Accelerated Visualization of Transparent Molecular Surfaces in Molecular Dynamics

Adam Jurčík*
Masaryk University
Brno, Czech Republic

Julius Parulek†
University of Bergen
Norway

Jiří Sochor‡
Masaryk University
Brno, Czech Republic

Barbora Kozlíková§
Masaryk University
Brno, Czech Republic

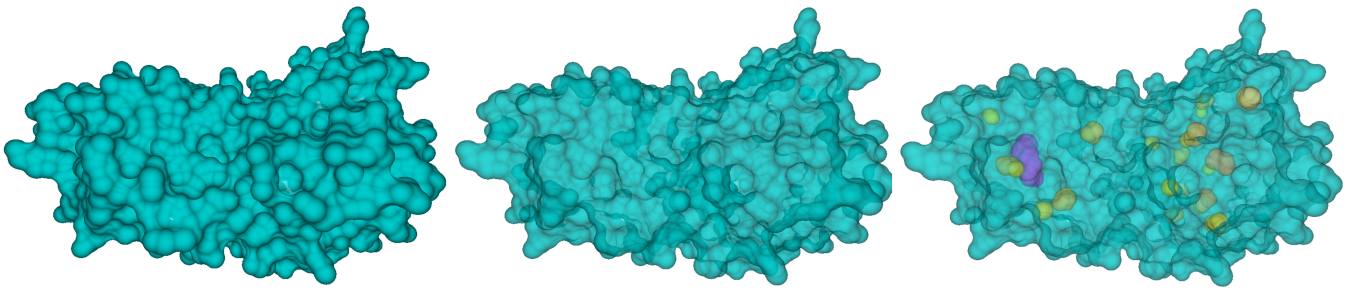

Figure 1: Example of protein with PDB ID *1VIS* demonstrating our visualization method. Left: the full SES model (33.3 FPS). Middle: with basic transparency (33.3 FPS). Right: a user-defined transparent visualization that includes cavities (32.2 FPS). The performance of the visualization was measured using the resolution of 1024×768 and the fill rate was 48.3%.

Abstract

The reactivity of the biomolecular structures is highly influenced by their structural features. Thus, studying these features along with the exploration of their dynamic behavior helps to understand the processes ongoing in living cells. This can be reached by the visual representation of these processes as visualization is one of the most natural ways to convey such information. However, none of the currently available techniques provides the biochemists with an intuitive real-time representation of the dynamic movements of molecules and precise geometrical based extraction of their structural features performed instantly. In this paper we introduce such a technique enabling the user to compute and also to visualize the molecular surface along with inner voids. To obtain a better insight into the molecule, our technique enables to visualize the molecular surface transparently. The opacity can be adjusted by changing user-defined parameters in order to enhance the perception of the surfaces of inner voids. All integrated algorithms run in real-time which gives the user a big variety of exploration possibilities. The importance of our approach is even amplified with respect to the fact that currently the size of molecular dynamics simulations is increasing dramatically and offline rendering thus becomes impracticable. The usability of our technique was evaluated by the domain experts.

Index Terms: I.3.5 [Computational Geometry and Object Modeling]—Boundary representations; I.3.7 [Three-Dimensional Graphics and Realism]—Visible line/surface algorithms

*e-mail: xjurc@fi.muni.cz
†e-mail:julius.parulek@uib.no
‡e-mail:sochor@fi.muni.cz
§e-mail:kozlikova@fi.muni.cz

1 Introduction

Detailed exploration of biomolecular structures and their functions has been in the focus of researchers in molecular biology for decades. Such knowledge helps to understand the biological processes in organisms and in consequence, to better target the design of new chemical matters (e.g., drugs). Many researchers have been focusing on the analysis of protein structures, i.e., their constitution. Recent discoveries confirm that the function of proteins is not fully determined only by their structure but also their dynamic movements play a significant role [11]. This even stresses the importance of studying and exploring the trajectories of molecular dynamics (MD) in detail. As the length of the captured or simulated trajectories is dramatically increasing, their real-time exploration becomes a necessity. The domain experts require a visual insight into the protein structure and its dynamic movement instantly, without tedious precomputation, offline rendering, or making previews which are currently their only options. This is especially crucial when analyzing MD trajectories containing thousands of frames, where the users cannot spend much time analyzing just a single frame either due to computational or visualization limitations. Moreover, the exploratory process of MD is often concerned with the visual identification of protein binding sites where a ligand can interact with the host protein. These sites represent different molecular surface features known as cavities, pockets, or tunnels. There is a legacy of tools and approaches that enable to extract these features. Two major challenges in regards to the surface feature analysis are their fast extraction and visualization in the most informative manner.

In our approach we are focusing on visualization of both molecular surface and its cavities that represent the protein inner void space, directly inaccessible from the molecular surface. We face the aforementioned challenges by introducing a novel approach to real-time visualization and exploration of protein molecular dynamics when the user can interactively manipulate with the structure and thus explore the protein, its inner cavities, and its behavior efficiently. This is reached by introducing several enhancements (Fig. 1), such as real-time computation and rendering of transparent molecular surface and real-time detection and rendering of inner cavities.

The main contribution of our solution consists of the introduction of the accelerated rendering pipeline improving the performance of visualization of transparent solvent-excluded surface (SES) of the molecule (in comparison with Kauker et al. [13]). To provide a better perception of inner surface features (cavities), we allow users to change the parameters of the opacity modulation. Moreover, we extended the existing state-of-the-art approach [16] to SES computation by proposing methods for computation and rendering of individual SES patches.

Currently existing methods provide a solution for each of these topics separately (i.e., computation of molecular surface and its transparency), including the computation of inner cavities. However, none of these methods combines all these topics in order to provide the domain experts with a solution for their real-time visualization and exploration. We fill this gap by integrating the computation and transparent visualization of both molecular surface and inner cavities. Moreover, our implementation of the individual parts often overcomes the performance limitations of the previous solutions.

2 Related Work

Our approach touches several research areas, including computation and visualization of protein cavities and their surfaces. Further we focus on the real-time visualization of surfaces and their transparency which is achieved by using the order-independent transparency approach.

2.1 Extraction and Visualization of Molecular Surfaces

There are several types of molecular surface representation proposed in the literature [14]. Regarding the cavity analysis, the solvent-excluded surface (SES) is the most commonly used representation preferred also by the domain experts. This representation allows us to directly asses whether a ligand, approximated by a sphere of a given radius, can penetrate through the molecular surface. The field of geometrical analysis of molecular structures focusing on the detection and further exploration of the void space is very vast and, therefore, we relate only to the techniques that we consider to be relevant and close to our work.

In 1983, Connolly proposed an analytical approach for construction of the SES representation [6]. In 1996, Sanner et al. [28] introduced the reduced surface algorithm to construct SES. In the same year, Totrov and Abagyan [32] introduced the contour-buildup algorithm to create SES representation. The algorithm is based on the sequential buildup of multi-arc contours. One of the advantages of this approach is its spatial localization meaning that it is applicable to partial molecular surfaces. In 1995, Edelsbrunner et al. [7] proposed to exploit alpha shapes to detect and measure the voids in proteins. Another approach, linearly scalable algorithm for computation of smooth molecular surfaces was then presented by Varshney et al. [33].

One of the significant improvements of the SES computation was published by Parulek and Viola [26]. Their approach does not require any precomputation. It is based on the theory of implicit surfaces where the value of the implicit function helps to determine the inner and outer points with respect to the surface. The implicit function is composed of three types of patches from which the SES is constructed.

However, none of these solutions dealt with molecular dynamics. Krone et al. [15] presented their approach to the visualization of the SES using GPU ray-casting technique which allowed to achieve interactive frame rates. Another approach by Lindow et al. [20] even accelerated the construction of the SES by scaling the parallel contour-buildup algorithm to more CPU cores and using boundary quadrangles as rasterization primitives. Moreover, Krone et al. [16] proposed a parallel version of the contour-buildup algorithm for GPUs that performed better for larger structures. In our approach, we further extend it so that the computed SES can be rendered using correct transparency.

Similarly, the analytical approaches are applicable to the protein inner voids. We focus only on the analytical computation and visualization of cavities which can contain a potential protein binding site. Parulek et al. [25] presented their approach to the computation and visual analysis of cavities in simulations of molecular dynamics. The computation is based on implicit function. The subsequent exploration is supported by graph based visualizations. Another approach to the visual analysis of dynamic protein cavities and binding sites was proposed by Krone et al. [17]. Their system combines semi-transparent surface visualizations with sequence diagrams and relational graphs to communicate different properties of detected cavities. Lindow et al. [19] presented an alternative approach to the exploration of the time-dependent changes of a cavity. To avoid the animation of the traced cavity, they visualize the dynamics of the cavity as a compact representation in a single image, color coded according to time.

2.2 Visualization of Surface Features

There are also several visualization techniques enhancing the visibility of molecular surface features located inside the molecule. The main source of these techniques is described in the computer graphics literature. Bair and House [2] presented an approach for visualization of layered surfaces, which uses grid-based surface textures. Their experiments focus on the perception of textured surfaces and the best configuration of the textures. Other illustrative techniques presented by Tarini et al. [31] and Lawonn et al. [18] represent alternatives to the visualization of the shape of molecular surfaces. These techniques are based on ambient occlusion or its combination with line integral convolution and were not designed to show more layered surfaces.

One of the most popular techniques for rendering transparent objects is the order-independent transparency (OIT) which does not require the geometry to be sorted. Here the traditional approaches are Virtual Pixel Maps (know also as Depth Peeling) [21] or A-buffer [5]. Thanks to the performance of the current hardware these methods can be successfully adopted to large and dynamic scenes. The Virtual Pixel Maps technique is based on the idea of sorting at the pixel level and accumulating the transparency effect on a multi-pass basis.

While the algorithms for rendering of correctly transparent objects in real-time have emerged already in the previous decade [9], new approaches removing the limitations of these methods are still emerging. Bavoil et al. [3] introduced the peeling of two depth layers in one rendering pass, thus halving the number of rendering passes performed by the original Depth Peeling approach [9]. However, in our case performing more rendering passes would become a bottleneck because of our ray-tracing approach to the surface rendering. Thus, techniques based on the A-buffer suit our technique since they enable to render the entire scene in one pass. When using single pass rendering, it is necessary to store all participating or at least all important fragments into a data structure for later composition. The first approach storing the participating fragments was introduced by Yang et al. [35]. Salvi et al. [27] presented another approach, wherein it is required only to store the important fragments. Storing the fragments leads to an unbounded requirement on the memory size that was addressed by Maule et al. [22] or Vasilakis et al. [34]. Some of the existing techniques aim to decrease both time and memory complexity. McGuire and Bavoil [23] employ the idea of partial coverage to reach this goal, whereas Enderton et al. [8] introduce stochastic transparency for the same purpose. Here each fragment of the transparent geometry covers a random subset of pixel samples of a size proportional to alpha. After averaging, the results possess correct alpha-blended colors.

Applying the aforementioned OIT approaches to molecular surfaces is straightforward since they are represented as meshes. For analytic surfaces, Kauker et al. [13] proposed to store lists (or arrays) of all fragments produced by the basic shapes for later processing using CSG operations. In our technique, we avoid storing and subsequent sorting of a high number of fragments by ray-casting only those fragments that belong to the molecular surface. For enhancing the understanding of the interior of the molecular structure, we took inspiration from the technique presented by Borland [4]. The technique, called ambient occlusion opacity mapping, enables to determine a variable opacity at each point on the molecular surface. In consequence, the interior structures can be more opaque than the outer structures, i.e., molecular surface.

3 Overview

The computation of SES is not a trivial task and requires substantial computation and algorithmic capabilities. Therefore, a technique that could provide the users with an instant computation and interactive and meaningful visualization of cavities in the context of molecular surfaces would be highly beneficial.

The input data comes in a form of MD trajectories describing the motion of individual atoms. Each trajectory snapshot includes a set of atoms, described by their positions and radii. Our rendering pipeline consists of several steps that are performed on a per-frame basis. For better explanation, we split the computations into two groups. The first group deals with data processing that involves the computation of the surface patches and inner voids, i.e., cavities. The second group of computations focuses on visualization of the generated patches. This group includes the estimation of area of cavities, ray-casting the SES patches, and opacity calculation; performed before the final stage represented by the image formation. More specifically, our pipeline consists of the following steps (Fig. 2):

1. We employ the contour-buildup algorithm to construct the molecular surface. Additionally, we enhance the computation to enable i) transparent rendering of the surface and ii) extraction of cavities (Sec. 4.1).

2. For cavity extraction, we utilize the so-called *surface graph* (Sec. 4.2), which allows us to detect the isolated surface components.

3. We estimate the area of the extracted cavities to enable color coding by their size and to decrease the potential clutter by hiding small cavities.

4. Surface elements are visualized using ray-casting (Sec. 5). We perform transparent surface rendering by means of the A-buffer and the opacity modulation. The modulation is based on an estimate of the molecular thickness per each surface fragment on a given ray and a set of user-defined parameters.

In the following we focus on a detailed description of these individual steps.

4 Feature Extraction

In this section we present the algorithms for the computation of molecular surfaces and cavities and then we show how we deal with special cases occurring within the surface construction.

4.1 Surface Computation

Our surface computation algorithm is based on the existing approach that utilizes the GPU capabilities [16]. In the former study, the SES is represented using three different sets of shapes – spheres, toroidal saddles, and spherical triangles depicted by red, blue, and green colors in Fig. 3 (left). However, rendering these shapes produces unnecessary pixels located below the surface of the molecule. As we aim for transparent surface visualization, these interior pixels are becoming unwanted. Therefore, we extended the existing algorithm that it computes the following SES patches: i) *spherical triangles*, as were already described by Krone et al. [16] and Lindow et al. [20], ii) *toroidal patches*, delimited by two spherical triangles, and iii) *spherical patches*, enclosed by edges of two or more toroidal patches.

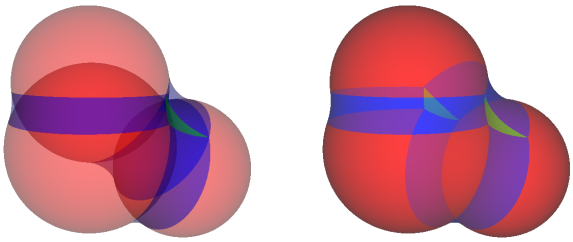

Figure 3: Comparison between the original method [16] rendered with transparency (left) and our extended method (right). The original method renders also parts of spheres (red) and tori (blue) that lie below the surface. Our method produces only patches that are part of the surface.

In comparison with the previous solution, our main contribution here lies in the computation of toroidal and spherical patches. By using them we avoid the situation when the previous algorithm renders the parts of tori and spheres which do not form the final surface (Fig. 3).

To be able to compute the toroidal patches, we employ a hash data structure that enables us to store and retrieve all spherical triangles incident to a torus (Fig. 4). We hash the triangles by three keys, i.e., one for each torus which is connected to a triangle. For this purpose, we implemented a simple hash table, which is based on linear addressing scheme [1]. We compute toroidal patches on a torus by retrieving all its neighboring triangles and sorting them relatively by their angular position around the torus axis. Since the number of neighboring triangles of a torus is low (for molecules with ~10,000 atoms we obtained maximally 8 of them) we employ the bubble sort algorithm for their sorting. After sorting, the pairs of neighboring triangles define the toroidal patches and the corresponding edges. Additionally, we check whether the first two triangles form a visible patch. If not, then the first triangle forms a visible patch with the last one and we rotate the sorted list of triangles by one item.

This allows us to ray-cast the toroidal patches separately instead of the whole tori.

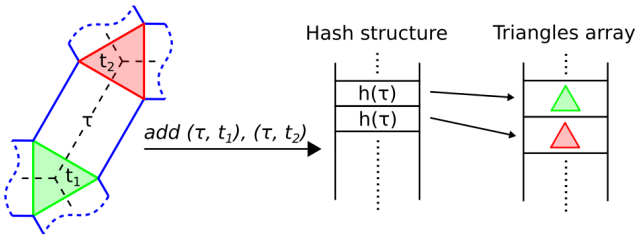

Figure 4: Illustration of the data structure for storing spherical triangles. Triangles t_1 and t_2 are stored linearly in an array and their incident torus τ is connected to them via a hash table.

As already described, a spherical patch is bounded by toroidal patches (Fig. 5). We extended the surface computation algorithm by adding three new GPU kernels that compute the sets of toroidal patches forming the spherical patches (see Sec. 4.2). When render-

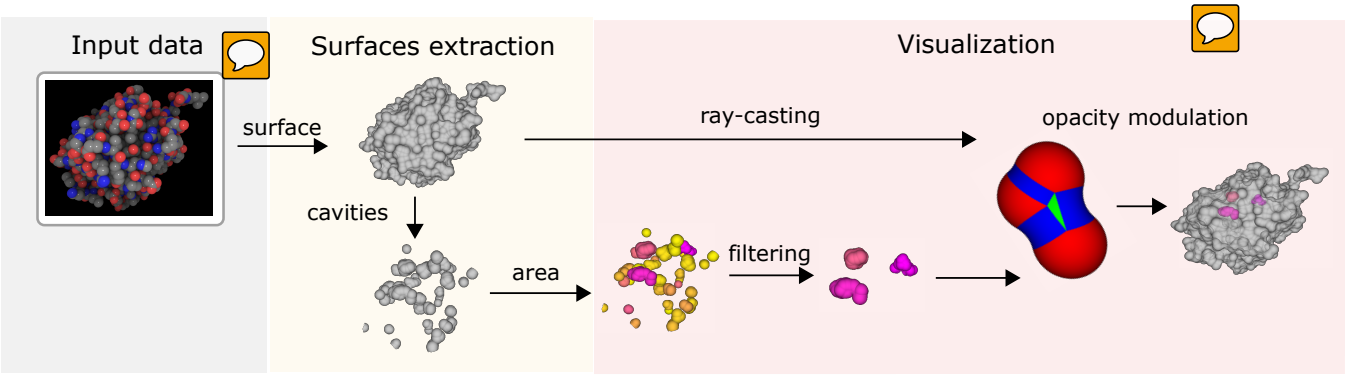

Figure 2: Illustration of the visualization pipeline. The input data (Sec. 3) contains snapshots of MD trajectories which are processed to construct the molecular surface and cavities (Sec. 4). Then, the surface areas of cavities are estimated (Sec. 5.3), which are used as the color codes, ranging from yellow (smallest) to magenta (largest). These areas serve for filtering out of too small cavities. Finally, the surface elements are ray-cast (Sec. 5.1) to compose the surface fragments used in the final stage to visualize the surfaces transparently via a user-defined opacity modulation (Sec. 5.2).

ing the surface, each set is used to ray-cast a spherical patch (see Sec. 5.1).

4.2 Cavity Computation

The computed surface contains patches of both the outer molecular surface and the surfaces of inner cavities. Kauker et al. [13] aimed at visualizing the outer surface solely. Therefore, they call the inner parts of the surface as inner remains as they represent the source of occlusion. However, in our case the inner cavities are important as well. Additionally, it is advantageous for the user to have the option to visualize both molecular surface and inner cavities and interactively alter between both of them, i.e., the cavities can be shown or hidden on demand.

Hiding the cavities is straightforward, since their surface is completely separated from the molecular surface. More specifically, one component represents the outer molecular surface and each single cavity corresponds to another component. Such independent components can be easily detected by applying connected component (CC) analysis to a structure which we call the *surface graph*. We define the surface graph using the triangles representing vertices and the toroidal patches representing edges (Fig. 5). In this graph, all the vertices are of degree three. In fact, the surface graph is an analogy to the graph of SAS (solvent-accessible surface) contours [32].

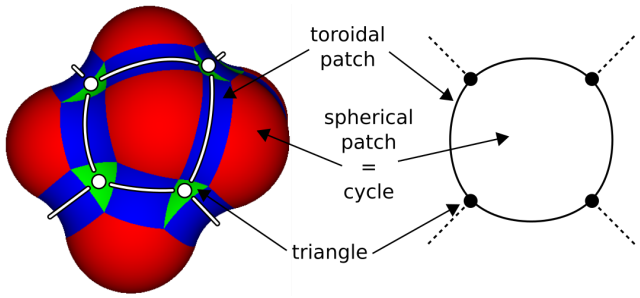

Figure 5: Illustration of the surface graph construction. Triangles form vertices and toroidal patches form edges between them. Spherical patches are represented as cycles in the graph.

Finally, the surface contains also tori that are not bounded by any triangles. We call these tori *isolated* since they do not have any neighboring triangles (Fig. 6). The isolated tori are excluded from the surface graph and are handled later in the computation (see Sec. 4.2.1).

The whole computation is done on the GPU in order to avoid additional data transfers between the main memory and the graphics card. We split the analysis of the surface into three steps:

1. Building the adjacency list of surface graph $G = (E,V)$.
2. Finding all connected components of G.
3. Finding cycles in G that represent patches on spheres.

For each step, we introduce a GPU kernel implemented as a GLSL compute shader.

First, we build the adjacency list of G from the set of edges E. We store the adjacency list of G as a $|V| \times 3$ matrix, since each vertex has three neighbors. In fact, E is prepared earlier in the surface computation phase, when we compute the toroidal patches for ray-casting. To compute these patches, we determine their delimiting triangles, thus knowing incident vertices of edges of G. For each non-isolated toroidal patch an edge e is added to E by writing its incident vertices into a buffer of edges. The first kernel then transforms the buffer of edges E into a matrix representing the adjacency list of G. The matrix has one row for each vertex $v \in V$ which stores neighbors of v as indices into the buffer of vertices V.

In the second step, all connected components of G are found and labeled using the breadth-first search (BFS) algorithm. We opt for a simple, quadratic-work implementation [24], because of the ability to have the BFS implemented in a single kernel. Our decision is supported by the performance measurements (see Sec. 6.1) where the computation of surface components takes ~5 ms for a molecule with ~10,000 atoms while the computation of SES and its ray-casting together takes ~41 ms.

In the last step, the cycles representing the spherical patches are extracted. To do this, we assign the edges in E by their neighboring spheres into buckets. We then order the edges in buckets by their adjacency, so that they represent one or more cycles in G. A spherical patch is labeled by the label of any of its delimiting edges.

4.2.1 Special Case – Isolated Torus

We handle the isolated tori in two steps. First, a label for a torus is retrieved. Note that an isolated torus is not a part of the surface graph. Second, for each isolated torus, we clip fragments in its neighboring spherical patches that are overlaid by the torus. Otherwise, these fragments would create a false surface layer between a torus and a spherical patch (Fig. 6).

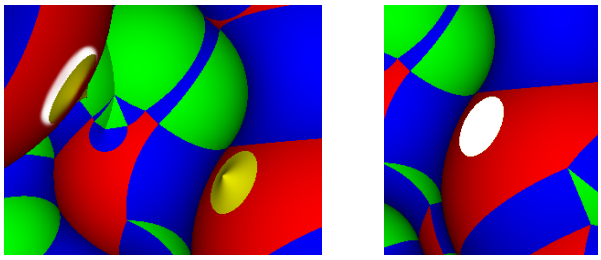

Figure 6: Example of an isolated torus between two spheres. Left: isolated tori (yellow) do not have any neighboring triangles. Therefore, the surface component to which they belong cannot be determined directly from the surface graph. Right: an isolated torus cuts circular holes in its neighboring spherical patches.

An isolated torus can be encountered while writing the toroidal patches for ray-casting. We add this torus into the buffer for ray-casting and we also mark it for later processing. Later on, when the detection of surface components is done, we run another kernel which assigns the isolated tori to their surface labels. The assignment is done by inspecting the neighboring spheres of the tori. There can be two cases:

1. One of the spheres forms only one patch. Then the torus lies in the same surface component as this patch.

2. Both spheres form two or more patches. We find a neighboring patch of the torus by employing the point in a spherical patch test (see Sec. 5.1). Then the torus lies in the same surface component as its neighboring patch.

The clipping of fragments in the affected spherical patches can be reached using a clipping plane or a *visibility sphere* (introduced in [15]) of a torus. We use the latter approach, since the visibility spheres for tori were already computed in a previous phase of our technique. Here, our kernel writes a texture, similarly to Krone et al. [16], which stores all visibility spheres that intersect the sphere. When ray-casting spherical patches, we use this texture to discard all fragments that are inside of the intersecting visibility sphere.

4.2.2 Special Case – Intersecting Surfaces

The SES surface, more specifically its part that is formed by spherical triangles, can intersect itself. These self-intersections were solved by Krone et. al [15]. In our case, we compute one or more SES surfaces that can mutually intersect in the same way. We handle two different cases of such surface-surface intersections:

- The outer surface is intersected by a cavity. We clip only the surface of the cavity to prevent the outer surface from containing holes.

- A cavity is intersected by another cavity that will not be visible. We do not clip the visible cavity, otherwise we would create a hole in its surface.

We implemented the handling of these two cases by extending the self-intersections handling algorithm described in [16].

5 Visualization

To enable the transparent visualization, we require an ordered list of fragments for each screen pixel. This list of fragments is also commonly referred as an A-buffer. We implemented the A-buffer using per-pixel linked lists [35]. In our case, the A-buffer fragments represent three types of surface patches analytically computed for a given ray.

5.1 Ray-casting

To form fragments of each surface patch we employ a ray-casting technique. To reach high performance, the ray-patch intersection is computed analytically. For each patch, we compute a tight oriented bounding box (OBB) which we project onto the image as a splat (Fig. 7). This way, we lower the number of intersection tests that will be performed.

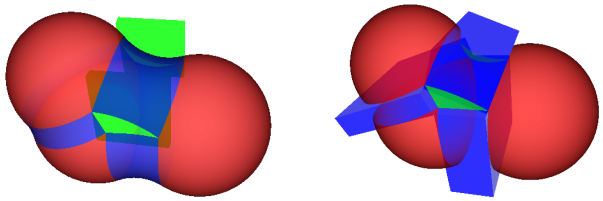

Figure 7: Generation of bounding boxes for the SES patches. Left: OBBs for spherical triangles. Right: OBBs for toroidal patches.

The ray-casting is implemented using GLSL shaders. The triangles for the OBB faces are generated on the GPU using a geometry shader. The OBB is computed from a unit cube for which we compute position, rotation, and scale according to a patch that will by bounded by it. For a spherical triangle, a cutting plane that contains the triangle's vertices is computed. Then, the OBB's position, rotation, and scale are computed such that the OBB bounds the smaller part of the sphere when cut by the plane. Here, we assume that the spherical triangle always fits into a hemisphere. The OBBs for spherical patches are computed in a similar way. The difference is in the calculation of the cutting plane. The plane's normal is computed using midpoints of arcs that bound a patch. The end points of these arcs are used to position the cutting plane so the half-space defined by it contains all the arc end points.

For the toroidal patches the orientation of their OBBs is computed first. We compute a vector defined by the centers of the spheres that contain bounding triangles of a toroidal patch. We use this vector together with the axis of a torus as a second vector to compute a third one – perpendicular to the two vectors. These three vectors define the axes of an OBB. Finally, the extreme points of a patch w.r.t. its OBB orientation are found and the OBB's position and scale are computed based on these points.

In 2013, Kauker et al. [13] proposed to ray-cast a toroidal patch using a saddle part of a torus and two clipping planes defined by its delimiting triangles (Fig. 8).

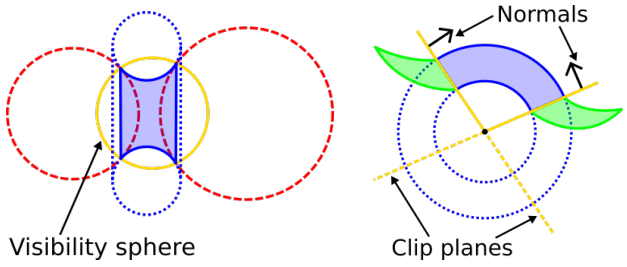

Figure 8: Ray-tracing a toroidal patch. Left: the saddle part of the torus (blue) is cut by the visibility sphere (yellow). Right: the patch (blue) is cut from the whole toroidal ring by clipping planes (yellow) defined by the spherical triangles (green).

We employ this approach and, moreover, we compute these clipping planes for a toroidal patch when writing it into a buffer for ray-casting. Additionally, we set normals of these clipping planes so that they are facing towards the toroidal patch and we define two

types of clipping operations: *AND* and *OR*. We use these two operations to clip the patches whose angular length around the torus axis is either at most π (*AND*) or greater (*OR*). The *AND* and *OR* operations are defined as their name suggests. The *AND* operation leaves only fragments that appear on the positive sides of both clipping planes, while the *OR* operation leaves fragments that appear on the positive side of the first or the second clipping plane. The ray-torus intersection is computed as described by Herbison-Evans [12].

Ray-casting the sphere is a trivial task. However, there can be one or more spherical patches that belong to the same surface atom sphere but they may form different surfaces. To avoid obvious rendering issues, i.e., distinct coloring, transparency, or visibility, we perform the ray-casting of each patch separately. Regarding the separation of patches, we experimented with applying an odd-even rule for polygons [29] and with computing a convex polyhedron bounds. We decided for the bounding volume solution because regarding the point-in-polygon (PIP) test there were non-trivial special cases, e.g., a ray going through a polygon's vertex or a very short edge. These special cases then resulted in pixel artifacts in the final image. Moreover, the implementation of the PIP test was performance demanding due to many texture memory accesses. On the other hand, the computation of a patch's bounding volume is based on the fact, that the spherical patch is bounded by planes defined by its bounding arcs. This way, we have a polyhedron B_{poly} which bounds all the patches belonging to an atom sphere. For each patch, we add another plane p_{add} which cuts B_{poly} in order to keep only the volume containing that patch. p_{add} is selected as an appropriate plane among planes of the OBB that was used for splatting the patch. In fact, by splatting, we clip B_{poly} by all the six planes of the OBB.

In the resulting image, there can still occur pixel artifacts caused by numerical issues when ray-casting the tori. These numerical issues arise when computing an intersection of a ray with a torus.

5.2 Opacity Modulation

After the intersection points are computed, we sort them in a front-to-back manner and store them to the linked list. Thus, for a given ray, we acquire a list of fragments $\{f_1,\ldots,f_n\}$, where each pair represents an entry and an exit point for the molecular surface. For simplicity, the values of f_i represent the depths of fragments. We denote the entire distance the ray passes through the molecule as $l = |f_1 - f_n|$. As we step along each fragment f_i, we define the opacity α_i of even fragments, i.e., those representing the entry surface points, as follows:

$$\alpha_i = O^{\phi(x)}, \tag{1}$$

where O represents a user-defined parameter affecting the overall opacity and $\phi(x)$ suppress or amplifies the opacity and is defined as

$$\phi(x) = K - (K-1)x, \tag{2}$$

where K is the maximum value of the exponent and $x = |f_{i+1} - f_i|/l$ represents the ratio of the fragment interval to the entire length l. Note that if $x = 1$, i.e., having just two fragments on the ray, then $\phi(x) = 1$ determining the $\alpha_i = O$. The opacity of odd fragments, i.e., representing the exist surface points, is defined as $\alpha_i = O$, thus keeping them unmodulated since they are less prominent in the final image. Figure 9 showcases four different combinations of parameters K and O.

5.3 Cavity Area Estimation

We enhance the visualization of cavities by coloring their surface by their approximate areas. To estimate the area, we sum the areas of all triangles that form the cavity surface. The area of a spherical triangle is calculated as follows:

$$S = r_{probe}^2\,[(A+B+C) - \pi], \tag{3}$$

where A, B, and C are angles of the triangle. Additionally, we neglect areas of spherical and toroidal patches since their influence on the exact cavity area is much smaller compared to triangles. Naturally, this observation does not hold for the molecular surface. We also do not handle the case when the triangles intersect each other thus they are clipping themselves.

6 Results

Here we present the performance analysis of our algorithm, which includes discussion of the limitations of our technique and also discussion of the improved memory complexity we implemented. Further, we introduce an informal evaluation and feedback we obtained from the domain experts.

6.1 Performance Analysis

We tested our technique on commodity hardware to show that it enables the users to solve their tasks in real-time without high hardware requirements. The tests were performed using Intel Core i5 760 (2.80 GHz) with 4 GB of RAM and NVIDIA GeForce 680 GTX with 4 GB of VRAM as a graphics card. For rendering, we used the resolution of 1024×768 while fitting the molecule to cover the most of the rendered image. The results of our measurements for both static and dynamic molecules are presented in Table 1. We choose the static structures from the Protein Data Bank (PDB) [30] so the performance of our technique could be easily compared with other existing or new approaches on the same dataset.

In Table 2, we compare an overall performance of our technique with the method presented by Kauker et al. [13] as it is currently the only existing method that enables a correct transparency of the SES in real-time. Comparing to the state-of-the-art methods [20, 16] for SES visualization, our method performs in the surface computation part similarly to the algorithm presented by Krone et al. [16] (see Table 1). Nevertheless, the performance of the ray-casting cannot be directly compared to these two methods, since we test twice as many intersections to obtain both front and back fragments and our intersection tests (especially for spherical patches) need to be more complex than those used by techniques that do not render the surface using full transparency.

Table 1: Performance of our technique for static and dynamic (*) structures. Table shows the timings of the key phases of our method: surface computation (CB), cavity computation (SG), area estimation, and ray-casting (RC). For ray-casting, we also include fill rate (FR). The performance of our technique does not vary significantly for static or dynamic data as it operates on a per-frame basis.

Molecule	# Atoms	FR (%)	CB (ms)	SG (ms)	Area (ms)	RC (ms)	Total (FPS)
1OGZ	~1000	41.5	3.5	0.3	0.2	5.9	47.8
1VIS	~2500	50.5	5.9	0.8	0.2	9.5	34.1
LINB-ACE*	~4500	29.8	17.2	0.6	0.2	9.3	22.6
4ADJ	~10000	43.1	21.0	4.1	0.4	20.2	15.2

Our technique was implemented mainly using OpenGL employing GLSL compute shaders as GPU kernels. We also used OpenCL to implement a kernel which computes the positions of spherical triangles in the original algorithm [16]. The GLSL implementation of this kernel performs for a molecule with ~10,000 atoms about 5x slower than the OpenCL/CUDA one. Based on the performed tests we assume that the key reason for this performance loss is the

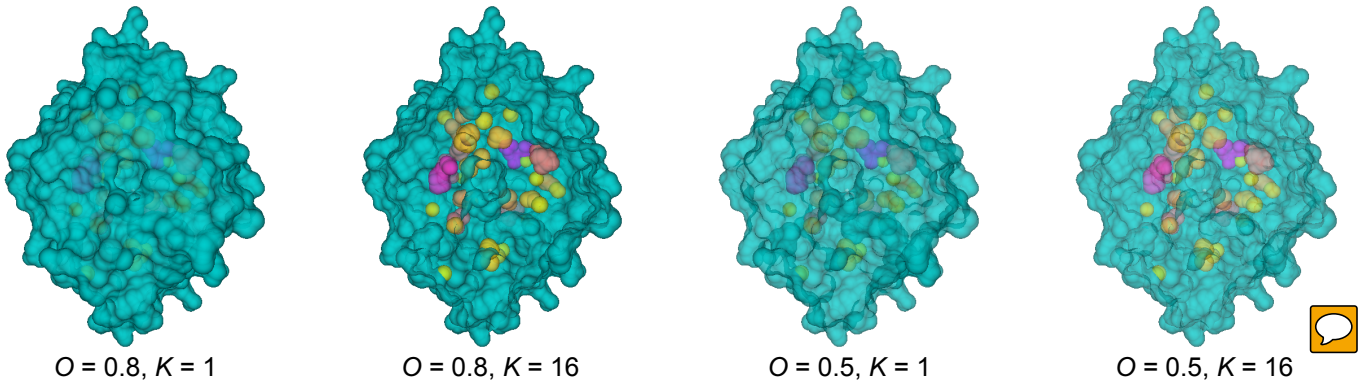

Figure 9: Example of applying parameters K and O to protein with PDB ID $1CQW$. Note that higher values of the overall opacity O emphasize the front molecular surface, while higher values of maximum exponent K give more prominence to the internal surfaces and cavities.

Table 2: Speedup of our technique in comparison with Kauker et al. [13]. Table shows the number of depth layers (DL), rendering speeds (FPS) of both methods, and the relative speedup. Fill rate (FR) which significantly affects the performance of ray-casting is also included.

Molecule	# Atoms	Our approach			Kauker et al.			Speedup
		FR (%)	DL	FPS	FR (%)	DL	FPS	
1YV8	~650	38.6	12	40.1	18.0	117	31.0	1.29
1VIS	~2500	52.7	15	34.1	48.7	135	11.2	3.04
4ADJ	~10000	41.5	19	20.2	42.8	188	6.2	3.26

extensive use of the global memory in the kernel, which is handled differently in OpenGL and OpenCL/CUDA.

6.1.1 Discussion of Limitations

The main limitations of our method are currently the applicability to large data and the precision of the surface area calculation. When applying it to molecules consisting of more than $10,000$ of atoms, the performance of the method influences the interactivity. We anticipate that the performance can be improved by applying the transparent SES only to some interesting parts of the molecule, while using another visualization method for the rest of it. The second limitation can be overcome, e.g., by applying the method presented by Connolly [6], to be done as a future work. The method also evinces single-pixel artifacts (Sec. 5.1) in the final image which can cause problems when creating high-quality images.

6.1.2 Improved Memory Complexity

As a side effect of exploiting the hash data structure (Sec. 4.1) for storing the spherical triangles, our technique consumes less GPU memory than the original approach [16]. This is due to the fact that the original method uses a linear buffer to store the spherical triangles. The index into this buffer, i.e., the position where a triangle is stored, is computed by means of i and j indices of spheres $i < j < k$ that formed the triangle. The range of j is optimized in terms of limiting the maximal number of neighbors ($maxNeighbors$) that a sphere can contain, so that the range of j can be remapped to $[0, maxNeighbors)$. There is also a limit on the total number of triangles ($maxTriangles$) that can be stored for each pair of spheres. Putting it all together, the memory complexity of the original data structure is:

$$|A| \cdot maxNeighbors \cdot maxTriangles, \tag{4}$$

where $|A|$ is the number of atoms of the molecule and the typical settings for the other two constants are $maxNeighbors = 64$, $maxTriangles = 64$. We adopted this settings from the MegaMol visualization framework [10].

On the other hand, we store the computed triangles linearly in an array and for each triangle we additionally store three hash records. The memory complexity of our hash structure is therefore:

$$|T| + hashFreeRatio \cdot 3|T|, \tag{5}$$

where $|T|$ is the number of computed spherical triangles and the typical setting for the constant is $hashFreeRatio = 2$.

To be able to compare the complexity of these two data structures, we estimate the ratio between the number of atoms and the number of triangles to be (taken from experiments):

$$\frac{|A|}{|T|} > \frac{1}{4}. \tag{6}$$

Then, the complexity ratio can be expressed as:

$$\frac{|A| \cdot maxNeighbors \cdot maxTriangles}{|T|(1+3 \cdot hashFreeRatio)} > \frac{64^2}{28} > 146, \tag{7}$$

which yields that the original structure is more than 100 times larger for only 64 neighbors. Actually, our maximum neighbor count is 128 to be able to compute the surface of molecules that contains hydrogens, which is a typical case for data produced by molecular dynamics simulations.

6.2 User Feedback

The resulting visualization technique was evaluated by three domain experts working in the field of protein engineering. They regarded the solution as highly practical with respect to the exploration capabilities of molecular surfaces and their inner cavities, mainly due to the time spent on the given task, which in our case was reduced dramatically. They also agreed that this approach has a big potential for studying protein-protein interactions where the reactions take place on the protein surfaces. Therefore, studying shapes of these surfaces and their change over time is considered to be very important. Moreover, they appreciated the visual appearance that was found more appealing than the existing solutions. They confirmed that our approach can be directly used for creating presentation materials, such as images to publications or animations for presentations.

The domain experts were also asked to identify the weak points of our current solution. They pointed out that using a highly transparent molecular surface complicates the assessment of the position of penetrating structures (ligands) when using only one viewpoint. In other words, the user needs to adjust the scene camera in order to decide whether the ligand is in the outer solvent or whether it has already penetrated the protein surface. Nevertheless, this ambiguity is caused by the transparency in general and solution could be achieved by its combination with other techniques, which opens possible directions for the future extension. Another potential extension suggested by the biochemists is to enable a selection of an interesting cavity (e.g., containing the active site) and focus on its changes. As the volume and area of the detected cavities are important parameters for the biochemists, they also proposed to replace the current approximated values by exact ones. According to the domain experts, our algorithm could be also extended to be applicable to other voids in proteins, namely tunnels.

7 Conclusion

In this paper we presented an accelerated and improved rendering pipeline for the real-time computation and visualization of the transparent molecular surface and inner cavities. We also improved the existing state-of-the-art method to the SES computation by proposing new methods for computation and rendering of individual SES patches. The usability of our solution was tested by the domain experts. They also revealed possible extensions of our solution which are discussed as well. Other possible extensions are related to the performance and the accuracy.

Acknowledgements

This work was supported through grants from the Norway grants project NF-CZ07-MOP-2-086-2014, and the PhysioIllustration research project 218023 funded by the Norwegian Research Council.

References

[1] D. A. F. Alcantara. *Efficient hash tables on the GPU*. University of California at Davis, 2011.

[2] A. Bair and D. H. House. Grid with a view: Optimal texturing for perception of layered surface shape. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1656–1663, 2007.

[3] L. Bavoil and K. Myers. Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK*, pages 1–12, 2008.

[4] D. Borland. Ambient occlusion opacity mapping for visualization of internal molecular structure. *Journal of WSCG*, 19(1):17–24, 2011.

[5] L. Carpenter. The A-buffer, an antialiased hidden surface method. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 103–108, 1984.

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

[7] H. Edelsbrunner, M. A. Facello, P. Fu, and J. Liang. Measuring proteins and voids in proteins. In *HICSS (5)*, volume 5, pages 256–264. IEEE Computer Society, 1995.

[8] E. Enderton, E. Sintorn, P. Shirley, and D. Luebke. Stochastic transparency. *IEEE Trans. Vis. Comput. Graph.*, 17(8):1036–1047, 2011.

[9] C. Everitt. Interactive order-independent transparency. *White paper, nVIDIA*, 2(6):7, 2001.

[10] S. Grottel, M. Krone, C. Muller, G. Reina, and T. Ertl. Megamol – a prototyping framework for particle-based visualization. *IEEE Trans. Vis. Comput. Graph.*, 21(2):201–214, 2015.

[11] U. Hensen, T. Meyer, J. Haas, R. Rex, G. Vriend, and H. Grubmüller. Exploring protein dynamics space: The Dynasome as the missing link between protein structure and function. *PLoS ONE*, 7(5), 2012.

[12] D. Herbison-Evans. Solving quartics and cubics for graphics. In *Graphics Gems V*. Morgan Kaufmann, 1995.

[13] D. Kauker, M. Krone, A. Panagiotidis, G. Reina, and T. Ertl. Rendering molecular surfaces using order-independent transparency. In *Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization*, pages 33–40. Eurographics Association, 2013.

[14] B. Kozlíková, M. Krone, N. Lindow, M. Falk, M. Baaden, D. Baum, I. Viola, J. Parulek, and H.-C. Hege. Visualization of biomolecular structures: State of the art. In R. Borgo, F. Ganovelli, and I. Viola, editors, *Eurographics Conference on Visualization (EuroVis) - STARs*. The Eurographics Association, 2015.

[15] M. Krone, K. Bidmon, and T. Ertl. Interactive visualization of molecular surface dynamics. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1391–1398, 2009.

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

[17] M. Krone, D. Kauker, G. Reina, and T. Ertl. Visual analysis of dynamic protein cavities and binding sites. In *IEEE Pacific Visualization Symposium, PacificVis 2014, Yokohama, Japan, March 4-7, 2014*, pages 301–305, 2014.

[18] K. Lawonn, M. Krone, T. Ertl, and B. Preim. Line integral convolution for real-time illustration of molecular surface shape and salient regions. *Comput. Graph. Forum*, 33(3):181–190, 2014.

[19] N. Lindow, D. Baum, A. Bondar, and H. Hege. Exploring cavity dynamics in biomolecular systems. *BMC Bioinformatics*, 14:S5, 2013.

[20] N. Lindow, D. Baum, S. Prohaska, and H.-C. Hege. Accelerated visualization of dynamic molecular surfaces. In *Proceedings of the 12th Eurographics / IEEE - VGTC Conference on Visualization*, pages 943–952, Chichester, UK, 2010.

[21] A. Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *Computer Graphics and Applications, IEEE*, 9(4):43–55, 1989.

[22] M. Maule, J. L. Comba, R. Torchelsen, and R. Bastos. Memory-efficient order-independent transparency with dynamic fragment buffer. In *Graphics, Patterns and Images (SIBGRAPI), 2012 25th SIBGRAPI Conference on*, pages 134–141. IEEE, 2012.

[23] M. McGuire and L. Bavoil. Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques (JCGT)*, 2(2):122–141, 2013.

[24] D. Merrill, M. Garland, and A. Grimshaw. Scalable GPU graph traversal. In *ACM SIGPLAN Notices*, volume 47, pages 117–128. ACM, 2012.

[25] J. Parulek, C. Turkay, N. Reuter, and I. Viola. Visual cavity analysis in molecular simulations. *BMC bioinformatics*, 14(Suppl 19):S4, 2013.

[26] J. Parulek and I. Viola. Implicit representation of molecular surfaces. In *Proceedings of the 2012 IEEE Pacific Visualization Symposium*, PACIFICVIS '12, pages 217–224, Washington, DC, USA, 2012. IEEE Computer Society.

[27] M. Salvi, J. Montgomery, and A. Lefohn. Adaptive transparency. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pages 119–126. ACM, 2011.

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

[29] M. Shimrat. Algorithm 112: position of point relative to polygon. *Communications of the ACM*, 5(8):434, 1962.

[30] J. L. Sussman, D. Lin, J. Jiang, N. O. Manning, J. Prilusky, O. Ritter, and E. Abola. Protein data bank (PDB): database of three-dimensional structural information of biological macromolecules. *Acta Crystallogr., Sect D: Biol. Crystallogr.*, 54(6):1078–1084, 1998.

[31] M. Tarini, P. Cignoni, and C. Montani. Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Trans. Vis. Comput. Graph.*, 12(5):1237–1244, 2006.

[32] M. Totrov and R. Abagyan. The contour-buildup algorithm to calculate the analytical molecular surface. *Journal of structural biology*, 116(1):138–143, 1996.

[33] A. Varshney, F. P. Brooks, Jr., and W. V. Wright. Linearly scalable computation of smooth molecular surfaces. *IEEE Computer Graphics and Applications*, 14(5):19 – 25, 1994.

[34] A.-A. Vasilakis, G. Papaioannou, and I. Fudos. k^+-buffer: an efficient, memory-friendly and dynamic k-buffer framework. *IEEE Trans. Vis. Comput. Graph.*, 21(6):688–700, June 2015.

[35] J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz. Real-time concurrent linked list construction on the GPU. *Computer Graphics Forum*, 29(4):1297–1304, 2010.