Efficient Molecular Surface Generation Using Level-Set Methods

Tolga Can [a], Chao-I Chen [b], Yuan-Fang Wang [b]

[a] *Department of Computer Engineering,*

Middle East Technical University, 06531 Ankara, Turkey

[b] *Department of Computer Science,*

University of California, Santa Barbara, CA 93106, USA

Abstract

Molecules interact through their surface residues. Calculation of the molecular surface of a protein structure is thus an important step for a detailed functional analysis. One of the main considerations in comparing existing methods for molecular surface computations is their speed. Most of the methods that produce satisfying results for small molecules fail to do so for large complexes. In this article, we present a level-set-based approach to compute and visualize a molecular surface at a desired resolution. The emerging level-set methods have been used for computing evolving boundaries in several application areas from fluid mechanics to computer vision. Our method provides a uniform framework for computing solvent-accessible, solvent-excluded surfaces and interior cavities. The computation is carried out very efficiently even for very large molecular complexes with tens of thousands of atoms. We compared our method to some of the most widely used molecular visualization tools (Swiss-PDBViewer, PyMol, and Chimera) and our results show that we can calculate and display a molecular surface 1.5 to 3.14 times faster on average than all three of the compared programs. Furthermore, we demonstrate that our method

is able to detect all of the interior inaccessible cavities that can accommodate one or more water molecules.

Key words: molecular surface, surface generation, cavity, visualization, level sets

1 Introduction

Interactions between molecules are usually induced by the properties of their surface components. Sequences may diverge and secondary structure arrangements may change topology in the evolutionary process. However, surface properties that are essential to protein function are usually conserved. Therefore, calculation and analysis of molecular surfaces play a main role in discovering the functional properties of a protein.

Three main molecular surface definitions exist in the literature[1]. Figure 1 shows an illustration of these definitions in 2D. The *van der Waals* surface is the surface area of the volume formed by placing van der Waals spheres at the center of each atom in a molecule. The *solvent-accessible surface*[3] is formed by rolling a solvent, or a *probe*, sphere over the van der Waals surface. The trajectory of the *center* of the solvent sphere defines the solvent-accessible surface. Whereas, the *solvent-excluded surface* is defined as the trajectory of the *boundary* of the solvent sphere in contact with the van der Waals surface. The solvent-excluded surface is also referred to as the *molecular surface*. As shown in Figure 1, the molecular surfaces can be further decomposed into the contact surfaces that are in common (or in contact) with the van der Waals

Email addresses: tcan@ceng.metu.edu.tr (Tolga Can), cichen@cs.ucsb.edu (Chao-I Chen), yfwang@cs.ucsb.edu (Yuan-Fang Wang).

surfaces and the re-entrant surfaces that are not. The molecular surface and the solvent-accessible surface are the most commonly used representations for both graphical visualizations and quantitative calculations of the surface area[1].

Protein molecules are usually well packed. In fact, the packing efficiency of atoms inside proteins is roughly that for the close packing of hard spheres[1]. Hubbard and Argos[2] analyzed internal packing defects or *cavities* (both empty and water-containing) within protein structures and defined three cavity classes: within domains, between domains, and between protein subunits. These cavities may have several important functions. Takano *et al.*[4] showed that buried water molecules in internal cavities contribute to protein stability. Water-filled cavities also play the role of modulating pK_a values of acidic and basic residues surrounding the cavities[5]. Therefore, in the absence of high-resolution structural data capable of resolving all the water molecules inside protein cavities, it is extremely useful to develop accurate and fast computational methods for quantitatively calculating the shapes and sizes of these cavities. The proposed technique addresses both the surface generation and cavity detection problems.

Related work. Numerous methods have been developed to compute molecular surfaces. Here, we describe some of these methods, and the interested readers are referred to Ref. [1] for a more thorough review of the area. One of the earliest algorithms was proposed by Connolly [6,7]. A molecular dot surface is formed as a combination of convex, toroidal, or concave patches when a probe sphere is tangent to one, two, or three atoms respectively.

A grid-based algorithm was described by Nicholls *et al.*[8] and used in the

program *GRASP*[9]. The method we propose in this article is similar to their algorithm in the processing steps, but differs significantly in the details. In particular, we use the level-set front-propagation method[10,11] to identify van der Waals, solvent-accessible, re-entrant, and contact surfaces in a unified and efficient framework. Our method is applicable and efficient regardless of the molecules are densely or sparsely packed.

Furthermore, we use the same level-set method for identifying enclosed cavities. Nicholls *et al.*[8] detect the cavities by choosing a seed point at an extrema that does not belong to a cavity. All points associated with it, those which can be reached by traveling along triangle edges, are deemed the *noncavity surface*. All others belong to cavities. Note that this will give an incorrect assessment if there is more than one disconnected surface. Unlike *GRASP*, our method can handle such topologies naturally without any effort. [1]

Sanner *et al.*[12] developed a method that relies on the reduced surfaces for computing the molecular surfaces. The reduced surface corresponds to the alpha shape[13] for that molecule with a probe radius α. An implementation of this method (MSMS package) is used by the UCSF Chimera[14] molecular graphics program, which is one of the programs that we compared our method to.

All described methods work well for small molecules. However, one often needs to analyze large protein/DNA, protein/RNA, or protein/protein complexes as the interaction of proteins with other macromolecules is essential for many cellular functions. Therefore, development of a method that is capable of in-

[1] Unfortunately, *GRASP* is currently available only on SGI machines, and is not among the programs that we used for experimental comparison.

teractively analyzing and visualizing the molecular surfaces of large complexes is very important.

An overview of our method. We use a level-set-based approach to compute the molecular surface of a protein of known structure. Our method, which we name LSMS: **L**evel **S**et method for **M**olecular **S**urface generation, proceeds in three stages:

• An *outward* propagation step that generates the van der Waals surface and the solvent-accessible surface.

• An *inward* propagation step that generates the re-entrant surfaces and contact surfaces, i.e., the solvent excluded or the molecular surface.

• Another *inward* propagation step to distinguish between the outer surface and interior cavities of the molecule.

The novelty of our algorithm is three-fold: First, we propose a unified framework for solving all the tasks above based on the level-set front-propagation method; second, our algorithm traverses each grid cell *at most once* and never visits grid cells that are outside the sought-after surfaces to guarantee efficiency; and third, our algorithm correctly detects interior cavities for all kinds of protein topologies, while *GRASP* fails for some.

The volume and area calculations of the molecular surface as well as the internal inaccessible cavities is then carried out very efficiently on the processed grid. For visualizing the molecular surface, a triangular mesh is generated using the marching cubes method[15].

We evaluated LSMS for generating molecular surfaces of very large molecular

structures having 27375 (PDB id: 1a8r) to 97872 (PDB id: 1hto) atoms. We compared our results to PyMol[16], Chimera[14], and Swiss-PDBViewer[17] and our results show that LSMS is faster than all of those tools. We also performed experiments to evaluate the extent of LSMS's interior cavity detection capabilities. We compare the internal cavities found by LSMS to the cavities found by Swiss-PDBViewer. Our results show that LSMS can find all the cavities that can accommodate one or more water molecules. Hence, our technique makes two significant contributions: (1) time and memory efficient mechanisms for computing and visualizing molecular surfaces and (2) accurate determination of interior cavities.

2 Methods

The inputs to our method are the atomic coordinates of the three dimensional molecular structure as a PDB[18] file. We ignore the hydrogen atoms during the surface computation and employ the commonly used *united atom* approach[1]. In this approach, the size of an atom is enlarged by accounting for its hydrogens. We use the same united atom radii applied in Rasmol's[19] spacefill rendering [2]. However, it should be noted that the united atom assumption is not an essential element of our algorithm — our algorithm works without modification with or without the assumption. We made the assumption to be consistent with the existing techniques for comparison purposes.

The molecular structure is then placed and centered on a three-dimensional orthogonal grid of a desired resolution. The size and the resolution of the

[2] http://www.umass.edu/microbio/rasmol/rasbonds.htm

grid are the same along all three dimensions and are held constant during the molecular surface computations. The resolution of the grid with respect to the size of the molecule defines a quality measure that directly corresponds to the quality measure employed by Swiss-PDBViewer[17], that is, the number of grid cells per 1.4 Å. We resize the molecule uniformly in all dimensions so that it fits completely inside the cubic grid. The quality is therefore given by $(N/L) \times 1.4$, where N is the number of grid cells and L is the length of the molecule (in Å) along the major axis.

A molecular surface for the input structure is computed in three stages. Firstly, we use an *outward* marching front to locate the van der Waals and the solvent-accessible surface of the molecule. Secondly, we use an *inward* marching front to compute the re-entrant surfaces and contact surfaces, i.e., the solvent-excluded surface. At the end of the second stage, the grid cells outside the solvent-excluded surface are readily identified as those not processed by our algorithm. However, the cavities inside the molecule are also marked as outside cells. The surfaces surrounding those cavities are not distinguished from the outside molecular surfaces that are accessible to solvent molecules. Therefore, we use the level-set method to distinguish the outer surface from the interior cavities, by shrinking a surface that initially encloses all of the molecule.

An *outward* marching front propagation to find van der Waals and solvent-accessible surfaces. The first stage in the molecular surface computation consists of locating the van der Waals and solvent-accessible surfaces. In the process, we mark the grid cells inside the volume defined by the solvent-accessible (or van der Waals) surface accordingly.

To start with, all grid cells are considered outside the surfaces. To mark the

volume inside the solvent-accessible (or van der Waals) surface, one may traverse all the atoms of the molecule and mark the cells, whose centers fall inside the volume defined by the solvent-accessible (or van der Waals) radii around the atoms, as inside. Figure 2 illustrates the process in two dimensions for one of the atoms of the molecule. The extension to three dimensions involves spheres instead of circles.

Some details deserve careful consideration to ensure efficiency in realizing the procedure described above. There are two general approaches to accomplish inside-outside demarcation: one is *grid-based* and the other is *molecule-based*. In a molecule-based approach, we iterate through the atoms in the molecule and mark the grid cells that each atom occupies. The marking of the grid cells that are inside takes $O(m \cdot k)$ time, where m is the number of atoms and k is the average number of grid cells occupied by an atom. As a final step, all grid cells need to be traversed once to locate the surfaces (a grid cell is on the surface if it is marked inside but is adjacent to some cells that are marked outside). The complexity is $O(N^3)$, where N is the grid resolution along one dimension. Note that this process, though simple to implement, is not efficient. This is because we often visit the same grid cell more than once, since the enlarged van der Waals volumes of the atoms may intersect. This repetition is particularly troublesome when the molecule is densely packed.

The grid-based approach overcomes this repetition by traversing each grid cell exactly *once* and checking if it is inside an atom or not. The brute-force implementation of this technique will take $O(N^3 \cdot m)$ time in the worst case, where each grid cell is checked for intersection with every atom. One can expedite this process by building an octree data structure over the atoms of the molecule. This will reduce the time complexity to $O(N^3 \cdot \log m)$.

However, even with a clever data structure this approach is not efficient, because cells that are outside the surfaces will be examined no matter what. The wasted effort in iterating through cells outside the surfaces is particularly pronounced if a molecule assumes a highly-linear and sparsely-packed 3D structure. In such cases, the embedding grid (or N) will be large but cells will only be sparsely occupied.

Furthermore, each step in the grid-based approach (checking if a cell is occupied or not) is more expensive than the corresponding step in a molecule-based approach (marking the cells that an atom occupies). Our experiments on a 256×256×256 grid showed that even with optimization a grid-based approach can be as much as six times slower than a molecule-based approach. It takes 11.34 seconds to process the protein 1pma (45892 atoms), on a 256×256×256 grid using a grid-based approach; however, it only takes 1.86 seconds when a molecule-based approach is used.

The solution that we propose is novel and different because: *Unlike a molecule-based approach, our technique does not examine a grid cell more than once, and unlike a grid-based approach, our technique does not waste time examining grid cells that are going to be outside the surfaces.* Hence, we combine the best properties of the molecule- and grid-based approaches to achieve efficiency.

Our solution is based on the level-set method. The level-set method[10,11] provides a mathematical framework to compute evolving boundaries. It is based on a continuous formulation represented by partial differential equations and allows one to deform or propagate an implicit surface, which is the zero iso-contour of a scalar (level set) function. The topological changes, e.g., split and merge, are handled naturally by the level-set method. The level-set formula-

tion works in any number of dimensions and the computation can easily be restricted to a narrow band near the zero level set for efficiency.

In more details, imagine a closed surface Γ in space that is propagating normal to itself with a velocity $\mathbf{F}$. One way to characterize the position of this expanding front is to compute the first arrival time $T(x, y, z)$ of the front as it crosses each point (x, y, z) in space. The equation that describes this arrival surface $T(x, y, z)$ is easily derived using the relationship that distance = speed $\times$ time. Hence, we have

$$d\mathbf{x} = \mathbf{F}dT, \quad \text{or}$$

$$|\nabla T||\mathbf{F}| = 1. \tag{1}$$

Thus the front motion is characterized as the solution to a *boundary-value*

problem. [3]

One can intuitively understand the use of level-set front-propagation for molecular surface generation as follows: We initially "seed" a closed surface (a sphere of radius 1 covering just one grid cell) at each and every atom that makes up the molecule. We then evolve (expand) the individual atomic surfaces outward until they reach the specified solvent-accessible (or van der Waals) radii. The radii can be the same or different for different atoms. In the process we mark those grid cells visited as inside. Three important points are discussed below about this procedure:

- First, when multiple atomic surfaces are evolving simultaneously, they often meet and merge — therefore changing the topology of the final solvent-accessible (or van der Waals) surfaces in drastic and unpredictable ways. Luckily, the level-set formulation takes care of surface merging and topology

[3] Conversely, a level-set problem can also be casted as an *initial-value* problem by identifying the position of the propagating front as the zero level set (iso-contour) of a higher dimensional function ϕ[10,11], which satisfies

$$\phi(\mathbf{x}(t), t) = 0.$$

By the chain rule

$$\phi_t + \nabla\phi(\mathbf{x}(t), t)\mathbf{x}'(t) = 0 \quad \text{or}$$

$$\phi_t + |F||\nabla\phi| = 0$$

where $\mathbf{x}'(t) \cdot \mathbf{n} = F$, and $\mathbf{n} = \nabla\phi/|\nabla\phi|$. This representation is more general than the boundary-value formulation. It allows the front to propagate both forward and backward. However, for our application, the boundary-value formulation is more appropriate.

change in a natural way with no undue restriction (details later).

- Second, in our formulation the boundary-value level-set computes the first arrival function $T(x, y, z)$ at each cell (x, y, z). As will be shown shortly, T represents the distance of a grid cell to the nearest atom. This is significant because once a cell is deemed in the solvent-accessible range from a seed atom based on T, it will never be visited later. This implies that each cell will be visited at most *once.* [4]
- Third, once the propagating front reaches the specified distances, an appropriate stopping criterion (detailed later) will be imposed to stop its propagation [5]. Hence, cells that are outside will never be visited to improve efficiency.

Two remaining questions are how to design such an **F** function and how to implement the front propagation efficiently and correctly. Different applications need to come up with suitable definitions of the **F** function based on relevant domain-specific knowledge. For example, in many image processing and edge detection applications, the **F** function represents a curvature-based evolution (expansion and contraction, depending on the sign of the curvature dependency term) with a stopping criterion satisfied at image regions with significant intensity gradient (or presence of edges)[10,11].

In our formulation, the **F** function represents a uniformly expanding surface front that expands one unit distance at a time until the front reaches the

[4] Or one can imagine a propagating surface front as a "wall of flames" that burns through the space. Once a cell is visited by a propagating fire wall, it will stay "burned."

[5] Or one can imagine that the propagating surface front simply runs out of fuel to go beyond the specified van der Waals or solvent-accessible distances.

desired van der Waals or solvent-accessible distance, at which time it stops. We have chosen the two-norm (or Euclidian) distance:

$$\begin{aligned} d[(x_i, y_i, z_i), (x_j, y_j, z_j)] &= \|(x_i, y_i, z_i) - (x_j, y_j, z_j)\| \\ &= \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \end{aligned}$$

which provides the best approximation of a sphere represented by discrete grid cells. Based on the distance metric, the **F** function will be a step function that assumes a value of 1 before the front travels the desired distance and 0 afterward.

To implement the front-propagation framework efficiently and correctly, we adopt a numerical solution scheme that is known as the Fast Marching Method[10,11]. The Fast Marching Method is suitable for boundary-value level-set formulation and can be many times more efficient than the general narrow-band level-set method[10,11]. The central idea is to systematically construct the solution by propagating the front to the nearest grid cells. The key is the observation that the front propagates only one way, that is, from smaller values of T to larger values. Hence, the Fast Marching Method solves Eq. 1 by discretizing it and building the solution *outward* from the smallest T values.

To better explain the central idea behind the efficient implementation of the Fast Marching Method, we use an illustration that is similar to that in Refs. [10,11]. In Figure 3, we depict the initial condition of seeding the propagating surface at one of the atoms in the molecule, which is located at location (x, y, z) in the grid. This provides possible values for T at the six neighboring grid points $T_{x+i,y+j,z+k}$, where $-1 \leq i, j, k \leq 1$ and $i^2 + j^2 + k^2 = 1$.

Now, to propagate the front, we would like to march the front to one of these neighboring points, but we do not know which one to choose. The answer lies in the observation that the smallest T value at these neighbors must be correct, because no point can be affected by grid cells containing larger values of T. Thus, we may freeze the value of T and propagate the front to that closest grid [6].

A pseudo-code algorithm is given below.

Initialization:

(1) NEAR $\leftarrow$ all seed cells, $ID_C = ID_{atom}$, $T_C = -r^{ID_c}_{van\_der\_Waals}, \forall C \in$ NEAR

(2) FAR $\leftarrow$ all other cells

(3) VISITED $\leftarrow \phi$

Loop until NEAR $= \phi$:

(4) VISITING $\leftarrow C_{(x,y,z)} = argmin_{NEAR} T$

(5) NEAR $\leftarrow$ NEAR - $\{C\}$

(6) **if** $T_C < r_{probe}$ then

(7) **if** (x+i,y+j,z+k) $\in$ FAR , $-1 \leq i,j,k \leq 1, i^2+j^2+k^2 = 1$

(8) NEAR $\leftarrow$ NEAR $\cup$ { (x+i,y+j,z+k) }

(9) FAR $\leftarrow$ FAR - { (x+i,y+j,z+k) }

(10) $ID_{x+i,y+j,z+k} \leftarrow ID_C$

(11) $T_{x+i,y+j,z+k} \leftarrow \|(x+i,y+j,z+k) - (x_{ID_c}, y_{ID_c}, z_{ID_c})\| - r^{ID_c}_{van\_der\_Waals}$

(12) **if** (x+i,y+j,z+k) $\in$ NEAR, $-1 \leq i,j,k \leq 1, i^2+j^2+k^2 = 1$

[6] Because we choose T to be a constant, the six neighbors will be reached at the same time and the ordering is less consequential. However, the algorithm is general enough to handle T that is spatially- and directionally-varying, or the front can arrive at the neighbors at different times.

(13) $ID_{x+i,y+j,z+k} \leftarrow argmin(T_{x+i,y+j,z+k}, T_c + 1)$

(14) $T_{x+i,y+j,z+k} \leftarrow min(T_{x+i,y+j,z+k}, T_c + 1)$

(15) `VISITED` $\leftarrow$ `VISITED` $\cup$ { C }

(16) **else**

(17) FAR $\leftarrow$ FAR $\cup$ { C }

Cells are assigned to one of four categories: `NEAR`: near the propagating front, `FAR`: far away from the propagating front, `VISITED`: already visited by the propagating front, and `VISITING`: currently being visited. Initially, `VISITED` is empty, `NEAR` comprises all seed cells, and `FAR` comprises all others. For cells in `NEAR`, we remember the closest atom ID (the seed atom) and the distance to the closest atom. Note that initially a cell marked as `NEAR` is at a distance of zero from the corresponding seed atom. However, in our algorithm we initialize that distance as the negative van der Waals radius of the closest seed atom. This choice of the initial value is important to ensure the correctness of our algorithm. We discuss the rationale of this distance assignment in more detail later.

In each iteration, we select the cell (C) in the `NEAR` list that is closest to the propagating front and mark the cell C as `VISITING` (line 4). We also remove C from the `NEAR` list (line 5). If the T value is still smaller than the probe radius (line 6), which means this cell is still within the solvent-accessible distance, we perform the following operations[7]: We examine the six neighbors of C. For those neighbors that are in the `FAR` list, we move them to the `NEAR` list

[7] If not, the cell lies outside the reach of the current propagating surface front. It will then be reverted back to the `FAR` list (line 17).

(line 8) and remove them from the `FAR` list (line 9). We also assign the nearest atom ID for this neighbor (the same ID as that of C, line 10) and compute the distance of the neighbor to that nearest atom (the Euclidian distance to the atom center, line 11) [8].

For those neighbors that are already in the `NEAR` list, we update their distances to the closest atoms and the IDs of the closest atoms if necessary (that is, if going through C proves shorter than the recorded path so far, lines 13 and 14). Finally, we move C to the `VISITED` list after the processing is done (line 15). As can be seen from the algorithm, cells marked as `VISITED` will never be involved in the processing loop again. This ensures that no cells will be visited more than once (but a cell might be examined more than once).

Validity. The validity of the algorithm—i.e., it marks as inside all grid cells within the solvent-accessible (or van der Waals) distance to a seed atom, but no others—is informally argued below. Three main claims underpin the validity assertion: (1) surface fronts will *never* propagate to the cells *outside* the solvent-accessible distances from the seed atoms, (2) surface fronts will *always* propagate to the cells *inside* the solvent-accessible distance from some seed atom, and (3) the algorithm computes and maintains, at each `NEAR` cell, the correct distance to the nearest seed atom. It should be intuitively clear that if the above three claims are true, then the algorithm is correct.

The first claim is easily verified as line 17 in the algorithm shows that cells are not processed further if the recorded distance to the closest atom is larger

[8] The van der Waals radius of the nearest atom is subtracted from the computed distance to account for the fact that distances are initialized with negative van der Waals radii to ensure correctness.

than the corresponding solvent-accessible distance. Hence, no cells outside the solvent-accessible distances will be processed.

The third claim is also easily verified. When a cell is first brought into the `NEAR` list, its only known path to a seed atom is through cell C, i.e., the cell's neighbor who is currently being visited. The cell's distance to the nearest seed atom is the Euclidian distance to that atom (line 11). If a cell in the `NEAR` list is being examined again (line 12), then a new path to a seed atom must have been found. Line 14 then updates the first arrival time that has been found so far. Hence, the algorithm correctly maintains the first arrival time at a `NEAR` cell.

The second claim deserves some thinking. Often times, a grid cell is within the solvent-accessible distances to multiple seed atoms (because van der Waals or solvent-accessible distances account for sharing of electrons in co-valence bonds). Hence, if a cell, called C, has been visited by the front originated from the closest seed atom and marked as `VISITED`, then another front—from a more distant atom and arriving some time later—will not be able to propagate through cell C, because it has already been visited. Then the question is: Will the late-arriving front still be able to propagate itself to reach all the cells within its solvent-accessible distance without the participation of cell C?

The answer to the above question is actually no. However, we claim that a cell, which lies *within* the solvent-accessible distance to an atom but is *not* reached by the atom's surface front, must have been assigned the `VISITED` label by the surface front originated from some other neighboring atom. Hence, *collectively* the surface fronts from all seed atoms will reach all cells that are within the solvent-accessible distances to some seed atoms.

To prove the above argument, we must first consider how the surface fronts from neighboring atoms meet and thus "quench" the propagation of each other. In our algorithm, the initial distance of a `NEAR` cell is assigned the negative value of the van der Waals radius of the corresponding seed atom (line 1 in our algorithm). This assignment is important because it ensures that *not until all propagating fronts finish filling in every cell inside their own van der Waals distances can the fronts start to propagate toward the solvent-accessible surfaces.* In other words, any atom which has completed filling its van der Waals volume must somehow wait all other atoms to catch up. However, a more elegant solution involves the simulation of this "waiting" effect at the beginning of the propagation process. Instead of starting the propagation of the fronts of all atoms at the same time, each atom may wait until proper time to start propagating its front and thus having all atoms fill their van der Waals volume at the same time. To achieve this goal, we let atoms with larger van der Waals radii start propagating their fronts earlier than atoms with smaller radii. The smaller atoms start propagating their surfaces when the remaining van der Waals distances of the larger atoms are equivalent to the radii of smaller atoms. In the algorithm, this is carried out by initializing the arrival time of surface at atom centers with negative van der Waals radii (line 1) and with cell picking strategy in which the cell with smallest distance is chosen always (line 4). The cell picking strategy forces our algorithm maintain the same propagation speed in all directions, implying that all the cells with shorter distances need to be filled in before any cell with larger distance can be reached. On the other hand, setting the initial distances with negative van der Waals radii is the same as setting the distance on the van der Waals surface to *zero* and every cell outside the van der Waals surface will have a distance larger than zero. Therefore, it becomes clear that the solvent-accessible surfaces will

not be propagated until all cells inside every atoms van der Waals distances have been filled in. Our algorithm is general enough to solve all known possible molecular structures, even the cases that neighboring atoms have some overlap on their van der Waals volume.

As mentioned before, the neighboring atoms meet and quench the propagation of each other. An interesting question on might ask is: What will the shape of this "fire quenching" surface be? To illustrate, consider a simple 2D case of two atoms in Figure 4. Without loss of generality, assume that one atom is centered at $(0, 0)$ and has a van der Waals radius r_A and the other atom is centered at $(x_B, 0)$ and has a van der Waals radius r_B. The "fire quenching" surface thus satisfies the equation

$$\sqrt{x^2 + y^2} - r_A = \sqrt{(x - x_B)^2 + y^2} - r_B \tag{2}$$

When $r_A = r_B$, it is readily shown that the quenching surface is $x = x_B/2$—a plane that passes through the mid point between the two atoms and is perpendicular to the line joining the two atoms. If all atoms have the same van der Waals radius, the union of these quenching surfaces is thus the Voronoi diagram of the seed atoms, which has been widely used for protein surface calculation and generation[1]. When $r_A \neq r_B$, the shape of the quenching surface is a parabola and is biased toward the small atom (see Figure 4). In any case, our choice of starting measuring the propagation distance at the negative van der Waals distance (instead of zero) ensures that Eq. 2 to be true. Eq. 2 states that cells that are closer to the van der Walls surface of an atom will be reached by its own propagating front. Those that are closer to another atom will be reached by the propagating front from the other atom. Collectively, all cells that are within the solvent-accessible distances will then

be reached and marked `VISITED` accordingly.

Complexity. The complexity of the algorithm can be informally analyzed as follows: Lines 1 to 3 represent a constant initialization overhead that is inconsequential. Lines 5 to 17 all can be executed in constant times. This is because in real implementation, it is not necessary to use any complicated data structures for `NEAR`, `FAR`, and `VISITED`. Each cells can be augmented with a 2-bit status flag that represents one of the following four states: `NEAR`, `FAR`, `VISITING`, and `VISITED`. The status flag can be checked and updated in constant time. The only line that deserves some careful implementation is line 4, which selects from the all `NEAR` cells one that is closest to the propagating front to visit next. This selection process can be implemented efficiently using a data structure called a partially ordered tree or a heap[20], which has logarithmic complexity for *insert* and *deletemin* operation. Furthermore, a heap can be implemented using an array (or a `Vector` in Java) without complicated pointer arithmetics.

So the question is how large can the `NEAR` set grow to? Intuitively speaking, the `NEAR` set will grow as the fronts from the seed atoms expand, and then starts to shrink when the expanding fronts merge and reach the solvent-accessible distances. We can readily establish an upper bound of the total surface area of the expanding fronts as follow: The surface area of a sphere (i.e., an atom) is $4\pi r^2$, where r can be either the van der Waals or solvent-accessible radius. When a surface front is expanding, we note that (1) the front maintains roughly a spherical shape. This is because we select the cell to visit based on the distance to the center of the atom. Hence, we cannot introduce big protrusions (too large a distance to explore next) or big depressions (too small a distance *not* to explore next) in the front. In fact, the difference in the dis-

tances to the center of the atom of points on the front can be at most 1 grid cell in our scheme, and (2) the radius is always less than or equal to the final r. Hence, the upper bound of the active surface areas to maintain in `NEAR` is at most $4\pi r^2$ for a single atom, and $4\pi r^2 \cdot m$ for m atoms. [9]

If the size of `NEAR` is $O(m)$, then the complexity of line 4 is $O(\log m)$. The algorithm will then have complexity that is $O(l \cdot \log m)$, where l is the number of cells that are inside the solvent-accessible surfaces. As it is necessary to visit each cell inside the solvent-accessible surfaces once (to mark them as inside), the l term is theoretically minimum and cannot be reduced further no matter what algorithms are used. Our algorithm achieves high efficiency (log) for operations per cell, and hence, theoretically it is better than either molecule- or grid-based alternatives. However, it is worthwhile to note that, experimental performance assessments at different grid resolutions showed that molecule-based determination of the solvent-accessible volume performs as well as our proposed level-set based approach in practice.

An *inward* marching front propagation to find the solvent-excluded (molecular) surfaces. After marking the grid cells inside the volume of the solvent-accessible surface, the next step is to mark out the parts that fall inside the solvent molecule, hence the name, *solvent-excluded surface*. A brute-force implementation will involve finding all the probe spheres centered on the solvent-accessible surface. Then, for each such probe, the grid cells that are inside the probe sphere are marked as outside the molecular surface. Figure 5 illustrates this procedure in two dimensions. The grid cells that were marked

[9] If the propagating fronts from multiple atoms merge, the total surface areas can only get smaller.

as inside in the previous stage are now marked outside by this procedure, if they fall inside a probe circle. Again, the extension of this illustration to three dimensions involves probe spheres instead of probe circles.

However, it is obvious that the brute-force method (which is used by the *GRASP* method) is very inefficient, as it requires every pair and triple of atoms to be examined for possible probe placements. Probe placement process can be made efficient by examining the atoms that are spatially close. However, placement of overlapping probe spheres, hence the redundant examination of grid cells, cannot be avoided. We propose a much more efficient method which guarantees that (1) only cells that are inside the solvent-accessible surfaces are examined, and (2) each cell will be examined at most once.

The key observation is that finding the solvent-excluded surfaces (or molecular surfaces) is just another front marching process as before. But this time, we "seed" the marching front at the solvent-accessible surfaces to start with, and march the front *inward* instead of *outward* for a distance specified by the solvent radius. For a protein embedded in space with complicated twists and turns, one might wonder how to decide the directions that are "inward." This turns out to be very easy — inward directions correspond to those that lead to cells that were visited in the previous step. Cells that were not visited in the previous steps lie in the outward directions.

An algorithm similar to the one presented above is used here. At the beginning the `NEAR` set comprises all cells that are on the solvent-accessible surfaces found in the previous iteration. As there is a single probe sphere and a single propagation distance, it is not necessary to maintain the ID information in this iteration as before. The `FAR` set then includes all other cells that were

visited in the previous step. Finally, the `VISITED` set includes all cells that were *not* visited in the previous step (we mark them as `VISITED` so they will not be examined here).

The time complexity of the second stage is similar to that of the first stage. Again, it will depend on the resolution of the grid as well as the complexity of the solvent-accessible surface. A protein surface which contains a lot of pockets (outside cavities) will have a much larger surface area compared to a surface that is smooth. Therefore, it would require more computation time. In any case, our technique will visit only cells that are inside the solvent-accessible surfaces and will not visit any cell more than once. Furthermore, our technique performs efficient operations at each loop iteration.

Interior cavity detection. The result of the first two stages is a grid that represents the volume occupied by solvent-excluded surface of the molecular structure. However, the interior inaccessible cavities are not distinguished from the surrounding space in this representation. The volume occupied by them should be excluded in the total molecular volume. Furthermore, if one computes the molecular surface area using that grid, the resulting surface area will not be the area of the molecule in contact with its surrounding environment, but will also include the surface areas of the interior inaccessible cavities. Hence, we need to distinguish between the outer surface and the interior cavities. Figure 6 illustrates an inaccessible cavity in two dimensions. In three dimensions the illustration should not be realized as a torus shape, but instead as a small sphere inside a larger sphere, where the small one would be the inaccessible cavity.

The idea is to initialize an evolving surface that encloses the molecule, then

shrink the surface at a constant speed. The stopping criterion in the speed function is the encounter with a grid cell that is marked as inside. The evolution of the surface stops completely when all the surface points are stopped by an inside grid cell. The key observation here is that during this shrinkage process, the surface has a fixed signed speed, i.e., a grid cell passed over by the surface will not be visited again by any part of the surface.

With this observation we can again apply the Fast Marching Method, in which the closest grid points to the evolving surface are considered first, and a grid cell that is processed is never processed again. In this procedure we again maintain a narrow band of grid cells in a heap that represents the current evolving surface and update that narrow band as the surface evolves. At the end of this procedure the outer surface is detected; and the voids inside the molecular surface are detected as interior inaccessible cavities. We present examples of interior cavities in the experimental results section.

3 Results

We conducted for sets of experiments to evaluate the performance and utility of our method. First, we evaluate the running time performance of LSMS for computing and visualizing molecular surfaces of very large complexes. Second, we compare the quantitative measures of molecular surfaces, such as solvent-accessible surface area and solvent-excluded surface area measures, computed by LSMS to the corresponding values reported by MSMS, a widely used tool to compute molecular surface measures. Third, we compare the cavity detection accuracy by comparing our results to cavities reported by Swiss-PDBViewer. Finally, we analyze the effect of varying the grid size on generated surfaces

and cavities. We explain the experiments in detail below.

Molecular surface generation and visualization performance. We selected a challenging set of large molecules as a benchmark dataset. The dataset consists of 15 large complexes that contain 27375 to 97872 atoms in their structure data. We compared LSMS's molecular surface generation performance to three other programs that are widely used in the computational biology community and are freely available. PyMOL[16] is an open source molecular graphics system with an embedded Python[21] interpreter designed for real-time visualization and rapid generation of high-quality molecular graphics images and animations. UCSF Chimera[14] is another tool implemented in Python. The solvent-excluded molecular surfaces produced by Chimera are created with the help of the MSMS package[12]. Swiss-PDBViewer (v3.7 SP5)[17], or SPDBV for short, is another molecular viewer with extended functionality. There is scanty documentation on the molecular surface component of SPDBV though. Nevertheless, it can be understood from the documentation that the surface computation is carried out on an orthogonal grid as in LSMS. The probe size is 1.4 Å as in other methods, however all the molecule atoms have a fixed radius. The current version of SPDBV does not allow changing of these parameters. The only value that can be altered is the smoothness (quality) parameter. By default it is 1, which means 1 grid point every 1.4 Å. This quality should be enough for most purposes as indicated by SPDBV's developers [10] .

Table 1 shows the molecular surface generation times. All of the tests were performed on a Microsoft Windows XP machine with Intel Pentium 4 Pro-

[10] http://us.expasy.org/spdbv/text/surface.htm

cessor at 2.0GHz and 512MB of RAM. The results we report here used the programs' default parameter sets and did not include the time taken to load the molecule into memory. The timings for Swiss-PDBViewer are acquired using a quality value of 1. We also computed a quality measure for LSMS that directly corresponds to Swiss-PDBViewers's quality measure, i.e., the number of grid cells per 1.4 Å. The corresponding quality values for LSMS are shown on the table. We used a 256×256×256 resolution grid for timing LSMS.

Figure 7 shows the largest protein in our dataset in 256×256×256 resolution and a quality of 1.28. Furthermore, LSMS can interactively render that surface with varying viewpoints with a 9-frames-per-second refresh rate.

Table 1 shows that LSMS is up to 2.46 times faster than SPDBV (achieved at protein 1j0b) and is 1.5 times faster on the average, while achieving a better quality for every test protein. LSMS is also 3.14 times faster than both PyMol and Chimera on average. Also, it is important to note that for some of the test cases SPDBV, PyMol, and Chimera are not even able to generate the molecular surfaces, whereas LSMS successfully computes the surfaces for all of the test cases.

Quantitative measures of molecular surfaces Apart from the generation of three-dimensional representations of molecular surfaces, reporting quantitative measures, such as the solvent-accessible surface area and solvent-excluded surface area, are among the desired functionalities of a molecular surface package. In this section, we compare the surface area and molecular volume values given by LSMS to the corresponding values reported by MSMS [12], a *de facto* standard in molecular surface calculations. Since LSMS produces a molecular surface that is represented by rectangular grid cells, based on the granularity

of the grid, molecular volume and surface area values are approximations to actual values at different levels of accuracy. Table 2 shows molecular volume and surface area values for nine different protein molecules. The last two proteins in the table are very large molecules that contain 45892 and 64281 atoms respectively and given for informative purpose rather than for comparison. It can be concluded from the table that, LSMS usually produces smaller area and volume values compared to the values generated by MSMS. This is expected since the Euclidian distance in grid space is only an approximation to actual distances in molecular space and the approximated spheres occupy a smaller volume than actual spheres. Nevertheless, the values provided by LSMS can be considered as close-enough approximations for many biological tasks. The important message here is that, approximations generated by LSMS may prove useful for very large molecules, where exact techniques are unable to produce any quantitative results (Table 2).

Interior cavity detection. The outer boundary of the solvent accessible surface is found by shrinking an initial enclosing boundary at a constant speed with the Fast Marching Method. Figure 8 shows such an outer surface of the protein 2ptn computed and displayed by LSMS. However, as we stated earlier there may exist inaccessible cavities inside the molecular surface that are not visible, i.e., occluded by the molecular surface. Nevertheless, analysis of these cavities may be required to study the buried water molecules inside them which may contribute to protein folding stability.

Figure 9 shows the internal cavities of the same protein 2ptn that can accommodate one or more water molecules. The C_α trace is also shown along with the cavities to provide visual clues of relative locations of the cavities inside the molecule.

In order to assess the interior cavity detection capabilities of LSMS, we analyzed internal cavities of a set of seven protein molecules. The interior cavities of the tested molecules were visually inspected and compared to the interior cavities reported by Swiss-PDBViewer (SPDBV). As for quantitative measures of detected cavities, we use the number of distinct cavities obtained by visual inspection and the total volume of cavities computed numerically by the programs (Table 3). We also provide the molecular volume, in Table 3, as a reference for possible differences in volume computation methodologies employed by the compared programs. The results show discrepancies between LSMS and SPDBV both in terms of the number of distinct cavities detected and the volume occupied by these interior cavities. In general, LSMS tends to find more distinct cavities compared to SPDBV. However, the total volume of the cavities are comparable, and in four of the test proteins, SPDBV reports a larger total cavity volume. We verified by further visual inspection that larger cavities reported by SPDBV is split by LSMS into a number of smaller cavities (with all of the large cavities in SPDBV accounted for).

Several factors may cause the disagreements in the number and total volume of detected cavities. As we address in the next section, grid size is an important parameter that effects the size and number of cavities detected by LSMS. Other factors include the employment of different van der Waals radii by the two methods and SPDBV's fixed atom radii strategy, i.e., fixed radius for all the atoms in the molecule.

Effect of the grid size. The chosen grid size is an important factor that effects the running time of the molecular surface generation process as well as the quality of the generated surfaces and cavities. Here, we provide both visual and numerical analysis of the effect of different grid sizes on the protein

molecule, 2cha. Figure 10 shows the molecular surface and cavities of 2cha under three different grid sizes. A coarser molecular surface is generated at a grid of size $64\times64\times64$, while the $256\times256\times256$ grid provides a more accurate surface of the same molecule. The figure also shows that the detected cavities increase in volume and in number with the size of the grid. With a more refined grid, the probe molecule (i.e, water molecule) is able to trace the interior cavities of the molecule more accurately. Table 4 gives numerical values of solvent-accessible and molecular surface areas, solvent-excluded volume, and the total surface area and volume of the interior cavities. With an increasing grid size the molecular surface is refined and therefore the volume decreases, i.e., the surface can be represented in more detail by smaller grids. As the molecular surface is refined, the details of the surface areas become apparent and the total size of the area increases.

4 Conclusions

In this article, we presented a method to calculate all kinds of protein surfaces, such as van der Waals, solvent-accessible, and solvent-excluded surface, as well as the interior inaccessible cavities of a molecular structure in a unified framework based on level-set methods. In particular, the proposed method is based on a fast marching level-set method that efficiently formulates a constant signed speed evolving boundary. We showed that the surface generation and cavity detection tasks can be solved in a unified and efficient manner using level-set surface evolution. The benefits of our proposed technique is apparent especially for very large molecular structures that contain 25K or more atoms, for which some of the existing molecular surface packages cannot

even generate a surface. However, potential users of LSMS should note that the grid based computation of molecular surfaces by LSMS provides approximate surfaces and for smaller molecules processed in a coarser grid, existing molecular surface packages will provide a more accurate and smoother looking surfaces than LSMS. Therefore, we do not envision LSMS as a replacement for all molecular surface packages, but as a molecular surface generation tool that can scale very well to extremely large molecules. Moreover, the quantitative surface measures and the interior cavities provided by LSMS are good approximations to the corresponding measures provided by commonly used packages such as MSMS and Swiss-PDBViewer.

Availability: LSMS is available free of charge with source code at http://www.ceng.metu.edu.tr/~tcan/LSMS/. The distribution is tested under Microsoft Windows XP environment.

References

[1] M. Gerstein, F. M. Richards, M. S. Chapman, M. L. Connolly, Protein surfaces and volumes: measurement and use, Vol. F of International Tables for Crystallography. Crystallography of Biological Molecules, Kluwer Academic Publishers, Dortrecht, Netherlands, 2001, Ch. 22.1, pp. 531–545.

[2] S. J. Hubbard, P. Argos, Cavities and packing at protein interfaces, Protein Sci. 3 (12) (1994) 2194–2206.

[3] F. M. Richards, Areas, Volumes, Packing, and Protein Structure, Ann. Rev. Biophys. Bioeng. Vol 6 (1977) 151–176.

[4] K. Takano, Y. Yamagata, K. Yutani, Buried water molecules contribute to the conformational stability of a protein, Protein Eng. 16 (1) (2003) 5–9.

[5] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, S. Subramaniam, Analytical shape computation of macromolecules: Ii. inaccessible cavities in proteins, Proteins 33 (1) (1998) 18–29.

[6] M. L. Connolly, Analytical molecular surface calculation, J. Appl. Crystallogr. 16 (1983) 548–558.

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

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

[9] A. Nicholls, GRASP: graphical representation and analysis of surface properties, Columbia University, New York (1992).

[10] S. Osher, R. Fedkiw, Level set methods and dynamic implicit surfaces, Vol. 153 of Applied Mathematical Sciences, Springer-Verlag, 2003.

[11] J. A. Sethian, Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision and materials science, 2nd Edition, Cambridge University Press, 1999.

[12] M. F. Sanner, A. J. Olson, J. C. Spehner, Reduced surface: an efficient way to compute molecular surfaces, Biopolymers 38 (3) (1996) 305–320.

[13] H. Edelsbrunner, E. P. Mucke, Three-dimensional alpha shapes, ACM T. Graphic. 13 (1) (1994) 43–72.

[14] E. F. Pettersen, T. D. Goddards, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin, UCSF Chimera – A Visualization System for Exploratory Research and Analysis, J. Comput. Chem. 25 (13) (2004) 1605–1612.

[15] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, Computer Graphics 21 (4) (1987) 163–169.

[16] W. L. DeLano, The PyMOL Molecular Graphics System User's Manual, DeLano Scientific, San Carlos, CA, USA (2002).

[17] N. Guex, M. C. Peitsch, SWISS–MODEL and the Swiss–PdbViewer: An environment for comparative protein modeling, Electrophoresis 18 (1997) 2714–2723.

[18] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, P. E. Bourne, The protein data bank, Nucleic Acids Res. 28 (1) (2000) 235–242.

[19] R. Sayle, E. J. Milner-White, RasMol: Biomolecular graphics for all, Trends Biochem. Sci. 20 (9) (1995) 374–376.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd Edition, MIT Press, Cambridge, MA, 1990.

[21] M. F. Sanner, Python: a programming language for software integration and development, J. Mol. Graph. Model. 17 (1) (1999) 57–61.

Table 1

Molecular surface generation times for LSMS compared to those of Swiss-PDBViewer, PyMol, and Chimera. *Note:* Protein sizes are shown as number of atoms. Surface quality is given as the the number of grid cells per 1.4 Å. *u* means that the program is not able to generate a molecular surface.

		surface generation time (sec.)				surface quality	
Protein	size	LSMS	SPDBV	PyMol	Chimera	LSMS	SPDBV
1a8r	27375	11.66	14.96	51.34	16.36	1.03	1
1h2i	32802	15.66	17.33	40.78	40.04	1.29	1
1fka	34977	37.14	51.56	85.14	77.25	1.34	1
1gtp	35060	15.81	19.75	50.17	67.04	1.28	1
1gav	43335	20.31	35.24	86.62	78.35	1.28	1
1g3i	45528	26.80	37.51	63.90	*u*	1.41	1
1pma	45892	40.78	*u*	51.10	*u*	1.67	1
1gt7	46180	14.91	22.60	57.75	54.39	1.16	1
1fjg	51995	30.34	48.44	85.79	*u*	1.33	1
1aon	58884	47.28	61.20	95.84	*u*	1.41	1
1j0b	60948	18.28	44.97	100.14	*u*	1.18	1
1ffk	64281	69.83	72.07	135.80	196.65	1.27	1
1otz	68620	42.05	51.27	78.05	*u*	1.45	1
1ir2	87087	21.09	*u*	120.52	93.87	1.23	1
1hto	97872	38.95	89.68	*u*	*u*	1.28	1

Table 2

Quantitative measures of molecular surfaces compared to corresponding measures reported by MSMS. (SAS: Solvent-accessible Surface Area, SES: Solvent-excluded Surface Ares, SEV: Solvent-excluded Volume) (*u: the program is unable to process the protein molecule)*

	LSMS			MSMS		
Protein id:	SAS ($Å^2$)	SES ($Å^2$)	SEV ($Å^3$)	SAS ($Å^2$)	SES ($Å^2$)	SEV ($Å^3$)
1eca	6797	5812	17139	6959	5875	20998
2act	8878	7477	27040	11484	9331	33941
2cha	10354	9021	29174	10607	9086	31078
2lyz	6471	5350	16101	7740	6420	19099
2ptn	8957	7746	27333	9153	7866	29342
5mbn	7946	6864	20262	8982	7790	24783
8tln	12005	10447	40760	12708	10975	44895
1pma	193157	185487	851062	*u*	*u*	*u*
1ffk	453675	463387	1272399	*u*	*u*	*u*

Table 3

Cavities computed using LSMS and comparison with results from Swiss-PDBViewer.

	# of cavities		cavity volume (Å^3)		molecule volume (Å^3)	
Protein	LSMS	SPDBV	LSMS	SPDBV	LSMS	SPDBV
1eca	1	1	31.46	134	16688.98	16824
2act	7	2	322.33	281	6842.76	6509
2cha	10	4	338.01	436	28728.13	27705
2lyz	3	2	96.12	162	15778.44	15133
2ptn	6	3	394.41	380	27037.94	25897
5mbn	4	2	127.05	189	19796.24	19768
8tln	14	2	356.28	170	40280.45	38498

Table 4

The effect of varying grid size on the generated molecular surface and detected interior cavities for the protein molecule 2cha. (SAS: Solvent-accessible Surface, SES: Solvent-excluded Surface)

	Outer Surface			Interior Cavities		
Grid Size	SAS (Å^2)	SES (Å^2)	Volume (Å^3)	#	Surface (Å^2)	Volume (Å^3)
64×64×64	9919	8969	34217	7	238	109
128×128×128	10289	9060	30629	10	395	241
256×256×256	10475	9163	28728	10	504	338

List of Figures

1 A two-dimensional illustration of surface definitions. 38

2 The grid cells whose centers fall inside the volume defined by the solvent-accessible surface is marked as inside. 39

3 Surface front propagation from an initial seed atom. The vertical dimension is exaggerated for viewing. 40

4 A 2D illustration of the shape of the surface where fronts from different atoms meet and quench each other. 41

5 The grid cells whose centers fall inside the probe circles are marked as outside. 42

6 A two-dimensional illustration of an inaccessible cavity. 43

7 The molecular surface of 1hto generated by LSMS. The boundaries of the 256×256×256 resolution grid are also shown. 44

8 The molecular surface of 2ptn generated by LSMS. The boundaries of the 256×256×256 resolution grid are also shown. 45

9 Cavities inside 2ptn along with its C_α trace. 46

10 The molecular surface (top row) and interior cavities (bottom row) of the protein molecule, 2cha, generated at varying grid sizes: (a) 64×64×64 grid, (b) 128×128×128 grid, (c) 258×256×256 grid. 47

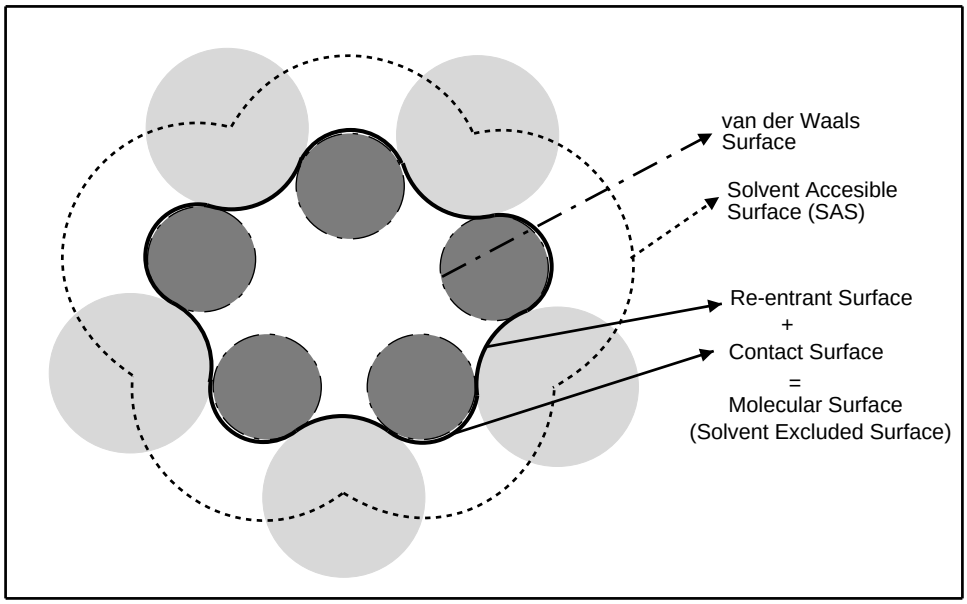

Fig. 1. A two-dimensional illustration of surface definitions.

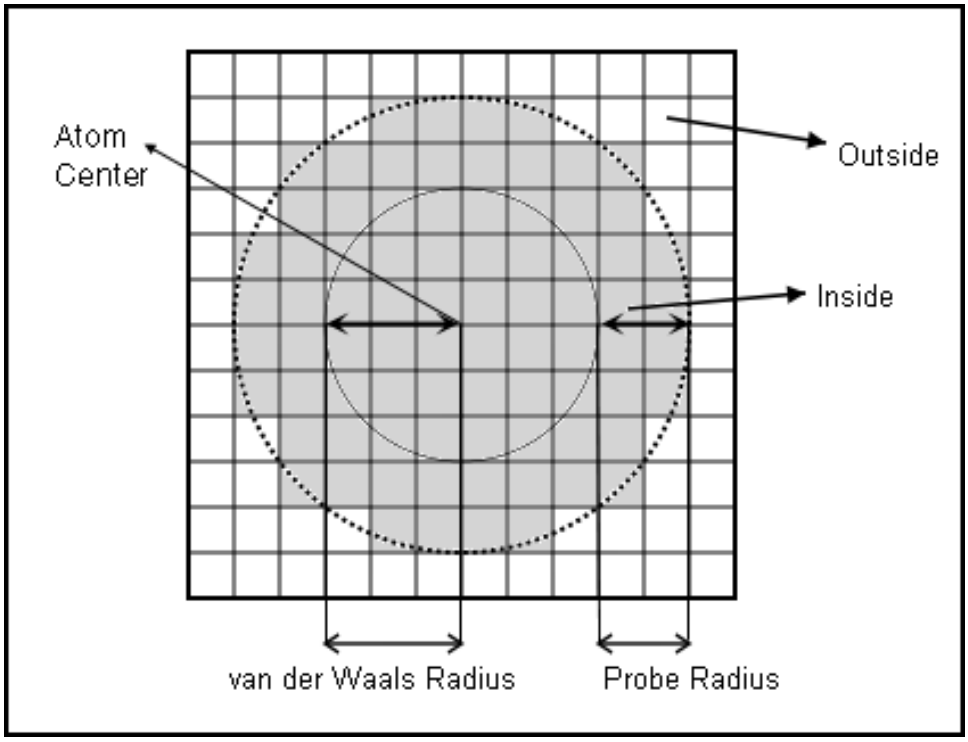

Fig. 2. The grid cells whose centers fall inside the volume defined by the solvent-accessible surface is marked as inside.

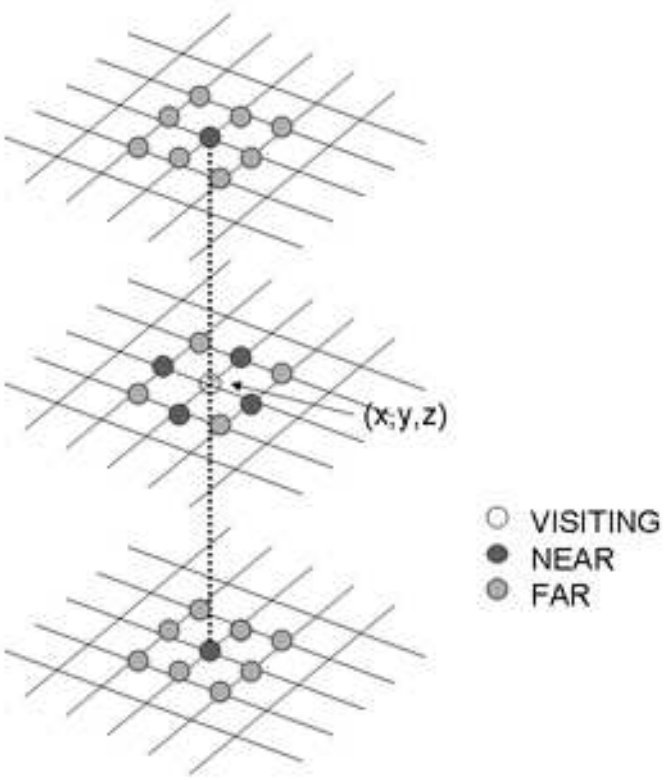

Fig. 3. Surface front propagation from an initial seed atom. The vertical dimension is exaggerated for viewing.

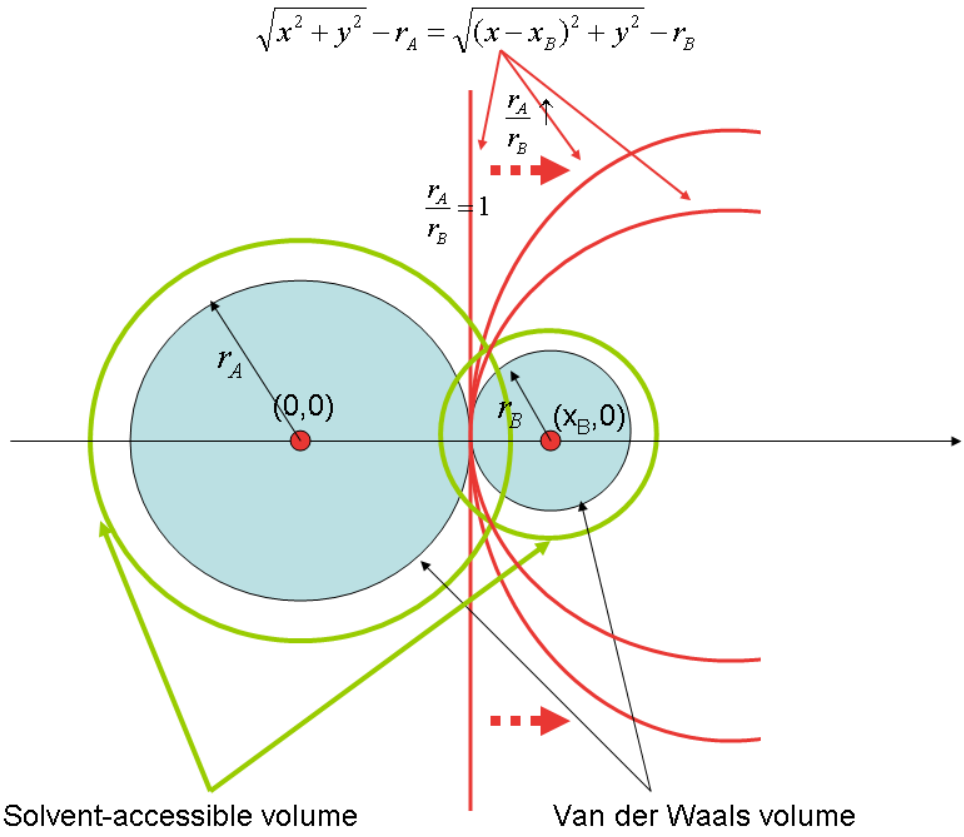

Fig. 4. A 2D illustration of the shape of the surface where fronts from different atoms meet and quench each other.

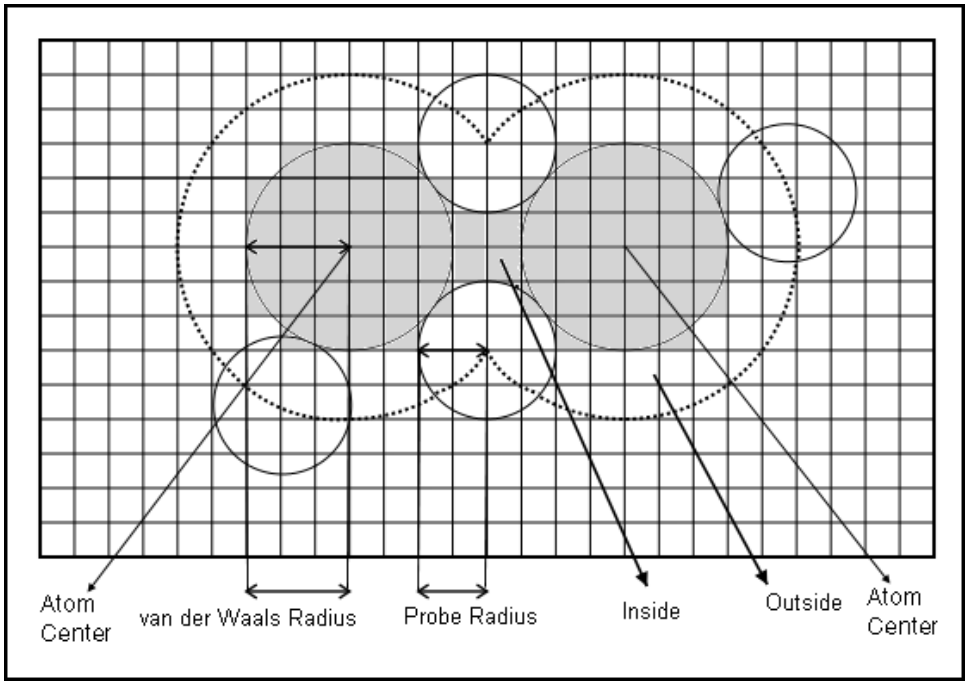

Fig. 5. The grid cells whose centers fall inside the probe circles are marked as outside.

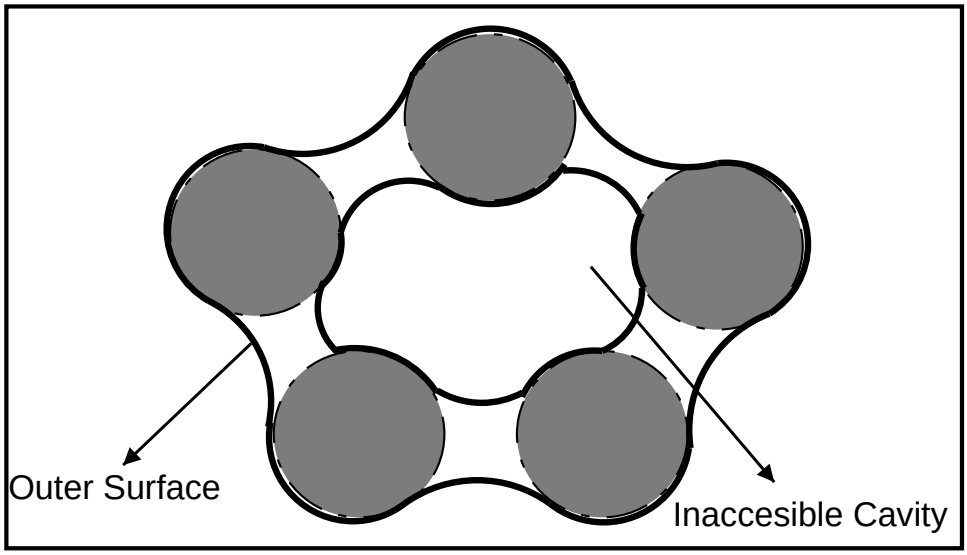

Fig. 6. A two-dimensional illustration of an inaccessible cavity.

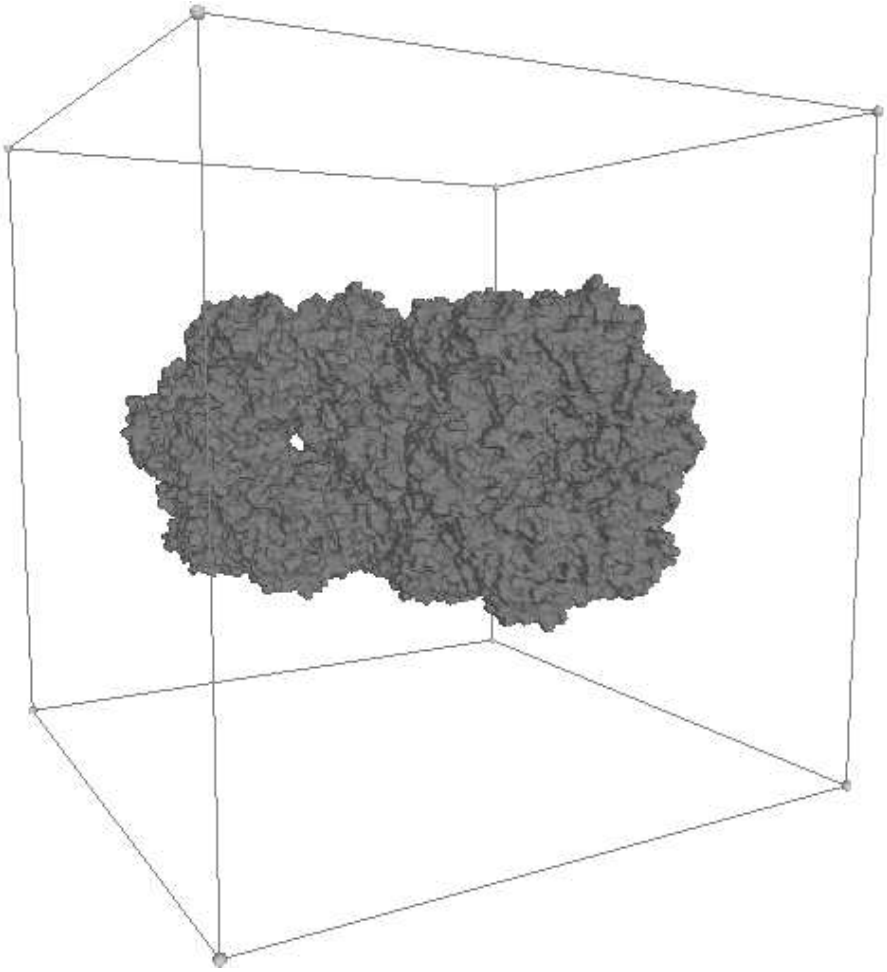

Fig. 7. The molecular surface of 1hto generated by LSMS. The boundaries of the 256×256×256 resolution grid are also shown.

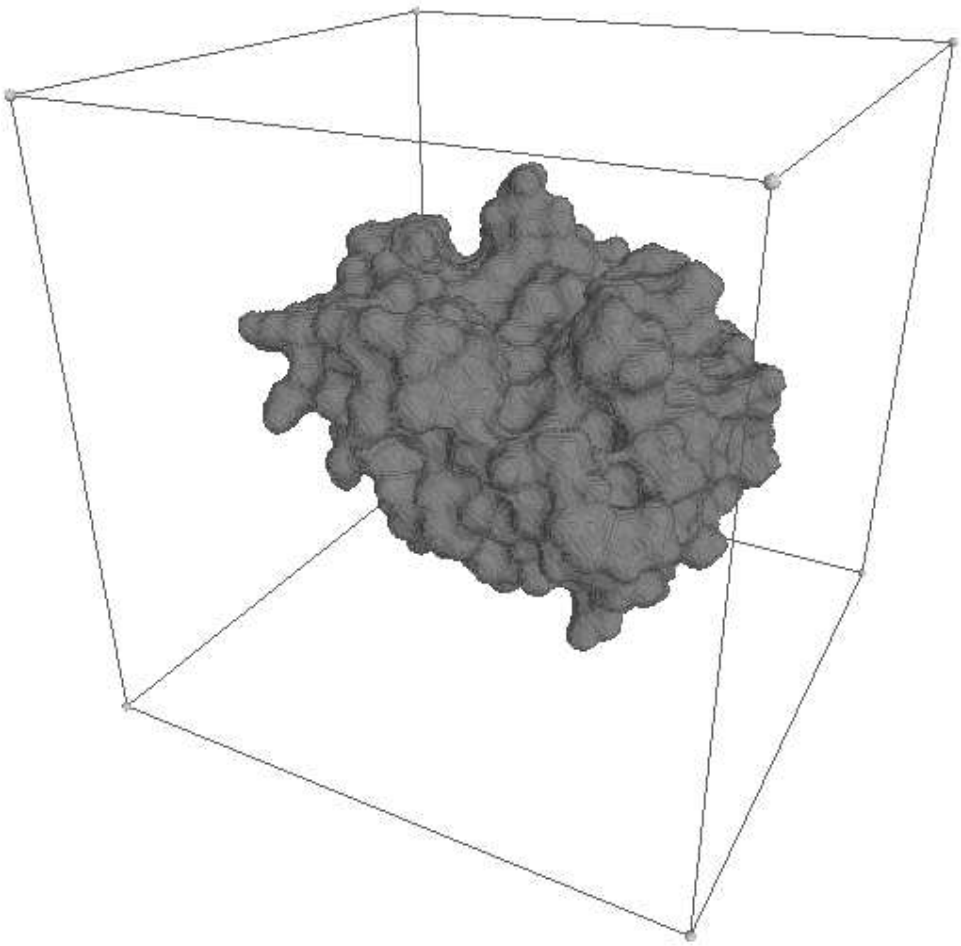

Fig. 8. The molecular surface of 2ptn generated by LSMS. The boundaries of the 256×256×256 resolution grid are also shown.

Fig. 9. Cavities inside 2ptn along with its C_α trace.

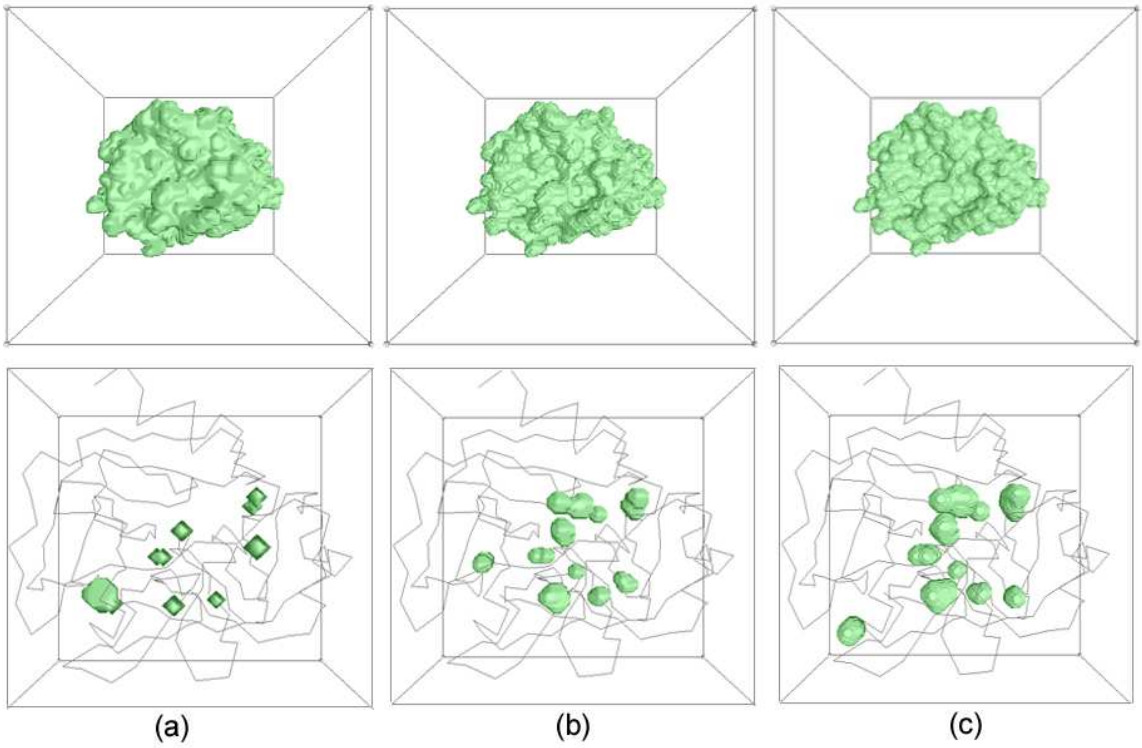

Fig. 10. The molecular surface (top row) and interior cavities (bottom row) of the protein molecule, 2cha, generated at varying grid sizes: (a) 64×64×64 grid, (b) 128×128×128 grid, (c) 258×256×256 grid.