Computational Aspects of Geometric Algebra Products of Two Homogeneous Multivectors

Stéphane Breuils, Vincent Nozick, Akihiro Sugimoto

Computational Aspects of Geometric Algebra Products of Two Homogeneous Multivectors

Stephane Breuils, Vincent Nozick and Akihiro Sugimoto

Abstract. This paper addresses the study of the complexity of products in geometric algebra. More specifically, this paper focuses on both the number of operations required to compute a product, in a dedicated program for example, and the complexity to enumerate these operations. In practice, studies on time and memory costs of products in geometric algebra have been limited to the complexity in the worst case, where all the components of the multivector are considered. Standard usage of Geometric Algebra is far from this situation since multivectors are likely to be sparse and usually full homogeneous, i.e., having their non-zero terms over a single grade. We provide a complete computational study on the main Geometric Algebra products of two full homogeneous multivectors, that are outer, inner, and geometric products. We show tight bounds on the number of the arithmetic operations required for these products. We also show that some algorithms reach this number of arithmetic operations.

Mathematics Subject Classification (2010). Primary 99Z99; Secondary 00A00.

Keywords. Geometric Algebra, Clifford Algebra, Computational complexity, Arithmetic operations.

1. Introduction

Geometric Algebra, also called Clifford algebra, presents intuitive solutions for problems related to geometry. Its theory is more and more investigated in various research fields like physics, mathematics or computational geometry, see [21, 22, 9] for some examples. In contrast, in the computer science field, the study of computational aspects of the Geometric Algebra operators is still limited. Studying computational aspects through complexity study has started thanks to the pioneering work of Fontijne et al. [13, 14], which gave some results about complexity of Geometric Algebra products in the worst

case. The worst case here means all the components of a multivector are considered in the product, even if the very large majority of them is likely to be zero and leads to useless computation.

Their study is based on the most widespread approach to deal with products in Geometric Algebras, using fast binary indices and per-bit operators like the exclusive OR operator, commonly referred as XOR. This method is originally based on the Walsh function presented by Hagmark et al. in [19], to iteratively construct Clifford algebra products. Hereafter, we refer to such an approach as the XOR method.

The most commonly used Geometric Algebra implementations are based on this XOR method using fast binary indices and per-bit operators, to generate products, see [2, 7, 10, 13, 11] for example.

1.1. Notation

The major part of the notation used in this paper is the same as in [4]. Also following the usages of [10] and [25], lower-case bold letters refer to vectors (vector $\mathbf{a}$) and lower-case non-bold letters refer to multivector coordinates (coefficient a). Upper-case bold letters denote blades (blade $\mathbf{A}$), see [10]. Other multivectors are denoted with upper-case non-bold letters (multivector A). Lower-case and Frakture letters denote multivector expressed over a tree structure (multivector $\mathfrak{a}$), this notion is detailed in Section 6. The part of grade k of a multivector A is denoted by $\langle A \rangle_k$. The total number of basis blades is 2^d, where d is the number of basis blades $\mathbf{e}_i$ of grade 1.

This paper focuses on only three products, namely, the outer product denoted by "$\wedge$", the inner product denoted by "$\cdot$", and the geometric product denoted by "$*$" for the sake of clarity. The inner and outer products are strongly related to the geometric product [10]. Indeed, given two multivectors A and B of respective grade k_a and k_b, the inner and outer product can be formulated as:

$$A \cdot B = \langle A * B \rangle_{|k_a - k_b|}, \qquad A \wedge B = \langle A * B \rangle_{k_a + k_b}. \tag{1}$$

Even though this formulation is simple, it is not computationally optimal. The core of the paper is precisely to address this point. There also exist more operators such as the dual or the inverse; see [10] for a more exhaustive list. However, all these operators can be obtained from the three aforementioned operators, see [20].

Throughout this paper, a basis blade of grade $k > 0$ will be denoted using the set theory notation. Similarly to the notation in [25], we consider an orthogonal basis $\mathcal{B} = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_d\}$, with d being the vector space dimension. Thus, a basis blade of grade g is denoted by

$$\mathbf{e}_{\boldsymbol{\mu}} = \mathbf{e}_{\mu_1} \wedge \mathbf{e}_{\mu_2} \wedge \cdots \wedge \mathbf{e}_{\mu_g} = \mathbf{e}_{\mu_1 \mu_2 \dots \mu_g}, \text{ where } \boldsymbol{\mu} = \{\mu_1, \mu_2, \dots, \mu_g\} \tag{2}$$

with a Greek letter as the subscript, and $\boldsymbol{\mu}$ an ordered set. Thus, the cardinal $|\boldsymbol{\mu}| = k$ of a set $\boldsymbol{\mu}$ refers to the grade of the k-blade $\mathbf{e}_{\boldsymbol{\mu}}$. If $k = 0$, $\mathbf{e}_{\boldsymbol{\mu}} = \mathbf{e}_{\emptyset} = \mathbf{1}$, where $\mathbf{1}$ is the 0-blade that supports scalars. To avoid sign ambiguity, this paper considers the basis blades of k-blades to be ordered (in ascending order

for example), meaning that the expression $\mathbf{e}_{21}$ would actually be represented by $-\mathbf{e}_{12}$ beforehand. This is actually a common practice in Geometric Algebra implementations. We note that $\boldsymbol{\mu} \subseteq \mathcal{P}(\mathcal{B})$ where $\mathcal{P}$ is the power set (the set of all the subsets of a set). Finally, the notation $\binom{d}{k}$ represents the binomial coefficient (choose k from d).

1.2. Full homogeneous multivectors

In a d-dimensional vector space, a *full homogeneous multivector* $\mathbf{A}$ of grade k is a homogeneous multivector with all its component of grade k being non-zero:

$$\begin{cases} \langle \mathbf{A} \rangle_k = \sum\limits_{|\boldsymbol{\mu}|=k} a_{\boldsymbol{\mu}} \mathbf{e}_{\boldsymbol{\mu}}, & a_{\boldsymbol{\mu}} \in \mathbb{R} - \{0\} \\ \langle \mathbf{A} \rangle_{l \neq k} = 0 \end{cases} \tag{3}$$

Note that $\mathbf{A} \in \bigwedge^k$ where $\bigwedge^k = \text{span}(\mathbf{e}_{\boldsymbol{\mu}}, |\boldsymbol{\mu}| = k)$ is the set of all basis blades of grade k in a d-dimensional vector space. Furthermore, $dim \bigwedge^k = \binom{d}{k}$.

Multivectors are, in their practical usage, likely to be homogeneous; e.g. see the representation of any geometric object in conformal geometric algebra [10]. Geometric Algebra usage also leads to non-homogeneous multivectors, like versors (see rotors, translators, etc. [10]). However, they are very limited in terms of their variety of grades and thus can be efficiently represented as a sum of homogeneous multivectors, so that their study is still relevant in this paper.

Some other elements can be homogeneous multivectors, but not full homogeneous multivectors. In this case, the study presented in this paper will give an estimation, with a complexity a bit overestimated. But in practical use, especially with vectorized programming, computing this few zero elements can be faster than detecting them to exclude them from the computation.

Moreover, considering a multivector as a list of arbitrary basis blades and coefficients leads to extremely complicated complexity study. For these reasons, this paper deals only with homogeneous multivectors. This assumption does not limit the scope of the paper. Indeed, in case the multivector is not homogeneous, then it is still defined as the sum of homogeneous multivectors. Furthermore, the operators of Geometric Algebra considered in this paper ($\wedge$, $\cdot$ and $*$) are distributive with respect to the addition. Then, any products between non-homogeneous multivectors can be reduced to some products of homogeneous multivectors.

1.3. Orthogonal and non-orthogonal basis

Although Section 1.1 refers to the set theory notation associated with an orthogonal basis, this paper also covers non-diagonal metric matrices, either directly in a product algorithm or computed via a basis change before and after the product computation.

In this latter situation, the product is internally performed with a diagonal metric matrix (regular or degenerated). This basis change can be applied automatically to the input multivectors and an inverse basis change for the

resulting multivector [4]. Such a basis change theoretically has a $\mathcal{O}(n^2)$ complexity where n is the size of the homogeneous multivector. However, in practice, if performed with matrices, the basis change algorithm in $\mathcal{O}(n^2)$ is often optimized in $\mathcal{O}(n)$ due to matrices' sparsity. Another very common approach to perform the basis change consists just in defining multivector variables to specify the basis transformation for the considered element, see [6, 8] for example.

1.4. Contributions

While many researches are conducted in the field of Geometric Algebra applications, we observe that the study of computational aspects of the Geometric Algebra operators is quite limited. To address this lack of study on the product's complexity, this paper focuses on the comparison between the theoretical minimum number of arithmetic operations required to encode each product, and the complexity of existing algorithms to compute such products.

The theoretical minimum number of arithmetic operations refers in this paper to the minimum number of operations resulting in the development of a product by keeping only the relevant operations. In practice, it is precisely the operations found in an optimal pre-computed source code for a specified product. On the other hand, existing algorithms are either dedicated to produce this source code or to compute the products at run-time (without pre-computed products), especially for high dimensional Geometric Algebra where the source code would be too big to be pre-computed. Thus, in addition to the cost of the product itself, these algorithms have to support beforehand some supplementary tasks like finding some product sign or selecting the right coefficients. These tasks produce an additional complexity cost that can greatly affect the performances. This paper aims to study the existing algorithms' complexity in regard to the theoretical best performance.

The paper is organized as follows: Section 2 reviews existing algorithms to compute Clifford algebra products and shows that the residual cost mentioned above is usually significant. Sections 3, 4, and 5 present the theoretical minimum number of arithmetic operations for the outer, inner and geometric product, respectively. These results are summarized in Table 1 and are considered as the target results for our complexity study. Then, Section 6 focuses on a specific algorithm [4] and details its complexity for each product. This paper shows that the resulting complexity of [4] reaches the optimal pre-computed source code complexity. This result, summarized in Table 2, is strictly better than the other state-of-the-art approaches.

There exist various algorithms to compute Clifford algebra products. They differ on the theoretical support used to design the computation, as well as on their resulting complexity. A good practical algorithm should lead to a fast implementation, and be numerically accurate, robust and memory efficient. This paper focuses on the speed aspect of the implementations via their algorithm complexity study.

TABLE 1. Number of arithmetic operations required for the products of two full homogeneous multivectors of respective grades g_a and g_b in a d-dimensional vector space, where $\mathcal{I}$ is the set of grades $\{|g_a - g_b|, |g_a - g_b| + 2, \ldots, g_a + g_b\}$.

Outer product	$2\binom{d}{g_a+g_b}\binom{g_a+g_b}{g_a}$
Inner product	$2\binom{d}{g_c}\binom{d-g_c}{\frac{1}{2}(g_a+g_b-g_c)}$
Geometric product	$2\binom{d}{g_a}\binom{d}{g_b}$

TABLE 2. Complexity of the products between two full homogeneous multivectors of respective grades g_a and g_b in a d-dimensional vector space, by the recursive approach over a prefix tree as defined by [4].

Recursive outer product	$\mathcal{O}\left(\binom{d}{g_a+g_b}\binom{g_a+g_b}{g_a}\right)$
Recursive inner product	$\mathcal{O}\left(\binom{d}{g_c}\binom{d-g_c}{\frac{1}{2}(g_a+g_b-g_c)}\right)$
Recursive geometric product	$\mathcal{O}\left(\binom{d}{g_a}\binom{d}{g_b}\right)$

2. Common Clifford algebra products [1]

2.1. Double sum over homogeneous multivectors

Most of the Clifford algebra products' computation are based on a double sum over the components of the two operands. In the full homogeneous multivectors case, any multivectors $\mathbf{A}$ of grade g_a can be represented by,

$$\mathbf{A} = \sum_{|\boldsymbol{\mu}|=g_a} a_{\boldsymbol{\mu}} \mathbf{e}_{\boldsymbol{\mu}} \tag{4}$$

where d is the dimension of the vector space. Given two full homogeneous multivectors $\mathbf{A}$ and $\mathbf{B}$, a straightforward solution to compute the number of operations required for their product is to sum over the $\binom{d}{g_a}$ terms of the first multivector $\mathbf{A}$, combined to the sum over the $\binom{d}{g_b}$ terms of the second multivector $\mathbf{B}$, like in

$$\mathbf{A} \odot \mathbf{B} = \sum_{|\boldsymbol{\mu}|=g_a} \sum_{|\boldsymbol{\nu}|=g_b} a_{\boldsymbol{\mu}} b_{\boldsymbol{\nu}} \mathbf{e}_{\boldsymbol{\mu}} \odot \mathbf{e}_{\boldsymbol{\nu}} \tag{5}$$

[1] There is, of course, only one product in a Geometric Algebra, namely, the geometric (Clifford) product. We follow common practice in also calling the derived bilinear operators inner and outer "products".

where $\odot$ is any distributive product over the addition (outer, inner or geometric in this paper). Hereafter, this method will be referred as the double sum computation. Each iteration of this double sum results in a product of scalars and one addition or subtraction (possibly just a sign change for the first element of the sum). Thus, the total number of required arithmetic operations $p_{\wedge}^{\mathrm{ds}}$ (where 'ds' stands for double sum) is

$$p_{\wedge}^{\mathrm{ds}} = p_{\cdot}^{\mathrm{ds}} = p_{*}^{\mathrm{ds}} = 2\binom{d}{g_a}\binom{d}{g_b}. \tag{6}$$

However, in practical situations, for all elements of the double sum (Equation (5)), one actually may have to identify the right coefficients a_i and b_j, have to compute the adequate sign associated to the product $a_i b_j$, as well as the resulting blade that will receive this computed coefficient. The way to perform these tasks makes all the difference between the existing algorithms.

The following sections present various algorithms to compute Clifford Algebra products, and their respective complexity. For the sake of conciseness, we focus on the geometric product.

2.2. Chevalley's recursive method

Chevalley's recursive method can be found in textbooks that introduce the theory of Clifford algebra products. Considering homogeneous multivectors, the recursive formula can be presented as:

$$\begin{aligned}
&\mathbf{e}_{\mu_1\cdots\mu_{g_a-1}\mu_{g_a}} * \mathbf{e}_{\nu_1\cdots\nu_{g_b-1}\nu_{g_b}} \\
&\qquad = \mathbf{e}_{\mu_1\cdots\mu_{g_a-1}}\left((\mathbf{e}_{\mu_{g_a}} \rfloor \mathbf{e}_{\nu_1\cdots\nu_{g_b-1}\nu_{g_b}}) + \mathbf{e}_{\mu_{g_a}} \wedge \mathbf{e}_{\nu_1\cdots\nu_{g_b-1}\nu_{g_b}}\right) \\
&\qquad = \mathbf{e}_{\mu_1\cdots\mu_{g_a-2}}\Bigg(\left(\mathbf{e}_{\mu_{g_a-1}} \rfloor (\mathbf{e}_{\mu_{g_a}} \rfloor \mathbf{e}_{\nu_1\cdots\nu_{g_b-1}\nu_{g_b}}) + \mathbf{e}_{\mu_{g_a}} \wedge \mathbf{e}_{\nu_1\cdots\nu_{g_b-1}\nu_{g_b}}\right) \\
&\qquad\qquad + \mathbf{e}_{\mu_{g_a-1}} \wedge \left((\mathbf{e}_{\mu_{g_a}} \rfloor \mathbf{e}_{\nu_1\cdots\nu_{g_b-1}\nu_{g_b}}) + \mathbf{e}_{\mu_{g_a}} \wedge \mathbf{e}_{\nu_1\cdots\nu_{g_b-1}\nu_{g_b}}\right)\Bigg),
\end{aligned} \tag{7}$$

where $\rfloor$ denotes the left contraction of two blades, see [10].

Complexity: Abłamowicz et al. in [1] proposed an algorithm of this method between two blades $\mathbf{A}$ and $\mathbf{B}$ of grade g_a and g_b, and by linearity of the products to general multivectors. In their proposed implementation, the sign associated with the product is stored in tables. A fair comparison with other methods should include the sign computation complexity, which is at least $\mathcal{O}(g_b)$ since each outer product requires up to grade of $\mathbf{B}$ permutations. Concerning the core of the algorithm, at each recursive depth, two products are performed. Hence, the complexity of the product between two blades is exponential with respect to the grade g_a of the first blade, i.e., in $\mathcal{O}(2^{g_a})$. This algorithm product is repeated for each blade of both multivectors. Thus, the

complexity of this product for full homogeneous multivectors is in

$$\mathcal{O}\left(g_b\binom{d}{g_a}\binom{d}{g_b}2^{g_a}\right). \tag{8}$$

This method is very elegant and is the only method that handles a non-symmetric bilinear form. For symmetric bilinear forms, however, Chevalley's method leads to multiple travels over the same recursive sub-trees, resulting in an exponential complexity, that makes it one of the slowest methods to compute any product in Clifford algebra.

2.3. Rota-Stein method

The paper [1] also presents an iterative algorithm based on Rota-Stein functions. The key formula to compute the geometric product between a blade $\mathbf{e}_{\boldsymbol{\mu}}$ and $\mathbf{e}_{\boldsymbol{\nu}}$ is to decompose the product as follows

$$\mathbf{e}_{\boldsymbol{\mu}} * \mathbf{e}_{\boldsymbol{\nu}} = \sum_{i=1}^{|\boldsymbol{\mu}|}\sum_{j=1}^{|\boldsymbol{\nu}|} \text{sign} \times \mathbf{e}_{(i)\mu}\mathbf{e}_{(j)\nu} \tag{9}$$

where $\mathbf{e}_{(i)\mu}$ and $\mathbf{e}_{(j)\nu}$ refers to the fine subsets of the set $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$, whose cardinal (grade) is i and j. The "fine subsets" are selected according to some constraints related to the considered product, among all possible subsets of cardinal i and j supported by $\mathbf{e}_\mu$ and $\mathbf{e}_\nu$.

Complexity: Here again, the computation of the sign is performed using a table. The most computationally expensive part of the algorithm is the decomposition of the blades. By considering only an orthogonal basis, the decomposition can be performed linearly with respect to the grades of the two multivectors. Finally, using the bilinearity of the products, the complexity of the product for any homogeneous multivectors is,

$$\mathcal{O}\left(\binom{d}{g_a}\binom{d}{g_b}g_a \times g_b^2\right). \tag{10}$$

Here, g_b is raised to the power of 2 since the computational cost of the sign involves up to g_b permutations for each decomposition. Although this method performs exponentially better than Chevalley's method (Section 2.2), its complexity is still far less effective than the recent methods described later in this paper.

2.4. XOR

The XOR method consists in representing any basis blades as an ordered array, where each element is associated to a basis vector (of grade 1) of the algebra. An array element gets the binary value `1` if the corresponding basis vector is used to describe the considered basis blade, and `0` otherwise. In the following example, the basis blade represented is $\mathbf{e}_{134}$.

$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$
1	0	1	1

$\rightarrow\ \mathbf{e}_{134}$

Given a set of $d = p + q$ basis vectors where p basis vectors square to 1 and q basis vectors square to -1, let us consider the geometric product between two basis blades **A** and **B** with respective binary representation a and b. The Walsh method, introduced by Hagmark and Lounesto in [19], consists in first computing the sign of the product between **A** and **B**, and then to compute the binary representation of the resulting basis blade using an XOR operator between a and b:

$$\mathbf{e}_{\mathtt{a}} * \mathbf{e}_{\mathtt{b}} = (-1)^{\sum_{i=1}^{p} \mathtt{a}_i \mathtt{b}_i} \text{walsh}\big(\mathtt{a}, h(\mathtt{b})\big) \mathbf{e}_{\mathtt{a}\texttt{XOR}\mathtt{b}} \tag{11}$$

The Walsh function is defined as:

$$\text{walsh}(\mathtt{a}, \mathtt{b}) = (-1)^{\sum_{i=1}^{d} \mathtt{a}_i \mathtt{b}_i}, \tag{12}$$

and $h(\mathtt{b})$ corresponds to the inverse of the gray code of b, computed as

$$h(\mathtt{b})_i = \left(\sum_{j=1}^{i} \mathtt{b}_j \right) \text{mod}(2). \tag{13}$$

Fontijne et al. [13] proposed a simpler and more computer friendly alternative based on a "convolution" between the two binary indices a and b. The convolution consists of right-shifting each bit of a until it is zero. At each iteration, the number of ones in common between the shifted index and the other index (i.e., the Hamming weight) is counted. The sign is obtained by raising -1 to the power of this sum. Like for the Walsh method, this method computes the resulting blade with an XOR operator between a and b.

Complexity: The Walsh method and the convolution approach have both the same time complexity. In both methods, the computational cost of these operations is linear to the dimension d. Indeed, for any grade, the number of right-shifting is d. Again, this computation is repeated for each blade of the multivectors. Hence, the complexity for full homogeneous multivectors is:

$$\mathcal{O}\left(d \binom{d}{g_a} \binom{d}{g_b} \right). \tag{14}$$

2.5. Matrix based multivector products

Leopardi [24] introduces a matrix based approach to design the products of the template library GluCat [23]. It consists in a fast conversion of both input multivectors into matrices and of the resulting matrix converted back into multivector. The concept of the conversion is based on a recursive process that could be compared to a fast Fourier transform. It was proved in the paper that the complexity to compute the conversion is in $\mathcal{O}(d \times 2^d)$. With this method, the products are performed in the matrix domain.

Complexity: The cost to compute the real matrix representation of the product is negligible compared to the product itself. Then, the computational

complexity of the product is the cost of a matrix multiplication. With the two homogeneous multivectors, the resulting complexity would be in

$$\mathcal{O}\left(\binom{d}{g_a}\binom{d}{g_b}\binom{d}{g_c}\right). \tag{15}$$

2.6. Multiplicative basis method

Fontijne in [13] defined a framework where blades are represented as outer or geometric products of basis vectors. Sousa and Fernandes in [26] further developed this approach and demonstrated impressive performances in high dimensional vector spaces; however, this approach is limited to k-blades and does not handle non-factorizable k-vectors (see Section 2.9.3 of [10] for more details about k-blades and k-vectors).

2.7. Recursive method over prefix trees

Finally, Breuils et al. [4] presented a recursive scheme to compute Clifford algebra products over prefix trees. This algorithm is based on a previous method using binary trees [3]. Let us consider the product of two multivectors $\mathbf{A}$ and $\mathbf{B}$ of respective grade g_a and g_b. Equation (5) describes the double loop over their respective components. For each couple of components, one first has to check if the result is non-zero (unlike in $\mathbf{e}_1 \wedge \mathbf{e}_{12}$) before computing the product, sign and resulting basis blade. This test can actually be avoided by representing a multivector over a prefix tree and by using the proper recursive formulas, as presented in [4], saving a significant amount of time. Moreover, this recursive approach embeds a constant time sign computation, when most common methods usually compute it in $\mathcal{O}(\min(g_a, g_b))$. A large part of the present paper (Section 6) is dedicated to its complexity study for full homogeneous multivectors.

3. Outer product

This section is about the minimum number of arithmetic operations required to compute an outer product between two full homogeneous multivectors. For an algorithm that computes outer products, finding the resulting sign has a cost, i.e., the cost of counting the number of required permutations to have resulting basis vectors in the canonical order. In contrast, the number of arithmetic operations computed below corresponds to the number of operations found on a pre-computed code performing the product between two homogeneous multivectors. This code would already include the correct signs, thus the complexities presented below do not include any sign computation part.

3.1. Properties

We denote by $\mathbf{A} \wedge \mathbf{B}$ the outer product between two homogeneous multivectors $\mathbf{A}$ and $\mathbf{B}$ with grades g_a and g_b in the d-dimensional vector space. This

product $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}$ can be written as:

$$\mathbf{C} = \sum_{|\lambda|=g_c} c_\lambda \mathbf{e}_\lambda = \left(\sum_{|\mu|=g_a} a_\mu \mathbf{e}_\mu \right) \wedge \left(\sum_{|\nu|=g_b} b_\nu \mathbf{e}_\nu \right). \tag{16}$$

As shown in [18], the resulting multivector is homogeneous and its grade g_c is

$$g_c = g_a + g_b. \tag{17}$$

Note that the grade of the resulting multivector has to be lower than or equal to the dimension of the vector space:

$$g_a + g_b \leq d, \tag{18}$$

for the result to be non-zero.

3.2. Number of arithmetic operations

As mentioned in Section 2.7, there exist products that result in zero, even though their respective components are non-zero (like in $2\mathbf{e}_1 \wedge 3\mathbf{e}_{12}$). This section focuses on the number of operations found in an outer product, ignoring such products intrinsically resulting in zero. In practice, this number corresponds to the minimum number of arithmetic operations for the outer product, i.e. the number of operations found on a pre-computed code performing the outer product between two homogeneous multivectors.

Theorem 3.1. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be homogeneous multivectors with grades* g_a *and* g_b *in a d-dimensional vector space . The theoretical number* $p_\wedge^{\text{th}}$ *of arithmetic operations involved in the outer product* $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}$ *with grade* g_c*, where* $g_c = g_a + g_b \leq d$ *(d is the dimension of the vector space) is given by*

$$p_\wedge^{\text{th}} = 2\binom{d}{g_a + g_b}\binom{g_a + g_b}{g_a}. \tag{19}$$

Proof. The outer product $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}$ can be considered as the process of splitting any basis blade of $\mathbf{C}$ in all possible two basis blades of respective grade g_a and g_b. This is equivalent to finding all the sub-blades of the basis blades of $\mathbf{C}$ whose grade is g_a. We know that there are $\binom{d}{g_a+g_b}$ possible blades whose grade is $g_a + g_b$. On the other hand, the number of possible sub-blades of grade g_a in any blade whose grade is $g_a + g_b$ is given as follows:

$$\binom{g_a + g_b}{g_a}. \tag{20}$$

Note that the above equation remains the same if we replace g_a by g_b. This comes from the fact that by definition $g_b = g_c - g_a$. From the symmetry property of the binomial coefficient, we have

$$\binom{g_a + g_b}{g_a} = \binom{g_a + g_b}{g_a + g_b - g_a} = \binom{g_a + g_b}{g_b}. \tag{21}$$

As shown in Equation (5), any step of the double loop consists of a product of scalars, followed by an addition to add this result in the corresponding basis

blade, i.e., two operations. Hence, we obtain the total number of required arithmetic operations by

$$p_{\wedge}^{\text{th}} = 2\binom{d}{g_a+g_b}\binom{g_a+g_b}{g_a}. \tag{22}$$

□

Example. Let $\mathbf{a}$ be a full vector and B a full bivector ($g_a = 1, g_b = 2$). In a 3-dimensional vector space ($d = 3$), we have

$$\mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3, \quad B = b_1\mathbf{e}_{12} + b_2\mathbf{e}_{13} + b_3\mathbf{e}_{23}. \tag{23}$$

Then

$$C = \mathbf{a} \wedge B = (-a_2b_2 + a_1b_3 + a_3b_1)\mathbf{e}_{123} \tag{24}$$

and it requires 3 products and 3 additions, resulting in $3 + 3 = 2\binom{3}{3}\binom{3}{1} = 6$ arithmetic operations. Whereas if $d = 4$,

$$\begin{gathered}\mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3 + a_4\mathbf{e}_4,\\ B = b_1\mathbf{e}_{12} + b_2\mathbf{e}_{13} + b_3\mathbf{e}_{14} + b_4\mathbf{e}_{23} + b_5\mathbf{e}_{24} + b_6\mathbf{e}_{34}\end{gathered} \tag{25}$$

the result is

$$\begin{aligned}C = \quad & (-a_2b_2 + a_1b_3 + a_3b_1)\mathbf{e}_{123} + (-a_2b_3 + a_1b_5 + a_4b_1)\mathbf{e}_{124} + \\ & (-a_3b_3 + a_1b_6 + a_4b_2)\mathbf{e}_{134} + (-a_3b_5 + a_2b_6 + a_4b_4)\mathbf{e}_{234}\end{aligned} \tag{26}$$

and it requires 12 products and 12 additions, resulting in $12 + 12 = 2\binom{4}{3}\binom{3}{1} = 24$ arithmetic operations.

3.3. Comparison with the double sum computation

To compare the outer product complexity using the double sum $p_{\wedge}^{\text{ds}}$ of Equation (6), with the really required operations $p_{\wedge}^{\text{th}}$ of Equation (19), let us compute the ratio between the two formulas as follows:

$$\frac{p_{\wedge}^{\text{th}}}{p_{\wedge}^{\text{ds}}} = \frac{\binom{d}{g_a+g_b}\binom{g_a+g_b}{g_a}}{\binom{d}{g_a}\binom{d}{g_b}}. \tag{27}$$

Using the trinomial revision property as defined in Chapter 5 of [17], we have

$$\frac{p_{\wedge}^{\text{th}}}{p_{\wedge}^{\text{ds}}} = \frac{\binom{d}{g_a}\binom{d-g_a}{g_b}}{\binom{d}{g_a}\binom{d}{g_b}}. \tag{28}$$

After simplification $\left(\forall 0 \leq g_a \leq d, \binom{d}{g_a} \neq 0\right)$, we have

$$\frac{p_{\wedge}^{\text{th}}}{p_{\wedge}^{\text{ds}}} = \frac{\binom{d-g_a}{g_b}}{\binom{d}{g_b}}. \tag{29}$$

The binomial coefficient $\binom{n}{k}$ increases as n increases when k is fixed. Hence, this fraction is less than 1. In practice, Equation (19) may result in high improvements with respect to Equation (6). As an example, let us assume that we compute the outer products of two trivectors in the algebra allowing to apply projective transformation of quadric surface, i.e., 8-dimensional vector space [16]. Then, the product using a double sum as in Equation (6) requires approximately 5 times more arithmetic operations than with Equation (19): 560 outer products instead of 3136 in a 8-dimensional vector space, meaning that a large part of the double sum iterations are actually useless.

4. Inner product

4.1. Properties

The inner product $\mathbf{C}$ between two multivectors $\mathbf{A}$ and $\mathbf{B}$ with respective grades g_a and g_b is defined by

$$\mathbf{C} = \sum_{|\boldsymbol{\lambda}|=g_c} c_{\boldsymbol{\lambda}} \mathbf{e}_{\boldsymbol{\lambda}} = \left(\sum_{|\boldsymbol{\mu}|=g_a} a_{\boldsymbol{\mu}} \mathbf{e}_{\boldsymbol{\mu}} \right) \cdot \left(\sum_{|\boldsymbol{\nu}|=g_b} b_{\boldsymbol{\nu}} \mathbf{e}_{\boldsymbol{\nu}} \right). \tag{30}$$

This product is also distributive with respect to the addition. The resulting multivector is homogeneous and its grade is

$$g_c = |g_a - g_b|. \tag{31}$$

Note that when $g_a > g_b$, the product corresponds to the right contraction as defined in [10]. Whereas, when $g_b > g_a$, the resulting product is the left contraction. When $g_b = g_a$, the product becomes the scalar product.

4.2. Number of arithmetic operations

Theorem 4.1. *The number $p_{\cdot}^{\text{th}}$ of the arithmetic operations involved in the inner product $\mathbf{A} \cdot \mathbf{B}$ between two homogeneous multivectors $\mathbf{A}$ and $\mathbf{B}$ with respective grades g_a and g_b is given by*

$$p_{\cdot}^{\text{th}} = 2 \binom{d}{|g_a - g_b|} \binom{d - |g_a - g_b|}{\frac{1}{2}(g_a + g_b - |g_a - g_b|)}. \tag{32}$$

Proof. Using the set notation defined in Section 1.1, the inner product between any two basis blades can be written as

$$\mathbf{e}_{\boldsymbol{\lambda}} = \mathbf{e}_{\boldsymbol{\mu}} \cdot \mathbf{e}_{\boldsymbol{\nu}} \qquad \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu} \in \mathcal{P}(\mathcal{B}). \tag{33}$$

By definition of the inner product in an orthogonal basis, we have two cases for $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ that lead to non-zero components.

The first case is $\boldsymbol{\mu} \subseteq \boldsymbol{\nu}$, with $|\boldsymbol{\lambda}| = |\boldsymbol{\nu}| - |\boldsymbol{\mu}|$. By definition, the operation consists of the left contraction. Equation (33) is then rewritten as

$$\exists \boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathcal{P}(\mathcal{B}) \setminus \{\varnothing\}, \boldsymbol{\beta} \cap \boldsymbol{\gamma} = \varnothing, \mathbf{e}_{\boldsymbol{\lambda}} = \mathbf{e}_{\boldsymbol{\beta}} \cdot \mathbf{e}_{\boldsymbol{\beta} \cup \boldsymbol{\gamma}}. \tag{34}$$

In such a case, $g_c = |\boldsymbol{\lambda}| = |\boldsymbol{\gamma}|$ and $\boldsymbol{\lambda} = \boldsymbol{\gamma}$. Computing the number of the products is reduced to determining the number of different possibilities

for $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. If we set $\boldsymbol{\gamma} = \boldsymbol{\lambda}$, then there is only one possibility for $\boldsymbol{\gamma}$. As $|\boldsymbol{\beta}| + |\boldsymbol{\gamma}| \leq d \Rightarrow |\boldsymbol{\beta}| \leq d - |\boldsymbol{\gamma}| = d - g_c$, any possible grades of β lower than or equal to $d - g_c$ is possible. As $|\boldsymbol{\beta}| = g_a$, any combination of g_a in $d - g_c$ is possible.

Similarly to the outer product, any combination of $|\boldsymbol{\lambda}| = g_c$ in d is possible, which results in the number of the products as follows:

$$\binom{d}{g_c}\binom{d-g_c}{g_a}. \tag{35}$$

Furthermore, as $g_a \leq g_b$, $|g_a - g_b| = g_b - g_a$. Then, Equation (35) can be rewritten as

$$\begin{aligned}\binom{d}{g_c}\binom{d-g_c}{\frac{2}{2}g_a} &= \binom{d}{g_c}\binom{d-g_c}{\frac{1}{2}(g_a+g_b-g_b+g_a)} \\ &= \binom{d}{g_c}\binom{d-g_c}{\frac{1}{2}(g_a+g_b-|g_a-g_b|)}.\end{aligned} \tag{36}$$

The second case is the symmetric case $\boldsymbol{\nu} \subseteq \boldsymbol{\mu}$. In this configuration,

$$\exists \boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathcal{P}(\mathcal{B}) \setminus \{\varnothing\}, \boldsymbol{\beta} \cap \boldsymbol{\gamma} = \varnothing, \mathbf{e}_{\boldsymbol{\lambda}} = \mathbf{e}_{\boldsymbol{\beta}\cup\boldsymbol{\gamma}} \cdot \mathbf{e}_{\boldsymbol{\beta}}. \tag{37}$$

In this case, $|\boldsymbol{\lambda}| = |\boldsymbol{\mu}| - |\boldsymbol{\nu}|$. By definition, the operation results in the right contraction. Reasoning as in the previous paragraph leads us to $g_c = |\boldsymbol{\lambda}| = |\boldsymbol{\gamma}|$ and $\boldsymbol{\lambda} = \boldsymbol{\gamma}$. Computing the number of the products is then reduced to determining the number of different possibilities for $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. If we set $\boldsymbol{\gamma} = \boldsymbol{\lambda}$, there is only one possibility for $\boldsymbol{\gamma}$. As $|\boldsymbol{\beta}| + |\boldsymbol{\gamma}| \leq d \Rightarrow |\boldsymbol{\beta}| \leq d - |\boldsymbol{\gamma}| = d - g_c$. Therefore, any possible grade of $\boldsymbol{\beta}$ lower than or equal to $d - g_c$ is possible. As $|\boldsymbol{\beta}| = g_b$, any combination of g_b in $d - g_c$ is possible.

In a similar way as for the outer product, any combinations of $|\boldsymbol{\lambda}| = g_c$ in d are possible, resulting in the number of the products as follows:

$$\binom{d}{g_c}\binom{d-g_c}{g_b}. \tag{38}$$

Furthermore, as $g_b \leq g_a$, $|g_a - g_b| = g_a - g_b$, Equation (38) can be rewritten as

$$\begin{aligned}\binom{d}{g_c}\binom{d-g_c}{\frac{2}{2}g_b} &= \binom{d}{g_c}\binom{d-g_c}{\frac{1}{2}(g_a+g_b-g_a+g_b)} \\ &= \binom{d}{g_c}\binom{d-g_c}{\frac{1}{2}(g_a+g_b-|g_a-g_b|)}.\end{aligned} \tag{39}$$

Finally, as shown in Equation (30), any step of the double loop consists in a product of scalars, followed by an addition to add this result in the fine basis blade, i.e. two operations. Hence, the total number of the required arithmetic operations is

$$p_{\cdot}^{\text{th}} = 2\binom{d}{|g_a-g_b|}\binom{d-|g_a-g_b|}{\frac{1}{2}(g_a+g_b-|g_a-g_b|)}. \tag{40}$$

□

Example. Let $\mathbf{a}$ be a full vector and B a full bivector ($g_a = 1, g_b = 2$). In a 3-dimensional vector space ($d = 3$), we have

$$\mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3, \quad B = b_1\mathbf{e}_{12} + b_2\mathbf{e}_{13} + b_3\mathbf{e}_{23}. \tag{41}$$

The result of the inner product is

$$C = \mathbf{a} \cdot B = (-a_2b_1 - a_3b_2)\mathbf{e}_1 + (-a_3b_3 + a_1b_1)\mathbf{e}_2 + (a_1b_2 + a_2b_3)\mathbf{e}_3 \tag{42}$$

and it requires 6 products and 6 additions, resulting in $6 + 6 = 2\binom{3}{1}\binom{2}{1} = 12$ operations. Whereas if $d = 4$,

$$\begin{array}{c} \mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3 + a_4\mathbf{e}_4 \\ B = b_1\mathbf{e}_{12} + b_2\mathbf{e}_{13} + b_3\mathbf{e}_{14} + b_4\mathbf{e}_{23} + b_5\mathbf{e}_{24} + b_6\mathbf{e}_{34} \end{array} \tag{43}$$

the result is

$$\begin{array}{rc} C = & (-a_2b_1 - a_3b_2 - a_4b_3)\mathbf{e}_1 + (-a_3b_4 + a_1b_1 - a_4b_5)\mathbf{e}_2 \\ & +(-a_4b_6 + a_1b_2 + a_2b_4)\mathbf{e}_3 + (a_1b_3 + a_2b_5 + a_3b_6)\mathbf{e}_4 \end{array} \tag{44}$$

and it requires 12 products and 12 additions, resulting in $12 + 12 = 2\binom{4}{1}\binom{3}{1} = 24$ arithmetic operations.

4.3. Comparison with the double sum computation

As in Section 3.3, let us compute the ratio between the effectively required operations $p_{\cdot}^{\text{th}}$ of Equation (32) and the number of operations $p_{\cdot}^{\text{ds}}$ used for the same product using a double sum as in Equation (6):

$$\frac{p_{\cdot}^{\text{th}}}{p_{\cdot}^{\text{ds}}} = \frac{\binom{d}{|g_a - g_b|}\binom{d - |g_a - g_b|}{\frac{1}{2}(g_a + g_b - |g_a - g_b|)}}{\binom{d}{g_a}\binom{d}{g_b}}. \tag{45}$$

After simplification, we find the gain as follows:

$$\frac{p_{\cdot}^{\text{th}}}{p_{\cdot}^{\text{ds}}} = \frac{\binom{d - \min(g_b, g_a)}{|g_a - g_b|}}{\binom{d}{\max(g_b, g_a)}}. \tag{46}$$

For the sake of readability, the details of the development are shown in Appendix A. As for the outer product case, Equation (32) may result in high improvements with respect to Equation (6). As an example, let us assume that we compute the inner product of two trivectors in a 8-dimensional vector space. Then, Equation (6) requires 28 times more arithmetic operations than Equation (32): 112 arithmetic operations instead of 3136 required to compute the inner product of two trivectors in a 8-dimensional vector space by the double sum. Again, the double sum involves a large part of useless iterations.

5. Geometric product

5.1. Properties

This section deals with the geometric product noted with the sign $*$ for sake of clarity. As mentioned in [25], the possible grades of the multivector $\mathbf{C} = \mathbf{A} * \mathbf{B}$ are

$$g_c \in \mathcal{I} = \{|g_a - g_b|, |g_a - g_b| + 2, \dots, g_a + g_b\}. \tag{47}$$

The geometric product between two multivectors is then defined by

$$C = \sum_{g_c \in \mathcal{I}} \sum_{|\boldsymbol{\lambda}|=g_c} c_{\boldsymbol{\lambda}} \mathbf{e}_{\boldsymbol{\lambda}} = \left(\sum_{|\boldsymbol{\mu}|=g_a} a_{\boldsymbol{\mu}} \mathbf{e}_{\boldsymbol{\mu}} \right) * \left(\sum_{|\boldsymbol{\nu}|=g_b} b_{\boldsymbol{\nu}} \mathbf{e}_{\boldsymbol{\nu}} \right). \tag{48}$$

Note that in contrast to a multivector obtained by the outer product or the inner product, the resulting multivector might not be homogeneous. One might note that this contradicts the assumption that we deal with only full homogeneous multivectors. However, as stated in Section 1, a non-homogeneous multivector is merely the sum of homogeneous multivectors. Moreover, the resulting homogeneous multivectors are also likely to be full. Thus, the assumption still holds.

5.2. Number of arithmetic operations

Theorem 5.1. *The number p_*^{th} of the arithmetic operations involved in the geometric product $\mathbf{A} * \mathbf{B}$ between two homogeneous multivectors $\mathbf{A}$ and $\mathbf{B}$ with respective grades g_a and g_b is given by*

$$p_*^{\text{th}} = 2 \sum_{g_c \in \mathcal{I}} \binom{d}{g_c} \binom{g_c}{\frac{1}{2}(g_a - g_b + g_c)} \binom{d - g_c}{\frac{1}{2}(g_a + g_b - g_c)}, \tag{49}$$

where $\mathcal{I} = \{|g_a - g_b|, |g_a - g_b| + 2, \dots, g_a + g_b\}$.

Proof. The geometric product between any two basis blades can be written as

$$\mathbf{e}_{\boldsymbol{\lambda}} = \mathbf{e}_{\boldsymbol{\mu}} * \mathbf{e}_{\boldsymbol{\nu}} \qquad \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu} \in \mathcal{P}(\mathcal{B}). \tag{50}$$

There are four cases with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$.

The first case is $\boldsymbol{\mu} \cap \boldsymbol{\nu} = \varnothing$, where the geometric product results in the outer product between the basis blades, and the number of products corresponds to Equation (19).

The second case corresponds to $\boldsymbol{\mu} \subseteq \boldsymbol{\nu}$, with $|\boldsymbol{\lambda}| = |\boldsymbol{\nu}| - |\boldsymbol{\mu}|$. By definition, the operation results in the left contraction. The number of the products is addressed in Section 4.2.

The third case corresponds to $\boldsymbol{\nu} \subseteq \boldsymbol{\mu}$, with $|\boldsymbol{\lambda}| = |\boldsymbol{\mu}| - |\boldsymbol{\nu}|$. By definition, the

operation is reduced to the right contraction. The computation of the number of products is already addressed in Section 4.2.

Finally, the last case is the situation where $\boldsymbol{\mu} \cap \boldsymbol{\nu} \neq \emptyset$ but $\boldsymbol{\nu} \nsubseteq \boldsymbol{\mu}$ nor $\boldsymbol{\mu} \nsubseteq \boldsymbol{\nu}$. More precisely, this corresponds to

$$\exists \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathcal{P}(\mathcal{B}) \setminus \{\emptyset\}, \boldsymbol{\alpha} \cap \boldsymbol{\beta} = \emptyset, \boldsymbol{\beta} \cap \boldsymbol{\gamma} = \emptyset, \mathbf{e}_{\boldsymbol{\lambda}} = \mathbf{e}_{\boldsymbol{\alpha} \cup \boldsymbol{\beta}} * \mathbf{e}_{\boldsymbol{\beta} \cup \boldsymbol{\gamma}}. \tag{51}$$

In such a case, $g_c = |\boldsymbol{\lambda}| = |\boldsymbol{\alpha}| + |\boldsymbol{\gamma}|$ and $\boldsymbol{\lambda} = \boldsymbol{\alpha} \cup \boldsymbol{\gamma}$. Computing the number of products is reduced to determining the number of different possibilities for $\boldsymbol{\alpha}, \boldsymbol{\beta}$, and $\boldsymbol{\gamma}$. Let us start with $\boldsymbol{\beta}$. The geometric product acts as the symmetric difference $\triangle$ between the two blades (union minus intersection) of Equation (50) as follows:

$$\mathbf{e}_{\boldsymbol{\lambda}} = \mathbf{e}_{(\boldsymbol{\alpha} \cup \boldsymbol{\beta}) \triangle (\boldsymbol{\beta} \cup \boldsymbol{\gamma})}. \tag{52}$$

The symmetric difference operation leads the cardinal of the resulting blade to be

$$g_c = g_a + g_b - 2|\boldsymbol{\beta}|. \tag{53}$$

Hence, we have

$$|\boldsymbol{\beta}| = \frac{1}{2}(g_a + g_b - g_c). \tag{54}$$

From the fact that $\boldsymbol{\beta} \cap \boldsymbol{\gamma} = \emptyset$ and $\boldsymbol{\beta} \cap \boldsymbol{\alpha} = \emptyset$, we obtain $\boldsymbol{\beta} \cap \boldsymbol{\lambda} = \emptyset$. Thus,

$$\boldsymbol{\beta} \in \mathcal{P}(\mathcal{B}) \setminus \{\boldsymbol{\lambda}, \emptyset\}, |\boldsymbol{\beta}| = \frac{1}{2}(g_a + g_b - g_c). \tag{55}$$

Furthermore,

$$\boldsymbol{\beta} \in \mathcal{P}(\mathcal{B}) \setminus \mathcal{P}(\boldsymbol{\lambda}). \tag{56}$$

Since the set of maximal cardinality in $\mathcal{P}(\mathcal{B}) \setminus \mathcal{P}(\boldsymbol{\lambda})$ is $d - g_c$, the number of possibilities for $\boldsymbol{\beta}$ is the number of combinations of $|\boldsymbol{\beta}|$ in $d - g_c$:

$$\binom{d - g_c}{\frac{1}{2}(g_a + g_b - g_c)}. \tag{57}$$

Note that we have to ensure that $\frac{1}{2}(g_a + g_b - g_c)$ is an integer. Two cases exist: either g_a and g_b have the same parity or not.

If both g_a and g_b have the same parity, then

$$\begin{array}{ll} \exists n \in \mathbb{Z}, g_a + g_b & = 2n, \\ \exists n' \in \mathbb{Z}, |g_a - g_b| & = 2n'. \end{array} \tag{58}$$

Furthermore, g_c is the sum of $|g_a - g_b|$ and an even number, thus g_c is also even. Since the sum of two even numbers is also even, $g_a + g_b - g_c$ is even.

Now, let assume that g_a and g_b do not have the same parity. Then, their sum and their difference are both odd. On the other hand, g_c is the sum of $|g_a - g_b|$ and an even number, indicating that the g_c is odd. Since the difference of two odd numbers is even, $g_a + g_b - g_c$ is even. Hence, in both cases, $g_a + g_b - g_c$ is even.

The number of the combinations for $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ is now computed. We know that $g_c = |\boldsymbol{\lambda}| = |\boldsymbol{\alpha}| + |\boldsymbol{\gamma}|$ and $\boldsymbol{\lambda} = \boldsymbol{\alpha}\boldsymbol{\gamma}$. Thus, the number of the combinations in this case is merely equivalent to the number of possibilities of the outer product associated to g_c and α, γ:

$$\binom{g_c}{|\boldsymbol{\alpha}|} = \binom{g_c}{|\boldsymbol{\gamma}|}. \tag{59}$$

Furthermore, $g_a = |\boldsymbol{\alpha}| + |\boldsymbol{\beta}| \Rightarrow |\boldsymbol{\alpha}| = g_a - |\boldsymbol{\beta}|$ and using the definition of $|\boldsymbol{\beta}|$ in Equation (54) results in the number of possibilities:

$$\begin{aligned} \binom{g_c}{|\alpha|} &= \binom{g_c}{g_a - |\beta|} \\ &= \binom{g_c}{g_a - \frac{1}{2}(g_a + g_b - g_c)} \\ &= \binom{g_c}{\frac{1}{2}(g_a - g_b + g_c)}. \end{aligned} \tag{60}$$

Note that for the same reason as in the above paragraphs, the term $g_a - g_b + g_c$ is even. Furthermore, $g_a - g_b + g_c \geq 0$ because by assumption $g_c > |g_a - g_b|$. This results in the number of products of

$$\binom{d}{g_c}\binom{g_c}{\frac{1}{2}(g_a - g_b + g_c)}\binom{d - g_c}{\frac{1}{2}(g_a + g_b - g_c)}. \tag{61}$$

Equation (48) shows that the geometric product is the sum over all possible grades $g_c \in \mathcal{I}$. Accordingly, Equation (61) yields

$$\sum_{g_c \in \mathcal{I}} \binom{d}{g_c}\binom{g_c}{\frac{1}{2}(g_a - g_b + g_c)}\binom{d - g_c}{\frac{1}{2}(g_a + g_b - g_c)}. \tag{62}$$

Finally, as for the inner and outer product, each operation referred in Equation (62) consists of a product of scalars, followed by an addition to add this result in the fine basis blade, i.e., two operations. Hence, the total number of the required arithmetic operations is

$$2 \sum_{g_c \in \mathcal{I}} \binom{d}{g_c}\binom{g_c}{\frac{1}{2}(g_a - g_b + g_c)}\binom{d - g_c}{\frac{1}{2}(g_a + g_b - g_c)}. \tag{63}$$

□

5.3. Comparison with the double sum computation

As presented for the outer and inner product in Sections 3.3 and 4.3, the comparison between the double sum and the really required operations indicates a significant amount of useless operations for the double sum. In the case of the geometric product, the development of Equation (63) leads to the same result as for the double sum of Equation (6).

Proposition 5.2.

$$\sum_{g_c \in \mathcal{I}} \binom{d}{g_c} \binom{g_c}{\frac{1}{2}(g_a - g_b + g_c)} \binom{d - g_c}{\frac{1}{2}(g_a + g_b - g_c)} = \binom{d}{g_a} \binom{d}{g_b}. \tag{64}$$

The proof is given in Appendix B.

6. Clifford Algebra products and recursive approach over a prefix tree

There exist several recursive methods to compute Clifford algebra products. As stated in Section 2, they are not equivalent in terms of complexity. This section focuses on the prefix tree algorithm introduced by Breuils et al., used in the library Garamon [4]. This section aims to show that this method reaches the complexity related to the theoretical minimal numbers of arithmetic operations for products between two full homogeneous multivectors presented in Sections 3, 4 and 5.

To make the paper self-contained, let us briefly review in Section 6.1 this recursive formulation to define multivectors and Geometric Algebra products (see [4] for more details). We start with the definition of multivectors using the prefix tree structure.

6.1. Multivectors

In the context of [4], each basis blade is associated to a node of a prefix tree and the nodes of depth k in the prefix tree correspond to the basis blades of grade k. The scalar basis blade, denoted by $\mathbf{1}$, is associated with the root node. The vector basis blades are associated with the children of the root node, the bivector basis blades are associated with the children of those nodes, and so on, as illustrated on Figure 1. By construction of the prefix tree, the index of a basis blade associated with a node is prefixed by the indexes of the basis blades associated with its parent nodes. Note that the breadth-first search of the basis blades over the prefix tree results in the list of basis blades in the canonical order. For instance, the list obtained from the prefix tree in Figure 1 is $(\mathbf{1}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_{12}, \mathbf{e}_{13}, \mathbf{e}_{23}, \mathbf{e}_{123})$.

Given a multivector $\mathbf{A}$, let us assume that $\mathfrak{a}_\gamma$ represents a node of the prefix tree, where γ is the set of basis vectors present in the basis blade. For example, the node $\mathfrak{a}_\gamma = \mathfrak{a}_{\{1,2\}}$ corresponds to the node associated with the blade $\mathbf{e}_{12}$ of $\mathbf{A}$. Then, the set of children of any node $\mathfrak{a}_\gamma$ can be recursively defined from depth n to the next depth $n+1$ as follows:

$$\begin{array}{rcl} \text{a node at depth: } n & \rightarrow & \mathfrak{a}_\gamma, \\ \text{its children at depth: } n+1 & \rightarrow & \mathfrak{a}_{\gamma+\mu},\ \mu \in [\max(\gamma)+1, \cdots, d], \end{array} \tag{65}$$

where d is the dimension of the vector space. Note that the function $\max()$ is self-sufficient since the integer is a totally ordered set. Furthermore, the addition sign between two sets (Greek letters) denotes the concatenation of the two sets. However, note that in expressions on sets elements, like

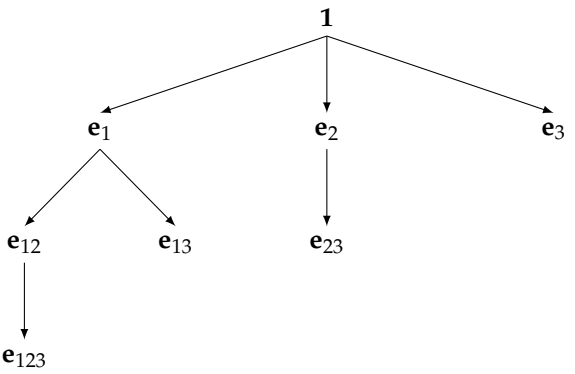

FIGURE 1. Prefix tree structure of the basis blades for a Geometric Algebra whose underlying vector space is of dimension 3.

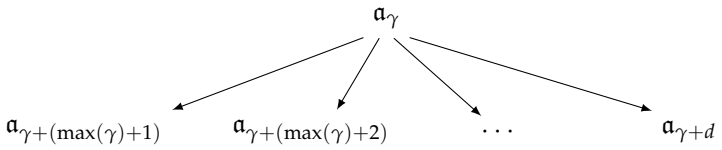

FIGURE 2. Labeling of the siblings of a child node.

"$\max(\gamma)+1$", the addition sign is really an addition on the highest element of the set γ. An illustration of the recursion from a node to its children is given in Figure 2. The starting call of the recursive formula for the breadth-first search is $\mathfrak{a}_0$ at a depth of 0 (grade 0 or scalar). The end of recursion is achieved when a node is a leaf (i.e. $\max(\gamma) = d$).

6.2. Recursive outer product over trees

The recursive outer product used in the prefix tree was introduced by Fuchs and Théry [15] and defined over the binary tree in [3], then it was adapted for the prefix tree in [4]. The resulting complexity of this recursive method for full multivectors in d-dimensional space is $\mathcal{O}(3^d)$ instead of $\mathcal{O}(d \times 4^d)$ for the XOR method, see proof in [4]. This section aims at computing the complexity for full *homogeneous* multivectors. In the following sections, we first recall the recursive outer product for general multivectors and then refine this product for homogeneous multivectors.

Definition 6.1 (Outer product over a prefix tree for general multivectors). Given two general multivectors $\mathbf{A}$ and $\mathbf{B}$, the recursive outer product associated with $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}$ is expressed as:

$$\begin{array}{rl} \text{at depth } n & \\ \text{computation:} & \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma \wedge \mathfrak{b}_\delta \\ \text{recursive calls:} & \mathfrak{c}_{\lambda+\sigma} = \mathfrak{a}_{\gamma+\sigma} \wedge \mathfrak{b}_\delta + \overline{\mathfrak{a}_\gamma} \wedge \mathfrak{b}_{\delta+\sigma} \qquad \sigma \in [\max(\lambda)+1, \cdots, d] \end{array} \tag{66}$$

where the overline denotes the anticommutativity property of the product.

The starting call of this recursive formula is $\mathfrak{c}_0 = \mathfrak{a}_0 \wedge \mathfrak{b}_0$, i.e. at a depth of 0 (grade 0 or scalar). The end of recursion is achieved when a node is a leaf, i.e. when $\max(\lambda) = d$.

Definition 6.2 (Anticommutativity). The construction of the operator that recursively captures the anticommutativity of the outer product is given below. Note that we follow the notation in [3] and apply this operator on a multivector.

$$\begin{array}{ccc} n & \rightarrow & \overline{\mathfrak{a}_\gamma} \\ n+1 & \rightarrow & -\overline{\mathfrak{a}_{\gamma+\mu}}, \quad \mu \in [\max(\gamma)+1, \cdots, d]. \end{array} \tag{67}$$

Let's now focus on the case of homogeneous multivectors, where the grades g_a, g_b, and g_c are known in advance. The recursive product $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}$ can then be slightly modified so that any update of $\mathfrak{c}$ are performed only at depth $g_c = g_a + g_b$.

Definition 6.3 (Recursive outer product of homogeneous multivectors over a prefix tree). Given two full homogeneous multivectors $\mathbf{A}$ and $\mathbf{B}$ of respective grade g_a and g_b, the recursive outer product associated with $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}$ of expected grade g_c is expressed as

$$\begin{array}{rl} \text{at depth } n & \\ \text{computation:} & \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma \wedge \mathfrak{b}_\delta, \qquad \text{if } |\lambda| = g_c \\ \text{recursive calls:} & \mathfrak{c}_{\lambda+\sigma} = \mathfrak{a}_{\gamma+\sigma} \wedge \mathfrak{b}_\delta + \overline{\mathfrak{a}_\gamma} \wedge \mathfrak{b}_{\delta+\sigma}, \qquad \sigma \in [\max(\lambda)+1, \cdots, d] \end{array} \tag{68}$$

where $|\lambda|$ denotes the cardinality of the set λ.

Thus, for homogeneous multivectors, the end of recursion is achieved when a node is a leaf (i.e. $\max(\lambda) = d$) or when the targeted grade g_c is reached (i.e. $|\lambda| = g_c$). Algorithm 1 presents a straightforward way to implement the recursive formulas presented in Definitions 6.1 and 6.3.

Figure 3 illustrates an example of the development of all the recursive outer products in the 3-dimensional vector space. The number of recursive calls depends only on the depth of the recursion, as stated with the following lemma whose proof is given in Appendix C.

Lemma 6.4. *During the recursive product $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}$, all the children $\mathfrak{c}_{\lambda+\sigma}$ of a node $\mathfrak{c}_\lambda$ of the prefix tree corresponding to $\mathbf{C}$ generate the same number of recursive outer products calls. In other words, the siblings at any depth of the prefix tree of $\mathbf{C}$ generate the same number of products.*

Algorithm 1: Pseudocode of the recursive outer product $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}$

```
1 Function outer
      Input: a_γ, b_δ: nodes of multivectors A and B,
             c_λ: nodes of the resulting multivector C
             complement: recursive value ( ±1).
             sign: recursive sign coefficient ( ±1).
2     if |λ| = g_c then // condition to remove for general multivectors
3         c_λ += sign × a_γ × b_δ        // product of components
4     foreach σ ∈ [max(λ) + 1, ··· , d] do
5         // a_{γ+σ} ∧ b_δ
6         outer(a_{γ+σ}, b_δ, c_{λ+σ}, sign × complement, complement)
7         // ā_γ ∧ b_{δ+σ}
8         outer(a_γ, b_{δ+σ}, c_{λ+σ}, sign, −complement)
9 First call: outer( a_0, b_0, c_0, 1, 1)
```

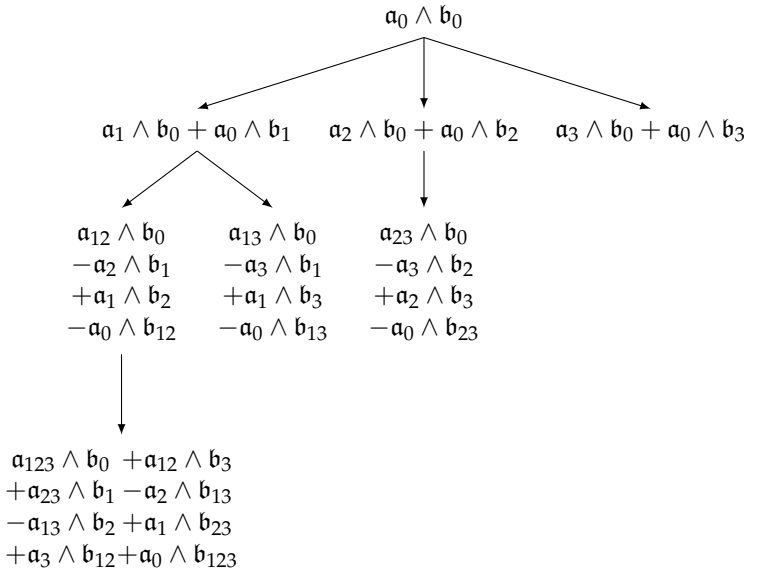

FIGURE 3. Prefix tree structure associated with the recursive outer product for a Geometric Algebra whose underlying vector space is of dimension 3. Note that for a given depth, each node presents the same number of outer products.

Theorem 6.5. *The complexity $c_{\wedge}^{\text{rec}}$ of the recursive outer product $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}$ between two homogeneous multivectors $\mathbf{A}$ and $\mathbf{B}$ of respective grades g_a and g_b, with resulting grade $g_c = g_a + g_b$, is expressed as*

$$c_{\wedge}^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_a + g_b} \binom{g_a + g_b}{g_a} \right), \tag{69}$$

where d is the dimension of the vector space.

Proof. Lemma 6.4 shows that during a recursive outer product, the siblings at any depth (grade) of the resulting prefix tree have the same number of outer products, i.e. the same number of recursive calls. Furthermore, there are $\binom{d}{g_c}$ nodes of grade g_c in the prefix tree represented in the d-dimensional vector space. The number of products of a given depth is thus the multiplication of the binomial coefficient and n_{g_c,g_a} (the number of outer products per node of grade g_c). Hence, the overall complexity is

$$c_{\wedge}^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_c} n_{g_c,g_a} \right). \tag{70}$$

We may focus on the computation of n_{g_c,g_a}, accordingly. The recursive formula of Definition 6.3 shows that at any depth of recursion, there is a sum of two recursive calls to be executed. Both the recursive calls increase the grade of the result. One increases the grade of $\mathfrak{a}$ and the other leaves it unchanged. Applying the recursion in the forward order yields

$$\begin{array}{rcl} n_{0,0} & = & n_{1,1} + n_{1,0} \\ n_{1,1} & = & n_{2,2} + n_{2,1} \\ n_{1,0} & = & n_{2,1} + n_{2,0} \\ & \vdots & \\ n_{g_c-2,g_a-1} & = & n_{g_c-1,g_a} + n_{g_c-1,g_a-1} \\ n_{g_c-1,g_a} & = & n_{g_c,g_a} \\ n_{g_c-1,g_a-1} & = & n_{g_c,g_a} \end{array} \tag{71}$$

When the final recursion is reached for the grade of g_c, two recursive calls n_{g_c,g_a-1} and n_{g_c,g_a+1} (corresponding to the respective ending conditions of the two terms of Equation (68)) are not executed. Now, going backward from the two final recursion equations n_{g_c,g_a} yields the recursive formula

$$n_{g_c,g_a} = n_{g_c-1,g_a} + n_{g_c-1,g_a-1}. \tag{72}$$

We verify that the cases where either $g_c = g_a$ or $g_a = 0$ correspond to a final recursion condition, and, thus, we have $n_{g_c,g_a} = 1$. This recursive definition corresponds to the recursive definition of the binomial coefficient:

$$\binom{g_c}{g_a} = \binom{g_c - 1}{g_a} + \binom{g_c - 1}{g_a - 1}. \tag{73}$$

Hence, the number of recursive calls is

$$n_{g_c,g_a} = \binom{g_c}{g_a} = \binom{g_a+g_b}{g_a}. \tag{74}$$

The complexity of the recursive outer product is

$$c_{\wedge}^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_c} n_{g_c,g_a}\right) = \mathcal{O}\left(\binom{d}{g_a+g_b}\binom{g_a+g_b}{g_a}\right). \tag{75}$$

□

Remark (Recursive outer product in actual implementation). In practice, there are some obvious speed-ups for Algorithm 1 on homogeneous multivectors, as stated in [4]. The first way is to avoid recursive calls on nodes where the operand $\mathfrak{a}$ and $\mathfrak{b}$ lead to grade $g_a + g_b > g_c$. This is introduced in Definition (6.3) as well as in Algorithm 1, line 2. An additional and more sophisticated speed-up is to discard a recursive call on a branch that never reaches the grade of the considered multivector, as shown in blue dashed arrows in Figure 4. These branch discard tests require only binary operators (very fast to compute) and can sometimes remove half of the recursive calls. The pseudocode of this speed-up way for the outer product is presented in Appendix E. The speed-up in running time is clear since it removes some calls in the original algorithm, but the complexity study becomes too complicated to be considered in this paper.

6.3. Recursive inner product

As stated in Section 4.1, when $g_b > g_a$, the inner product is defined by the left contraction whereas it is defined by the right contraction when $g_b \leq g_a$. These two cases are thus treated separately.

The left contraction is a metric product and requires a metric to be defined. Let $\mathrm{M}_{d\times d}$ be the $d \times d$ symmetric matrix defining the vector inner product of the vector space of dimension d. In this context, we assume that the metric matrix is diagonal. If not, the metric matrix is assumed to be diagonalized (see Section 1.3). Thus, the metric will be only referred as its diagonal vector $\mathbf{m} = \mathrm{diag}(\mathrm{M}_{d\times d})$, where $\mathbf{m}(i) = \mathrm{M}_{d\times d}(i,i)$, such that

$$\begin{aligned} \mathbf{m}(1) &= \mathbf{e}_1 \cdot \mathbf{e}_1, \\ \mathbf{m}(2) &= \mathbf{e}_2 \cdot \mathbf{e}_2, \\ &\vdots \\ \mathbf{m}(d) &= \mathbf{e}_d \cdot \mathbf{e}_d. \end{aligned} \tag{76}$$

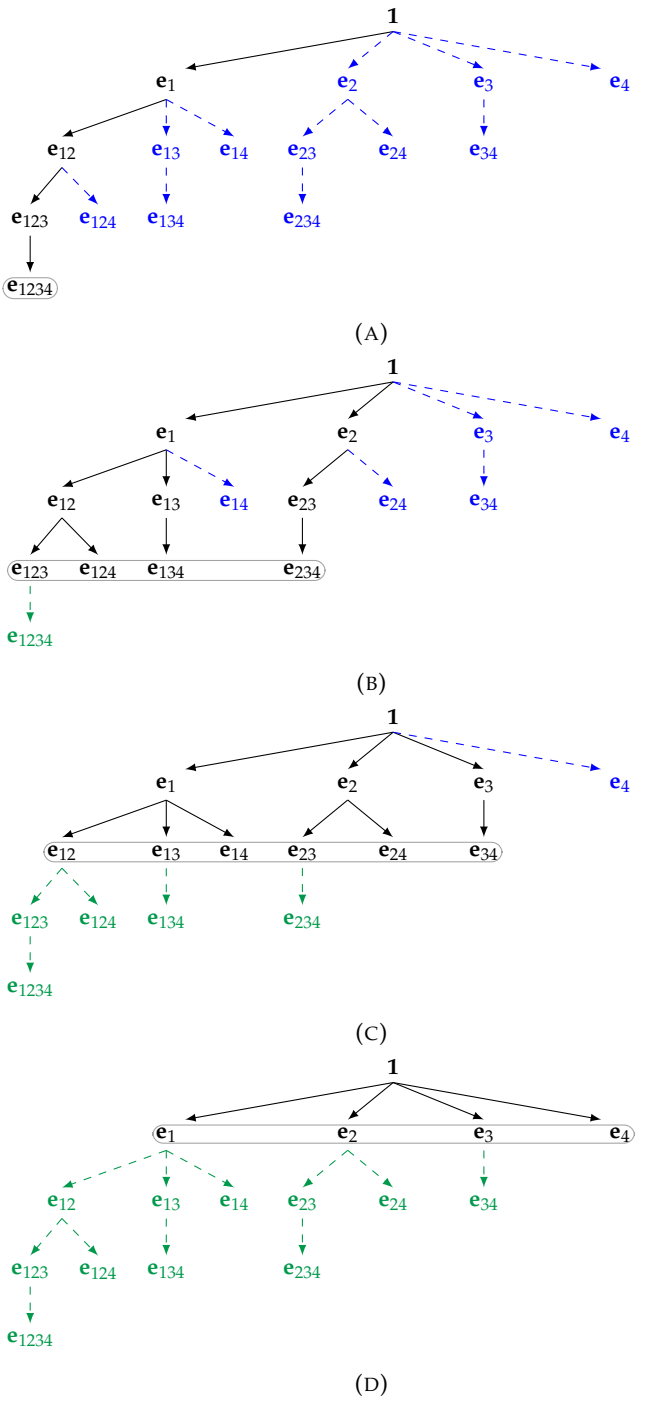

FIGURE 4. Tree structure for some resulting multivectors of grade 4 (A), grade 3 (B), grade 2 (C), grade 1 (D) in a 4-dimensional vector space, taken from the figure 8 of [4]. Useless branches are depicted in green dashed arrows above the targeted multivector and in blue below. The targeted nodes are surrounded in gray.

Definition 6.6 (Recursive left contraction). The construction of the recursive left contraction $\mathfrak{a}\rfloor\mathfrak{b}$ is defined as

$$\begin{aligned} &\text{at depth } n \\ &\text{computation: } \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma\rfloor\mathfrak{b}_\delta, \quad \text{if } |\delta| = g_b \\ &\text{recursive calls 1: } \mathfrak{c}_\lambda = \textstyle\sum_{i=\max(\lambda)+1}^{d} \mathbf{m}(i)\overline{\mathfrak{a}_{\gamma+i}}\rfloor\mathfrak{b}_{\delta+i} \\ &\text{recursive calls 2: } \mathfrak{c}_{\lambda+\sigma} = \overline{\mathfrak{a}_\gamma}\rfloor\mathfrak{b}_{\delta+\sigma} \qquad \sigma \in [\max(\lambda)+1, \cdots, d] \end{aligned} \tag{77}$$

Note that the above recursive formula is equivalent to

$$\begin{aligned} &\text{at depth } n \\ &\text{computation: } \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma\rfloor\mathfrak{b}_\delta, \quad \text{if } |\delta| = g_b \\ &\text{recursive calls: } \mathfrak{c}_{\lambda+\sigma} = \overline{\mathfrak{a}_\gamma}\rfloor\mathfrak{b}_{\delta+\sigma} + \mathbf{m}(\sigma)\overline{\mathfrak{a}_{\gamma+\sigma}}\rfloor\mathfrak{b}_{\delta+\sigma+\psi}, \sigma \in [\max(\lambda)+1, \cdots, d], \\ &\qquad\qquad \psi \in [\max(\lambda)+1, \cdots, d] \end{aligned} \tag{78}$$

Algorithm 2 presents a pseudocode of the recursive left contraction.

Algorithm 2: Recursive left contraction $\mathbf{C} = \mathbf{A}\rfloor\mathbf{B}$

```
1  Function leftCont
       Input: a_γ, b_δ: nodes of multivectors A and B,
              c_λ: nodes of the resulting multivector C
              complement: recursive value ( ±1).
              sign: recursive sign coefficient ( ±1).
              m: diagonal coefficients of the metric matrix.
2      if |δ| = g_b then
3          c_λ += sign × a_γ × b_δ          // product of components
4      foreach σ ∈ [max(λ)+1, ··· , d] do
5          // m(i) ā_{γ+σ}⌋ b_{δ+σ}
6          leftCont(a_{γ+σ}, b_{δ+σ}, c_λ, m(σ) × sign, −complement)
7      foreach σ ∈ [max(λ)+1, ··· , d] do
8          // ā_γ⌋ b_{δ+σ}
9          leftCont(a_γ, b_{δ+σ}, c_{λ+σ}, sign, −complement)
10 First call: leftCont(a_0, b_0, c_0, 1, 1)
```

Definition 6.7 (Recursive right contraction). The construction of the recursive right contraction $\mathfrak{a}\lfloor\mathfrak{b}$ is defined as

$$\begin{aligned} &\text{at depth } n \\ &\text{computation: } \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma\lfloor\mathfrak{b}_\delta, \quad \text{if } |\delta| = g_a \\ &\text{recursive calls 1: } \mathfrak{c}_\lambda = \textstyle\sum_{i=\max(\lambda)+1}^{d} \mathbf{m}(i)\overline{\mathfrak{a}_{\gamma+i}}\lfloor\mathfrak{b}_{\delta+i} \\ &\text{recursive calls 2: } \mathfrak{c}_{\lambda+\sigma} = \mathfrak{a}_{\gamma+\sigma}\lfloor\mathfrak{b}_\delta \qquad \sigma \in [\max(\lambda)+1, \cdots, d] \end{aligned} \tag{79}$$

The algorithm of the recursive right contraction is presented in Algorithm 3.

Algorithm 3: Recursive right contraction $\mathbf{C} = \mathbf{A} \lfloor \mathbf{B}$

```
1  Function rightCont
       Input: a_γ, b_δ: nodes of multivectors A and B,
              c_λ: nodes of the resulting multivector C
              complement: recursive value (±1).
              sign: recursive sign coefficient (±1).
              m: diagonal coefficients of the metric matrix.
2      if |γ| = g_a then
3          c_λ += sign × a_γ × b_δ        // product of components
4      foreach σ ∈ [max(λ) + 1, ··· , d] do
5          // m(i) a_{γ+σ} ⌊ b_{δ+σ}
6          rightCont(a_{γ+σ}, b_{δ+σ}, c_λ, m(σ) × sign, −complement)
7      foreach σ ∈ [max(λ) + 1, ··· , d] do
8          // a_{γ+σ} ⌊ b_δ
9          rightCont(a_{γ+σ}, b_δ, c_{λ+σ}, complement × sign, complement)
10 First call: rightCont(a_0, b_0, c_0, 1, 1)
```

Theorem 6.8. *The complexity $c_\cdot^{\text{rec}}$ of the recursive inner product $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ between two homogeneous multivectors $\mathbf{A}$ and $\mathbf{B}$ of respective grade g_a and g_b, with resulting grade $g_c = |g_a - g_b|$, is expressed as*

$$c_\cdot^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_c}\binom{d-g_c}{\frac{1}{2}(g_a+g_b-g_c)}\right), \tag{80}$$

where d is the dimension of the vector space.

Proof. Lemma 6.4 still holds even for the recursive inner product. Namely, the siblings at any depth (grade) of the resulting prefix tree have the same number of inner products, i.e. the same number of recursive calls. Furthermore, there are $\binom{d}{g_c}$ nodes of grade g_c in the prefix tree represented in the d-dimensional vector space. The number of products of a given depth is thus the multiplication of the binomial coefficient and the number n_{d-g_c,g_a} of inner products per node of grade g_c. The overall complexity is thus

$$c_\cdot^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_c} n_{d-g_c,g_a}\right). \tag{81}$$

To compute this number n_{d-g_c,g_a} of recursive calls, let us focus on the evolution of grade g_a with respect to a depth of recursion g_c. The following proof is divided in two parts. The first part is dedicated to the case $g_a \geq g_b$ while the second part focuses on the case $g_a < g_b$.

In the first case, the considered product is the recursive left contraction, resulting in the multivector $\mathfrak{c} = \mathfrak{a} \rfloor \mathfrak{b}$. As stated in Equation (78), for a given grade g_c of the result, both recursive calls increase grade g_c. The leftmost

term leaves grade g_a unchanged on one hand (and increases the grade of $\mathfrak{b}$):

$$n_{g_c,g_a} \to n_{g_c+1,g_a} \tag{82}$$

On the other hand, the second recursive call of Equation (78) increases g_a (and increases the grade of $\mathfrak{b}$).

$$n_{g_c,g_a} \to n_{g_c+1,g_a+1} \tag{83}$$

Using Equations (82) and (83) with value $d - g_c$ leads to

$$n_{d-g_c-1,g_a+1} \quad = \quad n_{d-g_c,g_a} + n_{d-g_c,g_a+1} \ . \tag{84}$$

In this context, the cases where either $d - g_c = g_a$ or $g_a = 0$ correspond to a final recursion condition, and thus we have $n_{d-g_c,g_a} = 1$. Therefore, Equation (84) corresponds to the recursive definition of the binomial coefficient:

$$n_{d-g_c,g_a} = \binom{d-g_c-1}{g_a+1} = \binom{d-g_c}{g_a} + \binom{d-g_c}{g_a+1}. \tag{85}$$

Hence, the complexity $c_\rfloor^{\text{rec}}$ of the recursive left contraction is

$$c_\rfloor^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_c}\binom{d-g_c}{g_a}\right). \tag{86}$$

In a similar manner, we have the complexity $c_\lfloor^{\text{rec}}$ of the recursive right contraction is

$$c_\lfloor^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_c}\binom{d-g_c}{g_b}\right). \tag{87}$$

Accordingly, the complexity of the recursive inner product is

$$c_\cdot^{\text{rec}} = c_\rfloor^{\text{rec}} + c_\lfloor^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_c}\binom{d-g_c}{\frac{1}{2}(g_a+g_b-g_c)}\right). \tag{88}$$

□

6.4. Recursive geometric product

Similarly to the two other products, let us start with the definition of the recursive geometric product.

Definition 6.9. Given two homogeneous multivectors $\mathbf{A}$ and $\mathbf{B}$ and the set $\mathcal{I} = \{|g_a - g_b|, |g_a - g_b| + 2, \ldots, g_a + g_b\}$, where g_a and g are respectively the grade of $\mathbf{A}$ and $\mathbf{B}$, the recursive geometric product $\mathbf{C} = \mathbf{A} * \mathbf{B}$ is expressed as

$$\begin{aligned}
&\text{at depth } n \\
&\text{computation: } \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma * \mathfrak{b}_\delta, \quad \text{if } |\lambda| \in \mathcal{I}, |\gamma| = g_a \\
&\text{recursive calls 1: } \mathfrak{c}_\lambda = \textstyle\sum_{i=\max(\lambda)+1}^{d} \mathbf{m}(i)\overline{\mathfrak{a}_{\gamma+i}} * \mathfrak{b}_{\delta+i} \\
&\text{recursive calls 2: } \mathfrak{c}_{\lambda+\sigma} = \mathfrak{a}_{\gamma+\sigma} * \mathfrak{b}_\delta + \overline{\mathfrak{a}_\gamma} * \mathfrak{b}_{\delta+\sigma} \qquad \sigma \in [\max(\lambda)+1, \cdots, d]
\end{aligned} \tag{89}$$

The pseudocode for this definition is presented in Algorithm 4.

Algorithm 4: Recursive geometric product $\mathbf{C} = \mathbf{A} * \mathbf{B}$

Function `geoProduct`
  Input: $\mathfrak{a}_\gamma, \mathfrak{b}_\delta$: node of multivectors $\mathbf{A}$ and $\mathbf{B}$,
    $\mathfrak{c}_\lambda$: nodes of the resulting multivector $\mathbf{C}$
    `complement`: recursive value (± 1).
    `sign`: recursive sign coefficient (± 1).
    $\mathbf{m}$: coefficients of the metric.
    $\mathcal{I} = \{|g_a - g_b|, |g_a - g_b| + 2, \dots, g_a + g_b\}$.
  if $|\lambda| \in \mathcal{I}$ **and** $|\gamma| = g_a$ **then**
    $\mathfrak{c}_\lambda$ += `sign` $\times\ \mathfrak{a}_\gamma \times \mathfrak{b}_\delta$ // product of components
  foreach $\sigma \in [\max(\lambda) + 1, \cdots, d]$ **do**
    $// \mathbf{m}(i)\mathfrak{a}_{\gamma+\sigma} * \mathfrak{b}_{\delta+\sigma}$
    `geoProduct`$(\mathfrak{a}_{\gamma+\sigma}, \mathfrak{b}_{\delta+\sigma}, \mathfrak{c}_\lambda, \mathbf{m}(\sigma) \times$ `sign`$, -$`complement`$)$
  foreach $\sigma \in [\max(\lambda) + 1, \cdots, d]$ **do**
    $// \mathfrak{a}_{\gamma+\sigma} * \mathfrak{b}_\delta$
    `geoProduct`$(\mathfrak{a}_{\gamma+\sigma}, \mathfrak{b}_\delta, \mathfrak{c}_{\lambda+\sigma},$ `complement` $\times$ `sign`$,$ `complement`$)$
    $// \overline{\mathfrak{a}_\gamma} * \mathfrak{b}_{\delta+\sigma}$
    `geoProduct`$(\mathfrak{a}_\gamma, \mathfrak{b}_{\delta+\sigma}, \mathfrak{c}_{\lambda+\sigma},$ `sign`$, -$`complement`$)$
First call: `geoProduct`$(\mathfrak{a}_0, \mathfrak{b}_0, \mathfrak{c}_0, 1, 1)$

Theorem 6.10. *The complexity c_*^{rec} of the recursive geometric product $\mathbf{C} = \mathbf{A} * \mathbf{B}$ between two homogeneous multivectors $\mathbf{A}$ and $\mathbf{B}$ of respective grade g_a and g_b, with resulting grade $g_c \in \mathcal{I} = \{|g_a - g_b|, |g_a - g_b| + 2, \dots, g_a + g_b\}$, is expressed as*

$$c_*^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_c}\binom{g_c}{\frac{1}{2}(g_a - g_b + g_c)}\binom{d - g_c}{\frac{1}{2}(g_a + g_b - g_c)}\right), \tag{90}$$

where d is the dimension of the vector space.

The proof can be found in Appendix D.

6.5. Recursive overall complexity

The complexities of the recursive products proposed by [4] are summarized in Tables 2. The next section opens a discussion about these results and the comparison with most commonly used methods, as well as practical implementations.

7. Concluding remarks

7.1. Complexity

The results of Section 6 are twofold. First, it shows that the recursive approach proposed by Breuils et al. [4] for inner and outer products present a better time complexity than the double sum approach used in the XOR method as well as in many others. For the geometric product, this recursive method

performs similarly to other methods, like XOR. Second, it shows that the complexity of this recursive approach is the same as the complexity associated to the number of operations used in precomputed source code. For a brief summary, all these complexities are presented in Tables 1 and 2.

The low complexity of this recursive method is due to two reasons. First, the sign computation in each product is usually quite expensive in state-of-the-art algorithms, when it is in constant time in the presented recursive method. Second, the recursive computation only considers basis blade products that do not intrinsically result in zero, e.g. $\mathbf{e}_{12} \wedge \mathbf{e}_1$ is never considered with this approach since no recursive call leads to this product.

Finally, the study presented in this paper naturally extends to the results of [4] about the worst case situation, where the multivectors are full. In such a situation, the complexity of the recursive method detailed in this paper is exponentially better than the XOR approach.

7.2. Inner product and duality

As stated in Tables 1 and 2, the complexity of the computation of the inner product is worse than the one for the outer product. It is noteworthy that the prefix tree defined in [4] representing a multivector $\mathbf{A}$ implicitly defines its dual $\mathbf{A}^*$ by reading the tree upside down. See [10] for more details about duality. In [4], the authors propose a recursive formula of $\mathbf{C} = \mathbf{A} \wedge \mathbf{B}^*$ with the same complexity as the recursive wedge product studied in this paper. Considering that $\mathbf{A} \cdot \mathbf{B} = (\mathbf{A} \wedge \mathbf{B}^*)^*$, the inner product can actually be computed with the same complexity as the outer product.

7.3. Asymptotic study and hidden constant in Big $\mathcal{O}$

In this paper, we consider computational aspects of Geometric Algebra products of homogeneous multivectors through an asymptotic study. In this context, the complexity of the outer product with the recursive prefix tree approach is asymptotically exponentially more efficient than the XOR approach. Naturally, any asymptotic study may sometimes hide some large constants that are important in practice. This constant can be related to the algorithm itself, as well as practical concerns, like programming language, memory access, hardware, etc. A close analysis requires a study through analytic combinatorics [12], as well a consequent effort in producing benchmarks that are out of the scope of this paper.

7.4. Implementation

This product complexity study naturally raises a subsidiary study about effective implementations of Geometric Algebra products. A first approach consists of pre-computing the products for a given algebra. The resulting code always reaches the best complexity for multivectors. A second approach consists of a syntax simplification of Geometric Algebra expression. In principle, this technique also reaches the best complexity and can sometimes perform even better by first simplifying some complex expressions.

In the context of pre-computed source code, the recursive approach is not necessary of a great help since whatever the method used, the generated source code is the same. However if the source code is generated of the fly, then the recursive approach can be very relevant. This can occur in meta-programming when the products are evaluated at compile time. This can also occur for high-dimensional Geometric Algebras where the source code can not be pre-computed anymore due to memory overflow.

Indeed, for high dimensional Geometric Algebras, the generated source code size can reach gigabytes. In such a situation, like in $\mathbb{R}^{9,6}$ of [5], the products should be computed at run time. A possibility is to use product tables to pre-compute signs and resulting blades. In this situation, each product between $\mathbf{A}$ and $\mathbf{B}$ requires to read all the entries of the table for the grades (g_a,g_b). Some entries lead to a pre-computed product, when many others just result in zero. The complexity is then in $\mathcal{O}(\binom{d}{g_a}\binom{d}{g_b})$ for every product. Thus, the table approach is only optimal for the geometric product, but neither for the outer product nor for the inner product. However, if the basis vector dimension is really high, these tables, which require at least $2^d \times 2^d$ elements for each product, will clearly show some memory limitations. In such high dimensions where complexity computation really matters, the recursive methods presented in Section 6 are not subject to such memory overflow and still benefit from a very favorable time complexity, as stated in Table 2.

In any case, the recommended implementation of the recursive products is the optimized version detailed in Algorithms 5, 6, 7 and 8 of Appendix E.

7.5. Perspective

As a perspective of this paper, we would focus on the computational complexity of products with more than two homogeneous multivectors, as well as their effective implementation. Another natural next step after this theoretical complexity analysis would be a more practical software benchmarking. In term of implementation, the recursive approach suffers from too much conditional statements. A reformulation of the recursive calls may have significant impact on the practical performances.

References

[1] Abłamowicz, R., and Fauser, B. Clifford and Graßmann Hopf algebras via the BIGEBRA package for Maple. *Computer Physics Communications 170*, 2 (2005), 115–130.

[2] Abłamowicz, R., and Fauser, B. On Parallelizing the Clifford Algebra Product for CLIFFORD. *Advances in Applied Clifford Algebras 24*, 2 (2014), 553–567.

[3] Breuils, S., Nozick, V., and Fuchs, L. A Geometric Algebra Implementation using Binary Tree. *Advances in Applied Clifford Algebras 27*, 3 (Sep 2017), 2133–2151.

[4] Breuils, S., Nozick, V., and Fuchs, L. Garamon: A Geometric Algebra Library Generator. *Advances in Applied Clifford Algebras 29*, 4 (Jul 2019), 69.

[5] Breuils, S., Nozick, V., Sugimoto, A., and Hitzer, E. Quadric Conformal Geometric Algebra of $\mathbb{R}^{9,6}$. *Advances in Applied Clifford Algebras 28*, 2 (Mar 2018), 35.

[6] Charrier, P., Klimek, M., Steinmetz, C., and Hildenbrand, D. Geometric algebra enhanced precompiler for C++, OpenCL and Mathematica's OpenCLLink. *Advances in Applied Clifford Algebras 24*, 2 (2014), 613–630.

[7] Colapinto, P. *Spatial computing with conformal geometric algebra*. PhD thesis, University of California Santa Barbara, 2011.

[8] De Keninck, S. Ganja. `https://github.com/enkimute/ganja.js.git`. Accessed: 2019-11-11.

[9] De Keninck, S., and Dorst, L. Geometric Algebra Levenberg-Marquardt. In *Advances in Computer Graphics* (Cham, 2019), M. Gavrilova, J. Chang, N. M. Thalmann, E. Hitzer, and H. Ishikawa, Eds., Springer International Publishing, pp. 511–522.

[10] Dorst, L., Fontijne, D., and Mann, S. *Geometric Algebra for Computer Science, An Object-Oriented Approach to Geometry*. Morgan Kaufmann, 2007.

[11] Fernandes, L. A. F. GATL: Geometric Algebra Template Library. `https://github.com/laffernandes/gatl.git`.

[12] Flajolet, P., and Sedgewick, R. *Analytic combinatorics*. Cambridge University Press, 2009.

[13] Fontijne, D. *Efficient Implementation of Geometric Algebra*. PhD thesis, University of Amsterdam, 2007.

[14] Fontijne, D., Dorst, L., Bouma, T., and Mann, S. GAviewer, interactive visualization software for geometric algebra. *http://www.geometricalgebra.net/downloads.html* (2010).

[15] Fuchs, L., and Théry, L. Implementing geometric algebra products with binary trees. *Advances in Applied Clifford Algebras 24*, 2 (2014), 589–611.

[16] Goldman, R., and Mann, S. R(4, 4) As a Computational Framework for 3-Dimensional Computer Graphics. *Advances in Applied Clifford Algebras 25*, 1 (Mar 2015), 113–149.

[17] Graham, R. L., Knuth, D. E., Patashnik, O., and Liu, S. Concrete mathematics: a foundation for computer science. *Computers in Physics 3*, 5 (1989), 106–107.

[18] Grassmann, H. Die lineale Ausdehungslehre: ein neuer Zweig der Mathematik, dargestellt und durch Anwendungen auf die übrigen Zweige der Mathematik,

wie auch die Statik, Mechanik, die Lehre von Magnetismus und der Krystallonomie erläutert. *Wigand, Leipzig* (1844).

[19] HAGMARK, P.-E., AND LOUNESTO, P. *Walsh Functions, Clifford Algebras and Cayley-Dickson Process*. Springer Netherlands, Dordrecht, 1986, pp. 531–540.

[20] HESTENES, D. Grassmann's vision. In *Hermann Günther Graßmann (1809–1877): Visionary Mathematician, Scientist and Neohumanist Scholar*. Springer, 1996, pp. 243–254.

[21] KANATANI, K. *Understanding Geometric Algebra: Hamilton, Grassmann, and Clifford for Computer Vision and Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2015.

[22] LASENBY, J., HADFIELD, H., AND LASENBY, A. Calculating the Rotor Between Conformal Objects. *Advances in Applied Clifford Algebras 29*, 5 (Oct 2019), 102.

[23] LEOPARDI, P. GluCat: Generic library of universal Clifford algebra templates. `http://glucat.sourceforge.net/`.

[24] LEOPARDI, P. A generalized FFT for Clifford algebras. *Bulletin of the Belgian Mathematical Society-Simon Stevin 11*, 5 (2005), 663–688.

[25] PERWASS, C. *Geometric algebra with applications in engineering*, vol. 4 of *Geometry and Computing*. Springer, 2009.

[26] SOUSA, E. V., AND FERNANDES, L. A. TbGAL: A Tensor-Based Library for Geometric Algebra. *Advances in Applied Clifford Algebras 30*, 2 (2020), 1–33.

Appendix A. Simplification of the ratio of Section 4.3

If $g_a < g_b$, then Equation (45) can be rewritten as

$$\frac{p^{\text{th}}_{\cdot}}{p^{\text{ds}}_{\cdot}} = \frac{\binom{d}{g_b - g_a}\binom{d + g_a - g_b}{g_a}}{\binom{d}{g_a}\binom{d}{g_b}}. \tag{91}$$

Simplifying this equation can be achieved by revealing either $\binom{d}{g_b}$ or $\binom{d}{g_a}$ in its upper term. This is merely performed through first applying the symmetry property of the binomial coefficient as follows.

$$\frac{p^{\text{th}}_{\cdot}}{p^{\text{ds}}_{\cdot}} = \frac{\binom{d}{d + g_a - g_b}\binom{d + g_a - g_b}{g_a}}{\binom{d}{g_a}\binom{d}{g_b}}. \tag{92}$$

Then, Equation (92) can be simplified using the trinomial property defined in [17]:

$$\frac{p^{\text{th}}_{\cdot}}{p^{\text{ds}}_{\cdot}} = \frac{\binom{d}{g_a}\binom{d - g_a}{d - g_b}}{\binom{d}{g_a}\binom{d}{g_b}}. \tag{93}$$

For any grade and any dimension, $\binom{d}{g_a} \neq 0$. We thus simplify Equation (93) as below.

$$\frac{p^{\text{th}}_{\cdot}}{p^{\text{ds}}_{\cdot}} = \frac{\binom{d-g_a}{d-g_b}}{\binom{d}{g_b}}. \tag{94}$$

Finally, the symmetry property of the binomial coefficient applied to the left term yields

$$\frac{p^{\text{th}}_{\cdot}}{p^{\text{ds}}_{\cdot}} = \frac{\binom{d-g_a}{g_b-g_a}}{\binom{d}{g_b}}. \tag{95}$$

As for the outer product, $\forall g_a \geq 0, \binom{d-g_a}{g_b-g_a} \leq \binom{d}{g_b}$.

If $g_a \geq g_b$, a similar reasoning results in:

$$\frac{p^{\text{th}}_{\cdot}}{p^{\text{ds}}_{\cdot}} = \frac{\binom{d-g_b}{g_a-g_b}}{\binom{d}{g_a}}, \tag{96}$$

and the same conclusion holds.

Appendix B. Proof of Proposition 5.2

Proof. In addition to the symmetry property and the trinomial property, we will use here the Vandermonde's convolution property of the binomial coefficient whose proof can be found in Chapter 5 of [17]. We first introduce a variable to drop divisions. Let us define

$$s = \frac{g_b - g_a + g_c}{2}. \tag{97}$$

Let us assume, without loss of generality, that $g_a > g_b$. Then, as $g_c \in \mathcal{I} = \{|g_a - g_b|, |g_a - g_b| + 2, \ldots, g_a + g_b\}$,

$$s \in \{0, 1, \cdots, g_b\}. \tag{98}$$

This yields

$$\begin{aligned} & \sum_{g_c \in \mathcal{I}} \binom{d}{g_c} \binom{g_c}{\frac{g_a - g_b + g_c}{2}} \binom{d-g_c}{\frac{g_a + g_b - g_c}{2}} \\ = & \sum_{s=0}^{g_b} \binom{d}{2s + g_a - g_b} \binom{2s + g_a - g_b}{s + g_a - g_b} \binom{d - 2s + g_b - g_a}{g_b - s}. \end{aligned} \tag{99}$$

We apply the trinomial revision property to the two leftmost terms in Equation (99), resulting in

$$\sum_{s=0}^{g_b} \binom{d}{s+g_a-g_b} \binom{d-s+g_b-g_a}{s} \binom{d-2s+g_b-g_a}{g_b-s}.$$

Next, we apply the same property to the two rightmost terms, yielding

$$\sum_{s=0}^{g_b} \binom{d}{s+g_a-g_b} \binom{d-s+g_b-g_a}{g_b} \binom{g_b}{s}. \tag{100}$$

The symmetry property is then applied to the leftmost term. We have

$$\sum_{s=0}^{g_b} \binom{d}{d-s+g_b-g_a} \binom{d-s+g_b-g_a}{g_b} \binom{g_b}{s}. \tag{101}$$

Again, we apply the trinomial revision property to the two leftmost terms in Equation (101). We now have

$$\binom{d}{g_b} \sum_{s=0}^{g_b} \binom{d-g_b}{d-s-g_a} \binom{g_b}{s}. \tag{102}$$

Note that $\binom{d}{g_b}$ does not depend on s. Applying the Vandermonde's convolution property to Equation (102) results in

$$\binom{d}{g_b} \binom{d}{d-g_a}. \tag{103}$$

After using the symmetry property on the right term, we see that Equation (64) holds. □

Appendix C. Proof of Lemma 6.4

Let us prove it by induction using the recursive formula (68). The base case is $g_c = 0$. The recursive formula (68) yields:

$$\begin{array}{c} \text{at depth } 0 \\ \text{computation: } \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma \wedge \mathfrak{b}_\delta, \quad \text{if } |\lambda| = g_c \\ \text{recursive calls: } \mathfrak{a}_\sigma \wedge \mathfrak{b}_0 + \overline{\mathfrak{a}_0} \wedge \mathfrak{b}_\sigma,\ \sigma \in [\max(\lambda)+1, \cdots, d] \end{array}. \tag{104}$$

We remark that each node of the resulting outer product prefix tree of grade 1 is 2. Then, all the siblings of grade 1 induce the same number of products. Let us assume that the proposition holds for a given grade of $\mathfrak{c}$, called $k_c \in \mathbb{N}$. Then the recursive products associated with any nodes $\mathfrak{c}_\lambda$ of grade k_c can be seen as the sum of products with the same number of terms. For any node, each single product can be written as

$$\mathfrak{c}_\lambda = \mathfrak{a}_\mu \wedge \mathfrak{b}_\nu. \tag{105}$$

This product expands at the grade of $k_c + 1$ is as follows.

$$\begin{aligned} &\text{at depth } k_c + 1 \\ &\text{computation: } \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma \wedge \mathfrak{b}_\delta, \quad \text{if } |\lambda| = g_c \\ &\text{recursive calls: } \mathfrak{c}_{\lambda+\sigma} = \mathfrak{a}_{\mu+\sigma} \wedge \mathfrak{b}_\nu + \overline{\mathfrak{a}_\mu} \wedge \mathfrak{b}_{\nu+\sigma},\ \sigma \in [\max(\lambda)+1, \cdots, d] \end{aligned} \tag{106}$$

Again, we remark that for any nodes of $\mathfrak{c}$ of grade $k_c + 1$, the number of products remains the same. Thus, by induction, the number of outer products remains the same for any node of the resulting prefix tree having the same grade (depth).

Appendix D. Proof of Theorem 6.10

Proof. This proof is split into three parts, each of which is dedicated to one term in Equation (90). As for the outer and inner products, the number of recursive calls remains the same for any nodes of grade g_c. Moreover, there are $\binom{d}{g_c}$ products for each node of grade g_c of the resulting multivector in the d-dimensional vector space. Let us denote by n_{g_a,g_b,g_c} the number of recursive calls with respect to grades g_a, g_b, and g_c. The overall complexity is then

$$c_*^{\text{rec}} = \mathcal{O}\left(\binom{d}{g_c} n_{g_a,g_b,g_c}\right). \tag{107}$$

Let us now employ the recursive formula of Equation (89). We remark that the recursive calls that increase the grade of the resulting multivector are those coming only from the outer product of Equation (68), corresponding to the last recursive call of Equation (89). As previously studied in Equation (69), for each possible grade of $\mathfrak{c}$, the number of calls associated with the recursive outer product is

$$\binom{g_c}{g_a} = \binom{g_c}{\frac{1}{2}(g_a - g_b + (g_a + g_b))} = \binom{d}{\frac{1}{2}(g_a - g_b + g_c)}. \tag{108}$$

Then, for any of the recursive outer product calls of the recursive geometric product, the recursive calls can be split into

$$\begin{aligned} &\text{at depth } n \\ &\text{computation: } \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma \times \mathfrak{b}_\delta, \quad \text{if } |\lambda| \in \mathcal{I}, |\gamma| = g_a \\ &\text{recursive calls 1: } \mathfrak{c}_\lambda = \textstyle\sum_{i=\sigma}^{d} \mathbf{m}(i)\overline{\mathfrak{a}_{\gamma+i}} * \mathfrak{b}_{\delta+i}\ \sigma \in [\max(\lambda)+1, \cdots, d] \\ &\text{recursive calls 2: } \mathfrak{c}_{\lambda+\sigma} = \mathfrak{a}_{\gamma+\sigma} * \mathfrak{b}_\delta,\ \sigma \in [\max(\lambda)+1, \cdots, d] \end{aligned} \tag{109}$$

and

$$\begin{aligned} &\text{at depth } n \\ &\text{computation: } \mathfrak{c}_\lambda \mathrel{+}= \mathfrak{a}_\gamma \times \mathfrak{b}_\delta, \quad \text{if } |\lambda| \in \mathcal{I}, |\gamma| = g_a \\ &\text{recursive calls 1: } \mathfrak{c}_\lambda = \textstyle\sum_{i=\sigma}^{d} \mathbf{m}(i)\overline{\mathfrak{a}_{\gamma+i}} * \mathfrak{b}_{\delta+i}\ \sigma \in [\max(\lambda)+1, \cdots, d] \\ &\text{recursive calls 2: } \mathfrak{c}_{\lambda+\sigma} = \overline{\mathfrak{a}_\gamma} * \mathfrak{b}_{\delta+\sigma},\ \sigma \in [\max(\lambda)+1, \cdots, d] \end{aligned} \tag{110}$$

We recognize the recursive right contraction of Equation (79) in Equation (109) whereas Equation (110) corresponds to the recursive left contraction of Equation (77). This indicates that for each recursive outer product call, recursive inner product calls are executed. Following the arguments of Theorem 6.8, we see that the number of required recursive calls is

$$n_{g_a,g_b,g_c} = \binom{g_c}{\frac{1}{2}(g_a - g_b + g_c)} \binom{d - g_c}{\frac{1}{2}(g_a + g_b - g_c)} \tag{111}$$

for any grade $g_c \in \mathcal{I}$. By merging the above arguments, we have Equation (90). □

Appendix E. Pseudo-codes of the recursive products

In the optimized pseudo-code, the indices of the basis blades are represented with a binary label. This binary label is useful to optimize paths in the prefix tree. The binary label of a node is recursively computed using the binary label of its parent node. A node with a binary label `u` has its first child binary label computed by,

$$\text{child\_label}(\texttt{u}, \texttt{msb}) = \texttt{u} + \texttt{msb}, \tag{112}$$

where $+$ is the binary addition and `msb` is the binary label of the basis vector "added" to the basis blade by the outer product. So, `msb` contains only a single bit set to 1. Note that this bit set to 1 in `msb` cannot be a bit already set to 1 in `u`, otherwise the parent node and its child would have the same grade.

The contribution of `msb` is the most significant bit of child_label(`label`, `msb`), i.e. the first bit to 1 encountered while reading the binary label from the left, which corresponds to the position of the 1-bit of `msb`.

We show the pseudo-code of the optimized outer product with the definition of these functions in Algorithm 5. In this algorithm, `labelToMsb(label)` computes `msb`, the most significant bit from the considered label, i.e., the first `1` encountered in the binary word `label` when reading from left to right.

We also give the pseudo-codes of the optimized left contraction, right contraction, and geometric product in Algorithms 6, 7, and 8, respectively. The functions called inside these pseudo-codes are the same as those in Algorithm 5.

Stephane Breuils
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000 Chambéry, France
e-mail: stephane.breuils@univ-smb.fr

Vincent Nozick
Laboratoire d'Informatique Gaspard-Monge, Equipe A3SI,
UMR 8049, Université Paris-Est Marne-la-Vallée, France
e-mail: vincent.nozick@u-pem.fr

Algorithm 5: Optimized recursive outer product $C = A \wedge B$

```
1  Function labelToMsb
       Input: label: binary word
2      return position of first 1 ∈ label when reading from left to right
3  Function gradeKReachable
       Input: label: the recursive position
              msb: a label of the last traversed vector
              k: the considered grade.
4      labelChildK ← label + msb(2^{k−grade(label)} − 1)
5      return labelChildK < 2^d
6  Function outer
       Input: A, B: two multivectors,
              C: resulting multivector,
              k_a, k_b and k_c: the respective grade of each multivector.
              label_a, label_b, label_c: recursive position on each tree.
              sign: recursive sign index.
              complement: recursive value (±1).
7      if grade(label_c) == k_c then                          //end of recursion
8          C[label_c] += sign × A[label_a] × B[label_b]
9      else                                                   //recursive calls
10         msb_a = labelToMsb(label_a)
11         msb_b = labelToMsb(label_b)
12         msb_c = labelToMsb(label_c)
13         foreach msb such that
               gradeKReachable(k_c, msb, label_c)== true do
14             label = label_c + msb
15             if gradeKReachable(k_a, msb, label_a) then
16                 outer(A, B, C, k_a, k_b, k_c, label_a +
                   msb, label_b, label, sign × complement, complement)
17             if gradeKReachable(k_b, msb, label_b) then
18                 outer(A, B, C, k_a, k_b, k_c, label_a, label_b +
                   msb, label, sign, −complement)
```

Akihiro Sugimoto
National Institute of Informatics, Tokyo 101-8430, Japan
e-mail: sugimoto@nii.ac.jp

Algorithm 6: Optimized recursive left contraction $C = A \rfloor B$

```
1  Function leftcont
       Input: A, B: two multivectors.
              C: resulting multivector.
              k_a, k_b and k_c: respective grade of each multivector.
              label_a, label_b, label_c: recursive position on each tree.
              sign: a recursive sign index.
2             complement: recursive value (±1).
3             m: vectors representing the metric diagonal matrix.
4      if grade(label_b) == k_b then                          //end of recursion
5          C[label_c] += m × sign × A[label_a] × B[label_b]
6      else                                                   // recursive calls
7          msb_a = labelToMsb(label_a)
8          msb_b = labelToMsb(label_b)
9          msb_c = labelToMsb(label_c)
10         foreach msb such that
            gradeKReachable(k_b, msb, label_b) == true do
11             label = label_b + msb
12             if gradeKReachable(k_a, msb, label_a) then
13                 leftcont(A, B, C, k_a, k_b, k_c, label_a + msb, label,
                    label_c, sign × complement, −complement,
                    metric × m(grade(label_b)))
14             if gradeKReachable(k_c, msb, label_c) then
15                 leftcont(A, B, C, k_a, k_b, k_c, label_a, label, label_c +
                    msb, sign, −complement, metric))
```

Algorithm 7: Optimized recursive right contraction $C = A \lfloor B$

```
1  Function rightcont
       Input: A, B: two multivectors.
              C: resulting multivector.
              k_a, k_b and k_c: respective grade of each multivector.
              label_a, label_b, label_c: recursive position on each tree.
              sign: recursive sign index.
2             complement: recursive value (±1).
3             metric: coefficients related to the metric.
4      if grade(label_b) == k_b then                          //end of recursion
5          C[label_c += metric × sign × A[label_a] × B[label_b]]
6      else                                                   // recursive calls
7          msb_a = labelToMsb(label_a)
8          msb_b = labelToMsb(label_b)
9          msb_c = labelToMsb(label_c)
10         foreach msb such that
            gradeKReachable(k_a, msb, label_a) == true do
11             label = label_a + msb
12             if gradeKReachable(k_b, msb, label_b) then
13                 rightcont(A, B, C, k_a, k_b, k_c, label, label_b + msb,
                    label_c, sign × complement, −complement,
                    metric × m(grade(label_b)))
14             if gradeKReachable(k_c, msb, label_c) then
15                 rightcont(A, B, C, k_a, k_b, k_c, label, label_b, label_c +
                    msb, sign, −complement, metric))
```

Algorithm 8: Optimized recursive geometric product $\mathbf{C} = \mathbf{A} * \mathbf{B}$

```
1  Function geometric
       Input: A, B: two multivectors.
              C: resulting multivector.
              k_a, k_b and k_c: respective grade of each multivector.
              label_a, label_b, label_c: recursive position on each tree.
              sign: a recursive sign index.
              complement: recursive value (±1).
              metric: coefficients related to the metric.
              depth: current depth in the prefix tree.
2      if grade(label_b) == k_b and grade(label_a) == k_a then
3          C[label_c] += metric × sign × A[label_a] × B[label_b]
                                                  // end of recursion
4      else
5          msb_a = labelToMsb(label_a)
6          msb_b = labelToMsb(label_b)
7          msb_c = labelToMsb(label_c)
8          for i in 2^depth, 2^(depth+1), ··· , 2^(d−1) do
9              if gradeKReachable(k_b, i, label_b) then
10                 if gradeKReachable(k_a, i, label_a) then
11                     geometric(A, B, C, k_a, k_b, k_c, label_a + i, label_b +
                         i, label_c, sign ×
                         complement, −complement, metric ×
                         m(i), depth + 1))
12             if gradeKReachable(k_a, i, label_a) then
13                 geometric(A, B, C, k_a, k_b, k_c, label, label_b, label_c +
                     msb, sign ×
                     complement, complement, metric, depth + 1))
14             if gradeKReachable(k_b, i, label_b) then
15                 geometric(A, B, C, k_a, k_b, k_c, label_a, label_b +
                     i, label_c + i, sign, −complement, metric), depth +
                     1)
```