Interactive Visualization of XXX (Voids) in Molecular Dynamics
Real-time Visualization of Protein Surface Features in Molecular Dynamics

Category: Research

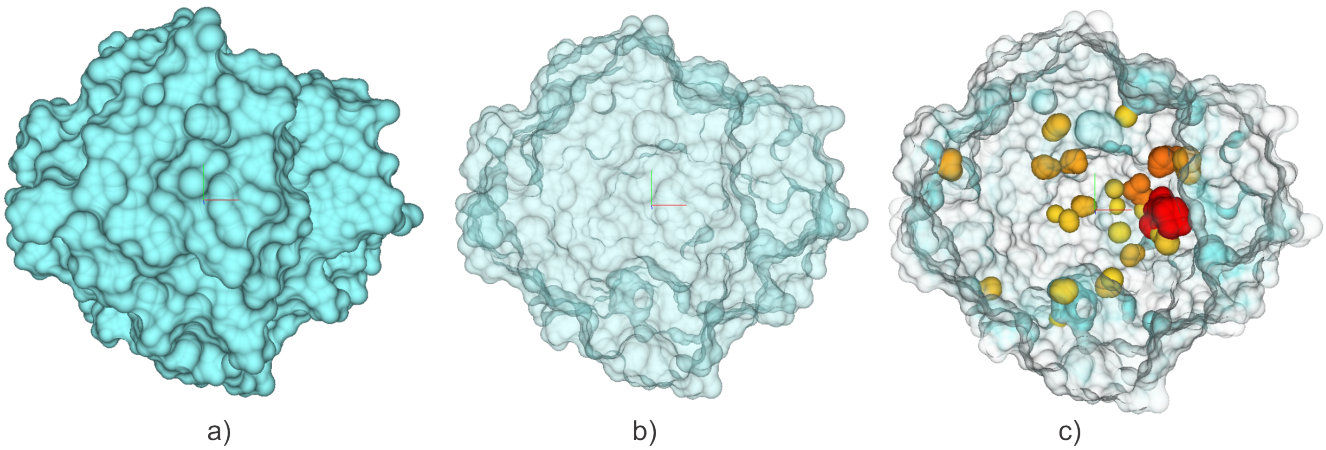


Figure 1: An example of protein XXX demonstrating our method. a) SES b) transparency c) transparency and cavities. TODO

Abstract

The reactivity of the biomolecular structures is highly influenced by the presence of an inside void space. Thus, studying the inner void space along with the exploration of their dynamic behavior helps to understand the processes ongoing in all living cells. This can be reached by the visual representation of this process as visualization is one of the most natural ways to convey such information. However, none of the currently available systems provides the biochemists with an intuitive real-time representation of the dynamic movements of molecules and extraction of their inner void space. In this paper we introduce such system which enables the user to compute and visualize the molecular surface and inner cavities instantly. Moreover, to obtain a better insight into the molecule, we propose a technique that allows us to visualize the molecular surface transparently. The whole visualization system is created in a focus and context visualization manner, where a 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 system is even amplified with respect to the fact that currently the size of molecular dynamics simulations is increasing dramatically and offline rendering thus becomes very impracticable. We present the usability of our system by a case study which reflects the real problem of the domain experts.

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. 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 a 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. We present the usability of our technique 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 (JP, BK)

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 [3]. This even stresses the importance of studying and exploring the trajectories of molecular dynamics (MD) in detail. As the captured or simulated trajectories are dramatically increasing their size, 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 setbacks. Moreover, the exploratory process of MD trajectories is often concerned with the visual identification

of binding sites of ligands to a host protein. These sites represent a molecular surface features known as cavities, pockets, or tunnels. There is a legacy of tools and approaches that allow us to extract these features. Two major challenges in regards to the surface feature analysis, i.e., for instance cavities, are their fast extraction and their visualization in the most informative manner.

We are facing all these challenges by introducing a novel system for 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:

- Enhanced computation of solvent-excluded surface (SES) of the molecule. We propose three new kernels that account for speedup of the existing state-of-the-art approach [**?**].
- A novel real-time algorithm for the detection of cavities (check Totrov — AJ). We present new fast cavity detection method that is based on solvent-excluded surface (SES).
- Improved performance of visualization of transparent molecular surfaces [**?**]. We describe our interactive focus and context visualization of cavities on the molecular surface.

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 (ALL)

PUXELS [4]

- Performance drop of SES rendering on newer hardware (GF 680 GTX) — we can perform better. We contacted authors, they do not know exactly why, they think it is change in the internal architecture between Fermi and Kepler (AJ)
- Slow rendering of SAS. Too many layers in pixels — we can do better by surface layers detection (AJ)

There are more solutions taken from computer graphics; e.g., OIT (JP)

AOOM

- We employ transparency modulation techniques presented in their paper [2]

2.1 Molecular Surface Representation

Shouldn't be this section part of motivation of our algorithm for SES? Also, it seems that it mixes two types of voids - cavities and pockets. It needs to be discussed.

There are several types of molecular surface representation proposed in the literature [5]. However, for 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. Such binding site can be located inside a cavity or a tunnel, while the molecular surface might contain tens of cavities per single simulation time step. Additionally, 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.

2.2 Extraction and Visualization of Cavities and Tunnels

3 Order-independent Transparency for Contour-Buildup (AJ, JP)

Overview (0.5 page) (AJ, JP)

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

3.1 Overview

The data comes in a form of trajectories describing motion of individual atoms. Each trajectory snap shot includes a set of atoms, described by their positions and their radii. Our rendering pipeline consists of several steps that are performed on per-frame basis. For better explaination, we split computations into two groups. The first group deals with data processing that involves 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, estimate of ambient occlusion and opacity calculation; before the final stage represented by the image formation. In more detail, we perform the following steps (TODO need more detail from Adam):

1. We employ 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 (Sec. 3.2).
2. To extract cavities we introduce the so-called *surface graph* (Sec. 3.3), which allows us to compute the internal voids / cavities.
3. Each one of the three primitives is visualization 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 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 [6]. 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.

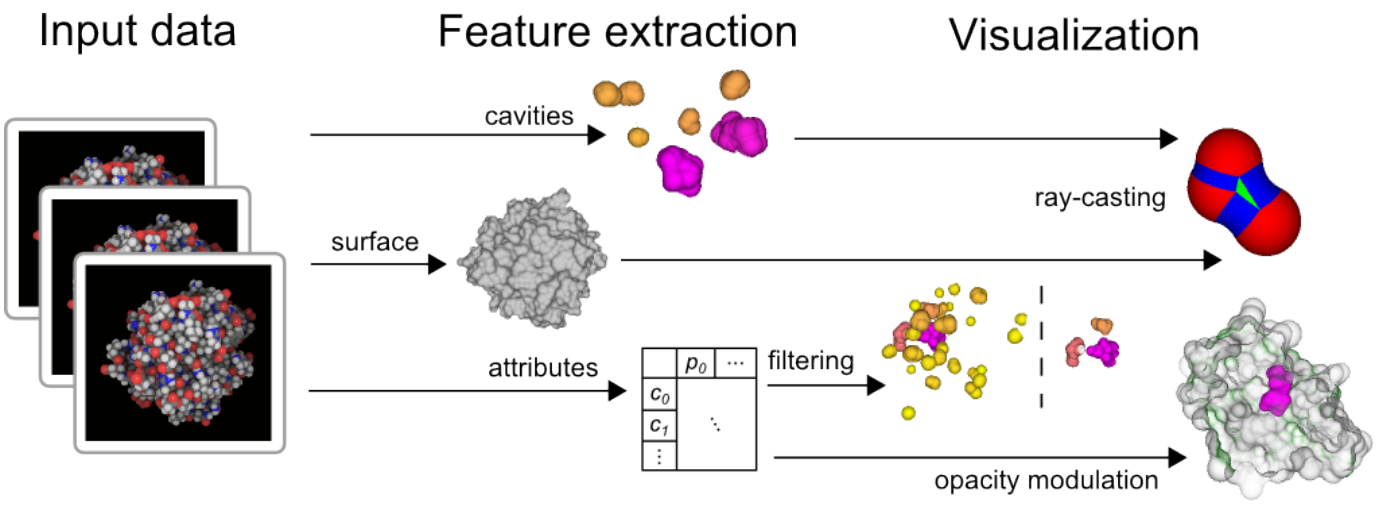
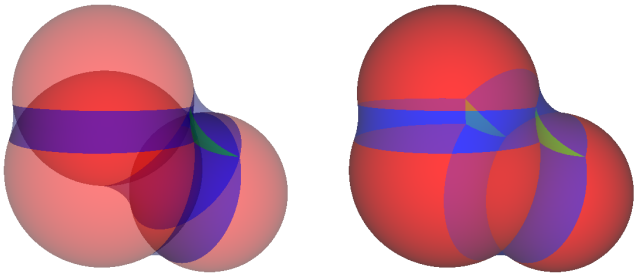


Figure 2: TODO.



(a) The original method [6] 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.

- 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. [4] 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 4) which is based on linear addressing scheme [1].

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).

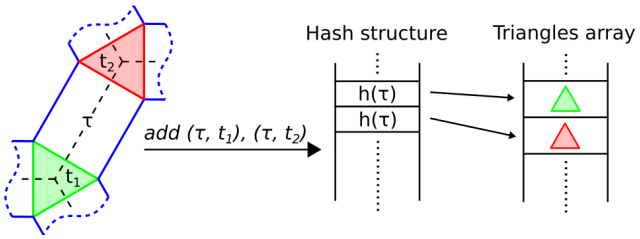


Figure 4: Our novel data structure for storing spherical triangles. Triangles t_1 and t_2 are stored linearly in an array and their incident torus τ is connected to them using a hash table.

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. [4] 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. [2]. 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 ver-

[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.

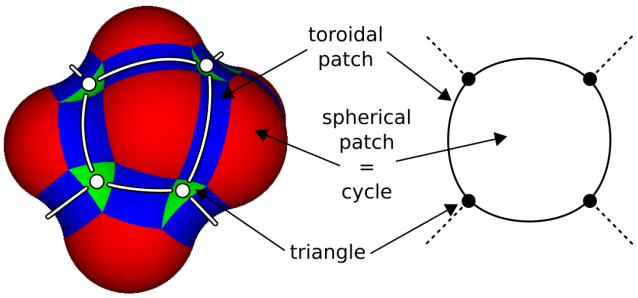


Figure 5: The surface graph. Triangles form vertices and toroidal patches form edges between them. Spherical patches are represented as cycles in the graph.

tices and toroidal patches (recall that a toroidal patch is delimited by two triangles) as edges connecting them (see Figure 5). 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 such tori as *isolated*, because they do not have any neighboring triangle. *Isolated* tori are excluded from the *surface graph* and they are handled later in the computation (see Section ?).

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 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 [7] quadratic-work implementation because of the ability of BFS implementation as a one kernel. Our decision is supported by the performance measurements (see Section 6.3) 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

The isolated torus must be handled in two ways. First, the label of the surface that an isolated torus forms must be found – recall that the torus is not part of the *surface graph*. Second, each isolated torus should clip the overlaid fragments of its spherical patch (see Figure ?).

4 Visualization (AJ, JP)

5 Real-time Visualization

(0.5 page) (AJ, JP) To enable our focus and context transparent rendering, we employ an A-buffer to acquire a list of participating fragments. These fragments represent three types of surface patches, computed in previous steps, analytically computed for a given ray. This is obtained via an analytical approach.

5.1 Rendering of spherical patches — polygons?

A sphere can produce one or more spherical patches which may form different surfaces. To be able to visualize isolated surfaces individually, we render (ray-cast) each spherical patch as a separate surface primitive. This way, we are able to visually distinguish between molecular and cavity surface and also among the detected cavities themselves. In fact, the sides of a spherical patch are formed by small circle arcs. When ray-casting a spherical patch, we firstly compute intersection points I_1 and I_2 of a ray and patch's sphere. Then, we employ the odd-even rule to test whether the computed intersections lie within the patch. We choose a point O outside the patch and test lines (lying on a great circle) OI_1 and OI_2 for intersection with each side of the patch. The outside (or inside) point must be specified because both the interior and exterior surfaces of the sphere are finite.

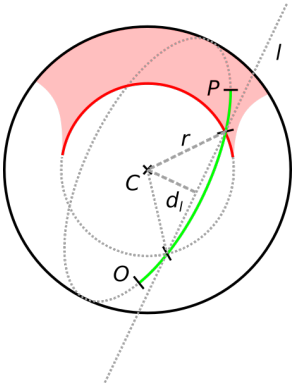


Figure 6: Point spherical patch test.

The intersection of a spherical segment with a small circle arc is computed in three steps:

1. The intersection of the circles containing the segment and the arc is computed — they can intersect in 0 to 2 points.
2. The intersection points are tested whether they lie on both the segment and the arc.
3. Cases with two intersections are marked as non-intersecting because the tested point lies outside the patch.

TODO: HTTP://HORIZON.DOCUMENTATION.IRD.FR/EXL-DOC/PLEINS_TEXTES/PLEINS_TEXTES_6/B_FDI_39-40/43404.PDF

5.2 Opacity Mapping

Borland [2] proposed to utilize ambient occlusion (AO) values to alter the opacity. Motivated by his approach, we exploit the ambient occlusion values as well. Since, we would like to maintain fast rendering performance, we need to remedy the issue of having an object space technique to evaluate the AO values. In the former work of Borland, the performance was not an important factor, which allows him to exploit the full object space AO evaluation. Here, we opted for the most recent approach, proposed by Grottel et al. [**?**], which renders ambient occlusion values to a $3D$ grid containing an estimate of the volume area of atoms inside a voxel. On the other side, this approach only reflects the volume of atoms and not the molecular surface, we find it as a good trade off between the visual precision and the performance measure taking into consideration that these values are not employed directly, but as opacity modulators.

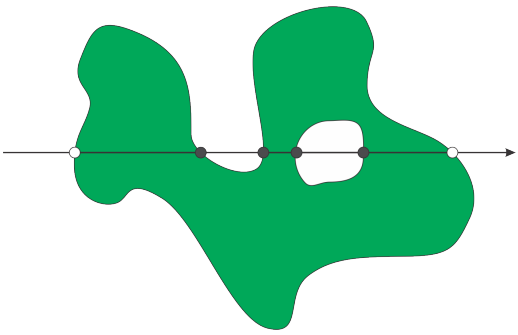


Figure 7: (TODO make more nice with overlay AO grid). An example of the list of fragments per a given ray. The color of the circles represent the obtain ambient occlusion value.

Interactive Analysis (0.5 page) (ALL)

- Possibilities (scenario, dynamics, pipeline)
- Feedback

5.3 Cavity area estimation

Observation: triangles take most area of a cavity.

We enhance the visualization of cavities by coloring their surface by their approximate area. To estimate the area, we sum areas of all triangles that form the cavity surface. Therefore, the cavity area we compute is underestimated – maybe an equation?. We decided to neglect areas of spherical and toroidal patches since from our observations their influence on the exact cavity area is much smaller compared to triangles. Of course, this observation does not hold for the molecular surface. What about coloring?

6 Discussion (BK)

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

6.1 Case Study 1 – Real-time visual exploration of MD simulation

The first 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 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 8), 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 8 is captured), it took two days to produce the solution which comprehensibly shows the whole process of ligand penetration to the protein active site.

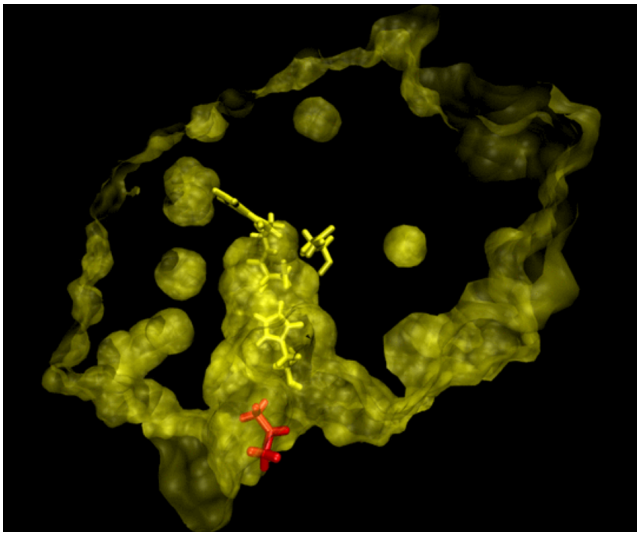


Figure 8: One time step from the animation aiming to show the penetration of the ligand to the protein active site. For better insight, different representations of the molecule and the ligand, surface transparency, clip plane, and different coloring was used.

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 9 shows one time step of the animation generated using the same dataset as for Figure 8.

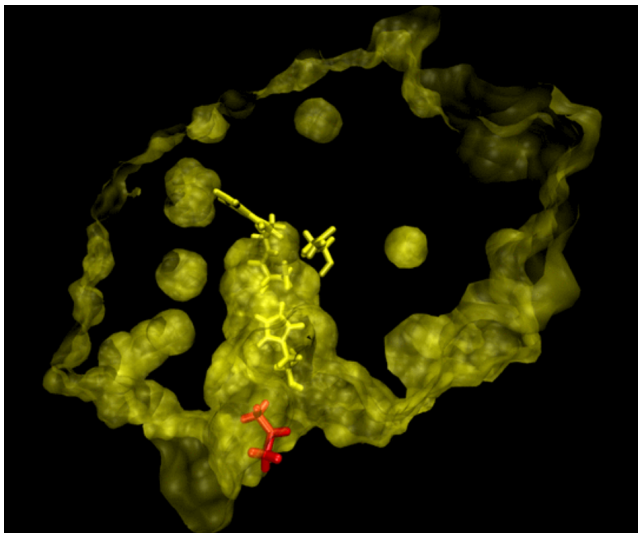


Figure 9: TODOOne time step of the animation generated using our system.

6.2 Case Study 2 – Studying the shape of protein tunnels

Tunnel in protein put into different organic cosolvents.

TODO

6.3 Performance Analysis

- Performance analysis
- Pros & cons
- Limits

Table 1: Performance in ms.

MolID	Atoms	Surf	Cav	Ray	Total
1OGZ	~1000	0.0	0.0	0.0	0.0
1VIS	~2500	0.0	0.0	0.0	0.0
4ADJ	~10000	0.0	0.0	0.0	0.0

7 Conclusion (ALL)

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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