Home

Mass Properties of the Union of Millions of Polyhedra

Wm Randolph Franklin

Rensselaer Polytechnic Institute

video

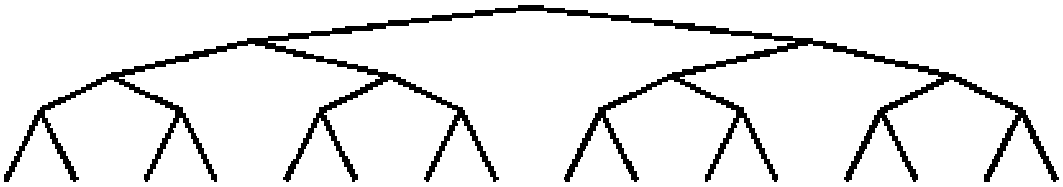
2. Goal

- Given many overlapping polyhedra:
- compute area, volume, etc, of the union.
- Useful for swept volumes, interference detection, etc.
- We present a prototype implementation for identical cubes.
- It's been tested to 20M cubes.
- Extendable to interference detection between two objects, each a union of many primitives.
- Extendable to online algorithm, with input presented incrementally.
- Extendable to deletions.

3. 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.

4. Traditional N-Polygon Union Method



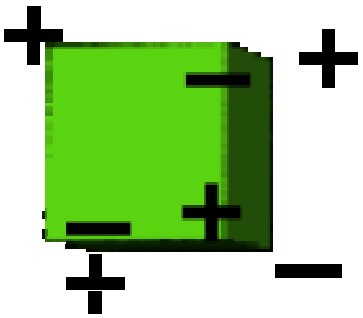
- Construct pairwise unions of primitives.
- Iterate.
- Time depends on assumed size of intermediates, and assumed elementary intersection time.
- Let P be size of union of an M-gon and an N-gon.
- $P=\Omega(\min(M,N))$. $P=O(MN)$.
- Time for union perhaps $T=\Theta(P \lg P)$.
- Total $T=\Omega(N \lg N)$, $T=O(N^2 \lg N)$.
- Hard to parallelize upper levels of computation tree.

5. Problems With Traditional Method

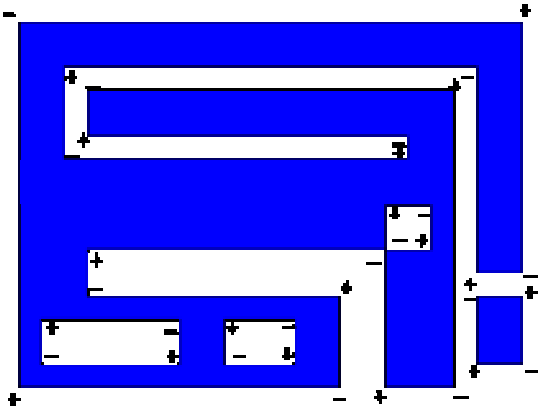
- *lg N* levels in computation tree cause *lg N* factor in execution time. If *N>20*, that's significant.
- *Intermediate swell*: gets worse as overlap is worse.
- The explicit volume has a complicated topology:
  - loops of edges,
  - shells of faces,
  - nonmanifold adjacencies.
  - hard to get right, and
  - not needed for mass properties.

6. Volume Determination

- Cube:
  - $V = \sum_i s_i x_i y_i z_i$
  - s_i: +1 or -1



- General rectilinear polygons:
  - 8 types of vertices, based on neighborhood
  - 4 are type "+", 4 "-"
  - Area = $\sum (s_i x_i y_i)$
  - Perimeter = $\sum (+\text{-}x_i + \text{-}y_i)$



7. Volume of Rectilinear Polyhedron

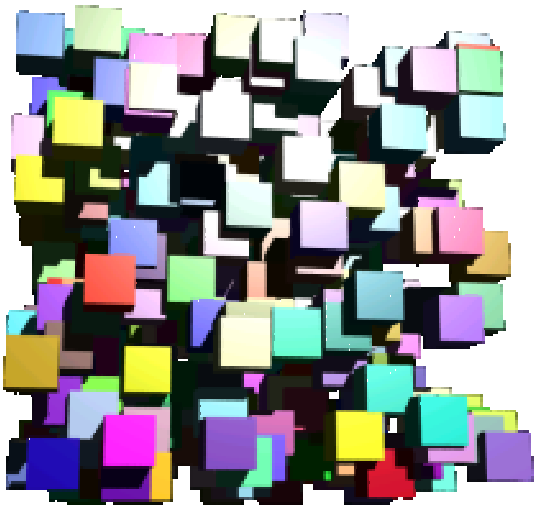
- $V = \sum_i s_i x_i y_i z_i$
- 16 classes of vertices
- $s_i = +1$ for 8 of them, -1 for 8.
- Extends to lower dimensional properties, like area and edge length.
- Extends to general polyhedra.

7.1 Properties

- no explicit edges; no edge loops
- no explicit faces; no face shells
- no component containment info
- can represent general polygons with multiple nested or separate components
- any mass property determinable in one pass thru the set
- parallelizable
- amenable to external storage implementation, if necessary.

8. Overview of Volume Computation

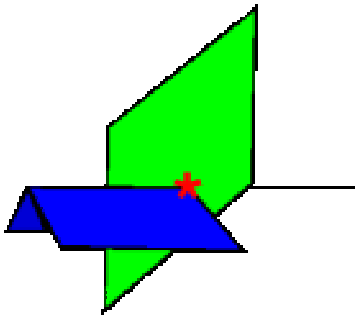
- Find all the vertices of the output object.
- For each vertex, find its location and local geometry.
- Sum over the vertices, applying the formula.



9. Finding the Vertices

3 types of output vertex:

- vertex of input cube
- intersection of edge of one cube with face of another,
- intersection of three faces of different cubes.



Find possible output vertices, and filter them

- An output vertex must not be contained in any input cube.

Isn't this too slow?

10. Use a 3D Grid

10.1 Summary

- Overlay a uniform 3D grid on the universe.
- For each input primitive (cube, face, edge), find which cells it overlaps.
- With each cell, store the set of overlapping primitives.

10.2 Properties:

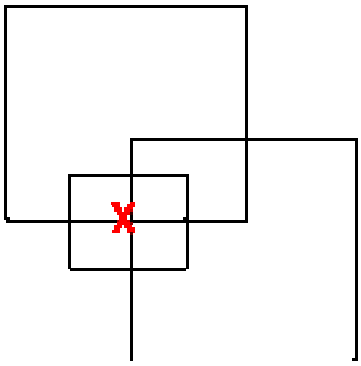
- Simple, sparse, uses little memory *if well programmed*.
- Parallelizable.
- Robust against moderate data nonuniformities.
- Bad worst-case performance: could be defeated be extremely nonuniform data.
- Ditto any hierarchical method like an octree.

10.3 Advantage of the Grid

- Intersecting primitives must occupy the same cell.
- The grid filters the set of possible intersections.

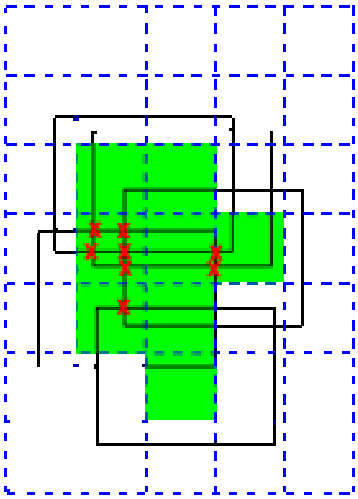
11. Covered Cell Concept

- Only visible intersections contribute to the output.
- That's often a small fraction - inefficient.
- Solution: add the cubes themselves to the grid.



12. Adding the Cubes Themselves to the Grid

- For each cubes, find which cells it *completely* covers.
- If a cell is completely covered by a cube, then nothing in that cube can contribute to the output.
- Find the covered cells first.
- Do not insert any objects into the covered cells.
- Intersect pairs and triples of objects in the noncovered cells.



If the cell size is somewhat smaller than the edge size then almost all the hidden intersections are never found. Good.

Expected time = Θ(size(input) + size(useful intersections)).

13. Face-Face Intersection Details

- Iterate over the grid cells.
- In each cell, test all triples of faces, where each face must be from a different cube, for intersection.
- Three faces intersect if their planes intersect, and the intersection is inside each face (2D point containment).
- If the 3 faces intersect, look up s_i in a table and add a term to the accumulating volume.
- The implementation is easier for cubes.

14. Point Containment Testing

Q: Is point *P* contained in any input cube?

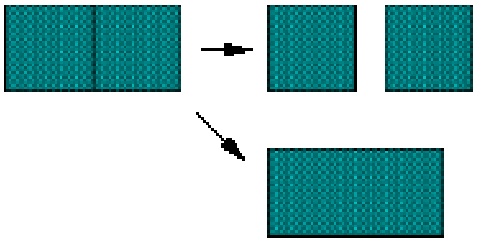
A:

- Find which cell, *C*, contains *P*.
- If *C* is covered then *P* is inside the covering cube.
- Otherwise, test *P* against all the cubes that overlap *C*.

The expected number of such cubes is constant, under broad conditions.

15. Simulation of Simplicity

- We must resolve degeneracies.
- For any equality, break the tie by comparing the cubes' ID numbers.
- However, now lower-dimensional mass properties give different results than for a regularized set operation.
- E.g., if two cubes abut on a face, then either both faces or neither face are included.
- That is topologically valid.



16. Execution Time

- Good choice of cell size: half the cube size.
- *N:* number of cubes
- *L:* edge length, in a *1x1x1* universe.
- Expected number of 3-face intersections = $\Theta(N L^6)$

16.1 Effect of Grid

- *G:* number of grid cells on a side = *2/L*.
- 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.
- With the grid, prob. of 3 tested faces actually intersecting = *c*, independent of *N* and *L(N)* .
- Big improvement!

17. Effect of Covered Cells

- Expected number of 3-face intersections = $\Theta(N^3L^6)$
- However, for uniform i.i.d. input, expected number of *visible* intersections = $\Theta(N)$.
- With covered cells, probability that a computed intersection that is visible = *c*, independent of *N* and *L(N)* .
- Big improvement!
- Time to test if a point is inside any cube is also constant.
- Total time reduces to $\Theta(N)$.

18. Limitations

- If the scene's depth is very large, and cubes are constrained to be completely inside the universe:
- Then all the interior cells, but none of the border cells are covered.
- This nonuniformity increases the time.
- If *L* is small, then proportionally small cells take too much memory.
- However, actually, there is a broad range of reasonable *G*.

19. Parallel Implementation

- Pentium-class processors allow only 3GiB memory per user process.
- Machine may have 12GiB real memory, so real memory > virtual memory!
- Solution: decompose problem into pieces and process in parallel.
- Cannot just slice up the input: this increases the area and edge length.
- Observation: inserting objects into cells is quicker than testing the cells for intersections.
- Solution:
    - Insert the {cubes, faces, edges} into the cells.
    - Dustribute the cells among the processors, and test for intersections in parallel.
    - Each process reads several GiB of data, but writes only 3 words: its part of the volume, area, length.

20. Implementation

- Very compact data structures.
- Random (Tausworth generator) uniform i.i.d. cubes.
- 1000 executable lines of C++.
- dual 2.4GHz Xeon with 4GiB memory, Linux.

20.1 Small Datasets are Fast

- 10,000 cubes
- 0.03 edge length
- 60x60x60 grid
- Not using parallelism
- Execution time: 2.5 CPU secs
- Memory: 15 MB.

20.2 Large Datasets are Feasible

- 20,000,000 cubes
- 0.009 edge length (quite small, few intersections).
- 500x500x500 grid
- Using 4 parallel processes.
- Execution time: 2000 to 2200 CPU secs per process. I.e., the work was evenly divided.
- Memory per process: 2.6 GB

Why not larger datasets? Address space limitations. Creating more parallel processes thrashes the memory. We're creating separate sequential processes, but still working on the interfacing.

21. Implementation Validation

1.
    - The terms summed for the volume range up to 10^{15}.
    - Errors would be unlikely to total to a number in [0,1].
2.
    - The expected volume is $1-(1-L^3)^N$.
    - Compare to computed volume.
    - Assume that coincidental equivalence is unlikely.
3.
    - Construct specific, maybe degenerate, examples with known volume.

22. Data Validity Test

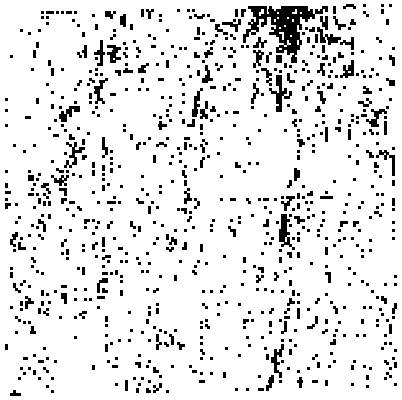
These formulae require consistent data.

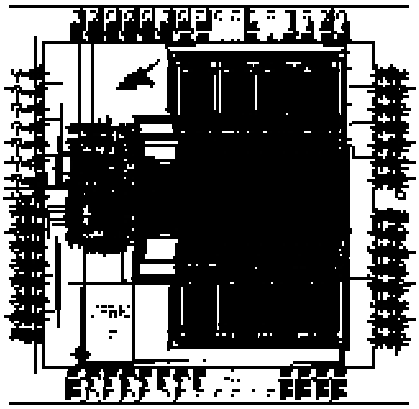
Use that property to validate the input data consistency:

- Randomly rigidly transform the polyhedron.
- If the computed mass properties change, the data is inconsistent.
- Otherwise, repeat.

23. Real Datasets are Nonuniform!

- Maybe we should subdivide the grid adaptively? Obvious, **but wrong!**
- The algorithm is very robust. Varying edge lengths are not a problem.
- Here are two datasets previously tested for edge intersection.
- Note how nonuniform they are.

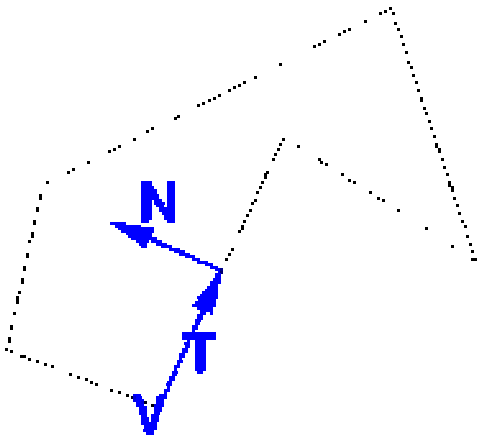




24. Extension to General Polygons

The only data type is the incidence of an edge and a vertex, and its neighborhood. For each such:

- **V** = coord of vertex
- **T** = unit tangent vector along the edge
- **N** = unit vector normal to **T** pointing into the polygon.



Polygon: {(**V, T, N**)} (2 tuples per vertex)

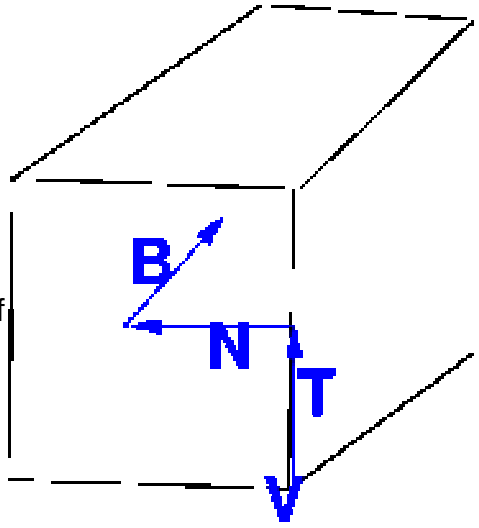
Perimeter = -Σ(**V.T**).
Area = 1/2 Σ(**V.T V.N**)

Multiple nested components ok.

25. Extension to 3D

The only data type is the incidence of a face, an edge and a vertex, and its neighborhood. For each such:

- **V** = coord of vertex
- **T** = unit tangent vector along the edge
- **N** = unit vector normal to **T** in the plane of the face, pointing into the face.
- **B** = unit vector normal to **T** and **N**, pointing into the polyhedron.



Polyhedron: {(**V, T, N, B**)}

Perimeter = -1/2 Σ(**V.T**)
Area = 1/2 Σ(**V.T V.N**)

Volume = -1/6 Σ(**V.T V.N V.B**)

26. Extension: Intersection Volumes from Overlaying 3-D Triangulations

Existing 2-D implementation: overlay two planar graphs; find areas of all intersection regions.

E.g. overlay counties of the coterminous US against hydrography polygons. Map stats:

Map #0: 55068 vertices, 46116 edges, 2985 polygons
Map #1: 76215 vertices, 69835 edges, 2075 polygons

Total time (excluding I/O): 1.58 CPU seconds.

Future: Extend to 3-D.

Application: Interpolate some property from one triangulation to the other.

27. Extension: More Complicated Boolean Operations

Given a red object defined as the union of many overlapping cubes, and a similar blue one.

- Do they intersect?
- Simplified to 2-D: N red edge segments, and N blue ones.
- There may be $O(N^2)$ red-red and blue-blue intersections.
- You don't know them.
- There are *O(1)* red-blue intersections.
- Find them.
- No subquadratic worst-case algorithm (SFAIK).

Moral: sometimes heuristics are necessary.

28. Other Possible Extensions

- Online algorithm, with incremental input insertions, would be easy.
- Online deletion of cubes should be only a little harder, if covered cells not used.
- This could be fast enough to be process hundreds of cubes at video update rates (1/30 sec).

29. Summary

Guiding principles:

- Use minimal possible topology, and compact data structures.
- Short circuit the evaluation of volume(union(cubes)).
- Design for expected, not worst, case input.
- External data structures unnecessary, tho possible.

Allows very large datasets to be processed quickly in 3-D with algorithms where sweep lines might have problems.

A paper on this subject is at http://www.ecse.rpi.edu/Homepages/wrf/research/union3/union3.pdf.

The html version of this talk is at http://www.ecse.rpi.edu/Homepages/wrf/research/union3/index.html.

Dr. Wm Randolph Franklin,
Email: Replace NOSPAM with RPI in: wrf@ecse.NOSPAM.edu
http://www.ecse.rpi.edu/Homepages/wrf/
+1 (518) 276-6077; Fax: 6261
ECSE Dept., 6026 JEC, Rensselaer Polytechnic Inst, Troy NY, 12180 USA

(GPG and PGP keys available)

file://localhost/wrf/web/research/union3/index.html, Thu, 09 Oct 2003 18:01:22 GMT