Geometric Algebra Computers

Dietmar Hildenbrand
TU Darmstadt, Germany
dhilden@gris.informatik.tu-darmstadt.de

ABSTRACT

Geometric algebra covers a lot of other mathematical systems like vector algebra, complex numbers, Plücker coordinates, quaternions etc. and it is geometrically intuitive to work with. Furthermore there is a lot of potential for optimization and parallelization.
In this paper, we investigate computers suitable for geometric algebra algorithms. While these *geometric algebra computers* are working in parallel, the algorithms can be described on a high level without thinking about how to parallelize them. In this context two recent developments are important. On one hand, there is a recent development of geometric algebra to an easy handling of engineering applications, especially in computer graphics, computer vision and robotics. On the other hand, there is a recent development of computer platforms from single processors to parallel computing platforms which are able to handle the high dimensional multivectors of geometric algebra in a better way.
We present our geometric algebra compilation approach for current and future hardware platforms like reconfigurable hardware, multi-core architectures as well as modern GPGPUs.

Keywords: Geometric algebra, GPGPU, multi-core-architecture, Verilog, OpenCL, CUDA, OpenMP, Ct, Larrabee.

1 INTRODUCTION

The foundation of geometric algebra was laid in 1844 and 1862 by Hermann Grassmann [8] whose 200th birthday we were celebrating in 2009 [25]. His work was continued by the English mathematician W. K. Clifford in 1878 [2]. Due to the early death of Clifford, the vector analysis of Gibbs and Heaviside dominated most of the 20th century, and not the geometric algebra. Geometric algebra has found its way into many areas of science, since David Hestenes treated the subject in the '60s [9]. In particular, his aim was to find a unified language for mathematics, and he went about to show the advantages that could be gained by using geometric algebra in many areas of physics and geometry [12], [10], [13] culminating in the development of the conformal geometric algebra [11]. Many other researchers followed and showed that applying geometric algebra in their field of research can be advantageous, e.g. in engineering areas like computer graphics, computer vision and robotics. Please find a survey on geometric algebra algorithms in [26].

During the past decades, especially from 1986 until 2002, processor performance doubled every 18 months. Currently, this improvement law is no longer valid because of technical limitations. Now, we can recognize a shift to parallel systems and most likely these systems will dominate the future. Thanks to multi-core architectures or powerful graphics boards for instance based on the CUDA technology from NVIDIA or on the future Larrabee technology of INTEL, one can expect impressive results using the powerful language of geometric algebra.

There is already a very advanced pure software solution called Gaigen (see [4] and [5]) as well as some pure hardware solutions geometric algebra algorithms (see for instance [24], [21] and [7] and a survey in [17]).

We propose to combine the advantages of both software and hardware solutions. We use a two-stage compilation approach for geometric algebra algorithms. In a first step we optimize geometric algebra algorithms with the help of symbolic computing. This kind of optimization results in very basic algorithms leading to highly efficient software implementations. These algorithms, foster a high degree of parallelization which are then used for hardware optimizations in a second step. As examples for geometric algebra computing we present

- a FPGA(field programmable gate array) implementation of an inverse kinematics algorithm.
- examples on how to implement geometric algebra algorithms on multi-core architectures. Since all of the coefficients of high dimensional multivectors can be computed in parallel, geometric algebra computing benefits a lot from highly parallel structures.
- a OpenCL/CUDA implementation for arbitrary geometric products using 2^n-dimensional multivectors of n-dimensional geometric algebras.

2 GEOMETRIC ALGEBRA COMPUTING APPROACH

Geometric algebra offers some very interesting properties like

- it is geometrically intuitive to work with
- it is easy to handle geometric objects like spheres, circles, planes etc. as well as geometric operations like rotations, reflections etc.

- geometric algebra algorithms are very compact
- it covers a lot of other mathematical systems like vector algebra, complex numbers, Plücker coordinates, quaternions etc.

How can we combine these properties with highly performant implementations? Multivectors of a n-dimensional geometric algebra are 2^n-dimensional. At first glance, this seems to be computationally very expensive. But, there is a lot of potential for optimization and parallelization of multivectors:

- the possibility of precomputing geometric algebra expressions
  - determine which of the coefficients are needed for the resulting multivector
  - symbolic simplification of the remaining coefficient computations
- Since all of the remaining coefficients can be computed in parallel, geometric algebra computations benefit a lot from parallel structures.

This is why we propose to separate geometric algebra computing in two layers

- geometric algebra (GA) compilation layer
- platform layer

At the GA compilation layer geometric algebra operations like geometric product, outer product, inner product, dual and reverse on multivectors are handled. This is compiled in a second step to the platform layer. On this layer only basic arithmetic operations on multivectors with a high potential for efficient computations on parallel platforms are available.

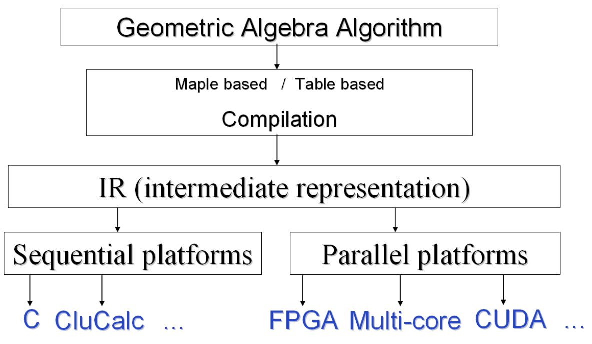
Geometric Algebra Algorithm
Maple based / Table based
Compilation
IR (intermediate representation)
Sequential platforms
Parallel platforms
C
CluCalc
...
FPGA
Multi-core
CUDA
...

Figure 1: Geometric algebra computing architecture. Algorithms are compiled to an intermediate representation for the compilation to different computing platforms.

Our geometric algebra computing architecture is presented in Figure 1. Algorithms (described by the geometric algebra programming language CLUCalc [23]) are compiled to an intermediate representation using a Maple based or a table based approach (see sections 3.1 and 3.2). Based on this representation implementations for different sequential and parallel platforms can be derived. See some examples for geometric algebra computer platforms based on FPGA- Multicore- and OpenCL/CUDA-architecture in sections 4.1 to 4.5.

3 GEOMETRIC ALGEBRA COMPILATION

In order to achieve highly efficient implementations, geometric algebra algorithms have to be optimized first. We use two different compilation approaches. The Maple based compilation needs the commercial Maple package and is restricted to geometric algebras with dimension $<= 9$. The table based compilation is able to handle higher dimensional algebras but it is currently not as powerful as the Maple based compilation.

3.1 Maple Based Compilation

The Maple based compilation uses the powerful symbolic computation feature of Maple [20]. Since all of the results of geometric algebra operations on multivectors are again multivectors we symbolically compute and simplify the resulting multivectors in order to determine which of the coefficients are actually needed and what is the most simple expression for each coefficient (in the Maple sense).

There is already a first implementation of a compiler for geometric algebra algorithms called Gaalop (Geometric algebra algorithms optimizer) working with this approach. Please find some information in [17]. You are able to download Gaalop from [16].

3.2 Table Based Compilation

The table based compilation approach uses precomputed multiplication tables inspired by the code generator Gaigen [6] from the university of Amsterdam. While Gaigen needs explicit specialization of multivectors this is done automatically in our approach (see the example below).

Multiplication tables In order to compute geometric algebra algorithms, the rules for the computation of the products of multivectors have to be known. These products of specific geometric algebras can be summarized (and precomputed) in multiplication tables describing the product of different blades of the algebra. You can find some examples of multiplication tables in the appendix. Table 1, for instance, describes the geometric product of the $8 = 2^3$ blades of the 3D Euclidean geometric algebra. Based on this information the geometric product of two multivectors, each defined as a linear combination of all the blades $mv = \sum mv_i E_i$ can be easily derived as described in the caption of Table 1.

The same procedure can be used for other products. Table 2, for instance, describes the outer product of the

mv_1	mv_2	mv_3	mv_4	mv_5	mv_6	mv_7	mv_8
E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}

3D Euclidean geometric algebra. Note that a lot of entries are zero corresponding to the outer product of two identical blades.

Example Let us compile the following CLUCalc script step by step:

```
a=a1*e1+a2*e2+a3*e3;
b=b1*e1+b2*e2+b3*e3;
?c=a*b;
d=a+c;
?f=a^d;
```

It computes the geometric product of two 3D vectors, adds two multivectors and computes the outer product of two multivectors.

The first two lines are used for the definition as well as for an automatic specialization of the two multivectors a

1	2	3	4	5	6	7	8
	a_1	a_2	a_3				

and b

1	2	3	4	5	6	7	8
	b_1	b_2	b_3				

For both, only the entries 2, 3 and 4 are needed since they correspond to the three basis vectors e_1, e_2, e_3 (see Table 1).

The question mark in the third line of the CLUCalc script indicates an explicit evaluation of this line, the geometric product of the two multivectors a and b. Table 3 shows the corresponding multiplication table for this product. It is derived from the Table 1 with empty rows and columns for multivector entries not needed for a and b. The resulting multivector c needs only the coefficients for the blades E_1, E_5, E_6, E_7 (see Table 3).

```
c[1]=a1*b1+a2*b2+a3*b3;
c[5]=a1*b2-a2*b1;
c[6]=a2*b3-a3*b2;
c[7]=a1*b3-a3*b1;
```

Each coefficient $c[k]$ can be computed by summing up the products $\pm a_i * b_j$ based on the E_k table entries, for instance $c_1 = a_1 * b_1 + a_2 * b_2 + a_3 * b_3$.

1	2	3	4	5	6	7	8
$c[1]$	a_1	a_2	a_3	$c[5]$	$c[6]$	$c[7]$	

In the fourth line of the CLUCalc script, two multivectors are added resulting in the following multivector d:

The evaluation of the outer product of a with this just computed multivector d leads to

```
f[2]=a1*c[1];
f[3]=a2*c[1];
f[4]=a3*c[1];
f[5]=a1*a2-a2*a1;
f[6]=a2*a3-a3*a2;
f[7]=a1*a3-a3*a1;
f[8]=-a2*c[7]+a3*c[5]+a1*c[6];
```

For this computation you can use the multiplication table 2. Associating the rows with the multivector a and the columns with d we are able to set the rows 1, 5, 6, 7, 8 as well as the column 8 to zero. We recognize that the remaining entries are for the coefficients 2, 3, 4, 5, 6, 7 and 8, E_2 for instance in the second row and the first column associated with the product $a_1 * c[1]$.

Note that the multivector entries 5, 6 and 7 lead to zero entries. This can be either recognized at compile time or at runtime. In both cases the resulting multivector f has the following form:

1	2	3	4	5	6	7	8
	$f[2]$	$f[3]$	$f[4]$				$f[8]$

4 GEOMETRIC ALGEBRA COMPUTERS

Here, computers suitable for geometric algebra algorithms, are called *geometric algebra computers* (GA computers).

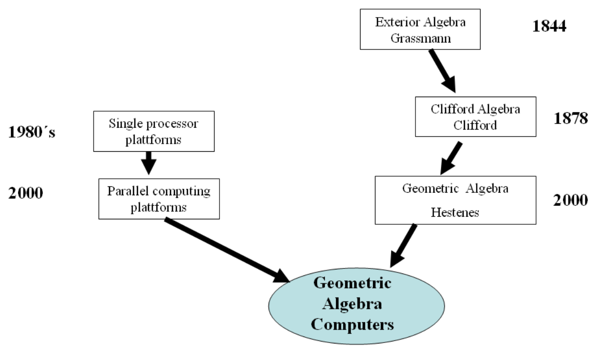

Figure 2: The mathematical development to geometric algebra and the computer development to parallel computing platforms leading to GA computers

There are mainly two recent developments leading to GA computers (see Figure 2):

- the development of mathematics from Grassmann´s exterior algebra to Clifford´s algebra to the geometric algebra of David Hestenes and especially the 5D conformal model leading to a lot of applications for instance in computer graphics, computer vision and robotics.
- the recent development of computer platforms for the mass market from single processors to parallel computing platforms which are able to handle the high dimensional multivectors of geometric algebra in a better way.

Figure 3 shows one example of an architecure able to compute the coefficients of a multivector in parallel.

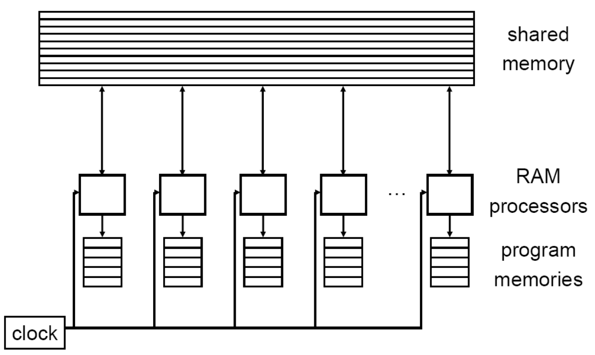

Figure 3: Computing architecture with a number of parallel processors, each consisting of local program memory. All the processors are able to communicate via global shared memory.

With the compilation approaches described in sections 3.1 and 3.2, geometric algebra algorithms are compiled into a description suitable for parallel computer platforms. In a next compilation step, the different platforms require different descriptions for their specific architecture. As follows, we describe solutions for a reconfigurable hardware implementation using Verilog, multi-core architectures using OpenMP and Ct as well as a GPGPU implementation using OpenCL/CUDA.

4.1 FPGA/Verilog implementation of a geometric algebra algorithm

There are general FPGA (field programmable gate arrays) implementations for geometric products ([24] and [7]). Our approach differs from these general solutions as we compile geometric algebra algorithms first into simplified algorithms that can be handled easily by FPGA´s. This is why we are not so much restricted in the length of the expressions to compute as well as in the dimension of the algebra.

Our FPGA implementations are always application specific. As one proof-of-concept for our approach we

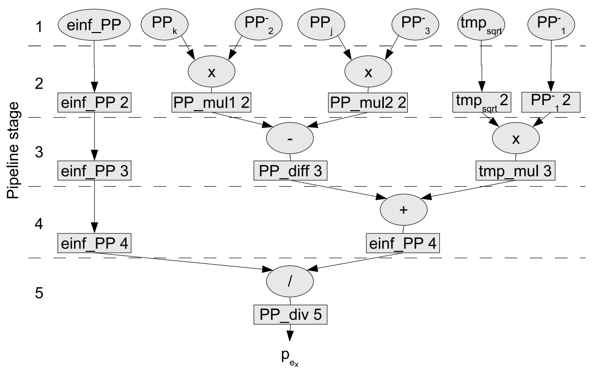

Figure 4: Pipeline schedule for the coefficient p_{ex} of a multivector. All the computations according to equation (1) of all the pipeline stages can be done in parallel.

implemented an inverse kinematics algorithm. First, we used our Maple based compilation approach (see section 3.1) and the software implementation of the optimized algorithm became three times faster than the conventional solution [14]. The FPGA implementation of the optimized algorithm used the Verilog programming language. See Figure 4 for the data flow and the pipeline schedule of the computation of the following part of the algorithm (one coefficient of one multivector)

$$\begin{aligned} p_{ex} = (PP_j(PP_{34} - PP_{35}) + PP_k(PP_{25} - PP_{24} \quad (1) \\ + tmp_{sqrt}(PP_{15} - PP_{14}))/einf\_PP. \end{aligned}$$

This implementation became about 300 times faster [15] (3 times by software optimization and 100 times by additional hardware optimization). The main advantage of this kind of implementation on reconfigurable hardware is that we are able to realize parallelism in two dimensions

- compute all the coefficients of one (or more) multivectors in parallel
- use the pipeline structure (computations in all pipeline stages at the same time).

4.2 OpenMP

OpenMP can be used in order to parallelize GA algorithms. The programming language C can be extended with OpenMP directives for an incremental approach to parallelizing code. For details on OpenMP, please refer to [1].

OpenMP supports task parallel computations. The data of all the different threads is shared by default. This is why the coefficients of multivectors can be computed in parallel (as well as independent multivectors). Using OpenMP for C, our above mentioned example looks as follows

```
#pragma omp parallel {
    #pragma omp sections {
      #pragma omp section
      c[1]=a1*b1+a2*b2+a3*b3;
      #pragma omp section
      c[5]=a1*b2-a2*b1;
      #pragma omp section
      c[6]=a2*b3-a3*b2;
      #pragma omp section
      c[7]=a1*b3-a3*b1;
    }/*End of sections block */

    #pragma omp sections
    {
      #pragma omp section
      f[2]=a1*c1;
      #pragma omp section
      f[3]=a2*c[1];
      #pragma omp section
      f[4]=a3*c[1];
      #pragma omp section
      f[5]=a1*a2-a2*a1;
      #pragma omp section
      f[6]=a2*a3-a3*a2;
      #pragma omp section
      f[7]=a1*a3-a3*a1;
      #pragma omp section
      f[8]=-a2*c[7]+a3*c[5]+a1*c[6];

    }/*End of sections block */

  } /*End of parallel region */
```

Each of the two multivectors *c* and *f* have to be computed sequentially because *f* needs the result of *c* for its computation (while all of their coefficients can be computed in parallel). In case of no dependance of the computations, multivectors can also be computed in parallel.

4.3 Ct

Intel researchers are developing Ct, or C/C++ for Throughput Computing [18] in order to support their new multi-core platform (code name Larrabee).

Ct offers parallelism on so-called *indexed vectors* suitable for sparse multivectors. The fist step of our example of section 3 generates a multivector which can be described as the following indexed vector

```
  c= [(1 -> a1*b1+a2*b2+a3*b3),
      (5 -> a1*b2-a2*b1),
      (6 -> a2*b3-a3*b2),
      (7 -> a1*b3-a3*b1),
      (_ -> 0)
     ]
```

Note that the underscore denotes a default value for empty coefficients.

All operators on indexed vectors are implicitly parallel. This is why the addition of multivectors of our example

```
  d=a+c;
```

can be done very easily in Ct.

4.4 ATI stream

The ATI stream technology combines multiple thread computing with parallel computing within the threads. The following sample code computes the geometric product of the above example with the help of float4 vectors. The 4 computations for the coefficients x,y,z,w are computed in parallel.

```
  kernel void MV (float4 a<>,
                  float4 b<>,
                  out float4 c<>){
    float4 result;
    result.x=a.x*b.x+a.y*b.y+a.z*b.z;
    result.y=a.x*b2-a2*b.x;
    result.z=a.y*b.z-a.z*b.y;
    result.w=a.x*b.z-a.z*b.x;
    c=result;
  }
```

Please find an investigation about a ray tracing application using this technology in [3].

4.5 OpenCL/CUDA implementation of arbitrary geometric products

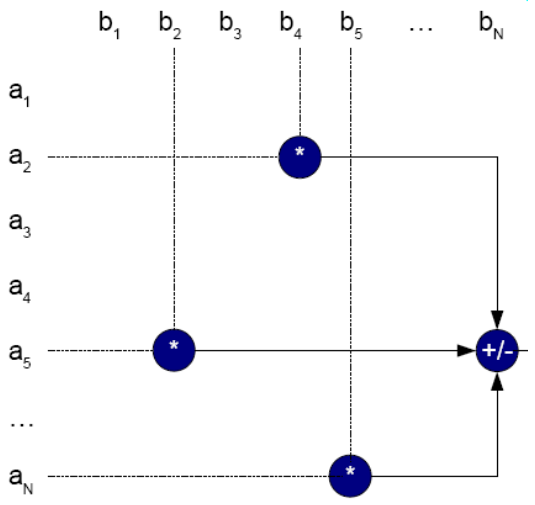

Figure 5: The result of a product of two multivectors a, b is again a multivector. Each of its coefficients is a sum of (signed or unsigned) products of coefficients of a and b.

OpenCL [19] is an open standard for parallel programming of heterogeneous systems. It is inspired by Nvidia´s CUDA technology [22]. Both, are supporting multiple threads which are able to run the same code with different data on many parallel processors.

The result of products of a n-dimensional geometric algebra are always 2^n-dimensional multivectors. Each of the 2^n coefficients can be computed as a sum of (signed or unsigned) products of coefficients of the multivectors to be multiplied (as indicated in Figure 5). We distribute this computation to 2^n threads, each computing one coefficient.

```
Calculate(const float* A, const float* B, float* C) {
    int coeff = threadIdx.x;
    float res = 0.0;
    int offset = ...;
    int length = ...;
    for each relevant summand:
    {
      Summand s = ...;
      res += A[s.first] * B[s.second] * s.sign;
    }
    C[coeff] = res;
}
```

Figure 6: Pseudo code for the computation of one coefficient of a geometric product.

Figure 6 describes the specific kernel code for one thread, each computing one coefficient of the 2^n-dimensional multivector.

Please find some details on this application in [27].

5 RESULTS

Our geometric algebra computing approach is able to generate implementations for different sequential and parallel platforms (see Figure 3). While some of the described implementations are still work in progress, we already have results for implementations in C, for a FPGA and for CUDA.

Our first test case was the inverse kinematics of the arm of a virtual character in a virtual reality application. Naively implemented on a sequential processor platform, the first geometric algebra algorithm was initially slower than the conventional one. However, with our Maple based optimization approach the software implementation became three times faster [14] than the conventional solution. The hardware implementation on a FPGA (as described in section 4.1) became even 300 times faster [15].

The results of our CUDA implementation of arbitrary geometric products can be found in [27].

Recently we investigated the runtime performance of a robotics grasping algorithm described in geometric algebra [28]. It turned out that the implementation on a sequential processor was 14 times faster and on the CUDA platform 44 times faster than the solution with conventional mathematics.

6 CONCLUSION AND FUTURE WORK

We presented the currently most suitable geometric algebra computing platforms. For the adaptation of the algorithms to the different platforms we presented our compilation approach. While the Maple based compilation approach is able to handle algebras up to a dimension of 9, the table based approach is restricted by the memory needed for the size of the multiplication tables. These tables are exponentially increasing with the dimension of the algebra. In this context, investigations for lower amounts of memory are needed, for instance the implementation on a multiplicative basis as described in [5].

Currently, the presented parallel computing platforms can be seen as approximations to perfect GA computers. As a long-term vision, we hope that this research will lead to computing platforms optimally supporting GA computers in the future.

ACKNOWLEDGEMENTS

This work was supported by the DFG (Deutsche Forschungsgemeinschaft) project HI 1440/1-1.

A MULTIPLICATION TABLES

Table 1: Multiplication table describing the geometric product of two multivectors $a = \sum a_i E_i$ and $b = \sum b_i E_i$ for the 3D euclidean GA. Each entry describes the geometric product of two basis blades E_i and E_j expressed in terms of the basis blades E_k. Each coefficient c_k of the product $c = ab$ can be computed by summing up the products $\pm a_i * b_j$ based on the E_k table entries, for instance $c_1 = a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + a_4 * b_4 - a_5 * b_5 - a_6 * b_6 - a_7 * b_7 - a_8 * b_8$ for the E_1 table entries

	b	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
a		1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}
E_1	1	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
E_2	e_1	E_2	E_1	E_5	E_7	E_3	E_8	E_4	E_6
E_3	e_2	E_3	-E_5	E_1	E_6	-E_2	E_4	-E_8	-E_7
E_4	e_3	E_4	-E_7	-E_6	E_1	E_8	-E_3	-E_2	E_5
E_5	e_{12}	E_5	-E_3	E_2	E_8	-E_1	E_7	-E_6	-E_4
E_6	e_{23}	E_6	E_8	-E_4	E_3	-E_7	-E_1	E_5	-E_2
E_7	e_{13}	E_7	-E_4	-E_8	E_2	E_6	-E_5	-E_1	E_3
E_8	e_{123}	E_8	E_6	-E_7	E_5	-E_4	-E_2	E_3	-E_1

Table 2: Multiplication table describing the outer product of two general multivectors $a = \sum a_i E_i$ and $b = \sum b_i E_i$ for the 3D euclidean GA.

	b	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
a		1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}
E_1	1	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
E_2	e_1	E_2	0	E_5	E_7	0	E_8	0	0
E_3	e_2	E_3	-E_5	0	E_6	0	0	-E_8	0
E_4	e_3	E_4	-E_7	-E_6	0	E_8	0	0	0
E_5	e_{12}	E_5	0	0	E_8	0	0	0	0
E_6	e_{23}	E_6	E_8	0	0	0	0	E_5	0
E_7	e_{13}	E_7	0	-E_8	0	0	0	0	0
E_8	e_{123}	E_8	0	0	0	0	0	0	0

Table 3: Multiplication table describing the geometric product of two vectors $a = a_1e_1 + a_2e_2 + a_3e_3$ and $b = b_1e_1 + b_2e_2 + b_3e_3$ for the 3D euclidean GA. Note that all the rows and columns for basis blades not needed for the vectors are set to 0.

		b		$\mathbf{b_1}$	$\mathbf{b_2}$	$\mathbf{b_3}$				
			E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
a			1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}
	E_1	1	0	0	0	0	0	0	0	0
$\mathbf{a_1}$	E_2	e_1	0	E_1	E_5	E_7	0	0	0	0
$\mathbf{a_2}$	E_3	e_2	0	-E_5	E_1	E_6	0	0	0	0
$\mathbf{a_3}$	E_4	e_3	0	-E_7	-E_6	E_1	0	0	0	0
	E_5	e_{12}	0	0	0	0	0	0	0	0
	E_6	e_{23}	0	0	0	0	0	0	0	0
	E_7	e_{13}	0	0	0	0	0	0	0	0
	E_8	e_{123}	0	0	0	0	0	0	0	0

REFERENCES

[1] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP : portable shared memory parallel programming*. The MIT Press, 2008.

[2] William Kingdon Clifford. *Applications of Grassmann's Extensive Algebra*, volume 1 of *American Journal of Mathematics*, pages 350–358. The Johns Hopkins University Press, 1878.

[3] Crispin Deul, Michael Burger, Dietmar Hildenbrand, and Andreas Koch. Raytracing point clouds using geometric algebra. In *submitted to the proceedings of the GraVisMa workshop, Plzen*, 2010.

[4] Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric Algebra for Computer Science, An Object-Oriented Approach to Geometry*. Morgan Kaufman, 2007.

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

[6] Daniel Fontijne, Tim Bouma, and Leo Dorst. Gaigen 2: A geometric algebra implementation generator. Available at `http://staff.science.uva.nl/~fontijne/gaigen2.html`, 2007.

[7] Antonio Gentile, Salvatore Segreto, Filippo Sorbello, Giorgio Vassallo, Salvatore Vitabile, and Vincenzo Vullo. Cliffosor, an innovative fpga-based architecture for geometric algebra. In *ERSA 2005*, pages 211–217, 2005.

[8] Hermann Grassmann. *Die Ausdehnungslehre*. Verlag von Th. Chr. Fr. Enslin, Berlin, 1862.

[9] David Hestenes. *Space-Time Algebra (Documents on Modern Physics)*. Gordon and Breach, 1966.

[10] David Hestenes. *New Foundations for Classical Mechanics*. Dordrecht, 1986.

[11] David Hestenes. Old wine in new bottles : A new algebraic framework for computational geometry. In Eduardo Bayro-Corrochano and Garret Sobczyk, editors, *Geometric Algebra with Applications in Science and Engineering*. Birkhäuser, 2001.

[12] David Hestenes and Garret Sobczyk. *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*. Dordrecht, 1984.

[13] David Hestenes and Renatus Ziegler. Projective Geometry with Clifford Algebra. *Acta Applicandae Mathematicae*, 23:25–63, 1991.

[14] Dietmar Hildenbrand, Daniel Fontijne, Yusheng Wang, Marc Alexa, and Leo Dorst. Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra. In *Eurographics conference Vienna*, 2006.

[15] Dietmar Hildenbrand, Holger Lange, Florian Stock, and Andreas Koch. Efficient inverse kinematics algorithm based on conformal geometric algebra using reconfigurable hardware. In *GRAPP conference Madeira*, 2008.

[16] Dietmar Hildenbrand and Joachim Pitt. The Gaalop home page. Available at http://www.gaalop.de, 2008.

[17] Dietmar Hildenbrand, Joachim Pitt, and Andreas Koch. Gaalop - high performance parallel computing based on conformal geometric algebra. In Eduardo Bayro-Corrochano and Gerik Scheuermann, editors, *Geometric Algebra Computing for Engineering and Computer Science*. Springer, 2009.

[18] Intel. Ct: C for throughput computing home page. Available at http://techresearch.intel.com/articles/Tera-Scale/1514.htm, 2009.

[19] Khronos-Group. The OpenCL home page. Available at `http://www.khronos.org/opencl/`, 2009.

[20] The homepage of maple. Available at http://www.maplesoft.com/products/maple, 2009. 615 Kumpf Drive, Waterloo, Ontario, Canada N2V 1K8.

[21] Biswajit Mishra and Peter Wilson. Hardware implementation of a geometric algebra processor core. In *Proceedings of IMACS International Conference on Applications of Computer Algebra (in press), Nara, Japan*, 2005.

[22] NVIDIA. The CUDA home page. Available at http://www.nvidia.com/object/cuda_home.html, 2009.

[23] Christian Perwass. The CLU home page. Available at http://www.clucalc.info, 2005.

[24] Christian Perwass, Christian Gebken, and Gerald Sommer. Implementation of a clifford algebra co-processor design on a field programmable gate array. In Rafal Ablamowicz, editor, *CLIFFORD ALGEBRAS: Application to Mathematics, Physics, and Engineering*, Progress in Mathematical Physics, pages 561–575. 6th Int. Conf. on Clifford Algebras and Applications, Cookeville, TN, Birkhäuser, Boston, 2003.

[25] Hans-Joachim Petsche. The Grassmann Bicentennial Conference home page. Available at http://www.uni-potsdam.de/u/philosophie/grassmann/Papers.htm, 2009.

[26] Alyn Rockwood and Dietmar Hildenbrand. Engineering graphics in geometric algebra. In Eduardo Bayro-Corrochano and Gerik Scheuermann, editors, *Geometric Algebra Computing for Engineering and Computer Science*. Springer, 2009.

[27] Christian Schwinn, A Goerlitz, and Dietmar Hildenbrand. Geometric algebra computing on the cuda platform. In *submitted to the proceedings of the GraVisMa workshop, Plzen*, 2010.

[28] Florian Wörsdörfer, Bayro-Corrochano Eduardo Stock, Florian, and Dietmar Hildenbrand. Optimization and performance of a robotics grasping algorithm described in geometric algebra. In *Iberoamerican Congress on Pattern Recognition 2009, Guadalajara, Mexico*, 2009.