Parallel Volume Computation of Massive Polyhedron Union

W. Randolph Franklin

Rensselaer Polytechnic Institute
Troy, NY, USA

FWCG 2013, 25 Oct 2013

Partially supported by NSF grant IIS-1117277.
• Larger geometric datasets $\gg 10^6$ objects
• New parallel HW — restricted capabilities
• $\therefore$ Need new algorithms, data structures.

**Why parallel HW?**

• More processing $\rightarrow$ faster clock speed
• faster $\rightarrow$ more electrical power
• faster $\rightarrow$ smaller features on chip
• smaller $\rightarrow$ greater electrical resistance!
• $\Rightarrow \Leftarrow$.

• Serial processors have hit a wall.
Parallel HW features

- IBM Blue Gene / Intel / NVidia GPU / other.
- Most laptops have NVidia GPUs.
- Thousands of cores / CPUs / GPUs.
- Lower clock speed 750MHz vs 3.4GHz.
- Hierarchy of memory: small/fast $\rightarrow$ big/slow.
- Communication cost $\gg$ computation cost.
- Reads much faster than writes.
- Semaphores protecting parallel writes are very slow.
- Regular memory access much faster than random.
- Efficient for blocks of threads to execute SIMD.
  1–186 run Linux variants.
Geometric Databases

- Larger and larger geometric databases now available, with tens of millions of primitive components.
- Needed operations:
  - interference detection
  - boolean: intersection, union
  - planar graph overlay
  - mass property computation of the results of some boolean operation
- Apps:
  - Volume of an object defined as the union of many overlapping primitives. Two object interfere iff the volume of intersection is positive.
  - Interpolate population data from census tracts to flood zones.
Algorithm Themes

- I/O more limiting than computation \(\rightarrow\) minimize storage
- For \(N \gg 1000000\), \(\lg N\) nontrivial \(\rightarrow\) deprecate binary trees
- Minimize explicit topology, especially 3D.
- Plan for 3D; many 2D data structures not easily extensible to 3D, e.g., line sweep.
- E.g., Voronoi diagram: 2D is \(\Theta(N \lg N)\). 3D is \(\Theta(N^2)\)
- Optimize function composition, e.g. Volume o union.
Unifying Example: Volume of Union of Many Cubes

- Nice unifying illustration of several ideas.
- Do a prototype on an easy subcase (congruent axis-aligned cubes).
- Idea extends to general polyhedra.
- **Not** statistical sampling — exact output, apart from roundoff.
- **Not** subdivision-into-voxel method — the cubes’ coordinates can be any representable numbers.
Application: Cutting Tool Path

- Represent path of a tool as piecewise line.
- Each piece sweeps a polyhedron.
- Volume of material removed is (approx) volume of union of those polyhedra.
- Image is from Surfware Inc’s Surfcam website.
Traditional N-Polygon Union

- Construct pairwise unions of primitives.
- Iterate.

Time depends on intermediate swell, and elementary intersection time.
- Let $P = \text{size of union of an M-gon and an N-gon}$. Then $P = O(MN)$.
- Time for union (using line sweep) $T = \Theta(P \lg P)$.
- Total $T = O(N^2 \lg N)$.

Hard to parallelize upper levels of computation tree.
Problems With Traditional Method

- $\lg N$ levels in computation tree cause $\lg N$ factor in execution time. Consider $N > 20$.
- Intermediate swell: worse as overlap is worse. Intermediate computations may be much larger than final result.
- The explicit output polyhedron has complicated topology: unknown genus, loops of edges, shells of faces, nonmanifold adjacencies.
- Tricky to get right.
- However explicit output not needed for computing mass properties.
- Set of vertices with neighborhoods suffices.
Volume Determination

Box: \( V = \sum_i s_i x_i y_i z_i \)

\( s_i : +1 \text{ or } -1 \)

General rectilinear polygons:
- 8 types of vertices, based on neighborhood
- 4 are type +, 4 –
- Area = \( \sum_i s_i x_i y_i \)

- Rectilinear polyhedra: \( V = \sum_i s_i x_i y_i z_i \)
- \( \exists \) formulae for general polyhedra.
Properties

Represent output union polyhedron as set of vertices with neighborhoods.

- no explicit edges; no edge loops.
- no explicit faces; no face shells.
- no component containment info.
- general polygons ok: multiple nested or separate comps.
- any mass property determinable in one pass thru the set.
- parallelizable.
- This is very useful because computing only the output vertices and neighborhoods is much faster than also computing the edges, faces, and global topology.
Volume Computation Overview

• Find all vertices of output object.
• For each vertex, find location and local geometry.
• Map-reduce sum over vertices, applying formula.

Challenge: to find those vertices in expected constant time per vertex.
Finding the Vertices

3 types of output vertex:
• Input vertex,
• Edge–face intersection,
• Face–face–face intersection.

• Find possible output vertices, and filter.
• An output vertex must not be contained in any input cube.
• Isn’t intersecting all triples of faces, then testing each candidate output vertex against every input cube too slow?
• No, if we do it right.
3D Uniform Grid

Summary
• Overlay a uniform 3D grid on the universe.
• For each input primitive — cube, face, edge — find overlapping cells.
• In each cell, store set of overlapping primitives.

Properties
• Simple, sparse, uses little memory if well programmed.
• Parallelizable.
• Robust against moderate data nonuniformities.
• Bad worst-case performance on extremely nonuniform data.
• Ditto any hierarchical method like octree.

Advantage
• Intersecting primitives must occupy the same cell.
• The grid filters the set of possible intersections.
Adding the Cubes Themselves to the Grid

- For each cube, find cells it completely covers.
- When a cell is completely covered by a cube: nothing in that cube can contribute to the output. So:
  - Find covered cells first.
  - Do not insert objects into covered cells.
  - Intersect pairs and triples of objects in non-covered cells.

For cell size somewhat smaller than edge size, almost no hidden intersections found. Good.
Expected time = $\Theta(\text{size(input)} + \text{size(useful intersections)})$. 
Uniform Grid Qualities

- **Major disadvantage:** It’s so simple that it apparently cannot work, especially for nonuniform data.

- **Major advantage:** For the operations I want to do (intersection, containment, etc), it works very well for any real data I’ve ever tried.
Show that time to find edge–edge intersections in \( E^2 \) is linear in input+output size regardless of varying number of edges per cell.

- N edges, length 1/L, \( G \times G \) grid, \( \eta \) edges per cell.
- \( \eta = \lambda \eta \Delta = \frac{N}{G^2} (G/L + 1) \)
- Poisson distribution, parameter \( \lambda \eta \).
- Expected number of edge–edge tests: \( \bar{\eta}^2 - \eta \)
- \( \eta = \lambda \eta \) and \( \eta^2 = \lambda \eta^2 + \lambda \eta \).
- Expected number of intersection tests per cell: \( \lambda \eta^2 = \frac{N^2}{G^4} (G/L + 1)^2 \)
- Expected total number of intersection tests, over the \( G^2 \) cells: \( \frac{N^2}{G^2} (G/L + 1)^2 \).
- Total time: insert edges into cells + test for intersections
  \( T = \Theta \left( N(LG + 1) + \frac{N^2}{G^2} (G/L + 1)^2 \right) \).
- Minimized when \( G = \Theta(L) \), giving \( T = \Theta \left( N + N^2 L^{-2} \right) \).
- Q.E.D.
Face–Face–Face Intersection Details

- Iterate over grid cells.
- In each cell, test all triples of faces, each from a different cube.
- Three faces intersect if their planes intersect, and the intersection is inside each face (2D point containment).
- Then look up $s_i$ in a table and update accumulating volume.
- Cubes are easier than general polyhedra.
Point Containment Testing

Question:
- P is a possible vertex of the output union polyhedron.
- Is point P contained in any input cube?

Answer:
- Find which cell, C, (if any) contains P.
- If C is completely covered by some cube then P is inside the covering cube.
- Otherwise, test P against all the cubes that overlap C.
- Expected number of such cubes is constant, under broad conditions.
- Expect test time per P: constant.
Face–Face-Face Intersection Execution Time

- $N$: number of cubes; $1/L$: edge length; $1 \times 1 \times 1$ universe.
- Expected number of 3-face intersections $= \Theta(N^3L^{-6})$.

**Effect of Grid**
- Choose $G$: number of grid cells on a side $= 2L$.
- Number of face triples: $N^3$
- Prob. of a 3-face test succeeding $= N^{-2}L^{-6}$.
- Depending on asymptotic behavior of $L(N)$, this tends to 0.
- Prob. of 3 tested faces actually intersecting $= c$, indep. of $N$ and $L(N)$.
- Big improvement!

**Effect of Covered Cells**
- Expected number of 3-face intersections $= \Theta(N^3L^{-6})$.
- However, for uniform i.i.d. input, expected visible number: $\Theta(N)$.
- Prob. computed intersection is visible $= c$, indep. of $N$ and $L(N)$.
- Time to test if a point is inside any cube also constant.
- Total time reduces to $\Theta(N)$.
Parallel Implementation

• 32 thread OpenMP.
• Dual 3.4GHz Xeon, 128GB memory.
• Don’t slice up the input spatially.
• Inserting objects into cells uses atomic increment and capture.
• Very compact data structures.
• Compute element-cell incidences twice: 1st time just to count size of each cell.
• Then allocate ragged array.
• Finally compute again and populate array.
• Intersection testing and volume computation writes only with sum reduce.
• 824 executable lines of C++.
Results

• Fast small datasets: $N=1000$, $L=10$: $V=0.573$, $A=19$, $T=0.02s$.
• Medium: $N=1,000,000$, $L=100$, $G = 200$: $V=0.6$, $T=3.3s$.
• Feasible large datasets: $N=100,000,000$, $L=400$, $G=1000$: $V=0.765$, $A=831$, 57GB, $T=473s$.
• 10x faster with 32 threads than with one thread.
• 258,461,149 of the 1,000,000,000 grid cells were completely covered by some cube.
• Smaller cubes are faster: $N=100,000,000$, $L=500$, $G=1000$: $T=396s$, 50GB.

Output vertex types for big case:
• 186,366,729 of the 800,000,000 input vertices outside all cubes.
• 395,497,686 of the intersections of 3 of the 600,000,000 input faces.
• 811,328,383 of the intersections of one of the 600,000,000 input faces with one of the 1,200,000,000 input edges.
It compiles and runs w/o crashing; why look for trouble?

- Terms summed for volume are large and mostly cancel.
- Errors unlikely to total to a number in $[0,1]$.
- Compare to computed volume.
- Assume that coincidental equivalence is unlikely.
- Construct specific, maybe degenerate, examples with known volume.
Extensions

To general boolean ops:
• Intersection of many convex polyhedra quite easy.
• Any boolean op expressible in CNF as union of intersections (common technique in logic design for computer HW).

To general polyhedra:
• Formulae are messier.
• Roundoff error would be biggest problem.
• Fatal to miss an intersection.
• Compute using rationals, perhaps with GMPXX.
• Time cost: factor of 100?

Testing polyhedron validity: Illegal volume or volume change after rigid transformation $\rightarrow$ invalid.
Summary — To Process Big Geometric Datasets on Parallel Machines

Guiding principles:

• Use minimal possible topology, and compact data structures.
• Short circuit the evaluation of volume(union(cubes)).
• I/O sensitive: design for expected input.
• Use lots of memory; run BIG examples to show the linear time.

Allows very large 3-D datasets to be processed quickly.

Source code (prototype quality) freely available for nonprofit research and education; I welcome stress tests and error reports.