福井大学 工学部 研究報告 第 53 巻 第 1 号 2005 年 3 月
Mem. Fac. Eng. Univ. Fukui, Vol. 53, No. 1 (March 2005)

The GeometricAlgebra Java Package – Novel Structure Implementation of 5D Geometric Algebra $R_{4,1}$ for Object Oriented Euclidean Geometry, Space-Time Physics and Object Oriented Computer Algebra

Eckhard MS HITZER[*] and Ginanjar UTAMA[**]

(Received February 18, 2005)

This paper first briefly reviews the algebraic background of the conformal (homogeneous) model of Euclidean space in Clifford geometric algebra $\mathbf{R}_{4,1}= Cl(4,1)$, concentrating on the subalgebra structure. The subalgebras include space-time algebra (STA), Dirac and Pauli algebras, as well as real and complex quaternion algebras, etc. The concept of the *Horosphere* is introduced along with the definition of subspaces that intuitively correspond to three dimensional Euclidean geometric objects. Algebraic expressions for the motions of these objects and their set theoretic operations are given. It is shown how 3D Euclidean information on positions, orientations and radii can be extracted.

The second main part of the paper concentrates on the GeometricAlgebra Java package implementation of the Clifford geometric algebra $\mathbf{R}_{4,1} = Cl(4,1)$ and the homogeneous model of 3D Euclidean space. Details are exemplified by looking at the structure and code of the basic MultiVector class and of the 3D Euclidean object model class Sphere. Finally code optimization issues and the ongoing open source project implementation are discussed.

Key Words **:** Clifford Geometric Algebra, Java Multivector Software, Object Oriented Euclidean Geometry, Space-Time Algebra, Computer Algebra

1. Introduction

The Clifford geometric algebra of three dimensional (3D) Euclidean space nicely encodes the algebra of 3D subspaces, providing geometric multivector product expressions of rotations and set theoretic operations [1]. But in this framework line and plane subspaces always contain the origin.

The homogeneous (conformal) model of 3D Euclidean space in the Clifford geometric algebra $\mathbf{R}_{4,1}$ provides a way out. Here positions of points, lines and planes, etc. off the origin can be naturally encoded. Other advantages are the unified treatment of rotations and translations and ways to encode point pairs, circles and spheres. The creation of such elementary geometric objects simply occurs by algebraically joining a minimal number of points in the object subspace. The resulting multivector expressions completely encode in their components positions, orientations and radii. The geometric algebra $\mathbf{R}_{4,1}$ can be intuitively pictured as the algebra of origin, Euclidean 3D space and infinity, where origin and infinity are represented by additional linearly independent null-vectors.
This algebra seems most suitable for applications in computer graphics, robotics and other fields [2], [3].

This paper therefore first gives a brief review of the basic concepts of the geometric algebra $\mathbf{R}_{4,1}$, its subalgebra structure and the socalled *Horosphere* [7]. Explicit details for the construction of fundamental geometric objects in this model are given, detailing how the 3D geometric information can be extracted. We further explain how the simple multivector representations of these objects can be manipulated in order to move them in three dimensions and to express set theoretic operations of union (join), intersection (meet), projections and rejections.

This algebraic encoding of geometric objects and their manipulations strongly suggests an object oriented software implementation. This would allow computers to calculate with

[*] Dept. of Physical Engineering
[**] Dept. of Engineering Physics, Institute of Technology Bandung, Indonesia

this algebra and provide programmers with the means to most suitably represent fundamental geometric objects, their 3D properties and ways (methods) to manipulate these objects. This happens on a higher algebraic level, so that the programmer actually is freed of the need to first investigate suitable less intuitive matrix representations.

The implementation chosen here is in the object oriented, platform independent programming language Java in the form of a Java package named *GeometricAlgebra*. This Java implementation of $\mathbf{R}_{4,1}$ *automatically includes* the implementations of a wide range of mathematically and physically important lower dimensional algebras and geometric algebras.

Finally, the reasons for this choice of implementation are discussed along with details of the code and perspectives for further improvement and development.

2. The Conformal Geometric Algebra *Cl*(4,1) of Origin, Euclidean Three Space and Infinity

2.1 Geometric Algebra of 3D Euclidean Space

By reverting back to Clifford's original expression *geometric algebra* we want to emphasize the geometric interpretation of this algebra. Let us assume an orthonormal vector basis for the real three dimensional Euclidean vector space $\mathbf{R}^3 = \mathbf{R}^{3,0}$ (NB: upper index.)

$$\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\} \tag{1}$$

Clifford's bilinear and associative geometric product [4] - [6] of vectors

$$\mathbf{ab} = \mathbf{a} \lrcorner \mathbf{b} + \mathbf{a} \wedge \mathbf{b} = \mathbf{a} \llcorner \mathbf{b} + \mathbf{a} \wedge \mathbf{b} = \mathbf{a} * \mathbf{b} + \mathbf{a} \wedge \mathbf{b} \tag{2}$$

generates the geometric algebra $\mathbf{R}_3 = \mathbf{R}_{3,0}$ (NB: lower index.)
The first term in each of the three expressions for the geometric product given above is a symmetric grade zero scalar and fully corresponds to the familiar scalar product of vectors (of algebraic grade one). The second term is an antisymmetric bivector as known from Grassmann's 19th century multivector algebra (extension theory). It represents the oriented parallelogram area which is spanned by the two vector factors in the *outer product* ($\wedge$), algebraically it has grade two. The three different expressions for the first term indicate, that *left contraction* ($\lrcorner$) and *right contraction* ($\llcorner$) and the *scalar product* ($*$) of vectors all yield the same scalar.

The left contraction of two simple multivectors (blades) A_k, B_l with grades k, l ($0 \leqq k,l \leqq$ max, max = dimension of the vector space, e.g. max = 3 for $\mathbf{R}^3$), respectively is defined as

$$A_k \lrcorner B_l = \langle A_k B_l \rangle_{l-k} , \tag{2'}$$

where the angular bracket $\langle \rangle_{l-k}$ means to extract only the grade l-k part from the full geometric product $A_k B_l$. Similarly the right contraction of two simple multivectors A_k, B_l is defined as

$$A_k \llcorner B_l = \langle A_k B_l \rangle_{k-l} . \tag{2''}$$

The scalar product of A_k and B_l is defined as

$$A_k * B_l = \langle A_k B_l \rangle = \langle A_k B_l \rangle_0 . \tag{2'''}$$

$\mathbf{R}_3$ is $2^3 = 8$ dimensional with a basis of scalars, vectors, bivectors and trivectors

$$\{1, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{i}_1 = \mathbf{e}_2\mathbf{e}_3, \mathbf{i}_2 = \mathbf{e}_3\mathbf{e}_1, \mathbf{i}_3 = \mathbf{e}_1\mathbf{e}_2, i = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3\}. \tag{3}$$

Scalars are *dual* (by multiplication with i) to the *pseudoscalar* trivectors (proportional to i) according to

$$i^2 = ii = -1, \tag{4}$$

and vectors are *dual* to bivectors according to

$$i\mathbf{e}_1 = \mathbf{i}_1, \quad i\mathbf{e}_2 = \mathbf{i}_2, \quad i\mathbf{e}_3 = \mathbf{i}_3. \tag{5}$$

We can therefore rewrite the basis of Eq. (3) as a "complex" scalar and vector basis

$$\{1, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, i\mathbf{e}_1 = \mathbf{i}_1, i\mathbf{e}_2 = \mathbf{i}_2, i\mathbf{e}_3 = \mathbf{i}_3, i = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3\}, \tag{3'}$$

because of Eq. (4). Or we can rewrite it as a "complex" quaternion basis

$$\{1, \mathbf{e}_1 = -i\mathbf{i}_1, \mathbf{e}_2 = -i\mathbf{i}_2, \mathbf{e}_3 = -i\mathbf{i}_3, \mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, i = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3\}. \tag{3''}$$

2.2 Basis and Subspaces of Conformal 5D Space of Origin, 3D Euclidean Space and Infinity

By introducing (similar to projective geometry) a linearly independent null-vector to represent the origin $\check{\mathrm{n}}$ and a second linearly independent null vector $\mathbf{n}$ to represent infinity, we extend the basis of Eq. (1) to the five dimensional basis of the vector space $\mathbf{R}^{4,1}$

$$\{\check{\mathrm{n}}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{n}\}. \tag{6}$$

As we see from this basis, $\mathbf{R}^{4,1}$ has two important subspaces, the three dimensional Euclidean space and its orthogonal complement, the Minkowski plane $\mathbf{R}^{1,1}$ spanned by the two extra dimensions for origin and infinity

$$\mathbf{R}^{1,1} = \mathrm{span}[\check{\mathrm{n}}, \mathbf{n}]. \tag{7}$$

With the basis transformation

$$\mathbf{e}_0 = (\mathbf{n} + 2\check{\mathrm{n}})/2, \quad \mathbf{e}_4 = (\mathbf{n} - 2\check{\mathrm{n}})/2, \tag{8}$$

and the inverse transformation

$$\mathbf{n} = \mathbf{e}_0 + \mathbf{e}_4, \quad \check{\mathrm{n}} = (\mathbf{e}_0 - \mathbf{e}_4)/2, \tag{9}$$

we can introduce an orthonormal non-null basis for $\mathbf{R}^{1,1}$ with vectors of positive and negative square

$$\{\mathbf{e}_0, \mathbf{e}_4\}, \quad \mathbf{e}_0^2 = -1, \ \mathbf{e}_4^2 = +1. \tag{10}$$

So we can either use for $\mathbf{R}^{4,1}$ the basis of Eq. (6) or

$$\{\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4\}. \tag{11}$$

These two basis are linked by the transformations of Eqs. (8) and (9). Usually it is convenient to work in the basis of Eq. (6), but sometimes the work with the non-null basis of Eq. (11) can be of advantage (e.g. for factorization).

2.3 Important Subalgebras

Apart from the Euclidean subalgebra $\mathbf{R}_3$ mentioned in subsection 2.1, the conformal geometric algebra $\mathbf{R}_{4,1}$ of the 5D vector space $\mathbf{R}^{4,1}$ has further three subalgebras, which are of particular importance for object oriented Euclidean geometry: one is isomorphic to the *complex numbers*, one to the *quaternions* and finally the subalgebra of the *Minkowski plane*. These three subalgebras also have major applications in physics and computer algebra. A more comprehensive survey of the rich subalgebra structure of the conformal geometric algebra $\mathbf{R}_{4,1}$ is given in section 4.1.

2.3.1 Subalgebra of Scalars and Pseudoscalars

Scalars ($\in\mathbf{R}$) and 5D pseudoscalars, proportional to

$$I = \mathbf{e}_0\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3\mathbf{e}_4 = -i\ \mathbf{e}_0\mathbf{e}_4 = i\mathrm{N},\ I^2 = -1, \qquad (12)$$

with

$$N = -iI = -\mathbf{e}_0\mathbf{e}_4 = \mathbf{n}\wedge\check{\mathbf{n}},\ N^2 = +1, \qquad (13)$$

form according to Eq. (12) under the geometric product a subalgebra of $\mathbf{R}_{4,1}$, which is isomorphic to complex numbers. Compare also the multiplication Table 1.

Table 1 Multiplication table of scalar-pseudoscalar subalgebra. (Products of left factor from left column and right factor from top row.)

	1	I
1	1	I
I	I	-1

2.3.2 Subalgebra Isomorphic to Quaternions

The geometric algebra $\mathbf{R}_3$ of the Euclidean subspace $\mathbf{R}^3$ is also a subalgebra of $\mathbf{R}_{4,1}$ and so is its even subalgebra, which has a basis, that consists of real scalars and the three unit bivectors that represent the oriented unit side faces of a unit cube oriented with its edges along the three vectors of the Euclidean basis of Eq. (1):

$$\{1, \mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3\} \qquad (14)$$

The multiplication table (Table 2) clearly shows the isomorphism of this subalgebra with Hamilton's algebra of quaternions.

Table 2 Multiplication table of subalgebra isomorphic to quaternions. (Products of left factor from left column and right factor from top row.)

	1	$\mathbf{i}_1$	$\mathbf{i}_2$	$\mathbf{i}_3$
1	1	$\mathbf{i}_1$	$\mathbf{i}_2$	$\mathbf{i}_3$
$\mathbf{i}_1$	$\mathbf{i}_1$	-1	$-\mathbf{i}_3$	$\mathbf{i}_2$
$\mathbf{i}_2$	$\mathbf{i}_2$	$\mathbf{i}_3$	-1	$-\mathbf{i}_1$
$\mathbf{i}_3$	$\mathbf{i}_3$	$-\mathbf{i}_2$	$\mathbf{i}_1$	-1

2.3.3 Subalgebra of the Minkowski Plane

The subalgebra $\mathbf{R}_{1,1}$ of the Minkowski plane subspace $\mathbf{R}^{1,1}$ has the following $2^2 = 4$ dimensional basis

$$\{1, \mathbf{n}, \check{\mathbf{n}}, \mathrm{N}\} \qquad (15)$$

of scalars, vectors and the bivector N (N fully characterizes the Minkowski plane subspace). Table 3 shows the corresponding multiplication table.

Table 3 Multiplication table of the Minkowski plane subalgebra. (Products of left factor from left column and right factor from top row.)

	1	$\mathbf{N}$	$\check{\mathbf{n}}$	N
1	1	$\mathbf{N}$	$\check{\mathbf{n}}$	N
$\mathbf{n}$	$\mathbf{n}$	0	-1+N	$\mathbf{n}$
$\check{\mathbf{n}}$	$\check{\mathbf{n}}$	-1-N	0	$-\check{\mathbf{n}}$
N	N	$-\mathbf{n}$	$\check{\mathbf{n}}$	1

2.4 Basis of the Conformal Geometric Algebra

We are now in a position to write down the complete $2^5 = 32$ element basis of the conformal geometric algebra $\mathbf{R}_{4,1}$. It contains grade by grade scalars, vectors, bivectors and the dual elements of trivectors, quadrivectors and pseudoscalars.

grade

(0) 1

(1) $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{n}, \check{\mathbf{n}}$

(2) $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, \mathbf{e}_1\mathbf{n}, \mathbf{e}_2\mathbf{n}, \mathbf{e}_3\mathbf{n}, \mathbf{e}_1\check{\mathbf{n}}, \mathbf{e}_2\check{\mathbf{n}}, \mathbf{e}_3\check{\mathbf{n}}, \mathrm{N}$

$$\text{---------------------------------------} \qquad (16)$$

(3) $\mathrm{I}\,\mathbf{i}_1, \mathrm{I}\mathbf{i}_2, \mathrm{I}\mathbf{i}_3, \mathbf{i}_1\mathbf{n}, \mathbf{i}_2\mathbf{n}, \mathbf{i}_3\mathbf{n}, \mathbf{i}_1\check{\mathbf{n}}, \mathbf{i}_2\check{\mathbf{n}}, \mathbf{i}_3\check{\mathbf{n}}, i = \mathrm{IN}$

(4) $\mathbf{i}_1\mathrm{N}, \mathbf{i}_2\mathrm{N}, \mathbf{i}_3\mathrm{N}, \mathrm{I}\,\mathbf{n}, \mathrm{I}\check{\mathbf{n}}$

(5) I

All elements of Eq. (16) below the horizontal line are dual (by multiplication with I) to elements above the line. This fact and the subalgebra structure of $\mathbf{R}_{4,1}$ explained in sections 2.3.1 to 2.3.3 yield a very neat notation for general multivectors in the 32 dimensional conformal geometric algebra.

2.5 Three Level Approach

The three levels which we use to categorize general multivectors in the conformal geometric algebra $\mathbf{R}_{4,1}$ correspond to the subalgebras of complex numbers, quaternions and the Minkowski plane algebra $\mathbf{R}_{1,1}$. We can now write each general multivector as a linear combination of four complex quaternions

$$m = q + q_n\mathbf{n} + q_{\check{n}}\check{\mathbf{n}} + q_N\mathrm{N}. \qquad (17)$$

Each of the four complex quaternions q, q_n, $q_{\check{n}}$, q_N is in turn a complex linear combination of two real quaternions. For example the complex quaternion

$$q_n = q_{n,r} + I\, q_{n,i} \quad (18)$$

where the indexes r and i signify the *real* and *imaginary* parts respectively. Every real quaternion can be written explicitly as a linear combination of the basis elements Eq. (14), e.g.

$$q_{n,i} = q_{n,i,0} + q_{n,i,1}\, \mathbf{i}_1 + q_{n,i,2}\, \mathbf{i}_2 + q_{n,i,3}\, \mathbf{i}_3 \quad (19)$$

where the word *real* of real quaternion means, that the four scalar coefficients are all real

$$q_{n,i,0}, q_{n,i,1}, q_{n,i,2}, q_{n,i,3} \in \mathbf{R} \quad (20)$$

This three level approach makes the programming indeed very modular and well structured. Eqs. (18) and (19) also show, that each complex quaternion q, q_n, $q_{\check{n}}$ and q_N is isomorphic to the geometric algebra of three-dimensional Euclidean space of Eq. (3), because of Eq. (12): I = *i*N.

3 Conformal (Homogeneous) Model of Euclidean Space

3.1 The Horosphere

The notion of *light cone* in Minkowski space $\mathbf{R}^{3,1}$ as the set of all light rays emanating from one point is familiar from special relativity. All vectors on the light cone square to zero. The *Horosphere* is a 3D section of the 4D light cone of the space $\mathbf{R}^{4,1}$ keeping the component in the direction of ň equal to one. It was introduced by F.A. Wachter (1792-1817), an assistant of Gauss [7], [10].

A point **x** in Euclidean space can be specified in terms of three orthonormal basis vectors of Eq. (1) as

$$\mathbf{x} = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + x_3\mathbf{e}_3 \quad (21)$$

Its one-to-one corresponding conformal point X on the Horosphere is defined by adding an infinity and an origin part

$$X = \mathbf{x} + 1/2\, x^2\mathbf{n} + \check{n} \quad (22)$$

with $x^2 = \mathbf{x}^2$, with the result that $X^2 = 0$.

3.2 Geometric Product

The geometric product of Eq. (2) of two conformal points

$$P_1 = \mathbf{p}_1 + 1/2\, p_1^2\mathbf{n} + \check{n}, \qquad P_2 = \mathbf{p}_2 + 1/2\, p_2^2\mathbf{n} + \check{n} \quad (23)$$

yields

$$P_1P_2 = P_1 * P_2 + P_1 \wedge P_2, \quad (24)$$

with the scalar contraction part

$$P_1 * P_2 = -1/2\ (\mathbf{p}_1 - \mathbf{p}_2)^2\ . \quad (25)$$

The contraction part therefore directly corresponds to the squared Euclidean distance. The second term on the right hand side of Eq. (24) fully corresponds to the pair of conformal points P_1, P_2 (or of Euclidean points $\mathbf{p}_1$, $\mathbf{p}_2$), which can be fully extracted from $P_1 \wedge P_2$, as explained in section 3.6.1.

3.3 Subspaces, Joining

Similar to the projective definition of lines with the help of the outer product by Grassmann, we have the following two useful propositions in geometric algebra of a real n-dimensional linear vector space [6]. For vectors **x**, $\mathbf{p}_1$, $\mathbf{p}_2$, … $\mathbf{p}_r$ in that vector space

$$\{\mathbf{p}_1, \mathbf{p}_2, \ldots \mathbf{p}_r\}\ (r \le n) \text{ linearly independent}$$
$$\Leftrightarrow \quad \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \ldots \wedge \mathbf{p}_r \neq 0 \quad (26)$$

and

$$\mathbf{x} \in \text{span}[\mathbf{p}_1, \mathbf{p}_2, \ldots \mathbf{p}_r] \quad (r \le n)$$
$$\Leftrightarrow \quad \mathbf{x} \wedge \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \ldots \wedge \mathbf{p}_r = 0 \quad (27)$$

We get the following subspaces of the Horosphere by *joining* general conformal points P_1, P_2, P_3, P_4 and infinity **n** with the outer product of geometric algebra. We list the subspaces in terms of their Euclidean equivalents.[10], [12]

- *Pairs of points* P_1, P_2 form $P_1 \wedge P_2$
- *Circles* through P_1, P_2, P_3 corresponding to $P_1 \wedge P_2 \wedge P_3$
- *Straight lines* through P_1, P_2 and infinity corresponding to $P_1 \wedge P_2 \wedge \mathbf{n}$
- *Spheres* through P_1, P_2, P_3, P_4 corresponding to $P_1 \wedge P_2 \wedge P_3 \wedge P_4$
- *Planes* through P_1, P_2, P_3 and infinity corresponding to $P_1 \wedge P_2 \wedge P_3 \wedge \mathbf{n}$

3.4 Translators and Rotors

It is a major benefit of introducing the extra dimensions of origin ň and infinity **n**, that translations can be implemented like rotations by monomial multivector products. The general form[10] of both transformations for conformal points X is

$$X \rightarrow X' = UX\tilde{U},\ U\tilde{U} = 1. \quad (28)$$

For rotations by an angle ϑ in the Euclidean bivector plane **i** about the origin, U becomes a *rotor*

$$R(\mathbf{i}\vartheta) = \pm \exp(-\mathbf{i}\vartheta/2),\ \tilde{R}(\mathbf{i}\vartheta) = \pm \exp(+\mathbf{i}\vartheta/2). \quad (29)$$

For translations by a 3D Euclidean translation vector **t** the multivector U becomes a *translator*

$$T(\mathbf{t}) = \pm \exp(\mathbf{nt}/2) = \pm (1 + \mathbf{nt}/2),$$
$$\tilde{T}(\mathbf{t}) = \pm (1 - \mathbf{nt}/2), \quad (30)$$

The two terms expansion formula holds because of the null property Eq. (6) of **n**. A rotation by an angle ϑ in the plane **i** about any Euclidean center of rotation **a** is obtained by shifting the center of rotation to and from the origin, before and after the rotation, respectively. For this combined transformation U has simply to be replaced by

$$R'(\mathbf{i}\vartheta, \mathbf{a}) = T(\mathbf{a})\ R(\mathbf{i}\vartheta)\ \tilde{T}(\mathbf{a}) \quad (31)$$

The reverse order [6] translator $\tilde{T}(\mathbf{a})$ first translates the center of rotation **a** by **-a** to the origin. A general combination of a rotation by an angle ϑ in the plane **i** about any Euclidean center of rotation **a** combined with a subsequent translation by a 3D Euclidean vector **t** is finally described by inserting for U in Eq. (28) the *motor* (motion operator)

$$M(\mathbf{i}\vartheta, \mathbf{a}, t) = T(\mathbf{t})\, R'(\mathbf{i}\vartheta, \mathbf{a})\ . \quad (32)$$

3.5 Join, Meet and Projection

The join[6] is the set theoretic union of subspaces. (NB: Some mistakes in [12] are put right here.) From Eq. (27) we see that in geometric algebra (and Grassmann algebra) the basic operation of *join* is the elementary operation of the outer product. This fully applies for joining linearly independent vectors (each representing a one dimensional subspace [11]). Compare the examples of joining two, three and four conformal point (vectors) in section 3.3. In general one can simply take the outer product of the basis vectors of a (sub)space to get a simple join multivector that fully represents that (sub)space according to Eqs. (26) and (27). If follows that for two simple multivectors K, L that represent two disjoint subspaces, the set theoretic union (join J) is also given by the outer product

$$J = K \wedge L . \qquad (33)$$

A different formula applies for J in case that the two subspaces K and L are not disjoint, i.e. that they have a common simple (blade) multivector factor M, that characterizes the set theoretic intersection, called *meet* according to

$$K = K' \wedge M, \quad L = M \wedge L'. \qquad (34)$$

Because M is a simple multivector it has with respect to the geometric product the inverse

$$M^{-1} = M/M^2 \qquad (35)$$

where we also need to assume that M is not a null-multivector, i.e. $M^2 \neq 0$.

Notice that the use of null-vectors in section 3.3 for modeling Euclidean objects does not necessarily mean that the resulting multivectors square to zero. In Table 3 we see that e.g. $N = \mathbf{n} \wedge \check{\mathbf{n}}$ has the square $N^2 = 1$. Another example are the radii of the circles and spheres, which are given by the properly normed squares of the multivectors $P_1 \wedge P_2 \wedge P_3$ and $P_1 \wedge P_2 \wedge P_3 \wedge P_4$ [37]. The meet M of two intersecting spheres is a circle and will therefore not square to zero, if the radius is not zero.

For non-disjoint subspaces, the simple join multivector (blade) can be calculated [8] by

$$J = K \wedge L' = K \wedge (M^{-1} \lrcorner L) . \qquad (36)$$

Knowing the join, we can in turn use it to calculate[8] the meet M by

$$M = (K \lrcorner J^{-1}) \lrcorner L . \qquad (37)$$

In general the sign of the square of the meet M^2 is of great importance [2],[9]. For example the meet of a line and a sphere is a bivector [38]. For $M^2 > 0$ it represents a pair of intersection points, but $M^2 < 0$ means that the line and the sphere are disjoint. Similarly the meet M of two spheres is a trivector. For $M^2 < 0$ it represents the circle of intersection, but $M^2 > 0$ means that the spheres are disjoint. (NB: The signs of M^2 are different for the cases of bivectors and trivectors.) In both cases $M^2 = 0$ encodes a single point of tangential intersection.

Finally the projection [8] of the subspace represented by K onto the subspace represented by L (think e.g. of the projection of a line onto a plane) is given by

$$P_L(K) = (K \lrcorner L^{-1}) \lrcorner L . \qquad (38)$$

Because of the linearity of the projection formula, the simple multivector K can even be replaced by a general inhomogeneous multivector (which can be understood as representing a collection of subspaces).

Complimentary to the projection is the rejection[8] of the subspace represented by K off the subspace represented by L. It defines the subspace of K perpendicular to L. It is given by

$$(K \wedge L^{-1}) \lrcorner L . \qquad (39)$$

3.6 3D Information in Homogeneous Objects

The homogenous multivectors of section 3.3 completely encode positions, directions, moments and radii of the corresponding three dimensional (3D) objects in Euclidean space. An overview of this is given in Table 4. Here we only give a brief summary of important formulas, for more details consult [37].

Table 4 3D information in homogeneous objects. The left column lists the homogeneous multivectors of section 3.3, that represent the geometric objects.

homogeneous object	3D information
point P	position $\mathbf{p}$
point pair $P_1 \wedge P_2$	positions $\mathbf{p}_1$, $\mathbf{p}_2$
line	direction vector, moment bivector
circle	plane bivector, center, radius
plane	plane bivector, location vector
sphere	center, radius

3.6.1 Point and Pair of Points

The (additive) *conformal split* returns the Euclidean position $\mathbf{p}$ of a conformal point P

$$\mathbf{p} = (P \wedge N)N . \qquad (40)$$

Definition (22) shows how to get P in terms of $\mathbf{p}$.

The Euclidean positions $\mathbf{p}_1$, $\mathbf{p}_2$ (without loss of generality: $p_1 = \sqrt{\mathbf{p}_1^2} \leq p_2 = \sqrt{\mathbf{p}_2^2}$) of a pair of points represented by the conformal bivector

$$V_2 = P_1 \wedge P_2 = \mathbf{b} + 1/2\ \mathbf{v}\mathbf{n} - \mathbf{u}\check{\mathbf{n}} - 1/2\ gN \qquad (41)$$

can be fully reconstructed from the various components of V_2. These are the Euclidean bivector $\mathbf{b}$, two Euclidean vectors $\mathbf{u}$, $\mathbf{v}$

(lengths $u = \sqrt{\mathbf{u}^2}$, $v = \sqrt{\mathbf{v}^2}$) and the real scalar g.

$$s = 1/2\, g^2 - \mathbf{u}*\mathbf{v}, \quad t = (s^2 - u^2v^2)^{1/2},$$
$$p_1 = (s+t)^{1/2}/u, \quad p_2 = (s-t)^{1/2}/u, \qquad (42)$$
$$\mathbf{p}_1 = p_1 (p_1^2\mathbf{u} + \mathbf{v}) / |\, p_1^2\mathbf{u} + \mathbf{v}|\,,$$
$$\mathbf{p}_2 = p_2 (p_2^2\mathbf{u} + \mathbf{v}) / |\, p_2^2\mathbf{u} + \mathbf{v}|$$

3.6.2 Lines

Given two conformal points P_1 and P_2 the conformal trivector

$$V_{line} = P_1 \wedge P_2 \wedge \mathbf{n} = \mathbf{m}\mathbf{n} + \mathbf{d}\, N \qquad (43)$$

consists of the Euclidean bivector momentum **m** and the Euclidean direction vector **d** of the line. A Euclidean parametric equation for the line is then

$$\mathbf{x} = (\mathbf{m} + a)\, \mathbf{d}^{-1}\,, \; a \in \mathbf{R}\,. \qquad (44)$$

3.6.3 Cirlces

General conformal trivectors of the form

$$V_3 = P_1 \wedge P_2 \wedge P_3$$
$$= \mathbf{c} \wedge I_c + [1/2\,(r^2 + c^2)\, I_c - \mathbf{c}(\mathbf{c} \lrcorner I_c)]\mathbf{n} + I_c \check{\mathbf{n}} - (\mathbf{c} \lrcorner I_c)\, N, \qquad (45)$$

with Euclidean circle plane bivector

$$I_c = -\{[V_3 + (V_3 * i)\, i] \wedge \mathbf{n}\} N\,, \qquad (46)$$

circle radius

$$r^2 = -V_3^2 / I_c^2 \qquad (47)$$

and Euclidean circle center

$$\mathbf{c} = \mathbf{c}_{\parallel} + \mathbf{c}_{\perp},$$
$$\mathbf{c}_{\parallel} = -[(V_3 \llcorner \mathbf{n}) \llcorner \check{\mathbf{n}}]\, I_c^{-1}\,, \quad \mathbf{c}_{\perp} = -\,(V_3 * i)\; i\; I_c^{-1}. \qquad (48)$$

For the special case that the circle plane includes the origin ($\mathbf{c}_{\perp} = 0$), we get a much simpler expression

$$V_3 = -\,[C - 1/2\, r^2 \mathbf{n}]\, I_c\, N\,. \qquad (49)$$

The conformal center

$$C = \mathbf{c} + 1/2\, c^2\, \mathbf{n} + \check{\mathbf{n}} \qquad (50)$$

can then be extracted as

$$C = -V_3/I_c N + 1/2\, r^2 \mathbf{n}\,. \qquad (51)$$

3.6.4 Planes

Given three conformal points P_1, P_2 and P_3, the conformal 4-vector

$$V_{plane} = P_1 \wedge P_2 \wedge P_3 \wedge \mathbf{n} = \mathbf{d}\, I_p \mathbf{n} - I_p\, N, \qquad (52)$$

represents the plane through the Euclidean points $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$. The orientation of the plane is given by the Euclidean bivector

$$I_p = -\,(V_{plane}\, \mathbf{n}) \llcorner \check{\mathbf{n}}\,. \qquad (53)$$

The Euclidean origin to plane distance vector **d** can be extracted by

$$\mathbf{d} = (V_{plane} \wedge \check{\mathbf{n}})\; I_p^{-1}\; N\,. \qquad (54)$$

3.6.5 Spheres

General conformal 4-vectors of the form

$$V_4 = P_1 \wedge P_2 \wedge P_3 \wedge P_4 = (C - 1/2\; r^2\; \mathbf{n})\; i_s\; N \qquad (55)$$

represent spheres through the conformal points P_1, P_2, P_3 and P_4. The sphere radius is obtained from

$$r^2 = V_4^2\; /\; (V_4 \wedge \mathbf{n})^2\,. \qquad (56)$$

The conformal sphere center is then given by

$$C = -\,V_4/(V_4 \wedge \mathbf{n}) + 1/2\, r^2 \mathbf{n}\,. \qquad (57)$$

The Euclidean pseudoscalar (proportional to *i*) of Eq. (55) is

$$i_s = -\,(V_4 \wedge \mathbf{n})\, N\,. \qquad (58)$$

4. Java Implementation of $R_{4,1}$, the Homogeneous Model of Euclidean Space and the Subalgebras of $R_{4,1}$

4.1 Software Implementation

We have seen that the (conformal) geometric algebra allows to work with an algebra of subspaces. Within this algebra, the homogeneous model of Euclidean space allows straight-forward definitions of elementary geometric *objects* (points, pairs of points, lines, circles, spheres, planes) from a minimum of points on these objects. The Euclidean geometric characteristics of these objects, like position orientation, radius, etc. appear as easy-to-identify component parts of the object multivectors. Rotations and translations, join, intersection, projection and rejection can all be realized as simple monomial geometric products of multivectors within the algebra.

These properties very much suggest an *object oriented* programming implementation of conformal geometric algebra. Because of the subalgebras of the conformal geometric algebra, this includes automatically a wealth of implementations of further important geometric algebras (GA):

- algebra of scalar real numbers
- algebra of complex numbers
- algebra of quaternions. Quaternions are isomorphic to the even subalgebra of the GA of 3D Euclidean space.
- algebra of complex quaternions
- GAs of Euclidean lines, planes and 3D and 4D spaces. Especially the GA of 3D Euclidean space is isomorphic to the Pauli matrix algebra of quantum mechanics.
- GAs of the non-Euclidean (Minkowski) vector spaces $\mathbf{R}^{1,1}$, $\mathbf{R}^{2,1}$, $\mathbf{R}^{3,1}$, $\mathbf{R}^{4,1}$, including all their subalgebras. Especially $\mathbf{R}^{3,1}$ is the algebra of spacetime (STA) important for physical applications and isomorphic to the algebra of Dirac matrices. The even subalgebra of the STA is in turn isomorphic to the GA of 3D Euclidean space [40].

Because geometric algebra presents a unified approach to mathematics, physics and whatever applications are needed in engineering sciences, especially including computer science, the unified software implementations of all the algebras listed above will be of great benefit.

There are several design choices available for implementing the conformal model in an object oriented manner. One approach is grade-by-grade classes so that we have (scalar,)

vector, bivector, trivector, quadrivector and pseudoscalar classes. Higher grade (grade > 1) objects would be generated by the geometric product of lower grade objects. For example a bivector is the grade two part result of the geometric product of two vectors. In this way vectors become primary class objects and the MultiVector class is composed of graded objects. One of the authors has tried this approach [39] for three dimensional geometric algebra and found on one hand that grade selection then simply returns the specific grade object, but on the other hand the full multivector multiplication is harder to implement, because we have to perform multiplications between different grade objects.

Another approach is to declare MultiVector the primary class object. This way we have to manage the 32 elements internally, either using selective matrix multiplication or by taking advantage of the subspace structure of the conformal model mentioned in section 2.5.

By now various implementations of the conformal algebra and the homogeneous model in C/C++ are available [13],[14]. Calculations can also be done with geometric algebra packages added to major computer algebra softwares. This is part of widely available (much of it freeware for download) geometric algebra software [15]. But it seems that so far no Java implementation of conformal geometric algebra exists.

4.2 Why Java?

Java is a free object oriented programming language introduced and maintained by Sun Microsystems [16]. It is platform independent, simple and provides many good libraries and tools. It has support from both commercial vendors and the open source community, and has become a major language in enterprise application development. Special characteristics are Java applets operating in web browsers, interactivity and facilities for networking. Java allows to process text, graphics and sounds, including animations. It is nowadays available on more than half a billion desktop computers, and used for over 300 million smart cards. 74 % of professional software developers make use of Java and it has become an international university standard for teaching and research.

Therefore the Java package *GeometricAlgebra* development began at the University of Fukui [17]. It is open source software freely available under the GNU Lesser General Public License [18].

4.3 The GeometricAlgebra Java Package

The GeometricAlgebra Java package so far consists of ten objects (classes) and associated methods. For an overview of these classes compare Table 5.

Table 5 Ten basic implemented classes of the GeometricAlgebra Java package.

ComplexNumber
ComplexQuat
MultiVector
BasicMultiVectors
PointC
Line
Circle
Sphere
GeometricObject
SwingDrawable

The main class is called MultiVector. It is constructed in a modular way with the help of the auxiliary classes ComplexNumber and ComplexQuat(*ernion*). These are all immutable classes, because they act as data types, so they can not be changed once created. A collection of frequently used basic multivectors is the BasicMultiVectors class. PointC, Circle and Sphere [12] are special kinds of the GeometricObject class. The GeometricObject class has reference to the MultiVector class. It has command methods that every of its child classes needs [such as .meet(MultiVector mv2)] that return another MultiVector. The GeometricObject class implements SwingDrawable which has drawing methods used e.g. in the visual application KamiWaAi [12]. In this respect the GeometricAlgebra package still shows dependence to Swing.

The methods allow the implementation of the geometric product and derived products, component and grade manipulations and other important geometric algebra operations. Further algebraically not essential methods are for visual application development. In the following we will concentrate on describing the MultiVector class and its methods. We will further choose the class Sphere in order to give an example for how a special geometric object is defined as a multivector and can be used by way of its associated methods.

4.3.1 The Java Class MultiVector

The most important class MultiVector is based on the class ComplexQuat, encoding complex quaternions. ComplexQuat in turn is based on the class ComplexNumber. A ComplexNumber simply is a pair of double floating point type numbers for real and imaginary parts with methods for returning their values. Further methods allow addition, subtraction and multiplication .mult(ComplexNumber cn) of ComplexNumber(s). In the GeometricAlgebra package the

class ComplexNumber implements the subalgebra of real scalars and 5D pseudoscalars I of section 2.3.1.

A ComplexQuat consists of a linear array of four ComplexNumber(s). The first encodes the complex scalar component and the other three the three basic Euclidean bivectors $\mathbf{i}_1$, $\mathbf{i}_2$, $\mathbf{i}_3$ and their duals (multiplied by I). The methods for ComplexQuat are largely similar to the ones of ComplexNumbers. The multiplication method .mult(ComplexQuat cq2) implements the quaternion multiplication table using the .add(ComplexNumber cn), .sub(ComplexNumber cn) and .mult(ComplexNumber cn) methods of the class ComlexNumber.

A MultiVector consists of a linear array of four ComplexQuat objects in one to one correspondence with the four complex quaternions of equation Eq. (17). The methods for returning them are therefore simply called .getScPart(), .getnPart(), .getnbarPart(), and .getnhnbPart(). MultiVector has methods for addition and subtraction and most important for forming the full geometric product of two instances of MultiVector. The multiplication method .mult(MultiVector mv2) implements the geometric product making use of the lower level ComplexQuat multiplication method, adding and substracting the four ComplexQuat component objects according to Table 3. Because of the central importance of the geometric product, we include its Java source code:

```
public MultiVector mult(MultiVector mv2)
  {
  ComplexQuat M1, Mn, Mnb, Mnnb;
  ComplexQuat N1, Nn, Nnb, Nnnb;
  ComplexQuat[] MVprod = new ComplexQuat[4];
  // defining the two sets
  // of complex quaternions
  M1   = MV[0];
  Mn   = MV[1];
  Mnb  = MV[2];
  Mnnb = MV[3];
  N1   = mv2.getScPart();
  Nn   = mv2.getnPart();
  Nnb  = mv2.getnbarPart();
  Nnnb = mv2.getnhnbPart();
    // scalar part
    MVprod[0]   =  (M1.mult(N1))
                  .add(Mnnb.mult(Nnnb))
                  .sub(Mn.mult(Nnb))
                  .sub(Mnb.mult(Nn));
    // n vector part
    MVprod[1]   =   (M1.mult(Nn))
                  .add(Mn.mult(N1))
                  .add(Mn.mult(Nnnb))
                  .sub(Mnnb.mult(Nn));
    // nbar vector part
    MVprod[2]   =   (M1.mult(Nnb))
                  .add(Mnb.mult(N1))
                  .sub(Mnb.mult(Nnnb))
                  .add(Mnnb.mult(Nnb));
    // n hat nbar part
    MVprod[3]   =   (M1.mult(Nnnb))
                  .add(Mnnb.mult(N1))
                  .sub(Mnb.mult(Nn))
                  .add(Mn.mult(Nnb));
  return new MultiVector(MVprod);
  }
```

The modular three level approach of complex numbers, complex quaternions and finally multivectors thus reduces the programming work to very well structured small pieces of code on each level.

A direct application of the geometric product implementation is the method .Powerof(int power) that returns the geometric product of a multivector with itself via a loop the number of times specified by the integer parameter "power".

The next rather fundamental method is the .getGrade(int g) method for grade selection. It will return the homogeneous multivector part of the specified grade. The integer g ranges between 0 and 5. For any other value of g, a zero MultiVector (all components set to zero) will be returned. With the help of the geometric product and grade selection, we can now define important derived products of multivectors like the scalar product, the outer product, left and right contractions.

The scalar product only has one line of code, taking the real scalar part of the full geometric product of two multivectors:

```
return new
  MultiVector(MV).mult(mv2).getScPart()
                 .getScPart().RealPart();
```

The outer product uses a simple double for loop over the grade part indexes of the two multivector factors, summing up the maximum grade results of products of grade part components. Left and right contraction work very similar. The selected grades in the result of products of grade part components correspond e.g. for the left contraction to differences of grades s-r, where r is the grade of the left factor and s the grade of the right factor [8].

```
Lcontr = Lcontr.add((Mg[r].mult(Ng[s]))
                            .getGrade(s-r));
```

The reverse of a multivector is implemented simply by summing over its grade parts and changing the signs of the grade 2 and 3 parts:

```
rev = (mv1.getGrade0())
       .add(mv1.getGrade1())
       .sub(mv1.getGrade2())
       .sub(mv1.getGrade3())
       .add(mv1.getGrade4())
       .add(mv1.getGrade5());
```

The method .magnitude() returns the magnitude of a multivector as a floating point number. It takes the square root of the real scalar part of the product of a multivector with its own reverse:

```
mag = Math.sqrt( (mv1.mult(mv1.reverse()))
                  .getScPart()
                  .getScPart()
                  .RealPart() );
```

Care should be taken, because the magnitude makes only sense for multivectors from positive definite subspace algebras. This also applies to the method .normalize(), which returns a multivector divided by its scalar magnitude.

The remaining methods are .multSc(double factor) for multiplying a multivector with a real floating point scalar factor, and .get3DMVector(). The last method returns the Euclidean vector part of the multivector it is applied to, and zero for all other components.

4.3.2 The Java Class Sphere

As an example for the definition of a geometric object, we explain the class Sphere. Its main variable is a MultiVector, representing the sphere in the homogeneous model according to section 3.6.5. Therefore, we have the constructor Sphere(PointC p1, PointC p2, PointC p3, PointC p4) for creating an object Sphere based on Eq. (55). Further obvious methods are for the calculation of the radius, with the main line

```
double r2 =
  (L.ScProd(L))*(1.0/(n.OutProd(L)
                .ScProd(n.OutProd(L))));
```

and .Center() for the conformal center vector. We further have a method for changing the center of a sphere, which uses translators of Eq. (30) to move the sphere multivector L to its new center. The method .meet(MultiVector mv) is now uniquely implemented in the GeometricObject class and used by both Line and Sphere. The method .MeetNo(Line line) returns an integer by analyzing the square of the meet, with values 0,1 or 2, depending on how many points of intersection there are between the sphere and the Line line. Finally the method .MeetLineMVs(Line line) returns a Java Vector (a linear list) of conformal point multivectors representing the point or pair of points of intersection. The formulas implemented in this method are that of Eq. (42).

The class Sphere also contains methods specific for drawing spheres on screens by nets of meridians. One such method .generator(int inc, MultiVector plane) allows the user to specify a plane bivector and an angle increment 2π/inc. The result will be a rotor MultiVector according to Eq. (31), that specifies rotations in the specified plane about the center of the sphere.

The interested reader is invited to inspect the freely available source code himself. A brief list of all GeometricAlgebra package classes and their methods including the variables they accept and types of returned data can be found in [12].

In the next two sections we will discuss how to optimize the code, and the necessity and the benefits of turning the GeometricAlgebra Java package (and an associated interactive 3D sketching application KamiWaAi) into an open Sourceforge.net community project.

5. Optimization and Open Source Strategy

5.1 Refactoring the GeometricAlgebra Java Package

The advantages of geometric algebra will make it possible to provide a new software basis for many scientific and engineering applications in the future. To make this happen, the developer must first understand the fundamentals of geometric algebra. But most application developers or programmers may not have sufficient knowledge of geometric algebra, so we need to provide a simplified computational layer (or framework) to them.

Developing a good framework is expensive. One way to develop it is to design it carefully from scratch, which requires much expertise. Another way is to extract it from existing applications [20]. We will follow the last route, as it is easier and we have first applications for interactive 3D sketching[12] and geometric calculations[19]. They use the GeometricAlgebra package, and we aim to extract the desired framework from them.

Thus, we want to increase the reusability of the code and give it the needed flexibility, so that it becomes easier to change and cheaper to maintain. In software engineering, a way to improve the design of existing code is called refactoring [21]. Refactoring consists of a series of small steps to transform the code while preserving its behavioral appearance.

Fig. 1 Separating presentation and application layer.

Here are four refactoring steps and targets to be pursued:

First we have to separate the presentation and the application logic into different layers like in Fig. 1. The presentation part will handle the user interface (UI) part and the application classes the business logic, which are simple and separate responsibilities. By introducing this layered architecture we can e.g. add Japplet as a web client to applications. If we want a fast GUI component for rendering and visualizing complex geometrical objects, we may need to add another client that uses native components like the Standard Widget Toolkit (SWT [22]).

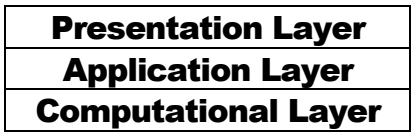

Fig. 2 Extracting computational layer from application layer.

Second, while still working on the architecture, we aim to further separate the application logic and the geometric algebra framework like in Fig. 2. Software like KamiWaAi will be sample applications that use the framework, but the framework is not limited to visual applications and can e.g. be used for algebraic computations, physical simulations or robotics applications. One possibly promising pattern is the Model-View-Controller (MVC) architecture, which originated in the late 1970s from the SmallTalk world, and is today widely used even in enterprise applications [23].

As we can see in Fig. 3, the MVC model for applications has no dependency on other package parts, neither on UI classes nor on controller classes. We can use Observer patterns [24] to decouple the model from the view. The controller takes user input, manipulates the model and causes the view to update appropriately. Because of its architectural nature this step also leads to important refactoring. If this step should run into problems, we can gain leverage from existing frameworks like JhotDraw [25] or the Graphical Editing Framework (GEF [26]) if we are using SWT.

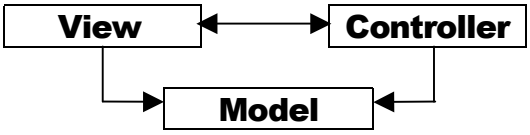

Fig. 3 Separating the presentation from the model and separating the controller from the view.

Yet total reliance on software tools will not suffice for achieving these first two steps.

Third, for other refactoring steps, we can largely depend on tools like Simian [27] or the Programming Mistake Detector (PMD [28]) to detect code duplication and other potential problems.

Fourth, if we apply automatic refactoring tools or Integrated Development Environments (IDE) like Eclipse [29], we don't really need to conduct unit testing. But if we do refactoring manually we will need to perform a series of tests by using e.g. the JUnit testing framework [30]. Tests do also increase the reliability of the software.

5.2 The GeometricAlgebra Java Package as Open Source Project at SourceForge.net

To achieve our goals we need outside help and to use the best available resources. Human ideas in a sense are the raw material of the software and humans are also its main producers. This makes humans a first order factor in software development. While we humans are creative and intelligent we to some degree are also unpredictable, irregular and inconsistent at the same time.

Big companies can hire professional software engineers and architects to get the best possible software. Well sponsored research enjoys similar advantages. But even then the maintainabilty problem remains, because no one lives forever.

Building software is hard, especially good one, and it is even harder to maintain it. Contrary to what most people believe, the cost of maintaining software is much higher than developing it. Software is principally maintained by testing and debugging, and this becomes a lot easier with user feedback. Users engage in this if they believe that developers are going to respond by improving the product and if they develop some sort of relationship with the community of developers and other users. We can build such a relationship by being transparent and by

opening our code, because the code cannot lie. Opening the source code has the further advantage, that other knowledgable individuals and groups can suggest improvements and use it to fully understand the inner workings of the GeometricAlgebra package. Open source code also strongly serves to encourage the use of the GeometricAlgebra package for building new applications. That is why we need to open the source code.

The GeometricAlgebra package is released under the GNU Lesser General Public License (LGPL [18]), so it gives opportunity to application and tool vendors to build commercial products on top of it [31],[32]. This may also attract them to join in the development. Because open source is guided by user needs and not by a business roadmap [33], it will result in "cleaner" code with less bugs.

There are other advantages of open source, with a devoted project community. Because of the nature of the internet and of geometric algebra, the collaboration scope becomes international and multi-disciplinary. This could attract experts from around the world to join the project and help solving problems. Of course, to be successful such a collaboration needs skilled leadership and effective management.

Building infrastructure to effectively manage such collaboration is not an easy task. SourceForge [35] already provides infrastructure and sufficient tools for hosting projects that conform to open source definitions according to the Open Source Initiative (OSI [34]). We will have a Concurrent Version System (CVS [46]) for repository and source code control. Further included are an online forum and a mailing list for discussions, and we also have a website to maintain and document an open source project. The GeometricAlgebra Java package along with the associated visual application KamiWaAi are now hosted at http://kamiwaai.sourceforge.net The authors really look forward that those interested in geometric algebra will join the future development of the package and increase the quality and variety of associated applications.

6. Conclusion

We have briefly reviewed the algebraic background of the conformal model of Euclidean space. We found the subalgebra structure of the geometric algebra $\mathbf{R}_{4,1}$ of special interest for designing an object oriented implementation of the geometric product of multivectors in a well structured three level approach. We further explained how to algebraically construct conformal (homogeneous) subspaces with very intuitive Euclidean interpretations. We introduced the algebraic expressions for arbitrary translations and rotations, and for the subspace operations of union (join), intersection (meet), projection and rejection. All these are implemented as methods in the GeometricAlgebra Java package.

We reviewed how the joining of conformal points yields explicit expressions for points, pairs of points, lines, circles, planes and spheres. After that we stated how in each case the Euclidean 3D information of positions, orientations and radii, etc. can be extracted. These formulas form the mathematical structure of the related Java methods each geometric object has in the GeometricAlgebra Java package implementation.

We found an object oriented software implementation to be most suitable. Wide spread use and platform independence motivated our adoption of Java.

Going into the details of the implementation of multivectors as MultiVector Java classes, we demonstrated the direct and intuitive correspondence between algebraic expressions and Java source code. As for the homogeneous model multivectors with Euclidean interpretation, we concentrated on the class Sphere, to elucidate construction and use (the *methods*) of these simple $\mathbf{R}_{4,1}$ multivectors.

Because the GeometricAlgebra Java package and related applications are still under development we identified the necessary steps for its optimization (refactoring). Further expansions of the package are likely to include the class Plane, and an implementation of the .plunge(…) product method for multivectors. To take the *plunge* product (for the co-incidence) [41] of multivectors A, B $\in \mathbf{R}_{4,1}$, means to calculate the outer product of the duals of A and B:

$$(AI^{-1}) \wedge (BI^{-1}) \qquad (59)$$

It may also be worthwhile to develop a range of browser integrated online Java applet calculators for important algebras like quaternions and the space-time (Dirac) algebra, etc.

Being concerned about the best possible match between geometric algebra and object oriented software language apart from Java other more radically object oriented software languages like Ruby [42] may bring about an even better correspondence between mathematical formulation and programming code.

Apart from the mathematical and logical beauty of a software implementation, optimization also means performance tuning. The geometric algebra software community is taking this aspect very serious [14]. As for the GeometricAlgebra Java package it may help to not always use the full Multivector class. For frequent computations, which involve only subalgebra objects like real scalars, “complex” scalars (scalar and pseudoscalar), vectors or quaternions (scalars and bivectors), etc. it may increase the efficiency to specifically use the corresponding lower level objects like real

numbers, ComplexNumber and ComplexQuat.

Apart from this first principles approach there is the future option to profile GeometricAlgebra Java package applications and libraries and change the code for reducing time expensive bottle necks. With such an approach care may have to be taken about code readability and sufficient documentation.

We finally argued why we think it is highly beneficial to convert all this into an open SourceForge project [17].

Acknowledgements

Soli Deo Gloria. E. Hitzer does want to thank his wife and children. We also thank our colleague Y. Fujii and D. Fontijne (Amsterdam) for checking the manuscript.

References

[1] D. Hestenes: *New Foundations for Classical Mechanics* (2nd ed.), Kluwer, Dordrecht, 1999.

[2] C. Doran, A. Lasenby, J. Lasenby: *Conformal Geometry, Euclidean Space and Geometric Algebra*, in J. Winkler (ed.), Uncertainty in Geometric Computations, Kluwer, 2002.

[3] G. Sommer (ed.): *Geometric Computing with Clifford Algebras*, Springer, Berlin, 2001.

[4] P. Lounesto: *Clifford Algebras and Spinors*, CUP, Cambridge, 2001.

[5] C. Chevalley: *The Algebraic Theory of Spinors and Clifford Algebras*, Springer, Berlin, 1997.

[6] D. Hestenes and G. Sobczyk: *Clifford Algebra to Geometric Calculus*, Kluwer, Dordrecht, reprinted with corrections 1992.

[7] G. Sobczyk: *Clifford Geometric Algebras in Multilinear Algebra and Non-Euclidean Geometries*, Lecture at *Computational Noncommutative Algebra and Applications*, July 6-19, 2003, http://www.prometheus-inc.com/asi/algebra2003/abstracts/sobczyk.pdf

[8] L. Dorst: *The Inner Products of Geometric Algebra*, in L. Dorst et. al. (eds.): Applications of Geometric Algebra in Computer Science and Engineering, Birkhaeuser, Basel, 2002.

[9] L. Dorst: *Interactively Exploring the Conformal Model,* Lecture at *Innovative Teaching of Mathematics with Geometric Algebra 2003*, Nov. 20-22, Kyoto University, Research Institute for Mathematical Sciences (RIMS), Japan.

[10] D. Hestenes, H. Li, A. Rockwood: *New Algebraic Tools for Classical Geometry*, in G. Sommer (ed.), Geometric Computing with Clifford Algebras, Springer, Berlin, 2001.

[11] J. Suter: *Geometric Algebra Primer*, http://www.jaapsuter.com/

[12] E.M.S. Hitzer: *KamiWaAi - Interactive 3D Sketching with Java based on Cl(4,1) Conformal Model of Euclidean Space*, Advances in Applied Clifford Algebras 13-1, 11-45 (2003).

[13] C. Perwass: *CLUCalc website*, http://www.clucalc.info/

[14] D. Fontijne, T. Bouma, L. Dorst: *Gaigen: a Geometric Algebra Implementation Generator*, July 28, 2002, preprint. http://www.science.uva.nl/ga/gaigen/files/20020728_gaigen.pdf

[15] E.M.S. Hitzer: Mem. Fac. Eng. Fukui Univ. 50-1, 109 – 125 (2002).

[16] Java homepage http://java.sun.com/

[17] *GeometricAlgebra Java package homepage* at Source-Forge http://kamiwaai.sourceforge.net and its original homepage at the University of Fukui http://sinai.mech.fukui-u.ac.jp/gcj/software/KamiWaAi/index.html

[18] GNU open software license http://www.gnu.org/copyleft/lesser.html

[19] E.M.S. Hitzer: Dihedral angle online applet calculator http://sinai.mech.fukui-u.ac.jp/gcj/software/Dihed/index.html

[20] D. Roberts and R. Johnson: *Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks*, http://st-www.cs.uiuc.edu/users/droberts/evolve.html

[21] M. Fowler, *Refactoring: Improving the design of existing code*, Addison Wesley, 1999.

[22] Standard Widget Toolkit (SWT) website, http://www.eclipse.org/swt

[23] M. Fowler: *Patterns of Enterprise Application Architecture*, Addison Wesley, 2003.

[24] Gamma, Helms, Johnson, Vlissides, *Design Patterns: Elements of Reusable Software*, Addison Wesley, 1995.

[25] *JHotDraw* website, http://jhotdraw.sourceforge.net

[26] Graphical Editing Framework (GEF) website, http://www.eclipse.org/gef

[27] *Simian* website, http://www.redhillconsulting.com.au/products/simian/

[28] Programming Mistake Detector (PMD) website, http://pmd.sourceforge.net/

[29] Integrated development environment *Eclipse* website

http://www.eclipse.org/
[30] Testing framework *JUnit* website, http://www.junit.org
[31] GNU website, http://www.gnu.org
[32] N. A. Rupp: *On Branding and Copyright - Open Source For Entrepreneurs*, http://weblogs.java.net/pub/wlg/586
[33] R. X. Cringely: *Unplugged: How Microsoft's Misunderstanding of Open Source Hurts Us All*, http://www.pbs.org/cringely/pulpit/pulpit20031023.html
[34] Open source definition from Open Source Initiative (OSI), http://opensource.org/docs/definition.php
[35] SourceForge website, http://sourceforge.net
[36] Concurrent Version System (CVS) website, http://www.cvshome.org
[37] E. M. S. Hitzer: *Euclidean Geometric Objects in the Clifford Geometric Algebra of {Origin, 3-Space, Infinity}*, to be printed in: Bulletin of the Belgian Mathematical Society – Simon Stevin.
[38] E. M. S. Hitzer: to be printed in H. Li, P. Olver, G. Sommer (eds.), *GIAE Xian conference proceedings*, Lecture Notes in Computer Science, Springer, 2005.
[39] G. Utama: *The Development of a Java Based API for some Geometric Algebra Operations*, Thesis, Institute of Technology Bandung, (2001).
[40] D. Hestenes: *Space-Time Algebra*, Gordon and Breach, New York, 1966.
[41] L. Dorst and D. Fontijne: *An Algebraic Foundation for Object-Oriented Euclidean Geometry*, in E.M.S. Hitzer, R. Nagaoka, H. Ishi (eds.): Proc. of *Innovative Teaching of Mathematics with Geometric Algebra Nov. 2003*, Research Institute for Mathematical Sciences (RIMS), Kyoto, Japan, RIMS **1378**, 138 – 153 (2004).
[42] Ruby website: http://www.ruby-lang.org/en/