Workshop on Molecular Graphics and Visual Analysis of Molecular Data (MolVA) (2019)
J. Byška, M. Krone, and B. Sommer (Editors)

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, Sorbonne Paris Cité, PSL Research University, 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*** $\rightarrow$ ***Molecular structural biology;*** •***Computing methodologies*** $\rightarrow$ *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 half a hypothetical solvent layer, but only the solvent excluded surface (SES) really provides a sense of the molecular shape as "seen" by an approaching molecule, be it 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] of the molecule under investigation. An analytical solution has been proposed [Con83], which offers improved precision and topological correctness, but these methods tend to be slow and some may prove hard to parallelize. Grid-based methods provide a coarser solution but offer specific advantages: the resolution can be adapted to address performance constraints, the results can be used for further computation of the molecular volume or the extraction of cavities, one can easily take advantage of multicore CPUs as well as massively parallel GPU architectures, and meshes can be generated in an easy way, allowing further processing by mesh-centric tools, for instance Maya and Unity3D.

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 and is not suitable for interactively computing surfaces during a trajectory. 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]. VMD provides SURF [VBW94], an intrinsically parallel alternative to MSMS, yet its distributed implementation is slower than the serial MSMS one. 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

DOI: 10.2312/molva.20191095

way of explaining why these tools are still widely used is that they output a standard mesh file.

Several CPU and GPU implementations have been proposed to interactively compute SES surfaces, for example parallelized variants of the Contour-Buildup algorithm [LBPH10,KGE11,JPSK16], which was originally devised by Totrov and Abagyan [TA96]. Another approach by Parulek et al. [PV12] tries to model the whole SES implicitly and uses GPU ray casting for rendering. 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 to calculate the surface mesh is not straightforward. 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. Our motivation to implement it relates to specific needs in extending our UnityMol [LTDS*13] molecular viewer software with an efficient mesh-generation of the SES that can be tuned for interactivity, for instance by starting with a coarse visualization that is refined on the fly, as well as to compute quantities such as the molecular volume or extract cavities from the grid used for the SES calculation.

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 of a pentameric ligand-gated ion channel membrane protein computed with QuickSES is shown in Figure 1. 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

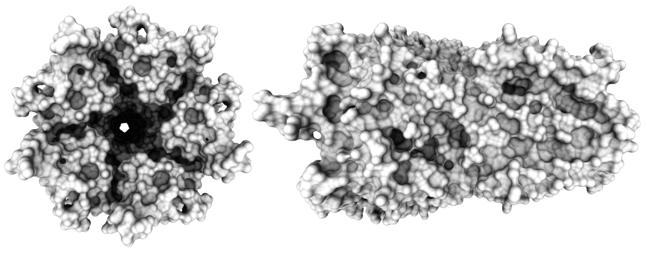


Figure 1: *Solvent Excluded Surface of the Glic ion channel (PDB-id 3EAM) computed with QuickSES with a 0.15 Å grid resolution (ambient occlusion of the generated OBJ file computed in MeshLab [CCC*08]).*

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 [LC87].

2.1. Neighbor Search

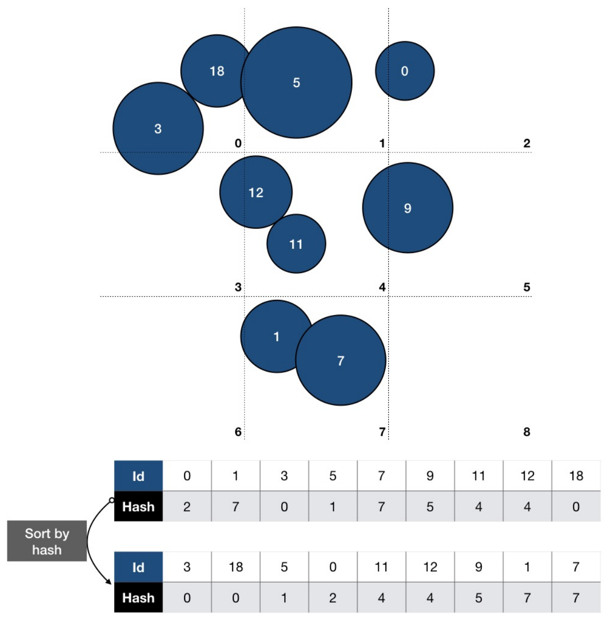
Fast access to an atom's neighbors is a key point of this method. Indeed, to classify atoms and to efficiently compute the distances at the border in the third step, a constant access time is mandatory to achieve acceptable performance.

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 2). 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.

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 computational 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.



Id	0	1	3	5	7	9	11	12	18
Hash	2	7	0	1	7	5	4	4	0

Id	3	18	5	0	11	12	9	1	7
Hash	0	0	1	2	4	4	5	7	7

Figure 2: *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.*

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 **bondary** 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 exceeds 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, by default a 300^3 cube and we process all the following steps for each subset (Figure 3). To process the full grid, we have to enlarge this sub-grid in all directions by $offset = \left\lceil \frac{r_{probe}}{V_{SES}} \right\rceil$.

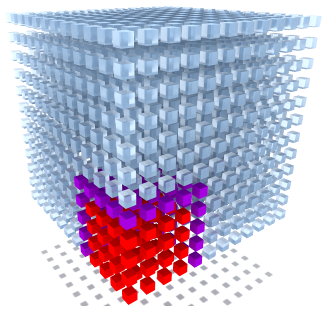


Figure 3: *Compute a 4^3 subset (red) of the 12^3 grid. During the distance refinement step, the method needs to access a larger part of the subset (represented here in purple) that has to be computed in the probe intersection step and that is ignored in the marching cubes step.*

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.

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 hypothetically 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 algorithm [LC87] 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 marching cubes 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 marching cubes 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 value of grid resolution. 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.

Table 1: *Time (in ms) and memory consumption (in parentheses; 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 count the time spent parsing the input file, but not writing the mesh to disk to have identical conditions for all three softwares. 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* 300^3 *voxels. Tests were run on a 1080 Ti GPU and a i7-6700 3.4Ghz CPU. MSMS fails to compute the two largest test 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 (*440*)
1GRU	58,688	5939 (*317*)	–	1888 (*798*)
4V59	169,748	6881 (*427*)	–	3988 (*746*)

3. Results and Discussion

3.1. Interactive rendering of molecular dynamics trajectories

We successfully integrated QuickSES in UnityMol (*cf.* Supp. Mat. Video) to test it's 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 1.

3.2. Speed and memory benchmark

The execution times and memory usage are detailed in Table 1. QuickSES is faster than EDTSurf and MSMS for larger molecules. 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. Please note that other GPU-based approaches to compute the SES (e.g., [KGE11, JPSK16] can also compute the SES of moderately large molecules interactively; however, they do not compute the surface meshes but rather use GPU ray casting for rendering, which makes it hard to re-use the surface in other software.

3.2.1. Impact of slicing the SES grid

Besides the resolution and the number of atoms, the size of the subgrid 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 450^3 voxels rather than with a box of 300^3 voxels).

4. Conclusion and Future Work

We propose an efficient, open-source GPU implementation of the SES mesh computation algorithm by Hermosilla *et al.* [HKG*17]. The source code of our implementation is available on GitHub: `http://tiny.cc/QuickSES`.

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 computation and rendering for moderately large molecules on current Nvidia desktop GPUs.

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.

There are several directions in which our current QuickSES implementation could be improved in the future. First of all, 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 marching cubes 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. 2

[Con83] Connolly M. L.: Analytical molecular surface calculation. *Journal of applied crystallography 16*, 5 (1983), 548–558. 1

[GKM*15] Grottel 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. 2

[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, 2

[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

[JPSK16] Jurčík A., Parulek J., Sochor J., Kozlikova B.: Accelerated visualization of transparent molecular surfaces in molecular dynamics. In *2016 IEEE Pacific Visualization Symposium (PacificVis)* (2016), IEEE, pp. 112–119. 2, 4

[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., Grottel 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. 2, 4

[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. 2

[LBPH10] Lindow N., Baum D., Prohaska S., Hege H.-C.: Accelerated visualization of dynamic molecular surfaces. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 943–952. 2

[LC87] 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. 2, 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. 2

[PGH*04] Pettersen 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. 2

[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. 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

[TA96] Totrov M., Abagyan R.: The contour-buildup algorithm to calculate the analytical molecular surface. *Journal of structural biology 116*, 1 (1996), 138–143. 2

[VBW94] Varshney A., Brooks F., Wright W.: Linearly scalable computation of smooth molecular, invited submission. *IEEE Computer Graphics and Applications* (1994). 1

[XZ09] Xu D., Zhang Y.: Generating triangulated macromolecular surfaces by euclidean distance transform. *PloS one 4*, 12 (2009), e8140. 1