Eurographics Symposium on Geometry Processing (2006), pp. 1–8
Konrad Polthier, Alla Sheffer (Editors)

Surface Area of the Union of Many Spheres

submission id 201

Abstract

We describe an algorithm and implementation for computing the surface area of the union of many spheres. By comparing the areas of the unions of different groups of spheres with different radii we can compute the buried surface area of the solvent accessible surface of two molecules. This can be used to efficiently evaluate many possible protein-ligand docking configurations.

The key component of our algorithm is to use local topological formulae to compute the surface area. They have been used to compute any mass property, such as volume or surface area, of a polyhedron as follows. For each such property, there is a function of a vertex's location and local geometry. "Local geometry" means the directions of the edges and faces adjacent on that vertex. It does not include any global information, such as edge length or other vertices of such faces. Summing that function over all the polyhedron's vertices gives the corresponding mass property. The current work concerns modifying these ideas for spheres, which lack vertices. The impact of these formulae arises because computing vertices and their neighborhoods of the union of many objects is much easier than computing the higher level subfacets, such as edges and faces. Also, the global topology of the result, such as the number of components and their containment relations, is irrelevant.

An initial implementation processes 2451 atom molecules in an average of 10 seconds to an accuracy of 0.05% on a laptop.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

We wish to find the area of the surface of the union of intersecting spheres. The application is computing the solvent accessible area of a protein.

There appear to be few results for object combination in 3-D, except for the following special cases: efficiently intersecting convex polyhedra [CD87, Cha89, MP78, Sug94], intersecting a convex polyhedron with a general one [DMY93, MS85], fast detection of polyhedral intersection [DK83], and uniting convex polyhedra [AST97]. Constructive Solid Geometry (CSG) is well documented, [Cam91, Epp92, LTH86], though its efficient methods tend to use bounding boxes.

[ZE02] presents a fast algorithm for intersecting boxes in 3D. Various heuristics are combined to optimize the implementation, for which tests on 500,000 boxes were reported. Although the general optimization techniques are very useful, extending this particular algorithm to compute properties of a union is not at all obvious since the characteristics of intersections and unions of boxes are quite different. E.g., intersections are convex, while unions are often not even connected. [Eri03] computes any Boolean combination of two *local* polyhedra in $O(N \log N)$ time. However since the faces of a union are often not simplices, our objects are not local. [Vig] gives a new algorithm to report intersecting pairs among n objects in a fixed dimension d. Again, this is considerably easier than computing unions.

In E^2, the well-known plane sweep methods for line segment intersection take from $T = \theta(K + N \log N)$ for a complex topological sweep up to $T = \theta((N+K)\log N)$ for a simple plane sweep. N is the number of input line segments, and K the number of output intersections. For large values of N, the $(\log N)$ is significant. Those methods' worst-case time is much better than ours, since an adversary can hurt the performance of our "input-sensitive" method. Also our computation model is more powerful than that used by the plane sweep methods; we can compute *modulo* in constant time.

The prior art on mass properties of boolean combinations of polyhedra is similarly sparse, [BN83, Mir96]. Some of

these local topological formulae could well date to the discovery of analytic geometry, [Des44], but actual citations earlier than [Fra87, NF91, FCK*90] are elusive. Large geometric and cartographic data sets are now being processed, perhaps with external algorithms, [TWA*01, Cen01].

Our longterm problem is to compute mass properties, including second order moments, center of gravity, volume, area, and edge length, of a very complex solid defined by a constructive solid geometry tree. The immediate goal is to demonstrate a solution to a restricted subproblem, where the only operation is *union*, and the primitive objects are spheres, and to demonstrate it on large input examples.

Franklin [Fra04] and [Fra05a] presents a related, simpler, problem: finding mass properties such as surface area and volume of the union of identical isothetic cubes. The largest test case processed $2 \cdot 10^7$ cubes, containing $5 \cdot 10^8$ subfacets, in an hour on a dual Xeon. The time depends on the cubes' sizes. In this case, their common edge length was 0.009 of the universe size. The grid size was 500x500x500 cells. The implementation also scales down; the volume and area of the union of 10,000 cubes of edge length 0.03 were computed in 2 seconds.

2. Nature of the Surface

The surface of the union of spheres may be partitioned into regions that are not exactly spherical triangles. Each region has 3 sides: arcs of 2 great circles, and an arc of a small circle. An example of a small circle is a parallel of latitude that is not the equator or a pole.

Unfortunately it appears that (except for special cases) the area of such a region has no closed form in terms of the usual elementary functions. Therefore, the area must be approximated. This raises the question of where in the computation process should the approximation occur. Here are some possibilities.

1. Approximate each sphere as a union of (possibly intersecting) cubes, and use [Fra05a] for the area of the union of cubes, which has been tested on $2 \cdot 10^7$ cubes. However, since the cubes' surface normals don't approach the spheres', neither would the area. Also, if we want to go this route, octrees, i.e., nonintersecting cubes, would be simpler.
2. Approximate each sphere by a polyhedron, which is a tradition in Computer Graphics. However, this uses an approximation in 2 dimensions, while we need to approximate in only 1. More approximation than necessary seems suboptimal.
3. Approximate the small circle's arc by a sequence of little pieces of great circles. This would work, but the next idea is better.
4. Build upon our procedure in 2D for finding the perimeter of the union of circles. This is better than the previous idea since it doesn't try to approximate each spherical non-triangle, but only their sum. The law of large numbers will be in our favor.

3. Why Not Use Global Topology?

The intersection process directly produces only local topological information about the output spherical curved triangles. It may look easy to link up this local topological info to find their boundary and construct a closed surface representation. In practice, it is not easy, even in E^2, and is much worse in E^3. Our estimate is that doing this would double the number of lines of code, more than double the execution time (since exact computation would now be required), and introduce assorted nasty special cases, with the variety of possible manifold and non-manifold instances that would need correct handling. Nevertheless, if this is desired, a sketch of the process goes as follows, to combine a set of isolated vertex neighborhoods into edges and faces, then into loops and shells. The result might be in the form of the very useful data structures in [GS83, Bri89], except that variable-length arrays are more efficient than linked lists.

1. Each output vertex neighborhood forms the *end* and *direction* of one or more edges. Compute these edge ends, as a set $\{(P, \hat{D})\}$, where P is the endpoint of an edge, and $\hat{D}$ is the unit vector along it.
2. Since the two ends of one edge must be on the same small circle, group those ends according to whether they are on the same small circle.
3. For each small circle: if there are exactly two ends on it, and they point to each other, then we have an edge. If there are an even number greater than two ends, then sort them by angle, and group them into adjacent pairs, each forming one edge. Otherwise, if there are an odd number of ends, then report an inconsistency. When sorting, handle the special case of angles wrapping from 2π around to 0.
4. Each output vertex neighborhood also forms a vertex of one or more output faces. Compute those vertices.
5. Group the vertices according to the spheres that they are incident on. This is well defined since each vertex knows the directions of its incident edges.
6. For each sphere found in the previous step, identify any edges that are incident on it.
7. In each sphere, connect up the vertices and edges on it into faces, while checking for inconsistencies.
8. If there is more than one face per sphere, then determine the inclusion relations among them.
9. Group the faces into shells.

The actual process is worse than is apparent from the above brief summary. Our experience is that the much simpler case of combining only two polygons in E^2 requires perhaps 1000 lines of code, of quality such that testing point inclusion in a polygon is only 10 lines. That is why we prefer to avoid global topology.

4. Perimeter of the Union of Circles in E^2

We will present the 2D case first, since the full 3D problem builds on it. Given $\mathcal{C} = \{c\}$, a set of circles with various radii, the problem is to compute $\mathcal{P}$, the perimeter of their union. We will not determine the individual visible arcs, or any other global topology, such as nested voids. This routine returns only the sum of the arcs' measures.

1. The perimeter is a set of arcs of the input circles.
2. An input circle that is completely visible generates an arc of size 2π.
3. Otherwise, each arc's ends are intersections of its circle with other circles.
4. Exactly the circle-circle intersections that are *visible*, i.e., not contained inside another circle form arc endpoints.
5. l_i, the length of a_i, arc i, is $e_i - s_i$ where e_i and s_i are the angles of a_i's endpoints relative to its center.
6. That requires that e and s be normalized into the range $[0, 2\pi)$.
7. That also requires that 2π be added if the positive x-axis passes through the arc.
8. $\mathcal{P} = \sum l_i = \sum e_i - \sum s_i$ does not require that the corresponding starting and ending angles of the arcs be associated with each other. That allows a major optimization as follows:
   a. Find $\mathcal{V} = \{v\}$, the set of all visible intersections of any two circles. Initially, do this exhaustively. Later, use a grid so that only pairs of circles likely to intersect are tested.
   b. Consider one v, formed by the intersections of circles c_1 and c_2, with respective radii r_1 and r_2. v is the start of an arc on one circle and the end of an arc on the other circle. Compute α_1 and α_2, the angles of v relative to the centers of c_1 and c_2.
   c. If $\alpha_1 - \alpha_2$, normalized into the range $[-\pi, \pi)$, < 0, then v is the end of an arc of c_1 and also the start of an arc of c_2. Otherwise, v is the end of an arc of c_2 and also the start of an arc of c_1.
   d. Normalize α into the range $[0, 2\pi)$ and add $r_1\alpha_1 - r_2\alpha_2$ to the perimeter subtotal.
   e. However, if either arc includes a ray from its circle's center to $(+\infty, 0)$, then the arclength is wrong by $2\pi r$.
   f. Also, circles with no intersections, and which are visible, also require $2\pi r$ be added.
   g. Fortuitously, because of the judicious use of angle normalizations above, these two cases collapse into one:
      i. Test the farthest right point of each circle.
      ii. If it it visible, then add $2\pi r$ to the perimeter.

5. Surface Area of the Union of Spheres

1. Conceptually compute $\mathcal{A}$, the surface area of the molecule $\mathcal{M}$, thus:

$$\mathcal{A} = \int a(z)dz$$

where the component of the area from z to $z + dz$ is $a(z)dz$.

2. Since there is apparently no closed form for $a(z)$, evaluate $\mathcal{A}$ with a Gauss-Kronrod adaptive quadrature, [Fre07, Wei07].
3. Let s be a section of the molecule, sliced at z. It is composed of a set of arcs, a_i. Arc a_i is slanted at an angle θ_i relative to the z-axis. Then

$$a(z) = \sum \frac{a_i}{\cos\theta_i}$$

4. However, since we are summing arc end angles instead of the actual arcs, which we don't have, weight the end angles by $1/\cos\theta_i$.

6. Gridding for Efficiency

The above process takes $\Omega(N)$ time, depending on the actual number of intersections. We may reduce it with a uniform grid, [Fra05b], [Fra80], as follows.

1. Superimpose a 3D grid onto the scene, whose cells have size slightly greater than the maximum sphere diameter.
2. When searching for intersections with a given sphere, or when testing whether a point is hidden, test only spheres in the $3 \times 3 \times 3$ block of cells around that sphere.
3. If any cell is completely contained in some sphere, then mark that cell as *covered*. Do not do any intersection tests in that cell.

7. Time Analysis

This algorithm is interesting only if it is efficient. This section will show that, if the spheres are independently and uniformly distributed, then the expected time to compute the area of the union is linear. Let

$N \triangleq$ number of input spheres

$R \triangleq$ sphere radius, on a scale where the universe is $1 \times 1 \times 1$

$G \triangleq$ number of grid cells on a side

The time analysis will be presented in several steps; starting with the analysis of the time to find all the intersections in E^2, ignoring the possibility of covered cells.

7.1. Circle Intersections in E^2, Ignoring Covered Cells

If the circles are independently and uniformly distributed, and $R \ll 1$ (to minimize effects near the border of the universe), then the expected number of cells incident on a given circle is $\alpha(LG+1)$, where α is slightly less than one. Therefore the average number of circles incident on any given cell is $\frac{N}{G^2}(LG+1)$. Since the circles are independent, the number of circles incident on a cell is a Poisson distributed random variable, call it η, with parameter $\lambda_\eta = \frac{N}{G^2}(LG+1)$.

In each cell, every circle must be tested against every other

circle for intersection, for an expected $\overline{(\eta^2-\eta)}$ tests (ignoring constant factors). Since $\overline{\eta}=\lambda_\eta$ and $\overline{\eta^2}={\lambda_\eta}^2+\lambda_\eta$, then the expected number of intersection tests per cell is ${\lambda_\eta}^2=\frac{N^2}{G^4}(LG+1)^2$, and the expected total number of intersection tests, over the G^2 cells, is $\frac{N^2}{G^2}(LG+1)^2$.

The total time is proportional to the number of circles inserted into all the cells plus the number of intersection tests (ignoring insignificant setup times proportional to the number of circles plus number of cells), or $T=\Theta\left(N(RG+1)+\frac{N^2}{G^2}(LG+1)^2\right)$. This is minimized when $G=\Theta(1/R)$, giving $T=\Theta\left(N+N^2R^2\right)$. Setting $S_i \triangleq$ size of the input $=\Theta(N)$, and $S_o \triangleq$ size of the output (the expected number of intersections) $=\Theta\left(N^2R^2\right)$, this gives

$$T=\Theta\left(S_i+S_o\right),$$

which is optimal for finding all intersections.

7.2. Circle Intersections in E^2, Using Covered Cells

This section introduces the covered cell concept, still working in E^2. It will show that not intersecting the circles in the covered cells reduces the total number of circle intersection tests from $\Theta(N^2L^2)$ to $\Theta(N)$. The definition of the output changes from all intersections to all intersections in cells that are not covered. The covered intersections are unnecessary since they cannot be vertices of the union.

The expected number of cells incident on a given circle is $k=(LG+1)^2$.

The expected number of cells covered by a given circle is $\max(LG-1,0)^2$. Therefore the probability that a given circle incident on a cell covers that cell is $q \triangleq \frac{\max(LG-1,0)^2}{(LG+1)^2}$. Setting $G=c/R$, for some constant $c>1$ reduces q to a constant. E.g., if $c=2$, then $q=1/9$ and $k=9$. The expected number of circles incident on a given cell is $\lambda \triangleq kN/G^2=\frac{9}{4}NR^2$. The probability of a given cell not being covered by any of the i circles incident on it, for $i\geq 0$ is $(1-q)^i$.

Let r be the number of *visible* circles per cell, defined as the number of circles incident on a given cell if it is not covered, or 0 if it is. Since the number of circles per cell follows a Poisson distribution with mean of λ, the number of visible circles per cell has this probability:

$$p(r) = e^{-\lambda}\frac{\lambda^r}{r!}(1-q)^r, \quad \text{for } r\geq 1 \tag{1}$$

$$p(0) = 1-\sum_{i=1}^{\infty} p(i) \tag{2}$$

Thus

$$\overline{r}=\lambda(1-q)e^{-\lambda q}=2NR^2e^{-NR^2/4}. \tag{3}$$

It attains its maximum of $8e^{-1}$ when $NR^2=4$. Thus, even though the number of circles per cell might be arbitrarily large, the expected number of *visible* ones is a small constant.

The expected number of intersection tests performed per cell is

$$N_{tpc} \leq \overline{\binom{2r}{2}} = 2R^2N\left(1+4R^2N\right)e^{-\frac{R^2N}{4}}$$

Since $\lambda^2e^{-\lambda}\leq 4e^{-2}$ and $\lambda e^{-\lambda}\leq e^{-1}$, it is easy to show that $N_{tpc}=\mathcal{O}(1)$, that is, is bounded above by a constant, but a tighter bound is possible. The total number of intersection tests over the $G^2=9R^{-2}$ cells is

$$N_{tt}=9R^{-2}N_{tpc}=18N\left(1+4R^2N\right)e^{-\frac{R^2N}{4}}<18(1+16e^{-1})N$$

That bound is not tight; and the above constant factors could be reduced.

Note the surprising effect of the covered cells: the number of visible intersections is reduced from $\Theta(N^2R^2)$ to $\Theta(N)$. Intuitively, why is this so? Only intersections around the perimeter of a group of overlapping polygons count, since those in the middle are covered. The size of the perimeter is $\Theta(N)$.

With $S_o \triangleq$ size of the output (the expected number of intersections) $=\Theta(N)$, the execution time

$$T=\Theta\left(S_i+S_o\right),$$

which is optimal for finding all intersections in cells that are not covered.

7.3. Point Classification Tests in E^2, Using Covered Cells

The mass property formulae require knowing which circle intersections are outside all the circles. How fast can point $\mathcal{P}$ be processed? The first step is to find which cell, $\mathcal{C}$, contains $\mathcal{P}$. If $\mathcal{C}$ is covered, then $\mathcal{P}$ is inside some circle, and can be ignored. On the other hand, if $\mathcal{C}$ is not covered, if $\mathcal{P}$ is inside a circle, then that circle must pass through $\mathcal{C}$. Comparing $\mathcal{P}$ against all the circles passing through $\mathcal{C}$ suffices to classify $\mathcal{P}$. However, the expected number of circles incident on a non-covered cell is constant, from equation 3. Therefore the expected time to classify each point is

$$T=\Theta(1),$$

which is optimal.

7.4. Point Classification Tests in E^3, Using Covered Cells

This analysis is identical to the E^2 case, if the grid cell size continues to be proportional to, and larger than, the sphere diameter. As the average number of spheres incident on a given cell rises, the probability that at least one of them will cover the cell also rises. Thus the expected number of spheres incident on a non-covered cell tends to a constant.

Let $G = 4/R$, so the number of cells is $8R^{-3}$. The expected number of cells incident on a given sphere is about 27, giving a total of $27N/8$ such incidences, or $\lambda = \frac{27}{64}NR^3$ per cell. The actual number of spheres incident on a given cell follows a Poisson distribution. Since the probability of a sphere that is incident on a given cell covering the cell is $q = 1/27$, r, the *visible* number of spheres per cell follows equation 1 with these q and λ, and so $\bar{r}$ is also a small constant here.

Classifying point $\mathcal{P}$ against all the spheres in a non-covered cell, to see whether any of those spheres contain $\mathcal{P}$, requires testing $\mathcal{P}$ against all the spheres incident on that cell. Since the expected number of those is constant, the expected time to classify a point is constant.

7.5. Intersection Determination in E^3

This section extends the timing analysis to visible intersections of 3 spheres in E^3. Only intersections that are outside all the spheres are interesting.

Consider the 3-sphere intersections first. Each of the spheres is incident on 9 cells, so the number of spheres incident on a cell follows a Poisson distribution, with $\lambda = c_1NR^3$. When a sphere is incident on a cell, from 0 to 3 of its spheres are also incident on that cell. The probability that a sphere that is incident on a cell covers that cell is $q = c_2$. If a cell is covered then define the number of visible spheres to be 0, else it is the number of spheres. Therefore, equation 1 still holds, with these new values of λ and q.

When there are r spheres in a cell, at most $\binom{r}{3}$ triples must be tested for intersection. Since it can be shown that

$$\overline{\binom{r}{3}} = c_3\lambda^3 e^{-\lambda/27}$$

the number of tests, summed over the $8R^{-3}$ cells, is

$$N_{ft} \triangleq c_4R^6N^3e^{-c_6R^3N}$$

Since $R^6N^3e^{-c_1R^3N} = c_2Nw^2e^{-w} \leq c_3N$, where $w = c_1R^3N$,

$$N_{ft} \leq \frac{c_7}{e^2}N$$

The constant factor is included to show that, while it is not on the order of 1, it is workable. Therefore the total time to find the visible vertices is

$$\Theta(N)$$

which is optimal.

8. Application: Protein Docking

Our fast surface area calculation can be used for the detection of collisions between rigid objects that are each defined as the union of simple primitives. The technique involves comparing the total surface area of the objects in a known, uncollided state to the surface area of the union of the objects in the query state. If the areas are equal (within the numerical tolerance of the implementation) then the objects do not intersect.

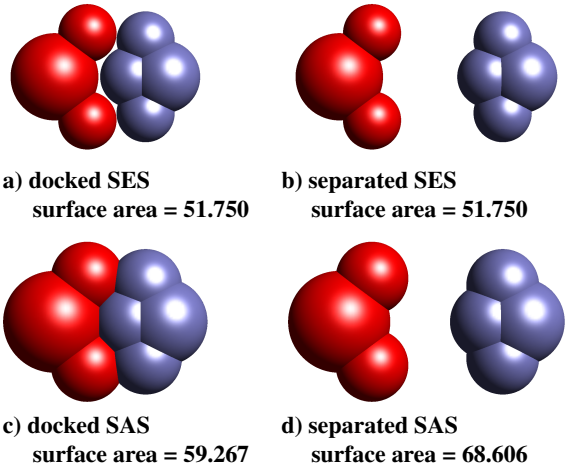

Figure 1: *The buried area configuration scoring metric rewards poses that have no intersection between the Solvent Excluded Surfaces (SES) of the two molecules, but maximal intersection of the Solvent Accessible Surface (SAS). In this example, the surface area computations confirm no intersection of the SES, and a buried surface area of 68.606 - 59.267 = 9.339 Angstrom^2.*

Since our algorithm can efficiently process millions of primitives, we propose to use efficient unioned surface area calculation for collision detection in the innermost loop of a computational protein docking application. This active and well-studied problem from molecular biology is computationally-intensive due to the enormous number of potential molecule configurations. To test the robustness and efficiency of our algorithm, we demonstrate the implementation on sample protein structures from the Protein Data Bank [aJWFG*00]. Each molecule in the database is defined by a set of atoms and their relative positions. Each atom can be modeled as a small sphere with radius equal to its measured atomic radius. The Protein Docking Benchmark [CMJW03] contains examples of known dockings between pairs of molecules. In Figure 2 we show a sample protein-ligand docking from the benchmark.

A computational protein docking system consists of several components:

- The generation of potential docking configurations;
- A scoring function by which to evaluate a particular configuration;
- A method of efficiently evaluating that scoring function; and
- A ranking all possible dockings.

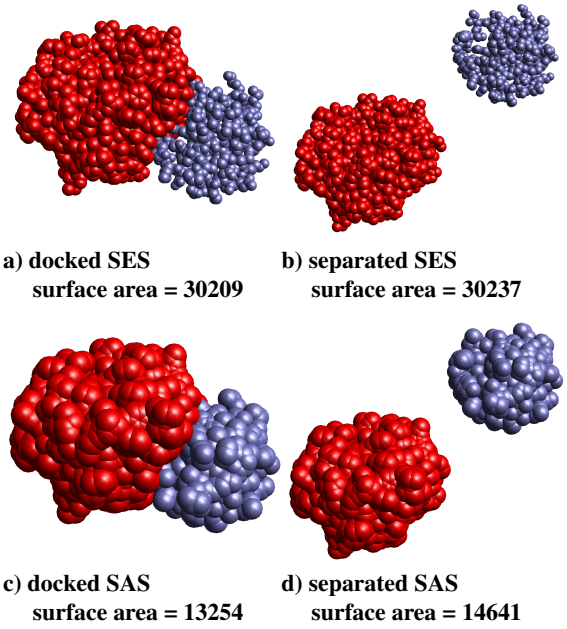

Figure 2: *Protein-ligand docking 2SNI from the Protein Docking Benchmark [CMJW03]. The protein receptor molecule (red) contains 1938 atoms and the ligand molecule (blue) contains 513 atoms. The surface area computation confirms that the SES of the molecules do not collide and computes the buried surface area to be 1387 Angstrom^2.*

A common calculation used in the scoring functions of many protein docking systems is the *buried surface area* between the molecules' *solvent excluded surfaces (SES)* and *solvent accessible surfaces (SAS)* [Con83, Ras84]. The SES is defined by the union of the spheres that represent each atom. The van der Waals radii of these atoms is 1.8-2.2 Angstroms, depending on the atom type. The solvent accessible surface is computed as an outward offset from the SES — typically 1.4 Angstroms, the radius of a water molecule. The best docking is then defined to be the configuration that maximizes the overlap between the solvent accessible surfaces, while preventing collision between the solvent excluded surfaces. Intuitively, this means that the molecules are close (but not touching) and well-aligned to minimize their surface exposure to solvent molecules. We adopt this common scoring function and demonstrate our union of spheres surface area algorithm to efficiently evaluate this metric. A toy example is shown in Figure 1 and an example with protein-ligand pair from the Protein Docking Benchmark is shown in Figure 2.

In prior art, the SES and SAS of the molecule must be explicitly constructed and represented as a triangle mesh [SaHBZ07] or rasterized into a 3D voxel grid [CLZ03]. Note that computation and storage of these surfaces is non-trivial (typically the surfaces are created as a pre-process). Furthermore, once computed, the molecule must be treated as a rigid body. The novelty of our approach is that the explicit construction and representation of these surfaces is unnecessary. We simply compute the unioned object surface area for the four cases shown in Figures 1 and 2. In the separated cases we compute the unioned object surface areas of the two molecules separately. For the SES cases we use the van der Waals radius for each atom (1.8-2.2 Angstrom) and for the SAS cases we use the van der Waals radius plus 1.4 Angstroms.

The program's time depends on the desired error, which affects the number of iterations taken by the adaptive quadrature. In practice, the actual error, as measured by rerunning the program with a smaller desired error is much better than the quadrature routine's claimed error. The number of iterations varies considerably between different molecules with the same number of atoms, depending on the variability of the cross sectional perimeter.

For the test docking configuration shown in Figure 2 with 2451 atoms (spheres), we were able to compute each of the four unioned surface area cases in an average of 11 seconds on an IBM T43p laptop. The actual error was about 0.5%. Improving the error by a factor of 100 increased the execution time by about a factor of 20. Currently, the fastest rigid molecule docking algorithms — ZDock [CLZ03] and Context Shapes [SaHBZ07] perform on the order of 100,000 configuration evaluations in an hour to generate a ranking of potential dockings between two rigid molecules. These implementations have been optimized to perform local rather than global complementarity of the molecules and these running times do not include precomputation of the SES and SAS. Furthermore, these techniques assume rigid molecules and performance will be significantly degraded if extended to allow flexible docking. Often the molecules' flexibility is critical to finding and evaluating the best docking configuration [WKK99].

We plan to continue to pursue this application of our fast union surface area technique. To improve performance, we plan to use the grid, which is not yet in our current implementation, and expect a considerable speed improvement. Currently, we start with one quadratic step to find all pairs of intersecting atoms. Then the adaptive quadrature intersects only pairs of circles whose spheres intersect.

We will also investigate the potential usefulness of other mass properties such as volume, center of gravity, etc in molecular biology.

9. Summary

This paper has several points. First, when operators such as union and volume are composed, sometimes optimizations are possible. Second, less explicit topology can be better. Third, simple data structures are well suited for processing large geometric datasets in E^3.

These techniques would also support other operations, not

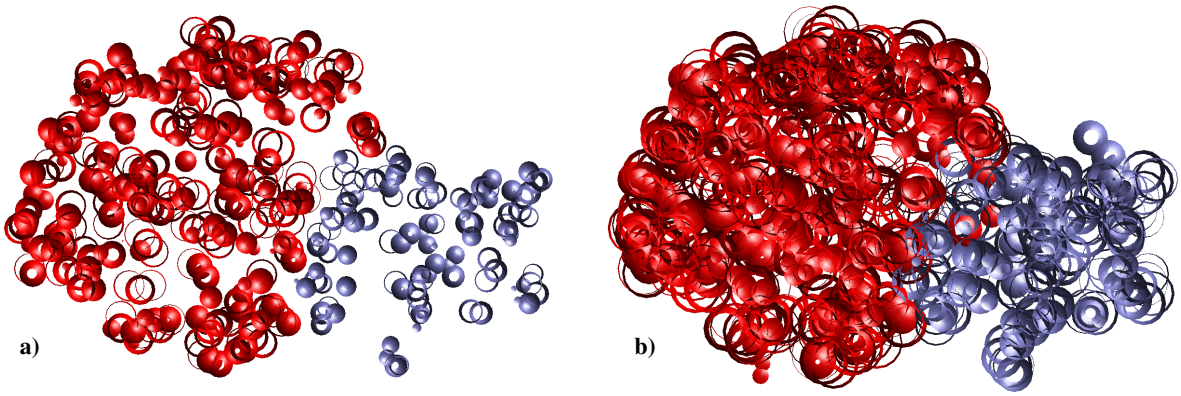

Figure 3: *To visualize the complexity of the docking shown in Figure 2 we use two clipping planes parallel to the z axis approximately 1 Angstrom apart. Note that a) no red sphere intersects any blue sphere in the solvent excluded surface, while b) the solvent accessible surfaces contain significant overlap.*

yet implemented, such as online computation of mass properties, as spheres are inserted and deleted. Insertion is easy. Deletion requires identifying the candidate output vertices that are now visible since they were hidden only by the deleted sphere. Computing properties of more complicated boolean operations is feasible. Producing the explicit output sphere is also possible, though considerably more complicated. Extensions to collision detection of moving objects, each composed of the union of many simple spheres, are now being considered.

References

[aJWFG*00] AMD J. WESTBROOK H. B., FENG Z., GILLILAND G., BHAT T., WEISSIG H., SHINDYALOV I., BOURNE P.: The protein data bank. *Nucleic Acids Research*, 1 (2000), 235–242.

[AST97] ARONOV B., SHARIR M., TAGANSKY B.: The union of convex polyhedra in three dimensions. *SIAM J. Comput. 26* (1997), 1670–1688.

[BN83] BIERI H., NEF W.: A sweep-plane algorithm for computing the volume of polyhedra represented in boolean form. *Linear Algebra and its Applications 52–53* (1983), 69–97.

[Bri89] BRISSON E.: Representing geometric structures in *d* dimensions: Topology and order. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.* (1989), pp. 218–227.

[Cam91] CAMERON S.: Efficient bounds in constructive solid geometry. *IEEE Comput. Graph. Appl. 11*, 3 (1991), 68–74.

[CD87] CHAZELLE B., DOBKIN D. P.: Intersection of convex objects in two and three dimensions. *J. ACM 34*, 1 (Jan. 1987), 1–27.

[Cen01] CENTER FOR GEOMETRIC AND BIOLOGICAL COMPUTING: External memory algorithms and data structures. http://www.cs.duke.edu/CGC/subjects/external.html, 2001.

[Cha89] CHAZELLE B.: An optimal algorithm for intersecting three-dimensional convex polyhedra. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.* (1989), pp. 586–591.

[CLZ03] CHEN R., LI L., Z Z. W.: Zdock: An initial-stage protein-docking algorithm. *Proteins 52* (2003), 80–87.

[CMJW03] CHEN R., MINTSERIS J., JANIN J., WENG Z.: A protein-protein docking benchmark. *Proteins Proteins* (2003), 88–91.

[Con83] CONNOLLY M. L.: Solvent-accessible surfaces of proteins and nucleic acids. *Science* (1983), 709–713.

[Des44] DESCARTES R.: *Principia Philosophiae*. Ludovicus Elzevirius, Amsterdam, 1644.

[DK83] DOBKIN D. P., KIRKPATRICK D. G.: Fast detection of polyhedral intersection. *Theoret. Comput. Sci. 27*, 3 (Dec. 1983), 241–253.

[DMY93] DOBRINDT K., MEHLHORN K., YVINEC M.: A complete and efficient algorithm for the intersection of a general and a convex polyhedron. In *Proc. 3rd Workshop Algorithms Data Struct.* (1993), vol. 709 of *Lecture Notes Comput. Sci.*, pp. 314–324.

[Epp92] EPPSTEIN D.: *Speed-ups in constructive solid geometry*. Report 92-87, Dept. Inform. Comput. Sci., Univ. California, Irvine, CA, 1992.

[Eri03] ERICKSON J.: Local polyhedra and geometric graphs, 2003.

[FCK*90] FRANKLIN W. R., CHANDRASEKHAR N.,

KANKANHALLI M., AKMAN V., WU P. Y.: Efficient geometric operations for CAD. In *Geometric Modeling for Product Engineering*, Wozny M. J., Turner J. U., Preiss K., (Eds.). Elsevier Science Publishers B.V. (North-Holland), 1990, pp. 485–498.

[Fra80] FRANKLIN W. R.: A linear time exact hidden surface algorithm. *Comput. Graph. 14*, 3 (1980), 117–123.

[Fra87] FRANKLIN W. R.: Polygon properties calculated from the vertex neighborhoods. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.* (1987), pp. 110–118.

[Fra04] FRANKLIN W. R.: Analysis of mass properties of the union of millions of polygedra. In *Geometric Modeling and Computing: Seattle 2003*, Lucian M. L., Neamtu M., (Eds.). Nashboro Press, Brentwood TN, 2004, pp. 189–202.

[Fra05a] FRANKLIN W. R.: Mass properties of the union of millions of identical cubes. In *Geometric and Algorithmic Aspects of Computer Aided Design and Manufacturing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Janardan R., Dutta D., Smid M., (Eds.), vol. 67. American Mathematical Society, 2005, pp. 329–345.

[Fra05b] FRANKLIN W. R.: Nearpt3 — Nearest point query on 184M points in E^3 with a uniform grid. In *Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG'05)* (Windsor, Ontario, 10-12 August 2005), pp. 239–242.

[Fre07] FREE SOFTWARE FOUNDATION: GSL – GNU scientific library. http://www.gnu.org/software/gsl/, (accessed) 25 Apr 2007.

[GS83] GUIBAS L. J., STOLFI J.: Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. In *Proc. 15th Annu. ACM Sympos. Theory Comput.* (1983), pp. 221–234.

[LTH86] LAIDLAW D. H., TRUMBORE W. B., HUGHES J. F.: Constructive solid geometry for polyhedral objects. *Comput. Graph. 20*, 4 (1986), 161–170. Proc. SIGGRAPH '86.

[Mir96] MIRTICH B.: Fast and accurate computation of polyhedral mass properties. *J. Graphics Tools 1*, 2 (1996), 31–50.

[MP78] MULLER D. E., PREPARATA F. P.: Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci. 7* (1978), 217–236.

[MS85] MEHLHORN K., SIMON K.: Intersecting two polyhedra one of which is convex. In *Proc. Found. Comput. Theory* (1985), Budach L., (Ed.), vol. 199 of *Lecture Notes Comput. Sci.*, Springer-Verlag, pp. 534–542.

[NF91] NARAYANASWAMI C., FRANKLIN W. R.: Determination of mass properties of polygonal CSG objects in parallel. In *Proc. Symposium on Solid Modeling Foundations and CAD/CAM Applications* (June 1991), Turner J., (Ed.), ACM/SIGGRAPH, pp. 279–?

[Ras84] RASHIN A. A.: Buried surface area, conformational entropy, and protein stability. *Biopolymers 23*, 8 (1984), 1605–1620.

[SaHBZ07] SHENTU Z., AL HASAN M., BYSTROFF C., ZAKI M.: Context shapes: Efficient complementary shape matching for protein-protein docking. *Proteins: Structure, Function and Bioinformatics* (2007).

[Sug94] SUGIHARA K.: A robust and consistent algorithm for intersecting convex polyhedra. *Comput. Graph. Forum 13*, 3 (1994), 45–54. Proc. EUROGRAPHICS '94.

[TWA*01] TOMA L., WICKREMESINGHE R., ARGE L., CHASE J. S., VITTER J. S., HALPIN P. N., URBAN D.: Flow computation on massive grids. In *Proc. ACM Symposium on Advances in Geographic Information Systems* (2001). see also http://www.cs.duke.edu/geo*/terraflow/papers/bib.html.

[Vig] VIGNERON A.: Reporting intersections among thick objects.

[Wei07] WEISSTEIN E. W.: Gauss-Kronrod quadrature, from Mathworld–a Wolfram web resource. http://mathworld.wolfram.com/Gauss-KronrodQuadrature.html, (accessed) 25 Apr 2007.

[WKK99] WANG J., KOLLMAN P., KUNTZ I.: Flexible ligand docking: a multistep strategy approach. *Proteins 36* (July 1999), 1–19.

[ZE02] ZOMORODIAN A., EDELSBRUNNER H.: Fast software for box intersections. *International Journal of Computational Geometry and Applications 12*, 1-2 (2002), 143–172.