahedron $Q_0\ (0,\ 0,\ 0) - Q_1\ (1,\ 0,\ 0) - Q_2\ (0,\ 1,\ 0) - Q_3\ (0,\ 0,\ 1)$

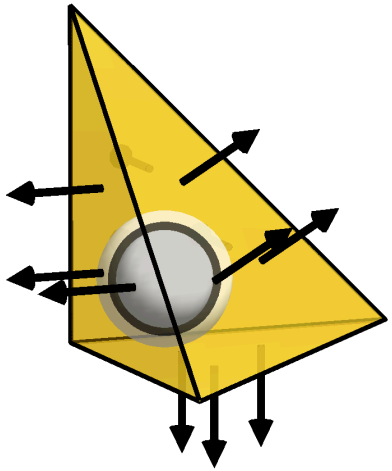
The k-th order Raviart-Thomas element $RT_k(K),\ k \in \mathbb{N}_0$, is defined by the polynomial space

$$RT_k(K) = \mathbb{P}_k(K)^d + x\tilde{\mathbb{P}}_k(K)$$

$\tilde{\mathbb{P}}$ —> is the space of monomials of degree k

- $\dim\ RT_0 = 4$ $\quad |\mathbb{P}_0|^3 + x\mathbb{P}_1$ 4 normal components
- $\dim\ RT_1 = 15$ $\quad |\mathbb{P}_1|^3 + x\mathbb{P}_2$ 12 normal components and 3 interior moments

The 1st order Raviart Thomas FE 3d *(H(div) FE)*



RT1 3D FE definition

- $\hat{K}$ is the reference tetrahedron
- $RT_1 = \mathbb{P}_1(\hat{K})^2 + x\tilde{\mathbb{P}}_1(\hat{K})$
- $\Sigma(\hat{K})$ design the set of the linear form $\sigma(\Psi)$ such as
  - $\sigma(\Psi_{ke}) = \int_{f_i} \Psi_{ke} \cdot \vec{n_i} q \, \mathrm{d}\tau = 0, \forall q \in \mathbb{P}_k(f_i)$ on face dof support and
  - $\sigma(\Psi_{kb}) = \int_K \Psi_{kb} q \, \mathrm{d}x = 0, \forall q \in (\mathbb{P}_{k-1})^2$ on bubble dof support

Basis functions Ψ for RT1 3d

$$RT_1(K) = \mathbb{P}_1(K)^3 + x\tilde{\mathbb{P}}_1(K)$$

Remark: The space defined by $RT0 \times \mathbb{P}_1(\hat{K})$ forms a generative basis of $RT_1(\hat{K})$

$$b(X) = \{bk = \Phi_i \lambda_j\} \text{ such as } 0 \leq i, j \leq 3$$

$$\begin{array}{llll} bk_0 = \Phi_0\lambda_0(i); & bk_1 = \Phi_0\lambda_1(e); & bk_2 = \Phi_0\lambda_2(e); & bk_3 = \Phi_0\lambda_3(e); \\ bk_4 = \Phi_1\lambda_0(e); & bk_5 = \Phi_1\lambda_1(i); & bk_6 = \Phi_1\lambda_2(e); & bk_7 = \Phi_1\lambda_3(e); \\ bk_8 = \Phi_2\lambda_0(e); & bk_9 = \Phi_2\lambda_1(e); & bk_{10} = \Phi_2\lambda_2(i); & bk_{11} = \Phi_2\lambda_3(e); \\ bk_{12} = \Phi_3\lambda_0(e); & bk_{13} = \Phi_3\lambda_1(e); & bk_{14} = \Phi_3\lambda_2(e); & bk_{15} = \Phi_3\lambda_3(i); \end{array}$$

2 sets are available : the bubble functions $b_{kb} = \{bk_i | i \in [0, 5, 10, 15]\}$
the faces functions $b_{kf} = \{bk_i | i \in [1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14]\}$

→ *hypothesis:* There are 16 monomial functions to build the basis functions of RT1 the bubble functions edge are independent.

Arbitrary, we can chose $b_{kb} = \{bk_0, bk_5, bk_{10}\}$

Add the RT1 3d FE

Basis functions Ψ for RT1 3d

$$RT_1(K) = \mathbb{P}_1(K)^3 + xRT0 \times \mathbb{P}_1(K)$$

The linear form for bubble-type DOFs:

$$\sigma(\Psi_{kb}) = \int_K \Psi_{kb} q \mathrm{d}x = 0, \forall q \in \mathbb{P}_0(\mathbb{R}^3) \qquad (1)$$

search 3 interior FB

Put $\mathrm{q} \in \mathbb{P}_0(\hat{K})$,

$$\Psi_{kb} = \sum_{i=0}^{2} a_i \Psi_i \lambda_i \rightarrow$$

$$\begin{pmatrix} \Psi_{12} \\ \Psi_{13} \\ \Psi_{14} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} bk_0 \\ bk_5 \\ bk_{10} \end{pmatrix}$$

Add the RT1 3d FE

Basis functions Ψ for RT1 3d

$$RT_1(K) = \mathbb{P}_1(K)^3 + xRT0 \times \mathbb{P}_1(K)$$

The linear form for face-type DOFs:

$$\sigma(\Psi_{kf}) = \int_{f_i} \Psi_{ke} \cdot \vec{n_i} q \, \mathrm{d}\tau = 0, \forall q \in \mathbb{P}_0(\mathbb{R}^3) \qquad (2)$$

search 12 face FB

Put $q \in \mathbb{P}_0(\hat{K})$,

$$\Psi_{kf} = bk_i^e + \sum_{l=0}^{2} c_{kj} bk_l^b \cdot (C_1 C_2 C_3)$$

Special normal basis $(C_1 C_2 C_3)$

—> must be Piola conservative

$$\begin{pmatrix} \Psi_0 \\ \Psi_1 \\ \Psi_2 \\ \Psi_3 \\ \Psi_4 \\ \Psi_5 \\ \Psi_6 \\ \Psi_7 \\ \Psi_8 \\ \Psi_9 \\ \Psi_{10} \\ \Psi_{11} \end{pmatrix} = \begin{pmatrix} bk_1 \\ bk_2 \\ bk_3 \\ bk_4 \\ bk_6 \\ bk_7 \\ bk_8 \\ bk_9 \\ bk_{11} \\ bk_{12} \\ bk_{13} \\ bk_{14} \end{pmatrix} + \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \\ a_{3,0} & a_{3,1} & a_{3,2} \\ a_{4,0} & a_{4,1} & a_{4,2} \\ a_{5,0} & a_{5,1} & a_{5,2} \\ a_{6,0} & a_{6,1} & a_{6,2} \\ a_{7,0} & a_{7,1} & a_{7,2} \\ a_{8,0} & a_{8,1} & a_{8,2} \\ a_{9,0} & a_{9,1} & a_{9,2} \\ a_{10,0} & a_{10,1} & a_{10,2} \\ a_{11,0} & a_{11,1} & a_{11,2} \end{pmatrix} \begin{pmatrix} bk_0 \\ bk_5 \\ bk_{10} \end{pmatrix}$$

Add the RT1 3d FE

Use of FreeFem++'s power

Resume: Build the RT1 3d elements basis functions with FreeFem script

1/ Define the reference element
2/ Build the local orthonormal basis $(C_1 C_2 C_3)$, respecting the Piola transformation
3/ Create the RT0 and P1 FE space
4/ Use the P23d to save the RT1 FE variables
5/ Save the sign of permutation to manage the orientation of faces and edges
6/ Build the 16 monomials to create a generative basis of $RT_1(\hat{K})$ and remove one
7/ Solve the system associated to the bubble DOFs —> 3 Interior BF are found
8/ Solve the system associated to the face DOFs —> 12 face BF are found
9/ Manage the permutation for support orientation DOFs according the numbering of the FB
10/ Build the basis functions
11 / Check the linear form of the DOFS —> unisolvance

Add the RT1 3d FE

Adding the RT1 element for FreeFem++

Source in exemples++-load/Element_Mixte3d.cpp

```
#include "ff++.hpp"
#include "AddNewFE.h"
```

To add a new finite element to FreeFem++, write a C++ plugin that defines a C++ class

Declaration of the new FE class

```
class TypeOfFE_RT1_3d: public GTypeOfFE<Mesh3>  {
    public:
        typedef Mesh3 Mesh;
        typedef  Mesh3::Element Element;
        typedef GFElement<Mesh3> FElement;
        static int dfon [];
        static const int d = Mesh::Rd::d;
        // quadrature Formula on a face
        static const GQuadratureFormular<R2> QFface;
        // quadrature Formula on an element
        static const GQuadratureFormular<R3> QFtetra;
        TypeOfFE_RT1_3d ();
        int edgeface[4][3];
        void FB (const What_d whatd, const Mesh &Th, const Mesh3::Element &K,
                 const RdHat &PHat, RNMK_ &val) const;
        void set (const Mesh &Th, const Element &K, InterpolationMatrix<RdHat> &M,
                  int ocoef, int odf, int *nump) const;
};
```

Adding the RT1 element for FreeFem++

Source in exemples++-load/Element_Mixte3d.cpp

Need to initialize some variables

```
int TypeOfFE_RT1_3d::dfon [] = {0, 0, 3, 3};// dofs per vertice, edge, face, volume

// Quadrature formula on a face
const GQuadratureFormular<R2> TypeOfFE_RT1_3d::QFface(QuadratureFormular_T_5);

// Quadrature formula on the tetrahedron
const GQuadratureFormular<R3> TypeOfFE_RT1_3d::QFtetra(QuadratureFormular_Tet_2);
```

The new FE class derivates from the base class GTypeOfFE<Mesh3>

The class constructor: `TypeOfFE_RT1_3d::TypeOfFE_RT1_3d (): GTypeOfFE<Mesh3>( … )`

Adding the RT1 element for FreeFem++

Source in exemples++-load/Element_Mixte3d.cpp

```
void TypeOfFE_RT1_3d::set (const Mesh &Th, const Element &K, InterpolationMatrix<RdHat>
                           &M, int ocoef, int odf, int *nump) const {
```

Here, the coefficient α_k ***is built***

$$\boxed{\Pi_h \boldsymbol{f} = \sum_{k=0}^{\texttt{kPi}-1} \alpha_k \boldsymbol{f}_{j_k}(P_{p_k}) \boldsymbol{\omega}_{i_k}^K}$$

Adding the RT1 element for FreeFem++

Source in exemples++-load/Element_Mixte3d.cpp

```
void TypeOfFE_RT1_3d::FB (const What_d whatd, const Mesh &Th,
                  const Mesh3::Element &K, const RdHat &PHat, RNMK_ &val) const {
```

Here it builds the 15 basis functions and their derivatives

Conclusion: The bootstrap is the strength of FreeFem++
(use FreeFem++ to build a new FE)

1/ A using of FreeFrem++ to find the theory about a finite element
- build basis functions
- check the linear form given by the Degree Of Freedom

2/ Add your own FE with a creation of derivate class without enter in the FreeFem++'s kernel

Remark: Be careful with the geometric orientation of the element
—> **face permutations**
—> **edge permutations on face**

2/ 3D Surface Finite Elements in FreeFem++

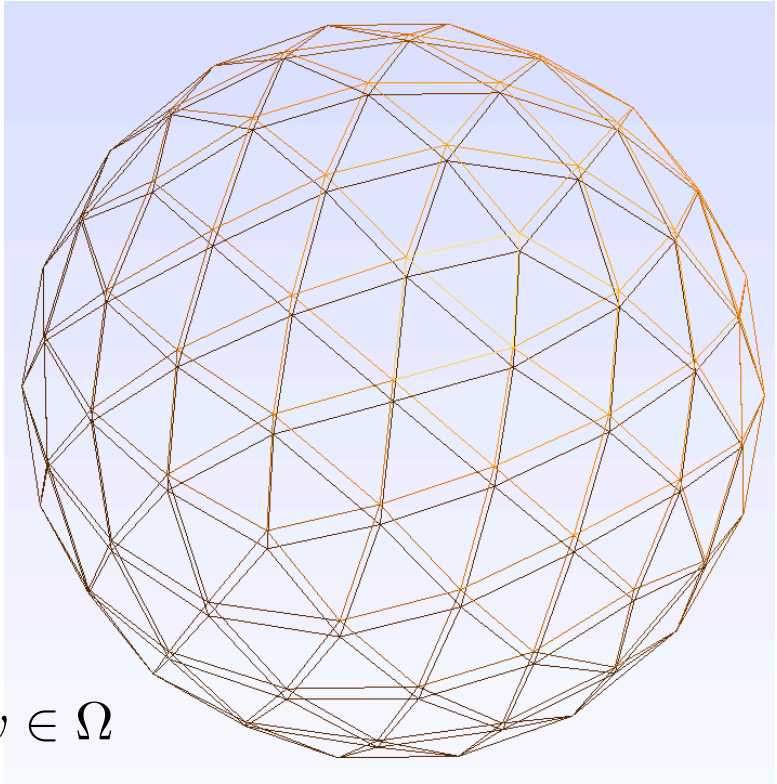
- A new type of mesh
- A new type of FE type
- The surface Pk Lagrange FE
- The surface FE space
- The assemblage step and resolution

Example of problem

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

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

Δ_{N-1} The Laplace-Beltrami operator



Problem:
Find a function u defined on Ω *such as*

$$-\Delta_{N-1}u = f \text{ +c.i.}$$

Strong form: $-\Delta_{N-1}u = f$

Weak form: $\int_\Omega \nabla_{N-1}u.\nabla_{N-1}v = \int_\Omega fv, \forall v \in \Omega$

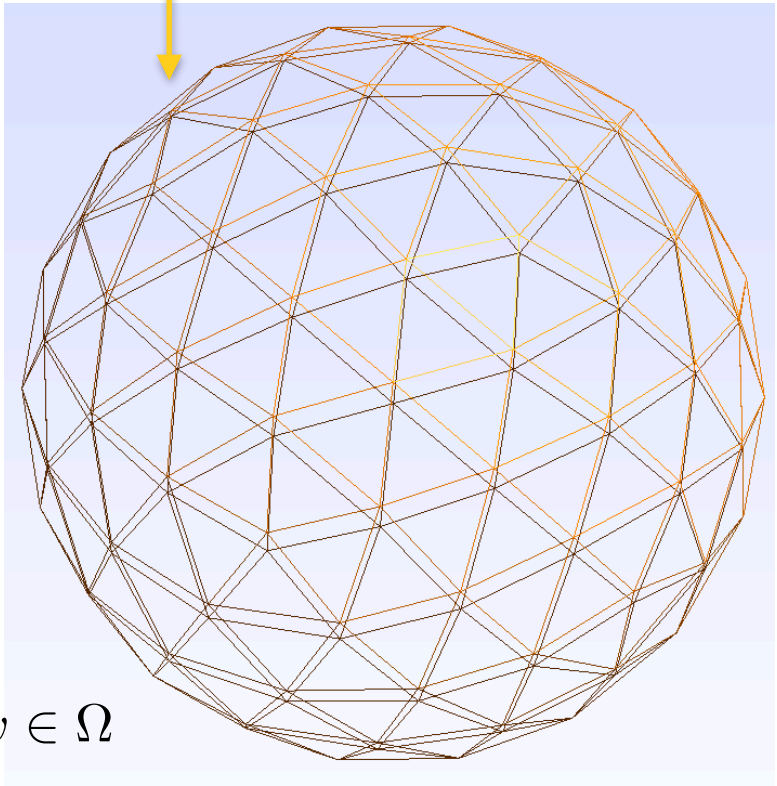
Example of problem

FreeFem's input ?
mesh given by vertices, triangles, edges

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

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

Δ_{N-1} The Laplace-Beltrami operator



Problem:
Find a function u defined on Ω *such as*

$$-\Delta_{N-1}u = f \text{ +c.i.}$$

Strong form: $-\Delta_{N-1}u = f$

Weak form: $\int_\Omega \nabla_{N-1}u.\nabla_{N-1}v = \int_\Omega fv, \forall v \in \Omega$

inventeurs du monde numérique

Steps to solve the problem with FreeFem++?

Solve the weak form $\int_\Omega \nabla_{N-1} u . \nabla_{N-1} v = \int_\Omega f v, \forall v \in \Omega$

laplace-beltrami.edp

```
load "msh3"
load "medit"
mesh3 Th3dS = readmesh3("ThSurf.mesh");

fespace VhS(ThS,P1S);
VhS uS,vS;

macro Grad3(uSVar) [dx(uSVar),dy(uSVar),dz(uSVar)]  // EOM

problem Lap3dS(uSPb,vSPb , solver = CG )  = int2d(ThS)(Grad3(uSPb)'*Grad3(vSPb))
 - int2d ( ThS ) ( f * vSPb )
  + on(1,uSPb=g);

Lap3dS;

plot(Th, uS, wait=1);
```

Steps to solve the problem with FreeFem++?

Solve the weak form $\int_\Omega \nabla_{N-1} u . \nabla_{N-1} v = \int_\Omega f v, \forall v \in \Omega$

laplace-beltrami.edp

```
load "msh3"
load "medit"
mesh3 Th3dS = readmesh3("ThSurf.mesh");

fespace VhS(ThS,P1S);
VhS uS,vS;

macro Grad3(uSVar) [dx(uSVar),dy(uSVar),dz(uSVar)]  // EOM

problem Lap3dS(uSPb,vSPb , solver = CG )  = int2d(ThS)(Grad3(uSPb)'*Grad3(vSPb))
 - int2d ( ThS ) ( f * vSPb )
  + on(1,uSPb=g);

Lap3dS;

plot(Th, uS, wait=1);
```

Steps to solve the problem with FreeFem++?

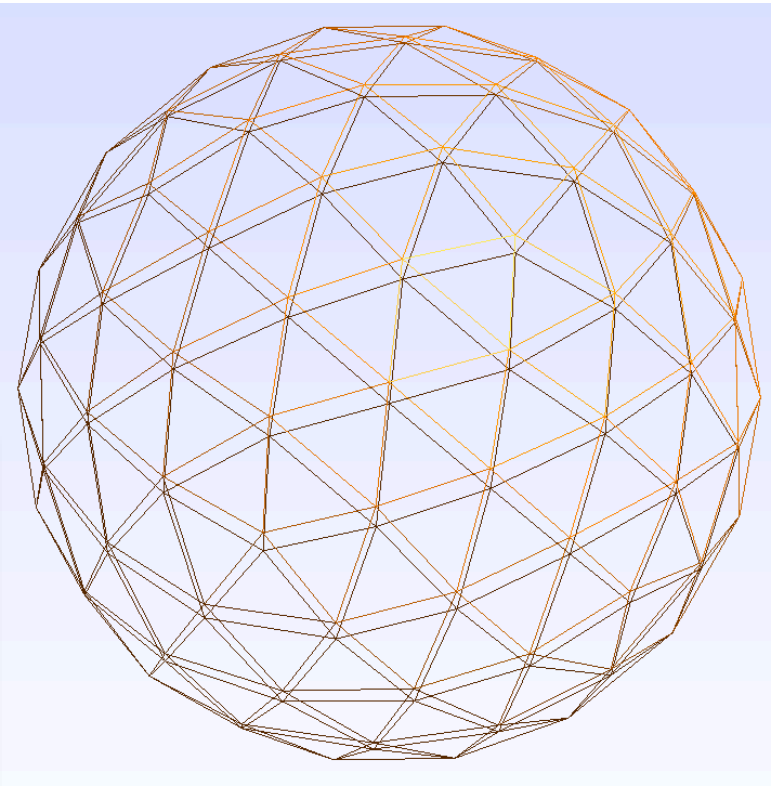
Step to solve the problem

1/ Input mesh of $\Omega \subset \mathbb{R}^3$

2/ Write the weak form in FreeFem++'s language

3/ Solve the associated linear system

4/ Visualisation tools



Now possible in FreeFem++ v4

mesh3, readmesh, savemesh, savesurface, movemesh23, movemesh3

problem / varf + apply BC

solve

plot

Add FreeFem++ class MeshS

3D mesh3 with	Tetrahedron elements	—>	3D Integration domain (x, y, z)	— Dim geometric elt = 3
	Triangle border elements	—>	2D Integration domain (x, y)	— Dim geometric elt = 2
2D mesh with	Triangle elements	—>	2D Integration domain (x, y)	— Dim geometric elt = 2
	Edge border elements	—>	1D Integration domain (x)	— Dim geometric elt = 1
3D meshS with	**Triangles elements**	**—>**	**3D Integration domain (x, y, z)**	**— Dim geometric elt = 2**
	Edge border elements	**—>**	**2D Integration domain (x, y)**	**— Dim geometric elt = 1**

Type of FreeFem++ Meshes	1D, 2D, 3D	where dim domain dim geometric element	i.e. Rd $\neq$ RdHat

Adding in mesh3 class a new pointer on MeshS class

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

Remark: the numbering of DOFs is already done

Treatment for FreeFem++ 3D meshes

*data *.mesh*

```
MeshVersionFormatted 4

Dimension 3

Vertices
Num Vertices (xV)
.
.
Tetrahedra
Num Tetrahedrons (xTet)
.
.
Triangles
Num Triangles (xTri)
.
.
```

FreeFem++ v4

Mesh3: tetrahedrons elements and triangle borders

```
class Mesh3
Mesh3::Mesh3( ):GenericMesh<Tet,Triangle3,Vertex3>
        nv=xV, nTet=xTet, nTri=xTri
Mesh3::MeshS=NULL
```

FreeFem++'s kernel

src/femlib/mesh3.*pp

Treatment for FreeFem++ 3D meshes

old version surface
Mesh3: surface with only triangle borders but missing surface boundaries

*data *.mesh*

```
MeshVersionFormatted 4

Dimension 3

Vertices
Num Vertices (xV)
.
.
Triangles
Num Triangles (xTri)
.
.
```

```
class Mesh3
Mesh3 —> GenericMesh<Tet,Triangle3,Vertex3>
        nv=xV, nTet=0, nTri=xTri
Mesh3::MeshS=NULL
```

FreeFem++'s kernel

src/femlib/mesh3.*pp

Treatment for FreeFem++ 3D meshes

MeshS: surface triangle elements and edge borders

*data *.mesh*

```
MeshVersionFormatted 4

Dimension 3

Vertices
Num Vertices (xV)
.
.
Triangles
Num Triangles (xTri)
.
.
Edges
Num Segments (xSeg)
.
.
```

FreeFem++ v4

```
class Mesh3
Mesh3 —> GenericMesh<Tet,Triangle3,Vertex3>
        nv=0, nTet=0, nTri=0
Mesh3::MeshS —> GenericMesh<TriangleS, BoundaryEdgesS,Vertex3>
        nv=xV, nTri=xTri, nEdges=xSeg
```

FreeFem++'s kernel

src/femlib/mesh3.*pp

Treatment for FreeFem++ 3D meshes

*data *.mesh*

```
MeshVersionFormatted 4

Dimension 3

Vertices
Num Vertices (xV)
.
.
Tetrahedra
Num Tetrahedrons (xTet)
.
.
Triangles
Num Triangles (xTri)
.
.
Edges
Num Segments (xSeg)
.
.
```

FreeFem++ v4

Volume + Surface meshes

class Mesh3

Mesh3 —> GenericMesh<Tet,Triangle3,Vertex3>
nv=xV, nTet=xTet, nTri=xTri

Mesh3::MeshS —> GenericMesh<TriangleS, BoundaryEdgesS,Vertex3>
nv=xV_surf, nTri=xTri, nEdges=xSeg

FreeFem++'s kernel

src/femlib/mesh3.*pp

Treatment for FreeFem++ 3D meshes

NOW in FreeFem++ v4

— **readmesh** function

— **savemesh** function

— **movemesh3** function: apply a transformation on meshS

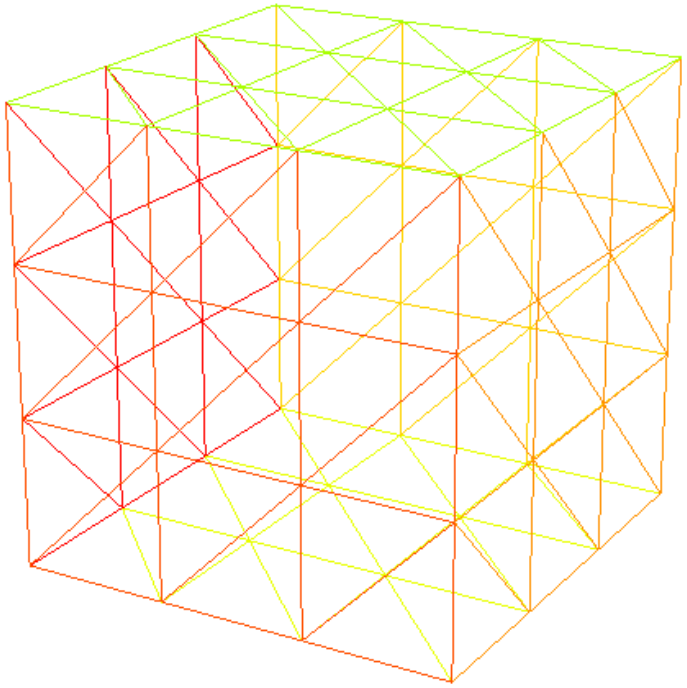
— **movemesh23** function: build a meshS from a 2D mesh

— **savesurface** on *.mesh (structured with tetrahedrons, triangles, edges)

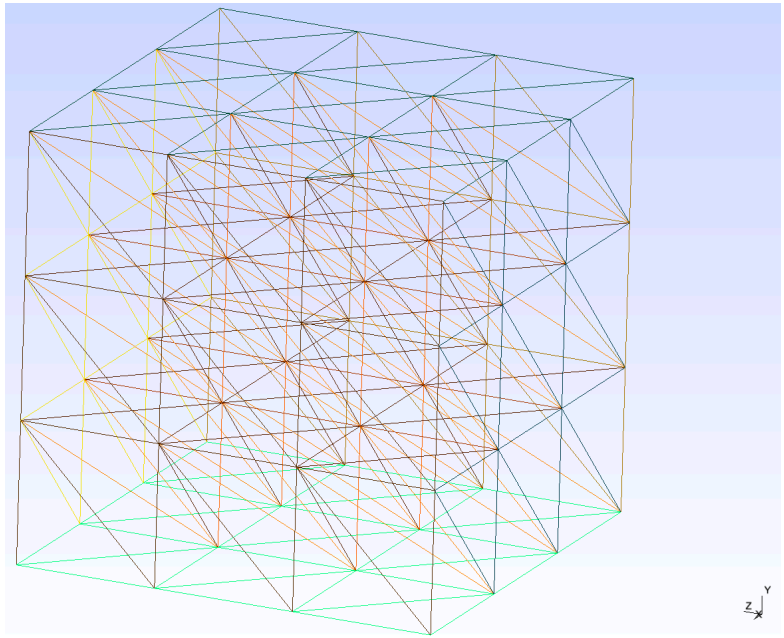
— **plot**(Th): plot a mesh3 or/and meshS

Treatment for FreeFem++ 3D meshes

```
Mesh3 Th3 = readmesh( " *.mesh " )
```



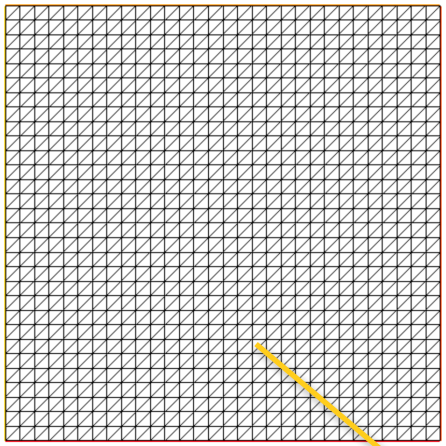
MeshS



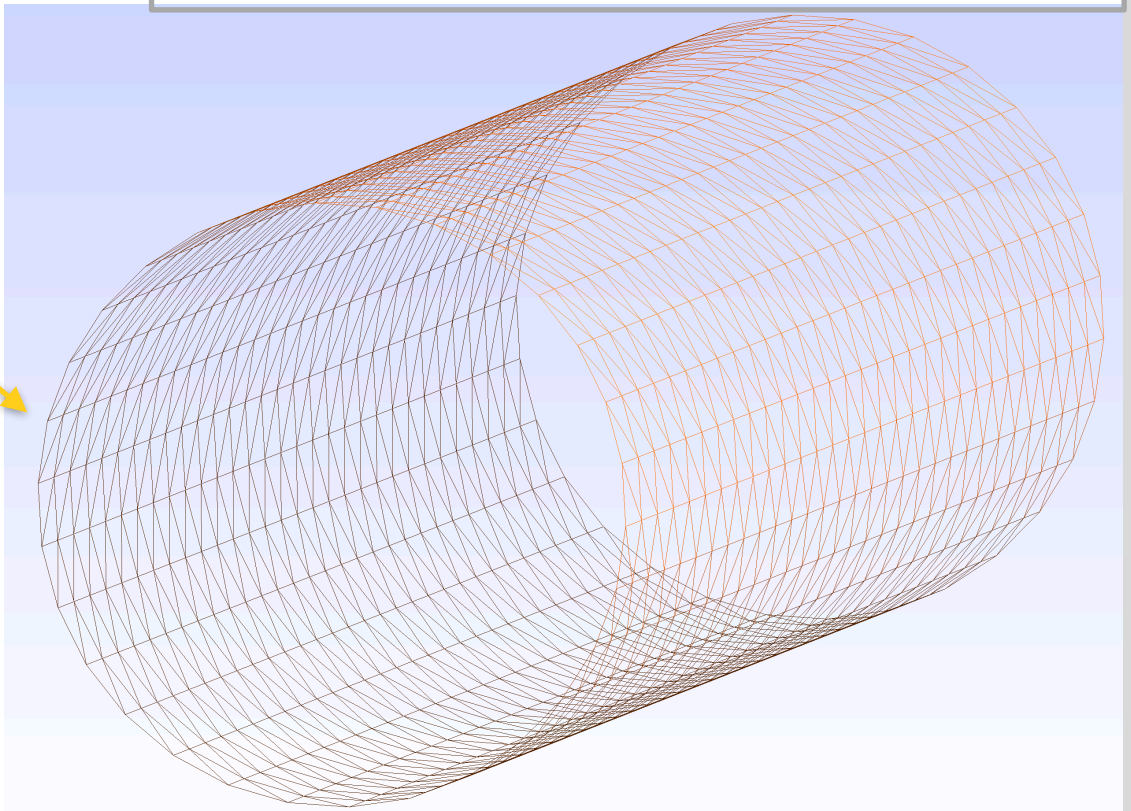
Mesh3

Input: — tetrahedrons + triangles + edges
— triangles +edges

Treatment for FreeFem++ 3D meshes



```
real x0=0.,x1=2.*pi;
real y0=0.,y1=2.*pi;
int n=30,m=30;
mesh Th=square(n,m,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
mesh3 ThS = movemesh23(Th, transfo=[cos(x),sin(x),y]);
```



Treatment for FreeFem++ 3D meshes

ONGOING in FreeFem++ v4

— Add generators of meshS (cube surface ….)

— code restructuration around to surface mesh (old surface Mesh3 and new MeshS)

— in class msh3, for MeshS:

- sum of meshS,
- constructor meshS,
- **buildlayers,**
- **adaptmesh**,
- **checkmesh,**
- **trunc**

Steps to solve the problem with FreeFem++?

Solve the weak form $\int_\Omega \nabla_{N-1} u . \nabla_{N-1} v = \int_\Omega f v, \forall v \in \Omega$

laplace-beltrami.edp

```
load "msh3"
load "medit"

mesh3 Th3dS = readmesh3("ThSurf.mesh");

fespace VhS(ThS,P1S);
VhS uS,vS;

macro Grad3(uSVar) [dx(uSVar),dy(uSVar),dz(uSVar)]  // EOM

problem Lap3dS(uSPb,vSPb , solver = CG )  = int2d(ThS)(Grad3(uSPb)'*Grad3(vSPb))
 - int2d ( ThS ) ( f * vSPb )
  + on(1,uSPb=g);

Lap3dS;

plot(Th, uS, wait=1);
```

3D Surface Finite Element Space

fespace Vh(< mesh > , < TypeofFE >)

meshS

- Build a new FE structure for Surface FE
- 3D surface Pk Lagrange with FreeFem++ FE definition
  P0S / P1S / P2S
  —> barycentric coordinates
  —> permutation of edges
  —> FE building derivate of base class TypeOfFE

New FE possible argument in FreeFem++'s language

FreeFem++'s kernel

src/fflib/lgfem.*pp

for the moment just scalar FE, Vh [u1,u2,u3] not possible

Steps to solve the problem with FreeFem++?

Solve the weak form $\int_\Omega \nabla_{N-1} u . \nabla_{N-1} v = \int_\Omega f v, \forall v \in \Omega$

laplace-beltrami.edp

```
load "msh3"
load "medit"

mesh3 Th3dS = readmesh3("ThSurf.mesh");

fespace VhS(ThS,P1S);
VhS uS,vS;

macro Grad3(uSVar) [dx(uSVar),dy(uSVar),dz(uSVar)]  // EOM

problem Lap3dS(uSPb,vSPb , solver = CG )  = int2d(ThS)(Grad3(uSPb)'*Grad3(vSPb))
 - int2d ( ThS ) ( f * vSPb )
  + on(1,uSPb=g);

Lap3dS;

plot(Th, uS, wait=1);
```

need to redefine a new identifiant for P1 because ThS is a mesh3 and not a meshS due to the coding structure

3D Surface Pk Lagrange

Purpose: Consider the 2D Pk Lagrange to define the 3D Surface Pk Lagrange FE

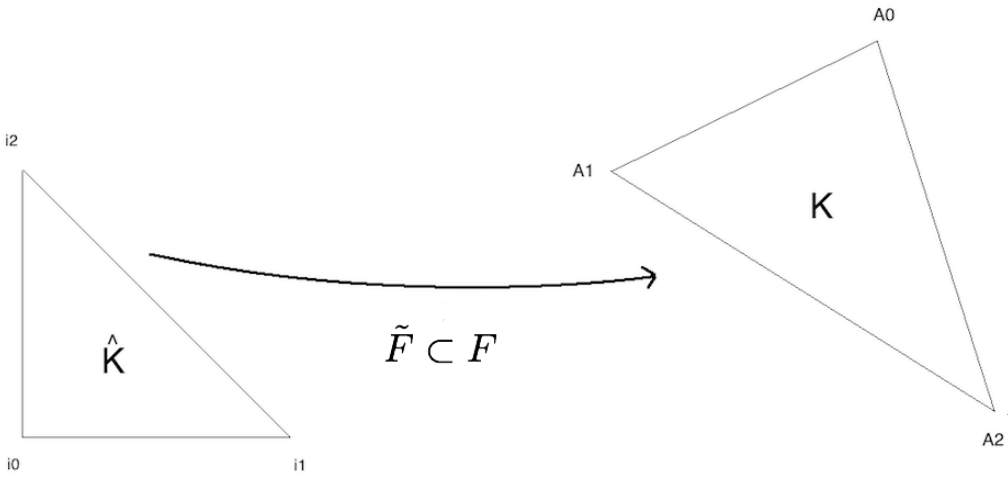
How? Find a geometric transformation allows to switch from the reference FE to a curent FE

Let $\hat{x}$ be a point of the 2D reference triangle $\hat{K} \subset \mathbb{R}^2$
and X a point of a 3D triangle $K \subset \mathbb{R}^3$

$$F : \mathbb{R}^2 \to \mathbb{R}^3$$
$$\hat{x} \to X$$

~~bijective application~~ *solution*:

$$\tilde{F}, \tilde{F} \subset F$$
$$\tilde{F} : \mathbb{R}^2 \to \mathbb{R}^3$$
$$\hat{x} \to X$$
$$\begin{pmatrix} x \\ y \\ 0 \end{pmatrix} \to \begin{pmatrix} \overrightarrow{A_0A_1} \\ \overrightarrow{A_0A_2} \\ \overrightarrow{A_0A_1} \wedge \overrightarrow{A_0A_2} \end{pmatrix} (\hat{x} - A_0)$$



$\wedge$ the usual product vectorial

$\overrightarrow{A0A1} \wedge \overrightarrow{A_0A_2} = \vec{n} = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$ a normal vector

3D Surface P1 Lagrange

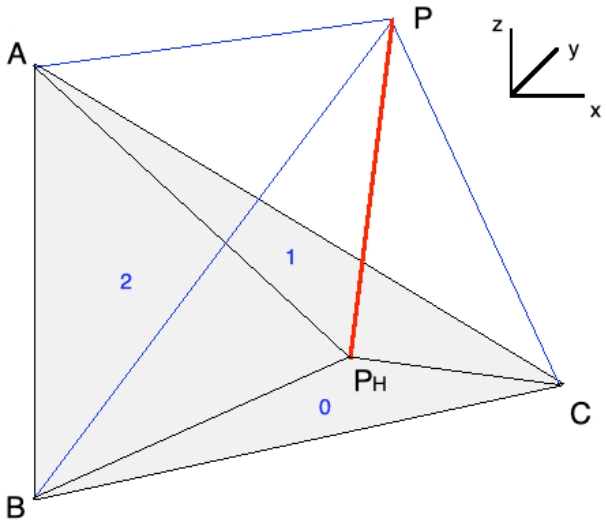
Method to determine the basis functions

Properties: 1/. Search the basis functions is equivalent to search the barycentric coordinates to a point X

2/. Barycentric coordinates are equivalent to the ratio of vectorial area in the triangle K

3D Surface P1 Lagrange

Method to determine the 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 $\mathbb{R}^3$

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

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

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

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

2D generalization

Basis functions

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

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

and the 1st derivative

$$\Delta_P \lambda_i(P) = \frac{(\vec{N} \wedge \vec{E}_i)}{(\vec{N}, \vec{N})}$$

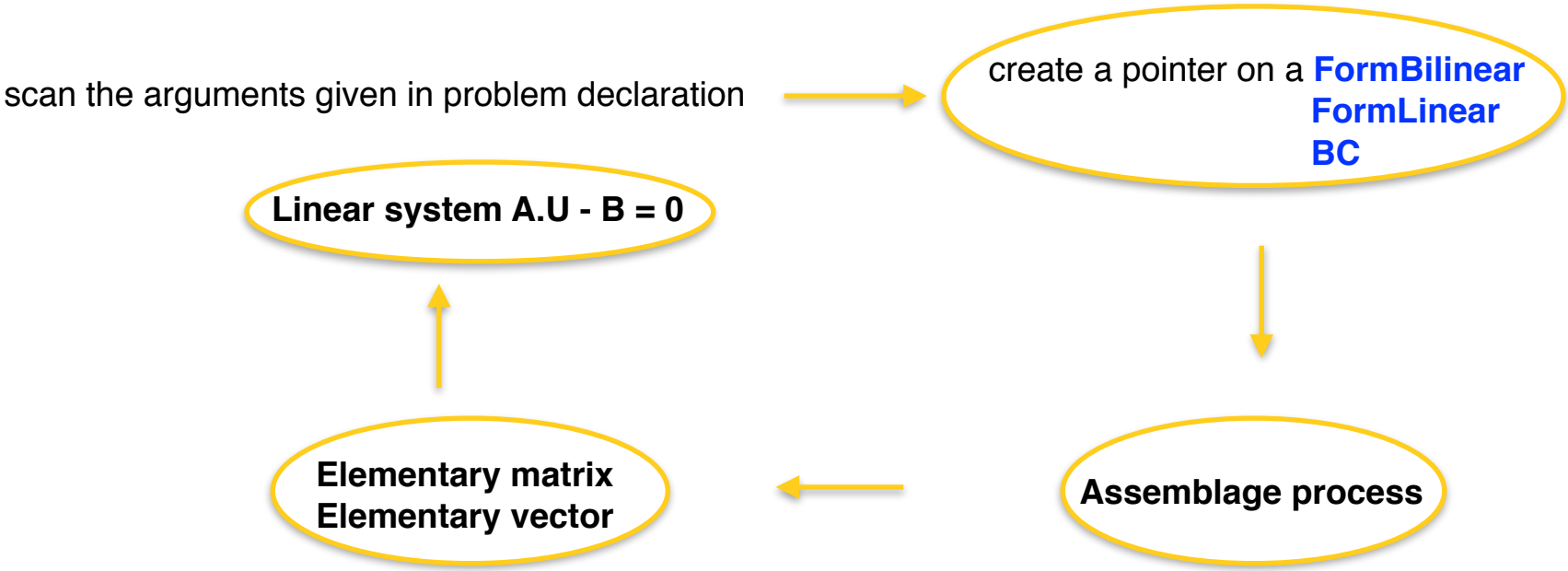
Steps to solve the problem with FreeFem++?

Solve the weak form $\int_\Omega \nabla_{N-1} u . \nabla_{N-1} v = \int_\Omega f v, \forall v \in \Omega$

laplace-beltrami.edp

```
load "msh3"
load "medit"
mesh3 Th3dS = readmesh3("ThSurf.mesh");

fespace VhS(ThS,P1S);
VhS uS,vS;

macro Grad3(uSVar) [dx(uSVar),dy(uSVar),dz(uSVar)]  // EOM

problem Lap3dS(uSPb,vSPb , solver = CG )  = int2d(ThS)(Grad3(uSPb)'*Grad3(vSPb))
 - int2d ( ThS ) ( f * vSPb )
  + on(1,uSPb=g);

Lap3dS;

plot(Th, uS, wait=1);
```

Build a FreeFem++ Surface FE problem



FreeFem++'kernel functions:

```
template<class R,class FESpace,class v_fes>
```

```
template<class R,typename MC,class FESpace >  bool AssembleVarForm
template<class R,class FESpace>  void AssembleBC
```

Build a FreeFem++ Surface FE problem

create an instantiation for FESpaceS < MeshS, TypeOfFES > for the functions which build:

- the global matrix A
- the RHS vector b
- to apply boundary conditions

```
template<class R>
void AssembleBilinearForm(Stack stack,const MeshS & Th,const FESpaceS & Uh,
  const FESpaceS & Vh, bool sym,MatriceCreuse<R>  & A, const  FormBilinear * b)
```

```
template<class R>
void AssembleLinearForm(Stack stack,const MeshS & Th,const FESpaceS & Vh,
                        KN_<R> * B,const  FormLinear * l )
```

```
template
void AssembleBC<double,FESpaceS>(Stack stack,const MeshS & Th,
   const FESpaceS & Uh, const FESpaceS & Vh,bool sym,MatriceCreuse<double> * A,
   KN_<double> * B,KN_<double> * X,const list<C_F0> &largs , double tgv  );
```

Steps to solve the problem with FreeFem++?

Solve the weak form $\int_\Omega \nabla_{N-1} u . \nabla_{N-1} v = \int_\Omega f v, \forall v \in \Omega$

laplace-beltrami.edp

```
load "msh3"
load "medit"
mesh3 Th3dS = readmesh3("ThSurf.mesh");

fespace VhS(ThS,P1S);
VhS uS,vS;

macro Grad3(uSVar) [dx(uSVar),dy(uSVar),dz(uSVar)]  // EOM

problem Lap3dS(uSPb,vSPb , solver = CG )  = int2d(ThS)(Grad3(uSPb)'*Grad3(vSPb))
 - int2d ( ThS ) ( f * vSPb )
  + on(1,uSPb=g);

Lap3dS;

plot(Th, uS, wait=1);
```

Visualization tool: ffglut

FreeFem++'s kernel

fflib/lgfem.*pp

Graphic/ffglut.*pp

- Plot a meshS with ff-glut

  plot (Th);

- Visualize the solution with iso-values and filling values for a Surface FE solution

  plot (Uh);

server/client model

server: fflib.*pp —> refer to the Plot operator in FreeFem++'s language
Scan the arguments given by User

FreeFem's kernel corresponding function to
- determine the type of values
- send in special file the information that must be plot with ff-glut graphic tools

client: ffglut.*pp —> receive and read the special file and plot the objects: mesh, solution, curve, …..

Futur work:

- Finalize the Surface FEM

- Interface with Xavier Clayes' BemTools library (new Variational Form with double int)

- Documentation

Thank you for your attention!