Implicit Representation of Molecular Surfaces

Julius Parulek *
Department of Informatics, University of Bergen

Ivan Viola†
Department of Informatics, University of Bergen.

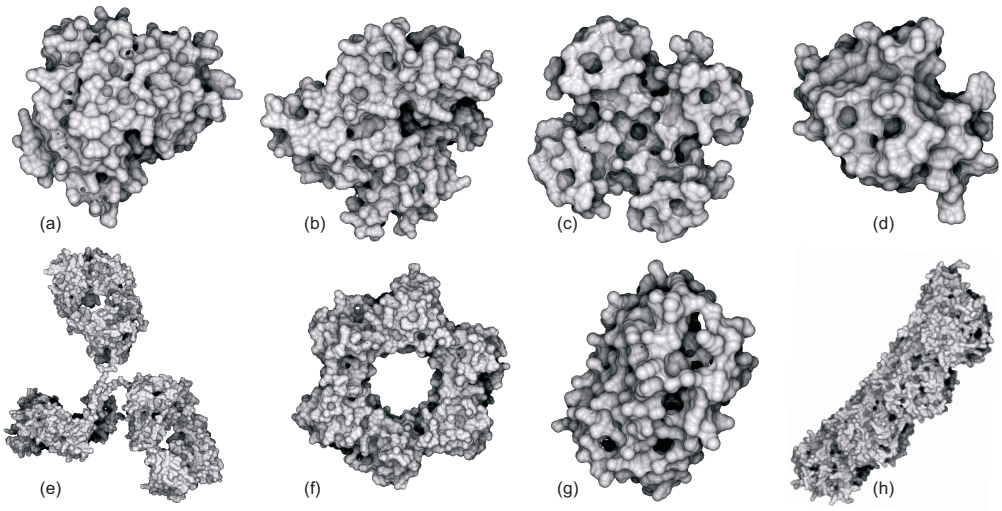

Figure 1: Ray-casting based visualization of eight proteins represented by the solvent excluded surfaces (*SES*). Surfaces are defined by implicit functions computed locally and without performing any precomputation steps. The proteins contain 3023 (a), 2872 (b), 2820 (c), 3346 (d), 12530 (e), 12555 (f), 1718 (g) and 14744 (h). Small discontinuities are caused by the precision parameter that accepts a close vicinity of the 0 iso-level of the implicit function.

Abstract

Molecular surfaces are an established tool to analyze and to study the evolution and interaction of molecules. One of the most advanced representations of molecular surfaces is called the solvent excluded surface. We present a novel and a simple method for representing the solvent excluded surfaces (*SES*). Our method requires no precomputation and therefore allows us to vary *SES* parameters outright. We utilize the theory of implicit surfaces and their *CSG* operations to compose the implicit function representing the molecular surface locally. This function returns a minimal distance to the *SES* representation. Additionally, negative values of the implicit function determine that the point lies outside *SES* whereas positive ones that the point lies inside. We describe how to build this implicit function composed of three types of patches constituting the *SES* representation. Finally, we propose a method to visualize the iso-surface of the implicit function by means of ray-casting and the set of rendering parameters affecting the overall performance.

Keywords: Visualization of Molecular Surfaces, Geometrical Modeling, Implicit Surfaces, Bioinformatics Visualization

Index Terms: J.3 [Computer Applications]: Life and Medical Sciences—Biology and Genetics I.3.4 [COMPUTER GRAPHICS]: Computational Geometry and Object Modeling—Boundary representations, Curve, surface, solid, and object representations

*e-mail: Julius.Parulek@uib.no
†e-mail: ivan.viola@uib.no

IEEE Pacific Visualisation Symposium 2012
28 February - 2 March, Songdo, Korea
978-1-4673-0864-9/12/$31.00 ©2012 IEEE

1 Introduction

From the biomolecular point of view, the structure of molecules is given by three-dimensional organization of atoms. While chemical structure of the atoms is often quite stable, the overall shape, generated by force fields of atoms, can vary rapidly. Therefore, to understand the shape in its full complexity, virtual computer models of molecules are instrumental.

A specific type of molecules are proteins that actively participate in various cell processes. Essentially, they are long-chained macromolecules that consist of standard amino acids. These amino acids can contain between 10 and 25 atoms each, where typically the molecular chain can contain up to a few thousand of amino acids. The set of atoms that is common for every amino acid is called the backbone. The set of atoms connected to the backbone is called the side chain and is unique for every amino acid.

Many proteins are enzymes that need to be triggered by other biomolecules called ligands. Such proteins can, for instance, transport ligands to different parts of the cell or even between the cells. In the latter case, these form so-called channels. In order to discover which ligands can bind to a given protein, one needs to communicate the protein shape accordingly. A common approach to represent protein surface to discover the ligand binding site, is to utilize a solvent molecule, e.g. the water molecule. The reason is that the ligand can practically access all the places that are reachable by the solvent itself. Therefore, one of the possible structural representations of the molecules is defined via the solvent which can reveal possible binding sites.

In molecular visualization, many different methods of representing molecules have been proposed [31]. One can use different structural representations like space fill, balls-and-sticks, licorice, backbone, and ribbon. These can be shown through distinct illustrative

techniques, e.g. cartoon representation, or by means of emphasizing the depth perception like halos and ambient occlusion [28].

One of the most employed advanced representations of molecular surfaces is called the solvent excluded surface (*SES*), which was first proposed by Richards [21]. So far, *SES* became a very well established representation of the iso-surface approximating the potential field of atoms while taking into consideration the solvent molecule. The main utilization of *SES* representation is studying the molecular dynamics in order to discover ligand binding sites and their evolution over the molecular dynamic simulation.

It is important to mention that one can apply more complex, physically-plausible models, which include electrostatic potential fields (EPF). The major advantage of using SES representation with respect to EPF-based representation is its relatively fast computation mainly when dealing with large molecular dynamic simulations (MDS) containing several thousands of animation frames. The EPF models are computationally very extensive and are usually rasterized via partial differential equations on a regular grid. Although, when such representation is precomputed, one can combine SES and EPF representations into a merged visualization.

The *SES* representation is obtained by rolling a solvent molecule (approximated by a sphere of radius R) on the so-called solvent-accessible surface (SAS). The SAS is represented as the boundary of extended van der Waals spheres by radius R. The resulting *SES* then decomposes into three types of surface patches: convex spherical patches, concave spherical patches (spherical triangles) and toroidal patches (Fig. 2).

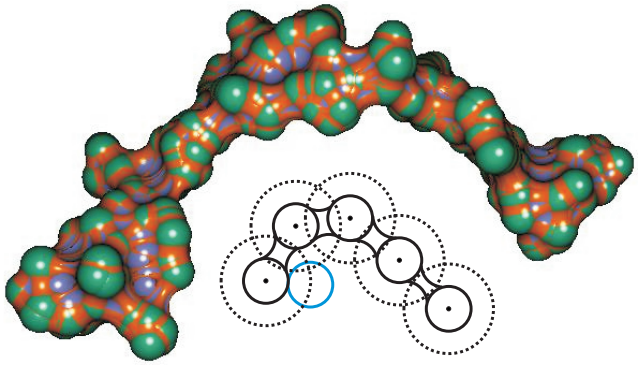

Figure 2: An example of a molecular surface. The solvent excluded surface (union of full circles and their toroidal interconnections) is formed by rolling a ball (blue circle) on van der Waals spheres while the center lies on the boundary of solvent accessible surface (union of dashed circles). The *SES* representation depicts a single residual sequence of proteinase 3 (Pr3) peptide containing 360 atoms. The *SES* surface can be be described by three types of patches, the convex spherical one (green), the concave spherical one (blue) and the toroidal one (red).

The main drawback when using *SES* representation in visual exploration is the lack of interactivity, since the geometric *SES* representation requires usually substantial precomputation [24, 30]. Here, to delimit the atom selection, participating in the *SES* representation, or just to adjust the solvent radius R, requires recreating the entire surface. The essential parameter settings interaction during the visual exploration of a macromolecule can not be performed in real-time. Moreover, to represent the surface for time-varying protein simulations the heavy load is put to the precomputation step even further. Essentially, one needs to to preprocess several time-steps in advance. Therefore, biologists often prefer to limit the binding-site analysis to the particular key frames of the simulation. With the recent advance of computational hardware capabilities, larger and larger dynamic protein sequences are being produced, which clearly demands more flexible *SES* representation that would allow for an instant surface representation without having done any precomputational steps.

In this paper, we propose a novel approach to represent *SES* surface without precomputation. Our new representation is based on theory of implicit surfaces, namely its generalization called functional representation [18]. We propose a rendering approach to achieve interactive visualization by introducing several rendering-quality parameters affecting the overall performance.

The instantaneous *SES* representation allows us to provide interactivity necessary for protein analysis. Particularly, we aim at providing biologists with the possibility of performing changes of solvent radius R and seeking at any time intervals of the simulation.

2 Related Work

We relate our work to three areas. Firstly, we introduce techniques of molecular visualization. Secondly, we describe implicit modeling techniques. Finally, we present real-time rendering techniques of implicits.

2.1 Visualization and Representation of Molecular Surface

There are several types of molecular surface representations introduced in the literature. One of the basic representations is when the protein atoms are depicted as spheres, where the radius corresponds to the van der Waals force (vdW surface) [13]. The extension of this surface by a solvent radius is called the solvent accessible surface (SAS). The most widely used representation is denoted as solvent excluded surface (*SES*) [21, 6]. In 1992, Edelsbrunner and Mücke introduced α-shape representation and in 2007, Ryu et al. extended it to the β-shape. In 1999, Edelsbrunner [4] also proposed a new representation of molecules called the skin-surface.

In our work, we focus on representing and visualizing the *SES* representation, because it is a standard technique and was requested by our biology cooperation partners for binding-site exploration.

There has been a lot of effort put into generation of *SES* in the literature. In 1983, Connolly [3] proposed an analytical description of the *SES* representation. More than ten years later, Sanner et al. [24] presented the reduced surface algorithm for construction of *SES* representation. In the same year, Totrov and Abygyan introduced the contour-buildup algorithm [30] to form the *SES* representation. Recently, Lindow et al. [14] proposed a speeding up and parallelization of contour-buildup algorithm. The visualization is then performed by a ray-casting method. In 2009, Krone et al. [11] introduced the utilization of the reduced surface method for direct ray-casting of molecular surfaces. Here, all techniques exploit precomputation steps necessary to render the final surface.

Triangularization methods of *SES* are popular as well, with the most recent ones being proposed in 2009 by Ryu et al. [22]. Additionally, *SES* can be decomposed into a set of quadrics, which can be efficiently rendered on the GPU (Section 2.3). In 2011, Krone et al. [12] introduced an approximation of the *SES* representation via quadratic polynomial kernels. This work was aimed at the tracking of protein cavities. Here, nevertheless, the resulting iso-surface only imitates the exact *SES* representation.

In our work, we utilize Protein Data Bank (PDB) file format, which stores the protein information (e.g., atom types, residual sequences). The trajectories of the atoms are stored in the *DCD* file format that is used as a standard in the Visual Molecular Dynamics (VMD) tool [9]. The file can contain tens of thousands of protein trajectories. The other popular molecular viewers are for instance PyMOL [25], MetaMol [2] and QuteMol [28] to name a few. So far, none of these tools can visualize dynamic trajectories and/or would allow for interactive visual protein explorations.

To the best of our knowledge, there are no approaches that would allow for direct visualization of *SES* representation without any precomputation. We cross this gap by introducing a new approach that computes the *SES* instantly on the fly during ray-casting. Contrary to the previous approaches, we employ a theory of implicit surfaces, where the geometry of a single atom is described by its distance based implicit function.

2.2 Implicit Modeling

Implicit surfaces (implicits) provide a way to easily model complex dynamically changing geometric objects. Moreover, they naturally enable the modeling of smooth, sponge like objects in a convenient way.

The set of techniques, known today as implicit modeling, was used for the first time by Blinn [1]. Pasko et al. generalized the representation of implicits, by combination of the different forms of implicit models [18], and denoted it as function representation (Frep). The inequality (1) describes an implicit solid (object):

$$f(\mathbf{p}) \geq 0, \tag{1}$$

where $\mathbf{p} = (x_1, x_2, x_3) \in E^3$. Function f is called an implicit surface function (implicit function), which classifies the space into two half-spaces $f(\mathbf{p}) > 0$ and $f(\mathbf{p}) < 0$.

Complex objects can be created from simple ones via Constructive Solid Geometry (CSG) operations. The basic set-theoretic operations can be defined using the *min* and *max* operators:

$$\begin{array}{rcccl} union(f_1, f_2) & = & f_1 \cup f_2 & = & max(f_1, f_2) \\ intersection(f_1, f_2) & = & f_1 \cap f_2 & = & min(f_1, f_2) \\ subtraction(f_1, f_2) & = & f_1 \backslash f_2 & = & min(f_1, -f_2). \end{array} \tag{2}$$

Analytical expressions that approximate these operators were proposed by Ricci [20]. The other analytical definitions of the set-theoretic operations are known as R-functions [18]. Nevertheless, the analytical versions do not preserve the distance characteristics as well as *min*/*max* operators do [5]. Therefore, in our work we utilize the basic *min* and *max* operators (2) since they are fast to compute and preserve distance characteristics of the implicit function better. However, they are discontinuous where $f_1 = f_2$. This has been studied by Fayolle [5]; which we took as an inspiration for our solution.

2.3 Real-time rendering of implicit objects

In order to visualize models based on implicits, one can convert them to a mesh representation prior to rendering them as a set of patch primitives [16]. However, when dealing with complex models and shapes, such as for instance the molecular surfaces, one would need to generate millions of triangles to perform a fully detailed surface representation.

Therefore, in recent years, authors turn to direct visualization techniques, which is being represented by ray-casting methods. Since implicts encompass different forms of geometrical models, the actual ray-casting method is proposed according to the type of implicits we are dealing with.

For example, the ray-casting of algebraic surfaces is covered extensively by computer graphics literature, which goes back several decades. Hanrahan introduced ray-casting of algebraic implicit models up to the fourth order [7]. Later work addressed ray-casting of large number of quadrics on GPU aimed at molecular visualization as well [29, 19, 15]. Later on, Sigg et al. [26] introduced very fast GPU rendering of spheres, ellipsoids and cylinders focusing mainly at molecular rendering. Nevertheless, these papers assume that the input set of quadrics is already formed, which is not the case with our instant implicit representation.

In 1992, Hart introduced a more robust approach for ray-casting of distance based implicit surfaces called sphere tracing [8]. Since in our work, we are dealing with distance based implicit functions also, we adopted Hart's technique, because it is easy to implement and performs very well on distance based surfaces.

The recent work addresses ray-casting of general implicit surfaces on GPU using interval arithmetics [10] and via so-called adaptive marching point method [27]. However, their methods assume that the function is defined globally, whereas our function is computed locally only.

3 Method Overview

Our algorithm computes the implicit function representing *SES* on the fly during ray-casting. This function returns the minimal distance from a sample point $\mathbf{p}$ (given on a ray) to the surface. To compute the function, the following procedures are performed.

The k closest atoms to the point $\mathbf{p}$ are retrieved in the ascending order. According to the number of atoms k the function f_{SES} is generated. Here, all possible pairs $\binom{k}{2}$ of atoms are tested, whether there might be an intersection between their solvent extended iso-surfaces. If the test is positive, the intersection point that is the closest one to $\mathbf{p}$ is estimated and stored. Then all possible triplets $\binom{k}{3}$ are checked according to the retrieved pair-wise intersection points and neighboring atoms. Again, if the test is positive, the intersection point of the triplet is estimated and stored. To evaluate f_{SES} all the stored intersection points then generate the solvent sphere function.

Afterwards, using computed f_{SES}, the ray is advanced to a next point, by the function value. When the value of f_{SES} is close to 0 (to the iso-surface), a depth and a normal at the point $\mathbf{p}$ are stored. The implementation of our ray-casting method is performed and parallelized using CUDA, where the ray-traversal and the function evaluation is done per every image pixel. Finally, according to the stored depth values and the computed normals, we perform the shading computation and enhance the depth perception by means of screen space ambient occlusion [17].

4 SES Representation

Let us first introduce the geometrical basics allowing us to represent the surface using implicit modeling techniques. The goal is to define an implicit function describing the surface in a certain $3D$ neighborhood of a given point $\mathbf{p}$, in order to evaluate $f_{SES}(\mathbf{p})$. The implicit function f_{SES} fulfils the following properties:

- for the given $3D$ point $\mathbf{p}$ the function $f_{SES}(\mathbf{p})$ will return the minimal distance to *SES* representation.
- $f_{SES}(\mathbf{p}) > 0$ for $\mathbf{p}$ inside the object, $f_{SES}(\mathbf{p}) < 0$ for $\mathbf{p}$ outside the object and $f_{SES}(\mathbf{p}) = 0$ means the $\mathbf{p}$ lies on the boundary (iso-surface).
- function f_{SES} is C^1 continuous in a neighborhood of the surface, i.e. gradient ∇f_{SES} is continuous in this neighborhood.

Let us assume that the set of atoms is defined as $C = \{(\mathbf{c}_1, r_1), \ldots, (\mathbf{c}_n, r_n)\}$ with the solvent radius defined by R. We define a set of implicit functions (implicits) defined as $F = \{f_1, f_2, \ldots, f_n\}$, where each $f_i(\mathbf{p}) = r_i - ||\mathbf{p} - \mathbf{c}_i||$ represents an atom $\mathbf{c}_i$ with the corresponding van der Waals radius r_i. Note that function f delimits the space into the interior and the boundary ($f \geq 0$) and the exterior ($f < 0$) half-space. We also define an extended set of implicits, where sphere atoms are enlarged by the solvent radius R, as $G = \{g_1 = f_1 + R, g_2 = f_2 + R, \ldots, g_n = f_n + R\}$. Since f_i and g_i are distance functions the gradient of both is defined via

$$\nabla f_i(\mathbf{p}) = \nabla g_i(\mathbf{p}) = \frac{\mathbf{c}_i - \mathbf{p}}{||\mathbf{c}_i - \mathbf{p}||}.$$

Additionally, we define a set of atom centers $S_\mathbf{p}$ that are closer to a point $\mathbf{p}$ than $2R$, $S_\mathbf{p} = \{\mathbf{c}_i | f_i(\mathbf{p}) \geq -2R \wedge \mathbf{c}_i \in C\}$. The set $S_\mathbf{p} \subseteq C$ delimits the point set C to atoms that might participate in

generating the surface (Fig. 3a), according to the point $\mathbf{p}$ that is a parameter of the implicit function. The expression of set S is derived directly from the detection of two incident atoms which holds if $||\mathbf{c}_i - \mathbf{c}_j|| < 2R + r_i + r_j$ (Fig. 3b).

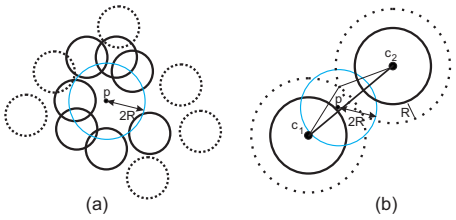

Figure 3: Forming the point set $S_\mathbf{p}$. a) Out of all the points, the set $S_\mathbf{p}$ contains those which belong to the area of influence of the point $\mathbf{p}$ (full circles). b) The point $\mathbf{p}$ lies above the R distance from the atom $\mathbf{c}_2$, but it might still participate in forming the final iso-surface.

After forming set $S_\mathbf{p}$ the resulting function can be expressed as follows:

$$f_{SES}(\mathbf{p}) = f_{SAS}(\mathbf{p}) - \left(\bigcup_{\mathbf{x} \in f_{SAS}^{-1}(0)} (R - ||\mathbf{x} - \mathbf{p}||) \right), \tag{3}$$

where f_{SAS} is defined as follows:

$$f_{SAS}(\mathbf{p}) = \bigcup_{i=1}^{|S|} g_i(\mathbf{p}), \tag{4}$$

where the $\bigcup_{i=1}^{|S|} g_i = max\{g_1, \ldots, g_{|S|}\}$.

The term $f_{SAS}^{-1}(0)$ represents all iso-surface points of the function f_{SAS}, on which the union of the solvent spheres is performed. This union is subsequently subtracted from the f_{SAS}. The formula (3), however; does not have a closed form solution. Therefore, our solution, reproducing Eq. 3, is based on building, in a piece-wise fashion, the *SES* implicit function according to a given point $\mathbf{p}$ and the size of the set S, $|S| = k$. We denote the resulting implicit function as $f_{SES}(\mathbf{p}) = f_S(\mathbf{p})$.

In the following sections, we describe the construction of f_S according to $|S|$. We introduce four essential cases, i.e. when S contains no atom, one atom, two atoms, and three or more atoms.

The first two cases can be solved easily. When there is no atom included in $S_\mathbf{p}$, the implicit function is defined as

$$f_{SES}(\mathbf{p}) = \bigcup_{i=1}^{n} f_i(\mathbf{p}). \tag{5}$$

In practice, since there is no atom in $2R$ neighborhood of the point $\mathbf{p}$, we can set the $f_{SES}(\mathbf{p}) = -2R$, in order not to evaluate all the points. In a case when there is only a single atom closer to $\mathbf{p}$ than $2R$, $S = \{\mathbf{c}_1\}$, the resulting $f_S(\mathbf{p})$ is defined solely by the atom $\mathbf{c}_1$, $f_S(\mathbf{p}) = f_1(\mathbf{p})$.

4.1 Toroidal implicit function

In a case when there are two atoms closer to $\mathbf{p}$ than $2R$, $|S| = 2$, the resulting $f_S(\mathbf{p})$ describes a toroidal patch that blends smoothly with both atom spheres. Let us assume that $S_\mathbf{p} = \{\mathbf{c}_1, \mathbf{c}_2\}$. We firstly detect whether point $\mathbf{p}$ lies in the toroidal section of both extended spheres, g_1 and g_2 (Fig. 5a, the blue triangle). The toroidal section clips the toroidal iso-surface within the possible extent.

In order to assess whether point $\mathbf{p}$ lies inside this area, we firstly generate two points $\mathbf{p}_1$ and $\mathbf{p}_2$ that are projections of the point $\mathbf{p}$ to both iso-surfaces of g_1 and g_2, i.e. $\mathbf{p}_1 = \mathbf{p} - g_1(\mathbf{p})\nabla g_1(\mathbf{p})$ and $\mathbf{p}_2 = \mathbf{p} - g_2(\mathbf{p})\nabla g_2(\mathbf{p})$ (Fig. 5b). Now we define a predicate

$$B_{1-2} \equiv g_1(\mathbf{p}_2) \geq 0 \wedge g_2(\mathbf{p}_1) \geq 0, \tag{6}$$

which represents whether point $\mathbf{p}$ lies within the the toriodal section.

Furthermore we define a distance based implicit function describing the surface inside the toroidal section. In order to do so, we exploit the fact that the toroidal iso-surface can be defined as minimal distance of $\mathbf{p}$ to the points belonging to the intersection circle of g_1 and g_2, $I = \{\mathbf{x} | g_1(\mathbf{x}) = g_2(\mathbf{x}) = 0\}$. However, computation of all the points would be very impractical. Therefore, we compute only the closest point $\mathbf{x}_{1-2} \in I$ to $\mathbf{p}$ directly

$$\mathbf{x}_{1-2} = arg\,min_{\mathbf{x} \in I} |\mathbf{x} - \mathbf{p}|. \tag{7}$$

Once we have this point $\mathbf{x}_{1-2}$, we define the resulting implicit function $f_S(\mathbf{p}) = f_{1-2}$, connecting $\mathbf{c}_1$ and $\mathbf{c}_2$, as follows:

$$f_{1-2}(\mathbf{p}) = \begin{cases} R - ||\mathbf{p} - \mathbf{x}_{1-2}|| & if\ B_{1-2} \\ f_1(\mathbf{p}) \cup f_2(\mathbf{p}) & otherwise, \end{cases} \tag{8}$$

where $f_1(\mathbf{p}) \cup f_2(\mathbf{p})$ is required to merge smoothly the toroidal patch with the both spheres. Figure 4 demonstrates the resulting implicit function iso-lines of the analogous scenario in 2D. Note that the resulting function correctly computes distance values up to the extent of R from the *SES* iso-surface.

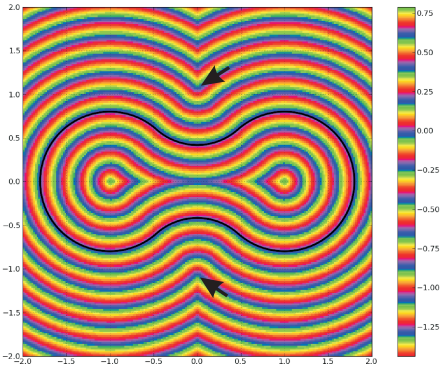

Figure 4: A 2D version of the toroidal implicit function with rendered iso-levels. The function returns correct distance values from the surface up to R distance from the surface (arrows).

To evaluate the intersection point $\mathbf{x}_{1-2}$ of g_1 and g_2 we exploit Newton's iterative formula. The natural solution would be to utilize an analytical approach to solve the intersection of two spheres; nevertheless we aim at providing a more robust approach that could possibly handle other types of atom representation than spheres. For example, in a case when two atoms are very close to each other (overlapping) they can be approximated by an implicit tube, which can occur quite frequently. Additionally we estimate the closest intersection point only, which is a prerequisite for Eq. 8. Therefore, we approximate both extended functions g_1 and g_2 at an unknown point $\mathbf{x}$, where $g_1(\mathbf{x}) = g_2(\mathbf{x}) = 0$, by means of their first order Taylor expansion at points $g_1(\mathbf{p})$ and $g_2(\mathbf{p})$:

$$\begin{aligned} 0 = g_1(\mathbf{x}) &\approx g_1(\mathbf{p}) - (\mathbf{x} - \mathbf{p}) \cdot \nabla g_1(\mathbf{p}) \\ 0 = g_2(\mathbf{x}) &\approx g_2(\mathbf{p}) - (\mathbf{x} - \mathbf{p}) \cdot \nabla g_2(\mathbf{p}), \end{aligned} \tag{9}$$

where $(\mathbf{x} - \mathbf{p})$ expresses the vector leading to the desired point $\mathbf{x}$. Additionally, it is required that the vector $(\mathbf{x} - \mathbf{p})$ lies in a plane

perpendicular to both gradients $\nabla g_1(\mathbf{p})$ and $\nabla g_2(\mathbf{p})$ (Fig. 5c):

$$(\nabla g_1(\mathbf{p}) \times \nabla g_2(\mathbf{p})) \cdot (\mathbf{x} - \mathbf{p}) = 0. \tag{10}$$

Using Eqs (6,9,10), we obtain a system of three linear equations where we would like to express an unknown point $\mathbf{x}$ from:

$$\begin{bmatrix} g_1(\mathbf{p}) \\ g_2(\mathbf{p}) \\ 0 \end{bmatrix} = (\mathbf{x} - \mathbf{p}) \cdot \begin{bmatrix} \nabla g_1(\mathbf{p}) \\ \nabla g_2(\mathbf{p}) \\ \nabla g_1(\mathbf{p}) \times \nabla g_2(\mathbf{p}) \end{bmatrix} \tag{11}$$

By denoting the matrix of gradients as $M(\mathbf{p})$ for the point $\mathbf{p}$, and the left side vector of the equation by $v(\mathbf{p})^T$, the solution can be obtained by

$$x = v(\mathbf{p})^T \cdot M(\mathbf{p})^{-1} + \mathbf{p} \tag{12}$$

Please note that when $B_{1-2} \equiv 1$, the point $\mathbf{p}$ lies in the close vicinity of point $\mathbf{x}$. However, it is still an approximation, and it is necessary to iterate through system (11) numerous times. Eq. 12 changes then to:

$$x_{i+1} = v(\mathbf{x}_i)^T \cdot M(\mathbf{x}_i)^{-1} + \mathbf{x}_i, \tag{13}$$

where $\mathbf{x}_0 = \mathbf{p}$ is the initial guess. The formula (13) has been already utilized to compute the intersection of two implicit functions in [5], which was aimed at generating distance approximation of boolean operations on functional represented objects.

The system (13) stops when the point $\mathbf{x}$ lies in close vicinity (ε_N) of both iso-surfaces, $|g_1(\mathbf{x})| \leq \varepsilon_N \wedge |g_2(\mathbf{x})| \leq \varepsilon_N$. Here, we can delimit the number of the maximal number of steps, since the point $\mathbf{p}$ lies close to $\mathbf{x}$. For instance we evaluated an average number of steps required for $\mathbf{x}_{1-2}$ evaluation between two spheres, for $\varepsilon_N = 0.0001$, to $3-7$ steps.

In a case when a self-intersection occurs i.e the condition described by Ryu et al. [23] (Fig. 5e), we benefit from the fact that point x_{1-2} always lies closest to point $\mathbf{p}$ (7). Therefore, self-intersection is tackled implicitly, which is an another reason for preferring Newton's iterative method over the analytical root finding.

In Figure 5d, we showcase the final toroidal iso-surface for two atoms, where we also demonstrate the self-intersection scenario (Fig. 5f).

In the following, the intersection point $\mathbf{x}$ retrieved via (11) between g_i and g_j is denoted as $\mathbf{x}_{i-j}$.

4.2 Spherical triangle implicit function

In a case when there are three or more points closer to $\mathbf{p}$ than $2R$, $|S| \geq 3$, the resulting $f_S(\mathbf{p})$ describes a spherical triangle. Here, the resulting function create spherical triangle patch that blends smoothly with three toroidal patches.

For simplicity, let us assume that $S_\mathbf{p}$ contains only 3 points, $S_\mathbf{p} = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\}$. In order to discover whether point $\mathbf{p}$ lies in the spherical triangle section (clipping the spherical triangle), we will exploit the intersection points computed in the previous case.

Firstly, we extend the predicates B_{1-2} (6) to B'_{1-2} to encompass not only toroidal section but also an additional region that might be encompassed by the area of influence of the third point $\mathbf{c}_3$ (Fig. 6a). Additionally, it must be fulfilled that intersection point $\mathbf{x}_{1-2}$, obtained via (11), lies within the opposite extended implicit function g_3 (Fig. 6b). Therefore, the new predicate B'_{i-j} is defined as follows:

$$B'_{1-2} \equiv g_1(\mathbf{p}) \geq -R \wedge g_2(\mathbf{p}) \geq -R \wedge g_3(\mathbf{x}_{1-2}), \tag{14}$$

where $g_3(\mathbf{x}_{1-2})$ is the extended function of the third point evaluated at the intersection point of g_i and g_j.

Now, in order to determine whether point $\mathbf{p}$ lies inside the spherical triangle section, we defined predicate B_{1-2-3} as follows:

$$B_{1-2-3} \equiv B'_{1-2} \wedge B'_{1-3} \wedge B'_{2-3}. \tag{15}$$

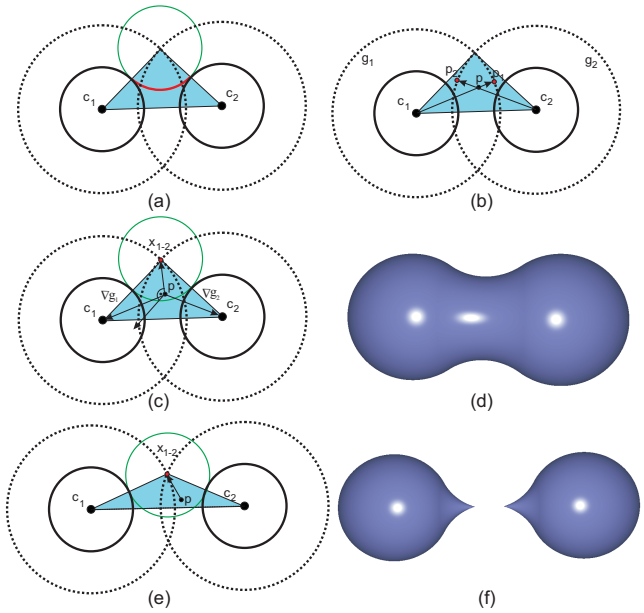

Figure 5: Forming the toroidal patch for two atoms. a) The toroidal patch (red) lies only in the area marked by the blue triangle. b) In order to determine that the given point $\mathbf{p}$ belongs to this area, we perform (1) projection of B_p to iso-surfaces $g_1(\mathbf{p}_1) = 0$ and $g_2(\mathbf{p}_2) = 0$ and (2) evaluation of both functions for opposite points. c) In a case that the point $\mathbf{p}$ lies inside the toroidal section, we retrieve the intersection point x_{1-2} of g_1 and g_2 using the fact that the vector $(\mathbf{x}_{1-2} - \mathbf{p})$ lies in the plane perpendicular to both gradients $\nabla g_1(\mathbf{p})$ and $\nabla g_2(\mathbf{p})$. d) The visualization of the toroidal patch. The atoms are close enough to get the continuous iso-surface. e) Self-intersection can occur when the solvent is thicker than the height of the actual toroidal section. f) The visualization of the self-intersected toroidal patch.

Visual comparison of forming a concave spherical patch using original predicates B and their updated version B' is shown in Figure 6c and 6d.

Furthermore, in order to specify the spherical implicit function defined by $\mathbf{c}_1$,$\mathbf{c}_2$ and $\mathbf{c}_3$, we locate an intersection point fulfilling $g_1(\mathbf{x}) = g_2(\mathbf{x}) = g_3(\mathbf{x}) = 0$. Accordingly, we define the resulting implicit function $f_S(\mathbf{p}) = f_{1-2-3}$, connecting $\mathbf{c}_1$,$\mathbf{c}_2$ and $\mathbf{c}_3$, as follows:

$$f_{1-2-3}(\mathbf{p}) = \begin{cases} R - ||\mathbf{p} - \mathbf{x}_{1-2-3}|| & if\ B_{1-2-3} \\ f_{1-2}(\mathbf{p}) \cup f_{1-3}(\mathbf{p}) \cup f_{2-3}(\mathbf{p}) & otherwise, \end{cases} \tag{16}$$

where the second branch is required to smoothly connect the spherical triangle with the all three toroidal patches.

In order to locate the point $\mathbf{x}$, we utilize Newton's iterative method of root finding of all three functions, where the system (11) changes to the following:

$$\begin{bmatrix} g_1(\mathbf{p}) \\ g_2(\mathbf{p}) \\ g_3(\mathbf{p}) \end{bmatrix} = (\mathbf{x} - \mathbf{p}) \cdot \begin{bmatrix} \nabla g_1(\mathbf{p}) \\ \nabla g_2(\mathbf{p}) \\ \nabla g_3(\mathbf{p}). \end{bmatrix} \tag{17}$$

In order to solve (17) we utilize again the iterative formula presented in (13). The system (17) stops when the point $\mathbf{x}$ lies in close vicinity (ε_N) of all three iso-surfaces, $|g_1(\mathbf{x})| \leq \varepsilon_N \wedge |g_2(\mathbf{x})| \leq \varepsilon_N \wedge |g_2(\mathbf{x})| \leq \varepsilon_N$. The definition of $f_S(\mathbf{p})$ for all the points in $S_\mathbf{p}$ is straightforward:

$$f_S(\mathbf{p}) = \bigcup_{(i,j,k) \in \binom{|S|}{3}} f_{i-j-k}(\mathbf{p}). \tag{18}$$

Similarly to the previous case, when a self-intersection occurs; we utilize the fact that point $\mathbf{x}_{1-2-3}$ lies closest to point $\mathbf{p}$ and the self-intersection is tackled implicitly (Fig. 6e). We showcased the example composed of 12 atoms in Figure 6f.

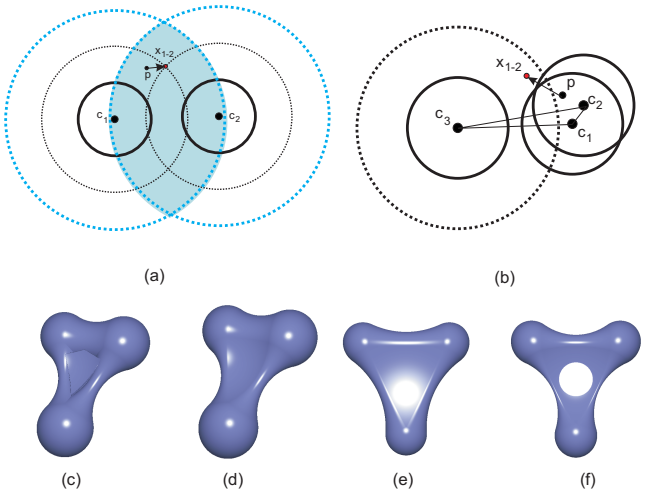

Figure 6: The formation of the spherical triangle patch. a) Evaluation of the predicate B'_{1-2}. The point x_{1-2}, representing the intersection of g_1 and g_2, must lie inside the area of influence of g_3. Therefore the area (the blue ellipse) for $\mathbf{x}_{1-2}$ estimation is extended to all the points that fulfil $g_1(\mathbf{p}) \geq -R \wedge g_2(\mathbf{p}) \geq -R$. b) Once the intersection point x_{1-2} is computed, it is evaluated against the third extended function $g_3(x_{1-2}) \geq 0$ (Eq. 14). c) When using the original predicates artifacts can appear caused by the fact that the intersection point $\mathbf{x}_{i-j}$ lies too far away from the point $\mathbf{c}_k$. d) The example of the corrected predicate B', where the cracks disappeared. e) The self-intersection issue is solved implicitly using the property of the point $\mathbf{x}_{1-2-3}$. f) An example of the SES, defined by (18), composed of 12 atoms.

5 Implementation

Our implementation of the visualization pipeline is performed using modern GP-GPU techniques, namely CUDA and GLSL. On CUDA, we perform the ray-casting of the whole scene and store the iso-surface points and their normals. Afterwards, using GLSL shaders we perform an image based enhancement by means of the screen space ambient occlusion [17]. The parallelization is done on the ray basis, where a single ray is generated for every pixel.

5.1 Data Structure

The crucial part of our method is to retrieve the set $S_\mathbf{p}$ that represents all neighboring points that might participate in forming the function. There have been several efficient *GPU* structures proposed to retrive k-closest points to a given one [33]. Nevertheless, we utilize a simple and straightforward approach that is based on an uniform spatial subdivision. This has has been already utilized by the broader molecular visualization community [11, 14]. The atoms are sorted into cubic voxels with a lateral length of $2radius_{max} + 2R_{max}$, where $radius_{max}$ represents maximum (van der Waals) radius of all included atoms and R_{max} represents maximal allowed radius of the solvent. We set the maximal allowed radius to 2, which means that values of R can be interactively varied from 0 to 2. The sorting of the atoms into the grid is done in $O(n)$, which also represents the only and the necessary precomputation step in our pipeline. Then in order to build the set $S_\mathbf{p}$ it is required to visit $3x3x3$ neighboring voxels for a given point $\mathbf{p}$. Thus, for a given time-step, we need to send to GPU only the atom centers and their radii, and the voxels with Ids of atoms.

5.2 Rendering Parameters

The rendering process depends on several parameters, which affect the overall rendering performance and the precision ratio. With this respect these parameters can be taken as level of detail parameters.

In the general case, the size of the set $S_\mathbf{p}$, might contain an unlimited number of atoms, which would make it impossible to store them in *CUDA* memory, even with the version 4.0 that allows to allocate dynamic arrays. Fortunately, Varshney et al. has experimentally shown that a typical atom in a protein has approximately 45 neighboring atoms within the radius of a water molecule [32]. Therefore, we limit the maximum size of S to $45 = |S|$. However, such a configuration would still produce $\binom{45}{3} = 14910$ triplets to evaluate in Eq. (18). Accordingly, we introduce parameter k_{max} which specifies the maximum number of atoms being stored when building the set $S_\mathbf{p}$, with the maximal limit being 45.

With respect to this limitation, we have to choose which atoms are retrieved. Therefore, all the atoms in $S_\mathbf{p}$ are stored in the increasing order, i.e. $S = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k | f_1(\mathbf{p}) \geq f_2(\mathbf{p}) \geq \ldots \geq f_{k_{max}}(\mathbf{p}) \geq -2R\}$. Since we are ordering only k_{max} number of items, the time complexity of the sorting is $O(1)$. We evaluated the quality of the resulting surface according to different k_{max} values. Here we noticed that for $k_{max} > 10$, we obtained only small surface improvements (Fig. 7). On the other hand, when rendering cavities in details, the number of S-included atoms has to be increased since all the atoms along the cavity circumference (and for which $f_i(\mathbf{p}) \geq -2R$) might participate to the final surface.

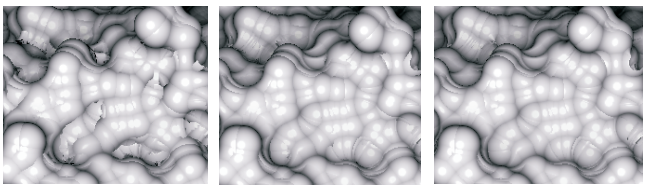

Figure 7: Comparison of the SES visual quality when having set the parameter $k_{max} = 5$ (left), $k_{max} = 10$ (middle) and $k_{max} = 15$ (right). Note that there are only negligible improvements between the last two.

The resulting implicit function defines the implicit surface at points $f_{SES}(\mathbf{p}) = 0$, which can cause a performance bottleneck during the evaluation of the exact points laying on the iso-surface. Therefore, we determine whether the point $\mathbf{p}$ lies on the iso-surface if and only if $|f_{SES}(\mathbf{p})| < \varepsilon$, where ε represents the minimal allowed proximity to the surface. Here, the exact ε value is specified on the fly; either to increase performance or to improve the surface quality.

When evaluating both systems (11) and (17), we specify the ε_N and the maximum allowed number of iterations L available for intersection point estimations, i.e $x_L = v(\mathbf{x}_{L-1})^T \cdot M(\mathbf{x}_{L-1})^{-1} + \mathbf{x}_{L-1}$. Since the intersection is always evaluated when point $\mathbf{p}$ is close to the intersection points themselves, we set $L = 10$.

5.3 Ray-casting

As was mentioned before, the parallelization is done on the ray basis. Here we compute the ray entry (t_{min}) and exit (t_{max}) parameters within the bounding box of the currently bounded scene. For simplicity, we perform only the first hit traversal. A generated ray is processed in a step-wise fashion until the t_{max} is reached or we hit the iso-surface. Since the set $S_\mathbf{p}$ is ordered increasingly from $\mathbf{p}$, we perform both toroidal and spherical triangle implicit computations only when $|S_\mathbf{p}| > 0$ and $g_1(\mathbf{p}) \geq 0$. This reflects the fact that $\mathbf{p}$ must lie at least in one area of influence of all the points in $S_\mathbf{p}$. Therefore, the function $f_{SES}(\mathbf{p})$ returns a signed distance value representing the closest proximity to the SES iso-surface, although only when being closer to the surface than R. Using this fact, the raycasting

procedure increment the ray parameter t by $-f$, since the function has negative values outside, i.e. $t = t - f$ (Fig. 8). The noticeable deficiency is that it is necessary to perform many steps when being close to the surface.

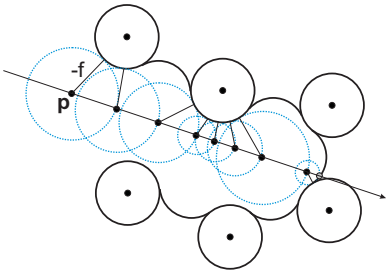

Figure 8: Ray-casting a simplified scene. We perform steps of the size of f_{SES}, in order not to skip the existing iso-surface.

5.4 Function evaluation

One of the disadvantages of implicit function evaluation is the time complexity when evaluating the function for all the points in $S_{\mathbf{p}}$. In general case, where the points have arbitrary topology, the evaluation of spherical triangle function would lead to $O(|S|^3)$ complexity. Fortunately the number of atoms participating generation of implicit function is upper bounded. With respect to this restriction we store the required combinations of $\binom{k}{2}$ and $\binom{k}{3}$, used by Eq. (18) into linear arrays into the constant CUDA memory. Computations of predicates, intersections and function values is achieved in $O(1)$, since we are iterating over systems (11,17) in the upper bounded number of computational steps. While evaluating, the only real bottleneck is being traversal of neighboring points in order to construct the set S. This is done in linear time $O(n)$. To optimize the ray-casting, we cache the intersection points while progressing along the ray.

5.5 Performance

We evaluated the performance of the new rendering method on the sequence of eight proteins (Fig. 1); murine coronin-1 (3023 atoms, a), g-actin (2872 atoms, b), potassium channel (2820 atoms, c), proteinase 3 (3346 atoms, d), immunoglobin (12530 atoms, e) proliferatic cell nuclear antigen (12555 atoms, f), bacteriorphodopsin (1718 atoms, f) and tubulin (14744 atoms, h). We were varying all presented parameters affecting the performance (k_{max}, ε, ε_N). The performance measurements were done on a workstation equipped with two (2 GHz) processors and 12.0 GB RAM and with the GPU, NVIDIA GeForce GTX 480. It is important to mention here that we do not aim at outperforming any existing rendering techniques [11, 14] regarding FPS, since we compute the iso-surface on the fly. On the other hand, our aim here is to define *SES* representation in a simple way and to provide users with an interactive response for changing any protein or solvent parameters. Furthermore, to study molecular dynamics using visualization of arbitrary sections of the sequence without the need of any precomputation steps. Therefore, the performance (FPS) is more about the fact that such a system can provide an interactive response when, for instance, included in the existing protein exploration environment.

The level of interactivity is dependent on the three aforementioned parameters, k_{max},ε and ε_N. We observed that the rendering performance is affected significantly by the parameter k_{max}. For instance, by setting $\varepsilon = 0.01$, $\varepsilon_N = 0.005$, $R = 1.4$ and by increasing incrementally k_{max} from 5 to 20, we get the following FPS:

k_{max}	a	b	c	d	e	f	g	h
5	15	18	18	14	9	10	20	8
10	13	15	16	12	8	9	16	7
20	4	5	6	3	2	2	8	2

Nevertheless, by setting $\varepsilon < 0.00001$ and setting $k_{max} > 20$ we get < 1 FPS for all the proteins. Therefore, this configuration can be used to provide high-quality images but a less interactive response. To guarantee constant framerates (above 15FPS), we implemented a progressive refinement strategy that automatically reduces these parameters,e.g.; $\varepsilon = 0.2$, $\varepsilon_N = 0.005$ and $k_{max} = 5$, when performing visual exploration (zooming, rotation) or seeking for different time-steps. When a user does not interact with the scene, these parameters are automatically increased to the higher level to get the finer details.

We demonstrated the sequence of two of presented proteins (protein d an f, where we set $k_{max} = 10$ as a constant) to the bioinformatics scientist, where we achieved satisfactory interactivity. Here, the feedback, which we have received was that visualization provides a sufficient amount of details for an overview on the protein surface with respect to the cavity exploration. Additionally, the provided interactivity was highly appreciated.

6 Error estimation and limitations

The visual surface precision is directly affect by the minimal allowed proximity, ε, to the iso-surface during the ray-casting. The parameters k_{max} and ε_N affect the function evaluation error. When a given point $\mathbf{x}$ on the ray r fulfils $f(\mathbf{x}) \leq \varepsilon$, the point $\mathbf{x}$ is considered to lie on the iso-surface, which actually holds only if ray r intersects the surface in a point $\mathbf{x}'$, i.e. $f(\mathbf{x}') = 0$. Since the function f is C^1 continuous, via the mean value theorem in several variables, there exist such a parameter $t \in [0,1]$ that

$$f(\mathbf{x}) - f(\mathbf{x}') = \nabla f\left((1-t)\mathbf{x} + t\mathbf{x}'\right) \cdot (\mathbf{x} - \mathbf{x}'). \tag{19}$$

When denoting point $\mathbf{c} = (1-t)\mathbf{x} + t\mathbf{x}'$ and since $f(\mathbf{x}') = 0$, we get the error E dependent on the following equation

$$\nabla f(\mathbf{c}) \cdot (\mathbf{x} - \mathbf{x}') = f(\mathbf{x}) \leq \varepsilon. \tag{20}$$

Using Eq. (20) we can identify the case, when function values vary rapidly and gradient magnitude can become very high, then distance between $\mathbf{x}$ and $\mathbf{x}'$ needs to be significantly smaller in order to fulfil Eq. (20). In our approach, this creates the artifacts since the parameter ε is taken as a constant during ray-casting. Instead, as a future work, it needs to be adjusted according to function complexity in the neighborhood of $\mathbf{x}'$.

We have discussed the corresponding visualization artifacts with the bioinformatical specialist. Since the *SES* representation is utilized in cavity discoveries predominantly, which are recognized only when a ball of radius R fits inside an opening, the artifacts were considered insignificant when setting $\varepsilon \ll R$.

As a consequence, the major limitation of utilizing our implicit representation for visualization purposes is that it does not have to be a 100% *SES* representation. Here, the difference to existing approaches lies in the fact that while these approaches generate a set of quadrics, our implicit function is defined globally and therefore we have to rely on the numerical approach while tracking the iso-surface.

Additionally, even one specifies $k_{max} = 45$, it can theoretically happen that the number of neighboring atoms can become higher (with the respect to MD simulations). Nevertheless, this is not the typical case, since we have noticed only little improvements when setting $k_{max} > 10$.

Another limitation is dependent more on the utilized graphics hardware, for which GPU can stall when evaluating too many points in set S.

7 Utility potential

Our implicit representation is tailored for the purpose of detecting, analyzing and visualizing cavities; i.e tunnels or channels. Tunnels are essentially pathways leading from a cavity to a protein core. Channels lead through the whole protein structure having cavities on both ends. The study of how these pathways evolve is highly important and required for drug design. Usually, to discover a channel, it is necessary to find the cavity and then track the cavity during the protein simulation, to find out whether it reaches the surface.

Contrary to previous approaches [11, 14], for a given point **x** our *SES* implicit function (Section 4) computes the closest distance to the surface, the gradient at **x**, and classifies **x** according to whether it lies inside or outside of the protein atoms. By means of numerical methods, we can compute the volume area of the cavity using point-to-object classification. Another utilization is to track down the cavity automatically using the distance property of the implicit function and using its gradient. We can delimit the cavity detection only to places, where a solvent will fit in. In other words, we locate points where $f_{SES} < -R$, i.e. points that are above the distance of solvent radius R.

8 Conclusion

We presented a new method of representing *SES* by means of functional representation of objects. The implicit function computes *SES* representation locally. For a given point **p** the function returns the correct minimal distance to the surface, although only up to the distance of the solvent radius R. Additionally, the function is positive inside the *SES* object and negative outside. The main contribution is that we represent the molecular surface without precomputation, thus enabling instantaneous visual analysis.

Moreover, we proposed a simple ray-casting method which is used to render even large proteins at reasonable interactivity. We introduced a set of parameters that affect the overall rendering performance. These can be adjusted in accordance with the rendering performance versus a precision ratio.

Since the work forms the first steps in a new direction of *SES* representation, we have performed only an informal evaluation. We showed the implementation of our method to specialists where we demonstrated it on a sequence of the two presented proteins. The possibility of changing the solvent radius or even to pick up atoms in related views, was found to be highly interesting and promising.

Acknowledgements

We give thanks to Nathalie Reuter for providing the molecular dynamics simulation datasets and the necessary feedback, and to Armin Pobitzer for final touches with mathematical formulas.

References

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

[2] M. Chavent, B. Levy, and B. Maigret. MetaMol: high-quality visualization of molecular skin surface. *Journal of molecular graphics & modelling*, 27(2):209–16, Sept. 2008.

[3] M. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16(5):548–558, 1983.

[4] H. Edelsbrunner. Deformable smooth surface design. *Discrete & Computational Geometry*, 21(1):87–115, 1999.

[5] P.-a. Fayolle. Technical Report 2009-001 Distance to set operations in constructive modeling of solids. *City*, 2009.

[6] J. Greer and B. L. Bush. Macromolecular shape and surface maps by solvent exclusion. *Proceedings of the National Academy of Sciences of the United States of America*, 75(1):303–7, Jan. 1978.

[7] P. Hanrahan. Ray tracing algebraic surfaces. *SIGGRAPH Comput. Graph.*, 17:83–90, July 1983.

[8] J. C. Hart. Sphere tracing : a geometric method for the antialiased ray tracing of implicit surfaces. *Computer*, pages 527–545, 1992.

[9] W. Humphrey, A. Dalke, and K. Schulten. VMD: visual molecular dynamics. *Journal of molecular graphics*, 1(14):33–38, 1996.

[10] A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic. *Computer Graphics Forum*, 28(1):26–40, Mar. 2009.

[11] M. Krone, K. Bidmon, and T. Ertl. Interactive visualization of molecular surface dynamics. *IEEE transactions on visualization and computer graphics*, 15(6):1391–8, 2009.

[12] M. Krone, M. Falk, and S. Rehm. Interactive Exploration of Protein Cavities. *Computer Graphics Forum*, 30(3):673–682, 2011.

[13] B. Lee and F. M. Richards. The interpretation of protein structures: estimation of static accessibility. *Journal of molecular biology*, 55(3):379–400, Feb. 1971.

[14] N. Lindow, D. Baum, S. Prohaska, and H. Hege. Accelerated Visualization of Dynamic Molecular Surfaces. In *Computer Graphics Forum*, volume 29, pages 943–952. Wiley Online Library, 2010.

[15] C. Loop and J. Blinn. Real-time GPU rendering of piecewise algebraic surfaces. *ACM Transactions on Graphics (TOG)*, 25(3):664–670, 2006.

[16] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, August 1987.

[17] T. Luft, C. Colditz, and O. Deussen. Image enhancement by unsharp masking the depth buffer. *ACM Transactions on Graphics*, 25(3):1206–1213, jul 2006.

[18] A. A. Pasko, V. Adzhiev, A. Sourin, and V. V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995.

[19] G. Reina and T. Ertl. Hardware-accelerated glyphs for mono-and dipoles in molecular dynamics visualization. In *Proceedings of EUROGRAPHICSIEEE VGTC Symposium on Visualization*, volume xx, pages 177–182, 2005.

[20] A. Ricci. A constructive geometry for computer graphics. *The Computer Journal*, 16(2):157–160, 1972.

[21] F. M. Richards. Areas, volumes, packing, and protein structure. *Annual Review of Biophysics and Bioengineering*, 6(1):151–176, 1977.

[22] J. Ryu, Y. Cho, and D.-S. Kim. Triangulation of molecular surfaces. *Computer-Aided Design*, 41(6):463–478, June 2009.

[23] J. Ryu, R. Park, and D. Kim. Molecular surfaces on proteins via beta shapes. *Computer-Aided Design*, 39(12):1042–1057, Dec. 2007.

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

[25] Schrödinger, LLC. The PyMOL molecular graphics system, version 1.3r1. August 2010.

[26] C. Sigg, T. Weyrich, M. Botsch, and M. Gross. GPU-based ray-casting of quadratic surfaces. In *Eurographics Symposium on Point-Based Graphics*, pages 59–65. Citeseer, 2006.

[27] J. M. Singh and P. J. Narayanan. Real-time ray tracing of implicit surfaces on the GPU. *IEEE transactions on visualization and computer graphics*, 16(2):261–72, 2010.

[28] M. Tarini, P. Cignoni, and C. Montani. Ambient occlusion and edge cueing to enhance real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1237–1244, 2006.

[29] R. Toledo and B. Lvy. Extending the graphic pipeline with new gpu-accelerated primitives. Technical report, INRIA-ALICE, 2004.

[30] M. Totrov and R. Abagyan. The contour-buildup algorithm to calculate the analytical molecular surface. *Journal of structural biology*, 116(1):138–43, 1996.

[31] M. van der Zwan, W. Lueks, H. Bekker, and T. Isenberg. Illustrative Molecular Visualization with Continuous Abstraction. pages 683–690, Bergen, Norway, 2011. Eurographics Association.

[32] A. Varshney, F. P. Brooks, Jr., and W. V. Wright. Computing smooth molecular surfaces. *IEEE Comput. Graph. Appl.*, 14:19–25, September 1994.

[33] K. Zhou, Q. Hou, R. Wang, and B. Guo. Real-time kd-tree construction on graphics hardware. In *ACM SIGGRAPH Asia 2008 papers*, SIGGRAPH Asia '08, pages 126:1–126:11. ACM, 2008.