Procedia Computer Science

Volume 80, 2016, Pages 269–278

ICCS 2016. The International Conference on Computational Science

Acceleration and Parallelization of ZENO/Walk-on-Spheres

Derek Juba[1], Walid Keyrouz[1], Michael Mascagni[1,2], and Mary Brady[1]

[1] Information Technology Laboratory
National Institute of Standards and Technology, Gaithersburg, MD 20899
{derek.juba,walid.keyrouz,michael.mascagni,mary.brady}@nist.gov
[2] Departments of Computer Science, Mathematics, and Scientific Computing
Florida State University, Tallahassee, FL 32306
mascagni@fsu.edu

Abstract

This paper describes our on-going work to accelerate ZENO, a software tool based on Monte Carlo methods (MCMs), used for computing material properties at nanoscale. ZENO employs three main algorithms: (1) Walk on Spheres (WoS), (2) interior sampling, and (3) surface sampling. We have accelerated the first two algorithms. For the sake of brevity, the paper will discuss our work on the first one only as it is the most commonly used and the acceleration techniques were similar in both cases.

WoS is a Brownian motion MCM for solving a class of partial differential equations (PDEs). It provides a stochastic solution to a PDE by estimating the probability that a random walk, which started at infinity, will hit the surface of the material under consideration. WoS is highly effective when the problem's geometry is additive, as this greatly reduces the number of walk steps needed to achieve accurate results. The walks start on the surface of an enclosing sphere and can make much larger jumps than in a direct simulation of Brownian motion. Our current implementation represents the molecular structure of nanomaterials as a union of possibly overlapping spheres. The core processing bottleneck in WoS is a Computational Geometry one, as the algorithm repeatedly determines the distance from query point to the material surface in each step of the random walk.

In this paper, we present results from benchmarking spatial data structures, including several open-source implementations of k-D trees, for accelerating WoS algorithmically. The paper also presents results from our multicore and cluster parallel implementation to show that it exhibits linear strong scaling with the number of cores and compute nodes; this implementation delivers up to 4 orders of magnitude speedup compared to the original FORTRAN code when run on 8 nodes (each with dual 6-core Intel Xeon CPUs) with 24 threads per node.

Keywords: Monte Carlo Methods, nanomaterial properties, Walk on Spheres, parallelization, k-D tree

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.319

1 Introduction

This work describes the optimization and parallelization of ZENO, a tool, originally developed with National Institute of Standards and Technology (NIST) support, for computing a class of material properties of arbitrary geometric shapes at nanoscale via Monte Carlo methods (MCMs) [7, 10–12, 15, 16, 22, 23]. ZENO is used to compute a variety of properties, is integrated into a data analysis package for hydrodynamic data, UltraScan-SOMO [5], and is cited as one of the reference tools for computing these properties [24–26]. Computing the capacitance using MCMs, as ZENO does, is an essential tool in many areas of technology, such as VLSI design [20, 21]. As some of these applications can run for hours or even days, accelerating the computation would be of great benefit.

The ZENO program is named for Zeno's Achilles Paradox, a reference to the main algorithm that the program performs, the Walk on Spheres (WoS) algorithm, which is described in Section 3. The program performs three basic operations: (1) ZENO/WoS, (2) Interior Sampling, and (3) Surface Sampling. The user specifies the properties to compute, which may require results from multiple operations; the user also specifies the number of random walks or samples to use. Our colleagues are interested in properties that require the ZENO/WoS and Interior Sampling computations. As such, we have focused our efforts on accelerating and parallelizing these two operations. This paper details the ZENO/WoS computation; the techniques we used to accelerate the Interior Sampling computation were similar.

The ZENO computation uses numerical path integration techniques, which are based on Brownian motion MCMs, to derive stochastic solutions to a family of elliptic PDEs. These stochastic solutions are the desired material properties or can be used to directly compute these properties. The mathematical derivation of the approach first maps these problems onto the electrostatic capacitance problem, which has a stochastic solution, namely the probability of a random walk starting from infinity hitting the surface of the material in question. It then performs a very efficient simulation of the Brownian motion walks by using the WoS algorithm. WoS can make big jumps when compared to Brownian dynamics techniques, which are associated with small jumps.

The paper is organized as follows. Section 2 describes the current ZENO/FORTRAN implementation. It gives the results of our profiling analysis and outlines our strategy for addressing several of the problems already identified in our new C++ implementation. Section 3 describes the WoS algorithm and then considers the computational issues that arise when WoS is applied to additive geometries. Section 4 describes how we accelerated WoS and gives benchmarks for several candidate spatial data structures and algorithms. Section 5 outlines the parallelization of the WoS algorithm. Section 6 then presents numerical results and shows that, with the optimal choice of geometry algorithms, the entire WoS code provides almost perfect parallel speedup, as expected of such Monte Carlo methods. Finally, Section 7 discusses the results, provides conclusions, and suggests future work.

2 ZENO

The "current" version of ZENO is 3.4[1] and is in use at NIST and elsewhere. It is written in FORTRAN 77 and consists of a single source file of nearly 10 500 lines of code. As with many FORTRAN 77 legacy codes, the ZENO code does not adhere to best software practices: it consists of a single source file with some very long routines, `common` blocks, `goto` statements,

[1] Version 3.3 is the last version available from the current ZENO website [12]; version 3.4 is available directly from the principal authors of ZENO.

implicit variables, unnamed constants, uncalled routines, and hard-coded mathematical constants, including 14 instances of the literal π with not all having sufficient precision.

Figures 1 and 2 show examples of molecular structures that NIST researchers are interested in analyzing with ZENO. For 10^6 walks running on an Intel Xeon 2.6 GHz CPU, the Figure 1 structure took 2.29 h to process, while the protein in Figure 2 took 1.03 h. Researchers are interested in using a larger number of walks (e.g., 10^7) or applying the program to materials databases (e.g., the Protein Data Bank [3]). ZENO's running time scales linearly with the number of walks. As such, an improvement in performance will clearly benefit these researchers.

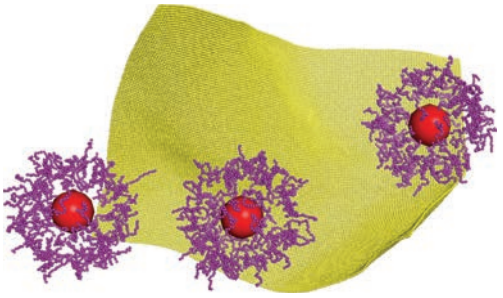

Figure 1: DNA-grafted gold nano-particles interacting with a two-dimensional DNA-based origami construct (23 115 spheres) [14].

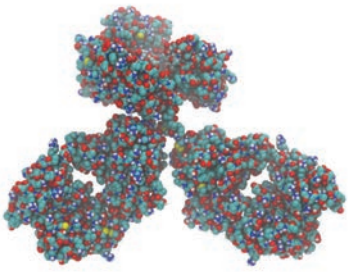

Figure 2: PDB protein 1IGY (12 750 spheres) [9].

To better understand the code, we ported the FORTRAN 77 code to Fortran 2008, reorganized it into multiple files and modules, and restructured the long routines to reveal their internal control structures and the underlying algorithms. We also added an option to specify an input random number seed; this allows us to run the code in deterministic mode and made debugging easier. Our subsequent analysis of the code revealed several issues, with two being the most critical. First, the random number generator saved its state in 31 bits; this is short enough that a feasibly-large input can potentially cause it to exhaust its rather modest period. Second, the nearest neighbor search algorithm used a mostly brute-force (with some caching) full traversal of the input primitives.

Profiling the Fortran 2008 code (Figure 3) on the example in Figure 1 with 10^4 walks revealed that over 98 % of the running time was spent on nearest-neighbor computations (function `distance`) and, more specifically, on a full traversal of the list of primitives (function `distance_scan_all_elts`). There is a mechanism for finding the nearest neighbor from a cached list of 3–4 neighbors. This mechanism (function `distance_scan_neighbors`) was invoked in 279 156 of the 314 268 nearest-neighbor queries; however, it failed for 147 260 of these, resulting in the full traversal still needing to be performed. As a result, the full traversal was performed 182 372 times and dominated the execution time. Since the algorithms scale linearly with the number of walks, these results should remain valid for much larger numbers of walks.

Based on this analysis, we created a plan for a redesigned version of ZENO to be implemented in C++. We selected SPRNG [17] as the random number generator library because it provides features that are useful for parallelization, such as multiple independent random number streams and built-in MPI support. For the nearest neighbor search algorithm, we evaluated a number of different options, as described in Section 4. Finally, we supported parallelism through both shared-memory multi-threading and MPI, as described in Section 5.

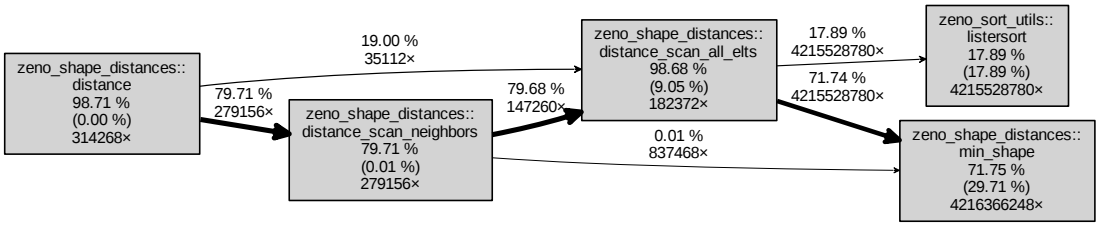

Figure 3: Partial profile of ZENO/Fortran-2008

3 The Walk on Spheres Method

The Walk on Spheres (WoS) algorithm simulates the Brownian motion of a particle (a "random walker") in the vicinity of an object. Brownian motion can be informally defined as taking a sequence of very small steps in random directions. Such a particle will eventually either hit the object or, with a non-zero probability, leave the object's vicinity and never hit it. Mathematically, Brownian motion can be thought of as the solution to a simple stochastic differential equation (SDE) and, as such, the most straightforward method for simulating such a walk is to use a SDE solver with a small time step. WoS avoids performing many small steps by observing that a random walker starting at the center of an empty sphere will reach all points on the surface of the sphere with equal probability. Therefore, the entire random walk that occurred within the sphere can be sampled by simply using a point uniformly distributed on the surface chosen as an exit point sample. The full walk can then be sampled by forming a series of spheres centered at the walker's locations and jumping to the surfaces of these spheres in sequence. The larger the spheres, the more efficient the walk will be, and so we form the largest possible empty sphere at each step.

The only two remaining issues are to define (1) when the walker has hit the object and (2) when it has left the object's vicinity and escaped. A hit is defined as simply the walker moving within some small distance of the object. An escape occurs with a known probability when the walker leaves a bounding sphere of a given radius around the object. If the walker leaves the bounding sphere but does not escape, it is re-positioned on the bounding sphere based on a known, nonuniform probability distribution that is weighted towards the walker's location outside the sphere. This gives rise to Algorithm 1.

Once a random walk algorithm has been defined, the capacitance of an object can be computed by performing a large number of random walks and computing the fraction of walks which hit the object. These walks can be performed either sequentially or in parallel. This computation is given in Algorithm 2.

The WoS Monte Carlo algorithm has many advantages over deterministic algorithms. One of these is that the region whose properties are being measured can be represented via additive geometry, resulting in a very small memory footprint. Furthermore, there is no need to discretize the geometry to improve accuracy (e.g., refining a mesh). As such, the most demanding part in WoS is computing the spheres needed to take the next WoS step. This allows us to conclude that the computational challenge in WoS is due to Computational Geometry. We need to repeatedly answer the following geometric query, where Ω is the region and $\partial\Omega$ its surface.

Given a query point $x \notin \Omega$, find $y \in \partial\Omega$ that minimizes $|x - y|^2$

When the region is represented via additive geometry as the combination (union, intersec-

Algorithm 1 ZENO/WoS

```
function Do Random Walk
    Place walker uniformly on bounding sphere
    loop
        Find nearest point on object
        if walker is within object skin then
            return Hit
        end if
        Form largest empty jump sphere centered at walker
        Place walker uniformly on jump sphere
        if walker is outside bounding sphere then
            if walker escapes then
                return Escape
            else
                Replace walker nonuniformly on bounding sphere
            end if
        end if
    end loop
end function
```

Algorithm 2 Capacitance

```
function Get Capacitance
    for all Walks do
        Record result of Do Random Walk
    end for
    Reduce results to fraction of Hits
    return fraction of Hits × bounding sphere radius
end function
```

tion, or complement) of M simple geometric shapes, this query requires at most $O(M)$ distance computations. Since many objects have very large M, we need algorithms that can do much better than $O(M)$ using additive geometry representations. Furthermore, we expect the number of queries N to be much larger than M (e.g., $M = 10^4$ and $N = 10^7$ to 10^8). As such, a significant amount of preprocessing time in terms of M for building an appropriate spatial data structure is acceptable, as this cost will be amortized over all queries.

4 Geometry Data Structures and Algorithms

As confirmed by our analysis in Section 2, the main computational operation in the WoS algorithm is finding the nearest geometric primitive to a given query point. As such, we are evaluating several data structures to identify the ones that can effectively answer a very large number of these queries (e.g., 10^7 to 10^8).

In this evaluation, we will consider a data structure's creation time (labeled as *pre-processing*) and *search* time for 10^7 queries. We will also compare the various data structures with the naive approach that uses a full search. Figures 1 and 2 show molecular complexes consisting of 23 115 and 12 750 spheres, respectively, that we will use as our evaluation examples.

Our current application uses spheres as its geometric primitive. As such, we seek data structures that can answer closest-sphere queries. There are many off-the-shelf algorithms and libraries to handle closest-point queries [2,4,18,19], but relatively few that handle closest-sphere queries directly. However, we can readily extend closest-point queries to handle spheres in one of two ways: (1) binning spheres with the same radius, solving the closest-point queries for the sphere centers in each bin, mapping the closest-point in each bin back to the corresponding radius, and selecting the closest sphere; (2) sampling sphere surfaces and applying the closest-point queries to the resulting point samples. Currently we are using the first method; this approach works well as long as the number of bins (number of sphere radii) remains small.

We have evaluated several k-D-tree-based implementations from the ANN 1.1.2 [2], FLANN 1.8.4 [18], nanoflann 1.1.8 [4], and CGAL 4.6 [19] libraries. In the future, we may also investigate grid-based algorithms, such as the one described by Franklin [8], or algorithms based on AABB-trees (Axially-Aligned Bounding Box), such as the one from CGAL [1], which may be capable of using spheres as their primitive.

In order to decide which data structure library to use in our C++ version of ZENO, we conducted a series of benchmarks. For each data structure we inserted a set of spheres (based on their center points, as described above) and then conducted a set of closest-sphere queries from randomly chosen points. The results are given in Table 1. Based on these benchmarks, we chose the nanoflann library [4] for our implementation.

Algorithm	Preprocess (ms)	Search (s)	Speedup
Linear	0	175 ± 0.4	1×
CGAL k-D	0.852 ± 0.037	2.67 ± 0.09	65.5×
ANN	9.38 ± 0.62	5.25 ± 0.04	33.3×
FLANN	4.59 ± 0.36	5.83 ± 0.15	30.0×
nanoflann	3.99 ± 0.16	2.43 ± 0.02	72.0×

Table 1: 10^6 closest-sphere queries on the example from Figure 1 run on a desktop PC[1]. Errors are ± 2 standard deviations of the mean with 20 replications.

The data structures discussed so far can be classified as *primitive-partitioning*, since they divide space into regions that contain small groups of input primitives. Answering the closest-primitive query consists of finding and checking the region in which the query point lies and neighboring regions to determine the closest. Other data structures, such as Voronoi diagrams, can be classified as *space-partitioning*, in which the points in each region share the same closest primitive. For these data structures, answering the closest-primitive query consists of finding the region in which the query point lies and checking the primitives in that region to see which is the closest; neighboring regions do not have to be searched.

5 Parallel Implementation

The ZENO/WoS algorithm from Section 3 is "embarrassingly parallel" in the sense that each walk and each sample are completely independent from every other walk and sample. As such, our parallelization schemes are fairly straight-forward.

[1] Desktop PC used contains dual Intel Xeon 2.3 GHz CPUs (total of 12 physical and 24 logical cores), 128 GiB RAM, Ubuntu 14.04, Kernel 3.13.0 x86_64, GCC 4.8.4 with -O3

Stochastic parallel programs require multiple independent streams of pseudo-random numbers. We generate these independent streams by using the "Scalable Parallel Random Number Generators" library (SPRNG) [17].

We have implemented two different parallelization schemes, one based on C++ Standard Library threads (using C++11) and the other based on MPI. These can be used simultaneously to run multiple threads on each node in a cluster of nodes connected via MPI.

For the shared-memory multi-threading parallelization, we first insert the input spheres into the k-D tree data structure. Since the k-D tree query functions do not modify the data structure state, we can simply have each thread execute the same walk or sample code as in the serial version. Once all the threads have completed, we perform a simple reduction operation by summing their results.

For the MPI parallelization, memory is not shared. As such, we first broadcast the set of input spheres to each process using MPI. Each process then builds its own local copy of the k-D tree data structure and performs its own walks and samples, either serially or using shared-memory multi-threading as described above. Once the processes complete, their results are combined using a reduction operation.

6 Results

Table 2 gives the speedups resulting from the various features of our implementation: the C++ port of the algorithm with a linear data structure, the addition of the nanoflann spatial data structure library, and multi-threaded and multi-node parallelism. On the Gold Nanoparticle example, we achieve over 4 orders of magnitude speedup over the Fortran 2008 implementation for the ZENO/WoS computation.

	Gold Nanoparticle			*Protein 1IGY*		
	Time (s)	Speedup	Cumul.	Time (s)	Speedup	Cumul.
Fortran 2008	8240 ± 110	1×	1×	3710 ± 6	1×	1×
C++/Linear	3970 ± 4	2.08×	2.08×	2090 ± 2	1.77×	1.77×
C++	42.0 ± 0.1	94.5×	196×	71.5 ± 0.1	29.3×	52.0×
C++ 24 T	3.27 ± 0.11	12.9×	2520×	5.42 ± 0.29	13.2×	685×
C++ 24 T, 8 N	0.797 ± 0.026	4.10×	10300×	1.14 ± 0.10	4.78×	3270×

Table 2: Single-node speedups for ZENO/WoS computation on the Gold Nanoparticle example (Figure 1) and PDB protein 1IGY (Figure 2) with 10^6 walks. C++/Linear performs exhaustive search; C++ uses nanoflann; T is the number of threads and N is the number of compute nodes. Errors are ±2 standard deviations of the mean with 20 replications. Benchmarks run on Raritan cluster[1].

Figure 4 illustrates the strong scaling of our implementation as the degree of parallelism increases for a fixed problem size. For these tests, we measured the effect of increasing either the number of threads per node or the number of compute nodes in a cluster communicating with MPI. The times reported are the total running times for the entire computation, including reading data from disk, distributing it among the nodes, performing the ZENO/WoS computation, and combining the results. On the Gold Nanoparticle example, we achieve greater than

[1]Raritan cluster nodes used contain dual Intel Xeon 2.6 GHz CPUs (total of 12 physical and 24 logical cores), 8 GiB RAM, RHEL 5.11, Kernel 2.6.18 x86_64, GCC 4.9.3 with -O3, MPICH 3.1.4

19x speedup with our multi-threaded single-node implementation with 24 threads (as shown in Table 2) through the use of multi-node parallelism with 32 nodes.

We validated the correctness of our results by comparing several output values (including capacitance and eigenvalues of the polarizability tensor) against theoretical values in the case of a sphere, and against the Fortran 2008 version's values for both a sphere as well as the examples in Figures 1 and 2. Our results were generally within 1 % of both the theoretical values and the Fortran 2008 version's values; these error bounds are similar to those of the Fortran 2008 version. The magnitude of these errors is consistent with what would be expected due to the algorithm's stochastic nature.

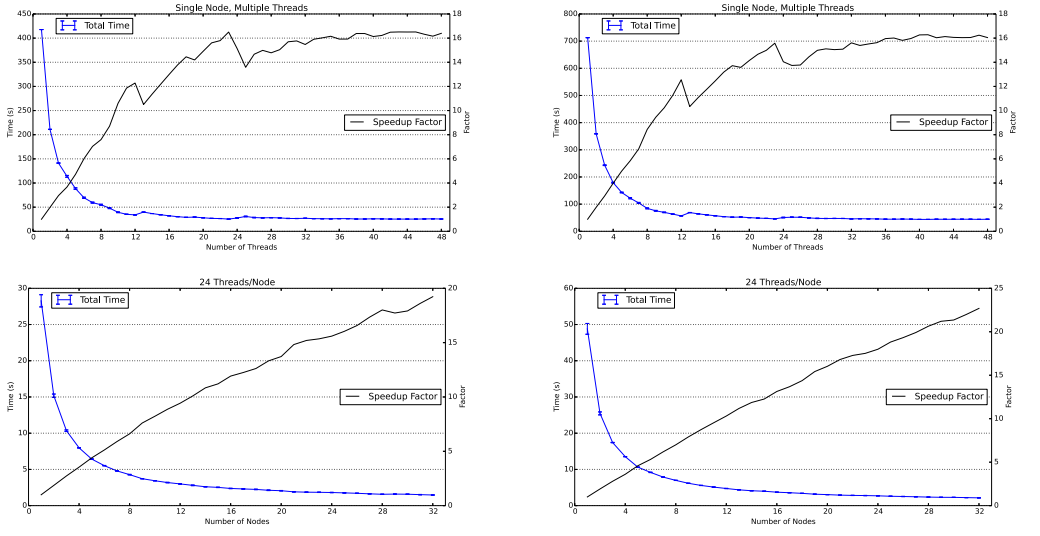

Figure 4: Strong scaling on Gold Nanoparticle example (Figure 1) (left) and PDB protein 1IGY (Figure 2) (right) with 10^7 walks. The first row demonstrates scaling threads on a single node. The second row demonstrates scaling nodes with 24 threads per node. Error bars are ± 2 standard deviations of the mean with 20 replications. Benchmarks run on Raritan cluster[1].

7 Conclusion and Future Work

We have examined the scalability of various spatial data structures and parallel performance in a new parallel C++ implementation of the ZENO/WoS algorithm. This new version improves on the Fortran-based implementations by 3 to 4 orders of magnitude; it should return results in a timely manner for models with up to 10^5 to 10^6 spheres using 10^7 to 10^8 walks. Also, and perhaps more importantly, such a performance improvement will allow ZENO/WoS to be used for new classes of applications, for example as the kernel call in an inner loop for searching a materials design space. We intend to release the ZENO C++ code in the near future.

In the future, we want to investigate the use of input types other than a union of spheres, such as triangle meshes or voxels. This will allow the program to be applied to a much wider range of objects, such as smooth molecular surfaces or even macro-scale objects.

[1]Raritan cluster nodes used contain dual Intel Xeon 2.6 GHz CPUs (total of 12 physical and 24 logical cores), 8 GiB RAM, RHEL 5.11, Kernel 2.6.18 x86_64, GCC 4.9.3 with -O3, MPICH 3.1.4

We are currently experimenting with point-sampled surfaces, which can enable support for a wide variety of shapes and primitives. In our current implementation, point samples are treated as discs of a certain radius that cover the surface of the object. For the ZENO/WoS computation, we approximate the nearest surface point on the object by the nearest point on the disc corresponding to the nearest sample point on the object. We can easily find the nearest sample point by using the data structures from Section 4. An alternative or complement to point-sampling is to develop and implement a family of Walk-on-X methods such as Walk-on-Planes and Walk-on-Parallelepipeds [6, 13].

Another potential area of future work is to investigate the use of approximate nearest-neighbor search, which returns a neighbor to which the distance is within some user-defined percent error of the distance to the true nearest neighbor. This is supported by most of the libraries we have tested. Approximate nearest-neighbor search is generally used for high-dimensional data where exhaustive search can be prohibitively expensive, but since we expect our computations to be very tolerant to error, we may be able to significantly benefit from it.

We may also investigate the use of space-partitioning data structures, as discussed in Section 4, using either spheres or points as primitives. Yet another research direction is to develop a tree-based data structure that is custom-built for the 3D nearest-neighbor queries encountered in the WoS method.

Acknowledgments

We want to acknowledge the help, explanations, and many fruitful discussions that we had with our NIST colleagues at the Material Measurement Laboratory, Debra Audus, Fernando Vargas-Lara, and Jack Douglas.

Disclaimer

Certain commercial software, products, and systems are identified in this report to facilitate better understanding. Such identification does not imply recommendations or endorsement by NIST nor does it imply that the software and products identified are necessarily the best available for the purpose.

References

[1] Pierre Alliez, Stéphane Tayeb, and Camille Wormser. 3D fast intersection and distance computation. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.6 edition, 2015.

[2] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.

[3] Helen Berman, Kim Henrick, and Haruki Nakamura. Announcing the worldwide Protein Data Bank. *Nat Struct Mol Biol*, 10(12):980–980, December 2003.

[4] Jose Luis Blanco. nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) wih KD-trees. https://github.com/jlblancoc/nanoflann, 2014. Accessed 2015-08-11.

[5] Emre Brookes, Borries Demeler, Camillo Rosano, and Mattia Rocco. The implementation of SOMO (SOlution MOdeller) in the UltraScan analytical ultracentrifugation data analysis suite: enhanced capabilities allow the reliable hydrodynamic modeling of virtually any kind of biomacromolecule. *European Biophysics Journal*, 39(3):423–435, February 2010.

[6] Madalina Deaconu and Antoine Lejay. A random walk on rectangles algorithm. *Methodology and Computing in Applied Probability*, 8(1):135–151, March 2006.

[7] Jack F. Douglas, Huan-Xiang Zhou, and Joseph B. Hubbard. Hydrodynamic friction and the capacitance of arbitrarily shaped objects. *Phys. Rev. E*, 49(6):5319–5331, June 1994.

[8] Wm. Randolph Franklin. Nearest point query on 184M points in E3 with a uniform grid. In *Proceedings of the 17th Canadian Conference on Computational Geometry, CCCG'05, University of Windsor, Ontario, Canada, August 10-12, 2005*, pages 239–242, 2005.

[9] Lisa J. Harris, Eileen Skaletsky, and Alexander McPherson. Crystallographic structure of an intact IgG1 monoclonal antibody. *Journal of Molecular Biology*, 275(5):861–872, February 1998.

[10] Joseph B. Hubbard and Jack F. Douglas. Hydrodynamic friction of arbitrarily shaped Brownian particles. *Phys. Rev. E*, 47(5):R2983–R2986, May 1993.

[11] Eun-Hee Kang, Marc L. Mansfield, and Jack F. Douglas. Numerical path integration technique for the calculation of transport properties of proteins. *Phys. Rev. E*, 69(3):031918, March 2004.

[12] Eun-Hee Kang, Marc L. Mansfield, and Jack F. Douglas. ZENO: an efficient method for charaterizing object shape and for calculating transport properties of nanoparticles and synthetic and biological macromolecules. http://web.stevens.edu/zeno/, 2006. Accessed 2015-08-10.

[13] In Chan Kim. An efficient brownian motion simulation method for the conductivity of a digitized composite medium. *KSME Internation Journal*, 17(4):545–561, April 2003.

[14] Seung Hyeon Ko, Fernando Vargas-Lara, Paul N. Patrone, Samuel M. Stavis, Francis W. Starr, Jack F. Douglas, and J. Alexander Liddle. High-speed, high-purity separation of gold nanoparticle–dna origami constructs using centrifugation. *Soft Matter*, 10(37):7370–7378, 2014.

[15] Marc L. Mansfield and Jack F. Douglas. Improved path integration method for estimating the intrinsic viscosity of arbitrarily shaped particles. *Phys. Rev. E*, 78(4):046712, October 2008.

[16] Marc L. Mansfield, Jack F. Douglas, and Edward J. Garboczi. Intrinsic viscosity and the electrical polarizability of arbitrarily shaped objects. *Phys. Rev. E*, 64(6):061401, November 2001.

[17] Michael Mascagni and Ashok Srinivasan. Algorithm 806: SPRNG: A scalable library for pseudo-random number generation. *ACM Trans. on Math. Software (TOMS)*, 26(3):436–461, 2000.

[18] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, Nov 2014.

[19] Hans Tangelder and Andreas Fabri. dD spatial searching. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.6 edition, 2015.

[20] Wenjian Yu, Hao Zhuang, Chao Zhang, Gang Hu, and Zhi Liu. RWCap: A floating random walk solver for 3-d capacitance extraction of very-large-scale integration interconnects. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 32(3):353–366, March 2013.

[21] Chao Zhang and Wenjian Yu. Efficient space management techniques for large-scale interconnect capacitance extraction with floating random walks. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 32(10):1633–1637, October 2013.

[22] Huan-Xiang Zhou. On the calculation of diffusive reaction rates using Brownian dynamics simulations. *The Journal of Chemical Physics*, 92(5):3092–3095, 1990.

[23] Huan-Xiang Zhou, Attila Szabo, Jack F. Douglas, and Joseph B. Hubbard. A Brownian dynamics algorithm for calculating the hydrodynamic friction and the electrostatic capacitance of an arbitrarily shaped object. *The Journal of Chemical Physics*, 100(5):3821–3826, 1994.

[24] Peter Zipper and Helmut Durchschlag. Hydrodynamic multibead modeling: problems, pitfalls, and solutions. 1. ellipsoid models. *European Biophysics Journal*, 39(3):437–447, 2010.

[25] Peter Zipper and Helmut Durchschlag. Hydrodynamic multibead modeling: problems, pitfalls, and solutions. 2. proteins. *European Biophysics Journal*, 39(3):481–495, 2010.

[26] Peter Zipper and Helmut Durchschlag. Hydrodynamic multibead modeling: problems, pitfalls and solutions. 3. comparison of new approaches for improved predictions of translational properties. *European Biophysics Journal*, 42(7):559–573, 2013.