

Spinor Product Computations for Protein Conformations

Pieter Chys and Pablo Chacón*

Spinor operators in geometric algebra (GA) can efficiently describe conformational changes of proteins by ordered products that act on individual bonds and represent their net rotations. Backward propagation through the protein backbone yields all rotational spinor axes in advance allowing the efficient computation of atomic coordinates from internal coordinates. The introduced mathematical framework enables to efficiently manipulate and generate protein conformations to any arbitrary degree. Moreover, several new formulations in the context of rigid body motions are

added. Emphasis is placed on the intimate relationship between spinors and quaternions, which can be recovered from within the GA approach. The spinor methodology is implemented and tested *versus* the state of the art algorithms for both protein construction and coordinate updating. Spinor calculations have a smaller computational cost and turn out to be slightly faster than current alternatives. © 2012 Wiley Periodicals, Inc.

DOI: 10.1002/jcc.23002

Introduction

The coordinate choice for macromolecules can have an important influence on the efficiency of modeling techniques. Cartesian coordinates (CC) always represent the atomic positions inside a macromolecule relative to a three-dimensional (3D) coordinate system and such atomic positions are always independent of each other. CC play an important role in biomodeling as the energy and scoring functions are most easily expressed and evaluated. Alternatively, atomic distances, valence angles, and dihedral angles can be used as internal coordinates (IC). IC allow for a substantial reduction in the number of degrees of freedom, especially when bond lengths and valence angles are considered rigid and fixed. Also, IC are more appropriate coordinates to describe the behavior of molecules as they represent more naturally molecular geometry. Molecular simulation packages such as ICM,^[1] X-PLOR^[2]/CNS,^[3,4] or TINKER,^[5] among others, extensively exploit the IC advantages.^[6–8] However, as CC are much more convenient for calculating the non-bonded energy terms and its derivatives, conversion from IC to CC is a critical step.

Efficient and fast algorithms for generating and updating conformations in Cartesian from IC have been studied fairly intensive in the past.^[9–15] Most coordinate methods use matrices, quaternions, or hybrid combinations of them.^[10–13] Recently, two other approaches have as well been proposed. The polyspherical coordinate (PSC) method^[14] describes the molecule as a forward tree of bond vectors generated from the two preceding vectors and their vector cross product. In Ref. [15], we first introduced geometric algebra (GA) and spinors to construct and manipulate polymer conformations. GA (see Supporting Information) is closely related to the Clifford algebras but focuses on geometrical interpretation instead of pure mathematics. See general Refs. [16–18]. Spinors in GA represent efficiently rotations in 3D space and are intimately related to quaternions from quaternion algebra. Here, the approach in Ref. [15] is further developed to generate a complete framework to manipulate biopolymer chains with spinors. We specifically apply the approach to proteins and show how different IC specifications for calculations can

be inferred from the general equations. Formal approximation has been greatly improved by reversing the order of the spinor operator products. It is also shown how rigid body motions can be superimposed on intramolecular motions and how the center of mass can be calculated with the bond vectors. We also highlight the connection between spinors and quaternions and show how to recover quaternion equations from within the GA framework. Finally, the approach is implemented to test its over-performance with respect to matrix methods, the PSC method, and the original spinor approach.

The main outline of the article is as follows. First, we develop the approach with all basic equations. Second, the relationship between spinors and quaternions is highlighted. Then the experimental results of actual computations are presented including comparative tests with state of the art algorithms. Finally, the main conclusions and some prospects for future research are given.

Theory

The current section forms the core of this article. It introduces spinors in GA as rotational operators and explains the notation and labeling rules here. Hereafter, it discusses how IC describe a protein structure. Next, the results from Ref. [15] are recapitulated and are the basis for the main derivations in this article. Finally, rigid body motions and chain construction are treated. Readers unfamiliar with GA are referred to Supporting Information.

P. Chys, P. Chacón
Structural Bioinformatics Group, Department of Biological Chemical Physics,
Institute of Physical Chemistry Rocasolano (IQFR), Consejo Superior de
Investigaciones Científicas (CSIC), Calle de Serrano 119, Madrid 28006, Spain
E-mail: pablo@chaconlab.org

Contract grant sponsor: Ministerio de Educación y Ciencia of Spain; Contract/grant number: BFU2009-09552; Contract/grant sponsor: Human Frontier Science Program; Contract/grant number: RGP003912008

© 2012 Wiley Periodicals, Inc.

Introduction to spinors

Spinors are four-component units from GA which encode efficiently spatial rotations. Spinors are sometimes also denoted rotors.^[14,17–19] A base spinor R is characterized by a rotation angle and a vector-like component, the latter representing the rotation plane and axis. Together with the corresponding reverse spinor R^\dagger , a pair of spinors is formed which constitutes the spinor operator \mathbf{R} . The reverse spinor is identical to the base spinor except the rotation angle that has opposite sign. The action of the complete spinor operator (pair) on a given vector rotates this vector in space along the specified rotation axis through the rotation angle of the base spinor. If ϕ denotes the rotation angle and $\hat{\mathbf{a}}$ the normalized rotation axis vector, the exact form of the base spinor $R = \alpha + \iota \mathbf{b}$ is:

$$R = \cos\left(\frac{\phi}{2}\right) + \sin\left(\frac{\phi}{2}\right) \iota \hat{\mathbf{a}} \quad (1)$$

with ι the pseudoscalar, which converts the vector basis of $\hat{\mathbf{a}}$ to a bivector basis. Though the rotation axis can be clearly inferred from Eq. (1), the last three spinor components -in bivector basis- rather encode the rotation plane than the axis itself. The reverse spinor R^\dagger uses $-\phi$ instead of ϕ :

$$R^\dagger = \cos\left(\frac{\phi}{2}\right) - \sin\left(\frac{\phi}{2}\right) \iota \hat{\mathbf{a}} \quad (2)$$

by virtue of the trigonometric functions. Since spinors are 4-tuples, we will sometimes represent them as $(\alpha, b_x, b_y, b_z) = (\alpha, \mathbf{b})$. The above equations can be compactified in the exponential analogues:

$$R = e^{\frac{\phi}{2} \iota \hat{\mathbf{a}}}, \quad R^\dagger = e^{-\frac{\phi}{2} \iota \hat{\mathbf{a}}} \quad (3)$$

Using both R and R^\dagger , a vector \mathbf{x} can be rotated:

$$\mathbf{x}' = R^\dagger \mathbf{x} R = \mathbf{R} \mathbf{x} \quad (4)$$

with \mathbf{R} the full spinor operator and \mathbf{x}' the rotated vector \mathbf{x} (Fig. 1). The notation $\mathbf{R} \mathbf{x}$ in Eq. (4) is allowed since the combined action of R and R^\dagger works as a linear operator.^[16–18] Obviously, $\mathbf{R} \mathbf{x}$ is an alternative for $A \mathbf{x}$ in $\mathbf{x}' = A \mathbf{x}$ with A being the typical rotation matrix.

Consecutive application of spinor operators \mathbf{R}_1 and \mathbf{R}_2 to a vector \mathbf{x} results in $\mathbf{x}' = R_2^\dagger R_1^\dagger \mathbf{x} R_1 R_2$ of which the RH side contains two spinor products. Spinor multiplication of a spinor R_1 with a second one R_2 yields a new spinor R_3 . The product $R_3 = R_1 R_2$ is equal to^[16]:

$$\alpha_3 + \iota \mathbf{b}_3 = (\alpha_1 + \iota \mathbf{b}_1)(\alpha_2 + \iota \mathbf{b}_2) \quad (5)$$

For completeness, $\alpha_i = \cos(\phi_i/2)$ and $\mathbf{b}_i = \sin(\phi_i/2) \hat{\mathbf{a}}_i$. The components α_3 and \mathbf{b}_3 can now be computed by^[16]:

$$\begin{cases} \alpha_3 &= \alpha_1 \alpha_2 - \mathbf{b}_1 \cdot \mathbf{b}_2 \\ \mathbf{b}_3 &= \alpha_1 \mathbf{b}_2 + \alpha_2 \mathbf{b}_1 - \mathbf{b}_1 \times \mathbf{b}_2 \end{cases} \quad (6)$$

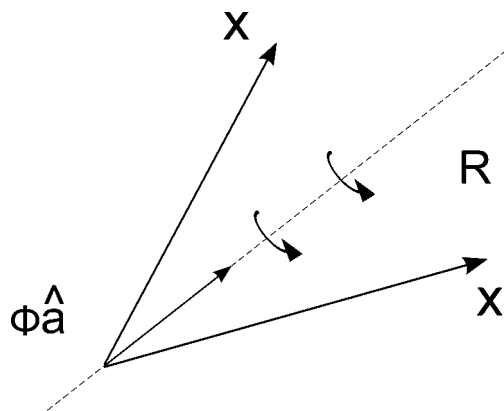


Figure 1. Action of a spinor operator \mathbf{R} on a vector \mathbf{x} . The operator \mathbf{R} performs a rotation of \mathbf{x} through an angle ϕ yielding the 1-vector \mathbf{x}' . A positive value of ϕ corresponds to a clockwise rotation if we look along the axis of the 1-vector $\phi \hat{\mathbf{a}}$ (right hand rule). Here ϕ is slightly less than $+180^\circ$.

in which the typical dot (\cdot) and cross (\times) products from Gibbs' vector algebra are used.^[20] It is observed that α_3 does not depend on the exact order of spinor operators in the product (\mathbf{R}_1 before \mathbf{R}_2) whereas \mathbf{b}_3 does by means of the cross product. Consequently, the complete product is noncommutative which corresponds with the fact that rotations in space do not form a commutative group. By means of Eq. (6) computation of product operators becomes trivial. Spinors resulting from spinor products will also be denoted as contracted spinors. The same terminology applies to the complete operators. Hence, spinor contraction is synonymous to spinor multiplication.

Protein labeling

We label the protein backbone atoms sequentially from zero to n , starting at the N-terminus. Bond i starts at atom $i - 1$ and ends at atom i (Fig. 2). Bond vector i is denoted by \mathbf{b}_i (Fig. 3) and the normalized vector by $\hat{\mathbf{b}}_i$. Valence angle θ_i is located at atom i and rotation (dihedral) angle ϕ_i corresponds to a rotation around bond \mathbf{b}_i . A spinor i is denoted by R_i , the reverse spinor by R_i^\dagger , and the full spinor operator by \mathbf{R}_i . Spinors and spinor operators, indexed by i , always describe rotation around the bond \mathbf{b}_i . These conventions are different from the ones in Ref. [15]. Spinor operators, constructed by backward propagation, will be denoted \mathbf{B}_i to distinguish them from forward spinor operators \mathbf{R}_i . In analogy, the base and reverse spinors are labeled B_i and B_i^\dagger . Furthermore, backward spinor products are abbreviated by bracket index notation: $B_{[k]} = B_1 B_2 \dots B_k$ where B_i may be further specified as well (e.g., $B_i = B_i^{(\phi)} B_i^{(\theta)}$).

The absolute position coordinate of atom i with respect to an arbitrarily chosen Cartesian reference frame is denoted by \mathbf{x}_i (Fig. 3). The vector $\Delta \mathbf{x}_{ij}$ is the vector joining coordinates \mathbf{x}_i and \mathbf{x}_j or $\Delta \mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$. Bond vector i can now be expressed as $\mathbf{b}_i = \mathbf{x}_i - \mathbf{x}_{i-1} = \Delta \mathbf{x}_{(i-1)i}$. The coordinates of the side chain atoms are in an analogous manner denoted by $\mathbf{x}_{k1}, \mathbf{x}_{k2}, \mathbf{x}_{k3}, \dots$ or shortly \mathbf{x}_{kj} (Fig. 3). The coordinate \mathbf{x}_k is equal to \mathbf{x}_{k0} and is the starting point of the side chain branch. The involved spinor operators

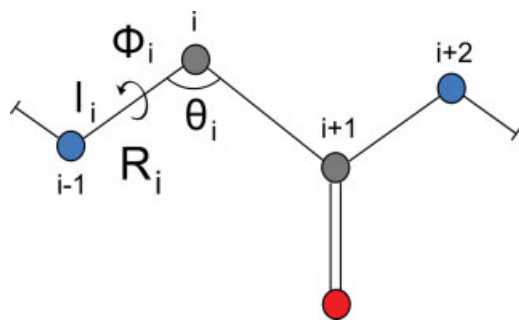


Figure 2. IC labeling of proteins. Atoms (i), bond lengths (l_i), dihedrals (ϕ_i), and valence angles (θ_i) as well as spinor operators R_i are depicted. A blue circle denotes a nitrogen atom, a gray one a carbon and a red one an oxygen atom. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

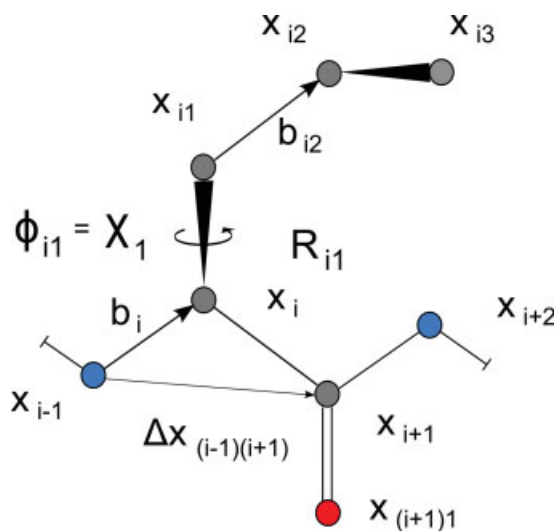


Figure 3. Vector labeling of proteins. Atom coordinates (x_i), bond vectors (b_i), and joint vectors (Δx_{ij}) are highlighted with respect to an arbitrary origin. Atom coordinates, bond vectors, and spinor operators in a side chain are denoted x_{ij} , b_{ij} , and R_{ij} with i and j the appropriate atom indices. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

are denoted $B_{k1}, B_{k2}, \dots, B_{km}$ with corresponding dihedral angles $\phi_{k1}, \phi_{k2}, \dots, \phi_{km}$.

Protein backbone IC set

A protein is a sequence of peptide units with a backbone formed by a repetition of N- C_α -C traces (Figure 4). The alpha carbon C_α is also denoted by CA below. The backbone can be characterized by the IC set $\{l_i, \theta_i, \phi_i\}$ of bond lengths, valence and dihedral angles. This set mainly depends on the geometrical properties of the peptide unit. In a peptide unit, starting from the N-terminus, we have bond lengths l_{N-CA} , l_{CA-C} , and l_{C-N} with corresponding valence angles θ_{C-N-CA} , θ_{N-CA-C} , and θ_{CA-C-N} . The corresponding protein dihedral angles^[21] are ω , φ , and ψ . The IC set $\{l_i, \theta_i, \phi_i\}$ can be rewritten in more detail as:

$$\{(l_{N-CA}, \theta_{C-N-CA}, \omega)_j, (l_{CA-C}, \theta_{N-CA-C}, \varphi)_j, (l_{C-N}, \theta_{CA-C-N}, \psi)_j\}$$

with a correctly modified index j . Similarly, IC sets $\{l_i, \theta_i, \phi_i\}$ apply to side chains (-R groups) with $\chi_i \equiv \phi_i$.

Application of spinor products

The rotation of a bond b_{i+1} through an angle ϕ_i around the preceding bond b_i is equal to:

$$b'_{i+1} = R_i^\dagger b_{i+1} R_i = R_i b_{i+1} \quad (7)$$

with b'_i the updated bond vector b_i , R_i the base spinor, R_i^\dagger the corresponding reverse spinor, and R_i the complete spinor operator.^[15] The exact forms of such R_i and R_i^\dagger in Eq. (7) are:

$$R_i = e^{\frac{\phi_i}{2} i \hat{b}_i}, \quad R_i^\dagger = e^{-\frac{\phi_i}{2} i \hat{b}_i} \quad (8)$$

As an example, we illustrate a spinor construction in the protein context. Let us rotate a C_α -C bond through a dihedral angle φ . Since the preceding bond is the N- C_α bond, normalization of bond vector b_{N-CA} yields \hat{b}_{N-CA} as rotation axis \hat{b}_i . The angle φ is clearly ϕ_i . As a result, the base spinor we seek is $R = \exp(\varphi/2 i \hat{b}_{N-CA})$ or $R = \cos(\varphi/2) + \sin(\varphi/2) i \hat{b}_{N-CA}$. Changing the sign of the last three components of the base spinor yields the reverse spinor R^\dagger . The spinor operator R is now available.

An updated atom coordinate x'_i can be calculated as follows:

$$x'_{i+1} = x_i + R_i b_{i+1} \quad (9)$$

Analogously, the rotation of more distant atoms j ($j > i$) around bond b_i can be expressed in terms of $\Delta x_{ij} = x_j - x_i$:

$$x'_j = x_i + R_i \Delta x_{ij} \quad (10)$$

Finally, it was derived that an atom q after m bond rotations along the polymer chain has coordinates:

$$x_q^{(m)} = x_i + R_i \Delta x_{ij} + R_j R_i \Delta x_{jk} + \dots + R_p R_o \dots R_i \Delta x_{pq} \quad (11)$$

in which the atoms indexes obey $i < j < \dots < p < q$ and $\Delta x_{ij}, \Delta x_{jk}, \dots$ are the join vectors spanning the subscripted atom positions. The spinor operators R_i, R_j, \dots always correspond to rotations around the bonds i, j, \dots and so on. Equation (11) summarizes well the application of GA and spinors to conformational manipulation of polymers as developed in Ref. [15]. A most useful equation is when the number of polymer segments m is maximal and equal to the number of bonds minus one:

$$x_n^{(n-1)} = x_1 + R_1 b_2 + R_2 R_1 b_3 + \dots + R_{n-1} R_{n-2} \dots R_1 b_n \quad (12)$$

in which each b_i can be thought of as $\Delta x_{(i-1)i}$ to see the relationship with Eq. (11). Equation (12) offers a practical scheme to calculate atom coordinates of a polymer structure and can be easily applied to a protein structure.

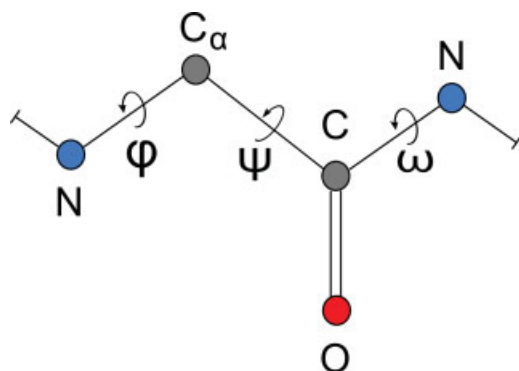


Figure 4. IC for proteins. The dihedrals φ , ψ , and ω are related to the correct bond lengths l and valence angles θ . See main text for more details.

Forward propagation through protein backbone. The basic action of the spinor operators in Eqs. (11) and (12) is highlighted in Figure 5. The subsequent action of two spinor operators R_1 and R_2 is considered. The net rotation of bond 3 is described by the third term in Eq. (12), namely $R_2 R_1 \mathbf{b}_3$. In Figure 5, bond 2 (\mathbf{b}_2) and 3 (\mathbf{b}_3) are first rotated around bond 1 (\mathbf{b}_1) by operator R_1 , yielding the green bonds 2' (\mathbf{b}'_2) and 3' (\mathbf{b}'_3). Next, bond 3' is rotated around bond 2' by spinor operator R_2 to obtain bond 3'' (\mathbf{b}''_3). The spinor operator R_2 can only be computed whence bond 2' has been computed. As a result, $R_2 R_1 \mathbf{b}_3$ in Eq. (12) can only be computed when $R_1 \mathbf{b}_2$ has yielded bond 2'. Consequently, an iterative scheme is necessary to compute Eq. (12) in case of forward propagation.

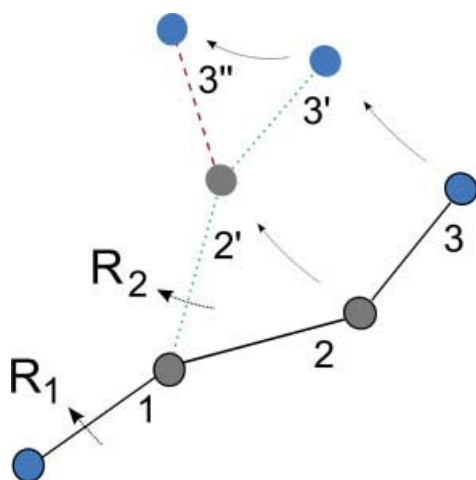


Figure 5. Forward propagation scheme to construct spinor operators R_1 and R_2 .

Backward propagation through the protein backbone. The conformational change highlighted in Figure 5 can also be performed by backward propagation through the protein backbone. Figure 6 illustrates this with the same example. Here, bond 3 first rotates around bond 2, yielding the green bond 3'. Next, bonds 2 and 3 are rotated around bond 1. Bond 1 is unaffected by the first rotation and spinor 2 can thus be constructed in advance. Denoting a backward propagation spinor (or simply backward spinor) by

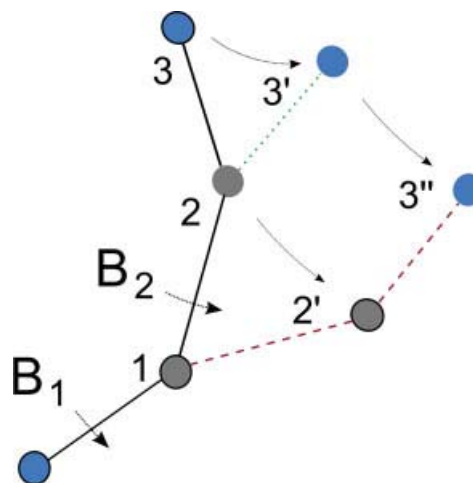


Figure 6. Backward propagation scheme to construct spinor operators B_1 and B_2 .

B , both B_1 and B_2 are known in advance and $B_1 B_2$ is immediately computable without the need for $B_1 \mathbf{b}_2$.

Such backward propagation scheme can easily be extended for the case of $n - 1$ bond rotations and results in the spinor series product:

$$B_1 B_2 \dots B_{n-2} B_{n-1} \quad (13)$$

By substitution of the spinor products, Eq. (12) becomes now:

$$\mathbf{x}_n^{(n-1)} = \mathbf{x}_1 + B_1 \mathbf{b}_2 + B_1 B_2 \mathbf{b}_3 + \dots + B_1 B_2 \dots B_{n-1} \mathbf{b}_n \quad (14)$$

in which the spinor products have reversed order. All of the rotation axes are known in advance and are given by the set $\{\hat{\mathbf{b}}_i\}$ with $i \in [1, n - 1]$. In line with Eqs. (11) and (12) it is trivial to derive that:

$$\mathbf{x}_q^{(m)} = \mathbf{x}_i + B_i \Delta \mathbf{x}_{ij} + B_i B_j \Delta \mathbf{x}_{jk} + \dots + B_i B_j \dots B_p \Delta \mathbf{x}_{pq} \quad (15)$$

in which protein backbone segments instead of individual bonds are rotated. Equations (14) and (15) have substantial advantage over Eqs. (11) and (12) since the reversed operator order allows computational spinor contraction before actually rotating the bond vectors. This results for Eq. (14) in:

$$\mathbf{x}_n^{(n-1)} = \mathbf{x}_1 + B_{[1]} \mathbf{b}_2 + \dots + B_{[n-1]} \mathbf{b}_n \quad (16)$$

with $B_{[i]}$ the contracted spinors, running from subscript index 1 to $n - 1$. The contraction of the involved operator products $B_1 B_2 \dots B_i$ to $B_{[i]}$ is performed by consecutive spinor products of basic form $B_{[i-1]} B_i = B_{[i]}$. The spinor operator $B_{[i-1]}$ is formed by similar recursion and ends at $B_{[1]} = B_1$. Starting at B_1 , we have thus, for example, $B_{[1]} = B_1$, $B_{[2]} = B_{[1]} B_2$, $B_{[3]} = B_{[2]} B_3$, and so on.

We conclude that Eqs. (14) and (15), the spinor product Eqs. (5) and (6) together with the exact spinor forms in Eq. (8) allow

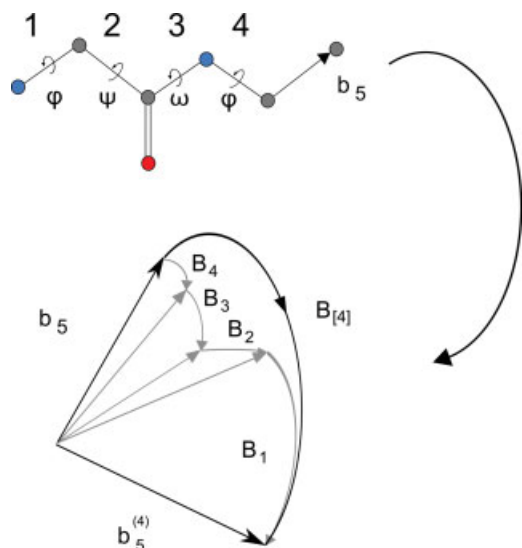


Figure 7. Contraction of spinor operators \mathbf{B}_j to $\mathbf{B}_{[j]}$. Gray arrows show the effect of single spinor operations on bond vector \mathbf{b}_5 but the black arrows illustrate the integrated effect of the contracted spinor operator $\mathbf{B}_{[4]}$.

for efficient computation of rotated bond vectors, also in case when multiple bond rotations in the preceding chain part are present (Fig. 7).

Side chains. A side chain can be treated as a branch from the protein backbone. However, the side chain together with the amino-terminal backbone part from which it branches also constitute a linear backbone. This allows for direct application of the equations derived above. We obtain thus for the side chain coordinates \mathbf{x}_{nl} :

$$\mathbf{x}_{nl}^{(n+l-1)} = \mathbf{x}_1 + \mathbf{B}_{[1]} \mathbf{b}_2 + \dots + \mathbf{B}_{[n-1]} \mathbf{b}_n + \mathbf{B}_{[n-1]} \mathbf{B}_{[n|0]} \mathbf{b}_{n1} + \dots + \mathbf{B}_{[n-1]} \mathbf{B}_{[n|l-1]} \mathbf{b}_{nl} \quad (17)$$

Herein, the $\mathbf{B}_{[ij]}$ are the contracted side chain spinors originating at backbone atom i and calculated from the individual side chain spinors \mathbf{B}_{ij} with running index j . This approach can in an analogous manner be applied to further branching with no qualitative differences. The tree structure of the protein has a one-to-one mapping with the tree of corresponding spinors.

Valence angle and bond length. Until now, the focus has been on the most important IC subset for proteins: the dihedral angle set $\{\phi_i\}$. It is now shown how changes in bond lengths and valence angles $\{l_i, \theta_i\}$ can be introduced naturally. First, Eq. (14) is written in slightly different notation:

$$\mathbf{x}_n^{(n-1)} = \mathbf{x}_1 + \mathbf{B}_1^{(\phi)} \mathbf{b}_2 + \mathbf{B}_1^{(\phi)} \mathbf{B}_2^{(\phi)} \mathbf{b}_3 + \dots + \mathbf{B}_1^{(\phi)} \mathbf{B}_2^{(\phi)} \dots \mathbf{B}_{n-1}^{(\phi)} \mathbf{b}_n \quad (18)$$

as to emphasize the fact that we deal with spinors performing dihedral rotations. A change in valence angle θ_i for a rotation around protein bond i corresponds to a spinor operator with as rotation axis the normal of the plane formed by the bonds $i-1$ and i . This is more clearly illustrated in Figure 8.

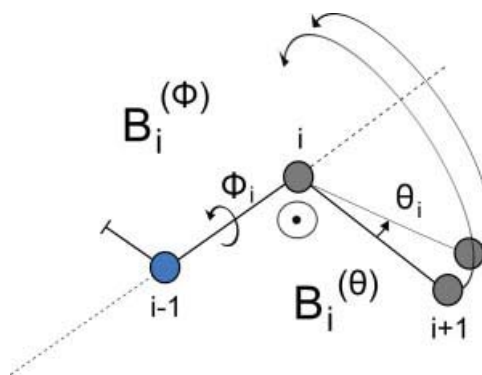


Figure 8. Illustration of a valence angle spinor $\mathbf{B}_i^{(\theta)}$. The spinor operator $\mathbf{B}_i^{(\theta)}$ rotates the protein bond $i+1$ through a valence angle change θ_i after which the dihedral spinor operator $\mathbf{B}_i^{(\phi)}$ executes a torsion ϕ_i of bond $i+1$. Applying $\mathbf{B}_i^{(\theta)}$ before $\mathbf{B}_i^{(\phi)}$ allows to construct the contracted spinor set $\{\mathbf{B}_{[ij]}\}$ in advance of actual bond rotations.

Computationally, the rotation axis is given by normalization of the normal vector $\iota \mathbf{b}_i \wedge \mathbf{b}_{i+1}$. If first a dihedral rotation is performed, the updated vector set will be needed. However, first performing the valence angle rotation before the dihedral rotation alleviates this problem and allows to use the initial bond vector set to construct the normal vectors (Fig. 8). In combination with the backward propagation scheme, the valence angle change can consequently be calculated without changing the basics of the already proposed method. To account in Eq. (18) for this extension, the operators $\mathbf{B}_i^{(\phi)}$ have to be substituted by $\mathbf{B}_i^{(\phi)} \mathbf{B}_i^{(\theta)}$ and hence we obtain:

$$\mathbf{x}_n^{(n-1)} = \mathbf{x}_1 + \mathbf{B}_1^{(\phi)} \mathbf{B}_1^{(\theta)} \mathbf{b}_2 + \mathbf{B}_1^{(\phi)} \mathbf{B}_1^{(\theta)} \mathbf{B}_2^{(\phi)} \mathbf{B}_2^{(\theta)} \mathbf{b}_3 + \dots + \mathbf{B}_1^{(\phi)} \mathbf{B}_1^{(\theta)} \dots \mathbf{B}_{n-1}^{(\phi)} \mathbf{B}_{n-1}^{(\theta)} \mathbf{b}_n \quad (19)$$

in which the order of the spinor products is crucial and the rotation axes of the spinors $\mathbf{B}_i^{(\theta)}$ are given by the normalized vector set $\{\iota \mathbf{b}_i \wedge \mathbf{b}_{i+1}\}$ (or $\{\mathbf{b}_{i+1} \times \mathbf{b}_i\}$ alternatively). Always, the initial bond vector set along the full protein backbone is used. The spinor products in Eq. (19) can be contracted and this yields in bracket notation:

$$\mathbf{x}_n^{(n-1)} = \mathbf{x}_1 + \mathbf{B}_{[1]} \mathbf{b}_2 + \mathbf{B}_{[2]} \mathbf{b}_3 + \dots + \mathbf{B}_{[n-1]} \mathbf{b}_n \quad (20)$$

These bracket spinors are the ones that are obtained on computation. Bond length changes can easily also be included since spinor operators preserve vector lengths (linear operators). Hence, if bond vector \mathbf{b}_i is scaled by a factor k_i , Eq. (20) can be modified to:

$$\mathbf{x}_n^{(n-1)} = \mathbf{x}_1 + k_2 \mathbf{B}_{[1]} \mathbf{b}_2 + k_3 \mathbf{B}_{[2]} \mathbf{b}_3 + \dots + k_n \mathbf{B}_{[n-1]} \mathbf{b}_n \quad (21)$$

in which a single bond operator is equal to:

$$k_{i+1} \mathbf{B}_{[i]} = k_{i+1} \mathbf{B}_2^{(\phi)} \mathbf{B}_2^{(\theta)} \mathbf{B}_3^{(\phi)} \mathbf{B}_3^{(\theta)} \dots \mathbf{B}_i^{(\phi)} \mathbf{B}_i^{(\theta)} \quad (22)$$

in which in principle the factor k_{i+1} could be written as a non-unit spinor with rotation angle equal to zero. Equation (22) thus

represents a single computable spinor which encodes all IC changes on a bond vector in a concise and effective way.

The approach here is in line with ideas to use spinor equations to formulate classical mechanics.^[16] The importance of the set $\{\mathbf{B}_i\}$ is not only to manipulate the bond vectors but also to act as manipulators of the spinors themselves. Physically, each spinor product is a rotation of the second spinor plane by the first spinor operator. Hence, spinor contraction is equivalent to operation of different spinor operators on a given end spinor and making a rotational integration on that spinor before rotating the actual bond vector. When the contracted spinor becomes available, a single spinor operation performs the complete effect of all spinor rotations on the given protein bond.

Rigid body motions

Here, proteins are considered as rigid body particles in relation to a laboratory-frame. Equations are derived for calculation of the center of mass and Euclidean motion of a single and multiple protein chains. Moreover, it is shown how superposition of internal and Euclidean motion can be accomplished.

Center of mass calculation. In certain types of calculation, the center of mass \mathbf{X} of a molecule may be needed. The construction of inertia moments for a protein particle, for example, relies on the relative coordinates $\mathbf{y}_i = \mathbf{x}_i - \mathbf{X}$ in which \mathbf{X} is subtracted from the absolute coordinates.^[16] In Appendix A, it is shown that the center of mass can be written as a function of the bond vector set $\{\mathbf{b}_i\}$:

$$\mathbf{X} = \mathbf{x}_0 + f_1 \mathbf{b}_1 + f_2 \mathbf{b}_2 + \cdots + f_n \mathbf{b}_n \quad (23)$$

with the coefficient f_i equal to:

$$f_i = \frac{1}{M} \sum_{j=i}^n m_j \quad (24)$$

The coefficient f_i is a cumulative mass fraction coefficient. It needs to be calculated once at the start of the computations. If the set $\{\mathbf{b}_i\}$ is updated, it follows from Eqs. (16) and (23) that

$$\mathbf{X} = \mathbf{x}_0 + f_1 \mathbf{b}_1 + f_2 \mathbf{B}_{[1]} \mathbf{b}_2 + \cdots + f_n \mathbf{B}_{[n-1]} \mathbf{b}_n \quad (25)$$

which shows that the center of mass \mathbf{X} can be computed in parallel with the calculation of $\{\mathbf{x}'_i\}$. Equation (25) is a weighted version of Eq. (16).

Euclidean motion. Rigid body motions can always be represented by a combination of rotations and translations and, such approach can be used to explicitly relate body-fixed CC $\{\mathbf{x}_i\}$ of a particle to their laboratory-frame coordinates $\{\mathbf{x}'_i\}$ (e.g., coordinates in absolute space). Generally, a translation \mathbf{t} of a protein followed by an external rotation \mathbf{R} around a point \mathbf{c} is equal to:

$$\mathbf{x}'_i = \mathbf{R}\mathbf{x}_i + \mathbf{R}(\mathbf{t} - \mathbf{c}) + \mathbf{c} \quad (26)$$

and such equation can be used to relate a local frame to a laboratory-frame. The spinor operator \mathbf{R} is a general one, not specifically a forward or backward one, and relates the protein conformation in the local frame to that in the laboratory-frame. The point of rotation \mathbf{c} is now chosen as the instantaneous center of mass \mathbf{X} of the protein and thereby eliminates the need to introduce explicitly a translation vector \mathbf{t} . Equation (26) becomes then:

$$\mathbf{x}'_i = \mathbf{R}\mathbf{y}_i + \mathbf{X} = \mathbf{y}'_i + \mathbf{X} \quad (27)$$

which shows that the transformed position coordinates can always be calculated from the transformed relative coordinates. We will now further show that Eq. (27) together with Eqs. (16) and (23), form the basis for a very interesting parallel set of equations for protein conformation calculations. To that purpose, we need to introduce the following Einstein summation-like convention:

$$\sum_{i=1}^n \mathbf{B}^{[i-1]} \mathbf{b}_i = \mathbf{B}^{[i-1]} \mathbf{b}_i$$

together with the introduced equivalence $\mathbf{B}_{[i-1]} \equiv \mathbf{B}^{[i-1]}$ which latter is only valid for this section. It is remarked that $\mathbf{B}^{[0]}$ is simply equal to the scalar one. This convention substantially compactifies the equations that follow. It allows rewriting Eq. (16) as

$$\mathbf{x}_n^{(n-1)} = \mathbf{x}_0 + \mathbf{B}^{[i-1]} \mathbf{b}_i \quad (28)$$

and for the center of mass \mathbf{X} we obtain:

$$\mathbf{X} = \mathbf{x}_0 + \mathbf{B}^{[i-1]} f_i \mathbf{b}_i \quad (29)$$

Replacing the atom index i from Eq. (27) by j the relative coordinates are now

$$\mathbf{y}_j^{(n-1)} = \mathbf{B}^{[i-1]} (h_j - f_i) \mathbf{b}_i \quad (30)$$

with h_j equal to one if $i \leq j$ and zero if $i > j$. When combining the former equations together with the rigid body equation (27) we arrive at:

$$\mathbf{x}'_j = \mathbf{x}_0 + \mathbf{R}\mathbf{B}^{[i-1]} (h_j - f_i) \mathbf{b}_i + \mathbf{B}^{[i-1]} f_i \mathbf{b}_i \quad (31)$$

which is the equation we are seeking that integrates rigid body motion and internal motion in a straightforward manner.

Description of multiple chains. We now extend Eq. (27) to describe multiple protein chains together. Two options exist: the first is to express all chains in an absolute manner, the second to work with an appropriately chosen reference chain. Both approaches are illustrated for the case of two proteins with initial coordinates $\{\mathbf{x}_i^{(1)}\}$ and $\{\mathbf{x}_i^{(2)}\}$.

In the absolute approach, each chain receives an attached bodyframe which is coupled to the laboratory-frame (absolute

space) by the appropriate spinor. Based on Eq. (27), we obtain the following set of equations:

$$\begin{cases} \mathbf{x}_i^{(1)} = \mathbf{R}_1 \mathbf{y}_i^{(1)} + \mathbf{X}^{(1)} \\ \mathbf{x}_i^{(2)} = \mathbf{R}_2 \mathbf{y}_i^{(2)} + \mathbf{X}^{(2)} \end{cases} \quad (32)$$

with all variables as defined before and spinor operators \mathbf{R}_1 and \mathbf{R}_2 being distinct and valid for the corresponding chain.

In the relative approach, we seek to express the motion of chain 2 with respect to that of chain 1. We, therefore, aim to introduce a vector \mathbf{t}_{12} and a spinor operator \mathbf{R}_{12} , describing the relative translation and rotation, respectively. The equation set (32) can be used as a basis for the derivations. The translation vector \mathbf{t}_{12} can be introduced by the substitution $\mathbf{X}^{(2)} = \mathbf{X}^{(1)} + \mathbf{t}_{12}$. The spinor \mathbf{R}_2 in Eq. (32) can now be seen as a combination of two rotations, namely a rotation \mathbf{R}_{12} to link the body frame of protein 2 with that of protein 1 and a second one \mathbf{R} relating the body frame of protein 1 with the laboratory-frame:

$$\begin{cases} \mathbf{x}_i^{(1)} = \mathbf{R} \mathbf{y}_i^{(1)} + \mathbf{X}^{(1)} \\ \mathbf{x}_i^{(2)} = \mathbf{R} \mathbf{R}_{12} \mathbf{y}_i^{(2)} + \mathbf{X}^{(1)} + \mathbf{t}_{12} \end{cases} \quad (33)$$

Both of the foregoing equation sets can be extended for internal motion inclusion by introducing $\{\mathbf{x}_i^{(k)}, \mathbf{y}_i^{(k)}\}$ and either $\{\mathbf{R}_k, \mathbf{X}^{(k)}\}$ (absolute approach) or $\{\mathbf{R}_{1k}, \mathbf{t}_{1k}\}$ (relative approach). Furthermore, the introduction of internal motions [Eq. (31)] is also applicable along the lines as presented in the previous section.

Chain construction

Equation (21) uses the existing bond vector set $\{\mathbf{b}_i\}$ to obtain new atom coordinates and constitutes the basis for an algorithm to update protein conformations. However, Eq. (21) can also be cast to construct a protein conformation from scratch without the bond vector set $\{\mathbf{b}_i\}$ being available. Basically, this is done using an alternative vector set which can be arbitrarily chosen. Consider a large line segment along the X-axis, divided in $n + 1$ segments of length one. These segments have bond vectors σ_1 . The last segment $n + 1$ is first rotated by a spinor operator $\mathbf{B}_n^{(\theta)}$ and thereafter by $\mathbf{B}_n^{(\phi)}$. The $\mathbf{B}_n^{(\theta)}$ has either Y-axis (0, 1, 0) or Z-axis (0, 0, 1) as rotation axis whereas $\mathbf{B}_n^{(\phi)}$ has the X-axis as rotation axis (1, 0, 0). Both spinor operators establish the geometrical relation of the bond to the preceding one (σ_1 vector). Application of such procedure for a protein by iterating toward its N-terminus yields the absolute orientation in space of a unit bond. The rotated σ_1 vector is now multiplied with the correct bond length. All bonds can be calculated in this way but in the implementation the bonds are calculated in an integrated and forward manner. The backbone is constructed starting from the N-terminus using the recursive relations of the backward spinors. Taking the Z-axis case for the $\mathbf{B}_n^{(\theta)}$ spinor operator yields for the spinors:

$$\begin{cases} \mathbf{B}_n^{(\theta)} = (\cos(\theta/2), 0, 0, \sin(\theta/2)) \\ \mathbf{B}_n^{(\phi)} = (\cos(\phi/2), \sin(\phi/2), 0, 0) \end{cases} \quad (34)$$

These spinors turn out to correspond to the quaternion forms derived in Ref. [11]. In fact, the quaternion algorithm in that paper is recovered here in GA. It should be mentioned that here the construction algorithm is derived as a specific subcase of the more general equations. In Ref. [11], further advantage is now taken of quaternion-matrix conversion since σ_1 has only one component. Since spinors are isomorphic to quaternions^[16,18] and GA encapsulates them, the typical matrix form derived from quaternions is also immediately valid for spinors. This matrix is in fact a simple rewriting of the quaternion product or the spinor operator product in matrix format (see Appendix B):

$$2 \begin{bmatrix} q_0^2 + q_1^2 - 1/2 & q_1 q_2 - q_0 q_3 & q_1 q_3 + q_0 q_2 \\ q_1 q_2 + q_0 q_3 & q_0^2 + q_2^2 - 1/2 & q_2 q_3 - q_0 q_1 \\ q_1 q_3 - q_0 q_2 & q_2 q_3 + q_0 q_1 & q_0^2 + q_3^2 - 1/2 \end{bmatrix} \quad (35)$$

with (q_0, q_1, q_2, q_3) the quaternion. For spinors the exact matrix is:

$$2 \begin{bmatrix} \alpha^2 + b_x^2 - 1/2 & b_x b_y - \alpha b_z & b_x b_z + \alpha b_y \\ b_x b_y + \alpha b_z & \alpha^2 + b_y^2 - 1/2 & b_y b_z - \alpha b_x \\ b_x b_z - \alpha b_y & b_y b_z + \alpha b_x & \alpha^2 + b_z^2 - 1/2 \end{bmatrix} \quad (36)$$

Spinor/quaternion-matrix conversion is useful when the bond vector has components with value zero and/or multiple bond vectors have to be rotated with the same spinor/quaternion. In that case, the redundancy of the full geometric/quaternionic products is avoided yielding more efficient computations.

Overall, the scheme for chain construction can be divided in five steps: (i) Computation of spinors, (ii) application of the spinor products, (iii) spinor-matrix conversion, (iv) bond retrieval, and (v) atom coordinate calculation. The exact details of the scheme will now be summarized and recapitulated. In Eq. (21), all \mathbf{b}_i are set equal to σ_1 . Atom 0 is put at the origin, \mathbf{x}_1 assigned $l_{N-CA}\sigma_1$, and the start spinor $B_{[0]}$ initialized to (1, 0, 0, 0). An iterative cycle starts now with the calculation of bond 2. To that purpose, $B_1^{(\theta)}$ and $B_1^{(\phi)}$ are created [Eq. (34)] (i) and multiplied [Eq. (6)] separately with the initialized $B_{[0]}$ (ii). The new spinor $B_{[1]}$ is converted with Eq. (36) into a matrix (iii) but only the first column elements are generated since we only need to multiply with $l_2\sigma_1$ (iv). Bond vector 2 is now available and addition to the previous atom coordinate yields the position coordinate for atom 2 (v). This procedure is iterated until the end of the chain. After the first cycle, the $B_{[i]}$ with growing index i is always passed on in the iterations. The described approach basically applies to a linear chain but can be trivially extended for a fully branched chain structure such as a protein. In that case, a tree of spinors must be computed in a one-to-one correspondence with the real protein structure but the computational scheme remains exactly the same. In each branch (side chain), the computations proceed as in a linear chain.

The above spinor scheme is advantageous for two reasons. First, the spinors $B_i^{(\theta)}$ and $B_i^{(\phi)}$ contain each two components which are zero and thus reduce the operation count. Second, $\mathbf{b}_{i+1} = l_{i+1}\sigma_1$ also has two zero components which further improve the efficiency of the method.

Spinors Versus Quaternions

Spinors from GA and quaternions are isomorphic and thus closely interrelated.^[16–18,22] Quaternions were discovered as an independent algebra by Hamilton^[16,17] and have several advantages over matrices in representing rotations (e.g., no gimbal lock). Some fields such as computer graphics do use extensively quaternions as their specific advantages cannot be neglected.^[23] Nonetheless, quaternion algebra has always stood a little apart from main stream mathematics. Spinors, on the other hand, follow naturally from derivations in the larger context of GA and can be easily related to complex numbers at the conceptual level. It seems more natural, therefore, to see quaternions as entities derived in GA.

A unit quaternion \mathbf{q} (norm $\|\mathbf{q}\| = 1$) is a 4-tuple:

$$\mathbf{q} = q_0 + \bar{\mathbf{q}} \quad (37)$$

with q_0 a scalar and $\bar{\mathbf{q}}$ a vector part: $\bar{\mathbf{q}} = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ with base $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$. The vector $\bar{\mathbf{q}}$ represents the unit axis vector of the rotation represented by the quaternion. The base $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ for quaternions is related to the unit 2-blades of the spinor s by:

$$\mathbf{i} = -\iota\sigma_1, \mathbf{j} = -\iota\sigma_2, \mathbf{k} = -\iota\sigma_3 \quad (38)$$

Strictly speaking for spinors, the bivector components are unit 2-blades and represent three orthogonal planes. Using these bivector components rotation planes can be presented and because of the duality properties (using the pseudoscalar ι) they also directly encode the normal vector to that rotation plane. Using \mathbf{q} as reference quaternion, the corresponding spinor s is now:

$$s = q_0 - q_3\sigma_1\sigma_2 - q_1\sigma_2\sigma_3 - q_2\sigma_3\sigma_1 \quad (39)$$

Beware that the exact order and definition of the unit bivectors allow for different implementations of Eq. (39). The proof of matrix forms (35) and (36) in appendix B, for example, is based on a different mapping between spinors and quaternions.

The quaternion product \mathbf{pq} can be considered as a geometric product \mathbf{ab} with \mathbf{a} and \mathbf{b} spinors in this case. The main advantage of GA is that the geometric product is generally valid for different types of multivectors. Spinors can thus be multiplied with simple vectors but also with bivectors, trivectors, or even higher analogues. In that view, the geometric product \mathbf{ab} is only one of the many types possible in the broad and powerful framework of GA. This is not the case with quaternions: the quaternion product is restricted to pure quaternions as rotational entities and vectors brought into quaternion format. Another important advantage of spinors is that spinors in the standard model for 3D naturally extend to more advanced models such as the projective and conformal GA models,^[18] allowing the user to take advantage of these models.^[24]

Comparative Results

The spinor methodology was compared with the state of the art algorithms to manipulate IC. Three common and real scenarios for

retrieving the CC from IC were tested: construction from scratch of a polypeptide backbone (backbone construction), updating backbone coordinates from an existing bond vector set (backbone updating), and construction of a complete polypeptide (full chain construction). Method comparison for backbone construction was done with 3×3 matrices,^[11] the PSC method,^[14] the original spinor approach^[15] (denoted spinor forward method or spinor FW method), and the self-normalizing natural extension reference frame method^[13] (SN-NeRF method). Briefly, the matrix method is based on orthogonal rotation matrix theory whereas the PSC method uses the two preceding bond vectors and IC to generate a conformation. The spinor FW approach follows a similar scheme as the PSC method but uses spinors and the geometric outer product. The SN-NeRF method is still another forward method, which combines a local frame representation in spherical coordinates with an appropriate frame rotation by a 3×3 rotation matrix to obtain bond and atom coordinates. For backbone updating, spinors are compared with classical rotation matrices which are based on an angle-axis description.^[10,12,23] For full chain construction, the backward spinor methodology is compared with the corresponding matrix framework and the algorithms are derived from the backbone construction algorithms. See Supporting Information for a short survey of the conformational methods.

All methods were thoroughly implemented and validated. Systematic tests have been carried out to assess their efficiency as well as to correlate with previous and available analyzes.^[10–15] For the backbone algorithms, we performed experiments for recovering the atom coordinates of polypeptide chains, generated by randomly choosing their backbone dihedral angles (φ , ψ , and ω). The ω angle changes could only have maximum amplitude of 5° around the trans position ($\omega = 180^\circ$). The size of the polypeptide backbone (333 amino acid residues, ≈ 1000 atoms) was chosen as to approach a medium protein size. Each individual timing covered a thousand of such chain calculations as to have a sufficiently large time value (ms range). This procedure was repeated 1000 times to have a good sampling statistic. Results were cross-checked by doing runs with smaller and bigger backbone sizes (10–30,000 residues) yielding equivalently scaled timings. The performances of the full chain construction algorithms were tested with a complete polylysine chain of 333 residues under almost identical test conditions. The versions we implemented account for changes in the complete IC set of both backbone and side chains. Each individual polylysine computation, the full internal coordinate set was altered with in case of the valence angles and bond lengths small random deviations from the equilibrium values.

In addition, the computational efficiency of the methods was examined theoretically by counting the number of arithmetic operations needed to be performed in a single step. A single step is the calculation of the CC of one atom.

Backbone construction

In Table 1, the estimated computational cost for protein backbone construction is shown for all tested methods.

Table 1. Estimated computational efficiency of different methods for protein backbone construction.

Method	Computational cost							Total
	×	±	÷	÷2	T	M	f_{norm}	
Spinor	26	15	0	2	4	4	0	135
Matrix	27	15	0	0	4	6	0	158
Polyspherical	23	13	3	0	4	6	1/6	160
Spinor FW	45	20	0	1	4	6	1	195
SN-NeRF	26	15	0	0	4	9	1	207

Operation counts apply to the calculation of CC of a single added atom. We estimated the relative weights with an in house program for the operations $\times/\pm/\div/T/M/\sqrt{}$ as 1/1/2/11/12/3 with T and M referring to trigonometric functions (sin, cos) and memory load. These values are in good agreement with Ref. [14]. Division by a constant two ($\div 2$) has a reduced relative weight of 1 versus 2 for a regular floating-point division. The vector normalization frequency f_{norm} corresponds with the number of normalizations added per atom. A normalization needs $3 \times /2 \pm /3 \div /1 \sqrt{}$ operations yielding an extra total cost of 14. Abbreviations: spinor FW = spinor forward method, SN-NeRF = self-normalizing natural extension reference frame method.

The computational cost of arithmetic operations (\times , \pm , \div , $\div 2$) for spinors corresponds to that given for quaternions in Ref. [11]. As mentioned before, spinors and quaternions are isomorphic and should in principle yield the same numbers.^[18] The arithmetic counts for rotation matrices are also in agreement with the literature.^[11] For the PSC method, our analysis includes the count to obtain the atom coordinate from the computed bond vector on top of the estimates in the original paper.^[14] We add thus $3 \pm$ and $3 \times$ to retrieve the actual atom coordinates from the bond coordinates. For the spinor forward method,^[15] a new analysis revealed that the \pm count from the original paper must be adapted: the count for the spinor operator product with the constructed bond vector is two higher than originally deduced (20 instead of 18). In our implementation of the SN-NeRF method, we obtain 26 multiplications and 15 addition operations which is even less than in the original work.^[13] All five methods have the same cost in computation of trigonometric functions (T), namely 4T. In line with the analysis in Ref. [14], we estimate a memory transfer load (M) for the methods. This does not refer to pure memory allocation since all arrays are preallocated in the algorithms. Rather, it deals with relative amounts of memory reads and writes in the codes. In principle, the number of data elements that must be passed on to the next iteration furnishes a good estimate. This yields 4/9/8/6/6 for the methods along the top-down order in Table 1. The accumulated backward spinor and matrix for example, have 4 (4M) and 9 (9M) elements, respectively. Both the spinor FW and SN-NeRF method pass on two vectors (6M) whereas the PSC method includes a further two (2M) for stored trigonometric values. We did not find good correlation with experimental results so that we made new estimates based on the largest working memory units in the codes. For backward spinors, this is four (4M) whereas the optimized matrix schemes calculates 3×2 arrays (6M) when updating the accumulated matrix.^[11] For the PSC and spinor FW method, two bond vectors are the largest units (6M) whereas the SN-NeRF method has a 3×3 array (9M) implemented as largest unit. This results in 4/6/6/6/9 as estimate in Table 1. In addition, in all of

the forward method codes we needed to introduce an explicit normalization step which must be included in an estimate (see Table 1 for details). For the PSC method only one normalization per six cycles was needed which agrees with the original implementation in Ref. [14]. Otherwise computations failed for backbone sizes of more than 60 atoms. In the spinor FW and SN-NeRF method, normalization needed to be done each iteration, hence yielding more overhead as compared to the PSC method. All of the normalizations were related to vector products in the codes and we believe that in the PSC method the implicit normalization in the main equation (see Supporting Information) allows for a lower normalization frequency. Nevertheless, this problem of normalization in the forward methods seems to indicate an intrinsic numerical weakness as compared to the backward methods. Backward spinors and matrices did not have this problem. Even for the largest backbone sizes tested (30 000 atoms) no normalization of the accumulated rotational operator was needed in both cases. Based on the estimates, the spinor method is the fastest for protein backbone construction and this is mainly due to the compactness of spinors (4M). Matrices and the PSC method are somewhat slower whereas the spinor FW and SN-NeRF methods are predicted to be 40-50% slower than backward spinors. The estimates also indicate that backward methods should have the edge on forward methods.

The experimental timings for backbone construction are shown in Table 2 which contains the results for the Gnu- and Intel-compiled source codes, both with (level -O3) and without standard optimization (level -O0). For the Intel -O3 versions, spinors are around 15% faster than matrices and 25% than the PSC method. The spinor FW and SN-NeRF methods lag further behind the PSC approach. The relative timings for the Gnu -O3 compiled versions agree well with those of the Intel counterparts but are on an absolute basis much slower. In addition, all the -O0 compiled executables yield virtually the same relative performances among the methods, pointing out consistent results among the differently compiled versions. Despite the basic assumptions about the memory load, the estimates are in good agreement with the experimental timings.

When considering now the absolute timings, the -O3 compiled versions are substantially faster than their -O0 compiled

Table 2. Experimental timings for construction of a 1000-atom protein backbone, iterated 1000 times with four different mathematical methods ($n = 1000$).

Method	Gnu (ms)		Intel (ms)		Cost (a.u.) Estimate
	-O0	-O3	-O0	-O3	
Spinor	142.6 ± 0.7	103.6 ± 0.5	125.0 ± 0.0	45.4 ± 0.2	135
Matrix	201.5 ± 0.6	120.5 ± 0.6	154.4 ± 0.1	53.2 ± 0.0	158
Polyspherical	205.1 ± 0.7	131.7 ± 0.7	150.5 ± 0.2	61.0 ± 0.1	160
Spinor FW	219.9 ± 0.5	145.0 ± 0.5	182.5 ± 0.2	78.3 ± 0.0	195
SN-NeRF	278.1 ± 0.6	156.3 ± 0.6	238.5 ± 0.1	82.0 ± 0.0	207

Both the Intel and Gnu C++ compiler were used with (-O3) and without optimization (-O0). Implementation estimates from Table 1 in arbitrary units (a.u.) are included for comparison. Abbreviations: spinor FW = spinor forward method, SN-NeRF = self-normalizing natural extension reference frame method.

counterparts. Furthermore, the optimized Intel executables are substantially faster than the Gnu counterparts. In the optimized versions, it is observed that the differences between the different methods are minimized. Concentrating on, for example, the fastest executables (Intel -O3 compiled), the gap between spinors and matrices is substantially narrowed as compared to the nonoptimized version inasmuch that the difference can be neglected on practical terms if speed is not of the uttermost importance. Compilers (especially Intel compiler in an Intel CPU) are able to exploit all kind of techniques to reduce the number of cycles per operation including parallelization and hence the performance gap.

In summary, the spinor method yields the fastest implementation but the differences are not that much with matrices and the PSC method. Both the spinor FW and SN-NeRF methods are the slowest methods and 30–50% slower than backward spinors. The performance differences are related to the memory transfer loads in the actual implementations and the need for explicit vector normalizations. In practice, the compiler choice allows a larger performance gain than choosing the fastest method when working with a given compiler. It can be noticed, for example, that the Intel -O3 compiled SN-NeRF method is 20% faster than the Gnu -O3 compiled backward spinor method.

Backbone updating

For updating of a protein backbone, both spinors and matrices were implemented. The algorithms we implemented for our test only accounted for dihedral angle changes. Bond lengths and valence angles were considered to be fixed. This type of test is similar to that in Ref. [12] but in this section we solely focus on the backbone structure. To correlate well with the estimated total costs all three dihedrals were changed (φ , ψ , and ω) during the calculations. Tables 3 and 4 show the theoretical estimates for the computational cost and experimental timings, respectively.

Table 3. Estimated computational efficiency for protein backbone updating with spinors and matrices.

Computational cost						
Method	×	±	÷2	T	M	Total
Spinor	41	30	1	2	4	142
Matrix	51	37	0	2	9	218
See Table 1 for details.						

Table 4. Experimental timings for updating of a 1000-atom protein backbone, iterated 1000 times ($n = 1000$).

Method	Gnu (ms)		Intel (ms)		Cost (a.u.) Estimate
	-O0	-O3	-O0	-O3	
Spinor	223.1 ± 0.5	98.1 ± 0.5	198.4 ± 0.2	70.0 ± 0.1	142
Matrix	438.4 ± 1.3	136.5 ± 0.9	393.5 ± 1.3	75.3 ± 0.1	218
Estimates from Table 3 in arbitrary units (a.u.) are included for comparison. See Table 2 for details.					

The theoretical estimates yield a clear advantage for spinors. Except for the trigonometric part which has equal counts, all the important counts are in favor of spinors. As a result, the total estimated count of the spinors is approximately 65% of that of matrices. However, the actual implementations show a similar trend as is the case for the backbone construction algorithms. The spinor method performs better but only slightly. The Intel -O3 compiled code for spinors uses only 93% of the time that matrices need which is a marginal difference. From the practical point of view, the differences are thus negligible unless dealing with very large molecular systems in which conformational computations constitute the bulk of the time. For the fastest code, the results here are in good agreement with those in Ref. [12] where it is found that quaternions (isomorphic to spinors) perform slightly better than rotation matrices. As in the case of the construction methods, the Intel compiler yields faster optimized code than the Gnu compiler albeit the difference is less for spinors (only 30%) than for matrices (double as fast). The timings for nonoptimized executables and that of the Gnu -O3 compilation seem to be much more in line with the theoretical estimates. As such, Table 4 for manipulation algorithms gives substantial evidence that the Intel compiler uses advanced tricks to improve code performance.

Full chain construction

For the construction of a complete polypeptide chain—including side chains, hydrogen, and carbonyl atoms—we implemented both the backward spinor and matrix schemes. As was stated before, the computation of side chains can be done by an extension of the linear backward scheme to tree-like structures. The full chain construction implementations are thus derived from the backbone construction algorithms for spinors and matrices. A polylysine was chosen since lysines are side chains large enough to assess the influence of branching on the linear scheme.

Table 5 shows the theoretical estimates for spinor and matrix implementations, both for terminal and nonterminal atoms. For backward spinors there is no difference between terminal and nonterminal atoms. The four spinor components must always be calculated for terminal atoms, even if the spinor does not need to be chained further. The total estimate per atom in a polypeptide is, therefore, 135 for spinors. However, for terminal atoms in the

Table 5. Estimated computational efficiency of backward spinors and matrices for a complete polypeptide construction.

Computational cost							
Method	Atom	×	±	÷2	T	M	Total
Spinor	Nonterminal	26	15	2	4	4	135
	Terminal	26	15	2	4	4	135
	Average						135
Matrix	Nonterminal	27	15	0	4	6	158
	Terminal	14	9	0	4	6	139
	Average						146
Separate counts included for nonterminal and terminal atoms. Average count for matrices apply to a polylysine chain. Weights are 8/22 and 14/22 for nonterminal and terminal atom counts, respectively. See text and Table 1 for details.							

matrix method only the first matrix column must be calculated instead of the complete matrix. This results for terminal atoms in a reduction of 13 multiplications and 6 additions as compared to a nonterminal atom count. This reduction in arithmetic operations for matrices was also discussed in Ref. [11] but not used in making estimates. These reduced counts result for the backward matrix method in an estimated average cost of 139 for terminal atoms. This is around 10% less than for nonterminal atoms (158). The average count for matrices must now take into account the number of terminal and nonterminal atoms in a peptide residue. In the illustrative case of lysine, we have eight nonterminal and 14 terminal atoms which results in an estimate of 146 per atom ($158 \cdot 8/22 + 139 \cdot 14/22 \approx 146$). This value is closer to the spinor estimate than in the pure backbone case. In general, a lysine residue has a high terminal/nonterminal atom ratio (14/8) favoring slightly the matrix method *versus* an average polypeptide chain. But, even for a glycine residue (4/3) with the lowest ratio the total cost is only 147. The estimate of 146 for lysine is thus a good estimate for all peptide residues. In conclusion, the theoretical estimates for backward spinors and matrices show that for the full chain algorithms both methods are closer in performance than in the backbone case.

Table 6. Experimental timings for construction of a complete 333-residue polylysine chain iterated 1000 times ($n = 1000$).

Method	Gnu (ms)		Intel (ms)		Cost (a.u.) Estimate
	-O0	-O3	-O0	-O3	
Spinors	1386.7 ± 1.0	1002.1 ± 0.4	1046.8 ± 0.3	371.8 ± 0.1	135
Matrices	1747.5 ± 0.2	1199.7 ± 0.3	1107.2 ± 0.3	420.4 ± 0.1	146

Estimates from Table 5 in arbitrary units (a.u.) are included for comparison. See Table 2 for details.

The results of the test runs are shown in Table 6. The given timings are valid for both alteration of the complete IC set and the dihedral subset at each polylysine calculation. The estimates agree relatively well with the experimental timings except for the data from the Gnu -O0 compiled versions. In the latter case, the matrices perform substantially more under par as to what is expected. For the Intel -O0 compiled versions on the other hand, matrices perform nearly as good as spinors. Also here, the Intel versions show significantly better performance than their Gnu counterparts which observation is critical if performance matters in conformational computations. In addition, it is remarked that the spinor/matrix timing ratios in the backbone and full chain algorithms do not differ that much for the fastest (-O3 compiled) codes. This indicates that the tree-like algorithm structure and the more advantageous terminal atom computation for matrices do not alter much the basic performance characteristics of the linear matrix and spinor scheme. Specifically for the Intel -O3 compiled versions, it is even noticed that the atom number ratios full chain/backbone ($7326/999 = 7.3$ for polylysine) correspond relatively well with the timing ratios (e.g., spinors: $370/45.4 = 8.2$) indicating little overhead from algorithmic branching itself. The results here illustrate well that

spinor Eq. (21) can be implemented and yields a very efficient computational framework for protein calculations. Spinors perform at least as good as matrices. It is not unlikely that with further tweaking and optimization performance differences can be further minimized.

Computational details

For the implementation, we programmed in C++ and computations were done in Linux (centOS 5.5) on a 64-bit machine. An Intel(R) Core (TM) i7 CPU 950@3.07 GHz processor with 12 Gb RAM was used. The programs were compiled with both the Intel (version 12.0) and Gnu (version 4.1.2) C++ compilers. Compilations were done both at -O3 and -O0 level optimization with default settings for each of the compilers. The code for the implemented algorithms and the tests are freely available on request.

Conclusions

Here, we presented a complete GA framework for effective manipulation of protein conformations. Spinors and their products are an attractive alternative to represent bond rotations. We demonstrated how the backward propagation of the conformational change through the protein backbone generates all the correct spinors in advance. And, how such spinors can act upon the individual protein bonds to efficiently retrieve atomic coordinates from IC. In real implementations, spinors are slightly more efficient for protein manipulation than the other standard methods. This applies to all tested cases: backbone construction, backbone updating and polypeptide construction. The profits over the best of the rest range from 5–15% in our tests. For backbone and full chain construction the test results are well in agreement with the estimates. However, the spinor overperformance *versus* matrices is below theoretical expectations for the updating algorithms. Here, optimization and parallelization causes alike performances among spinors and matrices. From the practical point of view, compiler choice proves to be critical to obtain performant executables for all of the examined methods. Spinors are the most costless methodology and are promising tools for conformational calculations in the protein field. Furthermore, an optimized construction algorithm is obtained based on the more general equations and it is shown that we recover existing quaternion equations. The intimate relationship between spinors and quaternions is highlighted as to point out the possible benefit for GA from results in quaternion research.^[25,26] As well GA and spinors are promoted since they constitute the more general framework than quaternion algebra.

Several aspects of the current approach are interesting for further research. A small own preliminary study points out that parallelization of the spinor approach is possible and publication^[27] already briefly touches on the subject in the framework of conformal GA. The conformal GA itself also constitutes an interesting field to be explored in view of protein calculations. In robotics, the potential of conformal GA has already been illustrated.^[24,28]

Appendix: Mathematical Proofs

Appendix A: Center of Mass Calculation

The center of mass \mathbf{X} can be derived in function of bond vectors \mathbf{b}_i as follows. The center of mass \mathbf{X} for a polyatom system such as a protein is equal to:

$$\mathbf{X} = \frac{1}{M} \sum_{i=0}^n m_i \mathbf{x}_i \quad (\text{A1})$$

with m_i the mass of atom i and M the sum of all masses: $M = \sum_{i=0}^n m_i$. If we expand this equation we obtain:

$$\mathbf{X} = \frac{1}{M} [m_0 \mathbf{x}_0 + m_1 \mathbf{x}_1 + \dots + m_n \mathbf{x}_n] \quad (\text{A2})$$

which can be rewritten as:

$$\mathbf{X} = \frac{1}{M} [m_0 \mathbf{x}_0 + m_1 \mathbf{x}_0 + m_1 \mathbf{b}_1 + m_2 \mathbf{x}_0 + m_2 \mathbf{b}_1 + m_2 \mathbf{b}_2 + \dots + m_n \mathbf{x}_0 + m_n \mathbf{b}_1 + \dots + m_n \mathbf{b}_n] \quad (\text{A3})$$

The bond vectors \mathbf{b}_i and \mathbf{x}_0 can now be collected:

$$\mathbf{X} = \frac{1}{M} [M \mathbf{x}_0 + (m_1 + m_2 + \dots + m_n) \mathbf{b}_1 + (m_2 + \dots + m_n) \mathbf{b}_2 + \dots + (m_n) \mathbf{b}_n] \quad (\text{A4})$$

and results in:

$$\mathbf{X} = \mathbf{x}_0 + \sum_{i=1}^n \left(\sum_{j=i}^n m_j / M \right) \mathbf{b}_i = \mathbf{x}_0 + \sum_{i=1}^n f_i \mathbf{b}_i \quad (\text{A5})$$

which is Eq. (23) and contains the coefficient f_i as stated in Eq. (24).

Appendix B: Spinors in Matrix Form

Here, we explicitly derive in GA the rotation action of a spinor operator \mathbf{R} on the unit vector σ_1 . The spinor operator \mathbf{R} consists of the base spinor $R = \alpha + i\mathbf{b}$ and its reverse spinor R^\dagger . We relate the 4-tuple (α, b_x, b_y, b_z) with the spinor R . The pseudoscalar i is equal to $\sigma_1 \sigma_2 \sigma_3$. The action of \mathbf{R} on σ_1 is now:

$$\begin{aligned} \sigma_1' &= R^\dagger \sigma_1 R \\ &= R^\dagger \sigma_1 (\alpha + \sigma_1 \sigma_2 \sigma_3 (b_x \sigma_1 + b_y \sigma_2 + b_z \sigma_3)) \\ &= R^\dagger \sigma_1 (\alpha + \sigma_2 \sigma_3 b_x - b_y \sigma_1 \sigma_3 + b_z \sigma_1 \sigma_2) \\ &= R^\dagger (\alpha \sigma_1 + \sigma_1 \sigma_2 \sigma_3 b_x - b_y \sigma_3 + b_z \sigma_2) \\ &= (\alpha - \sigma_2 \sigma_3 b_x + b_y \sigma_1 \sigma_3 - b_z \sigma_1 \sigma_2) \\ &\quad \cdot (\alpha \sigma_1 + \sigma_1 \sigma_2 \sigma_3 b_x - b_y \sigma_3 + b_z \sigma_2) \end{aligned} \quad (\text{B1})$$

in which the basic properties of the geometric products $\sigma_i \sigma_j$ have been used, namely $\sigma_i \sigma_i = 1$ and $\sigma_i \sigma_j = -\sigma_j \sigma_i$. Further elaboration of the last 2 lines in Eq. (B1) yields:

$$\begin{aligned} \sigma_1' &= (\alpha^2 + b_x^2 - b_y^2 - b_z^2) \sigma_1 + 2(\alpha b_z + b_x b_y) \sigma_2 \\ &\quad + 2(b_x b_z - \alpha b_y) \sigma_3 + 0i \end{aligned} \quad (\text{B2})$$

in which we collected the terms for the components σ_i . The 4-tuple (α, b_x, b_y, b_z) is now substituted by (s_0, s_1, s_2, s_3) to illustrate better the analogy to the quaternion (q_0, q_1, q_2, q_3) . Equation (B2), using $s_0^2 + s_1^2 + s_2^2 + s_3^2 = 1$, can now be rewritten as:

$$\begin{aligned} \sigma_1' &= 2(s_0^2 + s_1^2 - 1/2) \sigma_1 + 2(s_1 s_2 + s_0 s_3) \sigma_2 \\ &\quad + 2(s_1 s_3 - s_0 s_2) \sigma_3 + 0i \end{aligned} \quad (\text{B3})$$


which recovers the first column of the quaternion matrix in Eq. (35). Analogously, the coefficients for σ_2 and σ_3 can be calculated to recover the second and third columns of this matrix.

Acknowledgments

The authors thank Santiago García Sánchez for his advice on the making of the drawings. They also wish to thank the reviewers for the critical assessment of the submitted manuscript and their constructive advice on improvements.

Keywords: protein conformation • rotation • internal coordinate • modeling • geometric algebra • spinor • quaternion

How to cite this article: P. Chys, P. Chacón, *J. Comput. Chem.* **2012**, *00*, 000–000. DOI: 10.1002/jcc.23002

 Additional Supporting Information may be found in the online version of this article.

- [1] R. Abagyan, M. Totrov, D. Kuznetsov, *J. Comp. Chem.* **1994**, *15*, 488.
- [2] A. Brünger, J. Kuriyan, M. Karplus, *Science* **1987**, *235*, 458.
- [3] A. Brünger, P. Adams, G. Clore, P. Gros, R. Grosse-Kunstleve, J.-S. Jiang, J. Kuszewski, N. Nilges, N. Pannu, R. Read, L. Rice, T. Simonson, G. Warren, *Acta Cryst.* **1998**, *D54*, 905.
- [4] A. Brünger, *Nat. Protocols* **2007**, *2*, 2728.
- [5] J. Ponder, F. Richards, *J. Comp. Chem.* **1987**, *8*, 1016.
- [6] S.-H. Lee, K. Palmo, S. Krimm, *J. Comp. Chem.* **2007**, *28*, 1107.
- [7] P. Pulay, B. Paizs, *Chem. Phys. Lett.* **2002**, *353*, 400.
- [8] M. Totrov, R. Abagyan, *J. Comp. Chem.* **1994**, *15*, 1105.
- [9] K. Németh, M. Challacombe, M. Van Veenendaal, *J. Comp. Chem.* **2010**, *31*, 2078.
- [10] C. Alvarado, K. Kazeronian, *Protein Eng.* **2003**, *16*, 717.
- [11] C. Seok, E. Coutsiar, *Bull. Korean Chem. Soc.* **2007**, *28*, 1705.
- [12] V. Choi, *J. Chem. Inf. Model* **2005**, *46*, 438.
- [13] J. Parsons, J. B. Holmes, J. M. Rojas, J. Tsai, C. E. M. Strauss, *J. Comp. Chem.* **2005**, *26*, 1063.
- [14] J. Pesonen, O. E. Henriksson, *J. Comp. Chem.* **2009**, *31*, 1874.
- [15] P. Chys, *J. Chem. Phys.* **2008**, *128*, 104107–1.
- [16] D. Hestenes, *New Foundations for Classical Mechanics*; Reidel: Dordrecht, **1986**.
- [17] A. Lasenby, C. Doran, *Geometric Algebra for Physicists*; Cambridge University Press: Cambridge, **2002**.
- [18] L. Dorst, D. Fontijne, S. Mann, *Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry*; Morgan Kaufmann: San Francisco, **2007**.
- [19] D. Hestenes, D. Sobczyk, *Clifford Algebra to Geometric Calculus*; Reidel: Dordrecht, **1985**.
- [20] J. Gibbs, *The Scientific Papers: 2. Dynamics, Vector Analysis and Multiple Algebra*; Electromagnetic Theory of Light; Dover: New York, **1961**.
- [21] A. Fersht, *Enzyme Structure and Mechanism*; W. H. Freeman and Company: New York, **1985**.

- [22] J. Rooney, *Open Mech. Eng. J.* **2010**, *4*, 86.
- [23] D. Marsh, *Applied Geometry for Computer Graphics and CAD*; Springer-Verlag: London, **2005**.
- [24] E. Bayro-Corrochano, J. Zamora-Esquivel, *Robotica* **2007**, *25*, 43.
- [25] J. Chou, *IEEE Trans. Robot. Automat.* **1992**, *8*, 53.
- [26] E. Coutasias, C. Seok, K. Dill, *J. Comp. Chem.* **2004**, *25*, 1849.
- [27] J. Zamora-Esquivel, B. Bayro-Corrochano, In *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010—Conference Proceedings*; Taipei, **2010**; p. 2377.
- [28] E. Bayro-Corrochano, O. Carbajal-Espinosa, A. Loukianov, In *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*; Taipei, **2010**; p. 1378.

Received: 30 September 2011

Revised: 26 January 2012

Accepted: 1 April 2012

Published online on