A Dynamic Data Structure for Flexible Molecular Maintenance and Informatics *

Chandrajit Bajaj
Institute for Computational Engineering and Science
University of Texas
Austin, TX 78712
bajaj@cs.utexas.edu

Rezaul Alam Chowdhury
Institute for Computational Engineering and Science
University of Texas
Austin, TX 78712
shaikat@cs.utexas.edu

Muhibur Rasheed
Institute for Computational Engineering and Science
University of Texas
Austin, TX 78712
muhibur@cs.utexas.edu

ABSTRACT

We present the "Dynamic Packing Grid" (DPG) data structure along with details of our implementation and performance results, for maintaining and manipulating flexible molecular models and assemblies. DPG can efficiently maintain the molecular surface (e.g., van der Waals surface and the solvent contact surface) under insertion/deletion/ movement (i.e., updates) of atoms or groups of atoms. DPG also permits the fast estimation of important molecular properties (e.g., surface area, volume, polarization energy, etc.) that are needed for computing binding affinities in drug design or in molecular dynamics calculations. DPG can additionally be utilized in efficiently maintaining multiple "rigid" domains of dynamic flexible molecules. In DPG, each update takes only $\mathcal{O}(\log w)$ time w.h.p. on a RAM with w-bit words i.e., $\mathcal{O}(1)$ time in practice, and hence is extremely fast. DPG's queries include the reporting of all atoms within $\mathcal{O}(r_{max})$ distance from any given atom center or point in 3-space in $\mathcal{O}(\log\log w)$ $(= \mathcal{O}(1))$ time w.h.p., where r_{max} is the radius of the largest atom in the molecule. It can also answer whether a given atom is exposed or buried under the surface within the same time bound, and can return the entire molecular surface in $\mathcal{O}(m)$ worst-case time, where m is the number of atoms on the surface. The data structure uses space linear in the number of atoms in the molecule.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*boundary representations; curve, surface, solid, and object representations; geometric algorithms, languages, and systems; physically based modeling*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures; geometrical problems and computations*; J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

General Terms

Algorithms, Design, Performance

Keywords

shape modeling, *de novo* drug design, computer aided design, interactive software, protein folding, molecular docking

1. INTRODUCTION

Many human functional processes are mediated through the interactions amongst proteins, a major molecular constituent of our anatomical makeup. A computational understanding of these interactions provides important clues for developing therapeutic interventions related to diseases such as cancer and metabolic disorders. Computational methods such as automated docking through shape and energetic complementarity scoring, aim to gain insight and predict such molecular interactions.

The most common model for proteins is a collection of atoms represented by spherical balls, with radii equal to their van der Waals radii [35, 16]. The surface of the union of these spheres is known as the van der Waals surface. Lee and Richards introduced the concept of accessibility to the solvent [31]. Proteins are not isolated, but commonly present in solutions, especially water. Also, the van der Waals surface contains too many internal atoms and patches which are not accessible by the solvent or any other protein that may bind to it. Hence, Lee and Richards gave a new definition for the protein surface or protein-solvent interface as the surface accessible to the watery solvent. They modeled water molecules as spheres with radius 1.4Å, and considered the locus of the center of one such 'probe', as it rolled along the protein surface as the Solvent Accessible Surface (SAS). Richards then gave a more commonly used definition for molecular surface as a set of contact and reentrant patches [42]. Though Connolly considered this an alternative definition of the SAS surface in [13], now it is commonly known as the Solvent Contact Surface (SCS), or Solvent Excluded Surface (SES) or simply the molecular surface/interface of the protein.

Protein interactions or protein-protein docking involves induced complementary fit between flexible protein interfaces and additionally the interface conformational changes are often critical during the lock and key matching [43].

*This research was supported in part by NSF grant CNS-0540033 and NIH contracts R01-EB00487, R01-GM074258, R01-GM07308.

SIAM/ACM Joint Conference on Geometric and Physical Modeling 2009 San Francisco, California USA

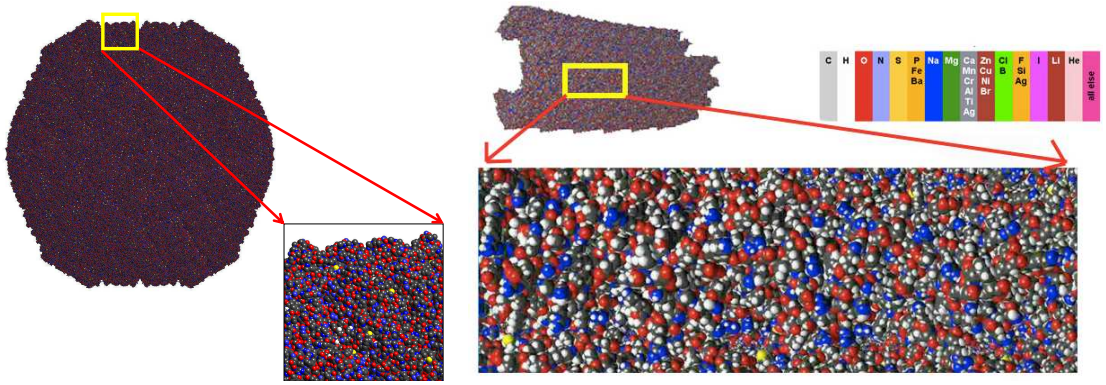


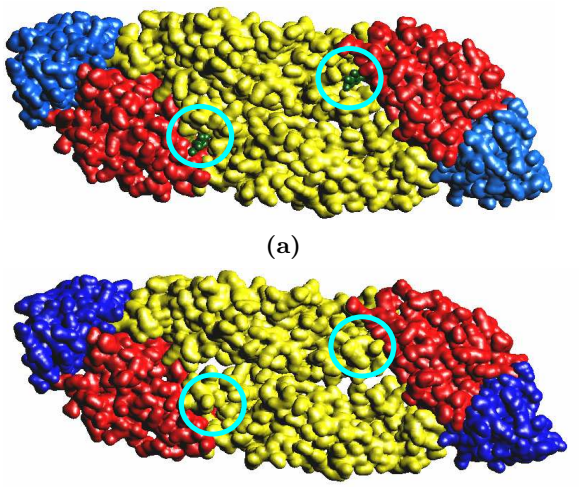
Figure 1: Rice Dwarf Virus (RDV) contains 3.5 million atoms (left) while Microtubule contains 1.2 million (right). In this figure, atoms are color-coded using the standard Corey, Pauling, Koltun (CPK) color scheme.

The flexible docking solution space consisting of all relative positions, orientations and conformations of the proteins, is searched, and the putative dockings are evaluated using combinations of interface complementarity scoring, and atomic pair-wise charged Coulombic interactions [27]. Since proteins function in their predominantly watery (solvent) environment, the computation of protein solvation energy (or known as protein - solvent interaction energy) also plays an important role in determining inter-molecular binding affinities "in-vivo" for drug screening, as well as in molecular dynamics simulations [52], and in the study of hydrophobicity and protein folding. When computing the solvation energy for molecules, it is crucial to correctly model and sample the protein - solvent interface.

Since Richards introduced the SES definition, a number of techniques have been devised for static construction of the molecular surface (e.g., [12, 13, 53, 17, 50, 3, 45, 44, 55, 23, 7, 6]). However, not much work has been done on dynamic maintenance of molecular surfaces. In [8] Bajaj et al. considered limited dynamic maintenance of molecular surfaces based on Non Uniform Rational BSplines (NURBS) descriptions for the patches. Eyal and Halperin [19, 20] presented an algorithm based on dynamic graph connectivity that updates the molecular surface after a conformational change in $\mathcal{O}\left(\log^2 n\right)$ amortized time per affected (by this change) atom.

In this paper we present the *Dynamic Packing Grid* (DPG) – a space and time efficient data structure that maintains a collection of balls (atoms) in 3-space allowing a range of spherical range queries and updates for rapid scoring of flexible protein-protein interactions. The efficiency of the data structure results from the assumption that the centers of two different balls in the collection cannot come arbitrarily close to each other, which is a natural property of molecules. A consequence of this assumption is that any ball in the collection can intersect at most a constant number of other balls. On a RAM with w-bit words, the data structure can report all balls intersecting a given ball or within $\mathcal{O}\left(r_{max}\right)$ distance from a given point in $\mathcal{O}\left(\log\log w\right)$ time w.h.p., where r_{max} is the radius of the largest ball in the collection. It can also answer whether a given ball is exposed (i.e., lies on the union boundary) or buried within the same time bound. At any time the entire union boundary can be extracted from the data structure in $\mathcal{O}\left(m\right)$ time in the worst-case, where m is the number of atoms on the boundary. Updates (i.e., insertion/deletion/movement of a ball) are supported in $\mathcal{O}\left(\log w\right)$ time (w.h.p.). The data structure uses linear space. A packing grid can maintain both the van der Waals surface and the solvent contact surface (SCS) of a molecule within the performance bounds mentioned above. Packing grids can be used to maintain the surface of a flexible molecule decomposed into rigid domains so that applying a bending/shearing/twisting motion between two domains takes $\mathcal{O}\left(1+\overline{m}\log w\right)$ time (w.h.p.), where $\overline{m}$ is the number of atoms in the connectors between the two domains. We also describe a *Hierarchical Packing Grid* (HPG) data structure that maintains a molecule at multiple resolutions (atomic and coarser) under updates, and can compute any mixed resolution surface efficiently. Packing grids can also aid in fast energetics calculation by rapidly locating the atoms close to each sampled quadrature point on the SCS.

DPG has potential applications in interactive software tools developed for *de novo* drug design (e.g., [30, 46, 18, 29]), protein folding (e.g., [28, 14]) and molecular docking (e.g., [33, 2]) that use human intuition and biological knowledge in order to steer the prediction process. These applications often need to handle extremely large molecules and macromolecules (e.g., as shown in Figure 1 Rice Dwarf Virus with 3.5 million atoms, and Microtubule has 1.2 million), and need to perform a sequence of dynamic updates on them in real time. The *Molecule Evaluator* [30, 18] is a *de novo* molecular design software based on adaptive interactive evolution. In a series of interactive steps it applies a set of problem-specific mutation (e.g., add/remove atom, add/remove group) and recombination operators on a set of evolving molecules, and keeps track of several chemical and biological properties of each molecule (e.g., molecular mass, hydrophobicity, etc.). The *ProteinShop* software [28, 14] allows the interactive creation of protein structures (e.g., through shape manipulation) given an amino acid sequence and a sequence of predicted secondary structure types for each amino acid. *DockingShop* [33] is a successor of ProteinShop, which provides an interactive docking environment with flexibility of side chains and backbone movement. Users can adjust the receptor protein structure by rotating the backbone dihedral angles, changing the dihedral angles of selected residues, substituting the side chain of selected residues using a rotamer library, or changing a residue for another while keeping the backbone fixed. Figure 2 shows an example where the flexible movement/rearrangement of the "kl" β-hairpin on the envelope (E) Glycoprotein of dengue virus opens up a hydrophobic pocket for ligand binding, and



(a)

(b)

Figure 2: Figures (a) and (b) show the structure of a soluble fragment of the envelope (E) Glycoprotein from DV (dengue virus) type 2. Figure (a) shows the crystals grown in the presence (pre-fusion) of the detergent n-octyl-β-D-glucoside (β-OG, colored in green), and Figure (b) shows the same in its absence (post-fusion). The key difference between these two structures is a local rearrangement of the "kl" β-hairpin (residues 268-280) and the concomitant opening up of a hydrophobic pocket for ligand binding. In Figure (a) this pocket is occupied by a molecule of β-OG [36].

the inhibitor n-octyl-β-D-glucoside docks into that pocket. *VRDD* [2] supports molecular visualization and interactive docking in a VR environment, and allows side-chain flexibility.

The molecular dynamic simulation tool *IMD* [49] allows interactive manipulation of bio-molecular systems. It combines interactive molecular visualization (using VMD [26]) with molecular dynamic simulation (using NAMD [38, 41]) in the background that supports manipulation of molecules by applying force to single atoms. Traditional all-atom molecular dynamics (MD) simulation reveals in detail the protein folding process, but it is restricted to small time scales on the order of nanosecond [47] and small length range on the order of nanometer [32, 34]. To fully investigate the folding process of a protein into its functional structure, a larger timescale from micro- to millisecond and larger length scale of micrometer are needed [4]. Protein coarse grained (CG) models which represent clusters of atoms with similar physical properties by CG beads and simplify the interactions significantly reduce the size of the system and therefore become a promising approach to reproduce large-scale protein motions.

The DPG data structure also has potential applications in tracking the dynamic structure of a particle system as particles move, appear and disappear [5, 22, 25]. Particle systems are used for modeling a number of physical world scenarios ranging from cosmological systems and plasma physics to molecular systems, where particles are defined as smooth functions with compact support. The applications are wide and varied and include chemistry, material science, and bioengineering. The dynamic re-meshing problem for time dependent particle systems arise in gas hydrodynamics simulations essential in the computational investigation of the formation of large scale structures, such as galaxies and galaxy clusters, in the universe [25]. For the meshing of particle systems, it suffices to consider particles as idealized balls, or radially symmetric domains of support of their kernels.

The rest of the paper is organized as follows. We describe and analyze the packing grid data structure in Section 2. We give some preliminaries in Section 2.1, describe the layout of the data structure in Section 2.2, and describe and analyze the supported queries and updates in Section 2.3. In Section 3 we describe how to use packing grids for maintaining the surface of a molecule decomposed into rigid domains, and in Section 4 we describe hierarchical packing grids for maintaining mixed resolution surfaces. In Section 5 we describe some applications of packing grids. Our experimental results are included in Section 6.

2. THE DYNAMIC PACKING GRID DATA STRUCTURE

We describe the *packing grid data structure* for maintaining a set M of balls in 3-space efficiently under the following set of queries and updates. By $B = (c, r)$ we denote a ball with center c and radius r.

Queries.

1. INTERSECT(c, r): Return all balls in M that intersect the given ball $B = (c, r)$. The given ball may or may not belong to the set M.
2. RANGE(p, δ): Return all balls in M with centers within distance δ of point p. We assume that δ is at most a constant multiple of the radius of the largest ball in M.
3. EXPOSED(c, r): Returns *true* if the ball $B = (c, r)$ contributes to the outer boundary of the union of the balls in M. The given ball must belong to M.
4. SURFACE( ): Returns the outer boundary of the union of the balls in M. If there are multiple disjoint outer boundary surfaces defined by M, the routine returns any one of them.

Updates.

1. ADD(c, r): Add a new ball $B = (c, r)$ to the set M.
2. REMOVE(c, r): Remove the ball $B = (c, r)$ from M.
3. MOVE(c_1, c_2, r): Move the ball with center c_1 and radius r to a new center c_2.

We assume that at all times during the lifetime of the data structure the following holds.

ASSUMPTION 2.1. *If r_{max} is the radius of the largest ball in M, and d_{min} is the minimum Euclidean distance between the centers of any two balls in M, then $r_{max} = \mathcal{O}(d_{min})$.*

In general, a ball in a collection of n balls in 3-space can intersect $\Theta(n)$ other balls in the worst case, and it has been shown in [11] that the boundary defined by the union of these balls has a worst-case combinatorial complexity of $\Theta(n^2)$. However, if M is a "union of balls" representation of the atoms in a molecule, then assumption 2.1 holds naturally [24, 51], and as proved in [24], in that case, both complexities improve by a factor of n. The following theorem states the consequences of the assumption.

Operations	Time Complexity	
	Assuming $t_q = \mathcal{O}(\log\log w)$, $t_u = \mathcal{O}(\log w)$	Assuming $t_q = \mathcal{O}(\log\log n)$, $t_u = \mathcal{O}\left(\frac{\log n}{\log\log n}\right)$
Range(p, δ) | Intersect(c, r) | Exposed(c, r) ($\delta = \mathcal{O}(r_{max})$)	$\mathcal{O}(\log\log w)$ (w.h.p.)	$\mathcal{O}(\log\log n)$ (w.h.p.)
Surface( )	$\mathcal{O}$(#balls on surface) (worst-case)	
Add(c, r) | Remove(c, r) | Move(c_1, c_2, r)	$\mathcal{O}(\log w)$ (w.h.p.)	$\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ (w.h.p.)
Assumptions: (*i*) RAM with w-bit Words, (*ii*) Collection of n Balls, and (*iii*) $r_{max} = \mathcal{O}$(minimum distance between two balls)		

Table 1: Time complexities of the operations supported by the packing grid data structure.

Theorem 2.1. *(Theorem 2.1 in [24], slightly modified) Let $M = \{B_1, \ldots, B_n\}$ be a collection of n balls in 3-space with radii $r_1, \ldots, r_n$ and centers at $c_1, \ldots, c_n$. Let $r_{max} = \max_i \{r_i\}$ and let $d_{min} = \min_{i,j} \{d(c_i, c_j)\}$, where $d(c_i, c_j)$ is the Euclidean distance between c_i and c_j. Also let $\delta M = \{\delta B_1, \ldots, \delta B_n\}$ be the collection of spheres such that δB_i is the boundary surface of B_i. If $r_{max} = \mathcal{O}(d_{min})$ (i.e., Assumption 2.1 holds), then:*

(*i*) *Each $B_i \in M$ intersects at most $216 \cdot (r_{max}/d_{min})^3 = \mathcal{O}(1)$ other balls in M.*
(*ii*) *The maximum combinatorial complexity of the boundary of the union of the balls in M is $\mathcal{O}\left((r_{max}/d_{min})^3 \cdot n\right) = \mathcal{O}(n)$.*

Proof. Similar to the proof of Theorem 2.1 in [24]. ■

Therefore, as Theorem 2.1 suggests, for intersection queries and boundary construction, one should be able to handle M more efficiently if assumption 2.1 holds. The efficiency of our data structure, too, partly depends on this assumption.

2.1 Preliminaries

Before we describe our data structure we present several definitions in order to simplify the exposition.

Definition 2.1 (r-grid and grid-cell). *An r-*grid *is an axis-parallel infinite grid structure in 3-space consisting of cells of size $r \times r \times r$ ($r \in \mathbb{R}$) with the* root *(i.e., the corner with the smallest x, y, z coordinates) of one of the cells coinciding with origin of the (Cartesian) coordinate axes. The grid cell that has its root at Cartesian coordinates (ar, br, cr) (where $a, b, c \in \mathbb{Z}$) is referred to as the (a, b, c, r)-cell or simply as the (a, b, c)-cell when r is clear from the context.*

Definition 2.2 (grid-line). *The (b, c, r)-*line *(where $b, c \in \mathbb{Z}$) in an r-grid consists of all (x, y, z, r)-cells with y and z fixed to b and c, respectively. When r is clear from the context the (b, c, r)-line will simply be called the (b, c)-line.*

Observe that each cell on the (b, c, r)-line can be identified with a unique integer, e.g., the cell at index $a \in \mathbb{Z}$ on the given line corresponds to the (a, b, c, r)-cell in the r-grid.

Definition 2.3 (grid-plane). *The (c, r)-*plane *(where $c \in \mathbb{Z}$) in an r-grid consists of all (x, y, z, r)-cells with z fixed to c. The (c, r)-plane will be referred to as the c-plane when r is clear from the context.*

The (c, r)-*plane* can be decomposed into an infinite number of lines each identifiable with a unique integer. For example, index $b \in \mathbb{Z}$ uniquely identifies the (b, c, r)-line on the given plane. Also each grid-plane in the r-grid can be identified with a unique integer, e.g., the (c, r)-plane is identified by c.

The proof of the following lemma is straight-forward.

Lemma 2.1. *Let $M = \{B_1, \ldots, B_n\}$ be a collection of n balls in 3-space with radii $r_1, \ldots, r_n$ and centers at $c_1, \ldots, c_n$. Let $r_{max} = \max_i \{r_i\}$ and let $d_{min} = \min_{i,j} \{d(c_i, c_j)\}$, where $d(c_i, c_j)$ is the Euclidean distance between c_i and c_j. Suppose M is stored in the $2r_{max}$-grid G. Then*

(*i*) *If $r_{max} = \mathcal{O}(d_{min})$ (i.e., Assumption 2.1 holds) then each grid-cell in G contains the centers of at most $64 \cdot (r_{max}/d_{min})^3 = \mathcal{O}(1)$ balls in M.*
(*ii*) *Each ball in M intersects at most 8 grid-cells in G.*
(*iii*) *For a given ball $B \in M$ with center in grid-cell C, the center of each ball intersecting B lies either in C or in one of the 26 grid-cells adjacent to C.*
(*iv*) *The number of* non-empty *(i.e., containing the center of at least one ball in M) grid-cells in G is at most n, and the same bound holds for grid-lines and grid-planes.*

At the heart of our data structure is a fully dynamic one dimensional integer range reporting data structure for word RAM described in [37]. The data structure in [37] maintains a set S of integers under updates (i.e., insertions and deletions), and answers queries of the form: report any or all points in S in a given interval. The following theorem summarizes the performance bounds of the data structure which are of interest to us.

Theorem 2.2. *(proved in [37]) On a RAM with w-bit words the fully dynamic one dimensional integer range reporting problem can be solved in linear space, and with high probability bounds of $\mathcal{O}(t_u)$ and $\mathcal{O}(t_q + k)$ on update time and query time, respectively, where k is the number of items reported, and*

(*i*) *$t_u = \mathcal{O}(\log w)$ and $t_q = \mathcal{O}(\log\log w)$ using the data structure in [37]; and*
(*ii*) *$t_u = \mathcal{O}(\log n/\log\log n)$ and $t_q = \mathcal{O}(\log\log n)$ using the data structure in [37] for small w and a fusion tree [21] for large w.*

The data structure can be augmented to store satellite information of size $\mathcal{O}(1)$ with each integer without degrading its asymptotic performance bounds. Therefore, it supports the following three functions:

1. INSERT(i, s): Insert an integer i with satellite information s.
2. DELETE(i): Delete integer i from the data structure.
3. QUERY(l, h): Return the set of all $\langle\ i,\ s\ \rangle$ tuples with $i \in [l, h]$ stored in the data structure.

2.2 Description (Layout) of the Packing Grid Data Structure

We are now in a position to present our data structure. Let DPG be the data structure. We represent the entire 3-space as a $2r_{max}$-grid (see Definition 2.1), and maintain the non-empty grid-planes (see Definition 2.3), grid-lines (see Definition 2.2) and grid-cells (see Definition 2.1) in DPG. A grid component (i.e., cell, line or plane) is non-empty if it contains the center of at least one ball in M. The data structure can be described hierarchically. It has a tree structure with 5 levels: 4 internal levels (levels 3, 2, 1 and 0) and an external level of leaves (see Figure 3). The description of each level follows.

The Leaf Level "Ball" Data Structure (DPG_{-1}). The data structure stores the center $c = (c_x, c_y, c_z)$ and the radius r of the given ball B. It also includes a Boolean flag *exposed* which is set to *true* if B contributes to the outer boundary of the union of the balls in M, and *false* otherwise. If another ball B' intersects B, it does so on a circle which divides the boundary δB of B into two parts: one part is buried inside B' and hence cannot contribute to the union boundary, and the other part is exposed w.r.t. B' and hence might appear on the union boundary. The circular intersections of all balls intersecting B define a 2D arrangement A on δB which according to Theorem 2.1 has $\mathcal{O}(1)$ combinatorial complexity. A face of A is exposed, i.e., contributes to the union boundary, provided it is not buried inside any other ball. Observe that if at least one other ball intersects B, and A has an exposed face f, then each edge of f separates f from another exposed face f' which belongs to the arrangement A' of a ball intersecting B. We store all exposed faces (if any) of A in a set F of size $\mathcal{O}(1)$, and with each face f we store pointers to the data structures of $\mathcal{O}(1)$ other balls that share edges with f and also the identifier of the corresponding face on each ball. Observe that if B does not intersect any other balls then F will contain only a single face and no pointers to any other balls.

The Level 0 "Grid-Cell" Data Structure (DPG_0). The "grid-cell" data structure stores the root (see Definition 2.1) (a, b, c) of the grid-cell it corresponds to. A grid-cell can contain the centers of at most $\mathcal{O}(1)$ balls in M (see Lemma 2.1). Pointers to data structures of all such balls are stored in a set S of size $\mathcal{O}(1)$. Since we create "grid-cell" data structures only for non-empty grid-cells, there will be at most n (and possibly $\ll n$) such data structures, where n is the current number of balls in M.

The Level 1 "Grid-Line" Data Structure (DPG_1). We create a "grid-line" data structure for a (b, c)-line provided it contains at least one non-empty grid-cell. The data structure stores the values of b and c. Each (a, b, c)-cell lying on this line is identified with the unique integer a, and the identifier of each such non-empty grid-cell is stored in an integer range search data structure RR as described in Section 2.1 (see Theorem 2.2). We augment RR to store the pointer to the corresponding "grid-cell" data structure with each identifier it stores. The total number of "grid-line" data structure created is upper bounded by n and possibly much less than n.

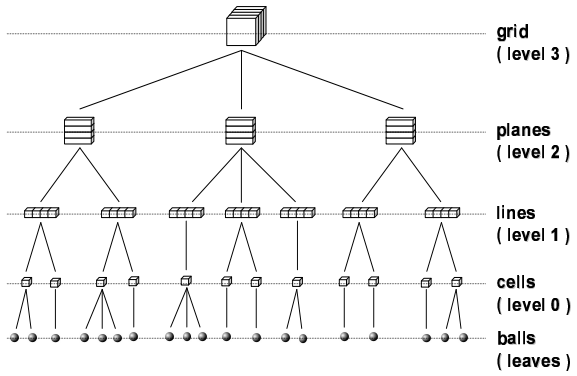


Figure 3: Hierarchical structure of DPG.

The Level 2 "Grid-Plane" Data Structure (DPG_2). A "grid-plane" data structure is created for a c-plane provided it contains at least one non-empty grid-line. Similar to the "grid-line" data structure it identifies each non-empty (b, c)-line lying on the c-plane with the unique integer b, and stores the identifiers in a range reporting data structure RR described in Section 2.1. A pointer to the corresponding "grid-line" data structure is also stored with each identifier. The data structure also stores c. The total number of "grid-plane" data structures created cannot exceed n, and will possibly be much less than n.

The Level 3 "Grid" Data Structure (DPG_3). This data structure maintains the non-empty grid-planes of the $2r_{max}$-grid in an integer range reporting data structure RR (see Section 2.1). Each c-plane is identified by the unique integer c, and each such integer stored in RR is also accompanied by a pointer to the corresponding "grid-plane" data structure. The "grid" data structure also stores a *surface-root* pointer which points to the "Ball" data structure of an arbitrary exposed ball in M.

We have the following lemma on the space usage of the data structure.

LEMMA 2.2. *Let M be a collection of n balls as defined in Theorem 2.1, and let Assumption 2.1 holds. Then the packing grid data structure storing M uses $\mathcal{O}(n)$ space.*

PROOF. The space usage of the data structure is dominated by the space used by the range reporting data structures, the grid-cells and the "ball" data structures. Since the range reporting data structures use linear space (see Theorem 2.2) and total number of non-empty grid components (i.e., planes, lines and cells) is $\mathcal{O}(n)$ (see Lemma 2.1), total space used by all such data structures is $\mathcal{O}(n)$. The grid cells contain pointers to "ball" data structures, and since no two grid-cells point to the same "ball" data structure, total space used by all grid-cells is also $\mathcal{O}(n)$. Each "ball" data structure contains the arrangement A and the face decomposition

F of the exposed (if any) faces of the ball. The total space needed to store all such arrangements and decompositions is $\mathcal{O}\left((r_{max}/d_{min})^3 \cdot n\right)$ (see Theorem 2.1) which reduces to $\mathcal{O}(n)$ under Assumption 2.1. Thus the total space used by the data structure is $\mathcal{O}(n)$. ■

2.3 Queries and Updates

The queries and updates supported by the data structure are implemented as follows.

2.3.1 Queries.

(1) Range(p, δ): Let $p = (p_x, p_y, p_z)$. We perform the following steps.

i. **Level 3 Range Query:** We invoke the function QUERY(l, h) of the range reporting data structure RR under DPG_3 (i.e., the level 3 "grid" data structure) with $l = \lfloor (p_z - \delta)/(2r_{max}) \rfloor$ and $h = \lfloor (p_z + \delta)/(2r_{max}) \rfloor$. This query returns a set S_2 of tuples, where each tuple $\langle\ c,\ P_c\ \rangle \in S_2$ refers to a non-empty c-plane with a pointer P_c to its level 2 "grid-plane" data structure.

ii. **Level 2 Range Query:** For each $\langle\ c,\ P_c\ \rangle \in S_2$, we call the range query function under the corresponding level 2 data structure with $l = \lfloor (p_y - \delta')/(2r_{max}) \rfloor$ and $h = \lfloor (p_y + \delta')/(2r_{max}) \rfloor$, where $(\delta')^2 = \delta^2 - (c - p_z)^2$ if $c - p_z < \delta$, and $\delta' = r_{max}$ otherwise. This query returns a set $S_{1,c}$ of triples, where each triple $\langle\ b,\ c,\ P_{b,c}\ \rangle \in S_{1,c}$ refers to a non-empty (b, c)-line with a pointer $P_{b,c}$ to its level 1 "grid-line" data structure. We obtain the set S_1 by merging all $S_{1,c}$ sets.

iii. **Level 1 Range Query:** For each $\langle\ b,\ c,\ P_{b,c}\ \rangle \in S_1$, we call the integer range query function under the corresponding level 1 "grid-line" data structure with $l = \lfloor (p_x - \delta'')/(2r_{max}) \rfloor$ and $h = \lfloor (p_x + \delta'')/(2r_{max}) \rfloor$, where $(\delta'')^2 = \delta^2 - (b - p_y)^2 - (c - p_z)^2$ if $\delta^2 > (b - p_y)^2 + (c - p_z)^2$, and $\delta'' = r_{max}$ otherwise. This query returns a set $S_{0,b,c}$ of quadruples, where each quadruples $\langle\ a,\ b,\ c,\ P_{a,b,c}\ \rangle \in S_{0,b,c}$ refers to a non-empty (a, b, c)-cell with a pointer $P_{a,b,c}$ to its level 0 "grid-cell" data structure. We obtain the set S_0 by merging all $S_{0,b,c}$ sets.

iv. **Ball Collection:** For each $\langle\ a,\ b,\ c,\ P_{a,b,c}\ \rangle \in S_0$, we collect from the level 0 data structure of the corresponding (a, b, c)-cell each ball whose center lies within distance δ from p. We collect the pointer to the leaf level "ball" data structure of each such ball in a set S, and return this set.

The correctness of the function follows trivially since it queries a region in 3-space which includes the region covered by a ball of radius δ centered at p. It is straight-forward to see that the function makes at most $\mathcal{O}\left(\pi \cdot (\lceil \delta/r_{max} \rceil + 1)^2\right)$ calls to a range reporting data structure, and collects balls from at most $\mathcal{O}\left(\frac{4}{3}\pi \cdot (\lceil \delta/r_{max} \rceil + 1)^3\right)$ grid-cells. Using Lemma 2.1 and Theorem 2.2, we conclude that w.h.p. the function terminates in $\mathcal{O}\left((\delta/r_{max})^2 \cdot t_q + ((\delta + r_{max})/d_{min})^3\right)$ time. Assuming $r_{max} = \mathcal{O}(d_{min})$ (i.e., Assumption 2.1) and $\delta = \mathcal{O}(r_{max})$, the complexity reduces to $\mathcal{O}(t_q)$ (w.h.p.).

(2) Intersect(c, r): Let $B = (c, r)$ be the given ball. We perform the following two steps.

i. **Ball Collection:** We call RANGE(c, $r + r_{max}$) and collect the output in set S which contains pointers to the data structure of each ball in M with its center within distance $r + r_{max}$ from c.

ii. **Identifying Intersecting Balls:** From S we remove the data structure of each ball that does not intersect B, and return the resulting (possibly reduced) set.

We know from elementary geometry that two balls of radii r_1 and r_2 cannot intersect unless their centers lie within distance $r_1 + r_2$ of each other. Therefore, step (i) correctly identifies all balls that can possibly intersect B, and step (ii) completes the identification. Step (i) takes $\mathcal{O}\left(t_q + (r_{max}/d_{min})^3\right)$ time w.h.p., and step (ii) terminates in $\mathcal{O}\left((r_{max}/d_{min})^3\right)$ time in the worst case. Therefore, under Assumption 2.1 w.h.p. this function runs in $\mathcal{O}(t_q)$ time.

(3) Exposed(c, r): Let $B = (c, r)$ be the given ball. We locate B's data structure by calling RANGE(c, 0), and return the value stored in its *exposed* field. Clearly, the function takes $\mathcal{O}\left(t_q + (r_{max}/d_{min})^3\right)$ time (w.h.p.) which reduces to $\mathcal{O}(t_q)$ (w.h.p.) under Assumption 2.1.

(4) Surface( ): The *surface-root* pointer under the level 3 "grid" data structure points to the "ball" data structure of a ball B on the union boundary of M. We scan the set F of exposed faces of B, and using the pointers to other exposed balls stored in F we perform a depth-first traversal of all exposed balls in M and return the exposed faces on each such ball. Let m be the number of balls contributing to the union boundary of M. Then according to Theorem 2.1 the depth-first search takes $\mathcal{O}\left((r_{max}/d_{min})^3 \cdot m\right)$ time in the worst case which reduces to $\mathcal{O}(m)$ under Assumption 2.1.

2.3.2 Updates.

(1) Add(c, r): Let $c = (c_x, c_y, c_z)$ and let $c'_u = \left\lfloor \frac{c_u}{2r_{max}} \right\rfloor$, where $u \in \{x, y, z\}$. We perform the following steps.

i. If $M \neq \emptyset$, let G be the grid data structure, otherwise create and initialize G. Add input ball to M.

ii. Query the range reporting data structure $G.RR$ to locate the data structure P for the c'_z-plane. If P does not exist create and initialize P, and insert c'_z along with a pointer to P into $G.RR$.

iii. Query $P.RR$ and locate the data structure L for the (c'_y, c'_z)-line. If L does not exist then create and initialize L, and insert c'_y along with a pointer to L into $P.RR$.

iv. Locate the data structure C for the (c'_x, c'_y, c'_z)-cell by querying $L.RR$. Create and initialize C if it does not already exist, and insert c'_x and a pointer to C into $L.RR$.

v. Create and initialize a data structure B for the input ball and add it to the set $C.S$.

vi. Call INTERSECT(c, r) and find the set I of the "ball" data structures of all balls that intersect the input ball. Create the arrangement $B.A$ using the balls in I. The new ball may partly or fully bury some of the balls it

intersects, and hence we need to update the arrangement $B'.A$, the set $B'.F$ and the flag $B'.exposed$ of each $B' \in I$. The set $B.F$ is created and $B.exposed$ is initialized using the information in the updated data structures in I. If the *surface-root* pointer was pointing to a ball in I that got completely buried by the new ball, we update it to point to B instead.

Observe that the introduction of a new ball may affect the surface exposure of only the balls it intersects (i.e., bury some/all of them partly or completely), and no other balls. Hence, the updates performed in step (vi) (in addition to those in earlier steps) are sufficient to maintain the correctness of the entire data structure.

Steps (i) and (v) take $\mathcal{O}(1)$ time in the worst case, and w.h.p. each of steps (ii), (iii) and (iv) takes $\mathcal{O}(t_q + t_u)$ time. Finding the intersecting balls in step (vi) takes $\mathcal{O}\left(t_q + (r_{max}/d_{min})^3\right)$ time w.h.p., according to Theorem 2.1 creating and updating the arrangements and faces will take $\mathcal{O}\left((r_{max}/d_{min})^3 \times (r_{max}/d_{min})^3\right) = \mathcal{O}\left((r_{max}/d_{min})^6\right)$ time (w.h.p.). Thus the ADD function terminates in $\mathcal{O}\left(t_q + t_u + (r_{max}/d_{min})^6\right)$ time w.h.p., which reduces to $\mathcal{O}(t_u)$ (w.h.p.) assuming $r_{max} = \mathcal{O}(d_{min})$ (i.e., Assumption 2.1).

(2) Remove(c, r)**:** This function is symmetric to the ADD function, and has exactly the same asymptotic time complexity. Hence, we do not describe it here.

(3) Move(c_1, c_2, r)**:** This function is implemented in the obvious way by calling **Remove**(c_1, r) followed by **Add**(c_2, r). It has the same asymptotic complexity as the two functions above.

Therefore, we have the following theorem.

THEOREM 2.3. *Let M be a collection of n balls in 3-space as defined in Theorem 2.1, and let Assumption 2.1 holds. Let t_q and t_u be as defined in Theorem 2.2. Then the packing grid data structure storing M on a word RAM:*

- (i) *uses $\mathcal{O}(n)$ space;*
- (ii) *supports updates (i.e., insertion/deletion/movement of a ball) in $\mathcal{O}(t_u)$ time w.h.p.;*
- (iii) *reports all balls intersecting a given ball or within $\mathcal{O}(r_{max})$ distance from a given point in $\mathcal{O}(t_q)$ time w.h.p., where r_{max} is the radius of the largest ball in M; and*
- (iv) *reports whether a given ball is exposed or buried in $\mathcal{O}(t_q)$ time w.h.p., and returns the entire outer union boundary of M in $\mathcal{O}(m)$ worst-case time, where m is the number of balls on the boundary.*

In Table 1 we list the time complexities of the operations supported by our data structure.

3. EFFICIENT MAINTENANCE OF FLEXIBLE MOLECULES UNDER DOMAIN MOTIONS

Suppose we are given a flexible molecule decomposed into several (mostly) rigid domains which interact either through connected chain segments or large interfaces. We refer to these chain segments and interfaces as connectors. Domains may move with respect to each other through motions applied to the connectors. Two domains connected by at least one connector may undergo bending motion applied to some hinge point around some hinge axis. If they are connected by only one connector, a twisting motion can also be applied to the connector by updating torsion angles along its backbone. If two domains share a large interface area they may undergo a shearing motion with respect to each other. However, though domains are mostly rigid they may have flexible loops and side-chains on their surfaces.

We maintain a separate packing grid data structure $\mathcal{P}_i$ for each domain $\mathcal{D}_i$. If two domains $\mathcal{D}_i$ and $\mathcal{D}_j$ are connected and $i < j$, the set S_{ij} of all connectors between these two domains are included in $\mathcal{P}_i$, and a transformation matrix M_{ij} is kept with $\mathcal{P}_i$ that describes the exact location and orientation of the grid structure of $\mathcal{P}_j$ with respect to that of $\mathcal{P}_i$. Whenever some motion is applied to the connectors in S_{ij}, we update $\mathcal{P}_i$ in order to reflect the changes in the locations of the atoms in these connectors, and also update M_{ij} in order to reflect the new relative position and orientation of $\mathcal{P}_j$ with respect to $\mathcal{P}_i$. Hence such an update requires $\mathcal{O}(1 + m_{ij} \log w)$ time (w.h.p.), where m_{ij} is the number of atoms in the connectors in S_{ij}. We defer the tests to check whether any two domains intersect due to these movements until we need to construct the surface of the entire molecule in response to a surface query. At that point we extract the surface atoms from each $\mathcal{P}_i$ and insert them into an initially empty packing grid data structure $\mathcal{P}$ after applying necessary transformations. Thus generating the surface of the entire molecule requires $\mathcal{O}(\widehat{m} \log w)$ time (w.h.p.), where $\widehat{m}$ is the sum of the number of atoms on the surface of each domain. If we need to update the conformation of a flexible loop or a side-chain on the surface of some domain $\mathcal{D}_i$, we directly update the locations of the atoms affected by this change in $\mathcal{P}_i$. Such an update requires $\mathcal{O}(\widetilde{m} \log w)$ time (w.h.p.), where $\widetilde{m}$ is the number of atoms affected. Therefore, we have the following lemma.

LEMMA 3.1. *The surface of a flexible molecule decomposed into (mostly) rigid domains can be maintained using packing grid data structures so that*

- (i) *updating for a bending/shearing/twisting motion applied between two domains takes $\mathcal{O}(1 + \overline{m} \log w)$ time (w.h.p.), where $\overline{m}$ is the number of atoms in the connectors between the two domains;*
- (ii) *updating the conformation of a flexible loop or a side-chain on the surface of a domain takes $\mathcal{O}(\widetilde{m} \log w)$ time (w.h.p.), where $\widetilde{m}$ is the number of atoms affected by this change; and*
- (iii) *generating the surface of the entire molecule requires $\mathcal{O}(\widehat{m} \log w)$ time (w.h.p.), where $\widehat{m}$ is the sum of the number of atoms on the surface of each domain.*

4. HIERARCHICAL PACKING GRIDS FOR MIXED RESOLUTION SURFACES

We construct a k-level hierarchical packing grid data structure $\mathsf{HPG}(k)$ as follows. For $i \in [0, k-1]$, level i contains a packing grid data structure $\mathsf{DPG}^{(i)}$ with parameters $\langle r_{max}^{(i)}, d_{min}^{(i)} \rangle$ for which Assumption 2.1 holds. We also assume that for $i \in [0, k-2]$, $r_{max}^{(i+1)} = \Theta\left(r_{max}^{(i)}\right)$ and $d_{min}^{(i+1)} = \Theta\left(d_{min}^{(i)}\right)$. The level 0 data structure $\mathsf{DPG}^{(0)}$ contains the

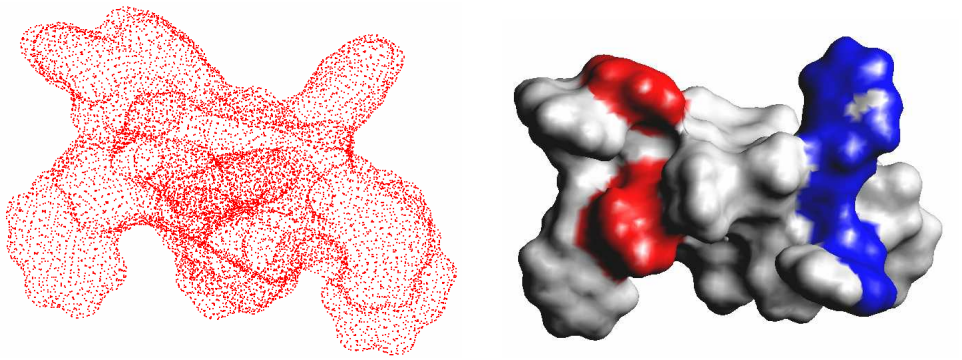


Figure 4: (LEFT) Gaussian integration points on the surface of peptide antibiotic Gramicidin A (1MAG). The surface is partitioned into 30,624 triangular patches, and there are three Gaussian quadrature nodes per triangle. The nodes are then mapped onto the ASMS to form the red point cloud. (RIGHT) Electrostatic solvation force computation for Gramicidin A. Atoms with the greatest electrostatic solvation force (top 5%) are colored in red; atoms having the weakest electrostatic solvation force (bottom 5%) are colored in blue.

atomic level union of balls representation of the given molecule M. For $i \in [1, k-1]$, $\mathsf{DPG}^{(i)}$ contains a coarser representation of the molecule represented in $\mathsf{DPG}^{(i-1)}$. Each ball in $\mathsf{DPG}^{(i)}$ represents a grouping several neighboring balls in $\mathsf{DPG}^{(i-1)}$. A single doubly linked list links the parent ball in $\mathsf{DPG}^{(i)}$ to all its child balls in $\mathsf{DPG}^{(i-1)}$. Additionally, each child ball maintains a direct pointer to its parent ball. Thus given the center of any ball in $\mathsf{DPG}^{(i)}$, the set of all its children in $\mathsf{DPG}^{(i-1)}$ can be found in $\mathcal{O}(t_q + l)$ time w.h.p., where l is the number of children of the given ball, and t_q is as defined in Theorem 2.2. We assume that each ball in $\mathsf{DPG}^{(i-1)}$ has at most one parent in $\mathsf{DPG}^{(i)}$, and thus the balls in $\mathsf{HPG}(k)$ form a forest. Now in order to create a mixed resolution surface of the given molecule M, we start at coarse resolution, say at some level $j > 0$, and copy $\mathsf{DPG}^{(i)}$ to an initially empty packing grid DPG with the same parameters. Now we selectively replace balls in DPG with finer resolution balls from the appropriate level in $\mathsf{HPG}(k)$, and we keep replacing until we get the required mixed resolution representation of M in DPG.

5. ADDITIONAL INFORMATICS

We briefly describe some applications of the packing grid data structure below.

Maintaining van der Waals Surface of Molecules. For dynamic maintenance of the van der Waals surface of a molecule we can use the packing grid data structure directly. Each atom is treated as a ball with a radius equal to the van der Waals radius of the atom (see [10] for a list of van der Waals radius of different atoms).

Maintaining Lee-Richards (SCS/SES) Surface. We can use the packing grid data structure for the efficient maintenance of the Lee-Richards surface of a molecule under insertion/deletion/movement of atoms. The performance bounds given in Table 1 remain unchanged. We maintain two packing grid data structures: DPG and DPG'. The DPG data structure keeps track of the patches on the Lee-Richards surface, and DPG' is used for detecting intersections among concave patches.

Before adding an atom to DPG, we increase its radius r_s, where r_s is the radius of the rolling solvent atom. The DPG data structure keeps track of all solvent exposed atoms, i.e., all atoms that contribute to the outer boundary of the union of these enlarged atoms. Theorem 2.1 implies that each atom in DPG contributes $\mathcal{O}(1)$ patches to the Lee-Richards surface, and the insertion/deletion/movement of an atom results in local changes of only $\mathcal{O}(1)$ patches. We can modify DPG to always keep track of where two or three of the solvent exposed atoms intersect, and once we know the atoms contributing to a patch we can easily compute the patch in $\mathcal{O}(1)$ time [6].

The Lee-Richards surface can self-intersect in two ways: (i) a toroidal patch can intersect itself, and (ii) two different concave patches may intersect [6]. The self-intersections of toroidal patches can be easily detected from DPG. In order to detect the intersections among concave patches, we maintain the centers of all current concave patches in DPG', and use the INTERSECT query to find the concave patch (if any) that intersects a given concave patch.

Energetics (Force) Computation. The solvation energy G_{sol} of a molecule consists of the energy to form cavity in the solvent (G_{cav}), the solute-solvent van der Waals interaction energy (G_{vdw}), and the electrostatic potential energy change due to the solvation (also known as the polarization energy, G_{pol}).

$$G_{\text{sol}} = G_{\text{cav}} + G_{\text{vdw}} + G_{\text{pol}} \tag{5.1}$$

The first two terms G_{cav} and G_{vdw} in the sum above are linearly related to the solvent accessible surface area Ω_{SAS} of the molecule.

$$G_{\text{cav}} + G_{\text{vdw}} = \gamma \cdot \Omega_{\text{SAS}} \tag{5.2}$$

The last term, G_{pol}, can be approximated using the *Generalized Born* (GB) theory as follows [48].

$$G_{pol} = -\frac{\tau}{2} \sum_{i,j} \frac{q_i q_j}{\sqrt{r_{ij}^2 + R_i R_j e^{-\frac{r_{ij}^2}{4R_i R_j}}}}, \tag{5.3}$$

where $\tau = 1 - \frac{1}{\epsilon}$, and R_i is the effective Born radius of atom i. The R_i's can be approximated as follows.

$$R_i^{-1} = \frac{1}{4\pi} \int_{\Gamma} \frac{(\mathbf{r} - \mathbf{x}_i) \cdot \mathbf{n}(\mathbf{r})}{|\mathbf{r} - \mathbf{x}_i|^4} dS, \tag{5.4}$$

where Γ is the boundary of the molecule, $\mathbf{n}(\mathbf{r})$ is the normal of the molecular surface at $\mathbf{r}$ pointing out of the molecule, and $\mathbf{x}_i$ is the center of atom i. A discrete approximation of R_i^{-1} based on equation 5.4 is as follows [9].

$$R_i^{-1} = \frac{1}{4\pi} \sum_{k=1}^{N} w_k \frac{(\mathbf{r}_k - \mathbf{x}_i) \cdot \mathbf{n}(\mathbf{r}_k)}{|\mathbf{r}_k - \mathbf{x}_i|^4}, \tag{5.5}$$

where the $\mathbf{r}_k$'s are N carefully chosen integration points on the boundary of the molecule, and w_k is a weight assigned to $\mathbf{r}_k$ in order to achieve higher order of accuracy for small N.

The non-polar terms G_{cav} and G_{vdw} can be computed directly from the solvent accessible surface (SAS) area Ω_{SAS} of the molecule (see equation 5.2). The SAS of the molecule can be extracted in $\mathcal{O}(\widetilde{m} \log w)$ (w.h.p.) time and $\mathcal{O}(\widetilde{m})$ space using a DPG data structure, where $\widetilde{m}$ is the number of atoms in the molecule. The DPG data structure outputs the SAS as a set of spherical (convex and concave) and toroidal patches, and we add up the area of each patch in order to calculate Ω_{SAS}.

In order to approximate the polar term G_{pol} first we need to approximate the Born radius R_i of each atom i. We use the discrete approximation equation 5.5 for computing R_i. Given the solvent excluded surface (SES) of the molecule, it has been shown in [9] how to choose N integration points $\mathbf{r}_k$ and weights w_k optimally in order to reduce the error in approximation. Figure 4 shows the distribution of integration points on the surface of 1MAG.PDB. We compute the SES of the molecule in $\mathcal{O}(\widetilde{m} \log w)$ time (w.h.p.) and $\mathcal{O}(\widetilde{m})$ space using a DPG data structure $\overline{\mathcal{D}}$, and then use the method in [9] in order to choose the integration points and weights in $\mathcal{O}(N)$ time. We use $\widetilde{m}$ initially empty buckets B_i ($i \in [1, \widetilde{m}]$) in order to collect in each B_i the integration points within distance $\widetilde{\delta}$ from atom center $\mathbf{x}_i$, where $\widetilde{\delta}$ is a user-defined distance threshold. Assuming that all atoms of the molecule have already been inserted into $\overline{\mathcal{D}}$, we perform a range query on $\overline{\mathcal{D}}$ for each integration point $\mathbf{r}_k$ in order to collect all atoms within distance $\widetilde{\delta}$ from that point and insert $\mathbf{r}_k$ into the bucket corresponding to each atom obtained from this query. Then for each atom i, we compute R_i using equation 5.5 using the integration points collected in bucket B_i. Assuming that $\widetilde{m}_{\widetilde{\delta}}$ is an upper bound on the number of atoms within distance $\widetilde{\delta}$ from any given point in space, the time spent for computing all R_i's is $\mathcal{O}\left(N \log\log w + N\widetilde{m}_{\widetilde{\delta}}\right)$ which reduces to $\mathcal{O}(N \log\log w)$ (w.h.p.) since $\widetilde{m}_{\widetilde{\delta}}$ is a constant (though could be quite large) for constant $\widetilde{\delta}$. Once all R_i's are computed G_{pol} can be computed using equation 5.3 in $\mathcal{O}\left(\widetilde{m}^2\right)$ time in the worst case. The space usage is $\mathcal{O}\left(\widetilde{m} + N\widetilde{m}_{\widetilde{\delta}}\right)$ which is $\mathcal{O}(\widetilde{m} + N)$ for constant $\widetilde{\delta}$.

Solvation force calculations require computations of similar integrals (as solvation energy) which in turn reduces to numerical summation of distances from quadrature points. Details are in [9].

6. IMPLEMENTATION AND EXPERIMENTAL RESULTS

In our current implementation, instead of the 1D integer range-reporting data structure presented in [37], we have implemented a much simpler data structure that supports both updates and distance queries in expected $\mathcal{O}(\log w)$ time and uses linear space [15]. Since w is usually not more than 64, for most practical purposes a $\mathcal{O}(\log w)$ query time should be almost as good as $\mathcal{O}(\log\log w)$ time. This data structure builds on binary search trees, dynamic perfect hashing, and y-fast trees [54]. However, instead of dynamic perfect hashing we used "cuckoo hashing" [40] since it is much simpler, and still supports lookups in $\mathcal{O}(1)$ worst-case time, and updates in expected $\mathcal{O}(1)$ time.

In subsequent discussion we report the results on the performance of our implementation of the packing grid data structure. All experiments are performed on a 3 GHz 2×dual-core (only one core was used) AMD Opteron 2222 processor with 4 GB RAM.

In our first experiment that measures the performance of the QUERY function of DPG, we use more than 180k quadrature points, generated for energetics computations by sampling uniformly at random on the surface of PSTI (a variant of human pancreatic trypsin inhibitor: 1HPT.pdb) after protonation using PDB2PQR [1]. These points were randomly partitioned into four equal groups. Group 1 was first inserted into DPG and range queries were performed from each atom center of the molecule to report all quadrature points lying within a given distance from the center. Average running time was measured after executing each query multiple times. The same experiment was carried out with query distances 2, 4, 8 and 16 . After running experiments with group 1, group 2 was also inserted into the data structure and the same set of experiments were performed again. In the same manner groups 3 and 4 were also added subsequently so that the results gives a clear measure of the scalability of the data structure. Table 2 shows the results of this experiment. The time required is $\mathcal{O}(\log w + K)$ where K is the size of the output or in this case, the number of points returned. The fifth column of the table shows that, as the point set becomes denser, the efficiency of the data structure remains almost the same.

Table 3 reports the performance of update functions of DPG's range reporting data structure. Four different macromolecules were used, and for each of them all atoms were first randomly inserted into the data structure followed by the random deletion of all atoms. The reported insertion and deletion times are averages of four such independent runs. The average time for a single insertion/deletion was never more than 5 μs.

We also compared the performance of the range reporting data structure used by DPG with the 3D hashing used in [19, 20] to produce molecular surfaces. As our experimental setup we used the same implementation of 3D arrangement and surface generation [20], but switched between the two different range query data structures. We measured the space and time requirements for generating the surface of various molecules and macromolecules. In addition to the molecules used in the experiments of [19, 20], we ran our experiments on some viruses and ribosomes we are interested in. To verify scalability, multiple chains of the same protein were inserted. For virus capsids as multiple chains are

Quadr. Points	Query Distance (Å)				Avg. Time (ms) / Query				Avg. # Points Returned / Query				Avg. # Points Returned / ms			
45,654	2	4	8	16	0.311	0.566	1.420	3.379	118	775	4,466	22,839	379	1,367	3,144	6,758
91,309	2	4	8	16	0.588	1.139	2.801	6.158	225	1,623	9,284	44,518	382	1,425	3,314	7,229
136,963	2	4	8	16	0.973	1.845	4.436	9.572	329	2,435	14,496	70,016	338	1,320	3,268	7,314
182,618	2	4	8	16	1.304	3,219	5.855	12.661	439	3,401	19,307	93,443	377	1,314	3,297	7,381

Table 2: Performance of the Query function of packing grid. We take a molecule (1HPT: a variant of human pancreatic trypsin inhibitor) consisting of about 850 atoms after protonation using PDB2PQR [1], and sample approximately 184,000 quadrature points uniformly at random on its surface. We randomly assign each point to one of four groups and thus obtain four approximately equal-sized groups. We then run queries from the 800 atom centers (100 queries per atom) on group 1; merge groups 1 and 2, and run queries on this merged group; merge groups 1, 2 and 3, and run queries again; and finally run queries on the entire set.

Molecule (PDB File)	Number of Atoms	Insert		Delete	
		Total Time (ms)	Avg. Time (μs)	Total Time (ms)	Avg. Time (μs)
GroEL (1GRL)	29,274	97	3.3	118	4.0
RDV P8 (1UF2: Chain P)	193,620	746	3.9	846	4.4
RDV P3 (1UF2: Chain A)	459,180	1,813	3.9	2,094	4.6
Dengue (1K4R)	545,040	2,176	4.0	2,432	4.5

Table 3: Insertion and deletion times of our current packing grid implementation. The results are averages of 4 runs. In each run, all atom centers are randomly inserted into the data structure followed by random deletion of all atom centers.

Molecule (PDB File)	Number of Chains	Number of Atoms	Number of Cells		Time (sec)	
			DPG	3D hash [20]	DPG	3D hash [20]
Trypsin Inhibitor (4PTI)	1	454	196	1,089	0.58	0.54
Carbonic Anhydrase I (1BZM)	1	2,034	856	3,360	2.73	2.58
Fasciculin2 - Acetylcholinesterase (1MAH)	1	4,116	1,726	8,568	6.20	5.70
Anthrax Lethal Factor - MAPKK2 (1JKY)	1	5,614	2,389	16,456	8.52	8.13
RNA Polymerase II (1I3Q)	1	11,114	4,682	45,177	17.36	16.23
Glutamine Synthetase (2GLS)	1	3,636	1,444	9,177	5.43	5.06
	5	18,180	7,275	41,400	37.10	34.80
Nicotinic Acetylcholine Receptor (2BG9)	1	2,991	1,199	10,752	4.44	4.29
	5	14,955	6,027	31,200	24.31	22.95
Rice Dwarf Virus (RDV) P8 (1UF2: Chain P)	1	3,227	1,348	9,261	4.47	4.23
	2	6,454	2,739	1,124,040	9.23	8.56
	3	9,681	4,115	2,506,480	15.17	14.31
	4	12,908	5,467	4,426,110	19.36	18.14
	5	16,135	6,848	4,426,110	30.79	30.20
	6	19,362	8,224	6,052,800	35.65	34.42
	7	22,589	9,605	6,052,800	40.28	38.86
	8	25,816	10,981	6,332,160	45.22	44.44
Rice Dwarf Virus (RDV) P3 (1UF2: Chain A)	1	7,653	3,229	38,760	10.99	10.23
	2	15,306	6,458	927,442	22.73	21.44
	3	22,959	9,739	1,992,747	40.48	39.62
	4	30,612	12,985	2,591,700	119.28	128.37
Dengue Virus (1K4R: Chains A & B)	2	6,056	2,622	20,706	8.46	7.71
	4	12,112	5,237	138,600	17.56	16.52
	6	18,168	7,846	333,060	33.73	32.62

Table 4: Comparison of the performance of the 3D range reporting data structure used by DPG, and the 3D hash table used in [20]. The same 3D arrangement code was used in both cases [20]. Table shows the comparative running times and the space requirement (in terms of the number of cells used) for surface generation of different molecules. To verify scalability, molecules of varying sizes and in some cases, multiple chains were used. To generate multiple copies of the molecule we used the transformation matrices given in the corresponding PDBs (e.g., to generate k copies we used the top k matrices).

inserted, not only the number of atoms increases but also the overall structure becomes sparser. For example, Figure 5 shows that though a single chain is dense, if four chains are considered together then their bounding volume becomes sparse. The results of this experiment are reported in Table 4. From the table, one can verify that the space requirement of the DPG range query data structure is linear in the number of atoms. Also, its running times are comparable with that of 3D hash while using much less memory. The difference in space requirement becomes more pronounced for larger and sparser structures. Though 3D hash performs insertions and queries in optimal constant time, using too much memory can adversely affect its running time when the set of atoms is sparse as in virus capsids. For example, in the case of RDV P3 with 4 chains, 3D hash operations run slower than DPG range reporting operations. We believe

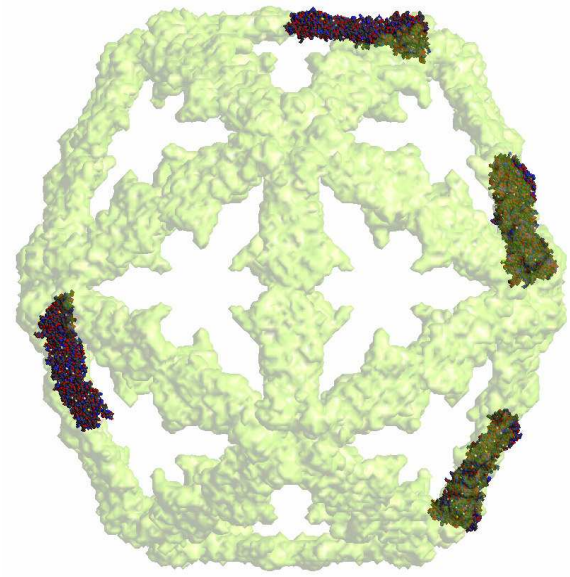


Figure 5: RDV capsid protein P3 chain A. The entire structure generated by applying all sixty transformations is rendered in transparent green. The chains generated by the first four transformations are rendered opaque and in atom-based coloring.

that this slowdown is due to page faults caused by excessive space requirement of 3D hash.

Acknowledgements. We would like to thank Aditi Saha for her contributions during the initial stages of implementing DPG. We are also thankful to Eran Eyal and Dan Halperin for giving us access to their C code for dynamic maintenance of flexible molecular surfaces [19, 20]. Our implementation of the packing grid data structure uses the C implementation of "cuckoo hashing" by Rasmus Pagh and Flemming Rodler [39].

7. REFERENCES

[1] PDB2PQR: An automated pipeline for the setup, execution, and analysis of Poisson-Boltzmann electrostatics calculations. http://pdb2pqr.sourceforge.net/.

[2] A. A. and W. Z. VRDD: applying virtual reality visualization to protein docking and design. *Journal of Molecular Graphics and Modelling*, 17(7):180–186, 1999.

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

[4] A. Arkhipov, P. L. Freddolino, K. Imada, K. Namba, and K. Schulten. Coarse-grained molecular dynamics simulations of a rotating bacterial flagellum. *Biophy. J.*, 91:4589–4597, 2006.

[5] C. Bajaj. A Laguerre Voronoi based scheme for meshing particle systems. *Japan Journal of Industrial and Applied Mathematics*, 22(2):167–177, June 2005.

[6] C. Bajaj, H. Y. Lee, R. Merkert, and V. Pascucci. NURBS based B-rep models for macromolecules and their properties. In *Proceedings of the 4th ACM Symposium on Solid Modeling and Applications*, pages 217–228, New York, NY, USA, 1997. ACM.

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

[8] C. Bajaj, V. Pascucci, A. Shamir, R. Holt, and A. Netravali. Dynamic maintenance and visualization of molecular surfaces. *Dis. App. Math.*, 127(1):23–51, 2003.

[9] C. Bajaj and W. Zhao. Fast molecular solvation energetics and forces computation. Technical Report ICES TR-08-20, The University University of Texas at Austin, 2008.

[10] S. Batsanov. Van der Waals radii of elements. *Inorganic Materials*, 37:871–885(15), September 2001.

[11] K. L. Clarkson, H. Edelsbrunner, L. J. Guibas, M. Sharir, and E. Welzl. Combinatorial complexity bounds for arrangements of curves and spheres. *Discrete Comput. Geom.*, 5(2):99–160, 1990.

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

[13] M. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221(4612):709–713, 19 August 1983.

[14] S. Crivelli, O. Kreylos, B. Hamann, N. Max, and W. Bethel. Proteinshop: A tool for interactive protein manipulation and steering. *Journal of Computer-Aided Molecular Design*, 18(4):271–285, 2004.

[15] E. Demaine. Advanced data structures (6.897) lecture notes (MIT), Spring 2005. http://courses.csail.mit.edu/6.897/spring05/lec/lec09.pdf.

[16] B. Duncan and A. Olson. Approximation and characterization of molecular surfaces. *Biopolymers*, 33(2):219–229, February 1993.

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

[18] M. T. M. Emmerich, J. W. Kruisselbrink, E. van der Horst, A. P. IJzerman, A. Bender, and T. Bäck. Combined interactive and automated adaptive search for molecular design. In *Proceedings of Adaptive Computing in Design and Manufacture*, 2008.

[19] 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, New York, NY, USA, 2005. ACM.

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

[21] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

[22] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Mon. Not. Roy. Astron. Soc.*, 181:375–389, November 1977.

[23] J. Grant and B. Pickup. A gaussian description of molecular shape. *Journal of Physical Chemistry*, 99:3503–3510, 1995.

[24] D. Halperin and M. H. Overmars. Spheres, molecules, and hidden surface removal. In *SCG '94: Proceedings*

of the 10th Annual Symposium on Computational Geometry, pages 113–122, New York, NY, USA, 1994. ACM.

[25] R. Hockney and J. Eastwood. *Computer Simulation Using Particles*. McGraw Hill, 1981.

[26] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[27] R. M. Jackson, H. A. Gabba, and M. J. E. Sternberg. Rapid refinement of protein interfaces incorporating solvation: application to the docking problem. *Journal of Molecular Biology*, 276(1):265–285, February 1998.

[28] O. Kreylos, N. L. Max, B. Hamann, S. N. Crivell, and E. W. Bethel. Interactive protein manipulation. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 77, Washington, DC, USA, 2003. IEEE Computer Society.

[29] J. W. Kruisselbrink, T. Bäck, A. P. IJzerman, and E. van der Horst. Evolutionary algorithms for automated drug design towards target molecule properties. In *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 1555–1562, New York, NY, USA, 2008. ACM.

[30] E.-W. Lameijer, J. N. Kok, T. Bäck, and A. P. IJzerman. The molecule evoluator. an interactive evolutionary algorithm for the design of drug-like molecules. *Journal of Chemical Information and Modeling*, 46(2):545–552, 2006.

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

[32] E. Lindahl and O. Edholm. Mesoscopic undulations and thickness fluctuations in lipid bilayers from molecular dynamics simulations. *Biophys. J.*, 79:426–433, 2000.

[33] T.-C. Lu, J. Ding, and S. N. Crivelli. DockingShop: a tool for interactive protein docking. In *2005 IEEE Computational Systems Bioinformatics Conference - Workshops*, pages 271–272, 2005.

[34] S. J. Marrink and A. E. Mark. Effect of undulations on surface tension in simulated bilayers. *J. Phys. Chem. B*, 105:6122–6127, 2001.

[35] P. G. Mezey. *Shape in Chemistry; An introduction to molecular shape and topology*. VCH Inc, 1993.

[36] Y. Modis, S. Ogata, D. Clements, and S. C. Harrison. A ligand-binding pocket in the dengue virus envelope glycoprotein. *Proceedings of the National Academy of Sciences*, 100(12):6986–6991, 2003.

[37] C. W. Mortensen, R. Pagh, and M. Pătraçcu. On dynamic range reporting in one dimension. In *STOC '05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 104–111, New York, NY, USA, 2005. ACM.

[38] M. T. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L. V. Kale, R. D. Skeel, and K. Schulten. NAMD: a parallel, object-oriented molecular dynamics program. *International Journal of High Performance Computing Applications*, 10(4):251–268, 1996.

[39] R. Pagh and F. Rodler. Cuckoo hashing (C code). http://www.it-c.dk/people/pagh/papers/cuckoo.tar.

[40] R. Pagh and F. Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[41] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005. 10.1002/jcc.20289.

[42] F. Richards. Areas, volumes, packing, and protein structure. *Annual Review of Biophysics and Bioengineering*, 6:151–176, June 1977.

[43] B. Sandak, H. J. Wolfson, and R. Nussinov. Flexible docking allowing induced fit in proteins: insights from an open to closed conformational isomers. *Proteins: Structure, Function, and Genetics*, 32(2):159–174, December 1998.

[44] 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.

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

[46] A. N. Sedwell and I. C. Parmee. Techniques for the design of molecules and combinatorial chemical libraries. In *2007 IEEE Congress on Evolutionary Computation*, pages 2435–2442, 2007.

[47] A. Y. Shih, I. G. Denisov, J. C. Phillips, S. G. Sligar, and K. Schulten. Molecular dynamics simulations of discoidal bilayers assembled from truncated human lipoproteins. *Biophys. J.*, 88:548–556, 2005.

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

[49] J. Stone, J. Gullingsrud, P. Grayson, and K. Schulten. A system for interactive molecular dynamics simulation. In J. F. Hughes and C. H. Séquin, editors, *2001 ACM Symposium on Interactive 3D Graphics*, pages 191–194, New York, 2001. ACM SIGGRAPH.

[50] A. Varshney and F. Brooks. Fast analytical computation of Richards's smooth molecular surface. In *Proceedings of the 4th Conference on Visualization*, pages 300–307, 1993.

[51] A. Varshney, J. Frederick P. Brooks, and W. V. Wright. Computing smooth molecular surfaces. *IEEE Comput. Graph. Appl.*, 14(5):19–25, 1994.

[52] M. Vieth, J. D. Hirst, A. Kolinski, and C. L. Brooks III. Assessing energy functions for flexible docking. *Journal of Computational Chemistry*, 19(14):1612–1622, 1998.

[53] 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.

[54] D. Willard. Log-logarithmic worst-case range queries are possible in space n. *Information Processing Letters*, 17(2):81–84, 1983.

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