Parallel, Stochastic Measurement of Molecular Surface Area

Derek Juba [a], Amitabh Varshney [a]

[a] *Department of Computer Science, University of Maryland, College Park, MD 20742*

Abstract

Biochemists often wish to compute surface areas of proteins. A variety of algorithms have been developed for this task, but they are designed for traditional single-processor architectures. The current trend in computer hardware is towards increasingly parallel architectures for which these algorithms are not well suited.

We describe a parallel, stochastic algorithm for molecular surface area computation that maps well to the emerging multi-core architectures. Our algorithm is also progressive, providing a rough estimate of surface area immediately and refining this estimate as time goes on. Furthermore, the algorithm generates points on the molecular surface which can be used for point-based rendering.

We demonstrate a GPU implementation of our algorithm and show that it compares favorably with several existing molecular surface computation programs, giving fast estimates of the molecular surface area with good accuracy.

Key words: Molecular Surface, Parallel, Progressive, GPU, Stochastic, Quasi-Random

1 Introduction

Computation of molecular surface area is important in the grand challenge problems of molecular docking and protein folding as it allows one to incorporate the effects of solvent in the potential energy calculations. Recent work on interactive manipulation [1] and visualization of large-scale proteins [2] shows us how interactive visualization offers a powerful front end for computational

Email addresses: juba@cs.umd.edu (Derek Juba), varshney@cs.umd.edu (Amitabh Varshney).

steering of calculations. In such settings, rapid calculation of protein conformations becomes especially important and fast solvent-solute interactions are an essential first step. In this paper we address the mapping of molecular surface area calculations on the emerging multi-core architectures for potential use in interleaved computation and visualization of large bio-macromolecular complexes.

To serve this need for molecular surface area computation, a wide variety of algorithms and programs have been developed— a few examples are the early works by Connolly [3] [4], MSMS [5] by Sanner *et al.*, GETAREA [6] by Fraczkiewicz and Braun, LSMS [7] by Can *et al.*, 3V [8] by Voss, and an adaptive grid-based algorithm [9] included in TexMol [10] by Bajaj *et al.*. These algorithms have been designed to work well on traditional, single-processor computer architectures using a serial programming model.

However, computer architectures are now facing the first major disruptive challenge in over two decades in the form of pervasive parallelism. For example, AMD and Intel have already changed their product lines to include dual-core and quad-core processors. According to the Intel road map, they plan to have hundreds of cores on a single chip becoming a reality over the next decade. The Cell processor has 8 stream processing cores in addition to a conventional scalar processor. GPUs have been at the forefront of the multi-core revolution in that they are already shipping with hundreds of cores. NVIDIA's G80 has 128 multiprocessors. Intel has recently disclosed their plans for a GPU consisting of 24-32 cores each involving a 16-wide SIMD vector processor with over 2 TFLOPs of performance. In addition, GPUs and CPUs are being merged thereby blurring the distinction between cores that specialize for graphics and cores that are more general-purpose. Both AMD and Intel are working on fused CPU-GPU cores; this will enable tight coupling between applications and graphics. Because of the large computer games market, these highly-parallel GPUs are being mass produced and are available for commodity prices. While the use of this hardware for scientific computation originally required some unpleasant hacks, recent development environments such as NVIDIA's CUDA (Compute Unified Device Architecture) [11] and ATI's CTM (Close To Metal) [12] make the use of this hardware much more elegant. Bringing GPU computing further into the mainstream is NVIDIA's Tesla product line, a GPU designed specifically for general-purpose computation.

Unfortunately, algorithms and programs designed for a single-processor architecture are often not able to directly take advantage of these new parallel processors. Algorithms designed for serial computation can sometimes be parallelized, but this can be a non-trivial task.

To take advantage of this new trend in high-performance computer architecture, we present a parallel algorithm, implemented for both CPU and GPU, to efficiently compute molecular surface area. In addition to its parallel nature,

the algorithm is also progressive, providing a rough estimate of surface area very quickly and refining the estimate over time until the desired accuracy is reached. Finally, the algorithm generates points on the molecular surface, which can be used to create point-based renderings of the molecule.

2 Related Work

2.1 Molecular Surface Area Computation

Molecular surface areas have been computed through several different methods. The program MSMS [5] by Sanner constructs the Solvent Accessible Surface (SAS) [13] and Solvent Excluded Surface (SES) [14] by considering the intersections of spheres representing Van der Waals radii of atoms of the molecule, and using this information to compute a set of patches which make up the surface. The reduced surfaces it computes correspond to alpha shapes [15]. The program GETAREA [6] by Fraczkiewicz and Braun also calculates surface area by computing surface patches based on sphere intersections, making use of some additional ideas from computational geometry. A different type of approach was used by Wodak and Janin[16], who give a fast method to estimate molecular surface area using only distances between pairs of atoms.

Additionally, any program that computes triangulations of molecular surfaces, such as SURF [17] by Varshney *et al.*, can be easily converted to give an estimate of molecular surface area by adding up the areas of all the generated triangles. SURF is designed to take advantage of data-parallelism at the granularity of individual atoms, but cannot scale to take advantage of an unlimited degree of parallelism as our algorithm can. The SURF algorithm is also restricted to molecules defined as a collection of discrete atoms, while our algorithm can be applied to molecular surfaces defined in virtually any manner.

More recently, the program LSMS [7] by Can *et al.* discretizes atomic Van der Waals spheres onto a regular grid, and then uses the level-set method to propagate fronts to compute the SAS and SES; it can also compute the Solvent Excluded Volume (SEV). The program 3V [8] by Voss also discretizes the molecule onto a regular grid and computes area and volume, but does not use the efficient level-set algorithm of LSMS.

Another recent algorithm by Bajaj and Siddavanahalli [9] can compute several different molecular surfaces. Their work models atoms using signed distance fields, which are similar to the radial basis functions used in our work. However, our algorithms are very different— Bajaj and Siddavanahalli's algorithm builds up molecular surfaces incrementally on a grid by adding atoms one at

a time, while our algorithm measures surface area using parallel, stochastic sampling.

2.2 General Purpose GPU Computing

Although Graphics Processing Units (GPUs) were originally specialized hardware suitable only for 3D graphics computations, modern GPUs have evolved into general-purpose high-performance parallel processors. NVIDIA's G80 product line, for example, features 128 programmable processor cores and advertises a maximum performance of 300 gigaflops. These processors are programmed in an SPMD (Single Program, Multiple Datastream) fashion; all processors execute the same program, but are allowed to take different branches at conditional statements at the cost of a performance penalty. The high peak performance of GPUs relative to CPUs is largely due to the fact that GPUs devote a larger proportion of their transistors to arithmetic computation instead of tasks such as memory caching. Because of this architecture, GPUs perform best with algorithms that do a large amount of computation relative to their number of memory accesses; this type of algorithm is referred to as having high arithmetic intensity.

Modern GPUs have large amounts of on-card memory; first generation Tesla cards, for example, will have 1.5 GB of RAM. Historically, each processor on a GPU was only able to write its output to a single location in memory, corresponding to the pixel whose value that processor was computing. Modern GPUs have overcome this limitation and allow full read and write access to any location in memory from any processor. Additionally, the processors have access to a small pool of very fast shared memory which is suitable for communication between processors within the inner loop of an algorithm.

In the past, writing a general-purpose program for a GPU meant casting the algorithm in terms of graphics operations, such as texture look-ups and RGB color vector manipulations. With the recent advent of development environments such as NVIDIA's CUDA and ATI's CTM, however, general-purpose algorithms can be written in much more natural terms. CUDA, for example, is basically equivalent to the C language with a few extensions to facilitate the launching of parallel computation kernels.

Even though GPU algorithms can now be written in development environments similar to those used for CPU algorithms, developing an algorithm for a highly parallel architecture such as a GPU requires a different approach than developing for current CPUs. For an algorithm to run efficiently on a GPU, it must be divided into a large number (at least on the order of hundreds) of independent tasks which can be executed simultaneously. We note that our algorithm is ideal for this, since it is based on taking a large number of in-

dependent random samples. Care must also be taken to reduce main memory access as much as possible, and to take advantage of the available fast shared memory.

3 Gaussian Molecular Modelling

We calculate the surface area of a protein which is represented as the level set of a sum of Gaussian Radial Basis Functions (RBFs), with one RBF being placed at the location of each atom's center. This implicit molecular surface representation has been used as far back as Blinn's 1982 work [18], as well as in many more recent works such as Grant and Pickup[19], Ritchie[20], and Bajaj and Siddavanahalli [9].

Symbolically, each Gaussian RBF $\phi(x)$ can be represented as

$$\phi_i(x) = w_i \; e^{-\| \frac{x-\mu_i}{\sigma_i} \|^2} \tag{1}$$

where w_i is the weight of the ith RBF, μ_i is the location of the ith RBF's center, and σ_i controls the width of the ith RBF.

The program reads the same XYZR file format used by MSMS [5], which can be generated from PDB [21] files by the *pdb_to_xyzr* utility that comes with MSMS. Note that multiple atoms may be combined into a single entry in the PDB file (merging with hydrogens, for example), in which case the number of RBFs will be different than the number of atoms in the protein.

For the results reported in this work, we set the μ_i to the RBF centers in the XYZR file. RBF weights and widths are set based on the constants given in Ritchie[20] and Grant and Pickup[19], which are designed to model the Van der Waals surface of a molecule. Specifically, we set $w_i = 2.70$ for all RBFs and $\sigma_i = r_i/\sqrt{2.3442}$, where r_i is the RBF radius from the XYZR file. We form the overall scalar field by summing together all RBFs, and treat the surface as being at an isovalue of 0.259.

To accelerate sampling of the scalar field, we insert the RBFs into a bucketing spatial data structure. We partition space into a regular grid, and store a pointer to a list of the RBFs that overlap each grid cell at the corresponding element of a three-dimensional array. The Gaussian RBFs are truncated to zero at a radius of 3σ.

4 Stochastic Area Measurement

To measure molecular surface area we make use of the Cauchy-Crofton formula (equation 2) from integral geometry, which relates the area of a surface to the number of intersections with the surface of a set of lines. This formula can be written as

$$\int m \; dL = \pi s \tag{2}$$

where s is the surface area, m represents the number of intersections along a given line, and the integration is taken over the space of all possible lines.

A numeric approximation to this integral can be made by taking a random sample of lines and counting their intersections with the surface. Approximating the integral in this manner gives $\frac{n_1}{N} \approx c\pi s_1$, where n_1 is the count of intersections, N is the number of sampled lines, s_1 is the surface area, and c is an unknown constant of proportionality. To get rid of c, we can intersect the same set of lines with a second surface, giving $\frac{n_2}{N} \approx c\pi s_2$. Combining these equations gives

$$s_1 \approx \frac{n_1}{n_2} s_2 \tag{3}$$

If the area of the second surface s_2 is known, we can then calculate the molecular surface area s_1. This derivation is given in more detail in Li *et al.*[22], and further applications are discussed in Liu *et al.*[23].

4.1 Sampling the Space of Lines

Several methods for generating lines from randomly chosen parameters are given in Li *et al.*[22]. We use a method called the Chord Model, which consists of picking two random points from a uniform distribution of points on the surface of a sphere and then taking the line that passes through them. Uniformly distributed points (x, y, z) on a sphere can be generated from pairs (u, θ) of uniformly distributed random numbers by using the formula

$$(x, y, z) = ((1 - u^2)^{\frac{1}{2}} \cos\theta, \; (1 - u^2)^{\frac{1}{2}} \sin\theta, \; u) \tag{4}$$

where u is in $[-1, 1]$ and θ is in $[0, 2\pi)$. Further discussion of generating uniformly distributed points on spheres is given on the Mathworld web site [24].

To generate random lines with this method, we must generate uniformly distributed random numbers. This would typically be done using a pseudo-

random sequence; however, better results can be obtained by using a quasi-random sequence (also called low-discrepancy sequences). These sequences have less clustering of values than pseudo-random sequences, which results in a more representative sampling of lines and actually provides an asymptotically lower error bound for the numeric integration [22].

In our implementation we used the Niederreiter quasi-random sequence [25], which can be found in the GNU Scientific Library [26]. We generate 4D quasi-random points (a, b, c, d), and use the first and second coordinate pairs (a, b) and (c, d) to generate the (u_1, θ_1) and (u_2, θ_2) for equation 4.

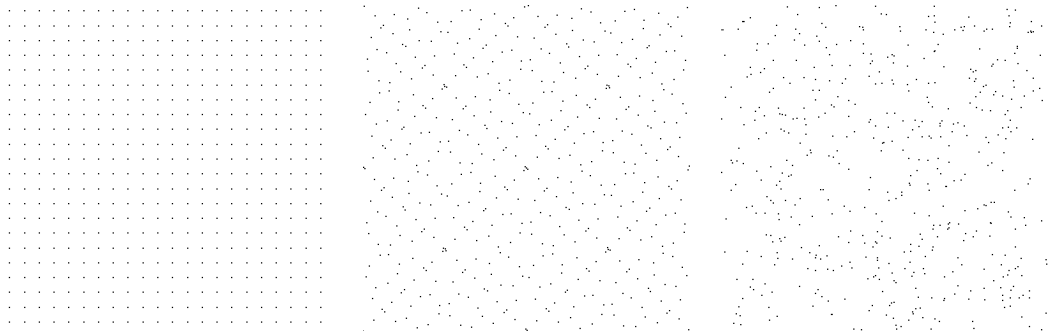

Fig. 1. 2D distributions of points generated on a regular grid (left), from the Niederreiter quasi-random sequence [25] (middle), and from a pseudo-random sequence (right).

Comparisons between 2D points generated from a pseudo-random, a quasi-random, and a regular grid distribution are given in figure 1. Note that the pseudo-random distribution has more clusters and bare regions than the quasi-random distribution. The regular grid distribution also avoids clustering, but is so regular that its use could cause aliasing artifacts. Further analysis of sampling points using quasi-random distributions can be found in Rovira *et al.* [27].

4.2 Intersection Counting

For our line intersection algorithm, we start by enclosing the RBFs representing the atoms in the tightest bounding sphere centered at the center of the molecule. One possible optimization would be to instead use the absolute tightest bounding sphere (without restriction on center location), perhaps computed by the method in Gartner [28].

We then generate a sequence of quasi-random lines using pairs of points on the surface of the bounding sphere as described above. For each line, we step in uniform increments from one point to the other, evaluating the scalar field at each step to determine whether the current point lies in the interior or exterior of the surface. The optimal step size is a function of the typical atomic radii

and packing densities. In this work we have used a step size of 0.25 Angstroms, which we have experimentally determined to be a reasonable value.

To evaluate the scalar field at a point, we iterate over all RBFs that overlap that point's bucket, adding their values to a running total until either all RBFs have been processed or the current total exceeds the surface's isovalue. If a point is found to be in the interior of the surface and the previous point was in the exterior (or vice versa), a running count of surface intersections is incremented.

Once the number of intersections of the lines with the isosurface has been computed, equation 3 can be used to estimate the surface area of the molecule (the area of the bounding sphere can be easily computed analytically, and the number of intersections with the bounding sphere is simply two times the number of lines intersected). The approximation improves as more lines are intersected.

5 Parallelization for GPU

Because the sampling along each line is completely independent from all other lines, this algorithm is a natural fit for a highly parallel architecture such as a GPU. In fact, our algorithm is able to linearly scale to take advantage of an unlimited amount of parallelism, since each additional available processor can be assigned to compute the intersections of another random line, increasing the speed at which the result converges.

We have implemented a version of the algorithm in NVIDIA's CUDA language that runs on a GPU, and compared its performance to the CPU version. In our GPU implementation the 4D quasi-random points that define the sample lines are generated on the CPU and then sent to GPU memory. After the per-line intersection counts are computed in parallel on the GPU, this data is sent back to CPU memory where the per-line counts are aggregated into an overall total. This process could potentially be optimized by computing the quasi-random points and performing the summation of the per-line counts on the GPU, which would not only take further advantage of the GPU's parallel processing capabilities but also avoid time-consuming data transfers to and from the GPU.

6 Test Results

Areas can be calculated for several different types of molecular surfaces. The Van der Waals surface is formed by a union of spheres located at the centers

of the molecule's atoms, with radii equal to the atoms' Van der Waals radii. The Solvent Accessible Surface (SAS) [13] is defined as the surface traced by the center of a probe sphere (representing a solvent molecule) as it is rolled along the Van der Waals surface. Finally, the Solvent Excluded Surface (SES) [14] is the boundary of the area that no part of such a probe sphere may penetrate. Programs can also calculate either the area of the outermost shell of the surface only, or include the area of any interior cavities as well.

For our tests, we set the parameters of our Gaussian RBF implicit surface representation to approximate the SES formed with a probe radius of 1.4 Angstroms. Our algorithm calculates the surface area of the outer surface as well as the interior cavities. We compare our results against several other programs that compute molecular surface area— MSMS [5], LSMS [7], and SURF [17].

Figure 2 illustrates how our algorithm converges on an estimate of the SES for the several proteins as increasingly more lines are intersected with the surface. In the remainder of the tests we set the number of intersected lines to 20,000, which we have found gives quick estimates with reasonable accuracy. The number of sample lines could be set higher or lower based on the speed and accuracy requirements of a particular application.

To evaluate the accuracy of our surface area computations, we would like to have some ground truth to compare our results against. Both MSMS and SURF compute the SES analytically, and their reported surface areas usually agree very closely. Therefore, we take the true SES area to be the average of the SES areas reported by MSMS and SURF, and measure algorithm accuracy as a percent difference from this average value. The differences between the area we report and this average area come from two main sources— our algorithm not having yet fully converged on the area of our surface, and the fact that the surface whose area we are converging on is not quite the same as the SES surface that we are comparing ourselves against.

As can be seen from table 1, our differences are comparable to the differences of LSMS when using a fine 256^3 grid, while our GPU running time is significantly faster than LSMS using a 256^3 grid, and is often even faster than LSMS using a coarse 128^3 grid. We observe that our running time depends mostly on the molecule size, while for LSMS the running time depends mostly on the grid resolution. Our GPU implementation is also faster than MSMS and SURF, especially for larger molecules. A graph of running times of the various algorithms is given in figure 3.

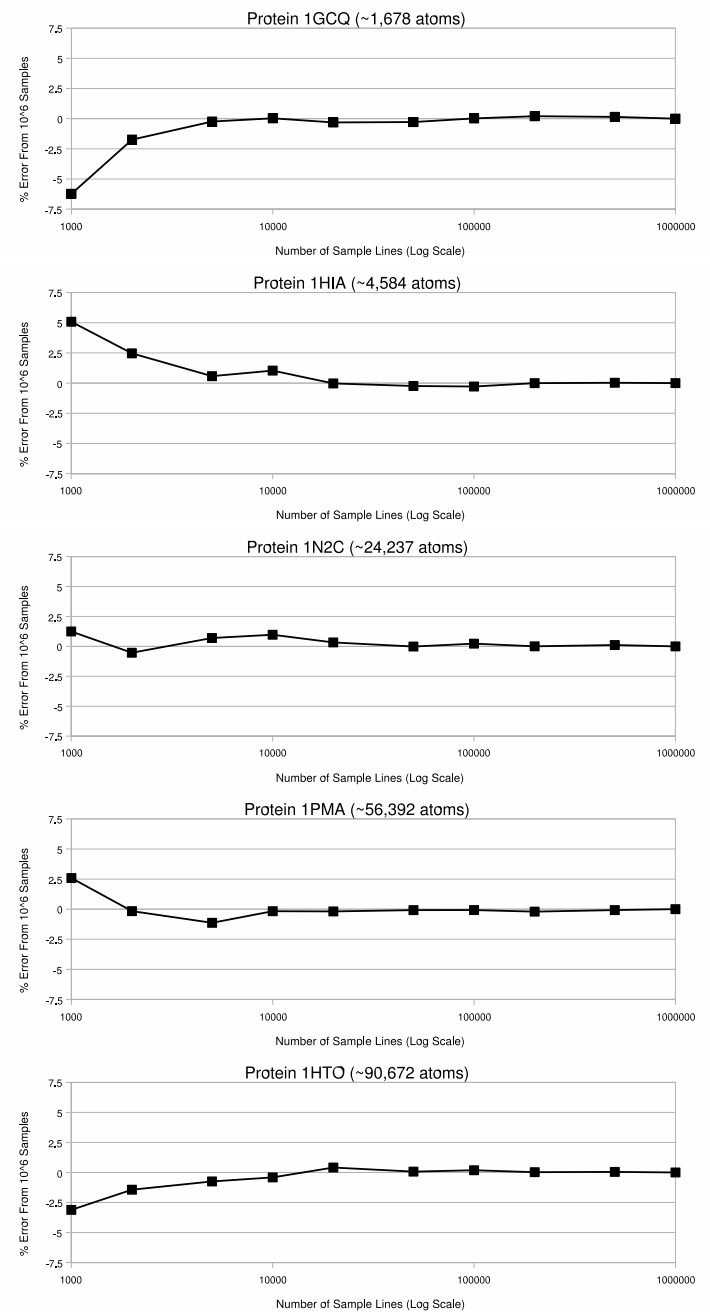

Fig. 2. Surface area approximation errors for several proteins using various numbers of intersected lines. Vertical axis is the percent error from the final value (the area returned with 10^6 sample lines). Note that all proteins have converged to near their final area by 20,000 lines.

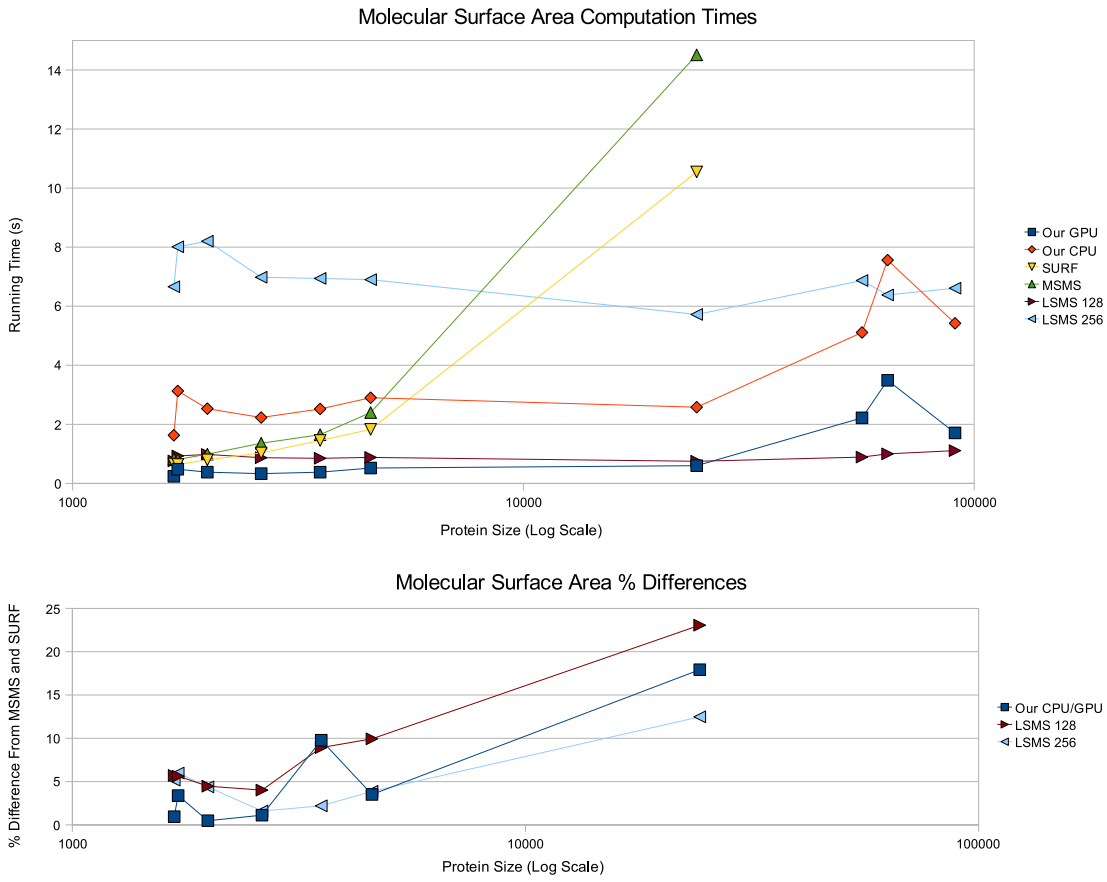

Fig. 3. Surface area computation times and percent differences for proteins of different sizes using several different algorithms. Data is from table 1. Protein size is given in RBFs, which is equal to the number of atoms listed in the PDB file. For MSMS and SURF, data is not available for the larger proteins since these programs were unable to compute the area for proteins of that size. Percent differences are given as absolute magnitudes.

Table 1:

Protein	1GCQ	2PTN	1PPE	8TLN	2CHA	1HIA	1N2C	1PMA	1FFK	1HTO
Size	1,678	1,712	1,991	2,621	3,542	4,584	24,237	56,392	64,268	90,672
Our Area (Å^2)	8,806.06	8,258.64	8,805.47	11,256.8	16,761.7	21,157.2	79,010.5	192,382	461,631	335,624
Area Diff	0.95 %	3.39 %	0.48 %	1.12 %	9.78 %	3.51 %	17.92 %	—	—	—
CPU Time (s)	1.63	3.13	2.53	2.23	2.52	2.90	2.58	5.11	7.56	5.42
GPU Time (s)	0.24	0.48	0.38	0.33	0.38	0.52	0.60	2.22	3.49	1.71
LSMS 128^3 Area (Å^2)	8,225.56	8,437.62	9,155.30	10,926.5	16,919.5	19,748.0	74,059.3	184,464	422,295	301,674
Area Diff	5.71 %	5.63 %	4.47 %	4.02 %	8.93 %	9.94 %	23.06 %	—	—	—
Time (s)	0.76	0.93	0.98	0.87	0.85	0.88	0.75	0.89	1.00	1.11
LSMS 256^3 Area (Å^2)	8,272.37	8,466.53	9,148.43	11,202.1	18,171.1	21,078.2	84,250.1	203,324	472,889	352,981
Area Diff	5.17 %	5.99 %	4.39 %	1.60 %	2.20 %	3.87 %	12.48 %	—	—	—
Time (s)	6.66	8.01	8.20	6.98	6.94	6.90	5.72	6.87	6.38	6.61
MSMS Area (Å^2)	8,724.65	8,039.77	8,807.21	11,364.4	18,538.6	21,944.8	97,129.4	—	—	—
Time (s)	0.83	0.81	0.99	1.36	1.65	2.40	14.51	—	—	—
SURF Area (Å^2)	8,722.10	7,935.88	8,719.81	11,404.0	18,619.9	21,909.9	95,388.7	—	—	—
Time (s)	0.66	0.64	0.80	1.03	1.45	1.83	10.55	—	—	—

Comparison of the surface areas and running times of our method, LSMS with a 128^3 grid, LSMS with a 256^3 grid, MSMS, and SURF. Protein size is given in RBFs, which is equal to the number of atoms listed in the PDB file. For our method, we used 20,000 sample lines. All methods computed the SES area (or an approximation to it) of all disconnected surface components using a probe sphere radius of 1.4 Å. Tests were performed on a machine with a GeForce 8800 GTX GPU, an Intel Xeon 3.0 GHz CPU, and 4 GB of RAM. A '—' means that MSMS or SURF was unable to compute a surface for this molecule.

7 Discussion and Future Work

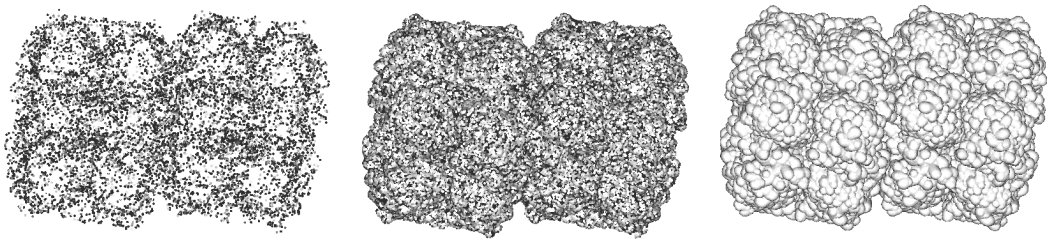

Fig. 4. Point-based renderings of the protein with Protein Data Bank ID 1HTO using 10^4 points (left), 10^5 points (middle), and 10^6 points (right).

Because our algorithm generates points on the molecular surface, it can easily be used to create point-based renderings [29] of the molecule. Surface normals for lighting calculations can also be easily generated by analytically computing the gradient of the implicit function at each surface point. Because of their light weight and simplicity, points are a good primitive for the representation of large models. Some point-based molecular renderings generated from our implementation using different numbers of points are shown in figure 4.

One nice feature of our algorithm that we have not explored is its progressive nature. As the algorithm runs, a rough approximation of the surface area is returned almost immediately, while increasingly accurate approximations are obtained as more and more line intersections are computed. This feature could be used to tune the speed versus accuracy of the algorithm for different applications, or to provide a rough estimate to decide whether or not more exact calculations are worth performing.

One possible area of future work might be to extend the program to compute other geometric properties of molecules, such as volume or mean curvature, as discussed in Schröder [30]. Molecular volume computation in particular would likely be an easy and useful extension.

One final issue worth mentioning is the treatment of hollow cavities within the interior of a molecule. Depending on the application, it may or may not be desirable to include interior cavity surface area in the overall surface area reported. A discussion of these interior cavities can be found in Liang *et al.* [31]. Our algorithm includes the surface area of these cavities in the final figures reported, as does SURF. MSMS and LSMS give an option to either include these areas or not. All surface areas and running times reported in this work are for the outer surface plus all cavities. If only the outer surface area is required, MSMS can compute this several times faster than it can compute the area of the outer surface plus all cavities.

8 Availability

The source code for our implementation is available on the GVIL web page at `http://www.cs.umd.edu/gvil/`.

9 Acknowledgements

We would like to thank Prof. Tolga Can for providing us with the source code and executable for LSMS, Prof. Chandrajit Bajaj for discussions regarding his work on molecular surface area computation and the TexMol molecular rendering program, Prof. Sergei Sukharev for discussions regarding applications of molecular measurement, Dr. David Luebke for being of great help getting us started with the early versions of CUDA, and the anonymous reviewers for providing constructive criticism of this work.

References

[1] O. Kreylos, N. Max, B. Hamann, S. Crivelli, W. Bethel, Interactive protein manipulation, in: Proceedings of IEEE Visualization, 2003, pp. 581–588.

[2] O. Lampe, I. Viola, N. Reuter, H. Hauser, Two-Level Approach to Efficient Visualization of Protein Dynamics, IEEE Transactions on Visualization and Computer Graphics 13 (6) (2007) 1616–1623.

[3] M. Connolly, Analytical molecular surface calculation, Journal of Applied Crystallography 16 (5) (1983) 548–558.

[4] M. Connolly, Solvent-accessible surfaces of proteins and nucleic acids, Science 221 (4612) (1983) 709.

[5] M. F. Sanner, A. J. Olson, J.-C. Spehner, Fast and robust computation of molecular surfaces, in: SCG '95: Proceedings of the eleventh annual symposium on Computational geometry, ACM Press, New York, NY, USA, 1995, pp. 406–407, http://www.scripps.edu/∼sanner/html/msms_home.html.

[6] R. Fraczkiewicz, W. Braun, Exact and efficient analytical calculation of the accessible surface areas and their gradients for macromolecules, Journal of Computational Chemistry 19 (3) (1998) 319–333, http://pauli.utmb.edu/cgi-bin/get_a_form.tcl.

[7] T. Can, C.-I. Chen, Y.-F. Wang, Efficient molecular surface generation using level-set methods, J Mol Graph Model 25 (4) (2006) 442–454.

[8] N. R. Voss, M. Gerstein, T. A. Steitz, P. B. Moore, The geometry of the ribosomal polypeptide exit tunnel, J Mol Bio 360 (4) (2006) 893–906, http://geometry.molmovdb.org/3v/.

[9] C. Bajaj, V. Siddavanahalli, An adaptive grid based method for computing molecular surfaces and properties, Tech. Rep. tr06-56, University of Texas (2006).

[10] C. Bajaj, P. Djeu, V. Siddavanahalli, A. Thane, Texmol: Interactive visual exploration of large flexible multi-component molecular complexes, in: VIS '04: Proceedings of the conference on Visualization '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 243–250.

[11] http://developer.nvidia.com/object/cuda.html.

[12] http://ati.amd.com/companyinfo/researcher/Documents.html.

[13] B. Lee, F. Richards, The interpretation of protein structures: estimation of static accessibility., J Mol Biol 55 (3) (1971) 379–400.

[14] F. Richards, Areas, Volumes, Packing, and Protein Structure, Annual Review of Biophysics and Bioengineering 6 (1) (1977) 151–176.

[15] H. Edelbrunner, E. Muecke, Three-dimensional alpha shapes, ACM Transactions on Graphics 13 (1994) 43–72.

[16] S. J. Wodak, J. Janin, Analytical approximation to the accessible surface area of proteins, Proc Natl Acad Sci USA 77 (4) (1980) 1736–1740.

[17] A. Varshney, F. P. Brooks, W. V. Wright, Linearly scalable computation of smooth molecular surfaces, IEEE Computer Graphics and Applications 14, No. 5 (1994) 19 – 25.

[18] J. F. Blinn, A generalization of algebraic surface drawing, ACM Trans. Graph. 1 (3) (1982) 235–256.

[19] J. A. Grant, B. T. Pickup, A gaussian description of molecular shape, Journal of Physical Chemistry 99 (11) (1995) 3503–3510.

[20] D. W. Ritchie, Evaluation of protein docking predictions using hex 3.1 in capri rounds 1 and 2, Proteins: Structure, Function, and Genetics 52 (1) (2003) 98–106.

[21] H.M.Berman, J.Westbrook, Z.Feng, G.Gilliland, T.N.Bhat, H.Weissig, I.N.Shindyalov, P.E.Bourne, The protein data bank, Nucleic Acids Research 28 (2000) 235–242, http://www.pdb.org/.

[22] X. Li, W. Wang, A. Bowyer, Using low-discrepency sequences and the crofton formula to compute surface areas of geometric models, Computer-Aided Design 35 (9) (2003) 771–82.

[23] Y.-S. Liu, J.-H. Yong, H. Zhang, D.-M. Yan, J.-G. Sun, A quasi-monte carlo method for computing areas of point-sampled surfaces, Computer-Aided Design 38 (1) (2006) 55–68.

[24] E. W. Weisstein, Sphere point picking, From MathWorld– A Wolfram Web Resource, http://mathworld.wolfram.com/SpherePointPicking.html.

[25] P. Bratley, B. L. Fox, H. Niederreiter, Implementation and tests of low-discrepancy sequences, ACM Trans. Model. Comput. Simul. 2 (3) (1992) 195–213.

[26] Gnu scientific library, http://www.gnu.org/software/gsl/.

[27] J. Rovira, P. Wonka, F. Castro, M. Sbert, Point Sampling with Uniformly Distributed Lines, Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings (2005) 109–118.

[28] B. Gärtner, Fast and robust smallest enclosing balls, in: ESA '99: Proceedings of the 7th Annual European Symposium on Algorithms, Springer-Verlag, London, UK, 1999, pp. 325–338.

[29] M. Gross, H. Pfister (Eds.), Point-Based Graphics, Morgan Kaufmann, 2007.

[30] P. Schröder, What can we measure?, in: SIGGRAPH '06: ACM SIGGRAPH 2006 Courses, ACM Press, New York, NY, USA, 2006, pp. 5–9.

[31] J. Liang, H. Edelsbrunner, P. Fu, P. Sudhakar, S. Subramaniam, Analytical shape computation of macromolecules: II. Inaccessible cavities in proteins, Proteins Structure Function and Genetics 33 (1) (1998) 18–29.