Spheres, Molecules, and Hidden Surface Removal*

Dan Halperin[†] Mark H. Overmars[‡]

December 18, 1996

Abstract

We devise techniques to manipulate a collection of loosely interpenetrating spheres in three-dimensional space. Our study is motivated by the representation and manipulation of molecular configurations, modeled by a collection of spheres. We analyze the sphere model and point to its favorable properties that make it more easy to manipulate than an arbitrary collection of spheres. For this special sphere model we present efficient algorithms for computing its union boundary and for hidden surface removal. The efficiency and practicality of our approach are demonstrated by experiments on actual molecule data.

1 Introduction

In this paper we devise techniques to represent and manipulate a collection of overlapping spheres in three-dimensional space. Often, manipulating three-dimensional geometric objects that intersect is time consuming. However, by imposing several constraints on the spheres and their interaction, we obtain a setting where efficient and practical techniques are quite easy to obtain. Our study is motivated by the representation and manipulation of molecular configurations, modeled by a collection of spheres.

A common approach to representing the three-dimensional geometric structure of a molecule, is to represent each of its atoms by a "hard" sphere. In certain applications, it is also assumed that the *nuclear arrangement*, that is, the relative displacement of the spheres, is fixed (it is often the so-called equilibrium nuclear configuration). There are recommended values for the radius of each atom sphere and for the distance between the centers of every pair of spheres. In this model, the spheres are allowed to interpenetrate one another, therefore it is sometimes referred to as the "fused spheres" model (see Figure 1). The envelope surface of the fused spheres may be regarded as a formal *molecular surface*. It is evident that various properties of molecules are disregarded in this simple model. However, in spite of its approximate nature, it has proven useful in many practical applications. For more background material and references, see the survey paper by Mezey [20].

We study the hard sphere model from a *computational geometry* point of view, that is, we study the combinatorial and algorithmic behavior of a collection of n (possibly intersecting) spheres in 3-space, having some special properties. We make several observations showing that, because of these special properties, the spheres in this model can be efficiently manipulated. For example, we show that the maximum *combinatorial complexity* of the boundary of the union of the fused spheres, that is, the overall number of vertices, circular arc edges, and spherical two-dimensional faces on the union boundary, is $O(n)$, whereas for an arbitrary

*Work on this paper by the first author has been supported by a Rothschild Postdoctoral Fellowship, by a grant from the Stanford Integrated Manufacturing Association (SIMA), by NSF/ARPA Grant IRI-9306544, and by NSF Grant CCR-9215219. Work on this paper by the second author has been partially supported by the ESPRIT III Basic Research Action No. 7141 (project ALCOM II) and by the Netherlands Organization for Scientific Research (N.W.O.).
A preliminary version of the paper appeared in *Proc. 10th ACM Symposium on Computational Geometry*, Stony Brook, 1994, pp. 113–122.

[†]Department of Computer Science, Tel Aviv University, Tel Aviv 69978, ISRAEL. Email: danha@math.tau.ac.il. Part of the work on the paper was carried out while this author was at the Robotics Laboratory, Department of Computer Science, Stanford University.

[‡]Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB, Utrecht, the Netherlands. Email: markov@cs.ruu.nl.

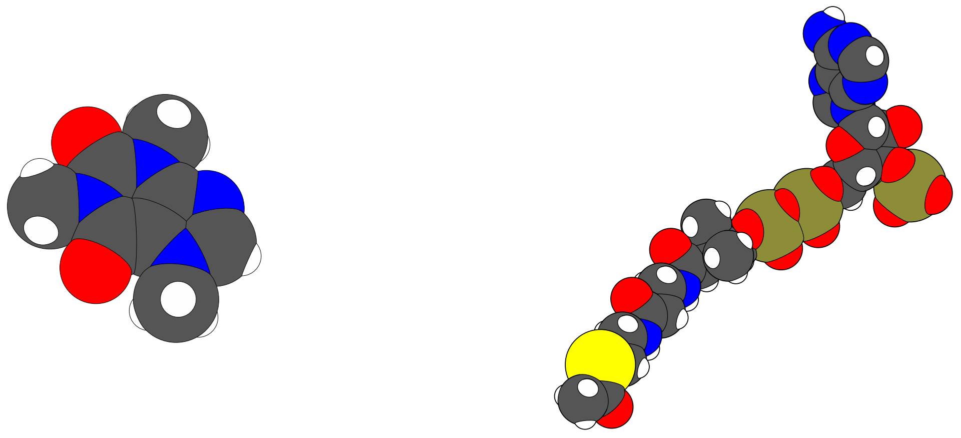
Figure 1: Test molecules caffeine (24 atoms) and acetyl (85 atoms).

collection of spheres (even unit spheres) this may be $\Theta(n^2)$ (see [17]). These results are described in Section 2.

In Section 3 we take advantage of the favorable behavior of the model to devise a data structure for answering *intersection queries* of the form: Given a hard sphere model M of a molecule and a query atom sphere Q (both given in a fixed placement in 3-space), report which spheres of M are intersected by Q. In Section 4 we use this data structure to efficiently compute the boundary of the union of the spheres. We also show that this algorithm can be used to compute a related type of molecular surfaces, the so-called *solvent accessible surfaces*. Finally, in Section 5, we present efficient algorithms for hidden surface removal, either by computing a depth order or by computing the visibility map of a molecule.

We support our claim for efficiency and practicality of the data structure and the algorithms that we present by reporting results of experiments that were carried out on actual molecule data retrieved from different sources, like the "Protein Data Bank" at Brookhaven National Laboratory [2, 3]. In Figures 1 through 3 you find pictures of some of the molecules that we used. All these pictures were produced using the algorithms described in Section 5.

Fast display of molecular models as well as other computational problems related to the geometry of molecules have long been studied by researchers in the areas of computer graphics, molecular biology and computational chemistry. (It is beyond the scope of this paper to give an exhaustive list of references on the subject; there is, for example, a journal—now in its fourteenth year—dedicated to molecular graphics [15].) However, we believe that our work sheds a new light on some of these problems by using tools from computational geometry in the design and analysis of the algorithms, and by providing algorithms that are provably efficient and and the smae moment fast in practice. Several recent publications by other authors [10], [26] also take a rigorous computational approach to problems in this domain.

Our work is closely related to the study of *fatness*. It has been shown that certain problems in computational geometry can be solved much more efficiently when the objects involved have no long and thin parts (see [19, 21, 25]). Clearly, spheres are "fat" in that sense. In 3-space, though, fatness is not enough to guarantee efficient algorithms when the objects are allowed to intersect. Fortunately, the extra properties of the hard sphere model provide sufficient additional constraints on the objects to allow for efficient algorithms. It exemplifies, yet another time, that in practical situations a collection of geometric objects having certain additional properties (obtained from observing real-life situations), behaves favorably.

2 The Hard Sphere Model

Overlapping spheres in three-dimensional space may in general be rather unwieldy objects. The *arrangement* defined by n spheres in 3-space (that is, the subdivision of 3-space into cells of dimensions $0, 1, 2$ and 3,

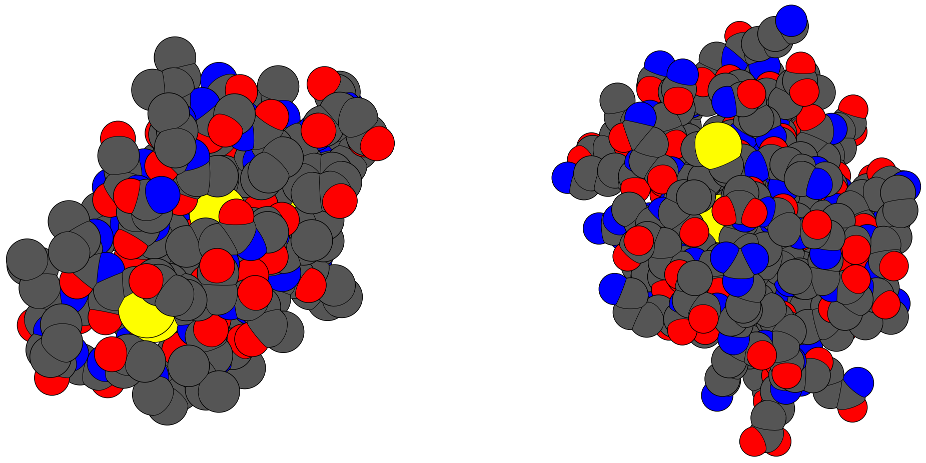

Figure 2: Test molecules crambin (327 atoms) and felix (613 atoms).

induced by the spheres; see [4, pp. 144–150], and below for more details) may have combinatorial complexity $\Theta(n^3)$ in the worst case and their union boundary may have combinatorial complexity $\Theta(n^2)$. Moreover, efficient algorithms for manipulating a collection of overlapping spheres are often complex and rather time consuming. However, the atom spheres in the model of a molecule have properties that we can exploit to obtain efficient and simple algorithms for manipulating them. One such property is that, although two spheres may interpenetrate, their centers cannot get too close to one another. The other useful property is that their radii range in a fairly restricted range; see [14, p. J-3] for a list of van der Waals radii, which are one type of acceptable radii. Actually, each molecular modeling package seems to use its own slightly different set of radii. The following table shows the list the radii (in Ångstrøm) of the spheres we use to represent the main types of atoms (taken from a molecular modeling package called Chem3D plus).

C	Cal	H	N	O	P	S
1.52	3.48	0.70	1.36	1.28	2.18	2.10

In Theorem 2.1 below, we state the conditions that make the sphere model of a molecule favorable. (Similar observations were recently, independently made by Varshney et al. [26].) From this point on, we will refer to *balls* and *spheres* interchangeably. Each atom with radius r_i and center at c_i induces a ball $B_i = \{p | d(p, c_i) \leq r_i\}$ and a sphere $S_i = \{p | d(p, c_i) = r_i\}$, where $d(p_1, p_2)$ is the Euclidean distance between the points p_1 and p_2 in 3-space.

Theorem 2.1 *Let $M = \{B_1, \ldots, B_n\}$ be a collection of n balls in 3-space with radii $r_1, \ldots, r_n$ and centers at $c_1, \ldots, c_n$. Let $r_{\min} = \min_i r_i$ and let $r_{\max} = \max_i r_i$. Also let $S = \{S_1, \ldots, S_n\}$ be the collection of spheres such that S_i is the boundary surface of B_i. If there are positive constants k, ρ such that $\frac{r_{\max}}{r_{\min}} < k$ and for each B_i the ball with radius $\rho \cdot r_i$ and concentric with B_i does not contain the center of any other ball in M (besides c_i), then:*

(i) *For each $B_i \in M$, the maximum number of balls in M that intersect it is bounded by a constant.*

(ii) *The maximum combinatorial complexity of the boundary of the union of the balls in M is $O(n)$.*

Proof. Part (i): Consider a ball B_i. Let B be the ball of radius $r_i + 2r_{\max}$, centered at c_i. Clearly, any ball in M that intersects B_i must lie completely inside B. For each B_j that lies completely inside B, let β_j denote the ball of radius $\frac{\rho}{2} \cdot r_{\min}$ centered at c_j. We claim that the β_j's are pairwise interior-disjoint. Suppose the opposite, namely, that there are two such balls β_{j_1} and β_{j_2} whose interiors intersect. This implies that c_{j_1} and c_{j_2} are less than $\rho \cdot r_{\min}$ apart. This, in turn, means that the ball with radius $\rho \cdot r_{j_1}$ and center at c_{j_1} contains the center of the ball B_{j_2}, contradicting the assumptions of the theorem.

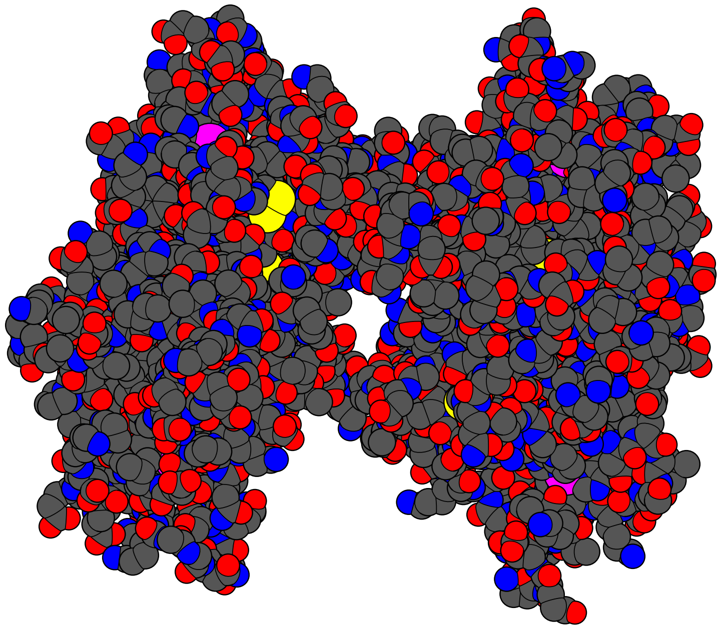

Figure 3: Test molecule SOD (SuperOxide Dismutase) with 4392 atoms.

Hence, by volume consideration, the total number of balls that are completely contained in B cannot exceed

$$\frac{(r_i + 2r_{\max})^3}{(\frac{\rho}{2} \cdot r_{\min})^3} \le \frac{(3r_{\max})^3}{(\frac{\rho}{2} \cdot r_{\min})^3} \le 216 \cdot (\frac{k}{\rho})^3.$$

Part (ii) follows from Part (i) because Part (i) implies that the number of features involving B_i on the union boundary is bounded by a constant. Indeed, if the balls are in general position then a feature involving a ball B_i is either a face on the boundary of B_i, or an intersection arc of B_i with another ball, or an intersection point of B_i with two other balls. All these features belong to the two-dimensional subdivision (or *arrangement*) formed on the boundary of B_i by a constant number of circles, each being the intersection of B_i with another ball. The complexity of this arrangement is evidently bounded by a constant. Thus, the overall complexity of the union boundary is $O(n)$. □

The following table gives the values of k, ρ and the maximal and average number of balls intersecting a single ball for our five example molecules (Figures 1 through 3).

molecule	k	ρ	max	aver.
caffeine	2.17	0.71	10	4.5
acetyl	3.11	0.67	16	5.4
crambin	1.64	0.78	10	5.5
felix	1.64	0.81	9	4.9
SOD	1.95	0.76	16	5.5

As can be seen k is small and ρ is large as required, resulting in a small number of intersections per ball (being much smaller than the worst-case bound of $216(k/\rho)^3$).

We now turn to discuss the subdivision of three-dimensional space induced by a collection of spheres: the *arrangement* of the spheres. The study of arrangements plays a fundamental role in geometric computing, and arrangements arise in the design and analysis of algorithms for a large variety of applications. We denote the arrangement of the spheres in M by $\mathcal{A}(M)$, that is, $\mathcal{A}(M)$ is the subdivision of 3-space into cells of dimensions $0, 1, 2$ and 3, induced by the spheres in M (see [4]). A 0-dimensional cell (a vertex) of the arrangement is the intersection point of three spheres; a 1-dimensional cell (an edge) is a maximal portion of the intersection circle of two spheres not intersecting any other sphere; a 2-dimensional cell (a face) is a maximal portion of one sphere not intersecting any other sphere; and a 3-dimensional cell is a maximal portion of 3-space not intersecting any sphere. The combinatorial complexity (or complexity, for short) of the arrangement $\mathcal{A}(M)$ is defined as the overall number of cells of various dimensions in the arrangement. For arbitrary sets of spheres the complexity of the arrangement can be $\Theta(n^3)$. Since, by Theorem 2.1 (i), each sphere interacts with at most some constant number of other spheres, we have the following:

Corollary 2.2 *The complexity of the arrangement $\mathcal{A}(M)$ of the spheres in M as defined in Theorem 2.1 is $\Theta(n)$.*

For most algorithmic uses, however, a raw arrangement is an unhandy structure. The difficulty is that many cells in an arrangement can have complex topologies, and thus navigating around them is difficult. What we often want is a further refinement of the cells of an arrangement into subcells that are each homeomorphic to a ball and have constant description complexity, ideally keeping the number of subcells in the refinement small.

A prevailing and general technique for decomposing arrangements is the *vertical decomposition* (see [4],[8]), defined as follows. For every sphere S_i, we call the curve of intersection of S_i and the horizontal plane through the center of S_i the *equator* of the sphere. Let E be the collection of curves on the spheres in M consisting of intersection curves between any pair of spheres together with the equators of the spheres. Let γ be a curve in E. We extend a vertical segment upwards and downwards from every point of γ until it hits a sphere in M or extends to infinity. We repeat this process for every curve in E. As a result we get a collection of vertical walls that together with the spheres in M subdivide 3-space into xy-monotone 3D cells. We then project each cell onto the xy-plane, and extend vertical segments (with respect to the y-direction) from every vertex of the projection and from every x-extreme point in the projection of a cell, such that the segments are maximal and contained inside the projection of the cell. We then extend each of these segments into a vertical wall (in the z-direction) contained inside the original 3D cell. For details and illustrations of vertical decompositions for arrangements of spheres in 3-space, see [4].

Clarkson et al. [4] show that the complexity of the vertical decomposition of an arrangement of spheres is dominated by the complexity of the vertical walls erected from curves in E. They show a slightly super-cubic upper bound on the complexity of the vertical decomposition of an arbitrary collection of n spheres in 3-space. In the next theorem we show another favorable property of the collection of spheres that we study:

Theorem 2.3 *The complexity of the vertical decomposition of the arrangement $\mathcal{A}(M)$ for a collection of spheres M as defined in Theorem 2.1 is $O(n^2)$, and this bound is tight in the worst case.*

Proof. We distinguish between two portions of 3-space: inside the union of the balls (that is, the balls corresponding to the spheres in M), and outside the union of the balls. The complexity of the vertical decomposition of $\mathcal{A}(M)$ inside the union of the balls is evidently $\Theta(n)$, by Theorem 2.1 (i). Hence, from this point on we bound the complexity of the vertical walls that lie completely outside the union of the balls. These walls are erected from curves that lie on the union boundary. Recall that the equators of the spheres are also considered curves in the arrangement $\mathcal{A}(M)$.

Fix an edge γ of $\mathcal{A}(M)$ that lies on the union boundary, and let H_γ be the vertical surface which is the union of vertical lines through points on γ. Let F be the collection of faces of $\mathcal{A}(M)$ that lie on the union boundary, where a face is a maximal portion of a sphere in M that does not meet any other sphere or the equator of that sphere, and is not contained in any other sphere. By Theorem 2.1 F consists of $O(n)$ faces (or surface patches), each bounded by a small number (bounded by a constant) of low-degree algebraic curves.

For every face f in F, we denote $f \cap H_\gamma$ by f'. f' is a collection of a small number of low degree algebraic curves on H_γ. By standard arguments, the complexity of the vertical wall extended from γ and that lies outside the union, is determined by the complexity of the lower envelope (or upper envelope, or both) defined by the collection $F' = \{f' | f \in F\}$ with regard to the curve γ. Since the curves F' are either pairwise disjoint, or a pair of curves in F' meet at an endpoint of at least one of the curves, and they do not intersect otherwise, the complexity of the envelope is $O(n)$. We repeat this argument for every edge γ on the union boundary. There are $O(n)$ such edges, and the upper bound follows.

To see that this bound is tight, consider the following configuration of n pairwise disjoint unit spheres in 3-space. Arrange $n/2$ of the spheres one above the other and below the xy-plane such that from the xy-plane one can see the right half of the equator of each sphere. (To be precise, the equator itself is not visible when looking vertically downwards from the xy-plane. However, an arc lying arbitrarily close to the equator, and above it, is visible.) The image seen from the xy-plane downwards is given in Figure 4. Next, arrange $n/2$ of the spheres in a similar way above the xy-plane such that the image when looking upwards from the xy-plane looks the same but rotated over 90 degrees. By carefully placing the two sets we can take care that the visible portions of the equators intersect to create a "grid" on the xy-plane. Each vertex of the grid corresponds to an intersection of two vertical walls outside the union of the spheres, and the complexity of the grid is evidently $\Theta(n^2)$. □

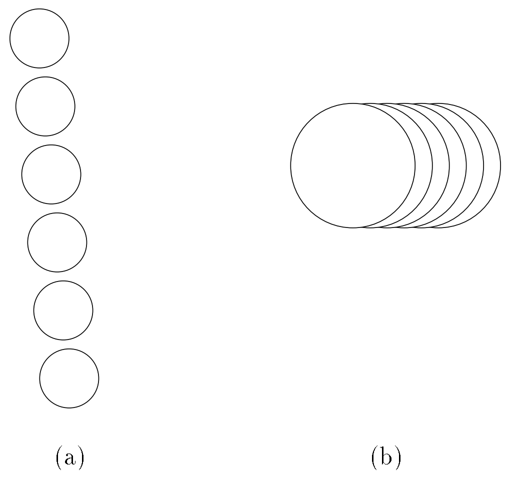

Figure 4: The construction used to create a vertical decomposition with $\Omega(n^2)$ complexity: (a) a projection of the spheres below the xy-plane onto a plane parallel to the z-axis; (b) the image of these spheres seen from the xy-plane

It is however possible to construct an $O(n)$ decomposition of the arrangement $\mathcal{A}(M)$. Consider a three-dimensional grid whose unit size (along each of the coordinate axes) is $2 \times r_{\max}$. The grid induces a partitioning of 3-space into axis parallel unit cubes (those cubes whose vertices are grid points and that do not contain any grid point in their interior). By applying the same type of arguments as in the proof of Theorem 2.1, one can show that every cube in this partitioning intersects at most some fixed number of spheres. Moreover, each sphere intersects at most 8 cubes. Now take the collection of $O(n)$ non-empty cubes. It is easy to see that the vertical decomposition of this set has linear complexity. For each of the cubes consider the arrangement of spheres that intersect it, restricted to the cube and construct its vertical decomposition (again restricted to the cube). This will give a decomposition of the cube into a bounded number of simple cells. As there are $O(n)$ cubes to decompose the total complexity will be linear. It is easy to see that the number of adjacencies between cells is linear as well, which is an important property in certain applications, like motion planning.

Theorem 2.4 *For a collection of spheres M as defined in Theorem 2.1 there exists a decomposition of the arrangement $\mathcal{A}(M)$ into simple cells of total complexity $O(n)$.*

3 Intersection Queries

In order to be able to manipulate molecules efficiently, it is desirable to prepare them for answering certain queries efficiently. In particular intersection queries frequently arise in the algorithmic manipulation of molecules. For example, in computer aided drug design, one wishes to manipulate a molecule in the presence of another molecule to see whether they fit together. During such manipulation the molecules should have only limited interpenetration. Hence, with every change in the manipulated molecule intersection queries with the other molecule must be performed. Also, it is often required to know whether particular positions in 3-space lie inside or outside the molecule, that is, whether a point intersects the molecule or not.

In this section we devise a data structure that can be used to answer intersection queries with either a point or with a ball whose radius is bounded by $r_{\max}$. The query should report all atom balls that contain the point or intersect the query ball. We will use this structure in the next section to compute the molecule's boundary. Clearly a point is a degenerated ball. Hence, a data structure that can answer queries for balls can also answer point queries. Below we only describe a structure for query balls. When implementing the structure though a distinction is made because point queries can be answered faster (that is, with smaller constants in the time bounds).

Our first approach to solving this problem transforms it into an orthogonal range query problem. Let M be the set of n balls (as defined in Theorem 2.1) that we wish to preprocess and let C be the set of centers of the balls. We store the set C in a three-dimensional range tree suitable for answering the following type of query: Given a query axis-parallel box, report the points of C contained in the box. To report all the balls of M intersecting a query ball Q with radius r_Q we construct a ball Q' with radius $r_Q + r_{\max}$ and concentric with Q. Clearly, the center of a ball of M intersecting Q must lie inside Q'. Next, let $\bar{Q}'$ denote the axis-parallel bounding box of Q'. We query the range search structure on C with the box $\bar{Q}'$. Using the same technique as in the proof of Theorem 2.1(i), it can be shown that the number of answers is bounded by a constant. For each of these answers we check whether the corresponding ball actually intersects Q. This takes $O(1)$ time. Using a standard range tree [22, Section 2.3.4] we obtain a solution that requires $O(n \log^2 n)$ preprocessing time and space and answers queries in time $O(\log^2 n)$.

However, due to the special properties of the collection of spheres that we study, they admit a simpler and more efficient data structure. Like in the previous section, we subdivide space into cubes whose side is $2 \times r_{\max}$ long. For each ball in M we compute the grid cubes that it intersects. Let $\mathcal{C}$ be the set of non-empty grid cubes. The size of $\mathcal{C}$ is bounded by $O(n)$. We arrange the cubes of $\mathcal{C}$ in a balanced binary search tree, ordered by the lexicographic order of the (x, y, z) coordinates of the say, bottom-left-front vertex of the cube. With each non-empty cube we store the list of (at most a constant number of) balls of M that intersect it.

Given a query ball Q, we compute all the (at most 8) grid cubes it intersects, and search for each of these cubes in the binary tree, to see whether it is non-empty. If it exists we check the balls stored in it for intersection with Q. We might find some balls more than once but duplicates can easily be removed. The total number of balls tested will be $O(1)$. This leads to the following result:

Theorem 3.1 *Given a collection of n balls M as in Theorem 2.1, one can construct a data structure using $O(n)$ space and $O(n \log n)$ preprocessing time, to answer intersection queries for balls whose radii are not greater than $r_{\max}$, in time $O(\log n)$.*

The binary search tree in the above approach is only used to locate the different cubes. Hence, we can easily replace it by a hashing structure. Using the perfect hashing results in [9] this leads to:

Theorem 3.2 *Given a collection M of n balls as defined in Theorem 2.1, one can construct a data structure using $O(n)$ space, to answer intersection queries for balls whose radii are not greater than $r_{\max}$, in $O(1)$ time. The randomized preprocessing time of the structure is $O(n)$.*

Recently, Fjällström and Petersson [11] have carried out a comparative study of the behavior in practice of various three-dimensional range search structures, motivated by simulation of deformation processes. An interesting result of their study is that range trees do not perform well in practice (for the special test data that they experimented with), as compared to other methods that they examined. In particular the best performing method in their study is a grid-like method, similar to the one mentioned above.

We implemented this data structure using double hashing rather than perfect hashing. The following table shows, for each of the five test molecules: their size, the time to build the structure, the time to answer 1000 point intersection tests and the time to answer 1000 intersection tests for a ball with radius $r_{\max}$. All running times are in seconds, on an Indy R4600 workstation (62.8 SPECint92, 49.9 SPECfp92).

molecule	n	build	1000 point queries	1000 ball queries
caffeine	24	0.00	0.06	0.07
acetyl	85	0.00	0.04	0.05
crambin	327	0.02	0.06	0.09
felix	613	0.05	0.05	0.11
SOD	4392	0.24	0.05	0.11

As can be seen the query time seems independent of the size of the molecule. The time bounds are more related to the density of the molecule. For example, acetyl is very long and thin. As a result, most queries, that are taken within the surrounding cube, for acetyl will be empty. Also, the query time bounds are very low allowing, for example, for testing whether a molecule with 100 atoms intersects a molecule with 4000 atoms in 0.01 seconds, that is, clearly fast enough for interactive use while manipulating the molecule.

4 Computing the Boundary

We present in this section an efficient algorithm for computing the boundary of the union of a collection of loosely interpenetrating spheres. We will show below that this algorithm can easily produce several types of molecular surfaces (the terminology here is borrowed from [5]): the van der Waals surface, Lee and Richards' solvent accessible surface [18], and Richards' smooth molecular surface [23].

Our algorithm uses the data structure of the previous section to efficiently compute the outer cell of the boundary of the union of the spheres. We assume that the union consists of one connected component, which is often the case in our application setting. If the union is composed of several pairwise-disjoint connected components, then additional measures must be taken.

4.1 Molecular Surfaces

The boundary of the outer cell of the collection of atom spheres of a molecule can be viewed as a formal molecular surface, which is of interest in various applications in molecular biology [5], [20]. In [5] this surface is referred to as the *van der Waals surface*. We will show that our algorithm, operating on a closely related set of balls, solves another problem in computational biology, sometimes referred to as computing the *approximate solvent accessible surface* [5],[18],[20]. This involves the interaction of solute and solvent molecules in a solution. In a simplified model, the question may be formulated as follows: Which parts of a molecular surface of a solute molecule are accessible to the solvent molecules, where the latter are modeled by spheres (a single sphere each). We will assume that the solute molecules are modeled by the hard sphere model.

An approach to solving this problem is proposed in [18]: Roll a sphere representing the solvent molecule on a reference surface, to obtain a new surface described by the center of the rolling sphere. The reference surface, in our case, is the boundary of the union of the balls in the hard sphere model. In most cases, the solvent "sphere" is assumed to be fairly small. We denote the solvent molecule sphere by R. (Richards [23] later proposed an alternative approach that yields a smooth surface whose discussion we postpone to Subsection 4.3 below.)

We can rephrase the above problem in terms of *motion planning in robotics*: Let R be a spherical robot moving in 3-space among spherical obstacles (the balls in M). Describe the obstacles in a configuration space where every point represents a possible placement of R by the position of the center of R. In this formulation, the solvent accessible surface becomes the boundary of the outer cell of the free portion of the configuration space. Unlike prevailing formulations of motion-planning problems, the "obstacles" in our case may intersect. However, as expressed in Theorem 2.1, these obstacles have other, favorable, properties.

To compute the boundary of the outer cell of the free configuration space, we follow a common practice in motion planning, and compute the Minkowski (vector) sum[1] of each obstacle and the robot, to obtain the configuration obstacles. Now, the Minkowski sum of two balls is a (larger) ball. Hence, our goal is actually to compute the outer component of the union boundary for a collection of balls in 3-space.

[1] The Minkowski sum of two spatial sets A and B, is the set $\{p+q | p \in A, q \in B\}$.

The solute molecule is modeled by balls as in Theorem 2.1. The radius of each ball in M is increased by r'—the radius of the solvent sphere. We assume r' to be of the same order of magnitude as $r_{\max}$. Note that, the expanded balls obey the conditions of Theorem 2.1 for different constants $r'_{\min}$, $r'_{\max}$, k' and ρ'. Therefore, an algorithm for computing the union boundary for a collection M of balls as in Theorem 2.1, is also applicable to computing solvent accessible surfaces. Thus, we present an algorithm for computing the union boundary of a collection of balls M as defined in Theorem 2.1.

4.2 The Algorithm

The scheme of the algorithm is to compute, for each $B_i \in M$, the contribution of its boundary to the union boundary and then to combine all this information to give the final output of the algorithm. The algorithm proceeds in three steps:

1. For each ball identify the other balls intersecting it.
2. For each ball compute its (potentially null) contribution to the union boundary.
3. Transform the local information into global structures describing the required connected component of the union boundary.

If all we need is a list of the faces of the union boundary (as is the case in the preparatory stage of the hidden surface removal algorithm of Section 5) then we only need Steps 1 and 2.

For Step 1, we use the data structure described in Theorem 3.2. The cost of the query for each ball is $O(1)$, hence the total cost of this step is $O(n)$ randomized time (or $O(n \log n)$ deterministic).

As for Step 2, consider a ball B_i and the family of (a constant number of) balls intersecting it. Let B_j be a ball intersecting B_i. If B_j fully contains B_i, then we stop the process for B_i, as it cannot contribute in any way to the union boundary. If B_j is fully contained in B_i, then we ignore B_j, for obvious reasons. Otherwise, we compute the intersection between the spheres S_i and S_j, which is a circle C_{ij} on S_i. The circle C_{ij} partitions S_i into two parts: One part may appear on the union boundary, and we will refer to it as the *free* part of S_i with respect to S_j, and the other part is completely contained in S_j and therefore cannot appear on the union boundary. We repeat the process for each ball intersecting B_i, to get a collection of circles on S_i. These circles form a 2D arrangement $\mathcal{A}_i$ on S_i. A face of $\mathcal{A}_i$ belongs to the union boundary if and only if it belongs to the free portion defined by each circle C_{ij}. Since the number of circles on S_i defined in that way is bounded by a constant, the arrangement can be computed by a brute force method in $O(1)$ time, together with the attribute whether a face is free or not (a free face is guaranteed to appear on the union boundary). The complexity of the arrangement $\mathcal{A}_i$ is evidently $O(1)$. For each ball in M, the above procedure will take $O(1)$ time.

In Step 3 we represent the outer connected component of the union boundary by a *quad-edge* structure [13]. To this end we have to augment the arrangements $\mathcal{A}_i$ slightly. If $\mathcal{A}_i$ is the whole sphere S_i we split it into two parts with some circle. Next, if a boundary component of $\mathcal{A}_i$ is a simple circle C we split C into two arcs adding two vertices. (If C lies in both $\mathcal{A}_i$ and $\mathcal{A}_j$ the same vertices should be added in both arrangements.) Finally, if a free face of $\mathcal{A}_i$ contains holes we split it by adding extra arcs. (To make these additions canonical, we fix a direction d, and add all the extra arcs along great circles that are intersection of the sphere with planes parallel to the direction d.) After this step, which can easily be performed in time $O(n)$, the union boundary will consist of simple faces where each face is bounded by at least two edges and each edge bounds exactly two faces (assuming general position).

Now we proceed as follows: We search for the ball $B_l \in M$ having the point with largest z-coordinate on its boundary. We then determine the face f_l of B_l containing the highest point. This face clearly belongs to the outer cell. From this face we will scan the entire connected component containing f_l. More generally, we take some free face f of some $\mathcal{A}_i$ (initially f is set to be f_l), that was not treated before, and determine its boundary. In this way we obtain for each edge e bounding f pointers to the previous and next edge along f, as required for the quad-edge structure. Also for each edge e bounding f we determine the arrangement $\mathcal{A}_j$ containing the face f' on the opposite side of f (j can be i because of the extra arcs we added). f' can be found in $O(1)$ time because only $O(1)$ spheres intersect S_i. We locate e in $\mathcal{A}_j$, add pointers between the copies of e and recursively treat f' in $\mathcal{A}_j$ if it was not visited before. In this way we continue until we have located the whole connected component of the boundary containing f_l. It is easy to see that this requires time $O(n)$ in total.

As a result of Steps 2 and 3 we get a quad-edge representation of the connected component of the union boundary of the balls in M containing the topmost point in the z-direction. As for space requirements, the data structure of Theorem 3.2 requires $O(n)$ space. The additional structures described above are easily verified to require linear space.

Theorem 4.1 *The outer portion of the boundary of the union of a connected collection of balls as defined in Theorem 2.1 can be computed in $O(n)$ randomized time (or $O(n \log n)$ deterministic), using $O(n)$ space.*

If the union of the balls is not connected, additional machinery is required to construct the full boundary of the outer cell. This can be done in time $O(n \log n)$ by computing the decomposition of $\mathcal{A}(M)$ as described in Theorem 2.4 and traversing the outer cell of this arrangement. We leave the simple details of this extension to the reader.

We only implemented Steps 1 and 2 of the above algorithm because that is all that is required for hidden surface removal. The following table shows the running time in seconds for our five molecules.

molecule	n	Step 1	Step 2
caffeine	24	0.00	0.05
acetyl	85	0.01	0.16
crambin	327	0.06	0.80
felix	613	0.10	1.10
SOD	4392	0.80	10.7

As can be seen Step 2 dominates the amount of time required, while theoretically both steps take time $O(n)$. It is easy to verify that the operations involved in Step 2 are more complicated and, hence, the constants in the bound are larger.

4.3 Computing a Smooth Molecular Surface

An alternative molecular surface was proposed by Richards [23]: it consists of portions of atom spheres together with portions of the rolling sphere as it is placed in various contacts on the boundary of the hard sphere model. These portions are joined together at circular arcs to yield a smooth surface. Connolly [6] describes a procedure for deriving this so-called analytical surface (to distinguish from an earlier and popular method by Connolly [5] that only produced sample points on this surface).

There are three types of surface patches comprising the smooth surface according to the number of simultaneous contacts that the rolling sphere makes with the solute molecule atoms:

(i) a *convex face* is formed when the rolling sphere R is in contact with only one atom S_i, and it is a maximal connected set of points on S_i that R touches in this manner (there can be several such faces per atom S_i);

(ii) a *saddle face* is created when R is in contact with two spheres. If R is placed in all the possible pairwise contacts of this type (assuming it does not intersect any other atom sphere throughout) then it sweeps a torus. The saddle face is defined to be the union of inward-facing arcs connecting the two points of contact on R in each such placement not hitting any other atom sphere, and the face is any maximal such union.

(iii) a *concave triangle* is created when R simultaneously touches three atom spheres. The three contact points define a spherical triangle on R, whose edges are arcs of great circles on R, and this is the concave triangle.

These three types of surface patches, when glued together in the natural way form a smooth molecular surface. For details see [6].

Remark. This type of molecular surfaces raises a problem of self-intersections [24]. These occur for example when the radius of the torus defining a saddle face is smaller than the radius of the rolling sphere. It is easy to detect self-intersections within the time bounds of our algorithm. Also, it is claimed in [24] that they occur in relatively small numbers. However, they make this type of surfaces not well-defined and we assume below that our algorithm stops when self-intersections have been detected. That is, our algorithm handles this type of surfaces only if they are not self-intersecting and stops and notifies of self-intersections otherwise.

One can easily verify that the three types of patches correspond to faces, edges and vertices of Lee and Richards' solvent accessible surface discussed above, respectively. For example, a vertex of Lee and Richards' solvent accessible surface corresponds to a placement of R where it is in contact with three atom spheres simultaneously, and from which the corresponding concave triangle can be trivially derived. The actual calculation of these objects once the union boundary of the solvent accessible surface as above was computed is straightforward and can be done in constant time per feature. Moreover, the extra arcs (parallel to a fixed direction) added to the arrangement on each sphere make the derivation of the convex faces in Connolly's algorithm even easier than in his description, because each such face is now simply connected. We summarize

Theorem 4.2 *Richards' smooth analytical surface for a molecule with n atoms, modeled by spheres as defined in Theorem 2.1 can be computed in $O(n)$ randomized time (or $O(n \log n)$ deterministic), using $O(n)$ space.*

5 Hidden Surface Removal

One of the tasks of molecular modeling packages is to display molecules. This should preferably be done so fast that the user can interact with the model by turning it around to look at it from different directions or by moving different molecules with respect to each other, to see for example whether they fit together in some nice way (that is, have a certain steric complementarity). Hence, one needs fast algorithms for hidden surface removal among sets of intersecting spheres.

In practice one often uses the *Z-buffer algorithm* for this purpose [12]. Almost all 3-D graphics workstation available nowadays use such an algorithm either in software or in their graphics hardware. Unfortunately, most implementations only handle polyhedral objects. Hence, as a first step, one has to approximate the spheres by triangular meshes. Different methods exist but all lead to a large number of faces. To get a reasonable approximation for a sphere one needs more than 100 triangles. So a molecule of 1000 atoms requires the drawing of 100,000 triangles in 3-space, which makes the display slow.

In what follows, we present two alternative efficient solutions to the hidden surface removal problem for the sphere model, that both eliminate the need for a large Z-buffer. As experiments show (see below) this improves the runtime on workstations without special graphics hardware. An even more important consequence though is that the methods can also be used when drawing molecules on a printer, e.g. a laser printer. This leads to high quality printed images, as the examples in this paper show. Note that in this section we only refer to the boundary of the union of the spheres, and not to Richards' smooth version discussed in Subsection 4.3.

5.1 The Painter's Algorithm

Another approach to hidden surface removal used in computer graphics is the *painter's algorithm.* Here one tries to define a depth order on the objects, sorting them from back to front. Next one draws the objects in this order on top of each other (like a painter) where each new object hides the parts of other objects that lie below it. Such an approach does not require special graphics hardware. The problem with this approach is that it requires a valid depth order on the objects. Such an order does not always exist (there can be cyclic overlap among objects). Also, for intersecting objects such an order obviously does not exist.

For sets of non-intersecting spheres an easy depth order exists: simply sort the spheres by z-coordinate of their center (we assume from this point on that the viewing direction is the negative z-direction). But for intersecting spheres this does not apply. Now, there is no need to draw the entire spheres. It is sufficient to draw the pieces of the spheres that constitute the boundary of the union, as computed in the previous section. Clearly, pieces of the spheres that do not contribute to the boundary cannot be visible. Also, the parts of the sphere where the normal points in the viewing direction cannot be seen. We will show that for the remaining part of the boundary a depth order does exist.

Theorem 5.1 *Let $S = S_1, ..., S_n$ be a collection of spheres sorted by decreasing z-coordinate of their center (that is, from back to front). For each S_i let H_i be the hemisphere facing the viewing direction. Let $H_i^1, ..., H_i^{j_i}$ be the collection of maximal pieces of H_i that are part of the boundary of the union of the spheres. Then*

$$H_1^1, ..., H_1^{j_1}, H_2^1, ..., H_2^{j_2}, ..., H_n^1, ..., H_n^{j_n}$$

is a valid depth order for the pieces on the boundary.

Proof. We will prove this by contradiction. Assume there are two pieces H_i^k and H_j^l that are in the wrong order. Clearly $i \neq j$ so let us assume $i < j$. H_i^k and H_j^l are in the wrong order so there must be a ray in the viewing direction that first hits H_i^k and than H_j^l (so H_i^k partially hides H_j^l). Because H_i^k and H_j^l belong to the union boundary, the ray must first hit H_i^k, then the back of H_i, then H_j^l, and finally the back of H_j. But this implies that the center of H_j lies behind the center of H_i and, hence, $j < i$, which contradicts our assumption. □

This leads to the following algorithm. We first compute the boundary of the union by the algorithm described in the previous section. In this way, for each sphere S_i we collect a (constant) number of pieces that it contributes to the boundary. For each of these pieces we cut off the part that does not lie in H_i. Next we sort the spheres by depth and draw the pieces in this order. As shown in the previous section computing the boundary takes deterministic time $O(n \log n)$ and results in $O(n)$ pieces. The sorting of the spheres by depth takes time $O(n \log n)$. We obtain the following result:

Theorem 5.2 *Given a set of n spheres as defined in Theorem 2.1, a valid depth order consisting of $O(n)$ pieces can be computed in time $O(n \log n)$.*

We implemented this algorithm. We split it into two phases: In the preprocessing phase we compute the boundary of the union and for each sphere we collect the pieces that are part of the boundary. (See the previous section for time bounds.) In the query phase we are given a particular viewing direction. Now for each sphere we compute the hemisphere facing the viewing direction and cut off the parts that lie outside it. Next we compute the depth order for the spheres and draw the pieces. All the molecule pictures in this paper were generated using this approach. The implementation (even though not fully optimized) is rather fast. We ran it on an Indy R4600 machine and compared it with the standard approach using the graphics library of the machine (that uses a highly-optimized Z-buffer algorithm in software). The following table shows the time (in seconds) required per frame for both the Z-buffer algorithm and our method.

molecule	Z-buf	new
caffeine	0.20	0.05
acetyl	0.35	0.15
crambin	1.4	0.6
felix	2.3	1.1
SOD	9.4	5.4

From these figures one might conclude that the new method runs at least twice as fast as the Z-buffer algorithm. This information should be treated with care. First of all, the pictures produced by the two methods are quite different. The way we implemented it, the Z-buffer algorithm produces a Gouraud shaded image while our new method produces a flat shaded image. On the other hand, our method draws boundary lines around the spheres and along the intersections (as in Figures 1 through 3) while the Z-buffer method does not (it cannot because it does not know where the intersections are). Also the picture quality differs largely. On 8-bit systems the Gouraud shaded image looks rather displeasing while the images produced by the new method look much nicer. Also there is the issue of precision. For the Z-buffer we used an approximation of the spheres with about 150 triangles. In the new method we used an approximation of the circles and ellipses (projected sphere boundaries and intersections) with 24 edges, which is better.

So we can conclude that our new approach produces nice images and can be used on simple graphics workstations without 3-D capabilities. What is probably most important though, is that the new method can be used to produce high quality Postscript pictures, which is impossible with the Z-buffer algorithm.

5.2 Visibility Map

In some applications a depth order is insufficient and one needs a combinatorial representation of the visible pieces, the *visibility map.* A lot of work has been done in computational geometry on computing visibility maps. See for example the book by de Berg [7]. For collections of arbitrary intersecting spheres the best known hidden surface removal algorithm runs in time $O(n^{2+\epsilon})$ for any $\epsilon > 0$ (using an algorithm for computing the lower envelope of low-degree algebraic surfaces in 3-space; see [1]). No output-sensitive method, where the time bound depends on the complexity of the resulting visibility map, is known in this case. In the case of a set of non-intersecting spheres the best known method is presented in [16]. It runs in time $O((n+k)\log^2 n)$ where k is the complexity of the visibility map (which can be $\Theta(n^2)$ in the worst case, even for non-intersecting unit spheres). We will now explain how the method of [16] can be adapted to work for intersecting spheres as well. We assume some familiarity with the method of [16].

For the method of [16] to work one needs to subdivide the objects into pieces that first of all have a depth order and, secondly, have the property that the union of the projection on the viewing plane of the pieces lying in any depth range has small complexity. One could use the collection of pieces as defined in Theorem 5.1, but it is unclear whether this collection satisfies the second condition. We define a different set of pieces.

For each sphere S_i we again consider the hemisphere H_i. We take the pieces of H_i that are contained in the boundary of the union of $\{H_1, ..., H_i\}$, that is, we ignore the spheres that lie nearer. Let $G_i^1, ..., G_i^{j_i}$ be the collection of maximal pieces of H_i produced that way. Note that the pieces contain parts that do not belong to the union boundary but one easily verifies that the pieces do not intersect one another.

Lemma 5.3 *The following order*

$$G_1^1, ..., G_1^{j_1}, G_2^1, ..., G_2^{j_2}, ..., G_n^1, ..., G_n^{j_n}$$

is a valid depth order for the pieces G_i^j as defined above.

We omit the proof of this lemma which is similar to the proof of Theorem 5.1.

That these pieces have the second property needed for the algorithm of [16] is less obvious:

Lemma 5.4 *For any contiguous z-range, the union of the projection of the pieces G_i^j defined above, whose spheres' centers lie in the range, is linear in the number of these spheres.*

Proof. Let S' be the set of spheres whose centers lie in some arbitrary contiguous z-range, that is, $S' = \{S_k, S_{k+1}, \ldots, S_l\}$ for $1 \leq k \leq l \leq n$. Let $m \leftarrow |S'| = l - k + 1$. Let $H' = \{H_k, H_{k+1}, \ldots, H_l\}$ be the corresponding collection of hemispheres facing the viewing direction. Our goal is to show that the combinatorial complexity of the boundary of the projection on the viewing plane of the pieces G_i^j on the hemispheres in H' is $O(m)$.

We first consider the full hemispheres in H'. Let $U(H')$ denote the projection of the hemispheres in H' onto the xy-plane, which is the union of a set of disks. The boundary of $U(H')$ clearly has $O(m)$ complexity [17]. The definition of the pieces G_i^j in this case means that we still have to cut out portions of the projections of the hemispheres in H'—those portions that are contained in spheres S_i, with $i < k$. By Theorem 2.1, the number of spheres that can interact with any hemisphere in H' is bounded by a constant. Thus we have to tear off $O(m)$ pieces of $U(H')$. The question now is how this cutting off process may increase the complexity of the boundary of the projection $U(H')$.

Construct a grid of squares of side $2r_{\max}$ in the projection plane. Let Q be a square of this grid. We observe that Q may be affected by at most a constant number of these cuts (where one cut is due to the interaction between two hemispheres: one in H' and the other not in H'). The reason is that all those cuts occur along a restricted z-range of length $2r_{\max}$—spheres farther apart cannot interact. Also the number of non-empty squares Q is bounded by $O(m)$. This implies that the interaction between projected cuts boundaries may increase the complexity of the boundary of $U(H')$ by at most $O(m)$.

We are left to consider the interaction between an original boundary edge e of $U(H')$ that remains after cutting, with an edge of a cut. The projection $\bar{e}$ of such an edge may intersect at most four grid squares. Hence its interaction with cut edges may increase its contribution to the final union size by at most a constant factor. Therefore the overall complexity of the projected union boundary is still $O(m)$. □

Lemmas 5.3 and 5.4 allow for applying the results in [16] to obtain:

Theorem 5.5 *Given a set of n spheres as defined in Theorem 2.1, the visibility map can be computed in time $O((n+k)\log^2 n)$, where k is the complexity of the resulting map.*

6 Conclusion

We have considered the hard sphere model of a molecule in a fixed nuclear configuration from a computational geometry point of view. We have shown that the collection of spheres representing the atoms of a molecule in that model behaves favorably in comparison with an arbitrary collection of spheres in 3-space. Using this observation we were able to devise a data structure that efficiently detects which atom spheres of a molecule are intersected by a query sphere (of bounded radius). Then, we presented an efficient algorithm for computing the boundary (or envelope) of the union of atom spheres. Furthermore, we presented efficient algorithms for hidden surface removal of a molecule. Our results have been supported by experiments carried out on actual molecule data, displaying the efficiency and practicality of our approach.

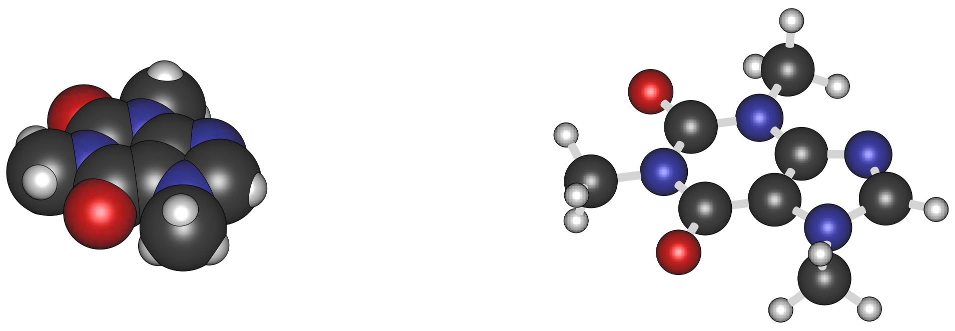

Figure 5: Drawing caffeine shaded or with balls and sticks.

The approach described in this paper can be extended to obtain other pictures of molecules. In particular, it is possible to obtain high-quality shaded pictures, again without using the 3-dimensional graphics hardware. The idea here is that we fill the visible areas of the atoms with the relevant portions of an image of a shaded atom. Because the number of different atom types in a single molecule is small (often less than 6) we can easily precompute such images with a very high quality (e.g., using Phong shading rather than Gouraud shading). Actually, when printing the picture of the molecule, the computation of the atom images can be done in Postscript itself, taking the resolution of the printer into account. See Figure 5 for an example. Another common way of displaying molecules is by using ball-and-stick models, where atoms are represented by small balls and the links between atoms by thin sticks. To extend our approach to drawing ball-and-stick models we have to extend the depth order to include the sticks. Under some mild additional assumptions, that seem to be satisfied by all molecules we checked, such a depth order can indeed be obtained (although it is sometimes required to split the sticks into three parts). See Figure 5 for a resulting image.

We believe that other algorithmic problems in molecular modeling can be solved efficiently using the data structures and properties presented in this paper. The area also provides many other challenging problems, like: Is it possible to preprocess two molecules such that one can efficiently determine whether, in a particular juxtaposition, the molecules intersect or not?

Acknowledgements

The authors wish to thank Ed Moret, Raquel Norel and Haim Wolfson for helpful discussions and for supplying various data and references on molecular structures. We would like to thank Geert-Jan Giezeman who did most of the implementation work.

References

[1] P.K. Agarwal, O. Schwarzkopf and M. Sharir, Overlay of lower envelopes and its applications, *Proc. 11th ACM Symposium on Computational Geometry*, Vancouver, 1995, pp. 182–189.

[2] E.E. Abola, F.C. Bernstein, S.H. Bryant, T.F. Koetzle and J. Weng, Protein data bank, in *Crystallographic Databases: Information Content, Software Systems, Scientific Applications*, F.H. Allen, G. Bergerhoff and R. Seivers, Eds., Data Commission of the International Union of Crystallography, Bonn/Cambridge/Chester, 1987, pp. 107–132.

[3] F.C. Bernstein, T.F. Koetzle, G.J.B. Williams, E.F. Meyer Jr., M.D. Brice, J.R. Rodgers, O. Kennard, T. Shimanouchi and M. Tasumi, The protein data bank: A computer-based archival file for macromolecular structure, *Journal of Molecular Biology* **112** (1977), pp. 535–542.

[4] K.L. Clarkson, H. Edelsbrunner, L.J. Guibas, M. Sharir and E. Welzl, Combinatorial complexity bounds for arrangements of curves and spheres, *Discrete and Computational Geometry* **5** (1990), pp. 99–160.

[5] M.L. Connolly, Solvent-accessible surfaces of proteins and nucleic acids, *Science* **221** (1983), pp. 709–713.

[6] M.L. Connolly, Analytical molecular surface calculation, *Journal of Applied Crystallography* **16** (1983), pp. 548–558.

[7] M. de Berg, *Ray shooting, depth orders and hidden surface removal*, LNCS 703, Springer-Verlag, 1993.

[8] M. de Berg, L.J. Guibas and D. Halperin, Vertical decompositions for triangles in 3-space, *Discrete and Computational Geometry*, **15** (1996), pp. 35–61.

[9] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert and R.E. Tarjan, Dynamic perfect hashing: upper and lower bounds, *Proc. 29th IEEE Symposium on Foundations of Computer Science*, 1988, pp. 524–531.

[10] H. Edelsbrunner, M. Facello, P. Fu, and J. Liang, Measuring proteins and voids in proteins, *Technical Report*, HKUST-CS94-19, Department of Computer Science, Hong Kong University of Science and Technology, 1994.

[11] P.-O. Fjällström and J. Petersson, Evaluation of algorithms for geometrical contact-searching problems, Department of Computer and Information Science, Linköping University, *Manuscript*, 1993.

[12] J.D. Foley, A. van Dam, S.K. Feiner and J.F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, Mass., 1990.

[13] L.J. Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Transactions on Graphics*, **4** (1985), pp. 74–123.

[14] *Handbook of Biochemistry*, H.A. Sober, Editor, The Chemical Rubber Co., 2nd Edition, 1970.

[15] *Journal of Molecular Graphics*, Elsevier, **14** (1996).

[16] M.J. Katz, M H. Overmars and M. Sharir, Efficient hidden surface removal for objects with small union size, *Comp. Geom.: Theory and Appl.*, **2** (1992), pp. 223–234.

[17] K. Kedem, R. Livne, J. Pach and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discrete and Computational Geometry* **1** (1986), pp. 59–71.

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

[19] J. Matoušek, J. Pach, M. Sharir, S. Sifrony and E. Welzl, Fat triangles determine linearly many holes, *SIAM J. Comput.* **23** (1994), pp. 154–169.

[20] P.G. Mezey, Molecular surfaces, in *Reviews in Computational Chemistry*, Vol. I, K.B. Lipkowitz and D.B. Boyd, Eds., VCH Publishers, 1990, pp. 265–294.

[21] M. H. Overmars, Point location in fat subdivisions, *Inform. Proc. Lett.*, **44** (1992), pp. 261–265.

[22] F.P. Preparata and M.I. Shamos, *Computational Geometry—An Introduction*, Springer Verlag, New York, 1985.

[23] F.M. Richards, Areas, volumes, packing, and protein structure, in *Annual Reviews of Biophysics and Bioengineering* **6** (1977), pp. 151–176.

[24] M.F. Sanner, A.J. Olson and J.-C. Spehner, Fast and robust computation of molecular surfaces, *Communication*, in *Proc. 11th ACM Symposium on Computational Geometry*, Vancouver, 1995, pp. C6–C7.

[25] A.F. van der Stappen, D. Halperin and M.H. Overmars, The complexity of the free space for a robot moving amidst fat obstacles, *Comput. Geom.: Theory and Appl.*, **3** (1993), pp. 353–373.

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