

From Grassmann's vision to Geometric Algebra Computing

Dietmar Hildenbrand

1. Introduction

What mathematicians often call *Clifford algebra* is called *geometric algebra* if the focus is on the geometric meaning of the algebraic expressions and operators. Geometric algebra is a mathematical framework to easily describe geometric concepts and operations. It allows us to develop algorithms fast and in an intuitive way. Geometric algebra is based on the work of Hermann Grassmann

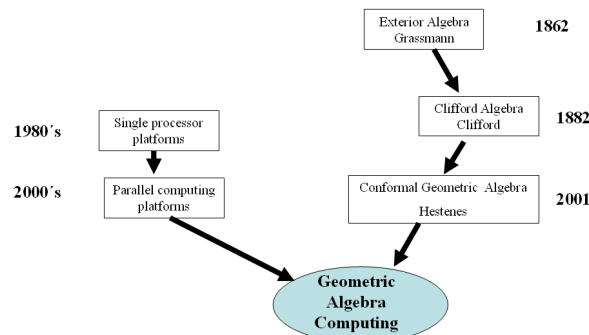


FIGURE 1. Two trends towards what we call geometric algebra computing.

[Grassmann, 1862] and his vision of a general mathematical language for geometry. William Clifford combined Grassmann's exterior algebra and Hamilton's quaternions [Clifford, 1882a], [Clifford, 1882b]. Pioneering work has been done by David Hestenes, who first applied geometric algebra to problems in mechanics and physics [Hestenes and Sobczyk, 1984] [Hestenes, 1986]. His work was culminating some years ago in the invention of conformal geometric algebra [Hestenes, 2001]. Grassmann's outer product leads to high dimensional multivectors. Currently, we can

recognize a shift from single processor platforms to parallel computing platforms which are able to treat these multivectors in an efficient way. 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 Currently, this improvement law is no longer valid because of technical limitations. The new trend to parallel systems will most likely dominate the future. Thanks to multi-core architectures or powerful graphics boards for instance based on the CUDA technology from NVIDIA, the ATI stream technology of AMD or on the future Larrabee technology of INTEL, one can expect impressive results using the powerful language of geometric algebra. We call this combination of applying geometric algebra on parallel platforms *geometric algebra computing*.

In this chapter we highlight some benefits of conformal geometric algebra and present some examples of *geometric algebra computing* on different parallel computing platforms.

2. Benefits of Conformal Geometric Algebra

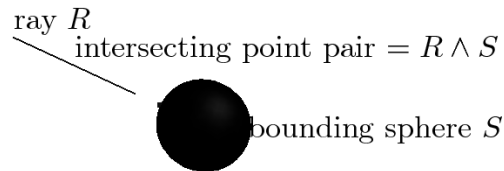


FIGURE 2. Spheres and lines are basic entities of geometric algebra to compute with. Operations such as the intersection of them are easily expressed with the help of their outer product. In a ray tracing application, for instance, the result of the intersection of a ray and a (bounding) sphere is another geometric entity: the point pair of the two points of the line intersecting the sphere. The sign of the square of the point pair easily indicates whether there is a real intersection or not.

Geometric algebra as a general mathematical system unites many mathematical concepts such as vector algebra, quaternions, Plücker coordinates and projective geometry, and it easily deals with geometric objects, operations and transformations. Many applications in computer graphics, computer vision and other engineering areas can benefit from these properties. In a ray tracing application, for instance, the intersection of a ray and a bounding sphere is needed. According to Figure 2, this can be easily expressed with the help of the outer product of these two geometric entities.

As follows we highlight some of the properties of geometric algebra that make it advantageous for many engineering applications.

2.1. Unification of mathematical systems

TABLE 1. Multiplication table of the 2D geometric algebra. This algebra consists of basic algebraic objects of grade (dimension) 0, the scalar, of grade 1, the two basis vectors e_1 and e_2 and of grade 2, the bivector $e_1 \wedge e_2$, which can be identified with the imaginary number i squaring to -1

	1	e_1	e_2	$e_1 \wedge e_2$
1	1	e_1	e_2	$e_1 \wedge e_2$
e_1	e_1	1	$e_1 \wedge e_2$	e_2
e_2	e_2	$-e_1 \wedge e_2$	1	$-e_1$
$e_1 \wedge e_2$	$e_1 \wedge e_2$	$-e_2$	e_1	-1

In the wide range of engineering applications many different mathematical systems are currently used. One notable advantage of geometric algebra is that it subsumes mathematical systems such as vector algebra, complex analysis, quaternions or Plücker coordinates. Table 1, for instance, describes how complex numbers can be identified within the 2D geometric algebra. This algebra does not only contain the two basis vectors e_1 and e_2 , but also basis elements of grade (dimension) 0 and 2. The grade 0 represents the scalar. With the help of the rules of geometric algebra it can be shown quite easily that the element $e_1 \wedge e_2$ (of grade 2) squares to -1 and can be identified with the imaginary unit i . The linear combination of these two elements describe all the complex numbers. This imaginary unit as well as all the imaginary units of quaternions, describing 3D rotations, can be identified in Figure 4 as elements in the conformal geometric algebra, the geometric algebra of conformal space.

2.2. Intuitive handling of geometric objects

Conformal geometric algebra is a 5D geometric algebra based on the 3D basis vectors e_1, e_2 and e_3 as well as on the two additional base vectors e_0 representing the origin and e_∞ representing infinity. This algebra is able to easily treat different geometric objects. Table 2 shows the representation of points, spheres, planes, circles and lines as the same entities algebraically. Consider the spheres of Figure 3, for instance. These spheres are simply represented by

$$S = P - \frac{1}{2}r^2e_\infty \quad (2.1)$$

based on their center point P , their radius r and the basis vector e_∞ . The circle of intersection of two spheres is then easily computed using the outer product to operate on the spheres as simply as if they were vectors.

$$Z = S_1 \wedge S_2 \quad (2.2)$$

This way of computing with conformal geometric algebra clearly benefits engineering applications.

TABLE 2. List of the basic geometric primitives provided by the 5D conformal geometric algebra. The bold characters represent 3D entities (\mathbf{x} is a 3D point, \mathbf{n} is a 3D normal vector and \mathbf{x}^2 is the scalar product of the 3D vector \mathbf{x}). The two additional basis vectors e_0 and e_∞ represent the origin and infinity. Based on the outer product, circles and lines can be described as intersections of two spheres, respectively two planes. The parameter r represents the radius of the sphere and the parameter d the distance of the plane to the origin.

entity	representation
Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$
Sphere	$S = P - \frac{1}{2}r^2 e_\infty$
Plane	$\pi = \mathbf{n} + d e_\infty$
Circle	$Z = S_1 \wedge S_2$
Line	$L = \pi_1 \wedge \pi_2$

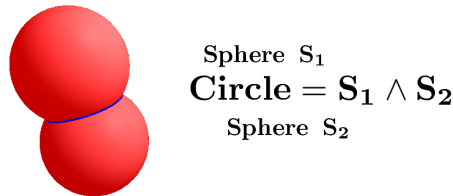
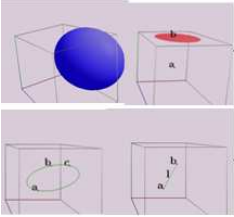


FIGURE 3. Spheres and circles are basic entities of geometric algebra. Operations such as the intersection of two spheres are easily expressed.

2.3. Intuitive handling of geometric operations

Looking more into the details of conformal geometric algebra, it consists of blades with *grades* (dimension) 0, 1, 2, 3, 4 and 5, whereby a scalar is a *0-blade* (blade of grade 0). The element of grade five is called the pseudoscalar. A linear combination of blades is called a *k-vector*. So a bivector is a linear combination of blades with grade 2. Other k-vectors are vectors (grade 1), trivectors (grade 3) and quadvectors (grade 4). Furthermore, a linear combination of blades of different grades is called a *multivector*. Multivectors are the general elements of a geometric algebra. Figure 4 lists all the 32 blades of conformal geometric algebra. It shows the power of algebraic expressions of geometric algebra in describing different geometric objects as well as geometric operations. Operations such as rotations, translations (see [Hildenbrand et al., 2004]) and reflections can be easily treated within the algebra. There is no need to change the way of describing them



grade	term	blades	nr.
0	scalar	1	1
1	vector	$e_1, e_2, e_3, e_0, e_\infty$	5
2	bivector	$e_1 \wedge e_2, e_1 \wedge e_3, e_2 \wedge e_3,$ $e_1 \wedge e_\infty, e_2 \wedge e_\infty, e_3 \wedge e_\infty,$ $e_1 \wedge e_0, e_2 \wedge e_0, e_3 \wedge e_0,$ $e_0 \wedge e_\infty$	10
3	trivector	...	10
4	quadvector	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty,$ $e_1 \wedge e_2 \wedge e_3 \wedge e_0,$ $e_1 \wedge e_2 \wedge e_0 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	5
5	pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	1

3.1416
i, j, k

FIGURE 4. The blades of conformal geometric algebra. Spheres and planes, for instance, are vectors. Lines and circles can be represented as bivectors. Other mathematical systems like complex numbers or quaternions can be identified based on their imaginary units i, j, k . This is why also transformations like rotations can be handled within the algebra.

with other approaches (vector algebra, for instance, additionally needs matrices in order to describe transformations).

2.4. Robotics application example

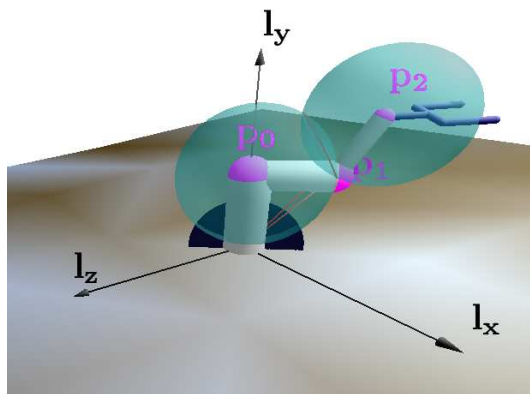


FIGURE 5. Computation of P_1

Let us look, for instance, at an inverse kinematics application of the simple robot of Figure 5. Assuming that we already know the locations of the joints P_0 and

P_2 , we have to compute the P_1 in the next step. Computing this point is usually a difficult task. However, using conformal geometric algebra we can determine it by intersecting the spheres S_1 and S_2

$$S_1 = P_0 - \frac{1}{2}d_2^2e_\infty, \quad (2.3)$$

$$S_2 = P_2 - \frac{1}{2}d_3^2e_\infty, \quad (2.4)$$

with the plane π_1 (describing the plane the joints are acting in)

$$Pp_1 = S_1 \wedge S_2 \wedge \pi_1. \quad (2.5)$$

The result is a point pair Pp_1 and we have to choose one point from this point pair.

Please find the complete algorithm in [Hildenbrand, 2005].

3. Geometric Algebra Computing Technology

The power of geometric algebra as described in the previous section comes along with a complex algebraic structure of high dimensional multivectors. Fortunately, this algebraic structure offers a high potential for optimization and parallelization that we can advantageously use for highly efficient implementations on current parallel processor platforms.

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 by symbolically pre-computing geometric algebra expressions. Since all of the coefficients of the multivectors can be computed in parallel, geometric algebra computations benefit significantly from parallel structures.

This is why we separate geometric algebra computing in two layers

- geometric algebra (GA) compilation layer
- platform layer

At the GA compilation layer, geometric algebra operations such as geometric product, outer product, inner product, dual and reverse on multivectors are handled. These operations are compiled 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.

Our geometric algebra computing architecture is presented in Figure 6. Algorithms (described by the geometric algebra programming language CLUCalc [Perwass, 2005]) are compiled to an intermediate representation using an adequate compilation approach. Based on this representation implementations for different sequential and parallel platforms can be derived.

We are just developing a compiler for reconfigurable hardware using the Maple based compilation approach (see section 3.1). A proof-of-concept of our approach has been done with the help of a FPGA(field programmable gate array)

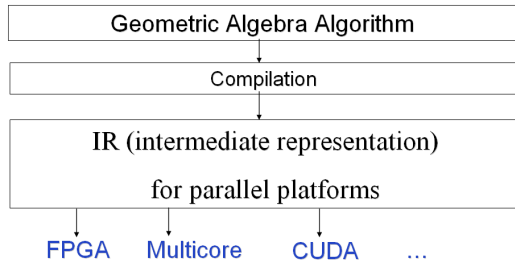


FIGURE 6. Geometric algebra computing architecture. Algorithms are compiled to an intermediate representation for the compilation to different computing platforms.

implementation of an inverse kinematics algorithm. Naively implemented, the inverse kinematics algorithm was initially slower than the conventional one. However, with our symbolic computation optimization approach the software implementation became three times faster [Hildenbrand et al., 2006] and with a hardware implementation about 300 times faster [Hildenbrand et al., 2008] (3 times by software optimization and 100 times by additional hardware optimization) than the conventional software implementation. Herewith, we could show for the first time that implementations of geometric algebra algorithms can be faster than solutions based on standard mathematics.

Please find some examples for geometric algebra computing based on current parallel architectures in section 3.2. They vary in different parallelization concepts such as multi-core or SIMD (single instruction multiple data).

3.1. Compilation

In order to achieve highly efficient implementations, geometric algebra algorithms have to be optimized at first. Currently we use a Maple based compilation approach. It needs the commercial Maple system [MAP, 2009] as well as a specific geometric algebra package [Ablamowicz and Fauser, 2009] and is restricted to geometric algebras with dimension ≤ 9 . The Maple based compilation uses the powerful symbolic computation feature of Maple. 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. The following short algorithm, for instance,

```

a=a1*e1+a2*e2+a3*e3;
b=b1*e1+b2*e2+b3*e3;
c=a*b;

```

computes the geometric product of two 3D vectors and assigns it to the multivector c . The compiler optimizes this algorithm in the following form

```

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;

```

with simple arithmetic operations for each coefficient which is needed for the multivector c .

3.2. Adaptation to different parallel processor platforms

One goal of this research proposal is to adapt geometric algebra computing to the best suitable parallel processor platforms. Here are some examples how this could work from the point-of-view of the programming languages of different example architectures.

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 [Chapman et al., 2008].

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, parallel sections can be programmed as follows

```

#pragma omp parallel {
  #pragma omp sections {
    #pragma omp section
    ... Section 1 ...
    #pragma omp section
    ... Section 2 ...

    #pragma omp section
    ... Section 3 ...

  }/*End of sections block */
} /*End of parallel region */

```

Each section consists of computations of different coefficients of a multivector as well as of computations of independent multivectors.

Intels Ct

Intel researchers are developing Ct, or C/C++ for Throughput Computing [Intel, 2009] in order to support their new multi-core platforms.

Ct offers parallelism on so-called *indexed vectors* suitable for sparse multivectors. The first step of our example 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

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

can be done very easily in Ct.

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.

```

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;
}

```

The 4 computations for the coefficients x,y,z,w are computed in parallel.

CUDA

CUDA [NVIDIA, 2009] is Nvidias technology for their parallel computing platforms. It is supporting multiple threads which are able to run the same code with different data on many parallel processors. This SIMD (single instruction multiple data) technology can be used - as usual - for geometric algebra algorithms tasks with the same code operating on different data.

For high dimensional algebras, there is also another advantage. The results of products in n -dimensional geometric algebras are always 2^n -dimensional multivectors. Using multiplication tables, 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. These computations can be distributed to 2^n threads, each computing one coefficient. The kernel code can be identical for all the threads, assuming that each threads knows its individual part of the multiplication table.

4. Conclusion

Geometric algebra and especially the 5D conformal geometric algebra can be applied in a wide range of engineering applications with geometric background. It is a very powerful mathematical framework in terms of geometric intuitiveness, compactness and simplicity. Who can benefit from the properties of geometric algebra? There is no need for students of learning different mathematical systems and the translations between them since a lot of other mathematical systems are already included in geometric algebra. Researchers can benefit when developing new solutions in their field of research. They are able to gain new insights based on this more global mathematical system. From the academic point of view, geometric algebra computing is a very inter-disciplinary topic between mathematics, computer science and engineering. It is still a basic research topic, but with a high potential for engineering applications in industry. For companies, geometric algebra technology can lead to an enhancement of quality as well as to a reduction of costs for the development, documentation and maintenance of their products and solutions.

Last but not least there is a longer term vision based on geometric algebra technology: For the hardware industry the need of suitable parallel architectures is quite evident. After the adaption of geometric algebra algorithms to current hardware architectures, future architectures can even be influenced and driven by geometric algebra computing technology.

Acknowledgements

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

References

- [Abłamowicz and Fauser, 2009] Abłamowicz, R. and Fauser, B. (2009). Clifford/bigebra, a maple package for clifford (co)algebra computations. ©1996-2009, RA&BF.
- [Chapman et al., 2008] Chapman, B., Jost, G., and van der Pas, R. (2008). *Using OpenMP : portable shared memory parallel programming*. The MIT Press.
- [Clifford, 1882a] Clifford, W. K. (1882a). Applications of grassmann's extensive algebra. In Tucker, R., editor, *Mathematical Papers*, pages 266–276. Macmillian, London.
- [Clifford, 1882b] Clifford, W. K. (1882b). On the classification of geometric algebras. In Tucker, R., editor, *Mathematical Papers*, pages 397–401. Macmillian, London.
- [Grassmann, 1862] Grassmann, H. (1862). *Die Ausdehnungslehre*. Verlag von Th. Chr. Fr. Enslin, Berlin.
- [Hestenes, 1986] Hestenes, D. (1986). *New Foundations for Classical Mechanics*. Dordrecht.

- [Hestenes, 2001] Hestenes, D. (2001). Old wine in new bottles : A new algebraic framework for computational geometry. In Bayro-Corrochano, E. and Sobczyk, G., editors, *Geometric Algebra with Applications in Science and Engineering*. Birkhäuser.
- [Hestenes and Sobczyk, 1984] Hestenes, D. and Sobczyk, G. (1984). *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*. Dordrecht.
- [Hildenbrand, 2005] Hildenbrand, D. (2005). Geometric computing in computer graphics using conformal geometric algebra. *Computers & Graphics*, 29(5):802–810.
- [Hildenbrand et al., 2004] Hildenbrand, D., Fontijne, D., Perwass, C., and Dorst, L. (2004). Tutorial geometric algebra and its application to computer graphics. In *Eurographics conference Grenoble*.
- [Hildenbrand et al., 2006] Hildenbrand, D., Fontijne, D., Wang, Y., Alexa, M., and Dorst, L. (2006). Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra. In *Eurographics conference Vienna*.
- [Hildenbrand et al., 2008] Hildenbrand, D., Lange, H., Stock, F., and Koch, A. (2008). Efficient inverse kinematics algorithm based on conformal geometric algebra using reconfigurable hardware. In *GRAPP conference Madeira*.
- [Intel, 2009] Intel (2009). Ct: C for throughput computing home page. Available at <http://techresearch.intel.com/articles/Tera-Scale/1514.htm>.
- [MAP, 2009] MAP (2009). The homepage of maple. Available at <http://www.maplesoft.com/products/maple>. 615 Kumpf Drive, Waterloo, Ontario, Canada N2V 1K8.
- [NVIDIA, 2009] NVIDIA (2009). The CUDA home page. Available at http://www.nvidia.com/object/cuda_home.html.
- [Perwass, 2005] Perwass, C. (2005). The CLU home page. Available at <http://www.clucalc.info>.

5. Abstract

The foundation of geometric algebra was laid in 1844 and 1862 by Hermann Grassmann whose 200th birthday we are celebrating this year. His vision of a mathematical language for geometry culminated some years ago in the invention of conformal geometric algebra by David Hestenes. Grassmann's outer product leads to high dimensional multivectors. Currently, we can recognize a shift to parallel computing platforms which are able to treat these multivectors in an efficient way. We call this combination of applying geometric algebra on parallel platforms *geometric algebra computing*. In this chapter we highlight some properties of conformal geometric algebra and present some examples of *geometric algebra computing* on different parallel computing platforms.

Dr. Dietmar Hildenbrand

Educational background: Study and Ph. D. in computer science

Current position: senior scientist at TU Darmstadt

Main research interest: the application of geometric algebra in computer graphics, computer vision and robotics, the efficient implementation of geometric algebra algorithms

Selected publications: "Geometric Computing in Computer Graphics using Conformal Geometric Algebra" (2005), Computers and Graphics

"Efficient Inverse Kinematics Algorithm Based on Conformal Geometric Algebra Using Reconfigurable Hardware" (2008) with Holger Lange, Florian Stock and Andreas Koch, Grapp conference

book chapter "Gaalop - High Performance Parallel Computing based on Conformal Geometric Algebra" (2009) with Joachim Pitt and Andreas Koch in Geometric Algebra Computing for Engineering and Computer Science, Springer Verlag

Dietmar Hildenbrand

Address: TU Darmstadt, Hochschulstr. 10, 64289 Darmstadt

e-mail: dhilden@gris.informatik.tu-darmstadt.de