Titre: Title:	Conception et optimisation d'un cathéter avec électrode oesophagienne sélective à l'EMGdi et deux capteurs de pression
Auteur: Author:	Luc Romain
Date:	2000
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Romain, L. (2000). Conception et optimisation d'un cathéter avec électrode oesophagienne sélective à l'EMGdi et deux capteurs de pression [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/6943/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/6943/
Directeurs de recherche: Advisors:	Mohamad Sawan
Programme: Program:	Non spécifié

UNIVERSITÉ DE MONTRÉAL

Conception et optimisation d'un cathéter avec électrode œsophagienne sélective à l'EMGdi et deux capteurs de pression.

LUC ROMAIN
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE ET GÉNIE INFORMATIQUE)
MAI 2000

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

Conception et optimisation d'un cathéter avec électrode oesophagienne sélective à l'EMGdi et deux capteurs de pression.

Présenté par : ROMAIN Luc

En vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

A été dûment accepté par le jury d'examen constitué de :

M. SAVARD Pierre, Ph.D., président du comité

M. SAWAN Mohamad, Ph.D., membre et directeur de recherche

M. LAURIN Jean-Jacques, Ph.D., membre

Remerciements

Je tiens à remercier M. François Bellemare, de m'avoir proposé ce projet de recherche ainsi que pour l'aide financière tout au long de ce travail.

Je tiens également à remercier mon directeur de recherche, M. Mohamad Sawan, professeur à l'École Polytechnique, de m'avoir guidé durant l'accomplissement de mes travaux.

Un merci spécial à Julie qui m'a épaulé tout au long de mes travaux de recherche.

Finalement, je dois remercier mes parents, Carl et Carole ainsi que ma sœur Céline de m'avoir encouragé durant mes études.

Résumé

La lecture de l'électromyogramme (EMG) du diaphragme ainsi que de la pression trans diaphragmatique est susceptible de fournir un grand nombre de renseignements sur le système respiratoire. Le diaphragme est le muscle principal du système respiratoire. Il est stimulé à plusieurs endroits sur sa surface par le nerf phrénique suite à une excitation provenant du cerveau. Lors d'une respiration, des potentiels d'actions ordonnent la contraction du diaphragme. Ces potentiels d'action de très faible amplitude peuvent être captés tout près du sphincter œsophagien à l'aide d'une électrode œsophagienne. Une différence de pression est exercée entre l'estomac et l'œsophage lors d'une respiration de part et d'autre du sphincter œsophagien. Cette différence de pression peut être captée avec des transducteurs de pression logés sur une deuxième sonde œsophagienne.

Un problème significatif limite l'efficacité et la précision des lectures de l'EMG_{di} : la contamination par l'artefact cardiaque appelé l'électrocardiogramme (ECG). Le cœur est un organe dont le fonctionnement est synchrone ; toutes les fibres du cœur sont activées en moins de 90 msec. La contraction du cœur débute par une impulsion électrique dans le nœud sinusal. Le nœud sinusal est l'endroit dans le cœur où la stimulation électrique qui initialise la contraction du cœur est amorcée. La stimulation du nœud sinusal n'est pas commandée par le cerveau mais elle est générée à même le nœud sinusal. Contrairement au cœur, le fonctionnement du diaphragme est asynchrone. La contraction du diaphragme est commandée à plusieurs endroits par le nerf phrénique.

Étant donné qu'il y a plusieurs points de stimulation électrique sur le diaphragme, il y a des annulations de potentiel qui ont pour effet de réduire l'amplitude résultante du signal EMG_{di}. Le signal résultant de l'activité du cœur est si fort qu'on peut facilement le capter avec des électrodes n'importe où sur le corps. Ce signal vient s'additionner à l'EMG_{di}. Le filtrage de l'ECG avec un filtre passe haut avec une fréquence de coupure de 10Hz parvient à réduire suffisamment l'ECG. Toutefois, ce filtrage passif atténue également une partie de l'EMG_{di} puisque le spectre fréquentiel de ce signal contient des composantes fréquentielles inférieures à 10Hz.

La seconde lecture qu'on effectue est la mesure de la pression trans-diaphragmatique. La méthode la plus répandue pour capter ce signal est d'utiliser une sonde sur laquelle sont logés deux ballons en latex. Le coût très faible ainsi que la facilité d'utilisation sont des facteurs qui expliquent leur popularité, mais leur faible réponse en fréquence est une limitation majeure. Toutefois, il existe d'autres méthodes pour capter la pression trans-diaphragmatique par des cathéters ayant des petits capteurs de pression piézo-résistifs. Il y a aussi les capteurs de pression à fibre optique qui offrent une précision remarquable et une faible sensibilité au bruit. Cependant, l'instrumentation nécessaire pour les faire fonctionner est encombrante et le coût de son acquisition est très élevé.

Les objectifs visés

Un premier objectif consiste à faire une recherche détaillée sur tous les capteurs de pression médicaux. Nous savons qu'il existe plusieurs types de capteurs de pression :

fibre optique, piézo-résistif, couplage au liquide, etc. Une description de chaque type de capteur sera effectuée dans le but de les comparer et de les évaluer pour nos besoins. Nous tenterons par la suite de fabriquer un cathéter sur lequel seront montés les capteurs de pression choisis afin de vérifier leur efficacité.

En ce qui concerne la méthode de lecture de l'EMG_{di}, une recherche détaillée sur les différents types d'électrodes servant à cette tâche sera effectuée. Une partie significative de ce mémoire décrira une méthode qui permettra d'analyser différents types d'électrodes oesophagiennes.

Il est important de noter qu'il existe des méthodes de filtrage sophistiquées qui réussissent à supprimer une source de contamination d'un signal. Un bref résumé de ces méthodes de filtrage sera présenté à titre d'information. Toutefois, la partie sur les algorithmes de filtrage ne fera que complémenter la section concernant la réalisation d'une nouvelle électrode. Un des buts principaux de cette recherche consiste à étudier les électrodes œsophagiennes en ce qui a trait à leurs caractéristiques physiques et à leur sensibilité à l'EMG_{di}. Il est à noter que cette recherche n'a pas pour but de faire l'étude d'algorithmes de filtrage de signaux mais elle consiste plutôt à étudier et à concevoir une électrode œsophagienne qui aura la capacité de capter l'EMG_{di} sans contamination cardiaque. Un type d'électrode œsophagienne particulière a été proposée en vue d'éliminer l'artefact cardiaque. Une attention particulière sera portée à l'égard de cette électrode.

Abstract

The analysis of the respiratory system is done by analysing the electromyogram of the diaphragm (EMG_{di}) and the trans-diaphragmatic pressure (P_{di}). The diaphragm is the main muscle involved in respiration. The role of the diaphragm is to increase the volume of the thorax in order to lower the abdominal pressure and fill the lungs with air. The diaphragm is a voluntary muscle that is stimulated by the brain. The phrenic nerve is responsible to carry the electric signal from the brain to the diaphragm. This nerve has several ramifications and all of these ramifications make contact with the diaphragm all over its surface. When the diaphragm is stimulated by the phrenic nerve, the nerve contacts on the diaphragm are the origin of action potentials that propagate along the muscular fibers of the diaphragm. These multiple stimulations over the diaphragm's surface yield a noise-like signal at the periphery of the esophageal sphincter. Action potential propagation can be seen as a domino effect on the nerve and muscle cells. This happens when a cell depolarizes when its neighboring cell are still depolarized. Unlike a domino, the cell recharges its potential slowly. Since the diaphragm and the esophageal sphincter are in contact, a slight potential can be observed by an esophageal electrode when breathing occurs. Furthermore, a differential pressure between the esophagus and the stomach can be observed during respiration. These measurements can be obtained with an esophageal catheter with two pressure sensors and an esophageal electrode.

However, the readings of the EMG_{di} signal is contaminated by the electrocardiogram (ECG), which is also known as the cardiac artefact. Unlike the diaphragm, the muscular fibers of the heart are synchronous. All of the cardiac fibers are stimulated by a single source: the sinusal node. The synchronous behavior of the cardiac fibers gives its signal a higher electrical potential than the fibers of the diaphragm whose fibers are depolarized asynchronously.

The other reading that has to be done is the trans-abdominal pressure P_{di}. A common way to measure this pressure is to use two latex balloons mounted on a catheter. This method is very popular because of its ease of use and calibration. However, the frequency response of this method is not so great. The pressure signal from the latex balloon to the pressure sensor is modulated by an air pressure wave through a small tube within the catheter. This air coupling has a very low frequency response and could be easily replaced by a much faster media such as an electrical or optical link.

The goals of this research

A first goal is to perform a detailed search and analysis on the different medical pressure sensors that could be used in this application. We know that the fiber optic and the microelectronic pressure sensors are good candidates to replace the latex balloons. An analysis will be performed on the pressure sensors in order to determine which of them are suitable for our application. When the pressure sensor will be selected, a catheter with two pressure sensors will be fabricated.

The analysis of the electrode fondamentals will be used to understand why a portion of the ECG is measured by the esophageal electrode. A model of the thorax and an electrode will be developped in a software simulator in order to study the effects of several electrode parameters on the EMG_{di} readings. We hope that this model will lead us to the fabrication of an electrode that will be sensible only to the EMG_{di}.

Some filtering techniques are capable of eliminating contaminating signals from a desired signal. However, these methods are applied to the signal when the measurements are done. Some of these methods cannot run in real time. A brief overview of these methods shall be presented in this research. It is important to know that the use or development of signal filtering techniques is not a goal here. There is a lot of research concerning the development of filtering techniques but nothing has been done to optimize the electrodes for reducing the contamination at the source. A particular shape of esophageal electrode has been proposed to eliminate the ECG during EMG_{di} recordings. Our focus will be oriented in studying this specific electrode and optimize its shape for EMG_{di} readings and a better ECG filtering.

Table des matières

Liste des tableaux

Liste des figures

Liste des annexes

Chapitre 1

Domaine d'application de la recherche

1.1 Introduction

L'enregistrement de l'EMG_{di} et de la pression trans-diaphragmatique à l'aide d'une électrode œsophagienne est susceptible de fournir des renseignements tout à fait uniques concernant l'activité des centres respiratoires bulbaires et du muscle primaire de la respiration, le diaphragme[13]. Les signaux captés sont sauvegardés afin de pouvoir être analysés ultérieurement. Plusieurs analyses du système respiratoire peuvent être effectuées avec la forme de l'EMG_{di} ainsi qu'avec sa valeur RMS. Ces signaux permettent de diagnostiquer des paralysies ou des fatigues au niveau du diaphragme et bien d'autres problèmes reliés au système respiratoire.

1.1.1 Physiologie du système respiratoire

La respiration est une fonction qui est commandée par le cerveau. Le cerveau peut ordonner une respiration en envoyant une impulsion par le nerf phrénique. Le nerf phrénique possède plusieurs ramifications et fait contact à plusieurs endroits sur la surface du diaphragme. Ces multiples points de contact sont à l'origine de plusieurs potentiels d'action qui font apparition sur le diaphragme et qui ordonnent sa contraction. Lors d'une contraction du diaphragme, les nombreux potentiels d'action qui circulent dans les fibres musculaires ne sont pas en phase. Le résultat de ces multiples potentiels

d'action hors phase aux bornes de l'électrode œsophagienne est un signal de faible amplitude qui ressemble beaucoup à un bruit.

D'autre part, lors d'une respiration, une différentielle de pression est exercée entre l'estomac et l'œsophage. Le sphincter œsophagien isole hermétiquement l'estomac et l'œsophage afin d'éviter tout refoulement gastrique lors de la digestion. L'activité électrique du diaphragme se propage jusque sur le sphincter œsophagien puisque ce dernier si situe tout près du diaphragme. Le diaphragme est un muscle plat ayant la forme d'un paraboloïde renversé. Le diaphragme est troué à l'endroit où passe l'œsophage. Ainsi, la pression trans-diaphragmatique et l'EMG_{di} peuvent être captés dans la région du sphincter œsophagien, dans l'œsophage.

Une autre source de potentiels d'action dans le corps humain est le cœur. Contrairement au diaphragme, le cœur est stimulé à partir du nœud sinusal, qui est situé dans le cœur. C'est le nœud sinusal qui amorce la contraction du cœur. Les potentiels d'action qui sont générés longent simultanément les fibres musculaires du cœur. Le cœur est situé au-dessus du diaphragme et en est séparé par le péricarde. Le péricarde isole le cœur et le diaphragme. Le corps humain est un milieu purement résistif. Lors des battements cardiaques, l'ECG peut être lu par des électrodes situées sur tout le corps.

On peut voir sur la Figure 1.1 la physiologie de l'estomac. Le diaphragme est le mince muscle transversal qui est situé au dessus de l'estomac. Le sphincter œsophagien entoure l'œsophage juste au dessus de l'estomac.

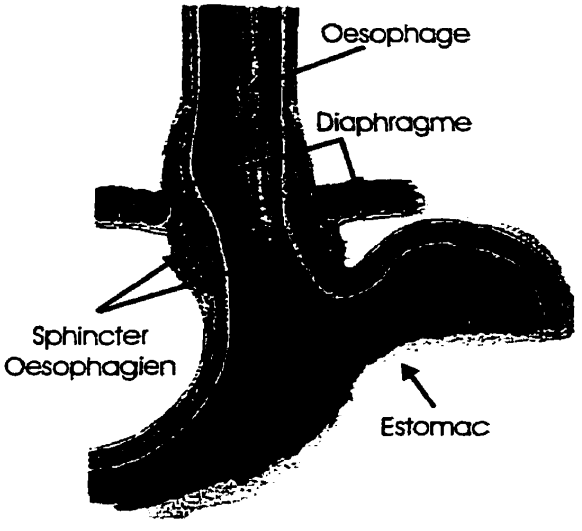

Figure 1.1: Estomac, diaphragme et sphincter œsophagien [32]

1.2 Instrumentation utilisée en clinique

Il existe plusieurs types de capteurs de pression qu'on peut utiliser afin de mesurer la pression trans-diaphragmatique. Ces capteurs, détaillés à la prochaine section, sont basés, entre autre, sur l'utilisation de ballons en latex couplés à des transducteurs de pression externes. Un couplage à l'air relie les ballons aux transducteurs de pression. Une autre méthode semblable consiste à effectuer un couplage au liquide sans ballon en latex. Des capteurs de pression plus sophistiqués basés sur des circuits microélectroniques intégrés sur silicium sont petits, peu coûteux et ils offrent des caractéristiques intéressantes par

rapport aux capteurs conventionnels. Enfin, les capteurs de pression à fibre optique sont les derniers-nés de la famille des capteurs de pression. La mesure de l'EMG_{di} est effectuée avec une électrode œsophagienne bipolaire. Il existe une variante de cette électrode qui possède dix contacts.

1.2.1 Les ballons en latex couplés à l'air

Cette méthode de lecture de pression trans-diaphragmatique est la plus commune et consiste à placer deux ballons en latex sur un cathéter et de les insérer de part et d'autre du sphincter œsophagien via les voies nasales. Les ballons en latex sont reliés à des transducteurs piézo-résistifs [12] externes par des petits tubes longeant le cathéter. Le principal désavantage dont souffre cette méthode est sa très faible réponse en fréquence due au couplage à l'air.

1.2.2 Le couplage à l'eau

Cette méthode a été tentée afin de pallier le problème de réponse en fréquence du couplage à l'air des ballons en latex. Le cathéter servant à faire la mesure a l'avantage d'être réutilisable. Dans ce cathéter, deux ouvertures qui le longent sont remplies d'eau. Les deux tubes qui longent le cathéter sont terminés par deux orifices qui permettent de communiquer la pression exercée de part et d'autre du sphincter œsophagien aux transducteurs de pression. On procède au calibrage de la même façon qu'avec les ballons en latex. On obtient une réponse en fréquence beaucoup plus élevée avec un couplage au liquide par rapport à un couplage à l'air. Cependant, un problème majeur de cette méthode est que les deux transducteurs doivent êtres exactement à la même hauteur que

les orifices, sinon, une erreur considérable s'ajoute aux mesures. Ce système est très sensible aux mouvements du patient puisqu'une légère variation de la hauteur des orifices du cathéter entraîneraient des erreurs de lecture significatives.

1.2.3 Les capteurs de pression à fibre optique

Ces capteurs de pression proposés récemment ont plusieurs avantages. Ils sont parfaitement insensibles aux champs électromagnétiques, offrent une précision inégalée, ont l'avantage de rester calibrés pendant plus de 24 heures et ils ont une réponse en fréquence de plus de 15KHz [9]. De plus, plusieurs capteurs peuvent être logés sur un seul cathéter et leur fabrication est relativement simple et peu coûteuse. Leur principe de fonctionnement peut être résumé de la façon suivante : une membrane sensible à la pression module la lumière incidente, une fraction de cette lumière incidente sera réfléchie dans la fibre optique de retour. La quantité de lumière réfléchie est reliée à la pression de façon non linéaire. Un système d'amplification et de calcul doit être en place afin de calculer la pression exercée en fonction de l'intensité du faisceau réfléchi. L'instrumentation servant à émettre le faisceau de lumière incident et le système d'amplification sont complexes et dispendieux.

1.2.4 Les transducteurs piézo-résistifs

Ces capteurs de pression relativement précis sont construits à partir de procédés de micro-machinage. Le principe derrière ces capteurs miniatures est le pont de résistances *wheatstone*. Ces résistances imprimées sur une membrane flexible sont construites à partir de semi-conducteur sensible à la tension (*strain gauge*). La valeur de la résistance

varie en fonction de l'étirement de la membrane. L'avantage qu'offre le pont de résistance est une sortie linéaire et une stabilité quant aux variations de pression et de température. Lorsque la membrane s'étire sous une augmentation de pression, deux résistances opposées augmentent alors que les deux autres résistances opposées diminuent. La réponse en fréquence est beaucoup plus grande que les transducteurs couplés à l'air (ballons en latex). La taille de ces capteurs est très faible également, ce qui nous permet d'en loger plusieurs sur un même cathéter. Le principal désavantage de ces capteurs de pression est leur fragilité : on doit faire des contacts miniatures sur les plots du capteur et ces contacts sont excessivement fragiles.

Les sections précédentes ont décrit les différents types de capteurs de pression pouvant être utilisés pour mesurer la pression trans-diaphragmatique. Les prochaines sections s'attarderont à décrire les différents types d'électrodes qui peuvent être utilisées pour capter le signal EMG_{di}.

1.2.5 Électrode œsophagienne

L'électrode œsophagienne est composée de deux anneaux métalliques séparés d'environ 10mm qui sont logés sur un cathéter. Un fil électrique est relié à chacun des anneaux et branché dans un amplificateur de lecture à l'extérieur. Les anneaux sont faits d'acier inoxydable et le cathéter est réutilisable. Cette électrode peut être logée près des parois du sphincter œsophagien lors de lectures de l'EMG_{di}. Aussi, il existe une électrode œsophagienne à dix électrodes indépendantes. Elle permet d'échantillonner les potentiels

sur une plus grande distance le long du sphincter œsophagien. Ainsi, on peut choisir le canal qui semble le plus près du sphincter œsophagien en choisissant la paire d'électrode qui capte mieux l'EMG_{di}.

1.3 Sources d'erreur lors de la lecture des signaux EMG_{di} et P_{di}

Lors de la respiration normale, plusieurs phénomènes sont observés. Tout au long de l'inspiration, on voit apparaître l'EMG_{di} aux bornes de certaines des électrodes intégrées dans le cathéter. On constate également une augmentation graduelle de la pression trans-diaphragmatique. La clarté des signaux et l'absence de toute source de contamination sont très importants. Or chacun des mouvements corporels induisent des erreurs dans les signaux car chaque muscle du corps en contraction génère des courants. Il est très difficile de capter des signaux de façon très claire et précise avec une bonne précision dans un milieu où le bruit est important.

1.3.1 Problèmes reliés à l'analyse de la pression trans-diaphragmatique

La précision limitée des transducteurs est souvent une cause d'erreur lors des mesures. Les capteurs utilisés sont sensibles à la température. La température à l'intérieur du corps est assez stable mais la température du capteur qu'on vient d'insérer ne se stabilise pas très rapidement. Il faut parfois attendre quelques minutes avant de calibrer le capteur de pression.

Il existe également un phénomène d'hystérésis qui peut nuire à la précision des résultats. Le phénomène d'hystérésis attribué aux capteurs de pression est principalement causé par

une non-linéarité mécanique de la membrane. Certains capteurs de pression peuvent perdre beaucoup de précision lorsque la fréquence des signaux atteint la limite du capteur de pression.

Il devient important de s'assurer que les sources de contamination décrites ci-haut ne soient pas présentes lors de la mesure des signaux. Idéalement, il serait préférable de mesurer la pression avec des capteurs de pression qui seraient passifs à ces perturbations.

1.3.2 Problème relié à la mesure de l'EMG_{di}

Le principal obstacle lors de la lecture de l'EMG_{di} est la présence de l'ECG lors d'un battement cardiaque. Le signal contaminant induit une erreur dans la valeur RMS de l'EMG_{di}, nuisant ainsi à l'analyse du système respiratoire. Des observations en clinique ont permis d'observer qu'il est possible d'éliminer presque entièrement le signal ECG en positionnant judicieusement l'électrode œsophagienne. Cependant, il est difficile de reproduire les conditions d'une telle mesure à cause du mouvement de haut en bas du diaphragme pendant la respiration. Il existe des méthodes de filtrage qui réussissent néanmoins à réduire l'ECG présent dans l'EMG_{di}. Toutefois, puisqu'une grande partie des spectres fréquentiels des signaux EMG_{di} et ECG se recoupent, les algorithmes de filtrage suppriment une partie du signal utile.

Des études ont démontré que le ratio de puissance EMG_{di} / ECG de l'ordre de 5dB introduit des erreurs significatives dans le signal EMG_{di}. Selon le même article, un ratio

EMG_{di} / ECG de 13 dB est suffisamment élevé pour que l'erreur introduite par l'ECG soit négligeable [20].

Une autre source de signaux électriques dans le corps est le mouvement. À chaque fois qu'un muscle exerce un travail, on peut mesurer son activité électrique dans le corps. Il est donc important que le patient soit le moins actif possible au moment des lectures.

On peut voir à la Figure 1.2 à quoi ressemble un signal EMG_{di} contaminé par un artefact cardiaque, l'ECG. Le signal présenté sur cette figure a été mesuré en clinique avec une électrode oesophagienne standard. On peut observer l'EMG_{di} qui s'étale de 500ms à 750ms. Tous les autres pics sont des artefacts cardiaques.

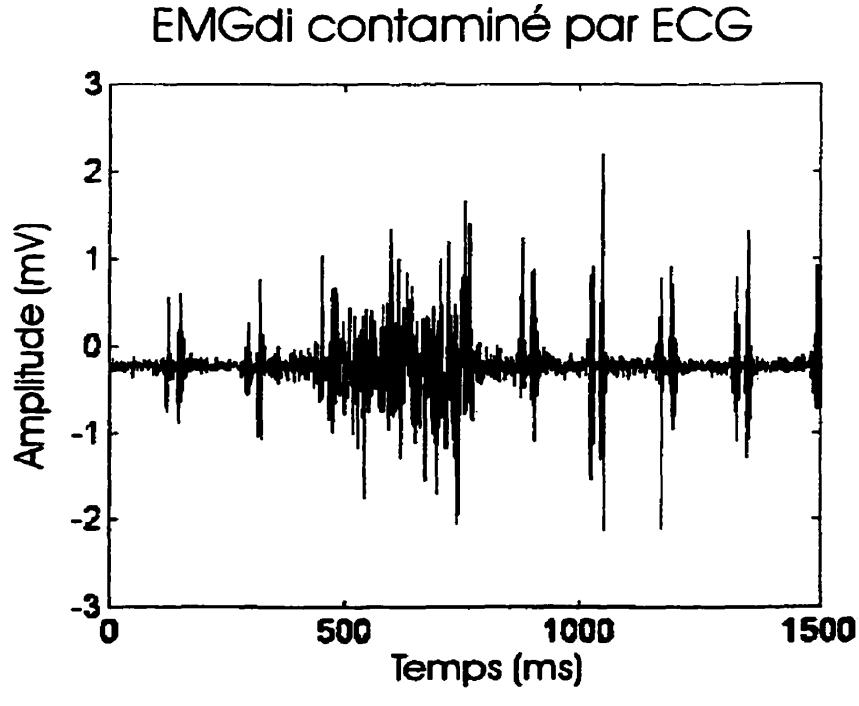

Figure 1.2: EMG_{di} contaminé par l'ECG

1.4 Solutions envisagées

Les ballons en latex couplés à l'air comportent plusieurs inconvénients qui réduisent la qualité des enregistrements. Au cours de cette recherche, nous envisagerons la possibilité d'effectuer la lecture de la pression trans-diaphragmatique à l'aide de capteurs plus perfectionnés. Une étude approfondie des capteurs de pression fibre optique et piézo-résistifs sera effectuée afin d'en faire ressortir lequel sera le mieux adapté à nos besoins.

Pour la mesure de l'EMG_{di}, une étude sur la propagation des potentiels d'action cardiaque et diaphragmatique sera effectuée. Bien qu'il existe des méthodes de filtrage des signaux, aucune méthode à notre connaissance n'a été proposée pour filtrer les signaux à la source. Cette recherche sera principalement axée sur la structure physique des électrodes pour en analyser leur sensibilité aux signaux ECG et EMG_{di}. Il sera important d'étudier les effets des variations des paramètres de l'électrode œsophagienne quant à la sélectivité de l'EMG_{di} et la capacité à filtrer l'ECG. D'autres types d'électrodes seront considérés telles l'électrode bipolaire verticale et l'électrode quadripolaire dans le chapitre 2.

Un différent type d'électrode oesophagienne a été proposée dans le but d'atténuer l'ECG sans avoir recours à un système de traitement de signal. La propriété intellectuelle quant à la structure et la forme de l'électrode appartient exclusivement au chercheur qui l'a proposée. C'est pour cette raison qu'aucune information sur cette électrode (diagrammes, description, etc) ne sera divulguée dans ce mémoire. Certaines expériences ont été effectuées avec cette électrode et les résultats de ces expériences sont présentés dans ce

mémoire. L'absence de toute information plus spécifiques ne nuit aucunement à la compréhension des méthodes qui ont été utilisées dans cette recherche pour simuler le thorax.

Le terme « prototype » qui est utilisé dans ce mémoire fera ainsi référence à cette électrode, par opposition au terme électrode « standard » qui désigne l'électrode bipolaire commerciale.

Une partie importante de ce mémoire sera l'étude d'une nouvelle méthode d'analyse d'électrodes œsophagiennes. Un cadre théorique servant à analyser les différents paramètres d'électrodes sera développé. Un prototype d'électrode sera réalisé afin de pouvoir valider la méthode d'analyse. De plus, un modèle tridimensionnel du thorax sera réalisé avec un simulateur de différences finies (FDTD). Ce simulateur permettra de mieux contrôler les paramètres expérimentaux et d'effectuer des comparaisons des différents types d'électrodes mentionnés.

Une recherche antérieure a été effectuée dans le but de convertir un simulateur FDTD du langage FORTRAN au C++. Les fonctions principales du moteur de calcul ainsi que les conditions de frontières absorbantes y ont déjà été programmées. Cependant, quelques erreurs de programmation doivent êtres corrigées afin de rendre ce code fonctionnel. De plus, il faudra rajouter des fonctions d'entrées des paramètres, des fonctions de génération du thorax et des organes internes ainsi que des fonctions d'exportation des résultats sous forme graphique.

La validation du logiciel ainsi que du modèle sera discutée plus en profondeur. Le simulateur sera principalement validé en comparant des mesures cliniques aux résultats expérimentaux simulés. La simulation s'effectuera avec un modèle qui ressemblera le plus fidèlement aux conditions expérimentales. Les ratios entre les signaux ainsi que leurs amplitudes serviront à juger si le simulateur donne des résultats semblables à ceux observés en clinique.

Quant au logiciel et au moteur de calcul FDTD, une simulation sera effectuée sur un modèle dont on connaît le résultat théorique. Des mesures seront prises à quelques endroits du modèle et seront comparées aux valeurs théoriques.

Chapitre 2

2. Méthodes et instruments utilisés pour mesurer l'EMG_{di} et P_{di}.

2.1 Mesure de la pression trans-diaphragmatique

Tel que décrit précédemment, il y a une différence de pression qui est exercée de part et d'autre du sphincter oesophagien lors de la respiration. Cette différence de pression doit être mesurée avec une imprécision minimale. Voici les principales spécifications auxquelles devront répondre les capteurs de pression :

- Lecture de la pression devra être linéaire dans la plage allant de 0 cm H_2O à 350 cm H_2O (0 à 260 mm Hg) ;
- Température d'opération : 37°C ± 3°C.
- Précision : 0.1 cm H_2O (≈0.07 mm Hg).
- Dimensions : doit entrer dans un cathéter dont le diamètre maximal est 5mm.

De plus, le capteur de pression devra résister à l'acide de l'estomac et doit être stérilisable.

2.1.1 Mesure de la pression avec des ballons en latex

La taille standard des ballons en latex est de 10cm de longueur et de 1.7cm de diamètre[13]. Les deux ballons doivent être hermétiquement reliés à des transducteurs externes par des tubes qui longent le cathéter. Une valve doit être insérée le long des tubes de couplage afin de pouvoir calibrer la pression initiale dans les tubes en latex. On

s'attend à ce que ces ballons permettent la mesure de pression à une fréquence égale ou inférieure à 2Hz.

Ces types de cathéters sont très populaires à cause de leur facilité d'usage et faible coût d'achat. Ils sont très simples à calibrer et donnent en général d'assez bons résultats. Cependant, ils ne sont pas appropriés pour effectuer des lectures qui s'étendent sur plusieurs heures. Des légères fuites font en sorte qu'ils doivent constamment être recalibrés. Lorsque la pression à mesurer a une fréquence supérieure à 2Hz, un effet d'hystérésis apparaît et induit des erreurs de lecture. Cet effet d'hystérésis est attribué à l'élasticité de la paroi du cathéter et des ballons en latex.

Afin d'augmenter la réponse en fréquence le couplage à l'air a été remplacé par un couplage à l'eau. Les ballons en latex ont été enlevés pour laisser place à deux ouvertures dans un nouveau cathéter. Chacune des ouvertures est reliée par des tubes indépendants à un transducteur de pression externe. Ces tubes sont remplis d'eau. Ainsi, la pression dans le milieu exerce une force sur la colone d'eau qui transmet cette pression à la membrane du transducteur de pression. La réponse en fréquence de ce nouveau cathéter est certainement beaucoup plus grande que celui basé sur des ballons en latex. Cependant, on doit recalibrer régulièrement le système à cause des fuites d'eau. De plus, cette méthode est instable. Il est difficile de garder une bonne référence de mesure puisque la colonne d'eau a une masse beaucoup plus grande que l'air. Des erreurs considérables sont

détectées lorsque le sujet bouge. Cette méthode donnerait de bons résultats si on pouvait contrôler parfaitement la hauteur des ouvertures où est exercée la pression.

2.1.2 Fonctionnement des transducteurs de pression

La majorité des transducteurs de pression fonctionnent sous le principe du *Wheatstone Bridge*. On présente le circuit équivalent du *Wheatstone Bridge* à la figure 2.1:

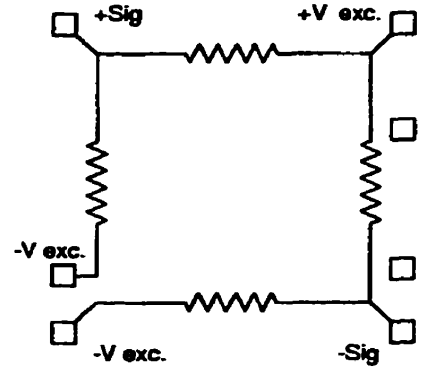

Figure 2.1: Pont Wheatstone

Il existe des semiconducteurs dont la résistivité change en fonction de l'étirement (*strain gauge*). On utilise ces semiconducteurs pour fabriquer le *Wheatstone Bridge*. Les résistances sont placées sur une membrane qui se déforme lorsqu'une pression est exercée. Les conductances de deux résistances augmenteront alors qu'elles diminueront pour les deux autres résistances. Donc, le pont sera déstabilisé d'un côté ou de l'autre en fonction de l'étirement de la membrane. La structure en pont est avantageuse puisque la sortie est. On peut facilement fabriquer des résistances identiques par micromachinage. De plus, le pont de Wheatstone demeure très linéaire à différentes températures même si

le comportement de chaque résistance varie avec la température. Le pont de Wheatstone doit être alimenté aux bornes $+V_{exc}$ et $-V_{exc}$. La lecture de la pression s'effectue aux bornes +Sig et –Sig.

Il existe deux types de capteurs de pression : absolue et relative. Un capteur de pression absolue donne la mesure de la pression par rapport au vide. Un capteur de pression relative donne la différence de pression entre deux milieux. Pour un capteur de pression absolue, un côté de la membrane doit être dans le vide. Ceci nécessite donc la fabrication d'une chambre à vide bien isolée lors de la fabrication du capteur de pression. D'autre part, lors de la mesure relative de pression, on mesure en général la pression interne en fonction de la pression atmosphérique. Ceci nécessite donc le passage de deux tubes par capteur de pression : le premier relie un côté de la membrane au transducteur de pression alors que le second tube relie l'autre côté de la membrane à l'extérieur.

La sensibilité varie d'un capteur à l'autre. La compagnie Silicon Microstructures fabrique un type de capteur (SM-5102-005A) dont la sensibilité est de 50µV/mmHg pour une stimulation aux bornes +Sig et -Sig de +5Volts et de –5Volts respectivement. La région d'opération linéaire de ce capteur est située entre 1mmHg et 100mmHg.

2.1.3 Les technologies de fabrication et le micro-machinage

Dans l'industrie microélectronique, le coût de fabrication de circuits dédiés est très élevé et les délais de fabrications sont très longs. Toutefois, la Société canadienne de

microélectronique (SCM) offre des services de fabrication de circuits dédiés aux institutions universitaires canadiennes à des coûts beaucoup moins élevés. De plus, les délais de fabrication sont moins longs qu'avec les autres compagnies.

Le micro-machinage permet la réalisation de très petits capteurs de pression en utilisant des procédés de photolithographie. Ce procédé permet de faire des capteurs dont les dimensions sont de l'ordre de quelques millimètres carrés. Toutefois, le coût et le temps associés à la conception d'un capteur de pression sont énormes. Le design et la fabrication de capteurs de pression peut faire l'objet de quelques mois de recherche. Plusieurs compagnies et laboratoires de recherche en Amérique du Nord commercialisent de tels capteurs de pression à faible coût. Entre autres, les compagnies Silicon Microstructures, Lucas Nova Sensors, et Task Microelectronics de Montréal qui s'intéresse récemment à la fabrication de tels capteurs. Finalement, des chercheurs de Simon Fraser University ont également proposé un capteur de pression dédié à leurs applications. On retrouve sur le marché une gamme importante de capteurs de pression qui peuvent convenir à presque toutes les applications. La fabrication par micro-machinage est un long processus. Il est beaucoup plus avantageux d'acheter ces capteurs de pression plutôt que de tenter d'en fabriquer soi-même.

2.1.4 Les capteurs de pression fibre optique

Il existe différents types de capteurs de pression à base de fibre optique. Un de ces types consiste à mesurer la différence de phase d'un faisceau lumineux qui a été réfléchi par

une substance sensible à la pression. Ce type de capteur donne une très grande précision et un très grand rapport signal sur bruit puisque le bruit est pratiquement inexistant. Ce type de capteur de pression est assez complexe à réaliser et son système de modulation-démodulation de la lumière est complexe.

Le second type de capteur de pression a fibre optique consiste à mesurer le pourcentage de la lumière réfléchie par rapport au faisceau incident. La membrane réfléchissante fait varier la quantité de lumière qui est réfléchie lorsque cette dernière se déplace sous l'effet de la pression. Un schéma de principe de ce capteur de pression est illustré à la figure 2.2.

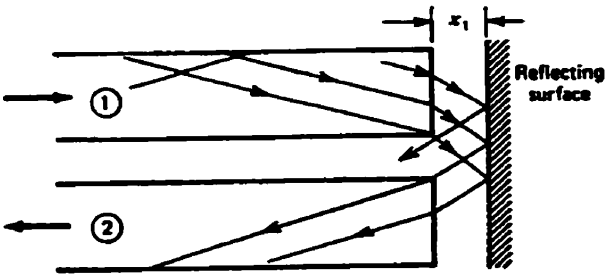

Figure 2.2: Schéma d'un capteur de pression fibre optique

La lumière incidente pénètre la fibre 1 et est dirigée jusqu'à l'ouverture qui sépare les bouts des fibres et la surface réfléchissante. Une quantité de lumière est réfléchie dans la fibre 2. La proportion de lumière qui est réfléchie dans la fibre 2 est fonction de la distance x_1.

Ce type de capteur de pression souffre principalement des pertes de lumière dues au rayon de courbure aigu. Ces pertes de lumières entraînent inévitablement des légères erreurs lors de la lecture de la pression.

Les capteurs de pression en fibre optique sont plus petits que tous les autres types de capteurs existants (micro-électroniques, ballons en latex). Ces cathéters sont faciles à implanter à cause de leur très faible diamètre. Ils sont également très sécuritaires puisqu'il n'y a aucun liquide pouvant contaminer le patient dans le cathéter. Ils ont l'avantage de ne pas se décalibrer, même lorsqu'on les utilise pendant plusieurs heures. Comme les capteurs micro-électroniques, le transducteur de pression en fibre optique peut être placé directement à l'endroit où l'on veut lire la pression. Comme le cathéter n'est couplé ni par l'air ni par un liquide, il est moins susceptible de fuites et de décalibration. Un autre grand avantage de ces capteurs de pression est qu'ils sont électriquement, chimiquement et magnétiquement passifs.

Afin de communiquer avec le monde extérieur, le capteur de pression transfère ses signaux à un circuit spécialisé. Ce circuit envoie une excitation (faisceau de lumière) et compare la fraction de lumière qui est réfléchie par rapport au faisceau émis. Sachant la fraction de lumière retournée, on peut évaluer la pression au bout du cathéter en se basant sur des courbes de ratio de réflexion en fonction de la distance de la membrane par rapport à l'ouverture de la fibre optique. Toutefois, le circuit en question peut être assez dispendieux par rapport au coût du capteur de pression

La précision d'un capteur de pression en fibre optique est de près de 0.03mmHg. Cette précision est plus que suffisante pour nos besoins. Les dimensions des micro-capteurs de pression sont environ 1mm de diamètre par 2.5mm de longueur. Les meilleurs capteurs de pression peuvent perdre au plus 0.1mmHg de précision par 24 heures. Toutefois, comme nos essais ne dureront que quelques minutes, ces légères variations n'auront pas d'effets significatifs. Il n'est pas dispendieux de fabriquer un capteur de pression fibre optique en laboratoire. C'est plutôt le système de modulation-démodulation qui exige de coûts élevés.

2.1.5 Capteurs de pression microélectroniques

Exar est une compagnie californienne qui produit des micro-capteurs de pression par micro-machinage[22]. Cette compagnie propose plusieurs modèles de micro-capteurs pour différents types d'applications. Le modèle SM-5102-005A, dont les spécifications sont présentées en annexe, peut être utilisé pour des applications biomédicales. Ses dimensions sont très adéquates pour notre application. Le coût, de l'ordre de quelques dollars par capteur de pression est très avantageux. Des fils doivent être soudés afin de relier les plots de contacts du capteur de pression à une plaquette. L'École Polytechnique possède les ressources nécessaires afin de faire de telles micro-soudures. Cinq soudures doivent être faites sur chaque capteur.

Une caractéristique principale différencie ce capteur de pression aux ballons en latex : le transducteur de pression est logé directement où la pression doit être mesurée. Ainsi, on

évite un couplage au liquide ou à l'air, réduisant ainsi toute source d'erreur pouvant provenir de fuites quelconques. La pression est mesurée par rapport au vide. Ceci est avantageux puisque cela nous évite d'avoir à passer le tube de référence, réduisant ainsi le diamètre du cathéter. Chaque capteur de pression a une dimension de 2mm*2mm*0.9mm et possède une petite chambre à vide, qui sert de référence. Aucune calibration n'est nécessaire puisque chaque capteur est bien isolé lors de sa fabrication. La réponse en fréquence de ces capteurs est beaucoup plus grande que ce qui est nécessaire pour notre application. On doit relier quatre fils à chaque capteur de pression : deux fils d'alimentation et deux fils pour la lecture du signal. Donc, on doit passer 8 fils en tout puisqu'il y a deux capteurs de pression. On pourrait réduire ce nombre à 6 si on relie les alimentations des deux capteurs ensemble mais ceci rend le montage du cathéter plus difficile et le cathéter risque de briser plus facilement.

Des tests ont été effectués en laboratoire avec ces capteurs de pression. Les tests avaient pour but de vérifier si l'ajout d'époxy flexible[1] , qui recouvre la membrane, aura un effet sur la précision des lectures. Les détails de ces tests sont présentés au chapitre 5. L'expérience nous a permis de conclure que l'ajout d'une mince couche protectrice n'a aucun effet sur la précision du capteur de pression.

[1] L'époxy flexible sert à solidifier les micro-soudures (*wire bond*)

2.2 Les électrodes servant à mesurer l'EMG$_{di}$

2.2.1 Analyse du fonctionnement d'une électrode

Le principe associé à la mesure du potentiel par une électrode est assez simple. Normalement, une électrode qui sert à mesurer une tension doit être passive, c'est-à-dire qu'elle ne doit pas perturber le milieu qu'elle mesure. Chaque borne de l'électrode repose sur une équipotentielle. Les équipotentielles sont représentées par des lignes continues sur lesquelles le potentiel est constant dans un milieu donné. Le différence de potentiel captée par l'électrode est la différence entre les équipotentielles touchées par ses deux bornes (figure 2.3).

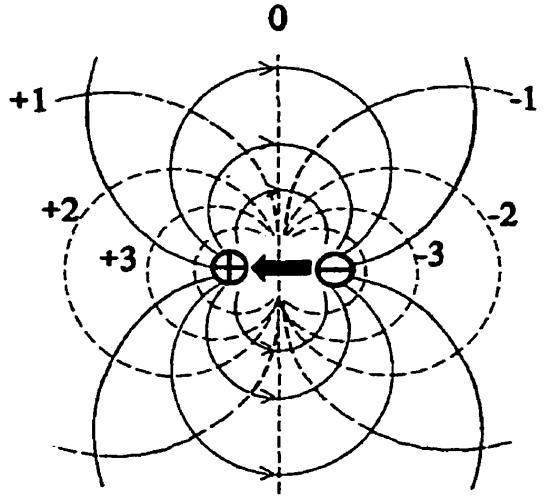

Figure 2.3: Dipôle de courant avec les équipotentielles en pointillés et les lignes de courant en traits pleins.

La méthode du *Lead Field* est couramment utilisée afin d'étudier le patron de sensibilité d'une électrode. Cette méthode exploite le problème inverse de l'électrode de mesure ; l'électrode devient une source de courant. En injectant un courant dans l'électrode, cette dernière génère des champs électriques et des équipotentielles dans le milieu où elle est située. L'analyse de ses courbes résultantes nous indique son patron de sensibilité.

Supposons que le dipôle montré à la figure 2.3 représente les deux bornes de notre électrode. Les lignes de courants (lignes pleines) nous indiquent les endroits où l'électrode est sensible. Lors de la mesure, les dipôles de courant qui seront tangents à ces lignes de courant auront un effet sur l'électrode, mais ceux ayant l'orientation perpendiculaire n'auront aucun effet sur l'électrode.

Mathématiquement, le potentiel total (Volts) lu par l'électrode est la somme sur tout le volume des densités de courants **J** parallèles aux parcours **L**,

$$E = \int \vec{L} \cdot \vec{J}_i dv \text{ (Volts)} \tag{2.1}$$

L'orientation de l'électrode est donc un facteur très important. Si on veut capter un signal provenant d'un dipôle quelconque, on doit absolument positionner les électrodes sur deux équipotentielles différentes. Ceci fera en sorte que le dipôle sera tangent aux lignes de sensibilité de l'électrode et la contribution de ce dipôle sera additionnée au potentiel lu par l'électrode. Par ailleurs, si on veut filtrer la contribution

d'un dipôle, on doit placer les deux électrodes sur une même équipotentielle provenant de cette électrode. Ainsi, le dipôle sera parallèle aux lignes de sensibilité de l'électrode et la contribution de ce dipôle sera nulle.

2.2.2 Électrode bipolaire conventionnelle

L'électrode oesophagienne la plus simple qui est utilisée pour capter l'EMG_{di} est l'électrode bipolaire conventionnelle. La figure 2.4 montre à quoi ressemble une telle électrode.

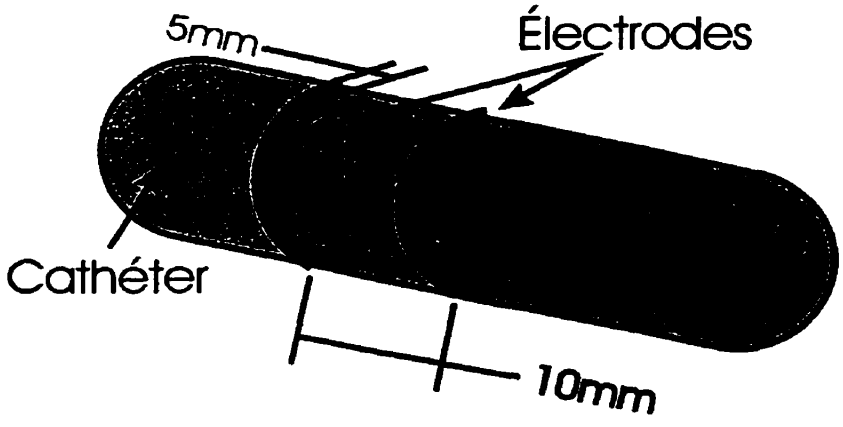

Figure 2.4: Électrode bipolaire conventionnelle

Deux anneaux en acier inoxydable sont montés sur un cathéter. Les deux anneaux sont séparés de 10mm et la largeur de chaque anneau est de 5mm. Le diamètre du cathéter est environ 5mm. Les amplitudes des signaux captés par une telle électrode est de l'ordre de quelques micro-volts (μV). Il est donc important d'avoir un métal qui soit le plus chimiquement stable possible puisqu'on mesure des signaux ayant de très faibles niveaux de tension. Ceci permet d'éviter de contaminer le signal à cause de réactions d'oxydation qui peuvent apparaître entre le métal et le milieu acide de l'estomac et qui

peuvent produire des courants électriques. Chaque anneau en acier inoxidable est soudé à un fil de même matériau qui longe le cathéter. Un fil en cuivre aurait certainement une plus faible impédance qu'un fil en acier inoxidable. Cependant, la soudure acier-cuivre est plus difficile à effectuer.

2.2.3 Autre types d'électrodes

Il existe d'autres types d'électrodes oesophagiennes utilisées pour capter l'EMG_{di}. Un type particulier d'électrode contient dix anneaux en acier. Cette électrode nécessite beaucoup d'espace dans le cathéter pour permettre le passage de dix fils avec gaine. L'utilisation d'une telle électrode peut donner de bons résultats puisqu'elle permet d'obtenir un meilleur échantillonnage des potentiels auprès du sphincter oesophagien. Un outil logiciel sélectionne le ou les électrodes qui donnent le meilleur signal d'EMG_{di} recherché et filtre l'ECG.

Un autre type d'électrode est la quadripolaire. Cette électrode consiste à prendre quatre mesures autour d'un cercle sur le cathéter [24]. Par la suite, on peut appliquer un simple calcul à partir des quatre signaux afin de déterminer la direction dont on veut mesurer le potentiel. Ensuite, on peut changer le patron de sensibilité de l'électrode tout simplement en appliquant un algorithme aux quatre signaux captés. Évidemment, le patron de sensibilité ne peut être orienté que dans un seul plan, celui dans lequel sont placées les électrodes. On pourrait augmenter le degré de flexibilité de ce type d'électrode en ajoutant quatre autres électrodes sur un autre plan. On pourrait dans ce cas

orienter le patron de sensibilité de l'électrode dans n'importe quelle direction. Toutefois, il est à noter que cette électrode requiert le passage de plus de huit fils.

Les électrodes décrites précédemment, accompagnées des outils logiciels, sont flexibles mais complexes et nécessitent des coûts de fabrication élevés. Notre application nécessite d'intégrer sur le même cathéter deux capteurs de pression en plus de l'électrode. Dans ce cas, l'électrode du cathéter doit avoir le moins possible de contacts et de fils à passer. Tel que mentionné dans l'introduction, nous souhaitons avoir une électrode qui puisse filtrer le signal à la source, sans l'intermédiaire d'algorithmes de filtrage, ce qui permettra de réaliser de mesures en temps réel. Nous proposons dans la prochaine section un nouveau type d'électrode à complexité réduite et qui est destiné à filtrer l'ECG .

2.2.4 Effet d'un bon contact électrode-sphincter

Lors de mesures cliniques, on a constaté que la position de l'électrode oesophagienne auprès du sphincter a un effet sur la qualité de l'EMG_{di}. En fait, lorsqu'on positionne judicieusement l'électrode sur le sphincter, on remarque l'absence d'ECG dans le signal EMG_{di}. Cette constatation a suscité chez nous l'étude et la fabrication d'une nouvelle électrode qui ferait contact à plusieurs endroits sur le sphincter oesophagien. Nous tentons d'analyser les caractéristiques de différentes électrodes et leurs effets sur la sensibilité à l'EMG_{di} et sa capacité de filtrer l'ECG. Une première étape pour l'analyse d'électrodes consiste à fabriquer plusieurs prototypes d'électrodes et de vérifier expérimentalement leur efficacité.

2.3 Fabrication d'une électrode oesophagienne prototype

Plusieurs méthodes ont été analysées pour fabriquer une électrode résistante et avec des matériaux biocompatibles. Deux problèmes doivent être réglés : le premier consiste à trouver un moyen de souder l'électrode en acier inoxydable à des fils en cuivre ou en acier; le deuxième consiste à fixer l'électrode sur le cathéter.

2.3.1 Choix du cathéter et de ses dimensions

Le diamètre du cathéter sur lequel sera montée l'électrode doit être le plus petit possible. Il doit être au moins 2mm puisque les capteurs de pression ont une largeur de 2mm. Cependant, il doit y avoir suffisamment d'espace à l'intérieur du dit cathéter pour y passer de 8 à 10 fils de calibre AWG26 avec gaine à l'intérieur. Le cathéter Lévine se trouve dans différents diamètres et longueurs. Un cathéter de dimension 12French sera suffisant pour fabriquer le montage d'une électrode d'essai. L'unité **F** (French) est utilisée pour mesurer le diamètre d'un cathéter (1F = 0.3mm).

2.3.2 Colle servant à lier l'électrode au cathéter

Une colle biocompatible doit être utilisée afin de fixer l'électrode au cathéter. Elle doit adhérer à des métaux ainsi qu'au cathéter et doit sécher rapidement pour faciliter la fabrication de l'électrode. Il faut également prévoir assez d'élasticité dans la colle. Ceci est du au fait que le cathéter se plie lorsqu'on l'insère par les voies nasales. Deux compagnies Nusil [25] et TRA-CON [26] ont été localisées et des échantillons de cathéters à base de Silastic ont été obtenus. Ces échantillons ont été essayés lors du

montage de l'électrode. Les principales qualités que devaient avoir les composés étaient l'élasticité, la durée de séchage et la solidité. Chez TRA-CON, les composés commandés sont : BOND-FDA-6, FDA-8 et TRA-CAST-3103. Ces composés ont séché assez rapidement et ont bien retenu l'électrode au cathéter. Cependant, ces composés sont beaucoup trop rigides. Ils craquent lorsqu'on tente de plier le cathéter. La compagnie Nusil produit des composés qui sont beaucoup plus flexibles. Nous avons utilisé le composé MED-6230 pour fabriquer une électrode. L'élasticité de ce silicone est excellente. Cependant, le problème majeur de ce composé est son temps de séchage. La durée minimale de séchage a été d'environ une semaine. Nous croyons que les faibles contaminations du silicone lors des manipulations auraient été responsables du long temps de séchage du silastic.

2.3.3 Matériau utilisé pour la fabrication de l'électrode

Deux choix s'offraient à nous quant au matériau utilisé pour fabriquer l'électrode : des feuillets métalliques ou des fils en acier. Nous avons choisi en premier lieu de fabriquer l'électrode avec les feuillets en acier à cause de leur faible épaisseur. La compagnie ESPI fabrique des feuillets métalliques très variés pour toutes sortes d'applications. Nous avons commandé des feuillets en acier inoxydable. Ces feuillets ont une épaisseur de 0.1mm. Les dimensions des feuillets sont de 10mm par 100mm de longueur. Des lisières de 5mm ont été tranchées sur une longueur de 100mm. De plus, le côté tranchant des feuillets métalliques représente un problème majeur qui doit être solutionné avant d'insérer une

telle électrode dans un patient. Ces feuillets n'ont pas été retenus pour la fabrication de l'électrode à cause des difficultés qu'elles apportent lors de leur montage sur le cathéter.

La seconde option consiste à enrouler deux fils multi-brins en acier non recouvert de gaine sur le cathéter. De tels fils sont disponibles chez Cooner Wire [27]. L'avantage d'utiliser de tels fils est qu'on n'a plus besoin de faire de soudures. Les fils qui forment l'électrode sont les mêmes qui longent le cathéter et qui se rendent à l'amplificateur externe. Ces fils sont beaucoup plus faciles à manipuler que les feuillets métalliques. Cependant, ils sont plus petits et il faut s'assurer qu'il n'y a pas de petits bouts de fils qui dépassent lors du montage. Ces petits fils pourraient irriter la voie nasale d'un patient lors de l'insertion du cathéter. Deux électrodes expérimentales à grande surface de contact ont été réalisées avec ces fils. Le montage a été relativement long et le séchage a duré près d'une semaine. Une électrode a été utilisée pour prendre quelques mesures cliniques. Les résultats seront présentés au cours du chapitre 5.

Il existe également des fils en acier mono-brins semi circulaires. Ces fils sont fabriqués par la compagnie Microdyne [28]. Le rebord du fil est poli et il n'y a pas de côté tranchant. Ce genre de fil est intéressant pour notre application. Des échantillons ont été commandés et essayés sur le cathéter. Le problème rencontré est que ces fils ne gardent pas leur forme après leur manipulation et de mise en forme particulière. On doit également souder un fil d'acier à ces électrodes. Après avoir effectué plusieurs tentatives

de soudure acier sur acier, nous savons que ces soudures ne sont pas faciles à faire et ne sont pas tellement résistantes.

2.3.4 Montage de l'électrode sur le cathéter

Un montage électromécanique a été réalisé afin de faciliter l'assemblage de l'électrode. Le cathéter est relié à un système de roulement à billes à une extrémité et est relié à un moteur électrique de l'autre. Le cathéter est à l'horizontale est une légère tension est appliquée à chaque bout afin qu'il reste le plus droit possible. Le cathéter tourne constamment et la vitesse de rotation peut être ajustée. Une couche de silastic est appliquée sur le cathéter partout où l'électrode devra être placée. Ensuite, les deux fils sont enroulés simultanément et collées sur le cathéter. Lorsque les fils atteignent le bout de l'emplacement où se termine l'électrode, deux élastiques sont placés afin d'immobiliser les fils sur le cathéter pour le temps du séchage. Une lampe de 100Watts est placée par dessus la région où le silastic a été déposé. Il est important d'assurer une rotation du cathéter afin d'éviter que le silastic ne tombe sur la table.

Nous avons constaté qu'il a été très difficile de monter les feuillets métalliques solidement sur le cathéter en gardant la même distance entre les deux lisières. Malgré toute l'attention portée sur la propreté du montage, il a été impossible de bien monter cette électrode sans toucher au silastic avec les doigts ou les instruments.

2.4 Les méthodes de filtrage passif et numérique

Le présent mémoire ne porte pas sur des méthodes de filtrage. Tel que mentionné dans l'introduction, notre but principal est développer une méthode qui permet d'analyser les caractéristiques d'une électrode quant à sa capacité d'atténuer l'ECG. La raison de ce choix est qu'il sera plus facile de commercialiser un cathéter pouvant s'adapter à tous les systèmes d'enregistrement que l'on retrouve actuellement dans les hôpitaux plutôt que de commercialiser un système plus coûteux qui exige des circuits électroniques dédiés et des microprocesseurs pour réaliser un filtrage plus ou moins complexe (filtrage adaptatif, régression linéaire multiple, etc). Toutefois, à titre de référence, voici des méthodes utilisées pour filtrer les signaux.

2.4.1 Filtrage passe-haut passif

Il n'est pas possible de filtrer simplement l'ECG du signal contaminé. Les composantes fréquentielles cardiaques contiennent de l'énergie entre 0.05 Hz et 100 Hz.

L'EMG_{di} est un signal qui ressemble beaucoup à un bruit. Ansi, on retrouve un contenu fréquentiel qui recouvre une bande de fréquence qui s'étend de 1Hz à 100Hz également. Il y a donc un recouvrement entre les spectres de l'EMG_{di} et l'ECG. Il est possible de filtrer l'EMG_{di} à l'aide d'un filtre passe-haut. Une fréquence de coupure trop élevée risque d'atténuer une portion importante du signal EMG_{di}. Une fréquence de coupure située entre 20Hz et 30Hz donne généralement des résultats passables quant à la suppression de l'ECG tout en préservant le plus d'information sur l'EMG_{di}.

Une fréquence de coupure de 60Hz a comme conséquence d'atténuer l'EMG_{di} sans supprimer davantage l'ECG. D'autre part, une fréquence de coupure de 10Hz ne supprime pas suffisamment la contamination cardiaque [19].

2.4.2 Filtrage adaptatif

Un filtre adaptatif est un système numérique qui adapte sa bande passante selon une fonction qui lui est déterminée. Un signal d'erreur peut être introduit dans le filtre et ce dernier modifiera sa bande passante afin de supprimer ce signal d'erreur. Le filtre adaptatif tente de trouver la présence du signal d'erreur par corrélation avec un patron d'un signal de référence. Lorsque le filtre adaptatif découvre la présence d'un tel signal, le patron de référence est soustrait du signal contaminé. Pour cette application, il s'agirait de fournir le signal contaminé (EMG_{di} et ECG) dans l'entrée du système adaptatif. D'autre part, il faudrait également fournir le signal d'erreur (ECG) qui soit le moins corrélé à l'EMG_{di}. La tâche du système adaptatif sera donc de supprimer l'ECG sans perturber le signal EMG_{di}.

2.4.3 Régression linéaire

Des techniques de traitement de signal basées sur l'analyse des composantes principales et/ou la régression linéaire multiple sont utilisées avec beaucoup de succès pour résoudre des problèmes très semblables, comme la récupération de l'ECG fétal qui est sévèrement contaminé par l'ECG maternel ainsi que le remplacement des dérivations ECG contaminées par l'électromyogramme durant les tests à l'effort sur tapis roulant. Dans le cas présent, on pourrait calculer par régression linéaire multiple une matrice de transfert

entre trois dérivations ECG orthogonales thoraciques et la dérivation œsophagienne durant une période de silence du diaphragme, pour ensuite soustraire le signal œsophagien estimé par cette matrice, du signal mesuré comprenant l'activité du diaphragme.

2.5 Conclusion

Nous avons vu au courant de ce chapitre les méthodes et instruments utilisés pour effectuer des lectures de pression et d'activité électrique musculaire. Nous avons également pris conscience des avantages et des inconvénients associés à ces méthodes.

Enfin, la dernière section de ce chapitre résume deux méthodes de filtrage qui permettent de réduire la présence de l'ECG dans l'EMG_{di}.

Le chapitre 3 s'attaquera plus directement aux solutions proposées pour filtrer l'ECG.

Chapitre 3

Modélisation avec FDTD

3. Introduction

Tel qu'il a été mentionné au chapitre 2, la fabrication d'une électrode est une étape laborieuse. L'étude analytique de certaines électrodes tridimensionnelles complexes peut être très difficile, voire impossible. La simulation numérique est une solution intéressante pour ce problème. Les différences finies permettent d'observer la propagation des ondes électromagnétiques de façon discrète à travers un milieu. Le simulateur FDTD (Finite Difference Time Domain) utilise une version discrétisée des équations de Maxwell et approxime les valeurs des champs électriques et magnétiques dans le temps et dans l'espace avec une précision finie. L'utilisation de cette méthode nous permettra d'étudier les caractéristiques des électrodes oesophagiennes ainsi que sa capacité à capter l'EMG_{di} et à filtrer l'ECG.

Nous désirons simuler la propagation des courants électriques résultants de l'activité musculaire du diaphragme et du cœur dans le thorax. Nous savons que l'activité musculaire des autres muscles du thorax est assez faible pour qu'on puisse la négliger dans nos simulations. Les deux seules sources émettrices de courant seront le cœur et le diaphragme. L'orientation des dipôles émetteurs sera semblable à l'orientation des cellules musculaires des organes respectifs.

Il existe plusieurs méthodes qui permettent d'analyser les champs électromagnétiques dus à des dipôles de courant. Cependant, la complexité de la géométrie des électrodes œsophagiennes rend impossible l'utilisation de méthodes analytiques.

La méthode d'analyse par éléments finis est très intéressante puisqu'elle permet de résoudre certains problèmes à géométrie complexe qui ne pourraient être solutionnés analytiquement. Chaque cellule ou élément est caractérisé par sa forme et ses conditions de frontières, ses valeurs de champs magnétique et électrique ainsi que sa position. Les champs électromagnétiques sont calculés à partir des constantes de propagation de chacune des cellules dans le milieu. Les ondes se propagent à travers les cellules en fonction des constantes de propagation et des conditions aux frontières propres à chaque cellule. Malgré sa popularité, cette méthode est très exigeante en temps d'exécution et les ressources informatiques nécessaires sont imposantes.

Par ailleurs, la méthode des différences finies permet de simuler des problèmes à géométrie complexe sans la lourdeur de calcul de la méthode des éléments finis. La méthode des différences finies effectue les calculs à partir de la matrice des champs électriques et la matrice des champs magnétiques. Dans une troisième matrice, nommée la matrice d'espace physique, sont mémorisés les indices des matériaux. Ces indices servent à associer un matériau déjà défini à un point de l'espace. Les objets tels le thorax, le cœur et le diaphragme sont modélisés par un regroupement d'index dans la matrice

tridimensionnelle d'espace physique. Cette modélisation est une image point à point en trois dimension des matériaux qui composent le thorax.

Chaque point de la matrice est séparé par une distance égale dans l'espace selon les axes X, Y et Z. La distance qui sépare les points de la matrice détermine la précision de la simulation. Plus la distance est faible, plus les valeurs des champs seront exactes. Les composantes X, Y et Z des champs électromagnétiques sont évaluées pour chaque point de l'espace à partir des équations de Maxwell. Puisque les valeurs des champs sont approximées, une erreur s'ajoute à chaque itération. Cependant, nous pouvons réduire cette erreur en réduisant l'incrément de temps sur lequel est basée l'approximation.

3.1 Description de la méthode FDTD

L'utilisation des équations de Maxwell sous forme différentielle a été exploitée par Yee [14] en 1966. Ces équations sont:

$$\frac{\delta \vec{B}}{\delta t} + \nabla \times \vec{E} = 0$$

$$\frac{\delta \vec{D}}{\delta t} - \nabla \times \vec{H} = -\vec{J} \qquad \text{(3.1 a,b,c,d)}$$

$$\vec{B} = \mu \vec{H}$$

$$\vec{D} = \varepsilon \vec{E}$$

Elles peuvent être transformées en équations de différences en utilisant des techniques numériques [23]. Les équations (3.1a) et (3.1b) sont utilisées dans le simulateur FDTD :

la première sert à calculer les composantes du champs électrique et la seconde pour calculer les composantes des champs magnétiques. Les composantes X, Y et Z des champs sont calculées de façon indépendante les unes des autres.

3.1.1 Caractéristiques électromagnétiques du thorax

Le comportement électrique d'un milieu quelconque qui est soumis à un champ électromagnétique peut être caractérisé par une constante que l'on nomme la tangente de perte et qui est égale à :

$$\frac{\sigma}{\omega\varepsilon}$$

où σ est la conductivité électrique du milieu (siemens/m), ω est la fréquence angulaire du champ appliqué (rad/s) et ε est la permittivité du milieu (Farad/m). La tangente de perte fait le rapport entre l'intensité des courants de conduction et l'intensité des courants de déplacement (elle n'a donc pas d'unités). Si ce rapport est élevé (>>1), ceci indique que le milieu est un bon conducteur, s'il est faible (<<1), ceci indique que le milieu est un bon isolant. Il est important de noter que la tangente de perte varie avec la fréquence et qu'un même milieu peut être considéré à la fois comme un bon conducteur et un bon isolant selon la fréquence. Quelle est la tangente de perte pour notre problème?

Selon les tissus que l'on retrouve dans le corps humain, la conductivité électrique varie entre 0,01 S/m (les os) et 0,60 S/m (sang), et sa valeur moyenne pour le thorax est de 0,2 S/m. La conductivité demeure relativement constante en fonction de la fréquence

pour des fréquences inférieures à 10 kHz (CRC). La permittivité des tissus biologiques montre un comportement beaucoup plus complexe car elle diminue en fonction de la fréquence. La permittivité s'exprime par le produit de la permittivité relative ε_r et de la permittivité du vide ε_0 ($8{,}85\times10^{-12}$ F/m). De 10 Hz à 100 Hz, la permittivité relative est très élevée et varie selon le tissu entre $1{,}5\times10^5$ (graisse) et 10^6 (muscle). Il en résulte que la tangente de perte est très élevée (>100) et que l'on peut considérer le thorax comme étant un milieu purement résistif pour des fréquences inférieures à 100 Hz.

Nous pouvons maintenant étudier la propagation des ondes électromagnétiques dans le thorax. La vitesse de propagation (m/s) de l'onde dans un milieu diélectrique est :

$$v = \frac{1}{\sqrt{\mu\varepsilon}} \tag{3.2}$$

Dans le vide, $v=3\times10^8$ m/s. Par contre, dans un milieu conducteur, la vitesse de propagation est plus faible et varie selon la fréquence (Kraus, 1992):

$$v = \sqrt{\frac{2\omega}{\sigma\mu}} \tag{3.3}$$

où μ est la perméabilité du milieu. Pour la majorité des tissus biologiques, la perméabilité est égale à celle du vide ($\mu_0=4\pi\times10^{-7}$). Pour une fréquence de 10 Hz et une conductivité électrique de 0,2 S/m, la vitesse est beaucoup plus faible que dans le vide, soit $v = 2.23\times10^4$ m/s. La longueur d'onde est $\lambda=(v/f)=2230$ m. Comme la longueur d'onde est beaucoup plus grande que la distance entre la source (le cœur) et n'importe quel point du thorax, on peut considérer que le champ électromagnétique est *quasi-statique*. Ceci

signifie que le potentiel électrique produit par l'activation d'une source biologique quelconque se reflète presque instantanément dans tout le thorax et, qu'à toutes fins utiles, il n'y a pas de propagation d'ondes électromagnétiques d'origine biologique à l'intérieur du thorax.

Du point de vue du calcul numérique, puisqu'il n'y a pas de propagation d'ondes, les champs électromagnétiques peuvent être calculés en solutionnant l'équation de Laplace plutôt que les équations de Maxwell, ce qui simplifie beaucoup la complexité mathématique du problème et demande moins de ressources informatiques (temps de calcul et espace mémoire). L'équation de Laplace peut alors être résolue par différentes méthodes numériques, comme les méthodes des éléments finis, des éléments frontières ou des différences finies. Ce sont des méthodes qui ont été déjà utilisées pour simuler les potentiels électrocardiographiques dans des volumes conducteurs thoraciques réalistes (Savard et al. 1985; Shahidi et al, 1994a et 1994b).

Nous avons plutôt choisi de solutionner les équations de Maxwell par la méthode des différences finies dans le domaines temporel (FDTD) même si cette méthode est plus complexe que les méthodes basées sur l'équation de Laplace. La méthode FDTD a récemment été utilisée par plusieurs chercheurs pour étudier l'absorption, dans la tête, des ondes électromagnétiques produites par des téléphones cellulaires (fréquences de l'ordre du gigahertz). Il nous a semblé intéressant de mettre au point des programmes FDTD qui pourront être utilisés par la suite pour aborder d'autres problèmes qui sont à l'étude dans notre laboratoire, comme la transmission, à haute fréquence, de signaux et d'énergie à

travers la peau pour contrôler des dispositifs implantés comme des stimulateurs et des pompes d'assistance ventriculaire.

3.1.2 Critère de stabilité de FDTD

Le simulateur FDTD a originalement été conçu pour calculer la propagation des ondes électromagnétiques à très haute fréquence pour des problèmes où la géométrie rendait impossible une approche analytique. Les signaux à très haute fréquence ont des périodes très courtes. L'incrément de temps ΔT calculé en fonction de la taille des cellules (3.2) est suffisamment petit pour donner une bonne précision et à la fois suffisamment grand pour simuler quelques longueurs d'ondes avec un nombre raisonnable d'itérations. Dans notre application, nos cellules sont très petites et la période des signaux est très longue. La simulation d'une période nécessite donc un très grand nombre d'itérations. Une erreur d'interpolation est ajoutée à chaque incrément de temps. Les erreurs d'interpolation s'ajoutent et peuvent causer une instabilité ou donner des résultats erronés pour un nombre d'itérations élevé. Il faut donc choisir un incrément de temps ΔT maximal en sorte qu'une simulation qui couvre quelques longueurs d'ondes puisse être simulée sans que l'erreur soit importante. Il est préférable d'éviter les longues simulations sachant qu'une erreur s'ajoute à chaque itération.

La précision de la simulation peut être augmentée lorsque la taille des cellules FDTD est réduite. Cependant, l'utilisation de cellules très petites implique que l'incrément de temps ΔT soit très petit afin d'assurer la stabilité et la convergence des résultats. Pour un milieu

sans perte caractérisé par ε et μ, le critère de stabilité de FDTD est décrit par l'équation 3.4 [14] :

$$\sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2} > c\Delta t = \sqrt{\frac{1}{\varepsilon\mu}}\Delta t \qquad (3.4)$$

Pour un milieu avec perte, l'équation (3.3) dans (3.4) donne :

$$\sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2} > v\Delta t = \sqrt{\frac{2\omega}{\sigma\mu}}\Delta t \qquad (3.5)$$

La variable **c** est la vitesse de la lumière dans le milieu caractérisé par ε et μ. Les valeurs Δx, Δy et Δz sont les dimensions des côtés d'une cellule. L'équation (3.5) impose une limite maximale sur l'incrément de temps Δt pour une taille de cellule définie. Cette équation stipule qu'une onde électromagnétique ne puisse parcourir plus que le côté d'une cellule par itération.

3.1.3 Définition des matériaux avec FDTD

Les matériaux peuvent être modélisés à l'aide de trois paramètres : leur permittivité électrique (ε), leur conductivité (σ) et leur perméabilité (μ). Ces trois paramètres sont requis dans les équations de différences. La permittivité et la perméabilité d'un matériau déterminent la vélocité de la lumière dans le milieu. La conductivité détermine la facilité avec laquelle le matériau peut conduire du courant électrique lorsqu'il est soumis à un

champ électrique. Différents matériaux peuvent être modélisés et placés n'importe où dans la matrice tridimensionnelle. Chaque cellule possède un indice qui fait référence à un élément dans la matrice de matériaux. Les trois paramètres du matériau ainsi que des constantes de calcul propres à ce matériau sont mémorisés dans cette matrice. Voici la table de matériaux utilisés :

Tableau 3.1 : Permittivité relative ($\varepsilon_r = 1$), conductivité (σ) et perméabilité relative($\mu_r = 1$) desmatériaux utilisés

Indice	Nom	σ
0	Vide	0
1	Muscle_emetteur	0.75
2	Thorax	0.75
3	Metal	30000.0
4	Air	0.01
5	Contour_du_thorax	0.251
6	Cœur_inactif	0.75
7	Diaphragme_inactif	0.75
8	Œsophage	0.2
9	Eau ($\varepsilon_r = 80$)	0.9

Tous les matériaux que nous utilisons ont une perméabilité relative de 1. Lors des calculs, la permittivité relative des matériaux a été fixée arbitrairement à 1, même si elle est très élevée (voir 3.1.1) car les courants de déplacement sont négligeables par rapport aux courants de conduction [31]. Le simulateur FDTD a été programmé pour utiliser des matériaux dont la perméabilité relative μ_r est unitaire, ce qui est très réaliste. La valeur de la perméabilité a été fixée comme étant constante tout au long de la simulation. Ceci évite plusieurs calculs supplémentaires qui auraient dû être exécutés pour chaque cellule.

3.1.4 Conditions aux frontières

Lorsqu'une onde électromagnétique provenant du cœur s'éloigne du thorax, elle n'est pas réfléchie mais s'échappe et s'atténue dans l'air à l'extérieur du corps. L'utilisation de conditions de frontières absorbantes a pour but de modéliser ces mêmes conditions.

La taille de la matrice dans laquelle est modélisé le thorax est limitée par la taille de la mémoire de l'ordinateur. L'algorithme FDTD ne peut être appliqué aux frontières de la matrice puisque l'algorithme a besoin des valeurs de cellules qui sont situées au delà de la frontière. Cependant, en supposant que la valeur des champs électromagnétiques au-delà des frontières est nulle, on peut résoudre les équations de Maxwell sur les frontières. Cependant, ce type de conditions aux frontières peut ne pas être approprié dans certains cas. Toute l'énergie qui atteint la frontière est réfléchie. Étant donné que la frontière est très rapprochée du thorax, nous ne pouvons la fixer à une constante. Des conditions de frontières constantes seraient plus acceptables si les frontières étaient très éloingées du thorax.

Pour notre application, nous devons avoir des conditions de frontières absorbantes. Afin d'éviter une grande complexité nécessitant des ressources de calcul inaccessibles, le thorax modélisé sera compris entre le cou et la taille. De plus, les conditions aux frontières absorbantes sont essentielles pour assurer la stabilité de l'algorithme FDTD lorsqu'on utilise des sources d'énergie sinusoïdales périodiques. Le cœur et le diaphragme sont modélisés par deux sinus périodiques. Sans ces conditions de frontières

absorbantes, toute l'énergie serait emprisonnée à l'intérieur de la matrice et il y aurait divergence des résultats.

Les conditions de frontières sont calculées à partir d'approximations de l'équation d'onde (3.3) en trois dimensions [15][18]. La valeur du champ sur une cellule de la frontière de la matrice est une approximation dans le temps de sa valeur à partir des valeurs des champs des cellules voisines.

$$\left(\partial_x^2 + \partial_y^2 + \partial_z^2 - c_0^{-2}\partial_t^2\right)E = 0 \tag{3.6}$$

L'équation (3.3) permet de calculer les valeurs futures des champs d'une cellule. Les dérivées selon les axes X, Y et Z représentent la variation du champs dans l'espace. Toutes ces valeurs sont connues et mémorisées dans une matrice. La dérivée selon **t** représente la variation du champ dans le temps. Seules les valeurs antérieures du champ selon **t** sont connues. En numérisant l'équation (3.3), il faut isoler la composante temporelle du champ afin d'obtenir sa valeur future.

3.1.5 Sources émettrices

Deux sources émettrices sont présentes dans notre modèle : le cœur et le diaphragme. Ces sources de courant sont modélisées comme des dipôles dont l'orientation correspond à l'orientation des fibres musculaires dans le cœur et le diaphragme. Ces dipôles sont excités par une onde sinusoïdale. Si les deux organes sont actifs en même temps, la

fréquence des deux ondes sinusoïdales doit être différente. Le signal capté par une électrode œsophagienne peut être séparé en fréquence et on peut observer la quantité d'énergie qui provient de chacun des organes séparément.

3.1.6 Calcul des différences finies

Les différences finies servent à approximer des équations différentielles par des techniques numériques. La propagation des ondes électromagnétiques peut être calculée en appliquant la méthode des différences finies aux équations de Maxwell. Deux matrices tridimensionnelles servent à mémoriser les valeurs des champs électriques et magnétiques. Initialement, toutes les valeurs des champs sont nulles au début de chaque simulation. La valeur des dipôles sinusoïdaux émetteurs sont calculés en fonction du temps et sont mis à jour dans la matrice. Le simulateur calcule ensuite les valeurs des champs électriques en se servant des valeurs des champs magnétiques. Ensuite, le temps est incrémenté d'un demi-pas ($\Delta t/2$). Les champs magnétiques sont ensuite calculés à partir des valeurs des champs électriques.

Les origines des matrices des champs électriques et magnétiques sont distancées d'une demie fois la taille des côtés des cellules. La figure 3.1 illustre les matrices des champs électriques et magnétiques dans un plan. Le même principe est appliqué en trois dimensions. Les champs magnétiques sont calculés à partir des champs électriques situés à un demi incrément dans l'espace. Un demi incrément de temps plus tard ($\Delta t/2$), les

champs électriques sont calculés à partir des champs magnétiques. Cette technique permet d'augmenter la précision à la fois dans l'espace et dans le temps.

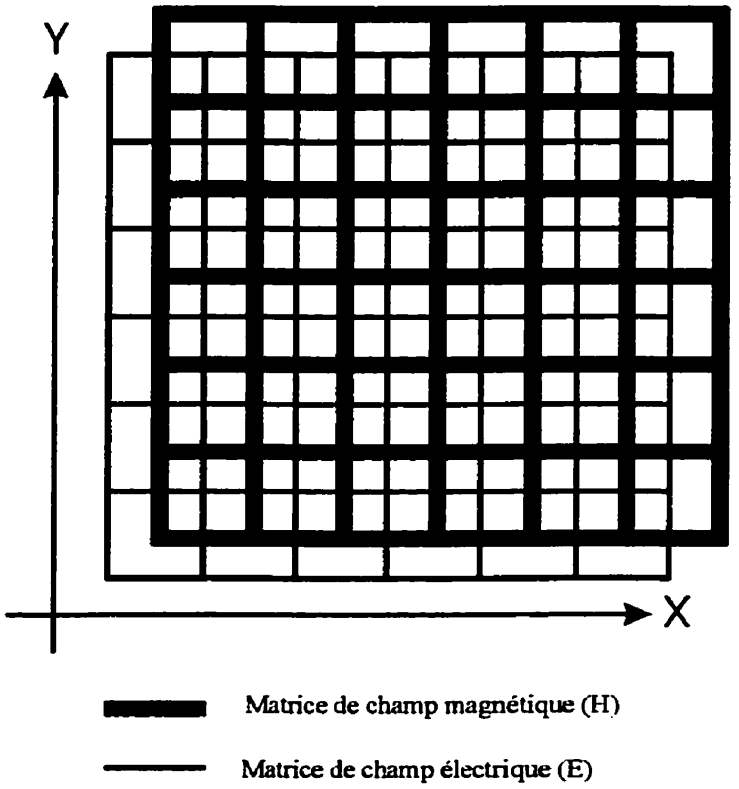

Figure 3.5: Matrices des champs électriques et magnétiques

3.2 Applications de la méthode FDTD à très faible fréquence

La méthode FDTD a originalement été conçue pour des applications à très haute fréquence. Le critère de stabilité décrit à l'équation (3.2) impose une limite maximale sur l'incrément de temps ΔT en fonction de la taille des cellules. Sachant que la période d'un signal à haute fréquence est très courte, il faudra moins d'itérations pour simuler un cycle complet à cette fréquence.

Voici un exemple pour un signal sinusoïdal de fréquence 10 Hz représentatif des sources cardiaques et diaphragmatiques. La période de ce signal est de 1/10 Hz = 0.1 seconde. Sachant que la vitesse de l'onde dans le milieu est de 2.23 km/s et que la taille des cellules est de 6mm, l'équation (3.5) donne une limite maximale pour ΔT de 0.47μs. Ainsi, chaque itération du simulateur fait avancer la simulation de 0.47μs seconde. Il faudra 214 580 itérations pour simuler une période complète du signal à 10Hz.

Nous désirons simuler cinq cycles afin d'atteindre la stabilité à long terme. Sachant qu'une itération prend environ trois secondes sur un PentiumII-266, une simulation complète nécessiterait 37 jours de calcul. Heureusement plusieurs techniques peuvent être appliquées pour diminuer ce temps de simulation.

3.2.1 Multiplication en fréquence

La quantité phénoménale d'itérations requises pour une simulation à très faible fréquence peut être réduite en augmentant la fréquence des sources d'émission du cœur et du diaphragme. Ceci réduit la période du signal et le nombre d'itérations requises pour simuler une période complète. Cette approche sera valide tant que la tangente de perte est beaucoup plus grande que 1.

Cependant, les matériaux ne se comportent pas de la même façon à ces fréquences. Un matériau excité par un signal de 10 MHz aura une permittivité relative beaucoup plus faible que s'il était excité par un signal dont la fréquence serait de 10 Hz. De plus, la

conductivité augmente pour des fréquences supérieures à 10 KHz. Ainsi, pour une fréquence de 10 MHz, une conductivité de 0.95 S/m et une permittivité relative de 200 [34], la tangente de perte est de 8.

En remplaçant les signaux cardiaques et diaphragmatiques par des signaux oscillant à 10MHz, le critère de stabilité est respecté, les courants de conduction dominent et le nombre d'itérations est considérablement réduit.

3.3 Modèle du thorax

Un modèle très simple du thorax comprenant un cœur et un diaphragme a été développé. Le cœur et le diaphragme sont les deux seules sources émettrices dans le corps. La quantité de mémoire d'ordinateur étant limitée, nous ne pouvons pas modéliser le corps en entier. Nous devons donc isoler le thorax et utiliser des parois absorbantes aux limites des matrices des champs électrique et magnétique. La taille des cellules de la matrice est de 6mm et au moins 5 cellules de vide sont requises entre l'extrémité du thorax et la paroi de la matrice.

La taille de la matrice a été calculée et choisie pour minimiser la mémoire requise afin d'accélérer les calculs. Cette taille appropriée est de 70 par 40 par 90 cellules selon les axes X, Y et Z, respectivement.

3.3.1 Le thorax

Le thorax est modélisé par un cylindre elliptique ayant un diamètre de 300 mm (50 cellules) selon l'axe X, 120mm (20 cellules) selon l'axe Y et une hauteur de 270 mm (45 cellules) sur l'axe Z. La conductivité du matériau choisi pour modéliser le thorax a été approximée par la moyenne des conductivités de tissus musculaires et gras présents dans le thorax. Cette approximation est nécessaire pour simuler la contribution des tissus qui n'ont pas été modélisés.

3.3.2 Le cœur et le diaphragme

Le cœur et le diaphragme sont modélisés par deux ensembles de sources ponctuelles de champ électrique (E). Dans le corps humain, ces deux organes émettent de l'énergie simultanément. Les spectres en fréquence de ces sources peuvent se chevaucher pour la plus grande partie du signal. Ceci veut dire que les deux muscles émettent des signaux dont l'énergie est concentrée aux mêmes fréquences. Nous utilisons le principe de superposition entre le cœur et le diaphragme afin d'observer la contribution de ces organes respectifs. Par le fait même, deux simulations seront requises : une première mesurera la propagation de l'énergie provenant du diaphragme lorsque le cœur sera « off » et la deuxième mesurera la propagation de l'énergie cardiaque lorsque le diaphragme sera « off ». Cette analyse en deux étapes nous permettra de déterminer séparément l'effet de chacun des organes émetteurs.

Lors de la simulation les amplitudes des sources de champ électrique ont été fixées arbitrairement à 5.3 V/mm.

Le cœur est modélisé par une sphère dont le diamètre est de 140 mm (24 cellules). Le diaphragme, quant à lui, est modélisé par un paraboloïde elliptique. Le cœur et le diaphragme sont des tissus dont la conductivité est sensiblement la même [16].

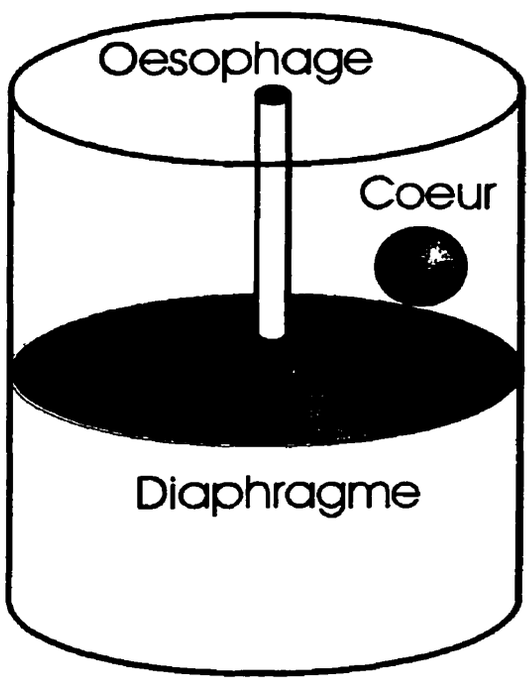

Figure 3.6: modèle thoracique

La figure 3.2 nous montre le modèle du thorax avec les éléments principaux : le cœur, le diaphragme et l'œsophage. L'œsophage est un cylindre dont le diamètre est de 24mm (4 cellules). Ce diamètre est légèrement supérieur à celui d'un vrai œsophage. Un plus grand

nombre de cellules à l'intérieur du diaphragme augmente la précision des résultats qui peuvent être obtenus sur l'électrode placée éventuellement à l'intérieur de l'œsophage.

Toutes les électrodes oesophagiennes qui ont été modélisées ont un diamètre de 2 cellules, environ deux fois plus grande qu'une électrode réelle. L'électrode ne peut toutefois pas être réalisée avec un diamètre d'une seule cellule puisque cette électrode ne serait qu'un bout de fil pour le logiciel FDTD.

Une extrémité de l'œsophage se trouve au sommet du thorax où le thorax est coupé et limité par une paroi absorbante. L'autre extrémité de l'œsophage se retrouve sous le diaphragme.

3.4 Les électrodes oesophagiennes

Le rôle des électrodes est de mesurer l'intensité du champ électrique dans la région du sphincter oesophagien. Ces électrodes oesophagiennes sont généralement construites avec de l'acier inoxydable qui a une conductivité d'environ $\sigma = 30000$ S/m. Toutes les électrodes qui seront construites dans FDTD auront cette conductivité.

3.4.1. L'électrode bipolaire

L'électrode standard pour mesurer l'EMG_{di} est bipolaire. Elle est fabriquée à partir de deux anneaux en acier inoxydable dont le diamètre est de 5 mm. Les deux anneaux sont séparés de 10 mm sur le cathéter. Ce type d'électrode ne peut être modélisé parfaitement avec FDTD puisque les cellules sont cubiques. Les électrodes sont en fait des carrés

puisqu'il est impossible de modéliser un cercle avec des cellules cubiques. Sachant que la taille des cellules du simulateur est de 6mm, le diamètre des anneaux sera de 12mm. La distance entre les anneaux sera également de deux cellules, soit 12mm.

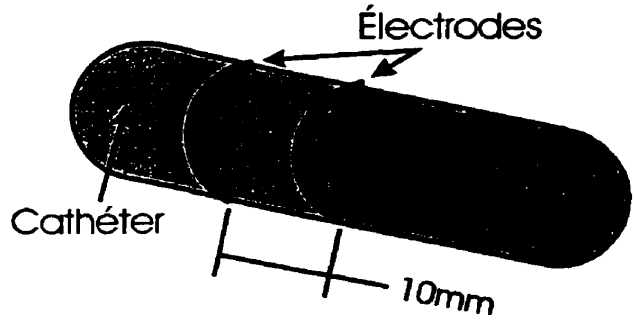

Figure 3.7: Électrode bipolaire standard

3.4.2 Conclusion

Le simulateur décrit dans ce chapitre a été validé en comparant les résultats simulés et la solution analytique d'un problème. Plusieurs points de l'espace ont été échantillonnés et comparés aux valeurs calculées analytiquement. Ces résultats seront montrées dans le chapitre 5. Les résultats du moteur FDTD que nous avons développé en C++ ont été comparés aux résultats d'un moteur FDTD qui a été programmé en FORTRAN. Ces deux outils donnent les mêmes résultats à une décimale près. Cependant, en appliquant les méthodes décrites dans ce chapitre, notre simulateur est en mesure d'effectuer des simulations à très faibles fréquences dans un contexte biologique.

Les conditions aux frontières peuvent jouer un rôle important dans la précision des résultats. Il existe plusieurs types de conditions de frontières pour différentes applications. Il est important de choisir le type le plus approprié pour notre application et de respecter les contraintes propres à ce type de frontière absorbante.

Chapitre 4

PROGRAMMATION DU LOGICIEL FDTD-A

4. Introduction

4.1 Origine de la méthode numérique FDTD

Les premiers calculs numériques des équations de Maxwell datent de la fin des années 1960 avec Yee [14]. À cette époque, les ordinateurs étaient limités en mémoire et en vitesse. Ces limites avaient un impact sur la taille et le nombre de cellules dans une simulation. Or, avec les progrès fulgurants dans le domaine de l'informatique, la méthode FDTD a progressivement pris de plus en plus d'importance dans plusieurs domaines de recherche de pointe. Les modèles sont plus complexes et le temps d'exécution est très raisonnable.

Le moteur de calcul de la méthode FDTD que nous avons utilisée dans nos simulations a été traduit à partir d'une version Fortran [17]. Cette version Fortran a été codée en C++ et optimisée pour être exécutée plus rapidement. Cette nouvelle version a été construite de façon hiérarchique et est un peu plus complexe que la version originale. La programmation par objets d'un tel logiciel augmente la lisibilité du code et facilite l'ajout de modules fonctionnels supplémentaires.

Une bonne partie de la traduction du Fortran à C++ avait déjà été implémentée lorsque ce projet fut mis sur pied. Cependant, le programme comportait beaucoup d'erreurs de

programmation et de synthèse. La complexité du nouveau code menait à certaines impasses qui rendaient plus difficile l'interprétation de la méthode FDTD. Les conditions aux frontières ne fonctionnaient pas correctement. Une grande partie de ce projet consistait à corriger toutes les erreurs de programmation. L'ensemble du code a été révisé et vérifié à plusieurs reprises. Les éléments qui ont été vérifiés et corrigés sont : les moteurs de calculs des champs électriques, magnétiques, les 6 moteurs de calculs de conditions de frontières, les sources d'ondes planes et l'insertion de cellules dans le domaine tridimensionnel, génération des matériaux, construction d'un cube de base.

Suite à la correction et la vérification du moteur de calcul FDTD, d'autres sections ont été ajoutées afin d'augmenter la flexibilité du logiciel. Voici les modules qui ont été rajoutés :

1- Structure des répertoires – DIR.INI
2- Sauvegarde des graphiques pour Matlab – PLANP.INI (Format et script)
3- Table des matériaux éditable – MATERIAL.INI
4- Récupération de simulation antérieure volontairement arrêtée (recovery)
5- Affichage du temps écoulé et restant
6- Lecture lors d'exécution des fichiers de configuration des électrodes – ELECTRODE.INI
7- Génération de fichiers exécutables MATLAB
8- Construction paramétrique du cœur et du diaphragme
9- Construction paramétrique et lecture automatique des électrodes

10- Lecture automatique du potentiel aux bornes d'une électrode

4.2 Initialisation du logiciel FDTD

La phase d'initialisation du logiciel est responsable de recueillir toutes les informations nécessaires au fonctionnement du logiciel à partir des fichiers de configuration. Par la suite, cette information est transmise au logiciel par des variables et des méthodes. Le format des fichiers de configuration ainsi que les répertoires doivent être conformes aux spécifications afin que la phase d'initialisation puisse être exécutée correctement. Les sections qui suivent décrivent les étapes d'initialisation du logiciel avec le format des données d'entrées.

4.2.1 Structure des répertoires

Le logiciel FDTD-A est composé de plus de 70 fichiers de programmation et de configuration. Il nécessite quelques fichiers de paramètres initiaux, une quinzaine de fichiers de récupération de simulation antérieure et peut générer quelques centaines de fichiers de sortie. La complexité d'un tel logiciel nécessite une structure de répertoire afin de simplifier son utilisation.

Tous les fichiers de programmation sont placés dans le répertoire de base. C'est à partir de ce répertoire qu'est exécutée la compilation et l'exécution. Dès que l'exécution est lancée, le programme lit le fichier *DIR.INI* dans lequel sont stockées les informations suivantes :

```
Répertoire des fichiers de reprise d'une simulation
Répertoire des fichiers de données
Démmarage d'une nouvelle simulation (O ou N)
```

Les répertoires décrits dans ce fichier sont relatifs au répertoire de base. Au cours du projet, le fichier *DIR.INI* contenait les informations suivantes :

```
.\recovery\
.\data\
0
```

La dernière information contenue dans ce fichier détermine si la simulation doit débuter au temps zéro. Si **Oui** est spécifié, toutes les valeurs des champs seront initialisées à zéro et la simulation débutera au temps zéro. Si **N** est spécifié, le logiciel fera la lecture de tous les fichiers qui ont été sauvegardés dans le répertoire `.\recovery\` et les chargera en mémoire comme conditions initiales. Dès qu'une simulation termine son exécution, le simulateur sauve toutes les variables nécessaires aux calculs dans le répertoire `.\recovery\`. Ainsi, il est possible de poursuivre une simulation en initialisant le simulateur avec les valeurs qui ont été sauvegardées précédemment.

4.2.2 Sauvegarde des graphiques pour Matlab

Le logiciel doit être accessible à toute personne qui possède peu de connaissance du langage de programmation C++ ou Matlab. Ce critère nécessite donc que le logiciel s'occupe de la génération de fichiers dits *scripts*, c'est-à-dire des fichiers dont le contenu sont des séquences d'opérations qui peuvent être interprétées par un logiciel. La génération du script nécessite certains paramètres que l'utilisateur doit entrer.

Lors du développement d'un modèle tridimensionnel, il est souvent utile de pouvoir visualiser le comportement des champs à l'aide d'un graphique tridimensionnel. Ainsi, on

peut facilement observer toute anomalie ou même la cause d'une divergence qui, autrement, serait difficile à déceler.

Une simulation FDTD peut facilement manipuler des structures de données de plusieurs méga-octets en mémoire. Les deux plus grandes structures sont les tableaux des champs électriques et magnétiques. Chaque tableau contient les valeurs des trois composantes (x, y et z) des champs électriques de chaque cellule. Il est impensable de stocker la totalité de ces tableaux pour chaque incrément de temps à cause de la quantité gigantesque de mémoire que cela nécessiterait.

Une solution moins coûteuse consiste à stocker un ou quelques plans de l'espace. Ces quelques plans nécessitent beaucoup moins de mémoire. Ces plans peuvent être sauvegardés à un intervalle de temps assez distancé. L'observation d'un plan à chaque 50 ou 100 pas peut donner suffisamment d'information pour déceler certains problèmes.

L'information relative à la sauvegarde des résultats est contenue dans le fichier *.\recovery\PLANP.INI* qui a le contenu suivant : `Y 19 100 200 1000`.
Ces informations représentent respectivement: le plan de sauvegarde, la coordonnée du plan de sauvegarde, le temps initial de sauvegarde, temps final de sauvegarde et temps total de la simulation. Le programme FDTD reçoit ces valeurs et génère un fichier script Matlab contenant toutes les informations nécessaires à l'affichage d'un plan. D'autres scripts Matlab a été écrit afin d'afficher successivement plusieurs informations de plans.

Ces scripts peuvent afficher successivement des courbes de niveau et de surface, à la fois pour des fichiers de champs électriques que magnétiques. Ces scripts doivent être dans un répertoire visible par Matlab (path) et se nomment *econt.m, esurf.m, mcont.m et msurf.m.*

4.2.3 Édition de la table des matériaux

Plusieurs matériaux peuvent être utilisés dans FDTD. Quelques uns ont déjà été implémentés mais l'utilisateur peut facilement en rajouter. Le fichier *MATERIAL.INI* contient la liste des matériaux que l'usager a besoin. Les matériaux 0 et 1 sont utilisés par le logiciel pour représenter le vide et un conducteur parfait. L'usager peut donc entrer des matériaux dont l'index suit les matériaux pré-construits. Chaque matériau est caractérisé par un index, une conductivité, sa permittivité et un nom ne dépassant pas 15 caractères. Le fichier doit débuter par un entier déterminant le nombre de matériaux définis par l'usager. Voici un fichier *MATERIAL.INI* :

```
9
Muscle_emetteur 0.05 1
Thorax 0.01 1
Metal 30000.0 1
Air 0.0001 1
Contour_du_thorax 0.005 1
Coeur_inactif 0.5 1
Diaphragme_inactif 1 1
Oesophage 0.02 1
Eau 0.05 1
```

L'édition des matériaux dans un fichier procure plusieurs avantages. L'ajout de matériaux est beaucoup plus simple, rapide et ne nécessite pas de compilation du code. De plus, elle évite que l'utilisateur n'aie à modifier le code source du programme.

4.2.4 Récupération d'une simulation antérieure

Le temps de simulation étant très long, il peut être parfois avantageux de pouvoir reprendre une simulation où elle a été arrêtée plutôt que de la repartir à zéro. Pour ce faire, une série de fichiers doivent être créés afin de sauvegarder toutes les informations contenues dans la mémoire. La simulation ne peut repartir avant que tous les paramètres de la simulation soient chargés. Voici la liste des paramètres qu'il faut sauvegarder :

- Tableau des champs électriques
- Tableau des champs magnétiques
- Tableau des index du model tridimensionnel
- Les 6 tableaux des conditions de frontières
- Paramètres des sources émettrices
- Paramètres de la simulation

Tous ces fichiers sont placés dans le répertoire *.\recovery*. Il est important de ne pas modifier ces fichiers puisqu'une simulation utilise l'information contenue dans ces fichiers comme conditions initiales. Ces fichiers peuvent provenir soit d'un éditeur permettant d'initialiser une simulation ou d'une simulation précédente qui a été interrompue.

4.2.5 Affichage du temps écoulé et restant

Ce module a été rajouté pour faciliter l'organisation des simulations. Il peut prendre plus de 24 heures pour certaines simulations à s'exécuter. Cependant, il est impossible de

savoir à quel niveau sont rendus les calculs et combien de temps va s'écouler avant la fin des calculs.

Ces fonctions calculent le nombre d'itérations qui ont été effectuées et le nombre total d'itérations. À partir de ces valeurs, sachant le temps requis pour exécuter une itération, il est possible d'estimer le temps restant avant la fin de la simulation.

Au cours de la simulation, plusieurs valeurs doivent être mémorisées sur disque. Pour accélérer le processus d'écriture, ces valeurs sont mémorisées dans la mémoire tampon du disque rigide. Dès que la mémoire tampon est pleine, le bloc est écrit sur le disque rigide. Cependant, il peut s'écouler beaucoup de temps avant que ce tampon soit plein. Ainsi, pendant une simulation, il est impossible de lire les fichiers de sortie afin d'observer les résultats les plus récents, à moins que la mémoire tamponne soit récemment vidée. Pour permettre une telle observation, une fonction spéciale a été implantée afin de transférer immédiatement la mémoire tampon sur le disque. Ainsi, lorsqu'on se rend compte qu'une simulation diverge, on peut l'arrêter immédiatement, ce qui nous permet de sauver beaucoup de temps. Pendant la simulation, on peut appuyer sur la touche **F** pour transférer la mémoire tampon sur le disque. La touche **Q** arrête la simulation et mémorise l'état complet de la simulation, afin qu'elle puisse être redémarrée et qu'elle puisse continuer la simulation à partir du point où elle a été interrompue.

4.2.6 Construction du modèle

L'espace mémoire est une ressource très précieuse et on doit en tenir compte lorsqu'on construit des modèles tridimensionnels avec FDTD. Le simple fait de doubler les dimensions de l'espace augmente la taille de la mémoire requise par huit. Cette limite sur la mémoire nous oblige à faire certains compromis. Nous souhaitons que notre simulation ait une précision raisonnable et un temps d'exécution assez court. Plus la taille des cellules est petite, plus la précision est bonne. Cependant, plus la taille est petite, plus ΔT est court (critère de stabilité) et il prendra plus de temps pour simuler une longueur d'onde. De plus, il faudra plus de cellules dans le milieu pour modéliser les objets (thorax, cœur, diaphragme).

Nous avons choisi de modéliser le thorax avec des cellules de 6mm de côté. Notre modèle ne représente que la partie centrale du thorax, c'est-à-dire de la ceinture à la gorge et il coupe aux épaules. Le modèle a 70 cellules de largeur par 90 cellules de hauteur par 40 cellules de profondeur, ce qui correspond environ aux dimensions d'une personne de taille normale.

Nous avons modélisé le thorax par un cylindre elliptique puisque cette forme ressemble beaucoup à la forme d'un vrai thorax. La taille de l'espace mémoire a été choisie de sorte qu'il n'y ait que cinq cellules de vides entre l'extrémité du thorax et la paroi absorbante du modèle.

Le diaphragme a été modélisé par un paraboloïde elliptique inversé comme on peut le voir à la Figure 4.1. Cette figure a été dessinée avec le logiciel Mathematica. L'endroit où le thorax coupe le diaphragme est illustré par l'ellipse dans le plan z=0.

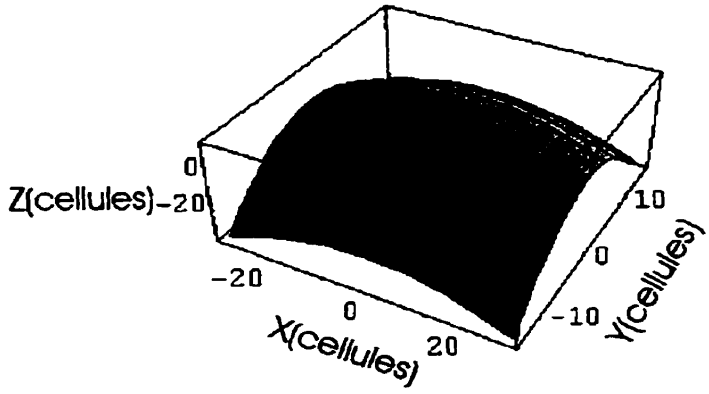

Figure 4.1: Modélisation mathématique du diaphragme

Le centre du diaphragme a été troué afin de permettre l'insertion du cathéter et de l'œsophage. Les dipôles de courant sont orientés dans la même direction que les fibres musculaires du diaphragme. On voit bien cette ouverture au sommet du diaphragme sur la figure 4.2.

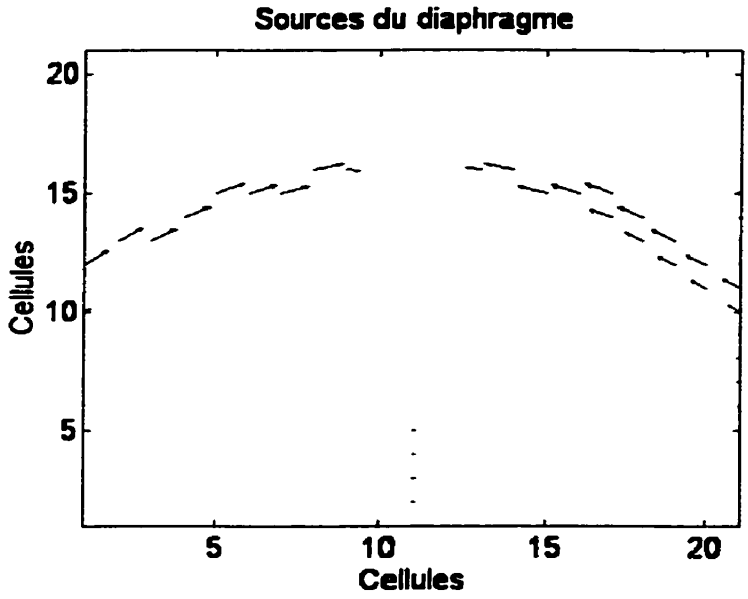

Figure 4.2: Sources du diaphragme

Chacune des sources du diaphragme peut être activée indépendamment. Deux fonctions ont été programmées dans le logiciel, mais d'autres fonctions peuvent très bien être rajoutées. Pour l'instant, chaque source peut être activée selon une sinusoïde ou un pulse gaussien. Différentes phases peuvent être appliquées à chaque cellule et même de façon aléatoire.

Le cœur a été modélisé comme une sphère pleine. Deux patrons d'activation du cœur peuvent être sélectionnés : l'activation tangentielle ou normale. Ces deux patrons sont illustrés sur les deux figures 4.3 et 4.4.

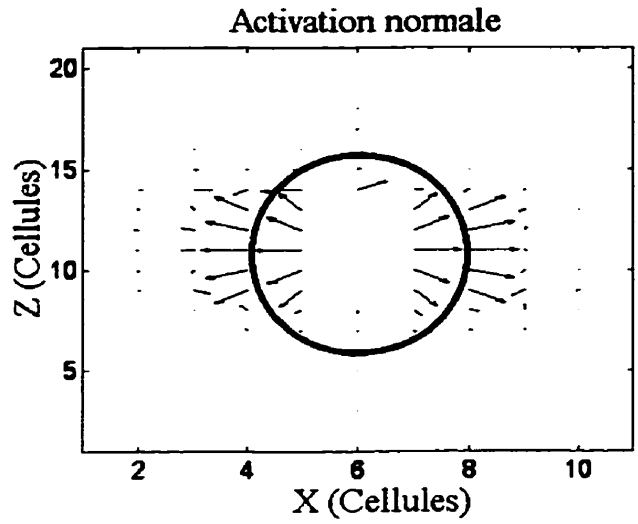

Figure 4.3: Activation normale du cœur

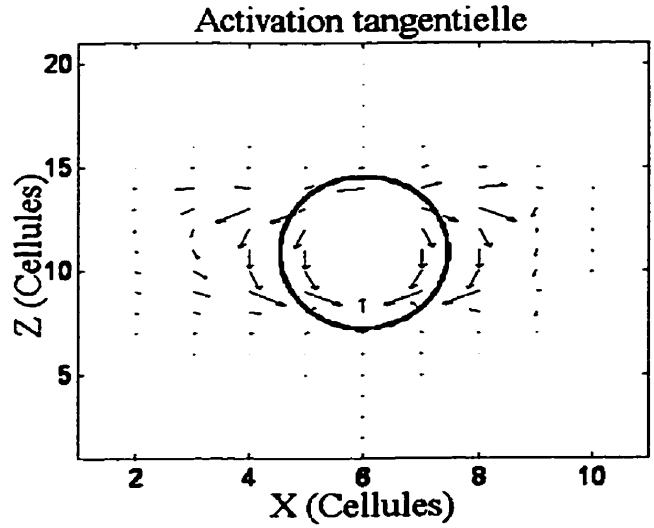

Figure 4.4: Activation tangentielle du cœur

Le cœur est illustré par un cercle sur les deux figures 4.3 et 4.4. Les flèches représentent l'amplitude du champ électrique source à chaque point de la matrice. L'orientation des flèches représentent la direction selon laquelle les champs électriques sont orientés. Les deux patrons de sensibilité déterminent de quelle manière l'énergie sera émise dans le thorax. Avec le patron de d'activation normale, l'énergie est émise comme une onde sphérique qui se propage partout dans le thorax. D'autre part, le patron d'activation tangentielle stimule un courant vers le bas du cœur. Ce dernier patron ressemble plus au vrai patron d'émission de courant du cœur. Cependant, le vecteur résultant de l'activation du cœur (le sens dans lequel le cœur émet un courant) n'est pas constant durant un battement. À première vue, il peut être modélisé par un vecteur constant orienté vers le bas. Les vecteurs tangents dans le modèle de la figure 4.4 modélisent plus les fibres musculaires du cœur. Le modèle tangentiel sera utilisé pour mesurer la quantité d'énergie cardiaque qui peut se rendre jusqu'à l'électrode. Quant au patron de polarisation normale, ce dernier est utilisé principalement pour tenter de comprendre le problème de divergence des champs aux frontières. Ce patron ne modélise aucune structure biologique.

4.2.7 Construction des électrodes

Plusieurs outils ont été programmés afin d'automatiser la construction d'électrodes dans le milieu. Des nombreux paramètres peuvent être spécifés pour la construction des électrodes. De plus, plusieurs électrodes peuvent être simultanément positionnées et lues dans le milieu. Les lectures de chaque électrode sont sauvegardées dans des fichiers indépendants. Les électrodes peuvent être fabriquées à partir de lignes parallèles aux axes

X, Y et Z seulement. On peut toutefois approximer une diagonale en combinant des composantes des trois axes de sorte que la résultante vectorielle pointe dans la direction désirée.

4.3 Validation du moteur de calcul FDTD

Le moteur de calcul ainsi que les conditions de frontières font le principal objet de la validation. Ces deux éléments jouent un rôle très important dans la précision et l'exactitude des résultats. Une simple erreur de programmation pourrait fausser complètement les résultats. Il est donc important de s'assurer que tout fonctionne correctement en ce qui concerne cet aspect du logiciel.

Un problème dont la solution analytique est connue a été simulée avec FDTD. Sachant la solution analytique, nous pouvons vérifier quelques points de l'espace afin de s'assurer que le simulateur obtienne des résultats semblables. Ce problème consiste à polariser une sphère par une onde plane qui se propage selon l'axe X. Une table d'échantillons recueillis à quatre endroits différents pour les 256 premières itérations est donnée [1]. Il s'agit de comparer nos résultats avec ceux présentés dans cette table. Les résultats de la validation sont présentés au cours du prochain chapitre.

4.4 Conclusion

Le présent chapitre regroupe l'ensemble des paramètres qui permettent d'utiliser et de programmer le logiciel FDTD. La structure hiérarchique de la programmation permet l'ajout de modules facilement. Le logiciel est à la fois flexible et modulaire. La flexibilité

qu'offre FDTD par rapport aux autres méthodes est grande. Des structures complexes peuvent être simulées sans problème. Voilà pourquoi il est important de donner tous les outils nécessaires pour créer des structures complexes à l'intérieur du modèle.

La validation du moteur de calcul nous assure que le simulateur fonctionne efficacement. Toutefois, les conditions de simulation doivent être valides au point de vue des limites permises pour la précision des calculs. Un tel système requiert plusieurs paramètres pour fonctionner et ces paramètres doivent satisfaire à certaines exigences comme le critère de stabilité de FDTD et la distance minimale entre le thorax et les frontières de la matrice.

Chapitre 5

Résultats

5. Introduction

Ce chapitre sera divisé en plusieurs parties qui sont les suivantes:

1. Lectures expérimentales des signaux provenant d'une électrode prototype vs électrode bipolaire
2. Courbes de lecture de pression par capteur SM5102-005-A avec et sans silastic
3. Validation du simulateur FDTD
4. Courbes expérimentales de l'EMG_{di} en fonction de la hauteur dans l'œsophage
5. Courbes simulées de l'EMG_{di} en fonction de la hauteur dans l'œsophage

Dans chaque section, les résultats seront d'abord présentés puis analysés. Chaque partie énumérée ci-haut constitue les étapes importantes menant à la construction d'une électrode prototype optimisée pour la lecture de l'EMG_{di}.

5.1 Lectures expérimentales par électrode prototype et bipolaire

Deux électrodes prototypes ont été fabriquées tel que décrit précédemment dans ce mémoire. Cependant, seulement une électrode a pu être essayée. Le silicone sur l'autre électrode n'a pas durçi suffisamment pour qu'on puisse les tester chez un patient sans risques. Les figures 5.1 et 5.2 montrent les courbes de signaux EMG_{di} qu'on a captés avec notre électrode prototype et avec l'électrode bipolaire respectivement.

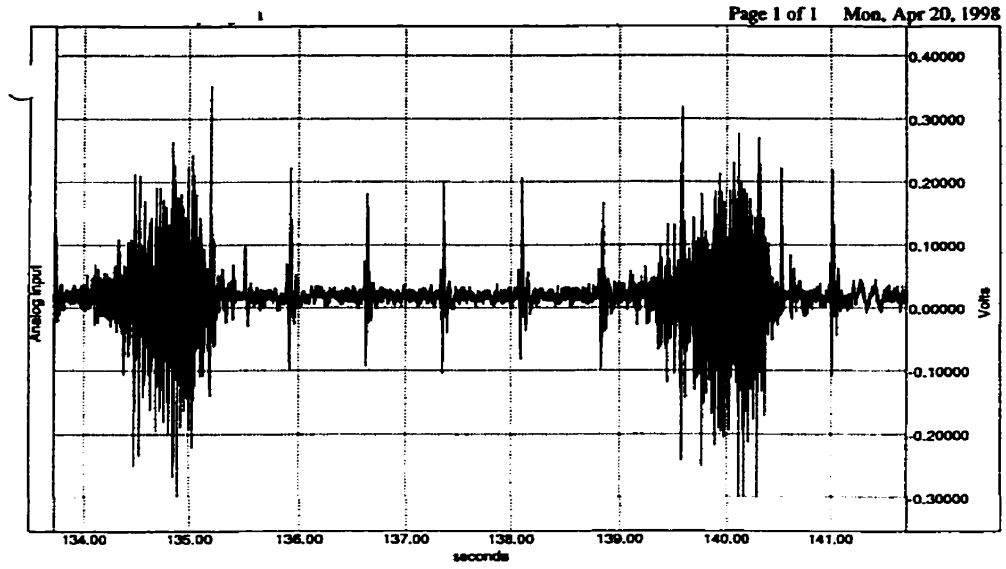

Figure 5.1: EMG_{di} et ECG capté avec une électrode bipolaire

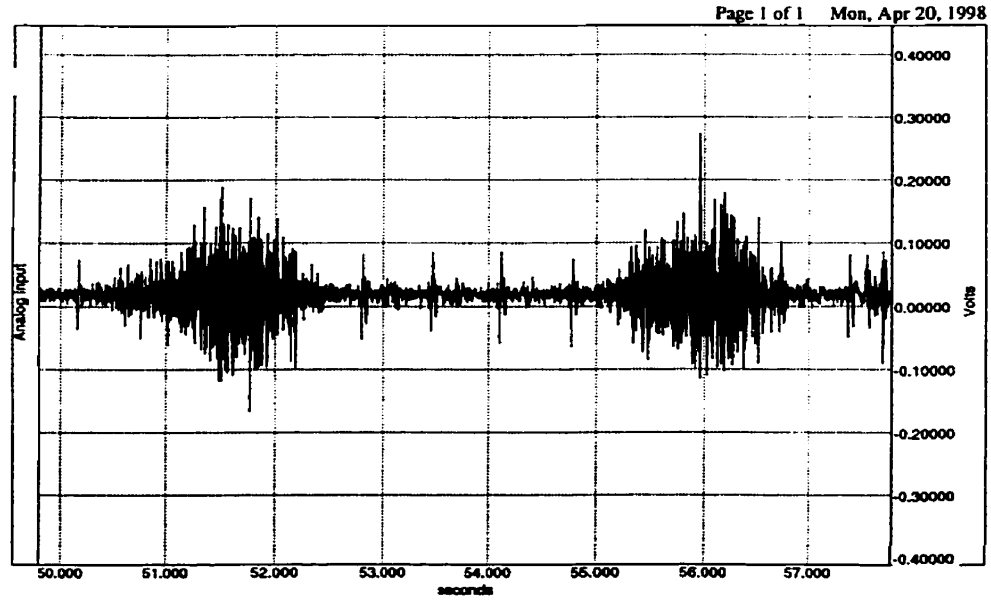

Figure 5.2: EMG_{di} et ECG capté avec l'électrode prototype

On constate que l'amplitude de l'ECG qui apparaît sur la Figure 5.2 est environ de moitié de ce qu'il est sur le graphique de la Figure 5.1. Il est également possible de constater que l'amplitude de l'EMG_{di} est un peu plus faible du côté de l'électrode hélicoïdale que du côté de l'électrode bipolaire. Cependant, le but de cette recherche vise à obtenir un rapport EMG_{di} / ECG le plus élevé possible.

5.2 Lectures de pression par capteur couvert par silastic

Le but de cette expérience est de vérifier la linéarité du capteur de pression de la compagnie Silicone-Microstructures dans la marge de pression allant de 0mmHg à 150mmHg par rapport à la pression atmosphérique. De plus, nous pourrons évaluer la perte de précision du capteur de pression occasionnée par l'ajout d'une mince couche de silastic par dessus le diaphragme. Du même coup, nous pourrons observer la résistance des *fils d'interconnexion (puce au substrat)* lors de la mise en place de la mince couche de silastic.

Les mesures seront effectuées en deux temps : d'abord, des mesures seront prises à partir du capteur de pression dont le diaphragme sera exposé à l'air libre ; ensuite, la même expérience sera effectuée sur le capteur de pression dont le diaphragme sera recouvert d'une mince couche de silastic.

5.2.1 Méthodologie

Pour mesurer la pression, le capteur est déposé dans un contenant fermé hermétiquement. Une pompe à air est installée afin de pouvoir faire varier la pression à l'intérieur du contenant. Une colonne de mercure est également reliée à l'intérieur du

contenant afin de vérifier la pression relative dans le contenant. Le schéma du montage est présenté à la Figure 5.3. La première expérience s'est effectuée sur le capteur de pression dont le diaphragme donne sur l'air ambiant.

Pour la seconde expérience, une mince couche de silastic est déposée par dessus le diaphragme du capteur de pression. Cette couche recouvre également les *fils d'interconnexion* qui relient les plots de contacts du capteur de pression à la micro-plaquette. L'épaisseur de la couche de silastic est d'environ 0.25mm.

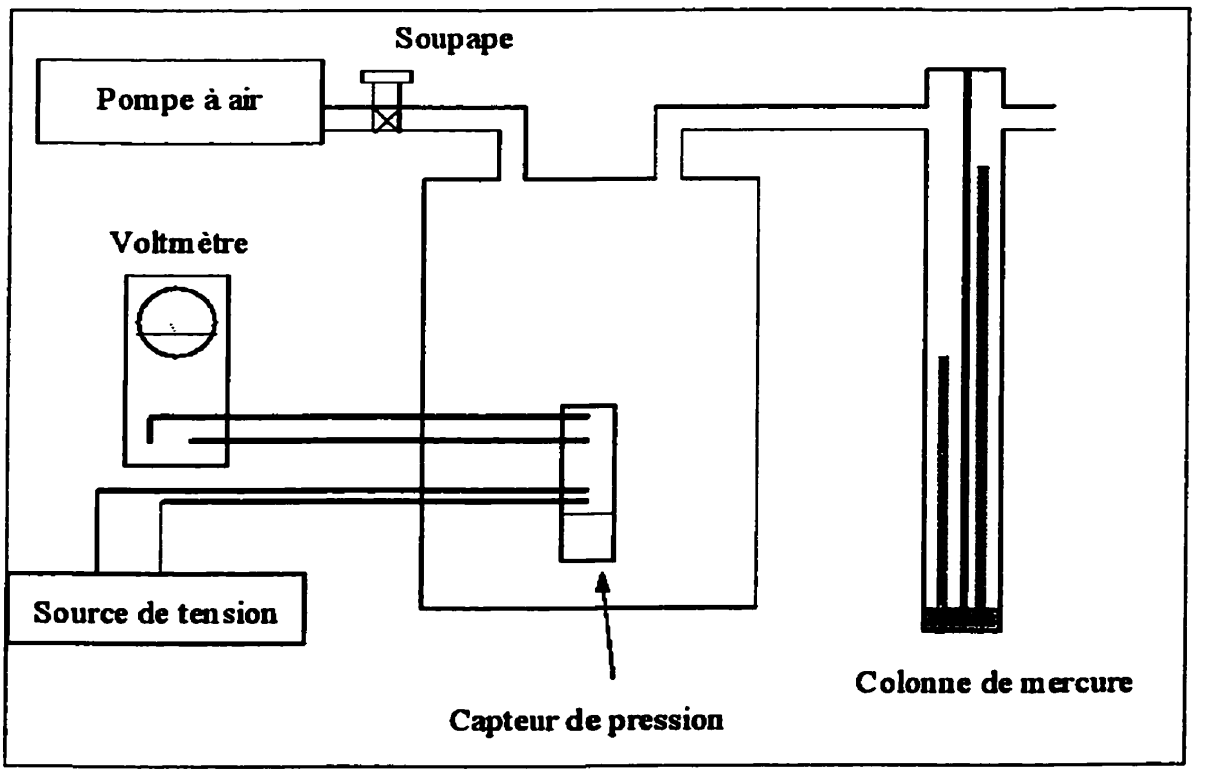

Figure 5.3: Schéma du montage servant à mesurer la sensibilité du capteur SM5102-005A avec et sans membrane de silastic.

Pour ces deux expériences, il s'agit de faire varier la pression à l'aide de la pompe et de calculer la tension aux bornes du voltmètre en fonction du niveau de mercure indiqué sur la colonne.

5.2.2 Lectures Pression-Tension sans couche de silastic

Les résultats qui sont présentés dans cette section ont été obtenus avec le capteur de pression sans recouvrement de silastic. Avec les données du Tableau 5.1, nous pouvons calculer la pente, l'écart type ainsi que la régression linéaire pour ainsi obtenir la sensibilité du capteur de pression.

Tableau 5.1: Lectures Pression-Tension avec capteur sans recouvrement de silastic

cmH2O	mmHg	Volts
1.36	1.0	0.6310
2.72	2.0	0.6325
4.08	3.0	0.6330
5.44	4.0	0.6340
6.81	5.0	0.6375
8.17	6.0	0.6390
9.53	7.0	0.6415
10.89	8.0	0.6430
12.25	9.0	0.6430
13.61	10.0	0.6445
16.33	12.0	0.6470
19.06	14.0	0.6500
20.42	15.0	0.6520
21.78	16.0	0.6530
25.86	19.0	0.6600
27.22	20.0	0.6580
31.31	23.0	0.6570
39.47	29.0	0.6700
42.20	31.0	0.6730
61.25	45.0	0.6890
77.59	57.0	0.7020
88.48	65.0	0.7190
89.84	66.0	0.7170
95.28	70.0	0.7260
106.17	78.0	0.7390
129.31	95.0	0.7560
136.12	100.0	0.7650
140.20	103.0	0.7670
160.62	118.0	0.7880
164.71	121.0	0.7910
170.15	125.0	0.7980
197.37	145.0	0.8250

La pression atmosphérique absolue de la journée était de 102.5KPa (1046.5cmH2O = 768.8mmHg$_{absolue}$).

Les résultats montrés aux équations 5.1 à 5.3 ont été obtenus avec les fonctions du logiciel EXCEL.

$$\text{Pente}_{air} = 1.334\text{mV / mmHg} = 0.98\text{mV/cmH}_2\text{O} \quad (5.1)$$

$$\sigma_{air} = 1.9\ \text{mV} \quad (5.2)$$

$$R^2 = 0.9989 \quad (5.3)$$

Où σ représente l'erreur type et R^2 représente le carré du coefficient de corrélation. Les résultats présentés au Tableau 5.1 la droite moyenne de ces lectures calculée aux équations (5.1) à (5.3) ont été compilés et sont présentés à la Figure 5.4:

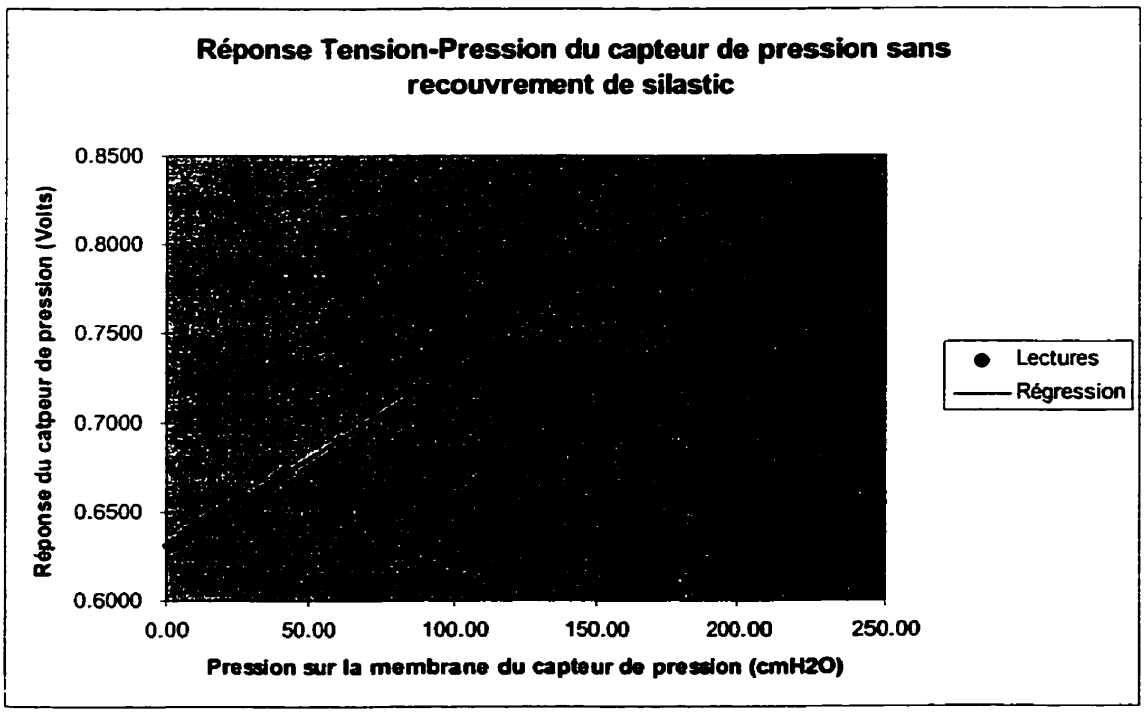

Figure 5.4: Tension en fonction de la pression mesurée sans recouvrement de silastic

Sur la Figure 5.4, une droite de pente moyenne passe à travers le nuage de points. Ces points sont les lectures qui ont été effectuées au laboratoire. La pente de la droite a été calculée par la méthode des moindres carrés.

5.2.3 Lectures Pression-Tension avec une mince couche de silastic

Les résultats qui sont présentés dans cette section ont été obtenus avec le capteur de pression avec une mince membrane de silastic. Avec les données du tableau 5.2, nous pouvons calculer la pente, l'écart type ainsi que la régression linéaire pour ainsi obtenir la sensibilité du capteur de pression. Ces valeurs nous permettront de comparer la sensibilité du capteur de pression lorsqu'une couche de silastic recouvre la membrane du capteur de pression.

Tableau 5.2: Lectures Pression-Tension du capteur avec recouvrement de silastic

cmH2O	mmHg	Volts
1.36	1	0.6260
2.72	2	0.6270
2.72	2	0.6330
4.08	3	0.6290
5.44	4	0.6300
5.44	4	0.6370
6.81	5	0.6320
8.17	6	0.6330
9.53	7	0.6340
13.61	10	0.6400
17.70	13	0.6440
25.86	19	0.6530
27.22	20	0.6540
27.22	20	0.6600
29.95	22	0.6630
42.20	31	0.6680
51.73	38	0.6800
55.81	41	0.6870
63.98	47	0.6910
69.42	51	0.7000
73.50	54	0.7030
83.03	61	0.7190
92.56	68	0.7260
102.09	75	0.7310
103.45	76	0.7350
108.90	80	0.7370
119.79	88	0.7465
130.68	96	0.7610
133.40	98	0.7640
138.84	102	0.7680
155.18	114	0.7910
170.15	125	0.7960
171.51	126	0.8000
186.48	137	0.8130
197.37	145	0.8230

La pression atmosphérique absolue de la journée était de 101.77KPa (1038.85cmH_2O = 763.3mm$Hg_{absolue}$).

Ces données nous donnent les résultats suivants

$$\text{Pente}_{\text{silastic}} = 1.379\text{mV / mmHg} = 1.01\text{mV/cmH}_2\text{O} \quad (5.4)$$

$$\sigma_{\text{silastic}} = 3.16\ \text{mV} \quad (5.5)$$

$$R^2_{\text{silastic}} = 0.9974 \quad (5.6)$$

Où σ représente l'erreur type et R^2 représente le carré du coefficient de corrélation. On constate un léger écart entre la sensibilité du capteur avant silastic (5.1) par rapport à sa sensibilité après l'ajout de la membrane de silastic (5.4). Cet écart est attribuable aux erreur de lectures ainsi qu'à l'instrumentation qui a été utilisée lors des manipulations. Les résultats présentés au Tableau 5.2 la droite moyenne de ces lectures calculée aux équations (5.4) à (5.6) ont été compilés et sont présentés à la Figure 5.5:

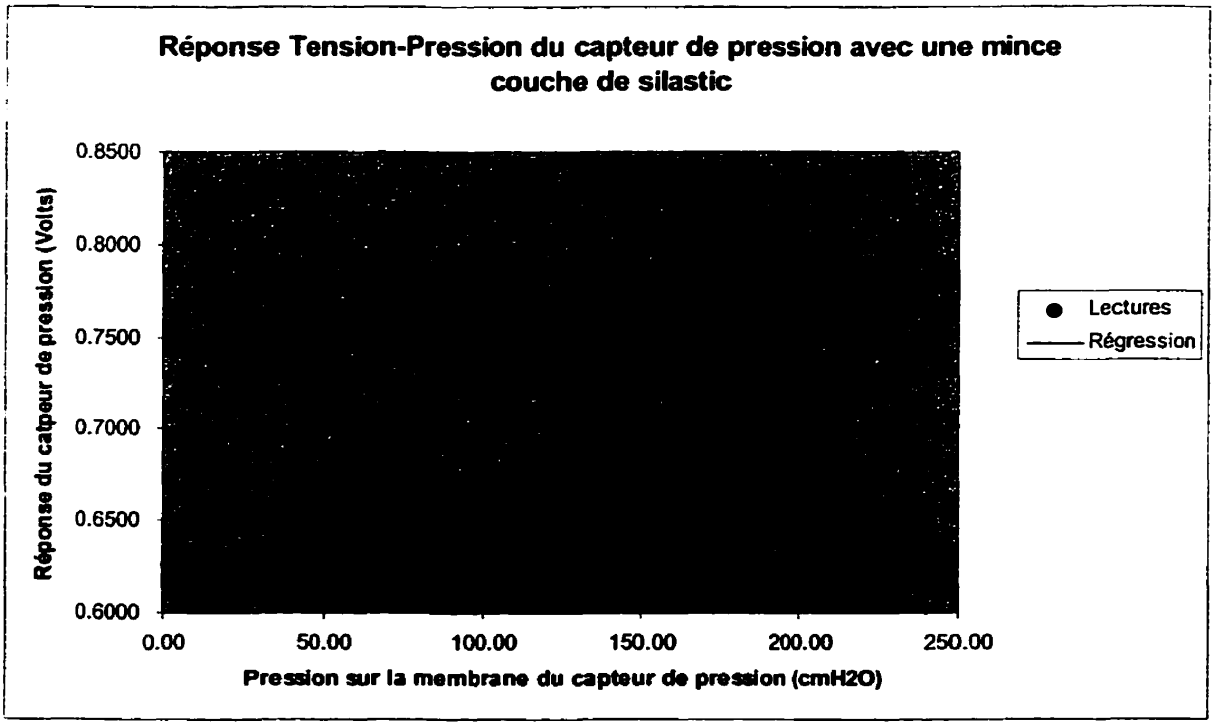

Figure 5.5 : Tension en fonction de la pression mesuré avec recouvrement de silastic

5.2.4 Analyse des résultats

Un des buts de cette expérience était de vérifier le fonctionnement linéaire du capteur de pression dans la marge de pression allant de 0mmHg à 150mmHg par rapport à la pression atmosphérique. Nous pouvons donc conclure que le capteur de pression a un comportement très linéaire dans la marge de pression variant entre 0mmHg et 150mmHg par rapport à la pression atmosphérique puisque le coefficient de corrélation des mesures expérimentales est de plus de 0.997 dans les deux cas.

La pente moyenne des résultats expérimentaux nous donne une indication de la résolution du capteur de pression. Le signal de sortie varie de 0.98mV/cmH_2O pour le capteur sans silastic à 1.01mV/cmH_2O pour le capteur recouvert de silastic. Puisque la précision requise pour notre application est de 0.01cmH_2O, il faudra un amplificateur capable de mesurer des tensions de près de 0.1mV. Il est également important de souligner la similitude des pentes pour le capteur avec et sans silastic. Cette similitude nous indique que l'ajout d'une mince couche de silastic ne modifie pratiquement pas la sensibilité du capteur de pression.

5.3 Validation du simulateur FDTD

Le moteur de calcul FDTD que nous utilisons a été développé par M. Luebbers[17]. Pour valider le simulateur, un problème simple dont la solution analytique est connue a été simulé. Ce problème consiste à émettre une onde plane dans le milieu où il n'y a que la présence d'une sphère ayant une permittivité relative de 4 et un rayon de 8mm. On

échantillonne le champ électrique à deux endroits et le champ magnétique à deux autres endroits. Les échantillons sont captés pendant 255 itérations. La composante X du champ électrique (Ex) a été échantillonné dans l'espace à deux coordonnées : (17,18,25) et (17,18,18). La composante Y du champ magnétique (Hy) a également été échantillonné à deux endroits : (17,18,24) et (17,18,17). Ces échantillons sont présentés aux Tableaux 5.3 et 5.4. La stimulation est une onde plane dont la forme est un pulse gaussien. Cette vérification s'effectue à haute fréquence et ne sert uniquement à vérifier le bon fonctionnement du moteur de calcul à haute fréquence.

Tableau 5.3 : Échantillons de référence calculés par le logiciel FDTD en FORTRAN

	Ex		Hy	
Itération	(17,18,25)	(17,18,18)	(17,18,24)	(17,18,17)
10	0.00E+00	0.00E+00	0.00E+00	0.00E+00
100	-8.31E+02	-3.04E+02	-1.93E+00	1.53E+00
150	-4.93E+02	-9.20E+01	-1.26E+00	-9.51E-01
200	-3.91E+01	-6.91E+01	1.05E-01	-1.08E-01
256	-3.65E+01	-4.32E+01	1.30E-01	4.92E-02

Tableau 5.4 : Échantillons calculés par le logiciel FDTD programmé en C++

	Ex		Hy	
Itération	(17,18,25)	(17,18,18)	(17,18,24)	(17,18,17)
10	0.00E+00	0.00E+00	0.00E+00	0.00E+00
100	-8.31E+02	-3.04E+02	-1.93E+00	1.53E+00
150	-4.95E+02	-9.23E+02	-1.25E+00	-9.49E-01
200	-3.91E+01	-6.41E+01	1.04E-01	-1.28E-01
256	-3.55E+01	-4.23E+01	1.24E-01	4.32E-02

En comparant les deux tableaux précédents, on remarque que les valeurs après 256 itérations sont presque identiques. Ces légères différences peuvent être causées par plusieurs facteurs. Ces erreurs peuvent provenir de différences entre les librairies FORTRAN77 et C++, du compilateur, de plateforme d'exécution ou des erreurs d'arrondis dans le calcul des constantes. Malgré ces différences, on peut conclure à l'aune de ces résultats que le moteur de calcul FDTD que nous utilisons est fonctionnel.

5.4 Courbes EMG_{di}-ECGexpérimentales en fonction de la profondeur dans l'oesophage

Des mesures de l'ECG et EMG_{di} ont été captées avec une électrode bipolaire standard. Plusieurs lectures ont été effectuées et notées en fonction de la profondeur du cathéter dans l'œsophage. L'insertion du cathéter a été effectuée par les voies nasales. La profondeur est calculée comme étant la distance séparant l'électrode du bout du nez. Ces lectures serviront à valider le simulateur FDTD qui devra obtenir des valeurs similaires.

Le but étant de valider le modèle informatique, nous devons comparer les résultats simulés aux résultats obtenus expérimentalement. Seule l'électrode bipolaire a été simulée. L'électrode bipolaire a une géométrie simple et elle est très facile à modéliser avec FDTD, ce qui n'est pas le cas avec l'électrode prototype. De plus, les résultats expérimentaux obtenus avec l'électrode bipolaire sont très accessibles.

Le peu de résultats expérimentaux que nous possédons avec l'électrode prototype ne nous permet ni de comparer ni de valider efficacement les résultats simulés avec cette

électrode prototype. La simulation de l'électrode prototype ne pourra être faite qu'après la validation du modèle informatique avec l'électrode bipolaire.

Tableau 5.5 : Lectures EMG et ECG en fonction de la profondeur

Profondeur (cm)	EMGdi (mV)			ECG (mV)		
	1	2	3	1	2	3
60	0.202	-	-	0.038	-	-
62.5	0.270	0.140	-	0.130	0.040	-
66	0.826	0.794	-	0.300	0.300	-
71	0.250	0.200	0.200	0.190	0.270	0.160
75	0.165	0.100	-	0.180	0.025	-

Toutes les valeurs des colonnes 1, 2 et 3 du Tableau 5.5 n'ont pas été amplifiées et correspondent à la différence de potentiel qui a été observée directement aux bornes de l'électrode bipolaire. Les valeurs représentent l'amplitude maximale des signaux EMG_{di} et ECG. Une première séance de lecture s'est effectuée en descendant l'électrode dans l'œsophage et une seconde séance de lecture en remontant l'électrode. Des lectures supplémentaires ont été effectuées aux endroits où les signaux EMG_{di} et ECG étaient plus importants. Pour certaines profondeurs, plusieurs lectures ont été effectuées. C'est pour cela qu'on observe plusieurs mesures à la même profondeur. Ces mesures serviront à comparer les résultats simulés avec FDTD. On devrait normalement arriver à des valeurs similaires à celles du Tableau 5.5.

À partir du Tableau 5.5 on peut générer un graphique de l'amplitude maximale des signaux ECG et EMG_{di} en fonction de la profondeur de l'électrode.

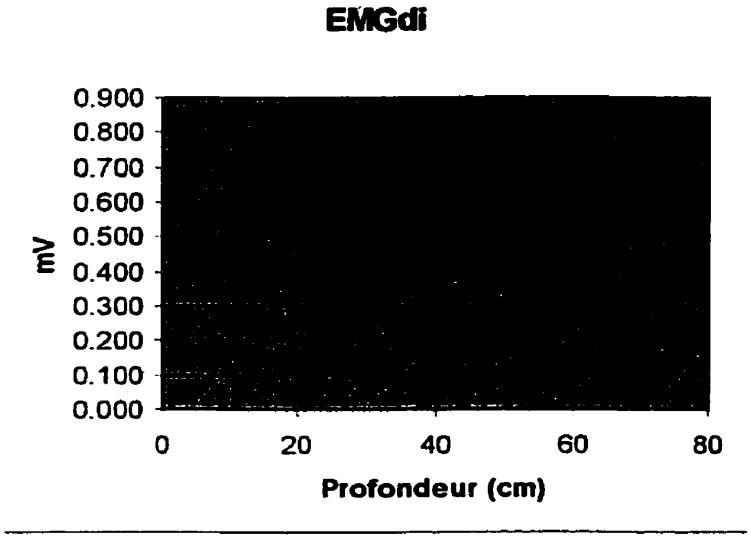

Figure 5.6: Amplitude de l'EMGdi en fonction de la profondeur

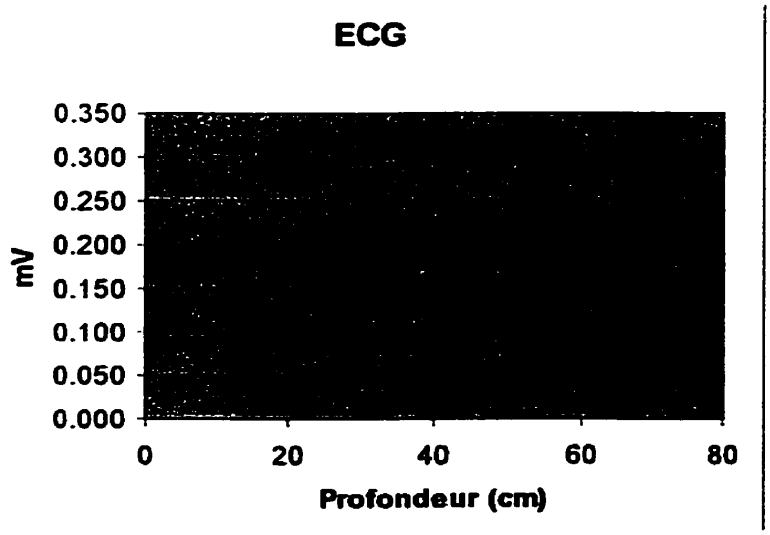

Figure 5.7: Amplitude de l'ECG en fonction de la profondeur

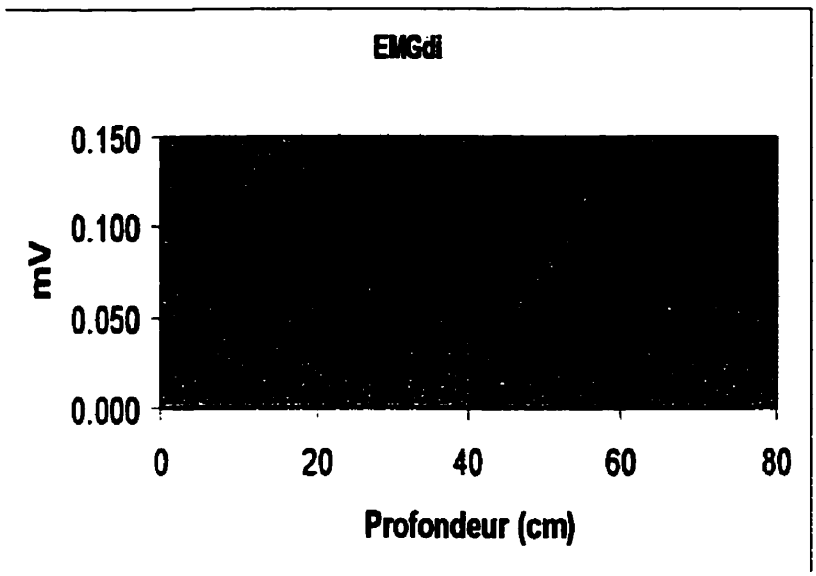

Figure 5.8: Simulations EMGdi en fonction de la profondeur

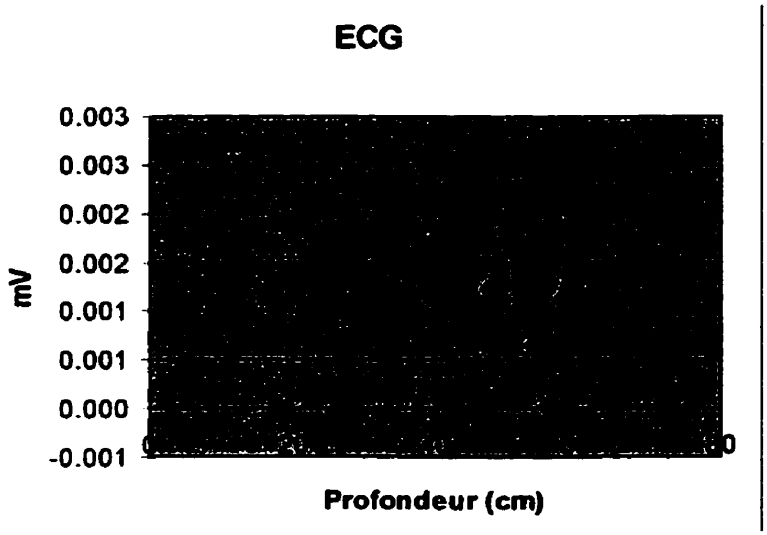

Figure 5.9: Simulations ECG en fonction de la profondeur

Les résultats qui viennent d'être présentés ont été simulés avec une électrode bipolaire modélisée avec FDTD. On peut observer que les courbes des lectures simulées sont loin

de ressembler à celles des courbes expérimentales. Ces courbes ont été obtenues à partir de simulations de 1000 itérations.

Chapitre 6

6. Conclusion

Le travail qui a été réalisé depuis les deux dernières années peut se diviser en trois grandes parties. D'abord, du côté de l'instrumentation, il y a les aspects qui traitent des capteurs de pression et ceux d'une électrode prototype. Le dernier aspect de cette recherche concerne la mise au point d'un logiciel de mesure de propagation des signaux électromagnétiques, FDTD. Nous avons mis sur pied les bases nécessaires permettant l'utilisation de l'outil FDTD qui permet d'effectuer des simulations dans un milieu biologique à de très basses fréquences. Dans ce qui suit, nous soulignons les principales conclusions de ces trois activités complémentaires.

6.1 Capteurs de pression

Plusieurs capteurs de pression ont été considérés afin de remplacer les capteurs de pression en latex. Des comparaisons entre les capteurs de pression à fibre optique, les ballons en latex et les capteurs de pression piézo-électriques ont été faites. L'application en question ne nécessite pas une précision extrême ni une réponse en fréquence très élevée. Dans ces conditions, les capteurs de pression piézo-électriques représentent le meilleur choix. En effet, ils sont très peu coûteux et très facile à fabriquer. De plus, notre expérience nous a démontré qu'une mince couche protectrice de silastic à sa surface

n'affecte aucunement les précisions des lectures. Cependant, il faut faire preuve d'une très grande agileté afin de fabriquer les micro-plaquettes sur lesquelles sont montés les capteurs de pression. Près d'une dizaine de capteurs de pression ont été fabriqués mais seulement deux ont fonctionné correctement. Les *fils d'interconnexion* reliant les plots de contact du capteur de pression à la micro-plaquette sont extrêmement fragiles. Il serait donc plus prudent de faire monter les capteurs de pression sur des substrats plus résistants et avec une colle protectrice. Il existe plusieurs compagnies qui ont la capacité d'effectuer de tels travaux miniaturisés.

6.2 L'électrode prototype

Nous avons tenté de concentrer nos efforts en vue de filtrer l'ECG à la source, c'est-à-dire à même l'électrode. Nous avons voulu obtenir un signal exempt d'ECG dès la lecture. L'électrode prototype a été proposée dans cette optique. Cette proposition est le fruit de plusieurs années d'expérience avec une électrode bipolaire. Ces expériences montrent que la position de l'électrode bipolaire dans l'œsophage a un impact sur le ratio de puissance EMG_{di} / ECG. On croit qu'un meilleur contact autour du sphincter réduit l'apport de l'ECG et augmente l'EMG_{di}.

Suite à cette proposition, une électrode prototype a été fabriquée afin de vérifier expérimentalement l'effet d'une telle électrode. Une seule expérience a été effectuée. Nous avons constaté que le ratio EMG_{di} / ECG avait augmenté de façon apparente. Plusieurs autres expériences cliniques auraient été nécessaires afin de bien valider et

justifier l'utilisation d'une telle électrode dans le but d'augmenter le ratio EMG_{di} / ECG. Cependant, plusieurs problèmes ont été rencontrés lors de la fabrication de l'électrode prototype. Le problème principal était le temps de durcissement du silastic. L'électrode expérimentale est une solution prometteuse au problème de la contamination de l'ECG mais sa validation est difficile. Nous avons donc proposé une méthode de validation effectuée à partir de simulations effectuées par ordinateur.

6.3 Simulations FDTD

Cette méthode a été proposée pour simuler l'impact d'une électrode quelconque quant à sa capacité de filtrer l'ECG. Le but est de construire un simulateur dans lequel on peut construire un modèle tridimensionnel du corps humain et dans lequel on y insérerait une électrode simulée. Le simulateur nous permettrait de modifier la géométrie du patient et de l'électrode dans le but de vérifier l'impact des multiples paramètres de l'électrode. Le but est de trouver les paramètres physiques qui rendent l'électrode prototype plus sensibles à l'EMG_{di} et moins sensible à l'ECG.

La méthode FDTD consiste à échantillonner l'espace à des intervalles réguliers selon les trois axes de l'espace. Le calcul des champs électriques et magnétiques s'effectue plus rapidement pour ces échantillons que pour des éléments finis. Il y a toutefois des règles à respecter afin de garantir la stabilité et l'erreur à chaque échantillon.

La très petite taille des cellules (0.6mm) exige que l'incrément de temps entre deux itérations soit très petit (0.47 μs). Avec un tel incrément de temps, le logiciel FDTD prend près de 214 580 itérations pour simuler une seule longueur d'onde à 10 Hz. Considérant qu'il faut environ 4 à 6 secondes pour simuler une unité de temps, il faudra plusieurs journées pour simuler une période complète. De plus, il faut simuler entre 4 à 6 périodes au minimum afin de permettre au simulateur d'entrer en régime permanent. Toutes ces conditions rendent plus difficile l'utilisation de FDTD.

6.3.1 Recommandations

Les résultats obtenus ne correspondent pas exactement aux attentes fixées au début de ce mémoire. Parmi les causes possibles d'erreur, on retrouve l'exactitude du modèle biologique utilisé. Une modélisation plus détaillée des organes et des tissus internes du thorax et une diminution de la taille des cellules augmenterait la précision et la validité du modèle informatique. Il existe un disque compact nommé *Labelled Visible Human* qui contient les images des plans de coupe d'un humain. La distance entre deux plans est de 1mm. Un modèle du thorax pourrait éventuellement être construit à partir de ces images. De plus, la taille des cellules du simulateur pourrait être diminué de 6mm à 1mm ou plus petit encore. Il est évident que de tels changements augmenteraient le temps d'exécution..

La permittivité des matériaux qui ont été utilisés dans les simulations sont modélisés comme étant linéaires en fréquence. Cela veut dire que leurs paramètres ne varient pas en fonction de la fréquence. Or, il en est autrement dans la nature. En effet, les matériaux

biologiques n'ont pas les mêmes caractéristiques à toutes les fréquences. Malgré tout, la variation de ces paramètres a été négligée. La modélisation des matériaux complexes pourrait ajouter de la précision aux simulations. Les matériaux complexes sont des matériaux dont les propriétés électriques changent de façon non-linéaire en fonction de la fréquence [29].

Une version presque complète du logiciel FDTD était disponible lorsque ce projet fut mis sur pied. Étant donné la disponibilité du logiciel, nous avons tenté de modifier le logiciel afin de mener des simulations à très basse fréquence. Nous savons très bien que la méthode FDTD est principalement utilisée pour des simulations à très hautes fréquences.

Quant au simulateur, nous avons utilisé des conditions de frontières absorbantes. Les conditions de frontières qui ont été utilisées dans notre simulateur ont été développées par Mur [18]. Ces conditions de frontières ne sont pas les meilleures pour absorber des signaux à faible fréquence. Il serait donc nécessaire de vérifier l'efficacité de ces conditions de frontières à de telles fréquences et d'en développer d'autres qui seraient plus performantes, telles les PML (Perfectly Matched Layer).

Lors de la validation du simulateur, nous avons constaté que la simulation du thorax avec une électrode bipolaire n'a pas donné les résultats escomptés. Le manque de ressemblance entre les valeurs des signaux EMG_{di} et ECG simulées et les lectures expérimentales serait vraisemblablement associé à l'utilisation du logiciel FDTD ainsi

qu'à la modélisation arbitraire des sources cardiaques et diaphragmatiques. Il est vrai que la taille des cellules dans le simulateur n'est pas très petite. Cependant, ce premier modèle thoracique, aussi imparfait soit-il, aurait dû donner des résultats suffisamment près des résultats obtenus en clinique. La divergence des résultats nous porte donc à croire que l'utilisation de la méthode FDTD n'est pas appropriée pour effectuer les simulations à de telles fréquences. À la lumière de ces explications, il serait préférable de considérer d'autres méthodes d'analyse afin d'étudier le comportement des électrodes œsophagiennes. Considérant les efforts et le temps requis pour développer un système de simulation informatique, il est important de s'assurer que les équations que nous voulons résoudre pourront être calculées dans un temps raisonnable, ce qui n'était pas nécessairement le cas avec FDTD. De plus, il faut savoir que le temps de développement d'un logiciel informatique peut devenir très imposant. La résolution de l'équation de Laplace qui s'applique à un milieu purement résistif serait plus propice dans un environnement comme le thorax. Les ressources informatiques requises pour résoudre l'équation de Laplace sont beaucoup moins importantes que celle requises par la méthode FDTD.

Enfin, il est important de souligner que le nombre d'heures de travail pour la programmation du simulateur FDTD surpasse le nombre d'heures qu'a exigé la fabrication d'une électrode prototype. De plus, cette électrode a donné des résultats intéressants puisque l'ECG lu par cette électrode avait diminué de moitié environ par rapport aux lectures effectuées avec l'électrode bipolaire. À ce stade de notre recherche,

la fabrication de prototypes d'électrodes semblerait intéressant comme approche plutôt que de procéder par simulation informatique. Il importe donc au futur chercheur d'améliorer le procédé de fabrication de prototypes d'électrodes, plus particulièrement en ce qui a attrait au temps exagéré de durcissement du silastic.

Bibliographie

[1] HEMING, D. G., KO, W. H., NEUMAN, M.R., Indwelling and implantable pressure transducers, CRC Press, USA, 1975, 211p.
Cote : R857T7I52

[2] NEUMAN, Michael R., HEMING, David, CHEUNG, Peter W., KO, Wen H., Physical sensors for biomedical applications, CRC Press, 1977, 160p.
Cote : R857T7P49

[3] ZHILIN, V. G., Optical-Fiber velocity and pressure transducers, New York, 1990, 168p.Cote : TA1800Z4713

[4] SYDENHAM, Peter, Transducers in measurements and control, 3rd ed., England, 1985, 112p. Cote : TJ223T7S92

[5] GAMBARDELLA, G., ZACCONE, C., CARDIA, E., TOMASELLO, F., Intracranial pressure monitoring in children : comparison of external ventricular device with the fiberoptic system, Child's Nervous System, vol. 9, pp. 470-473, 1993

[6] GARDNER, Reed M., Accuracy and reliability of disposable pressure transducers coupled with modern pressure monitors, Critical Care Medicine, vol. 24, pp. 879-882, Mai 1996

[7] WOLTHUIS, Roger, MITCHELL, Gordon, HARTL, James, SAASKI, Eltric, Development of a Dual Function Sensor System for Measuring Pressure and Temperature at the Tip of a Single Optical Fiber, IEEE transactions on biomedical engineering, vol. 40, no. 3, pp. 298-302, mars 1993

[8] NARENDRAN, Nadarajah, CORBO, Mark A., SMITH, William, Fiber Optic Pressure Sensor for Biomedical Applications, ASAIO Journal, Vol. 42, pp.M500-M505, Sept-Oct 1996

[9] GOODYER, Paul, FOTHERGILL, John C., JONES, N.B., HANNING, Christopher D., The Design of an Optical Fiber Pressure Transducer for Use in the Upper Airways, IEEE transactions on biomedical engineering, vol. 43, no. 6, pp. 600-605, juin 1996

[10] SONG, James T., ROZANSKI, Thomas A., BELVILLE, William D., Stress leak pressure : a simple and reproducible method utilizing a fiberoptic microtransducer, Urology, vol. 46, no. 1, pp. 81-84, 1995

[11] COBBOLD, Richard S.C., Transducers for biomedical measurements : Principles and Applications, 1974, John Wiley & Sons, 486 p.

[12] GEDDES, L. A., Principles of Applied Biomedical Instrumentation, 1989, John Wiley & Sons, 961p

[13] ÖNAL, E., LOPATA, M., ALAN,G., O'CONNOR, T., Diaphragmatic EMG and Transdiaphragmatic Pressure Measurements with a Single Catheter, Nov. 1981, Am Rev Respir Dis, Vol. 124, No. 5, pp. 563-565.

[14] YEE, K., Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media, May. 1966, IEEE tr. Ant. & Prop., Vol. AP-14, No. 3, pp 302-307

[15] ENGQUIST, B., MAJDA, A., Absorbing Boundary Conditions for the Numerical Simulation of Waves, July 1977, Mathematics of Computation, Vol. 31, No. 139, pp. 629-651

[16] GEDDES, L.A., BAKER, L.E., The Specific Resistance of Biological Material – A Compendium of Data for the Biomedical Engineer and Physiologist, 1967, Med. & Biol. Engng., Vol. 5, pp. 271-293

[17] LUEBBERS, Raymond, Finite Differences Time Domain Method for Electromagnetics, CRC Press, 1993

[18] MUR, Gerrit, Absorbing Boundary Conditions for the Finite-Difference Approximation of the Time-Domain Electromagnetic-Field Equations, Nov. 1981, IEEE tr. Electromag. Compatibility, Vol. EMC-23, No. 4, pp. 377-382

[19] REDFERN, M.S., HUGHES, R. E. CHAFFIN, D.B., High-pass filtering to remove electrocardiographic interferences from torso EMG recordings, 1993, Clin. Biomech., Vol. 8, pp. 44-48

[20] BARTOLO, A., ROBERTS, C., DZWONCZYK, R.R., GOLDMAN, E., Analysis of diaphragm EMG signals : comparison of gating vs. Subtraction for removal of ECG contamination, 1996, J. Appl, Physiol, Vol. 80, No. 6, pp. 1898-1902

[21] LEVINE, S., GILLEN, J., WEISER, P., GILLEN, M., KWATNY, E., Description and validation of an ECG removal procedure for EMGdi power spectrum analysis, 1986, J. Appl. Physiol., Vol. 60, No. 3, pp. 1073-1081

[22] Silicon Microstructures, http://www.exar.com

[23] Numerical Recipies, http://www.nr.com

[24] SAVARD, Pierre, Physiologie Cardio-Vasculaire – Du potentiel d'action à l'électrocardiogramme, Institut de génie biomédical

[25] Nusil, http://www.nusil.com

[26] TRA-CON silicones, http://www.tra-con.com

[27] Cooner Wire, http://www.coonerwire.com

[28] Microdyne Inc., Ohio, 1-800-872-3963

[29] MROZOWSKI, Michal, STUCHLY, Maria, Parameterization of Media Dispersive Properties for FDTD, IEEE tr. Ant & Prop, Vol. 45, No. 9, Sep 1997, pp. 1438-1439

[30] DE MOERLOOSE, Jan, DAWSON, Trevor, STUCHLY, Maria, Application of the finite difference time domain algorithm to quasi-static field analysis, Radio Science, Vol. 32, No. 2, March-April 1997, pp329-341

[31] FURSE, C.M., GANDHI, O.P., Calculation of Electric Fields and Currents Induces in a Millimeter-Resolution Human Model at 60Hz Using the FDTD Method, Bioelectromagnetics, 1998, Vol. 19, No. 5, pp. 293-299

[32] NETTER, Frank H., Atlas of Human Anatomy, Novartis, 1989, p. 224

[33] Labeled Visible Human, CD-ROM, Research Systems, Boulder, CO.

[34] CRC Handbook of biological effects of electromagnetic fields, POLK, C., POSTOW, E., eds, CRC Press Inc, Boca Raton, p. 88, 1986

[35] KRAUS, J.D., Electromagnetics, McGraw Hill Inc, New York, p.552, 1992

[36] SAVARD, P., ACKAOUI, A., GULRAJANI, RM., NADEAU, RA., ROBERGE, RA, GUARDO, R., Localization of cardiac ectopic activity by a single moving

dipole in man. Comparison of different computation techniques, J of Electrocardiol, 18:211-222, 1985

[37] SHAHIDI, V., SAVARD, P., NADEAU, RA., The forward and inverse problems of electrocardiography: modeling and recovery of epicardial potentials in humans, IEEE Trans on BME, 41:249-256, 1994a

[38] SHAHIDI, V., SAVARD, P., Finite element modeling of the human torso, Med and Biol Eng and Comput, 32:S25-33, 1994b

Annexe A

Code source en C++ du logiciel FDTD

Tabulation : .27
« courier new.
main.cpp

```
1   ////////////////////////////////////////////////////////////////////////////
2   //                                                                        //
3   //                  MAIN.CPP ( 18.09.97)                                  //
4   //                                                                        //
5   //                                                                        //
6   ////////////////////////////////////////////////////////////////////////////
7
8   //########################################################################//
9   #include <stdio.h>
10  #include <stdlib.h>
11  #include <ctype.h>
12  #include <math.h>
13  #include <time.h>
14  #include <malloc.h>
15  #include <string.h>
16  #include "ordi.h"                           // Verification de l'emplacement de compilation
17  #ifdef _HOME
18      #include <io.h>                         // Non necessaire a Poly
19      #include <conio.h>                      // Non compatible sur UNIX!!!
20  #endif
21  //########################################################################//
22  #include "constantes.h"                     // Definition des constantes
23  #include "Type.h"                           // Module de definition des types simples
24  #include "elem3D.h"                         // Definition de la classe elem3D
25  #include "TABLEAUX.H"                       // Definition de la classe tableaux ( tableau 3D
implante ds un tableau 1D)
26  #include "ESPACES3.H"                       // Classe de description d'un espace 3D de cellules
27  #include "TYPE2.H"                          // Instanciation des classes elem3D et espaces
28  #include "VAR_ENV.H"                  // Variables d'environnement
29  #include "var_env_comp.h"             // Constantes de calcul derivant des var d'environnement
30  #include "basic_file_io.h"            // Module d'E/S sur disque -- Luc Romain
31  #include "material.h"                       // Classe de description des materiaux
32  #include "material_computing.h"       // Description des materiaux augmentee des constantes de
calcul
33  #include "orbc_comp_const.h" // Constantes de calcul des orbc (classe)
34  #include "material_table.h"           // Module d'initialisation de la table des materiaux
35  #include "ei_fields.0.h"              // Module de calcul des champs electriques incidents version
test
36  #include "Es_field.h"                       // Module de calcul des champs electriques diffuses
37  #include "hs_fields.h"                      // Module de calcul des champs magnetiques diffuses
38
39  #include "Orbc_xy.h"                  // Module de calcul des orbc version 1
40  #include "Orbc_xz.h"                  // Module de calcul des orbc version 1
41  #include "Orbc_yx.h"                  // Module de calcul des orbc version 1
42  #include "Orbc_yz.h"                  // Module de calcul des orbc version 1
43  #include "Orbc_zx.h"                  // Module de calcul des orbc version 1
44  #include "Orbc_zy.h"                  // Module de calcul des orbc version 1
45  #include "build.h"                          // Construction de l'objet : sphere pour tests.
46  /*
47                                  << Luc Romain, 15 juin 1998 >>
48  */
49  #include "LChain.h"
50  #include "save_data.h"                      // Module de sauvegarde des champs (plans)
51  #include "Electrode.h"                      // Module de fabrication de l'electrode
52  #include "dipole.h"                   // Module dipole
53  #include "coeur.h"                          // Module de l'objet coeur
```

```
54   #include "update.h"                          // Module d'affichage de progres
55   #include "save_space.h"                      // Module de sauvegarde en format MATLAB
56   #include "restore.h"                  // Module de CRASH RECOVERY
57   #include "diaphragme2.h"
58   //#include "wiretest.h"
59   //#########################################################################//
60
61   void main(void)
62   {
63       //------------------------ Minuterie --------------------------
64       time_t   start, finish,stepl;
65       time( &start );
66       //------------------------ Variables --------------------------
67       // Valeurs par defaut.
68       char plan;                           // plan de sauvegarde
69       char org_slct;                   // Selection de l'organe de polarisation
70       typ_material  cmat,dmat; // Materiaux coeur et diaphragme
71       char Quit;
72       char repertoire[20];      // Repertoire de sauvegarde
73       char inirep[20];
74       int alt;                         // altitude du plan de sauvegarde
75       int GO;                                  // pas initial
76
77       int                                      Step=1,st;
78       int                                      stmx,stmn;
                        // temps min et max de sauvegarde
79       // Dimensions de l'espace
80       int                                      Dimx=70,Dimy=40,Dimz=90;
81   //  int                                      Dimx=20,Dimy=20,Dimz=20;
82       typ_distance             CellSize=0.006;                                    // Taille
d'une cellule
83       typ_coeff                        WaveAmplitude = 0.0;
         // Amplitude de l'onde
84   //  typ_coeff                        WaveAmplitude = 1000.0;
         // Amplitude de l'onde
85       typ_coeff                        time_swap;
86       bool                             newsim;
87
88       work_dir(inirep,repertoire,&newsim,"dir.ini");
89
90
91       if(!newsim)
92       {
93               printf ("------- Restoring stored simulation variables ");
94               newsim=restore_variables(&Dimx,&Dimy,&Dimz,&CellSize,
95
&WaveAmplitude,&GO,&time_swap,inirep,"var.ini");
96       }
97
98
99       var_env                          BaseEnv(CellSize,CellSize,CellSize);      // Environnement
associe
100      var_env_comp             CompEnv(BaseEnv,WaveAmplitude);                   // Const de calcul
d'environnement
101
102
103      cellule_champs           CellNoField      (0.0,0.0,0.0);
104      cellule_physique CellFreeSpace(0,0,0);
105
```

```
106     material_computing        MaterialTable[11];
107
108     espace_champs             ElcFields(0,0,0,Dimx,Dimy,Dimz);          // Espace des champs
electriques
109     espace_champs             MagFields(0,0,0,Dimx,Dimy,Dimz);          // Espace des champs
magnetiques
110     espace_physique           PhysSpace(0,0,0,Dimx,Dimy,Dimz);          // Espace physique (
materiaux)
111
112     tab_for_orbc      Exy1(Dimx,4,Dimz),Exy2(Dimx,4,Dimz);
113     tab_for_orbc      Exz1(Dimx,Dimy,4),Exz2(Dimx,Dimy,4);
114     tab_for_orbc      Eyx1(4,Dimy,Dimz),Eyx2(4,Dimy,Dimz);
115     tab_for_orbc      Eyz1(Dimx,Dimy,4),Eyz2(Dimx,Dimy,4);
116     tab_for_orbc      Ezx1(4,Dimy,Dimz),Ezx2(4,Dimy,Dimz);
117     tab_for_orbc      Ezy1(Dimx,4,Dimz),Ezy2(Dimx,4,Dimz);
118
119     // Coefficients pour le calcul des champs aux frontieres
120     orbc_const                CoefOrbcX(BaseEnv,'X');
121     orbc_const                CoefOrbcY(BaseEnv,'Y');
122     orbc_const                CoefOrbcZ(BaseEnv,'Z');
123
124     BasicFileIO Fe1(repertoire,"ef1.m","wb"); // write mode
125     BasicFileIO Fe2(repertoire,"ef2.m","wb");
126     BasicFileIO Source(repertoire,"source.m","wb");
127
128
129
130
131
132
133
134     //------------------- Fin declaration variables ------------------
135
136
137     //-------------------   Initialisations objets  ------------------
138
139     printf("------- Organe de polarisation: (C)oeur, (D)ia ou (T)ous? ");
140             org_slct=toupper(getchar());
141
142     switch (org_slct)
143     {
144     case 'D':
145             dmat=2; // Diaphragme = antenne
146             cmat=8; // Coeur = passif
147             break;
148     case 'C':
149             dmat=9; // Diaphragme = passif
150             cmat=2; // Coeur = antenne
151             break;
152     case 'T':
153             dmat=2;
154             cmat=2;
155             break;
156     }
157
158     printf ("################## Initialisations ################ ");
159     printf ("------- Table des materiaux ");
160
161     set_material_table_file(MaterialTable,BaseEnv);
```

```
162
163     if(newsim)
164     {
165             GO=1;           // Premier pas de simulation (Step)
166             printf ("------- Espaces des champs ");
167             ElcFields.init(CellNoField);
168             MagFields.init(CellNoField);
169             printf ("------- Espace physique (materiaux)  ");
170             PhysSpace.init(CellFreeSpace);
171             printf ("------- Tableaux pour ORBC  ");
172             Exy1.init(CHAMP_NUL);     Exy2.init(CHAMP_NUL);
173             Exz1.init(CHAMP_NUL);     Exz2.init(CHAMP_NUL);
174             Eyx1.init(CHAMP_NUL);     Eyx2.init(CHAMP_NUL);
175             Eyz1.init(CHAMP_NUL);     Eyz2.init(CHAMP_NUL);
176             Ezx1.init(CHAMP_NUL);     Ezx2.init(CHAMP_NUL);
177             Ezy1.init(CHAMP_NUL);     Ezy2.init(CHAMP_NUL);
178             printf ("------- Temps  ");
179             CompEnv.time=0.0;
180     }
181     else
182     {
183             Fe1.set_mode("a+b");      // Append mode.
184             Fe2.set_mode("a+b");
185             Source.set_mode("a+b");
186
187             printf ("------- Espaces des champs     [LOADING]");
188             restore_field(&ElcFields,inirep,"efld.ini");
189             restore_field(&MagFields,inirep,"mfld.ini");
190             printf ("------- Espace physique        [LOADING]  ");
191             restore_space(&PhysSpace,inirep,"phsp.ini");
192
193             printf ("------- Tableaux pour ORBC     [LOADING]  ");
194
195             restore_orbc(&Exy1,inirep,"exy1.ini");
196             restore_orbc(&Exy2,inirep,"exy2.ini");
197             restore_orbc(&Exz1,inirep,"exz1.ini");
198             restore_orbc(&Exz2,inirep,"exz2.ini");
199             restore_orbc(&Eyx1,inirep,"eyx1.ini");
200             restore_orbc(&Eyx2,inirep,"eyx2.ini");
201             restore_orbc(&Eyz1,inirep,"eyz1.ini");
202             restore_orbc(&Eyz2,inirep,"eyz2.ini");
203             restore_orbc(&Ezx1,inirep,"ezx1.ini");
204             restore_orbc(&Ezx2,inirep,"ezx2.ini");
205             restore_orbc(&Ezy1,inirep,"ezy1.ini");
206             restore_orbc(&Ezy2,inirep,"ezy2.ini");
207
208             printf ("------- Temps                  [LOADING]");
209             CompEnv.time=time_swap;  // restauration du temps.
210
211     }
212
213     printf ("------- Parametres MATLAB ");
214     restore_plane_parameters(&plan,&alt,&stmn,&stmx,&st,inirep,"planp.ini");
215     printf(" %c[%d] %d --> %d",toupper(plan),alt,stmn,stmx);
216     printf ("------- Total execution steps --> %d",st);
217
218     CompEnv.maj_delay(Dimx, Dimy, Dimz);
219     // vrai modele
220
```

```
221     printf ("------- Construction de l'espace");
222     // La ligne suivante peut être un problème... 30aout1999 LR
223     cube(1,1,1,69,39,89,5,PhysSpace);
224     printf ("------- Construction du modèle  ");
225     printf ("------- Thorax");
226     buildellipse(21,8,35,20,5,78,3,PhysSpace);// Thorax
227     ext_ellipse(4,21,8,35,20,5,78,6,PhysSpace); // Exterieur
228     //printf ("------- Oesophage");
229     //buildcylindre(1,35,20,50,77,9,PhysSpace);        // Oesophage
230     //buildcylindre(1,35,20,50,77,10,PhysSpace); // Eau dans oesophage
231
232     diaphragme2 diaphragme(35,20,55,dmat,&ElcFields,&PhysSpace,CompEnv);
233     Coeur coeur(&ElcFields,&PhysSpace,CompEnv,45,20,60,3,cmat);
234
235
236     // Construction par fichier : elec_specs.ini
237     Electrode e1(&ElcFields,&PhysSpace);
238     /*
239     Electrode e1(35,20,46,2,2,1,1,4,&ElcFields,&PhysSpace);
240     Electrode e2(35,20,48,2,2,1,1,4,&ElcFields,&PhysSpace);
241     Electrode e3(35,20,50,2,2,1,1,4,&ElcFields,&PhysSpace);
242     Electrode e4(35,20,52,2,2,1,1,4,&ElcFields,&PhysSpace);
243     Electrode e5(35,20,54,2,2,1,1,4,&ElcFields,&PhysSpace);
244     Electrode e6(35,20,56,2,2,1,1,4,&ElcFields,&PhysSpace);
245     Electrode e7(35,20,58,2,2,1,1,4,&ElcFields,&PhysSpace);
246     */
247  // Electrode e1(35,20,60,1,10,1,0,4,&ElcFields,&PhysSpace);
248  /*
249     cube(1,1,1,19,19,19,0,PhysSpace);
250     Coeur coeur(&ElcFields,&PhysSpace,CompEnv,12,12,12,1,2);
251     WireTest wiretest(6,9,6,8,1,3,&ElcFields,&PhysSpace);
252  // buildellipse(5,2,10,10,6,14,3,PhysSpace);
253  // Electrode e1(5,5,10,3,3,2,1,4,&ElcFields,&PhysSpace);
254    */
255     // ---------- Construction de l'electrode ---------------
256
257
258     printf ("################ Initialisations OK ############### ");
259
260
261     printf ("################ Calcul des champs ############### ");
262
263  // ------------------------------------------------------------------------
264  // -                                                                      -
265  // -************************** Debut des calculs ***************************-
266  // -                                                                      -
267  // ------------------------------------------------------------------------
268
269     time(&step1);
270     Quit=32;
271     Step=GO;
272     //for (Step=GO; Step<=st;Step++)
273     while((Step<=st)&&(Quit!='Q'))
274     {
275
276  #ifdef _HOME
277             if(_kbhit())
```

```
278             {
279                     Quit=toupper(_getch());
280                     if(Quit=='F')
281                     {
282                             if(_flushall())
283                                     printf("------- All Buffers Flushed.");
284                             else
285                                     printf("------- Error: Flushall() failed!");
286                     }
287             }
288 #endif
289
290
291             aff_progres(&step1,GO,Step,st);
292
293             coeur.polarise(CompEnv.time);
294             diaphragme.polarise(CompEnv.time);
295
296             // -----------      Champs electriques    -----------
297             electric_scattered_fields
298                     (ElcFields,MagFields,PhysSpace,MaterialTable,CompEnv);
299
300
301             // -----------             Orbc            -----------
302             orbc_yx(ElcFields,Eyx1,Eyx2,CoefOrbcX);
303             orbc_zx(ElcFields,Ezx1,Ezx2,CoefOrbcX);
304             orbc_zy(ElcFields,Ezy1,Ezy2,CoefOrbcY);
305             orbc_xy(ElcFields,Exy1,Exy2,CoefOrbcY);
306             orbc_xz(ElcFields,Exz1,Exz2,CoefOrbcZ);
307             orbc_yz(ElcFields,Eyz1,Eyz2,CoefOrbcZ);
308
309             // Increment d'un demi pas de temps
310             CompEnv.time= CompEnv.time + 0.5 * CompEnv.dt;
311
312
313             //-----------      Champs magnetiques    -----------
314             magnetic_scattered_fields
315                     (ElcFields, MagFields, PhysSpace,  MaterialTable[0]);
316
317
318     //      printf("/n -----------     Sauvegarde des champs   -----------     ");
319
320             if((Step>=stmn)&&(Step<=stmx))
321                     if(Step%200==0)  // Sauvegarde a chaque X pas.
322                     {
323                             if(!save_plane(repertoire,plan,alt,Step,ElcFields,MagFields))
324                                     printf("Erreur de sauvegarde du plan...");
325                     }
326
327             if(Step==10)
328             {
329                     if(!save_plane(repertoire,plan,alt,Step,ElcFields,MagFields))
330                                     printf("Erreur de sauvegarde du plan...");
331             }
332
333
334
335
336     //      printf("--------- Sauvegarde de l'electrode -------------");
```

```
337       /*
338             Fe1.write_to_file(e1.delta_e()*CompEnv.dz);
339             Fe1.write_to_file(e2.delta_e()*CompEnv.dz);
340             Fe1.write_to_file(e3.delta_e()*CompEnv.dz);
341             Fe1.write_to_file(e4.delta_e()*CompEnv.dz);
342             Fe1.write_to_file(e5.delta_e()*CompEnv.dz);
343             Fe1.write_to_file(e6.delta_e()*CompEnv.dz);
344             Fe1.write_to_file(e7.delta_e()*CompEnv.dz);
345             Fe1.write_to_file("");
346
347             Fe2.write_to_file(e1.delta_eb()*CompEnv.dz);
348             Fe2.write_to_file(e2.delta_eb()*CompEnv.dz);
349             Fe2.write_to_file(e3.delta_eb()*CompEnv.dz);
350             Fe2.write_to_file(e4.delta_eb()*CompEnv.dz);
351             Fe2.write_to_file(e5.delta_eb()*CompEnv.dz);
352             Fe2.write_to_file(e6.delta_eb()*CompEnv.dz);
353             Fe2.write_to_file(e7.delta_eb()*CompEnv.dz);
354             Fe2.write_to_file("");
355             */
356     Source.writexyz_to_file(ElcFields.get(40,15,65).cx(),
357                     ElcFields.get(40,15,65).cy(),ElcFields.get(40,15,65).cz());
358
359
360
361             Fe1.write_to_file(e1.delta_e()*CompEnv.dz);
362             Fe1.write_to_file(e1.delta_eb()*CompEnv.dz);
363             Fe1.write_to_file("");
364
365  /*
366             Fe1.write_to_file(wiretest.deltav1()*CompEnv.dy);
367             Fe1.write_to_file(wiretest.deltav2()*CompEnv.dy);
368             //Fe1.write_to_file(wiretest.delwir1()*CompEnv.dx);
369             //Fe1.write_to_file(wiretest.delwir2()*CompEnv.dx);
370             Fe1.write_to_file("");
371
372             Source.writexyz_to_file(ElcFields.get(10,12,12).cx(),
373                     ElcFields.get(10,12,12).cy(),ElcFields.get(10,12,12).cz());
374    */
375             //printf("_elec = %4.4e ,%4.4e",e1.delta_e(),e1.delta_eb());
376
377
378
379             // ------- Increment d'un demi pas de temps -------
380             CompEnv.time= CompEnv.time + 0.5 * CompEnv.dt;
381
382             Step++;
383     }
384
385     time( &finish );
386     if(Quit=='Q')
387             printf("------- Simulation terminated by user");
388
389     Fe1.close_file();
390     Fe2.close_file();
391     Source.close_file();
392
393  /*
394        ----------------------------------------------------
395                                                    CRASH RECOVERY
```

```
396
397         ------------------------------------------------
398 */
399
400     // Sauvegarde du champ electrique.
401
402     printf("---------- Saving Parameters ----------");
403     printf("----- Saving Electric Fields ...");
404     save_field(&ElcFields,inirep,"efld.ini");
405     printf("----- Saving Magnetic Fields ...");
406     save_field(&MagFields,inirep,"mfld.ini");
407     printf("----- Saving Physical space...");
408     save_space(&PhysSpace,inirep,"phsp.ini");
409     printf("----- ORBC -");
410     save_orbc(&Exy1,inirep,"exy1.ini"); printf(" 1");
411     save_orbc(&Exy2,inirep,"exy2.ini");printf(" 2");
412     save_orbc(&Exz1,inirep,"exz1.ini");printf(" 3");
413     save_orbc(&Exz2,inirep,"exz2.ini");printf(" 4");
414     save_orbc(&Eyx1,inirep,"eyx1.ini");printf(" 5");
415     save_orbc(&Eyx2,inirep,"eyx2.ini");printf(" 6");
416     save_orbc(&Eyz1,inirep,"eyz1.ini");printf(" 7");
417     save_orbc(&Eyz2,inirep,"eyz2.ini");printf(" 8");
418     save_orbc(&Ezx1,inirep,"ezx1.ini");printf(" 9");
419     save_orbc(&Ezx2,inirep,"ezx2.ini");printf(" 10");
420     save_orbc(&Ezy1,inirep,"ezy1.ini");printf(" 11");
421     save_orbc(&Ezy2,inirep,"ezy2.ini");printf(" 12");
422     printf("----- Saving MATLAB parameters...");
423     save_plane_parameters(plan,alt,stmn,stmx,st,inirep,"planp.ini");
424     printf("----- Saving simulation parameters...");
425     save_sim_parameters(Dimx,Dimy,Dimz,CellSize,WaveAmplitude,Step,CompEnv.time,
426                         inirep,"var.ini");
427     printf("**************CORE SAVED***************");
428
429
430     // Fin de la procedure de CRASH RECOVERY;
431
432     printf ("################     RESULTATS     ############### ");
433
434     printf ("=============================================");
435     printf ("Temps apres %d iterations : %e",Step,CompEnv.time);
436     printf ("=============================================");
437     printf ("Computations take %6.0f seconds.", difftime(finish,start));
438     printf ("=============================================");
439     printf ("Increment de temps dt: %e",CompEnv.dt);
440     printf ("Vitesse de la lumiere: %e",C);
441     printf ("=============================================");
442     printf ("terminated successfully.");
443
444 };
445
446
```

basic_file_io.h

```
1   ////////////////////////////////////////////////////////////////////////////
2   // BasicFileIO                                                            //
3   //------------------------------------------------------------------------//
4   // Cette classe gere les acces de base au disque.                         //
5   // Elle fait la gestion de l'ouverture ainsi que la fermeture des         //
6   // fichiers. Elle permet egalement d'effectuer des lectures, des          //
7   // ecritures ainsi que des deplacements dans un fichier.                  //
8   //------------------------------------------------------------------------//
9   // Auteur: Luc Romain                                                     //
10  // Date:    15 septembre 1998                                             //
11  // Derniere modification: Aucune                                          //
12  ////////////////////////////////////////////////////////////////////////////
13
14  class BasicFileIO
15  {
16  public:
17
18      // -------- Constructeurs -----------
19      BasicFileIO(char *rep,char *filename,char *Mode)
20      {
21              sprintf(file_name,"%s%s",rep,filename);
22              initOK=true;
23              file_open=false;
24              sprintf(mode,"%s",Mode);
25      };
26
27      bool init(char *rep,char *filename,char *Mode)
28      {
29              sprintf(file_name,"%s%s",rep,filename);
30              file_open=false;
31              initOK=true;
32              sprintf(mode,"%s",Mode);
33              return(true);
34      };
35
36      BasicFileIO()
37      {
38              initOK=false;
39      };
40
41      // ---------- Destructeur -------------
42
43      ~BasicFileIO(void)
44      {
45              if(file_open)
46              {
47                      fclose(file_handle);
48                      file_open=false;
49              }
50      };
51
```

```
52  //-------------------------------------------------------------------------//
53  // write_to_file(...)                                                      //
54  //-------------------------------------------------------------------------//
55  // Cette fonction ouvre le fichier, ecrit *str au fichier puis ferme le    //
56  // fichier.                                                                //
57  //-------------------------------------------------------------------------//
58  // Luc Romain, 29 septembre 1998                                           //
59  //-------------------------------------------------------------------------//
60
61      bool write_to_file(char *str)
62      {
63              if(!initOK)
64                      return(false);
65              else
66              {
67                      if(!file_open)
68                      {
69                              if(!open_file(mode))
70                              {
71                                      initOK=false;
72                                      return(false);
73                              }
74                              file_open=true;
75                      }
76                      fprintf(file_handle,"%s",str);
77                      return(true);
78              }
79      };
80
81  //-------------------------------------------------------------------------//
82  // write_to_file(...)                                                      //
83  //-------------------------------------------------------------------------//
84  // Cette fonction ouvre le fichier, ecrit la valeur "value" au fichier puis //
85  // ferme le fichier.                                                       //
86  // La fonction ne ferme pas le fichier apres execution... ATTENTION!!!     //
87  //-------------------------------------------------------------------------//
88  // Luc Romain, 29 septembre 1998                                           //
89  //-------------------------------------------------------------------------//
90
91      bool write_to_file(typ_champ value)
92      {
93              if(!initOK)
94                      return(false);
95              else
96              {
97                      if(!file_open)
98                      {
99                              open_file(mode);
100                             file_open=true;
101                     }
102                     fprintf(file_handle," %le",value);
103                     return(true);
104             }
```

```
105     };
106
107  //-------------------------------------------------------------------------//
108  // write_to_file(int)                                                      //
109  //-------------------------------------------------------------------------//
110  // Cette fonction ouvre le fichier, ecrit la valeur "value" au fichier puis //
111  // ferme le fichier.                                                       //
112  // La fonction ne ferme pas le fichier apres execution... ATTENTION!!!     //
113  //-------------------------------------------------------------------------//
114  // Luc Romain, 29 septembre 1998                                           //
115  //-------------------------------------------------------------------------//
116
117     bool write_to_file(int value)
118     {
119             if(!initOK)
120                     return(false);
121             else
122             {
123                     if(!file_open)
124                     {
125                             open_file(mode);
126                             file_open=true;
127                     }
128                     fprintf(file_handle," %d",value);
129                     return(true);
130             }
131     };
132
133  //-------------------------------------------------------------------------//
134  // writexyz_to_file(...)                                                   //
135  //-------------------------------------------------------------------------//
136  // Cette fonction ecrit les composantes x,y et z au fichier                //
137  // La fonction ne ferme pas le fichier apres execution...ATTENTION!!!      //
138  //-------------------------------------------------------------------------//
139  // Luc Romain, 29 septembre 1998                                           //
140  //-------------------------------------------------------------------------//
141
142     bool writexyz_to_file(typ_champ vx,typ_champ vy,typ_champ vz)
143     {
144             if(!initOK)
145                     return(false);
146             else
147             {
148                     if(!file_open)
149                     {
150                             if(!open_file(mode))
151                             {
152                                     initOK=false;
153                                     printf("- BasicFileIO::writexyz_to_file(...);");
154                                     return(false);
155                             }
156                             file_open=true;
157                     }
```

```
158                     fprintf(file_handle," %le %le %le",vx,vy,vz);
159                     return(true);
160             }
161     };
162
163 //-------------------------------------------------------------------------//
164 // create_file(...)                                                        //
165 //-------------------------------------------------------------------------//
166 // Cette fonction cree le fichier et l'ecrase s'il est deja existant.      //
167 // Ensuite, elle ecrit la valeur "value" au fichier puis                   //
168 // ferme le fichier.                                                       //
169 //-------------------------------------------------------------------------//
170 // Luc Romain, 29 septembre 1998                                           //
171 //-------------------------------------------------------------------------//
172
173     bool create_file(char *str)
174     {
175             if(!initOK)
176                     return(false);
177             else
178             {
179                     if(!file_open)
180                     {
181                             if(!open_file(mode))
182                                     return(false);
183                             file_open=true;
184                     }
185                     fprintf(file_handle,"%s",str);
186                     return(true);
187             }
188     };
189
190
191 //-------------------------------------------------------------------------//
192 // read_from_file(...)                                                     //
193 //-------------------------------------------------------------------------//
194 // Cette fonction ouvre le fichier, lit une chaine de caractere puis       //
195 // ferme le fichier.                                                       //
196 //-------------------------------------------------------------------------//
197 // Luc Romain, 29 septembre 1998                                           //
198 //-------------------------------------------------------------------------//
199
200     bool read_from_file(char *buffer)
201     {
202             char format_string[50];
203
204             // Bien s'assurer que la taille de buffer soit suffisante.
205             sprintf(format_string,"%%%is",sizeof(buffer)-1);
206
207             if(!initOK)
208                     return(false);
209
```

```
210             if(open_file(mode))
211             {
212                     fscanf(file_handle,format_string,buffer);
213                     close_file();
214                     return(true);
215             }
216             else
217                     return(false);
218     };
219 //-------------------------------------------------------------------------//
220 // read_from_file(buffer,int n)
                                //
221 //-------------------------------------------------------------------------//
222 // Cette fonction ouvre le fichier, lit n nombres puis
    //
223 // ferme le fichier.
                                //
224 // La fonction retourne le nombre de valeurs lues.
    //
225 //-------------------------------------------------------------------------//
226 // Luc Romain, 18 novembre 1998
                                //
227 //-------------------------------------------------------------------------//
228
229     int read_file(typ_champ *value,int n)
230     {
231             int x=0,fileOK;
232
233             if(!file_open)
234             {
235                     fileOK=open_file(mode);
236                     file_open=true;
237             }
238
239             if(!fileOK)
240             {
241                     printf("Erreur d'ouverture de fichier -- basic_file_io -- read_file");
242                     file_open=false;
243                     getchar();
244                     return(false);
245             }
246
247             while((!end_of_file())&&(x<n))
248             {
249                     fscanf(file_handle,"%le",&value[x]);
250                     x++;
251             }
252
253             close_file();
254             return(x);
255     };
256
257 //-------------------------------------------------------------------------//
258 // read_int()
                                        //
259 //-------------------------------------------------------------------------//
260 // Lecture d'un entier dans le fichier et retour de cet entier.
261 //-------------------------------------------------------------------------//
262 // Luc Romain, 18 novembre 1998
```

```
                                //
263 //--------------------------------------------------------------------------//
264
265 int read_int()
266 {
267     int x,fileOK;
268     if(!file_open)
269     {
270             fileOK=open_file(mode);
271             file_open=true;
272
273             if(!fileOK)
274             {
275                     printf("Erreur d'ouverture de fichier -- read_int()");
276                     file_open=false;
277                     getchar();
278                     return(false);
279             }
280     }
281     fscanf(file_handle,"%d",&x);
282     return(x);
283 };
284
285 //--------------------------------------------------------------------------//
286 // get_line()
                                        //
287 //--------------------------------------------------------------------------//
288 // Lecture d'un entier dans le fichier et retour de cet entier.
289 //--------------------------------------------------------------------------//
290 // Luc Romain, 18 novembre 1998
                                //
291 //--------------------------------------------------------------------------//
292
293 bool get_line(char *buffer,int n)
294 {
295     int fileOK;
296
297     if(!file_open)
298     {
299             fileOK=open_file(mode);
300             file_open=true;
301
302             if(!fileOK)
303             {
304                     printf("Erreur d'ouverture de fichier -- read_int()");
305                     file_open=false;
306                     getchar();
307                     return(false);
308             }
309     }
310
311     if(fgets(buffer,n,file_handle)==NULL)
312     {
313             buffer='0';
314             return(false);
315     }
316     return(true);
317 }
318
```

```
319 //---------------------------------------------------------------------------//
320 // read_char()                                                               //
321 //---------------------------------------------------------------------------//
322 // Lecture d'un entier dans le fichier et retour de cet entier.              //
323 //---------------------------------------------------------------------------//
324 // Luc Romain, 17 decembre 1998                                              //
325 //---------------------------------------------------------------------------//
326
327 char read_char()
328 {
329     int fileOK;
330     char x;
331
332     if(!file_open)
333     {
334             fileOK=open_file(mode);
335
336             if(!fileOK)
337             {
338                     printf("Erreur d'ouverture de fichier -- read_char()");
339                     file_open=false;
340                     getchar();
341                     return(false);
342             }
343     }
344
345     if(!end_of_file())
346     {
347             x=fgetc(file_handle);
348             //fscanf(file_handle,"%ls",&x);
349     }
350     return(x);
351 };
352
353
354 //---------------------------------------------------------------------------//
355 // read_coeff()                                                              //
356 //---------------------------------------------------------------------------//
357 // Lecture d'un entier dans le fichier et retour de cet entier.              //
358 //---------------------------------------------------------------------------//
359 // Luc Romain, 18 novembre 1998                                              //
360 //---------------------------------------------------------------------------//
361
362 typ_coeff read_coeff(void)
363 {
364     typ_coeff x;
365     int fileOK;
366     if(!file_open)
367     {
368             fileOK=open_file(mode);
369             file_open=true;
370             if(!fileOK)
371             {
372                     printf("Erreur d'ouverture de fichier -- read_coeff()");
373                     file_open=false;
```

```
374                     getchar();
375                     return(false);
376             }
377     }
378
379     fscanf(file_handle,"%Lf",&x);
380     return(x);
381 };
382
383
384
385
386
387 private:
388     char file_name[30],mode[4];
389     bool file_open,initOK;
390     FILE *file_handle;
391
392 protected:
393
394 //-------------------------------------------------------------------------//
395 // end_of_file()                                                           //
396 //-------------------------------------------------------------------------//
397 // Cette fonction retourne vrai si EOF est atteint; false sinon.           //
398 // bien effectuee.                                                         //
399 //-------------------------------------------------------------------------//
400 // Luc Romain, 18 novembre 1998                                            //
401 //-------------------------------------------------------------------------//
402     bool end_of_file()
403     {
404             if(feof(file_handle))
405                     return(true);
406             else
407                     return(false);
408     }
409
410
411 //-------------------------------------------------------------------------//
412 // open_file(...)                                                          //
413 //-------------------------------------------------------------------------//
414 // Cette fonction ouvre le fichier, et retourne vrai si l'ouverture s'est  //
415 // bien effectuee.                                                         //
416 //-------------------------------------------------------------------------//
417 // Luc Romain, 29 septembre 1998                                           //
418 //-------------------------------------------------------------------------//
419
420     bool open_file(char *mode)
421     {
422             if(!initOK)
423             {
424                     printf("Erreur d'initialisation -- BasicFileIO::open_file()");
425                     return(false);
426             }
```

```
427
428             file_handle=fopen(file_name,mode);
429
430
431             if(file_handle!=NULL)
432             {
433                     file_open=true;
434             }
435             else
436             {
437                     printf("d'ouverture de fichier -- BasicFIO::Open_file()");
438                     file_open=false;
439             }
440
441             return(file_open);
442     };
443
444 public:
445 //-------------------------------------------------------------------------//
446 // close_file(...)
                                        //
447 //-------------------------------------------------------------------------//
448 // Cette fonction ferme le fichier.
                        //
449 //-------------------------------------------------------------------------//
450 // Luc Romain, 29 septembre 1998
                        //
451 //-------------------------------------------------------------------------//
452
453     void close_file(void)
454     {
455             if(file_open)
456             {
457                     fclose(file_handle);
458                     file_open=false;
459             }
460     };
461
462 //-------------------------------------------------------------------------//
463 // set_mode()
                                                //
464 //-------------------------------------------------------------------------//
465 // Cette fonction fixe le mode du fichier.
                //
466 //-------------------------------------------------------------------------//
467 // Luc Romain, 24 decembre 1998
                                //
468 //-------------------------------------------------------------------------//
469 public:
470     void set_mode(char *Mode)
471     {
472             sprintf(mode,"%s",Mode);
473     }
474
475     /*
476             Fonction cont(...)
477             Cette fonction ouvre le fichier en mode UPDATE, enleve les caracteres
478             }; et poursuit l'ecriture dans le cas de la poursuite d'une simulation
479             qui a ete arretee.
```

```
480     */
481     bool cont(void)
482     {
483             int fileOK;
484
485             set_mode("r+b0");
486             if(!file_open)
487             {
488                     fileOK=open_file(mode);
489                     file_open=true;
490                     if(!fileOK)
491                     {
492                             printf("Erreur d'ouverture de fichier -- read_coeff()");
493                             file_open=false;
494                             getchar();
495                             return(false);
496                     }
497             }
498
499             fseek(file_handle,-2L,SEEK_END);
500             return(true);
501     }
502
503
504
505
506 //-------------------------------------------------------------------------//
507 // read_string()
                                        //
508 //-------------------------------------------------------------------------//
509 // Lecture d'une chaine de char dans le fichier et retour du pointeur.        //
510 //-------------------------------------------------------------------------//
511 // Luc Romain, 17 decembre 1998
                                //
512 //-------------------------------------------------------------------------//
513
514 bool read_string(char *buff)
515 {
516     int fileOK;
517
518     if(!file_open)
519     {
520             fileOK=open_file(mode);
521
522             if(!fileOK)
523             {
524                     printf("Erreur d'ouverture de fichier -- read_string()");
525                     file_open=false;
526                     getchar();
527                     return(false);
528             }
529     }
530
531     if(!end_of_file())
532     {
533             fscanf(file_handle,"%s",buff);
534     }
535     return(true);
536 }
```

```
537
538 };
539
540
541 ////////////////////////////////////////////////////////////////////////////
542 // StrAnlz                                                                //
543 //------------------------------------------------------------------------//
544 // Classe specialisee dans l'analyse des chaines de caracteres            //
545 // contenues dans les fichiers.                                           //
546 //                                                                        //
547 //------------------------------------------------------------------------//
548 // Auteur: Luc Romain                                                     //
549 // Date:    15 juin 1999                                                  //
550 ////////////////////////////////////////////////////////////////////////////
551 class StrAnlz
552 {
553 private:
554    char *pstr; // Pointeur sur le debut de la chaine
555    char *ppos; // Pointeur sur la position courante de la chaine.
556    int       len;   // Longueur de la chaine +1 (null char)
557
558 public:
559
560    // Constructeurs:
561
562    StrAnlz()
563    {
564            pstr=NULL;
565            len=0;
566    }
567
568    StrAnlz(char *p)
569    {
570            sets(p);
571    }
572
573    -StrAnlz()
574    {
575            if(pstr!=NULL)
576                    delete pstr;
577    }
578
579 //------------------------------------------------------------------------//
580 // sets(char *p)                                                          //
581 //------------------------------------------------------------------------//
582 // Initialise la classe : la chaine de char pointee par p est copiee      //
583 //                                                                        //
584 //------------------------------------------------------------------------//
585 // Luc Romain, 15 juin 1999                                               //
586 //------------------------------------------------------------------------//
587
588
589    void sets(char *p)
590    {
591            len=strlen(p)+1;
592            pstr=new char[len];
```

```
593             sprintf(pstr,"%s",p);
594             ppos=pstr;
595     }
596 protected:
597 //-------------------------------------------------------------------------//
598 // char *fcfp(char *pos)
599 //-------------------------------------------------------------------------//
600 // Cette fonction trouve la premiere occurence d'un caractere dans la chaine
601 // de caracteres pointee par pos.
602 // Valeur de retour: l'adresse du premier caractere pointe par *pos.
603 //

604 //-------------------------------------------------------------------------//
605 // Luc Romain, 15 juin 1999
606 //-------------------------------------------------------------------------//
607 char *fcfp(char *pos) // find char from ppos position.
608     {
609             char *p;
610
611             p=pos;
612
613             while(!isalpha(*p)&&!isdigit(*p)&&(*p!='0'))
614             {
615                     p++;
616             }
617
618             return(p);
619     }
620
621 //-------------------------------------------------------------------------//
622 // char *fsfp(char *pos)
623 //-------------------------------------------------------------------------//
624 // Cette fonction trouve la premiere occurence d'un espace dans la chaine
625 // de caracteres pointee par pos.
626 // Valeur de retour: l'adresse du premier espace pointe par *pos.
627 //

628 //-------------------------------------------------------------------------//
629 // Luc Romain, 15 juin 1999
630 //-------------------------------------------------------------------------//
631 char *fsfp(char *pos)
632     {
633             char *p;
634
635             p=pos;
636
637             while((*p!=32)&&(*p!='0'))
638             {
639                     p++;
640             }
641
642             return(p);
643     }
644
645 public:
646 //-------------------------------------------------------------------------//
647 // int gets(char *buff)
648 //-------------------------------------------------------------------------//
649 // Cette fonction retourne les mots separes par des espaces dans la chaine
```

```
650  // de caracteres de la classe. Les appels subsequents poursuivent la recherche
651  // dans la meme chaine de caractere. Lorsque '0' est rencontree, la recherche
652  // ne se poursuit plus et buff='0'.
653  //
654  // Valeur de retour: nombre de caracteres ecrits dans buff.
655  //

656  //-------------------------------------------------------------------------//
657  // Luc Romain, 15 juin 1999
658  //-------------------------------------------------------------------------//
659     int gets(char *buff)
660     {
661        char *p1,*p2; // Pointeur de debut et fin de la chaine.
662        int x;                  // Compteur
663        int slen;      // Longueur de la chaine trouvee.
664
665        p1=fcfp(ppos);
666        if((p1!=NULL)&&(*p1!=0))
667        {
668                p2=fsfp(p1);
669                ppos=p2;
670                slen=p2-p1;
671
672                // Copie de la chaine dans buffer.
673                for(x=0;x<slen;x++)
674                     {
675                             buff[x]=p1[x];
676                     }
677                     buff[slen]='0';
678
679                     return(slen);
680             }
681             else
682             {
683                slen=0;
684                buff='0';
685                return(0);
686             }
687     }
688
689
690  //-------------------------------------------------------------------------//
691  // void reset()
692  //-------------------------------------------------------------------------//
693  // Initialise le pointeur ppos sur le debut de la chaine de char de la
694  // classe.
695  //
696  //

697  //-------------------------------------------------------------------------//
698  // Luc Romain, 15 juin 1999
699  //-------------------------------------------------------------------------//
700     void reset()
701     {
702             ppos=pstr;
703     }
704
705  };
```

build.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                    BUILD.H         //
4   ////////////////////////////////////////////////////////////////////////
5   // Procedure                    : cube                                //
6   // ...................................................................//
7   // Role                         : construction d'un cube d'un materiau //
8   //                              : donne                               //
9   // ...................................................................//
10  // Valeur renvoyee              : aucune                              //
11  // Parametres d'appel           : coordonnees du point origine        //
12  //                              : dimensions du cube                  //
13  //                              : materiau                            //
14  //                              : espace physique                     //
15  // Parametres modifies          : espace physique                     //
16  ////////////////////////////////////////////////////////////////////////
17  void cube(int istart,int jstart, int kstart,
18                int NXwide, int NYwide, int NZwide,
19                int mat_index, espace_physique my_space)
20  {
21  int i,j,k;
22  int imax=istart+NXwide-1;
23  int jmax=jstart+NYwide-1;
24  int kmax=kstart+NZwide-1;
25  cellule_physique my_cell;
26
27  if (NXwide == 0)
28     for (k=kstart; k<=kmax; k++)
29             for (j=jstart; j<=jmax; j++)
30             {
31                     my_cell= my_space.get(istart,j,k);
32                     my_cell.sety(mat_index);
33                     my_cell.setz(mat_index);
34                     my_space.put(my_cell,istart,j,k);
35                     //.............................
36                     my_cell= my_space.get(istart,j,k+1);
37                     my_cell.sety(mat_index);
38                     my_space.put(my_cell,istart,j,k+1);
39                     //.............................
40                     my_cell= my_space.get(istart,j+1,k);
41                     my_cell.setz(mat_index);
42                     my_space.put(my_cell,istart,j+1,k);
43             }
44  else if  (NYwide == 0)
45     for (k=kstart; k<=kmax; k++)
46             for (i=istart; i<=imax; i++)
47             {
```

```
48                      my_cell= my_space.get(i,jstart,k);
49                      my_cell.setx(mat_index);
50                      my_cell.setz(mat_index);
51                      my_space.put(my_cell,i,jstart,k);
52                      //.............................
53                      my_cell= my_space.get(i,jstart,k+1);
54                      my_cell.setx(mat_index);
55                      my_space.put(my_cell,i,jstart,k+1);
56                      //.............................
57                      my_cell= my_space.get(i+1,jstart,k);
58                      my_cell.setz(mat_index);
59                      my_space.put(my_cell,i+1,jstart,k);
60              }
61  else if  (NZwide == 0)
62     for (j=jstart; j<=jmax; j++)
63              for (i=istart; i<=imax; i++)
64              {
65                      my_cell= my_space.get(i,j,kstart);
66                      my_cell.setx(mat_index);
67                      my_cell.sety(mat_index);
68                      my_space.put(my_cell,i,j,kstart);
69                      //.............................
70                      my_cell= my_space.get(i,j+1,kstart);
71                      my_cell.setx(mat_index);
72                      my_space.put(my_cell,i,j+1,kstart);
73                      //.............................
74                      my_cell= my_space.get(i+1,j,kstart);
75                      my_cell.sety(mat_index);
76                      my_space.put(my_cell,i+1,j,kstart);
77                      //.............................
78              }
79  else
80     for (k=kstart; k<=kmax; k++)
81              for (j=jstart; j<=jmax; j++)
82                      for (i=istart; i<=imax; i++)
83                      {
84                      my_cell=my_space.get(i,j,k);      // Luc Romain a rajoute ceci.
85                      my_cell.setx(mat_index);
86                      my_cell.sety(mat_index);
87                      my_cell.setz(mat_index);
88                      my_space.put(my_cell,i,j,k);
89                      //.............................
90                      my_cell= my_space.get(i,j,k+1);
91                      my_cell.setx(mat_index);
92                      my_cell.sety(mat_index);
93                      my_space.put(my_cell,i,j,k+1);
94                      //.............................
95                      my_cell= my_space.get(i,j+1,k+1);
96                      my_cell.setx(mat_index);
97                      my_space.put(my_cell,i,j+1,k+1);  // Luc Romain
98                      //.............................
99                      my_cell= my_space.get(i,j+1,k);
100                     my_cell.setx(mat_index);
101                     my_cell.setz(mat_index);
102                     my_space.put(my_cell,i,j+1,k);
103                     //.............................
104                     my_cell= my_space.get(i+1,j,k);
105                     my_cell.sety(mat_index);
106                     my_cell.setz(mat_index);
```

```
107                     my_space.put(my_cell,i+1,j,k);
108                     //.............................
109                     my_cell= my_space.get(i+1,j,k+1);
110                     my_cell.sety(mat_index);
111                     my_space.put(my_cell,i+1,j,k+1);
112                     //.............................
113                     my_cell= my_space.get(i+1,j+1,k);
114                     my_cell.setz(mat_index);
115                     my_space.put(my_cell,i+1,j+1,k);
116                     }
117 }
118
119
120 ////////////////////////////////////////////////////////////////////////
121 // Procedure                        : buildsphere
            //
122 // ..................................................... //
123 // Role                             : Procedure de test.
    //
124 //                                          : Construit une sphere
            //
125 //                                          : de materiau #2 dans l'espace
    //
126 //                                          : Sphere de centre (sc,sc,sc) et rayon ra //
127 // ..................................................... //
128 // Valeur renvoyee          : aucune
    //
129 // Parametres d'appel       : espace physique
    //
130 // Parametres modifies      : espace physique
    //
131 ////////////////////////////////////////////////////////////////////////
132 void buildsphere(int cx,int cy,int cz,int ra,int mat_index,
133                              espace_physique my_space)
134 {
135 //typ_coeff         sc=17.5;
136 //typ_coeff         ra=8.2;
137 typ_coeff           r;
138 //typ_material      mat_index=3;
139
140 int NX=my_space.nx();
141 int NY=my_space.ny();
142 int NZ=my_space.nz();
143 int i,j,k;
144
145 for ( i=1; i<= NX; i++ )
146     for ( j=1; j<=NY; j++ )
147             for ( k=1; k<=NZ; k++ )
148             {
149                     r= sqrt ((i-cx)**(i-cx)+(j-cy)**(j-cy)+(k-cz)*(k-cz));
150                     if (r<ra)
151                             cube(i,j,k,1,1,1,mat_index, my_space);
152             }
153
154 }
155
156 ////////////////////////////////////////////////////////////////////////
157 // Procedure                        : buildcylindre
            //
```

```
158 // ------------------------------------------------------------ //
159 // Role                              : Procedure de test.
    //
160 //                                           : Construit un cylindre
            //
161 //                                           : de materiau #2 dans l'espace
    //
162 //                                           : Cylindre de centre (sc,sc,sc) et rayon ra
    //
163 //                                           : et hauteur: hi<=z<=hf
164 // ------------------------------------------------------------ //
165 // Valeur renvoyee          : aucune
    //
166 // Parametres d'appel       : espace physique
    //
167 // Parametres modifies      : espace physique
    //
168 ////////////////////////////////////////////////////////////////////////
169 void buildcylindre(int ra,int cx,int cy,int hi,int hf,typ_material mat_index,
170                                  espace_physique my_space)
171 {
172
173 typ_coeff           r;
174
175 int NX=my_space.nx();
176 int NY=my_space.ny();
177 int NZ=my_space.nz();
178 int i,j,k;
179
180
181 //buildsphere(cx,cy,hi,ra,mat_index,my_space);
182 //buildsphere(cx,cy,hf,ra,mat_index,my_space);
183
184 for ( i=1; i<= NX; i++ )
185    for ( j=1; j<=NY; j++ )
186            for ( k=1; k<=NZ; k++ )
187            {
188                    r= sqrt ((i-cx)*(i-cx)+(j-cy)*(j-cy));
189                    if ((r<=ra) &&(k>=hi)&&(k<=hf))
190                            cube(i,j,k,1,1,1,mat_index, my_space);
191            }
192
193 }
194
195 ////////////////////////////////////////////////////////////////////////
196 // Procedure                         : ext_cylindre
            //
197 // ------------------------------------------------------------ //
198 // Role                              : Procedure de test.
    //
199 //                                           : Construit un cylindre vide a l'interieur  //
200 //                                           : avec les sommets pleins.
    //
201 //                                           : Selon les recommandations de M. Trueman.  //
202 //                                           : Pour reduire l'effet de changement de     //
203 //                                           : milieu.
                          //
204 // ------------------------------------------------------------ //
205 // Valeur renvoyee          : aucune
```

```
        //
206  // Parametres d'appel        : espace physique
        //
207  // Parametres modifies       : espace physique
        //
208  ////////////////////////////////////////////////////////////////////////
209  void ext_cylindre(int thick,int ra,int cx,int cy,int hi,int hf,int index,espace_physique
my_space)
210  {
211
212  typ_coeff           r;
213
214  int NX=my_space.nx();
215  int NY=my_space.ny();
216  int NZ=my_space.nz();
217  int i,j,k;
218
219
220  for ( i=1; i< NX; i++ )
221     for ( j=1; j<NY; j++ )
222             for ( k=1; k<NZ; k++ )
223             {
224                     r= sqrt ((i-cx)*(i-cx)+(j-cy)*(j-cy));
225
226                     if((r<=ra)&&(r>=ra-thick)&&(k>=hi)&&(k<=hf))
227                             cube(i,j,k,1,1,1,index, my_space);
228
229                     if((r<=ra)&&(((k>=hi)&&(k<=hi+thick))||((k<=hf)&&(k>=hf-thick))))

230                             cube(i,j,k,1,1,1,index, my_space);
231
232             }
233
234  }
235
236  ////////////////////////////////////////////////////////////////////////
237  // Procedure                        : buildellipse
             //
238  // ..................................................................... //
239  // Role                             : Procedure de test.
        //
240  //                                          : Construit une ellipse
             //
241  //                                          : de materiau #index dans l'espace
        //
242  //                                          : Cylindre de centre (sc,sc,sc) et rayon ra et rb
243  //                                          : et hauteur: hi<=z<=hf
244  // ..................................................................... //
245  // Valeur renvoyee          : aucune
        //
246  // Parametres d'appel       : espace physique
        //
247  // Parametres modifies      : espace physique
        //
248  ////////////////////////////////////////////////////////////////////////
249  void buildellipse(int ra,int rb,int cx,int cy,int hi,int hf,
250                               typ_material mat_index,espace_physique my_space)
251  {
252
```

```
253  typ_coeff           r;
254
255  int NX=my_space.nx();
256  int NY=my_space.ny();
257  int NZ=my_space.nz();
258  int i,j,k;
259
260
261  //buildsphere(cx,cy,hi,ra,mat_index,my_space);
262  //buildsphere(cx,cy,hf,ra,mat_index,my_space);
263
264  for ( i=1; i< NX; i++ )
265     for ( j=1; j<NY; j++ )
266             for ( k=1; k<NZ; k++ )
267             {
268                     r=(typ_coeff)((i-cx)*(i-cx)/(ra*ra)+(j-cy)*(j-cy)/(rb*rb));
269                     if ((r<=1) &&(k>=hi)&&(k<=hf))
270                             cube(i,j,k,1,1,1,mat_index, my_space);
271             }
272
273  }
274
275  ////////////////////////////////////////////////////////////////////////
276  // Procedure                        : ext_ellipse
             //
277  // ..................................................................... //
278  // Role                             : Procedure de test.
     //
279  //                                          : Construit une ellipse vide a l'interieur  //
280  //                                          : avec les sommets pleins.
     //
281  //                                          : Selon les recommandations de M. Trueman.  //
282  //                                          : Pour reduire l'effet de changement de     //
283  //                                          : milieu.
                              //
284  // ..................................................................... //
285  // Valeur renvoyee          : aucune
     //
286  // Parametres d'appel       : espace physique
     //
287  // Parametres modifies      : espace physique
     //
288  ////////////////////////////////////////////////////////////////////////
289  void ext_ellipse(int thick,int ra,int rb,int cx,int cy,int hi,int hf,
290                               int index,espace_physique my_space)
291  {
292
293  typ_coeff           r,rp;
294
295  int NX=my_space.nx();
296  int NY=my_space.ny();
297  int NZ=my_space.nz();
298  int i,j,k;
299
300
301  for ( i=1; i< NX; i++ )
302     for ( j=1; j<NY; j++ )
303             for ( k=1; k<NZ; k++ )
304             {
```

```
305                     r=(typ_coeff)((i-cx)*(i-cx)/(ra*ra)+(j-cy)*(j-cy)/(rb*rb));
306                     rp=(typ_coeff)((i-cx)*(i-cx)/((ra-thick)*(ra-thick))+
307                             (j-cy)*(j-cy)/((rb-thick)*(rb-thick)));
308
309                     if((r<=1)&&(rp>=1)&&(k>=hi)&&(k<=hf))
310                             cube(i,j,k,1,1,1,index, my_space);
311
312                     if((r<=1)&&(((k>=hi)&&(k<=hi+thick))||((k<=hf)&&(k>=hf-thick))))

313                             cube(i,j,k,1,1,1,index, my_space);
314
315             }
316
317 }
```

coeur.h

```
1   #define AMP (typ_coeff)10.0 // Amplitude du sinus cardiaque
2   #define FREQ (typ_coeff)10000000.0   // Frequence du sinus cardiaque
3   #define OFFSET (typ_coeff)0.0 // Offset du sinus cardiaque
4   ////////////////////////////////////////////////////////////////////////////
5   // Coeur                                                                  //
6   //------------------------------------------------------------------------//
7   // Cette classe englobe toutes les fonctions necessaires pour polariser   //
8   // plusieurs dipoles. Cette classe contient plusieurs dipoles dont        //
9   // chaque dipole possede des caracteristiques identiques                  //
10  // afin de bien simuler la stimulation synchrone du coeur.                //
11  //------------------------------------------------------------------------//
12  // Auteur:                                  Luc Romain                    //
13  // Date:                                    28 septembre 1998             //
14  // Derniere modification:   Aucune                                        //
15  ////////////////////////////////////////////////////////////////////////////
16
17  class Coeur
18  {
19  public:
20
21
22      // -- constructeurs
23      Coeur()
24      {
25              initok=false;
26      };
27
28      Coeur(espace_champs *Efld,espace_physique *Phsp,
29                      var_env_comp env,int cx,int cy,int cz,int rayon,
30                      typ_material Index)
31      {
32              nb_dipoles=0;
33              Cx=cx;
34              Cy=cy;
35              Cz=cz;
36              r=rayon;
37              index=Index;
38              //dipoles=new Dipole[nb_dipoles];
39              init_coeur2(Efld,Phsp,env);
40      };
41
42      ~Coeur()
43      {
44              delete[] dipoles;
45      };
46
47  //------------------------------------------------------------------------//
48  // polarise(...)                                                          //
49  //------------------------------------------------------------------------//
50  // Cette fonction appelle la fonction de polarisation de chaque cellule.  //
```

```
                                                    //
51  //-------------------------------------------------------------------------//
52  // Luc Romain, 29 septembre 1998
                        //
53  //-------------------------------------------------------------------------//
54
55      void polarise(typ_coeff time)
56      {
57              int x;
58              x=1;
59              DipoleList.top().polarise(time);
60
61              while(x<nb_dipoles)
62              {
63                      DipoleList.next().polarise(time);
64                      x++;
65              }
66      };
67
68
69  protected:
70      int nb_dipoles,Cx,Cy,Cz,r;
71      typ_material index;
72      Linked_Chain<Dipole> DipoleList;
73      Dipole *dipoles;
74      bool initok;
75
76  //-------------------------------------------------------------------------//
77  // init_coeur
                                                  //
78  //-------------------------------------------------------------------------//
79  // Cette fonction cree un coeur de forme spherique dans le cylindre.          //
80  //
                                                             //
81  //-------------------------------------------------------------------------//
82  // Luc Romain, 22 septembre 1998
                        //
83  //-------------------------------------------------------------------------//
84
85      void init_coeur(espace_champs *ElcFld,espace_physique *EspPhys,
86                                                  var_env_comp env)
87      {
88              int i,j,k,t;
89
90
91
92              cellule_physique ph_cell;
93              typ_coeff frequence,offset;
94
95              printf("------- Construction du coeur");
96              frequence=20000000;       // 20MHz
97
98              offset=0;
99
100             // Sommet de la demie-sphere:
101
102
103             t=0;
104
```

```
105             while(t<nb_dipoles)
106             {
107                     // 8 points / cercle * 5 cercles = 40 iterations.
108                     i=(int)(r*sin(2*PI*t/40)*cos(2*PI*t/8)+Cx);
109                     j=(int)(r*sin(2*PI*t/40)*sin(2*PI*t/8)+Cy);
110                     k=(int)(r*cos(PI*t/40)+Cz);
111
112                     ph_cell.setx(2);// antenne
113                     ph_cell.sety(2);// antenne
114                     ph_cell.setz(2);// antenne
115   //                                                                        anglex,anglez
116                     dipoles[t].init(i,j,k,ph_cell, 0, -3*PI/8,ElcFld,EspPhys);
117                     dipoles[t].set_fonction(0,50,frequence,offset,2000*env.dt);
118                     t++;
119             }
120             initok=true;
121     };
122
123 //-------------------------------------------------------------------------//
124 // init_coeur2
                                                //
125 //-------------------------------------------------------------------------//
126 // Cette fonction cree un coeur de forme spherique dans le cylindre.          //
127 // Cette fonction utilise la liste chainée DipoleList.
    //
128 // La structure du coeur est purement sphérique.
    //
129 //
                                                        //
130 //-------------------------------------------------------------------------//
131 // Luc Romain, 7 mai 1999
132 //-------------------------------------------------------------------------//
133
134     void init_coeur2(espace_champs *ElcFld,espace_physique *EspPhys,
135                                                 var_env_comp env)
136     {
137             int x,y,z;
138             int Nx,Ny,Nz;
139             int Rx2,Ry2,Rz2;
140             Dipole dipole;
141             cellule_physique ph_cell;
142             typ_coeff theta,phi;
143
144             Nx=EspPhys->nx()-1;
145             Ny=EspPhys->ny()-1;
146             Nz=EspPhys->nz()-1;
147
148             ph_cell.setx(index);
149             ph_cell.sety(index); // Cette direction ne polarise jamais.
150             ph_cell.setz(index);
151
152             for(z=1;z<Nz;z++)
153             {
154                     for(y=1;y<Ny;y++)
155                     {
156                             for(x=1;x<Nx;x++)
157                             {
158                                     Rx2=(x-Cx)*(x-Cx);
159                                     Ry2=(y-Cy)*(y-Cy);
```

```
160                                     Rz2=(z-Cz)*(z-Cz);
161                                     if(sqrt(Rx2+Ry2+Rz2)<r)
162                                     {
163                                     // Calcul de l'angle theta et phi

164                                             if((x-Cx)>=0)
165                                             {
166                                                     if(x==Cx)
167                                                     {
168                                                             if((y-Cy)>0)
169                                                                     theta=PI/2;
170                                                             else
171                                                                     theta=-PI/2;
172                                                     }
173                                                     else
174                                                             theta=atan((y-Cy)/(x-Cx));
175                                             }
176                                             else
177                                             {
178                                                     theta=atan((y-Cy)/(x-Cx))+PI;
179                                             }
180
181                                     // Calcul de phi.
182                                             if(Rx2+Ry2>0)
183                                                     phi=atan((z-Cz)/sqrt(Rx2+Ry2));
184                                             else
185                                                     phi=0;
186                                     // Fin du calcul des angle theta et phi.
187
188                                     dipole.init(x,y,z,ph_cell,theta,phi,ElcFld,EspPhys);
189                                     dipole.set_fonction(0,AMP,FREQ,OFFSET,2000*env.dt);
190
191                                     DipoleList.Add_elem(&dipole);
192                                     }
193                             }
194                     }
195             }
196
197
198             nb_dipoles=DipoleList.get_nb_elem();
199             initok=true;
200             printf("------- Coeur: %i Dipoles initialisés",nb_dipoles);
201     }
202
203 //-------------------------------------------------------------------------//
204 // init_coeur3                                          //
205 //-------------------------------------------------------------------------//
206 // Cette fonction cree un coeur de forme spherique dans le cylindre.           //
207 // Cette fonction utilise la liste chainée DipoleList.
    //
208 // La structure du coeur est purement sphérique.
    //
209 // Les dipoles sont tangents à la surface.
            //
210 //                                                  //
211 //-------------------------------------------------------------------------//
212 // Luc Romain, 7 mai 1999
```

```
213 //---------------------------------------------------------------------------//
214
215     void init_coeur3(espace_champs *ElcFld,espace_physique *EspPhys,
216                                             var_env_comp env)
217     {
218             int x,y,z;
219             int Nx,Ny,Nz;
220             int Rx2,Ry2,Rz2;
221             Dipole dipole;
222             cellule_physique ph_cell;
223             typ_coeff theta,phi;
224
225             Nx=EspPhys->nx()-1;
226             Ny=EspPhys->ny()-1;
227             Nz=EspPhys->nz()-1;
228
229             ph_cell.setx(index);
230             ph_cell.sety(index); // Cette direction ne polarise jamais.
231             ph_cell.setz(index);
232
233             for(z=1;z<Nz;z++)
234             {
235                     for(y=1;y<Ny;y++)
236                     {
237                             for(x=1;x<Nx;x++)
238                             {
239                                     Rx2=(x-Cx)*(x-Cx);
240                                     Ry2=(y-Cy)*(y-Cy);
241                                     Rz2=(z-Cz)*(z-Cz);
242                                     if(sqrt(Rx2+Ry2+Rz2)<r)
243                                     {
244                                     // Calcul de l'angle theta et phi

245                                             if((x-Cx)>=0)
246                                             {
247                                                     if(x==Cx)
248                                                     {
249                                                             if((y-Cy)>0)
250                                                                     theta=PI/2;
251                                                             else
252                                                                     theta=-PI/2;
253                                                     }
254                                                     else
255                                                             theta=atan((y-Cy)/(x-Cx));
256                                             }
257                                             else
258                                             {
259                                                     theta=atan((y-Cy)/(x-Cx))+PI;
260                                             }
261
262                                     // Calcul de phi.
263                                             if(Rx2+Ry2>0)
264                                                     phi=-PI/2*cos(PI*(z-Cz)/(2*r));
265                                             else
266                                                     phi=0;
267
268                                             if((z<Cz)&&(phi!=0))
269                                                     phi=PI-phi;
270
```

```
271                                     // Fin du calcul des angle theta et phi.
272
273                                     dipole.init(x,y,z,ph_cell,theta,phi,ElcFld,EspPhys);
274                                     dipole.set_fonction(0,AMP,FREQ,OFFSET,2000*env.dt);
275
276                                     DipoleList.Add_elem(&dipole);
277                                     }
278                                 }
279                             }
280                 }
281
282
283                 nb_dipoles=DipoleList.get_nb_elem();
284                 initok=true;
285                 printf("------- Coeur: %i Dipoles initialisés",nb_dipoles);
286         }
287
288
289 };
```

constantes.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                    CONSTANTES.H    //
4   //                                                                    //
5   ////////////////////////////////////////////////////////////////////////
6   #define  EPS0        (8.854E-12)
7   #define  XMU0        (1.2566306E-6)
8   #define  ETA0        (376.733341)
9   #define  C           (1.0/sqrt(XMU0*EPS0))
10  #define  PI          (4.0*atan(1.0))
11  ////////////////////////////////////////////////////////////////////////
12  #define CHAMP_NUL            (typ_champ)              0
13  #define PHYS_PRM_NUL(typ_phys_prm)   0
14  #define MATERIAU_NUL("MAT_NUL",0.0,0.0,0.0)
15  #define CELL_MAT_VIDE        (0,0,0)
16  #define CELL_CHAMP_NUL       (CHAMP_NUL,CHAMP_NUL,CHAMP_NUL)
17  ////////////////// Types d'electrodes ///////////////////////
18  #define     E_HELICOIDE              (int) 0
19  #define E_BIPOLAIRE          (int) 1
20  ////////////////////////////////////////////////////////////////////////
21  #define VAL_IN_ELEM3D_NULLE          (type)   0
22
23     // Dangereux...
24     // Pourquoi ne pas la definir directement ds le module Elem3D,
25     // puisque c'est le seul ou le forcage de type ait un sens ?
```

diaphragme2.h

```
1   #define DIA_START (int)2
2   #define CTE          (typ_coeff)16.0
3   #define A            (typ_coeff)20.0  // Largeur
4   #define B            (typ_coeff)7.0 // Profondeur
5   #define AMP_D        (typ_coeff)10.0
6   #define FREQ_D       (typ_coeff)10000000.0
7   #define OFFSET_D (typ_coeff)0.0
8
9   class diaphragme2
10  {
11  protected:
12     int Cx,Cy,Cz;             // Coordonnees du sommet reel
13     typ_material index;
14     Linked_Chain<Dipole> DipoleList;
15     espace_champs *Efld;
16     espace_physique *Psp;
17     var_env_comp env;
18
19  public:
20     diaphragme2(int px,int py,int pz,typ_material Index,
21             espace_champs *efld,espace_physique *psp,var_env_comp Env)
22     {
23             Cx=px;
24             Cy=py;
25             Cz=pz;
26             Efld=efld;
27             Psp=psp;
28             env=Env;
29             index=Index;
30             init_diaphragme2();
31     }
32
33     diaphragme2(){ } ;
34
35     ~diaphragme2(){};
36
37  protected:
38     void init_diaphragme2(void)
39     {
40             int x,y,z;
41             int Nx,Ny,Nz;
42             typ_coeff surface,theta,anglez;
43             typ_coeff rp,ri;
44             //typ_coeff ux,uy
45             typ_coeff uz; // vecteur de direction
46             //typ_coeff mx,my,mz; // vecteur unitaire de direction
47             cellule_physique p_cell;
48             Dipole dipole;
49
50             printf("------- Construction du diaphragme");
51             // Placement des dipoles
52
53             p_cell.setx(index);
54             p_cell.sety(index);
55             p_cell.setz(index);
56
```

```
57              printf("------- Initialisation du diaphragme2");
58
59    //        printf("hauteur=%i, diastart=%i, cz=%i",HAUT_DIA2,DIA_START,Cz);
60    //        getchar();
61              Nx=Efld->nx();
62              Ny=Efld->ny();
63              Nz=Efld->nz();
64
65              for(z=1;z<Nz;z++)
66              {
67                      for(y=1;y<Ny;y++)
68                      {
69                              for(x=1;x<Nx;x++)
70                              {
71                                      surface=Cz-CTE*((x-Cx)*(x-Cx)/(A*A)+(y-Cy)*(y-Cy)/(B*B));
72                                      rp=(x-Cx)*(x-Cx)/(A*A)+(y-Cy)*(y-Cy)/(B*B); // Rayon interne
de 1<ellipse
73                                      ri=sqrt((x-Cx)*(x-Cx)+(y-Cy)*(y-Cy)); // centre du
diaphragme=trou
74                                      if((fabs(surface-z)<0.5)&&(rp<=1)&&(ri>DIA_START))
75                                      {
76                                              if((x-Cx)<=0)
77                                              {
78                                                      if(x==Cx)
79                                                      {
80                                                              if((y-Cy)>0)
81                                                                      theta=-PI/2;
82                                                              else
83                                                                      theta=+PI/2;
84                                                      }
85                                                      else
86                                                              theta=atan((y-Cy)/(x-Cx));
87                                              }
88                                              else
89                                              {
90                                                      theta=atan((y-Cy)/(x-Cx))+PI;
91                                              }
92                                              uz=fabs(-2*CTE*(x-Cx)*cos(theta)/(A*A)-2*CTE*(y-
Cy)*sin(theta)/(B*B));
93                                              anglez=atan(uz);
94
95                                              dipole.init(x,y,z,p_cell,theta,anglez,Efld,Psp);
96
        dipole.set_fonction(0,AMP_D,FREQ_D,OFFSET_D,2000*env.dt);
97
98                                              if(DipoleList.Add_elem(&dipole)==false)
99                                                      printf(":
Diaphragme2::DipoleList.Add_elem");
100
101                                     }
102                             }
103                     }
104             }
105             printf("------- Diaphragme: %i Dipoles initialisés Sommet (%i,%i,%i)",
106                     DipoleList.get_nb_elem(),Cx,Cy,Cz);
107     }
108 public:
109     void polarise(typ_coeff time)
110     {
```

```
111             int x;
112             int Nb_Dipole;
113             x=1;
114
115             Nb_Dipole=DipoleList.get_nb_elem();
116             DipoleList.top().polarise(time);
117
118             while(x<Nb_Dipole)
119             {
120                     DipoleList.next().polarise(time);
121                     x++;
122             }
123
124     }
125 };
```

dipole.h

```
1   ////////////////////////////////////////////////////////////////////////////
2   // Dipole                                                                 //
3   //------------------------------------------------------------------------//
4   // Cette classe est une classe de base servant a la modelisation du       //
5   // diaphragme. Le dipole possede un position, une direction ainsi qu'une  //
6   // amplitude. Il peut etre polarise selon differentes methodes (random,   //
7   // sin, pulse).                                                           //
8   //------------------------------------------------------------------------//
9   // Auteur:                                  Luc Romain                    //
10  // Date:                                    18 septembre 1998             //
11  // Derniere modification:    Aucune                                       //
12  ////////////////////////////////////////////////////////////////////////////
13
14  class Dipole
15  {
16  public:
17
18     // ----- Constructeurs -----
19     Dipole()
20     {
21             initok=false;
22     };
23
24     Dipole(int px,int py,int pz,cellule_physique p_cell,typ_coeff AngleX,
25                typ_coeff AngleZ,espace_champs *efld,espace_physique *ph_sp)
26     {
27             init(px,py,pz,p_cell,AngleX,AngleZ,efld,ph_sp);
28             return;
29     };
30
31     // ----- destructeur -----
32
33     ~Dipole() {};
34
35  //------------------------------------------------------------------------//
36  // init(...)                                                              //
37  //------------------------------------------------------------------------//
38  // Cette fonction initialise tous les parametres utilises par DIPOLE      //
39  //------------------------------------------------------------------------//
40  // Luc Romain, 29 septembre 1998                                          //
41  //------------------------------------------------------------------------//
42
43     void init(int px,int py,int pz,cellule_physique p_cell,typ_coeff AngleX,
44               typ_coeff AngleZ,espace_champs *efld,espace_physique *esp_ph)
45     {
46             initok=true;
47             x=px;
48             y=py;
49             z=pz;
```

```
50              anglex=AngleX;
51              anglez=AngleZ;
52              dir_cell=p_cell;
53              omega=0;
54              amp=10;
55              theta=0;
56              ElcFld=efld;
57              PhysSpace=esp_ph;
58              place_dipole();                 // Insere l'index 2 dans le milieu physique.
59              return;
60      };
61
62  //-------------------------------------------------------------------------//
63  // set_value(...)                                                          //
64  //-------------------------------------------------------------------------//
65  // Cette fonction force la valeur value comme amplitude au dipole. Cette   //
66  // valeur est forcee selon l'angle specifie lors de l'initialisation.      //
67  //-------------------------------------------------------------------------//
68  // Luc Romain, 29 septembre 1998                                           //
69  //-------------------------------------------------------------------------//
70
71      void set_value(typ_coeff value)
72      {
73              if(!initok)
74                      return;
75              cellule_champs my_cell;
76
77              my_cell=ElcFld->get(x,y,z);
78
79              if(dir_cell.cx()==2)
80                      my_cell.setx(value*cos(anglez)*cos(anglex));
81
82              if(dir_cell.cy()==2)
83                      my_cell.sety(value*cos(anglez)*sin(anglex));
84
85              if(dir_cell.cz()==2)
86                      my_cell.setz(value*sin(anglez));
87
88              ElcFld->put(my_cell,x,y,z);
89
90      };
91
92  //-------------------------------------------------------------------------//
93  // set_fonction(...)                                                       //
94  //-------------------------------------------------------------------------//
95  // Cette fonction assigne les parametres qui seront utilises lors du calcul //
96  // de la fonction de polarisation specifique (sin,pulse ou autre)          //
97  //-------------------------------------------------------------------------//
98  // Luc Romain, 29 septembre 1998                                           //
99  //-------------------------------------------------------------------------//
100
101     void set_fonction(int fct,typ_coeff Amp,typ_coeff Omega,typ_coeff Theta,
102                                             typ_coeff Betadt)
103     {
104             Fonction=fct;
```

```
105             amp=Amp;
106             omega=Omega;
107             theta=Theta;
108             betadt=Betadt;
109     };
110
111 //-------------------------------------------------------------------------//
112 // polarise(...)                                                           //
113 //-------------------------------------------------------------------------//
114 // Cette fonction assigne la valeur de polarisation a la cellule           //
115 // selon la direction qui lui est propre.                                  //
116 //-------------------------------------------------------------------------//
117 // Luc Romain, 29 septembre 1998                                           //
118 //-------------------------------------------------------------------------//
119
120     void polarise(typ_coeff time)
121     {
122             typ_champ val;
123             cellule_champs my_cell;
124
125             if(!initok)
126             {
127                     printf("Dipole: initok=false!!!");
128                     getchar();
129                     return;
130             }
131             val=fonction(Fonction,time);      // calcul de la fonction desiree
132
133             my_cell=ElcFld->get(x,y,z);
134
135             // dir_cell ne provient pas de l'espace reel.
136             // C'est pour cela que cette fonction parvenait quand meme
137             // a polariser le milieu malgre le fait que l'espace reel
138             // n'est pas forme de materiau #2.
139
140             if(dir_cell.cx()==2)
141                     my_cell.setx(val*cos(anglez)*cos(anglex));
142
143             if(dir_cell.cy()==2)
144                     my_cell.sety(val*cos(anglez)*sin(anglex));
145
146             if(dir_cell.cz()==2)
147                     my_cell.setz(val*sin(anglez));
148
149             ElcFld->put(my_cell,x,y,z);
150             return;
151     };
152
153 protected:
154     bool initok;
155     typ_coeff anglex,anglez; // angles de direction selon x et z
156     typ_champ alpha,omega,theta;
157     typ_coeff amp,betadt;
158     int x,y,z;                                          // position 3D de la cellule
159     cellule_physique dir_cell;        // determine quelle composante a polariser
160     espace_champs *ElcFld;
```

```
161     espace_physique *PhysSpace;
162     int Fonction;                                   // identifie la fonction de polarisation
(sin,...)
163
164
165  //-------------------------------------------------------------------------//
166  // place_dipole(...)
                                //
167  //-------------------------------------------------------------------------//
168  // Cette fonction modifie l'espace physique du dipole et fixe le materiau      //
169  // a la valeur '2', ou 2 designe une antenne, valeur qui ne sera pas           //
170  // recalculee lors de l'algorithme electric_scattered_fields(...)                    //
171  //-------------------------------------------------------------------------//
172  // Luc Romain, 29 septembre 1998
                        //
173  //-------------------------------------------------------------------------//
174
175     void place_dipole()
176     {
177             if(!initok)
178                     return;
179             cellule_physique my_cell;
180
181             my_cell=PhysSpace->get(x,y,z);
182
183             // Attention: l'antenne ne fonctionne pas!!!
184             // Bien recalculer -- le champ # doit etre recalcule.
185             if(dir_cell.cx()==2)
186             {
187                     my_cell.setx(2);
188             }
189             if(dir_cell.cy()==2)
190             {
191                     my_cell.sety(2);
192             }
193             if(dir_cell.cz()==2)
194             {
195                     my_cell.setz(2);
196             }
197
198             PhysSpace->put(my_cell,x,y,z);
199     };
200
201  //-------------------------------------------------------------------------//
202  // fonction(...)
                                    //
203  //-------------------------------------------------------------------------//
204  // Cette fonction definit la forme fonctionnelle de polarisation du dipole.    //


205  // Le dipole peut etre polarise selon une sinusoidale ou un pulse.                   //
206  //-------------------------------------------------------------------------//
207  // Luc Romain, 29 septembre 1998
                        //
208  //-------------------------------------------------------------------------//
209
210     typ_champ fonction(int f,typ_coeff time)
211     {
212
```

```
213             typ_champ v,enveloppe=1;
214
215             alpha=(4/betadt)*(4/betadt);
216
217
218             switch(f)
219             {
220             case 0:  // sinus
221                     {
222                             //alpha=8/(betadt*betadt);
223                             //alpha=1/(betadt*betadt); // Lobe plus large.
224                             /*
225                             if(time<betadt)
226                             {
227                                     //enveloppe=exp(-alpha*(time-betadt)*(time-betadt));
228                                     enveloppe=1.0;
229                             }
230                             else
231                             {
232                                     enveloppe=1.0;
233                                     //enveloppe=exp(-alpha*(time-betadt)*(time-betadt));
234                             }
235                             */
236                             //v=enveloppe*amp*sin(2*PI*omega*time+theta);
237                             v=amp*sin(2*PI*omega*time+theta);
238                             break;
239                     }
240             case 1: // pulse
241                     {
242                             v=amp*exp(-alpha*(time-betadt)*(time-betadt));
243                             break;
244                     }
245             case 2: // other
246                     {
247                             break;
248                     }
249             }
250             return(v);
251     }
252
253 };
254
```

display_for_debug.h

```
1   void show_sample_values ( espace_champs space, char typfield )
2   {
3   // printf ("================================================");
4      if (typfield== 'E')
5      {
6               printf ("------------------------ Champs electriques");
7               space.show_valx(17,18,25);
8               space.show_valx(17,18,18);
9      }
10     if (typfield== 'M')
11     {
12              printf ("------------------------ Champs magnetiques");
13              space.show_valy(17,18,24);
14              space.show_valy(17,18,17);
15     }
16  // printf ("================================================");
17
18  }
19
20  void show_all_fields (espace_champs Elc,espace_champs Mag)
21  {
22     int i,j,k;
23     int dimx=Elc.nx();
24     int dimy=Elc.ny();
25     int dimz=Elc.nz();
26     int deltax = 14;
27     int deltay = 14;
28     int deltaz = 14;
29     for (i=deltax;i<=dimx-deltax;i++)
30              for (j=deltay;j<=dimy-deltay;j++)
31              {
32                      for (k=deltaz;k<=dimz-deltaz;k++)
33                      {
34                      printf("ELEC: ");Elc.show_valx(i,j,k);
35                      printf("MAG : ");Mag.show_valy(i,j,k);
36                      }
37                      printf ("==== Any key to see next results, q to abort ===");
38                      if(getchar()=='q')       exit(1);
39
40              }
41  }
```

ei_fields.0.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                EI_FIELDS.0.H       //
4   //                                                CHAMPS INCIDENTS    //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
8   // Fonction                 : source                                  //
9   // ..................................................................//
10  // Role                     : definit la forme fonctionnelle du champ //
11  //                          : incident                                //
12  // ..................................................................//
13  // Valeur renvoyee          : champ electrique                        //
14  // Parametres d'appel       : distance de la source au pt d'evaluation//
15  //                          : variables d'environnement               //
16  // Parametres modifies      : aucun                                   //
17  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
18  typ_champ source(typ_distance my_dist, var_env_comp env)
19   {
20     typ_coeff betadt= env.beta * env.dt;
21     typ_coeff tau    = env.time - my_dist/C;
22     typ_coeff resu   = 0.0;
23
24     if( (tau >= 0.0) && (tau <= env.period))
25              resu =  exp ( - env.alpha * ( tau - betadt) * ( tau - betadt));
26     return (resu);
27   }
28
29  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
30  // Fonction                 : derivated_source                        //
31  // ..................................................................//
32  // Role                     : definit la forme fonctionnelle du champ //
33  //                          : incident                                //
34  // ..................................................................//
35  // Valeur renvoyee          : derivee d'un champ electrique           //
36  // Parametres d'appel       : distance de la source au pt d'evaluation//
37  //                          : variables d'environnement               //
38  // Parametres modifies      : aucun                                   //
39  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
40  typ_champ derivated_source(typ_distance my_dist, var_env_comp env)
41   {
42     typ_coeff betadt= env.beta * env.dt;
```

```
43      typ_coeff tau    = env.time - my_dist/C;
44      typ_coeff resu   = 0.0;
45
46      if(( tau >= 0.0) && (tau <= env.period))
47              resu =  exp ( - env.alpha * ( tau - betadt)* ( tau - betadt))*(-2.0) * env.alpha * (
tau - betadt);
48      return (resu);
49    }
50
51  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
52  // Fonctions                          : distx
                    //
53  //                                          : disty
                           //
54  //                                          : distz
                           //
55  // ..................................................................... //
56  // Role                               : renvoient la distance a considerer du pt//
57  //                                          : d'evaluation a la source
            //
58  // ..................................................................... //
59  // Valeur renvoyee           : distance
            //
60  // Parametres d'appel        : coordonnees du point
      //
61  //                                          : variables d'environnement
            //
62  // Parametres modifies       : aucun
            //
63  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
64  inline typ_distance distx(int i,int j,int k, var_env_comp env)
65  {
66       return  ( env.dx * env.xdisp * ((i-1)  + 0.5 * env.offset )
67                       + env.dy * env.ydisp * (j-1)
68                       + env.dz * env.zdisp * (k-1)
69                       + env.delay );
70  }
71  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
72  inline typ_distance disty(int i,int j,int k, var_env_comp env)
73  {
74       return  ( env.dx * env.xdisp * (i-1)
75                       + env.dy * env.ydisp * ((j-1) + 0.5 * env.offset )
76                       + env.dz * env.zdisp * (k-1)
77                       + env.delay );
78  }
79
80  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
81  inline typ_distance distz(int i,int j,int k, var_env_comp env)
82  {
83     return   ( env.dx * env.xdisp * (i-1)
84                      + env.dy * env.ydisp * (j-1)
85                      + env.dz * env.zdisp * ((k-1) + 0.5 * env.offset )
86                      + env.delay );
87  }
88
89  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
90  // Fonctions                          : incident_fieldX
            //
91  //                                          : incident_fieldY
```

```
                    //
92  //                                          : incident_fieldZ
                    //
93  // ------------------------------------------------------------ //
94  // Role                            : Retournent les composantes en X, Y  et Z//
95  //                                          : du champ electrique incident en un point//
96  // ------------------------------------------------------------ //
97  // Valeur renvoyee          : champ electrique
        //
98  // Parametres d'appel       : coordonnees du point
        //
99  //                                          : variables d'environnement
            //
100 // Parametres modifies      : aucun
            //
101 // :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
102 inline typ_champ incident_fieldX( int i,int j,int k, var_env_comp env)
103  {
104     typ_distance     my_dist  = distx(i,j,k,env);
105     return   (env.ampx * source (my_dist, env));
106  }
107 // :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
108 inline typ_champ incident_fieldY ( int i, int j, int k, var_env_comp env)
109  {
110     typ_distance     my_dist  = disty(i,j,k,env);
111     return   (env.ampy * source (my_dist,env));
112  }
113 // :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
114 inline typ_champ incident_fieldZ (   int      i, int j, int k, var_env_comp env)
115 {
116     typ_distance     my_dist  = distz(i,j,k,env);
117     return   (env.ampz * source (my_dist,env));
118 }
119
120 // :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
121 // Fonctions                        : derivated_incident_fieldX
        //
122 //                                          : derivated_incident_fieldY
            //
123 //                                          : derivated_incident_fieldZ
            //
124 // ------------------------------------------------------------ //
125 // Role                             : Retournent les derivees des composantes //
126 //                                          : en X, Y et Z du champ electrique incident //
127 //                                          : en un point
                    //
128 // ------------------------------------------------------------ //
129 // Valeur renvoyee          : champ electrique
        //
130 // Parametres d'appel       : coordonnees du point
        //
131 //                                          : variables d'environnement
            //
132 // Parametres modifies      : aucun
            //
133 // :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
134 inline typ_champ derivated_incident_fieldX(int i,int j,int k, var_env_comp env)
135 {
136     typ_distance     my_dist  = distx(i,j,k,env);
```

```
137     return   (env.ampx * derivated_source(my_dist, env));
138  }
139  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
140  inline typ_champ derivated_incident_fieldY(int i,int j,int k,var_env_comp env)
141  {
142     typ_distance     my_dist = disty(i,j,k,env);
143     return   (env.ampy * derivated_source(my_dist, env));
144   }
145  // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
146  inline typ_champ derivated_incident_fieldZ(int i,int j,int k,var_env_comp env)
147  {
148     typ_distance     my_dist = distz(i,j,k,env);
149     return   (env.ampz * derivated_source(my_dist, env));
150  }
```

ei_fields.h

```
1     typ_champ incident_fieldX(int i,int j,int k)
2     {
3         typ_champ frite=0;
4         return (i*j*k*frite);
5     }
6
7     typ_champ incident_fieldY(int i,int j,int k)
8     {
9         typ_champ frite=0;
10        return (i*j*k*frite);
11    }
12
13    typ_champ incident_fieldZ(int i,int j,int k)
14    {
15        typ_champ frite=0;
16        return (i*j*k*frite);
17    }
18
19   typ_champ derivated_incident_fieldX(int i,int j,int k)
20   {
21        typ_champ frite=0;
22        return (i*j*k*frite);
23    }
24   typ_champ derivated_incident_fieldY(int i,int j,int k)
25   {
26        typ_champ frite=0;
27        return (i*j*k*frite);
28    }
29   typ_champ derivated_incident_fieldZ(int i,int j,int k)
30   {
31        typ_champ frite=0;
32        return (i*j*k*frite);
33    }
```

Electrode.h

```
1   ////////////////////////////////////////////////////////////////////////////
2   // Electrode                                                              //
3   //------------------------------------------------------------------------//
4   // Cette classe englobe toutes les fonctions necessaires a la             //
5   // sauvegarde des donnees des electrodes.                                 //
6   // Le format de fichier qui est genere par cette classe est MATLAB.       //
7   //------------------------------------------------------------------------//
8   // Auteur:                                  Luc Romain                    //
9   // Date:                                    15 septembre 1998             //
10  // Derniere modification:   Aucune                                        //
11  ////////////////////////////////////////////////////////////////////////////
12
13  class Electrode
14  {
15  public:
16
17      // Constructeurs
18      Electrode(void)
19      {
20      };
21
22      // Test de conductivite
23
24      // Construction a partir d'un fichier dans le repertoire courant.
25      Electrode(espace_champs *EFLD,espace_physique *PhysSpace)
26      {
27              BasicFileIO file("","elec_specs.ini","r");
28              cx=file.read_int();
29              cy=file.read_int();
30              cz=file.read_int();
31              pas=file.read_int();
32              hauteur=file.read_int();
33              rayon=file.read_int();
34              type_electrode=file.read_int();
35              index=file.read_int();
36              printf("------- Construction de l'electrode avec "ElecSpecs.ini"");
37              printf("------- E. Specs: (%i,%i,%i), R=%i, H=%i, P=%i, T=%i, I=%i",
38                      cx,cy,cz,rayon,hauteur,pas,type_electrode,index);
39
40              pspace=PhysSpace;
41              efld=EFLD;
42              init();
43      };
44
45      Electrode(int X,int Y,int Z,int Pas,int Hauteur,int Rayon,
46                      int Type,int mat_index,
47                      espace_champs *EFLD,espace_physique *PhysSpace)
48      {
49              cx=X;
50              cy=Y;
51              cz=Z;
```

```
52              pas=Pas;
53              hauteur=Hauteur;
54              rayon=Rayon;
55              type_electrode=Type;
56              index=mat_index;
57              pspace=PhysSpace;
58              efld=EFLD;
59              init();
60      };
61
62      ~Electrode(void)
63      {
64      };
65
66
67  protected:
68
69      int cx,cy,cz;     // axe de l'electrode
70      int hauteur;      // hauteur de l'electrode helicoidale; pas pour bipolaire.
71      int pas;          // Espacement minimal entre deux fils.
72      int rayon;
73      int type_electrode;
74      int index;
75      espace_champs *efld;
76      espace_physique *pspace;
77
78  /* ------------------------------------------------------------------
79   * void init(void)
                        *
80   * ------------------------------------------------------------------*
81   * Cette fonction place des morceaux de metal dans l'espace selon une*
82   * geometrie helicoidale.
                *
83   * ------------------------------------------------------------------*/
84      void init(void)
85      {
86
87              switch(type_electrode)
88              {
89              case 0:
90                      build_helicoide();
91                      break;
92              case 1:
93                      build_bipolaire();
94                      break;
95              }
96
97      };
98
99  public:
100
101 /* ------------------------------------------------------------------
102  *  Fonction delta_e(...)
                *
103  * ------------------------------------------------------------------*
104  * Cette fonction calcule la difference de potentiel entre les deux  *
105  * fils de l'electrode en suivant un parcours 1.                      *
106  * ------------------------------------------------------------------*
107  * Luc Romain, Janvier 1999
```

```
108   * ----------------------------------------------------------------*/
109
110   typ_coeff delta_e(void)
111      {
112               int i;
113               cellule_champs cell;
114               typ_champ val;
115
116               val=0;
117
118               switch(type_electrode)
119               {
120               case 0:  // Calcul pour l'electrode helicoidale
121                                 for(i=cz-2*pas;i<cz;i++)
122                                 {
123                                          cell=efld->get(cx,cy+rayon,i);
124                                          val+=cell.cz();
125                                 }
126                                 break;
127               case 1: // Calcul pour l'electrode bipolaire
128                                 for(i=cz-pas;i<cz;i++)
129                                 {
130                                          cell=efld->get(cx,cy+rayon,i);
131                                          val+=cell.cz();
132                                 }
133                                 break;
134               }
135
136               return(val);
137      };
138
139   /* ----------------------------------------------------------------
140    * Fonction delta_eb(...)
141    * ----------------------------------------------------------------*
142    * Cette fonction calcule la difference de potentiel entre les deux  *
143    * fils de l'electrode en suivant un parcours 2.                      *
144    * ----------------------------------------------------------------*
145    * Luc Romain, Janvier 1999
146    * ----------------------------------------------------------------*/
147
148   typ_coeff delta_eb(void)
149      {
150               int i;
151               cellule_champs cell;
152               typ_champ val;
153
154               val=0;
155
156               switch(type_electrode)
157               {
158               case 0:  // Calcul pour l'electrode helicoidale
159                                 //1ere methode:
160                                 if(hauteur<4*pas)
161                                 {
162                                          printf("'electrode trop courte;");
163                                          printf("(...) invalide. Electrode::Eb()");
```

```
164                             }
165
166                             for(i=cz-4*pas;i<cz-2*pas;i++)
167                             {
168                                     cell=efld->get(cx,cy-rayon,i);
169                                     val+=cell.cz();
170                             }
171                             // 2e methode:
172                             /*for(i=cz-2*pas;i<cz;i++)
173                             {
174                                     cell=efld->get(cx,cy-rayon,i);
175                                     val+=cell.cz();
176                             }*/
177                             break;
178             case 1: // Calcul pour l'electrode bipolaire
179                             for(i=cz-pas;i<cz;i++)
180                             {
181                                     cell=efld->get(cx,cy-rayon,i);
182                                     val+=cell.cz();
183                             }
184                             break;
185             }
186
187             return(val);
188     };
189
190
191 protected:
192 /* ------------------------------------------------------------------
193  * Fonction wire_x
194  * ------------------------------------------------------------------*
195  * Cette fonction trace un fil de xi a xf selon l'axe x.
196  * ------------------------------------------------------------------*
197  * Luc Romain, Janvier 1999
198  * ------------------------------------------------------------------*/
199     void wire_x(int xi,int xf,int y,int z)
200     {
201             cellule_physique cell;
202             int i,tmp;
203
204             if(xi>xf)
205             {
206                     tmp=xi;
207                     xi=xf;
208                     xf=tmp;
209             }
210
211             for(i=xi;i<xf;i++)
212             {
213                     cell=pspace->get(i,y,z);
214                     cell.setx(index);
215                     pspace->put(cell,i,y,z);
216             }
217     };
218 /* ------------------------------------------------------------------
219  * Fonction wire_y
```

```
220   * ------------------------------------------------------------------*
221   * Cette fonction trace un fil de yi a yf selon l'axe y.                *
222   * ------------------------------------------------------------------*
223   * Luc Romain, Mars 1999
                *
224   * ------------------------------------------------------------------*/
225     void wire_y(int x,int yi,int yf,int z)
226     {
227             cellule_physique cell;
228             int j,tmp;
229
230             if(yi>yf)
231             {
232                     tmp=yi;
233                     yi=yf;
234                     yf=tmp;
235             }
236
237             for(j=yi;j<yf;j++)
238             {
239                     cell=pspace->get(x,j,z);
240                     cell.sety(index);
241                     pspace->put(cell,x,j,z);
242             }
243     };
244
245   /* ------------------------------------------------------------------
246    * Fonction wire_z
                      *
247    * ------------------------------------------------------------------*
248    * Cette fonction trace un fil de zi a zf selon l'axe z.                *
249    * ------------------------------------------------------------------*
250    * Luc Romain, Mars 1999
                *
251    * ------------------------------------------------------------------*/
252     void wire_z(int x,int y,int zi,int zf)
253     {
254             cellule_physique cell;
255             int k,tmp;
256
257             if(zi>zf)
258             {
259                     tmp=zi;
260                     zi=zf;
261                     zf=zi;
262             }
263
264             for(k=zi;k<zf;k++)
265             {
266                     cell=pspace->get(x,y,k);
267                     cell.setz(index);
268                     pspace->put(cell,x,y,k);
269             }
270     };
271
272   /* ------------------------------------------------------------------
273    * Fonction build_helicoide()
                *
274    * ------------------------------------------------------------------*
```

```
275   * Cette fonction construit une electrode helicoidale.                    *
276   * ------------------------------------------------------------------*
277   * Luc Romain, Mars 1999
                 *
278   * ------------------------------------------------------------------*/
279
280     void build_helicoide(void)
281     {
282             int k;
283
284             k=cz;
285
286             while(k>cz-hauteur)
287             {
288                     wire_x(cx-rayon,cx+rayon,cy-rayon,k);
289                     wire_x(cx-rayon,cx+rayon,cy+rayon,k);
290                     wire_z(cx+rayon,cy+rayon,k-pas,k);
291                     wire_z(cx-rayon,cy-rayon,k-pas,k);
292                     k-=pas;
293                     wire_y(cx-rayon,cy-rayon,cy+rayon,k);
294                     wire_y(cx+rayon,cy-rayon,cy+rayon,k);
295                     wire_z(cx-rayon,cy+rayon,k-pas,k);
296                     wire_z(cx+rayon,cy-rayon,k-pas,k);
297                     k-=pas;
298             }
299     };
300   /* ------------------------------------------------------------------
301    * Fonction build_bipolaire()
                  *
302    * ------------------------------------------------------------------*
303    * Cette fonction construit une electrode helicoidale.                    *
304    * ------------------------------------------------------------------*
305    * Luc Romain, Mars 1999
                 *
306    * ------------------------------------------------------------------*/
307
308     void build_bipolaire(void)
309     {
310             int k;
311
312             k=cz;
313
314             wire_x(cx-rayon,cx+rayon,cy-rayon,k);
315             wire_x(cx-rayon,cx+rayon,cy+rayon,k);
316             wire_y(cx-rayon,cy-rayon,cy+rayon,k);
317             wire_y(cx+rayon,cy-rayon,cy+rayon,k);
318             k-=pas;
319             wire_x(cx-rayon,cx+rayon,cy-rayon,k);
320             wire_x(cx-rayon,cx+rayon,cy+rayon,k);
321             wire_y(cx-rayon,cy-rayon,cy+rayon,k);
322             wire_y(cx+rayon,cy-rayon,cy+rayon,k);
323
324
325     };
326
327   };
```

elem3D.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                          elem3D.H                                  //
4   //          DESCRIPTION ET MISE EN OEUVRE DE LA CLASSE 'elem3D'       //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   //                                                                    //
8   // La classe elem3D permet de figurer des objets dotes de             //
9   // 3 caracteristiques selon les 3 dimensions de l'espace              //
10  // C'est une classe generique : le type de ces caracteristiques       //
11  // (le meme pour les 3 dimensions) est defini a l'instantiation       //
12  // On l'utilisera pour stocker les valeurs des champs dans une        //
13  // cellule de l'espace, ou les valeurs des parametres physiques       //
14  // qui caracterisent cette cellule.                                   //
15  //                                                                    //
16  ////////////////////////////////////////////////////////////////////////
17
18  template <class type>
19  class elem3D
20  {
21  private:
22      type    Cx;             // Composante en x du elem3D considere
23      type    Cy;             // Composante en y du elem3D considere
24      type    Cz;             // Composante en z du elem3D considere
25
26  public:
27      // Constructeur
28      elem3D( type ValX=VAL_IN_ELEM3D_NULLE,
29                   type ValY=VAL_IN_ELEM3D_NULLE,
30                   type ValZ=VAL_IN_ELEM3D_NULLE)
31      {
32              Cx=(type)ValX;
33              Cy=(type)ValY;
34              Cz=(type)ValZ;
35      };
36
37      // Destructeur
38      virtual ~elem3D()       {};
39
40      // Accesseurs
41      inline type     cx()    {return (Cx);};
42      inline type     cy()    {return (Cy);};
43      inline type     cz()    {return (Cz);};
44
45      // Operateurs d'affectation
46      inline void setx(type ValX)       {Cx=ValX;}
47      inline void sety(type ValY)       {Cy=ValY;}
48      inline void setz(type ValZ)       {Cz=ValZ;}
49
50      // Affichage d'un element, variable selon le type
```

```
51      void show_values();
52      void show_valx();
53      void show_valy();
54      void show_valz();
55
56  };
57
58  // Procedures d'affichage
59
60  inline void elem3D<long double>::show_values()
61      {printf(" *cx: %g*cy: %gf*cz: %g",Cx,Cy,Cz);};
62  inline void elem3D<float>::show_values()
63      {printf(" *cx: %.6f*cy: %.6f*cz: %.6f",Cx,Cy,Cz);};
64  inline void elem3D<int>::show_values()
65      {printf(" *cx: %d*cy: %d*cz: %d",Cx,Cy,Cz);};
66  inline void elem3D<long double>::show_valx()
67      {printf("  [cx: %g] ",Cx);};
68  inline void elem3D<long double>::show_valy()
69      {printf("  [cy: %g] ",Cy);};
70  inline void elem3D<long double>::show_valz()
71      {printf("  [cz: %g] ",Cz);};
72  inline void elem3D<double>::show_valx()
73      {printf("  [cx: %g] ",Cx);};
74  inline void elem3D<double>::show_valy()
75      {printf("  [cy: %g] ",Cy);};
76  inline void elem3D<double>::show_valz()
77      {printf("  [cz: %g] ",Cz);};
78  inline void elem3D<float>::show_valx()
79      {printf("  [cx: %f] ",Cx);};
80  inline void elem3D<float>::show_valy()
81      {printf("  [cy: %f] ",Cy);};
82  inline void elem3D<float>::show_valz()
83      {printf("  [cz: %f] ",Cz);};
84  inline void elem3D<int>::show_valx()
85      {printf("  [cx: %d] ",Cx);};
86  inline void elem3D<int>::show_valy()
87      {printf("  [cy: %d] ",Cy);};
88  inline void elem3D<int>::show_valz()
89      {printf("  [cz: %d] ",Cz);};
90
91
```

es_field.0.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                          ESFIELDS.H                                //
4   //                  MODULE DE CALCUL DES CHAMPS ELECTRIQUES 'SCATTERED'                //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7
8   template <class material_computing>
9   void electric_scattered_fields       ( espace_champs                  elcfields,
10                                                                        espace_champs
        magfields,
11                                                                        espace_physique
        my_space,
12                                                                        material_computing
        material_table[],
13                                                                        var_env_comp
        my_env)
14  {
15  typ_champ                   new_valx, new_valy, new_valz;
16  cellule_champs              my_cell;
17  typ_material                material_index;
18  material_computing  my_material;
19
20  int i, j, k;
21
22  int NX1= my_space.nx() -1 ;
23  int NY1= my_space.ny() -1 ;
24  int NZ1= my_space.nz() -1 ;
25
26  /* ---------------------- Composante en X ----------------------- */
27  //printf(" Composante en x");
28  //getchar();
29  for (k=2; k<=NZ1 ;k++)
30      for (j=2; j<= NY1; j++)
31              for (i=1; i<= NX1; i++)
32              {
33                      material_index=(my_space.get(i,j,k)).cx();
34                      my_material = material_table[material_index];
35                      my_cell = elcfields.get(i,j,k);
36                      switch (material_index)
37                      {
38
39                      case 0:          // Free space
40                              {
41                              new_value=       my_cell.cx()
42                                                      + my_material.dtedy() *
(magfields.get(i,j,k).cz() + magfields.get(i,j-1,k).cz())
43                                                      + my_material.dtedz() *
(magfields.get(i,j,k).cz() + magfields.get(i,j-1,k).cz());
44                              break;
45                              }
46
47                      case 1:          // Perfect conductor
```

```
48                              {
49                              new_value = (- incident_fieldX(i,j,k,my_env));
50                              break;
51                              }
52                      default:// Dielectrics
53                              {
54                              new_value= my_material.esctc()  * my_cell.cx()
55                                                      -  my_material.eincc()  *
incident_fieldX(i,j,k,my_env)
56                                                      -  my_material.edevcn() *
derivated_incident_fieldX(i,j,k,my_env)
57                                                      +  my_material.ecrly()  * (magfields.get(i,j,k).cz()
- magfields.get(i,j-1,k).cz())
58                                                      +  my_material.ecrlz()  * (magfields.get(i,j,k).cy()
- magfields.get(i,j,k-1).cy());
59                              break;
60                              }
61                      }
62                      my_cell.setx(new_value);
63                      elcfields.put(my_cell,i,j,k);
64              }
65
66   // ---------------------- Composante en Y ----------------------- //
67   //printf(" Composante en y");
68   //getchar();
69
70   for (k=2; k<= NZ1; k++)
71       for (j=1; j<= NY1; j++)
72               for (i=2; i<= NX1; i++)
73               {
74                       material_index=(my_space.get(i,j,k)).cy();
75                       my_material = material_table[material_index];
76                       my_cell = elcfields.get(i,j,k);
77                       switch (material_index)
78                       {
79                               case 0:         // Free space
80                                       {
81                                       new_value=      my_cell.cy()
82                                                               + my_material.dtedz()
*(magfields.get(i,j,k).cx() + magfields.get(i,j,k-1).cx())
83                                                               + my_material.dtedx()
*(magfields.get(i,j,k).cz() + magfields.get(i-1,j,k).cz());
84                                       break;
85                                       }
86                               case 1:         // Perfect conductor
87                                       {
88                                       new_value = (- incident_fieldY(i,j,k,my_env));
89                                       break;
90                                       }
91                               default:// Dielectrics
92                                       {
93                                       new_value= my_material.esctc()  * my_cell.cy()
94                                                       -  my_material.eincc()  *
incident_fieldY(i,j,k,my_env)
95                                                       -  my_material.edevcn() *
derivated_incident_fieldY(i,j,k,my_env)
96                                                       +  my_material.ecrlz()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j,k-1).cx())
97                                                       +  my_material.ecrlx()  *
```

```
(magfields.get(i,j,k).cz() - magfields.get(i-1,j,k).cz());
98                                          break;
99                                          }
100                         }
101                         my_cell.sety(new_value);
102                         elcfields.put(my_cell,i,j,k);
103             }
104
105 // ---------------------- Composante en Z ------------------------ //
106 //printf(" Composante en z");
107 //getchar();
108
109 for (k=1; k<= NZ1; k++)
110     for (j=2; j<= NY1; j++)
111             for (i=2; i<= NX1; i++)
112             {
113                         material_index=(my_space.get(i,j,k)).cz();
114                         my_material = material_table[material_index];
115                         my_cell = elcfields.get(i,j,k);
116                         switch (material_index)
117                         {
118                                 case 0:          // Free space
119                                         {
120                                         new_value=       my_cell.cz()
121                                                                 + my_material.dtedx()
*(magfields.get(i,j,k).cy() + magfields.get(i-1,j,k).cy())
122                                                                 + my_material.dtedy()
**(magfields.get(i,j,k).cx() + magfields.get(i,j-1,k).cx());
123                                         break;
124                                         }
125                                 case 1:          // Perfect conductor
126                                         {
127                                         new_value = (- incident_fieldZ(i,j,k,my_env));
128                                         break;
129                                         }
130                                 default: // Dielectrics
131                                         {
132                                         new_value= my_material.esctc()  * my_cell.cz()
133                                                                 -  my_material.eincc()  *
incident_fieldZ(i,j,k,my_env)
134                                                                 -  my_material.edevcn() *
derivated_incident_fieldZ(i,j,k,my_env)
135                                                                 +  my_material.ecrlz()  *
(magfields.get(i,j,k).cy() - magfields.get(i-1,j,k).cy())
136                                                                 +  my_material.ecrlx()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j-1,k).cx());
137                                         break;
138                                         }
139                         }
140                         my_cell.setz(new_value);
141                         elcfields.put(my_cell,i,j,k);
142             }
143 }
```

Es_field.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                          ESFIELDS.H                                //
4   //                MODULE DE CALCUL DES CHAMPS ELECTRIQUES DIFFUSES    //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   ////////////////////////////////////////////////////////////////////////
8   // Procedure                    : electric_scattered_fields2          //
9   // ...................................................................//
10  // Role                         : Mise a jour des champs electriques dans //
11  //                                : tout l'espace considere           //
12  // ...................................................................//
13  // Valeur renvoyee              : aucune                              //
14  // Parametres d'appel           : espace des champs electriques       //
15  //                                : espace des champs magnetiques     //
16  //                                : espace physique                   //
17  //                                : table des materiaux               //
18  //                                : variables d'environnement         //
19  // Parametres modifies          : espace des champs electriques       //
20  // ...................................................................//
21  // Remarques :          Proposition d'amelioration par rapport au code de //
22  //                      Luebbers.                                     //
23  ////////////////////////////////////////////////////////////////////////
24  template <class material_computing>
25  void electric_scattered_fields2      ( espace_champs                  elcfields,
26                                                                        espace_champs
        magfields,
27                                                                        espace_physique
        my_space,
28                                                                        material_computing
        material_table[],
29                                                                        var_env_comp
        my_env)
30  {
31  typ_champ                   new_value;
32  cellule_champs              my_cell;
33  cellule_physique    space_cell;
34  typ_material                material_index;
35  material_computing  my_material;
36
37  int i, j, k;
38
39  int NX1 = my_space.nx() - 1 ;
40  int NY1 = my_space.ny() - 1 ;
41  int NZ1 = my_space.nz() - 1 ;
```

```
42
43   for (k=1; k<=NZ1 ;k++)
44      for (j=1; j<= NY1; j++)
45               for (i=1; i<= NX1; i++)
46               {
47                        my_cell                  = elcfields.get(i,j,k);
48                        space_cell               = my_space.get(i,j,k);
49                        //---------------------- Composante en X ------------------------ //
50                        if (k>1 && j>1)
51                        {
52                                 material_index   = space_cell.cx();
53                                 my_material              = material_table[material_index];
54                                 switch (material_index)
55                                 {
56                                 case 0:          // Free space
57                                          {
58                                          new_value=       my_cell.cx()
59                                                                    + my_material.dtedy() *
(magfields.get(i,j,k).cz() - magfields.get(i,j-1,k).cz())
60                                                                    - my_material.dtedz() *
(magfields.get(i,j,k).cy() - magfields.get(i,j,k-1).cy());
61                                          break;
62                                          }
63
64                                 case 1:          // Perfect conductor
65                                          {
66                                          new_value = (- incident_fieldX(i,j,k,my_env));
67                                          break;
68                                          }
69                                 case 2:          // Erreur probable...LR
70                                          {
71                                          new_value = my_cell.cx();
72                                          break;
73                                          }
74                                 default: // Dielectrics
75                                          {
76                                          new_value= my_material.esctc()  * my_cell.cx()
77                                                                    -  my_material.eincc()  *
incident_fieldX(i,j,k,my_env)
78                                                                    -  my_material.edevcn() *
derivated_incident_fieldX(i,j,k,my_env)
79                                                                    +  my_material.ecrly()  *
(magfields.get(i,j,k).cz() - magfields.get(i,j-1,k).cz())
80                                                                    -  my_material.ecrlz()  *
(magfields.get(i,j,k).cy() - magfields.get(i,j,k-1).cy());
81                                          break;
82                                          }
83                                 }
84                                 my_cell.setx(new_value);
85                        }
86                        // ---------------------- Composante en Y ------------------------ //
87                        if (k>1 && i>1)
88                        {
89                                 material_index=space_cell.cy();
90                                 my_material = material_table[material_index];
91                                 switch (material_index)
92                                 {
93                                          case 0:          // Free space
94                                                   {
```

```
95                                              new_value=      my_cell.cy()
96                                                      + my_material.dtedz() *
(magfields.get(i,j,k).cx() - magfields.get(i,j,k-1).cx())
97                                                      - my_material.dtedx() *
(magfields.get(i,j,k).cz() - magfields.get(i-1,j,k).cz());
98                                              break;
99                                              }
100                                     case 1:         // Perfect conductor
101                                             {
102                                             new_value = (- incident_fieldY(i,j,k,my_env));
103                                             break;
104                                             }
105
106                                     case 2:         // Antenne -- non recalcule.
107                                     {
108                                     new_value = my_cell.cy();
109                                     break;
110                                     }
111
112                                     default: // Dielectrics
113                                             {
114                                             new_value= my_material.esctc()  * my_cell.cy()
115                                                             - my_material.eincc()  *
incident_fieldY(i,j,k,my_env)
116                                                             - my_material.edevcn() *
derivated_incident_fieldY(i,j,k,my_env)
117                                                             + my_material.ecrlz()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j,k-1).cx())
118                                                             - my_material.ecrlx()  *
(magfields.get(i,j,k).cz() - magfields.get(i-1,j,k).cz());
119                                             break;
120                                             }
121                             }
122                             my_cell.sety(new_value);
123                     }
124                     // ---------------------- Composante en Z ----------------------- //
125                     if (i>1 && j>1)
126                     {
127                             material_index=space_cell.cz();
128                             my_material = material_table[material_index];
129                             switch (material_index)
130                             {
131                                     case 0:         // Free space
132                                             {
133                                             new_value=      my_cell.cz()
134                                                             + my_material.dtedx() *
(magfields.get(i,j,k).cy() - magfields.get(i-1,j,k).cy())
135                                                             - my_material.dtedy() *
(magfields.get(i,j,k).cx() - magfields.get(i,j-1,k).cx());
136                                             break;
137                                             }
138                                     case 1:         // Perfect conductor
139                                             {
140                                             new_value = (- incident_fieldZ(i,j,k,my_env));
141                                             break;
142                                             }
143
144                                     case 2:         // Antenne -- non recalcule.
145                                             {
```

```
146                                             new_value = my_cell.cz();
147                                             break;
148                                             }
149
150                                     default: // Dielectrics
151                                             {
152                                             new_value= my_material.esctc()  * my_cell.cz()
153                                                             -  my_material.eincc()  *
incident_fieldZ(i,j,k,my_env)
154                                                             -  my_material.edevcn() *
derivated_incident_fieldZ(i,j,k,my_env)
155                                                             +  my_material.ecrlx()  *
(magfields.get(i,j,k).cy() - magfields.get(i-1,j,k).cy())
156                                                             -  my_material.ecrly()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j-1,k).cx());
157                                             break;
158                                             }
159                                 }
160                                 my_cell.setz(new_value);
161                         }
162
163                         elcfields.put(my_cell,i,j,k);
164                 }
165     }
166
167     ////////////////////////////////////////////////////////////////////////
168     // Procedure                        : electric_scattered_fields
        //
169     // ..................................................................... //
170     // Role                             : Mise a jour des champs electriques dans //
171     //                                  : tout l'espace considere
        //
172     // ..................................................................... //
173     // Valeur renvoyee          : aucune
        //
174     // Parametres d'appel       : espace des champs electriques                          //
175     //                                  : espace des champs magnetiques
        //
176     //                                  : espace physique
                        //
177     //                                  : table des materiaux
                //
178     //                                  : variables d'environnement
                //
179     // Parametres modifies      : espace des champs electriques                          //
180     ////////////////////////////////////////////////////////////////////////
181     template <class material_computing>
182     void electric_scattered_fields      ( espace_champs                  elcfields,
183                                                                     espace_champs
        magfields,
184                                                                     espace_physique
        my_space,
185                                                                     material_computing
        material_table[],
186                                                                     var_env_comp
        my_env)
187     {
188     typ_champ                   new_value;
189     cellule_champs              my_cell;
```

```
190  typ_material                material_index;
191  material_computing  my_material;
192
193  int i, j, k;
194
195  int NX1 = my_space.nx() - 1 ;
196  int NY1 = my_space.ny() - 1 ;
197  int NZ1 = my_space.nz() - 1 ;
198
199
200  /* ---------------------- Composante en X ---------------------- */
201  for (k=2; k<=NZ1 ;k++)
202     for (j=2; j<= NY1; j++)
203              for (i=1; i<= NX1; i++)
204              {
205                      my_cell                  = elcfields.get(i,j,k);
206                      material_index   = (my_space.get(i,j,k)).cx();
207                      my_material              = material_table[material_index];
208                      switch (material_index)
209                      {
210                      case 0:          // Free space
211                              {
212                              new_value=       my_cell.cx()
213                                                      + my_material.dtedy() *
(magfields.get(i,j,k).cz() - magfields.get(i,j-1,k).cz())
214                                                      - my_material.dtedz() *
(magfields.get(i,j,k).cy() - magfields.get(i,j,k-1).cy());
215                              break;
216                              }
217
218                      case 1:          // Perfect conductor
219                              {
220                              new_value = (- incident_fieldX(i,j,k,my_env));
221                              break;
222                              }
223                      case 2:     // Antenne
224                              {
225                              new_value=my_cell.cx();
226                              break;   // Antenne -- Luc Romain, 4 sept 1998
227                              }
228                      default: // Dielectrics
229                              {
230                              new_value= my_material.esctc()  * my_cell.cx()
231                                                      -  my_material.eincc()  *
incident_fieldX(i,j,k,my_env)
232                                                      -  my_material.edevcn() *
derivated_incident_fieldX(i,j,k,my_env)
233                                                      +  my_material.ecrly()  * (magfields.get(i,j,k).cz()
- magfields.get(i,j-1,k).cz())
234                                                      -  my_material.ecrlz()  * (magfields.get(i,j,k).cy()
- magfields.get(i,j,k-1).cy());
235                              break;
236                              }
237                      }
238                      my_cell.setx(new_value);
239                      elcfields.put(my_cell,i,j,k);
240              }
241
242  // ---------------------- Composante en Y ---------------------- //
```

```
243  for (k=2; k<= NZ1; k++)
244      for (j=1; j<= NY1; j++)
245              for (i=2; i<= NX1; i++)
246              {
247                      material_index=(my_space.get(i,j,k)).cy();
248                      my_material = material_table[material_index];
249                      my_cell = elcfields.get(i,j,k);
250                      switch (material_index)
251                      {
252                              case 0:          // Free space
253                                      {
254                                              new_value=       my_cell.cy()
255                                                      + my_material.dtedz() *
(magfields.get(i,j,k).cx() - magfields.get(i,j,k-1).cx())
256                                                      - my_material.dtedx() *
(magfields.get(i,j,k).cz() - magfields.get(i-1,j,k).cz());
257
258                                      break;
259                                      }
260                              case 1:          // Perfect conductor
261                                      {
262                                      new_value = (- incident_fieldY(i,j,k,my_env));
263                                      break;
264                                      }
265                              case 2:    // Antenne
266                                      {
267                                      new_value=my_cell.cy();
268                                      break;  // Antenne -- Luc Romain 4 sept 1998
269                                      }
270                              default: // Dielectrics
271                                      {
272                                      new_value= my_material.esctc()  * my_cell.cy()
273                                                      - my_material.eincc()  *
incident_fieldY(i,j,k,my_env)
274                                                      - my_material.edevcn() *
derivated_incident_fieldY(i,j,k,my_env)
275                                                      + my_material.ecrlz()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j,k-1).cx())
276                                                      - my_material.ecrlx()  *
(magfields.get(i,j,k).cz() - magfields.get(i-1,j,k).cz());
277                                      break;
278                                      }
279                      }
280                      my_cell.sety(new_value);
281                      elcfields.put(my_cell,i,j,k);
282              }
283
284  // ---------------------- Composante en Z ------------------------ //
285  for (k=1; k<= NZ1; k++)
286      for (j=2; j<= NY1; j++)
287              for (i=2; i<= NX1; i++)
288              {
289                      material_index=(my_space.get(i,j,k)).cz();
290                      my_material = material_table[material_index];
291                      my_cell = elcfields.get(i,j,k);
292                      switch (material_index)
293                      {
294                              case 0:          // Free space
295                                      {
```

```
296                                     new_value=       my_cell.cz()
297                                             + my_material.dtedx() * (magfields.get(i,j,k).cy() -
magfields.get(i-1,j,k).cy())
298                                             - my_material.dtedy() * (magfields.get(i,j,k).cx() -
magfields.get(i,j-1,k).cx());
299                                     break;
300                                     }
301                             case 1:         // Perfect conductor
302                                     {
303                                     new_value = (- incident_fieldZ(i,j,k,my_env));
304                                     break;
305                                     }
306                             case 2:    // Antenne
307                                     {
308                                     new_value=my_cell.cz();
309                                     break;  // Antenne -- Luc Romain, 4 sept 1998
310                                     }
311                             default: // Dielectrics
312                                     {
313                                     new_value= my_material.esctc()  * my_cell.cz()
314                                                     -  my_material.eincc()  *
incident_fieldZ(i,j,k,my_env)
315                                                     -  my_material.edevcn() *
derivated_incident_fieldZ(i,j,k,my_env)
316                                                     +  my_material.ecrlx()  *
(magfields.get(i,j,k).cy() - magfields.get(i-1,j,k).cy())
317                                                     -  my_material.ecrly()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j-1,k).cx());
318                                     break;
319                                     }
320                     }
321                     my_cell.setz(new_value);
322                     elcfields.put(my_cell,i,j,k);
323             }
324  }
```

es_field.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                          ESFIELDS.H                                //
4   //                MODULE DE CALCUL DES CHAMPS ELECTRIQUES DIFFUSES    //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   ////////////////////////////////////////////////////////////////////////
8   // Procedure                    : electric_scattered_fields2          //
9   // ...................................................................//
10  // Role                         : Mise a jour des champs electriques dans //
11  //                                : tout l'espace considere           //
12  // ...................................................................//
13  // Valeur renvoyee              : aucune                              //
14  // Parametres d'appel           : espace des champs electriques       //
15  //                                : espace des champs magnetiques     //
16  //                                : espace physique                   //
17  //                                : table des materiaux               //
18  //                                : variables d'environnement         //
19  // Parametres modifies          : espace des champs electriques       //
20  // ...................................................................//
21  // Remarques :          Proposition d'amelioration par rapport au code de //
22  //                      Luebbers.                                     //
23  ////////////////////////////////////////////////////////////////////////
24  template <class material_computing>
25  void electric_scattered_fields2      ( espace_champs                 elcfields,
26                                                                       espace_champs
        magfields,
27                                                                       espace_physique
        my_space,
28                                                                       material_computing
        material_table[],
29                                                                       var_env_comp
        my_env)
30  {
31  typ_champ                   new_value;
32  cellule_champs              my_cell;
33  cellule_physique    space_cell;
34  typ_material                material_index;
35  material_computing  my_material;
36
37  int i, j, k;
38
39  int NX1 = my_space.nx() - 1 ;
40  int NY1 = my_space.ny() - 1 ;
41  int NZ1 = my_space.nz() - 1 ;
```

```
42
43   for (k=1; k<=NZ1 ;k++)
44      for (j=1; j<= NY1; j++)
45              for (i=1; i<= NX1; i++)
46              {
47                      my_cell                 = elcfields.get(i,j,k);
48                      space_cell              = my_space.get(i,j,k);
49                      //---------------------- Composante en X ----------------------- //
50                      if (k>1 && j>1)
51                      {
52                              material_index   = space_cell.cx();
53                              my_material              = material_table[material_index];
54                              switch (material_index)
55                              {
56                              case 0:          // Free space
57                                      {
58                                      new_value=       my_cell.cx()
59                                                              + my_material.dtedy() *
(magfields.get(i,j,k).cz() - magfields.get(i,j-1,k).cz())
60                                                              - my_material.dtedz() *
(magfields.get(i,j,k).cy() - magfields.get(i,j,k-1).cy());
61                                      break;
62                                      }
63
64                              case 1:          // Perfect conductor
65                                      {
66                                      new_value = (- incident_fieldX(i,j,k,my_env));
67                                      break;
68                                      }
69                              case 2:          // Erreur probable...LR
70                                      {
71                                      new_value = my_cell.cx();
72                                      break;
73                                      }
74                              default:// Dielectrics
75                                      {
76                                      new_value= my_material.esctc()  *my_cell.cx()
77                                                              -  my_material.eincc()  *
incident_fieldX(i,j,k,my_env)
78                                                              -  my_material.edevcn() *
derivated_incident_fieldX(i,j,k,my_env)
79                                                              +  my_material.ecrly()  *
(magfields.get(i,j,k).cz() - magfields.get(i,j-1,k).cz())
80                                                              -  my_material.ecrlz()  *
(magfields.get(i,j,k).cy() - magfields.get(i,j,k-1).cy());
81                                      break;
82                                      }
83                              }
84                              my_cell.setx(new_value);
85                      }
86                      // ---------------------- Composante en Y ----------------------- //
87                      if (k>1 && i>1)
88                      {
89                              material_index=space_cell.cy();
90                              my_material = material_table[material_index];
91                              switch (material_index)
92                              {
93                                      case 0:          // Free space
94                                              {
```

```
95                                              new_value=      my_cell.cy()
96                                                              + my_material.dtedz() *
(magfields.get(i,j,k).cx() - magfields.get(i,j,k-1).cx())
97                                                              - my_material.dtedx() *
(magfields.get(i,j,k).cz() - magfields.get(i-1,j,k).cz());
98                                              break;
99                                              }
100                                     case 1:         // Perfect conductor
101                                             {
102                                             new_value = (- incident_fieldY(i,j,k,my_env));
103                                             break;
104                                             }
105
106                                     case 2:         // Antenne -- non recalcule.
107                                     {
108                                     new_value = my_cell.cy();
109                                     break;
110                                     }
111
112                                     default:// Dielectrics
113                                             {
114                                             new_value= my_material.esctc()  * my_cell.cy()
115                                                             - my_material.eincc()  *
incident_fieldY(i,j,k,my_env)
116                                                             - my_material.edevcn() *
derivated_incident_fieldY(i,j,k,my_env)
117                                                             + my_material.ecrlz()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j,k-1).cx())
118                                                             - my_material.ecrlx()  *
(magfields.get(i,j,k).cz() - magfields.get(i-1,j,k).cz());
119                                             break;
120                                             }
121                             }
122                             my_cell.sety(new_value);
123                     }
124                     // ---------------------- Composante en Z ---------------------- //
125                     if (i>1 && j>1)
126                     {
127                             material_index=space_cell.cz();
128                             my_material = material_table[material_index];
129                             switch (material_index)
130                             {
131                                     case 0:         // Free space
132                                             {
133                                             new_value=      my_cell.cz()
134                                                             + my_material.dtedx() *
(magfields.get(i,j,k).cy() - magfields.get(i-1,j,k).cy())
135                                                             - my_material.dtedy() *
(magfields.get(i,j,k).cx() - magfields.get(i,j-1,k).cx());
136                                             break;
137                                             }
138                                     case 1:         // Perfect conductor
139                                             {
140                                             new_value = (- incident_fieldZ(i,j,k,my_env));
141                                             break;
142                                             }
143
144                                     case 2:         // Antenne -- non recalcule.
145                                             {
```

```
146                                             new_value = my_cell.cz();
147                                             break;
148                                             }
149
150                                     default: // Dielectrics
151                                             {
152                                             new_value= my_material.esctc()  * my_cell.cz()
153                                                             -  my_material.eincc()  *
incident_fieldZ(i,j,k,my_env)
154                                                             -  my_material.edevcn() *
derivated_incident_fieldZ(i,j,k,my_env)
155                                                             +  my_material.ecrlx()  *
(magfields.get(i,j,k).cy() - magfields.get(i-1,j,k).cy())
156                                                             -  my_material.ecrly()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j-1,k).cx());
157                                             break;
158                                             }
159                                 }
160                                 my_cell.setz(new_value);
161                         }
162
163                         elcfields.put(my_cell,i,j,k);
164                 }
165 }
166
167 ////////////////////////////////////////////////////////////////////////
168 // Procedure                    : electric_scattered_fields
    //
169 // ..................................................................... //
170 // Role                         : Mise a jour des champs electriques dans //
171 //                                : tout l'espace considere
    //
172 // ..................................................................... //
173 // Valeur renvoyee          : aucune
    //
174 // Parametres d'appel       : espace des champs electriques                          //
175 //                                : espace des champs magnetiques
    //
176 //                                : espace physique
                        //
177 //                                : table des materiaux
                //
178 //                                : variables d'environnement
                //
179 // Parametres modifies      : espace des champs electriques                          //
180 ////////////////////////////////////////////////////////////////////////
181 template <class material_computing>
182 void electric_scattered_fields       ( espace_champs                 elcfields,
183                                                                 espace_champs
    magfields,
184                                                                 espace_physique
    my_space,
185                                                                 material_computing
    material_table[],
186                                                                 var_env_comp
    my_env)
187 {
188 typ_champ                   new_value;
189 cellule_champs              my_cell;
```

```
190  typ_material                material_index;
191  material_computing  my_material;
192
193  int i, j, k;
194
195  int NX1 = my_space.nx() - 1 ;
196  int NY1 = my_space.ny() - 1 ;
197  int NZ1 = my_space.nz() - 1 ;
198
199
200  /* ---------------------- Composante en X ----------------------- */
201  for (k=2; k<=NZ1 ;k++)
202      for (j=2; j<= NY1; j++)
203              for (i=1; i<= NX1; i++)
204              {
205                      my_cell                  = elcfields.get(i,j,k);
206                      material_index   = (my_space.get(i,j,k)).cx();
207                      my_material              = material_table[material_index];
208                      switch (material_index)
209                      {
210                      case 0:          // Free space
211                              {
212                              new_value=       my_cell.cx()
213                                                      + my_material.dtedy() *
(magfields.get(i,j,k).cz() - magfields.get(i,j-1,k).cz())
214                                                      - my_material.dtedz() *
(magfields.get(i,j,k).cy() - magfields.get(i,j,k-1).cy());
215                              break;
216                              }
217
218                      case 1:          // Perfect conductor
219                              {
220                              new_value = (- incident_fieldX(i,j,k,my_env));
221                              break;
222                              }
223                      case 2:     // Antenne
224                              {
225                              new_value=my_cell.cx();
226                              break;  // Antenne -- Luc Romain, 4 sept 1998
227                              }
228                      default: // Dielectrics
229                              {
230                              new_value= my_material.esctc()  * my_cell.cx()
231                                                      -  my_material.eincc()  *
incident_fieldX(i,j,k,my_env)
232                                                      -  my_material.edevcn() *
derivated_incident_fieldX(i,j,k,my_env)
233                                                      +  my_material.ecrly()  * (magfields.get(i,j,k).cz()
- magfields.get(i,j-1,k).cz())
234                                                      -  my_material.ecrlz()  * (magfields.get(i,j,k).cy()
- magfields.get(i,j,k-1).cy());
235                              break;
236                              }
237                      }
238                      my_cell.setx(new_value);
239                      elcfields.put(my_cell,i,j,k);
240              }
241
242  // ---------------------- Composante en Y ----------------------- //
```

```
243  for (k=2; k<= NZ1; k++)
244      for (j=1; j<= NY1; j++)
245              for (i=2; i<= NX1; i++)
246              {
247                      material_index=(my_space.get(i,j,k)).cy();
248                      my_material = material_table[material_index];
249                      my_cell = elcfields.get(i,j,k);
250                      switch (material_index)
251                      {
252                              case 0:          // Free space
253                                      {
254                                              new_value=       my_cell.cy()
255                                                      + my_material.dtedz() *
(magfields.get(i,j,k).cx() - magfields.get(i,j,k-1).cx())
256                                                      - my_material.dtedx() *
(magfields.get(i,j,k).cz() - magfields.get(i-1,j,k).cz());
257
258                                      break;
259                                      }
260                              case 1:          // Perfect conductor
261                                      {
262                                      new_value = (- incident_fieldY(i,j,k,my_env));
263                                      break;
264                                      }
265                              case 2:    // Antenne
266                                      {
267                                      new_value=my_cell.cy();
268                                      break;   // Antenne -- Luc Romain 4 sept 1998
269                                      }
270                              default: // Dielectrics
271                                      {
272                                      new_value= my_material.esctc()   * my_cell.cy()
273                                                      - my_material.eincc()  *
incident_fieldY(i,j,k,my_env)
274                                                      - my_material.edevcn() *
derivated_incident_fieldY(i,j,k,my_env)
275                                                      + my_material.ecrlz()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j,k-1).cx())
276                                                      - my_material.ecrlx()  *
(magfields.get(i,j,k).cz() - magfields.get(i-1,j,k).cz());
277                                      break;
278                                      }
279                      }
280                      my_cell.sety(new_value);
281                      elcfields.put(my_cell,i,j,k);
282              }
283
284  // ---------------------- Composante en Z ------------------------ //
285  for (k=1; k<= NZ1; k++)
286      for (j=2; j<= NY1; j++)
287              for (i=2; i<= NX1; i++)
288              {
289                      material_index=(my_space.get(i,j,k)).cz();
290                      my_material = material_table[material_index];
291                      my_cell = elcfields.get(i,j,k);
292                      switch (material_index)
293                      {
294                              case 0:          // Free space
295                                      {
```

```
296                                     new_value=       my_cell.cz()
297                                             + my_material.dtedx() * (magfields.get(i,j,k).cy() -
magfields.get(i-1,j,k).cy())
298                                             - my_material.dtedy() * (magfields.get(i,j,k).cx() -
magfields.get(i,j-1,k).cx());
299                                     break;
300                                     }
301                             case 1:         // Perfect conductor
302                                     {
303                                     new_value = (- incident_fieldZ(i,j,k,my_env));
304                                     break;
305                                     }
306                             case 2:    // Antenne
307                                     {
308                                     new_value=my_cell.cz();
309                                     break;  // Antenne -- Luc Romain, 4 sept 1998
310                                     }
311                             default: // Dielectrics
312                                     {
313                                     new_value= my_material.esctc()  * my_cell.cz()
314                                                     -  my_material.eincc()  *
incident_fieldZ(i,j,k,my_env)
315                                                     -  my_material.edevcn() *
derivated_incident_fieldZ(i,j,k,my_env)
316                                                     +  my_material.ecrlx()  *
(magfields.get(i,j,k).cy() - magfields.get(i-1,j,k).cy())
317                                                     -  my_material.ecrly()  *
(magfields.get(i,j,k).cx() - magfields.get(i,j-1,k).cx());
318                                     break;
319                                     }
320                     }
321                     my_cell.setz(new_value);
322                     elcfields.put(my_cell,i,j,k);
323             }
324  }
```

espaces3.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                espaces3.H                          //
4   //                          DESCRIPTION DE LA CLASSE 'espaces'        //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   //                                                                    //
8   // La classe espaces3 derive de la classe tableaux, et permet donc    //
9   // de representer un espace 3D divise en cellules, dont chacune       //
10  // possede des caracteristiques qui seront   stockees dans un         //
11  // "tableaux".                                                        //
12  // Elle se dote en outre des coordonnees origine de cet espace        //
13  // Il s'agit d'une classe generique, dont l'instantiation fixera      //
14  // le type (elem3D dans notre cas) de ces caracteristiques.           //
15  //                                                                    //
16  ////////////////////////////////////////////////////////////////////////
17
18  template <class type>
19  class espaces: public tableaux<type>
20  {
21  private     :
22     // Point origine
23     int OX,OY,OZ;
24
25  public:
26     // Constructeur
27     espaces(int PosX=0, int PosY=0, int PosZ=0,
28                     int Dimx=1, int Dimy=1, int Dimz=1) : tableaux<type>(Dimx,Dimy,Dimz)
29             {OX=PosX;OY=PosY;OZ=PosZ;};
30
31     // Destructeur
32     virtual          ~espaces()       {};
33
34     // Accesseurs
35     inline int       ox()     {return(OX);};
36     inline int       oy()     {return(OY);};
37     inline int       oz()     {return(OZ);};
38
39     // Affichage des caracteristiques de l'objet
40     void     descripteur()
41     {
42             printf(" Point origine de la zone:");
43             printf(" *OX: %d*OY: %d*OZ: %d",OX,OY,OZ);
44             tableaux<type>::descripteur();
45     };
46  };
```

hs_fields.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //
                                        //
3   //                                          HSFIELDS.H
                        //
4   //          DESCRIPTION ET MISE EN OEUVRE DE LA CLASSE 'MATERIAL'              //
5   //
                                        //
6   ////////////////////////////////////////////////////////////////////////
7   ////////////////////////////////////////////////////////////////////////
8   // Procedure                        : magnetic_scattered_fields2
       //
9   // ..................................................................... //
10  // Role                             : Mise a jour des champs magnetiques dans //
11  //                                          : tout l'espace considere
       //
12  // ..................................................................... //
13  // Valeur renvoyee          : aucune
       //
14  // Parametres d'appel       : espace des champs electriques                          //
15  //                                          : espace des champs magnetiques
       //
16  //                                          : espace physique
                        //
17  //                                          : materiau: vide
                //
18  //                                          : variables d'environnement
                //
19  // Parametres modifies      : espace des champs magnetiques                          //
20  // ..................................................................... //
21  // Remarques :      Proposition d'amelioration par rapport au code de //
22  //                          Luebbers.
                                //
23  //                          Gain sur l'espace de test et 100 iterations : 10 a       //
24  //                          15 secondes. -
                        //
25  ////////////////////////////////////////////////////////////////////////
26
27  template<class material_computing>
28  void magnetic_scattered_fields2(espace_champs         elcfields,
29                                                  espace_champs           magfields,
30                                                  espace_physique         my_space,
31                                                  material_computing
       free_space)
32  {
33  int                         i, j, k;
34  typ_champ           new_value;
35  cellule_champs      my_cell;
36  int NX1=magfields.nx()-1;
37  int NY1=magfields.ny()-1;
38  int NZ1=magfields.nz()-1;
39
40  for (k=1; k<= NZ1;k++)
41     for (j=1; j<=NY1; j++)
42              for (i=1; i<= NX1; i++)
43              {
```

```
44                      my_cell   = magfields.get(i,j,k);
45                      if (i>1)  // --------- Composante en X --------- //
46                      {
47                              new_value =     my_cell.cx()
48                                              - free_space.dtmdy() * (elcfields.get(i,j+1,k).cz()
- elcfields.get(i,j,k).cz())
49                                              + free_space.dtmdz() * (elcfields.get(i,j,k+1).cy()
- elcfields.get(i,j,k).cy());
50                              my_cell.setx(new_value);
51                      }
52
53                      if (j>1)  // --------- Composante en Y --------- //
54                      {
55                              new_value =     my_cell.cy()
56                                              - free_space.dtmdz() * (elcfields.get(i,j,k+1).cx()
- elcfields.get(i,j,k).cx())
57                                              + free_space.dtmdx() * (elcfields.get(i+1,j,k).cz()
- elcfields.get(i,j,k).cz());
58                              my_cell.sety(new_value);
59                      }
60                      if (k>1)  // --------- Composante en Z --------- //
61                      {
62                              new_value =     my_cell.cz()
63                                              - free_space.dtmdx() * (elcfields.get(i+1,j,k).cy()
- elcfields.get(i,j,k).cy())
64                                              + free_space.dtmdy() * (elcfields.get(i,j+1,k).cx()
- elcfields.get(i,j,k).cx());
65                              my_cell.setz(new_value);
66                      }
67                      magfields.put(my_cell,i,j,k);
68              }
69      }
70
71      ////////////////////////////////////////////////////////////////////////
72      // Procedure                    : magnetic_scattered_fields
        //
73      // ..................................................................... //
74      // Role                         : Mise a jour des champs magnetiques dans //
75      //                                      : tout l'espace considere
        //
76      // ..................................................................... //
77      // Valeur renvoyee          : aucune
        //
78      // Parametres d'appel       : espace des champs electriques                                 //
79      //                                      : espace des champs magnetiques
        //
80      //                                      : espace physique
                        //
81      //                                      : materiau: vide
                //
82      //                                      : variables d'environnement
                //
83      // Parametres modifies      : espace des champs magnetiques                                 //
84      ////////////////////////////////////////////////////////////////////////
85
86      template<class material_computing>
87      void magnetic_scattered_fields(     espace_champs           elcfields,
88                                                                  espace_champs           magfields,
89                                                                  espace_physique         my_space,
```

```
90                                                                material_computing
      free_space)
91    {
92    int                          i, j, k;
93    typ_champ            new_value;
94    cellule_champs       my_cell;
95    int NX1=magfields.nx()-1;
96    int NY1=magfields.ny()-1;
97    int NZ1=magfields.nz()-1;
98
99    /* ----------------------- Composante en X ------------------------ */
100   for (k=1; k<= NZ1;k++)
101      for (j=1; j<=NY1; j++)
102               for (i=2; i<= NX1; i++)
103               {
104                        my_cell    = magfields.get(i,j,k);
105                        new_value =      my_cell.cx()
106                                                 - free_space.dtmdy() * (elcfields.get(i,j+1,k).cz()
- elcfields.get(i,j,k).cz())
107                                                 + free_space.dtmdz() * (elcfields.get(i,j,k+1).cy()
- elcfields.get(i,j,k).cy());
108                        my_cell.setx(new_value);
109                        magfields.put(my_cell,i,j,k);
110               }
111   /* ----------------------- Composante en Y ------------------------ */
112   for (k=1; k<= NZ1;k++)
113      for (j=2; j<=NY1; j++)
114               for (i=1; i<= NX1; i++)
115               {
116                        my_cell    = magfields.get(i,j,k);
117                        new_value =      my_cell.cy()
118                                                 - free_space.dtmdz() * (elcfields.get(i,j,k+1).cx()
- elcfields.get(i,j,k).cx())
119                                                 + free_space.dtmdx() * (elcfields.get(i+1,j,k).cz()
- elcfields.get(i,j,k).cz());
120                        my_cell.sety(new_value);
121                        magfields.put(my_cell,i,j,k);
122               }
123   /* ----------------------- Composante en Z ------------------------ */
124   for (k=2; k<= NZ1;k++)
125      for (j=1; j<= NY1; j++)
126               for (i=1; i<= NX1; i++)
127               {
128                        my_cell    = magfields.get(i,j,k);
129                        new_value =      my_cell.cz()
130                                                 - free_space.dtmdx() * (elcfields.get(i+1,j,k).cy()
- elcfields.get(i,j,k).cy())
131                                                 + free_space.dtmdy() * (elcfields.get(i,j+1,k).cx()
- elcfields.get(i,j,k).cx());
132                        my_cell.setz(new_value);
133                        magfields.put(my_cell,i,j,k);
134               }
135   }
```

LChain.h

```
1   /*
2      classe Liste
3
4     Cette classe represente un element contenu dans une liste pointee simple.
5     Les acces a l'element se font avec Set et Get.
6
7     L'element suivant est pointe par *next.
8
9     Date: 3 mai 1999
10    Luc Romain
11
12    */
13  template <class type>
14  class Liste
15  {
16  public:
17     Liste()
18     {
19             next=NULL;
20     }
21     ~Liste()
22     {
23     };
24
25  // ---------------------------------------------------------------------
26  // Set (....)
27  // Assignation de la valeur contenue dans element.
28  //
29  // Luc Romain
30  // ---------------------------------------------------------------------
31     void Set(type *new_elem)
32     {
33             element=*new_elem;
34     }
35  // ---------------------------------------------------------------------
36  // Get (....)
37  // Retour de la valeur contenue dans element.
38  //
39  // Luc Romain
40  // ---------------------------------------------------------------------
41
42     type Get()
43     {
44             return(element);
45     }
46
47  // ---------------------------------------------------------------------
48  // SetNext (....)
49  // Assigne l'adresse du prochain element.
50  //
51  // Luc Romain
52  // ---------------------------------------------------------------------
53
54     void SetNext(Liste *Next)
55     {
56             next=Next;
```

```
57      }
58
59  // ---------------------------------------------------------------------
60  // GetNext (....)
61  // Retourne l'adresse du prochain element.
62  //
63  // Luc Romain
64  // ---------------------------------------------------------------------
65      Liste* GetNext(void)
66      {
67              return(next);
68      }
69
70  private:
71      Liste *next;
72      type element;
73  };
74
75
76  // ---------------------------------------------------------------------
77  // class Linked_Chain
78  //
79  // Cette classe s'occupe de lier tous les elements qui seront crees
80  //
81  // Luc Romain
82  // ---------------------------------------------------------------------
83
84  template <class type>
85  class Linked_Chain
86  {
87  protected:
88      Liste<type> *head,*tail,*courant;
89      bool initOK;
90      int nb_elem;
91
92  // ---------------------------------------------------------------------
93  // delete_list()
94  // Efface completement le contenu de la liste chainee.
95  // Luc Romain
96  // ---------------------------------------------------------------------
97      void delete_list()
98      {
99              Liste<type> *tmp;
100
101             courant=head;
102             while(courant!=NULL)
103             {
104                     tmp=courant;
105                     courant=courant->GetNext();
106                     delete tmp;
107             }
108
109
110     }
111
112 public:
113     Linked_Chain()
114     {
115             head=NULL;
```

```
116             tail=NULL;
117             courant=NULL;
118             nb_elem=0;
119             initOK=false;
120     }
121
122     ~Linked_Chain()
123     {
124             delete_list();
125     }
126
127 // ----------------------------------------------------------------------
128 // Add_elem(...)
129 // Rajoute un element au bout de la liste chainee.
130 //
131 // Luc Romain
132 // ----------------------------------------------------------------------
133
134     bool Add_elem(type *elem)
135     {
136             Liste<type> *tmp;
137             if(initOK)
138             {
139                     tmp=new Liste<type>;
140                     if(tmp==NULL)
141                     {
142                             printf("d'allocation de memoire -- ListedChain::Add.");
143                             initOK=false;
144                             return false;
145                     }
146                     tmp->SetNext(NULL);
147                     tmp->Set(elem); // Assignation du pointeur sur element.
148                     tail->SetNext(tmp);
149                     tail=tmp;
150                     nb_elem++;
151             }
152             else // initOK=false
153             {
154                     tmp=new Liste<type>;
155                     if(tmp==NULL)
156                     {
157                             printf("d'allocation de memoire -- ListedChain::Add.");
158                             return false;
159                     }
160                     tmp->SetNext(NULL);
161                     tmp->Set(elem);
162                     head=tmp;
163                     tail=tmp;
164                     courant=tmp;
165                     nb_elem++;
166                     initOK=true;
167                     //printf("element memorise.");
168             }
169             return(true);
170     }
171
172 // ----------------------------------------------------------------------
173 // top()
174 //
```

```
175  // Place le pointeur de la liste sur le premier element puis retourne
176  // l'element situe a la premiere place de cette liste.
177  // Luc Romain
178  // ------------------------------------------------------------------------
179
180      type top()
181      {
182              if(!initOK)
183              {
184                      printf("de lecture. ListedChain::top()");
185                      if(get_nb_elem()==0)
186                              printf("element dans la liste.");
187                      exit(1);
188              }
189              courant=head;
190              if(courant!=NULL)
191              {
192                      return courant->Get();
193              }
194              else
195              {
196                      printf("'element n'existe pas. LinkedChain::top()");
197                      exit(1);
198              }
199              return courant->Get();
200      }
201
202  // ------------------------------------------------------------------------
203  // current()
204  // Renvoie l'element qui est actuellement pointe par le pointeur de
205  // position.
206  // Luc Romain
207  // ------------------------------------------------------------------------
208      type current()
209      {
210              if(!initOK)
211              {
212                      printf("de lecture. LChain::current().");
213                      exit(1);
214              }
215              if(courant!=NULL)
216                      return(courant->Get());
217              else
218              {
219                      printf("'element n'existe pas. LinkedChain::current()");
220                      exit(1);
221              }
222              return courant->Get();
223      }
224
225  // ------------------------------------------------------------------------
226  // next()
227  // Avance le pointeur d'un element puis renvoie l'element
228  //
229  // Luc Romain
230  // ------------------------------------------------------------------------
231      type next()
232      {
233              if(!initOK)
```

```
234             {
235                     printf("de lecture. LChain::next().");
236                     exit(1);
237             }
238
239             if(courant->GetNext()!=NULL)
240             {
241                     courant=courant->GetNext();
242                     return(courant->Get());
243             }
244             else
245             {
246                     printf("- L'element n'existe pas ou la fin de la liste a ete rencontree.");
247                     printf("- LinkedChain::next()");
248                     return(current());
249             }
250     }
251
252
253 // ---------------------------------------------------------------------
254 // get_nb_elem()
255 // renvoie le nombre total d'elements qui ont ete alloues.
256 // Luc Romain
257 // ---------------------------------------------------------------------
258     int get_nb_elem()
259     {
260             return nb_elem;
261     }
262
263 };
264
```

list.txt

```
1    main.cpp
2    basic_file_io.h
3    build.h
4    coeur.h
5    constantes.h
6    diaphragme2.h
7    dipole.h
8    display_for_debug.h
9    ei_fields.0.h
10   ei_fields.h
11   Electrode.h
12   elem3D.h
13   es_field.0.h
14   Es_field.h
15   es_field.h
16   espaces3.h
17   hs_fields.h
18   LChain.h
19   list.txt
20   material.h
21   material_computing.h
22   material_table.h
23   orbc_comp_const.h
24   Orbc_xy.h
25   Orbc_xz.h
26   Orbc_yx.h
27   Orbc_yz.h
28   Orbc_zx.h
29   Orbc_zy.h
30   ordi.h
31   restore.h
32   save_data.h
33   save_space.h
34   Source.h
35   tableaux.h
36   type.h
37   type2.h
38   update.h
39   var_env.h
40   var_env_comp.h
41   wiretest.h
```

material.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                          MATERIAL.H                                //
4   //          DESCRIPTION ET MISE EN OEUVRE DE LA CLASSE 'MATERIAL'     //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   //                                                                    //
8   // La classe material permet de definir un materiau, en stockant     //
9   //          * son nom                                                 //
10  //          * l'index qui le reference dans la table des materiaux    //
11  //          * sa permittivite epsilon                                 //
12  //          * sa conductivite sigma                                   //
13  //          *                                  rho                    //
14  //                                                                    //
15  //  Elle n'est jamais utilisee directement : les modules de calcul    //
16  // font appel a la classe "material_computing".                       //
17  // C'est une classe derivee de material, qui contient en outre        //
18  // les constantes multiplicatives fonction de ces parametres et       //
19  // effectivement utilisees dans les calculs                           //
20  //                                                                    //
21  ////////////////////////////////////////////////////////////////////////
22
23  class material
24  {
25  protected:
26      char                    nom[40];
27      int                     index;
28      typ_phys_prm    sigma;
29      typ_phys_prm    epsilon;
30      typ_phys_prm    rho;
31      bool                    initok;
32
33  public:
34      // Constructeur
35      material()
36      {
37              initok=false;
38      }
39
40      material(char *name,    typ_phys_prm sig = 0.0,
41                      typ_phys_prm eps =EPS0,         typ_phys_prm rh  =PHYS_PRM_NUL)
42              {init(name,sig,eps,rh);};
43
44      // Destructeur
45      virtual ~material()             {};
46
```

```
47      // Initialisation
48      void init(       char *name,typ_phys_prm  sig,
49                               typ_phys_prm     eps,typ_phys_prm rh)
50      {
51              sprintf(nom,"%s",name);
52              sigma=sig;
53              epsilon=eps;
54              rho=rh;
55      }
56
57      // Affectation de l'index referent dans la table des materiaux
58      void set_index(int ind)           {index=ind;}
59  };
```

material_computing.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //
                                        //
3   //                                          MATERIAL_COMPUTING.H
                  //
4   //
                                        //
5   ////////////////////////////////////////////////////////////////////////
6   class material_computing  : public material, public var_env
7   {
8   private :
9
10      // Constantes de calcul associees (calcul des champs )
11      typ_phys_prm     C1;
12      typ_phys_prm     C2;
13      typ_phys_prm     C3;
14      typ_phys_prm     C4;
15      typ_phys_prm     C5;
16      typ_phys_prm     C6;
17
18  public:
19      // Constructeurs
20
21      material_computing()      {initok=false;};
22
23      material_computing(var_env env):var_env (env)
24              {if (dx*dy*dz*dt==0.0) initok=false;};
25
26      material_computing(       char *name,typ_phys_prm  sig  ,
27                                                typ_phys_prm     eps ,typ_phys_prm         rh    ,
28                                                var_env                  env ):
material(name,sig,eps,rh), var_env (env)
29      {
30              if (dx*dy*dz*dt==0.0)
31                      {
32                      printf("Une des variables d'environnement est nulle.");
33                      printf("L'initialisation des constantes de calcul ne peut etre menee a
bien.");
34                      getchar();
35                      initok=false;
36                      }
37              else    {init();initok=true;}
38      };
39
40      material_computing(       int Indx,char *name,typ_phys_prm sig,
41                                                typ_phys_prm eps,typ_phys_prm     rh    ,
42                                                var_env  env ): material(name,sig,eps,rh),var_env
(env)
43      {
44              if (dx*dy*dz*dt==0.0)
45              {
46                      printf("Une des variables d'environnement est nulle.");
47                      printf("L'initialisation des constantes de calcul ne peut etre menee a
bien.");
48                      getchar();
49                      initok=false;
```

```
50              }
51              else
52              {
53                      set_index(Indx);
54                      init();
55                      initok=true;
56              }
57      };
58
59
60      // Destructeur
61      ~material_computing()     {};
62
63      // Initialisation
64      void init()
65      {
66
67              if (index == 0 || epsilon*sigma == 0)      // Free space
68              {
69              C1 = dt / (EPS0 * dx);                                          // dtedx
70              C2 = dt / (EPS0 * dy);                                          // dtedy
71              C3 = dt / (EPS0 * dz);                                          // dtedz
72              C4 = dt / (XMU0 * dx);                                          // dmtdx
73              C5 = dt / (XMU0 * dy);                                          // dmtdy
74              C6 = dt / (XMU0 * dz);                                          // dmtdz
75              }
76              else if (index!=1)                // Dielectrics
77              {
78              C1 = epsilon / (epsilon + sigma * dt);                                  // esctc
79              C2 = sigma * dt/ (epsilon + sigma * dt);                        //eincc
80              C3 = dt * (epsilon - EPS0) / (epsilon + sigma * dt);//edevcn
81              C4 = dt / ((epsilon + sigma * dt) * dx);                        //ecrlx
82              C5 = dt / ((epsilon + sigma * dt) * dy);                        //ecrly
83              C6 = dt / ((epsilon + sigma * dt) * dz);                        //ecrlz
84              printf("------- Material[%2i]->"%19s" EPSr->%2.2e Sig-
>%3.3e",index,nom,epsilon,sigma);
85              }
86      initok=true;
87      }
88
89      // Accesseurs : cas du vide
90      inline typ_phys_prm       dtedx()
91              {       if (index == 0 && initok) return (C1);
92                      else {irrelevant_call("vide");return (PHYS_PRM_NUL);}}
93
94      inline typ_phys_prm       dtedy()
95              {       if (index == 0 && initok) return (C2);
96                      else {irrelevant_call("vide");return (PHYS_PRM_NUL);}}
97
98      inline typ_phys_prm       dtedz()
99              {       if (index == 0 && initok) return (C3);
100                     else {irrelevant_call("vide");return (PHYS_PRM_NUL);}}
101
102     inline typ_phys_prm       dtmdx()
103             {       if (index == 0 && initok) return (C4);
104                     else {irrelevant_call("vide");return (PHYS_PRM_NUL);}}
105
106     inline typ_phys_prm       dtmdy()
107             {       if (index== 0 && initok)  return (C5);
```

```
108                     else {irrelevant_call("vide");return (PHYS_PRM_NUL);}}
109
110     inline typ_phys_prm       dtmdz()
111             {       if (index == 0 && initok)return (C6);
112                     else {irrelevant_call("vide");return (PHYS_PRM_NUL);}}
113
114     // Accesseurs : cas des dielectriques
115     inline typ_phys_prm       esctc()
116             {       if (index> 1 && initok)  return (C1);
117                     else {irrelevant_call("dielectrique");return (PHYS_PRM_NUL);}}
118
119     inline typ_phys_prm       eincc()
120             {       if (index> 1 && initok)  return (C2);
121                     else {irrelevant_call("dielectrique");return (PHYS_PRM_NUL);}}
122
123     inline typ_phys_prm       edevcn()
124             {       if (index> 1 && initok)  return (C3);
125                     else {irrelevant_call("dielectrique");return (PHYS_PRM_NUL);}}
126
127     inline typ_phys_prm       ecrlx()
128             {       if (index> 1 && initok)  return (C4);
129                     else {irrelevant_call("dielectrique");return (PHYS_PRM_NUL);}}
130
131     inline typ_phys_prm       ecrly()
132             {       if (index> 1 && initok)  return (C5);
133                     else {irrelevant_call("dielectrique");return (PHYS_PRM_NUL);}}
134
135     inline typ_phys_prm       ecrlz()
136             {       if (index> 1 && initok)  return (C6);
137                     else {irrelevant_call("dielectrique");return (PHYS_PRM_NUL);}}
138
139     void description()
140     {
141
142             printf("_computing (%s) -- Index (%d)",nom,index);
143             printf("= %4.4e",epsilon);
144             printf("= %4.4f",sigma);
145             printf("--> %d",initok);
146             printf("= %4.4e",C1);
147             printf("= %4.4e",C2);
148             printf("= %4.4e",C3);
149             printf("= %4.4e",C4);
150             printf("= %4.4e",C5);
151             printf("= %4.4e",C6);
152             getchar();
153
154     }
155     void set(int Indx,char *name,typ_phys_prm sig,
156                     typ_phys_prm eps,typ_phys_prm     rh    .
157                     var_env  env )
158     {
159             dx=env.dx;
160             dy=env.dy;
161             dz=env.dz;
162             dt=env.dt;
163             // Appel des constructeurs.
164             material::init(name,sig,eps,rh);
165
166             epsilon=eps;
```

```
167             sigma=sig;
168
169             if (dx*dy*dz*dt==0.0)
170             {
171                     printf("Une des variables d'environnement est nulle.");
172                     printf("L'initialisation des constantes de calcul ne peut etre menee a
bien.");
173                     getchar();
174                     initok=false;
175             }
176             else
177             {
178                     set_index(Indx);
179                     init();
180                     initok=true;
181             }
182     };
183
184  private:
185     // Message d'erreur en cas d'appel hors de propos
186     void irrelevant_call(char *mat)
187             {
188             if (!initok){    printf(" Defaut d'initialisation des constantes de %s ",mat);
189                                     printf(" Valeur renvoyee sans signification -
Index=%i",index);}
190             else             {       printf(" Irrelevant access to  %s constants ",mat);
191                                     printf(" Valeur renvoyee sans signification");}
192     }
193  };
```

material_table.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                MATERIAL_TABLE.H    //
4   //                                                                    //
5   ////////////////////////////////////////////////////////////////////////
6   // Gestion de la table des materiaux                                  //
7   ////////////////////////////////////////////////////////////////////////
8
9
10  void set_material_table(material_computing my_table[], var_env env)
11  {
12      static material_computing vide     (0,"VIDE"          ,0.0   ,EPS0    ,0.0, env);      // mat 0
13      static material_computing mat2     (2,"Muscle Emetteur",0.05,EPS0,0.0, env);
14      static material_computing mat3     (3,"Thorax" ,0.01 ,EPS0,0.5, env);
15      static material_computing mat4     (4,"Metal (Electrode)" ,30000,EPS0        ,0.5, env);
16      static material_computing mat5     (5,"Air", 0.0001 ,EPS0,0.5, env);
17      static material_computing mat6     (6,"Contour du thorax", 0.005 ,EPS0,0.5, env);
18      static material_computing mat7     (7,"Coeur inactif",0.5,EPS0,0.0, env);
19      static material_computing mat8     (8,"Diaphragme inactif",1,EPS0,0.0, env);
20      static material_computing mat9     (9,"Oesophage",0.02,EPS0,0.0, env);
21      static material_computing mat10    (10,"Eau",0.05,EPS0,0.0, env);
22
23      // Les constructeurs ci-haut initialisent les materiaux avec le bon index.
24      // Ne pas toucher a ces initialisations...
25
26      my_table[0]=vide;
27      my_table[2]=mat2;
28      my_table[3]=mat3;
29      my_table[4]=mat4;
30      my_table[5]=mat5;
31      my_table[6]=mat6;
32      my_table[7]=mat7;
33      my_table[8]=mat8;
34      my_table[9]=mat9;
35      my_table[10]=mat10;
36
37  }
38
39
40  // Cette fonction initialise les materiaux a partir du fichier
41  // material.ini dans le repertoire courant.
42
43  void set_material_table_file(material_computing my_table[], var_env env)
44  {
45      int nbmat,i;
46      typ_coeff sigma;
47      typ_coeff epsilonr;
48      char vide_mat[]="VIDE0";
49      char defrep[]="";
50      char filename[]="material.ini0";
51      BasicFileIO matfile(defrep,filename,"r");
52      material_computing vide(0,vide_mat,0.0,EPS0,0.0, env);
```

```
53      char matname[24];
54      char line_buff[40],f_val[40];
55      StrAnlz MatFileParam;
56
57      my_table[0]=vide;
58
59  //  nbmat=matfile.read_int();
60      // Lecture du nombre de materiaux:
61      matfile.get_line(line_buff,39);
62      MatFileParam.sets(line_buff);
63      MatFileParam.gets(f_val);
64      nbmat=atoi(f_val);
65      //printf("de materiaux: %i",nbmat);
66
67      printf("------- Opening material description file");
68      for(i=2;i<nbmat+2;i++)
69      {
70              //matfile.read_string(matname);
71              //sigma=matfile.read_coeff();
72              //epsilonr=matfile.read_coeff();
73
74              // Lecture des parametres:
75              matfile.get_line(line_buff,39);
76              MatFileParam.sets(line_buff);
77              MatFileParam.gets(f_val);
78              sprintf(matname,"%s",f_val);
79              MatFileParam.gets(f_val);
80              sigma=atof(f_val);
81              MatFileParam.gets(f_val);
82              epsilonr=atof(f_val);
83
84              //printf("=%s, sig=%3.3e, eps=%3.3e",matname,sigma,epsilonr);
85              my_table[i].set(i,matname,sigma,EPS0*epsilonr,0.0,env);
86      }
87
88      matfile.close_file();
89  }
```

orbc_comp_const.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                          ORBC_COMP_CONST.H                         //
4   //                                                                    //
5   ////////////////////////////////////////////////////////////////////////
6   class orbc_const : public var_env
7   {
8   public:
9       char            dir;
10  private:
11      typ_champ       C1;
12      typ_champ       C2;
13      typ_champ       C3;
14      typ_champ       C4;
15      bool            ok_for_init;
16
17  public:
18      // Constructeur
19      orbc_const( typ_distance delx, typ_distance dely,
20                              typ_distance delz, char val)            : var_env (delx, dely,
delz)
21      {
22              ok_for_init=(dx*dy*dz*dt!=0.0);
23              dir=val;
24              if (ok_for_init) init();
25      };
26
27      // Constructeur
28      orbc_const( var_env env, char val): var_env(env.dx, env.dy,env.dz)
29      {
30              ok_for_init=(dx*dy*dz*dt!=0);
31              dir=val;
32              if (ok_for_init) init();
33      };
34
35      // Destructeur
36      ~orbc_const()    {};
37
38      // Accesseurs
39      inline typ_champ cxd()    {check_init(); check_call('X');return (C1);}
40      inline typ_champ cxx()    {check_init(); check_call('X');return (C2);}
41      inline typ_champ cxy()    {check_init(); check_call('X');return (C3);}
42      inline typ_champ cxz()    {check_init(); check_call('X');return (C4);}
43
44      inline typ_champ cyd()    {check_init(); check_call('Y');return (C1);}
45      inline typ_champ cyx()    {check_init(); check_call('Y');return (C2);}
46      inline typ_champ cyy()    {check_init(); check_call('Y');return (C3);}
47      inline typ_champ cyz()    {check_init(); check_call('Y');return (C4);}
48
49      inline typ_champ czd()    {check_init(); check_call('Z');return (C1);}
50      inline typ_champ czx()    {check_init(); check_call('Z');return (C2);}
51      inline typ_champ czy()    {check_init(); check_call('Z');return (C3);}
52      inline typ_champ czz()    {check_init(); check_call('Z');return (C4);}
```

```
53
54  private:
55
56      // Initialisation des coefficients
57      void init()
58      {
59              switch (dir)
60              {
61              case 'X':
62                      {
63                      /*CXD*/ C1 = ( C * dt - dx) / ( C * dt + dx);
64                      /*CXX*/ C2 = 2.0 * dx / ( C * dt + dx);
65                      /*CXY*/ C3 = dx * dt * dt * C * C / (2.0 * dy * dy * ( C * dt + dx));
66                      /*CXZ*/ C4 = dx * dt * dt * C * C / (2.0 * dz * dz * ( C * dt + dx));
67                      break;
68                      }
69              case 'Y':
70                      {
71                      /*CYD*/ C1 = ( C * dt - dy) / ( C * dt + dy);
72                      /*CYX*/ C2 = dy * dt * dt * C * C / (2.0 * dx * dx * ( C * dt + dy));

73                      /*CYY*/ C3 = 2.0 * dy / ( C * dt + dy);
74                      /*CYZ*/ C4 = dy * dt * dt * C * C / (2.0 * dz * dz * ( C * dt + dy));

75                      break;
76                      }
77              case 'Z':
78                      {
79                      /*CZD*/ C1 =( C * dt - dz) / ( C * dt + dz);
80                      /*CZX*/ C2 = dz * dt * dt * C * C / (2.0 * dx * dx * ( C * dt + dz));

81                      /*CZY*/ C3 = dz * dt * dt * C * C / (2.0 * dy * dy * ( C * dt + dz));
82                      /*CZZ*/ C4 = 2.0 * dz / ( C * dt + dz);
83                      break;
84                      }
85              }
86      };
87
88      // Controle d'appel
89      inline void check_call(char val)
90      {
91              if (dir!= val)
92              {
93              printf("Orbc - Irrelevant call. Non significant value.");
94              printf("Called with %c instead of %s", val, dir);
95              printf("Press any key to continue, q to abort");
96              if(getchar()== 'q') exit(1);
97              }
98      };
99
100     inline void check_init()
101     {
102             if (!ok_for_init)
103             {
104                     printf(" Non initialized value .");
105                     printf("Press any key to continue, q to abort");
106                     if(getchar() == 'q') exit(1);
107             }
108     };
```

```
109 };
```

Orbc_xy.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                      ORBC_XY.H     //
4   //                                  MODULE DE CALCUL DES ORBC         //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   template <class orbc_const>
8   void orbc_xy        (espace_champs   elcfields,
9                        tab_for_orbc     prev1,
10                       tab_for_orbc     prev2,
11                       orbc_const               orbcs)
12  {
13     int i,k;
14     int NX=elcfields.nx();
15     int NY=elcfields.ny();
16     int NZ=elcfields.nz();
17     typ_champ                my_value;
18     cellule_champs   my_cell,new_cell;
19
20     // ------------------------ Ordre 1 ------------------------- //
21     for(k=2;k<=(NZ-1);k++)
22     {
23             // ------------------------------------------------- //
24             i=1;
25             my_cell  = elcfields.get(i,2,k);
26             new_cell = elcfields.get(i,1,k); // LR
27             my_value = prev1.get(i,2,k) + orbcs.cyd() * (my_cell.cx() - prev1.get(i,1,k));
28             new_cell.setx(my_value);
29             elcfields.put(new_cell,i,1,k);
30             // ------------------------------------------------- //
31             my_cell  = elcfields.get(i,NY-1,k);
32             new_cell  = elcfields.get(i,NY,k);//LR
33             my_value = prev1.get(i,3,k) + orbcs.cyd() * (my_cell.cx() - prev1.get(i,4,k));
34             new_cell.setx(my_value);
35             elcfields.put(new_cell,i,NY,k);
36
37             // ------------------------------------------------- //
38             i=NX-1;
39             my_cell  = elcfields.get(i,2,k);
40             new_cell  = elcfields.get(i,1,k);
41             my_value = prev1.get(i,2,k) + orbcs.cyd() * (my_cell.cx() - prev1.get(i,1,k));
42             new_cell.setx(my_value);
43             elcfields.put(new_cell,i,1,k);
44             // ------------------------------------------------- //
45             my_cell  = elcfields.get(i,NY-1,k);
46             new_cell  = elcfields.get(i,NY,k);
47             my_value = prev1.get(i,3,k) + orbcs.cyd() * (my_cell.cx() - prev1.get(i,4,k));
48             new_cell.setx(my_value);
49             elcfields.put(new_cell,i,NY,k);
50     }
51
52     for(i=2;i<=(NX-2);i++)
```

```
53      {
54              // ------------------------------------------------------ //
55              k=2;
56              my_cell  = elcfields.get(i,2,k);
57              new_cell  = elcfields.get(i,1,k);
58              my_value = prev1.get(i,2,k) + orbcs.cyd() * (my_cell.cx() - prev1.get(i,1,k));
59              new_cell.setx(my_value);
60              elcfields.put(new_cell,i,1,k);
61              // ------------------------------------------------------ //
62              my_cell  = elcfields.get(i,NY-1,k);
63              new_cell  = elcfields.get(i,NY,k);
64              my_value = prev1.get(i,3,k) + orbcs.cyd() * (my_cell.cx() - prev1.get(i,4,k));
65              new_cell.setx(my_value);
66              elcfields.put(new_cell,i,NY,k);
67              // ------------------------------------------------------ //
68              k=NZ-1;
69              my_cell  = elcfields.get(i,2,k);
70              new_cell = elcfields.get(i,1,k);
71              my_value = prev1.get(i,2,k) + orbcs.cyd() * (my_cell.cx() - prev1.get(i,1,k));
72              new_cell.setx(my_value);
73              elcfields.put(new_cell,i,1,k);
74              // ------------------------------------------------------ //
75              my_cell  = elcfields.get(i,NY-1,k);
76              my_cell  = elcfields.get(i,NY,k);
77              my_value = prev1.get(i,3,k) + orbcs.cyd() *(my_cell.cx() - prev1.get(i,4,k));
78              new_cell.setx(my_value);
79              elcfields.put(new_cell,i,NY,k);
80      }
81
82      // ------------------------- Ordre 2 ------------------------- //
83      for(k=3;k<=NZ-2;k++)
84              for(i=2;i<=NX-2;i++)
85              {
86                      my_cell  = elcfields.get(i,2,k);
87                      new_cell  = elcfields.get(i,1,k);
88                      my_value =
89                              - prev2.get(i,2,k)
90                              + orbcs.cyd() *( my_cell.cx()              +      prev2.get(i,1,k) )
91                              + orbcs.cyy() *  ( prev1.get(i,1,k)        +      prev1.get(i,2,k) )
92                              + orbcs.cyx() * ( prev1.get(i+1,1,k) -    2.0 * prev1.get(i,1,k) +
prev1.get(i-1,1,k)
93                                                                        + prev1.get(i+1,2,k) -   2.0 *
prev1.get(i,2,k) + prev1.get(i-1,2,k) )
94                              + orbcs.cyz() * ( prev1.get(i,1,k+1) -    2.0 * prev1.get(i,1,k) *
prev1.get(i,1,k-1)
95                                                                        + prev1.get(i,2,k+1) -   2.0 *
prev1.get(i,2,k) + prev1.get(i,2,k-1) );
96                      new_cell.setx(my_value);
97                      elcfields.put(new_cell,i,1,k);
98                      // ------------------------------------------------------ //
99                      my_cell = elcfields.get(i,NY-1,k);
100                     new_cell = elcfields.get(i,NY,k);
101                     my_value =
102                             -prev2.get(i,3,k)
103                             + orbcs.cyd() * ( my_cell.cx()             +      prev2.get(i,4,k) )
104                             + orbcs.cyy() *  ( prev1.get(i,4,k)        +      prev1.get(i,3,k) )
105                             + orbcs.cyx() * ( prev1.get(i+1,4,k) -    2.0 * prev1.get(i,4,k) +
prev1.get(i-1,4,k)
106                                                                       + prev1.get(i+1,3,k) -   2.0 *
```

```
prev1.get(i,3,k) + prev1.get(i-1,3,k) )
107                                 + orbcs.cyz() * ( prev1.get(i,4,k+1) -    2.0 * prev1.get(i,4,k) +
prev1.get(i,4,k-1)
108                                                                       + prev1.get(i,3,k+1) -   2.0 *
prev1.get(i,3,k) + prev1.get(i,3,k-1) );
109                         new_cell.setx(my_value);
110                         elcfields.put(new_cell,i,NY,k);
111                 }
112
113         // ------------ Sauvegarde des valeurs anterieures ------------- //
114         for (k=2;k<=NZ-1;k++)
115                 for(i=1;i<=NX-1;i++)
116                 {
117                         prev2.put(prev1.get(i,1,k),i,1,k);
118                         prev2.put(prev1.get(i,2,k),i,2,k);
119                         prev2.put(prev1.get(i,3,k),i,3,k);
120                         prev2.put(prev1.get(i,4,k),i,4,k);
121                         prev1.put(elcfields.get(i,1,k).cx(),i,1,k);
122                         prev1.put(elcfields.get(i,2,k).cx(),i,2,k);
123                         prev1.put(elcfields.get(i,NY-1,k).cx(),i,3,k);
124                         prev1.put(elcfields.get(i,NY  ,k).cx(),i,4,k);
125                 }
126   }
```

Orbc_xz.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                        ORBC_XZ.H   //
4   //                                                                    //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   template <class orbc_const>
8   void orbc_xz        (espace_champs   elcfields,
9                        tab_for_orbc     prev1,
10                       tab_for_orbc     prev2,
11                       orbc_const               orbcs)
12  {
13     int i,j;
14     int NX=elcfields.nx();
15     int NY=elcfields.ny();
16     int NZ=elcfields.nz();
17     typ_champ                my_value;
18     cellule_champs   my_cell,new_cell;
19
20     // ------------------------ Ordre 1 ------------------------- //
21
22     for(j=2;j<=NY-1;j++)
23     {
24             // ------------------------------------------------------- //
25             i=1;
26             my_cell  = elcfields.get(i,j,2);
27             new_cell  = elcfields.get(i,j,1);
28             my_value = prev1.get(i,j,2) + orbcs.czd() * (my_cell.cx() - prev1.get(i,j,1));
29             new_cell.setx(my_value);
30             elcfields.put(new_cell,i,j,1);
31             // ------------------------------------------------------- //
32             my_cell  = elcfields.get(i,j,NZ-1);
33             new_cell  = elcfields.get(i,j,NZ);
34             my_value = prev1.get(i,j,3) + orbcs.czd() * (my_cell.cx() - prev1.get(i,j,4));
35             new_cell.setx(my_value);
36             elcfields.put(new_cell,i,j,NZ);
37             // ------------------------------------------------------- //
38             i=NX-1;
39             my_cell  = elcfields.get(i,j,2);
40             new_cell  = elcfields.get(i,j,1);
41             my_value = prev1.get(i,j,2) + orbcs.czd() * (my_cell.cx() - prev1.get(i,j,1));
42             new_cell.setx(my_value);
43             elcfields.put(new_cell,i,j,1);
44             // ------------------------------------------------------- //
45             my_cell  = elcfields.get(i,j,NZ-1);
46             new_cell  = elcfields.get(i,j,NZ);
47             my_value = prev1.get(i,j,3) + orbcs.czd() * (my_cell.cx() - prev1.get(i,j,4));
48             new_cell.setx(my_value);
49             elcfields.put(new_cell,i,j,NZ);
50     }
51
52     for(i=2;i<=NX-2;i++)
```

```
53      {
54              // ------------------------------------------------------ //
55              j=2;
56              my_cell  = elcfields.get(i,j,2);
57              new_cell  = elcfields.get(i,j,1);
58              my_value = prev1.get(i,j,2) + orbcs.czd() * (my_cell.cx() - prev1.get(i,j,1));
59              new_cell.setx(my_value);
60              elcfields.put(new_cell,i,j,1);
61              // ------------------------------------------------------ //
62              my_cell  = elcfields.get(i,j,NZ-1);
63              new_cell  = elcfields.get(i,j,NZ);
64              my_value = prev1.get(i,j,3) + orbcs.czd() * (my_cell.cx() - prev1.get(i,j,4));
65              new_cell.setx(my_value);
66              elcfields.put(new_cell,i,j,NZ);
67              // ------------------------------------------------------ //
68              j=NY-1;
69              my_cell  = elcfields.get(i,j,2);
70              new_cell = elcfields.get(i,j,1);
71              my_value = prev1.get(i,j,2) + orbcs.czd() * (my_cell.cx() - prev1.get(i,j,1));
72              new_cell.setx(my_value);
73              elcfields.put(new_cell,i,j,1);
74              // ------------------------------------------------------ //
75              my_cell  = elcfields.get(i,j,NZ-1);
76              new_cell = elcfields.get(i,j,NZ);
77              my_value = prev1.get(i,j,3) + orbcs.czd() * (my_cell.cx() - prev1.get(i,j,4));
78              new_cell.setx(my_value);
79              elcfields.put(new_cell,i,j,NZ);
80      }
81
82      // ------------------------ Ordre 2 ------------------------- //
83
84      for(j=3;j<=NY-2;j++)
85              for(i=2;i<=NX-2;i++)
86              {
87                      my_cell  = elcfields.get(i,j,2);
88                      new_cell = elcfields.get(i,j,1);
89                      my_value =
90                              - prev2.get(i,j,2)
91                              + orbcs.czd() * ( my_cell.cx()              +       prev2.get(i,j,1) )
92                              + orbcs.czz() *  ( prev1.get(i,j,1)         +       prev1.get(i,j,2) )
93                              + orbcs.czx() * ( prev1.get(i+1,j,1) -    2.0 * prev1.get(i,j,1) +
prev1.get(i-1,j,1)
94                                                                      + prev1.get(i+1,j,2) -   2.0 *
prev1.get(i,j,2) + prev1.get(i-1,j,2) )
95                              + orbcs.czy() * ( prev1.get(i,j+1,1) -    2.0 * prev1.get(i,j,1) +
prev1.get(i,j-1,1)
96                                                                      + prev1.get(i,j+1,2) -   2.0 *
prev1.get(i,j,2) + prev1.get(i,j-1,2) );
97                      new_cell.setx(my_value);
98                      elcfields.put(new_cell,i,j,1);
99                      // ------------------------------------------------------ //
100                     my_cell = elcfields.get(i,j,NZ-1);
101                     new_cell = elcfields.get(i,j,NZ);
102                     my_value =
103                             - prev2.get(i,j,3)
104                             + orbcs.czd() * ( my_cell.cx()              +       prev2.get(i,j,4) )
105                             + orbcs.czz() *  ( prev1.get(i,j,4)         +       prev1.get(i,j,3) )
106                             + orbcs.czx() * ( prev1.get(i+1,j,4) -    2.0 * prev1.get(i,j,4) +
prev1.get(i-1,j,4)
```

```
107                                                                    + prev1.get(i+1,j,3) -    2.0 *
prev1.get(i,j,3) + prev1.get(i-1,j,3) )
108                                + orbcs.czy() * ( prev1.get(i,j+1,4) -    2.0 * prev1.get(i,j,4) +
prev1.get(i,j-1,4)
109                                                                    + prev1.get(i,j+1,3) -    2.0 *
prev1.get(i,j,3) + prev1.get(i,j-1,3) );
110                         new_cell.setx(my_value);
111                         elcfields.put(new_cell,i,j,NZ);
112                 }
113         // ------------ Sauvegarde des valeurs anterieures ------------- //
114         for(j=2; j<=NY-1; j++)
115                 for(i=1; i<=NX-1; i++)
116                 {
117                         prev2.put(prev1.get(i,j,1),i,j,1);
118                         prev2.put(prev1.get(i,j,2),i,j,2);
119                         prev2.put(prev1.get(i,j,3),i,j,3);
120                         prev2.put(prev1.get(i,j,4),i,j,4);
121                         prev1.put(elcfields.get(i,j,1).cx(),i,j,1);
122                         prev1.put(elcfields.get(i,j,2).cx(),i,j,2);
123                         prev1.put(elcfields.get(i,j,NZ-1).cx(),i,j,3);
124                         prev1.put(elcfields.get(i,j,NZ  ).cx(),i,j,4);
125                 }
126     }
127
128
```

Orbc_yx.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                      ORBC_YX.H     //
4   //                                                                    //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   // ================================================================== //
8   // ======================     ORBC_YX      ========================== //
9   // ================================================================== //
10
11  template <class orbc_const>
12  void orbc_yx        (espace_champs   elcfields,
13                       tab_for_orbc     prev1,
14                       tab_for_orbc     prev2,
15                       orbc_const               orbcs)
16  {
17     int j,k;
18     int NX=elcfields.nx();
19     int NY=elcfields.ny();
20     int NZ=elcfields.nz();
21
22     typ_champ                my_value;
23     cellule_champs   my_cell,new_cell;
24
25     // ------------------------ Ordre 1 -------------------------- //
26     for(k=2;k<=(NZ-1);k++)
27     {
28             // ------------------------------------------------------- //
29             j=1;
30             my_cell   = elcfields.get(2,j,k);
31             new_cell  = elcfields.get(1,j,k);
32             my_value = prev1.get(2,j,k) + orbcs.cxd() * (my_cell.cy() - prev1.get(1,j,k));
33             new_cell.sety(my_value);
34             elcfields.put(new_cell,1,j,k);
35             // ------------------------------------------------------- //
36             my_cell = elcfields.get(NX-1,j,k);
37             new_cell = elcfields.get(NX,j,k);
38             my_value =       prev1.get(3,j,k) + orbcs.cxd()  * (my_cell.cy() - prev1.get(4,j,k));
39             new_cell.sety(my_value);
40             elcfields.put(new_cell,NX,j,k);
41             // ------------------------------------------------------- //
42             j=NY-1;
43             my_cell = elcfields.get(2,j,k);
44             new_cell = elcfields.get(1,j,k);
45             my_value =       prev1.get(2,j,k) + orbcs.cxd()  * (my_cell.cy() - prev1.get(1,j,k));
46             new_cell.sety(my_value);
47             elcfields.put(new_cell,1,j,k);
48
49             // ------------------------------------------------------- //
50             my_cell = elcfields.get(NX-1,j,k);
51             new_cell = elcfields.get(NX,j,k);
52             my_value =       prev1.get(3,j,k) + orbcs.cxd()  * (my_cell.cy() - prev1.get(4,j,k));
```

```
53              new_cell.sety(my_value);
54              elcfields.put(new_cell,NX,j,k);
55
56      }
57      // ------------------------------------------------------------ //
58      for(j=2;j<=(NY-2);j++)
59      {
60              // ..................................................... //
61              k=2;
62              my_cell = elcfields.get(2,j,k);
63              new_cell = elcfields.get(1,j,k);
64              my_value =      prev1.get(2,j,k) + orbcs.cxd() * (my_cell.cy() - prev1.get(1,j,k));
65              new_cell.sety(my_value);
66              elcfields.put(new_cell,1,j,k);
67              // ..................................................... //
68              my_cell = elcfields.get(NX-1,j,k);
69              new_cell = elcfields.get(NX,j,k);
70              my_value =      prev1.get(3,j,k) + orbcs.cxd()  * (my_cell.cy() - prev1.get(4,j,k));
71              new_cell.sety(my_value);
72              elcfields.put(new_cell,NX,j,k);
73              // ..................................................... //
74              k=NZ-1;
75              my_cell = elcfields.get(2,j,k);
76              new_cell = elcfields.get(1,j,k);
77              my_value =      prev1.get(2,j,k) + orbcs.cxd()  * (my_cell.cy() - prev1.get(1,j,k));
78              new_cell.sety(my_value);
79              elcfields.put(new_cell,1,j,k);
80              // ..................................................... //
81              my_cell = elcfields.get(NX-1,j,k);
82              new_cell = elcfields.get(NX,j,k);
83              my_value =      prev1.get(3,j,k) + orbcs.cxd()  * (my_cell.cy() - prev1.get(4,j,k));
84              new_cell.sety(my_value);
85              elcfields.put(new_cell,NX,j,k);
86      }
87      // ------------------------ Ordre 2 ------------------------- //
88      for(k=3;k<=NZ-2;k++)
89              for(j=2; j<=NY-2; j++)
90              {
91                      // ..................................................... //
92                      my_cell = elcfields.get(2,j,k);
93                      new_cell = elcfields.get(1,j,k);
94                      my_value =
95                              - prev2.get(2,j,k)
96                              + orbcs.cxd() * ( my_cell.cy()              + prev2.get(1,j,k) )
97                              + orbcs.cxx() *  ( prev1.get(1,j,k)         + prev1.get(2,j,k) )
98                              + orbcs.cxy() * ( prev1.get(1,j+1,k) - 2.0 * prev1.get(1,j,k) +
prev1.get(1,j-1,k)
99                                                                  + prev1.get(2,j+1,k) - 2.0 *
prev1.get(2,j,k) +      prev1.get(2,j-1,k) )
100                             + orbcs.cxz() * ( prev1.get(1,j,k+1) - 2.0 * prev1.get(1,j,k) *
prev1.get(1,j,k-1)
101                                                                 + prev1.get(2,j,k+1) - 2.0 *
prev1.get(2,j,k) + prev1.get(2,j,k-1) );
102                     new_cell.sety(my_value);
103                     elcfields.put(new_cell,1,j,k);
104                     // ..................................................... //
105                     my_cell = elcfields.get(NX-1,j,k);
106                     new_cell = elcfields.get(NX,j,k);
107                     my_value =
```

```
                              -prev2.get(3,j,k)
                              + orbcs.cxd() * ( my_cell.cy()            + prev2.get(4,j,k) )
                              + orbcs.cxx() *  ( prev1.get(4,j,k)        + prev1.get(3,j,k) )
                              + orbcs.cxy() * ( prev1.get(4,j+1,k) - 2.0 * prev1.get(4,j,k) +
prev1.get(4,j-1,k)
                                                                 + prev1.get(3,j+1,k) - 2.0 *
prev1.get(3,j,k) +       prev1.get(3,j-1,k) )
                              + orbcs.cxz() * ( prev1.get(4,j,k+1) - 2.0 * prev1.get(4,j,k) +
     prev1.get(4,j,k-1)
                                                                 + prev1.get(3,j,k+1) - 2.0 *
prev1.get(3,j,k) +       prev1.get(3,j,k-1) );
                         new_cell.sety(my_value);
                         elcfields.put(new_cell,NX,j,k);
               }
               // ------------ Sauvegarde des valeurs anterieures ------------- //
               for (k=2; k<= NZ-1; k++)
                         for(j=1; j<=NY-1; j++)
                         {
                                   prev2.put(prev1.get(1,j,k),1,j,k);
                                   prev2.put(prev1.get(2,j,k),2,j,k);
                                   prev2.put(prev1.get(3,j,k),3,j,k);
                                   prev2.put(prev1.get(4,j,k),4,j,k);
                                   prev1.put(elcfields.get(1, j, k ).cy(),1,j,k);
                                   prev1.put(elcfields.get(2, j, k ).cy(),2,j,k);
                                   prev1.put(elcfields.get(NX-1,j,k).cy(),3,j,k);
                                   prev1.put(elcfields.get(NX  ,j,k).cy(),4,j,k);
                         }
 }
 // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
```

Orbc_yz.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                      ORBC_YZ.H     //
4   //                                  MODULE DE CALCUL DES ORBC         //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   template <class orbc_const>
8   void orbc_yz(       espace_champs    elcfields,
9                           tab_for_orbc     prev1,
10                          tab_for_orbc     prev2,
11                          orbc_const               orbcs)
12  {
13     int i,j;
14     int NX=elcfields.nx();
15     int NY=elcfields.ny();
16     int NZ=elcfields.nz();
17     typ_champ                my_value;
18     cellule_champs   my_cell,new_cell;
19     // ------------------------ Ordre 1 -------------------------- //
20     for(j=1;j<=(NY-1);j++)
21     {
22             // ------------------------------------------------------- //
23             i=2;
24             my_cell  = elcfields.get(i,j,2);
25             new_cell  = elcfields.get(i,j,1);
26             my_value = prev1.get(i,j,2) + orbcs.czd() * (my_cell.cy() - prev1.get(i,j,1));
27             new_cell.sety(my_value);
28             elcfields.put(new_cell,i,j,1);
29             // ------------------------------------------------------- //
30             my_cell  = elcfields.get(i,j,NZ-1);
31             new_cell  = elcfields.get(i,j,NZ);
32             my_value = prev1.get(i,j,3) + orbcs.czd() * (my_cell.cy() - prev1.get(i,j,4));
33             new_cell.sety(my_value);
34             elcfields.put(new_cell,i,j,NZ);
35             // ------------------------------------------------------- //
36             i=NX-1;
37             my_cell  = elcfields.get(i,j,2);
38             new_cell  = elcfields.get(i,j,1);
39             my_value = prev1.get(i,j,2) + orbcs.czd() * (my_cell.cy() - prev1.get(i,j,1));
40             new_cell.sety(my_value);
41             elcfields.put(new_cell,i,j,1);
42             // ------------------------------------------------------- //
43             my_cell  = elcfields.get(i,j,NZ-1);
44             new_cell  = elcfields.get(i,j,NZ);
45             my_value = prev1.get(i,j,3) + orbcs.czd() * (my_cell.cy() - prev1.get(i,j,4));
46             new_cell.sety(my_value);
47             elcfields.put(new_cell,i,j,NZ);
48     }
49
50     for(i=3;i<=(NX-2);i++)
51     {
52             // ------------------------------------------------------- //
```

```
53              j=1;
54              my_cell  = elcfields.get(i,j,2);
55              new_cell  = elcfields.get(i,j,1);
56              my_value = prev1.get(i,j,2) + orbcs.czd() ** (my_cell.cy() - prev1.get(i,j,1));
57              new_cell.sety(my_value);
58              elcfields.put(new_cell,i,j,1);
59              // ..................................................... //
60              my_cell  = elcfields.get(i,j,NZ-1);
61              new_cell  = elcfields.get(i,j,NZ);
62              my_value = prev1.get(i,j,3) + orbcs.czd() ** (my_cell.cy() - prev1.get(i,j,4));
63              new_cell.sety(my_value);
64              elcfields.put(new_cell,i,j,NZ);
65              // ..................................................... //
66              j=NY-1;
67              my_cell  = elcfields.get(i,j,2);
68              new_cell  = elcfields.get(i,j,1);
69              my_value = prev1.get(i,j,2) + orbcs.czd() ** (my_cell.cy() - prev1.get(i,j,1));
70              new_cell.sety(my_value);
71              elcfields.put(new_cell,i,j,1);
72              // ..................................................... //
73              my_cell  = elcfields.get(i,j,NZ-1);
74              new_cell  = elcfields.get(i,j,NZ);
75              my_value = prev1.get(i,j,3) + orbcs.czd() ** (my_cell.cy() - prev1.get(i,j,4));
76              new_cell.sety(my_value);
77              elcfields.put(new_cell,i,j,NZ);
78      }
79      // ------------------------ Ordre 2 ------------------------- //
80      for(j=2;j<=NY-2;j++)
81              for(i=3;i<=NX-2;i++)
82              {
83                      my_cell  = elcfields.get(i,j,2);
84                      new_cell  = elcfields.get(i,j,1);
85                      my_value =
86                              - prev2.get(i,j,2)
87                              + orbcs.czd() **( my_cell.cy()              +       prev2.get(i,j,1) )
88                              + orbcs.czz() ** ( prev1.get(i,j,1)         +       prev1.get(i,j,2) )
89                              + orbcs.czx() **( prev1.get(i+1,j,1) -    2.0 **prev1.get(i,j,1) +
prev1.get(i-1,j,1)
90                                                                      + prev1.get(i+1,j,2) -    2.0 *
prev1.get(i,j,2) + prev1.get(i-1,j,2) )
91                              + orbcs.czy() * ( prev1.get(i,j+1,1) -    2.0 * prev1.get(i,j,1) +
prev1.get(i,j-1,1)
92                                                                      + prev1.get(i,j+1,2) -    2.0 **
prev1.get(i,j,2) + prev1.get(i,j-1,2) );
93                      new_cell.sety(my_value);
94                      elcfields.put(new_cell,i,j,1);
95                      // ..................................................... //
96                      my_cell = elcfields.get(i,j,NZ-1);
97                      new_cell = elcfields.get(i,j,NZ);
98                      my_value =
99                              - prev2.get(i,j,3)
100                             + orbcs.czd() **( my_cell.cy()              +       prev2.get(i,j,4) )
101                             + orbcs.czz() ** ( prev1.get(i,j,4)         +       prev1.get(i,j,3) )
102                             + orbcs.czx() **( prev1.get(i+1,j,4) -    2.0 **prev1.get(i,j,4) +
prev1.get(i-1,j,4)
103                                                                     + prev1.get(i+1,j,3) -    2.0 *
prev1.get(i,j,3) + prev1.get(i-1,j,3) )
104                             + orbcs.czy() * ( prev1.get(i,j+1,4) -    2.0 * prev1.get(i,j,4) +
prev1.get(i,j-1,4)
```

```
105                                                                    + prev1.get(i,j+1,3) -   2.0 *
prev1.get(i,j,3) + prev1.get(i,j-1,3) );
106                         new_cell.sety(my_value);
107                         elcfields.put(new_cell,i,j,NZ);
108                 }
109         // ------------ Sauvegarde des valeurs anterieures ------------- //
110         for (j=1; j<= NY-1; j++)
111                 for(i=2; i<=NX-1; i++)
112                 {
113                         prev2.put(prev1.get(i,j,1),i,j,1);
114                         prev2.put(prev1.get(i,j,2),i,j,2);
115                         prev2.put(prev1.get(i,j,3),i,j,3);
116                         prev2.put(prev1.get(i,j,4),i,j,4);
117                         prev1.put(elcfields.get(i,j,1).cy(),i,j,1);
118                         prev1.put(elcfields.get(i,j,2).cy(),i,j,2);
119                         prev1.put(elcfields.get(i,j,NZ-1).cy(),i,j,3);
120                         prev1.put(elcfields.get(i,j,NZ  ).cy(),i,j,4);
121                 }
122     }
123
```

Orbc_zx.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                      ORBC_ZX.H     //
4   //                                MODULE DE CALCUL DES ORBC           //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
8   // ::::::::::::::::::::::     ORBC_ZX      :::::::::::::::::::::::: //
9   // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: //
10
11  template <class orbc_const>
12  void orbc_zx        (espace_champs   elcfields,
13                       tab_for_orbc     prev1,
14                       tab_for_orbc     prev2,
15                       orbc_const                orbcs)
16  {
17     int j,k;
18     int NX=elcfields.nx();
19     int NY=elcfields.ny();
20     int NZ=elcfields.nz();
21
22     typ_champ               my_value;
23     cellule_champs   my_cell,new_cell;
24
25     // ------------------------ Ordre 1 ------------------------- //
26
27     for(k=1;k<=(NZ-1);k++)
28     {
29             // ..................................................... //
30             j=2;
31             my_cell = elcfields.get(2,j,k);
32             new_cell = elcfields.get(1,j,k);
33             my_value =prev1.get(2,j,k) + orbcs.cxd() **(my_cell.cz() - prev1.get(1,j,k));
34             new_cell.setz(my_value);
35             elcfields.put(new_cell,1,j,k);
36             // ..................................................... //
37             my_cell = elcfields.get(NX-1,j,k);
38             new_cell = elcfields.get(NX,j,k);
39             my_value =      prev1.get(3,j,k) + orbcs.cxd()  **(my_cell.cz() - prev1.get(4,j,k));
40             new_cell.setz(my_value);
41             elcfields.put(new_cell,NX,j,k);
42             // ..................................................... //
43             j=NY-1;
44             my_cell = elcfields.get(2,j,k);
45             new_cell = elcfields.get(1,j,k);
46             my_value =      prev1.get(2,j,k) + orbcs.cxd()  **(my_cell.cz() - prev1.get(1,j,k));
47             new_cell.setz(my_value);
48             elcfields.put(new_cell,1,j,k);
49             // ..................................................... //
50             my_cell = elcfields.get(NX-1,j,k);
51             new_cell = elcfields.get(NX,j,k);
52             my_value =      prev1.get(3,j,k) + orbcs.cxd()  **(my_cell.cz() - prev1.get(4,j,k));
```

```
53              new_cell.setz(my_value);
54              elcfields.put(new_cell,NX,j,k);
55      }
56
57      // ------------------------------------------------------------ //
58      for(j=3;j<=(NY-2);j++)
59      {
60              // ------------------------------------------------------- //
61              k=1;
62              my_cell = elcfields.get(2,j,k);
63              new_cell = elcfields.get(1,j,k);
64              my_value =      prev1.get(2,j,k) + orbcs.cxd() * (my_cell.cz() - prev1.get(1,j,k));
65              new_cell.setz(my_value);
66              elcfields.put(new_cell,1,j,k);
67              // ------------------------------------------------------- //
68              my_cell = elcfields.get(NX-1,j,k);
69              new_cell = elcfields.get(NX,j,k);
70              my_value =      prev1.get(3,j,k) + orbcs.cxd()  * (my_cell.cz() - prev1.get(4,j,k));
71              new_cell.setz(my_value);
72              elcfields.put(new_cell,NX,j,k);
73              // ------------------------------------------------------- //
74              k=NZ-1;
75              my_cell = elcfields.get(2,j,k);
76              new_cell = elcfields.get(1,j,k);
77              my_value =      prev1.get(2,j,k) + orbcs.cxd()  * (my_cell.cz() - prev1.get(1,j,k));
78              new_cell.setz(my_value);
79              elcfields.put(new_cell,1,j,k);
80              // ------------------------------------------------------- //
81              my_cell = elcfields.get(NX-1,j,k);
82              new_cell = elcfields.get(NX,j,k);
83              my_value =      prev1.get(3,j,k) + orbcs.cxd()  * (my_cell.cz() - prev1.get(4,j,k));
84              new_cell.setz(my_value);
85              elcfields.put(new_cell,NX,j,k);
86      }
87
88      // ------------------------ Ordre 2 ------------------------- //
89      for(k=2;k<=(NZ-2);k++)
90              for(j=3; j<=(NY-2); j++)
91              {
92                      my_cell  = elcfields.get(2,j,k);
93                      new_cell  = elcfields.get(1,j,k);
94                      my_value =
95                              - prev2.get(2,j,k)
96                              + orbcs.cxd() * ( my_cell.cz()              + prev2.get(1,j,k) )
97                              + orbcs.cxx() *  ( prev1.get(1,j,k)         + prev1.get(2,j,k) )
98                              + orbcs.cxy() *( prev1.get(1,j+1,k) - 2.0 * prev1.get(1,j,k) +
        prev1.get(1,j-1,k)
99                                                                  + prev1.get(2,j+1,k) - 2.0 *
prev1.get(2,j,k) +      prev1.get(2,j-1,k) )
100                             + orbcs.cxz() *( prev1.get(1,j,k+1) - 2.0 * prev1.get(1,j,k) +
        prev1.get(1,j,k-1)
101                                                                 + prev1.get(2,j,k+1) - 2.0 *
prev1.get(2,j,k) +      prev1.get(2,j,k-1) );
102                     new_cell.setz(my_value);
103                     elcfields.put(new_cell,1,j,k);
104                     // ------------------------------------------------------- //
105                     my_cell = elcfields.get(NX-1,j,k);
106                     new_cell = elcfields.get(NX,j,k);
107                     my_value =
```

```
108                             -prev2.get(3,j,k)
109                             + orbcs.cxd() * ( my_cell.cz()              + prev2.get(4,j,k) )
110                             + orbcs.cxx() *  ( prev1.get(4,j,k)         + prev1.get(3,j,k) )
111                             + orbcs.cxy() * ( prev1.get(4,j+1,k) - 2.0 * prev1.get(4,j,k) +
        prev1.get(4,j-1,k)
112                                                                 + prev1.get(3,j+1,k) - 2.0 *
prev1.get(3,j,k) +      prev1.get(3,j-1,k) )
113                             + orbcs.cxz() * ( prev1.get(4,j,k+1) - 2.0 * prev1.get(4,j,k) +
        prev1.get(4,j,k-1)
114                                                                 + prev1.get(3,j,k+1) - 2.0 *
prev1.get(3,j,k) +      prev1.get(3,j,k-1) );
115                     new_cell.setz(my_value);
116                     elcfields.put(new_cell,NX,j,k);
117             }
118     // ------------ Sauvegarde des valeurs anterieures ------------- //
119
120             for (k=1; k<=NZ-1; k++)
121                     for(j=2; j<=NY-1; j++)
122                     {
123                             prev2.put(prev1.get(1,j,k),1,j,k);
124                             prev2.put(prev1.get(2,j,k),2,j,k);
125                             prev2.put(prev1.get(3,j,k),3,j,k);
126                             prev2.put(prev1.get(4,j,k),4,j,k);
127                             prev1.put(elcfields.get(1,j,k).cz(),1,j,k);
128                             prev1.put(elcfields.get(2,j,k).cz(),2,j,k);
129                             prev1.put(elcfields.get(NX-1,j,k).cz(),3,j,k);
130                             prev1.put(elcfields.get(  NX,j,k).cz(),4,j,k);
131                     }
132
133 }
```

Orbc_zy.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //
                                    //
3   //                                                          ORBC_ZY.H
                               //
4   //                                        MODULE DE CALCUL DES ORBC
       //
5   //
                                    //
6   ////////////////////////////////////////////////////////////////////////
7   template <class orbc_const>
8   void orbc_zy        (espace_champs   elcfields,
9                            tab_for_orbc     prev1,
10                           tab_for_orbc     prev2,
11                           orbc_const                orbcs)
12  {
13     int i,k;
14     int NX=elcfields.nx();
15     int NY=elcfields.ny();
16     int NZ=elcfields.nz();
17     typ_champ               my_value;
18     cellule_champs   my_cell,new_cell;
19
20     // ------------------------ Ordre 1 -------------------------- //
21     for(k=1;k<=(NZ-1);k++)
22     {
23             // ..................................................... //
24             i=2;
25             my_cell  = elcfields.get(i,2,k);
26             new_cell  = elcfields.get(i,1,k);
27             my_value = prev1.get(i,2,k) + orbcs.cyd() **(my_cell.cz() - prev1.get(i,1,k));
28             new_cell.setz(my_value);
29             elcfields.put(new_cell,i,1,k);
30             // ..................................................... //
31             my_cell  = elcfields.get(i,NY-1,k);
32             new_cell  = elcfields.get(i,NY,k);
33             my_value = prev1.get(i,3,k) + orbcs.cyd()  * (my_cell.cz() - prev1.get(i,4,k));
34             new_cell.setz(my_value);
35             elcfields.put(new_cell,i,NY,k);
36             // ..................................................... //
37             i=NX-1;
38             my_cell  = elcfields.get(i,2,k);
39             new_cell  = elcfields.get(i,1,k);
40             my_value = prev1.get(i,2,k) + orbcs.cyd()  * (my_cell.cz() - prev1.get(i,1,k));
41             new_cell.setz(my_value);
42             elcfields.put(new_cell,i,1,k);
43             // ..................................................... //
44             my_cell  = elcfields.get(i,NY-1,k);
45             new_cell  = elcfields.get(i,NY,k);
46             my_value = prev1.get(i,3,k) + orbcs.cyd()  **(my_cell.cz() - prev1.get(i,4,k));
47             new_cell.setz(my_value);
48             elcfields.put(new_cell,i,NY,k);
49     }
50
51     for(i=3;i<=NX-2;i++)
52     {
```

```
53              // ..................................................... //
54              k=1;
55              my_cell  = elcfields.get(i,2,k);
56              new_cell  = elcfields.get(i,1,k);
57              my_value = prev1.get(i,2,k) + orbcs.cyd() * (my_cell.cz() - prev1.get(i,1,k));
58              new_cell.setz(my_value);
59              elcfields.put(new_cell,i,1,k);
60              // ..................................................... //
61              my_cell  = elcfields.get(i,NY-1,k);
62              new_cell  = elcfields.get(i,NY,k);
63              my_value = prev1.get(i,3,k) + orbcs.cyd()  * (my_cell.cz() - prev1.get(i,4,k));
64              new_cell.setz(my_value);
65              elcfields.put(new_cell,i,NY,k);
66              // ..................................................... //
67              k=NZ-1;
68              my_cell  = elcfields.get(i,2,k);
69              new_cell  = elcfields.get(i,1,k);
70              my_value = prev1.get(i,2,k) + orbcs.cyd()  * (my_cell.cz() - prev1.get(i,1,k));
71              new_cell.setz(my_value);
72              elcfields.put(new_cell,i,1,k);
73              // ..................................................... //
74              my_cell  = elcfields.get(i,NY-1,k);
75              new_cell  = elcfields.get(i,NY,k);
76              my_value = prev1.get(i,3,k) + orbcs.cyd()  * (my_cell.cz() - prev1.get(i,4,k));
77              new_cell.setz(my_value);
78              elcfields.put(new_cell,i,NY,k);
79      }
80      // ------------------------ Ordre 2 ------------------------- //
81
82      for(k=2;k<=NZ-2;k++)
83              for(i=3; i<=NX-2;i++)
84              {
85                      my_cell  = elcfields.get(i,2,k);
86                      new_cell  = elcfields.get(i,1,k);
87                      my_value =
88                              - prev2.get(i,2,k)
89                              + orbcs.cyd() * ( my_cell.cz()             +      prev2.get(i,1,k) )
90                              +orbcs.cyy() *  ( prev1.get(i,1,k)         +      prev1.get(i,2,k) )
91                              + orbcs.cyx() *( prev1.get(i+1,1,k) -    2.0 * prev1.get(i,1,k) +
prev1.get(i-1,1,k)
92                                                                       + prev1.get(i+1,2,k) -   2.0 *
prev1.get(i,2,k) + prev1.get(i-1,2,k) )
93                              + orbcs.cyz() *( prev1.get(i,1,k+1) -    2.0 * prev1.get(i,1,k) +
prev1.get(i,1,k-1)
94                                                                       + prev1.get(i,2,k+1) -   2.0 *
prev1.get(i,2,k) + prev1.get(i,2,k-1) );
95                      new_cell.setz(my_value);
96                      elcfields.put(new_cell,i,1,k);
97                      // ..................................................... //
98                      my_cell = elcfields.get(i,NY-1,k);
99                      new_cell = elcfields.get(i,NY,k);
100                     my_value =
101                              -prev2.get(i,3,k)
102                             + orbcs.cyd() *( my_cell.cz()             +      prev2.get(i,4,k) )
103                             + orbcs.cyy() * ( prev1.get(i,4,k)        +      prev1.get(i,3,k) )
104                             + orbcs.cyx() *( prev1.get(i+1,4,k) -    2.0 * prev1.get(i,4,k) +
prev1.get(i-1,4,k)
105                                                                      + prev1.get(i+1,3,k) -   2.0 *
prev1.get(i,3,k) + prev1.get(i-1,3,k) )
```

```
106                             + orbcs.cyz() * ( prev1.get(i,4,k+1) -    2.0 * prev1.get(i,4,k) +
prev1.get(i,4,k-1)
107                                                                 + prev1.get(i,3,k+1) -   2.0 *
prev1.get(i,3,k) + prev1.get(i,3,k-1) );
108                     new_cell.setz(my_value);
109                     elcfields.put(new_cell,i,NY,k);
110             }
111     // ------------ Sauvegarde des valeurs anterieures ------------- //
112
113     for (k=1; k<= NZ-1; k++)
114             for(i=2; i<=NX-1; i++)
115             {
116                     prev2.put(prev1.get(i,1,k),i,1,k);
117                     prev2.put(prev1.get(i,2,k),i,2,k);
118                     prev2.put(prev1.get(i,3,k),i,3,k);
119                     prev2.put(prev1.get(i,4,k),i,4,k);
120                     prev1.put(elcfields.get(i,1,k).cz(),i,1,k);
121                     prev1.put(elcfields.get(i,2,k).cz(),i,2,k);
122                     prev1.put(elcfields.get(i,NY-1,k).cz(),i,3,k);
123                     prev1.put(elcfields.get(i,NY  ,k).cz(),i,4,k);
124             }
125 }
```

ordi.h

```
1    #define _POLY
2
```

restore.h

```
1   // Restore
2   // Ce fichier contient les fonctions necessaires au
3   // retablissement de la simulation suite a un arret
4   // prevu ou imprevu.
5   // Retour de fonction:
6   // La fonction retourne TRUE si step==0 ou si time==0
7
8   bool restore_variables(int *dx,int *dy,int *dz,typ_coeff *csize,
9                                          typ_coeff *w_amp,int *step,typ_coeff *time,
10                                         char *rep,char *filename)
11  {
12      // L'ordre de lecture des parametres est le meme que l'ordre dans
13      // lequel ils ont ete enregistres.
14
15      BasicFileIO fichier(rep,filename,"r");
16
17      *dx=fichier.read_int();
18      *dy=fichier.read_int();
19      *dz=fichier.read_int();
20
21      *csize=fichier.read_coeff();
22      *w_amp=fichier.read_coeff();
23      *step=fichier.read_int();
24      *time=fichier.read_coeff();
25
26      fichier.close_file();
27      if(*step==1)
28              return(true);
29      else
30              return(false);
31  }
32
33
34  // Restore_space
35  // Cette fonction lit le fichier d'entree et assigne les valeurs des materiaux
36  // x,y et z a l'espace physique
37
38
39  void restore_space(espace_physique *ph_sp,
40                                  char *rep,char *filename)
41  {
42      int x,y,z;
43      int nx,ny,nz;
44      BasicFileIO fichier(rep,filename,"rb");
45      cellule_physique cell;
46
47      nx=ph_sp->nx();
48      ny=ph_sp->ny();
49      nz=ph_sp->nz();
50
51      for(z=1;z<=nz;z++)
52              for(y=1;y<=ny;y++)
53                      for(x=1;x<=nx;x++)
54                      {
55                              cell=ph_sp->get(x,y,z);
56                              cell.setx(fichier.read_int());
```

```
57                              cell.sety(fichier.read_int());
58                              cell.setz(fichier.read_int());
59                              ph_sp->put(cell,x,y,z);
60                      }
61
62      fichier.close_file();
63
64  }
65
66  // Restore_fields
67  // Cette fonction restaure la valeur des champs E et H de la derniere simulation.
68
69  void restore_field(espace_champs *ch,char *rep,char *filename)
70  {
71      int x,y,z;
72      int mx,my,mz;
73      BasicFileIO fichier(rep,filename,"rb");
74      cellule_champs cell;
75
76      mx=ch->nx();
77      my=ch->ny();
78      mz=ch->nz();
79
80      for(z=1;z<=mz;z++)
81      {
82              for(y=1;y<=my;y++)
83              {
84                      for(x=1;x<=mx;x++)
85                      {
86                              cell=ch->get(x,y,z);
87                              cell.setx(fichier.read_coeff());
88                              cell.sety(fichier.read_coeff());
89                              cell.setz(fichier.read_coeff());
90                              ch->put(cell,x,y,z);
91                      }
92              }
93      }
94
95      fichier.close_file();
96
97  }
98
99  // restore_plane_parameters
100 // Fonction qui reinitialise les parametres de sauvegarde des champs
101 // plan, hauteur, debut et fin.
102
103 void restore_plane_parameters(char *plan,int *hauteur,int *Stmn,int *Stmx,
104                                                 int *totstep,char *rep,char *filename)
105 {
106     BasicFileIO fichier(rep,filename,"r");
107
108     *plan=fichier.read_char();
109     *hauteur=fichier.read_int();
110     *Stmn=fichier.read_int();
111     *Stmx=fichier.read_int();
112     *totstep=fichier.read_int();
113
114     fichier.close_file();
115 }
```

```
116
117
118  // restore_orbc
119  // Fonction qui reinitialise les frontieres.
120
121  void restore_orbc(tab_for_orbc *orbc,char *rep,char *filename)
122  {
123      BasicFileIO fichier(rep,filename,"rb");
124      int x,y,z;
125      int mx,my,mz;
126      typ_coeff value;
127
128      mx=orbc->nx();
129      my=orbc->ny();
130      mz=orbc->nz();
131
132      for(z=1;z<=mz;z++)
133              for(y=1;y<=my;y++)
134                      for(x=1;x<=mx;x++)
135                      {
136                              value=fichier.read_coeff();
137                              orbc->put(value,x,y,z);
138                      }
139
140      fichier.close_file();
141  }
142
143  /*
144   Fonction work_dir()
145   Cette fonction lit les repertoires de travail a partir du fichier
146   dir.ini.
147  */
148
149  void work_dir(char *rep1,char *rep2,bool *newsim,char *filename)
150  {
151      char def_rep[]="";
152      char nsim;
153      BasicFileIO fichier(def_rep,filename,"r");
154
155      if(!fichier.read_string(rep1))
156              printf("d'ouverture de DIR.INI");
157      if(!fichier.read_string(rep2))
158              printf("d'ouverture de DIR.INI");
159      do{
160              nsim=fichier.read_char();
161      }while((toupper(nsim)!='O')&&(toupper(nsim)!='N'));
162
163      /*if((toupper(nsim)!='O')&&(toupper(nsim)!='N'))
164              nsim=fichier.read_char();
165      */
166
167      if(toupper(nsim)=='O')
168              *newsim=true;
169      else
170              *newsim=false;
171
172      printf("------- Recovery dir:  %s",rep1);
173      printf("------- Data dir:      %s",rep2);
174      if(*newsim)
```

```
175             printf("------- Demarrage d'une nouvelle simulation");
176     else
177             printf("------- Recouvrement d'une ancienne simulation");
178
179     fichier.close_file();
180 }
181
```

save_data.h

```
#define PRE_STR " %1.3e0"
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                          SAVE_DATA.H                               //
//                                                                    //
//          MODULE DE SAUVEGARDE SUR FICHIER DES RESULTATS OBTENUS    //
//                   POUR UN ECHANTILLON DE POSITIONS DE L'ESPACE     //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

// ************************
// Les questions a resoudre
// ************************
//
// Format de specification des donnees a sauvegarder ?
// Format de sauvegarde des donnees ? ( cf matlab)

/*
* ------------------------------------------------------------- *
* Fonction save_fieldZ(char* fname,espace_champs ch,...);       *
*                                                               *
* Cette fonction sauve n portions des plans X,Y. Les valeur de ces *
* plans seront les valeurs du champ magnetique qui pourront etre   *
* visualises avec la fonction QUIVER de matlab. La fonction QUIVER *
* de matlab permet d'afficher des fleches dans la direction du      *
* gradient d'un champ quelconque.
*                                                               *
* Retour: la fonction retourne TRUE si tout s'est bien deroule; *
*             elle retourne FALSE autrement.                    *
*           pour le plan Z
*                                                               *
* Auteur: Luc Romain, 15 juin 1998
*                                                               *
* ------------------------------------------------------------- *
*/
bool save_fieldZ(char* fname,espace_champs esp_ch,int xi,int yi,int xf,int yf,int z)
{
   int i,j;
   char name_str[25];
   FILE *fout;

   sprintf(name_str,"%s.m",fname);

   // Verification et validation du volume a sauvegarder
   if((xi<0)||(yi<0)||(z<0))
           return(false);

```

```
47      if(esp_ch.nx()<xf)
48              return(false);
49
50      if(esp_ch.ny()<yf)
51              return(false);
52
53      if(esp_ch.nz()<z)
54              return(false);
55
56      fout=fopen(name_str,"wb");
57
58      if(fout==NULL) // Erreur d'ouverture de fichier.
59      {
60              printf("d'ouverture de fichier.");
61              printf(": save_data.h");
62              printf(": save_field(...)");
63              return(false);
64      }
65
66      // Format de fichier MATLAB
67      fprintf(fout,"x=%d:%d;=%d:%d;",xi,xf,yi,yf);
68
69      // Ecriture de la matrice champs_x:
70
71      fprintf(fout,"c_x=[");
72      for(j=yi;j<=yf;j++)
73      {
74              for(i=xi;i<=xf;i++)
75              {
76                      fprintf(fout,PRE_STR,esp_ch.get(i,j,z).cx());
77              }
78              fprintf(fout,";"); // Changement de ligne
79      }
80
81      fprintf(fout,"];"); // Fin de la matrice champs_y;
82
83      // Ecriture de la matrice champy:
84
85      fprintf(fout,"c_y=[");
86      for(j=yi;j<=yf;j++)
87      {
88              for(i=xi;i<=xf;i++)
89              {
90                      fprintf(fout,PRE_STR,esp_ch.get(i,j,z).cy());
91              }
92              fprintf(fout,";"); // Changement de ligne
93      }

94
95      fprintf(fout,"];"); // Fin de la matrice champs_y;
96
97      // Ecriture de la matrice champy:
98
99      fprintf(fout,"c_z=[");
100     for(j=yi;j<=yf;j++)
101     {
102             for(i=xi;i<=xf;i++)
103             {
104                     fprintf(fout,PRE_STR,esp_ch.get(i,j,z).cz());
```

```
105             }
106             fprintf(fout,";"); // Changement de ligne
107     }

108
109     fprintf(fout,"];"); // Fin de la matrice champs_z;
110
111 /*
112 Fonction de generation des graphiques
113 */
114
115 // fprintf(fout,"quiver(x,y,c_x,c_y);");
116 // fprintf(fout,"title('Champ electrique (plan Z=%d)');",z);
117 // fprintf(fout,"pause(x,y,contr);");
118 // fprintf(fout,"title('Champ electrique (plan Z=%d)');",z);
119 // fprintf(fout,"surf(contr);");
120 // fprintf(fout,"title('Champ electrique (plan Z=%d)');",z);
121
122     fclose(fout);     // Fermeture du fichier;
123     return(true);
124 };
125
126 /*
127 * ------------------------------------------------------------- *
128 * Fonction save_fieldX(char* fname,espace_champs ch,...);       *
129 *                                                               *
130 * Cette fonction sauve n portions des plans X,Y. Les valeur de ces *
131 * plans seront les valeurs du champ magnetique qui pourront etre  *
132 * visualises avec la fonction QUIVER de matlab. La fonction QUIVER *
133 * de matlab permet d'afficher des fleches dans la direction du    *
134 * gradient d'un champ quelconque.
*
135 *                                                               *
136 * Retour: la fonction retourne TRUE si tout s'est bien deroule;   *
137 *             elle retourne FALSE autrement.                    *
138 *           pour le plan Z
                *
139 *                                                               *
140 * Auteur: Luc Romain, 15 juin 1998
*
141 *                                                               *
142 * ------------------------------------------------------------- *
143 */
144 bool save_fieldX(char* fname,espace_champs esp_ch,int yi,int zi,int yf,int zf,int x)
145 {
146     int j,k;
147     char name_str[25];
148     FILE *fout;
149
150     sprintf(name_str,"%s.m",fname);
151
152     // Verification et validation du volume a sauvegarder
153     if((yi<0)||(zi<0)||(x<0))
154             return(false);
155
```

```
156     if(esp_ch.ny()<yf)
157             return(false);
158
159     if(esp_ch.nz()<zf)
160             return(false);
161
162     if(esp_ch.nx()<x)
163             return(false);
164
165     fout=fopen(name_str,"wb");
166
167     if(fout==NULL) // Erreur d'ouverture de fichier.
168     {
169             printf("d'ouverture de fichier.");
170             printf(": save_data.h");
171             printf(": save_field(...)");
172             return(false);
173     }
174
175     // Format de fichier MATLAB
176     fprintf(fout,"y=%d:%d;=%d:%d;",yi,yf,zi,zf);
177
178     // Ecriture de la matrice champs_x:
179
180     fprintf(fout,"c_x=[");
181     for(k=zi;k<=zf;k++)
182     {
183             for(j=yi;j<=yf;j++)
184             {
185                     fprintf(fout,PRE_STR,esp_ch.get(x,j,k).cx());
186             }
187             fprintf(fout,";"); // Changement de ligne
188     }
189
190     fprintf(fout,"];"); // Fin de la matrice champs_x;
191
192     // Ecriture de la matrice champy:
193
194     fprintf(fout,"c_y=[");
195     for(k=zi;k<=zf;k++)
196     {
197             for(j=yi;j<=yf;j++)
198             {
199                     fprintf(fout,PRE_STR,esp_ch.get(x,j,k).cy());
200             }
201             fprintf(fout,";"); // Changement de ligne
202     }
203
204     fprintf(fout,"];"); // Fin de la matrice champs_y;
205
206     fprintf(fout,"c_z=[");
207     for(k=zi;k<=zf;k++)
208     {
209             for(j=yi;j<=yf;j++)
210             {
211                     fprintf(fout,PRE_STR,esp_ch.get(x,j,k).cz());
212             }
213             fprintf(fout,";"); // Changement de ligne
214     }
```

```
215
216     fprintf(fout,"];"); // Fin de la matrice champs_z;
217
218
219  /*
220  Fonction de generation des graphiques
221  */
222
223  // fprintf(fout,"quiver(x,y,c_x,c_y);");
224  // fprintf(fout,"title('Champ electrique (plan Z=%d)');",z);
225  // fprintf(fout,"pause(x,y,contr);");
226  // fprintf(fout,"title('Champ electrique (plan Z=%d)');",z);
227  // fprintf(fout,"surf(contr);");
228  // fprintf(fout,"title('Champ electrique (plan Z=%d)');",z);
229
230     fclose(fout);     // Fermeture du fichier;
231     return(true);
232  };
233
234  /*
235  * ------------------------------------------------------------- *
236  * Fonction save_fieldY(char* fname,espace_champs ch,...);       *
237  *                                                               *
238  * Cette fonction sauve n portions des plans X,Y. Les valeur de ces *
239  * plans seront les valeurs du champ magnetique qui pourront etre   *
240  * visualises avec la fonction QUIVER de matlab. La fonction QUIVER *
241  * de matlab permet d'afficher des fleches dans la direction du     *
242  * gradient d'un champ quelconque.
243  *                                                               *
244  * Retour: la fonction retourne TRUE si tout s'est bien deroule;    *
245  *             elle retourne FALSE autrement.                    *
246  *          pour le plan Z
247  *                                                               *
248  * Auteur: Luc Romain, 15 juin 1998
249  *                                                               *
250  * ------------------------------------------------------------- *
251  */
252  bool save_fieldY(char* fname,espace_champs esp_ch,int xi,int zi,int xf,int zf,int y)
253  {
254     int i,k;
255     char name_str[25];
256     FILE *fout;
257
258     sprintf(name_str,"%s.m",fname);
259
260     // Verification et validation du volume a sauvegarder
261     if((xi<0)||(y<0)||(zi<0))
262             return(false);
263
264     if(esp_ch.nx()<xf)
265             return(false);
266
```

```
267     if(esp_ch.ny()<y)
268             return(false);
269
270     if(esp_ch.nz()<zf)
271             return(false);
272
273     fout=fopen(name_str,"wb");
274
275     if(fout==NULL) // Erreur d'ouverture de fichier.
276     {
277             printf("d'ouverture de fichier.");
278             printf(": save_data.h");
279             printf(": save_field(...)");
280             return(false);
281     }
282
283     // Format de fichier MATLAB
284     fprintf(fout,"x=%d:%d;=%d:%d;",xi,xf,zi,zf);
285
286     // Ecriture de la matrice champs_x:
287
288     fprintf(fout,"c_x=[");
289     for(k=zi;k<=zf;k++)
290     {
291             for(i=xi;i<=xf;i++)
292             {
293                     fprintf(fout,PRE_STR,esp_ch.get(i,y,k).cx());
294             }
295             fprintf(fout,";"); // Changement de ligne
296     }
297
298     fprintf(fout,"];"); // Fin de la matrice champs_x;
299
300
301     fprintf(fout,"c_y=[");
302     for(k=zi;k<=zf;k++)
303     {
304             for(i=xi;i<=xf;i++)
305             {
306                     fprintf(fout,PRE_STR,esp_ch.get(i,y,k).cy());
307             }
308             fprintf(fout,";"); // Changement de ligne
309     }
310
311     fprintf(fout,"];"); // Fin de la matrice champs_y;
312
313
314     fprintf(fout,"c_z=[");
315     for(k=zi;k<=zf;k++)
316     {
317             for(i=xi;i<=xf;i++)
318             {
319                     fprintf(fout,PRE_STR,esp_ch.get(i,y,k).cz());
320             }
321             fprintf(fout,";"); // Changement de ligne
322     }
323
324     fprintf(fout,"];"); // Fin de la matrice champs_z;
325
```

```
326  /*
327  Generation du Contour Plot.
328  */
329  // fprintf(fout,"contr=sqrt(c_x.^2+c_y.^2);");
330
331  /*
332  Fonction de generation des graphiques
333  */
334
335  // fprintf(fout,"quiver(x,y,c_x,c_y);");
336  // fprintf(fout,"title('Champ electrique (plan Z=%d)');",z);
337  // fprintf(fout,"pause(x,y,contr);");
338  // fprintf(fout,"title('Champ electrique (plan Z=%d)');",z);
339  // fprintf(fout,"surf(contr);");
340  // fprintf(fout,"title('Champ electrique (plan Z=%d)');",z);
341
342     fclose(fout);      // Fermeture du fichier;
343     return(true);
344  };
345
346  /*
347  * ------------------------------------------------------------------- *
348  * Fonction:  save_plane(...)
349  *
350  * Cette fonction sauve toutes les valeurs du champ electrique et
351  * magnetique sur un plan donne.
352  *
353  * Auteur: Luc Romain
354  * le 14 septembre 1998
355  * ------------------------------------------------------------------- *
356  */
357
358  bool save_plane(char *rep,char plan,int alt,int Step,
359                              espace_champs ElcFields,espace_champs MagFields)
360
361  {
362
363     char file_name[25],mfname[25];
364     int Dx,Dy,Dz;
365     bool result;
366
367     Dx=ElcFields.nx();
368     Dy=ElcFields.ny();
369     Dz=ElcFields.nz();
370
371     switch(toupper(plan))
372             {
373             case 'X':
374                     {
375                             sprintf(file_name,"%se%i",rep,Step);
376                             sprintf(mfname,"%sh%i",rep,Step);
377                             result=save_fieldX(file_name,ElcFields,1,1,Dy,Dz,alt);
378                             //save_fieldX(mfname,MagFields,1,1,Dy,Dz,alt);
```

```
379                             break;
380                     }
381             case 'Y':
382                     {
383                             sprintf(file_name,"%se%i",rep,Step);
384                             sprintf(mfname,"%sh%i",rep,Step);
385                             result=save_fieldY(file_name,ElcFields,1,1,Dx,Dz,alt);
386                             //save_fieldY(mfname,MagFields,1,1,Dx,Dz,alt);
387                             break;
388                     }
389             case 'Z':
390                     {
391                             sprintf(file_name,"%se%i",rep,Step);
392                             sprintf(mfname,"%sh%i",rep,Step);
393                             result=save_fieldZ(file_name,ElcFields,1,1,Dx,Dy,alt);
394                             //save_fieldZ(mfname,MagFields,1,1,Dx,Dy,alt);
395                             break;
396                     }
397             } // switch
398     return result;
399 }
```

save_space.h

```
1
2   /*
3   // Save_field.h
4   //
5   // Ce fichier contient les instructions necessaires a la sauvegarde
6   // des informations relatives a FDTD. Ces fonctions seront necessaires
7   // pour effectuer du "Crash recovery".
8
9   // Fonction save_field
10  // Cette fonction sauvegarde toutes les valeurs des trois composantes
11  // de l'espace champs dans un fichier
12  */
13
14  void save_field(espace_champs *ch,char *rep,char *filename)
15  {
16      int x,y,z;
17      int mx,my,mz;
18      BasicFileIO fichier(rep,filename,"wb");
19      cellule_champs cell;
20
21      mx=ch->nx();
22      my=ch->ny();
23      mz=ch->nz();
24
25      for(z=1;z<=mz;z++)
26      {
27              for(y=1;y<=my;y++)
28              {
29                      for(x=1;x<=mx;x++)
30                      {
31                              cell=ch->get(x,y,z);
32                              fichier.write_to_file(cell.cx());
33                              fichier.write_to_file(cell.cy());
34                              fichier.write_to_file(cell.cz());
35                      }
36              }
37      }
38
39      fichier.close_file();
40
41  }
42
43
44  // save_space
45  // Fonction qui sauvegarde les valeurs des materiaux.
46
47  void save_space(espace_physique *ph_sp,char *rep,char *filename)
48  {
49      int x,y,z;
50      int mx,my,mz;
51      BasicFileIO fichier(rep,filename,"wb");
52      cellule_physique cell;
53
54      mx=ph_sp->nx();
55      my=ph_sp->ny();
56      mz=ph_sp->nz();
```

```
57
58
59      for(z=1;z<=mz;z++)
60              for(y=1;y<=my;y++)
61                      for(x=1;x<=mx;x++)
62                      {
63                              cell=ph_sp->get(x,y,z);
64                              fichier.write_to_file(cell.cx());
65                              fichier.write_to_file(cell.cy());
66                              fichier.write_to_file(cell.cz());
67                      }
68
69      fichier.close_file();
70
71  }
72
73
74  // Fonction save_variables
75  // Cette fonction sauve: Dimx,Dimy,Dimz,CellSize,WaveAmplitude,Step
76  void save_sim_parameters(int dx,int dy,int dz,typ_coeff csize,typ_coeff wave_amp,
77                                                  int step,typ_coeff time,char *rep,char *filename)
78  {
79      BasicFileIO fichier(rep,filename,"w");
80
81      fichier.write_to_file(dx);
82      fichier.write_to_file(dy);
83      fichier.write_to_file(dz);
84      fichier.write_to_file(csize);
85      fichier.write_to_file(wave_amp);
86      fichier.write_to_file(step);      // Sauvegarde du step courant.
87      fichier.write_to_file(time);
88
89      fichier.close_file();
90  }
91
92
93  // save_plane_parameters
94  // Fonction qui sauve le plan ainsi que la hauteur de sauvegarde.
95  // Cette fonction sauve egalement le pas minimal et maximal
96  void save_plane_parameters(char plan,int h,int stmn,int stmx,int end_step,
97                                                    char *rep,char *filename)
98  {
99      char str[4];
100     BasicFileIO fichier(rep,filename,"w");
101
102     sprintf(str,"%c0",plan);
103
104     fichier.write_to_file(str);
105     fichier.write_to_file(h);
106     fichier.write_to_file(stmn);
107     fichier.write_to_file(stmx);
108     fichier.write_to_file(end_step);
109
110     fichier.close_file();
111 }
112
113 // save_orbc
114 // Fonction qui sauve le plan ainsi que la hauteur de sauvegarde.
115 // Cette fonction sauve egalement le pas minimal et maximal
```

```
116 void save_orbc(tab_for_orbc *orbc,char *rep,char *filename)
117 {
118     BasicFileIO fichier(rep,filename,"wb");
119     int x,y,z;
120     int mx,my,mz;
121
122     mx=orbc->nx();
123     my=orbc->ny();
124     mz=orbc->nz();
125
126     for(z=1;z<=mz;z++)
127             for(y=1;y<=my;y++)
128                     for(x=1;x<=mx;x++)
129                     {
130                             fichier.write_to_file(orbc->get(x,y,z));
131                     }
132
133     fichier.close_file();
134 }
135
```

Source.h

```
1
2
```

tableaux.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                          TABLEAUX.H                                //
4   ////////////////////////////////////////////////////////////////////////
5
6
7   template <class type_elem>
8   class tableaux
9   {
10  protected:
11      // Dimensions selon les axes x,y,z.
12      int     NX, NY, NZ;
13
14      // Pointeur sur la zone de stockage
15      type_elem       *array;                 //      La zone de stockage est un tableau
16                                                      //
        unidimensionnel,quoique figurant
17                                                      //      un espace 3D.
18                                                      //      Ce tableau est
surdimensionne de 1,
19                                                      //      de telle sorte
qu'on n'utilisera
20                                                      //      jamais sa premiere
case (indicee par 0)
21      // Dimension de la zone de stockage ( ie NX*NY*NZ +1), en nb de cases
22      int             Dimension;
23
24  /***************************** FONCTIONS ****************************/
25  public:
26      // Constructeur
27      tableaux(int DimX=1, int DimY=1, int DimZ=1)
28      {
29              NX=DimX;
30              NY=DimY;
31              NZ=DimZ;
32              Dimension=NX*NY*NZ+1;
33              array= (type_elem*)malloc (Dimension* sizeof(type_elem));
34              if(array== NULL)
35              {
36                      printf("Echec d'allocation memoire");
37                      exit(1);
38              }
39              init((type_elem) 0);
40      };
41
42      // Destructeur
43      virtual         ~tableaux() {};
44
45      // Initialisation
46      void    init(type_elem value)
47      {
48              int i,j,k;
49              for (i=1;i<=NX;i++)
50                      for (j=1;j<=NY;j++)
```

```
51                                  for (k=1;k<=NZ;k++)
52                                          put( value,i,j,k);
53      };
54
55      // Accesseurs
56      inline int                          nx()    {return(NX);};
57      inline int                          ny()    {return(NY);};
58      inline int                          nz()    {return(NZ);};
59      inline int                          dim()   {return(Dimension);};
60      inline type_elem arr()    {return(*array);};
61
62      // Accesseur aux elements du tableau
63      inline type_elem get(int i, int j, int k)
64      {
65              int indice;
66              type_elem resu;
67              indice = adresse(i,j,k);
68              if      (adresseOK(indice))
69                      resu=array[indice];
70              else
71              {
72                      printf(" Tableau - Tentative de lecture abusive - Resultat sans
signification");
73                      getchar();
74              }
75              return (resu);
76      };
77
78      // Operateur d'affectation des elements du tableau
79      void    put(type_elem elem, int i, int j, int k)
80      {
81              int indice;
82              indice = adresse(i,j,k);
83              if (adresseOK(indice))
84                      array[indice]=elem;
85              else
86              {
87                      printf(" Tableau - Tentative d'ecriture abusive");
88                      getchar();
89              }
90      };
91
92      // Affichage des caracteristiques du tableau
93      void    descripteur()
94      {
95              printf(" Dimensions du tableau:");
96              printf(" *NX: %d*NY: %d*NZ: %d",NX,NY,NZ);
97              printf(" Dimension de la zone de stockage associee (en nb de cellules):");
98              printf(" *Dimension: %d",Dimension);
99      };
100
101     // Procedures d'affichage des valeurs contenues dans le tableau
102     void show_values();
103     void show_values_int();
104     void show_value(int i, int j, int k);
105     void show_valx (int i, int j, int k);
106     void show_valy (int i, int j, int k);
107     void show_valz (int i, int j, int k);
108
```

```
109 protected:
110
111     // Calcul d'une adresse en zone de stockage
112     inline int      adresse(int i, int j, int k)
113     {
114             int result;
115
116             if(((i>NX)||(j>NY)||(k>NZ))||((i*j*k)==0))
117                     result=-1; // Erreur d'acces au tableau
118             else
119                     result=1 + (i-1)*NY*NZ + (j-1)*NZ + (k-1);
120
121             return(result);
122     }
123
124     // Controle d'adresse en zone de stockage
125     inline bool     adresseOK(int val)
126     {
127             if((val>0)&&(val<=Dimension))
128                     return(true);
129             else
130                     return(false);
131     };
132 };
133
134
135
136 // Procedures d'affichage
137
138 template<class type>
139 void tableaux<type>::show_values()
140 {
141     int i,j,k;
142     for (i=1;i<=NX;i++)
143             for (j=1;j<=NY;j++)
144                     for (k=1;k<=NZ;k++)
145                     {
146                             printf("[%d][%d][%d]=",i,j,k);
147                             get(i,j,k).show_values();
148                             /*
149                             get(i,j,k) retourne un <type elem>.
150                             show_values() est une methode du type Elem3D
151                             qui affiche un type approprie.
152                             << Luc Romain, 15 juin 1998>>
153                             */
154
155                     }
156 };
157
158 void tableaux<long double>::show_value(int i, int j, int k)
159     {printf("[%d][%d][%d]= %.6f ",i,j,k,get(i,j,k));};
160
161 void tableaux<int>::show_value(int i, int j, int k)
162     {printf("[%d][%d][%d]= %d ",i,j,k,get(i,j,k));};
163
164 template <class type>
165 void tableaux<type>::show_valx(int i, int j, int k)
166     {printf("[%d][%d][%d]=",i,j,k); get(i,j,k).show_valx();};
167
```

```
168  template <class type>
169  void tableaux<type>::show_valy(int i, int j, int k)
170      {printf("[%d][%d][%d]=",i,j,k);   get(i,j,k).show_valy();};
171
172  template <class type>
173  void tableaux<type>::show_valz(int i, int j, int k)
174      {printf("[%d][%d][%d]=",i,j,k);   get(i,j,k).show_valz();};
```

type.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                      type.H        //
4   //                  Definition des types de donnees simples           //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7
8
9   typedef     int                             typ_material;
10  /*
11  typedef     long double             typ_champ;
12  typedef     long double             typ_phys_prm;
13  typedef long double         typ_coeff;
14  typedef     long double             typ_time;
15  typedef     long double             typ_distance;
16  */
17  typedef     double          typ_champ;
18  typedef     double          typ_phys_prm;
19  typedef double              typ_coeff;
20  typedef     double          typ_time;
21  typedef double              typ_distance;
```

type2.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                    type2.H         //
4   //          Instantiation des classes generiques elem3D et espaces    //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7
8     typedef elem3D       <typ_champ>                    cellule_champs;
9     typedef elem3D       <typ_material>          cellule_physique;
10    typedef espaces <cellule_champs>     espace_champs;
11    typedef espaces <cellule_physique>   espace_physique;
12    typedef tableaux <typ_champ>         tab_for_orbc;
```

update.h

```
1   // ------------------------------------------------------------------------
2   // aff_progres()
3   // ------------------------------------------------------------------------
4   // Cette fonction calcule combien de temps il reste avant la fin des
5   // calculs. Par la suite, elle affiche les resultats quant au progres
6   // effectue au cours de la simulation.
7   // ------------------------------------------------------------------------
8   void aff_progres(time_t *step1,int go,int Step,int st)
9   {
10              time_t step2,e_time;
11              typ_coeff h_left,m_left,s_left,secondes;
12
13              time(&step2);
14              e_time=(long)difftime(step2,*step1);
15              if(Step==go)
16                      secondes=1;
17              else
18                      secondes=(st-go)*e_time/(Step-go)*(st-Step)/(st);
19
20              h_left=floor(secondes/3600);
21              secondes-=h_left*3600;
22              m_left=floor(secondes/60);
23              secondes-=m_left*60;
24              s_left=secondes;
25
26              printf ("%i%% Done. [%i/%i]",(int)(100*(float)Step/(float)st),Step,st);
27              printf ("  -- %1.0fh%1.0fm%1.0fs remaining...",h_left,m_left,s_left);
28  };
```

var_env.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //
                                        //
3   //                                                  VAR_ENV.H
                                  //
4   //          DESCRIPTION ET MISE EN OEUVRE DE LA CLASSE 'VAR_ENV'                //
5   //
                                        //
6   ////////////////////////////////////////////////////////////////////////
7   class var_env
8   {
9    public:
10     typ_time                dt;
11     typ_distance     dx,dy,dz;
12
13     // Constructeurs
14     /* 1 */ var_env()        {dx=0;dy=0;dz=0;dt=0;};
15
16     /* 2 */ var_env(typ_distance sizex,typ_distance sizey,typ_distance sizez)
17     {
18             dx = sizex;
19             dy = sizey;
20             dz = sizez;
21             dt = (typ_time)(1.0 /  sqrt ((C*C/dx/dx)+(C*C/dy/dy)+(C*C/dz/dz)));
22
23  //         printf("_env: dx=%e,dy=%e,dz=%e,dt=%e",dx,dy,dz,dt);
24     };
25
26     // Destructeur
27     ~ var_env()      {};
28  };
```

var_env_comp.h

```
1   ////////////////////////////////////////////////////////////////////////
2   //                                                                    //
3   //                                                  VAR_ENV_COMP.H    //
4   //   DESCRIPTION ET MISE EN OEUVRE DE LA CLASSE 'VAR_ENV_COMP'        //
5   //                                                                    //
6   ////////////////////////////////////////////////////////////////////////
7   // Cette classe, derivee de la classe VAR_ENV, contient les           //
8   //  variables d'environnement et les constantes de calcul qui en      //
9   // decoulent.                                                         //
10  ////////////////////////////////////////////////////////////////////////
11
12
13  #define             costh    (cos(PI*thinc/180.0))
14  #define             sinth    (sin(PI*thinc/180.0))
15  #define             cosph    (cos(PI*phinc/180.0))
16  #define             sinph    (sin(PI*phinc/180.0))
17
18  class var_env_comp : public var_env
19  {
20
21  public:
22
23      typ_time time;
24
25      typ_coeff       thinc, phinc;
26      typ_coeff       ethinc, ephinc;
27      typ_coeff       offset,delay;
28      typ_coeff       alpha,beta;
29      typ_time period;
30      typ_coeff       xdisp,ydisp,zdisp;
31      typ_coeff       ampx,ampy,ampz;
32
33      // Constructeurs
34
35      /* 1 */ var_env_comp(){};
36
37      /* 2 */ var_env_comp(var_env env):var_env(env)
38              {
39              init(180.0 , 180.0 , 1.0 , 0.0 , 1.0 , 0.0000003 , 64.0);
40              init_alpha_and_period();
41              init_relative_spatial_delays();
42              ampx=ampy=ampz   =0.0;
43              };
44
45      /* 3 */ var_env_comp(    typ_distance delx,typ_distance dely,
46                                                      typ_distance delz, typ_coeff amp) :
var_env(delx,dely,delz)
47              {
48              init(180.0 , 180.0 , 1.0 , 0.0 , 1.0 , 0.0000003,64.0);
49              init_alpha_and_period();
50              init_relative_spatial_delays();
51              init_amplitude_of_incident_fields_components(amp);
```

```
52              };
53
54      /* 4 */ var_env_comp(     var_env env,typ_coeff amp): var_env(env)
55              {
56  //                      theta    phi      eth    eph         off    delay       beta              --LR
57              init(180.0 , 180.0 , 1.0 , 0.0 , 1.0 , 0.0000003,64.0);
58              init_alpha_and_period();
59              init_relative_spatial_delays();
60              init_amplitude_of_incident_fields_components(amp);
61              };
62
63      // Destructeur
64      -var_env_comp()  {};
65
66  private:
67
68      // Initialisation
69      void init(       typ_coeff theta, typ_coeff phi,
70                               typ_coeff etheta,          typ_coeff ephi,
71                               typ_coeff off,             typ_coeff del,
72                               typ_coeff bet)
73      {
74      thinc    = theta;
75      phinc    = phi;
76      ethinc   = etheta;
77      ephinc   = ephi;
78      offset   = off;
79      delay    = del;
80      beta     = bet;
81
82      };
83
84      void init_alpha_and_period(void)
85      {
86              alpha = (4.0/beta/dt) * (4.0/beta/dt);
87              period= 2.0 * beta * dt;
88      };
89
90      void init_relative_spatial_delays(void)
91      {
92              xdisp = (- cosph * sinth);
93              ydisp = (- sinph * sinth);
94              zdisp = (- costh);
95
96      }
97
98      void init_amplitude_of_incident_fields_components(typ_coeff amp)
99      {
100
101             ampx = amp * ( ethinc * costh * cosph - ephinc * sinph );
102             ampy = amp * ( ethinc * costh * sinph + ephinc * cosph );
103             ampz = amp * ( -ethinc * sinth );
104
105     };
106
107 public:
108     // Mise  jour du delay.
109     void maj_delay(int dimx, int dimy, int dimz)
110     {
```

```
111             delay=0.0;
112             if (xdisp<0.0) delay=delay-(xdisp*(dimx-1)*dx);
113             if (ydisp<0.0) delay=delay-(ydisp*(dimy-1)*dy);
114             if (zdisp<0.0) delay=delay-(zdisp*(dimz-1)*dz);
115     };
116 };
117
```

wiretest.h

```
1   class WireTest
2   {
3   protected:
4       int x,y,z;
5       int lon,sep1,sep2;
6       int index;
7       espace_champs *Efld;
8       espace_physique *Psp;
9
10
11  public:
12      WireTest(){};
13
14      ~WireTest(){};
15
16      WireTest(int px,int py,int pz,int l,int s1,int s2,
17              espace_champs *efld, espace_physique *psp)
18      {
19              Efld=efld;
20              Psp=psp;
21              x=px;
22              y=py;
23              z=pz;
24              lon=l;
25              sep1=s1;
26              sep2=s2;
27              index=4;
28              init();
29      }
30
31      // Construction de l'electrode simple de verification.
32
33
34      // Cette fonction retourne la valeur du potentiel lue sur la distance
35      // s1.
36      // Luc Romain, 5 mai 1999
37      typ_coeff deltav1(void)
38      {
39              int j;
40              cellule_champs cell;
41              typ_coeff val;
42
43              val=0;
44
45              for(j=y;j<y+sep1;j++)
46              {
47                      cell=Efld->get(x,j,z);
48                      val+=cell.cy();
49
50              }
51              return(val);
52      }
53
54      // Cette fonction calcule la somme des champs le long du fil.
55
56      typ_coeff delwir1(void)
```

```
57      {
58              int i;
59              cellule_champs cell;
60              typ_coeff val;
61
62              val=0;
63
64              for(i=x;i<x+lon;i++)
65              {
66                      cell=Efld->get(i,y,z);
67                      val+=cell.cx();
68
69              }
70              return(val);
71      }
72
73              // Cette fonction calcule la somme des champs le long du fil 2.
74
75      typ_coeff delwir2(void)
76      {
77              int i,j;
78              cellule_champs cell;
79              typ_coeff val;
80
81              val=0;
82
83              for(i=x;i<x+lon/2;i++)
84              {
85                      cell=Efld->get(i,y+sep1,z);
86                      val+=cell.cx();
87              }
88              for(i=x+lon/2;i<x+lon;i++)
89              {
90                      cell=Efld->get(i,y+sep2,z);
91                      val+=cell.cx();
92              }
93              for(j=y+sep1;j<y+sep2;j++)
94              {
95                      cell=Efld->get(x+lon/2,j,z);
96                      val+=cell.cy();
97              }
98              return(val);
99      }
100
101
102     // Cette fonction retourne la valeur du potentiel lue sur la distance
103     // s2.
104     // Luc Romain, 5 mai 1999
105     typ_coeff deltav2(void)
106     {
107             int j;
108             cellule_champs cell;
109             typ_coeff val;
110
111             val=0;
112
113             for(j=y;j<y+sep2;j++)
114             {
115                     cell=Efld->get(x+lon,j,z);
```

```
116                     val+=cell.cy();
117
118             }
119             return(val);
120     }
121
122  /* ------------------------------------------------------------------
123   * Fonction wire_x
124   * ------------------------------------------------------------------*
125   * Cette fonction trace un fil de xi a xf selon l'axe x.              *
126   * ------------------------------------------------------------------*
127   * Luc Romain, Janvier 1999
128   * ------------------------------------------------------------------*/
129     void wire_x(int xi,int xf,int y,int z)
130     {
131             cellule_physique cell;
132             int i,tmp;
133
134             if(xi>xf)
135             {
136                     tmp=xi;
137                     xi=xf;
138                     xf=tmp;
139             }
140
141             for(i=xi;i<xf;i++)
142             {
143                     cell=Psp->get(i,y,z);
144                     cell.setx(index);
145                     Psp->put(cell,i,y,z);
146             }
147     };
148  /* ------------------------------------------------------------------
149   * Fonction wire_y
150   * ------------------------------------------------------------------*
151   * Cette fonction trace un fil de yi a yf selon l'axe y.              *
152   * ------------------------------------------------------------------*
153   * Luc Romain, Mars 1999
154   * ------------------------------------------------------------------*/
155     void wire_y(int x,int yi,int yf,int z)
156     {
157             cellule_physique cell;
158             int j,tmp;
159
160             if(yi>yf)
161             {
162                     tmp=yi;
163                     yi=yf;
164                     yf=tmp;
165             }
166
167             for(j=yi;j<yf;j++)
168             {
169                     cell=Psp->get(x,j,z);
170                     cell.sety(index);
```

```
171                     Psp->put(cell,x,j,z);
172             }
173     };
174
175 /* ------------------------------------------------------------------
176  * Fonction wire_z
                    *
177  * ------------------------------------------------------------------*
178  * Cette fonction trace un fil de zi a zf selon l'axe z.                *
179  * ------------------------------------------------------------------*
180  * Luc Romain, Mars 1999
             *
181  * ------------------------------------------------------------------*/
182     void wire_z(int x,int y,int zi,int zf)
183     {
184             cellule_physique cell;
185             int k,tmp;
186
187             if(zi>zf)
188             {
189                     tmp=zi;
190                     zi=zf;
191                     zf=zi;
192             }
193
194             for(k=zi;k<zf;k++)
195             {
196                     cell=Psp->get(x,y,k);
197                     cell.setz(index);
198                     Psp->put(cell,x,y,k);
199             }
200     };
201
202     void init(void)
203     {
204             wire_x(x,x+lon,y,z);      // Brin superieur
205             /*
206                                     -----------
207             <-      1/2-> |<- 1/2 ->
208                                     |
209             ----------|                       s2
210             s1
211             ---------- // brin1
212
213    La difference de potentiel devrait etre la meme entre s1 et s2.
214
215              */
216
217             wire_x(x,x+lon/2,y+sep1,z);       // Brin2.
218             wire_y(x+lon/2,y+sep1,y+sep2,z);
219             wire_x(x+lon/2,x+lon,y+sep2,z);
220
221     }
222 };
```

Une disquette accompagne ce mémoire de maîtrise.

Toute personne intéressée à se la procurer doit contacter :

École Polytechnique de Montréal

Service de fourniture de documents

B.P. 6079, Succursale Centre-Ville

Montréal, Québec H3C 3A7

Canada

Tél. : (514) 340-4846
Télécopieur : (514) 340-4026