Geometric Operations on Millions of Objects

W. Randolph Franklin

Rensselaer Polytechnic Institute
Troy, NY, USA

UF Viçosa 24 Jul 2013

Partially supported by FAPEMIG and NSF grants CMMI-0835762 and IIS-1117277.

Large Geometric Datasets vs New HW Capabilities

- 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 !
- $\Longrightarrow\Longleftarrow$.
- 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
- Efficient for blocks of threads to execute SIMD.
- OS: 187th fastest machine in 6/2013 top500.org runs Windows. 1–186 run Linux variants.

Geometric Databases

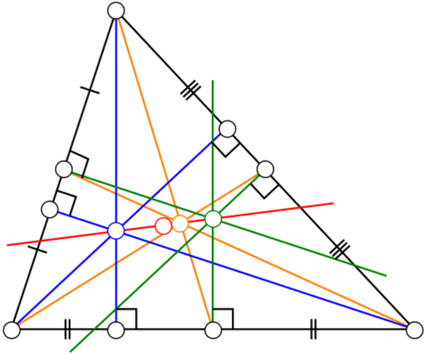
- Larger and larger geometric databases now available, with tens of millions of primitive components.
- Needed operations:
  - interference detection
  - boolean: intersection, union
  - planar graph overlay
  - mass property computation of the results of some boolean operation
- Apps:
  - Volume of 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, expecially 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 $\theta\left(N^2\right)$

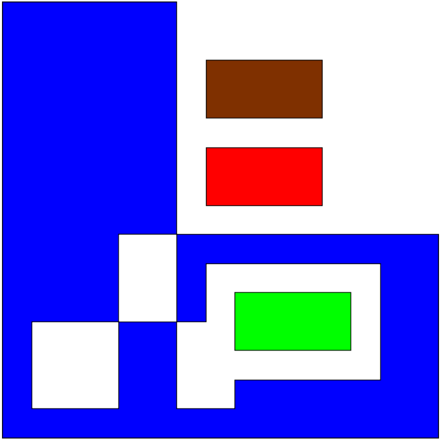
Confessions

- Not a deep philosophical thinker; always seeing holes in generalities.
- Prefer Galileo to Aristotle. Galileo experimented.
- Do small things well, lay a foundation, generalize.
- Driven by Euclidean geometry, where order is implicit in the axioms.
- Explicit representations unnecessary.
- Example of hidden order: the centroid, circumcenter, and orthocenter of a triangle collinear.



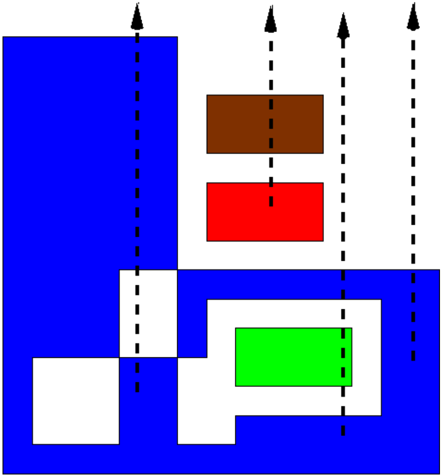
Theme: Minimum Explicit Topology

- What explicit info does the application need? Less $\rightarrow$simpler
- Object: polygon with multiple nested components and holes.
- Apps:
  - area
  - inclusion testing.
- Complete topology: loops of edges; the tree of component containments.
- Necessary info: the set of oriented edges.



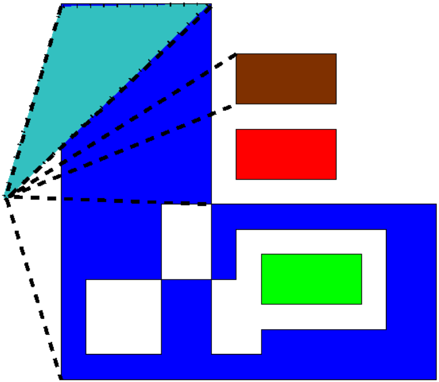
Point Inclusion Testing on a Set of Edges

- ”Jordan curve” method
- Extend a semi-infinite ray.
- Count intersections.
- Odd <==> Inside
- Obvious but bad alternative: sum subtended angles. Implementing w/o arctan, and handling special cases wrapping around 2π is tricky and reduces to Jordan curve.



Area Computation on a Set of Edges

- Each edge, with the origin, defines a triangle.
- Sum their signed areas $A(P) = \sum A(t_i)$

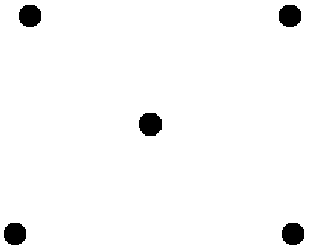


Advantages of Set of Edges Data Structure

- Simple enough to debug.
  SW can be simple enough that there are obviously no errors, or complex enough that there are no obvious errors.
- Less space to store.
- Easy parallelization.
  - Partition edges among processors.
  - Each processor sums areas independently, to produce one subtotal.
  - Total the subtotals.

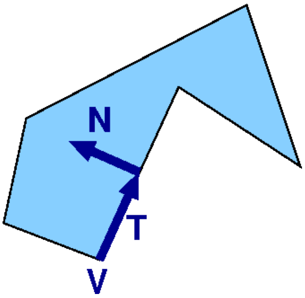
What About a Set of Vertices Data Structure?

- Too simple.
- Ambiguous: two distinct polygons may have the same set of edges.

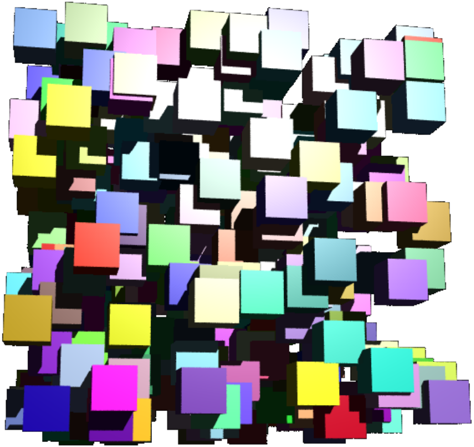


Set of Vertex-Edge Incidences

- Another minimal data structure.
- Only data type is 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 = $-\sum(V \cdot T)$.
- Area = $1/2 \sum(V \cdot T)(V \cdot N)$
- Multiple nested components ok.

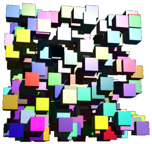


Demonstration: Mass Properties of the Union of Millions of Cubes



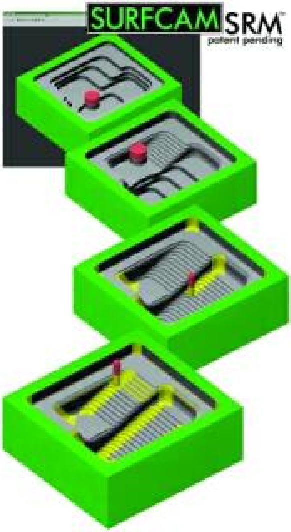
Unifying Example: Mass of Union

- Nice unifying illustration of several ideas.
- Do a prototype on an easy subcase (congruent axis-aligned cubes).
- However extends to general polyhedra.
- **Not** statistical sampling — exact output, apart from significant digit loss.
- **Not** subdivision-into-voxel method — the cubes' coordinates can be any representable numbers.

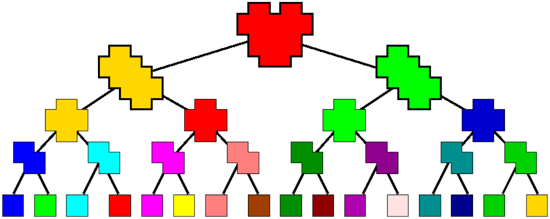


Application: Cutting Tool Path

- 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



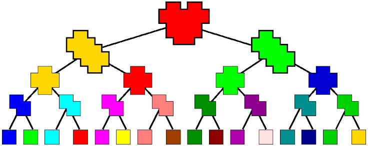
- Construct pairwise unions of primitives.
- Iterate.

Time depends on intermediate swell, and elementary intersection time.

- Let P = 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.

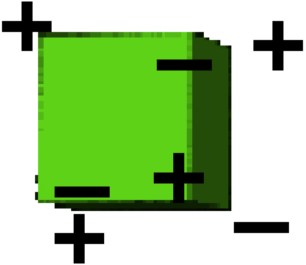
Problems With Traditional Method



- 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 volume has complicated topology: loops of edges, shells of faces, nonmanifold adjacancies.
- Tricky to get right.
- The explicit volume not needed for computing mass properties.
- Set of vertices with neighborhoods suffices.

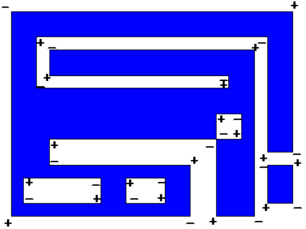
Volume Determination

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



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.

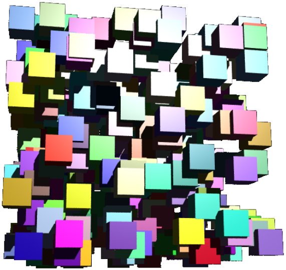
Properties

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.
- compatible with slow I/O.

Volume Computation Overview

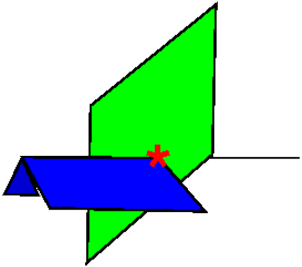
- Find all vertices of output object.
- For each vertex, find location and local geometry.
- Sum over vertices, applying formula.



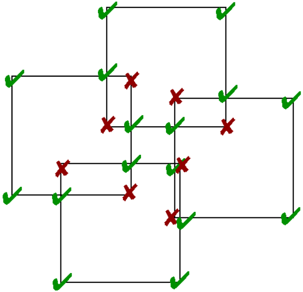
Finding the Vertices

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: defeatable by extremely nonuniform data.
- Ditto any hierarchical method like octree.

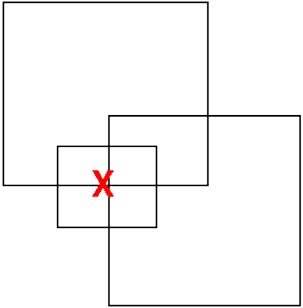
Advantage

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

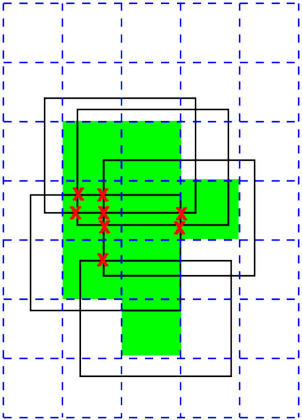
Covered Cell Concept

Optimization to prune objects before pairwise intersection tests.

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



Adding the Cubes Themselves to the Grid



- For each cubes, find cells it completely covers.
- When cell 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.

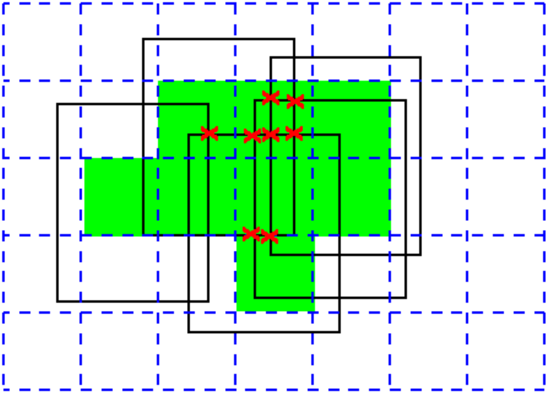
When cell size somewhat smaller than edge size, almost no hidden intersections found. Good.

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

Filter Possible Intersections

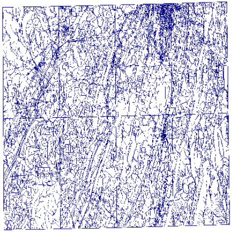
... by superimposing a uniform grid on the scene.

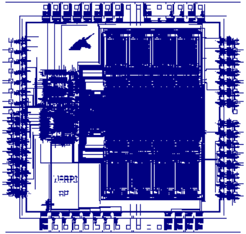
- For each input primitive (cube, face, edge), find which cells it overlaps.
- With each cell, store the set of overlapping primitives.
- Expected time = (size(input) + size(useful intersections)).

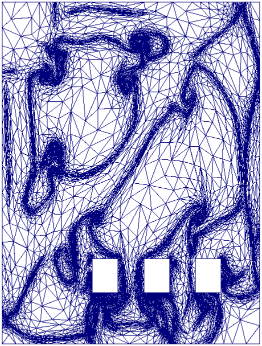


Uniform Grid Qualities

- 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.







USGS Digital Line Graph / VLSI Design / Mesh

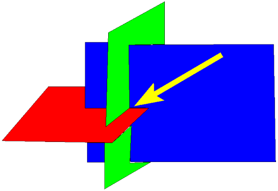
Uniform Grid Time Analysis

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 L, $G \times G$ grid, η edges per cell.
- $\overline{\eta} = \lambda_\eta = \frac{N}{G^2}(LG+1)$
- Poisson distribution, parameter λ_η.
- Expected number of edge–edge tests: $\overline{(\eta^2 - \eta)}$
- $\overline{\eta} = \lambda_\eta$ and $\overline{\eta^2} = {\lambda_\eta}^2 + \lambda_\eta$.
- Expected number of intersection tests per cell: ${\lambda_\eta}^2 = \frac{N^2}{G^4}(LG+1)^2$
- Expected total number of intersection tests, over the G^2 cells: $\frac{N^2}{G^2}(LG+1)^2$.
- Total time: insert edges into cells + test for intersections $T = \Theta\left(N(LG+1) + \frac{N^2}{G^2}(LG+1)^2\right)$.
- Minimized when $G = \Theta(1/L)$, giving $T = \Theta\left(N + N^2L^2\right)$.
- 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.
- Implementation easier for cubes.



Point Containment Testing

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

Answer:

- Find which cell, C, 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
- L: edge length, $1 \times 1 \times 1$ universe.
- Expected number of 3-face intersections = $\theta\left(N^3 L^6\right)$.

Effect of Grid

- Choose 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.
- 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 = $\theta(N^3 L^6)$.
- 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

(In progress)

- Can't slice up the input spatially: increases area edge length.
- Inserting objects into cells quicker than intersection testing.
- Solution:
  - Insert {cubes, faces, edges} into the cells.
  - Distribute cells among threads.
- Each thread reads much data, but writes only 3 words: its contribution to the volume, area, length.

Implementation

- Very compact data structures.
- Linear congruential rng not random for geometry, so use:
- Random (Tausworth generator) uniform i.i.d. cubes.
- 1000 executable lines of C++.
- Run on dual 3.4GHx Xeon, 128GB memory.
- Small Datasets are Fast: $N = 10^4$, $L = 1/20$, $G = 40$: T=0.64s, V=0.676, A=40.
- Medium: $N = 10^6$, $L = 1/100$, $G = 200$: T=37s, V=0.6, A=222. 3,125,877 output vertices, 2,775,644 face-face-face intersections.
- Large Datasets are Feasible: $N = 10^7$, $L = 1/200$, $G = 400$: T=395s, V=0.685, A=443. 24,868,405 output vertices, 33,996,760 face-face-face intersections.
-

Implementation Validation

~~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].
- - Expected volume: $1 - \left(1 - L^3\right)^N$.
  - 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 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 CGAL.
- 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)).
- Design for expected, not worst, case input.
- External data structures unnecessary, tho possible.

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