HAL
open science

Implantation des opérateurs de l'algèbre géométrique par arbres binaires récursifs

Stéphane Breuils, Vincent Nozick, Laurent Fuchs

To cite this version:

Stéphane Breuils, Vincent Nozick, Laurent Fuchs. Implantation des opérateurs de l'algèbre géométrique par arbres binaires récursifs. Journées du Groupe de Travail en Modélisation Géométrique 2016,, 2016, Dijon, France. hal-01926008

HAL Id: hal-01926008

https://hal.science/hal-01926008v1

Submitted on 18 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Implantation des opérateurs de l'algèbre géométrique par arbres binaires récursifs

Stéphane Breuils[1] stephane.breuils@u-pem.fr , Vincent Nozick[1] vincent.nozick@u-pem.fr et Laurent Fuchs[2] Laurent.Fuchs@univ-poitiers.fr

[1]Université Paris-Est Marne la Vallée
[2]Université de Poitiers

Résumé
L'algèbre géométrique est un outil permettant de représenter et manipuler les objets géométriques de manière générique, efficace et intuitive. Différentes approches ont été prospectées afin d'obtenir des implantations efficaces, mais restent toutefois perfectibles notamment sur leur attractivité face à un usage industriel. Cet article présente une implantation de ces algèbres basée sur une représentation de ses éléments sous forme d'arbres binaires. Cette représentation aboutit à des définitions récursives des produits permettant, une fois déroulés, la mise en œuvre de nouveaux algorithmes itératifs. Ces algorithmes consistent à assimiler les produits à des constructions itératives de listes. L'approche obtenue est valable en toutes dimensions. Les algorithmes proposés ont été implantés en C++ et comparés avec les méthodes de l'état de l'art.

Mots-clés : Algèbre géométrique, Arbre binaire

1. Introduction

L'algèbre géométrique est un outil mathématique de représentation et de manipulation d'objets géométriques initialement conceptualisé par Hermann Grassmann en 1844. À la même époque, Wiliam Rowan Hamilton (1805-1865) développait l'algèbre des quaternions utilisée pour les rotations 3D. Enfin William Kingdon Clifford (1845-1879) produisit une algèbre permettant d'englober toutes les algèbres (Grassmann, quaternion, ...). Leurs travaux n'ont été repris seulement qu'à la fin du XXe siècle grâce à l'avènement de l'informatique. David Hestenes a remis au goût du jour cette algèbre en résolvant des problèmes de mécanique quantique avec l'algèbre géométrique. Pour un historique plus complet, se référer à [**?**] ou [**?**].

Tout comme l'algèbre linéaire usuellement utilisée pour représenter les objets géométriques, cette algèbre est définie à partir d'un espace vectoriel. Sa spécificité est de pouvoir représenter des objets géométriques par des sous-espaces vectoriels engendrés par des opérateurs de l'algèbre.

Voici quelques exemples d'opérations typiques que l'on peut rencontrer en algèbre géométrique. Dans l'espace usuel de dimension 3, considérons 4 points de vecteurs positions p_1, p_2, p_3, p_4 représentés par 4 vecteurs $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$ de l'algèbre, il est possible de calculer la sphère $\mathbf{s}$ passant par ces 4 points de la manière suivante :

$$\mathbf{s} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{p}_3 \wedge \mathbf{p}_4$$

Avec le même opérateur et à partir de trois points dont les vecteurs sont $\mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7$ et un point à l'infini de vecteur $\mathbf{n}_i$ (similaire à la coordonnée homogène en géométrie projective), on détermine le plan π passant par ces trois points par la formule suivante :

$$\pi = \mathbf{p}_5 \wedge \mathbf{p}_6 \wedge \mathbf{p}_7 \wedge \mathbf{n}_i$$

Enfin avec un deuxième opérateur étant une extension du produit scalaire de l'algèbre linéaire appelé *contraction* et noté $\rfloor$, on calcule le cercle $\mathbf{c}$ obtenu par l'intersection entre le plan π et la sphère $\mathbf{s}$:

$$\mathbf{c} = \pi \rfloor \mathbf{s}$$

Ce résultat géométrique, représentée sur la figure **??**, est tirée du logiciel GAViewer [**?**]. Cet exemple illustre la capacité de compacité des expressions de l'algèbre géométrique pour la définition de primitives géométriques, points, sphères, plans, cercles. Par ailleurs, les calculs effectifs produits par ces opé-

rateurs sont, à l'image du produit vectoriel de $\mathbb{R}^3$, extrêmement compacts.

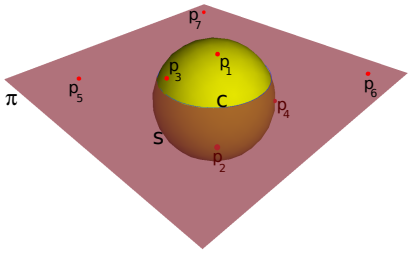

Figure 1: Intersection entre une sphère et un plan. Le cercle $\mathbf{c}$ est obtenu par intersection de la sphère $\mathbf{s} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{p}_3 \wedge \mathbf{p}_4$ et du plan $\pi = \mathbf{p}_5 \wedge \mathbf{p}_6 \wedge \mathbf{p}_7 \wedge \mathbf{n}_i$.

Un frein à l'utilisation de l'algèbre géométrique réside dans la montée en dimension qui survient quand on considère l'espace des sous-espaces vectoriels d'un espace vectoriel, ce qui est courant en algèbre géométrique. En effet, un espace vectoriel de dimension d nécessite un espace de sous-espaces vectoriels de dimension 2^d. Le stockage en mémoire de ces éléments devient coûteux. En pratique cependant, les objets géométriques ne sont représentés que par un seul sous-espace d'une dimension fixée. Ainsi, selon l'objet représenté, seule une partie du vecteur de l'algèbre sera utilisée, ce qui rend les vecteurs très creux. À titre d'exemple, une sphère, en dimension 3, sera représentée par un vecteur de l'espace des sous-espaces vectoriel d'un espace à 5 dimensions produisant 32 coordonnées dont seulement 5 seront non nuls.

Il existe plusieurs implantations des algèbres géométriques, notamment Gaigen [**?**] qui adapte la représentation et les calculs effectués en fonction de l'objet géométrique en utilisant des classes spécialisées. Gaalop [**?**] est une autre implantation utilisant un pré-compilateur pour générer les éléments de l'algèbre géométrique. Cette approche, bien qu'efficace en terme de temps de calcul, n'autorise pas de modifications autre que celle des valeurs représentant les objets déjà définis et n'est donc pas utilisable dans des logiciels où une interaction avec un utilisateur est prévue.

Enfin, Fuchs et Théry [**?**] définissent les éléments de l'algèbre géométrique à partir d'arbres binaires et présentent les opérateurs de l'algèbre géométrique de manière récursive. Dans cet article, nous proposons une construction des éléments de l'algèbre géométrique selon cette dernière approche. Les opérateurs de l'algèbre sont définis itérativement à partir de leur expression récursive. La suite de cet article est organisée de la façon suivante : la section 2 présente l'approche récursive de représentation des éléments de l'algèbre. Les sections 3 et 4 décrivent les méthodes itératives de calculs des opérateurs de l'algèbre géométrique, une étude de la complexité en temps et en mémoire y est explicitée. Enfin la section 5 donne les résultats expérimentaux.

2. Définitions et notations

L'algèbre géométrique est définie à partir d'un espace vectoriel de dimension d. La base canonique de cet espace vectoriel est notée :

$$(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \cdots, \mathbf{e}_d)$$

2.1. Opérateur de construction de sous-espaces

La construction des sous-espaces est effectuée grâce au produit extérieur noté $\wedge$. Le produit extérieur entre deux vecteurs $\mathbf{a}$ et $\mathbf{b}$ a les propriétés suivantes :

$$\begin{array}{lcl} \text{Antisymétrie} & : & \mathbf{a}\wedge\mathbf{b} = -\mathbf{b}\wedge\mathbf{a} \quad \Rightarrow \quad \mathbf{a}\wedge\mathbf{a} = 0 \\ \text{bilinéarité} & : & \left\{ \begin{array}{l} \mathbf{a}\wedge(\beta\mathbf{b}) = \beta(\mathbf{a}\wedge\mathbf{b}) \\ \mathbf{a}\wedge(\mathbf{b}+\mathbf{c}) = (\mathbf{a}\wedge\mathbf{b}) + (\mathbf{a}\wedge\mathbf{c}) \end{array} \right. \end{array}$$

où β est un scalaire.

Les sous-espaces ne sont pas exprimés dans l'espace vectoriel de départ, mais dans l'espace des sous-espaces vectoriels de l'espace de départ. Cet espace des sous-espaces vectoriels possède une base canonique construite à partir de la base canonique de l'espace vectoriel de départ. À titre d'exemple en dimension 3, la base de l'algèbre géométrique est composée des vecteurs suivants :

$$(\mathbf{1}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_{12}, \mathbf{e}_{13}, \mathbf{e}_{23}, \mathbf{e}_{123})$$

avec $\mathbf{e}_{ij}$ le résultat de $\mathbf{e}_i \wedge \mathbf{e}_j$.

Maintenant, nous introduisons quelques définitions. Les éléments de l'algèbre géométrique sont les combinaisons linéaires de la base canonique. Les éléments décomposables en produits extérieurs de vecteurs sont appelés des *blades*. On appelle *grade* d'un *blade* le nombre de vecteurs de base qui engendre le sous espace dans lequel se situe l'objet. En dimension 3, les grades des éléments de la base de l'algèbre géométrique sont les suivants :

$$(\underbrace{\mathbf{1}}_{grade\ 0}, \underbrace{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3}_{grade\ 1}, \underbrace{\mathbf{e}_{12}, \mathbf{e}_{13}, \mathbf{e}_{23}}_{grade\ 2}, \underbrace{\mathbf{e}_{123}}_{grade\ 3})$$

2.2. Multivecteurs

Les éléments de l'algèbre géométrique sont appelés des multivecteurs. Tout comme un vecteur nous notons les multivecteurs en gras et en lettre minuscule (le multivecteur $\mathbf{a}$). On dit qu'un multivecteur est de grade k si l'élément de plus haut grade constituant ce multivecteur est de grade k.

En dimension d, un multivecteur $\mathbf{a}$ peut être exprimé

comme une somme de blades pondérés par des coefficients a_i de la manière suivante :

$$\mathbf{a} = \sum_{i=0}^{2^d} a_i \mathbf{E}_i \tag{1}$$

Avec $\mathbf{E}_i$ un blade dont le grade va de 1 à d. On peut par exemple avoir $\mathbf{E}_3 = \mathbf{e}_2$ et $\mathbf{E}_8 = \mathbf{e}_{123}$

2.3. Arbres binaires

L'approche [**?**] propose une représentation des multivecteurs par des arbres binaires où chaque nœud est un multivecteur. L'arbre binaire $\mathfrak{a}$ peut s'exprimer de manière récursive par :

$$\mathfrak{a}^n = (\mathfrak{a}_1^{n+1}, \mathfrak{a}_0^{n+1})^n \tag{2}$$

$\mathfrak{a}_1$ et $\mathfrak{a}_0$ représentent les fils gauche et droit du nœud $\mathfrak{a}$. n est le niveau de récursion pouvant aller de 0 (racine de l'arbre) à d (feuille de l'arbre).

2.4. Arbres binaires et multivecteurs

À chaque profondeur de l'arbre, c'est à dire à chaque dimension de l'espace, correspond un vecteur de base de grade 1 qui contribue à la construction des multivecteurs de grades supérieurs, voir figure **??**. La notation $\mathfrak{a}^n = (\mathfrak{a}_1^{n+1}, \mathfrak{a}_0^{n+1})^n$ indique qu'à une profondeur n, le nœud $\mathfrak{a}^n$ sépare la partie $\mathfrak{a}_1$ contenant du $\mathbf{e}_n$ de la partie $\mathfrak{a}_0$ qui n'en contient pas. Les feuilles de l'arbre binaire $\mathfrak{a}$ représentent les vecteurs de base pouvant exprimer le multivecteur $\mathbf{a}$, en leur associant les coefficients a_i de $\mathbf{a}$ correspondant. Un nœud $\mathfrak{a}^n$ est lui même un multivecteur composé de vecteurs de base dont le grade maximal vaut n.

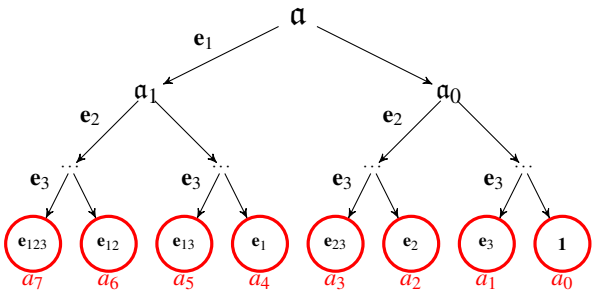

Figure 2: Arbre binaire de $\mathfrak{a}$ associé au multivecteur $\mathbf{a}$ en dimension 3. A chaque feuille de l'arbre correspond une composante du multivecteur $\mathbf{a}$.

3. Produit extérieur

Avant de présenter la méthode proposée dans cet article, nous décrivons la méthode générale de calcul du produit extérieur. Soit $\mathbf{c} = \mathbf{a} \wedge \mathbf{b}$ le résultat du produit extérieur de $\mathbf{a}$ par $\mathbf{b}$, on a alors :

$$\mathbf{c} = \sum_{k=0}^{2^d} c_k \mathbf{E}_k = \sum_{i=0}^{2^d} a_i \mathbf{E}_i \wedge \sum_{j=0}^{2^d} b_j \mathbf{E}_j \tag{3}$$

Comme le produit extérieur est distributif par rapport à l'addition, le produit extérieur de $\mathbf{a}$ par $\mathbf{b}$ devient :

$$\mathbf{c} = \sum_{i=0}^{2^d} \sum_{j=0}^{2^d} a_i b_j \left(s_{ij}\, \mathbf{E}_i \wedge \mathbf{E}_j \right) \tag{4}$$

avec s_{ij} le signe du produit extérieur entre deux vecteurs de base qui dépend de l'ordre dans lequel ce produit est effectué. Une composante c_k de $\mathbf{c}$ s'écrit alors :

$$c_k = \sum_{\substack{i,j \text{ tel que} \\ \mathbf{E}_i \wedge \mathbf{E}_j = s_{ij} \mathbf{E}_k}} s_{ij}.a_i \wedge b_j = \sum_{\substack{i,j \text{ tel que} \\ \mathbf{E}_i \wedge \mathbf{E}_j = s_{ij} \mathbf{e}_k}} s_{ij}.a_i b_j \tag{5}$$

Comme a_i et b_j sont des coefficients, alors le produit extérieur $a_i \wedge b_j$ revient à effectuer le produit des deux scalaires a_i et b_j.

3.1. Produit extérieur : version naïve

Une première approche naïve du calcul du produit extérieur entre deux multivecteurs $\mathbf{a}$ et $\mathbf{b}$ consiste à calculer le produit extérieur entre chaque vecteur de base de $\mathbf{a}$ et de $\mathbf{b}$ en utilisant la distributivité du produit extérieur.

Comme le produit extérieur entre vecteurs de base est calculé en temps constant, la complexité de cette approche sera alors déterminée par le nombre de parcours des multivecteurs. Sachant que le nombre de vecteurs de base d'un multivecteur de dimension d est 2^d. Le nombre de parcours effectués est donc de $2^{2d} = 4^d$. La complexité de l'algorithme est donc en $O(4^d)$. Cette approche ne permet pas d'éviter les produits extérieurs entre 2 vecteurs de base dépendants (ie., ayant un ou plusieurs vecteurs de base de grade 1 en commun). Le résultat de ce type de produit est nul, ce qui induit des opérations inutiles. Dans l'exemple suivant, le produit extérieur entre deux multivecteurs $\mathbf{a}$ et $\mathbf{b}$ en dimension 3 contient des calculs inutiles :

$$\begin{aligned} \mathbf{a} &= 2\,\mathbf{e}_1 \quad \mathbf{b} = 4\,\mathbf{e}_1 + 3\,\mathbf{e}_3 \\ \mathbf{a} \wedge \mathbf{b} &= 2 \times 4\,\mathbf{e}_1 \wedge \mathbf{e}_1 + 2 \times 3\,\mathbf{e}_1 \wedge \mathbf{e}_3 \\ &= 0 + 6\,\mathbf{e}_{13} \\ &= 6\,\mathbf{e}_{13} \end{aligned}$$

On remarque qu'une seule opération a abouti sur un résultat non nul.

3.2. Produit extérieur par récursion

Les opérateurs de l'algèbre géométrique sont également définis de manière récursive [**?**]. La formule ci-dessous définit le produit extérieur entre deux arbres $\mathfrak{a}$ et $\mathfrak{b}$ de même profondeur n :

$$
\begin{aligned}
&\mathfrak{a}^n \wedge \mathfrak{b}^n = (\mathfrak{a}_0^{n+1}, \mathfrak{a}_1^{n+1})^n \wedge (\mathfrak{b}_0^{n+1}, \mathfrak{b}_1^{n+1})^n \\
&\text{Si } n < d \ , \ \mathfrak{a}^n \wedge \mathfrak{b}^n = \\
&\qquad \left(\mathfrak{a}_1^{n+1} \wedge \mathfrak{b}_0^{n+1} + \overline{\mathfrak{a}_0}^{n+1} \wedge \mathfrak{b}_1^{n+1} \ , \mathfrak{a}_0^{n+1} \wedge \mathfrak{b}_0^{n+1}\right)^n \\
&\text{Si } n = d \ , \ \mathfrak{a}^n \wedge \mathfrak{b}^n = \mathfrak{a}^d \wedge \mathfrak{b}^d
\end{aligned}
\tag{6}
$$

où $\overline{\bullet}$ désigne un opérateur prenant en compte la propriété d'antisymétrie du produit extérieur. L'approche récursive de calcul du produit extérieur consiste à développer la formule ci dessus jusqu'au feuille de l'arbre.

La complexité de cette approche est déterminée par le nombre de parcours (appels récursifs) des nœuds de l'arbre $\mathfrak{a}$. Chacun des trois termes $\mathfrak{a}_i^{n+1} \wedge \mathfrak{b}_j^{n+1}$ de la formule (**??**) induit un appel récursif. On en déduit qu'à chaque niveau de récursion le nombre d'appels récursifs est multiplié par trois. À un niveau de récursion n, le nombre total d'appels récursifs de niveau de récursion n est de 3^n. Par construction en dimension d, le nombre total d'appels récursifs est de :

$$
\sum_{i=1}^{d} 3^i = -\frac{3}{2}\left(1 - 3^d\right) \tag{7}
$$

Cette complexité est inférieure à la complexité de l'approche naïve vue précédemment $\left(O(4^n)\right)$. Cette approche n'est toutefois pas parallélisable. nous cherchons à obtenir un algorithme itératif à partir de cet algorithme récursif.

3.3. Approche proposée

Le but de notre approche est de minimiser le temps de calcul du produit extérieur et donc de diminuer le nombre de calculs. Pour ce faire, elle s'appuie sur la méthode récursive présentée au paragraphe précédent pour obtenir un algorithme itératif.

3.3.1. Développement du produit extérieur

La formulation récursive (**??**) génère, pour chaque feuille de $\mathfrak{c} = \mathfrak{a} \wedge \mathfrak{b}$, un ensemble de vecteurs de base $\mathfrak{a}^d \wedge \mathfrak{b}^d$. Chacune de ces feuilles est parcourue plusieurs fois par le calcul récursif et par conséquent reçoit plusieurs contributions de **a** et de **b**. En effet, chacun de ces parcours génère un couple $\mathfrak{a}^d \wedge \mathfrak{b}^d$ auxquels sont associés une paire de composantes a_i de **a** et b_j de **b**. Chacune de ces paires s'additionne aux autres contributions de la feuille en question. Par exemple, le développement de la formule récursive en dimension 3 donne le résultat présenté dans la table **??**.

$\mathbf{e}_{123}$	$\mathbf{e}_{12}$	$\mathbf{e}_{13}$	$\mathbf{e}_1$	$\mathbf{e}_{23}$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{1}$
c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
a_7b_0	a_6b_0	a_5b_0	a_4b_0	a_3b_0	a_2b_0	a_1b_0	a_0b_0
a_6b_1	a_4b_2	a_4b_1	a_0b_4	a_2b_1	a_0b_2	a_0b_1	
$-a_5b_2$	$-a_2b_4$	$-a_1b_4$		$-a_1b_2$			
a_4b_3	a_0b_6	a_0b_5		a_0b_3			
a_3b_4							
$-a_2b_5$							
a_1b_6							
a_0b_7							

Table 1: Développement de $\mathbf{c} = \mathbf{a} \wedge \mathbf{b}$ en dimension 3. Chaque parcours de l'arbre aboutissant à une feuille génère une contribution de la forme $\pm a_i b_j$ sur une des composantes du multivecteur résultat.

Le mode opératoire de notre approche est de déterminer d'une part combien de produits $a_i b_j$ sont nécessaires pour le calcul de la feuille c_k de $\mathfrak{c}$, et finalement de pouvoir identifier ces produits. Se posera ensuite la question du signe de chacun de ces produits.

3.3.2. Indiçage des arbres binaires

Nous proposons dans cette approche une nouvelle formulation récursive du produit extérieur permettant d'extraire des formules explicites du résultat du produit extérieur $\mathfrak{c} = \mathfrak{a} \wedge \mathfrak{b}$. Cette formulation est obtenue en indiçant les nœuds et les feuilles de l'arbre binaire d'un multivecteur. L'indiçage utilisé est le code de Huffman des nœuds de l'arbre. Un exemple de cet indiçage est montré sur la figure **??**.

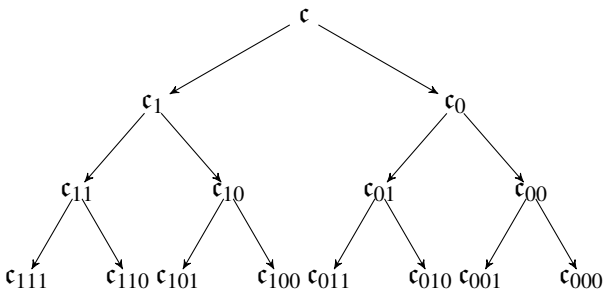

Figure 3: Indiçage des nœuds d'un arbre binaire par le code de Huffman.

La numérotation des feuilles de l'arbre binaire permet d'identifier le chemin emprunté dans l'arbre de la racine aux feuilles. Par exemple, on déduit du mot binaire 101 (Figure **??**) le fait que l'on emprunte tout d'abord le sous arbre gauche puis le sous arbre droite et en enfin le sous arbre gauche. À chaque blade correspond une position dans le mot binaire associé à un nœud. Ainsi, les 1 composant le mot binaire d'un nœud identifient les blades associés à ce nœud. Par exemple, en dimension 3, le blade $\mathbf{e}_{13} = \mathbf{e}_1 \wedge \mathbf{e}_3$ contient $\mathbf{e}_1$ et $\mathbf{e}_3$. Sa représentation 101 est la somme binaire

des représentations de $\mathbf{e}_1 = 100$ et $\mathbf{e}_3 = 001$. Cette somme binaire consiste à effectuer le "ou exclusif" $\mathbf{e}_1 \oplus \mathbf{e}_3$ entre les représentations de $\mathbf{e}_1 = 100$ et $\mathbf{e}_3 = 001$.

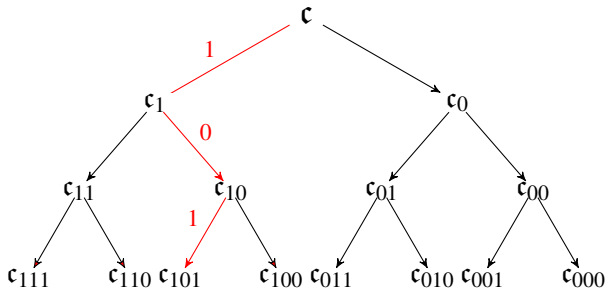

Figure 4: Chemin emprunté dans l'arbre jusqu'à $\mathfrak{c}_{101}$.

3.3.3. Produit extérieur assimilé à des constructions de listes

Avec l'indiçage vu précédemment, un nœud $\mathfrak{a}^n$ devient $\mathfrak{a}^n_u$, son fils gauche est $\mathfrak{a}^{n+1}_{u0}$ et son fils droit $\mathfrak{a}^{n+1}_{u1}$. En utilisant cet indiçage, la formule récursive du produit extérieur devient :

$$\begin{aligned}
&\mathfrak{a}^n_u \wedge \mathfrak{b}^n_v = (\mathfrak{a}^{n+1}_{u0}, \mathfrak{a}^{n+1}_{u1})^n \wedge (\mathfrak{b}^{n+1}_{v0}, \mathfrak{b}^{n+1}_{v1})^n \\
&\text{Si } n < d \ , \ \mathfrak{a}^n_u \wedge \mathfrak{b}^n_v = \\
&\qquad \left(\mathfrak{a}^{n+1}_{u1} \wedge \mathfrak{b}^{n+1}_{v0} + \overline{\mathfrak{a}_{u0}}^{n+1} \wedge \mathfrak{b}^{n+1}_{v1} \ , \mathfrak{a}^{n+1}_{u0} \wedge \mathfrak{b}^{n+1}_{v0}\right)^n \\
&\text{Si } n = d \ , \ \mathfrak{a}^n_u \wedge \mathfrak{b}^n_v = \ \mathfrak{a}^d_u \wedge \mathfrak{b}^d_v
\end{aligned} \tag{8}$$

Cet opérateur permet de déterminer le signe associé à chaque produit $a_i.b_j$. Dans la suite de l'article, nous calculerons le signe indépendamment de cet opérateur.

D'après la définition du produit extérieur, les feuilles $\mathfrak{c}^d_k$ peuvent s'exprimer comme une somme de composantes de $\mathfrak{a}$ et de $\mathfrak{b}$, formule (**??**). D'après la structure de l'algèbre géométrique, ce résultat est valide pour n'importe quel nœud de $\mathfrak{c}$. En intégrant ce résultat dans la formule récursive (**??**), on déduit une construction récursives des indices de $\mathfrak{a}$ intervenant dans le calcul de $\mathfrak{a} \wedge \mathfrak{b}$.

On appelle *Alist* et *Blist* ces constructions récursives d'ensembles de mots binaires de $\mathfrak{a}$ et $\mathfrak{b}$. Dans la suite de cet article, on ne s'intéressera qu'à la construction des indices de $\mathfrak{a}$. En effet on peut montrer par récurrence que les indices de $\mathfrak{b}$ intervenant dans le calcul d'une feuille $\mathfrak{c}_k$ sont obtenus par l'utilisation du "ou exclusif" entre les indices de $\mathfrak{a}$ et le mot binaire k. La définition récursive de *Alist* est la suivante :

$$\begin{aligned}
&\text{Si } n < d \ , \ \{Alist^n\} = \\
&\qquad \left(\{Alist(1)^{n+1}, Alist(0)^{n+1}\}, \{Alist(0)^{n+1}\}\right) \\
&\text{Si } n = d \ , \ \{Alist^n\} = \ \{Alist\}
\end{aligned} \tag{9}$$

Avec $\{. \ , \ .\}$ la concaténation de deux listes d'indices. $Alist(c)$ correspond à l'ajout du mot binaire c à la liste des mots binaires de *Alist*.

3.3.4. Méthode itérative de construction de *Alist*

Nous expliquons maintenant l'analyse de la définition récursive de *Alist* pour extraire une méthode itérative de construction de *Alist*. La figure **??** montre le développement de la formule récursive de *Alist* en dimension 3.

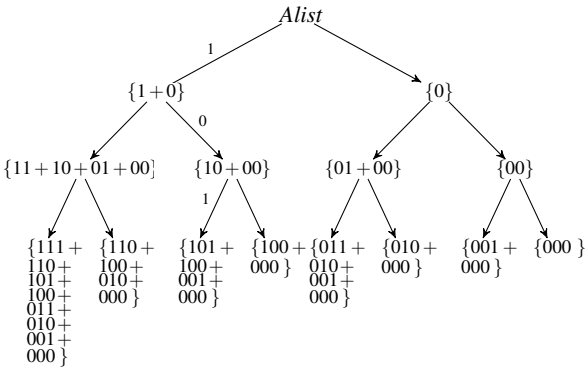

Figure 5: Développement de *Alist* en dimension 3 à partir de la formule (**??**).

On remarque tout d'abord qu'à chaque niveau de récursion n, et à chaque sous arbre gauche emprunté, le nombre d'éléments de *Alist* est multiplié par deux. Alors qu'à chaque sous arbre droit emprunté le nombre d'éléments de *Alist* reste le même. De plus, le mot binaire associé à une feuille de l'arbre (figure **??**) est suffisant pour déterminer le chemin dans l'arbre entre la feuille et la racine. Donc le nombre de produits à effectuer peut être déterminé à partir du mot binaire de l'indice. Ce nombre de produits p dépend du poids de Hamming du mot binaire u (ou nombre de 1 de la représentation binaire de l'indice du nœud, voir section **??**). On a :

$$p = 2^{h(u)}$$

avec h le poids de Hamming du mot binaire u.

L'étape suivante consiste à déterminer l'assemblage des indices $\mathfrak{a}$ pour chaque coefficient $\mathfrak{c}$. La formule récursive de la construction des indices de $\mathfrak{a}$ montre qu'à chaque sous arbre gauche emprunté $\left(\{Alist(1)^n, Alist(0)^n\}\right)$, une concaténation des mots binaires de $\{Alist\}$ avec 1 $\left(\{Alist(1)^n\}\right)$ est effectuée. On ajoute ensuite à cette liste une concaténation des mots binaires de $\{Alist\}$ avec 0 $\left(\{Alist(0)^n\}\right)$. Pour chaque sous arbre droit emprunté, la seule opération effectuée est la concaténation des mots binaires de $Alist(u)$ avec 0 $\left(\{Alist(0)^n\}\right)$.

Pour comprendre le déroulement de l'algorithme, prenons l'exemple du calcul de l'assemblage des indices de $\mathfrak{a}$ et $\mathfrak{b}$ pour le coefficient $\mathfrak{c}_{101}$. Ce calcul d'assemblage d'indices revient à déterminer les produits $a_i b_j$ associés à la colonne

c_5 $\left(5_{(10)} = 101_{(2)}\right)$ de la table **??**. Le calcul de la liste des coefficients $\mathfrak{a}$ intervenant dans $\mathfrak{c}_{101}$ est décrit table **??**.

bit lu	sous arbre correspondant	*Alist*
1	sous arbre gauche	$(0,1)$
	$\downarrow$	
0	sous arbre droit	$(00,10)$
	$\downarrow$	
1	sous arbre gauche	$(000,001,100,101)$

Table 2: *Alist* associée au mot binaire 101 correspondant au blade e_{13} de $\mathfrak{c}$.

Le calcul de *Alist* étant terminé, nous passons au calcul de *Blist*. Comme vu dans la section **??**, les éléments de *Blist* sont obtenus en calculant le "ou exclusif" entre k et les mots binaires de *Alist*. Les éléments de *Blist* associées au mot binaire 101 sont donc $(101,100,001,000)$. Avec le résultat de *Alist* et *Blist*, nous reconstituons les produits associés à la feuille $\mathfrak{c}_5 = \mathfrak{c}_{101}$:

$$\begin{array}{c} a_0.b_5 \\ a_1.b_4 \\ a_4.b_1 \\ a_5.b_0 \end{array}$$

Table 3: Produits obtenus pour le calcul de la feuille $\mathfrak{c}_5$.

Pour que ces produits correspondent à ceux de la table **??**, on cherche à déterminer le signe associé à chaque produit $a_i.b_j$.

3.3.5. Calcul du signe du produit

Cette partie présente le calcul du signe associé à un produit $\mathfrak{a}_u.\mathfrak{b}_v$ ce qui correspond à la recherche du signe associé à deux mots binaires u et v. Le signe associé à deux mots binaires peut être déterminé à partir de la méthode de calcul de signes de l'algorithme utilisé dans Gaigen. Cette méthode consiste à décaler chaque bit du mot binaire v puis après chaque décalage, déterminer le nombre de bits à 1 en commun avec les bits du mot binaire u.

Supposons qu'il faille chercher le signe associé à deux mots binaires $u = 001$ et $v = 011$. Les bits de v sont tout d'abord décalés à droite $v = 001$ puis le nombre $nbits$ de 1 en commun entre u et v est déterminé. On trouve $nbit = 1$. Le signe associé est $-1^{nbits} = -1$. Le signe est donc négatif.

3.3.6. Complexité de l'approche

Pour évaluer la complexité de cette approche, on considère que le calcul du signe peut être effectué en temps constant. Ainsi La complexité est évaluée en terme de parcours de la liste *Alist*. L'algorithme construit la liste de nœuds de $\mathfrak{a}$ et de $\mathfrak{b}$ intervenant dans le calcul de chaque feuille de $\mathfrak{c}$. Si on suppose qu'à un certain niveau de récursion n, le nombre d'éléments de *Alist* est m alors l'ajout à cette liste du mot binaire 1 se fera en m parcours tout comme l'ajout du mot binaire 0. Par ailleurs, pour chaque niveau de récursion, on effectue 3 ajouts de mots binaires à la même liste. On en déduit le nombre d'appels récursifs :

$$\sum_{i=1}^{d}(3^i) = -\frac{3}{2}(1-3^d) \tag{10}$$

La complexité de cet algorithme est donc en $O(3^d)$, si d est la dimension. Les complexités des deux derniers algorithmes sont équivalentes, ce dernier algorithme est parallélisable contrairement à l'algorithme récursif de la section **??**. Par exemple, la première contribution de $\mathfrak{a}$ dans le calcul de $\mathfrak{c}_{10}$ nécessite d'avoir calculé les deux premières contributions de $\mathfrak{a}$ dans le calcul de $\mathfrak{c}_{11}$.

4. Produit géométrique

Nous allons maintenant étendre la méthode proposée à l'autre opérateur fondamental de l'algèbre géométrique, appelé le produit géométrique. Avant de présenter la méthode, nous décrivons le calcul du produit géométrique. Le produit géométrique est usuellement noté sans symbole. Pour plus de lisibilité, on note toutefois $\mathbf{c} = \mathbf{a} * \mathbf{b}$, le résultat du produit géométrique de $\mathbf{a}$ par $\mathbf{b}$. Le multivecteur $\mathbf{c}$ peut s'écrire de la manière suivante :

$$\mathbf{c} = \sum_{k=0}^{2^d} c_k \mathbf{E}_k = \sum_{i=0}^{2^d} a_i \mathbf{E}_i * \sum_{j=0}^{2^d} b_j \mathbf{E}_j \tag{11}$$

Les propriétés du produit géométrique sont les suivantes :

Linéarité et distributivité	:	$\mathbf{a} * (\mathbf{b} + \mathbf{c}) = \mathbf{a} * \mathbf{b} + \mathbf{a} * \mathbf{c}$
Associativité	:	$\mathbf{a} * (\mathbf{b} * \mathbf{c}) = (\mathbf{a} * \mathbf{b}) * \mathbf{c}$

D'après la propriété de distributivité du produit géométrique, le résultat $\mathbf{a} * \mathbf{b}$ devient donc :

$$\mathbf{c} = \sum_{i=0}^{2^d} \sum_{j=0}^{2^d} a_i b_j (s_{ij}\, \mathbf{E}_i * \mathbf{E}_j) \tag{12}$$

Avec s_{ij} le signe du produit géométrique entre deux vecteurs de base qui dépend de l'ordre dans lequel ce produit est effectué.

4.1. Méthode naïve

Tout comme le produit extérieur, la première approche de calcul du produit géométrique consiste à calculer le produit géométrique entre chaque vecteur de base de $\mathbf{a}$ et de $\mathbf{b}$.

De manière analogue au produit extérieur, la complexité de cette approche est déterminée par le nombre de parcours des multivecteurs. Sachant que le nombre de vecteurs de base d'un multivecteur de dimension d est 2^d. Le nombre de parcours effectués est donc de $2^{2d} = 4^d$. La complexité de l'algorithme est donc en $O(4^d)$. Cette complexité ne pourra être diminuée. En effet, le produit géométrique entre 2 vecteurs de base non nuls et dépendants est non nul.

4.2. Méthode récursive

La formule récursive associée au produit géométrique est la suivante :

$$\begin{aligned}
&\mathfrak{a}^n * \mathfrak{b}^n = (\mathfrak{a}_0^{n+1}, \mathfrak{a}_1^{n+1})^n * (\mathfrak{b}_0^{n+1}, \mathfrak{b}_1^{n+1})^n \\
&\text{Si } n < d \ , \ \mathfrak{a}^n * \mathfrak{b}^n = \\
&\qquad (\mathfrak{a}_1^m * \mathfrak{b}_0^m + \overline{\mathfrak{a}}_0^m * \mathfrak{b}_1^m \ , \mathfrak{a}_0^m * \mathfrak{b}_0^m + \overline{\mathfrak{a}}_1^m * \mathfrak{b}_1^m)^n \\
&\text{Si } n = d \ , \ \mathfrak{a}^n * \mathfrak{b}^n = \ \mathfrak{a}^d * \mathfrak{b}^d
\end{aligned} \tag{13}$$

Avec $m = n+1$

L'approche récursive consiste à développer la formule (**??**) jusqu'aux feuilles de l'arbre. La complexité de cette approche est déterminée en fonction du nombre d'appels récursifs. Les quatre termes de la formule (**??**) induisent chacun un appel récursif. On en déduit qu'à chaque niveau de récursion le nombre d'appels récursifs est multiplié par quatre. Comme le nombre d'appels récursif est $d-1$, alors le nombre d'appels récursif est le suivant :

$$\sum_{i=1}^{d} 4^i = -\frac{4}{3}\left(1-4^d\right) \tag{14}$$

La complexité de cette algorithme est donc en $O(4^d)$, ce qui est équivalent à la précédente approche. Cette approche n'est cependant pas parallélisable et le nombre d'appels récursif est supérieur au nombre de parcours de la méthode naïve. La partie suivante décrit l'algorithme itératif se basant sur cette approche récursive. On cherche à ce que le nombre de parcours ne soit pas supérieur à 4^d.

4.3. Méthode proposée

4.3.1. Développement du produit géométrique

Le but de notre approche est d'étendre au produit géométrique la méthode vue dans la section précédente. De manière analogue au chapitre précédent, la formulation récursive de l'équation (**??**) génère, pour chaque feuille de $\mathfrak{c} = \mathfrak{a} * \mathfrak{b}$, un ensemble de vecteurs de bases $\mathfrak{a}^d * \mathfrak{b}^d$. Chacune de ces feuilles est parcourue plusieurs fois par le calcul récursif et par conséquent reçoit plusieurs contributions de **a** et de **b**. Par exemple, le développement de la formule récursive en dimension 3 donne le résultat suivant :

$\mathbf{e}_{123}$	$\mathbf{e}_{12}$	$\mathbf{e}_{13}$	$\mathbf{e}_1$	$\mathbf{e}_{23}$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{1}$
c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
a_7b_0	a_7b_1	$-a_7b_2$	$-a_7b_3$	a_7b_4	a_7b_5	$-a_7b_6$	$-a_7b_7$
a_6b_1	a_6b_0	a_6b_3	a_6b_2	$-a_6b_5$	$-a_6b_4$	$-a_6b_7$	$-a_6b_6$
$-a_5b_2$	$-a_5b_3$	a_5b_0	a_5b_1	a_5b_6	a_5b_7	$-a_5b_4$	$-a_5b_5$
a_4b_3	a_4b_2	a_4b_1	a_4b_0	a_4b_7	a_4b_6	a_4b_5	a_4b_4
a_3b_4	a_3b_5	$-a_3b_6$	$-a_3b_7$	a_3b_0	a_3b_1	$-a_3b_2$	$-a_3b_3$
$-a_2b_5$	$-a_2b_4$	$-a_2b_7$	$-a_2b_6$	a_2b_1	a_2b_0	a_2b_3	a_2b_2
a_1b_6	a_1b_7	$-a_1b_4$	$-a_1b_5$	$-a_1b_2$	$-a_1b_3$	a_1b_0	a_1b_1
a_0b_7	a_0b_6	a_0b_5	a_0b_4	a_0b_3	a_0b_2	a_0b_1	a_0b_0

Table 4: Développement de $\mathbf{c} = \mathbf{a} * \mathbf{b}$ en dimension 3.

4.3.2. Construction de *Alist* et *Blist*

Comme pour le produit extérieur, nous cherchons à déterminer combien de produits a_ib_j sont nécessaires pour le calcul d'une feuille $\mathfrak{c}_k$ et à identifier ensuite ces produits. L'indiçage utilisé est le même que précédemment. Cet indiçage est ajouté à la formule récursive (**??**) de la manière suivante :

$$\begin{aligned}
&\mathfrak{a}_u^n * \mathfrak{b}_v^n = (\mathfrak{a}_{u0}^{n+1}, \mathfrak{a}_{u1}^{n+1})^n * (\mathfrak{b}_{v0}^{n+1}, \mathfrak{b}_{v1}^{n+1})^n \\
&\text{Si } n < d \ , \ \mathfrak{a}_u^n * \mathfrak{b}_v^n = \\
&\qquad (\mathfrak{a}_{u1}^m * \mathfrak{b}_{v0}^m + \overline{\mathfrak{a}}_{u0}^m * \mathfrak{b}_{v1}^m \ , \mathfrak{a}_{u0}^m * \mathfrak{b}_{v0}^m + \overline{\mathfrak{a}}_{u1}^m * \mathfrak{b}_{v1}^m)^n \\
&\text{Si } n = d \ , \ \mathfrak{a}_u^n * \mathfrak{b}_v^n = \ \mathfrak{a}_u^d * \mathfrak{b}_v^d
\end{aligned} \tag{15}$$

De manière analogue à la section précédente, on peut extraire une construction récursive d'ensemble de mots binaires de $\mathfrak{a}$ et $\mathfrak{b}$. On appelle *Alist* et *Blist* ces constructions récursives. Les définitions récursive de *Alist* et *Alist* sont les suivantes :

$$\begin{aligned}
&\text{Si } n < d \ , \ \left\{Alist^n\right\} = \\
&\quad \left(\left\{Alist(1)^m, Alist(0)^m \ \right\}, \left\{Alist(0)^m, Alist(1)^m \ \right\}\right) \\
&\text{Si } n = d \ , \ \left\{Alist^n\right\} = \ \left\{Alist\right\}
\end{aligned} \tag{16}$$

$$\begin{aligned}
&\text{Si } n < d \ , \ \left\{Blist^n\right\} = \\
&\quad \left(\left\{Blist(0)^m, Blist(1)^m \ \right\}, \left\{Blist(0)^m, Blist(1)^m \ \right\}\right) \\
&\text{Si } n = d \ , \ \left\{Blist^n\right\} = \ \left\{Blist\right\}
\end{aligned} \tag{17}$$

Avec $m = n+1$.

4.3.3. Méthode itérative de calcul de *Alist* et *Blist*

Nous effectuons maintenant une analyse de la définition récursive de *Alist* et de *Blist* pour extraire une méthode itérative de calcule du produit géométrique. La figure **??** montre le développement des formules récursive de *Alist*.

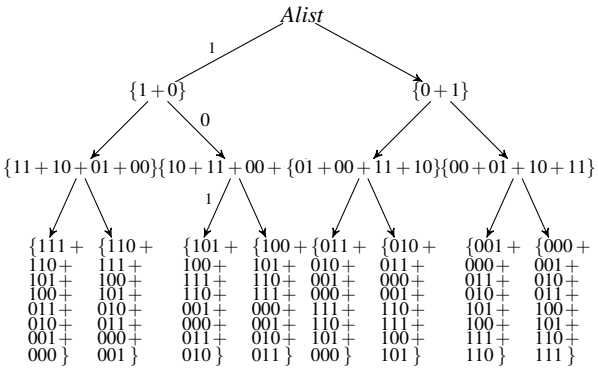

Figure 6: Développement de *Alist* en dimension 3.

La figure **??** montre le développement de *Blist* en dimension 3.

Les formules récursives de *Alist* et de *Blist* montrent que, quelque soit le niveau de récursion n, le nombre de mots

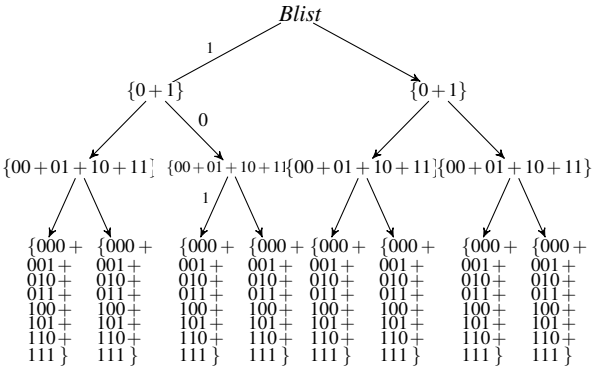

Figure 7: Développement de *Blist* en dimension 3.

binaires de *Alist* et de *Blist* est multiplié par deux. On en déduit le nombre de produits p en fonction du mot binaire identifiant une feuille c_k de c :

$$p = 2^d \tag{18}$$

avec d la dimension.

On remarque qu'à un niveau de récursion n, la construction des coefficients de $\mathfrak{b}$ reste la même quelque soit la situation du chemin (à droite ou à gauche) dans l'arbre de *Blist*. De plus, on remarque (peut être prouvé par récurrence) que l'ensemble de mots binaires obtenus forme la liste des mots binaires consécutifs de longueurs n allant de $(\underbrace{00\cdots00}_{n})$ jusqu'à $(\underbrace{11\cdots11}_{n})$.

L'analyse de la formule récursive (**??**) montre qu'il est possible de déterminer l'ensemble des mots binaires de $\mathfrak{a}$ associée à une feuille de $\mathfrak{c}$. Pour ce faire, on définit tout d'abord une liste de mots binaires *Rlist*. Cette liste est composée, à une profondeur n, des mots binaires consécutifs de longueur n partant de $(\underbrace{00\cdots00}_{n})$ jusqu'à $(\underbrace{11\cdots11}_{n})$. Ainsi la combinaison des mots binaires de *Alist* associée à un noeud k peut être déterminée par somme des éléments de *Rlist* avec le mot binaire k. La somme des éléments de *Rlist* avec k correspond ici à une somme bit à bit.

Prenons l'exemple de la recherche des combinaisons d'indices de *Alist* associée au mot binaire 101. Le noeud correspond au mot binaire 101 est de profondeur 3. La table **??** montre la construction de *Rlist* et de *Alist*.

Rlist	k	*Alist*
000	101	101 + 000 = 101
001	101	101 + 001 = 100
010	101	101 + 010 = 111
011	101	101 + 011 = 110
100	101	101 + 100 = 001
101	101	101 + 101 = 000
110	101	101 + 110 = 011
111	101	101 + 111 = 010

Table 5: Construction de *Rlist* et *Alist* associée au mot binaire 101.

La recherche itérative des combinaisons d'indices $\mathfrak{a}$ et $\mathfrak{b}$ étant déterminée, l'étape suivante consiste à identifier le signe associé à un mot binaire de *Alist* et un mot binaire de *Blist*.

4.3.4. Calcul du signe du produit

Le signe de chaque produit peut être déterminé de manière analogue à la section précédente à partir de la variante de la méthode de calcul de signes de Gaigen [**?**]. Supposons qu'il faille chercher le signe associé à deux mots binaires $u = 001$ et $v = 011$. Les bits de v sont tout d'abord décalés à droite $v = 001$ puis le nombre de 1 en commun $nbits$ entre u et v est déterminé. On trouve $nbit = 1$ bit en commun. Le signe associé est $-1^{nbits} = -1$, le signe est donc négatif.

4.3.5. Complexité de l'approche

La complexité de la méthode est évaluée en terme de nombre d'opérations effectué pour toutes les feuilles de $\mathfrak{c}$. On sait, d'après la formule (**??**) que le nombre d'opérations par feuille est de 2^d et que le nombre de feuilles est de 2^d. Par conséquent, le nombre total d'opérations à effectuer dans le pire des cas est de $2^d * 2^d = 4^d$. La complexité de cet algorithme est donc en $O(4^d)$, où d est la dimension. Notons que le nombre de parcours est inférieur au nombre d'appels récursif de la méthode récursive vue précédemment.

5. Résultats expérimentaux

Dans cette partie, nous présentons tout d'abord les implantations utilisées pour ensuite décrire les tests effectués permettant l'évaluation de la méthode proposée.

5.1. Implantations

5.1.1. Types d'implantation

Pour la suite, nous séparons les implantations en deux types. Cette séparation se base sur le niveau d'optimisation de l'implantation. Le premier type d'implantation est composé de méthodes qui effectuent les produits de l'algèbre indépendamment de la nature du multivecteur (grade, objet géométrique en particulier, ...). Le deuxième type d'implantation adapte la représentation et les calculs effectués en fonction de l'objet géométrique en spécialisant les multivecteurs. Gaigen et Gaalop appartiennent à ce type d'implantation.

5.1.2. Approches implantées

Nous avons produit plusieurs implantations en C++. La première est celle proposée dans cet article. La structure de données permettant de représenter l'arbre binaire est un tableau de 2^d cases. Les deux produits présentés sont implantés par

des fonctions template en utilisant la méta-programmation. L'utilisation de cette méta-programmation [**?**] permet d'une part de produire une implantation générique en dimension et d'autre part d'effectuer des calculs à la compilation, ce qui permet d'éviter de les faire à l'exécution. Ainsi, les calculs de *Alist*, de *Blist* et du signe sont effectués à la compilation.

Pour pouvoir évaluer cette implantation avec d'autres implantations de même type, nous avons programmé deux méthodes. Ces deux méthodes sont dérivées de Gaigen et Gaalop mais n'effectuent aucune spécialisation. La première, basée sur Gaigen, calcule les produits de l'algèbre géométrique pour tous les blades non nuls de **a** et **b**. La deuxième, basée sur Gaalop, utilise une table de multiplication d'un multivecteur par un autre. Cette table est calculée à la compilation et stocke le résultat de produits des vecteurs de base quelque soit la nature du multivecteur.

Afin de nous comparer à Gaigen et Gaalop, nous avons programmé une approche basée sur la méthode proposée et qui spécialise les multivecteurs selon leur grade. Les calculs générés sont vectorisés en utilisant des instructions SIMD [**?**]. L'intérêt de la vectorisation, ici, est la possibilité de tirer profit du caractère parallélisable de la méthode proposée.

5.2. Tests

Nous présentons ci-dessous les résultats de deux différents types de tests. Le premier type compare les approches sans spécialisation. Le deuxième type de tests donne un aperçu des performances de notre méthode à base d'instructions SIMD comparée à Gaigen.

5.2.1. Tests sans spécialisations

Le premier test consiste à déterminer l'intersection de deux sphères de rayons fixés (voir figure **??**) et comparer le temps de calcul à celui des méthodes sans spécialisations.

Le deuxième test consiste à comparer les temps de calcul du produit extérieur entre un multivecteur de grade 1 et un multivecteur de grade 2, pour différentes dimensions allant de 5 jusqu'à 9.

Le troisième et dernier test reproduit le deuxième test mais avec le produit géométrique.

5.2.1.1. Premier test

Le premier test consiste à déterminer l'intersection de deux sphères de rayons fixés (voir figure **??**). Nous nous plaçons dans un espace de sous-espaces à 32 dimensions, il s'agit de l'espace le plus communément utilisé. Nous avons vu dans l'introduction, une façon de calculer le vecteur représentant une sphère à partir de 4 vecteurs positions. Il est également

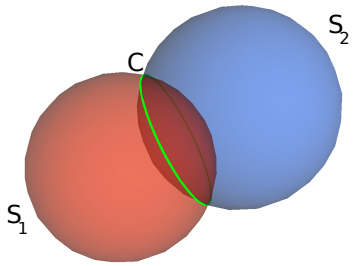

Figure 8: Intersection entre deux sphères. Le cercle **c** est obtenu par intersection des deux sphères $\mathbf{s}_1$ et $\mathbf{s}_2$.

possible de calculer le vecteur d'une sphère à partir du vecteur position du centre de cette sphère et du rayon de la sphère. Avec cette dernière méthode, l'intersection est simplement calculée en effectuant le produit extérieur entre les vecteurs des deux sphères. Nous répétons le calcul de l'intersection 10^5 fois en modifiant à chaque itération la position du centre de la deuxième sphère. La performance de nos algorithmes est mesurée en comparant les temps de calcul de ces intersections pour les trois approches vues précédemment. La table **??** montre le temps de calcul de l'intersection pour ces cinq approches.

Approches	Temps de calcul (s)
(1) Méthode basée sur Gaalop	1.3507
(2) Méthode basée sur Gaigen	1.5302
(3) Implantation proposée	0.8712

Table 6: Temps de calcul de l'intersection entre deux sphères pour trois approches différentes

Notons que pour des implantations ayant le même niveau d'optimisation $\big((1),(2)$ et $(3)\big)$, l'approche proposée dans cet article est la plus performante (40% plus rapide).

5.2.1.2. Deuxième test

Dans ce paragraphe, nous cherchons à déterminer l'évolution du temps de calcul lorsque la dimension de l'espace augmente. Pour ce faire, nous effectuons le produit extérieur entre un multivecteur de grade 1 et un multivecteur de grade 2 pour différentes dimensions de l'espace vectoriel. Le résultat de ce test est montré sur la figure **??**. On constate que l'approche décrite dans cet article est en moyenne plus performante que les méthodes dérivées de Gaigen ou Gaalop.

5.2.1.3. Troisième test

Ce test reproduit le deuxième test mais avec le produit géométrique. Les résultats sont montrés sur la figure **??**. On remarque que notre méthode devance les méthodes dérivées de Gaigen et Gaalop.

5.2.2. Test avec spécialisations

Dans cette partie, nous comparons notre méthode avec instructions SIMD avec Gaigen. Pour ce faire, nous effec-

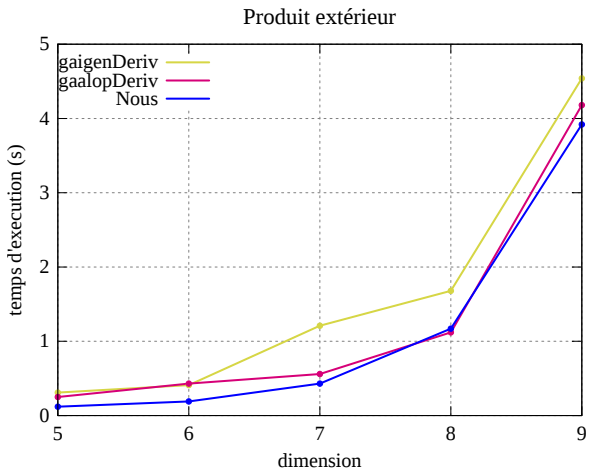

Figure 9: Evolution du temps de calcul du produit extérieur entre un multivecteur de grade 1 et un multivecteur de grade 2 pour cinq approches différentes.

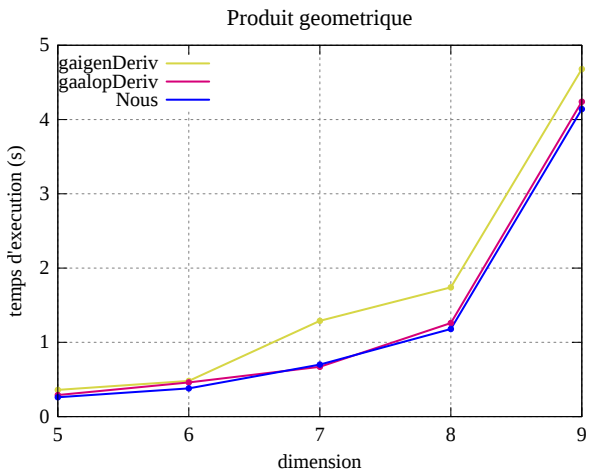

Figure 10: Evolution du temps de calcul du produit extérieur entre un multivecteur de grade 1 et un multivecteur de grade 2 pour cinq approches différentes.

tuons le produit extérieur entre d'un multivecteur de grade 1 et un multivecteur de grade 2 en dimension 5. Ce calcul est répété 10^5 fois. On constate que notre approche est 20% plus rapide que Gaigen.

5.2.3. Bilan des tests

Nous avons constaté que l'approche proposée semble être performante que ce soit avec ou sans spécialisation. Il conviendra pour la suite d'approfondir les tests effectués entre notre approche et Gaigen.

6. Conclusion

Dans ce travail, nous avons établi une nouvelle méthode itérative de calcul des produits de l'algèbre géométrique. Cette méthode permet de calculer efficacement les produits de l'algèbre sans se préoccuper de la dimension. Nous avons montré que la méthode proposée avait de bonnes performances comparées aux approches spécialisées et non spécialisées. Nous chercherons pour la suite à approfondir les tests effectués en haute dimension entre la méthode vue dans cet article et Gaigen. Un test envisagé consiste à tester le calcul d'intersection de surfaces quadratiques présenté dans [**?**].