Real-time Visualization of Protein Surface Features in Molecular Dynamics

Category: Research

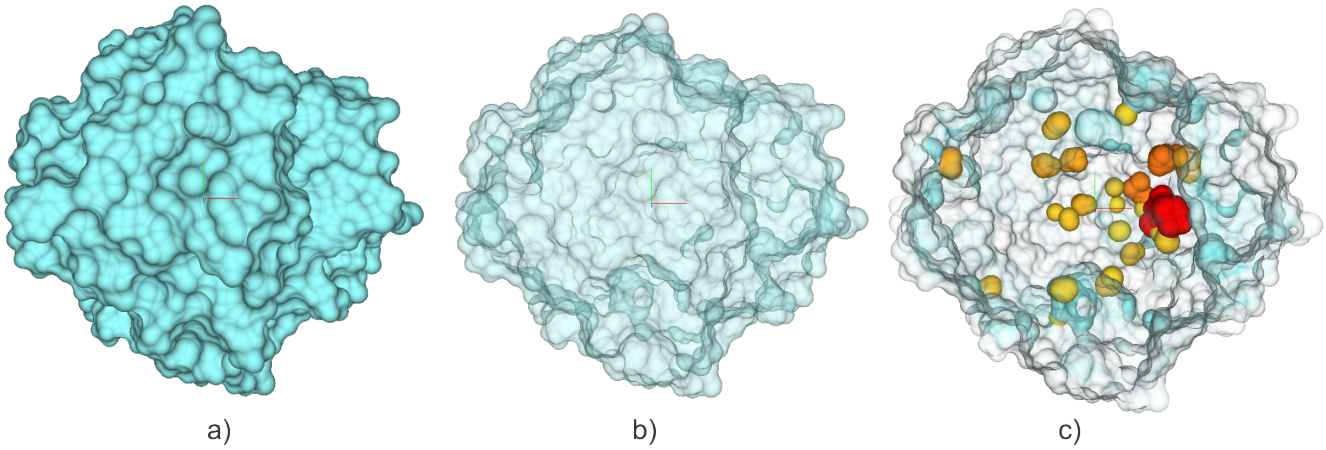


Figure 1: An example of protein XXX demonstrating our method. a) SES b) transparency c) transparency and cavities.

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 extraction of their structural features. In this paper we introduce such technique which enables the user to compute and visualize the molecular surface along with inner voids instantly and exactly???. To obtain a better insight into the molecule, our technique enables to visualize the molecular surface transparently. The whole visualization is created in a focus and context visualization manner, where the user can interactively choose the structures of interest. 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 is demonstrated by a case study which reflects real problems of the domain experts.

Index Terms: I.3.7 [Three-Dimensional Graphics and Realism]: Hidden line/surface removal—; I.3.8 [Applications]—

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 [8]. 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 or offline rendering which is currently their only option. This is especially crucial when analyzing MD trajectories containing thousands of frames, where the users cannot spent much time analyzing just a single frame either due to computational or visualization limitations. Moreover, the exploratory process of MD trajectories 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 their visualization in the most informative manner.

In our approach we are focusing mainly on cavities which represent the protein inner void space directly inaccessible from the molecular surface. We are facing the mentioned 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.

More specifically, the main contributions of our solution are:

- Improved representation of the solvent-excluded surface (SES) of the molecule. We compute the exact representation of individual surface primitives.
- A novel real-time algorithm for the extraction of cavities. We present new fast cavity extraction method that is based on solvent-excluded surface (SES).
- Improved performance of visualization of transparent molecu-

lar surfaces [9]. We describe our interactive focus and context visualization of cavities within the molecule.

!!!Currently existing methods provide a solution for each of these topics separately (i.e., computation of molecular surface, its transparency, and inner cavities). However, none of these solutions combines all these topics in order to provide the domain experts with a tool for their real-time visualization and exploration. We fill this gap by introducing our solution which not only provides the users with the combination of the mentioned contributions but even 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 using the order-independent transparency approach. Thus in this section we describe the existing techniques related to these topics.

2.1 Extraction and Visualization of Surfaces of Molecules and Their Cavities

There are several types of molecular surface representation proposed in the literature [10]. However, for the cavity analysis, the solvent-excluded surface (SES) is the most commonly used representation used by the domain experts.This representation allows us to directly asses whether a solvent, approximated by a sphere of a given radius, is able to reach a binding site of interest on the molecular surface.

The area of geometrical analysis of molecular structures focusing on the detection and further exploration of the void space is very vast and so we do not aim to provide the users with an exhaustive list of the existing solutions. Rather we focus on the techniques which we consider to be the closest to our work.

The molecular surface is one of the most significant studied features thus many solutions were published. Here we focus on the analytical approaches for the construction of the SES where Connolly [5] presented the first solution to this problem. Another approach to the detection of the analytic surface was later presented by Totrov and Abagyan [22]. The algorithm is based on the sequential build up of multi-arc contours. One of the advantages of this contour-buildup approach is that it is localized thus applicable on the computation of partial molecular surface.

One of the significant improvements of the SES computation was published by Parulek and Viola [19]. 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. [11] presented their approach to the visualization of the SES using GPU ray casting technique which allowed to achieve interactive frame rates. TODO: how it is in comparison with our solution? Another approach by Lindow et al. [13] even accelerates the construction of the SES by scaling the parallel contour-buildup algorithm to more GPU cores and using boundary quadrangles as rasterization primitives. TODO: how it is in comparison with our solution?

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. [18] 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. TODO: what else?

2.2 Order-Independent Transparency

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) [14] or A-buffer [4]. 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. The A-buffer uses the Fourier window which in general helps to resolve the visibility problem among an arbitrary collection of opaque, transparent, and intersecting objects.

While the algorithms for rendering of correctly transparent objects in real-time have emerged already in the previous decade [7], new approaches removing the limitations of these methods are still emerging. Bavoil et al. [2] introduced the peeling of two depth layers in one rendering pass, thus halving the time complexity of the original Depth Peeling approach [7].

!!!For us, performing more rendering passes, would become a bottleneck because of our ray-tracing approach to surface rendering. !!!Instead, techniques based on A-buffer enable to render the scene in one pass, thus fitting better to our approach.

When using single pass rendering, it is necessary to store all participating or all important fragments into a data structure for later composition. First approach storing the participating fragments was introduced by Yang et al. [24]. Salvi et al. [20] presented the second mentioned approach which stores all important fragments. This leads to an unbounded requirement on the memory size that was addressed by Maule et al. [15] or Vasilakis et al. [23]. Finally, some of the existing techniques aim to decrease both time and memory complexity. McGuire and Bavoil [16] employ specific coloring of objects to reach this goal, whereas Enderton et al. [6] introduce statistics for the same purpose.

Applying the mentioned OIT approaches to molecular surfaces is straightforward because they are represented as meshes. For analytic surfaces, Kauker et al. [9] proposed to store lists (or arrays) of all fragments produced by the basic shapes for later processing using CSG operations. In our technique, we try to avoid the storing and subsequent sorting of a high number of fragments by ray-casting only the fragments that belong to the molecular surface.

!!!For enhancing the understanding of the interior of the molecular structure, we employed the technique presented by Borland[3]. His 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 Order-Independent Transparency for Contour-Buildup

Overview (0.5 page) (AJ, JP)

- "Technical Part" (AJ)
- SES
- Cavity
- Ray-casting + parameters for vis

3.1 Overview

The computation of SES is not a trivial task, which also requires a substantial computation and algorithmic capabilities. Therefore, it would be essential to posses a technique that could provide us with instant computation and interactive and meaningful visualization of cavities in the context of molecular surface.

The data comes in a form of MD trajectories describing the motion of individual atoms. Each trajectory time step includes a set

of atoms, described by their positions and their 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 primitives, inner voids, and additional attributes used in later stages. The second group of computations relates to visualization. This includes ray-casting of the formed primitives, the estimate of ambient occlusion and opacity calculation; before the final stage represented by the image formation. More specifically, we perform the following steps (TODO need more detail from Adam):

1. We employ the contour-buildup algorithm to construct the molecular surface. Additionally, we enhance the computation as well as the performance (only memory efficiency, bit in the CB article, they do not describe their data structures. I inspected the MegaMol code.) by several changes (see Sec. 3.2).

2. For cavity extraction we introduce the so-called *surface graph* (Sec. 3.3), which allows us to compute the internal voids, i.e., cavities.

3. Each one of the three primitives Bara: which primitives?? you are talking about them later is visualized using ray-casting (Sec. 5). (ray-casting, opacity calculation, etc.)

4. We compute ambient occlusion, stored in the grid.

5. We perform transparent surface rendering by means of an A-buffer. The actual opacity of each surface element is modulated through ambient occlusion values.

6. In the last stage, we blend all acquired samples together.

3.2 Extended CB Algorithm

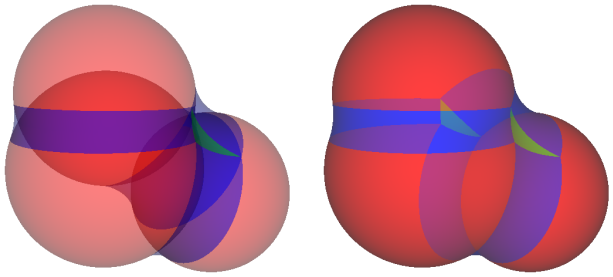
Our extended CB algorithm is based on the existing research that utilizes the GPU capabilities [12]. In the former study, the SES is represented as three different sets of surface primitives – spheres, tori, and spherical triangles, that are ray-casted to obtain opaque surface visualization. While ray-casting spheres and tori, there are also produced pixels that are not visible in the final image, because of their occlusion by pixels on the surface (see Figure 3a). To visualize the SES transparently, we extended the surface computation part of the algorithm such that it computes these primitives:

- Spherical triangles.
- Toroidal patches – a toroidal patch is delimited by two spherical triangles.
- Spherical patches – a spherical patch is enclosed by edges of three or more toroidal patches.

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

Regarding tori, Kauker et al. [9] proposed to ray-cast a toroidal patch using a torus and two clipping planes defined by its delimiting triangles.

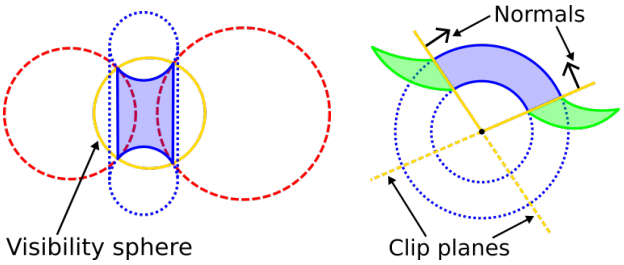
We employ his approach and modify the data structure that is used in the original algorithm to store spherical triangles in such a way that we are able to obtain all triangles incident to a torus. In order to get all neighboring triangles for a torus, we hash the triangles by three keys – one for each torus which is connected to a triangle. For this, we implemented a simple hash table (see Figure 9) which is based on linear addressing scheme [1].



(a) The original method [12] rendered with transparency.

(b) Our extended method.

Figure 3: Comparison between the original method rendered with transparency and our extended method. The original method (a) renders parts of spheres (red) and tori (blue) that lie below the surface. Our method (b) produces only primitives that are part of the surface.



(a) Clipping by *visibility sphere*.

(b) Clipping by planes defined by spherical triangles.

Figure 4: Ray-tracing of a toroidal patch. The saddle part of the torus (a) is cut by so called *visibility sphere*. A patch (b) is cut from the whole toroidal ring by clipping planes.

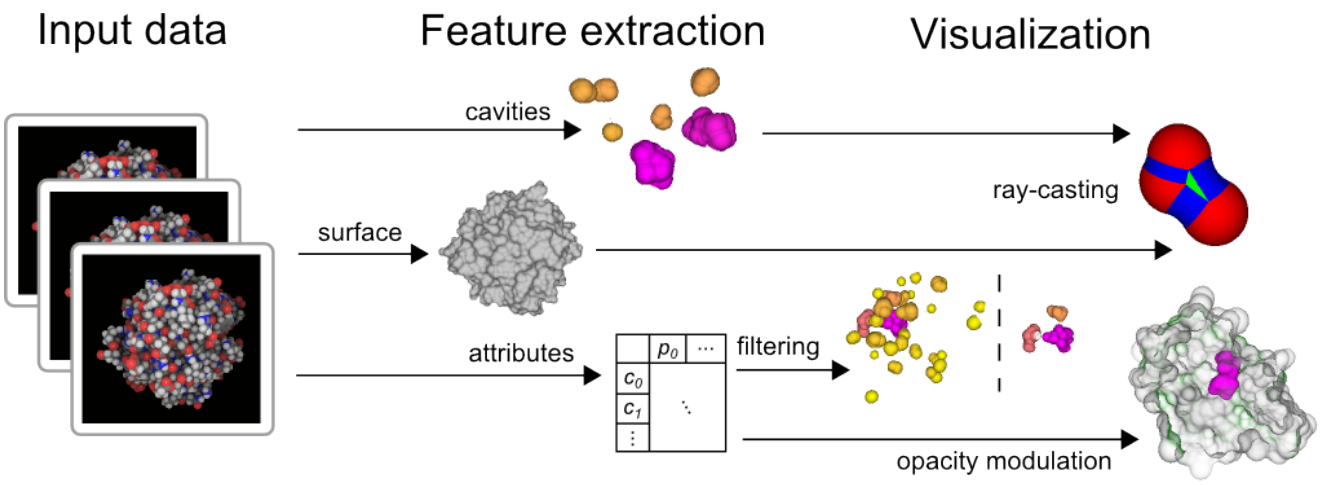


Figure 2: TODO.

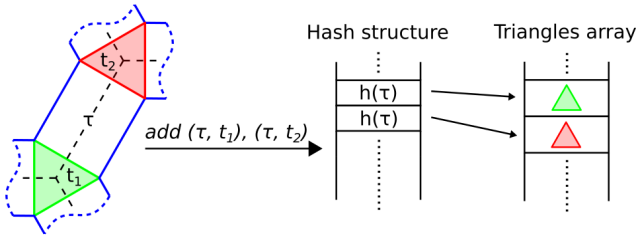


Figure 5: Our novel data structure for storing spherical triangles. Triangles t_1 and t_2 are stored linearly in an array and their incident torus τ is connected to them using a hash table.

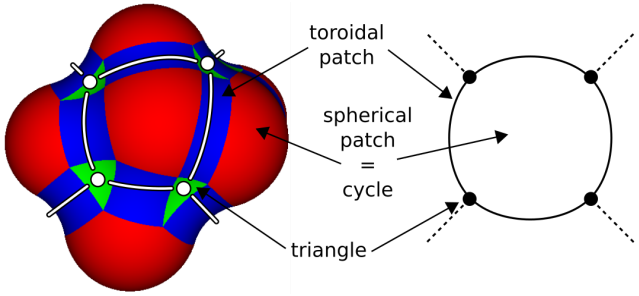


Figure 6: The surface graph. Triangles form vertices and toroidal patches form edges between them. Spherical patches are represented as cycles in the graph.

Since a spherical patch is bounded by toroidal patches, we extended the original algorithm by adding three new GPU kernels that compute the sets of toroidal patches forming the spherical patches (see Section 3.3). When visualizing the surface, each such set is used to ray-cast a spherical patch (see Section 5.1).

3.3 Surface graph

The computed surface contains primitives of both the outer molecular surface and the surfaces of inner cavities. [1] Kauker et al. [9] were in their study interested in visualizing only the outer surface. [2] Therefore, they called the inner parts of the surface as inner remains as it was source of occlusion. However, the domain experts are often interested in the exploration of inner cavities because of their significant role in reactivity of the molecule. So it is advantageous for the user to have the option to visualize both molecular surface and inner cavities and interactively change this decision. It means that the cavities can be shown or hidden on demand.

For both SAS and SES, the hiding of cavities is straightforward [3] because of the fact that their surfaces are completely separated from the molecular surface. [3]. More specifically, one component represents the outer molecular surface and each single cavity corresponds to another component. For the contour representation of the surface, isolated components can be easily detected by applying connected component (CC) analysis to a structure which we call the *surface graph*. [4] We build our *surface graph* using triangles as vertices and toroidal patches (recall that a toroidal patch is delimited by two triangles) as edges connecting them (see Figure 6). Moreover, we also tried to do the CC analysis on a graph with spheres as vertices and toroidal patches as edges. But, the idea showed to be not useful because spheres may contain more than one patches that can form parts of different surfaces. More, the implementation of used graph algorithms and data structures on the GPU would be more complex. This is caused mainly by the fact that a sphere has arbitrary number of neighboring tori, while a triangle has exactly three tori as its neighbors. Hence, all vertices in our graph are of degree three.

Finally, the surface contains also tori that are not cut by any triangle. We call these tori as *isolated*, because they do not have any neighboring triangle (see Figure 7b). *Isolated* tori are excluded from the *surface graph* and they are handled later in the computation (see Section 3.4).

3.3.1 Extraction of surfaces

To extract the surfaces, i.e., the outer surface and the surfaces of the inner voids, we build a *surface graph* from the computed surface

[1] JP: This should be written in more friendly fashion and with clear defined terms. Terms to be explained: surface is continuitous (geometrical?, since C^1 continuous it is not), graphs (2D/3D, does it contain loops?), primitive, component

[2] AJ: Maybe, this could be part of related work

[3] AJ: It was quite hard job to do it:)

[4] JP: Introduce here "terminus techniques"; i.e., SES is composed of three types of patches etc.

and apply the CC analysis to it. The whole computation is done on the GPU, so that we avoid additional data transfers between main memory and graphics card. We split the analysis of the surface into three steps:

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

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

To build the adjacency matrix of G we utilize the hash table of triangles. During the writing of toroidal patches into a buffer for ray-casting, we also write a list of edges E 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. Here, we find the incident vertices of e by querying the hash table of triangles. In the hash table, each triangle is hashed using all three pairs of atoms – (i, j), (i,k) and (j,k), that form the three small circles that produced it. Then, the toroidal patches (and also edges) for a torus can be found by retrieving all triangles incident to a torus and sorting them relatively by their angular position around the torus. We use the bubble sort algorithm for this sorting. When sorted, the pairs of neighboring triangles define the toroidal patches and corresponding edges. We also 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. The test is done... Finally, the buffer of edges E is transformed into the adjacency matrix by a kernel. The matrix will have one row for each vertex (triangle) and three columns for its three neighbors (toroidal patches).

In the second step, all connected components of G are found and labeled using the BFS algorithm. We decided to choose a very simple yet inefficient [17] quadratic-work implementation because of the ability of BFS implementation as a one kernel. Our decision is supported by the performance measurements (see Section 6.2) 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 take ~x ms.

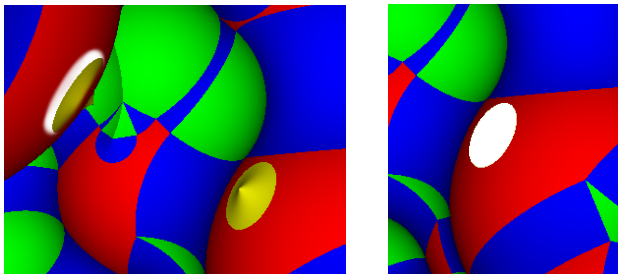
In the last step, the cycles representing the spherical patches are extracted. To do this, we assign the edges in E by their neighboring spheres into buckets. We then order the edges in buckets, so that they represent one or more cycles in G. A spherical patch is labeled by the label of any of its delimiting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

3.4 Special case – isolated torus

We handle *isolated* tori in two steps. First, a label for a torus must be found – recall that an *isolated* torus is not 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 (see Figure 7b).

An *isolated* torus can be encountered, while writing 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 with their surface labels. The assignment is done by inspecting 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 spherical patch test (see Section 5.1). Then, the torus lies in the same surface component as its neighboring patch.



(a) An *isolated* torus. (b) A hole in a spherical patch.

Figure 7: Isolated torus between two spheres. Isolated tori do not have neighboring triangles (a), therefore the surface component to which they belong can not be determined directly from the *surface graph*. An isolated torus cuts circular holes (b) into its neighboring spherical patches.

The clipping of fragments in affected spherical patches can be done using a clipping plane or a visibility sphere of a torus. We use the latter approach, since visibility spheres for tori were already computed in a previous step of our technique. Here, our kernel writes a texture, similarly to Krone et al. [12], which stores for an atom sphere all its intersecting visibility spheres. When ray-casting spherical patches, we use this texture to lookup all visibility spheres and discard all fragments being inside them.

4 Visualization

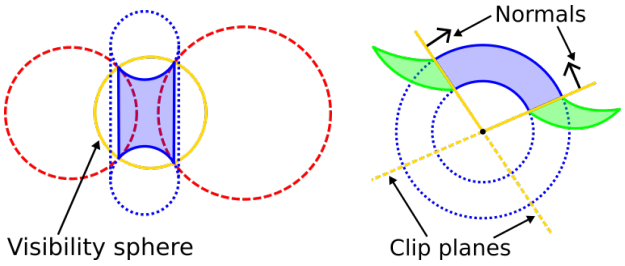
5 Real-time Visualization

(0.5 page) (AJ, JP)

To enable our focus and context transparent visual, 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 [24]. 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 provide a high performance, the ray-patch intersection is computed analytically. In 2013, Kauker et al. [9] proposed to ray-cast a toroidal patch using a torus and two clipping planes defined by its delimiting triangles (Fig. 8a).



(a) Clipping by *visibility sphere*. (b) Clipping by planes defined by spherical triangles.

Figure 8: Ray-tracing of a toroidal patch. The saddle part of the torus (a) is cut by so called *visibility sphere*. A patch (b) is cut from the whole toroidal ring by clipping planes.

We employ this approach and, moreover, we modify the data structure being used in the original algorithm to store and retrieve all spherical triangles incident to a torus (Fig. 9). In order to get

all neighboring triangles for a torus, 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]. This allows us, to ray-cast toroidal

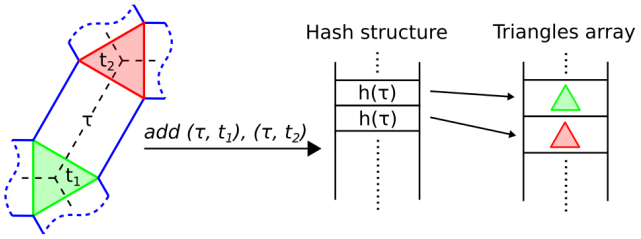


Figure 9: An illustration of the data structure for storing spherical triangles. Triangles t_1 and t_2 are stored linearly in an array and their incident torus τ is connected to them using a hash table.

patches directly instead of tori.

Ray-casting the sphere is a trivial task; nevertheless, in our case the surface sphere of molecules might be formed by an arbitrary number of spherical patches lying in different surface components. To avoid obvious rendering issues, i.e., distinct coloring, transparency or visibility, we perform ray-casting of each spherical patch separately. This allows us to visually distinguish between different surface components.

Since the boundaries of a spherical patch are formed by small circle arcs, we apply an odd-even rule for polygons [21] to test whether a point lies inside the patch. Firstly, we compute intersection points, P_{front} and P_{back}, of a given ray with the patch sphere. Here, we describe the point-patch test for point $P_{front} = P$ only, since the same test is applied to P_{back}. Before we apply the odd-even rule, we choose a point O lying outside the patch. Then, we test line OP, the shortest path on a sphere, for an intersection with each of the arcs delimiting the patch. The intersection of a line OP with the small circle arc, given by points AB, is computed in two steps (Fig. 10ba):

1. Intersections of the circles containing the line and the arc, which can intersect in up to 2 points.

2. Intersection points are tested whether they belong to both the segment OP and the arc AB.

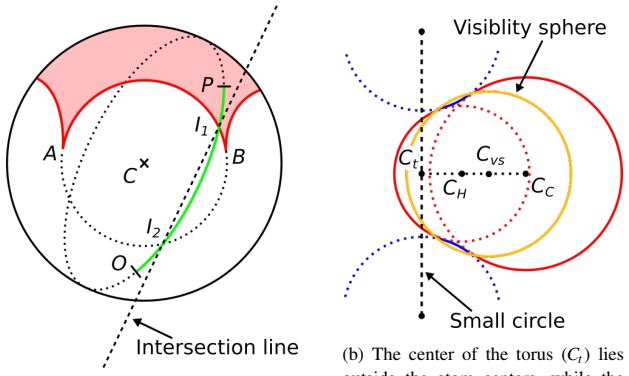
Finally, if the sum of all intersections, i.e., for all the sides, is even then the point lies inside the patch.

Since we deal with spherical patches and not the planar ones, the situation is bit more complex and we cannot choose point O arbitrarily. Instead, we compute O by intersecting a patch sphere by an axis of one of its delimiting tori. In this way, we get two intersection points from which we choose the one that lies in the direction of the torus visibility sphere (Fig. 10b).

5.2 Opacity modulation

After the intersection points were computed, we sort, in front-to-back manner, and store them in 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 represent 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:

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



(a) The intersection I_1 lies on an intersection line between two planes containing arcs OP and AB.

(b) The center of the torus (C_t) lies outside the atom centers, while the center of its visibility sphere C_{vs} lies between C_C and C_H.

Figure 10: Point in spherical patch test (a). A torus formed by carbon (C_C) and hydrogen (C_H) atoms (b).

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

$$\alpha_i = O^L, \tag{3}$$

where L is a user defined modulator to suppress or enhance the back-faces. Figure 11 showcases 4 different combinations of parameters K and O with the back-face modulator being constantly set to $L = 1$.

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 areas of all triangles that form the cavity surface. The area of a spherical triangle is calculated as follows:

$$S = r_{probe}^2\,[(A+B+C) - \pi], \tag{4}$$

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.

6 Discussion

The contribution of our proposed system will be demonstrated on a case study which corresponds to real problems the biochemists are currently facing. Then also the detailed performance analysis will be discussed.

6.1 Case Study – Real-time visual exploration of MD simulation

The case study deals with the situation when the biochemists want to visually explore the inner processes which are occurring inside the molecule. An example of such process can be the penetration of a small molecule (ligand) into the active site of the protein where the ligand reacts with the protein and the product of such reaction can form the basis of new chemical matters, e.g., new drugs. The

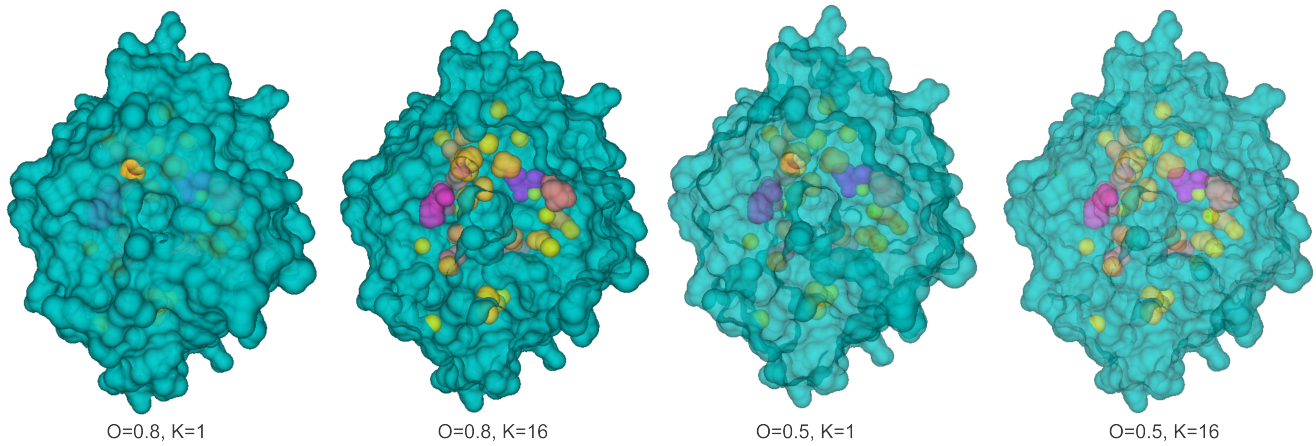


Figure 11: An example of application of parameters K and O on protein 1*cqw*. Note that higher values of the overall opacity O emphasize the front molecular surface, while higher values of maximum exponent K give more prominence to the internal surfaces and cavities.

current workflow generating the desired visual appearance of the animation showing such processes consists of many trial and error attempts which makes the whole workflow very lengthy. To describe it in more detail, the biochemists start in the first time step of the whole simulation and try to manually determine the best viewpoint. As the view inside the molecule, where the processes usually occur, is crucial, they are using clip planes to enable this. The manual setting of the clip planes introduces other possible error to the final animation. The dynamic movements of the molecule can cause its rotation and the clip plane set in the first time step can have in the following steps completely wrong position. When all features are set for the first frame (see Figure 12), the biochemists use an offline rendering tool for generating the whole animation. In this phase they do not have any control of this process so it is impossible to detect an error inside the animation and stop the generation. These errors, such as occlusions or wrong clip plane positions, are detected when playing the final animation. The only solution is to adjust the input settings and launch the whole process once again. For better imagination, in case of the example video (from which Figure 12 is captured), it took two days to produce the solution which comprehensibly shows the whole process of ligand penetration to the protein active site.

Our new system is able to overcome the following limitations of the above-described workflow:

- The MD simulation can be observed in real time which enables the user to interactively adjust the appearance and viewpoint.
- The highly transparent molecular surface removes the necessity of using clip planes.
- The user had full control of the animation process so we completely remove the trial and error phase of the workflow.

In consequence, our system enables to reach similar results in real-time. In one aspect it even overcomes the existing solution as the users can interactively manipulate with the scene on the fly – perform scene transformations, change the appearance of the protein and ligand, or change the probe size used for the generation of protein surface.

Figure 13 shows one time step of the animation generated using the same dataset as for Figure 12.

6.2 Performance Analysis

- Performance analysis

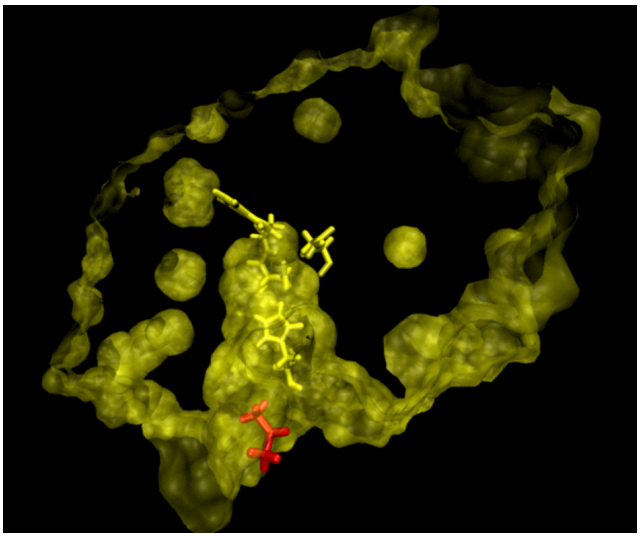


Figure 12: One time step from the animation aiming to show the penetration of the ligand to the protein active site. For better insight, different representations of the molecule and the ligand, surface transparency, clip plane, and different coloring was used.

- Pros & cons
- Limits

We tested our technique on commodity hardware to show that it enables users to solve their tasks in real-time without high demands on their hardware. Test also on high-end hardware — GTX 980? The tests were run on 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 resolution of 1024×768 and we tried to fit the molecule such that it covered most of the image. Regarding transparency, we limited the maximum depth complexity to 24 fragments per pixel. The results of our measurements for static molecules are given in Table 1.

Our technique is implemented mainly using OpenGL employing GLSL compute shaders as GPU kernels. We also utilize OpenCL to accelerate a kernel which computes positions of spherical triangles in the original algorithm [12]. The GLSL implementation of this kernel performed for a molecule with ~10000 atoms about 5x

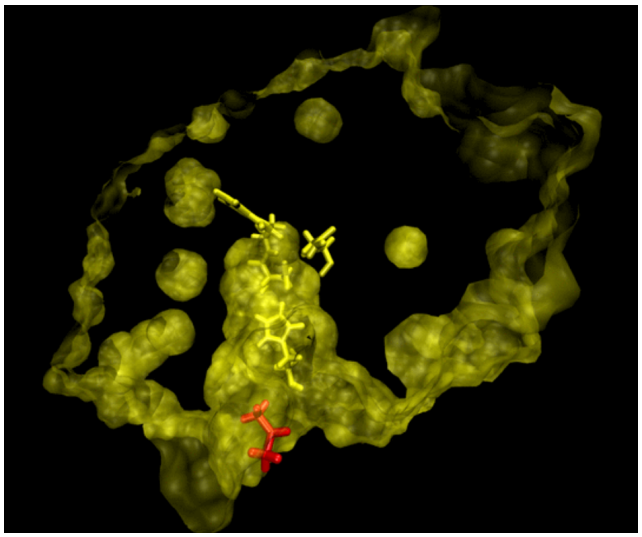


Figure 13: TODOOne time step of the animation generated using our system.

Table 1: Performance of our technique for static structures.

Molecule	# Atoms	CB (ms)	SG (ms)	AO+Area (ms)	Ray-casting (ms)	Total (FPS)
1OGZ	~1000	4.1	0.2	0.2	15.7 (36.7%)	31.0
1VIS	~2500	6.1	0.8	0.2	39.3 (52.7%)	16.4
4ADJ	~10000	21.1	4.3	0.4	97.7 (41.5%)	6.9

slower than the OpenCL/CUDA one. From the tests we did, we assume that the key of such performance loss is the extensive use of the global memory in the kernel which is handled differently in OpenGL and OpenCL/CUDA.

7 Conclusion

Future work

- More tight bounds for ray-casting may further improve the performance. This holds especially for tori because each torus is ray-casted xxx (about 2) times in average. (AJ)
- The ray-casting could be done using OpenCL which could lower bandwidth because the data common to many fragments could be fetched only once and shared using local (shared) memory. (AJ)
- Transfer function.

References

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

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

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

[4] 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, New York, NY, USA, 1984. ACM.

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

[6] E. Enderton, E. Sintorn, P. Shirley, and D. Luebke. Stochastic transparency. *Visualization and Computer Graphics, IEEE Transactions on*, 17(8):1036–1047, 2011.

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

[8] 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.

[9] 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.

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

[11] M. Krone, K. Bidmon, and T. Ertl. Interactive visualization of molecular surface dynamics. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1391–1398, 2009.

[12] 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.

[13] 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*, EuroVis'10, pages 943–952, Chichester, UK, 2010. The Eurographs Association & John Wiley & Sons, Ltd.

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

[15] 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.

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

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

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

[19] 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.

[20] 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.

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

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

[23] A.-A. Vasilakis, G. Papaioannou, and I. Fudos. k^+-buffer: An efficient, memory-friendly and dynamic k -buffer framework. *Visualization and Computer Graphics, IEEE Transactions on*, 21(6):688–700, June 2015.

[24] J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz. Real-time concurrent linked list construction on the gpu. In *Computer Graphics Forum*, volume 29, pages 1297–1304. Wiley Online Library, 2010.

A Original arcs structure vs. our hash structure size

Orignal structure complexity is $|A| \cdot maxNeighbors^2$ where $maxNeighbors = 64$.

Our structure complexity is $|T| + hashFreeRatio \cdot 3|T|$ where $hashFreeRatio = 2$.

The ratio of atoms to triangles is approximately (value from experiments) $\frac{|A|}{|T|} > \frac{1}{4}$ so that $\frac{|A| \cdot maxNeighbors^2}{|T|(1+3 \cdot hashFreeRatio)} > \frac{64^2}{28} > 146$ which means 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 surface of molecules that contain hydrogens.

http://idav.ucdavis.edu/~dfalcant/downloads/dissertation.pdf