Treatment of contact between finite deformable bodies

using FreeFem++

Pantz Olivier

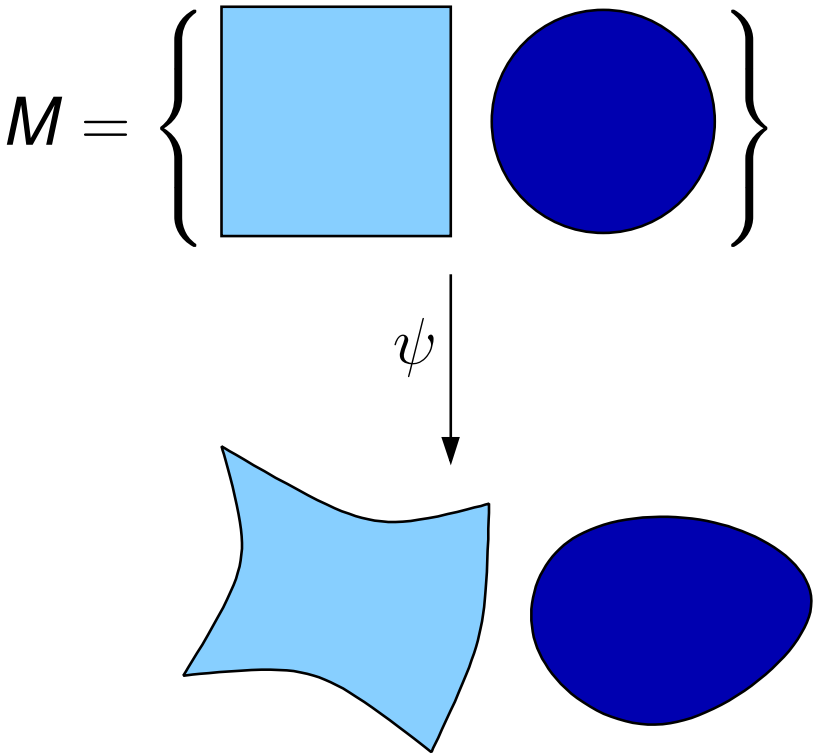
CMAP, Ecole Polytechnique

Decembre 2011

Plan

Setting of the problem

Let M be a set of deformable bodies and J be a functional that maps every deformation ψ of M to its enery $J(\psi)$.



Goal

Find $\varphi \in \mathcal{A}$ such that

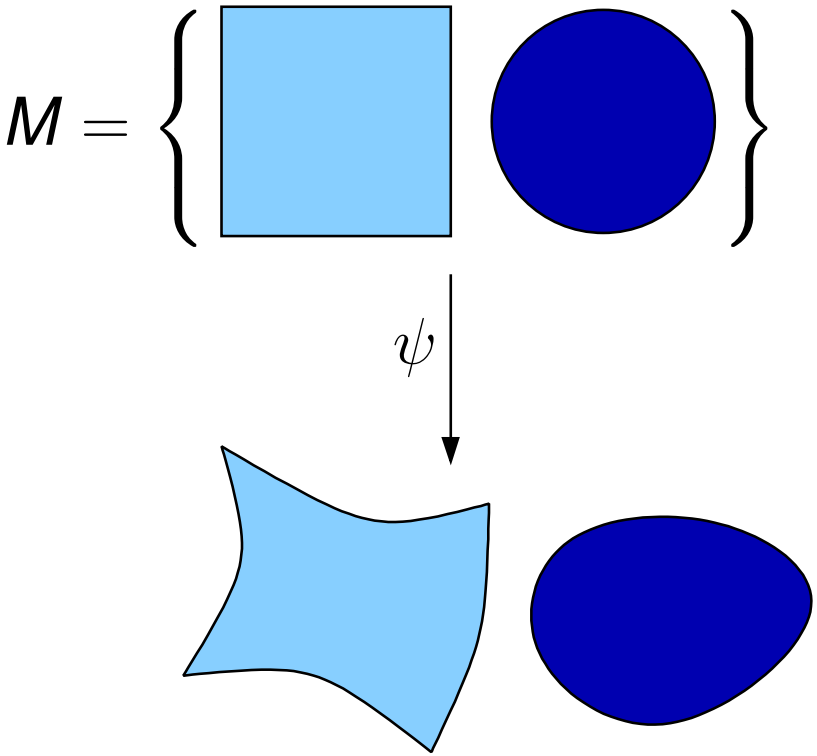
$$J(\varphi) = \min_{\psi \in \mathcal{A}} J(\psi)$$

$\mathcal{A}$: Set of admissible deformations (without (self)intersections)

How couls we define $\mathcal{A}$?

Setting of the problem

Let M be a set of deformable bodies and J be a functional that maps every deformation ψ of M to its enery $J(\psi)$.



Goal

Find $\varphi \in \mathcal{A}$ such that

$$J(\varphi) = \min_{\psi \in \mathcal{A}} J(\psi)$$

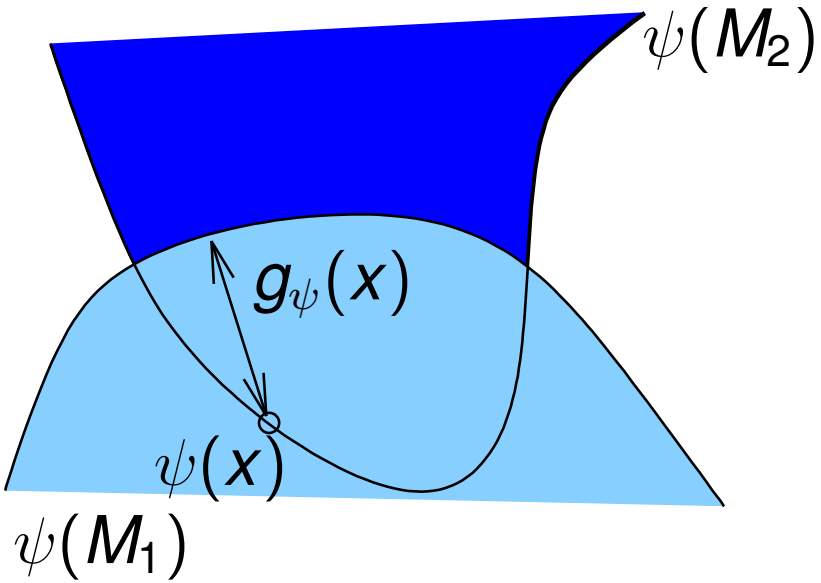
$\mathcal{A}$: Set of admissible deformations (without (self)intersections)

How couls we define $\mathcal{A}$?

The Master/Slave approach

Let us assume that $M = \{M_1, M_2\}$ and $\dim(M_1) = \dim(M_2) = n$.
For all $x \in \partial M_2$, we set

$$g_\psi(x) = \begin{cases} \operatorname{dist}(\psi(x), \psi(\partial M_1)) & \text{si } \psi(x) \notin \psi(M_1) \\ -\operatorname{dist}(\psi(x), \psi(\partial M_1)) & \text{si } \psi(x) \in \psi(M_1) \end{cases}$$



Definition of the constraints

$g_\psi(x) \geq 0$ for every $x \in \partial M_2$.
M_1 =Master body.
M_2 =Slave body.

Problems linked with the Master/Slave approach

1. Some deformation that satisfies the constraints are NOT intersection free.
2. All deformations are admissible in the case of
   - thin structures ($\dim(M) < n$),
   - self-contacts.
3. The map g is not derivable, leading to technical difficulties (like the “chatter” problem).

Problems linked with the Master/Slave approach

1. Some deformation that satisfies the constraints are NOT intersection free.
2. All deformations are admissible in the case of
   - thin structures ($\dim(M) < n$),
   - self-contacts.
3. The map g is not derivable, leading to technical difficulties (like the “chatter” problem).

Problems linked with the Master/Slave approach

1. Some deformation that satisfies the constraints are NOT intersection free.
2. All deformations are admissible in the case of
   - thin structures ($\dim(M) < n$),
   - self-contacts.
3. The map g is not derivable, leading to technical difficulties (like the “chatter” problem).

An alternative formulation

We define $\mathcal{A}$ as the closure of the embeddings.

$$\mathcal{A} = \overline{\mathrm{Emb}(\mathrm{M}; \mathbb{R}^{\mathrm{n}})}$$

where $\mathrm{Emb}(\mathrm{M}; \mathbb{R}^{\mathrm{n}})$ is the set of embeddings if M into $\mathbb{R}^n$.

Pros

- Could be applied whatever the dimension(s) of M is/are.
- Take into account contacts and self-contacts in a single setting.

Cons

- Implicit definition of $\mathcal{A}$
- Optimality Conditions ?

An alternative formulation

We define $\mathcal{A}$ as the closure of the embeddings.

$$\mathcal{A} = \overline{\mathrm{Emb}(\mathrm{M}; \mathbb{R}^{\mathrm{n}})}$$

where $\mathrm{Emb}(\mathrm{M}; \mathbb{R}^{\mathrm{n}})$ is the set of embeddings if M into $\mathbb{R}^n$.

Pros

- Could be applied whatever the dimension(s) of M is/are.
- Take into account contacts and self-contacts in a single setting.

Cons

- Implicit definition of $\mathcal{A}$
- Optimality Conditions ?

An alternative formulation

We define $\mathcal{A}$ as the closure of the embeddings.

$$\mathcal{A} = \overline{\mathrm{Emb}(\mathrm{M}; \mathbb{R}^{\mathrm{n}})}$$

where $\mathrm{Emb}(\mathrm{M}; \mathbb{R}^{\mathrm{n}})$ is the set of embeddings if M into $\mathbb{R}^n$.

Pros

- Could be applied whatever the dimension(s) of M is/are.
- Take into account contacts and self-contacts in a single setting.

Cons

- Implicit definition of $\mathcal{A}$
- Optimality Conditions ?

Discretization

Admissible set of deformations $\mathcal{A}_{h,\varepsilon}$

$\mathcal{A}_{h,\varepsilon}$ is the set of admissible deformations ψ finite elements P_1 such that

$$\text{dist}(\psi(a), \psi(b)) \geq \varepsilon,$$

for all elements a and b of $\mathcal{T}_h$, mesh of M, such that $a \cap b = \emptyset$.

Discretized Problem

Find $\varphi_{h,\varepsilon} \in \mathcal{A}_{h,\varepsilon}$

$$J(\varphi_{h,\varepsilon}) = \min_{\psi \in \mathcal{A}_{h,\varepsilon}} J(\psi).$$

- Explicit of the admissible set.
- Optimality conditions easy to derive.
- Convergence of $\varphi_{h,\varepsilon}$ towards φ.

Discretization

Admissible set of deformations $\mathcal{A}_{h,\varepsilon}$

$\mathcal{A}_{h,\varepsilon}$ is the set of admissible deformations ψ finite elements P_1 such that

$$\text{dist}(\psi(a), \psi(b)) \geq \varepsilon,$$

for all elements a and b of $\mathcal{T}_h$, mesh of M, such that $a \cap b = \emptyset$.

Discretized Problem

Find $\varphi_{h,\varepsilon} \in \mathcal{A}_{h,\varepsilon}$

$$J(\varphi_{h,\varepsilon}) = \min_{\psi \in \mathcal{A}_{h,\varepsilon}} J(\psi).$$

- Explicit of the admissible set.
- Optimality conditions easy to derive.
- Convergence of $\varphi_{h,\varepsilon}$ towards φ.

Discretization

Admissible set of deformations $\mathcal{A}_{h,\varepsilon}$

$\mathcal{A}_{h,\varepsilon}$ is the set of admissible deformations ψ finite elements P_1 such that

$$\text{dist}(\psi(a), \psi(b)) \geq \varepsilon,$$

for all elements a and b of $\mathcal{T}_h$, mesh of M, such that $a \cap b = \emptyset$.

Discretized Problem

Find $\varphi_{h,\varepsilon} \in \mathcal{A}_{h,\varepsilon}$

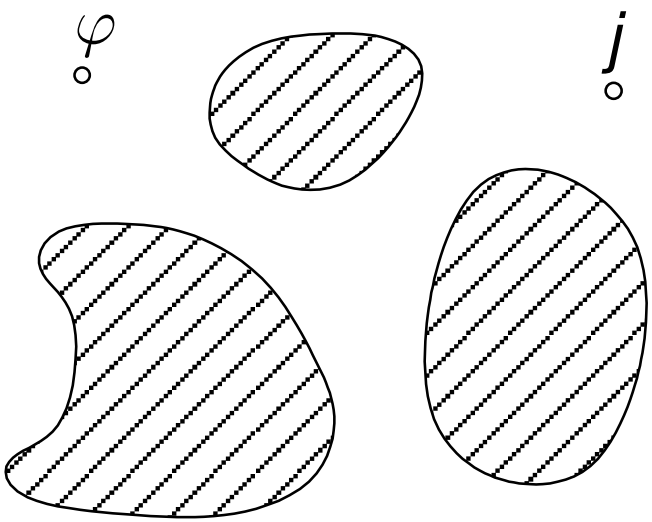
$$J(\varphi_{h,\varepsilon}) = \min_{\psi \in \mathcal{A}_{h,\varepsilon}} J(\psi).$$

- Explicit of the admissible set.
- Optimality conditions easy to derive.
- Convergence of $\varphi_{h,\varepsilon}$ towards φ.

Treatment of the non convexes constraints!

Definition

$T(\psi) =$ Convex close "Neighborhood" of ψ included in $\mathcal{A}_{h,\varepsilon}$.



Algorithm

1. Initialization: $\varphi_0 = j$.
2. For every $n \geq 0$,

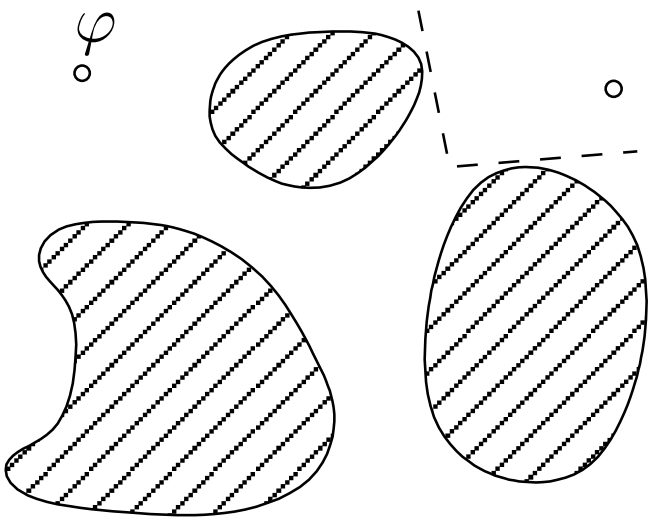
$$J(\varphi_{n+1}) := \min_{\psi \in T(\varphi_n)} J(\psi).$$

3. STOP when $J(\varphi_{n+1}) \simeq J(\varphi_n)$.

Treatment of the non convexes constraints!

Definition

$T(\psi) =$ Convex close "Neighborhood" of ψ included in $\mathcal{A}_{h,\varepsilon}$.



Algorithm

1. Initialization: $\varphi_0 = j$.
2. For every $n \geq 0$,

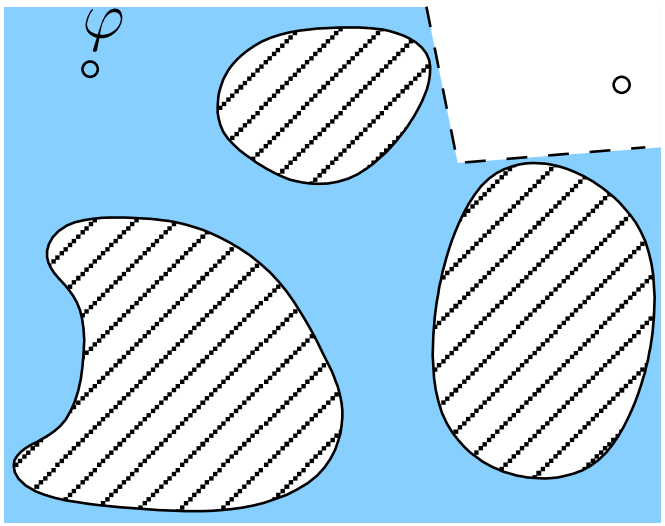
$$J(\varphi_{n+1}) := \min_{\psi \in T(\varphi_n)} J(\psi).$$

3. STOP when $J(\varphi_{n+1}) \simeq J(\varphi_n)$.

Treatment of the non convexes constraints!

Definition

$T(\psi) =$ Convex close "Neighborhood" of ψ included in $\mathcal{A}_{h,\varepsilon}$.



Algorithm

1. Initialization: $\varphi_0 = j$.
2. For every $n \geq 0$,

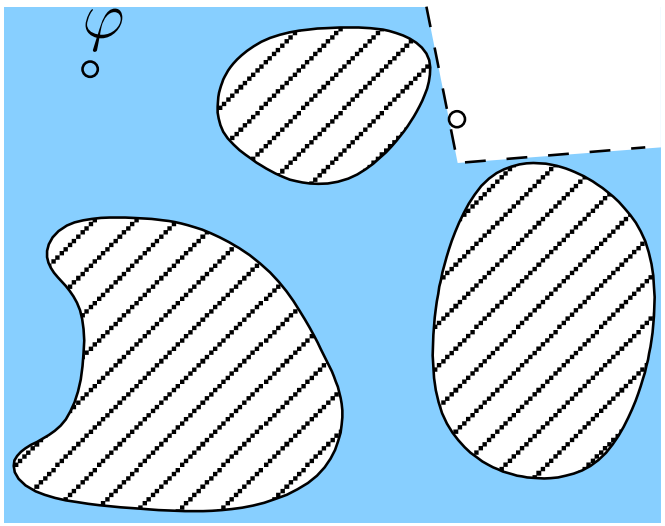
$$J(\varphi_{n+1}) := \min_{\psi \in T(\varphi_n)} J(\psi).$$

3. STOP when $J(\varphi_{n+1}) \simeq J(\varphi_n)$.

Treatment of the non convexes constraints!

Definition

$T(\psi) =$ Convex close "Neighborhood" of ψ included in $\mathcal{A}_{h,\varepsilon}$.



Algorithm

1. Initialization: $\varphi_0 = j$.
2. For every $n \geq 0$,

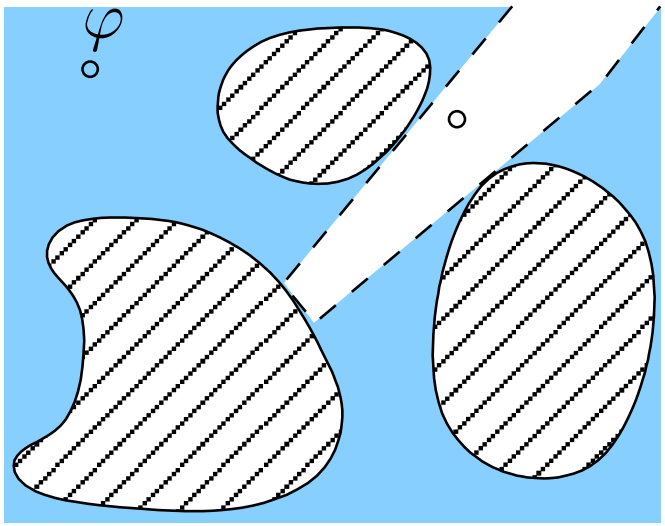
$$J(\varphi_{n+1}) := \min_{\psi \in T(\varphi_n)} J(\psi).$$

3. STOP when $J(\varphi_{n+1}) \simeq J(\varphi_n)$.

Treatment of the non convexes constraints!

Definition

$T(\psi) =$ Convex close "Neighborhood" of ψ included in $\mathcal{A}_{h,\varepsilon}$.



Algorithm

1. Initialization: $\varphi_0 = j$.
2. For every $n \geq 0$,

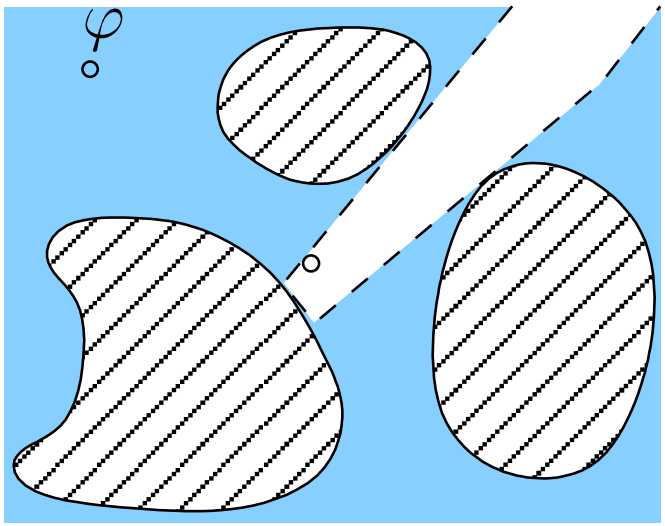
$$J(\varphi_{n+1}) := \min_{\psi \in T(\varphi_n)} J(\psi).$$

3. STOP when $J(\varphi_{n+1}) \simeq J(\varphi_n)$.

Treatment of the non convexes constraints!

Definition

$T(\psi) =$ Convex close "Neighborhood" of ψ included in $\mathcal{A}_{h,\varepsilon}$.



Algorithm

1. Initialization: $\varphi_0 = j$.
2. For every $n \geq 0$,

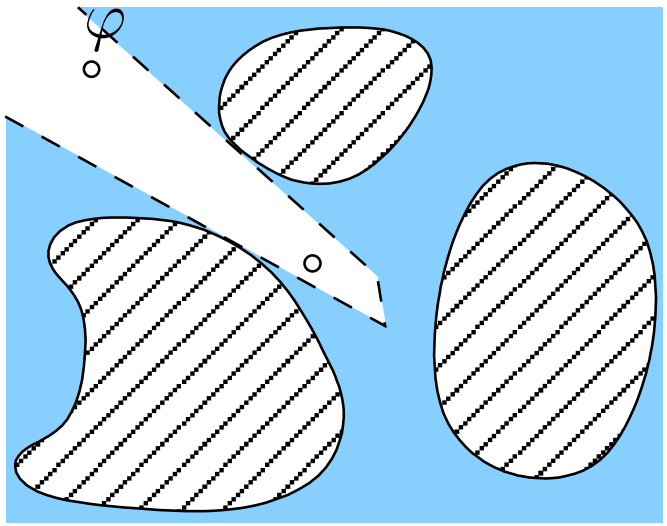
$$J(\varphi_{n+1}) := \min_{\psi \in T(\varphi_n)} J(\psi).$$

3. STOP when $J(\varphi_{n+1}) \simeq J(\varphi_n)$.

Treatment of the non convexes constraints!

Definition

$T(\psi) =$ Convex close "Neighborhood" of ψ included in $\mathcal{A}_{h,\varepsilon}$.



Algorithm

1. Initialization: $\varphi_0 = j$.
2. For every $n \geq 0$,

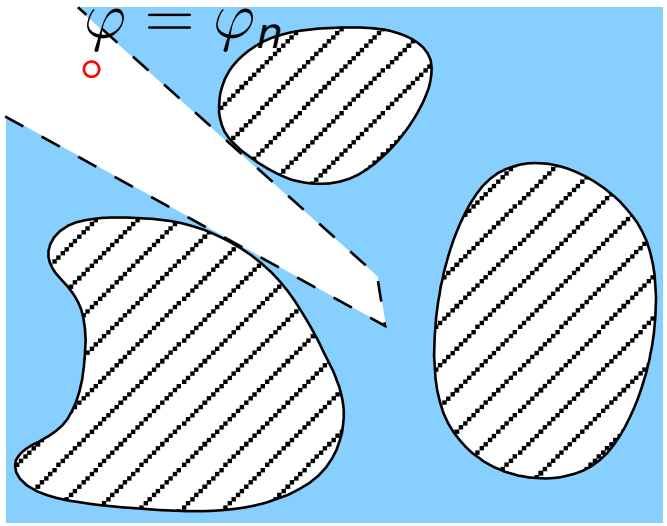
$$J(\varphi_{n+1}) := \min_{\psi \in T(\varphi_n)} J(\psi).$$

3. STOP when $J(\varphi_{n+1}) \simeq J(\varphi_n)$.

Treatment of the non convexes constraints!

Definition

$T(\psi) =$ Convex close "Neighborhood" of ψ included in $\mathcal{A}_{h,\varepsilon}$.



Algorithm

1. Initialization: $\varphi_0 = j$.
2. For every $n \geq 0$,

$$J(\varphi_{n+1}) := \min_{\psi \in T(\varphi_n)} J(\psi).$$

3. STOP when $J(\varphi_{n+1}) \simeq J(\varphi_n)$.

Choosing $T(\psi)$

An alternative

Can we define $T(\psi)$ by a simple linearization of the constraints $\text{dist}(\psi(a), \psi(b)) \geq \varepsilon$ for all a and b of $\mathcal{T}_h$ such that $a \cap b = \emptyset$?

$T(\psi)$ is not included in $\mathcal{A}_{h,\varepsilon}$!

An admissible choice (in dimension $n = 2$)

$$T(\psi) = \{\varphi \in X_h \text{ such that for every } i, j \text{ such that } x_i \notin a_j, \\ (\varphi(a_j) - \varphi(x_i)) \cdot n_{ij} \geq \varepsilon\}$$

X_h =set of P_1 finite elements,
x_i = vertices of the mesh,
a_j = edges of the mesh,
$n_{ij}(\psi)$ such that $\|n_{ij}(\psi)\| = 1$
and

$$\min(\psi(y) - \psi(x_i)) \cdot n_{ij}(\psi) = \text{dist}(\psi(x_i), \psi(a_j))$$

Choosing $T(\psi)$

An alternativeNOT WORKING

Can we define $T(\psi)$ by a simple linearization of the constraints $\text{dist}(\psi(a), \psi(b)) \geq \varepsilon$ for all a and b of $\mathcal{T}_h$ such that $a \cap b = \emptyset$?

$T(\psi)$ is not included in $\mathcal{A}_{h,\varepsilon}$!

An admissible choice (in dimension $n = 2$)

$$T(\psi) = \{\varphi \in X_h \text{ such that for every } i, j \text{ such that } x_i \notin a_j,$$
$$(\varphi(a_j) - \varphi(x_i)) \cdot n_{ij} \geq \varepsilon\}$$

X_h =set of P_1 finite elements,
x_i = vertices of the mesh,
a_j = edges of the mesh,
$n_{ij}(\psi)$ such that $\|n_{ij}(\psi)\| = 1$
and

$$\min(\psi(y) - \psi(x_i)) \cdot n_{ij}(\psi) = \text{dist}(\psi(x_i), \psi(a_j))$$

Choosing $T(\psi)$

An alternativeNOT WORKING

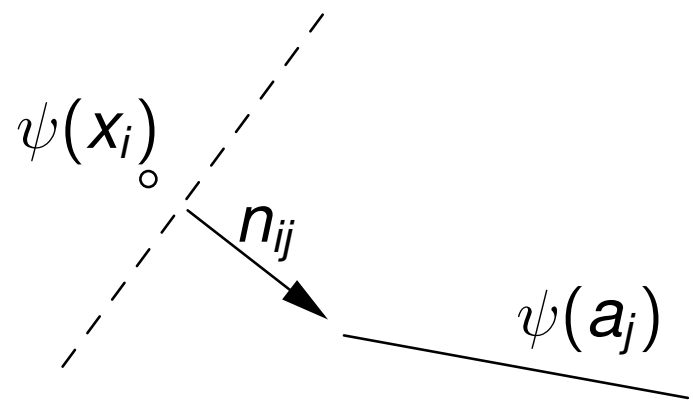
Can we define $T(\psi)$ by a simple linearization of the constraints $\text{dist}(\psi(a), \psi(b)) \geq \varepsilon$ for all a and b of $\mathcal{T}_h$ such that $a \cap b = \emptyset$?

$T(\psi)$ is not included in $\mathcal{A}_{h,\varepsilon}$!

An admissible choice (in dimension $n = 2$)

$$T(\psi) = \{\varphi \in X_h \text{ such that for every } i, j \text{ such that } x_i \notin a_j,$$
$$(\varphi(a_j) - \varphi(x_i)) \cdot n_{ij} \geq \varepsilon\}$$

X_h =set of P_1 finite elements,
x_i = vertices of the mesh,
a_j = edges of the mesh,
$n_{ij}(\psi)$ such that $\|n_{ij}(\psi)\| = 1$
and



$$\min(\psi(y) - \psi(x_i)) \cdot n_{ij}(\psi) = \text{dist}(\psi(x_i), \psi(a_j))$$

Optimality Conditions

Question / Answer

Are fix points of the algorithm local minimizers of J over $\mathcal{A}_{h,\varepsilon}$?

NO !

But, they satisfy the optimality conditions up to the order h(Good enough).

Optimality Conditions

Question / Answer

Are fix points of the algorithm local minimizers of J over $\mathcal{A}_{h,\varepsilon}$?
NO !

But, they satisfy the optimality conditions up to the order h(Good enough).

Optimality Conditions

Question / Answer

Are fix points of the algorithm local minimizers of J over $\mathcal{A}_{h,\varepsilon}$?
NO !
But, they satisfy the optimality conditions up to the order h(Good enough).

Abstract

1. Initialization: $\varphi_0 = j$.
2. For all $n \geq 0$,
$$J(\varphi_{n+1}) = \min_{\psi \in T(\varphi_n)} J(\psi).$$
3. STOP when $J(\varphi_{n+1}) \simeq J(\varphi_n)$.

$$T(\psi) = \{\varphi \in X_h \text{ such that for every } i,j \text{ such that } x_i \notin a_j,$$
$$F_{ij}^0(\psi) \leq 0 \text{ and } F_{ij}^1(\psi) \leq 0\}$$

$x_i =$ vertices of the mesh, $a_j =$ edges of the mesh, $\|n_{ij}(\psi)\| = 1$

$$\min_{y \in a_j} (\psi(y) - \psi(x_i)) \cdot n_{ij}(\psi) = \text{dist}(\psi(x_i), \psi(a_j))$$

and

$$F_{ij}^{\alpha}(\psi) = \varepsilon - (\psi(a_j^{\alpha}) - \psi(x_i)) \cdot n_{ij}(\psi),$$

where a_j^{α} $(\alpha = 0, 1)$ are the nodes of the edge a_j.

Numerical Applications

- Application 1 - Contacts between two linear elastic bodies (Blocks.swf).
- Application 2 - Self-Contacts for a non linear elastic beam (Poutre2.swf).
- Application 3 - Balloons in a rotating box (Dynamic.swf en c++).

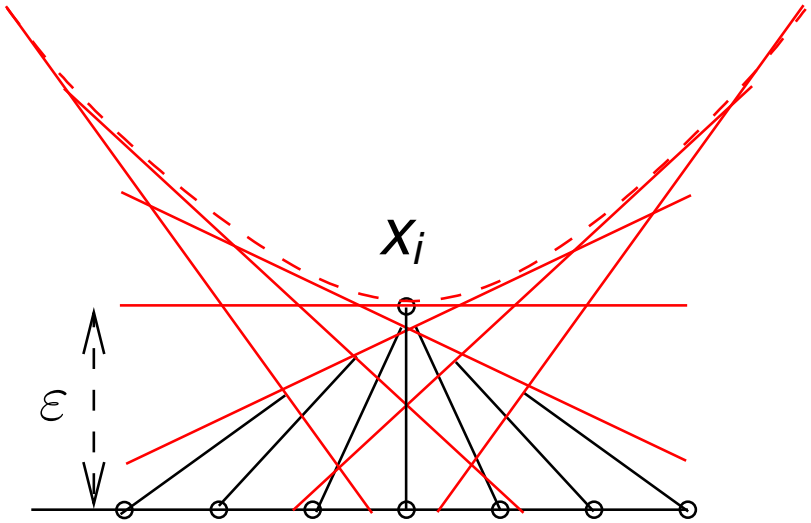
Time of Computation

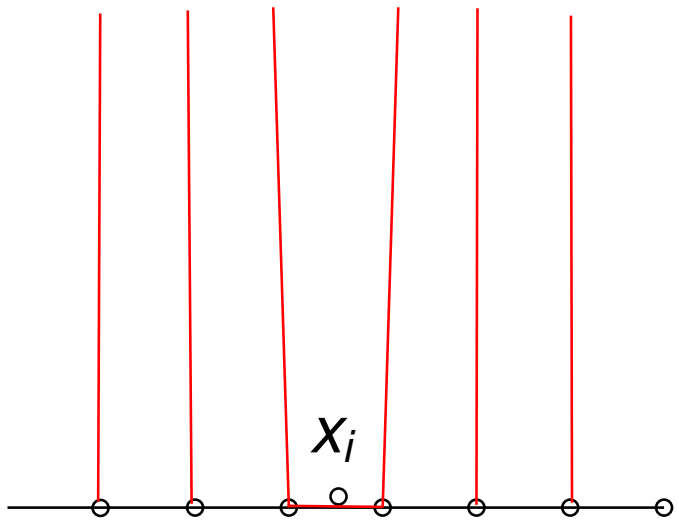
Number of constraints

are of the order N_{dof}^2 where (N_{dof}=number of degrees of freedoms)

Number of iterations

The size of a convex neighborhood $T(\psi)$ previously defined is approximately $\max(h, \varepsilon)$.
The number of iteration of the algorithm is of order $\max(h, \varepsilon)^{-1}$.



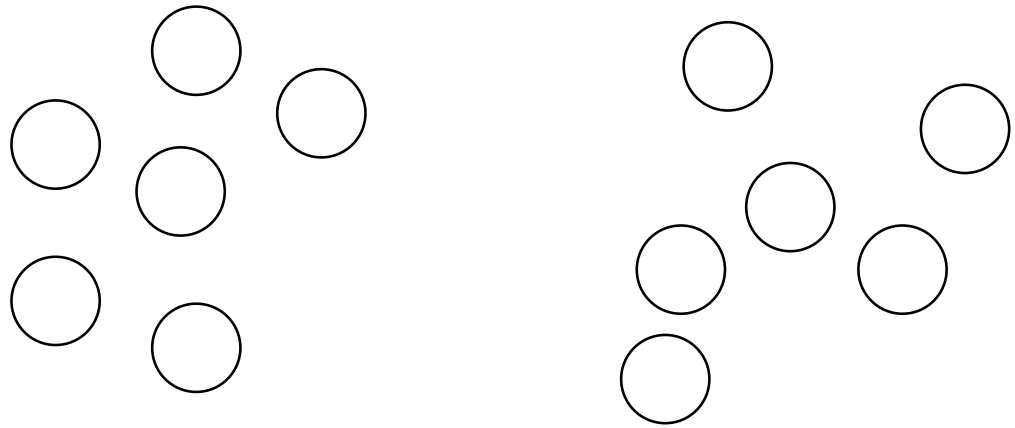


Uzawa not exact...

Uzawa does not lead to a solution that satisfies exactly the constraints. Moreover, the more ε is small, the more is the

Less constraints !

How to ensure non overlapping of the balls located on the left with the one located on the right ?



The first method consists to impose for each couple of balls to be locate at each side of a plane. It leads to N_{dof}^2 constraints.
The second method of treatment by ”bundles”, that is to impose that the left balls are located on one side of a plane and the right balls located on the other. Only N_{dof} constraints.
Proceeding recursively, we can ensure non-intersection between the balls with $N_{dof} \log(N_{dof})$ constraints.

Internal Approximation II

1. Initialization: $\varphi_0 = j$.
2. For all $k \geq 0$
   - Partition $\mathcal{P}_k$ of $M \times M$ such that there exists n_k such that for every $(x_i, x_j) \in \mathcal{P}_k$

$$n_k.(\varphi_k(x_i) - \varphi_k(x_j)) > \varepsilon. \tag{1}$$

     (and $\mathcal{P}_k$ is a "block")
   - φ_{k+1} obtained after the minimization of J by a sequence of neighborhoods of the form (1) where $\mathcal{P}_k$ is fixed.
3. STOP as soon as $J(\varphi_{k+1}) \simeq J(\varphi_k)$

What you don't want to see...

```
// Uzawa step
real epsilon=0.001;                    // Minimal distances
real tol=0.1;                          // tolerance sur la condition de contact (entre 0+ et 1)
real alpha=0.001;                              // Uzawa step
//real alpha=0.5;                              // Uzawa step
real deltamin=1.e-4*alpha;       // Critere d'arret Uzawa
real err=0.001;

int NbAdapt=1;                         // Nombre d'adaptation de maillage (Attention a l'adaptation de maillage qui doit etre compatible av
// dans ce cas de ne pas considerer les autocontacts... a faire)

// Modules de Lame

real E=1.;                          //Module de Young (toujours positif)
//real E=5.;                        //Module de Young (toujours positif)
real nu=0.35;
real lambda=E*nu/((1.+nu)*(1.-2.*nu));// coef de Lame
real mu=E/(2.*(1.+nu));


// Definition of the mesh
real Lx=0.1;
real Ly=5.;

int Neumann=1;
int Dirichlet=2;

macro boundary
1,2,3,4//


// Definition du maillage
mesh Sh,Th;
{
        // Le premier solide
        border bord(t=0,Lx){x=t;y=0.;label=Dirichlet;};
        border bord1(t=0,Ly){x=Lx;y=t;label=Neumann;};
        border bord2(t=Lx,0){x=t;y=Ly;label=Dirichlet;};
        border bord3(t=Ly,0){x=0;y=t;label=Neumann;};
        plot(bord(5),bord1(5),bord2(5),bord3(5),wait=1);


        real dn=40.;// nb de points par unite de longueur
        Sh=buildmesh( bord(Lx*dn)+bord1(Ly*dn)+bord2(Lx*dn)+bord3(Ly*dn),fixeborder=1);
        Th=Sh;
}

// Nombre de regions

        fespace Vh0(Sh,P0);
```

```
        Vh0 reg=region;
        int regmax=floor(reg[].max+0.5);
        int[int] Active(regmax+1);
        Active=0;
        for (int i=0;i<Vh0.ndof;i++) Active[floor(reg[][i]+0.5)]=1;
        int nbreg=Active.sum;
        cout<<"Nb de regions="<<nbreg<<endl;
        int[int] labelreg(nbreg);
        int k=0;
        for(int i=0;i<regmax+1;i++) if (Active[i]==1) {labelreg[k]=i;k++;}
        cout<<"Les regions sont "<<labelreg<<endl;

plot(Sh,wait=1);

// Definition des forces appliquees
fespace Vh1(Sh,P1);
Vh1 f1,f2;
{
        f1=0;
        f2=0.;//-0.1*(reg==0);
//      plot([f1,f2],wait=1);
}

// Definition du probleme d'elasticite
int initialize;
//Vh1 u1,u2,v1,v2;
Vh1 Fc1,Fc2;    // Forces de contact

// Contact forces on the deformed mesh
fespace Wh(Th,P1);
Wh Fcdef1,Fcdef2;
// Construction de la matrice de Masse sur le maillage deforme
varf Masse2(u,v)=int1d(Th,qfe=qf1pElump)(u*v);


// La deformations initiales
Vh1 phi1=x,phi2=y;
Vh1 phiref1=phi1,phiref2=phi2;

Vh1 dphi1,dphi2,Dphi1,Dphi2;

// elasticite non lineaire

macro phi [phi1,phi2]           // Déformation
macro phiref [phiref1,phiref2]          // Déformation precedente
macro dphi [dphi1,dphi2]        //
macro Dphi [Dphi1,Dphi2]        //

macro Dx(phi) [dx(phi[0]),dx(phi[1])]// Dérivée par rapport à x d'un vecteur
macro Dy(phi) [dy(phi[0]),dy(phi[1])]// Dérivée par rapport à y
macro E(phi)  [(Dx(phi)'*Dx(phi)-1.)/2.,(Dx(phi)'*Dy(phi))/2., (Dy(phi)'*Dy(phi)-1.)/2.] // tenseur de déformation
macro dE(phi,dphi)
```

```
        [Dx(phi)'*Dx(dphi),
        (Dx(phi)'*Dy(dphi)+Dx(dphi)'*Dy(phi))/2.,
        Dy(phi)'*Dy(dphi)]// dérivée du tenseur de déformation


macro W(phi)
        mu*(E(phi)[0]*E(phi)[0]+2.*E(phi)[1]*E(phi)[1]+E(phi)[2]*E(phi)[2])
        +lambda/2.*(E(phi)[0]+E(phi)[2])*(E(phi)[0]+E(phi)[2]) // densité d'énergie
macro dW(phi,dphi)
        2.*mu*(E(phi)[0]*dE(phi,dphi)[0]+2.*E(phi)[1]*dE(phi,dphi)[1]+E(phi)[2]*dE(phi,dphi)[2])
        +lambda*(E(phi)[0]+E(phi)[2])*(dE(phi,dphi)[0]+dE(phi,dphi)[2]) // dérivée de la densité d'énergie
macro HW(phi,dphi,Dphi)
        (2.*mu*(dE(phi,Dphi)[0]*dE(phi,dphi)[0]
        +2.*dE(phi,Dphi)[1]*dE(phi,dphi)[1]
        +dE(phi,Dphi)[2]*dE(phi,dphi)[2])
        +lambda*(dE(phi,Dphi)[0]+dE(phi,Dphi)[2])*(dE(phi,dphi)[0]+dE(phi,dphi)[2])) // "approximation grossiere" de la dérivée seconde de l

problem elasticite(dphi1,dphi2,Dphi1,Dphi2,init=initialize)=
                int2d(Sh)(HW(phiref,dphi,Dphi))
                +int2d(Sh)(dW(phiref,Dphi))
                -int1d(Sh,qfe=qf1pElump)(Fc1*Dphi1+Fc2*Dphi2)
                +on(Dirichlet,dphi1=0,dphi2=0);


// Construction de la matrice de Masse
varf Masse(u,v)=int1d(Sh,qfe=qf1pElump)(u*v);

real J,Jprec;    //Energie
bool newconvex;
ofstream NRJ("NRJ");
real gammaprev=1.;

// We solve minmisation problem for an increasing compression
int igamma=0;
for (real gamma=1.;gamma>0.1;gamma-=0.02){
        phi2=gamma/gammaprev*phi2;
        gammaprev=gamma;
        for (int iAdapt=0;iAdapt<NbAdapt;iAdapt++){
        ////////////////////////////////////////////
        // Initialisation                         //
        ////////////////////////////////////////////

        // Assemblage de la matrice de masse
        real[int] omega(Vh1.ndof);
        {
                matrix M=Masse(Vh1,Vh1);
                omega=M.diag;
        }
        newconvex=true;
        // Determination de la liste des points du bord
        int[int] subnbpointsbord(nbreg);
        int nbpointsbord=0;
```

```
int1d(Sh,qfe=qf1pE)(1.*(nbpointsbord++));

for(k=0;k<nbreg;k++){
                        int nreg=labelreg[k];
                        subnbpointsbord[k]=0;
                        int1d(Sh,qfe=qf1pE)(1.*(subnbpointsbord[k]+=(reg==nreg)));
}
nbpointsbord=subnbpointsbord.sum;
cout<<"Nb de points du bord="<<nbpointsbord<<endl;

// Determination des numeros des noeuds sur le bord
// et de la connecitivite du maillage
int[int,int] a(2,nbpointsbord);
int[int,int] e(2,nbpointsbord);
int[int] Idv(nbpointsbord);
{
        int[int] aa(2*nbpointsbord);

        Vh1 index=0;
        for(int i=0;i<Vh1.ndof;i++) index[][i]=i;

        int n=0;
        int[int] first(nbreg);
        for(int k=0;k<nbreg;k++){
                int m=n;
                int1d(Sh,qfe=qf1pElump)(1.*((reg==labelreg[k])?(aa[n++]=(1*index)):0.));
                first[k]=aa[m];
                cout<<"n="<<n<<endl;
        }
        cout<<"first="<<first<<endl<<"--------------"<<endl;
        Vh1 next=0;
        for(int i=0;i<nbpointsbord;i++) next[][aa[2*i]]=aa[2*i+1];


        n=0;
        for(int k=0;k<nbreg;k++){
                Idv[n]=first[k];
                for(int i=1;i<subnbpointsbord[k];i++) Idv[n+i]=next[][Idv[n+i-1]];
                n+=subnbpointsbord[k];
        }

        Vh1 numero=0;
        for (int i=0;i<nbpointsbord;i++) {numero[](Idv(i))=i;}

        n=0;
        int1d(Sh,qfe=qf1pElump)(1.*(aa(n++)=(1*numero)));
        for (int i=0;i<nbpointsbord;i++) {a(0,i)=aa(2*i+1); a(1,i)=aa(2*i); e(0,aa(2*i))=i; e(1,aa(2*i+1))=i;}
/*
        // Determination du max et min de Idv sur chacun des solides
```

```
        maxup=0; maxdown=0;
        minup=nbpointsbord;mindown=nbpointsbord;
        int1d(Sh,Neumannup,qfe=qf1pElump)(1.*((minup=1*min(minup,numero))*(maxup=1*max(maxup,numero))));
        int1d(Sh,Neumanndown,qfe=qf1pElump)(1.*((mindown=1*min(mindown,numero))*(maxdown=1*max(maxdown,numero))));
        cout<<"min,max up="<<minup<<","<<maxup<<endl;
        cout<<"min,max down="<<mindown<<","<<maxdown<<endl;
*/
        ofstream deformation("deformationInit");
        n=0;
        for (int k=0;k<nbreg;k++){
                for(int i=n;i<n+subnbpointsbord[k];i++) deformation<<Sh(Idv[i]).x<<" "<<Sh(Idv[i]).y<<endl;
                n+=subnbpointsbord[k];
                deformation<<endl;
        }
        plot(Sh,cmm="deformationInit saved");
}

// La deformations initiales (interpolation)
phi1=phi1;phi2=phi2;
phiref1=phiref1;phiref2=phiref2;
initialize=0;

// Les normales arete/noeud
real[int,int] n1(nbpointsbord,nbpointsbord),n2(nbpointsbord,nbpointsbord);
n1=0;n2=0;

// Determination des normales pour tous les couples de points arete/noeud

func int computen(){
        real alpha,sg,sd,tx,ty,nt,t1;
        for(int i1=0;i1<nbpointsbord;i1++)       // We list the edges
        {
                nt=(phi1[][Idv(a(0,i1))]-phi1[][Idv(a(1,i1))])^2+(phi2[][Idv(a(0,i1))]-phi2[][Idv(a(1,i1))])^2;
                nt=sqrt(nt); // longueur de l'arete
                tx=(phi1[][Idv(a(0,i1))]-phi1[][Idv(a(1,i1))]) / nt;
                ty=(phi2[][Idv(a(0,i1))]-phi2[][Idv(a(1,i1))]) / nt;
                for (int i2=0;i2<nbpointsbord;i2++)      // and the vertices
                if (!((a(0,i1)==i2)|(a(1,i1)==i2))) {
                        sg=tx*(phi1[][Idv(i2)]-phi1[][Idv(a(0,i1))])+ty*(phi2[][Idv(i2)]-phi2[][Idv(a(0,i1))]);
                        sd=-(tx*(phi1[][Idv(i2)]-phi1[][Idv(a(1,i1))])+ty*(phi2[][Idv(i2)]-phi2[][Idv(a(1,i1))]));
                        if ( (sg< 0) && (sd< 0) ) {
                                alpha=1.-2*(((phi1[][Idv(i2)]-phi1[][Idv(a(1,i1))])*ty-(phi2[][Idv(i2)]-phi2[][Idv(a(1,i1))])*tx)>0
                                n1(i1,i2)= alpha*ty ;
                                n2(i1,i2)=-alpha*tx;
                        }
                        else  if ( sg>= 0) {
                                t1=(phi1[][Idv(i2)]-phi1[][Idv(a(0,i1))])^2+(phi2[][Idv(i2)]-phi2[][Idv(a(0,i1))])^2;
                                t1=sqrt(t1);
                                n1(i1,i2)=-(phi1[][Idv(i2)]-phi1[][Idv(a(0,i1))])/t1;
```

```
                                n2(i1,i2)=-(phi2[][Idv(i2)]-phi2[][Idv(a(0,i1))])/t1;
                                }
                                else {
                                        t1=(phi1[][Idv(i2)]-phi1[][Idv(a(1,i1))])^2+(phi2[][Idv(i2)]-phi2[][Idv(a(1,i1))])^2;
                                        t1=sqrt(t1);
                                        n1(i1,i2)=-(phi1[][Idv(i2)]-phi1[][Idv(a(1,i1))])/t1;
                                        n2(i1,i2)=-(phi2[][Idv(i2)]-phi2[][Idv(a(1,i1))])/t1;
                                }
                        }
        }
}

// normales reduites Remarque: on pourrait reduire le temps de calcul et le stockage en utilisant la symetrie du probleme
real[int,int] Mn1(nbpointsbord,nbpointsbord),mn1(nbpointsbord,nbpointsbord),Mn2(nbpointsbord,nbpointsbord),mn2(nbpointsbord,nbpoints

// Reduction of the constraints
func int reduction() {
        for(int i1=0;i1<nbpointsbord;i1++)      // We list the vertices ...
        for (int i2=0;i2<nbpointsbord;i2++)     // ... and the vertices
        if (i1!=i2){
                bool Firstvalm=true;
                bool FirstvalM=true;
                real v1=phi1[](Idv(i1))-phi1[](Idv(i2));
                real v2=phi2[](Idv(i1))-phi2[](Idv(i2));
                for (int c=0;c<2;c++) // Normales a tester = n(e(0,i1),i2); n(e(1,i1),i2); -n(e(0,i2),i1); -n(e(1,i2),i1);
                        if ((n1(e(c,i1),i2)!=0)|(n2(e(c,i1),i2)!=0)){
                        real alpha =n1(e(c,i1),i2)*v2-n2(e(c,i1),i2)*v1;
                        real Malpha=Mn1(i1,i2)*v2-Mn2(i1,i2)*v1;
                        real malpha=mn1(i1,i2)*v2-mn2(i1,i2)*v1;
                        if ((alpha>Malpha)|FirstvalM) {
                                Mn1(i1,i2)=n1(e(c,i1),i2);
                                Mn2(i1,i2)=n2(e(c,i1),i2);
                                FirstvalM=false;
                                }
                        if ((alpha<malpha)|Firstvalm) {
                                mn1(i1,i2)=n1(e(c,i1),i2);
                                mn2(i1,i2)=n2(e(c,i1),i2);
                                Firstvalm=false;
                                }
                        }
                for (int c=0;c<2;c++) // Normales a tester = n(e(0,i1),i2); n(e(1,i1),i2); -n(e(0,i2),i1); -n(e(1,i2),i1);
                        if ((n1(e(c,i2),i1)!=0)|(n2(e(c,i2),i1)!=0)){
                        real alpha =-n1(e(c,i2),i1)*v2+n2(e(c,i2),i1)*v1;
                        real Malpha=Mn1(i1,i2)*v2-Mn2(i1,i2)*v1;
                        real malpha=mn1(i1,i2)*v2-mn2(i1,i2)*v1;
                        if ((alpha>Malpha)|FirstvalM) {
                                Mn1(i1,i2)=-n1(e(c,i2),i1);
```

```
                        Mn2(i1,i2)=-n2(e(c,i2),i1);
                        FirstvalM=false;
                    }
                    if ((alpha<malpha)|Firstvalm) {
                        mn1(i1,i2)=-n1(e(c,i2),i1);
                        mn2(i1,i2)=-n2(e(c,i2),i1);
                        Firstvalm=false;
                    }
                    }
                }
}

real[int,int] Mlambda(nbpointsbord,nbpointsbord);       // Lagrange Multipliers for the const raints for all couple (vertex,vertex)
real[int,int] mlambda(nbpointsbord,nbpointsbord);       // ... and for the normal mn
real[int,int] MC(nbpointsbord,nbpointsbord);            // Les contraintes
real[int,int] mC(nbpointsbord,nbpointsbord);            //
real[int] nbconstraints(nbpointsbord);                  // Number of active constraints
real[int] nblambda(nbpointsbord);                       // Number of active constraints

Mlambda=0; mlambda=0;
int iterconvex=0;
int iterUzawa=0;
Vh1 phiprec1,phiprec2;                                  // The deformations

// end of initialisation ////////////////////////////////////////////////////////////////////////////
while ((iterUzawa!=1) &(newconvex))                                     // Convexes loop
{

        phiref1=phi1;phiref2=phi2;      // il faudrait ajouter une boucle ici, jusqu'a convergence entre phiref et phi
        initialize=0;
        // Computation of the initial normals
        computen();
        reduction();

        bool nonlinearLoop=true;
        iterUzawa=0;
        int[int] I1(0);
        int[int] I2(0);


        while(nonlinearLoop){

        phiref1=phi1;phiref2=phi2;

        initialize=0;
        bool notenough=true;
        while (notenough){
        cout<<"---------- Uzawa :"<<I1.n<<" constraints ----------"<<endl;

        bool admissible=false;                                          // if admissible=true there is no intersections
```

```
bool poursuit=true;                                          // Shall we still performing Uzawa steps ?
real alphaeff=alpha;                                         // effective Uzawa steps.

while(poursuit) {                                            // Uzawa Loop
        cout<<"Nb-adap: "+iAdapt<<";  Iterations: Convex->"<<iterconvex<<";  Uzawa->"<<iterUzawa<<endl;
        admissible=true;

        // Compute contact forces ================================================================================//
        Fc1=0;Fc2=0;

        for (int i=0;i<I1.n;i++){
        int i1=I1[i];int i2=I2[i];
        {
                if (i1!=i2) {
                        Fc1[][Idv(i2)]-=(mlambda(i1,i2)*mn1(i1,i2)+Mlambda(i1,i2)*Mn1(i1,i2))*omega(Idv(i1));
                        Fc2[][Idv(i2)]-=(mlambda(i1,i2)*mn2(i1,i2)+Mlambda(i1,i2)*Mn2(i1,i2))*omega(Idv(i1));
                        Fc1[][Idv(i1)]+=(mlambda(i1,i2)*mn1(i1,i2)+Mlambda(i1,i2)*Mn1(i1,i2))*omega(Idv(i2));
                        Fc2[][Idv(i1)]+=(mlambda(i1,i2)*mn2(i1,i2)+Mlambda(i1,i2)*Mn2(i1,i2))*omega(Idv(i2));
                }
        }}
        // Solving Primal Problem (linear elasticity here) ===================================================//
        verbosity=0;
        phiprec1=phi1;phiprec2=phi2;
        Jprec=J;
        elasticite;
        //plot([Fc1,Fc2],coef=0.1);
        initialize=1;   // we do not rebuild the rigidity matrix for the next steps.
        phi1=phiref1+dphi1; phi2=phiref2+dphi2;
        real delta=int2d(Sh)((phiprec1-phi1)^2+(phiprec2-phi2)^2);

        // Updating the constraints ==========================================================//
        nbconstraints=0;
        int nbconstraintstot=0;
        real Cmax=0;
        for (int i=0;i<I1.n;i++){
        int i1=I1[i];int i2=I2[i];
        if (i1!=i2) {
                mC(i1,i2)=epsilon-(mn1(i1,i2)*(phi1[][Idv(i1)]-phi1[][Idv(i2)])+mn2(i1,i2)*(phi2[][Idv(i1)]-phi2[][Idv(i2)]))
                MC(i1,i2)=epsilon-(Mn1(i1,i2)*(phi1[][Idv(i1)]-phi1[][Idv(i2)])+Mn2(i1,i2)*(phi2[][Idv(i1)]-phi2[][Idv(i2)]))
                nbconstraints(i2)+=omega(Idv(i1))*((MC(i1,i2)>0)+(mC(i1,i2)>0));
                nbconstraints(i1)+=omega(Idv(i2))*((MC(i1,i2)>0)+(mC(i1,i2)>0));
                nbconstraintstot+=(MC(i1,i2)>0)+(mC(i1,i2)>0);
                Cmax=max(Cmax,MC(i1,i2));
                Cmax=max(Cmax,mC(i1,i2));
                if (!((MC(i1,i2)<epsilon*tol)& (mC(i1,i2)<epsilon*tol))) admissible=false;
        }}

        // Updating Lagrange Multipliers ==========================================================//
        nblambda=0;
        int nblambdatot=0;
```

```
        real deltalambda=0;
        for (int i=0;i<I1.n;i++){
        int i1=I1[i];int i2=I2[i];
        if (i1!=i2) {
                real interm=mlambda(i1,i2);
                real interM=Mlambda(i1,i2);
                mlambda(i1,i2)=max(0.,mlambda(i1,i2)+alphaeff/(nbconstraints(i2)+nbconstraints(i1)+omega(Idv(i1))+omega(Idv(
                Mlambda(i1,i2)=max(0.,Mlambda(i1,i2)+alphaeff/(nbconstraints(i2)+nbconstraints(i1)+omega(Idv(i1))+omega(Idv(
                //mlambda(i1,i2)=max(0.,mlambda(i1,i2)+alphaeff*mC(i1,i2));
                //Mlambda(i1,i2)=max(0.,Mlambda(i1,i2)+alphaeff*MC(i1,i2));
                deltalambda+=(abs(mlambda(i1,i2)-interm)+abs(Mlambda(i1,i2)-interM))*omega(Idv(i1))*omega(Idv(i2));
                nblambda(i2)+=(mlambda(i1,i2)>0)+(Mlambda(i1,i2)>0);
                nblambda(i1)+=(mlambda(i1,i2)>0)+(Mlambda(i1,i2)>0);
                nblambdatot+=(mlambda(i1,i2)>0)+(Mlambda(i1,i2)>0);
        }}

        // Just to check if everything is going all right
        cout<<"; Nb constraints violated ->"<<nbconstraintstot<<" ; Nb constraints active ->"<<nblambdatot<<endl;
        cout<<"delta="<<delta<<endl;
        cout<<"deltalambda="<<deltalambda<<endl;
        // Adapting Uzawa step ============================================================//
        alphaeff=alpha*max(1.,0.5*log(delta+1.e-10)^2);
        //alphaeff=alpha;
        cout<<"pas effectif="<<alphaeff<<endl;
        cout<<"Cmax="<<Cmax<<endl;
        cout<<"-----------------------------------------"<<endl;
        iterUzawa++;

        // Keep on or leave ? ============================================================//
        if (admissible && (delta<deltamin)) poursuit=false;
        //poursuit=true;
}// End of Uzawa loop

// Compute new activated constraints
real[int,int] activated(nbpointsbord,nbpointsbord);
for(int i1=0;i1<nbpointsbord;i1++)      // We list the vertices ...
for (int i2=0;i2<i1;i2++)       // ... and the vertices
if (i1!=i2) {
        activated(i1,i2)=((epsilon-(mn1(i1,i2)*(phi1[][Idv(i1)]-phi1[][Idv(i2)])+mn2(i1,i2)*(phi2[][Idv(i1)]-phi2[][Idv(i2)])
                                        ((epsilon-(Mn1(i1,i2)*(phi1[][Idv(i1)]-phi1[][Idv(i2)])+Mn2(i1,i2)*(phi2[][Idv(i1)]-
}
for (int i=0;i<I1.n;i++) activated(I1(i),I2(i))=1;
matrix Inter=activated;
int nbprec=I1.n;
real[int] C(0);
[I1,I2,C]=Inter;
if (C(C.n-1)==0){
        I1.resize(I1.n-1);
```

```
                l2.resize(l2.n-1);
        };
        int nbnew=l1.n;

        cout<<"========================================="<<endl;
        cout<<"NB constraints activated="<<nbnew<<endl;
        cout<<"========================================="<<endl;
        if (nbnew==nbprec) notenough=false;
}
cout<<"----------------- END OF UZAWA -----------------"<<endl;
real deltanonlinear=sqrt(int2d(Sh)((phi-phiref)'*(phi-phiref)));
cout<<" delta non linear="<<deltanonlinear<<endl;
if (deltanonlinear<1.e-3) nonlinearLoop=false;
}
        cout<<"---------------- END OF NON LINEAR LOOP ----------------"<<endl;
{
        real[int] b1(nbpointsbord+1),b2(nbpointsbord+1);
        for(int i=0;i<nbpointsbord;i++) {b1(i)=phi1[][ldv(i)];b2(i)=phi2[][ldv(i)];};
        b1(nbpointsbord)=b1(0);b2(nbpointsbord)=b2(0);
        plot ([b1,b2],wait=1,bb=[[-Ly/10-Ly/2.,-Ly/10],[Ly*1.1-Ly/2.,Ly*1.1]],cmm="end of nonlinear loop gamma="+gamma);
}
        J=int2d(Sh)(W(phi));
        if ((J>Jprec)&(iterconvex!=0)) {newconvex=false;} else {newconvex=true;};
        NRJ<<J<<endl;

// Plot the forces on the deformed configuration
        string  legende="Nb-adaptation="+iAdapt+"; Iteration convexe "+iterconvex+"; J="+J;

        Th=movemesh(Sh,[phi1,phi2]);
        Fcdef1=0;Fcdef2=0;
        Fcdef1[]=Fc1[];Fcdef2[]=Fc2[];
        matrix M2=Masse2(Wh,Wh);
        real[int] omega2(Wh.ndof);
        omega2=M2.diag;
        for(int i=0;i<nbpointsbord;i++){
                Fcdef1[][ldv(i)]=Fcdef1[][ldv(i)]*omega(ldv(i))/omega2(ldv(i));
                Fcdef2[][ldv(i)]=Fcdef2[][ldv(i)]*omega(ldv(i))/omega2(ldv(i));
        }
        plot([Fcdef1,Fcdef2],cmm=legende,wait=1);

        iterconvex++;
}// End of convexes loop
NRJ<<endl;
// save the deformation and the contact forces for plotting
{
                ofstream deformation("deformation"+iAdapt+"-g"+gamma);
                ofstream contactforce("contactForces"+iAdapt+"-g"+gamma);
                int n=0;
                deformation<<nbreg<<endl;
```

```
                        contactforce<<nbpointsbord<<endl;
                        for (int k=0;k<nbreg;k++){
                                deformation<<subnbpointsbord[k]<<endl;
                                for(int i=n;i<n+subnbpointsbord[k];i++) deformation<<phi1[][Idv[i]]<<" "<<phi2[][Idv[i]]<<endl;
                                for(int i=n;i<n+subnbpointsbord[k];i++){
                                        //contactforce<<phi1[][Idv[i]]<<" "<<phi2[][Idv[i]]<<"  "<<Fcdef1[][Idv[i]]<<" "<<Fcdef2[][Idv[i]]<<
                                        contactforce<<phi1[][Idv[i]]<<" "<<phi2[][Idv[i]]<<"  "<<Fc1[][Idv[i]]<<" "<<Fc2[][Idv[i]]<<endl;
                                }
                                n+=subnbpointsbord[k];
                        }
                        plot(Th,ps="Th"+iAdapt+"-g"+gamma+".eps",bb=[[-Ly,-Ly],[Ly,Ly]]);
                        savemesh(Th,"Th"+igamma+".msh");
        }

        // Adaptation du maillage
        if (iAdapt+1<NbAdapt) {
                        // Erreur sur les conditions de contact
                        Vh1 error=0;
                        for(int i1=0;i1<nbpointsbord;i1++)      // We list the vertices ...
                        for (int i2=0;i2<i1;i2++)       // ... and the vertices
        //              for(int i1=minup;i1<=maxup;i1++)        // We list the vertices ...
        //              for (int i2=mindown;i2<=maxdown;i2++)   // ... and the vertices
                        {
                                if (((mlambda(i1,i2)!=0)|(Mlambda(i1,i2)!=0))){
                                        real errm=mlambda(i1,i2)*((phi1[][Idv(i1)]-phi1[][Idv(i2)])*mn2(i1,i2)-(phi2[][Idv(i1)]-phi2
                                        real errM=Mlambda(i1,i2)*((phi1[][Idv(i1)]-phi1[][Idv(i2)])*Mn2(i1,i2)-(phi2[][Idv(i1)]-phi2
                                        error[][Idv(i1)]+=errm+errM;
                                        error[][Idv(i2)]-=errm+errM;
                                }
                        }
                        Vh1 erroext,derr;
                        solve forAdaptation(erroext,derr)=int2d(Sh)(0.01*(dx(erroext)*dx(derr)+dy(erroext)*dy(derr)) + erroext*derr)
                        -int1d(Sh)(error*derr);
                        plot(erroext,wait=1);
                        Sh=adaptmesh(Sh,[phi1,phi2,erroext],err=err,hmin=2*epsilon);
                        plot(Sh,cmm="adapted",wait=1);

                }
        }
igamma++;
} // end of the loop on the parametrized problems
```

What remains to be done ...

- Treatment of contacts with the help of a C++ code called by FreeFem++ !
- Implementation of the method using ”bundles”.
- Improve the Uzawa method by translating the Lagrange Multipliers (and not only by incrementation).
- Reduce the conditioning based on a definition of the admissible set on the non-discretized space.

Thank you for your Attention...