Optimal Parenthesizing of Geometric Algebra Products

Stéphane Breuils, Vincent Nozick, Akihiro Sugimoto

Optimal Parenthesizing of Geometric Algebra Products

Stéphane Breuils[1][0000−0002−8636−4977], Vincent Nozick[2][0000−0001−8792−7164], and Akihiro Sugimoto[1][0000−0001−9148−9822]

[1] National Institute of Informatics, Tokyo, Japan {breuils,sugimoto}@nii.ac.jp
[2] Université Gustave Eiffel, LIGM, CNRS - ENPC - ESIEE Paris -UPEM, France
vincent.nozick@univ-eiffel.fr

Abstract. Manipulating objects using geometric algebra may involve several associative products in a single expression. For example, an object can be constructed by the outer product of multiple points. This number of products can be small for some conformal algebra and high for higher dimensional algebras such as quadric conformal geometric algebras. In these situations, the order of products (i.e. the choice of the parenthesis in the expression) should not change the final result but may change the overall computational cost, according to the grade of the intermediate multivectors. Indeed, the usual left to right way to evaluate the expression may not be most computationally efficient. Studies on the number of arithmetic operations of geometric algebra expressions have been limited to products of only two homogeneous multivectors. This paper shows that there exists an optimal order in the evaluation of an expression involving geometric and outer products, and presents a dynamic programming framework to find it.

1 Introduction

1.1 Geometric algebra products

Geometric 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 [9,10,5] for some examples. In contrast, in the computer science field, the study of computational aspects of the geometric algebra operators is still limited. The pioneering work [7] gave some results about complexity of geometric algebra products in the worst case. The worst case here means that all the elements of a multivector with multiple grades are non-zero. This study was then extended by [2] by investigating the number of arithmetic products required for the geometric product, the outer product, and the inner product of two full homogeneous multivectors, i.e., multivectors with non-zero components only for a single grade. The scope of [2] is well suited for expressions involving two full multivectors like intersections of two geometric objects in conformal geometric algebras. However, the representation of geometric objects in quadric conformal geometric algebras [3] includes more

than two multivectors in operands, and the arguments of [2] are not valid any more in such a case. The investigation on products of more than two multivectors is desired.

In this paper, we focus on products having the associative property. If the expression consists of only products that have the associative property, we only have to pay our attention to efficiently parenthesize operands in computation. The usual left to right way of computing the expression is not always the most computationally efficient. Moreover, we focus on full homogeneous multivectors since they are the representation of most geometric algebra objects. Note that in many applications, the geometric algebra entity representing any rigid transformations is also often represented as a sum of quasi full homogeneous multivectors.

1.2 Contributions

This paper focuses on products of more than two multivectors where all the products in the expression have the associative property. More specifically, this paper considers only products of more than two multivectors where the products are either outer products or geometric products. We then show that there exists an optimal parenthesizing order with respect to computational cost in the evaluation of an expression. Then, we describe a dynamic programming algorithm that yields the optimal product order. We remark that the inner product is not our interest in this paper because it is not associative.

2 Preliminaries

2.1 Notations

As commonly used in the state-of-the-arts ([6] and [11]), lower-case bold letters refer to vectors (vector $\mathbf{a}$) and lower-case non-bold letters to multivector coordinates (coefficient a_i). Multivectors and k-vectors are denoted with upper-case non-bold letters (multivector A). 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 dimension of the vector space, and in this case, the number of basis blades $\mathbf{e}_i$ of grade 1.

By assumption, a multivector A is not necessarily homogeneous but is defined as the sum of homogeneous multivectors. Given the dimension d of the vector space, the set of possible grades for A is $\mathcal{K}_A$ where $\mathcal{K}_A \subseteq \{0, 1, \cdots, d\}$. Then, A can be defined as

$$A = \sum_{k \in \mathcal{K}_A} \langle A \rangle_k. \tag{1}$$

2.2 Product of two multivectors

Following the notation of Equation (1), the product $\odot$ of two multivectors A and B is then

$$A \odot B = \Big(\sum_{k_A \in \mathcal{K}_A} \langle A \rangle_{k_A} \Big) \odot \Big(\sum_{k_B \in \mathcal{K}_B} \langle B \rangle_{k_B} \Big), \tag{2}$$

where $\odot$ can be either the outer product or the geometric product. The linearity of the products of geometric algebra yields

$$A \odot B = \sum_{k_A \in \mathcal{K}_A} \sum_{k_B \in \mathcal{K}_B} \langle A \rangle_{k_A} \odot \langle B \rangle_{k_B}. \tag{3}$$

Theorems 2.1 and 4.1 of [2] give the optimal number of arithmetic operations of the product between two homogeneous multivectors of respective grades k_a and k_b. As for the outer product $\wedge$, this number of arithmetic operations is:

$$p^{\wedge}_{k_a,k_b} = 2\binom{d}{k_a + k_b}\binom{k_a + k_b}{k_a}, \tag{4}$$

where $\binom{n}{k}$ is the binomial coefficient. On the other hand, the number of arithmetic operations for the geometric product $*$ is:

$$p^{*}_{k_a,k_b} = 2 \sum_{k_c \in \mathcal{I}} \binom{d}{k_c}\binom{k_c}{\frac{k_a - k_b + k_c}{2}}\binom{d - k_c}{\frac{k_a + k_b - k_c}{2}}, \tag{5}$$

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

In the more general situation where the two multivectors may not be homogeneous, the product distributivity mentioned in Equation (3) leads to a double loop over the respective grades of the two multivectors. Thus, the number of required operations is the sum of all per-grade contributions, as described in Algorithm 1.

Algorithm 1: Number of arithmetic operations required for outer product and geometric product of two general multivectors

1 **Function** $\mathtt{P}^{\wedge}(\mathcal{K}_A, \mathcal{K}_B)$
  Input: $\mathcal{K}_A$: set of grades for the multivector A
  $\mathcal{K}_B$: set of grades for the multivector B
  Output: total number of arithmetic operations resulting from $A \wedge B$
2 **return** $2 \sum_{k_A \in \mathcal{K}_A} \sum_{k_B \in \mathcal{K}_B} \binom{d}{k_a + k_b}\binom{k_a + k_b}{k_a}$

3 **Function** $\mathtt{P}^{*}(\mathcal{K}_A, \mathcal{K}_B)$
  Input: $\mathcal{K}_A$: set of grades for the multivector A
  $\mathcal{K}_B$: set of grades for the multivector B
  Output: total number of arithmetic operations resulting from $A * B$
4 **return** $2 \sum_{k_A \in \mathcal{K}_A} \sum_{k_B \in \mathcal{K}_B} \sum_{k_c \in \mathcal{I}} \binom{d}{k_c}\binom{k_c}{\frac{k_a - k_b + k_c}{2}}\binom{d - k_c}{\frac{k_a + k_b - k_c}{2}}$

The resulting multivector usually has a different grade from the two operands used for its computation. For the outer product of homogeneous multivectors

$C = A \wedge B$, the grade of C will just be the sum of the grades of A and B. For general multivectors, we again have to follow Equation 3 to compute the set of resulting grades. The geometric product is a bit more complex and can generate a non-homogeneous multivector even from two homogeneous multivectors. The resulting grades of a product between two general multivectors is summarised in Algorithm 2. For more details about these results, the reader can refer to [2]. Note that $\mathtt{K}^{\wedge}$ or $\mathtt{K}^{*}$ can return an empty set of grades (e.g. the wedge of $d+1$ vectors in a d-dimensional vector space), then the computational cost is 0 since the resulting product is also 0.

Algorithm 2: Computation of the set of grades resulting to the product of two multivectors.

```
1  Function K^(K_A, K_B)
       Input: K_A: set of grades for the multivector A
              K_B: set of grades for the multivector B
       Output: set of grades K_C of the result of A ∧ B
2      K_C = ∅
3      foreach k_a ∈ K_A do
4          foreach k_b ∈ K_B do
5              if k_a + k_b ≤ d then
6                  K_C = K_C ∪ (k_a + k_b)
7  Function K*(K_A, K_B)
       Input: K_A: set of grades for the multivector A
              K_B: set of grades for the multivector B
       Output: set of grades K_C of the result of A * B
8      K_C = ∅
9      foreach k_a ∈ K_A do
10         foreach k_b ∈ K_B do
11             foreach k_c ∈ {|k_a − k_b|, |k_a − k_b| + 2, ··· , k_a + k_b} do
12                 if k_c ≤ d then
13                     K_C = K_C ∪ k_c
```

3 Optimal parenthesising of products

We show that the choice of the order of the product in an expression can affect the complexity of the product. Consider, for example, the following expression $A_1 \wedge A_2 \wedge A_3$ in a 10-dimensional space, where the grade of each multivector is

$$\text{grade}(A_1) = 4, \quad \text{grade}(A_2) = 3, \quad \text{grade}(A_3) = 2.$$

There are in that case two possible parenthesings, the left to right way as

$$(A_1 \wedge A_2) \wedge A_3, \tag{6}$$

and the right to left way:

$$A_1 \wedge (A_2 \wedge A_3). \tag{7}$$

In the first case, the product $A_1 \wedge A_2$ of Equation (6) generates a multivector of grade 7, that is finally wedged to A_3. According to Equation (4), the number of operations for $A_1 \wedge A_2$ is 8400 and then 720 for the second outer product, leading to a total of 9120 arithmetic operations. On the other hand, the first product $A_2 \wedge A_3$ of Equation (7) generates a multivector of grade 5, that is wedged to A_1. The overall computational cost of this product is 5040 for the first product and 2520 for the second, resulting in 7560 arithmetic operations in total. Thus, in this case, choosing the second way of parenthesising brings a 1.5 times gain in terms of numerical operations. Obviously, the gain can be much higher for longer expressions.

3.1 Expressions

Throughout this paper, we consider an expression as the products of n multivectors:

$$A_1 * A_2 * \cdots * A_n, \qquad n \in \mathbb{N} \tag{8}$$

or

$$A_1 \wedge A_2 \wedge \cdots \wedge A_n, \qquad n \in \mathbb{N}. \tag{9}$$

Since the outer product and the geometric product are binary operators, i.e. operators between two operands, this overall computation involves the computation of intermediate results. Let $A_{1,n}$ be the chain of multivectors defined by the expression to compute, and $A_{i,j} (0 < i < j \leq n)$ a sub-chain resulting from the computation of the product from the multivector A_i to the multivector A_j. The number of arithmetic operations resulting from the product of two successive multivector chains $A_{i,j}$ and $A_{j+1,k}$ is denoted by $\mathrm{P}^{\odot}_{i,j,k}$. where $\odot$ can be either $\wedge$ or $*$.

3.2 Problem formulation

Let $C_{i,j}$ be the minimum number of arithmetic operations of the computation (the cost to minimise) of the product between the i^{th} multivector up to the j^{th} multivector. In the final result, we seek for the computation of C_{1n} related to the full expression. For $i < j$, this is equivalent to seek for

$$C_{i,j} = \min_{s \in [i,j-1]} C_{i,s} + C_{s+1,j} + \mathrm{P}^{\odot}_{i,s,j}. \tag{10}$$

This optimal cost $C_{i,j}$ will be used to define the optimal parenthesising indices 2D table S of size $n \times n$, where $S_{i,j} = s$ $(i < j)$ means that the expression $A_i \odot A_{i+1} \odot \cdots \odot A_j$ should be parenthesised as

$$(A_i \odot \cdots \odot A_s) \odot (A_{s+1} \odot \cdots \odot A_j). \tag{11}$$

3.3 Minimisation

An easy way to achieve this minimisation is to use a recursive method. However, such an approach will lead to multiple travels over the same recursive sub-trees, resulting in an exponential complexity.

This kind of problems was already addressed for the matrix chain product, for example in [1], by using dynamic programming. The problem is the computation of the same sub-problems for different depth of recursion. The proposed approach to solve it consists in memorising the solution to sub-problems in a bottom-up scheme. Each sub-problem is uniquely identified by the two bounding indices (i, j), in the computation of $C_{i,j}$, $S_{i,j}$ and $\mathcal{K}_{i,j}$ as $2D$ tables. These tables can be iteratively filled with a bottom up approach.

We achieve our minimisation using the dynamic programming framework. Our proposed method first considers the sub-chains of length one. They correspond to single multivectors and thus take a cost $C_{i,i} = 0$, meaning no optimal parenthesising index. The sub-chains of length two can also be directly computed from Algorithms 1 and 2, where $C_{i,i+1} = \mathrm{P}^{\odot}_{i,i,i+1} = \mathrm{P}^{\odot}(\mathcal{K}_{A_i}, \mathcal{K}_{A_{i+1}})$. Obviously, $S_{i,i+1} = i$ and $\mathcal{K}_{i,i+1} = \mathrm{K}^{\odot}(\mathcal{K}_{A_i}, \mathcal{K}_{A_{i+1}})$. Then, the sub-chains of length 3 can be computed from the sub-chains of length 2 using Equation (10), and so on. The resulting algorithm that computes the optimal parenthesising for each sub-expressions is shown in Algorithm 3.

Once 2D table S is computed, the product can be optimally computed by a recursive travel starting from $S_{1,n}$, as shown in Algorithm 4. In case of a code optimisation, a similar recursive scheme can be adopted to add the optimal parenthesising on the code.

We remark that the complexity of this algorithm is not exponential but polynomial in $\mathcal{O}(n^3)$, see [1]. Note that in the matrix chain product context, there exist some faster methods in $\mathcal{O}(n \times \log n)$, like [8] however, these methods are complex to setup. Moreover, in geometric algebra, n is usually not high enough to see a significant difference.

4 Discussion and applications

It is not difficult to modify Algorithm 3 so that it computes the worst parenthesis instead of the best, which in turn results in the maximum number of arithmetic operations. We merely have to replace the sign '$<$' by '$>$' in line 19 of Algorithm 3 and to change the initialisation of $C_{i,j}$ to 0 in line 16.

From these two algorithms, we can also extract the gain of the optimal parenthesis. The results we have, show that the maximum gain increases as n (the number of multivectors) increases. Moreover, the gain also increases as the dimension increases. For a fixed number of multivectors in the expression, the gain becomes highest when the sum of the grades of the operands is near the dimension for expressions where the outer product appears.

Furthermore, a practical usage of these kinds of algorithms is in code generators and optimizers. For example, the code developed from the expression

Algorithm 3: Computation of the optimal parenthesising of a geometric algebra associative product.

```
1  Function OptimalParenthesising
       Input: A_1, ..., A_n: chain of general multivectors.
              ⊙: the considered associative product.
       Output: S_{i,j}: 2D optimal parenthesising index table.
       // define a 2D cost table
2      C ← 2D table
       // Sub-chain of length 1
3      for i ∈ [1, n] do
4          C_{i,i} = 0
5          K_{i,i} = grades(A_i)
       // Sub-chain of length 2
6      for i ∈ [1, n − 1] do
7          C_{i,i+1} = P^⊙(grades(A_i), grades(A_{i+1}))        // from Algo 1
8          K_{i,i+1} = K^⊙(grades(A_i), grades(A_{i+1}))        // from Algo 2
9          S_{i,i+1} = i
       // Grade of all sub-chains of length u > 2, starting at index i
10     for u ∈ [3, n] do
11         for i ∈ [1, n − u + 1] do
12             K_{i,i+u} = K^⊙(K_{i,i}, K_{i+1,i+u})  // grade from any arbitrary cut
       // For all possible length u > 2 of sub-chains
13     for u ∈ [3, n] do
           // For all sub-chains of length u, starting at i
14         for i ∈ [1, n − u] do
15             j = i + u                                  // sub-chain from i to j
16             C_{i,j} = ∞
               // For all possible cut of the sub-chain
17             for s ∈ [i, j − 1] do
                   // compute the cost c of this cut with Eq.(10)
18                 c = C_{i,s} + C_{s+1,j} + P^⊙(K_{i,s}, K_{s+1,j})
                   // if the cost is better than before, update
19                 if c < C_{i,i+u} then
20                     C_{i,j} = c
21                     S_{i,j} = s
22     return S
23 First call: S = OptimalParenthesising( A_1, ..., A_n, ⊙)
```

of Equation (6) in the code generator Gaalop [4] results in more arithmetic operations than in the code generated from Equation (7) arithmetic operations. Note that we consider full multivectors for the three multivectors that appear in the expression.

Algorithm 4: Evaluation of the product $A_1 \odot A_2 \odot \cdots \odot A_n$ using the optimal parenthesising.

```
1 Function ComputeProduct
    Input: A_1, ··· , A_n: chain of general multivectors,
           S: optimal 2D parenthesising index table
           i, j: resp. indices of the start and end of the sub-expression
           ⊙: the considered associative product
    Output: result of the product.
2   if i=j then
3     return A_i
4   s = S_{i,j}
    // left side recursive call
5   A = ComputeProduct(A_1, ··· , A_n, S, i, s, ⊙)
    // right side recursive call
6   B = ComputeProduct(A_1, ··· , A_n, S, s+1, j, ⊙)
    // do the product
7   return A ⊙ B
8 First call: Result = ComputeProduct(A_1, ··· , A_n, S, 1, n, ⊙)
```

5 Conclusion

In this paper, we focused on products of more than two multivectors and addressed that there exists an optimal order in the evaluation of an expression. Then, we gave a dynamic programming algorithm that yields the optimal product order in polynomial time. The benefits of this approach arise from the product of three multivectors.

References

1. Aho, A.V., Hopcroft, J.E.: The design and analysis of computer algorithms. Pearson Education India (1974)
2. Breuils, S., Nozick, V., Sugimoto, A.: Computational aspects of geometric algebra products of two homogeneous multivectors. ArXiv **abs/2002.11313** (2020)
3. Breuils, S., Nozick, V., Sugimoto, A., Hitzer, E.: Quadric conformal geometric algebra of $\mathbb{R}^{9,6}$. Advances in Applied Clifford Algebras **28**(2), 35 (Mar 2018). https://doi.org/10.1007/s00006-018-0851-1, `https://doi.org/10.1007/s00006-018-0851-1`
4. Charrier, P., Klimek, M., Steinmetz, C., Hildenbrand, D.: Geometric algebra enhanced precompiler for C++, OpenCL and Mathematica's OpenCLLink. Advances in Applied Clifford Algebras **24**(2), 613–630 (2014)
5. De Keninck, S., Dorst, L.: Geometric algebra levenberg-marquardt. In: Gavrilova, M., Chang, J., Thalmann, N.M., Hitzer, E., Ishikawa, H. (eds.) Advances in Computer Graphics. pp. 511–522. Springer International Publishing, Cham (2019)
6. Dorst, L., Fontijne, D., Mann, S.: Geometric Algebra for Computer Science, An Object-Oriented Approach to Geometry. Morgan Kaufmann (2007)

7. Fontijne, D.: Efficient Implementation of Geometric Algebra. Ph.D. thesis, University of Amsterdam (2007)
8. Hu, T., Shing, M.: Computation of matrix chain products. part i. SIAM Journal on Computing **11**(2), 362–373 (1982)
9. Kanatani, K.: Understanding Geometric Algebra: Hamilton, Grassmann, and Clifford for Computer Vision and Graphics. A. K. Peters, Ltd., Natick, MA, USA (2015)
10. Lasenby, J., Hadfield, H., Lasenby, A.: Calculating the rotor between conformal objects. Advances in Applied Clifford Algebras **29**(5), 102 (Oct 2019). https://doi.org/10.1007/s00006-019-1014-8, `https://doi.org/10.1007/s00006-019-1014-8`
11. Perwass, C.: Geometric algebra with applications in engineering, Geometry and Computing, vol. 4. Springer (2009)