Eurographics/ IEEE-VGTC Symposium on Visualization 2010
G. Melançon, T. Munzner, and D. Weiskopf
(Guest Editors)

*Volume 29 (2010), Number 3*

# Accelerated Visualization of Dynamic Molecular Surfaces

Norbert Lindow, Daniel Baum, Steffen Prohaska, and Hans-Christian Hege

Zuse Institute Berlin (ZIB), Germany

**Abstract**

*Molecular surfaces play an important role in studying the interactions between molecules. Visualizing the dynamic behavior of molecules is particularly interesting to gain insights into a molecular system. Only recently it has become possible to interactively visualize dynamic molecular surfaces using ray casting techniques.*
*In this paper, we show how to further accelerate the construction and the rendering of the solvent excluded surface (SES) and the molecular skin surface (MSS). We propose several improvements to reduce the update times for displaying these molecular surfaces. First, we adopt a parallel approximate Voronoi diagram algorithm to compute the MSS. This accelerates the MSS computation by more than one order of magnitude on a single core. Second, we demonstrate that the contour-buildup algorithm is ideally suited for computing the SES due to its inherently parallel structure. For both parallel algorithms, we observe good scalability up to 8 cores and, thus, obtain interactive frame rates for molecular dynamics trajectories of up to twenty thousand atoms for the SES and up to a few thousand atoms for the MSS. Third, we reduce the rendering time for the SES using tight-fitting bounding quadrangles as rasterization primitives. These primitives also accelerate the rendering of the MSS. With these improvements, the interactive visualization of the MSS of dynamic trajectories of a few thousand atoms becomes for the first time possible. Nevertheless, the SES remains a few times faster than the MSS.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—ray tracing, visible line/surface algorithms, algebraic surfaces J.3 [Life and Medical Sciences]: Biology and Genetics—molecular dynamics, molecular surfaces

## 1. Introduction

The function of a biomolecule is driven to a large extent by its 3-dimensional structure. While the chemical structure formed by the covalent bonds between the atoms of the molecule is relatively stable, the shape of a molecule changes rapidly. For proteins these changes are mainly restricted to the side chains of the amino acids, while the secondary structure formed by hydrogen bonds is preserved. Most proteins act as enzymes, which are activated by smaller molecules, called ligands. The ligands interact with the protein, thereby inducing changes in the geometry of the protein, which modify its function.

For more than two decades, several types of molecular surfaces have been used to study interactions between proteins and ligands. The most widely used type of molecular surface is the solvent excluded surface (SES). The molecular skin surface (MSS) is not yet used that often, but has a lot of potential and might become more important in the future. Until recently, surfaces of both types had to be triangulated to visualize them. With modern GPUs, fast direct visualization of these surfaces using ray casting has become possible. Krone et al. [KBE09] report interactive SES visualization of dynamics trajectories for a few thousand atoms. Interactive MSS rendering for such molecules is also possible [CLM08]. But the time required to construct the MSS prevented its use for dynamic trajectories of proteins and other macromolecules.

The interactive visualization of dynamic molecular surfaces requires a balanced combination of an efficient computation of the surface description and its rendering. The data transfer between CPU and GPU also needs to be considered if the surface is computed on the CPU and rendered on the GPU. In this paper, we propose several improvements that speed up the interactive visualization of dynamic molecular surfaces. We evaluate the suitability of the contour-buildup algorithm (Sect. 4.1) and an approximate Voronoi diagram


Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

algorithm (Sect. 4.2) for the parallel computation of molecular surfaces (Sect. 4.3). For the MSS, we define an atom neighborhood that results in a speedup of the surface computation by more than one order of magnitude. For rendering, we use tight-fitting bounding quadrangles to reduce the number of fragments to be considered for ray intersection (Sect. 5.2), and we employ geometry shaders to reduce the amount of data being sent to the GPU (Sect. 5.3). Note that we do not exploit coherency between subsequent frames, but recompute the whole surface except for partial updates, if these are possible.

The rest of the paper is structured as follows. We summarize related work (Sect. 2) and the different molecular surface types (Sect. 3), before we present our improvements to surface construction (Sect. 4) and rendering (Sect. 5). We report timings in Sect. 6 and discuss our findings in Sect. 7.

## 2. Related Work

### 2.1. Molecular Surfaces

In 1971, Lee and Richards [LR71] presented a program to draw the van der Waals (vdW) surface of a molecule. In the same article, they define the accessibility of atoms to a solvent. This definition became later known as the solvent accessible surface (SAS). In 1977, Richards [Ric77] defined the solvent excluded surface (SES), whose name was proposed by Greer and Bush [GB78] in 1978. In 1981, Connolly [Con83] presented the first algorithm for the computation of an analytical description of the SES, which was improved by Perrot et al. [PCG*92]. Varshney et al. [VJWW94] proposed a parallelized version based on the computation of an approximate Voronoi diagram. The reduced surface algorithm was presented by Sanner et al. [SOS96] and shortly after it was extended to partial updates for dynamic data [SO97]. At the same time, Totrov and Abagyan proposed the contour-buildup algorithm [TA96]. Besides articles describing the analytical surface computation of the SES, numerous publications deal with its triangulation, for example [Con85, ZXB07, RCK09]. Only recently, Krone et al. presented the first work to visualize the complete SES using ray casting [KBE09].

In 1999, Edelsbrunner [Ede99] proposed the skin surface for a set of weighted points. In the same paper, he describes an application to molecular surfaces, the molecular skin surface (MSS). Methods for triangulating the skin surface were, for example, presented by Cheng and Shi [CS04, CS05] and Kruithof and Vegter [KV07]. A recent paper by Chavent et al. [CLM08] describes ray casting of the MSS.

### 2.2. Ray Casting of Algebraic Surfaces

The MSS and the SES can both be decomposed into piecewise algebraic surfaces. These algebraic surfaces can be rendered using ray casting on the GPU using shader languages, such as GLSL. Ray casting of algebraic surfaces of degree two, called quadrics, has been shown to be very efficient [Gum03, KE04, RE05, SWBG06, dTLP07]. Ray casting algebraic surfaces of higher degree is more costly. Toledo et al. [dTLP07] evaluated several iterative methods for ray-intersection with cubics and quartics and compare their results to the analytical intersection computation used by Loop and Blinn [LB06]. While Toledo and Lévy [dTL08] use the iterative Newton-Raphson algorithm, which they found to be superior to Hart's sphere tracing [Har96], Singh and Narayanan [SN10] and Krone et al. [KBE09] successfully apply the analytic stabilized Ferrari-Lagrange method presented by Herbison-Evans [HE95].

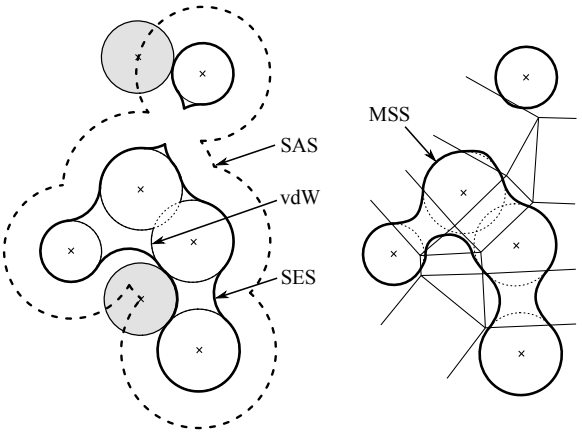


**Figure 1:** *Surface definitions in 2D. Left: The fine continuous line is the vdW surface. The SAS is represented by the thick dashed line and the SES by the thick continuous line. Two positions of the probe (gray) are depicted. Right: MSS with mixed complex.*

## 3. Molecular Surface Definitions

In this section, we mathematically define all molecular surfaces referred to in this paper. As the solvent excluded surface (SES) is closely related to the van der Waals (vdW) surface and the solvent accessible surface (SAS), we will define all three kinds of surfaces. A 2-dimensional sketch of these three surfaces is depicted in Fig. 1. Besides, we also define the molecular skin surface (MSS).

For all molecular surface definitions, we need the notion of a molecule. For the purpose of this paper, a molecule simply consists of a set of $n$ atoms. Each atom $i$, $1 \leq i \leq n$, is characterized by its position $p_i \in \mathbb{R}^3$ and its van der Waals (vdW) radius $r_i$. We denote the respective vdW spheres by $\sigma(p_i, r_i)$, or simply $\sigma_i$, $1 \leq i \leq n$.

### 3.1. Van der Waals (vdW) Surface

As the name suggests, the vdW surface is the boundary of the union of all vdW spheres $\sigma_i$, $1 \leq i \leq n$. At the intersections of the vdW spheres, the surface contains sharp edges and in

some places, deep crevices can be present (see Fig. 1, left). The vdW surface consists solely of convex spherical patches which are enclosed by contours consisting of circular arcs and complete circles lying on the vdW spheres.

### 3.2. Solvent Accessible Surface (SAS)

The SAS can be defined as the boundary of the extended vdW spheres $\sigma(p_i, r_i + r_p)$, $1 \leq i \leq n$, denoted by $\tilde{\sigma}_i$, where $r_p$ is the radius of a sphere approximating a solvent molecule, hence the name solvent accessible surface. We will refer to this sphere as 'probe sphere' and denote it by $\sigma_p = \sigma(p_p, r_p)$. In general, the smallest possible solvent is considered, which is water. The SAS can also be considered as the surface defined by all centers of the probe sphere while it touches at least one vdW sphere without intersecting any of them. Similarly to the vdW surface, the SAS consists solely of convex spherical patches which are enclosed by contours consisting of circular arcs and complete circles lying on $\tilde{\sigma}_i$.

### 3.3. Solvent Excluded Surface (SES)

We can describe the SES as the inward-facing surface of the probe sphere while its center lies on the SAS (see Fig. 2). We can properly define the SES as the boundary of the complementary volume of the union of all probe spheres that do not intersect any vdW sphere, that is, as the boundary of

$$\left\{ \mathbb{R}^3 \setminus \bigcup_{p_p \in \mathbb{R}^3} \sigma_p \mid \sigma_p \cap \sigma_i = \emptyset,\ \forall 1 \leq i \leq n \right\}.$$

The SES decomposes into surface patches of three different types: convex spherical, concave spherical, and toroidal (or saddle) patches. The convex spherical patches are composed of those points of the vdW surface that the probe sphere can touch without intersecting any vdW sphere. The concave spherical patches are formed when the probe simultaneously touches three or more atoms. Finally, the toroidal patches are formed when the probe sphere roles along two vdW spheres without touching or penetrating any other vdW sphere.

### 3.4. Molecular Skin Surface (MSS)

The skin surface [Ede99] is a smooth surface defined for a set of $n$ weighted points, with positions $q_i$ and weights $w_i$, and a shrink factor $s \in [0,1]$. One nice property of the skin surface is that it is tangent continuous at each point on the surface for $s \in (0,1)$. Furthermore, it is free of self-intersections. The molecular skin surface (MSS) is a skin surface where the vdW spheres $\sigma(p_i, r_i)$ determine the weighted points with $q_i = p_i$ and $w_i = {r_i}^2/s$, $s \in (0,1]$ [Ede99, KV04].

The molecular skin surface is defined based on a Voronoi diagram with distance functions $d_i(x) = ||x - q_i||^2 - w_i$ for the weighted points $i$. From the Voronoi diagram and its corresponding Delaunay triangulation, a mixed complex is computed, which depends on the shrink factor $s$. Let $V_i$ be the Voronoi region of the weighted point $i$ defined as

$$V_i := \left\{ x \in \mathbb{R}^3 \mid d_i(x) \leq d_j(x) : \forall j = 1, \ldots, n,\ j \neq i \right\}.$$

Furthermore, let $\nu_I := \bigcap_{i \in I} V_i$, with $I \subseteq \{1, \ldots, n\}$. Then $\nu_I$ is either empty or a convex polyhedron. For a non-empty polyhedron $\nu_I$, its dimension is $l = 4 - |I|$ and we can define its corresponding $k$-simplex, $k = 3 - l$, as $\delta_I := \text{conv}(\{q_i | i \in I\})$, where $\text{conv}(X)$ is the convex hull of $X$. We can now define a mixed cell $\mu_I(s) := s \cdot \nu_I + (1-s) \cdot \delta_I$ and the mixed complex of the molecular skin surface as

$$\text{Mix}(s) := \left\{ \mu_I \mid I \subseteq \{1, \ldots, n\} \right\}.$$

For $s = 0$, the mixed complex is equal to the Delaunay triangulation, while for $s = 1$, it is the Voronoi diagram. A 2-dimensional example of an MSS with its corresponding mixed complex is depicted in Fig. 1, right. The mixed complex $\text{Mix}(s)$ forms the basis to compute the molecular skin surface, which decomposes into piecewise quadratic patches of four different types: convex spherical patches contained in Voronoi regions, hyperbolic patches corresponding to Voronoi faces, hyperbolic patches corresponding to Voronoi edges, and concave spherical patches corresponding to Voronoi vertices. All patches are defined as the intersection of two scaled orthogonal affine hulls of spheres, called flats [Ede99].

Note that the choice of the weights $w_i$ for the molecular skin surface ensures that the radii of the convex spherical patches are independent of $s$ and equal to the vdW radii $r_i$.

## 4. Molecular Surface Construction

In this section, we describe two algorithms for the computation of molecular surfaces. We begin with the contour-buildup algorithm [TA96] for the SES. We then describe an approximate Voronoi diagram algorithm for weighted points. This algorithm was previously used for the computation of the SES [VJWW94]. We define an atom neighborhood for the MSS that allows adopting this algorithm for the computation of the MSS.

### 4.1. Contour-Buildup Algorithm for SES

For the computation of the SES, several algorithms have been proposed, some of which are mentioned in Sect. 2.1. We propose to use the contour-buildup algorithm [TA96]. It is very efficient and easy to parallelize, because the contour of each atom can be computed independently.

The contour-buildup algorithm computes the SES in three steps. First, it computes the SAS. From the SAS, all patches of the SES are computed (Fig. 2). These patches might still contain self-intersections, which are removed in a third step.

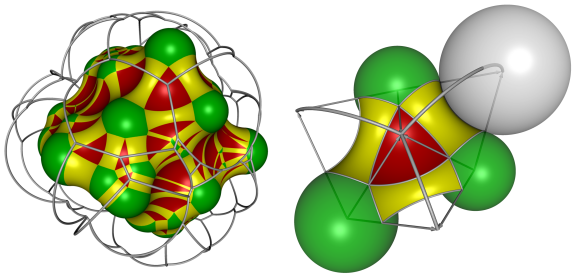

**Figure 2:** *SAS and SES. Left: The complete contours of the SAS and the SES are depicted. Each contour arc represents a saddle (or toroidal) patch (yellow), and each vertex, the point where three arcs meet, creates a concave spherical patch (red). Right: Detailed view of the contour. The gray sphere shows a position of the sphere.*

**Computation of SAS.** This part of the algorithm is the most involved one. It is illustrated in Fig. 3. For each of the extended vdW spheres $\sigma(p_i, r_i + r_p)$, the contour describing the part of the boundary of $\tilde{\sigma}_i$ contributing to the SAS has to be computed. As already mentioned in Sect. 3.2, all contours consist of circular arcs and full circles. The arcs and circles describing the contour of $\tilde{\sigma}_i$ are created by intersecting all neighboring spheres of $\tilde{\sigma}_i$, where the neighborhood $N_{r_p}(i)$ of $\tilde{\sigma}_i$ is defined as

$$N_{r_p}(i) := \{ j \mid ||p_j - p_i|| < r_j + r_i + 2r_p, 1 \leq j \leq n, j \neq i \}.$$

To quickly determine $N_{r_p}(i)$, we use a flexible 3-dimensional grid with at most $M$ grid cells, where $M$ is an integer number that should be chosen proportionally to the number of atoms (see Sect. 4.3). In each cell, the references to all atoms whose centers lie in the cell are stored. Since the atom density within a molecule is bound, the number of references in each cell can also be fixed. The fixed grid size $M$ and the fixed number of references per cell allow us to quickly adjust the grid to the changes of the molecule without re-allocating memory. The dimension of the grid is such that it encloses all positions $p_i$. The grid is then filled with $m$ equally sized grid cells, with $m \leq M$ (see Fig. 4).

In the first step, we compute for each $\tilde{\sigma}_i$ all intersection circles $c_{ij}$ with all $\tilde{\sigma}_j$, $j \in N_{r_p}(i)$. These circles are analyzed, and circles that do not contribute to the final contour are removed. In the second step, we iteratively compute for each $\tilde{\sigma}_i$ the contours from all remaining circles $c_{ij}$. For the details, we refer the reader to [TA96].

**Computation of SES.** As mentioned in Sect. 3.3, the SAS contains all information to compute the SES. The SAS contours simply need to be projected onto the respective vdW spheres. These projections will generate the SES contours (see Fig. 2). From each convex spherical patch of the SAS, a convex spherical patch of the SES is generated; from each contour arc of the SAS, a toroidal patch of the SES is generated; and from each end point of a contour arc, a concave spherical patch of the SES is generated.

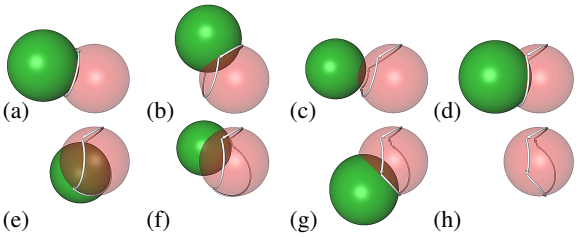


**Figure 3:** *Stepwise creation of the contour of the red atom. (a) Start contour (circle) created by the first neighboring atom. (b) The second circle splits the first circle, thereby creating two arcs. (c) Both arcs are cut by the third circle, one at the end point and one at the starting point. (d) The fourth circle deletes some arcs and cuts the starting and end points of one arc. (e, f, g) Analogously to (c). (h) Complete contour of the red atom.*

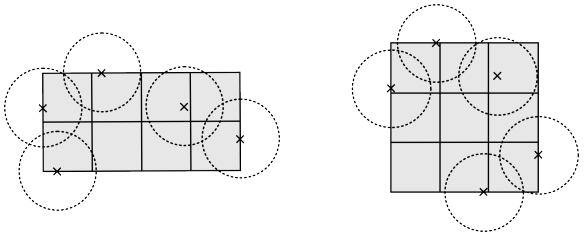

**Figure 4:** *Flexible 3-dimensional grid for neighborhood search. Two examples are given for different atom positions of the same molecule.*

The generated SES description might contain self-intersections, which arise in two cases. First, if the small circle of the torus is larger than its big circle, the torus intersects itself. When this self-intersection is removed, two cusps remain. This intersection can be easily identified. The intersection of two concave spherical patches is more costly to identify. Here, all concave spherical patches that are close enough to each other need to be tested for intersections. Similarly to the neighborhood search, these patches can be found using a 3-dimensional grid. In the original contour-buildup algorithm, these intersections were truly computed and the contours of the concave spherical patches were updated accordingly. Since we do not triangulate the analytical surface, for each concave spherical patch, we only store the clipping planes describing these self-intersections.

**Partial update of the SAS.** In protein-ligand docking simulations [BW09], the number of flexible atoms is often reduced to the amino acids in the active site of the molecule to reduce the computational cost. In this case, we only need to re-compute the contours of the flexible atoms $i$ together with their neighborhoods $N_{r_p}(i)$.

### 4.2. Approximate Voronoi Diagram for MSS

In this section, we show how the idea to approximate the Voronoi diagram to compute the SES [VJWW94] can be transferred to the computation of the MSS.

To compute the MSS, we first need to compute the Voronoi diagram for the weighted points $(q_i, w_i)$ with distance function $d_i(x) = ||x - q_i||^2 - w_i$, with $w_i = {r_i}^2/s$. Instead of computing the whole Voronoi diagram, we only compute feasible cells as suggested by Varshney et al. [VJWW94]. To properly define the feasible cell for the MSS, we first need to define the neighborhood $N_s(i)$ of an atom $i$ for the skin surface with shrink factor $s$. Let $\nu_I \neq \emptyset$, $|I| = 2$, be a Voronoi face. Furthermore, let $H_{i,j}$ be the half plane between the weighted points $i, j \in I$ defined by all points $x$ with $||x-q_i||^2 - w_i = ||x-q_j||^2 - w_j$. The orthogonal distance $d_i(H_{i,j})$ from $q_i$ to $H_{i,j}$ is given by

$$d_i(H_{i,j}) = \left| \frac{(q_j - q_i)^2 + w_i - w_j}{2 \cdot \|q_j - q_i\|} \right| .$$

Now we can define the neighborhood $N_s(i)$ as

$$N_s(i) := \left\{ j \mid s \cdot d_i(H_{i,j}) < r_i, 1 \leq j \leq n, j \neq i \right\} .$$

Then a feasible cell for weighted point $i$ can be defined as

$$F_i := \left\{ x \in \mathbb{R}^3 \mid d_i(x) \leq d_j(x) : \forall j \in N(i) \right\} ,$$

where $N(i) = N_s(i)$. $F_i$ is generally larger than $V_i$, because we consider less neighbors for the computation of $F_i$, thus less half planes restrict the cell. To correctly compute the MSS, we need to consider a weighted point $j$ for the computation of the feasible cell $F_i$ only if the scaled Voronoi face $\nu_I$ intersects the vdW sphere $\sigma_i$. This is true if the orthogonal distance $d_i(H_{i,j})$ scaled by $s$ is smaller than $r_i$. All weighted points $j$ for which this holds belong to $N_s(i)$ by definition.

Due to computing feasible cells instead of Voronoi cells, the algorithm scales linearly with the number of weighted points, because for physically correct molecules, the size of the neighborhood of the weighted points is bound by a constant. Each feasible cell can be computed independently. Hence, the algorithm can get trivially parallelized as was described by Varshney et al. [VJWW94].

To obtain the analytical description of the MSS, we need to compute the mixed complex for a specific shrink factor $s$ from the approximate Voronoi diagram. For each of the 0- to 3-dimensional Voronoi facets, we then generate the implicit function as described by Edelsbrunner [Ede99].

### 4.3. Implementation Details

We implemented parallelized versions of the contour-buildup algorithm and the approximate Voronoi diagram algorithm using OpenMP. To avoid memory allocation in the parallelized part of the algorithm, we estimate the necessary memory requirements and allocate the memory in advance. For the flexible neighborhood grid, we use $M = 2 \cdot n$.

In contrast to [VJWW94], we initialize each feasible cell with a tetrahedron large enough to enclose the molecule. We then iteratively intersect the feasible cell with the half planes $H_{i,j}$ of all neighbored atoms. All elements that are part of the initial tetrahedron are subsequently ignored.

## 5. Ray Casting of SES and MSS

We use ray casting for directly visualizing the molecular surfaces, similarly to Chavent et al. [CLM08] and Krone et al. [KBE09]. Since both MSS and SES are composed of piecewise algebraic surfaces, they are ideally suited for ray casting. Algebraic surfaces are defined by implicit functions that can be described by polynomials. For the MSS, the polynomials are of degree two, called quadrics. For the SES, the algebraic surfaces are of degree two and four. The latter are also known as quartics.

The ray casting performance of algebraic surfaces is mainly determined by two operations. First, the intersection of rays with the algebraic surface needs to be computed. Thus, the faster the intersections can be computed, the faster the surface can be rendered. Second, the number of rays that need to be tested for intersections is determined by the size of the rasterization primitives, because for each rasterized fragment, we need to test for intersection. The smaller these primitives are, the less rays need to be tested. However, a small number of rays will only be worthwhile if not too much time is spent for computing the primitives. In the following, we describe our choice of parameters concerning these two operations.

### 5.1. Ray Intersection

We compute the ray intersection for quadrics analytically. Details can be found, for example, in [SWBG06]. All spherical patches as well as the hyperbolic patches of the MSS can thus be rendered.

For intersecting a ray with the toroidal patch, we use sphere tracing [Har96]. We first compute the minimal bounding conic object of the toroidal patch and start the iterative search at the intersection of the ray with the cone (see Fig. 5). If the ray does not intersect the cone or if it intersects the conic object at the flat ends, the ray does not intersect the toroidal patch. Otherwise, we use the intersection point $x_0$ (see Fig. 5) with the conic object as starting point for sphere tracing. In each iteration step $i$, we compute the distance $l_i$ to the algebraic surface from the current point $x_i$. We then move forward along the ray to point $x_{i+1}$ by a step length $l_i$ and compute $l_{i+1}$. We terminate sphere tracing after a fixed maximum number of steps. If we are too far from the surface at this point, we ignore the ray. Otherwise, we take the last point as intersection point.

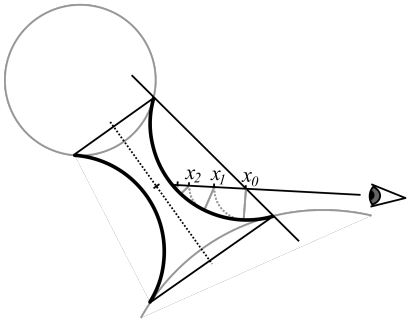


**Figure 5:** *Ray casting of saddle patches. First, we compute the intersection point with the bounding cone. We use this point as starting point for sphere tracing. The distance to the surface determines the step size.*

### 5.2. Rasterization Primitives

**SES.** Instead of rendering the convex spherical patches, we always render the whole vdW spheres. This produces correct results, because the part of the vdW sphere not contributing to the SES is completely enclosed by the surface. For each sphere, we use a square as rasterization primitive. This square is placed orthogonally to the line from the camera to the sphere center. The size of the square depends on the distance to the camera and the radius of the sphere.

For the other two kinds of SES patches, we also compute tight-fitting quadrangles. To achieve this, we first compute a bounding cylinder for each patch. This cylinder can be easily constructed for both kinds of patches. We then compute a tight-fitting quadrangle for each cylinder.

Ignoring the self-intersections, a concave spherical patch is a spherical triangle with vertices $v_1$, $v_2$, and $v_3$. We construct the bounding cylinder such that the cylinder radius is equal to the radius of the circumcircle $c$ of $v_1$, $v_2$, and $v_3$. The cylinder axis starts in the midpoint of $c$ and is orthogonal to the plane $H_{sph}$ spanned by $v_1$, $v_2$, and $v_3$. The cylinder length is equal to the probe radius $r_p$ (see Sect. 3.2) minus the orthogonal distance from the probe center $p_p$ to $H_{sph}$. For the toroidal patch, we compute the bounding cylinder such that its axis is parallel to the torus axis and the cylinder radius is set to the radius of the circumcircle of the projection of the toroidal patch onto the plane orthogonal to the torus axis. This is the minimal radius that can be achieved.

For the bounding cylinder, we determine the bounding quadrangle as follows (see Fig. 6). Let $H_{cyl}$ be the plane spanned by the cylinder axis and the vector from the camera position to the cylinder center. Then, we determine the four intersection points of $H_{cyl}$ with the cylinder caps. We need the right-most and left-most points, which are depicted by the red points in Fig. 6 (a) and (b). Through these two extremal points we span a trapezoid whose parallel sides are orthogonal to $H_{cyl}$. The lengths of these sides need to be chosen such that the trapezoid completely encloses the cylinder (Fig. 6 (c)). Fig. 6 (d) depicts the projected image of the cylinder and its projected quadrangle.

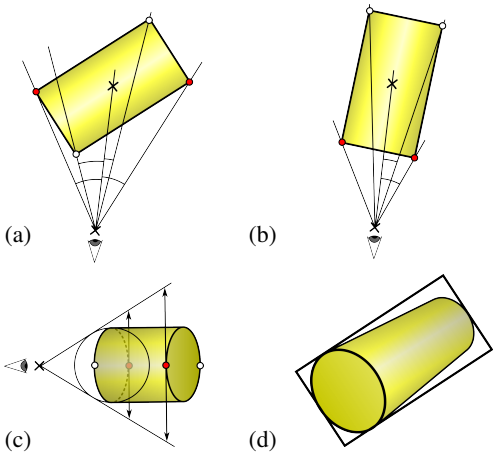


**Figure 6:** *Computation of a tight planar object that includes a cylinder after rasterization (d). In (a) and (b), two possibilities for the outer points of the cylinder are depicted. (c) displays how the outer points are used to span the object.*

**MSS.** For the convex spherical patches, we use the same bounding quadrangles as for the SES. For the hyperbolic patches corresponding to Voronoi faces, we use truncated pyramids. For the other two types of patches, we directly use the mixed cells. These cells are prisms with triangular base area and tetrahedra, respectively. Note that here is room for further optimization, but optimizing the rasterization primitives for the other three patch types is more complex.

### 5.3. Implementation Details

We store the information to be transferred to the GPU in vertex attributes, which are 4-dimensional vectors, and in a single texture.

**SES.** As mentioned before, we render a whole sphere for each convex spherical patch, hence we only need to send the sphere center and the radius, which together are stored in one vector. For the toroidal patch, we use 4 vectors. We store the start and end points of the toroidal axis together with the distances to the torus axis midpoint in the first two vectors. The third vector is used for the bounding cylinder. Since the cylinder axis may be dislocated from the torus axis, we store the translation of this axis and the cylinder radius. Furthermore, we store the small and big radii of the torus and two cone parameters, which are needed for the ray intersection. For the concave spherical patch, we need 4 vectors: one for the sphere center and three for the vertices. To remove the self-intersections, we store the clipping planes describing the self-intersection in a texture as done by Krone et al. [KBE09]. The shader program then tests each intersection point whether it is clipped by any clipping plane. To reduce the data transfer, we compute all bounding quadrangles

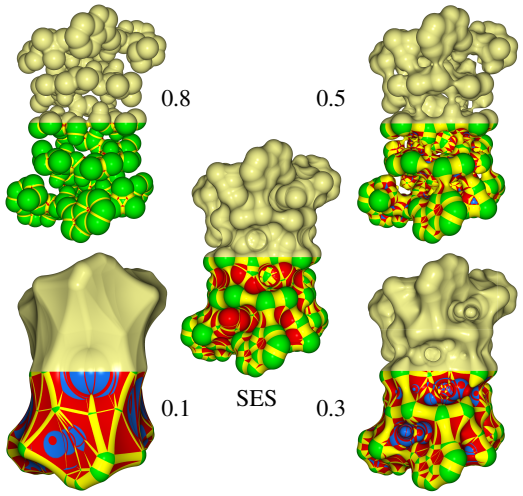


**Figure 7:** *Ray casting molecular surfaces of Gramicidin A (1GRM): SES with probe radius* 1.4 *(center) and MSS with different shrink factors. The patch colors denote different patch types. SES: green=convex spherical, red=concave spherical, yellow=toroidal. MSS: green=convex spherical, red=hyperbolic (Voronoi edges), yellow=hyperbolic (Voronoi faces), blue=concave spherical.*

on the GPU using geometry shaders. For sphere tracing, we use a maximum number of 30 iterations, whereby the actual number of iterations depends on the distance of the patch to the camera.

**MSS.** For each convex spherical patch, we render a whole sphere. For each hyperbolic patch, we transfer the axis, the hyperboloid's midpoint, a parameter describing the hyperboloid's shape, and the patch's bounding object. For each concave spherical patch, we transfer the sphere center, its radius, and the bounding tetrahedron.

## 6. Results

We tested our implementations on several molecules of different size. We used static as well as dynamic molecular data. The static molecules (see Tab.1, first row) were taken from the PDB [PDB]. They allow us to compare the results with previous publications. The dynamic data sets were provided by our collaborators and show the potential of our methods in real applications.

### 6.1. Surface Computation

In this section, we present the results for the parallelized versions of the contour-buildup algorithm (SES) and the Voronoi-based MSS algorithm. All tests were performed on an 8 core 2,53 GHz Intel system. Tab. 1 gives the timings for the molecular surface computation performed on 1 and 8 cores. For our algorithms, these timings include the computation of all patch information that was sent to the GPU.

| PDB | #Atoms | CB[1] | | AVD[1] (MSS) | | RS[2] |
|---|---|---|---|---|---|---|
| ID | | 1 | 8 | 1 | 8 | 1 |
| 1VIS | 2531 | 102 | 18 | 451 | 83 | 80 |
| 1AF6 | 10517 | 451 | 73 | 2113 | 369 | 360 |
| 1GKI | 20150 | 880 | 145 | 4184 | 698 | 770 |
| 1AON | 58870 | 2407 | 405 | 9924 | 1683 | 2680 |
| 3G71 | 99174 | 4488 | 759 | 18507 | 3353 | – |

[1] System: 2 Intel Xeon E5540 2.53 GHz. [2] System: Intel Core 2 Duo 3 GHz.

**Table 1:** *Update times in ms of the OpenMP versions with* 1 *and* 8 *cores. For the SES, we used the contour-buildup algorithm (CB) and for the MSS, the approximate Voronoi diagram (AVD) algorithm. The last column shows the update times for the reduced surface (RS) given in [KBE09].*

Using 8 cores, we measured a speedup of approximately 6 for the SES and 5 to 6 for the MSS. The MSS was computed for a shrink factor $s = 0.3$, because we found the MSS for this value to be most similar to the SES with a probe radius of 1.4 (see Fig. 7 and supplementary material).

The plots (a) and (c) in Fig. 8 show the speedup that we measured for the computation of the SES and MSS for molecule 1AON with $\sim$ 60k atoms. The plots contain the speedup for the overall computation and for the individual parts. Note that for the main part of the computation (the blue parts), we measured a speedup of more than 7 on 8 cores. The plots (b) and (d) in Fig. 8 show the proportionate time for each part of the algorithm.

Compared to the reduced surface (RS) algorithm [SOS96] used by Krone et al. [KBE09], the contour-buildup (CB) algorithm scales better with the number of atoms (see Tab. 1). While for small molecules the computational time is slightly higher for the CB algorithm, for the largest molecule, 1AON, the CB algorithm is faster than the RS algorithm. Note that the computational times for the CB algorithm include patch generation and data transfer to the GPU. Compared to the approximate Voronoi diagram (AVD) algorithm for SES, the CB algorithm was almost 1.3 times faster.

Chavent et al. [CLM08] report a computation time of 15 s for the MSS of the 1J4N [PDB] molecule. For our algorithm, we measured a computational time of 320 ms for the same molecule with $s = 0.3$ on a comparable single CPU core, hence the speedup is approximately a factor of 40.

### 6.2. Rendering

We measured the frame rates for five molecules (see Tab. 2). We used the same static molecules and the same GPU as Krone et al. [KBE09] and measured a speedup factor of at least 1.3 for the SES. Compared to our MSS rendering, the frame rates for the SES were higher by a factor of 2 to 4 for $s = 0.5$ and even higher for $s = 0.3$. We also compared our MSS rendering to MetaMol [CLM08]. On an NVIDIA GeForce 8800 GTX, we measured 30 frames per second

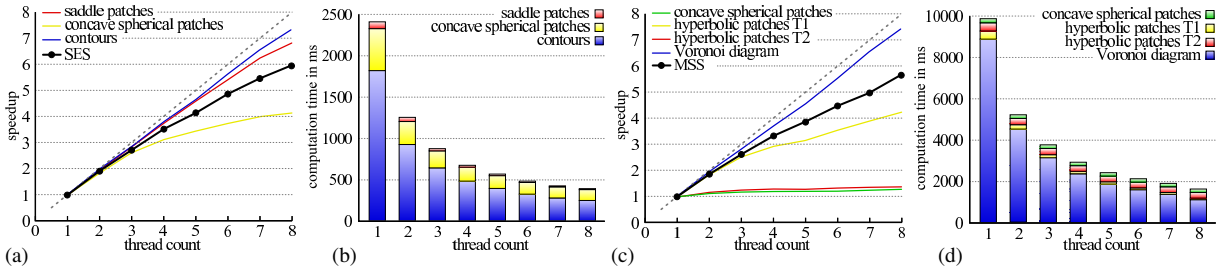


**Figure 8:** *Timings for SES and MSS of molecule 1AON on an 8 core 2,53 GHz Intel system. Diagrams (a), for SES, and (c), for MSS, show the speedup for a complete update of the surface depending on the number of threads. Diagrams (b) and (d) show the timings for the individual parts.*

| PDB ID | #Atoms | FR (%) | rendering performance (in fps) | | | |
|---|---|---|---|---|---|---|
| | | | SES | SES [KBE09] | MSS 0.5 | MSS 0.3 |
| 1VIS | 2,531 | 60 | 74 | 60 | 39 | 25 |
| 1AF6 | 10,517 | 70 | 46 | 27 | 14 | 10 |
| 1GKI | 20,150 | 70 | 26 | 19 | 8 | 6 |
| 1AON | 58,870 | 55 | 18 | 13 | 4 | 3 |
| 3G71 | 99,174 | 60 | 14 | – | – | – |

Graphics card: NVIDIA GeForce GTX280.

**Table 2:** *Rendering performance of static molecular surfaces. The table shows our SES results compared to the SES results of [KBE09] as well as our results for rendering of the MSS with shrink factor 0.5 and 0.3. The fill rate (FR) and the resolution (*$1024 \times 1024$*) are the same as in [KBE09].*

(fps) for the 1J4N molecule, compared to 7 fps in MetaMol [CLM08] on the same GPU. Thus, our MSS rendering is faster by a factor of approximately 4.

For both SES and MSS, we implemented typical colorization and simple transparency using blending. Furthermore, we implemented visualization techniques like depth darkening [LCD06] and silhouettes, which were also implemented by Krone et al. [KBE09]. The overhead introduced by these techniques is constant, because they use simple image filters. For example, for the more expensive depth darkening of 1AON, the performance dropped from 18 to 16 fps.

### 6.3. Dynamic Molecular Surfaces

We tested the system with real dynamic molecular data to evaluate how the combination of multi-threaded surface construction with rendering performs under load. To obtain the maximal performance, we used all 8 cores. The overall update rates are given in Tab. 3. Please also see the supplementary video.

In the simulation of DynMol-2, 500 of 4,500 atoms were flexible [BW09]. Including the neighborhoods of these atoms, we had to recompute the contours for 1,500 atoms (cf. Sect. 4.1). Using partial updates, we measured a speedup of 2.5 compared to re-computing the whole SES.

| Molecule | #Atoms | overall update rate (in fps) | | |
|---|---|---|---|---|
| | | SES | MSS (0.5) | MSS (0.3) |
| DynMol-1 | 1200 | 110 | 25 | 20 |
| DynMol-2 | 4500 | 33 | 7 – 8 | 5 |
| DynMol-3 | 6500 | 20 | 5 – 6 | 3 – 4 |

**Table 3:** *Overall update rates for dynamic molecules using OpenMP with 8 cores on a 2 Intel Xeon E5540 2.53 GHz system and an NVIDIA GeForce GTX280 graphics card.*

## 7. Discussion

We now discuss the performance improvements that we observed (see Tab. 4 for a summary).

| | construction | rendering |
|---|---|---|
| SES | 1 core: 1x to [KBE09]<br>8 cores: 6x to 1 core | >1.3x to [KBE09] |
| MSS | 1 core: 40x to [CLM08]<br>8 cores: 5x to 1 core<br>200x to [CLM08] | >3x to [CLM08] |
| SES to MSS | SES 4x to MSS | SES >3x to MSS |

**Table 4:** *Approximate speedups of SES and MSS compared to previous approaches, and speedup of SES over MSS.*

For the computation of the SES, the contour-buildup (CB) algorithm seems to be the method of choice. While it is trivial to parallelize, which is not obvious for the reduced surface algorithm [SOS96], it performs better than the approximate Voronoi diagram (AVD) algorithm [VJWW94]. Furthermore, the AVD algorithm has the drawback that it needs more memory, because it also stores additional data structures for the feasible cells. The computation of these additional data structures might also be the reason for its worse running time. Nevertheless, the AVD algorithm has the nice property that it can be used for the computation of both MSS and SES. The plots in Fig. 8 show that the patch computation limits scalability, while the CB and the AVD algorithms alone seem to scale well beyond 8 cores. The most likely reason for the limited scalability is the concurrent memory ac-

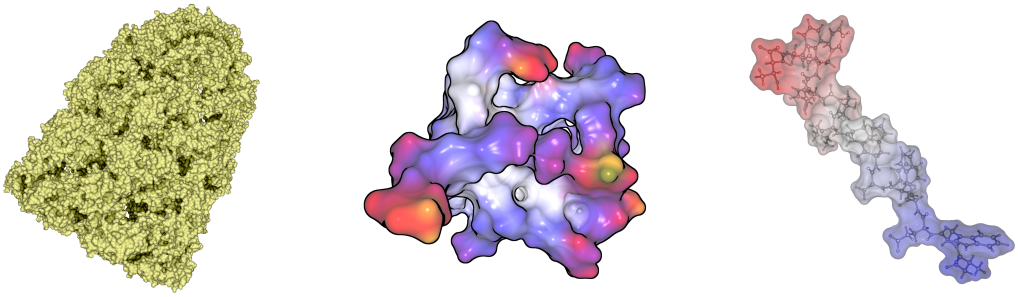

**Figure 9:** *Visualization techniques. In the left image, the SES of molecule 1AON is rendered with depth-dependent lighting. The middle image shows the colored MSS ($s = 0.3$) of molecule 1X5V with surface silhouettes. In the right image, the SES of molecule 2JQC is blended with its ball-and-stick representation.*

cess from all threads during patch creation. Note also that the patch creation for the MSS has not been fully parallelized.

We believe that using tight-fitting bounding quadrangles as rasterization primitives is the main reason for our improved SES rendering performance (1.3 times faster than [KBE09]). To confirm this, we also tested the point-based approach used by Krone et al. [KBE09] and measured frame rates similar to theirs, which supports our explanation. Note that we compute the bounding quadrangles in the geometry shader, so the data transfer to the GPU is the same for our bounding quadrangles and basic points. For the high computational load caused by sphere tracing in the fragment shader, tight-fitting bounding geometry seems to be superior to basic points, which confirms that "geometry shaders [offer] a viable option for the construction of bounding geometry" [GRE09]. We also tried the Newton-Raphson algorithm instead of sphere tracing and were able to further improve the rendering times of the toroidal patches by 10-20%. But we noticed a few pixel errors at the patch contours due to starting points that are too far from the intersection point. Hence, we recommend using sphere tracing [Har96] or the stabilized Ferrari-Lagrange method [HE95, KBE09]. Furthermore, we tested if storing the clipping planes in vertex attributes is more efficient than the texture-based approach [KBE09]. This is not the case. Moreover, using vertex attributes has the disadvantage that the number of vertex attributes is limited, so we recommend using a texture.

We see several reasons for the improvement in MSS rendering speed (3 times faster than MetaMol [CLM08]). First, we use tight-fitting bounding quadrangles for the convex spherical patches. Second, for the hyperbolic patches corresponding to the Voronoi faces, we use 3-dimensional polyhedra, which are possibly smaller than the mixed cells used in MetaMol [CLM08]. Furthermore, we remove empty mixed cells already on the CPU. Hence, we do not need to perform intersection tests for these cells.

Compared to the SES rendering, the MSS rendering is clearly more costly. We believe that this is mainly due to the larger number of patches of the MSS. For shrink factors of $s = 0.5$ and $s = 0.3$, the number of patches is approximately 4 times as large for the MSS. Moreover, we did not fully optimize the rasterization primitives of the MSS. Hence, further improvements in the rendering performance might be possible.

The results for dynamic data sets show that the overall update rates are limited by the surface construction. With our optimizations, data transfer to the GPU and rendering does not represent a bottleneck on an 8 core system.

## 8. Conclusion

We presented improvements for the computation and rendering of the solvent excluded surface (SES) and the molecular skin surface (MSS). We accelerated the MSS construction by two orders of magnitude. With these improvements, interactively visualizing the MSS of a molecule with a few thousand atoms has become possible for the first time. Nevertheless, in terms of computational and rendering speed, the SES remains superior to the MSS by a factor of approximately 4. Hence, we suggest using the SES for larger molecules.

A potential area of future work is the clipping of patches. Ignoring patches that lie completely inside the MSS could reduce the rendering time. Furthermore, correctly displaying a transparent SES requires to clip the convex spherical and the toroidal patches. In this case, preserving the rendering performance could be challenging, since the rendering of correctly clipped patches is more complex.

Coarse grained simulations present a further application area, where partial updates of the molecular surface computation might be rewarding. It would be interesting to investigate whether any speedup can be gained by applying such a strategy for this kind of data.

## Acknowledgments

We thank Marcus Weber and Bernd Kallies from Zuse Institute Berlin (ZIB) for providing the dynamic molecular data sets. Furthermore, we thank the anonymous reviewers for their helpful comments.

## References


[BW09] BUJOTZEK A., WEBER M.: Efficient simulation of ligand-receptor binding processes using the conformation dynamics approach. *Journal of Bioinformatics and Computational Biology 7*, 5 (2009), 811–831.

[CLM08] CHAVENT M., LÉVY B., MAIGRET B.: High quality visualization of molecular skin surface. *Journal of Molecular Graphics and Modelling 27*, 2 (2008), 1391–1398.

[Con83] CONNOLLY M. L.: Analytical molecular surface calculation. *Journal of Applied Crystallography 16*, 5 (1983), 548–558.

[Con85] CONNOLLY M. L.: Molecular surface triangulation. *Journal of Applied Crystallography 18* (1985), 499–505.

[CS04] CHENG H.-L., SHI X.: Guaranteed quality triangulation of molecular skin surfaces. In *Proceedings of IEEE Visualization* (2004), pp. 481–488.

[CS05] CHENG H.-L., SHI X.: Quality mesh generation for molecular skin surfaces using restricted union of balls. In *Proceedings of IEEE Visualization* (2005), pp. 399–405.

[dTL08] DE TOLEDO R., LÉVY B.: Visualization of industrial structures with implicit GPU primitives. In *Proceedings of the International Symposium on Visual Computing (ISVC) ) - Lecture Notes in Computer Science* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 139–150.

[dTLP07] DE TOLEDO R., LÉVY B., PAUL J.-C.: Iterative methods for visualization of implicit surfaces on GPU. In *Proceedings of the International Symposium on Visual Computing (ISVC) - Lecture Notes in Computer Science* (2007).

[Ede99] EDELSBRUNNER H.: Deformable smooth surface design. *Discrete & Computational Geometry 21*, 1 (1999), 87–115.

[GB78] GREER J., BUSH B.: Macromolecular shape and surfacemaps by solvent exclusion. *Proceedings of the National Academy of Sciences USA 75* (1978), 303–307.

[GRE09] GROTTEL S., REINA G., ERTL T.: Optimized data transfer for time-dependent, GPU-based glyphs. In *Proceedings of IEEE Pacific Visualization Symposium* (2009), pp. 65–72.

[Gum03] GUMHOLD S.: Splatting illuminated ellipsoids with depth correction. In *Proceedings of Vision, Modeling, and Visualization (VMV)* (2003), pp. 245–252.

[Har96] HART J. C.: Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10 (1996), 527–545.

[HE95] HERBISON-EVANS D.: *Graphics Gems V*. Academic Press, 1995, ch. Solving Quartics and Cubics for Graphics, pp. 1–15.

[KBE09] KRONE M., BIDMON K., ERTL T.: Interactive visualization of molecular surface dynamics. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1391–1398.

[KE04] KLEIN T., ERTL T.: Illustrating magnetic field lines using a discrete particle model. In *Proceedings of Vision, Modeling, and Visualization (VMV)* (2004), pp. 387–394.

[KV04] KRUITHOF N., VEGTER G.: Approximation by skin surfaces. *Computer-Aided Design 36*, 11 (2004), 1075–1088.

[KV07] KRUITHOF N., VEGTER G.: Meshing skin surfaces with certified topology. *Computational Geometry 36*, 3 (2007), 166–182.

[LB06] LOOP C., BLINN J.: Real-time GPU rendering of piecewise algebraic surfaces. *ACM Transactions on Graphics 25*, 3 (2006), 664–670.

[LCD06] LUFT T., COLDITZ C., DEUSSEN O.: Image enhancement by unsharp masking the depth buffer. *ACM Transactions on Graphics 25*, 3 (2006), 1206–1213.

[LR71] LEE B., RICHARDS F. M.: The interpretation of protein structures: Estimation of static accessibility. *Journal of Molecular Biology 55*, 3 (1971), 379–380.

[PCG*92] PERROT G., CHENG B., GIBSON K. D., VILA J., PALMER K. A., NAYEEM A., MAIGRET B., SCHERAGA H. A.: MSEED: a program for the rapid analytical determination of accessible surface areas and their derivatives. *Journal of Computational Chemistry 13*, 1 (1992), 1–11.

[PDB] Protein Data Bank. http://www.pdb.org.

[RCK09] RYU J., CHO Y., KIM D.-S.: Triangulation of molecular surfaces. *Computer Aided Design 41*, 6 (2009), 463–478.

[RE05] REINA G., ERTL T.: Hardware-accelerated glyphs for mono- and dipoles in molecular dynamics visualization. In *Proceedings of EuroVis* (2005), pp. 177–182.

[Ric77] RICHARDS F. M.: Areas, volumes, packing, and protein structure. *Annual Review of Biophysics and Bioengineering 6*, 1 (1977), 151–176.

[SN10] SINGH J. M., NARAYANAN P. J.: Real-time ray tracing of implicit surfaces on the GPU. *IEEE Transactions on Visualization and Computer Graphics 16* (2010), 248–260.

[SO97] SANNER M. F., OLSON A. J.: Real time surface reconstruction for moving molecular fragments. In *Proceedings of the Pacific Symposium on Biocomputing, Maui* (1997), pp. 385–396.

[SOS96] SANNER M. F., OLSON A. J., SPEHNER J.-C.: Reduced surface: An efficient way to compute molecular surfaces. *Biopolymers 38*, 3 (1996), 305–320.

[SWBG06] SIGG C., WEYRICH T., BOTSCH M., GROSS M.: GPU-based ray-casting of quadratic surfaces. In *Proceedings of the Eurographics Symposium on Point-Based Graphics* (2006), pp. 59–65.

[TA96] TOTROV M., ABAGYAN R.: The contour-buildup algorithm to calculate the analytical molecular surface. *Journal of Structural Biology 116*, 1 (1996), 138–143.

[VJWW94] VARSHNEY A., JR. F. P. B., WILLIAM J., WRIGHT W. V.: Linearly scalable computation of smooth molecular surfaces. In *IEEE Computer Graphics and Applications* (1994), vol. 14, pp. 19–25.

[ZXB07] ZHAO W., XU G., BAJAJ C.: An algebraic spline model of molecular surfaces. In *Proceedings of the Symposium on Solid and Physical Modeling (SPM)* (New York, NY, USA, 2007), ACM, pp. 297–302.