A hybrid approach for computing products of high-dimensional geometric algebras

Stéphane Breuils, Vincent Nozick, Laurent Fuchs, Dietmar Hildenbrand, Werner Benger, Christian Steinmetz

A hybrid approach for computing products of high-dimensional geometric algebras

Stéphane Breuils
LIGM (UMR 8049),
F-77454, Marne-la-Vallée, France
stephane.breuils@u-pem.fr

Vincent Nozick
JFLI (UMI 3527), CNRS, NII
Tokyo, Japan
vincent.nozick@u-pem.fr

Laurent Fuchs
XLIM-ASALI (UMR 7252), CNRS,
Université de Poitiers, Poitiers, France
Laurent.Fuchs@univ-poitiers.fr

Dietmar Hildenbrand
Hochschule RheinMain
Rüsselsheim, Germany
dietmar.hildenbrand@gmail.com

Werner Benger
AHM Software GmbH / Louisiana
State University, Center for
Computation & Technology
Innsbruck, Austria /
Baton Rouge, USA
w.benger@ahm.co.at

Christian Steinmetz
TU Darmstadt
Darmstadt, Germany
steinmetz@dik.tu-darmstadt.de

ABSTRACT

Geometric Algebra is considered as a very intuitive tool to deal with geometric problems and it appears to be increasingly efficient and useful to deal with computer graphics solutions. For example, the Conformal Geometric Algebra includes circles, spheres, planes and lines as algebraic objects, and intersections between these objects are also algebraic objects. More complex objects such as conics, quadric surfaces can also be expressed and be manipulated using an extension of the conformal Geometric Algebra. However due to high dimension of their representations in Geometric Algebra, implementations of Geometric Algebra that are currently available do not allow efficient realizations of these objects. This paper presents a Geometric Algebra implementation dedicated for both low and high dimensions. The proposed method is a hybrid solution for precomputed code with fast execution and runtime computations with low memory requirement. More specifically, the proposed method combines a precomputed table approach with a recursive method using binary trees. Some rules are defined to select the most appropriate choice, according to the dimension of the algebra and the type of multivectors involved in the product. The resulting implementation is well suited for high dimensional spaces (e.g. algebra of dimension 15) as well as for lower dimensional space. This paper details the integration of this hybrid method as a plug-in into Gaalop, which is a very advanced optimizing code generator. This paper also presents some benchmarks to show the performances of our method, especially in high dimensional spaces.

CGI '17, Yokohama, Japan

978-1-4503-5228-4/17/06…$15.00
DOI: 10.1145/3095140.3097284

CCS CONCEPTS

•**Mathematics of computing** →**Mathematical software performance;** •**Theory of computation** →*Computational geometry;* •**Computing methodologies** →*Representation of mathematical functions;*

KEYWORDS

Geometric Algebra, Implementation, High dimensional space

ACM Reference format:
Stéphane Breuils, Vincent Nozick, Laurent Fuchs, Dietmar Hildenbrand, Werner Benger, and Christian Steinmetz. 2017. A hybrid approach for computing products of high-dimensional geometric algebras. In *Proceedings of CGI '17, Yokohama, Japan, June 27-30, 2017,* 6 pages.
DOI: 10.1145/3095140.3097284

1 INTRODUCTION

Geometric algebras can be understood as a set of very intuitive tools to represent, construct and manipulate geometric objects. Some works [7, 13, 14, 16] proved that Geometric Algebra can be used very advantageously in computer graphics and computer vision. Other works [9] and [8] show that geometric algebra can be used to express and transform more complex objects such as conics and quadrics.

There already exists numerous geometric algebra implementations, and most of the programming languages or famous mathematical frameworks can find a geometric algebra library well suited for a comfortable use. However, very few of these libraries can handle computations in high dimensional spaces, i.e. more than 12. Moreover, the huge size of the subspace generated by high dimensional spaces makes these few libraries very slow to process. This paper presents a new method based on binary trees and tables that efficiently computes Geometric Algebra for both low and higher dimensional spaces. The key feature of the proposed method is to assign the computation to either the binary tree or the table according to the dimension of the multivectors involved in the product.

The paper is organized as follows: Section 2 presents a state of the art of the main geometric algebra implementations. Section 3

discusses the implementation constraints related to high dimensional spaces. Section 4 describes the proposed hybrid method. Finally, Section 5 details the integration of the hybrid approach as a plug-in into Gaalop, a very advanced geometric algebra tool.

2 STATE OF THE ART

There exist various implementations of geometric algebra, each of them dedicated for a specific use. Some of these implementations are designed to fit to a specific framework as a plug-in. This is the case for the library CLIFFORD [2] developed by Rafal Ablamowicz and Bertfried Fauser, based on Maple. More recently, Steve Sangwine and Eckhard Hitzer developed a Multivector Toolbox [19] for Matlab. Maxima also finds its Clifford Algebra implementation [18] presented by Dimiter Prodanov. Some implementations of geometric algebra target a specific programming language like GAlgebra [4] for Python or GALua [15] for Lua. Versor [6] developed by Pablo Colapinto is a C++ library that focuses on 5D conformal geometry. CLUCalc [17], conceived and written by Christian Perwass handles more general geometric algebras. Finally, Gaalet [20] proposed by Florian Seybold, standing for Geometric Algebra ALgorithms Expression Template, is also a C++ library and uses expression templates and metaprogramming techniques.

All these libraries present a comfortable use in their dedicated framework or programming language and are very well suited for experimentations in geometric algebra. Some other libraries expressly focus on computation performance. The main way to speed up the process is to precompute a selection of geometric algebra operations. The level of optimization can range from very high but quite static to less optimized with more flexibility.

The most optimized libraries perform most of the computation during the compilation stage from a simple high level language. The optimizations are obtained by a low level source code generation as well as with some symbolic optimizations in terms of algebra. The most advanced program in this category is certainly Gaalop [5, 13] developed by Dietmar Hildenbrand et al. Gaalop, standing for Geometric Algebra Algorithms Optimizer, uses symbolic operations written in CLUCalc to produce C++, C++ AMP (Accelerated Massive Parallelism), OpenCL, CUDA, CLUCalc or LaTeX optimized output. GMac [10] developed by Ahmad Hosny Eid is also a very optimized library for geometric algebra. GMac stands for Geometric MACro, and is closely coupled with .NET languages like C++, C#, VB.NET, F#, and IronPython. GMac presents advanced conception and implementation, but it is not open source. These compilation optimizations produce a very optimized code dedicated for a specific task, however they are not well suited for runtime computation where the operations are not decided in advance.

A more flexible way to use geometric algebra consists in using libraries where the product between some multivectors with defined grade or structure are optimized. Typically, products between homogeneous multivectors (non-zero vector basis with same grade) can be hardcoded. This is the case for Gaigen [7, 11] presented by Daniel Fontijne. Gaigen stands for Geometric Algebra Implementation GENerator and can produce C++, C, C# and Java source code which implements a geometric algebra with a specified dimension and metric. The geometric algebra products are computed at run time using the hard coded functions. Exotic multivectors (part of non-homogeneous multivectors) are computed with a general class that presents a lower optimization level. Breuils et al. [3] introduce a similar library that produces hard coded products in C++ code with SIMD instructions, leading to higher performances.

3 HIGH DIMENSIONS

Geometric algebra in a large sense includes many algebras among which Conformal Geometric Algebra (CGA) is certainly one of the most studied [16]. CGA is built from 5 dimensional vector space, and thus includes $2^5 = 32$ basis vectors. Although some people consider 32 components per multivector to be already high, some studies are conducted to explore higher dimensions. Easter and Hitzer [9] represent some quartics and quadrics 3-d shapes using a double conformal geometry of $\mathbb{R}^3$. Extending this process to a triple conformal geometry would lead to a 15 dimensional algebra containing $2^{15} = 32,768$ elements. For such geometry, the memory requirement for optimized libraries explodes far beyond consumer grade hardware capabilities. More regular approaches lead to very long processing time. This section gives some details about this memory and processing time issues, as well as an efficient solution for this problem.

3.1 Geometric algebra products

The geometric product, the inner product and the outer product between 2 multivectors $\mathbf{a}$ and $\mathbf{b}$ are distributive over the addition and can conceptually be computed by iterating over the components of both $\mathbf{a}$ and $\mathbf{b}$. For conciseness purpose, we limit our description to the outer product but the overall method remains true for the geometric product and the inner product. The outer product $\mathbf{c} = \mathbf{a} \wedge \mathbf{b}$ can be expressed as:

$$\mathbf{c} = \sum_{k=0}^{2^d-1} c_k \mathbf{E}_k = \left(\sum_{i=0}^{2^d-1} a_i \mathbf{E}_i \right) \wedge \left(\sum_{j=0}^{2^d-1} b_j \mathbf{E}_j \right) \quad (1)$$

where $\mathbf{E}_i$ refers to a basis vector, i.e $\mathbf{E}_i \in \{\mathbf{1}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_{12}, \ldots\}$. Each pair $\{a_i, b_j\}$ leads to a computed element that contributes to the final value of $\mathbf{c}$, and should be assigned to the adequate basis vector of $\mathbf{c}$, namely $\mathbf{E}_k = \pm\, \mathbf{E}_i \wedge \mathbf{E}_j$. This assignment can be decided either by a precomputed table that requires some memory space or by a function that requires some additional processing time.

3.2 Table based methods

An efficient way to compute the geometric algebra products is to use precomputed 2D-tables indicating the result of the products between any basis blades. This approach can be used by Gaalop. A product between two multivectors $\mathbf{a}$ and $\mathbf{b}$ consists of a loop over each basis blades present in the two multivectors. For each couple $\{a_i, b_j\}$, the program refers to the table to know where to put the result. Although this approach is simple and effective, it includes some useless operations. Indeed, the outer product between dependent basis blades (e.g. $\mathbf{e}_1 \wedge \mathbf{e}_{12}$) requires a table access and are discarded if the table says so. The pseudo-code of this approach is presented on Algorithm 1. In the worst case, each basis blade product is a constant time algorithm, however, this

Algorithm 1: Product using the table based method.

```
   Input: multivectors: A and B,
          Table: T and S,
          Dimension of the algebra: d
   Output: resulting multivector: C
1  for i from 0 to 2^d − 1 do
2      if A[i] non-zero blade then
3          for j from 0 to 2^d − 1 do
4              if B[j] non-zero blade then
5                  index ← T[i, j]
6                  if index ≠ null then        // i.e. a e_i ∧ b e_j ≠ 0
7                      sign ← S[i, j]
8                      C[index] += sign × A[i] × B[j]
```

product can be optimized by storing the multivector components in a list, which may speedup the loops of lines 1 and 3. The main drawback of this approach is the memory consumption of the table. A table should include all the basis vectors of the algebra in the table, leading to a $2^d \times 2^d = 4^d$ size table, where d is the dimension of the algebra. Each element of the table is composed of an index on where to put the result and a sign (sometimes, the result is negative). Moreover, geometric algebra involves at least 3 products (geometric, outer and inner products), leading to at least 3 tables. If we assume that the sign and index are stored in a signed integer of 6 bits, then the size of the tables would be $2^5 \times 2^5 \times 6 \times 3 \approx 18$ kbits. However this would be a lower bound. Indeed, a product between two basis basis blades might result in a sum of basis blades, as encountered in Conformal Geometric Algebra. Thus, in practice, the tables contain a structure element that may enclose more than one index and sign. In order to estimate the precise memory requirement of the table based methods, we ran a profiler on Gaalop. In a 5-dimensional space, Gaalop requires 1.1 MB of memory and requires 710 MB for dimension 10. According to the progression of a $O(4^d)$ memory complexity, we can infer that for a dimension 11, the table size will occupy more than 2 GB memory and for a dimension 15, it would be around 360 GB. This memory requirement becomes a critical issue for high dimensional spaces.

3.3 Function based methods

3.3.1 Iterative methods.

During the product between **a** and **b**, the access to a table can be replaced by a call of a function that specifies where to put the result of a product between an element of **a** and an element of **b**. In practice, this approach just changes lines 5 and lines 7 of Algorithm 1. Gaalop as well as Gaigen propose this kind of functions, based on logical operations on a binary representation of the basis vectors. Another function, also based on binary operators, is required to specify the sign of a product contribution. Breuils et al. introduce a faster sign computation function [3] derived from [12].

The library CLIFFORD [2] uses linear algebra to decompose and compute each product. This decomposition makes possible to handle high dimensional space computations. Moreover, the authors extend this program to a parallel version in [1].

3.3.2 Recursive methods.

Fuchs and Théry [12] present a method to recursively compute the products and [3] gives some implementation elements. Each product is represented by a recursive function scanning a binary tree whose leaves define the basis vectors of the considered algebra. The multivectors product thus takes the following recursive form:

$$\mathfrak{a} \wedge \mathfrak{b} = \left(\mathfrak{a}_1, \mathfrak{a}_0\right) \wedge \left(\mathfrak{b}_1, \mathfrak{b}_0\right) \tag{2}$$

where the subscript $\mathfrak{a}_0$ or $\mathfrak{a}_1$ respectively refers to the right and left subtree of $\mathfrak{a}$. The recursive outer product starts with the iteration $n = 0$ and is defined as:

$$\begin{aligned} &\text{if } n < d,\ \mathfrak{a}^n \wedge \mathfrak{b}^n = \left(\mathfrak{a}_1^{n+1} \wedge \mathfrak{b}_0^{n+1} + \overline{\mathfrak{a}_0}^{n+1} \wedge \mathfrak{b}_1^{n+1}, \mathfrak{a}_0^{n+1} \wedge \mathfrak{b}_0^{n+1}\right)^n \\ &\text{if } n = d,\ \mathfrak{a}^n \wedge \mathfrak{b}^n = \mathfrak{a}^d \wedge \mathfrak{b}^d \end{aligned} \tag{3}$$

where $\overline{\mathfrak{a}}$ expresses the anticommutativity of the outer product. As showed in [3], this method avoids useless products like outer product between dependent basis blade, resulting in a complexity of $O(3^d)$ instead of $O(4^d)$. This complexity enhancement has a huge effect in high dimensional spaces.

To handle sparse multivectors, Fuchs and Théry propose to use unbalanced binary trees where the subtrees leading to leaves with coefficient set to 0 are removed. A recursive call leading to nodes without subtrees is immediately discarded, as presented in Algorithm 2. Note that for clarity purpose, Algorithm 2 omits sign computations.

Algorithm 2: Recursive outer product (unbalanced binary tree)

```
1  Function wedge
       Input: C: resulting binary tree, A, B: binary tree
2      if A is a leaf then
3          C += A × B            // (× is the product of scalars)
4      else
5          if A.hasLeftChild() and B.hasRightChild() then
6              wedge(C.leftChild,A.leftChild,B.rightChild)
7          if A.hasRightChild() and B.hasLeftChild() then
8              wedge(C.leftChild,A.rightChild,B.leftChild)
9          if A.hasRightChild() and B.hasRightChild() then
10             wedge(C.rightChild,A.rightChild,B.rightChild)
```

4 HYBRID APPROACH

We propose a new hybrid approach combining the table based approach of Gaalop [5] with the recursive approach [3, 12].

4.1 Recursive method revisited

This section focuses on the recursive method and more specifically on how to minimize the number of conditionals in the recursive products. Indeed, a recursive call from a recursion level to the next involves some tests on the existence of subtrees, i.e. line 5, 7 and 9

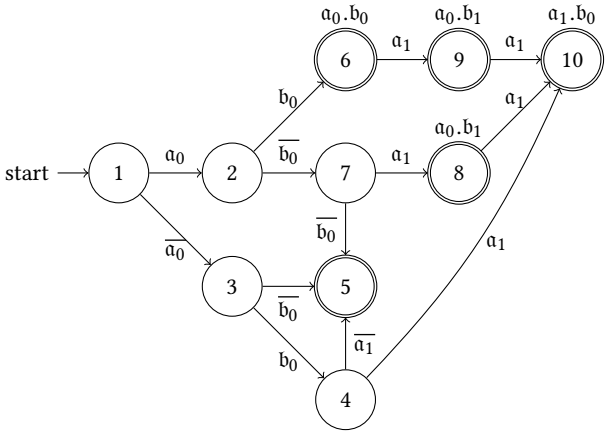

Figure 1: Automaton of the set of products of the outer product from an recursion level to the next.

of Algorithm 2 for the outer product. We express this set of conditions as a deterministic finite automaton with boolean transitions according to the existence of left subtree $\mathfrak{a}_1$ and right subtree $\mathfrak{a}_2$ of the current node $\mathfrak{a}$ (respectively for $\mathfrak{b}$). As an example, the automaton of the outer product is depicted in Figure 1. To each final state of the automaton is associated a subset of the set of products of equation (3), except for state 5 where no product is required. We remark that in the worst case four conditionals are required but less in the average case. For example, if we consider that $\mathfrak{a}_2$ and $\mathfrak{b}_2$ are non-null binary tree whereas $\mathfrak{a}_1$ and $\mathfrak{b}_1$ are null binary subtrees, then the control flow will be formed of two conditional instead of four.

4.2 Table and recursive method hybridization

The table method developed in this paper is a restructured version of the table based approach described in Section 3.2. Instead of looping over two multivectors, the proposed algorithm uses the recursive algorithm presented in section 4.1 and switch to the table based method when the recursive call reaches the leaves level of the trees. The pseudo-code of our method is presented in Algorithm 3 for the outer product, where the function `outerRecursiveFlow`$(\mathfrak{a}_0, \mathfrak{a}_1, \mathfrak{b}_0, \mathfrak{b}_1)$ recursively calls the function `outerHybrid` with the automaton process of section 4.1.

The Algorithm 3 enables the two following properties:

- to compute the per-blade product in constant time.
- to avoid useless products, for example, outer product between dependent basis blade.

However, this method still requires the storage of a table whose size is 4^d, where d is the dimension.

4.3 Dimension restrictions

To fix the issue raised above, we limit the table size to a decided threshold. Thus, the table will be used only for the computation between elements that are represented in the table. The remaining computation will be performed by the fully recursive process.

Algorithm 3: Hybrid recursive outer product

```
1  Function outerHybrid
       Input: binary tree: A and B
              Table: T and S
       Output: resulting binary tree: C
2      if A is a leaf then
3          i ← A.index
4          j ← B.index
5          index ← T[i, j]
6          sign ← S[i, j]
7          C[index] += sign × A × B        // (product of scalars)
8      else
9          a1 ← A.leftChild()
10         a0 ← A.rightChild()
11         b1 ← B.leftChild()
12         b0 ← B.rightChild()
13         outerRecursiveFlow(a1, a0, b1, b0)
```

Dimension	5	6	7	8	9	10
Memory occupation (MB)	1.1	3.7	13	40	142	710

Table 1: Gaalop memory occupation, including tables.

The threshold dimension is decided such that the memory used for the table does not exceed a certain limit. To know the memory requirement of the tables in real conditions, we use the results of our profiling tests conducted on Gaalop. The generated tables concern outer, geometric and inner products between the general multivectors. We tested the memory occupation of the leading program for different dimensions ranging from 5 to 10. The results are depicted in Table 1.

Firstly, the Table 1 shows high memory occupation in 5-dimensional space compared to the low bound previously computed. This could be explained by the fact that Gaalop is doing symbolic algebra, not optimized to the bit level. Thus, each entry of the table stores an expression for evaluation in symbolic computing rather than mere bits, therefore this memory optimization becomes crucial even at smaller dimensions than a mere numerical implementation. From this table we can also infer that for a dimension greater than 10, the occupation of all the program will be greater than 1GB. Indeed, the occupation is roughly multiplied by 4 at each incrementation of the dimension. Thus, in 11-dimensional space, the memory occupation would be 2.8 GB and dimension 15 would require around 360 GB.

4.4 Select table components

Once the size of the table is decided, we have to decide which basis blade will be represented in the table. Firstly, we assume that the basis blades of the table are in grade-ascending order. Here is an example of this order in a 3-dimensional Euclidean space:

$$1,\ \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3,\ \mathbf{e}_{12}, \mathbf{e}_{13}, \mathbf{e}_{23},\ \mathbf{e}_{123} \tag{4}$$

In the average case, the bigger the grade of a basis blade, the higher will be the number of products between these two basis blades. To prove this, we consider the geometric product between two basis

blades **A** and **B** with respective grade k and l. These two basis blades will be denoted as $\mathbf{A}_{\langle k \rangle}$ and $\mathbf{B}_{\langle l \rangle}$ which is the notation used by Perwass in [16]. The geometric product of these two basis blades is:

$$\mathbf{AB} = \langle \mathbf{A}_{\langle k \rangle} \mathbf{B}_{\langle l \rangle} \rangle_{|k-l|} + \langle \mathbf{A}_{\langle k \rangle} \mathbf{B}_{\langle l \rangle} \rangle_{|k-l|+2} + \cdots + \langle \mathbf{A}_{\langle k \rangle} \mathbf{B}_{\langle l \rangle} \rangle_{k+l-2} + \langle \mathbf{A}_{\langle k \rangle} \mathbf{B}_{\langle l \rangle} \rangle_{k+l} \quad (5)$$

Thus, the number of elements is bounded above by:

$$k + l - |k - l| \quad (6)$$

Now suppose that we increase the grade of **A** and the grade of **B** by i and j respectively ($i, j \in \mathbb{N}$). The grade of **A** is now $k + i$ and the grade of **B** is $l + j$. Then from Equations (5) and (6), the new number of elements is upper bounded by:

$$k + l + i + j - \Big|(k - l) + (i - j)\Big| \quad (7)$$

From the triangle inequality, we get an upper bound on the number of elements of lower grade:

$$\Big|(k - l) + (i - j)\Big| \leq |k - l| + |i - j| \quad (8)$$

The latter inequality leads to a new upper bound to the number of elements:

$$k + l + i + j - \Big(|k - l| + |i - j|\Big) \quad (9)$$

By difference between the two upper bounds of Equation (9) and (6), we get the following formula:

$$i + j + |i - j| \quad (10)$$

If $i < j$ then $|i - j| = -i + j$ and Equation (10) becomes:

$$i + j - (i - j) = 2j \quad (11)$$

When $i > j$, then $|i - j| = i - j$ and Equation (10) is:

$$i + j + (i - j) = 2i \quad (12)$$

We get the same result if $i = j$.

This result shows that when the grade of the basis blades increases, the number of elements increases. Furthermore, the per-blade product of the table approach is computed in constant time, regardless of the grade of the multivectors. Thus, in the average case, it will be more efficient to use the table based approach for highest graded multivectors than for low graded multivectors.

5 INTEGRATION INTO GAALOP

The resulting implementation was integrated into Gaalop as a plug-in that includes the fully recursive approach explained in Section 4.1, the table computations presented in Section 4.2 and the hybrid approach explained in Section 4. For the latter approach, we first need to set the thresholds. Firstly, the threshold dimension is determined such that the memory used for the table does not exceed a limit. In the case of Gaalop, this limit is the heap size of the JVM, i.e. the total size of the table is set to 1GB. The limit dimension will thus be set to 10. From this limit dimension, the size of the table is thus 4^{10}. Secondly, the components of the table are the elements whose index are greater $2^d - 2^{10}$. Then, all the basis blades whose grade are higher than the grade of $2^d - 2^{10}$ will be computed using the table.

Dimension	6	7	8	9	10
Table (KB)	53	198	816	3,283	12,732
Binary tree (KB)	8.1	18	38	71	131

Table 2: Binary tree and table memory occupation

Dimension	11	12	13	14	15
Binary tree (KB)	258	524	1,031	2,108	4,124

Table 3: Binary tree memory occupation for dimensions from 11 to 15

To test our Plug-in, we compare it to the table based approach. Firstly, we test the memory occupation of our binary tree approach with the table based approach in the worst case. This worst case corresponds to the full binary tree (2^d non-null leaves). We compute the inner, geometric and outer product for dimensions ranging from 6 to 15, and run a memory profiler on the resulting programs. We also run the profiler on the binary tree program. All the results are detailed in Table 2. We can observe that in 10-dimensional space, the occupation of the table approach is 10 times greater than the binary tree approach. This factor can be explained by the fact that we also profile our binary tree approach for higher dimensional spaces from 11 to 15, see Table 3. Finally, we compared the runtime performance of our binary hybrid method compared to the iterative methods of section 3.3.1 for dimensions ranging from 5 to 15. The results show that the hybrid method is 9 times faster in a 15-dimensional space. This gain is higher for the outer product that is 30 times faster in 15-dimensional space.

6 CONCLUSION

In this paper, we presented a new Geometric Algebra implementation with a hybrid design between precomputations with tables and on-the-fly computations with a recursive binary tree. The program selects the first or the second method according to the dimension of the algebra and the grade of multivectors. Some tests were performed in high dimensional spaces and demonstrate the better performance of the proposed method, both in terms of memory and speed computation. The resulting implementation was integrated into Gaalop as a Plug-in. As future work, we would like to extend our approach to handle higher dimension applications, namely higher than 15.

REFERENCES

[1] Ab lamowicz, R., and Fauser, B. On parallelizing the clifford algebra product for CLIFFORD. *Advances in Applied Clifford Algebras 24*, 2 (2014), 553–567.

[2] Ab lamowicz, R., and Fauser, B. Using periodicity theorems for computations in higher dimensional clifford algebras. *Advances in Applied Clifford Algebras 24*, 2 (2014), 569–587.

[3] Breuils, S., Nozick, V., and Fuchs, L. A geometric algebra implementation using binary tree. *Advances in Applied Clifford Algebras* (2017), 1–19.

[4] Brombors ky, A. Galgebra. https://github.com/brombo/galgebra. Accessed: 2017-04-06.

[5] 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.

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

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

[8] Druoton, L., Fuchs, L., Garnier, L., and Langevin, R. The non-degenerate dupin cyclides in the space of spheres using geometric algebra. *Advances in Applied Clifford Algebras 24*, 2 (2014), 515–532.

[9] Easter, R. B., and Hitzer, E. Double conformal geometric algebra. *Advances in Applied Clifford Algebras* (April 2016), 28 pages.

[10] Eid, A. H. A. Optimized automatic code generation for geometric algebra based algorithms with ray tracing application. *arXiv preprint arXiv:1607.04767* (2016).

[11] Fontijne, D. Gaigen 2:: a geometric algebra implementation generator. In *Proceedings of the 5th international conference on Generative programming and component engineering* (2006), ACM, pp. 141–150.

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

[13] Hildenbrand, D. *Foundations of Geometric Algebra Computing*. Springer, 2013.

[14] Hildenbrand, D., Perwass, C., Dorst, L., and Fontijne, D. Geometric Algebra and its Application to Computer Graphics. In *Eurographics 2004 - Tutorials* (2004), Eurographics Association.

[15] Parkin, S. T. Galua. https://github.com/spencerparkin/GALua. Accessed: 2017-04-06.

[16] Perwass, C. *Geometric algebra with applications in engineering*, vol. 20. Springer, 2009.

[17] Perwass, C. CLUCal/CLUViz interactive visualization [online]. http://www.clucalc.info/, 2010.

[18] Prodanov, D. Clifford algebra implementation in maxima. *Alterman Conference on Geometric Algebra and Summer School on Kahler Calculus* (2016).

[19] Sangwine, S. J., and Hitzer, E. Clifford multivector toolbox (for MATLAB). *Advances in Applied Clifford Algebras 27*, 1 (2017), 539–558.

[20] Seybold, F., and Wössner, U. Gaalet-a c++ expression template library for implementing geometric algebra. In *6th High-End Visualization Workshop* (2010).