

1 Geometric Algebra Computing

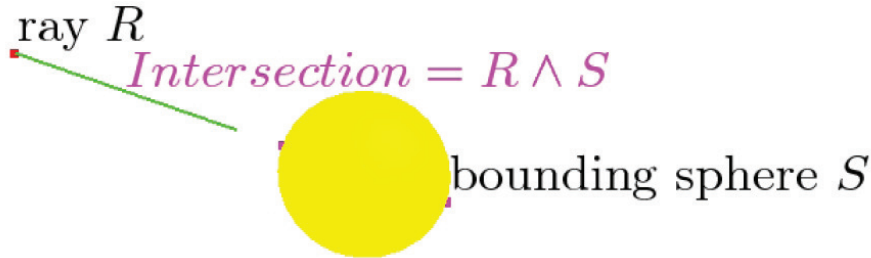


Fig. 1 Spheres and lines are basic entities of geometric algebra to compute with. Operations like the intersection of them are easily expressed with the help of their outer product. The result of the intersection of a ray and a (bounding) sphere is another geometric entity, the point pair of the two 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. A lot of 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 1, this can be easily expressed with the help of the outer product of these two geometric entities.

Geometric algebra is based on the work of Hermann Grassmann (see the conference [21] celebrating his 200th birthday in 2009) and William Clifford ([5], [6]). Pioneering work has been done by David Hestenes, who first applied geometric algebra to problems in mechanics and physics [13] [12].

The first time geometric algebra was introduced to a wider Computer Graphics audience, was through a couple of courses at the SIGGRAPH conferences 2000 and 2001 (see [18]) and later at the Eurographics [14]. Researchers at the University of Cambridge, UK have applied geometric algebra to a number of graphics related projects. Geomerics [1] is a start-up company in Cambridge specializing in simulation software for physics and lighting, which presented its new technology allowing real-time radiosity in videogames utilizing commodity graphics processing hardware. The technology is based on geometric algebra wavelet technology. Researchers at the University of Amsterdam, the Netherlands, are applying their fundamental research on geometric algebra to 3D computer vision, to ray tracing and on the efficient software implementation of geometric algebra. Researchers from Guadalajara, Mexico are primarily dealing with the application of geometric algebra in the field of computer vision, robot vision and kinematics. They are using geometric algebra for instance for tasks like visual guided grasping, camera self-localization and reconstruction of shape and motion. Their methods for geometric

neural computing are used for tasks like pattern recognition ([3]). Registration, the task of finding correspondences between two point sets, is solved based on geometric algebra methods in [23]. Some of their kinematics algorithms are dealing with inverse kinematics, fixation and grasping as well as with kinematics and differential kinematics of binocular robot heads. At the University of Kiel, Germany, researchers are applying geometric algebra to robot vision and pose estimation [24]. They also do some interesting research dealing for instance with neural networks based on geometric algebra ([4]). In addition to these examples there are many other applications like geometric algebra fourier transforms for the visualization and analysis of vector fields [8] or classification and clustering of spatial patterns with geometric algebra [22] showing the wide area of possibilities of advantageously using this mathematical system in engineering applications.

1.1 Benefits of geometric algebra

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

- **Unification of mathematical systems**

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 like vector algebra, complex analysis, quaternions or Plücker coordinates. Table 1, for instance, describes how complex num-

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

bers 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 representing the scalar and imaginary part of complex numbers.

Other examples are Plücker coordinates based on the description of lines in conformal geometric algebra (see section 1.2) or quaternions as to be identified in Figure 4 with their imaginary units.

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	$L = \pi_1 \wedge \pi_2$

- **Uniform handling of different geometric primitives**

Conformal geometric algebra, the geometric algebra of conformal space we focus on, is able to easily treat different geometric objects. Table 2 presents the representation of points, lines, circles, spheres and planes as the same entities algebraically. Consider the spheres of Figure 2, for instance. These spheres are

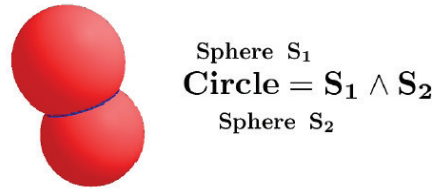


Fig. 2 Spheres and circles are basic entities of geometric algebra. Operations like the intersection of two spheres are easily expressed.

simply represented by

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

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

$$Z = S_1 \wedge S_2 \quad (2)$$

This way of computing with geometric algebra clearly benefits computer graphics applications.

- **Simplified geometric operations**

Geometric operations like rotations, translations (see [14]) and reflections can be easily treated within the algebra. There is no need to change the way of describing them with other approaches (vector algebra, for instance, additionally needs matrices in order to describe transformations).

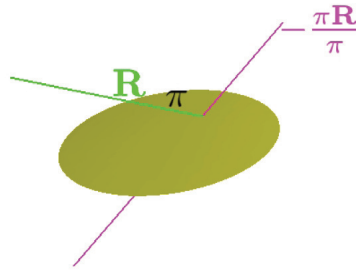


Fig. 3 The ray R is reflected from the plane π computing $-\frac{\pi R}{\pi}$.

Figure 3 visualizes the reflection of the ray R from one plane

$$\pi = \mathbf{n} + de_{\infty} \quad (3)$$

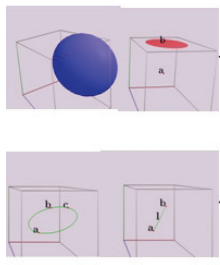
(see Table 2). The reflected line, drawn in magenta,

$$\mathbf{R}_{reflected} = -\frac{\pi R}{\pi} \quad (4)$$

is computed with the help of the reflection operation including the reflection object as well as the object to be reflected.

- **More efficient implementations**

Geometric algebra as a mathematical language suggests a clearer structure and greater elegance in understanding methods and formulae. But, what about the runtime performance for derived algorithms? In section 1.3 we present a dramatically improved optimization approach based on Gaalop [17]. In section 1.3, we will see that geometric algebra inherently has a large potential for creating optimizations leading to more highly efficient implementations especially for parallel platforms.



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

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \\ \dots & & \end{pmatrix}$$

Fig. 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.

1.2 Conformal geometric algebra

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.

Blades are the basic computational elements and the basic geometric entities of geometric algebras. The 5D conformal geometric algebra 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. Table 4 lists all the 32 blades of conformal geometric algebra. The indices indicate 1: scalar, 2...6: vector, 7...16: bivector, 17...26: trivector, 27...31: quadvector, 32: pseudoscalar.

A point $P = x_1 e_1 + x_2 e_2 + x_3 e_3 + \frac{1}{2} \mathbf{x}^2 e_\infty + e_0$ (see Table 2), for instance, can be written in terms of a multivector as the following linear combination of blades $b[i]$

$$P = x_1 * b[2] + x_2 * b[3] + x_3 * b[4] + \frac{1}{2} \mathbf{x}^2 * b[5] + b[6] \quad (5)$$

with multivector indices according to Table 4.

Figure 4 describes some interpretations of the 32 basis blades of conformal geometric algebra. Scalars like the number π are grade 0 entities. They can be combined with the blade representing the imaginary unit i to complex numbers or with the blades representing the imaginary units i, j, k to quaternions. Since quaternions describe rotations, this kind of transformation can be handled within the algebra. Geometric objects like spheres, planes, circles and lines can be represented as vectors and bivectors.

Table 3 lists the two representations of the conformal geometric entities. The inner product null space IPNS and the outer product null space OPNS [20] are dual to each other. While Table 2 already presented the IPNS representation of spheres and planes, they can be described also with the outer product of 4 points being part of them. In the case of a plane one of these 4 points is the point at infinity e_∞ . Circles can be described with the help of the outer product of 3 conformal points lying on the circle or as the intersection of two spheres. Lines can be described with the help

Table 3 The extended list of the two representations of the conformal geometric entities. The IPNS representation as described in Table 2 have also an OPNS representation, which are dual to each other (indicated by the star symbol). In the OPNS representation the geometric objects are described with the help of the outer product of conformal points that are part of the objects, for instance lines as the outer product of two points and the point at infinity.

entity	IPNS representation	OPNS representation
Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	
Sphere	$S = P - \frac{1}{2}r^2 e_\infty$	$S^* = P_1 \wedge P_2 \wedge P_3 \wedge P_4$
Plane	$\pi = \mathbf{n} + d e_\infty$	$\pi^* = P_1 \wedge P_2 \wedge P_3 \wedge e_\infty$
Circle	$Z = S_1 \wedge S_2$	$Z^* = P_1 \wedge P_2 \wedge P_3$
Line	$L = \pi_1 \wedge \pi_2$	$L^* = P_1 \wedge P_2 \wedge e_\infty$
Point Pair	$Pp = S_1 \wedge S_2 \wedge S_3$	$Pp^* = P_1 \wedge P_2$

of the outer product of 2 points and the point at infinity e_∞ or with the help of the outer product of 2 planes (i.e. intersection in IPNS representation). An alternative expression is

$$L = \mathbf{u}e_{123} + \mathbf{m} \wedge e_\infty, \quad (6)$$

with the 3D pseudoscalar $e_{123} = e_1 \wedge e_2 \wedge e_\infty$, the two 3D points \mathbf{a}, \mathbf{b} on the line, $\mathbf{u} = \mathbf{b} - \mathbf{a}$ as 3D direction vector, and $\mathbf{m} = \mathbf{a} \times \mathbf{b}$ as the 3D moment vector (relative to origin). The corresponding six Plücker coordinates (components of \mathbf{u} and \mathbf{m}) are (see Figure 5)

$$(\mathbf{u} : \mathbf{m}) = (u_1 : u_2 : u_3 : m_1 : m_2 : m_3). \quad (7)$$

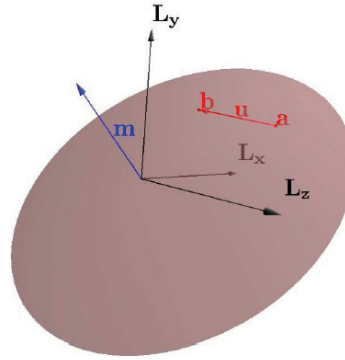


Fig. 5 The line L through the 3D points a , b and the visualization of its 6D Plücker parameters based on the two 3D vectors u and m of equation (7).

1.3 Computational efficiency of geometric algebra using Gaalop

Because of its generality geometric algebra needs some optimizations for efficient implementations.

Gaigen [10] is a geometric algebra code generator developed at the university of Amsterdam (see [7] and [9]). The philosophy behind Gaigen 2 is based on two ideas: generative programming and specializing for the structure of geometric algebra. Please find some benchmarks comparing Gaigen 2 with other pure software solutions as well as comparing five models of 3D Euclidean geometry in a ray tracing application ([9] and [11]).

Gaalop [17] combines the advantages of software optimizations and the adaptability on different parallel platforms. As an example, an inverse kinematics algorithm of a computer animation application was investigated [15]. With the optimization approach of Gaalop the software implementation became three times faster and with a hardware implementation about 300 times faster [16] (3 times by software optimization and 100 times by additional hardware optimization) than the conventional software implementation.

Figure 6 shows an overview over the architecture of Gaalop. Its input is a geometric algebra algorithm written in CLUCalc [19], a system for the visual development of geometric algebra algorithms. Via symbolic simplification it is transformed into an intermediate representation (IR) that can be used for the generation of different output formats. Gaalop supports sequential platforms with the automatic generation of C and JAVA code while its main focus is on supporting parallel platforms like reconfigurable hardware as well as modern accelerating GPUs.

Gaalop uses the symbolic computation functionality of Maple (using the Open Maple interface and a library for geometric algebras [2]) in order to optimize a geometric algebra algorithm. It computes the coefficients of the desired multivector symbolically, returning an efficient implementation depending just on the input variables.

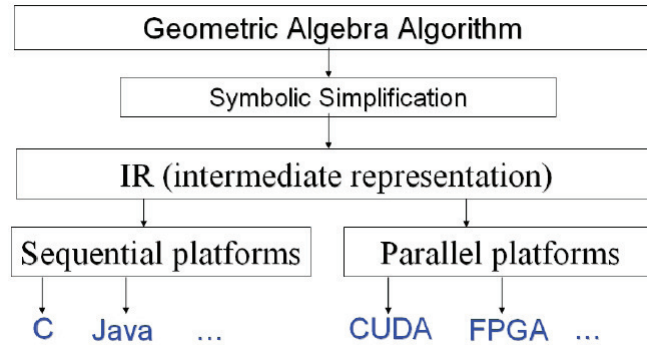


Fig. 6 Architecture of Gaalop

As an example, the following CLUCalc code computes the intersection circle C of two spheres $S1$ and $S2$ according to Figure 2.

```

P1 = x1*e1 +x2*e2 +x3*e3
    +1/2*(x1*x1+x2*x2+x3*x3)*einf +e0;

P2 = y1*e1 +y2*e2 +y3*e3
    +1/2*(y1*y1+y2*y2+y3*y3)*einf +e0;

S1 = P1 - 1/2 * r1*r1 * einf;
S2 = P2 - 1/2 * r2*r2 * einf;

?C = S1 ^ S2;
  
```

See Table 2 for the computation of the conformal points $P1$ and $P2$, the spheres $S1$ and $S2$ as well as the resulting circle based on the outer product of the two spheres.

The resulting C code generated by Gaalop for the intersection circle C is as follows and depends only on the variables $x1, x2, x3, y1, y2, y3, r1$ and $r2$ for the 3D center points and radii:

```

float C [32] = {0.0};

C[7] = x1*y2-x2*y1; C[8] = x1*y3-x3*y1;

C[9] = -0.5*y1*x1*x1-0.5*y1*x2*x2
    -0.5*y1*x3*x3+0.5*y1*r1*r1
    +0.5*x1*y1*y1+0.5*x1*y2*y2
    +0.5*x1*y3*y3-0.5*x1*r2*r2;

C[10] = -y1+x1;

C[11] = -x3*y2+x2*y3;
  
```


$$\begin{aligned} C[12] = & -0.5*y2*x1*x1-0.5*y2*x2*x2 \\ & -0.5*y2*x3*x3+0.5*y2*r1*r1 \\ & +0.5*x2*y1*y1+0.5*x2*y2*y2 \\ & +0.5*x2*y3*y3-0.5*x2*r2*r2; \end{aligned}$$

$$C[13] = -y2+x2;$$

$$\begin{aligned} C[14] = & -0.5*y3*x1*x1-0.5*y3*x2*x2 \\ & -0.5*y3*x3*x3+0.5*y3*r1*r1 \\ & +0.5*x3*y1*y1+0.5*x3*y2*y2 \\ & +0.5*x3*y3*y3-0.5*x3*r2*r2; \end{aligned}$$

$$C[15] = -y3+x3;$$

$$\begin{aligned} C[16] = & -0.5*y3*y3+0.5*x3*x3 \\ & +0.5*x2*x2+0.5*r2*r2 \\ & -0.5*y1*y1-0.5*y2*y2 \\ & +0.5*x1*x1-0.5*r1*r1; \end{aligned}$$

In a nutshell, Gaalop always computes optimized 32-dimensional multivectors. Since a circle is described with the help of a bivector, only the blades 7 to 16 (see Table 4) are used. As you can see, all the corresponding coefficients of this multivector are very simple expressions with basic arithmetic operations.

Table 4 The 32 blades of the 5D conformal geometric algebra

index	blade	grade	index	blade	grade
1	1	0	17	$e_1 \wedge e_2 \wedge e_3$	3
2	e_1	1	18	$e_1 \wedge e_2 \wedge e_\infty$	3
3	e_2	1	19	$e_1 \wedge e_2 \wedge e_0$	3
4	e_3	1	20	$e_1 \wedge e_3 \wedge e_\infty$	3
5	e_∞	1	21	$e_1 \wedge e_3 \wedge e_0$	3
6	e_0	1	22	$e_1 \wedge e_\infty \wedge e_0$	3
7	$e_1 \wedge e_2$	2	23	$e_2 \wedge e_3 \wedge e_\infty$	3
8	$e_1 \wedge e_3$	2	24	$e_2 \wedge e_3 \wedge e_0$	3
9	$e_1 \wedge e_\infty$	2	25	$e_2 \wedge e_\infty \wedge e_0$	3
10	$e_1 \wedge e_0$	2	26	$e_3 \wedge e_\infty \wedge e_0$	3
11	$e_2 \wedge e_3$	2	27	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty$	4
12	$e_2 \wedge e_\infty$	2	28	$e_1 \wedge e_2 \wedge e_3 \wedge e_0$	4
13	$e_2 \wedge e_0$	2	29	$e_1 \wedge e_2 \wedge e_\infty \wedge e_0$	4
14	$e_3 \wedge e_\infty$	2	30	$e_1 \wedge e_3 \wedge e_\infty \wedge e_0$	4
15	$e_3 \wedge e_0$	2	31	$e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	4
16	$e_\infty \wedge e_0$	2	32	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	5

References

1. The homepage of geomerics ltd. Available at <http://www.geomerics.com>
2. Ablamowicz, R., Fauser, B.: Clifford/bigebra, a maple package for clifford (co)algebra computations (2009). ©1996-2009, RA&BF
3. Bayro-Corrochano, E., Vallejo, R., Arana-Daniel, N.: Geometric preprocessing, geometric feedforward neural networks and Clifford support vector machines for visual learning. Special issue of *Journal Neurocomputing* **67**, 54–105 (2005)
4. Buchholz, S., Hitzer, E.M.S., Tachibana, K.: Optimal learning rates for Clifford neurons. In: *International Conference on Artificial Neural Networks*, vol. 1, pp. 864–873. Porto, Portugal (2007). 9-13
5. Clifford, W.K.: Applications of grassmann's extensive algebra. In: R. Tucker (ed.) *Mathematical Papers*, pp. 266–276. Macmillian, London (1882)
6. Clifford, W.K.: On the classification of geometric algebras. In: R. Tucker (ed.) *Mathematical Papers*, pp. 397–401. Macmillian, London (1882)
7. Dorst, L., Fontijne, D., Mann, S.: *Geometric Algebra for Computer Science, An Object-Oriented Approach to Geometry*. Morgan Kaufman (2007)
8. Ebling, J.: Clifford fourier transform on vector fields. *IEEE Transactions on Visualization and Computer Graphics* **11**(4), 469–479 (2005). IEEE member Scheuermann, Geric
9. Fontijne, D.: Efficient implementation of geometric algebra. Ph.D. thesis, University of Amsterdam (2007)
10. Fontijne, D., Bouma, T., Dorst, L.: Gaigen: A geometric algebra implementation generator. Available at <http://www.science.uva.nl/ga/gaigen> (2005)
11. Fontijne, D., Dorst, L.: Modeling 3D euclidean geometry. *IEEE Computer Graphics and Applications* **23**(2), 68–78 (2003)
12. Hestenes, D.: *New Foundations for Classical Mechanics*. Dordrecht (1986)
13. Hestenes, D., Sobczyk, G.: *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*. Dordrecht (1984)
14. Hildenbrand, D., Fontijne, D., Perwass, C., Dorst, L.: Tutorial geometric algebra and its application to computer graphics. In: *Eurographics conference Grenoble* (2004)
15. Hildenbrand, D., Fontijne, D., Wang, Y., Alexa, M., Dorst, L.: Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra. In: *Eurographics conference Vienna* (2006)
16. Hildenbrand, D., Lange, H., Stock, F., Koch, A.: Efficient inverse kinematics algorithm based on conformal geometric algebra using reconfigurable hardware. In: *GRAPP conference Madeira* (2008)
17. Hildenbrand, D., Pitt, J.: The Gaalop home page. Available at <http://www.gaalop.de> (2008)
18. Naeve, A., Rockwood, A.: Course 53 geometric algebra. In: *Siggraph conference Los Angeles* (2001)
19. Perwass, C.: The CLU home page. Available at <http://www.clucalc.info> (2005)
20. Perwass, C.: *Geometric Algebra with Applications in Engineering*. Springer (2009)
21. Petsche, H.J.: The Grassmann Bicentennial Conference home page. Available at <http://www.uni-potsdam.de/u/philosophie/grassmann/Papers.htm> (2009)
22. Pham, M.T., Tachibana, K., Hitzer, E.M.S., Yoshikawa, T., Furuhashi, T.: Classification and clustering of spatial patterns with geometric algebra. In: *AGACSE conference Leipzig* (2008)
23. Reyes-Lozano, L., Medioni, G., Bayro-Corrochano, E.: Registration of 3d points using geometric algebra and tensor voting. *Journal of Computer Vision* **75**(3), 351–369 (2007)
24. Rosenhahn, B., Sommer, G.: Pose estimation in conformal geometric algebra. *Journal of Mathematical Imaging and Vision* **22**, 27–70 (2005)