UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Visualization of molecule surface dynamics

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Raphael Menges

Erstgutachter: Prof. Dr.-Ing. Stefan Müller
Institut für Computervisualistik, AG Computergraphik

Zweitgutachter: Nils Lichtenberg, M.Sc.
Institut für Computervisualistik, AG Computergraphik

Koblenz, im Oktober 2016

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Masterarbeit

Raphael Menges

(211100712)

Topic: **Visualization of molecule surface dynamics**

The surface of a molecule gives important indications of its bioactivity and interaction behavior with other molecules. During dynamic simulation processes the molecule's surface changes over time. Therefore a highly accurate and fast implementation of surface atom detection is necessary to provide scientists an efficient workflow.

This thesis will examine the implementation of surface extraction algorithms on highly parallel graphics processing units as well as a visualization of surface dynamics for given data of simulated processes. Particularly the change of the atomistic surface composition of a molecule during a simulation is important information for scientists which shall be visually extracted. Realtime interactivity is desired but optional. The whole implementation should be embedded into existing work of the Computer Graphics Research Group at the University of Koblenz.

Tasks:

1. Investigation of existing approaches for surface atom extraction.
2. Implementation of one or more algorithms for surface extraction on graphics processing units.
3. Comparison of accuracy and speed of own implementation to existing ones.
4. Visualization of molecule dynamics.
5. Evaluation of the overall results.

Koblenz, den 21.04.2016

Raphael Menges

Prof. Dr. Stefan Müller

Abstract

The surface of a molecule holds important information about the interaction behavior with other molecules. Amino acid residues with different properties change their position within the molecule over time. Some rise up to the surface and contribute to potential bindings. Other descent back into the molecular structure.

Surface extraction algorithms are discussed and for the most appropriate one a highly parallel implementation is proposed. Layers of atoms are extracted by an iterative application of the algorithm. This allows one to track residues in their movement within the molecule in respect to their distance to the surface or core. Sampling of the surface is utilized for approximations of further values of interest, like surface area.

Novel visualization methods are presented to support scientists in inspection of simulated molecule foldings. Atoms are colored according to their movement activity or an arbitrary group of atoms can be highlighted and analyzed. Proximity of residues to surface or core can be calculated over simulation time and allow conclusions about their contribution.

Zusammenfassung

Die Oberfläche eines Moleküls bietet wichtige Rückschlüsse über die Interaktionsfähigkeit mit anderen Molekülen. Aminosäuren mit unterschiedlichen Eigenschaften ändern ihre Position innerhalb der Molekülstruktur über die Zeit hinweg. Einige steigen zur Oberfläche auf und tragen so zu potentiellen Bindungen bei. Andere fallen zurück in die Molekülstruktur.

Verschiedene Extraktionsalgorithmen für die Molekülоberfläche werden diskutiert und für den geeignetsten eine hoch parallele Implementation aufgezeigt. Schichten von Atomen werden durch eine iterative Anwendung des Algorithmus extrahiert. Diese erlauben die Verfolgung von Aminosäuren in Bezug auf Distanz zur Oberfläche bzw. des Kerns des Moleküls. Außerdem wird eine Abtastung der Oberfläche zur Annäherung von weiteren aussagekräftigen Werten genutzt, wie zum Beispiel dem Flächeninhalt der Oberfläche.

Neuartige Visualisierungsmethoden werden präsentiert, um Wissenschaftler bei der Betrachtung von simulierten Molekülfaltungen zu unterstützen. Atome werden korrespondierend zur ihrer Bewegungsaktivität gefärbt oder frei wählbare Gruppen von Atomen können hervorgehoben und analysiert werden. Die Nähe von Aminosäuren zur Oberfläche oder dem Kern kann über die Simulationsdaten hinweg berechnet werden und erlaubt Rückschlüsse auf ihren Einfluss.

Acknowledgements

I would like to name some persons who supported my thesis with their knowledge and work:

Adrian Derstroff has implemented a fast neighborhood search for molecular structures by extending the framework. He supported me greatly in integration and debugging purposes.

Ajay Abisheck Paul George, **Pascal Heimer** and **Prof. Dr. Diana Imhof** from Pharmaceutical Institute at University of Bonn provided me with important feedback about biomolecular aspects.

Nils Lichtenberg as he gave me many references in existing publications and advices for the novel methods that are discussed in this thesis.

Daniel Müller and **Milan Dilberovic** helped me with extensive feedback about my thesis.

Contents

1 Introduction

The surface of molecules provides important information about the binding behavior. In 1971, Lee and Richards described the topology of the surface as directly related to the molecule's function and interaction with other molecules [8]. This information is highly useful in fields like drug design or medical research where molecules are arranged, simulated and evaluated for interaction behavior with existing molecules. Since the outcome of these evaluations is an important indicator for the scientists, it is worth to be supported by advanced visualization techniques and data acquisition methods. Modern consumer hardware can be utilized to provide results within seconds, even in situations with thousands of simulation steps and a high count of atoms. Fast precomputation and real-time rendering is a key factor for an efficient work flow. Especially when parameters of surface evaluation are controllable, no heavy recomputations for visualization should be necessary. This thesis covers the extraction of molecular surface atoms after traditional definitions as well as novel visual representations and data acquisition methods to support scientists in their decisions.

1.1 Motivation

Simulation of molecular folding generates a huge amount of data with thousands of positions for each atom over time. This is hard to evaluate without utilities, which support the inspection by highlighting interesting aspects. Existing suites like Chimera[1] or VMD[2] enable users to extract the triangulated SES, but struggle to support the evaluation of surface dynamics in simulated foldings. Therefore an application, named *SurfaceDynamicsVisualization*, specialized for evaluation of the surface dynamics over a computed simulation is proposed. Multiple visualization techniques and data acquisition methods are integrated to provide scientists with more compact information about the molecule's or amino acid residues' behavior.

1.2 Molecular Background

A protein is a composition of certain molecules or atom groups, called amino acids. Amino acid residues are connected as a linear chain, called primary sequence, which forms the protein. There exist twenty essential amino acids. Structure of the backbone atoms is shared by all amino acids and does not differ. The side chain is unique for each amino acid and defines its characteristics. Linkage of the linear amino acid chain is solely maintained over the backbones.

[1] `https://www.cgl.ucsf.edu/chimera`

[2] `http://www.ks.uiuc.edu/Research/vmd`

This chain of residues folds in order to settle down in an energy minimized region. The mentioned linear chain is therefore an idealized model, which is not directly recognizable in the folded protein. The appearance is more of a coiled three-dimensional structure. Proteins get folded in such a way that some of the residues remain within the core and others stay at the surface. The binding and interaction properties are mainly defined by the folded structure and is of special interest in research.

There are three structure levels within a protein:

- **Primary Structure** Linkage between amino acid residues on a covalent basis. Referred as linear chain or primary sequence in prior.
- **Secondary Structure** Areas of folding within the protein, which are stabilized by hydrogen bounding. For example helices or pleated sheets.
- **Tertiary Structure** Three-dimensional structure of a protein is result of a high number of non-covalent interactions between residues. For example two Cysteine (CYS) residues can form a disulfide bridge.

Our research cooperation partners at University of Bonn are especially interested in conotoxins. These are proteins that consist of 10-50 amino acid residues and are extracted from the venom of marine cone snails. This venom interacts with biological targets like ion channels or transmembrane receptors. Conotoxins are disulfide-rich proteins and certain residues, located on the surface, cause distinct bioactivity. The synthesis of such peptides' disulfide formation is achieved during folding from a linear precursor. Multiple μ-conotoxins are evaluated, which differ in the order of amino acids within the primary sequence. This results in unique affinity and specificity for their biological target.

Our cooperation partners want to examine the surface and general structure of this kind of proteins in order to get a deeper insights into their behavior during folding simulation. In further writing, the general term *molecule* is mostly preferred over protein.

1.3 Surface Definition

There are three widely used definitions of molecular surface that are closely related, as visualized in Figure 1. The atoms forming a molecule are generally interpreted as force field around their centers with a specific radius of influence. This radius is defined by the *Van der Waals* (VdW) value per element type and can be looked up in a table. It is assumed to be static. For simplification of the model this value is interpreted as radius of a sphere, which is positioned at the atom's center. By drawing these spheres, the

molecule is visualized as sphere cloud without explicit surface but consists of single spheres.

An improvement has been proposed by Lee and Richards [8] with the *Solvent Accessible Surface* (SAS). It takes a binding partner in appearance of a solvent probe into account. The probe is defined by the VdW value of hydrogen and tested for chances to access the atoms of the molecule without intersection. This is visually achieved by adding the radius of that probe to the radius of each atom in the molecule. Now having that *extended* hull for each atom, the rendering of the molecule with extended hulls represents valid positions of a virtual probe's center, which is rolled over the molecule and touches – but never intersects – the atoms. The atoms that extended hull is not completely covered by other atoms' extended hull are considered as surface atoms, because they can be accessed by that probe. Otherwise atoms are classified as internal.

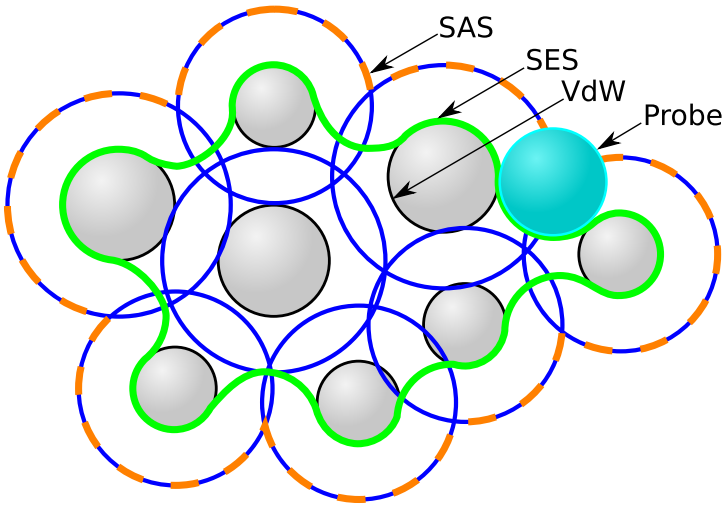


Figure 1: Definitions of molecular surface. The atoms are displayed in gray color, an exemplary probe is colored in cyan. VdW hulls are marked in blue, where the union of all extended hulls (SAS) is displayed as orange, dashed line. The SES is visualized as green line.

Further development of this technique is the later called *Solvent Excluded Surface* (SES), originally presented by Richards [11] in 1977. The surface is described by the solvent probe's sphere which is rolled over the molecule, too. The boundary of the volume that is not reachable by the probe without intersection with atoms is defined as the surface. The SES gives both impression of the binding possibilities and the spatial structure of the molecule. This marks an improvement because the SAS hides the original spatial structure of the molecule from an inspector.

1.4 Proposal

The SAS is sufficient for the intentions of this thesis, because the classified surface atoms are for both SAS and SES identically. The atoms that are accessible for a solvent probe are providing the major binding information. Therefore, a fast and exact extraction algorithm for these surface atoms is necessary in order to achieve useful results in later visualizations. Multiple approaches for this task are evaluated in the next section. The most appro-

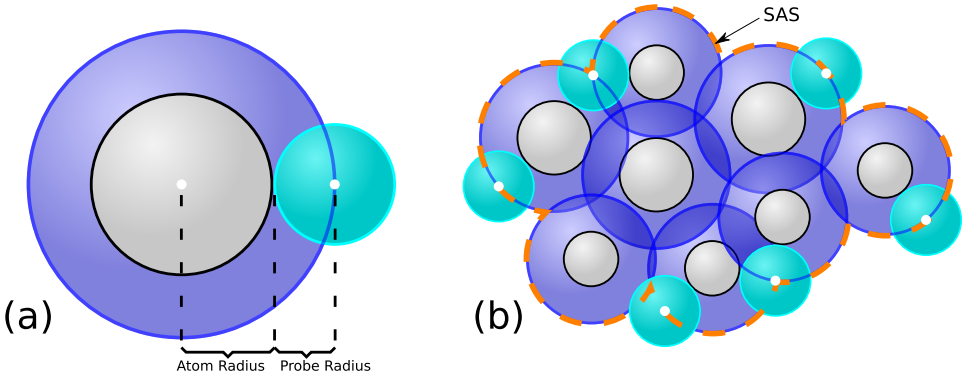


Figure 2: Definition of extended hull **(a)** and *Solvent Accessible Surface* (SAS) **(b)** in a cut through visualization. The atoms hull in gray is extended by the radius of the probe in cyan. This results in the extended hull of an atom colored in blue. The SAS is highlighted as orange dashed line. Only probes which lie on that line touch one or more atoms of the molecule.

priate one is discussed in depth in the third section and an implementation is realized in the fourth section. A novel visualization technique for the as- and descension of atoms to and from the surface during simulation is implemented and evaluated, enabling scientists to visually determine areas of high surface activity. Iterations of the surface extraction are applied to calculate multiple layers within the molecule, which are used to track the position of interesting atom groups or amino acid residues in the internal structures. In addition, samples on the SAS are generated for each computed step to approximate values like area or ratio of surface and internal samples. This may give hints about the stability of the molecule or the binding availability of an analyzed atom group at a given time step. The evaluation of the proposed and implemented ideas is documented in the fifth section.

2 Related Work

This section is split into two subsections. At first, established algorithms for surface extraction are discussed. Then, related work in surface dynamics visualization is presented.

2.1 Surface Extraction

Since the original publication about the Solvent Accessible Surface (SAS) was already written in 1971 and later extended to the Solvent Excluded Surface (SES), several methods have been proposed to calculate the surface of a molecule in an efficient manner. The ones listed below are representatives of separate approaches, which are shortly described and evaluated.

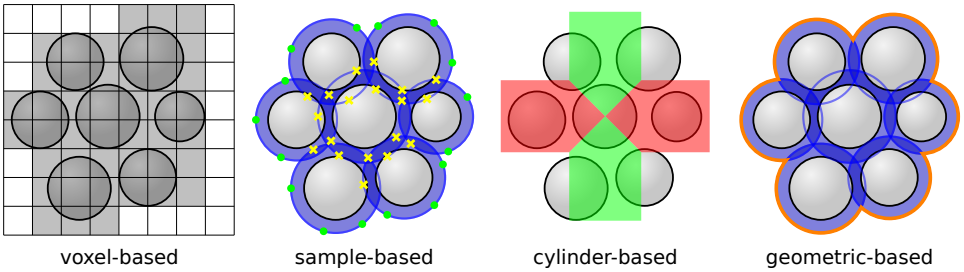


Figure 3: There are multiple methods for extracting the surface atoms of a molecule. **Voxel-based** algorithms utilize a voxel grid for classification. The **sample-based** approach classifies atoms by checking single sample points on the extended hulls. Another method – here called **cylinder-based** – searches within axis-aligned cylinders for atoms at the molecule's boundary. The **geometric-based** approach is implemented by accurate intersection tests of the extended hulls.

Original Algorithm

In the original publication of Lee and Richards [8] a drawing mechanism is employed to determine the accessibility of the atoms in terms of SAS. Each atom of the molecule is given as three-dimensional coordinate and VdW radius. The VdW radius of the solvent probe is added to each atom's radius and the molecule is sectioned by uniform spaced parallel planes. This results in an arbitrary count of circles per slice. These circles are merged in overlapping areas and form the area of the molecule for that slice. The drawn arcs per atom are collected in each slice and summed over the atom, while considering the spacing of the sectioning planes. The sum of the arc lengths is brought into relation with the computed area[3] of

[3] $A = 4\pi r^2$

the atom's extended sphere, which results in a percentage of accessibility for each atom contained in the molecule.

Since this algorithm relies on a drawing mechanism that is fed with slices, a perfect surface extraction is only possible by dividing the molecule into infinite or at least a very high count of slices and the application of an accurate drawing algorithm.

Voxel-based Algorithm

A similar approach is presented by Wei Lee and Bargiela [9]. They place the molecule inside a three-dimensional grid, but work on slices of it in order to reduce complexity of the algorithm. The voxel size is evaluated by experiment and a voxel is only marked as filled with atoms when it is occupied at a certain minimum of percentage with atoms, called *voxel occupancy*. An occupied voxel is classified as surface voxel by checking whether any of the four borders of the voxel inside the slice is neighboring to an unoccupied voxel. Instead of directly considering all atoms intersecting such a surface voxel as surface atoms, further filtering is applied. Another value named *degree-of-belonging* is introduced to decide whether an atom is considered to belong to voxel. Only atoms that intersect over a given percentage a surface voxel are classified as surface. After that step, an additional peeling process is executed to eliminate further false-positive classifications. For each voxel is marked the atom which is furthest away from the averaged center of the molecule. All other atoms of that voxel are checked for external exposure. It is decided by the ratio of external and internal exposure whether a classification as surface or internal is applied.

Output is a list of potential surface atoms, which is compared – amongst others – to SAS and SES calculation outputs of other tools. Because this algorithm has many both false-positives and false-negatives, it is not pursued further. In addition, voxel occupancy, degree-of-belonging and the voxel size are determined by experiment and may depend on the reviewed molecular structure.

Sample-based Algorithm

For extraction of the SES, Byungjoo, Ku-Jin and Joon-Kyung [5] suggest a sample based approach. At first a voxel grid is defined, which includes the bounding box of the molecule. For each cell every atom's extended hull is checked for intersection and listed in the cell for a positive result. Sample points are generated on the extended hull of each atom and tested against the extended hull of the neighboring atoms for inclusion. If a sample point is included by at least one other atom's extended hull, it is discarded. This test is accelerated by making use of the formerly created voxel grid, through supporting the choice of neighbors. Every other atom of all cells

that includes the atom – which created the sample point – is tested against the sample point. The surviving sample points are sorted into a kd-tree structure for faster spatial access. This acceleration structure is used to build up a grid of cubes, which is input for a *Marching Cubes* algorithm. The Marching Cubes algorithm extracts the SES of the molecule. This is performed by calculating the distance for each vertex of the cubes to the nearest sample point in the kd-tree and subtracting the probe's radius from this value. The result is then stored for each vertex in the grid. This forms some kind of distance field or volume, storing the distance to the SES at a certain position. In order to extract the SES, the Marching Cubes algorithm just has to output the zero level iso-surface.

This approach could be used for atom classification but since it is based on samples, the outcome depends on a high sampling density to fetch all surface atoms. In this thesis, sampling is not utilized for surface atom classification but for approximating the area of the surface and further analysis.

Cylinder-based Algorithm

Karampudi and Bahadur present a method to compare different molecules [3]. They define an unique layer peeling method for that purpose, where the surface computation is a kind of side effect. However the authors compare it to common methods like SAS. For each atom not yet classified in a layer, they propose to collect the unclassified atoms that have a maximum distance to the checked atom and whose center lies within one of three cylinders. These cylinders have to pass the checked atom and one is aligned to the x-, one to the y- and one to the z-axis of the global coordinate system. The user defines an uniform radius for the cylinders. It controls the count of atoms classified in a layer. All atoms, whose position is determined to be in maximum distance – in positive and negative direction – to a checked one inside a cylinder, are collected to form the next layer and are not further considered in the algorithm's executions. This is iterated until all atoms are classified into a layer. The outermost layer is marked as surface, the innermost as core. Every layer in between is called sandwich layer.

The peeled layers are utilized to mark the first occurrence of amino acids residues going from the innermost to the outermost layer of the molecule. Since the layer peeling is stated as being non-random, translation and rotation invariant, the order in occurrence of residues can be used as comparison method for different molecules.

This algorithm is highly depending on the choice in the cylinders' radius and it may be doubted that the alignment of the cylinders to the global coordinate system axes provides rotation invariant layer classification. However, the layer definition is interesting for dynamics analysis in this thesis. An iterative application of the surface extraction algorithm is utilized to extract multiple layers of the reviewed molecule.

Geometric-based Algorithm

An advanced approach for surface extraction is presented by Krone, Grottel and Ertl [7], as they propose a highly parallel version of the *Contour-Buildup* algorithm. In contrast to the former techniques, this is based on a three-dimensional geometric solution that takes the extended hull of the atoms and intersects each hull with the ones of all neighboring atoms. This generates circles, which indicate the direction the hull is cut away by a neighbor. These circles are checked against each other to filter only necessary ones, discard the whole atom instantly or to partly cut away each other, which results in circle arcs. The union of circle arcs forms the contour of the molecule, which is the starting point for rendering the SES. It is achieved by collecting the circle arcs, intersection points of arcs – called endpoints – and spaces in between for drawing polynomial surface patches with a pixel shader on impostor geometry, while avoiding singularities.
Using modern graphics hardware programming – as in the implementation part for this thesis – the authors describe a complex but highly sophisticated combination of rather small CUDA kernels to optimize the contour-buildup in many aspects. Instead of the naive approach to calculate the contour of one atom per GPU thread, the remaining arc of a single intersection circle is targeted as a thread's result. This saves one from calculating the identically intersection of neighboring atoms in two separate threads, which would handle both atoms independently. The rather small computation kernels are assigned to unique jobs:

1. Atoms are put into a grid and sorted spatially for faster access.
2. All intersection circles are calculated for each atom and over all its neighbors.
3. Unnecessary intersection circles are removed and some atoms are discarded due to being completely cut away by two intersections.
4. All circles are cut with every circle of the two atoms which created that circle by intersection.

The remaining arcs are used to render the SES. This algorithm scales very well for a higher number of processing units and is the upper limit to compare with.

Surface Atom Extraction based on Extended Spheres Since a SES hull is not necessary for the visualizations aimed for in this thesis, a related algorithm is chosen to determine the surface atoms regarding the accessibility to a solvent probe. *Surface Atom Extraction based on Extended Spheres* (SAEES) presented by Zhang and Shi [14] is similar to the

Contour-Buildup, but has no demand to save the exact split circle arcs. Instead, only those endpoints are evaluated, where two or more intersection circles cut each other. When an atom is at surface, either no endpoint is ever generated or at least one survives the cutting process with the other circles.

2.2 Surface Dynamics

It has not been found much information about approaches for surface dynamics visualization. Most publications concentrate solely on the real-time rendering of SES with ray-casting on the GPU. Krone, Bidmon and Ertl [6] present a combination of existing techniques like depth darkening [10], linear fogging and rendering of silhouettes. An interesting aspect of their publication is the simplification of most calculations by abstracting the atoms in back bones and side chains of amino acids to spheres.
Although their main contribution is about real-time SES rendering, too, they propose a novel method for visualizing the so-called *Spatial Probability Density*. They filter the global movement of the molecule over simulation time to extract the inner movement. Then a three-dimensional grid of uniform spaced voxels is defined and occurrence of atoms over simulation time is counted per voxel. Every voxel represents an atom that is positioned at the voxel's center and scaled uniformly. Its radius is defined by the user. This is necessary since atoms with different element types in one voxel may occur and no VdW value can be assigned. A threshold is set in order to select voxels that have collected at least a certain number of atoms over time. For these atoms a SES representation is computed and rendered with an opacity corresponding to the set threshold. Multiple of these representations can be blended together. Therefore, no further color encoding is suggested because the colors may be blended over each other and confuse the user.

3 Theory

Following section preserves an overview about the concepts implemented in context of this thesis. Implementation details are delayed to the next section.

3.1 Rendering

In order to be able to debug the surface extraction, a first step is to visualize the atoms' hull as sphere. Two approaches have been established for that task. Back in time when the pixel shader performance has been insufficient, spheres were rendered as triangle mesh. This enables both perspective and orthographic projections. Since the spheres are approximated by triangles, a high triangle count is necessary at low camera distance. This results in the demand for some sort of level-of-detail algorithm, when a large number of atoms shall be displayed. A more modern approach is called impostor rendering. The triangle geometry is limited to a single quad consisting of two triangles per atom. Local coordinates of that quad are used as coordinate system to determine the shading and depth for each rendered pixel on screen for representing a sphere. This approach guarantees pixel-perfect spheres without any need for level-of-detail algorithm. Because the impostors are oriented towards the camera's center point, a perspective projection becomes erroneous towards screen borders. Therefore, only orthographic rendering is enabled in the application developed as part of this thesis. This simplifies the cut with a view parallel plane, too.

Plane Cutting Cutting of the atoms with a view parallel plane is useful for checking the correctness of the surface extraction algorithm. All structures in front of that plane are clipped. When performing SAS rendering, all internal atoms should be surrounded by surface atoms at the cutting area, as far as the surface atoms have been calculated correctly. In addition, internal structures can be made visible to the user without extensive use of transparencies, which are expensive to render.

3.2 Surface Extraction

Starting point of the Surface Atom Extraction based on Extended Spheres (SAEES) [14] algorithm is the definition of the extended hull of an atom. This hull is a sphere with the origin at the atom's center and radius equal to atom's VdW value plus the radius of a probe[4]. The probe is virtually rolled over the molecule to extract the accessible surface of the molecule, as defined in SAS. The extended hull describes the volume in which the

[4] Radius of the used probe is 1.4 Angstrom in this thesis, if not other indicated.

probe's center cannot lie without the probe intersecting an atom. If the extended hull of an atom is completely covered by the extended hulls of other atoms, it is classified as internal, as shown in Figure 4. When there are areas on the extended hull of an atom that are not covered by any other extended hull, a probe's center could lie there and the atom is classified as surface atom in terms of SAS. Since the authors explain the algorithm mainly with sketches and some textual description, the exact steps and values to store were developed as part of this thesis.

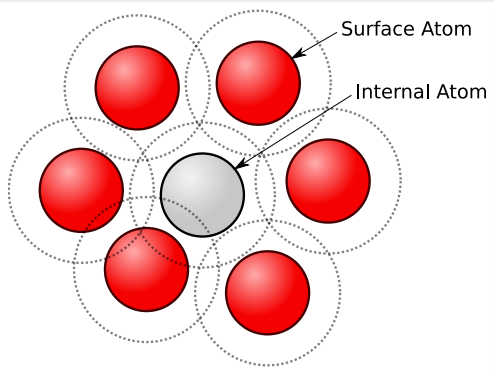


Figure 4: Idea of SAEES algorithm, visualized in 2D. The center atom's extended hull – drawn as dotted line – is completely covered by the extended hulls of other atoms. Every position where a probe could touch the atom's hull is inside the extended hull of another atom and the probe would collide with that atom. Therefore the atom in the center is declared as internal.

The concepts of sphere-sphere, plane-plane and line-sphere intersection are necessary to compute the cut of the hulls. Initially, cutting faces are generated by sphere-sphere intersection. These cutting faces describe the cut away of hull by another atom, through storing the plane equation of intersection and direction into which the hull is cut away. The list of computed cutting faces is optimized, because some are included by others and are not necessary for further execution. Or two cutting faces cut away the complete extended hull. This would not be covered by later tests, because these faces do not produce endpoints. At last, cutting faces are pairwise intersected at plane-plane basis and the resulting line is again intersected with the extended hull of the processed atom, described by a sphere. This line-sphere intersection may result in zero, one or two points, which are called endpoints. Every generated endpoint is then checked against all other cutting faces, whether it is cut away by any of them. If one or more endpoints survive that check or no endpoint has been generated at all, the atom is classified as surface. Otherwise as internal.

Layers An iterative execution of the SAEES algorithm provides a mechanism to extract multiple layers of atoms corresponding to the SAS definition. Layer zero is the outermost one and contains the same atoms as the SAS of the complete molecule. Further iterations take the internal atoms of the

previous algorithm execution's output and add the now as surface classified atoms as new layers to the data structure. This is done until no internal atoms are output by the algorithm. There are two interpretations for the layers that are both considered in the chosen implementation. Starting with layer zero as the outermost, the layer of an atom measures the distance to the molecule's surface.

This value may vary often for inner atoms, because the SAS definition is applied and the layer count may depend only on few active atoms at the surface. This is visualized in Figure 5. Therefore, a symmetrical definition is introduced, which marks the innermost layer as zero.

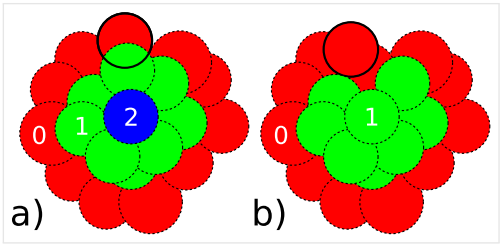


Figure 5: Cut through a molecule with colored layers. Position of outlined atom changes between **a)** and **b)**. This movement reduces the global layer count from three to two for this slice.

The general layer definition over the SAS differs from the publication by Karampudi and Bahadur [3], where atoms are searched inside axis-aligned cylinders for determining the layers of a molecule. The purpose of amino acid residue tracking is discussed in the following sections, too. However by computing the average instead of minimal layer of all atoms the residue consists of. This enables analysis of smaller molecules, since the residues' atoms minimal layer is almost always at core level in those.

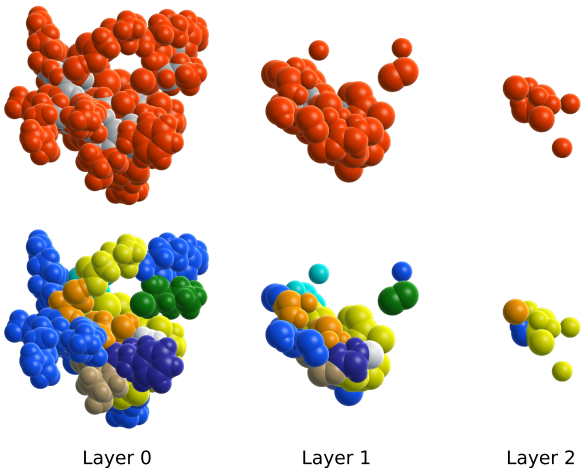


Figure 6: Three layers can be extracted for the depicted molecule by repeated application of the SAEES algorithm. Upper row shows the Hull visualization, lower the Amino Acids rendering mode.

3.3 Ascension

Over simulation time, some atoms ascent to the surface and other descent back into the interior of the molecule. It is important to know which atoms are rising up to the surface and are influencing the binding behavior of the molecule. Therefore, a visual feedback about the as- and descension process of atoms is proposed to enable users to determine areas of activity on the molecule's surface. Atom coloring of the process is an appropriate implementation of this feedback. The hue circle – known from HSV color space – is utilized for coloring the atoms depending on their state. The current state is encoded as angle between 0° and 360°, which is taken as input for the hue circle in order to fetch the color to display.

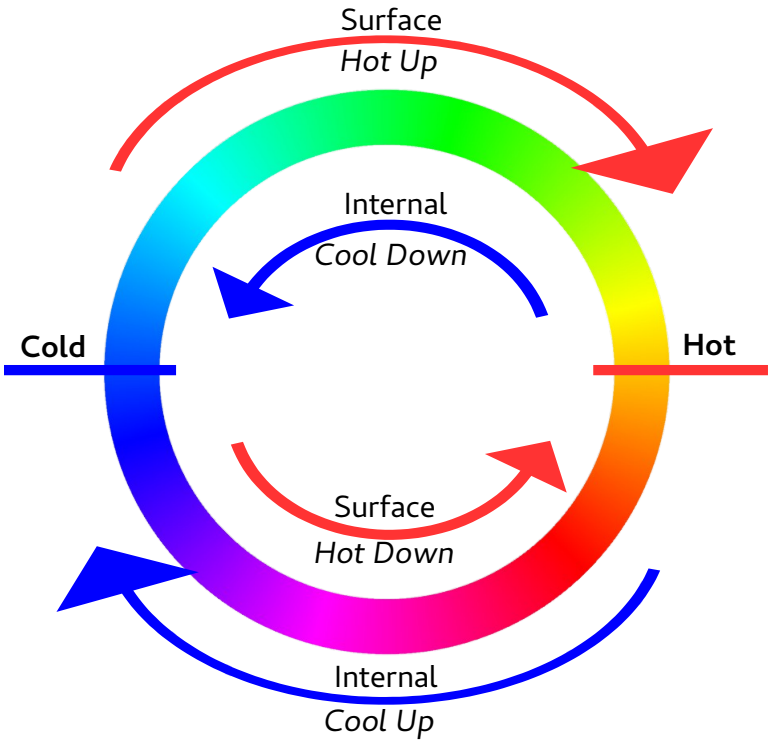


Figure 7: States of as- and descension are visualized by picking colors from the hue circle. An atom that is at the surface for a longer time becomes hot (yellow). Another one that is inside the molecule for a while becomes cold (blue). When a change in classification being internal or at surface is detected, movement towards the other convergence angle is started. If classification recovers to the prior state, the angle from which the color is taken falls back to the prior convergence angle.

There are two convergence angles on the circle that symbolize either a *hot* state at 45° – atom is for a longer period of time at surface – or a

cold state at 225° – atom is for a longer period of time internal. At the beginning of simulation the atoms' angle is at one of these two convergence angles, depending on whether the atom is classified as surface or internal in the first time step. During simulation the classification per atom whether surface or internal is processed and either the angle remains at one of the convergence angles or is interpolated between both. Figure 7 gives a visual impression of the process on the hue circle. There are four parameters that are presented to the user for tweaking:

- **Hot Up** Amount of simulation steps to get from cold to hot convergence angle while being classified as surface.
- **Hot Down** Amount of simulation steps to fall back to hot convergence angle on the way to cold while being classified as surface.
- **Cool Up** Amount of simulation steps to get from hot to cold convergence angle while being classified as internal.
- **Cool Down** Amount of simulation steps to fall back to cold convergence angle on the way to hot while being classified as internal.

3.4 Analysis

In order to enable further dynamics analysis of the molecule or groups of atoms, samples are generated on the atoms' extended hull. This is necessary because the information, which atoms are at surface and which are internal, is not sufficient to deliver approximations of surface area. For all atoms in the molecule, the same count of uniformly distributed samples is generated. In each folding simulation it step is checked whether a sample is classified as surface or internal, regarding the SAS.

Molecule Analysis

For the complete molecule, two values are extracted utilizing the samples on the extended hull:

- **Surface Amount** Ratio of samples on surface and internal samples. Gives indication about stability of the molecule at specific time step.
- **Surface Area** Approximation of surface area by taking the complete sphere's area of each extended hull and the ratio between surface sample and internal sample count. This is used to decide how much of the hull's area is actually at the SAS of the molecule and disposed to binding partners.

Atom Group Analysis

The user is able to select an arbitrary group of atoms, which may hold properties of interest. The values of surface amount and surface area are also extracted for these atoms. In addition, the iterative extraction with SAEES provides a layer classification, which is used to generate information about the position of the group within the molecule. Following values are computed for the analyzed group over folding simulation:

- **MinLayer** Minimal layer of all atoms in the group.
- **AvgLayer** Average layer of all atoms in the group.
- **Surface Amount** Ratio of the hull samples classified as surface and internal, regarding the SAS of the complete molecule.
- **Surface Area** Approximation of the surface area on the SAS of the complete molecule.

Path The mean center of the atoms in the analyzed group is displayed as path over the simulation steps. An user can determine the global spaced path length for a start- and end-step as well as the length of the locally spaced path. There, the molecule's center is subtracted from the group's center in each time step in order to provide only the relative movement of the analyzed group. This feature is useful to determine the movement and activity within the molecule for a group of atoms.

3.5 Residue Surface-Proximity

Special interest of scientists lies in the behavior of amino acid residues. There are residues that are expected to actively change their position with respect to surface or core of the molecule and others are considered stable. For evaluating their behavior, the average layer from the iterative surface extraction is tracked over simulation time for each set of atoms belonging to a residue.

Each atom is assigned a corresponding layer L_A. This value is collected for all atoms, of which an amino acid residue is made up. For the residue the mean of all their atoms' L_R value is computed. The L_R value provides a score of *Residue Surface-Proximity* in regards to SAS for

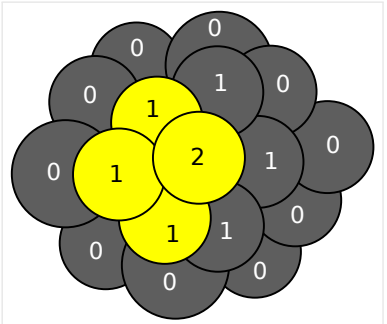


Figure 8: Cut through a molecule drawn with extended hulls. Layer membership L_A is written to atoms. L_R is $\frac{5}{4}$ for the yellow atom group.

each residue. If one subtracts L_A from the current layer count and proceeds with computations, core-proximity is given.

The average layer of an residue $L_R(x)$ is computed for each time step of simulation, with x as time step parameter. A delta value between consecutive steps is defined by:

$$\Delta L_R(x) = |L_R(x) - L_R(x-1)|$$

This delta value itself may be averaged for one residue over folding simulation in order to measure the activity in terms of layer changing during simulation. The approach of layer tracking is decoupled from the general spatial movement of the molecule. It reveals different residues competing within the molecular structure and influencing the layer memberships. The method differs from established ones like radius of gyration[5] or RMSD[6], which rely on euclidean space. See Section 5.2 for an application of the proposed Residue Surface-Proximity score.

[5] See Section 7.4 explanation.

[6] Root-Mean-Square deviation. Measure of the average distance between atoms and reference structure.

4 Implementation

This section provides details about the implementation of the presented concepts. Both CPU and GPU are utilized, depending on the benefit of parallel scheduling. Since some algorithms are more time complex than others, it is also decided by impact on later work flow.

4.1 Rendering

Drawing of atoms is accomplished by rendering an impostor geometry, which is replaced by a sphere in the pixel shader stage. Figure 9 gives an overview of the jobs assigned to each shading stage. In the C++ part of code **(a)**, all necessary buffers like SSBOs with atom radii and trajectory are filled and bound. The binding of further buffers depends on the chosen rendering mode. Most time, OpenGL is then advised to draw as many points as atoms are contained in the molecule. Each point triggers a vertex shader execution **(b)**, which takes the vertex id of the execution, the current time step and the atom count to determine the atom's position by a lookup in the trajectory buffer. Furthermore, the radius of the sphere is taken from the radii buffer and optionally summed with the probe's radius. Coloring of the sphere is depending on the chosen rendering mode. Smoothing of animation is realized by calculating the average of the atom's position at multiple steps in past and future. This means visual smoothing only, no data fed to calculations is altered.

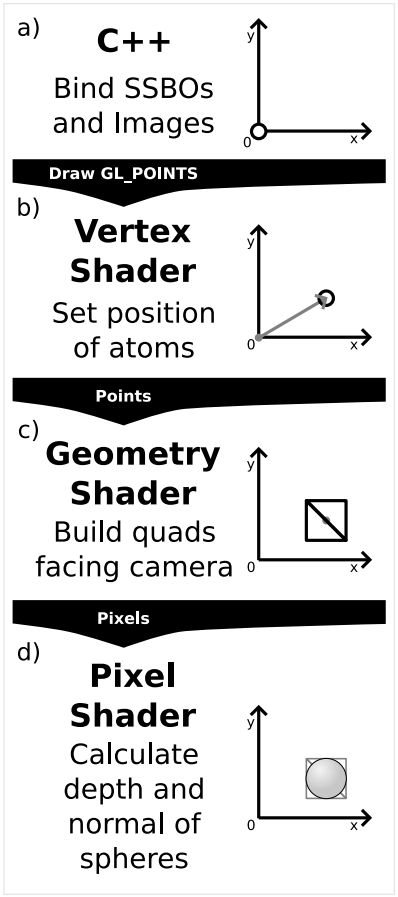


Figure 9: Process of rendering atoms.

The position is used in the geometry shader stage **(c)** for building up a camera facing quad, which replaces the input point. The base vectors of the camera coordinate system are read from the view matrix in order to define the camera-facing quad' coordinates, which has to be generated. In addition, the local coordinates between zero and one are stored for each vertex of the quad. This enables one to define a sphere in the next stage, the pixel shader.

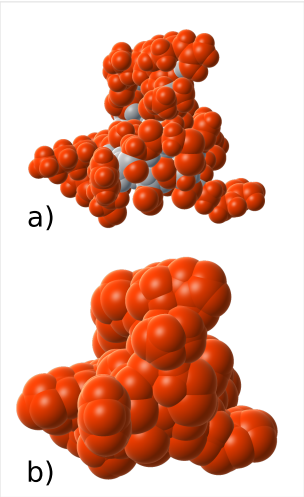


Figure 10: a) depicts the VdW hull and **b)** the SAS.

The pixel shader stage **(d)** is responsible to calculate the output color for each pixel generated by the rasterization of the quad. The local coordinates of the quad are used to discard all pixels that are outside of the sphere's radius measured from the center of the quad. Afterwards, the normal vector of the sphere is computed at the current pixel position. It is necessary for shading like Phong and Rim lighting, which are both combined in the visualization. Although there are multiple coloring modes like Hull, Element Type or Ascension, where both geometry and fragment shader are reused. If an include-system for GLSL would be employed, most parts of the vertex shaders could also be shared.

Plane Cutting Cutting the atoms at near plane instead of *popping* them out is already implemented in other applications [13, section 5.3, Z-clipping of impostors]. But instead of aesthetics, here clipping is a tool for visual validation of the surface extraction algorithm. Therefore, a distinct cutting plane distance can be set by an user. In addition, the camera can be aligned to one of the main axis for easier inspection.

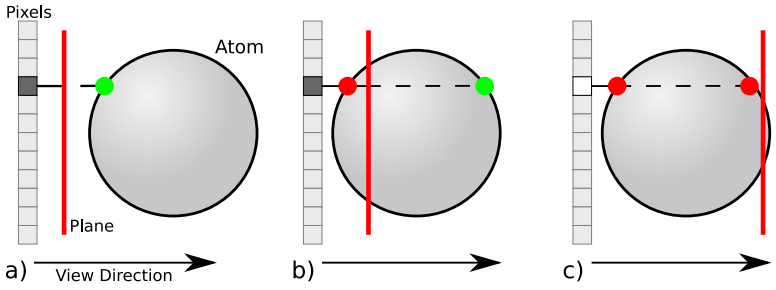


Figure 11: a) The cutting plane is in front of the atom and no cut occurs at the given pixel. **b)** Cutting plane cuts through the atom and pixel is drawn as cutting area. **c)** Since both the font and back intersection points of view-ray lie in front of the cutting plane, the pixel is discarded.

The algorithm to handle the cutting plane in the pixel shader stage is visualized in Figure 11. There are three possible cases for each rendered pixel. First case **a)** is the most trivial one, since the cutting plane is between viewer and atom without any intersection. The pixel can be rendered as if no cutting plane is set. In the second case **b)** and third case **c)**, the cutting plane lies behind the calculated depth of the pixel. To decide whether the pixel is shaded as cutting area or discarded, the depth at the back of the

sphere is tested for being in front or behind the cutting plane. If the depth at the sphere's back does not lie in front of the cutting plane, the pixel is used to visualize the cutting area, like depicted in **b)**. This means the normal vector is directed to the camera and the depth is set to zero. This differs from the implementation in [13], which yields for a realistic internal intersection by taking the calculated depth of the pixel into account. However, the primary usage here is to validate the surface extraction. So all internal atoms are rendered first, the surface atoms afterwards. These cannot overlay the internal cutting area due to the already set depth value for the processed pixel. In the last case **c)** the pixel is discarded, since it neither contributes to the sphere nor the cutting area.

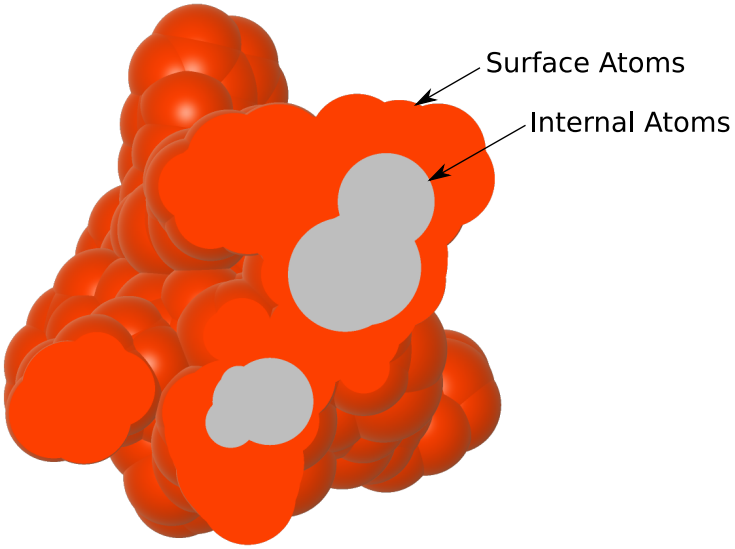


Figure 12: Cutting the atoms with a plane makes visual inspection of correctness of SAS possible. Atoms are rendered as spheres sized like extended hulls.

Framebuffer Structure There are multiple framebuffers, which are used for distinct purposes. This is necessary through the use of super sampling for molecule rendering and optional transparency:

- **Molecule** For impostor rendering only. The bound shaders depend on the chosen rendering mode. For a visually smooth experience, super sampling is optionally available. This scales up both width and height of the framebuffer with a factor of two. At composition, fast hardware linear interpolation is used. A drawback is the imperfect blending of the molecule's border with the background, because the black base

color of the framebuffer's color attachment bleeds into the pixels. In order to implement mouse picking, the atoms' index is encoded in an additional color attachment.

- **Selected Atom** This makes also use of impostor rendering but renders only the selected atom in green color. The output is drawn as transparent overlay at composition for easier localization of the selected atom.
- **Overlay** This framebuffer collects all other overlay related information in its color attachment, like path visualization or outline of analyzed atom group. Here, no super sampling is used, because the filtering would absorb parts of one-pixel-thick lines.
- **Standard** The standard backbuffer provided by the window library is initially filled with the cubemap background. Then all other framebuffers' attachments are blended together to the composited result by drawing it onto a screen filling quad. The user interface is rendered on top, which is powered by an immediate GUI implementation by *ocornut*[7].

Mouse Picking In an earlier version of the implementation, a geometric intersection test between view-ray and all atoms has been implemented in the C++ part of the code. But as the optional animation smoothing in the impostor vertex shader stage manipulates the display position of the atoms, an alternative approach must be employed. An extra color attachment of the molecule framebuffer has been implemented[8]. Each pixel writes the index of the atom – it is generated from – encoded as color into the additional color attachment of the framebuffer, while the molecule is rendered. When a right mouse click is perceived by the application, the attached texture is read at the click position and the atom's index is decoded by using the fetched color value.

Rendering Modes Multiple rendering modes for the molecule are available. They provide the user with diverse information about the atoms:

- **Hull** Rendering surface atoms in red and internal in white. Useful for inspecting the output of surface extraction algorithm, especially in combination with the cutting plane feature.
- **Ascension** Colors calculated by *Ascension* algorithm, which is described in Section 3.3 and implemented in Section 4.3.

[7] `https://github.com/ocornut/imgui`

[8] `http://www.opengl-tutorial.org/miscellaneous/clicking-on-objects/picking-with-an-opengl-hack`

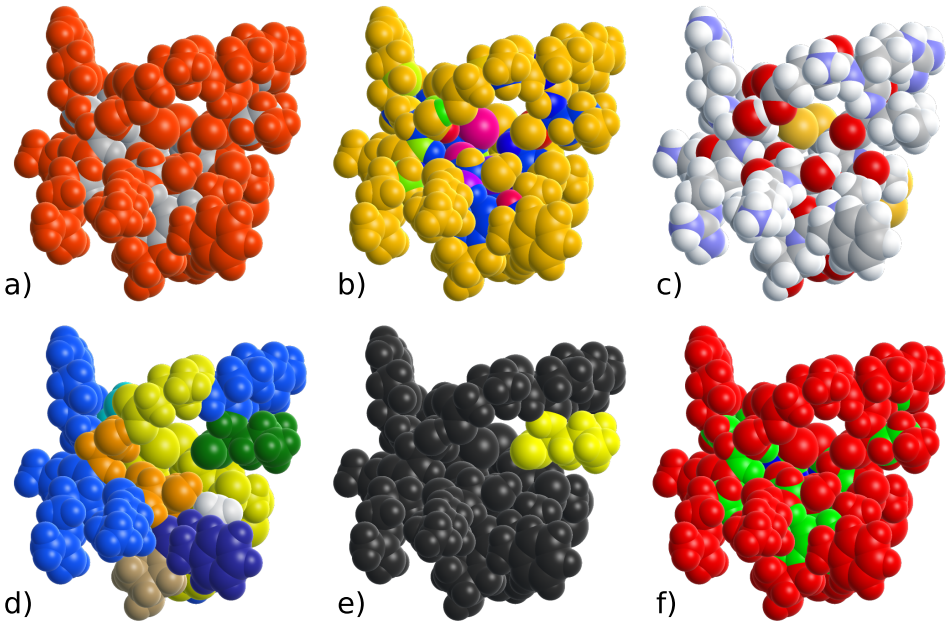


Figure 13: Available rendering modes: **a)** Hull, **b)** Ascension, **c)** Elements, **d)** Amino Acids, **e)** Analysis and **f)** Layers.

- **Elements** Atoms are colored according to their element type[9].
- **Amino Acids** Atoms are colored according to their amino acid[10].
- **Analysis** Coloring the atoms belonging to the analyzed group in yellow and others in black.
- **Layers** Coloring layers extracted with SAEES in different colors.

Atom Group Analysis The group of analyzed atoms is of special interest for the user. Therefore multiple approaches for highlighting the group has been implemented:

- **Pattern** Within the fragment shader stage of impostor rendering, the screen coordinates of the processed pixels are used to draw a moving pattern on the atoms that belong to the analyzed group. This is achieved by a combination of trigonometric functions and modulo operations. It helps the user to distinguish between atoms that are part of the analyzed group or not.

[9] Color scheme taken from section *CPK Colours* here:
`http://acces.ens-lyon.fr/biotic/rastop/help/colour.htm`
[10] Color scheme taken from section *Amino Colours* here:
`http://acces.ens-lyon.fr/biotic/rastop/help/colour.htm`

- **Outline** A thin boundary is drawn around the atoms of the analyzed group. Rendered into the overlay framebuffer utilizing the stencil test. The outcome is displayed on top of the molecule and supports the user in localization of the group.
- **On Top Rendering** At composition of framebuffers, the opacity of the molecule framebuffer can be adjusted by the user. In addition, the group of analyzed atoms may be rendered on top of the molecule framebuffer. This is achieved by collecting the fragments – which belong to atoms of the group – during the standard molecule rendering and storing them into a custom texture image. The depth test for these fragments is performed manually and needs support by a semaphore image structure. This feature enables an user to stay focused on the group of chosen atoms, regardless which rendering mode is active.

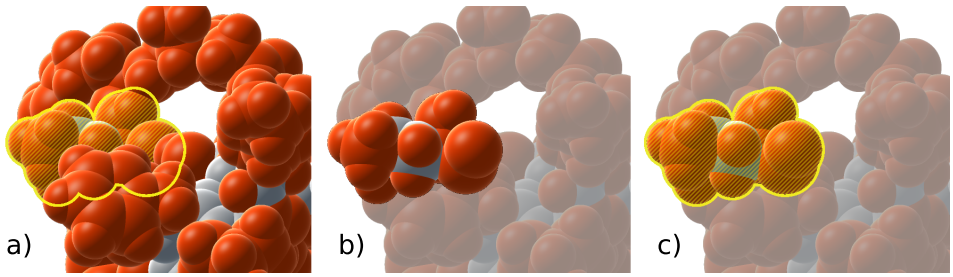


Figure 14: Pattern and outline of analyzed group is activated in screenshot **a)**. In screenshot **b)**, the molecule opacity is lowered and the group is rendered on top. The last screenshot **c)** depicts a combination of the three highlighting techniques.

4.2 Surface Extraction

Surface Atom Extraction based on Extended Spheres (SAEES) [14] has been realized by the original authors on CPU. Since each atom is independently processed, the algorithm can be adapted in a straightforward manner with a Compute Shader program on GPU. It utilizes one graphics processor for each atom and writes the results into global buffers. These can be accessed by CPU for further computations. For debugging purposes, a multi-threaded CPU version is also maintained, which shares the exact same logic and whose performance is also evaluated.

Setup

Before the algorithm can be executed, multiple buffers either for input or output data must be initialized on the graphics hardware.

Input Buffers set up and filled by CPU and read on GPU:

- **SSBO** *RadiiBuffer* with VdW radius given in Angstrom for each atom. Elements of the atoms are taken out of a PDB file and fed into an atom radii lookup table.
- **SSBO** *TrajectoryBuffer* with structures of {`float x, float y, float y`} for each atom in every step of the folding simulation. A custom struct with three `float` does seem to conform with the GLSL padding requirements, in contrast to a simple `vec3`. The simulation data is taken out of a XTC file. For addressing the position at a specific step, the atom count, step number and atom index is necessary[11].
- **uimageBuffer** *InputIndices* are the indices of the input atoms of the current algorithm execution. This enables one to use the same *RadiiBuffer* and *TrajectoryBuffer* for extraction of multiple layers.

Output Buffers set up by CPU and filled on GPU with results:

- **uimageBuffer** *InternalIndices* are the indices of the processed atoms that are classified as internal by the algorithm.
- **atomic_uint** *InternalCount* stores the count of atoms classified as internal. This atomic counter is also used to save internal indices in the image buffer at unique positions.
- **uimageBuffer** *SurfaceIndices* are the indices of the processed atoms that are classified as surface by the algorithm.
- **atomic_uint** *SurfaceCount* stores the count of atoms classified as surface. This atomic counter is also used to save surface indices in the image buffer at unique positions.

In addition, there are local variables and arrays in each Compute Shader thread to cache information about the cutting faces. Since the amount of reserved memory in a Compute Shader must be known at compilation time, maximum number of cutting faces has to be estimated. Since every neighbor creates potentially one cutting face, the maximum number of neighbors *neighborsMaxCount* is used for an overestimation. This value is hardcoded into the shader. For each cutting face, the center and the plane in which the face lies is locally stored for further processing:

- **int** *cuttingFaceCount* is initialized with zero and used to determine the next free index in *cuttingFaces* and later the total count of cached cutting faces.

[11] globalAtomIndex = (step * atomCount) + atomIndex

- **vec4** *cuttingFaces[neighborsMaxCount]* stores the plane equation of cutting faces with the normal vector in the x, y and z components and positive distance to the origin in w.
- **vec3** *cuttingFaceCenters[neighborsMaxCount]* stores the corresponding center of cutting faces.

Hence the arrays are already reserved in maximum size, there is no direct way to determine the index addressing the next slot to write to. To overcome this, the number of cutting faces is saved in an extra integer *cuttingFaceCount*. It is only valid within a thread and therefore locally stored.

The structures above imply redundant data, because the normal vector could be also calculated from x-, y-coordiantes and the normalized length of one. The distance can be calculated by dot product between center and normal vector too. However storing this values in the arrays keeps the code more structured and saves computation time, since both calculations are cheap but would have often been necessary in each Compute Shader execution. Especially due to their appearance in loops.

Further structures are necessary to cache the optimized list of cutting faces for each thread. Since the plane equations and centers of the cutting faces are not changed by that process, the structures in *cuttingFaces* and *cuttingFaceCenters* remain unchanged. Because some faces are discarded, it is sufficient to create an array of valid indices, called *cuttingFaceIndices*. As for the *cuttingFaces*, a simple counter for stored indices is necessary. In addition one indicator per cutting face is utilized while optimization. Their usage is defined later:

- **int** *cuttingFaceIndicators[neighborsMaxCount]* is initialized with value of one for each corresponding cutting face. A value of zero – which may be assigned during optimization – indicates a discard of the face.
- **int** *cuttingFaceIndicesCount* is the count of cutting faces surviving the optimization process and used to detect the next unique index in *cuttingFaceIndicators*.
- **int** *cuttingFaceIndices[neighborsMaxCount]* are the indices of the surviving faces. These indices can be used to address cutting faces in *cuttingFaces* and *cuttingFaceCenters*.

Execution

Execution is dispatched in groups of 64 threads, whereby the count of work groups is depending on the number of input indices addressing the atoms in the molecule. The optimal count of threads per work group has been determined by test runs like in Appendix 7.6 and the value may vary for

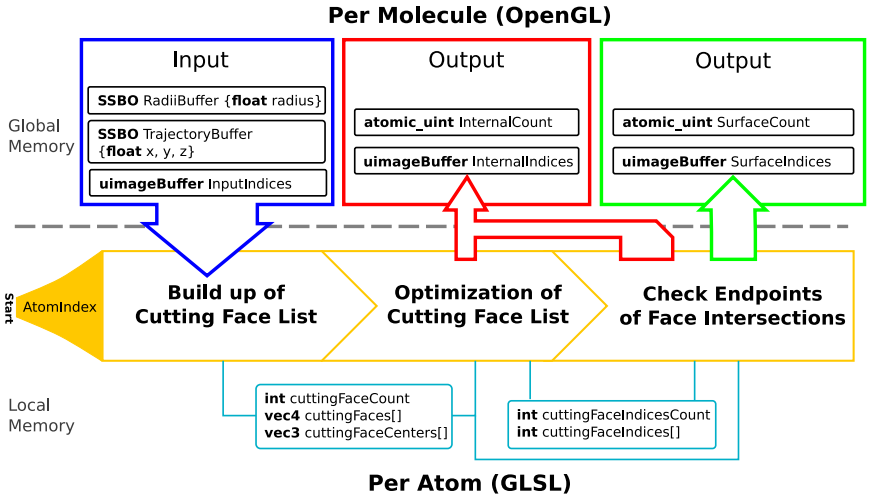


Figure 15: Implementation of SAEES as GLSL Compute Shader on a GPU.

different molecules and GPUs. For each atom, one Compute Shader thread is launched that can operate independently from all others. The threads are only synchronized in accessing the output image buffers by two atomic counters called *InternalCount* and *SurfaceCount*. The algorithm can be split into three parts, as indicated in Figure 15. The steps are displayed in the centered lane within the yellow border. At first, the list of cutting faces is built up. Then, this list is optimized by pairwise testing of cutting faces. This can already result in a classification as internal atom. Last step is to check all endpoints on the extended hull. These are generated by the intersection of two cutting faces and the extended hull. If at least one endpoint is not cut away by any other cutting face or no endpoint has been generated during the whole process, the atom is classified as surface atom. Otherwise it is classified as internal. These three steps are explained in detail within the next paragraphs.

Build up of Cutting Face List At first, each atom of the molecule is tested against the currently processed one. This is essential in order to determine which atom's extended hull is a candidate to have an intersection with the currently processed atom's extended hull. There can occur different spatial configurations of the atom pairs, as shown in Figure 16. For testing the

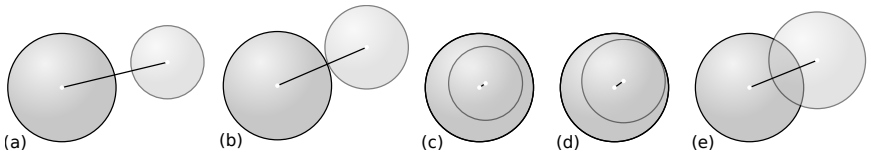


Figure 16: Configurations of two atoms' extended hull.

distance d between both atoms' centers, both extended radii r_0, r_1 and their sum r_{sum} is mandatory. The following items correspond to the letters assigned in Figure 16:

(a) $d > r_{sum}$ Both atoms are too far away and no intersection takes place.

(b) $d = r_{sum}$ The atoms' extended hulls touch each other in exactly one point. This case is omitted in this implementation of the algorithm, since it is very unlikely.

(c) $r_0 > r_1 + d$ or $r_1 > r_0 + d$ The currently processed atom's extended hull does completely cover the other's hull or the other's hull completely covers the processed one's. That neighbor is ignored in the first configuration. The second configuration directly classifies the processed atom as internal, since it is completely covered and there is no valid position for the probe to touch the atom.

(d) $r_0 = r_1 + d$ or $r_1 = r_0 + d$ Because the configuration is quite similar to the previous, the same behavior is applied.

(d) $d < r_sum$ There is an intersection which results in a circle that is stored as cutting face.

Only for the last configuration **(d)** the intersection between both is calculated like described in Appendix 7.1 and shown in Figure 17. For each intersection, both the plane equation of the plane including the intersection circle and the circle's center in world coordinates is saved. For that purpose the local arrays *cuttingFaces* and *cuttingFaceCenters* are filled at the next free index indicated by *cuttingFaceCount*, which is incremented afterwards. The plane's normal vector is defined by the connection vector between the atom's center that is currently processed and the neighboring atom's center. It can be interpreted as direction into which the extended hull is cut away by the neighbor's.

Optimization of Cutting Face List The next step creates a list of cutting face indices, which have survived an optimization process. This optimization does not only minimize the list of cutting faces but can also lead to an instant classification as internal atom. This is the case when two extended hulls cut away the atom's completely. The classification must be performed at this point, because no endpoints would be generated by the two cutting faces and the atom may be falsely classified as surface. During the optimization process no extra cutting faces are introduced or existing changed in parameters, why the original ones can be used further. Only the indices of these cutting faces that survive the optimization are stored in *cuttingFaceIndices*.

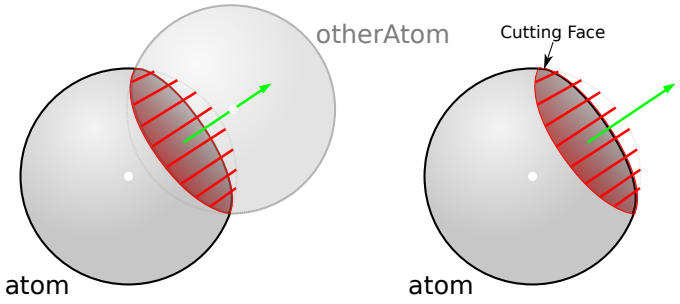


Figure 17: Atom intersects with other atom. The other atom peels away some of it's extended hull's surface, indicated by red color. The vector of connection of both centers is colored in green, which is interpreted as normal vector of the cutting face's plane.

During the optimization, an array of integers *cuttingFaceIndicators* is used to indicate whether the corresponding cutting face is not yet discarded. This indicator is initialized for of each cutting face with a value of one and set to zero during the optimization, if it is not necessary to consider the cutting face anymore.

The cutting faces are iterated pairwise to check their configuration, but only if both corresponding indicators are still equal one. Initially it has to be checked that the intersection line of both cutting faces does not intersect with the processed atom's extended hull. Otherwise the optimization can be skipped for that pair, since endpoints exist, which are evaluated in the last step. See Figure 18 **(a)**. The check is implemented by testing for parallelism of the two cutting faces, in which case no intersection would take place. If they are not parallel, the intersection line of both cutting faces is calculated as shown in Appendix 7.2. This intersection line is used to partly solve for endpoints with the algorithm for line-sphere intersection from Appendix 7.3. Since the pure existence of endpoints is sufficient to skip that iteration, only the value beneath the square root of the calculation is needs to be computed. If the value is higher or equal to zero, the optimization is skipped for that cutting face pair, because there exists at least one endpoint.

When the cutting faces do not cut within the atom's extended hull, there are three possible configurations as shown in Figure 18 **(b)**, **(c)** and **(d)**. To classify the occurrence, the connection between both cutting faces' center v and a point on the middle of that connection p is utilized:

(a) Faces produce endpoint(s) and are not discarded, as described above.

(b) The normal vectors of the faces point into the same direction. So one face is included by the other and can be discarded. To check which face can be discarded, it is tested in which positive half space of the face's plane the point p lies. The indicator of the discarded face is set to zero.

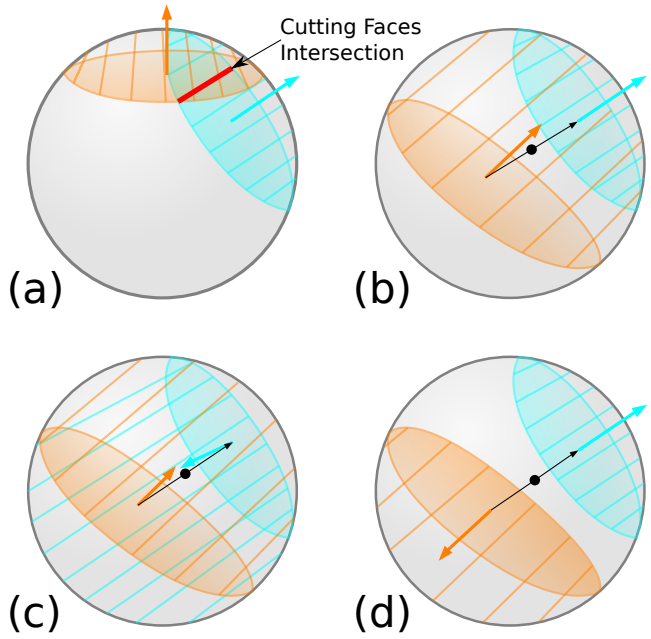


Figure 18: Possible configurations of two cutting faces.

(c) The normal vectors of the faces point in different directions. The pair of tested cutting faces does completely cut away the atom, if p lies in the negative half space of both faces. This classifies the tested atom immediately as internal and the algorithm terminates after saving the index of the atom to the output buffer for internal atom indices.

(d) The normal vectors of the faces point in different directions, but p is in positive half space of both planes. These faces do not influence each other and no change of indicators is necessary.

Last step in optimization is to collect the indices of the surviving cutting faces, which are recognizable by an indicator equal to one. These indices are written to *cuttingFaceIndices* and *cuttingFaceIndicesCount* is incremented in order to fill the array correctly. From now on these indices in *cuttingFaceIndices* are used to address the cached cutting faces.

Check Endpoints of Face Intersections Testing for endpoints concluded the classification. These are intersection positions of the intersection line of two cutting faces. This is implemented by a pairwise intersection of the optimized cutting faces as done in previous paragraph, too. The intersection line itself is checked for intersection with the atom's extended hull. This results in zero, one or two points, as shown in Appendix 7.3.

The generated endpoint(s) are tested for inclusion by any other cutting face than the two that created the endpoint(s). If no other cutting face

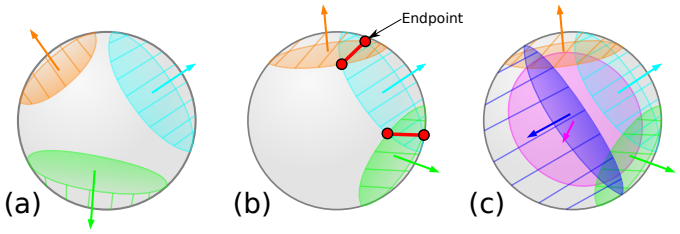


Figure 19: Last step of the surface extraction algorithm visualized. In **(c)** only cutting faces at the front are displayed. Since the atom is completely cut away, there should be also some in the back. They are omitted for an easier interpretation. In **(a)** no endpoint is generated and atom is classified as surface. The same classification is given in **(b)**, because at least one endpoint (here even all four) survives the inclusion test with the other faces. In **(c)** the atom is classified as internal because all endpoints are cut away.

includes the endpoint(s), the algorithm can terminate after classifying the atom as surface by writing its index to the corresponding buffer. If no endpoint has been generated by any cutting face pair, the atom can be classified as surface, too. It is internal elsewise.

Optimization

This section examines ideas for optimization of the presented SAEES implementation on graphics hardware. Multiple tactics are presented, which can potentially decrease computation time and memory consumption. Since the algorithm is rather complicated, it can be thought of many different approaches to improve the performance, but success is questionable or implementation complexity is high.

Cache In Different Memory Locations For classification of atoms the caching of cutting face information like plane equation and face center is necessary. There are multiple memory locations that can be used to store these values. Since the count of cutting faces depends on spatial structure of the molecule, radii of atoms and radius of the probe, the count of neighbors that intersect may be higher than 100 for some atoms. The local and shared memory usage of Compute Shader must be defined at their compilation time. Therefore an upper bound of for example 200 is used as allocation size for the arrays presented in 4.2. Cache in global memory like in a SSBO could be reserved with a combined estimation of a maximum number of all neighbors for every atom in the molecule at runtime. This was has been not done for testing purposes, since the advantage in performance is questionable and the consumption of global memory is not worth

noticing on modern hardware[12]. In the implementation of global memory cache, every Compute Shader thread has its private section of memory. It has the same size as the local variant does have:

- **Local** Cache is declared and defined in the beginning of the Compute Shader source code. It has been found no specification about how much memory is available for each thread. But simple tests indicate that there is a check at shader compilation, which fails if too much memory is tried to be allocated.
- **Shared** Put into the same shader source code location as local memory, but with `shared` prefix. Since OpenGL guarantees only 32KB[13], the local size for dispatching must be reduced to 6 threads per work group on available hardware. This has enormous negative impact on performance.
- **Global** Reserving memory in size of count of maximum neighbors times count of atoms in molecule results in a SSBO where every Compute Shader thread has its private section for caching.

Furthermore, the global variant could be used to eliminate the symmetric calculation of cutting faces in two Compute Shader threads by computing them only a single time. A mapping may be saved for two atoms to this single cutting face which is then used by both. It may be done in combination with the split into smaller computation units. Because that mapping means an additional overhead, it has not been implemented, yet. One can find benchmark results in Appendix 7.5.

More Caching There are even more values which could be cached. For example endpoints are indirectly computed at cutting face optimization in order to check whether the cutting face pair intersects within the extended hull of the atom. Because there could be up to two endpoints for each cutting face pair – whose count is upper bounded by maximum number of neighbors – the size for the cache would exaggerate. In addition there may be some lookup overhead. It was roughly tested in the CPU implementation with an `std::map`. The result has bad performance compared to the version with no caching of endpoints.

Neighborhood Search A fast neighborhood search algorithm implemented for OpenGL structures is available in the framework used as code basis for the developed application. The implemented method is inspired by a CUDA neighborhood search in *fluids v3*[14], written by Rama C. Hoetzlein. A grid

[12] For examples: 4 * 4 byte (vec4) * 200 neighbors * 1500 atoms = 4.8 MB

[13] `https://www.opengl.org/wiki/Compute_Shader`

[14] `http://fluids3.com`

around the molecule is defined and the atom centers are inserted. This is quite similar to the first part of the sample-based algorithm [5], presented in Related Work. After the grid is filled, the atom centers are sorted by their appearance in grid and written to another buffer. This improves the memory access at later usage in shaders, because the probability is higher that the hardware driver is able to cache data into local memory which is required by the computation. The fast neighborhood implementation could be utilized for speeding up the cutting face calculation in SAEES. Instead of checking all atoms of the molecule for neighboring the currently classified atom, only atoms within a certain range would be necessary to check. An integration has been attempted but some problems occurred.

Splitting Into Smaller Compute Shaders The three steps of the algorithm as indicated by structure of paragraphs in Section 4.2 and graphics in Figure 15 could be split into three Compute Shader dispatches. These can share a cache over global memory. In addition, the outer for-loops in that steps could also be replaced by shader dispatches, but some coherent global memory accesses would be necessary.

However publications like the Parallel Contour-Buildup algorithm [7] show the potential of splitting the surface extraction into more compact but numerous dispatches. They choose a complex approach, which does not generate one thread per atom but for each cutting face.

Loop Unrolling A popular approach to optimize performance for GPU processing is to manually unroll loops. This approach is not applicable for the loops used in the proposed Compute Shader, because the count of executions may vary from zero to the number of calculations necessary for pairwise comparison of the cutting faces.

Validation

The validation of the atom classification solely on a visual basis is not sufficient. Therefore a sample based validation algorithm is proposed. For every atom in the molecule an user defined number of random sample points[15] with uniform spherical distribution[16] is generated on all extended hulls. It is marked whether that sample has been created by an atom classified as internal or as surface by the surface extraction algorithm. Then all samples are checked to be included by any other atom's extended hull. If at least one sample point generated by an atom classified as internal is not included by another atom's extended hull, the surface extraction algorithm would have failed in terms of missing at least one surface atom.

[15] C++ standard library `std::rand` is used, which outputs pseudo random numbers.

[16] `http://mathworld.wolfram.com/SpherePointPicking.html`

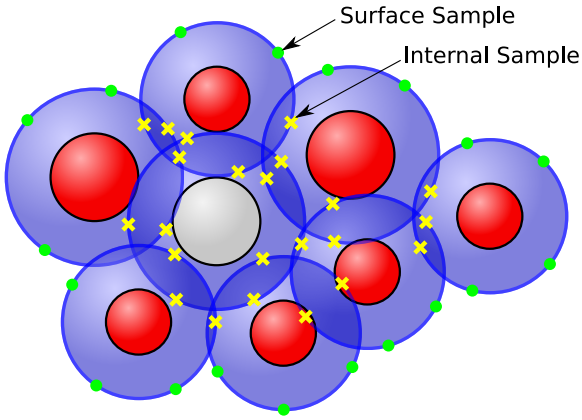


Figure 20: Validation of SAS extraction by using uniform sampling points on all atoms' extended hulls. Green dots are sampling points that are not included in any other hull. This should only happen for atoms that are classified as surface. Yellow crosses symbolize sampling points that are included by other hulls and therefore classified as internal. All sampling point originating from an atom classified as internal should be included to support the result of the algorithm, as happened for the gray colored atom.

The same idea can be applied for samples generated by classified surface atoms, too. When all generated sample points are included by other atoms' extended hull, it may indicate that this atom is not at the surface. In contrast to the sample points generated by internal atoms, this no proof for failure. It gives an indication that the algorithm may have failed. There is only a limited number of samples generated for each atom and the area on the surface may be very small for the given atom. The approach is applied in Section 5.1.

4.3 Ascension

Each atom's *Ascension* value is initialized at first time step of surface extraction. Either the initial value is set to an angle of 0° – if classified as internal – or 180° – else. Then a loop over the consecutive time steps executes the following algorithm for each atom to update the Ascension value v_i at the given step i. Input is the value from the previous step v_{i-1} and the classification $surface$ by the surface extraction algorithm:

$h_{up} := 180°/upToHotStepCount;$
$h_{down} := 180°/downToHotStepCount;$
$c_{up} := 180°/upToColdStepCount;$
$c_{down} := 180°/downToColdStepCount;$
if $surface = true$ **then** ▷ Atom classified as surface

if $v_{i-1} = 180°$ **then**
$v_i = 180°$
else if $v_{i-1} < 180°$ **then**
$v_i = min(180°, v_{i-1} + h_{up})$
else
$v_i = min(180°, v_{i-1} - h_{down})$
end if
else ▷ Atom classified as internal
if $v_{i-1} = 0°$ **then**
$v_i = 0°$
else if $v_{i-1} < 180°$ **then**
$v_i = min(0°, v_{i-1} - c_{down})$
else
$v_i = mod(min(360°, v_{i-1} + c_{up}), 360°)$
end if
end if

The result of coloring rise and descent of the atoms is visualized in Figure 21. For shading, an offset angle of 45° is added to yield blue color for cold atoms and yellow for hot ones. In addition, the original angle without offset is taken as argument for a cosinus function whose absolute value is extracted. Optionally, the displayed atom radius is multiplied with this value to highlight atoms that are currently rising to the surface or falling back into the inner of the molecular structure.

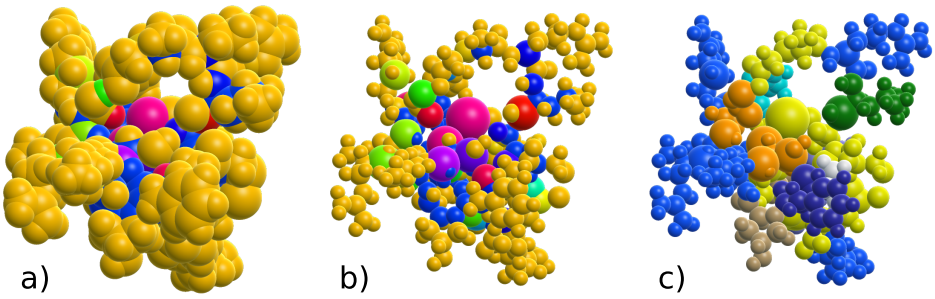


Figure 21: **a)** depicts a molecule rendered with VdW hull and colored with Ascension values. In **b)** the radius is multiplied with the current change in Ascension, additionally. The last screenshot **c)** shows the combination of radius adaption to activity and Amino Acid rendering mode.

4.4 Analysis

For further analysis uniformly distributed samples are generated on the extended hull of all atoms. For each sample the binary information is saved,

whether it is part of the SAS or classified as internal. Because the algorithm runs on GPU and the sample classification should be rendered as color of the samples, manual packing of bits is necessary. In GLSL, single bits cannot addressed in a buffer, since the smallest unit is a byte, which holds eight bits. Therefore a more complex approach for storing the sample classification is applied. There are two buffers allocated on the GPU. One contains the position of the samples relative to the atom they belong to. These values do not change over time. The other one stores a large number of `unsigned int`, which hold the classification for each time step per sample. It is calculated how may unsigned integers are necessary to hold the classification for one sample over the complete simulation time. The result is ceiled and multiplied with the count of samples covering the complete molecule. This number is used as size to allocate the classification buffer, which is then filled by a Compute Shader. Because only samples of surface atoms have a chance to be classified as surface, exclusively these samples that belong to a surface atom are considered in the calculation. All other bits remain zero, which indicates the classification of being internal. The classification by the Compute Shader is performed straight forward. If no other atom's extended hull includes the sample, it is classified as surface. The samples are finally used to approximate the values presented in Section 3.4.

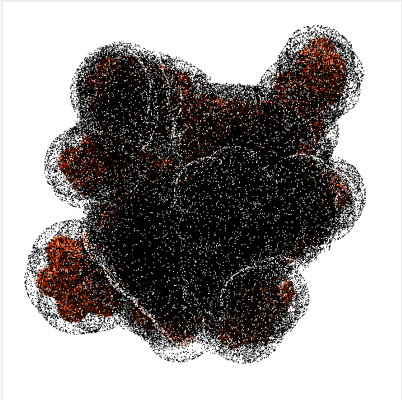


Figure 22: Visualization of hull samples, while atoms rendered with VdW radius. Black samples are classified as internal, white as surface.

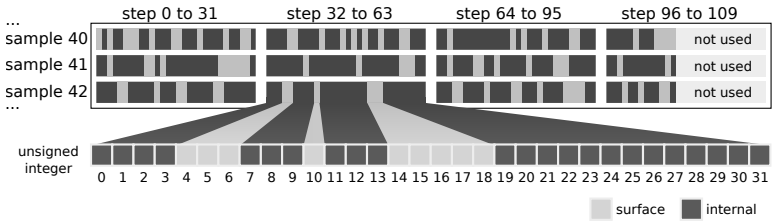


Figure 23: Visualization of buffer structure for samples 40 to 42, which are classified for the simulation time steps 0 to 109. Since the step count of 110 is no divisor of the 32 bit a `unsigned int` can hold, the last 18 bits of each row are unused.

Surface Area Approximation One usage for the hull samples is the approximation of the molecule's surface area or of a group of atoms. This

is achieved by taking the formula for calculating the surface of a complete sphere as starting point:

$$A = 4\pi r^2$$

The ratio between samples classified as being surface and classified as internal is computed and multiplied with the area of a complete sphere, which is defined by an atom's extended hull. By summing the result for all atoms of the molecule, the area of SAS is approximated.

Path For the analyzed group of atoms, a path is displayed, where each step takes the mean center of the group atoms as position. For simpler computation of partial lengths, the accumulated path length instead of the distances between the single steps is stored. The positions at the steps can be optionally smoothed over multiple steps for an easier perception of the path. For rendering a combination of lines between the steps and points at the steps is chosen. Future parts of the path are colored green, the past is colored in red.

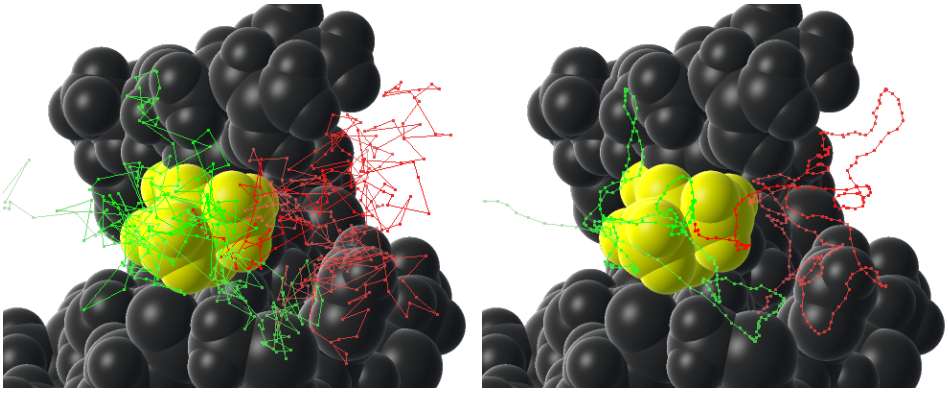


Figure 24: Left hand side raw and right hand side smoothed path of analyzed group of atoms.

5 Evaluation

As part of this thesis, an application named *SurfaceDynamicsVisualization* has been developed. All proposed features are implemented directly there or in on of the libraries that are linked to the application. In this section the algorithms' outcomes like surface extraction are compared to results by researchers of the area. Then, the proposed novel surface dynamics visualization and data acquisition methods are evaluated in cooperation with scientists in pharmaceutics at University of Bonn.

5.1 Technical Evaluation

The technical evaluation concentrates on the comparison with other publications in terms of correctness and performance of surface extraction and further data acquisition like surface area approximation.

Surface Extraction Validation

Approaches like the sample-based [4] or voxel-based [9] surface extraction struggle with the computation of the complete set of surface atoms containing neither false-positive nor false-negative classified atoms. The following table provides a comparison of the count of classified surface atoms. The implemented algorithm is titled by its original name, SAEES:

PDB id	# atoms	SAEES	Sample-based	Voxel-based
1A19	1,438	930	681	911
1O3Y	2,664	1,661	1,261	1,603
1QB5	3,750	2,104	1,482	2,009
1EAI	4,540	2,940	2,151	2,854

However there is no indication given by the authors, which atoms exactly are classified as surface and what is the ground truth. Therefore, a custom validation has been presented in Section 4.2 and is successfully applied for the molecules above by generating 1000 sample points per atom. Figure 25 provides an visual impression of the validation method.

Surface Extraction Performance

The time necessary for surface computation is compared to the sample-based approach [4] and the Parallel Contour-Buildup [7]. Researchers of the other presented methods have not published a performance benchmark of their implementation. Because the chosen publications do not share the same data sets for evaluation, some cells of the table are marked with *n.a.*

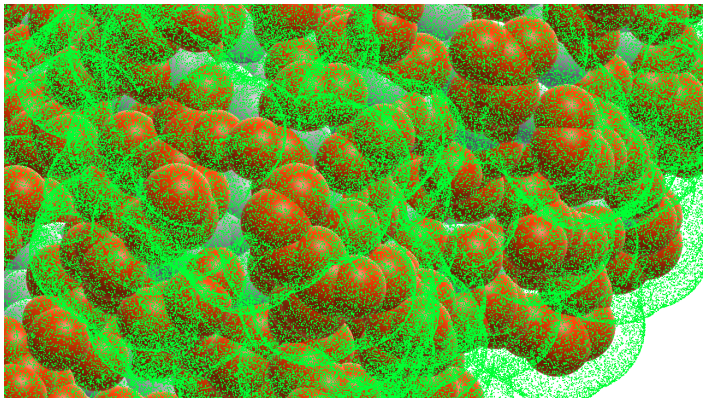


Figure 25: Validation with 5000 sample points for each atom of the molecule with PDB id 1A19. Only sample points that are not included in any atom's extended hull are rendered as green dot. No sample point created by an atom classified as internal by the SAEES algorithm implementation survives the inclusion test, as reported by the validation.

(not available). None of the authors linked to executable code, which could have been used for more exact comparisons. There are four columns of computation time per data set:

- **GPU** Implementation of SAEES on GPU as presented in Section 4.2. Running on benchmark environment 2.
- **CPU** Multi-threaded implementation of SAEES on CPU utilizing the eight virtual cores of benchmark environment 2.
- **Sample-based** The approach for SES extraction presented by Byungjoo, Ku-Jin and Joon-Kyung [5] has been also published as method for surface atom classification [4]. A voxel grid is employed for acceleration in neighborhood access. Their test environment consists of an Intel i5 4 x 2.8GHz CPU, 4.0GB RAM and a NVIDIA GeForce GTS 450 graphics card with 1.0GB VRAM. The algorithm is implemented in C++ and CUDA.
- **Parallel Contour-Buildup** The parallel version of Contour-Buildup algorithm [7] is implemented in CUDA and has been evaluated by the authors on an Intel i7 4 x 2.66GHz CPU, 6.0GB RAM and a NVIDIA GeForce GTX 580 with 1.5GB VRAM.

As indicated in Related Work, the Parallel Contour-Buildup implementation acts as upper limit of performance, while probably providing the exact same result in classification. However, real-time computation is not necessary for the application. The surface is calculated as preprocessing step over a chosen simulation time frame consisting of many steps. Every discussed

PDB id	# atoms	GPU	CPU	Sample-based	Parallel Contour-Buildup
1OGZ	944	16.58 ms	50.38 ms	n.a.	9.5 ms
1A19	1,438	17.35 ms	133.71 ms	7.37 ms	n.a.
1EAI	4,540	67.05 ms	1366.91 ms	28.45 ms	n.a.
1AF6	10,050	192.39 ms	6,647.9 ms	n.a.	36.1 ms

algorithm would demand some waiting time from the user. This does not affect the visualization of surface dynamics or other rendering. Therefore the presented multi-threaded CPU or GPU implementation is sufficient for the purposes of this thesis.

Surface Area Approximation

For further data acquisition, a sample-based approach as described in Section 4.4 is employed. One use case is the approximation of the molecule's surface area regarding the SAS. This can be compared to results in the publication by Can, Chen and Wang [1], in which they compare their LSMS algorithm against the established MSMS [12]. The column with results by the proposed method is named after the application's name SurfaceDynamicsVisualization. For the surface area approximation, 1000 samples on each atoms' extended hull have been generated. The areas of LSMS and MSMS are taken from [1]:

PDB id	SurfaceDynamicsVisualization	LSMS	MSMS
1ECA	7,159 Å^2	6,797 Å^2	6,959 Å^2
2ACT	9,200 Å^2	8,878 Å^2	11,484 Å^2
2CHA	19,630 Å^2	10,354 Å^2	10,607 Å^2
2LYZ	6,696 Å^2	6,471 Å^2	7,740 Å^2
2PTN	9,323 Å^2	8,957 Å^2	9,153 Å^2
5MBN	8,351 Å^2	7,946 Å^2	8,982 Å^2
8TLN	12,712 Å^2	12,005 Å^2	12,708 Å^2

The huge difference for 2CHA may be explained by some incorrect molecular data, although the file has been checked and downloaded twice. Otherwise the results are satisfying and support the suggested implementation.

5.2 Biomolecular Evaluation

For the evaluation of the novel visualization and data acquisition methods, our cooperation partners at University of Bonn were consulted. They are interested in the influence of disulfide bridges to the folding process of conotoxins. The disulfide bridges of conotoxins are systematically removed and

stability of the proteins is evaluated. It is hoped to find favorable structures for the five considered contoxins GIIIA, KIIIA, PIIIA, SIIIA and SmIIIA. PIIIA is taken as example for the following evaluation. The probe radius for SAS extraction is set to 2.8 Å in order to extract a higher count of layers. This is reasonable since the possible interaction with other molecules is not of interest but the internal activity of the protein is observed. At first, an active CYS residue is identified in Ascension visualization and behavior of interest is tracked. Then, the activity is evaluated using the Residue Surface-Proximity score. The CYS residues are of special interest since they are forming these disulfide bridges that are systematically removed during the experiment.

Ascension

Visualization of as- and descension of atoms may support scientists in detecting active regions at the surface of molecules. Residues rise to surface or descent into the inner of the molecular structure. This influences the binding behavior of the protein. The visualization has been applied for PIIIA and results are available in Figure 26.

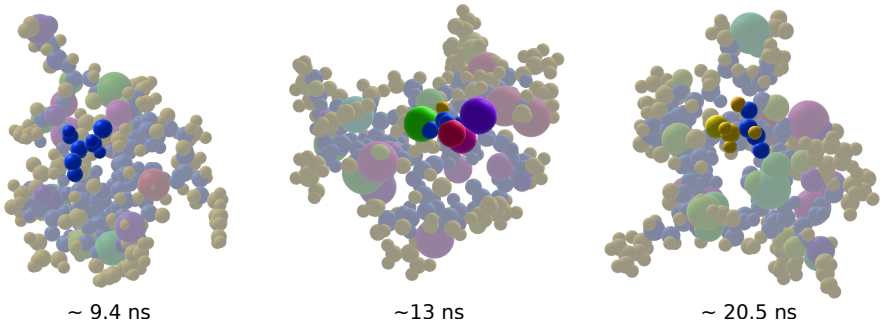


Figure 26: PIIIA of 100 ns folding simulation is depicted at different time steps. Ascension rendering mode in combination with group highlighting is used for generating the screenshots. Parameters of Ascension computation – as presented in 3.3 – are set to 100. The opaque group consists of the atoms in amino acid residue CYS4. At ~9.4 ns all atoms are inside the protein, as indicated by the blue coloring and small radius. The atoms start to rise to the surface during the simulation. This is visible at ~13 ns, as the radius of some atoms in the group is larger and colored differently from blue. The green one is moving from internal to surface, whereby the purple and red ones have been already at surface and may fall back into the internal structure of the protein. One atom is rendered small and colored yellow, which implies a stable position at the protein's surface. At ~20.5 ns, the atom positioning in relation to surface is stable. Majority of atoms is colored yellow and rendered with a small radius, indicating stability

This kind of visual data processing gives only an indication about the residue behavior. It is designed to support scientists in scanning vast amount of time steps while preserving structural overview. The detected change of CYS4 in surface proximity is evaluated in depth within the next section. The usage of Residue Surface-Proximity score yields insight to the movement of the residue within the molecular structure.

Residue Surface-Proximity

Application of the Residue Surface-Proximity – defined in Section 3.5 – may be used for data acquisition in a future publication of our cooperation partners. Figure 27 plots the average layer L_R for the PIIIA residues over folding simulation time into a heatmap.

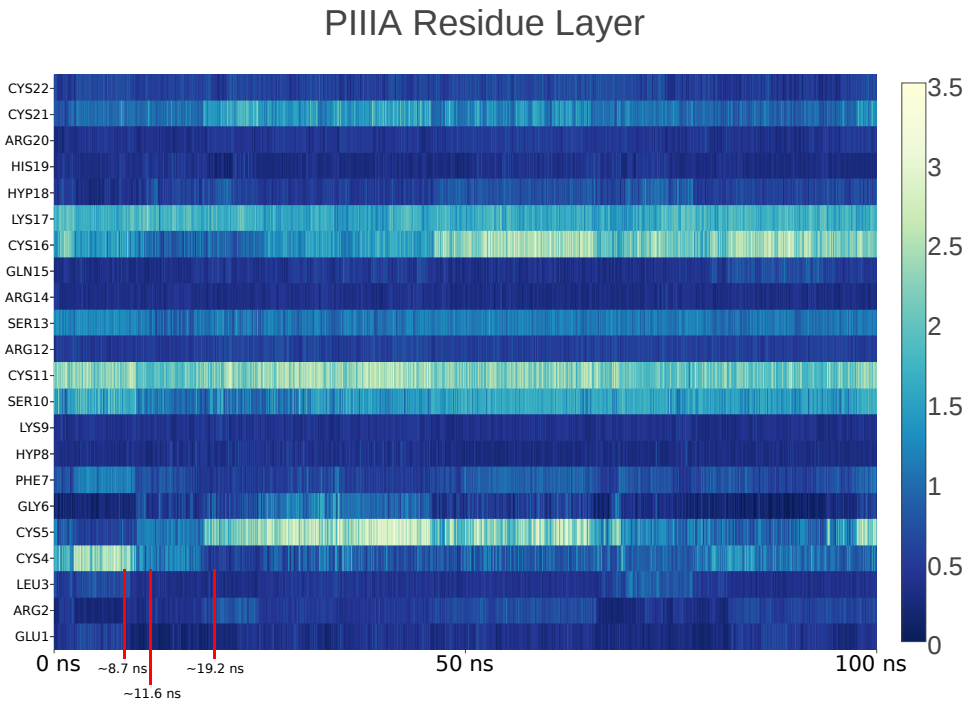


Figure 27: Heatmap renders L_R for the residues of PIIIA (see vertical axis) over a 100 ns simulation computed with 20k time steps (see horizontal axis). Dark blue refers to surface proximity and a bright color symbolized core proximity at the given time step. As one can easily percept, some residues are more likely to change their layer membership than others. For example, CYS4 is close to the core until ~10 ns in the simulation time frame. Afterwards its average layer is mostly found between zeroth and first layer of the protein. Red bars mark times at which CYS4 is depicted in Figure 28. Data extracted by Nils Lichtenberg and plotted with `https://plot.ly`.

The behavior deduced from the plot can be also inspected with the proposed visualization techniques, as shown in Figure 28. The rise of the CYS4 residue can be observed like in caption of Figure 27 announced.

Our cooperation partners are still interpreting the extracted data and have given some preliminary insights. It has become clear that the amino acid residues are most time not really changing their position within the protein but being flexible. The outer ones may move within a certain range and uncover residues that lie more closely to the core of the protein. When an residue at the surface moves and uncovers another residue below, it becomes surface in our definition. Although itself has not really moved. This interpretation of Residue Surface-Proximity describes not completely what has been proposed before, but our cooperation partners are positive about use cases in publications and future research.

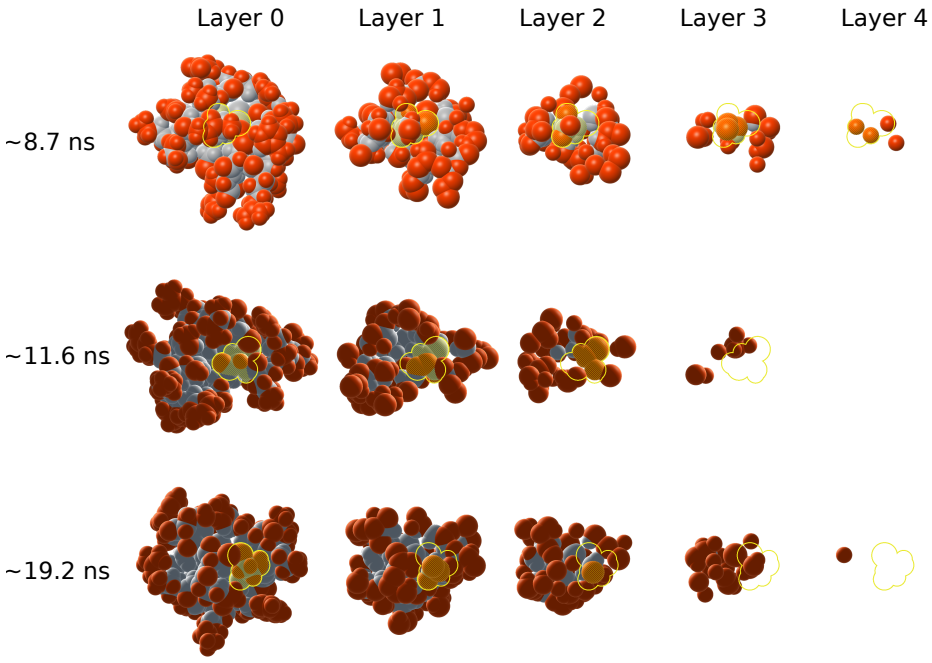


Figure 28: Observation of CYS4 residue in visualization of PIIIA conotoxin, which has been simulated over 100 ns. As indicated with the heatmap in Figure 27, the average layer L_R of CYS4 changes noticeable at about 10 ns within the simulation. At ~8.7 ns, the residue is part of layers one to four. In the next depicted time step at ~11.6 ns, CYS4 atoms reside in layers zero to two, which marks a high surface-proximity. The same layers are occupied after ~19.2 ns, however a higher count of atoms can be found in layer zero and one.

6 Conclusion

This thesis covers multiple approaches for supporting scientists in examining and analyzing molecular simulation data. Most publications and tools concentrate on the surface extraction and data acquisition for static molecule structures, but none focuses on the visualization of dynamic surface data. This is remarkable and it may be expected to see more publications about dynamic surface data analysis in the future.

The presented implementation for surface extraction can be improved in different aspects. A main problem may be the decision to use one thread for classification of one atom. As other publications have shown, distribution of workload to a higher count of threads yields better results in performance. In addition, some work can be avoided through the global scope of cached data. If something similar to Contour-Buildup algorithm is implemented, the additional information about circle arcs may be used to avoid the extra hull sampling performed here. Instead of using samples for approximation of surface area, one could calculate the exact surface area at the outcome of the surface extraction. Another application field would be the as- and descension visualization, which bases on the binary surface classification. One could use the approximated or even exact surface area to improve this visualization by replacing the binary input with a more accurate floating point value.

As indicated, surface dynamics of molecular structures is a rather unexplored area of research. One can propose novel ideas, which may help a lot in the interpretation of simulations' outcome. Visualizations have much potential to support scientists in identifying interesting aspects in the vast amount of raw data. This is a promising example of valuable interdisciplinary research where the two fields of computer graphics and pharmaceutics complement each other. Most software for molecular analysis is rather old and our cooperation partners were more than one time astonished about the possibilities of modern consumer hardware. In return they provided us with real world data and gave insightful interpretations.

7 Appendix

7.1 Sphere - Sphere Intersection

Following is inspired by an answer of *DMGregory* at *gamedev.stackexchange.com*[17]. Two spheres with the centers $\boldsymbol{c}_1$, respectively $\boldsymbol{c}_2$, and radii r_1, respectively r_2, intersect either not, in one point or in a circle with center $\boldsymbol{c}_i$ and radius r_i. This is depending on the distance d between both centers and the sum of both radii.

No Intersection If the sum of the radii is smaller than the distance or one of the radii is bigger than the distance, then no intersection is given. Further calculations are omitted.

Single Intersection Point When the sum of radii and centers' distance are equal, both spheres intersect in the point $p_i = \boldsymbol{c}_1 + (r_1/d) \cdot (\boldsymbol{c}_2 - \boldsymbol{c}_1)$. If one's radius plus distance is equal to the other's radius, the same formula can be used with $\boldsymbol{c}_1$ as the center of the larger circle and r_1 as the radius of the larger one.

Intersection Circle The center $\boldsymbol{c}_i$ of the intersection circle is expressed as a point on the connection $\boldsymbol{d}$ of both spheres' centers, with $\boldsymbol{d} = \boldsymbol{c}_2 - \boldsymbol{c}_1$ and $\boldsymbol{c}_1$ as initial point, see Figure 29. The variable $h \in [0..1]$ is used to

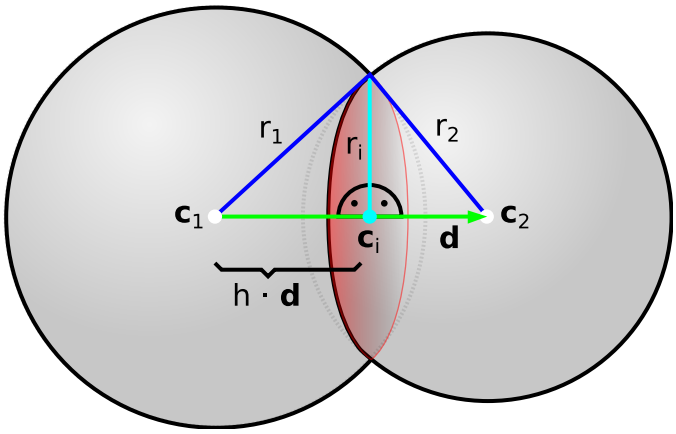


Figure 29: Two spheres intersect in one circle.

formulate a line equation $\boldsymbol{c}_i = \boldsymbol{c}_1 + h \cdot \boldsymbol{d}$ that aims to find the circle's center by finding the value of h. Since $h \cdot \boldsymbol{d}$, or symmetrically $-(1-h) \cdot \boldsymbol{d}$, and any vector from the center of the circle to its bounding represent the legs of

[17] http://gamedev.stackexchange.com/a/75775

a triangle with right angle, one can make use of Pythagoras' theorem and extract following initial equations, where $d = |\boldsymbol{d}|$:

$$r_1^2 = r_i^2 + (hd)^2 \Leftrightarrow r_i^2 = r_1^2 - (hd)^2$$
$$r_2^2 = r_i^2 + ((1-h)\cdot d)^2$$

The first equation can be put into the second one, which erases the unknown r_i^2:

$$\Rightarrow \quad r_2^2 = r_1^2 - (hd)^2 + ((1-h)\cdot d)^2$$
$$\Leftrightarrow \quad r_2^2 = r_1^2 - (hd)^2 + ((d-hd)^2$$

Now the second binomial formula is applied and the equation can be shortened:

$$\Leftrightarrow \quad r_2^2 = r_1^2 - h^2d^2 + d^2 - 2hd^2 + h^2d^2$$
$$\Leftrightarrow \quad r_2^2 = r_1^2 - d^2 - 2hd^2$$
$$\Leftrightarrow \quad 2hd^2 = -r_2^2 + r_1^2 + d^2$$
$$\Leftrightarrow \quad 2h = \frac{r_1^2 - r_2^2}{d^2} + 1$$
$$\Leftrightarrow \quad h = \frac{r_1^2 - r_2^2}{2d^2} + 0.5$$

The value of h is used to calculate the center $\boldsymbol{c_i}$ with the presented formula. It also necessary to get the radius r_i by putting it back into one of the initial two equations of the Pythagoras' theorem:

$$h = \frac{r_1^2 - r_2^2}{2d^2} + 0.5$$
$$\Leftrightarrow \quad h = \frac{r_1^2}{2d^2} - \frac{r_2^2}{2d^2} + 0.5$$
$$\Leftrightarrow \quad \frac{r_2^2}{2d^2} = \frac{r_1^2}{2d^2} + 0.5 - h$$
$$\Leftrightarrow \quad r_2^2 = r_1^2 + d^2 - 2d^2h$$

This equation for r_2^2 can be put into the second initial equation to erase it:

$$r_1^2 + d^2 - 2hd^2 = r_i^2 + ((1-h)\cdot d)^2$$
$$\Leftrightarrow \quad r_1^2 + d^2 - 2hd^2 = r_i^2 + (d-hd)^2$$
$$\Leftrightarrow \quad r_1^2 + d^2 - 2hd^2 = r_i^2 + d^2 - 2hd^2 + h^2d^2$$
$$\Leftrightarrow \quad r_1^2 = r_i^2 + h^2d^2$$
$$\Leftrightarrow \quad r_i = \sqrt{r_1^2 - h^2d^2}$$

The plane in which the circle lies is defined by the center point $\boldsymbol{c}_i$ and the unit vector $\boldsymbol{d}/|\boldsymbol{d}|$ as normal vector.

7.2 Plane - Plane Intersection

The initial idea for the algorithm is taken from the intersection of three planes shown in [2, page 305] by Ronald Goldman from University of Waterloo. Each plane is defined by a point P_k and an unit vector denoted with $\boldsymbol{V}_k$, where $k = 1, 2, 3$. The closed form to calculate the intersection point of all three planes is:

$$\begin{aligned} P_i = ((P_1 \cdot \boldsymbol{V}_1)(\boldsymbol{V}_2 \times \boldsymbol{V}_3) + (P_2 \cdot \boldsymbol{V}_2)(\boldsymbol{V}_3 \times \boldsymbol{V}_1) \\ + (P_3 \cdot \boldsymbol{V}_3)(\boldsymbol{V}_1 \times \boldsymbol{V}_2)) \ / \ det(\boldsymbol{V}_1 \ \boldsymbol{V}_2 \ \boldsymbol{V}_3) \end{aligned}$$

There is no intersection when two or more planes are parallel, which expresses in a denominator equal to zero.

On *stackoverflow.com* the user *ideasman42*[18] simplifies that calculation for finding the line of intersection between two planes that are not parallel. In contrast to Goldman, the general form of a plane with $ax+by+cz+d = 0$ is used to save some calculations and the normal vector given by $V_k = (a_k, b_k, c_k)$ does not need to be unit:

$$\begin{aligned} \boldsymbol{V}_3 &= \boldsymbol{V}_1 \times \boldsymbol{V}_2 \\ det &= |\boldsymbol{V}_3|^2 \\ P_i &= ((d_1(\boldsymbol{V}_3 \times \boldsymbol{V}_2)) + (d_2(\boldsymbol{V}_1 \times \boldsymbol{V}_3))) \ / \ det \end{aligned}$$

Again, if det is equal to zero, both planes are parallel and no intersection takes place.

7.3 Line - Sphere Intersection

Following is taken from an Wikipedia article[19] and enriched with further derivation steps. The dot sign "$\cdot$" denotes a dot product between two vectors within this section. A sphere is given:

$$\|\boldsymbol{x} - \boldsymbol{c}\|^2 = r^2$$

- $\boldsymbol{c}$ is the center point
- r is radius
- $\boldsymbol{x}$ is arbitrary point on sphere

And a line with o as initial point is defined:

$$\boldsymbol{x} = \boldsymbol{o} + d\boldsymbol{l}$$

- d is distance on line from initial point
- $\boldsymbol{o}$ is origin of line
- $\boldsymbol{l}$ is direction of line

[18] http://stackoverflow.com/a/32410473

[19] https://en.wikipedia.org/wiki/Line-sphere_intersection

- $\boldsymbol{x}$ is arbitrary point on line

These equations are put together to find solutions for d that satisfy both. So that one can put the solution for d back into the line equation:

$$
\begin{aligned}
& \|\boldsymbol{o} + d\boldsymbol{l} - \boldsymbol{c}\|^2 = r^2 \\
\Leftrightarrow \quad & (\boldsymbol{o} + d\boldsymbol{l} - \boldsymbol{c}) \cdot (\boldsymbol{o} + d\boldsymbol{l} - \boldsymbol{c}) = r^2 \\
\Leftrightarrow \quad & \boldsymbol{o} \cdot \boldsymbol{o} + \boldsymbol{o} \cdot d\boldsymbol{l} - \boldsymbol{o} \cdot \boldsymbol{c} + \boldsymbol{o} \cdot d\boldsymbol{l} + (d\boldsymbol{l}) \cdot (d\boldsymbol{l}) - \boldsymbol{c} \cdot d\boldsymbol{l} - \boldsymbol{o} \cdot \boldsymbol{c} - \boldsymbol{c} \cdot d\boldsymbol{l} + \boldsymbol{c} \cdot \boldsymbol{c} = r^2 \\
\Leftrightarrow \quad & (d\boldsymbol{l}) \cdot (d\boldsymbol{l}) + 2(\boldsymbol{o} \cdot d\boldsymbol{l}) - 2(\boldsymbol{c} \cdot d\boldsymbol{l}) + \boldsymbol{o} \cdot \boldsymbol{o} - 2(\boldsymbol{o} \cdot \boldsymbol{c}) + \boldsymbol{c} \cdot \boldsymbol{c} = r^2 \\
\Leftrightarrow \quad & d^2(\boldsymbol{l} \cdot \boldsymbol{l}) + 2d(\boldsymbol{l} \cdot (\boldsymbol{o} - \boldsymbol{c})) + (\boldsymbol{o} - \boldsymbol{c}) \cdot (\boldsymbol{o} - \boldsymbol{c}) = r^2 \\
\Leftrightarrow \quad & d^2(\boldsymbol{l} \cdot \boldsymbol{l}) + 2d(\boldsymbol{l} \cdot (\boldsymbol{o} - \boldsymbol{c})) + (\boldsymbol{o} - \boldsymbol{c}) \cdot (\boldsymbol{o} - \boldsymbol{c}) - r^2 = 0
\end{aligned}
$$

The last equation forms a quadratic formula of the appearance $ad^2+bd+c = 0$, with d as parameter and where:

- $a = \boldsymbol{l} \cdot \boldsymbol{l} = \|\boldsymbol{l}\|^2$
- $b = 2(\boldsymbol{l} \cdot (\boldsymbol{o} - \boldsymbol{c}))$
- $c = (\boldsymbol{o} - \boldsymbol{c}) \cdot (\boldsymbol{o} - \boldsymbol{c}) - r^2 = \|\boldsymbol{o} - \boldsymbol{c}\|^2 - r^2$

Thus, d can be solved with:

$$
\begin{aligned}
& d = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \\
\Leftrightarrow \quad & d = \frac{-2(\boldsymbol{l} \cdot (\boldsymbol{o} - \boldsymbol{c})) \pm \sqrt{4(\boldsymbol{l} \cdot (\boldsymbol{o} - \boldsymbol{c}))^2 - 4\|\boldsymbol{l}\|^2(\|\boldsymbol{o} - \boldsymbol{c}\|^2 - r^2)}}{2\|\boldsymbol{l}\|^2} \\
\Leftrightarrow \quad & d = \frac{-(\boldsymbol{l} \cdot (\boldsymbol{o} - \boldsymbol{c})) \pm \sqrt{(\boldsymbol{l} \cdot (\boldsymbol{o} - \boldsymbol{c}))^2 - \|\boldsymbol{l}\|^2(\|\boldsymbol{o} - \boldsymbol{c}\|^2 - r^2)}}{\|\boldsymbol{l}\|^2}
\end{aligned}
$$

Because the direction of the line $\boldsymbol{l}$ is an unit vector, $\|\boldsymbol{l}\|^2 = 1$. This simplifies the equation for d:

$$d = -(\boldsymbol{l} \cdot (\boldsymbol{o} - \boldsymbol{c})) \pm \sqrt{(\boldsymbol{l} \cdot (\boldsymbol{o} - \boldsymbol{c}))^2 - \|\boldsymbol{o} - \boldsymbol{c}\|^2 + r^2}$$

Three different cases can occur through the nature of the equation for d.

No Intersection When the result below the square root is less than zero, no solution exists and the line does not cut the sphere.

Single Intersection Point A result under the square root equal to zero occurs for a single intersection point.

Two Intersection Points Because the result beneath the square root is positive, there exist two solutions for d. This indicates two intersection points of line and sphere.

7.4 Radius of Gyration

Stability of a molecule folding can be measured by the *Radius of Gyration*. It describes the distribution of particles around the center of the molecule or an axis. In the case of a center $\mathbf{r}_{mean}$, the position of all particles is averaged. A sum over the squared distances between $\mathbf{r}_{mean}$ and the particles' position $\mathbf{r}_k$ is normalized by particle count N. This results in the squared Radius of Gyration R_g^2:

$$R_g^2 = \frac{1}{N}\sum_{k=1}^{N}|\mathbf{r}_k - \mathbf{r}_{mean}|^2$$

Our cooperation partners at University of Bonn use Gromacs[20] for extraction of R_g over simulation time for different conotoxins, including mass weighting. It is expected to correlate with the surface amount and surface area calculated with the proposed implementation. Since the Radius of Gyration is taken as indicator for stability and compactness of the molecule, a lower value should correspond to a lower value of surface area. The more compact a molecule becomes, the more similar it gets to the appearance of a sphere, which has minimal area in respect to content. So the Radius of Gyration is correlated both with the approximated surface amount and surface area proposed in Section 3.4. The results for some conotoxin foldings provided by our cooperation partners support the expectation:

Conotoxin	KIIIA	PIIIA	SIIIA
Correlation Coefficient Surface Amount	0.58	0.59	0.66
Correlation Coefficient Surface Area	0.6	0.59	0.67

7.5 SAEES: Memory Location Benchmark

Cache for calculating SAS with Compute Shader located in different memory locations. Benchmark for computation time of SAS was recorded for molecule with PDB structure id 1A19, which consists of 1438 atoms. Benchmark Environment 1 has been used.

[20] `http://www.gromacs.org`

	Local Cache	Shared Cache	Global Cache
1. Run	13.7786 ms	68.0221 ms	24.4276 ms
2. Run	13.7843 ms	68.0331 ms	24.3772 ms
3. Run	13.8041 ms	68.062 ms	24.4686 ms
4. Run	13.7819 ms	68.0269 ms	24.3744 ms
5. Run	13.8108 ms	68.0875 ms	24.4202 ms
Average	13.79194 ms	68.04632 ms	24.4136 ms

7.6 SAEES: Local Size Benchmark

Benchmark for determining optimal count of threads per work group for surface computation. Molecule with PDB structure id 1A19 is input of the algorithm. Benchmark Environment 1 has been used.

	Local Size = 16	Local Size = 32	Local Size = 64	Local Size = 128
1. Run	25.604 ms	20.1609 ms	13.8161 ms	14.0726 ms
2. Run	25.5797 ms	20.1651 ms	13.789 ms	14.7402 ms
3. Run	25.62464 ms	20.1981 ms	13.8175 ms	14.0763 ms
4. Run	25.5896 ms	20.1696 ms	13.7876 ms	14.0899 ms
5. Run	25.5973 ms	20.1632 ms	13.8245 ms	14.0756 ms
Average	25.599048 ms	20.17138 ms	13.80694 ms	14.21092 ms

7.7 Benchmark Environments

1. **Environment 1** ArchLinux, Intel i7 4712MQ 4 x 2.3GHz, 16GB DDR3 RAM, NVIDIA GeForce 860m 2GB VRAM, NVIDIA Driver 367.27
2. **Environment 2** Ubuntu 16.04, Intel i7 4712MQ 4 x 2.3GHz, 16GB DDR3 RAM, NVIDIA GeForce 860m 2GB VRAM, NVIDIA Driver 367.44

7.8 Links

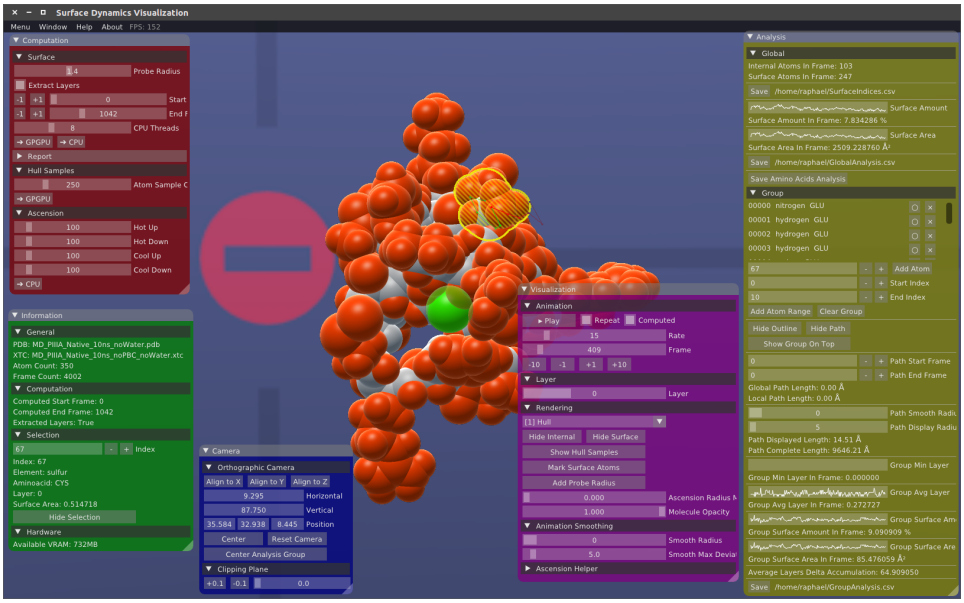
Content of all URLs has been checked at 7th October 2016.

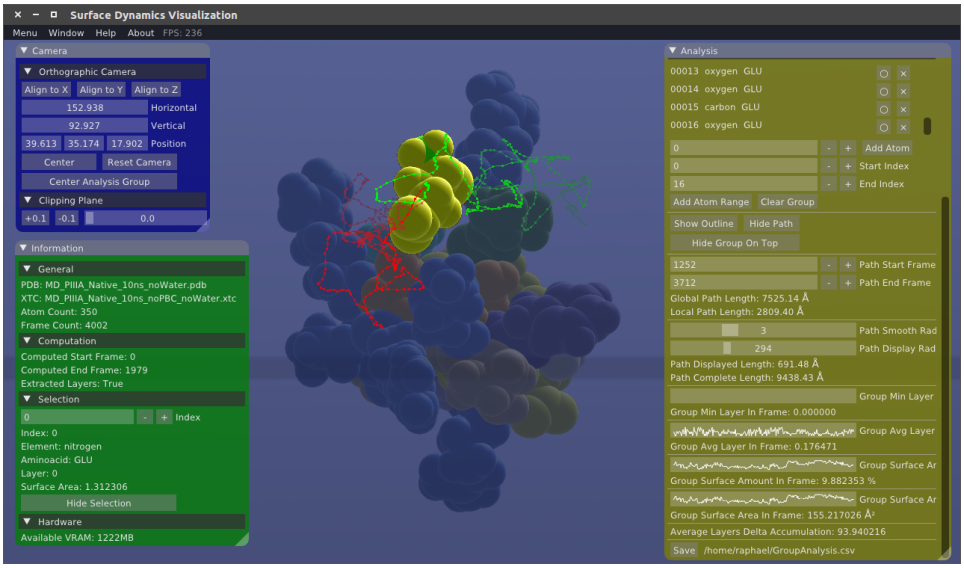
7.9 Data Sources

Simulated data as PDB and XTC has been provided by our cooperation partners at University of Bonn. Static molecule structures in PDB file format are obtained from `http://www.rcsb.org`.

7.10 Impressions

Uncommented impressions of the application SurfaceDynamicsVisualization, which has been developed as part of this thesis.





References

[1] Tolga Can, Chao-I Chen, and Yuan-Fang Wang. Efficient molecular surface generation using level-set methods. *Journal of Molecular Graphics and Modelling*, 25(4):442–454, 2006.

[2] Andrew S. Glassner. *Graphics Gems.* Academic Press, Inc., Orlando, FL, USA, 1990.

[3] Naga Bhushana Rao Karampudi and Ranjit Prasad Bahadur. Layers: A molecular surface peeling algorithm and its applications to analyze protein structures. *Scientific Reports*, 5, 2015.

[4] Byungjoo Kim, Ku-Jin Kim, Ji-Hoon Choi, Nakhoon Baek, Joon-Kyung Seong, and Yoo-Joo Choi. Finding surface atoms of a protein molecule on a gpu. 2011.

[5] Byungjoo Kim, Ku-Jin Kim, and Joon-Kyung Seong. Gpu accelerated molecular surface computing. *Applied Mathematics & Information Sciences*, 6:185–194, 2012.

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

[7] Michael Krone, Sebastian Grottel, and Thomas Ertl. Parallel contour-buildup algorithm for the molecular surface. pages 17–22, 2011.

[8] Byungkook Lee and Frederic M. Richards. The interpretation of protein structures: estimation of static accessibility. *J Mol Biol*, 55(3):379–400, 1971.

[9] Ling Wei Lee and Andrzej Bargiela. Protein surface atoms extraction: Voxels as an investigative tool. 2012.

[10] Thomas Luft, Carsten Colditz, and Oliver Deussen. Image enhancement by unsharp masking the depth buffer. *ACM Transactions on Graphics*, 25(3):1206–1213, 2006.

[11] Frederic M. Richards. Areas, volumes, packing, and protein structure. *Annual Review of Biophysics and Bioengineering.*, 6(1):151–176, 1977.

[12] Michel F. Sanner, Arthur J. Olson, and Jean-Claude Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996.

[13] Marco Tarini, Paolo Cignoni, and Claudio Montani. Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1237–1244, 2006.

[14] Jingqiao Zhang and Zhe Shi. Triangulation of molecular surfaces based on extracting surface atoms. *Computers & Graphics*, 38:291–299, 2014.

List of Figures

Glossary

Angstrom [Å] Unit of length equal to 10^{-10} m or 0.1 nm. 10, 23

atomic_uint Atomic counter in GLSL based on an unsigned integer structure. 23

Compute Shader Input of Compute Shader stage of OpenGL. 22–25, 29–31, 34, 47, 52

CPU Central processing unit. 17, 22, 23, 30, 37, 38

GLSL OpenGL Shading Language. 18, 23, 25, 34, 52

GPU Graphics processing unit. 8, 9, 17, 22, 23, 25, 31, 34, 37, 38, 52

HSV Hue Saturation Value. 13

PDB Human readable format for static molecular data. 23, 36–38, 47, 48, 53

SAEES Surface Atom Extraction based on Extended Spheres. 8, 10–12, 15, 21, 22, 25, 29, 31, 36, 37, 51–53

SAS Solvent Accesible Surface. 3–7, 10–12, 14–16, 18, 19, 34, 35, 38, 39, 47, 51, 52

SES Solvent Excluded Surface. 1, 3–9, 37, 51

SSBO Shader Storage Buffer Object is a buffer object that stores arbitray data on the graphics card memory. 17, 23, 29, 30

uimageBuffer Image buffer on graphics card memory with arbitrary unsigned data type. 23

VdW Van der Waals force. 2, 3, 5, 9, 10, 18, 23, 33, 34, 51, 52

XTC Portable format for molecule trajectories, used by Gromacs. 23, 48