EG UK Theory and Practice of Computer Graphics (2013)
Silvester Czanner and Wen Tang (Editors)

Geometry-based Algorithm for Detection of Asymmetric Tunnels in Protein Molecules

Jan Byška†, Adam Jurčík‡, Jiří Sochor

Faculty of Informatics, Masaryk University, Brno, The Czech Republic

Abstract

We present a novel geometry-based method for computing asymmetric tunnels and voids in proteins, approximating their real shape with selected precision. Our method combines ideas from Voronoi and grid based approaches for protein analysis. We represent tunnels in protein using voxel data grid which allows us to store their shape more accurately. Our algorithm employs a tunnel skeleton computed using Voronoi diagram. The skeleton allows us to perform grid computation in a bounded space, with lower time and memory demands, and easily identify and measure individual tunnels.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Boundary representations; I.3.8 [Computer Graphics]: Applications—Geometry-based Analysis of Proteins

1. Introduction

Proteins play an essential role in many biological processes. These macromolecules are large biological structures consisting of one or more chains of amino acids. Protein chains are folded forming highly complex structures of atoms and empty space. Along with many unimportant tiny hollows proteins contain voids that may form cavities of specific functions, tunnels connecting these cavities with a molecule's surface, or channels and pores penetrating throughout the whole molecule. Tunnels, channels or pores play an essential role in the functioning of large number of proteins since they represent potential transport pathways for small molecules, ions and water molecules. These small molecules can than participate on biochemical reactions within the protein. [ZM10] [ABN95] [GM05] [GDC07].

Advanced in-silico analysis and visualization of proteins is a necessity due to their complex structure, behaviour and high costs of laboratory experiments. Recently, many geometric-based tools for rapid analysis of protein tunnels, channels and voids were presented. They can be classified according to the method they compute and represent interior space of a protein molecule.

The first category of the tools is represented by HOLLOW 1.2 [BF08] and 3V 1.0 [VG10]. Both HOLLOW and 3V can sufficiently describe a geometry of nearly any type of void using a grid-based approach. However, they provide only limited information on tunnels and channels characteristics. Another disadvantage of these tools is their large demands on time and memory for detailed visualization [BCG*13].

The drawbacks of the first group have been partially overcome by the second category of algorithms including MOLE 1.2 [PKKO07], MolAxis 1.4 [YFW*08] and CAVER 3.0 [CPB*12]. These tools employ Voronoi diagram to compute tunnels (in case of MolAxis tunnels and channels) within the protein structure. The MOLE algorithm constructs the Voronoi diagram without considering variability in radii of individual protein atoms. MolAxis and CAVER lower the computation error by approximating each atom in the protein by a set of smaller spheres with the same radius. Nevertheless, the main shortcoming of these tools is that none of them is able to capture and describe asymmetric tunnels. They can only detect tunnels with circular cross-sections. However, in reality these tunnels have usually more complex cross-sections of an arbitrary shape.

† xbyska@fi.muni.cz
‡ xjurc@fi.muni.cz

DOI: 10.2312/LocalChapterEvents.TPCG.TPCG13.017-024

The third category of algorithms is designed mainly for the channel and pore analysis. It includes HOLE 2.2 [SNW*96], CHUNNEL 1.0 [CS09] and PORE-WALKER 1.0 [PCMT09]. These tools employ different algorithms and use different assumptions and starting condition for computation. Only the HOLE tool is able to handle asymmetry in channel dimensions. It represents the resultant channel as a set of maximal capsules while the other tools in this category use only spherical probes. However, the HOLE algorithm can produce a discontinuous pathway deviating from the ideal channel axis [BCG*13].

In this paper we present a novel method for computing the protein tunnels that preserves their real shape as much as possible. Our method combines ideas from the first two mentioned groups of tools. We represent a tunnel in protein using voxel data grid which allows us to store its shape in a simple and enumerable form. Our algorithm employs a tunnel skeleton computed using Voronoi diagram. The tunnel skeleton allows us to reduce the computation only to a limited area near the tunnel, with lower time and memory demands, and easily identify and measure volumes of individual tunnels.

2. Asymmetric tunnels

2.1. Problem definition

Current tools for protein analysis such as CAVER, MOLE and MolAxis, utilize Voronoi diagram to compute all possible pathways connecting a given inner cavity with the protein surface. Dijkstra´s algorithm together with a specific cost function for Voronoi edges is used afterwards to find a set of optimal pathways.

Let us suppose that we have computed an appropriate Voronoi diagram tessellation of the protein molecule. The adequate space subdivision can be computed either as a weighted Voronoi diagram for atoms with different radii, or as a normal Voronoi diagram by approximating each atom in the analysed protein by a set of smaller spheres with the same radius. The pathway describing tunnel is then actually a list of Voronoi edges equidistant to three nearest atoms. An empty space inside the protein molecule therefore can be described by spheres placed on these Voronoi edges. The radius of each sphere is given by the distance between the point on a Voronoi edge and the neighbouring atoms. Summing up, a computed tunnel is a void defined by its *skeleton* – list of annotated Voronoi edges, and is often visualized as a set of spheres. The 2D simplified sketch of this method is depicted in Figure 1.

The representation of voids by a set of spheres has both advantages and disadvantages. The advantage is that the smallest sphere of a tunnel can be considered a bottleneck with diameter d, where some ligand smaller then d can pass through it, for details see [MBS07]. However, it is clear that when describing a void in a molecule by a simple set of

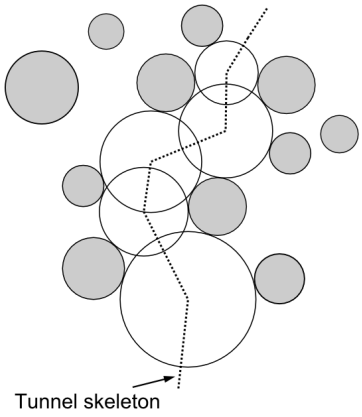


Figure 1: Simplified 2D representation of a computed tunnel. The tunnel is visualised as a set of empty spheres (white) touching the atoms (grey).

spheres placed on Voronoi edges, some information is missing. The cross-section of a tunnel does not have to be circular, and Voronoi based approach may detect two spherical tunnels instead of one asymmetric tunnel (see Figure 2). In the worst case, a part of the void surrounding the tunnel may remain undetected [BCG*13].

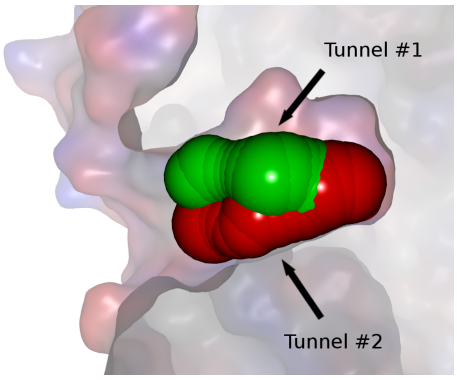


Figure 2: Two tunnels (red and green) computed by CAVER. It can be observed that these tunnels do not cover the entire void.

The need of more accurate representation of molecular voids was first addressed by tools such as HOLLOW or 3V. These tools employ a grid-based approach with a *rolling probe method*. Two or more probe spheres with different radii are placed on each grid point and checked whether they do not intersect with any protein atom. This way, it is possible to find empty volumes that can be combined together to define external and internal empty space (see Figure 3). The resultant voids are, however, computed over the whole protein and it is hard to automatically identify individual tunnels [BCG*13].

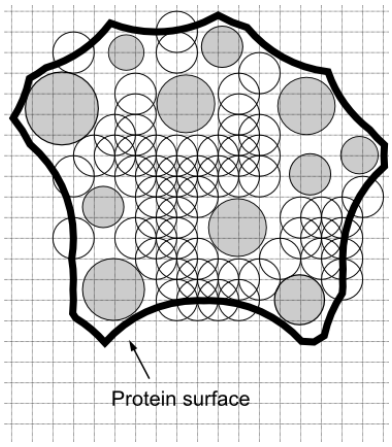


Figure 3: Example of void computation used by HOLLOW or 3V tools. The outer surface of the protein is defined by a probe with greater radius, a smaller probe is used to detect voids.

2.2. Proposed solution

We propose an algorithm which combines both Voronoi diagram and grid-based method. Our algorithm exploits information acquired from tunnels computed by the CAVER algorithm, however, in general it is possible to use any Voronoi diagram based tool and its output. The information about tunnel skeleton is used to define a *region of interest* (ROI) where the asymmetric tunnel is supposed to pass trough. Inside this area the modified rolling probe method is used to determine all voids (see Figure 4). Afterwards, all isolated cavities that do not belong to the main tunnel body, are removed. The resultant shape of the tunnel is stored using a voxel grid.

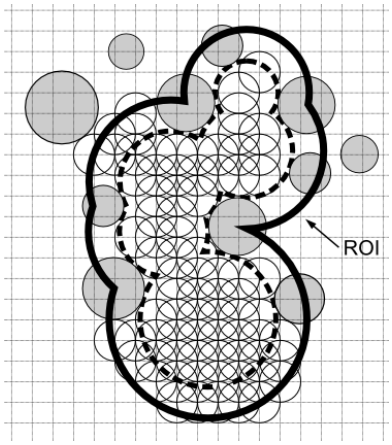


Figure 4: 2D representation of our approach. The ROI is determined as an extension of the pre-computed tunnel (dashed line).

This way, we are able to compute a more realistic shape of the tunnel. Moreover, since we can easily determine which voxel interior belongs to which tunnel, we can easily measure volume-related properties of asymmetric tunnels or merge different overlapping tunnels.

3. Algorithm

In this section, we introduce our algorithm in general and we describe selected parts in more detail. The pseudocode in Algorithm 1 gives an overview of the entire algorithm.

Algorithm 1 Algorithm for voids detection

Input: A – atoms, T – tunnel, ε – extension, d – grid density, r – probe radius
Output: g – grid with voids

```
1:  procedure DetectVoids(A, T, ε, d, r)
2:      b ← BoudingBox(T, ε, r)                    ▷ b is AABB
3:      A_in = {a ∈ A | Intersects(a, b)}
4:
5:      g ← MacroGrid(b)                           ▷ g has size b
6:      RegionOfInterest(g, T, ε)
7:      PlaceAtoms(g, A_in, r)
8:
9:      P_surf ← FindSurfacePoints(g)
10:     PlaceProbes(g, P_surf, r)
11:     return g                                   ▷ g contains voids
12: end procedure
```

The algorithm requires the set of protein atoms A, the tunnel T computed by Voronoi based tool – represented as a set of spheres, ε influencing the region where the algorithm will look for a solution, density d of the resultant grid and a probe radius r. The result of our algorithm is the uniform grid g of voxels of size d. The resultant data are actually stored in vertices of the grid.

The algorithm exploits the information obtained from a tunnel computed by CAVER method. The sphere representation of the tunnel together with an extension parameter ε is used to determine the ROI. The ROI will bound the area where the algorithm will look for the extended shape of the tunnel. The radius of each sphere $s \in T$ is enlarged by ε resulting in a new sphere s'. The union of all spheres s' defines the ROI (see Figure 4).

As a first step, an axis-aligned bounded box (AABB) b is computed, containing all spheres from the T whose radii were enlarged with the value $\varepsilon + r$. Then, all spheres (atoms) from A are checked whether they intersect with bounding box b. The intersecting spheres are included into the set A_{in}, hence the set A contains only spheres occupying a space in ROI.

On line 5, a uniform grid g with resolution d is created, so that it spans bounding box b. Lines 6 and 7 are key to the

speed of our algorithm. Instead of fitting a probe with radius r, i.e. checking collision with spheres in A_{in}, we inverse the process. We label the ROI in grid g (line 6) by storing ones in vertices of voxels inside ROI. Now, the void in g can be thought as there were no spheres in A_{in}. To take into account the spheres in A_{in}, we enlarge their radii by r and store zeros to all vertices in g that are covered by these enlarged spheres (line 7). The labeled grid g now describes interior of voids in the ROI. Each vertex in g with value "1" represents a centre of a void with probe's shape and size. To obtain the actual void, we take the probe, place its centre to a labeled *void vertex* and set all vertices contained in the probe also to one. In fact, this step can be reduced to placing probes only at vertices that have some *non-void vertex* (labeled with zero) in their neighbourhood. Therefore, we firstly find all relevant vertices – denoted as P_{surf} and then place probes only at them (see lines 8 and 9).

The algorithm in its basic form, outlined in this section, would quickly exceed memory or would take too long in processing. For instance, storing grid data directly as a bit array or placing a probe (in step 9) into each void vertex would noticeably limit algorithm's performance. In the following parts, we describe important concepts that our algorithm employs.

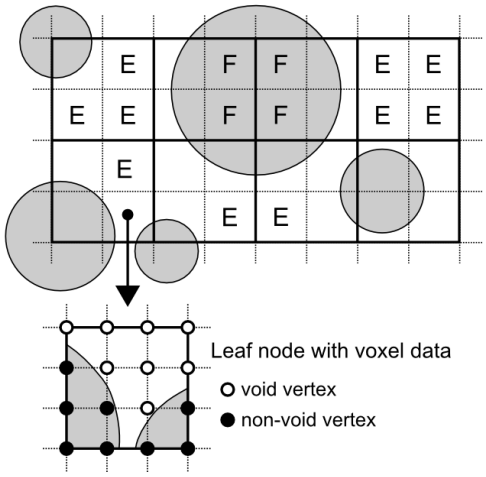


Figure 5: Multi-level grid with segmentation 4x4 on leaf level and 2x2 on parent level with full (F), empty (E) and partially occupied nodes.

3.1. Multi-level grid data structure

As described above, our algorithm utilizes a uniform grid to store information about the real shape of an enlarged tunnel. There are several ways to store a set of voxel data (e.g. [LK11], [HKK07]). We decided to use a three dimensional bit array. In order to lower the memory requirements, we have implemented it as a *multi-level grid*.

The multi-level grid is actually a simple tree-like data structure similar to octree – it decomposes a space by recursive subdivision into a given number of rectangular blocks. Each block stores information whether the corresponding area is empty (i.e. a void), fully or partially occupied by atoms. In the last case, the block either stores a 3D array of children descriptors or it is a leaf block. Every leaf block contains a 3D bit array which fully covers its space and stores the actual voxel data (see Figure 5).

This arrangement is much more memory efficient compared to uniform tree structure, since each voxel in the leaf block can be expressed by a single bit, however, the empty or fully occupied blocks can be indicated by an information stored in their parent blocks.

The resolution of the multi-level grid is adjustable on each level and depends on specific requirements. In fact, there are two properties that influence the size of a data structure in memory. First, the parameter d (see Section 3) defines the initial resolution of a regular grid and hence directly determines the amount of memory taken by the data structure. Second, the memory demands are also affected by the subdivision settings. Every level of the grid structure can define a different grouping of siblings to allow the data structure to be more flexible. For example, a parent node in octree groups always 8 children (2x2x2 boxes). On the contrary, the multi-level grid data structure defines the grouping for leaf nodes which sets dimensions of the bit array and another grouping that defines dimensions of children descriptor arrays in nodes.

The multi-level grid is build in the bottom-up manner. First the size of the low level grid is determined. The computed number of voxels together with the user defined subdivision settings gives the maximum depth of the multi-level grid. On each level the group of voxels or children blocks is clustered to create a parent node. Note that if all the clustered voxels or children blocks are of the same value, the parent block stores only this value. In some cases it is not necessary (or it might even be inconvenient) to create all possible levels since the top level would be too coarse. For instance, for the tests described in the section 5 we used only two-level grid.

There is no explicit rule on how to set the multi-level grid so that it would use memory efficiently. The data structure efficiency depends on a specific application, therefore it may require to do some tuning on target data. For example, among researched groups of proteins, some may have thousands, and others hundreds of thousands of atoms. The different groups will probably contain tunnels of different dimensions and required grid settings will differ significantly.

3.2. Enhancing region of interest

In the algorithm overview, we have sketched how we select ROI. We make use of spheres provided with the tunnel skeleton from the CAVER algorithm, but using only that information could produce unwanted results. The set of spheres can contain one or more spheres lying outside the protein molecule. Even when the computed set itself is correct, users could set the value of the extension ε big enough such that the ROI would exceed the molecule and hence the algorithm would look for the extended shape of a tunnel outside the protein. To avoid this, we compute the protein surface using LSMS algorithm [TCIYF06] and then simply cut of the redundant parts of the ROI lying outside the molecule. The difference is depicted in Figure 6.

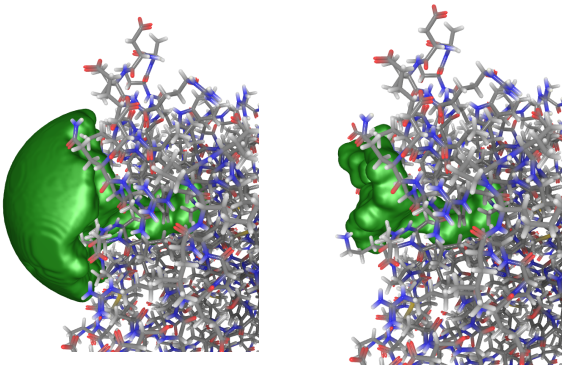


Figure 6: The tunnel computed without (left) and with (right) surface constraints defined by LSMS algorithm.

3.3. Molecular surfaces

Now, we show that both partial and final results of our algorithm actually represent concepts widely used in biochemistry. In different stages, our algorithm produces a grid of voxels representing two types of molecular surfaces.

At first, *solvent accessible surface* (SAS) of the tunnel is found. The SAS is defined by centre of a probe of given radius rolling along the atoms (see Figure 7). In our algorithm, this occurs after the ROI is marked and all enlarged atom spheres that influence voids in the ROI, are placed. Usually, the SAS is found by trying to fit a probe to all vertices of a regular grid. In our algorithm, we have inverted the process. The inversion is correct with respect to the SAS definition. From Figure 7, it is obvious that the SAS can actually be defined as a surface of voxelized void complementary to atoms enlarged by the probe radius.

Note, that every atom which can influence the SAS has to satisfy the condition that the distance of its surface from the ROI is smaller then the radius of the probe. Since the ROI is defined as a set of spheres, the collision test falls back to a trivial test of distance between two points. Moreover, we can

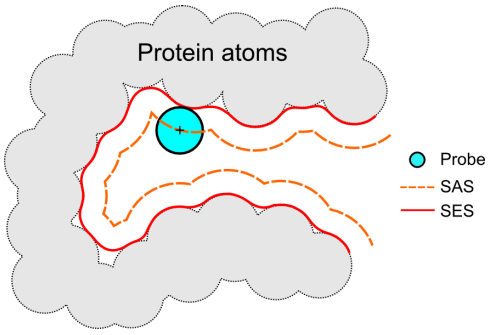


Figure 7: The difference between SAS and SES. SAS is defined by the centre of a rolled probe while SES is defined by points on probe's surface that are closest to molecule's atoms (when rolled).

store all atoms in some efficient data structure (e.g. in kd-tree or octree) to speed up the selection of the nearest atoms.

Finally, the SAS obtained in early discussed steps of the algorithm, is a reduction of the real void. Basically, this means that the acquired tunnel is smaller than it actually is. Therefore, the steps that find boundary vertices P_{surf} of SAS and place probes to every vertex $v \in P_{surf}$, adjust the computed voxel body to create *solvent excluded surface* (SES). The SES is defined by points on probe's surface that are the nearest to atoms when the probe is rolled (see Figure 7).

To obtain the SES, the algorithm has to find vertices from P_{surf}, i.e. vertices defining the SAS in the computed grid. All void vertices in the grid are evaluated using 6-adjacency – if a void vertex has at least one of its six neighbours marked as non-void it is noted as a *surface vertex*. Details on correctness of using 6-adjacency are presented in the next subsection.

3.4. Accelerating probe placement

The algorithm computes resultant voids by placing probes in surface vertices. It is in fact the dilatation of the SAS as defined in mathematical morphology using probe as a structuring element. For vertex evaluation, we use 6-adjacency because using 18-adjacency or even 26-adjacency would not change the result of the algorithm. If we have a void vertex that is not a surface vertex in 6-adjacency but would be in 18- or 26-adjacency, then it has to have six neighbouring vertices set as void. These neighbouring vertices together influence all the vertices that would be influenced by the non-surface vertex too. Then, it is correct to place probes only at 6-adjacent surface vertices which lowers the placed probe count.

Using 6-adjacency we obviously lower the computation time of the algorithm. To accelerate the probe rendering step even more we prepare the shape of probe transferred to a

uniform low-level voxel grid in advance. We also utilize the properties of the multi-level grid structure and do not place probes in nodes that are already completely empty. This way, we obtain a voxel body whose surface represents a SES of the tunnel. The process of generating resultant surface from the computed voxel data is described in section 4.

3.5. Special cases: Two or more voids

In order to complete the description of our algorithm, we also mention its special cases. By definition, the algorithm detects any void greater then probe with radius r which is located within the ROI specified by tunnel's sphere representation and parameter ε. Therefore, it is possible that we detect two or more spatially disconnected voids. Often, this happens when a large ε, or a probe greater than the original probe used to detect the tunnel in Voronoi diagram, are applied. In our algorithm, we suppose the analysed tunnel to be the largest void found. We label all the voids found, as components w.r.t. 6-adjacency and take the one with greatest volume (approximated by void vertex count). Finally, the event of detecting two ore more voids is reported to let the user decide on result's relevancy.

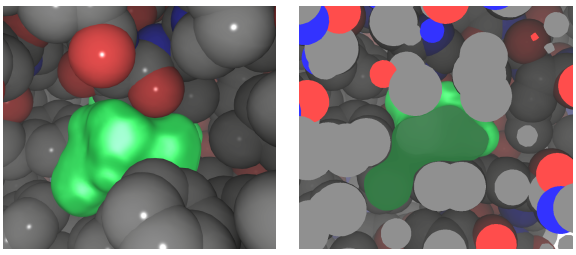
4. Visualization

The tunnel volume is visualized to provide user with clearly understandable information of its real shape. To create a surface mesh of the tunnel volume, we chose Marching Cubes [LC87] algorithm. Subsequently, the resulting mesh is smoothed by the Loop subdivision [Loo87]. The creation of the surface takes only seconds (see section 5.2), the parameters of the computation can be tweaked based on immediate results. Our algorithm was integrated into CAVER Viewer [Cav12], a GUI for molecular analysis by CAVER algorithm, where it can be easily controlled to satisfy various user needs.

4.1. Visualization of cross-sections

Often, it is useful to visualize a cross-section of a tunnel at some user specified spot. For example, biochemists want to know whether the narrowest region in a tunnel (represented by bottleneck value) has circular or rather elliptical shape. Visualizing the shape of a tunnel's cross-section together with its volume enriches the information given about the tunnel. We propose two methods for tunnel's cross-section visualization. Either a *direct visualization* in a molecule or an isolated *plot visualization*.

The direct visualization method (see Figure 8b) uses a clip plane to clip the mesh representation of a tunnel. The clip plane can be set perpendicularly to the skeleton of a tunnel or freely defined by user. Although the mesh representation is only an approximation, clipping gives users interactive information about tunnel's cross-sections. On the other hand, the plot method computes the area and other properties of a cross-section based on an intersection of a plane with tunnel's volume data. Here, the plane is always chosen such that it is perpendicular to the skeleton of the tunnel.



(a) (b)

Figure 8: (a) Asymmetric tunnel visualized without a clip plane; (b) visualization using interactively positioned clip plane.

The plot visualization algorithm uses voxel representation of a tunnel as was presented in section 3.1. Recall, that the data are stored in voxel vertices which means that the algorithm only has the information about probes with radius r that fitted (or did not) into some void in the molecule. Such a representation would only allow to compute intersection of vertices with the chosen plane.

When the condition expressed by equation 1 is satisfied, probes of radius r placed in vertexes of a three dimensional grid of density d will cover the space continuously. In such case, a representation of space using voxel vertices can be viewed as a valid cell representation with a smaller probe radius r'. Cells are cubes similar to voxels, with edges of length d for which it is trivial to compute intersections with a given arbitrarily oriented plane.

$$r \geq \frac{\sqrt{3}d}{2} \tag{1}$$

The resulting area of the cross-section is obtained by computing union of all intersected cells. Equation 2 expresses error E (in Å) that the plot visualization algorithm produces.

$$E = r - r', r' = \sqrt{r^2 - \frac{d^2}{2}} \tag{2}$$

The error is easy to deal with. The equation 2 can be used to calculate the needed probe radius r from the effective radius r'. Figure 9 shows example of a cross-section plot visualization.

5. Analysis & Results

Figure 10a shows the example of two tunnels in a protein computed by CAVER algorithm. Figure 10b shows

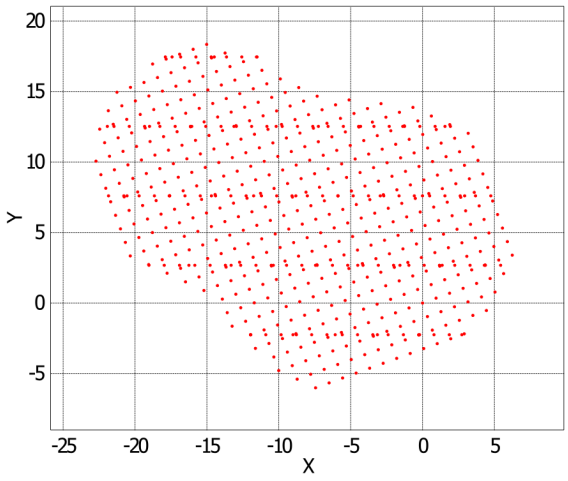
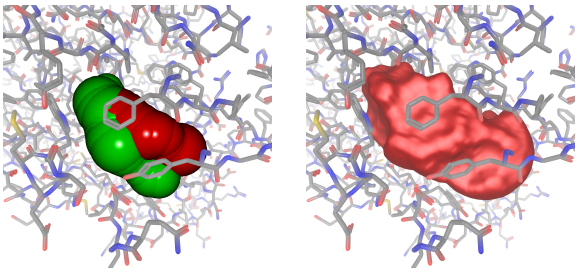


Figure 9: Plot visualization of cross-section of an asymmetric tunnel, calculated in the position of tunnel's bottleneck. Dots represent points where cell edges intersect with a plane perpendicular to original tunnel's skeleton.

the asymmetric tunnel computed by our algorithm using CAVER results as an input.



(a) (b)

Figure 10: (a) Two overlapping tunnels computed by CAVER; (b) asymmetric tunnel computed by our algorithm using precomputed tunnel as an input (red).

In the rest of the section we present results that demonstrate the performance and stability of the proposed algorithm. The algorithm was implemented in the Java programming language as a part of CAVER Viewer application. We performed series of tests on proteins contained in the on-line *Protein Data Bank* (PDB) chemical database [RCS03] – we use the PDB IDs from the database instead of long molecular names.

5.1. Stability tests

Since the algorithm is using multi-level grid which is axis aligned and discrete, the resultant shape of the tunnel depends on the grid orientation relatively to the protein position in the space. We have performed several tests to experimentally determine the stability of the algorithm with respect to protein rotation. In each test we have successively rotated the tested molecule by ten degrees around one of the axes: $(1,0,0),(0,1,0),(0,0,1),(1,1,1)$. Thus we have created 144 different space orientations for each tested protein. As an example, the results of three different tests on 1BRT molecule are shown in Figure 11.

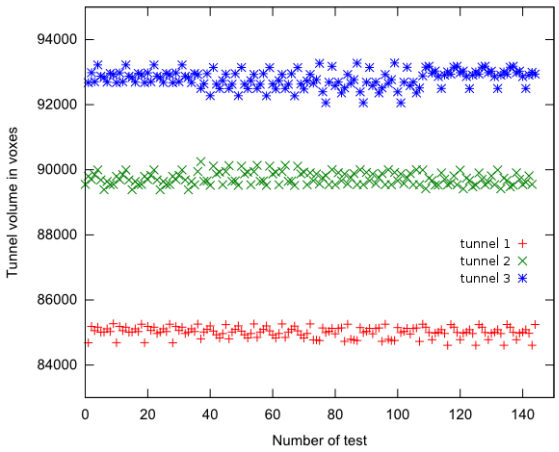


Figure 11: The test showing the stability of the algorithm during rotation of the protein molecule 1BRT

The tests were performed with settings: $\varepsilon = 2.0$ Å, $d = 0.2$ Å and $r = 0.9$ Å. The measurements provided the standard deviations in computed voxels 167.181, 194.381 and 267.868 which gives 0.20%, 0.22% and 0.29% deviations from the average number of computed voxels. All other tests we performed, provide similar results.

5.2. Average running time

We have also tested the average running time related to the volume of the computed tunnel. As the input for testing, several protein molecules, each with multiple precomputed tunnels, were used. We have run the algorithm with different settings.

The average time of ten computations with eight different tunnels in 1BRT molecule (related to the total volume of the multi-level grid and hence to the size of the tunnel) are depicted in Table 1. The test was performed on Intel Core2 Quad Q9550/2.83 GHz, 4GB RAM, Windows 7 Pro. The algorithm was run with the settings: $\varepsilon = 2.0$ Å, $d = 0.2$ Å and $r = 0.9$ Å.

6. Conclusion and Future work

In this paper, we have described the algorithm for rapid and more accurate protein tunnels computation. The proposed algorithm is successfully combining two different approaches

#	Volume of grid (Å^3)	Average running time (s)
1	1806.336	0.521
2	2654.208	0.656
3	4505.600	1.068
4	5947.392	1.299
5	7614.464	1.868
6	9371.648	1.861
7	10383.360	2.747
8	14536.704	2.886

Table 1: Average running time of the algorithm

from current tools focused on protein analysis. In the last section, we have briefly discussed the reasonable stability of the algorithm with respect to the protein rotation. We have also shown that it is possible to compute a more realistic shape of the tunnel in the order of seconds for medium-sized proteins, on common desktop equipment.

For future research, many issues remain as challenges for geometry-based analysis and for interactive visualization of proteins. Here we mention only few topics. First, the multi-level grid data structure described in section 3.1 was designed to lower the memory demands of the algorithm. We suppose that it will be also possible to use its properties to run the algorithm in parallel on GPU. Second, we plan to develop an intuitive method for voxel body segmentation. The segmentation method would allow us to reason about importance of tunnel's parts instead of using ad-hoc parameter ε to define important parts to be included in the ROI. Finally, we hope that it will be possible to use the proposed algorithm to improve current tools for protein tunnel analysis. This topic, however, requires a focused biochemical research and evaluation of biochemical relevance, at first. The combination of computational geometry, visualization and in-silico inspection of proteins and ribosomes opens new fields of research.

References

[ABN95] AGRE P., BROWN D., NIELSEN S.: Aquaporin water channels: unanswered questions and unresolved controversies. *Current Opinion in Cell Biology 7*, 4 (Aug 1995), 472–483. 1

[BCG*13] BREZOVSKY J., CHOVANCOVA E., GORA A., PAVELKA A., BIEDERMANNOVA L., DAMBORSKY J.: Software tools for identification, visualization and analysis of protein tunnels and channels. *Biotechnology Advances 31*, 1 (2013), 38–49. 1, 2

[BF08] BOSCO H., FRANZ G.: Hollow: Generating accurate representations of channel and interior surfaces in molecular structures. *BMC Structural Biology 8*, 1 (2008), 49. 1

[Cav12] CAVERSOFT: Caver - software tool for protein analysis and visualisation, 2012. http://www.caver.cz [accessed 21-May-2013]. 6

[CPB*12] CHOVANCOVÁ E., PAVELKA A., BENEŠ P., STRNAD O., BREZOVSKÝ J., KOZLÍKOVÁ B., GORA A., ŠUSTR V., KLVAŇA M., MEDEK P., BIEDERMANNOVÁ L., SOCHOR J., DAMBORSKÝ J.: Caver 3.0: A tool for the analysis of transport pathways in dynamic protein structures. *PLoS Computational Biology 8*, 10 (2012), e1002708. 1

[CS09] COLEMAN R. G., SHARP K. A.: Finding and characterizing tunnels in macromolecules with application to ion channels and pores. *Biophysical Journal 96*, 2 (Jan 2009), 632–645. 2

[GDC07] GOLD V. A., DUONG F., COLLINSON I.: Structure and function of the bacterial sec translocon. *Molecular Membrane Biology 24*, 5-6 (2007), 387–394. 1

[GM05] GOUAUX E., MACKINNON R.: Principles of selective ion transport in channels and pumps. *Science 310*, 5753 (Dec 2005), 1461–1465. 1

[HKK07] HARADA T., KOSHIZUKA S., KAWAGUCHI Y.: Sliced data structure for particle-based simulations on gpus. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia* (New York, NY, USA, 2007), GRAPHITE '07, ACM, pp. 55–62. 4

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics* (1987), vol. 21, ACM, pp. 163–169. 6

[LK11] LAINE S., KARRAS T.: Efficient sparse voxel octrees. *Visualization and Computer Graphics, IEEE Transactions on 17*, 8 (2011), 1048–1059. 4

[Loo87] LOOP C.: *Smooth Subdivision Surfaces Based on Triangles*. Department of mathematics, University of Utah, Utah, USA, Aug. 1987. 6

[MBS07] MEDEK P., BENEŠ P., SOCHOR J.: Computation of tunnels in protein molecules using delaunay triangulation. *Journal of WSCG* (2007). 2

[PCMT09] PELLEGRINI-CALACE M., MAIWALD T., THORNTON J. M.: Porewalker: a novel tool for the identification and characterization of channels in transmembrane proteins from their three-dimensional structure. *PLoS Computational Biology 5*, 7 (Jul 2009), e1000440. 2

[PKKO07] PETŘEK M., KOŠINOVÁ P., KOČA J., OTYEPKA M.: Mole: A voronoi diagram-based explorer of molecular channels, pores, and tunnels. *Structure 15*, 11 (2007), 1357–1363. 1

[RCS03] RCSB: Rcsb protein data bank - rcsb pdb, 2003. http://www.rcsb.org/pdb/home/home.do [accessed 05-February-2013]. 7

[SNW*96] SMART O. S., NEDUVELIL J. G., WANG X., WALLACE B., SANSOM M. S.: Hole: a program for the analysis of the pore dimensions of ion channel structural models. *Journal of Molecular Graphics 14*, 6 (1996), 354–360. 2

[TCIYF06] TOLGA C., CHAO-I C., YUAN-FANG W.: Efficient molecular surface generation using level-set methods. *Journal of Molecular Graphics and Modelling 25*, 4 (2006), 442–454. 5

[VG10] VOSS N. R., GERSTEIN M.: 3v: cavity, channel and cleft volume calculator and extractor. *Nucleic Acids Research 38*, Web Server issue (Jul 2010), W555–562. 1

[YFW*08] YAFFE E., FISHELOVITCH D., WOLFSON H., HALPERIN D., NUSSINOV R.: Molaxis: efficient and accurate identification of channels in macromolecules. *Proteins 73*, 1 (2008), 72–86. 1

[ZM10] ZHOU H. X., MCCAMMON J. A.: The gates of ion channels and enzymes. *Trends in Biochemical Sciences 35*, 3 (Mar 2010), 179–185. 1