TEL-AVIV UNIVERSITY
RAYMOND AND BEVERLY SACKLER
FACULTY OF EXACT SCIENCES
SCHOOL OF COMPUTER SCIENCE

Dynamic Maintenance of Molecular Surfaces under Conformational Changes

Thesis submitted in partial fulfillment of the requirements for the M.Sc. degree in the School of Computer Science, Tel-Aviv University

by

Eran Eyal

The research work for this thesis has been carried out at
Tel-Aviv University
under the supervision of Prof. Dan Halperin

July 2005

Acknowledgments

I would like to thank my supervisor Prof. Dan Halperin for his guidance and great help during this research.

I thank all the fellow students in the Robotics and Vision lab for their assistance and support.

I thank Itay Lotan for useful discussions and helpful comments.

I would also like to thank the authors of code used in my implementation: Christian Shelton for his implementation of the static construction of a molecular surface, Itay Lotan for his implementation of the ChainTree, and Iyer *et al* for their implementation of the dynamic graph connectivity algorithm.

Work reported in this thesis 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.

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 [38], and (ii) data structures for self-collision testing and energy maintenance of proteins that change conformation [49]. 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, in particular in Monte Carlo Simulation. In addition we propose, analyze and implement 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. This method greatly improves the running time of our algorithm on most inputs, as we demonstrate in the experiments reported here.

Contents

Chapter 1

Introduction

Molecular simulations are an important tool in the study of conformations adopted by proteins, which in turn is an important topic in structural molecular biology. Instead of representing individual solvent molecules, some of these simulations use implicit solvent models, which require information regarding the surface area of the studied molecules exposed to the engulfing solution. This motivates fast methods of maintenance of the surface area of molecules dynamically during conformation changes.

In our work we maintain the boundary and surface area of protein molecules 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 representation of the outer 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 outer boundary and to the voids, which can be useful in various applications such as Monte Carlo simulation. Our use of controlled perturbation ensures the robustness of our implementation even while using floating-point arithmetic.

Our work extends and combines two previous results: (i) controlled perturbation for static molecular surfaces [38], and (ii) data structures for self-collision testing and energy maintenance of proteins that change conformation [49].

A major novel component of our work is an efficient method for maintaining the topology of the surface when a few torsion angles are changed in a single simulation step. This method is 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 algorithm works very well when few torsion angles are changed in each step, in which case p is much smaller than the number of moving atoms.

In our best experimental results we managed to update a molecular surface under conformational changes (with a single degree of freedom (DOF) modified at each iteration) in about 1% 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. The algorithm is useful in particular for Monte Carlo simulation, where few DOFs are modified in each simulation step.

A paper summarizing the main ideas and results of this thesis [24] has been recently presented in the 21st ACM Symposium on Computational Geometry (SoCG 2005). A paper describing the implementation of improved maintenance of molecular surfaces using dynamic graph connectivity [25] will be presented in the 5th Workshop on Algorithms in Bioinformatics (WABI 2005).

Thesis Outline

The rest of the thesis is organized as follows. In Chapter 2 we review the background and related work on molecular surfaces and molecular simulations, as well as the related work in dynamic graph connectivity. In Chapter 3 we describe our algorithm for dynamic maintenance of molecular surfaces. Chapter 4 deals with the efficient maintenance of the connected components of the boundary. Chapter 5 surveys the major implementation details. We report on experimental results in Chapter 6. Chapter 7 concludes the thesis and suggests possible directions for future work.

Chapter 2

Background

2.1 Molecular Simulations

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, in drug design, in studies of solvation and hydrophobicity, in research of the protein folding problem, and in more areas. 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. The van der Waals radii are best suited for determining which parts of space cannot contain solvent molecules (usually water molecules), because they are based on the van der Waals potential, which prevents atom centers from becoming too close to each other. There are two types of surfaces closely related to the van der Waals surface: The *solvent accessible surface* introduced by Lee and Richards [45] 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 [56] is defined by the part of the surface of the solvent probe-sphere that faces the molecule. See also [15, 16, 20] and the survey by Mezey [52] for an extensive discussion on molecular surfaces.

The functions of proteins are largely dependent on their spatial structure. Therefore, study of the conformations adopted by proteins is an important topic in structural molecular biology. Computer simulations are powerful tools in this study. The two most popular methods are Monte Carlo Simulation (MCS) [9, 39, 51] and Molecular Dynamics Simulation (MDS) [5, 44, 50, 55]. While the latter produces physically meaningful pathways, the former is more efficient at sampling the conformation space.

MCS is usually used to find low energy conformations for a given protein, and in particular its native structure. It consists of a random walk in the conformation space of the studied molecule. Each 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 (and the simulation moves to the new conformation) with probability $\min\{1, e^{-\triangle E/k_bT}\}$ (the Metropolis criterion [51]), where E is an energy function defined over the conformation space, $\triangle E$ is the difference in energy between the new and previous conformations, k_b is the Boltzman constant, and T is the temperature of the system. A downhill step to a lower-energy conformation is always accepted, while an uphill step is accepted with a probability that goes to zero as the energy difference becomes large.

MDS is a more detailed and physically accurate simulation of the dynamics of a single protein molecule in space and time. It consists of computing the forces on the atoms at each step, and using them to calculate the atom positions at the next step. To be accurate, MDS proceeds by small time steps, resulting in slow progress through conformation space. As a result, the folding of most proteins is still out of reach of even the fastest computers. 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, 64], which allows a more compact representation of the conformation space, as well as the simulation of larger time steps.

The motion of the molecules in such simulations greatly depends on the engulfing solution, which is mostly water. Computer simulations that explicitly use a large number of water molecules represent one of the most detailed approaches to study the influence of solvation on complex biomolecules [10]. In such simulations, a large fraction of the time is spent calculating a detailed trajectory of the solvent molecules, even though the main interest is the solute's behavior. Despite their cost, computer simulations with explicit solvent molecules still use approximations in some of the energy calculations [6]. These problems with the explicit solvent representation motivate different approaches, where the effect of the solvent is taken into account implicitly. See [58] for an extensive review on implicit solvent models. The implicit models can reduce the required computing power by a factor of 10 to 50 [27].

It is convenient to express the solvent effects on a molecule in an effective potential, $W = W_{elec} + W_{np}$, in which the first term accounts for electrostatic contributions and the second for non-polar contributions. W_{elec} is usually represented by continuum electrostatics [65]. W_{np} is represented in many solvation models as a weighted sum of the solvent exposed or accessible surface area of each atom of the solute [21, 27, 71]. It is represented as a linear sum of atomic contributions weighted by solvent-exposed area: $W_{np} = \sum_i \gamma_i A_i(X)$, where A_i is the solvent-exposed area of atom i (which depends on the solute configuration X) and γ_i is an estimate of the atomic free energy per unit area (which depends on the atom type).

Therefore MCS, which relies on differences of potential energy, can greatly benefit from dynamic maintenance of the surface area of a molecule during conformation changes. MDS, on the other hand, would benefit from dynamic maintenance of the derivatives of the surface area with respect to atomic position. Those derivatives contribute to the force that drives the motion [11].

Exact analytical or numerical representations of the solvent-accessible surface area or volume are generally too slow to compete with explicit solvent simulations [27]. To overcome this difficulty, approximate expressions for the solvent-accessible surface area were used [13, 27, 40, 65, 72]. For example, Cavallo *et al* [13] use a model in which each amino acid is represented by a single sphere centered on the C_α atom. Fast methods to maintain the exact surface area of a molecule dynamically during conformation changes can greatly improve these simulations.

2.2 Related Work

2.2.1 Molecular Surfaces

Several algorithms and their software implementation for calculation of the various surfaces mentioned above have been designed in the last two decades. The computational methods that evaluate the area in these surfaces can be divided into approximate and exact methods.

Most of the approximate methods rely on numerical integration, by representing the surface with a large number of dots [46, 61]. Some of the approximations are analytical but treat multiple overlapping balls probabilistically or ignore them [40, 66, 72]. Another approximate method is a grid-based algorithm [53].

The first exact analytical methods for computing the solvent accessible and smooth molecular surfaces were introduced by Connolly [15] and Richmond [57]. They have been improved in recent years [68, 69]. Some improvements were achieved in terms of computational efficiency [26, 70]. Other improvements were achieved in terms of stability [22, 30]. They handle potential degenerate positions of the input, such as nearly tangent atom spheres, by setting a user defined threshold ε that determines for example if two close vertices are considered to be the same or if two spheres are overlapping, tangent or disjoint. This approach is similar to our controlled perturbation approach, in the way that is recognizes potential degeneracies. However, once the degeneracies are identified, they are handled explicitly, unlike our approach, which eliminates the degeneracies by slightly perturbing the atom centers. Our method is easier to implement, since it does not have to handle degeneracies, and it guarantees consistent evaluation of the predicates, and hence results in *correct* topological structure of the surface of the perturbed molecule. Even though our method slightly affects the accuracy of the output by perturbing the input, the effects are negligible, and the other method also affects the accuracy when, for example, it treats atoms with a small overlap as if they do not overlap at all.

Edelsbrunner [19] used the Alpha Shape theory to compute the surface area and volume of proteins as well as for detecting and measuring cavities in proteins [47].

Sanner *et al* [60] developed a method that relies on reduced surfaces for computing the molecular surfaces. The reduced surface corresponds to the alpha shape [19] for that molecule with a probe radius α.

Halperin and Shelton [38] used *controlled perturbation* to calculate the van der Waals and the solvent accessible surfaces robustly.

2.2.2 Available Software for Molecular Surfaces

There are many implementations of molecular surfaces online. While some of them are proprietary, others are distributed freely. Connolly's extensive review [17] on molecular surfaces points to existing applications that compute them. The following applications compute properties of molecular surfaces which are similar to those that we compute in our application. AlphaShapes [1] uses alpha shapes to compute the surface area and volume of the solvent accessible surface and the molecular surface. It also finds voids and cavities and calculates their surface area and volume. ProShape [3] is an extension of AlphaShapes that also computes the derivatives of the surface area and volume with respect to atomic position. CGAL [2] contains code that calculates the alpha complex, which can be used to compute molecular surfaces.

2.2.3 Dynamic Molecular Surfaces

Limited dynamic maintenance of molecular surfaces was presented by Bajaj *et al* [7]. 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* [14] 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* [11] 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 [59] 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.

2.2.4 Molecular Simulations

Several researchers have adapted tools from robotics to the molecular simulations of proteins. Finding a self-clash in the protein chain is similar to finding a self-intersection in a multi-link robot. Some of the algorithms demand that no self-intersection occurs during the motion (caused by changes in dihedral angles), while others only require that there are no self-intersections *after* each angle change.

Soss and Toussaint [63] study polygonal chains of n edges where sub-chains may rotate rigidly around interior edges of the chain. They prove an $\Omega(n \log n)$ time bound for determining if a torsion ϕ angle change can be done at a selected edge of the chain without causing any self-intersections during the motion. They also describe a brute force algorithm for that problem that runs in $O(n^2)$ time and $O(n)$ space. Soss *et al* [62] try to use preprocessing in order to quickly determine the feasibility of an arbitrary dihedral rotation. In the static case, which only tests for feasibility without updating the chain, they give strong support for the conjecture that preprocessing a chain of n edges and performing n static dihedral rotation queries has a $\Omega(n^2)$ lower bound. In the more general dynamic case, where a feasible rotation modifies the chain, they give strong support for the conjecture that in any preprocessing scheme, the worst case query time is $\Omega(n)$, regardless of the preprocessing time.

Halperin *et al* [35] study several models for dynamic maintenance of kinematic structures where update and range query operations are supported. They prove a $\Theta(\sqrt{n})$ bound for performing update or range query operations in an n link chain where the joints connecting two links can be updated.

Lotan *et al* [49] introduced a fast implementation of MCS of proteins where a large number of atoms may move in each step. They exploit the fact that proteins are long kinematic chains. While their $O(n^{\frac{4}{3}})$ time bound for collision detection is not optimal in the worst case, it performs well in practice.

2.2.5 Dynamic Graph Connectivity

Several algorithms for dynamic graph connectivity have been designed in the last two decades. The first non-trivial fully-dynamic connectivity algorithm was presented by Frederickson [28] and supported $O(\sqrt{m})$ time updates, where m is the number of edges, and constant time queries. Eppstein *et al* [23] improved the update time to $O(\sqrt{n})$, where n is the number of vertices. Henzinger and King [41] presented a randomized algorithm supporting updates in $O(\log^3 n)$ expected amortized time and $O(\log n/\log\log n)$ time queries. Holm *et al* [42] presented a deterministic fully dynamic algorithm with $O(\log^2 n)$ amortized time updates and $O(\log n/\log\log n)$ time queries. Both poly-logarithmic algorithms use $O(m + n\log n)$ space. Thorup [67] further improved these bounds to $O(\log n(\log\log n)^3)$ expected amortized time updates, $O(\log n/\log\log\log n)$ time queries and $O(m)$ space.

Chapter 3

The Algorithm

3.1 Terminology

1. **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 [4, 33].)

2. Given a set of spheres $S = \{s_1, s_2, ..., s_n\}$, let C_i denote the collection of circles $C_i = \{s_i \cap s_j \mid s_j \in S, j \neq i\}$ which are formed by the intersections of a sphere s_i with the other spheres of S. The **spherical arrangement** $\mathcal{A}(C_i)$ is the subdivision of s_i induced by C_i. Figure 3.1(a) illustrates a spherical arrangement.

3. **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 $\mathbb{R}^3 \setminus \bigcup_{i=1}^{n} b_i$.

4. **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).

5. **Trapezoidal decomposition** — a refinement applied to each sphere s_i in S. Given a collection C_i 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 $\mathcal{A}(C_i)$. It is a standard refinement procedure that makes each face of the arrangement homeomorphic to a disc with at most four edges on its boundary (see [33] 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*. If we are only concerned with making each face

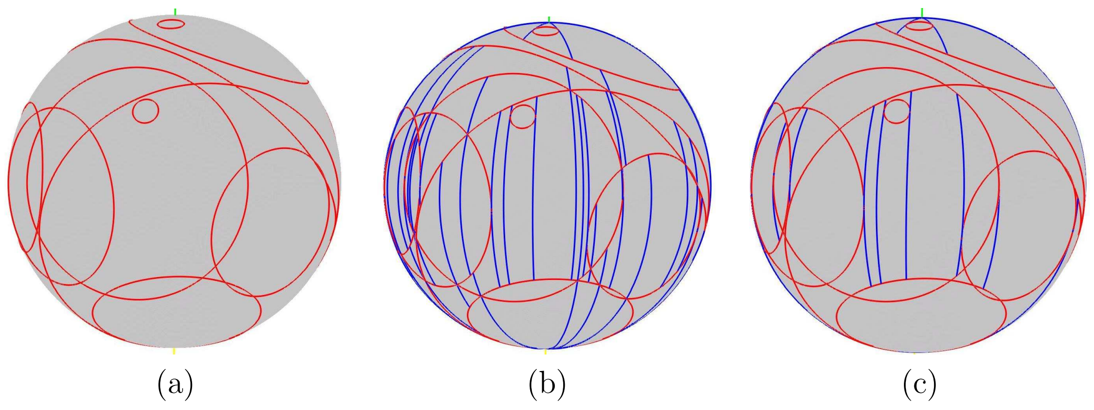


Figure 3.1: A spherical arrangement (a), its full trapezoidal decomposition (b) and its partial trapezoidal decomposition (c).

simply connected and making the graph of all the edges of the arrangement connected, we can use the *partial trapezoidal decomposition*, in which polar arcs are extended only from polar tangency points and not from intersections. Figure 3.1 illustrates both the full and partial trapezoidal decompositions of a spherical arrangement.

3.2 Static Construction of the Surface

We compute a highly accurate[1] 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 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 [38]), and then we explain the extensions for the dynamic maintenance of the surface under conformation changes.

3.2.1 The Initial Construction of the Surface

Halperin and Shelton [38] 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,

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

controlled perturbation, that overcomes degeneracies and precision problems in computing spherical arrangements while using floating-point arithmetic. We recently [34] 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 [38] uses a simple grid based solution [37]. This data structure (called *3D-hash*) exploits the fact that Van der Waals potentials prevent atom centers from coming very close to one another. In [37] it is formally shown that in a collection S of n possibly intersecting spheres of similar radii, such that no two sphere centers are closer than a small fixed distance, the number, m, of spheres that intersect any given sphere of S is bounded by a constant. It follows from that proof that the 3D-hash can be computed in $\Theta(n)$ time, and that 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 traversed regions are calculated and summed, to find the total surface area. This is repeated for each connected component of the boundary.

3.2.2 Static Controlled Perturbation

As mentioned earlier, the original static construction [38] 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 describe here the original scheme, and later (in Section 3.3.5) describe the modifications required for the dynamic case.

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 [54], with arrangements of circles [36], and recently with Delaunay triangulations [29]. The magnitude of the perturbation is utterly negligible in the context of molecular surfaces. We only sketch the method here; for details, see [36, 38].

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. See Figure 3.2 for an illustration of the degeneracies prevented by this perturbation scheme. We call these degeneracies type I degeneracies.

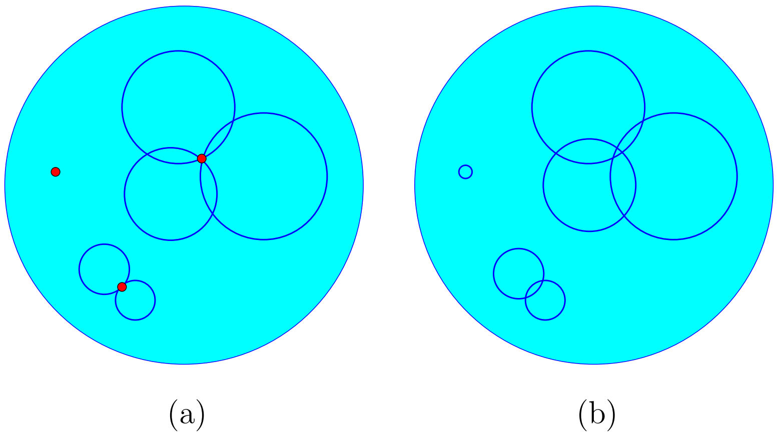


Figure 3.2: A spherical arrangement with type I degeneracies (a) : tangency of two spheres, which causes their intersection circle to become a point (left), three spheres intersecting in a single point (bottom), four spheres intersecting in a common point ; and the same arrangement after perturbing the spheres to remove the degeneracies (b).

As mentioned earlier, the arrangement $\mathcal{A}(S)$ of spheres in space 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, m (the maximum number of spheres intersecting any single sphere, which is a constant for the hard sphere model [37]) and R (the maximum atom radius) is determined. Each sphere center is perturbed by at most δ to resolve all the degeneracies. It is shown in [38] that if $\delta > 2m\varepsilon^{1/3}R^{2/3}$, all type I degeneracies can be eliminated in expected $O(n)$ time. See Chapter 6 for the values of δ and ε in our experiments.

Another kind of degeneracies, called type II degeneracies, results from the trapezoidal decomposition (Section 3.1). 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. It is shown in [38] that if $\sin(\omega) < \frac{1}{2m(m-1)}$, all type II degeneracies can be eliminated in expected $O(n)$ time. See Figure 3.3 for an illustration of this type of degeneracies.

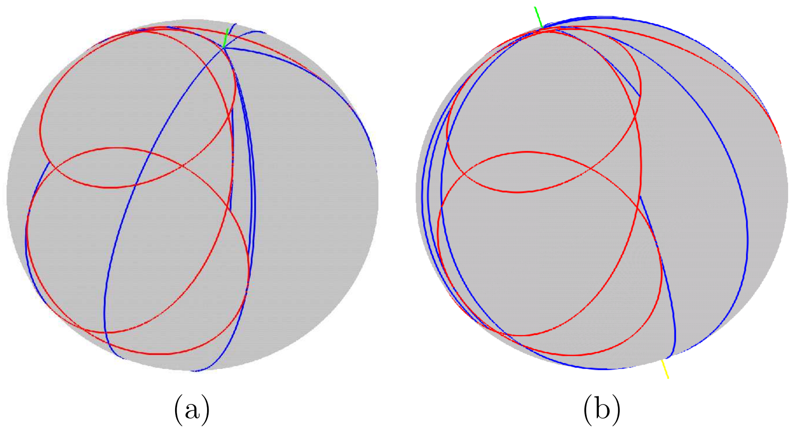


Figure 3.3: A spherical arrangement with a type II degeneracy — two polar arcs with a too small angular separation (a), and the same arrangement after perturbing the pole direction to remove the degeneracy (b).

3.3 Dynamic Maintenance under Conformational Changes

3.3.1 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 [49] 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 amino acids are almost constant across all conformations at room temperature [31], 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 [18], 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. Figure 3.4 illustrates a fragment of a protein backbone with its DOFs.

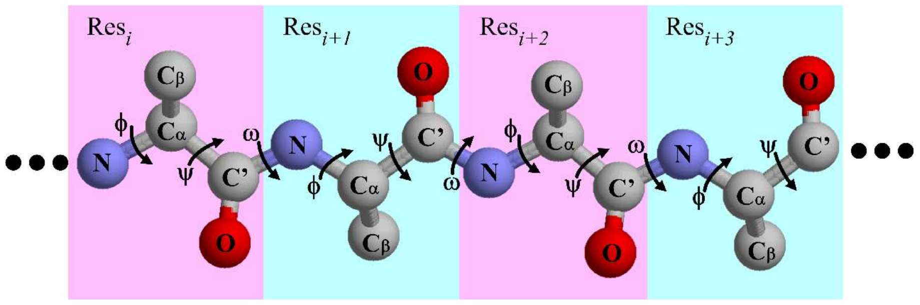


Figure 3.4: An illustration of a protein fragment with its backbone DOFs (taken from [48], courtesy of Itay Lotan). The C_β atoms are part of the side chains, and the rest of the atoms belong to the backbone. The ϕ torsion angle is the angle between the plane of $\triangle C'NC_\alpha$ and the plane of $\triangle NC_\alpha C'$. The ψ torsion angle is the angle between the plane of $\triangle NC_\alpha C'$ and the plane of $\triangle C_\alpha C'N$.

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 in the chain. 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 3.5 illustrates the CT. Two algorithms are described for the CT in [49]. 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 [49].

Theorem 3.1 [49] *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.*

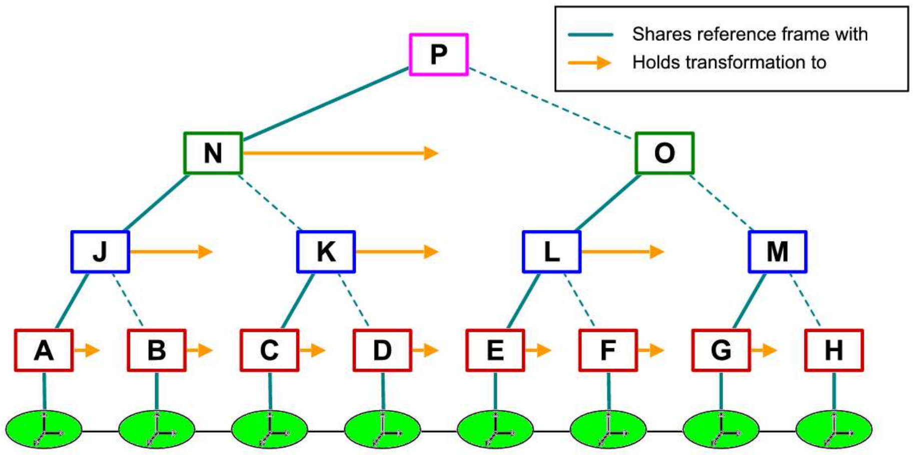


Figure 3.5: The ChainTree : A binary tree that combines the transform and BV hierarchies (taken from [48], courtesy of Itay Lotan).

3.3.2 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 [49] which was used to store partial energy sums in 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, 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:

$$I(\alpha, \beta) = I(\alpha_l, \beta_l) \cup I(\alpha_r, \beta_r) \cup I(\alpha_l, \beta_r) \cup I(\alpha_r, \beta_l). \tag{3.1}$$

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

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

$$I(\alpha, \alpha) = I(\alpha_l, \alpha_l) \cup I(\alpha_r, \alpha_r) \cup I(\alpha_l, \alpha_r). \tag{3.2}$$

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 clashes). 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 [8]) 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 (3.1) and (3.2). 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. For each new intersection found in the leaves of the IT, we check if that intersection already exists in the IT. If it does, this is an updated pair. If not, it is an inserted pair. If the atoms of a tested pair do not intersect, but the pair is found in a leaf of the IT, it is a deleted pair. Finally, 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. All of these pairs are deleted pairs. We keep a list of all the pairs of deleted, inserted and updated atoms that we found during the update of the IT.

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

Theorem 3.2 *The overall cost of updating the IT is $O(n^{\frac{4}{3}})$.*[3]

The proof of this theorem is the same proof given for the running time of the testing algorithm in [49]. 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

[3]The $O(n^{\frac{4}{3}})$ bounds in Theorems 3.1 and 3.2 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 6.3).

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.

3.3.3 Updating the Arrangements

As was mentioned before, we store in a separate list, called the *Modified Intersections List* (MIL), all the modified intersecting pairs (deleted, inserted or updated) we found during the update of the IT. This list is then used to update the spherical arrangements:

1. For each pair of inserted intersecting atoms, we add their intersection circle to the spherical arrangements of both atoms.
2. 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.
3. For each pair of deleted intersecting atoms, we remove their old intersection circle from the spherical arrangements of both atoms.

During the addition or removal of intersection circles to/from the spherical arrangements we add or remove arcs to/from these arrangements. The addition of new arcs splits existing regions, while the removal of old arcs merges existing regions. We identify the regions affected by these changes, and use this information later, while updating the connectivity of the surface, to avoid unnecessary computations for regions that were not modified.

Lemma 3.3 *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 [37], 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 $O(p)$. $\square$

The relation between p and the number of simultaneous DOF changes in experiments is shown in Figure 6.1.

In Figure 3.6 we demonstrate the effect of a single DOF change on the structure of the backbone. On the left-hand side we see the backbone atoms of Trypsin Inhibitor (4PTI) in their original conformation. On the right-hand side 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.

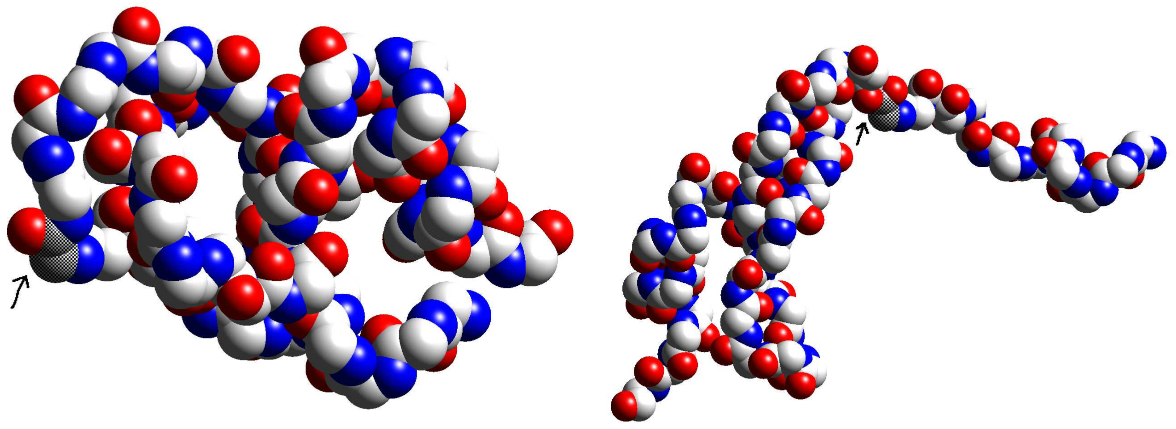

(a) (b)

Figure 3.6: The Backbone of 4PTI before (a) and after (b) a 1-DOF change. The changed DOF is pointed to by an arrow.

In Figure 3.7 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.

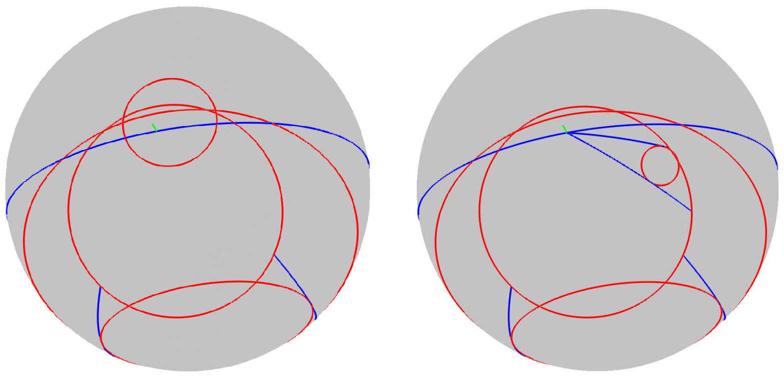

(a) (b)

Figure 3.7: The spherical arrangement of one of the atoms of 4PTI that were affected by the DOF change, before (a) and after (b) the change.

3.3.4 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.

The outer boundary of the molecule is constructed by starting from the bottommost region (of the bottommost atom), and traversing the arrangement of spheres, adding regions to the surface as we move along. Each time we reach an arc that connects two intersecting atoms, we move from the spherical arrangement of the current atom to that of the other. For each visited region of the outer boundary we calculate its area and sum the areas to get the total surface area. Later we calculate the void boundaries. This is done by finding the set of exposed regions, and excluding from this set all the regions on the already computed outer boundary. Then we traverse the remaining regions and construct the void boundaries, in the same way that we construct the outer boundary.

The computation of the exposed regions is done as follows. We do a single traversal of the regions of the atom a, and find for each region how many atoms cover it. We start with an arbitrary region which we assume to be exposed (covered by 0 atoms). Whenever we cross an arc to a new region, we determine if we are entering or leaving an intersection circle, and update the number of atoms covering the newly visited region accordingly. During the traversal we maintain a list of the regions covered by the minimum number of atoms (this number is 0 if the initial region is really exposed, and negative if not). After we finish the traversal, this list holds all the exposed regions of a, unless the entire atom is buried within other atoms (which can be determined by checking a single region from this list against the atoms that intersect a to see if any of them cover it).

This construction 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. We have to find the exposed regions only for atoms whose spherical arrangement was modified in the current step. We have to calculate the areas only for exposed regions that were modified since their area was last calculated or exposed regions whose area was never calculated before (and such regions can be found only on modified atoms).

3.3.5 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 was mentioned earlier, 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 case we used the 3D-hash (the structure mentioned in Section 3.2.1) 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. In the static controlled perturbation we perturbed the atom center around its original center in its global frame (Cartesian) coordinates. However, in the dynamic maintenance code we no longer keep track of the coordinates of the atom center in the global frame. We only know its coordinates in its local reference frame. Therefore 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, because the changes performed in the torsion angles add errors to the transformations between the links of the chain, which are caused by the use of floating-point arithmetic to compute sines and cosines of the angle changes. To reduce the error accumulated in the transformation of a single torsion angle, we sum the angle changes of that DOF (since the beginning of the simulation), and each time this angle is changed we compute a single rotation matrix of the sum of all the changes of this angle. This partial solution still does not solve the inaccuracy of the rotation transformations, and errors may still accumulate when we multiply transformations to get the transformations of the higher levels of the CT. In order to keep the kinematic backbone of the protein exact, we intended to use exact arithmetic with arbitrary-precision rational numbers to compute the sines and cosines [12]. However, the use of such a mechanism for our purpose turns out to be very time consuming,[4] and we decided to postpone it until after we optimize this code; it is therefore not used in the experimental results reported below.

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 on the spherical arrangements of these other atoms. Updating these latter circles may cause spherical arrangements that were not affected directly by the DOF change to be modified. However, as we show in Theorem 3.4, the modification of these extra spherical arrangements does not affect the asymptotic bound on the number of circles added or removed, nor the asymptotic bound on the number of modified atoms.

After resolving the degeneracies of the spherical arrangements, 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, but not the intersection circles of this atom with other atoms on the spherical arrangements of these other atoms, since these intersection circles do not move.

Perturbing the pole direction of an atom cannot induce new degeneracies on other atoms, because it does not change the position of any intersection circle. This, however, is not the case when perturbing atom centers. In the static perturbation, we make sure that perturbing an atom center does not induce new degeneracies by testing the new position of the atom center against all the atoms that intersect this atom and were already added (and possibly perturbed). In the dynamic case, we test only atoms that belong to pairs that either were inserted or updated in the last DOF change, or that nearly intersect. As was mentioned earlier, whenever we perturb the center of an atom, we have to update its intersection circles with other atoms on the spherical arrangements of these other atoms. If such an atom was not directly affected by the DOF change, it will not belong to the list of atoms tested for degeneracies. Let us consider such an atom A_i that was not directly affected by the DOF change, but its intersection circle C_{ij} with some perturbed atom A_j was updated. Now we wish to find out which types of degeneracies may occur on A_i. Degeneracies of the spherical arrangement may only be caused by the update of the intersection circle C_{ij} (the only circle that moved on the spherical arrangement of A_i). However, such a degeneracy must involve the perturbed atom A_j, and we already made sure that the new center of A_j induces no degeneracies. This leaves us with possible degeneracies of the trapezoidal decomposition. These degeneracies may occur due to the update of the intersection circle C_{ij}. Therefore we must test for these degeneracies, and possibly change the pole direction of A_i, and re-construct its spherical arrangement from scratch. At this point, no further perturbations are required. We show in the following

[4]In our experiments, using exact arithmetic in the transformations is 300 times slower than using floating-point arithmetic for the same task.

theorem that these extra tests and possible perturbations have no effect on the asymptotic running time of the dynamic perturbation.

Theorem 3.4 *Using the same perturbation parameters δ and ω[5] 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 [38] 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 per perturbed atom, since the size of each spherical arrangement is bounded by a constant. Therefore this reconstruction takes at most $O(p)$ time. In addition to this reconstruction we also update for each perturbed atom the intersection circles of this atom with other atoms on the spherical arrangements of these other atoms. There are at most m such circles for each perturbed atom (recall that m is the maximum number of atoms that intersect any given atom), and since there are at most p perturbed atoms, the total number of such circles is bounded by $mp = O(p)$ (since m is bounded by a constant [37]). Finally, we add the time it takes to test for degeneracies of the trapezoidal decomposition on spherical arrangements that were indirectly updated. Again, the number of these arrangements is bounded by $mp = O(p)$, and for each of them we can find a valid pole direction if necessary in expected constant time and re-construct the spherical arrangement from scratch in constant time. Therefore, the total time it takes to test for degeneracies, perturb the necessary atoms, modify the spherical arrangements affected by these perturbations and possibly perturb them as well, is $O(p)$, which is the same time it takes to update the spherical arrangements without the perturbation. $\square$

[5]Note that Halperin and Shelton [38] calculate two alternative constraints on ω — one constraint for independent pole directions for each atom, and the other for imposing the same pole direction for all the atoms. While they use the latter constraint, we use the former, which is easier to satisfy.

Chapter 4

Dynamic Connectivity

Avoiding the traversal of the spherical arrangements that have not changed requires some more care in terms of identifying connected components of the boundary. The main difficulty is that in general there can be topological changes to the boundary 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 surface area of every boundary component in total time $O(p \log^2 n)$, where p is the number of atoms whose spherical arrangements changed. For that purpose we adapt tools from dynamic maintenance of graph connectivity. In Chapter 6 we show the speedup in practice gained by replacing the naïve connectivity maintenance with the dynamic graph approach.

4.1 The Boundary Graph

We define the *boundary graph* $G = (V, E)$: 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. See Figure 4.1 for an illustration. As the molecule undergoes DOF changes, some regions are modified, some regions are 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.

4.2 The Algorithm

In [42] Holm *et al* present a poly-logarithmic deterministic fully-dynamic algorithm for graph connectivity. Their algorithm maintains a spanning forest of a graph, answers con-

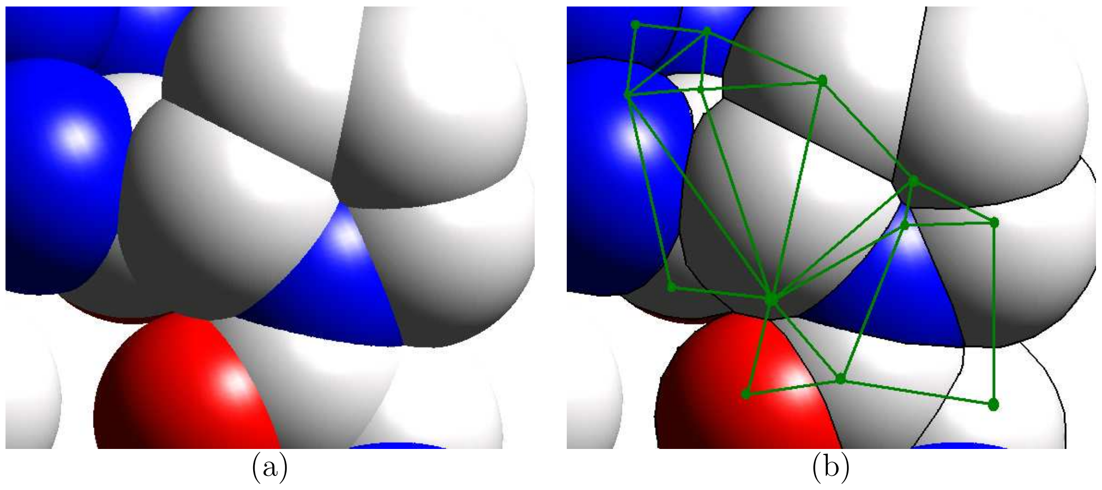


Figure 4.1: A portion of the union of all spheres (a) and the sub-graph induced by it (b).

nectivity queries in $O(\log n)$ time in the worst case[1] 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 $O(n)$ [37, 49], and therefore the algorithm still works with the same amortized time bound. We next describe the original algorithm and our extension of it that efficiently maintains the *surface area* of the boundary of the molecule without traversal of the entire boundary.

The connectivity algorithm in [42] maintains a spanning forest F of the input graph G, and uses for this purpose a data structure called ET-tree. An *ET-tree* is 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(s). 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 Figure 4.2 for an illustration. For more details cf. [41, 42].

The edges of the graph are split into $\ell_{\max} = \lfloor \log_2 n \rfloor$ levels, and a hierarchy $F = F_0 \supseteq F_1 \supseteq \ldots \supseteq F_{\ell_{\max}}$ of spanning forests is maintained, where F_i is the sub-forest of F induced by the edges of level $\geq i$.

[1]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.

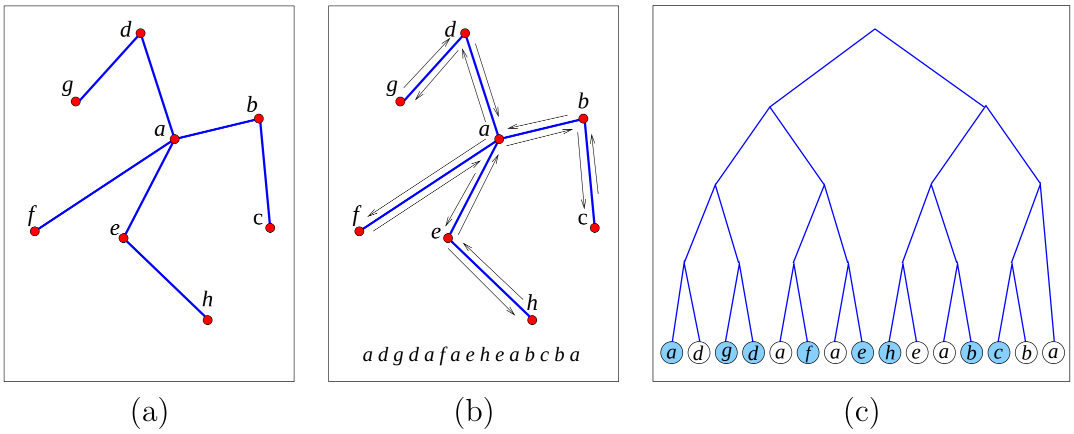


Figure 4.2: A tree (a), an Euler Tour of that tree (b), and the ET-tree of that Euler Tour with its representative occurrences marked (c).

Two invariants are maintained throughout the algorithm :

1. If (v, w) is a non-tree edge, v and w are connected in $F_{\ell(v,w)}$.
2. The maximal number of nodes in a tree (component) of F_i is $\lfloor n/2^i \rfloor$.

Inserting an edge e to the graph is simple — e is given level 0, and if its end-points were not connected, we merge their spanning trees in level 0. Removing a non-tree edge e is also simple (it has no effect on the spanning forests). Removing a tree edge $e = (v, w)$ requires finding a replacement edge, reconnecting the two trees T_v and T_w created by the removal of e. Such an edge can only be found in levels $\leq l(e)$ (due to the invariant 1). The replacement edge is searched recursively in the levels $\leq l(e)$ starting with level $l(e)$. In each level, the edges of the smaller tree (of T_v and T_w) are promoted to the next level. The non-tree edges incident to the vertices of that tree are searched for a replacement edge, and each edge that does not serve for that purpose is also promoted to the next level. The amortization argument of the algorithm is based on increasing the levels of the edges (since the level of each edge can be increased at most $\ell_{\max}$ times).

In [42] each representative node 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 node 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. See Figure 4.3 for an illustration. 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.

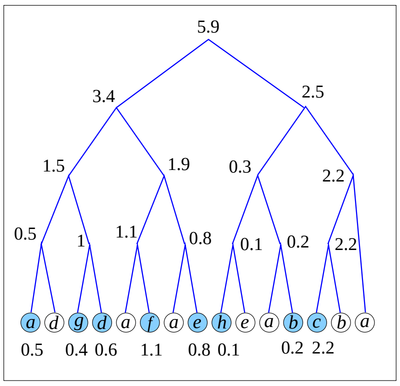


Figure 4.3: An ET-tree with area stored in its representative leaves and area sums in its internal nodes.

To summarize:

Theorem 4.1 *(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 each of the components on whose boundaries it appears in $O(\log n)$ time. $\square$

Chapter 5

Implementation Details

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 [38]. 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 as 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 find a valid direction. 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 [34]. Table 6.1 in Chapter 6 shows the construction times of the surface before and after the modifications mentioned above.

5.2 Building the Protein Chain

Our software reads PDB files [8]. As we read the atoms data from a PDB file, we first identify the backbone and side-chain atoms — for each amino acid, the first four atoms are the N, C_α, C' and O atoms of the backbone and the remaining atoms belong to the side chain. Then we partition the atoms into maximal rigid groups (or links) without DOFs —

each link (except for the first link that contains only the first backbone N atom) consists of either the C_α atom of an amino acid with all the side-chain atoms connected to it or the C' and O backbone atoms of an amino acid with the N backbone atom of the next amino acid.

Then 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 [73]: The origin of each frame (except for the first frame whose origin is the center of the first 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. Figure 5.1 illustrates a short protein backbone with its partition to links and the axes of each reference frame.

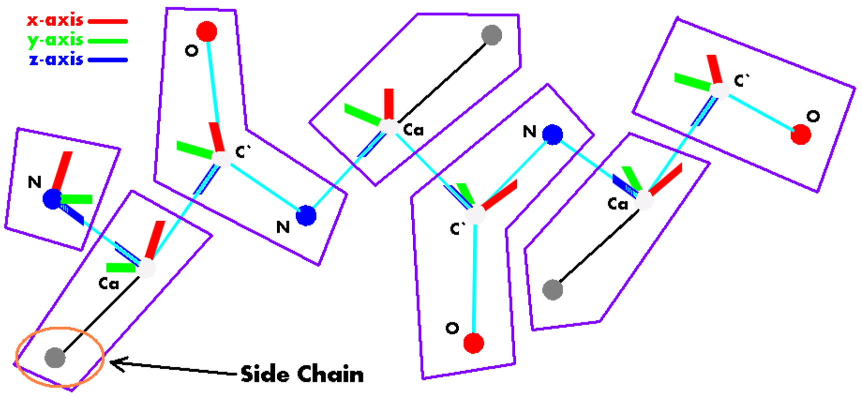


Figure 5.1: A short protein backbone (3 amino acids) with the partition of its atoms to links and the axes of each reference frame.

Once we have a coordinate frame for each rigid link we can compute the transformation from each frame to its following frame, and vice versa. Suppose that the frame at link i is $F_i = \{Q_i; \mathbf{u}_i, \mathbf{v}_i, \mathbf{w}_i\}$ (where Q is the origin and $\mathbf{u}$,$\mathbf{v}$,$\mathbf{w}$ are the axes) and the frame of the next link in the chain is $F_{i+1} = \{Q_{i+1}; \mathbf{u}_{i+1}, \mathbf{v}_{i+1}, \mathbf{w}_{i+1}\}$. Let P be a point in space with coordinates (x_i, y_i, z_i) in frame F_i and coordinates $(x_{i+1}, y_{i+1}, z_{i+1})$ in frame F_{i+1}. Then :

$$\begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{u}_i \cdot \mathbf{u}_{i+1} & \mathbf{u}_i \cdot \mathbf{v}_{i+1} & \mathbf{u}_i \cdot \mathbf{w}_{i+1} & \mathbf{u}_i \cdot (Q_{i+1} - Q_i) \\ \mathbf{v}_i \cdot \mathbf{u}_{i+1} & \mathbf{v}_i \cdot \mathbf{v}_{i+1} & \mathbf{v}_i \cdot \mathbf{w}_{i+1} & \mathbf{v}_i \cdot (Q_{i+1} - Q_i) \\ \mathbf{w}_i \cdot \mathbf{u}_{i+1} & \mathbf{w}_i \cdot \mathbf{v}_{i+1} & \mathbf{w}_i \cdot \mathbf{w}_{i+1} & \mathbf{w}_i \cdot (Q_{i+1} - Q_i) \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \\ 1 \end{pmatrix}. \tag{5.1}$$

This transformation is stored in link i of the chain.

Now we can use a combination of these transformations to compute the local frame coordinates of each atom center. However, it is more efficient to directly calculate for each link the transformation from its frame to the global frame, which is the transformation we get when we substitute $\mathbf{u}_{i+1}, \mathbf{v}_{i+1}, \mathbf{w}_{i+1}$ of Equation 5.1 with the global axes $(1,0,0), (0,1,0), (0,0,1)$ and Q_{i+1} with the global origin $(0,0,0)$. We get :

$$\begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} = \begin{pmatrix} u_{i_x} & u_{i_y} & u_{i_z} & -\mathbf{u}_i \cdot Q_i \\ v_{i_x} & v_{i_y} & v_{i_z} & -\mathbf{v}_i \cdot Q_i \\ w_{i_x} & w_{i_y} & w_{i_z} & -\mathbf{w}_i \cdot Q_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \tag{5.2}$$

Here (x, y, z) are the global coordinates of a point, and (x_i, y_i, z_i) are the coordinates of that point in frame F_i.

After calculating the frame coordinates of all the atom centers, we use only local frame coordinates in any future computations (we only need to compute the global coordinates of the atoms for displaying the molecule).

Whenever the torsion angle between links i and $i+1$ is changed by an angle θ_i, the transformation stored at link i becomes :

$$\begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{u}_i \cdot \mathbf{u}_{i+1} & \mathbf{u}_i \cdot \mathbf{v}_{i+1} & \mathbf{u}_i \cdot \mathbf{w}_{i+1} & \mathbf{u}_i \cdot (Q_{i+1} - Q_i) \\ \mathbf{v}_i \cdot \mathbf{u}_{i+1} & \mathbf{v}_i \cdot \mathbf{v}_{i+1} & \mathbf{v}_i \cdot \mathbf{w}_{i+1} & \mathbf{v}_i \cdot (Q_{i+1} - Q_i) \\ \mathbf{w}_i \cdot \mathbf{u}_{i+1} & \mathbf{w}_i \cdot \mathbf{v}_{i+1} & \mathbf{w}_i \cdot \mathbf{w}_{i+1} & \mathbf{w}_i \cdot (Q_{i+1} - Q_i) \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \\ 1 \end{pmatrix}. \tag{5.3}$$

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. The required transformation is a combination of the transformations stored in the levels of the ChainTree (see Section 3.3.1). It is calculated during the update of the IntersectionsTree, and stored with each pair of intersecting atoms in the Modified Intersections List (see Section 3.3.3). Computing the transformation between any given pair of frames takes $O(\log n)$ time using the ChainTree, but we do not use this property in our algorithm.

5.3 Dynamic Connectivity

Our implementation of the dynamic graph connectivity algorithm is based on the implementation by Iyer, Karger, Rahul and Thorup [43] of the algorithm by Holm *et al* [42].

5.3.1 Creating the Boundary Graph

After the initial construction of the spherical arrangements, we find the exposed regions of each atom. Each such region will be represented by a vertex in our graph. Then for each such region we create an edge from its vertex to the vertices of its adjacent exposed regions. The vertices and the edges are then passed on to the dynamic graph connectivity structure, and the initial spanning forest of the graph is constructed. We maintain a list of the connected components of the graph, for easier access to the outer boundary and voids. Each component is represented by the root of its tree, which holds its surface area.

5.3.2 Updating the Boundary Graph

During each simulation step, we mark all the regions that were modified (regions which are split into smaller regions or merged into larger regions, due to updates of the spherical arrangements). The vertices of these regions will be removed from the graph. After the spherical arrangements are updated, we find the exposed regions of each modified atom and collect the newly created exposed regions. Those regions will be added as vertices to the graph and their areas will be calculated. For each of these vertices we find their adjacent exposed regions and create edges corresponding to the adjacencies.

Next we remove all the vertices of the modified regions and their adjacent edges from the graph. Note that whenever we remove an edge that belongs to a spanning tree of some connected component (a *tree edge*), we search for a replacement edge, and this search is the most costly part of the algorithm. Since each deleted edge is adjacent to some vertex all whose edges are deleted, if we remove those edges in an arbitrary order, the algorithm is likely to replace deleted tree edges with edges about to be deleted, and thus work harder than is necessary. The solution to this problem is simply to first remove all the *non-tree edges* and then remove the tree edges.

The original implementation [43] does not handle deletion of graph vertices. Therefore, whenever we want to delete a vertex, we simply store that vertex in a list of vertices to be recycled. When new vertices will be added to the graph, the recycled vertices will be reused.

After the modified vertices and their adjacent edges are removed from the graph, the new vertices and edges are added. At the end of this addition process, we have a spanning forest of the new graph, and each connected component of this graph holds the area of a boundary component of the molecule.

Whenever we require to find the contribution of an atom to the outer boundary of the molecule and to the voids, we simply go over the exposed regions of the atom, and for each such region find the component it belongs to in $O(\log n)$ time, by finding the root of its tree in the spanning forest.

5.3.3 Heuristics

The implementation by Iyer *et al* has some heuristics that may run faster than the original algorithm of Holm *et al* on certain inputs. These heuristics are aimed to reduce the cost of searching for a replacement edge for a deleted tree-edge: (i) *Sampling*: Searches for a replacement edge within the first s (the *sampling threshold*) non-tree edges of the smaller tree created by the removal of the tree-edge, without promoting any edges. To keep the $O(\log^2 n)$ time of the operation, s should be at most $O(\log n)$. (ii) *Truncating Levels*: At a high level of the hierarchy, where the trees are guaranteed to be small, it is no longer worth doing anything sophisticated. Therefore it may be more efficient to simply check *all* the non-tree edges. For that purpose we choose a *base size* b, and for trees with less than b nodes we perform this simple search. We experimented with various values of s and b and report the results in Chapter 6.

5.4 Overview of the Software

As mentioned above, our software reads a PDB file as input. It also accepts a chain number for that protein, and reads only the atoms of the requested chain from the PDB file. First the software constructs the initial molecular surface, and computes the surface area of its boundary. Figure 5.2 displays a screen shot of the application displaying the van der Waals surface of a molecule and Figure 5.3 displays the spherical arrangements of two selected atoms of the molecule. In each spherical arrangement the arcs of intersection circles are colored red, while the polar arcs of the partial trapezoidal decomposition are colored blue. The atom windows also display statistics on each atom. The window label shows the index of the atom, as well as its element, its amino acid and its role in that amino acid. For example, the atom depicted in Figure 5.3(b) is the backbone Nitrogen atom of an Asparagine amino acid. The north pole picked for the trapezoidal decomposition of each atom is also displayed. For instance, the atom on the right-hand side has the default $(0, 0, 1)$ north pole, while the atom on the left-hand side has a different pole direction, which means that is was perturbed (since before the perturbation, all atoms have the default north pole). The rest of the information shown in the atom window is the number of regions and arcs of the spherical arrangement of the atom, the area and percentage of the atom that belong to the outer boundary of the molecule, and the contribution of the atom to the voids. Figure 5.4 shows the spherical arrangement of another atom of the molecule. This time the arcs are colored based on the contribution of the regions they bound to the boundary. White arcs belong to buried regions. Red arcs belong to regions on the outer boundary. Arcs with other colors belong to regions on void boundaries. In this case, we can see blue arcs that bound a region that belongs to a void. The contribution of the atom to that void is displayed in the atom statistics.

Additional statistics of the entire molecule are displayed in the statistics window, as can be seen in Figure 5.5(a). We can see the surface area of the outer component of the molecule boundary, and some information regarding the size of the arrangements. When the boundary of the molecule contains bounded components (voids), they can be accessed from the statistics window. Figure 5.5(b) displays information regarding voids. For each void we see its surface area, the number of atoms contributing to it and the number of regions on the surface of this void.

After the surface is initially constructed we can perform torsion-angle changes. We can choose a specific DOF and choose an angle change for it (see Figure 5.6(a)), which will be performed if it incurs no self collisions. We can also perform a simulation by choosing the number of iterations, the number of simultaneous DOF changes in each step and the maximum torsion-angle change in each DOF change (see Figure 5.6(b)). During the simulation, the surface area of the molecular surface is updated in the statistics window after each accepted step. After the simulation is finished, a window is displayed, showing the number of accepted and rejected simulation steps.

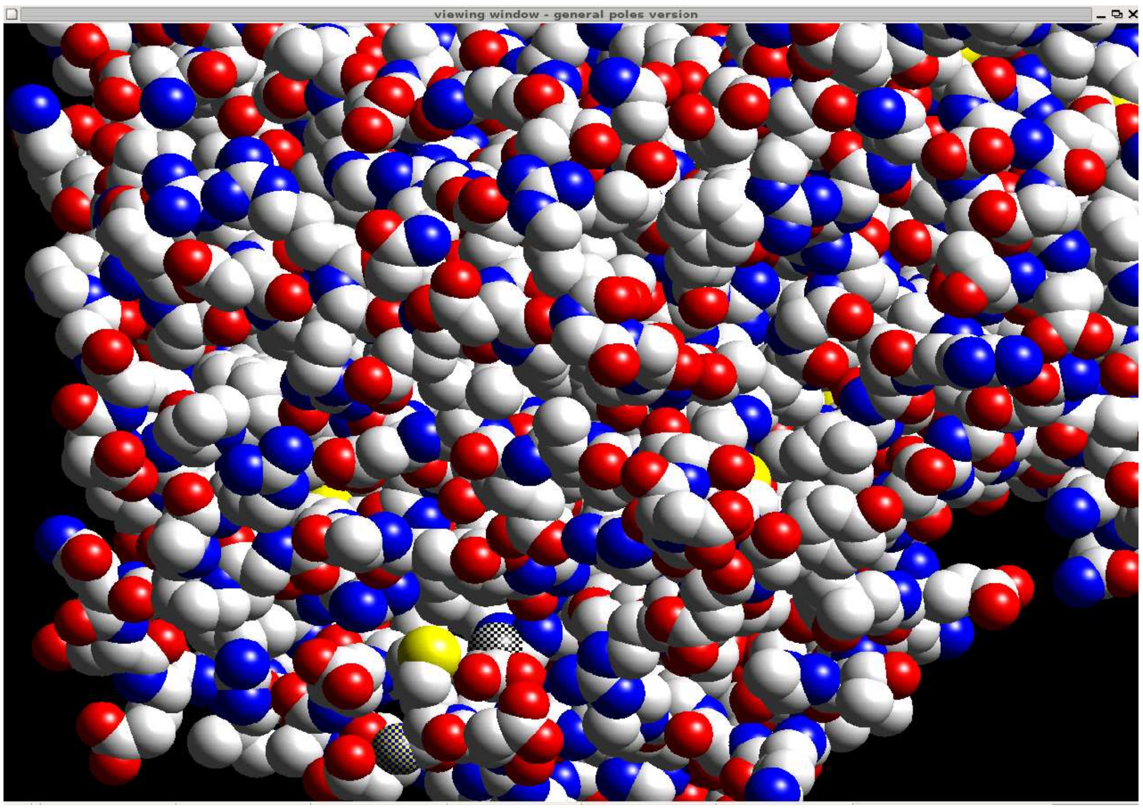
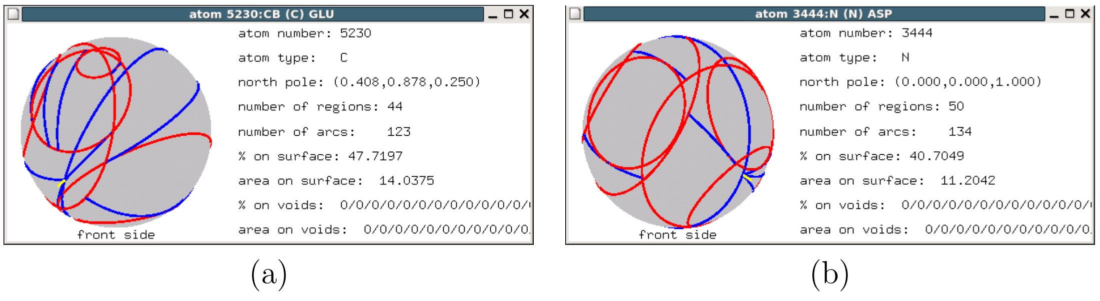


Figure 5.2: A screen shot of the application, displaying the van der Waals surface of the molecule Glutamate Synthase (1EA0). The two checkered atoms in the molecule window have been selected.



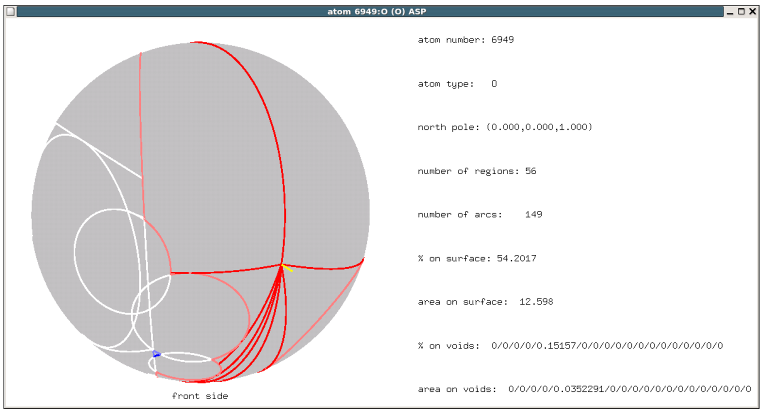
```
atom 5230:CB (C) GLU
atom number: 5230
atom type:   C
north pole: (0.408,0.878,0.250)
number of regions: 44
number of arcs:    123
% on surface: 47.7197
area on surface:  14.0375
% on voids:  0/0/0/0/0/0/0/0/0/0/0/
area on voids:  0/0/0/0/0/0/0/0/0/0/0.
front side
```

(a)

```
atom 3444:N (N) ASP
atom number: 3444
atom type:   N
north pole: (0.000,0.000,1.000)
number of regions: 50
number of arcs:    134
% on surface: 40.7049
area on surface:  11.2042
% on voids:  0/0/0/0/0/0/0/0/0/0/0/
area on voids:  0/0/0/0/0/0/0/0/0/0/0.
front side
```

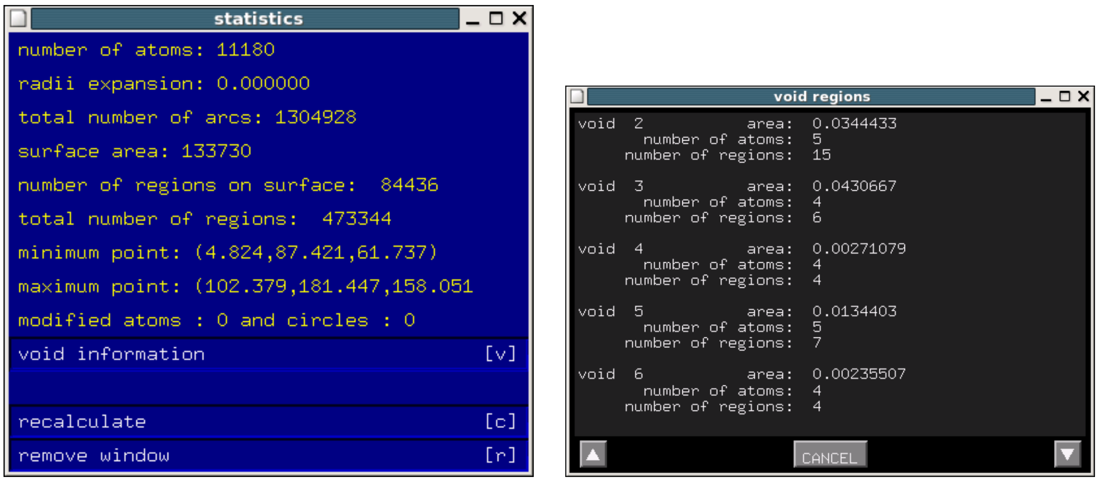
(b)

Figure 5.3: A screen shot of the application, showing the spherical arrangements of the two checkered atoms from Figure 5.2.



```
atom number: 6949

atom type:   O

north pole: (0.000,0.000,1.000)

number of regions: 56

number of arcs:    149

% on surface: 54.2017

area on surface:  12.598

% on voids:  0/0/0/0/0.15157/0/0/0/0/0/0/0/0/0/0/0/0/0

area on voids:  0/0/0/0/0.0352291/0/0/0/0/0/0/0/0/0/0/0/0/0
```

Figure 5.4: A screen shot of the application, showing the spherical arrangement of an atom that contributes to a void boundary.



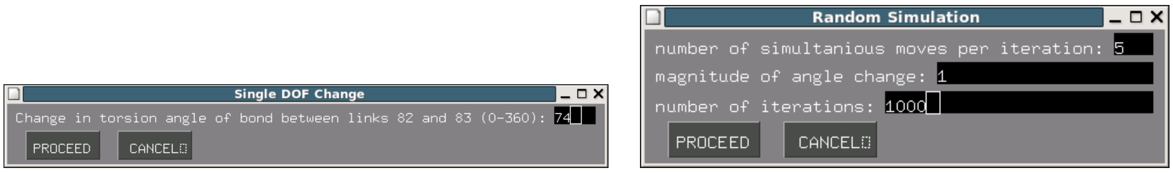
```
number of atoms: 11180
radii expansion: 0.000000
total number of arcs: 1304928
surface area: 133730
number of regions on surface:  84436
total number of regions:  473344
minimum point: (4.824,87.421,61.737)
maximum point: (102.379,181.447,158.051
modified atoms : 0 and circles : 0
void information                          [v]

recalculate                               [c]
remove window                             [r]
```

(a)

```
void  2          area:  0.0344433
      number of atoms:  5
    number of regions:  15

void  3          area:  0.0430667
      number of atoms:  4
    number of regions:  6

void  4          area:  0.00271079
      number of atoms:  4
    number of regions:  4

void  5          area:  0.0134403
      number of atoms:  5
    number of regions:  7

void  6          area:  0.00235507
      number of atoms:  4
    number of regions:  4
```

(b)

Figure 5.5: A screen shot of the application, showing a statistics window (a) and a voids window (b).



```
Change in torsion angle of bond between links 82 and 83 (0-360): 74
```

(a)

```
number of simultanious moves per iteration: 5
magnitude of angle change: 1
number of iterations: 1000
```

(b)

Figure 5.6: A screen shot of the application, showing a single torsion-angle change window (a) and a simulation request window (b).

Chapter 6

Experiments

The experiments described in this chapter 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 $1 - \cos(\omega) = 10^{-9}$.

6.1 Improvements to the Static Construction

Table 6.1 shows the total time it takes to build the static spherical arrangements (including the perturbation time) with the original implementation [38] vs. the improved implementation. The number of atoms that intersect a single atom is denoted by m. The new implementation has other technical improvements in addition to the improvement described in Section 5.1 [34]. As can be seen, the improvements to the original implementation have been a crucial prerequisite to the effectiveness of the dynamic solution.

Table 6.1: 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 6.2: Proteins used in experiments; m is the number of spheres intersecting a single sphere.

Input File	# of Atoms	# of Amino Acids	# of Links	Max m	Mean m
4PTI.pdb	454	58	117	10	5.79
1BZM.pdb	2034	260	521	10	5.74
2GLS.pdb	3636	468	937	13	6.33
1JKY.pdb	5614[1]	748	1497	13	6.24
1KEE.pdb	8181	1058	2117	13	5.87
1EA0.pdb	11180	1452	2905	13	6.14

6.2 Dynamic Maintenance with Naïve Connectivity Algorithm

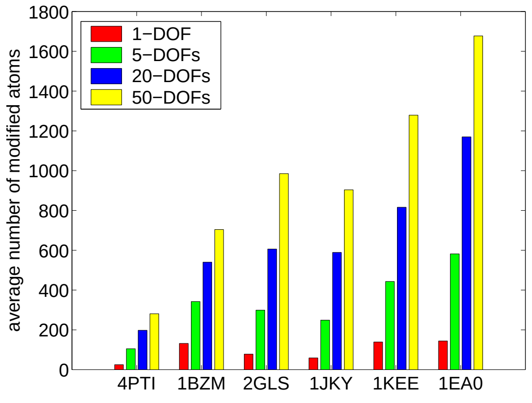
Table 6.2 describes the proteins used in our experiments reported here. In PDB files that contain more than one backbone chain, we handle only the first chain. The number of links is the number of rigid atom groups, which is the number of DOFs plus one, or two times the number of amino acids plus one.

Each simulation consisted of a 1,000 steps. At each step the changed DOFs were chosen 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 the following figures and tables, 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).

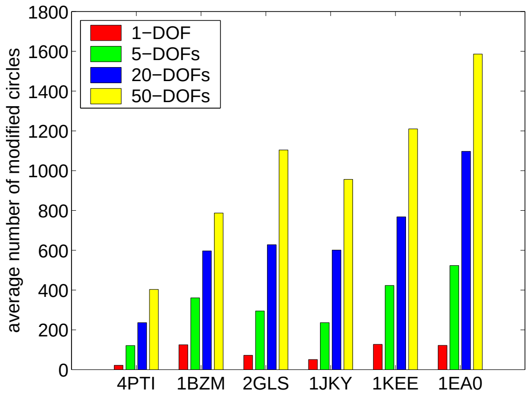
Since the update time of the spherical arrangements depends on the number of modified intersecting circles in each step of the simulation, and the update time of the surface area depends on the number of modified atoms, we tested the relation between the number of simultaneous DOF changes, and the number of modified intersection circles and modified atoms. The results in Figure 6.1 show a strong correlation 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.

To test the effect of the dynamic controlled perturbation on our implementation, we counted the number of intersection circles modified due to the perturbation. In Figure 6.2 we see the average number of intersection circles that were modified due to the dynamic controlled perturbation, compared to the average number of intersection circles modified due to the DOF changes. We see that the number of circles modified due to the perturbation

[1]The number of atoms here is smaller than the number given for the same molecule in Table 6.1, because here we only count the atoms of the first chain of the molecule, whereas in Table 6.1 we count all the atoms of the molecule.



(a)



(b)

Figure 6.1: The average number of modified atoms (a) and average number of modified intersection circles (b) 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.

is about 3%-9% of the total number of modified circles.

In Table 6.3 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 takes to update the CT, the IT, the spherical arrangements and the surface. We made this comparison for several values of simultaneous DOF changes (k). 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 6.3 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.

6.3 Dynamic Connectivity

For each of the input proteins we show in Table 6.4 the initial size of the boundary graph. We can see that both the number of vertices and the number of edges are proportional to n (the number of atoms) which verifies the proof [37] that the complexity of the boundary of a molecule is linear. The ratio between the number of edges and the number of vertices is

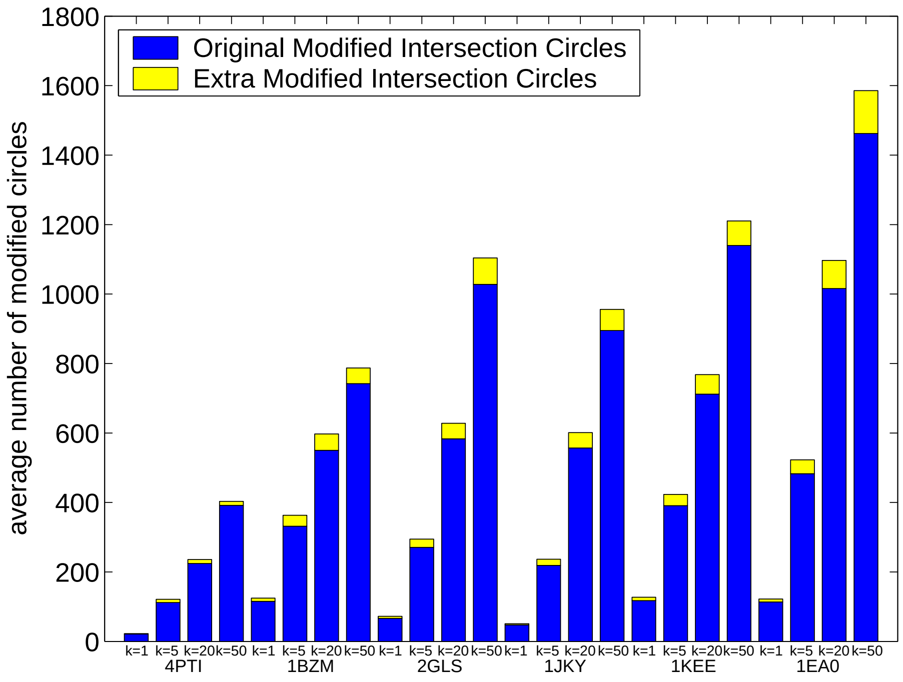


Figure 6.2: The average number of modified intersection circles as a function of the number of simultaneous DOF changes. The original modified intersection circles are the circles modified due to the DOF change. The extra modified intersection circles are the circles modified due to the dynamic controlled perturbation. The average is only over the accepted simulation steps out of a total of a 1,000 steps in each simulation.

Table 6.3: Time (in seconds) of static reconstruction vs. dynamic modification of the surface, using the naïve connectivity algorithm.

Input File	# of Atoms	static	1-DOF	5-DOFs	20-DOFs	50-DOFs
4PTI.pdb	454	1.95	0.11 (5.5%)	0.48 (24.4%)	0.83 (42.6%)	1.32 (67.5%)
1BZM.pdb	2034	8.79	0.61 (7%)	1.49 (16.9%)	2.24 (25.5%)	2.79 (31.7%)
2GLS.pdb	3636	18.25	0.57 (3.1%)	1.45 (7.9%)	2.65 (14.5%)	4.3 (23.5%)
1JKY.pdb	5614	27.31	0.61 (2.3%)	1.43 (5.2%)	2.81 (10.3%)	4.15 (15.2%)
1KEE.pdb	8181	36.48	1.10 (3%)	2.29 (6.3%)	3.51 (9.6%)	4.92 (13.5%)
1EA0.pdb	11180	53.53	1.29 (2.4%)	2.91 (5.4%)	4.79 (8.9%)	6.25 (11.7%)

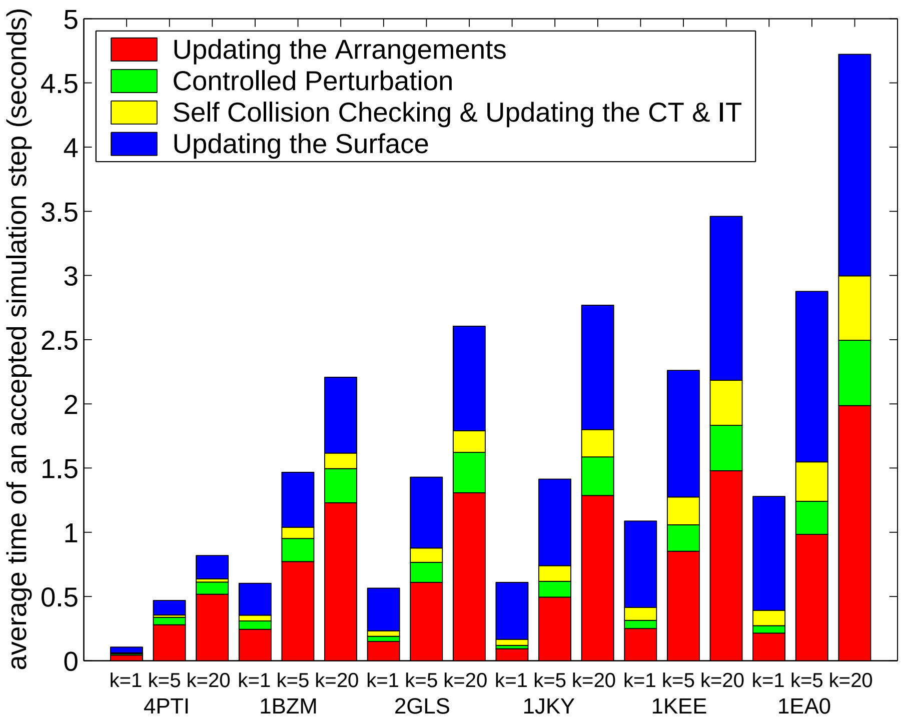


Figure 6.3: Average breakdown of the running time of the main components of our application in a single accepted simulation step for different k values using the naïve algorithm.

Table 6.4: Proteins used in the experiments; the numbers of vertices and edges are in the initial boundary graph (induced by the boundary of the molecule at the original conformation).

Input File	# of Atoms	# of Amino Acids	# of Links	# of Vertices	# of Edges
`4PTI.pdb`	454	58	117	3405	10553
`1BZM.pdb`	2034	260	521	15254	47266
`2GLS.pdb`	3636	468	937	29385	90820
`1JKY.pdb`	5614	748	1497	45558	138818
`1KEE.pdb`	8181	1058	2117	62308	191317
`1EAO.pdb`	11180	1452	2905	84536	260096

Table 6.5: Time (in seconds) of static reconstruction vs. dynamic modification of the surface, using the dynamic connectivity algorithm.

Input File	# of Atoms	static	1-DOF	5-DOFs	20-DOFs
`4PTI.pdb`	454	2.05	0.09 (4.7%)	0.51 (24.9%)	0.92 (44.7%)
`1BZM.pdb`	2034	9.27	0.56 (6%)	1.57 (17%)	2.46 (26.6%)
`2GLS.pdb`	3636	19.27	0.37 (1.9%)	1.39 (7.2%)	2.82 (14.6%)
`1JKY.pdb`	5614	28.91	0.27 (1%)	1.18 (4.1%)	2.81 (9.7%)
`1KEE.pdb`	8181	38.62	0.65 (1.7%)	2.03 (5.3%)	3.55 (9.2%)
`1EAO.pdb`	11180	56.49	0.64 (1.1%)	2.54 (4.5%)	4.95 (8.8%)

similar for all the tested inputs — about 3 — which means that the average degree of each vertex is about 6. Due to the linear bound on the complexity of a molecule boundary, the overall size of the boundary graph remains bounded by $O(n)$ during conformation changes.

Table 6.5 shows the results of the same experiments as Table 6.3, but this time using the dynamic connectivity algorithm. Note that the static construction times are different for the two implementations, since the initial construction of the surface is different (the initial construction of the connectivity graph is slightly slower than the naïve construction of the surface). However, for the dynamic updates of the surface, the dynamic algorithm is faster in most cases.[2] We can see that the dynamic connectivity algorithm works better for small numbers of simultaneous DOF changes, but as the size of the molecules grows, it becomes faster than the naïve algorithm even for larger numbers of simultaneous DOF changes. The dynamic connectivity algorithm runs up to 55% faster compared to the naïve algorithm.

Figure 6.4 shows the fractions of the average running time taken by the main compo-

[2]For 50-DOF simulations the dynamic connectivity algorithm runs a little slower (up to 11%) than the naïve algorithm.

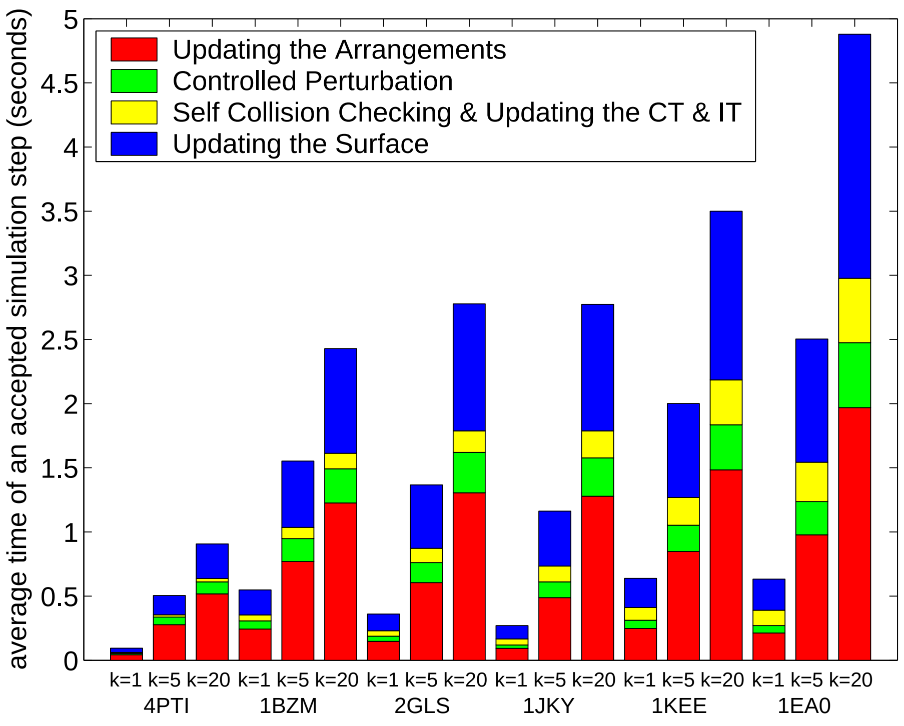


Figure 6.4: Average breakdown of the running time of the main components of our application in a single accepted simulation step for different k values using the dynamic connectivity algorithm.

nents of our application for the dynamic connectivity algorithm. It can be seen that for most inputs updating the surface is faster with the dynamic connectivity algorithm, and takes a smaller fraction of the total time.

We also experimented with the heuristic parameters proposed by Iyer *et al* to the dynamic connectivity algorithm (See Section 5.3.3). We tested different values of s and b (the sampling threshold and the base size).

We reran the experiments of the 1-DOF and 20-DOF simulations. Each was executed in 5 variants of the connectivity algorithm HDT(s,b), where HDT(0,0) is the original algorithm and the other variants use different values of s and b (see Section 5.3.3 for definitions).

Figure 6.5 compares the performance of the different heuristics. It shows the relation between the size of the molecule and the average time it takes to remove a tree edge from the boundary graph (removing a non-tree edge is trivial and does not depend on the heuristic parameters).

The two extreme variants are HDT(0,0), which always promotes edges, and HDT(0,100000), which never promotes any edges (since our largest molecule induces a graph with less than 100000 vertices — see Table 6.4). These two variants turned out to

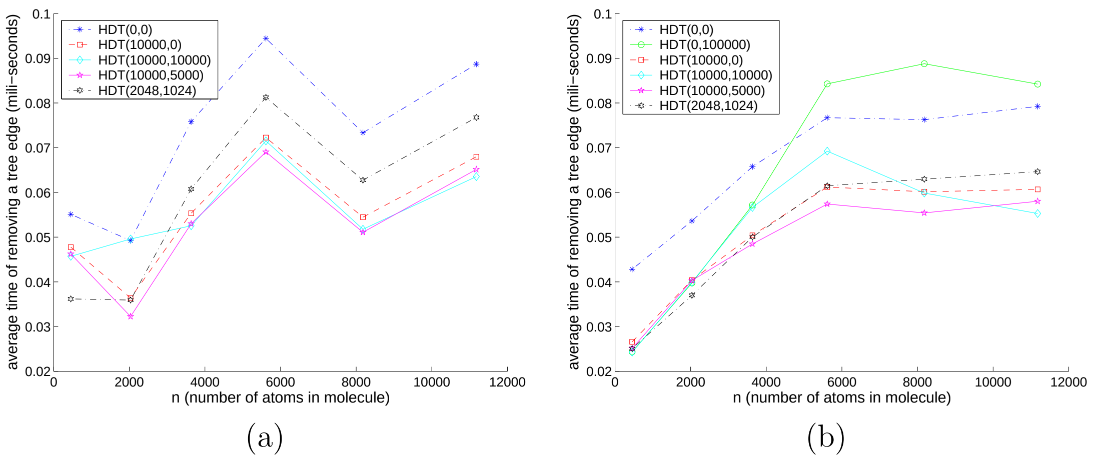


Figure 6.5: The average time of removing a tree edge from the boundary graph as a function of the size of the molecule for 1-DOF simulations (a) and 20-DOF simulations (b).

be the slowest variants.[3] On the other hand, the four other variants, which sample some edges before promoting and perform a simple search on small trees, give better results than the original algorithm. They delete tree edges up to 30% faster for the larger molecules (and even faster for the small molecules).

It is important to note that the points on the graphs of Figure 6.5 result from different molecules. One difference, reflected in the graph, is the size of the molecules, which also affects the size of the boundary graph. However, another difference not reflected in the graph is the number of tree edges deleted in each simulation step. Since the performance of the dynamic connectivity algorithm depends on the number of operations performed (the algorithm better "learns" the structure of the graph when a large number of operations are performed), the average time of tree-edge deletion is expected to become smaller when the number of operations is larger. This explains the unusual behavior of the graph of the 1-DOF simulation — its peak in the middle results from a molecule for which a relatively small number of tree edges were deleted. We do not see this behavior in the graph of the 20-DOF simulation because in this simulation the number of deleted edges is much larger in all the molecules (because the average number of atoms affected by a conformational change — p — is much higher for large numbers of simultaneous DOF changes, which also increases the amount of changes in the boundary graph), and the differences in these numbers are smaller.

The best heuristic on most inputs was HDT(10000,5000), and therefore we tested all our inputs again on that heuristic, and the results can be seen in Table 6.6.

[3]For the 1-DOF simulation, HDT(0,100000) was so slow (two times slower than HDT(0,0)) that plotting it would have compressed the axes in Figure 6.5, making the other variants indiscernible.

Table 6.6: Time (in seconds) of static reconstruction vs. dynamic modification of the surface, using the heuristic dynamic connectivity algorithm HDT(10000,5000).

Input File	# of Atoms	static	1-DOF	5-DOFs	20-DOFs
`4PTI.pdb`	454	2.03	0.09 (4.9%)	0.50 (24.8%)	0.90 (44.5%)
`1BZM.pdb`	2034	9.16	0.54 (5.9%)	1.54 (16.8%)	2.42 (26.4%)
`2GLS.pdb`	3636	19.11	0.36 (1.9%)	1.35 (7.1%)	2.75 (14.4%)
`1JKY.pdb`	5614	28.67	0.27 (0.9%)	1.15 (4%)	2.76 (9.6%)
`1KEE.pdb`	8181	38.27	0.63 (1.6%)	1.97 (5.2%)	3.45 (9%)
`1EAO.pdb`	11180	56.2	0.63 (1.1%)	2.48 (4.4%)	4.82 (8.6%)

Remark: comparison to other software We did not perform an extensive comparison of our implementation to other applications that compute molecular surfaces, since we are not aware of any application that maintains these surfaces during conformational changes. Existing applications that compute molecular surfaces statically do not compute exactly what we compute, which makes comparisons to such applications irrelevant at the moment. We did, however, attempt to compare the alpha shapes method to our work. Since there is no dynamic scheme to update the alpha complex during conformational changes, it is interesting to compare its construction times from scratch to our dynamic update times. We checked a simple implementation that constructs the alpha complex using CGAL [2]. For our largest input, the alpha complex was computed much faster (roughly 7 times faster) than our initial construction of the surface, and since the molecular surface can be quickly constructed from the alpha complex, this implies that the alpha shapes method can be used to compute the molecular surface from scratch faster than our initial construction of the surface from scratch. However, our average update times of the surface in simulations where few DOFs change in each step are much faster (roughly 10 times faster) than the construction time of the alpha complex, which implies that our software would do better in Monte Carlo simulation, where few DOFs change in each step.

Chapter 7

Conclusions and Future Work

We presented an algorithm and its implementation for dynamically maintaining molecular surfaces under conformational changes. It maintains surface information (such as surface area and the contribution of each atom to the surface) that can be useful in simulations such as MCS, in which a small number of DOFs change in each step.

7.1 General Extensions

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 [37] 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. Yet another extension is to calculate and dynamically maintain the volume bounded by the molecular surface (or by parts of it).

7.2 Improvement of the Construction of the Spherical Arrangements

Several modifications can improve the construction and update times of the spherical arrangements. First of all, we can further reduce the number of arcs added by the partial trapezoidal decomposition. Since the sole purpose of this decomposition is to make each face of each spherical arrangement simply connected (we gave up on the four edges boundary limit when we decided not to use the full trapezoidal decomposition), we only have to add polar arcs for faces with holes. A face with holes is created by an intersection circle that does not intersect with any other circles. This means that we need to find the intersection circles that do not contain intersection points of three spheres, which can be found in constant time for each spherical arrangement. Polar arcs will be added only for those special intersection circles, which will reduce the number of arcs and faces of the spherical arrangements, as well as reduce the size of the boundary graph.

Another possible improvement is in the update of the spherical arrangements after a simulation step. Consider three atom spheres S_i,S_j and S_k that intersect each other. Suppose that we change a torsion angle whose axis of rotation is the line between the centers of S_i and S_j, and the third atom S_k belongs to the same rigid link as S_j. Let us assume that the atoms that actually move in this rotation are S_j and S_k. In that case the intersection pattern on the spherical arrangement $\mathcal{A}(C_i)$ changes, due to the movement of the intersection circle of S_k and S_i relative to the fixed position of S_i. However, the intersection pattern on the spherical arrangement $\mathcal{A}(C_k)$ does not change, since the intersection circle of S_k and S_i moves together with the sphere S_k. We can save the time it takes to delete such intersection circles and to re-insert them at the same position.

7.3 Improvement of the Graph Connectivity Algorithm

The graph connectivity algorithm used in our work was designed for general graphs. It may be possible to develop a more efficient algorithm that better suits the graph used in our application, in which all vertices have a degree bounded by a small constant. Our implementation may also be improved by detecting small changes in the molecular surface that do not affect the topology of the graph. Finally, we observe that in a typical scenario of protein motion simulation there is one very big component of the molecular surface (the outer boundary) and several much smaller components (the voids). It would be interesting to use this imbalance of component sizes to improve their maintenance.

Bibliography

[1] AlphaShapes. http://biogeometry.duke.edu/software/alphashapes/.

[2] The CGAL project homepage. http://www.cgal.org.

[3] ProShape. http://biogeometry.duke.edu/software/proshape/.

[4] 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.

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

[6] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids.* Oxford Science Publications, Clarendon Press, Oxford, 1989.

[7] 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.

[8] 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.

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

[10] C. L. Brooks III, M. Karplus, and B. M. Pettitt. Proteins: a theoretical perspective of dynamics, structure and thermodynamics. In I. Prigogine and S. A. Rice, editors, *Advances in Chemical Physics*, volume LXXI. John Wiley and Sons, New York, 1988.

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

[12] J. Canny, B. Donald, and E. K. Ressler. A rational rotation method for robust geometric algorithms. In *ACM Symposium on Computational Geometry*, pages 251–260, Berlin, Germany, 1992.

[13] L. Cavallo, J. Kleinjung, and F. Fraternali. POPS : a fast algorithm for solvent accessible surface areas at atomic and residue level. *Nucleic Acids Research*, 31(13):3364–3366, 2003.

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

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

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

[17] M. L. Connolly. Molecular surfaces: A review. http://www.netsci.org/Science/Compchem/feature14.html, 1996.

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

[19] H. Edelsbrunner. The union of balls and its dual shape. *Discrete Compute. Geom.*, 13:415–440, 1995.

[20] 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.

[21] D. Eisenberg and A. McLachlan. Solvation energy in protein folding and binding. *Nature*, 319:199–203, 1986.

[22] F. Eisenhaber and P. Argos. Improved strategy in analytic surface calculation for molecular systems: handling of singularities and computational efficiency. *J. Comput. Chem*, 14(11):1272–1280, 1993.

[23] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification — a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, 1997.

[24] E. Eyal and D. Halperin. Dynamic maintenance of molecular surfaces under conformational changes. In *Proceedings of the 21st ACM Symposium on Computational Geometry (SoCG)*, pages 45–54, 2005.

[25] E. Eyal and D. Halperin. Improved maintenance of molecular surfaces using dynamic graph connectivity. To appear in Proceedings of the 5th Workshop on Algorithms in Bioinformatics (WABI), 2005.

[26] R. Fraczkiewicz and W. Braun. Exact and efficient analytical calculation of the accessible surface area and their gradient for macromolecules. *J. Comput. Chem.*, 19:319–333, 1998.

[27] F. Fraternali and W. F. van Gunsteren. An efficient mean solvation force model for use in molecular dynamics simulations of proteins in aqueous solution. *Molecular Biology*, 256:939–948, 1996.

[28] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Computing*, 14(4):781–798, 1985.

[29] 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.

[30] V. Gogonea and E. Osawa. An improved algorithm for the analytical computation of solvent-excluded volume. The treatment of singularities in solvent accessible surface area and volume functions. *J. Comput. Chem.*, 16(7):817–842, 1995.

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

[32] P. Güntert, C. Mumenthaler, and K. Wüthrich. Torsion angle dynamics for NMR structure calculation with the new program DYANA. *J. of Molecular Biology*, 273:283–298, 1997.

[33] 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.

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

[35] D. Halperin, J. Latombe, and R. Motwani. Dynamic maintenance of kinematic structures. In J. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 155–170. A K Peters, Wellesley, MA, 1997.

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

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

[38] D. Halperin and C. R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Computational Geometry: Theory and Applications*, 10(4):273–287, 1998.

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

[40] W. Hasel, T. F. Hendrickson, and W. C. Still. A rapid approximation to the solvent accessible surface areas of atoms. *Tetrahedron Computer Methodology*, 1:103–116, 1988.

[41] 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.

[42] 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.

[43] R. Iyer, D. Karger, H. Rahul, and M. Thorup. An experimental study of polylogarithmic, fully dynamic, connectivity algorithms. *J. Exp. Algorithmics*, 6:4, 2001.

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

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

[46] 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.

[47] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam. Analytical shape computation of macromolecules: I. molecular area and volume through alpha shape. *Proteins: Structure, Function, and Genetics*, 33:1–17, 1998.

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

[49] 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.

[50] J. A. McCammon, B. R. Gelin, and M. Karplus. Dynamics of folded proteins. *Nature*, 267:585–590, 1977.

[51] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

[52] 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.

[53] A. Nicholls, K. Sharp, and B. Honig. Protein folding and association: insights from the interfacial and thermodynamic properties of hydrocarbons. *Proteins*, 11(4):281–296, 1991.

[54] 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.

[55] A. Rahman and F. H. Stillinger. Molecular dynamics study of liquid water. *J. Chem. Phys.*, 55:3336–3359, 1971.

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

[57] T. J. Richmond. Solvent accessible surface area and excluded volume in proteins. *J. Mol. Biol.*, 178:63–89, 1984.

[58] B. Roux and T. Simonson. Implicit solvent models. *Biophysical Chemistry*, 78:1–20, 1999.

[59] M. F. Sanner and A. J. Olson. Real time surface reconstruction for moving molecular fragments. In *Pacific Symposium on Biocomputing '97*, volume 2, pages 385–396, Maui, Hawaii, 1997.

[60] 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.

[61] A. Shrake and J. A. Rupley. Environment and exposure to solvent of protein atoms in lyzozyme and insulin. *Molecular Biology*, 79:351–371, 1973.

[62] M. Soss, J. Erickson, and M. H. Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Computational Geometry: Theory and Applications*, 26(3):235–246, 2003.

[63] M. Soss and G. T. Toussaint. Geometric and computational aspects of polymer reconfiguration. *J. Math. Chemistry*, 27:303–318, 2000.

[64] 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.

[65] W. C. Still, A. Tempczyk, R. C. Hawley, and T. F. Hendrikson. Semianalytical treatment of solvation for molecular mechanics and dynamics. *J. Am. Chem. Soc.*, 112:6127–6129, 1990.

[66] A. G. Street and S. L. Mayo. Pairwise calculation of protein solvent-accessible surface areas. *Folding Design*, 3:253–258, 1998.

[67] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 343–350, New York, NY, USA, 2000. ACM Press.

[68] 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.

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

[70] B. Von Freyberg, T. J. Richmond, and W. Braun. Surface-area included in energy refinement of proteins: a comparative study on atomic solvation parameters. *J. Mol. Biol.*, 233:275–292, 1993.

[71] L. Wesson and D. Eisenberg. Atomic solvation parameters applied to molecular dynamics of proteins in solution. *Protein Science*, 1:227–235, 1992.

[72] S. J. Wodak and J. Janin. Analytical approximation to the accessible surface area of proteins. In *Proceedings of the National Academy of Sciences of the USA*, volume 77, pages 1736–1740, 1980.

[73] 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.