Analysis of Mass Properties of the Union of Millions of Polyhedra

W. Randolph Franklin

Abstract. This paper analyzes the performance of UNION3, a fast algorithm for computing the volume, area, and other mass properties, of the union of many polyhedra. These values are computed exactly, up to floating roundoff, without statistical sampling. If the input objects are independently, identically, and uniformly distributed, then the expected output time is linear in the size of the input.

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.

The principal data structure is a grid of uniform cells in E^3. Each cell records incidences (overlaps) of itself with any edge, face, or polyhedron. Intersection tests are performed only between objects incident on the same cell. Also, if a cell is completely contained in some polyhedron, then no intersection tests are performed in it, since none of those intersections would be visible.

UNION3 is well suited for parallel machines. A prototype implementation for identical random cubes has processed 20,000,000 polyhedra, containing half a billion subfacets, on a dual processor Pentium Xeon workstation in about one elapsed hour.

§1. Introduction

The union of several input objects is sometimes required only for its volume; the output subfacets (vertices, edges, and faces) are otherwise unnecessary. For example, in CAD/CAM, the polyhedron swept out by a

xxx and xxx (eds.), pp. 1–4.

ISBN 0-9728482-x-x

cutting tool may be approximated by computing one polyhedron for the region removed by each segment of the tool path, and then finding the union of these polyhedra. The amount of material removed is the volume of this union polyhedron. There may be many input polyhedra to be united. There are several opportunities for optimizing the composition of the volume and union operators.

First, the usual method for computing the volume of the union of N polyhedra proceeds by forming a computation tree with depth $\log(N)$, which combines successively larger intermediate polyhedra pair by pair. The execution time may range up to $\Theta(N^2(\log N)^2)$ depending on the size of the intermediate polyhedra and the efficiency of the polyhedron intersections. The intermediate polyhedra may have many vertices even when the output polyhedron has few.

Next, solving this in E^3 is much harder than in E^2. 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, it is unnecessary completely to compute the union polyhedron when only its volume, or similar mass property, is required. Indeed, its vertices with their neighborhood geometries suffice.

This paper analyzes UNION3, an algorithm with several optimizations addressing the above issues. Then it mentions an implementation for the restricted input case of identical cubes, which has processed test cases as large as $20,000,000$ cubes. More algorithm and implementation details are presented in [13].

There appear to be few prior results for object combination in E^3, except for the following special cases: efficiently intersecting convex polyhedra [6, 5, 18, 20], intersecting a convex polyhedron with a general one [9, 16], fast detection of polyhedral intersection [8], and uniting convex polyhedra [1]. Constructive Solid Geometry (CSG) is well documented, [3, 11, 15], though its efficient methods tend to use bounding boxes.

The prior art on mass properties of boolean combinations of polyhedra is similarly sparse, [2, 17]. Some of these local topological formulae could well date to the discovery of analytic geometry, [7], but actual citations earlier than [12, 14, 19] are elusive. Large geometric and cartographic data sets are now being processed, perhaps with external algorithms, [4, 21].

§2. Strategy

UNION3 works by combining three techniques.

1. The first is a set of local topological formulae for computing polyhedral mass properties. Let $\{v_i\}$ be some polyhedron $\mathcal{P}$'s set of vertices. Let M be any one of the mass properties listed earlier.

Then there exists a function f_M such that $M = \sum_i f_M(v_i)$. $f_M(v)$ 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 possible.

2. The second technique is a grid of uniform cells in E^3. Each cell records incidences (overlaps) of itself with any edge, face, or polyhedron. Intersection tests are performed only between objects incident on the same cell. Therefore, UNION3 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, it can easily intersect all the edges in two anisotropic meshes, whose edges' lengths have a spread of 100:1.
3. 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. If the grid cell size is smaller than the object size, then as the set of objects gets denser, with ever higher probability a given 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 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, such as a vertex in the plane of a nonadjacent face, are processed with Simulation Of Simplicity (SoS), [10]. SoS breaks ties in equality tests on input coordinates by pretending to add an infinitesimal of some order to each input number, and then determining its effect on equality tests with that number. For instance, when a point is tested against a face, if the point is on the face, the polyhedron numbers of the face and the point are compared to break the tie.

Since UNION3 never explicitly determines the output polyhedron, messy non-manifold cases become irrelevant. No complicated topological structures are computed. UNION3's simple flat data structures permit it to fork copies of itself to utilize multi-processor machines. The expected time is linear in the input size, even when the number of intersections is superlinear.

UNION3 is not a Monte Carlo or other statistical sampling technique. The output is exact to the arithmetic precision of the computations. 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, the 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.

§3. Volume Determination

The volume of a cube with vertices (x, y, z) may be expressed as $\mathcal{V} = \sum_i s_i x_i y_i z_i$, where $s_i = \pm 1$. For any given 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 isothetic polyhedron, whose edges and faces are all aligned with the coordinate axes, is still $\mathcal{V} = \sum_i s_i x_i y_i z_i$, if the definition of s is generalized for every possible local vertex geometry. Our implementation uses this formula. The above formula easily generalizes to other mass properties, such as moments of inertia of any order. Some properties, such as the center of mass, are the ratio of two such sums.

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

All the above extends to general polyhedra, as described in [12, 19], although the implementation is considerably harder. Indeed, there are varying classes of formula, depending on how much topology is available. E.g., consider 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, $\hat{E}$ is a unit vector along the edge incident on it, $\hat{F}$ is a unit vector normal to $\hat{E}$ in the plane of the face, and $\hat{B}$ is a unit vector normal to both $\hat{E}$ and $\hat{F}$ pointing into the polyhedron, then the volume is $\mathcal{V} = -\frac{1}{6}\sum (P \cdot \hat{E})\,(P \cdot \hat{F})\,(P \cdot \hat{B})$, where $\cdot$ is the scalar dot product. Similar formulae obtain for other mass properties. [17] also describes efficient polyhedron formulae.

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.

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, a uniform grid data structure reduces that, as follows. Figure 1 illustrates this, in E^2 for five polygons.

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. Figure 1 uses a 5×5 grid.
2. For each cell, maintain a list of items, to be determined later, incident on it.
3. For each polyhedron, determine which grid cells it completely encloses. Mark those cells as *covered*. In Figure 1, cell 22 is one of the seven covered cells.
4. For the next several steps, whenever an item would be inserted into a cell, do not insert it if that cell is covered.
5. For each polyhedron, determine which cells it overlaps with. Add the polyhedron to those cells' lists. In Figure 1, each square partly or completely overlaps 9 cells.

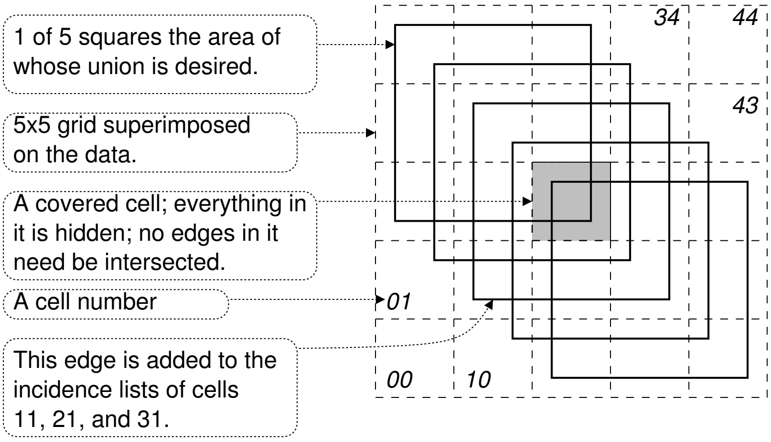
1 of 5 squares the area of whose union is desired.
5x5 grid superimposed on the data.
A covered cell; everything in it is hidden; no edges in it need be intersected.
A cell number
This edge is added to the incidence lists of cells 11, 21, and 31.
34
44
43
01
00
10

Fig. 1. Example, in E^2, of Using Uniform Grid for Union

6. Ditto for each face and edge. In Figure 1, each edge partly or completely overlaps 3 cells.
7. Iterate through the cells. For each cell:
   (a) Test all triples of faces, 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 $\mathcal{P}$, test if $\mathcal{P}$ 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.

   In Figure 1, of the 20 pairs of intersecting edges, 10 are not in a covered cell, and so are computed. Of them, 8 are outside all the squares and so contribute to the area of the union.

   (b) Test all pairs of edges and faces, from two different polyhedra, for intersection, and process the intersections as before. A face and edge intersect if the intersection point of the face plane and the edge line is inside both the face and edge. The intersection's sign is a complicated function of the directions of the two faces adjacent to the edge, and of the intersection face.
8. Iterate through the vertices, testing whether each is outside all the polyhedra. For each outside vertex, then determine its s, and add $sxyz$ to the volume running total. (Computing all these correct values of s was quite tricky.) In Figure 1, of the 20 vertices among the 5 squares, only 12 are outside all the other squares.

Determining whether point $\mathcal{P}$ is contained in any polyhedron proceeds as follows.

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

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

The importance of the covering cell concept requires emphasis. The bad case for any intersection algorithm is when the objects are large, so that there are a cubic number of 3-face intersections. However only the small fraction of the intersections that are outside all the polyhedra are necessary. Call them *visible*. The number of visible intersections grows much more slowly than the total number of intersections. Indeed, under some reasonable assumptions, 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*. Although it is counterintuitive, it has been 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. Examples include the following.

1. Intersecting the edges of two anisotropic meshes, with edge lengths varying over a range of 100:1
2. Overlaying two planar graphs, one of coterminous US counties, and the other of hydrography polygons. Although the urban polygons are much smaller than the rural ones, that was not a problem.
3. Intersecting all the edges in an integrated circuit design, which has both dense and sparse regions.

The bad case for both a grid and a quadtree would be many close edges that do not intersect. A topological sweep handles that, but requires 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 E^3. Finally, and most important, the sweep finds all intersection vertices, including all $O(N^3)$ hidden ones, which is impractically slow.

§4. Time Analysis

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

$N \triangleq$ number of input vertices

$L \triangleq$ length of each edge, 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 edge intersections in E^2, ignoring the possibility of covered cells.

4.1. Edge Intersections in E^2, Ignoring Covered Cells

If the edge segments are rectilinear, and independently and uniformly distributed, and $L \ll 1$ (to minimize effects near the border of the universe), then the expected number of cells incident on a given edge is $(LG + 1)$. For squares in E^2, N is also the number of edges. Therefore the average number of edge segments incident on any given cell is $\frac{N}{G^2}(LG+1)$. Since the edges are independent, the number of edges 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 edge must be tested against every other edge 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 edges inserted into all the cells plus the number of intersection tests (ignoring insignificant setup times proportional to the number of edges plus number of cells), or $T = \Theta\left(N(LG+1) + \frac{N^2}{G^2}(LG+1)^2\right)$. This is minimized when $G = \Theta(1/L)$, giving $T = \Theta\left(N + N^2L^2\right)$. Setting $S_i \triangleq$ size of the input $= \Theta(N)$, and $S_o \triangleq$ size of the output (the expected number of intersections) $= \Theta\left(N^2L^2\right)$, this gives

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

which is optimal for finding all intersections.

4.2. Edge 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 edges in the covered cells reduces the total number of edge 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 number of faces (here, squares) is $N/4$. The expected number of cells incident on a given face is $k = (LG + 1)^2$.

The expected number of cells covered by a given face is $\max(LG-1, 0)^2$. Therefore the probability that a given face incident on a cell covers that cell is $q \triangleq \frac{\max(LG-1,0)^2}{(LG+1)^2}$. Setting $G = c/L$, 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 faces incident on a given cell is $\lambda \triangleq kN/G^2 = \frac{9}{4}NL^2$. The probability of a given cell not being covered by any of the i faces incident on it, for $i \geq 0$ is $(1-q)^i$.

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

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

It attains its maximum of $8e^{-1}$ when $NL^2 = 4$. Each visible face in a cell induces 1 or 2 visible edges in that cell. Thus, even though the number of edges (and faces) 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

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

Since $\lambda^2 e^{-\lambda} \leq 4e^{-2}$ 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 $G^2 = 9L^{-2}$ cells is

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

That bound is not tight; and the above constant factors could be reduced. Indeed the average number of edges incident on a cell caused by an visible face incident on that cell is 1.5. Also edges belonging to the same face are not tested for intersection. Therefore the actual number of intersection tests performed would be smaller than shown above, but still linear.

Note the surprising effect of the covered cells: the number of visible intersections is reduced from $\Theta(N^2L^2)$ 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.

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

The mass property formulae require knowing which vertices and edge intersections are outside all the faces. 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 face, and can be ignored. On the other hand, if $\mathcal{C}$ is not covered, if $\mathcal{P}$ is inside a face, then some edge of that face must pass through $\mathcal{C}$. Comparing $\mathcal{P}$ against all the edges passing through $\mathcal{C}$ suffices to classify $\mathcal{P}$. However, the expected number of edges 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.

4.4. 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 edge length. As the average number of polyedra (for simplicity, congruent isothetic cubes, i.e., cubes aligned with the coordinate axes) 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 polyedra incident on a non-covered cell tends to a constant.

More precisely, the number of polyhedra is $N/8$. Let $G = 2/L$, so the number of cells is $8L^{-3}$. The expected number of cells incident on a given polyedron is 27, giving a total of $27N/8$ such incidences, or $\lambda = \frac{27}{64}NL^3$ per cell. The actual number of polyhedra incident on a given cell follows a Poisson distribution. Since the probability of a polyedron that is incident on a given cell covering the cell is $q = 1/27$, r, the *visible* number of polyhedra per cell follows equation 1 with these q and λ, and so $\overline{r}$ is also a small constant here.

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

4.5. Intersection Determination in E^3

This section extends the timing analysis to visible intersections of a face and an edge, or of 3 faces, of polyhedra in E^3. Only intersections that are outside all the polyhedra are interesting.

Consider the 3-face intersections first. Each of the $\frac{3}{4}N$ faces is incident on 9 cells, so the number of faces incident on a cell follows a Poisson distribution, with $\lambda = \frac{27}{32}NL^3$. When a polyhedron is incident on a cell, from 0 to 3 of its faces are also incident on that cell. The probability that a polyhedron that is incident on a cell covers that cell is $q = 1/27$. If a cell is covered then define the number of visible faces to be 0, else it is the number of faces. Therefore, equation 1 still holds, with these new values of λ and q.

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

$$\overline{\binom{r}{3}} = \frac{8788}{59049}\lambda^3 e^{-\lambda/27}$$

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

$$N_{ft} \triangleq \frac{2197}{3072}L^6 N^3 e^{-\frac{L^3 N}{32}}$$

Since $L^6N^3e^{-c_1L^3N} = c_2Nw^2e^{-w} \leq c_3N$, where $w = c_1L^3N$,

$$N_{ft} \leq \frac{8788}{3e^2}N = \Theta(N)$$

The constant factor is included to show that, while it is not on the order of 1, it is workable. The analysis of the number of edge-face tests similarly gives $\Theta(N)$. Therefore the total time to find the visible vertices is

$$\Theta(N)$$

which is optimal.

§5. Limitations

This analysis has a few limitations.

1. The edges and faces are not completely independent of each other; e.g., each vertex has 3 incident edges, which will all be incident on the same cell. However, since each cube has a constant number of edges and faces, regardless of the total number of cubes, this effect becomes relatively smaller as N increases. It is insignificant for the values of N for which this algorithm was designed ($> 10^6$).

2. Edges and faces with a general slope will be incident on more cells than isothetic edges are. However, this adds only a small constant factor, which does not affect the asymptotic time or space.
3. Edges have varying lengths. However, the number of edges incident on a given cell is still a Poisson distribution.
4. If the sum of all the polyhedra's volumes gets very large, then the union volume approaches 1. If $GL \gg 1$ then all $(G-2)^3$ interior cells will be covered, while the, noncovered, cells adjacent to the universe's exterior will have ever more polyhedra, edges, and faces. The above analysis does not capture this unevenness, which can become important since the processing time in each cell grows as the cube of that cell's contents. However, such dense datasets are both unrealistic and also cause problems for any algorithm.
5. A more detailed analysis might confirm the experimental observation that G larger than $2/L$, perhaps large enough that the expected number of items per cell is constant, may significantly reduce the constant factor in the asymptotic value of T.
6. Storage grows almost as fast as G^3. For large N, there may not be sufficient storage for the optimal G. However, a proper perspective is important, since this algorithm is still very storage-efficient. The next section describes successfully using a $500 \times 500 \times 500$ grid.
7. Insignificant terms that are ignored might become significant sometime. E.g., the G^3 time to initialize the grid, depending on its concrete data structure, is ignored since for any reasonable G, processing the grid contents is much more costly. Nevertheless, for an unrealistically large G this time could be significant.

§6. Implementation Tests

UNION3 has been implemented for the restricted input case of congruent cubes, [13]. The biggest test was $N = 160,000,000$, i.e., 20,000,000 cubes, with over half a billion subfacets of various dimensions. Their edge length was 0.009; the grid was $500 \times 500 \times 500$. The execution time was 8300 CPU seconds, spread among 4 processes, on a 2.4GHz dual Xeon running linux, so that the elapsed time was only an hour. This example pushed the limits so that there was insufficient memory to use an optimal grid, or even covered cells, which increased the running time.

A typical smaller example that utilized covered cells was this: $3,000,000$ cubes of size 0.005 were processed with a $400 \times 400 \times 400$ grid in 1000 seconds.

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.) One solution to the memory problem would be to subdivide the problem, then execute the parts sequentially, not in parallel.

Even an apparently simple data structure such as the uniform grid is trickier to implement efficiently than may be apparent. For instance, the input polyhedra are processed against the grid twice. The first time is only to count the number of incidences on each grid cell. Then exactly the necessary storage for the cells is allocated with a ragged array. The polyhedra are then processed a second time, and now the incidence information is stored. This reduces the required storage by almost half, since linked lists and storage reallocations are not required, at a slight time cost.

Are the formulae used to compute the mass properties correct? Although errors are always possible, several indicators give confidence. First, the final volume is the sum of many addends of opposite signs, which almost cancel each other. Any error is likely to produce a clearly illegal volume. Second, nasty test cases, involving many cubes exactly incident on or aligning along faces, were tried. (This also tested the SoS code.) The computed volume was always correct.

Third, for random input, the volume may be estimated 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. 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 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. Extensions to collision detection of moving objects, each composed of the union of many simple polyhedra, are now being considered.

§8. References

1. Aronov, B., M. Sharir, and B. Tagansky. The union of convex polyhedra in three dimensions. SIAM J. Comput. **26**, (1997), 1670–1688.

2. Bieri, H. 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.

3. Cameron, S. Efficient bounds in constructive solid geometry. IEEE Comput. Graph. Appl. **11**(3), (1991), 68–74.

4. Center for Geometric and Biological Computing. External memory algorithms and data structures. `http://www.cs.duke.edu/CGC/subjects/external.html`, 2001.

5. Chazelle, B. An optimal algorithm for intersecting three-dimensional convex polyhedra. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, (1989), 586–591.

6. Chazelle, B. and D. P. Dobkin. Intersection of convex objects in two and three dimensions. J. ACM **34**(1), (1987), 1–27.

7. Descartes, R. *Principia Philosophiae.* Ludovicus Elzevirius, Amsterdam, 1644.

8. Dobkin, D. P. and D. G. Kirkpatrick. Fast detection of polyhedral intersection. Theoret. Comput. Sci. **27**(3), (1983), 241–253.

9. Dobrindt, K., K. Mehlhorn, and M. Yvinec. A complete and efficient algorithm for the intersection of a general and a convex polyhedron. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes Comput. Sci.*, (1993), 314–324.

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

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

12. Franklin, W. R. Polygon properties calculated from the vertex neighborhoods. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.* (1987) 110–118.

13. Franklin, W. R. Mass properties of the union of millions of polyhedra. In Dutta, D., R. Janardan, and M. Smid (eds.) *Computer Aided Design and Manufacturing.* American Mathematical Society, 2003.

14. Franklin, W. R., N. Chandrasekhar, M. Kankanhalli, V. Akman, and P. Y. Wu. Efficient geometric operations for CAD. In Wozny, M. J., J. U. Turner, and K. Preiss (eds.) *Geometric Modeling for Product Engineering.* Elsevier Science Publishers B.V. (North-Holland), (1990), 485–498.

15. Laidlaw, D. H., W. B. Trumbore, and J. F. Hughes. Constructive solid geometry for polyhedral objects. Comput. Graph. **20**(4), (1986), 161–170. Proc. SIGGRAPH '86.

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

17. Mirtich, B. Fast and accurate computation of polyhedral mass properties. J. Graphics Tools **1**(2), (1996), 31–50.

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

19. Narayanaswami, C. and W. R. Franklin. Determination of mass properties of polygonal CSG objects in parallel. Internat. J. Comput. Geom. Appl. **1**(4), (1991), 381–403.

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

21. Toma, L., R. Wickremesinghe, L. Arge, J. S. Chase, J. S. Vitter, P. N. Halpin, and D. Urban. 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`.

W. Randolph Franklin
ECSE Dept, 6003 JEC
Rensselaer Polytechnic Institute
110 8th St
Troy NY 12180
`research@wrfranklin.org`
`http://wrfranklin.org`