Interactive Visualization of Cavities in Molecular Dynamics

Category: Research

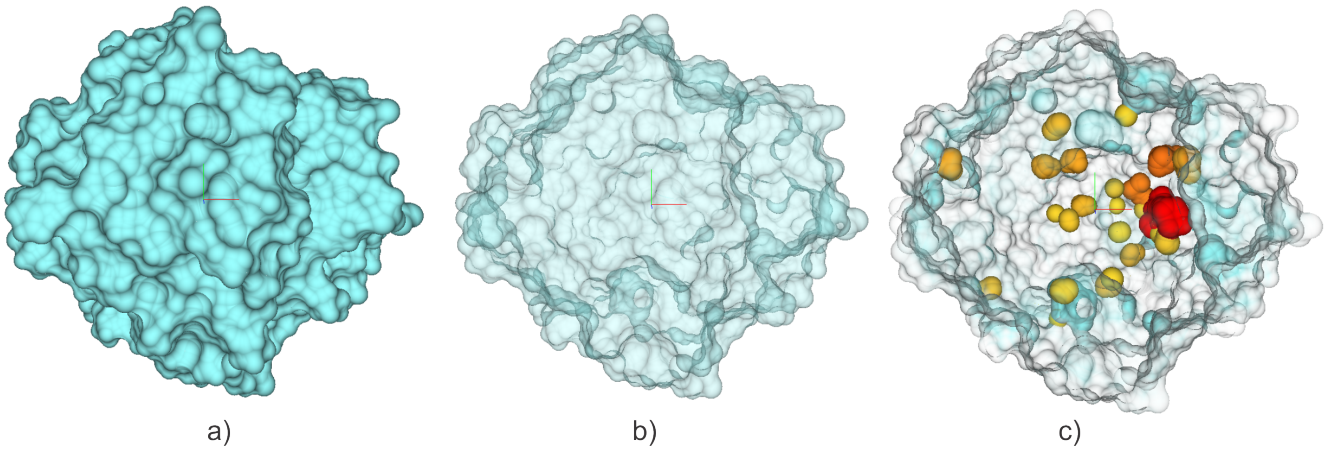

Figure 1: An example of protein XXX demonstrating our method. a) SES b) transparency c) transparency and cavities. TODO

ABSTRACT

TODO: Abstract.

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

1 INTRODUCTION (JP, BK)

Contribution

- Transparent surface rendering speedup (AJ)
- Novel real-time cavity detection method (AJ)
- More precise (analytic computation + ray-casting) interactive visualization of cavities in molecular simulations (JP)
  - Focus and context visualization of cavities within transparent molecular surfaces
  - Opacity modulation by cavity features (surface area, AO, etc.)
- ? Memory efficient CB (AJ)
- ? Vendor independent implementation (OpenGL + OpenCL) (AJ)

Problem

- Artifacts & occlusion, e.g. for secondary structures (AJ)

Nafuknut na catch up with big data analysis from MD simulation... (JP) The exploratory process of MD simulations is often concerned with the visual identification of binding sites of ligands to a host macromolecule. These sites represent a molecular surface feature known as cavities, pockets or as 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 analyzes, i.e., for instance cavities, are their fast extraction and their visualization in the most informative manner. This is especially crucial when analyzing MD simulations containing thousands of frames, where we cannot spent much time analyzing just a single frame either due to computational or visualization setbacks.

There are several types of molecular surface representation proposed in the literature [**?**]. Nevertheless, for cavity analysis, the solvent-excluded surface (SES) belongs to the most used representation [**?**] amongst biologists. This representation allows us to directly asses whether a solvent, approximated a sphere of a certain radius, is able to reach a binding site of interest on the molecular surface. Such a binding site is located inside a cavity or a tunnel, while the molecular surface might contain tens of cavities per a single simulation snap-shot. 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 an instant computation and an interactive and meaningful visualization of the cavities in the context of molecular surface.

In this paper, we introduce visualization technique (Fig. 1) that enables to ...

The contributions are as follows:

- An enhanced computation of SES
- A novel real time algorithm for detection of cavities (check Totrov — AJ)
- Focus and context visualization of cavities in the context of molecular surface
- Improved performance of visualization of transparent molecular surfaces

2 RELATED WORK (ALL)

PUXELS [2]

- 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 [1]

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

Overview of the algorithm: (JP)

- Per frame we perform the following steps: ...
- The data comes in form of trajectories describing motion of individual atoms, we do not assume anything about the data.
- We split computations to two groups, which are performed on per frame basis.
  1. Data preprocessing (surface, cavity and attributes computations)
  2. Visualization (ray-casting, opacity calculation, etc.)

3.2 Our Modifications

- Storage of arcs — hashing to enable clipping
- We build a surface graph in the write kernel
- Spherical patches are written based on surface analysis
- TODO: image

3.3 Surface graph

- Idea: computed surfaces are continuous (closed) and graphs of their primitives form isolated components in the whole graph of all surface primitives
- Modification of parallel CB of Krone et. al — aaaa
- Extension of parallel CB of Krone et. al — 3 new kernels:
  1. Adjacency matrix is built (only 3 edges at each vertex)
  2. Labeling of connected components (parallel BFS — suboptimal)
  3. Circles of edges for each spherical patch are computed
- Assign spheres with edges
- Detect circles in edges — bubble sort $O(n^2)$
- Step 3 — one sphere can form two or more surfaces
- Rendering of spherical patches — spherical polygons
- Odd-even rule + point outside polygon
- Special case: isolated tori

Seeds — The computed surface contains also surface of cavities that was inaccurately called by Kauker et. al. as inner remains [2]. For SES, the user might want to visualize cavities within the molecular surface. Contrary, for SAS, the inner surface forms only seeds of the cavities, and the seeds are not useful for assessing the real shape of a cavity, so that clipping these seeds enhances the visualization. The user might also want to hide cavities inside a SES to lower possible occlusion of other structures such as tunnels or cartoons. The clipping of cavities is enabled by observing that both SAS and SES are continuous and therefore there has to be more than one continuous surface component when the molecule contains a cavity [1]. Too many ideas in one sentence. Split into two or more.

Introduce here "terminus techniques"; i.e., SES is composed of three types of patches etc. More precisely, there is one component for the outer surface of the molecule and one component for each cavity, and the components can be easily detected by applying connected component (CC) analysis to the graph formed by surface primitives (spherical patches — polygons?, toroidal patches and triangles). The spherical patches can not be used because they are not known for now. We do not know whether a sphere forms one or more patches and who are their neighboring tori, i.e., edges. Therefore the surface graph is built by triangles (vertices) and toroidal patches (edges). The surface contains also tori that are not cut by any triangle, i.e., they do not have any neighboring triangle. Such toroidal patches are excluded from the surface graph and has to be handled differently (see Section ?).

We do the CC analysis on the GPU to avoid synchronization and data transfer costs. First, we modified the output of the original GPU parallel CB to obtain the input which is needed for the analysis and rendering of transparent toroidal patches. In the original algorithm [3], an arc intersection was stored only for atoms whose indices fulfilled $i < j < k$. This is insufficient for rendering the toroidal patches transparent as they can't be rendered as a one whole patch because the parts that would be hidden by the opaque surface would be visible. Instead, we split each torus into its visible patches based on their neighboring triangles that delimit them. Since each torus is defined by a small circle between atoms i and j, we are interested in all triangles that were produced by atoms i, j and some other atom k. For this purpose, we store the computed arc intersections in a linear buffer (employing atomics) and together produce a hash structure which enables us to find the triangles by their two of the three atom indices. As a benefit to this hash structure, we save GPU memory because the original arcs structure was very sparse. Now, we store n arcs together with $3n$ keys in a hash table which data/free ratio is 2. TODO: More precision.

The analysis part is split into three steps and for each we implemented one GLSL compute shader. First, the adjacency matrix of the surface graph is computed. The matrix will have one row for each vertex and three columns, because each triangle has exactly (at most?) three neighbouring toroidal patches in the surface. In the next step, all connected components are detected and labeled using BFS. Our implementation of the BFS algorithm is suboptimal, because its time complexity is $O(d \cdot n)$ where d is the length of the longest shortest path among all vertices in a component. In the worst case, d can be n. The reason we choose such ineffecient implementation was the ability of BFS implementation in one compute shader [4]. Our decision is also supported by performance measures (see Section ?) where the computation of labels takes only 5 ms for a molecule with 10000 atoms while the computation of SES takes 5x more. Finally, we sort all edges neighboring with a sphere to get one or more circles where each circle delimits a spherical path. The lables for the spherical pathes are obtained from the labels of some of their delimiting edges.

3.4 Special case — isolated tori

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

4 Visualization (AJ, JP)

Visualization (0.5 page) (AJ, JP)

- Transparency
- Opacity mapping (surface, AO, ...)

Interactive Analysis (0.5 page) (ALL)

- Possibilities (scenario, dynamics, pipeline)
- Feedback

4.1 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.

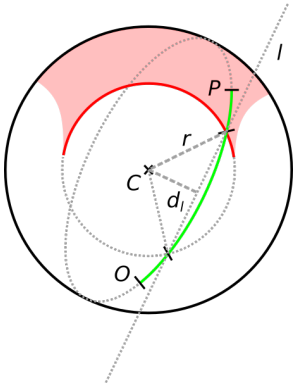

Figure 2: Point spherical patch test.

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

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

4.2 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?

5 Discussion (BK)

- Performance analysis
- Pros & cons
- Limits

6 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)

References

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

[2] 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.

[3] 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.

[4] 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