MINIMAL REPRESENTATIONS OF POLYGONS AND POLYHEDRA

W. Randolph Franklin
Electrical, Computer, and Systems Engineering Dept.
Rensselaer Polytechnic Institute
Troy NY USA 12180
mail@wrfranklin.org

Salles Viana Gomes de Magalhães
Departamento de Informática, Universidade Federal de Viçosa
Viçosa MG Brazil
salles@ufv.br

ABSTRACT

We present several simple representations of polygon and polyhedra that permit the efficient parallel computation of area and volume. They are particularly useful for computing the areas of the nonempty intersections between pairs of faces in two overlapping planar graphs in GIS, or the volumes of nonempty intersections between pairs of tetrahedra in two overlapping triangulations of a polyhedron in CAD. Both applications have been implemented on multicore Intel Xeons and tested on large datasets. The representations store the minimal types of information required for computation, and never need to store edge loops and face shells, or even most adjacency relations. The representations are sets of tuples or small fixed-size sets, and can be processed in parallel with map-reduce operations.

***K*eywords** topology · geometric computation · polygon properties

1 Introduction

This gem presents several different data structures to represent a geometric object. They facilitate efficient parallel computation of mass properties, especially when the object is the result of an intersection, union, or overlay. The object might be a polygon or polyhedron, and may have multiple nested or disjoint components. It may be a partition of a polygon or polyhedron, such as a 2D and 3D triangulation. 2D triangulations are also known as planar graphs and GIS maps. These ideas have been validated with several implementations in CAD and GIS, tested on large datasets.

Each data structure is simply a set of objects of some fixed-size type. The set has no extra structure, i.e., it is unordered. The element object type is an (ordered) tuple or an (unordered) set. This gem first reviews some classic data structures, such as representing a polygon $\mathcal{P}$ as a set of edges, where each edge is an ordered pair of its vertices. Then it presents several original data structures, created by the first author unless otherwise specified.

Each data structure is designed to optimize some operation. E.g., with the above data structure, $\mathcal{P}$'s area is easily computable by summing the areas of the triangles subtended by each edge at the origin. Each operation is expressible in the *map-reduce* form, and parallelizes quite well. We have implementations of a number of these in OpenMP on a multicore Intel Xeon.

Other operations include point inclusion testing in 2D and 3D, volume of the union of many isothetic cubes, and areas of all the nonempty intersections of the faces of two overlapping planar graphs. These ideas are particularly suitable for computing mass properties of boolean combinations because they require only incomplete information about the boolean combination, and so are easier to implement and run faster. Nevertheless, the computation is exact, apart from the usual floating point roundoff (apart from our implementations that compute exactly with big rationals).

These data structures have no pointers or trees. Explicitly storing global information like loops of edges and shells of faces is not necessary (although that information could be deduced if another application needed it).

These techniques can often reduce the storage needed to represent geometric objects, compared to representations that store complete topologies with many pointers from each element to all its adjacent elements. Since a good laptop computer may have 32GB of main memory, and a good workstation 256GB, there is the question of, what's the point? The reason is the datasets are also getting larger. There is an enormous speed advantage to being able to store the dataset in real memory instead of in virtual memory or on another node of a cluster. On modern supercomputers, the cost of accessing data often dominates the cost of processing it. Reductions in storage lead directly to reductions in total wall-clock time. Efficient, compact, data structures improve performance considerably.

Minimizing space usage is especially true when the target platform is a Graphics Processing Unit (GPU). GPUs are not just for gamers; as of June 2019, the fastest known supercomputer [26], the Summit, has 27,648 NVIDIA Volta V100 GPUs [24]. So, for maximum performance, we need to target GPUs. Although the Summit has over 10PB of memory, most of it is relatively slow to access. Each GPU has only up to 96KB of fast shared memory for each of its 14 streaming multiprocessors (SMs) [23]. Data too large for the fast shared memory must spill over to the main memory on the GPU, which is 100 times slower. A similar situation obtains when the platform is a network of processing nodes programmed with the Message Passing Interface (MPI). Accessing data over the network is orders of magnitude slower than accessing local data.

These ideas were partly motivated by Antoine de Saint-Exupery's proverb that,

> "A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away."

as a reaction to representations storing every possible topological relation. Some of these could well date to the invention of analytic geometry, although we are unaware of such prior art.

2 Representing one polygon

Here are several representations for one polygon $\mathcal{P}$. $\mathcal{P}$ need not be simply connected, and nested holes and islands are allowed; see Figure 1a. Non-manifold vertices, which have more than two edges, are also ok if they are considered to be several vertices whose coordinates coincide.

2.1 Unoriented edges

Here, $\mathcal{P}$ is represented as the set $\mathcal{U} = \{u_i\}$ where each $u_i = \{(x_{i0}, y_{i0}), (x_{i1}, y_{i1})\}$ is one unoriented edge of $\mathcal{P}$, represented as the set of its endpoints. E.g., the triangle with vertices $(0,0), (1,0), (0,2)$ would be represented as the set $\big\{\{(0,0),(1,0)\}, \{(0,0),(0,2)\}, \{(1,0),(0,2)\}\big\}$. Which side of u_i is on the inside of $\mathcal{P}$ is not stored. Since $\mathcal{U}$ is a set, its elements are not ordered.

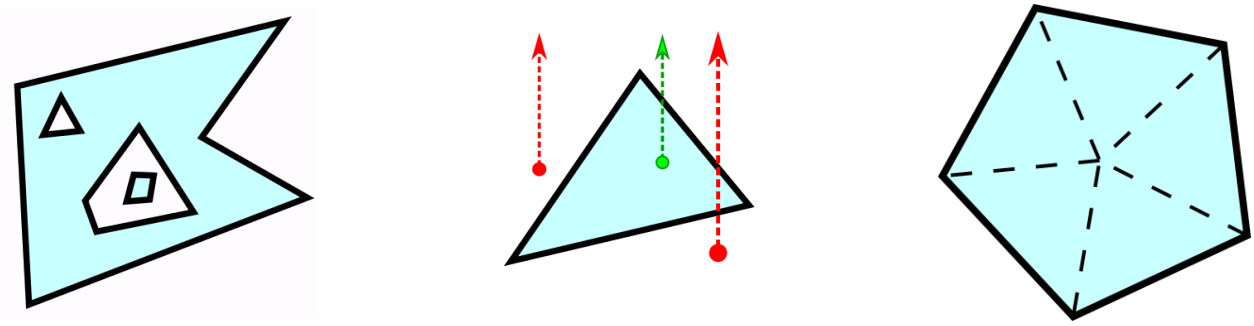
The set of unoriented edges suffices to uniquely represent $\mathcal{P}$. From a point-set view, it determines the points contained in $\mathcal{P}$. If the more global topology of $\mathcal{P}$ is required, such as the edge loops, then the two edges with a common vertex can be determined, so we can walk around each loop by traversing edge-vertex-edge-vertex... The set of loops can be determined with a union-find algorithm, or by repeating the previous traversal until every edge has been used. The hierarchical nesting of loops, such as shown in Figure 1a can be determined by inclusion testing of a sample vertex of each loop against each other loop. More efficient algorithms are available if speed is a problem. However the point of this gem is that we usually don't need this info.

This representation supports testing whether a query point q is included inside $\mathcal{P}$. Using the classic Jordan curve theorem [22], first implemented in [25], we test whether q is below each u_i. It runs an semi-infinite ray up from q and computes what edges it intersects. q is inside $\mathcal{P}$ iff the number of intersections is odd; see Figure 1b, where the green point is inside, but the red points outside.

2.2 Oriented edges

Here, $\mathcal{P}$ is represented as the set $\mathcal{E} = \{e_i\}$ where each $e_i = \big((x_{i0}, y_{i0}), (x_{i1}, y_{i1})\big)$ is one oriented edge of $\mathcal{P}$, represented as the ordered pair of its two endpoints. E.g., the triangle with vertices $(0,0), (1,0), (0,2)$ would be represented as the set $\big\{\big((0,0),(1,0)\big), \big((0,2),(0,0)\big), \big((1,0),(0,2)\big)\big\}$. The two vertices of each edge are ordered so that when traveling from the first vertex to the second, the inside of the polygon is on the left. The edge $\big((0,2),(0,0)\big)$ is different from the edge $\big((0,0),(0,2)\big)$.

This representation supports computing mass properties of $\mathcal{P}$ such as *area*, by summing the signed areas subtended by each edge and the origin. See Figure 1c.

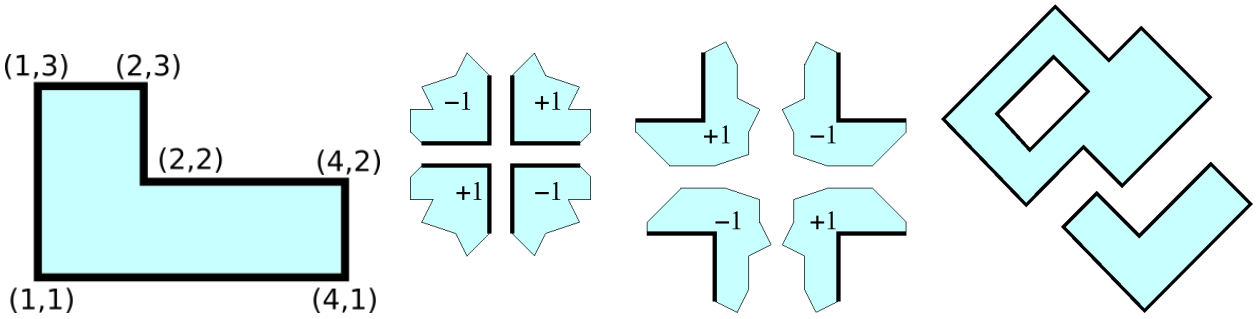


(a) Polygon with holes and island

(b) Jordan curve point inclusion testing on set of unoriented edges

(c) Areas subtended by edges

Figure 1: General polygon figures



(a) Axis-aligned polygon

(b) The 8 Cases of s

(c) Diamond Polygon

Figure 2: Axis-aligned and diamond polygons

With oriented edges, point inclusion becomes a little easier if the order of the edges intersecting the ray drawn up from the query point q towards $y = \infty$ is known, or alternatively, all the edges intersecting closer than a certain distance, or the closest intersecting edge, are known. This can be determined with, e.g., a uniform grid, often in constant expected time. By inspecting which side of the closest inspecting edge that q is on, we immediately know whether q is inside $\mathcal{P}$.

This can also be used to locate a point in a planar graph, i.e., to compute which face (aka polygon) of a planar graph contains q. That is a subroutine of Overprop [20], and in 3D of PinMesh [3]. The expected preprocessing time is linear in the number of vertices; the expected query time constant.

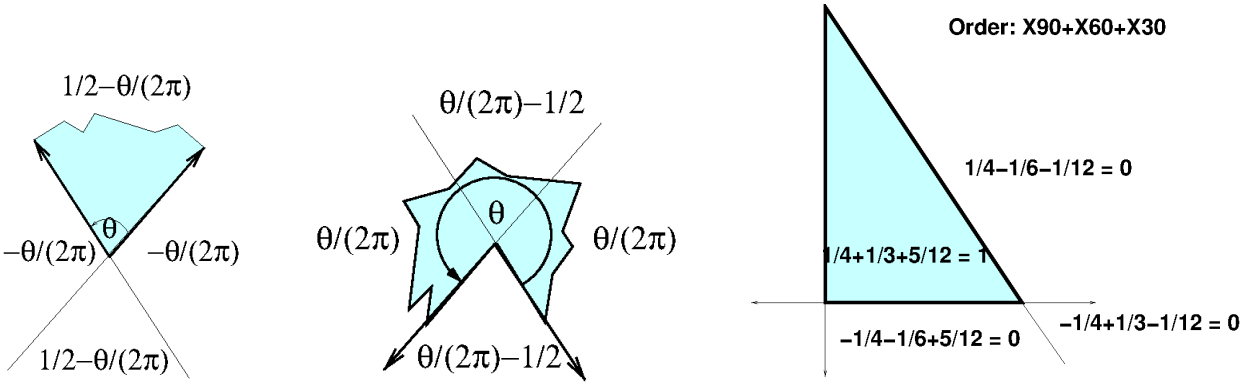
2.3 Augmented axis-aligned vertices

This is the special case of a polygon whose edges are all horizontal or vertical. It motivates the more general case to follow. The set of vertices is insufficient to uniquely define a polygon. However, augmenting each vertex with a little extra info suffices. For axis-aligned edges, as in Figure 2a, one additional bit of information is sufficient to compute mass properties such as area.

Define an augmented vertex as $a = (x, y, s)$ where (x, y) is the vertex's coordinates, and s is a property of that vertex's local geometry. There are eight cases; s is $+1$ for four and -1 for the other four, as shown in Figure 2b. Then the area is $\sum x_i y_i s_i$, summed over the augmented vertices of the polygon. E.g., the area of Figure 2a is

$$1 \cdot 1 \cdot 1 - 1 \cdot 4 \cdot 1 + 1 \cdot 4 \cdot 2 - 1 \cdot 2 \cdot 2 + 1 \cdot 2 \cdot 3 - 1 \cdot 1 \cdot 3 = 5$$

Testing whether point q is inside is possible by computing which quadrant of each vertex includes q, combining that with that vertex's s to form a characteristic function of $\mathcal{P}$, and summing. (The characteristic function $\chi_{\mathcal{P}}(q) = 1$ iff q is contained in $\mathcal{P}$ and 0 otherwise.)



(a) χ Function For a Vertex with $\theta < \pi$

(b) χ Function For a Vertex with $\theta > \pi$

(c) The Characteristic Function of a Triangle By Summing χ for each Vertex

Figure 3: Vertex neighborhood for general polygons

2.3.1 Implementation considerations

Although this data structure is provided for pedagogical purposes as a simple demonstration of the concepts, efficiently storing that extra bit per vertex may be of interest. First, we might use a separate bit vector for all the vertices' bits. This is easy and efficient with modern languages. Alternatively, if the vertex coordinates don't need the full precision of, say, a four byte int, we might steal one bit from a coordinate field, reducing its precision to only 31 bits.

2.4 Augmented diagonal vertices

This is a similar special case of polygons whose edges all have slopes of ± 1; see Figure 2c. As before, we supplement each vertex with one additional bit of information, used to compute mass properties such as area. Define an augmented vertex as $a = (x, y, s)$ where (x, y) is the vertex's coordinate, and s is defined similarly to in Figure 2b.

Then there is a whole class of area functions:

$$\forall\alpha\forall\beta\forall\gamma\forall\delta \quad A = \sum_i (\alpha x_i^2 + (1-\alpha) y_i^2 + \beta x_i + \gamma y_i + \delta) s_i$$

Special cases include $A = \sum_i x_i^2 s_i$ and $A = \sum_i y_i^2 s_i$.

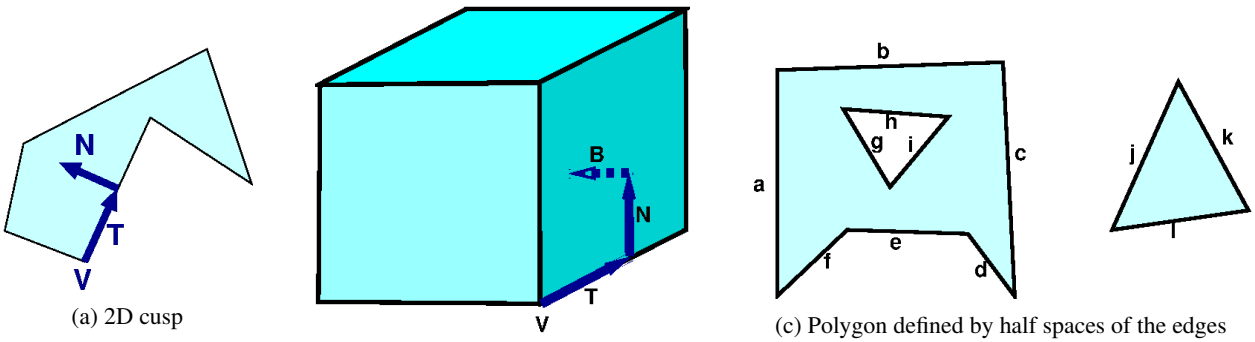
2.5 Augmented general vertices, aka vertex neighborhoods

A similar algorithm exists for general polygons. Since the set of vertices is insufficient uniquely to represent a polygon, we will augment each vertex V with information on its local neighborhood by including a generalization of the above s. This will abstract the information about the directions that the edges leave the vertex, and which sector is interior to the polygon.

Specifically, for each vertex, v, define $\chi_v : E^2 \to \Re$, i.e., a scalar function on all points in the plane depending on how they relate to v's neighborhood, as illustrated in Figure 3a. For $\theta < \pi$:

$$\chi_v(p) = \begin{cases} \frac{1}{2} - \frac{\theta}{2\pi} & \text{if } p \text{ is in the angle between the two rays} \\ -\frac{\theta}{2\pi} & \text{if } p \text{ is in the other 2 wedges} \\ \frac{1}{4} - \frac{\theta}{2\pi} & \text{if } p \text{ is on either ray, but not on } v \\ 0 & \text{if } p = v \end{cases}$$

For $\theta > \pi$:



(a) 2D cusp

(b) One 3D cusp of a cube

(c) Polygon defined by half spaces of the edges

Figure 4: Cusps and half-spaces

$$\chi_v(p) = \begin{cases} \frac{\theta}{2\pi} - \frac{1}{2} & \text{if } p \text{ is in the angle between the two rays as shown in Figure 3b} \\ \frac{\theta}{2\pi} & \text{if } p \text{ is in the other 2 wedges} \\ \frac{\theta}{2\pi} - \frac{1}{4} & \text{if } p \text{ is on either ray, but not on } v \\ 0 & \text{if } p = v \end{cases}$$

$\theta = \pi$ does not occur since it would be a straight line with no vertex.

Adding the χ functions for all the vertices of a polygon gives a characteristic function for the polygon. Figure 3c shows an example of a triangle. This can be extended to all polygons, including concave with multiple components and holes. Other formulae are presented in [12, 13, 17]. With this characteristic function, we can do point inclusion tests.

In spirit, this slightly resembles Green's theorem. However, while Green's theorem computes a polygon's area by integrating along its boundary edges, this formula uses only the boundary vertices.

2.6 Vertex–edge direction adjacencies, or cusps

We can get even simpler than the above representation, by splitting each vertex into two pieces called *cusps*, one for each adjacent edge. Note that we do not know the complete edge, but only its angle and which side is inside. See Figure 4a. Let v be the vertex's position. It is convenient to represent the extra info as two unit vectors, t and n. t is a unit tangent vector along the edge, and n is a unit vector normal to t pointing to the inside side of the edge. n adds only one bit of new info. So, $\mathcal{P}$ is now $\{(v, t, n)\}$. A k-gon will have $2k$ set elements.

Then the polygon's area is

$$\frac{1}{2}\sum_i (v_i \cdot t_i)(v_i \cdot n_i)$$

where $\cdot$ is the scalar vector product here. The perimeter (boundary) length is

$$-\sum_i (v_i \cdot t_i) \quad .$$

For example, consider a rectangle with vertices $(0,0), (2,0), (2,3), (0,3)$. $(0,0)$ has two adjacent edges. The edge from $v = (0,0)$ to $(2,0)$ has tangent vector $t = (1,0)$ and normal vector $n = (0,1)$. The edge from $v = (0,0)$ to $(0,3)$ has tangent vector $t = (0,1)$ and normal vector $n = (1,0)$. So the two cusps created from vertex $(0,0)$ are $\big((0,0),(1,0),(0,1)\big)$ and $\big((0,0),(0,1),(1,0)\big)$. Those two cusps contribute $0 \cdot 0 + 0 \cdot 0 = 0$ to the area and $0 + 0 = 0$ to the length.

2.7 Halfplanes

If we extend each edge of $\mathcal{P}$ to an infinite line and then to a halfplane of the inside side, then we may represent $\mathcal{P}$'s interior, in Constructive Solid Geometry (CSG) style, as a conjunctive normal form boolean expression of them, with each line used only once, Dobkin et al [6]. See Figure 4c for example, where

$$\mathcal{P} = abc(d + e + f)(g + h + i) + jkl$$

may be used to test point inclusion. In more detail, a, for example, represents a half plane defined by that edge extended to infinity, with the half plane interior on the inside side of the edge. In the above formula, $+$ means union and an omitted multiplication operator, intersection.

This concept does not extend to E^3; the formula may have to use some lines more than once. Also computing volume is an open question.

3 Representing one polyhedron

This section presents several data structures to represent a 3D polyhedron $\mathcal{P}$. A complete description would include the loops of edges and shells of faces, with many explicit adjacency relations, such as pointers from each face to its adjacent vertices and edges. However that is unnecessary for the computation of many properties, and not storing that saves a lot of space. To simplify the descriptions, assume that the faces of $\mathcal{P}$ are triangles.

3.1 Unoriented faces

To test whether $\mathcal{P}$ contains a test point q, it suffices to run a semi-infinite ray up from q, and count how many faces of $\mathcal{P}$ it crosses (the Jordan curve algorithm). When testing q against a face of $\mathcal{P}$, we do not need to know which side of the face is on the interior of $\mathcal{P}$. That is, the set of *unoriented faces* suffices.

3.2 Oriented faces

The volume of $\mathcal{P}$ can be computed by decomposing $\mathcal{P}$ into tetrahedra and summing their signed volumes. Each tetrahedron is defined by one face of $\mathcal{P}$ and the coordinate origin (or any other fixed point). The volume of a tetrahedron is a determinant whose entries are simple expressions of the tetrahedron's vertices. The volume may be negative. This algorithm requires knowing the inside side of each face, that is, the faces are *oriented.*

3.3 Axis-aligned faces

Perhaps the faces of $\mathcal{P}$ are all parallel with one of the xy, xz, or yz planes. This may not be that common, but when it does happen, the volume formula is particularly simple. The vertices of such a polyhedron may be either convex or concave, and may open out in any of eight directions, making 16 possible types of vertices. Each type of vertex is assigned a type bit, s, in a 3D version of Figure 2b. Then the volume of $\mathcal{P}$ is $\sum_i x_i y_i z_i s_i$, summed over all the vertices.

3.4 Vertex neighborhoods

For v, a trihedral vertex in E^3, $\chi_v(p)$ is defined using the vertex angles of the adjacent faces. Let the 3 faces be f_1, f_2, and f_3, and their angles at v be α, β, and γ, respectively. Then

$$\chi_v(p) = \begin{cases} \frac{2\pi-\alpha-\beta-\gamma}{4\pi} & \text{if } p \text{ is inside all 3 face planes.} \\ \frac{\alpha-\beta-\gamma}{4\pi} & \text{if } p \text{ is outside } f_1 \text{ but inside } f_2 \text{ and } f_3. \\ - - - & \text{etc. for two similar cases.} \\ \frac{\alpha+\beta-\gamma}{4\pi} & \text{if } p \text{ is outside } f_1 \text{ and } f_2 \text{ but inside } f_3. \\ - - - & \text{etc. for two similar cases.} \\ \frac{\alpha+\beta+\gamma-2\pi}{4\pi} & \text{if } p \text{ is outside all 3 face planes.} \end{cases}$$

3.5 Cusps

We can also represent polyhedron $\mathcal{P}$ as a set of 3D *cusps*, where each cusp is an adjacency relation between a vertex, and edge, and a face. However, neither the complete edge or face is stored, but only the definitions of the infinite lines or planes that include them, plus a face's inside side. Specifically, v is a vertex's coordinates, t is a unit tangent vector from v along an edge, and n is a unit vector normal to t in the plane of a face of $\mathcal{P}$ that points into the interior of that face. (Unlike in 2D, n adds more than one bit of information.) Define also one more unit vector, b, perpendicular to both n and t, pointing into the interior of $\mathcal{P}$. b adds only one bit of information. $\mathcal{P}$ is represented as the set of its cusps.

E.g., a cube has 48 cusps, since each of 8 vertices is incident on 3 edges, each of which is incident on 2 faces; see Figure 4b. In general, a vertex incident on k faces will be part of $2k$ cusps.

Some mass properties may be computed as follows.

- The total edge length, $L = -\frac{1}{2}\sum_i v_i \cdot t_i$. The $\frac{1}{2}$ is needed because each edge is counted twice.
- The surface area, $A = \frac{1}{2}\sum_i (v_i \cdot t_i)\ (v_i \cdot n_i)$.
- The volume $V = -\frac{1}{6}\sum_i (v_i \cdot t_i)\ (v_i \cdot n_i)\ (v_i \cdot b_i)$.

The volume, area, and length formulae may be derived by dropping a perpendicular from $\mathcal{O}$ onto each face, and onto each edge, and joining $\mathcal{O}$ to each vertex. All these edges, plus the faces joining them, partition $\mathcal{P}$ into one simplex for each cusp. The volume of each simplex is one term of the volume, area, or length formula for $\mathcal{P}$.

4 Don't we always know the edges?

One might argue that, in practice, we always know the edges, and so, vertex-based representations are pointless.

However, perhaps the polygon is the output of a function, such as the boolean intersection of two other polygons. One application is computing whether two polygons interfere or collide with each other, which happens iff their intersection area is nonzero. In this case, computing only the output vertices and their neighborhoods may be much easier than computing the output edges. Indeed, there are two types of output vertices here: (a) some input vertices, and (b) some intersections of input edges. Since each output vertex must be inside an input polygon, we filter the candidate output vertices with a point inclusion function. Note that this application requires the input polygons' edges; its output is a different representation from its input.

Additionally computing the output edges would require partitioning the input edges at their intersections with each other, requiring sorting the intersections along each edge, and then testing whether to include each resulting segment. That is much more work, and also harder to express in a map-reduce form.

5 Applications

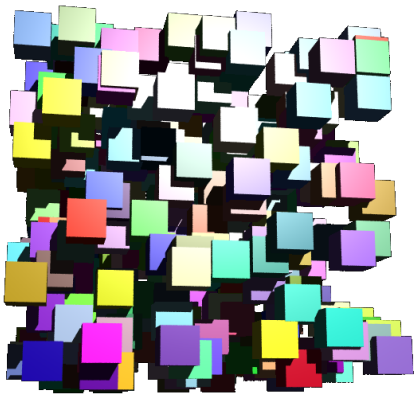
The above representations permit certain problems to be solved much more efficiently, and in parallel, such as computing mass properties of geometric intersections and unions. Here are two example problems, for which we have have been designed, implemented, and tested algorithms on large datasets. The current implementations are on multicore Intel Xeons, but the techniques would be even more useful on manycore Nvidia GPUs. Smaller test implementations are now being developed on that platform.

5.1 Volume of union of cubes

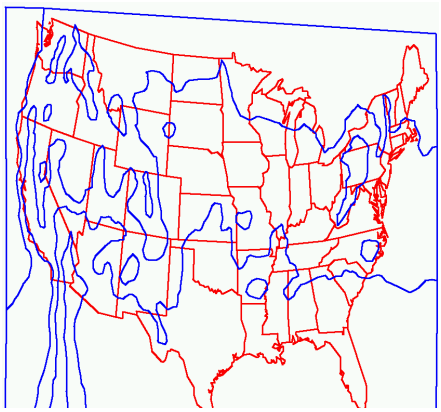
The problem is to compute the volume of the union of tens of millions of identical isothetic cubes; the idea is shown with 30 cubes in Figure 5a. The algorithm optimizes the composition of the union and the volume operations, so that it is not necessary to compute the explicit output union polyhedron. It is sufficient to compute only the set of output vertices, together with their neighborhoods. A vertex's neighborhood consists of the directions of any adjacent edges and faces, and for a face, which side is interior. The neighborhood does not include the other vertices on those adjacent edges and faces. Not needing to compute that is a considerable simplification. Components of the algorithm include finding all the face-face-face intersections and edge-face intersections among the input cubes. If the input is independently and identically distributed uniform random, then there is an equation for the expected output volume, and it agrees with what is computed. Theoretical analysis and implementation and test results are in [7, 16].

5.2 Overlay

From [8], "Suppose that a polygon $\mathcal{P}$ has been meshed, or partitioned, into smaller faces, in two different ways, M_0 and M_1 for two different applications. Each mesh is optimal for some application, and would be suboptimal for the other application. Assume that the faces of M_0 have some property, such as mass, that would be useful for the faces of M_1. (If the density varies, then the mass is not simply the area.) One quick approximation for the mass of f', a face of M_1. is a weighted sum of the masses of the overlapping faces of M_0, with the weights being the areas of the intersections of f' with the overlapping faces of M_0. The compute-bound component is to identify all the nonempty intersections of any face of M_0 with any face of M_1, and compute their areas. PAROVER2 is an algorithm and preliminary parallel



(a) Union of many cubes



(b) Overlaying state and isotherm maps

Figure 5: Applications

implementation for that, whose execution time is linear in the number of intersections, which is generally linear in the number of faces. PAROVER2 uses local topological formulae to compute the areas of the output faces (aka outfaces) from their vertices and half-edges."

This builds on ideas and implementations first described in [20, 21]. Figure 5b illustrates overlaying maps (planar graphs) for US coterminous states and July isotherms. [2] reports on using EPUG-Overlay to intersect the surface drainage system of the United States (21,652,410 vertices, 21,060,354 edges, and 219,831 faces) with the 2010 United States Census block groups (32,762,740 vertices, 32,103,306 edges, and 518,837 faces). Computing the overlaid planar graph, excluding I/O, took 322 elapsed seconds on the dual 8 core Xeon. The parallel speedup was a factor of 11. This also used multiprecision rational numbers to avoid roundoff errors.

[11] reports on intersecting two 3D meshes using these algorithms, in addition to big rationals and other techniques. Extensions that add one or more of 3D, rational numbers to avoid roundoffs, and Simulation of Simplicity to handle geometric degeneracies include [1, 2, 4, 5, 8–11, 14, 15, 18, 19, 21, 27].

6 Checking Polytope Validity

All these formulae require that $\mathcal{P}$ be a legal polytope; otherwise the volume is not even defined. Indeed, in that case, the computed volume will not even be invariant with respect to many rigid transformations. Therefore, we may test $\mathcal{P}$'s legality with a Monte Carlo algorithm as follows. Perform several random transformations, such as translations, scalings, or rotations, whose effect on $\mathcal{P}$'s volume is known. If $\mathcal{P}$'s volume does not always change correctly, then $\mathcal{P}$ is illegal. Otherwise, $\mathcal{P}$ probably does not have any errors such as missing facets. However, this does not detect errors such as $\mathcal{P}$'s boundary including a double loop.

7 Summary

These equations are reductions; they sum a function applied to each element of a set. Therefore they are easily parallelizable. One concern is that, especially if $\mathcal{O}$ is well outside $\mathcal{P}$, many significant digits may be lost from summing offsetting positive and negative terms. Therefore, double precision, at least, is recommended.

Determining the explicit edges and faces from any of these representations is similar to determining those properties of a polytope represented as a CSG tree.

Extensions to higher E^d, and also extensions to curved facets appear feasible. For example, the key to the cusp representation is that either vertex and tangent of an edge uniquely determine the infinite line including the edge. There is a unique point on the line, the foot of a perpendicular dropped from $\mathcal{O}$, which is used to partition the edge into two pieces, and hence to partition $\mathcal{P}$ into simplices. Any point that is a property only of the infinite line would serve as well as the foot of the perpendicular. For curved lines, it suffices to define a mapping from each line to a unique point on the line, and the rest of the theory follows. These minimal representations have many advantages.

Simplicity: A set of tuples is a much leaner representation than representing a hierarchy of nested face shells and edge loops.

Ease of computing the data: If $\mathcal{P}$ is being computed as the output of a Boolean operation, determining these local data structures is easier than determining $\mathcal{P}$'s global information. We save perhaps half the lines of code, and time, and reduce the number of special cases.

Fewer special cases: Multiple disconnected or nested components are automatically handled without ever recognizing their existence. Therefore, there is also a smaller opportunity for erroneously handling the special cases, increasing reliability.

Faster to code: This means that fewer lines of code are necessary, reducing coding time, and allowing more time for debugging and testing. Also, the data structures are fixed size and don't use pointers. This avoids slow memory management operations, which also fragment memory.

Faster to execute, especially on parallel machines: The simplicity and regular data structures allow faster execution on both serial and parallel machines. Virtual memory is not thrashed. Current data structures are too complex for easy parallelization, which prefers simple regular data structures, such as structures of arrays of plain old datatypes. If the platform is an Nvidia GPU, then warps of 32 threads are required to execute the same instruction (or be idle). Ideally, the data used by adjacent threads is adjacent in memory. That disparages pointers, linked lists, and trees.

Efficient operations when intersecting planar graphs: Instead of processing each face separately, the fact that two adjacent faces share an edge is utilized.

References

[1] Samuel Audet, Cecilia Albertsson, Masana Murase, and Akihiro Asahara. Robust and efficient polygon overlay on parallel stream processors. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, pages 304–313, New York, NY, USA, 2013. ACM.

[2] Salles V. G. de Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and Wenli Li. Fast exact parallel map overlay using a two-level uniform grid. In *4th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BigSpatial)*, Bellevue WA USA, 3 Nov 2015.

[3] Salles V. G. de Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and Wenli Li. PinMesh – Fast and exact 3D point location queries using a uniform grid. *Computer & Graphics Journal, special issue on Shape Modeling International 2016*, 58:1–11, August 2016. (online 17 May). Awarded a reproducibility stamp, `http://www.reproducibilitystamp.com/`.

[4] Salles V. G. de Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, Wenli Li, and Maurício Gouvêa Gruppi. Exact intersection of 3D geometric models. In *Geoinfo 2016, XVII Brazilian Symposium on GeoInformatics*, Campos do Jordão, SP, Brazil, November 2016.

[5] Salles V. G. de Magalhães and W. Randolph Franklin. Exactly computing map overlays using rational numbers. In *Autocarto 2014*, Pittsburgh PA, 5–7 Oct 2014. Cartography and Geographic Information Society. (abstract only).

[6] D. P. Dobkin, Leonidas J. Guibas, J. Hershberger, and J. Snoeyink. An efficient algorithm for finding the CSG representation of a simple polygon. *Comput. Graph.*, 22(4):31–40, 1988. Proc. SIGGRAPH '88.

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

[8] W. Randolph Franklin and Salles V. G. de Magalhães. Computing intersection areas of overlaid 2d meshes. In *IGS2019 International Geometry Summit Posters' proceedings*, Vancouver, Canada, 17–21 June 2019.

[9] W. Randolph Franklin, Salles V. G. de Magalhães, and Marcus V. A. Andrade. 3D-EPUG-Overlay: Intersecting very large 3D triangulations in parallel. In *2017 SIAM conference on industrial and applied geometry*, Pittsburgh PA USA, 10–12 July 2017. (talk).

[10] W. Randolph Franklin, Salles V. G. de Magalhães, and Marcus V. A. Andrade. An exact and efficient 3D mesh intersection algorithm using only orientation predicates. In *S3PM-2017: International Convention on Shape, Solid, Structure, & Physical Modeling, Shape Modeling International (SMI-2017) Symposium*, Berkeley, California, USA, 19–23 June 2017. (poster).

[11] W. Randolph Franklin, Salles V. G. de Magalhães, and Marcus V. A. Andrade. Exact fast parallel intersection of large 3-D triangular meshes. In *27th International Meshing Roundtable*, Alberqueque, New Mexico, 2 Oct 2018.

[12] Wm Randolph Franklin. Rays — new representation for polygons and polyhedra. *Computer Graphics and Image Processing*, 22:327–338, 1983.

[13] Wm Randolph Franklin. Polygon properties calculated from the vertex neighborhoods. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 110–118, 1987.

[14] Wm Randolph Franklin. Calculating map overlay polygon' areas without explicitly calculating the polygons — implementation. In *4th International Symposium on Spatial Data Handling*, pages 151–160, Zürich, 23-27 July 1990.

[15] Wm Randolph Franklin. Map overlay area animation and parallel simulation. In David H. Douglas, editor, *Proceedings, SORSA'92 Symposium and Workshop*, pages 200–203, July 28–August 2 1992.

[16] Wm. Randolph Franklin. Mass properties of the union of millions of identical cubes. In Ravi Janardan, Debashish Dutta, and Michiel 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.

[17] Wm Randolph Franklin, Narayanaswami Chandrasekhar, Mohan Kankanhalli, Varol Akman, and Peter YF Wu. Efficient geometric operations for CAD. In Michael J. Wozny, Joshua U. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 485–498. Elsevier Science Publishers B.V. (North-Holland), 1990. Selected and expanded papers from the IFIP WG 5.2/NSF Working Conference on Geometric Modeling, Rensselaerville, USA, 18-22 September 1988.

[18] Wm Randolph Franklin and Mohan S. Kankanhalli. Volumes from overlaying 3-D triangulations in parallel. In D. Abel and B.C. 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.

[19] Wm Randolph Franklin and Venkatesh Sivaswami. OVERPROP — calculating areas of map overlay polygons without calculating the overlay. In *Second National Conference on Geographic Information Systems*, pages 1646–1654, Ottawa, 5-8 March 1990.

[20] Wm Randolph Franklin, Venkateshkumar Sivaswami, David Sun, Mohan Kankanhalli, and Chandrasekhar Narayanaswami. Calculating the area of overlaid polygons without constructing the overlay. *Cartography and Geographic Information Systems*, pages 81–89, April 1994.

[21] Wm Randolph Franklin and Peter YF Wu. A polygon overlay system in prolog. In *Autocarto 8: Proceedings of the Eighth International Symposium on Computer-Assisted Cartography*, pages 97–106, Baltimore, Maryland, 29 March – 3 April 1987.

[22] M. C. Jordan. *Cours d'analyse de l'École Polytechnic, Tome Troisième, Calcul Intégral, équations différentielles*. Gauthier-Villars, Paris, 1887. `https://www.maths.ed.ac.uk/~v1ranick/jordan/jordan.pdf`, last accessed on 2019-11-01.

[23] Nvidia. NVIDIA Tesla V100 GPU architecture, the world's most advanced data center gpu, wp-08608-001v1.1, 2017. `https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`, last accessed on 2019-11-01.

[24] Oak Ridge National Lab. *Summit – Oak Ridge Leadership Computing Facility*. 2019. `https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/`, last accessed on 2019-11-01.

[25] M. Shimrat. Algorithm 112: Position of point relative to polygon. *Commun. ACM*, 5(8):434–, August 1962.

[26] Top500. Top 500 supercomputer sites. `http://www.top500.org`, 2019.

[27] Peter Y.F. Wu and W. Randolph Franklin. A logic programming approach to cartographic map overlay. *Journal of Computational Intelligence*, 6(2):61–70, May 1990.