



**HAL**  
open science

## QuickSES: A Library for Fast Computation of Solvent Excluded Surfaces

Xavier Martinez, Michael Krone, Marc Baaden

► **To cite this version:**

Xavier Martinez, Michael Krone, Marc Baaden. QuickSES: A Library for Fast Computation of Solvent Excluded Surfaces. MolVa: Workshop on Molecular Graphics and Visual Analysis of Molecular Data 2019, Jun 2019, Porto, Portugal. 10.2312/molva.20191095 . hal-02370900

**HAL Id: hal-02370900**

**<https://hal.science/hal-02370900v1>**

Submitted on 27 Sep 2020

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# QuickSES: A Library for Fast Computation of Solvent Excluded Surfaces

Xavier Martinez (1), Michael Krone (2) and Marc Baaden (1)

(1) Laboratoire de Biochimie Théorique, CNRS, UPR9080, Univ Paris Diderot, France

(2) Big Data Visual Analytics, University Tübingen, Tübingen, Germany

---

## Abstract

Recently, several fast methods to compute Solvent Excluded Surfaces (SES) on GPUs have been presented. While these published methods reportedly yield interesting and useful results, up to now no public, freely accessible implementation of a fast and open-source SES mesh computation method that runs on GPUs is available. Most molecular viewers, therefore, still use legacy CPU methods that run only on a single core, without GPU acceleration. In this paper, we present an in-depth explanation and a fully open-source CUDA implementation of the fast, grid-based computation method proposed by Hermosilla *et al.* [HKG\*17]. Our library called QuickSES runs on GPUs and is distributed with a permissive license. It comes with a standalone program that reads Protein Data Bank (PDB) files and outputs a complete SES mesh as a Wavefront OBJ file. Alternatively it can directly be integrated in classical molecular viewers as shared library. We demonstrate the low memory consumption to enable execution on lower-end GPUs, and compare the runtime speed-up to available state-of-the-art tools.

## CCS Concepts

• **Applied computing** → **Molecular structural biology**; • **Computing methodologies** → **Massively parallel algorithms**;

---

## 1. Introduction

Molecular surface representations form a core part of standard molecular viewers to visualize molecules. They ease the perception of molecular interactions by providing a visual representation of the accessibility and complementarity of molecules. Therefore, molecular surfaces are an essential tool for the analysis of complex molecular structures.

Several types of surfaces depict different molecular properties: the Van der Waals (VDW) surface yields an approximation of the space that atoms are taking, the solvent accessible surface (SAS) represents the volume taken by the molecule and a hypothetical solvent layer, but only the solvent excluded surface (SES) really provides a sense of the molecular shape as "seen" by the solvent or a ligand prone to bind to it. The proposed method discussed here focuses on fast SES computation, for instance to render dynamic datasets in interactive time.

### 1.1. State of the Art

Initially, the SES was defined as the surface obtained by rolling a probe sphere on the atoms [Ric77]. An analytical solution has been proposed [Con83], but these methods tend to be slow and may prove hard to parallelize. On the other hand, grid-based methods provide a coarser solution but can take advantage of multicore CPUs and massively parallel GPU architectures.

Two largely used available tools to compute a mesh representation of the SES are MSMS [SOS96] and EDTSurf [XZ09]. MSMS

is an analytical method that computes the patches of the SES and triangulates them afterwards. It speeds up the analytical computations using the so-called reduced surface algorithm, but runs only on a single core of the CPU. Attempts to parallelize this algorithm on GPUs had only little success [KDE10], as it is inherently sequential. MSMS is still used by many molecular viewers, for example the popular and freely available VMD [HDS96] or UCSF Chimera [PGH\*04]. EDTSurf is a grid-based method implemented on CPUs that can generate SES meshes at different resolutions. Similar to MSMS, EDTSurf takes multiple seconds to generate the SES of a larger biomolecule. A way of explaining why these tools are still widely used is that they output a standard mesh file.

Several GPU implementations have been proposed to interactively compute SES surfaces [KBE09, KGE11, PV12]. QuickSurf [KSES12] is an example of a fast GPU method to compute an isosurface based on a Gaussian density map, using the CUDA language. It has been successfully integrated in VMD [HDS96] and MegaMol [GKM\*15]. However, extracting the implementation from these tools is not straightforward and it provides only an approximation of the SES, not the exact surface. Recently, Hermosilla *et al.* [HKG\*17] presented a GPU-parallelized method that computes the SES on a grid, similar to EDTSurf. Their implementation makes use of a progressive grid refinement to achieve interactive frame rates even for large, dynamic data. For rendering, GPU volume ray marching (i.e., direct volume rendering) is used instead of extracting a triangle mesh. A detailed survey of visualizations for molecular data, including an extensive discussion of methods to

compute and visualize the SES and other molecular surfaces, was recently presented by Kozlíková *et al.* [KKF\*17].

## 1.2. Motivation and Contribution

While the methods discussed in subsection 1.1 have been published and reportedly yield interesting results, a public, freely accessible open-source implementation of such a fast SES mesh computation method that runs on GPUs and outputs a mesh usable outside of a molecular viewer is not currently available.

In this paper, we explain a CUDA implementation of Hermosilla *et al.*'s [HKG\*17] grid-based SES computation method. The corresponding tool named *QuickSES* is published under a permissive license, runs all computations on the GPU, and outputs a complete SES mesh as Wavefront OBJ files. An example of the SES computed with QuickSES is shown in Figure 3. It can be executed as a standalone program that reads Protein Data Bank (PDB) files, or can be called from classical molecular viewers as a shared library.

## 2. Method and Implementation Details

The algorithm is extensively described in the original paper by Hermosilla *et al.* [HKG\*17], therefore we will focus only on the specific details of our CUDA implementation. To generate the SES, the method requires to compute a signed distance field (named *SES grid*) with positive values outside the SES and negative ones inside. To speed up parts of the computation, a second grid is used to access neighboring atoms in constant time.

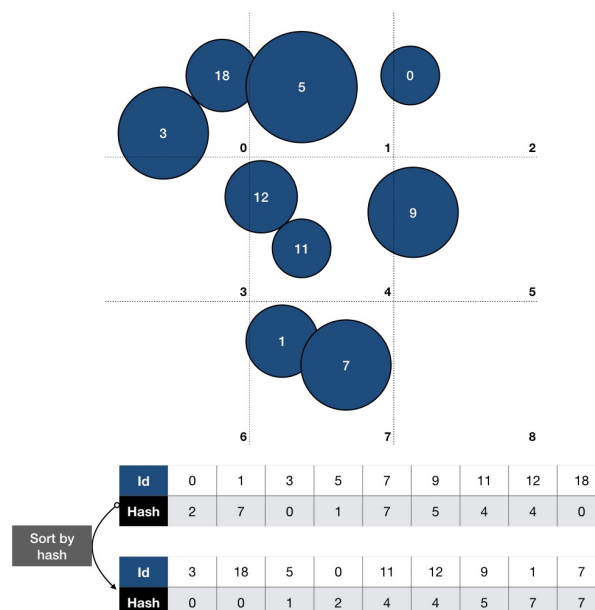
To emulate the rolling of the probe sphere, usually with a 1.4 Å radius representing water, the method is composed of 4 main steps: (1) computing the neighbor grid, (2) assigning voxels inside, at the border or outside of the molecule, (3) processing the "border" voxels to refine the distance field, and (4) transforming the distance field into a mesh using the Marching Cubes algorithm [LC87a].

### 2.1. Neighbor Search

Fast access to an atom's neighbors is a key point of this method. Indeed, to classify atoms and to efficiently compute the distances at the border in the third step, a constant access time is mandatory to achieve acceptable performance.

QuickSES implements a standard uniform 3D grid around the molecule, fully executed on the GPU [Gre10]. The resolution of this grid is given by the size of the probe radius ( $r_{probe}$ ) plus the maximum Van der Waals atom radius of the molecule ( $r_{max}$ ):  $R_{neighbor} = r_{probe} + r_{max}$ .

A *hash* value is assigned to each atom by converting its 3D index position in the neighbor grid into its 1D index. The *hash* is stored together with the index (*id*) of the atom in the array of positions, this array is called *hashIndex*. This procedure effectively assigns each atom to a neighbor cell. After sorting the *hashIndex* array by the *hash* value (using the Thrust library [BH12]), we obtain the distribution of atoms per grid cell (cf Figure 1). To improve coalesced access of memory, an array of atom position sorted by *hash* is also maintained. To count the number of atoms per grid cell, we record the position of the first and last *hash* in the *hashIndex* array.



**Figure 1:** Uniform 3D grid for fast neighbor access : each atom is assigned to a cell. Sorting the array of index *id* and *hash* by *hash* yields a simple data structure to query atoms in a cell.

We now have a data structure enabling us to fetch neighboring information in constant time. The relatively small construction time of this uniform grid is outweighed by the gains during the subsequent steps.

Instead of using the radix sort implementation from the Thrust library, a speed improvement has been proposed by Hoetzlein [Hoe14], who achieved significantly lower computation times using atomic operations in a count sort. However, since this is not the limiting step in our implementation, we left this optimization as future work.

### 2.2. Probe Intersection and Voxel Classification

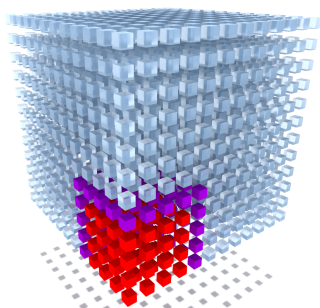
The goal of this step is to determine the voxels of the *SES grid* that are at the boundary of the surface and to quickly classify other voxels as **inside** or **outside** the SES. The SES grid encompasses the whole protein and has a user-defined resolution, which is determined by the size of one voxel  $V_{SES}$  (which is given in Å). Smaller  $V_{SES}$  result in a higher quality of the final SES, but also lead to a higher computation time.

- A voxel is **inside** the molecule if the distance between its center and at least one atom is less than the radius of this atom plus the voxel size  $V_{SES}$ . The voxel value is set to  $-V_{SES}$ .
- A voxel is at the **boundary** of the SES if the distance between this grid point and an atom is less than the atom radius minus the voxel size  $V_{SES}$ . The voxel is temporarily set to 0.0 and will be processed in the next step.
- Otherwise, the voxel is **outside** the molecule, meaning that there is no atom close to this voxel (the atom-voxel distance is more than the atom radius  $+V_{SES}$ ). The voxel value is set to  $r_{probe}$ .

To keep track of voxels at the border, we record the 1D index of this cell inside an array the size of the sub-grid. If the voxel is inside or outside the SES, we record a large number instead. We then sort this array and count the number of cells at the border.

As the user defines the *SES grid* resolution, the size of this grid can be very large (more than  $500^3$  voxels, resulting in more than 500 MB of memory). Depending on the GPU specifications, allocating the full *SES grid* in GPU memory is not always possible. To avoid this memory limitation, we only allocate a subset of this grid, usually a  $300^3$  cube and we process all the following steps for each subset (Figure 2). To process the full grid, we have to offset this sub-grid in all directions by  $offset = \left\lceil \frac{r_{probe}}{V_{SES}} \right\rceil$ , leading to a total of  $\left\lceil \frac{gridSize}{(sliceSize - offset)} \right\rceil^3$  sub-grids processed.

The user may define the size of the slice to speed this step up on GPUs with more memory. The slice can then cover the whole grid. On the contrary, the user can decrease the sub-grid size to lower memory consumption and run QuickSES on lower-end GPUs.



**Figure 2:** Compute a  $4^3$  subset (red) of the  $12^3$  grid. The subset of the grid needs to access a larger part in the distance refinement step (represented here in purple) that has to be computed in the probe intersection step.

### 2.3. Distance Refinement

As stated by Hermosilla *et al.* [HKG\*17], the limiting step of the algorithm is clearly the distance refinement step for large grid sizes. For each voxel found at the border of the SES in the previous step, we look for the closest voxel that is outside of the SES, in a cube of size  $offset \times 2$  cells around the current cell. If no such voxel is found, the value in the *SES grid* is set as **inside** the SES. Otherwise, the value is set to  $r_{probe} - minDist$  where  $minDist$  is the minimum distance found.

To avoid any issue with the watchdog timer that controls the execution time of a CUDA kernel, the kernel is split into several smaller ones. Thereby, we bypass processing all the voxels during a single kernel, which could take more than the 5 seconds time budget. In order to maximize high-end GPU usage, CUDA streams start these sub-kernels asynchronously.

This step bears a high probability of a cache miss, as not all voxels are always aligned. The lower the resolution of the *SES grid*, the larger this cube, increasing the execution time for this step.

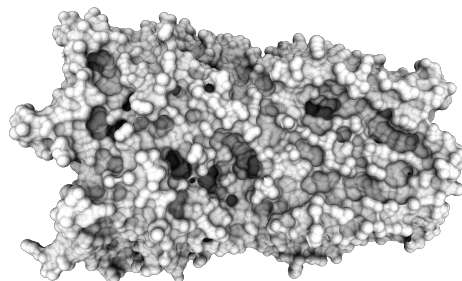
### 2.4. Marching Cubes on GPU

Now that the distance field is correctly computed, a GPU-parallelized variant of the Marching Cubes (MC) algorithm [LC87b] is executed on the sub-grid to triangulate a surface mesh. Our implementation is similar to the example provided with the Nvidia GPU Computing Toolkit. The first step outputs the number of vertices computed for each cell of the sub-grid. A prefix sum counts the overall number of vertices for the mesh to be allocated, thereby limiting the size of the data to transfer from the GPU to the CPU. The actual MC step can then be executed on the relevant voxels only.

To further reduce the size of the vertex array, we added a welding vertex step that consists in rounding the vertices to a lower precision, sorting the array based on the  $x$  values, then on the  $y$  values, then on the  $z$  values. We remove duplicates keeping track of the new order of the vertices, which is needed to connect the vertices using triangles. The Thrust library provides a set of functions to efficiently execute these steps.

The classical edge and triangle tables are stored in constant memory arrays for cached access during the first and the last step of the MC GPU implementation. Once the mesh is returned to the CPU, it is smoothed using a Laplacian step as the mesh can be coarse at high voxel size (small box size). At last, once all sub-grids have been processed, we merge the meshes of each subset into a larger mesh by offsetting the triangle indices.

## 3. Results



**Figure 3:** Solvent Excluded Surface of the PDB 3EAM computed with QuickSES with a  $0.15 \text{ \AA}$  grid resolution (ambient occlusion of the generated OBJ file computed in MeshLab [CCC\*08]).

### 3.1. Interactive molecular dynamics trajectory rendering

We successfully integrated QuickSES in UnityMol [LTDS\*13] (*cf.* Supp. Mat. Video) to test its suitability for interactive rendering of molecular dynamics trajectories in surface representation. Furthermore, this test suggests that QuickSES could be used similarly easily in any other visualization software. As standard mesh files can be generated as well, it is straightforward to post-process them as illustrated in Figure 3.

**Table 1:** Time (in ms) and memory consumption (in brackets; peak in MB) for different SES computation methods, averaged over 5 executions. The probe radius was set to 1.4 Å. Note that we did not count the time spent writing the mesh to disk. MSMS was executed with a density of 10, and no area computed. QuickSES voxel size was set to 0.5 Å and the slice size was the default value of 300<sup>3</sup> voxels. Tests were run on a 1080Ti GPU and a i7-6700 3.4Ghz CPU. MSMS fails to compute large molecules (1GRU and 4V59).

PDB ID	#atoms	EDTSurf	MSMS	QuickSES
1CRN	327	746 (45)	88 (5)	333 (10)
1KX2	1,249	1403 (77)	188 (10)	340 (20)
3EAM	13,505	4586 (280)	1743 (90)	627 (439)
1GRU	58,688	5939 (317)	–	1888 (805)
4V59	169,748	6881 (427)	–	2893 (695)

### 3.2. Speed and memory benchmark

The execution times and memory usage are detailed in Table 1. QuickSES is always faster than EDTSurf. The resolution of the grid for QuickSES is set to 0.5 Å meaning that for large molecules (more than 10k atoms), the resolution of the mesh is higher than for EDTSurf. Indeed, by default, EDTSurf lowers the resolution of the mesh to fit the molecule inside the box.

It should be noted that using QuickSES as library will reduce the execution time as no parsing of the input file, nor mesh writing will be necessary. One could also use the mesh directly from the GPU without copying it to the system memory.

#### 3.2.1. Impact of slicing the SES grid

Besides the resolution and the number of atoms, the size of the sub-grid impacts the performance of our method. As expected (*cf.* Supp. Mat. Table 1), if the total grid fits inside the slice, we can compute the SES slightly faster. The impact on the performance is limited (it is  $\approx 30\%$  faster to run QuickSES on PDB-id 3eam with a sub-grid of 450<sup>3</sup> voxels rather than with a box of 200<sup>3</sup> voxels).

### 4. Conclusion

We propose an efficient, open-source GPU implementation of the SES mesh computation algorithm by Hermosilla *et al.* [HKG\*17], available at (Link to GitHub repository removed for double-blind review). Our QuickSES can be used as a standalone executable to parse PDB files and output OBJ mesh files. We also provide a simple way to use it as a shared library and integrate it into already existing visualization softwares, providing access to interactive SES rendering.

An effort has been made to lower the memory footprint, in order to make it possible to tackle large structures even with a modest GPU, *i. e.* with less than 512 MB of GPU RAM.

#### 4.1. Future work

The execution speed may further be increased by optimizing CUDA kernels and by using shared memory more extensively, in

particular during the stencil operations done in the probe intersection step and the distance refinement step. Another area of improvement could be to process several voxels per GPU thread to do more work and hide the memory access latency.

As a complete distance field is computed, we could compute smooth mesh normals during the MC step. Interestingly, one of the only steps currently run on the CPU is the Laplacian mesh smoothing. For large meshes, this step can take a significant amount of time. A GPU implementation of a mesh smoothing algorithm could thus provide a speed-up for this part. The current implementation is only computing the SES but can easily compute the VDW surface or the SAS instead.

Finally, to target a broader spectrum of hardware, we plan to release an OpenCL implementation of QuickSES to be able to use it on AMD and Intel GPUs, as well as to take advantage of multicore CPUs.

### References

- [BH12] BELL N., HOBEROCK J.: Thrust: A productivity-oriented library for cuda. In *GPU computing gems Jade edition*. Elsevier, 2012, pp. 359–371. 2
- [CCC\*08] CIGNONI P., CALLIERI M., CORSINI M., DELLEPIANE M., GANOVELLI F., RANZUGLIA G.: MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference (2008)*, Scarano V., Chiara R. D., Erra U., (Eds.), The Eurographics Association. 3
- [Con83] CONNOLLY M. L.: Analytical molecular surface calculation. *Journal of applied crystallography* 16, 5 (1983), 548–558. 1
- [GKM\*15] GROTTTEL S., KRONE M., MÜLLER C., REINA G., ERTL T.: Megamol—A prototyping framework for particle-based visualization. *IEEE transactions on visualization and computer graphics* 21, 2 (2015), 201–214. 1
- [Gre10] GREEN S.: Particle simulation using cuda. *NVIDIA whitepaper* 6 (2010), 121–128. 2
- [HDS96] HUMPHREY W., DALKE A., SCHULTEN K.: Vmd: visual molecular dynamics. *Journal of molecular graphics* 14, 1 (1996), 33–38. 1
- [HKG\*17] HERMOSILLA P., KRONE M., GUALLAR V., VÁZQUEZ P.-P., VINACUA À., ROPINSKI T.: Interactive gpu-based generation of solvent-excluded surfaces. *The Visual Computer* 33, 6-8 (2017), 869–881. 1, 2, 3, 4
- [Hoe14] HOETZLEIN R.: Fast fixed-radius nearest neighbors: interactive million-particle fluids. In *GPU Technology Conference (2014)*, vol. 18. 2
- [KBE09] KRONE M., BIDMON K., ERTL T.: Interactive visualization of molecular surface dynamics. *IEEE transactions on visualization and computer graphics* 15, 6 (2009), 1391–1398. 1
- [KDE10] KRONE M., DACHSBACHER C., ERTL T.: Parallel Computation and Interactive Visualization of Time-varying Solvent Excluded Surfaces. In *Proc. of ACM International Conference on Bioinformatics and Computational Biology (2010)*, vol. 1, pp. 402–405. doi:10.1145/1854776.1854840. 1
- [KGE11] KRONE M., GROTTTEL S., ERTL T.: Parallel contour-buildup algorithm for the molecular surface. In *2011 IEEE Symposium on Biological Data Visualization (BioVis)*. (2011), IEEE, pp. 17–22. 1
- [KKF\*17] KOZLÍKOVÁ B., KRONE M., FALK M., LINDOW N., BAADEN M., BAUM D., VIOLA I., PARULEK J., HEGE H.-C.: Visualization of Biomolecular Structures: State of the Art Revisited. *Computer Graphics Forum* 36, 8 (2017), 178–204. doi:10.1111/cgf.13072. 2



- [KSES12] KRONE M., STONE J. E., ERTL T., SCHULTEN K.: Fast visualization of gaussian density surfaces for molecular dynamics and particle system trajectories. *EuroVis-Short Papers 2012* (2012), 67–71. [1](#)
- [LC87a] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics* (1987), vol. 21, ACM, pp. 163–169. [2](#)
- [LC87b] LORENSEN W. E., CLINE H. E.: Marching Cubes: A High Resolution 3d Surface Construction Algorithm. In *ACM SIGGRAPH Computer Graphics and Interactive Techniques* (1987), vol. 21, pp. 163–169. [doi:10.1145/37402.37422](https://doi.org/10.1145/37402.37422). [3](#)
- [LTDS\*13] LV Z., TEK A., DA SILVA F., EMPEREUR-MOT C., CHAVENT M., BAADEN M.: Game on, science-how video game technology may help biologists tackle visualization challenges. *PLoS one* 8, 3 (2013), e57990. [3](#)
- [PGH\*04] PETERSEN E. F., GODDARD T. D., HUANG C. C., COUCH G. S., GREENBLATT D. M., MENG E. C., FERRIN T. E.: UCSF Chimera - A Visualization System for Exploratory Research and Analysis. *Journal of Computational Chemistry* 25, 13 (2004), 1605–1612. [1](#)
- [PV12] PARULEK J., VIOLA I.: Implicit representation of molecular surfaces. In *2012 IEEE Pacific Visualization Symposium* (2012), IEEE, pp. 217–224. [1](#)
- [Ric77] RICHARDS F. M.: Areas, Volumes, Packing, and Protein Structure. *Annual Review of Biophysics and Bioengineering* 6, 1 (1977), 151–176. [doi:10.1146/annurev.bb.06.060177.001055](https://doi.org/10.1146/annurev.bb.06.060177.001055). [1](#)
- [SOS96] SANNER M. F., OLSON A. J., SPEHNER J.-C.: Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers* 38, 3 (1996), 305–320. [1](#)
- [XZ09] XU D., ZHANG Y.: Generating triangulated macromolecular surfaces by euclidean distance transform. *PLoS one* 4, 12 (2009), e8140. [1](#)