CUDA Accelerated Blobby Molecular Surface Generation

Daniele D'Agostino[1], Sergio Decherchi[2], Antonella Galizia[1], José Colmenares[2], Alfonso Quarati[1], Walter Rocchia[2] and Andrea Clematis[1]

[1] Institute for Applied Mathematics and Information Technologies, National Research Council of Italy, Genoa, Italy
[2] Department of Drug Discovery and Development, Italian Institute of Technology Genoa, Italy

Abstract. A proper and efficient representation of molecular surfaces is an important issue in biophysics from several view points. Molecular surfaces indeed are used for different aims, in particular for visualization, as support tools for biologists, computation, in electrostatics problems involving implicit solvents (e.g. while solving the Poisson-Boltzmann equation) or for molecular dynamics simulations. This problem has been recognized in the literature, resulting in a multitude of algorithms that differ on the basis of the adopted representation and the approach/technology used. Among several molecular surface definitions, the blobby surface is particularly appealing from the computational and the graphics point of view. In the paper we describe an efficient software component able to produce high-resolution blobby surfaces for very large molecules using the CUDA architecture. Experimental results shows a speedup of 35.4 considering a molecule of 91,000 atoms and a resulting mesh of 168 million triangles.

Keywords: Blobby Molecular Surface, GPU Computing

1 Introduction

Molecular surface computation is a key issue from at least two perspectives: from both the visualization and the biophysical computation view point.

In the first case a user is interested in the overall rendering quality of the molecular model: classical paradigms triangulate the surface and then visualize the mesh [6]. Furthermore, the use of present Graphic Processing Units (GPU) capabilities allows the direct rendering of quadrics patches by ray tracing [7].

From the biophysical stand point a user is interested in a molecular surface that is able to capture the physical problem at hand and that is computationally efficient. An example of such use case is the Poisson-Boltzmann equation (PB), where the electrostatic potential of a molecule (solute) in water (solvent) is sought. In particular in PB it is usually accepted that the Van Der Waals surface (VDW) should not be directly used: to solve this problem the Solvent Excluded Surface (SES) (or Connolly surface) was introduced [8] (an algorithm for calculation was given in [9]) which expands the VDW surface by smothing its concave parts with spherical patches that represent the rolling of a water molecule (the *probe*) over the surface. The Richards model has been largely used in the solution of PB [11], however it still presents some limits [10]: among them, the fact that it leads to a non differentiable surface is of particular importance.

This issue calls for alternative surface models that retain a physically sound definition and overcome SES limitations. Whatever is the final goal, either visualization or biophysical computations, some properties of the molecular surface are desiderable. Among them the computational efficiency, the differentiability everywhere and a good performance scalability for large molecular models. To this aim the Blobby surface [12][13] represents an interesting alternative to the SES surface.

In this paper we describe an efficient algorithm, based on the isosurface extraction, able to produce high-resolution blobby surfaces for very large molecules using the CUDA architecture. Our aim was to design the algorithm able to act as a software component producing an output suitable for both the direct visualization and the efficient storage of the surface. In fact the possibility of the direct use of the produced mesh without any preprocessing step is of particular importance for the performance view points if the surface have to be further processed in a biophysical worflow.

The following Sections respectively present: related work, the definition of the Blobby surface, the CUDA accelerated algortihm for the generation of the Blobby surface and experimental results. At the end some conclusions are drawn.

2 Related Work

Molecular surface computation is a long standing problem. Among the others, we mention the most used ones: the Connolly surface [9], the *Skin* surface [14], and the Blobby surface [12].

From the computational point of view, the recent emerging availability of relatively inexepensive GPU systems stimulated the research on the usability of GPU devices to accelerate the molecular surface processing and visualization.

In [7], it is shown how GPU can be used to ray trace the *Skin* surface [14]. The Skin surface in fact is composed by a set of quadric patches (i.e. spheres and hyperboloids), each of them bounded by a solid of the mixed complex [14]. This allows to use OpenGL shading language for the real time rendering. In [15] it is shown how both the SES and the Skin surfaces can be built in parallel on CPU and effectively rendered on the GPU by ray tracing, obtaining high frame rates; analogous considerations hold for the Connolly [15].

These works aim exclusively at improving the final rendering phase, while we are interested in exploiting the GPU computing capabilities also in the generation of the surface, in a format that is able to suit both the visualization and further processing steps in a biophysical analysis worflow.

A similar approach is followed in [1], where a parallel workflow for the extraction of SES surfaces is described. It is based on the construction of the volumetric dataset starting from the atomic coordinates of the atoms that form the molecule, which is then processed using the isosurface extraction operation to produce the SES as a mesh of triangles. The main drawback of this work is that it do not considers the use of GPU devices.

Such aspect is addressed in [2], where an approach similar to the previous one is implemented for the CUDA architecture and extended to build smooth molecular surfaces. In particular this work considers also the creation of the Blobby surfaces but, as the previous ones, disregards the aspect related to the effective storing of the produced meshes.

3 Blobby Surface Definition

The blobby surface S [12] [13] is defined as:

$$S := \{\mathbf{x} \in R^3 : G(\mathbf{x}) = 1\}, \;\; G(\mathbf{x}) = \sum_{i=1}^{n_a} e^{B\left(\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{r_i^2}-1\right)} \tag{1}$$

where r_i are the radii of atoms, n_a is the number of atoms, $\mathbf{x}$ is the query point, $c(\mathbf{x}_i)$ is the $i-$th atom center and B is a negative parameter (the *blobbyness*) that plays the role of the probe radius when compared to the Connolly surface [9].
Blobby surface has some salient pros and cons from both the computational and physically soundeness point of view: first the surface is easy to implement because the central computation is simply an evaluation of a kernel function. The surface is also tangent continuous and is self-intersections and singularities free. From the physical model point of view it is not completely clear if this surface is superior to the SES [9] when solving, for instance, electrostatics problems. Indeed it can be argued that the right setting of the blobbyness value B is a key parameter in order to obtain reliable energy estimations of molecular systems. Another point is that the surface it is not partioned in analytical patches as in the skin [14] or in the Solvent Excluded Surface [9]. Additionally some values of B can modify the size of the atoms leading to non physically acceptable surfaces as observed in [16]. Despite these drawbacks the implicit models are becoming rather used [17] when dealing with biophysical problems mainly because of the smoothness nature of the surface and because gaussian functions mimick model electronic density functions.
This surface not only is continuous and differentiable everywhere but also its computation can be (apparently) trivially parallelized in at least two ways; first each scalar field value $G(\mathbf{x})$ can be computed independently from any other element; secondly every kernel evaluation (the exponential) in the for loop can be carried in parallel. Due to this explicit parallel nature the surface computation can be effectively parallelized. In the following it will be discussed how to exploit this parallelizzability on GPU and experimental results will be shown.

4 CUDA Accelerated Blobby Surface Generation

The high resolution representation of molecules is a key aspect for their satisfactory visualization and also for the effectiveness of analysis operations, but their modeling is a costly process that may require several minutes. This is the reason why we implemented a parallel algorithm for the generation of the Blobby surface. In particular we made use of the CUDA architecture, that represents a cost-effective solution for many compute-intensive applications on regular domains.

Following the CUDA naming convention, we define *host* the workstation and *device* the Nvidia card providing the GPU of which CUDA is the computing engine. The algorithm we propose is based on two main operations, the Scalar Field Generation and the Isosurface Extraction, both performed on the device with a minimal amount of data transfer with the host.

The main input of the system is a PDB file containing the coordinates of the atoms that form a molecule, while the resulting isosurface is represented by a triangulated mesh. The output format of the isosurface, that is the coordinates of the vertices and the triangle/vertex incidence relation, suits both the direct visualization and the efficient storing for following processing steps, because these two data structures allow to reconstruct all the incidence relations among the elements of a triangulated irregular network.

4.1 Scalar Field Generation

The first operation of the algorithm is the generation of the three-dimensional grid containing the volumetric data representing the molecule. The size of the grid is determined on the basis of the coordinates of the atomic centers and on the required sampling step. Typical step values are chosen between 0.7 and 0.1 Å, according to the desired level of resolution. Smaller step values correspond to dense grids and high resolution surfaces, and vice versa. Within the grid, atoms are modeled as spheres having different radii.

The grid is usually considered as a set of XY planes, called *slices*. As the amount of memory of a device is limited, and the isosurface extraction operation requires to process a pair of slices at a time, we implemented this and the following operation in an iterative way for increasing values of the Z coordinate. This means that one slice is created at each iteration (except for the first one, where the slices for Z=0 and 1 are created) in order to replace the slice having the lowest Z value. In this way we are able to process very large data sets if the size of a pair of slices does not exceed the device memory.

The value of a grid point is the result of the influence of all the atoms on it. For large molecules (e.g. 10^5 atoms) this translates to considering several million points. The present CUDA architecture limits the number of threads (i.e. up to 1024 threads for a block and up to a grid of 65535x65535x1 blocks), and this means that is not possible to generate a thread for each pair atom-point. Therefore we have to group this large number of operations on the points or on the atoms.

We experimented that, even if the partitioning on the number of points allows a greater scalability and parallelism degree, the achieved performance is lower than with the alternative strategy due to the large number of non-local memory access. In fact even if we store the coordinates of the atomic centers and the radii in the constant memory, each of these values has to be accessed a number of times equal to the number of threads.

Also the association of one thread for each atom has one drawback, that is the need to perform the updates of each point with atomic operations, because in principle the value of a point is the sum of the influence of all the atoms. This means that each point update has to be performed without race conditions, resulting in possible overhead due to the update serialization. However, as noted in [1] and [2], each atom influences in a significative way only the points within a limited bounding box surrounding it. This consideration has two important consequences. The first is to reduce the number of operation to be performed, since it is useless to consider all the atom-point pairs. Moreover, the concurrent updates are very limited, in the order of hundreds of atoms for each points, and therefore the impact of the serializations is negligible.

4.2 Isosurface Extraction

The *Marching Cubes* algorithm [3] is the most popular method used to extract triangulated isosurfaces from volumetric datasets. In the Marching Cubes algorithm the triangular mesh representing the isosurface is defined piecewise over the cells in which the grid is partitioned. A cell is intersected by the isosurface represented by the *isovalue* if the isovalue is between the minimum and the maximum of the values assumed by the eight points of the grid that defines each cell. This kind of cells is called *active cells*. An active cell contributes to approximate the isosurface for a patch made of triangles, and the union of all the patches represents the isosurface. The algorithm consists of two main operations, the *Cell Classification* and the *Active Cell Triangulation*.

The *Cell Classification* is the operation that determines if a cell is intersected by the isosurface or not. This is done using a bit vector of 8 fields of one bit, each of them corresponding to one point of the cell. Points with values greater or equal to the isovalue are marked with 1, otherwise with 0: therefore a cell is an *active cell* if the bit vector has a value different from 0 (all points values lower than the isovalue) and 255 (all points values greater than or equal to the isovalue).

In these cases the *Active Cell Triangulation* operation is performed, consisting in the approximation of the intersection with the isosurface, using a triangular patch. Considering that a surface may intersect a cell in 254 ways, that is all the values of the bit vector except 0 and 255, a *look-up table* is used to enumerate all the possible connectivity schema. The coordinates of the vertices of the triangles are computed as a linear *interpolation* of the values of intersected edges.

The parallelization of the original algorithm for the CUDA architecture is a quite straightforward task, because it is achieved by assigning one cell for each thread, and it is provided as a C code example in the NVIDIA CUDA SDK [3]. Obviously this application is only an example and it has many limitations, as for instance the small size of the volume that is able to process. Other proposals were published, with the main aim to speed up the processing of extracting and visualizing very large isosurfaces (see [4] for a survey). The main issue with these algorithms is the fact that they are designed for a direct visualization of the produced isosurfaces, and not for storing them. This means that they do not consider one major issue of the algorithm, that is the duplications of the vertices. Each active cell is in fact processed independently, and this means that a vertex may be recalculated up to four times in adjacent cells (see Figure 1(a)). The duplicated vertexes are useless and they may have a considerable impact on the computing time and on the size of the resulting mesh for further processing operations if these algorithms are used in a workflow. Obviously the vertices can be merged using a post-processing step, but this limits the achievable speed up.

In [4] we proposed a novel algorithm that is able to produce an isosurface equivalent to that produced by the sequential algorithm in an efficient way using the solution proposed in [5], that makes use of five auxiliary array data structures. The coordinates of a vertex are computed only the first time the corresponding edge intersected by the isosurface is considered. These coordinates are inserted in the Vertex table and the index corresponding to the vertex position in the table is stored in the correct position of one of the five auxiliary array data structures shown in Figure 1(b). As indicated in the Figure, in a generic cell (i.e., a cell which is not on the border of the volume) nine edges were previously considered in the processing of adjacent cells, therefore it is possible to produce at most three new vertices. The values in the auxiliary data structures are updated during the subsequent processing of all cells. For example, considering Figure 1(b), the black vertex is computed by the bottom left cell and its index is inserted in the corresponding position of the Ledge array structure. The next cell being processed is the bottom right one. This cell uses the stored index and moves it to the proper position in the Yedge array. When the next pair of slices is considered, the top left cell uses the index without needing to modify Yedge. Finally, the top right cell uses the index for the last time.

The CUDA-based version of the algorithm is composed by the following four kernels: *VerticesCalc*, where the coordinates of the vertices are computed; *VerticesCompact*, where vertices are associated with labels to be used to represent triangles and they are grouped to reduce transfer time; *TrianglesCalc*, where the triangles are computed as three labels of vertices; *TrianglesCompact*, to group the resulting triangles. The data transfer operations represent a considerable part of the time spent in performing the isosurface extraction operation on a device. Therefore we overlapped the data transfers and the kernel executions. In particular we overlapped: a) VerticesCalc with the transfer of the triangles found considering the previous pair of slices; b) TrianglesCalc with the transfer of the vertices; c) TrianglesCompact and the transfer of the next slice. This last overlap does not apply in this case, because the slice are created by the

[3] http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html

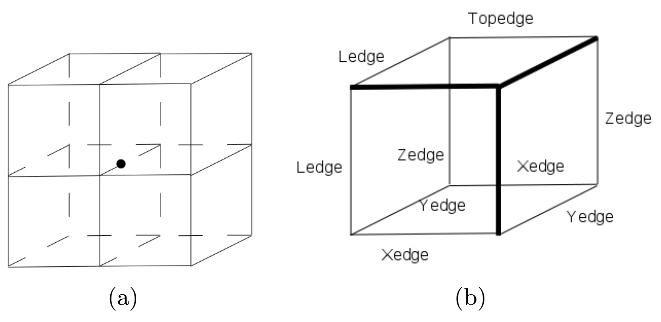

Fig. 1. The original algorithm computes the black vertex four times, one for each cell. The use of auxiliary data structures allows to avoid it. Thick lines represent edges not previously considered.

previous operation directly in the memory of the device. More details on this CUDA-based version of the algorithm are provided in [4].

5 Experimental Results

Experimental results were collected considering two implementations of the Blobby surface generation, one sequential and one parallel, for the CUDA architecture[4]. The two programs were executed on a workstation equipped with an Intel i5-750 CPU and an Nvidia GTX480 device. In particular the sequential implementation makes use of one core of the processor to perform the whole computation.

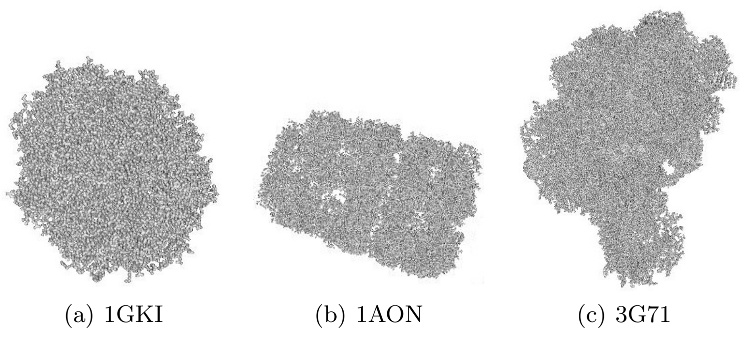

Fig. 2. The Blobby surfaces associated to the three molecules selected for the tests. We obtained them considering the volume with the resolution of 0.5 Å.

[4] Our software is available upon request, and will be made publicly available in case the paper is accepted.

Molecule	Atoms	Resolution	Grid	Triangles
1GKI	19,536	0.5	226x235x237	1,410,816
		0.1	1132x1177x1185	36,583,252
1AON	58,674	0.5	312x477x469	4,256,936
		0.1	1563x2385x2346	110,392,108
3G71	90,898	0.5	379x474x491	6,485,168
		0.1	1894x2367x2458	167,548,496

Table 1. This table summarizes the characteristics of the three considered molecules.

Three molecules of the Protein Data Bank repository, chosen on the basis of their size, were considered. The smallest one is the Plasmid coupling protein TrwB, identified as *1GKI* and made up by 19,536 atoms, followed by the crystal structure of the asymmetric GroEL-GroES-(ADP)7 chaperonin complex, identified as *1AON* and made up by 58,674 atoms, and one of the largest structure in the PDB repository, the co-crystal structure of Bruceantin bound to the large ribosomal subunit, identified as *3G71* and made up by 90,898 atoms. They are presented in Figure 2.

The B parameter was set to -2.3 and two steps, 0.5 and 0.1 Å were considered for the Scalar Field Generation operation. They represent, respectively, a medium and a high detailed resolution. The characteristics of the molecules, of the resulting volumetric datasets and the Blobby surfaces are shown in Table 1, while Table 2 shows the performance of the sequential and the parallel implementations.

		Scalar Field Generation		Isosurface Extraction		Total	
		Seq	CUDA	Seq	CUDA	Seq	CUDA
1GKI	0.5	2.78	0.11 (25.3)	1.38	0.07 (19.2)	4.16	0.18 (23.1)
	0.1	294.08	7.95 (37.0)	89.76	2.60 (34.5)	383.84	10.55 (36.4)
1AON	0.5	8.53	0.28 (30.9)	6.63	0.20 (32.8)	15.16	0.48 (31.6)
	0.1	832.14	20.70 (40.2)	472.03	12.90 (36.6)	1304.17	33.60 (38.8)
3G71	0.5	13.20	0.42 (31.4)	8.68	0.25 (34.7)	21.88	0.67 (32.7)
	0.1	1057.10	30.78 (34.3)	625.71	16.73 (37.4)	1682.81	47.51 (35.4)

Table 2. This table presents the times, in seconds, for executing the sequential and the parallel implementations of the Blobby Surface Generation. In brackets the achieved speedups. It is worth noting that, in the total time for the CUDA version, we do not considered the initialization time, that is of about 6.5 seconds in all the cases.

We can see that, except in the smallest case, the speedups achieved vary between 30 and 40. This is an incouraging result considering the issues related to the implementation of both the Scalar Field Generation and the Isosurface Extraction in CUDA. As regards the Scalar Field Generation we can see that the fixed parallelism degree do not allow to scale in proportion to the volume size, but this limit neither involves a degradation. Each CUDA thread in fact is responsible to assign the value to a few points for each slides, whose number varies from 2 to 60, therefore we are able to achieve good performance. This is also due to the fact that no data movement are required: each slide is created on the device memory, used for the isosurface extraction and then replaced with a new one without the need to involve the host memory.

The data movement instead is the factor that limits the performance of the Isosurface Extraction operation. We have to consider in fact that it requires the transfer of the triangular mesh representing the Blobby surface: in the largest case this mesh is composed by about 168 million triangles and 84 million vertices, resulting in about 9 GB of data to transfer. However the overlaps between data transfers and kernel executions permit to achieve high performance figures also in this case.

A final issue, common to all the CUDA programs, is represented by the time required to initialize the CUDA device, that it is performed in correspondence of the first call to a CUDA function within a program. In our case this time is equal to about 6.5 seconds, therefore the use of the CUDA version is unfeasible for small datasets as the 1GKI with a step of 0.5. It is however to consider that the Blobby surface generation is an operation that can be inserted in a workflow where other CUDA-based operation are executed: in these case the impact of the initialization time on the whole processing time is limited.

6 Conclusions and Future Works

This work presented a CUDA-based efficient algorithm for the Blobby molecular surface generation. In particular, the algorithm is able to achieve a speedup of 35.4 considering a molecule of 91,000 atoms and a resulting mesh of 168 million triangles. We experimented that a parallelization on the atoms, even if involves a lower level of parallelization, is able to provide higher performance figures than a parallelization on the points of the grid containing the scalar field representing the molecule due to the lower number of device memory accesses.

Two future works are forecasted. The first one is a further improvement of the performance of the algorithm, in particular by an in depth analysis of the role of the B parameter on the performance. The second one is the adoption of the algorithm in tools for molecular surface construction [11], in order to use the produced meshes to solve the Poisson-Boltzmann equation and/or visualization purposes.

7 Acknowledgments

This work was partially supported by NIGMS, NIH, grant number, 1R0GM093937-01

References

1. D'Agostino, D., Merelli, I., Clematis, A., Milanesi, L., Orro., A.: A parallel workflow for the reconstruction of molecular surfaces. Parallel Computing: Architectures, Algorithms and Applications, Advances in Parallel Computing, 15, 147–154, 2008.
2. Dias, S., Bora, K., Gomes, A.: CUDA-based triangulations of convolution molecular surfaces. Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10), 531–540, 2010.
3. Lorensen, W. E., Cline, H. E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Computer Graphics (Proceedings of SIGGRAPH 87), 21:4, 163–169, 1987.
4. D'Agostino, D. , Seinstra, F.J.: An Efficient Isosurface Extraction Component for Visualization Pipelines based on the CUDA Architecture. Technical Report IR-CS-64-2010, Vrije Universiteit, Amsterdam, The Netherlands.
   An extended version was submitted to the "Special Issue on Accelerators for High-Performance Computing" of the Journal of Parallel and Distributed Computing.
5. Watt and Watt: Advanced Animation and Rendering Techniques Theory and Practice. Addison-Wesley/ACM Press, 1992.
6. Yu, Z. , Holst, M. J., Cheng, Y., McCammon J.A.: Feature-preserving adaptive mesh generation for molecular shape modeling and simulation. Journal of Molecular Graphics and Modelling, 26:8, 1370–1380,2008.
7. Chavent ,M., Levy ,B., Maigret B.: MetaMol: High-quality visualization of molecular skin surface. Journal of Molecular Graphics and Modelling, 27:2,209–216, 2008.
8. Richards, FM.: Areas, volumes, packing and protein structure. Annu Rev Biophys Bioeng, 6, 151-176, 1977.
9. Connolly, M. L.: Analytical molecular surface calculation, J. Appl. Cryst, 16:5, 548-558, 1983.
10. Vorobjev, Y. N., Hermans, J.: SIMS: Computation of a Smooth Invariant Molecular Surface, Biophysical Journal, 73, 722–732, 1997.
11. Rocchia, W., Sridharan, S., Nicholls, A., Alexov, E., Chiabrera, A., Honig, B.: Rapid Grid-Based Construction of the Molecular Surface and the Use of Induced Surface Charge to Calculate Reaction Field Energies: Applications to the Molecular Systems and Geometric Objects, Journal of Computational Chemistry, 23:1, 128–137, 2001.
12. Blinn, J.: A generalization of algebraic surface drawing. ACM Transactions on Graphics, 1:3, 235-256, 1982.
13. Zhang, Y., Xu, G., Bajaj, C.: Quality meshing of implicit solvation models of biomolecular structures, Journal Computer Aided Geometric Design - Special issue: Applications of geometric modeling in the life sciences, 23:6, 2006.
14. Edelsbrunner, H.: Deformable Smooth Surface Design, Discrete and Computational Geometry, 21:1, 87 – 115, 1999.
15. Lindow, N., Baum, D., Prohaska, S., Hege, H.C.: Accelerated Visualization of Dynamic Molecular Surfaces, Eurographics/ IEEE-VGTC Symposium on Visualization,29:3, 943–952, 2010.
16. Lu, Q., Luo,R., A Poisson Boltzmann dynamics method with nonperiodic boundary condition, J. Chem. Phys. 119,11035, 2003.
17. Im, W., Beglov , D., Roux, B.: Continuum solvation model: Electrostatic forces from numerical solutions to the Poisson-Bolztmann equation, Comp. Phys. Comm. 111, 59–75, 1998.