Dynamic Maintenance of Molecular Surfaces under Conformational Changes*

Eran Eyal Dan Halperin

School of Computer Science
Tel Aviv University, Israel
{eyaleran,danha}@tau.ac.il

ABSTRACT

We present an efficient algorithm for maintaining the boundary and surface area of protein molecules as they undergo conformational changes. We also describe a robust implementation of the algorithm and report on experimental results with our implementation on proteins with hundreds of residues. Our work extends and combines two previous results: (i) controlled perturbation for static molecular surfaces [18], and (ii) data structures for self-collision testing and energy maintenance of proteins that change conformation [26]. As our method keeps a highly accurate representation of the boundary surface and of the voids in the molecule, it can be useful in various applications such as Monte Carlo Simulation or Molecular Dynamics Simulation. In addition we propose and analyze an alternative method for efficiently recalculating the surface area under conformational (and hence topological) changes based on techniques for efficient dynamic maintenance of graph connectivity; initial results of the implementation of this method show great promise.

Categories and Subject Descriptors

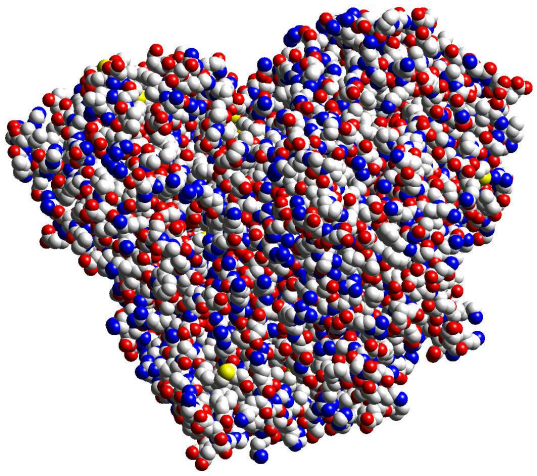
F.2.2 [**Theory of Computation**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*; J.3 [**Computer Applications**]: Life And Medical Sciences—*Biology and genetics*

General Terms

Algorithms, Experimentation

*Work reported in this paper has been supported in part by the IST Programme of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

SCG'05, June 6–8, 2005, Pisa, Italy.



Keywords

Molecular Surfaces, Robust Geometric Computing, Controlled Perturbation, Molecular Simulations, Dynamic Data Structures

1. INTRODUCTION

1.1 Background

A common approach to modeling the three-dimensional geometric structure of molecules is to represent each atom as a sphere of fixed radius in a fixed placement relative to the other atoms. The radius assigned to each atom depends on the type of the atom. There are various sets of recommended values for atom radii, depending on the specific application. The spheres are allowed to penetrate one another. This model, called the *hard sphere model*, has proven useful in many practical applications, in spite of its approximate nature.

Molecular surfaces have many uses, such as drug design, studies of solvation and hydrophobicity, the protein folding problem, and more. One type of molecular surfaces is simply the outer boundary of the union of the spheres in the hard sphere model. This type uses the van der Waals radii, and is often referred to as the van der Waals surface. There are two closely related types of surfaces: The *solvent accessible surface* introduced by Lee and Richards [23] is defined by the center of a solvent molecule, modeled as a hard probe sphere, when it rolls over the van der Waals surface. The

smooth molecular (solvent excluded) surface introduced by Richards [29] is defined by the part of the surface of the solvent probe-sphere that faces the molecule. See also [8, 9, 11] and the survey by Mezey [27] for an extensive discussion on molecular surfaces.

The study of the conformations adopted by proteins is an important topic in structural molecular biology. Some of the methodologies used for this study are Monte Carlo Simulation (MCS) [5, 19] and Molecular Dynamics Simulation (MDS) [2, 22]. In MCS we randomly generate a series of trial steps in the conformation space of the studied molecule. Each such step consists of changing some degrees of freedom (DOFs) of the molecule, in general torsion (dihedral) angles around bonds. Classically, a trial step is accepted with a probability that depends on the difference in energy between the new and previous conformations, moving toward the lower energy conformation. In MDS we compute the forces on the atoms at each step, and use them to calculate the atom positions at the next step. Most MDS techniques update the Cartesian coordinates of the atoms at each step, but recently there has been growing interest in directly updating torsion angles [32].

In the context of molecular simulations, the surface area of a molecule is required when calculating the energy of the molecule. To avoid using explicit solvent (actual water molecules) in such simulations, which considerably slows down the computations, implicit solvent models were introduced. In these models all solvent effects on a molecule can be included in an effective potential, which includes a term for non-polar contributions. This term is often modeled as a weighted sum of the solvent exposed or accessible surface area of each atom of the molecule (see [6] for a discussion and more references). Therefore fast methods to maintain the surface area of a molecule dynamically during conformation changes are desired.

1.2 Related Work

Several algorithms and their software implementation for calculation of the various surfaces mentioned above have been designed in the last two decades. The smooth molecular surface was first computed by Connolly [8] and later in many other works [11, 31, 33, 34]. Simpler methods computed approximations of the surfaces [24]. Halperin and Shelton [18] used *controlled perturbation* to calculate the van der Waals and the solvent accessible surfaces robustly.

Limited dynamic maintenance of molecular surfaces was presented by Bajaj *et al* [3]. They use non-uniform rational B-splines (NURBS) to represent molecular surfaces and dynamically maintain them as the radius of the solvent probe-atom changes continuously.

Edelsbrunner *et al* [7] developed an algorithm for maintaining an approximating triangulation of a deforming surface in $\mathbb{R}^3$, that adapts dynamically to changing shape, curvature, and topology of the surface. Bryant *et al* [6] calculate the area derivatives of molecular surfaces in motion, for a molecular dynamics simulation. At each step of the simulation they re-compute the entire Delaunay triangulation required for their calculations.

Sanner and Olson [30] presented surface reconstruction for moving molecular fragments. In their work they achieve a real-time reconstruction of the molecular surface when a small number of atoms move in each step (for example, a conformation change of a single side chain of the protein). The complexity of their algorithm is $O(t \log t)$, where t is at least as high as the number of moving atoms. This means that a change in a single torsion angle located near the center of a protein chain, which moves the location of about half the atoms of the molecule, will take as much time asymptotically as it takes to recompute the entire surface.

Lotan *et al* [26] introduced a fast implementation of MCS of proteins where a large number of atoms move in each step. They exploit the fact that proteins are long kinematic chains.

1.3 Our Results

In our work we maintain the boundary and surface area of proteins as they undergo conformational changes. We exploit the fact that proteins are long kinematic chains (and not an arbitrary collection of spheres). As the conformations change, we update the torsion angles of the protein backbone, instead of updating the Cartesian coordinates of the atoms. This allows us to modify the boundary of the molecule quickly even when a large number of atoms move, as is usually the case in conformation changes of proteins. The update time of the boundary depends on the number of intersecting pairs of atom spheres whose intersection circle changed, which is relatively small when just a few torsion angles are changed in each step of the simulation. Maintaining a highly accurate[1] representation of the boundary surface and of the voids of the molecule allows us to keep track of the surface area of the molecule and the contribution of each atom to the boundary and to the voids, which can be useful in various applications such as MCS and MDS. Our use of controlled perturbation ensures the robustness of our implementation even while using floating-point arithmetic.

In our best experimental results we managed to update a molecular surface under conformational changes in 3% of the total time it would take to construct that surface from scratch. Our results indicate that our algorithm gives better gains for larger molecules.

We also suggest an alternative method to improve our implementation, based on efficient maintenance of graph connectivity, which yields an amortized update time of $O(p \log^2 n)$ for each accepted conformational change where n is the total number of atoms in the molecule and p is the number of atom spheres whose intersection pattern with the other atom spheres was affected by a conformational change. The number p is typically much smaller than the number of moving atoms. We are currently implementing this method.

1.4 Paper Outline

The rest of the paper is organized as follows. In the next section we present the terminology that will be used throughout the paper. In Section 3 we describe our method of dynamic maintenance of molecular surfaces under conformation change. In Section 4 we describe our method for avoiding degeneracies in the implementation, using controlled perturbation. Section 5 deals with some implementation details. Section 6 gives highlights of our experimental results. Suggestions for future work are given in Section 7.

[1] We use the description *highly accurate* rather than *exact* to avoid confusion with exact geometric computing, since we are using floating point arithmetic.

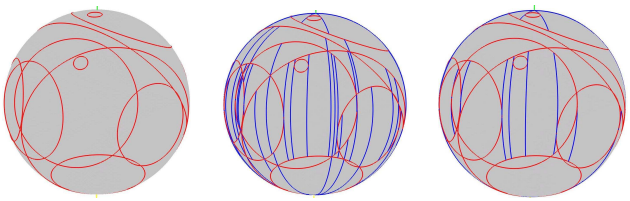


Figure 1: A spherical arrangement (left), its full trapezoidal decomposition (center) and its partial trapezoidal decomposition (right).

2. TERMINOLOGY

An arrangement of spheres — the subdivision of $\mathbb{R}^3$ into vertices, arcs, faces and three-dimensional cells induced by a given finite collection of spheres. (Arrangements of curves and surfaces have been intensively studied and are widely used in Computational Geometry [1, 14].)
Given a set of spheres $S = \{s_1, s_2, ..., s_n\}$, the **spherical arrangement** $\mathcal{A}(C)$ is the subdivision of a sphere s_i induced by the collection of circles $C = \{s_i \cap s_j \mid s_j \in S, j \neq i\}$ which are formed by its intersections with the other spheres of S. The left-hand side image in Figure 1 illustrates a spherical arrangement.
A void — Let b_i be the ball representing the atom whose boundary is the sphere s_i. A void of the molecule is a bounded maximal connected component of $\mathbb{R}^3 \setminus \bigcup_{i=1}^{n} b_i$.
An exposed face of a spherical arrangement is a face that appears on the boundary of the union of the spheres (outer boundary or void).
Trapezoidal decomposition — Given a collection C of little circles on s_i (namely intersections of the sphere s_i with other spheres and hence not necessarily great circles), the trapezoidal decomposition is a refinement of the spherical arrangement $\mathcal{A}(C)$ that makes each face of the arrangement homeomorphic to a disc with at most four edges on its boundary (see [14] for more details on trapezoidal decompositions). In this context, we fix a pair of antipodal points as *poles*. We call the great circles through the poles *polar circles* and the arcs of polar circles *polar arcs*. Any point on a little circle that is tangent to a polar circle is called a *polar tangency*. For every polar tangency of every circle (except for circles that encompass a pole), we extend a polar arc in either direction until it hits another little circle or reaches a pole. We do the same from every intersection point of a pair of little circles. This refinement is called the *full trapezoidal decomposition*. Occasionally it is sufficient to use the sparser *partial trapezoidal decomposition*, in which polar arcs are extended only from polar tangency points. Figure 1 illustrates both the full and partial trapezoidal decompositions of a spherical arrangement.
ET-tree — a dynamic balanced binary tree over some *Euler tour* around a tree T. An Euler tour around a tree is a maximal closed walk over the graph obtained from the tree by replacing each edge by a directed edge in each direction. The walk uses each directed edge once, so if T has n vertices, the cyclic Euler tour has length $2n-2$. If we merge two trees or split a tree, the new Euler tours can be constructed by at most two splits and two concatenations of the original Euler tours, which take $O(\log n)$ time while maintaining the balance of the ET-tree. Each vertex of the tree may occur several times in the Euler tour, and one of these occurrences is chosen arbitrarily as a representative. Each ET-node represents the set of representative leaves below it, and may hold data that represent these leaves. See [20, 21] for more details.

3. DYNAMIC MAINTENANCE UNDER CONFORMATIONAL CHANGES

We compute a highly accurate representation of the boundary of a molecule (both the outer boundary and the voids), and the surface area of each connected component of the boundary. The contributions in terms of surface area of each atom to the outer boundary of the molecule and to the voids are also calculated. We initially compute this information when the molecule is first loaded. For that purpose we construct the spherical arrangement (Section 2) for each atom sphere and connect these spherical arrangements of intersecting atoms to form a subset of the 3D arrangement of the spheres of the atoms, which is traversed in order to find the two-dimensional faces of the arrangement that form the boundary of the molecule.

First we outline the static construction of the surface (based on [18]), and then we explain the extensions for the dynamic maintenance of the surface under conformation changes.

3.1 The Initial Construction of the Surface

Halperin and Shelton [18] presented a software package for computing the boundary surface of the union of spheres, the surface area of that boundary and the intersection pattern of any sphere with all the other spheres in a given set. They introduced a perturbation scheme, controlled perturbation, that overcomes degeneracies and precision problems in computing spherical arrangements while using floating-point arithmetic. We recently [15] modified this package to improve the running time, mainly by generalizing the implementation of the trapezoidal decomposition, which significantly reduced the perturbation time for large molecules.

Given a collection $S = \{s_1, s_2, ..., s_n\}$ of n spheres, their arrangement $\mathcal{A}(S)$ is built in an incremental fashion (that is, adding one sphere at a time). The spherical arrangement of each sphere is connected to the spherical arrangements of the spheres that intersect it, to construct a subset of the three-dimensional arrangement of spheres that includes all the features of that arrangement except the 3-dimensional cells. A subset of the 2-dimensional faces of this structure forms the boundary surface of the molecule. We compute both the outer boundary and the boundary of each of the voids.

In order to build the arrangements, it is required to find all pairs of intersecting atoms. The implementation described in [18] uses a simple grid based solution [17]. This data structure (called *3D-hash*) exploits the fact that Van der Waals potentials prevent atom centers from coming very close to one another. It can be computed in $\Theta(n)$ time, and determining which spheres intersect any given sphere of S takes $O(1)$ time. Hence, finding all pairs of intersecting spheres takes $\Theta(n)$ time.

After the arrangement of the spheres is built, the boundary of the molecule is found by traversing the *regions* (two-dimensional faces) of the arrangement, starting from the bottommost region. During this traversal, the areas of the

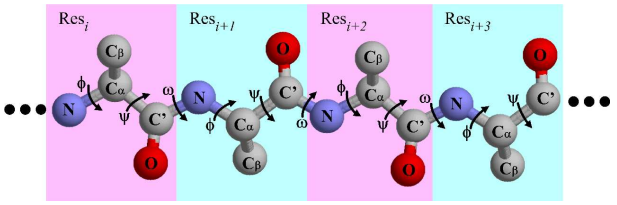


Figure 2: An illustration of a protein fragment with its backbone DOFs (taken from [25]). The C_β atoms are part of the side chains, and the rest of the atoms belong to the backbone. The ϕ torsion angle is the angle between the plane of $\triangle C'NC_\alpha$ and the plane of $\triangle NC_\alpha C'$. The ψ torsion angle is the angle between the plane of $\triangle NC_\alpha C'$ and the plane of $\triangle C_\alpha C'N$.

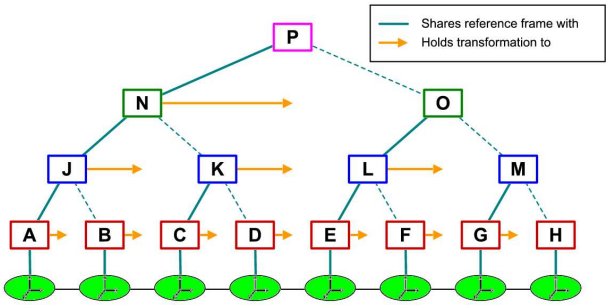


Figure 3: The ChainTree : A binary tree that combines the transform and BV hierarchies (taken from [25]).

traversed regions are calculated and summed, to find the total surface area. This is repeated for each connected component of the boundary.

3.2 The ChainTree

When we allow the atoms of the molecule to move, it is practically expensive to update the 3D-hash and reconstruct the arrangements and the surface. Even though the grid algorithm is asymptotically optimal in the worst case, it requires reconstruction from scratch of the entire structure, which may be prohibitively slow for large molecules.

In [26] Lotan *et al* introduced a novel data structure called the *ChainTree* (CT) aiming to speed up the energy computation during Monte Carlo Simulation of proteins. They take advantage of the fact that proteins are long kinematic chains (and not an arbitrary collection of spheres) and that few degrees of freedom (DOFs) are changed at each step of the simulation.

A protein is the concatenation of small molecules (the amino acids) forming a long backbone chain with small side chains (called residues). Since bond lengths and angles between any two successive bonds are almost constant across all conformations at room temperature [13], it is common practice to assume that the only DOFs of a protein are its torsion angles. Each amino acid contributes two torsion DOFs to the backbone — the so-called ϕ and ψ angles. Thus, the backbone is commonly modeled as a long chain of links separated by torsion joints. A link, which designates a rigid part of a kinematic chain [10], is a group of atoms with no DOFs between them. The side chains may also have degrees of freedom (between 0 and 4), but in our current implementation we assume there are no DOFs in the side chains (see Section 7). Figure 2 illustrates a fragment of a protein backbone with its DOFs.

The CT is motivated by the following properties of the kinematic chain model of the protein: Local changes have global effects, small angular changes may cause large motions and large sub-chains remain rigid at each step. It is made of two hierarchies: A transform hierarchy that maintains the kinematics of the backbone and a bounding-volume (BV) hierarchy that approximates the geometry of the protein. It is a balanced binary tree that combines those two hierarchies. The leaves of that tree correspond to the links of the protein's backbone with their attached side chains. Each leaf holds both the bounding box that bounds the corresponding link and side chain and the transform to the reference frame of the next link. Each internal node has the frame of the leftmost link in its sub-tree associated with it. It holds both the bounding box of the boxes of its two children, and the transform to the frame of the next node at the same level (which is the product of the transforms of its two children). Figure 3 illustrates the CT. Two algorithms are described for the CT in [26]. The *updating* algorithm updates a minimal set of transforms and BVs of the CT after a k-DOFs change. The *testing* algorithm uses the CT to detect self-collisions after a k-DOFs change.

The performance of the CT is summarized in the following theorem, which is proved in [26].

THEOREM 1. *[26] Updating the CT after a k-DOFs change takes $O(k\log(\frac{n}{k}))$ time and using the CT to test for self-collisions after a k-DOFs change takes $O(n^{\frac{4}{3}})$ time.*

3.3 The IntersectionsTree

In our application we use the CT to detect self-collisions and to find the new pairs of intersecting atoms after performing DOF changes. When a DOF change is accepted (when it incurs no self-collisions), we have to modify some of the spherical arrangements and portions of the arrangement of the spheres in order to compute the new boundary surface of the molecule and its area[2]. We need to find all the intersecting pairs of atoms which changed due to the last DOF change. These pairs include *deleted pairs* that intersected before the change but no longer intersect, *inserted pairs* that did not intersect before the change but intersect after the change and *updated pairs* that intersected before and intersect after the change, but have moved relative to each other. Only pairs of atoms that belong to different leaves of the CT can be among those pairs we seek (because a pair of intersecting atoms from the same leaf can never move relative to one another). To find these pairs we introduce a data structure called the *IntersectionsTree* (IT), similar to the EnergyTree in [26] which was used to store partial energy sums in the Monte Carlo Simulation. In our case we store pairs of intersecting atoms.

Let α and β be any two nodes (not necessarily distinct) from the same level of the CT. If they are not leaf nodes,

[2]In order to use the surface area in energy calculations for the acceptance criteria, these calculations will have to be done in each step of the simulation, and in rejected steps will be reversed.

let α_l and α_r (β_l and β_r) be the left and right children of α (β), respectively. Let $I(\alpha,\beta)$ denote a set that contains all the pairs of intersecting atoms in which one atom belongs to the sub-chain corresponding to α (the section of the protein chain contained in the leaves of the sub-tree of α) and the other atom belongs to the sub-chain corresponding to β. If $\alpha \neq \beta$, we have:

$$I(\alpha,\beta) = I(\alpha_l,\beta_l) \cup I(\alpha_r,\beta_r) \cup I(\alpha_l,\beta_r) \cup I(\alpha_r,\beta_l). \quad (1)$$

Similarly, the set $I(\alpha,\alpha)$ of intersecting atom pairs inside the sub-chain corresponding to α can be decomposed as follows:

$$I(\alpha,\alpha) = I(\alpha_l,\alpha_l) \cup I(\alpha_r,\alpha_r) \cup I(\alpha_l,\alpha_r). \quad (2)$$

These two recursive equations yield the IT. The IT has as many levels as the CT. Its nodes at any level are all the pairs (α,β), where α and β are nodes from the same level of the CT. If $\alpha \neq \beta$ and they are not leaves of the CT, then the node (α,β) of the IT has four children $(\alpha_l,\beta_l), (\alpha_r,\beta_r), (\alpha_l,\beta_r)$ and (α_r,β_l). A node (α,α) has three children $(\alpha_l,\alpha_l), (\alpha_r,\alpha_r)$ and (α_l,α_r). The leaves of the IT are all pairs of leaves of the CT (hence correspond to pairs of links of the protein chain). Each node (α,β) of the IT holds the intersecting atom pairs of $I(\alpha,\beta)$ after the last accepted simulation step. The root holds all the intersecting pairs.

To find the modified intersecting pairs (deleted, inserted or updated pairs), we use the testing algorithm (the same algorithm that finds self intersections). The difference between our definition of a pair of intersecting atoms and a pair of atoms that cause a self clash is in the distance between their centers. We determined the minimal distance allowed between two atom centers by choosing a distance such that the original conformation of our input molecules (taken from the Protein Data Bank [4]) is free from self clashes. Whenever the testing algorithm prunes a search path, it marks the corresponding node in the IT to indicate that the intersecting pairs stored in this node are unaffected. The update of the intersecting pairs at the nodes of the IT is done by a recursive traversal of the tree. To update the intersecting pairs at an unmarked node, we first update the intersecting pairs of its unmarked children. Then we compute the intersecting pairs of that node using Equations (1) and (2). The intersecting pairs of an unmarked leaf are found by checking all the pairs of atoms from the two links that correspond to that leaf. When we test a node that corresponds to a pair of nodes from the CT whose BVs became too far apart after the last change, we clear the intersecting pairs at this node and at the sub-tree of that node.

We summarize the worst-case performance of the IT in the following theorem.

THEOREM 2. *The overall cost of updating the IT is*[3] $O(n^{\frac{4}{3}})$.

The proof for this Theorem is the same proof given for the running time of the testing algorithm in [26]. Note that this running time holds even if we allow DOFs in the side chains. In such case, when a side chain DOF contained in some link of the CT is changed, the coordinates of the atoms of this link will be updated and these atoms will have to be tested for intersections against each other. The bounding volume of this link will be updated as well. Since the number of atoms in each link is bounded by a constant (up to 20 atoms), this extra work will be done in constant time per leaf of the IT and will not affect the asymptotic running time.

[3]The $O(n^{\frac{4}{3}})$ bounds in Theorems 1 and 2 are worst-case bounds, but the typical practical performance is much better and constitutes a negligible fraction of the overall time of an update step (see Figure 5).

3.4 Updating the Arrangements

As we update the IT, we store in a separate list, called the *Modified Intersections List* (MIL), all the modified intersecting pairs (deleted, inserted or updated) we found. This list is then used to update the spherical arrangements: For each pair of inserted intersecting atoms, we add their intersection circle to the spherical arrangements of both atoms. For each pair of updated intersecting atoms, we remove their old intersection circle from their spherical arrangements and add their new intersection circle to both arrangements. For each pair of deleted intersecting atoms, we remove their old intersection circle from the spherical arrangements of both atoms. (See Section 6 for an illustration.)

LEMMA 3. *The overall cost of updating the spherical arrangements is $O(p)$, where p is the number of atoms whose spherical arrangement is involved in a change.*

PROOF. Since the complexity of each spherical arrangement is constant [17], the cost of adding (removing) an intersection circle to (from) a spherical arrangement is $O(1)$. Since the number of intersection circles on each atom is bounded by a constant, the number of modified intersection circles on each atom is also bounded by a constant. Therefore the number of modified intersection circles is $O(p)$, and the overall cost of updating the spherical arrangements is $O(p)$. □

The relation between p and the number of simultaneous DOF changes in experiments is shown in Figure 4.

3.5 Updating the Connectivity of the Surface

After the modification of the spherical arrangements, we have to reconstruct the outer boundary and void boundaries of the molecule and to calculate their areas, as well as the contribution of each atom to the outer boundary and to the voids. This update takes $\Theta(n)$ time, since we traverse the entire boundary, which has an overall $\Theta(n)$ complexity in the worst case. However, a great deal of the required calculations depend on the number p of modified atoms in the current step.

Avoiding the traversal of the spherical arrangements that have not changed requires some more care in terms of identifying connected components of the boundary. Consider a degenerate scenario where in two consecutive steps the boundary consists of a single connected component — the outer boundary; in such case it would have been trivial to recalculate these attributes in time proportional to the number p of modified atoms. However, in general there can be topological changes and connected components of the boundary may merge, split, newly appear or disappear. We now present an efficient approach that despite the topological changes can accurately recompute the attributes of every boundary component in total time $O(p \log^2 n)$.

For that purpose we adapt tools from dynamic maintenance of graph connectivity. We define the following graph: Each exposed region of the spherical arrangements becomes a vertex of the graph; two vertices of the graph are connected by an edge if their respective regions are adjacent on

the boundary of the union of all spheres. As the molecule undergoes DOF changes, some regions are modified or deleted and new regions are created. These changes are reflected in the graph by deleting the vertices of deleted and modified regions and adding the vertices of new and modified regions. For each deleted region, all the edges incident to its vertex in the graph are deleted. In order to maintain the connected components of the boundary of the molecule, we simply need to maintain the connected components of this graph as the molecule undergoes DOF changes. One connected component of the graph represents the outer boundary of the molecule and the rest of the components represent the voids.

In [21] Holm *et al* present a poly-logarithmic deterministic fully-dynamic algorithm for graph connectivity. Their algorithm maintains a spanning forest of a graph, answers connectivity queries in $O(\log n)$ time in the worst case[4] and uses $O(\log^2 n)$ amortized time per insertion or deletion of an edge. Here n, the number of vertices of the graph, is assumed to be fixed as edges are added and removed. In our case the vertices are not fixed, since we create and delete regions during the DOF changes. However, the number of vertices throughout the simulation remains $O(n)$ [17, 26], and therefore the algorithm still works with the same amortized time bound. We next describe our extension of this algorithm to efficiently maintain the *surface area* of the boundary of the molecule without traversal of the entire boundary.

The connectivity algorithm in [21] maintains a spanning forest F of the input graph G, and uses for this purpose a data structure called ET-tree (see Section 2). The edges are split into $\ell_{\max} = \lfloor \log_2 n \rfloor$ levels, and a hierarchy $F = F_0 \supseteq F_1 \supseteq ... \supseteq F_{\ell_{\max}}$ of spanning forests is maintained, where F_i is the subforest of F induced by the edges of level $\geq i$. The amortization argument of the algorithm is based on increasing the levels of the edges (the level of each edge can be increased at most $\ell_{\max}$ times).

In [21] each representative vertex of an ET-tree in the forest F_i holds a key for each incident level i edge and each internal node of the ET-tree holds the number of representative leaves and one of the incident edges in its sub-tree. This information is maintained in $O(\log n)$ time per split or merge of the ET-trees. In a similar fashion, we add to each representative vertex the area of its respective region. Each internal node of the ET-tree will hold the sum of the areas of the representative leaves in its sub-tree. The root of each tree of F will hold the total surface area of that connected component. Maintaining the area information in the ET-trees takes $O(\log n)$ time per each split or merge of the ET-trees. Maintaining this information in the spanning forest F takes $O(\log^2 n)$ amortized time when an edge is inserted or deleted. To summarize:

THEOREM 4. *(i) The amortized cost of recalculating the surface area of the outer boundary and voids of the molecule is $O(p \log^2 n)$, where p is the number of atoms whose spherical arrangement is involved in a change. (ii) The cost of computing the contribution of an atom to the boundary and all the voids is $O(\log n)$.*

PROOF. (i) The number of inserted and deleted regions involved in a change is $O(p)$, as the complexity of each spherical arrangement is bounded by a constant. Since each insertion or deletion of an edge of G takes $O(\log^2 n)$ amortized time, the overall amortized cost is $O(p \log^2 n)$. (ii) The number of regions in an atom is bounded by a constant. Given any region of the atom, we can find the connected component it belongs to in $O(\log n)$ time by finding the root of its tree in the spanning forest F. Therefore we can compute the contribution of the atom to the surface area of all the components in $O(\log n)$ time. □

We are currently implementing the graph connectivity algorithm suggested in this section. Early results show that this implementation improves the running time of our application by up to 60% when changing a small number of DOFs simultaneously. We intend to report on more detailed results of this implementation in the near future.

4. CONTROLLED PERTURBATION

As mentioned earlier, the original static construction [18] uses controlled perturbation to overcome degeneracies and precision problems in the computation of the molecular surfaces with floating-point arithmetic. We extended the static scheme to work in the dynamic setting. We first describe the original scheme, and then the modifications required for the dynamic case.

4.1 Static Controlled Perturbation

A possible way to compute robustly without resorting to exact computation during the evaluation of predicates, is to (slightly) perturb the geometric objects such that consistent results of the predicates can be certified even when using finite precision arithmetic. A degeneracy occurs when a predicate evaluates to zero. The goal of the perturbation scheme is to cause all predicates used during the algorithm to evaluate sufficiently far away from zero so that finite precision arithmetic could enable us to safely determine whether they are positive or negative. Hence, while certifying the consistency of the predicates, all degeneracies are eliminated. Controlled perturbation has been successfully used with arrangements of polyhedral surfaces [28], with arrangements of circles [16], and recently with Delaunay triangulations [12]. The magnitude of the perturbation is utterly negligible in the context of molecular surfaces.

In the case of arrangements of spheres, the general position (non-degeneracy) assumption means that there is no outer or inner tangency between two spheres, that no three spheres intersect in a single point, and that no four spheres intersect in a common point. The controlled perturbation scheme ensures that all the features of the spherical arrangements (vertices and arcs) are at least some given ε apart.

As mentioned earlier, the arrangement $\mathcal{A}(S)$ is built incrementally. Each time we check if there is a potential degeneracy induced by the newly added sphere. If so, we perturb that sphere, so no degeneracies will occur. The main idea is to carefully relocate the sphere — move the sphere sufficiently to avoid all degeneracies, but not too much. We use a resolution parameter ε that depends on the floating-point precision and the type of operations (but is assumed to be given here). For any given resolution value $\varepsilon > 0$, a parameter δ that depends on ε and m (the maximum number of spheres intersecting any single sphere, which is a constant for the hard sphere model [17]) is determined. Each sphere center is perturbed by at most δ to resolve all the

[4]This time bound can be further improved to $O(\log n / \log\log n)$ if we use $\Theta(\log n)$-ary trees instead of binary trees to store the ET-trees of the spanning forest.

degeneracies. See Section 6 for the values of δ and ε in our experiments.

Another kind of degeneracies taken care of is degeneracies that result from the trapezoidal decomposition (Section 2). Since in the trapezoidal decomposition we are free to choose a direction for the poles, the poles are chosen so that the angular separation of the additional arcs (of the trapezoidal decomposition) will be above a certain threshold ω.

4.2 Dynamic Controlled Perturbation

When we extend the controlled perturbation scheme to the dynamic case, we have two goals in mind: (1) perturb as few atoms as possible, for efficiency reasons, and (2) avoid cascading errors as we perturb an atom several times or change a torsion angle several times.

As described in Section 3, after each set of simultaneous DOF changes we build a list of atom pairs (the MIL) whose intersection circles should be removed or added (or both) to their respective spherical arrangements. Removing the old intersection circles cannot induce new degeneracies, but adding the new circles can. We must therefore test, after each DOF change, for new degeneracies, and perturb the atoms if needed. As mentioned, we wish to test as few atoms as possible for degeneracies after each DOF change.

As in the static perturbation, we want to keep all the features of the spherical arrangements at least ε apart, for the given resolution parameter ε. For that goal it is not enough to know the new intersecting atom pairs, because a degeneracy occurs also when atoms *almost* intersect each other. Therefore we modify the MIL to include pairs of atoms which almost intersect each other. These pairs are identified while finding the intersecting pairs of unmarked leaves of the IT. When the two checked atoms do not intersect, we check if their centers are less than $r_1 + r_2 + \varepsilon + 2\delta$ apart (where r_1 and r_2 are the radii of the atoms, ε is the resolution parameter of the perturbation and δ is the perturbation parameter). If so, we add them to the MIL (but not to the IT itself). By adding these pairs, we ensure that all pairs of atoms that intersect, or are less than ε apart, will be available to the dynamic perturbation routine. The added 2δ term is required to ensure that a pair of atoms that were more than ε apart will remain more than ε apart after the (possible) perturbation of both of them.

Now that we have a list of all the modified intersections and near intersections, we construct a list of all the atoms that might induce degeneracies. These are the atoms that belong to inserted and updated pairs and the atoms that belong to near intersecting pairs. For each of these atoms we perform the same tests that were performed in the static perturbation to assure that all the features are ε apart. In the static perturbation we used the 3D-hash (the structure mentioned in Section 3.1) to find all the atoms that intersect or nearly intersect the tested atom. Now we do not have the 3D-hash. However, when we built the spherical arrangement of each atom, we kept for each spherical arrangement a list of the atoms that intersect that atom. Taking this list and modifying it with the information stored in the MIL (removing atoms of deleted pairs and adding atoms of inserted and almost intersecting pairs), we can get the list of atoms intersecting or almost intersecting the tested atom after the current DOF change. Each atom tested for degeneracies is checked against this list.

When we find a degeneracy, we perturb the atom center within a sphere of radius δ around the *original center* of the atom *within its reference frame*. This ensures that the center of the perturbed atom will be at most δ apart from its accurate place within the frame, which prevents cascading of errors caused when perturbing the same atom many times (in different steps). This, however, is not the case when we compute the global coordinates of the atom center. We have taken measures to avoid cascading of errors in transformations; we omit details for lack of space. The perturbation process is repeated until the atom no longer induces a degeneracy. For each perturbed atom we must re-compute its entire spherical arrangement including the circles of intersections of this atom with other atoms.

Next we need to take care of the degeneracies that result from the trapezoidal decomposition. The atoms that need to be tested are atoms on which we add new polar arcs (due to changes in the relative position of some of the atoms that intersect with them). These are the same atoms whose centers were tested earlier for degeneracies. Again we have to re-compute the entire spherical arrangement of any atom whose poles direction is changed (as well as the intersection circles of this atom with other atoms).

THEOREM 5. *Using the same perturbation parameters δ and ω of the static perturbation, the expected update time of the spherical arrangements including the perturbation time is $O(p)$.*

PROOF. The perturbation parameters δ and ω calculated in [18] ensure the finding of a non degenerate atom center or pole direction for a given atom in expected $O(1)$ time. Since the perturbation tests (and possibly the perturbations themselves) are done only for modified atoms, the dynamic perturbation takes expected $O(p)$ time (recall that p is the number of atoms whose spherical arrangement is involved in a change). For each perturbed atom we have to reconstruct its entire spherical arrangement, which takes constant time, since the size of each spherical arrangement is bounded by a constant. □

5. IMPLEMENTATION DETAILS

Our software is written in C++ with an Open GL graphics interface.

5.1 Improvements to the Static Construction of the Surface

As mentioned earlier, we modified the static construction of the surface as originally described in [18]. The main improvement is in the implementation of the trapezoidal decomposition. The original implementation finds a single pole direction (for all the spherical arrangements at once) that induces no degeneracies in all the atoms; this uniform direction indeed considerably simplifies the implementation (for example, assuming a north pole at $(0, 0, 1)$ simplifies the calculation of the polar tangency points). However, using a single pole direction for all the atoms incurs a huge performance burden in some cases. When running the application with large molecules (thousands of atoms), finding a single pole direction that eliminates all degeneracies may take a long time (a large number of constraints must be met for each pole direction, which takes a long time to determine for large molecules, and in addition to that a large number of pole directions are sampled and checked before we

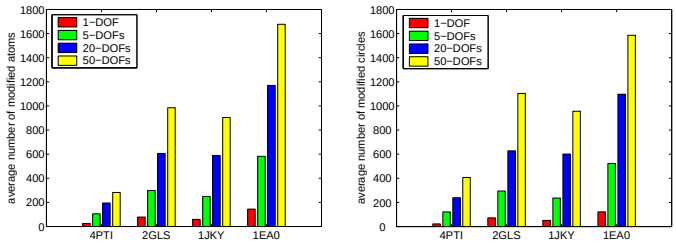


Figure 4: The average number of modified atoms (left) and average number of modified intersection circles (right) as a function of the number of simultaneous DOF changes. The average is only over the accepted simulation steps out of a total of a 1,000 steps in each simulation.

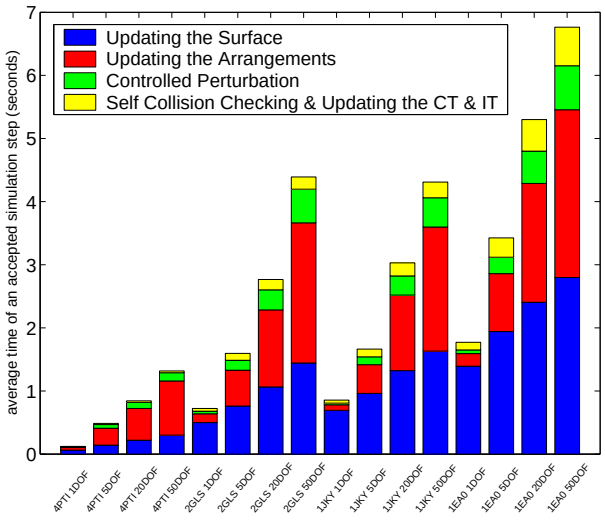


Figure 5: The average relative times of the main components of our application in a single accepted simulation step.

find a valid direction), or even be impossible. Therefore we modified the application to choose a (possibly) different pole direction for each atom. This modification later became essential for our dynamic maintenance under conformation changes, since it allows us to change the pole direction of a single atom without affecting the pole directions of the rest of the atoms. For more details regarding this and other improvements, see [15]. Table 1 in Section 6 shows the construction times of the surface before and after our modifications.

5.2 Building the Protein Chain

Our software reads PDB files [4]. We identify the backbone and side-chain atoms, and divide them to maximal rigid groups (or links) without DOFs. We construct the backbone chain, and for each link of the chain compute its reference frame, and the transformation to the next link in the chain. For this purpose we use Atomgroup Local Frames as described in [35]: The origin of each frame (except for the first frame whose origin is the center of the backbone N atom) is the center of the backbone C' or C_α atom that belongs to the relevant link. The z-axis of each frame (except for the first frame whose axes are the global axes) is the vector from the frame origin along the rotatable bond that connects this link to the previous link. The x-axis is perpendicular to the z-axis, and the y-axis completes the frame to form a right-hand system. Once we have a coordinate frame for each rigid link we can compute the transformation from each frame to its following frame. Next we use these transformations to compute the local frame coordinates of each atom center. From that point on we use only local frame coordinates in our computations. When we need to work with coordinates that belong to different frames, we use the transformation between them to convert the coordinates to the same frame. We only have to compute the global coordinates of the atoms for displaying the molecule.

6. EXPERIMENTAL RESULTS

The experiments described in this section were all executed on a 1 GHz Pentium III machine with 2 GB of RAM. The perturbation parameters that were used are $\delta = 10^{-7}, \varepsilon = 10^{-8}$ and $1 - \cos(\omega) = 10^{-9}$.

Table 1 shows the total time it takes to build the static spherical arrangements (including the perturbation time) with the original implementation as well as with the improved implementation, where m is the number of atoms that intersect a single atom. The new implementation has other technical improvements in addition to the improvement described in Section 5.1 [15]. As can be seen, the improvements to the original implementation have been a crucial prerequisite to the effectiveness of the dynamic solution.

Table 2 describes the proteins used in our experiments reported here. In PDB files that contain more than one backbone chain, we handle only the first chain.

Since the update time of the spherical arrangements depends on the number of modified intersecting circles in each step of the simulation, and the update time of the surface area depends on the number of modified atoms, we tested the relation between the number of simultaneous DOF changes, and the number of modified intersection circles and modified atoms. Each simulation consisted of a 1,000 steps. At each step the changed DOFs were picked uniformly at random and the magnitude of the change was chosen uniformly at random between -1° and 1° (we chose small angle changes in order to increase the number of accepted steps). The results, reported in Figures 4 & 5 and in Table 3, refer only to accepted simulation steps whose number was usually several hundreds (the time taken by rejected simulation steps is negligible compared to accepted steps). The results show a strong connection between the number of simultaneous DOF changes and the number of modified atoms and intersection circles. They also show that for small values of the number of simultaneous changes, the number of modified atoms and intersection circles is less affected by the size of the protein.

In Table 3 we compare the time it takes to update the surface after a k-DOF change to the time it takes to reconstruct the surface from scratch. The reconstruction time is the time it takes to construct the static surface (not including the time spent on the construction of the CT and IT). The update time is the average time (for accepted steps) it

Table 1: Total time (in seconds) of computing the surface (including the perturbation).

Input File	# of Atoms	Max m	Mean m	Single Pole Direction	Multi Pole Directions
1BZM.pdb	2034	10	5.74	14.92	8.31
1JKY.pdb	5734	13	6.24	163.68	26.94
7AT1.pdb	7106	12	5.70	63.80	28.40
1L7X.pdb	12882	12	5.85	$>$ 24 hours	54.50

Table 2: Proteins used in experiments; m is the number of spheres intersecting any single sphere.

Input File	# of Atoms	# of Amino Acids	# of Links	Max m	Mean m
4PTI.pdb	454	58	117	10	5.79
2GLS.pdb	3636	468	937	13	6.33
1JKY.pdb	5614[5]	748	1497	13	6.24
1EA0.pdb[6]	11180	1452	2905	13	6.14

takes to update the CT, the IT, the spherical arrangements and the surface. We made this comparison for several values of simultaneous DOF changes. For each update time, we give the percentage of that time from the reconstruction time. We can see that as the proteins grow larger, our method becomes more effective. As expected, the update is faster for small numbers of simultaneous DOF changes. It is interesting to notice that the percentages in this table are very similar to the percentages of the modified atoms in each simulation, which means that in practice our implementation runs in time proportional to p.

Figure 5 shows the fractions of the average running time taken by the main components of our application. It is important to notice that the update of the IT, while being the component with the highest asymptotic worst-case time complexity, takes a small percentage of the total running time.

In Figure 6 we see the effect of a single DOF change on the structure of the backbone. On the top we see the backbone atoms of 4PTI in their original conformation. On the bottom we see the backbone after a 180° change in the ψ angle of the 13th residue. We can see that all the atoms located after that residue in the chain moved. However, our application detected only 13 modified intersection circles and that only 14 atoms out of 454 were affected by this change.

In Figure 7 we see the spherical arrangement of the N atom of the 14th residue that was affected by the single DOF change in 4PTI. The image on the left shows the arrangement before the change, and the image on the right shows it after the change.

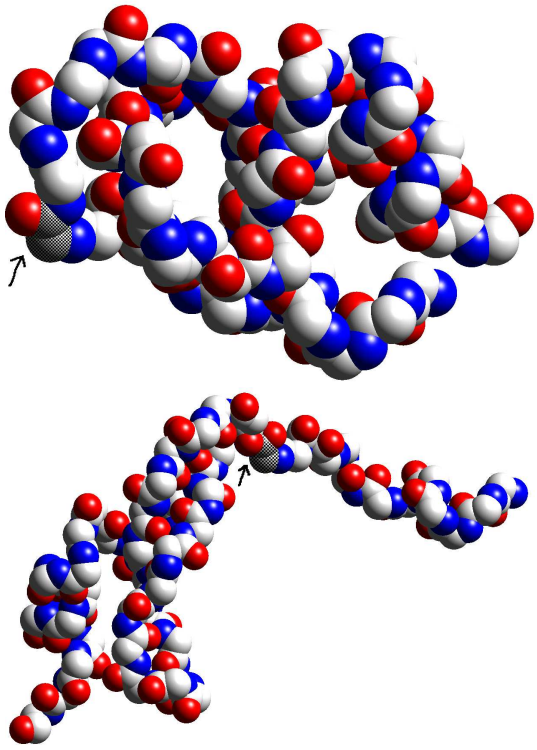


Figure 6: The Backbone of 4PTI before (top) and after (bottom) a 1-DOF change. The changed DOF is marked.

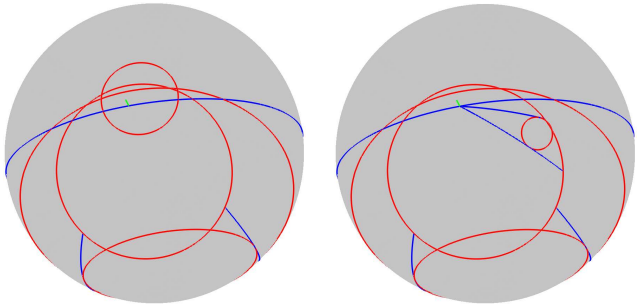


Figure 7: The spherical arrangement of one of the atoms of 4PTI that were affected by the DOF change, before (left) and after (right) the change.

7. FUTURE DIRECTIONS

In our implementation we dynamically maintain the van der Waals and solvent accessible surfaces. A natural extension is to dynamically maintain the smooth molecular surface. In [17] a simple transformation is described for computing the smooth surface from the solvent accessible surface. Another possible (and fairly straightforward) extension is to allow DOFs in the side chains of the protein. We also plan to measure how the graph connectivity algo-

[5]The number of atoms here is smaller than the number given for the same molecule in Table 1, because here we only count the atoms of the first chain of the molecule, whereas in Table 1 we count all the atoms of the molecule.

[6]This molecule appears on the first page.

Table 3: Time (in seconds) of static reconstruction vs. dynamic modification of the surface.

Input File	# of Atoms	static	1-DOF	5-DOFs	20-DOFs	50-DOFs
4PTI.pdb	454	1.99	0.12 (6%)	0.49 (24.6%)	0.85 (42.7%)	1.33 (66.8%)
2GLS.pdb	3636	18.75	0.73 (3.9%)	1.62 (8.6%)	2.81 (15%)	4.45 (23.7%)
1JKY.pdb	5614	27.97	0.86 (3.1%)	1.68 (6%)	3.07 (11%)	4.37 (15.6%)
1EA0.pdb	11180	54.72	1.78 (3.2%)	3.46 (6.3%)	5.37 (9.8%)	6.86 (12.5%)

rithm suggested in Section 3.5 would improve the running time of our implementation.

Acknowledgments

Our implementation is based on code by Itay Lotan (the CT) and by Christian Shelton (the static construction of a molecular surface). We thank Itay Lotan for useful discussions and helpful comments on an earlier draft of this paper, as well as for letting us use Figures 2 and 3 [25]. We also thank Efi Fogel for his help with the Open GL interface.

8. REFERENCES

[1] P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[2] B. J. Alder and T. E. Wainwright. Phase transition for a hard sphere system. *J. Chem. Phys.*, 27:1208–1209, 1957.

[3] C. L. Bajaj, V. Pascucci, A. Shamir, R. J. Holt, and A. N. Netravali. Dynamic maintenance and visualization of molecular surfaces. *Discrete Applied Mathematics*, 127(1):23–51, 2003.

[4] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[5] K. Binder and D. Heerman. *MCS in Statistical Physics*. Springer Verlag, Berlin, 2nd edition, 1992.

[6] R. Bryant, H. Edelsbrunner, P. Koehl, and M. Levitt. The area derivative of a space-filling diagram. *Discrete & Computational Geometry*, 32:293–308, 2004.

[7] H. L. Cheng, T. K. Dey, H. Edelsbrunner, and J. Sullivan. Dynamic skin triangulation. *Discrete & Computational Geometry*, 25:525–568, 2001.

[8] M. L. Connolly. Analytical molecular surface calculation. *J. of Applied Crystallography*, 16:548–558, 1983.

[9] M. L. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221:709–713, 1983.

[10] J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Prentice Hall, 3rd edition, 2005.

[11] H. Edelsbrunner, M. Facello, P. Fu, and J. Liang. Measuring proteins and voids in proteins. Technical report, Department of Computer Science, Hong Kong University of Science and Technology, 1994.

[12] S. Funke, C. Klein, K. Mehlhorn, and S. Schmitt. Controlled perturbation for Delaunay triangulations. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1047–1056, 2005.

[13] A. Grosberg and A. Khokhlov. *Statistical physics of macromolecules*. AIP Press, New York, 1994.

[14] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. Chapman & Hall/CRC, 2nd edition, 2004.

[15] D. Halperin and E. Eyal. Improved implementation of controlled perturbation for arrangements of spheres. Technical Report ECG-TR-363208-01, Tel-Aviv University, 2003.

[16] D. Halperin and E. Leiserowitz. Controlled perturbation for arrangements of circles. *International Journal of Computational Geometry and Applications*, 14(4 & 5):277–310, 2004.

[17] D. Halperin and M. H. Overmars. Spheres, molecules, and hidden surface removal. *Computational Geometry: Theory and Applications*, 11(2):83–102, 1998.

[18] D. Halperin and C. R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Comput. Geom. Theory Appl.*, 10:273–287, 1998.

[19] H. Hansmann and Y. Okamoto. New Monte Carlo algorithms for protein folding. *Current Opinion in Structural Biology*, 9(2):177–183, 1999.

[20] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.

[21] J. Holm, K. De Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001.

[22] A. R. Leach. *Molecular modeling: Principles and applications*. Addison Wesley Longman Limited, 1996.

[23] B. Lee and F. M. Richards. The interpretation of protein structure: Estimation of static accessibility. *J. of Molecular Biology*, 55:379–400, 1971.

[24] S. M. LeGrand and K. M. Merz. Rapid approximation to molecular surface area via the use of boolean logic and lookup tables. *Comput. Chem.*, 14:349–352, 1993.

[25] I. Lotan. *Algorithms Exploiting the Chain Structure of Proteins*. PhD thesis, Dept. Comp. Sci., Stanford Uni., 2004.

[26] I. Lotan, F. Schwarzer, D. Halperin, and J.-C. Latombe. Algorithm and data structures for efficient energy maintenance during Monte Carlo simulation of proteins. *Journal of Computational Biology*, 11(5):902–932, 2004.

[27] P. Mezey. Molecular surfaces. In K. B. Lipkowitz and D. B. Boyd, editors, *Reviews in Computational Chemistry*, volume I, pages 265–294. VCH Publishers, 1990.

[28] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th ACM Sympos. Comput. Geom.*, pages 163–172, 1999.

[29] F. M. Richards. Areas, volumes, packing, and protein structure. *Annual Reviews of Biophysics and Bioengineering*, 6:151–176, 1977.

[30] M. F. Sanner and A. J. Olson. Real time surface reconstruction for moving molecular fragments. In *Pacific Symposium on Biocomputing '97*, Maui, Hawaii, 1997.

[31] M. F. Sanner, A. J. Olson, and J. C. Spehner. Fast and robust computation of molecular surfaces. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C6–C7, 1995.

[32] E. Stein, L. Rics, and A. Brünger. Torsion-angle molecular dynamics as a new efficient tool for NMR structure calculation. *J. of Magnetic Resonance*, 124:154–164, 1997.

[33] O. V. Tsodikov, M. T. Record, Jr., and Y. V. Sergeev. Novel computer program for fast exact calculation of accessible and molecular surface areas and average surface curvature. *J. Computational Chemistry*, 23:600–609, 2002.

[34] A. Varshney, F. P. Brooks Jr., and W. V. Wright. Computing smooth molecular surfaces. *IEEE Computer Graphics and Applications*, 14:19–25, 1994.

[35] M. Zhang and L. E. Kavraki. A new method for fast and accurate derivation of molecular conformations. *J. of Chemical Information and Computing Sciences*, 42:64–70, 2002.