Applied Mathematics & Information Sciences
An International Journal

GPU Accelerated Molecular Surface Computing

Byungjoo Kim[1], Ku-Jin Kim[2*] and Joon-Kyung Seong[3]
[1]Graduate School of Electronics, Kyungpook National University, 702-701 Daegu, Korea
[2]School of Computer Science & Engineering Kyungpook National University, 702-701 Daegu, Korea
[3]School of Computer Science & Engineering, Soongsil University, Seoul, Korea
Email Address: [1] byungjoo.kim@gmail.com, [2] kujinkim@yahoo.com, [3] seong@ssu.ac.kr

Received: Received May 02, 2011; Revised July 25, 2011; Accepted September 12, 2011
Published online: 1 January 2012

Abstract: A method is presented for computing the SES (solvent excluded surface) of a protein molecule in interactive-time based on GPU (graphics processing unit) acceleration. First, the offset surface of the van der Waals spheres is sampled using an offset distance *d* that corresponds to the radius of the solvent probe. The SES is then constructed by extracting the surface at distance *d* from the sample points. For interactive-time computing, two space partitioning schemes are used, a voxel map and *k*d-tree, with data parallel schemes accelerated by GPU. In experiments using an average 1,848 atoms, a SES with a resolution of $1/2^7 \times 1/2^7 \times 1/2^7$ of the original bounding box is obtained in 66.53ms on average.

Keywords: Molecular surface, solvent-excluded surface, offset, voxel map, GPU acceleration

1 Introduction

In the area of biochemistry, molecular surfaces are widely used for visualizing the shape of a molecule, computing the areas and volumes of molecules, molecular docking, and drug design. Among the many types of molecular surface, such as a van der Waals surface, SAS (solvent accessible surface), SES (solvent excluded surface), and MSS (molecule skin surface), SESs are most frequently used, since being defined by Connolly [5].

A protein molecule is generally represented as a set of spheres, where each sphere corresponds to an atom and the center position and radius of each sphere are determined by the center position and van der Waals radius of the atom, respectively. The SES is then defined as the inner boundary surface of the union of touching solvent probes to a given molecule *M*, where *M* is represented as a set of 3-dimensional spheres with van der Waals radii. Fig.1 (a) shows an example of the offset surface of a molecule, called the SAS, where the gray spheres correspond to the atoms, the dashed curve represents the solvent probe, and the offset surface is represented by the thick solid curve that corresponds to the center trajectory of the probe sphere. Fig.1 (b) shows an example of an SES, which is the boundary of the union of all possible solvent probes that touch the molecule *M* without colliding with the atoms, as represented by the thick curve. Thus, the SES essentially consists of a set of points whose minimum distance from the SAS is the radius of the solvent probe.

This paper presents an interactive-time algorithm for computing the SES of a protein molecule that is both easy to implement and robust in numerical errors. The molecule is represented as a sphere set *M*. Given a probe solvent that is considered as a sphere with radius *d*, points are sampled from the offset surface of *M* with offset distance *d*. A *k*d-tree is then created from the sample points, resulting in the construction of a

* **Corresponding Author:** Ku-Jin Kim, **kujinkim@yahoo.com**

distance field. Defined as a spatial field of minimum distances for a given set of primitives [7], the computation of distance fields around complex objects has already been developed using graphics hardware [14, 18, 19]. Based on the distance field, the SES is then approximated in a piecewise linear form by applying the marching cube method [11, 22].

To obtain interactive-time computing, two kinds of space-partitioning structure, a voxel map and *k*d-tree, are effectively applied using a data parallel algorithm. Containing information on the offset spheres that cover each voxel, the voxel map allows the offset surface of the sphere set to be computed efficiently. Meanwhile, the *k*d-tree enables the efficient construction of a distance field from the offset surface. Plus, GPU acceleration helps with constructing the voxel map, sampling the offset surface, and constructing the distance field when applying the marching cube algorithm.

The remainder of this paper is organized as follows. Section 2 discusses related work, then the proposed algorithm for molecular surface generation is explained in Section 3. Experimental results are presented in Section 4, and some final conclusions are given in Section 5.

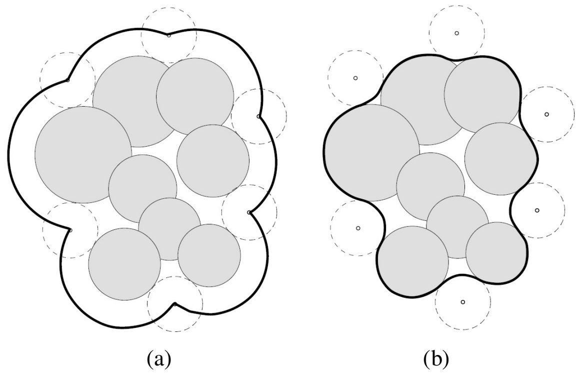


Fig.1 Examples of molecular surfaces: offset surface (or SAS) (left) and SES (right)

2 Related Work

Various researchers have already attempted to compute the molecular surfaces of protein structures. Connolly [4, 5] categorized the molecular surface into three types of patch: convex, toroidal, and concave, which are generated when the solvent probe touches only one atom, two atoms, and three atoms, respectively. Nicolls et al. [12, 13] developed the GRASP system that generates the molecular surface by dividing the space into grids. Sanner et al. [17] computed molecular surfaces based on the alpha shape of the molecule, and this method has been used to implement the UCSF Chimera system [15, 21].

Can et al. [2] proposed a method to compute the van der Waals surface, SAS, and SES based on the propagation of a level set by partitioning the space into uniform grids. Using a seed-filling method, they first find a grid that contains the molecular surface, then use it as a seed to search for neighbor grids that also contain the molecular surface. Ryu et al. [16] computed the beta shape of a molecule based on a Voronoi diagram computed for the atom set. This beta shape is used to generate the contact and re-entrant surfaces in the SES, and the SES is then constructed by connecting them.

Juba and Varshney [8] computed the area of a molecular surface using graphic hardware. They represent the atoms as Gaussian radial basis functions (RBFs) based on the center of the atoms. The molecular surface is then represented as the level set of the summation of the Gaussian RBFs. They generate a set of random lines, and by computing the intersection of the lines with the implicit surface, they compute the area of the molecular surface. Their algorithm also uses GPU in parallel intersecting each line with the implicit surface.

Most recently, there have been several studies that focused on the fast visualization of molecular surfaces mainly for molecular dynamic applications. Lindow et al. [10] proposed an interactive-time method to generate the SES and MSS using an approximate Voronoi diagram and contour-buildup algorithm [20]. They reduce the rendering time by parallel computing using multi-core CPUs. Krone et al. [9] presented an approach for interactive SES visualization for molecular dynamic simulation trajectories based on GPU ray casting. Chavent et al. [3] presented a GPU accelerated ray-tracing method for visualizing molecules that directly uses a piecewise-defined algebraic equation of the molecular skin surfaces. Yet, despite a high performance for visualizing molecules in an image space, such methods are limited as regards producing a mesh geometry of the molecular surface, thereby restricting their use with other applications that require the geometric properties of molecular surfaces, such as computing their area/volume, molecular docking, and drug design.

3 Algorithm for Interactive-time SES Computation

3.1 Overview

An overview of the proposed algorithm is given in Algorithm 1. The first step involves the computation of a set of sample points, P, on the offset surface of M with offset distance d. After constructing a kd-tree based on the set of points P, a marching cube algorithm is applied to extract the *SES* that corresponds to the iso-surface whose distance is d from P. The functions used to generate the sample points P and extract the SES using the marching cube algorithm are implemented as GPU kernel functions to achieve an interactive-time performance.

Algorithm 1: Interactive-Time SES Computation

```
function ComputeSES ($M$, $X$, $X^d$)
Input:   $M$;  /* a set of balls corresponding to the molecule */
         $X$;  /* the bounding box of $M$ */
         $X^d$;  /* the bounding box of the offset surface of $M$ */
Begin
Step 1.  /* Find points on the offset surface */
         $FP$ = SamplingOffset($M$, $X^d$)
Step 2.  /* Construct a kd-tree for the sample points in $FP$ */
         $T^d$ = ConstructKdTree($FP$);
Step 3.  $SES$ = MarchingCube ($T^d$, $X$);
         Return $SES$;
End
```

3.2 GPU-Based Extraction of Sampling Points from Offset Surface of Molecule

Let us denote the ball with center point $\mathbf{c}$ and radius r as $B(\mathbf{c}, r)$: $B(\mathbf{c}, r) = \{ \mathbf{p} \mid \|\mathbf{c} - \mathbf{p}\| \le r \}$. The surface of $B(\mathbf{c}, r)$ is denoted as $S(\mathbf{c}, r)$: $S(\mathbf{c}, r) = \{ \mathbf{p} \mid \|\mathbf{c} - \mathbf{p}\| = r \}$. The protein molecule is then represented as follows:

$$M = \{ B_i \mid 0 \le i < n \},$$

where $B_i = B(\mathbf{c}_i, r_i)$ corresponds to each atom with center $\mathbf{c}_i$ and van der Waals radius r_i. When the radius of the given solvent probe is d, the offset surface of M with offset distance d can be considered as a boundary surface of M^d:

$$M^d = \{ B^d_i \mid 0 \le i < n \},$$

where $B^d_i = B(\mathbf{c}_i, r_i + d)$ corresponds to the d-offset of B_i. We also define S^d_i as follows:

$$S^d_i = S(\mathbf{c}_i, r_i + d).$$

When we partition the bounding box of M^d, X^d, into $n_x \times n_y \times n_z$ voxels, the bounding box can be represented as a set of voxels as follows:

$$V = \{ V^{\alpha\beta\gamma} \mid 0 \le \alpha < n_x, 0 \le \beta < n_y, 0 \le \gamma < n_z \}.$$

For each ball B_i^d in M, we find every voxel $V^{\alpha\beta\gamma}$ it intersects, and then add B_i^d to $V^{\alpha\beta\gamma}.balls$:

$$V^{\alpha\beta\gamma}.balls = \{ B_i^d \mid B_i^d \cap V^{\alpha\beta\gamma}.box \neq \varnothing \},$$

where $V^{\alpha\beta\gamma}.box$ represents the voxel area. By using the voxel map V, we can efficiently determine whether or not the points embedded in S^d_i are contained in the offset surface of M. For a point $\mathbf{p} \in S^d_i$, if there is a ball B^d_j, $i \neq j$, which includes $\mathbf{p}$ inside, then $\mathbf{p}$ is not embedded in the offset surface of M . In Fig.2 (a), the offset of each atom (dashed curve) is represented as a solid curve. The sample points from the offset of each atom are shown as dots. After removing the sample points that are inside other atoms, we can extract the sample points embedded in the offset surface of the molecule (Fig.2 (b)).

The functions used to construct the voxel map and determine whether each sample point on a sphere, S^d_i is embedded in the offset surface of M are both accelerated using GPU. The algorithm details are presented in Algorithm 2.

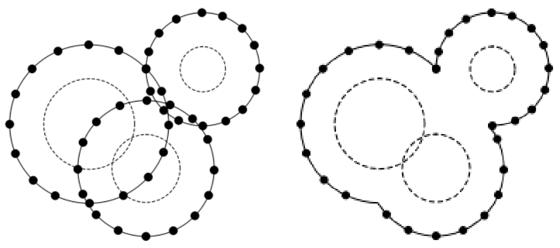

(a) (b)

Fig.2 Sample points on offset surface of atoms

Algorithm 2: Sampling the Points on SAS

```
function SamplingOffset ($M$, $X^d$)
Begin

/* CPU code */
$FP = \varnothing$;

/* GPU kernel call (Input: $M$ and $V^{\alpha\beta\gamma}$, Output: $V^{\alpha\beta\gamma}.balls$) */
For each $B_i^d \in M^d$, $0 \le i < n$, do in parallel
    $V^{\alpha\beta\gamma}.balls = \{ B_i^d \mid B_i^d \cap V^{\alpha\beta\gamma}.box \neq \varnothing \}$;
/* GPU kernel call (Input: $M$ and $V^{\alpha\beta\gamma}$, Output: $P_i^d$) */
For each $B_i^d \in M^d$, $0 \le i < n$, do in parallel
    Generate $P_i^d$ = { Sample points on $S_i^d$};
/* GPU kernel call (Input: $P_i^d$ and $V^{\alpha\beta\gamma}$, Output: $FP$) */
For each $\mathbf{q} \in P_i^d$, $0 \le i < n$, do in parallel
    For each $B_j^d \in V^{\alpha\beta\gamma}.balls$, where $V^{\alpha\beta\gamma}.balls$ contains
    $B_i^d$, $i \neq j$, do
        if $\mathbf{q} \cap B_j^d \neq \varnothing$ then
            Remove $\mathbf{q}$ from $P_i^d$;
        else
            add $\mathbf{q}$ to $FP$;
    end

/* CPU code */
return FP;
End.
```

3.3 Construction of *k*d-tree for sample points on offset surface

The bounding box X^d of a point set *FP* is the smallest axially aligned box that contains all the points in *FP*. The initial box is subdivided until the number of points contained in a box X^d is less than or equal to a threshold *m*. Each subdivision is constructed by a splitting plane which is orthogonal to the axis *A* that corresponds to the longest side of the current box. The splitting plane passes through the median of the points aligned along axis *A*. Algorithm 3 shows the algorithm used to construct a *k*d-tree on the point set *FP*. Fig.3 shows an example of *k*d-tree construction for a set of points.

Algorithm 3: Construct a *k*d-tree for the sample points on the offset surface

```
Function ConstructKdTree(FP, $X^d$)
Input:   FP; /* A set of points on the offset surface of M */

Begin
    if FP contains fewer than m points then return;
    new node;
    A := axis corresponding to the largest dimension of
         $X^d$, where $X^d$ is the bounding box of FP;
    P := plane that orthogonally cuts axis A at the median
         of the point positions in $X^d$;
    Subdivide FP into two sets $S_L$ and $S_R$ by the point
    positions with respect to P;
    node.lChild := ConstructKdTree ($S_L$);
    node.rChild := ConstructKdTree ($S_R$);
    return node;
End
```

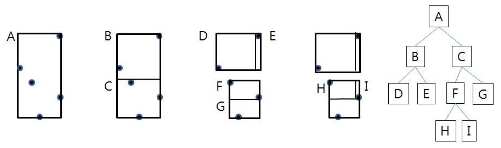


Fig.3 Example of *k*d-tree construction

3.4 Extracting SES

The marching cube algorithm [11] is widely used for extracting a polygonal mesh that approximates an iso-surface from a 3-dimensional scalar field. It uses a divide-and-conquer approach to find the cuboids that contain the iso-surface. It determines the type of intersection between the surface and the cuboids by computing the data values given to each vertex of the cuboid, and then the final topology of the surface in each cuboid is determined.

Here, a GPU-accelerated marching-cube algorithm [22] is used to extract the SES, where the whole space is partitioned into uniform size cuboids and the data value for each vertex of the cuboids is evaluated in parallel. The type of intersection between a surface and the cuboids is also evaluated in parallel.

When **p** is an arbitrary point, the distance from **p** to *FP* can be computed using function Dist(**p**, T^d) in Algorithm 4. Let us partition the bounding box *X* into a set of small boxes X^{abc}. When we label the eight corners of the bounding box X^{abc} as v_i, $i = 1, 2, \ldots, 8$ (see Fig.4), the distances from each vertex v_i to *FP* can be computed using Dist (v_i, T^d). Let d_i = Dist (v_i, T^d), then, we can store a scalar value $s_i = d_i - d$ for each vertex. We use s_i for each vertex as the data value assigned to the marching cube algorithm. Since v_i always contains $s_i = d_i - d$, it effectively represents the signed distance as a distance field. Finally, for each box X^{abc}, the SES is constructed as a linear approximation of the zero-level iso-surface of the distance field.

Fig.4 shows an example of computing the distance field, where each dashed circle represents an original atom. The sample points around the atoms are used to approximate the offset surface of the atoms. Some voxels are shown as rectangles with the signs of $s_i = d_i - d$ values, where d_i is computed as the distance to the closest sample point from each vertex. The marching cube algorithm decides that a voxel that has vertices with the same sign does not contain the SES. Otherwise, it extracts the SES by linearly interpolating the s_i values at the vertices.

Algorithm 4: Distance from a spatial point to sample point-set *FP*

```
function Dist (p, T)
begin
    if T is a leaf node then
        return Min(Dist(p, $q_i$)) for all $q_i$ in T;
    else begin
        Let $T_1$ point to the child of T that contains p, and
        $T_2$ point to the other;
        MinDist = Dist (p, $T_1$);
        if any part in the bounding box of $T_2$ is within a
        distance MinDist from p then
            return Min(MinDist, Dist (p, $T_2$));
        else
```

```
            return MinDist;
    end
end
```

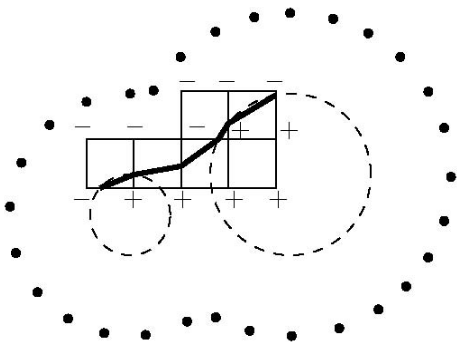


Fig.4 Computation of distance field for set of sample points on offset surface

4. Experimental Results

The proposed algorithm was implemented using Microsoft Visual C++, and its effectiveness assessed based on experiments using a PC with an Intel i5 2.8GHz CPU, Nvidia GeForce GTX590 graphic card, and 4Gbytes of memory. Nineteen protein molecule examples were tested, which are downloaded from the Protein Data Bank (PDB) website. The GPU algorithm was implemented using CUDA [23].

The SES of a protein molecule was computed using its default alignments and orientations in the PDB file. The radius of the given probe solvent was assumed to be 1.4Å, representing the radius of a water molecule. The atoms were represented by spheres using their van der Waals radii [1]. The experimental results for the sample sets of the 19 test cases are shown in Table.1. The first column, PDB ID, shows the identification code in the PDB, the second column indicates the number of atoms (except hydrogen) in the protein, and the third column shows the SES computation time in milliseconds.

PDB id.	Number of atoms	Number of triangles in SES	Computation time (ms)				
			Sample offset of *M*	Construct *kd* tree	Marching cube	Render	Total
110D	120	15,168	3.79	3.82	1.74	16	25.35
200D	259	21,340	5.01	7.61	2.25	16	30.87
1QL1	322	16,676	5.01	5.74	2.07	16	28.82
4PTI	381	23,844	5.79	6.36	3.09	16	31.24
1BK2	468	32,808	6.16	6.11	4.22	16	32.49
2QZF	479	31,060	5.89	7.01	3.75	16	32.65
2QZD	507	27,760	5.78	7.58	3.11	16	32.47
2OT5	545	30,752	5.45	10.12	3.18	16	34.75
1HH0	691	18,860	6.42	8.03	2.26	16	32.71
1HGV	691	16,788	6.44	9.07	2.25	16	33.76
1HGZ	691	18,664	6.26	9.57	2.24	16	34.07
2INS	781	35,584	5.93	9.84	4.09	16	35.86
1QL2	966	24,388	5.24	24.82	2.29	16	48.35
1IZH	1,521	39,744	6.33	20.35	4.59	16	47.27
1GT0	2,746	49,808	6.85	62.77	4.01	16	89.63
1BIJ	4,387	64,424	10.66	64.89	6.14	16	97.69
1G50	5,886	45,684	12.16	79.01	5.09	16	112.26
168L	6,449	58,516	12.74	104.59	6.74	16	140.07
1QGK	7,231	59,984	13.82	106.16	5.86	16	141.84
average	1,848	33,255	7.14	29.13	3.63	16	55.90

Table.1 SES computation when using $2^6 \times 2^6 \times 2^6$ voxels

PDB id.	Number of atoms	Number of triangles in SES	Computation time (ms)				
			Sample offset of M	Construct *kd* tree	Marching cube	Render	Total
110D	120	62,280	3.79	3.82	5.64	16	29.25
200D	259	87,524	5.01	7.61	8.72	16	37.34
1QL1	322	70,096	5.01	5.74	6.52	16	33.27
4PTI	381	101,748	5.79	6.36	11.03	16	39.18
1BK2	468	139,564	6.16	6.11	17.44	16	45.71
2QZF	479	131,476	5.89	7.01	14.63	16	43.53
2QZD	507	118,060	5.78	7.58	12.23	16	41.59
2OT5	545	129,728	5.45	10.12	11.24	16	42.81
1HH0	691	79,380	6.42	8.03	7.58	16	38.03
1HGV	691	70,800	6.44	9.07	6.84	16	38.35
1HGZ	691	79,064	6.26	9.57	7.33	16	39.16
2INS	781	154,036	5.93	9.84	16.87	16	48.64
1QL2	966	102,524	5.24	24.82	9.31	16	55.37
1IZH	1,521	176,308	6.33	20.35	17.55	16	60.23
1GT0	2,746	218,244	6.85	62.77	15.97	16	101.59
1BIJ	4,387	288,676	10.66	64.89	26.96	16	118.51
1G50	5,886	207,860	12.16	79.01	20.76	16	127.93
168L	6,449	265,064	12.74	104.59	28.97	16	162.30
1QGK	7,231	273,404	13.82	106.16	25.23	16	161.21
average	1,848	145,044	7.14	29.13	14.25	16	66.53

Table.2 SES computation when using $2^7 \times 2^7 \times 2^7$ voxels

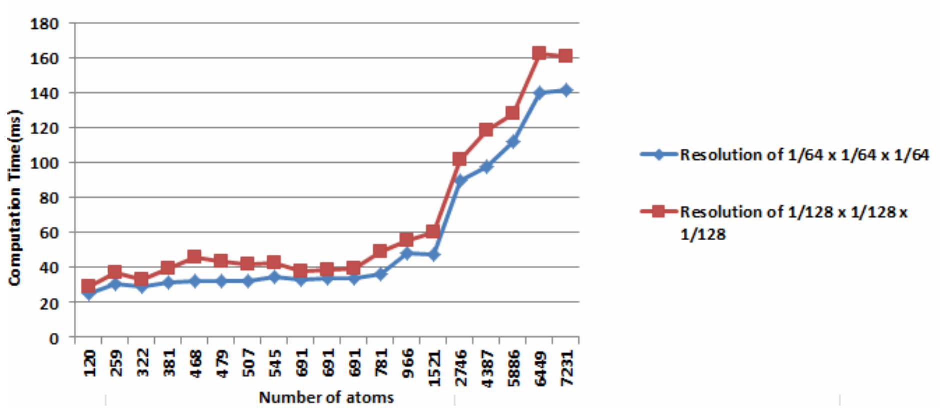


Fig.5 Graph of computation time according to number of atoms

As shown in Table.1, in 19 experiments with up to 7,231 spheres, when dividing the bounding box of each molecule into $2^6 \times 2^6 \times 2^6$ voxels, the SES was obtained in 55.9 ms on average. Table.2 shows the case when using $2^7 \times 2^7 \times 2^7$ voxels, where the SES was obtained in 66.53 ms on average. The graph in Fig.5 shows the computation time changes according to the number of atoms. The CUDA-based implementation can be used in an interactive manner. Fig.6 shows some examples of the SES generated using the proposed method. Fig.6 (a) and Fig.6 (b) show the molecule represented as a set of spheres and its SES, respectively.

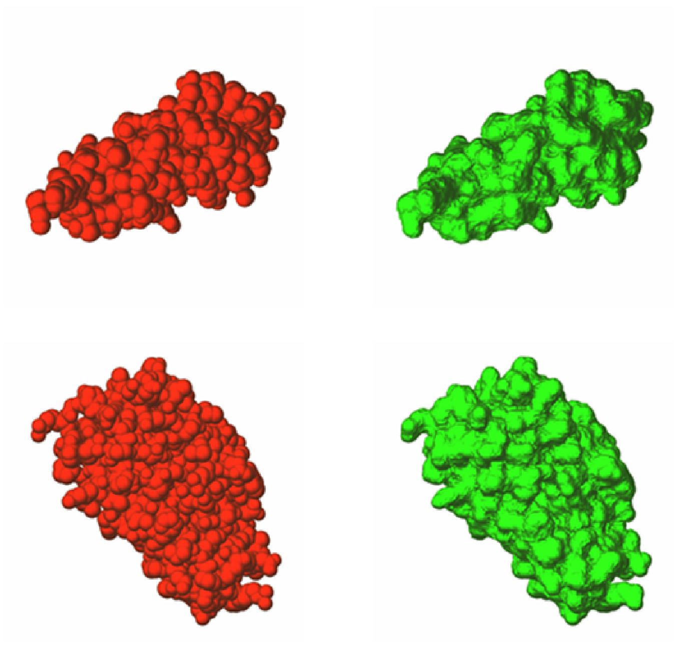
To our knowledge, there has been no previous attempt to compute the SES using graphics hardware; thus, the performance of the proposed algorithm was compared with the previous work by Dias and Gomes [6] that computes the Blinn molecular surface based on CUDA. The Blinn molecular surface is computed by representing the molecular surface as an implicit function that is the summation of local functions that describe the electrical field of the atoms. Although the surfaces generated by the algorithm in [6] and the proposed algorithm are different, a rough comparison is made of the performances of these algorithms. Usually the performance of an algorithm implemented with CUDA greatly depends on the optimization, so rather than implementing their code, the experimental results shown in [6] are compared with ours under an environment equipped with the same graphic card (Nvidia GeForce GTX 280). We used $2^7 \times 2^7 \times 2^7$ voxels and 1.4Å as the radius of the solvent probe in this experiment. Table.3 compares the performance of the algorithm suggested in [6] and the proposed algorithm. Except for a small number of atoms as the input, the proposed algorithm showed a better performance in almost all cases that was ten times faster on average compared to the method in [6].

5 Conclusion

This paper presented an interactive-time algorithm to compute the SES of a protein molecule that is both easy to implement and robust in numerical errors. By using data parallel scheme provided by graphics hardware, the proposed algorithm generates an SES in interactive time. As regards user-specified accuracy limits, the proposed algorithm computes the SES with a resolution of $(1/2^7)^3$ of the original bounding box in 66.53 milliseconds on average for an average of 1,848 atoms. In further studies, the proposed algorithm will be improved to allow more sophisticated control of the resolution of the SES by adaptive construction of the voxel map and distance field, along with the development of interactive-time algorithms for molecular docking and drug design.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (grant number 2011-0004094).



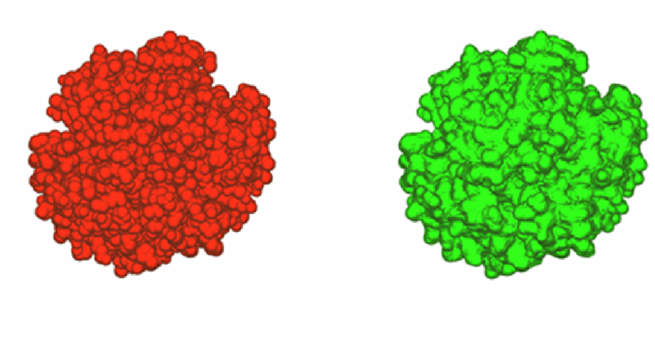


Fig.6 The molecule and its SES for the molecules (PDB id: 2QZD, 1IZH, and 1BIJ from top to bottom) with the solvent probe of radius 1.4Å

PDB id.	Number of atoms	Algorithm in [6]			Our algorithm			Speedup (T.D/ T.O)
		CPU: Intel Quad Core Q9550 2.83 GHz RAM: 1Gbytes GPU: Nvidia GeForce GTX 280			CPU: AMD AthlonII X4 3.0GHz RAM: 8Gbytes GPU: Nvidia GeForce GTX 280			
		# of Voxels	# of triangles	T.D	# of Voxels	# of triangles	T.O	
110D	120	232,960	8,520	30	2,097,152	62,280	58.78	0.51
200D	259	386,560	17,794	60	2,097,152	87,524	65.72	0.91
1QL1	322	960,640	23,948	100	2,097,152	70,096	70.26	1.42
4PTI	381	654,336	27,820	120	2,097,152	101,748	80.69	1.49
1BK2	468	410,368	28,424	100	2,097,152	139,564	95.82	1.04
2QZF	479	2,057,984	31,972	250	2,097,152	131,476	84.03	2.98
2QZD	507	2,268,928	32,952	310	2,097,152	118,060	78.87	3.93
2OT5	545	707,072	40,564	140	2,097,152	129,728	82.01	1.71
1HH0	691	999,808	33,586	200	2,097,152	79,380	82.76	2.42
1HGV	691	1,184,640	34,058	230	2,097,152	70,800	84.38	2.73
1HGZ	691	985,856	33,026	200	2,097,152	79,064	83.89	2.38
2INS	781	808,192	44,400	250	2,097,152	154,036	95.72	2.61
1QL2	966	2,666,752	72,858	750	2,097,152	102,524	100.72	7.45
1IZH	1521	1,514,240	82,512	780	2,097,152	176,308	120.62	6.47
1GT0	2746	3,244,160	148,788	2,890	2,097,152	218,244	168.05	17.20
1BIJ	4387	4,882,944	212,604	6,200	2,097,152	288,676	194.18	31.93
1G50	5886	7,552,640	258,236	7,400	2,097,152	207,860	212.94	34.75
168L	6449	6,175,872	307,932	9,300	2,097,152	265,064	256.23	36.30
1QGK	7231	7,012,736	335,544	10,800	2,097,152	273,404	267.97	40.30
Average	1,848	2,352,984	93449	2,111	2,097,152	145,044	120.19	10.45

T.D: computation time (ms) for the algorithm suggested in [6]
T.O: computation time (ms) for our algorithm

Table.3 Comparison of the algorithm in [6] and our algorithm

References

[1] Bondi, A.: Van der Waals Volumes and Radii. The Journal of Physical Chemistry 68 (3) (1964) 441–451.

[2] Can, T., Chen, C.I., and Wang, Y.F.: Efficient molecular surface generation using level-set methods. J. of Molecular Graphics and Modeling 25 (4) (2006) 442-454.

[3] Chavent, M., Levy, B., and Maigret, B.: Metamol: High-quality visualization of molecular skin surface. Journal of Molecular Graphics and Modelling 27 (2) (2008) 209–216.

[4] Connolly, M.L.: Analytical molecular surface calculation. J. Appl. Crystallogr. (1983) 548-558.

[5] Connolly, M.L.: Solvent-accessible surfaces of proteins and nucleic acids. Science 221 (1983) 709-713.

[6] Dias, S. E. and Gomes, A. J.: Graphics processing unit-based triangulations of Blinn molecular surfaces. Concurrency and Computation: Practice and Experience. doi: 10.1002/cpe.1783 (online version) (2011)

[7] Jones, M. W., Bærentzen, J. A., Sramek, M.: 3D Distance Fields: A Survey of Techniques and Applications. IEEE Transactions on Visualization and Computer Graphics. (2006) 581-599.

[8] Juba, D. and Varshney, A.: Parallel, stochastic measurement of molecular surface area. Journal of Molecular Graphics and Modeling 27 (2008) 82-87.

[9] Krone, M., Bidmon, K., and Ertl, T.: Interactive visualization of molecular surface dynamics. IEEE Transactions on Visualization and Computer Graphics 15 (6) (2009) 1391–1398.

[10] Lindow, N., Baum, D., Prohaska, S., and Hege, H.C.: Accelerated visualization of dynamic molecular surfaces. Computer Graphics Forum 29 (3) (2010) 943–952.

[11] Lorensen, W. E. and Cline, H. E.: Marching Cubes: A high resolution 3D surface construction algorithm. Computer Graphics 21 (4) (1987) 163-169.

[12] Nicholls, A., Sharp, K., and Honig, B.: Protein folding and association: insights from the interfacial and thermodynamic properties of hydrocarbons. Proteins 11 (4) (1991) 281-296.

[13] Nicholls, A.: GRASP: Graphical Representation and Analysis of Surface Properties. Columbia University, New York (1992).

[14] Park, T., Lee, S. –H., Kim, J. -H., and Kim, C. -H.: CUDA-based Signed Distance Field Calculation for Adaptive Grids. 10th IEEE International Conference on Computer and Information Technology (2010).

[15] Pettersen, E. F., Goddards, T. D., Huang, C. C., Couch, G. S., Greenblatt, D. M., Meng, E. C., and Ferrin, T. E.; UCSF chimera: a visualization system for exploratory research and analysis. J. Comput. Chem. 25 (13) (2004) 1605-1612.

[16] Ryu, J., Park, R., and Kim, D. -S.: Molecular surfaces on proteins via beta shapes. Computer-Aided Design 39 (2007) 1042-1057.

[17] Sanner, M. F., Spehner, J. C., and Olson, A. J.: Reduced surface: an efficient way to compute molecular surfaces. Biopolymers 38 (3) (1996) 305-320.

[18] Sud, A., Otaduy, M. A. and Manocha, D.: DiFi: Fast 3D Distance Field Computation Using Graphics Hardware. Computer Graphics Forum 23(3) (2004) 557-566.

[19] Sud, A., Govindaraju, N. K., Gayle, R. and Manocha, D.: Interactive 3D Distance Field Computation Using Linear Factorization. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D) (2006).

[20] Totrov, M. and Abagyan, R.: The contour-buildup algorithm to calculate the analytical molecular surface. J. Struct. Biol. 115 (1996) 1–6.

[21] Chimera website http://www.cgl.ucsf.edu/chimera/

[22] Marching cube algorithm as a CUDA SDK (http://developer.nvidia.com/cuda-toolkit-sdk)

[23] Nvidia CUDA website (http://developer.nvidia.com)

ByungJoo Kim is currently a Ph.D. candidate in the Graduate School of Electronics at Kyungpook National University, Korea. He received his B.S. and M.S. degrees in Electronics from Kyungpook National University. From 2006 to 2009, he was a senior research engineer at HUONE Co., Ltd. In 2010, he was a senior research engineer at LG Electronics Co., Ltd. His research interests include computer graphics, computer vision, and geometric/surface modeling.

Ku-Jin Kim is an Associate Professor in the School of Computer Science & Engineering at Kyungpook National University, Korea. Her research interests include computer graphics, computer vision, and geometric/surface modeling. Prof. Kim received her BS degree from Ewha Womans University in 1990, MS degree from KAIST in 1992, and Ph.D. degree from

POSTECH in 1998, all in Computer Science. She was a PostDoctoral fellow at Purdue University in 1998-2000. Prof. Kim also held faculty positions at Ajou University, Korea and the University of Missouri, St. Louis, USA.

Joon-Kyung Seong is an Assistant Professor in the School of Computer Science and Engineering at Soongsil University, Korea. His research interests include computer graphics, geometric modeling, and computational brain imaging. Prof. Seong received his B.S. and Ph.D. degrees from Seoul National University in 2000 and 2005, respectively. He was a Postdoctoral fellow in the School of Computing at the University of Utah from 2005 to 2008. After that he was a Research Professor in the Department of Computer Science at KAIST.