COMPLEX CONJUGATION – RELATIVE TO WHAT?

ALEXANDER M. SOIGUINE
191011 St. Petersburg, Fontanka 53 #37
Russia

Abstract. Some initial, technically simple but fundamentally important statements concerning the very origin of the notion of a complex number are formulated in terms of the Clifford (Geometric) algebra generated by vectors in some geometrically and physically sensitive dimensions. A new insight into the sense of geometrical product is given. It is shown that it makes no sense to speak about complex numbers without identifying a corresponding two-dimensional plane. This is particularly important if the given physical situation is set in higher dimensions. Because of great importance of these questions in education and because of increasing use of graphical computer programs in mathematical education and research, some components of a computer program implementing the Geometric Algebra approach are outlined in terms of classes of the object-oriented computer language C++.

Key words: Cognitive process – imaginary unit – CLICAL

1. Introduction

Teaching of mathematics has improved significantly particularly where mathematical formalism can be represented by a computer program able to manipulate symbols according to certain rules. The improvement is most effective if the involved mathematical objects have an explicit and unambiguous geometric interpretation transformed into computer visual images. In such case the cognitive process is enforced radically by including in it human visual system – our most powerful sensor system for communication with the outer world.

Clifford (or, maybe, better to say "Geometric") algebra is one of the best examples of such a field in mathematics, the more so as its mathematical formalism is tending to become a unified language for theoretical physics (Hestenes and Sobczyk, 1987). Geometric algebra is superbly convenient to be supplied with adequate computer representation programs. It is extremely important today when we have, from one side, a tendency among students of mathematics or physics for a more intuitively obvious and even visualizable basis for mathematical constructions and, from another side, we have available computer facilities, in hardware and software, to meet such desires.

For several years now we have had CLICAL, a well known and widely recognized computer program developed by professor Pertti Lounesto and his colleagues (Lounesto *et al.*, 1987). This program allows a student to make all algebraic calculations in the field of Clifford algebras – from complex numbers to the special relativity theory. At the same time it should be said that CLICAL code is written (and, sure, couldn't be written in another way at those days) in the text mode. While the program can only manipulate numbers and symbols, objects of the Geometric Algebras, at least in low, visualizable dimensions, have direct geometrical interpretation. So today we have a challenge to transform CLICAL into a graphical interface or, really, to develop a *Clicalian* computer program able to appropriately manipulate and visualize corresponding geometrical objects. We have now good computer displays and we have adequate and powerful computer languages to deal with graphical images namely object-oriented languages such as, first of all, C++. Development and implementation of such program could have far reaching educational consequences and may be considered as the next generation of Lounesto's CLICAL.

Before delving into a discussion of C++ classes representing geometric objects and their behaviors, we clarify some basic and well known geometric facts about Clifford algebras to make them absolutely unambiguous in their geometrical interpretation hence more suitable for computer implementation.

Hestenes said that "physicists quickly become impatient with any discussion of elementary concept" (Hestenes, 1985). This statement is not applicable to the author because his goal here is not to make a show on super complicated mathematical wisdom but to reveal, at a maximal degree, initial geometrical sense of some basic operations in the Clifford algebra and track possible relations with corresponding C++ units. From this point of view the current work has a methodological and educational content. So, we shall start with recollection of some ideas.

2. Geometric algebra of the plane

The arena of the performance will now be a two-dimensional plane denoted here as E_2. At the very beginning I emphasize that in no way E_2 is viewed here as a two-dimensional linear space, that is as a set of linear combinations $\alpha^1 e_1 + \alpha^2 e_2$ where e_1, e_2 are two arbitrary linearly independent vectors from E_2 and α^1, α^2 are scalars.[1] For our purposes E_2 should be thought of as a geometrical object, an abstraction arising from such real things as a table-desk or a sheet of paper. I won't make any attempt to give a formal definition of E_2, I am just considering it as an intuitively obvious geometrical object.

Certainly, elements of the form $\alpha^1 e_1 + \alpha^2 e_2$ are in E_2. These are directed segments of straight lines in the plane, or arrows. We shall call them (two-dimensional) vectors. A vector may be identified by its length (value) and direction (orientation).

However, there exists at least one another sort of geometrical objects inhabiting E_2 which are of the greatest importance for in Geometric Algebra considerations. I mean closed oriented curves without self-intersections lying in E_2.

Closed oriented curves are identified, similarly to usual vectors, by value and orientation. By the value of such a geometrical object we mean a non-negative number which is equal to the area inside the curve. By the orientation we mean one of the two possible directions of movement along the closed curve. One will be called positive, another negative. For shortness, we will call the oriented closed curves on E_2 *bivectors* as is the common practice.

Ordinary free vectors in the plane are defined up to parallel movements in E_2.

[1] I do not say *real* scalars since, I hope, it would be evident soon that it has a little sense to speak about complex-valued scalars. If we would follow that practice, then elements of any field, over which a linear space is considered, should be called scalars.

Bivectors are defined up to arbitrary movements and curve deformations which don't change the area inside the curve.

Let us make some agreements about notation. Usual vectors will be denoted by small Latin letters: $a, b, \ldots$. Bivectors will be denoted by capital Latin letters: $A, B, \ldots$. For scalars we will use Greek alphabet: $\alpha, \beta, \ldots$. Values of vectors and bivectors are denoted, along with the absolute values of scalars, as $|a|, |A|, \ldots$. Orientation of a bivector A will be denoted as $\mathcal{O}_A$ and if it is given then the two possible values are denoted as $\mathcal{O}_A^+$ and $\mathcal{O}_A^-$.

The set of vectors in E_2 is a linear space with the known geometrical sense of the operations. Now we have to supply the set of bivectors in E_2 with the structure of a linear space.

Given two bivectors, A, B, their sum is a bivector $C \equiv A+B$, the value of which is equal to $|A| + |B|$ in the case of coinciding orientations of A and B, $\mathcal{O}_A = \mathcal{O}_B$, and $||A| - |B||$ in the case of opposite orientations, $\mathcal{O}_A = -\mathcal{O}_B$. The orientation of C is the same as the common orientation of A and B in the first case and is equal to the orientation of bivector with the greater value in the second case.

Multiplication by a scalar is defined in an obvious way. Given a bivector A and a scalar λ, the product λA is a bivector with $|\lambda A| = |\lambda||A|$ and $\mathcal{O}_{\lambda A} = \mathcal{O}_A$ if $\lambda > 0$, and $\mathcal{O}_{\lambda A} = -\mathcal{O}_A$, if $\lambda < 0$.

The zero element for addition is the equivalence class of oriented curves with zero inside area.

The unit positive oriented bivector will be denoted by i_{E_2}, or simply i, because we have only one plane E_2 at our disposal for the moment. Generally, an "imaginary unit" must be supplied with a subscript denoting the plane in which the unit bivector lies.

All axioms of a linear space are verified immediately for bivectors. The linear space of bivectors in E_2 will be denoted by $\mathbf{B}$, the linear space of vectors will be denoted by $\mathbf{V}$, and the linear space of scalars will be denoted here by $\mathbf{S}$.

Now we can form outer direct sums of these linear spaces in various combinations: $\mathbf{S} \oplus \mathbf{V}, \mathbf{S} \oplus \mathbf{B}, \mathbf{V} \oplus \mathbf{B}$ and $\mathbf{S} \oplus \mathbf{V} \oplus \mathbf{B}$. Recall that the direct sum of linear spaces is a linear space of all sequences of elements belonging to the corresponding spaces. We may consider direct sum as the set of all given linear spaces "put in one bag", where they don't mix up, similar to distinct purchases in a bag. Linearity is checked easily, for example, addition in $\mathbf{V} \oplus \mathbf{B}$ is defined as:

$$\mathbf{V} \oplus \mathbf{B} \ni M_1, M_2 : M_1 + M_2 = \big((M_1)_{\mathbf{V}} + (M_2)_{\mathbf{V}}, (M_1)_{\mathbf{B}} + (M_2)_{\mathbf{B}}\big) \in \mathbf{V} \oplus \mathbf{B}$$

where $(M_i)_{\mathbf{V}}$ and $(M_i)_{\mathbf{B}}$ are, respectively, the vector and the bivector components of $M_i \in \mathbf{V} + \mathbf{B}$.

The linear space $\mathbf{C} \equiv \mathbf{S} \oplus \mathbf{B} \equiv \mathbf{S} \oplus i\mathbf{B}$ is the set of complex numbers in E_2. It is a subalgebra of $\mathbf{G}_2 \equiv \mathbf{S} \oplus \mathbf{V} \oplus \mathbf{B}$, the Geometric (Clifford) Algebra of the plane E_2. To explain the usage of the term "algebra" we shall define multiplication in $\mathbf{G}_2$.

Let's begin with a definition of vector multiplication since multiplication by scalars is trivial. Given two vectors $a, b \in \mathbf{V}$ we define their product as a map

$$\mathbf{V} \times \mathbf{V} \ni (a, b) \to ab = (a \cdot b, a \wedge b) \in \mathbf{S} \oplus \mathbf{B}$$

where $a \cdot b$ is a scalar, equal to the scalar product of vectors a and b computed as

$|a||b|\cos\widehat{(a,b)}$ and $a \wedge b$ is a bivector with the value $|a||b||\sin\widehat{(a,b)}|$ and orientation defined by rotation of a to b through the angle which is less than π.

If we identify in a natural way elements of each space, comprising the direct sum, with elements of the form $(\alpha, 0_{\mathbf{V}}, 0_{\mathbf{B}})$, $(0_{\mathbf{S}}, a, 0_{\mathbf{B}})$ and $(0_{\mathbf{S}}, 0_{\mathbf{V}}, A)$ and denote the addition in $\mathbf{S} \oplus \mathbf{V} \oplus \mathbf{B}$ by the usual "plus", then an arbitrary $M \in \mathbf{S} \oplus \mathbf{V} \oplus \mathbf{B}$ is written as $M = \alpha + a + A$. In the same way $\mathbf{S} \oplus \mathbf{B}$ is identified with the subalgebra of $\mathbf{G}_2$ of elements of the form $\alpha + A$, so $ab = a \cdot b + i_{E_2}|a \wedge b|$.

It remains to define product of a vector with a bivector and product of two bivectors.

For $a \in \mathbf{V}$ and $A \equiv i_{E_2}\alpha \in \mathbf{B}$, multiplication of a by A from the right, aA, is vector of value $|\alpha||a|$, rotated in the direction of the positive orientation in E_2 by the angle $\frac{\pi}{2}$, with the opposite direction in the case of negative α. If the operands have opposite order of multiplication, Aa, the rotation is made in the direction of negative orientation on E_2. Clearly, $aA = -Aa$.

To calculate ${i_{E_2}}^2$ we first check the associativity of the geometric product of vectors. If the distributivity of multiplication of vectors relatively to their addition is valid then it suffices to consider products of three basis vectors. The proof of distributivity is a tedious exercise in trigonometry, and we take it for granted. Then we have for basis vectors $\mathrm{e}_1(\mathrm{e}_1\mathrm{e}_2) = \mathrm{e}_1 i_{E_2} = \mathrm{e}_2$ and $(\mathrm{e}_1\mathrm{e}_1)\mathrm{e}_2 = \mathrm{e}_2$, thus associativity holds. Hence, together with anticommutativity of orthogonal vectors, it follows that ${i_{E_2}}^2 = \mathrm{e}_1\mathrm{e}_2\mathrm{e}_1\mathrm{e}_2 = -1$.

Now let's turn to complex conjugation. In conventional geometric interpretation complex numbers are represented as vectors in some given orthogonal reference frame on the plane which is called the *complex plane.* This counterclockwise frame has the first axis as the *real* one, and the second axis, rotated by $\frac{\pi}{2}$ in positive direction, as the *imaginary* axis. A complex number (vector) has the first projection called the *real* part, and the second projection called the *imaginary* part. Conventionally, when written algebraically as $a + ib$, where a and b are usual (real) numbers and i is formally introduced as "imaginary unit" with the crucial property $i^2 = -1$, the complex numbers possess the operation of "complex conjugation": $a + ib \rightarrow a - ib$, which will be seen to be basis dependent.

At the same time, unambiguous and the only correct representation of complex numbers as elements of $\mathbf{S} \oplus \mathbf{B}$ and having the form $\alpha + i_{E_2}\beta$ has no such deficiency. The map $\alpha + i_{E_2}\beta \rightarrow \alpha - i_{E_2}\beta$ doesn't depend on a basis in E_2 and means that orientation of the bivector part in a pair $(\alpha, i_{E_2}\beta)$ is changed to the opposite.

Then, what to do with the conventional complex conjugation considered as the reflection of the vector relative to the real axis? First, we remember that there exists a correspondence between elements $\alpha + i_{E_2}\beta$ of $\mathbf{S} \oplus \mathbf{B}$ considered as complex numbers in a strict sense and vectors from $\mathbf{V}$.

Suppose, we've chosen a unit vector e_1 on E_2, the latter is by assumption counterclockwise positive oriented. Given a complex number $\alpha + i_{E_2}\beta$, multiply it by e_1 from the left. Then we get the vector $\alpha\mathrm{e}_1 + \beta\mathrm{e}_2$, where e_2 is the unit vector rotated from e_1 by $\frac{\pi}{2}$ in the positive direction. So, we get the conventional geometric representation of a complex number as a vector. Arbitrarily chosen unit vector e_1 fixes the real axis, e_2 – the imaginary one. Complex conjugation in these terms is the reflection of $\alpha\mathrm{e}_1 + \beta\mathrm{e}_2$ relative to e_2.

We can call $\alpha e_1 + \beta e_2$ a *relative vector representation* of a complex number $\alpha + i_{E_2}\beta$ defined by the vector e_1.

If we choose another unit vector, say f_1, in E_2, we get the same "vectorial" picture together with complex conjugation as the reflection relative to the direction orthogonal to f_1 but the picture is rotated in E_2 by the angle between e_1 and f_1. Here we have maybe the simplest example of a gauge transformation connecting different relative representations of geometrically invariant object, a complex number in this case. It is worth mentioning that a rotation in E_2 is generated by multiplication by elements $\alpha + i_{E_2}\beta$ where $\alpha^2 + \beta^2 = 1$. For an explanation of this well known fact see (Gull *et al.*, 1993). Because of the above condition on α and β, bivector generating a rotation may be written as $\cos\phi + i_{E_2}\sin\phi$, where ϕ is the angle of rotation. Since the rules for addition of the angle arguments in multiplication of such bivectors coincide with those in multiplication of exponents, rotation generating bivectors are symbolically written as $e^{i_{E_2}\phi}$, taking into account the Euler's formulas and accepting the fact that the function sin *feels* orientation, whereas cos doesn't.

Having reviewed briefly some facts about the two-dimensional plane, which we consider as insufficiently clarified in the works on Geometric Algebra, we pass to even more fragmental discussion on higher dimensions.

3. Higher dimensions

Now we'll touch briefly the cases of higher physical dimensions. All results are principally known and only the interpretation, transparent geometrical sense and unambiguous relations between different representations are important to us. Some additional interesting details may be found in (Soiguine, 1990).

Suppose we are in the physical three dimensions. The Geometrical product of any two vectors is the same as in the plane, but here the plane of the bivector component is defined as the plane spanned by vector multipliers. Orientation of that plane is determined by given orientation of the three-dimensional space, which may be left or right by definition.

We are to define a geometrical product of a vector and bivector when the former is not in the bivector plane. To do this, we expand vector in two components, parallel and perpendicular to the bivector plane, $a = a_{\|} + a_{\perp}$. The product of the parallel component $a_{\|}$ with given bivector B is calculated by the rule of the previous section. It gives vector $a_{\|} i_B |B|$ or $i_B a_{\|} |B|$ depending on the order of the multipliers. The product $a_{\perp} B$ gives oriented volume equal to $|a_{\perp}||B|$, the orientation being determined by a screw representing both rotation in the bivector B plane in the direction of B and movement along $a_{\perp}$. It is very convenient to imagine an oriented volume as a cylinder of the given volume with a string wound around it in one of the two possible ways. The oriented volumes, or trivectors, may be written as αi_3, where i_3 is a positively oriented unit volume.

Geometric product of a vector a with a trivector αi_3 gives, independently of the order, a bivector of value $|\alpha||a|$ lying in the plane orthogonal to a and oriented in such way that its orientation and the direction of a restore the orientation of αi_3.

This rule actually gives the correspondence of duality between vectors and bivectors in three dimensions. If we have $A = i_3 a$ then A is called *dual* to a.

If A is a bivector then $i_3 A = A i_3$ is vector of the value $|A|$, orthogonal to the plane of A and having orientation which, together with the orientation of A, gives three-dimensional orientation opposite to that of i_3.

If $a = i_3 A \equiv A i_3$ then a is called dual to A.

The inversion of the resulting common orientation in duality between vector and bivector, and bivector and vector follows from the fact that double duality changes the sign of a geometrical object. For example, if $i_3 a = A$ then multiplying by i_3 from the left gives $-a = i_3 A$. Here we use the fact that $i_3^2 = -1$, which may be checked from possible representation $i_3 = \mathrm{e}_1\mathrm{e}_2\mathrm{e}_3$ where $\{\mathrm{e}_i\}$ – unit orthogonal basis of given orientation in three dimensions.

Now, a general element of the Geometrical algebra in the three physical dimensions, $\mathbf{G}_3 \equiv \mathbf{S} \oplus \mathbf{V} \oplus \mathbf{B} \oplus \mathbf{T}$, has the form:

$$\mathbf{G}_3 \ni M = \alpha + a + i_3 b + i_3 \beta.$$

What is the complex conjugation in this case? For example, the subset of elements of the form $\alpha + i_3 b$ comprise a subalgebra, if we mean the algebraic form of elements. At the same time any such element determines its own "complex plane", defined by the direction of vector b in three dimensions and orientation of i_3. In multiplication two such elements give an element of the same algebraic form but with its new complex plane. If $i_3 b = B$ then always we can write $\alpha + i_3 b = \alpha + i_B \beta$, where $|\beta| = |b| = |B|$. So, we can conclude that quaternions are really complex numbers differing from the convention in a single aspect that a quaternion "complex plane" is arbitrarily imbedded in the three dimensions.

At the end of this section I would like to suggest an interesting question to the reader: Is it possible to adjoin to the indefinite metric in the special relativity a "complex conjugation" remembering that the latter is nothing else but an inversion of orientation?

4. C++ classes implementation of Geometric Algebra in the plane

Now we consider possible computer C++ language implementation of geometrically obvious operations in Geometric algebra in the plane restricting ourselves to the simplest case of complex numbers. The C++ language is adequate in its `class` type variable structure to describe real physical things, and geometric objects in the plane in particular.

As it was mentioned earlier, the forthcoming computer program should be written in a graphical mode and because of that user input operations and screen output operations will become more complicated. The program is menu-controlled. Input may be either digital or graphical while complex numbers are input as vectors with the mouse manipulations.

The `class` type variable representing complex numbers together with operations on them may be defined as:

```
typedef class tagCOMPLEXNUMBER {
double RealPart;
```

```
   double ImagPart;
   int ScrRealPart;
   int ScrImagPart;
   char StrRealPart[];
   char StrImagPart[];
   int xScreenOrigine;
   int yScreenOrigine;
   int RSEntered;
   int ISEntered;
   int GetStrReal(int,int,char*);
   int GetStrImag(int,int,char*);
   int WriteNumber(int);
   void EraseNumberPicture(void);
   void EnlargeNumberPicture(double times);
   void ConvertRealToString(void);
   void ConvertImagToString(void);
   void ConvertStringToReal(void);
   void ConvertStringToImag(void);

      public:

   friend tagCOMPLEXNUMBER operator+(tagCOMPLEXNUMBER,
tagCOMPLEXNUMBER);

   friend tagCOMPLEXNUMBER operator-(tagCOMPLEXNUMBER,
tagCOMPLEXNUMBER);

   friend tagCOMPLEXNUMBER operator*(tagCOMPLEXNUMBER,
tagCOMPLEXNUMBER);

   friend tagCOMPLEXNUMBER operator/(tagCOMPLEXNUMBER,
tagCOMPLEXNUMBER);

   void SetRealPart(double value) {RealPart = value;}
   double GetRealPart(void) {return RealPart;}
   void SetImagPart(double value) {ImagPart = value;}
   double GetImagPart(void) {return ImagPart;}
   void SetScrRealPart(int value) {ScrRealPart = value;}
   int GetScrRealPart(void) {return ScrRealPart;}
   void SetScrImagPart(int value) {ScrImagPart = value;}
   int GetScrImagPart(void) {return ScrImagPart;}
   int EnterNumber(int);
   void DrawNumber(void);
         } COMPLEXNUMBER;
```

Only a part of the type definition is written here, consisting of comprehensive class variables and functions.

Partition between private and public functions in the COMPLEXNUMBER class depends on concrete realization. For example, one of the main public functions, EnterNumber, has the C++ code:

```
int tagCOMPLEXNUMBER::EnterNumber(int number) {
int result=0;
do {
switch(mode % 2) {
case 0:
if (number==1)
{
if
(WriteNumber((strlen("operand")+2)*charwidth))
result=1;
else result=0; } else
if (number==2)
{
if
(WriteNumber((strlen("operand")+4)*charwidth+X_ max/2))
result=1;
else result=0; }
break;
case 1:  break;
} } while(!result);
return result;
}
```

The private function WriteNumber, in turn, has the code:

```
int tagCOMPLEXNUMBER::WriteNumber(int leftboundary) {
static int RResult=0;
static int IResult=0;
char* invfigptr = 0;
if (!(mode % 2))
do {
if (leftboundary<X_ max/2)
WashRect(leftboundary,Ymax+1,X_ max/2-2,Ystatusline-17);
else if(leftboundary>=X_ max/2)
WashRect(leftboundary,Ymax+1,X_ max-1,Ystatusline-17);
GetStrReal(leftboundary+4,Ymax+11,"Real part = ");
if (RSEntered) RResult=1;
SetRealPart(strtod(StrRealPart,& invfigptr));
if(*invfigptr!=NULL) Beep();
}
while ((*invfigptr!=NULL)&& (!(mode % 2)));
else { WashRect(1,Ymax+1,X_ max-1,Ystatusline-1); }
invfigptr = 0;
if (!(mode % 2))
do {
if (leftboundary<X_ max/2)
WashRect(leftboundary,Ymax+17,X_ max/2-2,Ystatusline-1);
else if(leftboundary>=X_ max/2)
WashRect(leftboundary,Ymax+17,X_ max-1,Ystatusline-1);
GetStrImag(leftboundary+4,Ystatusline-5,"Imaginary part = ");
if (ISEntered) IResult=1;
SetImagPart(strtod(StrImagPart,& invfigptr));
if(*invfigptr!=NULL) Beep();
}
while ((*invfigptr!=NULL)&& (!(mode % 2)));
else { WashRect(1,Ymax+1,X_ max-1,Ystatusline-1); }
if (RResult&& IResult) return 1; else return 0;
}
```

Here, for example, the function GetStrReal, implementing the input of a string containing the value of the real (scalar) part of complex number, has the code:

```
int tagCOMPLEXNUMBER::GetStrReal(int X,int Y,char* mes)
{ RSEntered = 0;
int len=strlen(mes), i=0, j, maxlen, k;
char ch, s2[2];
maxlen = charwidth*(len+12); s2[1] = '\0';
outtextxy(X,Y,mes);
while(1)
{ j=(len+i)*charwidth;
if(j>=maxlen) break;
TxtCursor(X+j, Y-6);
ch=getch();
if (!ch) ch=getch();
TxtCursor(X+j, Y-6);
if(ch==ESC || ch==BACKSPACE || ch==ENTER)
{ if (ch==ESC)
{
if(i>0) {k=(len+i)*charwidth; i=0;
WashRect(X+len*charwidth,Y-charheight,X+k+charwidth,Y);}
} else
if (ch==BACKSPACE)
{ if(i>0) i--;
k=(len+i)*charwidth;
WashRect(X+k,Y-charheight,X+k+charwidth,Y);
}
else
if (ch==ENTER) {RSEntered=1;break;}
```

```
  }
  else
  { StrRealPart[i]=s2[0]=ch;
  outtextxy(X+j,Y,s2);
  i++; } } if (i==0) {StrRealPart[i]=s2[0]='0';
outtextxy(X+j,Y,s2);i++;}
  StrRealPart[i]='\0';
  return *StrRealPart;
  }
```

The above portions of C++ source code illustrate work which is in progress now and, I hope, will soon result in a valuable computer program implementing Geometric algebra visual interpretation.

5. Conclusions

The author's intentions here were mainly pedagogical and methodological. The point is that Geometric algebra of the plane or the three-dimensional physical space has unambiguous and transparent geometric interpretations not realized yet in all details. This interpretation may be realized in computer images in terms of adequate object-oriented language C++. The work is in progress and is expected to yield next graphical version of the well known CLICAL.

References

S. Gull, A. Lasenby, Ch. Doran: 1993, 'Imaginary numbers are not real – the Geometric Algebra of spacetime', *Found. Phys.* **23**, pp. 1175–1201.

D. Hestenes, G. Sobczyk, *Clifford Algebra to Geometric Calculus*, Reidel, Dordrecht.

D. Hestenes: 1985, 'Clifford algebra and interpretation of quantum mechanics', *Proc. NATO and SERC Workshop on Clifford Algebras and Their Applications in Mathematical Physics*, Canterbury, pp. 321–346.

P. Lounesto, R. Mikkola, V. Vierros: 1987, *CLICAL, Program and User Manual*, Helsinki University of Technology, Helsinki.

A.M. Soiguine. *Vector Algebra in Applied Problems*, Naval Academy Publ., St. Petersburg, 1990. (in Russian)