JOURNAL OF CHEMICAL INFORMATION AND MODELING

Article

pubs.acs.org/jcim

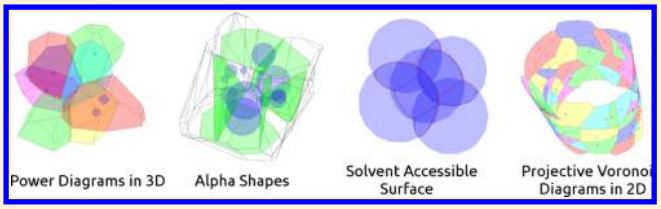
# DMG-$\alpha$—A Computational Geometry Library for Multimolecular Systems

Robert Szczelina*[,†,‡] and Krzysztof Murzyn[§]

[†]Faculty of Mathematics and Computer Science, [‡]Malopolska Centre of Biotechnology, and [§]Faculty of Biochemistry, Biophysics and Biotechnology, Department of Computational Biophysics and Bioinformatics, Jagiellonian University, 31-007 Krakow, Poland

Supporting Information

**ABSTRACT:** The DMG-$\alpha$ library grants researchers in the field of computational biology, chemistry, and biophysics access to an open-sourced, easy to use, and intuitive software for performing fine-grained geometric analysis of molecular systems. The library is capable of computing power diagrams (weighted Voronoi diagrams) in three dimensions with 3D periodic boundary conditions, computing approximate projective 2D Voronoi diagrams on arbitrarily defined surfaces, performing shape properties recognition using $\alpha$-shape theory and can do exact Solvent Accessible Surface Area (SASA) computation. The software is written mainly as a template-based C++ library for greater performance, but a rich Python interface (`pydmga`) is provided as a convenient way to manipulate the DMG-$\alpha$ routines. To illustrate possible applications of the DMG-$\alpha$ library, we present results of sample analyses which allowed to determine nontrivial geometric properties of two *Escherichia coli*-specific lipids as emerging from molecular dynamics simulations of relevant model bilayers.




## INTRODUCTION

In recent years, Molecular Dynamics (MD) simulation has become one of the most successful methods of modeling atom interactions allowing to gain insight at the atomic resolution into molecular bases for many biological pheonomena. The direct result of an MD simulation is a set of consecutive configurations of all atoms in a molecular system (a trajectory), which are usually equally spaced in time. A careful analysis of such a trajectory allows to extract the meaningful information describing physical properties of the system. While different molecular systems require determination of usually distinct sets of descriptive parameters, the elaborate analyses of such systems are often based on calculated interatomic distances and extraction of information related to the nearest neighborhood of an atom. For example, in an analysis of MD simulation trajectories of model lipid bilayers, the set of routinely determined parameters includes surface area per lipid molecule (SA), the thickness of the membrane, molecular volume ($V_m$) and, in specific case studies, properties such as the topology of membrane pores,[1] the number of water molecules trapped inside the membrane, and dimensions of ion channels,[2−4] to name just a few. On the other hand, while studying individual protein molecules or protein complexes, it is often useful to calculate the Solvent Accessible Surface Area (SASA),[5,6] discriminate between native and crystal contacts made by protein domains, identify the superficial and buried amino acid residues,[7] and characterize the binding pockets which are large enough to accommodate a ligand.[8]

All the quantities mentioned above depend on the three-dimensional (3D) structure of the molecular system, at the atomic scale represented by the union of balls. Analysis of the geometric properties of an union of balls can be in general performed in two ways, i.e. by approximate and exact geometrical computations.[9]

The approximate methods usually employ Monte Carlo-like methods to create a finite representation of a set of points in 3D space contained in the union of balls. The main advantage of approximate algorithms is the simplicity of their implementation and straightforward parallelization.

Exact computations use various kinds of space filling diagrams and tessellations, usually power diagrams (weighted Voronoi diagrams) or their dual structures called regular triangulations. For a given set $\mathbb{S}$ of three-dimensional spheres $S_i$, $i \in \{1, ..., n\}$ described by centers $p_i$ and radii $r_i$ a *3D weighted Voronoi diagram* (a *power diagram*) of $\mathbb{S}$ is defined as a division of space $\mathbb{R}^3$ into polyhedral sets $C_i$ such that for each point $p \in C_i$ we have $\|p - p_i\|^2 - r_i^2 \leq \|p - p_j\|^2 - r_j$ for all $j \neq i$, where $\|\cdot\|$ denotes euclidean norm in 3D. Spheres $S_i$ and $S_j$ for which $C_i \cap C_j$ is a 2D polygon are called *neighbors*. The dual structure, a *regular triangulation*, is a tessellation of a set of points $\{p_i\}$ consisting of all tetrahedrons defined with four centers $\{p_i, p_j, p_k, p_l\}$ such that $C_i \cap C_j \cap C_k \cap C_l$ consist of a single point. Equivalently, each face of a tetrahedron in a regular triangulation is defined by three centers of spheres which are neighbors in a power diagram. The power diagram or the regular triangulation may be used to test proximity relations between spheres and to compute efficiently various quantities related to the structure of the union of balls such as the total surface area and volume. It can also be used to investigate topological properties of the set such as cavity and

**Received:** May 7, 2014
**Published:** October 8, 2014

ACS Publications © 2014 American Chemical Society



dx.doi.org/10.1021/ci500273s | *J. Chem. Inf. Model.* 2014, 54, 3112−3123

tunnel detection. This versatility, however, comes at the cost of a significantly higher complexity of the algorithms than in the case of approximate algorithms.

The exact computation methods have received much attention in recent years.[5,10−15] Nonetheless, it seems that approximate methods are still widely used in many available applications.[9,16−20] This might be due to the fact that only a scarce of a freely available software for efficient computing of tesselations in 3D exists: CGAL,[21] Voro++,[22] Quickhull,[23] DeWall,[24] and VoroProt[25] and most of them are not straightforward to use. Moreover, the algorithms used in these libraries seem to be not well suited for use in the molecular setting, as they are focused on handling topological correctness for any, even the most degenerate, case which occur rarely in molecular systems. What is more, the most commonly employed incremental flipping algorithm (for a regular triangulation, a kind of tessellation commonly used in many applications) has very poor performance on big, uniformly distributed systems.[26] Finally, let us note that to our best knowledge handling periodic boundary conditions when dealing with periodic systems such as lipid membranes or protein crystals is supported only by CGAL and Voro++ and this fact strongly limits applicability of other libraries in such cases.

Since the libraries mentioned above are written in C++, the ability for easy scripting is severely restricted for people with no computer science background. In last decades we could see growing interest in using Python as a scripting language in various fields of science. The Python language has acquired a great scientific audience in last years partly because it is a high-level script language with easy to learn syntax and many available libraries for scientific computing such as numpy (www.numpy.org) or scipy (www.scipy.org). Python has the ability to blend with libraries written in other languages such as C, C++, or FORTRAN, which makes it a good choice for scientific programming[27] in various disciplines of science, including the field of molecular modeling.[28] To our knowledge only the Quickhull[23] library is available for Python programmers allowing them to compute Voronoi diagrams in 3D (as part of scipy[29] project); however, it does not support power diagrams (weighted Voronoi diagrams) and periodic boundary conditions. CGAL, which seems to be the obvious choice as a source of valuable implementations of numerous computational geometry (CG) algorithms, has ceased to support its Python interface recently, which effectively renders this library almost unusable for people outside the community of computer scientists involved in active development of CG applications due to its high level of abstraction.

The DMG-$\alpha$ library is an attempt to address the problem of insufficient accessibility of important CG algorithms to researchers in the field of the computational biology, chemistry and biophysics. The solution that we propose is an open-sourced, easy to use and intuitive library for performing geometric analysis of molecular systems, which efficiently and elegantly handles multimolecular systems, including those under periodic boundary conditions. While developing the DMG-$\alpha$ library, we have put a lot of effort to provide the possibly finest access to the underlying DMG-$\alpha$ data structures through the rich, practically self-explanatory Python interface, which hopefully allows for extensive scripting and straightforward implementations of algorithms for customized data analysis—the feature which distinguishes methodology of DMG-$\alpha$ from other, specialized software packages. We believe that, contrary to the currently available command line and GUI utilities[25,30,31] implementing various CG algorithms, the possible applications of the DMG-$\alpha$ library are not limited to the analyses of particular aspects of the molecular systems (e.g., cavity detection, pockets, tunnels, or detecting binding sites). Instead, the Python interface being the integral part of the DMG-$\alpha$ library makes development of virtually any application employing relevant CG algorithms based on the power diagram construction a relatively easy and moderately time-consuming task for many researchers, especially those working in the field of computational biophysics and crystallography.

The present paper, which introduces the DMG-$\alpha$ library and its Python interface (pydmga), is organized as follows. To make this paper more self-contained in the Methods section, we briefly present the theoretical background of calculations of Voronoi diagrams in 3D, projective geometries, $\alpha$-shapes, and solvent accessible surface, which are implemented in the library. In the Results and Discussion section, we present design principles of the library and the implementation details together with sample codes in Python. In the Application to Lipid Bilayer MD Simulations section, we show possible applications of the DMG-$\alpha$ library in a real case scenario where we perform nontrivial calculations of selected geometric properties of two *Escherichia coli*-specific lipids as emerging from MD simulations of the relevant fully hydrated lipid bilayers under 3D periodic boundary conditions. We also discuss performance of the DMG-$\alpha$ library basing on results of the carefully designed benchmarks. Finally, in the Conclusions, we emphasize the most important features of the DMG-$\alpha$ library and reiterate the most important aspects of its implementation to end with outlining the future plans for development of the library and its Python interface.

## METHODS

In this section we gather all necessary definitions to make this work more self-contained. We also describe how the mentioned structures and algorithms are used in DMG-$\alpha$. Overview of the design of the library along with sample codes and an exemplary application to a real-life problem are presented in the Results and Discussion.

The present version of the DMG-$\alpha$ library implements algorithms for computing weighted Voronoi diagrams (power diagrams) in 3D, Weighted Voronoi diagrams for projections to arbitrary 2D surfaces, $\alpha$-shape-like constructs, and exact computation of SASA. Algorithms used in DMG-$\alpha$ support 3D periodic boundary conditions and extensively use local properties of densely packed molecular systems for high performance, flexibility, and easy parallelization of computations.

**Voronoi Diagram in 3D.** In the original definition of a power diagram[33] the cells $C_i$ sum up to the whole space (i.e., $\mathbb{R}^3$). In particular, this means that in a classic power diagram, there exist some *infinite* cells (the outer cells). In case of periodic molecular systems (e.g., lipid bilayer configurations generated in MD simulations) the configuration space is limited to some compact subset $M$ of $\mathbb{R}^3$, usually some box with finite dimensions and with the arbitrarily set origin of the reference frame. We will call the set $M$ the *master cell* in the sequel. As the master cell $M$ is compact, that is it has finite diameter, the power diagram constructed on $M$ is made exclusively of finite Voronoi cells, which are defined as above but only for points inside the master cell. Observe that in a periodic system some Voronoi cells may have sides which are defined by its neighbors determined according to the nearest image convention.

The regular triangulation is a concept *dual* to the power diagram (assuming property of *general position* of the spheres[34]),

in the sense that we can construct one from the another. When general position is assumed then it can be shown that there are no five spheres $S_{i_1}$, ..., $S_{i_5}$ such that $C_{i_1} \cap \cdots \cap C_{i_5} \neq \varnothing$. Then for each four spheres $S_{i_1}$, ..., $S_{i_4}$ such that $C_{i_1} \cap \cdots \cap C_{i_4} \neq \varnothing$, we can create tetrahedron $T$ with vertexes at $p_{i_1}$, ..., $p_{i_4}$. The set of all such $T$ is called a regular triangulation of $\mathbb{S}$ and it is a 3D tessellation of a set of points $\{p_i\}$ into tetrahedrons that intersects only on their faces or edges (interiors of those tetrahedrons are disjoint).

Given a finite-size master cell, we may define some of its directions *periodic*, i.e. we virtually create copies of the master cell and all spheres in the given periodic direction. Up to 26 additional master cells may be needed to impose periodic boundary conditions in a case when all three directions are periodic. Such a brute-force approach involving real periodic copies of all spheres should be avoided since it impacts the performance and memory usage of any method of computing periodic Voronoi diagrams. To our knowledge, there are currently only two publicly available software libraries for computing Voronoi diagrams (or its dual Delaunay triangulations) which support periodic boundary conditions explicitly: `CGAL` for 3D Delaunay triangulations and `Voro++`. The concepts of the diagram cell, the diagram limited to the master cell, and the periodic Voronoi diagram are visualized in Figure 1.

The DMG-$\alpha$ library uses `Voro++` routines[22] for computing basic power diagrams and provides completely new procedures for custom iteration over Voronoi cells. Procedures implemented in `Voro++` employ the cutting plane approach consisting of consecutive cuts of the master cell by the power planes. The *power plane* between spheres $S_i$ and $S_j$ is the set of points in the 3D space for which $\|p - p_i\|^2 - r_i^2 = \|p - p_j\|^2 - r_j$. The value $\|p - p_i\|^2 - r_i^2$ is called the *power distance* of $p$ to sphere $S_i$. The cutting plane algorithm computes the Voronoi cell for each sphere $S_i$ separately by consecutive cuts of the master cell by the power planes between $S_i$ and all other spheres $S_j$, $j \neq i$. To speed up computations, the algorithm performs cuts by the planes ordered by the power distance. This algorithm is simply extended to handle periodic boundary conditions (without the need for duplication of the master cell) by using the nearest image convention.

The resulting Voronoi cell is stored by the `Voro++` library in a structure similar to a doubly connected edge list representing a polyhedra in 3D. The `Voro++` library in its original form does not support directly basic operations to navigate around the Voronoi cell. Instead, it only allows to print aggregated information about volume, total area and neighbors. In the DMG-$\alpha$ library we have implemented custom iterators to be able to directly access vertices, edges, and sides of rather complicated Voronoi cell data structure. The iterators allow for easy identification of cell elements (vertexes, edges, sides), navigate through them and update the data structure. Those operations are essential for constructing custom algorithms described further in this article. Another additional feature we have added in the DMG-$\alpha$ library is support for generic data structures to be hold along the atoms (in `Voro++` only position and radius are held for each sphere).

In comparison to other software libraries (i.e., `CGAL`,[21] `DeWall`[24]), we actually operate directly on a power diagram instead of a regular triangulation. The regular triangulation gives only qualitative information on the local neighborhood and it needs to be transformed into the power diagram at an extra computational cost to extract quantitative data (e.g., cell volume and area). Moreover, it should be noted here that regular triangulations are usually computed with assumption that the set of spheres is in general position which is achieved by symbolic perturbation techniques like simulation of simplicity (SoS)[35] and that may further affect performance of the procedure of recovering of the power diagram.

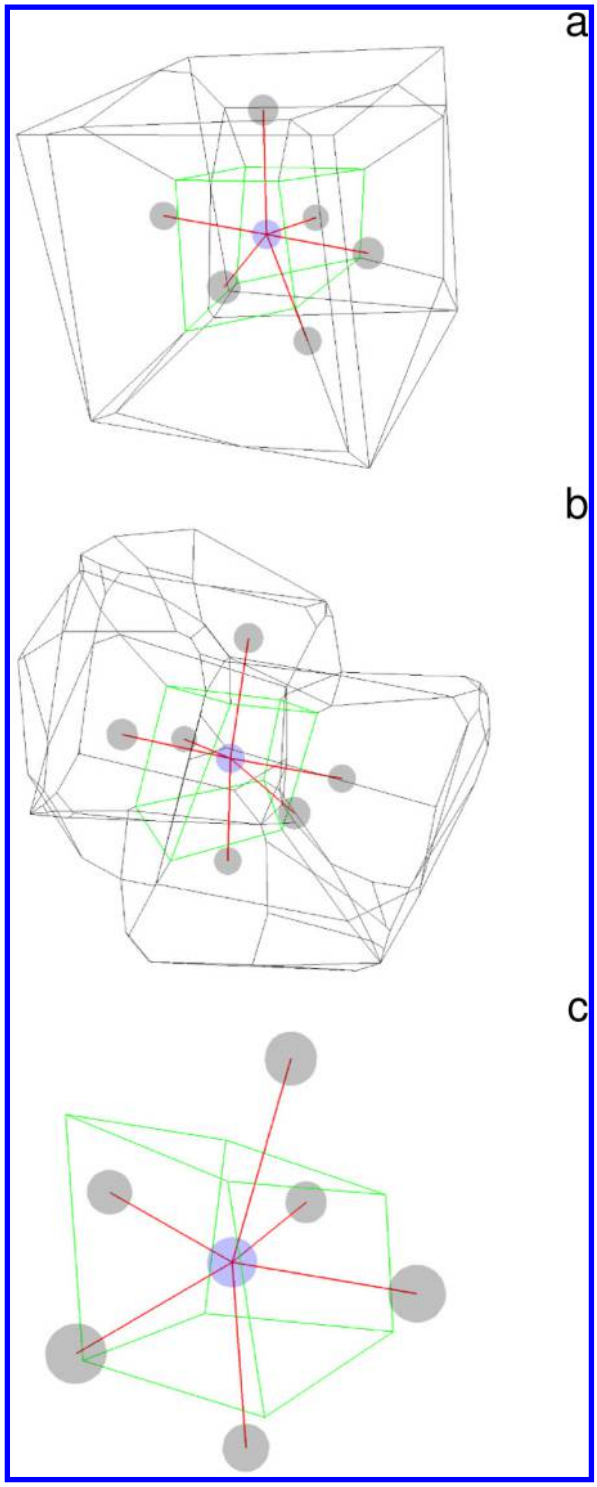



**Figure 1.** (a) 3D Voronoi diagram for a set of seven spheres inside a master cell. No periodic conditions were applied, so the power diagram fits perfectly into the master cell. (b) Voronoi diagram for the same spheres with 3D periodic boundary conditions. (c) Close-up on the middle cell of both diagrams. Neighbors connected to the central sphere are shown with red lines.

The expected linear running time for the method of power plane cuts is achieved by the assumption that spheres are distributed uniformly across the master cell, which allows to apply an optimization technique of localized computations. In principal, the master cell is divided into smaller cells $b_{ijq}$ which should contain small number of spheres each. Namely, there should exist constant $d$, sufficiently small, that each cell contain at most $d$ spheres. We start computation with the whole master cell centered at a given sphere in some cell $b_{ijq}$, then we cut it with cutting planes defined by the given sphere and all other spheres in $b_{ijq}$. During computations, we are able to determine the diameter of the newly created cell to test if we need to proceed to spheres in neighboring cells $b_{i\pm1,q\pm1,q\pm1}$. The heuristics used here is as follows: when spheres are uniformly distributed then it suffices to check small number of neighboring cells close to the given sphere to construct its Voronoi cell. A similar approach was used in `PowerSASA`,[15] where authors used chemical bonds extracted from the input PDB file to start construction of a cell from (probably) the nearest neighbor. Then the `PowerSASA` algorithm found the real closest neighbor and used it to create a Voronoi cell for it. In contrast to the `PowerSASA` algorithm, the cutting plane method with localization can be used even when no information on chemical bonds is available.

For the optimal performance, $d$ should be chosen between 3 and 12.[36] We use $d = 6$ and compute the division of the master cell dynamically based on the total number of atoms in the molecular system under study. This approach allows for efficient handling of periodic boundary conditions, as there are only atoms in neighboring cells on the other side of the master cell which need to be checked while constructing a current Voronoi cell. We give detailed analysis of the algorithm performance in the appropriate section of Results and Discussion.

**Projective Geometries.** One of the advantages of the power diagram construction approach based on cutting planes described in the previous section is its flexibility. This allows for simple modifications of computed cells by additional cuts. This may result in many useful procedures, for example in restricting possible domains to other master cell shapes or for computing approximate power diagrams of projections onto the given 2D surface. Those geometries may be simply implemented for arbitrary shapes of the surfaces using `C++` API, provided that the shape being approximated is at least locally convex (for example spheres, cylinders, toruses, etc. which may occur in some biological configurations). This approach has an advantage over computing classical 2D Voronoi diagrams as it can be used straightforward on the input set of points without finding transformation from the surface of the object to the plane and vice versa. Moreover, for some shapes the transformation may not be available due to the complicated topology of the surface, e.g. for two connected toruses.

We implemented basic projective geometries and provided a Python interface for them. Those projections are realized in the DMG-$\alpha$ library as a special kind of geometries with the `Cast` prefix. Currently there are two kinds of projective geometries: `CastOrthoPlaneGeometry` (see an example in Figure 2) and `CastCylinderGeometry`. The first one was implemented to allow creation of standard 2D Voronoi diagrams inside the framework of DMG-$\alpha$ library which can be used for examples in computation of surface area per lipid in the lipid membrane.[3] The second one allows computation of projection onto the cylinder and can be used for examples in investigating lipid hexagonal phases.

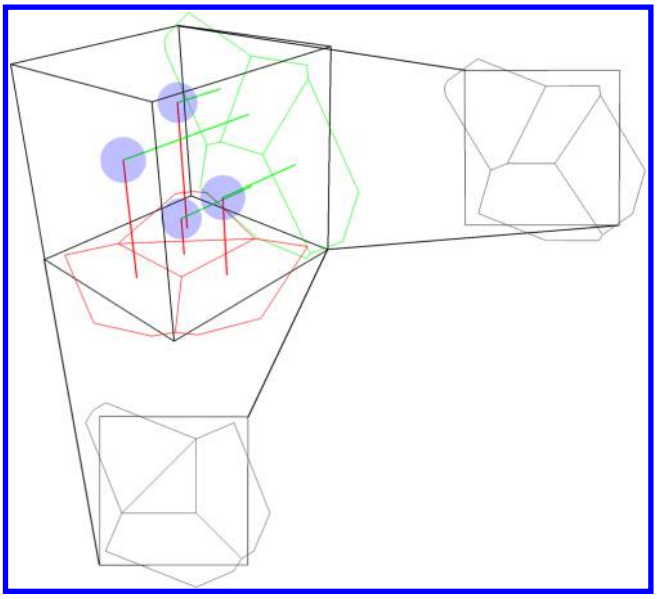


**Figure 2.** Two of three possible orthogonal projections to a plane for a simple periodic system for four spheres. Those projections are realized as `CastOrthoPlaneGeometry` in the DMG-$\alpha$ library.

**Alpha Shapes and Solvent Accessible Surface.** Given a power diagram of a set of atoms we may ask what geometric properties it has. In the last few decades, one of the fundamental concepts in CG was the concept of a shape of a geometric set, usually a shape of a set of points or balls. This question has given rise to a new field: computational topology which is one of the most rapidly evolving branch of CG. Nowadays, a commonly accepted definition of a shape of a set of balls (or points) can be obtained using concept of an *$\alpha$-shape* (alpha shape).[37−39] An $\alpha$-shape, defined for a given parameter value $\alpha \geq 0$, is a subset of the regular triangulation which is topologically equivalent to the union of all balls $S_i$ with modified radius $r_i' = (r_i^2 + \alpha)^{1/2}$ (Figure 3). For $\alpha = 0$ we get the set which is topologically equivalent to the union of spheres in the input set $\mathbb{S}$. Topological equivalence, in principal, is a relation between geometrical sets that preserves its topological properties—number of holes, connectivity, etc.[40] The information enclosed in an $\alpha$-shape can be then used to identify pockets and holes in the configuration of atoms where other molecules can enter. Or, vice versa, it can be used to locate blocked pathways. The definition of the value of $r_i'$ is crucial in the efficient construction of the $\alpha$-shape as it can be shown that for spheres with $r_i' = (r_i^2 + \alpha)^{1/2}$ the regular triangulation is the same as for spheres with unmodified radii $r_i$.

As the regular triangulations are dual to the concept of power diagrams, we may also use a power diagram to construct a data structure that represents the same information as the $\alpha$-shape. In the DMG-$\alpha$ library it is done in the following way: we can identify alpha value $\alpha_{\text{threshold}}$ at which given part of a Voronoi cell will be first crossed by a ball with radius $r_i'$ located at $p_i$ and we say that this part is in the $\alpha$-shape for $\alpha \geq \alpha_{\text{threshold}}$. In three dimensions we have three parts of a cell: sides, edges, and vertexes. It can be shown that channels in molecules can be identified using cell edges.[30,31] Thus, identifying which edge is blocked i.e. is in $\alpha$-shape for a given $\alpha$ (usually 0) is crucial in identifying pockets and tunnels inside molecules.

A similar concept to the $\alpha$-shape, but much more practical from the point of view of a biophysicist or a chemist is the concept of solvent accessible surface (SAS), particularly the area of SAS (SASA). SASA is defined as a total area of the union of

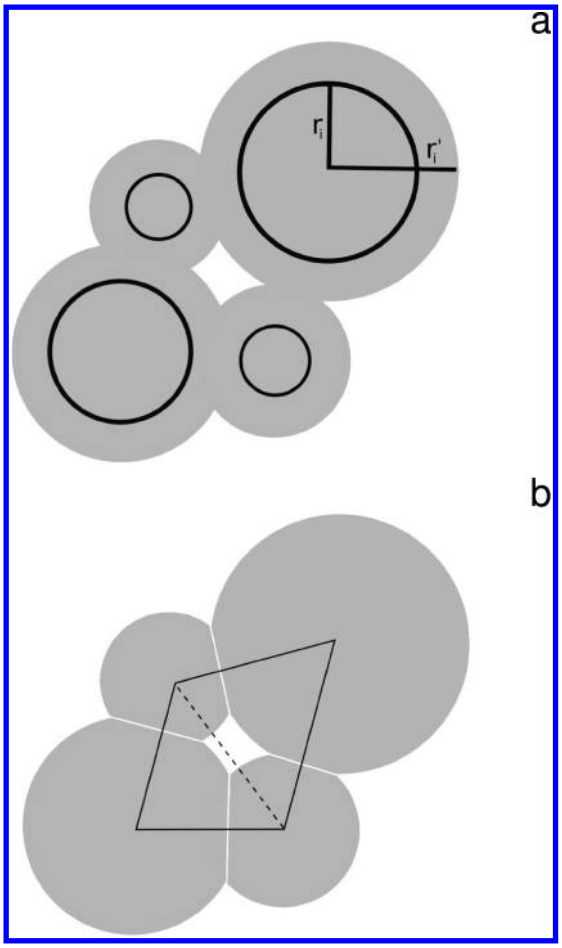



**Figure 3.** Example demonstrating the concept of an $\alpha$-shape, for simplicity drawn in 2D. (a) Four balls with basic radii $r_i$ drawn as solid black spheres. Then the union of modified spheres with $r_i' = (r_i^2 + \alpha)^{1/2}$ are drawn in gray. (b) Regular triangulation of the set of spheres with the black lines indicating its edges. The solid lines belong to the $\alpha$-shape and the dashed edge does not. This line represents the hole in the center of the shape. The lines separating gray fragments of spheres belong to the edges of the power diagram—dual to the regular triangulation.

balls. For each ball in the set the part of its surface that is occluded by other balls is called an *excluded surface*. There is vast literature concerning computation of SASA (Figure 4).[9,14,15,19,20,41−43] Some of the methods use Monte Carlo simulations which are inexact, some use the inclusion−exclusion principle, which may give inaccurate results when some optimizing simplifications are used (optimization is needed as the computational cost is high for the inclusion−exclusion principle).

In comparison, our approach uses the information gathered during construction of the power diagram to define which spheres are in fact intersecting each other. Thus, we can rigorously compute the surface contour necessary in computation of area of the excluded surface for each sphere. The main advantage when compared to Monte Carlo methods is that, instead of approximate value of the SASA, we get exact value up to the arithmetic error of the floating-point data type used in our computer architecture (usually double precision floating point numbers). Also, the expected running time complexity is linear with respect to the input size (the number of spheres) and does not depend on the expected accuracy of the result as it is in the case of approximate Monte Carlo methods. Our algorithm is similar to those proposed by a few authors,[12,14,15] but instead of

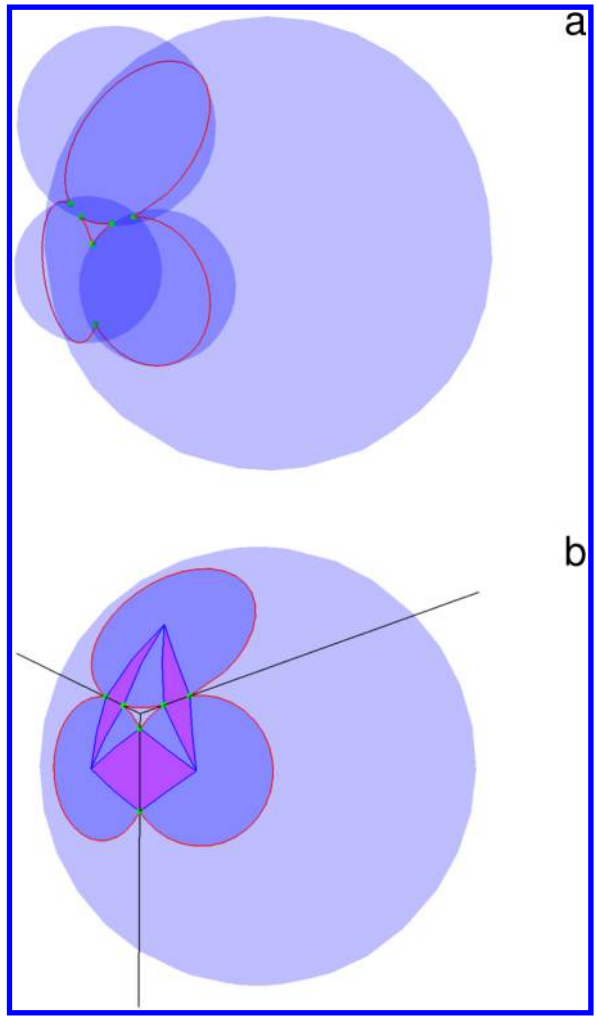



**Figure 4.** Red arcs form a contour which is an example of contact surface between spheres, i.e. the excluded surface (a). This contour can be used to compute SASA for a fragment belonging to the biggest of spheres by subtracting the area of dark-blue and dark-violet regions from the area of the sphere. To compute area of the dark-blue and dark-violet shaded surfaces intersection between three neighboring spheres need to be calculated and the information about connections between these points need to be collected (b). This information can be obtained from the power diagram, as the intersection points lie on the edges of the power diagram cell for that sphere. The area is then computed separately for each fragment using sphere geometry axioms.[44]

working on regular triangulations we use power diagrams directly as an underlying structure.

## RESULTS AND DISCUSSION

**Overview of the DMG-$\alpha$ Library.** *Design and Structure of the Library.* Designing the DMG-$\alpha$ library, we had the following guidelines in mind:

1. The library should be fast to allow analysis of big molecular systems (especially lipid bilayers).
2. The library should handle periodic boundary conditions usually employed in MD simulations of biomolecular systems.
3. The library should postpone computations until it is really needed (for efficiency reasons): For example, we should avoid computation of all data structures for water molecules as those are not always crucial to the analysis.

**Scheme 1. Computing a Power Diagram**

```
from pydmga.geometry import OrthogonalGeometry
from pydmga.container import Container
from pydmga.diagram import Diagram
from pydmga.io import pdb
geometry, container = pdb.load("file.pdb")      # load box coordinates and data
diagram = Diagram(container)                    # apply Voronoi diagram
sum_volume = 0.0
tst_volume = geometry.volume()                  # get the box volume
for cell in diagram:
    print "Cell area:", cell.area()             # print this Voronoi cell area
    print "Cell volume:", cell.volume()         # print this Voronoi cell volume
    sum_volume += cell.volume()                 # sum volume (for simple test)
    for side in cell:                           # browse sides of the Voronoi cell
        print "side area:" side.area()
        print "side neighbour:", side.neighbour_id()
        print "side coords:", side.as_coords()
print "volume test:", sum_volume, "vs", tst_volume  # (should be equal)
```

4. The library should intelligently cache already computed data for future use.
5. The library should be easily scriptable within Python language to make it more user-friendly and to enlarge the group of its potential users.
6. The library should be as minimalistic as possible, yet it should allow for easy access to underlying data structures in a user-friendly and intuitive manner.

For the first four requirements we decided to implement all internal algorithms as a template-based C++ library, both to allow flexible definition of data structures at compilation time and to increase the performance of computations. For the construction of the basic Voronoi diagram we have used Voro++[22] which supports periodic boundary conditions. Other algorithms (projective geometries, $\alpha$-shapes, SASA) were implemented to use Voronoi diagrams as an underlying data structure. We have modified Voro++ with custom iteration procedures which were not available in the library. By the design, Voro++ was implemented as a pipeline tool which outputs a predefined set of quantities associated with the power diagram. In order to use power diagrams produced in Voro++ we needed to expose internal data structures to our algorithms. First of all, we have created a custom iterator for the Voronoi cell data structure and a couple of functions that are allowed to navigate over a doubly connected edge list. Having this, we have wrapped Voro++ classes inside templated C++ DMG-$\alpha$ classes which hide all low-level operation on Voro++ data structures. The class structure of the C++ DMG-$\alpha$ library was designed to match the independently developed Python pydmga interface.

All implemented algorithms take the periodic boundary conditions into consideration by the nearest image convention. All the computations are done "on demand", that is, the data structures are computed only when the corresponding function is invoked. In case of Voronoi diagrams the user can control if the structure should cache the computed Voronoi cells. This is done due to the fact that big input data sets can create Voronoi diagrams that cannot be at once stored in a memory. In such a case the user can turn off caching and analyze structure sequentially, one cell at a time, which requires substantially less memory.

The rest of the requirements were realized in a rich Python (version 2) interface which is an integral part of the library. Python interface is called pydmga to make it distinct from the dmga namespace used in the C++ basic library. The Python interface consists of the following modules:

- pydmga.model, which holds all basic primitives (like points in space, vertexes, edges, and sides of Voronoi cells) along with all iterators needed for easy and intuitive access to underlying data structures.
- pydmga.geometry, which holds geometry objects defining shapes of the master cells of the simulation box. It also contains projective geometries that allow computation of 2D Voronoi diagrams on various surfaces.
- pydmga.container, which hold the Container class for storing positions of the molecules inside a simulation box.
- pydmga.diagram, which provides the Diagram class for computing basic power diagrams.
- pydmga.shape, which holds classes that extend Diagram to various biologically relevant structures. Currently, it provides the SASAShape class for computing SAS Area and AlphaShape class which allows computation of an $\alpha$-shape of a set of molecules.

For the convenience of the user, we also included the two following subpackages:

- pydmga.io, which allows for handling popular file formats used in MD simulations. Currently only PDB files are supported, but we are going to extend the range of supported files in the future releases of the library.
- pydmga.draw, which allows for simple drawing and manipulating objects in 3D. This subpackage requires Panda3d—a freeware third-party software library that can be downloaded for free from the creators webpage www.panda3d.org.

The library follows the design pattern of a decorator: each data structure adds new functionality to the one that was used to create it. For example if we create an instance of the Diagram class using some object of the Container class, then we will have all functions of container accessible via the instance of the class Diagram. Next, if we create SASAShape from Diagram we will have access to all functions for Diagram and also new features that allow for computing SASA. Moreover, the SASACell objects which are created by the iteration over SASAShape are also instances of basic Voronoi Cell objects. This way we can decide how many algorithms are computed in our scripts and thus how much computational cost is needed. If we need only basic information then we use the bare Diagram class. But if we want to extract additional information (about SASA or topological structure) we need to "decorate" the


dx.doi.org/10.1021/ci500273s | J. Chem. Inf. Model. 2014, 54, 3112−3123

**Scheme 2. Projection onto the *XY* Plane**

```
from pydmga.geometry import OrthogonalGeometry, CastOrthoPlaneGeometry
from pydmga.container import Container
from pydmga.diagram import Diagram
from pydmga.io import pdb
geometry = pdb.load_geometry("file.pdb")              # load only box geometry
projXY = CastOrthoPlaneGeometry(geometry, (0,0,1))    # projective geometry along (0,0,1) vector
container = Container(projXY)                         # make container
container.add(pdb.load_atoms("file.pdb"))             # load atom positions
diagram = Diagram(container)                          # apply Voronoi diagram
for i, cell in enumerate(diagram):                    # iterate over all cells
    for side in cell:                  # in OrthoPlaneGeometry only one side is 'on_boundary'
        if projXY.on_boundary(side):   # this side is the one computed on the projective plane XY
            print "atom", i, "SA is", side.area()
```

**Scheme 3. Computing SASA**

```
from pydmga.geometry import OrthogonalGeometry
from pydmga.container import Container
from pydmga.diagram import Diagram
from pydmga.io import pdb
geometry, container = pdb.load("file.pdb")   # load data
sasa_shape = SASAShape(Diagram(container))   # apply Voronoi diagram the SASA Shape
print "SAS area:", sasa_shape.area()         # compute SASA
```

instance of the `Diagram` class with additional algorithms (this usually require extra computation time).

*Sample Codes.* The sample code that presents basic usage of DMG-$\alpha$ to compute a 3D Voronoi diagram of a set of molecules stored in a PDB file is presented in Scheme 1. Computation of a projection of the same set onto the *XY* plane is presented in Scheme 2. An example of the decorator design pattern in action is presented in a script that computes SASA of a molecule in Scheme 3. A graphical roadmap presenting typical applications of the library is presented in Figure 5. The source codes for more extensive examples, namely the scripts used to analyze data in the following section, can be found on the project webpage dmgalpha.scirsc.org and in the Supporting Information.

*Dependencies and Source Codes.* The Python module `pydmga` depends only on standard Python libraries with the exception of the `pydmga.render` graphical module used in visualization of the computed power diagrams which requires the `Panda3D`[32] graphical engine. The `pydmga.render` module is well separated so that the basic `pydmga` package may be used safely without need for `Panda3D` installation.

In order to use the `C++ dmga` library one needs a gcc-compatible compiler with support for `C++0x` (`C++` 11). All other dependencies are incorporated into the source code. In order to use the Python `pydmga` package one needs also Python `distutils` and `python-dev` packages. These packages can be downloaded and installed from the Python homepage (www.python.org) and many Linux-based systems have precompiled packages already available. Detailed installation instructions are provided on the download webpage of the DMG-$\alpha$ library.

The source codes were tested on Ubuntu versions 12.04–13.10 with gcc version 4.6.3 and Python version 2.7.3.

The source code can be found on the project web page dmga.scirsc.org along with online documentation and an extensive set of examples. The library is distributed under a Modified BSD License, which allows for easy integration in an existing code without worry about license problems.

**Application to Lipid Bilayer MD simulations.** To show potential applications of the DMG-$\alpha$ library, we determined several geometric properties of two fully hydrated lipid bilayers in the liquid crystalline phase. The bilayers differed in lipid composition as they were made of two kinds of lipid molecules both occurring in the outer bacterial membrane of *Escherichia coli*. The chemical structures of Lipid-A (ECLA) and deep-rough lipopolysaccharide (DREC) are shown in Figure 6. The ECLA molecule contains six acyl chains (five myrystoil (14 C) and one lauroil (12 C) hydrocarbon tails) and two glucosamine (GlcN) residues forming the Lipid-A backbone. GlcN residues are phosphorylated at positions 1 (the proximal residue) and 4′ (the distal residue). The DREC molecule additionally has two 3-deoxy-D-manno-oct-2-ulosonic acid (Kdo) residues, which are attached to each other and joined to the distal GlcN residue. For the purpose of the analysis of structural features of membranes made by these two different lipid molecules it is worth noticing that the DREC molecule can be considered as the extended version of the ECLA molecule since the DREC lipid headgroup is made of two GlcN and two KDO residues while the ECLA lipid headgroup contains only two GlcN residues.

The ECLA and DREC molecules were assigned OPLS/AA force field parameters[45,46] and water molecules TIP3P parameters.[47] The negative charge of ECLA ($-2e$) and DREC ($-4e$) molecules was counterbalanced by sodium cations.[48] The total number of atoms in the ECLA and DREC model systems was 40 248 and 46 728, respectively. Submicrosecond MD simulations were carried out in Gromacs v4.5 package.[49] The further details concerning MD simulation protocol can be found elsewhere.[46]

In our calculations we used 11 consecutive frames, sampled every one nanosecond, from fully developed MD trajectories. Relatively simple `pydmga` scripts (see the project web page dmga.scirsc.org) were run to calculate $V_m$, SASA, and SA. $V_m$ was calculated as the sum of volumes of Voronoi polyhedrons assigned to every heavy atom in a lipid molecule. SASA was calculated as the sum of areas of the Voronoi facets shared by two neighboring cells, i.e. the case where one cell was occupied by water oxygen atom and the other by lipid atom (hydrogen atoms present in water and lipid molecules were omitted, and they were implicitely present in Voronoi cells assigned to appropriate heavy

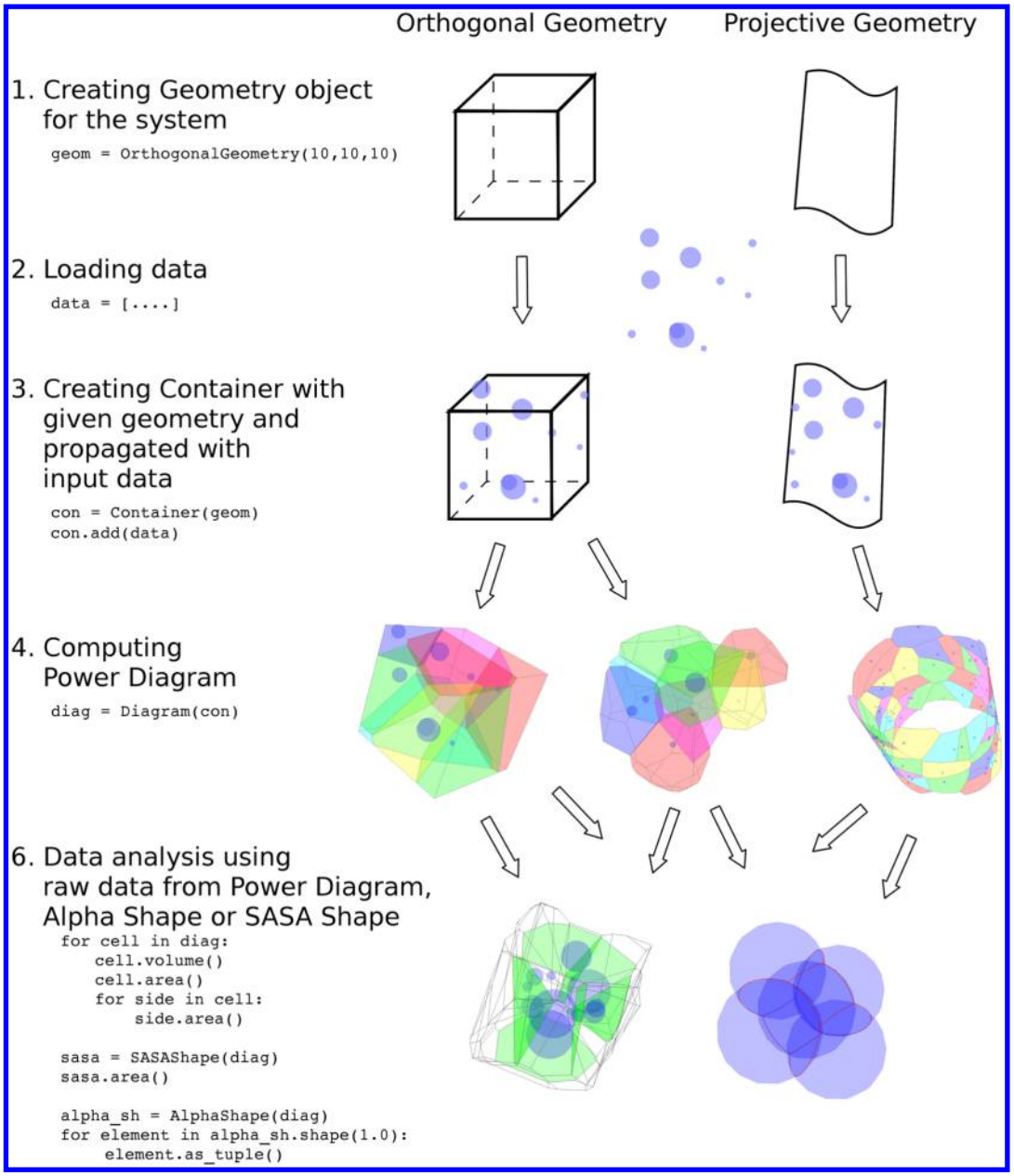



**Figure 5.** Roadmap presenting typical applications of the DMG-$\alpha$ library.

atoms). SA was calculated according to two alternative criteria defining how an individual lipid molecule was represented. In our first approach, a geometric center of two GlcN residues (the common part of ECLA and DREC lipid headgroups) was used (Figure 7a), while in the second approach all heavy atoms of both GlcN residues of ECLA and DREC were projected on the 2D plane defined by bilayer surface, and 2D Voronoi diagram of this projection was computed (Figure 7b). Other `pydmga` scripts were used to determine distributions of sodium ions and water molecules in contact with membrane lipids. In that case, we counted Voronoi polyhedrons assigned to, respectively, ions and water oxygen atom, which have at least one shared facet with any heavy lipid atom.

As can be seen from Figure 8a, $V_m$ of ECLA molecules is significantly smaller than in case of DREC molecules, which is the expected result in view of the apparent differences in size and chemical structure of headgroups of both lipid molecules (Figure 6). The average values of $V_m$ for ECLA and DREC are, respectively, 2.86 and 3.27 nm$^3$. A similar pattern (Figure 8b) is seen when comparing SASA. The DREC molecule with the larger headgroup and the larger number of atoms able to form hydrogen bonds with water has the larger SASA than the ECLA molecule. The average values of SASA for ECLA and DREC are, respectively, 4.15 and 6.82 nm$^2$. Interestingly, distribution of SA for ECLA and DREC bilayers (Figure 8c and d) are very similar to the average values, respectively, 1.50 and 1.55 nm$^2$. This clearly indicates that the main difference between these two bilayers, both in liquid crystalline phase, is not in SA but in the width of lipid/water interphase.

In the view of differences in $V_m$ for ECLA and DREC molecules, it is interesting to analyze in more detail contribution of various parts of these lipid molecules into the total value of $V_m$. The volumes of the hydrophobic parts of ECLA and DREC molecules are almost the same in the studied bilayers and equal to 1.97 and 1.96 nm$^3$, respectively. To make comparison of volumes of the lipid headgroup feasible, we divided molecules


dx.doi.org/10.1021/ci500273s | J. Chem. Inf. Model. 2014, 54, 3112−3123

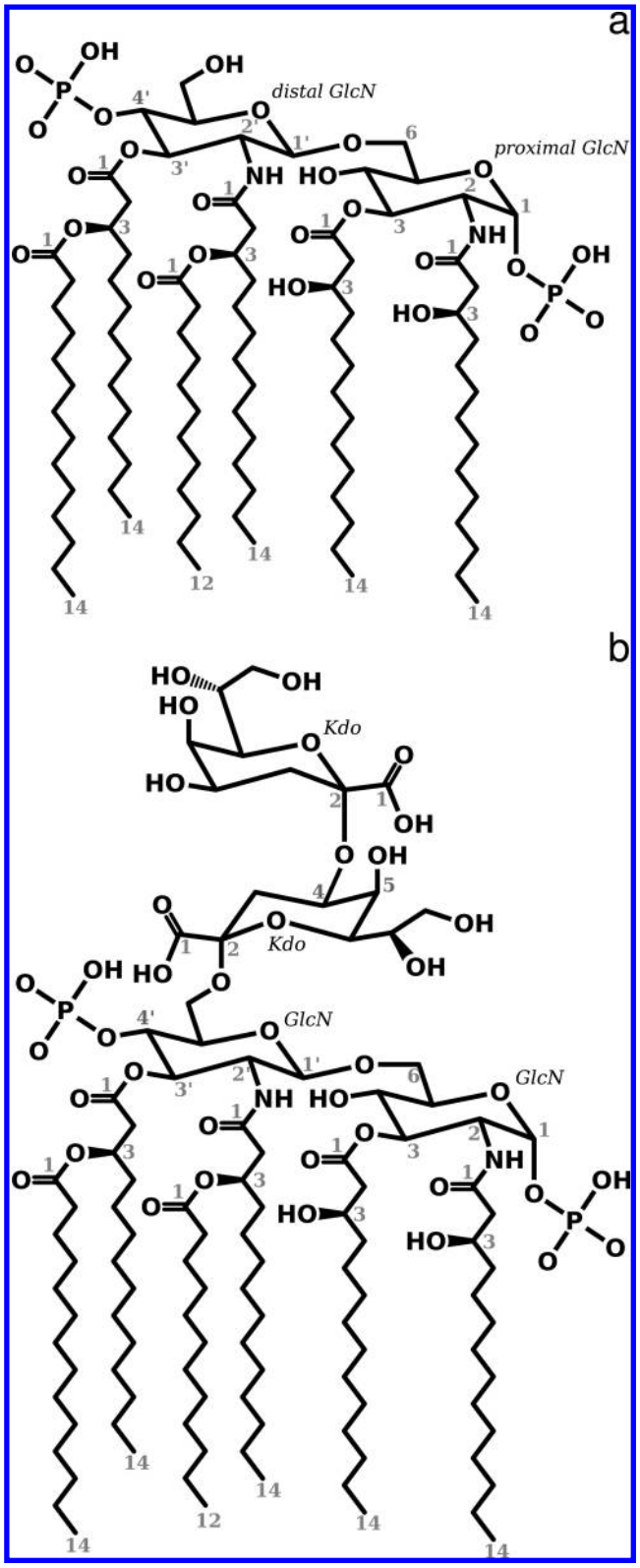



**Figure 6.** Chemical structure of (a) *Escherichia coli* Lipid-A (ECLA) and (b) *Escherichia coli* deep-rough lipopolysaccharide (DREC). Two glucosamine (GlcN) residues present in both lipid molecules and two 3-deoxy-D-manno-oct-2-ulosonic acid (Kdo) present only in DREC molecule together with polar parts of attached acyl chains form the lipid headgroup. Four (*R*)-3-hydroxymyristoyl chains (HMy, 14:0(3-OH)) attached to the GlcN residues at positions 2, 3, 2′, and 3′ together with laurate and myristate acylating HMy attached to the distal GlcN form the hydrophobic part of the lipid molecules.

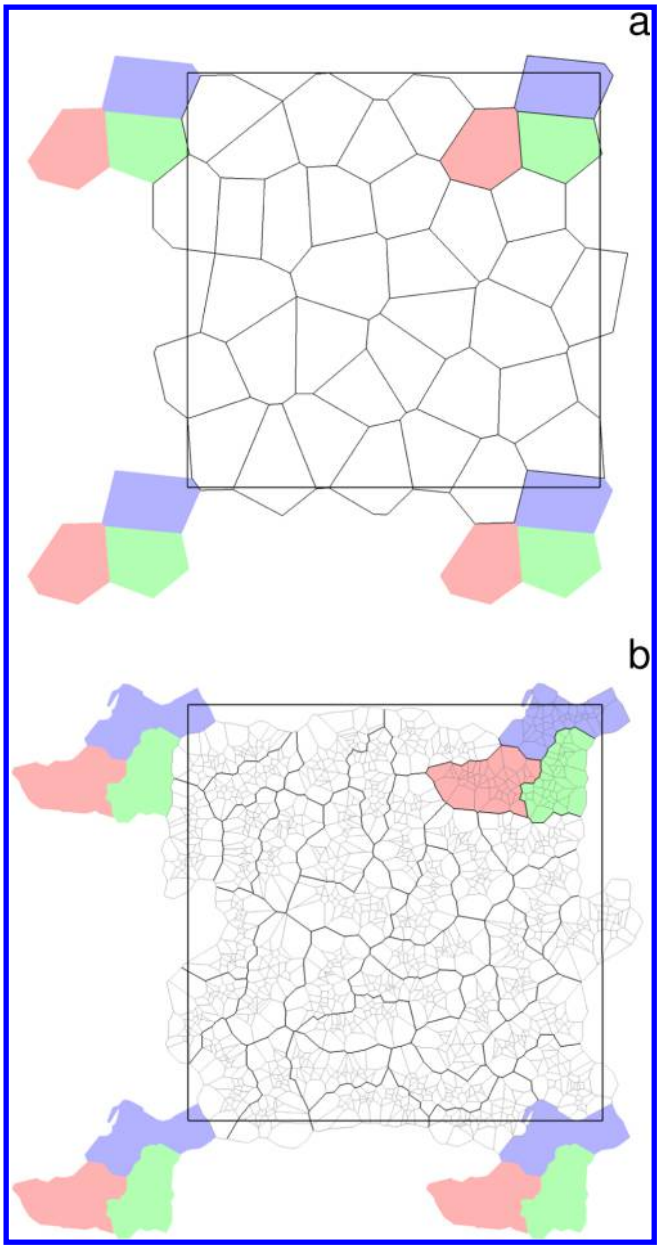



**Figure 7.** Sample projection of the upper leaflet of the DREC lipid bilayer system onto the plane orthogonal to the normal of the membrane. Comparison of two alternative criteria used in calculation of SA: GlcN geometric centers (a) vs all GlcN heavy atoms (b). Simulation box boundaries and Voronoi cell edges are shown. In part b, clusters of Voronoi cells belonging to individual lipid molecules are outlined. For further details see the main text.

into two (ECLA) or three (DREC) distinct fragments. The first fragment is made by polar parts of acyl chains attached to the glucosamine backbone of ECLA and DREC molecules. The second fragment is glucosamine disaccharide phosphorylated at positions 1 and 4′ (Figure 6). And finally, the third fragment present only in the DREC molecule contains two Kdo residues. $V_m$ of the first fragment of ECLA and DREC equals 0.423 and 0.420 nm$^3$, respectivly. In case of the second fragment, we obtained 0.466 and 0.451 nm$^3$. $V_m$ of two Kdo residues present in DREC molecule is 0.445 nm$^3$. As can be seen, molecular volumes of the two fragments of lipid headroup present in ECLA and DREC are almost identical; each of the three parts has approximately the same volume, and the total volume of the


dx.doi.org/10.1021/ci500273s | J. Chem. Inf. Model. 2014, 54, 3112−3123

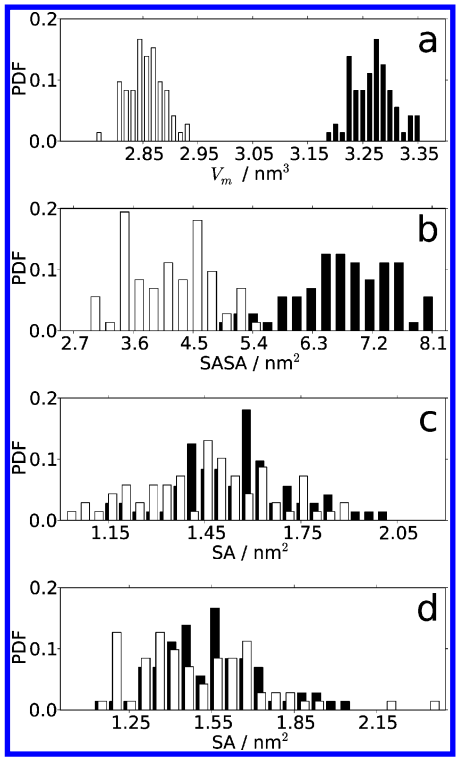



**Figure 8.** Distribution of (a) $V_m$, (b) SASA, and SA calculated for geometric center (c) and all heavy atoms (d) of GlcN residues in ECLA and DREC molecules. See the main text for details.

ECLA molecule (0.89 nm$^3$) is roughly 68% of the value characterizing the DREC molecule (1.32 nm$^3$). The reported average values have standard deviations 3.3% or less.

Other differences in the structure of lipid/water interphase of ECLA and DREC bilayers are very well seen in comparison of the number of water and ions interacting directly with membrane lipids (Figure 9). ECLA and DREC molecules make contacts on average with 2.35 and 6.43 ions, respectively. Distribution of water molecules forming the first hydration layer of ECLA and DREC bilayers is also distinctly different with the average water molecules per lipid equal to 47.3 and 68.8, respectively. It should

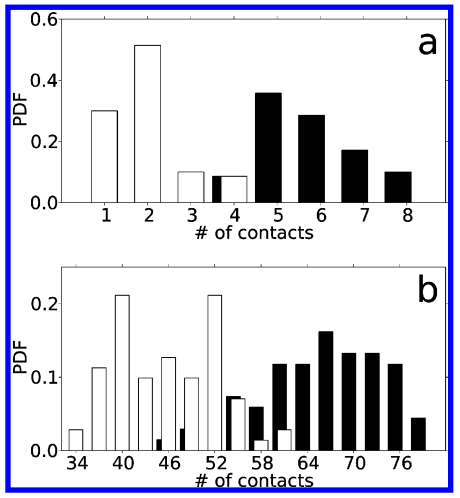



**Figure 9.** Histogram of the number of $Na^+$ ions (a) and water molecules (b) in direct contact with a lipid molecule. White and black bars correspond, respectively, to the ECLA and DREC bilayers.

be noted that the average values of parameters mentioned above give just a general overview of lipid/water interphase of ECLA and DREC bilayers. An in-depth analysis of these interphases will be published elsewhere.

**Performance.** The most numerically demanding part of computations in the DMG-$\alpha$ library is the creation of the power diagram. Currently there are no known theoretically optimal algorithms for computing power diagrams in 3D; however, there are some known upper and lower bounds on the complexity of the problem.[50] There are also some strategies and techniques that can boost the performance in special cases. As we are interested in molecular dynamics simulations of multimolecular systems, especially biological membrane simulations, we may assume that the input to the procedure is a densely packed, uniformly distributed collection of spheres. Thus, we can use algorithms that take this assumption into consideration to expect linear size of the resulting structure.[51,52] One interesting technique that is useful for large and single-component molecular systems relies on biological databases to identify bonds between atoms and subsequently to determine which atoms are likely to be true neighbors in a power diagram. This heuristic was successfully implemented in the `PowerSASA` program, which despite its theoretical complexity being $O(|V|^2)$, where $|V|$ is the size of the power diagram $V$ (i.e., sum of the number of edges, sides, and vertices), exhibits linear dependence on $n$, i.e. the total number of atoms in the master cell. Another approach, used for example in `Voro++`, is the localization of the data structure. This technique heuristically search for nearest neighbors in a master cell divided into a finite grid of smaller cells and usually it allows to reduce expected complexity to linear when the number of neighbors per cell is constant. This approach is very useful in the case of analysis of MD simulation trajectories where we do not have other a priori knowledge of the neighborhood of the atoms, but we assume only a roughly uniform distribution of atoms in space.

We have performed a numerical study on several randomly generated test cases and on two data sets (hydrated lipid bilayers) discussed in this section. Computations in these real case scenarios were done in two ways. We used either all atoms in the molecular system or only heavy atoms (i.e., we discarded hydrogen atoms). For each data set the power diagram was computed and basic statistics was accumulated (SA, $V_m$, list of neighbors). The computational times were plotted against the size of the data in Figure 10. We can see a nice linear scaling with the increasing data size. Analogous parallel calculations run on 2 and 4 cores of an Intel i7 processor scale well too.

Other algorithms ($\alpha$-shapes and SASA computation) implemented in the DMG-$\alpha$ library have theoretical complexity $\Theta(|V|)$, because, in principal, they require visiting each element of the power diagram only once.

## ■ CONCLUSIONS

We have created a computational geometry package for computing weighted Voronoi diagrams, $\alpha$-shapes, and solvent accessible surfaces with fine-grain access to underlying structures that allow for extensional scripting. The package is an open-sourced `C++` template-based library and contains a rich built-in Python interface. Source code with documentation and numerous examples may be downloaded from the project webpage dmga.scirsc.org.

For core computations of cells of the power diagram, we used `Voro++` open source routines, that is, to our knowledge, the only freely available library that supports power diagrams in 3D

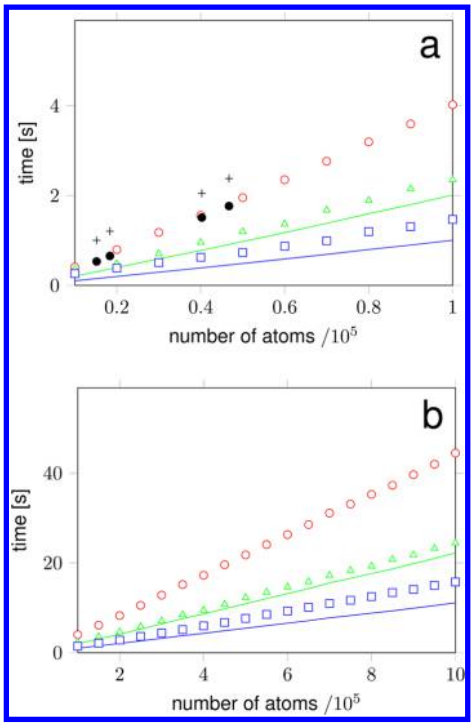



**Figure 10.** Computational times vs increasing size of the input data. The data sets were randomly generated in two intervals, first from $10^4$ to $10^5$ with step $10^4$ (a), second from $10^5$ to $10^6$ with step $5 \times 10^4$ (b). The data sets were generated sequentially, then the computations of power diagram were done with 1, 2, and 4 processes using a Python multithreading package. Red circles, green triangles, and blue squares denote respectively computations with 1, 2, and 4 cores. Green and blue solid lines denote the best possible boost for 2 and 4 cores as compared to only one process. Black cross marks correspond to computational time for ECLA and DREC bilayers derived as a mean of running times for 11 data frames, computed with and without hydrogen atoms, on a single core. The points are significantly above the red circles as the time needed to read input data from a PDB file is included. Solid black circles present running times after subtracting the time needed to read the input data.

with periodic boundary conditions. The `Voro++` library is written entirely in `C++` and is designed mainly as an pipeline tool that takes input files in a given format and produces other output files with constrained format. To allow for better handling of the underlying data structures we developed an API to access programmatically the information stored in Voronoi cells during computations. We also added support for generic data structures to be used for defining input data sets. Having this fine-grained access to power diagrams we developed algorithms to efficiently compute geometric properties like volume of cells, area per side of the cell, neighboring information, $\alpha$-shape description of the cells, and SASA calculation and computing projections to arbitrary 2D surfaces. SASA is calculated exactly using spherical geometry computations. Projections to 2D surfaces are realized as an intersection of 3D cells with a triangulated surface. Currently, projections to planes orthogonal to the master cell and projections to spheres and cylinders are directly supported, but more general cases may be created using built-in `C++` APIs for projective geometries.

All calculations are made on demand, that is, no item is computed when not necessary. This allows for selective and/or incremental computation of only some parts of the diagrams and other data structures ($\alpha$-shape, SASA), which boost performance when only a few elements need to be computed. For example, incremental computing based on graph theory were used successfully by the authors to construct procedure for efficient identification of solvent molecules trapped inside lipid bilayers.[53] Construction of a complete power diagram runs in expected time $O(n)$ where $n$ is the size of the input data, and other algorithms ($\alpha$-shapes, SASA, 2D projection) have optimal complexity of $O(|V|)$ where $|V|$ is size of the underlying power diagram. All computations are easily parallelizable through modules of the Python standard library.

Future plans include creation of routines for fine-grained topology discovery algorithms like lipid membrane pore and/or channel detection, implementing more projective geometries, and automatic handling of whole trajectories using the Kinetic Data Structures (KDS) framework.

## ■ ASSOCIATED CONTENT

### Supporting Information

Source code for the scripts used to do the analysis of lipid bilayers presented in this article. This material is available free of charge via the Internet at http://pubs.acs.org.

## ■ AUTHOR INFORMATION

### Corresponding Author

*E-mail: robert.szczelina@uj.edu.pl.

### Notes

The authors declare no competing financial interest.

## ■ ACKNOWLEDGMENTS

R.S. and K.M. acknowledge financial support from the Polish National Science Centre under grant nos. 2011/01/N/ST6/07173 and 2011/01/B/NZ1/00081, respectively.

## ■ REFERENCES

(1) Murzyn, K.; Pasenkiewicz-Gierula, M. Construction of a toroidal model for the magaininpore. *J. Mol. Model.* **2003**, *9*, 217−224.

(2) Gapsys, V.; Groot, B. L.; Briones, R. Computational analysis of local membrane properties. *J. Comput.-Aided Mol. Des.* **2013**, *27*, 845−858.

(3) Nagle, J. F.; Tristram-Nagle, S. Structure of lipid bilayers. *Biochim. Biophys. Acta, Rev. Biomembr.* **2000**, *1469*, 159−195.

(4) Braun, A.; Brandt, E.; Edholm, O.; Nagle, J.; Sachs, J. Determination of Electron Density Profiles and Area from Simulations of Undulating Membranes. *Biophys. J.* **2011**, *100*, 2112−2120.

(5) Gerstein, M.; Chothia, C. Packing at the protein-water interface. *Proc. Natl. Acad. Sci. U.S.A.* **1996**, *93*, 10167−10172.

(6) Murzyn, K.; Rog, T.; Blicharski, W.; Dutka, M.; Pyka, J.; Szytula, S.; Froncisz, W. Influence of the disulfide bond configuration on the dynamics of the spin label attached to cytochrome C. *Proteins* **2006**, *62*, 1088−1100.

(7) Sledz, P.; Zheng, H.; Murzyn, K.; Chruszcz, M.; Zimmerman, M.; Chordia, M.; Joachimiak, A.; Minor, W. New surface contacts formed upon reductive lysine methylation: Improving the probability of protein crystallization. *Protein Sci.* **2010**, *19*, 1395−1404.

(8) Porebski, P.; Klimecka, M.; Chruszcz, M.; Nicholls, R.; Murzyn, K.; Cuff, M.; Xu, X.; Cymborowski, M.; Murshudov, G.; Savchenko, A.; Edwards, A.; Minor, W. Structural characterization of Helicobacter pylori dethiobiotin synthetase reveals differences between family members. *FEBS J.* **2012**, *279*, 1093−1105.

(9) Esque, J.; Oguey, C.; de Brevern, A. G. Comparative Analysis of Threshold and Tessellation Methods for Determining Protein Contacts. *J. Chem. Inf. Model.* **2011**, *51*, 493−507.

(10) Esque, J.; Oguey, C.; de Brevern, A. G. A Novel Evaluation of Residue and Protein Volumes by Means of Laguerre Tessellation. *J. Chem. Inf. Model.* **2010**, *50*, 947−960 PMID: 20392096..

(11) Mach, P.; Koehl, P. Geometric measures of large biomolecules: Surface, volume, and pockets. *J. Comput. Chem.* **2011**, *32*, 3023−3038.
(12) Liang, J.; Edelsbrunner, H.; Fu, P.; Sudhakar, P. V.; Subramaniam, S. Analytical shape computation of macromolecules: I. molecular area and volume through alpha shape. *Proteins: Struct., Funct., Bioinf.* **1998**, *33*, 1−17.
(13) Poupon, A. Voronoi and Voronoi-related tessellations in studies of protein structure and interaction. *Curr. Opin. Struct. Biol.* **2004**, *14*, 233−241.
(14) Mach, P.; Koehl, P. An analytical method for computing atomic contact areas in biomolecules. *J. Comput. Chem.* **2013**, *34*, 105−120.
(15) Klenin, K. V.; Tristram, F.; Strunk, T.; Wenzel, W. Derivatives of molecular surface area and volume: Simple and exact analytical formulas. *J. Comput. Chem.* **2011**, *32*, 2647−2653.
(16) *The PyMOL Molecular Graphics System*, Version 1.3r1; Schrödinger, LLC: Portland, OR, 2010.
(17) Lukat, G.; Krüger, J.; Sommer, B. APL@Voro: A Voronoi-Based Membrane Analysis Tool for GROMACS Trajectories. *J. Chem. Inf. Model.* **2013**, *53*, 2908−2925.
(18) Hoff, K. E., III; Keyser, J.; Lin, M.; Manocha, D.; Culver, T. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. *26th International Conference on Computer Graphics and Interactive Techniques*, Los Angeles, Aug 8−13, 1999, DOI: 10.1145/311535.311567.
(19) Till, M.; Ullmann, G. McVol - A program for calculating protein volumes and identifying cavities by a Monte Carlo algorithm. *J. Mol. Model.* **2010**, *16*, 419−429.
(20) Brezovsky, J.; Chovancova, E.; Gora, A.; Pavelka, A.; Biedermannova, L.; Damborsky, J. Software tools for identification, visualization and analysis of protein tunnels and channels. *Biotechnol. Adv.* **2013**, *31*, 38−49 Prague Symposium 2011..
(21) Boissonnat, J.-D.; Devillers, O.; Pion, S.; Teillaud, M.; Yvinec, M. Triangulations in {CGAL}. *Computational Geometry* **2002**, *22*, 5−19.
(22) Rycroft, C. H. VORO++: A three-dimensional Voronoi cell library in C++. *Chaos* **2009**, *19*, 041111.
(23) Barber, C. B.; Dobkin, D. P.; Huhdanpaa, H. The Quickhull Algorithm for Convex Hulls. *ACM Trans. Math. Softw.* **1996**, *22*, 469−483.
(24) Cignoni, P.; Montani, C.; Scopigno, R. DeWall: A fast divide and conquer Delaunay triangulation algorithm in $E^d$. *Comput. Aided Des.* **1998**, *30*, 333−341.
(25) Olechnovič, K.; Margelevičius, M.; Venclovas, Č. Voroprot: an interactive tool for the analysis and visualization of complex geometric features of protein structure. *Bioinformatics* **2011**, *27*, 723−724.
(26) Amenta, N.; Choi, S.; Rote, G. Incremental Constructions Con BRIO. *Proceedngs of the 19th Annual Symposium on Computational Geometry*, San Diego, June 8−10, 2003, DOI:10.1145/777792.777824.
(27) Oliphant, T. E. Python for Scientific Computing. *Comput. Sci. Eng.* **2007**, *9*, 10−20.
(28) Hinsen, K. The molecular modeling toolkit: A new approach to molecular simulations. *J. Comput. Chem.* **2000**, *21*, 79−85.
(29) Jones, E.; Oliphant, T.; Peterson, P. *SciPy: Open source scientific tools for Python*. http://www.scipy.org/ (accessed October 16, 2014).
(30) Petřek, M.; Košinová, P.; Koča, J.; Otyepka, M. MOLE: A Voronoi Diagram-Based Explorer of Molecular Channels, Pores, and Tunnels. *Structure* **2007**, *15*, 1357−1363.
(31) Chovancova, E.; Pavelka, A.; Benes, P.; Strnad, O.; Brezovsky, J.; Kozlikova, B.; Gora, A.; Sustr, V.; Klvana, M.; Medek, P.; Biedermannova, L.; Sochor, J.; Damborsky, J. CAVER 3.0: A Tool for the Analysis of Transport Pathways in Dynamic Protein Structures. *PLoS Comput. Biol.* **2012**, *8*, e1002708.
(32) Goslin, M.; Mine, M. The Panda3D graphics engine. *Computer* **2004**, *37*, 112−114.
(33) Aurenhammer, F. Voronoi Diagrams—a Survey of a Fundamental Geometric Data Structure. *ACM Comput. Surv.* **1991**, *23*, 345−405.
(34) Edelsbrunner, H. *Weighted Alpha Shapes*; 1992; https://www.cs.duke.edu/~edels/Papers/1992-R-08-WeightedAlphaShapes.pdf (accessed on October 16, 2014).
(35) Edelsbrunner, H.; Mücke, E. P. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Trans. Graph* **1990**, *9*, 66−104.
(36) Rycroft, C. H. *Testing the speed of the code*. http://math.lbl.gov/voro++/examples/timing_test (accessed on October 16, 2014).
(37) Edelsbrunner, H.; Kirkpatrick, D.; Seidel, R. On the shape of a set of points in the plane. *IEEE Trans. Inf. Theory* **1983**, *29*, 551−559.
(38) Edelsbrunner, H. *Shape Reconstruction with Delaunay Complex*. http://dblp.uni-trier.de/db/conf/latin/latin98.html#Edelsbrunner98 (accessed October 16, 2014).
(39) Edelsbrunner, H.; Mücke, E. P. Three-dimensional Alpha Shapes. *ACM Trans. Graph.* **1994**, *13*, 43−72.
(40) Edelsbrunner, H.; Harer, J. *Computational Topology: An Introduction*; Applied mathematics; American Mathematical Society, 2010.
(41) Lee, B.; Richards, F. The interpretation of protein structures: Estimation of static accessibility. *J. Mol. Biol.* **1971**, *55*, 379−IN4.
(42) Wang, H.; Levinthal, C. A vectorized algorithm for calculating the accessible surface-area of macromolecules. *J. Comput. Chem.* **1991**, *12*, 868−871.
(43) Legrand, S.; Merz, K. Rapid approximation to molecular-surface area via the use of boolean logic and look-up tables. *J. Comput. Chem.* **1993**, *14*, 349−352.
(44) Meserve, B. *Fundamental Concepts of Geometry*; Addison-Wesley mathematics series; Dover Publications, 1983.
(45) Price, M. L. P.; Ostrovsky, D.; Jorgensen, W. L. Gas-Phase and Liquid-State Properties of Esters, Nitriles, and Nitro Compounds with the OPLS-AA Force Field. *J. Comput. Chem.* **2001**, *22*, 1340−1352.
(46) Murzyn, K.; Bratek, M.; Pasenkiewicz-Gierula, M. Refined OPLS All-Atom Force Field Parameters for n-Pentadecane, Methyl Acetate, and Dimethyl Phosphate. *J. Phys. Chem. B* **2013**, *117*, 16388−16396.
(47) Jorgensen, W. L.; Chandrasekhar, J.; Madura, J. D.; Impey, R. W.; Klein, M. L. Comparison of simple potential functions for simulating liquid water. *J. Chem. Phys.* **1983**, *79*, 926−935.
(48) Aqvist, J. Ion-Water Interaction Potentials Derived from Free Energy Perturbation Simulations. *J. Phys. Chem.* **1990**, *94*, 8021−8024.
(49) van der Spoel, D.; Lindahl, E.; Hess, B.; Groenhof, G.; Mark, A. E.; Berendsen, H. J. GROMACS: fast, flexible, and free. *J. Comput. Chem.* **2005**, *26*, 1701−1718.
(50) Aronov, B. A Lower Bound on Voronoi Diagram Complexity. *Inf. Process. Lett.* **2002**, *83*, 183−185.
(51) Dwyer, R. Higher-dimensional voronoi diagrams in linear expected time. *Discrete Comput. Geom.* **1991**, *6*, 343−367.
(52) Dwyer, R. Higher-dimensional voronoi diagrams in linear expected time. *Discrete Comput. Geom.* **1991**, *6*, 343−367.
(53) Szczelina, R.; Murzyn, K. *Lipid bilayer analysis using 3D Voronoi diagrams*. http://dmgalpha.scirsc.org/?p=poster-bit13 (accessed October 16, 2014).