Développement d'un logiciel de calcul sur les domaines de probabilité maximale (MPDs)

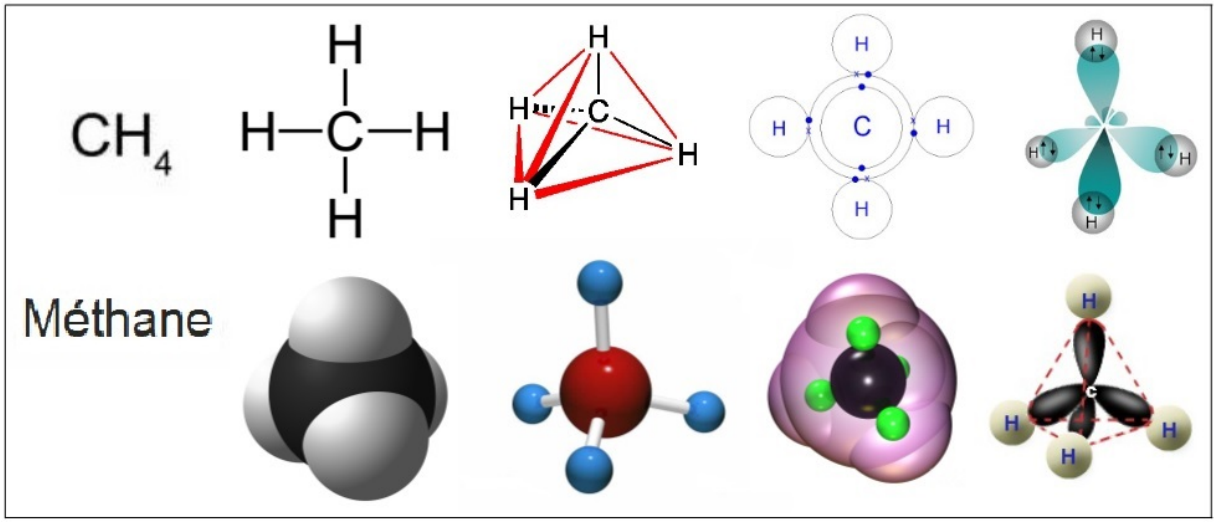
Présentation au Laboratoire de Chimie Théorique

Jérémy Dalphin (Ingénieur de recherche) – Jeudi 22 Février 2018

Travail en collaboration avec Pascal Frey (ISCD), Benoît Braïda (LCT), Yannick Privat (LJLL), Andreas Savin (LCT), Charles Dapogny (LJK), Guillaume Acke (Université de Gand), Eric Cancès (CERMICS)

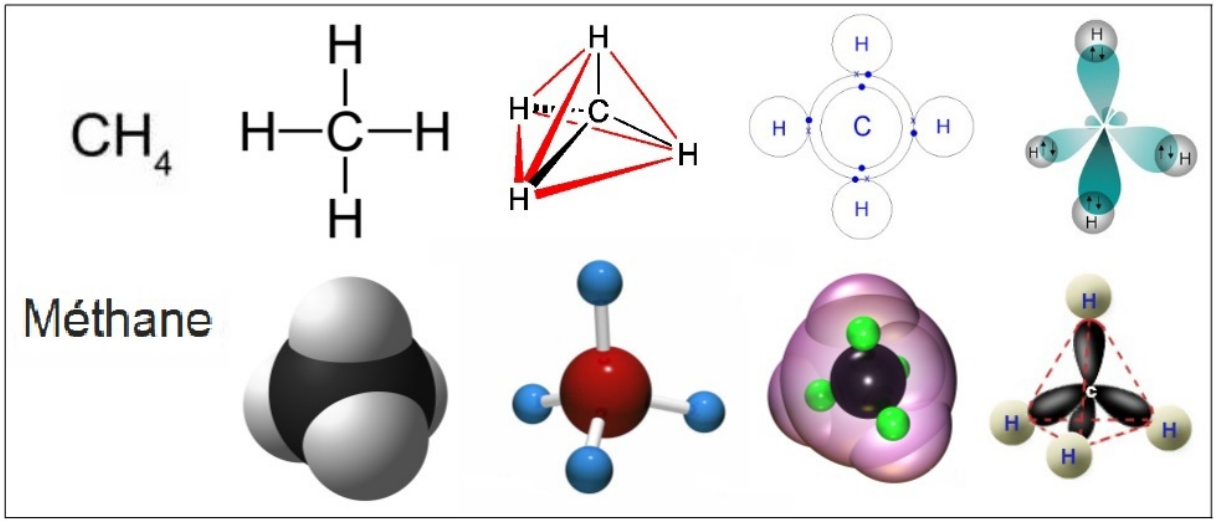
Introduction : présentation du modèle des MPDs

- D'un côté, la vision chimique traditionnelle a été fortement influencée par une vision localisée des électrons autour des noyaux.



Introduction : présentation du modèle des MPDs

- D'un côté, la vision chimique traditionnelle a été fortement influencée par une vision localisée des électrons autour des noyaux.



- D'un autre côté, la mécanique quantique permet aux électrons d'être délocalisés dans tout l'espace grâce à la fonction d'onde :

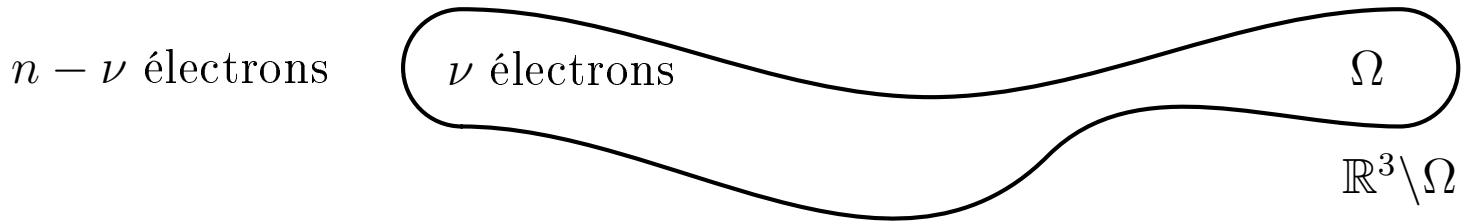
$$\begin{array}{rccc} \Psi : & \left(\mathbb{R}^3 \times \left\{-\frac{1}{2}, \frac{1}{2}\right\}\right)^n & \longrightarrow & \mathbb{C} \\ & \left[\binom{\mathbf{x}_1}{\sigma_1}, \ldots, \binom{\mathbf{x}_n}{\sigma_n}\right] & \longmapsto & \Psi\left[\binom{\mathbf{x}_1}{\sigma_1}, \ldots, \binom{\mathbf{x}_n}{\sigma_n}\right], \end{array}$$

qui permet de décrire précisément tout système chimique.

Introduction : présentation du modèle des MPDs

La mécanique quantique tente de réunifier ces deux approches par diverses techniques et modèles. Etant donné un

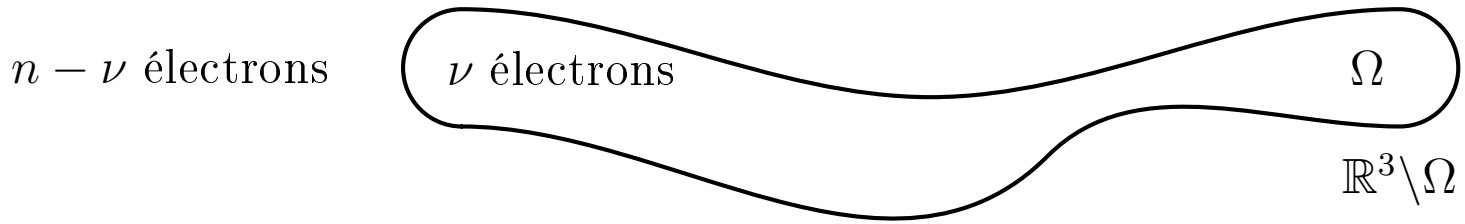
- système chimique à n électrons caractérisé par sa fonction d'onde Ψ,
- domaine ou région donnée Ω de l'espace tridimensionnel réel $\mathbb{R}^3$,
- nombre fixé d'électrons $\nu \in \{0, \ldots, n\}$ à rechercher,



Introduction : présentation du modèle des MPDs

La mécanique quantique tente de réunifier ces deux approches par diverses techniques et modèles. Etant donné un

- système chimique à n électrons caractérisé par sa fonction d'onde Ψ,
- domaine ou région donnée Ω de l'espace tridimensionnel réel $\mathbb{R}^3$,
- nombre fixé d'électrons $\nu \in \{0, \ldots, n\}$ à rechercher,



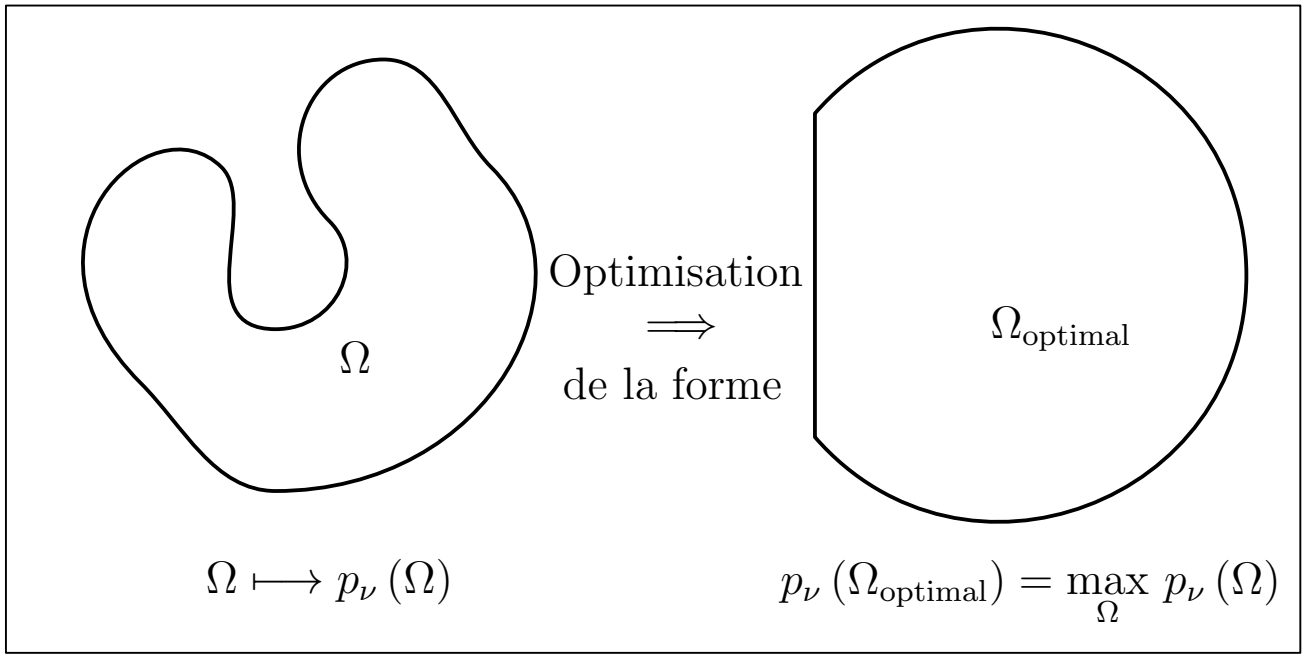
on peut calculer la probabilité $p_\nu(\Omega)$ de trouver exactement ν électrons dans Ω (et donc $n - \nu$ électrons sont dans le complémentaire $\mathbb{R}^3 \backslash \Omega$) :

$$p_\nu(\Omega) = \binom{n}{\nu} \sum_{(\sigma_1,\ldots,\sigma_n) \in \left\{-\frac{1}{2},\frac{1}{2}\right\}^n} \int_{\Omega^\nu \times \left(\mathbb{R}^3 \backslash \Omega\right)^{n-\nu}} \left| \Psi\left[\binom{\mathbf{x}_1}{\sigma_1}, \ldots, \binom{\mathbf{x}_n}{\sigma_n}\right] \right|^2 \mathrm{d}\mathbf{x}_1 \ldots \mathrm{d}\mathbf{x}_n.$$

Hypothèses : Ψ antisymétrique, de carré intégrable, et $p_n(\mathbb{R}^3) = 1$.

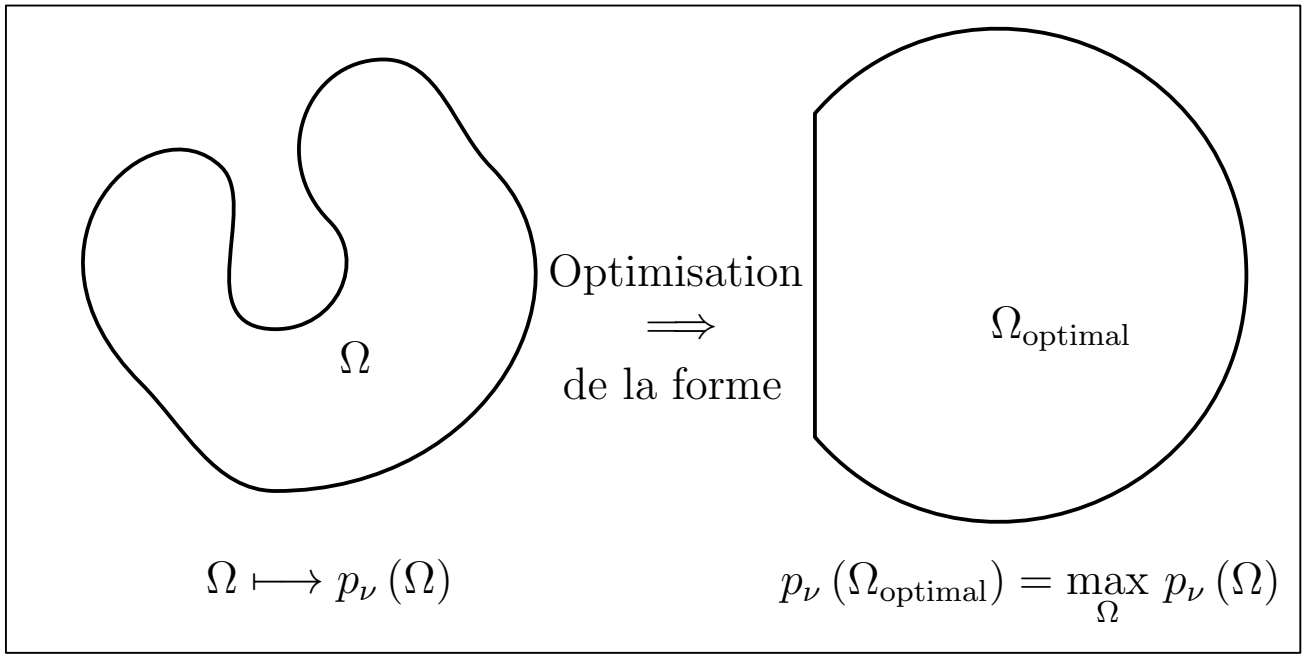
Introduction : présentation du modèle des MPDs

Proposition (Savin, 2002) : tenter de chercher le domaine qui maximise cette probabilité, en résolvant un problème d'optimisation.



Introduction : présentation du modèle des MPDs

Proposition (Savin, 2002) : tenter de chercher le domaine qui maximise cette probabilité, en résolvant un problème d'optimisation.



Définition : les *domaines de probabilité maximale* ou *MPDs* (pour *Maximal Probability Domains*) consiste à rechercher théoriquement et numériquement les domaines ou régions de l'espace qui sont maximiseurs (globaux/locaux/points critiques) pour la fonctionnelle $\Omega \mapsto p_\nu(\Omega)$.

Introduction : présentation du modèle des MPDs

Le modèle des MPDs permet de caractériser géométriquement et de visualiser dans l'espace réel tridimensionnel la structure électronique des molécules et de leurs interactions, et cela directement à partir de la fonction d'onde.

Introduction : présentation du modèle des MPDs

Le modèle des MPDs permet de caractériser géométriquement et de visualiser dans l'espace réel tridimensionnel la structure électronique des molécules et de leurs interactions, et cela directement à partir de la fonction d'onde.

- Par exemple, la probabilité de trouver exactement deux électrons peut être apparentée à une représentation d'une paire de Lewis.
- Savin (2002) a montré que les MPDs fournissent un rayon qui décrit très bien la séparation spatiale entre couches électroniques d'atomes lourds, ce qui n'est pas forcément le cas pour d'autres méthodes.
- Les MPDs permettent de visualiser les régions de valence (*Braïda et al.*, 2015) que ce soit dans une molécule simple (*Cancès et al.*, 2004), un liquide (*Agostini et al.*, 2015), un cristal (*Causà et al.*, 2012), ou autres composants inorganiques (*Causà et al.*, 2011).

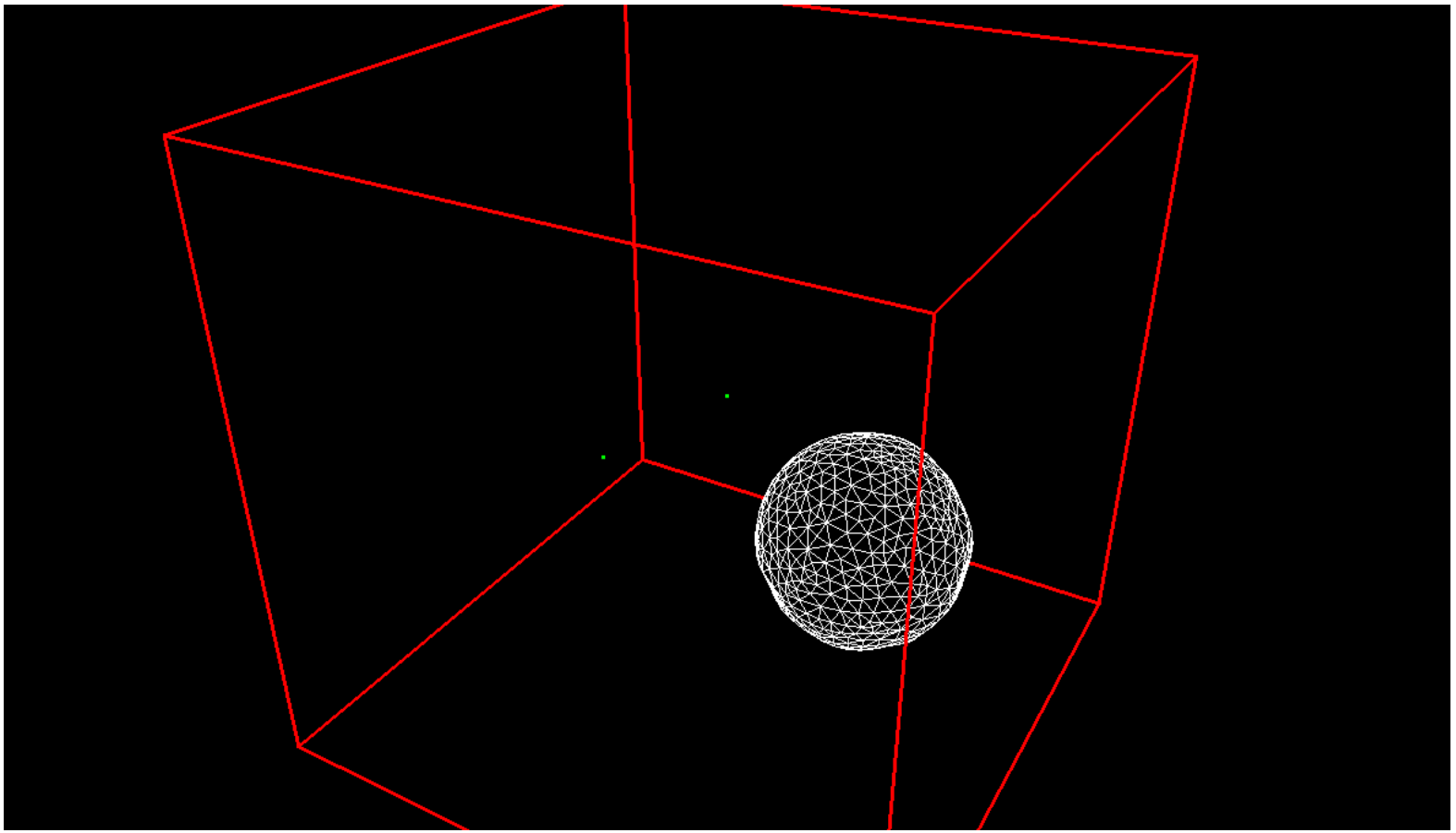
Introduction : présentation du modèle des MPDs

Le modèle des MPDs permet de caractériser géométriquement et de visualiser dans l'espace réel tridimensionnel la structure électronique des molécules et de leurs interactions, et cela directement à partir de la fonction d'onde.

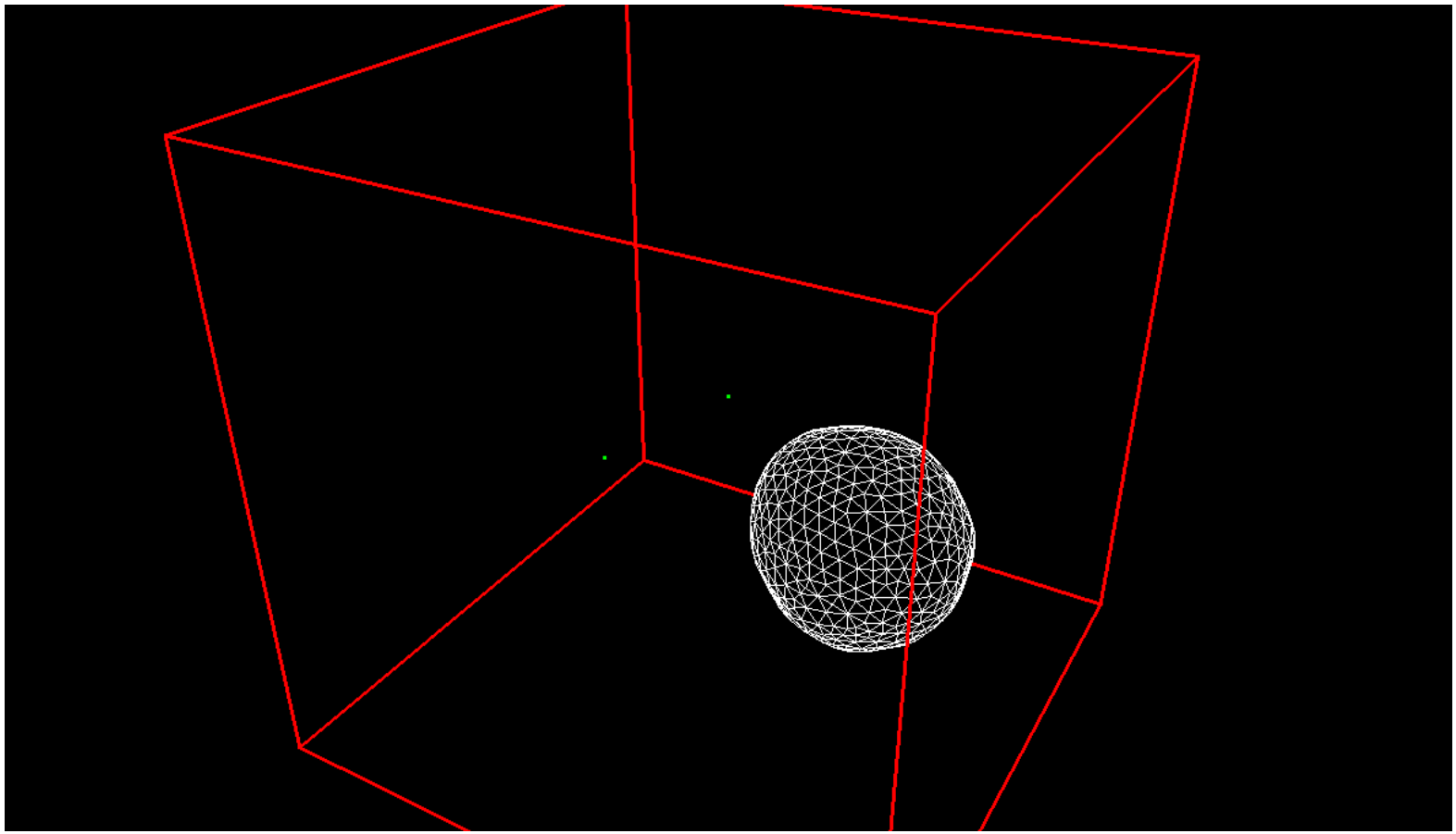
- Par exemple, la probabilité de trouver exactement deux électrons peut être apparentée à une représentation d'une paire de Lewis.
- Savin (2002) a montré que les MPDs fournissent un rayon qui décrit très bien la séparation spatiale entre couches électroniques d'atomes lourds, ce qui n'est pas forcément le cas pour d'autres méthodes.
- Les MPDs permettent de visualiser les régions de valence (*Braïda et al.*, 2015) que ce soit dans une molécule simple (*Cancès et al.*, 2004), un liquide (*Agostini et al.*, 2015), un cristal (*Causà et al.*, 2012), ou autres composants inorganiques (*Causà et al.*, 2011).

Objectif : développer un logiciel pour obtenir numériquement des MPDs.

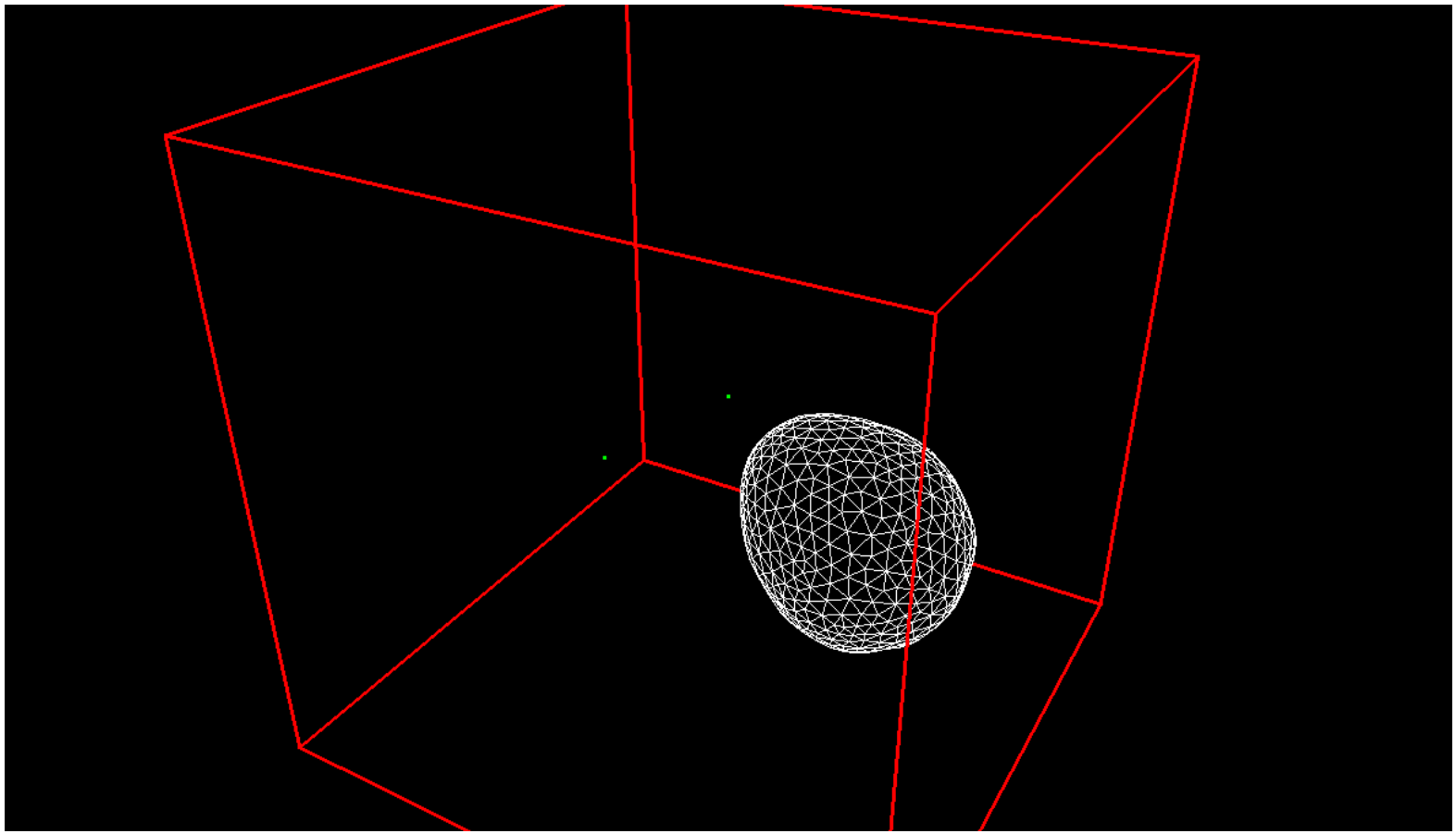
Introduction : exemple de la molécule H_2O (itération 0)



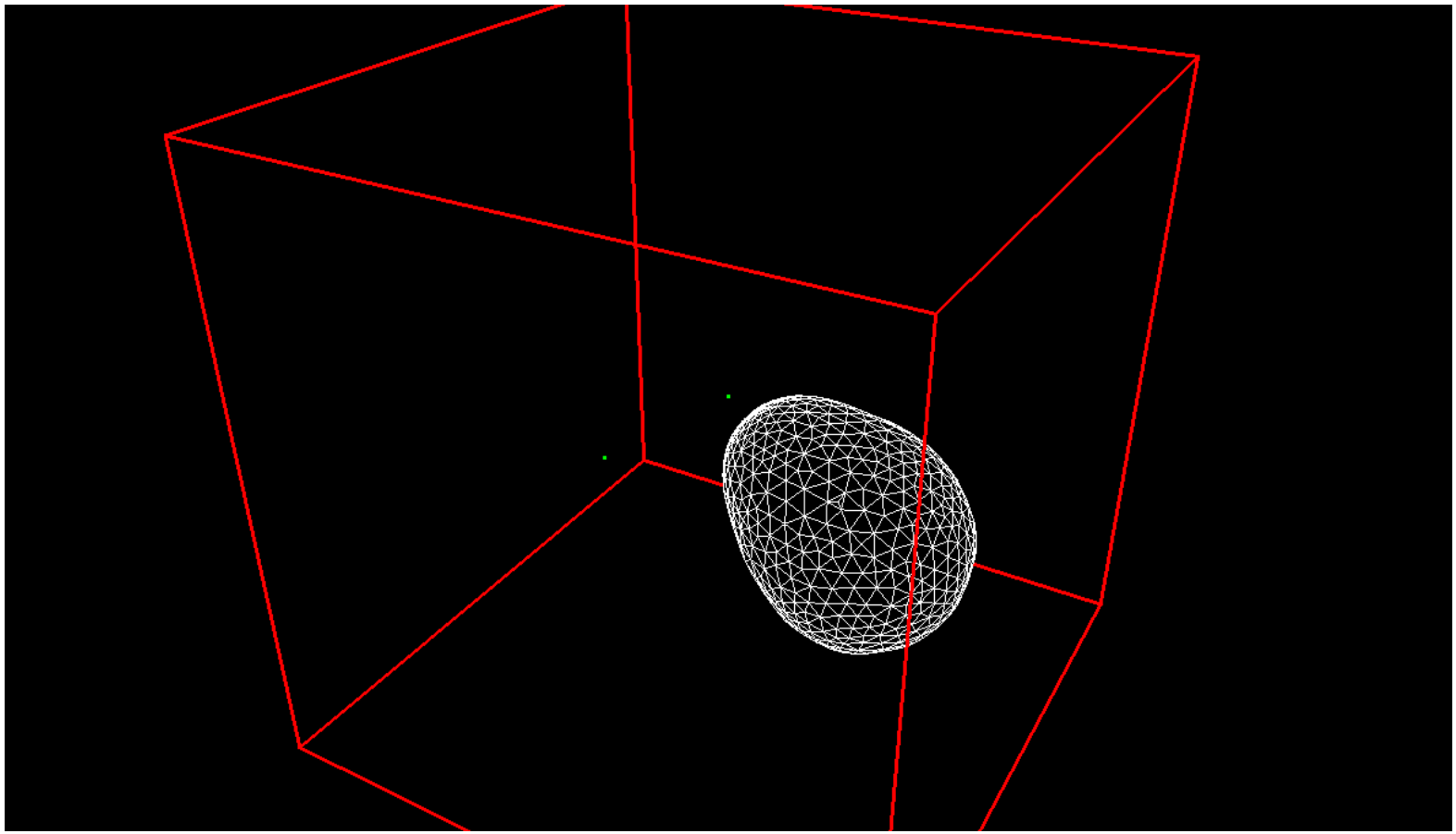
Introduction : exemple de la molécule H_2O (itération 1)



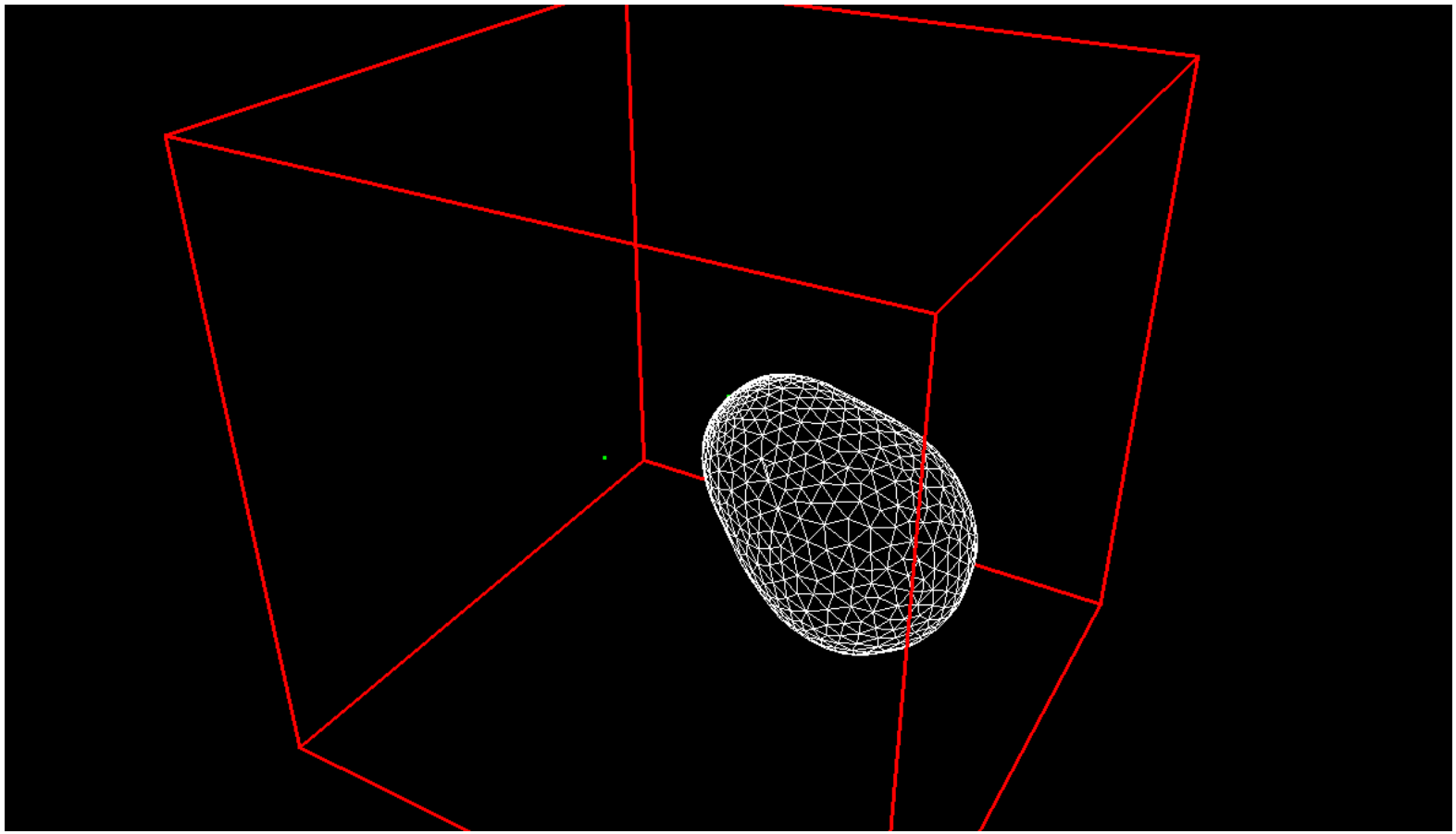
Introduction : exemple de la molécule H_2O (itération 2)



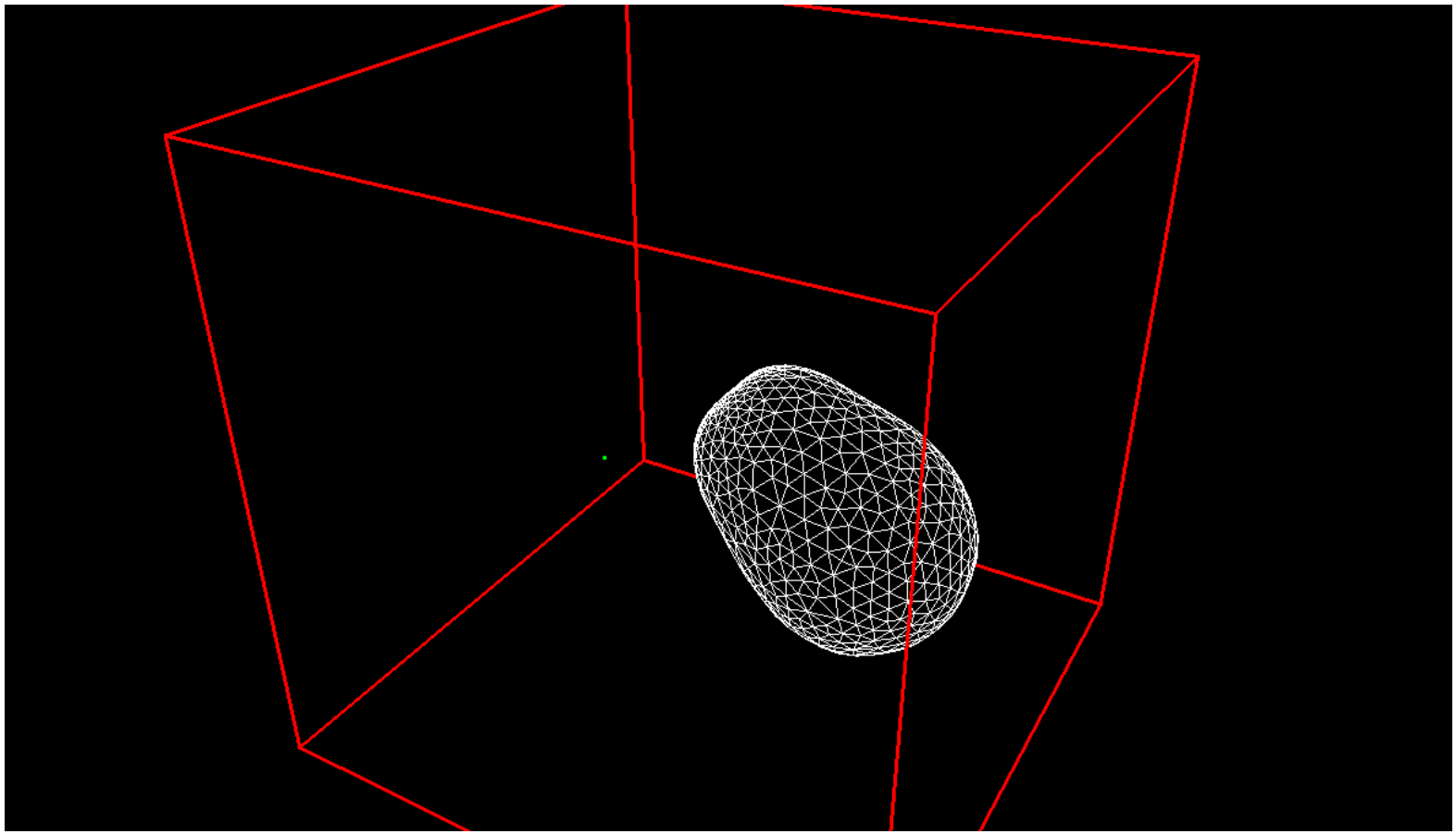
Introduction : exemple de la molécule H_2O (itération 3)



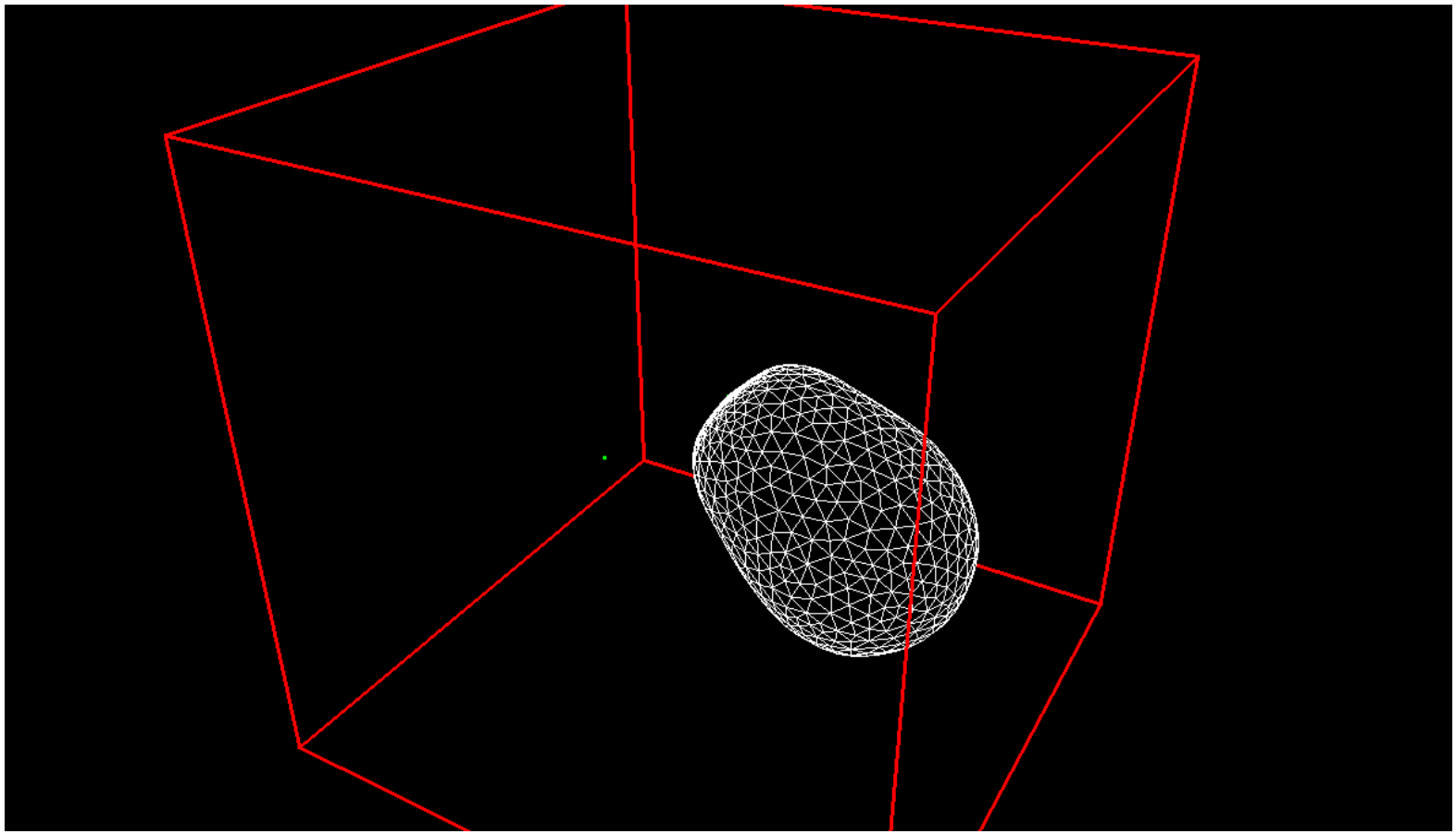
Introduction : exemple de la molécule H_2O (itération 4)



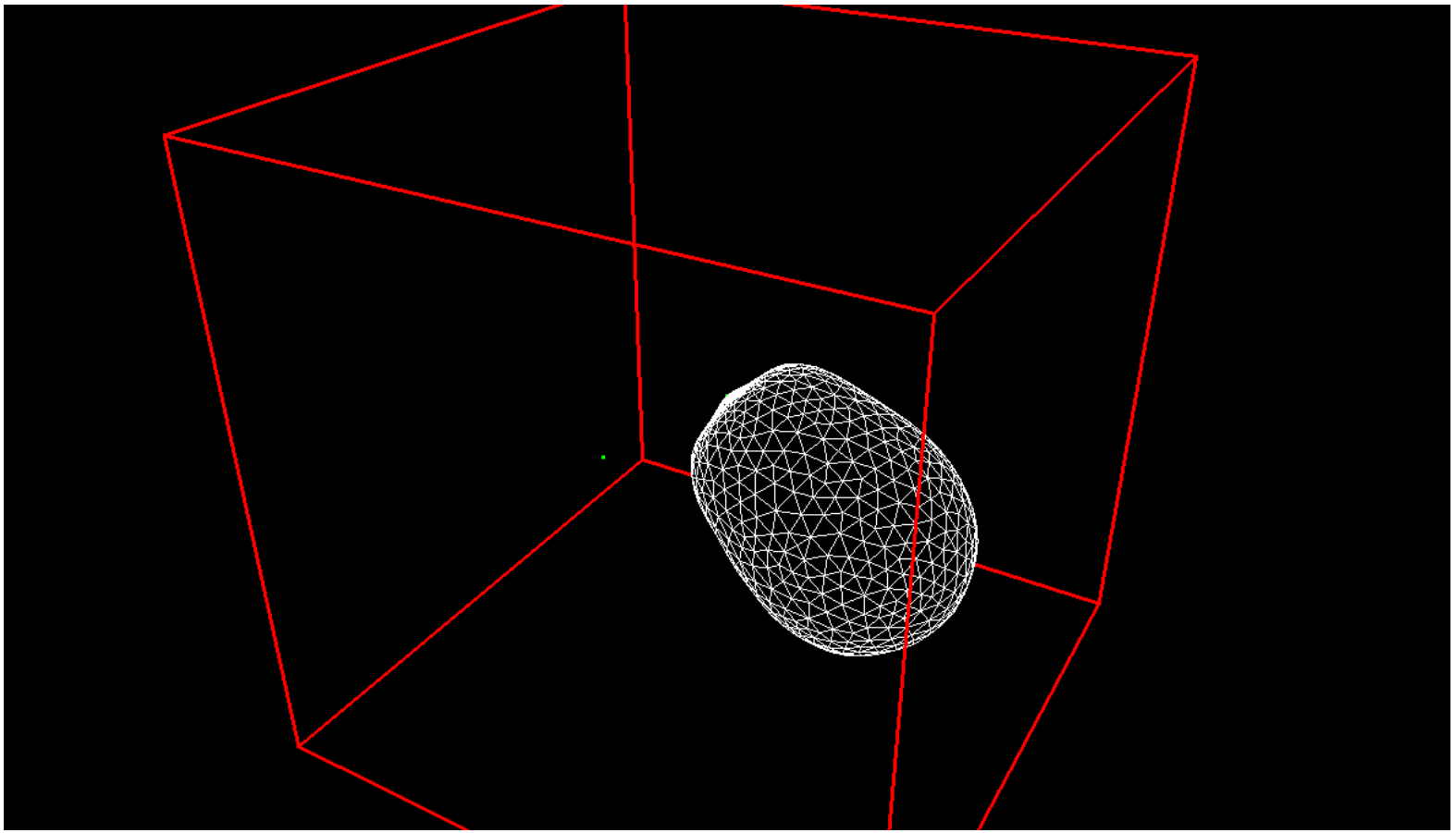
Introduction : exemple de la molécule H_2O (itération 5)



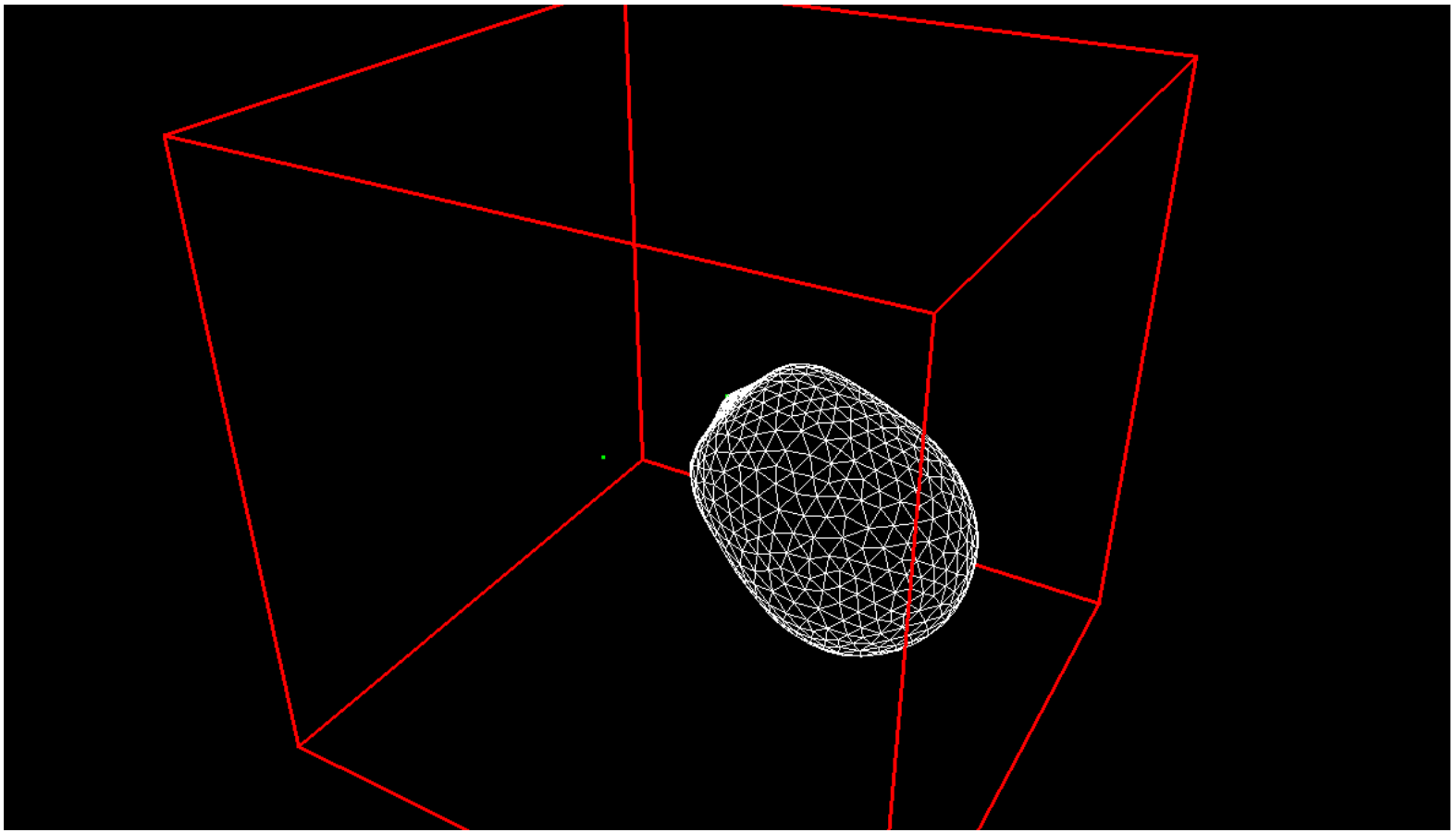
Introduction : exemple de la molécule H_2O (itération 6)



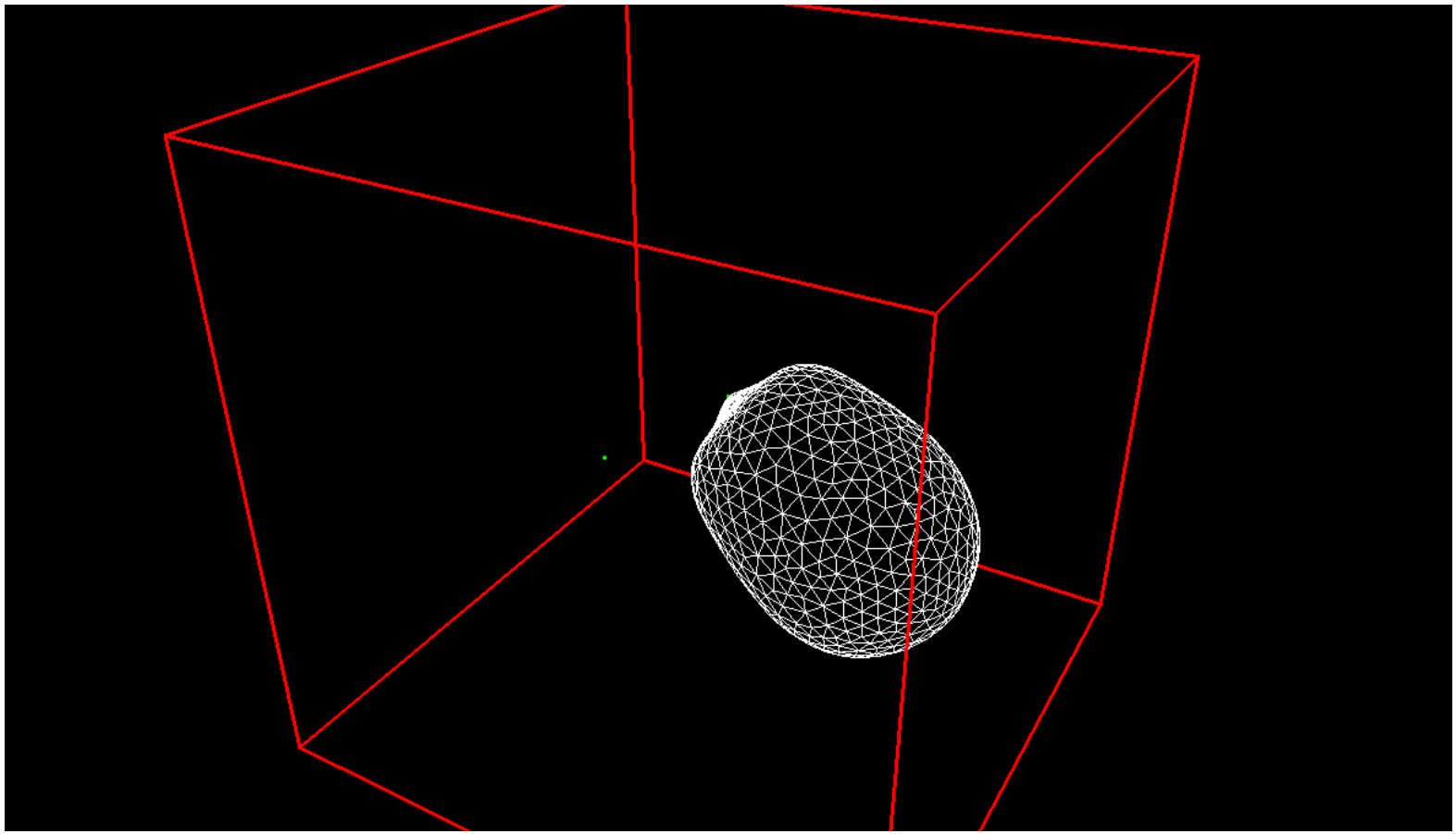
Introduction : exemple de la molécule H_2O (itération 7)



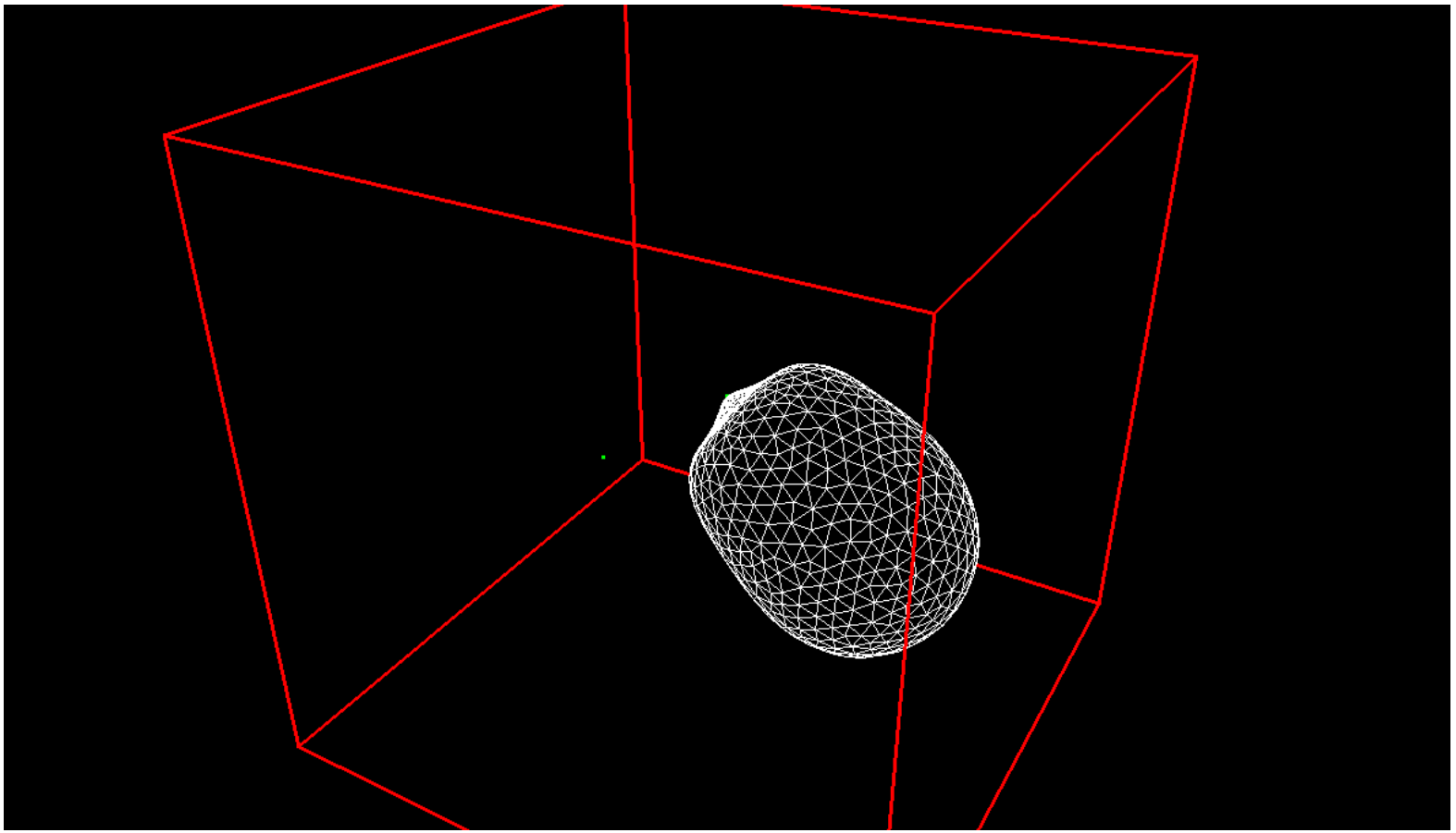
Introduction : exemple de la molécule H_2O (itération 8)



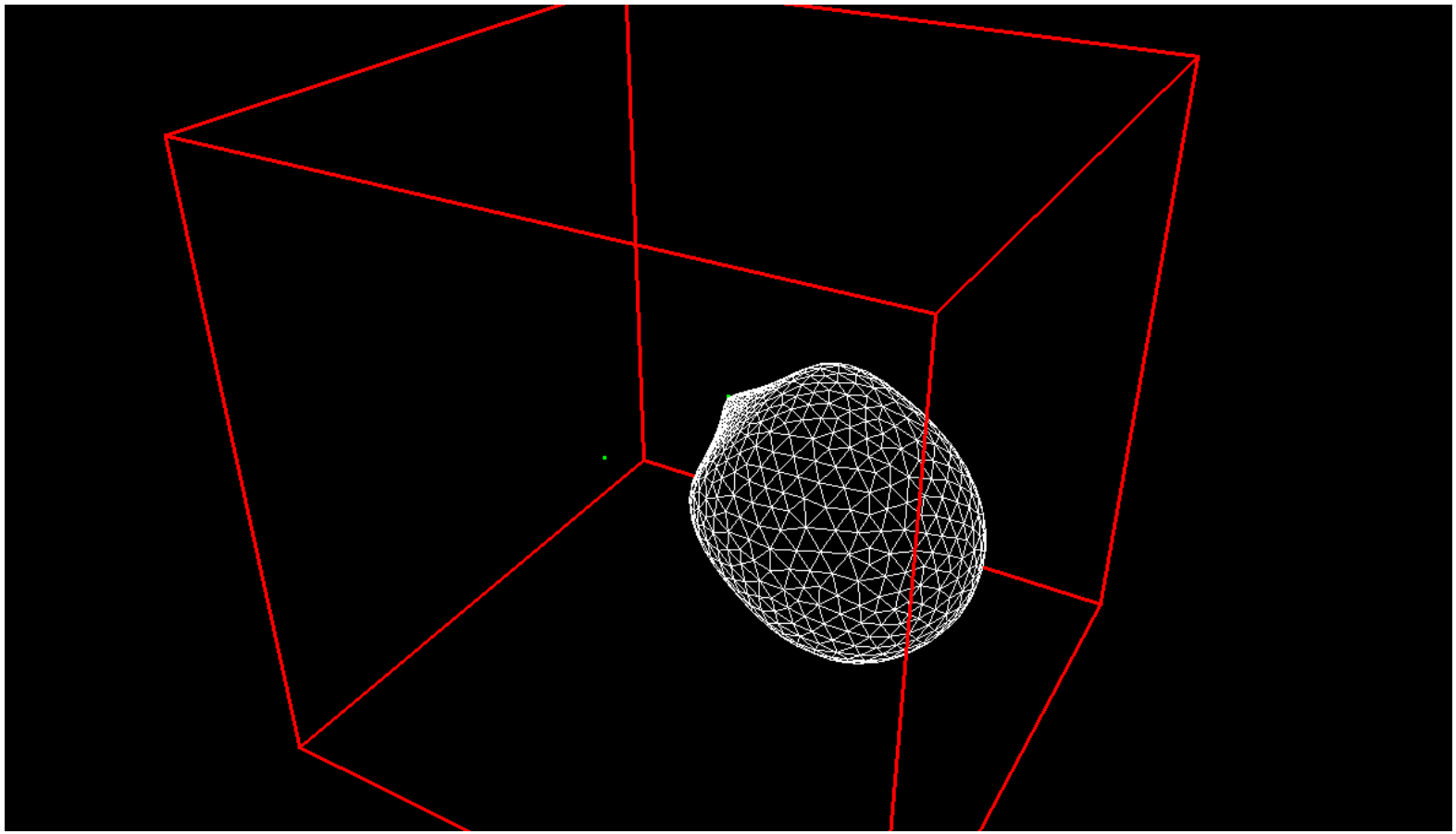
Introduction : exemple de la molécule H_2O (itération 9)



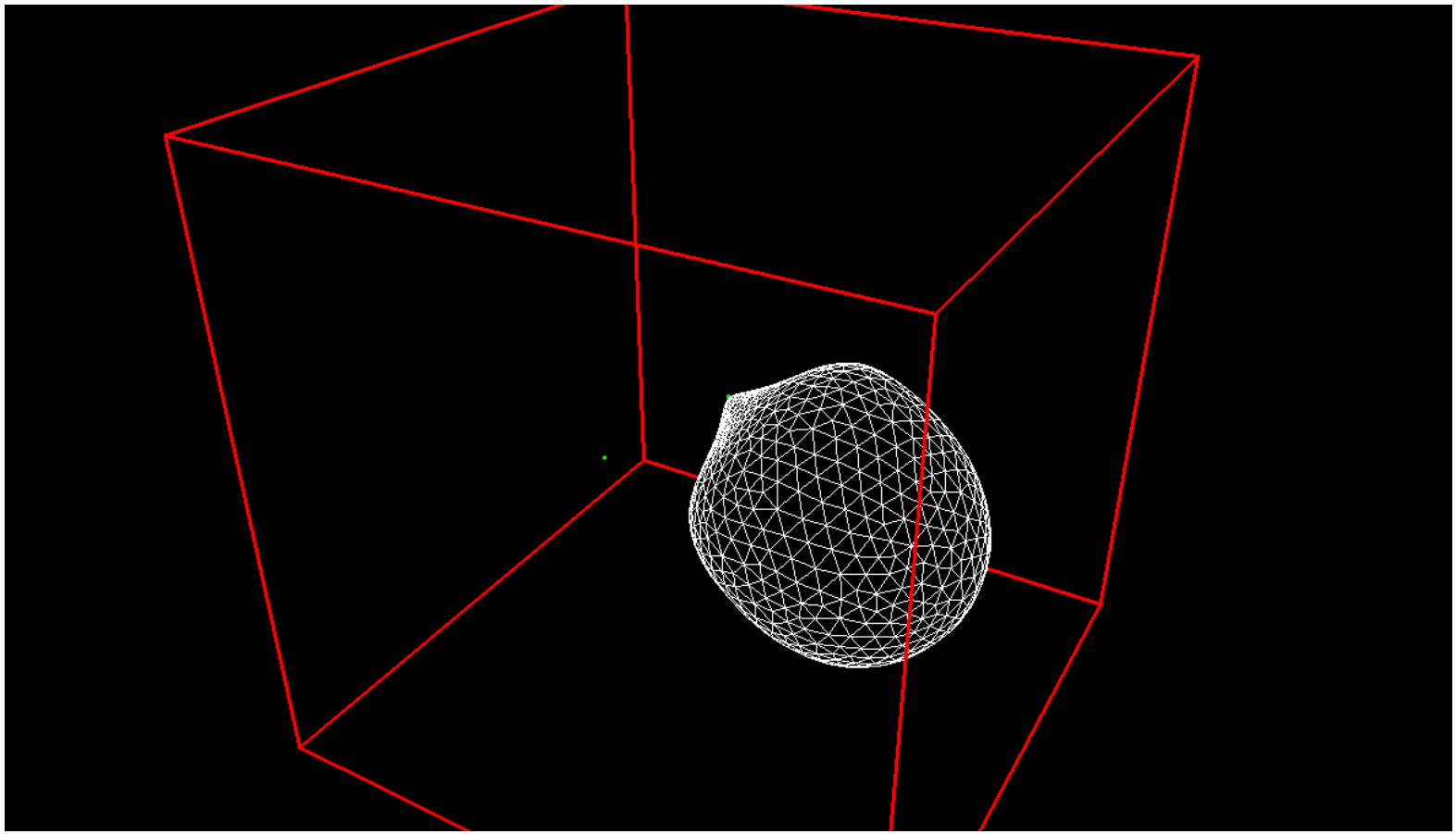
Introduction : exemple de la molécule H_2O (itération 10)



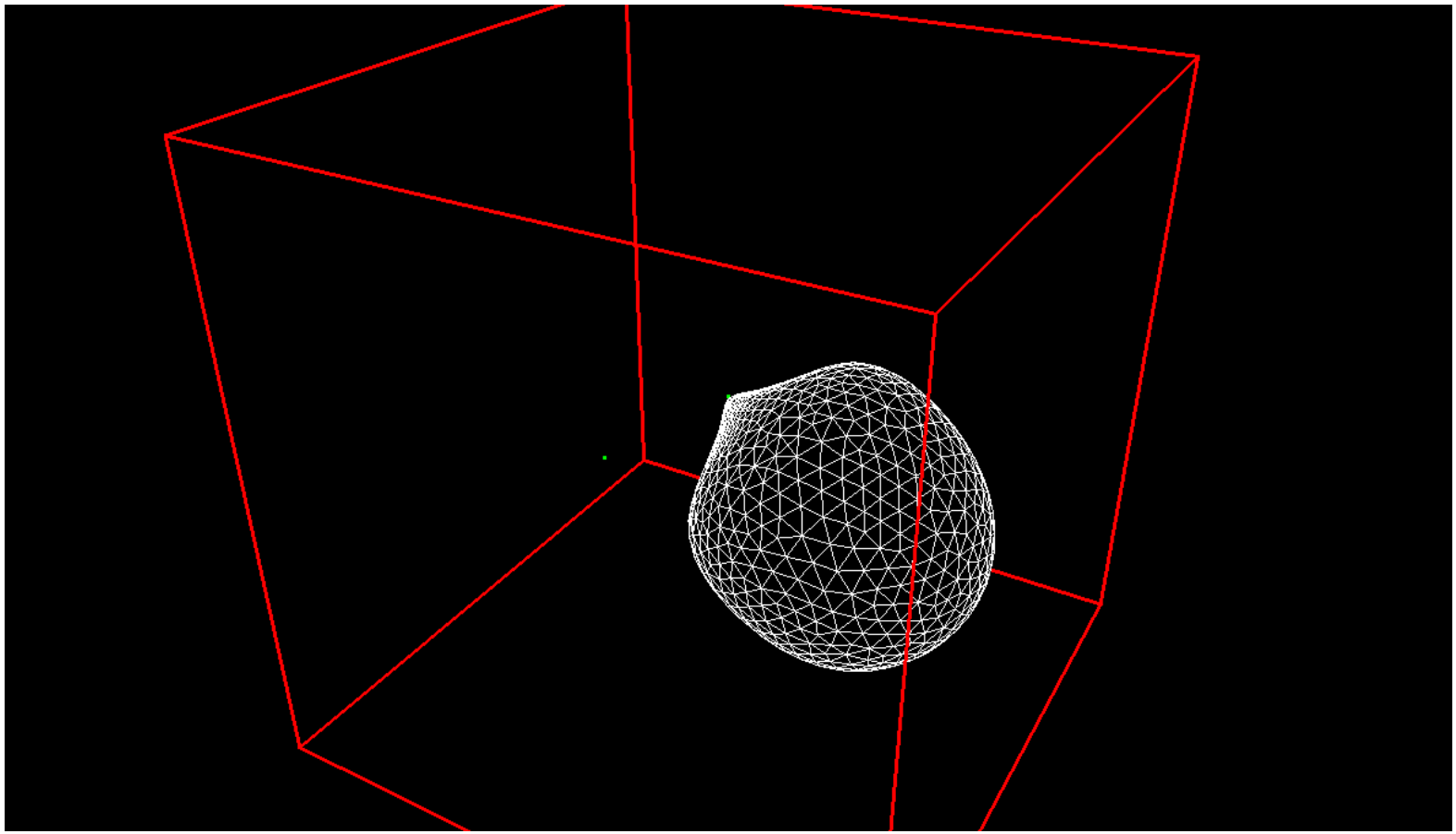
Introduction : exemple de la molécule H_2O (itération 20)



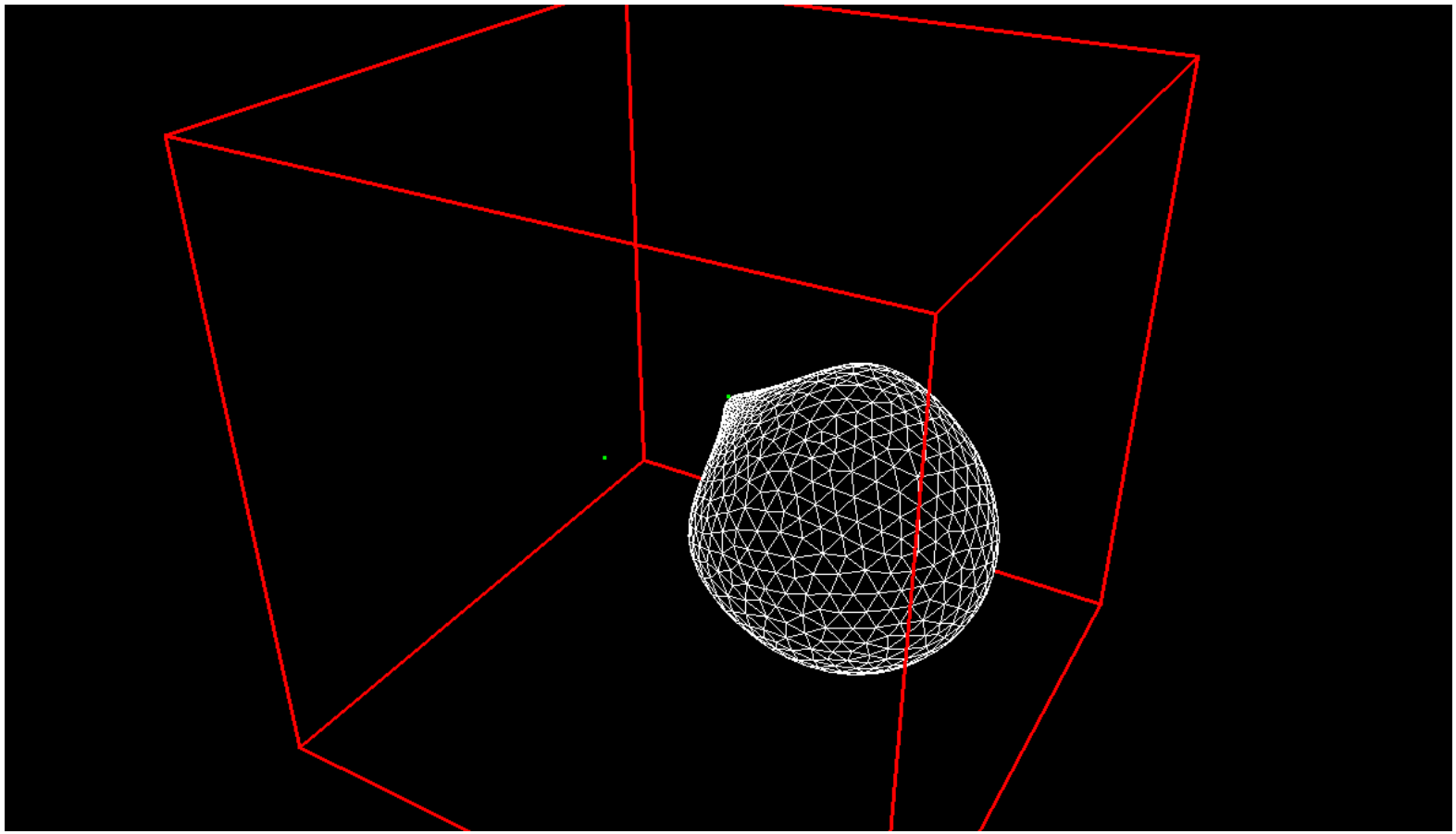
Introduction : exemple de la molécule H_2O (itération 30)



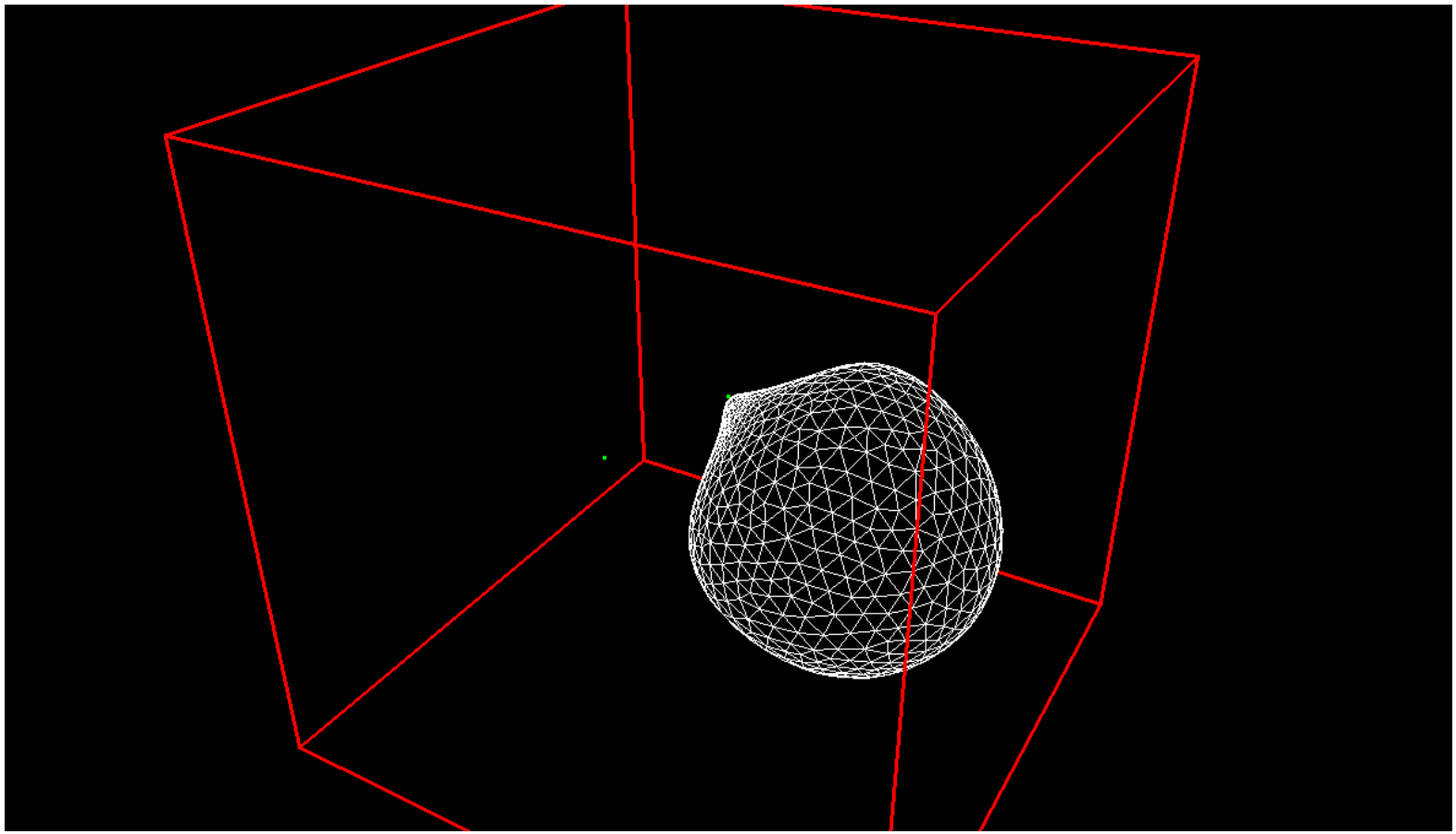
Introduction : exemple de la molécule H_2O (itération 40)



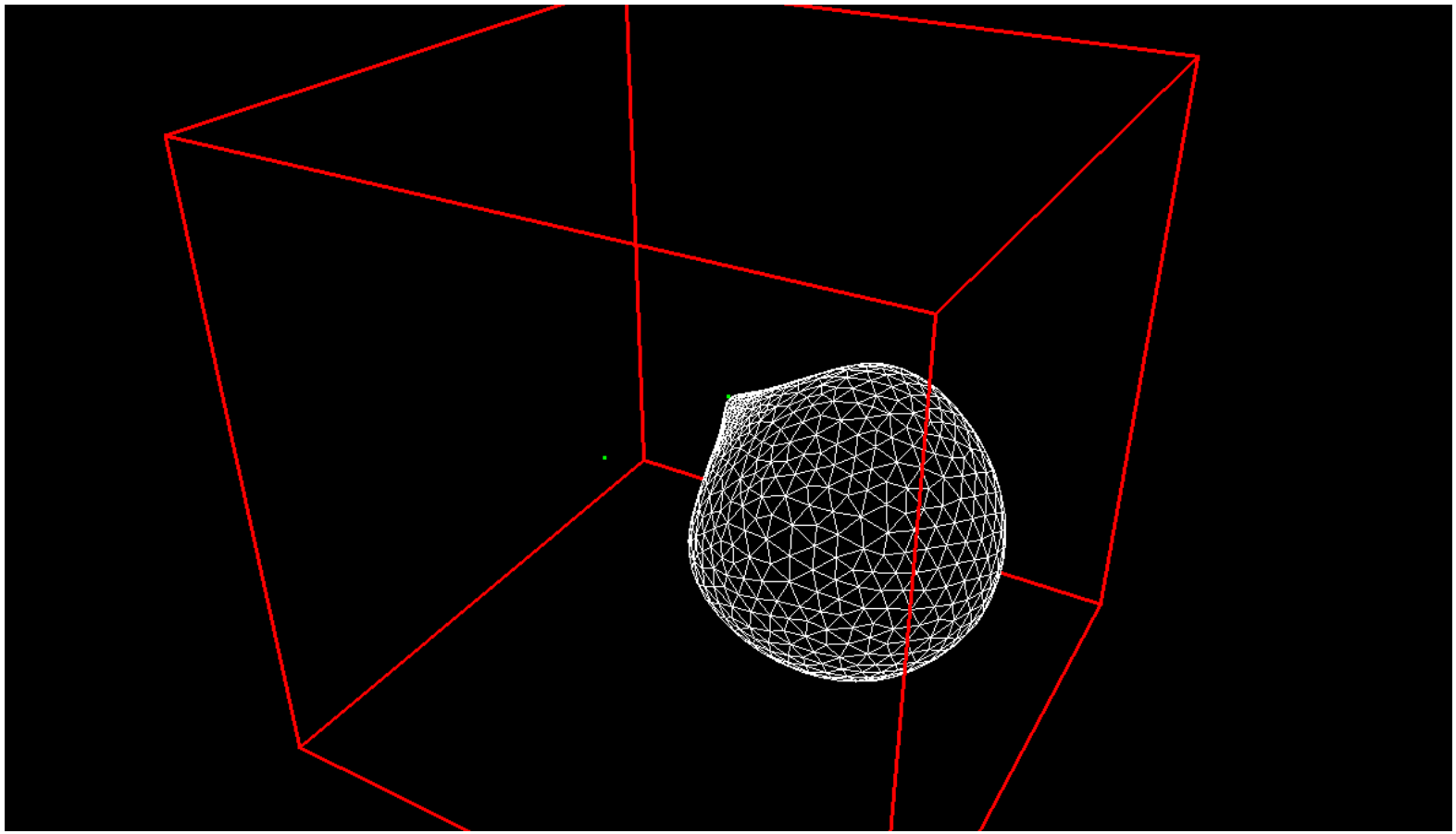
Introduction : exemple de la molécule H_2O (itération 50)



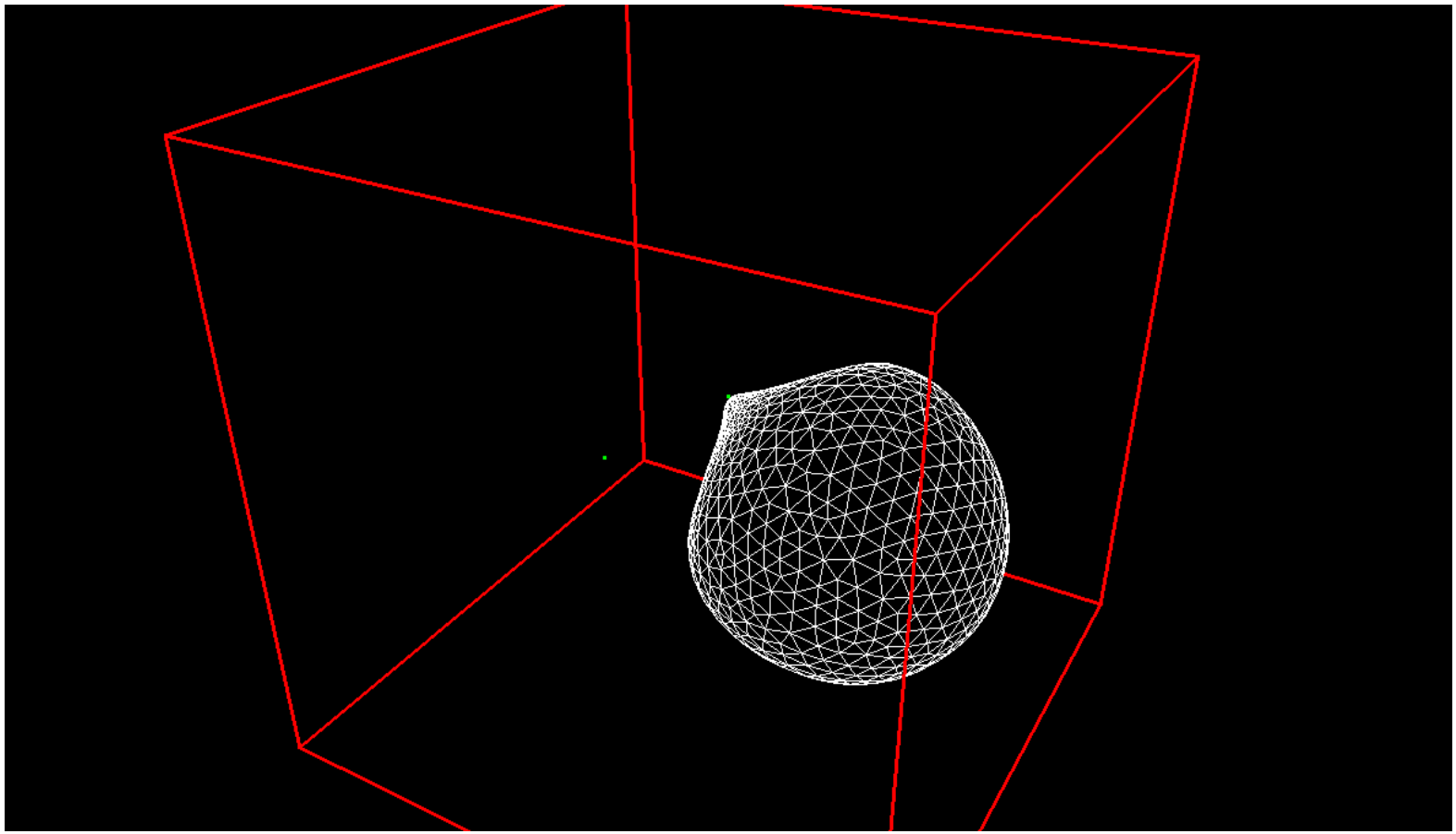
Introduction : exemple de la molécule H_2O (itération 60)



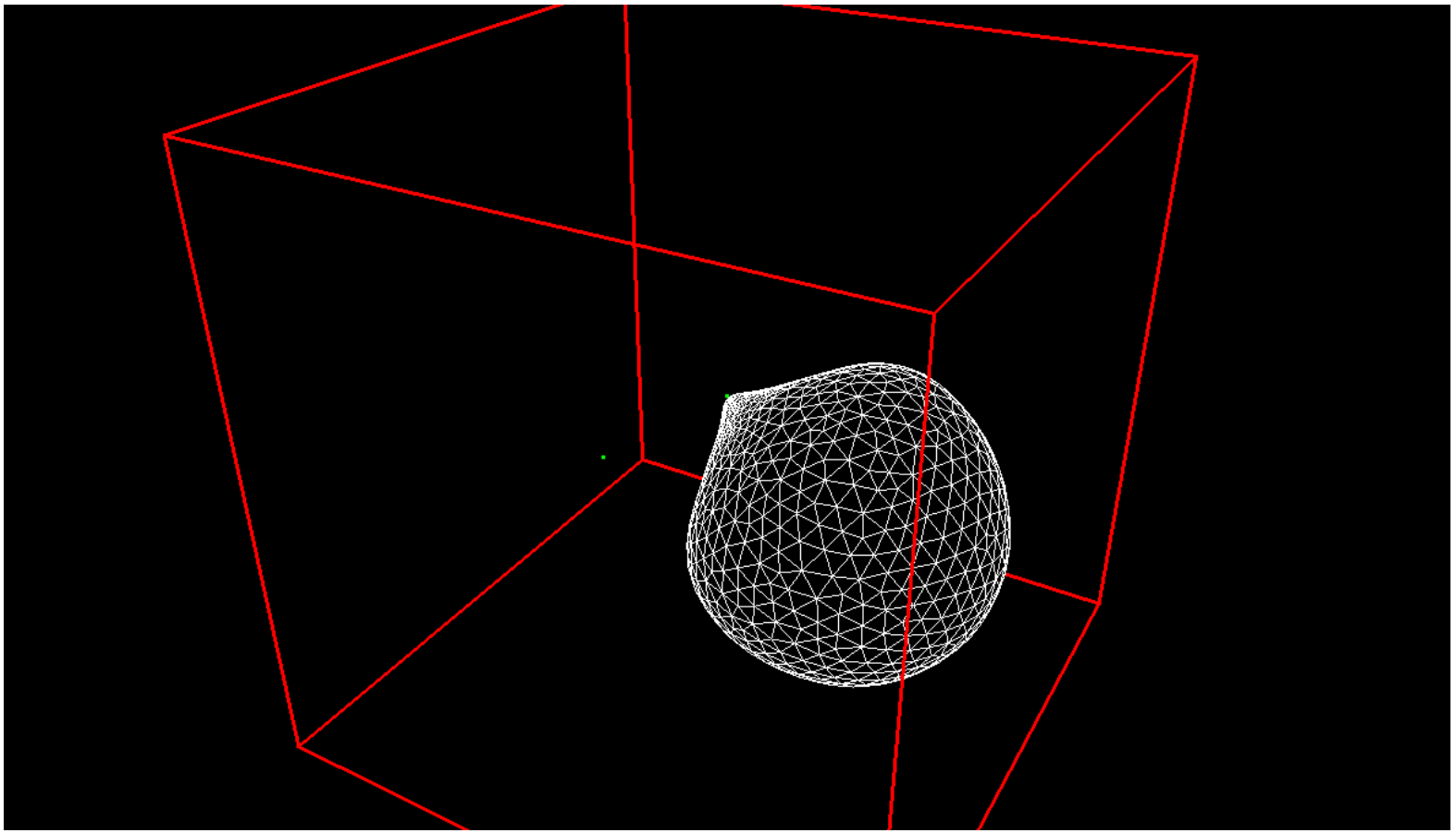
Introduction : exemple de la molécule H_2O (itération 70)



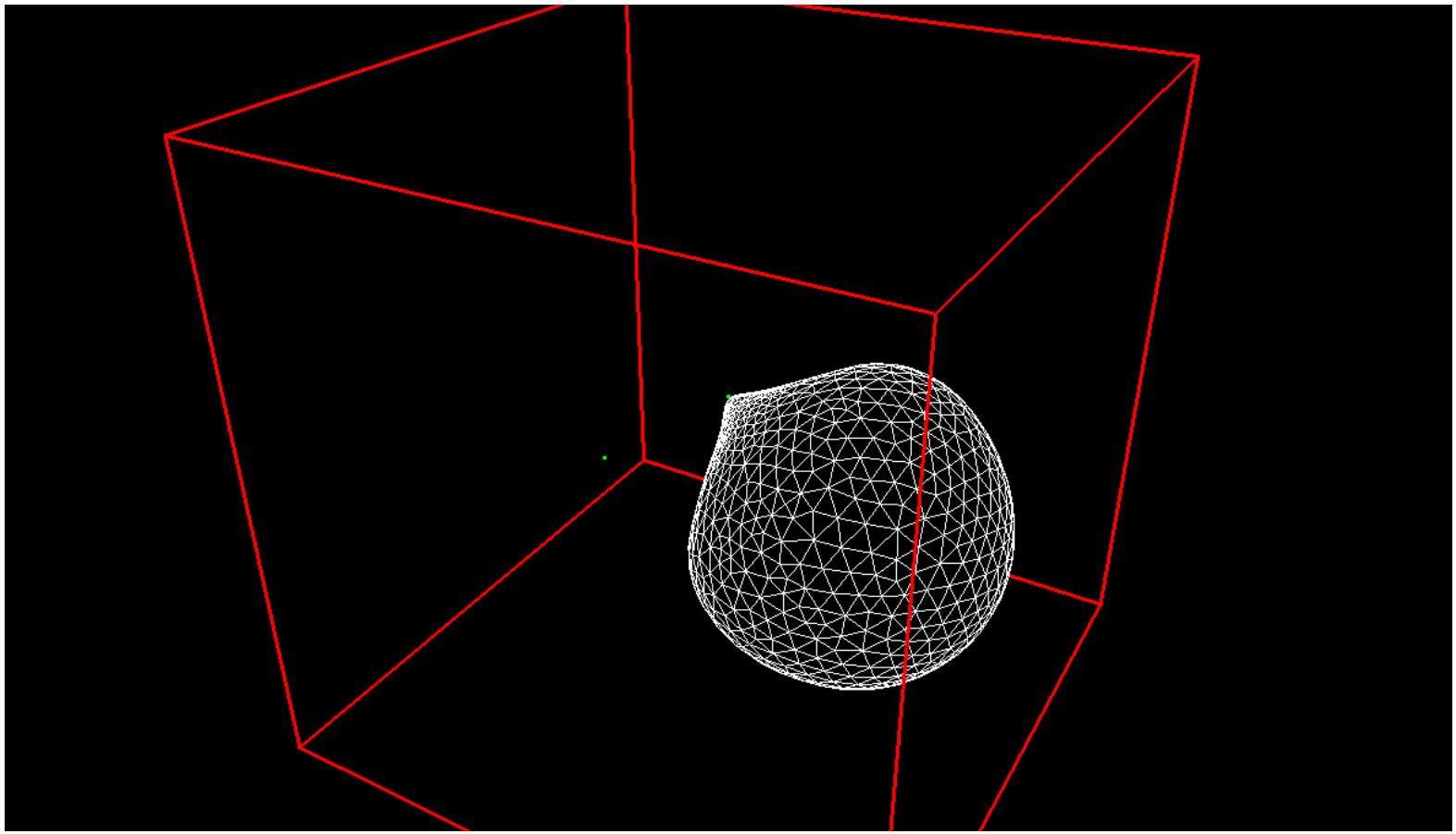
Introduction : exemple de la molécule H_2O (itération 80)



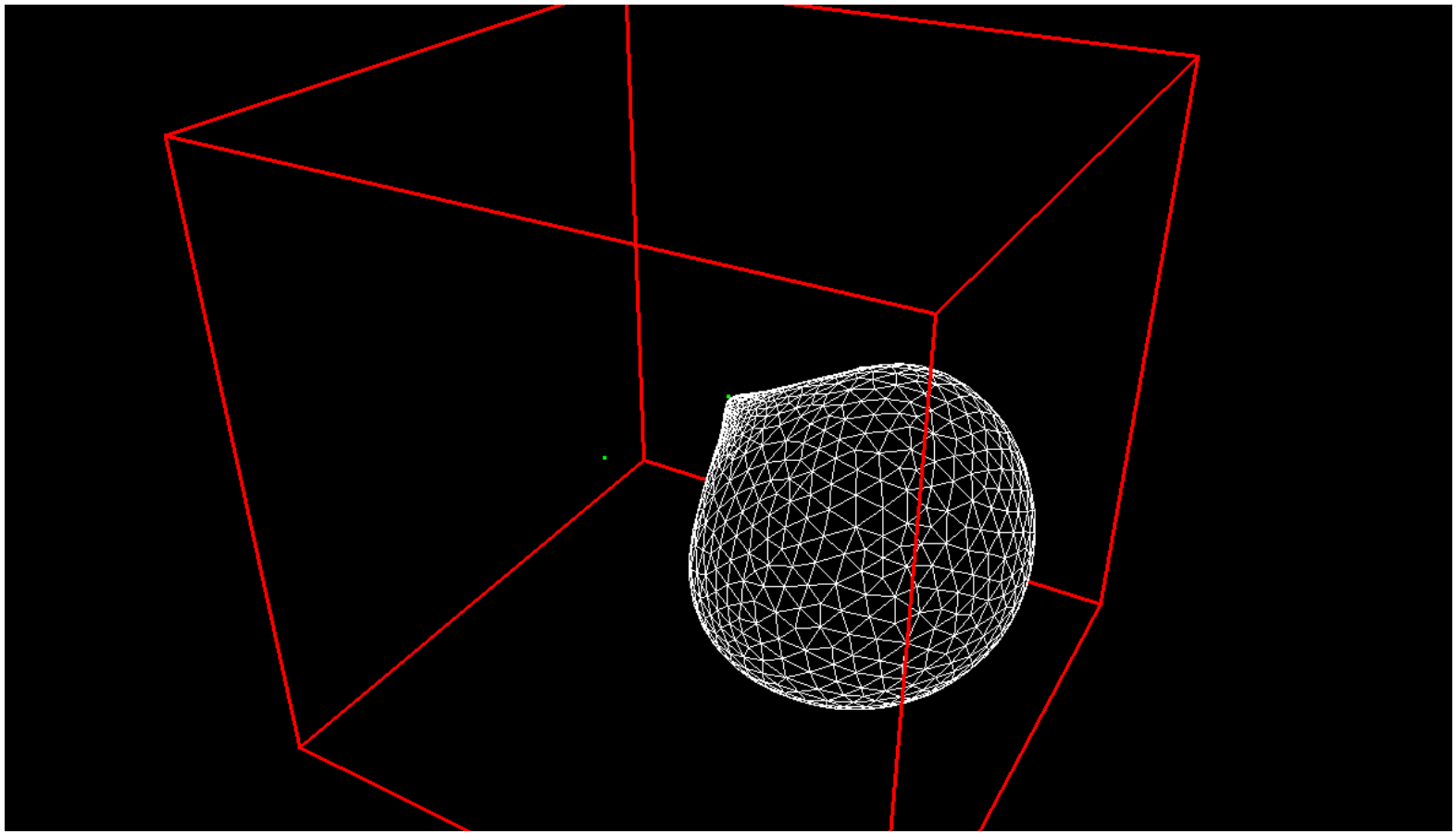
Introduction : exemple de la molécule H_2O (itération 90)



Introduction : exemple de la molécule H_2O (itération 100)

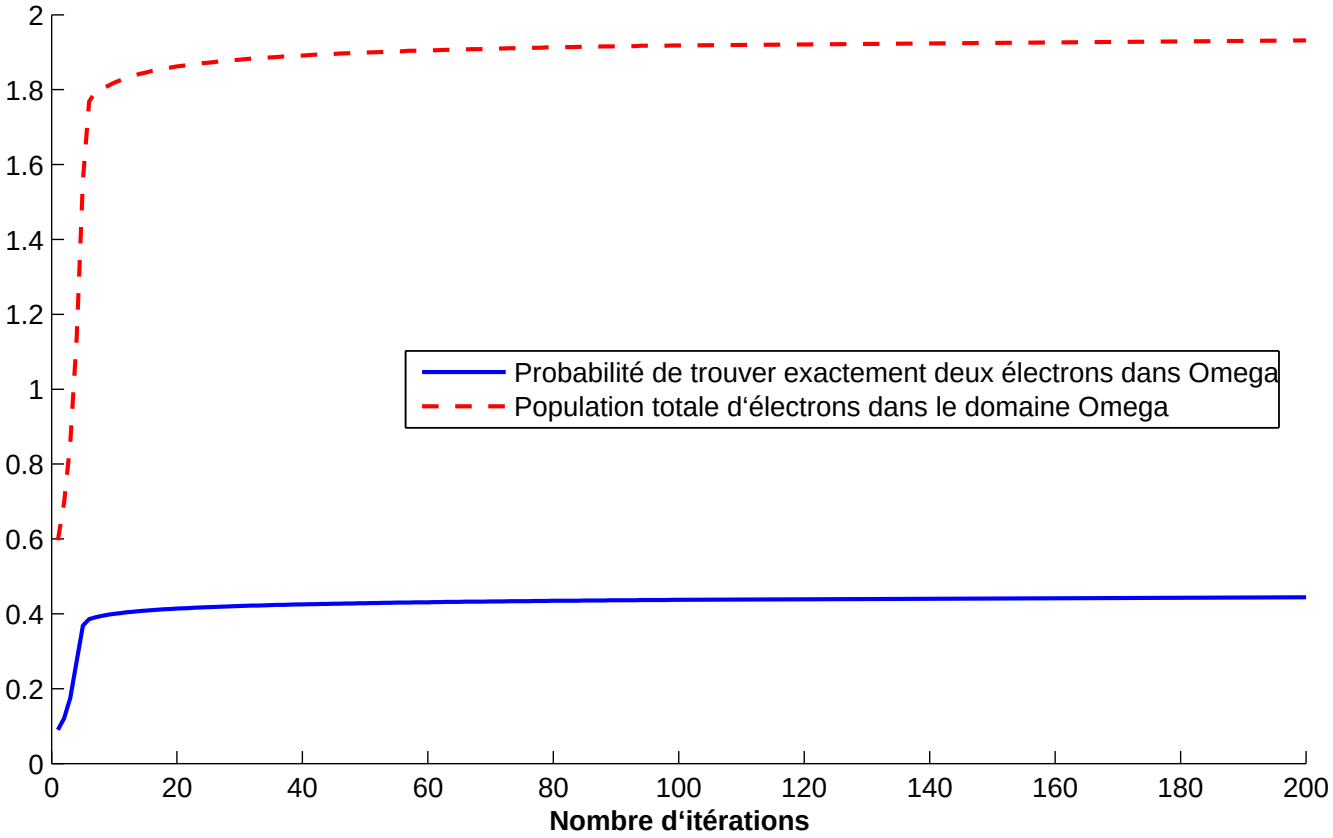


Introduction : exemple de la molécule H_2O (itération 200)



Introduction : exemple de la molécule H_2O

Temps de calcul : 30 sec. par itération soit 5 min. pour 10 itérations.



Introduction : comment se calculent de telles probabilités ?

Comme la fonction d'onde est antisymétrique, une formulation simple est donnée par l'approximation Hartree-Fock et un déterminant de Slater :

$$\Psi\left[\begin{pmatrix}\mathbf{x}_1\\ \sigma_1\end{pmatrix},\ldots,\begin{pmatrix}\mathbf{x}_n\\ \sigma_n\end{pmatrix}\right]=\det\left[\begin{pmatrix}\phi_1\left(\mathbf{x}_1,\sigma_1\right)\\ \vdots\\ \phi_n\left(\mathbf{x}_1,\sigma_1\right)\end{pmatrix},\ldots,\begin{pmatrix}\phi_1\left(\mathbf{x}_n,\sigma_n\right)\\ \vdots\\ \phi_n\left(\mathbf{x}_n,\sigma_n\right)\end{pmatrix}\right],$$

où les $\phi_1,\ldots,\phi_n:\mathbb{R}^3\times\left\{-\frac{1}{2},\frac{1}{2}\right\}\to\mathbb{C}$ désignent les orbitales moléculaires.

Introduction : comment se calculent de telles probabilités ?

Comme la fonction d'onde est antisymétrique, une formulation simple est donnée par l'approximation Hartree-Fock et un déterminant de Slater :

$$\Psi\left[\begin{pmatrix}\mathbf{x}_1\\ \sigma_1\end{pmatrix},\ldots,\begin{pmatrix}\mathbf{x}_n\\ \sigma_n\end{pmatrix}\right]=\det\left[\begin{pmatrix}\phi_1\left(\mathbf{x}_1,\sigma_1\right)\\ \vdots\\ \phi_n\left(\mathbf{x}_1,\sigma_1\right)\end{pmatrix},\ldots,\begin{pmatrix}\phi_1\left(\mathbf{x}_n,\sigma_n\right)\\ \vdots\\ \phi_n\left(\mathbf{x}_n,\sigma_n\right)\end{pmatrix}\right],$$

où les $\phi_1,\ldots,\phi_n:\mathbb{R}^3\times\left\{-\frac{1}{2},\frac{1}{2}\right\}\to\mathbb{C}$ désignent les orbitales moléculaires. Chacune est associée à un spin particulier (up or down) :

$$\begin{cases}\phi\left(\mathbf{x},\frac{1}{2}\right) &= \phi^{\uparrow}\left(\mathbf{x}\right)\\ \phi\left(\mathbf{x},-\frac{1}{2}\right) &= 0\end{cases}\quad\text{ou}\quad\begin{cases}\phi\left(\mathbf{x},\frac{1}{2}\right) &= 0\\ \phi\left(\mathbf{x},-\frac{1}{2}\right) &= \phi^{\downarrow}\left(\mathbf{x}\right)\end{cases},$$

Introduction : comment se calculent de telles probabilités ?

Comme la fonction d'onde est antisymétrique, une formulation simple est donnée par l'approximation Hartree-Fock et un déterminant de Slater :

$$\Psi\left[\begin{pmatrix}\mathbf{x}_1\\ \sigma_1\end{pmatrix},\ldots,\begin{pmatrix}\mathbf{x}_n\\ \sigma_n\end{pmatrix}\right]=\det\left[\begin{pmatrix}\phi_1\left(\mathbf{x}_1,\sigma_1\right)\\ \vdots\\ \phi_n\left(\mathbf{x}_1,\sigma_1\right)\end{pmatrix},\ldots,\begin{pmatrix}\phi_1\left(\mathbf{x}_n,\sigma_n\right)\\ \vdots\\ \phi_n\left(\mathbf{x}_n,\sigma_n\right)\end{pmatrix}\right],$$

où les $\phi_1,\ldots,\phi_n:\mathbb{R}^3\times\left\{-\frac{1}{2},\frac{1}{2}\right\}\to\mathbb{C}$ désignent les orbitales moléculaires. Chacune est associée à un spin particulier (up or down) :

$$\begin{cases}\phi\left(\mathbf{x},\frac{1}{2}\right) &= \phi^{\uparrow}\left(\mathbf{x}\right)\\ \phi\left(\mathbf{x},-\frac{1}{2}\right) &= 0\end{cases}\quad\text{ou}\quad\begin{cases}\phi\left(\mathbf{x},\frac{1}{2}\right) &= 0\\ \phi\left(\mathbf{x},-\frac{1}{2}\right) &= \phi^{\downarrow}\left(\mathbf{x}\right)\end{cases},$$

et se décompose en une combinaison linéaire de primitives gaussiennes :

$$\phi^{\uparrow\downarrow}\left(\mathbf{x}\right)=\sum_i c_i\left(x-x_i\right)^{p_i^x}\left(y-y_i\right)^{p_i^y}\left(z-z_i\right)^{p_i^z}e^{-a_i\left[(x-x_i)^2+(y-y_i)^2+(z-z_i)^2\right]},$$

où les c_i désignent des coefficients, les $a_i>0$ des exposants, les (x_i,y_i,z_i) des centres de noyaux, et les (p_i^x,p_i^y,p_i^z) sont liés à des types prédéfinis.

Introduction : comment se calculent de telles probabilités ?

Soit la matrice de recoupement $S(\Omega)_{ij} := \sum_{\sigma \in \{-\frac{1}{2}, \frac{1}{2}\}} \int_{\Omega} \phi_i(\mathbf{x}, \sigma) \phi_j(\mathbf{x}, \sigma) \, \mathrm{d}\mathbf{x}$.

Proposition

$$p_\nu(\Omega) = \sum_{\substack{I_\nu \subset \{1,\dots,n\} \\ \text{card } I_\nu = \nu}} \det S_{I_\nu}(\Omega) \quad \text{avec } S_{I_\nu}(\Omega)_{ij} = \begin{cases} S(\Omega)_{ij} & \text{si } i \in I_\nu \\ S(\mathbb{R}^3 \backslash \Omega)_{ij} & \text{si } i \notin I_\nu . \end{cases}$$

Cette formule est inexploitable en pratique (complexité en $O\binom{n}{\nu}$) mais

- $S(\Omega)$ est une $(n \times n)$-matrice symétrique donc diagonalisable et
- $S(\mathbb{R}^3) = I_n$ *i.e.* les orbitales forment une base orthonormée sur $\mathbb{R}^3$.

Introduction : comment se calculent de telles probabilités ?

Soit la matrice de recoupement $S(\Omega)_{ij} := \sum_{\sigma \in \{-\frac{1}{2}, \frac{1}{2}\}} \int_\Omega \phi_i(\mathbf{x}, \sigma)\phi_j(\mathbf{x}, \sigma)\, \mathrm{d}\mathbf{x}$.

Proposition

$$p_\nu(\Omega) = \sum_{\substack{I_\nu \subset \{1,\dots,n\} \\ \text{card } I_\nu = \nu}} \det S_{I_\nu}(\Omega) \quad \text{avec } S_{I_\nu}(\Omega)_{ij} = \begin{cases} S(\Omega)_{ij} & \text{si } i \in I_\nu \\ S(\mathbb{R}^3 \backslash \Omega)_{ij} & \text{si } i \notin I_\nu. \end{cases}$$

Cette formule est inexploitable en pratique (complexité en $O\binom{n}{\nu}$) mais

- $S(\Omega)$ est une $(n \times n)$-matrice symétrique donc diagonalisable et
- $S(\mathbb{R}^3) = I_n$ *i.e.* les orbitales forment une base orthonormée sur $\mathbb{R}^3$.

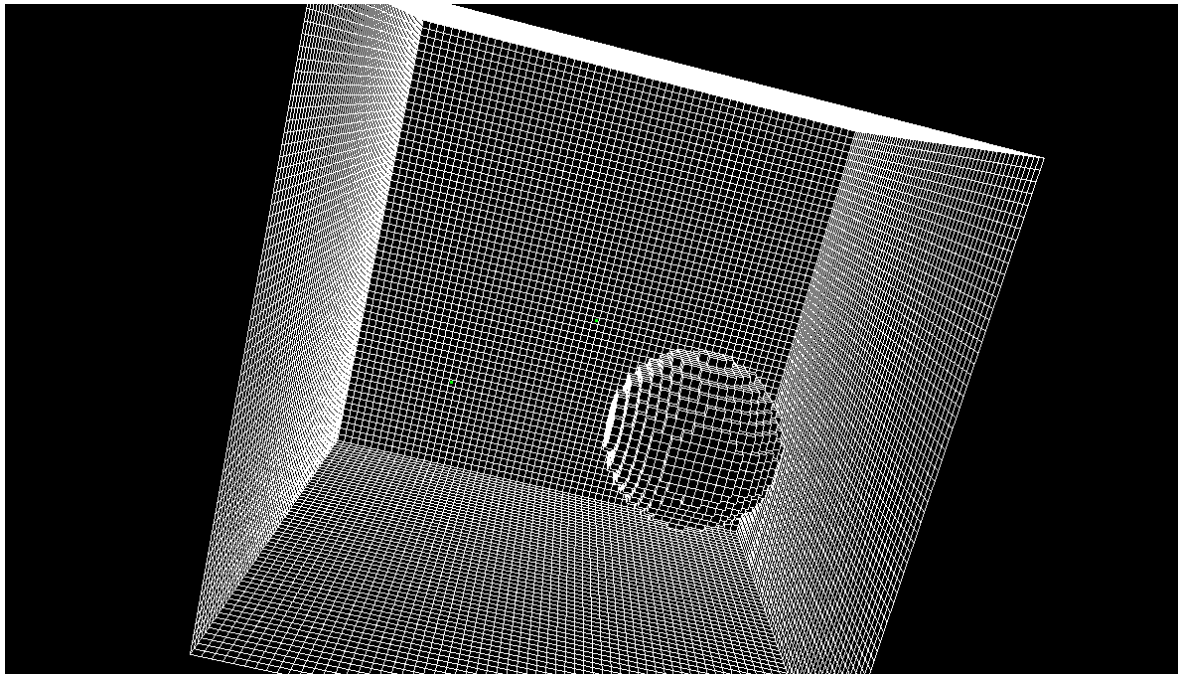
Proposition (2004, Cancès, Keriven, Lodier, Savin)

$$\forall t \in \mathbb{R}, \sum_{\nu=0}^{n} p_\nu(\Omega)\, t^\nu = \det\left[t\, S\left(\Omega\right) + S\left(\mathbb{R}^3 \backslash \Omega\right)\right] = \prod_{i=1}^{n} \left[t\, \lambda_i(\Omega) + 1 - \lambda_i(\Omega)\right].$$

C'est une évaluation de p_ν qui se fait avec une complexité en $O(n^2)$.

Introduction : un premier algorithme simple d'optimisation

Question : quel mode de représentation numérique pour un domaine ?



- Grille cubique uniforme : ensemble de cubes identiques repérés par leur centre, labellisés un s'ils appartiennent au domaine, sinon zéro.
- Maillage : ensemble de sommets, d'éléments volumiques pour décrire l'intérieur/extérieur du domaine et surfaciques pour décrire son bord.

Introduction : un premier algorithme simple d'optimisation

Une première version du logiciel MPD écrite en C (5 fichiers, 8000 lignes) a été mise en ligne à l'adresse : *https ://github.com/ISCDtoolbox/MPD* où chaque fichier contient une fonction principale du même nom.

- *loadChemistry.c* s'occupe de charger les données chimiques initiales.
- *meshCube.c* se charge d'initialiser la boîte numérique de calcul.
- *adaptCube.c* permet d'initialiser le domaine initial dans la boîte.
- *shapeDerivative.c* s'efforce ici de calculer la probabilité.
- *main.c* contient l'algorithme d'optimisation de formes.

Introduction : un premier algorithme simple d'optimisation

Une première version du logiciel MPD écrite en C (5 fichiers, 8000 lignes) a été mise en ligne à l'adresse : *https ://github.com/ISCDtoolbox/MPD* où chaque fichier contient une fonction principale du même nom.

- *loadChemistry.c* s'occupe de charger les données chimiques initiales.
- *meshCube.c* se charge d'initialiser la boîte numérique de calcul.
- *adaptCube.c* permet d'initialiser le domaine initial dans la boîte.
- *shapeDerivative.c* s'efforce ici de calculer la probabilité.
- *main.c* contient l'algorithme d'optimisation de formes.

Après avoir chargé les données chimiques, initialisé la boîte de calcul et le domaine initial Ω_0, pour tout $k = 0, \ldots, K$, étant donné Ω_k :

- on choisit aléatoirement un carré C_k sur le bord de Ω_k et on cherche le cube extérieur C_k^{ext} et intérieur C_k^{int} ayant C_k en commun ;
- si $p_\nu(\Omega_k \sqcup C_k^{\text{ext}}) > p_\nu(\Omega_k)$, alors on définit $\Omega_{k+1} := \Omega_k \sqcup C_k^{\text{ext}}$;
- sinon on voit si $p_\nu(\Omega_k \backslash C_k^{\text{int}}) > p_\nu(\Omega_k)$ et alors $\Omega_{k+1} := \Omega_k \backslash C_k^{\text{int}}$;
- et si ce n'est toujours pas le cas, alors on ne fait rien : $\Omega_{k+1} := \Omega_k$.

loadChemistry.c : quel format pour les données chimiques ?

Le format de base utilisé est celui fourni par le logiciel Gaussian : **.wfn*.

loadChemistry.c : quel format pour les données chimiques ?

Le format de base utilisé est celui fourni par le logiciel Gaussian : **.wfn*.

Commentaire
GAUSSIAN n_o MOL ORBITALS n_p PRIMITIVES n_n NUCLEI
Be 1 (CENTRE 1) x_1 (double) y_1 z_1 CHARGE $= c_{h1} \cdots$

CENTER ASSIGNMENTS c_{a1} (int) $c_{a2} \cdots$
CENTER ASSIGNMENTS $c_{a21} \cdots$

TYPE ASSIGNMENTS t_{a1} (int) $t_{a2} \cdots$
TYPE ASSIGNMENTS $t_{a21} \cdots$

EXPONENTS e_1 (double) $0.1234567\mathrm{D}{+}01 \cdots$
EXPONENTS $e_6 \cdots$

MO 1 MO 0.0 OCC NO $= o_{n1}$ (double) ORB. ENERGY $= o_{e1}$
c_1^1 (double) $c_2^1 \cdots$
c_6^1 (double) $\cdots$
$\cdots$

END DATA Commentaire

loadChemistry.c : quel format pour les données chimiques ?

Le format de base utilisé est celui fourni par le logiciel Gaussian : **.wfn*.

> Commentaire
> GAUSSIAN n_o MOL ORBITALS n_p PRIMITIVES n_n NUCLEI
> Be 1 (CENTRE 1) x_1 (double) y_1 z_1 CHARGE $= c_{h1}$ $\cdots$
>
> CENTER ASSIGNMENTS c_{a1} (int) c_{a2} $\cdots$
> CENTER ASSIGNMENTS c_{a21} $\cdots$
>
> TYPE ASSIGNMENTS t_{a1} (int) t_{a2} $\cdots$
> TYPE ASSIGNMENTS t_{a21} $\cdots$
>
> EXPONENTS e_1 (double) 0.1234567D+01 $\cdots$
> EXPONENTS e_6 $\cdots$
>
> MO 1 MO 0.0 OCC NO $= o_{n1}$ (double) ORB. ENERGY $= o_{e1}$
> c_1^1 (double) c_2^1 $\cdots$
> c_6^1 (double) $\cdots$
> $\cdots$
>
> END DATA Commentaire

Inconvénient : bien que largement utilisé, il ne se lit pas très rapidement.

loadChemistry.c : quel format pour les données chimiques ?

On utilise aussi un format développé dans le cadre du projet MPD : **.txt*.

loadChemistry.c : quel format pour les données chimiques ?

On utilise aussi un format développé dans le cadre du projet MPD : **.txt*.

Ligne 1 $\cdots$	MolecularOrbitals n_o (int)
Ligne 3 $\cdots$	Primitives n_p (int)
Ligne 5	Nuclei
Ligne 6	n_n (int)
Ligne 7 $\cdots$	x_1 (double) y_1 (double) z_1 (double)
Ligne $8 + n_n$	Coefficient Exponent Center Type
Ligne $9 + n_n$	1
Ligne $10 + n_n$	Spin ± 1
Ligne $11 + n_n$ $\cdots$	c_1^1 (double) e_1 (double) c_{a1} (int) t_{a1} (int)
Ligne $11 + n_n + n_p$	
Ligne $12 + n_n + n_p$	2
Ligne $13 + n_n + n_p$	Spin ± 1
Ligne $14 + n_n + n_p$ $\cdots$	c_1^2 (double) e_1 (double) c_{a1} (int) t_{a1} (int)
Dernière ligne	End Commentaire

loadChemistry.c : quel format pour les données chimiques ?

On utilise aussi un format développé dans le cadre du projet MPD : **.txt*.

Ligne 1 $\cdots$	MolecularOrbitals n_o (int)
Ligne 3 $\cdots$	Primitives n_p (int)
Ligne 5	Nuclei
Ligne 6	n_n (int)
Ligne 7 $\cdots$	x_1 (double) y_1 (double) z_1 (double)
Ligne $8 + n_n$	Coefficient Exponent Center Type
Ligne $9 + n_n$	1
Ligne $10 + n_n$	Spin ± 1
Ligne $11 + n_n$ $\cdots$	c_1^1 (double) e_1 (double) c_{a1} (int) t_{a1} (int)
Ligne $11 + n_n + n_p$	
Ligne $12 + n_n + n_p$	2
Ligne $13 + n_n + n_p$	Spin ± 1
Ligne $14 + n_n + n_p$ $\cdots$	c_1^2 (double) e_1 (double) c_{a1} (int) t_{a1} (int)
Dernière ligne	End Commentaire

Avantage : il est plus facile de charger les données à partir de ce fichier.

loadChemistry.c : quel format pour les données chimiques ?

Dans le fichier *main.h*, on a défini des structures pour les données.

```
typedef struct {
    double x;
    double y;
    double z;
} Nucleus;
```

```
typedef struct {
    int spin;
    double* coeff;
    double* exp;
    int* nucl;
    int* type;
} MolecularOrbital;
```

```
typedef struct {
    int nmorb;
    MolecularOrbital* pmorb;
    int nnucl;
    Nucleus* pnucl;
    int ngauss;
} ChemicalSystem;
```

loadChemistry.c : quel format pour les données chimiques ?

Dans le fichier *main.h*, on a défini des structures pour les données.

```
typedef struct {
    double x;
    double y;
    double z;
} Nucleus;
```

```
typedef struct {
    int spin;
    double* coeff;
    double* exp;
    int* nucl;
    int* type;
} MolecularOrbital;
```

```
typedef struct {
    int nmorb;
    MolecularOrbital* pmorb;
    int nnucl;
    Nucleus* pnucl;
    int ngauss;
} ChemicalSystem;
```

Dans *loadChemistry.h*, on écrit ORB_SPIN 1 si on veut dupliquer les orbitales doublement occupées, sinon on doit mettre ORB_SPIN 0.

int ***loadChemistry*** (char* fileLocation, ChemicalSystem* pChemicalSystem, int nuElectrons) {

getChemicalFormat (fileLocation)

**.txt* ↙ ↘ **.wfn*

readTxtFileAndAllocateChemicalSystem
(fileLocation, pChemicalSystem)

readAndConvertWfnFile
(fileLocation, 'D', 'e')
readWfnFileAndAllocateChemicalSystem
(fileLocation, pChemicalSystem)
readAndConvertWfnFile
(fileLocation, 'e', 'D')
writingChemicalFile
("ChemicalOut.txt", pChemicalSystem)

check nuElectrons. }

meshCube.c : quel format utiliser pour un domaine ?

Pour une grille cubique uniforme, le format **.cube* est utilisé en chimie.

Ligne 1	Commentaire				
Ligne 2	Commentaire				
Ligne 3	n_a (int)	$x_{\min}$ (double)	$y_{\min}$	$z_{\min}$	
Ligne 4	n_x	Δx (double)	0.0	0.0	
Ligne 5	n_y	0.0	Δy	0.0	
Ligne 6	n_z	0.0	0.0	Δz	
Ligne 7	c_{a1} (int)	c_{a1} (double)	x_1	y_1	z_1
...					
Ligne 7 + n_a	l_{p1} (double)	l_{p2}	...		
Ligne 8 + n_a	l_{p7}	...			
...					

meshCube.c : quel format utiliser pour un domaine ?

Pour une grille cubique uniforme, le format **.cube* est utilisé en chimie.

Ligne 1	Commentaire				
Ligne 2	Commentaire				
Ligne 3	n_a (int)	$x_{\min}$ (double)	$y_{\min}$	$z_{\min}$	
Ligne 4	n_x	Δx (double)	0.0	0.0	
Ligne 5	n_y	0.0	Δy	0.0	
Ligne 6	n_z	0.0	0.0	Δz	
Ligne 7	c_{a1} (int)	c_{a1} (double)	x_1	y_1	z_1
...					
Ligne 7 + n_a	l_{p1} (double)	l_{p2}	...		
Ligne 8 + n_a	l_{p7}	...			
...					

Avantage : le format est très utilisé et il est très concis.

Inconvénient : c'est long d'extraire rapidement la géométrie du fichier.

On est en train d'interfacer ce format avec le programme. Cela demande notamment de stocker la charge des atomes dans la structure Nucleus.

meshCube.c : quel format utiliser pour un domaine ?

On a fait le choix d'utiliser un maillage et donc le format **.mesh* associé.

Ligne 1	MeshVersionFormatted 2
Ligne 3	Dimension 3
Ligne 5	Vertices
Ligne 6	n_v (int)
Ligne 7	x_1 (double) y_1 z_1 l_{p1} (int)
...	
Ligne $8 + n_v$	Quadrilaterals
Ligne $9 + n_v$	n_q
Ligne $10 + n_v$	p^1_{q1} (int) p^2_{q1} p^3_{q1} p^4_{q1} l_{q1} (int)
...	
Ligne $11 + n_v + n_q$	Hexahedra
Ligne $12 + n_v + n_q$	n_h
Ligne $13 + n_v + n_q$	p^1_{h1} (int) p^2_{h1} $\cdots$ p^8_{h1} l_{h1} (int)
...	
Ligne $14 + n_v + n_q + n_h$	End
...	Commentaires

meshCube.c : quel format utiliser pour un domaine ?

Dans le fichier *main.h*, on a défini des structures pour le maillage.

```
typedef struct {
    double x;
    double y;
    double z;
    int label;
    double value;
} Point;

typedef struct {
    int p1;
    int p2;
    int p3;
    int p4;
    int label;
} Quadrilateral;
```

```
typedef struct {
    int p1;
    int p2;
    int p3;
    int p4;
    int p5;
    int p6;
    int p7;
    int p8;
    int label;
} Hexahedra;

typedef struct {
    int quad;
    int hexin;
    int hexout;
} Adjacency;
```

```
typedef struct {
    int nver;
    Point* pver;
    int nqua;
    Quadrilateral* pqua;
    int nhex;
    Hexahedra* phex;
    int nadj;
    Adjacency* padj;
} Mesh;
```

meshCube.c : comment initialiser la boîte de calcul ?

Dans *loadChemistry.h* et *main.h* (commun à tous les **.h*), la boîte de calcul initiale est définie grâce à des constantes de préprocesseur :

- X_MIN, X_MAX, Y_MIN, Y_MAX, Z_MIN, Z_MAX (double) qui définissent les dimension de la boîte et vérifie *_MIN $<$ *_MAX .
- N_X, N_Y, N_Z (int) qui fournit le nombre de points et vérifie N_* $>$ 2.
- DELTA_X, DELTA_Y, DELTA_Z (double) qui gère la discrétisation et vérifie DELTA_*==(*_MAX-*_MIN)/(double)(N_*-1) (à 10^{-10} près).

meshCube.c : comment initialiser la boîte de calcul ?

Dans *loadChemistry.h* et *main.h* (commun à tous les **.h*), la boîte de calcul initiale est définie grâce à des constantes de préprocesseur :

- X_MIN, X_MAX, Y_MIN, Y_MAX, Z_MIN, Z_MAX (double) qui définissent les dimension de la boîte et vérifie *_MIN $<$ *_MAX .
- N_X, N_Y, N_Z (int) qui fournit le nombre de points et vérifie N_* $>$ 2.
- DELTA_X, DELTA_Y, DELTA_Z (double) qui gère la discrétisation et vérifie DELTA_*==(*_MAX-*_MIN)/(double)(N_*-1) (à 10^{-10} près).

int ***meshCube*** (char* fileLocation, Mesh* pMesh) {

initialFileExists (fileLocation)

yes ↙ ***copyFileLocation*** (fileLocation, "cube.mesh")

↘ no ***initializeCubeDiscretization*** (pMesh) → no

↓ yes ***allocateInitialMeshMemory*** (pMesh)

discretizeCube (pMesh)

writingMeshFile ("cube.mesh", pMesh)

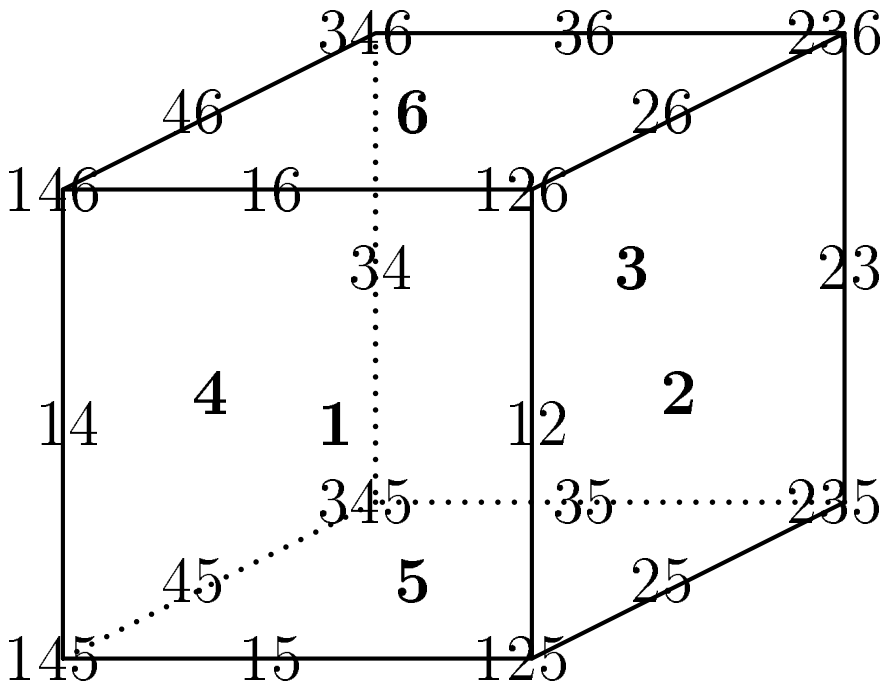
}

Si aucun maillage n'est donné, on doit confirmer la création du fichier *cube.mesh*, sinon une copie du maillage initial est faite dans *cube.mesh*.

meshCube.c : comment initialiser la boîte de calcul ?

Dans la fonction *discretizeCube*, les labels sont initialisés.

- pMesh->pver[i].label=0 si le point n'appartient pas au bord de la boîte ;
- sinon pMesh->pver[i].label est donnée par le retour de la fonction *labelPoint* pour les points appartenant au bord de la boîte ;



- on procède de la même manière pour les carrés (pMesh->pqua[i].label).

adaptCube.c : le concept des lignes de niveaux (level-set)

Définition : Etant donné un domaine $\Omega \subset \mathbb{R}^d$, $d \geqslant 2$, on dit qu'une application continue $\omega : \mathbb{R}^d \mapsto \mathbb{R}$ est une fonction *level-set* si elle vérifie :

$$\Omega = \left\{ \mathbf{x} \in \mathbb{R}^d, \omega\left(\mathbf{x}\right) < 0 \right\} \quad \text{et} \quad \partial\Omega = \left\{ \mathbf{x} \in \mathbb{R}^d, \omega\left(\mathbf{x}\right) = 0 \right\}$$

adaptCube.c : le concept des lignes de niveaux (level-set)

Définition : Etant donné un domaine $\Omega \subset \mathbb{R}^d$, $d \geqslant 2$, on dit qu'une application continue $\omega : \mathbb{R}^d \mapsto \mathbb{R}$ est une fonction *level-set* si elle vérifie :

$$\Omega = \left\{ \mathbf{x} \in \mathbb{R}^d, \omega\left(\mathbf{x}\right) < 0 \right\} \quad \text{et} \quad \partial\Omega = \left\{ \mathbf{x} \in \mathbb{R}^d, \omega\left(\mathbf{x}\right) = 0 \right\}$$

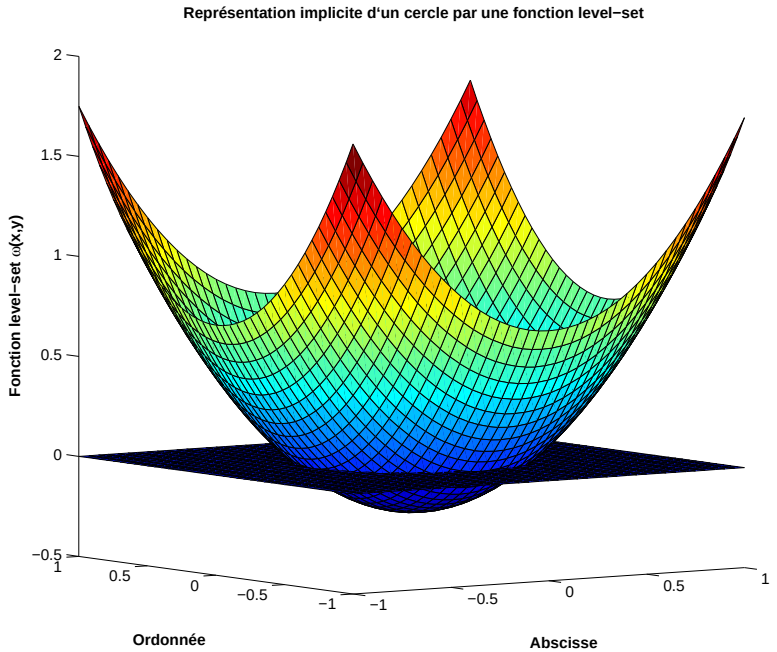
Contre-exemple : $\omega(\mathbf{x}) = -1$ si $\mathbf{x} \in \Omega$, zéro si $\mathbf{x} \in \partial\Omega$, et un si $\mathbf{x} \notin \Omega$.

adaptCube.c : le concept des lignes de niveaux (level-set)

Définition : Etant donné un domaine $\Omega \subset \mathbb{R}^d$, $d \geqslant 2$, on dit qu'une application continue $\omega : \mathbb{R}^d \mapsto \mathbb{R}$ est une fonction *level-set* si elle vérifie :

$$\Omega = \left\{ \mathbf{x} \in \mathbb{R}^d, \omega\left(\mathbf{x}\right) < 0 \right\} \quad \text{et} \quad \partial\Omega = \left\{ \mathbf{x} \in \mathbb{R}^d, \omega\left(\mathbf{x}\right) = 0 \right\}$$

Contre-exemple : $\omega(\mathbf{x}) = -1$ si $\mathbf{x} \in \Omega$, zéro si $\mathbf{x} \in \partial\Omega$, et un si $\mathbf{x} \notin \Omega$.
Exemple du cercle : $\omega(x, y) = (x - x_0)^2 + (y - y_0)^2 - r_0^2$.



adaptCube.c : le concept des lignes de niveaux (level-set)

Définition : On dit qu'une application différentiable (presque partout) $\omega : \mathbb{R}^d \to \mathbb{R}$ est la fonction *distance signée* à un domaine $\Omega \subset \mathbb{R}^d$, $d \geqslant 2$ si c'est la fonction level-set qui vérifie de plus $|\nabla\omega(\mathbf{x})| = 1$ (p.p.) $\mathbf{x} \in \mathbb{R}^d$.

adaptCube.c : le concept des lignes de niveaux (level-set)

Définition : On dit qu'une application différentiable (presque partout) $\omega : \mathbb{R}^d \to \mathbb{R}$ est la fonction *distance signée* à un domaine $\Omega \subset \mathbb{R}^d$, $d \geqslant 2$ si c'est la fonction level-set qui vérifie de plus $|\nabla\omega(\mathbf{x})| = 1$ (p.p.) $\mathbf{x} \in \mathbb{R}^d$.

Avantage : le signe décrit implicitement l'intérieur/extérieur ou le bord de Ω. La condition de normalisation permet de s'assurer, sans connaître Ω, que :

$$(\text{p.p.})\ \mathbf{x} \in \mathbb{R}^d, \quad \omega(\mathbf{x}) = \begin{cases} -\text{dist}\,(\mathbf{x}, \partial\Omega) & \text{si } \mathbf{x} \in \Omega \\ +\text{dist}\,(\mathbf{x}, \partial\Omega) & \text{si } \mathbf{x} \notin \Omega \end{cases}$$

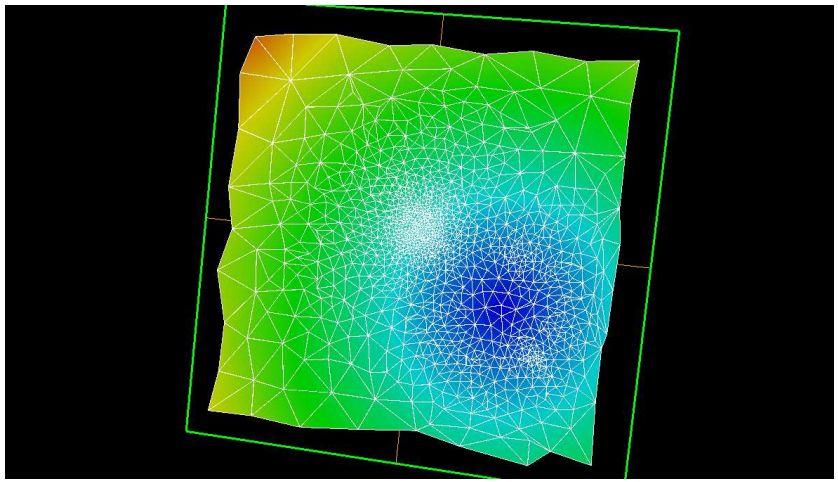
adaptCube.c : le concept des lignes de niveaux (level-set)

Définition : On dit qu'une application différentiable (presque partout) $\omega : \mathbb{R}^d \to \mathbb{R}$ est la fonction *distance signée* à un domaine $\Omega \subset \mathbb{R}^d$, $d \geqslant 2$ si c'est la fonction level-set qui vérifie de plus $|\nabla\omega(\mathbf{x})| = 1$ (p.p.) $\mathbf{x} \in \mathbb{R}^d$.

Avantage : le signe décrit implicitement l'intérieur/extérieur ou le bord de Ω. La condition de normalisation permet de s'assurer, sans connaître Ω, que :

$$(\text{p.p.})\ \mathbf{x} \in \mathbb{R}^d, \quad \omega(\mathbf{x}) = \begin{cases} -\text{dist}\,(\mathbf{x}, \partial\Omega) & \text{si } \mathbf{x} \in \Omega \\ +\text{dist}\,(\mathbf{x}, \partial\Omega) & \text{si } \mathbf{x} \notin \Omega \end{cases}$$

Exemple : $\omega(x, y, z) = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} - r_0$.



adaptCube.c : comment initialiser le domaine ?

Dans *main.h*, le domaine initial est une sphère prédéfinie par son centre et son rayon de nouveau grâce à des constantes de préprocesseur :

- LS_X, LS_Y, LS_Z (double) qui fournit les coordonnées du centre.
- LS_R (double) qui gère le rayon et doit vérifier LS_R > 0.

Dans l'idée d'interfacer le programme avec des fichiers *.cube*, on souhaite pouvoir prochainement initialiser un domaine par un cube.

adaptCube.c : comment initialiser le domaine ?

Dans *main.h*, le domaine initial est une sphère prédéfinie par son centre et son rayon de nouveau grâce à des constantes de préprocesseur :

- LS_X, LS_Y, LS_Z (double) qui fournit les coordonnées du centre.
- LS_R (double) qui gère le rayon et doit vérifier LS_R > 0.

Dans l'idée d'interfacer le programme avec des fichiers *.cube*, on souhaite pouvoir prochainement initialiser un domaine par un cube. **Convention :**

- pMesh->phex[i].label=3 signifie que le cube appartient au domaine Ω ;
- pMesh->phex[i].label=2 signifie que le cube n'appartient pas à Ω ;
- pMesh->pqua[i].label=10 signifie que le carré appartient au bord $\partial\Omega$.

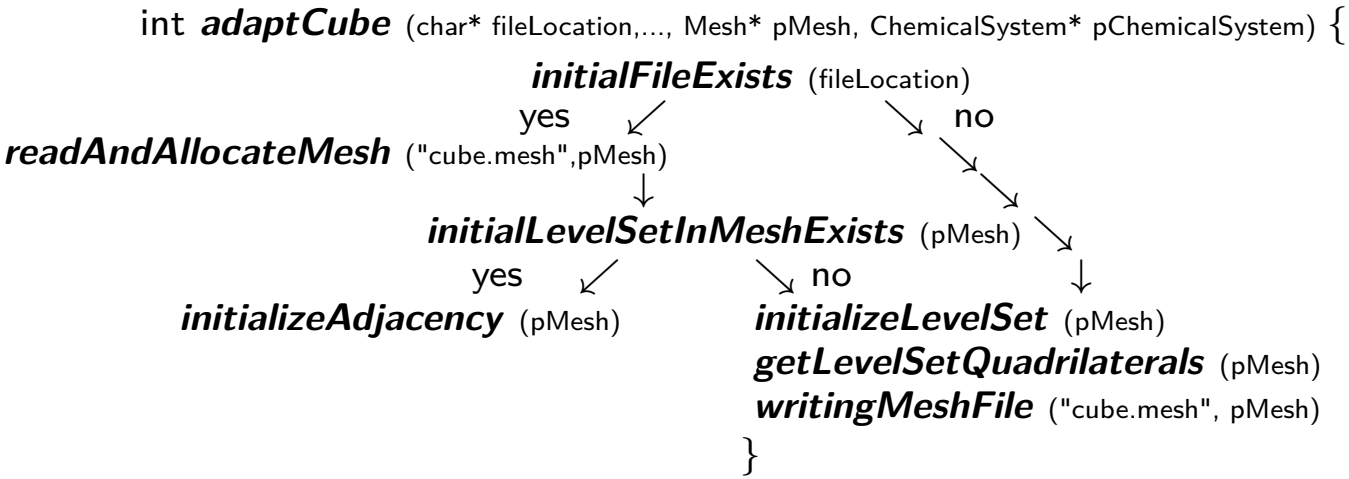
adaptCube.c : comment initialiser le domaine ?

Dans *main.h*, le domaine initial est une sphère prédéfinie par son centre et son rayon de nouveau grâce à des constantes de préprocesseur :

- LS_X, LS_Y, LS_Z (double) qui fournit les coordonnées du centre.
- LS_R (double) qui gère le rayon et doit vérifier LS_R > 0.

Dans l'idée d'interfacer le programme avec des fichiers *.cube*, on souhaite pouvoir prochainement initialiser un domaine par un cube. **Convention :**

- pMesh->phex[i].label=3 signifie que le cube appartient au domaine Ω ;
- pMesh->phex[i].label=2 signifie que le cube n'appartient pas à Ω ;
- pMesh->pqua[i].label=10 signifie que le carré appartient au bord $\partial\Omega$.



shapeDerivative.c : comment calculer la matrice ?

Rappel : $p_\nu(\Omega)$ se calcule à partir des valeurs propres de la matrice :

$$S(\Omega)_{ij} := \sum_{\sigma \in \{-\frac{1}{2}, \frac{1}{2}\}} \int_\Omega \phi_i(\mathbf{x}, \sigma)\phi_j(\mathbf{x}, \sigma)\, \mathrm{d}\mathbf{x}.$$

shapeDerivative.c : comment calculer la matrice ?

Rappel : $p_\nu(\Omega)$ se calcule à partir des valeurs propres de la matrice :

$$S(\Omega)_{ij} := \sum_{\sigma\in\{-\frac{1}{2},\frac{1}{2}\}} \int_\Omega \phi_i(\mathbf{x},\sigma)\phi_j(\mathbf{x},\sigma)\,\mathrm{d}\mathbf{x}.$$

- Notons que $S(\Omega)_{ij} = 0$ si ϕ_i et ϕ_j représentent des spins opposés. De plus, la matrice étant symétrique, on peut calculer seulement pour $i \leqslant j$.
- Découpons Ω en cubes $C_k := [x_k^-, x_k^+] \times [y_k^-, y_k^+] \times [z_k^-, z_k^+]$, et décomposons les orbitales moléculaires dans une base de gaussiennes :

$$S\left(\Omega\right)_{ij} = \sum_{r,s} c_i^r c_j^s \sum_{C_k\in\Omega} \int_{x_k^-}^{x_k^+} \int_{y_k^-}^{y_k^+} \int_{z_k^-}^{z_k^+} g_i^r\left(x,y,z\right) g_j^s\left(x,y,z\right)\,\mathrm{d}x\mathrm{d}y\mathrm{d}z.$$

- Puis, on observe qu'un produit de gaussiennes est à variables séparables. Le calcul de cette intégrale triple se ramène donc à calculer trois fois :

$$\int_{x^-}^{x^+} \left(x - x_i^r\right)^{p_i^r} \left(x - x_j^s\right)^{p_j^s} e^{-a_i^r\left(x-x_i^r\right)^2 - a_j^s\left(x-x_j^s\right)^2}\,\mathrm{d}x.$$

shapeDerivative.c : comment calculer la matrice ?

- Effectuons le changement de variables $t = \sqrt{a_i^r + a_j^s}(x - \frac{a_i^r x_i^r + a_j^s x_j^s}{a_i^r + a_j^s})$:

$$\frac{e^{\frac{a_i^r a_j^s}{a_i^r + a_j^s}(x_i^r - x_j^s)^2}}{2(a_i^r + a_j^s)^{\frac{1}{2}(1+p_i^r+p_j^s)}} \int_{t^-}^{t^+} 2\left(t - t_i^r\right)^{p_i^r} \left(t + t_j^s\right)^{p_j^s} e^{-t^2} \mathrm{d}t.$$

- En développant le polynôme, on se rend compte qu'on peut finalement évaluer de manière exacte les intégrales à partir des quantités de base :

$$\int_{t^-}^{t^+} 2t^\alpha e^{-t^2} dt.$$

shapeDerivative.c : comment calculer la matrice ?

- Effectuons le changement de variables $t = \sqrt{a_i^r + a_j^s}\left(x - \frac{a_i^r x_i^r + a_j^s x_j^s}{a_i^r + a_j^s}\right)$:

$$\frac{e^{\frac{a_i^r a_j^s}{a_i^r + a_j^s}(x_i^r - x_j^s)^2}}{2\left(a_i^r + a_j^s\right)^{\frac{1}{2}(1+p_i^r+p_j^s)}} \int_{t^-}^{t^+} 2\left(t - t_i^r\right)^{p_i^r} \left(t + t_j^s\right)^{p_j^s} e^{-t^2} \mathrm{d}t.$$

- En développant le polynôme, on se rend compte qu'on peut finalement évaluer de manière exacte les intégrales à partir des quantités de base :

$$\int_{t^-}^{t^+} 2t^{\alpha} e^{-t^2} dt.$$

int ***ComputeOverlapMatrixOnGrid*** (Mesh* pMesh, ChemicalSystem* pChemicalSystem, double* overlapMatrix, int label){

for all int i, j, r, s, k do

convertingType (int type)
return value $(100p^x + 10p^y + p^z)$

$\leftrightarrow$

evaluateTripleIntegralIJ
(pMesh, pChemicalSystem, ...(i,j)...., int r, int s, int k)
evaluateOneIntegralIJ
(int p_i^r, int p_j^s, double t_i^r, double t_j^s, double t^-, double t^+)
evaluateGaussianIntegral (int α, double t^-, double t^+)

}

main.c : quelques remarques pour la compréhension

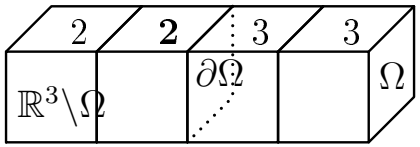
Quand on ajoute ou retire un cube au domaine, il y a deux façons de procéder qui sont gérées par la variable booléenne trick dans *main.h* :

- la fonction (dans *shapeDerivative.c*)

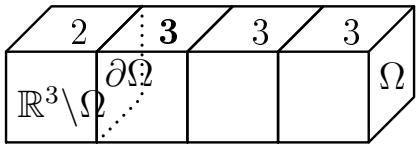
int ***AddOrRemoveHexahedron*** (Mesh* pMesh, int *j*, int addOrRemove)

modifie selon AddOrRemove le label des cubes ayant le carré *j* en commun

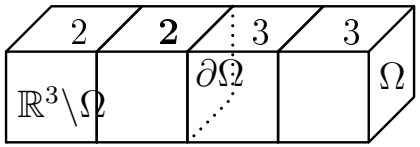
Cas où on a trick = 0



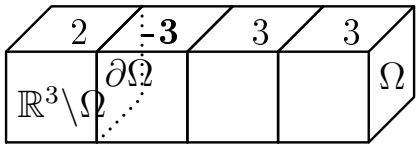
↓ addOrRemove = 1 ↓



Cas où on a trick = 1



↓ addOrRemove = -3 ↓



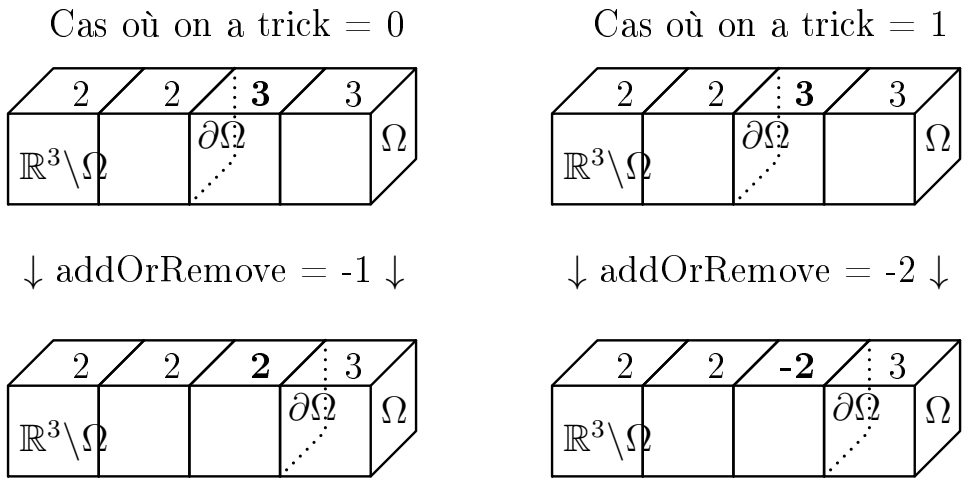
main.c : quelques remarques pour la compréhension

Quand on ajoute ou retire un cube au domaine, il y a deux façons de procéder qui sont gérées par la variable booléenne trick dans *main.h* :

- la fonction (dans *shapeDerivative.c*)

int ***AddOrRemoveHexahedron*** (Mesh* pMesh, int *j*, int addOrRemove)

modifie selon AddOrRemove le label des cubes ayant le carré *j* en commun



- et cela grâce à la table d'adjacence que l'on peut recréer facilement avec la fonction (dans *adaptCube.h*)

int ***getLevelSetQuadrilaterals*** (Mesh* pMesh)

main.c : quelques remarques pour la compilation

- Si trick $= 0$, la matrice doit être recalculée à chaque fois par

  int ***shapeDerivative*** (Mesh* pMesh, ChemicalSystem* pChemicalSystem, double* overlapMatrix, double* eigenvectors, double* eigenvalues, int nuElectrons, double* pProbability, double* pProbabilityOld, int label, int i) {

  computeOverlapMatrixOnGrid (pMesh, pChemicalSystem, overlapMatrix, label, i)
  diagonalizeOverlapMatrix (overlapMatrix, eigenvectors, eigenvalues,pChemicalSystem->nmorb)
  *pProbabilityOld = *pProbability
  *pProbability = ***diagonalizeOverlapMatrix*** (eigenvalues,pChemicalSystem->nmorb,nuElectrons) }

  et dans ce cas la variable label est celle qui représente les cubes non calculés (-1 pour $\mathbb{R}^3$, 2 pour $\Omega = 3$ et donc 3 pour $\mathbb{R}^3 \backslash \Omega = 2$)

main.c : quelques remarques pour la compilation

- Si trick $= 0$, la matrice doit être recalculée à chaque fois par

  int ***shapeDerivative*** (Mesh* pMesh, ChemicalSystem* pChemicalSystem, double* overlapMatrix, double* eigenvectors, double* eigenvalues, int nuElectrons, double* pProbability, double* pProbabilityOld, int label, int i)

  et dans ce cas la variable label est celle qui représente les cubes non calculés (-1 pour $\mathbb{R}^3$, 2 pour $\Omega = 3$ et donc 3 pour $\mathbb{R}^3 \backslash \Omega = 2$)

- et si trick $= 1$, la variable label représente l'intégrale du cube à calculer (-2 pour C_j^{ext}, -3 pour C_j^{int}), valeur qui est ensuite ajoutée ou retranchée aux coefficients de la matrice. Cette méthode est beaucoup plus rapide.

main.c : quelques remarques pour la compilation

- Si trick $= 0$, la matrice doit être recalculée à chaque fois par

  int ***shapeDerivative*** (Mesh* pMesh, ChemicalSystem* pChemicalSystem, double* overlapMatrix, double* eigenvectors, double* eigenvalues, int nuElectrons, double* pProbability, double* pProbabilityOld, int label, int i)

  et dans ce cas la variable label est celle qui représente les cubes non calculés (-1 pour $\mathbb{R}^3$, 2 pour $\Omega = 3$ et donc 3 pour $\mathbb{R}^3 \backslash \Omega = 2$)
- et si trick $= 1$, la variable label représente l'intégrale du cube à calculer (-2 pour C_j^{ext}, -3 pour C_j^{int}), valeur qui est ensuite ajoutée ou retranchée aux coefficients de la matrice. Cette méthode est beaucoup plus rapide.
- Le calcul des intégrale nécessite d'inclure la librairie de mathématique (*math.h*) dans le fichier *main.h* (option -lm avec gcc).
- La diagonalisation se fait avec la fonction *dsyev* de la librairie LAPACK. Etant écrite en Fortran, on doit l'interfacer avec C grâce à la librairie LAPACKE. On doit donc la pré-installer sur l'ordinateur et inclure *llapacke.h* dans le fichier *main.h* (option -llapacke dans gcc).
- Le calcul de l'intégrale (boucle sur k) peut se paralléliser grâce à la librairie OPENMP. On doit donc la pré-installer sur l'ordinateur et inclure *omp.h* dans le fichier *main.h* (option -fopenmp dans gcc).

La dérivée de formes comme direction privilégiée

L'algorithme précédent est simple à concevoir mais il est lent car il nécessite de modifier le domaine cube par cube.

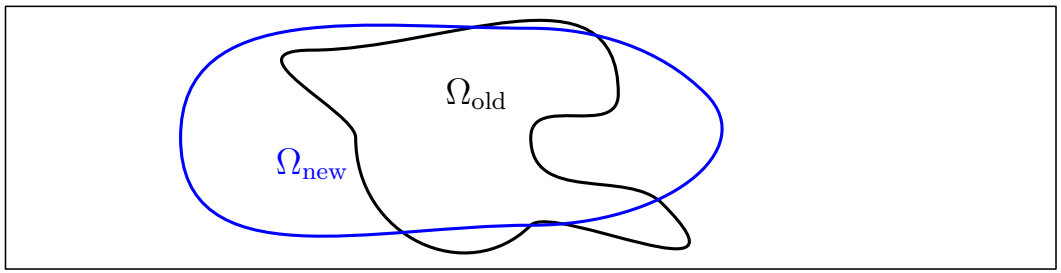
On regarde maintenant s'il n'est pas possible de procéder avec plusieurs cubes en même temps, tout en s'assurant que la probabilité augmente.

La dérivée de formes comme direction privilégiée

L'algorithme précédent est simple à concevoir mais il est lent car il nécessite de modifier le domaine cube par cube.

On regarde maintenant s'il n'est pas possible de procéder avec plusieurs cubes en même temps, tout en s'assurant que la probabilité augmente.

Question : qu'est ce qu'une (petite) perturbation d'un domaine ?

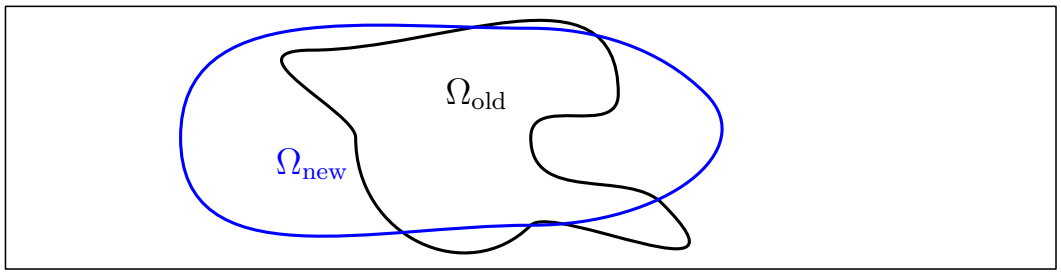


La dérivée de formes comme direction privilégiée

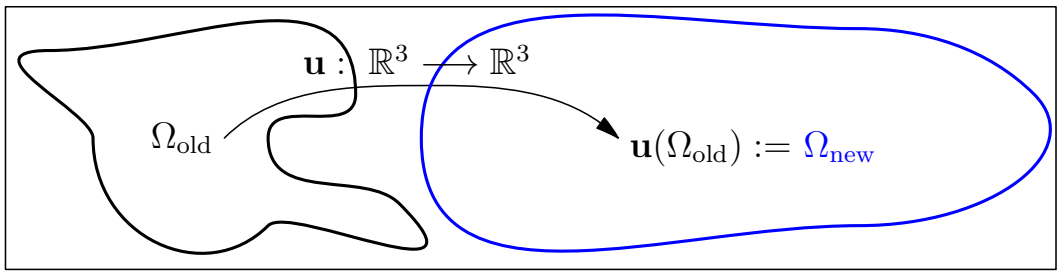
L'algorithme précédent est simple à concevoir mais il est lent car il nécessite de modifier le domaine cube par cube.

On regarde maintenant s'il n'est pas possible de procéder avec plusieurs cubes en même temps, tout en s'assurant que la probabilité augmente.

Question : qu'est ce qu'une (petite) perturbation d'un domaine ?



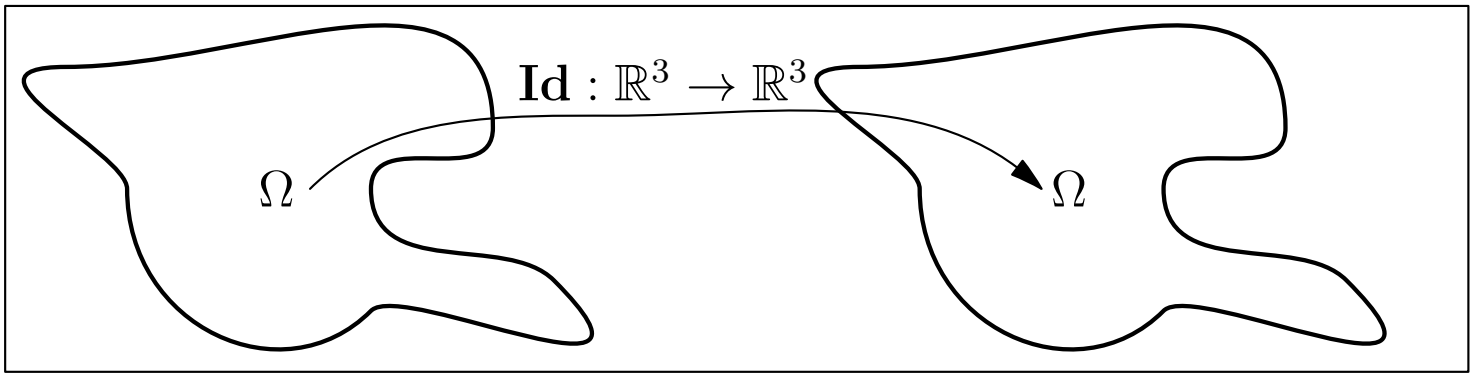
Idée : identifier les déformations d'un domaine comme les ensembles images d'une famille d'applications.



La dérivée de formes comme direction privilégiée

On peut donc travailler sur les $\mathbf{u} : \mathbb{R}^3 \to \mathbb{R}^3$ (analyse fonctionnelle) au lieu de considérer des domaines (pas de topologie sur les parties de $\mathbb{R}^3$).

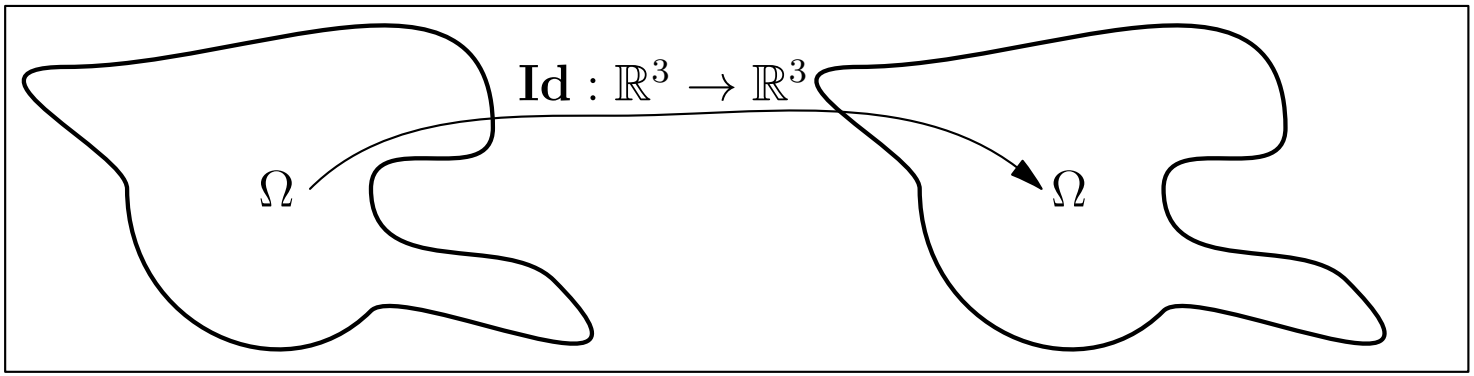
Pas de déformation : c'est l'application identité $Id : \mathbf{x} \in \mathbb{R}^3 \longmapsto \mathbf{x} \in \mathbb{R}^3$.



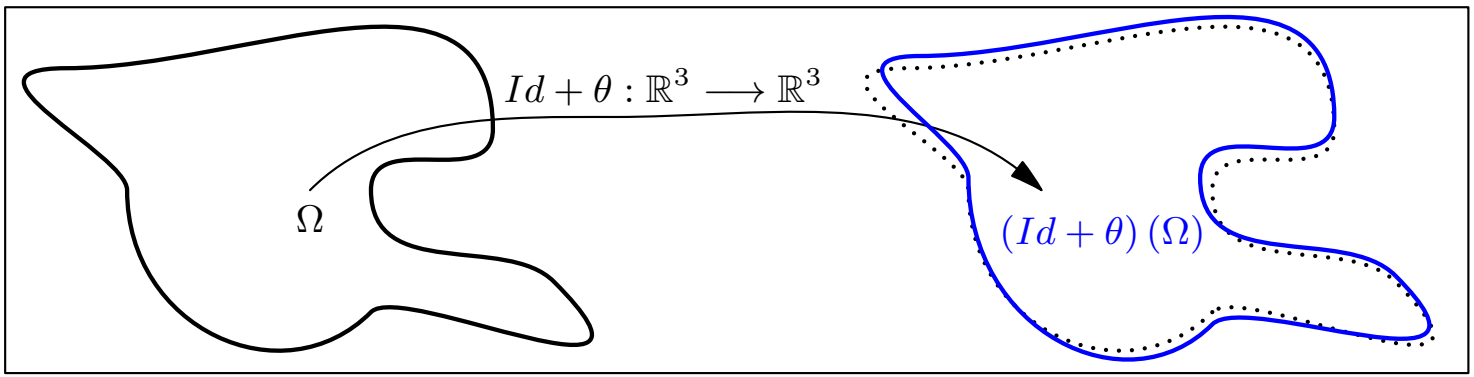
La dérivée de formes comme direction privilégiée

On peut donc travailler sur les $\mathbf{u} : \mathbb{R}^3 \to \mathbb{R}^3$ (analyse fonctionnelle) au lieu de considérer des domaines (pas de topologie sur les parties de $\mathbb{R}^3$).

Pas de déformation : c'est l'application identité $Id : \mathbf{x} \in \mathbb{R}^3 \longmapsto \mathbf{x} \in \mathbb{R}^3$.



Petite déformation : c'est une perturbation $\theta : \mathbb{R}^3 \to \mathbb{R}^3$ de l'identité.



La dérivée de formes comme direction privilégiée

Pour gérer les déformations, on introduit une fonctionnelle modifiée de la probabilité $\Omega \mapsto p_\nu(\Omega)$

$$\tilde{p}_{\nu,\Omega} : \theta \longmapsto \tilde{p}_{\nu,\Omega}(\theta) := p_\nu \left[(Id + \theta)(\Omega) \right].$$

La dérivée de formes comme direction privilégiée

Pour gérer les déformations, on introduit une fonctionnelle modifiée de la probabilité $\Omega \mapsto p_\nu(\Omega)$

$$\tilde{p}_{\nu,\Omega} : \theta \longmapsto \tilde{p}_{\nu,\Omega}(\theta) := p_\nu \left[(Id + \theta)(\Omega) \right].$$

Dérivée de formes de p_ν : c'est un développement de Taylor à l'origine de la fonctionnelle modifiée $\tilde{p}_{\nu,\Omega}$:

$$\tilde{p}_{\nu,\Omega}(\theta) \quad = \quad \tilde{p}_{\nu,\Omega}(\mathbf{0}) \quad + \quad \mathrm{D}_{\mathbf{0}}\tilde{p}_{\nu,\Omega}(\theta) \quad + o(\theta)$$

La dérivée de formes comme direction privilégiée

Pour gérer les déformations, on introduit une fonctionnelle modifiée de la probabilité $\Omega \mapsto p_\nu(\Omega)$

$$\tilde{p}_{\nu,\Omega} : \theta \longmapsto \tilde{p}_{\nu,\Omega}(\theta) := p_\nu \left[(Id + \theta)(\Omega) \right].$$

Dérivée de formes de p_ν : c'est un développement de Taylor à l'origine de la fonctionnelle modifiée $\tilde{p}_{\nu,\Omega}$:

$$\tilde{p}_{\nu,\Omega}(\theta) = \tilde{p}_{\nu,\Omega}(\mathbf{0}) + \mathrm{D}_{\mathbf{0}}\tilde{p}_{\nu,\Omega}(\theta) + o(\theta) \quad \begin{array}{c} \text{on veut} \\ \geqslant \tilde{p}_{\nu,\Omega}(\mathbf{0}) = p_\nu(\Omega) \end{array}$$

La dérivée de formes comme direction privilégiée

Pour gérer les déformations, on introduit une fonctionnelle modifiée de la probabilité $\Omega \mapsto p_\nu(\Omega)$

$$\tilde{p}_{\nu,\Omega} : \theta \longmapsto \tilde{p}_{\nu,\Omega}(\theta) := p_\nu \left[(Id + \theta)(\Omega) \right].$$

Dérivée de formes de p_ν : c'est un développement de Taylor à l'origine de la fonctionnelle modifiée $\tilde{p}_{\nu,\Omega}$:

$$\begin{array}{rcccl|c} \tilde{p}_{\nu,\Omega}(\theta) & = & \tilde{p}_{\nu,\Omega}(\mathbf{0}) & + & \mathrm{D}_{\mathbf{0}}\tilde{p}_{\nu,\Omega}(\theta) + o(\theta) & \text{on veut} \\ & & & & & \geqslant \tilde{p}_{\nu,\Omega}(\mathbf{0}) = p_\nu(\Omega) \\ p_\nu(\Omega_\theta) & = & p_\nu(\Omega) & + & \displaystyle\int_{\partial\Omega} \frac{\partial p_\nu}{\partial \Omega}\theta_{\mathbf{n}} + o(\theta) & \text{pour } \theta \text{ petit} \end{array}$$

La dérivée de formes comme direction privilégiée

Pour gérer les déformations, on introduit une fonctionnelle modifiée de la probabilité $\Omega \mapsto p_\nu(\Omega)$

$$\tilde{p}_{\nu,\Omega} : \theta \longmapsto \tilde{p}_{\nu,\Omega}(\theta) := p_\nu\left[(Id + \theta)(\Omega)\right].$$

Dérivée de formes de p_ν : c'est un développement de Taylor à l'origine de la fonctionnelle modifiée $\tilde{p}_{\nu,\Omega}$:

$$\begin{array}{rcl|c} \tilde{p}_{\nu,\Omega}(\theta) & = & \tilde{p}_{\nu,\Omega}(\mathbf{0}) + \mathrm{D}_{\mathbf{0}}\tilde{p}_{\nu,\Omega}(\theta) + o(\theta) & \text{on veut} \\ & & & \geqslant \tilde{p}_{\nu,\Omega}(\mathbf{0}) = p_\nu(\Omega) \\ p_\nu(\Omega_\theta) & = & p_\nu(\Omega) + \displaystyle\int_{\partial\Omega} \frac{\partial p_\nu}{\partial \Omega}\theta_{\mathbf{n}} + o(\theta) & \text{pour } \theta \text{ petit} \end{array}$$

Inégalité de Cauchy-Schwarz : on a une borne optimale pour le terme

$$\int_{\partial\Omega} \frac{\partial p_\nu}{\partial \Omega}\theta_{\mathbf{n}} \leqslant \sqrt{\int_{\partial\Omega}\left(\frac{\partial p_\nu}{\partial \Omega}\right)^2}\sqrt{\int_{\partial\Omega}\theta_{\mathbf{n}}^2},$$

et l'égalité a lieu si et seulement si $\theta_{\mathbf{n}} = t\frac{\partial p_\nu}{\partial \Omega}$ avec $t > 0$.

La dérivée de formes comme direction privilégiée

Gradient de formes : c'est la meilleure perturbation locale ($t > 0$ petit) $\boxed{\theta(\mathbf{x}) = t\frac{\partial p_\nu}{\partial \Omega}(\mathbf{x})\mathbf{n}_{\partial\Omega}(\mathbf{x})}$ a priori définie pour tout $\mathbf{x} \in \partial\Omega$. On a :

$$p_\nu(\Omega_\theta) \;=\; p_\nu(\Omega) \;+\; t\int_{\partial\Omega}\left(\frac{\partial p_\nu}{\partial \Omega}\right)^2 + o(t) \;\geqslant\; p_\nu(\Omega),$$

La dérivée de formes comme direction privilégiée

Gradient de formes : c'est la meilleure perturbation locale ($t > 0$ petit) $\boxed{\theta(\mathbf{x}) = t\frac{\partial p_\nu}{\partial \Omega}(\mathbf{x})\mathbf{n}_{\partial\Omega}(\mathbf{x})}$ a priori définie pour tout $\mathbf{x} \in \partial\Omega$. On a :

$$p_\nu(\Omega_\theta) \;=\; p_\nu(\Omega) \;+\; t\int_{\partial\Omega}\left(\frac{\partial p_\nu}{\partial \Omega}\right)^2 + o(t) \;\geqslant\; p_\nu(\Omega),$$

Interprétation Physique : le gradient de formes défini sur le bord donne l'intensité avec laquelle on doit déformer la surface le long de la normale afin d'augmenter (localement) la fonctionnelle de manière optimale.

La dérivée de formes comme direction privilégiée

Gradient de formes : c'est la meilleure perturbation locale ($t > 0$ petit) $\boxed{\theta(\mathbf{x}) = t\frac{\partial p_\nu}{\partial \Omega}(\mathbf{x})\mathbf{n}_{\partial\Omega}(\mathbf{x})}$ a priori définie pour tout $\mathbf{x} \in \partial\Omega$. On a :

$$p_\nu(\Omega_\theta) \;=\; p_\nu(\Omega) \;+\; t\int_{\partial\Omega}\left(\frac{\partial p_\nu}{\partial \Omega}\right)^2 + o(t) \;\geqslant\; p_\nu(\Omega),$$

Interprétation Physique : le gradient de formes défini sur le bord donne l'intensité avec laquelle on doit déformer la surface le long de la normale afin d'augmenter (localement) la fonctionnelle de manière optimale.

Proposition (2004, Cancès, Keriven, Lodier, Savin)

On rappelle que $p_\nu(\Omega) = \sum_{\mathrm{card}\, I_\nu = \nu} \prod_{l\in I_\nu} \lambda_l \prod_{l\notin I_\nu}(1-\lambda_l)$ *mais on a aussi :*

$$\frac{\partial p_\nu}{\partial \Omega}(\mathbf{x}) = \sum_{i,j,k=1}^{n}\left(\sum_{\substack{I_\nu\subset\{1,\dots,n\}\\ \mathrm{card}\;I_\nu=\nu}} \varepsilon_{I_\nu}(k)\prod_{\substack{l\in I_\nu\\ l\neq k}}\lambda_l\prod_{\substack{l\notin I_\nu\\ l\neq k}}(1-\lambda_l)\right) X_{ik}X_{jk}\phi_i(\mathbf{x})\phi_j(\mathbf{x}),\ \textit{où}$$

$SX = XD$ et $X^TX = I_n$ avec D la matrice diagonale formée des valeurs propres et chaque colonne de X un vecteur propre normalisé.

Un nouvel algorithme d'optimisation cubique

Après avoir chargé les données chimiques, initialisé la boîte de calcul et le domaine initial Ω_0, pour tout $k = 0, \ldots, K$, étant donné Ω_k :

- pour tout carré C_j^k sur le bord de Ω_k et on cherche les coordonnées de son centre $\mathbf{x}_j^k$.
- si $\frac{\partial p_\nu}{\partial \Omega}(\mathbf{x}_j^k) > 0$, alors on définit $\Omega_{k+1} := \Omega_k \sqcup C_k^{\text{ext}}$;
- sinon on voit si $\frac{\partial p_\nu}{\partial \Omega}(\mathbf{x}_j^k) < 0$ et alors $\Omega_{k+1} := \Omega_k \setminus C_k^{\text{int}}$;
- et si ce n'est toujours pas le cas, alors on ne fait rien : $\Omega_{k+1} := \Omega_k$.

Un nouvel algorithme d'optimisation cubique

Après avoir chargé les données chimiques, initialisé la boîte de calcul et le domaine initial Ω_0, pour tout $k = 0, \ldots, K$, étant donné Ω_k :

- pour tout carré C_j^k sur le bord de Ω_k et on cherche les coordonnées de son centre $\mathbf{x}_j^k$.
- si $\frac{\partial p_\nu}{\partial \Omega}(\mathbf{x}_j^k) > 0$, alors on définit $\Omega_{k+1} := \Omega_k \sqcup C_k^{\text{ext}}$;
- sinon on voit si $\frac{\partial p_\nu}{\partial \Omega}(\mathbf{x}_j^k) < 0$ et alors $\Omega_{k+1} := \Omega_k \setminus C_k^{\text{int}}$;
- et si ce n'est toujours pas le cas, alors on ne fait rien : $\Omega_{k+1} := \Omega_k$.

L'avantage de cette méthode est qu'elle est très rapide, mais elle oscille une fois la convergence atteinte. De plus, on souhaiterais pouvoir l'interfacer ensuite avec l'algorithme précédent pour affiner localement la forme. Cependant, l'approximation de la géométrie (normale) est très mal réalisée par cet algorithme. Et on souhaite aussi réduire le nombre de mailles tout en gagnant en précision.

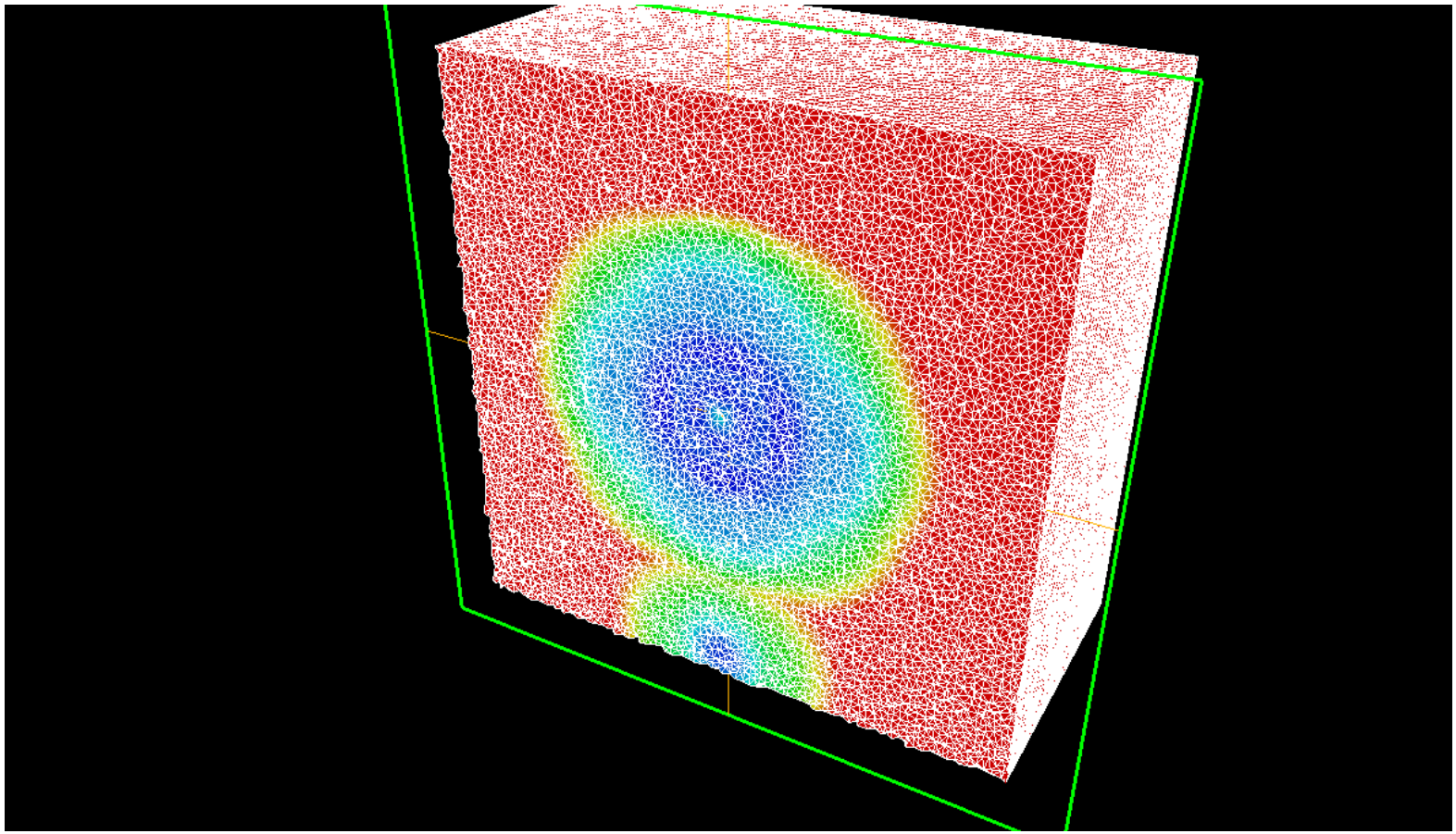
Le passage à un maillage tétraédrique

- On considère maintenant un maillage fait de tétraèdres, triangles, et sommets. Les bords du cube sont aussi stockés sous forme de segments, et on calcule aussi les normales de points du bords.
- Ce changement de programme utilise la constante de préprocesseur LS_MODE. Si elle n'est pas définie dans *main.h*, alors le programme utilise des tétraèdres. Sinon, c'est la version cubique : LS_MODE 0 pour la méthode aléatoire, LS_MODE 1 pour utiliser la dérivée de formes.

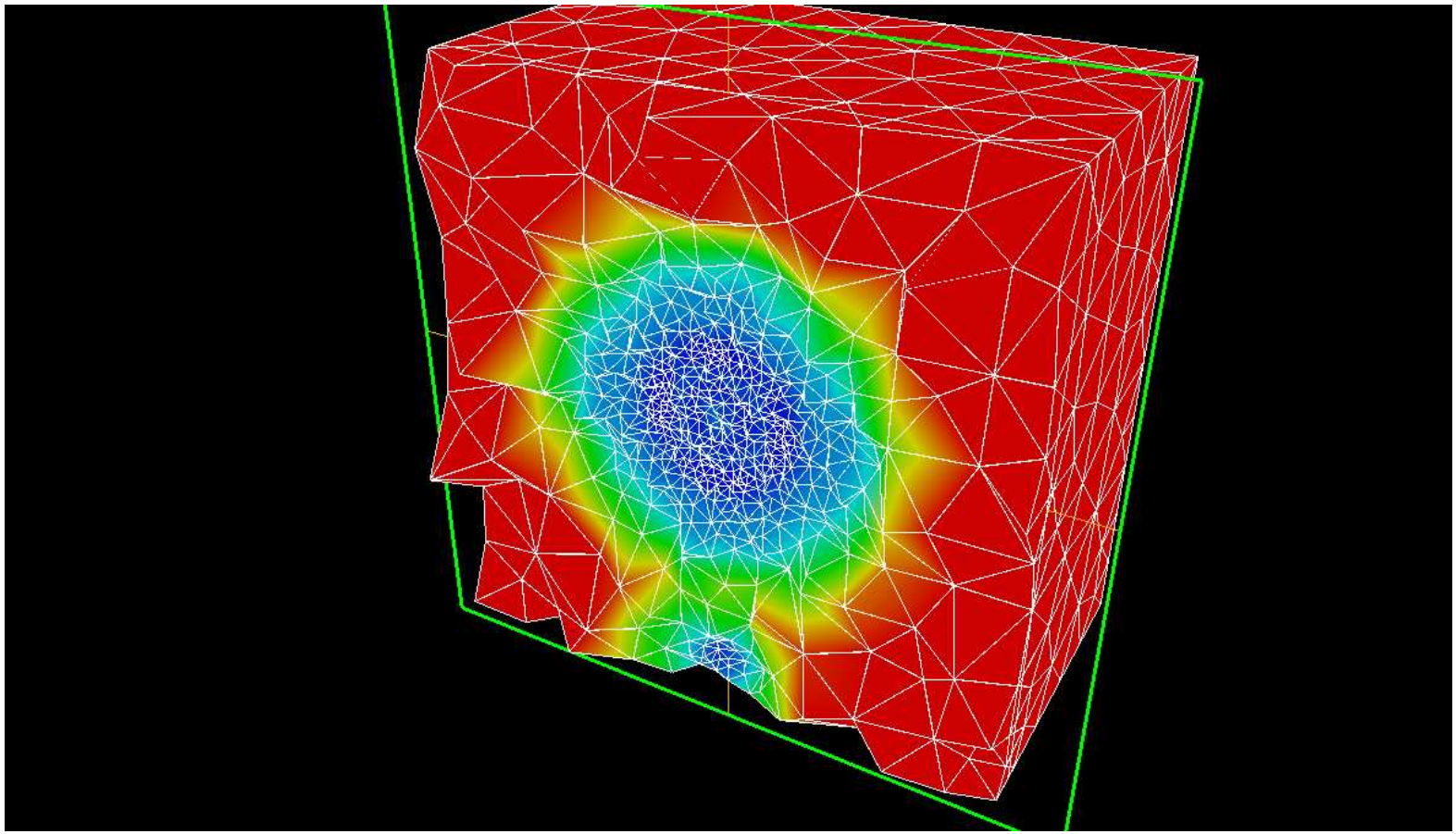
Le passage à un maillage tétraédrique

- On considère maintenant un maillage fait de tétraèdres, triangles, et sommets. Les bords du cube sont aussi stockés sous forme de segments, et on calcule aussi les normales de points du bords.
- Ce changement de programme utilise la constante de préprocesseur LS_MODE. Si elle n'est pas définie dans *main.h*, alors le programme utilise des tétraèdres. Sinon, c'est la version cubique : LS_MODE 0 pour la méthode aléatoire, LS_MODE 1 pour utiliser la dérivée de formes.
- Le programme utilise de nombreux autre logiciels développé au sein de l'iscd : medit pour la visualisation, mshdist pour la renormalisation de la distance signée, elastic pour l'extension d'un champ de vecteurs, advect pour l'advecteur de la fonction level-set suivant un champs de vecteurs prescrit, mmg3d (version modifiée) pour adapter la maillage à une métrique et à la fonction level-set.
- Adapter la maillage, c'est prescrire une taille locale de maille et modifier le maillage pour qu'il respecte cette cartographie des tailles. Cela permet de raffiner là où on doit être précis, et d'enlever des mailles là où ce n'est pas important. On utilise un fichier *.sol* pour stocker ces données.

Le concept d'adaptation de maillage pour M_oH



Le concept d'adaptation de maillage pour M_oH



L'algorithme associé à un maillage tétraédrique

Etant donné une surface correctement initialisée, on effectue la boucle :

- Calcul d'une métrique (calcul de hessiennes) associées aux orbitales (minimisation des erreurs d'intégrales) ainsi qu'à la fonction level-set (minimisation des erreurs de géométrie) : ***evaluatingMetricOnMesh***et ***writingSolFile***

L'algorithme associé à un maillage tétraédrique

Etant donné une surface correctement initialisée, on effectue la boucle :

- Calcul d'une métrique (calcul de hessiennes) associées aux orbitales (minimisation des erreurs d'intégrales) ainsi qu'à la fonction level-set (minimisation des erreurs de géométrie) : ***evaluatingMetricOnMesh***et ***writingSolFile***
- Redistanciation de la fonction level-set : *mshdist cube.mesh -dom -ncpu 4*

L'algorithme associé à un maillage tétraédrique

Etant donné une surface correctement initialisée, on effectue la boucle :

- Calcul d'une métrique (calcul de hessiennes) associées aux orbitales (minimisation des erreurs d'intégrales) ainsi qu'à la fonction level-set (minimisation des erreurs de géométrie) : ***evaluatingMetricOnMesh***et ***writingSolFile***
- Redistanciation de la fonction level-set : *mshdist cube.mesh -dom -ncpu 4*
- Calcul de la dérivée de formes grâce aux fonctions ***computeShapeDerivative***, ***writingShapeSolFile*** et ***writingMeshFile***.

L'algorithme associé à un maillage tétraédrique

Etant donné une surface correctement initialisée, on effectue la boucle :

- Calcul d'une métrique (calcul de hessiennes) associées aux orbitales (minimisation des erreurs d'intégrales) ainsi qu'à la fonction level-set (minimisation des erreurs de géométrie) : ***evaluatingMetricOnMesh***et ***writingSolFile***
- Redistanciation de la fonction level-set : *mshdist cube.mesh -dom -ncpu 4*
- Calcul de la dérivée de formes grâce aux fonctions ***computeShapeDerivative***, ***writingShapeSolFile*** et ***writingMeshFile***.
- Extension de la dérivée de formes en champs de vecteurs en dehors du bord du domaine : *elastic cube.mesh -p ../inputFiles/cube.elas -s cube.sol -o cube.new.sol*

L'algorithme associé à un maillage tétraédrique

Etant donné une surface correctement initialisée, on effectue la boucle :

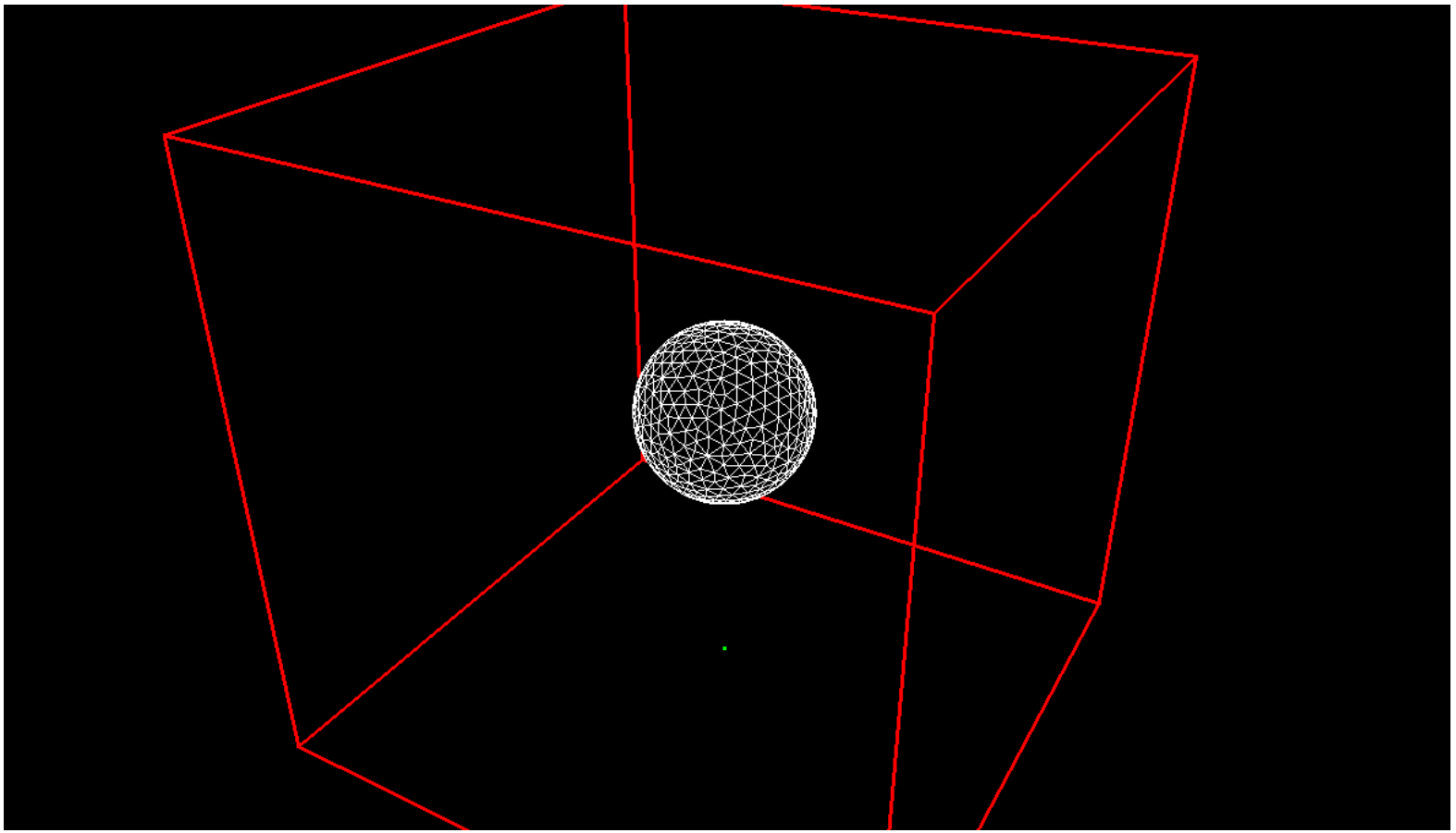
- Calcul d'une métrique (calcul de hessiennes) associées aux orbitales (minimisation des erreurs d'intégrales) ainsi qu'à la fonction level-set (minimisation des erreurs de géométrie) : ***evaluatingMetricOnMesh***et ***writingSolFile***
- Redistanciation de la fonction level-set : *mshdist cube.mesh -dom -ncpu 4*
- Calcul de la dérivée de formes grâce aux fonctions ***computeShapeDerivative***, ***writingShapeSolFile*** et ***writingMeshFile***.
- Extension de la dérivée de formes en champs de vecteurs en dehors du bord du domaine : *elastic cube.mesh -p ../inputFiles/cube.elas -s cube.sol -o cube.new.sol*
- Advection de la fonction level-set selon ce champs de vecteurs : *advect -dt 1.0 -nocfl cube.mesh -c cube.chi.sol -s cube.sol -o cube.new.sol*

L'algorithme associé à un maillage tétraédrique

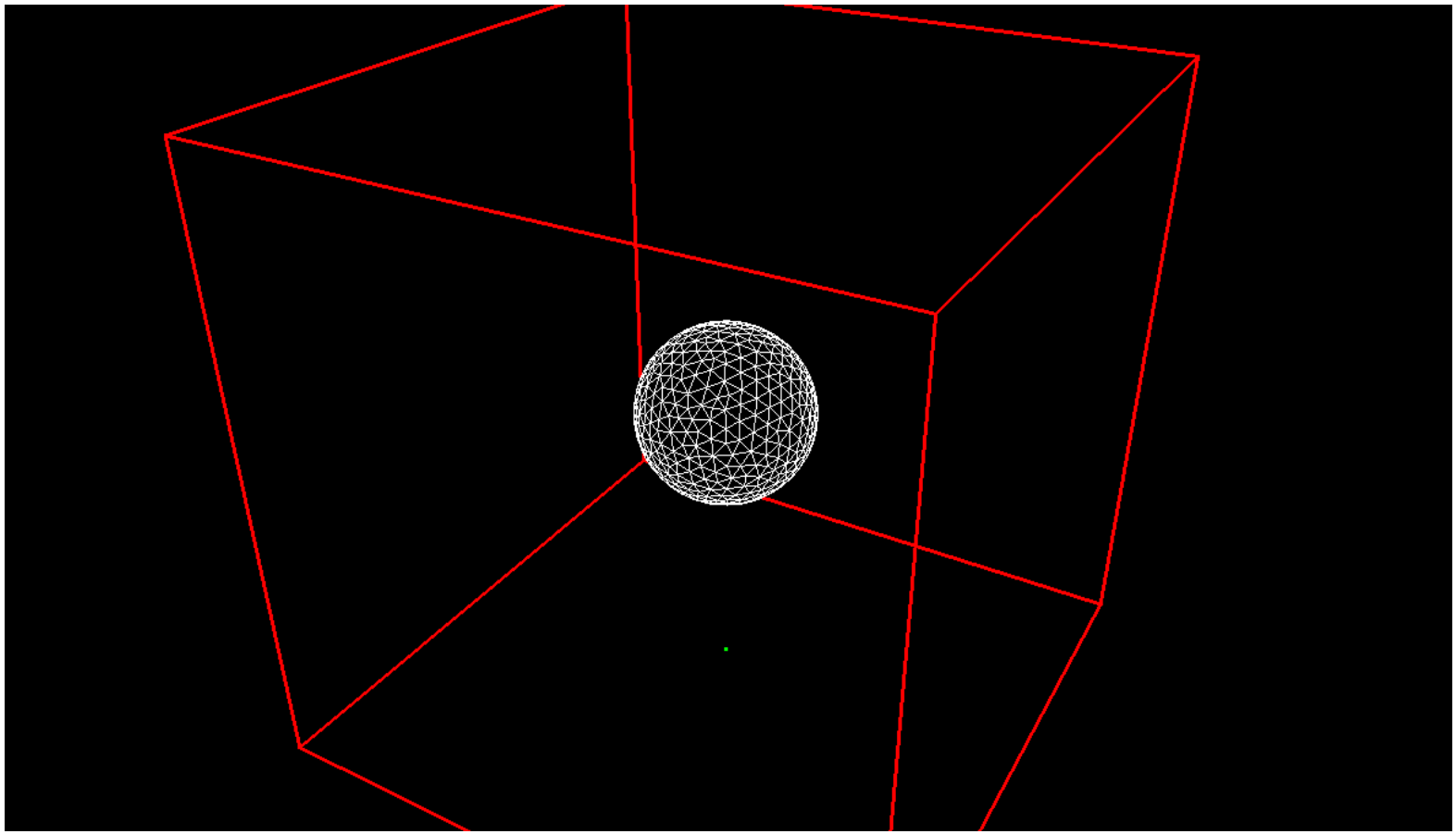
Etant donné une surface correctement initialisée, on effectue la boucle :

- Calcul d'une métrique (calcul de hessiennes) associées aux orbitales (minimisation des erreurs d'intégrales) ainsi qu'à la fonction level-set (minimisation des erreurs de géométrie) : ***evaluatingMetricOnMesh***et ***writingSolFile***
- Redistanciation de la fonction level-set : *mshdist cube.mesh -dom -ncpu 4*
- Calcul de la dérivée de formes grâce aux fonctions ***computeShapeDerivative***, ***writingShapeSolFile*** et ***writingMeshFile***.
- Extension de la dérivée de formes en champs de vecteurs en dehors du bord du domaine : *elastic cube.mesh -p ../inputFiles/cube.elas -s cube.sol -o cube.new.sol*
- Advection de la fonction level-set selon ce champs de vecteurs : *advect -dt 1.0 -nocfl cube.mesh -c cube.chi.sol -s cube.sol -o cube.new.sol*
- Récupération de la nouvelle surface à partir de la fonction level-set advectée puis adaptation du maillage à la level-set tout en respectant la métrique liée aux orbitales (beaucoup de mailles près des noyaux) : *mmg3d_O3 -in cube.mesh -ls -sol cube.sol -out cubeAdapt.mesh -nr -hmin 0.05 -hmax 0.1 -hgrad 2.0 -hausd 0.01*

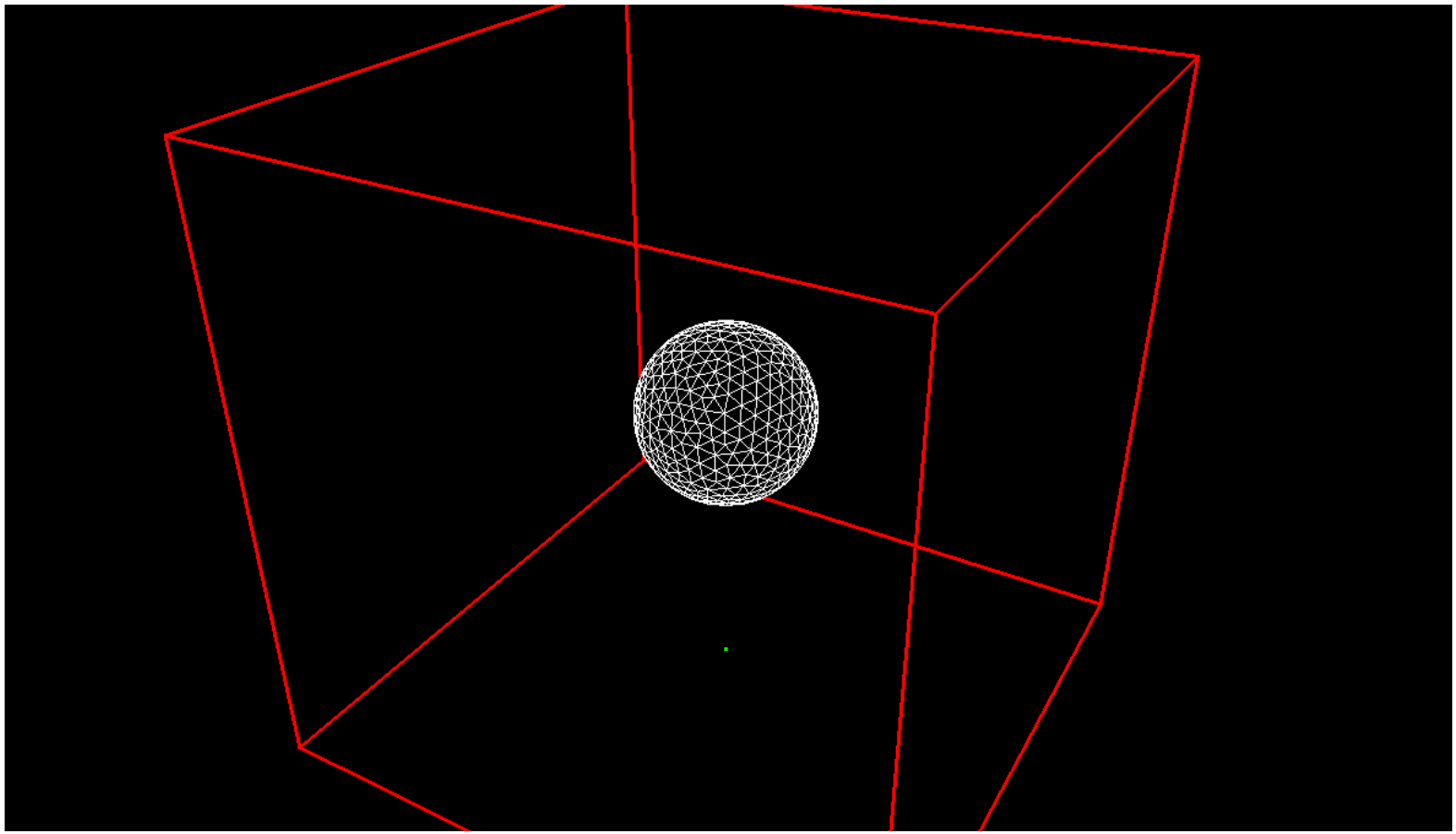
Résultats : exemple de la molécule M_oH (itération 0)



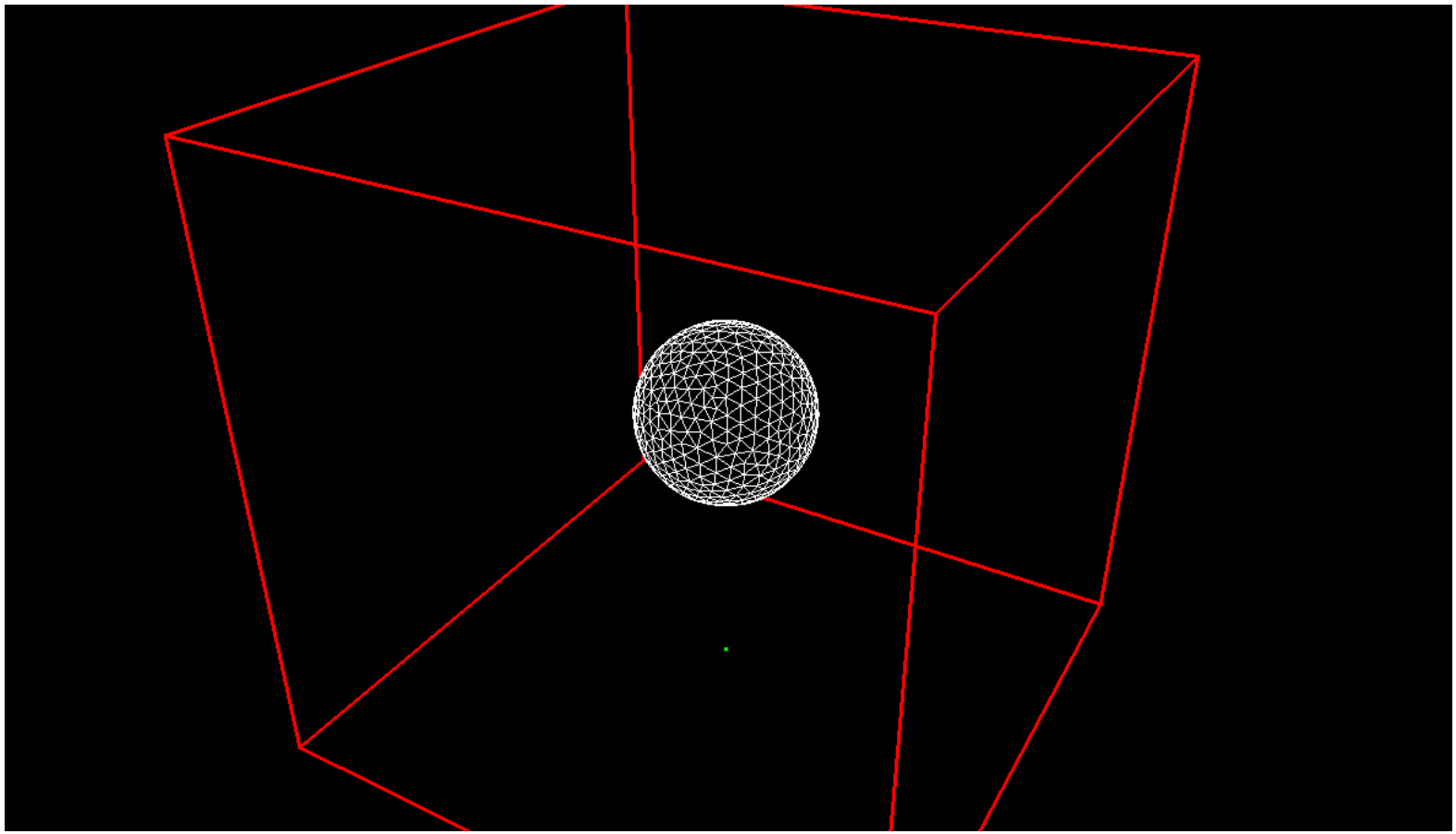
Résultats : exemple de la molécule M_oH (itération 1)



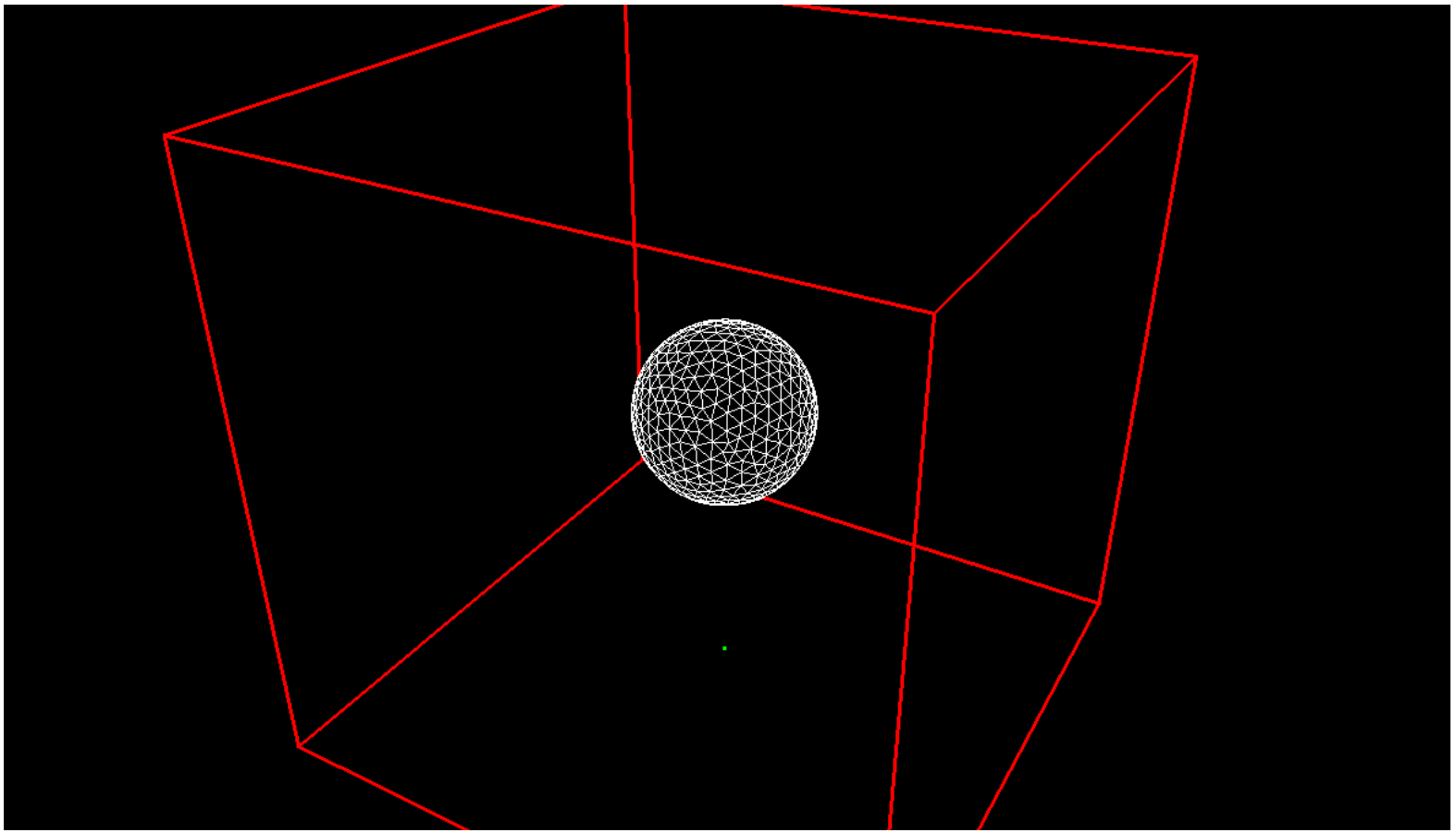
Résultats : exemple de la molécule M_oH (itération 2)



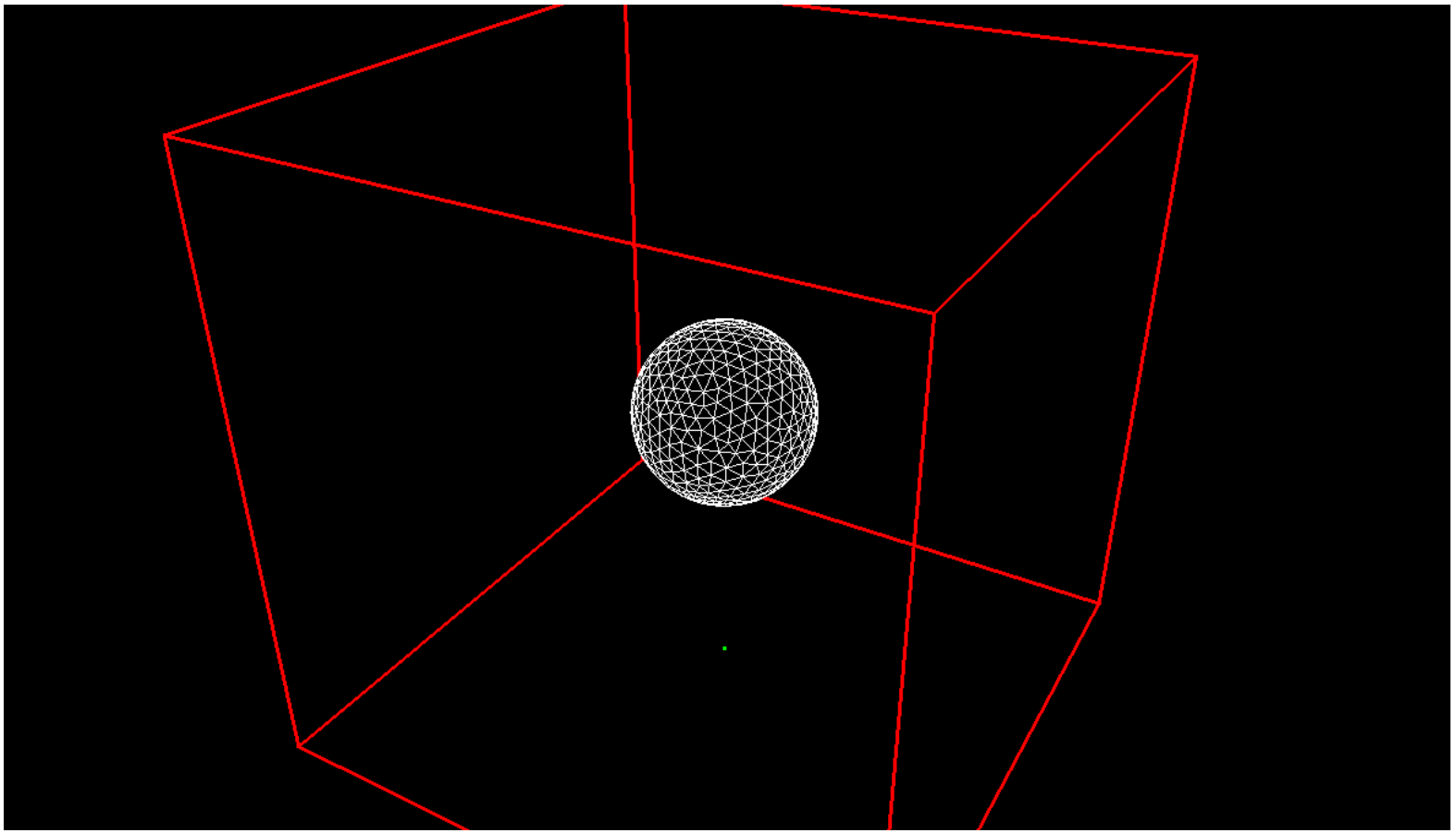
Résultats : exemple de la molécule M_oH (itération 3)



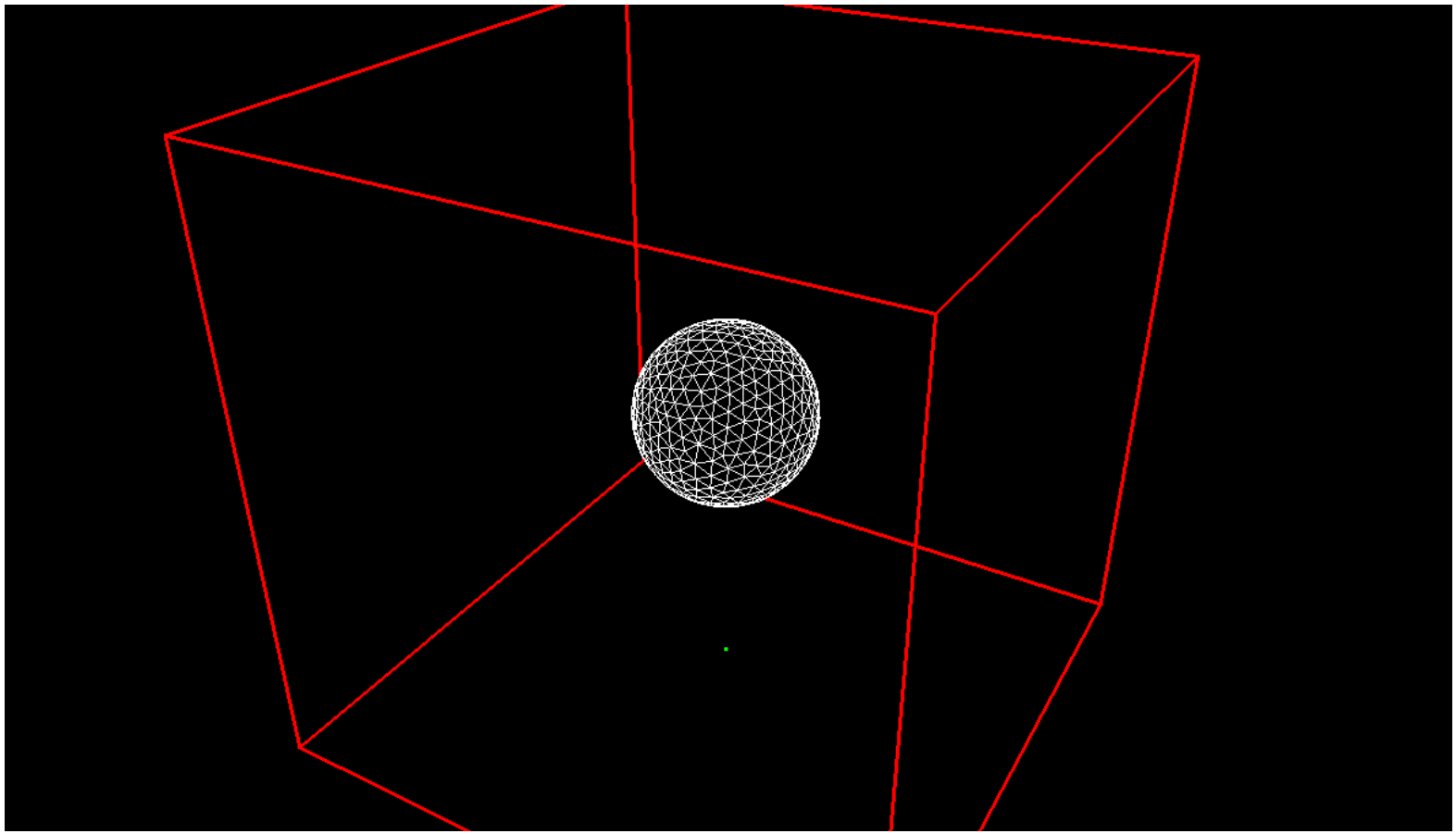
Résultats : exemple de la molécule M_oH (itération 4)



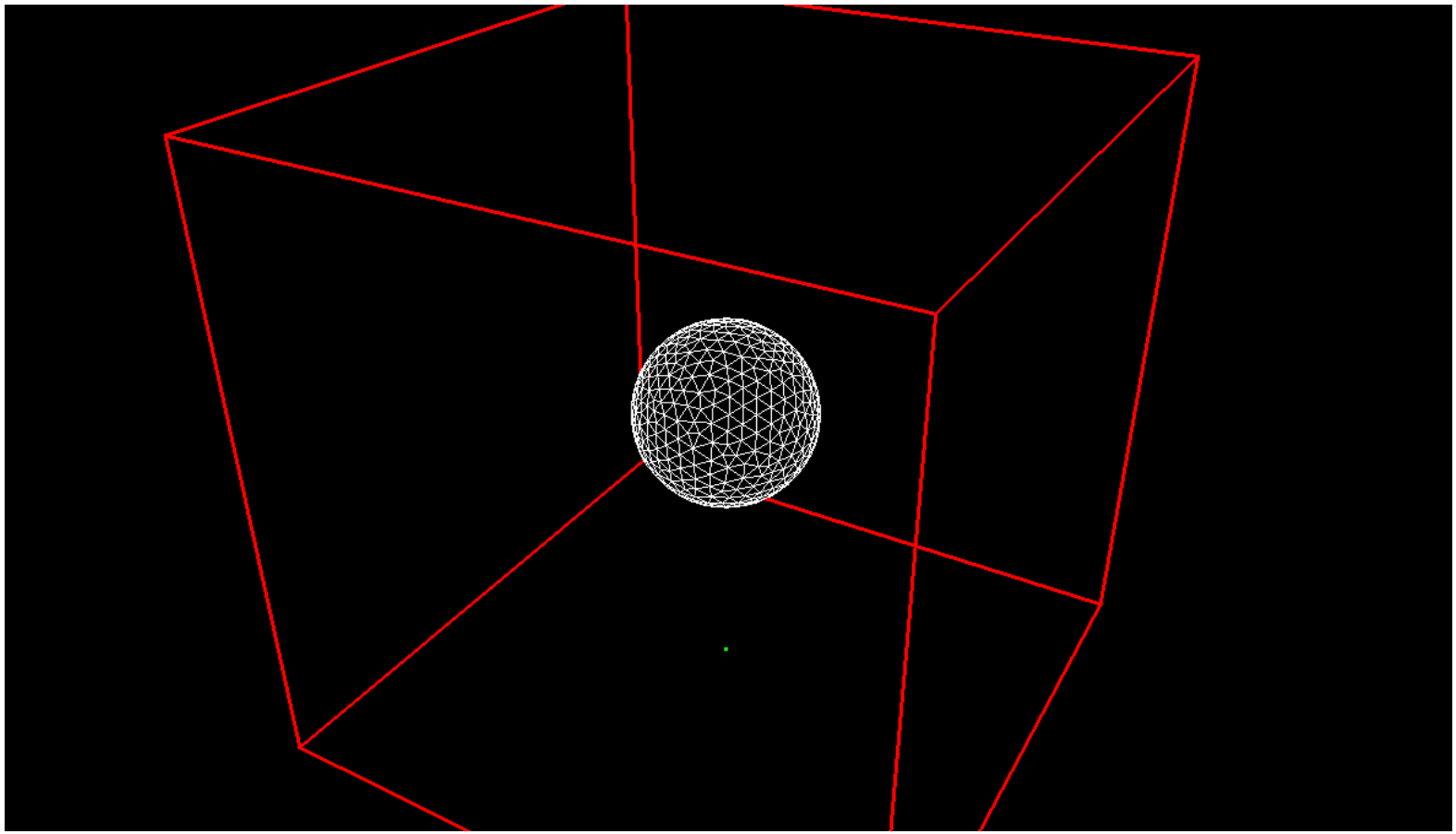
Résultats : exemple de la molécule M_oH (itération 5)



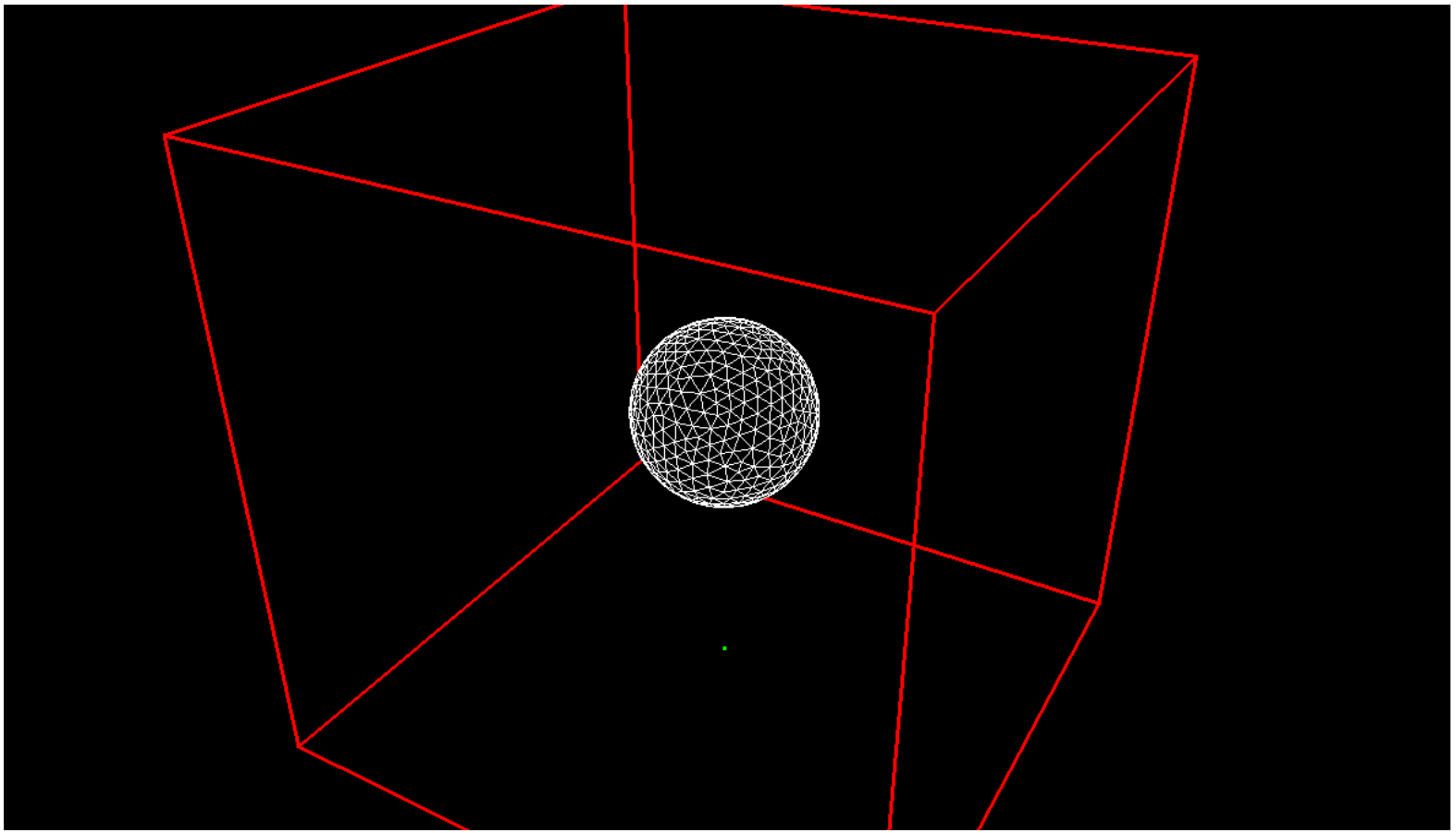
Résultats : exemple de la molécule M_oH (itération 6)



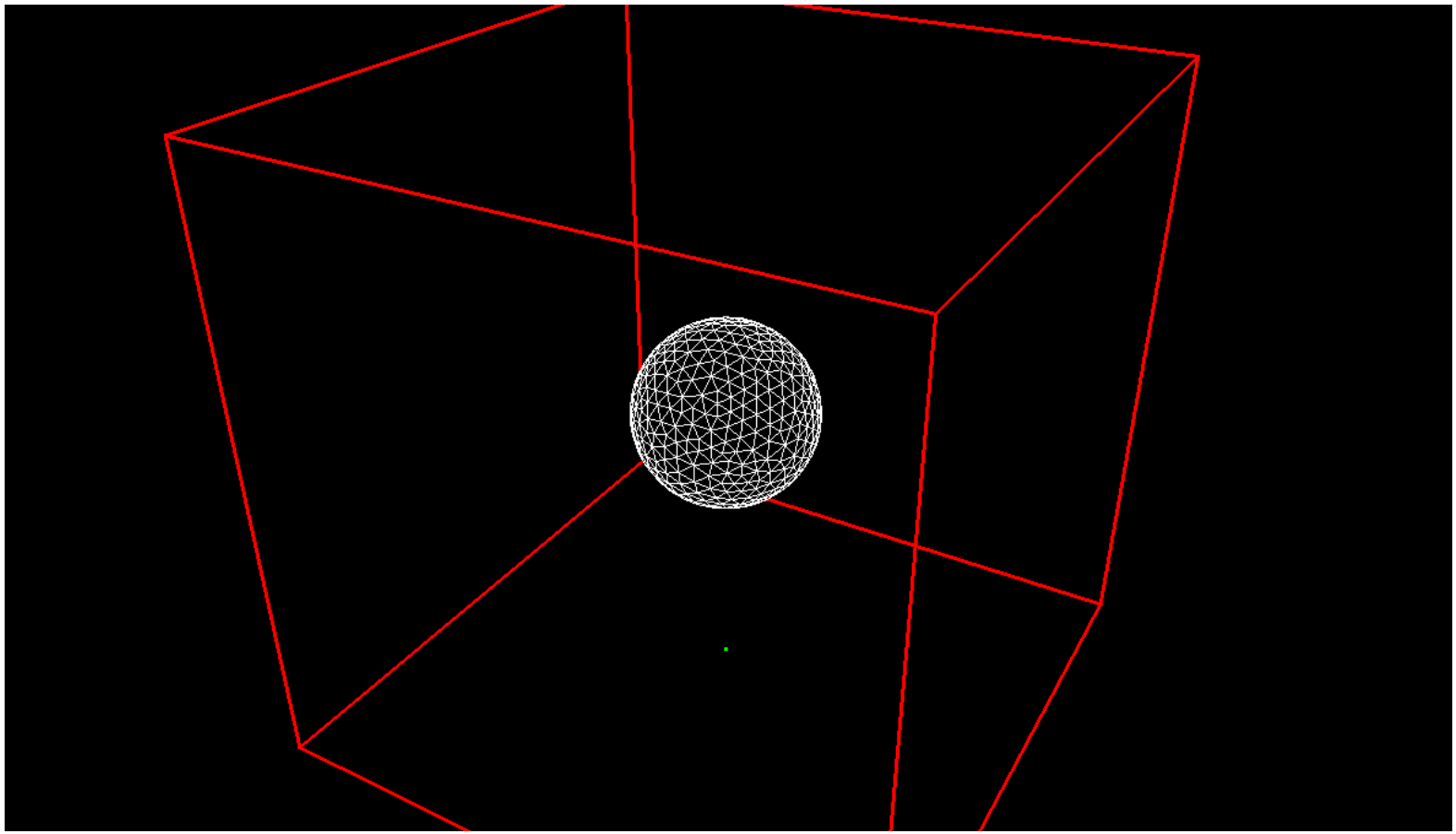
Résultats : exemple de la molécule M_oH (itération 7)



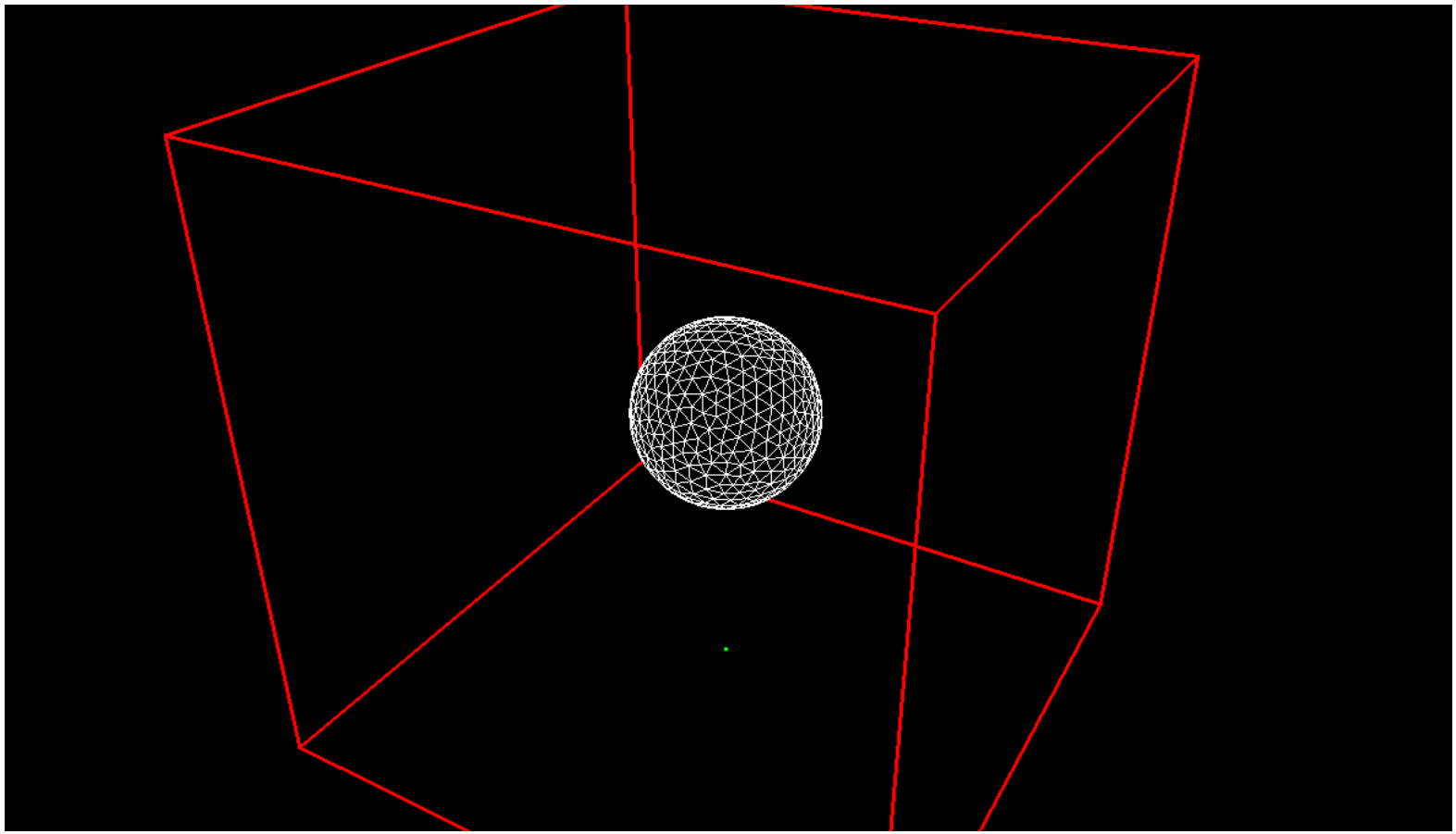
Résultats : exemple de la molécule M_oH (itération 8)



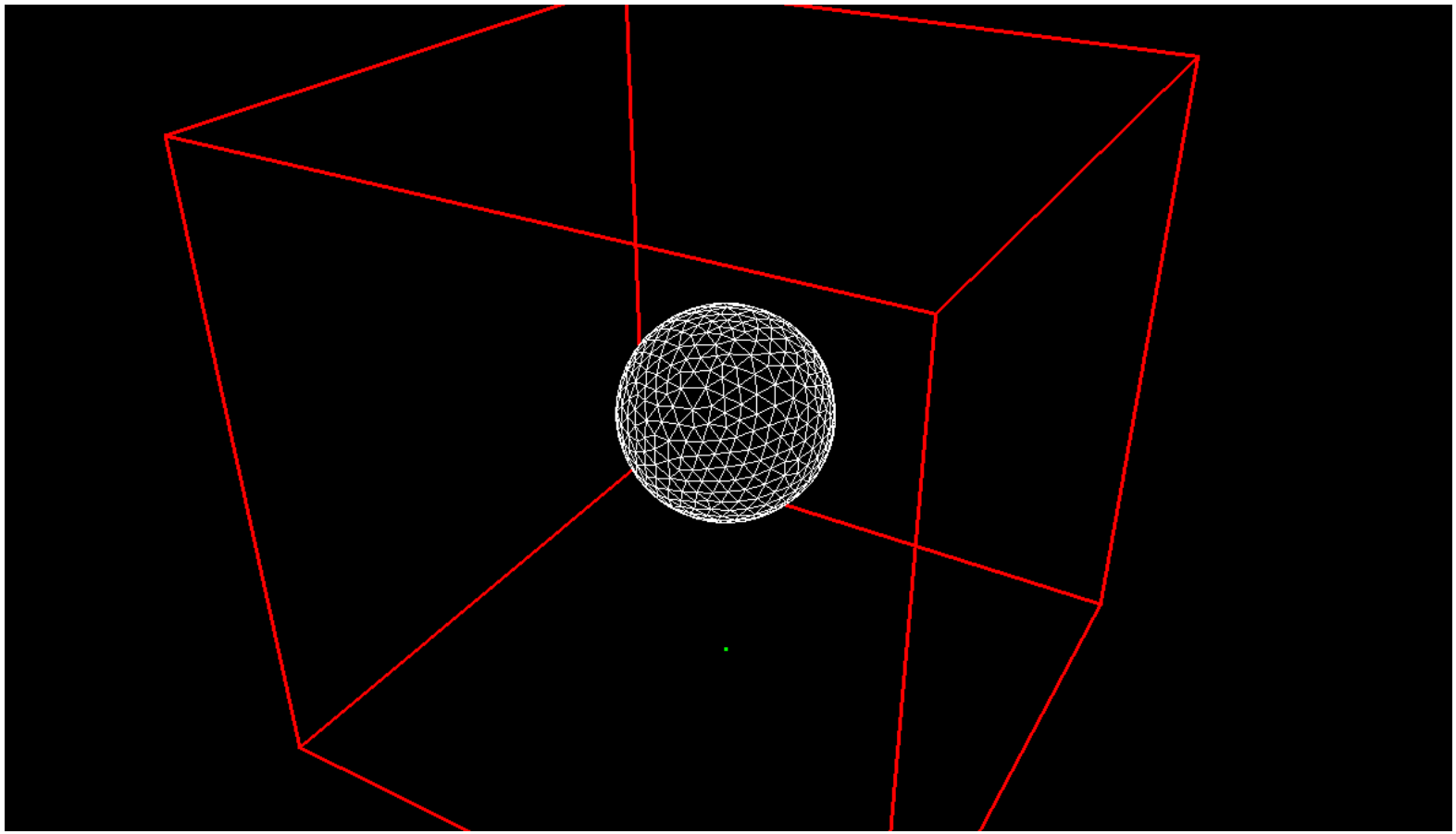
Résultats : exemple de la molécule M_oH (itération 9)



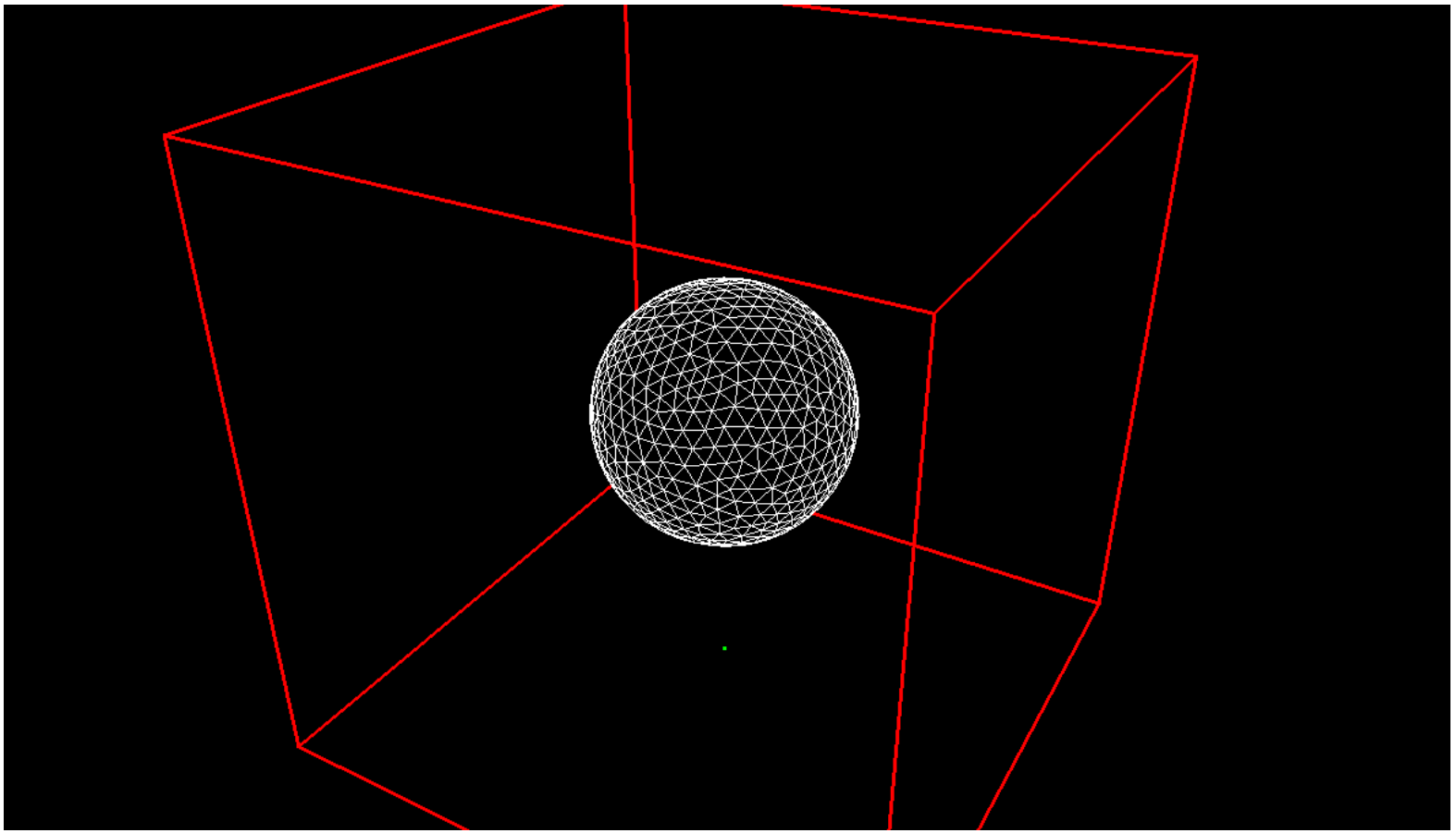
Résultats : exemple de la molécule M_oH (itération 10)



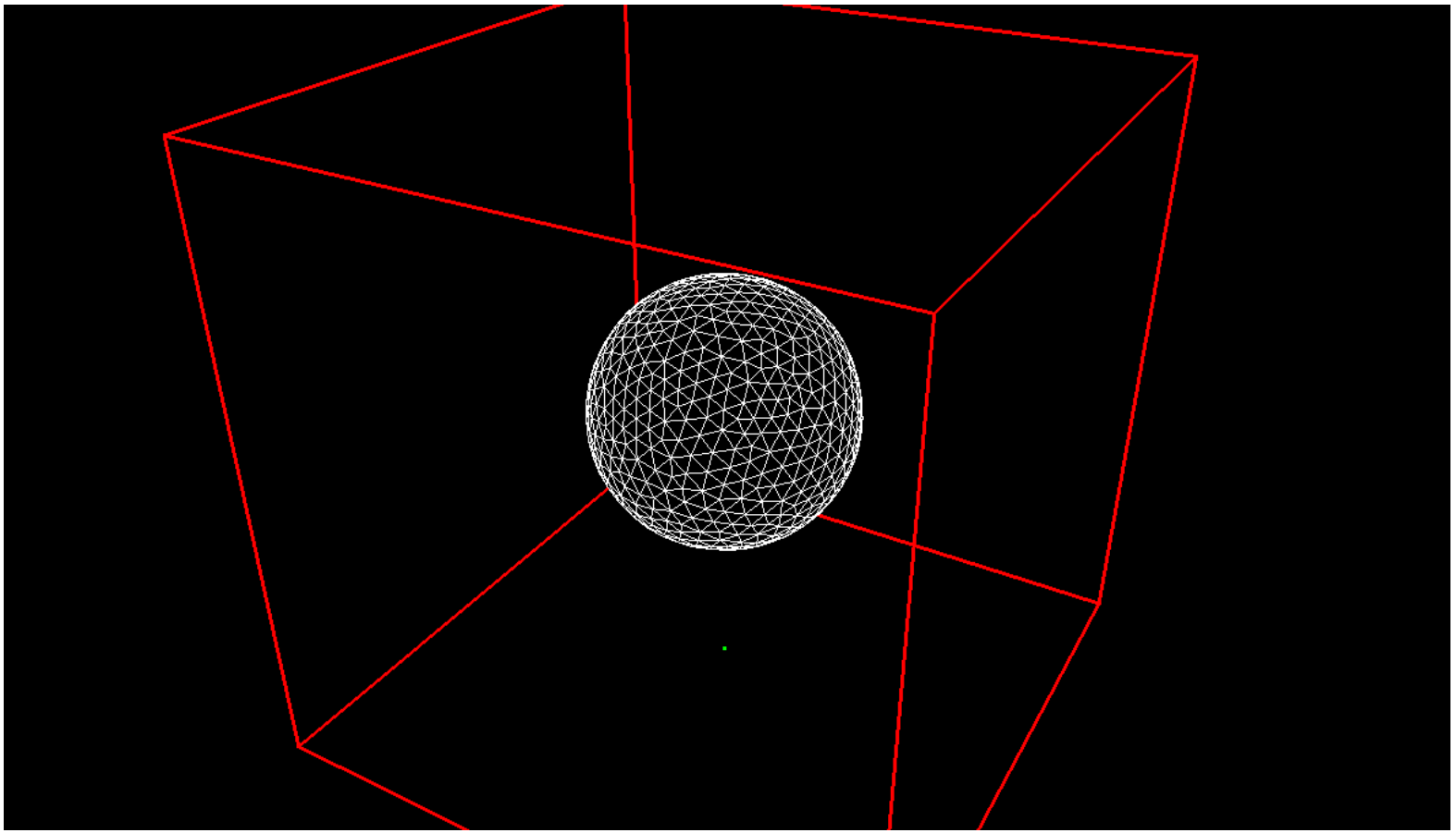
Résultats : exemple de la molécule M_oH (itération 20)



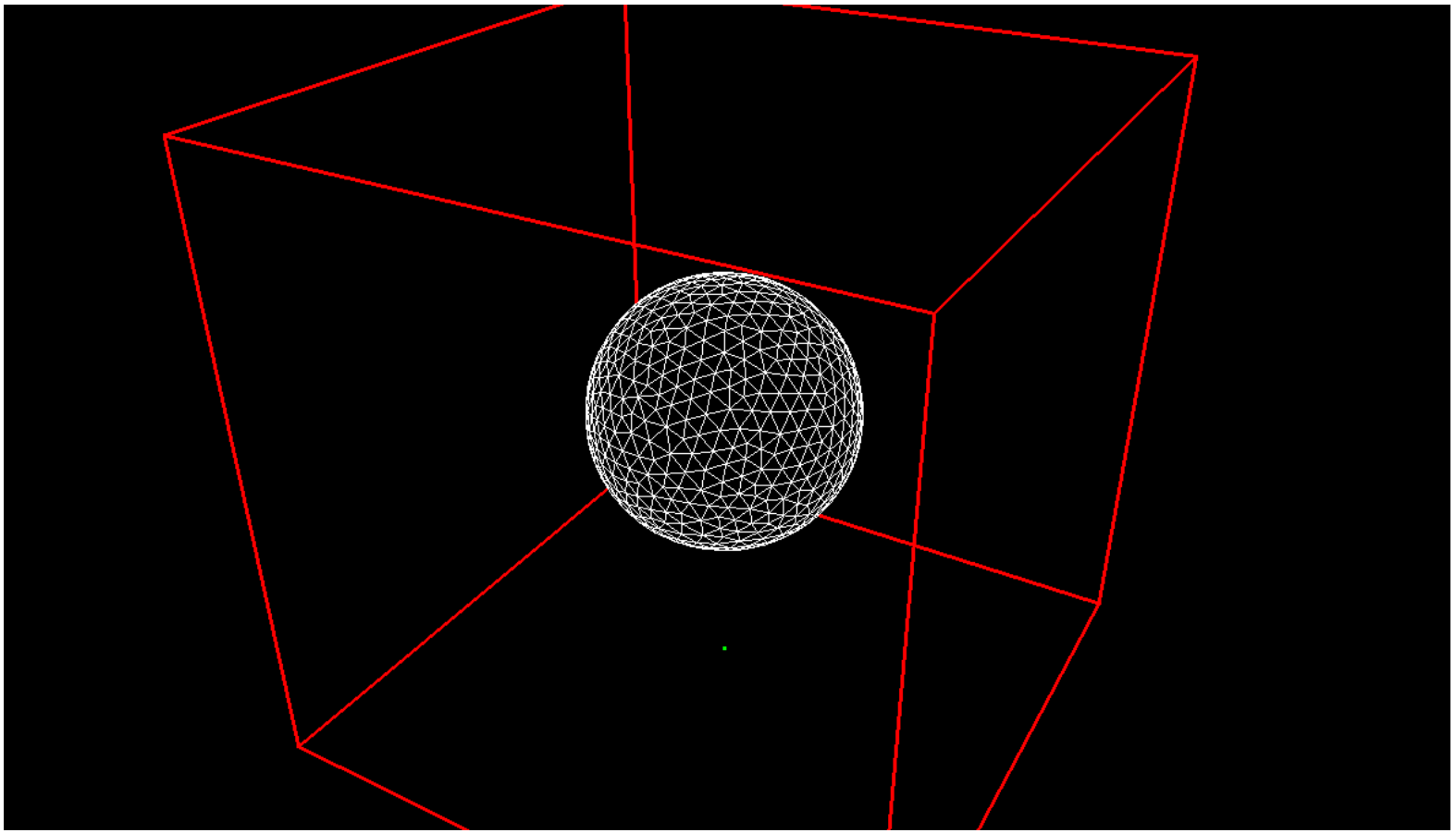
Résultats : exemple de la molécule M_oH (itération 30)



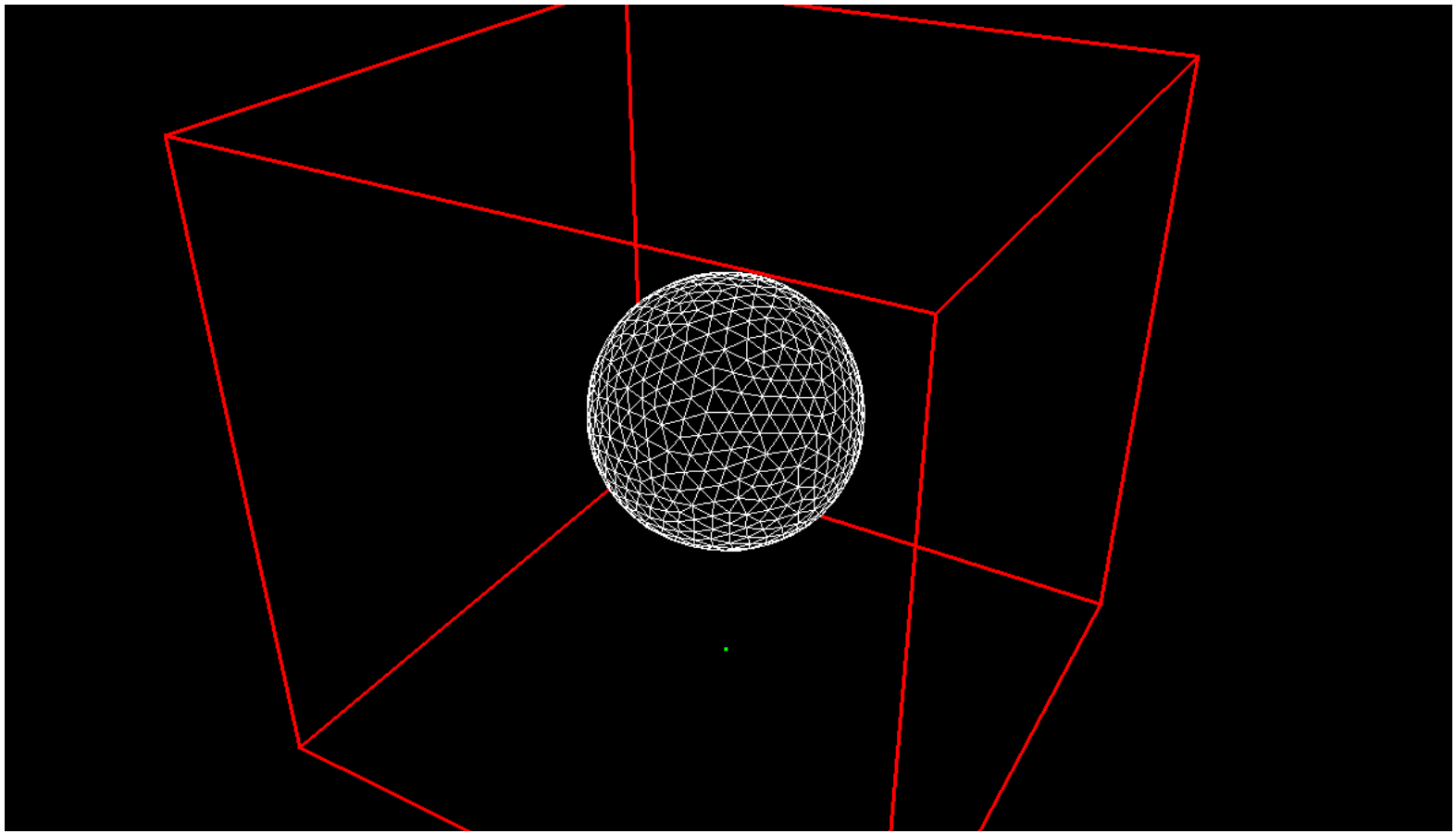
Résultats : exemple de la molécule M_oH (itération 40)



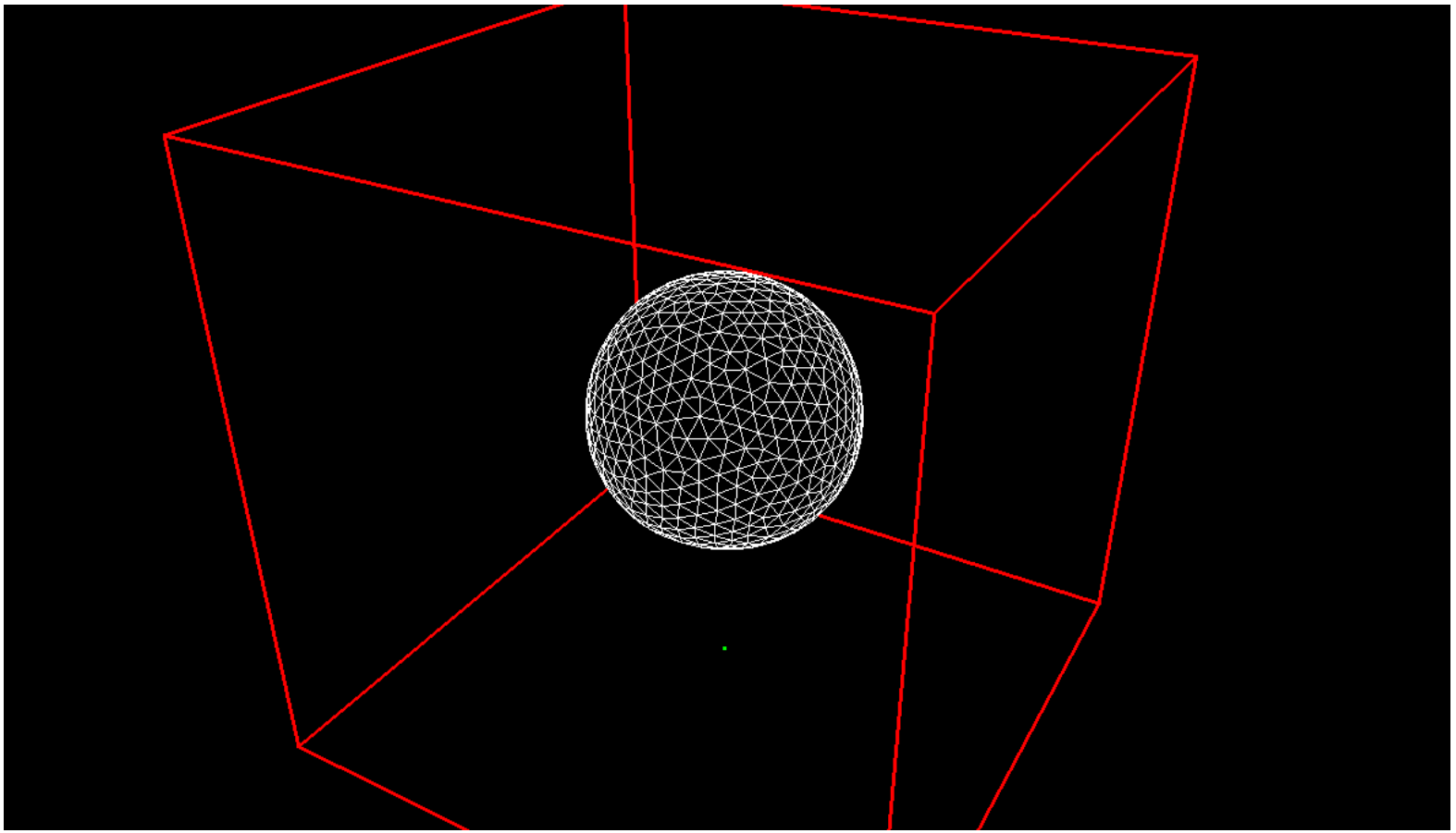
Résultats : exemple de la molécule M_oH (itération 50)



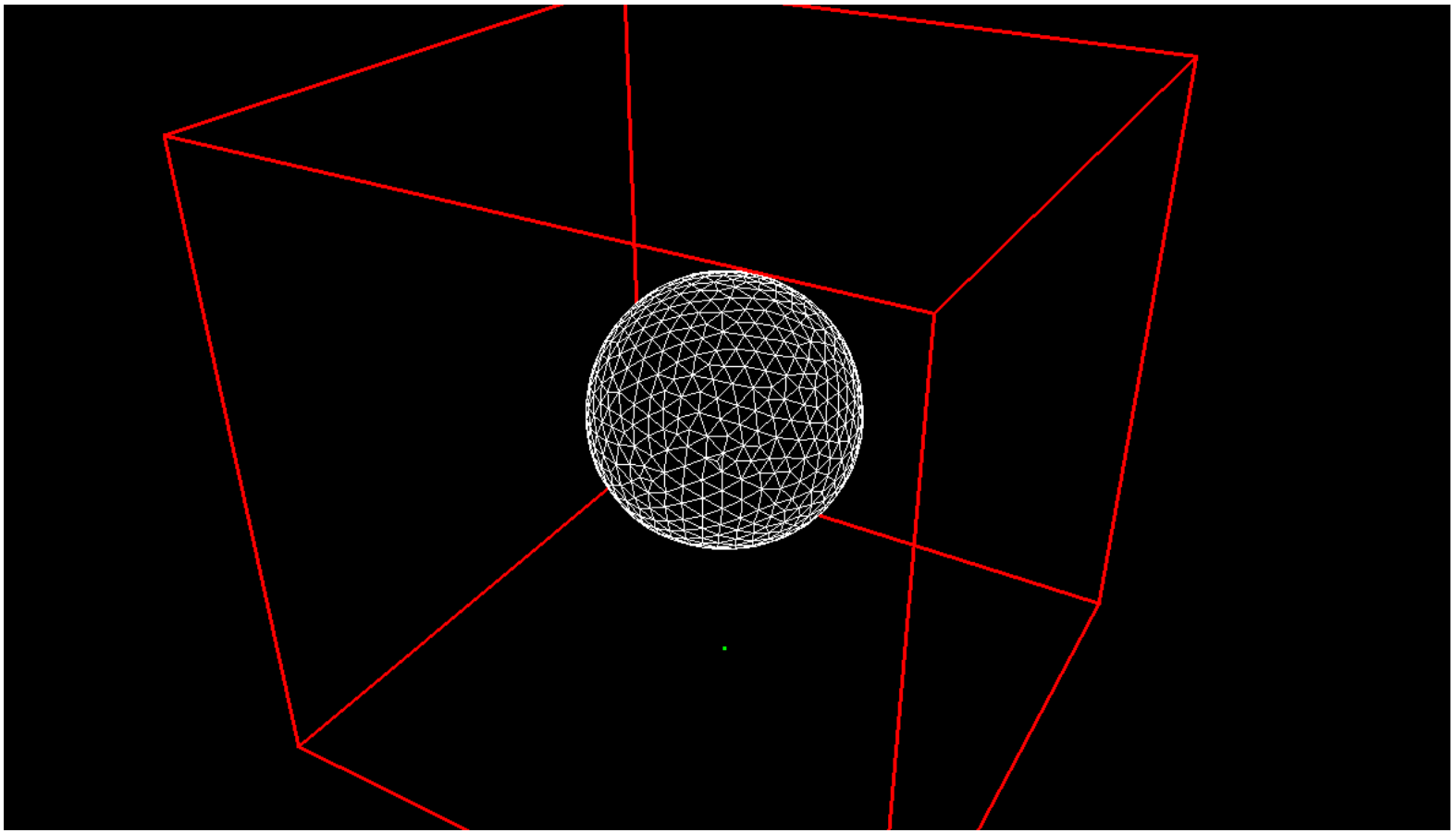
Résultats : exemple de la molécule M_oH (itération 60)



Résultats : exemple de la molécule M_oH (itération 70)

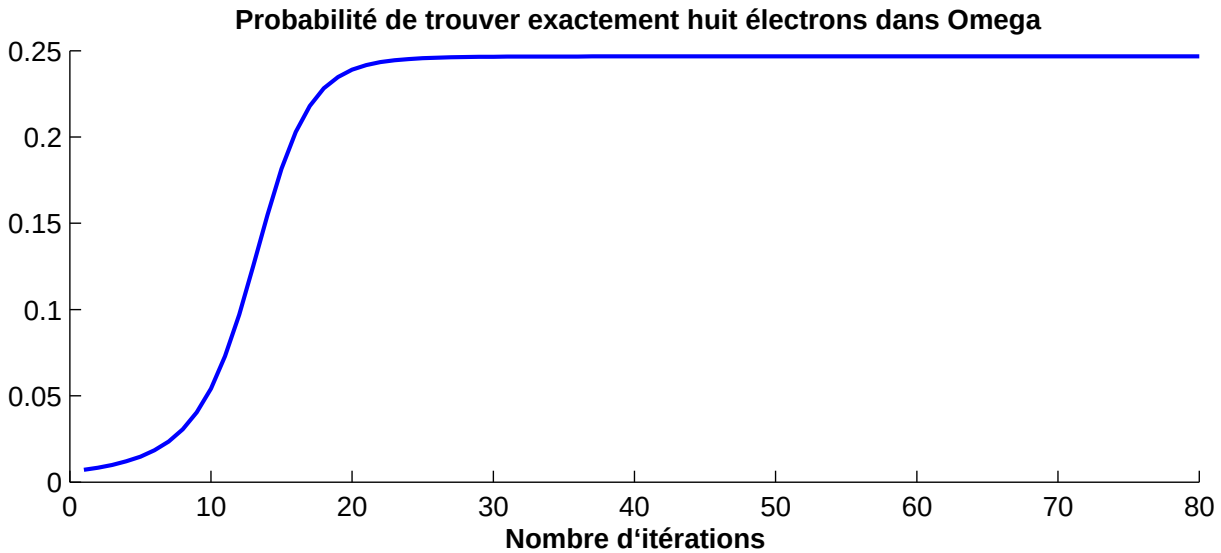


Résultats : exemple de la molécule M_oH (itération 80)



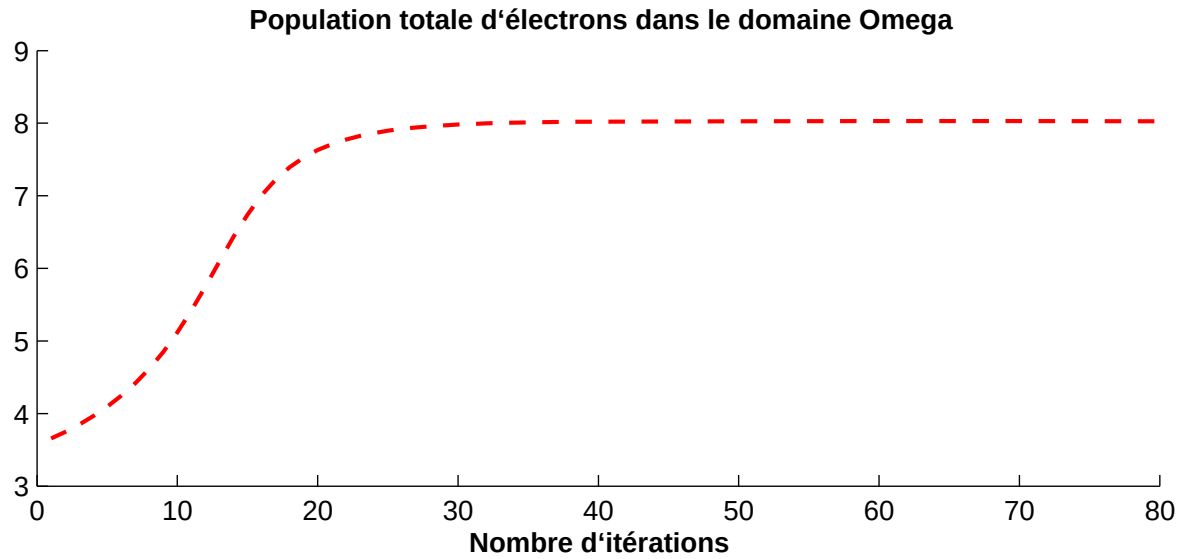
Résultats : exemple de la molécule M_oH

Temps de calcul : 45 sec. par itération soit 7.5 min. pour 10 itérations.

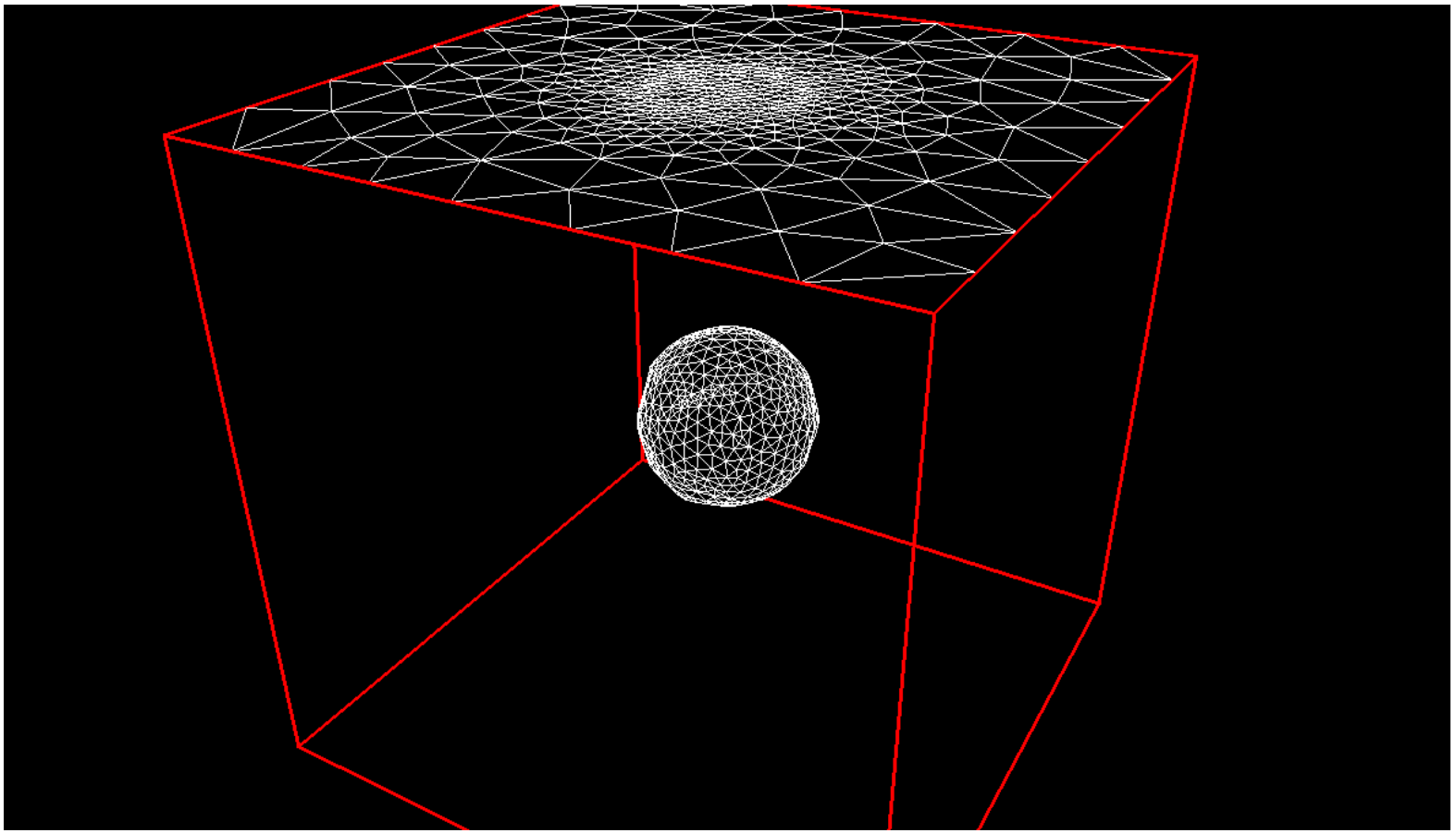


Résultats : exemple de la molécule M_oH

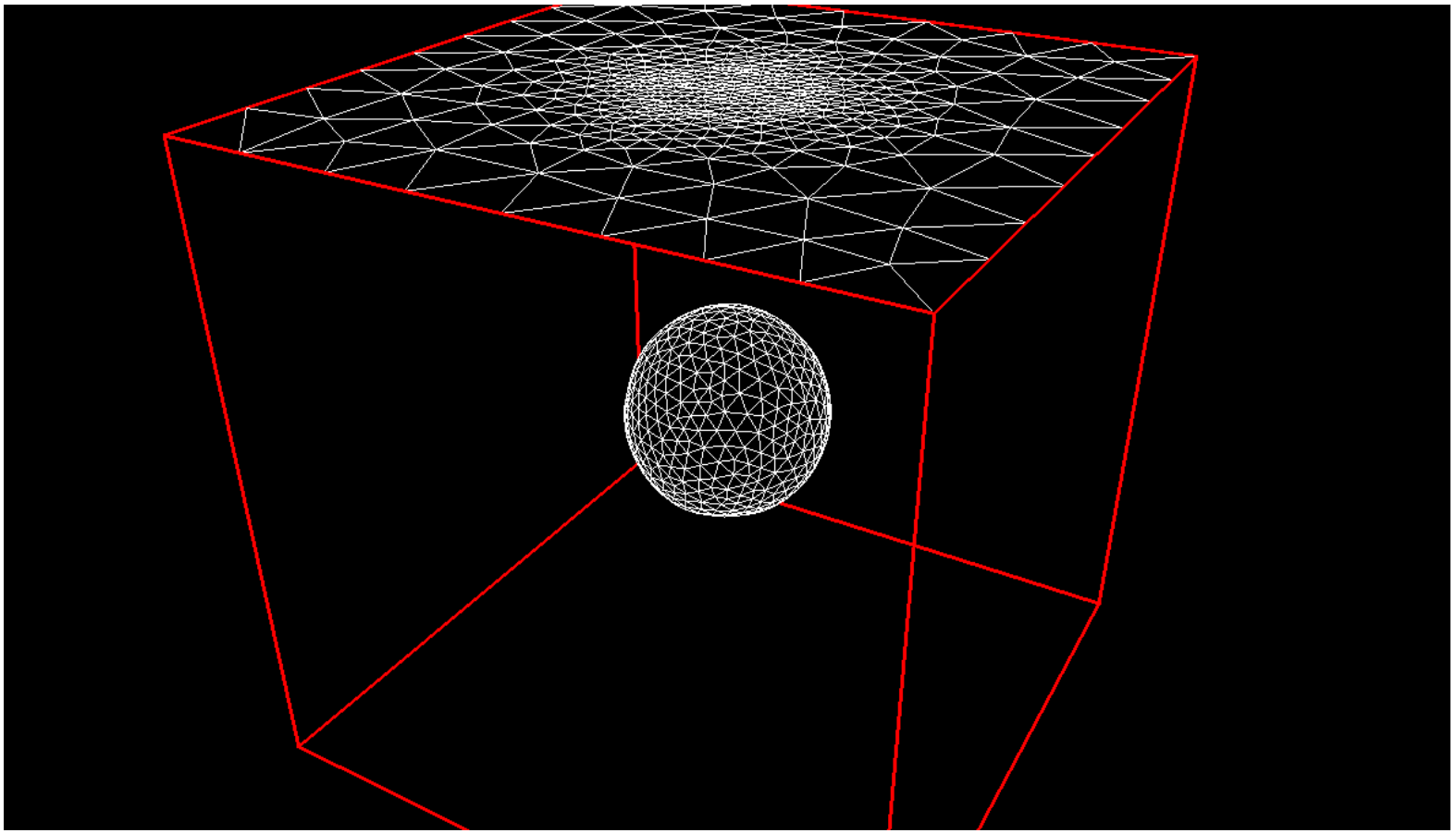
Temps de calcul : 45 sec. par itération soit 7.5 min. pour 10 itérations.



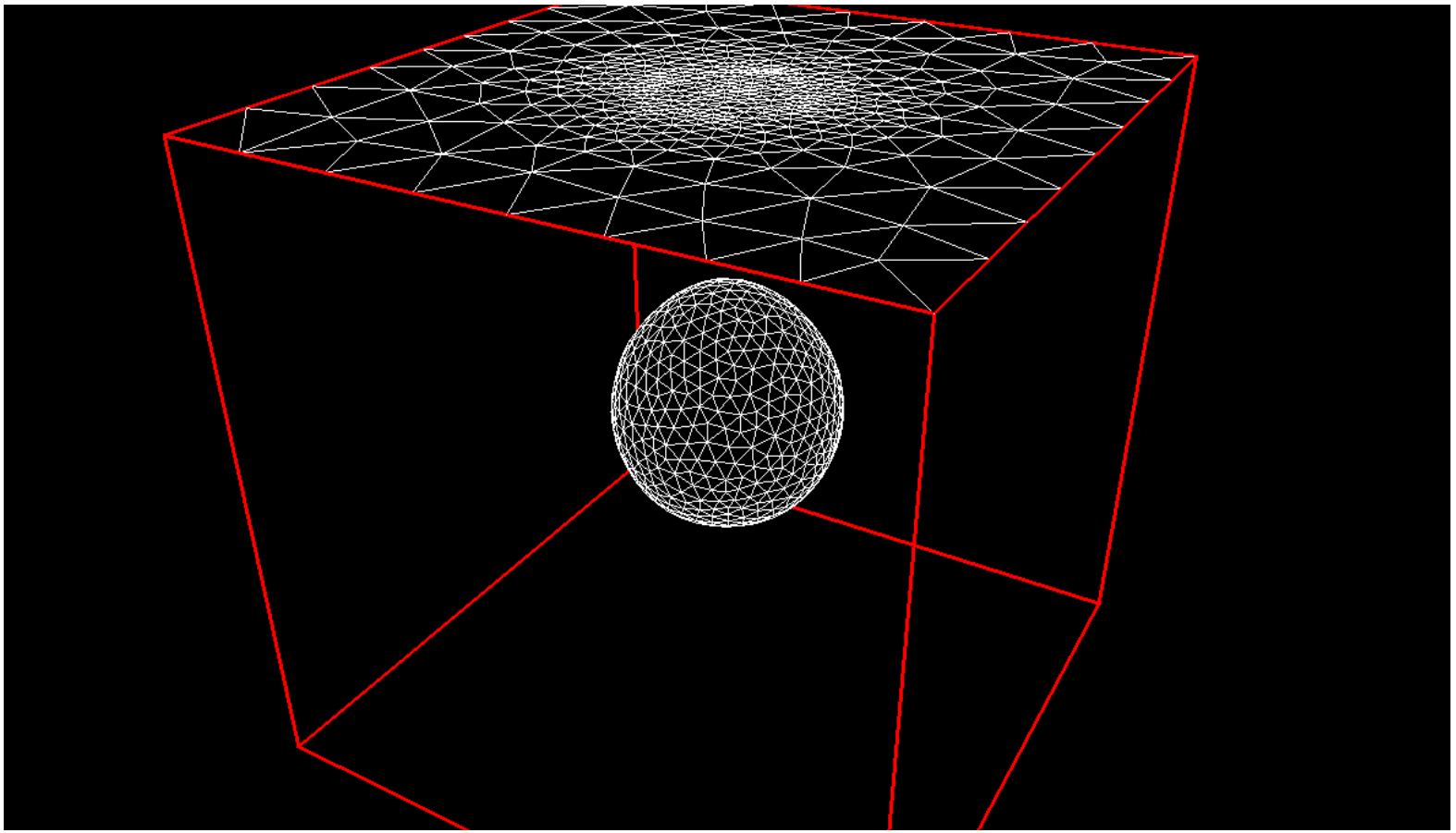
Résultats : exemple de la molécule M_oH (itération 0)



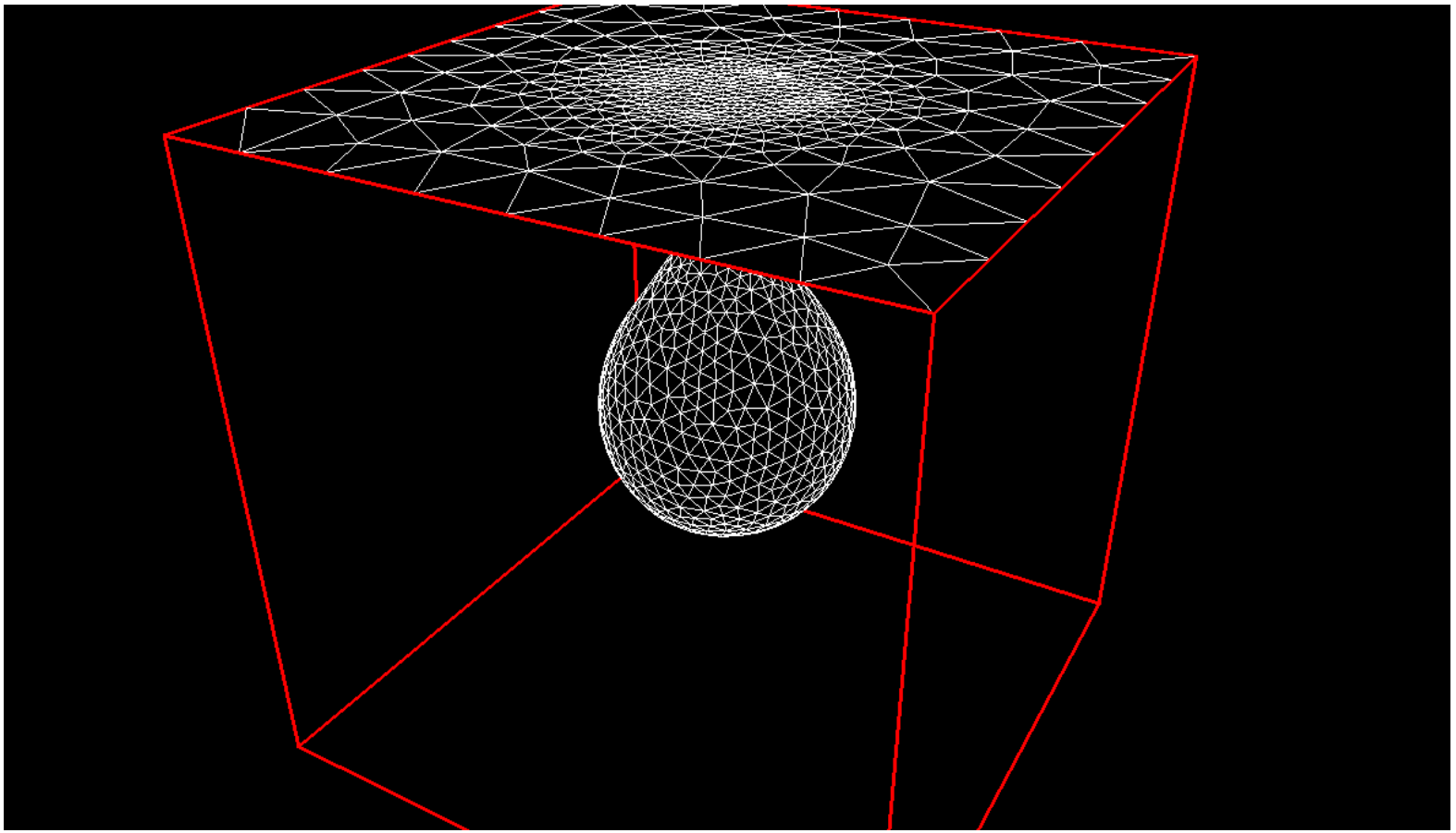
Résultats : exemple de la molécule M_oH (itération 10)



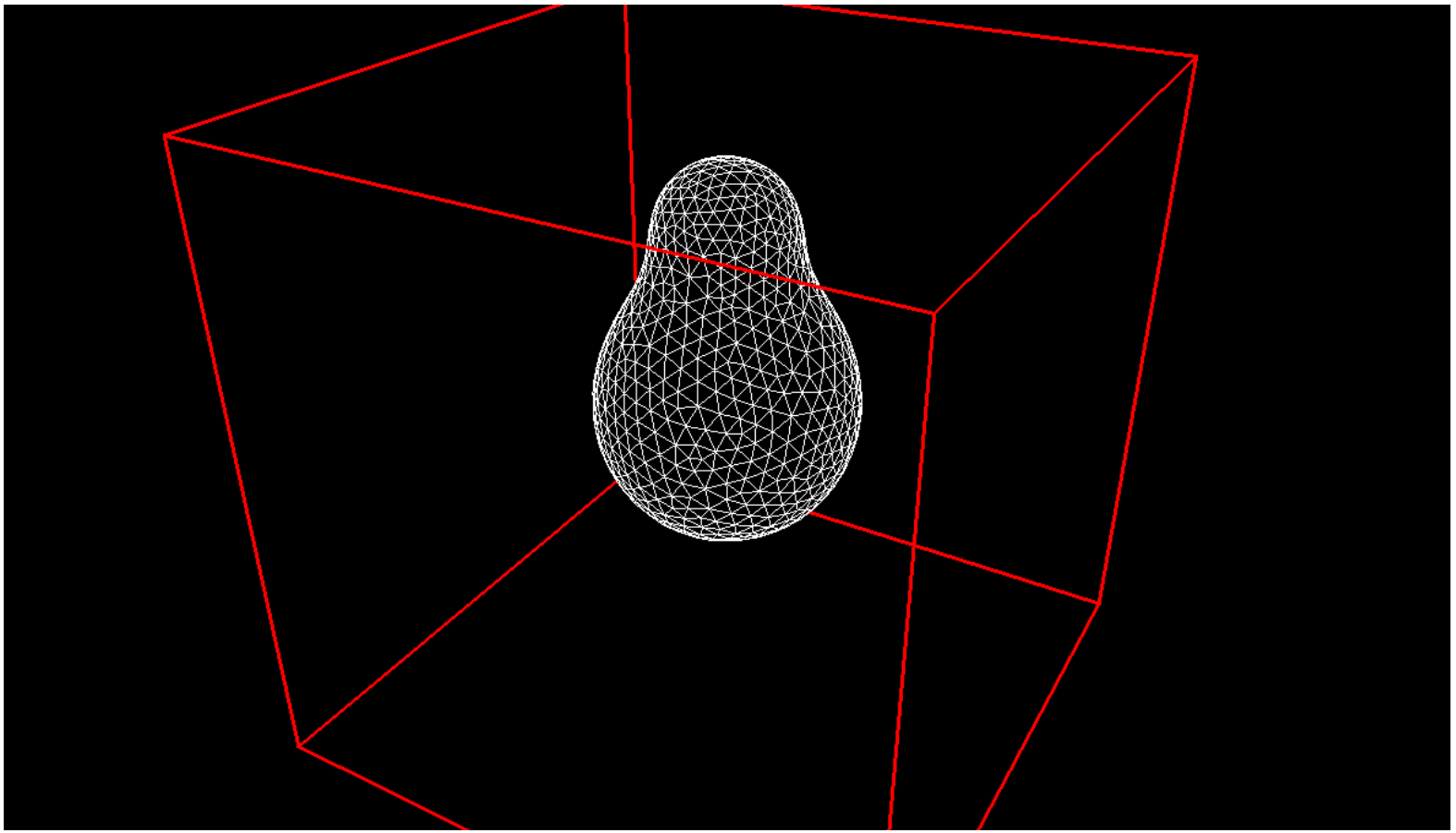
Résultats : exemple de la molécule M_oH (itération 20)



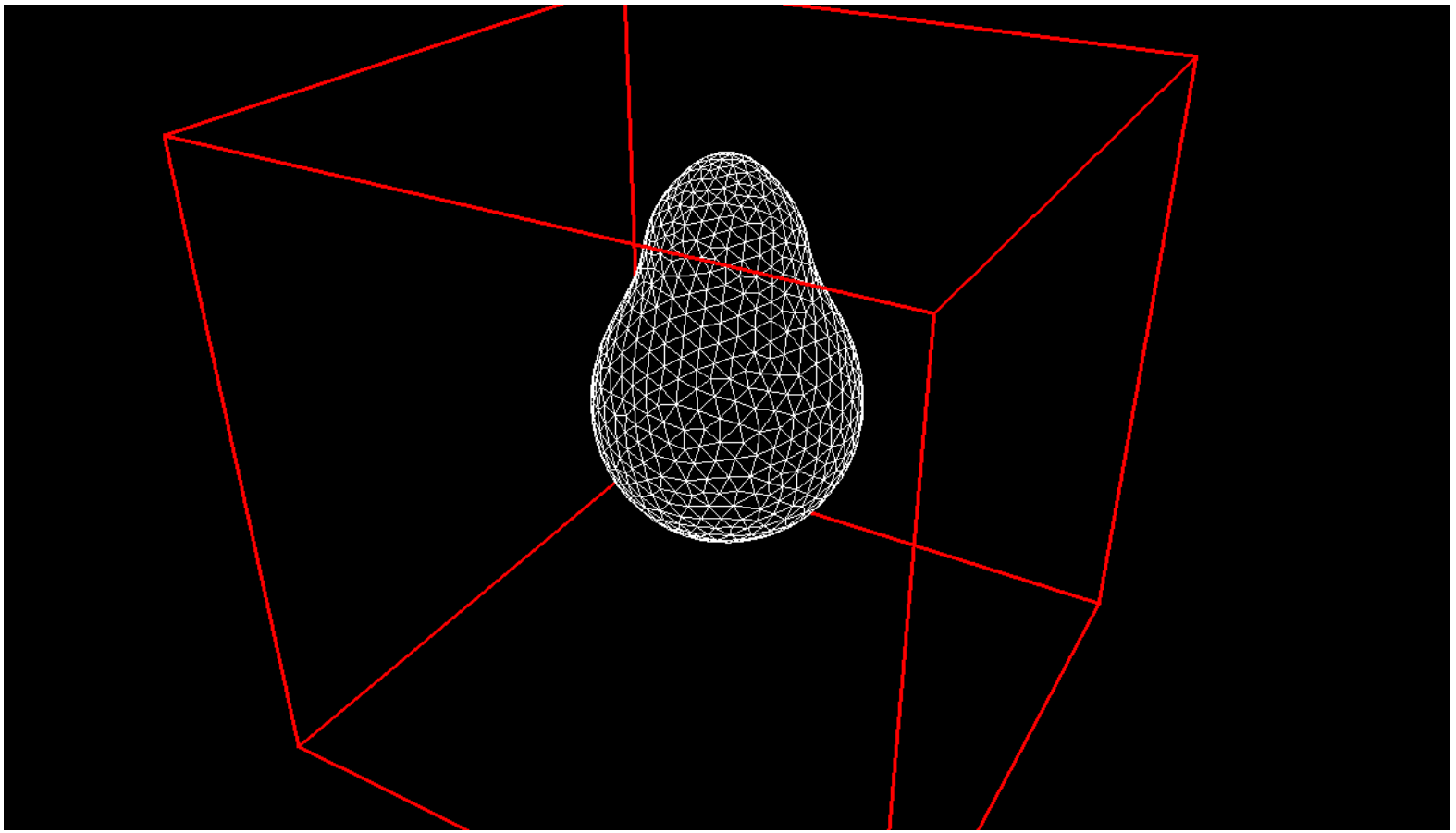
Résultats : exemple de la molécule M_oH (itération 30)



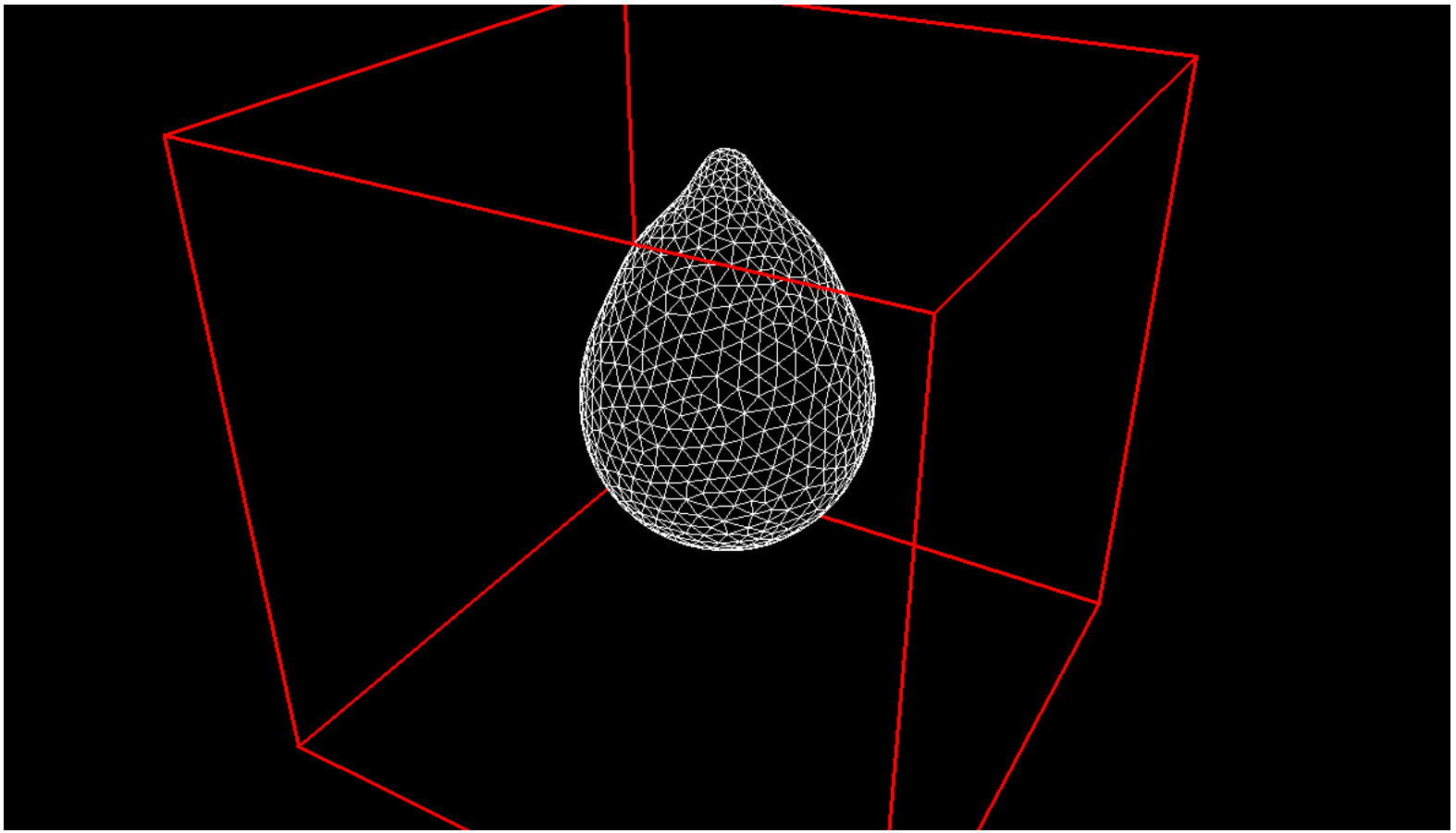
Résultats : exemple de la molécule M_oH (itération 40)



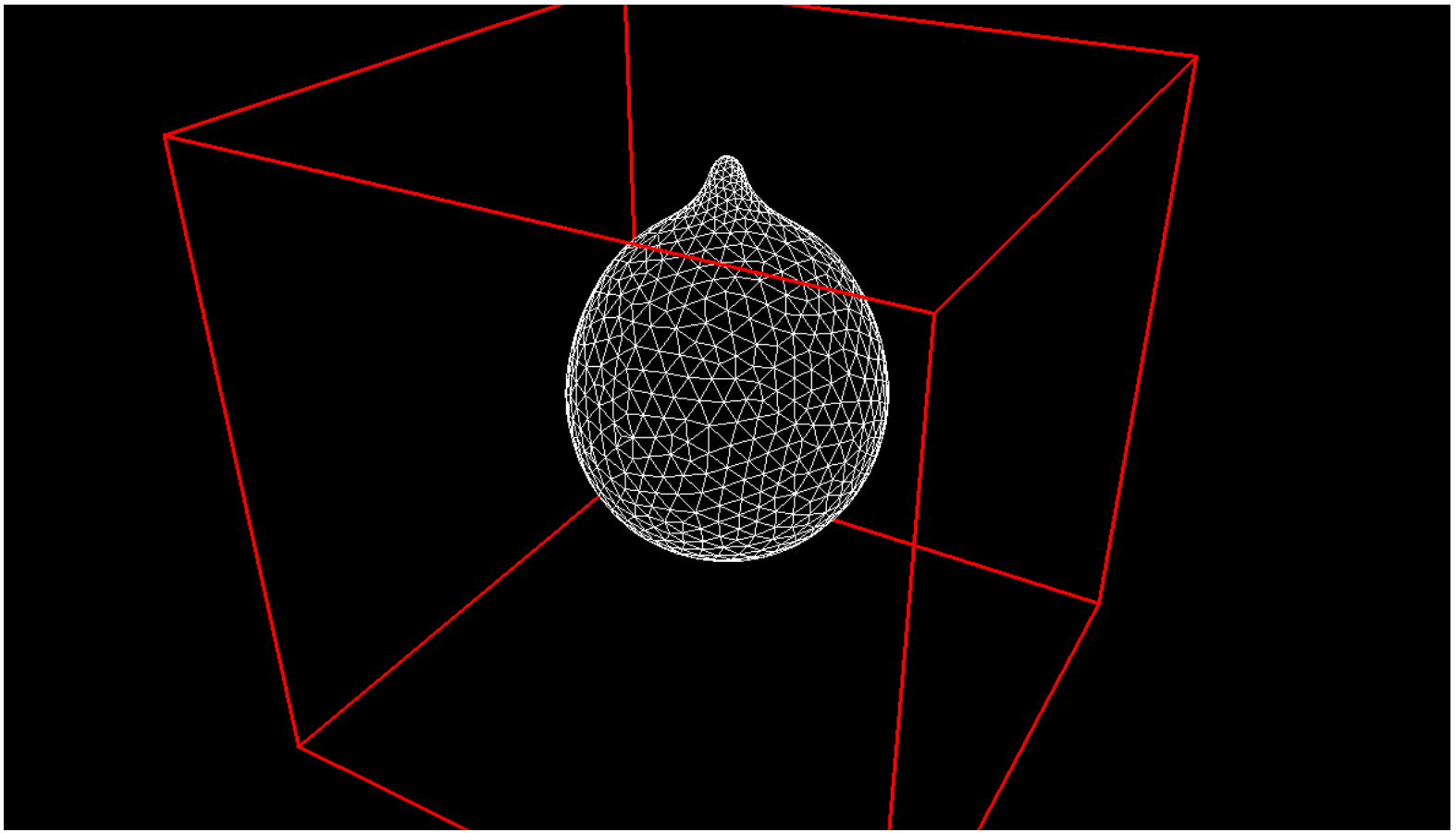
Résultats : exemple de la molécule M_oH (itération 50)



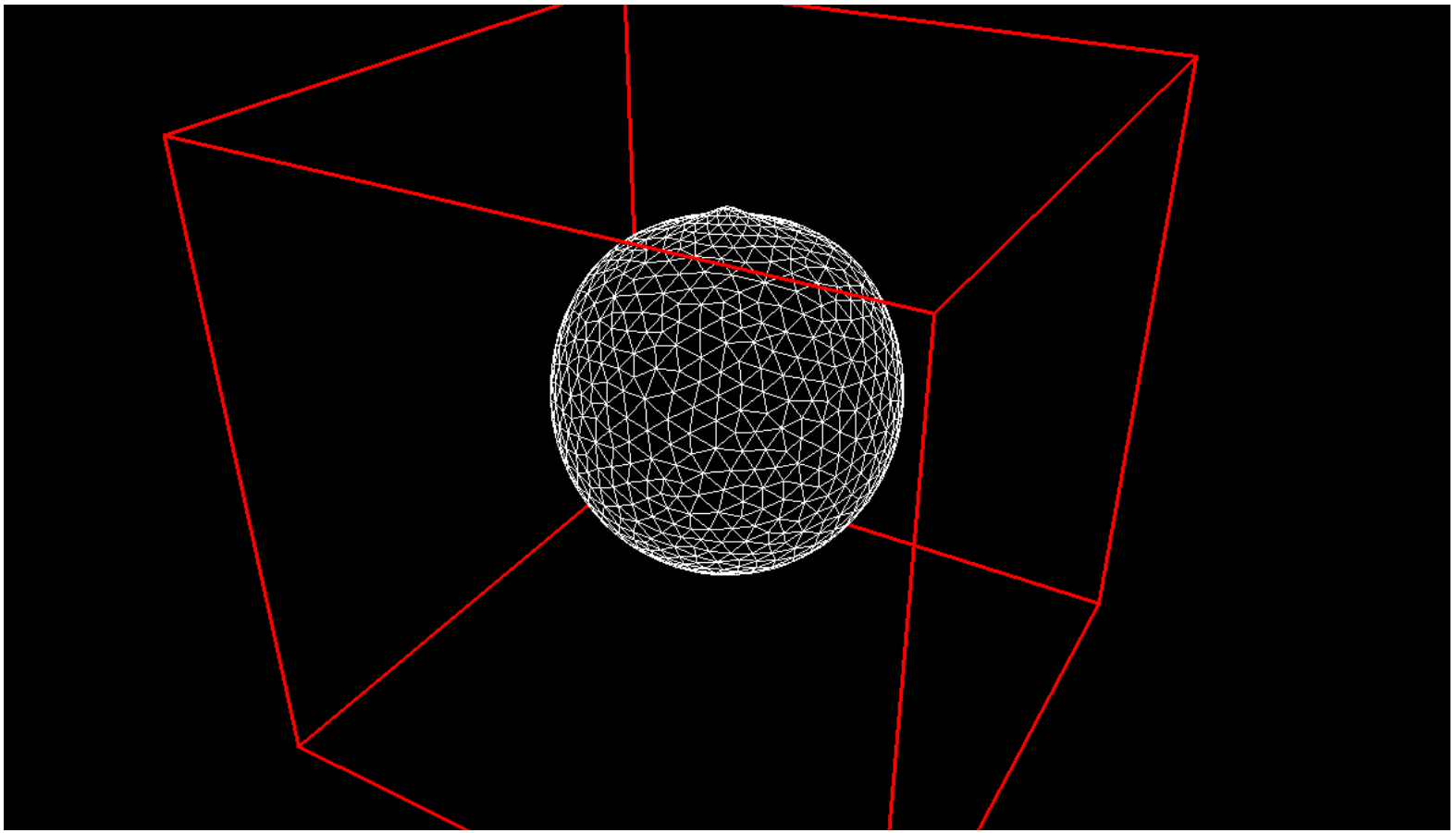
Résultats : exemple de la molécule M_oH (itération 100)



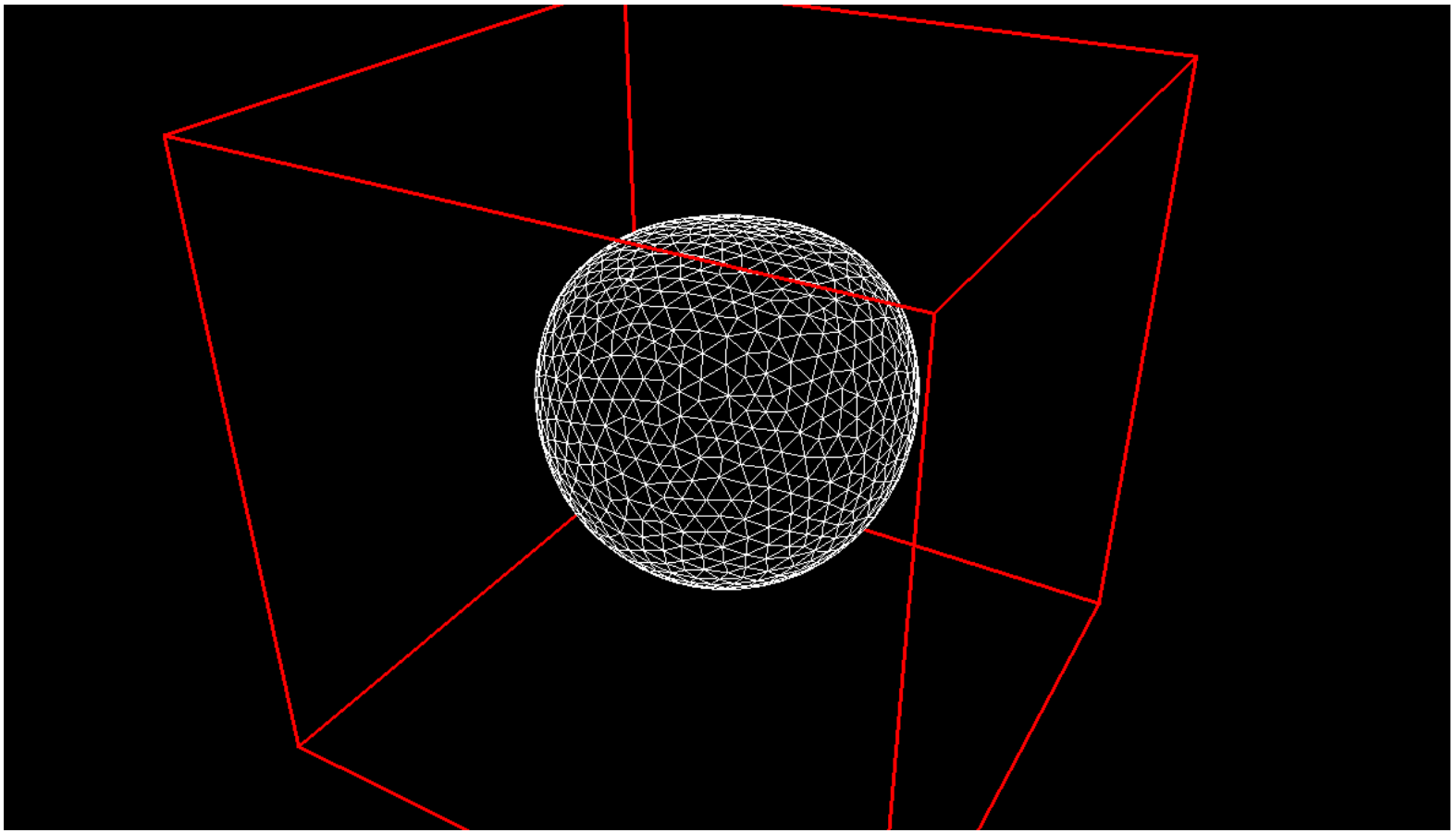
Résultats : exemple de la molécule M_oH (itération 150)



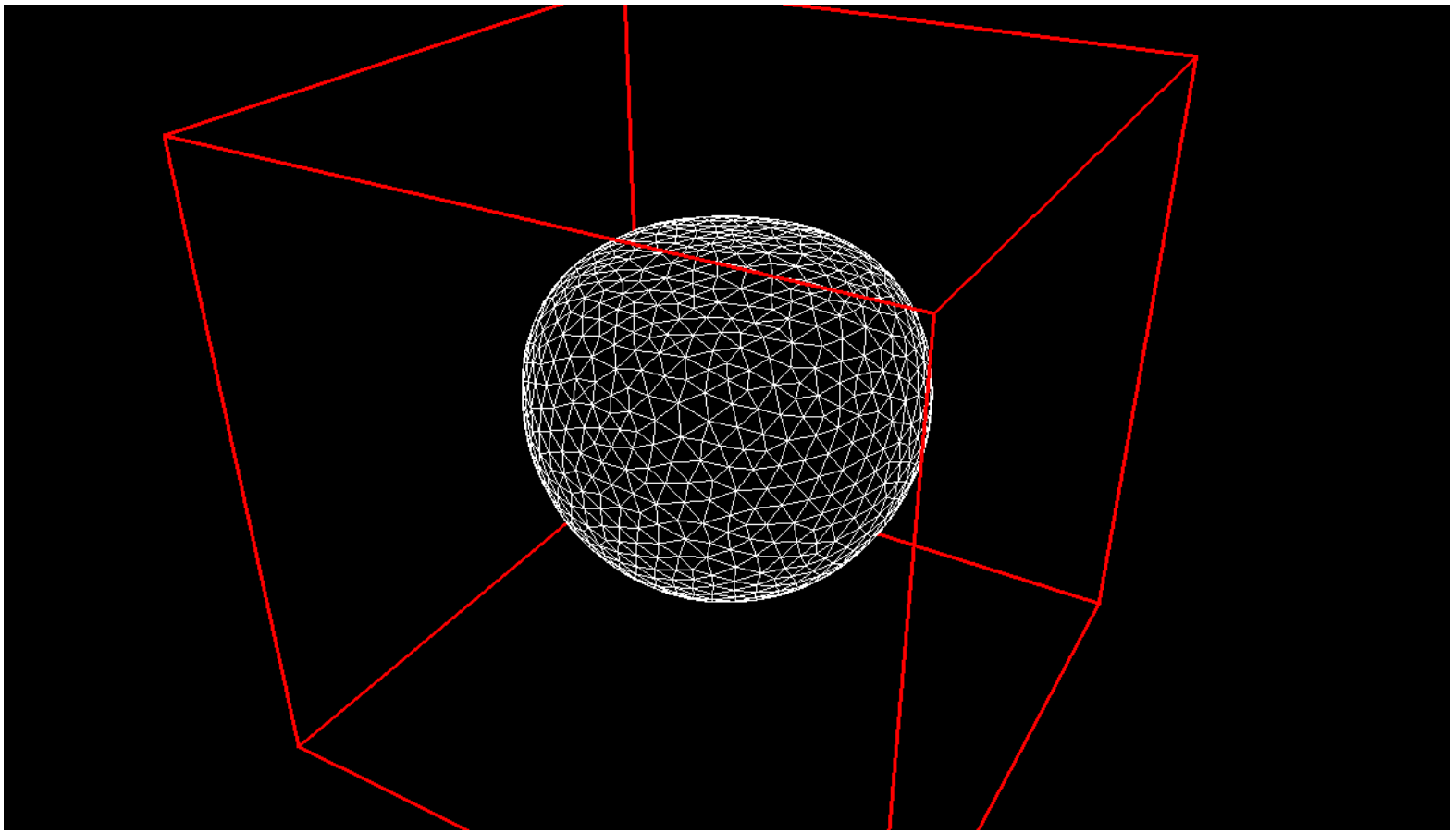
Résultats : exemple de la molécule M_oH (itération 200)



Résultats : exemple de la molécule M_oH (itération 250)

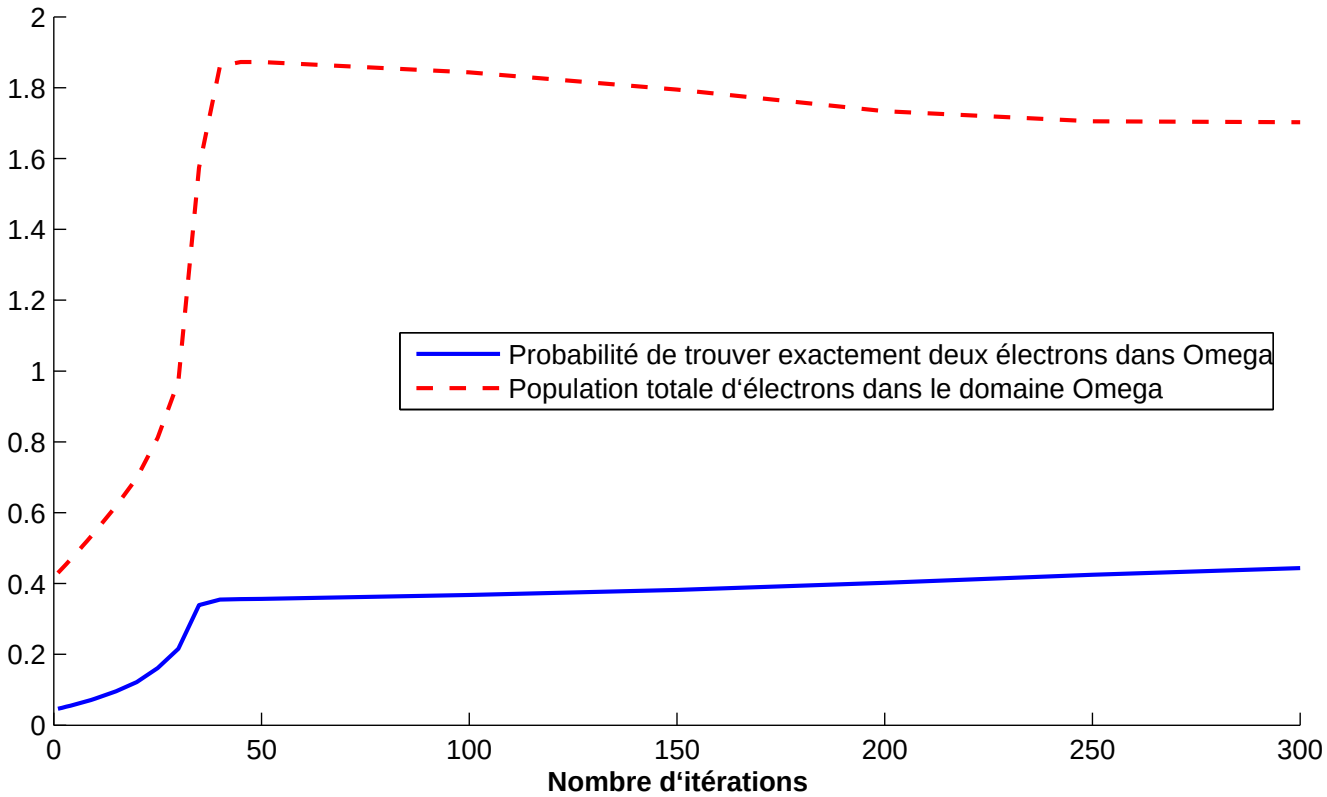


Résultats : exemple de la molécule M_oH (itération 300)



Résultats : exemple de la molécule M_oH

Temps de calcul : 30 sec. par itération soit 5 min. pour 10 itérations.



- Le code actuellement disponible nécessite de recompiler à chaque modification de paramètres. On est en train de réfléchir à le rendre facile à utiliser et à la documenter (guide utilisateur)
- Le code doit maintenant être testé (test unitaire) informatiquement et avec des exemples chimiques réels plus avancés. Pour l'algorithme cubique, les résultats sont similaires à ceux donnés par un autre programme MPDs.
- Théoriquement, on sait comment adapter ce formalisme à une fonction d'onde multi-déterminant. De nombreuses difficultés surviennent alors (matrice non-symétrique, ne vaut plus l'identité sur $\mathbb{R}^3$, problème de boîte, de parallélisation).
- On souhaite mettre en place dans le cas simple déterminant une méthode d'ordre deux de type Newton afin d'accélérer la convergence
- Des résultats d'existence mathématique sur le caractère bien posé du problème ont été développés. Un aperçu d'algorithme novateur est en train de voir le jour (mixte tétra-cube, par densité).
- Vos remarques sont les bienvenues !