Operating on Large Geometric Datasets

FWCG2008

W. Randolph Franklin[1]

Rensselaer Polytechnic Institute
Troy, NY, 12180

1 Nov, 2008

[1] (518) 276-6077, frankwr@rpi.edu, http://wrfranklin.org/

The Problem and the conventional solution

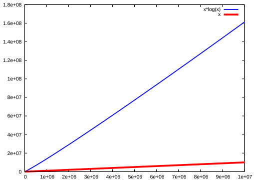
$\because$ Geometric datasets grow too large to easily fit into core.

$\therefore$ Must use external algorithms.

1. I/O is expensive.
2. Random I/O is very expensive.

My approach

1. Stay in core as long as possible — fast, random.
2. Use compact data structures.
3. Minimize explicit topological data structures.
4. Use input-sensitive algorithms
5. Optimize operator compositions.
6. Try for Linear time — superlinear times, even $T = \theta(N \log N)$, are too slow

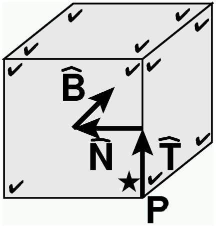


$$x \log(x) > x$$

Minimize the explicit topology

1. Explicitly storing the minimum possible structure saves space
2. and often facilitates simpler algorithms.
3. E.g., how simple can a polyhedron $\mathcal{P}$ be?
4. The set of faces, $\mathcal{F} = \{f_i\}$, often suffices.
5. That permits
    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.
6. Global topology of the hierarchy of nested inclusions of shells of faces is never required
7. That could have been derived if necessary.
8. We can get even simpler.

Set of Incidences



1. Polyhedron: $\{(P, \hat{T}, \hat{N}, \hat{B})\}$
2. One per incidence, 6 per cube vertex.
3. $\mathbb{V} = -\frac{1}{6}\sum(P\cdot\hat{T})\,(P\cdot\hat{N})\,(P\cdot\hat{B})$
4. That's a Google map reduce.
5. Irrelevant: multiple nested components, nonmanifold vertices.

Input-sensitive

1. Line segment intersections in E^2
   1. $K \triangleq$ number of intersections among N line segments of length L in E^2 1×1 region.
   2. $K_{\text{max}} = N^2/2$
   3. i.i.d input: $\overline{K} = N^2 L^2/4$
2. Visible edge intersections among overlaid squares
   1. Overlay, 1-by-1, N $L \times L$ random squares inside 1×1 region.
   2. Later squares hide earlier squares.
   3. $K \triangleq$ number of edge intersections
   4. i.i.d input: $\overline{K} = \theta(N^2 L^2)$
   5. $K_v \triangleq$ number of visible edge intersections
   6. $K_v\text{max} = N^2/2$
   7. i.i.d input: $\overline{K_v} = \theta(N)$
   8. independent of depth $((NL^2))$ of scene.

Time for visible edge intersections among overlaid squares

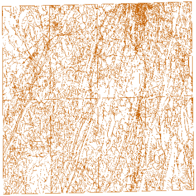
1. This linearity is key to linearity of volume of cube union later.
2. $N \triangleq$ number of squares, $L \triangleq$ edge length, $G \triangleq$ number of grid cells per side $= 3/L$
3. $\cdots$ tedious derivation$\cdots$
4. expected number of intersection tests performed per cell is

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

5. Total number of intersection tests

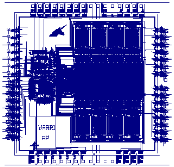
$$N_{tt} = 9L^{-2}N_{tpc} = 18N\left(1+4L^2N\right)e^{-\frac{L^2N}{4}} < 18(1+16e^{-1})N$$

Segment intersection is linear time for every non i.i.d application we've tried

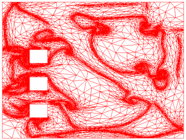


Roads

Counties, hydrog



VLSI



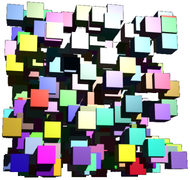
Nonuniform mesh

Optimize operator compositions

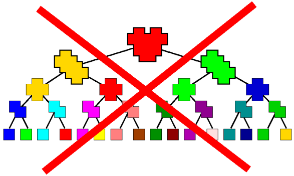
1. Goal$_1$ Volume of union of two polyhedra
2. Do not first compute union.
3. Requires only set of vertices of result, with their neighborhoods.
4. Does not require any global info. *No edges. No faces.*

1. Goal$_2$ Volume of union of many polyhedra
2. Requires only $\cdots$
3. Does not require $\cdots$
4. Does not require Building a computation tree of depth $\lg(N)$
5. No intermediate swell.

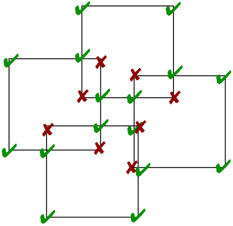
Volume of the union of many cubes



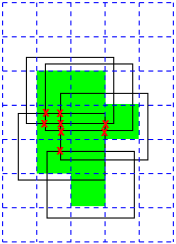
Union of cubes



Don't need this computation tree



Output vertices

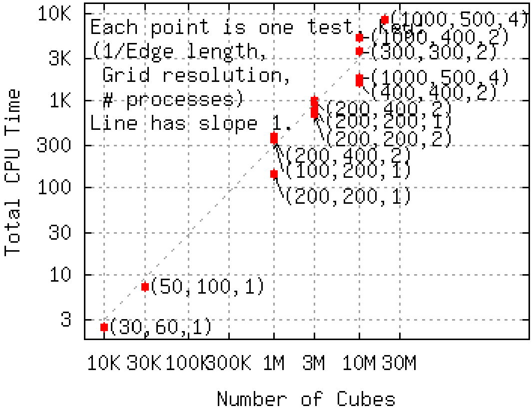


Culling possible intersections

Volume of the union of many cubes

1. $V = \sum s_i x_i y_i z_i$
2. This is exact, not Monte Carlo.
3. Output vertex is input vertex, or union of input face and edge, or union of 3 faces.
4. Output vertex is not in any input cube.
5. Determine neighborhood of each output vertex.
6. Superimpose uniform grid, proportional to cube size
7. In a cell contained in one cube, no output vertices in that cell.
8. Number of surviving vertices is linear.
9. Linear time.

Volume of the union of many cubes — implementation



Number of cubes vs time

1. $<$ 1000 lines of $C++$ on 2.4GHz dual Xeon.
2. *Input:* Up to 30 000 000 identical isothetic random cubes.

Conclusion

1. Simple, local topological, data structures
2. Optimizing composition of operators

facilitate

1. linear expected time algorithms
2. for processing large data structures