Proc. of the Fourth Int. Symposium on Large Spatial Databases (SSD `95),
Portland, Maine, USA, Lecture Notes in Computer Science, Springer, 1995.

A 3D Molecular Surface Representation Supporting Neighborhood Queries[1]

Thomas Seidl and Hans-Peter Kriegel

Institute for Computer Science, University of Munich
Leopoldstr. 11 B, D-80802 München, Germany
email: { seidl | kriegel }@dbs.informatik.uni-muenchen.de

Abstract: Applications in molecular biology more and more require geometric data management along with physicochemical data handling. Thus, 3D structures and surfaces of molecules become basic objects in molecular databases. We propose the neighborhood query on graphs such as molecular surfaces as a fundamental query class concerning topological information on patch adjacency. Furthermore, we suggest a patch-based data structure, called the *TriEdge structure,* first, to efficiently support neighborhood query processing, and second, to save space in comparison to common 2D subdivision data structures such as the quad-edge structure or the doubly-connected edge list. In analogy to the quad-edge structure, the TriEdge structure has an algebraic interface and is implemented via complex pointers. However, we achieve a reduction of the space requirement by a factor of four. Finally, we investigate the time performance of our prototype which is based on an object-oriented database management system.

Keywords: 3D molecular modeling, graphs in spatial databases, surface representation, neighborhood query, surface approximation, database systems in molecular biology.

1 Introduction

The fundamental 3D objects in molecular biology and computational biochemistry are large molecules with several hundreds to thousands of atoms. There are various applications that require access to the 3D structure of the molecules, as it is provided for proteins by the Brookhaven Protein Data Bank (PDB) [Ber 77]. Up to now, the PDB contains 3,000 proteins, enzymes, and viruses [PDB 95]. For each entry, along with information on the chemical structure of the protein, the 3D coordinates of its atoms are stored in a text file.

In the last years, a new topic has been emerging in the area of protein engineering: the prediction of molecular interaction, called the *docking problem.* Several methods has been suggested to meet the one-to-one docking problem [Con 86], [BMH 92], [FNNW 93], [HT 94]: which constellation of two given proteins represents a stable complex? A constellation is the relative position of a molecule with respect to its docking partner, and may be described by six parameters: three coordinates for translation in 3D space, and three Euler angles for rotation (cf. figure 1). Since all of these six degrees of freedom to compose two molecules together in 3D space are continuous, the constel-

1. This research was funded by the German Ministry for Research and Technology (BMFT) under grant no. 01 IB 307 B. The authors are responsible for the content of this paper.

lation space is infinite. Common discretizations of protein surfaces result in thousands of points, and each of them can represent a possible docking site.

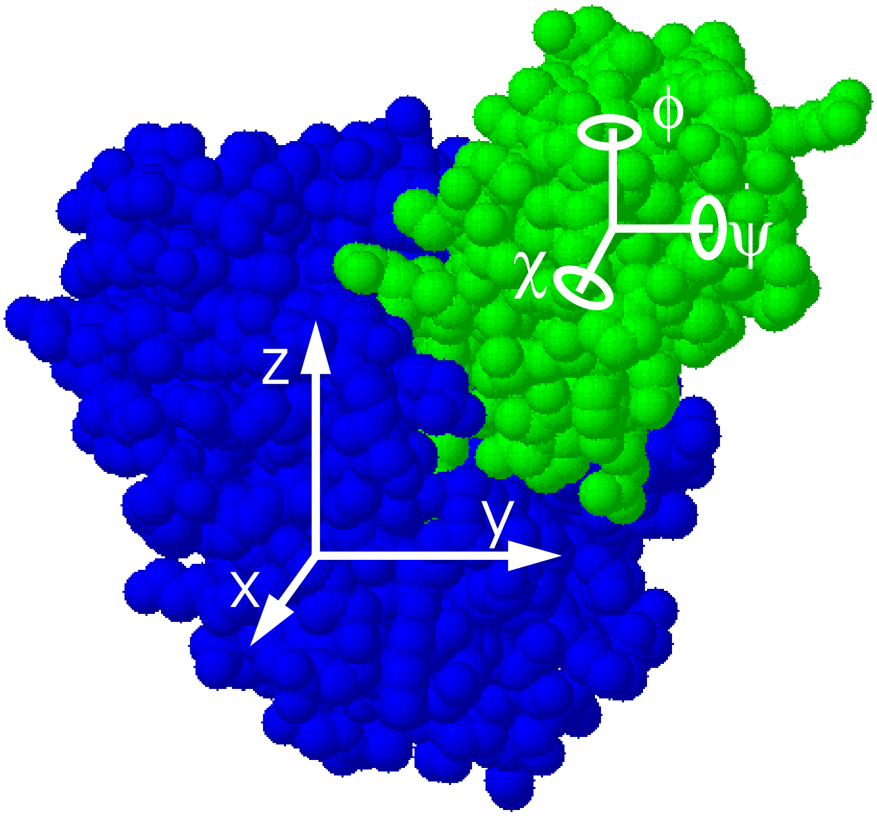


Fig. 1. Example complex: beta trypsin with trypsin inhibitor (PDB code 2PTC)

In our project "BIOWEPRO: a Database System for Protein Docking", we are faced with the one-to-many docking problem: select such proteins from the PDB that are able to form a stable complex in interaction with a given query protein, and determine appropriate constellation parameters. When considering the number of proteins in the database ($3 \cdot 10^3$) and the number of possible docking sites on a protein in the database ($> 10^3$) and on the query protein ($> 10^3$), the search space at least has an overall size of billions of protein-protein-constellations [EKSX 95].

Thus, docking retrieval is a new and challenging application for spatial database systems. Due to the enormous size of the search space, a multi-step query processing architecture is recommended. In spatial database systems, this paradigm efficiently supports the processing of point and region queries as well as spatial joins [BHKS 93], [KSB 93], [BKSS 94]. Additionally, we perform various steps of preprocessing: first, we determine various geometric and physicochemical features of the molecular surface, e.g. local shape index values [Koe 90]. From these, we build up a feature index [Ald 94] to improve the filter step. Second, we apply a segmentation method to diminish the constellation space [EKX 95].

The main object of interest in protein docking is the 3D molecular surface, rather than the 3D arrangement of the atoms a molecule consists of. Moreover, the *local shape* of the surface at a possible docking site is the fundamental geometric criterion for docking retrieval, rather than the location of a docking site in space. This is analogous to similarity retrieval in CAD databases [SKSH 89]. Since the interaction of molecules is restricted to the docking sites, we do not require a description of the global shape (outline, contour) of molecules. An access method for molecular surfaces has to support a representation of possible docking sites that is independent from location and direction in space, i.e. invariant with respect to translation and rotation. Along with the geometry, also physicochemical properties determine molecular interactions and, therefore, have to be considered when representing molecular surfaces.

In section 3, we will see that molecular surfaces are smooth but quite bumpy (uneven). Since all of the bumps have similar atomic size, the selectivity of a local shape index would be very low when based on infinitesimal neighborhoods. Instead, for each surface primitive (patch or point, resp.), we collect its neighbors within an appropriate radius, and determine the local shape via a paraboloid as a simple approximation. The basic retrieval operation is the *neighborhood query:* select a connected set of patches (or points) around a given patch (or point, resp.) bound by a neighborhood condition, e.g. a distance or similarity criterion.

Another application of the neighborhood query is the segmentation of protein surfaces: adjacent surface elements will be grouped to segments as long as they are similar. We will change our notion of docking site from the surface elements mentioned above to these segments. Due to the resulting homogeneity within the segments, we expect a high effectivity for the docking retrieval, and due to the reduction of the number of docking sites, we expect an improved efficiency. For segmentation, a region growing method is performed: we select elements that are interesting in some sense and, for each of them, we perform a neighborhood query controlled by an appropriate similarity criterion.

Neighborhood query processing requires access to adjacency information, as it is provided by common data structures like the quad-edge structure [GS 85], or the DCEL [PS 85]. In this paper, we propose a new basic technique to store molecular surfaces, reducing the storage requirement by a factor of four. Since our method is based on topology, it supports effective and efficient processing of queries on the molecular surface structure. We formally introduce the general neighborhood query, and give an algorithm to process it. Two applications are considered in detail: local surface approximation and surface segmentation. We implemented our method in C++ on top of the object-oriented database management system ObjectStore [OHMS 92].

The paper is organized as follows: in section 2, we refer to related work, and in section 3, we review a common subdivision of molecular surfaces and give a specification to represent their topological structure. In section 4, we formally define the neighborhood query and show its application to surface approximation and segmentation. In section 5, the TriEdge data structure is presented that supports efficient processing of neighborhood queries. Section 6 contains first evaluation results, and section 7 concludes the paper with a summary and an outlook to future work.

2 Related Work

As we will illustrate in section 3, molecular surfaces are 2-manifolds that can be represented by a common subdivision. Therefore, we investigate the literature for any hints how to efficiently store 2D subdivisions in database systems. The articles describing molecular surface calculation as found in molecular biology [Con 83], computer graphics [VBW 94], and computational geometry [HO 94] do not mention their surface representation method. In [HO 94] only, there is a hint that extended van-der-Waals surfaces are stored with the quad-edge structure. More investigations of surface graph structures and retrieval are found in the area of solid modeling, graphs in spatial databases, and graph algorithms for recursive query processing.

In the field of solid modeling, the quad-edge structure is a quite common storage method for 2D subdivisions [GS 85]. As a short review to the concepts behind the quad-edge structure, we cite from [GS 85], page 80, while carefully adapting the notation to the style as required in our implementation environment: "For any oriented and directed edge *e* we can define unambiguously its vertex of *origin, e.Orig(),* its *destination, e.Dest(),* its *left face, e.Left(),* and its *right face, e.Right().* We define also the *flipped* version *e.Flip()* of an edge *e* as being the same unoriented edge taken with *opposite orientation* and same direction, as well as the *symmetric* of *e, e.Sym(),* as being the same undirected edge with the *opposite direction* but the same orientation as *e*." On page 81, the basic traversal functions are introduced: "We can define the *next edge with same origin, e.LEdge(),* as the one immediately following *e* (counterclockwise) in this ring. Similarly, given an edge *e* we define the *next counterclockwise edge with same left face,* denoted by *e.Lnext(),* as being the first edge we encounter after *e* when moving along the boundary of the face F = *e.Left()* in the counterclockwise sense as determined by the orientation of F. The edge *e.Lnext*() is oriented and directed so that *e.Lnext().Left()* = F (including orientation)."

The quad-edge structure is implemented by records representing four views to a single undirected edge, or two directed half-edges, respectively. Each edge record contains four pointers *Data*, two of them to the adjacent vertices, providing the access required for the evaluation of *Orig()* and *Dest()*, and the remaining two to the adjacent faces *Left()* and *Right()*. The traversal functions *Sym(), LEdge(), Lnext()* etc. are supported by four complex pointers (e,r) from an edge record to the edges following in clockwise and counterclockwise direction around the corresponding vertices *Orig()* and *Dest()* (cf. figure 2). This way, a cycle (e,r).*Sym().Sym()* = (e,r) is stored as a whole in one record: the evaluation of (e,r).*Sym()* does not require dereferencing the pointer r, but only changing the 'view' component r, whereas the other operations like *LEdge()* require dereferencing t. In the TriEdge structure, *LEdge()* cycles are clustered rather than all of the *Sym()* cycles (cf. section 5).

Since molecular surfaces are 2-manifolds, we do not require generalizations as for the modeling of (non-planar) 3D subdivisions [DL 89] or of subdivided d-manifolds for arbitrary dimensions d [Bri 93]. Molecular surfaces are orientable, therefore we drop the *Flip()* operation. For reasons of static typing, we want to distinguish explicitly between the surface graph and its dual graph, and thus, do not provide the operation *Rot()* which serves for changing the view to the dual of the graph, exchanging vertices with faces. Whereas supporting *Flip()* would cost a few bits per edge record, *Rot()* would be for free.

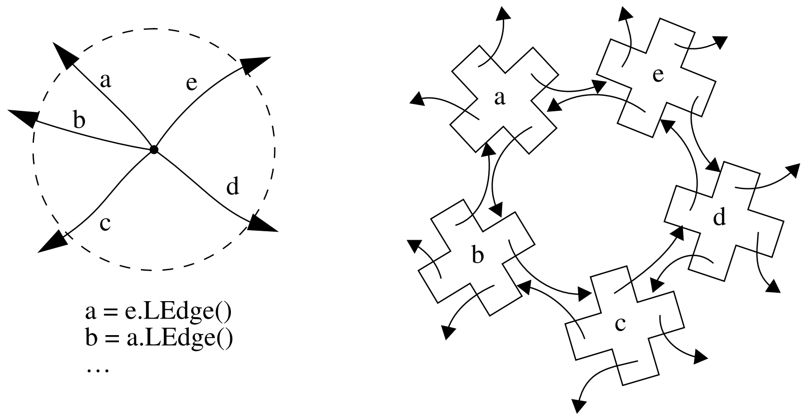


Fig. 2. An LEdge ring and the corresponding linked quad-edge records (cf. [GS 85])

The quad-edge implementation is similar to the doubly-connected-edge-list (DCEL) from [PS 85], with two differences: first, the quad-edge structure is defined in terms of an edge algebra, leading to a more comfortable interface to edges, their symmetric and dual edges, in comparison to the DCEL. Second, the complex pointers of the quad-edge structure contain a simple pointer to an adjacent edge record, and additionally, unlike a DCEL entry, an offset value representing the view to the referenced edge. Exactly this idea of an algebraic interface together with a complex pointer concept is used for our TriEdge structure as described below. This approach is well supported by the object-oriented data model and leads to an easy integration.

In [DMP 93], a unified topological model called Plane Euclidean Graph (PEG) is proposed to integrate the various domains of information a spatial database system is concerned with. Spatial queries are classified into topological, set-theoretic, and metric queries. Our neighborhood query obviously qualifies for the topological class, but also for the metric or a thematic one, dependent on the neighborhood criterion specified. As a data structure for the PEG, a modification of the DCEL is proposed that additionally can hold isolated points as well as isolated edge components inside faces.

Questions concerning graphs in spatial databases also are investigated in [EG 94]. The graphs consist of nodes, edges, and explicit paths, and are used to represent networks. The examples are taken from the domain of geographical information systems with highways, roads, and rivers, etc. as instances of explicit paths. A type `REG` is mentioned occurring as parameter for operations `inside` and `intersection`, but the concept of faces that are defined inherently by edge cycles is not considered explicitly. In our application domain, we require these faces e.g. for visualizing molecular surfaces. The faces represent the patches of a molecular surface, and provide access to the underlying solid which is an atom sphere, a probe sphere, or a torus. The edge cycles bounding the faces represent the bordering arcs that form the trimming curves when rendering the patches. The dual view of the graph — considering the patches to be vertices instead of faces — helps when traversing the surface graphs.

As an interesting contribution in [EG 94], the selection of a subgraph around a node is provided by the operation circle: $graph_i \times node_i \times num_j \times (edge_i \rightarrow num_j) \rightarrow graph_i$. The circle function is specified by parameters for the graph, center, and radius along with an edge cost function, and is processed performing Dijkstra's single-source shortest path algorithm. The specification of our conditional neighborhood is very similar to this concept. As an extension, we do not restrict it to the case of a radius bounding the path length, but support general conditions as relations between an arbitrary node and the center of the neighborhood. Additionally, the neighborhood query also supports symbolic similarity criteria and is not restricted to numerical distance comparisons.

Our system is based on an object-oriented data model. In [Jia 92] and [Jia 94], graph structures in relational databases are considered for the purpose of effective expression and efficient processing of recursive queries. For instance, SQL is extended by path expressions for the specification of transitive closures of relations. The graph structure of recursive relations is exploited to perform efficient query processing via graph algorithms. In consideration of a paging environment, algorithms for efficient graph traversal are analyzed. As a result, breadth-first traversal algorithms appear to be preferable.

3 Molecular Surfaces

In molecular biology, the surface of a molecule is defined to be the solvent accessible surface for any solvent probe radius α [Ric 77]. This surface is equivalent to the boundary of the weighted 3D α-hull: for $\alpha > 0$, the α-*hull* of a set s of spheres is the complement of the union of all open spheres of radius α intersecting no sphere of s [VBW 94]. For $\alpha = 0$, the α-hull is equal to the union of all spheres of s, and for $\alpha \rightarrow \infty$, the α-hull of s coincides with the convex hull of s. Various implementations for the calculation of molecular surfaces are published [Con 83], [VBW 94], [Sch 94], [HO 94].

Molecular surfaces have a strong regularity, and there are three types of patches they consist of: convex spherical patches, saddle-shaped rectangles, and concave spherical triangles (cf. figure 3). These types depend on the number of atoms that the probe sphere is in simultaneous contact with when rolling over the molecule. Since the algorithms enforce the atoms to be in general position, ensuring the probe sphere never being in simultaneous contact with more than three atoms, the complexity of the algorithms is reduced drastically. Obviously, this also holds for the representation of molecular surfaces.

At first sight, such patch-based representations of molecular surfaces seem to be difficult to manage: there are different types of patches with a different number of vertices, etc. Thus, triangulated surface representations are quite common to be used for docking purposes, and also for visualization. However, there are strong advantages for the patch-based representation: first, for a particular molecule and a given probe radius α, the solvent accessible surface is well defined in its structure and its shape. Second, for any given point density, the patchwork can be refined to a dotted surface, but not vice versa. Third, every patch is homogeneous with respect to the curvature, and the normal vector of a surface point is determined by the geometric parameters of the patch it lays on, i.e. via the associated atom sphere, torus, or probe sphere, respectively. Overall, we prefer a patch-based surface representation rather than a point-based method or an arbitrary triangulation.

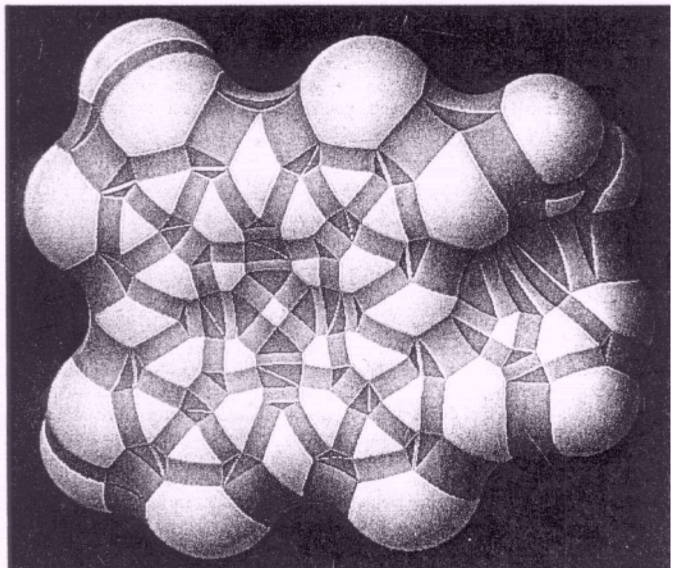


Fig. 3. Surface of a portion of hemoglobin [Con 83].

In the following, we present a patch-based representation for molecular surfaces, providing access to the patches via an edge algebra. Guibas and Stolfi introduced such an algebra for 2D subdivisions as a specification for their quad-edge structure [GS 85]. Since the quad-edge structure is general enough to store 2D subdivisions which are embedded in 3D space, it is an appropriate storage structure for subdivisions of molecular surfaces as described above: let the faces stand for the surface patches, the edges for the arcs bordering each patch, and the vertices for the corner points of the patches. This view is adequate e.g. for visualization purposes, since the patches provide access to the underlying solid being an atom sphere, a probe sphere, or a torus. The edge cycles bounding the faces represent the bordering arcs that form the trimming curves required for rendering the patches. The dual view to this surface graph is intuitive for traversal purposes: consider the patches to be the vertices of the graph, the edges remain being associated to the arcs, and the faces represent the corner points (cf. table 1).

molecular surface	2D subdivision	ditto, dual view
corner point	vertex	face
patch border arc	edge	edge
surface patch	face	vertex

Tab. 1. Components of molecular surfaces, mapped to components of a 2D subdivision

As an interface for our surface representation, we specify required edge functions as follows. An illustration of the operations is given in figure 4. We implement this specification as a C++ module for the object-oriented database system ObjectStore. How-

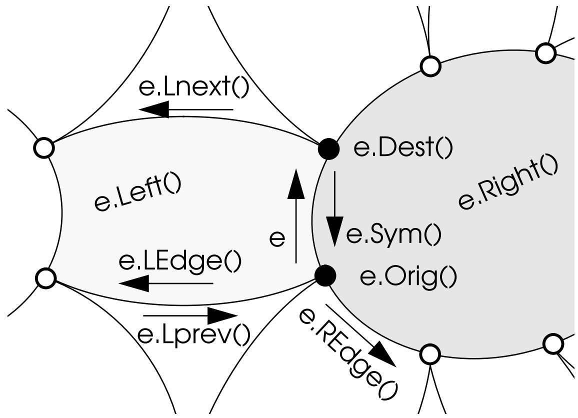


Fig. 4. Illustration of the MolSurface algebra

ever, to demonstrate the relationships between the functions, we give an algebraic specification with axioms. From the `EDGE` → `EDGE` functions, `Sym` and `LEdge` or `Sym` and `Lnext` are primitive, the others are derived with respect to the axioms:

```
SPEC MolSurface;
 TYPES EDGE, FACE, VERTEX;
 FUNCTIONS
     Sym, LEdge, REdge,
      Lnext, Lprev:        EDGE -> EDGE;
     Left, Right:          EDGE -> FACE;
     Orig, Dest:           EDGE -> VERTEX;
     an_edge:              FACE -> EDGE;
     an_edge:              VERTEX -> EDGE;
 AXIOMS (for all EDGE e, FACE f, VERTEX v)
     e.Sym().Sym() == e;
     e.REdge().LEdge() == e;
     e.Lnext() == e.Sym().REdge();
     e.Lprev() == e.LEdge().Sym();
     e.Sym().Orig() == e.Dest();
     e.LEdge().Orig() == e.Orig();
     f.any_edge().Left() == f;
     v.any_edge().Orig() == v;
END MolSurface.
```

In addition to the algebra, we define an iteration statement to assign all the edges e around a face or a vertex counterclockwise to an edge variable e. Thereby, the edge cycle is traversed by the `Lnext` or `LEdge` function, resp., until the first edge is reached a second time. Syntactically, we map the iteration clause to a `for` statement via the macro mechanism of C++: `forall_around_face(e,face){...}` and `forall_around_vertex(e,vertex){...}`.

Until now, we only specified the topological structure of molecular surfaces. To yield the location of the vertices and some representative points of the patches, further functions provide access to geometric attributes, e.g. location: `VERTEX` $\rightarrow$ `3D_POINT`, and representative: `FACE` $\rightarrow$ `3D_POINT`. Similar functions deliver thematic attributes and shape information.

In this section, we specified our patch-based representation of molecular surfaces. In the following, we show how neighborhood queries are performed when they are based on this specification, and in section 5, we explain our MolSurface algebra implementation.

4 Neighborhood Queries

As mentioned above, neighborhood queries are required for various applications such as local surface approximation and surface segmentation. Therefore, we select such patches or points from the surface that fulfill a particular condition, e.g. an euclidean distance criterion in case of the approximation, or a criterion of similarity in case of the segmentation.

Formally, we specify the conditional neighborhood of a patch p as follows: for any molecular surface graph g and predicate c over a pair of patches, the *c-neighborhood* of a patch p, $n_c(p)$, is defined to be the maximal connected subgraph of g restricted to the patches v that fulfill the predicate c(v,p). Thus, for each $v \in n_c(p)$, one of the following properties holds: (i) c(v,p) and (v=p), or (ii) c(v,p) and an adjacent patch of v is in $n_c(p)$. In other words we can say: $n_c(p)$ contains only patches p that (1) fulfill the condition c together with p: c(v,p), and (2) are reachable from p via patches that all belong to $n_c(p)$.

The neighborhood query simply consists in the selection of a c-neighborhood: for any patch p of the surface s, and any predicate $c(\bullet,\bullet)$, select all patches from s that belong to $n_c(p)$. In the following, we explain two applications of neighborhood queries.

Various applications of the general concept of neighborhood queries can be thought of. The condition c can be a criterion of surface distance, equivalence, etc. We give three examples taken from the context of docking retrieval.

(1) Segmentation of molecular surfaces. As a condition c, we provide a similarity criterion, e.g. two patches are similar if they both are saddle-shaped, or if they have the same sign for the electrostatic potential. We perform segmentation via a region growing algorithm: therefore, we have to provide a set of patches to be used as sprouts. For each of the patches in the sprout set, we perform a neighborhood query that is controlled by condition c. The resulting segments may have various extensions on the surface and in 3D space, since they are not bound by a spatial distance criterion. Also, they may overlap. When providing a similarity criterion with a very low significance, a single segment can include the whole surface.

(2) Local approximation of the molecular surface. As an early step in the computation of the local shape index [Koe 90] of a patch p, we approximate the neighborhood of p on the surface, e.g. by a paraboloid (cf. figure 5). With a radius parameter r, the locality of the approximation is controlled. For instance, a so called euclidean neighborhood query consists in the selection of all the patches that are reachable from p within the euclidean distance r. In this case, the condition c simply is a comparison: c(v,p) = '$\mathrm{dist}_{\mathrm{euclid}}(v, p) < r$'.

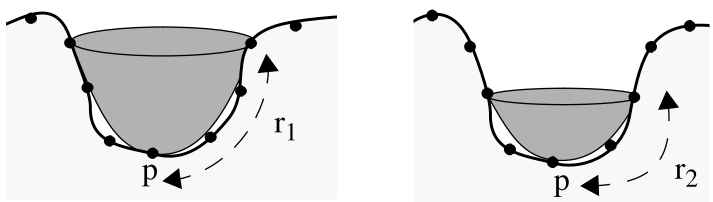


Fig. 5. Local approximation of the molecular surface with different radii r_1 and r_2

(3) Determination of local extrema. In [Con 86], knobs and holes are defined to be surface points that have an extreme solid angle value with respect to their neighbors. As a generalization, our neighborhood query supports specifying arbitrary neighborhood radii leading to different degrees of locality. Thus, various radii can be investigated with respect to their appropriateness for the purpose of docking retrieval.

Neighborhood query processing. After sketching some applications of the neighborhood query, we now present a simple implementation. We perform neighborhood query processing via a graph traversal algorithm:

```
void Neighborhood ( PATCH* p, CONDITION* cond,
                    os_Set <PATCH*>& result)
{
  os_List <PATCH*> open();
  PATCH *h, *v;
  result.clear();

  if (cond->eval(p,p))
    { open.insert(p); result.insert(p); }

  while (h = open.remove_first())
    // expand current patch h:
    forall_around_face( e, h )
    {
      v = e.Right();
      if (not result.contains(v)
          and cond->eval(v,p))
        { result.insert(v); open.insert(v); }
    }
  return;           // result is reference param
}
```

Two container objects are used in our algorithm: `result` and `open`. In `result`, we collect all the patches that belong to the specified neighborhood, and deliver it at the end of our procedure. The only methods applied to result are empty initialization, insertion, and lookup (contains). There is no relevance for any order on the patches, and, therefore, an arbitrary data structure supporting fast lookup and fast insertion is appropriate to hold the `result` variable. We decided to hold the `result` collection explicit in a container object, rather than to mark the patches that belong to `result`. For the latter, the database objects have to be modified during query processing, causing avoidable effort for concurrency control.

The other container object, `open`, is used to control the algorithm. It exactly contains such patches that are known to qualify for the result, and that have to be expanded later. A patch v that does not fulfill the condition c(v,p) will not be inserted into `open`, since none of its successors belongs to the specified neighborhood $n_c(p)$, except if it is reachable from p by another path not containing v. Each step of the iteration begins with the extraction of an element from `open`. We always fetch the first element from `open`, and since we always insert the new elements at the end, a breadth first search is performed on the surface. This strategy could be changed to other (heuristic) strategies by inserting new patches at appropriate positions into `open`. For instance, the insertion of patches at the beginning of `open` would result in a depth first search. The capability of specifying the traversal strategy requires `open` to be an ordered collection.

5 The TriEdge Data Structure

Up to here, we declared our objects of interest, specified access operations, and presented a basic query class with applications. We presented a specification for a surface representation that supports the expression of neighborhood queries. Since the schema is topology-based, providing connectivity information, neighborhood queries are supported very efficiently. The remaining question is how to implement this specification. In this section, we explain our technique, whereas in section 6, the space reduction factor is shown in detail, and a first runtime evaluation of our prototype is presented.

Obviously, the quad-edge structure qualifies for this purpose. However, it requires 36 bytes per record, when adjusted to a multiple of four. Real molecular surfaces have a size of some thousand patches and edges, for a common probe radius of 1.4 Å which is the size of a water molecule. Therefore, the storage requirement for the surface topology information of a single molecule is hundreds of kilobytes. In table 2, a sample is given for a few molecules from the PDB, which are identified by the PDB entry code. Since the number of proteins in the PDB currently is 3,000 [PDB 95], the size of our 3D protein database for docking retrieval comes into the range of gigabytes. Therefore, an efficient implementation is crucial.

Our data structure for the MolSurface specification reduces the storage requirement for topology information by a factor of four compared to a straightforward implementation by the quad-edge structure. The approach to implement the MolSurface specification consists of two steps: first, we reduce the complexity of the representation by mapping the molecular surface graph to an equivalent but simpler graph structure. Second, we exploit the strong regularity of the new structure as a key property for an efficient data structure. Overall, we save space and, therefore, time for the reduced amount of data transfer within the database system.

The main observation concerns the structure of molecular surfaces and the role of the saddle patches: every saddle patch connects two convex patches and two concave

molecule	PDB code	number of atoms	number of patches	number of edges	size for quad-edge records
Prealbumin, chain a	2PAB-A	872	3,710	7,380	260 kbytes
Prealbumin, chain b	2PAB-B	872	3,776	7,524	265 kbytes
Beta-trypsin	2PTC-E	1,629	5,606	11,160	393 kbytes
Trypsin inhibitor	2PTC-I	454	1,990	3,972	140 kbytes
Subtilisin novo	2SNI-E	1,983	6,220	12,396	436 kbytes
Chymotrypsin inhibitor	2SNI-I	513	2,116	4,224	149 kbytes

Tab. 2. Surfaces of real proteins

triangles. Both types, the convex and the concave patches, are surrounded by cycles of saddle patches. This is analogous to faces and vertices of a 2D subdivision graph that are surrounded each by a cycle of edges. We exploit the analogy to develop a new storage method for molecular surfaces, and call the new structure the *simplified surface graph* (SSG) (figure 6).

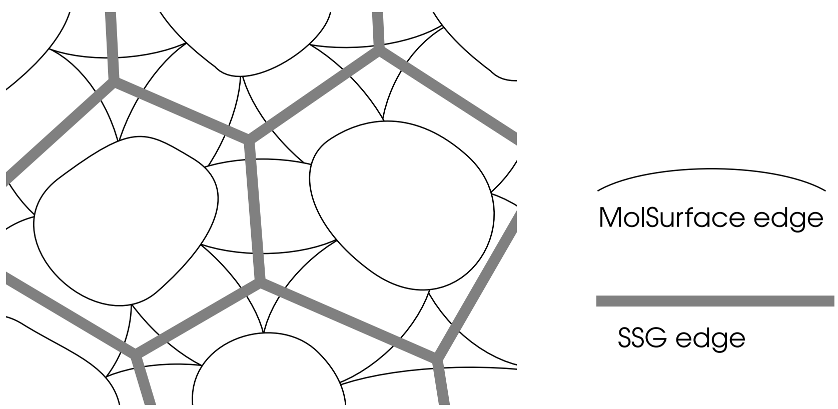


Fig. 6. Detail of a molecular surface: MolSurface edges and SSG edges

Due to this strong relationship to molecular surfaces, an SSG also is a 2D subdivision graph, consisting of vertices, edges, and faces. However, instead of mapping the vertices to corner points, the edges to arcs, and the faces to patches as we did for molecular surfaces, we now associate the vertices to the concave triangles of the molecular surface, the edges to the saddle patches, and the faces to the convex patches (cf. table 3). The number of edges to be stored is reduced drastically: every edge of a molecular surface graph belongs to exactly one saddle, and every saddle is bound by four (molecular) edges. Therefore, a molecular surface graph has four times as many edges as saddle patches, and, equivalently, four times as many edges than the SSG.

A second key observation leads us to a further reduction of the storage space. Since an SSG is a 2D subdivision, we could use the quad-edge structure to store it in the database. However, we can exploit a basic property of an SSG: all of the SSG vertices have a degree of three, since there are only triangles among the concave patches, due to the

molecular surface	molecular surface graph	simplified surface graph
corner point	vertex	—
patch border arc	edge	—
concave triangle	face	vertex
saddle rectangle	face	edge
convex patch	face	face

Tab. 3. Association of molecular surfaces to graph components

general position of the atoms: $\texttt{e.LEdge()}^3 = \texttt{e}$ for every $\texttt{EDGE}_{SSG}$ e. Therefore, we store the edge cycles around the SSG vertices in arrays with a fixed length of three, rather than using any dynamic structure. This observation leads us to the TriEdge structure which consists of records that contain three SSG half-edges, all of them belonging to the same $\texttt{LEdge}_{SSG}$ cycle. Similar to the quad-edge structure, an edge is represented by a complex pointer (t,v) that consists of a simple pointer t to the TriEdge record, and of a component v ('view') that specifies which SSG half-edge of the LEdge cycle is represented (cf. figure 7).

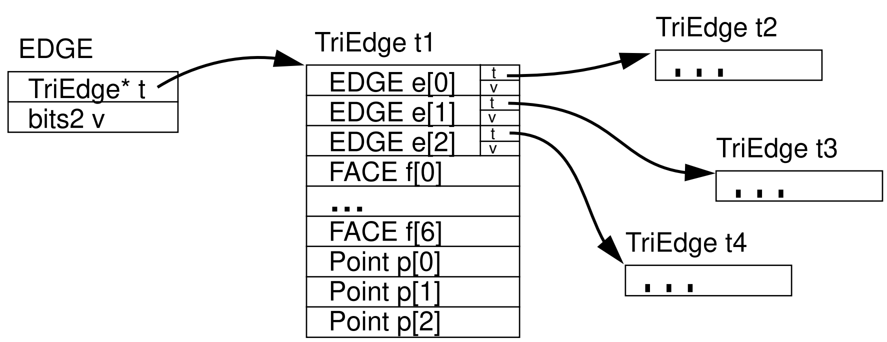


Fig. 7. The TriEdge data structure

A TriEdge record contains three complex edge pointers to store the connectivity information. Due to the mapping from SSG to molecular surface, a TriEdge record has to provide access to seven patches: the concave triangle as an image of the SSG vertex, the three saddle patches as images of the three SSG edges, and the three convex patches as images of the SSG faces that are adjacent to the saddles. Since the corner points are shared between the patches, we associate them to the TriEdge records rather than to the patches. For SSG, the implementation of the basic operations would be as follows:

$\texttt{LEdge}_{SSG}$: $\quad \mathbf{(t,v)} \rightarrow \mathbf{(t,\ v} \oplus_3 \mathbf{1)}$

$\texttt{Sym}_{SSG}$: $\quad \mathbf{(t,v)} \rightarrow \mathbf{(*t).e[v]}$

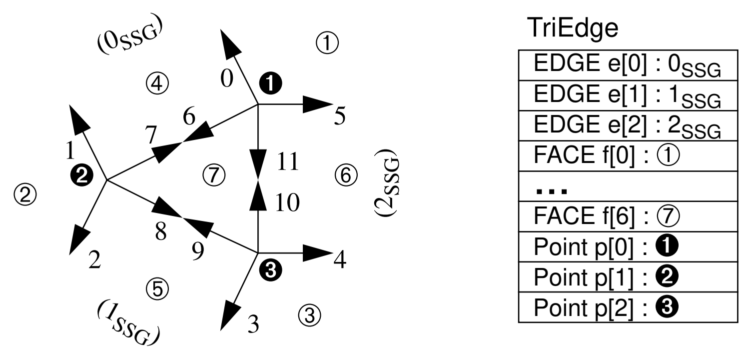
Since a TriEdge record represents three SSG half-edges, the view v is in the range [0..2]. For molecular surfaces, a TriEdge record represents twelve half-edges (cf.

figure 8a) and, therefore, requires the view v being in the range of `[0..11]`. Since all of the elements of an `LEdge` cycle are represented by the same TriEdge record, the `LEdge` operation requires no dereferencing of the TriEdge pointer of an edge, but only a change of the view v. This function is the same for all edges and, therefore, can be carried out via lookup in a constant table `LEDGE: [0..11]` → `[0..11]`, implemented simply as an array. For the `Sym` operation, dereferencing a TriEdge pointer is required in six cases, in the other six cases a change of the view is sufficient. When providing further arrays `SYM`, `LEFT`, and `ORIG`, the basic MolSurf operations are implemented as follows:

```
LEdge MolSurf:   (t,v) → (t, LEDGE[v])
Sym MolSurf:     (t,v) → if (v >= 6) then (t, SYM[v])
                             else (*t).e[v/2].correct(v)
Left MolSurf:    (t,v) → (*t).f[LEFT[v]]
Orig MolSurf:    (t,v) → (*t).p[ORIG[v]]
```

In figure 8b, the arrays mentioned are shown. The asterisk (*) entries in the array `SYM` indicate that a traversal to the adjacent TriEdge record has to be performed. In these cases, the view component v of the result has to be adjusted to refer to its correct partner half-edge. This issue is delegated to the edge method `correct` which is implemented as follows: (t,w).correct(v) → (t, 2*w + (if odd(v) then 0 else 1)).

a)



b)

view	0	1	2	3	4	5	6	7	8	9	10	11
LEDGE	6	2	8	4	10	0	11	1	7	3	9	5
SYM	* 0_{SSG}		* 1_{SSG}		* 2_{SSG}		7	6	9	8	11	10
LEFT	④	②	⑤	③	⑥	①	⑦	④	⑦	⑤	⑦	⑥
ORIG	❶	❷	❷	❸	❸	❶	❶	❷	❷	❸	❸	❶

Fig. 8. a) a TriEdge record represents twelve molecular half-edges
b) four arrays to implement the basic operations

6 Evaluation

First, we investigate the space requirement for the TriEdge structure in comparison to directly storing molecular surfaces using the quad-edge structure, and in comparison to an implementation of the SSG by the quad-edge structure. We assume the vertices and the patches to be referenced by 4-byte pointers, and the records being adjusted to a multiple of four bytes. In a packed representation, the bits for the 'view' component can be held in a single 4-byte word. Let e denote the overall number of edges on the molecular surface. Table 4 shows the reduction of storage space for molecular surfaces by the factor 3.9 when considering topological information as well as references to geometrical information.

storage requirement		MolSurf via quad-edge	SSG via quad-edge	SSG via TriEdge
references per record to …	adjacent records	4	4	3
	patch corners	2	4	3
	surface patches	2	4	7
bytes per record (bpr)		**36**	**52**	**56**
number of molecular half-edges represented by a single record		2	8	12
number of records (#r)		**e**	**e / 4**	**e / 6**
total bytes (bpr · #r)		36 · e	13 · e	9.33 · e
factor of reduction		**1**	**2.8**	**3.9**

Tab. 4. Storage requirement for molecular surfaces by different techniques

When changing from MolSurf via quad-edge to SSG via quad-edge, the number of records is reduced by a factor of four, due to the mapping explained above. Figure 8a helps to illustrate the reduction of factor six when changing from MolSurf via quad-edge to SSG via TriEdge: a TriEdge record covers twelve (molecular) half-edges, whereas a quad-edge record only represents two half-edges of the molecular surface. Thus, the reduction factor for the number of records is $12/2 = 6$ while representing the same number of molecular (half-)edges.

We integrated the TriEdge structure into our object-oriented protein database system based on the C++ interface of the OODBMS ObjectStore. From our prototype, we obtained the following results of the runtime behavior of the TriEdge structure on an HP-9000/735 workstation under HP-UX 9.01: inserting molecular surfaces takes only a few seconds of elapsed time. The steps performed were reading the surface structure from a text file, creating the TriEdge records as well as the patch objects, and connecting the TriEdge records, all embraced by a transaction begin and commit (table 5).

From further experiments, we obtained the processing time for some selected euclidean neighborhood queries. The elapsed time is shown in table 6 for various radii. We determined the values as an average over 1,000 calls each, performed on a patch on the protein 2pab.a. From the first experiments, we could not recognize any difference in the

molecule	2pab.a	2pab.b	2ptc.e	2ptc.i	2sni.e	2sni.i
number of patches	3,710	3,776	5,606	1,990	6,220	2,116
insertion time (sec)	6.1	6.1	8.8	3.3	9.7	3.6

Tab. 5. Insertion time for molecular surfaces into the database

runtime between a breadth-first and a depth-first evaluation strategy. As expected, the runtime grows nearly linear with respect to the number of neighbors (cf. figure 9).

radius (Å)	number of neighbors	msec per query
1.0	3	0.1
2.0	7	0.6
3.0	16	2.0
4.0	21	4.2
5.0	71	7.4
6.0	130	14.2
7.0	177	19.3
8.0	231	24.0
9.0	280	31.2
10.0	355	36.3
11.0	452	51.1
20.0	1,607	164.0

Tab. 6. Runtime of neighborhood queries for a patch of the protein 2pab.a (average over 1,000)

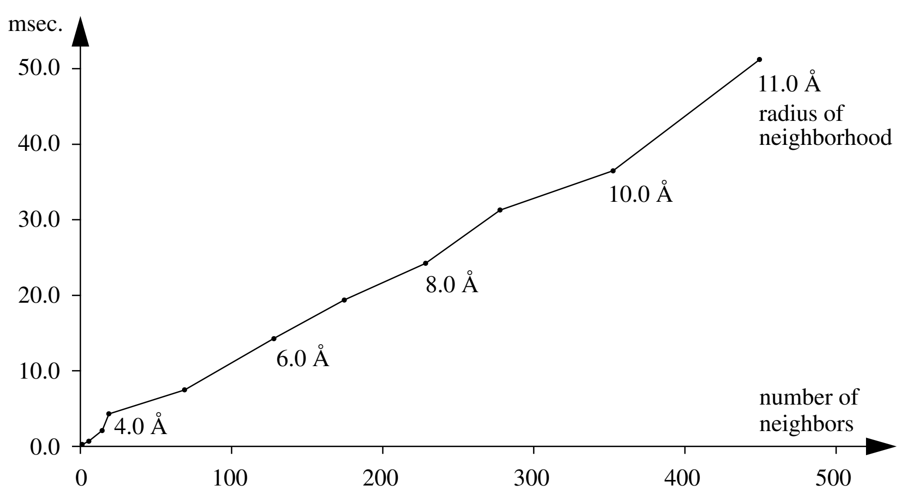


Fig. 9. Dependency of the runtime on the number of neighbors

7 Conclusion

In this paper, we presented the notion of molecular surfaces as fundamental 3D objects in database systems for molecular biology and computational chemistry. Along with the geometric and the physicochemical properties, in particular the topological structure of molecular surfaces has to be represented. As a basic operation, we defined the neighborhood query that is required for applications like local surface approximation and surface segmentation. The technical contribution of the paper is the introduction of the TriEdge data structure as an implementation of molecular surface graphs providing efficient support for neighborhood query processing. It leads to a considerably reduced space requirement compared to the well-known quad-edge structure. First experimental results show the query processing time to be in the range of milliseconds. The insertion time may be improved when integrating the molecular surface calculation program into the database system, thus avoiding text file input.

In our future work, we will integrate (1) circular edges (these are not bound by any corner point) that occur on molecular surfaces in some cases, e.g. when small atom chains stick out from the molecule, and (2) convex patches that are bound by more than one edge cycle which also may occur in some rare cases. After all, we will apply our representation and query processing technique to the docking problem.

A very challenging problem is the clustering of TriEdge records: how can query processing be improved by applying clustering techniques? In our prototype, we still rely on the default clustering of the database management system. A promising approach is managing the records by a spatial access method, since topological proximity implies geometric vicinity but not necessarily vice versa. Also the question about the adequate traversal strategy needs further investigation.

Acknowledgement

We thank our colleagues Martin Ester and Thomas Schmidt for engaging and fruitful discussions as well as the anonymous referees for their comments to improve the presentation of our concepts.

References

[Ald 94] Aldinger K., Ester M., Förstner G., Kriegel H.-P., Seidl T.: *'A Database System Supporting Protein-Protein-Docking: an Efficient and Robust Feature-Index',* Proc. 2nd GI Conference on Computer Science and Biology, Jena, Sept. 1994, pp. 41-52, in German.

[Ber 77] Bernstein F. C., Koetzle T. F., Williams G. J., Meyer E. F., Brice M. D., Rodgers J. R., Kennard O., Shimanovichi T., Tasumi M.: *'The Protein Data Bank: a Computer-based Archival File for Macromolecular Structures',* Journal of Molecular Biology, Vol. 112, 1977, pp. 535-542.

[BHKS 93] Brinkhoff T., Horn H., Kriegel H.-P., Schneider R.: *'A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems',* Proc. 3rd Int. Symp. on Large Spatial Databases (SSD '93), Singapore, 1993, Lecture Notes in Computer Science, Vol. 692, Springer, pp. 357-376.

[BKSS 94] Brinkhoff T., Kriegel H.-P., Schneider R., Seeger B.: *'Multi-Step Processing of Spatial Joins',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, MN, 1994, pp. 197-208.

[BMH 92] Badel A., Mornon J. P., Hazout S.: *'Searching for geometric molecular shape complementarity using bidimensional surface profiles',* Journal of Molecular Graphics, Vol. 10, 1992, pp. 205-211.

[Bri 93] Brisson E.: *'Representing Geometric Structures in d Dimensions: Topology and Order',* Discrete & Computational Geometry, Vol. 9, 1993, pp. 387-426.

[Con 83] Connolly M. L.: *'Solvent-Accessible Surfaces of Proteins and Nucleic Acids'*, Science, Vol. 221, 1983, pp. 709-713.

[Con 86] Connolly M. L.: *'Shape Complementarity at the Hemoglobin* $\alpha_1\beta_1$ *Subunit Interface',* Biopolymers, Vol. 25, 1986, pp. 1229-1247.

[DL 89] Dobkin D. P., Laszlo M. J.: *'Primitives for the Manipulation of Three-Dimensional Subdivisions',* Algorithmica, Vol. 4, 1989, pp. 3-32.

[DMP 93] De Floriani L., Marzano P., Puppo E.: *'Spatial queries and data models',* Proc. European Conference on Spatial Information Theory (COSIT '93), Lecture Notes in Computer Science, Vol. 716, Springer, 1993, pp. 113-138.

[EG 94] Erwig M., Güting R. H.: *'Explicit Graphs in a Functional Model for Spatial Databases',* IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 5, 1994, pp. 787-804.

[EKSX 95] Ester M., Kriegel H.-P., Seidl T., Xu X.: *'Shape-based Retrieval of Complementary 3D Surfaces from a Protein Database',* Proc. GI Conf. on Database Systems for Office Automation, Engineering, and Scientific Applications (BTW '95), Informatik aktuell, Springer, 1995, pp. 373-382, in German.

[EKX 95] Ester M., Kriegel H.-P., Xu X.: *'Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification',* Proc. 4th Int. Symposium on Large Spatial Databases (SSD '95), Portland, Maine, 1995.

[FNNW 93] Fischer D., Norel R., Nussinov R., Wolfson H. J.: *'3-D Docking of Protein Molecules',* Proc. 4th Annual Symposium on Combinatorial Pattern Matching (CPM '93), Padova, Italy, in: Lecture Notes in Computer Science, Vol. 684, Springer, 1993, pp. 20-34.

[GS 85] Guibas L., Stolfi J.: *'Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams',* ACM Trans. Graphics, Vol. 4, No. 2, 1985, pp. 74-123.

[HO 94] Halperin D., Overmars M. H.: *'Spheres, Molecules, and Hidden Surface Removal',* Proc. 10th ACM Symp. Computational Geometry, 1994, pp. 113-122.

[HT 94] Helmer-Citterich M., Tramontano A.: *'PUZZLE: A New Method for Automated Protein Docking Based on Surface Shape Complementarity',* Journal of Molecular Biology, Vol. 235, 1994, pp. 1021-1031.

[Jia 92] Jiang B.: *'I/O-Efficiency of Shortest Path Alogrithms: An Analysis',* Proc. IEEE Int. Conf. Data Engineering, 1992, pp. 12-19.

[Jia 94] Jiang B.: *'Processing non-relational database queries—a comparison of various proposals',* Informatik Spektrum, Springer, Vol. 17, 1994, pp. 373-383, in German.

[Koe 90] Koenderink J. J.: *'Solid Shape',* MIT Press, Cambridge, MA, 1990.

[KSB 93] Kriegel H.-P., Schneider R., Brinkhoff T.: *'Potentials for Improving Query Processing in Spatial Database Systems',* invited talk, Proc. 9èmes Journées Bases de Données Avancées (9th Conference on Advanced Databases), Toulouse, France, 1993.

[OHMS 92] Orenstein J., Haradhvala S., Margulies B., Sakahara D.: *'Query Processing in the ObjectStore Database System',* Proc. ACM SIGMOD 1992, pp. 403-412.

[PDB 95] Protein Data Bank: *'Quarterly Newsletter No. 71 (Jan 1995)',* Brookhaven National Laboratory, Upton, NY, 1995.

[PS 85] Preparata F. P., Shamos M. I.: *'Computational Geometry—An Introduction',* Springer, 1985.

[Ric 77] Richards F. M.: *'Areas, Volumes, Packing, and Protein Structure',* Annual Reviews in Biophysics and Bioengineering, Vol. 6, 1977, pp. 151-176.

[Sch 94] Schmidt T.: *'Calculation of Protein Surfaces using Spatial Access Methods',* Master thesis, Institute for Computer Science, Technical University of Munich, 1994, in German.

[SKSH 89] Schneider R., Kriegel H.-P., Seeger B., Heep S.: *'Geometry-based Similarity Retrieval of Rotational Parts',* Proc. Int. Conf. on Data and Knowledge Systems for Manufacturing and Engineering, Gaithersburg, ML, 1989, pp. 150-160.

[VBW 94] Varshney A., Brooks F. P., Wright W. V.: *'Computing Smooth Molecular Surfaces',* IEEE Computer Graphics & Applications, Vol. 14, No. 5, 1994, pp. 19-25.