ICES REPORT 14-23

August 2014

Maintaining Flexible Molecular Surfaces Using Augmented Dynamic Octrees

by

Muhibur Rasheed, Alexander Rand and Chandrajit Bajaj

The Institute for Computational Engineering and Sciences
The University of Texas at Austin
Austin, Texas 78712

Reference: Muhibur Rasheed, Alexander Rand and Chandrajit Bajaj, "Maintaining Flexible Molecular Surfaces Using Augmented Dynamic Octrees," ICES REPORT 14-23, The Institute for Computational Engineering and Sciences, The University of Texas at Austin, August 2014.

Maintaining Flexible Molecular Surfaces Using Augmented Dynamic Octrees

Muhibur Rasheed [1], Alexander Rand [1] and Chandrajit Bajaj [1]

[1]Department of Computer Science, and
Institute of Computational Engineering and Sciences
The University of Texas at Austin
Austin, Texas, USA

Abstract

We report the Dynamic Adaptive Grid data structure which adaptively subdivides space to provide higher resolution sampling near the boundary (surface) of a molecule with n atoms in $O(n \log n)$ time. It can support any surface approximation as long as it is expressed as a level set of a volumetric function. Our implementation includes the van der Waals surface, the solvent accessible surface (SAS), and the solvent excluded surface (SES) all of which requires the evaluation of an anlytical signed distance function at each gridpoint, and a faster Gaussian integration based surface approximation. This grid-based approach prodives simultaneous maintenance of molecules in atomic, smooth surface and volumetric representations, making it applicable for a wide range of applications. Also, experiments on a large set of proteins showed that our algorithm runs faster and requires less memory than regular uniform grid-based approaches. Finally, when an atom is moved (added or removed), the surface can be dynamically and locally updated by our algorithm in $O(\log n)$ time. Experiments showed that the constant hidden in the $O(\log n)$ update is not negligible, and dynamic updates are profitable if fewer than 10% of the atoms are moved in a step, otherwise reconstruction of the entire grid is faster. This indicates that, in many applications, for example in docking and structure refinement, where only a fraction of the atoms need to be updated frequently, the dynamic algorithm we report here is beneficial.

1 Introduction

Probably the simplest model for molecules is as a collection of atoms, each atom represented as hard spheres, with radii equal to their van der Waals radii. The boundary of this arrangement of spheres is a possible surface representation and is usually referred to as the van der Waals Surface (vdW). Lee and Richards [23] proposed the Solvent Accessible Surface (SAS) as the boundary of the region which is accessible for solvent molecules. It is modeled as the locus of the center of a water molecule, considered as a sphere with radius 1.4Å, as it rolled along the protein surface. While both of these models are sufficient for visualization purposes, they are not suitable for simulations and energetics computations that require numerical integrations over the surface, since it contains singularities at the sphere-sphere intersections. We need smooth surfaces for such applications.

Richards [24] proposed a model commonly known as the Solvent Contact Surface (SCS), or Solvent Excluded Surface (SES), as the boundary of the volume not penetrated by a solvent molecule. This surface is composed of convex patches where the probe touches the atom surfaces, concave spherical patches when the probe touches more than 2 atoms simultaneously and toroidal patches when the probe rolls between two atoms. An even smoother representation is to represent each atom using a Gaussian kernel whose mean is at the center of the atom [10, 14, 9, 21, 20]. A volumetric function is defined over the entire space as a summation of these Gaussian kernels, and a level set of this volumetric function is considered the molecular surface, usually called the Gaussian Molecular Surface (GS). Some variations on the Gaussian formulation were discussed in [15, 19, 33]. Figure 1 shows examples for each of the four representations we discussed here.

Since Richards introduced the SES definition, a number of techniques have been devised to compute the surface, for both static and dynamic cases, and using both implicit and explicit representations. Connolly introduced two algorithms to com-

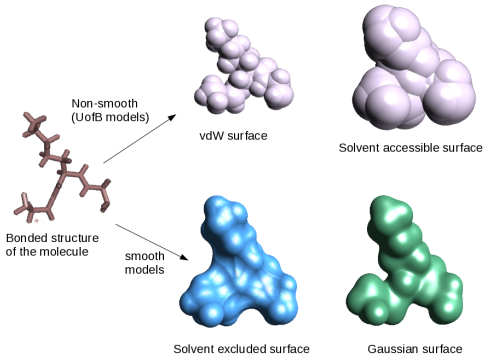


Figure 1: Different surface representations for the same collection of atoms. The vdW and SAS surfaces are non-smooth and represents each atom as a hard sphere with radius r and $r + r_p$ respectively, where r is the van der Waals radius of the atom, and r_p is the radius of the solvent (1.4Åin the case of water). The SES is a smooth surface, where the singuralities of the vdW surface are removed by using toroidal patches to cover pairwise intersections, and concave patches to cover 3-way or higher intersections of the vdW spheres. Finally the Gaussian surface is defined as a level set of a volumetric function defined as a sum of Gaussian kernels placed at the centers of the atoms.

pute the surface. First, a dot based numerical surface construction and second, an enumeration of the patches that make up the analytical surface (See [13]). In [28], the authors describe a distance function defined over a grid for computing surfaces of varying probe radii. Our data structure contains approaches similar to their idea. A number of algorithms were presented using the intersection information given by voronoi diagrams and the alpha shapes introduced by Edelsbrunner [16], including parallel algorithms in [27] and a triangulation scheme in [1]. SES based on arrangment of spheres for both static and dynamic cases were reported in [17, 18]. Another computation of SES is described in [25], using Reduced sets, which contains points where the probe is in contact with three atoms, and faces and edges connecting such points. Non Uniform Rational BSplines (NURBs) descriptions for the patches of the molecular surfaces are given in [6], [5] and [7]. You and Bashford in [32] defined a grid based algorithm to compute a set of volume elements which make up the Solvent Accessible Region. Another grid-based algorithm using 2D arrangment of circles was reported in [3].

Various modifications of the Gaussian kernels and their effects were analyzed in [15, 19, 33]. Recently, Zhang et al. considered representing the molecular surface as a level set of a high order polynomial and showed that the additional degrees of freedom available from such a model can be used to satisfy user-defined constrains and quality metrics for the molecular surface [8]. In this paper, we also report that the prevalent choice of Gaussian parameter blobbiness and isovalue to be 2.3 and 1.0 respectively does not always lead to surfaces close to the SES model. We found that the choice of the parameters depend on the property of the molcule one is interested in. For example, to get a better estimate of the surface area, small blobbiness (1.0-1.2) with large isovalue (1.8-2.2) is required.

In many applications such as docking and pairwise alignment of molecules [11, 22, 2, 12], a volumetric/grid representation is more amenable for applying FFT-based fast computations. So, in this paper we first show that each of the four surface types can be expressed as level sets of volumetric functions and develop a data structure and algorithms to support and maintain atomic, smooth surface and volumetric representation of a molecule simultaneously. Additionally, our octree based scheme allows one to sample/construct the surface at multiple resolutions with provable approximation errors. Note that other existing fast approximation algorithms like [29, 31, 30] do not provide theoretical error bounds.

2 Molecular Surfaces as Level Sets

Given a set of atoms $M = \{a_i, \ldots, a_n\}$, where the center and radius of each atom is $\mathbf{c}_i$ and r_i, we define a volumetric function $\Phi^M(\mathbf{x}) : \mathbb{R}^3 \mapsto \mathbb{R}$ and use its iso-contours $\Gamma(\Phi^M, v) = \{\mathbf{x} | \Phi^M(\mathbf{x}) = v\}$, where v is a scalar, to provide a family of molecular surfaces. In the following, we analyze three specific cases.

2.1 Level Set of Sum of Gaussian Kernels

Define $\Phi^M_{GS}(\mathbf{x})$ as follows-

$$\Phi^M_{GS}(\mathbf{x}) = \sum_i e^{-\beta(1-\frac{\|\mathbf{c}_i-\mathbf{x}\|^2}{r_i^2})} \tag{2.1}$$

where β is called the blobbiness parameter, which controls the width (or the decay rate) of the Gaussian kernel. Different choices of blobbiness and isovalues v would lead to different surfaces, not only in terms of the area/volume, but also in terms of topology. Traditionally, one chooses a β and v such that the resulting isosurface approaches the vdW or SES surface.

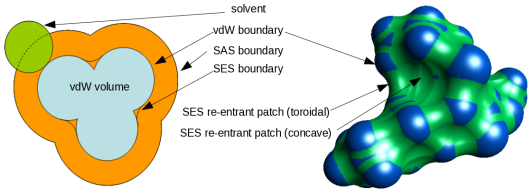


Figure 2: The different molecular surfaces and regions are shown for a 3 atom model in 2D. The SAS surface is the locus of the center of the rolling probe sphere. The VDW surface is the exposed union of spheres representing atoms with their van der Waals radii and contains the VDW volume. The lower side of the rolling probe defines the smooth SES which contains parts of the VDW surface and reentrant patches. We also define the SAS volume as the region between the SAS and SES. The region between the SAS and VDW volumes is later refered to as the SES volume.

2.2 vdW Surface as a Level Set

Define $\Phi^M_{vdW}(\mathbf{x})$ as follows-

$$\Phi^M_{vdW}(\mathbf{x}) = s^M_{vdW}(\mathbf{x})\delta^M_{vdW}(\mathbf{x}) \tag{2.2}$$

where $s^M_{vdW}(\mathbf{x})$ is a sign function indicating whether the point $\mathbf{x}$ is inside or outside the vdW sphere of any atom. In other words, $s^M_{vdW}(\mathbf{x}) = 1$ if $\exists_i \|\mathbf{c}_i - \mathbf{x}\|^2 \leq r_i^2$, and -1 otherwise. And $\delta^M_{vdW}(\mathbf{x})$ is the shortest distance from the point $\mathbf{x}$ to the vdW sphere of any atom, in other words, $\delta^M_{vdW}(\mathbf{x}) = min_i|\|\mathbf{c}_i - \mathbf{x}\| - r_i|$.

Functions of this form are called signed distance functions. Note that the $\Gamma(\Phi^M_{vdW}, 0)$ defined the vdW surface.

2.3 SAS and SES Surfaces as Level Sets

Since, the definitions of both SAS and SES depends on rolling a solvent molecule, perhaps it comes as no surprise that both can be defined as level sets of the same volumetric function defined as follows-

$$\Phi^M_{SAS}(\mathbf{x}) = s^M_{SAS}(\mathbf{x})\delta^M_{SAS}(\mathbf{x}) \tag{2.3}$$

where $s^M_{SAS}(\mathbf{x})$ is a sign function indicating whether the point $\mathbf{x}$ is inside or outside the solvent enlarged sphere of any atom. In other words, $s^M_{SAS}(\mathbf{x}) = 1$ if $\exists_i \|\mathbf{c}_i - \mathbf{x}\| \leq (r_i + r_p)$, and -1 otherwise. And $\delta^M_{SAS}(\mathbf{x})$ is the shortest distance from the point $\mathbf{x}$ to the SAS surface. Computing $\delta^M_{SAS}(\mathbf{x})$ is not trivial and is described in the next subsection.

- $\Gamma(\Phi^M_{SAS}, 0)$ defines the SAS surface.
- $\Gamma(\Phi^M_{SAS}, r_p)$ defines the SES surface.

See Figure 2 for 2D and 3D examples of the vdW, SAS and SES surfaces for an intuitive understanding of these particular signed distance functions and the chosen iso-values.

2.3.1 Computing Signed Distance Function $\Phi^M_{SAS}(\mathbf{x})$ from the SAS

Note that, we do not really have an analytical representation of the SAS, and hence cannot compute the distance $\delta^M_{SAS}(\mathbf{x})$ directly. We do however, have analytical representation of the SAS for individual atoms. Here we want to compute $\delta^M_{SAS}(\mathbf{x})$ using only local per atom definitions of the SAS.

Figure 3 provides an example that shows the complexity of computing $\delta^M_{SAS}(\mathbf{x})$ using local information only. In that 2D example, we see that the SAS boundary of individual atoms gets buried when two or more atoms intersect and hence do not contribute to the SAS boundary of the entire molecule. As such, if the closest point on the molecular SAS, from any point $\mathbf{x}$ may not be the closest point on the atomic SAS from the same point. We need to consider several cases. First we define some regions of the intersections of the spheres.

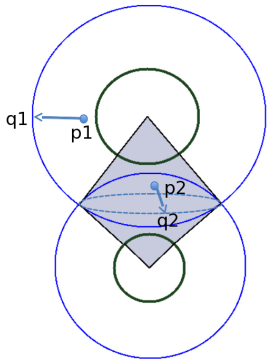


Figure 3: A 2D example with only 2 atoms showing that multiple cases should be considered when computing distance to the solvent accessible surface (SAS). The point p_1 lies inside the SAS of only one atom, and the its closest point on the SAS is simply the closest point on the atom. But in case when the point lies within the intersection region of 2 atoms (eg. p_2, the closest point on the SAS is no longer the closest point on either atom (since that closest point is not part of the SAS, but is actually buried inside the SAS). In this case, the closest point must be chosen from the intersection of the two atoms.

- **Cone of intersection** If the SAS boundaries of two atoms a_i and a_j intersect, then let C_{ij} define the circle of intersection of the spheres. Then we define two infinite cones with apexes

at $\mathbf{c}_i$ and $\mathbf{c}_j$ and going through C_{ij}. The intersection of these cones are defined as CI_{ij}. A 2D anlog of this can be seen in 3 as the shaded quadrilateral. Note that for any point $\mathbf{x}$ inside CI_{ij}, the closest boundary point, not buried by a_i or a_j, must lie on C_{ij}.

- **Tetrahedron of intersection** When 3 spheres a_i, a_j and a_k have a common intersection, then common region is like a trigon, three spherical patches with converging onto exactly 2 points of intersections $\mathbf{t}_{ijk1}$, and $\mathbf{t}_{ijk2}$. Then we define two tetrahedrons $T_{ijk1} = \mathbf{c}_i\mathbf{c}_j\mathbf{c}_k\mathbf{t}_{ijk1}$ and $T_{ijk2} = \mathbf{c}_i\mathbf{c}_j\mathbf{c}_k\mathbf{t}_{ijk2}$ as the tetrahedrons of intersection. For any point inside T_{ijk1}, the closest boundary point, not buried by a_i, a_j or a_k, is exactly $\mathbf{t}_{ijk1}$; and similarly for T_{ijk2}.

Let $exposed^M(\mathbf{x})$ is an indicator function which is true if $\mathbf{x}$ is not inside the SAS of any atom of M. Let $closest(\mathbb{P}, \mathbf{x})$ define the closest point $\mathbf{y}$ belonging to the set of the point $\mathbb{P}$ from the given point $\mathbf{x}$; and let $d(\mathbb{P}, \mathbf{x})$ be the Euclidean distance between them if $\mathbf{y}$ is exposed, otherwise it is Rr_p, where $R > 1$ is some constant.

Now, for each point $\mathbf{x}$, let $N^A(\mathbf{x})$ be a set of atoms such that $\mathbf{x}$ is inside the SAS of the atoms; $N^{CI}(\mathbf{x})$ be a set of cone intersections that contain $\mathbf{x}$; and $N^T(\mathbf{x})$ be a set of tetrahedral intersections that contain $\mathbf{x}$. Let n^A, n^{CI} and n^T be the cardinalities of these sets. Finally, we define d^A, d^{CI} and d^T as follows-

- $d^A = min_{a_i \in N^A} d(a_i, \mathbf{x})$
- $d^{CI} = min_{CI_{ij} \in N^{CI}} d(C_{ij}, \mathbf{x})$
- $d^T = min_{T_{ijkl} \in N^T} d(\mathbf{t}_{ijkl}, \mathbf{x})$

We are now ready to define the distance function.

- If $\mathbf{x}$ is exposed, then $\delta^M_{SAS}(\mathbf{x}) = min_i |\|\mathbf{c}_i - \mathbf{x}\| - r_i - r_p|$
- If $\mathbf{x}$ is not exposed, then $\delta^M_{SAS}(\mathbf{x}) = min(d^A, d^{CI}, d^T)$

Note that efficient computation of this function requires efficient computation of $exposed^M(\mathbf{x})$, $N^A(\mathbf{x})$, $N^{CI}(\mathbf{x})$ and $N^T(\mathbf{x})$. We use the Dynamic Packing Grid [4, 3] data structure to compute these in quasi-constant time. In Section 4, we shall discuss its augmentation for the purposes of efficient computation of SES.

3 Dynamic adaptive grids for molecular surface computation

Given a molecule M, our algorithm constructs an adaptive octree T. The octree is subdivided at a higher resolution near the boundary of the molecule and is coarsely subdivided elsewhere. The dual of the highest resolution cells of the octree is contoured based on a user-defined isovalue to generate a smooth surface approximation of M. We define the resolution of the mesh as the resolution of the dual grid, which is equal to the resolution of the smallest cells (d_{min}) of the octree. The construction runs in $\mathcal{O}(n \log n)$ time provided that d_{min} is linearly related to the maximum radius of the atoms. Refer to Sections 3.1 and 3.2 for details about the model and the construction.

We define an update as adding, removing or moving an atom a_i. In Section 3.4 we discuss a $\mathcal{O}(\log n)$ time algorithm for locally updating the octree while maintaining the adaptive subvision property.

3.1 The octree and its dual

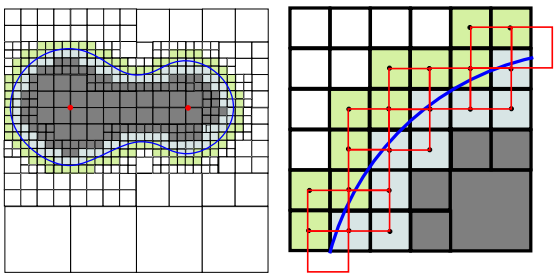
We define an octree node u as a 3-dimensional cube with a specific length, a center and a function value. The length $\textsc{length}(u)$ is the Euclidean length of an edge of the cube, $\textsc{box}(u)$, representing the node. The center $\textsc{center}(u)$ is the geometric center of the cube. The function value $\textsc{val}(u)$ is defined as a volumetric function $\Phi^M(\textsc{center}(u)))$, where M is the set of atoms. We define the sign of a node $\textsc{sign}(u) = 1$ if $\textsc{val}(u) - isoValue$ is positive and -1 otherwise. Refer to Section 2 for possible choices of $\mathcal{F}$ and corresponding iso-values.

By $\textsc{child}(u)$ we denote the set of non-empty octree nodes obtained by subdividing node u, and $\textsc{parent}(u)$ points to the parent of u in the octree. $\textsc{child}(u) = \textsc{nil}$ if the node is a leaf of the octree and $\textsc{parent}(u) = \textsc{nil}$ if u is the root. For convenience, we set $\textsc{leaf}(u)$ to $\textsc{true}$ if u is a leaf, and $\textsc{false}$ otherwise. $\mathcal{N}(u)$ denotes the neighbor cells u. $\textsc{depth}(u)$ denotes the depth of u. Each leaf u of the octree stores a list of atoms $\textsc{atom}(u)$ which intersect its bounding box.

The octree T contains a reference to the root node of the tree, a list of atoms M, a positive number d_{min} specifying the minimum acceptable length of any node of the octree, and a number $isoValue$ corresponding to the level-set of $\mathcal{F}$ representing the molecular surface.

Given M, d_{min} and $isoValue$, the octree is constructed by iteratively refining (splitting) nodes which contain the contour corresponding to the

specified $isoValue$. Other parts of the octree remain coarse. The refinement continues until $d_{min} \leq \text{LENGTH}(v) \leq 2 * d_{min}$ for every node v containing the contour. Let $\text{BOUNDARY}(T)$ define the set of all such nodes. We provide a simple 2D example in Figure 4(a).



(a) (b)

Figure 4: **(a)** A 2D example of adaptive decomposition. Cells are adaptively subdivided depending on their distance from the boundary. In the figure, we color the a cell u- (i)light green if $u \in \text{BOUNDARY}(T)$ and $\text{SIGN}(u) = 1$, (ii) light blue if $u \in \text{BOUNDARY}(T)$ and $\text{SIGN}(u) = -1$, (iii) dark gray if $u \notin \text{BOUNDARY}(T)$ and $\text{SIGN}(u) = 1$, and (iv) white if $u \notin \text{BOUNDARY}(T)$ and $\text{SIGN}(u) = -1$. Atom centers are shown in red and the isocontour representing the molecular surface is shown in blue. (b) A zoomed in view of a part of the octree with the dual grid superimposed on it. Note that, the dual grid is very sparse, and yet sufficient to compute the contour.

We construct a dual grid G based on the nodes in $\text{BOUNDARY}(T)$. The length of each cell in the dual grid is equal to $\text{LENGTH}(v)$ where $v \in \text{BOUNDARY}(T)$. The vertices/grid-points of the dual grid are made congruent to the center of the nodes in $\text{BOUNDARY}(T)$. See Figure 4(b) for an example dual grid superimposed on an octree. Each grid-point gp of G is assinged a value equal to $\text{VAL}(w)$, where $gp = \text{CENTER}(w)$. Finally, we use fast marching cubes to produce a mesh for the isocontour in each of the gridcells of G.

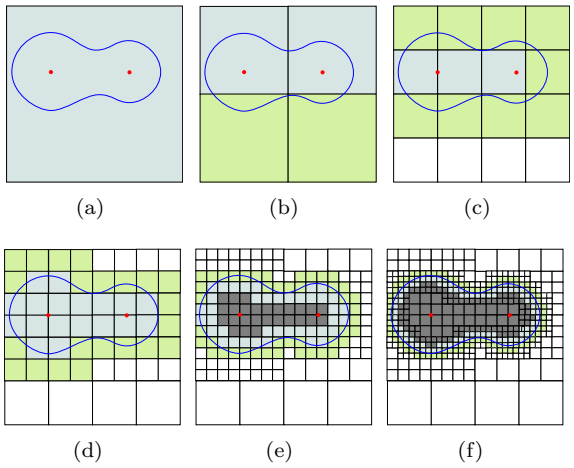
Theorem 3.1. *The dual of a dynamic octree containing a molecule with n atoms has $\theta(k)$ cells provided that $d_{min} = \Theta(r_{max})$ where r_{max} is the radius of the largest atom in M and $k < n$ is the number of atoms on the boundary of the molecule.*

Proof. Since $d_{min} = \Theta(r_{max})$, each atom intersects a constant number of octree leaves at the lowest level (see Theorem 2.1 and Lemma 2.1 in [4]). So, if k atoms contribute to the molecular surface, the number of such leaves can be at most Ck, where C is a constant. Hence, the number of dual cells is $\theta(k)$. ■

3.2 Construction

Given a molecule M, a positive number d_{min} specifying the expected resolution of the dual grid and the isovalue for the surface, we initialize an Octree T with a single node u (root of T), such that $\text{BOX}(u)$ is large enough to contain all of the atoms. All atoms are inserted into u and $\text{VAL}(u)$ is computed.

The octree is refined by iteratively subdividing cells which are on the boundary of the molecule. Since we define the boundary as a isocontour $\Gamma(\Phi^M, isoValue)$, the boundary intersects an edge (v_i, v_j) of dual cell gc if and only if $\text{SIGN}(v_i)$ and $\text{SIGN}(v_j)$ are different, where v_i and v_j are neighboring octree cells. We consider all such (v_i, v_j) pairs as being on the boundary and mark them for subdividing.



(a) (b) (c)

(d) (e) (f)

Figure 5: Figure (a) shows the initial octree, which is iteratively refined and the results of each successive iterations are shown in Figues (b)-(f). We follow the same coloring convention used in Figure 4. Note that only the cells belonging to $\text{BOUNDARY}(T)$ are split in the next iteration.

To split a node u, we create its children $u_1, \ldots, u_8$. Then, we update the list of atoms in the children as $\text{ATOMS}(v_i) = \{a | a \in \text{ATOMS}(u) \& (\text{BOX}(v_i) \cap a) \neq \phi$. The intersection detection is performed using the Dynamic Packing Grids data structure in constant time per atom. The number of times an atom is checked for intersection is bounded by the maximum depth of the octree. After assigning the atoms, we compute $\text{VAL}(v_i)$.

Signs of each of the newly created nodes are compared with their neighbors to identify pairs of nodes on the boundary and marked for splitting. The process continues until for each leaf cell v, $d_{min} \geq \text{LENGTH}(v)$.

Special note on initialization Let u and v be two neighboring cells of the octree such that $\text{SIGN}(u) \neq \text{SIGN}(v)$, then the isocontour crosses the line connecting $\text{CENTER}(u), \text{CENTER}(v)$ an odd number of times (and at least once). Otherwise, the isocontour crosses the line an even number of times (or never). So, if we define the minimum topological feature size $\epsilon_{\Phi^M, isoValue}$ (eg. width of a pocket/tunnel) of the molecule M under the given function and isoValue, then no grid-based algorithm whose finest level grid-size $d_{min} > \epsilon_{\Phi^M, isoValue}$ can produce a topologically accurate surface. So, we assume that $epsilon_{\Phi^M, isoValue}$ is known or the user specifies a specific value ϵ such that s/he is willing to tolerate errors below that threshold.

We incorporate this into the algorithm by ensuring that the octree is uniformly subdivided until every leaf has length at most ϵ. Let the minimum depth required to achieve this is l_{min}. Note that after this point, we continue to subdivide adaptively near the boundary until the lowest level leaves $d_{min} \geq \text{LENGTH}(v)$.

Contouring Once the octree is constructed, the surface is computed by dual contouring (see previous section), based on the cells corresponding to any level (greater than l_{min}) of the octree, with progressively finer (deeper) level nodes providing progressively better approximation of the surface.

Figure 5 provides a detailed example of the octree construction algorithm.

3.3 Analysis

Our grid-based algorithm is a discrete approximation of a continuous domain (the isocontour). Hence it does not generate a perfect representation as analytical methods would produce. However, we can provide theoretical bounds on the topological and geometric errors.

Theorem 3.2. *Dual contouring is performed using gridcells at any level $l > l_{min}$ of the octree produces a isocontour with bounded geometric error if $\epsilon_{\Phi^M, isoValue} > d_{min}$.*

Proof. There can be three types of error. The severest error is like the one shown in Figure 6(a), where very small features lie on in between grid centers and hence discrete evaluation of Φ^M at the grid-centers fail to detect such features. In molecular applications, however, typically the feature sizes are bounded by the smallest radius. There can sometimes be small tunnels and pockets whose width is less than the smallest radii, but for most purposes such tunnels are inaccessible to other molecules including water, and hence loss of such features will not affect the molecular properties calculation too much. The second type of error can that happen is when the isocontour goes through a dual cell which is not identified, but its neighbor is identified (see Figure 6(b)). In this case, the maximum distance between the actual and the approximate contour is at most d_{min}. Since, we assumed that the smallest topological feature size $\epsilon_{\Phi^M, isoValue} > d_{min}$, this error is only a geometric error. Finally, the last type of error is simply due to the limitation of contouring where the correct dual cells are contoured, but the approximated contour inside a dual cell is slightly off the real isocontour within the same cell. Here, the maximum error, in terms of Hausdorff distance, can be at most $d_{min}/2$. The proof follows. ■

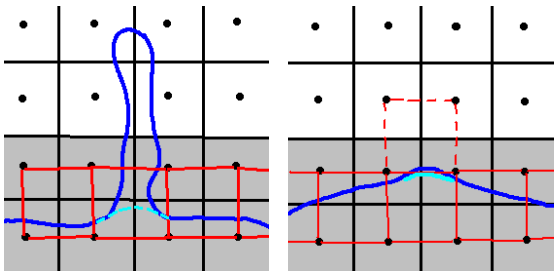


Figure 6: **(a)** Topological error. If the minimum dimension of such a topological feature (like the tunnel in the figure) is less than ϵ, then it is lost during the contouring. **(a)** Geometric error. The approximate isocontour (light blue) is only slightly off the actual isocontour (dark blue). The error is bounded by the cell size.

Theorem 3.3. *A dynamic octree containing a molecule with n atoms is constructed in $\mathcal{O}(n \lg n)$ time if $d_{min} = \Theta(r_{max})$.*

Proof. Since, $d_{min} = \Theta(r_{max})$, following Theorem 2.1 and Lemma 2.1 in [4], the number of cells in the lowest level (with length d_{min}) is $\mathcal{O}(n)$. By the same logic, the number of nodes at any level above that is also bounded by $\mathcal{O}(n)$. Finally, since the maximum dimension of the bounding box is at most nr_{max} assuming that the molecule M is compact,

the depth of the octree is $\log \frac{nr_{max}}{d_{min}} = \log n$.

At each node, the decisions to split/merge takes constant time, computing the function value requires summing over the atoms intersecting the cell and may require $BigOh(n^3)$ time per node at worst. This is addressed in section 4 to make sure that only a constant number of atom is required to compute the value at a cell. Finally, during split, we must go over the list of atoms and copy them to new nodes, indicating a worst case $\mathcal{O}(n)$ time per node. However, note that evergy atom will be copied/moved at most $\log n$ time during the entire construction, hence the cost is amotized to $\mathcal{O}(1)$ per node.

Hence the overall amostized cost of construction is $\mathcal{O}(n \lg n)$. ■

3.4 Update

When an atom is updated from a to a' by either changing the position or the radius, we must update the function values $\text{VAL}(v)|v \in T\&(\text{BOX}(v) \cap (a \cup a')) \neq \phi$. For each updated cell v, we verify whether it should be split or merged. An internal node v is marked for merging if all of its children have the same sign and none of them are candidates for splitting. Once a cell is marked for merging, it cannot be split again for the same update operation. The decision to split is made based on the same criteria described in Section 3.2. We iteratively split cells and identify new cells for splitting/mergning until no more cells are marked for splitting. Then we merge the cells marked for such.

Note that, if the update of the atom does not change the boundary of the molecule, then our algorithm would only update the function values, but would not split/merge any cells.

Theorem 3.4. *After completion of* UpdateOctree(a, a')*, if dual contouring is performed using gridcells at any level $l > l_{min}$ of the octree produces a isocontour with bounded geometric error if $\epsilon_{\Phi^M, isoValue} > d_{min}$.*

Proof. The accuracy of the merge operation (and decision) follows from the accuracy of split operation (and decision) discussed in Theorem 3.2, since the merger is performed if only if the merged cell would not have been marked for splitting if we had started constructing the octree with the moved atom in the new position. ■

Theorem 3.5. UpdateOctree(a, a') *completes in $\mathcal{O}(\lg n)$ time for a dynamic octree containing a molecule with n atoms.*

Proof. A single atom intersects at most a constant number of leaf cells. Hence, the total number of nodes affected by the move at any level of the octree is bounded by the depth $\mathcal{O}(\log n)$. The proof follows, since the algorithm spends $\mathcal{O}(1)$ time per node (see proof of Theorem 3.3). ■

4 Augmented Dynamic Packing Grids for SES Molecular Surface Computation

We describe a data structure and algorithm which supports $O(1)$ time updates (add/remove/move an atoms) such that the molecular surface after the update is consistent. As a by-product, the algorithm also updates the list of exposed atoms and boundary cells (SAS).

The algorithm is an augmentation of the octree based surface construction algorithm for the purpose of computing the Solvent Excluded Surface. As we have dicussed in Section 2, SES computation requires evaluation of the signed distance function and the computation is slightly more involved that than the sum of Gaussian kernels. The octree always uses all atoms inside a cell to define the function values in the cell, but this becomes extremely costly when $\theta(n)$ atoms are in a cell, during the initial stages of the construction and leads to $O(n^3)$ computations for the double and triple intersections. Here, we propose to use a mixed model, where a simple uniform grid G with a constant grid spacing of $r_p/2$ is created initially, but the SDF is not computed on the grid directly. Rather, we also create separate Dynamic Packing Grids (DPG) with a constant grid spacing of $2r_{max}$ and use the DPGs to label gridpoints and gridcells of G. We show that the labeling is sufficient to identify all the gridcells where the SES boundary lies, and also the atoms which contribute to the SES boundary. Once this classification is achieved, the regular octree based refinement can be performed only at the boundary cells and the number of atoms that contributes to the boudary and intersects a given cell is also reduced to a constant, and hence the overall construction time reduces from $O(n^3)$ to $O(n \log n)$, without any loss of accuracy. Furthermore, we show that the labeling can be updated under dynamic motions of the atoms in constant time per atom.

4.1 Notations

Let the molecule M is represented as a collection of atoms a_i and each atom is represented using a

center $\vec{c}_i$ and radius r_i. Let the radius of the probe be r_p. Let the dynamic adaptive grid be defined as a set of gridpoints and a set of gridcells. We also assume that each grid cell contains exactly 8 gridpoints.

A grid point gp is marked as V_{VDW} if it is inside the vdW surface of at least one atom, is marked V_{SAS} if it is not inside the vdW surface of any atoms, but inside SAS surface of at least one atom, and is marked V_{OUT} if it is not inside the SAS of any atoms. We mark a gridcell gc as $C_{BURRIED}$ iff $\forall gp_j \in gc, (gp_j \in V_{VDW})$, as C_{SAS} iff $\exists gp_j \in gc, (gp_j \in V_{VDW}) \wedge \exists gp_k \in gc, (gp_k \in V_{SAS})$, as C_{BAND} iff $\forall gp_j \in gc, (gp_j \in (V_{SAS} - V_{VDW}))$, C_{VDW} iff $\exists gp_j \in gc, (gp_j \in (V_{SAS} - V_{VDW})) \wedge \exists gp_k \in gc, (gp_k \notin V_{SAS})$, and C_{OUT} iff $\forall gp_j \in gc, (gp_j \notin V_{SAS})$. See Figure 7 for a 2D example.

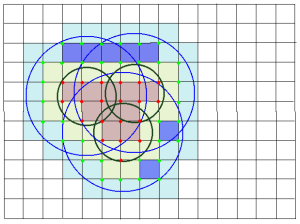


Figure 7: A 2D example grid with uniform grid spacing showing how the gridpoints and the gridcells are labeled based on thier positions with respect to the vdW and SAS surfaces. SAS boundary of the atoms are shown as blue lines, vdW boundary of atoms are shown as dark green lines. Gridpoints labeled vdW, SAS and OUT are marked red, green and unmarked in the figure. Gridcells labeled $C_{BURRIED}$, C_{VDW}, C_{BAND}, C_{SAS} and C_{OUT} are shaded brown, light green, dark blue, light blue and white.

If the SAS boundary of an atom a_i contains at least one gridpoint marked as V_{SAS}, then a_i is marked as an exposed atom (see Lemma 4.1 for details).

4.2 Algorithm Sketch

During an update, the algorithm uses DPG to identify and re-classify the affected grid-points (as $V_{VDW}/V_{SAS}/V_{OUT}$) and then uses these classification to mark atoms and gridcells. The algorithm maintains 4 separate packing grids, namely, (i) for all atoms, (ii) for exposed atoms, (iii) for grid-points marked as V_{SAS}, and (iv) for gridcells marked as C_{SAS}.

Add When an atom is added (cf. Figure 8), it can have the following effects-

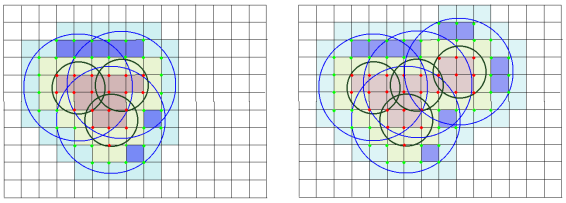


Figure 8: Updating the labeling of the grid points and gridcells when an atom is added. (left) before adding, (right) after adding.

- Some grid-points might be marked as V_{SAS}. This is reflected in steps 2-3 of the pseudocode for the *Add* method.
- Some grid-points might be marked as V_{VDW}. This is reflected in steps 4-5 of the pseudocode for the *Add* method.
- The new atom will be marked either buried/exposed. The new atom is exposed if it marks a gridpoint as V_{SAS} (i.e. it contributes to the surface). This is handled by step $3(a)iii$ of the pseudocode.
- Some exposed atoms might get buried. Only exposed atoms intersecting the current atom are affected. If any of those exposed atoms no longer contributes to the surface (does not contain any gridpoint marked V_{SAS} within its solvent enlarged volume), then it is marked as buried (see step 6 of the algorithm for *Add*).

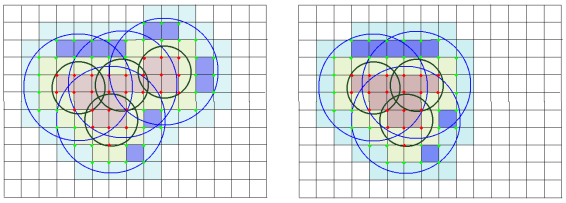


Figure 9: Updating the labeling of the grid points and gridcells when an atom is added. (left) before removing, (right) after removing.

Remove When an atom a_i is removed (cf. Figure 9), it can have the following effects-

- Some gridpoints marked as V_{SAS} can become V_{OUT}. (see step 4a of *Remove*).
- Some gridpoints marked as V_{VDW} can become V_{SAS}. And a previously buried atom which contains the gridcell within its solvent inflated volume would become exposed. Step 4b of the algorithm handles this case.

Updating the surface Any change in the classification of grid-points immediately affects the classification of the eight cells neighboring that point. Once a cell's classification is updated the contour/mesh inside the cell need to be recomputed. We use three auxiliary lists C_{rem}, C_{add} and C_{mod} to facilitate the management of the update. C_{rem} is the list of cells whose classification changed from C_{SAS} to something else. C_{add} are the cells whose classification changed from something else to C_{SAS}. C_{mod} contains cells whose classification remain C_{SAS}, but the surface inside the cell needs to be modified (re-contour) to account for the recent movement of atoms.

4.3 Analysis

Correctness

Lemma 4.1. *In a grid with a grid spacing of $r_p/2$, an atom a_i is buried if $S_{SAS}(a_i)$ does not contain any grid-point marked as V_{SAS}.*

Proof. First of all, if the grid spacing is $r_p/2$, then it is easy to show that if one corner of a grid-cell is marked as V_{VDW}, then no part of the cell is outside the S_{SAS}.

$S_{SAS}(a_i)$ can intersect a cell in two different ways. Either the intersection contains at least one corner of the cell or it does not contain any corners.

If the intersection contains a corner, then that corner must be marked as V_{VDW}. And since all of these cells are completely inside the S_{SAS} (before adding a_i), a_i cannot have exposed surface inside these cells.

On the other hand if $S_{SAS}(a_i)$ intersects only a face f of a cell G_k and does not contain any corner, then clearly it has to contain at least one corner g of the neighboring cell G_l on the opposite side. According to the assumption of the lemma, g is marked V_{VDW} and whichever atom a_j has g inside its VDW surface, has the entire face f buried inside its SAS and hence the portion of a_i inside G_k is also completely buried ubsude the SAS of inside a_j.

Hence, a_i is not exposed inside any cell it intersects. In other words, it is buried. ■

Lemma 4.2. ***Add***(a_i, c_i, r_i) *and* ***Remove***(a_i, c_i, r_i) *correctly updates the list of solvent exposed atoms.*

Proof. An atom a_j is marked buried only if $S_{SAS}(a_j)$ contains no gridpoints marked as V_{SAS} (see step 5(c) of *Add* method). And an atom a_j is marked as exposed as soon as $S_{SAS}(a_j)$ has at least one gridpoint marked as V_{SAS} (see step 2(a)i of *Add* and step 3(b)i of *Remove*). The correctness follows from Lemma 4.1. ■

Lemma 4.3. ***UpdateCells***(g) *correctly updates the lists C_{rem}, C_{add} and C_{mod}.*

Proof. Trivially follows from the definitions and the algorithm. ■

Time Complexity

Lemma 4.4. ***UpdateCells***(g) *operate in constant time.*

Proof. There are exactly 8 cells neighboring g and to classify each cell, the classification of exactly 8 gridpoints need to be checked. So, the total number of operations is constant. ■

Lemma 4.5. ***Add***(a_i, c_i, r_i) *and* ***Remove***(a_i, c_i, r_i) *operate in constant time.*

Proof. Each range query of DPG runs in expected time $O(lglgh + k)$ where h is the word size in the memory, and k is the number of results returned (see Theorem 2.3 in [4]). h is clearly constant. The value of k depend on the type of the query. We discuss each of them separately.

A query in $DPG_{expatom}$ or DPG_{atom} with a distance cutoff of $O(r_{max})$ always returns $O(1)$ results (see Theorem 2.1 in [4]).

A query in DPG_{grid} or $DPG_{sasgrid}$ would return $O(\frac{r_max+r_p}{gridspacing})^3$. Provided that the grid spacing is $O(r_p)$ and $r_p \approx r_{max}$, the number of results returned becomes $O(1)$.

Hence all the queries runs in constant time and also returns constant number of results. Which means all internal loops run in constant time as well and hence the total complexity remains constant.

The rest follows from Lemma 4.4. ■

5 Results

5.1 Construction Accuracy

We have previously provided theoretical proofs that only the cells marked as boundary cells need to be contoured to get an airtight surface. In Figure 10, we provide practical examples showing that the SES is indeed correctly recovered.

The generated smooth solvent excluded surfaces using MSMS [26] for the molecules in Zlab's non-redundant protein benchmark. For the same dataset, we also applied our dynamic octree algorithm to compute approximate SES. Since MSMS

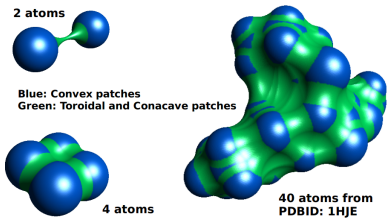


Figure 10: 3 separate examples with 2, 4 and 40 atoms showing the results of SES computation using the DAG. Notice that our level set of signed distnace function formulation correctly produced the re-entrant patches (green) of the SES.

computes an analytical representation of the SES, it is considered the ground truth in this context, and we report the error in terms of surface area, volume and Hausdorff distance between surfaces produced by MSMS and our algorithm (see Figure 11). We found that the average errors for area and volumes are 2.7% and 1.53% respectively. The average distance between the surfaces are 0.255Å.

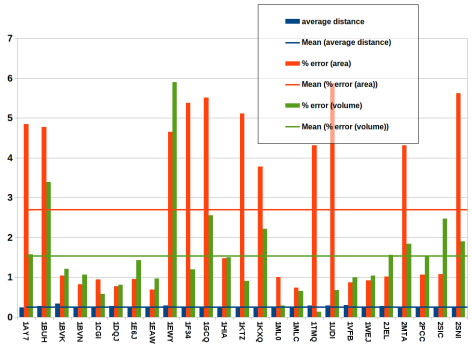


Figure 11: The plot compares the surface area, volume and average distance of the solvent excluded surfaces (SES) for various proteins generated by our DAG data structure to those generated by analytically modeling the SES as a collection of spherical and toroidal patches.

5.2 Construction Time

We compared the time required to compute volume representation and isovalue based surface construction using regular uniform grids, and octrees such that the grid-size of the lowest level of the octree matched the grid-size of the regular grid. Note that, the accuracy/quality of the surfaces produced from both would be similar, but the runtime is expected to be different. The regular grid would have to compute the function values at more positions, on the other hand an irregular data structure like an octree incurs some maintenance overhead. In our experiment (see Figures 12 and 13) on the dataset mentioned above, we found that the dynamic octree requires less time and less memory than the regular grids.

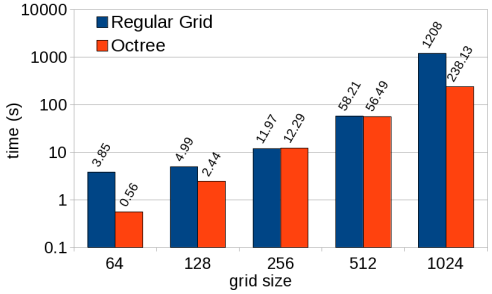


Figure 12: Comparison of the construction time using regular grids and adaptive grids. The numbers on the x-axis represent the resolution of the grid in terms of the number of gridpoints on each direction. The octree based construction clearly outperforms regular grid based construction. Note that this plot shows the runtimes for Gaussian surfaces.

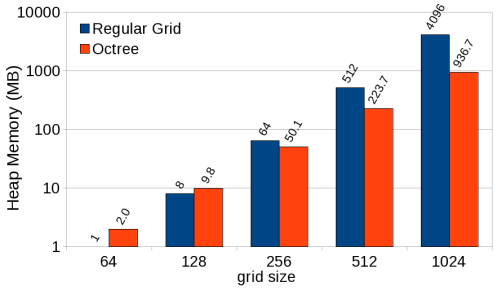


Figure 13: Comparison of the memory requirement of regular grids and adaptive grids.

5.3 Updates

Theoretically, each update operation runs in $O(\log n)$ time and the overall construction from the scratch runs in $O(n \log n)$ time (see Theorems 3.3 and 3.5). This would indicate that it is always preferable or at least equivalent to construct the octree using a sequence of updates. However, the runtime for update has a larger constant which makes it slightly slower. For a large set of proteins, we moved different number of atoms and applied the update operation to recompute/update the surface to identify a threshold where reconstruction from scratch becomes more attractive than locally updating the surface. We found that even when upto 6% of the atoms are moved, dynamic update provides 4 times faster performance than reconstruction. In fact, reconstruction only becomes a better

choice if more than 10% of the atoms are moved (see Figure 14.

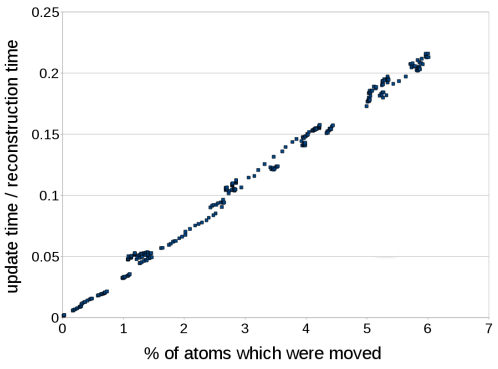


Figure 14: The plot compares the time required to dynamically update the surface when a fraction of the total atoms are moved, vs, the time required to reconstruct the entire surface.

5.4 Choice of blobbiness and isovalue for Gaussian surfaces

Gaussian representation of the molecular surface is attractive for several reasons including the higher degree of continuity, simple computation (as opposed to computing arrangment of spheres for SES), and an intuitive relationship of the Gaussian blurring model to electron scattering of molecules, making it a better choice for energetics and force calculations. However, for shaphe based analysis or comparison of molecules, an SES approximation seems more favorable. We postulate that for such applications, it is possible to choose appropriate values for the Gaussian paramters to make the isosurface as close to the SES as possible. Similar ideas and analysis were also reported in [34] and [8]. In [34], in particular it was reported that a blobbiness (β) of 0.9 was a good choice for shape complementarity analysis. In this paper, we compared the Gaussian surfaces at different blobbiness and isovalues to their SES counterparts, in terms of surface area, volume, average distance and Hausdorff distance. Note that, given two surfaces A and B represented as a collection of points, we define the average distance as $\frac{1}{|A|}\sum_{p\in A} min_{q\in B} dist(p,q)$ and the Hausdorff distance as $max_{p\in A} min_{q\in B} dist(p,q)$.

We found that low blobbiness is better for area, and Hausdorff distances. Average distance is also low for low blobbiness, if a large enough isovalue is selected. For instance, a blobbiness of 1 and isovalue of 2.0 provides low errors in all three terms. However, to minimize volume error, an even larger isovalue might be chosen. Interestingly, there are other choices which provide similar error bounds for volume and average distace. But we believe the area and Hausdorff distance provides a more reliable estimate of the closeness of the surfaces.

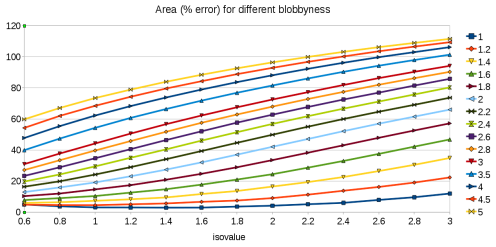


Figure 15: Difference, in terms of area, between Gaussian and SES surfaces for different choices of blobbiness and isovalues. The different lines represent different choices of blobbiness, then for each blobbiness the x-axis represents different isovalues and the y-axis represents the percentage error.

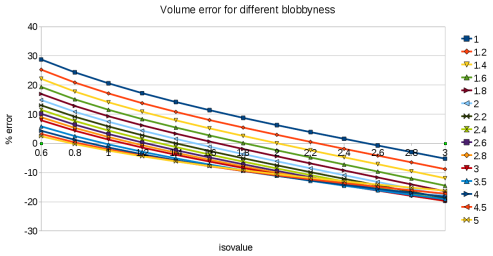


Figure 16: Difference, in terms of volume, between Gaussian and SES surfaces for different choices of blobbiness and isovalues. The different lines represent different choices of blobbiness, then for each blobbiness the x-axis represents different isovalues and the y-axis represents the percentage error.

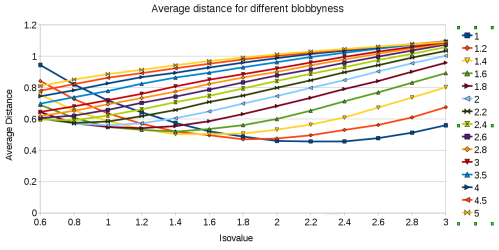


Figure 17: Difference, in terms of average distance between surfaces, between Gaussian and SES surfaces for different choices of blobbiness and isovalues. The different lines represent different choices of blobbiness, then for each blobbiness the x-axis represents different isovalues and the y-axis represents the percentage error.

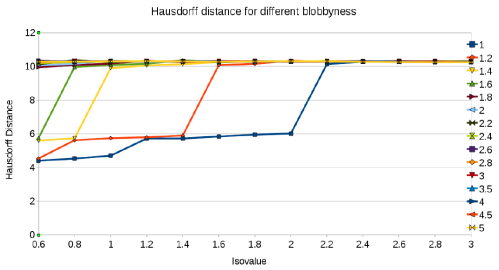


Figure 18: Difference, in terms of Hausdorff distance, between Gaussian and SES surfaces for different choices of blobbiness and isovalues. The different lines represent different choices of blobbiness, then for each blobbiness the x-axis represents different isovalues and the y-axis represents the percentage error.

References

[1] N. Akkiraju and H. Edelsbrunner. Triangulating the surface of a molecule. *Discrete Applied Mathematics*, 71(1-3):5–22, 1996.

[2] C. Bajaj, B. Bauer, R. Bettadapura, and A. Vollrath. Nonuniform fourier transforms for rigid-body and multi-dimensional rotational correlations. *SIAM Journal of Scientific Computing*, 35(4):B821–B845, 2013.

[3] C. Bajaj, R. A. Chowdhury, and M. Rasheed. A dynamic data structure for flexible molecular maintenance and informatics. In *SIAM/ACM Conf. Geom. Phys. Model.*, pages 259–270, 2009.

[4] C. Bajaj, R. A. Chowdhury, and M. Rasheed. A dynamic data structure for flexible molecular maintenance and informatics. *Bioinformatics*, 27(1):55–62, 2011.

[5] C. Bajaj, H. Y. Lee, R. Merkert, and V. Pascucci. Nurbs based b-rep models for macromolecules and their properties. In *Proceedings of the fourth ACM symposium on Solid modeling and applications*, pages 217–228. ACM Press, 1997.

[6] C. Bajaj, V. Pascucci, A. Shamir, R. Holt, and A. Netravali. Multiresolution molecular shapes. Technical report, TICAM, Univ. of Texas at Austin, Dec. 1999.

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

[8] C. Bajaj, G.-L. Xu, and Q. Zhang. Higher-order level-set method and its application in biomolecular surfaces construction. *Journal of Computer Science and Technology*, 23(6):1026–1036, November 2008.

[9] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[10] S. F. Boys. Electronic wave functions. I. a general method of calculation for the stationary states of any molecular system. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 200(1063):542–554, 1950.

[11] R. Chen and Z. Weng. A novel shape complementarity scoring function for protein-protein docking. *Proteins: Structure, Function, and Genetics*, 51(3):397–408, 2003.

[12] R. Chowdhury, M. Rasheed, D. Keidel, M. Moussalem, A. Olson, M. Sanner, and C. Bajaj. Protein-protein docking with f2dock 2.0 and gb-rerank. *PLoS ONE*, 8(3):e51307, 2013.

[13] M. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221(4612):709–713, 1983.

[14] R. Diamond. A real-space refinement procedure for proteins. *Acta Crystallographica Section A*, 27:436–452, 1971.

[15] B. S. Duncan and A. J. Olson. Approximation and characterization of molecular surfaces. *Biopolymers*, 33:219–229, 1993.

[16] H. Edelsbrunner and E. Mucke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[17] E. Eyal and D. Halperin. Dynamic maintenance of molecular surfaces under conformational changes. In *SCG '05: Proceedings of the 21st Annual Symposium on Computational Geometry*, pages 45–54, 2005.

[18] E. Eyal and D. Halperin. Improved maintenance of molecular surfaces using dynamic graph connectivity. *Algorithms in Bioinformatics*, pages 401–413, 2005.

[19] R. R. Gabdoulline and R. C. Wade. Analytically defined surfaces to analyze molecular interaction properties. *J. Mol. Graph.*, 14:341–353, 1996.

[20] J. A. Grant, M. A. Gallardo, and B. T. Pickup. A fast method of molecular shape comparison: A simple application of a Gaussian description of molecular shape. *Journal of Computational Chemistry*, 17(14):1653–1666, 1996.

[21] J. A. Grant and B. Pickup. A Gaussian description of molecular shape. *Journal of Phys. Chem.*, 99:3503–3510, 1995.

[22] D. Kozakov, R. Brenke, S. Comeau, and S. Vajda. PIPER: An FFT-based protein docking program with pairwise potentials. *Proteins: Structure, Function, and Bioinformatics*, 65(2):392–406, 2006.

[23] B. Lee and F. Richards. The interpretation of protein structures: estimation of static accessibility. *Journal of Molecular Biology*, 55(3):379–400, 1971.

[24] F. Richards. Areas, volumes, packing, and protein structure. *Ann. Rev. of Biophysics and Bioengineering*, 6:151–176, 1977.

[25] M. Sanner, A. Olson, and J. Spehner. Fast and robust computation of molecular surfaces. In *Proceedings of the 11th Annual Symposium on Computational Geometry*, pages 406–407. ACM Press, 1995.

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

[27] A. Varshney, F. P. Brooks Jr., and W. V. Wright. Computing smooth molecular surfaces. *IEEE Computer Graphics Applications*, 14(5):19–25, 1994.

[28] R. Voorintholt, M. T. Kosters, G. Vegter, G. Vriend, and W. G. Hol. A very fast program for visualizing protein surfaces, channels and cavities. *Journal of Molecular Graphics*, 7(4):243–245, December 1989.

[29] J. Weiser, P. Shenkin, and W. Still. Approximate atomic surfaces from linear combinations of pairwise overlaps (LCPO). *Journal of Computational Chemistry*, 20(2):217–230, 1999.

[30] J. Weiser, P. S. Shenkin, and W. C. Still. Fast, approximate algorithm for detection of solvent-inaccessible atoms. *Journal of Computational Chemistry*, 20(6):588–596, 1999.

[31] J. Weiser, A. A. Weiser, P. S. Shenkin, and W. C. Still. Neighbor-list reduction: Optimization for computation of molecular van der Waals and solvent-accessible surface areas. *Journal of Computational Chemistry*, 19(7):797–808, 1998.

[32] T. You and D. Bashford. An analytical algorithm for the rapid-determination of the solvent accessibility of points in a 3-dimensional lattice around a solute molecule. *Journal of Computational Chemistry*, 16(6):743–757, 1995.

[33] Z. Yu, M. P. Jacobson, and R. A. Friesner. What role do surfaces play in GB models? a new-generation of surface-generalized Born model based on a novel Gaussian surface for biomolecules. *J. Comput. Chem.*, 27:72–89, 2006.

[34] Q. Zhang, M. Sanner, and A. Olson. Shape complementarity of protein–protein complexes at multiple resolutions. *Proteins: Structure, Function, and Bioinformatics*, 75(2):453–467, 2009.

Algorithm Details

Algorithms for Constructing Octree

ConstructOctree($Molecule, d_{min}, \epsilon, isoValue$)
{*Constructs a octree for the atoms in set* $Molecule$}

Inputs: $Molecule$ is a set atoms in 3-dimensions. d_{min} is a positive number specifying the minimum length of a cell of the octree. ϵ is the expected minimum feature size. $isoValue$ is a real number used to define the surface.
Output: This routine constructs a dynamic octree from the atoms in $Molecule$. The octree is adaptively refined based on function values in cells and the given $isoValue$, while maintaining the constraint that Length(u) $\geq d_{min}$ for all cells in the octree. Where, Box(u) denotes the cube corresponding to any octree node (cell) u and Length(u) denotes the length of a side of Box(u).

1. create an octree $\mathcal{T}$ with a single node u as the root
2. make Box(u) large enough to contain all atoms in $Molecule$
3. $minDepth \leftarrow \log_2(\text{length}(u))$
4. depth(u) $\leftarrow 0$
5. intialize set $newLeaves \leftarrow \{u\}$
6. intialize set $markedNodes \leftarrow \phi$
7. ***while*** MarkForRefinement($newLeaves$, $minDepth, isoValue, markedNodes$)
{*Finds nodes for splitting*}
8. clear $newLeaves$
9. ***for*** each node $v \in markedNodes$
10. Split($v, d_{min}, newLeaves$)
{*refines the octree and populates* $newLeaves$}
11. ***return*** $\mathcal{T}$

Figure 19: Algorithm for the construction of a dynamic octree from a given set M of atoms in 3D

MarkForRefinement ($newLeaves, minDepth$, $isoValue, markedNodes$)
{*Identifies all nodes which should be split*}

Input: A set of newly created octree leaves $newLeaves$, the $isoValue$ and a boolean $contourDetected$ denoting whether the the current octree contains a contour. If depth of a node is less than $minDepth$, then it is automatically marked for splitting. Otherwise, $isoValue$ is used to mark nodes for splitting. **Output:** Populates the $markedNodes$ set of nodes which should be split. Returns true if such nodes are found.

1. $refineReq \leftarrow$ false
2. ***for*** each node $u \in newLeaves$
3. ***if*** depth(u) $<$ $minDepth$ or SplitRequired($u, isoValue$)
4. ***for*** each node $w \in \mathcal{N}(u)$
5. ***if*** (val(u) $-$ $isovalue$) $*$ (val(w) $-$ $isovalue$) < 0 ***then*** {*contour is between the centers of* u *and* w}
6. ***if*** $w \notin markedNodes$ ***then*** add w to $markedNodes$
7. ***if*** $u \notin markedNodes$ ***then*** add u to $markedNodes$
8. $refineReq \leftarrow$ true
9. return $refineReq$

Figure 20: Algorithm for identifying candidates for splitting

Split($u, d_{min}, newLeaves$) {*Splits the node* u}

Input: An octree node u and a positive number d_{min}. $newLeaves$ is a set where reference of the children are added. **Output:** This routine splits node u, by creating its children and distributing the atoms intersecting u to the children. And finally, it computes $\mathcal{F}(M, \text{center}(u))$ for each $v \in$ child(u).

1. ***if*** Length(u)/2 $< d_{min}$ ***then*** return
2. ***else***
3. create children nodes of u
4. ***for*** each $v \in$ child(u) ***do***
5. copy atoms from u which intersect box(v) to v
6. val(v) $\leftarrow \mathcal{F}(M, \text{center}(v))$
7. leaf(v) $\leftarrow$ true
8. depth(v) $\leftarrow$ depth(u) $+ 1$
9. add v to $newLeaves$
10. clear the list of atoms of u
11. leaf(u) $\leftarrow$ false

Figure 21: Algorithm for splitting an octree node u.

Algorithms for Updating Octree

5.5 Details for efficient cell classification using augmented DPG

- **Initialization:**

1. Consider a grid GP with uniform grid-

SplitRequired $(u, isoValue)$
{*Decides whether to split u*}

Input: An octree node u and the $isoValue$. **Output:** Based on val(u) and the values of its neighbors, decides whether the iso-contour potentially intersects u, in which case u should be split and SplitRequired returns true.

1. ***for*** each node $w \in \mathcal{N}(u)$
2. ***if*** $(\text{val}(u) - isovalue) * (\text{val}(w) - isovalue) < 0$ ***then***
3. return true
4. return false

Figure 22: Algorithm for deciding whether a specific node u should be split

MergeRequired $(u, isoValue)$
{*Decides whether to merge u*}

Input: An octree node u and the $isoValue$. **Output:** Returns true if u can be coarsened, i.e. u does not contain the isocontour.

1. $merge \leftarrow$ true
2. ***for*** each node $v \in$ child(u)
3. ***if*** child $(v) \neq$ nil ***then***
4. $merge \leftarrow merge$&MergeRequired$(v, isoValue)$
5. ***else***
6. $merge \leftarrow merge$&!SplitRequired$(v, isoValue)$
7. return $merge$

Figure 23: Algorithm for deciding whether children of a specific node u should be merged

Merge(u) {*Merges the subtree under u*}

Input: An octree node u and a positive number d_{min}. $newLeaves$ is a set where reference of the children are added. **Output:** This routine merges the subtree under u. A node is merged by removing its children and copying the atoms in the children back to the parent. Finally, u is marked as a leaf.

1. ***if*** child(u) = nil ***then*** return {*It is already a leaf*}
2. ***else***
3. ***for*** each $v \in$ child(u) ***do***
4. ***if*** child(u) $\neq$ nil
5. Merge(v)
6. copy atoms from v to u
7. delete v
8. leaf(u) $\leftarrow$ true

Figure 24: Algorithm for merging the children of an octree node u.

size of $r_p/2$

2. Mark all cells of GP as C_{OUT}. Each cell contains a list of pointers to the mesh inside the cell. Initially the list is empty.
3. Mark all grid-points of GP as V_{OUT}
4. Insert all grid-points into a DPG. Let's call it DPG_{GP}
5. Create an empty DPG, named DPG_{atom} for storing all the atoms
6. Create an empty DPG, named $DPG_{expatom}$ for storing all the exposed atoms
7. Create an empty DPG, named $DPG_{sasgrid}$ for storing all grid points marked V_{SAS}

UpdateOctree$(a, a', isoValue)$
{*Updates the octree when atom a moves to a new position a'*}

Input: The previous and new positions of an atom, and the $isoValue$. **Output:** This routine updates the octree by modifying the function values of affcted cells. The change in function values might produce a shift in the iso-contour. So, the octree is refined/coarsened locally to reflect the change.

1. $affectedNodes \leftarrow \{u | (\text{box}(u) \bigcap (a \cup a')) \neq \varnothing\}$ {*identified by traversing the octree*}
2. intialize set $markedForMerge \leftarrow \phi$
3. intialize set $markedForSplit \leftarrow \phi$
4. ***while*** $affectedNodes \neq \phi$ ***do***
5. ***for*** each $u \in affectedNodes$ ***do***
6. ***if*** MergeRequired(u, isoValue) ***then***
7. add u to $markedForMerge$
8. mark u
9. remove u from $markedForMerge$
10. ***for*** each $u \in affectedNodes$ ***do***
11. remove u from $affectedNodes$
12. ***for*** each node $w \in \mathcal{N}(u)$
13. ***if*** $(\text{val}(u) - isovalue) * (\text{val}(w) - isovalue) < 0$ ***then***
14. ***if*** w is not marked ***then***
15. add w to $markedForSplit$
16. mark w
17. ***if*** u is not marked ***then***
18. add u to $markedForSplit$
19. mark u
20. ***for*** each node $v \in markedForSplit$
21. Split$(v, d_{min}, affectedNodes)$
22. unmark v
23. remove v from $markedForSplit$
24. ***for*** each $u \in markedForMerge$ ***do***
25. Merge(u)
26. unmark u

Figure 25: Algorithm for updating the octree due to atom a moving to a'

8. Define empty lists C_{rem}, C_{add} and C_{mod}.

- **Add**(a_i, c_i, r_i)**:**
  Given an atom a_i with center c_i and radius r_i, (i) updates classifications of atoms, gridpoints and gridcells, (ii) updates the DPGs DPG_{atom}, $DPG_{expatom}$ and $DPG_{sasgrid}$, (iii) updates the lists C_{rem}, C_{add} and C_{mod}, and finally (iv) updates the surface mesh.

  1. Reset C_{rem}, C_{add} and C_{mod}
  2. Let, $G1_i$ = all gridpoints inside $S_{SAS}(a_i)$.
  3. For each grid-point $g \in G1_i$
    (a) If g is marked V_{OUT}
      i. Mark g as V_{SAS}
      ii. Insert g into $DPG_{sasgrid}$
      iii. Mark a_i as exposed.
      iv. Insert a_i into $DPG_{expatom}$
      v. UpdateCells(g)
  4. Let, $G2_i = DPG_{GP} -> Range(c_i, r_i)$
  5. For each grid-point $g \in G2_i$
    (a) If g is marked as V_{SAS}
      i. Mark g as V_{VDW}
      ii. Remove g from $DPG_{sasgrid}$
      iii. UpdateCells(g)
  6. If a_i is exposed
    (a) Let $A_i = DPG_{expatom} -> Range(a_i, r_i + r_{max})$
    (b) For each $a_j \in A_i$
      i. If $DPG_{sasgrid} -> Range(a_j, r_j + r_p)$ is empty then remove a_j from $DPG_{expatom}$
      ii. Mark a_j as buried.
  7. Insert a_i into DPG_{atom}
  8. UpdateContour()

- **Remove**(a_i, c_i, r_i)**:**
  Given an atom a_i with center c_i and radius r_i, (i) updates classifications of atoms, gridpoints and gridcells, (ii) updates the DPGs DPG_{atom}, $DPG_{expatom}$ and $DPG_{sasgrid}$, (iii) updates the lists C_{rem}, C_{add} and C_{mod}, and finally (iv) updates the surface mesh.

  1. Reset C_{rem}, C_{add} and C_{mod}
  2. Remove a_i from DPG_{atom}
  3. Let, $G1_i = DPG_{GP} -> Range(C_i, r_i + r_p)$
  4. For each grid-point $g \in G1_i$
    (a) If g is marked as V_{SAS}
      i. Remove a_i from $DPG_{expatom}$
      ii. If $DPG_{expatom} -> Range(g, r_max + r_p)$ is an empty set
      A. Mark g as V_{OUT}
      B. Remove g from $DPG_{sasgrid}$
      C. UpdateCells(g)
    (b) Else if g is marked as V_{VDW}
      i. If $A1_g = DPG_{atom} -> Range(g, r_max)$ is an empty set and $A2_g = DPG_{atom} -> Range(g, r_max + r_p)$ is not empty
      A. Mark g as V_{SAS}
      B. Mark each $a_j \in A2_g$ as exposed and insert it into $DPG_{expatom}$
      C. UpdateCells(g)
  5. UpdateContour()

- **UpdateCells**(g)**:**

  1. For each gridcell C which is a neighbor of g
    (a) Update the classification of C based on the definition of the classes.
    (b) If applicable, add C to the appropriate list among C_{rem}, C_{add} and C_{mod}.

- **UpdateContour():**

  1. For each gridcell $C \in C_{rem} \cup C_{mod}$
    (a) Remove the mesh inside C.
  2. For each gridcell $C \in C_{add} \cup C_{mod}$
    (a) Compute a function F inside C: either a sum of Gaussian of atoms in the neighborhood of C, or a sign distance function for C
    (b) Contour a level set F
    (c) Add the mesh to C

- **Move**$(a_i, c_{i1}, c_{i2}, r_i)$**:**

  1. Remove (a_i, C_{i1}, r_i)
  2. Add (a_i, C_{i2}, r_i)

- **getExposedAtoms():** Return all atoms in $DPG_{expatom}$