

Practical Conversion from Torsion Space to Cartesian Space for *In Silico* Protein Synthesis

JEROD PARSONS,¹ J. BRADLEY HOLMES,¹ J. MAURICE ROJAS,¹ JERRY TSAI,¹
CHARLIE E. M. STRAUSS²

¹Texas Agricultural Experiment Station, Texas A&M University, College Station, Texas 77843

²Los Alamos National Laboratory, Los Alamos, New Mexico 87545

Received 17 January 2005; Accepted 4 March 2005

DOI 10.1002/jcc.20237

Published online in Wiley InterScience (www.interscience.wiley.com).

Abstract: Many applications require a method for translating a large list of bond angles and bond lengths to precise atomic Cartesian coordinates. This simple but computationally consuming task occurs ubiquitously in modeling proteins, DNA, and other polymers as well as in many other fields such as robotics. To find an optimal method, algorithms can be compared by a number of operations, speed, intrinsic numerical stability, and parallelization. We discuss five established methods for growing a protein backbone by serial chain extension from bond angles and bond lengths. We introduce the Natural Extension Reference Frame (NeRF) method developed for Rosetta's chain extension subroutine, as well as an improved implementation. In comparison to traditional two-step rotations, vector algebra, or Quaternion product algorithms, the NeRF algorithm is superior for this application: it requires 47% fewer floating point operations, demonstrates the best intrinsic numerical stability, and offers prospects for parallel processor acceleration. The NeRF formalism factors the mathematical operations of chain extension into two independent terms with orthogonal subsets of the dependent variables; the apparent irreducibility of these factors hint that the minimal operation set may have been identified. Benchmarks are made on Intel Pentium and Motorola PowerPC CPUs.

© 2005 Wiley Periodicals, Inc. J Comput Chem 26: 1063–1068, 2005

Key words: conversion; torsion space; Cartesian space; protein synthesis

Introduction

The backbones of common biological polymers such as proteins and nucleic acids consist of consecutively linked (bonded) series of atoms occupying a three-dimensional structure (Fig. 1). Mathematically, this structure can be represented by the Cartesian coordinates of the atoms or alternatively, as bond lengths and angles. The former is the natural basis describing physical force fields and dielectric densities while the latter is more natural to chemical description of covalent and hydrogen bonding as well as certain kinds of experimental information.

Ab initio protein structure modeling algorithms search through many atomic conformations on a potential surface. Some methods, such as Rosetta^{1,2} and Dyana,³ generate perturbations or trial moves in angle space and then convert these to Cartesian atom positions where radially dependent potential functions or force fields can be evaluated. This conversion, dubbed "refolding," can dominate the time it takes to evaluate a trial move; Rosetta typically spends one-quarter to half of its CPU time refolding. Thus, efficient conversion of bond-centric torsion angles to Cartesian representations are of vital interest. Numerical accuracy is also

essential because a typical protein chain can loop back on itself, bringing two atoms separated by thousands of intervening swiveling bonds into the range of van der Waals forces, which vary sharply over hundredths of an angstrom.

A straightforward procedure to convert from torsion angles to Cartesian coordinates is to work from one end of the chain to the other, sequentially placing one bonded atom at a time in relation to the previously placed ones. All that is necessary to place a particular atom is the Cartesian coordinates of the three previous atoms

Correspondence to: C. Strauss; e-mail: cems@lanl.gov

Contract/grant sponsor: NSF-UBM; contract/grant number: DSM-0211458 (to J.R.P. and J.B.H.)

Contract/grant sponsor: Barry M. Goldwater Foundation (to J.B.H.)

Contract/grant sponsor: Los Alamos Directed Research and Department of Energy's Genomes-GTL Program (to C.E.M.S.)

Contract/grant sponsor: NSF; contract/grant numbers: DMS-0138446 and DMS-0211458 (to J.M.R.)

Contract/grant sponsor: Welch Foundation; contract/grant number: A-1549 (to J.T.)

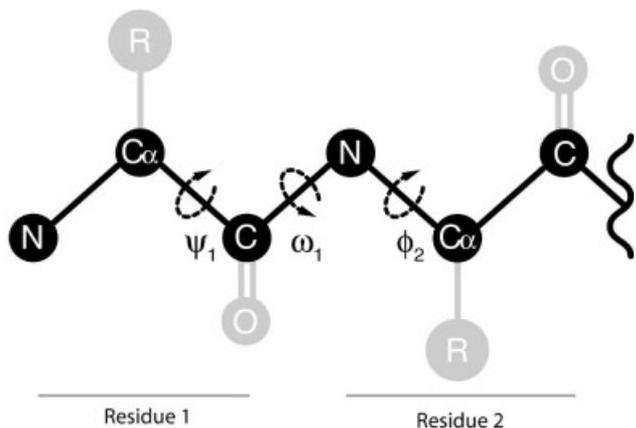


Figure 1. Schematic of the peptide backbone. A peptide's nitrogen (N), alpha carbon (Ca), carbon (C) are shown in black, while its oxygen (O) and side chain (R) atoms are in grey. Three backbone torsion angles (psi, phi, omega) are shown. Each peptide subunit has three bond angles, three bond lengths, and three torsion angles. This peptide-centric picture is irrelevant to the algorithms: all methods merely place the next consecutive atom with each iteration using just the bond angle with the previous two atoms, and the torsion angle about the bond between the previous two atoms.

in the chain and three bond parameters: bond length of the new bond, bond angle relative to the previous bond, and torsion angle about the previous bond.

Intuitively, and in common practice, this is done by first placing the new atom a bond length away from the previous atom in line with the previous bond axis (see Fig. 2). Then this new bond is rotated so that the position agrees with the desired bond and torsion angles. The rotation is done in two steps, first setting the angle between the two bonds, then swiveling about the previous bond to the correct torsion angle. Similar to a previous study,⁴ this work evaluates three such two step algorithms—General Rotations, Quaternions,⁵ and the Rodrigues-Gibbs formula.^{6,7} Another study,⁸ based on the Denavit-Hartenberg method, makes a number of assumptions such that all bond angles and planar torsion angles are constant. Our methods remain more generally applicable to sequential rotations about any axis, not just the customary two torsion angles.

A perhaps less intuitive but algorithmically simpler strategy for conversion of torsion angles to Cartesian coordinates is to first place the new atom in a default coordinate system where the position can be easily computed from just the angles alone, and then rotate this coordinate system in a single step into the reference frame of orientation defined by the three previous atoms. We call this the Natural Extension Reference Frame (NeRF) algorithm. This approach requires just over half as many operations as the other two step methods mentioned above (Table 1). In addition to a much smaller operation count, we find that this method also has greater intrinsic numerical stability and improved prospects for parallelization.

This work provides a critical analysis of the NeRF algorithm. The method itself is well proven because it is the refolding engine written for the Rosetta structure prediction algorithm. It is literally

applied trillions of times per year and is the most frequently called subroutine in our labs.^{1,9,10} In this study we show that this scheme as implemented in Rosetta is, in fact, excessively cautious numerically. Here we introduce an even faster variant, the Self-Normalizing Natural Extension Reference Frame (SN-NeRF) by removing unneeded renormalization steps that account for 30% of the operations.

As part of our analysis, we make a practical comparison of NeRF and SN-NeRF approaches to the three aforementioned two-step methods for converting rapidly from the bond-centric basis to the Cartesian representation. Analogous situations occur in the placement of multijointed robot arms and in spherical coordinate random-walk problems, which are the consecutive summation of a series of vectors described in spherical-polar coordinates. Evaluations are made between the five algorithms by totaling their floating point operation counts, benchmarking their speed, and indicating prospects for parallelization.

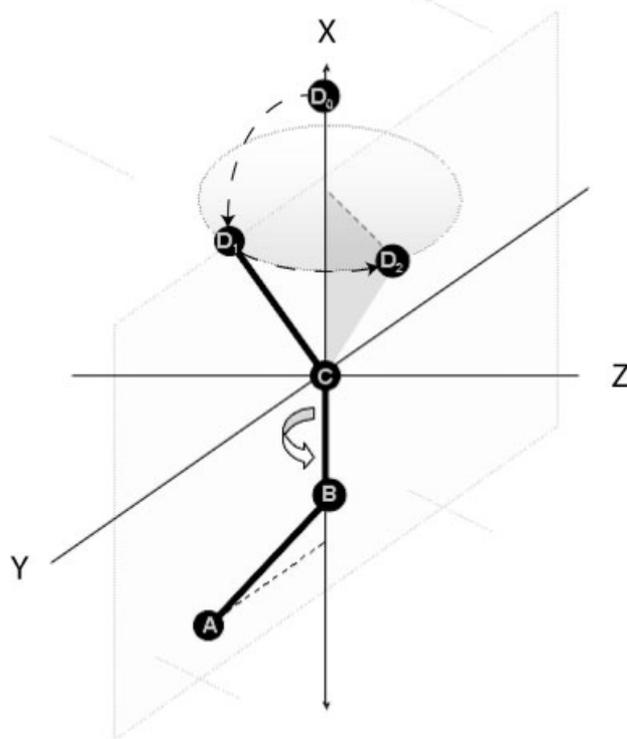


Figure 2. The spherical to Cartesian coordinate transformation. In the two-step procedures the next atom is first placed at a point D_0 , which is an extension along the BC bond axis by the new bond length. This point is first rotated about the point C in the ABC plane by the bond angle to the position D_1 . Then it is rotated about the BC bond axis by the torsion angle to D_2 . In the NeRF methods, the atom is placed directly into an XYZ coordinate system at the point D_2 . This coordinate system is then transformed so that the XY plane lies in the ABC plane. This transformation is applied to D_2 .

Table 1. Benchmarks and Floating Point Operation Counts.

| | Two-step vector methods | | | Natural reference methods | |
|----------------------|-------------------------|-----------------|------------|---------------------------|-----------|
| | General rotation | Rodriguez-Gibbs | Quaternion | NeRF | SN-NeRF |
| Intel B/S (relative) | 723 (0.78) | 797 (0.86) | 732 (0.79) | 728 (0.78) | 930 (1.0) |
| PPC B/S (relative) | 325 (0.78) | 348 (0.83) | 411 (0.98) | 300 (0.72) | 418 (1.0) |
| Square roots | 2 | 2 | 1 | 3 | 1 |
| Sin/Cos | 4 | 4 | 4 | 4 | 4 |
| Division | 9 | 9 | 9 | 9 | 6 |
| Multiplication | 59 | 59 | 64 | 37 | 31 |
| Add/Sub | 55 | 50 | 47 | 34 | 24 |
| Total | 129 | 124 | 125 | 87 | 66 |

Left of the table shows the number of protein backbones built per second on two different computing platforms (Intel and PPC). The ratio of these rates to the fastest algorithm (SN-NeRF) is given in parentheses. Right shows the number of floating point operations, by kind, for one iteration of the atom placement algorithm. Notably, SN-NeRF has the lowest operation-count in every category.

Methods

As shown in Figure 1, the repeating “peptide” unit of a protein backbone consists of three atoms (nitrogen, alpha carbon, and carbonyl carbon) whose positions can be described by three bond lengths, three bond angles, and three torsion angles.¹¹ By convention, when four consecutive atoms lie in a plane, the torsion angle of the middle bond is 0°.

Because all of the methods we describe are exact and nominally produce the same result for a given input, they are simply different factorings of a single underlying mathematical transformation. Because of differing variable representation, the methods differ in intrinsic numerical stability, which permits some methods to achieve the same accuracy using fewer operations. We can therefore observe differences in their efficiency and speed. For pedagogical purposes we can divide the algorithms into two groups: two-step vector rotation and the one-step NeRF formulation. We compare three implementations of the two-step method and two implementations of the NeRF method.

A single iteration of any of the algorithms is the placement of the next atom D following three previous atoms A , B , and C in the chain (Fig. 2). The inputs for each iteration are the Cartesian positions of the sequentially bonded atoms (A , B , and C), the bond length from atom C to D (bond_{CD}), the bond angle with vertex C (angle_{BCD}), and the torsion angle around the bond from B to C (torsion_{BC}). Additionally, two of the methods exploit the chained nature of the buildup by accepting as input the bond length from B to C (bond_{BC}), which is supplied in the previous atom placement iteration. The output of all methods is the Cartesian position of the fourth sequentially bonded atom in the chain, D . For convenience, we define the vectors along bonds as $\vec{AB} \equiv \vec{B} - \vec{A}$ and $\vec{BC} \equiv \vec{C} - \vec{B}$ and the corresponding unit vector as $\hat{b}c \equiv \frac{\vec{BC}}{|\vec{BC}|}$.

Two-Step Vector Rotations

To distinguish the three two-step algorithms, we refer to them as General-Rotation, Rodrigues-Gibbs, and Quaternion. In these

methods, the atom D is initially placed at \vec{D}_0 , a distance bond_{CD} away from atom C extending along the (\vec{BC}) bond axis (see Fig. 2). This atom is then rotated around C in the ABC plane (i.e., about normal \hat{n}) by the bond angle angle_{BCD} where:

$$\hat{n} \equiv \frac{\vec{AB} \times \hat{b}c}{|\vec{AB} \times \hat{b}c|} \quad (1)$$

The atom is then rotated a second time around the \vec{BC} by torsion $_{BC}$.

Because both rotations are about atom C , it's efficient to treat it as the origin, so the initial placement of the D atom is $\vec{D}_0 = (\text{bond}_{CD}) * \hat{b}c$. After the rotations this is then offset by \vec{C} to obtain the final \vec{D} .

General Rotation (Gen. Rot.)

The General Rotation method^{12,13} calculates a rotation matrix for a vector around an arbitrary rotation axis unit vector \vec{u}_i by a particular magnitude θ . The general formula is

$$\hat{M}(\vec{u}, \theta) = \hat{I} \cos \theta + \vec{u}\vec{u}^T(1 - \cos \theta) + \vec{u} \sin \theta \quad (2)$$

where \vec{u} is the antisymmetric or skew-symmetric matrix of \vec{u}_i , and \hat{I} is the identity matrix. In matrix form this equation reads

$$\begin{array}{ccc} \vec{u}_x^2(1 - c\theta) + c\theta & \vec{u}_x\vec{u}_y(1 - c\theta) - \vec{u}_z s\theta & \vec{u}_x\vec{u}_z(1 - c\theta) + \vec{u}_y s\theta \\ \vec{u}_x\vec{u}_y(1 - c\theta) + \vec{u}_z s\theta & \vec{u}_y^2(1 - c\theta) + c\theta & \vec{u}_y\vec{u}_z(1 - c\theta) - \vec{u}_x s\theta \\ \vec{u}_x\vec{u}_z(1 - c\theta) - \vec{u}_y s\theta & \vec{u}_y\vec{u}_z(1 - c\theta) + \vec{u}_x s\theta & \vec{u}_z^2(1 - c\theta) + c\theta \end{array}$$

where $s \equiv \sin$ and $c \equiv \cos$.

Two such matrices are sequentially applied to \vec{D}_0 , first for angle $_{BCD}$ with $\vec{u} = \hat{n}$ and then for torsion $_{BC}$ with $\vec{u} = \hat{b}c$. It is faster to apply the separate rotation matrices to \vec{D}_0 sequentially rather than combining the rotations into one transformation before applying it.

Rodrigues-Gibbs Formulation (RG)

This vector algebra transformation^{6,7} does not use rotation matrices. The formula for rotating any vector \vec{K} about axis \hat{u} by angle θ is

$$\vec{K}' = \vec{K} \cos \theta + \hat{u} \times \vec{K} \sin \theta + (\vec{K} \cdot \hat{u})\hat{u}(1 - \cos \theta)$$

As before, this must be done twice per atom: once to apply the bond angle, and again to apply the torsion angle to \vec{D}_0 .

Quaternion Rotation

Quaternion Rotations⁵ compactly represent the transformation of a rotation around an arbitrary axis \hat{u} by a particular magnitude θ as a four element vector $Q(\theta, \hat{u}) = (s, \vec{v}) = (s, x, y, z)$, where

$$s = \cos \frac{\theta}{2}, \quad \vec{v} = (x, y, z) = \sin \frac{\theta}{2} \hat{u}$$

To apply a Quaternion transformation one computes the following rotation matrix and applies this to the vector.

$$\begin{array}{ccc} s^2 + x^2 + y^2 + z^2 & 2xy + 2sz & 2xz - 2sy \\ 2xy - 2sz & s^2 - x^2 + y^2 - z^2 & 2yz + 2sx \\ 2xz + 2sy & 2yz - 2sx & s^2 - x^2 - y^2 + z^2 \end{array}$$

Unlike rotation matrices, when applying multiple transformations it is faster to combine two Quaternions and apply the net rotation matrix to the vector than it is to iteratively compute the two implied rotation matrices and apply them sequentially. To combine Quaternions

$$Q_1 * Q_2 = (s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2, s_1 \vec{v}_2 + s_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2) \quad (3)$$

This new Quaternion is applied to \vec{D}_0 . For completeness, we note that there is an alternative vector algebra Quaternion rotation method but it is not as efficient as the matrix method.³

Natural Extension Reference Frame

Rather than initially placing D along an extension of the BC bond axis, we compute the position of atom D in a special coordinate reference frame. Then, we transform this frame to the proper orientation that lines up with the existing A , B , and C coordinates. The problem now factors elegantly, because the first placement operation depends only on the angles and bond length, and the transformation only depends upon the A , B , and C atom positions.

Natural Extension Reference Frame (NeRF)

This is the method used by the Rosetta structure modeling algorithm.² We use a frame of reference where atom C is at the origin, atom B on the negative x -axis, and atom A lies in the xy -plane. Then, the initial position of the D atom is given by the usual spherical coordinate representation

$$\vec{D}_2 = (R \cos(\theta), \quad R \cos(\phi)\sin(\theta), \quad R \sin(\phi)\sin(\theta))$$

where

$$R = \text{bond}_{CD}, \quad \theta = \text{angle}_{BCD}, \quad \phi = \text{torsion}_{BC}$$

This initial placement is independent of the positions of A , B , and C and depends solely on the bond angles and length. To obtain the final position of D we transform this frame to the protein chain's frame of reference by \hat{M} . That is, $\vec{D} = \hat{M}\vec{D}_2 + \vec{C}$, where the component vectors form the columns of \hat{M} :

$$\hat{M} \equiv [\hat{bc}, \quad \hat{n} \times \hat{bc}, \quad \hat{n}] \quad (4)$$

This final transformation depends solely upon the locations of A , B , and C and is independent of the bond parameters.

Self-Normalizing Natural Extension Reference Frame (SN-NeRF)

In the NeRF implementation in Rosetta, the first two columns of the matrix \hat{M} are explicitly normalized as follows:

$$\hat{M}_x = \hat{bc} = \frac{\overrightarrow{BC}}{|\overrightarrow{BC}|}, \quad \hat{M}_y = \frac{\hat{n} \times \hat{bc}}{|\hat{n} \times \hat{bc}|} \quad (5)$$

Both of these operations are quite stringent and can be relaxed. The calculation of $|\overrightarrow{BC}|$ can simply be replaced with bond_{BC} , the bond length supplied in the previous iteration. Also, by construction, the magnitude of the crossproduct $\hat{n} \times \hat{bc}$ is 1; thus, it does not need any normalization. The original algorithm was overcautious, and included both of these normalizations to avoid possible rotation matrix skew or nonunitarity created by cumulative errors (see Discussion).

Enumeration of Operations

Each of the aforementioned methods for building a protein backbone was concretely implemented in a computer program where the number of mathematical operations were counted. Table 1 tallies additions/subtractions, multiplications, divisions, square roots, and sine/cosine calls for each, separately. We note that in general the relative time expense of these operations varies by processor architecture. A previous study of only the three two-step vector methods⁴ did not differentiate between the high cost of trigonometric and square root operations and the relatively low cost of additions/subtractions in their operation count, nor did it provide accurate calculations of the operations necessary for each implementation. Our practical implementation provides a precise operation count based on the realistic requirements for a folding algorithm.

Speed Test

Tests were run on a Linux based 800 MHz Intel® Pentium III processor and an Apple Macintosh G4 (667 MHz). The build-rate

of each algorithm was evaluated by constructing the minimal backbone (N, C α , C) of an arbitrarily chosen protein 1fug (chain A of S-Adenosylmethionine Synthetase)¹⁴ from torsion parameters and bond lengths. The program constructed this 383 residue (1149 atom) backbone 10,000 times per trial. Rates reported are based on elapsed real time, not processor time, using timers in the code itself to remove any overhead of the test harness from the folding algorithm. Any unaccounted overhead was sufficiently amortized to be undetectable: plotting the number of builds per trial from 5000 to 40,000 against the elapsed time produced a line intercepting the origin with an R^2 of 0.99 (data not shown). The initial three atoms of the first residue in the chain were prepositioned; thus, the first atom to be built was the second amino acid's nitrogen (Fig. 1). The number of structures built per second is also reported in Table 1, and the number of atoms placed per second is 1146 times this value.

Results

The "Total" column of Table 1 shows the number of floating point operations necessary to add a single atom. The SN-NeRF method has roughly half as many operations (66) as the two-step vector methods (over 123). As shown in Table 1, the SN-NeRF method builds 930 ± 3 model proteins per second. The next fastest method is the RG formulation, which builds 12.3% fewer models. The other three algorithms all build approximately 22% under intel architecture. All methods can rebuild the protein from the bond lengths, bond angles, and torsion angles computed from the original PDB file. These methods return the rebuilt PDB coordinates with 0.000016 \AA Root-Mean-Square-Deviation from the original crystal coordinates. The NeRF algorithm runs considerably slower than expected based solely on a judgment of the number of operations. Even removing the compiler optimization flags still shows a disproportionate relationship between the operation count and the build-rate, suggesting that hardware pipelining of floating point instructions is playing a role, which is difficult to analyze.

Discussion

We show the SN-NeRF to be the fastest performing method for refolding atoms. This superiority is due to the difference in computational complexity, as discussed above and shown in Table 1. The removal of two square roots, six multiplications, three divisions, and 10 additions in the SN-NeRF method results in a 22% decrease in the time required to build models over the original NeRF algorithm. This decrease in number of computations is due to the fact that the SN-NeRF method is self-normalized. However, this normalization cannot be applied to all methods.

Accumulated rounding errors plague attempts to self-normalize the Gen. Rot. and RG methods. The results of these errors is the failure to generate an orthonormal matrix. The Quaternion method goes to extra lengths to make its matrix orthonormal, while the Gen. Rot. and RG methods do not. Both the Quaternion and SN-NeRF methods (by virtue of being orthonormal) are able to self-normalize with bond length_{AB} and bond length_{BC} that are stored in memory. The SN-NeRF method generates its three or-

thonormal vectors prior to self-normalizing, thus eliminating any possibility for such errors.

We attribute the origin of the discrepancy in the operation count vs. the benchmark speed to efficiencies gained from hardware pipelining of instructions. In the methods with higher operation counts, more work than necessary is done. However, in these methods the computation of the rotation matrices tends to involve combinations of previously used variables, rather than the results of a dependent series of calculations. In the SN-NeRF method, as written, the extraneous operations are fewer but a greater number of lines depend on the results from a previous line. Thus, the higher count algorithms may not be paying the full price for their excess because the independent calculations pipeline more efficiently in the CPU.

Building a long chain by serial extension is intolerant of round-off errors, that introduce cumulative skew and stretch into the net rotation matrix. When creating a 3×3 rotation matrix by independently filling in all of the elements, one has not explicitly addressed this issue. In the NeRF formalism, the \hat{M} matrix is constructed in a way to assure orthogonal and unit-vector columns. For example, \hat{M}_y is computed from the crossproduct of the other two rows (assuring orthogonality), both of which are orthogonal unit vectors themselves. In the NeRF or nonself-normalizing version, this is explicitly normed as well. One could, of course, add these explicit orthonormalizations to the construction of the Gen. Rot. or RG matrices, but this would further inflate their operation counts. Because the NeRF methods are orthonormal by construction, their intrinsic numerical stability allows them to pay no price for achieving the same accuracy with a lower operation count. Each time one combines multiple terms, particularly in subtraction, one is losing precision when small differences of large numbers occur. Thus, the matrix methods with their noncompact representation invite this effect. Conversely, by making the \hat{M}_y dependent on the other columns we avoid some of this redundancy.

Parallelization and Suggested Improvements

Despite, having just indicated that our as-written implementation may not be pipelining well, this is not an intrinsic limitation. In fact, we believe the NeRF formalism actually lends itself better than the other methods to the various types of parallelization: multiprocessor, pipeline, or vector. First, because the calculation factors out the terms that depend solely on the bond parameters, the initial atom placements \vec{D}_2 can be done independently for all the residues on as many processors as are available. In other algorithms, you cannot compute the next initial position \vec{D}_0 until the last atom placement is complete, making it less friendly to parallelization. The best one could do would be to precompute all the trigonometric evaluations. Next, most of the NeRF method, even the filling out of the rotation matrix, is conveniently written in terms of vector operations like crossproducts and norms. This strongly suggests explicitly rewriting the algorithm to take advantage of vector processing opportunities, such as SIMD-type operations found on many modern math processors (e.g., MMX, SSE2, and AltiVec). We did not do so because this would involve non-ANSI C++ compiler and language specific functions calls or externally optimized libraries.

Thus, we believe the latent speed implicit in the lower operation count for the NeRF methods could be recovered by rewriting the chain extension protocol to first compute all the initial placements. Their independence would pipeline well. Then, the NeRF methods could explicitly exploit the vector parallelization of the processors. Moreover, in the case of multiple parallel processors, the factorization is better suited to this role more than the other methods. In all these ways, the NeRF formalism more easily exploits parallelization.

Each of the methods share the same number of trigonometric operations, and thus all the methods could benefit from reducing the cost of these calls. For example, many compilers support extensions that compute the sine and cosine simultaneously. Moreover, in applications like Rosetta, often the angles are discrete and known ahead of time, permitting the calculations to be replaced by precomputed look-up tables. If these time-dominant operations were greatly reduced, the lower operation count of the SN-NeRF method would become even more distinct in our benchmark of speed.

Perhaps the largest latent gain lies in exploiting the superior intrinsic stability of the SN-NeRF formalism by switching to single precision from the double precision used in this work. Not only will all the calculations be more swift but vector processing instructions can pack twice as many floats into their vector size. Some of these improvements may be available to compilers with better optimizing routines without rewriting the program.

For our applications, in conjunction with Rosetta, we could not use nonstandard C++ functions nor use lower precision variables. We assume this is true of the major uses of chain-building algorithms. Thus, the scope of this study was kept to ANSI-standard code, and we did not further explore these options.

Conclusion

After a rigorous study, we conclude that the SN-NeRF buildup method is best for calculating Cartesian coordinates from torsion space parameters. Both the RG formulation and the Gen. Rot. algorithm are susceptible to round-off error. We believe that the most practical and effective approach to calculating Cartesian

coordinates is to step through the protein systematically. Atomic positions are determined as you walk down the protein backbone and then used in subsequent calculations. We would also like to point out that although we used only a minimal protein backbone for testing in this study, these general methods are applicable to protein side chains, or any atoms described by torsion parameters, without modification. Programs and source code are available by contacting the corresponding author.

Acknowledgments

We thank David Baker for helpful discussions and support. We would also like to thank David Schell for thoughtful input and advice.

References

1. Simons, K. T.; Kooperberg, C.; Huang, E.; Baker, D. *J Mol Biol* 1997, 268, 209.
2. Rohl, C. A.; Strauss, C. E.; Misura, K. M.; Baker, D. *Methods Enzymol* 2004, 383, 66.
3. Guntert, P.; Mumenthaler, C.; Wuthrich, K. *J Mol Biol* 1997, 273, 283.
4. Alvarado, C.; Kazerounian, K. *Protein Eng* 2003, 16, 717.
5. Hamilton, W. R. *Lectures on Quaternions*; Hodges and Smith: Dublin, 1853.
6. Bottema, O.; Roth, B. *Theoret Kinemat* 1979, 56–62; 518.
7. Goldstein, H. *Classical Mechanics*; Addison-Wesley: Reading, MA, 1980.
8. Kavraki, L.; Zhang, M. *J Chem Inf Comput Sci* 2002, 42, 64.
9. Chivian, D.; Kim, D. E.; Malmstrom, L.; Bradley, P.; Robertson, T.; Murphy, P.; Strauss, C. E. M.; Bonneau, R.; Rohl, C. A.; Baker, D. *Proteins* 2003, 53(Suppl 6), 524.
10. Kim, D. E.; Chivian, D.; Baker, D. *Nucleic Acids Res* 2004, 32(Suppl 2), W526.
11. Holmes, J. B.; Tsai, J. W. *Protein Sci* 2004, 13, 1636.
12. Suh, C. H.; Radcliffe, C. W. *Kinematics and Mechanisms Design*; Wiley: New York, 1978, p. 45.
13. Mortari, D. *J Astronaut Sci* 2001, 49, 401.
14. Fu, Z.; Hu, Y.; Markham, G. D.; Takusagawa, F. *J Biomol Struct Dyn* 1996, 13, 727.