DIMACS Series in Discrete Mathematics
and Theoretical Computer Science

Mass Properties of the Union of Millions of Identical Cubes

W. Randolph Franklin

Abstract. We present an algorithm and implementation, UNION3, for computing mass properties of the union of many identical axis-aligned cubes. This implementation has processed 20,000,000 cubes on a dual processor Pentium Xeon workstation in about one elapsed hour. The ideas would extend to the union of general polyhedra.

UNION3 works by combining three techniques. The first is a set of local topological formulae for computing polyhedral mass properties. No global topology is used, such as complete face or edge information, or edge loops and face shells. For example, the volume of a multiple component, concave, rectilinear object is $\sum s_i x_i y_i z_i$, summed over the vertices (x_i, y_i, z_i), where each s_i is a function of the directions of the incident edges. The second technique is a 3-D grid of uniform cells, which is overlaid on the data. Each cell has a list of edges, faces, and cubes that overlap it. Only pairs of objects in the same cell are tested for intersection. Experimentally, the uniform grid is quite tolerant of varying densities in the input. The third technique, building on the second, is the covered-cell concept. When a cell is completely contained within a cube, its overlap list is cleared, and objects in it are not intersected against each other (unless they both also overlap the same other, noncovered, cell). These techniques combine to reduce the expected computation time to linear in the number of cubes, regardless of the number of intersections.

In the slowest step of UNION3, which is testing pairs of objects in each cell for intersection, no inter-cell communication is required. Therefore UNION3 parallelizes quite well.

1. Introduction

3-D geometric applications sometimes create an object as the union of a large set of polyhedra. For example, in CAD/CAM, the volume swept by a cutting tool may be approximated by one polyhedron for each segment of the tool path. On occasion we don't need the resulting polyhedron itself, but only certain mass properties. This presents several opportunities for optimization.

2000 *Mathematics Subject Classification.* Primary 65D32; Secondary 68U07, 65D18, 49Q15, 26B15, 28A75, 51M25.

Key words and phrases. boolean operation, union, polyhedron, volume, mass property, uniform grid, parallel geometry computation.

This research was supported by NSF grant CCR 03-06502.

First, the usual method for computing the volume of the union of many polyhedra proceeds by forming a $(\log N)$ level computation tree that combines successively larger intermediate polyhedra pair by pair. The execution time is probably $N(\log N)^2$ to $N^2(\log N)^2$ depending on the size of the intermediate polyhedra and the efficiency of the polyhedron intersections. There may be many intermediate vertices even when the output polyhedron has few.

Next, efficiently combining polyhedra is more difficult than combining polygons. Sweeping space with a plane, while keeping order among all the active faces, is more intricate than sweeping a plane with a line that keeps track of active edges.

Third, if we need only the volume, or similar mass property, of the computed polyhedron, then it is unnecessary completely to compute the union polyhedron first. Indeed, we need only its vertices with their neighborhood geometries.

2. Prior Art

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 combo of two *local* polyhedra in $O(N \log N)$ time. However since the faces of a union are often not simplices, our polyhedra 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 2-D, 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 the large values of N in this paper, the $(\log N)$ is significant. Of course, these 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; NF91b; FCK$^+$90) are elusive. Large geometric and cartographic data sets are now being processed, perhaps with external algorithms, (TWA$^+$01; Cen01).

Our big problem is to compute mass properties, including second order moments, center of gravity, volume, area, and edge length, of a very complex polyhedron 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 identical cubes, and to demonstrate it on as large an input example as possible.

We have produced a prototype implementation, UNION3, of the restricted case, which finds mass properties of the union of many identical cubes. This implementation has processed 20,000,000 cubes, containing half a billion subfacets, on a dual processor Pentium Xeon workstation in about one hour. Smaller examples take proportionally less time. E.g., processing 1000 cubes with edge length 0.1 takes 2 elapsed seconds on a 1600 MHz laptop.

UNION3 works by combining three techniques. The first is a set of local topological formulae for computing polyhedral mass properties. Let $\{v_i\}$ be some polyhedron $\mathbb{P}$'s set of vertices. Let M be any one of the mass properties listed earlier. Then there exists a function f_M such that $M = \sum_i f_M(v_i)$. (The center of mass is the ratio of two such formulae.) $f_M(v)$ depends on only v's location and local geometry and topology. That includes the directions in which any edges or faces are adjacent on v, and which adjacent sector is inside. No global topology is used, such as complete face or edge information, or edge loops and face shells. Provided that the polyhedron is valid, these formulae are valid regardless of global connectivity such as the number of components and their containment hierarchy.

The first advantage of such formulae is that determining vertices and local information of a boolean combination is much easier than determining global information, such as complete output edges and faces. The second is that our data structures are much more compact, so larger examples are feasible.

The second technique is a 3-D grid of uniform cells. Each cell records the overlaps of itself with any edge, face, or polyhedron. Intersection tests are performed only between objects overlapping the same cell. Therefore, we can quickly cull out pairs of groups of objects that cannot possibly intersect. This simple technique is effective because testing two line segments (or other primitives) for intersection is much faster than updating a complicated topological data structure. Therefore, testing, say, ten pairs to find one intersection is faster than performing a plane sweep operation while maintaining ordering information. This is also why the uniform grid also performs quite well in practice on even quite irregular actual data. For example, we describe intersecting all the edges in two anisotropic meshes, whose edges' lengths have a spread of 100:1.

The third technique, building on the second, is the covered-cell concept. It solves the problem that, for fixed-size input objects, the number of intersections of subfacets grows as $O(N^3)$. Thus, no initial cull can suffice. However, although the

total number of intersections may be $O(N^3)$, the number of *visible* intersections, that is, intersections that are not contained in some input object, grows only linearly, under broad conditions. Only these visible intersections contribute to the result. However, if the grid cell size is smaller than the object size, then as the set of objects gets denser, with ever higher probability a particular cell will fall completely inside some object. Thus, in that cell, no intersections will visible, and so no intersections need be determined in that cell. The number of tests that need to be performed, i.e., in cells not completely covered by objects, is linear, not cubic.

UNION3's simple flat data structures permit it to fork copies of itself to utilize multi-processor machines. This contrasts favorably to topological sweep methods, which are harder to parallelize. The parallelizability of the uniform grid is presented in more detail in (KF95; FK93; NF92a; NF92b; NF91a; FK90; Kan90; Nar91).

UNION3 processes all the polyhedra in one pass instead of repeatedly combining them pair by pair. The first step finds the candidate output vertices. These are the 3-face intersections, edge-face intersections, and input vertices. Next, the candidates are culled by deleting those inside any polyhedron. The volume is the sum of a function of each survivor. Input degeneracies are processed with Simulation Of Simplicity. Since UNION3 never explicitly determines the output polyhedron, messy non-manifold cases become irrelevant. No complicated topological structures are computed. The expected time is linear in the number of input, even when the number of intersections is superlinear. That the time is indeed linear is supported by our implementation's ability to process 20M small cubes.

3. What UNION3 is Not

(1) We do not propose that objects be modeled as the union of identical cubes. The point of this paper is to demonstrate the feasibility of the concept by means of a prototype implementation, using an easily implementable subset of all possible polyhedra.
(2) Although this prototype implementation is for identical cubes, the concepts are not restricted to this, but extend to general polyhedra.
(3) This is not a Monte Carlo or other statistical sampling technique. The output is exact to the arithmetic precision of the computations.
(4) UNION3 is also not a voxelization or octree method, where the universe is partitioned into voxels, and each object is represented as the union of a set of voxels, and everything is only as accurate as the voxel size. Although superficially similar, our uniform grid is quite different. Different grid sizes affect only the execution time and space, but have no effect on the result. Input cubes generally overlap grid cells only partially.
(5) The general concept is also not limited to unions. Processing general constructive solid geometry trees appears feasible. However, efficiently culling the impossible output vertices in the general case is still future research, which would build on the foundation established in (Til84). Special cases, such as two level sum-of-products formulae, appear quite doable, efficiently.

4. Volume Determination

The volume of a cube with vertices (x, y, z) may be expressed as $\mathbb{V} = \sum_i s_i x_i y_i z_i$, where $s_i = \pm 1$. For any particular vertex, $s = +1$ iff an odd number of the three faces adjacent to that vertex are on the high side of the cube. The volume of any rectilinear polyhedron, whose edges and faces are all aligned with the coordinate axes, is still $\mathbb{V} = \sum_i s_i x_i y_i z_i$, if the definition of s is generalized for every possible local vertex geometry. Our implementation uses this formula. The above formula easily extends to other mass properties, such as mass moments of any order. These formulae are related to Greene's theorem. (LR82a; LR82b) is an excellent summary of Greene's theorem and many related formulae for different representations.

Likewise, lower dimensional mass properties, such as face area and edge length may be computed as $\mathbb{A} = \sum s_i x_i y_i$ and $\mathbb{L} = \sum s_i x_i$ with the s_i in each case a function of the local geometry of the vertex. Throughout this paper "volume" includes any mass property.

All the above extends to general polyhedra, as described in (NF91a; Fra87), although the implementation is considerably harder. Indeed, there are varying classes of formula, depending on how much topology is available. (Mir96) also describes efficient polyhedron formulae.

Suppose that we have only the set of incidences of output vertices, edges lines, and face planes, together with which side is inside. For instance, for a cube, each vertex would induce six such incidences. Then if P is the vertex position, $\hat{T}$ is a unit tangent vector along the edge incident on it, $\hat{N}$ is a unit vector normal to $\hat{T}$ in the plane of the face, and $\hat{B}$ is a unit binormal vector, normal to both $\hat{T}$ and $\hat{N}$ pointing into the polyhedron, then the volume is $\mathbb{V} = -\frac{1}{6} \sum P.\hat{T}\, P.\hat{N}\, P.\hat{B}$. Similar formulae obtain for other mass properties. Figure 1 illustrates this for a cube. Each visible *(vertex, edge, face)* tuple is check marked, except for one that is starred. For that one, the vectors P, $\hat{T}$, $\hat{N}$, and $\hat{B}$ are shown.

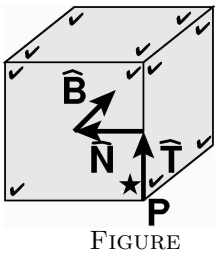

FIGURE 1. $P, \hat{T} \hat{N}, \hat{B}$

There are three classes of output vertices resulting from uniting a set of polyhedra: (1) a vertex of one of the input polyhedra, (2) an intersection of an edge with a face, and (3) an intersection of 3 faces.

An output vertex must not be contained in any polyhedron. Therefore, the process goes as follows. (1) Generate a candidate output vertex of one of the above types. (2) Test to see if it's outside all the polyhedra. (3) If so, then compute its sign based on its neighborhood, and add another term into the running total of the volume.

We define "intersection" to exclude components of the same polyhedron. That is why edge-face and 3-face intersections are different cases, even though every edge might be considered as the intersection of two adjacent faces.

The naive algorithm would test all triples of faces for intersection, requiring $T = \theta(N^3)$ time. However we use a uniform grid data structure, introduced by (Fra78), with applications, i.a., in (Nar91), (Kan90), (AFKN89), (HH90), (FK90), (KF95), and (FSS$^+$94), to reduce that, as follows.

(1) Superimpose a 3-D grid with $G \times G \times G$ cells on the universe. A reasonable cell size is half the average polyhedron size; however a wide range of G is acceptable
(2) For each cell, initialize an empty list, to be populated later, of items overlapping it.
(3) Iterate through the polyhedra. For each polyhedron, $\mathbb{P}$:
    (a) Determine which grid cells $\mathbb{P}$ *completely encloses*. Mark those cells as *covered*.
    For a cube, a range test is done in each dimension separately, and a cartesian product is taken of the results. For instance, let $G = 10$, and let the x-range of a test cube be $[.25, .57]$. Then, in the x-dimension it will cover cells 3 and 4, counting from 0.
    (b) For steps 3c and 3d below, whenever an item would be inserted into a cell, do not insert it if that cell is covered.
    (c) Determine which cells $\mathbb{P}$ *overlaps*. Add the polyhedron to those cells' lists. Continuing the example from item 3a above, that cube overlaps cells 2, 3, 4, and 5, in the x-dimension. Since it covers cells 3 and 4, the cube is added to the overlap lists of only cells 2 and 5.
    (d) Ditto for each face and edge.
(4) Iterate through the cells. Process the contents of each cell's overlap list, $\mathbb{L}$, as follows.
    (a) Test all triples of faces from $\mathbb{L}$, which are from three different polyhedra, for intersection. Three faces intersect if the intersection point of their three planes is contained in each face polygon. For each triple that does intersect, say at point $\mathbb{X}$, test if $\mathbb{X}$ is outside all polyhedra. If it is, then look up its sign in a table, based on the directions of its three faces, and add a term to the running total for the volume.
    (b) Test all pairs of edges and faces from $\mathbb{L}$, from two different polyhedra, for intersection, and process the intersections as before. A face and edge intersect if (a_0, a_1, a_2), the intersection point of the face plane and the edge line, is inside both the face and edge. The contribution to the volume of the union is computed as follows.

$$
\begin{aligned}
f &\triangleq \text{index of this face in its cube, from 0 to 5} \\
e &\triangleq \text{index of this edge in its cube, from 0 to 11} \\
\Delta v &= (2(f\&1) - 1)(2(e\&1) - 1)(2((e \gg 1(-1))))a_0 a_1 a_2
\end{aligned}
\tag{1}
$$

where & means *bitwise and* and $\gg$ means to shift the left argument right by the number of bits specified by the right argument. The

other operations are arithmetic. This assumes that the faces of each cube are ordered: lo X, hi X, lo Y, hi Y, lo Z, hi Z, and that the edges likewise have a fixed precise order.
The contributions of this intersection to the surface area and edge length of the union are two other similar formulae.

(5) Iterate through the vertices, testing whether each is outside all the polyhedra. For each outside vertex, then determine its s, and add $s \cdot x \cdot y \cdot z$ to the volume running total.

Determining whether point $\mathbb{X}$ is contained in any polyhedron proceeds as follows. (The precise identity of the containing polygon is unnecessary.)

(1) Compute which cell, $\mathbb{C}$, contains $\mathbb{X}$.
(2) If $\mathbb{C}$ is covered, then $\mathbb{X}$ is contained in some polyhedron.
(3) Otherwise, test whether any polyhedron in $\mathbb{C}$'s list contains $\mathbb{X}$.

Our implementation handles only identical cubes, although the theory extends to general polyhedra. Allowing only cubes removed numerical roundoff problems and vastly simplified the implementation, while still demonstrating the algorithm.

This implementation handles degeneracies via Simulation of Simplicity (SoS), (EM88) for volume computation. For instance, whenever a point is tested against a face, the three polyhedra whose faces intersected to create that point are brought along. In cases of a coordinate equality, the polyhedron numbers are compared to break the tie. However, then the computed area and length differ from the result of a regularized set union. Say that two polyhedra share a face. With SoS, either both faces will be counted, or neither will, depending on which polyhedron has the lower number.

The importance of the covering cell concept requires emphasis. The bad case for any intersection algorithm is when the objects are large, as shown in Figure 2, so that there are a cubic number of 3-face intersections. However we don't need all the intersections, but only those that are outside all polyhedra, i.e., *visible*. Figure 3 shades the covered cells, and X's the covered intersections. The number of visible intersections grows much more slowly than the total number of intersections. Indeed, under some reasonable assumptions analyzed in (Fra04), it grows only linearly. Therefore, the covering cell concept can reduce the intersection time from cubic to linear. It also reduces the query time to test point containment from linear time per point down to constant time per point. Indeed, the average number of polyhedra contained in each cell is constant, with reasonable cell sizes.

A major objection to the uniform grid is that it will fail when presented with real world data, since the real world is not uniform. This point is obvious, but *wrong*, as described in (AFKN89). Although it is counterintuitive, it can be shown that, in these applications, the grid structure tolerates nonuniform input as well as a hierarchical structure such as the quadtree, and the implementation is simpler. Consider, for example, the simpler case of finding 2D edge intersections. Let the number of edges in the i-th grid cell be k_i, and the total number of edges be N,

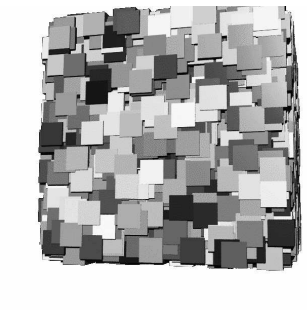

FIGURE 2. Seriously Overlapping Objects

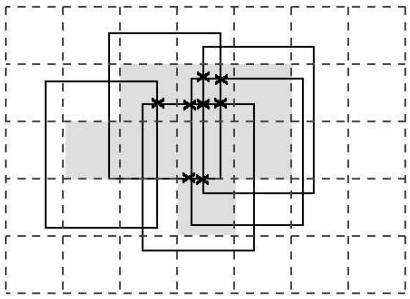

FIGURE 3. Covered Cells

and the grid be $G \times G$ cells. A reasonable value is $G = 1/\sqrt{N}$. $\sum_i k_i = cN$ where c is the average number of cells that an edge is incident on. The average number of edges per cell is $\overline{k} = cN/G^2 = c$. If the edges are uniformly, independently, and identically distributed, then the number of edges per cell follows a Poisson distribution with $\lambda = \overline{k} = c$, and so $\overline{k^2} = \overline{k}^2 = c^2$. The total number of edge pairs tested by pairwise comparing all the edges in each cell to find the intersections is

$$\begin{aligned} T &= \sum_i k_i(k_1 - 1)/2 \\ &= \frac{1}{2}G^2\left(\overline{k^2} - \overline{k}\right) \\ &= c_2 N \end{aligned} \tag{2}$$

An analysis of the asymptotic execution time for the full algorithm is given in (Fra04).

The bad case for both a grid and a quadtree would be many close, correlated, edges, which do not intersect. Imagine $N/2$ edges uniformly distributed, and the other $N/2$ edges uniformly distributed in a square of size $2^{-N} \times 2^{-N}$ in one corner. One grid size is inadequate for both groups of edges. However, a quadtree would require so many levels that it would also be too slow. A topological sweep could handle this case, but at the cost of a complicated data structure that is difficult to parallelize and takes $O(N \log N)$ time. However, first, $\log N$ is nontrivial for current N. Second, topological sweeps are much harder in 3D. Finally, and most important, the sweep finds all intersection vertices, including all $\Omega(N^3)$ hidden ones.

The parallel implementation of the algorithm goes as follows.

(1) Do steps 1 to 3 in one process (the main process); they are only a small fraction of the total time.
(2) Let M be the desired number of parallel processes.

(3) Partition the G^3 cells into M groups, one per process, preferably so that the groups are not too uneven.
(4) Repeat the following M times.
    (a) The main process creates a fifo pipe.
    (b) It forks a subprocess.
    (c) It gives each subprocess the data on all the input polyhedra, and the overlap lists of the G^3/M cells in its group.
    (d) Each subprocess totals up the contributions to the total volume, area, and length caused by the cells in its group. This is the compute-bound step.
    (e) Each subprocess returns these three numbers to the main process via the pipe.
(5) The main process waits until all M processes have completed and reads the volume, area, and length subtotals from each pipe.
(6) Finally, it sums the M volume subtotals, etc, and reports them.

5. Why Not Use Global Topology?

The intersection process directly produces only local topological information about the output polyhedron. It may look easy to link up this local topological info to find the output polyhedron's boundary. In practice, it is not easy, even in 2-D, and is much worse in 3-D. 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 infinite line, group those ends according to whether they are on the same infinite line.
(3) For each infinite line: 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 along the infinite line, and group them into adjacent pairs, each forming one edge. Otherwise, if there are an odd number of ends, then report an inconsistency.
(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 infinite planes that they are incident on. This is well defined since each vertex knows the directions of its incident edges.

(6) For each infinite plane found in the previous step, identify any edges that are incident on it.
(7) In each infinite plane, connect up the vertices and edges on it into faces, while checking for inconsistencies.
(8) If there is more than one face per plane, 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 (2-D) polygons requires perhaps 1000 lines of code, of quality such that testing point inclusion in a polygon is only 10 lines.

When combining only two polyhedra, there is an alternative to computing an unorganized set of vertices and then determining the global topology. That is, to trace out the boundary of the resulting polyhedron by following along the boundaries of the two input polyhedra. As always, nonmanifold cases cause complications. In addition, it is completely unclear how to extend this to trace out the union of many polyhedra in one step, without forming a tree of unions of smaller sets of input polyhedra.

6. Implementation Tests

The HW is a dual 2.4 GHz Xeon with 4GiB of real memory. The SW is SuSE 8.2 linux and the Intel C++ compiler. The program is about 1000 lines of code, excluding debugging lines, comments, and blank lines. The input cubes are generated with a combination of three Tausworth random number generators, which is much better than the widely used class of linear congruential generators. One problem with even the best linear congruential generators is that, if we let the generated numbers be x_i, then the 3-D points (x_i, x_{i+1}, x_{i+2}) fall on a relatively small number of parallel planes.

Figure 4 shows some sample runs, varying the number of cubes, edge length, grid resolution, and number of processors. The grid resolution and number of processors affect the time for a given input. For each run, the CPU time is reported, conservatively, as the sum over all the processes. However, since 4 threads can execute in parallel, the elapsed time is usually much less.

The added line has a slope of one, which would be $T = \Theta(N)$. The observed time performance appears to be slightly worse than linear. This is principally caused by memory limitations, which force a suboptimally small grid resolution to be used for large datasets.

Table 1 shows some statistics from the largest case. Lengths are scaled so that the universe is $1 \times 1 \times 1$. The actual edge length is 32768/(approx inverse edge length), rounded to a multiple of 10 (to make the numbers easier to read when debugging).

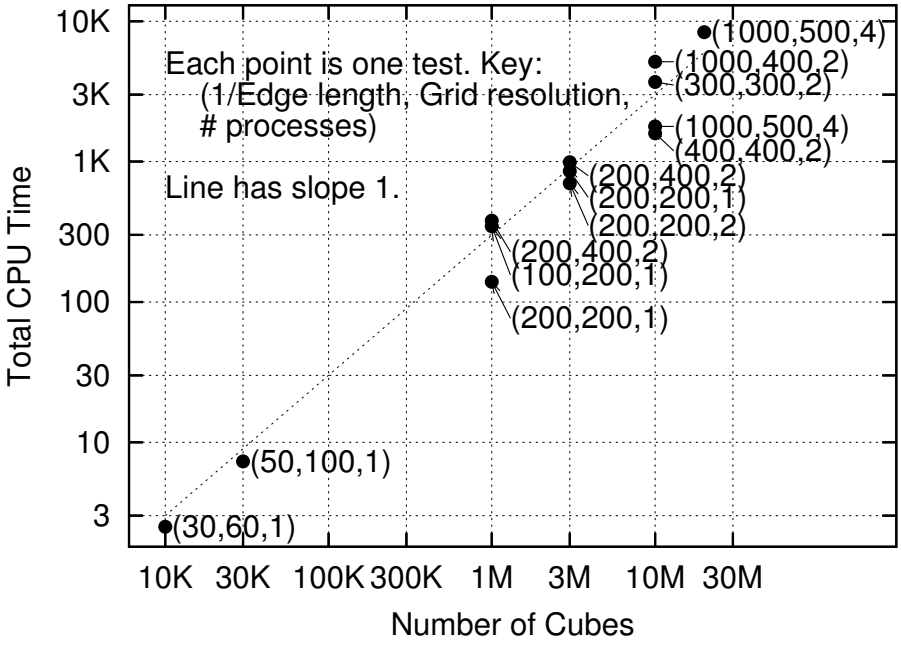

FIGURE 4. Execution Time vs Number of Cubes

Number of cubes	20,000,000
Edge length	0.00916
Number of processes	4
Number of grid cells	125,000,000
Number of cell – cube overlaps	61,890,769
Number of cell – face overlaps	254,823,296
Number of cell – edge overlaps	349,733,052
Sum of the cubes' volumes	0.015
Volume of the union	0.009
Sum of all the cubes' surface areas	100.583
Surface area of the union	99.053
Sum of all the cubes' edges	219,727
Edge length of the union	219,708
Number of input vertices	160,000,000
Number of those vertices that were outside all cubes	157,561,445
Number of 3-face intersections	37,170
Number face–edge intersections	7,259,255

TABLE 1. Statistics of the 20M Cube Test

The hard limit affecting the implementation is the available real memory. (If the total virtual memory of all the processes greatly exceeds the real memory then the amount of paging can hurt performance by more than an order of magnitude.) This limit also inhibits the use of a finer grid, which would reduce the time in some cases. In this largest case, that limitation meant that there were no covered cells. One solution to the memory problem is to subdivide the problem, then execute the parts sequentially, not in parallel.

How do we know that the above numbers are correct? Although errors are always possible, several indicators give us confidence. First, summing the addends mostly cancels out so that the final volume is in the range $[0, 1]$. Any erroneous addend is likely to produce a total that is out of range. Second, we constructed nasty test cases involving many cubes exactly overlapping or aligning along faces. (This also tested the SoS code.) The computed volume was always correct.

Third, for random input, we can estimate the volume as follows. For N independent and uniformly distributed cubes, each of volume v, the expected union volume is $V_{est} = 1 - (1 - v)^N \approx 1 - e^{-vN}$. The agreement between the predicted and computed volumes is excellent.

7. Non-Uniform Data and the Uniform Grid

It's not a priori obvious that the uniform grid can handle non-uniform data, at least without adapting into a quadtree. Formally, since the execution time contains terms that both increase and decrease with G, the best G is a broad optimum, and therefore if some cells contain, say, an order of magnitude more objects than the average, there's no problem. However, the best demonstration is by using real data, from diverse application domains.

Figure 5 shows the intersections of the edges of two anisotropic meshes from Mark Shephard and Xiangrong Li. Figure 6 is a detail from it. Together these two meshes have 54057 edges, whose lengths are quite variable, ranging in length from about 0.002 to 0.23, a factor of 100. These figures show the edges' extremely uneven density and orientation, which is a bad case for the uniform grid. Nevertheless, we can easily process this data. Before these experimental results were available, we considered using a multilevel grid, but that was unnecessary.

With a 90×90 uniform grid, we found all 64273 intersections of these edges in under 2 CPU seconds on a 1600 MHz laptop. That time includes reading the edge file and writing the intersections. No drastic optimization steps were necessary. Serious profiling and optimizing would probably reduce that time considerably. The average edge crossed 3 grid cells. In each of the 8100 cells, we tested all pairs of edges for intersection; 2,650,725 tests in all. Those tests are quite fast, so it's better not to filter the edges more thoroughly. There resulted 127,606 intersections, or 63,273 after duplicates were removed.

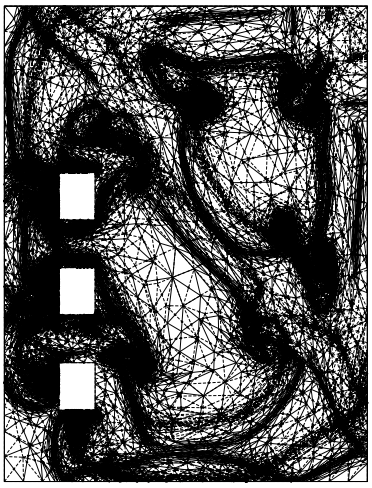

FIGURE 5. Two Anisotropic Meshes Intersected

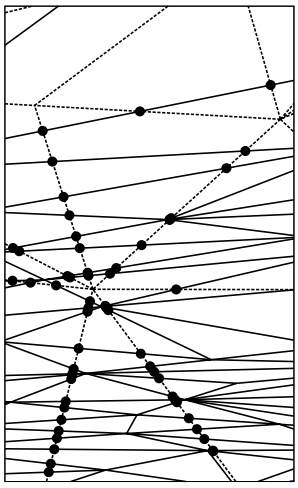

FIGURE 6. Detail from the Previous Figure

Figure 7 shows a cartographic application, with data from Denis White. The input is two planar graphs or maps. The first map is coterminous US counties, with 55068 vertices, 46116 edges, and 2985 polygons. The second map shows hydrography polygons, with 76215 vertices, 69835 edges, and 2075 polygons. The density variation between urban and rural regions is clear. Intersecting the two maps and reporting the areas of all the nonempty polygon intersections took 1.58 CPU seconds on the laptop, plus I/O time.

Finally, Figure 8 shows an integrated circuit designed by Jim Guilford, again with a wide variation in line segment density. With a uniform grid, we easily found all the intersections. Since these three examples are 2-D, we are searching for large 3-D examples to test.

8. Conclusion

This prototype implementation demonstrates that simple data structures, with no global topology, are an excellent method for processing large geometric datasets in 3D. The resulting implementations are simpler, faster, can process very large datasets, and can be parallelized.

Our method performs particularly well for very dense sets of objects, where most cells are covered, which is the worst case for sweep-line methods. It also is not merely a toy demonstration, but scales up to very large datasets.

FIGURE 7. Counties Overlaid on Hydrography Polygons

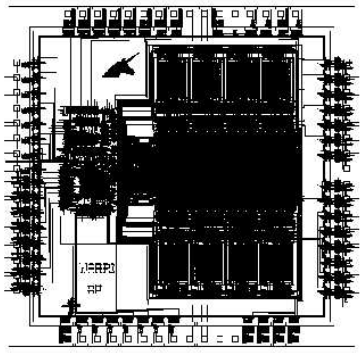

FIGURE 8. VLSI Design

In addition, this technique would also support other operations, not yet implemented, such as online computation of mass properties, as polyhedra 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 polyhedron. Computing properties of more complicated boolean operations is feasible. Producing the explicit output polyhedron is also possible, though considerably more complicated. This could even be extended to collision detection of moving objects, each composed of the union of many simple polyhedra.

References

[AFKN89] Varol Akman, Wm Randolph Franklin, Mohan Kankanhalli, and Chandrasekhar Narayanaswami, *Geometric computing and the uniform grid data technique*, Computer Aided Design **21** (1989), no. 7, 410–420.

[AST97] Boris Aronov, Micha Sharir, and Boaz Tagansky, *The union of convex polyhedra in three dimensions*, SIAM J. Comput. **26** (1997), 1670–1688.

[BN83] H. Bieri and W. Nef, *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] E. Brisson, *Representing geometric structures in d dimensions: Topology and order*, Proc. 5th Annu. ACM Sympos. Comput. Geom., 1989, pp. 218–227.

[Cam91] S. Cameron, *Efficient bounds in constructive solid geometry*, IEEE Comput. Graph. Appl. **11** (1991), no. 3, 68–74.

[CD87] Bernard Chazelle and D. P. Dobkin, *Intersection of convex objects in two and three dimensions*, J. ACM **34** (1987), no. 1, 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] Bernard Chazelle, *An optimal algorithm for intersecting three-dimensional convex polyhedra*, Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci., 1989, pp. 586–591.

[Des44] R. Descartes, *Principia philosophiae*, Ludovicus Elzevirius, Amsterdam, 1644.

[DK83] D. P. Dobkin and D. G. Kirkpatrick, *Fast detection of polyhedral intersection*, Theoret. Comput. Sci. **27** (1983), no. 3, 241–253.

[DMY93] Katrin Dobrindt, Kurt Mehlhorn, and Mariette Yvinec, *A complete and efficient algorithm for the intersection of a general and a convex polyhedron*, Proc. 3rd Workshop Algorithms Data Struct., Lecture Notes Comput. Sci., vol. 709, 1993, pp. 314–324.

[EM88] H. Edelsbrunner and E. P. Mücke, *Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms*, Proc. 4th Annu. ACM Sympos. Comput. Geom., 1988, pp. 118–133.

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

[Eri03] J. Erickson, *Local polyhedra and geometric graphs*, 2003.

[FCK+90] Wm Randolph Franklin, Narayanaswami Chandrasekhar, Mohan Kankanhalli, Varol Akman, and Peter YF Wu, *Efficient geometric operations for CAD*, Geometric Modeling for Product Engineering (Michael J. Wozny, Joshua U. Turner, and K. Preiss, eds.), Elsevier Science Publishers B.V. (North-Holland), 1990, pp. 485–498.

[FK90] Wm Randolph Franklin and Mohan Kankanhalli, *Parallel object-space hidden surface removal*, Proceedings of SIGGRAPH'90, vol. 24, August 1990, pp. 87–94.

[FK93] Wm Randolph Franklin and Mohan S. Kankanhalli, *Volumes from overlaying 3-D triangulations in parallel*, Advances in Spatial Databases: Third Intl. Symp., SSD'93 (D. Abel and B.C. Ooi, eds.), Lecture Notes in Computer Science, vol. 692, Springer-Verlag, June 1993, pp. 477–489.

[Fra78] Wm Randolph Franklin, *Combinatorics of hidden surface algorithms*, Technical Report, Center Res. Comput., Harvard Univ., Cambridge, MA, June 1978.

[Fra87] ———, *Polygon properties calculated from the vertex neighborhoods*, Proc. 3rd Annu. ACM Sympos. Comput. Geom., 1987, pp. 110–118.

[Fra04] W. Randolph Franklin, *Analysis of mass properties of the union of millions of polygedra*, 2004, (to appear).

[FSS+94] Wm Randolph Franklin, Venkateshkumar Sivaswami, David Sun, Mohan Kankanhalli, and Chandrasekhar Narayanaswami, *Calculating the area of overlaid polygons without constructing the overlay*, Cartography and Geographic Information Systems (1994), 81–89.

[GS83] Leonidas J. Guibas and J. Stolfi, *Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*, Proc. 15th Annu. ACM Sympos. Theory Comput., 1983, pp. 221–234.

[HH90] Sara Hopkins and Richard G. Healey, *A parallel implementation of Franklin's uniform grid technique for line intersection detection on a*

large transputer array, 4th International Symposium on Spatial Data Handling (Zürich) (Kurt Brassel and H. Kishimoto, eds.), 23-27 July 1990, pp. 95–104.

[Kan90] Mohan Kankanhalli, *Techniques for parallel geometric computations*, Ph.D. thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute, October 1990.

[KF95] Mohan Kankanhalli and Wm Randolph Franklin, *Area and perimeter computation of the union of a set of iso-rectangles in parallel*, J. Parallel Distrib. Comput. **27** (1995), 107–117.

[LR82a] Y. T. Lee and A. A. G. Requicha, *Algorithms for computing the volume and other integral properties of solids. I. Known methods and open issues*, Commun. ACM **25** (1982), 635–641.

[LR82b] ———, *Algorithms for computing the volume and other integral properties of solids. II. A family of algorithms based on representation conversion and cellular approximation*, Commun. ACM **25** (1982), 642–650.

[LTH86] D. H. Laidlaw, W. B. Trumbore, and J. F. Hughes, *Constructive solid geometry for polyhedral objects*, Comput. Graph. **20** (1986), no. 4, 161–170, Proc. SIGGRAPH '86.

[Mir96] B. Mirtich, *Fast and accurate computation of polyhedral mass properties*, J. Graphics Tools **1** (1996), no. 2, 31–50.

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

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

[Nar91] Chandrasekhar Narayanaswami, *Parallel processing for geometric applications*, Ph.D. thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute, 1991, UMI no. 92-02201.

[NF91a] C. Narayanaswami and Wm Randolph Franklin, *Determination of mass properties of polygonal CSG objects in parallel*, Internat. J. Comput. Geom. Appl. **1** (1991), no. 4, 381–403.

[NF91b] Chandrasekhar Narayanaswami and Wm. Randolph Franklin, *Determination of mass properties of polygonal CSG objects in parallel*, Proc. Symposium on Solid Modeling Foundations and CAD/CAM Applications (Joshua Turner, ed.), ACM/SIGGRAPH, June 1991, pp. 279–288.

[NF92a] Chandrasekhar Narayanaswami and Wm Randolph Franklin, *Boolean combinations of polygons in parallel*, Proceedings of the 1992 International Conference on Parallel Processing, August 1992.

[NF92b] ———, *Edge intersection on the Hypercube computer*, Information Processing Letters **41** (1992), no. 5, 257–262.

[Sug94] Kokichi Sugihara, *A robust and consistent algorithm for intersecting convex polyhedra*, Comput. Graph. Forum **13** (1994), no. 3, 45–54, Proc. EUROGRAPHICS '94.

[Til84] R. B. Tilove, *A null-object detection algorithm for constructive solid geometry*, Commun. ACM **27** (1984), 684–694.

[TWA+01] Laura Toma, Rajiv Wickremesinghe, Lars Arge, Jeffrey S. Chase, Jeffrey Scott Vitter, Patrick N. Halpin, and Dean Urban, *Flow computation on massive grids*, Proc. ACM Symposium on Advances in Geographic Information Systems, 2001, see also `http://www.cs.duke.edu/geo*/terraflow/papers/bib.html`.

[Vig] Antoine Vigneron, *Reporting intersections among thick objects*.

[ZE02] Afra Zomorodian and Herbert Edelsbrunner, *Fast software for box intersections*, International Journal of Computational Geometry and Applications **12** (2002), no. 1-2, 143–172.

ECSE Dept, 6003 JEC,, Rensselaer Polytechnic Institute, 110 Eighth St, Troy NY 12180, USA

E-mail address: `mail@wrfranklin.org`

URL: `http://wrfranklin.org/`