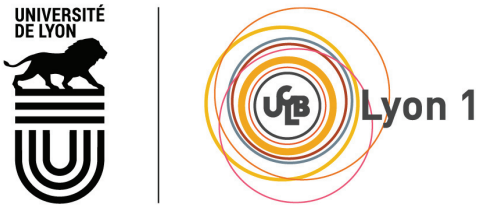
Simulations interactives de champ ultrasonore pour des configurations complexes de contrôle non destructif

Hamza Chouh



N° d'ordre NNT : 2016LYSE1220

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
l'Université Claude Bernard Lyon 1

École Doctorale ED512
InfoMaths

Spécialité de doctorat : Informatique

Soutenue publiquement le 22/11/2016, par :
Hamza Chouh

Simulations interactives de champ ultrasonore pour des configurations complexes de contrôle non destructif

---

Devant le jury composé de :

Bouatouch Kadi, IRISA	Rapporteur
Cassereau Didier, ESPCI ParisTech	Rapporteur
Cani Marie-Paule, Université de Grenoble INP, ENSIMAG	Examinatrice
Digne Julie, Université Claude Bernard Lyon I	Examinatrice
Ostromoukov Victor, Université Claude Bernard Lyon I	Directeur de thèse
Rougeron Gilles, CEA LIST	Encadrant CEA
Farrugia Jean-Philippe, Université Claude Bernard Lyon I	Co-directeur de thèse
Iehl Jean-Claude, Université Claude Bernard Lyon I	Co-directeur de thèse

Table des matières

Glossaire

API *Application Programming Interface*, ou interface de programmation applicative, est l'ensemble des méthodes, fonctions et classes par lequel une bibliothèque logicielle expose des services. 63, 64

AVX *Advanced Vector Extensions* (voir 319433-011 (2011)) est un jeu d'instructions *Central Processing Unit* ou processeur généraliste (CPU) datant de 2008 supporté par l'architecture *Sandy Bridge* d'Intel et à partir de 2011 par AMD avec l'architecture *Bulldozer* étendant les instructions SSE en largeur de 256 bits, soit 8 nombres à virgule flottante en simple précision ou 4 nombres à virgule flottante en double précision, tels que définis par la norme IEEE 754. 13, 62, 65, 66, 161, 164, 172–174, 176, 177

AVX2 *Advanced Vector Extensions 2* (voir 319433-012A (2012)) est un jeu d'instructions CPU datant de 2013 supporté par les architectures d'Intel à partir de *Haswell* et à partir de 2015 par AMD avec l'architecture *Carrizo*. Il étend les instructions AVX en ajoutant notamment les opérations de type FMA et certaines opérations entières issues du jeu SSE non couvertes par AVX. 13, 62, 164, 173, 175–177

AVX-512 *Advanced Vector Extensions-512* est un jeu d'instructions CPU datant de 2013 supporté en 2016 par les puces *Knights Landing* d'Intel. Il élargit le jeu d'instructions AVX à 512 bits, soit 16 nombres à virgule flottante en simple précision ou 8 nombres à virgule flottante en double précision, tels que définis par la norme IEEE 754 (voir IEEE (2008)). 62, 161, 164, 204

BVH *Bounding Volume Hierarchy* ou hiérarchie de boîtes englobantes. 10, 60–63, 140, 171

CND Contrôle Non Destructif. 9, 24–26

CPU *Central Processing Unit* ou processeur généraliste. 11, 21, 62–64, 89, 90, 160, 161, 166, 168, 170, 177, 182, 194, 197, 198, 200, 205

CUDA est une plateforme de calcul généraux sur *Graphics Processing Unit* (GPU). Elle permet d'effectuer des calculs parallélisables à grande échelle afin de bénéficier des nombreux cœurs de calcul présents sur un GPU. 64

dispersion est un phénomène intervenant lors de la propagation d'ondes. Il implique que la célérité de l'onde dépend de sa fréquence. Il est de faible am-

pleur dans le cas des ondes ultrasonores pour les plages de fréquences généralement utilisées en contrôle non destructif. Il peut donc être négligé dans le cadre de la modélisation de contrôles par ultrasons. 37

EMAT *ElectroMagnetic Acoustic Transducer*, ou traducteurs acoustiques électromagnétiques. 28

FMA *Fused Multiply Add*, extension d'instructions CPU incluse dans le jeu d'instructions AVX2 permettant la combinaison d'une opération de multiplication et d'une opération d'addition afin d'accélérer leur traitement. 173, 175

GPU *Graphics Processing Unit*. 21, 64, 90, 177, 182, 205

IEEE 754 est la norme de représentation de nombre à virgule flottante en binaire la plus utilisée pour les calculs impliquant des nombres réels. Un nombre flottant f est représenté comme suit :

$$f = s.m.2^e$$

On appelle m la mantisse et e l'exposant. s est le bit de signe (-1 ou +1). On distingue quatre représentations pour les nombres à virgule flottante dans cette norme :

— Simple précision : 32 bits incluant un bit de signe, 8 bits d'exposant et 23 bits de mantisse.
— Simple précision étendue ($\geq$ 43 bits, obsolète).
— Double précision : 64 bits incluant un bit de signe, 11 bits d'exposant et 52 bits de mantisse.
— Double précision étendue : 80 bits incluant un bit de signe, 15 bits d'exposant et 64 bits de mantisse.

Ces représentations ne sont exactes que pour 0 et des nombres égaux à des sommes de puissances de 2 et d'inverses de puissances de 2 dans les limites de la mantisse et de l'exposant. Pour les autres nombres, un arrondi est pratiqué, au plus proche lorsque le nombre est dans les bornes de la représentation ou vers 0, $+\infty$ ou $-\infty$ selon les cas. 76, 141

ITBB *Intel Thread Building Blocks*, une bibliothèque de haut niveau de parallélisation par tâches basée sur le principe du *work-stealing*, voir Intel Corporation (2016b).. 163, 168, 170, 195, 200

MBA *Multilevel B-spline Approximation*. 190

MIC *Many Integrated Core*. 62, 161, 182

MMX *MultiMedia eXtensions* est un jeu d'instructions CPU datant de 1997 supporté en premier lieu par les puces *Pentium MMX* d'Intel. Il définit 8 registres de 64 bits qui permettent d'exécuter des instructions sur paquets de 2 entiers de 32 bits, de 4 entiers de 16 bits ou de 8 entiers de 8 bits à la fois.. 161

POSIX *Portable Operating System Interface*, est un ensemble de normes techniques visant à standardiser les API des systèmes d'exploitation. 162

RAM *Random Access Memory*, la mémoire vive d'un ordinateur, utilisée pour le stockage des données de traitement. 160, 161

SAH *Surface Area Heuristic*. 60, 61

SIMD *Simple Instruction, Multiple Data*, mode d'instruction issu de la taxinomie de Flynn (voir Snyder (1988)) définissant l'exécution d'un même traitement sur des données multiples. 13, 62, 66, 89, 133, 161–165, 172–174, 176, 177, 194, 238

SSE *Streaming SIMD Extensions* (voir D91561-003 (2007)) est un jeu d'instructions vectorielles pour CPU datant de 1999 utilisé à partir des architectures *Pentium II* d'Intel et *Athlon XP* d'AMD. Il contient des instructions pour des registres d'une largeur maximale de 128 bits pour la gestion simultanée de 4 nombres à virgule flottante en simple précision tels que définis par la norme IEEE 754. Il a été enrichi à plusieurs reprises :

— *SSE2* en 2001 complète *SSE* en y ajoutant 144 instructions, permettant notamment l'usage de registres contenant deux flottants à double précision, 2 entiers de 64 bits, 4 entiers de 32 bits, 8 entiers de 16 bits ou encore 16 entiers de 8 bits.

— *SSE3* en 2004 ajoute à *SSE2* des instructions de traitement horizontal des registres, implémentant de façon matérielle des opérations arithmétiques de réduction.
Exemple :
$$\{a, b, c, d\} \rightarrow \{a + b, c + d\}$$

— *SSE4* en 2007 complète *SSE3* en ajoutant 54 opérations (47 pour *SSE4.1* et 7 pour *SSE4.2*) non dédiées au multimédia, contrairement aux extensions précédentes.

Il existe également une extension *SSE4a* propre à AMD sur l'architecture *Barcelona*. 62, 65, 66, 164, 174

SSIM *Structural SIMilarity*, voir Wang et collab. (2004). 189

VTK *Visualisation ToolKit*. 200

Table des figures

Liste des tableaux

Remerciements

Aboutissement de trois années de recherche et compilation de mes résultats de thèse, ce document a également vocation à narrer le cheminement intellectuel m'ayant mené de l'étape zéro de ces travaux à leur achèvement. Ces trois dernières années ont été enrichissantes à de nombreux égards et leur fin ne fait que marquer le début d'un autre cycle d'apprentissage que j'espère aussi gratifiant. Puisque ces travaux sont loin d'être le fait de ma seule personne, j'entends, au cours des quelques paragraphes à venir, faire la part belle aux remerciements en tous genres et tenter, autant que faire se peut, de n'oublier personne.

Pour commencer, je remercie Gilles Rougeron pour son excellent travail d'encadrement, ses compétences scientifiques et techniques, son extrême disponibilité et sa bonne humeur de tous les instants au cours de ces trois années. J'adresse également mes sincères remerciements à Jean-Claude Iehl et Jean-Philippe Farrugia pour le partage de leur expertise en informatique graphique et leur implication, ainsi qu'à Victor Ostromoukhov pour la direction de ma thèse et ses conseils toujours pertinents.

Pour l'intérêt et le soutien qu'ils ont apporté à mes travaux, je remercie Stéphane Leberre et Vincent Bergeaud, chefs successifs du LDI, ainsi que Sylvain Chatillon, chef du LSMA. Pour leurs interventions et leur aide à divers moments de ma thèse, je remercie également Philippe Ribier, Nicolas Leymarie et Thibaud Fortuna. Je n'oublie pas mes cobureaux successifs, avec qui j'ai apprécié de partager le bureau 2005 : Jason Lambert pour son goût de la fraîcheur et ses nombreuses idées, Loïc « Le Mordu » Sifferlen pour son humour et son aide lors de mes quelques plongées dans le monde impitoyable du Java, Michaël Roynard malgré sa sensibilité au froid et pour tous ces échanges techniques enrichissants, Raphaël Loyet pour sa sympathie, ses nombreux conseils essentiels et son implication lors de ses passages au laboratoire, et, plus récemment, Carlos Carrascal à qui je souhaite la réussite pour ses propres travaux de thèse.

De même, pour leur accueil, leur amitié, leur humour et leur patience à table, je remercie toute la tablée des thésards et stagiaires du DISC, les anciens comme les nouveaux. Plus généralement, je souhaite adresser mes plus sincères remerciements à l'ensemble des membres du DISC, trop nombreux pour être tous cités, mais pas assez pour être oubliés.

Merci également à l'ensemble de mes amis, parmi lesquels je souhaite notamment citer l'original Mustafa Ben Azzouz, l'aventurier Riadh Niati, le sage Bachir Chraïbi et ceux qui sont également mes compagnons de marche usuels, à savoir le

musicien célèbre Florestan Labey, le relativiste en devenir Jean-David Pailleron et l'imprévisible Aurélien Soulié. À ces derniers, qui se sont engagés ou s'engageront bientôt dans la folle aventure du doctorat : c'est un périple de longue haleine, les difficultés sont nombreuses, mais vous les surmonterez et le résultat vaut le détour !

Ma gratitude et mes remerciements les plus sincères vont à ma famille pour leur soutien inconditionnel en toutes circonstances, tout partculièrement à mes parents dont l'exemple et l'éducation ont largement contribué à éveiller chez moi cette passion pour les sciences et en particulier l'informatique, dès mon plus jeune âge sur les deux vieux 486 avec MSDOS et Windows 3.1. J'ai depuis lors tant appris et découvert et, même s'il reste tant à faire et à voir, pour ce que j'ai pu accomplir aujourd'hui, je vous suis reconnaissant.

Un autre remerciement me tient particulièrement à cœur, il est destiné à l'ensemble des enseignants dont j'ai pu suivre les cours pendant ma scolarité : ceux qui ont fait de leur objectif l'éveil et l'éducation d'une génération et qui, chaque jour, contribuent à construire les individus de demain.

Je termine enfin cette série par un ensemble de témoignages de gratitude, envers les grands esprits de toutes les époques, qui ont contribué à faire de la science l'édifice bancal mais ô combien sublime qu'il est aujoud'hui, envers les plumes qui se sont acharnées à construire ces myriades de mondes imaginaires dans lesquels nos esprits s'évadent pour mieux penser, envers les quarante jours de Brubeck et toutes ces notes à entendre, envers les sentiers visités et ceux qui restent à explorer, envers les merveilles qui m'ont ébloui et qui n'en finiront probablement jamais, ces admirables nouveautés que j'aime imaginer si sublimes…

Merci.

Avant-propos

Ce document présente un certain nombre d'algorithmes utilisés pour la simulation rapide de champ ultrasonore. Dans un objectif de clarté, nous avons choisi d'adopter la méthodologie de la **programmation littérale** introduite par Donald Knuth (1984). Cette méthodologie a notamment été utilisée par Pharr et Humphreys (2004), dont nous nous sommes inspirés pour la rédaction de cette thèse.

Ainsi, en plus de présenter le déroulement de certains algorithmes essentiels, nous proposons des portions de code suivant une syntaxe proche de celle du C++ dont une partie des détails d'implémentation a été extraite afin d'en simplifier la lecture. Sans aller jusqu'à détailler la totalité du code développé au cours de cette thèse, nous ré-utilisons le système de référencement et de fragments de code de Pharr et Humphreys (2004).

Ces portions de code sont identifiées par un nom et, lorsqu'elles font appel à d'autres portions définies ailleurs, comportent une mention en marge indiquant le nom de ces portions et leur emplacement dans le document. La structure générale adoptée suit l'exemple suivant, pour une structure de données :

⟨*Vecteur3D*⟩≡

```
struct Vecteur3D {
   float x;
   float y;
   float z;
}
```

De la même manière, nous définissons des fonctions comme suit :

⟨*vecteurOrthogonal*⟩≡

```
Vecteur3D vecteurOrthogonal(Vecteur3D vecteur) {
   Vecteur3D orthogonal = produitVectoriel(vecteur, VecteurAxeZ);
   if(orthogonal.estNul()) {
      orthogonal = VecteurAxeX;
   }
   return orthogonal;
}
```

Notons que certaines fonctions, dont le détail ne nous semble pas utile à la compréhension des algorithmes proposés, sont utilisées sans être détaillées.

Première partie

Contexte

Introduction

Le contrôle non destructif est un ensemble de méthodes visant à contrôler l'intégrité d'une pièce mécanique sans l'altérer. Le domaine d'application de ces méthodes est très large, incluant des secteurs critiques tels que le transport, l'aérospatial, l'énergie ou la construction.

Les phénomènes physiques exploités pour le contrôle non destructif font notamment intervenir des ondes mécaniques (ultrasons) et électromagnétiques (rayon X, rayons γ, courants de Foucault...). L'émission et la réception de telles ondes permettent, au moyen d'outils d'analyse, de déduire des informations sur la structure interne des pièces contrôlées grâce aux phénomènes d'interaction des ondes avec la matière.

Ces phénomènes pouvant être complexes, les inspections nécessitent à la fois un travail de **conception** des tests et d'**analyse** de leurs résultats. L'utilisation d'outils de simulation physique simplifie grandement ces deux tâches en permettant de prévoir le déroulement d'un contrôle et de mieux comprendre les résultats des mesures effectuées.

La plate-forme logicielle *CIVA*, développée par le Département d'Imagerie pour la Simulation et le Contrôle du CEA-LIST, fournit un ensemble d'outils de simulation et d'analyse facilitant le travail de conception et d'inspection. Elle se base sur un ensemble de modèles physiques adaptés aux différents contextes de contrôle et aux diverses méthodes.

Bien que ces modèles physiques permettent l'exploitation de méthodes semi-analytiques plus performantes que les méthodes numériques pures pour la simulation, certaines simulations, notamment de contrôle par ultrasons, peuvent nécessiter des temps de calcul importants.

Le besoin croissant en outils de simulation toujours plus rapides et l'apport potentiel de simulations interactives de contrôle non destructif a mené le CEA-LIST à entreprendre des recherches portant à la fois sur des optimisations algorithmiques des modèles existants et sur une exploitation plus efficace des processeurs de calcul modernes. Ainsi, Pedron (2013) a développé des algorithmes rapides s'exécutant sur GPU et CPU pour l'imagerie et la reconstruction à partir d'acquisitions de capteurs à ultrasons pour le contrôle non destructif. Lambert (2015) a quant à lui utilisé un modèle de simulation d'ondes ultrasonores adapté à des configurations de contrôle simples (pièces planes, matériaux isotropes, inspections directes et avec un rebond) pour tirer parti des architectures de calcul modernes et arriver à des simulations

interactives de champ ultrasonore.

Les travaux que nous allons présenter visent l'extension des outils de simulation rapides à des configurations plus complexes (matériaux anisotropes, géométries variées, modes d'inspection quelconques). Les temps de calcul sur ces configurations ont été considérablement réduits et un outil de visualisation interactive de champ ultrasonore a été développé pour étendre l'interactivité à des simulations présentant des temps de calcul supérieurs au plafond de l'interactivité.

La démarche employée pour parvenir à cette fin a dans un premier temps consisté en une étude approfondie des phénomènes physiques régissant la propagation des ondes ultrasonores pour adapter le modèle utilisé dans *CIVA*, que nous détaillons dans le chapitre 2 à l'exploitation d'outils performants de lancer de rayons issus du domaine du rendu photo-réaliste. Cela a conduit à la réalisation d'un outil complet et rapide de suivi de rayons d'onde ultrasonore dans des milieux hétérogènes, isotropes et anisotropes, dont les principes sont détaillés dans le chapitre 3.

La suite de ces travaux a consisté en l'utilisation de cet outil pour un processus de reconstruction des fronts d'ondes ultrasonores au moyen d'un échantillonnage par rayons. Au moyen de méthodes d'interpolation à précision contrôlée tirant parti de la régularité des fronts d'ondes ultrasonores et détaillées dans le chapitre 4, les grandeurs physiques utiles au calcul des valeurs du champ ultrasonore peuvent être calculées à un coût très inférieur à celui d'un lancer de rayons, accélérant nettement le processus de simulation.

Exploitant directement les résultats du chapitre 4, un algorithme de calcul des réponses d'un milieu de propagation au passage d'une impulsion mécanique a été réalisé. Il simplifie et rend plus régulier le calcul de champ ultrasonore au moyen d'une méthode d'inversion et d'un échantillonnage régulier de la surface du traducteur ultrasonore. Nous en expliquons le fonctionnement dans le chapitre 5.

Des optimisations algorithmiques, que le chapitre 6 explicite, réduisent sensiblement le nombre de rayons nécessaires à la recherche des zones de visibilité du traducteur et à la construction d'interpolateurs efficaces des grandeurs physiques des rayons ultrasonores. Leur utilisation améliore les performances du calcul de champ ultrasonore.

Suite à cela, un travail d'optimisation architecturale visant à exploiter plus efficacement les capacités de calcul parallèle de processeurs généralistes modernes a été mené. Le chapitre 7 montre comment le parallélisme de tâches et le parallélisme de données ont accéléré le processus de simulation.

Enfin, nous proposons dans le chapitre 8 une méthode de calcul progressif de champ applicable à des images présentant des propriétés de régularité. Cette méthode se sert d'outils de reconstruction d'images pour fournir une visualisation du champ à partir d'échantillonnages lacunaires et pour guider les calculs dans un objectif de raffinement optimisé de l'image. Les problématiques liées à la parallélisation de cette méthode sur des processeurs multi-cœurs sont abordées et solutionnées au cours du chapitre 9.

CHAPITRE 1

Contrôle non destructif

Ce chapitre a pour objectif d'expliciter le contexte dans lequel les travaux de simulation interactive de champ ultrasonore se sont inscrits. Nous y présentons le contrôle non destructif, ses applications et les problématiques de simulation qui y sont associées.

1.1 Un besoin de l'industrie

Dans de nombreux domaines tels que l'industrie, les transports, l'aérospatial ou la production d'énergie, les conséquences potentielles en cas de défaillance d'un système induisent des exigences de sécurité particulières.

1.1.1 Le nucléaire civil

Dans le cas de l'industrie nucléaire civile, pour assurer la production massive d'énergie électrique, les centrales nucléaires entretiennent une réaction de fission atomique d'éléments radioactifs lourds tels que l'uranium et le plutonium. Cette réaction produit une chaleur qui est transformée en énergie mécanique au moyen de chaudières et en énergie électrique grâce à des systèmes inductifs.

Cette suite de transformations d'énergie fait intervenir de nombreux composants mécaniques dont la défaillance causerait un arrêt de fonctionnement du système. D'autres composants, incluant les tuyaux convoyant les fluides chauffés par le combustible radioactif, présentent un danger majeur. En effet, toute fuite de fluide radioactif pourrait entraîner une contamination à grande échelle de la population avoisinante.

Pour cette raison, il est crucial d'avoir des garanties sur l'intégrité des pièces utilisées dans un réacteur nucléaire pour pouvoir effectuer les opérations de maintenance en prévention d'accidents. De plus, il est préférable d'éviter l'arrêt du fonctionnement d'une centrale nucléaire tout en effectuant les contrôles nécessaires.

1.1.2 Les transports et l'aérospatial

De la même façon, que ce soit pour les transports ferroviaires, aériens ou automobiles, les défaillances mécaniques peuvent avoir des conséquences catastrophiques sur la sécurité des passagers et des personnes autour du dispositif de transport. Dans le cas du domaine aérospatial, les missions, même inhabitées, représentent un investissement considérable et une défaillance mécanique peut causer la perte totale d'une mission. Ainsi, il est également essentiel d'être en mesure de vérifier l'intégrité des composants de divers moyens de transport, que ce soit avant, pendant ou après leur utilisation.

De manière plus générale, on effectuera des contrôles sur une pièce pour deux raisons principales :

— Le coût de production de la pièce est élevé et le coût de l'inspection est inférieur au risque causé par une absence de contrôle,
— La fiabilité de la pièce est essentielle pour des raisons de sécurité.

1.2 Principe de base du contrôle non destructif

L'ensemble des méthodes visant à contrôler la présence de défauts dans des pièces mécaniques de manière non intrusive et conservative de l'intégrité du système inspecté sont regroupées sous le terme CND. Ces méthodes se basent sur un principe

commun : il s'agit d'émettre une onde dans la zone à contrôler. En interagissant avec la structure interne de la pièce, elle se trouve modifiée. La mesure des valeurs d'amplitude qu'elle prend après sa propagation peut permettre de conclure quant à la présence ou l'absence d'imperfections dans la pièce. Le schéma présenté en figure 1.1 résume ce principe de base.

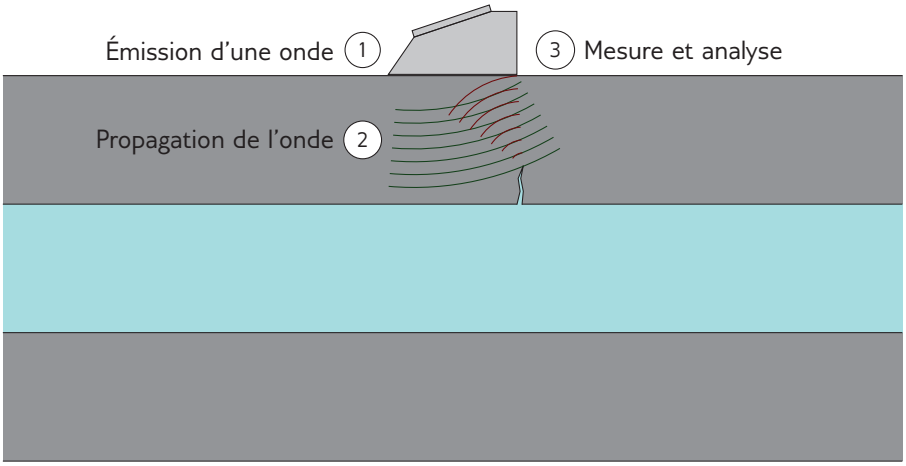


Figure 1.1 – Schématisation du principe de base du CND

En plus de permettre de détecter des défauts, il est possible dans certaines situations d'estimer des paramètres du défaut tels que son type, sa position, ses dimensions, son orientation. De cette façon, il est possible de déterminer son degré de gravité et de décider s'il est nécessaire ou non d'entreprendre une opération de maintenance.

1.3 Différents types de contrôles

En fonction des matériaux à inspecter et des conditions de contrôle, différents types de contrôles faisant intervenir plusieurs types d'ondes peuvent être utilisés, parmi lesquels le contrôle par radiographie, par courants de Foucault ou par ondes ultrasonores.

1.3.1 Contrôle par radiographie

Dans le cas du contrôle par radiographie, l'onde mise en jeu peut être un rayonnement de type X ou γ. La pièce à contrôler est soumise à une irradiation d'un côté et un détecteur est placé de l'autre côté. En fonction de la composition du matériau traversé, les rayonnements émis subissent une atténuation plus ou moins prononcée. En mesurant les amplitudes reçues en chaque point de sa surface, le détecteur permet de produire une image de contraste qui illustre la distribution d'atténuation des rayons.

L'atténuation étant directement liée à l'épaisseur et aux propriétés des matériaux traversés par chaque rayon, il est possible d'extraire de l'image de contraste obtenue des informations sur la structure interne de la pièce inspectée.

Cette méthode fournit une imagerie de contraste fine et précise tout en ne nécessitant que peu de traitements sur les données mesurées, bien que l'interprétation des images obtenues puisse s'avérer complexe. Par ailleurs, en raison du danger présenté

par une exposition aux rayons γ ou X, il est nécessaire de garantir des conditions de sécurité pour l'opérateur du contrôle.

En pratique, la nécessité d'accéder à deux côtés de la pièce à inspecter peut être contraignante dans le cas de structures à accessibilité réduite. La radiographie reste néanmoins une méthode très utilisée pour le CND, notamment dans les centrales nucléaires ou pour le contrôle de pièces industrielles complexes. Dans le cadre de la conception de tests utilisant d'autres techniques, elle peut aussi servir de référence.

1.3.2 Contrôle par courants de Foucault

Dans le cas d'une pièce à contrôler composée d'un matériau conducteur, une variation contrôlée du champ magnétique proche entraîne l'apparition de courants électriques appelés **courants de Foucault**. Ces courants induisent eux-mêmes un champ magnétique que l'on peut mesurer à l'aide d'une bobine (souvent, la bobine émettrice sert également de récepteur).

La présence d'un défaut dans la pièce inspectée entraîne une perturbation des courants de Foucault, ce qui se traduit par une modification du champ magnétique mesuré par la bobine. Cette variation peut être mesurée et, par un processus d'inversion, il est à la fois possible de détecter et de dimensionner le défaut avec une précision variable.

Ces contrôles peuvent être effectués sans nécessiter de contact avec la pièce à inspecter, ce qui les rend plus simples à mettre en œuvre que les contrôles par radiographie. De plus, le processus de contrôle par courants de Foucault peut être automatisé, ce qui facilite son utilisation dans des conditions difficiles pouvant impliquer des radiations, de hautes températures ou pressions, etc. Il n'est cependant applicable qu'au contrôle de matériaux conducteurs, ce qui le restreint au contrôle de pièces métalliques.

1.3.3 Contrôle par ultrasons

Le contrôle par ultrasons se sert des phénomènes de réflexion, de transmission, d'absorption et de diffraction des ondes élastodynamiques pour analyser la structure d'une pièce. Pour effectuer un contrôle par ultrasons, on émet une onde dans la pièce à l'aide d'un **traducteur ultrasonore**. Il s'agit le plus souvent d'un dispositif piézoélectrique transformant les impulsions électriques reçues en entrée en vibrations.

L'onde émise par le traducteur se propage, subit des interactions avec des interfaces, est atténuée et peut être diffractée. Lorsqu'elle rencontre un défaut, l'onde interagit avec celui-ci, parfois de manière complexe et produit un **écho**. La mesure des temps d'arrivée de ces échos permet, connaissant la vitesse de propagation de l'onde dans le milieu inspecté, d'estimer la distance au défaut. Des analyses plus poussées des résultats ainsi que des contrôles plus sophistiqués offrent la possibilité de caractériser les défauts.

Les techniques de contrôle basées sur les ultrasons nécessitent pour la plupart un accès direct ou en immersion à la pièce à inspecter, bien que des méthodes hybrides utilisant des lasers ou des EMAT (*ElectroMagnetic Acoustic Transducer*) permettent de s'affranchir de cette limitation.

Les ondes ultrasonores présentant une atténuation moins marquée que les ondes électromagnétiques, elles permettent d'inspecter à une plus grande profondeur et, grâce aux échos, à des distances importantes. Dans le cadre de ces travaux, nous nous sommes intéressés spécifiquement au contrôle par ultrasons.

1.4 Contrôle non destructif par ultrasons

Nous présentons dans cette section quelques fondements du contrôle non destructif afin d'introduire les bases de ce que nous cherchons à simuler. Nous y introduisons également quelques éléments de vocabulaire nécessaires qui seront utilisés par la suite.

1.4.1 Scène de contrôle non destructif

Les éléments de base d'une scène de contrôle non destructif par ultrasons comportent :

— Un dispositif d'émission d'ondes ultrasonores, qui sert souvent aussi de dispositif de mesure, le capteur ultrasonore. Il existe divers types de capteurs que nous détaillerons.
— Une pièce à contrôler dont les caractéristiques sont connues (type de matériau, densité, vitesse de propagation des ondes...).
— Le milieu dans lequel la pièce est immergée, ou milieu couplant, qui servira notamment à des contrôles avec un traducteur sans contact direct avec la pièce.

Un exemple de scène de contrôle non destructif est présenté en figure 1.2.

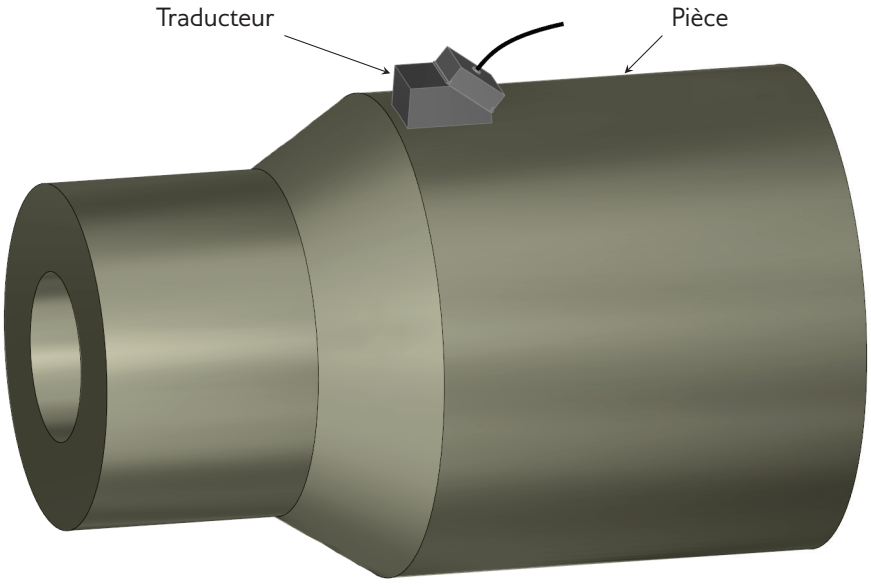


Figure 1.2 – Exemple de scène de contrôle non destructif

1.4.2 Capteurs ultrasonores

Principe de fonctionnement

Un capteur ultrasonore est un dispositif électromécanique dont les fonctions principales sont l'émission et la réception d'ondes ultrasonores. Il est composé d'un ou plusieurs traducteurs ultrasonores qui transforment une impulsion électrique en une excitation mécanique. Le fonctionnement inverse est aussi possible avec certains types de traducteurs, selon les phénomènes physiques utilisés pour engendrer l'excitation mécanique.

L'émission ou la réception d'ondes ultrasonores peuvent notamment se baser sur les techniques suivantes :

— L'effet piézo-électrique, ou la propriété qu'ont certains matériaux de se déformer lorsqu'ils sont soumis à un champ électrique et, inversement, de se polariser électriquement sous l'action d'une contrainte mécanique. On retrouve parmi ces matériaux le quartz (SiO_2), certains semi-conducteurs ou encore des structures cristallines telles que le tungstène-bronze ou le niobate de lithium.
— Des effets électromagnétiques tels que les forces de Lorentz au moyen de traducteurs dits *ElectroMagnetic Acoustic Transducer*, ou traducteurs acoustiques électromagnétiques (EMAT). Ceux-ci permettent de contrôler une pièce mécanique sans contact au moyen d'une excitation électromagnétique volumique. Ils sont constitués de bobines et d'aimants permanents : un courant électrique dans une bobine génère des courants de Foucault dans le matériau, ainsi qu'un champ magnétique dynamique se superposant au champ magnétique statique induit par le(s) aimant(s) permanent(s). L'interaction de ces champs magnétiques donne naissance aux forces volumiques dites de Lorentz, à l'origine des perturbations mécaniques.
— L'utilisation de lasers pour chauffer la pièce à contrôler et provoquer des dilatations du matériau. Le laser peut également être utilisé pour mesurer les perturbations mécaniques longitudinales à la surface d'une pièce.

Les capteurs les plus utilisés en contrôle non destructif par ultrasons sont ceux composés de traducteurs piézo-électriques en raison de leurs bandes passantes adaptées aux fréquences couramment utilisées pour la détection de défauts. De plus, ils permettent de combiner émission et réception en un même composant, sont peu coûteux, peu encombrants et simples d'utilisation.

Typologie des capteurs ultrasonores

Suivant les conditions des contrôles effectués, le capteur ultrasonore peut être placé au contact de la pièce ou à distance. Pour les traducteurs piézo-électriques, puisque le point de départ de l'onde ultrasonore est le capteur lui-même, le milieu environnant est déterminant dans le choix de la position du capteur.

En effet, si un contrôle est effectué à l'air libre, les propriétés de l'air à la pression atmosphérique font que les ondes ultrasonores se propagent à une vitesse avoisinant les 340 $m.s^{-1}$. Cette vitesse, inférieure d'un ordre de grandeur à celle de ces mêmes ondes dans des matériaux usuellement contrôlés tels que l'acier ou l'aluminium, empêche une transmission efficace des ondes entre l'air et le matériau contrôlé. Pour un contrôle à l'air libre, il est donc nécessaire d'apposer le capteur au contact de la pièce. Pour assurer une meilleure transmission de l'onde ultrasonore, on interpose généralement un gel couplant entre les deux.

Lorsque cela est possible, immerger la pièce à contrôler dans un milieu fluide (dit milieu **couplant**) permet d'effectuer un contrôle à distance. En effet, la différence de vitesse entre les milieux couplants utilisés et les pièces contrôlées est plus faible, ce qui permet de transmettre suffisamment d'énergie dans la pièce pour obtenir une réponse exploitable. Le contrôle par immersion permet notamment de couvrir de plus larges zones de la pièce inspectée simplement en changeant l'orientation du capteur.

Capteurs mono et multi-éléments

Le capteur piézo-électrique le plus simple concevable est constitué d'une unique membrane vibrante piézo-électrique. Bien que peu de paramètres soient ajustables lors des contrôles effectués avec de tels capteurs, ils sont utilisés pour des opérations d'inspection simples.

Bien souvent en contrôle non destructif, l'opérateur cherche à viser une zone particulière de la pièce inspectée, connue pour sa tendance à présenter des défauts. Par exemple, certains points soumis à de fortes contraintes ou les soudures sont régulièrement inspectés. Il est donc utile de pouvoir concentrer le faisceau d'ultrasons émis dans ces zones.

Pour ce faire, il est possible d'adjoindre une lentille sur la surface émettrice d'un capteur mono-élément. La conception de celle-ci permet, de la même manière qu'une lentille optique, de retarder les ondes ultrasonores émises afin de naturellement faire adopter au champ ultrasonore une focalisation (voir la figure 1.3).

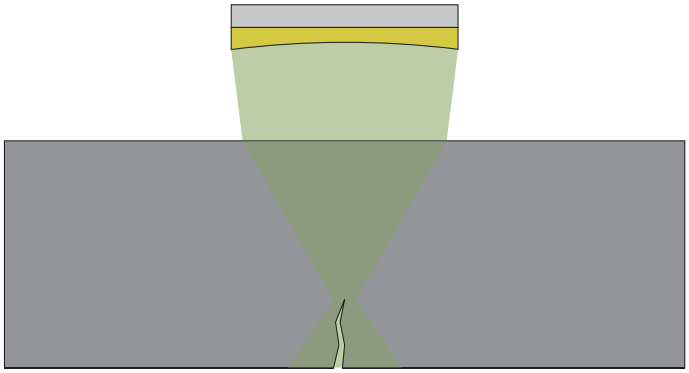


Figure 1.3 – Lentille pour traducteur ultrasonore

Ces méthodes sont cependant peu flexibles et requièrent une conception spécifique d'une lentille pour une focalisation donnée ou la préparation du capteur avant utilisation. Une solution plus souple consiste à utiliser des capteurs dotés de plusieurs éléments contrôlés électroniquement. En appliquant des retards précalculés au signal émis par chaque élément du capteur, il est possible de reproduire un effet de lentille et donc de focaliser le champ ultrasonore émis, comme le montre le schéma en figure 1.4.

Cette méthode, très flexible, permet d'utiliser plusieurs focalisations au cours d'un même contrôle et de balayer une zone spatiale sans déplacer ni changer l'orientation du capteur. Elle nécessite néanmoins une électronique de contrôle performante, capable de délivrer de manière simultanée et synchrone des signaux différents sur chaque élément avec des décalages de temps et des amplitudes variables.

1.5 Simulation de contrôle non destructif

1.5.1 De la nécessité de simuler

Le contrôle non destructif s'avère indispensable dans de nombreux domaines de l'industrie. Pour autant, bien que de nombreuses méthodes différentes existent et permettent des contrôles variés et adaptables aux diverses situations rencontrées, les contrôles présentent un coût en temps non négligeable.

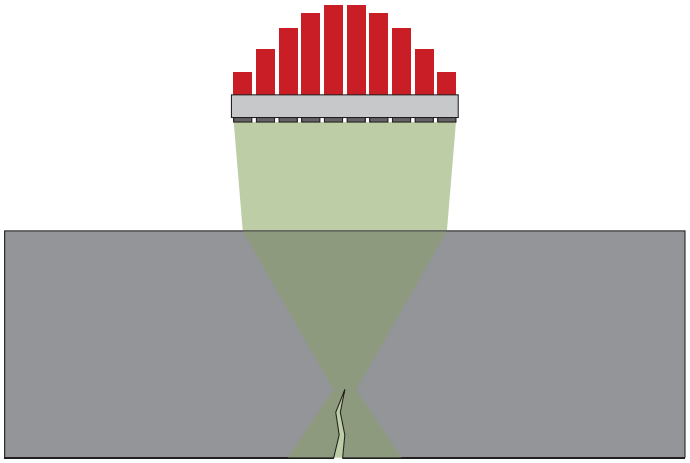


Figure 1.4 – Focalisation à l'aide d'un traducteur multi-éléments

Ainsi, réaliser des contrôles sur l'intégralité d'une structure n'est techniquement pas envisageable. Des études préalables des structures à contrôler, associées à une connaissance des contraintes usuelles appliquées sur celles-ci ainsi qu'à une expérience pratique du contrôle non destructif, peuvent permettre de réduire le nombre de contrôles nécessaires à garantir la fiabilité d'un système.

En ultrasons, la possibilité de prévoir le trajet de l'onde émise par le traducteur et d'estimer les zones fortement insonifiées au cours du contrôle permet de déduire la zone d'efficacité d'un contrôle. Ceci permet à un opérateur de concevoir des protocoles de contrôle destinés à la détection de défauts prévisibles (typiquement dans les zones les plus fragiles ou subissant les contraintes les plus fortes).

On peut donc, à l'aide d'outils de simulation, réduire drastiquement le nombre de contrôles nécessaires et augmenter leur efficacité à moindre coût. De tels outils offrent également la possibilité à l'opérateur d'interagir avec un système de contrôle non destructif, étape essentielle à une préhension intuitive des phénomènes physiques régissant la propagation des ondes dans les milieux contrôlés. La simulation de contrôle non destructif est donc essentielle au bon déroulement, à l'efficacité et à la rapidité d'exécution des inspections.

1.5.2 Différents types de simulation pour différents objectifs

Les outils logiciels offrant des fonctionnalités de simulation et d'analyse de contrôle non destructif sont nombreux et conçus pour répondre à des besoins différents. En simulation d'ultrasons, on peut considérer trois catégories principales d'outils de simulation de contrôle non destructif :

— **Les outils numériques**, généralement basés sur des méthodes par éléments finis, simulant à l'échelle microscopique l'ensemble des phénomènes physiques à l'origine des ondes ultrasonores. Ces outils sont très précis et permettent de reproduire les résultats obtenus par acquisition numérique. Cependant, ils nécessitent des temps de calcul très importants et produisent des résultats difficiles à interpréter.

— **Les outils qualitatifs** dont l'objectif n'est pas de simuler précisément les ondes ultrasonores, mais de fournir une visualisation des zones insonifiées par le faisceau d'ondes émis par le traducteur ultrasonore. Ils produisent des résultats peu précis, négligent certains phénomènes et servent surtout à esti-

mer rapidement un bon positionnement du traducteur pour une situation de contrôle donnée.

— Enfin, une dernière catégorie d'outils se place entre les deux précédentes. Souvent basés sur des méthodes de simulation semi-analytiques, ces outils permettent d'effectuer une simulation quantitative de la propagation des ondes ultrasonores en appliquant des **modèles** plus rapides que les méthodes à éléments finis, mais à domaine de validité restreint. Ces outils, utilisés à bon escient, rendent possible un usage plus systématique des simulations tout en produisant des résultats très proches de ceux obtenus lors des acquisitions.

La plate-forme logicielle *CIVA*, développée au Département d'Imagerie pour la Simulation et le Contrôle (DISC) du Commissariat à l'Énergie Atomique et aux Énergies Alternatives (CEA), propose un ensemble d'outils de simulation et d'analyse de contrôle non destructif. Dans le cas du contrôle par ultrasons, une méthode de simulation semi-analytique a été adoptée et validée expérimentalement.

Au travers d'une interface visuelle reprenant l'ensemble des éléments d'une inspection, l'utilisateur peut saisir les données d'un contrôle et lancer des simulations afin d'obtenir :

— Des cartographies d'amplitude au sein de la pièce inspectée, localisant les zones d'efficacité du contrôle.

— Les signaux des échos des ondes émises par le traducteur qui, comparés aux valeurs obtenues lors d'une acquisition, permettent de détecter la présence d'éventuels défauts.

L'objectif des travaux entrepris au cours de cette thèse est d'améliorer les outils existants de simulation de champ ultrasonore existant actuellement dans *CIVA* pour permettre une visualisation interactive de cartographies d'amplitude. En augmentant la fluidité de l'interaction d'un utilisateur avec le logiciel, les possibilités de conception de contrôle sont augmentées et, de manière générale, les temps nécessaires à l'obtention de résultats de simulation sont diminués.

Nous cherchons donc à adapter les modèles d'ondes ultrasonores existants au contexte du calcul à hautes performances afin d'exploiter au mieux les performances des processeurs parallèles courants. Notre objectif d'interactivité concernant la fluidité de l'interaction entre l'outil de simulation et l'utilisateur, une image de champ doit pouvoir être calculée en un temps de l'ordre d'un dixième de seconde. Il s'agit en effet d'un temps nominal mis par le cerveau humain pour traiter une information. Au-delà de cette cadence, seul le confort visuel est amélioré.

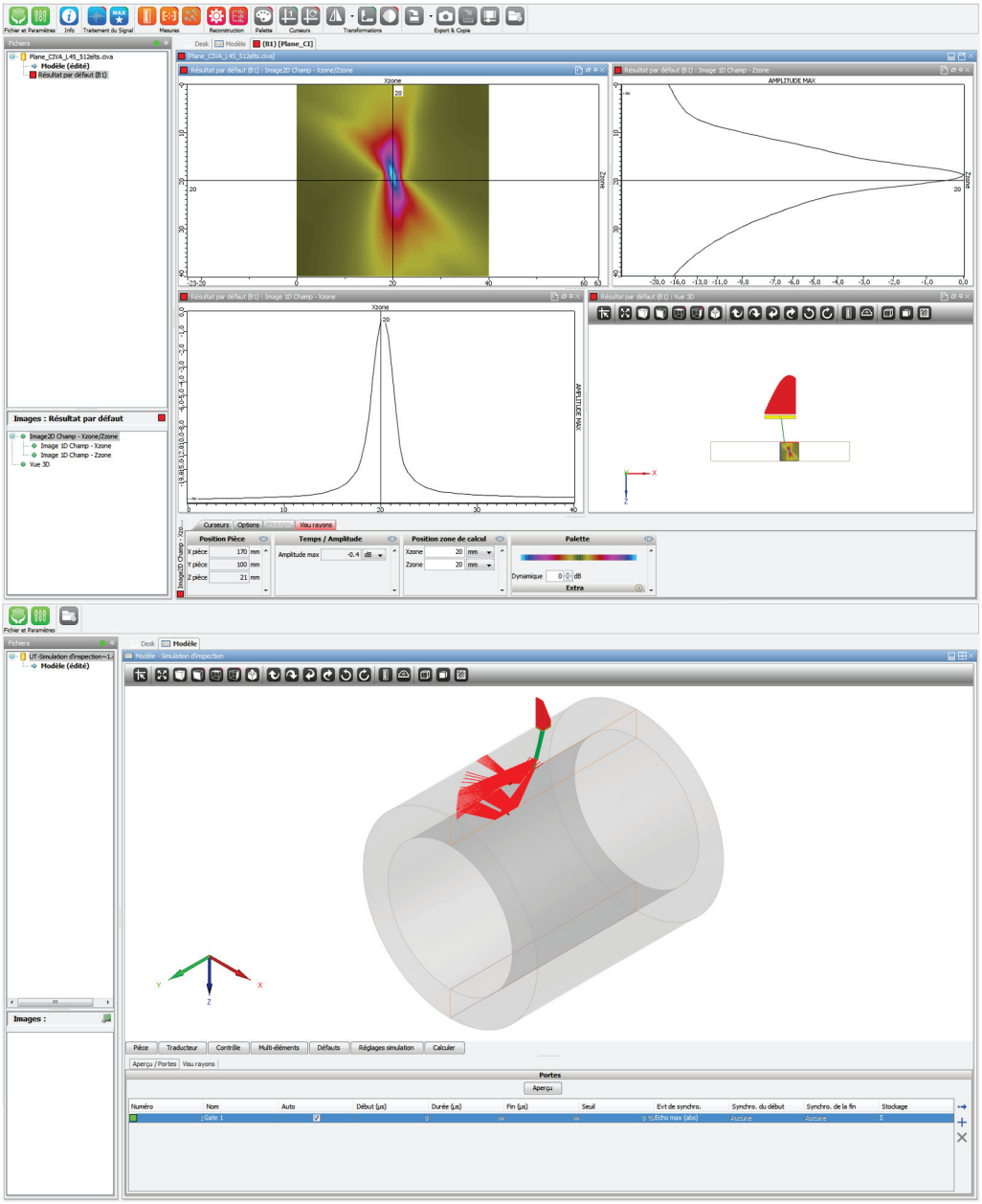


Figure 1.5 – Interface du logiciel CIVA

Deuxième partie

Simulation de champ ultrasonore

CHAPITRE 2

Modélisation des ondes ultrasonores

Sommaire

Le contrôle non destructif par ultrasons est entièrement basé sur l'émission et la réception d'ondes ultrasonores. Ainsi, pour pouvoir le simuler efficacement, nous proposons dans ce chapitre une étude des phénomènes physiques associés aux ultrasons.

Les simulations que nous entreprenons dans ces travaux requièrent un modèle d'onde ultrasonore. Après avoir défini le formalisme théorique dans lequel nous nous plaçons pour nos simulations, nous effectuons quelques développements théoriques nécessaires à la simulation de propagation d'ondes ultrasonores, tant en milieux isotropes qu'anisotropes, qu'ils soient homogènes ou hétérogènes.

2.1 Onde ultrasonore

2.1.1 Perturbation mécanique

Une onde ultrasonore est avant tout une onde sonore et donc une **onde mécanique** tridimensionnelle. Il s'agit d'une perturbation locale de la matière, oscillant autour d'une position de repos. Ce phénomène ne peut avoir lieu dans le vide. Il nécessite donc un milieu de propagation qui peut être solide ou liquide. La nature de ce matériau a un impact sur le comportement de la perturbation, comme nous le verrons par la suite. Cette perturbation influence, selon le milieu :

— La structure du matériau à l'échelle microscopique dans un milieu solide,

— La pression dans un milieu fluide (liquide ou gazeux).

Dans les deux cas, il s'agit d'un **déplacement** local de matière quantifiable que nous chercherons à caractériser par la suite.

2.1.2 Champ de déplacement

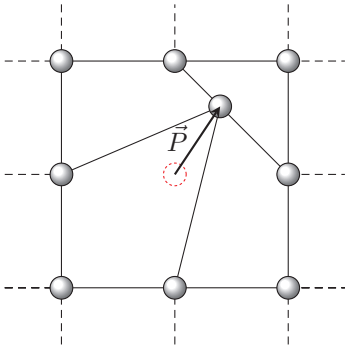


Figure 2.1 – Exemple de perturbation

La figure 2.1 illustre un exemple possible de perturbation mécanique dans un milieu solide. Celle-ci est caractérisée par le vecteur déplacement $\vec{P}(t)$ entre la position de repos et la position perturbée. Si l'on considère un milieu constitué de N particules $(\alpha_i)_{1\leq i\leq N}$, on peut définir un déplacement dépendant du temps $P_i(t)$ pour chaque particule α_i.

Par ailleurs, puisque l'on se place à l'échelle macroscopique, on peut assimiler le matériau à un milieu continu, chaque particule pouvant alors être identifiée par une coordonnée spatiale $\vec{u} = (x, y, z)$ plutôt que de manière discrète.

On considère alors la déformation $\vec{P}$ comme un **champ vectoriel** que l'on note $\vec{P}(\vec{u}, t)$. C'est ce champ que nous cherchons à déterminer lorsque nous simulons la

propagation des ondes ultrasonores.

2.1.3 Polarisation

Une onde ultrasonore peut donc être caractérisée par la perturbation $\vec{P}(\vec{u}, t)$ qu'elle induit sur le champ de déplacement local dans un matériau donné. Pour une onde ultrasonore, cette perturbation suit une direction caractéristique de l'onde, que l'on nomme la **polarisation**.

Ainsi, en notant $\vec{p_0}$ la polarisation de la perturbation $\vec{P}(\vec{u}, t)$ associée à une onde ultrasonore, on a :

$$\vec{P}(\vec{u}, t) = A(\vec{u}, t)\vec{p_0}$$

avec $A(\vec{u}, t)$ la norme du déplacement induit par la perturbation à l'instant t et à la position $\vec{u}$ dans le matériau.

2.1.4 Propagation

Lorsque cette perturbation locale se déplace dans le milieu de propagation, on parle d'onde mécanique **progressive**. C'est le cas pour une onde ultrasonore. On appelle **source** l'origine de la perturbation, la zone spatiale où une contrainte a été imposée au milieu.

Alors que la déformation de matière reste cantonnée à l'échelle microscopique dans la majorité des cas considérés en simulation ultrasonore (l'éloignement par rapport à la position de repos restant faible par rapport aux dimensions du matériau), un transfert d'énergie se produit à l'échelle macroscopique.

En effet, une perturbation implique une énergie et si cette perturbation se déplace, l'énergie fait de même. Une onde ultrasonore induit donc un **transfert d'énergie**, mais n'implique **aucun transfert de matière**, puisque les seuls déplacements ayant lieu n'opèrent qu'à l'échelle microscopique autour d'une position de repos. On considère par ailleurs que les déplacements étudiés ont des amplitudes assez faibles pour ne pas avoir d'effet sur les alentours de la particule considérée.

2.1.5 Onde progressive périodique

Une caractéristique supplémentaire des ultrasons réside dans l'aspect périodique de la perturbation du champ de déplacement. On définit donc aussi une onde ultrasonore par sa périodicité temporelle, ou période T. Par ailleurs, la vitesse de propagation d'une onde ultrasonore donnée dans un matériau homogène et isotrope est constante, on la note c.

Usuellement, le phénomène de dispersion est négligeable dans le cadre de la modélisation de contrôle non destructif par ultrasons, on peut donc considérer, pour un matériau donné, une célérité caractéristique qui ne dépend que de la polarisation de l'onde.

La périodicité temporelle et la vitesse de propagation constante impliquent une périodicité spatiale, que l'on caractérise à l'aide de la relation suivante :

$$c = \lambda f$$

On note λ la période spatiale de l'onde, ou longueur d'onde. La figure 2.2 schématise les concepts de périodicité temporelle et spatiale dans le cas d'une sinusoïde.

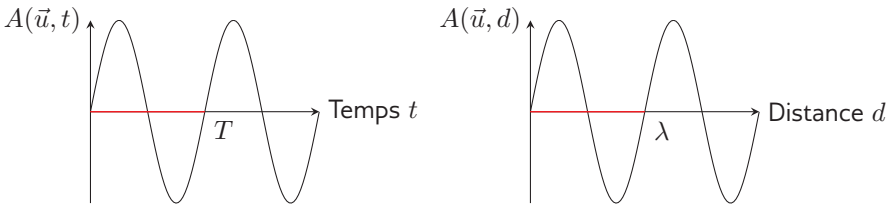


Figure 2.2 – Longueur d'onde et période d'une onde

La fréquence $f = \frac{1}{T}$ d'une onde ultrasonore est supérieure à 20 kHz et les fréquences typiques utilisées en contrôle non destructif sont de l'ordre du mégahertz. Lors de l'émission d'une onde ultrasonore de fréquence f imposée par l'émetteur ultrasonore, la célérité et donc la longueur d'onde dépendent des caractéristiques propres au matériau et de la polarisation de l'onde. Par exemple, une onde ultrasonore de fréquence 2 MHz se propageant dans l'eau a une célérité de 1 480 $m.s^{-1}$ et une longueur d'onde de $7.4.10^{-4}$ m.

2.1.6 Front d'onde

Pour une onde progressive périodique, le front d'onde correspond physiquement à la forme que prend la perturbation lorsqu'elle se propage dans l'espace. Pour chaque point M d'un front d'onde, la perturbation mécanique constituant l'onde met le même temps à se propager depuis son origine jusqu'à M. Le front d'onde est donc la **surface isochrone** de la perturbation. Nous définissons ici les types de fronts d'onde que nous utilisons pour simuler la propagation d'ondes ultrasonores.

Onde sphérique

Dans le cas théorique de l'excitation d'un milieu isotrope et homogène par une source de perturbations mécaniques ponctuelle, on peut modéliser le front de l'onde émise par une sphère. En effet, dans un tel milieu, une perturbation s'étend de manière identique dans toutes les directions autour du point d'origine.

Figure 2.3 – Ondes à la surface de l'eau

En ramenant le problème au cas plan, cela revient à un front d'onde circulaire tel qu'il peut par exemple être observé à la surface de l'eau (voir la figure 2.3).

Pour une onde sphérique d'origine O de longueur d'onde λ avec un signal d'excitation sinusoïdal de fréquence f, le champ de déplacement induit en un point M tel que $\overrightarrow{OM} = \vec{r}$ s'écrit :

$$\vec{P}(\vec{r},t) = \frac{A_0}{r} e^{j(kr-\omega t)} \overrightarrow{p_0}$$

avec :
— A_0 l'amplitude initiale de la perturbation,
— $k = \dfrac{2\pi}{\lambda}$ le nombre d'onde,
— $\omega = 2\pi f$ la pulsation,
— $\vec{p}_0$ la polarisation.

Onde plane

Le modèle d'onde plane assimile le front d'onde à un plan infini. Il correspond au cas théorique d'un plan vibrant infini émettant une perturbation mécanique. Bien qu'il ne corresponde à aucune situation réelle, il peut servir d'approximation locale à un front d'onde sphérique pour des angles solides réduits, comme présenté en figure 2.4, ou à modéliser le front d'onde émis par une surface plane de dimensions grandes devant celles considérées pour la modélisation.

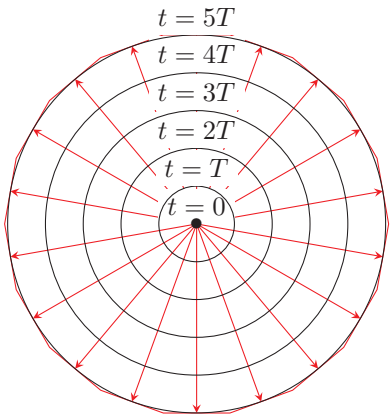


Figure 2.4 – Front d'onde sphérique approximé par des fronts d'onde plans

Pour une onde plane de longueur d'onde λ avec un signal d'excitation sinusoïdal de fréquence f, le champ de déplacement induit en un point M s'écrit :

$$\vec{P}(M,t) = A_0 e^{j(kr-\omega t)} \overrightarrow{p_0}$$

avec :
— r la distance algébrique de M au plan d'émission,
— A_0 l'amplitude initiale de la perturbation,
— $k = \dfrac{2\pi}{\lambda}$ le nombre d'onde,
— $\omega = 2\pi f$ la pulsation,
— $\vec{p}_0$ la polarisation.

Direction de propagation

On définit la direction de propagation d'une onde ultrasonore comme la normale au front d'onde en une position donnée. Ainsi, pour une onde sphérique d'origine O, en tout point M de l'espace, la direction de propagation de l'onde est donnée par le vecteur unitaire de direction $\overrightarrow{OM}$.

Cas général

Dans le cas d'un front d'onde de forme quelconque, le champ de déplacement induit en un point M par une onde ultrasonore émise depuis un point O, de signal d'excitation sinusoïdal de fréquence f et de longeur d'onde λ, s'écrit :

$$\vec{P}(M,t) = A_0 e^{j(\vec{k}\cdot\vec{r}-\omega t)}\vec{p_0}$$

avec :
— $\vec{r} = \overrightarrow{OM}$
— A_0 l'amplitude initiale de la perturbation,
— $\vec{k} = \frac{2\pi}{\lambda}\vec{d}$ le vecteur d'onde dirigé par $\vec{d}$ la direction de propagation locale, ou normale au front d'onde,
— $\omega = 2\pi f$ la pulsation,
— $\vec{p}_0$ la polarisation.

2.1.7 Types de polarisation

Comme mentionné en section 2.1.3, la polarisation est la direction selon laquelle la perturbation a lieu. En pratique, on considère deux types d'ondes principaux en fonction de leur polarisation :

Ondes longitudinales (ondes L) : parfois appelées « ondes de compression », elles se caractérisent par une polarisation colinéaire à la direction de propagation. Physiquement, elles sont équivalentes à une série de compressions et de dilatations du matériau dans la direction de propagation, comme présenté en figure 2.5.

Ondes transversales (ondes T) : également appelées « ondes de cisaillement », leur polarisation est orthogonale à la direction de propagation. Un exemple de cisaillement en deux dimensions est représenté en figure 2.6.

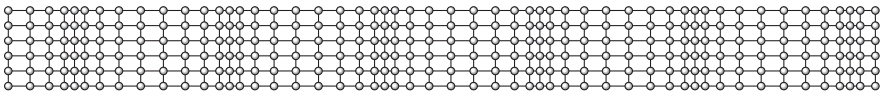


Figure 2.5 – Schématisation des déformations de la structure cristalline d'un solide au passage d'une onde de compression

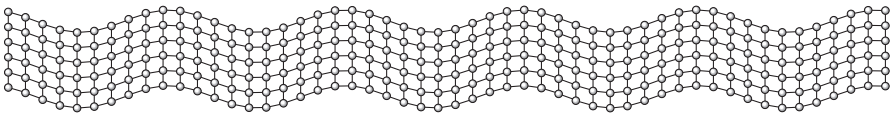


Figure 2.6 – Schématisation des déformations de la structure cristalline d'un solide au passage d'une onde de cisaillement

Le comportement d'une onde tant en propagation qu'en interaction avec des interfaces entre différents matériaux dépend de la polarisation. Ainsi, dans un même matériau solide, qu'il soit ou non dispersif, les ondes de cisaillement et de compression se propagent à des vitesses différentes. Pour cette raison, il est nécessaire de distinguer les différents modes de polarisation lors de la simulation d'ondes ultrasonores.

Par ailleurs, les ondes de cisaillement impliquent un mouvement de traction qui n'est possible que dans un matériau dont les constituants à l'échelle microscopique sont liés entre eux. Pour cette raison, les ondes transversales ne peuvent se propager qu'en milieux solides. À l'inverse, les ondes longitudinales ne mettant en jeu que des effets de compression, elles peuvent se propager dans les matériaux solides et fluides, qu'ils soient liquides ou gazeux.

Des exemples de vitesses de propagation pour différents modes dans différents matériaux sont présentés en table 2.1.

Matériau	Vitesse des ondes L	Vitesse des ondes T
Air	340 $m.s^{-1}$	N/A
Eau	1 680 $m.s^{-1}$	N/A
Plexiglas	2 680 $m.s^{-1}$	1 320 $m.s^{-1}$
Cuivre	4 900 $m.s^{-1}$	2 260 $m.s^{-1}$
Acier	5 900 $m.s^{-1}$	3 230 $m.s^{-1}$
Aluminium	6 300 $m.s^{-1}$	3 080 $m.s^{-1}$

Table 2.1 – Vitesse de propagation des ondes ultrasonores L et T dans différents matériaux

2.2 Propagation des ondes ultrasonores dans un milieu homogène

2.2.1 Conditions de propagation

Milieux de propagation

Comme mentionné précédemment, une onde ultrasonore ne peut exister qu'en présence d'un milieu de propagation. Celui-ci a un impact sur de nombreux paramètres de l'onde tels que sa vitesse de propagation, sa polarisation, son atténuation et, au niveau des interfaces entre deux matériaux, sur la direction des ondes générées et l'atténuation (coefficient de Fresnel, voir en annexe D).

Nous considérons dans la suite trois types de milieux de propagation différents :

Les milieux fluides : les liquides et les gaz. Ces milieux sont isotropes et ne peuvent porter que des ondes longitudinales.

Les milieux solides isotropes : milieux solides dans lesquels se propagent des ondes longitudinales et transversales à des vitesses différentes, mais indépendantes des directions de propagation.

Les milieux solides anisotropes : milieux solides dont la structure favorise la propagation des ondes selon des directions précises. On distingue dans le cas des matériaux anisotropes trois modes de polarisation :

— Polarisation qL : ondes quasi-longitudinales, la direction de polarisation est proche de la direction de propagation.

— Polarisations qT1 et qT2 : ondes quasi-transversales, la direction de polarisation est proche d'être contenue dans le plan orthogonal à la direction de propagation. Les polarisations des ondes qT1 et qT2 sont orthogonales.

En notant v_{qL}, v_{qT1} et v_{qT2} les vitesses respectives des ondes qL, $qT1$ et $qT2$ dans un matériau donné, on a systématiquement :

$$v_{qL} > v_{qT1} > v_{qT2}$$

Surfaces de lenteur

En notant $\vec{v}$ le vecteur vitesse, le **vecteur lenteur** désigne le vecteur ayant la même direction que $\vec{v}$ et pour norme l'inverse de la norme du vecteur vitesse :

$$\vec{s} = \frac{1}{\|\vec{v}\|^2}\vec{v}$$

La **lenteur** s correspond ainsi à l'inverse de la vitesse :

$$s = \frac{1}{v}$$

Pour visualiser la lenteur d'une onde pour un mode (longitudinal ou transversal) dans un matériau donné, nous utiliserons par la suite les surfaces de lenteur telles qu'introduites par Ogilvy (1985). Celles-ci correspondent à la surface décrite par l'ensemble des vecteurs lenteur pour toutes les directions de propagation depuis l'origine dans un matériau. Plus formellement, ces surfaces sont décrites par l'équation en coordonnées sphériques (r, θ, φ) :

$$r(\theta, \varphi) = \|\vec{s}(\theta, \varphi)\|$$

avec $\vec{s}(\theta, \varphi)$ le vecteur lenteur pour la direction de latitude θ et de longitude φ.

La figure 2.7 montre des exemples de surfaces de lenteur pour les différents modes de polarisation dans de l'acier austénitique.

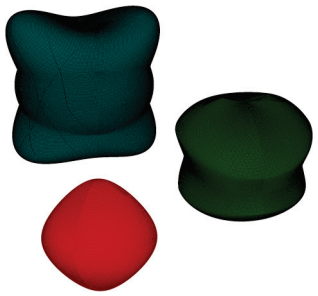


Figure 2.7 – Surfaces de lenteur dans l'acier austénitique pour les modes qL (en rouge), qT1 (en vert) et qT2 (en bleu)

Les surfaces de lenteur pour les milieux isotropes et fluides se résument à des sphères de rayons s, avec s la lenteur du mode de polarisation considéré.

Direction de phase, direction d'énergie

Dans le cas des milieux anisotropes, il convient de distinguer les notions de **direction de phase** et de **direction d'énergie**. La distinction à pratiquer est identique à celle faite, en ondes électromagnétiques, entre vitesse de phase et vitesse de groupe.

La direction de phase correspond à la direction d'évolution du front d'onde, de la même manière que la vitesse de phase est la vitesse de propagation de la phase dans l'espace pour les ondes électromagnétiques. Ainsi, la direction de phase est orthogonale à la surface de phases, le front d'onde.

La direction d'énergie correspond quant à elle à la direction dans laquelle se propage l'énergie d'un paquet d'onde émis depuis le point source. Cette énergie, caractérisée par le vecteur de Poynting, comme l'expliquent Royer et Dieulesaint (1996), se propage selon les directions de lenteur constante. Par conséquent, la direction d'énergie est orthogonale à la surface de lenteur.

Pour des milieux isotropes, les directions de phase et d'énergie sont confondues.

2.2.2 Propagation en milieux isotropes homogènes

Un milieu isotrope est caractérisé par le fait que sa réponse aux stimulations mécaniques ne dépend pas de son orientation. Ainsi, on peut, de manière générale, caractériser la propagation des ondes ultrasonores en milieux isotropes en ne tenant compte que du mode de polarisation de l'onde, duquel dépend sa vitesse de propagation. Les lenteurs des ondes L et T dans un matériau, ainsi que sa densité, notées respectivement s_L, s_T et ρ sont considérées comme caractéristiques de celui-ci dans tout ce qui suit.

Ondes longitudinales

Une perturbation longitudinale dans un matériau isotrope homogène émise dans une direction $\vec{d}$ se propage à la lenteur s_L suivant une trajectoire colinéaire à $\vec{d}$. L'onde induite est caractérisée par :

— Une polarisation $\vec{p_0} = \vec{d}$,
— Une direction de phase $\vec{d}$,
— Une direction d'énergie $\vec{d}_e = \vec{d}$,
— Une lenteur de propagation $s = s_L$.

Elle se propage de manière rectiligne depuis son origine jusqu'à arriver aux limites du milieu qu'elle traverse.

Ondes transversales

Une perturbation transversale dans un matériau solide isotrope homogène émise dans une direction $\vec{d}$ se propage à la lenteur s_T suivant une trajectoire colinéaire à $\vec{d}$. L'onde induite est caractérisée par :

— Une polarisation $\vec{p_0}$ orthogonale à $\vec{d}$,
— Une direction de phase $\vec{d}$,
— Une direction d'énergie $\vec{d}_e = \vec{d}$,
— Une lenteur de propagation $s = s_T$.

De la même façon qu'une onde longitudinale, elle se propage de manière rectiligne dans son milieu jusqu'aux limites du matériau.

2.2.3 Propagation en milieux anisotropes homogènes

Comme mentionné précédemment, la structure des matériaux anisotropes implique une dépendance de la vitesse de propagation à l'orientation du matériau. Afin de modéliser la propagation d'une onde ultrasonore dans un tel matériau, il est nécessaire de modéliser le comportement de celui-ci.

Sans effectuer l'ensemble des développements théoriques nécessaires à l'établissement d'un modèle de propagation pour les ondes ultrasonores en milieux anisotropes (tels qu'effectués par Gengembre (1999)), nous détaillons dans la suite quelques concepts nécessaires à nos objectifs de modélisation et de simulation.

Anisotropie

L'anisotropie mécanique s'explique par une différence de raideur en fonction de l'orientation considérée dans le matériau. Cette raideur est généralement due à la structure du matériau à l'échelle microscopique, qui peut, tout en étant homogène à l'échelle macroscopique, présenter une certaine hétérogénéité (de densité ou de composition) à l'échelle microscopique.

Par exemple, certains matériaux composites sont constitués d'une structure à base de fibres, le renfort, qui vient s'insérer dans une matrice, comme schématisé en figure 2.8.

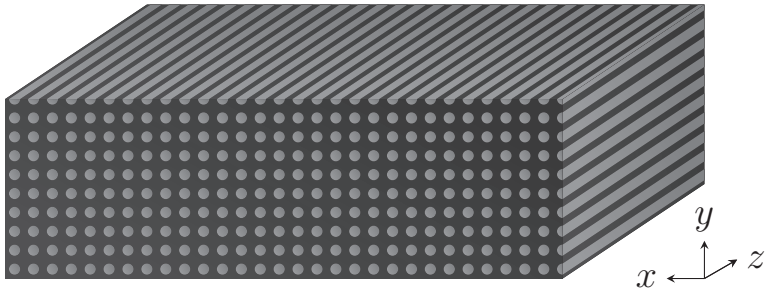


Figure 2.8 – Schématisation de la structure d'un exemple de matériau composite

Dans cette situation, la rigidité du matériau dans la direction z est supérieure à celle dans les directions x et y en raison de sa structure même. Il est nécessaire de modéliser cette différence pour pouvoir correctement simuler la propagation d'ondes ultrasonores dans des matériaux anisotropes.

Tenseur des contraintes

Nous cherchons dans un premier temps à caractériser, pour un élément de matériau infinitésimal cubique d'arête dx, les forces de tension qui l'incitent à rejoindre son état de repos. Dans le cas d'une force simple (traction, compression unidirectionnelle, torsion...), un scalaire donnant la valeur de la force suffit à décrire l'état de contrainte du système.

Cependant, dans le cas général, les trois composantes spatiales des contraintes appliquées sur chaque face du cube infinitésimal sont à différencier (voir figure 2.9). On appelle σ_{ij} la composante selon l'axe i de la contrainte appliquée sur la face j.

L'état de contrainte peut être caractérisé au moyen du **tenseur des contraintes**, modélisant des forces surfaciques et défini de la sorte :

$$\sigma(M) = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}$$

Les termes σ_{11}, σ_{22} et σ_{33} correspondent aux effets de compression et de traction. Les termes croisés σ_{12}, σ_{21}, σ_{23}, σ_{32}, σ_{33} et σ_{13} représentent les effets de cisaillement.

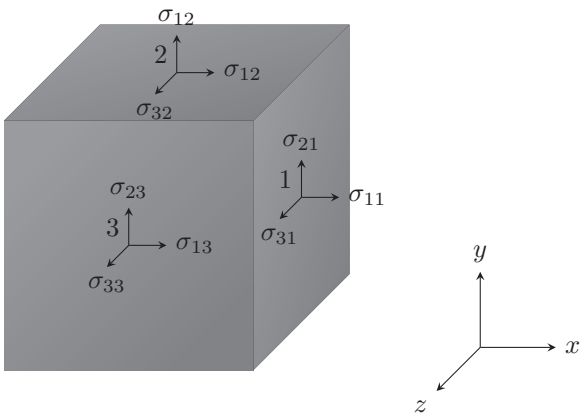


Figure 2.9 – Représentation des contraintes appliquées sur un cube élémentaire de matière

$$\sigma(M) = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{z3} \end{bmatrix}$$

À l'état statique, le tenseur des contraintes est symétrique. Il se réduit donc à six composantes indépendantes que l'on note, selon la notation de Voigt :

$$\begin{array}{cccccc} \sigma_{11} & \sigma_{22} & \sigma_{33} & \sigma_{23} & \sigma_{13} & \sigma_{12} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 & \sigma_5 & \sigma_6 \end{array}$$

On se ramène ainsi à un vecteur à six dimensions que nous utilisons dans la suite :

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix}$$

Tenseur des déformations

En parallèle avec la définition du tenseur des contraintes qui caractérise la contrainte appliquée à chaque face de l'élément infinitésimal, on définit le **tenseur des déformations** qui quantifie la déformation induite sur chaque face de l'élément infinitésimal.

En notant $\overrightarrow{D_1}$ la déformation appliquée à la face 1, $\overrightarrow{D_2}$ la déformation appliquée à la face 2 et $\overrightarrow{D_3}$ la déformation appliquée à la face 3 (voir figure 2.9), le tenseur des déformations s'écrit :

$$e = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{z3} \end{bmatrix} = \begin{bmatrix} \overrightarrow{D_1} & \overrightarrow{D_2} & \overrightarrow{D_3} \end{bmatrix}$$

Physiquement, le terme e_{ij} correspond à l'effet sur la face j d'une perturbation sur l'axe i. Ainsi, les éléments diagonaux représentent les effets de compression et de dilatation tandis que les éléments hors de la diagonale quantifient les effets de cisaillement.

De même que le tenseur des contraintes, le tenseur des déformations est symétrique et peut aussi s'écrire en utilisant la notation de Voigt :

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \end{bmatrix}$$

Remarque : Nous appliquons dans la suite la convention de sommation d'Einstein. La présence répétée d'un indice non défini dans un terme indique la sommation de ce terme sur toutes les valeurs que peut prendre l'indice.
Exemple :

$$y = a_i x_i = \sum_{i=1}^{3} a_i x_i$$

Loi de Hooke

En supposant les déplacements d'ampleur très inférieure aux dimensions du milieu de propagation (échelle microscopique), on peut admettre que les perturbations engendrées par une onde ultrasonore n'affectent que localement l'état du matériau et donc poser une hypothèse de **linéarité des déformations**.

Sous cette hypothèse, les déformations subies suivent la «loi des ressorts», ou **loi de Hooke**. Cette loi spécifie que la force F nécessaire pour appliquer un déplacement Δx à un point M dépend d'une constante appelée constante de raideur K et de Δx :

$$F = K\Delta x$$

On peut ainsi construire une relation linéaire entre le tenseur des déformations e et le tenseur des contraintes σ, valable dans tout le matériau et tenant compte de l'anisotropie. Celle-ci passe par la définition du **tenseur des constantes élastiques** c_{ijkl}, tenseur d'ordre 4 à 3 dimensions, tel que, en application de la loi de Hooke :

$$\sigma_{ij} = c_{ijkl} e_{kl} \tag{2.1}$$

Sans propriétés de symétrie, ce tenseur comporterait $3^4 = 81$ composantes indépendantes. Or, comme mentionné en section 2.2.3, le tenseur des contraintes est symétrique. Ainsi :

$$\forall (i,j) \in \{1,2,3\}, \sigma_{ij} = \sigma ji$$

Or, d'après la loi de Hooke $\sigma_{ij} = c_{ijkl} e_{kl}$. Donc, pour tout $(e_{kl})_{1 \leq k,l \leq 3}$:

$$\sum_{k=1}^{3}\sum_{l=1}^{3} c_{ijkl} e_{kl} = \sum_{k=1}^{3}\sum_{l=1}^{3} c_{jikl} e_{kl}$$

Ceci implique que $\forall(i,j,k,l) \in \{1,2,3\}, c_{ijkl} = c_{jikl}$, ce qui nous ramène à $6 * 3 * 3 = 54$ composantes indépendantes.

De même, la symétrie du tenseur des déformations $(e_{kl})_{1\leq k,l\leq 3}$ donne la propriété de symétrie suivante pour le tenseur des constantes élastiques :

$$\forall(i,j,k,l) \in \{1,2,3\}, c_{ijkl} = c_{ijlk}$$

Ceci ramène le nombre de composantes indépendantes à 36.

Enfin, l'hypothèse faite pour l'application de la loi de Hooke nous permet de supposer que, de la même manière qu'on associe une énergie potentielle à un ressort, le tenseur des contraintes peut être dérivé d'une énergie potentielle notée U. On a alors :

$$\forall(i,j) \in \{1,2,3\}, \sigma_{ij} = \frac{\partial U}{\partial e_{ij}}$$

Puis, d'après la loi de Hooke :

$$\forall(i,j,k,l) \in \{1,2,3\}, c_{ijkl} = \frac{\partial^2 U}{\partial e_{ij} \partial e_{kl}}$$

Or, puisque les dérivées partielles sont commutatives, on obtient :

$$\forall(i,j,k,l) \in \{1,2,3\}, c_{ijkl} = \frac{\partial^2 U}{\partial e_{ij} \partial e_{kl}} = \frac{\partial^2 U}{\partial e_{kl} \partial e_{ij}} = c_{klij}$$

Finalement, on en déduit la règle de symétrie globale suivante pour le tenseur des constantes élastiques :

$$\forall(i,j,k,l) \in \{1,2,3\}, c_{ijkl} = c_{klij} = c_{jikl} = c_{ijlk}$$

Les composantes indépendantes du tenseur des constantes élastiques sont donc au nombre de 21. Puis, suivant la notation de Voigt, on peut se ramener à la notation matricielle suivante :

$$c_{ijkl} \Rightarrow C_{\alpha\beta} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{12} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{13} & C_{23} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{14} & C_{24} & C_{34} & C_{44} & C_{45} & C_{46} \\ C_{15} & C_{25} & C_{35} & C_{45} & C_{55} & C_{56} \\ C_{16} & C_{26} & C_{36} & C_{46} & C_{56} & C_{66} \end{bmatrix}$$

De cette façon, en reprenant les notations de Voigt introduites précédemment, la loi de Hooke s'écrit :

$$\sigma_\alpha = C_{\alpha\beta} e_\beta$$

Équation de l'élastodynamique

L'équation de l'élastodynamique traduit, à l'instar de la relation fondamentale de la dynamique, la relation entre force et accélération. En notant $\vec{u}$ le déplacement, elle s'écrit comme suit :

$$\sum \vec{F} = \rho \vec{u} \tag{2.2}$$

$\sum \vec{F}$ étant la somme des forces appliquées par unité de volume et ρ étant la masse volumique de milieu, incluant notamment les forces de tension de la matière.

On peut montrer (voir Royer et Dieulesaint (1996)) que la force $\vec{T}$ par élément de surface résultant des tensions s'exprime à l'aide du tenseur des contraintes tel que suit, en application de la loi de Hooke :

$$T_i = c_{ijkl} \frac{\partial^2 u_l}{\partial x_j \partial x_k}$$

Ainsi, la relation fondamentale de la dynamique 2.2 devient, avec $\ddot{u}_i = \frac{\partial^2 u_i}{\partial^2 t}$:

$$c_{ijkl} \frac{\partial^2 u_l}{\partial x_j \partial x_k} + f_i = \rho \ddot{u}_i$$

f étant la somme des forces volumiques appliquées qui, en l'absence de sollicitation majeure, inclut principalement la force de gravitation, laquelle peut être négligée. On retient donc, pour la relation fondamentale de la dynamique, l'équation de propagation pour les petits déplacements :

$$c_{ijkl} \frac{\partial^2 u_l}{\partial x_j \partial x_k} = \rho \ddot{u}_i \tag{2.3}$$

La solution générale de cette équation, comme le montre Royer et Dieulesaint (1996) est une onde plane progressive dont l'amplitude s'écrit :

$$\vec{u}(\vec{x}, t) = \vec{U} exp\left(j\omega \left(t - \frac{\vec{n} \cdot \vec{x}}{V_\varphi} \right) \right) \tag{2.4}$$

$\vec{U}$ étant la polarisation de l'onde de vitesse V_φ et de direction de propagation $\vec{n}$.

Équation de Christoffel

En reportant la solution 2.4 à l'équation 2.3 dans cette même équation, on obtient :

$$\rho u_i = c_{ijkl} \frac{n_j n_k}{V_\varphi^2} u_l$$

Soit :

$$\rho V_\varphi^2 u_i = c_{ijkl} n_j n_k u_l \tag{2.5}$$

On appelle matrice de Christoffel la matrice Γ définie par :

$$\Gamma_{il} = c_{ijkl} n_j n_k$$

Notons que Γ est symétrique. Ses composantes sont les suivantes :

$$\begin{cases} \Gamma_{11} = C_{11}n_1^2 + C_{66}n_2^2 + C_{55}n_3^2 + 2(C_{16}n_1n_2 + C_{15}n_1n_3 + C_{56}n_2n_3) \\ \Gamma_{22} = C_{66}n_1^2 + C_{22}n_2^2 + C_{44}n_3^2 + 2(C_{26}n_1n_2 + C_{46}n_1n_3 + C_{24}n_2n_3) \\ \Gamma_{33} = C_{55}n_1^2 + C_{44}n_2^2 + C_{33}n_3^2 + 2(C_{45}n_1n_2 + C_{35}n_1n_3 + C_{34}n_2n_3) \\ \Gamma_{12} = C_{16}n_1^2 + C_{26}n_2^2 + C_{45}n_3^2 + (C_{12} + C_{66})n_1n_2 + (C_{14} + C_{56})n_1n_3 + \\ \quad (C_{46} + C_{25})n_2n_3 \\ \Gamma_{13} = C_{15}n_1^2 + C_{46}n_2^2 + C_{35}n_3^2 + (C_{14} + C_{56})n_1n_2 + (C_{13} + C_{55})n_1n_3 + \\ \quad (C_{36} + C_{45})n_2n_3 \\ \Gamma_{23} = C_{56}n_1^2 + C_{24}n_2^2 + C_{34}n_3^2 + (C_{46} + C_{25})n_1n_2 + (C_{36} + C_{45})n_1n_3 + \\ \quad (C_{23} + C_{44})n_2n_3 \\ \Gamma_{21} = \Gamma_{12} \\ \Gamma_{31} = \Gamma_{13} \\ \Gamma_{32} = \Gamma_{23} \end{cases}$$

Ceci nous amène à l'expression simplifiée suivante pour l'équation 2.5 :

$$\rho V_\varphi^2 u_i - \Gamma_{il} u_l = 0 \tag{2.6}$$

Ce qui revient à résoudre l'équation aux valeurs propres, en posant $\lambda = \rho V_\varphi^2$:

$$|\Gamma_{ik} - \lambda\delta_{ik}| = 0$$

$$\begin{vmatrix} \Gamma_{11} - \lambda & \Gamma_{12} & \Gamma_{13} \\ \Gamma_{21} & \Gamma_{22} - \lambda & \Gamma_{23} \\ \Gamma_{31} & \Gamma_{32} & \Gamma_{33} - \lambda \end{vmatrix} = 0$$

Après développement du déterminant et simplification, on obtient :

$$\lambda^3 - P\lambda^2 + Q\lambda - R = 0$$

$$\text{avec : } P = tr(\Gamma), R = |\Gamma| \text{ et } Q = \begin{vmatrix} \Gamma_{11} & \Gamma_{12} \\ \Gamma_{12} & \Gamma_{22} \end{vmatrix} + \begin{vmatrix} \Gamma_{22} & \Gamma_{23} \\ \Gamma_{23} & \Gamma_{33} \end{vmatrix} + \begin{vmatrix} \Gamma_{11} & \Gamma_{13} \\ \Gamma_{13} & \Gamma_{33} \end{vmatrix}$$

Dans le cas général, on trouve trois solutions λ_1, λ_2 et λ_3 positives qu'on prend dans l'ordre décroissant. On a ainsi trois vitesses de phase $V_\varphi^{(1)}$, $V_\varphi^{(2)}$ et $V_\varphi^{(3)}$:

$$\begin{cases} V_\varphi^{(1)} = \sqrt{\frac{\lambda_1}{\rho}} \\ V_\varphi^{(2)} = \sqrt{\frac{\lambda_2}{\rho}} \\ V_\varphi^{(3)} = \sqrt{\frac{\lambda_3}{\rho}} \end{cases}$$

Enfin, la résolution du système d'équations exprimé pour chacune des vitesses de phase permet d'obtenir la polarisation $\vec{u}$:

$$\begin{cases} \Gamma_{11}u_1 + \Gamma_{12}u_2 + \Gamma_{13}u_3 = \rho V_\varphi^{(1)} u_1 \\ \Gamma_{21}u_1 + \Gamma_{22}u_2 + \Gamma_{23}u_3 = \rho V_\varphi^{(2)} u_2 \\ \Gamma_{31}u_1 + \Gamma_{32}u_2 + \Gamma_{33}u_3 = \rho V_\varphi^{(3)} u_3 \end{cases} \tag{2.7}$$

En général, le classement par ordre décroissant des vitesses de phase suffit à différencier les modes qL, qT1 et qT2. Cependant, il peut arriver que les surfaces de lenteur se croisent et que la vitesse de phase du mode qT2 soit supérieure à celle du mode qT1, comme sur les surfaces de lenteur représentées en figure 2.10. Dans cette situation, on peut suivre les courbes de lenteur par continuité de la normale.

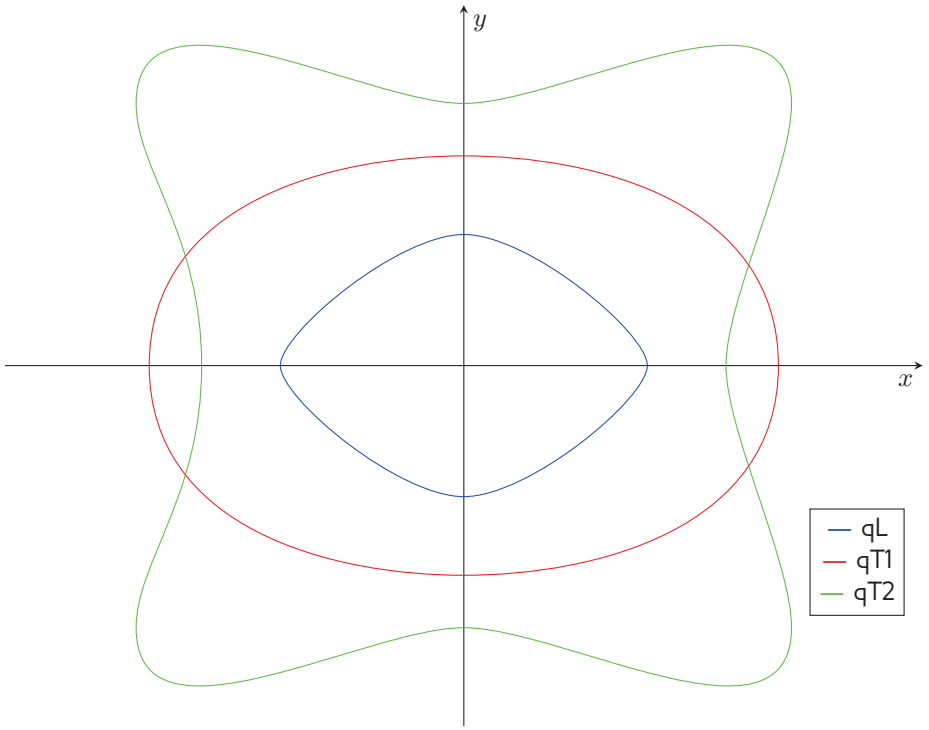


Figure 2.10 – Exemple de courbes de lenteur avec inversion de modes qT1 et qT2

On peut enfin montrer que la direction d'énergie se calcule de la sorte, pour chacune de ses composantes :

$$V_i^e = c_{ijkl} \frac{u_j u_l n_k}{\rho V_\varphi}$$

Remarque : Dans le cas le plus général d'anisotropie, ces modes ne sont ni complètement longitudinaux ni complètement transversaux, d'où le préfixe « quasi ».

2.3 Interaction d'ondes ultrasonores à une interface entre deux milieux

Le modèle exposé jusqu'ici suffit à déterminer les conditions de propagation d'une onde ultrasonore dans un milieu homogène isotrope ou anisotrope infini. Cependant, dans le cadre des simulations de contrôle non destructif, il est nécessaire de modéliser l'interaction d'une onde avec une interface entre deux milieux : les phénomènes de réflexion et de transmission.

2.3.1 Mise en situation

On considère une onde ultrasonore monochromatique de mode quelconque incidente sur une surface σ séparant le milieu d'incidence (1) d'un milieu (2), comme présenté en figure 2.11.

Nous cherchons à caractériser la transformation subie par les caractéristiques de l'onde (lenteur, direction de propagation et d'énergie, polarisation) pour une direction de propagation donnée telle que l'interaction avec l'interface entre (1) et (2) ait

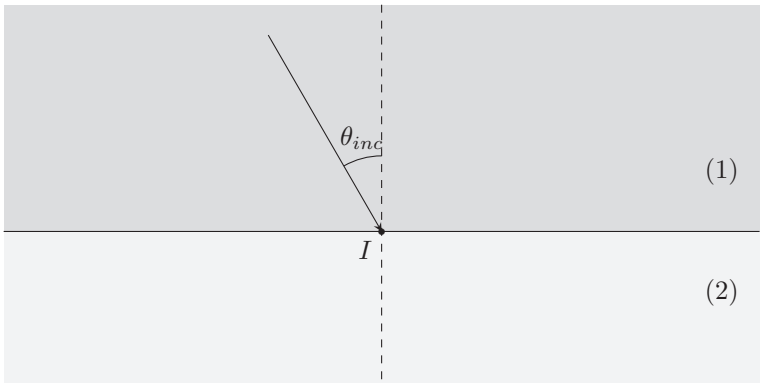


Figure 2.11 – Onde incidente sur une interface entre deux milieux

lieu au point I.

2.3.2 Contraintes de continuité

En supposant un contact parfait entre les deux milieux au niveau de l'interface, la continuité des deux grandeurs suivantes est assurée :

— Le déplacement $\vec{u}(\vec{x},t)$,

— La contrainte $\vec{T}(\vec{x},t)$.

Ces conditions de continuité ne peuvent être satisfaites que par l'apparition d'ondes transmises et réfléchies au niveau du point d'incidence. Dans le cas le plus général, des ondes de mode L et T en milieu isotrope et de mode qL, qT1 et qT2 en milieu anisotrope sont générées de chaque côté de l'interface, comme présenté en figure 2.12.

Leurs directions de propagation, lenteurs et directions d'énergie dépendent à la fois de la direction de propagation de l'onde incidente et des caractéristiques des milieux (1) et (2). Par ailleurs, par application du principe de conservation de l'énergie, l'amplitude de ces ondes est plus faible que celle de l'onde incidente. On introduit pour modéliser ce phénomène un coefficient d'amplitude, le **coefficient de Fresnel** (voir l'annexe D).

2.3.3 Loi de Snell-Descartes généralisée

D'après l'équation 2.4, chacune des ondes incidente, réfléchies et transmises peut s'écrire, avec $\vec{k}$ le vecteur d'onde, sous la forme :

$$\vec{u}(\vec{x},t) = \vec{U}e^{j(\omega t - \vec{k}\cdot\vec{x})}$$

La continuité des déplacements se traduit par l'égalité suivante :

$$\vec{u}_{inc} + \sum \vec{u}_{ref} = \sum \vec{u}_{tra} \tag{2.8}$$

Ceci étant vrai à tout instant t, les pulsations des ondes générées sont égales à celle de l'onde incidente. La fréquence est donc conservée à une interface entre différents matériaux, quelles que soient leurs caractéristiques.

Par ailleurs, l'égalité 2.8 est aussi vraie pour toute position $\vec{x}$ sur le plan d'incidence. Par conséquent, pour toute position $\vec{x}$ incluse dans le plan de l'interface entre les milieux (1) et (2) et dans le plan d'incidence :

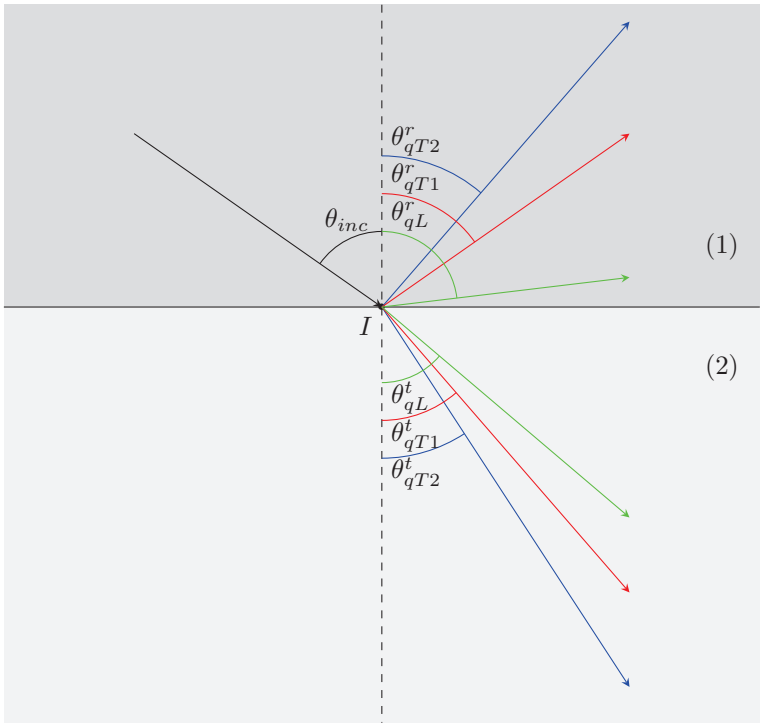


Figure 2.12 – Ondes incidente, réfléchies et transmises à une interface entre deux milieux anisotropes

$$\vec{k}_{inc} \cdot \vec{x} = \vec{k}_{ref} \cdot \vec{x} = \vec{k}_{tra} \cdot \vec{x}$$

Ainsi, pour deux positions $\vec{x}_0$ et $\vec{x}_1$ de l'interface :

$$\begin{cases} \vec{k}_{ref} \cdot \vec{x}_0 &= \vec{k}_{inc} \cdot \vec{x}_0 \\ \vec{k}_{ref} \cdot \vec{x}_1 &= \vec{k}_{inc} \cdot \vec{x}_1 \end{cases}$$

Donc :

$$\vec{k}_{ref}.(\vec{x}_0 - \vec{x}_1) = \vec{k}_{inc} \cdot (\vec{x}_0 - \vec{x}_1) \quad (2.9)$$

$$(\vec{k}_{ref} - \vec{k}_{inc}) \cdot (\vec{x}_0 - \vec{x}_1) = 0 \quad (2.10)$$

Le vecteur $\vec{k}_R - \vec{k}_I$ est donc orthogonal à l'interface. Ainsi, toute onde de vecteur d'onde $\vec{k}$ générée lors de l'interaction entre une onde incidente de vecteur d'onde $\vec{k}_I$ et une interface vérifie :

$$\vec{k}.\vec{e}_x = \vec{k}_I.\vec{e}_x$$

où $\vec{e}_x$ est un vecteur unitaire inclus dans le plan d'incidence et le plan de l'interface.

La pulsation étant conservée, comme expliqué précédemment, on en déduit la loi de Snell-Descartes, avec les notations utilisées en figure 2.12, pour tout mode m :

$$s^{inc} sin(\theta^{inc}) = s^{ref}_m sin(\theta^{ref}_m) = s^{tra}_m sin(\theta^{tra}_m) \quad (2.11)$$

s^{inc} étant la lenteur de l'onde incidente et s^{ref}_M et s^{tra}_M, respectivement, les lenteurs des ondes réfléchies et transmises en mode M.

2.3.4 Calcul des ondes générées à une interface

Entre deux milieux isotropes

À une interface entre deux milieux isotropes, une onde incidente est susceptible de générer quatre ondes, comme présenté en figure 2.13 :

— Une onde T réfléchie,
— Une onde L réfléchie,
— Une onde T transmise,
— Une onde L transmise.

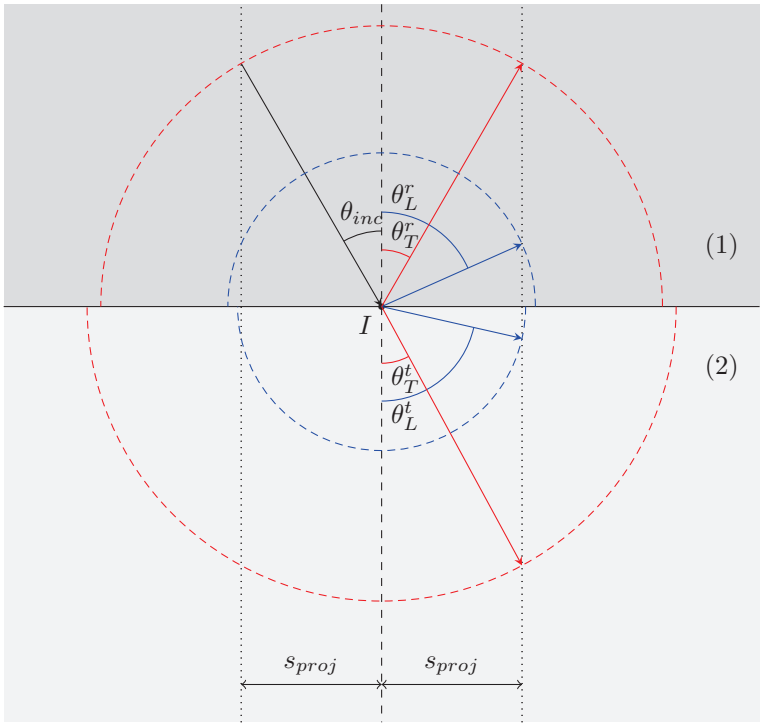


Figure 2.13 – Vecteurs lenteur des ondes incidente, réfléchies et transmises à une interface entre deux milieux

D'après la loi de Snell-Descartes, en notant $s_{T,1}$ et $s_{L,1}$ les lenteurs des modes L et T dans le milieu (1), les conditions de continuité aux interfaces entre milieux isotropes se traduisent par les égalités suivantes :

$$s_I sin(\theta^{inc}) = s_{T,1} sin(\theta_T^{ref}) = s_{T,2} sin(\theta_T^{tr}) = s_{L,1} sin(\theta_L^{ref}) = s_{L,2} sin(\theta_L^{tr})$$

Par conséquent, pour chaque mode m, on obtient le vecteur lenteur $\vec{s}_m$ associé à l'onde générée pour le mode m, dans le repère de l'interface tel que représenté en figure 2.13 :

$$\vec{s}_m = \langle \vec{s}_i \cdot \vec{e}_x \rangle \vec{e}_x + s_m \sqrt{1 - \frac{\langle \vec{s}_I \cdot \vec{e}_x \rangle^2}{s_m^2}} \vec{e}_y \tag{2.12}$$

Cette équation peut également être considérée géométriquement à l'aide des surfaces de lenteur (sphériques en milieu isotrope). En considérant l'intersection de la surface de lenteur des deux milieux avec le plan d'incidence, on obtient des courbes

de lenteur. Les intersections entre l'axe de lenteur projetée ($x = s_{proj}$) et les surfaces de lenteur correspondent aux modes générés lors de l'interaction avec l'interface et indiquent les directions de phase et d'énergie, comme schématisé en figure 2.13.

On constate que lorsque la lenteur dans le milieu (2) est inférieure à la lenteur dans le milieu (1) pour un mode m donné, il n'existe pas forcément de solution à l'équation de Snell-Descartes. L'angle θ_c maximal pour lequel une solution existe est appelé **angle critique** (voir figure 2.14). Au-delà de cet angle, le mode m est généré sous la forme d'une **onde évanescente** qui se propage uniquement dans la surface d'interface entre les deux milieux et est rapidement atténuée.

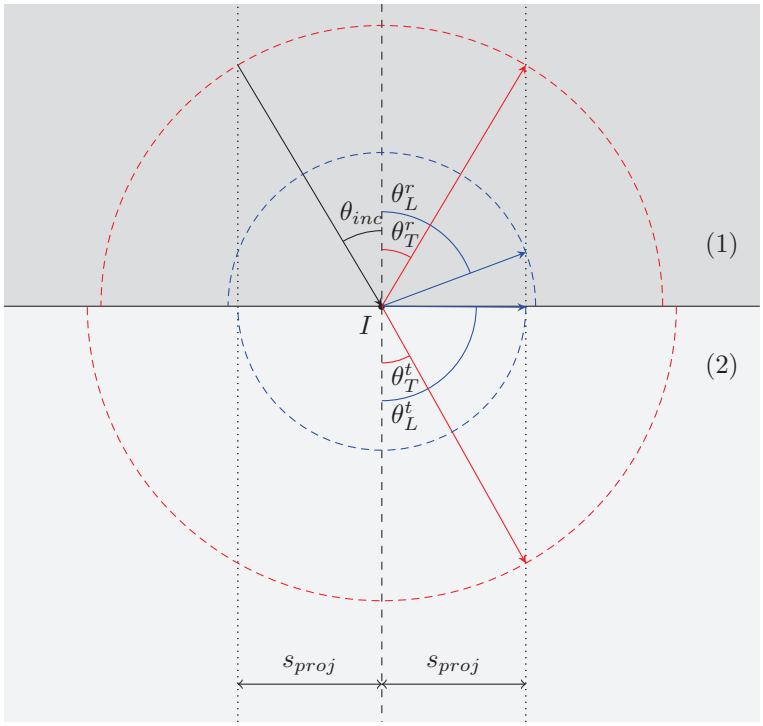


Figure 2.14 – Interaction à une interface entre deux matériaux isotropes pour un angle incident égal à l'angle critique en mode T dans le milieu (2)

Entre deux milieux anisotropes

Calcul des lenteurs La loi de Snell-Descartes s'applique aussi bien aux milieux isotropes qu'anisotropes et se traduit par la même condition. Notons $\vec{s}$ le vecteur lenteur généré à l'interaction d'une onde incidente I avec une interface entre deux milieux anisotropes. En reprenant l'équation de Christoffel 2.5 et en notant $G_{il} = c_{ijkl}s_j s_k$, on obtient :

$$(\rho\delta_{il} - G_{il})U_l = 0 \tag{2.13}$$

L'application de la loi de Snell-Descartes nous permet de décomposer $\vec{s}$ comme suit, avec $\vec{n}$ la normale à l'interface et $\vec{s}_{Ipar}$ la projection de la lenteur incidente sur l'interface :

$$\vec{s} = \vec{s}_{Ipar} + \sigma\vec{n}$$

Ceci est vrai pour tous les modes générés à l'interaction de l'onde incidente avec l'interface. Chacun de ces modes est donc caractérisé par sa composante σ. En injectant cette décomposition dans l'équation 2.13, on a :

$$det|\rho\delta_{il} - G_{il}| = det|\rho\delta_{il} - c_{ijkl}(s_{Ipar} + \sigma n_j)(s_{Ipar} + \sigma N_k)| = 0 \tag{2.14}$$

Il s'agit d'une équation polynomiale de degré 6 en σ qui doit être résolue pour chacun des milieux (1) et (2). Puisque les coefficients de cette équation sont réels, les solutions σ que l'on obtient sont soit réelles, soit complexes conjuguées. On construit alors les caractéristiques des ondes générées à l'interface telles que suivent :

— Lenteur $\vec{s} = \vec{s}_{Ipar} + \sigma\vec{n}$,

— Direction de propagation $\vec{d} = \frac{\vec{s}}{\|\vec{s}\|}$.

Les valeurs imaginaires de σ que l'on peut obtenir correspondent à des ondes évanescentes.

Il est aussi possible d'utiliser une méthode géométrique comme dans le cas des matériaux isotropes pour le calcul des lenteurs, à partir de descriptions des surfaces de lenteurs, comme présenté en figure 2.15.

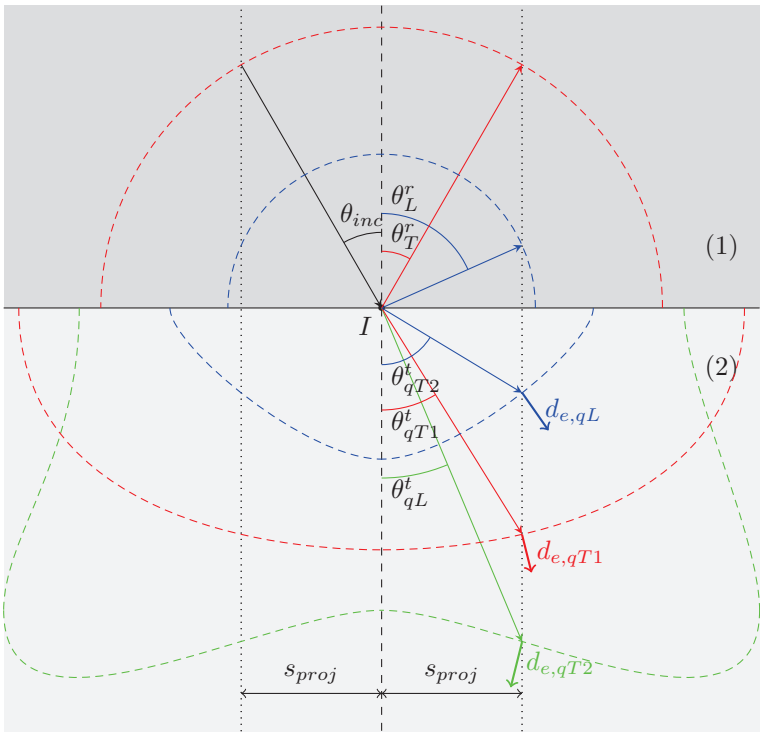


Figure 2.15 – Interaction à une interface entre un matériau isotrope (1) et un matériau anisotrope (2)

Calcul des polarisations et directions d'énergie Une fois les lenteurs et directions de propagation calculées, le calcul des polarisations et directions d'énergie est identique à celui énoncé en section 2.2.3. En effet, la direction de propagation suffit à caractériser l'onde dans un milieu homogène, qu'il soit isotrope ou anisotrope.

Remarquons enfin que le calcul des directions d'énergie peut également être effectué géométriquement en utilisant la normale à la surface de lenteur du mode

considéré au point d'intersection entre l'axe vertical de la lenteur projetée et la surface de lenteur, comme schématisé en figure 2.15.

2.4 Conclusion

Au cours de ce chapitre, nous avons présenté les phénomènes physiques causant les ondes ultrasonores et introduit des notions utiles à leur caractérisation. Puis, partant de l'étude à l'échelle microscopique de la structure de la matière, nous avons déduit un ensemble de lois permettant de quantifier les ondes ultrasonores en milieux homogènes.

L'extension de ces considérations nous a permis de prouver l'existence dans le cas général de trois solutions aux équations de propagation des ondes ultrasonores en milieux anisotropes. Nous avons caractérisé ces solutions, apportant ainsi une méthode générale de calcul de la propagation des ondes ultrasonores dans un milieu homogène isotrope ou anisotrope.

Puis, en considérant les conditions aux interfaces entre des matériaux de propagation, nous avons déduit l'équation de Snell-Descartes généralisée, qui régit le comportement des ondes ultrasonores aux limites d'un milieu. Nous avons déduit de ces équations la nécessité d'ondes générées à une interface, tant en réflexion qu'en réfraction.

Enfin, la résolution de l'équation de Snell-Descartes, combinée à l'équation de propagation, nous a permis de déduire le comportement exact des ondes ultrasonores à une interface ainsi qu'une méthode générale pour le quantifier.

CHAPITRE 3

Lancer de rayons ultrasonores

Sommaire

Maintenant que nous disposons d'éléments de modélisation physique suffisants pour calculer les valeurs des déplacements induits par une onde ultrasonore en milieux isotropes ou anisotropes, homogènes ou hétérogènes, nous allons décrire les outils mis en place afin de simuler la propagation d'une onde ultrasonore.

Puisque les déformations subies par la surface de phase d'une onde ultrasonore rendent sa modélisation volumique trop complexe pour être gérée analytiquement, il est nécessaire de l'échantillonner afin d'en simuler la propagation. Cet échantillonnage peut être réalisé au moyen de l'approximation des rayons, équivalente à l'approximation de l'optique géométrique pour les ondes électromagnétiques.

Ce chapitre a pour objectif de présenter les méthodes mises en œuvre et les principes utilisés pour obtenir un outil de lancer de rayons ultrasonores aussi rapide et précis que possible. Cet outil permet le suivi du trajet d'un rayon dans une pièce comportant des milieux homogènes séparés par des interfaces bien définies et composés de matériaux isotropes ou anisotropes. Des résultats de performance et de validité sont également détaillés.

3.1 Lancer de rayons

3.1.1 Méthodes

Principe de base

Le lancer de rayons est une méthode de rendu graphique initialement développée par Appel (1968) dont l'idée consiste à créer une visualisation d'une scène virtuelle à trois dimensions comportant des objets modélisés en simulant la propagation de **rayons** lumineux virtuels depuis un point d'observation jusqu'aux objets constituant la scène. En appliquant le principe de réciprocité des trajectoires, pour reconstituer une image, il suffit d'associer à chaque pixel de l'image un rayon dont la direction initiale relie le point d'observation au pixel. Le point d'arrivée du rayon détermine la couleur à donner au pixel ainsi qu'une évaluation de l'éclairage au moyen d'un modèle local, comme schématisé en figure 3.1.

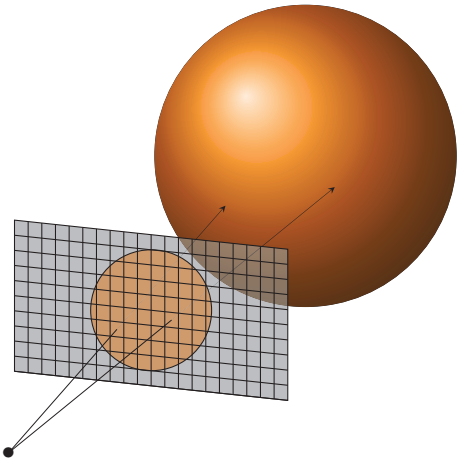


Figure 3.1 – Schématisation du principe du lancer de rayons

Si les trajets des rayons lancés suivent un modèle conforme à la réalité de la propagation lumineuse (modèle d'éclairage physiquement correct et suivi de rayons secondaires) cette méthode fournit des résultats de rendu réalistes particulièrement

proches de ce que l'on pourrait obtenir en photographie, comme le rendu en figure 3.2.

Figure 3.2 – Exemple de rendu photo-réaliste obtenu par lancer de rayons

L'opération basique du lancer de rayons consiste, pour un point d'origine O, une direction de départ $\vec{d}$ et une scène donnée, à réaliser l'opération d'intersection géométrique de la droite passant par O et dirigée par $\vec{d}$ avec l'ensemble des objets de la scène.

Complexité

L'efficacité de cette technique réside dans la complexité du modèle d'éclairage local adopté. En effet, la lumière est soumise à des phénomènes divers lors de sa propagation, qu'il convient de modéliser. Ainsi, il sera nécessaire, à chaque interaction avec un objet, de considérer la dispersion, la réflexion, la réfraction, la diffraction, la diffusion... Chacun de ces phénomènes engendre lui-même des rayons secondaires qui devront être modélisés de la même façon.

Ainsi, pour obtenir un rendu exhaustif et parfaitement exact, le nombre de rayons à tracer est théoriquement infini. Cependant, une telle condition d'exhaustivité et d'exactitude n'est pas nécessaire pour obtenir un rendu photo-réaliste : certains effets influent peu sur l'image finale et peuvent être négligés sans crainte d'altération du résultat final. De plus, l'atténuation de l'intensité lumineuse avec la distance de propagation limite la profondeur d'exploration de l'arbre des rayons utile au rendu. Malgré cela, pour obtenir une image de résolution raisonnable, il est nécessaire de tracer de nombreux rayons.

Par ailleurs, l'opération de lancer d'un rayon $(O, \vec{d})$ seul nécessite de déterminer précisément la première intersection rencontrée par la droite (D) qui le porte avec l'ensemble des primitives géométriques qui constituent la scène. En notant N le nombre total de primitives de la scène, une approche naïve du processus d'intersection consistant à effectuer un test d'intersection entre la droite et l'ensemble des N primitives résulterait en une complexité linéaire en nombre d'intersections :

$$C(N) = O(N)$$

Structures accélératrices

Il est possible de bénéficier des informations de positionnement des primitives de la scène de lancer de rayons pour réduire le coût d'une opération élémentaire d'intersection. Plusieurs solutions à ce problème existent. Nous présentons brièvement

deux méthodes basées sur des structures accélératrices arborescentes couramment utilisées pour accélérer les opérations de lancer de rayons.

Le *kd-tree* L'opération de base de la construction d'un *kd-tree* consiste à partager l'espace en deux parties séparées par un plan (P). Plusieurs méthodes existent pour effectuer ce partage, la plus répandue étant la méthode *Surface Area Heuristic* (SAH), telle que décrite dans Wald (2007). Pour construire un *kd-tree*, on applique récursivement cette opération aux ensembles obtenus après subdivision, de sorte à arriver à des ensembles contenant un nombre réduit d'éléments.

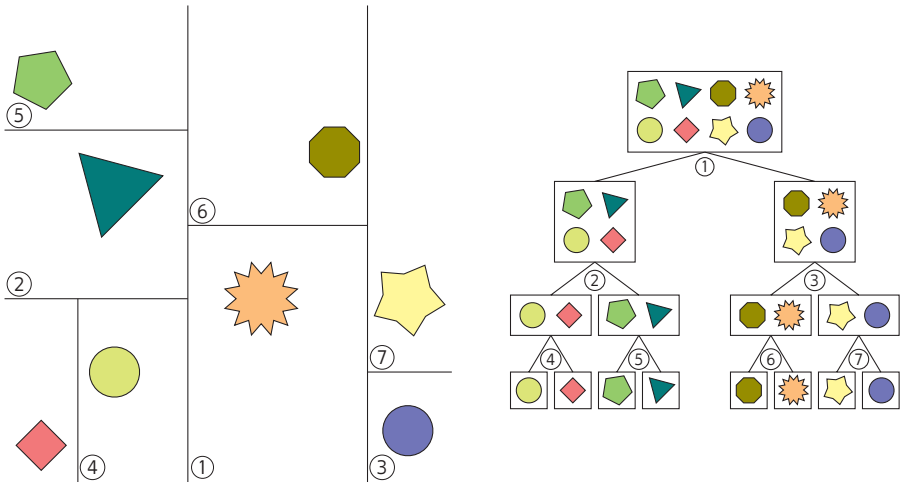


Figure 3.3 – Exemple de *kd-tree*

Ces plans de subdivision permettent de construire un arbre décrivant la structure spatiale de la scène. Un exemple de *kd-tree* est donné en figure 3.3.

À partir de cette structure accélératrice, la détermination de l'intersection d'un rayon donné avec les primitives de la scène peut être effectuée récursivement en un nombre d'étapes égal à la profondeur de l'arbre. À partir du nœud racine de l'arbre, on effectue un test pour savoir de quel côté du plan de subdivision le rayon se situe. En fonction du résultat, on ajoute à une pile de branches à explorer soit l'une, soit les deux branches du nœud, soit aucune. Une fois arrivé à une feuille de l'arbre, il suffit d'effectuer le test d'intersection avec la primitive contenue dans la feuille. En fonction du résultat, on termine le parcours de l'arbre ou on dépile une branche à explorer. Un ordre de priorité basé sur la distance des branches à l'origine peut être imposé lors de l'empilement afin de trouver l'intersection la plus proche de l'origine du rayon.

On a ainsi une complexité en intersections pour la traversée de l'arbre, dans le cas d'un *kd-tree* bien construit :

$$C(N) = O(log_2(N))$$

Les BVH Les BVH sont une série de boîtes englobantes contenues les unes dans les autres. Pour construire une structure accélératrice à base de BVH, on commence par construire la boîte englobante de l'ensemble des primitives de la scène. On sépare ensuite l'ensemble des primitives contenues dans cette boîte en deux groupes, que l'on englobe également. Le processus de subdivision récursif est répété jusqu'à ce que les ensembles soient indivisibles.

Plusieurs heuristiques de subdivision de boîtes englobantes existent, permettant de favoriser les performances d'exploration ou de construction, ou encore la taille de la structure. Pour le détail des caractéristiques de ces heuristiques et leur implémentation, nous nous référons à Pharr et Humphreys (2004).

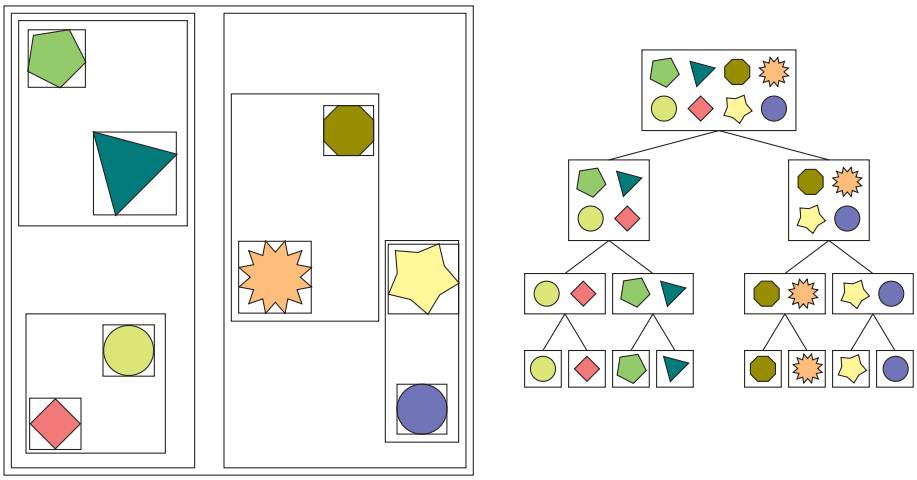


Figure 3.4 – Exemple de BVH

On obtient, de la même façon que pour les *kd-trees*, un arbre de subdivisions de l'espace tel que dans l'exemple présenté en figure 3.4.

L'exploration de cette structure implique, à chaque étape, un test d'intersection avec chaque boîte englobante incluse dans le nœud pour décider quelles branches doivent être explorées. De la même façon, il peut être nécessaire d'explorer une, plusieurs, ou aucune branche. Une fois que la traversée de l'arbre arrive à un nœud, il suffit d'intersecter la primitive que le nœud contient.

Ainsi, on retrouve une complexité en intersections pour la traversée de la BVH, dans le cas d'une BVH bien construite :

$$C(N) = O(log_2(N))$$

Comparaison des deux méthodes Les structures accélératrices de type *kd-tree* fournissent un meilleur partitionnement de l'espace limitant la profondeur d'exploration lors du lancer de rayon. L'utilisation de la méthode SAH permet notamment de gérer efficacement les différences de niveaux de détail dans des scènes complexes en minimisant la profondeur d'exploration pour les zones peu complexes.

Cependant, ce type de structure n'est adapté que pour des scènes statiques : si la scène est modifiée, il est nécessaire de reconstruire l'ensemble de la structure accélératrice. Or, la construction d'un *kd-tree* est coûteuse, sa complexité est en $O(Nlog(N))$ avec N le nombre de primitives de la scène.

Les structures de type BVH sont nettement plus souples pour les scènes dynamiques et leur construction est plus rapide. Si les modifications appliquées à la scène sont locales, il n'est pas nécessaire de reconstruire la BVH. Cependant, la complexité d'exploration de ces structures les rend moins efficaces que les *kd-tree* pour des scènes statiques pour une exploration non vectorisée.

De nombreux travaux ont été menés pour l'amélioration des heuristiques de construction de BVH, améliorant le partitionnement de l'espace et accélérant encore leur temps de construction. De plus, l'utilisation de BVH à arité constante permet

la vectorisation de l'exploration de la structure. Ces améliorations font qu'actuellement, les BVH sont à privilégier dans la grande majorité des situations pour accélérer le lancer des rayons, y compris sur des scènes statiques.

3.1.2 Outils de lancer de rayons

Bien que l'utilisation de structures accélératrices permette une nette amélioration des performances du lancer de rayons, particulièrement sur les scènes comportant de nombreuses primitives, cela reste un procédé particulièrement coûteux. Afin de maximiser les performances de lancer de rayons dans le cadre du calcul de champ ultrasonore, nous nous sommes intéressés à deux bibliothèques principales de lancer de rayons, principalement utilisées dans le domaine du rendu photo-réaliste.

Intel *Embree*

La bibliothèque *Embree* (voir Woop et collab. (2013) et Intel Corporation (2011)) est développée par Intel à destination des CPU et des coprocesseurs de calcul *Many Integrated Core* (MIC) [1] et diffusée sous la licence Apache 2.0 (voir Apache Software Foundation (2004)). Elle a été optimisée pour ses architectures cibles et exploite ainsi les instructions SIMD des CPU modernes issues des jeux d'instructions SSE, AVX, AVX2 et AVX-512. L'ensemble des opérations est effectué sur des nombres à virgule flottante en simple précision (32 bits).

Pour tirer parti de ces jeux d'instructions parallèles, les structures accélératrices sont construites de façon à ce que leurs nœuds contiennent un nombre de branches égal au nombre de flottants à simple précision pouvant être traités simultanément par le jeu d'instructions SIMD utilisé (4 pour le SSE, 8 pour l'AVX et l'AVX2 et 16 pour l'AVX-512). Ainsi, l'exploration des structures accélératrices au moment du lancer de rayons se trouve accélérée.

Il est également possible de lancer des rayons par paquets de 4, 8 ou 16. Dans ce cas, les différentes branches des nœuds des structures accélératrices sont testées séquentiellement, mais les rayons sont traités simultanément. Ce mode d'exploitation des instructions SIMD est plus efficace que le précédent dans le cas de rayons **cohérents**, ou suivant des parcours d'exploration de l'arbre des boîtes englobantes proches.

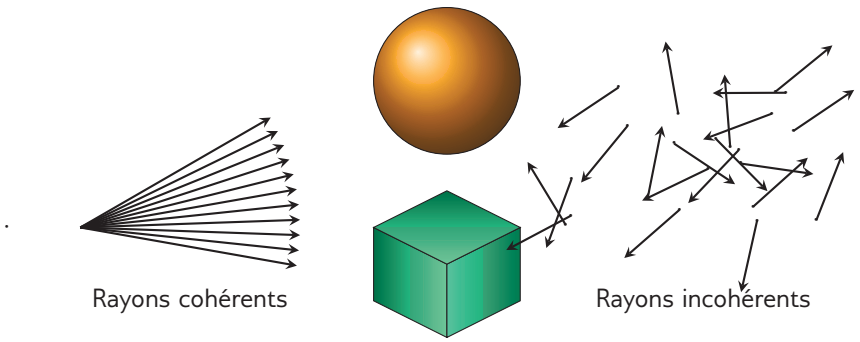


Figure 3.5 – Exemples de rayons cohérents (à gauche) et de rayons incohérents (à droite) lancés dans une scène

1. Intel *Many Integrated Core*, une architecture multiprocesseur de la marque Intel.

Embree propose une API de lancer de rayons géométrique exposant des services parmi lesquels on trouve :

- La construction de structures accélératrices de type BVH selon différentes heuristiques de construction. Celles-ci permettent, au choix :
  - De minimiser l'occupation mémoire de la BVH construite,
  - D'optimiser les performances de lancer de rayons cohérents,
  - D'optimiser, à l'inverse, les performances de lancer de rayons incohérents,
  - D'optimiser globalement le lancer de rayons sans préférence entre cohérence et incohérence.
- Des méthodes d'intersection de rayons avec la scène :
  - `rtcIntersect`, `rtcIntersect4`, `rtcIntersect8` et `rtcIntersect16`, permettant de lancer des rayons seuls ou par paquets en vue d'obtenir les informations précises de l'intersection trouvée (distance à l'origine, identifiant de la primitive intersectée, coordonnées locales de l'intersection...).
  - `rtcOccluded`, `rtcOccluded4`, `rtcOccluded8` et `rtcOccluded16`, fournissant une information booléenne sur la présence ou l'absence d'une intersection pour le rayon donné avec une limite de distance d'intersection donnée.
- Nativement, *Embree* supporte les maillages triangulaires, quadrangulaires, les géométries courbes, les surfaces de subdivision de Catmull-Clark (voir Catmull et Clark (1978)) et les géométries de cheveux (voir Woop et collab. (2014)).
- La possibilité de spécifier manuellement des géométries personnalisées (quadriques, quartiques, NURBS...) en fournissant des méthodes d'intersection, d'occlusion et de construction de boîtes englobantes.
- Des méthodes d'interpolation par paquets de grandeurs liées à la géométrie (typiquement, des textures).

Les appels aux méthodes d'intersection, d'occlusion et d'interpolation sont réentrants [2], qu'ils soient effectués par paquets ou pour des rayons seuls. Ainsi, appliquer un parallélisme de tâches sur des appels aux méthodes d'*Embree* devrait suffire à exploiter correctement, pour la phase de lancer de rayons, un CPU multicœur moderne doté d'unités d'exécution vectorielles.

Cette bibliothèque, utilisée par de nombreux moteurs de rendu professionnels, offre l'ensemble des méthodes nécessaires à un lancer de rayons performant en encapsulant les mécaniques liées à l'accélération du lancer de rayons. Un des principaux apports d'*Embree* est la vectorisation des opérations d'intersection géométrique et d'exploration d'arbres, qui nécessite un code complexe pour efficacement gérer l'ensemble des architectures visées.

Cette complexité est camouflée, mais l'API se place néanmoins à un niveau d'implémentation assez bas pour permettre son utilisation à des fins autres que le rendu photo-réaliste. Ainsi, il nous est possible de l'utiliser pour la simulation de champ ultrasonore en utilisant l'opération géométrique d'intersection d'un rayon avec une scène, indépendante des calculs physiques.

Nvidia *OptiX*

La principale cause du coût important des méthodes de rendu par lancer de rayons vient du fait qu'il est nécessaire de lancer un nombre de rayons très important

2. Utilisables simultanément par plusieurs tâches sans causer de problèmes de concurrence ou de blocages

pour obtenir une image. Ainsi, selon les effets modélisés, pour une image d'environ un million de pixels d'une scène raisonnablement complexe, il peut être nécessaire de lancer plusieurs dizaines voire centaines de millions de rayons.

Chaque rayon étant indépendant des autres rayons à lancer, le problème est aisément parallélisable. Ainsi, l'utilisation d'architectures massivement parallèles comme le GPU semble particulièrement adaptée. La bibliothèque *OptiX* de Nvidia (voir Parker et collab. (2010)) fournit un service permettant de créer un outil de lancer de rayons géométrique basé sur CUDA pour les GPU de la marque Nvidia.

Ce service est exposé à un niveau plus élevé que l'API d'*Embree*. Pour utiliser *OptiX*, après avoir renseigné une géométrie et ordonné la construction d'une structure accélératrice associée à celle-ci, l'utilisateur doit fournir des paquets de rayons sous la forme de listes de points d'origine et de directions de lancer que le GPU se charge de lancer. En plus de cela, il est nécessaire de spécifier, selon les géométries intersectées, les comportements suivants :

— L'algorithme d'intersection entre une primitive et un rayon,
— L'algorithme permettant la construction de la boîte englobante d'une primitive,
— Les instructions exécutées à une intersection entre un rayon et une primitive,
— Les instructions exécutées dans le cas d'un rayon n'ayant rien intersecté.

Ces comportements sont spécifiés sous la forme de code CUDA que le GPU exécute à chaque intersection ou occlusion en fonction des paramètres spécifiés par l'utilisateur.

OptiX laisse globalement moins de liberté et de flexibilité à l'utilisateur du fait que son code source n'est pas distribué (l'utilisation des binaires distribués à des fins commerciales requiert d'ailleurs une licence) et nécessite de travailler avec des paquets de rayons importants pour bénéficier efficacement de la puissance de calcul du GPU. Ainsi, *OptiX* est tout à fait indiqué pour des problèmes nécessitant le lancement de nombreux rayons (idéalement cohérents) simultanément, tel que le rendu graphique. Il présente moins d'intérêt pour des algorithmes nécessitant des traitements divergents entre les rayons à lancer ou impliquant des processus itératifs ou adaptatifs.

Puisqu'*OptiX* n'est pas un moteur de rendu complet, mais un outil de construction et d'exploration de structures accélératrices, il est possible de s'en servir pour toute autre application nécessitant un lancer de rayons géométrique. Il nous est donc possible de l'utiliser pour de la simulation d'ondes ultrasonores. Cependant, les travaux de simulation de champ ultrasonore que nous présentons ont été effectués sur CPU, nous nous intéressons donc à la bibliothèque *Embree* dans la suite.

3.1.3 Performances du lancer de rayons *Embree*

Afin d'évaluer le coût des opérations géométriques de lancer de rayon sur CPU, nous procédons à des essais de performance de la bibliothèque *Embree* en effectuant des lancers de rayons sur un maillage triangulaire d'une pièce industrielle et en faisant varier les paramètres suivants :

— Finesse du maillage (nombre de triangles),
— Largeur des paquets de rayons,
— Cohérence/incohérence des rayons.

Embree propose, à son instanciation, de choisir d'utiliser ou non un mode d'intersection de structure accélératrice dit « robuste ». Celui-ci limite les optimisations réduisant la précision des opérations arithmétiques. Une différence majeure que nous

avons pu observer en activant ce mode est la façon dont les arêtes d'un maillage sont intersectées. L'utilisation du mode robuste permet de réduire drastiquement le nombre de rayons passant entre deux triangles d'un maillage.

Comme nous le verrons par la suite, le fait de perdre un rayon peut singulièrement ralentir la convergence de notre algorithme de lancer de rayons. Le mode robuste présente également l'intérêt de pouvoir intersecter des maillages plus fins à des distances plus importantes. Pour ces raisons, nous choisissons d'activer le mode robuste d'*Embree* lors des opérations de lancer de rayons.

Influence de la largeur des paquets de rayons

Afin d'évaluer l'efficacité de la vectorisation du lancer de rayons dans *Embree*, nous en avons comparé les performances pour des rayons identiques seuls, par paquets de quatre et de huit rayons. Ces essais ont été effectués sur une machine unique, sans parallélisation de tâches, en utilisant la version **2.10.0** d'*Embree* compilée pour le jeu d'instructions AVX. Une version compilée pour le jeu d'instructions SSE2 a également été utilisée pour servir de référence.

Le processeur utilisé est un Intel Xeon E5-1650v2@3.5GHz, de la génération *Sandy Bridge*, en mesure d'exécuter des instructions SSE et AVX. Pour ces mesures de performances, les calculs d'intersection de rayons sont effectués sur le maillage présenté en figure 3.6, comprenant 23 774 triangles. Les rayons sont lancés depuis un point à l'intérieur de la pièce.

Figure 3.6 – Maillage d'une variation de section sur un tuyau en acier, 23 774 triangles

Dans un premier temps, nous construisons des paquets homogènes de rayons. Leur origine est la même et la variation de direction est aléatoire, mais faible : ils sont aléatoirement choisis dans un cône d'une ouverture angulaire d'environ 6°. Ils sont ensuite lancés par paquets de taille variable 100 000 fois d'affilée.

Dans ces conditions, l'exploration de la structure accélératrice est similaire pour l'ensemble des rayons de chaque paquet. Par ailleurs, nous choisissons l'option de construction de structure accélératrice dans *Embree* optimisant les performances

Taille des paquets	Performances (Mrays/s)	Accélération
1	3.9	× 1.00
4	3.7	× 0.95
8	—	—

(a) Jeu d'instructions SSE2

Taille des paquets	Performances (Mrays/s)	Accélération
1	4.0	× 1.03
4	4.5	× 1.15
8	—	—

(b) Jeu d'instructions SSE4.2

Taille des paquets	Performances (Mrays/s)	Accélération
1	4.0	× 1.03
4	4.6	× 1.18
8	5.2	× **1.33**

(c) Jeu d'instructions AVX

Table 3.1 – Performances en lancer de rayons homogènes

pour les rayons cohérents. Les conditions sont donc réunies pour observer de bonnes accélérations en SIMD.

Pour mesurer l'apport des instructions SIMD, nous exécutons le même test de performance en utilisant des versions d'*Embree* compilées respectivement en SSE2, SSE4.2 et AVX. Les résultats de ce test sont donnés en table 3.1.

Les gains de performances en lien avec l'utilisation de paquets de rayons sont inférieurs aux rapports théoriques de largeur de paquets. Ainsi, en utilisant des instructions AVX de largeur 256 bits, soit 8 flottants en simple précision, on observe un gain de performances maximal d'environ 30%. Les deux causes principales de ce résultat sont :

— L'exploration de structures accélératrices est un problème qui, bien qu'il soit aisément parallélisable, se prête mal à la vectorisation. En effet, même dans le cas de rayons homogènes, selon la complexité du maillage et la distance d'intersection du rayon (impactant directement son ouverture), l'exploration de la structure accélératrice peut fortement varier d'un rayon à l'autre.

— Nous comparons deux versions exploitant les instructions SIMD de manière différente. En effet, même en utilisant le lancer de rayons unitaire, l'exploration de la structure accélératrice est vectorisée puisque les tests d'intersection avec les boîtes englobantes aux différents niveaux de la structure accélératrice se font par paquets.

Pour mesurer l'impact de l'ouverture angulaire et donc de la régularité des explorations de structure accélératrice sur les performances de lancer de rayons, nous

la faisons varier et mesurons l'évolution des performances, toujours sur la même configuration.

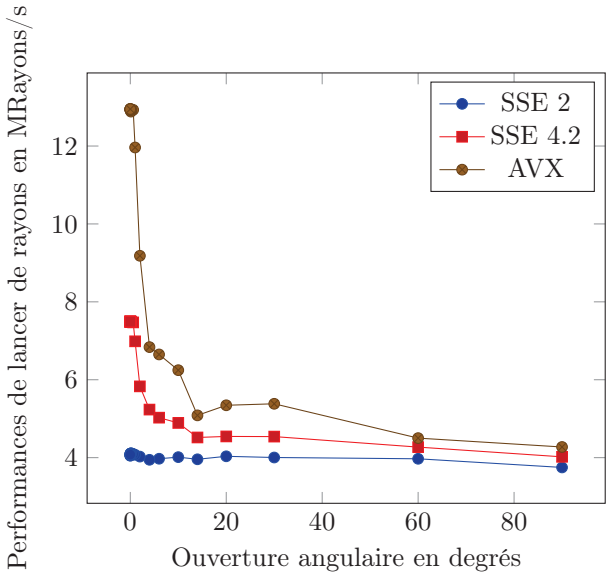


Figure 3.7 – Évolution des performances de lancer de rayons en fonction de l'ouverture angulaire, en SSE2, SSE4.2 et AVX

Les résultats en figure 3.7 mettent en évidence une chute de performances du lancer de rayons par paquets avec l'augmentation de l'ouverture angulaire des paquets de rayons. Dans la suite, nous utilisons le lancer de rayons paquets uniquement dans le cas de rayons très proches les uns des autres.

Influence de la complexité du maillage

Lors de l'exploration de la structure accélératrice, l'outil d'intersection de rayons utilise un algorithme d'exploration d'arbre qui réduit la complexité en intersection de $O(n)$ à $O(log(n))$ avec n le nombre de primitives géométriques de la scène.

Nous mesurons les cadences maximales de lancer de rayons sur plusieurs géométries en faisant varier la densité de triangles sur un même maillage afin de vérifier l'impact réel de la complexité du maillage intersecté sur les performances de l'outil *Embree*. Nous reprenons tout d'abord le maillage utilisé pour le test précédent avec différentes densités de triangles.

On observe en figure 3.8 l'évolution des performances en fonction du nombre de triangles du maillage. Nous y avons également tracé la régression en logarithme inverse des données de mesure. Bien que les données de mesure suivent une tendance à la décroissance similaire à une courbe en logarithme inverse, nous constatons des écarts importants entre le modèle de complexité logarithmique et les données expérimentales.

Ce phénomène s'explique par le fait que les raffinements successifs du maillage utilisés ont été construits indépendamment. Ainsi, même si les rayons utilisés d'un test à l'autre sont les mêmes, ils intersectent des zones de structures accélératrices dont la densité et la profondeur sont susceptibles de varier d'un maillage à l'autre.

De plus, nous n'avons pas de maîtrise sur le mode de construction des structures accélératrices et il est probable que certains maillages résultent en des structures

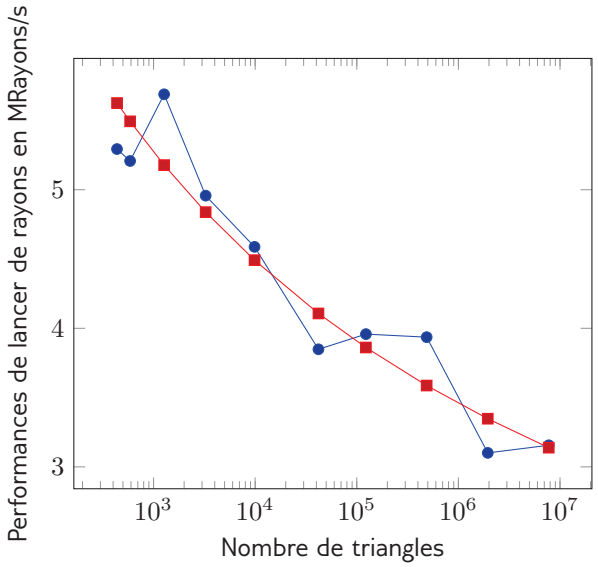


Figure 3.8 – Évolution des performances de lancer de rayons en fonction de la densité du maillage

accélératrices mieux équilibrées que d'autres, résultant en des performances d'intersection supérieures à d'autres comportant pourtant moins de triangles.

Cependant, la conclusion principale de ces mesures confirme la tendance générale suivie par les performances en lancer de rayons avec *Embree* lors de l'augmentation du nombre de triangles de la géométrie intersectée. En effet, nous pouvons ici constater qu'en augmentant le nombre de triangles d'un facteur $\sim \times 18\ 000$ les performances du lancer de rayons ne diminuent que d'un facteur d'environ 1.8.

3.1.4 Application à la simulation d'ondes ultrasonores

Calcul inverse

Le rayonnement ultrasonore produit par un traducteur est modélisable par une intégrale de diffraction, dite intégrale de Rayleigh-Sommerfeld (voir Royer et Dieulesaint (1996)). Pour des configurations de contrôle simples, il est possible de calculer analytiquement le résultat de cette intégrale. Les configurations que nous cherchons à simuler ne permettant pas un calcul analytique, nous évaluons cette intégrale numériquement.

Ainsi, pour calculer le champ ultrasonore émis par un traducteur en un point M donné, il est nécessaire de cumuler l'ensemble des contributions reçues en M dans toutes les directions de l'espace. Une première façon de faire peut consister à simuler, comme schématisé en figure 3.9, l'émission des fronts d'onde émis par chaque élément d'une discrétisation du traducteur. La sommation de l'ensemble des déplacements induits donne la valeur du champ de déplacement en M.

Pour un traducteur décomposé en n points sources, il est ainsi nécessaire de simuler la propagation de n fronts d'onde en provenance du traducteur et de trouver leur intersection avec chaque point de champ. Par ailleurs, selon le découpage choisi pour la surface du traducteur (celui-ci influant sur la précision de la reconstitution du champ), le nombre n de fronts d'onde simulés peut être important, limitant de fait les performances du calcul de champ.

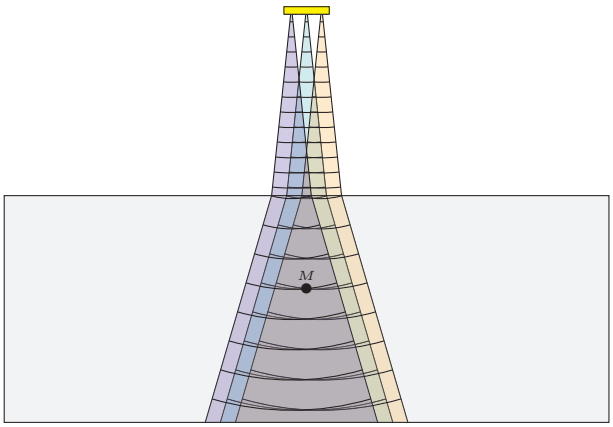


Figure 3.9 – Simulation de la propagation de fronts d'onde

Par principe de réciprocité, puisque la réponse du champ de déplacement en un point M à une impulsion émise depuis un élément du traducteur E est égale à la réponse du champ de déplacement en ce même élément E à une impulsion égale émise depuis le point M, il est possible d'effectuer le **calcul de champ en inverse**. Ainsi, au lieu de simuler la propagation des fronts d'onde émis par chaque point source du traducteur et de déterminer leur valeur en chaque point d'échantillonnage du calcul de champ, on calcule la propagation d'un front d'onde sphérique virtuel émis par le point de champ dans toutes les directions de l'espace, comme présenté en figure 3.10.

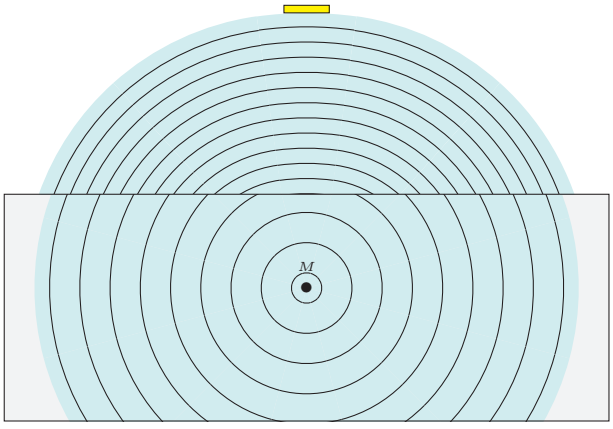


Figure 3.10 – Calcul de champ inverse

Cette façon de calculer le champ ultrasonore présente l'avantage de réduire le nombre de fronts d'onde à propager par point de champ, quel que soit l'échantillonnage du traducteur considéré. La méthode inverse est particulièrement intéressante pour des traducteurs étendus et finement échantillonnés. Par ailleurs, nous le verrons par la suite, il est plus simple et moins coûteux d'intersecter le front d'onde avec le traducteur qu'avec un point de champ. Nous choisissons donc d'effectuer l'ensemble de nos calculs depuis le point de champ.

3.1.5 Échantillonnage du front d'onde ultrasonore

Le calcul inverse fournit une méthode pour calculer la somme des fronts d'onde arrivant à un point de champ. Pour ce faire, il suffit de calculer la propagation d'un front d'onde sphérique virtuel partant du point de champ.

Pour connaître le champ induit lors de l'émission d'un signal par le traducteur, nous pouvons dans un premier temps calculer la réponse du milieu de propagation à une impulsion virtuelle émise par le point de champ et mesurer son impact sur chaque élément du découpage du traducteur.

Ce faisant, on obtient pour chaque élément une réponse impulsionnelle élémentaire qui traduit le déplacement réel induit par le point source sur le point de champ. En sommant l'ensemble des contributions, on obtient une réponse impulsionnelle globale modélisant l'effet d'une impulsion mécanique émise par le traducteur sur le point de champ considéré. La figure 3.11 schématise ce processus.

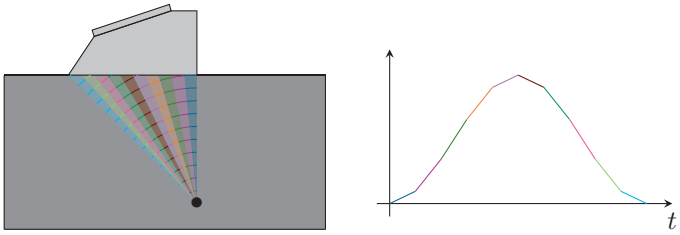


Figure 3.11 – Calcul des réponses impulsionnelles élémentaires

Pour un traducteur multiéléments, on procède de la même manière en calculant l'ensemble des réponses impulsionnelles élémentaires associées à un découpage de chaque élément. Avant d'en effectuer la sommation, les lois de retard et d'amplitude sont appliquées conformément à leur définition pour chaque élément du traducteur.

Pour calculer la réponse impulsionnelle élémentaire induite par un élément du découpage du traducteur sur le point de champ considéré, il est nécessaire de connaître à la fois l'amplitude et l'instant d'arrivée d'une perturbation unitaire après propagation. Pour ce faire, nous utilisons des rayons d'onde ultrasonore échantillonnant le front d'onde.

3.2 Modèle de rayons ultrasonores

Cette section a pour premier objectif de préciser le modèle de lancer de rayons ultrasonores en explicitant les grandeurs calculées pour un rayon. Puis, nous y indiquons les informations sur les milieux de propagation nécessaires à la simulation. Nous détaillons ensuite le procédé de lancer de rayons ultrasonores dans des milieux homogènes et hétérogènes avant d'aborder les questions de précision et de performance de l'implémentation réalisée.

La figure 3.12 schématise la propagation d'un rayon ultrasonore dans une pièce comportant trois milieux. Comme cela a été vu en section 2.3, une même onde peut donner naissance à plusieurs ondes de modes de polarisation différents à une interface. Ainsi, le suivi d'un rayon ultrasonore est en fait l'exploration d'un arbre de rayons. Nous nous intéressons dans cette section au suivi d'une branche de l'arbre, caractérisée par la séquence de modes de polarisation qui lui est associée.

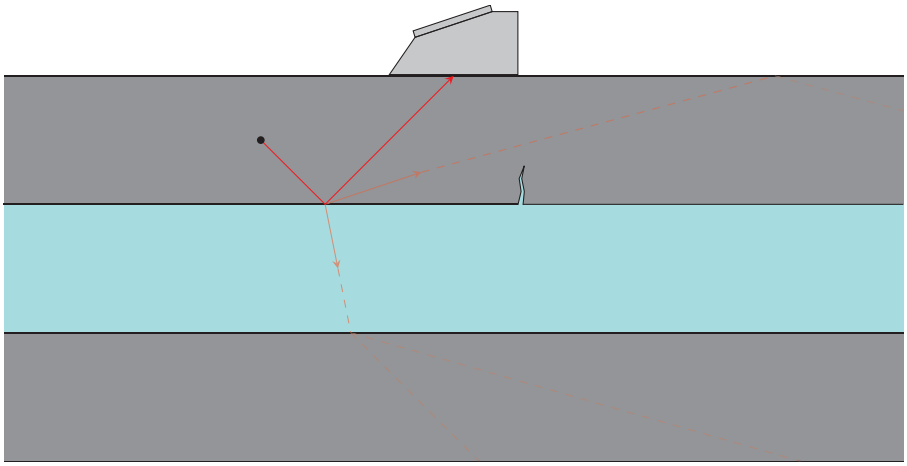


Figure 3.12 – Suivi de rayons ultrasonores

3.2.1 Rayon ultrasonore

⟨*RayonUS*⟩≡

```
struct RayonUS {
    ⟨Données de RayonUS  75⟩
};
```

Un rayon est tout d'abord un objet géométrique défini par un point d'origine dans l'espace et par une direction. Cette direction est donnée par la **direction d'énergie** de l'onde ultrasonore. En milieux isotropes, elle est identique à la direction de phase. Comme vu en section 2.2.1, en milieux anisotropes, ces deux directions sont différentes.

Afin d'éviter des calculs redondants et puisque les directions de phase et d'énergie sont abondamment utilisées lors des calculs de rayons ultrasonores, nous choisissons des les conserver en mémoire.

En général, lorsque l'on instancie un rayon, la donnée d'entrée est la direction de phase. En effet, même s'il est possible de calculer numériquement les directions de phase associées à une direction d'énergie en utilisant par exemple la méthode proposée par Vavryčuk (2006), le résultat n'est pas unique. Géométriquement, cela revient à chercher, pour une direction $\vec{d}$ donnée, l'ensemble des points d'une surface de lenteur tels que la normale en ces points égale $\vec{d}$.

⟨*Données de RayonUS*⟩≡

```
Vecteur3D origine;
Vecteur3D directionPhase;
Vecteur3D directionÉnergie;
```

Vecteur3D 21

Par ailleurs, il peut être utile de filtrer les intersections du rayon géométrique à l'aide de bornes de distance d'intersection. On spécifie donc des distances minimale et maximale d'intersection. En général, la distance minimale est spécifiée à un seuil faible (typiquement 10^{-4} pour des flottants en simple précision) visant à éviter les auto-intersections de primitives dues à un défaut de précision de la position du point d'origine. La distance maximale est, dans la suite, toujours spécifiée à la valeur maximale possible selon le type de flottants utilisé.

⟨*Données de RayonUS*⟩+≡
```
float distanceMin;
float distanceMax;
```

⟨*Limites de rayon*⟩≡
```
rayon.distanceMin = 10^-4;
rayon.distanceMax = ∞;
```

Pour calculer le temps mis par une onde pour aller d'un point à un autre, ce qui est indispensable pour la reconstitution des réponses impulsionnelles élémentaires, il est nécessaire de disposer de la lenteur de propagation de l'onde.

En choisissant de stocker la lenteur s plutôt que la vitesse v, le temps de vol t d'un rayon ayant parcouru une distance d se calcule de la sorte :

$$t = sd$$

Avec la vitesse :

$$t = \frac{d}{v}$$

Les deux approches étant sensiblement équivalentes du point de vue des résultats, nous choisissons de stocker la lenteur plutôt que la vitesse pour éviter d'utiliser des divisions.

⟨*Données de RayonUS*⟩+≡
```
float lenteur;
```

De plus, des informations sur la polarisation sont nécessaires pour reproduire le déplacement induit par l'onde simulée en un point de l'espace. Enfin, pour pouvoir appliquer un filtrage des ondes par mode, on choisit d'identifier le mode de polarisation (L, T, qL, $qT1$ ou $qT2$) pour chaque rayon ultrasonore.

⟨*Données de RayonUS*⟩+≡
```
Mode mode;
Vecteur3D polarisation;
```
Vecteur3D 21

⟨*Mode*⟩≡
```
enum Mode {
    ⟨ModeIsotrope 76⟩
    ⟨ModeAnisotrope 76⟩
};
```

⟨*ModeIsotrope*⟩≡
```
Mode_L,
Mode_T,
```

⟨*ModeAnisotrope*⟩≡
```
Mode_qL,
Mode_qT1,
Mode_qT2
```

3.2.2 Milieux de propagation

⟨*Milieu*⟩≡
```
class Milieu {
   ⟨Données de Milieu  77⟩
   ⟨Méthodes de Milieu ⟩
}
```

La définition d'un milieu de propagation doit permettre, pour une direction de propagation et un mode donnés, de déduire l'ensemble des caractéristiques d'un rayon ultrasonore.

⟨*Méthodes de Milieu*⟩+≡
```
virtual RayonUS propagation(Vecteur3D origine,
                            Vecteur3D direction,
                            Mode modePropagé) = 0;
```
RayonUS 75
Vecteur3D 21
Mode 76

De même, elle doit permettre simultanément la résolution de l'équation de Snell-Descartes et de l'équation de propagation, développée en section 2.3.

⟨*Méthodes de Milieu*⟩+≡
```
virtual RayonUS interaction(RayonUS rayonIncident,
                            Vecteur3D normale,
                            Mode modeGénéré) = 0;
```
RayonUS 75
Vecteur3D 21
Mode 76

Par ailleurs, comme présenté en annexe D, quel que soit le type de milieu considéré, afin de résoudre la condition d'égalité des contraintes à une interface et calculer les coefficients de Fresnel, il est nécessaire de disposer de la densité ρ (en $kg.m^{-3}$) de chaque matériau modélisé.

⟨*Données de Milieu*⟩≡
```
float densité;
```
Milieu 77

Milieu isotrope

⟨*MilieuIsotrope*⟩≡
```
class MilieuIsotrope: public Milieu {
   ⟨Données de MilieuIsotrope  77⟩
   ⟨Méthode de propagation en milieu isotrope  78⟩
   ⟨Méthode d'interaction en milieu isotrope  86⟩
}
```
Milieu 77

Puisque les ondes d'un mode donné se propagent toujours à la même vitesse dans un milieu isotrope, il suffit d'inclure les lenteurs de propagation des ondes L et T. Dans les milieux isotropes liquides, la lenteur des ondes L suffit puisque les ondes T ne peuvent pas se propager.

⟨*Données de MilieuIsotrope*⟩≡
```
float lenteurL;
float lenteurT;
```

Milieu anisotrope

⟨*MilieuAnisotrope*⟩≡
```
class MilieuAnisotrope: public Milieu {
   ⟨Données de MilieuAnisotrope  78⟩
   ⟨Méthode de propagation en milieu anisotrope ⟩
   ⟨Méthode d'interaction en milieu anisotrope ⟩
}
```
Milieu 77

Comme cela a été détaillé en section 2.2.3, le calcul des grandeurs caractéristiques d'une onde ultrasonore se propageant dans un milieu anisotrope nécessite la matrice de Christoffel Γ. Pour une direction donnée, celle-ci est calculée à l'aide

du tenseur des constantes élastiques. Il caractérise le comportement mécanique du matériau dans une approximation de petits déplacements.

Ainsi, en plus de comporter la densité ρ du matériau, notre représentation d'un milieu anisotrope doit comporter ce tenseur.

⟨*Données de MilieuAnisotrope*⟩≡

```
float tenseurConstantesÉlastiques[21];
```

3.3 Propagation en milieu homogène

La propagation d'un rayon ultrasonore en milieu isotrope commence par la détermination des caractéristiques de l'onde. Pour une direction de propagation donnée, le processus permettant de calculer la vitesse de propagation, la direction d'énergie et la polarisation de l'onde est détaillé en section 3.3.1.

On s'intéresse également à la première intersection rencontrée par le rayon et à ses caractéristiques, qui servent soit à traiter les milieux hétérogènes, soit à connaître la nature de la contribution des ondes arrivées sur le traducteur. Le détail de ces calculs est donné section 3.3.2.

Dans cette section, nous nous appuyons sur les développements théoriques effectués en section 2.2 pour en dégager un algorithme général de lancer de rayons ultrasonores, dans un premier temps en milieux homogènes.

3.3.1 Initialisation des rayons

En milieu isotrope

Premièrement, comme cela a été expliqué en section 2.2.2, les directions de phase et d'énergie sont identiques en milieux isotropes. Puisque la direction de phase est une donnée d'entrée, aucun calcul n'est nécessaire pour obtenir ces résultats.

⟨*Méthode de propagation en milieu isotrope*⟩≡

RayonUS 75
Vecteur3D 21
Mode 76
modeIsotrope 76

```
RayonUS propagation(Vecteur3D origine,
                    Vecteur3D direction,
                    Mode modePropagé) {
   RayonUS rayon;
   if(modeIsotrope(modePropagé))
   {
      rayon.origine = origine;
      rayon.directionPhase = direction;
      rayon.directionÉnergie = direction;
      ⟨Limites de rayon 76⟩
      ⟨Calcul de lenteur en milieu isotrope 78⟩
      ⟨Calcul de la polarisation en milieu isotrope ⟩
   }
   return rayon;
}
```

Le calcul des lenteurs se réduit quant à lui à une distinction selon les modes de polarisation et ne dépend pas de la direction de propagation.

⟨*Calcul de lenteur en milieu isotrope*⟩≡

```
if(modePropagé == Mode_L) {
   rayon.lenteur = lenteurL;
}
else {
   rayon.lenteur = lenteurT;
}
```

Enfin, le calcul de la polarisation dépend du mode de l'onde. On distingue le mode longitudinal pour lequel la polarisation est colinéaire à la direction de propagation du mode transversal, dont la polarisation est orthogonale à la direction de propagation.

⟨*Calcul de polarisation en milieu isotrope*⟩≡

```
if(modePropagé == Mode_L) {
   rayon.polarisation = direction;
}
else {
   rayon.polarisation = vecteurOrthogonal(direction);
}
```

vecteurOrthogonal 21

En milieu anisotrope

Le calcul de la lenteur dans une direction de propagation donnée en milieu anisotrope nécessite la résolution d'une équation aux valeurs propres faisant intervenir la matrice de Christoffel Γ, définie en section 2.2.3 à partir du tenseur des constantes élastiques.

⟨*Méthode algébrique de propagation en milieu anisotrope*⟩≡

```
RayonUS propagation(Vecteur3D origine,
                    Vecteur3D direction,
                    Mode modePropagé) {
   if(modeAnisotrope(modePropagé)) {
      rayon.origine = origine;
      rayon.directionPhase = direction;
      ⟨Initialisation de la matrice de Christoffel 80⟩
      ⟨Résolution de l'équation de Christoffel ⟩
      ⟨Calcul des lenteurs 81⟩
      ⟨Tri des lenteurs 81⟩
   }
}
```

RayonUS 75
Vecteur3D 21
Mode 76
modeAnisotrope 76

Nous rappelons la définition de Γ pour une direction de propagation $\vec{d}$:

$$\Gamma_{il} = c_{ijkl} d_j d_k$$

⟨*indicesVoigt*⟩≡

```
static int tabVoigt[3][3] = { { 1, 6, 5 },
                      { 6, 2, 4 },
                      { 5, 4, 3 } };

int indicesVoigt(int i, int j) {
   return tabVoigt[i][j];
}

static int tabVoigt6[6][6] = { { 0, 1, 2, 3, 4, 5 },
                       { 6, 7, 8, 9, 10, 4 },
                       { 11, 12, 13, 14, 9, 3 },
                       { 15, 16, 17, 13, 8, 2 },
                       { 18, 19, 16, 12, 7, 1 },
                       { 20, 18, 15, 11, 6, 0 } };

int indicesVoigt(int i, int j, int k, int l) {
   int p = indicesVoigt(i, j);
   int q = indicesVoigt(k, l);

   return tabVoigt6[p][q];
}
```

⟨*Initialisation de la matrice de Christoffel*⟩≡
```
float matriceChristoffel[3][3] =
   { {0, 0, 0},
     {0, 0, 0},
     {0, 0, 0} };

for(int i, j, k, l ∈ ⟦0; 2⟧) {
   int id = indiceVoigt(i, j, k, l);
   matriceChristoffel[i][l] += tenseurConstantesÉlastiques[id] * direction[j] *
     direction[k];
}
```

Dans le cas d'une interaction avec une interface entre deux matériaux faisant intervenir au moins un matériau anisotrope, comme détaillé en section 2.3.4 on résout l'équation aux valeurs propres associée à une matrice G dépendante de l'onde incidente qui nécessite également la connaissance du tenseur des constantes élastiques.

L'ensemble des grandeurs nécessaires à la modélisation d'une onde ultrasonore pour la simulation de contrôle non destructif par ultrasons peut être calculé à l'aide du tenseur des constantes élastiques. Ainsi, la structure de données minimale utilisable pour modéliser le comportement d'un matériau anisotrope conformément aux développements proposés dans le chapitre 2 est le tenseur des constantes élastiques c_{ijkl} doté de 21 composantes indépendantes.

⟨*Données de MilieuAnisotropeAlgébrique*⟩≡
```
float tenseurConstantesÉlastiques[21];
```

Pour une direction de propagation et un mode donné, les lenteurs sont calculées à partir des valeurs propres de la matrice de Christoffel Γ. Avec λ une valeur propre de Γ, on calcule la lenteur s :

$$s = \sqrt{\frac{\rho}{\lambda}}$$

L'équation aux valeurs propres de Γ s'écrit :

$$|\Gamma - \lambda I_3| = 0$$

Soit :

$$\begin{vmatrix} \Gamma_{11}-\lambda & \Gamma_{12} & \Gamma_{13} \\ \Gamma_{21} & \Gamma_{22}-\lambda & \Gamma_{23} \\ \Gamma_{31} & \Gamma_{32} & \Gamma_{33}-\lambda \end{vmatrix} = 0$$

Après simplifications, on obtient l'équation polynomiale de degré 3 en λ suivante :

$$\lambda^3 + P * \lambda^2 + Q * \lambda + R = 0$$

Avec :

$$\begin{cases} P = \Gamma_{11} + \Gamma_{22} + \Gamma_{33} \\ Q = \Gamma_{11} * \Gamma_{22} - \Gamma_{12} * \Gamma_{12} + \Gamma_{22} * \Gamma_{33} - \Gamma_{23} * \Gamma_{23} + \Gamma_{11} * \Gamma_{33} - \Gamma_{13} * \Gamma_{13} \\ R = det(\Gamma) \end{cases}$$

⟨*Calcul des lenteurs*⟩≡
```
float polynome[4] = { 0, 0, 0, 0 };
polynome[0] = 1;
polynome[1] = matriceChristoffel[0][0] + matriceChristoffel[1][1] +
  matriceChristoffel[2][2];
polynome[2] = matriceChristoffel[1][1] * matriceChristoffel[2][2] -
            matriceChristoffel[1][2] * matriceChristoffel[1][2] +
            matriceChristoffel[2][2] * matriceChristoffel[3][3] -
            matriceChristoffel[2][3] * matriceChristoffel[2][3] +
            matriceChristoffel[1][1] * matriceChristoffel[3][3] -
            matriceChristoffel[1][3] * matriceChristoffel[1][3];
polynome[3] = déterminant(matriceChristoffel);

array<float, 3> valeursPropres;
array<float, 3> lenteurs;

valeursPropres = resolutionPolynome(polynome);

for(int i = 0; i < 3; ++i) {
   lenteurs[i] = racineCarrée(densité / valeursPropres[i]);
}
```

On distingue chacun des modes de polarisation en triant les lenteurs dans l'ordre décroissant, de telle sorte que $s_{qL} < s_{qT1} < s_{qT2}$.

⟨*Tri des lenteurs*⟩≡
```
trier(lenteurs, valeursPropres);

float valeurPropre;
valeurPropre = valeursPropres[indiceMode(mode)];

rayon.lenteur = lenteurs[indiceMode(mode)];
```

Tri des lenteurs, mêmes permutations appliquées aux valeurs propres.

La polarisation $\vec{u}$ est le vecteur propre de Γ associé à la valeur propre λ du mode considéré. La résolution du système linéaire à trois équations et trois inconnues suivant donne donc la polarisation :

$$\begin{cases} \Gamma_{11}u_1 + \Gamma 12u_2 + \Gamma 13u_3 = \lambda u_1 \\ \Gamma_{21}u_1 + \Gamma 22u_2 + \Gamma 23u_3 = \lambda u_2 \\ \Gamma_{31}u_1 + \Gamma 32u_2 + \Gamma 33u_3 = \lambda u_3 \end{cases}$$

⟨*Calcul de polarisation en milieu anisotrope*⟩≡
```
rayon.polarisation = calculVecteurPropre(matriceChristoffel, valeurPropre);
```

Enfin, pour la direction d'énergie, nous reprenons la formule donnée en section 2.2.3 aux facteurs constants près puisqu'on normalise le résultat.

⟨*Calcul de direction d'énergie en milieu anisotrope*⟩≡
```
for(int i, j, k, l ∈ ⟦0; 2⟧) {
     rayon.directionÉnergie[i] = tenseurConstantesÉlastiques[indicesVoigt(i, j,
       k, l)] *
      rayon.polarisation[j] * rayon.polarisation[l] * rayon.direction[k];
}
normaliser(rayon.directionÉnergie);
```

indicesVoigt 79

3.3.2 Calcul d'intersection

Une fois les caractéristiques du rayon initialisées, il nous faut calculer le premier impact du rayon avec une primitive de la scène de contrôle. Celle-ci peut être la surface du traducteur ou une limite de matériau, constituant une interface avec un autre matériau. Dans les deux cas, nous déterminons :

— La **position de l'intersection** du rayon avec la primitive, pour en déduire l'élément du traducteur multiéléments intersecté ou le point de départ des rayons fils générés à une interface,
— La **distance de propagation** du rayon à l'intersection, qui nous permet de calculer le **temps de vol**, ou temps mis par l'onde pour atteindre l'intersection, afin d'en déduire la phase de l'onde.
— La normale au point d'intersection, utile aux calculs d'interface et au moment de l'intersection avec le traducteur.

Ces données peuvent être calculées à l'aide des résultats de l'opération d'intersection d'un rayon avec une scène telle que décrite en section 3.1.1 et fournie par la bibliothèque *Embree*. Spécifiquement, la méthode d'intersection d'*Embree* que nous utilisons prend en paramètres :

— Une scène de lancer de rayons construite au préalable et contenant les structures accélératrices (voir section 3.1.1),
— Un rayon identifié par une origine, une direction, une distance minimale et une distance maximale d'intersection.

⟨*Calcul d'intersection*⟩≡
```
void calculIntersection(RayonUS rayon, Vecteur3D & positionIntersection, float &
  tempsVol) {
   Intersection intersection = intersectionEmbree(scène, rayon);

   positionIntersection = positionIntersection(intersection, scène);
   tempsVol = tempsVol(intersection, rayon);
}
```
tempsVol 84

Position de l'intersection

Après le calcul d'intersection, on dispose d'une part d'un identifiant unique correspondant à la primitive géométrique intersectée ainsi que des coordonnées barycentriques u et v spécifiques à cette primitive et d'autre part de la distance d'intersection notée t. On dispose alors de deux méthodes pour retrouver la position finale M d'intersection :

— L'utilisation des coordonnées barycentriques u et v et des coordonnées des sommets v_0, v_1 et v_2 de la primitive, dans le cas d'un triangle :

$$M = (1-u-v).v_0 + u.v_1 + v.v_2$$

— Le calcul vectoriel depuis l'origine :

$$\vec{M} = \vec{O} + t.\vec{d}$$

La méthode basée sur les coordonnées barycentriques offre une meilleure précision, en particulier pour de fortes valeurs de t. En effet, l'ensemble des calculs est effectué sur des nombres à virgule flottante en simple précision (32 bits, norme IEEE 754). Les nombres sont représentés sont la forme d'approximations à la puissance de 2 la plus proche avec une précision limitée (voir en annexe B). Pour de fortes valeurs de t, l'arrondi choisi par l'unité de calcul flottant tend à causer des erreurs pouvant situer l'intersection en amont de la primitive intersectée, engendrant ainsi des auto-intersections, comme schématisé en figure 3.13.

Ces problèmes de précision sont réduits par l'approche barycentrique puisque les coordonnées barycentriques servent à localiser le point d'intersection dans une étendue spatiale délimitée par les sommets de la primitive triangulaire intersectée.

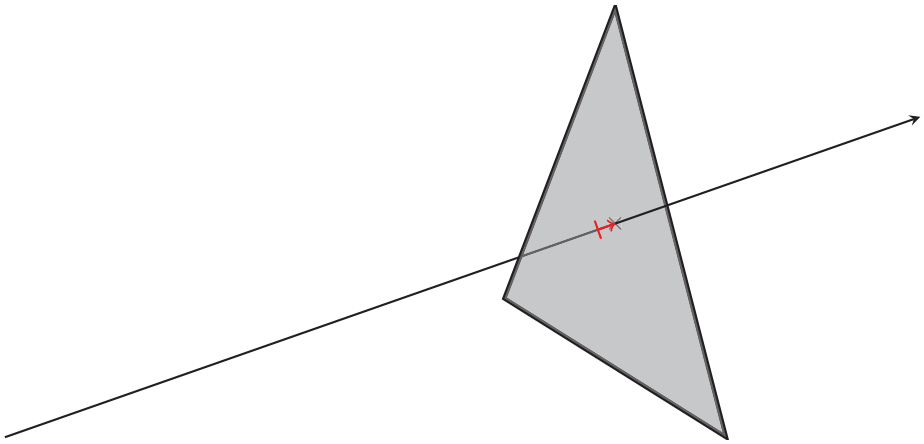


Figure 3.13 – Schématisation de l'erreur de précision en lancer de rayons

Par ailleurs, puisque le calcul est directement effectué en fonction des sommets du triangle, le point d'intersection est inclus dans le plan formé par ces trois plans avec plus de précision.

⟨*positionIntersection*⟩≡

```
Vecteur3D positionIntersection(Intersection intersection, Scène scène) {
   float u = donnéesIntersection.u;
   float v = donnéesIntersection.v;
   Triangle triIntersecté = scène.triangles[donnéesIntersection.idPrimitive];

   Vecteur3D posIntersection = (1 - u - v) * triIntersecté.v0 +                    Vecteur3D 21
                           u * triIntersecté.v1 +
                           v * triIntersecté.v2;
   return positionIntersection;
}
```

Temps de vol

Pour obtenir le temps de vol, ou temps mis par une onde pour aller de son origine à son intersection, dans un milieu isotrope, il suffit de considérer le rapport entre la distance d parcourue et la vitesse de propagation. On obtient, avec v la vitesse de propagation et s la lenteur :

$$t_{vol} = \frac{d}{v} = d.s$$

Cette relation est valable dans les milieux isotropes puisque la direction d'énergie et la direction de phase sont confondues. Or, en milieux anisotropes, ce n'est pas le cas. Lorsque nous lançons un rayon ultrasonore, la direction du rayon est celle de l'énergie de la perturbation mécanique. Elle se déplace selon la direction d'énergie (définie en section 2.2.1) avec une vitesse V_e dite vitesse d'énergie.

Cette vitesse est différente de la vitesse de phase V_φ qui correspond à la vitesse de propagation du front d'onde. En effet, comme le montre le schéma en figure 3.14, en considérant l'onde portée par le rayon de manière isolée sous la forme d'une onde plane, l'énergie parcourt en une période une distance supérieure à celle parcourue par la phase.

Ainsi, la vitesse d'énergie est supérieure à la vitesse de phase. Sa projection sur la direction de phase est égale à la vitesse de phase. On a donc :

$$V_\varphi = V_e.cos(\alpha) = \langle \vec{V}_e \cdot \vec{n} \rangle$$

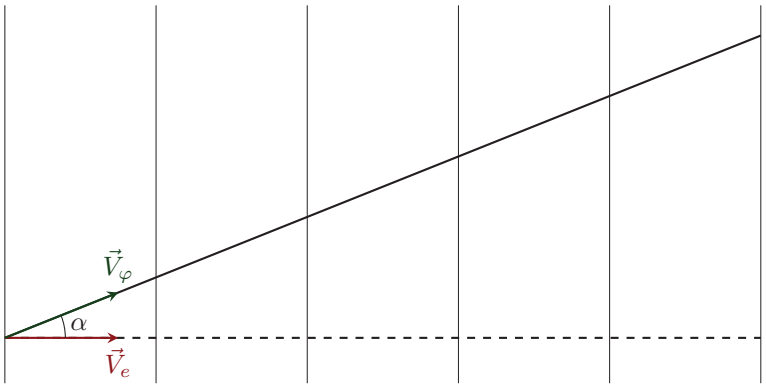


Figure 3.14 – Différence entre les distances parcourues par la phase (ligne pleine) et l'énergie(ligne pointillée)

Avec $\vec{n}$ la direction de phase normalisée.

Par conséquent, le temps de vol d'une onde en milieu anisotrope s'écrit :

$$t_{vol} = \frac{d}{V_e} = \frac{d.cos(\alpha)}{V_\varphi} = d.s.\langle \vec{V}_e \cdot \vec{n} \rangle$$

⟨*tempsVol*⟩≡

```
float tempsVol(Intersection intersection, RayonUS rayon) {
   return intersection.distance * rayon.lenteur *
         produitScalaire(rayon.directionÉnergie,
                         rayon.directionPhase);
}
```

Normale au point d'intersection

Le calcul de la normale au point d'intersection est calculé de la même manière que la position d'intersection. En effet, à la construction de la géométrie de la pièce, utilisée par l'outil d'intersection, chaque point du maillage se voit attribuer une normale. La normale au point d'intersection est construite par interpolation linéaire des normales de chaque sommet du triangle, à l'aide des coordonnées barycentriques du point d'intersection dans le triangle. Ce vecteur interpolé est ensuite normalisé.

⟨*normale*⟩≡

```
Vecteur3D normale(Intersection intersection, RayonUS rayon) {
   float u = donnéesIntersection.u;
   float v = donnéesIntersection.v;
   Triangle triIntersecté = scène.triangles[donnéesIntersection.idPrimitive];

Vecteur3D 21   Vecteur3D normale = (1 - u - v) * triIntersecté.n0 +
                           u * triIntersecté.n1 +
                           v * triIntersecté.n2;
   normaliser(normale);
   return normale;
}
```

Ce choix de construction de normale n'étant pas anodin, les raisons le justifiant sont détaillées en section 3.5.3.

3.4 Interaction avec des interfaces

Au cours de la simulation de propagation d'une onde ultrasonore par lancer de rayons, nous devons correctement gérer les cas d'intersection d'interfaces. Dans cette situation, comme cela a été expliqué en section 2.3, les conditions de continuité de contraintes et de déplacement à une interface présentant un contact parfait imposent de considérer des ondes générées de différents modes. Celles générées dans le milieu d'incidence sont les ondes réfléchies tandis que celles se propageant dans le milieu de l'autre côté de l'interface sont les ondes réfractées (ou transmises).

Dans cette section, nous détaillons la méthode de calcul de ces ondes en lancer de rayons pour étendre l'algorithme de lancer de rayons ultrasonores aux milieux hétérogènes, tout d'abord isotropes, puis anisotropes. Dans les deux cas, il s'agit de transcrire algorithmiquement les solutions aux équations de propagation et de Snell-Descartes à l'interface.

3.4.1 Interaction entre deux milieux isotropes

À une interface entre deux milieux isotropes, puisque les lenteurs pour chaque mode de polarisation sont connues de part et d'autre de l'interface, seules les directions de propagation restent à déterminer. Celles-ci sont soumises à l'équation de Snell-Descartes qui impose la valeur de la projection du vecteur lenteur sur l'interface ainsi qu'à l'équation de propagation qui impose la norme du vecteur lenteur dans chaque milieu (voir section 2.3.4).

On remarque que les milieux de part et d'autre de l'interface peuvent être considérés indépendamment une fois que la lenteur projetée a été déterminée. Par conséquent, pour chaque côté de l'interface, on distingue les modes L et T (sauf pour les milieux fluides, où seules les ondes longitudinales existent) et on peut générer jusqu'à deux ondes supplémentaires dont la direction vérifie :

$$\begin{cases} \langle \vec{s} \cdot \vec{i} \rangle = \langle \vec{s}_{incidente} \cdot \vec{i} \rangle \\ ||\vec{s}|| = s_{milieu} \end{cases}$$

Avec $\vec{i}$ un vecteur unitaire porté par l'interface et le plan d'incidence et s_{milieu} la lenteur imposée par le milieu de réflexion ou de réfraction. Ces conditions ont une solution analytique, détaillée dans l'équation 2.12.

⟨*directionGénéréeIsotrope*⟩≡

```
Vecteur3D directionGénéréeIsotrope(Rayon rayonIncident, Vecteur3D normale, float
  lenteur) {
   Vecteur3D vecteurLenteurProjeté = projectionLenteur(rayonIncident, normale);
   float lenteurNormale = racineCarrée(carré(lenteur) -
     normeCarrée(vecteurLenteurProjeté));
   Vecteur3D vecteurLenteur = vecteurLenteurProjeté + normale * lenteurNormale;
   Vecteur3D direction = normaliser(vecteurLenteur);

   return direction;
}
```

Le rayon généré cumule le temps de vol du rayon incident et a pour origine le point d'intersection du rayon incident avec l'interface. Le calcul de la polarisation reste le même qu'en propagation dans un milieu isotrope homogène.

⟨*Méthode d'interaction en milieu isotrope*⟩≡

```
RayonUS interaction(RayonUS rayonIncident,
                    Mode modeGénéré) {
   RayonUS rayon;

   if(!modeIsotrope(modePropagé)) {
      return rayon;
   }

   Vecteur3D positionIntersection;
   float tempsVol;
   calculIntersection(rayonIncident, positionIntersection, tempsVol);

   rayon.origine = positionIntersection;
   rayon.tempsVol = rayonIncident.tempsVol + tempsVol;
   ⟨Calcul de lenteur en milieu isotrope 78⟩

   Vecteur3D direction = directionGénéréeIsotrope(rayonIncident, normale,
     rayon.lenteur);

   rayon.directionPhase = direction;
   rayon.directionÉnergie = direction;
   ⟨Calcul de polarisation en milieu isotrope 79⟩
}
```

RayonUS 75
Mode 76
modeIsotrope 76
Vecteur3D 21
Calcul d'intersection 82

3.4.2 Interaction entre deux milieux anisotropes

L'anisotropie d'un matériau empêche la résolution directe de l'équation de Snell-Descartes en conjonction avec l'équation de propagation. Comme en propagation dans un milieu homogène anisotrope, la résolution d'une équation aux valeurs propres s'impose pour résoudre l'équation de propagation. Cette fois-ci, la direction de propagation n'est pas connue, mais on dispose de sa composante tangente à l'axe de l'interface.

En incorporant la décomposition du vecteur lenteur en une partie tangente et une partie normale σ à l'axe de l'interface dans la matrice de Christoffel, on obtient une matrice de taille 3×3 dont on cherche les valeurs propres. Le polynôme caractéristique de cette matrice est de degré 6 en σ. Le calcul de cette matrice, fastidieux mais sans difficulté particulière, n'est pas détaillé ici. Il reprend simplement l'équation 2.14.

La recherche des racines de ce polynôme peut nécessiter une méthode numérique telle que la méthode de Newton ou la méthode de Durand-Kerner (voir l'annexe E). Les racines obtenues correspondent aux composantes normales des lenteurs des modes générés à l'interaction dans le milieu anisotrope considéré. Puisque les coefficients du polynôme sont réels, les racines sont soit réelles, soit complexes conjuguées. Les racines réelles sont des modes propagés tandis que les racines complexes correspondent aux ondes évanescentes, que nous ne considérons pas dans le cadre de nos simulations.

⟨*Méthode algébrique d'interaction en milieu anisotrope*⟩≡

```
RayonUS interaction(RayonUS rayonIncident,
                    Mode modeGénéré) {
   RayonUS rayon;

   if(!modeAnisotrope(modeGénéré)) {
      return rayon;
   }

   Vecteur3D positionIntersection;
   float tempsVol;
```

RayonUS 75
Mode 76
modeAnisotrope 76
Vecteur3D 21

```
    calculIntersection(rayonIncident, positionIntersection, tempsVol);

    rayon.origine = positionIntersection;
    rayon.tempsVol = rayonIncident.tempsVol + tempsVol;

    float matriceChristoffel[3][3];
    ⟨ Calcul de la matrice de Christoffel pour Snell-Descartes, voir l'équation 2.14 ⟩
    float polynomial[7];
    ⟨ Calcul du polynôme de degré 6, voir l'équation 2.14 ⟩

    float racines[6] = racinesRéelles(polynomial);
    Vecteur3D directions[6];
    float lenteurs[6];

    Calcul des directions et lenteurs à une interaction en anisotrope
    ⟨identifierMode 87⟩
    return rayon;
}
```

Calcul d'intersection 82

On constate que les modes propagés peuvent être au nombre maximal de 6. En pratique, nous simulons uniquement les modes dont la direction d'énergie est cohérente avec le mode d'interaction recherché (réflexion ou transmission). La correspondance des solutions obtenues avec les modes de polarisation est effectuée à l'aide d'un calcul de propagation anisotrope dans la direction de phase trouvée.

⟨*identifierMode*⟩≡

```
identifierMode(Vecteur3D position, Vecteur3D directions[6], float lenteurs[6],
  Mode modePropagé) {
   for(int i = 0; i < 6; ++i) {
      RayonUS rayonModePropagé = propagation(positionIntersection,
        directions[i], modePropagé);
      if(rayonQL.lenteur ≃ lenteurs[i]) {
         rayon = rayonModePropagé;
      }
   }
}
```

3.5 Anisotropie géométrique

Nous disposons d'un ensemble d'outils permettant de simuler la propagation d'un rayon d'onde ultrasonore ainsi que ses interactions avec des interfaces entre matériaux, qu'ils soient isotropes ou anisotropes. Cependant, la simulation de propagation en milieux anisotropes implique des calculs coûteux. Dans le cas d'une simulation de contrôle non destructif, on cherche à calculer la valeur du champ ultrasonore en de nombreux points. Ainsi, pour plusieurs points de champ, on peut être amenés à effectuer des lancers de rayons dans des directions initiales identiques pour simuler la propagation d'une onde ultrasonore.

Or, bien que l'anisotropie implique une dépendance de la vitesse de propagation d'une onde ultrasonore à la direction de propagation, les caractéristiques du matériau ne changent pas avec la position, dans l'hypothèse d'un matériau homogène. Ainsi, on a :

$$s(\theta, \varphi, x, y, z) = s(\theta, \varphi)$$

Pour cette raison, il semble qu'effectuer un calcul algébrique complet des caractéristiques des ondes ultrasonores pour chaque direction de propagation simulée induise une certaine redondance. Nous présentons dans cette section une méthode de lancer de rayons ultrasonores en milieux anisotropes permettant de tirer parti de

cette redondance afin de réduire le coût calculatoire de la simulation de propagation d'ondes ultrasonores en milieux anisotropes.

3.5.1 Principe

Comme nous le verrons par la suite, les directions de propagation utilisées pour le lancer de rayons pour chaque point de champ varient. Il n'est donc pas possible de définir *a priori* un ensemble de directions exhaustif qui est utilisé par l'ensemble des points de champ.

Pour une direction de propagation $\vec{d}$ donnée, nous cherchons à obtenir la lenteur et la direction d'énergie correspondantes en réduisant les calculs nécessaires. Ceci revient à construire une fonction f rapide à évaluer et approchant le calcul de lenteur et de direction d'énergie. On a alors :

$$\forall(\theta,\varphi)\in[0,2\pi]\times[0,\pi], f(\theta,\varphi)=s(\theta,\varphi)$$

Puisqu'elles permettent d'obtenir simplement à la fois la lenteur et la direction d'énergie d'une onde tant en propagation, pour résoudre l'équation de propagation, qu'en interaction, pour résoudre l'équation de Snell-Descartes combinée à l'équation de propagation, les surfaces de lenteur (définies en section 2.2.1) sont très utiles dans cette situation.

Ainsi, pour calculer les lenteurs et directions d'énergie dans un milieu anisotrope avec un coût calculatoire moindre, il suffit de construire une représentation géométrique simple, mais précise des surfaces de lenteur. Cette représentation géométrique caractérise pleinement un matériau anisotrope homogène, il n'est donc nécessaire de la construire qu'une fois par matériau. Enfin, en utilisant un outil de lancer de rayons, nous verrons qu'il est alors possible de déterminer rapidement et efficacement les caractéristiques d'une onde ultrasonore.

Comme nous le verrons par la suite, une simple tabulation bidimensionnelle des lenteurs et directions d'énergie sur la sphère unité, si elle suffit au calcul des lenteurs et directions d'énergie pour simuler la propagation d'une ultrasonore dans un milieu anisotrope homogène, ne permet pas la résolution de l'équation de Snell-Descartes à une interface entre deux milieux anisotropes.

3.5.2 Construction des surfaces de lenteur

La construction des surfaces de lenteurs s'apparente au problème de reconstruction de surfaces. À partir d'un échantillonnage en points et en normales d'une surface définie par une équation polaire en θ et φ, on cherche à obtenir une structure interpolant les points d'échantillonnage facilement intersectable à l'aide d'un outil de lancer de rayons.

Balayage de la sphère unité

Une première solution pour échantillonner les surfaces de lenteur peut consister à utiliser un balayage régulier basé sur le système de coordonnées sphériques. Chaque position est alors identifiée par un couple latitude/longitude (θ,φ). Sur la sphère unité, les coordonnées cartésiennes du point associé au couple de coordonnées sphériques (θ,φ) s'écrivent :

$$\vec{r} = \begin{cases} x(\theta,\varphi) = sin(\varphi)cos(\theta) \\ y(\theta,\varphi) = sin(\varphi)sin(\theta) \\ z(\theta,\varphi) = cos(\varphi) \end{cases} \tag{3.1}$$

On constate cependant qu'un tel échantillonnage présente des variations de densité. Comme le schématise la figure 3.15, la densité des points d'échantillonnage augmente à l'approche des pôles de la sphère unité. En effet, la dérivée de la position par rapport à l'angle θ dépend de φ :

$$\vec{r} = \begin{cases} \frac{\partial x}{\partial \theta}(\theta,\varphi) = -sin(\varphi)sin(\theta) \\ \frac{\partial y}{\partial \theta}(\theta,\varphi) = sin(\varphi)cos(\theta) \\ \frac{\partial z}{\partial \theta}(\theta,\varphi) = 0 \end{cases} \quad \text{avec } (\theta,\varphi) \in [0,2\Pi] \times [0,\Pi]$$

En particulier, on observe que $\left\| \frac{\partial \vec{r}}{\partial \theta} = sin(\varphi) \right\|$. Par conséquent, la densité linéaire d'échantillons sur le méridien de latitude φ est inversement proportionnelle à $sin(\varphi)$. Ce phénomène génère un écart de précision entre les pôles et les autres zones de la sphère d'échantillonnage. La densité de points aux pôles est inutilement grande tandis que celle au niveau de l'équateur est plus faible. Un tel écart n'a pas de raison d'être puisque, sans connaissance préalable de la surface à mailler, la densité de points doit rester stable sur l'ensemble du maillage.

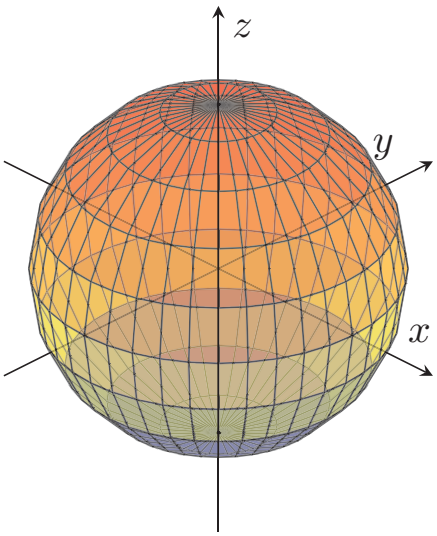


Figure 3.15 – Balayage uniforme en longitude et en latitude

Il nous faut donc utiliser une transformation T autre que celle proposée dans l'équation 3.1, associant un couple de coordonnées (u,v) du carré unité $[0,1]\times[0,1]$ à une coordonnée (x,y,z) de la sphère unité. Cette transformation, assimilable à une projection cartographique, doit vérifier les conditions suivantes afin d'être utilisable pour mailler efficacement les surfaces de lenteur :

— Elle doit conserver les rapports de surface. Ainsi, pour toute portion p du plan unité P_1, on a :

$$\frac{surface(p)}{surface(P_1)} = \frac{surface(T(p))}{surface(T(P_1))}$$

De cette façon, la densité de points d'échantillonnage est stable.

— La transformation T doit préserver les propriétés d'adjacence pour simplifier la construction du maillage. Ainsi, deux points voisins sur le plan de départ ont des images par T voisines. On peut alors se baser sur un maillage bidimensionnel simple sur le plan de départ pour construire le maillage de la surface de lenteur.

De nombreuses projections cartographiques existent, applicables dans différentes situations selon les propriétés qu'elles conservent, parmi les suivantes :

— Conservation des formes, on parle de projection **conforme**.
— Conservation des distances sur les méridiens, on parle de projection **équidistante**.
— Conservation locale des surfaces (et donc des rapports de surface), on parle de projection **équivalente**.

Il est possible de montrer mathématiquement qu'aucune projection ne peut être à la fois conforme et équivalente. Toute transformation applicable au maillage des surfaces de lenteur ne peut donc conserver les angles.

Shirley et Chiu (1997) proposent une transformation du plan vers un hémisphère vérifiant les propriétés d'équivalence et de continuité. En associant des carrés concentriques du plan à des cercles concentriques sur la sphère, cette transformation permet de conserver les rapports de surface tout en conservant la continuité (voir la figure 3.16).

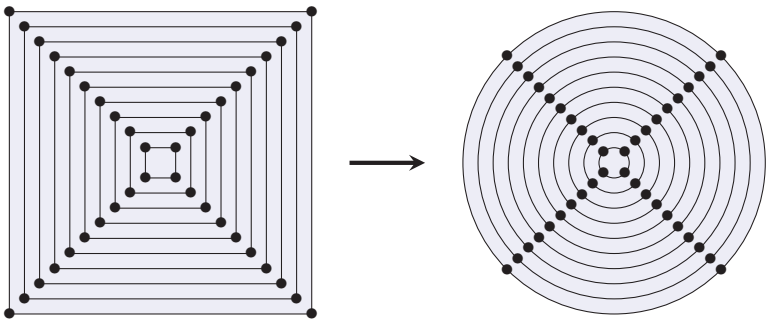


Figure 3.16 – Transformation concentrique de Shirley et Chiu (1997) autour de l'hémisphère

Praun et Hoppe (2003) explicitent une façon de découper la sphère unité en huit triangles, eux-mêmes associés chacun à un triangle dans le carré unité. Ce découpage, présenté en figure 3.17, conserve aussi les surfaces et la continuité.

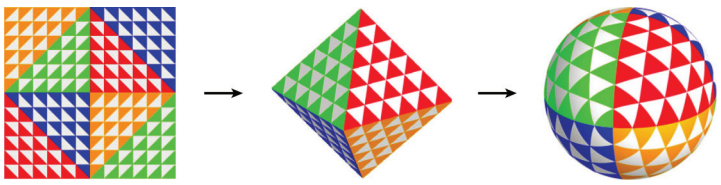


Figure 3.17 – Découpage de la sphère en triangles, obtenu par pliage du carré en un octaèdre projeté sur la sphère unité (image issue de Praun et Hoppe (2003))

Clarberg (2008) présente une implémentation d'une méthode associant chaque point du carré unité à un unique point de la sphère unité. Cette implémentation utilise la transformation de Shirley et Chiu (1997) tout en appliquant le découpage

proposé par Praun et Hoppe (2003) pour obtenir une transformation équivalente et continue du carré unité sur la sphère unité.

Nous choisissons d'utiliser cette méthode pour la construction de maillages des surfaces de lenteur. Un balayage uniforme de la sphère unité obtenu par cette méthode est présenté en figure 3.18.

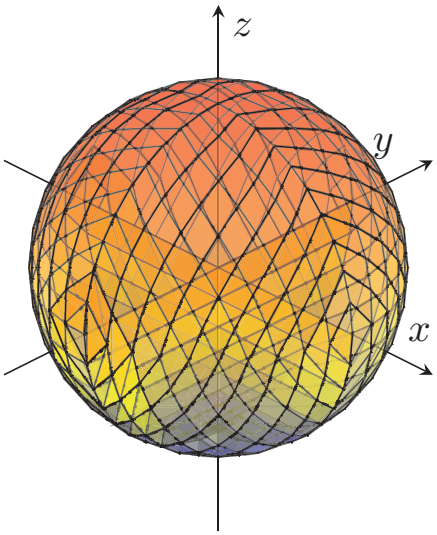


Figure 3.18 – Balayage uniforme de la sphère unité selon la méthode de Clarberg (2008)

Construction de la surface de lenteur

Une fois le choix des points d'échantillonnage réalisé, nous disposons d'un ensemble D de directions de propagation sur lesquelles nous effectuons les calculs de propagation en milieu anisotrope, tels que détaillés en section 3.3.1.

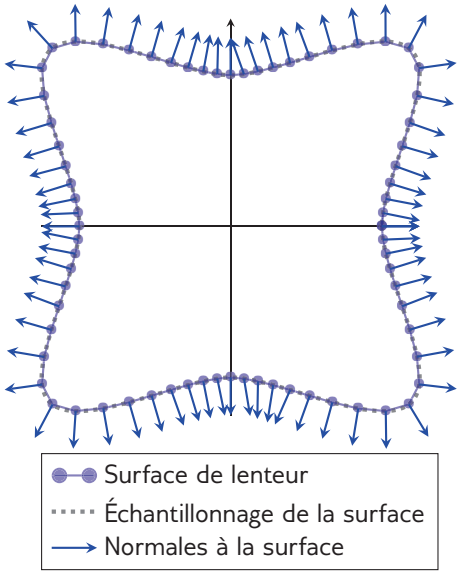


Figure 3.19 – Échantillonnage d'une surface de lenteur

Pour chaque direction $\vec{d}$ de l'ensemble D, nous obtenons donc une lenteur s et une

direction d'énergie $\vec{d}_e$. Ceci nous permet de placer un point de la surface de lenteur en position $s.\vec{d}$. Nous pouvons également caractériser la normale à la surface de lenteur en ce point : elle est égale à la direction d'énergie $\vec{d}_e$. La figure 3.19 schématise ce procédé.

En l'appliquant à l'ensemble des points d'échantillonnage de la sphère unité déterminés précédemment en section 3.5.2 pour chacun des modes de polarisation du milieu anisotrope considéré, on obtient un échantillonnage régulier des trois surfaces de lenteur qL, $qT1$ et $qT2$. Notons que puisque chaque direction de propagation est associée à une unique lenteur, la surface de lenteur est entièrement visible depuis l'origine (voir la figure 3.20).

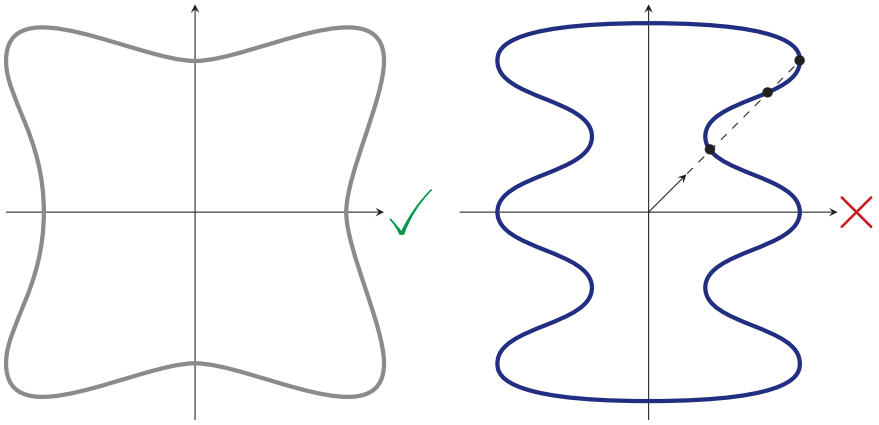


Figure 3.20 – Une surface de lenteur étant entièrement visible depuis l'origine, le schéma à droite ne représente pas une surface de lenteur

Plusieurs méthodes de construction de maillages à partir de nuages de points existent, tels que la triangulation de Delaunay (1934). Dans notre cas, il suffit cependant de construire un maillage triangulaire du carré unitaire, lequel sera reporté sur la surface de lenteur. Le maillage présenté en figure 3.17 en est un exemple.

Il reste finalement à construire une géométrie intersectable à partir de ces points d'échantillonnage. Pour ce faire, plusieurs méthodes sont utilisables, telles que la construction d'un maillage triangulaire, simple, rapide, mais pouvant nécessiter de nombreux triangles pour être précis. L'utilisation de transformées en ondelettes (voir Press (2007)) ou l'utilisation de surfaces polynomiales telles que les B-splines ou les *NURBS* (voir Piegl et Tiller (2012)) permettent une meilleure précision avec moins de primitives à stocker, limitant ainsi l'espace mémoire utilisé, mais compliquent le processus d'intersection.

Au vu de l'efficacité de l'opération d'intersection d'un rayon avec un maillage triangulaire et de l'influence modérée du nombre de triangles sur les performances d'intersection à l'aide de l'outil *Embree*, nous choisissons d'utiliser des surfaces de lenteur triangulées. Puisque les surfaces que nous construisons sont de classe $\mathscr{C}^1$, il n'est pas nécessaire de chercher à identifier les discontinuités lors de l'opération de maillage.

Pour une direction donnée, une intersection de maillage triangulaire, telle que proposée par les bibliothèques *OptiX* ou *Embree*, permet d'obtenir le triangle intersecté ainsi que les coordonnées barycentriques dans le triangle. Une interpolation linéaire permet alors de calculer les valeurs approchées des lenteurs et directions d'énergie.

Remarquons que, en application de la méthode de découpage octaédrique de la sphère de Praun et Hoppe (2003), ce maillage du carré unité est replié sur lui-même, ce qui signifie que les points en bordure sont identifiables aux autres points de bordure avec lesquels ils sont confondus dans le maillage tridimensionnel.

Enfin, pour chaque échantillon du maillage de la surface de lenteur construite, on calcule également la polarisation, que l'on conserve afin de pouvoir la calculer facilement lors de la construction géométrique des ondes.

Maillages adaptatifs

Dans le cas des surfaces que nous utilisons pour la simulation de contrôle non destructif, les courbures mises en jeu limitent les besoins de finesse des maillages utilisés pour obtenir une approximation précise de la surface. Cependant, dans le cas de surfaces aux courbures plus prononcées, l'utilisation d'un maillage à densité d'échantillonnage uniforme n'est pas adaptée. En effet, les zones plus courbées requièrent un échantillonnage plus fin et cet écart de finesse doit être pris en compte lors de la construction du maillage.

Comme le montrent Frey et Borouchaki (2003), il est possible de majorer l'erreur ϵ_{max} maximale commise par un maillage :

$$\epsilon_{max} \leq \frac{2}{9} L^2 M$$

avec L la longeur maximale des côtés des triangles du maillage et M un terme caractérisant la courbure maximale de la surface dans la zone de définition du maillage.

Frey et Borouchaki (2003) donnent une méthode pour estimer la valeur de M en connaissant la hessienne de la surface à approximer. Ce faisant, on peut donner un majorant pour la longueur des segments des triangles du maillage de sorte à respecter une contrainte d'erreur maximale entre les surfaces approximées et les surfaces maillées :

$$L \leq \frac{9\epsilon_{max}}{2M}$$

Puisque nous ne connaissons pas la valeur de la hessienne des surfaces de lenteur en tout point, nous proposons la méthode suivante afin de limiter l'erreur commise par l'approximation du maillage :

1. Construction d'un maillage uniforme selon la méthode présentée précédemment,
2. Calcul de la lenteur réelle au centre de chaque triangle du maillage,
3. Sélection des triangles où l'erreur est supérieure à un seuil ϵ fixé,
4. Subdivision de ces triangles.
5. Retour à l'étape 2.

De cette façon, il est possible de réduire l'erreur commise en approximant les lenteurs par un maillage triangulaire de manière adaptative. Notons cependant que l'erreur maximale commise par l'approximation dans le voisinage d'un point de la surface ne peut être connue qu'à l'aide d'une approximation locale de M.

Dans l'ensemble des configurations que nous avons testées, les maillages de surfaces de lenteur utilisés présentent une courbure suffisamment homogène pour qu'un

maillage uniforme suffise à simuler efficacement le comportement des matériaux anisotropes modélisés. Cette méthode s'applique donc en particulier aux fortes anisotropies.

3.5.3 Calcul géométrique de propagation en milieu anisotrope

La construction des maillages des surfaces de lenteur, effectuée une fois par matériau, permet d'effectuer un calcul rapide de lenteur et de direction d'énergie pour une direction de propagation donnée. D'après la définition des surfaces de lenteur, il suffit en effet, pour une direction $\vec{d}$ et un mode donnés, de déterminer l'intersection entre le maillage de la surface de lenteur associé à ce mode et le rayon d'origine O et de direction $\vec{d}$.

Pour calculer la lenteur, donnée par la distance entre l'origine et la surface, il est possible de considérer celle fournie par un outil d'intersection de rayons tels qu'*Embree* ou *OptiX*. Elle correspond, comme présenté en figure 3.21, à la distance parcourue par le rayon avant une intersection avec le maillage de la surface de lenteur, qui est une interpolation linéaire par morceaux de la surface de lenteur.

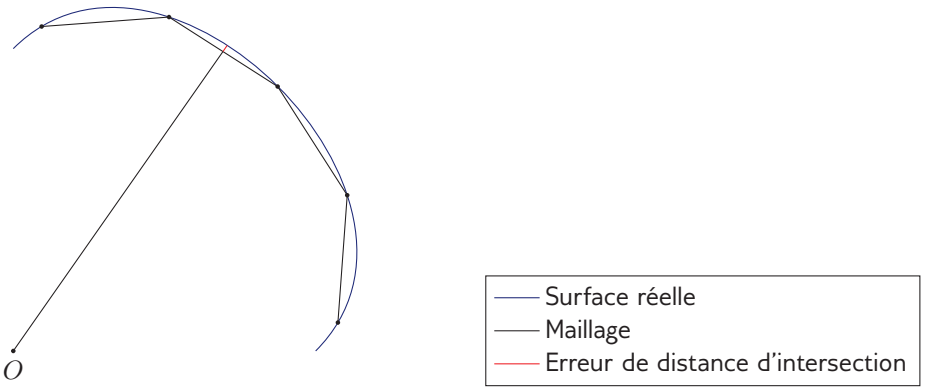


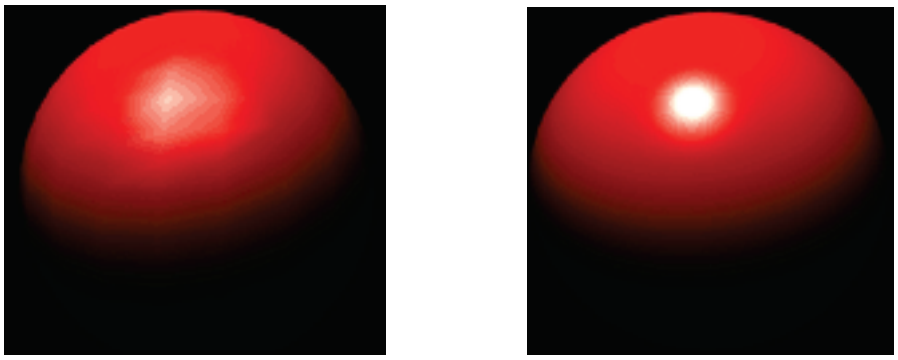
Figure 3.21 – Erreur sur la distance d'intersection commise lors d'un lancer de rayon sur un maillage

En fonction de la finesse du maillage, la valeur obtenue par cette interpolation peut être plus ou moins éloignée de la valeur réelle. Nagata détaille une méthode d'interpolation quadratique se servant des normales aux points du maillage (donc des directions d'énergie) pour fournir une approximation à un ordre plus élevé de cette valeur.

La direction d'énergie correspond à la normale à la surface de lenteur au point d'intersection. De nouveau, l'outil d'intersection peut nativement proposer une valeur pour cette normale. Celle-ci est la normale géométrique associée à la face du maillage intersectée. Elle est donc constante sur chaque face du maillage.

Le problème du calcul des normales sur un maillage triangulaire se pose aussi en rendu graphique. En effet, l'utilisation des normales a un effet direct sur la composante spéculaire du rayonnement simulé. Utiliser des normales constantes par facettes, comme dans le cas du modèle d'ombrage de Gouraud (1971) a pour effet, tout particulièrement en rayonnement spéculaire, de produire un rendu faisant apparaître les facettes du maillage (voir figure 3.22a).

L'utilisation de normales linéairement interpolées, telle qu'introduite par Phong (1975), a pour effet de reproduire la courbure du maillage et donc de fournir un



(a) Ombrage de Gouraud : normales constantes par facette

(b) Ombrage de Phong : interpolation linéaire des normales

Figure 3.22 – Comparaison des ombrages de Gouraud et de Phong, impact de l'interpolation des normales (images issues de Valm2410)

rendu plus lisse (voir la figure 3.22b). Il est également possible d'utiliser des normales interpolées quadratiquement. Van Overveld et Wyvill proposent une approche utilisant des normales interpolées de cette façon, permettant d'améliorer le réalisme du rendu dans des cas où l'interpolation linéaire nécessite un échantillonnage supérieur pour être efficace. Un exemple d'un tel cas est schématisé en figure 3.23.

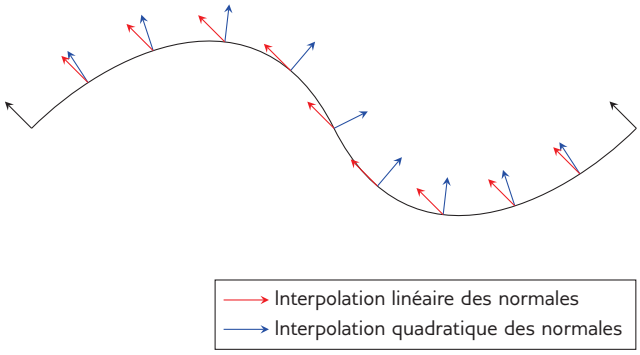


Figure 3.23 – Comparaison d'une interpolation linéaire et d'une interpolation quadratique de normales dans un cas problématique

Dans le cas des surfaces de lenteur, une interpolation linéaire des normales suffit dans la mesure où, dans un objectif de précision, les maillages de surfaces de lenteur que nous utilisons sont construits de sorte à être suffisamment fins pour éviter les situations telles que celle de la figure 3.23. On calcule donc la direction d'énergie à l'aide des coordonnées barycentriques locales au triangle intersecté :

$$\vec{d}_e = \frac{u.\vec{n}_0 + v.\vec{n}_1 + (1-u-v).\vec{n}_2}{\|u.\vec{n}_0 + v.\vec{n}_1 + (1-u-v).\vec{n}_2\|}$$

avec $\vec{n}_0$, $\vec{n}_1$ et $\vec{n}_2$ les normales respectives des sommets du triangle. Enfin, pour les calculs de polarisation, on utilise le même procédé que pour la direction d'énergie :

$$\vec{u} = \frac{u.\vec{p}_0 + v.\vec{p}_1 + (1-u-v).\vec{p}_2}{\|u.\vec{p}_0 + v.\vec{p}_1 + (1-u-v).\vec{p}_2\|}$$

Il est également possible de procéder à un calcul algébrique de la polarisation. En effet, une fois l'équation de propagation résolue et la lenteur obtenue, le calcul de la polarisation est plus direct.

3.5.4 Calcul géométrique d'interaction en milieu anisotrope

Le calcul des ondes générées lors de l'interaction d'une onde ultrasonore avec une interface entre milieux anisotropes peut également être effectué simplement en utilisant des maillages de surfaces de lenteur. En effet, en reprenant les développements théoriques effectués en section 2.3.4 ainsi que la figure 2.15, on constate que le calcul des lenteurs des ondes générées pour chaque mode de polarisation passe par la recherche de l'intersection entre l'axe vertical contenant le point $(s_{proj}, 0)$ et l'axe horizontal de l'interface (voir la figure 3.24).

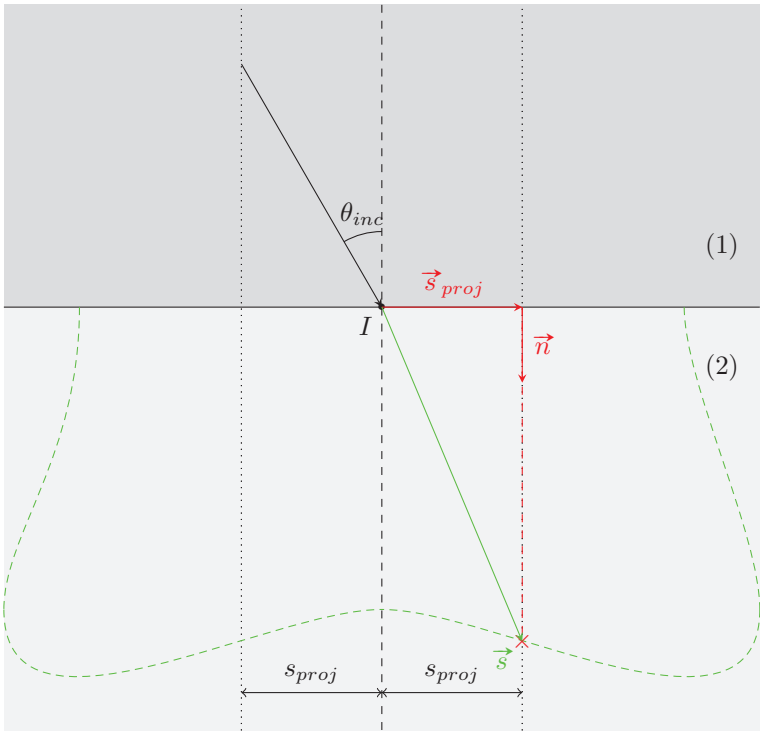


Figure 3.24 – Calcul géométrique des ondes générées à une interface

Ce processus, répété pour chacune des surfaces de lenteur du matériau d'incidence pour les réflexions et pour chacune des surfaces de lenteur du matériau opposé pour les réfractions, permet de retrouver, de la même façon qu'en propagation, la lenteur, la direction d'énergie et la polarisation de chaque mode en milieu anisotrope. Nous retrouvons, dans le cas de surfaces de lenteur concaves, la possibilité pour un mode de présenter deux solutions à l'équation de Snell-Descartes combinée à l'équation de propagation. Dans ce cas, nous considérons la solution dont la direction d'énergie est cohérente avec le côté de l'interface considéré.

3.6 Suivi de rayons ultrasonores

Nous avons présenté dans les sections précédentes l'ensemble des opérations utiles au suivi de rayons ultrasonores en milieux isotropes et anisotropes : propa-

gation en milieux homogènes, interaction avec des interfaces entre milieux et calcul d'intersection et de temps de vol.

Cette section résume le déroulement du lancer d'un rayon ultrasonore de son initialisation au terme de son suivi et détaille les informations obtenues au cours de ce processus, utilisables pour la simulation de champ ultrasonore dans le cadre du contrôle non destructif.

3.6.1 Données d'initialisation et résultats

Étant donnée une scène de contrôle, identifiée par des pièces, les géométries les délimitant et les matériaux les constituant, l'identification d'un rayon nécessite les données suivantes :

— L'origine du rayon, ou position de départ de l'onde ultrasonore simulée.
— La direction de propagation du rayon de l'onde simulée que l'on suit.
— Une suite de modes de polarisation et de modes d'interactions : un premier mode de polarisation dans le matériau de départ, puis, pour chaque interaction avec une interface, le mode généré que l'on choisit de suivre et le type d'interaction suivi.

⟨*TypeInteraction*⟩≡

```
struct {
   Mode mode;
   bool estRéfraction;
}
```

Ces données caractérisent un unique trajet physique de rayon ultrasonore que les principes exposés précédemment permettent de suivre. Le processus de suivi d'un rayon nous permet d'obtenir des informations utiles à la simulation d'ondes ultrasonores. Parmi celles-ci, nous nous intéressons en particulier aux suivantes :

— Le temps de vol total du rayon, utile à l'intégration de sa contribution à l'onde ultrasonore.
— La position d'intersection finale du rayon, qui sert notamment à déterminer, le cas échéant, l'élément de traducteur que le rayon a intersecté.
— La direction initiale du rayon, que l'on conserve pour des calculs d'angle solide initial (voir en section 4.3.1).
— La polarisation initiale de l'onde, pour connaître la direction de la contribution du rayon au déplacement particulaire.
— La lenteur initiale, dont la valeur est utilisée pour le calcul de la divergence du faisceau d'ondes (voir en section 4.3.1).
— La lenteur finale, servant à la détermination de l'étalement temporel des réponses impulsionnelles pour une méthode de calcul de pinceaux.
— Le coefficient de Fresnel cumulé sur l'ensemble des interactions aux interfaces traversées par le rayon.

⟨*RésultatRayon*⟩≡

```
struct RésultatRayon {
   float tempsVol;
   float lenteurInitiale;
   float lenteurFinale;
   Vecteur3D positionFinale;
   Vecteur3D directionInitiale;
   Vecteur3D polarisationInitiale;
   complex<float> coefficientFresnel;
};
```

3.6.2 Déroulement du lancer de rayons ultrasonore

Pour une origine, une direction de propagation et une suite de modes de polarisation données, dans un matériau donné, le processus de lancer de rayons suit les étapes suivantes :

1. Initialisation des données de propagation dans la direction donnée pour le matériau considéré et avec le premier mode de polarisation de la séquence.
2. Calcul de la première intersection avec une interface et cumul du temps de vol.
3. Si la séquence de modes de polarisation n'est pas vide, dépilage du mode de polarisation suivant et calcul de l'interaction avec l'interface en fonction du mode et dans le nouveau milieu. Celui-ci est déterminé en fonction du type d'interaction : s'il s'agit d'une réflexion, le nouveau milieu est le milieu courant. Sinon, il s'agit du milieu de l'autre côté de l'interface. Puis, retour à l'étape 2.
4. Calcul et stockage des données finales du rayon.

⟨*Suivi de rayon ultrasonore*⟩≡

```
RésultatRayon suiviRayon(Vecteur3D origine, Vecteur3D direction,
                         queue<TypeInteraction> sequenceInteractions,
                         Milieu milieuDépart) {
   RésultatRayon résultat;
   RayonUS rayon = milieuDépart.propagation(origine, direction,
RayonUS 75           sequenceInteractions.top().mode);
   sequenceInteractions.pop();

   résultat.polarisationInitiale = RayonUS.polarisation;
   résultat.directionInitiale = direction;
   résultat.lenteurInitiale = RayonUS.lenteur;

   Milieu milieuCourant = milieuDépart;

   complex<float> coefficientFresnel = 1;

   while(!sequenceInteractions.empty()) {
      TypeInteraction interaction = sequenceInteractions.top();
      sequenceModes.pop();

      Intersection intersection = intersectionEmbree(scène, rayon);

      Milieu nouveauMilieu = milieuSuivant(intersection, interaction,
        milieuCourant);
      RayonUS nouveauRayon = nouveauMilieu.interaction(rayon, interaction.mode);
      coefficientFresnel *= calculCoefficientFresnel(rayon, nouveauRayon,
        milieuCourant, nouveauMilieu);

      rayon = nouveauRayon;
   }

   résultat.tempsVol = rayon.tempsVol;
   résultat.lenteurFinale = rayon.lenteur;
   résultat.coefficientFresnel = coefficientFresnel;

   return résultat;
}
```

⟨*milieuSuivant*⟩≡

```
Milieu milieuSuivant(Intersection intersection, TypeInteraction interaction,
  Milieu milieuCourant) {
   if(!interaction.estRéfraction) {
      return milieuCourant;
   }
   else {
      if(intersection.milieu1 == milieuCourant) {
         return intersection.milieu2;
      }
      else {
         return intersection.milieu1;
      }
   }
}
```

3.7 Performances et validation

3.7.1 Lancer de rayons par paquets

Comme nous avons pu le voir en section 3.1.3, un gain de performances peut être obtenu en lancer de rayons sur CPU avec la bibliothèque *Embree* en utilisant le lancer de rayons par paquets. Puisqu'il apparaît clairement que les opérations de lancer de rayons seront appelées de très nombreuses fois au cours des simulations d'ondes ultrasonores, il semble utile d'exploiter cette fonctionnalité.

Nous choisissons donc d'implémenter, en plus de l'outil de lancer de rayons ultrasonores détaillé précédemment, une version permettant de traiter simultanément des rayons par paquets. Nous utilisons pour ce faire la bibliothèque *Vector Class Library*, permettant une manipulation portable et transparente des registres SIMD et gérant les jeux d'instructions modernes.

3.7.2 Validation

Nous avons mené des tests de validation de l'outil de lancer de rayons ultrasonores que nous avons développé afin de vérifier l'adéquation des résultats qu'il produit avec ceux fournis par l'outil de lancer de rayons ultrasonores implémenté dans *CIVA*. Ces tests ont été réalisés en comparant les valeurs de temps de vol et de positions finales de rayon lancés en milieux isotropes et anisotropes avec des trajets de longueur variable et en utilisant les calculs d'anisotropie algébrique et géométrique.

En milieux isotropes, les résultats fournis par notre outil de lancer de rayons sont pratiquement identiques à ceux produits par *CIVA* pour des rayons identiques, la seule différence étant la précision des nombres à virgule flottante utilisés. En effet, *CIVA* utilise des flottants à double précision alors que notre outil, basé sur *Embree*, se restreint à des nombres à précision simple. L'erreur observée est donc de l'ordre de l'erreur numérique, négligeable lors des calculs de champ ultrasonore. Pour le calcul algébrique en milieux anisotropes, puisque les résultats sont calculés exactement de la même manière que dans *CIVA*, le constat est le même qu'en milieux isotropes et les erreurs restent négligeables.

Enfin, dans le cas du calcul géométrique en milieux anisotropes, l'utilisation d'une approximation des surfaces de lenteur par un maillage triangulaire cause des erreurs de précision que l'on peut observer en lisant les résultats du lancer de rayons. Dans l'ensemble des configurations de champ testées, ces erreurs restent de l'ordre de l'erreur numérique et n'ont pas d'impact sur le résultat final de champ.

Pour de plus fortes anisotropies, l'utilisation de maillages adaptatifs avec contrôle préalable de la précision de l'approximation permettrait de réduire significiativement l'erreur commise par la méthode géométrique de calcul de rayons ultrasonores en milieux anisotropes.

3.7.3 Impact de l'anisotropie géométrique sur les performances

Afin d'évaluer l'apport en performances de lancer de rayons du calcul géométrique de rayons ultrasonores en milieux anisotropes, nous comparons les performances de méthodes algébriques et géométriques de lancer de rayons ultrasonores en milieux anisotropes. Ces résultats de performances sont ensuite comparés à ceux obtenus dans les mêmes conditions dans un milieu isotrope.

La pièce utilisée pour ce test de performance est une pièce plane et de nombreux rayons sont lancés dans des directions aléatoires (mais identiques d'un test à l'autre) depuis un point d'origine. Chaque rayon subit un nombre n de réflexions sur les bords de la pièce au cours du calcul, que l'on fait varier de 0 à 2. Le mode suivi est systématiquement le mode longitudinal (L en milieu isotrope et qL en milieu anisotrope).

Le maillage de surfaces de lenteur utilisé est uniforme et composé de 65 000 triangles tandis que le maillage de la pièce en comporte 12. On lance pour chaque test 1 million de rayons aléatoires en utilisant un unique cœur d'un processeur Intel Xeon E5-1650v2. Les résultats de ces mesures de performances sont présentés dans la table 3.2.

Réflexions	0	1	2
Isotrope	3.8	2.1	1.5
Anisotrope géométrique	0.51 (/7.6)	0.32 (/6.6)	0.23 (/6.3)
Anisotrope algébrique	0.089 (/43)	0.044 (/48)	0.030 (/49)

Table 3.2 – Performances (en millions de rayons par seconde) du lancer de rayons ultrasonores dans des matériaux isotropes et anisotropes, en calcul algébrique et géométrique, pour différents nombres de réflexions

On constate un rapport de performances entre les calculs anisotropes algébriques et géométriques d'environ ×8. L'utilisation de surfaces de lenteur géométriques apporte donc un net gain de performances en lancer de rayons ultrasonores. L'écart de performances entre calculs isotropes et calculs anisotropes géométriques se situe quant à lui aux alentours de 6 à 7. Ainsi, les calculs de rayons ultrasonores en milieux anisotropes restent sensiblement plus coûteux qu'en milieux isotropes, tout en étant nettement améliorés par rapport à la version algébrique.

3.8 Conclusion

Dans ce chapitre, nous avons commencé par rappeler les principes de base de la méthode de lancer de rayons et ses applications au rendu photo-réaliste. Puis, évoquant les problèmes de performances communs dans ce domaine, nous avons présenté deux solutions logicielles exploitant deux architectures différentes : le CPU et le GPU. Celles-ci offrent des outils géométriques optimisés applicables à tout type de simulation basé sur le lancer de rayons.

Nous avons ensuite établi dans quelle mesure le lancer de rayons peut s'appliquer aux simulations de champ ultrasonore avant de proposer un modèle de rayons ultrasonores permettant l'échantillonnage d'un front d'onde. Ceci nous a permis de mettre en place les bases d'une implémentation optimisée de lancer de rayons ultrasonores exploitant la bibliothèque *Embree*.

Enfin, au vu du coût calculatoire important du lancer de rayons en milieux anisotropes, nous avons proposé une solution géométrique moins coûteuse que la solution algébrique et exploitant de nouveau les performances de l'outil de lancer de rayons *Embree*. En construisant des maillages des surfaces de lenteur des matériaux anisotropes considérés, nous avons pu améliorer significativement les performances du lancer de rayons ultrasonores en milieux anisotropes. Des résultats de performance et de validation ont également été détaillés.

CHAPITRE 4

Reconstruction du front d'onde émis

Sommaire

Ce chapitre détaille la méthode utilisée pour reconstruire un front d'onde ultrasonore lancé depuis un point de champ en direction de la surface d'un traducteur de contrôle non destructif par ultrasons, en application du principe de calcul inverse expliqué en section 3.1.4.

Après avoir présenté la méthode des pinceaux introduite par Gengembre (1999), nous proposons une variation de modèle simplifiant le processus de construction des réponses impulsionnelles. Cette variation utilise uniquement le lancer de rayons ultrasonores pour déterminer la divergence du front d'onde et les caractéristiques des pinceaux.

Puis, nous détaillons une méthode permettant de réduire le nombre de rayons nécessaires à la construction du front d'onde en exploitant les propriétés de régularité et de continuité des fronts d'onde ultrasonore. Celle-ci se base sur une approximation d'ordre supérieur à l'ordre 1 de l'approximation utilisée dans le cadre de la méthode des pinceaux.

4.1 Principe général

Nous présentons dans cette section les hypothèses de base et les objectifs de l'opération de reconstruction de fronts d'onde ultrasonore dans le cadre de la simulation de contrôle non destructif. Après avoir brièvement récapitulé les éléments de contexte et de mise en situation présentés dans les chapitres précédents, nous formalisons la problématique de ce chapitre.

4.1.1 Mise en situation

Une scène de contrôle non destructif comporte usuellement les éléments suivants :

— Une **pièce** à contrôler, caractérisée par un **matériau** et une **géométrie**.
— Un **traducteur** qui émet les ondes ultrasonores du contrôle. Il peut être monoélément ou multiéléments et de forme quelconque. Nous nous intéressons en particulier au cas des traducteurs plans et utilisons un **découpage** de la surface du traducteur en échantillons.
— Un **milieu couplant** séparant le traducteur de la pièce contrôlée.
— Une **zone de champ** qui représente l'ensemble des **points de champ** ou points pour lesquels nous cherchons à calculer le champ ultrasonore.

La figure 4.1 schématise une scène de contrôle usuelle en légendant ses principaux éléments constitutifs.

4.1.2 Méthode semi-analytique

Notre objectif dans le cadre de ces simulations est d'effectuer un calcul rapide et précis des valeurs du champ ultrasonore engendré par un traducteur ultrasonore en chaque point de la zone de champ, contenue dans une pièce. L'objectif de précision que nous nous sommes fixé peut être atteint à l'aide de méthodes numériques ou de méthodes semi-analytiques.

Les méthodes numériques de type éléments finis ou différences finies ont pour intérêt d'être particulièrement précises et de modéliser efficacement l'ensemble des phénomènes liés à la propagation d'ondes ultrasonores puisque les équations de l'élasto-dynamique sont directement utilisées. Cependant, les temps de calcul induits par de telles méthodes empêchent de les utiliser pour de la simulation interactive.

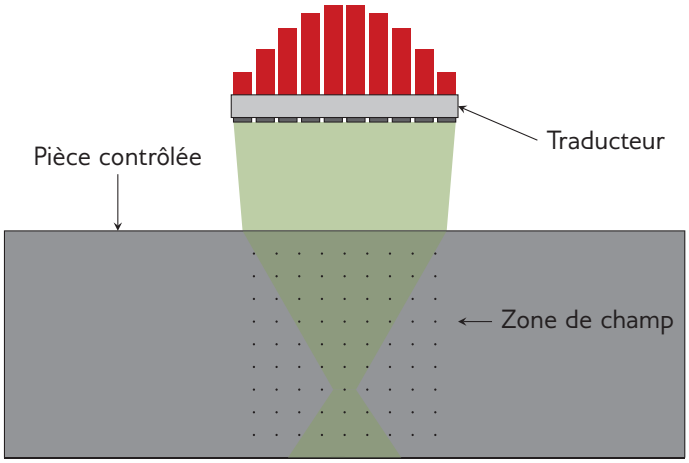


Figure 4.1 – Une scène de contrôle

Le niveau de modélisation adopté dans le cadre des méthodes semi-analytiques considère les équations de l'élasto-dynamique à un niveau plus élevé puisqu'on utilise directement une solution à ces équations, modélisée par des ondes ultrasonores. Selon l'approche choisie pour modéliser ces ondes, il demeure toutefois assez difficile de quantifier certains phénomènes avec ces approches, tels que la diffraction sur arêtes ou des dièdres, les interactions entre défauts et les ondes de surface.

Ainsi, le compromis que nous choisissons d'effectuer entre exactitude de la simulation et performances se situe au niveau de l'exhaustivité de notre modèle. En privilégiant la précision pour la modélisation des phénomènes dont l'impact est le plus élevé en contrôle non destructif, au détriment de phénomènes plus marginaux, nous pouvons arriver à un outil de simulation à la fois efficace pour le contrôle non destructif et plus rapide qu'une simulation exhaustive.

Bien entendu, pour toute approximation effectuée, il est nécessaire de connaître les limites du modèle considéré afin de ne pas l'appliquer à tort dans des conditions où les phénomènes qu'il ne modélise pas précisément prennent une importance majeure. La connaissance *a priori* des phénomènes les plus importants dans une configuration donnée de contrôle non destructif peut donc particulièrement profiter à la simulation.

Notons enfin que, en raison de la complexité des phénomènes physiques mis en jeu lors de la propagation d'ondes ultrasonores, aucun modèle purement analytique modélisant de manière suffisamment précise le comportement de la matière au passage d'une onde ultrasonore n'existe à notre connaissance.

4.2 Méthode des pinceaux

Cette section présente une méthode semi-analytique de simulation d'ondes, appelée « méthode des pinceaux ». Après avoir établi un historique des travaux autour de celle-ci, nous présentons son principe de base.

4.2.1 État de l'art

La méthode des pinceaux est une méthode généraliste de simulation d'ondes. Bien qu'elle puisse être appliquée aux ondes ultrasonores comme nous le verrons par la suite, elle a initialement été développée pour des applications de rendu graphique.

Les premières approches de lancer de pinceaux avaient pour objet de prendre en compte des relations de voisinage dans des paquets de rayons afin d'en extraire une information facilitant le processus de rendu. Heckbert et Hanrahan (1984) proposent ainsi un premier modèle de lancer de faisceaux de rayons efficace en réflexion sur des polygones. Amanatides (1984) utilise quant à lui des cônes notamment pour appliquer de l'*anti-aliasing*. Ces premières méthodes manquaient de fondements mathématiques solides et restaient cantonnées à des applications spécifiques.

Shinya et collab. (1987) donnent des fondements théoriques au problème du lancer de pinceaux et fournit une méthode générale pour tracer et suivre la divergence géométrique d'un pinceau au moyen d'une approximation linéaire, dite approximation paraxiale au voisinage du pinceau, elle-même issue de travaux précédents en électromagnétisme (voir les travaux de Deschamps (1972)). Il met également en place une analyse de l'erreur commise par cette approximation pour contrôler la précision des résultats obtenus.

Gengembre (1999) se base sur les travaux de Deschamps (1972) pour le formalisme de l'approximation paraxiale et sur des développements en lancer de rayons en élasto-dynamique (voir par exemple les travaux de Ogilvy (1985) et de Červenỳ (1972)) pour appliquer la méthode des pinceaux à la simulation d'ondes ultrasonores en milieux hétérogènes et anisotropes. Il y démontre l'équivalence de cette méthode avec la méthode de la phase stationnaire dans le cas homogène isotrope.

De cette façon, il obtient une méthode précise et moins coûteuse que du lancer de rayons pur tel que proposé par Ogilvy (1985), en particulier dans le cas tridimensionnel et pour des matériaux anisotropes et hétérogènes. Nous reprenons dans la suite les principes de cette méthode pour les adapter au calcul de champ ultrasonore interactif.

4.2.2 Principe de base

Un pinceau peut être représenté géométriquement par un cône de section quadrangulaire et d'ouverture angulaire donnée. Il est identifié par son sommet, le **point d'origine** O, sa surface de base S, un rayon ultrasonore central et un rayon ultrasonore dit paraxial, tels que représentés en figure 4.2.

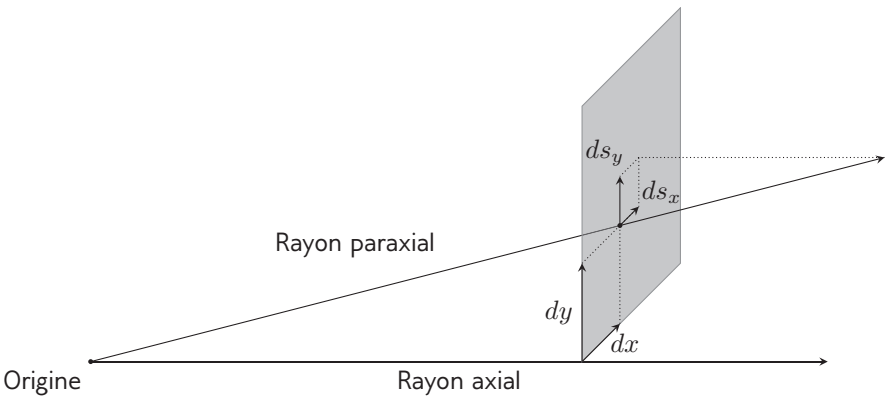


Figure 4.2 – Un pinceau et les grandeurs qui lui sont associées

Physiquement, le pinceau représente un flux d'énergie acoustique en provenance de la source d'énergie ponctuelle O et se propageant en direction de la surface S. Puisque O est la seule source d'énergie considérée dans la modélisation, la loi de

conservation de l'énergie nous indique que le flux d'énergie total traversant la surface du pinceau est nul. Ainsi, l'énergie surfacique traversant la surface S est inversement proportionnelle à son aire.

Le rayon central du pinceau suit la trajectoire de Fermat et donc les lois de l'optique géométrique appliquées aux ondes ultrasonores telles que formalisées dans le chapitre 2 et explicitées dans le chapitre 3.

Le rayon paraxial suit une direction différente de celle du rayon central, mais que l'on considère comme assez proche pour que l'approximation paraxiale soit valable (l'approximation paraxiale étant une approximation d'ordre 1, l'erreur augmente à mesure que l'on s'éloigne de l'axe). On le caractérise par quatre grandeurs différentielles exprimées dans un repère inclus dans le plan orthogonal à l'axe central du pinceau :

— dx et dy caractérisent la position du point d'intersection entre le rayon paraxial et la surface S du pinceau.
— ds_x et ds_y donnent la projection du vecteur lenteur du rayon paraxial sur la surface S.

On identifie alors un rayon paraxial par un vecteur $\vec{\Psi}$:

$$\vec{\Psi} = \begin{pmatrix} dx \\ dy \\ ds_x \\ ds_y \end{pmatrix}$$

Ceci permet de définir un rayon paraxial du pinceau par un vecteur à 4 coordonnées. On remarque que pour $\vec{\Psi} = 0$, le rayon considéré est le rayon axial et si $ds_x = ds_y = 0$, le rayon est parallèle au rayon axial.

4.2.3 Matrices de propagation

Définition

Dans le cadre de l'approximation paraxiale, l'évolution d'un vecteur de pinceau dans un milieu homogène peut être caractérisée par une matrice 4×4 dépendante du milieu de propagation. On peut de même définir des matrices 4×4 caractéristiques d'interfaces entre deux matériaux. On appelle ces matrices les **matrices de propagation** et on les note usuellement avec une notation par blocs :

$$L = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

où les matrices A, B, C et D sont des matrices de taille 2×2.

Calcul des pinceaux

Le vecteur $\vec{\Psi'}$ du pinceau issu de la propagation du pinceau de vecteur $\vec{\Psi}$ au travers du milieu ou de l'interface caractérisé par la matrice L est alors défini par :

$$\vec{\Psi'} = L\vec{\Psi}$$

Ou encore :

$$\vec{\Psi'} = \begin{pmatrix} dx' \\ dy' \\ ds'_x \\ ds'_y \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \vec{\Psi} = \begin{pmatrix} A\begin{pmatrix} dx \\ dy \end{pmatrix} + B\begin{pmatrix} ds_x \\ ds_y \end{pmatrix} \\ C\begin{pmatrix} dx \\ dy \end{pmatrix} + D\begin{pmatrix} ds_x \\ ds_y \end{pmatrix} \end{pmatrix}$$

Milieu hétérogène

En considérant un pinceau $\vec{\Psi}$ traversant une suite de milieux homogènes et d'interfaces caractérisés par les matrices de propagation $(L_i)_{1 \leq i \leq N)}$, le pinceau $\vec{\Psi'}$ résultant peut s'écrire :

$$\vec{\Psi'} = \left(\prod_{i=1}^{n} L_i \right) \vec{\Psi}$$

Puisqu'un milieu hétérogène peut être modélisé par une suite de milieux homogènes et d'interfaces les séparant, ces matrices suffisent à décrire l'évolution d'un pinceau lors de la traversée d'un milieu hétérogène.

Calcul des matrices de propagation

Au sein d'un milieu homogène, la propagation d'un pinceau se traduit par un accroissement de la surface S le long de l'axe du rayon central. Cette ouverture est caractérisée par la projection du rayon paraxial sur le plan orthogonal à l'axe du pinceau. La matrice de propagation d'un milieu homogène s'écrit donc :

$$L = \begin{pmatrix} \tilde{\tilde{1}} & \frac{r}{s}\tilde{\tilde{1}} \\ \tilde{\tilde{0}} & \tilde{\tilde{1}} \end{pmatrix}$$

avec r la distance de propagation et s la lenteur de propagation du rayon axial dans le milieu.

À une interface entre deux milieux, le comportement du pinceau peut être entièrement déterminé par l'application de la loi de Snell-Descartes. Cela se traduit par une variation de l'ouverture du pinceau au passage de l'interface. Ainsi, à une interface entre deux milieux isotropes, la matrice de propagation L s'écrit [1] :

$$L = \begin{pmatrix} \tilde{\tilde{\Theta}}_2 \tilde{\tilde{\Theta}}_1^{-1} & \tilde{\tilde{0}} \\ \tilde{\tilde{0}} & \tilde{\tilde{\Theta}}_2^{-T} \tilde{\tilde{\Theta}}_1^{T} \end{pmatrix}$$

avec

$$\tilde{\tilde{\Theta}}_i = \begin{pmatrix} cos(\theta_i) & 0 \\ 0 & 1 \end{pmatrix}$$

θ_1 et θ_2 sont respectivement l'angle formé entre le rayon incident et la normale et entre le rayon réfléchi/réfracté et la normale.

Le cas plus général de l'interaction entre deux milieux anisotropes fait également intervenir la loi de Snell-Descartes. Enfin, il est possible de tenir compte de la courbure d'une interface dans le calcul des pinceaux en connaissant sa matrice de

1. La notation M^{-T} fait référence à l'inverse de la transposée de la matrice M

courbure. Pour le détail des calculs des matrices de propagation à une interface entre milieux anisotropes et pour le cas des courbures, nous nous référons aux travaux de Gengembre (1999).

Divergence

L'utilisation des pinceaux pour le calcul de champ ultrasonore permet de simuler la propagation spatiale d'une onde ultrasonore. Afin d'obtenir l'amplitude finale du pinceau, nous devons tenir compte de sa **divergence**. En effet, la loi de conservation de l'énergie à l'intérieur du pinceau fait que le flux d'énergie diminue avec l'ouverture du pinceau, celui-ci étant, dans le cadre de l'approximation d'un pinceau à faible ouverture, réparti uniformément sur la surface de base S.

Le rapport entre l'intensité par unité d'angle solide initiale I_Ω et l'intensité surfacique finale s'écrit donc comme le rapport entre l'angle solide de départ $d\Omega$ et la surface de base finale du pinceau dS :

$$\frac{I_S}{I\Omega} = \frac{d\Omega}{dS}$$

Le vecteur pinceau final $\vec{\Psi'}$ nous permet de calculer la surface finale dS du pinceau :

$$dS = dx'dy'$$

De la même façon, le vecteur pinceau initial $\vec{\Psi}$ contient l'information d'angle solide initial, déduite de la projection initiale du vecteur lenteur sur la surface du pinceau :

$$d\Omega = \frac{ds_x ds_y}{s_i^2}$$

avec s_i la norme du vecteur lenteur initial. On déduit donc un facteur de divergence en intensité R_I :

$$R_I = \frac{1}{s_i\sqrt{det(B)}}$$

L'amplitude du déplacement particulaire au point de champ considéré égale la moitié de la valeur du vecteur de Poynting puisque l'on considère une onde hémisphérique émise par un point de la surface du traducteur ultrasonore. On obtient, comme cela est détaillé par Gengembre et collab. (2006), le coefficient de divergence en amplitude :

$$DF = \frac{1}{2\Pi\sqrt{det(B)}}$$

Facteurs d'amplitude à une interface

En plus du facteur d'amplitude lié à la divergence du pinceau au cours de sa propagation, chaque interface traversée réduit l'énergie du pinceau puisque différents modes sont générés à chaque interface, comme nous avons pu le voir en section 2.3.

La répartition de l'énergie entre les différents modes (transmis, réfléchis et évanescents pour chaque type de polarisation) est obtenue à partir des conditions de continuité aux interfaces et se traduit par le coefficient de Fresnel, noté $T_{1\to2}$, dont le calcul est détaillé en annexe D.

Pour obtenir la valeur du coefficient de transmission en amplitude, applicable directement à la valeur du déplacement et donc de l'amplitude du champ ultrasonore, le rapport entre les sections de pinceau avant et après l'interaction est à considérer (voir la figure 4.3).

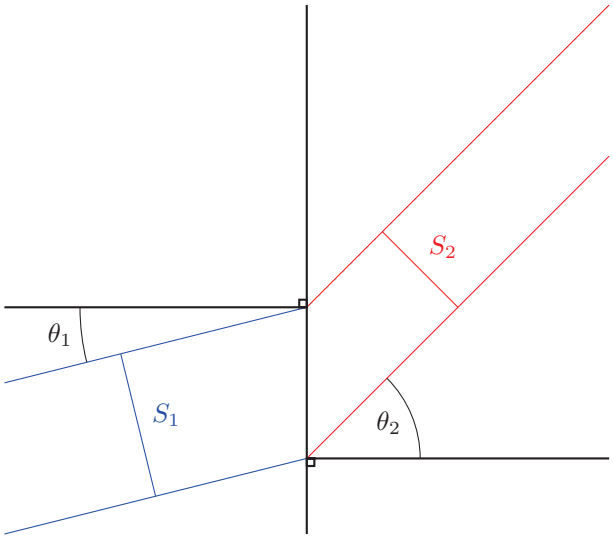


Figure 4.3 – Rapport de section des pinceaux avant et après une interface

Puisque l'énergie est proportionnelle au carré de l'amplitude du champ, le coefficient de transmission en amplitude T_A s'écrit comme le produit du coefficient de transmission en énergie et de la racine du rapport entre les sections de pinceau de part et d'autre de l'interface :

$$T_A = T_{1\to2}\sqrt{\frac{|cos(\theta_2)|}{|cos(\theta_1)|}}$$

avec θ_1 et θ_2 les angles d'incidence et de réfraction/réflexion à l'interface considérée. Finalement, à une interface, l'amplitude du pinceau A_2 après interaction est calculée à partir de l'amplitude du pinceau avant interaction A_1 comme suit :

$$A_2 = T_A A_1 = T_{1\to2}\sqrt{\frac{|cos(\theta_2)|}{|cos(\theta_1)|}}A_1$$

4.2.4 Intégration des pinceaux

Pour chaque pinceau émis depuis le point de champ considéré et dont la surface finale intersecte la surface du traducteur, nous pouvons extraire une réponse impulsionnelle élémentaire. La sommation de ces réponses élémentaires constitue la réponse impulsionnelle globale qui sert ensuite à déterminer la valeur d'amplitude du champ au cours du temps.

La réponse impulsionnelle élémentaire d'un pinceau est assimilée à un échelon temporel dont l'amplitude, l'étalement temporel et le temps central sont extraits des

caractéristiques finales du pinceau. La valeur d'amplitude correspond à une énergie. Par conséquent, elle est distribuée sur l'étalement temporel du pinceau, ce qui revient à ajouter un facteur inverse de l'étalement temporel à la valeur d'amplitude. Finalement, on a :

Amplitude $A = \frac{A_0 \times DF \times T_A \times dS}{\Delta t}$, avec A_0 l'amplitude initiale, DF le facteur de divergence, T_A le coefficient de transmission en amplitude, dS la surface du pinceau à l'arrivée et Δt son étalement temporel.

Étalement temporel Calculé comme la différence de marche maximale évaluée sur le pinceau, en fonction de l'angle d'incidence du pinceau sur le traducteur et de son étendue spatiale.

Temps central Égal au temps de vol du rayon central du pinceau.

Pour un traducteur multiéléments utilisant des lois de retard ou d'amplitude, le temps central et l'amplitude de chaque pinceau sont modifiés en cohérence avec l'élément du traducteur intersecté. Les pinceaux intersectant des éléments de traducteur différents sont découpés de sorte à n'en intersecter qu'un.

Les coefficients de Fresnel pouvant être complexes, l'amplitude du déplacement est elle-même potentiellement complexe. De plus, le déplacement étant un champ vectoriel, la réponse calculée est un signal à 3 dimensions complexes, soit 6 dimensions réelles.

4.2.5 Calcul du signal final et extraction de l'amplitude maximale

Une fois les réponses impulsionnelles élémentaires calculées pour un point de champ donné, nous les sommons sur une réponse impulsionnelle unique. Celle-ci correspond au champ rayonné par le traducteur vers le point de champ lorsque chaque élément du traducteur émet une impulsion de Dirac temporelle.

Par conséquent, étant donné un signal d'entrée S_0 auquel est soumis le traducteur ultrasonore, le champ ultrasonore en un point de champ s'obtient par convolution des signaux complexes de réponse impulsionnelle avec le signal d'entrée. Pour chaque composante x, y et z du déplacement, la partie réelle est convoluée avec le signal de référence tandis que la partie complexe est convoluée avec la transformée de Hilbert du signal de référence. Le calcul de chaque composante scalaire s_i du champ s'écrit donc, en notant $RI_i(t)$ la réponse impulsionnelle :

$$s_i(t) = RI_i(t) * S_0(t)$$

Ou encore, avec $\mathscr{F}$ et $\mathscr{H}$ les symboles des transformées de Fourier et de Hilbert :

$$s_i(t) = \mathscr{F}^{-1}(\mathscr{F}(Re(RI_i(t))) \times \mathscr{F}(S_0(t)) + \mathscr{H}(Im(RI_i(t)) \times \mathscr{H}(S_0(t))))$$

On calcule ensuite le module du signal :

$$module(t) = \sqrt{s_x(t)^2 + s_y(t)^2 + s_z(t)^2}$$

On en extrait l'amplitude maximale A_{max} sur l'ensemble de son domaine de définition :

$$A_{max} = \max_t(module(t))$$

Ce qui nous permet de construire l'image de champ, en calculant pour chaque point de la zone de champ l'amplitude maximale.

4.3 Modèle des pinceaux simplifié

Si les pinceaux lancés couvrent de manière exhaustive l'ensemble des directions de l'espace porteuses d'une énergie acoustique non nulle (allant donc en direction du traducteur), le lancer de pinceaux permet de reconstituer fidèlement la réponse d'un milieu de propagation à une impulsion ultrasonore sous l'hypothèse de pinceaux de faible étendue spatiale.

Cependant, l'approximation paraxiale utilisée par le modèle des pinceaux dispose d'un domaine de validité restreint autour du rayon axial. Ainsi, lorsque l'étendue spatiale du traducteur ultrasonore est importante, l'obtention d'un signal ultrasonore correct requiert un grand nombre de pinceaux et donc le calcul d'un nombre important de rayons.

De plus, même si le calcul des matrices de propagation en milieux isotropes ne présente pas de complexité calculatoire particulièrement importante, la généralisation du processus aux milieux anisotropes et aux interfaces courbes peut s'avérer coûteuse pour un nombre de rayons élevé.

Pour ces raisons et afin de réduire autant que possible les besoins en calcul de la phase de reconstitution des réponses impulsionnelles, nous proposons dans cette section une simplification du modèle de pinceaux basée sur les outils de lancer de rayons performants dont nous disposons.

4.3.1 Calcul de pinceaux par lancer de rayons

Idée

Le modèle paraxial de pinceaux se base sur les matrices de propagation pour proposer une approximation linéaire des rayons paraxiaux autour du rayon central d'un pinceau. Une approximation des rayons inclus dans le pinceau peut également être obtenue à l'aide d'une interpolation. Pour ce faire, nous proposons de simuler la propagation des quatre rayons r_0, r_1, r_2 et r_3 délimitant le pinceau en plus de propager le rayon central r_c, comme cela est présenté en figure 4.4.

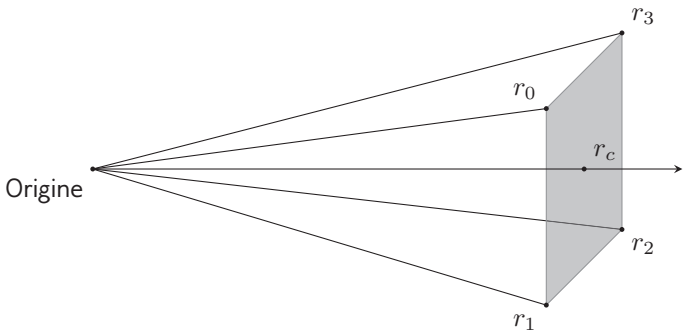


Figure 4.4 – Pinceau délimité par des rayons

Cette approche permet de s'affranchir du calcul des matrices de propagation le long du trajet du pinceau tout en rendant accessible le calcul des grandeurs nécessaires à l'intégration de la réponse impulsionnelle élémentaire associée au pinceau, comme nous allons le voir. De la même façon que pour un pinceau usuel, la réponse impulsionnelle élémentaire est approximée par un créneau dont l'amplitude et la fenêtre de temps dépendent des caractéristiques de rayons.

Calcul des temps

La fenêtre temporelle du créneau modélisant le pinceau peut être déterminée de sorte à encadrer l'ensemble des temps d'arrivée des rayons composant le pinceau. Ainsi, les bornes t_{min} et t_{max} du créneau représentant la réponse du pinceau sont les suivantes :

$$\begin{cases} t_{min} = \min_i(t_i) \\ t_{max} = \max_i(t_i) \end{cases}$$

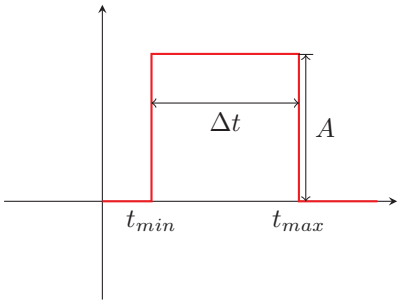


Figure 4.5 – Créneau temporel utilisé pour intégrer un pinceau

Calcul du coefficient de divergence

En utilisant le modèle de pinceau à quatre rayons, la surface finale du pinceau est l'aire délimitée par les points d'arrivée des quatre rayons. L'angle solide initial est quant à lui défini par la surface du pinceau de rayons à une distance unitaire. On peut alors calculer le coefficient de divergence à partir de ces grandeurs et de la lenteur initiale s_0 du rayon axial :

$$DF = \frac{s_0}{2\pi\sqrt{\frac{dS}{d\Omega}}}$$

De cette façon, en considérant le coefficient de transmission en amplitude cumulé T_A du rayon central et l'amplitude initiale A_0, on peut calculer de la même façon que précédemment l'amplitude du déplacement au point de champ considéré en la répartissant sur l'étalement temporel du pinceau :

$$A = \frac{A_0 \times DF \times T_A \times dS}{t_{max} - t_{min}}$$

Intérêt de l'approche

Ce modèle simplifié de pinceau permet de reproduire une réponse impulsionnelle élémentaire de pinceau sans avoir à calculer de matrices paraxiales, supprimant la nécessité de calculs matriciels coûteux (particulièrement en milieux anisotropes) et les remplaçant par des opérations de lancer de rayons ultrasonores.

Pour un pinceau isolé, cette approche nécessite un total de 5 rayons ultrasonores pour produire une réponse impulsionnelle élémentaire. Cependant, dans la mesure

où l'on cherche à couvrir la sphère unité de pinceaux pour produire une réponse impulsionnelle exhaustive, l'exploitation de propriétés de voisinage entre rayons nous permet de mutualiser des rayons et donc de réduire le nombre total de rayons à environ 2 rayons par pinceau pour un échantillonnage uniforme de la sphère unité.

Enfin, la possibilité de calculer les réponses impulsionnelles élémentaires des pinceaux à partir de rayons seuls ramène le problème du calcul de champ à un problème de reconstruction de fonctions ce qui, comme nous le verrons par la suite, permet des optimisations efficaces en termes de lancer de rayons.

4.3.2 Un problème de reconstruction de fonctions

Effectuer un calcul de champ ultrasonore par la méthode des pinceaux revient à découper les portions de la sphère unité contribuant au champ ultrasonore autour d'un point en approximations linéaires du temps de vol et de l'amplitude des ondes ultrasonores. Le modèle de pinceaux simplifié que nous avons choisi d'utiliser permet de construire une approximation similaire à partir d'un échantillonnage des temps de vol et amplitudes des rayons ultrasonores.

À partir d'un point d'origine O, pour un rayon R de direction de propagation initiale portée par les coordonnées sphériques (θ, φ), considérons :

— $A(\theta, \varphi)$ l'amplitude du rayon R tenant compte des atténuations aux interfaces, de la divergence du front d'onde ultrasonore autour de R et de l'amplitude initiale.
— $\vec{p}(\theta, \varphi)$ la polarisation de l'onde modélisée par R.
— $t(\theta, \varphi)$ le temps de propagation de l'onde portée par R.

Le calcul de la réponse impulsionnelle en O peut se résumer au calcul de l'intégrale suivante :

$$RI(t) = \iint A(\theta, \varphi)\delta(t, t(\theta, \varphi))dS.\vec{p}(\theta, \varphi) \tag{4.1}$$

où δ est le symbole de Kronecker. Par conséquent, le calcul de la réponse impulsionnelle $RI(t)$ pour chaque point de champ peut être grandement simplifié par l'utilisation de fonctions approchant l'amplitude, la polarisation et le temps de vol de l'onde ultrasonore simulée.

Le domaine d'intégration peut être réduit sans risque d'erreur à la zone de visibilité du traducteur puisque les autres zones présentent une amplitude nulle. Nous chercherons donc dans la suite à construire des fonctions approchant les amplitudes, temps de vol et polarisations sur la zone de visibilité du traducteur pour chaque point de champ.

4.4 Approximation des rayons

Cette section a pour objectif de proposer des méthodes d'approximation des grandeurs nécessaires à la construction d'une réponse impulsionnelle, obtenues par lancer de rayons ultrasonores. Ces méthodes doivent permettre une reconstitution fidèle de la réponse impulsionnelle d'un pinceau en minimisant autant que possible le nombre de points d'échantillonnage requis. Nous comparons plusieurs approches possibles et présentons leurs avantages et inconvénients avant de détailler celles que nous avons retenues.

4.4.1 Interpolation linéaire

Construction des sous-pinceaux

Une première approche pour construire une approximation des fonctions d'amplitude, de temps de vol et de polarisation peut consister à effectuer une interpolation linéaire de ces grandeurs pour séparer les pinceaux en sous-pinceaux.

Ainsi, chaque pinceau tracé selon la méthode décrite en section 4.3.1 fait l'objet d'un ré-échantillonnage régulier à l'aide d'une interpolation linéaire. On construit par interpolation des sous-pinceaux, délimités par quatre rayons interpolés et dont les réponses sont calculées comme suit :

— Le coefficient de divergence DF est calculé globalement sur le pinceau, sa valeur est donc la même pour chaque sous-échantillon du pinceau.
— Les positions des points d'arrivée des rayons sur le capteur sont interpolées et servent à calculer la surface dS couverte par chacun des sous-pinceaux.
— Le coefficient de transmission T_A est interpolé linéairement à l'aide des valeurs associées à chacun des rayons du pinceau principal.
— Pour chaque rayon interpolé du sous-pinceau, un temps de vol est calculé par interpolation linéaire. On extrait de ces quatre temps de vol un temps de vol maximal et un temps de vol minimal qui servent à déterminer l'étalement temporel du sous-pinceau.
— Enfin, la polarisation est également interpolée entre les polarisations obtenues pour les quatre rayons délimitant le pinceau.

Pour un pinceau constitué de rayons $(R_i)_{1\leq i\leq 4}$ comme présenté en figure 4.6, on note respectivement $(T_i)_{1\leq i\leq 4}$, $(V_i)_{1\leq i\leq 4}$ et $(\vec{P}_i)_{1\leq i\leq 4}$ les temps de vol, positions et polarisations des rayons $(R_i)_{1\leq i\leq 4}$.

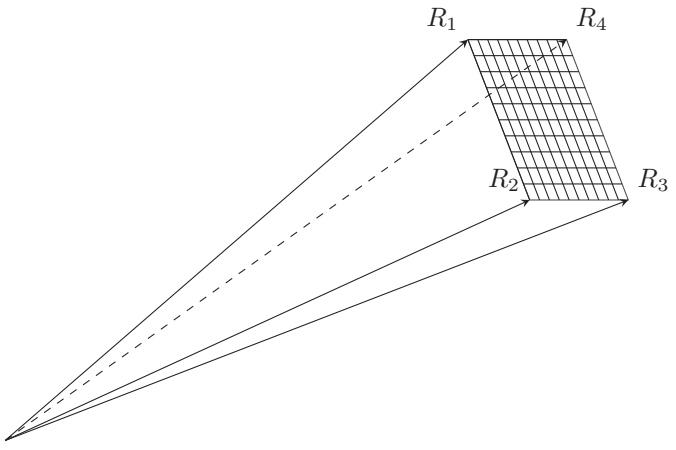


Figure 4.6 – Subdivision d'un pinceau par interpolation

On construit des fonctions interpolantes locales pour les temps de vol (notée $t(x, y)$, les positions d'arrivée (notée $v(x, y)$) et les polarisations (notée $\vec{p}(x, y)$). Celles-ci prennent une paire de coordonnées comprises entre 0 et 1, sont linéaires et prennent les valeurs des rayons du pinceau aux bornes :

$$\begin{cases} t(0,0) = T_0, v(0,0) = v_0, \vec{p}(0,0) = \vec{P_0} \\ t(0,1) = T_1, v(0,1) = v_1, \vec{p}(0,1) = \vec{P_1} \\ t(1,1) = T_2, v(1,1) = v_2, \vec{p}(1,1) = \vec{P_2} \\ t(1,0) = T_3, v(1,0) = v_3, \vec{p}(1,0) = \vec{P_3} \end{cases}$$

On a donc :

$$\begin{cases} t(x,y) = xy.T_0 + x(1-y).T_1 + (1-x)(1-y).T_2 + (1-x)y.T_3 \\ v(x,y) = xy.v_0 + x(1-y).v_1 + (1-x)(1-y).v_2 + (1-x)y.v_3 \\ \vec{p}(x,y) = xy.\vec{P_0} + x(1-y).\vec{P_1} + (1-x)(1-y).\vec{P_2} + (1-x)y.\vec{P_3} \end{cases}$$

Cette interpolation linéaire d'un pinceau permet alors d'appliquer le processus d'intégration de pinceau détaillé en section 4.3.1 en utilisant les grandeurs interpolées sur un découpage régulier du pinceau principal.

Limites

L'interpolation linéaire d'un pinceau permet de découper la surface émettrice du traducteur couverte par le pinceau en sous-pinceaux afin de simuler son balayage par le front d'onde virtuel émis par le point de champ. Elle restreint cependant l'évolution des temps de vol à une fonction linéaire sur le domaine couvert par le pinceau.

Or, le temps de vol est la variable d'intégration lors de la phase de construction de la réponse impulsionnelle. Tout l'enjeu de cette phase consiste donc à retrouver avec précision l'évolution du temps de vol sur l'angle solide de visibilité du traducteur. En effet, en un point de champ donné, la façon dont l'énergie d'un pinceau est répartie en fonction du temps, ou la façon de construire les réponses impulsionnelles élémentaires, détermine l'amplitude maximale de la réponse et a donc un impact notable sur l'image de champ.

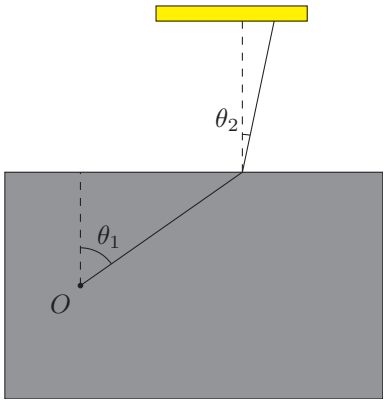


Figure 4.7 – Configuration de contrôle basique en deux dimensions

Considérons la configuration de contrôle en deux dimensions présentée en figure 4.7. Afin de mesurer l'efficacité de la méthode d'interpolation linéaire des positions et temps de vol, nous construisons dans un premier temps la réponse impulsionnelle associée à la propagation d'une onde virtuelle émise depuis le point source O vers le traducteur. Pour ce faire, nous utilisons la méthode des pinceaux avec un échantillonnage fin. La réponse impulsionnelle obtenue est présentée en figure 4.8.

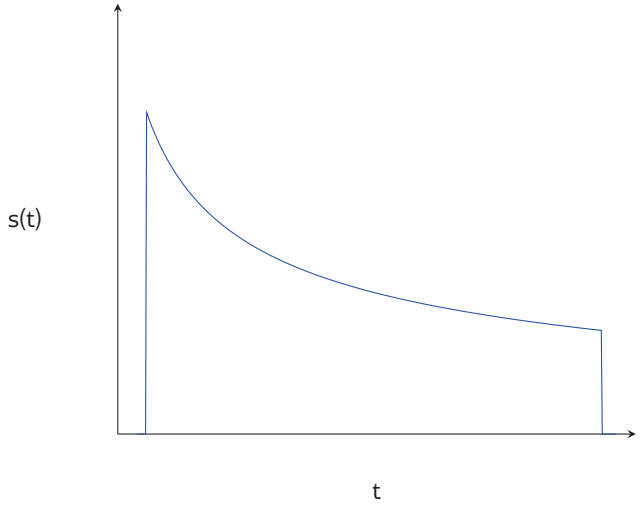


Figure 4.8 – Réponse impulsionnelle exacte du traducteur sur le point

Dans cette situation, l'utilisation de l'interpolation linéaire des pinceaux n'apporte aucune précision supplémentaire en termes d'information spatiale ou temporelle. En effet, dans le cas de notre exemple en deux dimensions, l'équation 4.1 devient :

$$RI(t) = \int A(\theta)\delta(t, t(\theta))dx\vec{p}(\theta)$$

En considérant une amplitude A_0 constante sur l'ensemble de la surface d'intégration et en ignorant le terme de polarisation pour ne considérer que l'effet de l'interpolation des temps de vol et positions d'arrivée, on a :

$$RI(t) = \int \frac{A_0}{dt}\delta(t, t(\theta))dx$$

Lorsque les fonctions de temps de vol et de position d'arrivée sont linéaires, on peut écrire t et x sous la forme de fonctions affines de θ :

$$\begin{cases} t(\theta) = t_0 + \lambda\theta \\ x(\theta) = x_0 + \mu\theta \end{cases}$$

Par conséquent, la variation de x en fonction de t est constante :

$$\frac{dx}{dt} = \frac{\mu}{\lambda}$$

Finalement, on peut écrire la réponse impulsionnelle comme suit :

$$RI(t) = \begin{cases} A_0\dfrac{\mu}{\lambda} \text{ si } t_{min} \leq t \leq t_{max}. \\ 0 \text{ sinon.} \end{cases}$$

Ce qui correspond à un créneau temporel, identique à celui obtenu par la méthode des pinceaux. L'interpolation linéaire ne peut donc, dans cette configuration, apporter d'informations supplémentaires sur l'étalement temporel de la réponse impulsionnelle élémentaire d'un pinceau. Les figures 4.9 et 4.10 montrent l'évolution

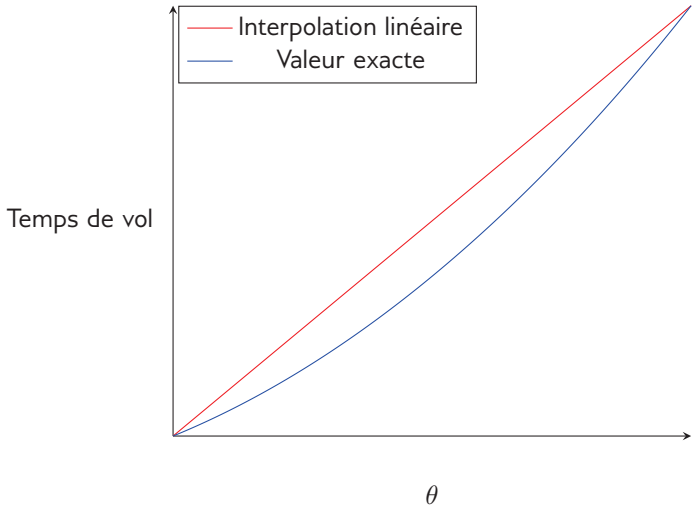


Figure 4.9 – Comparaison des temps de vol exacts et interpolés linéairement

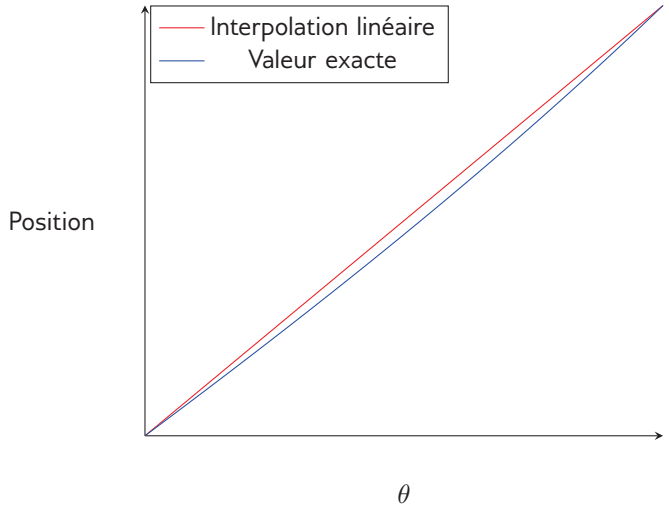


Figure 4.10 – Comparaison des positions exactes et interpolées linéairement

des temps de vol et positions finales des rayons en fonction de l'angle de départ θ ainsi que leur interpolation linéaire.

Il apparaît donc clairement que l'interpolation linéaire échoue à modéliser efficacement l'évolution des grandeurs de rayons utiles à la construction de la réponse impulsionnelle élémentaire d'un pinceau.

4.4.2 Interpolation polynomiale de Lagrange

Pour diminuer l'erreur commise par la méthode d'interpolation linéaire lors du ré-échantillonnage des pinceaux, il est possible d'utiliser des polynômes de degrés supérieurs, approximant plus efficacement les fonctions de temps de vol et de position finale des rayons. En reprenant la configuration précédente, telle que présentée en figure 4.7, nous cherchons ainsi à améliorer le processus d'interpolation des temps de vol et positions.

Formalisation du problème

Étant donnée une fonction $f : \mathbf{R} \to \mathbf{R}$, construire un interpolateur revient à définir une seconde fonction $g : \mathbf{R} \to \mathbf{R}$ telle qu'il existe un ensemble $(x_i)_{1 \leq i \leq n}$ tel que :

$$\forall i \in [\![1, n]\!], f(x_i) = g(x_i)$$

Si l'interpolateur est un polynôme, la taille n de l'échantillon de valeurs $(x_i)_{1 \leq i \leq n}$ sur lequel la condition d'égalité est vérifiée quelle que soit la fonction f interpolée est majorée par $d + 1$, avec d le degré du polynôme.

Ainsi, de manière générale, lorsque l'on cherche à interpoler une fonction en n échantillons au moyen d'un polynôme, l'utilisation d'un polynôme de degré $n-1$ ou supérieur garantit l'existence de l'interpolateur. On peut par ailleurs prouver l'unicité du polynôme interpolateur dans le cas où $d = n - 1$.

Construction de polynômes interpolateurs

Construire un polynôme interpolant une fonction $f : \mathbf{R} \to \mathbf{R}$ sur un ensemble de valeurs $(x_i)_{1 \leq i \leq n}$ revient à trouver un polynôme $P \in \mathbf{R}_{n-1}[X]$ tel que :

$$\begin{cases} P(x_1) = f(x_1) \\ \vdots \\ P(X_n) = f(x_n) \end{cases}$$

La solution à ce système d'équations est donnée par les **polynômes interpolateurs de Lagrange**, que l'on peut écrire sous la forme :

$$L(X) = \sum_{i=0}^{n} f(x_i) L_i(X)$$

avec

$$\forall i \in [\![1, n]\!], L_i(X) = \prod_{j=1}^{n} \frac{X - x_j}{x_i - x_j}$$

En effet, on montre de manière immédiate que :

$$\forall i \in [\![1, n]\!], \begin{cases} L_i(x_i) = 1 \\ \forall j \in [\![1, n]\!] \setminus \{i\}, L_i(x_j) = 0 \end{cases}$$

Ce qui prouve que le polynôme L ainsi construit interpole effectivement la fonction f aux points d'échantillonnage $(x_i)_{1 \leq i \leq n}$.

Erreur d'interpolation

Dans le cas d'une fonction f dérivable n fois, Atkinson (1980) (théorème **3.2**, page 134) montre que sur l'ensemble de l'intervalle réel I contenant les échantillons $(x_i)_{1 \leq i \leq n}$, il est possible de caractériser l'erreur d'interpolation :

$$\forall t \in I, \exists \xi \in I, f(t) - L(t) = \prod_{i=1}^{n} \frac{t - x_i}{i} f^{(n)}(\xi)$$

Par conséquent, on peut majorer l'erreur d'interpolation :

$$\forall t \in I, |f(t) - L(t)| \leq \prod_{i=1}^{n} \frac{t - x_i}{i} \max_{t \in I} f^{(n)}(t)$$

Ainsi, l'interpolation polynomiale est particulièrement efficace pour des fonctions dont les dérivées d'ordres élevés prennent des valeurs faibles. Notons cependant que l'utilisation de polynômes de degrés élevés, bien qu'elle assure une erreur nulle au niveau des points d'échantillonnage, peut provoquer une erreur importante en dehors de ces points.

Plus précisément, l'utilisation d'un nombre croissant de points d'échantillonnage uniformément répartis sur l'intervalle d'interpolation entraîne généralement une divergence vers $+\infty$ de l'erreur d'interpolation en dehors des échantillons. Ce phénomène, appelé **phénomène de Runge**, peut être évité :

— En limitant le degré des polynômes interpolateurs, par exemple en découpant l'intervalle d'interpolation (voir section 4.4.3).
— En utilisant les points d'échantillonnage dits de Tchebychev (voir Tchebychev (1899)).

Application au calcul de réponse impulsionnelle en une dimension

Nous reprenons la configuration présentée en figure 4.7 en utilisant des interpolations polynomiales de degrés croissants afin d'en comparer l'efficacité pour la construction des réponses impulsionnelles. À partir d'ensembles de respectivement 2, 3, 4 et 5 échantillons des valeurs de temps de vol et positions d'arrivée des rayons, nous construisons des interpolateurs linéaire, quadratique, cubique et quartique sur la totalité de l'angle solide couvrant le traducteur.

Ces interpolateurs nous permettent de diviser le pinceau principal en un nombre fixe de sous-pinceaux interpolés. Leurs caractéristiques sont calculées, comme précédemment, à partir des valeurs de temps de vol et de position interpolées telles que représentées en figures 4.11 et 4.12.

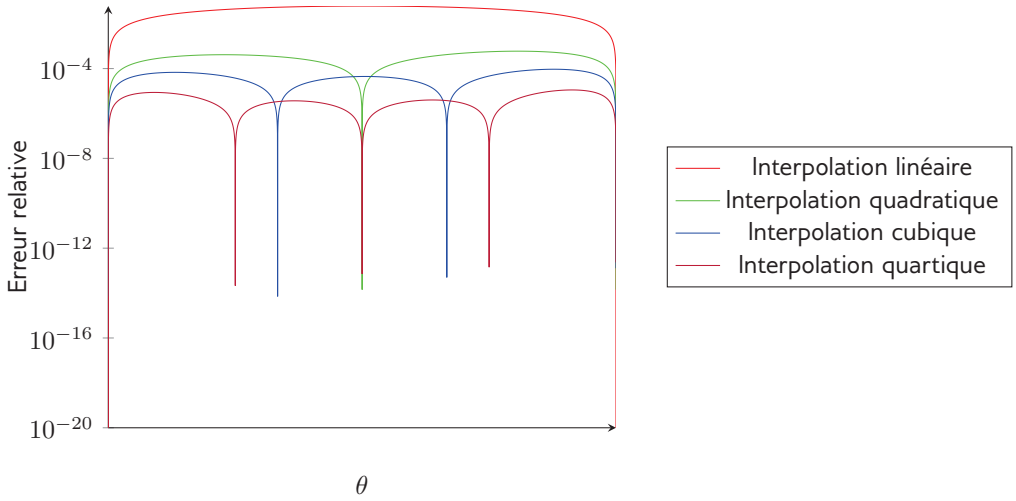


Figure 4.11 – Comparaison des temps de vol exacts et interpolés par des polynômes

Dans cette situation, l'augmentation du degré du polynôme interpolateur résulte en une approximation plus efficace des valeurs non interpolées, réduisant comme

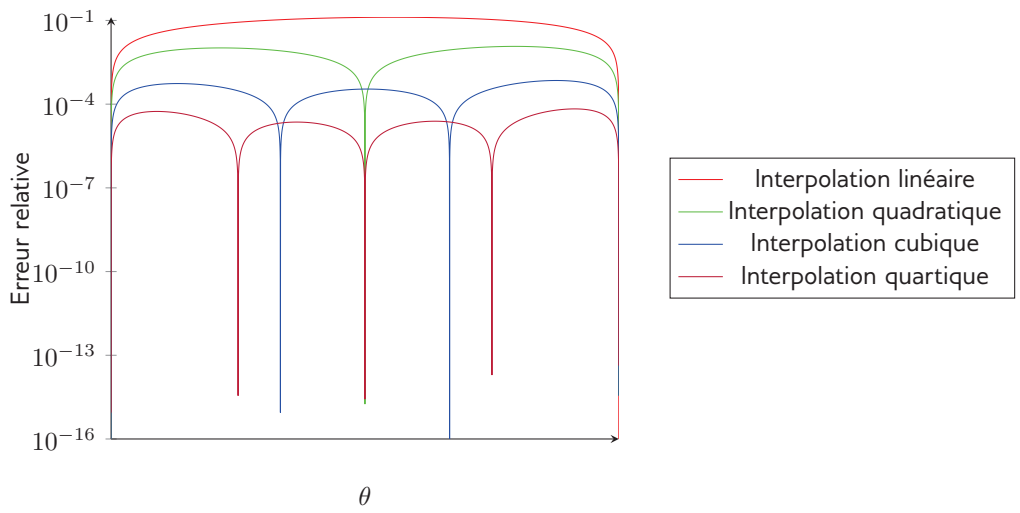


Figure 4.12 – Comparaison des positions exactes et interpolées linéairement

prévu l'erreur maximale commise sur la plage interpolée.

En intégrant ces sous-pinceaux, nous obtenons des réponses impulsionnelles différentes que nous comparons à celle obtenue en effectuant un échantillonnage de même finesse de l'angle solide couvrant le traducteur, cette fois-ci sans interpolation. Celles-ci sont présentées en figure 4.13.

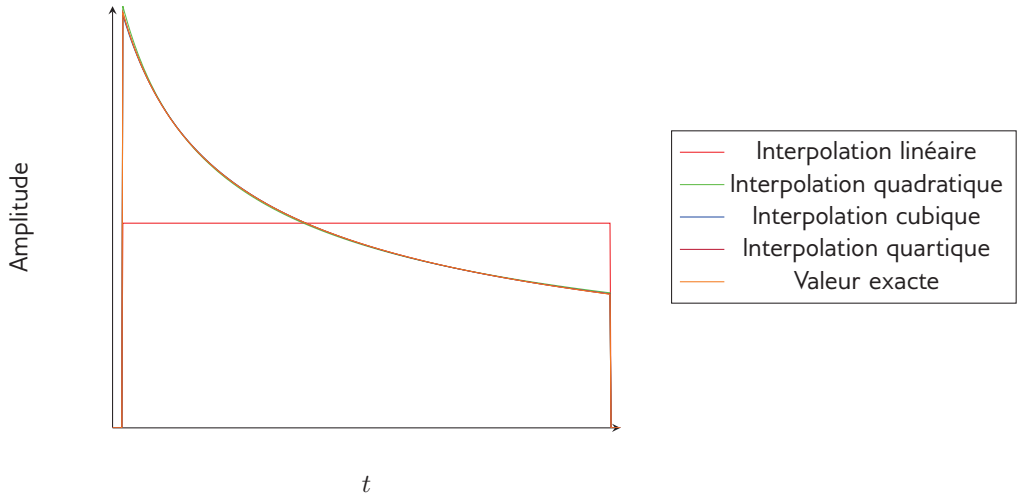


Figure 4.13 – Réponses impulsionnelles produites à l'aide des interpolateurs polynomiaux, comparées avec la réponse impulsionnelle exacte

On constate que l'utilisation d'une interpolation quadratique améliore très nettement la construction de la réponse impulsionnelle : la forme générale est préservée, ce qui n'est pas le cas en interpolation linéaire. Les résultats obtenus avec des degrés d'interpolation plus élevés approchent mieux la réponse en termes d'amplitudes.

Extension aux fonctions à deux variables

Pour pouvoir utiliser les interpolateurs polynomiaux de degrés supérieurs dans le cadre du découpage de pinceaux, la méthode d'interpolation utilisée doit également

s'appliquer aux fonctions $\mathbf{R}^2 \to \mathbf{R}$. Considérons une fonction $f : \mathbf{R}^2 \to \mathbf{R}$ et un ensemble de paires de coordonnées réelles $(x_i, y_j)_{1\leq i\leq n, 1\leq j\leq m} \in \mathbf{R}^n \times \mathbf{R}^m$. Étant donnés les polynômes $(L_{ij})_{1\leq i\leq n, 1\leq j\leq m}$ définis tels que suit :

$$\forall i \in [\![1,n]\!], L_{ij}(X,Y) = \prod_{k=1}^{n} \frac{Y - x_k}{x_i - x_k} \prod_{l=1}^{m} \frac{X - y_l}{y_j - y_l}$$

De même que précédemment, on a :

$$\forall (i,j) \in [\![1,n]\!] \times [\![1,m]\!], \begin{cases} L_{i,j}(x_i, y_j) = 1 \\ \forall (k,l) \in [\![1,n]\!] \times [\![1,m]\!] \setminus \{i,j\}, L_{ij}(x_k, y_l) = 0 \end{cases}$$

Ce qui nous permet de construire un polynôme interpolateur de Lagrange appliqué sur l'espace réel à deux dimensions :

$$L(X,Y) = \sum_{i=0}^{n} \sum_{j=0}^{m} f(x_i, y_j) L_i(X)$$

Interpolation biquadratique sur un pinceau

En pratique, pour diviser un pinceau en sous-pinceaux par interpolation, nous nous munissons d'un repère à deux dimensions local au pinceau. Il est construit de façon à ce que les coordonnées $(0,0)$, $(0,1)$, $(1,0)$ et $(1,1)$ représentent ses quatre coins. Les coordonnées des points d'échantillonnage pour l'interpolation sont fixées sur une grille régulière dont la dimension dépend du degré du polynôme.

Ainsi, pour une interpolation biquadratique, la grille d'interpolation comporte $3 \times 3 = 9$ éléments. De manière générale, on utilise la grille de coordonnées suivante, avec d le degré du polynôme interpolateur :

$$\begin{bmatrix} (0,0) & (1/d,0) & \cdots & (0,1) \\ (0,1/d) & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ (1,0) & \cdots & \cdots & (1,1) \end{bmatrix}$$

Dans le cas de l'interpolation biquadratique, la grille d'interpolation est la suivante :

$$\begin{bmatrix} (0,0) & (0,1/2) & (0,1) \\ (1/2,0) & (1/2,1/2) & (1/2,1) \\ (1,0) & (1,1/2) & (1,1) \end{bmatrix}$$

En notant $P(X,Y) = \sum_{i=0}^{2} \sum_{j=0}^{2} a_{ij} X^i Y^j$ le polynôme interpolateur, les conditions

d'interpolation s'écrivent sous la forme du système matriciel :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1/2 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1/2 & 0 & 0 & 1/4 & 0 & 0 \\ 1 & 1/2 & 1/4 & 1/2 & 1/4 & 1/8 & 1/4 & 1/8 & 1/16 \\ 1 & 1 & 1 & 1/2 & 1/2 & 1/2 & 1/4 & 1/4 & 1/4 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1/2 & 1/4 & 1 & 1/2 & 1/4 & 1 & 1/2 & 1/4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{20} \\ a_{21} \\ a_{22} \end{pmatrix} = \begin{pmatrix} f(0,0) \\ f(0,1/2) \\ f(0,1) \\ f(1/2,0) \\ f(1/2,1/2) \\ f(1/2,1) \\ f(1,0) \\ f(1,1/2) \\ f(1,1) \end{pmatrix}$$

En inversant la matrice 9×9 du système, on obtient les valeurs des coefficients du polynôme en fonction des valeurs prises par la fonction f aux points d'échantillonnage :

$$\begin{pmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{20} \\ a_{21} \\ a_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & -0 & 0 & 0 & 0 & 0 \\ -3 & 4 & -1 & 0 & -0 & 0 & 0 & 0 & 0 \\ 2 & -4 & 2 & 0 & -0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 4 & -0 & 0 & -1 & 0 & 0 \\ 9 & -12 & 3 & -12 & 16 & -4 & 3 & -4 & 1 \\ -6 & 12 & -6 & 8 & -16 & 8 & -2 & 4 & -2 \\ 2 & 0 & 0 & -4 & -0 & 0 & 2 & 0 & 0 \\ -6 & 8 & -2 & 12 & -16 & 4 & -6 & 8 & -2 \\ 4 & -8 & 4 & -8 & 16 & -8 & 4 & -8 & 4 \end{pmatrix} \begin{pmatrix} f(0,0) \\ f(0,1/2) \\ f(0,1) \\ f(1/2,0) \\ f(1/2,1/2) \\ f(1/2,1) \\ f(1,0) \\ f(1,1/2) \\ f(1,1) \end{pmatrix}$$

De cette façon, on peut déduire les coefficients du polynôme interpolateur d'un simple produit matriciel.

Enfin, une fois l'interpolateur construit pour chacune des fonctions de temps de vol et de position finale des rayons, il suffit d'évaluer chacun des polynômes interpolateurs pour chaque coordonnée locale du pinceau considérée. Afin d'accélérer l'opération d'évaluation du polynôme, la méthode de Ruffini-Horner est appliquée :

$$\begin{aligned} P(x,y) = & a_{00} + x.(a_{10} + x.a_{20}) + \\ & y.(a_{01} + x.(a_{11} + x.a_{21}) + \\ & y.(a_{02} + x.(a_{12} + x.a_{22}))) \end{aligned}$$

De cette façon, on peut décomposer chaque pinceau tracé en sous-pinceaux à condition d'effectuer le tracé de 5 rayons correspondant aux points d'échantillonnage supplémentaires requis pour la construction d'un interpolateur biquadratique (voir la figure 4.14).

4.4.3 Interpolation polynomiale par morceaux

Un problème de continuité

Nous avons montré comment, à partir d'un pinceau composé de rayons ultrasonores, il est possible d'approcher les fonctions de temps de vol et de position à moindre coût pour construire la réponse impulsionnelle élémentaire d'un pinceau unique.

Plus généralement, à partir d'un balayage en rayons ultrasonores de la sphère unité, nous cherchons à construire une approximation des fonctions de temps de vol

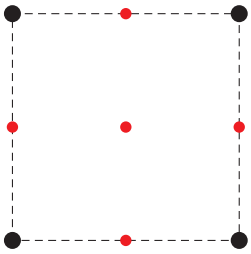


Figure 4.14 – Rayons requis pour une interpolation biquadratique (en rouge), en plus des rayons du pinceau initial (en noir)

et des positions d'arrivée des rayons ultrasonores sur l'intégralité du domaine couvert par le balayage. Pour ce faire, plusieurs possibilités basées sur des constructions de polynômes sont envisageables :

1. La construction d'un polynôme interpolateur global pour l'ensemble des rayons ultrasonores.
2. L'utilisation d'un polynôme approchant les données sans les interpoler au moyen d'une régression polynomiale.
3. L'utilisation d'une interpolation polynomiale par morceaux sur chacun des pinceaux.

La première solution nécessite l'utilisation de polynômes de degrés élevés ce qui, comme nous avons pu le voir en section 4.4.2, peut entraîner des phénomènes d'oscillations incohérents avec la forme usuelle des fronts d'onde. De plus, la construction de tels polynômes implique l'inversion de matrices de taille $n \times n$ avec n le nombre de rayons interpolés et peut donc nécessiter des temps de calcul importants.

La seconde solution, bien qu'elle mette en jeu des polynômes de degrés choisis, produit une approximation des fonctions de temps de vol et de position des rayons. Selon la forme des fonctions interpolées, cette approximation peut engendrer des erreurs importantes.

L'interpolation polynomiale par morceaux réunit les avantages des deux approches :

— Les échantillons sont interpolés et non approximés, elle a donc une valeur exacte aux points d'échantillonnage.
— Elle fait intervenir des polynômes de degrés choisis et évite ainsi les problèmes liés au coût calculatoire de polynômes de degrés élevés et le phénomène de Runge.

En reprenant la méthode d'interpolation quadratique présentée en section 4.4.2, on peut construire un interpolateur local pour chaque pinceau. On dispose alors d'une fonction polynomiale par morceaux sur l'ensemble de la zone de couverture des pinceaux. La continuité de cette fonction est assurée par le fait que les pinceaux voisins partagent des rayons. Les interpolateurs de deux pinceaux voisins sont donc égaux sur l'axe qu'ils partagent. Notons cependant que la continuité de la dérivée n'est pas assurée.

Les fonctions de temps de vol et de position des rayons ne sont alors pas de classe $\mathscr{C}^1$. Or, dans les configurations de contrôle que nous cherchons à traiter dans un premier temps, les géométries traversées par les ondes sont lisses, de sorte à éviter les effets de diffraction que le modèle des pinceaux ne traite pas correctement. Ainsi, les fonctions à interpoler sont généralement à dérivée continue, propriété que nous exploitons dans la suite de manière à améliorer l'efficacité des interpolants que nous construisons.

Interpolation polynomiale d'Hermite

L'interpolation polynomiale d'Hermite est une extension de l'interpolation polynomiale de Lagrange qui permet de spécifier des conditions de dérivée en plus des conditions de valeur pour l'interpolation. Étant donnés une fonction $f : \mathbf{R} \to \mathbf{R}$, un ensemble d'échantillons $(x_i)_{1 \leq i \leq n}$ et un polynôme interpolateur $P \in \mathbf{R}[X]$, l'équation d'interpolation s'écrit :

$$\forall i \in [\![1, n]\!], \begin{cases} P(x_i) = f(x) \\ P'(x_i) = f'(x_i) \end{cases}$$

Dans le cas général, le degré minimal du polynôme P pour vérifier ces conditions est $2n-1$ puisque $2n$ conditions de valeur lui sont imposées. La construction des polynômes interpolateurs d'Hermite utilise le carré des polynômes de Lagrange :

$$q_i(X) = L_i(X)^2 = \prod_{j=0, j \neq i}^{n} \left(\frac{X - x_j}{x_i - x_j} \right)^2$$

Ces polynômes vérifient les conditions suivantes :

$$\forall i \in [\![1, n]\!] \begin{cases} q_i(x_i) = 1, q_i'(x_i) = \displaystyle\sum_{j=0, j \neq i}^{n} \frac{2}{x_i - x_j} \\ \forall j \in [\![1, n]\!] \setminus i, q_i(x_j) = 0, q_i'(x_j) = 0 \end{cases}$$

On pose :

$$P(X) = \sum_{i=0}^{n} q_i(X)(f(x_i) + (X - x_i)(f'(x_i) - q_i'(x_i) f(x_i)))$$

P est au plus de degré $2n+1$ et on vérifie aisément qu'il interpole effectivement la fonction f et sa dérivée aux points d'échantillonnage. De cette façon, on dispose d'une interpolation de classe $\mathscr{C}^1$ de la fonction f.

Erreur d'interpolation

De même que pour l'interpolation de Lagrange, l'erreur commise par l'interpolation d'Hermite peut être estimée. Atkinson (1980) démontre qu'il est possible de caractériser l'erreur d'interpolation sur l'intervalle réel I contenant les échantillons $(x_i)_{1 \leq i \leq n}$ dans le cas d'une fonction f dérivable $n+2$ fois :

$$\forall t \in I, \exists \xi \in I, f(t) - P(t) = \frac{f^{(n+2)}(\xi)}{(n+1)!} \prod_{i=1}^{n} (t - x_i)^2$$

Ainsi, on peut majorer l'erreur comme précédemment :

$$\forall t \in I, |f(t) - P(t)| \leq \frac{\max_{t \in I} |f^{(n+2)}(t)|}{(n+1)!} \prod_{i=1}^{n} (t - x_i)^2$$

Interpolation cubique

L'interpolation d'Hermite produit des polynômes de degré au minimum égal à 3. Dans le cas unidimensionnel, il est donc nécessaire de calculer la valeur de la fonction $f : \mathbf{R} \to \mathbf{R}$ à interpoler et de sa dérivée en au moins deux points d'échantillonnage x_0 et x_1. On construit un polynôme interpolateur P :

$$P = \sum_{i=0}^{3} a_i X^i$$

Le calcul des coefficients de P nécessite la résolution du système d'équations suivant :

$$\begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 \\ 1 & x_1 & x_1^2 & x_1^3 \\ 0 & 1 & 2x_0 & 3x_0^2 \\ 0 & 1 & 2x_1 & 3x_1^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ f'(x_0) \\ f'(x_1) \end{pmatrix}$$

Dans le cas où les coordonnées des échantillons vérifient $x_0 = 0$ et $x_1 = 1$, on a :

$$\begin{aligned} a_0 &= f(0) \\ a_1 &= f'(0) \\ a_2 &= -3f(0) + 3f(1) - 2f'(0) - f'(1) \\ a_3 &= 2f(0) - 2f(1) + f'(0) + f'(1) \end{aligned}$$

Interpolation bicubique

Dans le cas d'une fonction $f : \mathbf{R}^2 \to \mathbf{R}$, on peut construire un interpolateur bicubique :

$$P = \sum_{i=0}^{3} \sum_{i=0}^{3} a_{ij} X^i Y^j$$

P comporte 16 coefficients indépendants. Par conséquent, nous devons disposer de 16 équations indépendantes pour que le système d'équations admette une solution unique. En plus des équations concernant les valeurs de f (en 4 échantillons), les dérivées premières selon la coordonnée x (au nombre de 4), selon la coordonnée y (au nombre de 4 également), on peut considérer la dérivée seconde croisée $\frac{\partial^2 f}{\partial x \partial y}$.

Autrement, il est également possible de ne considérer que les équations portant sur les valeurs et dérivées premières. Dans ce cas, le système est sous déterminé. Le choix usuellement opéré dans cette situation consiste à chercher le polynôme solution du système d'équations de norme minimale afin de limiter la variance de la fonction interpolée sur son ensemble de définition.

Pour la suite, dans l'objectif d'approcher au mieux les fonctions interpolées, nous choisissons de considérer les dérivées croisées lors des calculs d'interpolateurs bicubiques par interpolation d'Hermite.

La démarche de calcul des coefficients du polynôme est identique à celle du cas unidimensionnel. Pour une interpolation sur des échantillons $(x_i, y_j)_{1 \leq i,j \leq 3}$ aux valeurs connues $(0,0)$, $(0,1)$, $(1,1)$ et $(1,0)$, on peut les calculer au moyen du produit

matriciel suivant :

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{pmatrix} F \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

avec

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}, F = \begin{pmatrix} f(0,0) & f(0,1) & f_y(0,0) & f_y(0,1) \\ f(1,0) & f(1,1) & f_y(1,0) & f_y(1,1) \\ f_x(0,0) & f_x(0,1) & f_{xy}(0,0) & f_{xy}(0,1) \\ f_x(1,0) & f_x(1,1) & f_{xy}(1,0) & f_{xy}(1,1) \end{pmatrix}$$

où $f_x = \frac{\partial f}{\partial x}$, $f_y = \frac{\partial f}{\partial y}$ et $f_{xy} = \frac{\partial f}{\partial x \partial y}$

Interpolation bicubique sur un pinceau

De la même manière que pour l'interpolation biquadratique de pinceaux présentée en section 4.4.2, nous utilisons un repère local au pinceau. Cette fois-ci, la grille d'interpolation est de taille 2×2, mais l'échantillonnage est effectué sur les fonctions de temps de vol et de position d'arrivée des rayons ainsi que sur les dérivées en x et y et sur la dérivée croisée en x, y.

En général, nous ne disposons pas de l'expression analytique de la dérivée des temps de vol et positions d'arrivée. Pour construire les interpolateurs d'Hermite, nous effectuons un calcul numérique de dérivées. Celui-ci permet d'obtenir une approximation de la dérivée d'une fonction à l'aide de points d'échantillonnage de celle-ci. Les détails de ce calcul de dérivées sont abordés dans l'annexe C.

Une fois les interpolateurs de temps de vol et de position d'arrivée construits, on peut obtenir une estimation de temps de vol et de position d'arrivée pour une coordonnée (x, y) du repère local au pinceau en appliquant, de la même manière que précédemment, le schéma d'évaluation de Ruffini-Horner. En notant $P(X, Y) = \sum_{i,j=0}^{3} a_{ij} X^i Y^j$ un polynôme interpolateur de degré 3 pour le temps de vol ou la position d'arrivée :

$$\begin{aligned} P(x,y) = & a_{00} + x*(a_{10} + x*a_{20} + x*a_{30}) + \\ & y*(a_{01} + x*(a_{11} + x*a_{21} + x*a_{31}) + \\ & y*(a_{02} + x*(a_{12} + x*a_{22} + x*a_{32}) + \\ & y*(a_{03} + x*(a_{13} + x*a_{23} + x*a_{33})))) \end{aligned}$$

Ceci nous permet, comme dans le cas de l'interpolation biquadratique, de découper chaque pinceau en sous-pinceaux pour un coût inférieur à celui du calcul complet du sous-pinceau par lancer de rayons. L'interpolation d'Hermite par morceaux permet la construction de fonctions continues de temps de vol et de position finale des rayons et approchant plus efficacement les fonctions réelles.

4.5 Conclusion

Nous avons vu dans ce chapitre une méthode semi-analytique de calcul de réponse impulsionnelle de pinceaux de rayons ultrasonores. Cette méthode, actuellement utilisée dans *CIVA*, se base sur une approximation paraxiale linéaire du front

d'onde. Nous en avons détaillé les mécanismes, de la propagation au calcul de signal ultrasonore.

Puis, nous avons adapté les principes du modèle des pinceaux à un mode de calcul basé uniquement sur des rayons, exploitant un paquet de rayons pour déduire les informations que la méthode des pinceaux obtient au travers des matrices paraxiales d'approximation linéaire. De cette façon, à balayage identique et pour un nombre de rayons comparables, on obtient des résultats identiques sans avoir à calculer les matrices paraxiales.

Enfin, nous avons présenté une méthode d'interpolation permettant d'approximer le front d'une onde ultrasonore plus efficacement en exploitant sa régularité au moyen d'interpolateurs polynomiaux. Les grandeurs résultant du tracé de rayons sont interpolées et permettent la constitution de réponses impulsionnelles de manière analogue à l'approche originale des pinceaux.

CHAPITRE 5

Reconstitution de la réponse impulsionnelle

Sommaire

Dans ce chapitre, nous proposons une façon plus efficace de constituer la réponse impulsionnelle de la surface d'un traducteur ultrasonore sur un point de champ à partir de pinceaux recouvrant le capteur. Le front d'onde est approximé comme cela a été expliqué dans le chapitre 4, mais nous cherchons à minimiser le nombre de sous-pinceaux calculés.

Pour ce faire, nous construisons un échantillonnage de la surface du traducteur en éléments rectangulaires de taille prédéterminée. Pour chaque point de cet échantillonnage, nous calculons une approximation des grandeurs de rayon ultrasonore (temps de vol, amplitude, direction de départ...) à l'aide des polynômes interpolateurs. Puis, pour chaque élément de l'échantillonnage, nous calculons la réponse impulsionnelle élémentaire correspondante.

Les éléments développés au cours de ce chapitre utilisent :

— L'outil de lancer de rayons ultrasonores présenté dans le chapitre 3,

— La méthode de reconstruction de fonctions par interpolation polynomiale par morceaux proposée dans le chapitre 4,

Par ailleurs, l'hypothèse de couverture du traducteur par des pinceaux sur laquelle se base ce chapitre résulte de résultats que nous présentons dans le chapitre 6.

5.1 Hypothèses de départ et objectifs

5.1.1 Hypothèse de couverture

Comme nous l'avons mentionné dans le chapitre 2 de modélisation des ondes ultrasonores, pour obtenir une réponse impulsionnelle exacte du traducteur ultrasonore sur un point de champ, nous devons considérer l'effet de l'ensemble de la surface émettrice du traducteur sur le point de champ.

Pour ce faire, les pinceaux utilisés pour le calcul de champ doivent efficacement couvrir le traducteur et permettre une approximation efficace et suffisamment précise des grandeurs résultant d'un lancer de rayons ultrasonores. Les détails de l'algorithme permettant d'arriver à ce résultat tout en minimisant le nombre de pinceaux lancés font l'objet du chapitre 6.

Dans l'ensemble de ce chapitre, les pinceaux couvrant la surface du traducteur sont considérés comme une donnée d'entrée. Nous supposons donc que nous disposons d'un ensemble $(P_i)_{1\leq i\leq n_p}$ de pinceaux représentant de manière exhaustive la zone de visibilité du traducteur depuis le point de champ et approximant efficacement les grandeurs nécessaires à la reconstitution de réponse impulsionnelle, comme dans l'exemple en figure 5.1

5.1.2 Objectifs

À partir de ces pinceaux, l'objectif est de calculer efficacement la réponse en déplacement au niveau du point de champ considéré de l'émission d'une impulsion ultrasonore par la surface du traducteur. Comme nous l'avons vu précédemment, un traducteur peut être composé d'éléments pouvant être soumis à des impulsions différentes, ce que nous devons prendre en compte lors du calcul.

Dans la suite de ce chapitre, nous considérons deux méthodes principales pour calculer les contributions de chaque élément du traducteur sur la réponse impulsionnelle globale. Disposant d'approximations pour chacune des grandeurs de rayons ultrasonores sur ce que nous supposons être l'angle solide de visibilité du traducteur, une première approche consiste à subdiviser chaque pinceau en suffisamment de

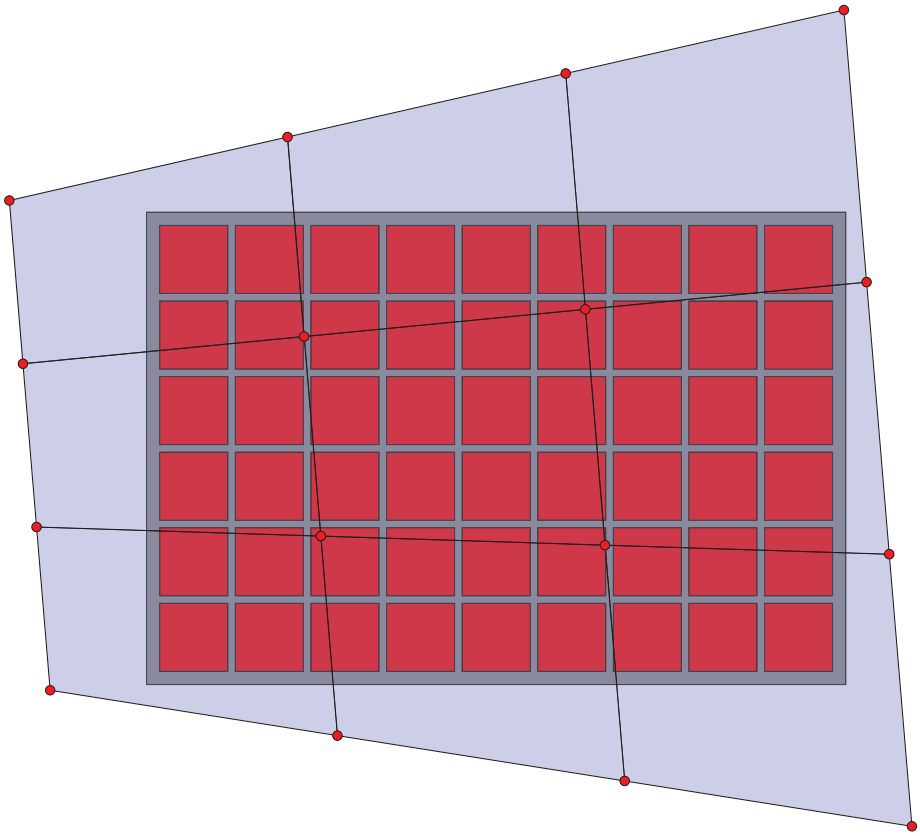


Figure 5.1 – Couverture de la surface du traducteur par des pinceaux

sous-pinceaux par interpolation afin d'avoir une précision suffisante pour construire la réponse impulsionnelle globale.

La seconde approche que nous proposons consiste à supposer l'injectivité de la fonction qui associe une position sur le traducteur à une direction de départ. Comme nous allons le voir par la suite, cette hypothèse simplifie le processus de reconstruction de la réponse impulsionnelle, augmentant les possibilités de parallélisation à l'aide d'instructions SIMD. Elle présente néanmoins des limites, que nous détaillons dans la suite, tout en proposant des perspectives pour les outrepasser.

5.1.3 Critère de précision pour la reconstruction de réponse impulsionnelle

Comme cela a été vu au cours du chapitre 4, la finesse de l'échantillonnage du traducteur influe tout particulièrement sur la forme de la réponse impulsionnelle élémentaire associée à un sous-pinceau. Connaissant le spectre du signal d'excitation du traducteur, on peut encadrer celui du signal mesuré au point de champ en considérant le milieu de propagation comme un **filtre passif**.

Ainsi, pour reconstruire le signal final au niveau du point de champ, on peut appliquer le **critère de Shannon**. Selon ce critère, si l'échantillonnage du signal reconstruit est effectué avec une fréquence supérieure ou égale à deux fois la fréquence maximale de ce dernier, sa représentation discrète est exacte.

Autrement dit, un échantillonnage de fréquence $2f_{max}$ avec f_{max} la fréquence du signal reconstruit contient la totalité de l'information de celui-ci. En considérant la méthode de calcul de l'étalement temporel d'un pinceau détaillée dans le chapitre 4,

on a :

$$\Delta t = dx * s * cos(\alpha)$$

avec dx la dimension maximale du pinceau à son arrivée sur le traducteur, α l'angle entre le rayon axial du pinceau et la surface du traducteur et s la lenteur de l'onde considérée. La figure 5.2 schématise ce calcul.

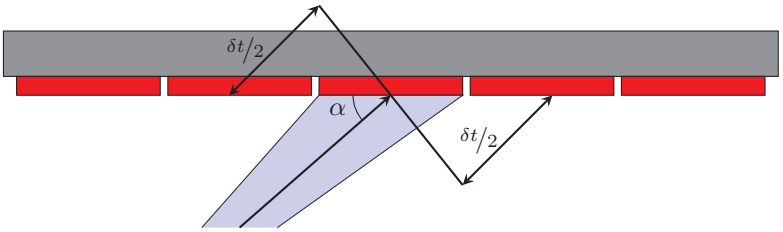


Figure 5.2 – Calcul de l'étalement temporel d'un pinceau

Le critère de Shannon se traduit alors par :

$$dx * s * cos(\alpha) < \frac{1}{2f_{max}}$$

Soit :

$$dx < \frac{1}{2f_{max} * s * cos(\alpha)} < \frac{1}{2f_{max} * s}$$

Ainsi, en imposant une dimension maximale d'échantillonnage du traducteur égale à $\frac{1}{2f_{max}*s}$, le critère de Shannon est respecté et l'exactitude de la reconstruction du signal est garantie.

5.2 Approche par suréchantillonnage des pinceaux

5.2.1 Respect du critère de Shannon

Comme supposé, l'ensemble de pinceaux couvrant le traducteur $(P_i)_{1\leq i\leq n_p}$ est associé à des interpolateurs dont la précision a été vérifiée et jugée suffisante pour l'intégration de la réponse impulsionnelle. De plus, la totalité de la surface du traducteur visible est couverte par ces pinceaux.

Ainsi, pour obtenir une réponse impulsionnelle correcte, il suffit de respecter le critère de Shannon pour la construction du signal. Comme nous avons pu le voir en section 5.1.3, ce critère se traduit par un étalement temporel maximal pour chaque pinceau et peut être rapporté à un critère de dimension.

Connaissant les positions finales des rayons constituant un pinceau P_i, on peut déterminer si ses dimensions dépassent la dimension maximale déduite du critère d'échantillonnage de Shannon. Le cas échéant, le pinceau est subdivisé en sous-pinceaux, de manière équitable selon les coordonnées locales au pinceau, comme dans l'exemple en figure 5.3.

Ce processus est répété tant qu'il existe des pinceaux ou des sous-pinceaux ne vérifiant pas le critère de dimensions. À la fin, on dispose d'une liste de pinceaux et de sous-pinceaux dont les contributions peuvent être calculées, sans dépasser l'étalement temporel plafond pour la construction du signal.

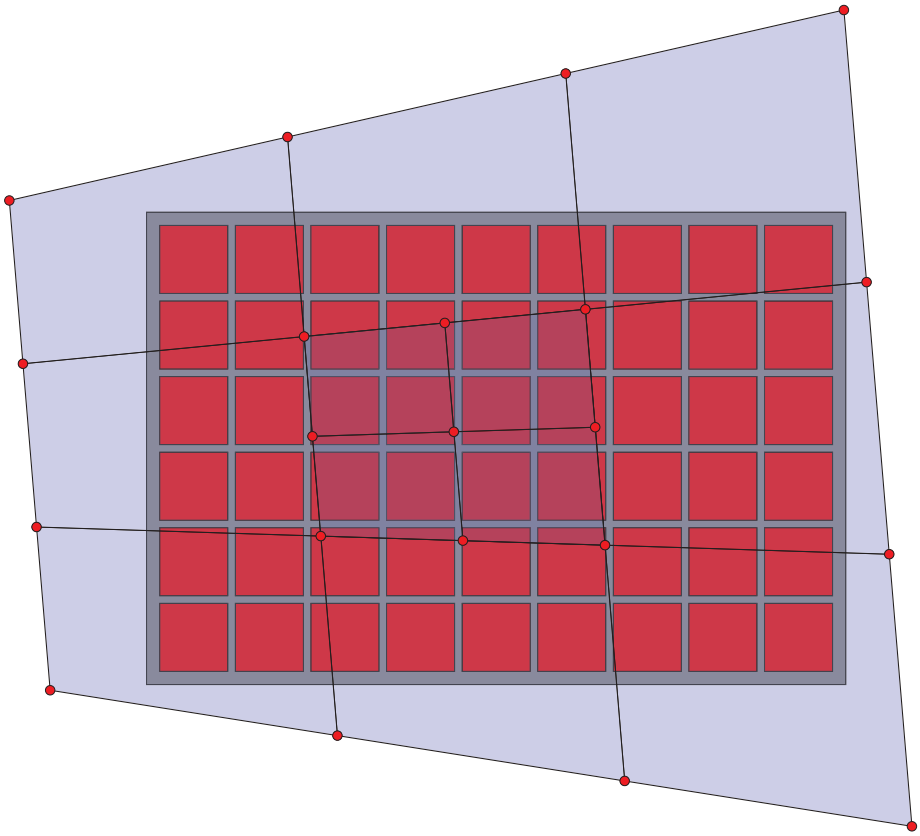


Figure 5.3 – Subdivision d'un pinceau

5.2.2 Critère de couverture d'élément

La surface d'un traducteur ultrasonore peut comporter plusieurs éléments, soumis à des excitations d'amplitudes et de déphasages variables. De plus, entre ces éléments, il existe un espace inactif, l'**espace inter-éléments**, qui ne participe pas à l'émission de perturbations mécaniques.

Dans le cas d'un sous-pinceau entièrement contenu dans la surface d'un élément de traducteur, le déphasage et le facteur d'amplitude appliqués lors de la construction de réponse impulsionnelle élémentaire sont ceux de l'élément du traducteur contenant le sous-pinceau. Cependant, puisque les pinceaux que l'on considère sont issus d'un algorithme de lancer de rayons (détaillé dans le chapitre 6), il est possible que des sous-pinceaux résultant de leur subdivision couvrent plusieurs éléments du traducteur à la fois, ou de l'espace inter-éléments.

Dans ce cas, pour construire la réponse impulsionnelle, il est nécessaire de procéder à la subdivision du pinceau de manière à correctement évaluer la contribution qu'il représente. Comme précédemment, on peut procéder à une subdivision en quatre sous-pinceaux de taille égale selon les coordonnées locales du pinceau. Néanmoins, cette façon de procéder peut nécessiter de nombreuses subdivisions pour arriver à une couverture efficace de la surface du traducteur.

5.2.3 Avantages et inconvénients de l'approche

Nous venons de présenter une solution possible pour la construction de réponse impulsionnelle qui permet de respecter le critère de Shannon pour la construction

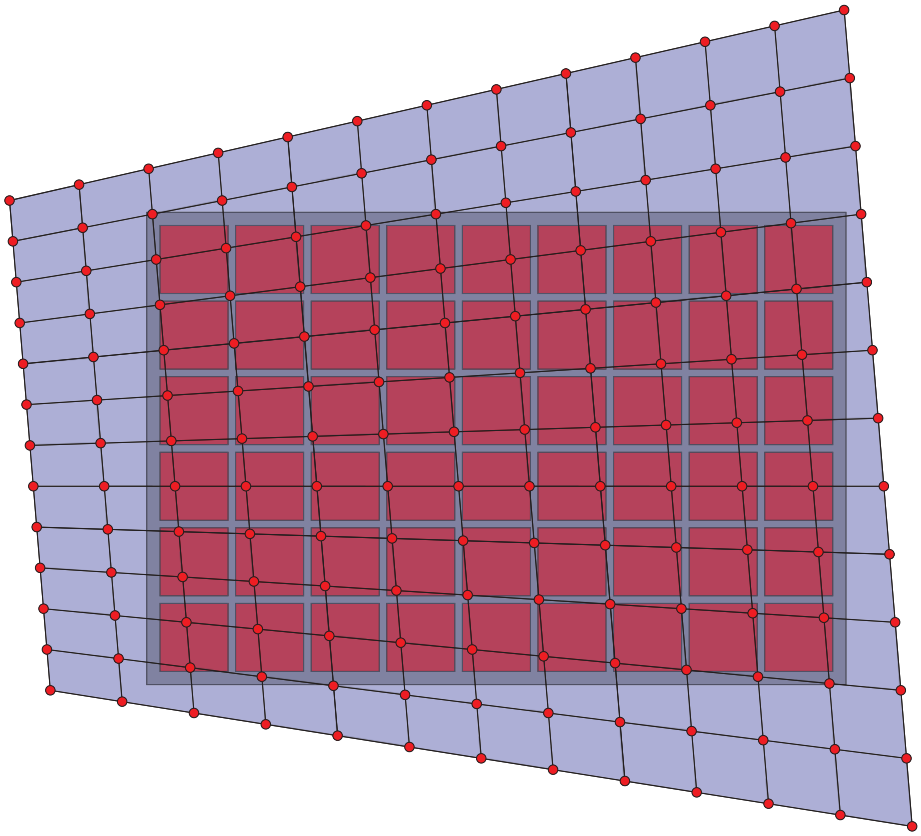


Figure 5.4 – Raffinement des pinceaux sur la surface du traducteur

du signal et de décomposer les sous-pinceaux de sorte qu'ils ne chevauchent pas plusieurs éléments du traducteur simultanément.

Cependant, dans le cas général de pinceaux non orientés selon les limites du traducteur ou de traducteurs aux formes complexes, le processus de subdivision des pinceaux peut s'avérer particulièrement coûteux pour suivre efficacement la bordure du traducteur. En particulier, pour des traducteurs multiéléments composés de nombreux éléments de petite taille, un grand nombre de sous-pinceaux est requis pour reproduire la structure géométrique du traducteur.

En pratique, cette structure ne devrait pas avoir d'impact sur la densité d'échantillonnage en sous-pinceaux : le critère de Shannon n'est pas affecté par ce facteur. Pour cette raison, nous proposons dans la suite une méthode évitant les problèmes liés à la forme ou à la distribution d'éléments du traducteur.

5.3 Approche par inversion d'interpolation

5.3.1 Principe

Nous avons vu en section 5.2.1 que le respect du critère de Shannon peut être obtenu en fixant un plafond de taille des sous-pinceaux intégrés. La seconde condition pour obtenir une réponse impulsionnelle exacte est que chaque sous-pinceau ne doit couvrir qu'un unique élément du traducteur à la fois.

De ces deux conditions, nous pouvons produire un échantillonnage « idéal » de la surface du traducteur. Celui-ci représente une distribution de sous-pinceaux qui,

si elle pouvait être obtenue par la première approche de construction de réponse impulsionnelle présentée en section 5.2, permettrait à la fois d'obtenir un signal exact et d'éviter les écueils présentés précédemment, liés à la structure du traducteur.

Un tel échantillonnage, pour être valable sur l'ensemble des points de champ, doit respecter le critère de Shannon pour toutes les directions d'incidence possibles sur le traducteur. Ainsi, son pas d'échantillonnage est fixé à la valeur suivante :

$$\frac{1}{2f_{max}.s}$$

Les espaces inter-éléments ne sont pas pris en compte par cet échantillonnage et chaque élément est séparé des autres éléments, de telle sorte que chaque sous-pinceau ne couvre qu'un élément du traducteur à la fois. Sur cet échantillonnage, les sous-pinceaux voisins partagent des échantillons communs, comme présenté en figure 5.5.

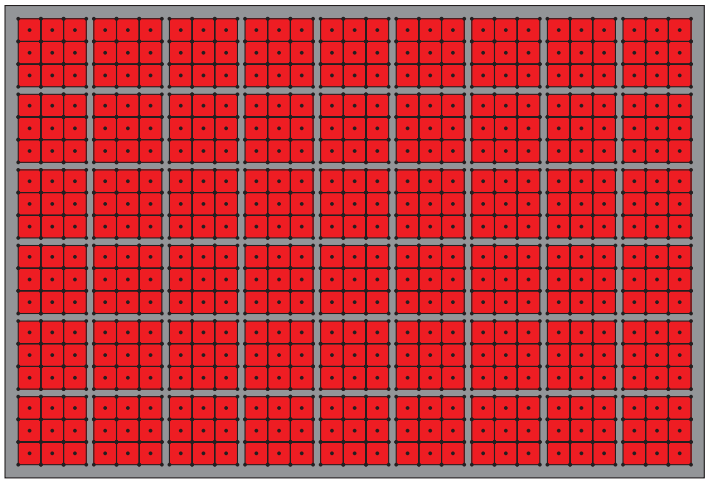


Figure 5.5 – Échantillonnage régulier de la surface du traducteur à pour respecter le critère de Shannon

Chaque échantillon est associé à un indice et une paire de coordonnées en deux dimensions qui correspondent à un système de coordonnées local au traducteur. Un sous-pinceau est alors identifié par quatre indices d'échantillons correspondant à ses quatre coins ainsi qu'un indice d'échantillon central, comme cela est illustré en figure 5.5.

Pour exploiter cet échantillonnage lors de la construction de réponse impulsionnelle, il faut pouvoir attribuer à chaque échantillon l'ensemble des grandeurs de rayons ultrasonores utiles au calcul d'une réponse impulsionnelle élémentaire. Une fois ceci fait, le processus de construction de réponse impulsionnelle est identique à celui utilisé dans le cadre de l'approche par suréchantillonnage des pinceaux détaillée en section 5.2.

Notons que nous émettons pour ce faire l'hypothèse de l'existence d'une fonction associée à chaque grandeur ayant pour domaine de définition la surface du traducteur. Cela revient à supposer que chaque échantillon est recouvert par au plus un pinceau.

5.3.2 Identification du pinceau recouvrant

Pour calculer les grandeurs nécessaires à la construction des réponses impulsionnelles élémentaires de chaque sous-pinceau, il nous faut, pour chaque échantillon :

1. Identifier le pinceau recouvrant,
2. Utiliser les interpolateurs de ce pinceau pour estimer les grandeurs de rayons ultrasonores.

Test d'appartenance rapide

Un premier élément essentiel afin de déterminer le pinceau recouvrant d'un échantillon est un test de recouvrement qui, pour un pinceau et un échantillon identifié par paire de coordonnées, décide si l'échantillon est contenu dans la surface couverte par le pinceau. Étant donné notre mode de représentation d'un pinceau, il s'agit donc de tester l'appartenance d'un point à la surface d'un quadrangle.

Notons que puisque la méthode d'interpolation que nous utilisons (interpolation d'Hermite) est continue et à dérivée continue aux bornes des pinceaux, si un pinceau est à la frontière entre deux pinceaux, qu'il soit identifié comme appartenant à l'un ou l'autre des pinceaux ne change rien si les deux interpolateurs sont correctement construits. Par conséquent, le test d'appartenance que nous utilisons peut présenter une marge d'erreur aux bornes du quadrilatère sans que cela ne cause de problèmes.

Nous choisissons de décomposer la surface du pinceau à l'arrivée sur le traducteur en deux triangles. Le test d'appartenance au quadrangle est alors la conjonction de tests d'appartenance aux deux triangles composant le quadrilatère du pinceau. Notons que la frontière entre les deux triangles ne pose aucun problème dès lors qu'une tolérance numérique est appliquée.

Pour le test d'appartenance d'un point $M = (x, y)$ à un triangle, nous utilisons la méthode barycentrique, notamment utilisée dans Möller et Trumbore (2005). Il s'agit, étant donné un triangle composé des points v_0, v_1 et v_2, de déterminer des coordonnées (u, v) telles que :

$$M = u.v_0 + v.v_1 + (1-u-v).v_2$$

Ces coordonnées sont déterminées analytiquement en développant les formules de résolution du système de deux équations à deux inconnues. Le point (x, y) appartient alors au triangle si et seulement si :

$$\begin{cases} 0 \leq u \leq 1, \\ 0 \leq v \leq 1, \\ 0 \leq u+v \leq 1 \end{cases}$$

⟨*appartientTriangle*⟩≡

```
bool appartientTriangle(vec2<float> M, vec3<float> v0, vec3<float> v1, vec3<float>
  v2) {
   e0 = v2 - v0;
   e1 = v1 - v0;
   e2 = M - v0;

   float produit00 = produitScalaire(e0, e0);
   float produit01 = produitScalaire(e0, e1);
   float produit02 = produitScalaire(e0, e2);
   float produit11 = produitScalaire(e1, e1);
   float produit12 = produitScalaire(e1, e2);
```

```
    float inverseDéterminant = 1 / (produit00 * produit11 - produit01 * produit01);
    float u = (produit11 * produit02 - produit01 * produit12) * inverseDéterminant;
    float v = (produit00 * produit12 - produit01 * produit02) * inverseDéterminant;

    return (u >= 0) && (v >= 0) && (u + v) <= 1;
}
```

⟨*appartientQuadrangle*⟩≡

```
bool appartientQuadrangle(vec2<float> M, vec3<float> v0, vec3<float> v1,
              vec3<float> v2, vec3<float> v3) {
    return appartientTriangle(M, v0, v1, v2) || appartientTriangle(M, v0, v2, v3);
}
```

Accélération de la recherche

Puisque la recherche de pinceau est effectuée une fois par échantillon du traducteur pour chaque point de champ, ce processus peut représenter un temps de calcul considérable. Pour l'accélérer, nous utilisons les solutions suivantes :

— La mise en place d'un test préalable d'intersection approximative basé sur une boîte englobante,

— L'utilisation de notions de voisinage pour accélérer la recherche du pinceau recouvrant,

— L'utilisation d'une structure accélératrice sur les pinceaux.

Le test approximatif d'intersection permet d'éliminer la majorité des pinceaux, comme l'illustre la figure 5.6, en utilisant les boîtes englobantes des pinceaux. La construction et le test d'appartenance à une boîte englobante sont rapides et nécessitent peu de mémoire en comparaison avec les données stockées pour les pinceaux.

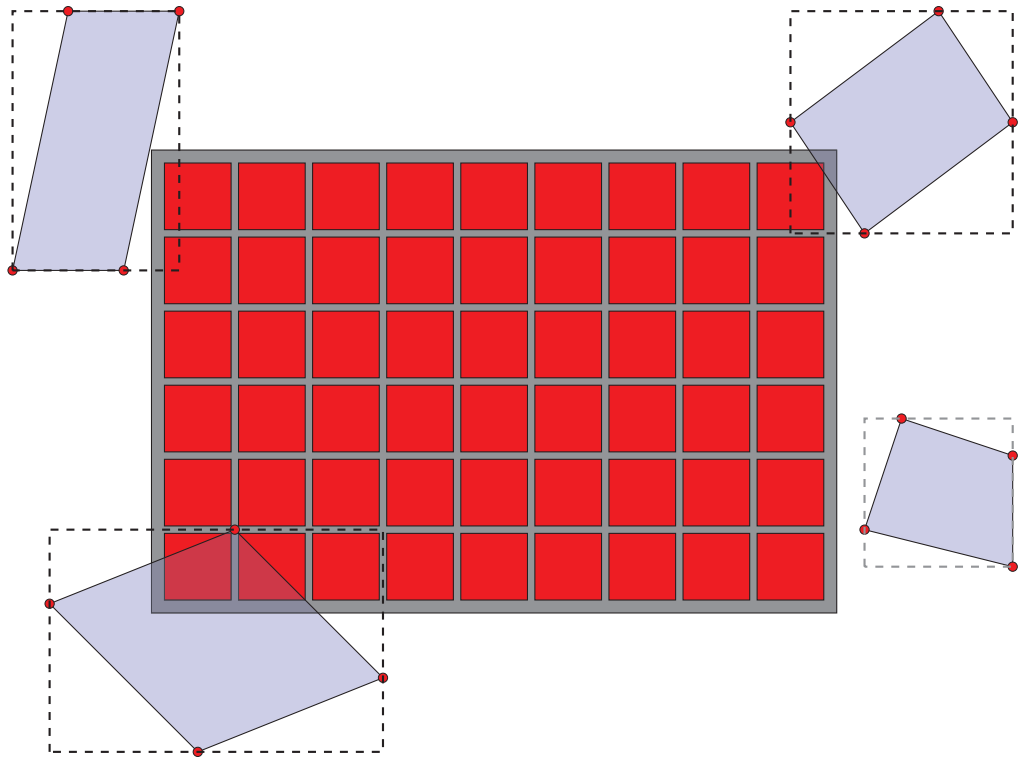


Figure 5.6 – Exemples de boîtes englobantes de pinceaux dans les différentes situations possibles

Dans le cas où le processus de détermination des pinceaux recouvrants pour les échantillons du traducteur est effectué de manière séquentielle, il est possible de supprimer un grand nombre de tests pour un échantillon $i+1$ en le testant en premier lieu avec le pinceau trouvé pour l'échantillon i. De manière plus générale, deux échantillons proches ont de fortes chances d'être recouverts par le même pinceau, en particulier lorsque peu de pinceaux sont à considérer.

Enfin, dans le cas où le nombre de pinceaux considérés est important (de l'ordre de quelques dizaines), nous construisons une structure accélératrice basée sur les BVH, de la même manière que pour le lancer de rayons. Nous reprenons le test d'appartenance et de boîte englobante des pinceaux pour l'intégrer aux algorithmes de construction et d'exploration de structures accélératrices de type BVH proposés par Pharr et Humphreys (2004).

5.3.3 Inversion d'interpolation

Une fois le pinceau recouvrant un échantillon de coordonnées locales au capteur $M = (x, y)$ identifié, nous cherchons à déterminer les coordonnées locales au pinceau (u, v) telles que l'interpolateur de position du pinceau recouvrant coïncide avec l'échantillon. Notons :

— $P_x^{(3)}(X, Y)$, $P_y^{(3)}(X, Y)$ et $P_z^{(3)}(X, Y)$ les interpolateurs des trois coordonnées des points d'arrivée des rayons ultrasonores pour le pinceau recouvrant,
— $projection(x, y, z)$ la projection du point (x, y, z) de l'espace dans le repère local au traducteur,
— P_{trad} le plan du traducteur.

L'équation vérifiée par les coordonnées locales recherchées s'écrit alors :

$$projection(P_x^{(3)}(u, v), P_y^{(3)}(u, v), P_z^{(3)}(u, v)) = (x, y)$$

Si l'on considère cette fois l'interpolateur $P^{(2)}$ des projections des points d'arrivée des rayons ultrasonores sur le traducteur, on a :

$$\begin{cases} P_x^{(2)}(u, v) = x \\ P_y^{(2)}(u, v) = y \end{cases}$$

Puisqu'il n'existe, à notre connaissance, pas de méthode analytique permettant de déterminer la solution d'un système de deux équations à deux inconnues faisant intervenir des polynômes de degré 3, nous utilisons une méthode numérique.

Comme nous disposons d'un encadrement de la valeur du couple de coordonnées solution ($0 \leq x, y \leq 1$ en principe), la **méthode de Newton-Raphson** fournit une convergence rapide vers la solution lorsque celle-ci existe. Broyden (1965) détaille le principe de cette méthode basée sur une extrapolation de premier ordre à l'aide du jacobien de la fonction dont on recherche les zéros.

Partant d'une valeur initiale donnée, l'algorithme répète l'itération suivante jusqu'à convergence :

$$\begin{pmatrix} u_{k+1} \\ v_{k+1} \end{pmatrix} = \begin{pmatrix} u_k \\ v_k \end{pmatrix} - J_{P^{(2)}}^{-1}(u_k, v_k) \begin{pmatrix} u_k \\ v_k \end{pmatrix}$$

avec $J_{P^{(2)}}^{-1}$ le jacobien de $P^{(2)}$. La convergence de cette méthode est quadratique.

En plus de fournir une estimation de solution (u, v) pour le système d'équations d'inversion d'interpolation, la méthode de Newton-Raphson donne la valeur de l'erreur commise :

$$\begin{pmatrix} P_x^{(2)}(u,v) - x \\ P_y^{(2)}(u,v) - y \end{pmatrix}$$

En pratique, nous initialisons la recherche de zéro au centre du domaine de recherche, à savoir le point de coordonnées $(0.5, 0.5)$. À chaque itération, nous contrôlons la variation de l'estimation de la solution, notée $\vec{\delta}_n$.

Nous utilisons les critères d'arrêt suivants :

— $\|\vec{\delta}_n\| < \epsilon$ avec ϵ une tolérance (que nous fixons à 10^{-4}),

— Le nombre d'itérations n dépasse une valeur fixée. En pratique, pour notre cas d'utilisation et sur des flottants à simple précision (32 bits selon la norme IEEE 754), une valeur raisonnable pour ce seuil est 6.

Enfin, une fois le critère d'arrêt atteint, on peut vérifier d'une part que l'erreur qu'il renvoie est inférieure au plafond ϵ et d'autre part qu'il est, à une tolérance près, dans le domaine $(0,1) \times (0,1)$ de validité de l'interpolation polynomiale.

5.3.4 Construction des réponses impulsionnelles

Interpolation des échantillons

Nous pouvons utiliser les coordonnées obtenues à l'aide de la méthode décrite dans le paragraphe précédent pour déterminer les valeurs des grandeurs de rayons telles que le temps de vol, l'amplitude, la direction initiale de rayon ultrasonore... Ces grandeurs permettent la construction de réponses impulsionnelles élémentaires pour chaque sous-pinceau de l'échantillonnage du traducteur.

Nous distinguons deux types d'échantillons sur le traducteur :

Les échantillons centraux au centre des sous-pinceaux, pour lesquels on détermine l'amplitude, la lenteur initiale, la polarisation et le temps de vol.

Les échantillons de coin délimitant les sous-pinceaux, pour lesquels on détermine le temps de vol et la direction initiale.

Détail des interpolateurs par grandeur interpolée

Ainsi, pour chaque pinceau, nous devons construire un interpolateur pour les grandeurs mentionnées. Ces grandeurs ne présentent pas le même impact sur le résultat du calcul. En effet, comme nous l'avons expliqué dans le chapitre 4, la précision de l'estimation des temps de vol et positions d'arrivée des rayons est déterminante pour la forme de la réponse impulsionnelle construite. C'est la raison pour laquelle nous utilisons des interpolateurs de degré 3 pour ces grandeurs.

À l'inverse, l'impact des variations des coefficients d'amplitude des rayons ultrasonores et de leurs polarisations sur les résultats de simulation de champ ultrasonore est moindre. En effet, à l'échelle d'un pinceau, les variations de ces grandeurs sont suffisamment faibles pour qu'une interpolation linéaire à l'échelle du pinceau suffise à les modéliser efficacement.

Le calcul des coefficients de divergence, tel que détaillé en section 4.2.3, requiert les valeurs de lenteur et de direction initiale des rayons. L'incertitude sur la valeur de lenteur initiale et sur les directions initiales de rayons ultrasonores se propage linéairement sur la valeur du coefficient de divergence DF.

En milieux isotropes, la variation de lenteur initiale étant nulle, aucune interpolation n'est requise pour déterminer s_0 quel que soit l'échantillon du traducteur. En milieux anisotropes, pour un pinceau arrivé sur le traducteur, la variation de lenteur

initiale est suffisamment faible pour que l'utilisation d'un interpolateur linéaire pour cette grandeur suffise.

Enfin, le même argument est applicable aux directions initiales des rayons qui, dans le cas des pinceaux d'angle solide initial relativement faibles recouvrant le traducteur, varient peu d'une extrémité à l'autre du pinceau et peuvent donc être interpolées linéairement.

Finalement, on utilise pour chaque pinceau les interpolateurs suivants :

Temps de vol Interpolateur d'Hermite de degré 3 à deux variables.

Projection de la position d'arrivée Interpolateur d'Hermite de degré 3 à deux variables.

Direction initiale de rayon Interpolateur bilinéaire.

Lenteur initiale Interpolateur bilinéaire.

Coefficient de transmission en amplitude Interpolateur bilinéaire.

Constitution des réponses impulsionnelles élémentaires

En résumé, le processus de constitution des réponses impulsionnelles élémentaires par la méthode d'inversion d'interpolation se compose des étapes suivantes :

1. Pour chaque échantillon du traducteur, qu'il soit central ou de coin, détermination du pinceau recouvrant.
2. Pour chaque échantillon du traducteur, qu'il soit central ou de coin, détermination des coordonnées locales dans le repère du pinceau recouvrant, par inversion d'interpolation.
3. Pour chaque échantillon de coin, calcul du temps de vol et de la direction initiale en utilisant les interpolateurs du pinceau recouvrant.
4. Pour chaque échantillon central, calcul de l'amplitude, de la polarisation, de la lenteur initiale et du temps de vol à l'aide des interpolateurs du pinceau recouvrant.

Puis, pour chaque sous-pinceau (composé d'un échantillon central et de 4 échantillons de coin), on détermine les valeurs d'amplitude et d'étalement temporel du créneau modélisant la réponse impulsionnelle élémentaire :

— Le temps minimal du créneau t_{min} est le minimum des temps de vol interpolés sur les échantillons du sous-pinceau, auquel on ajoute le retard τ_i appliqué à l'élément i du traducteur auquel appartient le sous-pinceau :

$$t_{min} = min(\{\text{temps des échantillons}\}) + \tau_i$$

— Le temps maximal du créneau t_{max} est le maximum des temps de vol interpolés sur les échantillons du sous-pinceau, auquel on ajoute le retard appliqué à l'élément du traducteur auquel appartient le sous-pinceau :

$$t_{max} = max(\{\text{temps des échantillons}\}) + \tau_i$$

— Le coefficient de divergence DF est calculé comme suit :

$$DF = \frac{s_0}{2\pi\sqrt{\frac{dS}{d\Omega}}}$$

avec s_0 la lenteur initiale interpolée de l'échantillon central, dS l'aire du sous-pinceau sur la surface du traducteur et $d\Omega$ l'angle solide, calculé comme la surface décrite par les directions initiales interpolées des quatre échantillons de coin du sous-pinceau.
— Le coefficient de transmission T_A est la valeur interpolée pour l'échantillon central du sous-pinceau.

L'amplitude du créneau est multipliée par $A_{0,i}$, la valeur d'amplitude appliquée sur l'élément i du traducteur auquel appartient le sous-pinceau. On a donc comme valeur finale d'amplitude pour le créneau :

$$\frac{A_0 \times DF \times T_A \times dS}{t_{max} - t_{min}}$$

5.4 Conclusions et perspectives

Nous avons présenté deux méthodes permettant d'exploiter des pinceaux recouvrant la surface d'un traducteur ainsi que les interpolateurs qui leur sont associés pour construire la réponse impulsionnelle associée à ce traducteur sur le point d'origine des pinceaux.

La première consiste à procéder à des subdivisions par interpolation des pinceaux recouvrant la surface du traducteur jusqu'à ce que chacun des sous-pinceaux respecte le critère de Shannon et puisse être intégré sous la forme d'une réponse impulsionnelle élémentaire modélisée par un créneau temporel.

À partir d'un échantillonnage du traducteur, cette méthode permet de construire une réponse impulsionnelle en un nombre d'itérations indépendant de la structure géométrique du traducteur. La régularité de cette approche, comme nous le verrons dans le chapitre 7, permet une vectorisation efficace et donc de meilleures performances que l'approche par subdivision des pinceaux.

5.5 Conclusion

Nous basant sur les éléments présentés dans le chapitre 3 et sur les résultats de l'algorithme que nous détaillons dans le chapitre 6, nous avons mis en place dans ce chapitre les éléments constitutifs d'approches de reconstitution de réponses impulsionnelles.

La première approche consiste à procéder à des subdivisions par interpolation des pinceaux en utilisant les interpolateurs détaillés dans le chapitre 4. Ces subdivisions doivent permettre de respecter le critère de Shannon sur la taille des contributions à la réponse impulsionnelle et être de taille suffisamment faible pour ne couvrir qu'un élément de la décomposition du traducteur.

En pratique, cette approche est dépendante de la structure du traducteur et de sa décomposition en éléments indépendants. En particulier, elle requiert un grand nombre de subdivisions lorsque les éléments du traducteur sont petits. Notre objectif étant de traiter des traducteurs de forme quelconque, nous avons développé la seconde approche par inversion d'interpolation.

L'approche que nous avons adoptée dans la suite, plus systématique, se base sur un mécanisme d'inversion d'interpolation. Le traducteur est découpé en échantillons réguliers et chacun est associé à un pinceau le recouvrant. Les interpolateurs de

ce pinceau sont utilisés pour extraire des coordonnées locales de l'échantillon, qui servent à calculer les grandeurs de rayon qui lui sont associées.

Notons cependant que ce procédé dépend de l'hypothèse d'unicité du pinceau recouvrant pour chaque échantillon. Si cette hypothèse couvre un grand nombre de cas pratiques, il existe des situations, notamment dans des matériaux anisotropes, dans lesquelles le front d'onde présente des recouvrements. Il est alors possible qu'un échantillon puisse être couvert par plusieurs pinceaux à la fois.

Dans une situation de ce type, l'approche que nous avons présentée doit être adaptée pour être exacte. Pour ce faire, le processus d'identification de pinceau recouvrant pour un échantillon doit déterminer non plus un identifiant unique de pinceau, mais la liste des pinceaux le recouvrant. Puis, pour chaque sous-pinceau de la surface du traducteur, plusieurs réponses impulsionnelles élémentaires peuvent être produites. L'identification des contributions peut être réalisée en tenant compte de l'hypothèse de continuité du front d'onde.

Les deux approches présentées permettent de construire pour chaque point de champ une réponse impulsionnelle qui, une fois convoluée au signal d'excitation du traducteur, donne le signal ultrasonore au point de champ. Une fois ce signal calculé pour chaque point de champ, on peut calculer l'image des maximums qui constitue l'image de champ finale.

Troisième partie

Optimisations du calcul de champ

CHAPITRE 6

Optimisations algorithmiques

Sommaire

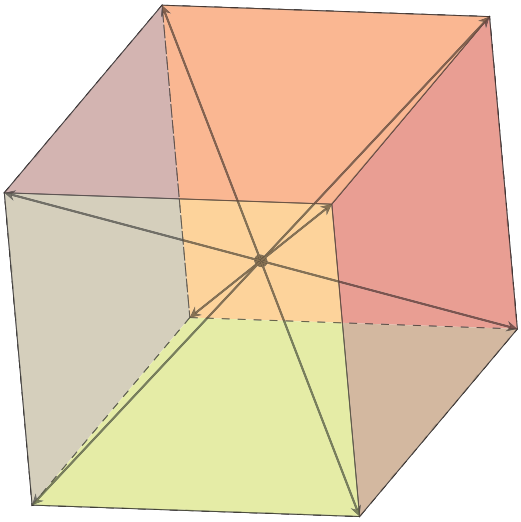


Figure 6.1 – Pinceaux initiaux formant le cube unité de centre $(0, 0, 0)$

L'objectif de ce chapitre est de présenter une méthode de recherche de la zone de visibilité de la surface du traducteur basée sur du lancer de rayons appliquée à la recherche de la surface du traducteur ultrasonore dans le cadre de la simulation de contrôle non destructif. Cette méthode vise la minimisation du nombre de rayons lancés pour trouver le traducteur.

Une fois le traducteur trouvé, sa surface est couverte par des pinceaux. Puisque ceux-ci sont utilisés dans la suite pour interpoler les grandeurs de rayons à l'aide de la méthode décrite dans le chapitre 5, un contrôle de la précision des interpolateurs est effectué afin que tous les pinceaux utilisés interpolent les temps de vol sans erreurs d'échantillon temporel.

6.1 Recherche de surface

6.1.1 Lancer de pinceaux

L'objectif final de la méthode de recherche de surface étant la couverture du traducteur par un ensemble de pinceaux, il est utile d'imposer une notion de voisinage et de structure parmi les rayons lancés lors de la recherche de la surface du traducteur. Pour ce faire, nous disposons les rayons lancés lors de la phase de recherche du traducteur en pinceaux.

Initialisation

Afin de couvrir de manière exhaustive les directions de lancer de rayons possibles, les directions de départ des rayons des pinceaux que nous lançons doivent couvrir l'ensemble de la sphère unité. Les notions de voisinage entre pinceaux permettent le partage de rayons et donc une couverture complète de la sphère unité. Une première étape du balayage de la sphère unité peut consister à lancer 6 pinceaux dont les directions de départ forment un cube inclus dans la sphère unité, comme présenté en figure 6.1.

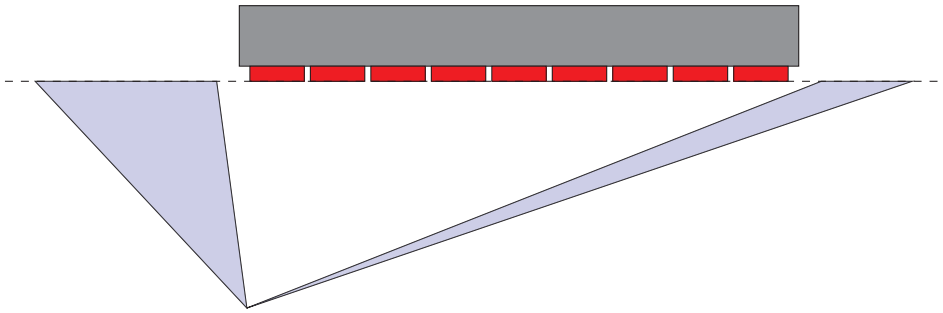


Figure 6.2 – Pinceaux ayant atteint le plan du traducteur sans en toucher la surface

Une première classification des pinceaux

En procédant au lancer des rayons de chaque pinceau à l'aide de la méthode de lancer de rayons ultrasonores explicitée dans le chapitre 3, on peut déterminer pour chaque rayon une information sur la position de son intersection finale. Dans un premier temps, on peut séparer les rayons en deux catégories :

— Les rayons dont l'intersection finale est incluse dans la surface du traducteur, qu'elle ait atteint une partie de la surface émettrice du traducteur ou un espace inter-éléments.

— Les rayons n'ayant pas atteint la surface du traducteur.

Un algorithme de recherche de la surface du traducteur se basant uniquement sur cette information binaire ne peut adopter de stratégie de recherche plus efficace qu'un balayage homogène ou un balayage aléatoire. Dans l'objectif de guider la recherche du traducteur pour réduire le nombre de rayons nécessaires pour trouver la surface du traducteur, il est utile de disposer d'une notion de proximité d'un rayon à la surface du traducteur.

Extension de la surface du traducteur

Dans le cadre de nos simulations, nous nous sommes intéressés à des traducteurs à surface plane. En complétant virtuellement la surface du traducteur par le plan la contenant, il est possible de compléter la classification précédente en y ajoutant la catégorie des rayons n'ayant pas atteint le traducteur, mais ayant intersecté le plan le contenant, comme cela est présenté en figure 6.2.

De cette façon, en calculant la distance du point d'intersection du rayon avec le plan du traducteur à sa surface, on peut déterminer la proximité du rayon au traducteur. Puisque cette distance ne sert qu'à établir un critère de priorité, elle peut être approximée par le minimum de distance du rayon à chacun des coins de la surface du traducteur. Dans le cas où le rayon a atteint le traducteur, cette distance est automatiquement fixée à 0. Ce critère de distance permet, dans une hypothèse de continuité géométrique de la scène, de localiser plus facilement le traducteur.

Raffinement prioritaire du balayage

En effet, toujours dans une hypothèse de continuité géométrique, un raffinement du balayage dans les zones où les rayons sont proches du traducteur permet d'augmenter la densité de rayons aux alentours des zones d'intérêt. De cette manière, les chances de trouver le traducteur avec moins de rayons sont augmentées par rapport à une approche de recherche non informée.

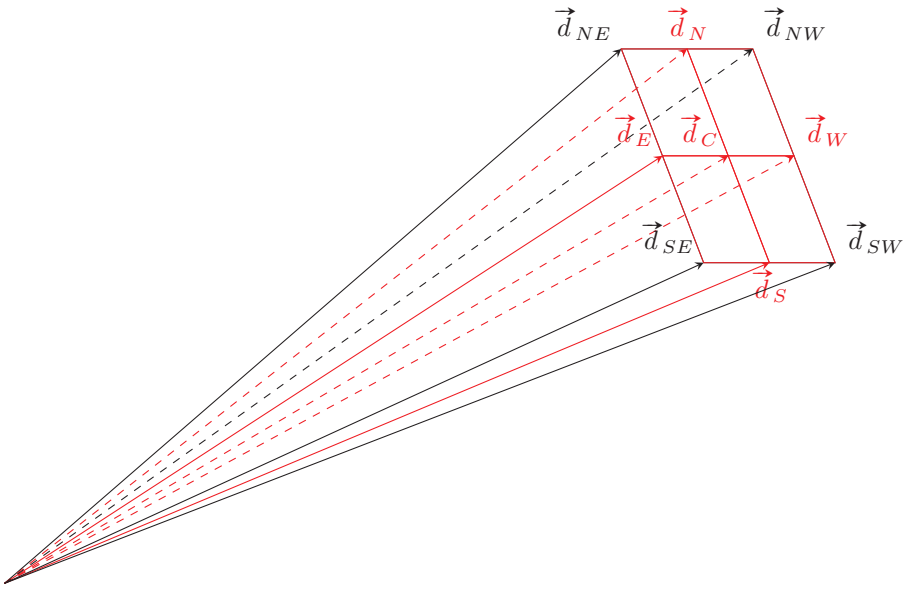


Figure 6.3 – Subdivision d'un pinceau de rayons en 4 sous pinceaux

Pour raffiner le balayage dans une zone donnée, nous procédons à la subdivision d'un pinceau. Dans le cas général, cette subdivision est régulière sur les directions de départ. Étant donné un pinceau constitué des directions de départ $\vec{d}_{NE}$, $\vec{d}_{NW}$, $\vec{d}_{SW}$ et $\vec{d}_{SE}$, nous construisons 5 nouvelles directions (voir la figure 6.3) :

$$\begin{cases} \vec{d}_N = \frac{\vec{d}_{NE}+\vec{d}_{NW}}{\|\vec{d}_{NE}+\vec{d}_{NW}\|} \\ \vec{d}_S = \frac{\vec{d}_{SE}+\vec{d}_{SW}}{\|\vec{d}_{SE}+\vec{d}_{SW}\|} \\ \vec{d}_E = \frac{\vec{d}_{NE}+\vec{d}_{SE}}{\|\vec{d}_{NE}+\vec{d}_{SE}\|} \\ \vec{d}_W = \frac{\vec{d}_{NW}+\vec{d}_{SW}}{\|\vec{d}_{NW}+\vec{d}_{SW}\|} \\ \vec{d}_C = \frac{\vec{d}_{NE}+\vec{d}_{NW}+\vec{d}_{SE}+\vec{d}_{SW}}{\|\vec{d}_{NE}+\vec{d}_{NW}+\vec{d}_{SE}+\vec{d}_{SW}\|} \end{cases}$$

Ce qui nous permet de constituer les 4 sous-pinceaux issus de la subdivision :

$$\begin{cases} \{\vec{d}_{NE}, \vec{d}_N, \vec{d}_C, \vec{d}_E\} \\ \{\vec{d}_N, \vec{d}_{NW}, \vec{d}_W, \vec{d}_C\} \\ \{\vec{d}_C, \vec{d}_W, \vec{d}_{SW}, \vec{d}_S\} \\ \{\vec{d}_E, \vec{d}_C, \vec{d}_S, \vec{d}_{SE}\} \end{cases}$$

Ce processus de subdivision reprend les idées principales de la méthode *Hierarchical Triangular Mesh* proposée par Szalay et collab. (2007) en l'adaptant aux pinceaux quadrangulaires. De cette façon, il est possible d'obtenir une distribution de rayons convergeant vers la surface du traducteur et organisée en pinceaux de rayons.

Arbre de subdivisions

Les subdivisions successives des pinceaux créent une hiérarchie. En considérant la subdivision comme une relation de parenté entre les pinceaux, il est possible de construire un **arbre de subdivisions** des pinceaux. Le nœud racine de cet arbre

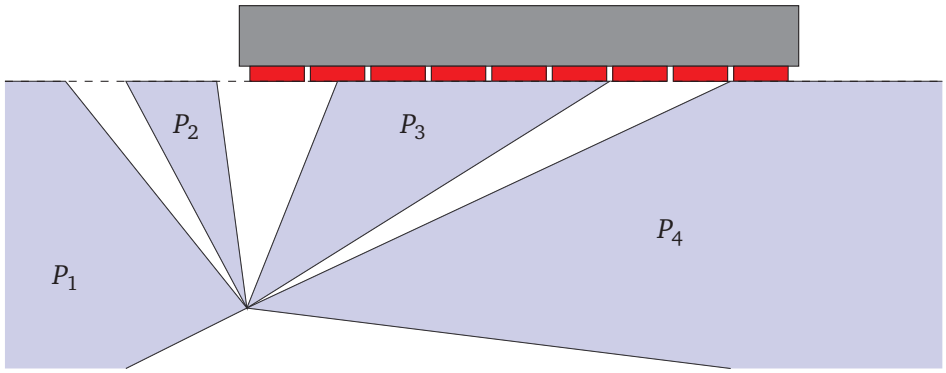


Figure 6.4 – Différents états de pinceaux à l'arrivée sur la surface du traducteur

possède 6 nœuds fils qui sont les 6 pinceaux correspondant aux faces du cube de départ.

De cette façon, la recherche de la surface du traducteur en explorant l'espace par lancer des rayons peut être rapportéee à une exploration d'arbre. On peut alors considérer diverses stratégies pour cette exploration :

— Le parcours en largeur, qui revient à subdiviser uniformément l'ensemble des pinceaux jusqu'à trouver la surface du traducteur.
— Le parcours en profondeur, inadapté dans cette situation. En effet, explorer une branche de l'arbre jusqu'à son extrémité revient à subdiviser autant que possible une partie d'un pinceau avant d'explorer les autres zones. Un tel type de parcours n'est pas adapté à la recherche de surfaces.
— Un parcours informé ou adaptatif, explorant en priorité les zones les plus susceptibles de contenir la surface du traducteur.

Si toutes ces stratégies peuvent théoriquement converger vers le résultat (à condition d'imposer une limite de subdivision pour le parcours en profondeur), le parcours informé de l'arbre des subdivisions permet, à condition de disposer d'un critère de priorité efficace, de converger plus rapidement que les deux autres parcours envisagés vers une zone de visibilité de la surface du traducteur depuis le point d'origine.

6.1.2 Algorithme de recherche de surface

Critères de classification

On définit une classification des pinceaux en 3 attributs, en fonction de l'état d'arrivée des rayons d'un pinceau :

1 Surface du traducteur intersectée.

2a Plan du traducteur couvert (tous les rayons sont arrivés sur le plan du traducteur).

2b Plan du traducteur intersecté (au moins un rayon est arrivé sur le plan du traducteur).

La figure 6.4 représente différents pinceaux arrivés différemment sur la surface du traducteur. Pour les pinceaux P_1, P_2, P_3 et P_4 représentés, on a :

— P_1 a l'attribut **2b**.
— P_2 a les attributs **2a** et **2b**.
— P_3 a les attributs **1**, **2a** et **2b**.
— P_4 a les attributs **1** et **2b**.

Critères de classification

Pour classifier un pinceau donné selon les attributs que nous avons définis, deux conditions sont testées dans l'ordre suivant :

1. On vérifie si au moins un des rayons a intersecté la surface du traducteur.
2. Dans le cas où tous les rayons ont atteint le plan du traducteur et où aucun des rayons n'a atteint la surface du traducteur, on vérifie si le quadrangle délimité par les points d'arrivée des quatre rayons intersecte la surface du traducteur.

Si l'une de ces conditions est vérifiée, le pinceau est considéré comme ayant atteint la surface du traducteur et les attributs **1** et **2b** sont appliqués. Puis, si l'ensemble des rayons a atteint le plan du traducteur, l'attribut **2a** est également appliqué et le pinceau peut être conservé pour la suite.

Si l'attribut **2a** n'est pas applicable, le pinceau ne peut être utilisé immédiatement et doit être subdivisé afin d'éventuellement en extraire un pinceau doté de l'attribut **2a**. De même, si un des attributs **1** et **2b** n'est pas applicable, le pinceau ne peut être utilisé et doit être subdivisé.

Critère de priorité

Pour chaque pinceau dont la classification a été déterminée, nous calculons un **score** déterminant la priorité de ce pinceau à être subdivisé. Cette priorité tient compte des critères suivants :

— L'état de classification du pinceau : si le pinceau atteint le traducteur, le score est maximal de façon à le traiter au plus vite.

— La distance du pinceau au traducteur lorsque celui-ci a intersecté le plan du traducteur : plus le pinceau est proche du traducteur, plus il est prioritaire. Cette distance est calculée comme la distance du rayon le plus proche des bords du traducteur.

— La profondeur du pinceau dans l'arbre de subdivision : dans l'objectif d'éviter une exploration trop hétérogène de l'arbre de subdivision, une profondeur dans l'arbre de subdivisions faible induit une priorité plus élevée. De cette façon, la profondeur d'exploration de l'arbre est homogénéisée et on favorise la découverte de plusieurs zones de visibilité.

En pratique, le score attribué aux pinceaux ayant atteint le plan du traducteur est inversement proportionnel au carré de la distance du pinceau au traducteur. Un facteur multiplicatif constant est ajouté pour favoriser ces pinceaux.

La valeur de ce facteur est ajustable et caractérise la tendance de l'algorithme à subdiviser des pinceaux n'ayant pas atteint le plan du traducteur par rapport à des pinceaux l'ayant atteinte à une distance importante. Nous l'avons fixée de sorte que des pinceaux situés à une distance de 1000 *mm* du traducteur aient la même priorité que des pinceaux n'ayant pas atteint le plan du traducteur.

⟨*calculScore*⟩≡

```
float score(Pinceau p) {
  float résultat = 0;

  if(p.traducteurAtteint()) {
    résultat = ∞;
  }
  else if(p.planTraducteurAtteint()) {
    float distance = p.distanceTraducteur();
    résultat = (1000 * 1000) / (distance * distance);
  }
```

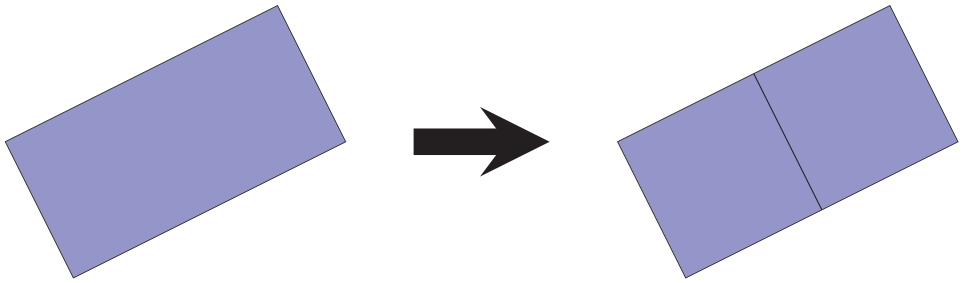


Figure 6.5 – Compensation de l'aplatissement d'un pinceau par subdivision selon l'axe le plus long

```
  else {
    résultat = 1;
  }
  résultat /= profondeur;

  return résultat;
}
```

Algorithme d'exploration

L'exploration avec priorités de l'arbre de subdivisions peut être effectuée au moyen d'une file de priorités dans laquelle les pinceaux sont ajoutés et qui est vidée au fur et à mesure de la progression de l'exploration. En pratique, en dehors de pinceaux ayant atteint le traducteur, une telle exploration ne se termine que lorsque chacune des branches de l'arbre de subdivisions a été explorée jusqu'à la limite de subdivision. Un critère d'arrêt de l'algorithme est donc nécessaire pour limiter la recherche.

Une fois qu'un pinceau intersectant le traducteur est trouvé, l'algorithme d'exploration change d'objectif et de phase : il s'agit alors de cerner la zone de visibilité du traducteur autour du pinceau trouvé. L'ensemble des pinceaux n'ayant pas intersecté le plan du traducteur est éliminé et une subdivision supplémentaire est autorisée pour les pinceaux ayant couvert le plan du traducteur. Les pinceaux ayant intersecté la surface du traducteur, mais possédant au moins un rayon hors du plan du traducteur sont subdivisés pour obtenir des sous-pinceaux couvrant le plan du traducteur.

Remarque : En procédant de la sorte, dans le cas où plusieurs zones de visibilité du traducteur existent, il est possible de ne pas détecter des zones placées plus en profondeur dans l'arbre des subdivisions. Nous faisons implicitement l'hypothèse que les zones plus en profondeur dans l'arbre ont une contribution négligeable devant celle des premières zones trouvées.

Dans le cas d'un pinceau dont l'ensemble des rayons a atteint le plan du traducteur, les dimensions du quadrangle formé par les quatre rayons du pinceau sur le plan du traducteur sont comparées et, dans le cas d'un pinceau aplati, on privilégie la subdivision selon l'axe permettant de compenser l'aplatissement du pinceau, comme schématisé en figure 6.5.

L'algorithme de recherche du traducteur renvoie une liste de pinceaux ayant atteint la surface du traducteur. Ses étapes sont les suivantes :

Initialisation

— Initialisation des 6 pinceaux de départ correspondant aux faces d'un cube sur la sphère unité.
— Calcul des 6 pinceaux.
— Ajout des 6 pinceaux calculés à la file de priorité.

Boucle principale

— Retrait du pinceau P en tête de la file de priorité (si elle n'est pas vide).
— Analyse du pinceau.

Pour la phase de recherche du traducteur avant qu'un premier pinceau n'ait intersecté le traducteur, la phase d'analyse est telle que suit :

— Si P vérifie les attributs **1**, **2a** et **2b**, stockage de P dans la liste résultat et passage de l'algorithme à la deuxième phase.
— Sinon, subdivision du pinceau P en deux (pour compenser l'aplatissement) ou en quatre et calcul des sous-pinceaux résultants puis ajout des sous-pinceaux à la file.

Pour la seconde phase, la boucle principale est identique, mais la phase d'analyse et de traitements change :

— Si P ne vérifie aucun attribut, il est ignoré.
— Si P vérifie les attributs **1**, **2a** et **2b**, stockage de P dans la liste de résultats.
— Si P vérifie les attributs **1** et **2a**, subdivision de P, ajout des sous-pinceaux à la file de priorité.
— Si P a été marqué pour élimination, il est alors ignoré.
— Sinon, subdivision de P, ajout des sous-pinceaux à la file de priorité et marquage des sous-pinceaux pour élimination.

Une fois que la file de priorité ne comporte plus aucun pinceau, l'algorithme s'arrête et renvoie la liste des pinceaux ayant atteint le traducteur. Si le traducteur n'est pas trouvé au-delà d'un nombre limite de pinceaux calculés [1], l'algorithme est arrêté et le point de champ sur lequel il a été exécuté n'est pas calculé.

6.2 Contrôle d'erreur

À partir des pinceaux obtenus à l'aide de l'algorithme de recherche de la surface du traducteur, nous construisons des interpolateurs selon la méthode d'interpolation polynomiale présentée dans le chapitre 4. Pour pouvoir être utilisés dans le processus de constitution des réponses impulsionnelles détaillé dans le chapitre 5, ces interpolateurs doivent approcher efficacement les valeurs des grandeurs de rayons.

6.2.1 Critère de précision

Une première étape consiste à définir un seuil d'acceptabilité en matière de précision permettant de décider de la qualité des interpolateurs d'un pinceau. Pour que l'erreur d'interpolation n'ait pas d'effet sur la qualité de la constitution de la réponse impulsionnelle, l'erreur commise en termes de temps de vol doit être inférieure au temps d'un échantillon temporel :

$$\delta t < \frac{1}{f_{ech}}$$

1. En pratique, nous avons fixé ce nombre à 50 000, qui est une limite suffisamment large pour les configurations que nous avons testées.

avec f_{ech} la fréquence d'échantillonnage du signal produit. On peut relier l'erreur de temps à l'erreur de position à l'aide de la lenteur s :

$$\delta t \approx s\delta x$$

Soit :

$$\delta x < \frac{1}{s.f_{ech}}$$

6.2.2 Test de précision d'un pinceau

Suivant ce critère, un test de validité des fonctions d'interpolation d'un pinceau consiste à choisir une direction $\vec{d}$ incluse dans le pinceau et à vérifier l'interpolation sur cette direction :

1. Lancer du rayon ultrasonore selon la direction $\vec{d}$ depuis le point d'origine du pinceau.
2. Interpolation selon cette direction de la position finale d'arrivée du rayon à l'aide de l'interpolateur de position du pinceau.
3. Comparaison de la distance entre l'estimation et la valeur obtenue par lancer de rayons avec le plafond d'erreur défini précédemment.

Si l'erreur commise par l'interpolation est inférieure au plafond, on peut estimer que l'interpolation respecte le critère de précision. Nous rappelons la majoration d'erreur proposée pour l'interpolation d'Hermite en section 4.4.3 :

$$\forall t \in I, |f(t) - P(t)| \leq \frac{\max_{t\in I} |f^{(n+2)}(t)|}{(n+1)!} \prod_{i=1}^{n} (t - x_i)^2$$

Bien qu'il ne soit pas possible de choisir *a priori* l'échantillon produisant l'erreur maximale sur le domaine d'interpolation, sachant que les interpolateurs que nous construisons utilisent les valeurs 0 et 1 pour les valeurs de $(x_i)_{1\leq i\leq n}$ et $(y_i)_{1\leq i\leq n}$, la majoration d'erreur la moins stricte a lieu sur l'échantillon maximisant la fonction :

$$(t,u) \rightarrow t^4(1-t)^4u^4(1-u)^4$$

Le couple de valeurs $(1/2, 1/2)$ vérifie cette condition. Ainsi, en effectuant l'échantillonnage sur ce couple de valeurs, nous faisons le choix de la majoration la moins stricte sur l'ensemble de l'espace d'interpolation.

Cela ne garantit néanmoins pas la stricte vérification du critère de précision sur l'ensemble du domaine d'interpolation. Une étude probabiliste pourrait nous permettre de mieux estimer l'efficacité de ce critère qui, dans le cadre de nos simulations, s'est avéré suffisant au vu de la régularité des fronts d'onde interpolés.

6.2.3 Application du test de précision

Ce test de précision est appliqué à chaque pinceau résultant de l'étape de recherche du traducteur. Les pinceaux utilisés lors de la constitution de réponse impulsionnelle vérifient la condition de précision. Ainsi, les pinceaux ne respectant pas cette condition ne peuvent directement servir à la suite des calculs.

Pour améliorer la qualité des interpolateurs, nous procédons à la subdivision systématique des pinceaux invalides et ce, jusqu'à n'obtenir que des pinceaux valides ou atteindre un niveau limite de subdivisions. Le processus appliqué utilise une file de pinceaux dans laquelle sont cumulés les pinceaux à contrôler et sont ajoutés les pinceaux issus de la subdivision des pinceaux invalides.

De cette façon, les pinceaux conservés vérifient tous le critère de précision et peuvent être utilisés pour la constitution de réponse impulsionnelle. Ils sont donc ajoutés à une liste qui est la donnée d'entrée de l'algorithme présenté dans le chapitre 5.

6.2.4 Subdivision informée

Lors de la subdivision des pinceaux invalides, nous pouvons profiter des interpolateurs de position pour effectuer une subdivision plus efficace du pinceau. En effet, la subdivision en deux ou quatre sous pinceaux égaux au niveau des directions de départ ne garantit pas l'équivalence des pinceaux à l'arrivée.

En utilisant l'interpolateur de positions, on peut estimer les directions de départ de rayons permettant une subdivision équitable du pinceau à l'arrivée. Il suffit pour cela d'effectuer une inversion d'interpolation pour la position centrale du pinceau à l'arrivée en suivant la même méthode que celle utilisée en section 5.3 pour l'algorithme de constitution de réponse impulsionnelle.

Cela permet d'éviter des subdivisions trop nombreuses autour de zones géométriques saillantes causant une variation brutale des directions des rayons, comme dans l'exemple en figure 6.6. Puisque cette approche est dépendante de la validité de l'interpolation, elle ne peut être utilisée lorsque le polynôme interpolateur des positions d'arrivée est mal conditionné ou commet des erreurs trop importantes.

Ainsi, la subdivision par inversion d'interpolation n'est utilisée que lorsque l'erreur commise par le polynôme est inférieure à une valeur inférieure à une fraction de sa taille d'arrivée (cette valeur a été fixée à $1/2$ de la taille du pinceau après des tests pratiques).

6.3 Conclusions et perspectives

L'algorithme de recherche de surface que nous avons développé et utilisé dans le cadre des simulations de champ ultrasonore permet en pratique d'obtenir une couverture de la surface du traducteur dans la majorité des configurations que nous avons testées.

Associé aux méthodes d'interpolation développées dans le chapitre 4, il utilise un mécanisme de subdivision informée des pinceaux de rayons permettant une convergence rapide autour de la surface du traducteur à partir d'un ensemble de pinceaux de départ couvrant la sphère unité. De plus, au cours du processus de subdivision, la précision des interpolateurs des pinceaux est vérifiée de manière à respecter une erreur en interpolation de temps de vol inférieure à un échantillon temporel et donc inobservable en pratique.

Cependant, le mécanisme de subdivision de pinceaux sur lequel il est fondé se base lui-même sur une hypothèse implicite de continuité des surfaces géométriques intersectées. Par son utilisation d'interpolateurs d'Hermite pour la subdivision des pinceaux ayant atteint le plan du traducteur, il s'avère d'autant plus efficace que la géométrie est lisse.

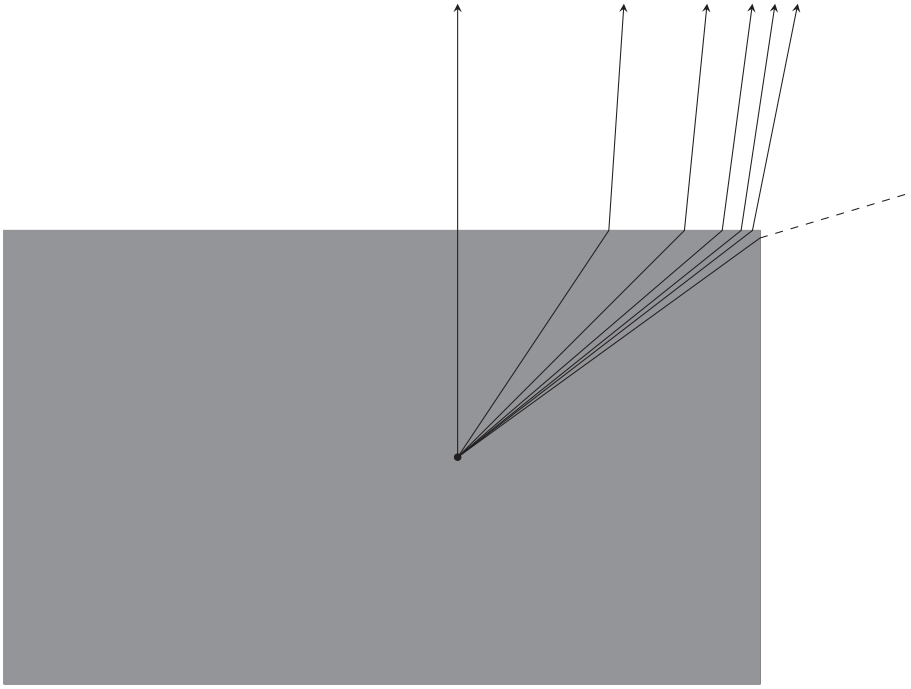


Figure 6.6 – Biais d'estimation de la subdivision en division régulière de pinceaux, causant de nombreuses subdivisions successives

Nous avons pu constater que l'algorithme a plus de difficultés à converger autour de la zone de visibilité de la surface du traducteur sur des maillages bruités. En particulier, il s'avère très sensible aux variations de normales. Ces cas représentent une limite intrinsèque de notre modèle de pinceaux : lorsque l'hypothèse de continuité de la dérivée n'est pas vérifiée, les interpolateurs échouent à convenablement approximer les pinceaux.

Notons par ailleurs que notre algorithme a peu de chances de réussir à détecter de petites discontinuités présentes sur un maillage puisqu'il exploite une approche purement basée sur du lancer de rayons. L'utilisation de méthodes d'intersection volumique des pinceaux avec la géométrie de la pièce pourrait corriger ce problème.

Enfin, le test de précision des interpolateurs des pinceaux, même s'il s'avère efficace dans de nombreux cas, ne garantit pas une majoration de l'erreur d'estimation des temps de vol et positions d'arrivée des rayons sur l'ensemble du domaine d'interpolation. Une étude probabiliste du problème pourrait permettre de déterminer si l'approche que nous avons choisie peut théoriquement être améliorée.

CHAPITRE 7

Optimisations architecturales

Sommaire

Dans ce chapitre, nous détaillons l'ensemble des optimisations de code effectuées afin d'exploiter au mieux les processeurs généralistes modernes. Après avoir présenté les spécificités des architectures visées par nos outils de simulation de champ ultrasonore, nous présentons les améliorations apportées au code et leur impact sur les performances globales du calcul.

Dans un premier temps, nous étudions l'efficacité du parallélisme de tâches sur différents CPU multicœurs, avant de procéder à un profilage du code afin de mettre en évidence les fonctions les plus calculatoires, sur lesquelles il est nécessaire de procéder à des optimisations. Nous terminons ce chapitre en détaillant les optimisations réalisées et en mesurant leur impact sur les performances globales du calcul.

7.1 Description des architectures visées

7.1.1 Processeur généraliste

Le CPU, ou processeur généraliste, est le circuit responsable de l'exécution des instructions d'un programme informatique sur un ordinateur, qu'elles soient arithmétiques, logiques, ou d'entrées/sorties. La plupart des processeurs modernes suivent le modèle d'architecture dit de Von Neumann, utilisant une structure de stockage unique pour les instructions et les données à traiter. Le modèle du CPU le plus simple comporte les éléments suivants :

L'unité de contrôle chargée de récupérer les instructions puis de les transmettre aux unités chargées de leur exécution.

L'unité de décodage dont le rôle est de décoder les instructions.

Les registres mémoire interne du processeur à très faible latence et très forte bande passante, mais de faible capacité.

L'unité arithmétique et logique qui exécute les instructions d'arithmétique et de logique sur les nombres entiers.

L'unité de calcul flottant effectuant les opérations arithmétiques sur les nombres à virgule flottante.

L'interface mémoire qui fait le lien entre les registres et la mémoire RAM de l'ordinateur.

La figure 7.1 résume les interactions entre ces différents éléments.

Un CPU fonctionne de manière synchrone à une fréquence fixée. Le nombre d'instructions qu'il peut exécuter en une seconde est proportionnel à cette fréquence. Pour une même fréquence, ce nombre peut varier d'un processeur à l'autre en fonction du nombre d'unités de traitement du processeur et de l'ordonnancement des instructions.

De plus, il est relié à une mémoire RAM fonctionnant également de manière synchrone à une fréquence qui lui est propre. Les instructions et les données à traiter par le processeur transitent par le biais d'un bus de largeur fixée. La largeur du bus et la fréquence de fonctionnement de la RAM impactent directement la vitesse de transfert des informations de la mémoire au processeur.

7.1.2 Vers plus de performances

Depuis le début des années 60, l'industrie de l'informatique connaît une croissance exponentielle, suivant la loi empirique de Moore (voir Moore (1995)) : tous

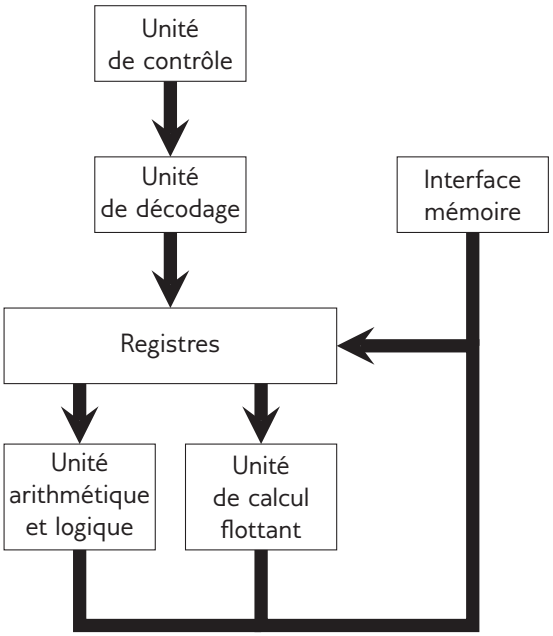


Figure 7.1 – Diagramme fonctionnel simplifié d'un CPU

les deux ans, le nombre de transistors sur les puces de silicium composant les ordinateurs est doublé. Cet effet a permis une augmentation également exponentielle des performances des processeurs et des mémoires.

Jusqu'au début des années 2000, cette augmentation était principalement due à l'augmentation de la fréquence de fonctionnement des puces, passant de 740 kHz pour le premier microprocesseur vendu (Intel 4004) en 1971 à près de 4 gigahertz en 2004. Cependant, les problèmes de surchauffe occasionnés par l'utilisation de fréquences de fonctionnement élevées, combinées à des limites physiques liées notamment à la vitesse de propagation d'un signal électrique sur une puce, ont empêché la poursuite de cette croissance en fréquence.

Dès lors, des efforts ont été mis en œuvre pour multiplier le nombre de cœurs de calcul par processeur, afin de permettre l'exécution simultanée d'instructions multiples. Un cœur de calcul comprend l'ensemble des unités précédemment mentionnées ainsi qu'une mémoire cache rapide. Les cœurs de calcul ont en commun deux niveaux de mémoire cache ainsi que l'accès à la mémoire RAM.

De plus, un second niveau de parallélisme, le mode d'instructions SIMD, mis en œuvre dès la fin des années 90 par le biais du jeu d'instructions MMX sur les *Pentium MMX* d'*Intel*, permet d'augmenter les performances de calcul. Il consiste à exécuter simultanément des instructions identiques sur des données différentes. Ainsi, dans le cas où un même traitement est appliqué à un grand nombre d'informations différentes, il permet d'exécuter en une passe une instruction sur un vecteur plutôt que sur un scalaire.

Apparu en 2008, le jeu d'instructions AVX permet de travailler sur des registres d'une largeur de 256 bits, soit 8 flottants en simple précision. Le jeu d'instructions AVX-512, disponible pour le moment sur les coprocesseurs Intel MIC, dispose quant à lui d'une largeur de 512 bits.

7.1.3 Exploitation des architectures parallèles

Lors de la phase de croissance en fréquence des processeurs, le simple fait d'exécuter un programme sur un processeur de fréquence plus élevée suffisait à bénéficier d'au moins une partie des gains de performance. En effet, la simple augmentation de la cadence de traitement des informations raccourcit le temps d'exécution d'un programme sans modifications de celui-ci.

À l'inverse, pour bénéficier d'une accélération en exécutant un code sur plusieurs cœurs de calcul, un programme doit comporter des sections pouvant être exécutées en parallèle. Deux sections de code peuvent être efficacement exécutées en parallèle si les conditions suivantes sont remplies :

1. Elles ne doivent pas avoir de relations de dépendance.
2. Elles ne doivent pas nécessiter de verrouillage de mêmes ressources (mémoire, disque dur, périphériques...).
3. Les accès concurrents à la mémoire ne doivent pas modifier le résultat de l'exécution du programme.

En particulier, les situations de type *lecture après écriture*, *écriture après écriture* ou *écriture après lecture* présentent des dangers potentiels de comportements imprévisibles du programme et peuvent nécessiter la mise en place de mécanismes de verrouillage sur les données écrites/lues.

D'autre part, pour exploiter efficacement les unités de traitement vectoriel des processeurs modernes, il faut disposer de suffisamment de données sur lesquelles un traitement identique peut être effectué simultanément. Cela impose une stricte absence de dépendance entre les traitements et l'alignement des données en mémoire pour exploiter les instructions SIMD.

Ainsi, même si les performances des processeurs continuent de croître, il est désormais nécessaire d'adapter l'implémentation des algorithmes aux architectures. Il est même nécessaire, dans certains cas, de repenser les algorithmes utilisés ou la manière de représenter et de stocker les données en mémoire pour faciliter leur parallélisation, qu'il s'agisse de bénéficier de l'accélération d'un processeur multicœur ou des unités vectorielles.

7.2 Méthodes et solutions de parallélisme

7.2.1 Parallélisme de tâches

L'exploitation du parallélisme de tâches requiert la conception d'algorithmes pouvant être décomposés en tâches indépendante et, idéalement, travaillant sur des données disjointes. Un tel algorithme est qualifié d'*embarassingly parallel*. L'algorithme de lancer de rayons géométrique proposé par *Embree* (voir la section 3.1.2) en est un exemple : chaque opération de lancer de rayon est strictement indépendante des autres et les seules données partagées (les données de la géométrie) ne sont que lues.

Des outils, le plus souvent distribués sous forme de bibliothèques, permettent d'exploiter le parallélisme de tâches sur un processeur multicœur. Parmi ceux-ci, on peut citer les outils usuels suivants :

— Les implémentations des *threads* système selon le standard POSIX, *pthreads*.
— Les directives de préprocesseur *OpenMP*, permettant de paralléliser une boucle au prix d'une simple annotation de code source.

— Les bibliothèques *MIT Cilk*, *Intel Cilk Plus* et *Cilk++* qui permettent, au moyen de quelques mots-clés ajoutés au langage C (C++ pour *Intel Cilk Plus* et *Cilk++*), de spécifier des zones de code exécutées de manière asynchrone (mot-clé `cilk_spawn`), des boucles parallélisées (mot-clé `cilk_for`) et les points de synchronisation des zones parallèles (mot-clé `cilk_wait`).
— La bibliothèque *Intel Thread Building Blocks* qui fournit des méthodes de haut niveau pour appliquer des schémas standard de parallélisation tels que le `parallel_for` ou les *pipelines* parallèles.

pthreads fournit la gestion du parallélisme de niveau le plus bas et se base sur l'ordonnanceur système pour l'allocation de temps processeur à chacun des *threads*. L'ensemble des mécanismes de synchronisation et de partage des ressources doit être géré lors de l'implémentation d'un algorithme parallèle. L'utilisation de *pthreads* est donc particulièrement intrusive et s'adresse à des cas spécifiques où l'utilisateur souhaite une maîtrise complète des *threads* utilisés par son programme.

Le mode de parallélisation proposé par *OpenMP* est au contraire très peu intrusif puisqu'il ne change en principe que très peu le code et peut-être désactivé simplement. En particulier, pour paralléliser une boucle *for*, une seule ligne de directives pré-processeur suffit généralement. Cependant, pour des schémas de parallélisme plus complexes, l'implémentation avec *OpenMP* peut s'avérer plus intrusive et moins efficace.

ITBB et *Intel Cilk Plus* (ainsi qu'*OpenMP* depuis la version 3.0) utilisent le mécanisme de *work-stealing* : un nombre de *threads* égal au nombre de cœurs logiques du processeur sont créés et chacun se voit attribuer une file de tâches élémentaires. Lorsque l'un des *threads* a épuisé sa file de tâches, il retire une tâche de la file d'un autre *thread* dont l'exécution est plus longue. Ainsi, la charge est naturellement répartie entre les *threads*.

McCool et collab. (2012) proposent un ensemble de modèles standard d'algorithmes parallèles (tels que le *pipeline* parallèle, le *map*, la réduction...) ainsi que leurs cas d'application et leur efficacité. Des implémentations utilisant ITBB, *Intel Cilk Plus* et *OpenMP* sont proposées.

Nous utilisons ITBB pour la parallélisation des codes de calcul de champ. Comme nous le verrons par la suite, le problème du calcul de champ peut-être ramené à un algorithme parallèle de type *embarassingly parallel* et donc bénéficier aisément d'un bon facteur d'accélération, que ce soit avec *OpenMP* ou avec ITBB. Cependant, dans le cas des problématiques de visualisation interactive abordées dans la partie IV, les schémas de parallélisme utilisés, plus complexes, peuvent être plus efficacement implémentés à l'aide d'ITBB.

7.2.2 Parallélisme de données

Ordres de traitement

Pour être exploité efficacement dans un algorithme, le parallélisme de données impose d'effectuer simultanément des traitements identiques sur des paquets de données alignés et contigus en mémoire. Cette condition peut s'avérer très intrusive au point de nécessiter la réorganisation d'un algorithme afin de favoriser l'exploitation des instructions SIMD.

Un exemple typique est le passage d'un code dit « vertical » à un code dit « horizontal ». Considérons un algorithme constitué de trois tâches T_1, T_2 et T_3 formant une chaîne de dépendances (le résultat final de l'algorithme est donc $T_3(T_2(T_1(x)))$ et

appliquées séquentiellement à un ensemble de données d'entrée $(x_i)_{1 \leq i \leq n}$. Plusieurs modèles d'exécution sont envisageables, parmi lesquels les suivants :

— Une exécution verticale : pour chaque élément, on exécute successivement les trois tâches T_1, T_2 et T_3. On ne conserve donc en mémoire que le résultat du dernier traitement appliqué.
— Une exécution horizontale : on commence par exécuter T_1 sur toutes les données d'entrée et on stocke chaque résultat. Puis, on exécute sur ces résultats la tâche T_2 en stockant les résultats, avant d'exécuter la tâche T_3.
— Un mode d'exécution mixte : on sépare le jeu de données d'entrée en paquets de taille fixée, sur lesquels on applique une exécution verticale des trois tâches.

Le mode d'exécution verticale, s'il minimise l'empreinte mémoire de l'algorithme, empêche un traitement vectoriel des données. Le mode d'exécution horizontale favorise l'utilisation d'instructions SIMD mais a une empreinte mémoire importante. Un bon compromis consiste à constituer des paquets de taille suffisamment grande pour exploiter efficacement les instructions SIMD avec la bonne largeur de registre, tout en limitant l'empreinte mémoire.

Organisation de la mémoire

Les instructions SIMD sont appliquées sur des registres de largeur fixée par le jeu d'instructions utilisé. Ainsi, avec le jeu d'instructions SSE, on peut traiter simultanément 4 flottants en simple précision ou 2 flottants en double précision puisque la largeur du registre est de 128 bits. Le jeu d'instructions AVX double cette capacité et le jeu d'instructions AVX-512 la quadruple.

Pour qu'une instruction SIMD puisse être appliquée, il est nécessaire de réunir l'ensemble des données d'entrée dans un **registre** SIMD. Pour ce faire, des instructions dédiées au chargement sont proposées dans les jeux d'instructions SIMD usuels. Jusqu'au jeu d'instructions AVX, les instructions de chargement disponibles fonctionnent uniquement sur des données **groupées** : les données d'un même registre doivent être contigües en mémoire. Si elles ne le sont pas, il est nécessaire d'effectuer une opération de rassemblement dite « *gather* ».

Bien que les derniers jeux d'instructions SIMD en date (AVX2 et AVX-512) proposent des instructions de *gather* nativement, les performances du chargement aligné restent meilleures. Ainsi, pour pleinement bénéficier de l'accélération liée à l'utilisation des unités vectorielles d'un processeur, il est préférable de stocker les données de manière groupée en privilégiant par exemple les structures de tableaux aux tableaux de structures.

Outils de vectorisation

Les solutions logicielles pour faciliter l'exploitation des unités SIMD sont nombreuses et se basent sur des approches variées. On peut cependant distinguer les catégories suivantes :

— Les approches de bas niveau, telles que l'utilisation de fonctions intrinsèques ou de code assembleur. Elles nécessitent l'écriture de code spécifique à chaque architecture ciblée, sont difficilement maintenables, mais offrent les meilleurs résultats de performance.
— La catégorie des approches « vectorielles », qui regroupe les solutions logicielles proposant d'écrire un code source utilisant des structures de vecteurs

et en surchargeant les opérateurs arithmétiques usuels pour ces structures. Le choix des instructions SIMD à utiliser est effectué à la compilation. Elles sont moins intrusives et plus maintenables que les approches de bas niveau tout en étant évolutives, mais requièrent cependant une modification du code assez profonde. *Boost SIMD* (voir Estérie et collab. (2014)) et la *Vector Class Library* (voir Fog (2016)) sont deux solutions de ce type.

— Les approches semi-automatiques, telles que la vectorisation automatique des compilateurs guidée par des directives pré-processeur, *OpenMP 4.0* ou encore le compilateur *Intel SPMD Program Compiler*, proposent d'annoter les parties que l'utilisateur estime susceptibles de bénéficier de la vectorisation. Des traitements sont alors effectués à la compilation afin de transformer un code scalaire en code vectoriel. Ces annotations sont typiquement placées sur des boucles.

— Les compilateurs modernes disposent d'outils de vectorisation automatique exécutés à la compilation, détectant les boucles et les zones de code susceptibles d'être vectorisées à partir d'une analyse du code source.

Lambert (2015) (chapitre 3) propose un comparatif détaillé de ces approches de vectorisation en prenant en compte les critères de performance, de facilité d'utilisation, de maintenabilité et d'industrialisation.

Quelle que soit l'approche, il convient de noter que la vectorisation ne peut être efficace que si le code ciblé effectue de manière répétée un traitement identique et sans branchements conditionnels sur des données différentes. Les cas idéaux de vectorisation réalisent un grand nombre d'opérations arithmétiques sur de larges plages de données contigües en mémoire sans branchements conditionnels.

Le choix de la méthode de vectorisation à utiliser fait intervenir plusieurs critères :

— Complexité du code à vectoriser : les outils de vectorisation automatique s'avèrent peu efficaces sur des codes complexes faisant intervenir, notamment, des branchements conditionnels.

— Portabilité : en utilisant les instructions SIMD intrinsèques ou du code assembleur, le code écrit n'a aucune portabilité dans la mesure où il cible un jeu d'instructions et une largeur de vecteur spécifiques. À l'inverse, les outils intermédiaires tels que la *Vector Class Library* ou les outils de vectorisation automatique sont prévus pour adapter les instructions utilisées à une architecture cible.

— Maintenabilité : plus les modifications apportées au code vectorisé sont intrusives, plus la maintenabilité et l'industrialisation sont compliquées. En particulier, l'utilisation d'instructions intrinsèques ou de code assembleur produit un code particulièrement complexe à relire et donc à maintenir.

— Performance : les meilleures performances sont obtenues en adaptant le code à une architecture cible à l'aide des instructions intrinsèques ou de code assembleur.

Il s'agit donc de faire un compromis entre tous ces critères en fonction de l'application visée. Dans notre cas d'utilisation, les codes de calcul de champ ont pour vocation d'être intégrés à la plate-forme logicielle *CIVA*. Des impératifs d'industrialisation, même s'ils ne sont pas cruciaux à ce stade des travaux, sont donc à prendre en compte. Nous privilégions donc dans la suite les outils *OpenMP 4.0* et *Vector Class Library*.

7.3 Une première mesure de performances

Afin de guider les opérations d'optimisation visant à mieux exploiter les capacités de calcul des CPU, nous procédons à une série de mesures de performances du code séquentiel et scalaire afin d'identifier les portions les plus coûteuses en temps de calcul.

Nous effectuons pour cela du **profilage de code**, ce qui nous permet :

— De mesurer le temps passé dans chaque fonction et donc de connaître celles qu'il convient d'optimiser.
— De remarquer d'éventuels points d'engorgement du calcul.
— De mesurer l'utilisation du processeur et l'efficacité du parallélisme.
— D'identifier les causes d'éventuels problèmes de parallélisme.

7.3.1 Configurations de test

Le calcul de champ ultrasonore se décompose en trois phases principales : la recherche du traducteur par lancer de rayons (voir le chapitre 6), le calcul de réponse impulsionnelle (voir les chapitres 4 et 5) et la convolution du signal. Nous cherchons ici à optimiser les deux premières étapes de ce calcul, la troisième ayant fait l'objet d'un travail d'optimisation par Lambert (2015) que nous réutilisons.

Selon les configurations de simulation de champ ultrasonore, la distribution de temps entre ces phases varie. Ainsi, dans le cas d'une pièce dotée d'une géométrie complexe et d'un traducteur peu étendu et simple, la phase de recherche du traducteur prend plus de temps que la phase de constitution de réponse impulsionnelle. À l'inverse dans le cas d'un traducteur étendu et composé de nombreux éléments, mais d'une pièce à géométrie simple, la phase de recherche du traducteur est plus rapide, tandis que la phase de constitution de réponse impulsionnelle est plus coûteuse.

Afin de mettre à l'épreuve les capacités de nos algorithmes de calcul de champ, nous utilisons pour ces mesures de performances préalables à l'optimisation trois configurations de champ principales permettant d'illustrer la variété des distributions de temps de calcul. Les figures 7.2, 7.3 et 7.4 et la table 7.1 présentent les caractéristiques de ces configurations.

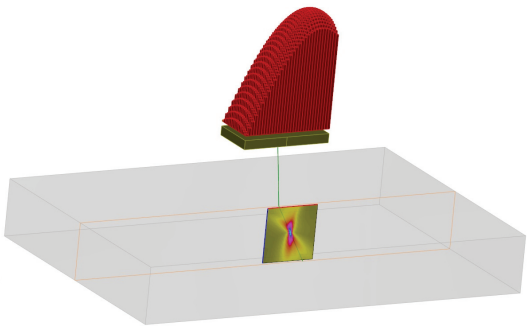


Figure 7.2 – Configuration de contrôle n° 1 utilisée pour le profilage

7.3.2 Résultats du profilage

La table 7.2 montre que la proportion de temps passé dans chacune des trois étapes principales du calcul de champ varie. On constate que malgré cela, chacune

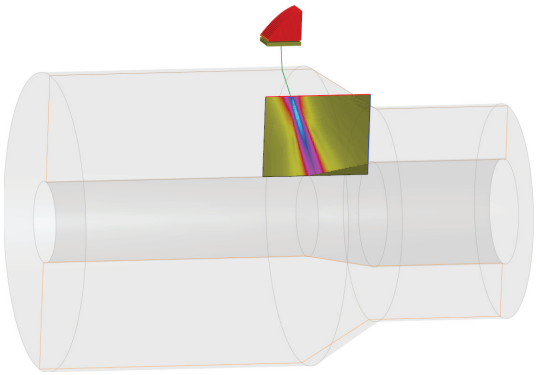


Figure 7.3 – Configuration de contrôle n° 2 utilisée pour le profilage

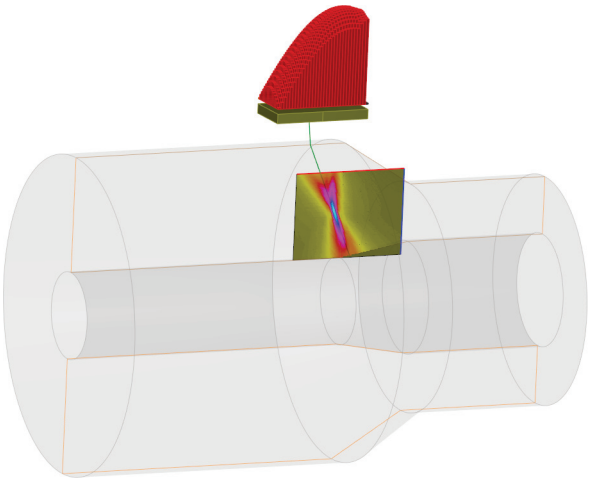


Figure 7.4 – Configuration de contrôle n° 3 utilisée pour le profilage

des trois étapes constitue un temps de calcul non négligeable. Par conséquent, l'amélioration des performances du calcul de champ passe nécessairement par l'optimisation de ces trois étapes.

La première configuration met en jeu une pièce à géométrie simple, la phase de recherche du traducteur et raffinement des pinceaux est rapide. À l'inverse, le traducteur est étendu et comporte de nombreux éléments, ce qui rend les étapes d'interpolation des rayons et de sommation des réponses impulsionnelles plus coûteuses.

Dans la seconde configuration, à l'inverse, le traducteur est plus petit tandis que la géométrie est plus complexe et comporte de nombreuses primitives. Ainsi, la phase de recherche du traducteur et de raffinement des pinceaux est plus coûteuse relativement au temps de calcul total et la phase de constitution de réponse impulsionnelle se fait plus rapidement.

La troisième configuration est un mélange des deux autres configurations : elle allie la géométrie complexe de la configuration 2 au traducteur étendu de la configuration 1. On retrouve donc une distribution des temps de calcul intermédiaire.

Configuration	1	2	3
Pièce	Plane	Tuyau	Tuyau
Triangles	24	1 100 000	1 100 000
Fréquence du signal	2.5 MHz	2.5 MHz	2.5 MHz
Traducteur	512 éléments	1 élément	512 éléments
Zone	10201 points	10201 points	10201 points

Table 7.1 – Configurations de simulation de champ ultrasonore utilisées pour le profilage de code

Configuration	1	2	3
Lancer de rayons	39.9 %	47.4 %	46.3%
Localisation des échantillons	7.2 %	7.2 %	8.6%
Interpolations	39.2 %	24.6 %	34.4%
Sommations	9 %	10.1 %	7.9%
Signal	3.2 %	4.4 %	2.5%
Autres	1.5 %	4.7 %	0.3%

Table 7.2 – Distribution des temps de calcul entre les différentes étapes du calcul de champ sur trois configurations

7.4 Exploitation du parallélisme de tâches

Pour améliorer chacune de ces trois étapes simultanément, une première optimisation consiste à exploiter le parallélisme de tâches, et donc à exécuter plusieurs calculs en parallèle sur chacun des cœurs du CPU. Dans le cadre de notre méthode de calcul de champ, tous les points sont calculés indépendamment et les écritures en mémoire se font sur des zones distinctes d'un point à l'autre, à l'exception du calcul d'amplitude maximale à l'échelle de l'image de champ.

Ainsi, effectuer en parallèle le calcul des différents points de champ est possible sans utiliser d'exclusions mutuelles ni d'instructions atomiques. En utilisant le modèle standard de `parallel_for` inclus dans la bibliothèque ITBB, nous implémentons une version parallèle du calcul de champ ultrasonore.

Pour mesurer son efficacité, nous procédons à des mesures de performances sur un processeur multicœur en faisant varier le nombre de cœurs utilisés. La table 7.3 recense les processeurs utilisés dans le cadre de ces mesures de performances.

Configuration	CPU	Instructions	Architecture
1	Xeon E5-1650v2 (6 cœurs)	AVX	*Ivy Bridge*
2	Xeon E5-2630v3 (2 × 8 cœurs)	AVX2	*Haswell*
3	Xeon E5-2697v2 (2 × 12 cœurs)	AVX	*Ivy Bridge*

Table 7.3 – Configuration matérielle des machines utilisées pour les tests de performance

Dans l'ensemble des tests, l'*HyperThreading* a été désactivé. Cette fonctionnalité fait cohabiter deux *threads* simultanément sur un même cœur de CPU pour tirer parti des défauts de cache d'un programme en exécutant les instructions d'un *thread*

lorsque l'autre *thread* alloué à la puce est en attente d'opérations mémoire et inversement. L'apport de cette fonctionnalité sur nos codes de simulation s'est avéré faible (voire sensiblement négatif à partir d'une vingtaine de *threads*).

Par ailleurs, cette fonctionnalité purement matérielle a des effets variables dépendant de l'ordonnancement, des facteurs limitants du code (accès mémoire ou calculs), de la bande passante de la mémoire... Ainsi, Lambert (2015) met en évidence des gains non négligeables liés à l'*HyperThreading* pour des algorithmes de simulation de champ ultrasonore sur *Intel Xeon E5-2697v2* (2×12 cœurs). Kinghorn (2014) met quant à lui en garde contre les effets potentiellement négatifs de l'*HyperThreading* avec des résultats de performance sur *Intel Xeon E5-4624Lv2* (4×10 cœurs).

La démarche que nous adoptons concernant l'*HyperThreading* consiste à développer les codes de calcul, mesurer leurs performances et les optimiser en désactivant cette fonctionnalité, puis à mesurer son impact sur ces codes pour décider de son utilisation.

Nous reprenons la configuration de contrôle 3 des tests présentés en section 7.3.2 et faisons varier le nombre de *threads* actifs pour le même calcul de champ sans autres optimisations architecturales que celles d'*Embree* et le parallélisme de tâches sur différents processeurs. Les résultats de ces tests sont présentés en figures 7.5, 7.6 et 7.7.

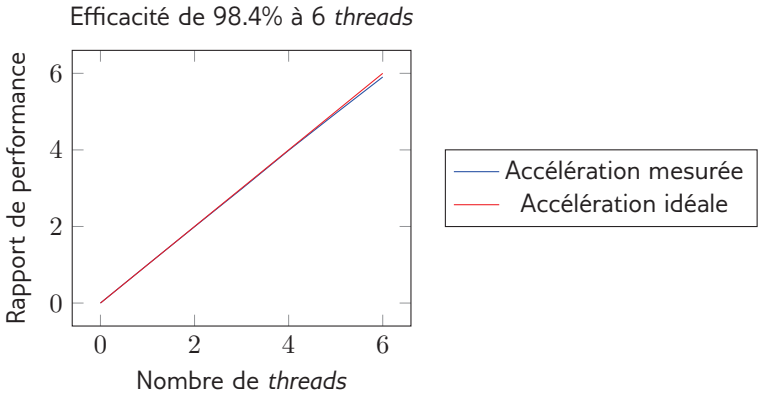


Figure 7.5 – Accélération du calcul de champ grâce au parallélisme de tâches sur un processeur Intel Xeon E5-1650v2

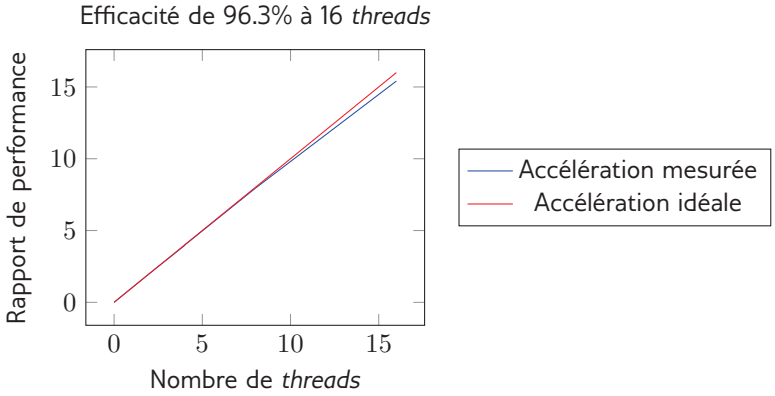


Figure 7.6 – Accélération du calcul de champ grâce au parallélisme de tâches sur deux processeurs Intel Xeon E5-2630v3

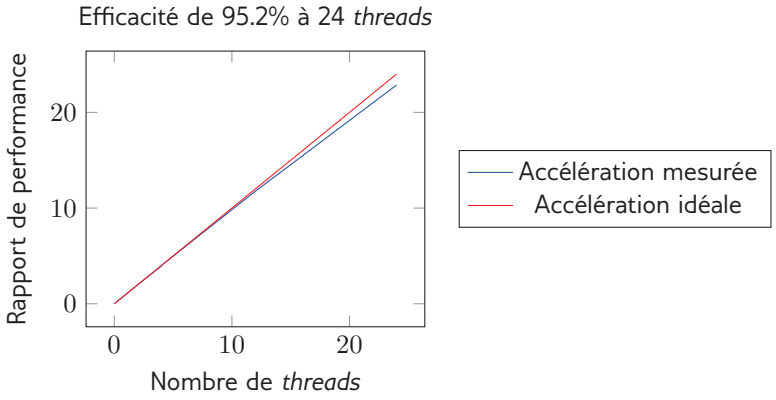


Figure 7.7 – Accélération du calcul de champ grâce au parallélisme de tâches sur deux processeurs Intel Xeon E5-2697v2

Nous constatons une bonne efficacité de la parallélisation du code par tâches sur les trois processeurs utilisés. Nous notons cependant une diminution sensible de cette efficacité avec l'augmentation du nombre de cœurs exploités. Puisque les cœurs du CPU partagent le même bus mémoire, l'augmentation du nombre de traitements en parallèle augmente les besoins en termes de lecture et d'écriture de données en mémoire, ce qui explique la baisse d'efficacité observée.

L'utilisation de l'*HyperThreading* peut aider à camoufler ce problème en permettant l'exécution de deux *threads* simultanément sur un même cœur. Ainsi, lorsqu'un *thread* est en attente de lectures ou d'écritures en mémoire, l'autre peut exécuter des calculs sur des données déjà chargées.

7.5 Optimisation de la phase de lancer de pinceaux

La phase de lancer de pinceaux consiste principalement en des opérations de lancer de rayons ultrasonores. Ainsi, les opérations principales effectuées sont le calcul des intersections de rayons avec la scène de calcul et des grandeurs physiques associées aux rayons.

Le nombre de rayons lancés au cours de cette phase varie fortement d'un point de champ à l'autre : pour une même configuration de champ, le traducteur peut être plus difficilement visible depuis certains points de champ et le contrôle de précision des interpolateurs des pinceaux peut requérir plus de raffinements. Ainsi, le nombre de rayons requis pour un point de champ, en plus d'être inconnu à l'avance, n'est pas constant d'un point de champ à l'autre.

Il n'est donc pas possible de pré-allouer un espace mémoire de taille fixe pour l'ensemble de rayons lancés. Des allocations dynamiques auraient pour effet de considérablement réduire l'efficacité du parallélisme et les performances globales du calcul de champ. L'ensemble des allocations d'espace mémoire pour les résultats de lancer de rayons est donc effectué au moyen d'un allocateur spécifique issu de la bibliothèque ITBB, le `scalable_allocator` (voir Intel Corporation (2016b)). Celui-ci permet de limiter le coût d'allocations trop fréquentes dans un code de calcul intensif en groupant les allocations de mémoire par paquets et en évitant la sérialisation des appels d'allocation de mémoire entre les *threads*. De plus, de cette façon, la mémoire allouée est réutilisée d'un point de champ à l'autre et désallouée uniquement à la fin du calcul.

Le travail d'optimisation algorithmique présenté dans le chapitre 6, combiné aux optimisations architecturales et algorithmiques du processus de lancer de rayons apportées par la bibliothèque *Embree*, permet d'améliorer les performances de cette première étape du calcul de champ.

Le lancer de rayons par paquets n'est utilisé que lors des phases de calcul de dérivée par différences finies sur les grandeurs des rayons ultrasonores (voir l'annexe C). En effet, en dehors de ces conditions, la divergence des rayons empêche une utilisation efficace du lancer de rayons par paquets. L'utilisation des structures vectorielles fournies par la *Vector Class Library* lors du lancer de rayons par paquets permet de vectoriser les étapes de calcul physique en plus de celle de lancer de rayons géométrique.

7.6 Optimisation de la phase de constitution des réponses impulsionnelles

7.6.1 Décomposition en étapes

La phase de constitution de réponse impulsionnelle, telle que détaillée dans le chapitre 5, est une phase particulièrement calculatoire. Nous en rappelons les étapes principales, pour chaque point de champ :

1. Construction d'une structure accélératrice de type BVH pour les pinceaux recouvrant la surface du traducteur.
2. Pour chaque échantillon du traducteur, détermination du pinceau le recouvrant au moyen d'un parcours de la structure accélératrice.
3. Pour chaque échantillon du traducteur, inversion de l'interpolation de la projection sur la surface du traducteur du point d'arrivée du rayon ultrasonore pour obtenir des coordonnées locales au pinceau recouvrant. Puis, interpolation des quantités utiles à la constitution de la réponse impulsionnelle par interpolations.

L'utilisation d'une BVH en deux dimensions (voir la section 5.3.2) pour la détermination du pinceau recouvrant un échantillon permet de réduire la complexité de cette recherche, passant d'une complexité linéaire par rapport au nombre total de pinceaux recouvrant la surface du traducteur à une complexité logarithmique.

L'opération d'exploration de la structure accélératrice présente peu de régularité puisqu'il s'agit d'une exploration d'arbre. À chaque niveau de l'arbre, des branchements conditionnels sont opérés, limitant les possibilités de vectorisation de cette étape.

Des mécanismes de vectorisation similaires à ceux utilisés dans la bibliothèque *Embree* pour l'exploration des structures accélératrices de géométrie pourraient être utilisés pour améliorer les performances de cette étape. Dans le cadre de nos travaux, nous n'avons pas réalisé cette optimisation en raison d'un gain potentiel faible en comparaison avec le temps de développement requis.

L'opération 3 de calcul des grandeurs de rayons associées à chaque échantillon du découpage du traducteur est quant à elle très régulière. En effet, même si la première phase de résolution polynomiale pour l'inversion d'interpolation présente des branchements conditionnels, Lambert (2015) a montré (chapitre 5) qu'il est possible d'obtenir des accélérations intéressantes en groupant des descentes de Newton par paquets.

Une fois cette inversion effectuée, les opérations d'interpolation des grandeurs de rayons sont purement calculatoires et ne présentent aucun branchement conditionnel autre que la vérification de la validité de l'inversion d'interpolation. La vectorisation, à ce stade, peut donc apporter une accélération particulièrement intéressante.

7.6.2 Adaptation de l'algorithme

Pour bénéficier de l'accélération liée à l'utilisation d'instructions SIMD, les données sur lesquelles les opérations de calcul sont exécutées doivent être alignées et contigües en mémoire. En effet, les jeux d'instructions vectorielles actuels n'incluent pas de chargements de données dispersées arbitrairement en mémoire.

Considérons les étapes à vectoriser de l'algorithme de calcul des échantillons de la surface du traducteur :

1. Inversion de l'interpolation de la projection sur le traducteur de la position finale du rayon.
2. Interpolation des quantités physiques associées au rayon.

Pour la première étape, les données d'entrée sont l'interpolateur de projection du pinceau recouvrant l'échantillon et les coordonnées de l'échantillon sur la surface du traducteur. La structure de données contenant les coordonnées des échantillons du traducteur peut être organisée de sorte que les coordonnées puissent être chargées directement par paquets. On privilégie dans ce cas le modèle dit de *structure of arrays* : plutôt que de stocker les données sous forme d'un tableau de paire de coordonnées, on stocke deux tableaux de coordonnées. La dimension de ces tableaux est un multiple de la largeur des vecteurs utilisés. Ainsi, pour exploiter les instructions AVX, on alloue des tableaux de dimension $8n$ avec n un entier.

Pour les interpolateurs de position, puisque l'indice du pinceau recouvrant n'est pas connu à l'avance, on ne peut aligner les données de manière à ce que les coefficients des polynômes interpolateurs soient directement chargés. Il nous faut donc allouer un espace mémoire où les coefficients des polynômes sont copiés avant le traitement. Pour chaque interpolateur de degré 3, on alloue 16 tableaux de largeur $8n$ correspondant à chacun des coefficients du polynôme interpolateur d'Hermite utilisé. En pratique, pour l'interpolateur de projection des positions finales des rayons, on alloue deux tableaux de 16 flottants en simple précision.

De la même manière, pour la seconde étape, on rassemble toutes les données d'entrée utilisées pour les calculs d'interpolation dans des tableaux de dimension $8n$. Les données intermédiaires (en particulier les coordonnées locales au pinceau) et les données de sortie sont aussi stockées dans des tableaux de dimension $8n$. À la fin du traitement, une opération de *scatter* (ou dispersion) des données est réalisée afin de les stocker dans les structures utilisées dans la suite du calcul.

Enfin, l'ensemble des traitements à vectoriser est regroupé dans une boucle de taille fixée et connue à la compilation, de sorte que le compilateur puisse en générer une version vectorisée.

7.6.3 Vectorisation

Le code que nous cherchons à vectoriser est principalement calculatoire et comporte peu de branchements conditionnels, ce qui en fait un bon candidat pour les outils de vectorisation automatique. La version 4.0 de la bibliothèque *OpenMP* permet de spécifier un certain nombre de règles de vectorisation autour des boucles et des fonctions à vectoriser. Parmi celles-ci, nous appliquons :

Instructions	Coût scalaire	Coût vectorisé	Accélération théorique
SSE2	884	305	×2.9
SSE3	884	305	×2.9
AVX	864	145	×6.0
AVX2	730	140	×5.2

Table 7.4 – Accélérations théoriques de la vectorisation sur la boucle de calcul des échantillons de coin selon le jeu d'instructions

Instructions	Coût scalaire	Coût vectorisé	Accélération théorique
SSE2	1254	375	×3.3
SSE3	1254	375	×3.3
AVX	1209	173	×7.0
AVX2	1100	169	×6.5

Table 7.5 – Accélérations théoriques de la vectorisation sur la boucle de calcul des échantillons de centre selon le jeu d'instructions

— `#pragma omp declare simd`, qui permet de spécifier au compilateur que des versions vectorielles d'une fonction doivent être générées afin de permettre son utilisation dans une boucle vectorisée.
— `#pragma omp simd`, qui, lorsqu'il est placé avant une boucle, indique au compilateur qu'il doit tenter de la vectoriser.
— `#pragma omp simd aligned <DONNÉES>:<TAILLE>` dont l'objectif est d'indiquer que des données sont fournies alignées en mémoire sur un nombre spécifié de bits et que, par conséquent, les instructions de chargement aligné peuvent être directement utilisées, permettant de gagner en performances sur certaines architectures.

Après compilation, nous obtenons du compilateur un rapport de vectorisation donnant une prévision d'accélération basée sur les vitesses d'exécution des instructions SIMD en comparaison avec leurs équivalents séquentiels et la largeur des paquets. Les boucles de calcul des échantillons de coin et de centre des sous-pinceaux sont séparées puisque des grandeurs différentes sont calculées dans les deux cas.

Les tables 7.4 et 7.5 résume les prévisions d'accélération obtenues sur les deux fonctions de calcul d'échantillons selon les jeux d'instructions choisis.

On constate de bonnes accélérations théoriques, particulièrement sur le calcul des échantillons de centre. En effet, pour ces échantillons, le nombre de grandeurs interpolées est plus important que pour un échantillon de coin. Ainsi, la proportion de temps de calcul effectif en rapport au temps total (calcul et lectures/écritures) est supérieure, ce qui explique les accélérations supérieures.

La différence de coût scalaire entre AVX et AVX2 s'explique par l'apparition en AVX2 des instructions FMA qui regroupent une multiplication et une addition en une instruction unique. Puisque les codes d'interpolation font apparaître de nombreuses opérations de multiplications suivies d'additions, un gain d'environ 10% sur les versions scalaires est observable.

7.6.4 Mesure de performances unitaires

Nous mesurons l'efficacité du processus de vectorisation de la phase d'interpolation des échantillons de la surface du traducteur en fonction du jeu d'instructions utilisé. Les mesures de performances que nous présentons ici ne concernent que les étapes d'inversion d'interpolation et d'interpolation des grandeurs de rayons.

Pour les échantillons de centre puis de coins, nous lançons l'exécution du processus d'inversion d'interpolation et d'interpolation des grandeurs sur 32 millions d'échantillons avec les jeux d'instruction SSE et AVX. Une version non vectorisée est également lancée pour comparaison. Les résultats obtenus sont présentés en tables 7.6 et 7.7.

Instructions	Largeur des paquets	Temps (s)	Accélération	Max. théorique
x86	1	5.71	×1	1
SSE3	4	1.68	×3.4	4
AVX	8	0.99	×5.8	8

Table 7.6 – Performances et accélérations de la phase d'interpolation des échantillons centraux avec différents jeux d'instructions SIMD

Instructions	Largeur des paquets	Temps (s)	Accélération	Max. théorique
x86	1	4.70	×1	1
SSE3	4	1.54	×3.1	4
AVX	8	0.88	×5.4	8

Table 7.7 – Performances et accélérations de la phase d'interpolation des échantillons de coin avec différents jeux d'instructions SIMD

Lorsque le jeu d'instructions utilisé est SSE, on constate une bonne adéquation entre les prévisions du compilateur et les accélérations mesurées en pratique. En AVX, les accélérations observées sont sensiblement inférieures aux valeurs théoriques. Ceci est dû au fait que les estimations théoriques produites par le compilateur tiennent seulement compte des cadences d'exécution et des latences des instructions SIMD utilisées par le programme sur les architectures *Intel* correspondant aux jeux d'instructions visés.

En pratique, des ralentissements par rapport à ces estimations théoriques peuvent être causés par les accès mémoire, des défauts de cache... L'utilisation d'instructions AVX permet d'exécuter des opérations sur des registres plus larges qu'en SSE. Cela implique la nécessité de charger plus de données en registres et le nombre de registres requis.

En particulier, le code d'inversion d'interpolation nécessite beaucoup de données puisqu'il requiert le chargement de deux interpolateurs bicubiques pour appliquer une descente de Newton. Ce code est donc plus facilement limité par la bande passante de la mémoire que par les capacités de calcul, particulièrement lorsqu'il est vectorisé. Il est dit *memory bound* et est plus sensible à une amélioration de bande passante qu'à l'utilisation d'unités de calcul plus nombreuses.

À l'inverse, des codes nécessitant le chargement de moins de données en mémoire pour un nombre de calculs importants sont dits *core bound* et sont limités

par la vitesse de traitement des unités de calcul. Les codes d'interpolation bicubique comportent de nombreuses opérations arithmétiques pour une quantité de données inférieure à la phase d'inversion, ils sont un exemple de code *core bound*.

Des profilages plus poussés au moyen des outils d'analyse de performances de la suite *Intel VTune* nous ont permis de confirmer ces hypothèses. Les résultats d'accélération obtenus au moyen des unités d'exécution vectorielles sont satisfaisants et témoignent d'une utilisation efficace des ressources de calcul pour l'étape d'interpolation des échantillons du traducteur.

7.6.5 Mesures de performance à l'échelle du calcul de champ

Enfin, nous procédons à des mesures de performance à l'échelle du calcul de champ pour mesurer l'impact de la vectorisation de l'étape de calcul des échantillons de pinceaux à l'échelle du calcul de champ complet. Pour cela, nous comparons les temps d'exécution du calcul de champ parallélisé avec et sans vectorisation en activant les différents jeux d'instructions possibles.

Ces mesures sont effectuées sur les trois configurations de champ présentées en section 7.3.1. Puisque celles-ci présentent des distributions inégales de temps de calcul entre les trois étapes principales du calcul de champ, cela nous permet de constater l'impact des optimisations de la phase de constitution des réponses impulsionnelles dans trois cas représentatifs des différentes configurations de champ d'un point de vue calculatoire.

Nous réalisons ce test sur les architectures 2 et 3 présentées en table 7.3 afin de mesurer l'effet du passage au jeu d'instructions AVX2 et donc de l'utilisation des opérations FMA. Chacun des tests est exécuté en exploitant le nombre maximal de cœurs et avec trois versions du calcul :

- La version **(a)** purement scalaire, sans vectorisation autre que celle d'*Embree* sans paquets de rayons.
- La version **(b)** exploitant le lancer de rayons par paquets d'*Embree*, mais sans vectorisation de la phase de calcul des échantillons.
- La version **(c)** sans le lancer de rayons par paquets d'*Embree*, mais avec vectorisation de la phase de calcul des échantillons.
- La version **(d)** exploitant le lancer de rayons par paquets d'*Embree* avec vectorisation de la phase de calcul des échantillons.

Les résultats sont détaillés dans les tables 7.8 et 7.9.

Configuration	Temps de calcul (s)				Accélération		
	(a)	**(b)**	**(c)**	**(d)**	**(b)**	**(c)**	**(d)**
1	5.82	4.97	3.67	3.39	×1.17	×1.59	**×1.72**
2	3.00	2.61	1.96	1.82	×1.15	×1.53	**×1.65**
3	10.17	8.68	6.63	5.96	×1.17	×1.54	**×1.71**

Table 7.8 – Impact de la vectorisation de l'interpolation des échantillons du traducteur sur le calcul de champ sur Intel Xeon E5-2630v3 (AVX2, 2 × 8 cœurs)

Il apparaît clairement dans ces résultats que l'impact du lancer de rayons par paquets à l'échelle du calcul de champ est nettement inférieur à celui de la vectorisation de la phase d'interpolation des échantillons. Ceci peut être expliqué par deux raisons principales :

Configuration	Temps de calcul (s)				Accélération		
	(a)	**(b)**	**(c)**	**(d)**	**(b)**	**(c)**	**(d)**
1	3.69	3.38	2.50	2.35	×1.09	×1.47	**×1.57**
2	1.91	1.76	1.31	1.24	×1.09	×1.46	**×1.54**
3	6.44	5.80	4.47	4.17	×1.11	×1.44	**×1.54**

Table 7.9 – Impact de la vectorisation de l'interpolation des échantillons du traducteur sur le calcul de champ sur Intel Xeon E5-2697v2 (AVX, 2 × 12 cœurs)

— Le lancer de rayons *Embree* est vectorisé même lorsque les rayons ne sont pas lancés par paquets. Ainsi, le rapport d'accélération ne reflète pas le rapport entre un code scalaire de lancer de rayons et un code vectoriel de lancer de rayons.
— Les rayons ne sont lancés par paquets que lors des calculs de dérivées des rayons. En effet, lors de la phase de recherche du traducteur, l'impact du lancer de rayons par paquets serait encore plus réduit puisque les rayons lancés sont incohérents.

On remarque que le jeu d'instruction AVX2 améliore sensiblement le gain total occasionné par la vectorisation. Ceci peut s'expliquer par l'amélioration de la cadence d'exécution de certaines instructions SIMD au passage de l'architecture AVX à l'architecture AVX2. En effet, en nous référant à Intel Corporation (2016a), nous résumons les valeurs de latence et de temps d'exécution en cycles des instructions arithmétiques usuelles sur des paquets de 8 flottants en tables 7.10 et 7.11.

Instruction	Exécution (cycles)	Latence (cycles)
Multiplication	0.5	5
Addition	1	3
Division	13	21
Soustraction	1	3

Table 7.10 – Performances des opérations arithmétiques sur des paquets de 8 flottants sur l'architecture *Ivy Bridge*

Instruction	Exécution (cycles)	Latence (cycles)
Multiplication	1	5
Addition	1	3
Division	14	21
Soustraction	1	3

Table 7.11 – Performances des opérations arithmétiques sur des paquets de 8 flottants sur l'architecture *Haswell*

On constate que l'architecture *Haswell* bénéficie de légères améliorations sur les opérations de multiplication et de division, lesquelles se répercutent sur les performances globales.

Enfin, l'architecture *Haswell* a apporté avec le jeu d'instructions AVX des améliorations permettant le chargement de registres depuis des zones de mémoire non

contigües, ce qui a tendance à augmenter les performances générales du code, y compris hors des zones vectorisées à l'aide d'*OpenMP*.

En effet, nous avons utilisé pour compiler ces codes le compilateur *Intel C++ Compiler*, lequel procède automatiquement à des vectorisations partout où cela est possible. Cette fonctionnalité, même si elle n'apporte pas les accélérations que l'on peut obtenir d'un code préparé pour la vectorisation, peut expliquer la différence de performance observée entre AVX et AVX2.

7.7 Mesures de performances globales

Enfin, pour terminer les mesures de performances, nous avons constitué un jeu de configurations de simulation de contrôle variées. Pour chacune de ces configurations, nous avons dans un premier temps validé le résultat de champ par rapport à la référence *CIVA* en tolérant une erreur relative maximale de 0.2 dB puis, nous avons comparé les temps d'exécution de notre code de calcul sur une station de travail standard (équipée d'un processeur Intel Xeon E5-1650v2 à 6 cœurs) à ceux de *CIVA* sur les mêmes configurations et sur la même machine. L'ensemble de ces résultats est recensé en annexe A.

7.8 Conclusion

Au cours de ces travaux, nous avons cherché à exploiter efficacement les processeurs multicœurs. Pour ce faire, une adaptation des algorithmes a été réalisée afin de bénéficier au mieux des outils de lancer de rayons à haute performance.

Suite aux optimisations algorithmiques présentées dans le chapitre 6, nous avons mis en place un parallélisme de tâches au moyen de la bibliothèque *Intel Thread Building Blocks* et obtenu une très bonne mise à l'échelle sur les processeurs que nous avons pu utiliser pour les tests de performance.

Certaines étapes parmi les plus intenses en calcul ont été identifiées au moyen d'une étape de profilage sur des configurations de simulation variées et ont été modifiées afin de pouvoir exploiter les instructions vectorielles des CPU modernes.

Les accélérations obtenues se sont avérées intéressantes et cette étape a été concluante, mais il apparaît néanmoins clair que la vectorisation reste un processus coûteux en termes de développement et de maintenance en comparaison avec les bénéfices qu'il apporte. Cependant, l'évolution des largeurs de paquets SIMD et l'automatisation des outils de vectorisation laissent présager une amélioration de cette situation.

Finalement, les temps de calcul que nous obtenons sur la majorité des configurations testées varient entre quelques dixièmes de secondes et quelques secondes pour des configurations prenant quelques dizaines de secondes à quelques minutes dans *CIVA*. Les résultats produits présentent des écarts inférieurs à un seuil métier de 0.2 dB.

Pour améliorer encore les performances du calcul de champ, il est possible de vectoriser les opérations qui ne l'ont pas encore été, telles que l'identification des pinceaux recouvrants, la sommation des réponses impulsionnelles (une version vectorielle de cette opération a été proposée par Lambert (2015))... Enfin, l'exploitation d'architectures massivement parallèles telles que le GPU et les coprocesseurs dédiés au calcul intensif devrait bénéficier particulièrement aux opérations répétitives et simples telles que les interpolations.

Quatrième partie

Visualisation interactive de champ ultrasonore

CHAPITRE 8

Visualisation progressive de champ ultrasonore

Sommaire

Les chapitres précédents décrivent un ensemble d'outils et de méthodes de calcul de champ ultrasonore permettant de simuler rapidement des contrôles non destructifs sur des configurations variées. Cependant, l'objectif d'interactivité reste à atteindre pour un certain nombre de ces configurations étant donnée leur complexité.

Dans ce chapitre, nous apportons des éléments visant à atteindre l'interactivité sur ces configurations au moyen d'un calcul progressif. La première étape consiste à produire une visualisation progressive de champ, et donc de constituer des images à partir d'échantillonnages incomplets.

Puis, à partir de critères de priorités liés aux variations de l'image de champ, nous établissons un système de calcul des échantillons par priorités afin de converger en un nombre d'échantillons aussi réduit que possible vers une image qualitativement et quantitativement proche de l'image finale.

8.1 Visualisation progressive

Nous avons présenté dans ce qui précède un ensemble de modèles, algorithmes et optimisations permettant d'améliorer nettement les performances du calcul de champ ultrasonore. Cependant, pour des configurations complexes ou mettant en jeu des zones de calcul très denses en points de champ, les performances observées ne sont pas suffisantes pour visualiser interactivement le champ.

Comme nous avons pu l'observer précédemment, les images de champ ultrasonore présentent généralement des propriétés de continuité et de régularité. En tirant parti de ces régularités et en posant une hypothèse de continuité de l'image de champ, nous pouvons produire une image visuellement semblable à l'image de champ complète à partir d'un échantillonnage incomplet.

Puisque nous nous intéressons à une visualisation progressive d'images de champ dont le temps de calcul est supérieur au seuil d'interactivité, nous devons pouvoir construire une image visuellement cohérente à partir d'échantillonnages réduits de l'image finale. La section 8.1.3 détaille la méthode d'interpolation que nous utilisons.

Pour un même nombre de points d'échantillonnage, différentes répartitions de ces points mèneront à des images de qualité variable. Après avoir proposé un ensemble de critères permettant de choisir des répartitions de points plus efficaces qu'une répartition homogène, en correspondance avec la méthode de reconstruction d'images que nous utilisons, nous posons les bases d'un algorithme d'ordonnancement des calculs de champ pour une visualisation progressive en section 8.1.4.

8.1.1 Travaux similaires

Huthwaite (2014) utilise le GPU pour accélérer des simulations d'ultrasons par éléments finis d'un à deux ordres de grandeur par rapport aux résultats précédents sur CPU. Les performances, bien que très satisfaisantes pour une méthode à éléments finis, ne suffisent pas à atteindre l'interactivité.

Lambert (2015) a développé un outil de calcul de champ interactif ultrasonore sur CPU, GPU et MIC en mesure de produire des images de champ ultrasonore pour des configurations de contrôle non destructif simples faisant intervenir des pièces planes, des matériaux isotropes et des inspections directes ou avec une réflexion à une cadence de 20 à 30 images par seconde.

Il propose également une visualisation progressive avec un raffinement uniforme pour les configurations qu'il ne peut calculer de manière interactive. Il s'agit, à notre

connaissance et à ce jour, de la seule approche ayant abouti sur une simulation interactive de contrôle non destructif par ultrasons.

8.1.2 Interactivité

En visualisation, l'interactivité est définie comme la capacité pour un système informatique de réagir aux entrées utilisateur en un temps inférieur au temps moyen de traitement d'une information par un cerveau humain. Miller (1968) et Card et collab. (1991) fournissent des bases théoriques et pratiques pour déterminer un temps acceptable de rafraîchissement d'images pour un système interactif. Ainsi, on considère qu'une visualisation est interactive à partir d'un temps de rafraîchissement de l'ordre de $100ms$, soit une fréquence de $10Hz$, correspondant au temps de traitement nominal d'une information par le cerveau humain. Une fréquence d'environ $25Hz$ permet de dépasser le seuil de persistance rétinienne, ce qui résulte en une impression de fluidité.

Ce critère nous permet d'obtenir une valeur cible de fréquence de rafraîchissement : une valeur trop faible résulterait en un affichage saccadé ou un système trop lent pour être interactif, tandis qu'une valeur trop élevée correspond à un nombre inutilement grand de reconstructions d'images de champ et donc en une utilisation excessive des capacités de calcul. Cela permet donc de limiter le nombre d'appels à la méthode de reconstruction d'images nécessaires à une visualisation interactive fluide.

8.1.3 Reconstruction d'image

Afin d'obtenir une image visuellement acceptable d'un champ ultrasonore à partir d'un échantillonnage incomplet, nous nous intéressons aux méthodes de reconstruction d'images. Étant donné un ensemble de points d'échantillonnage du champ ultrasonore dans une zone de calcul, nous cherchons une fonction coïncidant avec cet échantillonnage en chaque point le constituant et présentant les propriétés de régularité et de continuité des images de champ.

Les méthodes de reconstruction d'images peuvent être séparées en deux catégories principales :

— Les méthodes d'interpolation, telles que les splines (voir Hou et Andrews (1978) et Lee et collab. (1997)) et les méthodes d'interpolation polynomiales (présentées dans le chapitre 4).

— Les méthodes d'approximation, telles que l'approximation *B-spline* ou la réduction polynomiale.

Giunta et collab. (1998) proposent une comparaison d'un large panel de méthodes de reconstruction d'images et de surfaces en les différenciant selon les cas d'application. Les méthodes d'approximation sont adaptées aux situations dans lesquelles l'aspect général de l'image prime sur l'aspect quantitatif des valeurs attribuées à chaque point.

Dans le cas de la simulation de champ ultrasonore, les valeurs calculées des échantillons doivent être exactement retranscrites dans l'image produite puisque l'analyse est aussi quantitative. Pour cette raison, nous nous sommes intéressés aux méthodes d'interpolation, exactes aux points d'échantillonnage et fournissant une approximation dans les zones non calculées.

Puisque les images de champ que nous cherchons à construire ne présentent pas de discontinuités et sont lisses, une méthode d'interpolation basée sur des polynômes

de degrés faibles, comme nous l'avons appliquée à la construction du front d'onde en section 4, fournit de bons résultats.

Nous avons choisi d'utiliser la méthode MBA (*Multilevel B-spline Approximation*) telle que proposée par Lee et collab. (1997). Il s'agit d'un processus itératif appliquant successivement plusieurs approximations par des *B-splines* à l'image de champ et à l'image des erreurs sur les échantillons calculés de manière à converger vers une interpolation présentant toutes les caractéristiques de régularité et de lissage des approximations par *B-splines*.

Cette méthode est particulièrement efficace pour construire des fonctions lisses à partir de données éparses réparties de manière quelconque dans l'espace d'interpolation. Notons que son coût, plus élevé que celui des méthodes d'interpolation proposées pour la construction des fronts d'onde ultrasonore, n'est pas un frein aux performances de la visualisation interactive puisque cette interpolation n'est exécutée qu'à la cadence de la visualisation.

8.1.4 Raffinement adaptatif

Identification de zones de priorité

Étant donné un échantillonnage incomplet d'une image de champ composé de n points, nous cherchons à déterminer un ensemble de points d'échantillonnage supplémentaires afin d'améliorer la qualité de l'image. La méthode d'interpolation que nous avons choisie utilise des polynômes de degré faible et tend à minimiser les fortes variations.

Les zones présentant les variations les plus fortes sont donc susceptibles de nécessiter plus de points d'échantillonnage pour être efficacement reconstruites. Ainsi, pour améliorer la précision de la reconstruction, il est utile d'effectuer une détection de ces zones à fortes variations.

Calculer le champ des points voisins des points d'échantillonnage nous permet de déduire les dérivées secondes selon les axes horizontaux et verticaux en chaque point d'échantillonnage. D'après l'hypothèse de continuité du champ et de ses dérivées, les valeurs les plus élevées de dérivées secondes caractérisent les voisinages des zones à forte variation. On peut donc chercher à calculer en priorité les zones avoisinant les points où le champ présente une forte dérivée seconde.

À ce critère de variation vient s'ajouter un critère propre au contexte du contrôle non destructif. En effet, dans l'usage courant des images de champ ultrasonore, les zones de fortes amplitudes caractérisent la focalisation des ondes ultrasonores et donc les zones dont les échos éventuels ont la plus forte amplitude. Priorité est donc également donnée aux points présentant les plus fortes amplitudes.

Pour définir une valeur de priorité à partir des amplitudes et gradients, il nous faut donc connaître les maximums de ces deux grandeurs sur les points échantillonnés. En notant respectivement $(a_k)_{1\leq k\leq n}$, $(\partial_{2x}a_k)_{1\leq k\leq n}$ et $(\partial_{2y}a_k)_{1\leq k\leq n}$ l'amplitude, la dérivée seconde selon l'axe x et la dérivée seconde selon l'axe y du champ aux n points de l'échantillonnage, on peut définir des scores de priorité comme suit :

$$\begin{cases} p_k^{amp} = \dfrac{|a_k|}{\|(a_i)_{1\leq i\leq n}\|_\infty} \\ p_k^{diff} = max\left(\dfrac{|\partial_{2x}a_k|}{\|(\partial_{2x}a_i)_{1\leq i\leq n}\|_\infty}, \dfrac{|\partial_{2y}a_k|}{\|(\partial_{2y}a_i)_{1\leq i\leq n}\|_\infty}\right) \end{cases}$$

Puis un score de priorité global entre 0 et 2 pour la zone avoisinant le point d'échantillonnage k :

$$p_k = p_k^{amp} + p_k^{diff}$$

Raffinement d'images

Pour construire une distribution de scores de priorité dans l'image de champ, nous construisons une grille à partir des points d'échantillonnage. Nous choisissons un échantillonnage de départ de $5 \times 5 = 25$ points uniformément répartis. Chaque échantillon de l'image finale est identifié par ses coordonnées entières x et y.

⟨*Point2DEntier*⟩≡

```
struct Point2DEntier {
   int x, y;
   float amplitude;
};
```

⟨*Image*⟩≡

```
struct Image {
   int xMinimal, xMaximal,
      yMinimal, yMaximal;
};
```

L'image est décomposée en quatre **nœuds**, chacun composé de 9 points d'échantillonnage que l'on identifie par les points cardinaux Nord-Est (**NE**), Nord (**N**), Nord-Ouest (**NO**), Ouest (**O**), Sud-Ouest (**SO**), Sud (**S**), Sud-Est (**SE**), Est (**E**) et Centre (**C**).

⟨*Noeud*⟩≡

```
struct Noeud {
   Point2DEntier pointNE, pointN, pointNO,
              pointE, pointC, pointO,
              pointSE, pointS, pointSE;

   Noeud(int xMinimal, int xMilieu, int xMaximal,
        int yMinimal, int yMilieu, int yMaximal) {
      pointNE.x = xMaximal; pointNE.y = xMaximal;
      pointN.x = xMilieu; pointN.y = xMaximal;
      pointNO.x = xMinimal; pointNO.y = xMaximal;
      pointO.x = xMinimal; pointO.y = xMilieu;
      pointSO.x = xMinimal; pointSO.y = xMinimal;
      pointS.x = xMilieu; pointS.y = xMinimal;
      pointSE.x = xMaximal; pointSE.y = xMinimal;
      pointE.x = xMaximal; pointSE.y = xMilieu;
      pointC.x = xMilieu; pointC.y = xMilieu;
   }

   ⟨estDivisible 186⟩
   ⟨score 185⟩

   float amplitudeMax;
   float dérivéeSecondeMax;
   static float amplitudeMaxGlobale;
   static float dérivéeSecondeMaxGlobale;
   static
};
```

Point2DEntier 183

Pour raffiner l'image, nous procédons à la subdivision d'un ou plusieurs nœuds de l'image. En adoptant la structure arborescente schématisée en figure 8.1 pour la

représentation des nœuds de l'image, le problème de l'ordre de raffinement de l'arbre se rapporte à une exploration d'arbre.

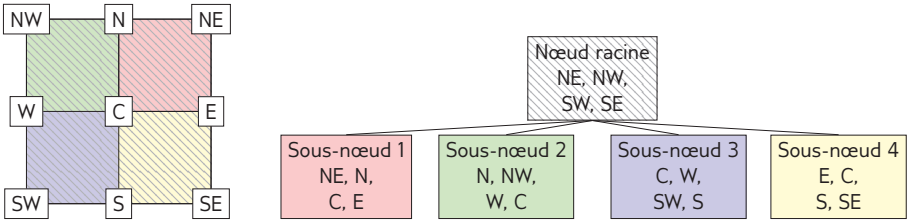


Figure 8.1 – Représentation en *quad-tree* d'une image de champ

De la même façon que ce qui a été fait pour l'algorithme de lancer de pinceaux, nous utilisons un algorithme d'exploration de l'arbre se servant des critères de priorité. Un score est donné à chaque nœud de l'image et les nœuds aux scores les plus élevés sont subdivisés prioritairement.

Le critère de score calculé prend en compte le maximum d'amplitude sur le nœud et les dérivées secondes discrètes calculées à l'échelle du nœud. Ainsi, on a :

$$\begin{cases} \partial_{2x} a_k = \dfrac{a_{E,k} - 2a_{C,k} + a_{W,k}}{4\Delta x^2} \\ \partial_{2y} a_k = \dfrac{a_{E,k} - 2a_{C,k} + a_{W,k}}{4\Delta y^2} \end{cases}$$

avec Δx et Δy les pas d'échantillonnage respectivement horizontal et vertical du nœud.

Enfin, pour empêcher une exploration en profondeur dans les zones à forte amplitude menant à une densité très hétérogène de l'échantillonnage, nous choisissons d'appliquer un coefficient lié à la taille du nœud :

$$p_k = (p_k^{amp} + p_k^{diff}) * \sqrt{\Delta x \Delta y}$$

Cela revient à diviser par deux le score d'un nœud pour chaque niveau de subdivision déjà parcouru avant de l'atteindre. De cette façon, les nœuds les plus profonds dans l'arbre sont pénalisés par rapport aux autres, de sorte à équilibrer l'exploration.

En pratique, de la même manière que pour le calcul de champ ultrasonore, nous utilisons une file de priorité pour l'exploration de l'arbre d'échantillonnage de l'image de champ :

⟨*Algorithme d'exploration de l'image*⟩≡

```
CalculChamp    void explorationImage(Image image, CalculChamp calculateur) {
Noeud 183         priority_queue<Noeud, ComparaisonNoeuds> file;
ComparaisonNoeuds
186               Noeud noeudRacine(image.xMinimal,
Noeud 183                         (image.xMinimal + image.xMaximal) / 2,
                                  image.xMaximal,
                                  image.yMinimal,
                                  (image.yMinimal + image.yMaximal) / 2,
                                  image.yMaximal);

calculerNoeud     calculer(calculateur, noeudRacine);
185               file.push(noeudRacine);

                  while(!file.empty()) {
                     Noeud noeudCourant = file.pop();
estDivisible         if(noeudCourant.estDivisible()) {
186                     array<Noeud, 4> noeudsFils = noeudCourant.subdiviser();
subdiviser
186
```

```
         for(int i = 0; i < 4; ++i) {
            calculateur.calculer(noeudsFils[i]);
            file.push(noeudsFils[i]);
         }
      }
   }
}
```

Dans un premier temps, le nœud racine est initialisé pour couvrir l'ensemble de la surface de l'image de champ. Les coordonnées de ses 9 échantillons permettent une subdivision équitable du nœud en sous-nœuds de surface égale (à condition que le nœud soit de largeur et de hauteur paires). Ses échantillons et son score de priorité sont calculés et il est ajouté à la file de priorité. Les maximums globaux d'amplitude et de dérivée seconde sont mis à jour à chaque calcul de score.

⟨*calculerScore*⟩≡

```
void calculerScore(Noeud noeud) {
   amplitudeMax = max(noeud.pointNE.amplitude, ...,
                   noeud.pointC.amplitude);
   Noeud::amplitudeMaxGlobale = max(Noeud::amplitudeMaxGlobale,
                               amplitudeMax);

   dérivéeSecondeMax = max(abs(dérivéeSecondeX(noeud),
                       abs(dérivéeSecondeY(noeud)));
   Noeud::dérivéeSecondeMaxGlobale = max(Noeud::dérivéeSecondeMaxGlobale,
                                    dérivéeSecondeMax);
}
```

Maximum d'amplitude des échantillons

⟨*score*⟩≡

```
float score() {
   float facteurTaille = sqrtf((float)((pointE.x - pointW.x) *
                                (pointN.y - pointS.y)));
   float termeAmplitude = amplitudeMax / amplitudeMaxGlobale;
   float termeDérivéeSeconde = dérivéeSecondeMax /
                           dérivéeSecondeMaxGlobale;

   return facteurTaille * (termeAmplitude + termeDérivéeSeconde);
}
```

⟨*calculerNoeud*⟩≡

```
void calculerNoeud(CalculChamp calculateur, Noeud noeud) {
   noeud.pointNE.amplitude = calculateur.calculAmplitudeMax(noeud.pointNE.x,
     noeud.pointNE.y);
   ... Calcul de tous les échantillons du nœud.
   noeud.pointC.amplitude = calculateur.calculAmplitudeMax(noeud.pointNE.x,
     noeud.pointC.y);

   ⟨calculerScore 185⟩
}
```

Vient ensuite la boucle principale qui est exécutée tant que la file de priorité contient au moins un élément. À chaque itération, le nœud de plus haut score est retiré de la file de priorité. Avant de lancer le processus de subdivision, un test est effectué pour vérifier que le nœud est suffisamment étendu pour être subdivisé. Le cas contraire signifie que l'exploration de la branche de l'arbre contenant le nœud courant est terminée et donc que la définition finale de l'image est atteinte pour cette zone.

⟨*estDivisible*⟩≡
```
bool estDivisible() {
   bool testHorizontal = pointE.x > pointC.x + 1 &&
                         pointC.x > pointW.x + 1;
   bool testVertical = pointN.x > pointC.x + 1 &&
                       pointC.x > pointS.x + 1;
   return testHorizontal && testVertical;
}
```

Dans le cas où le nœud est assez étendu, il est subdivisé équitablement (sauf s'il est de largeur ou de hauteur impaire). Les échantillons des nœuds résultant de cette subdivision sont calculés pour pouvoir extraire les valeurs de score de priorité.

⟨*subdiviser*⟩≡
```
array<Noeud, 4> subdiviser() {
   Noeud noeudNE { pointE.x, (pointE.x + pointC.x) / 2, pointC.x,
                   pointN.y, (pointN.y + pointC.y) / 2, pointC.y };
   Noeud noeudNW { pointC.x, (pointC.x + pointW.x) / 2, pointW.x,
                   pointN.y, (pointN.y + pointC.y) / 2, pointC.y };
   Noeud noeudSW { pointC.x, (pointC.x + pointW.x) / 2, pointW.x,
                   pointC.y, (pointC.y + pointS.y) / 2, pointS.y };
   Noeud noeudSE { pointE.x, (pointE.x + pointC.x) / 2, pointC.x,
                   pointC.y, (pointC.y + pointS.y) / 2, pointS.y };

   array<Noeud, 4> noeudsFils = { noeudNE, noeudNW, noeudSW, noeudSE };
   return noeudsFils;
}
```

Les noeuds ainsi créés sont ajoutés à la file de priorité. Un opérateur de comparaison entre nœuds est utilisé pour cela. Celui-ci se base sur les scores pour établir une relation d'ordre entre les nœuds. Le score est calculé à la volée afin de tenir compte de l'évolution des maximums globaux.

⟨*ComparaisonNoeuds*⟩≡
```
bool ComparaisonNoeuds(Noeud noeudA, Noeud noeudB) {
   return noeudA.score() < noeudB.score();
}
```
score 185

À la fin de l'exploration, il est garanti que l'ensemble des points d'échantillonnage a été calculé, l'image est donc complète. Notons que le critère de score ici utilisé est spécifique au cas d'application des images de champ ultrasonore pour le contrôle non destructif et s'appuie sur des propriétés connues de ces images. Puisque la méthode que nous présentons ici est adaptable à d'autres applications, le calcul du score peut être différent pour favoriser d'autres critères.

De cette façon, nous disposons d'un algorithme de raffinement progressif et adaptatif d'images de champ. Il favorise à la fois les zones de forte amplitude, pour favoriser un critère métier associé au contrôle non destructif et les zones à fortes variations, pour que l'interpolation polynomiale de degré faible s'approche des valeurs réelles plus rapidement.

8.1.5 Visualisation progressive d'images reconstruites

Les outils présentés en sections 8.1.3 et 8.1.4 permettent respectivement de reconstruire une image à partir d'un échantillonnage incomplet et d'obtenir une séquence d'échantillonnages incomplets qui favorise la convergence rapide vers une image de champ exploitable. En les combinant, il nous est donc possible d'obtenir une visualisation progressive du champ ultrasonore convergeant rapidement vers l'image complète.

Pour obtenir une visualisation progressive, interactive et fluide de champ, les calculs d'images détaillés en section 8.1.3 à partir des échantillonnages incomplets fournis par la méthode présentée en section 8.1.4 peuvent être effectués avec une période temporelle de 40ms.

Afin d'y parvenir, les opérations de calcul de champ et d'exploration de l'arbre des subdivisions sont découplées de la reconstruction d'image et de l'affichage. On a donc une tâche d'exploration d'arbre et de calcul de champ et une tâche dédiée à la reconstruction et à l'affichage d'images de champ. Ces deux tâches fonctionnent simultanément : l'une fonctionne en continu tandis que l'autre est appelée régulièrement.

8.2 Résultats

Dans cette section, nous mettons en évidence les améliorations apportées par la méthode de reconstruction utilisée pour produire des images approchées du champ à partir d'échantillonnages incomplets et par le mécanisme d'exploration guidée de l'arbre des subdivisions.

8.2.1 Mesure de la qualité

Les outils présentés en section 8.1 partagent l'objectif de fournir une visualisation plus rapide et plus fluide du champ ultrasonore par le biais d'une approximation exploitant des propriétés connues *a priori* de l'image finale. Cette approximation n'a d'intérêt que si elle permet effectivement de produire rapidement des images exploitables d'un point de vue métier.

Pour mesurer leur efficacité, il nous faut disposer d'une **métrique de qualité** des images de champ. Celle-ci doit nous permettre de vérifier l'intérêt de la méthode progressive que nous avons exposée pour un même nombre d'échantillons, en comparaison avec une méthode raffinement uniforme de l'image.

Selon les cas d'application, plusieurs méthodes de mesure de qualité d'image existent. Nous les classons en deux catégories principales :

Les méthodes quantitatives telles que la mesure de la moyenne quadratique, la corrélation ou toute autre méthode numérique de mesure d'erreur. Fréquemment utilisées en traitement du signal, elles visent à mesurer des écarts entre signaux d'un point de vue purement numérique.

Les méthodes perceptuelles qui visent non pas à chiffrer les différences entre deux structures de données, mais bien à les comparer en se basant sur des modèles de perception visuelle humaine. Lavoué et collab. (2015) comparent l'efficacité de certaines de ces techniques pour l'évaluation de la qualité de modèles tridimensionnels. On peut en particulier citer la méthode *Structural SIMilarity*, voir Wang et collab. (2004) (SSIM), largement utilisée dans des contextes d'application variés et qui se sert de données structurelles pour comparer des images.

Les images de champ sont des distributions discrètes de maximums d'amplitude de champ ultrasonore. Il s'agit donc de matrices de flottants. Dans notre cas, la qualité de perception de l'image (représentée en fausses couleurs) est influencée par le lissage de la surface d'amplitudes. Les métriques basées sur des modèles perceptuels ne s'appliquent pas puisque la correspondance entre amplitude et couleur est dépendante des extremums des valeurs de champ de l'image.

Bien que la forme des zones de fortes amplitudes joue un rôle important dans l'interprétation des images de champ, la précision de l'approximation de l'image de champ scalaire est ce qui détermine, en pratique, leur qualité. Pour cette raison, nous choisissons d'utiliser une méthode quantitative, la moyenne quadratique, pour évaluer l'erreur commise par les approximations successives de la visualisation progressive de champ.

8.2.2 Mesures de différences d'images de champ à l'aide d'une métrique quantitative

Pour évaluer l'efficacité de l'algorithme de visualisation progressive sur une configuration donnée, nous commençons par effectuer un calcul complet du champ sur toute la zone. L'image obtenue nous sert de référence de comparaison pour les images calculées en visualisation progressive.

Puis, nous lançons l'algorithme de visualisation progressive et, à chaque image produite, nous calculons une valeur de qualité déterminée par la moyenne quadratique de la différence entre l'image de référence et l'image approximée. Nous répétons ce processus pour chacun des trois modes de raffinement considérés et obtenons les courbes de qualité présentées en figure 8.2.

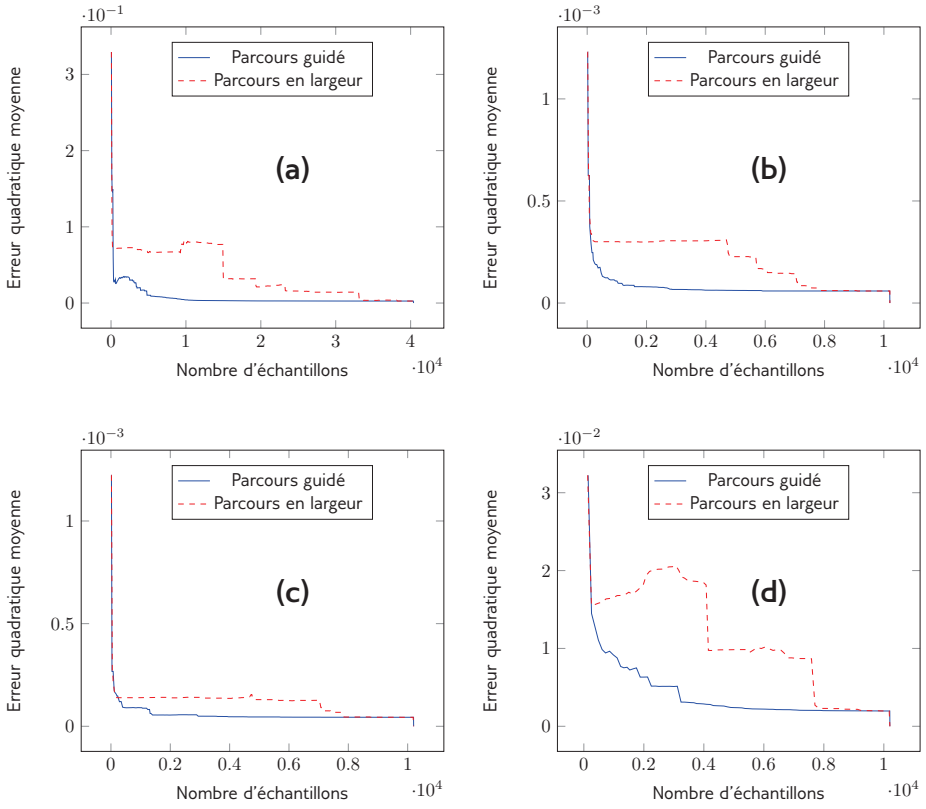


Figure 8.2 – Évolution de l'erreur RMS avec le nombre d'échantillons

L'interpolation *Multilevel B-spline Approximation* (MBA) construit des images de bonne qualité à partir d'échantillonnages incomplets assez réduits. Nous constatons que, conformément à nos prévisions, pour un même nombre d'échantillons, l'exploration adaptative fournit des images de meilleure qualité que l'exploration uniforme, en particulier sur des images où les zones de fortes amplitudes sont réduites.

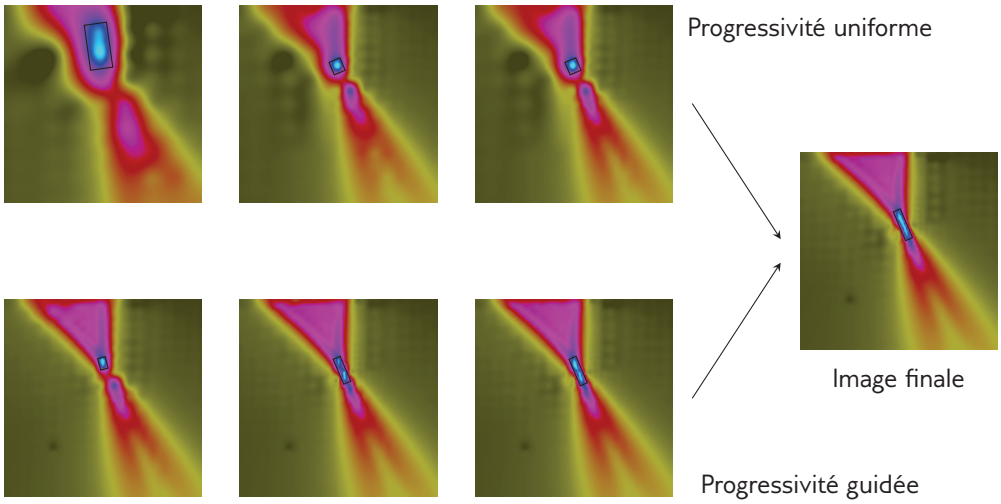


Figure 8.3 – Séquence d'images raffinées uniformément (en haut) et en progressivité guidée (en bas)

La figure 8.3 montre quelques étapes de l'évolution de l'image de champ visualisée avec et sans raffinement adaptatif. Ces images illustrent bien la convergence autour des zones à forte variation et forte amplitude et la rapide convergence vers un résultat exploitable. L'algorithme de subdivision adaptative améliore donc sensiblement la convergence de l'image d'un point de vue numérique en guidant le raffinement sur les zones mal approximées en premier lieu. Il peut donc efficacement servir d'ordonnanceur de calcul de champ ultrasonore dans un objectif de production rapide d'images de champ.

8.2.3 Mesure de critères métier

En contrôle non destructif, les images de champ ultrasonore sont souvent utilisées pour rapidement trouver la zone de couverture du faisceau d'ondes, qui correspond à la zone de fortes amplitudes. Ce critère, plus qualitatif, peut être mesuré en encadrant les zones de forte amplitude, ce que nous avons fait en figure 8.3. Depuis l'amplitude maximale de l'image, cette boîte englobe une zone où la chute d'amplitude est inférieure ou égale à $3dB$.

Nous pouvons alors mesurer l'écart de position, de taille et d'angle pour ces boîtes englobantes. On constate que les zones de couverture du faisceau d'ondes convergent rapidement vers le résultat final grâce à la méthode de raffinement adaptatif.

L'exploitabilité des images de champ n'est pas directement reliée à un critère quantitatif, mais des observations qualitatives nous permettent de conclure que des images interprétables peuvent être obtenues avec très peu de points de champ sur les configurations testées, soulignant l'intérêt de l'approche adaptative pour l'exploration de l'arbre des subdivisions.

Cette amélioration constitue un pas important pour la visualisation interactive de champ ultrasonore. En effet, la capacité d'obtenir un aperçu réaliste du champ ultrasonore avec peu de points réduit le temps de calcul nécessaire pour obtenir une image de champ exploitable en simulation. L'étape suivante est donc la mesure et l'optimisation des performances de l'approche progressive et adaptative de visualisation de champ pour atteindre l'interactivité sur un maximum de configurations.

8.3 Conclusion

Avec pour objectif de fournir une visualisation interactive de champ dans un maximum de cas et également sur des processeurs usuels de stations de travail, nous avons mis en place dans ce chapitre un calcul progressif de champ qui produit des images de champ progressivement raffinées au fur et à mesure de l'avancement du calcul de champ.

Pour construire ces images à partir d'échantillonnages incomplets du champ ultrasonore, nous utilisons une méthode d'interpolation lisse basée sur les *splines*. Ce faisant, nous exploitons une hypothèse de régularité et de lissage des images de champ.

La séquence d'images ainsi produite converge vers l'image de champ finale. En utilisant un critère de priorité sur le calcul des points de champ, nous améliorons la vitesse de convergence de cette séquence et donc la qualité visuelle des résultats. Ainsi, on peut obtenir une image exploitable avec moins de points de champ.

Le critère de priorité que nous utilisons calcule numériquement la dérivée seconde du champ en un point pour estimer la tendance de variation locale. Ce critère part du principe qu'une zone à fortes variations nécessite plus de points d'échantillonnage pour être efficacement interpolée.

Nous lui associons un critère orienté métier qui favorise les zones à forte amplitude puisqu'en contrôle non destructif, un utilisateur cherche en priorité à visualiser les parties de l'image où l'amplitude du champ est élevée.

Notons que ce critère a pour effet collatéral d'accélérer la convergence puisque, de manière générale, les points de faible amplitude requièrent un temps de calcul plus élevé puisque leur réponse impulsionnelle est plus étalée dans le temps. Indirectement, le critère de forte amplitude est équivalent à un ordonnancement calculant les points de champ les plus rapides à traiter en premier.

CHAPITRE 9

Exploitation des architectures parallèles

Sommaire

Nous avons mis en place un algorithme permettant de calculer progressivement une image de champ ultrasonore. Il met en jeu un système de priorités améliorant la convergence de processus vers l'image complète et tenant compte à la fois des spécificités des images de champ ultrasonore et de l'outil de reconstruction utilisé.

Cette section a pour objectif de mettre en évidence les problèmes de parallélisme posés par cet algorithme. Nous y proposons ensuite une solution réduisant très nettement les coûts du calcul progressif par rapport à un calcul complet standard en exploitant efficacement le parallélisme de tâches sur les processeurs multicœurs actuels.

9.1 Formalisation du problème

Nous proposons la décomposition en tâches suivante pour le processus de visualisation progressive d'images de champ avec raffinement adaptatif :

1. L'exploration de l'arbre des subdivisions de l'image de champ. C'est un processus peu coûteux, mais comme chaque étape de cette exploration est dépendante de ses étapes parentes, les différentes étapes du processus global sont liées par des relations de dépendance.
2. Le calcul de champ ultrasonore tel que décrit en partie II. Il représente les calculs les plus coûteux et notre objectif est de faire en sorte qu'il représente la grande majorité du temps de calcul tout en l'accélérant autant que possible.
3. La reconstruction d'images à partir d'échantillonnages incomplets, présentée en section 8.1.3. Elle est effectuée environ 25 fois par seconde et représente un temps de calcul nettement plus faible que la phase de calcul de champ.
4. L'affichage de l'image de champ.

Sur les CPU modernes, plusieurs pistes d'accélération sont envisageables, parmi lesquelles l'utilisation du parallélisme d'instructions, exploitant les opérations SIMD et le parallélisme de tâches, permettant d'exécuter simultanément des tâches différentes sur des données différentes en faisant travailler de concert les multiples cœurs du processeur.

Comme nous avons pu le voir en partie III, deux niveaux de parallélisme sont utilisables pour améliorer les performances du calcul de champ : le parallélisme de tâches exploitant simultanément les multiples cœurs d'un CPU et le parallélisme de données effectuant des opérations identiques sur plusieurs données simultanément.

Dans le cas du calcul complet, en séparant équitablement les points de champ entre les *threads* système, nous avons pu exploiter efficacement les différents cœurs du CPU. Dans le cas du calcul interactif, des relations de dépendance existent entre les différentes étapes de processus qui empêchent une parallélisation aussi simple. Dans la suite, nous détaillons ces étapes et expliquons comment il est possible de tirer pleinement parti du parallélisme de tâches dans cette situation.

Pour commencer, nous effectuons une liste des relations de dépendance d'une tâche à l'autre pour déterminer quelles opérations peuvent être effectuées simultanément et quelles opérations doivent être effectuées séquentiellement :

— L'étape 1, d'exploration de l'arbre des subdivisions, dépend des priorités attribuées aux nœuds et donc du calcul de champ.
— L'étape 2, de **calcul** de champ ultrasonore, est ordonnée par l'exploration de l'arbre des subdivisions.

— L'étape 3, de **reconstruction** d'images, requiert les résultats de calcul de champ.
— L'étape 4, ou opération d'**affichage**, prend en données d'entrée les résultats de la reconstruction.

On constate en premier lieu une relation de dépendance circulaire entre l'exploration de l'arbre des subdivisions et le calcul de champ. Cela traduit le fait que l'algorithme d'exploration d'arbre empêche une exploration parallélisée de tous les nœuds puisque la construction des nœuds d'un niveau de profondeur requiert l'exploration préalable d'au moins un nœud du niveau précédent.

Les nœuds d'un même niveau et des nœuds de niveaux différents ne présentant pas de relations de filiation peuvent cependant être explorés en parallèle. De plus, les étapes 3 et 4 de reconstruction et d'affichage peuvent être exécutées en parallèle des deux autres à condition de disposer d'une structure de données partagée. La figure 9.1 représente sous forme d'un diagramme l'ensemble des tâches du processus de visualisation interactive de champ ultrasonore ainsi que les communications entre celles-ci.

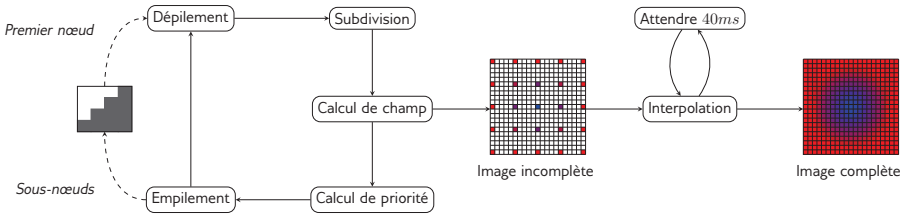


Figure 9.1 – Tâches du processus de visualisation progressive de champ et communications

Un nœud de l'arbre des subdivisions possède dans le cas général 4 sous-nœuds directs (sauf pour les nœuds terminaux de l'arbre et les nœuds divisibles selon une dimension seulement). Quel que soit le mode de parallélisme envisagé, l'exploration d'un arbre entraîne une augmentation exponentielle du nombre de nœuds à traiter à un niveau. Le nombre de tâches exécutables en parallèle risque de croître exponentiellement. Pour cette raison, nous choisissons d'utiliser un modèle de parallélisme de tâches plutôt qu'un modèle basé sur les *threads* système.

Nous utilisons pour cela la bibliothèque ITBB, qui propose un ensemble d'outils de haut niveau pour exploiter des architectures parallèles au moyen d'un ordonnanceur de tâches basé sur le principe du *work-stealing*. En utilisant ITBB, un jeu de *threads* systèmes est créé, à raison d'un *thread* par cœur logique. Ceux-ci sont statiques et se voient fournir une pile de tâches par l'ordonnanceur. Si un *thread* a vidé sa pile de tâches, il récupère une tâche de la pile d'un autre *thread* choisi au hasard. Ce mécanisme, opposé à l'ordonnancement statique, permet un équilibrage naturel des charges de travail entre les différents cœurs du processeur et ce, même pour un problème dont on ne connaît pas la distribution *a priori*.

L'ensemble des modèles de parallélisme les plus courants sont détaillés dans McCool et collab. (2012). Leurs implémentations en utilisant des bibliothèques de parallélismes usuelles (ITBB et *Cilk*) y sont présentées, ainsi que des considérations algorithmiques et des exemples. Nous utilisons dans la suite les modèles de pipeline et de `parallel_for`.

9.2 Modèle de parallélisme

En calcul complet, il n'existe pas de priorité sur le calcul des points de champ, ce qui permet d'exécuter les calculs unitaires de champ par point dans un ordre arbitraire. Par ailleurs, le grand nombre de points de champ permet de tirer parti aisément des cœurs du processeur puisque la distribution des tâches est naturelle.

En revanche, en mode de calcul progressif, les points de champ à calculer sont obtenus au fur et à mesure de l'exploration de l'arbre des subdivisions. À chaque fois qu'un nœud est dépilé de la file de priorité, il est subdivisé. Chacun des sous-nœuds contient alors un certain nombre de points à calculer pour déterminer son score. Comme nous le montrons en figure 9.2, le total des points de champ à calculer est au maximum de 16. Il peut être inférieur lorsque des nœuds voisins ont déjà été calculés.

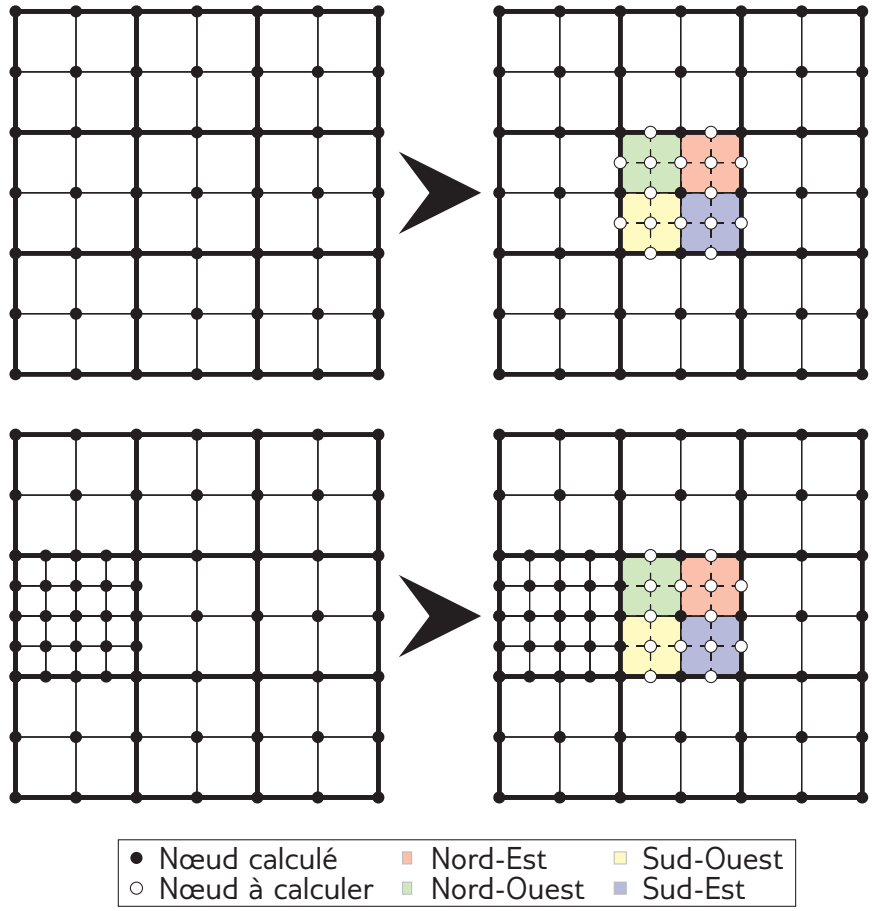


Figure 9.2 – Subdivision des nœuds de l'image de champ, calcul de 16 échantillons. Certains nœuds parmi ceux-ci ont pu être calculés au préalable

Comme la phase de calcul de champ est la plus coûteuse du processus, une attention particulière est apportée à sa parallélisation. Nous présentons dans la suite plusieurs solutions de parallélisation pour obtenir la meilleure efficacité possible. Au moyen d'un cas d'exemple théorique pour lequel 10 nœuds $(N_i)_{1\leq i\leq 10}$ non voisins doivent être calculés sur un processeur doté de 8 cœurs physiques. Pour clarifier les schémas à venir, pour un nœud N_i, nous notons 4 tâches $N_{i,NE}$, $N_{i,NO}$, $N_{i,SO}$, $N_{i,SE}$ correspondant au calcul des points de champ manquants pour chacun des sous-pinceaux

Nord-Est, Nord-Ouest, Sud-Ouest et Sud-Est. Nous les considérons comme étant insécables dans la suite.

9.2.1 Parallélisation par échantillon dans un nœud

Une première approche pour paralléliser les calculs de champ pour les échantillons après une subdivision peut consister à calculer en parallèle les échantillons de chaque nœud. Ainsi, pour le nœud N_i, le calcul des échantillons $N_{i,NE}$, $N_{i,NO}$, $N_{i,SO}$, $N_{i,SE}$ pourrait être effectué en parallèle.

On peut pour cela utiliser le modèle de parallélisme `parallel_for`. De cette façon, chaque paquet d'échantillons $N_{i,NE}$, $N_{i,NO}$, $N_{i,SO}$, $N_{i,SE}$ est attribué à un *thread* différent. L'étape de calcul de champ du processus se décompose alors en trois sous-étapes :

— Distribution des paquets d'échantillons à des *threads* différents.
— Calcul de champ en parallèle.
— Synchronisation des *threads* à la fin du calcul de champ.

La meilleure accélération potentiellement atteignable avec cette façon de procéder est de 4 pour notre exemple théorique et au plus 16 pour le processus réel de subdivision des nœuds. Il n'est donc pas possible d'exploiter efficacement des processeurs de plus de 16 cœurs avec cette méthode, même dans le meilleur des cas.

Remarquons par ailleurs que la synchronisation requise à la fin de l'étape de calcul de champ peut limiter l'accélération dans le cas de temps de calcul variables d'un paquet d'échantillons à l'autre. Or, l'algorithme de calcul de champ présenté en partie II peut présenter de très fortes variations de calcul d'un point de champ à l'autre, en fonction de la difficulté à trouver la zone de visibilité du traducteur depuis le point de champ.

En effet, cette barrière de synchronisation empêche la poursuite de l'exploration de l'arbre des subdivisions et donc des calculs supplémentaires. L'accélération théorique $A(N_i)$ obtenue dans cette situation s'écrit en fonction des temps de calcul $T(N_{i,K}$ des paquets d'échantillons :

$$A(N_i) = \frac{\sum_{K=NE,NO,SO,SE} T(N_{i,K})}{\max_{K=NE,NO,SO,SE} T(N_{i,K})}$$

L'accélération est donc très réduite si les différences de temps d'exécution entre les paquets d'échantillons sont grandes. Le diagramme d'ordonnancement présenté en figure 9.3 illustre ce problème.

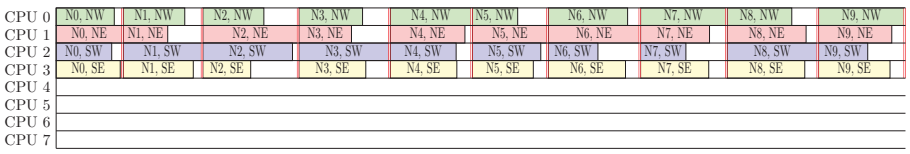


Figure 9.3 – Un exemple d'ordonnancement possible. L'utilisation du CPU est sous-optimale puisque des cœurs ne sont pas utilisés et en raison des barrières de synchronisation

9.2.2 Instances multiples

Principe

Pour pallier ce problème, il est possible de fournir plus de nœuds par étape d'exploration : cela pourrait par exemple consister à autoriser l'exploration d'un nœud et de ses fils en une fois afin de paralléliser le calcul de plus d'échantillons. Il apparaît cependant clair que cette solution ne fait que repousser le problème. Elle permet en effet l'exploitation d'un plus grand nombre de cœurs, mais ce nombre reste limité. De plus, elle reste sensible aux distributions inégales de temps de calcul puisque la barrière de synchronisation reste inchangée.

Une autre solution consiste à autoriser l'exploration simultanée de l'arbre des subdivisions par de multiples instances. De cette façon, plusieurs opérations de calcul de champ peuvent être ordonnées simultanément, permettant à la fois une meilleure occupation des cœurs du processeur et une réduction de l'effet des barrières de synchronisation.

Exemples théoriques

La figure 9.4 donne un exemple possible d'ordonnancement avec deux instances. On y constate que, bien que l'occupation des cœurs du processeur soit meilleure, l'effet des barrières de synchronisation existe toujours. En augmentant le nombre d'instances simultanées, comme le montre l'exemple de la figure 9.5, il peut être efficacement réduit.

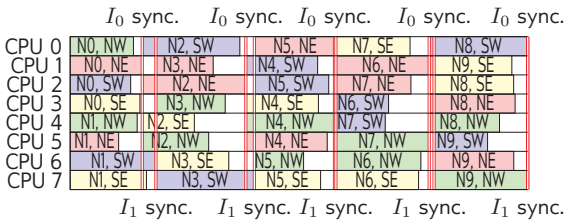


Figure 9.4 – Deux instances simultanées de parcours de l'arbre des subdivisions. L'usage du CPU est meilleur puisque tous les cœurs sont utilisés, mais reste sous-optimal en raison des barrières de synchronisation

En pratique, en autorisant un nombre arbitraire d'instances, à chaque fois qu'un *thread* est inoccupé, il est utilisé pour l'étape 1 d'exploration de l'arbre des subdivisions. En subdivisant un nœud, il génère ainsi des tâches de calcul de champ supplémentaires qui sont distribuées entre les *threads*. De fait, il sort de son état d'inactivité. De cette façon, tant que des nœuds à subdiviser sont disponibles dans la file de priorité, des tâches sont disponibles pour être distribuées entre les *threads* systèmes.

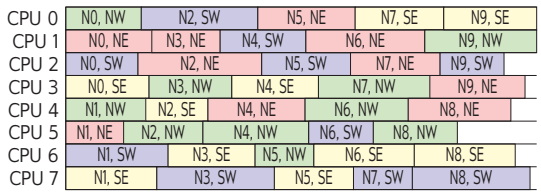


Figure 9.5 – Instances multiples de parcours de l'arbre des subdivisions. Le parallélisme est amélioré et les effets de barrières de synchronisation sont annulés

Croissance exponentielle de la file

Notons que cette stratégie d'instances multiples se base sur le fait que la file de subdivisions contient toujours au moins un nœud à subdiviser. Puisque l'exploration de l'arbre cause une augmentation exponentielle du nombre de nœuds à subdiviser, sa taille est très rapidement suffisante pour que les *threads* n'aient pas à être en attente de tâches.

9.3 Résultats

9.3.1 Tests de mise à l'échelle

Une première mesure que nous pouvons effectuer est la mise à l'échelle afin d'évaluer l'impact de la visualisation progressive sur les performances et l'intérêt de l'exploration de l'arbre des subdivisions par des instances multiples. Pour mesurer l'efficacité du parallélisme, nous comparons les temps de convergence vers une image finale de trois versions différentes de l'outil de visualisation progressive de champ ultrasonore :

Calcul complet : non interactif, parallélisé sur l'ensemble des points de champ, tel que décrit en section 7.2.1.

Calcul progressif avec visualisation : le mode de calcul progressif parallélisé tel que présenté précédemment, ordonnançant les calculs en accord avec un critère de priorité et produisant une visualisation à une cadence d'environ 25 images par seconde.

Calcul progressif sans visualisation : le même mode de calcul sans la visualisation. Il s'agit donc des seules opérations de calcul de champ utilisant l'ordonnancement guidé par le critère de priorité.

La comparaison des temps de convergence vers l'image de champ complète de ces trois versions nous permet de conclure quant à l'effet de l'ordonnancement guidé sur l'efficacité du parallélisme de tâches ainsi que sur le temps de calcul global. La figure 9.6 présente les résultats de ce test de mise à l'échelle et la table 9.1 compare les pourcentages d'efficacité.

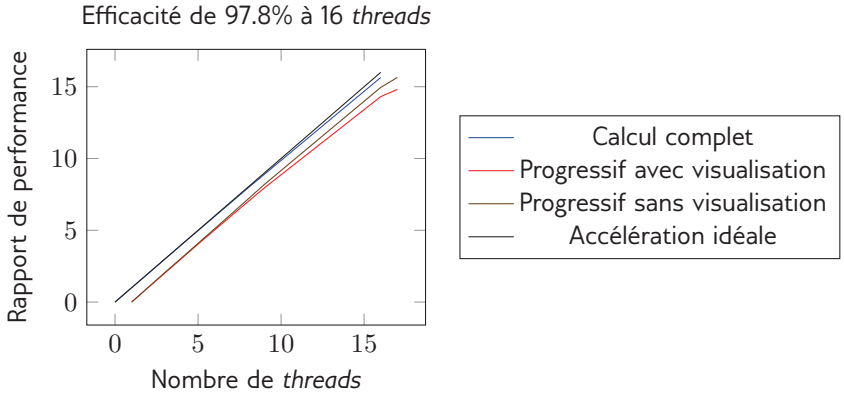


Figure 9.6 – Comparaison de l'accélération liée au parallélisme pour les différents modes de calcul avec le modèle de parallélisme idéal

Mode de calcul	Meilleure accélération	Efficacité (rapport à 24)
Complet	× 15.64	97.8%
Progressif + visu.	× 14.81	92.5%
Progressif seul	× 15.64	97.8%

Table 9.1 – Meilleures accélérations et les efficacités correspondantes pour les différents modes de calcul (16 *threads*)

Les tests sont effectués sur une machine équipée de deux processeurs Intel Xeon E5-2630v3 dotés de 8 cœurs chacun.

Les différences d'accélération entre les versions complète, progressive et progressive avec visualisation sont faibles et mettent en évidence l'efficacité du parallélisme de tâches utilisé pour le calcul de champ : la différence de performances en calcul progressif avec visualisation est nulle sans visualisation et de 5.3% avec visualisation. Les différents cœurs du CPU sont aussi bien utilisés en calcul complet qu'en calcul progressif.

Dans le cas du calcul progressif avec visualisation, le mode d'affichage utilisé implique l'utilisation de la bibliothèque de vision 3D *Visualisation ToolKit* (VTK) (voir Schroeder et collab. (2004)). Ceci fait intervenir une boucle d'affichage qui consomme une partie des ressources CPU disponibles, ce qui explique les performances légèrement inférieures du calcul lorsque l'affichage est activé.

Notons cependant que lorsque le nombre de *threads* systèmes utilisés par ITBB égale le nombre de cœurs du CPU, les ressources de calcul dédiées à l'affichage sont partagées avec le noyau de calcul et, puisque la tâche d'affichage n'est appelée que toutes les 40*ms*, les performances maximales avec la visualisation sont identiques à celles sans visualisation.

Les performances du calcul de champ sont donc très faiblement impactées par l'ordonnancement guidé. Ainsi, pour des performances globales sensiblement égales, le calcul progressif guidé fournit une séquence d'images intermédiaires dès les premiers instants de son fonctionnement pour répondre aux critères d'interactivité. Cette séquence converge plus rapidement vers l'image de champ finale lorsque l'ordonnancement est guidé que lorsqu'il ne l'est pas et le surcoût de ce guidage est très faible.

9.3.2 Tests pratiques

Pour évaluer l'apport pratique de notre approche de visualisation progressive de champ, nous l'utilisons sur des configurations de contrôle non destructif et observons les premières images produites par les approches guidées et non guidées ainsi que le temps de convergence des deux approches. Ces tests sont réalisés sur un CPU de station de travail standard, le *Intel Xeon E5-1650v2@3.5GHz* (6 cœurs).

Nous observons en figure 9.7 que pour les configurations testées, les premières images obtenues en raffinement non guidé sont assez éloignées visuellement du résultat final. À l'inverse, l'approche guidée permet le calcul prioritaire des zones mal approximées de l'image ainsi qu'une convergence rapide autour des zones de forte amplitude.

On constate que les premières images encadrent correctement les zones de forte amplitude. Le résultat produit est rapidement très proche du résultat final dans la zone d'intérêt et les différences les plus visibles sont localisées dans les zones de faible

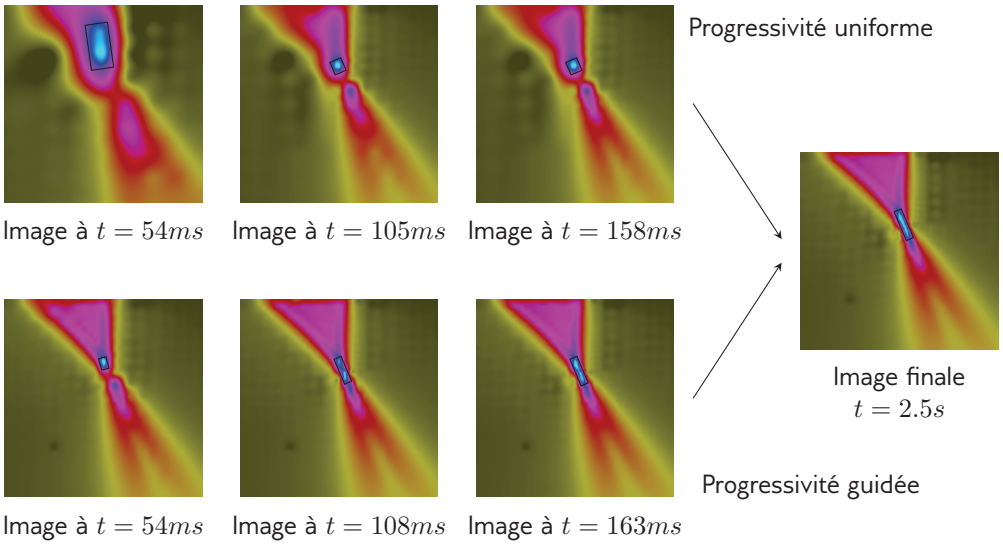


Figure 9.7 – Séquence d'images raffinées uniformément (en haut) et en progressivité guidée (en bas) avec les temps de calcul correspondants

amplitude. On a donc une visualisation interactive d'images de champ utilisable en pratique pour de la simulation de contrôle non destructif sur des configurations dont les temps de calcul complet sont de l'ordre de quelques secondes avec notre outil de calcul de champ rapide.

9.4 Conclusion

Ce chapitre propose une étude du parallélisme de l'algorithme de visualisation progressive de champ proposé dans le chapitre 8. Nous mettons en évidence les problèmes potentiels qu'il peut poser en prévoyant le comportement possible d'un ordonnanceur de tâches.

Suite à cela, nous proposons des méthodes de parallélisation basées sur les modèles de parallélisme usuels en discutant de leur potentiel. Un modèle de parallélisme basé sur un pipeline parallèle est proposé pour exploiter le parallélisme de tâches des processeurs multicœurs.

Enfin, une étude de performance est effectuée pour montrer l'impact faible de l'algorithme de visualisation interactive de champ sur le temps d'exécution de la simulation. Nous mettons également en évidence la bonne mise à l'échelle de cet algorithme, associé au pipeline parallèle.

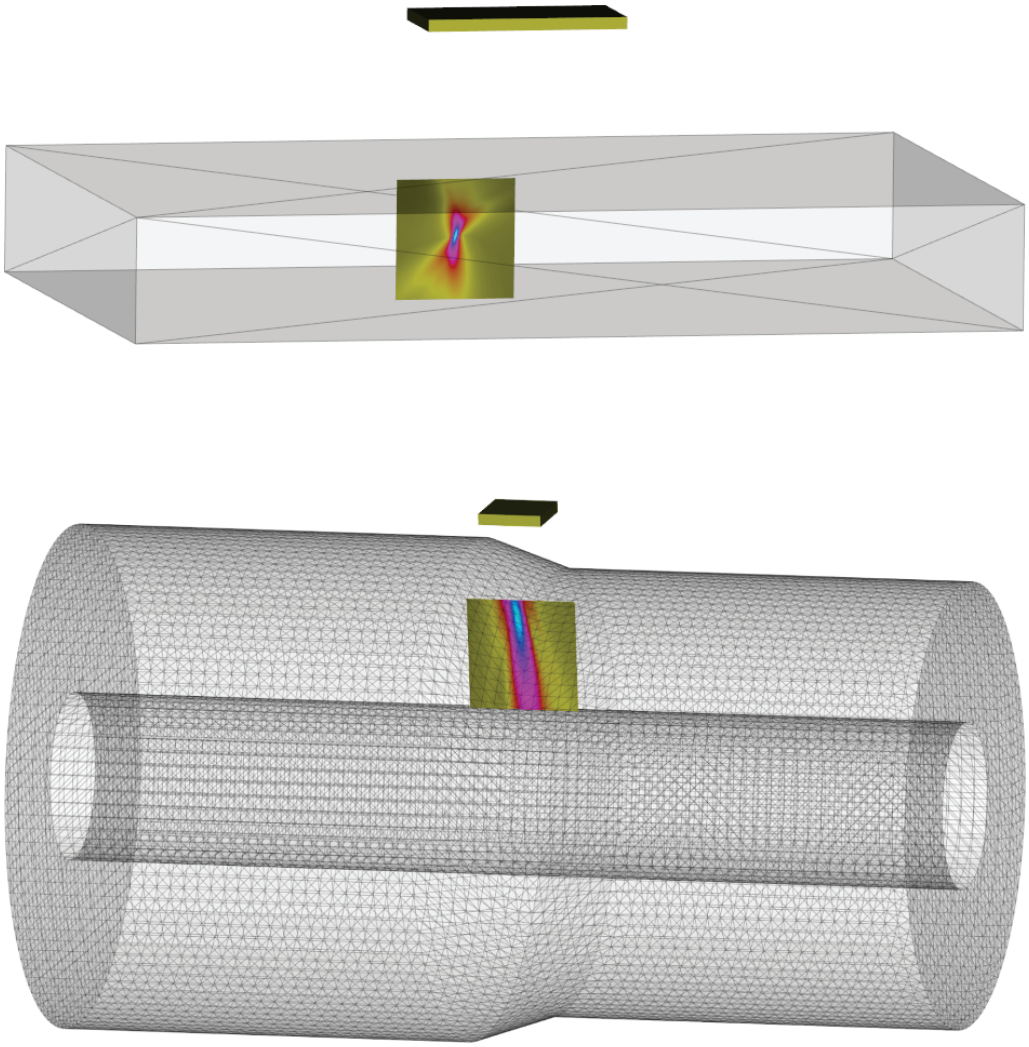


Figure 9.8 – Captures d'écran de l'outil de visualisation interactive de champ ultrasonore

Conclusion générale

Au cours de cette thèse, nous avons mis en œuvre un ensemble d'outils, de méthode et d'algorithmes qui, assemblés, permettent d'atteindre l'objectif final de simulation interactive de champ ultrasonore pour des configurations complexes de contrôle non destructif.

Dans un premier temps, une étude poussée des phénomènes physiques relatifs à la propagation des ondes mécaniques dans les solides et les fluides a été nécessaire afin de modéliser efficacement le comportement des ondes ultrasonores dans les matériaux usuellement contrôlés dans le cadre des inspections.

Suite à cela, nous avons développé un outil complet de lancer de rayons ultrasonores exploitant la bibliothèque *Embree* de lancer de rayons à hautes performances. Cet outil utilise des solutions analytiques aux équations de propagation et de Snell-Descartes en milieux isotropes. Pour les milieux anisotropes, puisqu'aucune solution analytique n'existe à notre connaissance et puisque la méthode numérique s'avère coûteuse, nous avons mis en place une méthode alternative basée l'utilisation de surfaces de lenteur. Cette approche, purement géométrique, exploite de nouveau les capacités de lancer de rayons à hautes performances d'*Embree*.

Partant de cet outil, nous avons étudié le problème du calcul de réponse impulsionnelle pour un traducteur plan de forme quelconque émettant une onde ultrasonore dans un milieu donné. En reprenant le principe de l'approximation paraxiale de la méthode des pinceaux, nous avons développé une méthode d'approximation par des polynômes de degré 3 à deux dimensions du front d'onde ultrasonore. Celle-ci fait l'hypothèse d'un front d'onde lisse et continu et s'avère nettement plus efficace pour construire une réponse impulsionnelle qu'une interpolation linéaire.

Pour systématiser le procédé de construction de la réponse impulsionnelle d'un traducteur ultrasonore plan, l'approche que nous proposons consiste à échantillonner la surface du traducteur avec une finesse suffisante pour respecter le critère d'échantillonnage de signal de Shannon (et donc produire un résultat aussi précis que possible avec un minimum d'échantillons). Chaque point de cet échantillonnage se voit attribuer des valeurs de rayons virtuels au moyen des interpolateurs de pinceaux recouvrant la surface du traducteur, ce qui permet de construire la réponse impulsionnelle de façon optimale à condition que les interpolateurs soient suffisamment précis.

Afin de garantir la précision de ces interpolateurs tout en limitant le nombre de rayons requis pour obtenir une couverture complète de la surface du traducteur par

des pinceaux, un algorithme itératif de recherche de surface a été mis au point. Il se base sur un lancer de pinceaux de rayons successivement raffinés avec un ordre basé sur leur proximité à la surface du traducteur. En se basant sur une hypothèse de continuité du front d'onde, il permet, en exploitant les interpolateurs des pinceaux, de couvrir rapidement la surface du traducteur pour des géométries présentant peu de discontinuités. De plus, une étape de raffinement des pinceaux permet de vérifier l'exactitude des interpolateurs pour la phase de construction des réponses impulsionnelles.

Les éléments ainsi construits permettent de simuler efficacement et avec précision un champ ultrasonore dans un large panel de configuration, incluant des géométries maillées, des matériaux isotropes et anisotropes, homogènes ou hétérogènes et des trajectoires de rayons composées de rebonds. Afin d'améliorer la rapidité de notre outil de simulation, nous avons procédé à une étude de performances qui nous a permis d'identifier les étapes les plus coûteuses du calcul, dans l'objectif de les optimiser en priorité.

Un travail d'optimisation a donc été mené afin d'exploiter au mieux les processeurs généralistes modernes lors des étapes les plus calculatoires. Un parallélisme dynamique de tâches ainsi qu'une gestion adaptée de la mémoire ont été mis en place pour calculer en parallèle le champ ultrasonore sur l'ensemble de la zone de calcul, résultant en de très bons facteurs d'accélération étant donné l'aspect *embarassingly parallel* du problème.

Puis, une phase de vectorisation des codes de calcul intensif présentant la plus grande régularité nous a permis de réduire encore les temps de calcul. En pratique, l'investissement en temps développement pour cette phase peut sembler important en rapport avec le gain de performances obtenu. Cependant, les moyens de vectorisation que nous avons utilisés sont suffisamment peu intrusifs et évolutifs pour permettre l'exploitation de futurs jeux d'instructions permettant le calcul simultané de plus de données. Ainsi, les adaptations à réaliser pour exploiter le jeu d'instructions AVX-512 seront minimes.

L'ensemble des optimisations architecturales et algorithmiques que nous avons menées permet l'exécution rapide d'un grand nombre de simulations de champ ultrasonore. Cependant, pour les configurations complexes visées par ces travaux, l'interactivité en calcul complet reste pour le moment hors de portée de notre code de calcul. Des optimisations supplémentaires sur certaines phases du calcul de champ devraient permettre d'élargir le nombre de configurations calculables interactivement.

Partant du constat que les images de champ sont souvent suréchantillonnées et contiennent une information qui peut être exprimée avec moins de points de champ que n'en contient leur définition, nous avons privilégié le développement d'un outil de visualisation interactive de champ, notamment pour les configurations dont les temps de calcul complet excèdent les limites de l'interactivité. Exploitant les propriétés de régularité des images de champ, celui-ci produit une séquence d'images interpolées à partir d'échantillonnages incomplets du champ convergeant vers l'image complète.

Cette convergence est accélérée par des critères de raffinement, rendant le processus global de calcul de champ équivalent à une exploration d'arbre de subdivisions. Pour conserver les bons rapports d'accélération sur processeurs multicœurs (le problème n'étant plus *embarassingly parallel*), nous avons mis en place un pipeline parallèle basé sur le modèle de parallélisme de tâches de la bibliothèque *Intel TBB*. Le résultat final est une visualisation progressive d'images de champ ultraso-

nore convergeant vers l'image finale avec un surcoût par rapport au calcul complet du champ comparable à la variance usuelle du temps de calcul.

Perspectives

De multiples pistes de recherche et de développement sont envisageables pour améliorer les performances et élargir le domaine de validité de notre méthode. En particulier, l'algorithme de recherche de surface que nous avons développé a vocation à être étendu à des géométries moins régulières, par exemple au moyen de méthodes de type Montecarlo.

Par ailleurs, le calcul de dérivées par différences finies pourrait se heurter dans le cas de pinceaux très fins aux limites de précision de la représentation des réels en flottants à simple précision. Pour cette raison, l'utilisation de méthodes de différenciation automatique (voir Piponi (2004)) pour pouvoir calculer de manière plus précise les dérivées des quantités des rayons permettrait d'améliorer signficiativement la qualité des interpolateurs dans ces situations.

L'amélioration des performances pures pourrait passer par une vectorisation de sections de code pour le moment scalaires telles que la recherche de pinceau recouvrant ou le calcul et la sommation des réponses impulsionnelles. De plus, l'utilisation des architectures de type GPU ou de coprocesseurs de calcul augmenterait sans nul doute les performances du calcul à l'aide d'un fonctionnement hybride avec le CPU.

Les applications des travaux effectués au cours de cette thèse sont nombreuses dans le domaine du contrôle non destructif. En effet, les temps de calcul atteints pour les configurations complexes qu'ils couvrent rendent possible une utilisation plus large de la simulation de contrôle non destructif. En particulier, la visualisation interactive de champ ultrasonore permettra la conception de méthodes de contrôle non destructif (positionnement, définition des lois de retard...) et de traducteurs beaucoup plus rapidement.

De plus, des applications en calcul d'échos de défaut peuvent être envisagées, au moyen d'un couplage avec des modèles de diffraction d'ondes ultrasonores. L'utilisation de certaines parties des codes de simulation que nous avons développés est aussi possible pour des applications de simulation de techniques de thérapie médicale utilisant des ondes ultrasonores focalisées pour échauffer et détruire des tumeurs.

L'outil de visualisation interactive de champ que nous avons développé au cours de ces travaux s'applique sans effort à tout type de simulation de champ présentant des propriétés de régularité suffisantes pour que les hypothèses que nous avons formulées s'appliquent. Une application directe de cette méthode dans le cadre du contrôle non destructif pourrait être de s'en servir avec d'autres méthodes de simulation dont les temps de calcul sont actuellement trop longs pour permettre l'interactivité, tels que des codes à éléments finis ou des outils de reconstruction.

Bibliographie

319433-011. 2011, *Intel Advanced Vector Extensions Programming Reference*, Intel Corporation.

319433-012A. 2012, *Intel Architecture Instruction Set Extensions Programming Reference*, Intel Corporation.

Amanatides, J. 1984, «Ray tracing with cones», dans *ACM SIGGRAPH Computer Graphics*, vol. 18, ACM, p. 129–135.

Appel, A. 1968, «Some techniques for shading machine renderings of solids», dans *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, ACM, p. 37–45.

Atkinson, K. E. 1980, *An Introduction to Numerical Analysis*, John Wiley & Sons.

Broyden, C. G. 1965, «A class of methods for solving nonlinear simultaneous equations», *Mathematics of Computation*, vol. 19, n° 92, p. 577–593. URL `http://www.jstor.org/stable/2003941`.

Card, S. K., G. G. Robertson et J. D. Mackinlay. 1991, «The information visualizer, an information workspace», dans *Proceedings of the SIGCHI Conference on Human factors in computing systems*, ACM, p. 181–186.

Catmull, E. et J. Clark. 1978, «Recursively generated b-spline surfaces on arbitrary topological meshes», *Computer-aided design*, vol. 10, n° 6, p. 350–355.

Červenỳ, V. 1972, «Seismic rays and ray intensities in inhomogeneous anisotropic media», *Geophysical Journal International*, vol. 29, n° 1, p. 1–13.

Clarberg, P. 2008, «Fast equal-area mapping of the (hemi) sphere using simd», *Journal of Graphics, GPU, and Game Tools*, vol. 13, n° 3, p. 53–68.

D91561-003. 2007, *Intel SSE4 Programming Reference*, Intel Corporation.

Delaunay, B. 1934, «Sur la sphère vide. A la mémoire de Georges Voronoï», *Bulletin de l'Académie des Sciences de l'URSS*, , n° 6, p. 793–800. URL `http://www.mathnet.ru/php/archive.phtml?wshow=paper&#38;jrnid=im&#38;paperid=4937&#38;option_lang=eng`.

Deschamps, G. A. 1972, «Ray techniques in electromagnetics», *Proceedings of the IEEE*, vol. 60, n° 9, p. 1022–1035.

Estérie, P., J. Falcou, M. Gaunard et J.-T. Lapresté. 2014, «Boost. simd : generic programming for portable simdization», dans *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing*, ACM, p. 1–8.

Fog, A. 2016, «Vector class library», URL `http://www.agner.org/optimize/#vectorclass`.

Frey, P. J. et H. Borouchaki. 2003, «Surface meshing using a geometric error estimate», *International Journal for Numerical Methods in Engineering*, vol. 58, n° 2, doi : 10.1002/nme.766, p. 227–245, ISSN 1097-0207. URL `http://dx.doi.org/10.1002/nme.766`.

Gengembre, N. 1999, *Modélisation du champ ultrasonoré rayonne dans un solide anisotrope et hétérogene par un traducteur immergé*, thèse de doctorat, Université Paris-VII.

Gengembre, N., A. Lhémery et P. Calmon. 2006, *Matériaux et acoustique 2 : propagation des ondes acoustiques 2*, Lavoisier.

Giunta, A. A., L. T. Watson et J. Koehler. 1998, «A comparison of approximation modeling techniques : polynomial versus interpolating models», *AIAA paper*, , n° 98-4758.

Gouraud, H. 1971, «Continuous shading of curved surfaces», *IEEE transactions on computers*, vol. 100, n° 6, p. 623–629.

Heckbert, P. S. et P. Hanrahan. 1984, «Beam tracing polygonal objects», *ACM SIGGRAPH Computer Graphics*, vol. 18, n° 3, p. 119–127.

Hou, H. et H. Andrews. 1978, «Cubic splines for image interpolation and digital filtering», *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 26, n° 6, p. 508–517.

Huthwaite, P. 2014, «Accelerated finite element elastodynamic simulations using the GPU», *Journal of Computational Physics*, vol. 257, Part A, n° 0, doi :http://dx.doi.org/10.1016/j.jcp.2013.10.017, p. 687 – 707, ISSN 0021-9991. URL `http://www.sciencedirect.com/science/article/pii/S0021999113006931`.

IEEE. 2008, «IEEE standard for floating-point arithmetic», *IEEE Std 754-2008*, doi : 10.1109/IEEESTD.2008.4610935, p. 1–70.

Kinghorn, D. 2014, «Hyper-threading may be killing your parallel performance», URL `https://www.pugetsystems.com/labs/hpc/Hyper-Threading-may-be-Killing-your-Parallel-Performance-578/`.

Knuth, D. E. 1984, «Literate programming», *The Computer Journal*, vol. 27, n° 2, p. 97–111.

Lambert, J. 2015, *Parallelization of ultrasonic field simulations for non destructive testing*, Theses, Université Paris Sud - Paris XI. URL `https://tel.archives-ouvertes.fr/tel-01239899`.

Lavoué, G., M.-C. Larabi et L. Vasa. 2015, «On the Efficiency of Image Metrics for Evaluating the Visual Quality of 3D Models», *IEEE Transactions on Visualization and Computer Graphics*, doi :10.1109/TVCG.2015.2480079. URL https://hal.archives-ouvertes.fr/hal-01205349.

Lee, S., G. Wolberg et S. Y. Shin. 1997, «Scattered data interpolation with multilevel b-splines», *Visualization and Computer Graphics, IEEE Transactions on*, vol. 3, n° 3, p. 228–244.

McCool, M., J. Reinders et A. Robison. 2012, *Structured Parallel Programming*, Elsevier.

Miller, R. B. 1968, «Response time in man-computer conversational transactions», dans *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, ACM, p. 267–277.

Möller, T. et B. Trumbore. 2005, «Fast, minimum storage ray/triangle intersection», dans *ACM SIGGRAPH 2005 Courses*, ACM, p. 7.

Moore, G. E. 1995, «Lithography and the future of moore's law», doi :10.1117/12.209244. URL http://dx.doi.org/10.1117/12.209244.

Nagata, T. «Simple local interpolation of surfaces using normal vectors», vol. 22, n° 4, doi :10.1016/j.cagd.2005.01.004, p. 327–347, ISSN 01678396. URL http://linkinghub.elsevier.com/retrieve/pii/S016783960500021X.

Nayfeh, A. H. 1991, «Elastic wave reflection from liquid-anisotropic substrate interfaces», *Wave motion*, vol. 14, n° 1, p. 55–67.

Ogilvy, J. 1985, «Computerized ultrasonic ray tracing in austenitic steel», *NDT international*, vol. 18, n° 2, p. 67–77.

Parker, S. G., J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison et collab.. 2010, «Optix : a general purpose ray tracing engine», dans *ACM Transactions on Graphics (TOG)*, vol. 29, ACM, p. 66.

Pedron, A. 2013, *Développement d'algorithmes d'imagerie et de reconstruction sur architectures à unités de traitement parallèles pour des applications en contrôle non destructif*, thèse de doctorat. URL http://www.theses.fr/2013PA112071.

Pharr, M. et G. Humphreys. 2004, *Physically based rendering : From theory to implementation*, Morgan Kaufmann.

Phong, B. T. 1975, «Illumination for computer generated pictures», *Communications of the ACM*, vol. 18, n° 6, p. 311–317.

Piegl, L. et W. Tiller. 2012, *The NURBS book*, Springer Science & Business Media.

Piponi, D. 2004, «Automatic differentiation, c++ templates, and photogrammetry», *Journal of Graphics Tools*, vol. 9, n° 4, p. 41–55.

Praun, E. et H. Hoppe. 2003, «Spherical parametrization and remeshing», dans *ACM Transactions on Graphics (TOG)*, vol. 22, ACM, p. 340–349.

Press, W. H. 2007, *Numerical recipes 3rd edition : The art of scientific computing*, Cambridge university press.

Apache Software Foundation. 2004, «Apache license, version 2.0», URL http://www.apache.org/licenses/LICENSE-2.0.

Intel Corporation. 2011, «Embree», URL https://embree.github.io.

Intel Corporation. 2016a, «Intel intrinsics guide», URL https://software.intel.com/sites/landingpage/IntrinsicsGuide.

Intel Corporation. 2016b, «Intel thread building blocks», URL https://www.threadingbuildingblocks.org.

Ribeiro Monteiro, H. 1992, *Étude de la réfexion-transmission des ondes ultrasonores en présence d'une interface séparant deux milieux anisotropes : prise en compte des ondes inhomogènes*, thèse de doctorat. URL http://www.theses.fr/1992COMPD506.

Royer, D. et E. Dieulesaint. 1996, «Ondes élastiques dans les solides, tome 1 : Propagation libre et guidée», .

Schroeder, W. J., B. Lorensen et K. Martin. 2004, *The visualization toolkit*, Kitware.

Shinya, M., T. Takahashi et S. Naito. 1987, «Principles and applications of pencil tracing», dans *ACM SIGGRAPH Computer Graphics*, vol. 21, ACM, p. 45–54.

Shirley, P. et K. Chiu. 1997, «A low distortion map between disk and square», *Journal of graphics tools*, vol. 2, nº 3, p. 45–52.

Snyder, L. 1988, «A taxonomy of synchronous parallel machines», cahier de recherche, DTIC Document.

Szalay, A. S., J. Gray, G. Fekete, P. Z. Kunszt, P. Kukol et A. Thakar. 2007, «Indexing the sphere with the hierarchical triangular mesh», *arXiv preprint cs/0701164*.

Tchebychev, P. L. 1899, *Œuvres*, vol. 1, St. Petersbourg, Commissionaires de l'Academie imperiale des sciences. https://archive.org/details/117744684_001.

Valm2410. *via Wikimedia Commons, sous licence CC BY-SA 4.0*. https://commons.wikimedia.org/wiki/File:SHADING.GIF.

Van Overveld, C. W. A. M. et B. Wyvill. «Phong normal interpolation revisited», vol. 16, nº 4, doi :10.1145/263834.263849, p. 397–419, ISSN 0730-0301. URL http://doi.acm.org/10.1145/263834.263849.

Vavryčuk, V. 2006, «Calculation of the slowness vector from the ray vector in anisotropic media», dans *Proceedings of the Royal Society of London A : Mathematical, Physical and Engineering Sciences*, vol. 462, The Royal Society, p. 883–896.

Wald, I. 2007, «On fast construction of sah-based bounding volume hierarchies», dans *2007 IEEE Symposium on Interactive Ray Tracing*, IEEE, p. 33–40.

Wang, Z., A. C. Bovik, H. R. Sheikh et E. P. Simoncelli. 2004, «Image quality assessment : from error visibility to structural similarity», *IEEE transactions on image processing*, vol. 13, nº 4, p. 600–612.

Woop, S., C. Benthin, I. Wald, G. S. Johnson et E. Tabellion. 2014, «Exploiting local orientation similarity for efficient ray traversal of hair and fur.», dans *High Performance Graphics*, p. 41–49.

Woop, S., L. Feng, I. Wald et C. Benthin. 2013, «Embree ray tracing kernels for cpus and the xeon phi architecture», dans *ACM SIGGRAPH 2013 Talks*, ACM, p. 44.

ANNEXE A

Résultats de validation et de performance

Dans cette section, nous présentons les résultats de validité du calcul de champ que nous avons développé au cours de ces travaux. Puis, nous détaillons ses performances en temps de calcul pour des configurations variées. Le processeur de calcul utilisé est un Intel Xeon E5-1650v2@3.50 GHz (6 cœurs), qui correspond à une station de travail standard.

Dans un premier temps, nous montrons des résultats de validation détaillés par rapport aux résultats produits par *CIVA* sur la même configuration en précisions [1] 1 et 10. Nous y présentons également les temps de calcul avec notre outil et avec *CIVA*. Puis, pour les configurations suivantes, nous présentons un comparatif plus succint des temps de calcul et des résultats de validité du calcul de champ.

1. La précision de calcul *CIVA* correspond à la finesse du balayage en pinceaux. La précision 10 permet d'obtenir un résultat ayant convergé.

Configuration n° 1 : Pièce plane

Caractéristiques

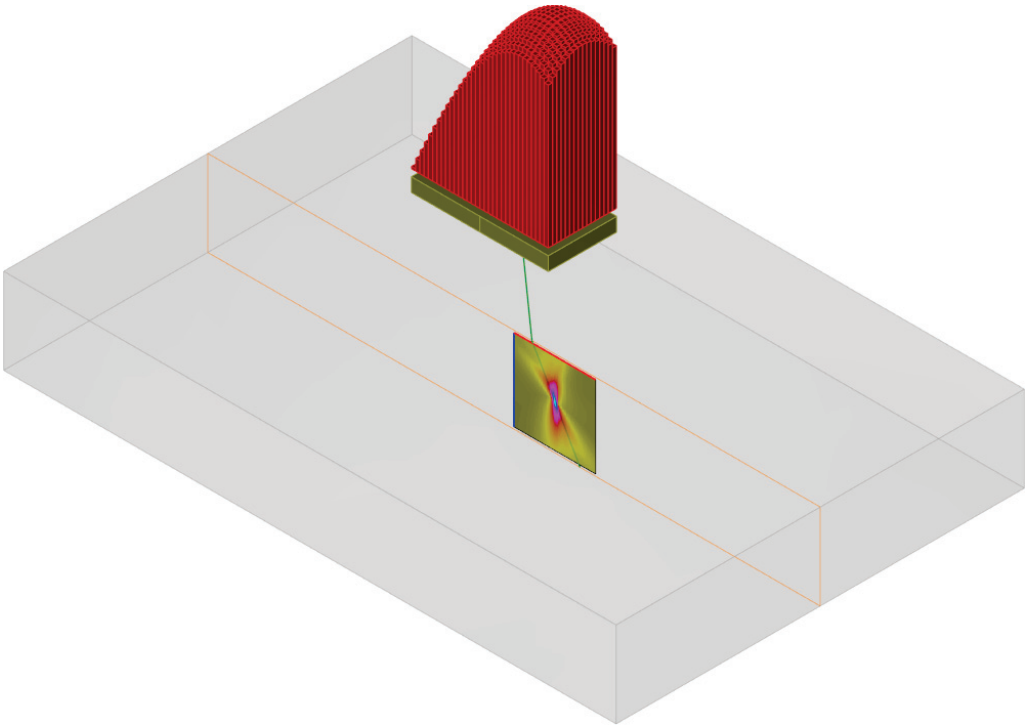


Figure A.1 – Configuration de contrôle n° 1

Pièce Plane, $300mm \times 200mm \times 42mm$ maillage de 24 triangles (pièce analytique pour *CIVA*).

Traducteur $67,1mm \times 33.5mm$, 512 éléments.

Milieu Acier isotrope.

Inspection Directe, mode L.

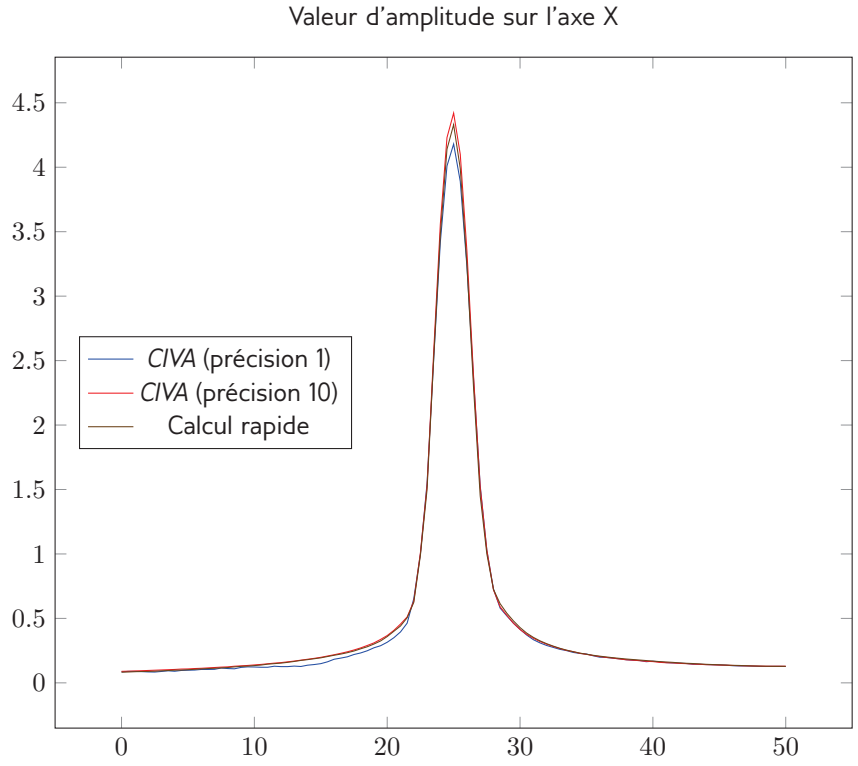
Fréquence maximale 5 MHz.

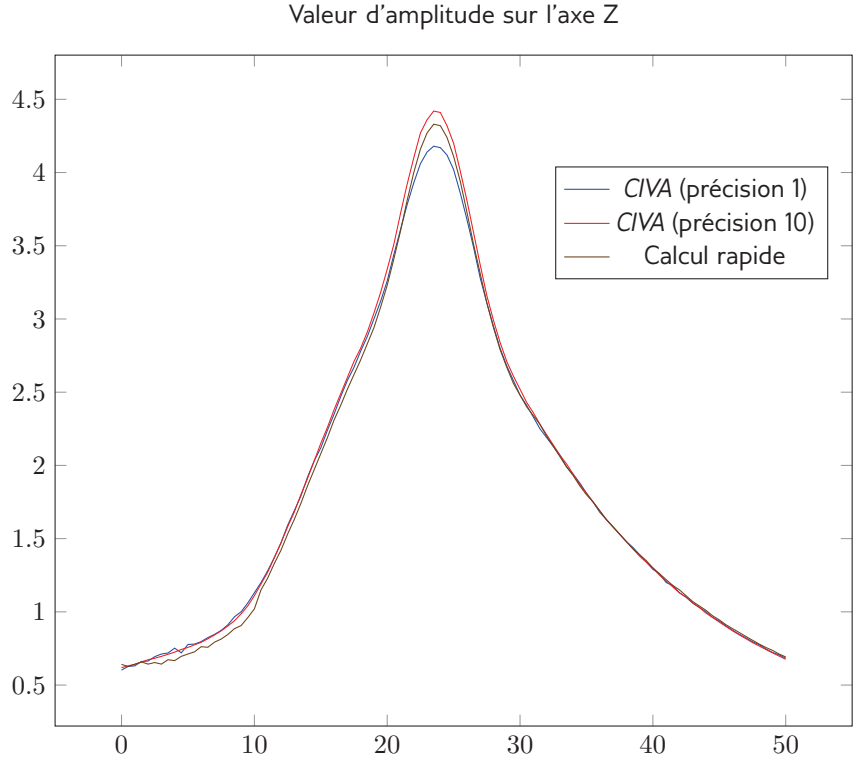
Fréquence d'échantillonnage 90 MHz.

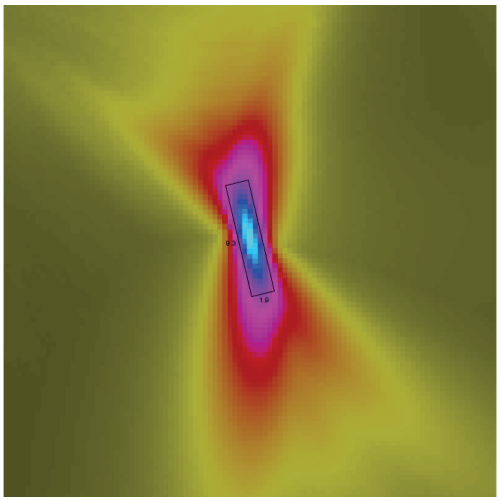
Zone 10 201 points.

Comparatif

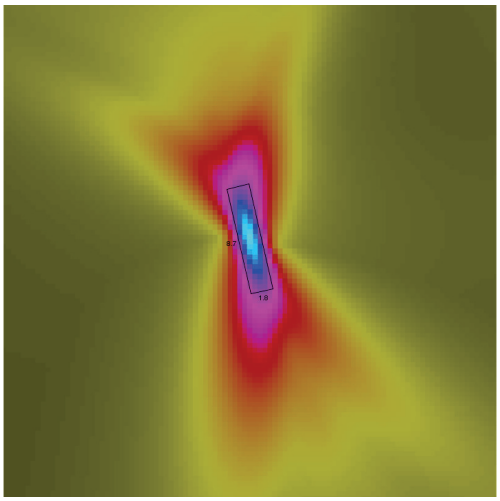
Outil de calcul	Temps (s)	Erreur (dB)
CIVA (précision 10)	288	0
CIVA (précision 1)	151	0.5
Calcul rapide	7.2	0.2


Valeur d'amplitude sur l'axe X
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
CIVA (précision 1)
CIVA (précision 10)
Calcul rapide

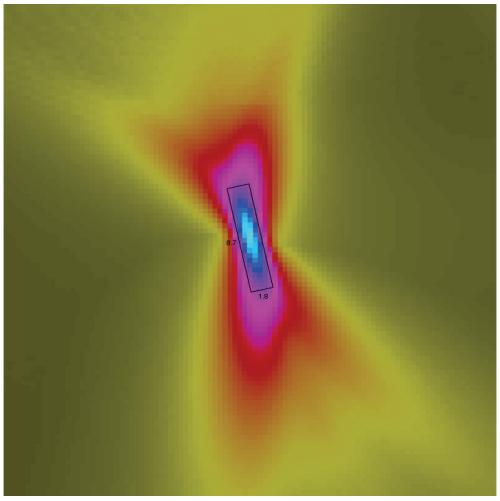

Valeur d'amplitude sur l'axe Z
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
CIVA (précision 1)
CIVA (précision 10)
Calcul rapide



(a)



(b)



(c)

Figure A.2 – Images de champ et tâches focales produites par CIVA en précision 1 **(a)**, en précision 10 **(b)** et par notre code de calcul rapide **(c)**

Configuration n° 2 : Cylindre à variation de section

Caractéristiques

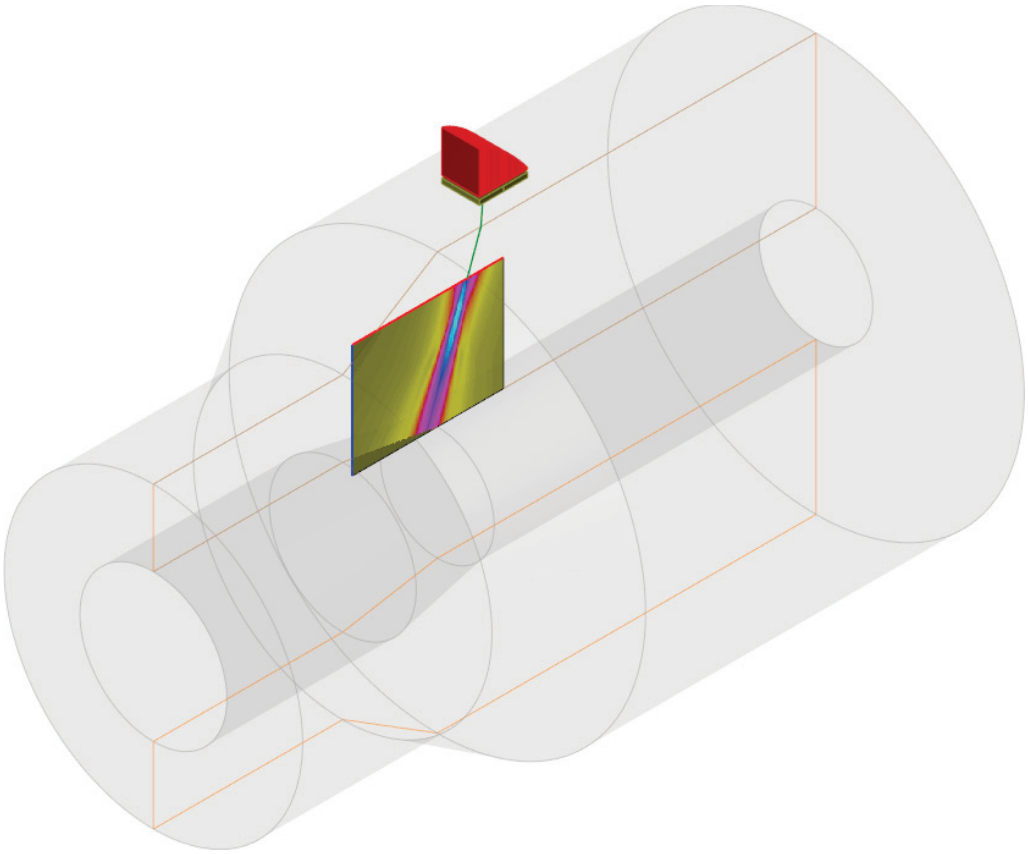


Figure A.3 – Configuration de contrôle n° 1

Pièce Cylindre à variation de section de 350mm de longueur, maillage de 552 616 triangles (pièce analytique pour *CIVA*).

Traducteur $25,5mm \times 20mm$, 32 éléments.

Milieu Acier isotrope.

Inspection Directe, mode L.

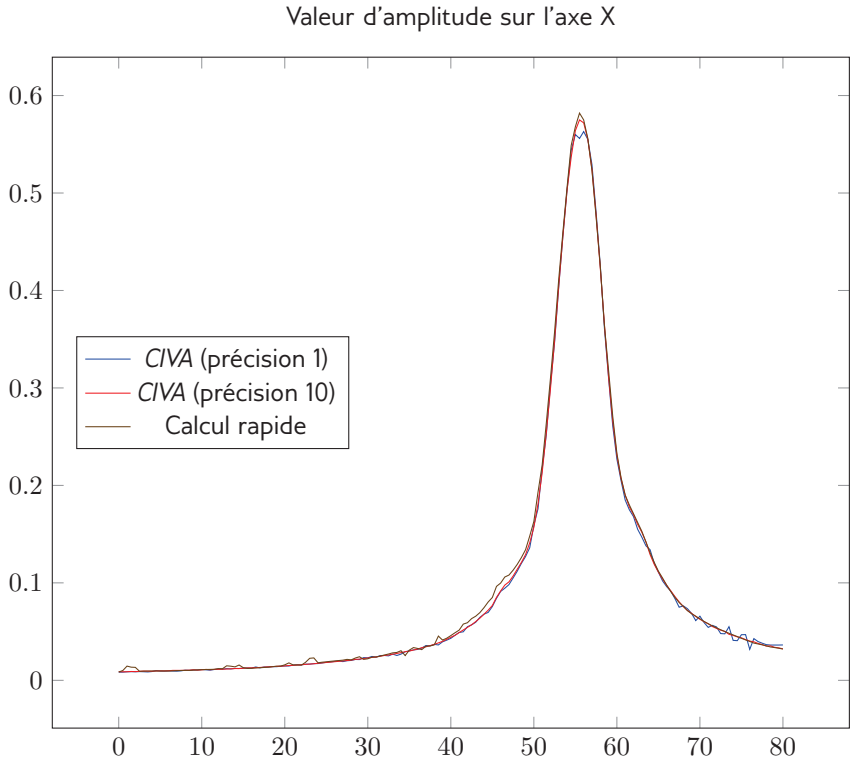
Fréquence maximale 4 MHz.

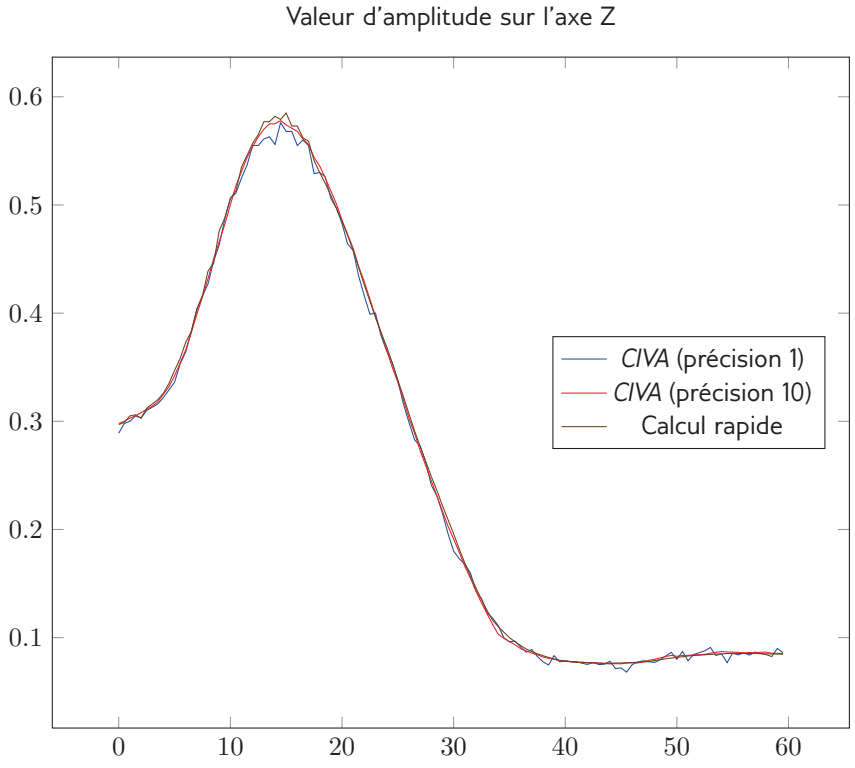
Fréquence d'échantillonnage 70 MHz.

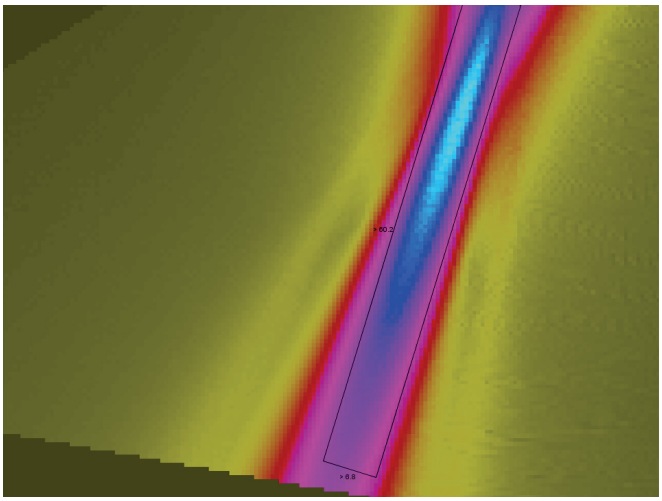
Zone 19 481 points.

Comparatif

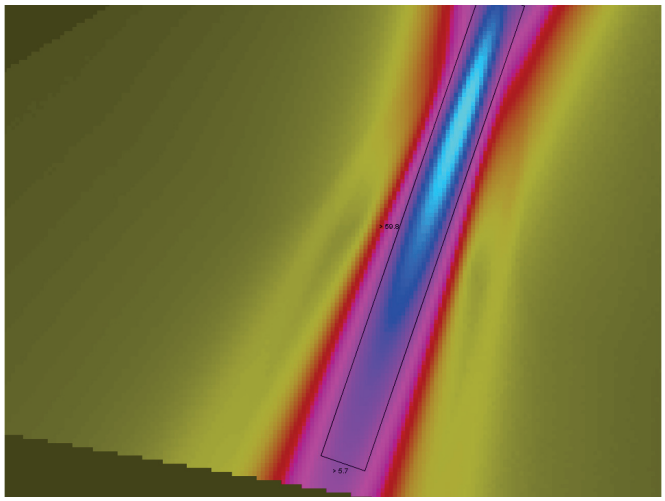
Outil de calcul	Temps (s)	Erreur (dB)
CIVA (précision 10)	491	0
CIVA (précision 1)	44	0
Calcul rapide	3.7	0


Valeur d'amplitude sur l'axe X
0.6
0.5
0.4
0.3
0.2
0.1
0
0
10
20
30
40
50
60
70
80
CIVA (précision 1)
CIVA (précision 10)
Calcul rapide

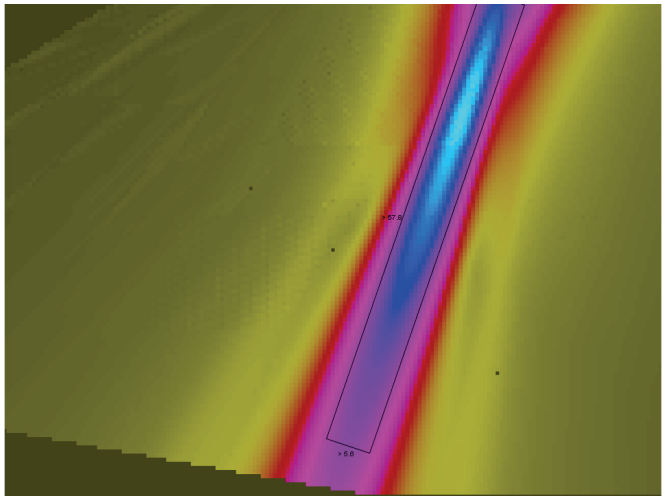

Valeur d'amplitude sur l'axe Z
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
60
CIVA (précision 1)
CIVA (précision 10)
Calcul rapide



(a)



(b)



(c)

Figure A.4 – Images de champ et tâches focales produites par CIVA en précision 1 **(a)**, en précision 10 **(b)** et par notre code de calcul rapide **(c)**

Configuration n° 3 : Pièce plane anisotrope

Caractéristiques

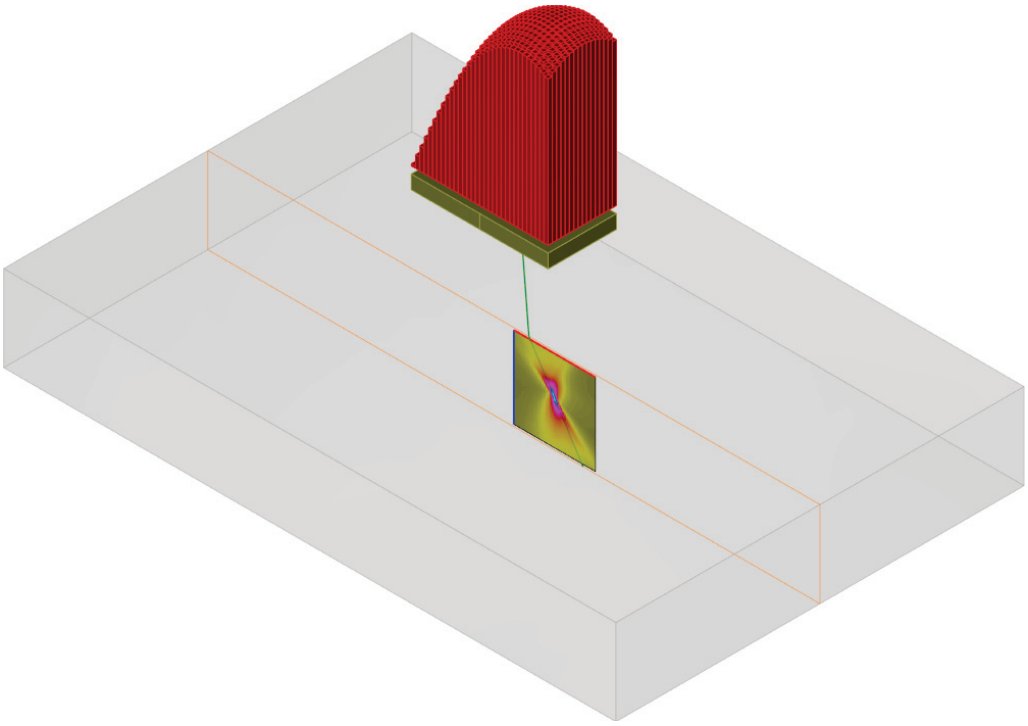


Figure A.5 – Configuration de contrôle n° 3, équivalent anisotrope de la configuration n° 1

Pièce Identique à la configuration n° 1.

Traducteur $67,1mm \times 33.5mm$, 512 éléments.

Milieu Acier anisotrope.

Inspection Directe, mode qL.

Fréquence maximale 5 MHz.

Fréquence d'échantillonnage 90 MHz.

Zone 10 201 points.

Comparatif

Outil de calcul	Temps (s)	Erreur (dB)
CIVA (précision 10)	288	0
CIVA (précision 1)	98	0.5
Calcul rapide	20.6	0

On constate que, malgré la complexité supplémentaire induite par l'anisotropie des matériaux lors de la phase de lancer de pinceaux, le rapport de temps d'exécution entre une configuration avec une pièce anisotrope et son équivalent isotrope est de l'ordre de 3.

Configuration n° 4 : Pièce plane avec rebond, équivalent de la configuration n° 1 avec rebond

Caractéristiques

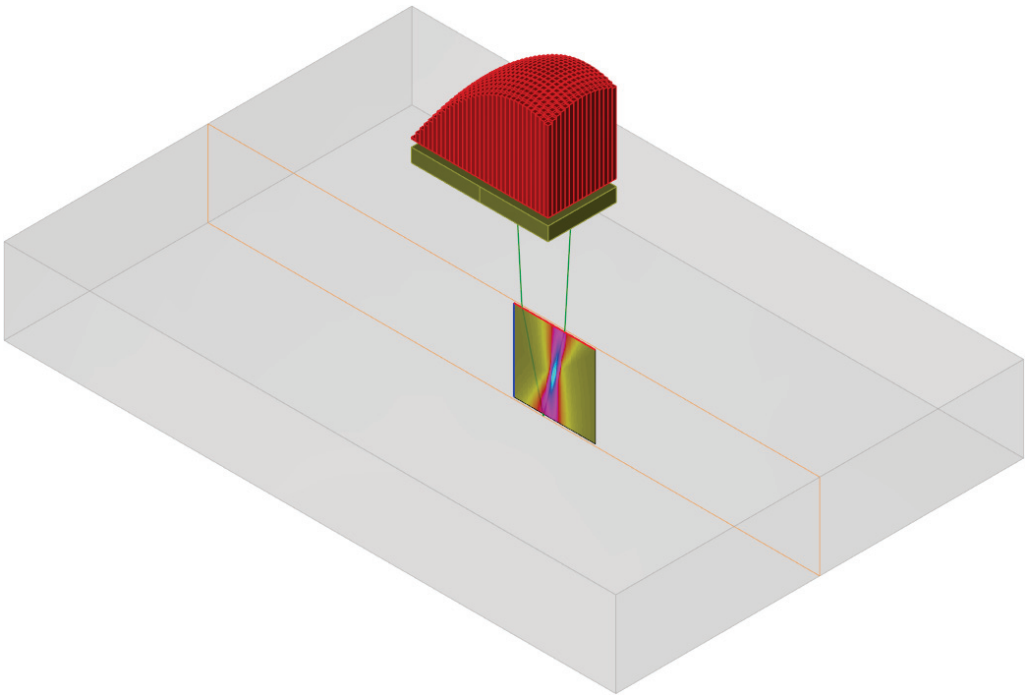


Figure A.6 – Configuration de contrôle n° 4

Pièce Identique à la configuration n° 1.

Traducteur $67,1mm \times 33.5mm$, 512 éléments.

Milieu Acier isotrope.

Inspection Avec rebond, mode qL.

Fréquence maximale 5 MHz.

Fréquence d'échantillonnage 90 MHz.

Zone 10 201 points.

Comparatif

Outil de calcul	Temps (s)	Erreur (dB)
CIVA (précision 10)	381	0
CIVA (précision 1)	121	0.1
Calcul rapide	6.1	0.1

Le temps de calcul pour une simulation d'inspection avec rebond est, dans cette situation, très proche et inférieur au temps de simulation de l'inspection directe. Ceci s'explique par le fait que, dans ce cas précis, le rebond, augmentant la distance de proppagation et donc la divergence des pinceaux à l'arrivée, facilite la recherche du traducteur.

Configuration n° 5 : Cylindre à variation de section avec rebond

Caractéristiques

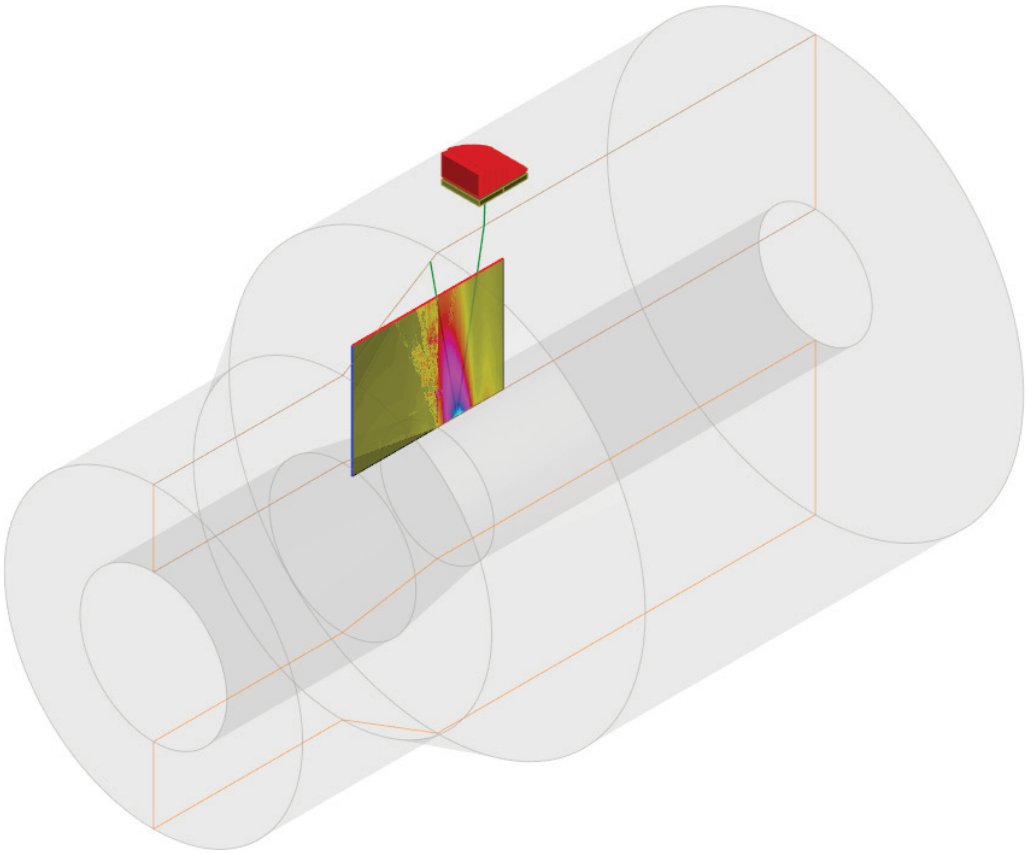


Figure A.7 – Configuration de contrôle n° 5

Pièce Identique à la configuration n° 2.

Traducteur $25,5mm \times 20mm$, 32 éléments.

Milieu Acier isotrope.

Inspection Avec rebond, mode L.

Fréquence maximale 4 MHz.

Fréquence d'échantillonnage 70 MHz.

Zone 19 481 points.

Comparatif

Outil de calcul	Temps (s)	Erreur (dB)
CIVA (précision 10)	426	0
CIVA (précision 1)	55	0.1
Calcul rapide	5.6	0.1

Des artefacts apparaissent au niveau de l'angle de jonction entre la partie conique et la partie cylindrique de la pièce. Ils sont dus à des trous topologiques dans le maillage utilisé.

Configuration n° 6 : Cylindre à variation de section anisotrope

Caractéristiques

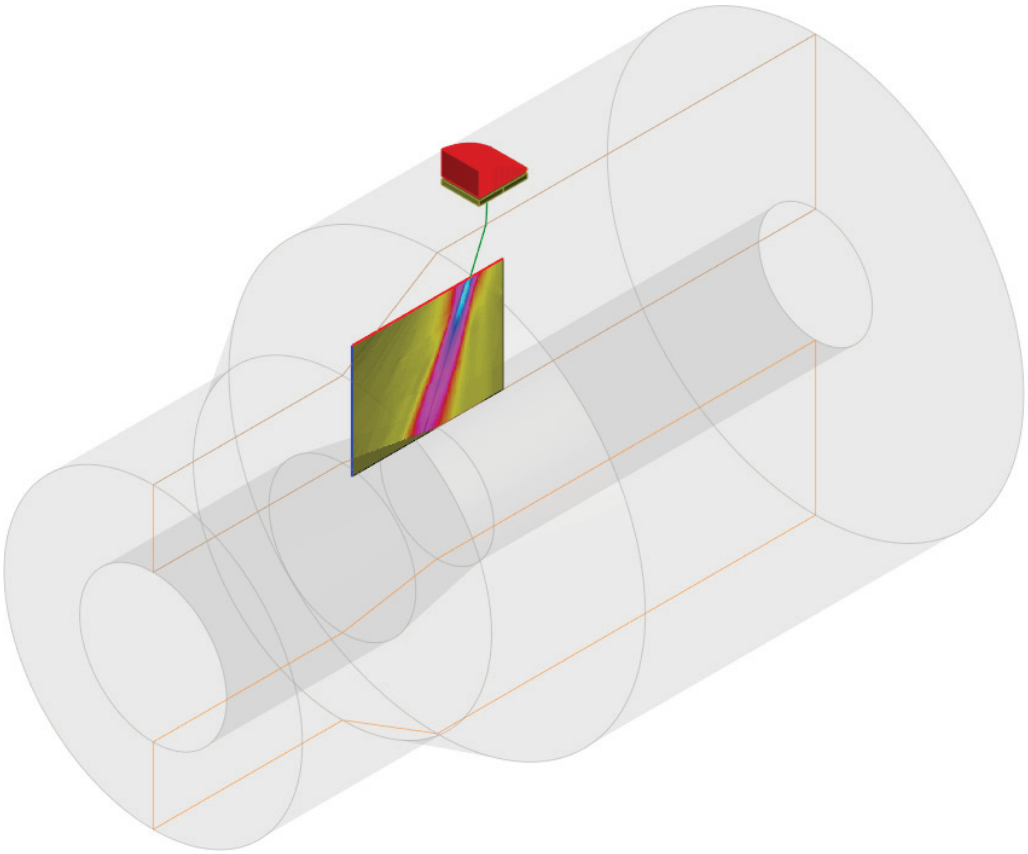


Figure A.8 – Configuration de contrôle n° 6

Pièce Identique à la configuration n° 2.

Traducteur $25,5mm \times 20mm$, 32 éléments.

Milieu Acier anisotrope.

Inspection Directe, mode L.

Fréquence maximale 4 MHz.

Fréquence d'échantillonnage 70 MHz.

Zone 19 481 points.

Comparatif

Outil de calcul	Temps (s)	Erreur (dB)
CIVA (précision 10)	711	0
CIVA (précision 1)	75	0.2
Calcul rapide	8.5	0.1

Configuration n° 7 : CAO de tuyau coudé

Caractéristiques

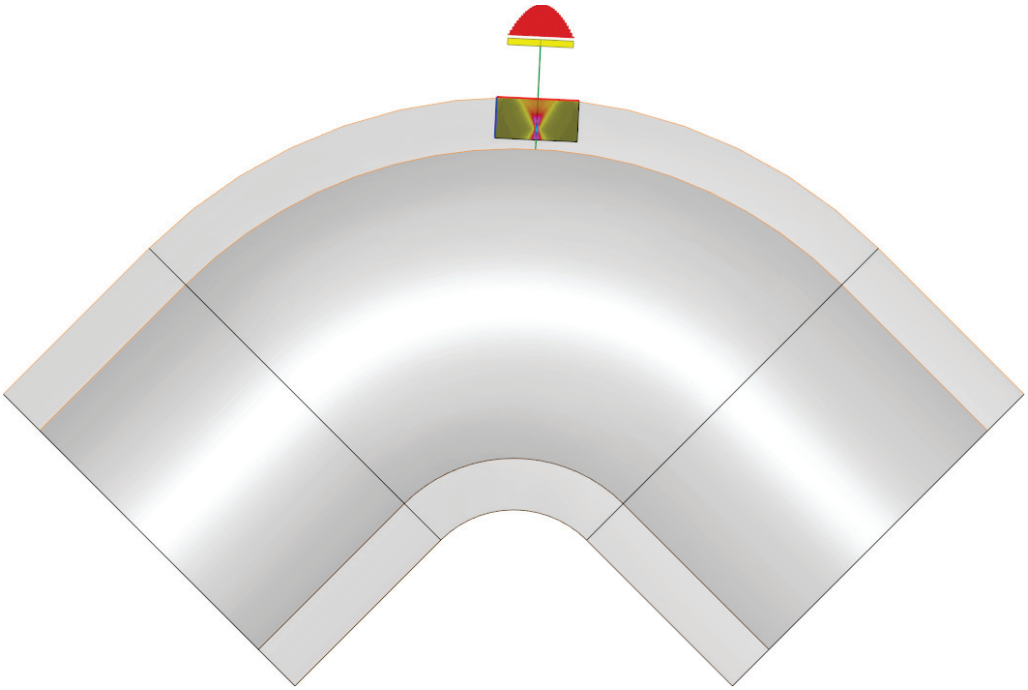


Figure A.9 – Configuration de contrôle n° 7

Pièce Tuyau coudé de 300mm de rayon, maillage de 4 639 760 triangles (même maillage pour *CIVA*).

Traducteur $63,9mm \times 20mm$, 32 éléments.

Milieu Acier anisotrope.

Inspection Directe, mode L.

Fréquence maximale 2 MHz.

Fréquence d'échantillonnage 60 MHz.

Zone 13 041 points.

Comparatif

Outil de calcul	Temps (s)	Erreur (dB)
CIVA (précision 10)	813	0
CIVA (précision 1)	83	0.2
Calcul rapide	4.5	0.1

Les temps de calcul pour cette configuration restent proches de ceux obtenus pour les configurations précédentes, mettant en jeu des maillages avec nettement moins de triangles. Ceci met en évidence l'effet de l'utilisation des structures accélératrices lors de la phase de lancer de rayons.

ANNEXE B

Calculs flottants

Représentation des nombres réels

Les processeurs, tels que nous les présentons en section 7.1.1, ont pour caractéristique de pouvoir effectuer des calculs rapides et sans erreur sur les nombres entiers, jusqu'aux limites de représentation. Dans le cadre de ces calculs, les nombres sont représentés sous forme binaire et les opérations sont effectuées sur ces représentations au moyen des **unités arithmétiques et logiques**.

La quantité d'espace mémoire utilisée pour la représentation d'un entier en mémoire étant limitée, l'ensemble des nombres qu'il est possible de représenter est borné. Ainsi, par exemple, le type entier non signé sur 32 bits permet de représenter tous les nombres de l'ensemble $[\![0, 2^{32} - 1]\!]$.

Pour réaliser des calculs sur des réels, la représentation des nombres est nécessairement approximative. En effet, le cardinal d'un intervalle réel non réduit à un singleton est infini et l'espace mémoire disponible pour représenter un nombre est limité.

La méthode de représentation la plus utilisée pour représenter des nombres réels est la méthode dite de la « virgule flottante ». Un nombre à virgule flottante est caractérisé par trois valeurs :

— Le **signe**,
— La **mantisse**,
— L'**exposant**.

Un système de représentation de nombres à virgule flottante comporte également une **base** b, de sorte à ce qu'un nombre x de signe s, de mantisse m et d'exposant e s'écrive :

$$x = s.m.b^e$$

Arrondis et approximation

De cette façon, on peut représenter de manière approchée de larges ensembles de réels. La norme IEEE 754 (voir IEEE (2008)) standardise la représentation des

nombres réels par des nombres à virgule flottante sur 32 et 64 bits. La base choisie est la base 2 et les bits disponibles sont répartis tel que suit :

— Pour les nombres à virgule flottante sur 32 bits : 1 bit de signe, 23 bits de mantisse et 8 bits d'exposant.
— Pour les nombres à virgule flottante sur 64 bits : 1 bit de signe, 52 bits de mantisse et 11 bits d'exposant.

Ainsi, les seuls nombres dont la représentation est exacte selon cette norme sont des sommes de puissances de 2 et d'inverses de puissances de 2 ou 0. Tout autre nombre est arrondi au nombre le plus proche, à zéro, à $+\infty$ ou à $-\infty$. On constate donc en général une erreur d'arrondi lors de la construction de la représentation d'un réel.

De plus, comme le résultat de toute opération est aussi arrondi, l'associativité n'est pas strictement garantie en calcul flottant. L'exemple suivant illustre les problèmes potentiels que cela peut occasionner :

```
int main(int argc, char ** argv) {
   float a = (123.f + 5.46e-6f) - 123.f;
   float b = (123.f - 123.f) + 5.46e-6f;

   std::cout << a << std::endl;
   std::cout << b << std::endl;

   return 0;
}
```

La sortie obtenue est la suivante :

```
7.62939e-06
5.46e-06
```

Dans le premier cas, pour le calcul de `a`, le calcul de la somme `123.f + 5.46e-6f` occasionne un arrondi dont l'erreur, bien que faible relativement à la somme, est importante relativement au résultat final de l'opération. Pour le calcul de `b`, `123.f - 123.f` vaut exactement 0 et la somme totale est correcte. L'erreur relative remet donc en cause l'associativité des calculs arithmétiques.

De manière générale, lors de la manipulation de nombres à virgule flottante, il est essentiel de tenir compte des limites de la précision et du domaine de représentation afin d'éviter des erreurs trop importantes liées à des débordements ou des absorptions. En pratique, les propriétés des opérateurs arithmétiques telles que la distributivité, l'associativité et la commutativité ne sont plus strictement respectées.

Au cours des travaux de cette thèse, nous avons utilisé les nombres flottants à simple précision sur des calculs géométriques et physiques. Nous n'avons pas rencontré de problèmes de stabilité numérique ou de précision justifiant l'utilisation de flottants à double précision. L'ensemble des résultats obtenus a fait l'objet de validations par rapport à *CIVA* (calculs en double précision) et l'erreur observée est restée en dessous du critère métier de précision.

ANNEXE C

Calcul des dérivées

Dans cette annexe, nous décrivons la méthode que nous avons utilisée pour calculer les grandeurs dérivées nécessaires à la construction des interpolateurs polynomiaux à deux variables décrits dans le chapitre 4. Puisque nous ne disposons pas de méthode analytique de calcul de dérivée sur les grandeurs des rayons lancés, ce calcul utilise la méthode des différences finies.

Puisque les dérivées calculées sont potentiellement utilisées pour plusieurs pinceaux et pour éviter des calculs redondants, nous mettons en place un système de cache permettant de calculer les différentes dérivées dans des repères différents.

Calculs de dérivées par différences finies

Étant donnée une fonction réelle $f : \mathbf{R} \to \mathbf{R}$, on définit la fonction dérivée f' par :

$$\forall x \in \mathbf{R}, f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Ainsi, pour calculer numériquement une approximation de la dérivée d'une fonction, on peut fixer une valeur de h proche de zéro et évaluer le taux d'accroissement en se basant sur ce pas. Ainsi, pour un réel x et un pas h, on estime la dérivée tel que suit :

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Plus la valeur de h est faible, plus cette approximation est précise. Cependant, puisque ces calculs sont effectuées numériquement au moyen de nombres à virgule flottante, le choix du pas h est déterminant. En effet, comme expliqué en annexe B, si la valeur du pas h est trop faible, un arrondi peut entraîner des erreurs importantes sur l'estimation de la dérivée.

Pour une fonction f de n variables, on peut, de même, calculer les dérivées partielles par différences finies :

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_0, \ldots, x_i + h, \ldots, x_n) - f(x_0, \ldots, x_i, \ldots, x_n)}{h}$$

En étendant ce principe, on peut également calculer des dérivées d'ordres supérieurs. Ainsi, la dérivée seconde d'une fonction f peut être approximée comme suit :

$$f''(x) \approx \frac{f'(x+h) - f'(x)}{h} \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

De même, on peut approximer la dérivée croisée d'une fonction :

$$\frac{\partial^2 f}{\partial x \partial y} \approx \frac{f(x+h, y+h) + f(x,y) - f(x+h,y) - f(x,y+h)}{4h^2}$$

Changement de repère

Dérivées

Connaissant les dérivées partielles d'une fonction en un point dans un système de coordonnées (x, y) donné, on peut calculer les dérivées de cette fonction au même point dans un autre système de coordonnées. Notons (x', y') un repère tel que :

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \alpha_1 & \alpha_2 \\ \beta_1 & \beta_2 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

On a alors :

$$\begin{aligned} \frac{\partial f}{\partial x'} &= \frac{\partial f}{\partial x}\frac{\partial x}{\partial x'} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial x'} \\ &= \alpha_1 \frac{\partial f}{\partial x} + \alpha_2 \frac{\partial f}{\partial y} \end{aligned}$$

De même :

$$\begin{aligned} \frac{\partial f}{\partial x'} &= \frac{\partial f}{\partial x}\frac{\partial x}{\partial y'} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial y'} \\ &= \beta_1 \frac{\partial f}{\partial x} + \beta_2 \frac{\partial f}{\partial y} \end{aligned}$$

Ainsi, la connaissance des dérivées partielles d'une fonction dans un système de coordonnées suffit à déterminer la valeur de ces mêmes dérivées partielles dans tout système de coordonnées issue d'une transformation linéaire du premier.

Dérivées secondes

Pour une fonction f de plusieurs variables, on appelle **matrice hessienne** la matrice suivante :

$$H(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Pour une fonction à deux variables, on a :

$$H(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y} \end{pmatrix}$$

La donnée cette matrice permet d'effectuer un changement de système de coordonnées. En reprenant le système de coordonnées (x', y') défini précédemment, on peut déduire les valeurs de dérivées secondes et dérivées croisées tel que suit :

$$\frac{\partial^2 f}{\partial x'^2} = \begin{pmatrix} \alpha_1 & \alpha_2 \end{pmatrix} \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}$$

$$\frac{\partial^2 f}{\partial y'^2} = \begin{pmatrix} \beta_1 & \beta_2 \end{pmatrix} \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$$

$$\frac{\partial^2 f}{\partial x' \partial y'} = \begin{pmatrix} \alpha_1 & \alpha_2 \end{pmatrix} \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$$

Par conséquent, pour disposer des valeurs de dérivées nécessaires à la construction des interpolateurs des grandeurs de rayons, il suffit, pour chaque rayon utilisé par ces interpolateurs, de construire la matrice hessienne de chaque quantité à interpoler. Elle peut alors être utilisée pour déduire les dérivées secondes et croisées de chaque quantité interpolée.

ANNEXE D

Coefficients de Fresnel

Dans cette annexe, nous présentons brièvement le des coefficients de Fresnel aux interfaces entre deux milieux de propagation. À chacune de ces interfaces, l'énergie d'une onde incidente est répartie entre des ondes qui peuvent être :

— Réfléchies, donc générées dans le matériau d'incidence,
— Transmises, donc générées dans le matériau opposé,
— Évanescentes, s'atténuant à l'interface entre les deux matériaux.

Ainsi, pour chaque onde générée, on peut calculer un coefficient caractérisant la part d'énergie qui lui est associée. Ce coefficient, dit « coefficient de Fresnel », dépend des caractéristiques des deux matériaux et de l'angle entre le rayon de l'onde incidente et la normale à l'interface.

Des méthodes permettant de les calculer ont été proposées dans la littérature décrivant la physique des ondes ultrasonores (voir par exemple Royer et Dieulesaint (1996), Nayfeh (1991) et Ribeiro Monteiro (1992)), nous ne détaillons pas l'ensemble des calculs.

Chaque couple de modes et de milieux induisant un calcul différent, le nombre de cas à gérer est important si l'on considère l'ensemble des matériaux isotropes et anisotropes ainsi que les modes qu'ils permettent de propager. Notons par ailleurs que la détermination de ces coefficients requiert le calcul de l'ensemble des modes générés à l'interface.

Conditions de continuité

Pour les milieux isotropes, les conditions de continuité des contraintes et déplacements aux interfaces entre deux matériaux, en supposant un contact parfait, peuvent être traduites par des équations admettant une solution analytique.

Dans le cas général de milieux solides isotropes ou anisotropes, les conditions de continuité sont les suivantes :

— Continuité de la contrainte normale à l'interface,
— Continuité de la contrainte tangentielle à l'interface,
— Continuité du déplacement normal à l'interface,
— Continuité du déplacement tangentiel à l'interface.

Si au moins l'un des milieux est fluide, la contrainte tangentielle est nulle en raison de l'absence de cohésion de la matière dans le milieu fluide. On a alors :

— Continuité de la contrainte normale à l'interface,
— Nullité de la contrainte tangentielle à l'interface,
— Continuité du déplacement normal à l'interface.

Cas des milieux isotropes

Dans le cas d'une interface entre un milieu isotrope liquide et un milieu isotrope fluide, on considère le plus souvent les situations suivantes :

— Transmission d'une onde L depuis un milieu fluide isotrope vers une onde L ou T dans un milieu solide isotrope.
— Réflexion d'une onde L dans un milieu fluide isotrope vers une onde L ou T dans le même milieu.

Pour exemple, nous proposons l'implémentation du calcul du coefficient de Fresnel dans les cas de transmission entre un milieu fluide et un milieu solide isotropes :

```
complex<float> transmission_IsotropeL_FluideL(MilieuLiquide milieuLiq,
  MilieuIsotrope milieuIso, RayonUS ondeIncidente, Vecteur3D normale) {
    const float composanteNormalInc =
      produitScalaire(ondeIncidente.directionPhase, normale) *
      milieuLiq.lenteurL;

   float SF = milieuLiq.L_slowness;
   float SL = milieuIso.L_slowness;
   const float SzF = composanteNormalInc;
   const float SzF_2 = SzF * SzF;

   const float SL_2 = SL * SL;
   const float SF_2 = SF * SF;
   const float Sx_2 = SF_2 - SzF_2;

   if(Sx_2 < SL_2) // En dessous de l'angle critique L
   {
      float ST_2 = milieuIso.T_slowness * milieuIso.T_slowness;
      float r_rho = milieuLiq.densité / milieuIso.densité;
      float SzL = sqrtf(SL_2 - Sx_2);
      float SzT = sqrtf(ST_2 - Sx_2);
      float val = ST_2 - float(2.f)*Sx_2;
      float denom = val * val + (4 * Sx_2 * SzT + r_rho * ST_2 * ST_2 / SzF) *
        SzL;
      const float num = 2 * r_rho * val * ST_2;
      coeff.real(SL/SF*num / denom);
      coeff.imag(0.f);
   }
}
```

Cas des milieux anisotropes

À une interface faisant intervenir au moins un matériau anisotrope, une fois les caractéristiques des ondes générées à l'interface connues, le calcul des coefficients de Fresnel pour chacun des modes passe par le calcul et l'inversion d'une matrice de conditions, dépendant de la matrice des contraintes à l'interface.

La dimension de cette matrice carrée dépend du nombre d'ondes différentes à l'interface. Ainsi, dans le cas général de deux matériaux anisotropes, puisque 6 modes coexistent de part et d'autre de l'interface, la matrice est de taille 6×6. Dans les autres cas, on a :

— À une interface entre un matériau fluide et un matériau anisotrope, une matrice de taille 4×4.
— À une interface entre un matériau solide isotrope et un matériau anisotrope, une matrice de taille 5×5.

ANNEXE E

Méthodes de résolution polynomiale

Nous présentons dans cette annexe la méthode de résolution polynomiale que nous avons utilisée dans le cadre des travaux de cette thèse. Elle vise à rechercher numériquement les racines réelles de fonctions pour lesquelles les méthodes analytiques ne sont pas applicables.

Méthode de Newton

En une dimension

La méthode de Newton permet de trouver une approximation d'une racine réelle d'une fonction f lorsque l'on dispose d'un moyen d'évaluer sa dérivée. Il s'agit d'une méthode itérative qui, partant d'une valeur initiale x_0, estime un pas qui permet d'approcher une racine. Des valeurs x_i sont calculées, convergeant la plupart du temps vers une racine. Plus la valeur initiale est proche d'une racine, plus la méthode converge rapidement.

L'itération appliquée utilise une approximation linéaire de la fonction au moyen de sa dérivée en résolvant l'équation affine :

$$f(x_i) + f'(x_i).\delta_i = 0$$

À chaque étape, on obtient donc x_{i+1} de la sorte :

$$x_{i+1} := x_i - \frac{f(x_i)}{f'(x_i)}$$

Pour toute racine α de la fonction f, si f est continue, on peut montrer qu'il existe un voisinage de α tel que la méthode de Newton converge vers α. Sa convergence est quadratique à condition que la point de départ soit dans un domaine de convergence. Dans le cas où la valeur de départ est trop éloignée de toute racine de la fonction f, il est possible que la méthode de Newton diverge. Par conséquent, un nombre limite d'itérations doit être imposé.

En deux dimensions

Cette méthode peut être étendue au cas de fonctions de deux variables à images bidimensionnelles en utilisant le même type d'approximation linéaire. Cette fois-ci, on calcule le jacobien de la fonction $f : \mathbf{R}^2 \rightarrow \mathbf{R}^2$ dont on cherche les racines pour en déduire un pas. En notant (f_x, f_y) les deux composantes de f, on a :

$$J(x,y) = \begin{pmatrix} \dfrac{\partial f_x}{\partial x} & \dfrac{\partial f_x}{\partial y} \\ \dfrac{\partial f_y}{\partial x} & \dfrac{\partial f_y}{\partial y} \end{pmatrix}$$

Le pas $(\delta x_i, \delta y_i)$ est choisi de façon à vérifier le système d'équations suivant :

$$\begin{pmatrix} f_x(x_i, y_i) \\ f_y(x_i, y_i) \end{pmatrix} + J^{-1}(x_i, y_i) \begin{pmatrix} \delta x_i \\ \delta y_i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

De la même façon, la répétition de cette itération permet de converger vers une racine de la fonction f sous les mêmes conditions qu'en une dimension. En pratique, nous avons utilisé cette méthode pour l'inversion d'interpolation d'Hermite sur les domaines de couverture de pinceaux (voir le chapitre 5).

En fixant comme valeur initiale le milieu du domaine, à savoir le point $(1/2, 1/2)$, la valeur ciblée par la méthode de Newton se trouve à une distance maximale de $1/\sqrt{2}$ du la valeur initiale. Une limite de 6 itérations suffit généralement à une bonne convergence vers la solution.

Méthode de Durand-Kerner

La méthode de Durand-Kerner permet la recherche simultanée de toutes les racines d'un polynôme de degré quelconque. Son principe de base consiste à initialiser n valeurs initiales $\{z_0^{(1)}, z_0^{(2)}, \ldots, z_0^{(n)}\}$ différentes et à effectuer l'itération suivante jusqu'à convergence (ou dépassement d'un nombre limite d'itérations) :

$$z_i^{(k+1)} = z_i^k - \frac{f(z_i^{(k)})}{P_i^{(k)}}$$

avec

$$P_i^{(k)} = \prod_{\substack{j=0 \\ j \neq i}}^{n} (z_i^{(k)} - z_j^{(k)})$$

Usuellement, les valeurs sont choisies complexes sur le cercle trigonométrique, de manière à y être espacées régulièrement :

$$z_i^{(0)} = e^{j\frac{2i\Pi}{n}}$$

Cette méthode est numériquement stable à condition que les valeurs initiales soient toutes différentes les unes des autres. Sa convergence est quadratique à partir du moment où les estimations de racines sont proches des racines du polynôme et linéaire sinon ;

En pratique, il est intéressant d'utiliser la méthode de Durand-Kerner pour trouver toutes les racines (y compris les racines complexes) d'un polynôme. Nous pouvons donc l'utiliser pour la résolution algébrique de l'équation de Snell-Descartes en milieux anisotropes, telle que présentée dans le chapitre 2.

Résumé

Pour répondre à des impératifs croissants de fiabilité et de sûreté, les procédés mis en œuvre dans le cadre du contrôle non destructif sont en constante évolution. Au vu de la complexité des techniques utilisées, la simulation prend une part importante dans leur développement. Nous présentons des travaux ayant abouti à un outil rapide de simulation du champ émis par un traducteur ultrasonore plan quelconque dans des configurations complexes de contrôle non destructif impliquant des géométries maillées sans arêtes saillantes, des matériaux isotropes ou anisotropes, homogènes ou hétérogènes et des trajectoires d'ondes pouvant comporter des rebonds et des transmissions. Les fronts d'onde ultrasonore sont approximés à l'aide d'interpolateurs polynomiaux locaux à des pinceaux de rayons ultrasonores. Ceux-ci sont obtenus au moyen d'un algorithme de recherche de surface par lancer de pinceaux et subdivisions successives. Ils permettent le calcul des grandeurs utiles à la constitution de la réponse impulsionnelle en chaque point d'un échantillonnage du traducteur respectant le critère de Shannon. De cette façon, nous pouvons calculer une réponse impulsionnelle qui, convoluée au signal d'excitation du traducteur, donne le champ ultrasonore. Les performances des simulations ont été accrues par l'exploitation du parallélisme de tâches et des instructions SIMD dans les parties les plus coûteuses du calcul. Enfin, un outil de calcul progressif continu a été développé pour permettre une visualisation interactive d'images de champ. Il exploite une méthode de reconstruction d'images et ordonnance les calculs de champ de manière à accélérer la convergence des images produites.

Mots-clés : simulation, contrôle non destructif, ultrasons, anisotropie, lancer de rayons, reconstruction de fonctions, recherche de visibilité, parallélisme, vectorisation, interactivité, progressivité.

Abstract

In order to fulfill increasing reliability and safety requirements, non-destructive testing techniques are constantly evolving and so does their complexity. Consequently, simulation is an essential part of their design. We developed a tool for the simulation of the ultrasonic field radiated by any planar probes into non-destructive testing configurations involving meshed geometries without prominent edges, isotropic and anisotropic, homogeneous and heterogeneous materials, and wave trajectories that can include reflections and transmissions. We approximate the ultrasonic wavefronts by using polynomial interpolators that are local to ultrasonic ray pencils. They are obtained using a surface research algorithm based on pencil tracing and successive subdivisions. Their interpolators enable the computation of the necessary quantities for the impulse response computation on each point of a sampling of the transducer surface that fulfills the Shannon criterion. By doing so, we can compute a global impulse response which, when convoluted with the excitation signal of the transducer, results in the ultrasonic field. The usage of task parallelism and of SIMD instructions on the most computationally expensive steps yields an important performance boost. Finally, we developed a tool for progressive visualization of field images. It benefits from an image reconstruction technique and schedules field computations in order to accelerate convergence towards the final image.

Keywords : simulation, non destructive testing, ultrasounds, anisotropy, ray-tracing, function reconstruction, visibility, parallelism, vectorization, interactivity, progressivity.