# Efficient GPU computation of large protein Solvent-Excluded Surface

Cyprien Plateau-Holleville, Maxime Maria, Stéphane Mérillou, Matthieu Montes

# Efficient GPU computation of large protein Solvent-Excluded Surface

Cyprien Plateau–Holleville, Maxime Maria, Stéphane Mérillou, Matthieu Montes

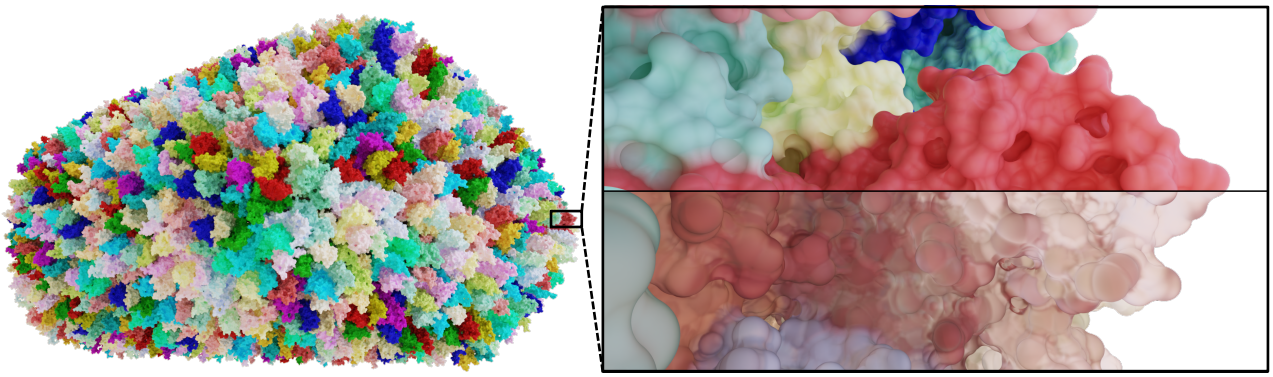

Fig. 1: Path-traced images of the entire HIV-1 capsid's (PDB: 3J3Q) Solvent-Excluded Surface. It is composed of more than 2.4M atoms and its surface is computed in 700ms on an NVIDIA RTX 2080 with our method. As shown in the right image, the surface is fully defined and can be rendered with a transmissive material giving an overview of its inner part.

**Abstract**—The Solvent-Excluded Surface (SES) is an essential representation of molecules which is massively used in molecular modeling and drug discovery since it represents the interacting surface between molecules. Based on its properties, it supports the visualization of both large scale shapes and details of molecules. While several methods targeted its computation, the ability to process large molecular structures to address the introduction of big complex analysis while leveraging the massively parallel architecture of GPUs has remained a challenge. This is mostly caused by the need for consequent memory allocation or by the complexity of the parallelization of its processing. In this paper, we leverage the last theoretical advances made for the depiction of the SES to provide fast analytical computation with low impact on memory. We show that our method is able to compute the complete surface while handling large molecular complexes with competitive computation time costs compared to previous works.

**Index Terms**—Scientific visualization, Massively parallel algorithms.

## 1 Introduction

Computer science, and especially computer graphics, is widely used by computational biochemists to support molecular modeling, drug discovery and design applications. From the development of enhanced shading to the implementation of hardware accelerated computation tools passing by specialized user interfaces, computer graphics research contributed many times to the improvements of molecular visualization software [1].

With the recent developments in molecular simulation and experimental structure resolution techniques, molecular datasets now include larger and more complex structures including molecular motion. Their studies are thus challenging the available tools in terms of memory, performance, and visualization clarity. Even if recent works were able to leverage the repetitive patterns in complex synthetic systems to provide visualization from molecular to cellular structures and thus demonstrate the ability to handle very large amounts of data [2], [3], they are not compatible with motion and molecular dynamics simulation data. Indeed, motion induces structural conformational changes in the molecular scene which rely on atom-level forces prohibiting the use of procedural generation or instancing. Additionally, recent developments in HPC allowed the generation of massive simulations at the atomic scale [4] that would benefit from appropriate illustration methods.

Scientific visualization has been oriented toward the development of tools supporting interactive analysis. While giving valid and insightful visualization is a mandatory feature of scientific software, the addition of high-quality illustration techniques supports both the understanding and the popularization of scientific knowledge [5], [6]. However, the adoption of the last computer graphics strategies and the creation of dedicated ones to handle the targeted domain's concepts are required to deliver adequate methods.

Richards [7] introduced the Solvent-Accessible Surface (SAS) and the Solvent-Excluded Surface (SES), both built from the atomic


- *Cyprien Plateau–Holleville, Maxime Maria and Stéphane Mérillou are with the XLIM Laboratory, UMR CNRS 7252, University of Limoges, France, E-mail: cyprien.plateauholleville@unilim.fr, maxime.maria@unilim.fr, stephane.merillou@unilim.fr*
- *Matthieu Montes is with the GBCM - Laboratoire Génomique, Bioinformatique et Chimie Moléculaire, EA7528, Conservatoire National des Arts et Métiers, Hésam Université, France and Institut Universitaire de France (IUF). E-mail: matthieu.montes@cnam.fr*

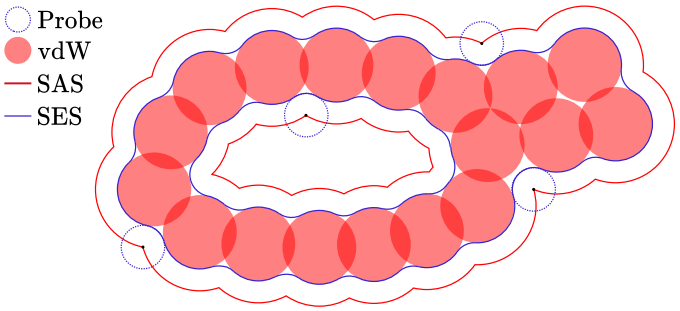


Fig. 2: The different molecular surfaces introduced by Richards [7]. Both the SAS and the SES describe the probe's interaction with the vdW surface and give insights on its interactions with the protein.

structure of a molecule using a spherical probe representing the solvent. As illustrated in fig. 2, the spherical probe is rolling over the van der Waals (vdW) surface and describes the SAS as the path of its center, an inflated version of the vdW surface, and the SES as its reachable frontier. While the SAS is especially studied for its area giving hydrophobia insights [8], the SES is the interacting surface in molecular interactions which guides the analysis of molecular complexes (protein-small molecule, protein-protein, etc.). For instance, it can support the identification of cavities based on the extracted geometry [9].

Several methods showed the ability for fast computation and visualization of the analytic exterior SES. However, the most used softwares [10]–[12] only offer triangulated approximation of the surface which level of detail is strongly linked to its memory consumption. This may be explained by the complexity to handle large structures and the complete surface, especially on applications targeting GPU computation and a wide variety of hardware. Indeed, the embedding of the geometry in the restricted GPU memory is a challenge strengthened by the complexity of its construction. Even if recent works [13] have been conducted to allow the handling of large proteins on graphic hardware, this was often at the cost of additional computation time and without the complete surface acquisition. This is not only limiting the support of large structures on dedicated software, but also the availability of these methods.

This paper presents a method for GPU computation of the SES with reduced memory consumption, without trade-offs between surface quality and processing time. Our contributions are:

- Based on the last SES depiction [8], we provide a dedicated data structure targeting a compute-over-store pattern for both memory and computation efficiency.
- We tackle one of the main memory consumption issues in previous works by leveraging a classification scheme which is strongly reducing the runtime memory footprint.
- To overcome the additional computation induced by memory footprint reduction features, we propose significant improvements to the parallelism of SES processing steps that we identified as important computation bottlenecks.
- We show that thanks to our development, the computation of the complete surface can be made in competitive computation time and without strong memory footprint which was not possible before.

To our knowledge, our method is the first to be able of fast GPU computation of large molecular structures complete surface. The data produced could thus be employed for various processes on the GPU such as area and volume computation, geometry processing, or illustration. We study its use for illustrative rendering benefiting from the quality and clarity of the generated surface, as illustrated in fig. 10, and decreased surface acquisition times. Finally, we show that the last developments made in the analytical depiction of the SES [8] suits especially well the rendering of this surface thanks to its implicit singularity handling.

In the following sections, we give the previous works on SES computation which are followed by the surface geometric definition. We then present the key steps of our algorithm and its GPU-oriented optimizations. Finally, we provide an analysis of the pipeline performance, its limitations and conclude this paper.

## 2 Related works

Since the first studies made by Richards [7] and Connolly [14], many works improved the SES computation under different orientations. To provide a reliable presentation of the related works, we propose to group these methods by their goals. We first present the several works providing fundamental approach to the depiction of the surface which are followed by those focusing on its fast approximating and analytical generation. For more information regarding visualization of biomolecular data, we orient the reader toward previous wider studies [15].

**Fundamental works and analytic depiction.** Based on Richards [7] and Connolly [14]'s works, several algorithms have been developed mostly relying on the links between the SES and the SAS. Reduced Surface [16] is a sequential algorithm producing an $\alpha$-shape [17] structural representation which has been widely used for its robustness and availability in the software MSMS [16]. It is based on an iterative computation of the structure validating and appending part of it. It has then been leveraged to allow rebuilding from previous computation [18]. The same year, another structural algorithm, Contour-Buildup [19], has been presented. It relies on SAS contours construction by modifying incrementally its arcs based on intersection points. In Connolly [14]'s analysis, the processing of the surface requires to handle singularities. Quan & Stamm [8] presented an analytic Signed Distance Function (SDF) representation of the surface, implicitly handling singularities for a more precise calculation. This representation has then been leveraged to offer accurate meshing of the surface [20].

**Approximating applications.** While several fundamental works have presented analytical studies of the surface, the need for fast visualization led to the development of approximating and/or discrete space methods mostly based on a signed distance field. The latter can be built by classifying every voxel depending on their belonging to the vdW surface or the SAS. It allows the introduction of complex surfaces such as the Ligand-Excluded Surface [21], which takes into account the geometry of the ligand for the surface approximation. Hermosilla et al. [22] presented an interactively refining method based on the ray-marching of a grid, which has then been leveraged by Martinez et al. [23] to provide a stand-alone library for SES mesh GPU computation. Parulek et al. [24] presented an SDF representation of the SES, which can directly be ray-casted but suffered from numerical errors. Bruckner [25] proposed a Gaussian surface-based approach that enables on-the-fly computation for real-time visualization of large molecular complex surface thanks to its simpler structure. However, it does not offer the SES features and insights.

**Analytic exterior molecular surface applications.** Analytic computation of the surface has been leveraged several times to provide an accurate representation of the exterior molecular surface

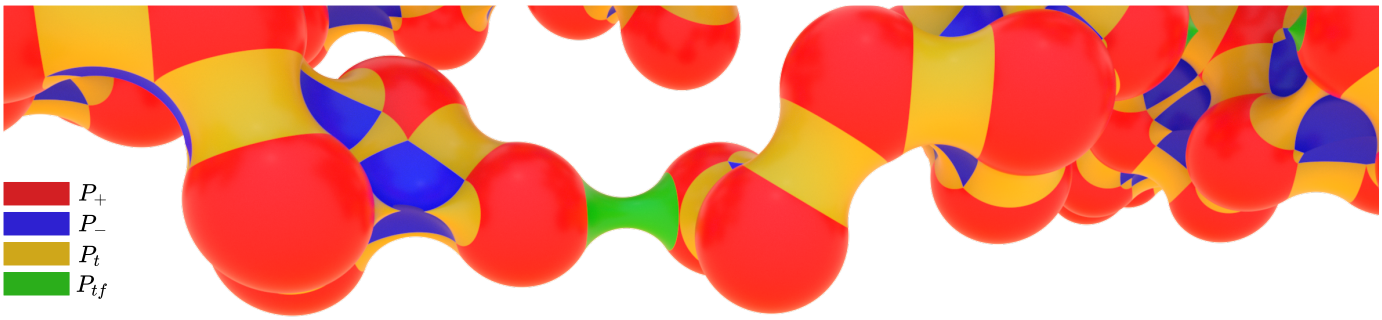


Fig. 3: Rendering of the SES of a molecule (PDB: 3DIK) with emphasized patches. The different types of patches, spherical convex $P_+$, spherical concave $P_-$, toroidal segment $P_t$ and complete circle $P_{tf}$ compose the complete SES and are defined in section 3.

after its structural computation. This is achieved by hiding unwanted geometry parts to the user within the surface. This is made for faster rendering but is thus not compatible with the visualization of the inside of the surface. Krone et al. [26] proposed a Reduced-Surface [16] based method providing interactive visualization by ray-casting. This work has then been improved with a GPU implementation of Reduced-Surface [27] and Contour-Buildup [28], allowing the parallelization of both algorithms. However, the first method suffers from artifacts due to its view-dependent nature while the second requires consequent memory allocation, limiting its use for large structures. Schäfer et al. [13] demonstrated a reduced allocation scheme at the cost of a slower computation.

**Analytic complete surface applications.** While the exterior molecular surface computation can be sufficient for some applications, users can need the complete surface to obtain more information such as tunnels and pocket visualization. For this purpose, Kauker et al. [29] presented a framework allowing transparent visualization by using Reduced Surface's primitives and cutting planes. This allowed to provide a rendering of the complete exterior molecular surface but not its inner part which can remain inside the volume. Jurcik et al. [30] proposed a method relying on a GPU implementation of Contour-Buildup [19], ray-casting, and cutting geometries which suffered from numerical errors and difficulties to handle large structures. Manak et al. [31] proposed a method based on Apollonius diagram, the Voronoï diagram of spheres and Euclidean distance, allowing fast probe radius update but requiring heavy pre-computation time. Finally, Rau et al. [32] presented an improved CPU ray-tracing method based on Contour-Buildup handling large complex at the cost of an order of magnitude longer computation times compared to the original GPU implementation.

The fast computation of the complete SES remains a challenge. We identified that the main difficulties are closely related to the amount of memory that can be needed to store the data structure, and the complexity to parallelize its exploration. In this paper, we propose to tackle both issues based on the last theoretical and technical advances made in SES depiction and GPU architectures as well as our improvements to the processing of the surface parts.

## 3 Geometric definition

Before going into the details of the SES computation, we recall the theory at the basis of our method, and how it can be leveraged for rendering purposes. The nomenclature used throughout this article are listed in table 1.

| Symbol | Signification |
|---|---|
| $\mathcal{M}$ | A molecule. |
| $a_i$, $c_i$, $r_i$, $\mathcal{N}(a_i)$ | The $i^{\text{th}}$ atom, its center, vdW radius and neighborhood. |
| $r_p$ | The probe radius. |
| $\mathscr{C}_{ij}$, $c_{ij}$, $r_{ij}$, $n_{ij}$ | A SAS circle between $a_i$ and $a_j$, its center, radius and normal. |
| $\mathcal{I}$ | The set of intersected SAS circles. |
| $x$, $\mathcal{X}$ | An SAS intersection and their set. |
| $\mathcal{X}(\mathscr{C})$ | The set of SAS intersections of a given SAS circle. |
| $P_+$ | A convex patch. |
| $P_-$, $T$ | A concave patch and its tetrahedron. |
| $P_t$, $P_{tf}$ | Segment and full toroidal patches. |

TABLE 1: Nomenclature

### 3.1 SES patches

To offer a homogeneous presentation, we rely on Quan & Stamm [8]'s depiction to present the parts of the surface, refer to as patches. Its implicit geometric singularity handling and structural simplification make it an appropriate foundational design which is also adequate for rendering purposes.

As previously stated, the SES is described as the interaction surface of a spherical probe rolling on the vdW surface. The latter is composed of spheres defined by the center $c_i$ and vdW radius [7] $r_i$ of every atom $a_i$ belonging to the molecule $\mathcal{M}$. The surface accessible to the probe is the SAS and is defined as the union of the inflated vdW spheres where every atom's radius is increased by the probe's radius $r_p$. As demonstrated by Quan & Stamm [8], the individual component types of the SES can be described as distance functions from their corresponding parts on the SAS. They aim at giving the distance from a point $p \in \mathbb{R}^3$ to the surface which can be used for rendering with sphere-tracing [33] for instance. In this paper, we refer to values less than 0 as inside the surface and values greater than 0 as outside of the surface. An illustration of these patches for a given molecule is provided in fig. 3. They are based on SAS geometry bits created from the intersection of SAS spheres: SAS circles and intersections. SAS circles, noted $\mathscr{C}$, are small circles delimiting the intersection of two SAS spheres while SAS intersections, noted $x$, are points coming from the intersection of three SAS spheres or two SAS circles.

**Convex Spherical Patches $\mathbf{P_+}$.** As illustrated in fig. 4, a SAS sphere can be intersected by other spheres. The resulting geometry

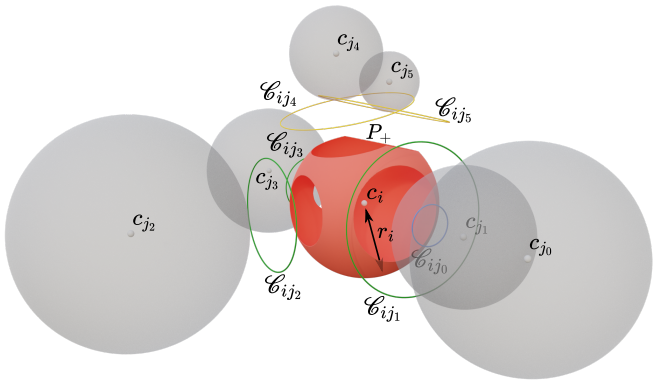


Fig. 4: Illustration of a convex patch $P_+$. We can notice its corresponding set of small circles at the basis of its composing sector list. In red: convex patch $P_+$. In green: full SAS circles. In yellow: intersected SAS circles. In blue: $\mathscr{C}_{ij_0}$, a circle buried in $a_{j_1}$, which does not contribute to the surface.

is a partial sphere which frontier is derived from intersecting SAS circles delimiting its surface. Then, the point $p$ is on the SES if it is on the vdW sphere and if it does not belong to an SAS circle $p \notin \mathscr{C}_{ij}$, $\forall a_j \in \mathcal{N}(a_i)$ as $a_i$'s neighborhood creating SAS circles $\mathscr{C}_{ij}$. This test was originally performed by maintaining circle intersection topology in Quan & Stamm [20]'s data structure. However, such construction is computationally expensive as well as being heavy. To alleviate this cost, we rely on per-patch sets of spherical sector as used by previous works [32]. A sector is composed of an orientation from the sphere's center and an angle derived from its corresponding circle. It can then be used to test if $p$ is part of the surface and allow to simplify Quan & Stamm [20]'s data structure. Ours is composed of the atom position and radius as well as a list of vectors and radii representing its sectors.

**Toroidal Patches $\mathbf{P}_t$ and $\mathbf{P}_{tf}$.** As depicted in fig. 5, the intersection of two SAS spheres is an SAS circle. If $p$ is located in the triangle $\triangle(p' c_i c_j)$ described by both $a_i$ and $a_j$ centers and $p'$ the projection of $p$ onto their resulting SAS circle $\mathscr{C}_{ij}$, then it belongs to an SES toroidal patch $P_t$. Hence, its distance $d(P_t,\, p)$ to the SES is its signed distance to $p'$ adding $r_p$:

$$\begin{aligned} d(P_t,\, p) &= -||p' - p|| + r_p,\; p \in \triangle(p' c_i c_j) \\ p' &= c_{ij} + r_{ij}\frac{v}{||v||} \\ v &= p + ((c_{ij} - p)\cdot n_{ij})n_{ij} - c_{ij} \end{aligned} \tag{1}$$

with $c_{ij}$, $n_{ij}$ and $r_{ij}$ as respectively the center, normal and radius of $\mathscr{C}_{ij}$ and $v$ as the vector from $c_{ij}$ to $p'$.

Circles become segments when intersected by other circles. The SDF remains the same but its enclosing shape is bounded by $x^l$ and $x^m$, the two bounding intersections creating an arc that must be taken into account during evaluation. This can be achieved by testing if $p'$ is located between $x^l$ and $x^m$:

$$\measuredangle\left(x^l - c_{ij}, p' - c_{ij}\right) < \measuredangle\left(x^l - c_{ij}, x^m - c_{ij}\right) \tag{2}$$

with $\measuredangle(\vec{v}, \vec{w})$ as the angle between $\vec{v}$ and $\vec{w}$. To ease the differentiation between both shapes, we use $P_{tf}$ for patches belonging to complete circles and $P_t$ for those belonging to segments, respectively in green and yellow in fig. 5.

In contrast with previous depictions [14], this representation is not suffering from the spindle torus singularity and does not require the use of a solver leading to instabilities [32]. Moreover, while previous works store cutting geometry planes, we use Quan & Stamm [20]'s data structure to avoid these additional costs on memory and computation. It was originally including, in addition to both atoms and intersections indices, the circle center and radius as well as the angle of the segment. Based on the presented equations these data can be quickly recomputed on demand with both atoms and intersections. Our data structure for $P_t$ and $P_{tf}$ is then simplified by only storing the two atoms indices as well as the intersection indices for $P_t$.

**Concave Spherical Patches $\mathbf{P}_-$.** The intersection of three SAS spheres leads to one or two SAS intersections, as illustrated in fig. 6. $p$ belongs to an SES concave spherical patch $P_-$ if it is located within the SAS cavity and is not belonging to any other type of patch. In such context, the point is then positioned in the union of every tetrahedron $T$ described by the three atoms centers $c_i$, $c_j$, $c_k$ and their resulting SAS intersection point $x \in \mathcal{X}$, with $\mathcal{X}$ as the set containing all intersections belonging to $\mathcal{M}$.

The distance between $p$ and the SES $d(P_-,\, p)$ is the distance to the closest probe sphere of radius $r_p$. While this allows to completely define the SES cavity, it is not sufficiently compact to fit a rendering algorithm. We can then add that, deriving these statements, a concave patch $P_-^v$ can be more compactly rendered as a sphere centered in its SAS intersection $x^v$, of radius $r_p$, intersected by the three planes describing the sides of its tetrahedron $T^v$ and by refining the distance based on neighboring $P_-$. Moreover, such test is only required if the sphere is intersecting the plane of $T^v$ described by $c_i$, $c_j$ and $c_k$ [16], [31]. In our data structure, these patches are represented by their position, the indices of their corresponding atoms, and the position of their potential neighbors. This is heavier than Quan & Stamm [20]'s data structure since we directly provide $P_-$'s neighbors to meet rendering speed requirements as in previous works [28], [31].

By using our data structures, we are able to render the SES without additional data or preprocessing. Then, they represent the target of the pipeline described in the following sections.

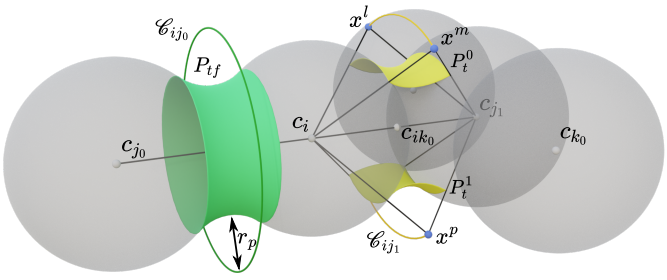


Fig. 5: Illustration of toroidal patches $P_t$ and $P_{tf}$. On the illustration, we can see three toroidal patches and their corresponding SAS structures. Both kinds of patches are defined by the SAS circle coming from the intersection of the SAS spheres of their atoms, while $P_t$ are also bounded by their corresponding intersections. In green: a complete circle patch $P_{tf}$ with its circle. In yellow: two segments patches $P_t$ on the same circle.

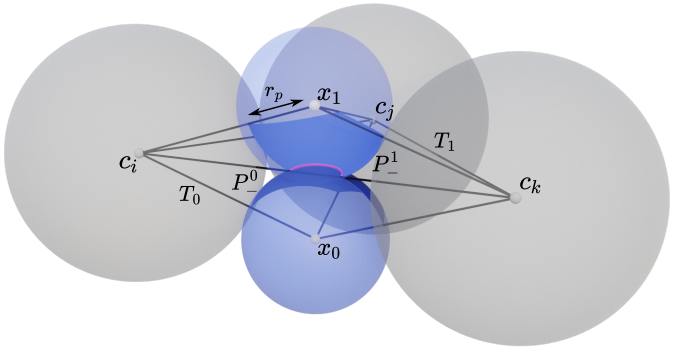


Fig. 6: Illustration of the concave patches $P_-$. We can notice two intersecting concave patches $P_-$ and their respective tetrahedra. As illustrated, two $P_-$'s spheres can intersect resulting in a hole in the surface.

### 3.2 Computation of circles and intersections

To acquire the previously presented patches, several key geometry parts need to be computed. We chose to rely on the set of equations provided by Totrov et al. [19] and Quan & Stamm [20] in their implementation which ally ease of execution and compactness in terms of data. More details regarding these equations can be found in the supplemental materials.

We especially rely on the circle classification provided by Quan & Stamm [20]. As illustrated in fig. 7, we can test if a circle $\mathscr{C}_{ij}$ is occluded by the SAS sphere of an atom $a_k$ by comparing the distance between $c_k$ and its furthest point on $\mathscr{C}_{ij}$ to the radius of the SAS sphere. If the distance is smaller, the circle is fully inside the sphere. In the same way, we can test for an intersection between $\mathscr{C}_{ij}$ and $a_k$'s SAS sphere by testing if the distance between $c_k$ and its closest point on $\mathscr{C}_{ij}$ is smaller than the radius of the sphere.

In the following sections of this paper, we show that these computations can be the basis of a strong diminution of the memory consumption.

## 4 Structural SES computation

In this section, we present the main steps of the processing pipeline. As many previous works [13], [28], [30]–[32], it takes place during the construction of the SAS structural information. Once the geometry is produced, it can be used for various purposes, including rendering or geometry analysis that can be required for in-depth surveys. Our pipeline mostly features a SAS circle classification and two processing dedicated to SAS intersection and segment computation. The processing pipeline, the hierarchy of its main steps, and their corresponding novelties compared to previous works are given in fig. 8.

### 4.1 SAS circles computation and classification

The circle acquisition process is divided into two main parts: computation and classification. The first one aims at finding the intersection between all SAS spheres while the other is oriented toward the identification of circles that will be used to produce SES data. This last part is essential since it allows to strongly reduce the memory used in the following stages.

To compute every atom's neighborhood $\mathcal{N}(a)$, we find the set of SAS spheres intersecting the current atom by checking if the distance between $c_i$ and their center is less than $r_i + r_j + 2r_p$. This spatial query is achieved by using a uniform acceleration grid as usually performed in the related works [13], [28], [32] and in GPU algorithm relying on neighborhood queries [34], [35] for both its competitive construction and query performance. The acceleration structure is built by first computing the hash value of each atom describing their corresponding grid cell, sorting atoms on this basis, and finally compute for each cell their start and end atom sorted indices. We rely on an empirical dynamic cell size $g = \min\left(2^m > \lceil \log_2(\#\mathcal{M}) \rceil\right)$ which slowly grows in power of two with the atom number $\#\mathcal{M}$.

Once atoms' neighborhood has been discovered, the challenge is located in its processing. Indeed, as the neighbor search is achieved on an atom basis, it contains a lot of duplicated circles or buried into neighboring atoms. Even if some previous works were using intersection discovery to find the limited set of useful circles [13], we rely on a circle classification scheme as performed by other works [19], [28]. However, whereas the latter compute hidden atoms and circles, we leverage this process to provide an additional refinement. As further studied in fig. 11 they can directly serve to gather the neighborhood by circle type.

For every circle $\mathscr{C}_{ij}$ we check if a neighbor SAS sphere associated with an atom $a_k \in \mathcal{N}(a_i)$ is occluding or intersecting the circle. While the computation can be directly stopped if the circle is occluded, it is mandatory to test every neighboring sphere to know if a visible circle is full or intersected. These operations can be achieved in a single step since they rely on the same data. From these tests, a circle can be classified as:

- Buried: $\mathscr{C}_{ij}$ is located inside another atom and is not contributing to the surface.
- Full: $\mathscr{C}_{ij}$ is creating a sector in $P_+^i$ and will produce a $P_{tf}$.
- Intersected: $\mathscr{C}_{ij}$ is creating a sector in $P_+^i$ and may produce several SAS intersection points and segments.

We refer to $\mathcal{I}$ as the set of intersected circles. It is also important to note that all intersected circles may not result in intersections and segments. Indeed, their detected intersections can be hidden by other atoms which case will be handled in the next part of this section. Based on this classification scheme, we can directly derive

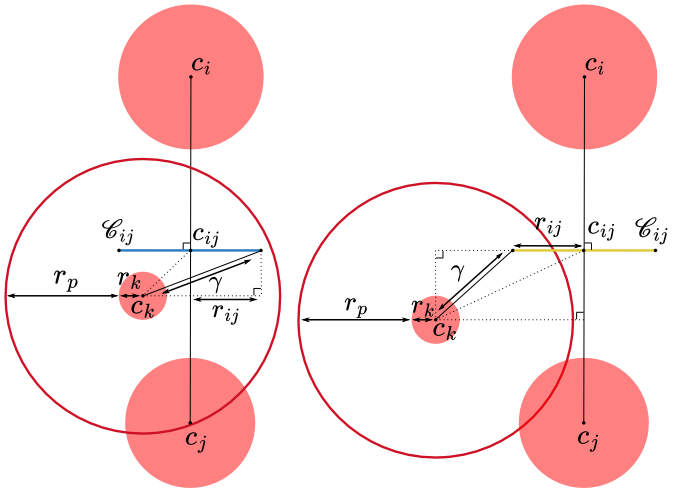


Fig. 7: Circle classification tests. The left schema represents the occlusion test while the right represents the intersection test. $\gamma$ is the distance that is compared to the radius of $a_k$'s SAS sphere.

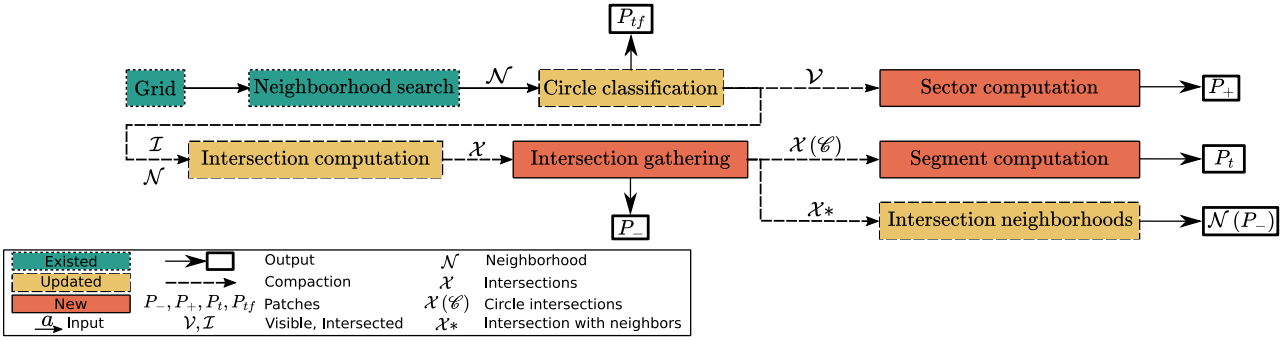


Fig. 8: Illustration of the computation pipeline giving the geometry of the complete SES. The schema also emphasizes the modification we made to the state-of-the-art SES computation method on GPUs [28].

both $P_{tf}$ and $P_+$ patch data. The set of $P_{tf}$ is composed of full circles, while the set of $P_+$ is created from each atom by creating a sector for every visible $\mathscr{C}_{ij}$. Its axis is equal to the circle's normal $n_{ij}$ and its radius can be derived with:

$$\cos^{-1}\left(n_{ij} \cdot \frac{o - c_i}{r_i + r_p}\right),\ o \in \mathscr{C}_{ij} \tag{3}$$

### 4.2 SAS intersections

SAS intersections are built from $\mathcal{I}$ since it represents the restricted set of visible circles which are intersected by neighboring atoms. However, as a part of the complex molecular structure, we must verify that every discovered intersection is not occluded by an atom next to the three spheres at the origin of its creation.

To do so, for every intersected circle, we test if intersections occur with every neighboring intersected circle $\mathscr{C}_{ik} \in \mathcal{I}$, $a_k \in \mathcal{N}(a_i) \cap \mathcal{N}(a_j)$ and compute the two possible SAS intersections $x_0$ and $x_1$ accordingly. The resulting intersection points can then be filtered by checking if they are hidden by a neighboring atom as achieved in previous works [8], [13], [27].

As stated in section 3, even if $P_-$'s singularities are implicitly handled in Quan & Stamm [8]'s work thanks to their SDF formulation, we still rely on neighborhood search to alleviate the cost of our method for rendering. This can be achieved in a similar way as the circle discovery discussed in section 4.1 by finding the neighboring ones whose center is closer than $2r_p$.

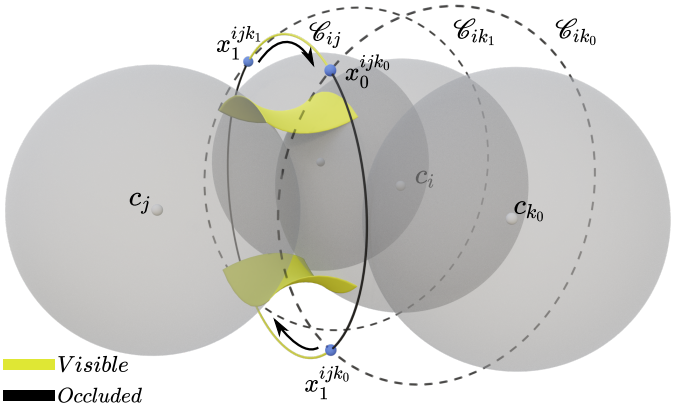


Fig. 9: Illustration of the $P_t$ construction process. $\mathscr{C}_{ij}$'s intersection list is processed starting at intersection $x_1^{ijk_0}$ and by sorting according to the sign used for its computation. In this context, $x_1^{ijk_0}$ leads to a trigonometric ordering from $c_i$'s perspective. Based on this statement, we know that black parts of the circle are occluded while yellow parts yield to segments.

### 4.3 SAS segments

SAS segments result from intersected circles found during the SAS intersections computation. Then, as every segment is created from two intersections, we can assert that:

$$\#\mathrm{P}_t = \sum \frac{\#\mathcal{X}(\mathscr{C}_{ij})}{2},\ \forall \mathscr{C}_{ij} \in \mathcal{I} \tag{4}$$

with $\#\mathcal{X}(\mathscr{C}_{ij})$ as the number of intersections of $\mathscr{C}_{ij}$.

In previous works [13], [14], [28], [32], intersections and segments were sometimes established together producing redundancy. However, segments can be directly created from the circle intersection list by applying an angle-wise sort as used by alternative methods [8], [31]. Finally, intersection equations describe the relative position of the intersection point compared to the sphere triplet [20]. It can thus be used to recover the occluded part of $\mathscr{C}_{ij}$.

As illustrated in fig. 9, we can select a first intersection, no matter its position on the circle, and retrieve its angular ordering to avoid creating segments in the occluded region. We also have to take into account the order of the circle's indices in the triplet $\{i, j, k\}$ since the angular ordering is based on a circle perspective. Hence, if the circle is indexed as $\mathscr{C}_{ik}$ the ordering is inverted compared to $\mathscr{C}_{ij}$ and $\mathscr{C}_{jk}$. The sorting algorithm is then performed on the angle of every intersection compared to the first one. Every intersection couple can then be gathered as a resulting segment.

To avoid storing the angular ordering used after the segment computation, we inverse both segment intersections if the picked direction is reversed. Every segment can then be processed with the same operations without relying on other information.

## 5 GPU Implementation

Our method is designed to benefit from parallel environments such as modern GPU's SIMT. We then describe several optimizations performed in our implementation regarding the memory management process as well as the computation load distribution.

We rely on CUDA 11.6 for its available features like its memory pool to limit temporary allocation costs and libraries such as Thrust and CUB [36]. However, any other GPU APIs supporting these characteristics could be used. In the following section, we refer to CUDA vocabulary and its two hierarchical categories:

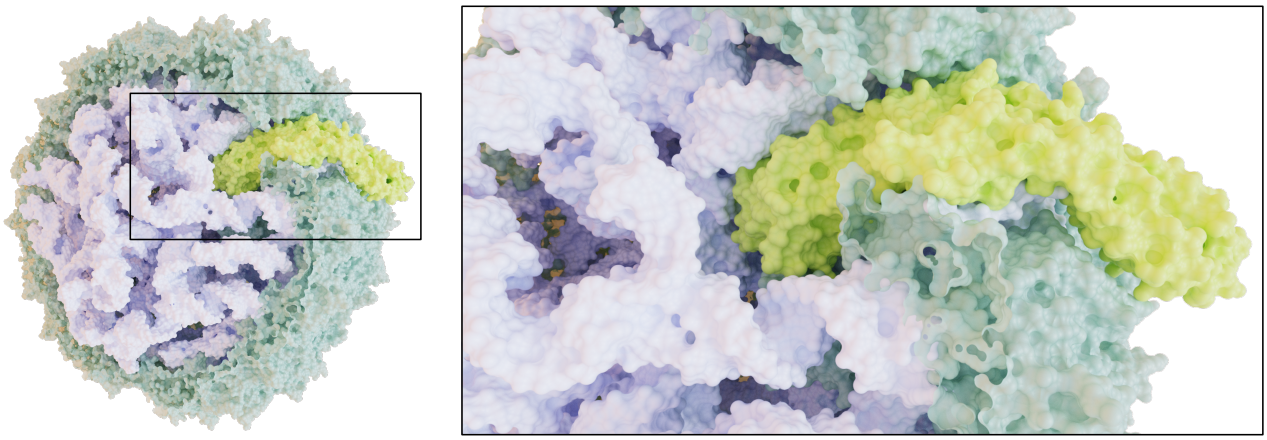

Fig. 10: Rendering of the SES of the complete model of phage Qbeta virion (PDB: 7LHD) which capsid is partially removed through plane culling to allow visualization of RNA and the maturation protein. It also emphasizes the completeness of the acquired surface, allowing tunnels and pocket visualization. The left image illustrates how the SES can be useful to give a proper visualization of the overall shape of this large complex, especially with RNA's twists, while the right image shows that it remains true with thinner details regarding structural interactions.

block and warp. When a computation is launched, threads are grouped into blocks sharing characteristics such as memory level and synchronization behavior. These blocks are themselves split into small groups of few threads (32 or 64) called warps. Inside a block, warps can be executed both sequentially and in parallel while, within a warp, threads are fully concurrent and support special instructions allowing cooperative processing. In this section, we present the improvements we made regarding the two highly parallel and cooperative processing of both intersections and segments leveraging these GPU characteristics.

### 5.1 Circle types handling

The first optimizations are based on the classification presented in section 4.1. Due to their respective types, circles require very different processing which would introduce execution divergence.

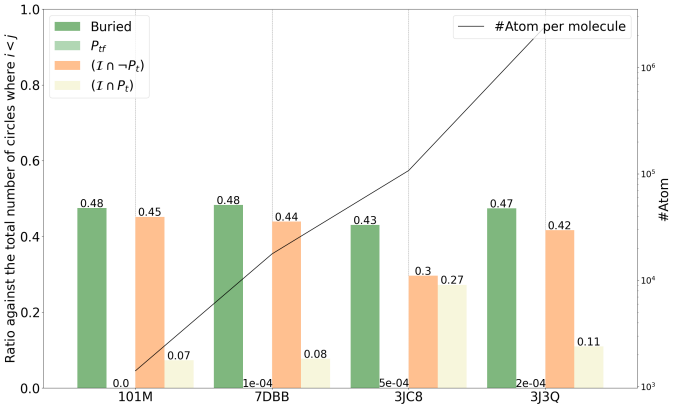


Fig. 11: Ratios of the number of circles $\mathscr{C}_{ij}$ where $i < j$ and classified as buried, full $P_{tf}$, intersected without visible intersections $(\mathcal{I} \cap \neg P_t)$ or intersected with visible intersections $(\mathcal{I} \cap P_t)$ out of their total and generated with a probe radius $r_p = 1.4\,\text{Å}$. Almost 50% of the circles are buried. $P_{tf}$ and $(\mathcal{I} \cap P_t)$ are the only categories of circles really creating surface patches, in contrast with $(\mathcal{I} \cap \neg P_t)$ circles which contributes through sectors to $P_+$, and represents a small portion of the overall number of circles.

To address this issue, we handle each circle types through a dedicated process. Since $P_{tf}$ only represent a very small part of the total amount of circle as shown in fig. 11, they can be directly saved during the classification using atomic operations. In contrast, the intersected circles set $\mathcal{I}$ is efficiently built by removing other circle types through stream compaction. From these operations, we derive the set of visible circles $\mathcal{I} \cup P_{tf}$ at the basis of $P_+$ patches construction.

### 5.2 Cooperative intersection processing

One of the most critical steps of the SES computation is to find SAS intersections. This is mainly caused by the need to traverse the neighborhood. As illustrated in fig. 11, the ratio of intersected circle $\mathcal{I}$ out of the total number of circles is about 50% and remains steady across all the samples of our test dataset, even with strongly varying atom number and positional density. The complexity of the processing is then located in the number of tests required to compute and validate intersections with neighboring circles. Based on these statements, we can assert that the parallelization of this exploration allows better throughput. Furthermore, the processing of a single circle is by itself heavy and a dedicated processing is interesting. Thus, we introduce a cooperative and parallel processing strategy of the neighborhood of every atom $a_i$, which is described in a high-level language in algorithm 1.

We start with the construction of a per atom task list composed of its set of intersected circles. Each warp then fetches a circle $\mathscr{C}_{ij}$ of the block for processing. For each $a_k \in \mathcal{N}(a_i), k \neq j$, we test sequentially with the entire warp if there exists an intersected circle $\mathscr{C}_{jk}$ where $i < j < k$ (l.8-10 in algorithm 1). In order to efficiently compare both neighborhoods $\mathcal{N}(a_i)$ and $\mathcal{N}(a_j)$, we process subsets of $\mathcal{N}(a_i)$ of the size of the warp to benefit from its local compute power. We also rely on a bit mask $\mathcal{K}$ of the size of the warp to keep track of intersections requiring further processing as illustrated in fig. 12. After the neighborhood exploration, threads directly calculate resulting intersection data which allows to further refined the mask $\mathcal{K}$ based on the existence of each intersection (l.11-13 in algorithm 1). Finally, we perform the visibility test (l.14-18 in algorithm 1) by assigning a subset of $\mathcal{N}(a_i)$ to each threads of the warp. Intersections are tested for occlusion sequentially with

an intra-warp parallel test. Visible intersections are saved to shared memory before block-wise saving to device memory. These steps are repeated until all $a_i$'s neighborhood $\mathcal{N}(a_i)$ has been processed for a given circle $\mathscr{C}_{ij}$.

This process allows a more uniform distribution of the workload as well as making memory access more coherent at warp level while reducing the amount of high-level synchronization. Moreover, it benefits from efficient use of caching strategies thanks to the processing locality. Indeed, $\mathcal{N}(a_i)$ is shared between the full block while $\mathcal{N}(a_j)$ is loaded at warp level. This cache helps the search across different parallelism levels strengthening its value.

Finally, to reduce the cost of the following processing, we refine the number of $P_-$ patches that actually require a neighbor search. Intersections with neighbors are then placed at the beginning of their buffer which allows us to directly operate on contiguous data.

### 5.3 Cooperative segment processing

The segment creation process requires to retrieve for a given circle its set of intersection. While it can be simply achieved by appending to a list in a sequential implementation, this interleaved pattern is a major concern in a concurrent system. Previous work [28] solved it by computing all the intersections of a given circle at the cost of repeated computation. This allows avoiding synchronization but it also increases a lot the workload.

To address this issue, we compute for each circle its intersection count $\#\mathcal{X}(\mathscr{C}_{ij})$ and corresponding intersection indices $\mathcal{X}(\mathscr{C}_{ij})$ through a two-step process. During the intersection computation (fig. 8), we only record their circle indices and increment atomically the three corresponding counts $\#\mathcal{X}(\mathscr{C})$ (l.22-24 in algorithm 1). A second process, intersection gathering (fig. 8) is then launched to create the mapping between circles and intersections $\mathcal{X}(\mathscr{C})$. This is made by using a thread per intersection to reserve a writing location within $\mathcal{X}(\mathscr{C})$ through an atomic operation on its element count $\#\mathcal{X}(\mathscr{C})$ which is done for its three corresponding circles. The atomic operations only produce a low amount of thread concurrency since they only happen on a circle basis. This fast processing not only reduces overlapping dependencies but also allows to postpone the allocation of the intersection position buffer which is only performed when the exact intersection number is known, between intersections computation and gathering.

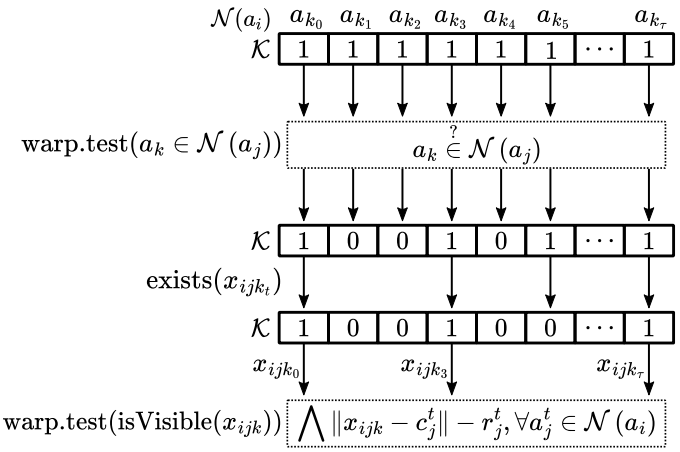


Fig. 12: Schema of the cooperative SAS intersection process used in algorithm 1. The warp is refining intersections that must be computed by iteratively modifying the bit mask $\mathcal{K}$. Dotted lines: intra-warp operation processed in parallel with all warp's threads but iteratively for each entry in $\mathcal{K}$ set to one.

**Algorithm 1:** Intersections cooperative processing

**Data:** $\mathcal{N}(a)$, $\mathcal{I}$
**Result:** SAS Intersections $\mathcal{X}$

1 block.load($\mathcal{N}(a_i)$) ▷ To shared memory
2 $\mathcal{X}' \leftarrow \emptyset$ ▷ To shared memory
3 $t \leftarrow$ thread.id() ▷ Relative index in warp
4 block.sync()
5 **forall** $\mathscr{C}_{ij} \in \mathcal{N}(a_i) \cap \mathcal{I}, i < j$ **do** ▷ A circle per warp
6  warp.load($\mathcal{N}(a_j)$) ▷ From device memory
7  $\mathcal{K} \leftarrow \neg 0$ ▷ Initialize bit mask
8  **forall** $k \in \{0, \ldots, \text{size(warp)}\}$ and $\mathcal{K}$[k] $\neq 0$ **do**
9  $\mathcal{K}[k] \leftarrow$ warp.test($a_k \in \mathcal{N}(a_j)$)
10  **end**
11  **if** $\mathcal{K}$[t] $\neq 0$ **then**
12  $\mathcal{K}$[t] $\leftarrow$ exists($x_{ijk_t}$)
13  **end**
14  **forall** $k \in \{0, \ldots, \text{size(warp)}\}$ and $\mathcal{K}$[k] $\neq 0$ **do**
15  **if** warp.test(visible($x_{ijk}$, $\mathcal{N}(a_i)$)) **then**
16  $\mathcal{X}' \leftarrow \mathcal{X}' \cup \{x_{ijk}\}$ ▷ Temporary save
17  **end**
18  **end**
19 **end**
20 block.sync()
21 **forall** $x_{ijk} \in \mathcal{X}'$ **do** ▷ To device memory
  ▷ Per circle atomic operation
22  $\#\mathcal{X}(\mathscr{C}_{ij}) \leftarrow \#\mathcal{X}(\mathscr{C}_{ij}) + 1$
23  $\#\mathcal{X}(\mathscr{C}_{ik}) \leftarrow \#\mathcal{X}(\mathscr{C}_{ik}) + 1$
24  $\#\mathcal{X}(\mathscr{C}_{jk}) \leftarrow \#\mathcal{X}(\mathscr{C}_{jk}) + 1$
25  $\mathcal{X} \leftarrow \mathcal{X} \cup \{x_{ijk}\}$
26 **end**

Thanks to the previous step, we know the intersections of every circle at the origin of their segments. However, before their actual computation, we find the indices of the intersected circles which contribute to some intersections. As previously stated, $\mathcal{I}$ contains the circles that are able to produce an intersection but we don't know before the computation step if this intersection is visible. As shown in fig. 11, only a small part of these circles actually contribute to the surface. By only launching threads on circles producing intersections, we avoid unnecessary overhead. Furthermore, based on SAS circles intersection count $\#\mathcal{X}(\mathscr{C}_{ij})$, we can compute the exact memory required to store segments data.

Once circles intersections are retrieved, we directly proceed to the angular sorting, as illustrated in algorithm 2. If applied on a circle basis, this step could result in highly divergent processing due to sorting algorithm properties. However, direct use of parallel cooperative sorting algorithms would lose the locality of the computation or give poor levels of parallelism. To address this issue, we perform the sorting of multiple circles at once. This is made possible by applying an offset to the data (l.12 in algorithm 2). This allows reserving a specific range for a given circle while benefiting from the parallelism offered by cooperative sorting algorithms. Circles intersection indices and relative angles are stored in threads memory to execute the sorting at warp level. After this computation and based on previous stream compaction, we are able to store segments data fully concurrently on a per-circle basis.

This part of the processing strongly benefits from the data

structure derived from Quan & Stamm [8]'s depiction and allows creating $P_t$ geometry in a fast and efficient parallel scheme.

### 5.4 Estimating the required memory

Even if we chose to closely compact arrays used during the computation, the allocation scheme remains a challenging aspect of large structure processing. This is especially true regarding the intersections which were often one of the most consuming geometry bits of the surface. In previous works, this was solved with pre-allocation from the base circle number [37] or with an estimated percentage of the possible triplets [13]. While the latter allowed to improve the memory consumption process, we rely on the properties determined during circles classification which fit more the architecture of our pipeline and allows a thinner configuration.

Since the number of circles able to produce an intersection is theoretically known, we can bound the maximum number of intersections by:

$$\#\mathcal{X} \leq \#\mathcal{I} \ \max\left(\#\mathcal{X}\left(\mathscr{C}_{ij}\right)\right) \tag{5}$$

with $\#\mathcal{I}$ as the number of visible and intersected circles and $\max\left(\#\mathcal{X}\left(\mathscr{C}_{ij}\right)\right)$ as a constant bounding the maximum number of intersections per intersected circle.

As illustrated in fig. 11, more than half of the intersected circles are not contributing to the surface. $\max\left(\#\mathcal{X}\left(\mathscr{C}_{ij}\right)\right)$ is then set by taking into account this ratio of unused circles and serve as a mean of the possible count of intersection per circle. It allows to strongly reduce the size of the pre-allocated buffer since the size of $\mathcal{I}$ is significantly smaller than the size of $\mathcal{N}$.

From our experiments, we have bound the maximum size of the neighborhood of an atom to 128 (for a standard probe radius $r_p = 1.4$Å) as well as the mean intersection number per intersected circles to 2. This allows to handle large molecules without parameter change. Circles intersections bound is in practice not restricting since it only aims to be an overall mean. We do allow each circle to contain up to 16 intersections during $P_t$ creation process in our implementation.

After the execution of this pipeline, we have obtained all patches data presented in section 3 which can be directly rendered as presented by previous works [26], [31].

**Algorithm 2:** Segments cooperative processing

**Data:** $\mathcal{X}\left(\mathscr{C}_{ij}\right)$
**Result:** SAS Segments $\mathcal{S}$

$x \leftarrow \mathcal{X}\left(\mathscr{C}_{ij}\right)$
reversed $\leftarrow$ isReversed$\left(\mathscr{C}_{ij}, x[0]\right)$
$t \leftarrow$ thread.id() ▷ Relative index in warp
$\delta_t \leftarrow t\delta,\ \delta > 2\pi$
angles $\leftarrow \{0\}$ ▷ To thread memory
indices $\leftarrow \{0\}$ ▷ To thread memory
**for** $a \in \left\{1, \ldots, \#\mathcal{X}\left(\mathscr{C}_{ij}\right) - 1\right\}$ **do**
  $\alpha, \beta \leftarrow x[a-1], x[a]$
  **if** *reversed* **then**
    $\alpha, \beta \leftarrow \beta, \alpha$
  **end**
  $\theta \leftarrow \delta_t +$ angleBetween($\alpha$, $\beta$)
  angles $\leftarrow$ angles $\cup \{\theta\}$
  indices $\leftarrow$ indices $\cup \{a\}$
**end**
indices $\leftarrow$ warp.sortByKey(angles, indices)
**for** $s \in \left\{0, \ldots, \#\mathcal{X}\left(\mathscr{C}_{ij}\right)/2 - 1\right\}$ **do**
  $a, b \leftarrow$ indices$[2s]$, indices$[2s+1]$
  $\alpha, \beta \leftarrow x[a], x[b]$
  **if** *reversed* **then**
    $\alpha, \beta \leftarrow \beta, \alpha$
  **end**
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{i, j, \alpha, \beta\}$ ▷ To device memory
**end**

## 6 Performance and discussion

To compare the performance of our pipeline against previous works, we chose the GPU Contour-Buildup implementation [28] publicly available in the framework Megamol [37]. While several methods presented many improvements, none of them provided similar or faster computation time [13], [32]. We also present the results of our method by only producing the exterior molecular surface which can be preferred for fast visualization and computation. This is made by replacing $P_+$ with complete vdW spheres and unifying $P_{tf}$ and $P_t$ processing which result in a simplification of the algorithm and its memory requirements. However, this implementation respects the final data structure used by the original algorithm and can thus be rendered with the same overall strategy. Unlike the original method, we use a dedicated buffer rather than limited texture memory allowing larger allocation without impact on the rendering time.

Experiments were conducted on an AMD Ryzen 5 1600, an NVIDIA RTX 2080 and with a dataset of fifteen molecules from hundreds to millions of atoms, gathered from the RCSB Protein Data Bank (PDB) [38]. Rendered views of the dataset molecules are given as supplemental material. All benchmarks have been achieved with a probe radius equal to the water molecule radius 1.4 Å [39], as it is a common solvent. However, as illustrated in fig. 13, our implementation remains stable with larger probe radius.

Performance benchmarks are presented in table 2. First, we can notice that our method provides similar computation times compared to the parallel Contour-Buildup while acquiring the complete surface information. The exterior molecular surface

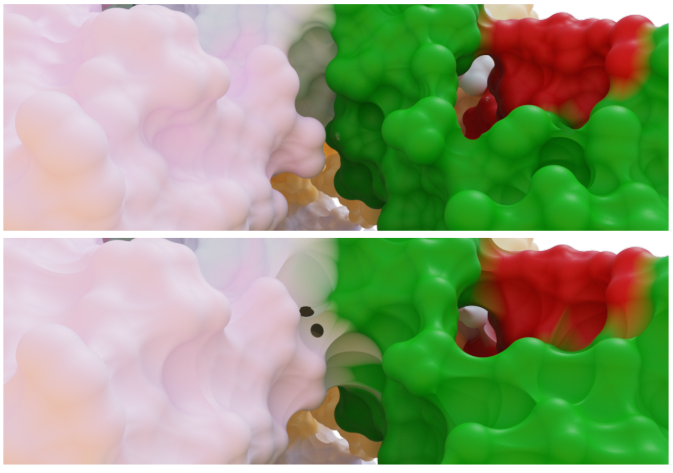

Fig. 13: Comparison of two SES (PDB: 1AON) computed with our method, using a standard probe radius (top: 1.4 Å) and a larger one (bottom: 2.4 Å). The surface is emphasized by the patches corresponding chains color to provide structural information. On the bottom image, we can notice typical $P_-$ intersections.

| PDB Id | #Atom | Krone et al. [28] | | Ours | | Ours exterior | |
|---|---|---|---|---|---|---|---|
| | | Time (ms) | Mem. (MB) | Time (ms) | Mem. (MB) | Time (ms) | Mem. (MB) |
| 1AGA | 126 | 4.50 | 11.92 | 3.22 | 0.39 | 2.21 | 0.15 |
| 101M | 1413 | 4.51 | 133.55 | 3.75 | 4.50 | 2.51 | 1.76 |
| 1VIS | 2531 | 4.64 | 238.66 | 3.67 | 8.01 | 2.65 | 3.52 |
| 7SC0 | 11638 | 5.64 | 1108.02 | 4.70 | 36.17 | 3.54 | 16.16 |
| 3EAM | 13505 | 6.44 | 1282.09 | 5.20 | 42.69 | 4.01 | 19.63 |
| 7DBB | 17733 | 7.00 | 1675.43 | 5.80 | 56.18 | 4.49 | 25.50 |
| 1A8R | 45625 | 11.14 | 4307.71 | 9.48 | 144.92 | 7.36 | 65.79 |
| 7O0U | 55758 | 12.39 | 5263.23 | 10.71 | 177.25 | 8.49 | 80.29 |
| 1AON | 58870 | 11.95 | 5552.49 | 10.75 | 185.33 | 8.37 | 86.86 |
| 7RGD | 65008 | 16.86 | 6123.51 | 18.06 | 214.33 | 14.64 | 105.35 |
| 3JC8 | 107640 | - | - | 14.99 | 324.03 | 11.60 | 147.57 |
| 7CGO | 335722 | - | - | 47.64 | 1054.85 | 36.34 | 486.03 |
| 4V4G | 717805 | - | - | 104.72 | 2249.47 | 81.64 | 1130.35 |
| 6U42 | 1358547 | - | - | 200.79 | 4287.12 | 157.23 | 2162.99 |
| 3J3Q | 2440800 | - | - | 700.17 | 7631.24 | 324.22 | 3744.73 |

TABLE 2: Comparison of our method compared to the publicly available implementation of GPU Contour-Buildup [28] in Megamol [37]. Times are given in milliseconds (ms) and represent the mean of 1000 iterations per case, while maximum memory use is given in megabytes (MB). Every experiment has been conducted with 100 warm-up iterations which were not considered in the final results.

computation remains consistently and moderately faster in all experiments. However, the compared method is bound in our test dataset to the protein 7RGD composed of 65 008 atoms while ours is capable of handling more than 2 million atoms thanks to its optimized memory consumption scheme. We can notice that the memory consumption of all methods grows linearly depending on the number of atoms. However, our exterior computation consumes almost 60 times less memory than the compared method on the largest complex (PDB Id: 7RGD) while the complete one uses 30 times less. These results illustrate how the parallelization of the process as well as the precise allocation scheme help to provide fast results while handling large molecular complexes. The complete surface acquisition can thus be made in low computation time and consumer-like hardware. On larger complexes, the exterior-only molecular surface computation remains almost constantly twice more memory efficient. These improvements mostly come from the lack of $P_+$ processing as well as the unification of $P_t$ and $P_{tf}$. Based on these results, our method can compute the complete surface of large complexes coming from small viruses, as illustrated in fig. 10, or part of larger ones, as illustrated in fig. 1, which was not possible before on GPUs. This is made without trade-offs regarding the computation time and/or the surface quality. These improvements could then lead to wider use of analytic computations for molecular geometry study or visualization and illustration. Moreover, the consequent memory consumption improvements could also be at the basis of larger availability across the various user configurations that would not be restricted anymore to high-end hardware. Experiments have also been re-conducted with an NVIDIA Titan RTX and an Intel i9 9900K and are given as supplemental material as well as a plot comparing the memory scalability of the studied methods.

To give a deeper analysis of our algorithm costs, we produced detailed GPU benchmarks given in fig. 14. First of all, we can estimate the additional costs such as allocation and CPU-GPU transfers required for the precise patch number estimation from the difference between GPU times and those presented in table 2. The proportions of these additional costs out of the total computation are, however, decreasing from approximately 60% for the smallest molecule of the dataset to 25% for the largest, as expected. Secondly, our method's costs are mainly shared by the first two stages. The $P_t$ discovery and the construction of $P_-$ neighborhoods are made in a short fraction of the total computation thanks to the highly concurrent scheme as well as the identification of intersecting probes resulting in complexity reduction. Even if our parallel implementation demonstrates fast computation time compared to previous works, the creation of $P_-$ is still one of the most consuming part of the computation. As shown in fig. 11, a consequent part of this process is dedicated to intersected circles without visible intersection. Finally, we can notice that the molecule and its geometrical properties have an impact on the different stages. As presented in other works, the ratio between the initial number of atoms compared to the ones contributing to the surface results in different computational cost at similar atom counts. This explains the varying ratios across the dataset regarding the different stages.

To test the data produced by our method, we implemented two rendering engines targeting different purposes. The first one is a path-tracer implemented with OptiX [40] which has been used to produce the molecule illustrations of this paper and demonstrates that it is suitable for high-quality rendering. The second one targets real-time visualization purposes with rasterization using impostors and geometry intersection in screen space as in previous works [26], [31], [41]. Both engines were built based on the expressions described in section 3 which are analytically intersected or sphere-traced [33]. The data produced by the pipeline is then directly accessed at rendering time. Even if the scope of this paper is oriented toward the construction of the SES, we provide performance benchmarks of our real-time rendering engine for information purposes only as supplemental material.

## 7 Limitations and future works

As previously presented, our method can compute the complete surface of large proteins with reduced impact on the computation time. However, it can suffer from limitations in specific cases.

First, to compute the surface of a single protein with varying probe radius, a full re-computation is needed and no previously computed data could be re-employed. Indeed, as the radius changes, all circle data may change due to non-uniform impacts on atoms radius ratio. Therefore, the complete pipeline must be re-executed. Additional parameter changes can also be needed regarding the

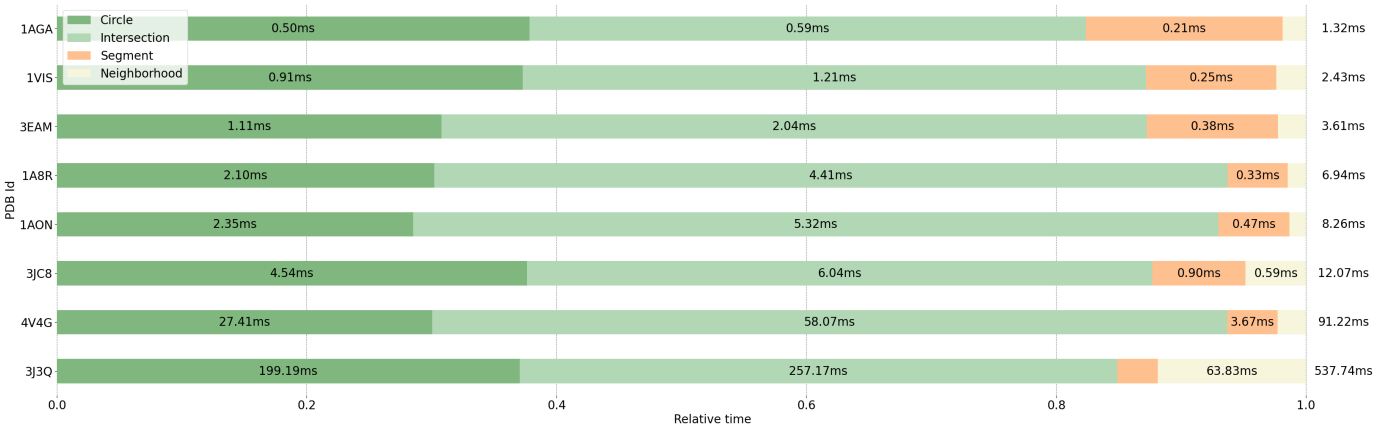


Fig. 14: Detailed GPU benchmarks of the full surface computation with a probe radius $r_p = 1.4\,\text{Å}$ on a subset of our test dataset. As illustrated, respective relative computation times are correlated to the molecule geometry influencing stage complexity. A similar analysis performed for Krone et al. [28]'s method is given as supplemental material.

maximum neighbor per atom [42]. Even if this might not be a drawback in the context of illustration since the probe radius may not vary too often, it could slow down a cavity visualization process.

Second, even if we rely on a circle classification to bound the number of circles able to produce an intersection, an important part of the computation is still dedicated to circles without visible intersections. A meaningful enhancement could then be made from a fast identification of these circles.

Deriving Quan & Stamm [8]'s SDF, evaluations of both $P_t$ and $P_{tf}$ are achieved through sphere-tracing. This allows varying precision and quality settings, but it may also be slower than direct ray-surface intersection computation. In this case, the exterior-only molecular surface representation could represent a good fallback.

Our implementation is based on a hierarchical pipeline as it was commonly achieved in the related works, allowing dynamic allocation thanks to computed information. However, even if we improved the parallelization of the algorithm as well as the memory access coherency and allocation scheme, improvements could also be achieved by performing the computation without the need for CPU-GPU synchronization.

Finally, a mesh is often required in simulations. Thus, producing such geometry from the result of our pipeline could be interesting. This could be achieved by adapting Quan & Stamm [20]'s work on GPU.

## 8 Conclusion

In this paper, we have presented a method targeting GPU computation of the complete SES by leveraging the last theoretical advances made for its depiction. This allowed us to strongly reduce the data structure footprint on GPU dedicated memory. To alleviate its impact on the computation time, we have also proposed several additions for a better parallelization of the overall process. Our method achieves competitive time compared to the previous fastest method while being able to handle large molecular entities and without being restricted to the exterior molecular surface which was not possible before. Based on these improvements, our strategy could be used for illustration and visualization purposes as well as geometry processing, all strongly benefiting from the quality of the acquired surface. The progress made for the amelioration of the memory consumption might also support the use of analytical methods in hardware constrained applications.

## Acknowledgments

Cyprien Plateau–Holleville is supported by institutional grants from the National Research Agency under the Investments for the future program with the reference ANR-18-EURE-0017 TACTIC. Matthieu Montes is supported by the European Research Council Executive Agency under the research grant numbers 640283 and 101069190. The authors thank NVIDIA for providing a Titan RTX.

## References

[1] A. J. Olson, "Perspectives on structural molecular biology visualization: From past to present," *Journal of Molecular Biology*, vol. 430, no. 21, pp. 3997–4012, Oct. 2018. [Online]. Available: https://doi.org/10.1016/j.jmb.2018.07.009

[2] G. T. Johnson, L. Autin, M. Al-Alusi, D. S. Goodsell, M. F. Sanner, and A. J. Olson, "cellPACK: a virtual mesoscope to model and visualize structural systems biology," *Nature Methods*, vol. 12, no. 1, pp. 85–91, Dec. 2014. [Online]. Available: https://doi.org/10.1038/nmeth.3204

[3] T. Klein, L. Autin, B. Kozlíková, D. S. Goodsell, A. Olson, M. E. Gröller, and I. Viola, "Instant Construction and Visualization of Crowded Biological Environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 862–872, Jan. 2018.

[4] J. Jung, W. Nishima, M. Daniels, G. Bascom, C. Kobayashi, A. Adedoyin, M. Wall, A. Lappala, D. Phillips, W. Fischer, C.-S. Tung, T. Schlick, Y. Sugita, and K. Y. Sanbonmatsu, "Scaling molecular dynamics beyond 100, 000 processor cores for large-scale biophysical simulations," *Journal of Computational Chemistry*, vol. 40, no. 21, pp. 1919–1930, Apr. 2019. [Online]. Available: https://doi.org/10.1002/jcc.25840

[5] E. Sunden and T. Ropinski, "Efficient volume illumination with multiple light sources through selective light updates," in *2015 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, Apr. 2015. [Online]. Available: https://doi.org/10.1109/pacificvis.2015.7156382

[6] B. Johnston, "Bradyajohnston/molecularnodes: Molecularnodes v2.2.0 for blender 3.4.x," 2023. [Online]. Available: https://zenodo.org/record/7498428

[7] F. M. Richards, "AREAS, VOLUMES, PACKING, AND PROTEIN STRUCTURE," *Annual Review of Biophysics and Bioengineering*, vol. 6, no. 1, pp. 151–176, Jun. 1977. [Online]. Available: https://doi.org/10.1146/annurev.bb.06.060177.001055

[8] C. Quan and B. Stamm, "Mathematical analysis and calculation of molecular surfaces," *Journal of Computational Physics*, vol. 322, pp. 760–782, Oct. 2016. [Online]. Available: https://doi.org/10.1016/j.jcp.2016.07.007

[9] M. Krone, B. Kozlíková, N. Lindow, M. Baaden, D. Baum, J. Parulek, H.-C. Hege, and I. Viola, "Visual analysis of biomolecular cavities: State of the art," *Computer Graphics Forum*, vol. 35, no. 3, pp. 527–551, jun 2016.

[10] L. Schrödinger and W. DeLano, "Pymol," May 2020. [Online]. Available: http://www.pymol.org/pymol

[11] E. F. Pettersen, T. D. Goddard, C. C. Huang, E. C. Meng, G. S. Couch, T. I. Croll, J. H. Morris, and T. E. Ferrin, "Ucsf chimerax: Structure visualization for researchers, educators, and developers," *Protein Science*, vol. 30, no. 1, pp. 70–82, Oct. 2020. [Online]. Available: https://doi.org/10.1002/pro.3943

[12] D. Sehnal, S. Bittrich, M. Deshpande, R. Svobodová, K. Berka, V. Bazgier, S. Velankar, S. K. Burley, J. Koča, and A. S. Rose, "Mol∗ viewer: modern web app for 3d visualization and analysis of large biomolecular structures," *Nucleic Acids Research*, vol. 49, no. W1, pp. W431–W437, May 2021. [Online]. Available: https://doi.org/10.1093/nar/gkab314

[13] M. Schäfer and M. Krone, "A massively parallel cuda algorithm to compute and visualize the solvent excluded surface for dynamic molecular data," *Workshop on Molecular Graphics and Visual Analysis of Molecular Data*, 2019. [Online]. Available: https://diglib.eg.org/handle/10.2312/molva20191094

[14] M. L. Connolly, "Analytical molecular surface calculation," *Journal of Applied Crystallography*, vol. 16, no. 5, pp. 548–558, Oct. 1983. [Online]. Available: https://doi.org/10.1107/s0021889883010985

[15] B. Kozlíková, M. Krone, M. Falk, N. Lindow, M. Baaden, D. Baum, I. Viola, J. Parulek, and H.-C. Hege, "Visualization of biomolecular structures: State of the art revisited," *Wiley*, vol. 36, no. 8, pp. 178–204, Online 2016.

[16] M. F. Sanner, A. J. Olson, and J.-C. Spehner, "Reduced surface: An efficient way to compute molecular surfaces," *Biopolymers*, vol. 38, no. 3, pp. 305–320, Mar. 1996. [Online]. Available: https://doi.org/10.1002/(sici)1097-0282(199603)38:3⟨305::aid-bip4⟩3.0.co;2-y

[17] H. Edelsbrunner and E. P. Mücke, "Three-dimensional alpha shapes," *ACM Transactions on Graphics*, vol. 13, no. 1, pp. 43–72, jan 1994.

[18] M. Sanner and A. Olson, "Real time surface reconstruction for moving molecular fragments," *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, p. 385—396, 1997.

[19] M. Totrov and R. Abagyan, "The contour-buildup algorithm to calculate the analytical molecular surface," *Journal of Structural Biology*, vol. 116, no. 1, pp. 138–143, Jan. 1996. [Online]. Available: https://doi.org/10.1006/jsbi.1996.0022

[20] C. Quan and B. Stamm, "Meshing molecular surfaces based on analytical implicit representation," *Journal of Molecular Graphics and Modelling*, vol. 71, pp. 200–210, Jan. 2017. [Online]. Available: https://doi.org/10.1016/j.jmgm.2016.11.008

[21] N. Lindow, D. Baum, and H.-C. Hege, "Ligand excluded surface: A new type of molecular surface," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2486–2495, dec 2014.

[22] P. Hermosilla, M. Krone, V. Guallar, P.-P. Vázquez, À. Vinacua, and T. Ropinski, "Interactive GPU-based generation of solvent-excluded surfaces," *The Visual Computer*, vol. 33, no. 6-8, pp. 869–881, may 2017.

[23] X. Martinez, M. Krone, and M. Baaden, "Quickses: A library for fast computation of solvent excluded surfaces," *Workshop on Molecular Graphics and Visual Analysis of Molecular Data*, 2019.

[24] J. Parulek and I. Viola, "Implicit representation of molecular surfaces," in *2012 IEEE Pacific Visualization Symposium*. IEEE, Feb. 2012. [Online]. Available: https://doi.org/10.1109/pacificvis.2012.6183594

[25] S. Bruckner, "Dynamic visibility-driven molecular surfaces," *Computer Graphics Forum*, vol. 38, no. 2, pp. 317–329, May 2019. [Online]. Available: https://doi.org/10.1111/cgf.13640

[26] M. Krone, K. Bidmon, and T. Ertl, "Interactive visualization of molecular surface dynamics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1391–1398, nov 2009.

[27] M. Krone, C. Dachsbacher, and T. Ertl, "Parallel computation and interactive visualization of time-varying solvent excluded surfaces," in *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*. ACM, Aug. 2010. [Online]. Available: https://doi.org/10.1145/1854776.1854840

[28] M. Krone, S. Grottel, and T. Ertl, "Parallel contour-buildup algorithm for the molecular surface," in *2011 IEEE Symposium on Biological Data Visualization (BioVis)*. IEEE, Oct. 2011. [Online]. Available: https://doi.org/10.1109/biovis.2011.6094043

[29] D. Kauker, M. Krone, A. Panagiotidis, G. Reina, and T. Ertl, "Rendering Molecular Surfaces using Order-Independent Transparency," in *Eurographics Symposium on Parallel Graphics and Visualization*, F. Marton and K. Moreland, Eds. The Eurographics Association, 2013.

[30] A. Jurcik, J. Parulek, J. Sochor, and B. Kozlikova, "Accelerated visualization of transparent molecular surfaces in molecular dynamics," in *2016 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, Apr. 2016. [Online]. Available: https://doi.org/10.1109/pacificvis.2016.7465258

[31] M. Manak, L. Jirkovsky, and I. Kolingerova, "Interactive analysis of connolly surfaces for various probes," *Computer Graphics Forum*, vol. 36, no. 6, pp. 160–172, may 2016.

[32] T. Rau, S. Zahn, M. Krone, G. Reina, and T. Ertl, "Interactive cpu-based ray tracing of solvent excluded surfaces," *Eurographics Workshop on Visual Computing for Biology and Medicine*, 2019. [Online]. Available: https://diglib.eg.org/handle/10.2312/vcbm20191249

[33] J. C. Hart, "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, Dec. 1996. [Online]. Available: https://doi.org/10.1007/s003710050084

[34] R. Hoetzlein, "Fast fixed-radius nearest neighbors: Interactive million-particle fluids," in *GPU Technology Conference (GTC)*, 2014.

[35] J. Basselin, L. Alonso, N. Ray, D. Sokolov, S. Lefebvre, and B. Lévy, "Restricted power diagrams on the GPU," *Computer Graphics Forum*, vol. 40, no. 2, pp. 1–12, may 2021.

[36] NVIDIA, "Cuda toolkit documentation v11.6.0," 2022. [Online]. Available: https://docs.nvidia.com/cuda/archive/11.6.0/

[37] P. Gralka, M. Becher, M. Braun, F. Frieß, C. Müller, T. Rau, K. Schatz, C. Schulz, M. Krone, G. Reina, and T. Ertl, "MegaMol – A Comprehensive Prototyping Framework for Visualizations," *The European Physical Journal Special Topics*, vol. 227, no. 14, pp. 1817–1829, Mar 2019.

[38] H. M. Berman, "The protein data bank," *Nucleic Acids Research*, vol. 28, no. 1, pp. 235–242, jan 2000.

[39] J. S. D'Arrigo, "Screening of membrane surface charges by divalent cations: an atomic representation," *American Journal of Physiology-Cell Physiology*, vol. 235, no. 3, pp. C109–C117, sep 1978.

[40] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "OptiX," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1–13, Jul. 2010.

[41] C. Sigg, T. Weyrich, M. Botsch, and M. Gross, "GPU-Based Ray-Casting of Quadratic Surfaces," in *Symposium on Point-Based Graphics*, M. Botsch, B. Chen, M. Pauly, and M. Zwicker, Eds. The Eurographics Association, 2006.

[42] A. Varshney, F. Brooks, and W. Wright, "Computing smooth molecular surfaces," *IEEE Computer Graphics and Applications*, vol. 14, no. 5, pp. 19–25, sep 1994.



**Cyprien Plateau–Holleville** is a PhD candidate at the XLIM laboratory and the Université de Limoges, France. His main research interests are visualization and illustration of complex molecular systems. Plateau–Holleville received his engineering diploma in computer science from the Université de technologie de Belfort Montbéliard, France.

**Maxime Maria** is an Associate Professor at the XLIM laboratory and the Université de Limoges, France. His main research interests are visualization and interactive simulation of complex molecular structures. Maria received his PhD from the Université de Poitiers, France.

**Stéphane Mérillou** is a Professor at the XLIM laboratory and the Université de Limoges, France. His research interests include aging and weathering, physically-based rendering, natural phenomena, and visualization of complex molecular systems. Mérillou received his PhD in computer science from the Université de Limoges, France.

**Matthieu Montes** is a Professor of bioinformatics and head of the molecular modeling and drug discovery team of the GBCM laboratory at Conservatoire National des Arts et Métiers, Paris, France. His research interests include molecular modeling, drug discovery and design, interactive simulation methods and computational geometry. Montes received his PhD in pharmaceutical sciences from Paris Descartes University and a habilitation in structural biochemistry from Paris Sud University. In 2014 and 2022, he was a fellow of the European Research Council. He is a senior member of the Institut Universitaire de France (IUF).