Fast Blending Scheme for Molecular Surface Representation

Julius Parulek, *Member, IEEE*, and Andrea Brambilla, *Student Member, IEEE*

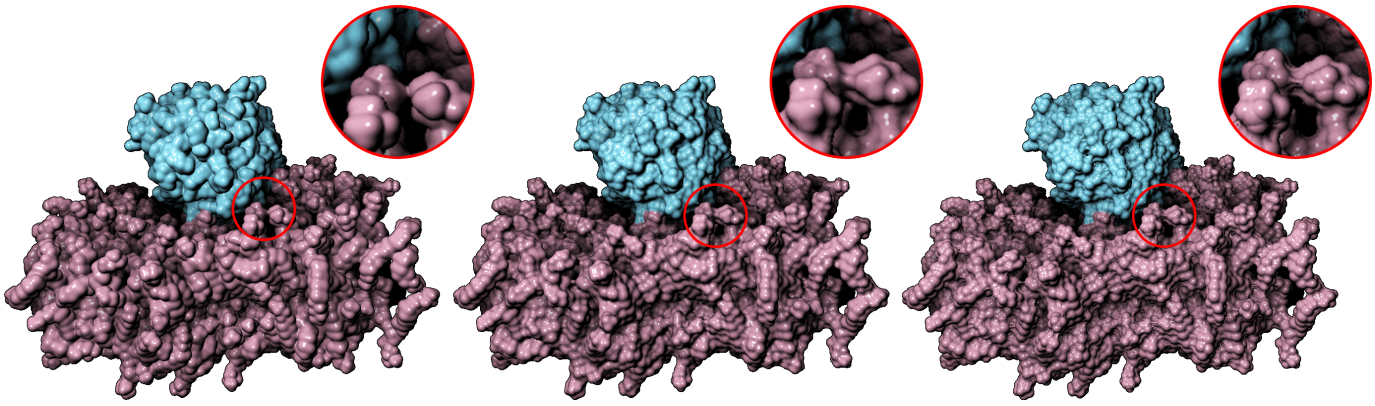

Fig. 1. Three molecular surface representations of the phospholipase (blue) bound to the lipid membrane (red) (34490 atoms in total). Left: Gaussian surface model. Middle: Our representation. Right: Solvent excluded surface model. The function evaluation of the implicit surface has linear complexity for the Gaussian model and cubic complexity for the solvent excluded surface. Our technique has linear complexity and is able to produce a surface representation that resembles the solvent excluded surface model.

Abstract—Representation of molecular surfaces is a well established way to study the interaction of molecules. The state-of-the-art molecular representation is the SES model, which provides a detailed surface visualization. Nevertheless, it is computationally expensive, so the less accurate Gaussian model is traditionally preferred. We introduce a novel surface representation that resembles the SES and approaches the rendering performance of the Gaussian model. Our technique is based on the iterative blending of implicit functions and avoids any pre-computation. Additionally, we propose a GPU-based ray-casting algorithm that efficiently visualize our molecular representation. A qualitative and quantitative comparison of our model with respect to the Gaussian and SES models is presented. As showcased in the paper, our technique is a valid and appealing alternative to the Gaussian representation. This is especially relevant in all the applications where the cost of the SES is prohibitive.

Index Terms—Molecular visualization, geometry-based techniques, implicit surfaces

1 Introduction

The research field of computational molecular biology aims at improving the understanding of the molecular machinery of life at the highest magnification level. Molecules are not static objects and it is necessary to take their dynamics and their mutual interactions into account. Molecular interactions can be studied through the analysis of molecular dynamics (MD) simulations. The simulations result in large datasets, called *trajectories*, containing the sequence of molecular structures (thousands of MD structures) as they vary along the simulation time.

With respect to molecular visualization, the essential requirement is to display a large amount of atoms at interactive frame rates in order to enable the visual analysis of molecular surfaces. Moreover, simulated datasets do not longer consist of only one moderately sized macromolecules, but instead of molecular systems representing complex interactions, e.g., a phospholipid vesicle membrane together with proteins anchored in the membrane (Fig. 1). Therefore, one can easily obtain datasets where tens or hundreds of thousands of atoms are animated throughout thousands of time-steps.

- *Julius Parulek is with the Department of Informatics, University of Bergen, Norway. E-mail: julius.parulek@uib.no*
- *Andrea Brambilla is with the Department of Informatics, University of Bergen, Norway. E-mail: andrea.brambilla@uib.no*

Manuscript received 31 March 2013; accepted 1 August 2013; posted online 13 October 2013; mailed on 4 October 2013.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

Computational biologists are now approaching a new level of interactive molecular dynamics that allows for changing the MD structure in real-time by utilization of coarse grained simulations [6]. Here, MD datasets have to be visualized directly from the simulation: streamed molecular surface visualization. In such a case, preprocessing of frames should be eliminated and the sequence should be rendered in real-time without adopting any proxy structure, such as octree or other space partitioning schemes.

The exploratory process of MD simulation is often concerned with the visual identification of binding sites of ligands to a host macromolecule. Usually 3D molecular visualization conveys the molecular structure so that the binding sites can be located through visual inspection. The state-of-the-art molecular representation is called *solvent excluded surface* (SES) [33]. It reveals possible binding sites by rolling a solvent (approximated by a ball of radius R) on the *solvent accessible surface*. The main drawbacks of the *SES* representation, with respect to visual exploration, are the high computational complexity and the need for substantial pre-computations [35, 42]. There are several approaches in the literature that generate and visualize the SES representation on the GPU [22, 16]. They can achieve fast rendering performances, but they rely on pre-computations and their scalability is limited. Additionally, their primary focus is on the rendering aspects, while the actual volumetric representation still remains unsolved.

For these reasons, the Gaussian model is still preferred in practical applications. It is easier to implement, to render and it can be also easily converted to a voxel-based representation [3]. However, the approximation provided by the Gaussian model does not fully imitate the solvent accessibility [17]. Recently, Parulek and Viola [29] introduced an implicit representation for SES. Their approach allows to perform,

1077-2626/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

for instance, the voxelization of the model. However, the computational complexity still remains very high.

In our paper, we propose a new representation for molecular surfaces that is easy to implement and evaluates the molecular function locally, similarly to the Gaussian model. As a reference we take the SES representation, where we imitate the rolling of the solvent sphere using a new blending scheme. Initially our scheme blends two atoms together while computing the first and second order partial derivatives, that are later employed to iteratively merge all other atoms. Moreover, we introduce a method that efficiently visualizes such a function by means of ray-casting. We adopt the A-buffer technique [4, 44] that lets us determine the ray-surface intersection without employing any underlying data structure. We compare our new representation qualitatively and quantitatively with the Gaussian and SES models. Our contribution can be summarized as follows:

i) a new solvent excluded representation for molecular surface, that needs no pre-computations, and is free of any supporting structure, i.e., no grids nor octrees.

ii) a demonstration of the potential of our approach for interactive molecular dynamics visualization, i.e., real-time adjustments of solvent radius and atom positions.

2 Related Work

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

2.1 Representation of Molecular Surfaces

There are several types of molecular surface representations that can be found in the literature. Due to space limitation, we address only the ones closely related to our technique. One of the basic representations is given by depicting the atoms as spheres. The radius of the spheres corresponds to the so-called *van der Waals force* (vdW surface) [21]. The extension of spherical radii by a solvent radius R is called *solvent accessible surface*, and it provides information about the areas accessible to a solvent molecule of radius R. Closely related to the solvent accessible surface is the SES representation. It is formed by rolling a sphere of radius R (the solvent), over the vdW spheres, while its center lies on the boundary of the solvent accessible surface [33, 10]. The SES is the most widely used representation for binding site analysis. However, due to its high-computational cost, convolution kernels are frequently preferred [36].

One of the most popular kernels with respect to molecular surfaces is based on the Gaussian model [3]. Our approach does not involve a kernel function, but instead it iteratively blends atoms together according to the solvent sphere radius. We develop a method that efficiently incorporates the solvent sphere and, at the same time, retain interactivity. In 2007, Bates et al. [2] described a method that generates the molecular surface by means of curvature minimization. This entire approach requires a grid representation of the molecular structure, where the surface is iteratively minimized by computing eigenvalues of Hessian matrices. In contrast, in our blending scheme, we specify one of the surface principal curvatures using the circular model directly.

There has been a lot of effort put into efficient generation and visualization of SES in the literature. In 1983, Connolly [5] proposed an analytical description of the *SES* representation. More than ten years later, Sanner et al. [35] presented the reduced surface algorithm for construction of SES. In the same year, Totrov and Abygyan introduced the contour-buildup algorithm [42] to form the SES representation, which was later optimized and parallilized by Lindow et al. [22]. The visualization is then performed using a ray-casting method. Recently, Krone et al. [18] improved the computation of the contour-buildup algorithm. In 2009, Krone et al. [16] also introduced the utilization of the reduced surface method for direct ray-casting of molecular surfaces. Both these GPU techniques focus only on rendering aspects, and build quadric and quartic primitives prior to surface rendering.

In 2012, Parulek and Viola [29] proposed an algorithm that computes a signed distance to the SES representation. Their approach provides direct computation and visualization of the molecular surface. Nevertheless, the implicit function evaluation at a given point has cubic time complexity, which limits the interactive performance. Moreover, it also requires an underlying data structure to hold the atom locations. In our approach the function evaluation complexity is linear. Additionally, no supporting data structure is required, which allows us to instantly render molecular surfaces from a given set of atoms.

2.2 Modeling through Implicit Space

Implicit surfaces (*implicits*) provide a way to easily model biological structures [7]. They naturally enable modelling of smooth objects in a convenient way.

The set of techniques, known today as *implicit modeling*, was proposed for the first time by Blinn [3]. Blinn introduced the Gaussian convolution kernel in order to blend atom potentials. Later on, Pasko et al. generalized the representation of implicits, by combination of the different forms of implicit models [30]. This generalization was described by the inequality $f(\mathbf{x}) \geq 0$. This predicate describes an implicit solid object, where $\mathbf{x} = (x_1, x_2, x_3) \in R^3$. Function f is called implicit surface function (implicit function), and it classifies the space into two half-spaces, i.e., $f(\mathbf{x}) > 0$ and $f(\mathbf{x}) < 0$. We employ this representation to form the molecular surface function. The major advantage is the possibility to build more complex objects using binary operators, such as union or intersection [8]. More advanced modeling techniques use the so-called *implicit space*, proposed by Barthe et al. [1]. A user can interactively sketch the required shape modifications in the space formed by the potential fields of both input functions, $I^2 = [f_1(\mathbf{x}), f_2(\mathbf{x})]$. Nevertheless, this method does not reflect the mutual orientation and position of the objects at a given point $\mathbf{x}$. In 2008, Fayolle et al. [8] proposed a method that computes a correct distance surface measure for CSG operations in I^2. However, the obvious limitations are the same as in the work of Barthe et al. [1]. In 2012, Gourmel et al. [9] introduced a technique that allows to specify the final surface shape according to the function values and also their gradients. In our approach, we use I^2 to form a circular model based on the radius R and the gradients of both input objects. Therefore, the work of Gourmel et al. is related to ours in terms of gradient-based modeling. Their work was primarily focused on merging two objects only, while blending more than two objects was just sketched, and thus remained unsolved. Here we propose how to solve the blending between multiple objects that in our case are molecular atoms.

2.3 Real-time Rendering of Implicit Objects

In order to visualize models based on implicits, one can convert them to a mesh representation and later render them as a set of patch primitives [24]. For instance, efficient triangulation of SES was proposed by Ryu et al. [34]. However, when dealing with complex models and shapes such as molecular surfaces, we would need to generate millions of triangles in order to obtain a fully detailed surface representation.

Therefore, with the recent advancements of GPU hardware, researchers turned to direct visualization techniques, in particular to ray-casting methods. Since implicts encompass different forms of geometrical models, the various ray-casting methods are designed according the type of the implicit that has to be rendered. The ray-casting of SES presented by Krone et al. [16] and Lindow et al. [22] is performed by ray-tracing quadrics, after the molecular surface has been decomposed into set of basic spherical and toroidal primitives. In general, the ray-casting of algebraic surfaces, such as quadrics, is extensively covered by the computer graphics literature and goes back several decades. Hanrahan introduced ray-casting of algebraic implicit models up to the fourth order [11]. Later work addressed ray-casting of large number of quadrics on GPU aimed at molecular visualization as well [41, 32, 23]. Later on, Sigg et al. [37] introduced very fast GPU rendering of spheres, ellipsoids and cylinders, focusing mainly on molecular rendering.

In 1994, Hart introduced a more robust approach for ray-casting of distance-based implicit surfaces called *sphere tracing* [12]. Hart's technique is efficient and easy to implement. So, since we also deal with distance-based implicit functions, we included it in our pipeline.

Several studies address ray-casting of general implicit surfaces on the GPU using interval arithmetic [15]. Nevertheless, interval arithmetic requires the definition of dedicated mathematical operators, which is very challenging when dealing with compound objects. Another GPU-based ray-casting of implicits is achieved via the *adaptive marching point* method [38]. However, this method assumes the function is defined globally, whereas our function is computed only locally.

The fast rendering of metaballs on GPU hardware was proposed by Szecsi and Illes [39], which employs the *A-buffer* technique or *fragment linked list*. Here we present a similar approach, although the function computation is different, since, in their work, the main goal was to render molecules defined by blobby objects. Additionally, they assume that the scene is already ordered. In contrast, we make no assumptions regarding ordering, and our rendering parameters defining the molecular surface can be varied arbitrarily.

3 Proposed Approach

MD datasets are composed of a large set of atoms describing the structure of one or more molecules. Specifically, let m be the number of atoms in a dataset. The atom $i \in \{1,2,\ldots,m\}$ is described by its *center* $\mathbf{c}_i \in \mathbb{R}^3$ and the corresponding *van der Waals radius* $r_i \in \mathbb{R}$.

We define a set of m implicit functions $G = \{g_1, g_2, \ldots, g_m\}$, such that

$$g_i(\mathbf{x}) = r_i - ||\mathbf{x} - \mathbf{c}_i||, \tag{1}$$

with $\mathbf{x} \in \mathbb{R}^3$. Then, every atom i can be represented by the iso-surface $g_i(\mathbf{x}) = 0$ of the corresponding implicit function. Each function g_i is a distance function and it divides the spatial domain into two half-spaces: the interior of the atom ($g_i(\mathbf{x}) \geq 0$) and the exterior ($g_i(\mathbf{x}) < 0$). From now on, when the meaning is clear, we will not specify the spatial location $\mathbf{x}$, e.g., we will write $g_i = 0$ instead of $g_i(\mathbf{x}) = 0$.

We define an implicit function $l : \mathbb{R}^3 \to \mathbb{R}$ such that the iso-level $l = 0$ represents the surface of the molecule. For a given point $\mathbf{x}$, $l(\mathbf{x})$ expresses a distance measure of the point $\mathbf{x}$ from the molecular surface, with positive values inside and negative values outside the molecule. The function $l(\mathbf{x})$ is evaluated locally by blending a subset $G_\mathbf{x} \subseteq G$ of the atom functions according to the solvent radius R. At a given location our blending scheme requires the gradient ∇g_i and the Hessian H_{g_i} of each atom function g_i. The whole computation process is iterative, and the gradient ∇l and the Hessian H_l of function l have to be evaluated at each iteration (Alg. 1).

Algorithm 1 Evaluation of the implicit function for a point $\mathbf{x}$.

```
1:  build the set of atoms G_x = {g_1, g_2, ..., g_n} affecting x (Eq. 4)
2:  for each g_i, compute ∇g_i and H_{g_i} (Eqs. 1,2 and 3)
3:  initialize l, ∇l and H_l
4:  for all g ∈ G_x do
5:      if x lies in the area influence of l and g (Eq. 6) then
6:          compute the dot product k = ∇l · ∇g
7:          update l (Eq. 9)
8:          update ∇l (Eq. 11)
9:          update H_l (Eq. 13)
10:     end if
11: end for
12: return l
```

Let us first give a formal definition of ∇g_i and H_{g_i}. Recall that the gradient is a vector given by the first order partial derivatives. Let $\mathbf{v} = (\mathbf{x} - \mathbf{c}_i)$. Then, the gradient of an atom function g_i is

$$\nabla g_i(\mathbf{x}) = \frac{1}{|\mathbf{v}|}[v_x, v_y, v_z]^T. \tag{2}$$

The Hessian H_{g_i}, instead, is the matrix of second order partial derivatives

$$H_{g_i}(\mathbf{x}) = \frac{1}{|\mathbf{v}|^{\frac{3}{2}}} \begin{bmatrix} -(v_y^2 + v_z^2) & v_x v_y & v_x v_z \\ v_x v_y & -(v_x^2 + v_z^2) & v_y v_z \\ v_x v_z & v_y v_z & -(v_x^2 + v_y^2) \end{bmatrix}. \tag{3}$$

The algorithm begins by determining the set of atoms $G_\mathbf{x} \subseteq G$ that may affect the function value in $\mathbf{x}$ (line 1). This is in analogy to the Gaussian representation, which takes into account only the atoms within the area of influence of the Gaussian kernel. More details are provided in Section 3.1.

The function l is then evaluated by looping over the set $G_\mathbf{x}$. At every iteration, a new atom $g_i \in G_\mathbf{x}$ is taken into account. A simple test (line 5) determines if g_i has to be blended with the current function l. If the test is passed, the merging procedure is carried out. The blending scheme is based on the implicit space approach: the two functions are blended by an implicit circle whose radius is defined by the solvent radius R (Sec. 3.3). This results in a new function value representing a distance measure to the spherical blend of both input functions. The computation involves the gradient ∇l and the Hessian H_l of the current function. These values have to be updated every time a new atom is blended, since they will be required in the next iteration (Sec. 4).

The computation of l is performed on the fly during ray-casting. After l is evaluated at a location $\mathbf{x}$, the ray-casting procedure continues and the ray advances to the next point. The intersection between the ray and the molecular surface is detected when the value of $l(\mathbf{x})$ is close to 0 (up to a small threshold). We based our ray-casting procedure on the A-buffer technique, since it allows for fast empty space skipping. Moreover, it lets us determine the set of relevant atoms $G_\mathbf{x}$ in a simple and efficient way (Sec. 5.1).

The ray-casting algorithm is implemented in CUDA and runs in parallel on the GPU. Ray-traversal and the consequent function evaluations are performed for every image pixel. Finally, according to the evaluated function values, depth and normals/gradients, we compute shading, silhouettes and depth enhancement by means of screen space ambient occlusion [25].

3.1 Relevant Atoms

The set $G_\mathbf{x} = \{g_1, g_2, \ldots, g_n\}$ is given by all the atoms that may affect the value of the target function l at point $\mathbf{x}$. Specifically, $G_\mathbf{x}$ is constituted by the functions g_i such that $\mathbf{x}$ lies within a distance of $2R$ from the iso-surface $g_i = 0$, i.e.,

$$G_\mathbf{x} = \{g_i | g_i(\mathbf{x}) \geq -2R\}. \tag{4}$$

An example of this can be seen in Figure 2. The definition of $G_\mathbf{x}$ is directly derived from the detection of two incident atoms, which holds if $||\mathbf{c}_i - \mathbf{c}_j|| - r_i - r_j \leq 2R$. This property was already shown by Varshney et al. [43].

Before initiating the main loop, in order to improve efficiency, each function g_i is evaluated in $\mathbf{x}$. The gradients ∇g_i and the Hessians H_{g_i} are computed as well.

Function l is computed iteratively, so its initial value (and the initial values of its gradient and Hessian) has to be set. Theoretically, any $g_i \in G_\mathbf{x}$ could be used for the initialization l. However, we obtain better results if we use the function g_i with the highest value, i.e., the one corresponding to the closest atom to $\mathbf{x}$. This difference is related

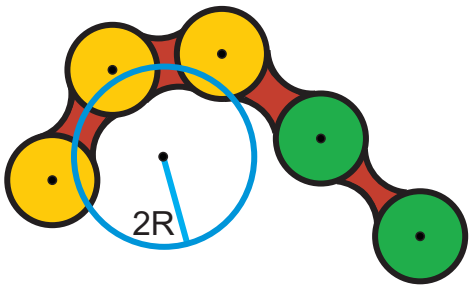

Fig. 2. Forming the point set $G_\mathbf{x}$. Out of all the atoms, the set $G_\mathbf{x}$ represents those (yellow circles) that belong to the area of influence of the point $\mathbf{x}$ (blue circle).

to the approximation scheme adopted for the higher order derivatives (see Section 4.2).

3.2 Implicit Space Model

For the sake of simplicity, let us first focus on the case where only two atoms are contributing to the evaluation of l. Let f and g be the implicit functions representing the two atoms, that is $G_{\mathbf{x}} = \{f, g\}$. Our goal is to determine the new value of l by blending f and g in a way that resembles rolling a ball of radius R.

This is achieved by performing a circular blend in the so-called implicit space [1, 8]. The current point $\mathbf{x}$ is mapped to a bi-dimensional space where each axis represents one of the participating functions. In our case, the implicit coordinates of point the $\mathbf{x}$ are $[f(\mathbf{x}), g(\mathbf{x})]$.

The direct way of computing the union of two implicit functions at $\mathbf{x}$ is to take the maximum of the values of the two functions. However, this approach generates sharp discontinuities, as shown in Figure 3 (top). In the implicit space, our goal can be reformulated as defining an implicit function that smoothly blends the iso-lines of f and g according to a circle of radius R. It is also required that the shape of the circular blend is preserved for every iso-level (Fig. 3 bottom-right). Such a function can be constructed by sweeping an implicit circle of radius R along the line $f = g$, passing through the origin. We express the center of this circle as $(l-R, l-R)$. Then the circle is given by the equation:

$$(f-(l-R))^2 + (g-(R-l))^2 = R^2.$$

Here l represents the distance of the current point from the circumference, which is also the value we need to express. For more designing principles, we refer the readers to the study of Fayolle et al. [8].

Solving the previous equation for l yields two solutions:

$$l = \frac{1}{2}\left(2R + f + g \pm \sqrt{2R^2 - (f-g)^2}\right). \tag{5}$$

It is sufficient to consider only the top-right circular section (Fig. 3 bottom-right), which is represented by the smaller of both distances, i.e., the one with the minus sign in Equation 5.

Additionally, Equation 5 has to be evaluated only for points that lie between the two dark blue lines (Fig. 3). Outside this area we employ the direct union operator $max(f, g)$ in order to merge the two functions. This test corresponds to line 5 of Algorithm 1. In order to evaluate this condition, the following inequalities have to be fulfilled: $f \leq g + R$ and $g \leq f + R$. For simplicity, they can be put together into a single predicate:

$$\frac{(f(\mathbf{x}) - g(\mathbf{x}))^2}{R^2} \leq 1. \tag{6}$$

Figure 3 (bottom-left) shows the result of this approach when merging two atoms. Although the iso-contour of f is indeed smooth, the curvature increases as the iso-contour approaches the area lying in-between the circles. This distortion is due to the fact that the implicit space model does not take into account the actual spatial relationship between the two input functions. Additionally this causes holes (Fig. 5 left) for atoms having their iso-contours separated by a distance within $[R, 2R]$, while they should be merged.

3.3 Model Scaling

One of the possible ways to overcome this issue is to adaptively scale the radius R. We propose to encode the information regarding the spatial relationship between the functions using the dot product of their gradients, i.e., $k(\mathbf{x}) = \nabla f(\mathbf{x}) \cdot \nabla g(\mathbf{x})$. The utilization of gradients to encode spatial relationships has been already proved to be efficient for implicit modeling [9]. We replace the constant radius R with the function $r(\mathbf{x}) = R r_k(\mathbf{x})$, where $r_k : [-1, 1] \to \mathbb{R}$ depends on the dot product k. The radius scaling function r_k affects the curvature of the blending regions. It can be defined in different ways, but it has to satisfy the following constraints. Since we have to compute its second derivatives (Sec. 4.1), r_k must be at least C^2 continuous. It should be monotonically decreasing, because the radius should decrease as the angle between the gradients gets smaller (and vice versa). Finally, the

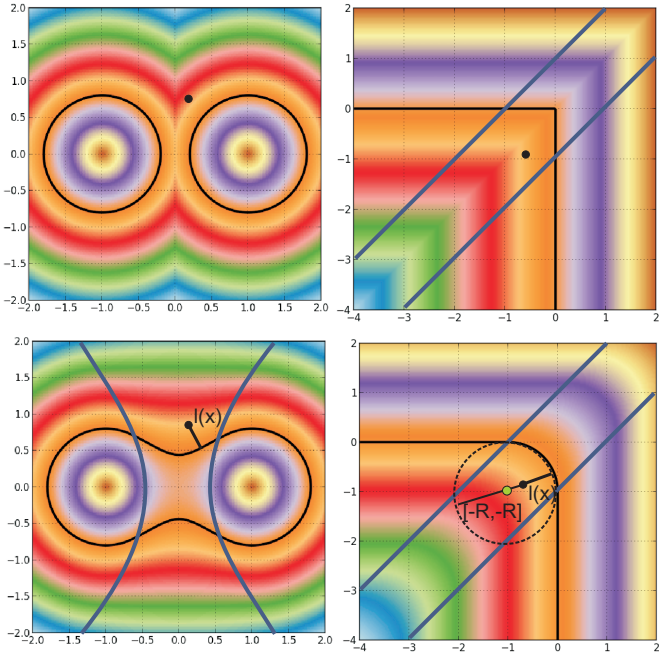

Fig. 3. An illustration of implicit space blending. Top-Left: Function values of the functions f and g. Function f represents an implicit circle with center $[1,0]$ and radius 0.8, and function g a circle with center $[-1,0]$ and the same radius 0.8. The 0 iso-contours (in black) are defined using the max operation, i.e., the union of f and g. Top-Right: The function values represent a two dimensional point in the implicit space. The x-axis stands for the 0 iso-surface of function f while the y-axis is the 0 iso-surface of function g. The iso-contours are defined again by the max operator although in the implicit space this time. Bottom-Right: When the point belongs to the area delineated by two blue lines $f+R$ and $g+R$ (Eq. 6), we evaluate the implicit circle model of center $[-R,-R]$ with radius R (Eq. 5). Elsewhere, the resulting function equals $max(f,g)$. Bottom-left: The final object obtained in the spatial domain. The effective area (Eq. 6) in object space is delineated by the two blue curves.

scaled radius should be positive, therefore r_k must be positive as well in $[-1, 1]$. According to these constraints, we set

$$r_k(\mathbf{x}) = 1 - \sin\left(\frac{\pi}{4} k(\mathbf{x})\right), \tag{7}$$

which we found suitable for mimicking the SES representation (Sec. 6). Figure 4 (left) shows a plot of r_k against the values of the dot product, while Figure 4 (right) shows $r_k(\mathbf{x})$ with respect to the two atom examples of Figure 3.

We replace R with $r(\mathbf{x})$ in Equations 5 and 6. Using the scaled radius, our target function l becomes

$$l = \frac{1}{2}\left(2r + f + g - \sqrt{2r^2 - (f-g)^2}\right). \tag{8}$$

By applying Equation 8 to the example in Figure 3, the resulting function is shown in Figure 4 (bottom). Now, in the situation when the distance between the iso-contours is within $[R, 2R]$, Equation 8 correctly produces a smooth blend between the two atoms (Fig. 5 right).

4 Iterative Representation

The previous considerations are limited to the simple case of two atoms. When the set $G_{\mathbf{x}}$ includes more than two atoms, we propose to compute l using an iterative approach.

The basic idea is to start from a single atom. Then, at every iteration, a new function from $G_{\mathbf{x}}$ is blended with the current function, according to the implicit space model. In practice, we are iteratively

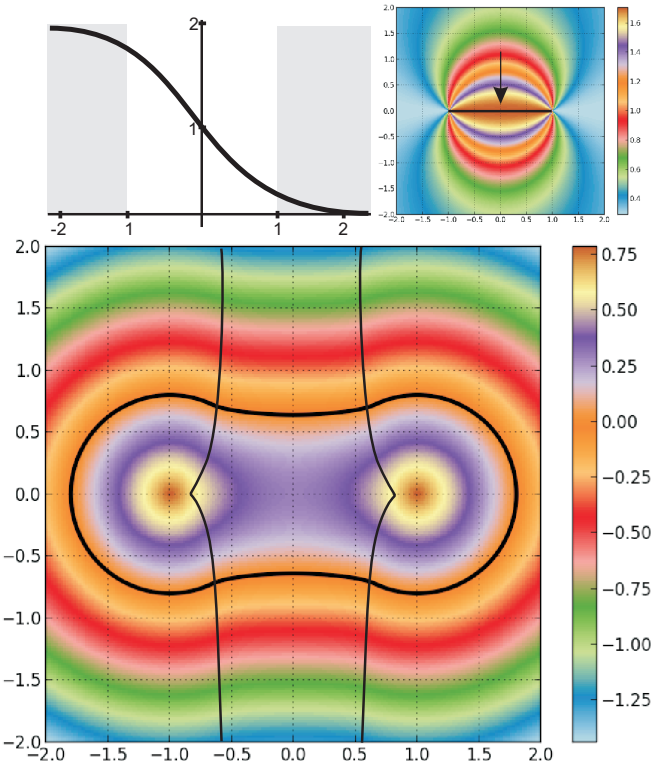

Fig. 4. An example of scaling the circular model. Top-left: To scale the solvent radius R, we employ function (7) applied to the dot product of both input gradients. Top-right: Depiction of the function r_k in $\mathbb{R}^2$. Note the way how the function/radius increases (the arrow) towards the line segments connecting both sphere centers (line segment). Bottom: An example of the final iso-contour obtained by the scaled radius r_k for the same spheres as in Figure 3. The area of influence (6) is marked by two vertical curves.

evaluating Equation 8, and every time we replace f with the function computed in the previous iteration.

Formally, given the set of relevant atom functions $G_{\mathbf{x}} = \{g_1, g_2, \ldots, g_n\}$, we consider a series of functions $\{l_1, l_2, \ldots, l_n\}$. The i-th function l_i is given by the iterative blending of the functions $\{g_1, g_2, \ldots, g_i\}$. Therefore, the n-th function l_n corresponds to our target function l.

As mentioned before, we start the iteration from the atom closest to the point $\mathbf{x}$. Therefore, without loss of generality, we assume that the first function g_1 represents such an atom. We start by setting $l_1 = g_1$, $\nabla l_1 = \nabla g_1$ and $H_{l_1} = H_{g_1}$. Then, at each iteration, l_i is computed according to Equation 8 with $f = l_{i-1}$ and $g = g_i$:

$$l_i = \frac{1}{2}\left(2r + l_{i-1} + g_i - \sqrt{2r^2 - (l_{i-1} - g_i)^2}\right). \tag{9}$$

The main difference is that, after the first iteration, function f does not represent an atom but a compound object, so we cannot evaluate it according to Equation 1. Similarly, we cannot use Equation 2 to compute its gradient, but we have to define a proper formula.

It is worth pointing out that, at every iteration, the predicate of Equation 6 has to be evaluated using the atom g_i and the previous function l_{i-1}. If the condition is not satisfied, g_i is not taken into account. In rare cases, it may happen that the test fails because l_{i-1} has not been fully evaluated yet. This issue can be simply addressed by iterating over $G_{\mathbf{x}}$ one more time. During this second pass, l_n has been already computed, so the functions g_i, that were incorrectly excluded, will now be correctly taken into account.

4.1 Derivatives of l

Let us have a closer look at a single iteration i. For the sake of simplicity, in the following discussion we adopt the notation of Equation 8. We refer to the current function l_i as l, to the previous function l_{i-1} as f, and to the current atom g_i as g. In every single iteration, our goal is to compute $l(\mathbf{x})$. The function l depends on the scaled radius function r, which in turn depends on the dot product $k = \nabla f \cdot \nabla g$. Since $g = g_i$ represents an atom, ∇g can be computed analytically (Eq. 2). On the other hand, $\nabla f = \nabla l_{i-1}$ is the gradient of l from the previous iteration, so it has to be computed iteratively. Since l is a composite function, its derivation is based on the *chain rule*:

$$\frac{dl}{d\mathbf{x}} = \frac{\partial l}{\partial f}\frac{df}{d\mathbf{x}} + \frac{\partial l}{\partial g}\frac{dg}{d\mathbf{x}} + \frac{\partial l}{\partial r}\frac{dr}{d\mathbf{x}}. \tag{10}$$

Switching to the nabla notation, and taking into account that r is a function of the dot product k, Equation 10 can be rewritten as:

$$\nabla l = \frac{\partial l}{\partial f}\nabla f + \frac{\partial l}{\partial g}\nabla g + \frac{\partial l}{\partial r}\frac{\partial r}{\partial k}\nabla k. \tag{11}$$

The partial derivatives of l are obtained deriving Equation 8:

$$\frac{\partial l}{\partial f} = \frac{\partial l}{\partial g} = \frac{1}{2}\left(1 + \frac{(f-g)}{q}\right),$$

$$\frac{\partial l}{\partial r} = \left(1 - \frac{r}{q}\right),$$

where $q = \sqrt{2r^2 - (f-g)^2}$. The partial derivative of r depends on how r_k is defined, and in our case (Eq. 7) it is given by

$$\frac{\partial r}{\partial k} = -\frac{\pi R}{4}\cos\left(\frac{\pi}{4}k\right).$$

Finally, the gradient ∇k can be obtained by applying the derivation rule for the dot product [20]:

$$\nabla k = \nabla(\nabla f \cdot \nabla g) = H_f \nabla g + H_g \nabla f, \tag{12}$$

where H_f and H_g are the Hessian matrices of function f and g respectively.

In analogy with the gradients, H_g can be computed analytically (Eq. 3). In contrast, H_f should be computed iteratively and the resulting formula would include third order derivatives. In other words, the exact evaluation of l_i requires derivatives up to order $i-1$ from the previous iterations.

It is worth mentioning that the product $H_f \nabla g$ could be computed directly using the algorithm proposed by Pearlmutter [31]. However, his approach is based on special differential operators applied in an iterative manner. This would mean evaluating a whole iterative procedure every time a new function g_i is taken into account, resulting in a drastic loss of performance.

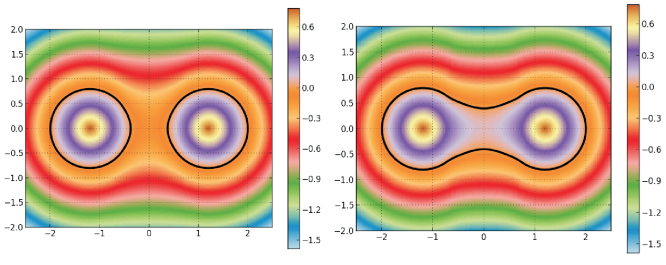

Fig. 5. Two atoms whose iso-contours are separated by a distance between R and $2R$. Using the constant radius R (Eq. 5), the two atoms will not blend. When applying the scaled radius (Eq. 8), the correct blended surface is obtained.

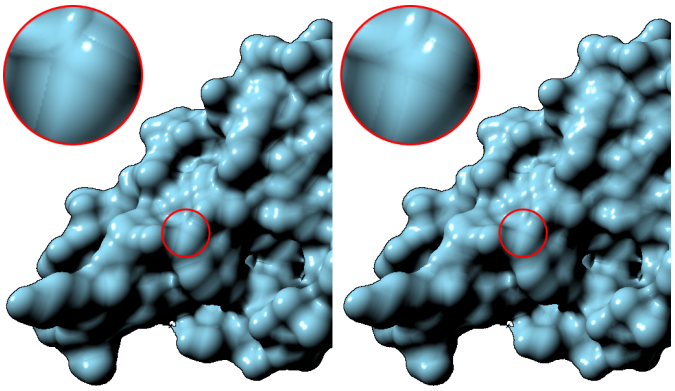

Fig. 6. Comparison of two different approximations. Left: second order derivatives are neglected. Right: second order derivatives are computed iteratively, while third order derivatives are neglected; this results in a smoother surface with less discontinuities.

4.2 Approximated Evaluation

Dealing with derivatives of arbitrary order is highly challenging, especially when real-time performance is a requirement. However, we do not need an exact evaluation of the function l: since thousands of atoms are shown simultaneously, the global structure of the molecule is much more relevant than the fine details. We can achieve different trade-offs between accuracy and performance (and memory consumption) simply disregarding the derivatives from a certain order on.

A first rough approximation can be obtained by neglecting the second order derivatives, i.e., setting $H_f(\mathbf{x}) = H_g(\mathbf{x}) = 0$. In practice, we are implicitly assuming that the gradients $\nabla f(\mathbf{x})$ and $\nabla g(\mathbf{x})$ do not change within an infinitesimal neighborhood. Consequently, their dot product $k(\mathbf{x})$ does not change as well, which means $\nabla k(\mathbf{x}) = 0$. The result obtained from this approximation can be seen in Figure 6 (left). The global structure of the molecule is well conveyed, and the typical blobby appearance of the Gaussian representation is avoided (compare with Figure 1 left).

The red circle in Figure 6 (left) shows a 3x magnification of a small area of the molecule. The connections between atoms present small discontinuities, which are due to the approximation scheme. Since higher magnification levels are sometimes required for specific tasks, such as cavity inspection and analysis, it is worth investigating better approximations.

Second order derivatives can be also computed. The Hessian H_g can be computed analytically according to Equation 3. In contrast, $H_f = H_{l_{i-1}}$ is the Hessian of our iterative function l, so it has to be computed iteratively as well. The components of the Hessian H_l are given by the second derivatives of l:

$$H_l^{ij} = \frac{d^2 l}{dx_i dx_j} = \frac{d}{dx_i}\left(\frac{dl}{dx_j}\right).$$

In practice, we have to derive once again the gradient ∇l, given by Equation 11. Applying again the chain rule, H_l can be rewritten as:

$$H_l = \frac{\partial l}{\partial f}H_f + \frac{\partial l}{\partial g}H_g + J^T \dot{H}_l J + \frac{\partial l}{\partial r}\frac{\partial^2 r}{\partial k^2}\nabla k(\nabla k)^T + \frac{\partial l}{\partial r}\frac{\partial r}{\partial k}H_k, \quad (13)$$

where J is the matrix obtained from function gradients taken column-wise

$$J = \left[\nabla f, \nabla g, \frac{\partial r}{\partial k}\nabla r\right],$$

and $\dot{H}_l$ is the symmetric matrix of second partial derivatives

$$\dot{H}_l = \begin{bmatrix} \frac{\partial^2 l}{\partial f^2} & \frac{\partial^2 l}{\partial f \partial g} & \frac{\partial^2 l}{\partial f \partial r} \\ . & \frac{\partial^2 l}{\partial g^2} & \frac{\partial^2 l}{\partial g \partial r} \\ . & . & \frac{\partial^2 l}{\partial r^2} \end{bmatrix} = \frac{1}{q^3}\begin{bmatrix} r^2 & -r^2 & -r(f-g) \\ . & r^2 & r(f-g) \\ . & . & (f-g)^2 \end{bmatrix}.$$

The second derivative of r is in our case

$$\frac{\partial^2 r}{\partial k^2} = \frac{\pi^2 R}{16}\sin\left(\frac{\pi}{4}k\right).$$

The last unknown variable in Equation 13 is the Hessian H_k of the gradients' dot product k. It can be computed by applying the derivation rules for sum and product to the right-hand side of Equation 12:

$$H_k = \frac{d}{d\mathbf{x}}(\nabla k) = \frac{dH_f}{d\mathbf{x}}\nabla f + \frac{dH_g}{d\mathbf{x}}\nabla g + 2H_f H_g,$$

which includes third order derivatives.

At this point, we can define a new approximation scheme by neglecting the third order derivatives. In practice, we compute the Hessian of k as $H_k = 2H_f H_g$. Figure 6 (right) shows a result obtained with this approximation: the surface has a smoother appearance and, even when magnified, no significant discontinuities can be detected.

Notice that the actual difference between this and the previous approximation scheme is the evaluation of Equation 13. We have implemented our technique on the GPU, which is particularly efficient in handling vector and matrix operations, therefore there is just a slight decrease in performance. But most importantly, the computational complexity of the algorithm is unchanged.

Considering also third or higher order derivatives may lead to even smoother results. However, deriving the necessary equation would be increasingly challenging, while the actual visual improvements would be smaller and smaller.

5 Visualization

We describe a method to visualize our molecular function instantly using the well-known A-buffer technique. Afterwards, we discuss the benefits for MD and also the potential for interactive MD.

5.1 Ray-casting

Since the molecular representation is evaluated on the fly, the rendering pipeline consists of several steps. In the first one, we render the van der Waals atoms as spheres with an increased radius that defines their area of influence (Fig. 7). This area is defined by the solvent diameter $2R$, i.e., each atom is rendered as a sphere with its van der Waals radius increased by $2R$. We do not perform sphere ray-casting, we just quickly splat spheres using billboarding [40]. The splatted spheres are not displayed, but stored in the so-called A-buffer [4]. For each pixel of the image space sphere, the correct entry and exit depths are computed and stored [39, 27]. The implementation exploits the recent shader extension that allows to simultaneously read and write data in

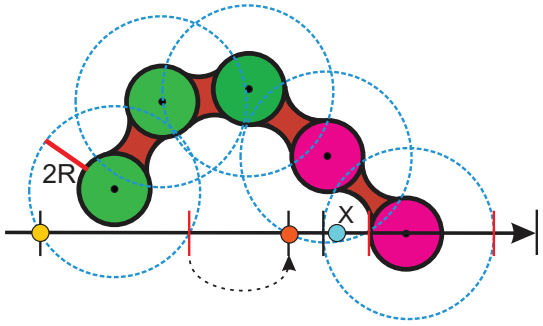

Fig. 7. An illustration of the ray-casting procedure based on the A-buffer. The entry and exit point of the extended spheres can be located along the ray (black and red lines). Therefore, at any location along the ray the set of influencing atoms $G_\mathbf{x}$ can be easily computed. For example, point X (blue point) is affected by the two pink atoms. Ray-casting starts from the first sphere along the ray (yellow point). When a point is in an area not affected by any atom, it is automatically shifted to the first unprocessed atom boundary along the ray (orange point).

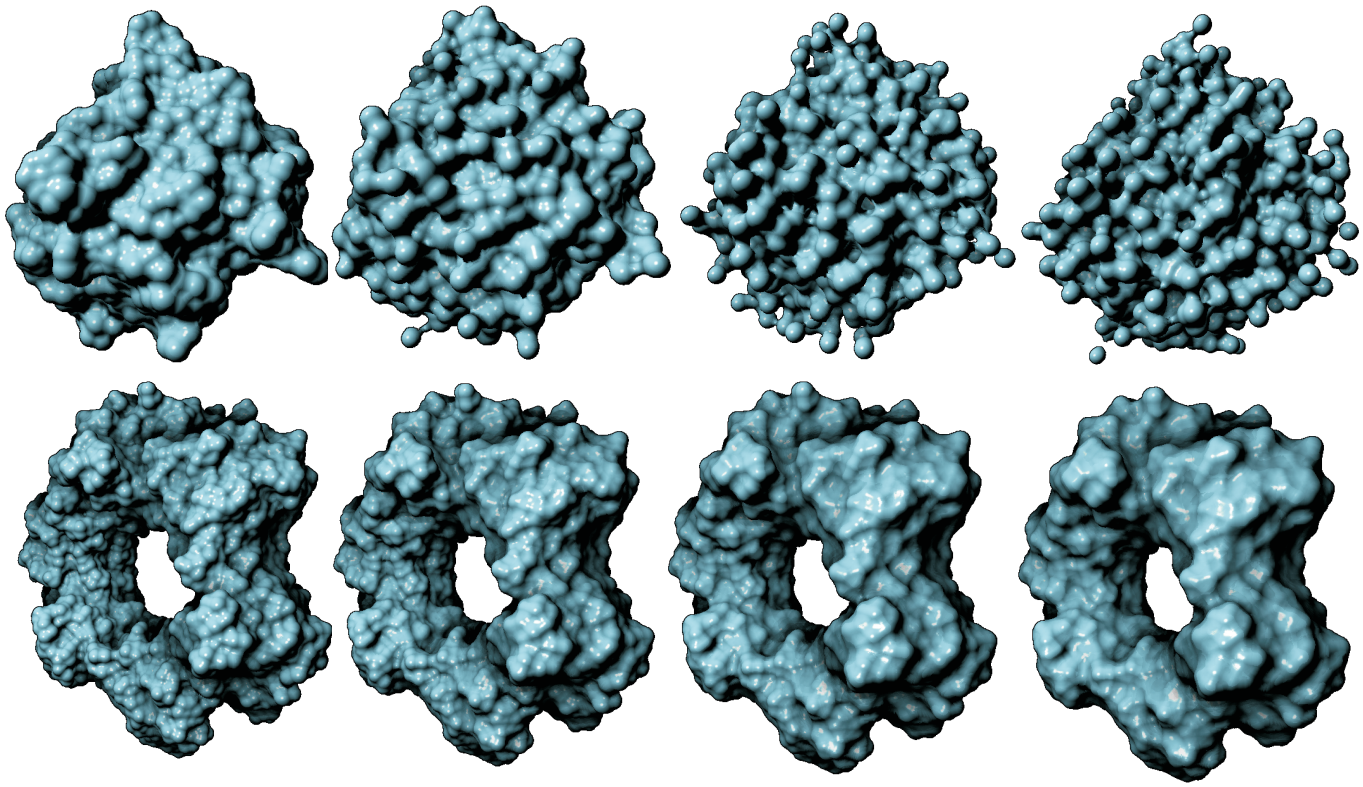

Fig. 8. Top: Illustration of instantaneous updates after the MD structure is changed. Interactive displacements of atoms demonstrates the interactive rendering capabilities of our approach. The overall displacement increases from left to right. The example is demonstrated on Proteinase 3 (3523 atoms). Bottom: Interactively changing the solvent radius on proliferatic cell nuclear antigen (12555 atoms). Our method allows quick adjustments of the solvent radius, from left to right: $R = 1.4\mathring{A}$, $R = 1.8\mathring{A}$, $R = 2.2\mathring{A}$ and $R = 2.6\mathring{A}$.

the fragment shader. Essentially, the A-buffer is a linked list of fragments generated for every individual pixel using atomic operations on the GPU [44]. Here, the limitation is the amount of graphics memory, since the linked lists can become very long for large datasets. This can be easily solved by avoiding rendering all the atoms at once, but rather render them in a slab-wise fashion.

In the second step, before the actual ray-casting, we sort the fragment records in ascending order of entry depth. This is a worthy investment, since, when looking for the ray-surface intersection, it lets us easily identify the relevant atoms (Fig. 7 vertical lines on the ray).

In the third step, the A-buffer is rendered. Here, a ray is cast for each image pixel, and an input 3D point $\mathbf{x}$ is generated according to the entry depth of the first sphere (Fig. 7, yellow point). Afterwards, in analogy with the sphere tracing algorithm [12], we process the ray in a step-wise fashion until either the iso-surface is hit, or the last sphere exit depth is reached. The step size is determined by the approximated distance function value $s = l(\mathbf{x})/|\nabla l(\mathbf{x})|$. Additionally, when the ray is in an area where no sphere of influence is present, it is automatically advanced to the first unprocessed sphere along the ray, i.e., the next one in the linked list (Fig. 7, orange point). This allows us to perform empty space skipping very efficiently.

The precision threshold used to detect the ray-surface intersection ($|l| \leq \varepsilon$) drives the overall performance of the rendering. It is crucial to specify the threshold value ε reasonably in order to maintain the surface details, especially for areas close to the viewer. On the other hand, we can afford a lower precision for areas far away from the viewer. Therefore we adjust ε adaptively according to the depth of the current point $\mathbf{x}$. We set $\varepsilon = \varepsilon_0 d(\mathbf{x})/(2c)$, where $d(\mathbf{x})$ is the ray depth of the point $\mathbf{x}$, c represents the camera distance to the projection plane and ε_0 is the overall iso-surface precision. In our demonstrations we specify $\varepsilon_0 = 0.1R$, where R is the solvent radius.

When the surface is hit, we compute the Phong shading model. Additionally, we compute silhouettes using edge-detection in image space. In the last step of our rendering pipeline, we add screen space ambient occlusion based on the method proposed by Luft et al. [25].

5.2 Molecular Dynamics

We demonstrate the potential of our approach to visualize the structures of MD simulations. We employ the Protein Data Bank (PDB) file format, which stores the protein information (e.g., atom types and initial positions). The MD trajectories of the atoms are stored in the *DCD* file format that is a standard in the Visual Molecular Dynamics (VMD) tool [13]. We studied two MD datasets, one containing only Proteinase 3 (Pr3, 3523 atoms) (Fig. 8 top) and one where Pr3 is bound to a lipid membrane (34490atoms) (Fig. 1). These datasets are composed by thousands of timesteps, therefore it can be difficult to identify the ones containing features of interest. In order to visualize a molecular surface, the proposed approach simply requires to upload the corresponding set of atoms to the GPU. Therefore, our system can be used as a tool for browsing through large temporal datasets (see the attached video). The only limitation here is given by the time required to upload the large amount of data to the GPU.

Thanks to the lack of pre-computations, our approach can be potentially integrated into steering systems for MD simulations. The area studying interactive steering of molecular dynamics is called *interactive molecular dynamics* [6]. The goal is to provide instant feedback from the MD simulation, so that a domain expert can interactively tune, for example, simulation parameters, the molecular context and the environment. We do not have access to such simulation systems yet, therefore, in order to imitate the changes of MD structures, we simply apply a random motion to each atom. Through a simple user interface, we can add a small displacement to the positions of the atoms. As shown in the enclosed video, the updated molecular representation is instantly visualized. An example illustrating the instant update of

the MD structure is shown in Figure 8 (top).

Our technique also allows to interactively configure the molecular representation. In MD applications, the solvent radius is typically set to approximate the size of the water molecule ($R = 1.4\AA$). Our method allows to change the radius on real-time basis. Typically, existing approaches have to firstly update underlying support structure, and then recompute the surface primitives, either by reduced surface [16] or Voronoi diagrams [22]. In our case, we just need to splat again the extended spheres using the updated solvent diameter $2R$. A demonstration of increasing the solvent radius is depicted in Figure 8 (bottom).

6 Quantitative and Qualitative Comparison

We compare our technique with Gaussian and SES models via visual comparison and in terms of frames per second (FPS). We also compute volumes and surface areas of all the three representations, in order to provide a quantitative comparison. We considered the following biomolecular structures: (a) Proteinase 3 bound to the lipid membrane (34490 atoms), (b) Proteinase 3 alone (3523 atoms), (c) Proliferatic cell nuclear antigen (12555 atoms), (d) Aquaporin (1852 atoms), (e) Immunoglobin (12530 atoms) and (f) Tubulin (14744 atoms).

6.1 Visual Comparison

Figures 1 and 10 showcase the three molecular surface models in a side-by-side visualization. Our technique is in the middle. The first noticeable difference is that the sharp surface features present in the SES model (Fig. 10 right) appear often smoothed in our model. This is a direct consequence of the definition of our implicit surface, since Equation 9 cannot represent sharp discontinuities. On the other side, compared to the Gaussian model, our representation approximates the solvent radius more closely. We discussed our results with computational biologists. The most relevant point they raised was that our visualization provides sufficient details with respect to binding site exploration. More specifically, the information conveyed by our model is equivalent to the one provided by the SES representation.

6.2 Performance

We evaluated the performance of our approach on several static structures and MD simulations. We implemented our framework in Python, utilizing the pyCUDA and PyOpenGL libraries for the implementation of the GPU-based ray-casting algorithm. In order to achieve better performance, we plan to migrate our framework to C++.

It is important to mention here that the existing rendering approaches [16, 22] can achieve higher FPS compared to our approach, since they rely on pre-computed auxiliary primitives that are then efficiently rendered. On the other hand, due to the simple volumetric representation, it is possible to voxelize our implicit function into a 3D scalar grid, which then can be efficiently rendered via marching cubes on the GPU, similarly to Krone et al. [19].

Nevertheless, we are able to achieve interactive frame rates via ray-casting even for very large proteins (Fig. 9). The performance measurements were carried out on a workstation equipped with two 2GHz processors, 12GB of RAM and a NVIDIA GeForce GTX 680 GPU. We implemented the Gaussian model [3] and the full SES representation [29] in the same framework, and we computed them on the same workstation. We evaluated the performance on six proteins, with the solvent radius set to the approximate size of the water molecule, i.e., $R = 1.4\AA$. Please note that the A-buffer is re-created every time the scene is rendered. The performance results, including the A-buffer formation, are summarized in the following table (in FPS):

Models\Molecules	(a)	(b)	(c)	(d)	(e)	(f)
SES	4	4	3	5	4	4
Our	9	12	10	10	11	10
Gauss	13	16	13	12	14	13

Note that our framework shows almost the same performance for either small (such as (e)) or large molecules (such as (d)). This can be explained by the actual size of the A-buffer. Some molecules can occupy the buffer evenly, while others can lead to unbalanced fragments

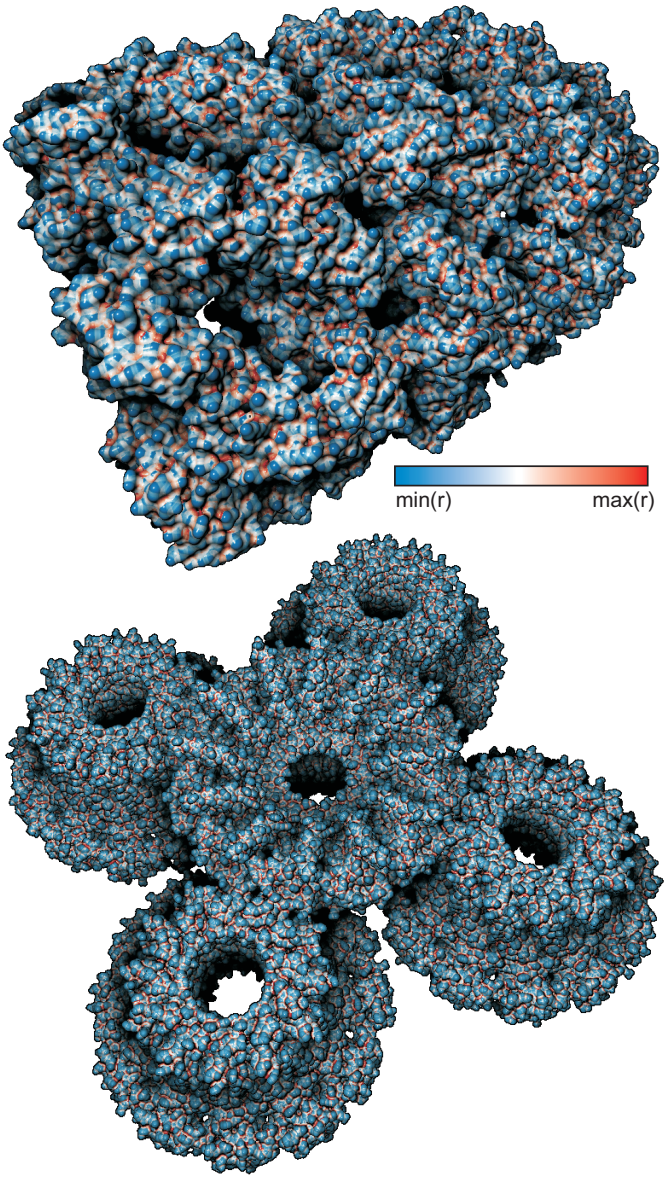

Fig. 9. Visualization of asymmetric chaperonin complex (58674 atoms) and bacteriophage containing 5 monomers (150720 atoms). Even for these large molecules, we are still able to achieve 9 and 4 FPS respectively. The colors represent the scaled radius r of the implicit circle.

distributions, causing bottlenecks for certain pixel areas. When compared to the Gaussian model, our technique is almost at the same level of performance. On the other hand, the improvement with respect to the implicit SES representation is at least by factor of 2.

Domain experts found this level of interactivity satisfactory. Although they have not been provided with the tool yet, they positively commented our demonstrations. They stated that our model has a strong potential with respect to the interactive update of MD structures and the rapid browsing through large temporal datasets. They compared our system with the tool they normally use, i.e., VMD. They found our models on the same qualitative level as the ones obtained with VMD. Moreover, in VMD, it is necessary to wait until the model is polygonized for each simulation time step.

6.3 Volume and Surface Areas

We performed a quantitative model verification by computing two global surface measures. Specifically, we computed the volumes and

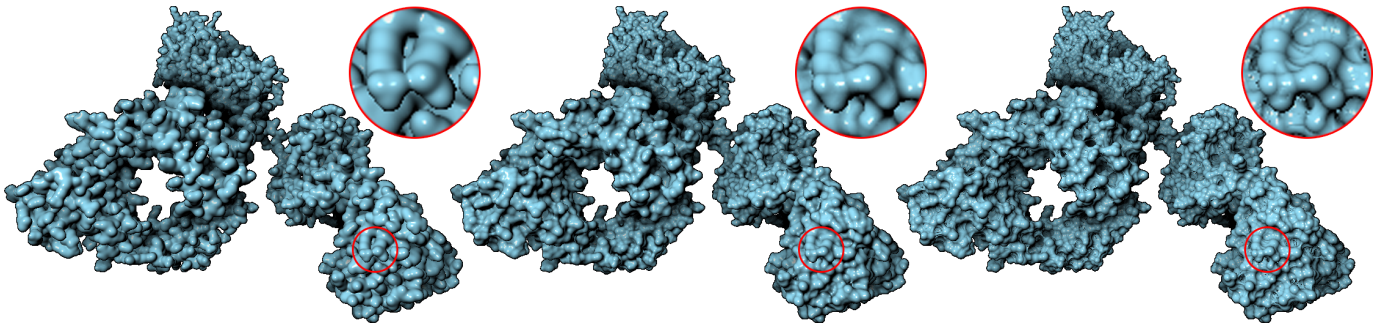

Fig. 10. Side-by-side comparison of the three surface models of immunoglobin. Our method (middle) resembles solvent excavation more closely than the Gaussian model (left). On the other hand, our representation is not as sharp as the SES (right).

the surface areas of the molecular representations for each of the three models. Note that the evaluation of local shape measures, such as the mean or Gaussian curvature, would require the definition of a molecular metric for the curvatures. To the best of our knowledge, there is no literature describing such a measure.

A possible way to compute volume and surface area is to convert the functional representation into a volumetric one. However, this would require a very high grid resolution in order to achieve the necessary level of detail. In the case of large molecules the resulting grid would be massive and consequently difficult to handle. Therefore we opted for an evaluation based on Monte-Carlo integration techniques. The basic idea is to approximate the volume and surface integrals by evaluating the implicit function in randomly placed sampling points. The points are normally generated within a bounded region, e.g., the bounding box of the molecule [26]. The estimated volumes and surface areas are summarized in the following tables:

Vol[$\AA^3$]	Gauss	Our	SES
(a)	1055454.04	1018930.58	1029501.77
(b)	93373.02	91135.80	91567.08
(c)	508396.12	503230.08	502167.23
(d)	122147.32	117167.49	117610.99
(e)	2125091.32	2096048.53	2102671.54
(f)	1951053.17	1895995.82	1898555.65

Surf[$\AA^2$]	Gauss	Our	SES
(a)	15372.75	10762.57	12574.02
(b)	1319.60	949.49	1077.82
(c)	4203.28	3375.89	3511.86
(d)	1725.94	1555.23	1699.44
(e)	12240.70	11018.95	11701.34
(f)	16558.48	13981.01	12875.31

We observe that our model is very similar to SES with respect to volumes. The surface area is instead slightly underestimated, but, on average, much closer to the SES than the Gaussian model.

7 Summary and Future Work

We proposed a conceptual framework for representing an approximation of the solvent excluded molecular surfaces. The core of our system is a blending operator defined by a circular model in implicit space. We compared our model qualitatively and quantitatively with the Gaussian and the SES representations. The comparisons showed that our technique is better than the Gaussian in approximating the SES model. At the same time, the rendering performance of our technique is much better than the SES and it approaches the same frame rates of the Gaussian.

The necessary mathematical foundations has been discussed and the resulting equations are described throughout the paper. Therefore, re-implementing our function evaluation procedure is straightforward.

One of the disadvantages is that our representation does not match any specific physical property of the model: while the SES correctly conveys the rolling of the solvent sphere, we just provide an approximation of the solvent radius.

As a next step, we believe that our model can be reformulated as a convolution-based approach. The related kernel would take into account the two areas of influence defined by $2R$ and by Equation 6. This will be subject of future investigations.

Our system allows for the immediate visualization of MD structures as molecular surfaces. This allows for a wide range of future extensions. As already discussed, we plan to integrate our technique into interactive molecular dynamics tools. This would let domain experts investigate potential binding sites while modifying the MD structure.

The functional representation can be easily sampled, as we showcased in Section 6.3. This property can be exploited in order to ease the detection and the analysis of potential binding sites [28]. We could also take advantage of the functional representation, for example, for developing a probe for $3D$ molecular exploration. A user could place the probe in the spatial domain and observe the actual distance of the probe from the molecular surface.

Due to the space limitation, we have not gone into details about different definitions of the scaling function r_k. A spline, for instance, would be a valid alternative, since it could be interactively edited through a transfer function widget, while the resulting surface is immediately displayed.

The implementation of clipping objects is straightforward. Essentially, it is sufficient to take into account only the atoms that lie inside (or outside) the clipping primitive. The clipping object could be displaced or modified, and the surface would be updated simultaneously.

There is also a strong potential in exploiting the computed function values, gradients and Hessian matrices. For example, they could be used to provide non photorealistic rendering effects such as the ones described by Kindlmann et al. [14].

Finally, notice that building blocks of our implicit circular model are the gradient and the Hessian of the function. Therefore, our technique can be easily extended to generic implicit objects, as long as their first and second partial derivatives can be computed.

Acknowledgments

We thank Nathalie Reuter for providing the molecular dynamics simulation datasets and the necessary feedback. We would also like to thank Paolo Angelelli, Ivan Kolesar and Mattia Natali for helping in fine-tuning the paper. Thanks to Helwig Hauser and the VisGroup in Bergen for the numerous useful discussions. Finally, we thank the anonymous reviewers for their valuable comments.

References

[1] L. Barthe, V. Gaildrat, and R. Caubet. Extrusion of 1d implicit profiles: Theory and first application. *International Journal of Shape Modeling*, 7:179–199, 2001.

[2] P. W. Bates, G. W. Wei, and S. Zhao. Minimal molecular surfaces and their applications. *Journal of Computational Chemistry*, 29(3):380–391, 2008.

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

[4] L. Carpenter. The a -buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph.*, 18(3):103–108, Jan. 1984.

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

[6] O. Delalande, N. Frey, G. Grasseau, and M. Baaden. Complex molecular assemblies at hand via interactive simulations. *Journal of Computational Chemistry*, 30(15):2375–2387, 2009.

[7] R. Durikovic and S. Czanner. Implicit surfaces for dynamic growth of digestive system. In *Shape Modeling International, 2002. Proceedings*, pages 111–117, 2002.

[8] P. Fayolle, A. Pasko, and B. Schmitt. Sardf: signed approximate real distance functions in heterogeneous objects modeling. In *Heterogeneous objects modelling and applications*, pages 118–141. Springer-Verlag, 2008.

[9] O. Gourmel, L. Barthe, M.-P. Cani, B. Wyvill, A. Bernhardt, M. Paulin, and H. Grasberger. A Gradient-Based Implicit Blend. *ACM Transactions on Graphics*, 2012.

[10] 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.

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

[12] J. C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1994.

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

[14] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Moller. Curvature-based transfer functions for direct volume rendering: methods and applications. In *Visualization, 2003. VIS 2003. IEEE*, pages 513–520, 2003.

[15] 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.

[16] 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.

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

[18] M. Krone, S. Grottel, and T. Ertl. Parallel contour-buildup algorithm for the molecular surface. In *Biological Data Visualization (BioVis), 2011 IEEE Symposium on*, pages 17–22, 2011.

[19] M. Krone, J. E. Stone, T. Ertl, and K. Schulten. Fast visualization of gaussian density surfaces for molecular dynamics and particle system trajectories. In *EuroVis 2012 Short Papers*, volume 1, 2012.

[20] R. E. Larson, R. P. Hostetler, and B. H. Edwards. *Calculus (with Analytical Geometry): 8th Edition*. Brooks Cole, 2005.

[21] 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.

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

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

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

[25] 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.

[26] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer Verlag, 2003.

[27] J. Parulek, T. Ropinski, and I. Viola. Seamless abstraction of molecular surfaces. In *Proceedings of the 29th Spring Conference on Computer Graphics*, SCCG '13, pages 120–127, 2013.

[28] J. Parulek, C. Turkay, N. Reuter, and I. Viola. Implicit surfaces for interactive graph based cavity analysis of molecular simulations. In *Biological Data Visualization (BioVis), 2012 IEEE Symposium on*, pages 115–122. IEEE, 2012.

[29] J. Parulek and I. Viola. Implicit representation of molecular surfaces. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis 2012)*, pages 217–224, March 2012.

[30] 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.

[31] B. A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Comput.*, 6(1):147–160, Jan. 1994.

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

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

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

[35] 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.

[36] A. Sherstyuk. Kernel functions in convolution surfaces: a comparative analysis. *The Visual Computer*, 15(4):171–182, 1999.

[37] 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.

[38] 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.

[39] L. Szecsi and D. Illes. Real-Time Metaball Ray Casting with Fragment Lists. pages 93–96, Cagliari, Sardinia, Italy, 2012. Eurographics Association.

[40] 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.

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

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

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

[44] J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz. Real-time concurrent linked list construction on the gpu. In *Proceedings of the 21st Eurographics conference on Rendering*, EGSR'10, pages 1297–1304, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.