CGI2017 manuscript No.
(will be inserted by the editor)

Interactive GPU-based Generation of Solvent Excluded Surfaces

Pedro Hermosilla · Michael Krone · Victor Guallar · Pere-Pau Vázquez · Àlvar Vinacua · Timo Ropinski

Abstract The Solvent Excluded Surface (SES) is a popular molecular representation that gives the boundary of the molecular volume with respect to a specific solvent. SESs depict which areas of a molecule are accessible by a specific solvent, which is represented as a spherical probe. Despite the popularity of SESs, their generation is still a compute-intensive process, which is often performed in a pre-processing stage prior to the actual rendering (except for small models). For dynamic data or varying probe radii, however, such a pre-processing is not feasible as it prevents interactive visual analysis. Thus, we present a novel approach for the on-the-fly generation of SESs, a highly parallelizable, grid-based algorithm where the SES is rendered using ray-marching. By exploiting modern GPUs, we are able to rapidly generate SESs directly within the mapping stage of the visualization pipeline. Our algorithm can be applied to large time-varying molecules and is scalable, as it can progressively refine the SES if GPU capabilities are insufficient. In this paper, we show how our algorithm is realized and how smooth transitions are achieved during progressive refinement. We further show visual results obtained from real world data, and discuss the performance obtained, which improves upon previous techniques in both the size of the molecules that can be handled and the resulting frame rate.

Keywords Molecular Visualization

P. Hermosilla, P.-P. Vázquez, À. Vinacua
Visualization, Virtual Reality and Graphics Interaction (ViRVIG), Universitat Politècnica de Catalunya, Spain

M. Krone
Visualization Research Center, University of Stuttgart, Germany

V. Guallar
Barcelona Supercomputing Center, Spain

T. Ropinski
Research Group Visual Computing, Ulm University, Germany

1 Introduction

When analyzing molecular structures, derived surfaces often are taken into account, as they represent the accessibility of a molecule with respect to binding ligands. A frequently used surface definition is the Solvent Excluded Surface (SES) [21], which gives the boundary of the solvent-excluded molecular volume with respect to a specific solvent. Accordingly, SESs provide means for analyzing potential binding sites, as they depict which areas of a molecule are accessible by a specific solvent. Due to the importance of SESs, several algorithms that facilitate their computation have been proposed (see Section 2).

Since the SES is very well suited for analyzing the molecular interface and cavities, its use is widespread in facilitating understanding molecular dynamics (MD) data. An MD simulation can for example calculate the behavior of a molecule and a ligand, and outputs a set of consecutive spatial atom configurations (trajectories). The steps in these trajectories are commonly known as *frames*. Due to the complexity of the SES computation process, it is often performed in a preprocessing stage, rather than the mapping stage of the visualization pipeline (except for small models). While this allows for SES generation for static molecules and fixed probe radii, preprocessing is not feasible for dynamic setups, where the atom positions change over time, or the probe radius is altered during a visual analysis. To allow for an interactive visual analysis of large, dynamic molecular data, our aim was an on-the-fly generation of the SES for each frame.

We propose a novel SES computation algorithm that exploits modern GPUs to support the interactive generation of SESs. The algorithm has been developed specifically for dynamic data. We further do not assume or rely on any particular frame ordering, since modern systems usually generate many of those configurations at once in parallel. To achieve interactive SES updates, we have used a grid-based

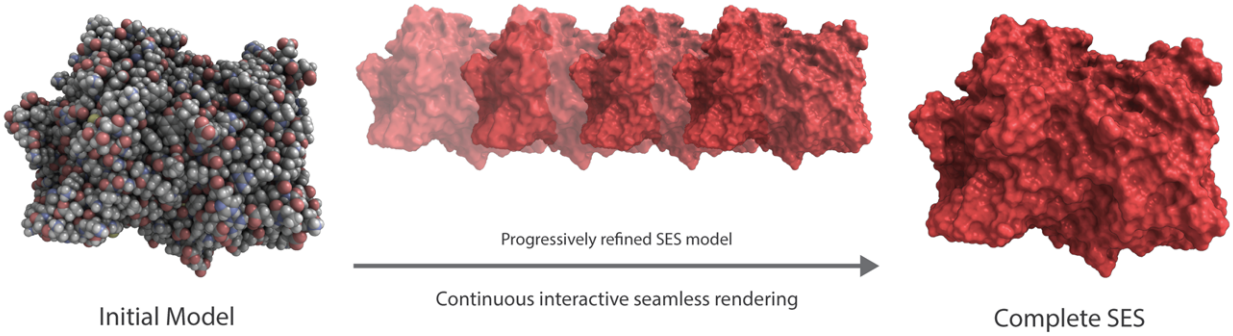

Fig. 1 Illustration of the SES generation algorithm. To maintain interactivity, if GPU capabilities are not enough to generate the full SES in a single frame, a coarse model is generated first, which is then progressively refined several steps. Progressive refinement occurs in a seamless way, and the user can explore the molecule meanwhile.

approach, which is suitable for GPU implementation. The grid is used to facilitate computation of the two phases of our algorithm. In phase one, intersection points between the probe and the molecule are computed, while the resulting distances are computed in the second phase. Based on the results, we are able to generate the actual SES. To make our algorithm scalable, it has been designed such that for large molecules, where real-time frame rates cannot be achieved at full detail, a progressive update of the SES is supported (see Figure 1). Thus, our contributions are as follows:

- Interactive SES generation without any preprocessing
- Progressive SES refinement to support full scalability
- Smooth transition between different levels of detail

2 Previous Work

As mentioned above, the idea behind the SES is to show the surface of a molecule with respect to a certain solvent. That is, everything within the surface is not reachable by this solvent. The solvent is represented by a so-called *probe sphere* with an appropriate radius (e.g., water is usually represented by a sphere of radius 1.4 Å). The SES was first described by Richards [21] as *Smooth Molecular Surface*. It can be defined as the surface that is traced out by the spherical probe rolling over the van der Waals (vdW) surface of the molecule (see Figure 2). The vdW surface is the union of spheres for all atoms where each sphere is centered at the location of the corresponding atom and has a radius equal to the vdW radius of the respective chemical element. Greer and Bush [7] gave an alternative definition of the SES: The SES is the topological boundary of the union of all possible probes that do not intersect any atom of the molecule. Their work also coined the term *Solvent Excluded Surface*.

Over the last four decades, many methods to compute and visualize the SES were developed. The SES can be analytically described as a set of patches: convex spherical patches (the remaining parts of the vdW surface), concave spherical triangles, and saddle-shaped toroidal patches (see Figure 3). Shortly after Richards [21] defined the SES, Connolly found the analytical equations to compute the SES patches [4]. Connolly's work lead to the development of several accelerated methods in the mid-90s that compute the SES analytically: Varshney et al. [26] developed a parallelizable algorithm based on Power Diagrams. Similarly, α-shapes can be used to compute the SES [5]. The Contour-Buildup method by Totrov and Abagyan [25] extracts the SES patches from the contours of the Solvent Accessible Surface [21]. Sanner et al. [22] developed the Reduced Surface algorithm. Due to the nowadays widespread availability of multi-core CPUs and programmable GPUs, optimized parallel implementations of some of these algorithms were recently presented: Lindow et al. [18] parallelized the Contour-Buildup for multi-core CPUs. Krone et al. [15] adapted the same algorithm for the GPU. These methods were the first ones that allowed to compute the SES interactively for larger molecules (up to 10k atoms) and are to date still among the fastest ways to compute the SES analytically. Both methods render the patches of the SES using GPU-based ray casting in the fragment shader as proposed by Krone et al. [14]. Jurcik et al. [12] extended the method of Krone et al. [15] to support transparent rendering of the SES. A detailed review of SES visualizations was given by Kozlikova et al. [13].

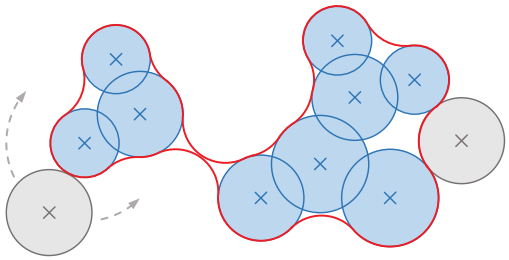

Fig. 2 The SES (red) can be defined analytically as the surface that is traced out by a spherical probe (gray, shown in two sample positions) rolling over the vdW surface of the atoms (blue) of a molecule.

Besides the analytical algorithms, which are computationally very involved, Kozlikova et al. [13] identified a sec-

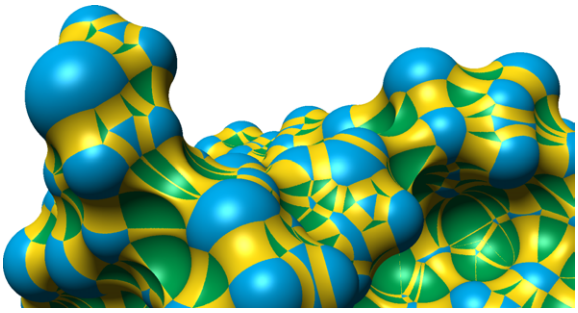

Fig. 3 SES colored by the analytical patches that form the surface. Blue: convex spherical patches, green: concave spherical triangles; yellow: toroidal patches. Image generated with MegaMol [8].

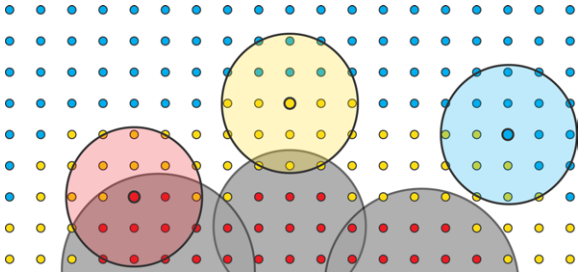

Fig. 4 Step 1: Probe intersection. The figure shows how the tests at the grid points are used to classify them. Red points are classified as *interior* to the SES, blue ones are classified as *exterior* to the SES and yellow ones as points on the *boundary* of the SES.

ond class of algorithms that compute the SES. These methods discretize the space around a molecule, that is, they compute the SES based on a three-dimensional grid. Usually, each voxel of the grid is first classified as within or outside of the surface. Subsequently, an isosurface that matches the SES can be extracted from the grid. While the resulting surfaces are less accurate than the analytical descriptions, these algorithms have the advantage that they are are fast since the computations are less involved, and that they are relatively simple to implement. Can et al. [3] developed a method based on level sets. Yu [28] presented an efficient algorithm using lists to speed up the computations. EDTSurf by Xu and Zhang [27] extracts high-quality SES meshes based on Euclidean distance transformation.

Molecular surface descriptions that are related to the SES are the Ligand Excluded Surface (LES) presented by Lindow et al. [17] and Gaussian surfaces (*Metaballs*) introduced by Blinn [2]. The LES can be seen as a generalization of the SES. Here, the ligand is not represented by a spherical probe but by the vdW representation of the actual atoms. Lindow et al. presented a grid-based algorithm to compute the LES. Gaussian surfaces can be used to approximate the SES. A very fast grid-based algorithm to compute Gaussian surfaces of very large molecules on the GPU was presented by Krone et al. [16]. A comparison between SES, LES, and Gaussian surfaces can be found in the report by Kozlikova et al. [13].

3 Algorithm

As mentioned above, Kozlikova et al. [13] classified the algorithms to compute the SES in two main categories: the ones that compute an analytical representation of the surface and the ones that compute the surface by discretizing the space around the molecule. Our algorithm falls in the second category, as we use a regular 3D grid that represents a signed distance field to the SES. Despite the resulting memory consumption requirements, this representation allows us to easily compute a coarse representation of the SES in real-time, to refine this coarse representation progressively, and to create a smooth transition between different detail levels.

The work-flow of our algorithm is the following: When the application receives a new frame of the simulation we compute a signed distance field to the SES using a low-resolution grid. This computation is carried out in milliseconds due to the low resolution of the grid, so the user does not perceive any drop in performance. This coarse representation is immediately rendered and shown to the user. Meanwhile, the algorithm computes refined versions in the background by increasing the resolution of the grid. Once a new level is computed, the algorithm performs a smooth transition between the current level and the one just computed.

3.1 SES Computation

Our algorithm to compute the SES is based on the one by Lindow et al. [17]. It generates a 3D signed distance field with positive values outside the SES and negative values inside. Similar to Lindow et al., we limit the range of the distances, that is, we only have to compute the exact distances in the proximity of the SES. In our case, this range is $[-r_g, r_p]$, where r_g is the distance between two neighbor grid points and r_p is the probe radius.

Our algorithm, unlike the one proposed by Lindow et al. [17] for the LES, has no collision problems and it can be executed in parallel without using synchronization mechanisms, making it suitable for GPU implementations. This is accomplished by dividing the computation into two distinct steps: *probe intersection* and *distance field refinement*.

Probe intersection: In the first step of the algorithm, the points of the grid are classified as points located outside the SES, inside the SES, or on the boundary of the SES. This classification uses two simple tests: checking both the center of the grid point and a probe located at the center of the grid point for intersections with the atoms of the molecule (using the vdW radius). An illustration of these two tests is shown in Figure 4. The interpretation of these tests is:

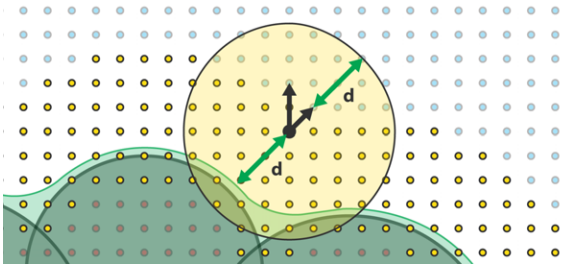

Fig. 5 Step 2: Distance field refinement. For each yellow point the algorithm searches for the closest blue point in a neighborhood. The distance to the SES is then computed as the difference between the probe radius and the distance between these two points.

Outside SES: If there is no intersection between the probe and the atoms, the grid point is classified as a point outside the SES and the algorithm assigns r_p as the distance to the SES (blue points in Figure 4).

Inside SES: If the distance between the grid point and at least one atom of the molecule is less than the radius of this atom minus r_g (the distance between two closest neighbors on the grid), the point is classified as an interior point of the SES (red points in Figure 4). In this case, the algorithm assigns $-r_g$ as its distance to the SES.

Boundary: If there is an intersection between the probe and an atom but the distance between the grid point and all the atoms of the molecule is larger than the radius of the atoms minus r_g, the grid point is classified as being on the boundary of the SES (yellow points in Figure 4). The distance between such a point and the SES is determined in the second step of the algorithm.

Distance field refinement: In this second step, the remaining distances to the SES for the points in the border region are computed. Thus, the algorithm searches the neighborhood of these boundary points for adjacent points that are outside the SES. This neighborhood is defined by the set of points at a distance $r_p + r_g$ or less from the point of interest. The algorithm initializes the distance of the central point to $-r_g$, and then it iterates over all the points in the neighborhood and selects the closest one that lies outside the SES. If no neighboring grid point was found that is outside the SES, the distance of the point is not updated (e.g., it remains $-r_g$). If there is at least one, the distance of the current point is updated using the following equation: $d = r_p - \|pos_n - pos_c\|$, where r_p is the probe radius, $pos_n \in \mathbb{R}^3$ the position of the neighbor grid point that is classified as outside and closest to the current grid point, and $pos_c \in \mathbb{R}^3$ the position of the current grid point. Figure 5 illustrates this process. Note that updating the distances in parallel requires no synchronization. The algorithm only searches the neighborhood of each sample for points outside the SES and does not modify these values in this step; they were computed in the first step.

3.1.1 Implementation

We have implemented our algorithm to run on the GPU, using compute shaders for both steps. To represent the signed distance field we use a 3D texture centered at the molecule. Each thread of the first compute shader classifies a grid point of the distance field as described in the first step of the algorithm. Then, each thread of the second compute shader computes the distance to the SES of one *boundary* grid point. In order to perform these computations efficiently, some optimizations are applied, which we now describe.

In order to classify a point, the algorithm has to check for intersections of every grid point against the atoms of the molecule. This can be prohibitive when the number of atoms increases, so it is crucial to only perform the intersection test with close-by atoms and discard the ones that are farther away. Efficient retrieval of neighboring atoms within a fixed radius is a common problem that is usually solved using data structures for spatial subdivision (see, e.g., [6, 1, 11]). We opted for the method of Green [6], which was for example also used by Krone et al. [15], and Skånberg et al. [23] for molecular visualization. This method subdivides the space into a regular grid and sorts the atoms into the grid cells based on their centers. Then, we can obtain neighboring atoms within a fixed radius in constant time. In our case, we set the cell size to the probe radius plus the maximum vdW radius. This cell size guarantees that we only have to visit 27 cells of the grid to obtain all the atoms that possibly intersect with our probe.

The second compute shader has to execute a thread for each point classified as *boundary*, since these points have no distance assigned yet. However, keeping track of all these points is not a straightforward task in the context of a parallel algorithm. One possible solution is to mark these points in the first step and, on the CPU, pack them into a buffer, so the second step can be executed only for them. Although this is a simple solution, it does not scale well when the resolution of the grid increases and we lose spatial coherence in the execution of a workgroup. Instead, we chose a more GPU-friendly solution. We grouped the grid points into bricks of 8^3, and then, we selected those that need further refinement. These bricks are defined as the ones that have at least one boundary point (see the yellow bricks in Figure 7). The chosen algorithm reduces the workload of the CPU and accelerates the selection process. In addition, it lowers the data transfer between GPU and CPU, and it adds local coherence inside the workgroups in the second step as we use a workgroup size equal to the brick size (8^3).

3.2 SES Progressive Refinement

One of our main goals in this project was to ensure real-time interaction, so we decided to design a progressive approach.

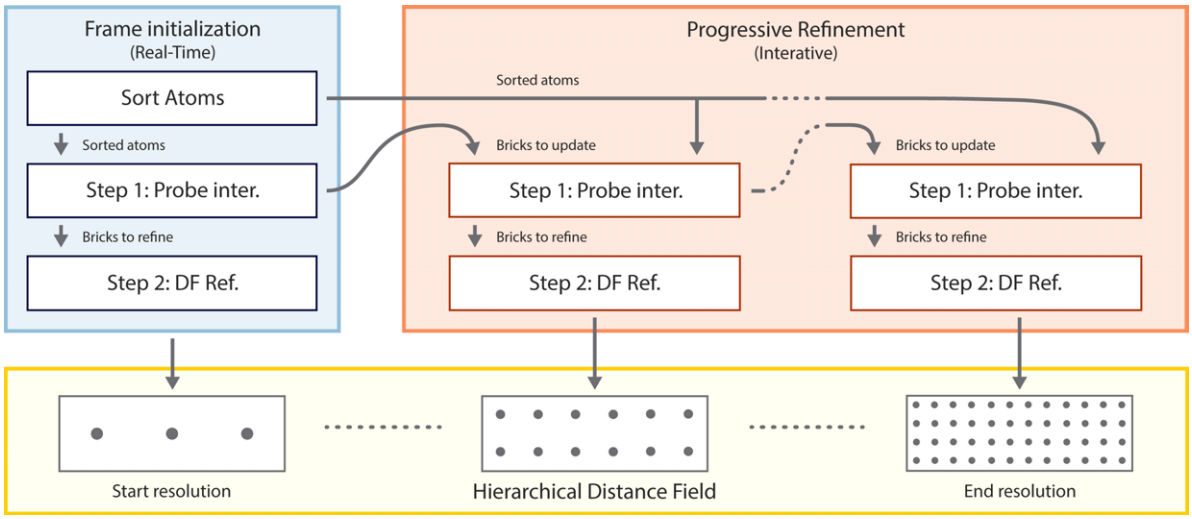

Fig. 6 Overview of the progressive refinement process: First, the atoms of the molecule are sorted and then, a coarse version of the SES is computed in real-time. The sorted atoms and the brick list obtained in the first computation are then used to compute a refined version of the SES. Along the frames, new versions of the SES are computed using the brick list of the previous resolutions until the highest resolution is reached.

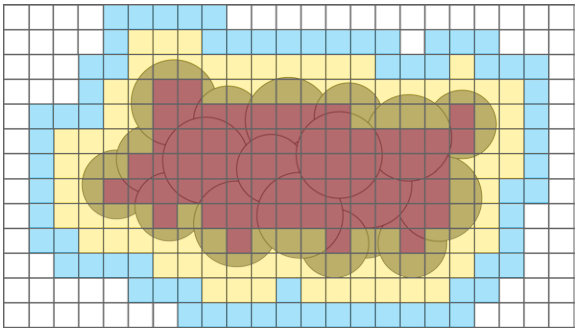

Fig. 7 The grid points are packed together into bricks. In the first step of the algorithm, these bricks are classified. Yellow bricks have at least one grid point labeled as SES border. The blue bricks are the ones close to the yellow ones. Red ones have all grid points classified as interior. The rest of the bricks (white) are not taken into account in the computation of the refined SES.

In our system, the application computes a SES using a low resolution 3D grid in real-time and, in the background, this coarse surface is refined to provide a more exact SES.

Our data structure is composed of a mipmapped 3D texture with a base resolution of 512^3, which is initialized with the value of the probe radius. We have chosen this resolution as a compromise between SES quality and memory consumption. For a virus capsid of 500k atoms (1K4R), the grid cells are still below 1 Å.

When new data is requested (e.g., the user selects another step of a simulation), the algorithm first has to determine the lowest and highest grid resolutions that will be used to compute the different SES levels of detail. The main bottleneck of the algorithm is in the distance field refinement step. When increasing the number of neighboring points that are considered, the performance decreases. Thus, for the coarse level, we choose the smallest neighborhood that keeps an acceptable SES quality. This neighborhood is defined by the relation between the probe radius and the distance between neighbor points, so for the coarse SES, we choose the highest level in the hierarchy where the distance between two neighbor grid points is less than the probe radius. With this configuration, the algorithm considers for each grid point the cells at a (Manhattan) distance of less than or equal to two, which makes a maximum of 124 neighbors. In order to select the highest resolution, the algorithm follows a similar criterion based on the number of neighbors to consider. It chooses the lowest level in the hierarchy where the distance between two neighbor grid points is less than the probe radius divided by 7—for each grid point the algorithm has to visit the cells at a distance less than or equal to seven. These numbers have been chosen empirically from tests performed on our hardware, but they can be modified to tune the algorithm to different hardware.

Once the initial and final levels are defined, the algorithm computes the SES for the starting level. To do so, the algorithm first sorts the atoms into the spatial subdivision grid; the same spatial subdivision grid is used for all resolution levels, so it has to be computed only once. Next, the two steps of the algorithm are executed to calculate the coarse version of the SES. In the first step, the classification of the grid points and the selection of the bricks that need to be refined are performed. In addition, this step also keeps track of the bricks that need to be updated in higher resolutions (all colored bricks in Figure 7). In the second step, the bricks that need to be refined are updated with the distance to the SES. At the coarse level, this whole computation takes milliseconds and can be performed in real-time during rendering. The outputs of this stage are: a distance field that represents a coarse SES, the regular spatial subdivision grid

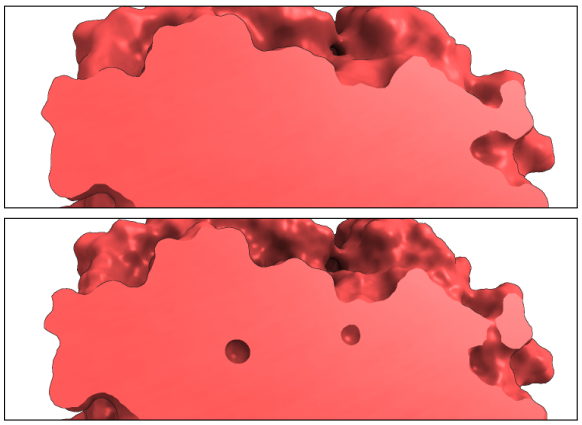

Fig. 8 Discrete sampling can miss features inside the surface (e.g., cavities or tunnels). This image shows how a cavity is not detected in a surface generated with a low grid resolution (top, generated with a 128^3 grid resolution), while it is in a high resolution (bottom, generated with a 256^3 grid resolution). Updating interior bricks in all levels is mandatory to overcome this limitation of the discrete sampling.

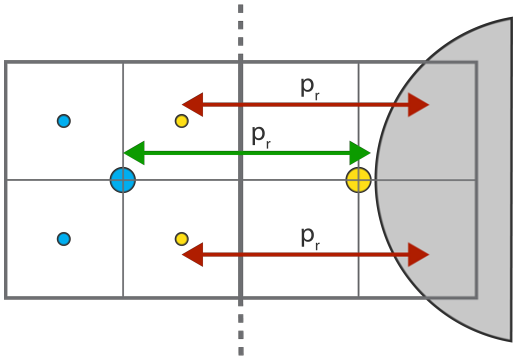

Fig. 9 The big blue point on the left belongs to a brick that does not need to be refined but has a neighbor that belongs to another brick that needs to be refined: the big yellow point to the right. In a higher resolution, some of the points of the left brick became yellow (they have an intersection with the molecule) even if all the points in the previous resolution had no intersection. This bricks have to be updated in the higher resolution levels to avoid artifacts on the resulting surface.

over all atoms, and the list of bricks that need to be recomputed at the next levels. The algorithm immediately renders the coarse SES, so that the user can interact and inspect. Meanwhile, in the background, the algorithm computes the following levels of the hierarchy using the information obtained from the coarse level calculation.

In order to reduce the computation in the remaining levels, only the volume of the SES or close to it is recomputed. These areas of the volume are defined by three types of bricks: the bricks in the border of the SES (yellow bricks in Figure 7), the bricks inside the SES (red bricks in Figure 7) and the bricks adjacent to the SES (blue bricks in Figure 7). The first ones are defined as the bricks containing at least one point in the border of the SES, and they are obviously recomputed as they contain the actual surface. However, the two other types of bricks are also recomputed to avoid artifacts in the refined versions. The bricks inside of the SES (red bricks in Figure 7) contain points completely inside the SES, and they are recomputed to not miss internal features of the SES (cavities or tunnels). Small interior features of the SES are missed if the surface is poorly sampled, which can happen in the first, coarse levels of the hierarchy. Recomputing these areas with a higher sampling in the lower levels can recover these missing features (see Figure 8).

The third type of bricks, the ones adjacent to the SES (blue bricks in Figure 7), contains grid points completely outside the SES that have at least one neighboring brick in the boundary region. These bricks have to be recomputed in the remaining levels to avoid incoherent distance values between adjacent grid points. A brick can be composed only of grid points outside the SES at a certain resolution, but, when more samples are used, that could not be the case anymore. The new samples can be part of the SES boundary as illustrated in Figure 9, hence they need to be updated.

Once this brick list is computed, it is downloaded to CPU memory. In the background, the application then executes the algorithm of Section 3.1 for the selected bricks. The algorithm works exactly in the same way as for the coarse level. The result is presented to the user using a smooth transition between the previous level and the new, refined one. This process is repeated for further levels in the hierarchy until the algorithm computes the last one. These computations are performed in parallel to the rendering, so for each frame a fixed number of bricks are processed. To maintain an interactive framerate, we process only 512 bricks per frame, but this number could be reduced on slower GPUs.

3.3 Detection of missing features

Due to the discrete nature of our representation, the volume of the SES can be overestimated. Each point of the grid stores only an approximation of the distance between the point and the surface. The real distance, however, is in the range $[d, d+(\sqrt{3}\cdot r_g)]$, where d is the approximated distance stored in a grid cell and r_g the distance between two neighboring cells along one axis. That is, r_g defines the maximum error of our solution. Consequently, the probability of missing internal features of the molecule (cavities or tunnels) increases if r_g increases. Simply speaking, the chances of missing a small cavity are higher if the space is sampled by a small number of points (i.e., a coarser grid).

These limitations may not be relevant for small r_g, but, when r_g is larger than a certain value, the result can deviate significantly from the analytical solution. Thus, we implemented a solution that refines the last resolution level of our SES in certain areas. The algorithm analyzes the bricks of the currently highest resolution grid to find areas with possible missing features. These bricks have been defined as hav-

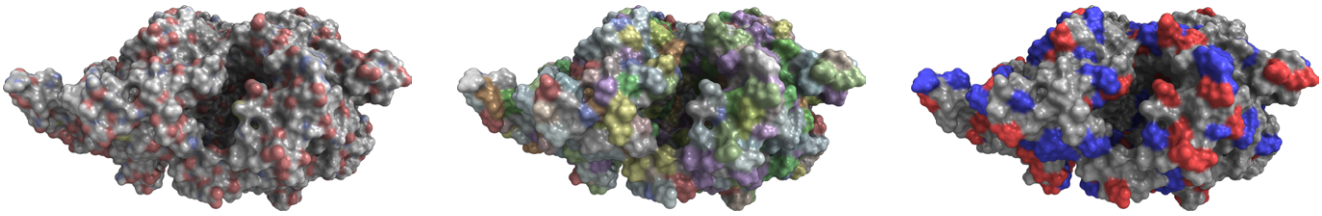

Fig. 10 Left: the SES is colored using the CPK color convention to identify the atoms; center: color mapping showing the residue types; right: (per residue) electrostatic potential.

ing at least one point that satisfies the following conditions: first, this point has to be located inside the SES but outside of the atoms; second, it has to be surrounded only by interior points; and third, one of its neighbors also has to be outside of the atoms and, together, they have to satisfy the following condition: $d_{p1} + d_{p2} + |pos_1 - pos_2| \geq 2 \cdot r_p \cdot \mu$, where d_{p1} and d_{p2} are the distances between the points and their closest atoms, $pos_1 \in \mathbb{R}^3$ and $pos_2 \in \mathbb{R}^3$ are the positions of the grid points, r_p is the probe radius, and μ is a user-defined parameter that controls the number of selected bricks. If two points satisfy these conditions, a probe might fit between them. For these bricks, our algorithm is executed with a resolution of 64^3. If a probe can be placed in one of these new sample points, our mipmap is updated with the newly calculated distances at the highest resolution.

3.4 SES Coloring

Researchers often use the SES to detect tunnels or cavities in the surface, but it is also important for them be able to identify the atoms/residues, the electrostatic potential, the hydrophobicity, and other properties in different parts of the surface. To support the visual analysis, we store another 3D texture with the lowest resolution selected for this molecule, and for each grid point we store the color or property associated to the nearest atom. Using this texture, the surface can be smoothly colored according to the stored properties. Using the lower resolution allows us to have smooth transitions between colors, since higher resolutions make the borders between colors clearly visible. Moreover, it does not add extra computational costs—it can be computed once together with the coarsest level—and it uses limited extra memory, as the selected resolution is never greater than 128^3. Different coloring methods are presented in Figure 10.

3.5 SES Rendering

In this section, we detail the rendering of the SES from the previously computed distance field. The most common techniques to visualize a surface from a distance field are either by extracting a mesh using Marching Cubes (MC) [19], or by ray-marching it directly [9]. We chose to ray-march the distance field, as this does not require extra storage and can be done efficiently from any resolution level of the distance field. The classical algorithm of ray-marching takes steps along the ray separated by a specific distance, but we use the distance stored in the grid points instead. This speeds up the rendering and removes some possible artifacts (e.g. missing the surface on the border). When the ray hits the surface, we use a Sobel filter to compute the normal at that point. This method results in high-quality normals but needs 26 texture lookups. On less performant hardware, a simpler filter could be used instead (e.g. central differences).

Another advantage of ray-marching over MC is that it allows us to perform smooth transitions between the different levels of detail. When a new level i of the distance field hierarchy is computed, the renderer has to perform a transition from the current level $i+1$ to the level i along a time frame t. This transition is done by the hardware trilinear interpolation. The render performs a linear interpolation along the time of the level identifier, from $i+1$ to i. This interpolated value is then used as the LOD parameter in the *textureLod* call during the ray-marching. When the hit point is reached and the normal has to be calculated, the algorithm also linearly interpolates the distance between the hit point and the neighbor samples used to compute it. This simple algorithm gives us a smooth transition between different grid resolutions for both surface shape and lighting. In Figure 11, a molecule was rendered with different grid resolutions from left to right using this algorithm for the transitions.

Moreover, we improved the performance of our ray-marching algorithm using a well-known technique from GPU-based volume raycasting. In order to reduce the ray traversal distance, we render the bricks to be refined to encode a texture with the entry and exit points of the rays, skipping thus the empty space of the volume.

Similar to Tarini et al. [24], we apply rendering techniques that enhance the visualization and facilitate the understanding of the shape of the surface. We have implemented the ambient occlusion algorithm proposed by Hermosilla et al. [10]. We have chosen this algorithm despite it was designed only for the Space-filling and Ball-and-Stick representations since it works in real-time without pre-computations. It works in object-space and is, therefore, independent of the camera orientation. As the SES is derived

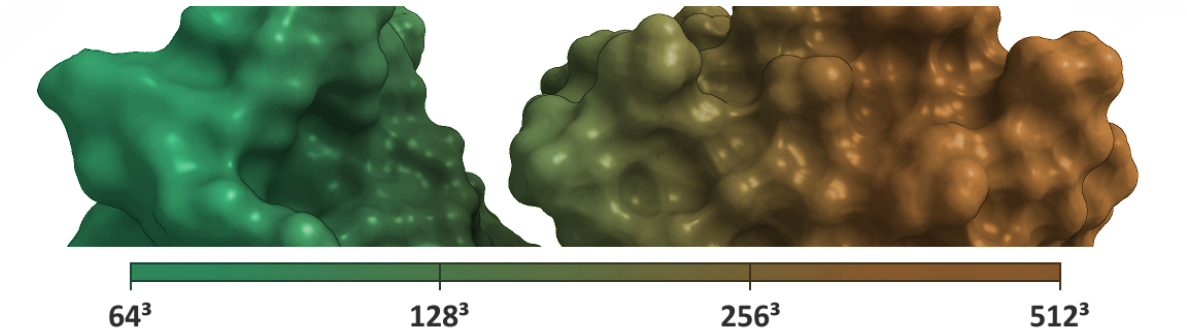

Fig. 11 The image shows from left to right the progressive refinement in the SES generation process. Note the continuity through the different steps thanks to our algorithm.

Table 1 Performance for different molecule obtained using the following hardware configuration: GeForce GTX 970, Intel i7 and 12 GB RAM. The different columns show the time required to sort the atoms in the regular grid, the resolution of the distance field in the base level and the time in milliseconds needed to compute it, the resolution of the distance field used in the last refined level, the time in milliseconds needed to compute all the refined levels, the mean frames per second obtained during the refinement and, in the last column, the cell size of the last refined level.

Molecule (#atoms)	Sort atoms (ms)	Base level resolution	Base level computation (ms)	Last refined level resolution	Computation refined levels (ms)	FPS during computation	Cell size last refined level (Å)
Traj 1 (2,066)	0.16	64^3	5.42	256^3	775	51.61	0.21
Traj 2 (3,967)	0.20	64^3	5.23	256^3	464	79.74	0.28
Traj 3 (11,224)	0.47	128^3	10.09	512^3	4,149	45.55	0.20
2G47 (16,962)	0.60	128^3	9.72	512^3	2,151	90.63	0.29
1S3S (22,367)	0.80	128^3	7.58	512^3	930	133.29	0.40
3J3A (46,276)	1.55	128^3	8.94	512^3	1,822	136.66	0.43
3EXG (83,339)	2.18	128^3	8.81	512^3	1,242	165.06	0.51
1CWP (227k)	6.24	128^3	11.52	512^3	2,468	164.51	0.58
1K4R (545k)	11.96	128^3	15.89	512^3	1,649	177.08	0.98

from the Space-filling representation, we can directly apply this algorithm. In a previous pass to the rendering, the atoms are used to compute an occupancy grid, and the ambient occlusion factor is computed in screen-space for each pixel. We also added a simple silhouette rendering algorithm to facilitate the understanding of the molecular shape. This algorithm detects depth differences between neighboring pixels and renders a black silhouette where this difference is higher than a user-defined threshold value. Figure 13 shows a SES rendering including those two effects.

4 Results & Discussion

We have evaluated our technique by comparing it to the GPU-based Contour-Buildup method by Krone et al. [15], which is implemented using CUDA and included in the publicly available visualization system MegaMol [8]. Our results show that our technique is at least two to three times faster than the CUDA-Contour-Buildup for the lowest (base) resolution. Owing to the progressive refinement, our method can deal with large models seamlessly, with a sustained interactive frame rate. The progressive SES computation thus enables a fast rendering of large molecular simulation trajectories on the fly. Users can interactively get a feel for the evolution of the simulated molecule over time running the computation with the base resolution. When the playback stops at any point of the trajectory, the progressive refinement will automatically produce a high-quality SES within seconds, even for very large molecular complexes like virus capsids (1CWP and 1K4R) (see Table 1). A drawback of the CUDA implementation of the Contour-Buildup in MegaMol is that it requires a lot of GPU memory. For the test data set 1S3S with 22 k atoms, it requires 1.9 GB of memory (the SES computation takes 32 ms on a Nvidia GTX 970), while a protein of about 30 k atoms (PDB ID: 3K19) requires already 2.7 GB of memory. Consequently, even newer consumer graphics cards with 4 GB VRAM or more will quickly run out of memory for very large structures like virus capsids. This prevents the use of the CUDA-Contour-Buildup for such large structures. On the other hand, our technique requires in the worst scenario just 620 MB for the data structure and 32b for each atom of the molecule, making it very scalable. Notice that for small test cases, like *Traj 3* in Table 1, our algorithm takes longer to complete the computation of the highest resolution, whereas times descend for larger molecules. This is because for small molecules the cells in the high-resolution grid are very small, and there is a large number of them inside the probe. Therefore, the algorithm needs to visit many neighbors and slows down. This can be easily avoided in practice since such high resolution is not needed for small models.

Table 2 Performance in frames per second when rendering the distance field for different resolutions and molecules using the ray marching algorithm. Moreover, the last column shows the frames per second of our method using ambient occlusion. When we compare our method with MegaMol (the last column *RC* shows the performance in fps of that system), we see that our system is capable of achieving sustained interactive framerates even for large molecules. The timings were computed by averaging the rendering time from 512 distinct random directions that uniformly sample a sphere, to even out variations due to camera placement.

Molecule	#atoms	128^3	256^3	512^3	RC[15]	512^3 + AO
Traj 1	2,066	250.80	233.69	—	—	—
Traj 2	3,967	254.34	248.42	—	—	—
Traj 3	11,224	225.11	237.54	227.90	—	176.76
2G47	16,962	212.77	221.33	205.89	188.1	175.08
1S3S	22,367	229.86	223.73	212.74	126.3	151.46
3J3A	46,276	202.17	185.91	137.60	72.0	114.79
3EXG	83,339	200.18	202.94	180.56	69.2	137.81
1CWP	227 k	175.47	167.71	158.93	27.7	110.57
1K4R	545 k	165.71	162.27	144.35	12.6	76.40

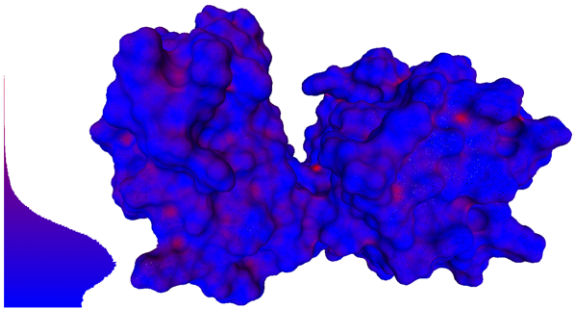

Fig. 12 Visualization of the distance between our result and the analytical solution. The distance is represented by a color scale from blue (distance 0) to red (distance equal to the probe radius). On the left part of the image, a histogram of the distances is shown.

We also compared the rendering speed of our method to that of MegaMol. While our method uses volume ray marching (see section 3.5), MegaMol uses GPU-based ray casting to render the implicit patches of the SES [14]. A comparison of the frame rates can be found in Table 2. Despite having similar performance for up to medium-sized proteins (2G47), ray marching clearly outperforms ray casting for very large data sets even when using high-resolution volumes. The rendering performance on MegaMol is limited by the number of implicit patches generated (i.e. number of atoms), our rendering method, on the contrary, depends on the grid resolution and the shape of the molecule. All tests were run at resolution 1024×768 px.

We also compared the performance of our algorithm to that of EDTSurf [27] (see Table 3). Although the comparison is not precise since we only use powers of two as grid resolutions, it shows that our algorithm is consistently faster.

We have compared the exact SES surface using chimera for a molecule of 3,967 atoms (this computation is not practical for very large molecules), and the result of our algorithm using a resolution of 256^3. To this end, we sampled

Table 3 Performance comparison between our method and the EDTSurf software [27]. The table shows the resolution of the discretization used by the EDTSurf algorithm and the time in seconds needed to compute the SES with the CPU implementation provided in their website. We compared these results with the time required by our technique to compute—progressively—the SES until a grid resolution of 256^3 was reached, since we use power-of-two grid resolutions and they do not.

Molecule	#atoms	EDTSurf Grid Resolution	EDTSurf Time (s)	Our method at 256^3 (s)
Traj 1	2,060	$203\times181\times172$	1.64	0.78
Traj 2	3,967	$217\times194\times271$	3.22	0.46
Traj 3	11,224	$275\times225\times299$	5.19	0.20
2G47	16,962	$201\times299\times234$	3.34	0.16
1S3S	22,367	$254\times299\times89$	1.99	0.09
3J3A	46,276	$299\times287\times299$	4.44	0.19
3EXG	83,339	$216\times225\times299$	3.64	0.16

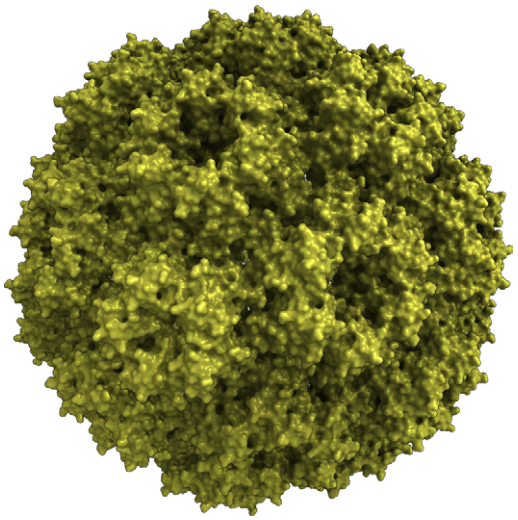

Fig. 13 SES of the virus capsid 1CWP with 227 k atoms generated with a distance field resolution of 512^3.

2,676,613 random points uniformly distributed over the surface given by our algorithm, and computed their distance to the exact surface computed by Chimera [20] (see Figure 12). The result was an average distance of 0.231743Å with a root mean square error of 0.269498Å. Notice that this amounts to a standard deviation of roughly 0.14Å, so even if there are points at zero distance of the exact SES, and others at up to 1.43Å, they are extremely rare (see histogram on figure 12). This means that our algorithm yields a surface with a small outward bias (of the order of magnitude of the spacing between samples) originated by its conservative nature. Occasionally, it may miss some small cavity, but this will happen only if inspecting a whole molecule, at a scale at which this cavity cannot be seen. In order to explore all possible cavities, we could adapt our data structure to only the visible part of the molecule when zooming in, so that a higher resolution computation (for a smaller number of atoms) is performed.

5 Conclusions & Future Work

In this paper we have presented a new GPU-accelerated algorithm that computes the SES on the fly. In contrast to previous approaches, we are able to progressively refine the result, which allows calculation and rendering of SES at interactive framerates for very large models like the virus capsid shown in Figure 13. Our system requires no precomputation and thus, we can handle dynamic models. The progressive component of our algorithm is achieved using a space discretization. At first sight, this might be considered a disadvantage, since too coarse a refinement might lead to missed cavities or incorrect surface shapes. However, the resolution we use is fine enough to avoid such problems, as shown by the tests where we compared the surface obtained by our method to the one computed by an analytical method, and found that they only had small discrepancies, as shown in Figure 12. Another advantage of our approach is that we support visually smooth transitions between refinement levels, thus, the progressive improvement happens seamlessly. Finally, we have also shown how we can encode atom properties on the generated surface using color to support visual analysis.

In the future, we are planning to reduce the memory consumption using virtual texturing, opening the possibility of using even larger resolutions for very large molecules.

Acknowledgements This work has been partially supported by grant TIN2014-52211-C2-1-R from the Spanish *Ministerio de Economía y Competitividad* with FEDER funds, and by the German Research Foundation (DFG) as part of Collaborative Research Center SFB 716.

References

1. Behley, J., Steinhage, V., Cremers, A.B.: Efficient radius neighbor search in three-dimensional point clouds. In: 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 3625–3630 (2015)
2. Blinn, J.F.: A Generalization of Algebraic Surface Drawing. ACM Transactions on Graphics **1**(3), 235–256 (1982)
3. Can, T., Chen, C.I., Wang, Y.F.: Efficient Molecular Surface Generation Using Level-Set Methods. Journal of Molecular Graphics and Modelling **25**(4), 442–454 (2006)
4. Connolly, M.L.: Analytical Molecular Surface Calculation. Journal of Applied Crystallography **16**(5), 548–558 (1983)
5. Edelsbrunner, H., Mücke, E.P.: Three-dimensional Alpha Shapes. ACM Transactions on Graphics **13**(1), 43–72 (1994)
6. Green, S.: White paper: Cuda particles. Tech. rep. (2007)
7. Greer, J., Bush, B.L.: Macromolecular shape and surface maps by solvent exclusion. Proceedings of the National Academy of Sciences **75**, 303–307 (1978)
8. Grottel, S., Krone, M., Müller, C., Reina, G., Ertl, T.: Megamol—a prototyping framework for particle-based visualization. IEEE Transactions on Visualization and Computer Graphics **21**(2), 201–214 (2015)
9. Hadwiger, M., Sigg, C., Scharsach, H., Bhler, K., Gross, M.: Real-time ray-casting and advanced shading of discrete isosurfaces. Computer Graphics Forum **24**(3), 303–312 (2005)
10. Hermosilla, P., Guallar, V., Vinacua, A., Vázquez, P.: High quality illustrative effects for molecular rendering. Computers & Graphics pp. – (2015)
11. Hoetzlein, R.C.: Fast fixed-radius nearest neighbors: Interactive million-particle fluids. In: GPU Technology Conference (2014)
12. Jurcik, A., Parulek, J., Sochor, J., Kozlikova, B.: Accelerated Visualization of Transparent Molecular Surfaces in Molecular Dynamics. In: IEEE Pacific Visualization Symposium, pp. 112–119 (2016)
13. Kozlíková, B., Krone, M., Lindow, N., Falk, M., Baaden, M., Baum, D., Viola, I., Parulek, J., Hege, H.C.: Visualization of molecular structure: State of the art revisited. Computer Graphics Forum (2016). (to appear)
14. Krone, M., Bidmon, K., Ertl, T.: Interactive Visualization of Molecular Surface Dynamics. IEEE Transactions on Visualization and Computer Graphics **15**(6), 1391–1398 (2009)
15. Krone, M., Grottel, S., Ertl, T.: Parallel Contour-Buildup Algorithm for the Molecular Surface. In: IEEE Symposium on Biological Data Visualization, pp. 17–22 (2011)
16. Krone, M., Stone, J.E., Ertl, T., Schulten, K.: Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories. In: EuroVis - Short Papers, vol. 1, pp. 67–71 (2012)
17. Lindow, N., Baum, D., Hege, H.C.: Ligand excluded surface: A new type of molecular surface. IEEE Transactions on Visualization and Computer Graphics **20**(12), 2486–2495 (2014)
18. Lindow, N., Baum, D., Prohaska, S., Hege, H.C.: Accelerated Visualization of Dynamic Molecular Surfaces. Computer Graphics Forum **29**(3), 943–952 (2010)
19. Lorensen, W.E., Cline, H.E.: Marching Cubes: A High Resolution 3d Surface Construction Algorithm. In: ACM SIGGRAPH Computer Graphics and Interactive Techniques, vol. 21, pp. 163–169 (1987)
20. Pettersen, E.F., Goddard, T.D., Huang, C.C., Couch, G.S., Greenblatt, D.M., Meng, E.C., Ferrin, T.E.: UCSF Chimera - A visualization system for exploratory research and analysis. Journal of Computational Chemistry **25**(13), 1605–1612 (2004)
21. Richards, F.M.: Areas, Volumes, Packing, and Protein Structure. Annual Review of Biophysics and Bioengineering **6**(1), 151–176 (1977)
22. Sanner, M.F., Olson, A.J., Spehner, J.C.: Reduced Surface: An Efficient Way to Compute Molecular Surfaces. Biopolymers **38**(3), 305–320 (1996)
23. Skånberg, R., Vázquez, P.P., Guallar, V., Ropinski, T.: Real-time molecular visualization supporting diffuse interreflections and ambient occlusion. IEEE Transactions on Visualization and Computer Graphics **22**(1), 718–727 (2016)
24. Tarini, M., Cignoni, P., Montani, C.: Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization. IEEE Transactions on Visualization and Computer Graphics **12**(5), 1237–1244 (2006)
25. Totrov, M., Abagyan, R.: The Contour-Buildup Algorithm to Calculate the Analytical Molecular Surface. Journal of Structural Biology **116**, 138–143 (1995)
26. Varshney, A., Brooks, F.P., Wright, W.V.: Linearly Scalable Computation of Smooth Molecular Surfaces. IEEE Computer Graphics and Applications **14**(5), 19–25 (1994)
27. Xu, D., Zhang, Y.: Generating Triangulated Macromolecular Surfaces by Euclidean Distance Transform. PLOS ONE **4**(12), e8140 (2009)
28. Yu, Z.: A List-Based Method for Fast Generation of Molecular Surfaces. In: Int. Conf. of the IEEE Engineering in Medicine and Biology Society, vol. 31, pp. 5909–5912 (2009)