Operating on Large Geometric Datasets

W. Randolph Franklin
Rensselaer Polytechnic Institute
ECSE Dept, 110 8th Street, Troy, NY 12180, USA
frankwr@rpi.edu, http://wrfranklin.org/

1. INTRODUCTION

We describe some successful strategies for storing and operating on sets of up to tens of millions of simple geometric objects. Users wish to process datasets derived from laser scanners that may range up petabytes in size. At this point on the asymptotic time curve, superlinear times, even $T = \theta(N \log N)$, are too slow. We illustrate this with some implementations.

2. MINIMIZE THE EXPLICIT TOPOLOGY

A data structure that explicitly stores the mininum possible structure will be more compact. Surprisingly, this may also lead to simpler algorithms. For example, how simple can a polyhedron $\mathcal{P}$ be? Consider the set of faces, $\mathcal{F} = \{f_i\}$. That permits the following operations

1. *Inclusion determination:* Point x is contained in $\mathcal{P}$ iff a semi-infinite ray from x crosses an odd number of f_i.
2. *Volume computation:* The volume $\mathbb{V}$ of $\mathcal{P}$ is the sum of the volumes of all the pyramids determined by the f_i and the origin. Other mass properties follow similarly.

In no case was the global topology of the hierarchy of nested inclusions of shells of faces required, although it could have been derived if necessary.

However, a simpler data structure is possible. Suppose that we have only the set of incidences of vertices, edges lines, and oriented face planes. 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 $\mathbb{V} = -\frac{1}{6}\sum P \cdot \hat{T}\, P \cdot \hat{N}\, P \cdot \hat{B}$. 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.

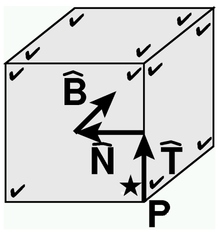

Figure 1: $P, \hat{T}\hat{N}, \hat{B}$

3. DESIGN FOR EXPECTED INPUT

We find pairs of coinciding input objects with a *uniform grid*, which is simple to implement and fast to execute. Although the worst-case time is $\theta(N^2)$ or $\theta(N^3)$, the expected time is $\theta(N)$.

4. SHORT-CIRCUIT OPERATOR COMPOSITIONS

E.g., computing the volume of the union $\mathcal{C}$ of two polyhedra $\mathcal{A}$ and $\mathcal{B}$ does not require completely computing $\mathcal{C}$. From Section 2, computing certain local information of $\mathcal{C}$ suffices. This principle becomes even more important with more complex operations. Suppose that we want the volume of the union of N polyhedra? The naive algorithm proceeds by uniting the polyhedra pair by pair, then four by four, and so on, building a tree of depth $\lg N$, to explicitly compute the union polyhedron before finding its volume. Worse, the intermediate polyhedra may have many more vertices than the final one. However, that whole tree may be flattened as described in the following section.

5. VOLUME OF THE UNION OF MANY POLYHEDRA

This is an algorithm to compute the volume of the union of many polyhedra in expected linear time. Algorithm details are in [3] and the time analysis in [2]. We have implemented and tested this for up to $3 \cdot 10^7$ identical isothetic cubes.

1. Superimpose a uniform grid on the input data. A good cell size is 1/2 the cube size.
2. Iterate through the input cubes. Mark any grid cell that is completely contained in a cube as *covered.* Record every non-covered cell that an input vertex, edge, face, or cube intersects, in a data structure indexed by the cell.
3. Initialize variables $\mathbb{V}$, $\mathbb{A}$ and $\mathbb{L}$ to zero. They will accumulate the total volume, surface area, and edge length of the union of the input cubes.
4. Iterate through the cells. In each cell,find all intersections of an edge and a face, or of three faces. These are potential output vertices. Other potential output vertices are the input vertices.
5. Test each potential output vertex against the input cubes in the same call, to cull those contained in any cube. The rest are the output vertices.
6. Knowing each output vertex's neighborhood from how it was formed, compute its contribution to $\mathbb{V}$, $\mathbb{A}$ and $\mathbb{L}$ and update them.

5.1 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

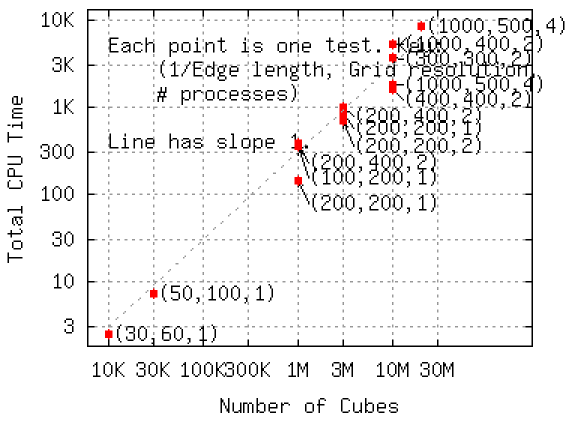

Figure 2: Execution Time vs Number of Cubes

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 2 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.

6. COMPUTING THE AREAS OF THE NON-EMPTY INTERSECTIONS OF PAIRS OF CELLS FROM TWO OVERLAPPING TRIANGULATIONS

Consider two different triangulations, $\mathcal{T}_1$ and $\mathcal{T}_2$, over the same region. In E^2, for the United States, $\mathcal{T}_1$ might be counties and $\mathcal{T}_2$ hydrography regions. A polygon p of $\mathcal{T}_1$ is generally not contained in any one polygon q of $\mathcal{T}_2$, but overlaps several q_j. Suppose that we know the populations of the p_i, and wish to infer the populations of the q_j. One way to estimate $pop(q)$ is to pro-rate the populations of the p_i overlapping q, each weighted by the fraction of $area(q)$ that overlaps each p_i. This *cross-area* problem requires determining all the pairs (p_i, q_j) with non-empty intersections, and the areas of those intersections.

The first optimization is not to work with individual polygons, but with the planar graph as a whole. The second is to realize that computing the area of the intersection of two polygons requires only the intersecting polygons' set of vertices together with their local topologies. That motivates the following algorithm.

1. Assume that for $\mathcal{T}_1$ and $\mathcal{T}_2$, we know each vertex's position, and each edge's adjacent vertices and polygons. No global information will be used.
2. Initialize a hash table $\mathcal{H}$ keyed by (i, j) with an entry for each (p_i, q_j) with non-zero intersection, and contents the partial or complete area of the corresponding polygon.
3. Computing the area of an intersection requires computing its vertices, and each vertex's local neighborhood, i.e., the directions of the two adjacent edges on it. These output vertices are input vertices or intersections of input edges.
4. Find all the intersections s between edges of $\mathcal{T}_1$ and $\mathcal{T}_2$ using a uniform grid. Each s is a vertex of four output intersection polygons. Compute s's contribution to those polygons' areas and create or update four entries in $\mathcal{H}$.
5. Process all the input vertices similarly.
6. The non-empty entries of $\mathcal{H}$ are the desired information.

An implementation in E^2 is described in [5], with code at [1]. The E^3 algorithm is described in [4]. The expected execution time, assuming i.i.d. input, is linear in the size of the input plus output, with the computation being quicker than the I/O. $\mathcal{T}_1$, coterminous US counties has 55068 vertices, 46116 edges, and 2985 polygons, while $\mathcal{T}_2$, hydrography polygons, has 76215 vertices, 69835 edges, 2075 polygons. The total time (excluding I/O) is 1.58 CPU seconds on a machine that is several years old.

7. ACKNOWLEDGEMENTS

This research was supported by DARPA/DSO/GeoStar.

8. REFERENCES

[1] W. R. Franklin. Overlay area calculation program. `ftp://ftp.cs.rpi.edu/pub/franklin/overlay.tar.gz`, 1994.

[2] W. R. Franklin. Analysis of mass properties of the union of millions of polygedra. In M. L. Lucian and M. Neamtu, editors, *Geometric Modeling and Computing: Seattle 2003*, pages 189–202. Nashboro Press, Brentwood TN, 2004.

[3] W. R. Franklin. Mass properties of the union of millions of identical cubes. In R. Janardan, D. Dutta, and M. Smid, editors, *Geometric and Algorithmic Aspects of Computer Aided Design and Manufacturing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 67, pages 329–345. American Mathematical Society, 2005.

[4] W. R. Franklin and M. S. Kankanhalli. Volumes from overlaying 3-D triangulations in parallel. In D. Abel and B. Ooi, editors, *Advances in Spatial Databases: Third Intl. Symp., SSD'93*, volume 692 of *Lecture Notes in Computer Science*, pages 477–489. Springer-Verlag, June 1993.

[5] W. R. Franklin, V. Sivaswami, D. Sun, M. Kankanhalli, and C. Narayanaswami. Calculating the area of overlaid polygons without constructing the overlay. *Cartography and Geographic Information Systems*, pages 81–89, Apr. 1994.