Efficient algorithms for polygonal aggregation based on grid tessellation

Simone Santini[1] Amarnath Gupta[2] Yujun Wang[2]

[1] Escuela Politécnica Superior, Universidad Autónoma de Madrid
[2] San Diego Supercomputer Center, University of California, San Diego

Abstract. In this paper, we examine the problem of efficiently computing aggregate functions over polygonal regions of space. We first formalize a class of efficient region-based aggregation model, where the aggregation query is computed by representing the query region with pre-defined regions using set operations. By focusing on a grid tessellation, we first generalize the aggregation problem from the case of query regions that are isothetic rectangles to polygons with isothetic edges, and show that the aggregation query can be answered linear in the number of vertices of the polygonal region. It is efficient since it is independent of the size of query region or the number of objects intersecting with the database. We further show how to produce approximate aggregations for query regions having the shape of arbitrary polygons, and support optimal block reads from the disk.

1 Introduction

It is becoming a relatively common occurrence in data base problems that the data that one has to access can be considered elements of some metric space, and that the structure of the space enters prominently into query processing considerations. In this paper we are interested in a specific area of query processing for such types of data, namely *spatial aggregation*. A typical example is constituted by two-dimensional, possibly dense, distributions of values over rectangular domains, which we call *images*. (Actual images are, of course, a sub-class of our definition. Entities that fit our definition are, e.g., records of the geographic distribution of certain quantities.) It will be convenient to model these data distribution as functions from a given (rectangular, n-dimensional) domain to the set of all possible values of point-wise *observations*, so the data that we are interested in are functions $g : R \to \nu$, where $R \subseteq \mathbb{R}$ is a finite rectangle isothetic to the axes of $\mathbb{R}^n$, and ν is the observations data type. A typical query condition on a data base of such objects would be formulated in terms of aggregated values over sub-regions of R, that is, using a polygon P, and assuming ν ordered and additive, we would ask for conditions of the type $\sum_{i \in P} g(i) > a$.

The amounts of data involved are sizable: an image, e.g. reporting the localization of certain proteins in a sizable area of the brain can have a size of several GB; the size of the data collected in other disciplines, from meteorology to oceanography, can easily be of the same order of magnitude.

In this paper we study polygonal aggregation queries that can be efficiently performed on point-indexed data overlaid on a uniform grid tessellation. The efficiency of

the algorithms presented here allows this tessellation to be very fine, to include thousands or tens of thousands of cells per image. A naïve scheme for evaluating the example query could be as follows.

i) store the value of the obwerved quantity for all the cells in a 2D array;
ii) determine the cells that intersect the query polygon;
iii) compute the aggregate (i.e., the sum) for all the cells the pass the test ii).

The complexity of this solution is in the intersection, plus the cost of accessing the qualifying cells, which increases with the area of the query polygon. Solutions based on spatial indices suffer, to a lesser extent, of the same problem [4]. In this paper we show that with suitable pre-computation, the linear dependency on the area of the query polygon can be eliminated, making the time complexity of the aggregation computation a function of the complexity of the *boundary* of the query region.

We have chosen not to deal with any specific aggregation function, but to determine the minimal properties that the aggregation mechanism must satisfy for our algorithm to be applicable. We consider this the only formally correct methodology, and we strongly advocate it for future work on the subject.

There is a significant body of research related to this problem, mainly in the areas of computational geometry and on-line analytical processing.

Agrawal and Erickson [1] survey a number of related problems that have been classified under the term *range searching* in computational geometry. Ho *et. al.*, [6], introduced *range-sum* queries in data cubes, and considered the aggregate operations SUM and MAX computed over rectangular regions. For computing sums, the efficiency of computation is achieved through the use of the so-called *prefix sums*. The technique has been further studied and applied in [5, 2, 7]. However, the prefix sum algorithm is restricted to query regions that are rectangles isothetic to the axes, and punctiform data. A variant of the same approach is the *box sum* query ([12]), in which the data as well are isothetic rectangles.

In this paper, we study the efficient aggregation over arbitrary isothetic polygonal regions. By using a form of prefix sums, we generalize the range sum aggregation [6] from rectangles to isothetic polygons, and show that the aggregation can be computed in a time linear in the number of vertices of the query polygon. This method yields constant aggregation time for range sum over rectangles, for which the number of vertices is constant and equal to four.

2 A General Polygonal Aggregation Model

In this section we introduce our abstract model of geometric aggregation. We are interested in (possibly dense) sets of points in a domain that, for the time being, we will assume to be $\mathbb{R}^2$. A *region* in $\mathbb{R}^2$ is a compact, closed subset of $\mathbb{R}^2$ and a database D of $\mathbb{R}^2$ is a countable set of regions of $\mathbb{R}^2$. We will be concerned with a rather restricted class of functions on these regions, a class that we shall call *(geometric) aggregation functions*.

In general, two functions enter into play in aggregation over a region: a *weight function* $g : \mathbb{R}^d \to \tau$ defined over the region and an aggregation functional f such that, for the region $r_i \subseteq dom(g)$, $f(r_i)(g)$ is the aggregation of g over the region r_i.

Definition 1. *Given a database D of regions in $\mathbb{R}^d$, a* (geometric) aggregation structure *on D is a pair $(f, +)$ where $f : 2^{\mathbb{R}^d} \to (\mathbb{R}^d \to \tau) \to \tau$, and τ has a Abelian group structure, $+$ being its group function (that we will often write in the guise of a binary operator). As customary for Abelian groups, we will write $-b$ in lieu of b^{-1} and use the shorthand $a - b$ for $a + (-b)$.*

Example: Consider a region in which a *weight function* $g : \mathbb{R}^d \to \mathbb{R}$ is defined. Then the pair $(f, +)$ where $f(r)(g) = \int_r g$ and $+$ is the sum of real numbers is a geometric aggregation structure.

Note that the operation $+$, defined on the group τ can be extended to functions $g, h : A \to \tau$ in a point-wise manner: $(g + h)(x) = g(x) + h(x)$. Similarly, for the function f, if $X, Y \subseteq \mathbb{R}^d$, then $(f(X) + f(Y))(g) = f(X)(g) + f(Y)(g)$. (e.g. $(\int_X +]int_Y)(g) = \int_X g + \int_Y g$.)

Consider now a database $D = \{t_1, \ldots, t_n\}$ in which each region t_i has associated a point function $g_{t_i} : t_i \to \tau$.

Definition 2. *Given a database D, a function $\mathbf{\Sigma} : \mathbb{R}^{2^d} \to \mathbb{R}$ is a* (region-based) aggregation function *for D if there exists a geometric aggregation structure $(f, +)$ such that:*

i) $\forall q \subseteq \mathbb{R}^d, \mathbf{\Sigma}(q) = \sum_{t \in D} f(q \cap t)(g_t)$ *(q, t compact, closed);*
ii) $\forall t_1, t_2 \in \mathbb{R}^d, t_1 \cap t_2 = \emptyset \Rightarrow \mathbf{\Sigma}(t_1 \cup t_2) = \mathbf{\Sigma}(t_1) + \mathbf{\Sigma}(t_2)$.

A few observations should be made on this formula. Consider a database $D = \{t_1, t_2\}$ such that $\bar{t} = t_1 \cap t_2 \neq \emptyset$ and a region q such that $t_1 \cup t_2 \subseteq q$. Then

$$\mathbf{\Sigma}(q) = \sum_i f(q \cap t_i)(g_i) = f(t_1)(g_1) + f(t_2)(g_2) \tag{1}$$

Note that, from the aggregation point of view, the region $\bar{t}$ is counted twice, once in $f(t_1)(g_1)$ and the other in $f(t_2)(g_2)$. This generates the wrong semantics for certain aggregations. For example, if $\mathbf{\Sigma}(t) = |t|$ then aggregating over f would not be the correct way of computing the area of a composite region.

This assumption produces correct results when the measurements represented by the regions are logically independent. In most applications, f is an operator that acts on a function defined on the objects of the database. If the point-wise functions represent independent measurements then the semantics that we are using is sensible.

Example: In the database $D = \{t_1, t_2\}$, let there be functions $g_i : t_i \to \mathbb{R}$ associated to each region t_i, and let $f(t_i)(g_i) = \int_{t_i} g_i$. If t_1, t_2 are geographic areas and g_1 measures, say the local production of wheat while g_2 measures the local production of oat then the aggregate of g_1 and g_2 measures the whole production of grain in the area "q." In this case, it is correct to count the region $t_1 \cap t_2$ twice: once for wheat and once for oat.

This assumption is justified on pragmatic grounds, and it has no practical consequences: our technique will apply exactly to isothetic polygons (and approximately to

arbitrary polygons), a circumstance that eliminates the need to overlap simple regions in order to create a complex one, and from which most of the unwanted intersections come.

Proposition 1. *Let $\mathbf{\Sigma}$ be an aggregation function built on the aggregation $(f, +)$. Then, for all regions $t, t' \subseteq \mathbb{R}^d$ the following three equalities hold:*

i) $\mathbf{\Sigma}(t \cup t') = \mathbf{\Sigma}(t) + \mathbf{\Sigma}(t') - \mathbf{\Sigma}(t \cap t')$*;*
ii) $\mathbf{\Sigma}(t \cap t') = \mathbf{\Sigma}(t) + \mathbf{\Sigma}(t') - \mathbf{\Sigma}(t \cup t')$*;*
iii) $\mathbf{\Sigma}(t - t') = \mathbf{\Sigma}(t) - \mathbf{\Sigma}(t \cap t')$.

The technique that we are presenting here, much like that in [6] is based on the idea of pre-computing the values of $\mathbf{\Sigma}$ for a relatively low number of regions in such a way that the values $\mathbf{\Sigma}(q)$ can be computed for a certain set of *query* objects Q by combining the pre-computed values using the equalities i)-iii) of Proposition 1 (in addition to the group properties of +).

We begin with a controlled way of generating the regions over which we want to compute the aggregation. The regions are computed starting with a *base* (a set of elementary regions), and combining them with formulas of limited complexity in the set algebra (R, Ω) where R is a set of compact, closed region over $\mathbb{R}^d$ and $\Omega = \{\cap, \cup, -\}$.

A formula on this algebra using the variables of a set of names X is an element of the set $\Omega(X)$ defined as the smallest set such that $X \subseteq \Omega(X)$ and for all $e, e' \in \Omega(X)$ and $\otimes \in \{\cup\cap, -\}$ we have $e \otimes e' \in \Omega(X)$.

Given a formula $e(x_1, \cdots, x_n) \in \Omega(X)$, its length $|e|$ is the number of occurrences in it of the symbols of Ω. It is easy to see that if $|e| = k$, then e contains at most $2k$ variables.

The general idea of our method is to translate limited length formulas in the set algebra (that is, formulas that can be used to compose "query" regions) into limited length formulas in the algebra $(\text{dom}(\tau), \{+, -\})$ in such a way that all the values $\mathbf{\Sigma}(t)$ that appear in the target formula belong to a finite set of pre-computed values.

Due to the equality $a \cap b = a \cup b - (a - b) - (b - a)$ every formula in the set algebra can be expressed using the reduced set of operators $\Omega = \{\cup, -\}$. From now on, any query region will be intended to be composed using these two operators only, although, for the sake of simplicity, we will keep using the $\cap$ operator for general formulas.

Theorem 1. *Let r be a region expressed as $q = e(a_1, \cdots, a_n)$, where $e \in \Omega(a_1, \cdots, a_n)$ then there are integers ξ_i, η_i such that $\mathbf{\Sigma}(q) = \sum_i \xi_i \mathbf{\Sigma}(a_i) + \sum_i \eta_i \mathbf{\Sigma}(\cap_{j=1}^{k_i} a_{i_j})$.*

The proof is based on a repeated application of Proposition 1, throught which we can eliminate all the set operation $\cup$ and $-$ from $\mathbf{\Sigma}(q)$.

Lemma 1. *Let $Q = \{q_1, \cdots, q_n\}$ be a set of regions, m an integer, and let*

$$Q' = \{\bigcap_{i=1}^{k} q_{j_i} \mid q_{j_i} \in Q \wedge k \leq m\}$$
$$\Phi = \{\mathbf{\Sigma}(a) \mid a \in Q \cup Q'\}.$$

Then for every region q in the set $F = \{q \mid q = e, e \in \Omega(Q)$, and $|e| < m\}$ there is a formula $e' \in (\Phi, \{+, -\})$ such that $\mathbf{\Sigma}(q) = e'$, and $|e'| < 2m$

Proof. Since every e is in conjunctive normal form, so we only need to apply properties i) and iii) of Proposition 1. We introduce two operations ($+$ and $-$) in e' by applying i) and one operator ($-$) by applying iii). Since $|e| < m$, at most $2m$ operators will be introduced. So $|e'| < 2m$.

As it is, this result is not very useful because the set Q' grows like n^m and the number of pre-computed values that it is necessary to consider becomes soon excessive. There are cases, however, in which many of the intersections that appear in Q' are empty, so that the size of Q' is much smaller than the theoretical maximum.

In the following theorem, we will consider one of these cases, of great interest for applications to geometric data.

Theorem 2. *Let $Q = \{d_1, \cdots, d_m, c_1, \cdots, c_m\}$ be a set of regions such that for all $i \neq j$, $d_i \cap d_j = \emptyset$, $c_i \cap c_j = \emptyset$, and $\cup_{i=1}^m c_i \subseteq \cup_{i=1}^m d_i$. Let $r = \cup_{i=1}^m d_i - \cup_{i=1}^m c_i$, then $\alpha(r) = \sum_{i=1}^m \alpha(d_i) - \sum_{i=1}^m \alpha(c_i)$.*

Proof.

$$\begin{aligned}
\mathbf{\Sigma}(r) &= \mathbf{\Sigma}(\cup_{i=1}^m d_i - \cup_{i=1}^m c_i) \\
&= \mathbf{\Sigma}(\cup_{i=1}^m d_i) - \mathbf{\Sigma}(\cup_{i=1}^m d_i \cap \cup_{i=1}^m c_i) && \text{(by iii) of Prop.1)} \\
&= \mathbf{\Sigma}(\cup_{i=1}^m d_i) - \mathbf{\Sigma}(\cup_{i=1}^m c_i) && \text{(because } \cup_{i=1}^m c_i \subseteq \cup_{i=1}^m d_i\text{)} \\
&= \textstyle\sum_{i=1}^m \mathbf{\Sigma}(d_i) - \sum_{i=1}^m \mathbf{\Sigma}(c_i) && \text{(from } d_i \cap d_j = \emptyset, c_i \cap c_j = \emptyset, i \neq j\text{)}
\end{aligned} \tag{2}$$

Note that this is essentially equivalent to the range sum proof for the two-dimensional case.

In our setting, we consider only grid tessellations [8], called simply *tessellations* when this generates no ambiguity. The pre-computation over a tessellation $\mathbf{T}$ is usually stored a two dimensional array $\mathtt{Ts}[n_x, n_y]$, each element containing the aggregate value over a cell of the tessellation. As in [6], we use a *prefix array* to contain pre-computed values over $\mathtt{Ts}$. This is also a two dimensional array, $F[n_x, n_y]$, with $F[i,j] = \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} \mathtt{Ts}[i,j]$, where $\sum$ is the sum operator of definition 1.

Every point (x, y) in the domain of interest belongs to a unique cell of the tessellation, whose aggregation value is contained in an element $\mathtt{Ts}[i,j]$ of the tessellation array. If $\Delta x, \Delta y$ are the sizes of the cell along the two axes, and x_T, y_T the coordinates of the point of $\mathtt{Ts}[0,0]$ with minimal coordinates, then $i = \lfloor (x - x_T)/\Delta x \rfloor$, and $j = \lfloor (y - y_T)/\Delta y \rfloor$. We assume that an origin x_0, y_0 is given outside of the region of interest, with $x_o < x_T$ and $y_0 < y_T$.

3 Polygonal aggregation over the prefix array

A *isothetic polygon* P over a tessellation $\mathbf{T}$ is a non self-intersecting polygon whose vertices are vertices of the cells of $\mathbf{T}$. It follows from this that all the edges of P are either edges of $\mathbf{T}$ or unions of contiguous edges. A isothetic polygon with k sides can be represented by the list $P = [p_0, \cdots, p_{k-1}]$, of its vertices p_i, $0 \leq i \leq k-1$, its edges being the segments $(p_i, p_{(i+1) \bmod k})$, $0 \leq i < k-1$ (from now on we will write

simply (p_i, p_{i+1}), the modulo k being implicit). We assume that the representation is minimal, viz. that no three consecutive vertices in the sequence are aligned. An edge of a isothetic polygon is *vertical* (resp. *horizontal*) if its two vertices have the same x (resp. y) coordinate.

Since we assume that the origin O has smaller coordinate values that any of the vertices in the tessellation **T**, and every edge of a isothetic polygon is formed by edges of **T**, O is never collinear to any edge of the polygon: any line from O intersects an edge of P at most once, and the lines that intersect an edge are neither horizontal nor vertical. Based on the Jordan curve theorem [3], we define the following notions. Given a isothetic polygon P, an edge e of P, and the origin O, we call e *entering* (resp. *leaving*) if a straight line segment from O through any point in e intersects P an odd (resp. even) number of times (without counting the intersection at P).

In order to determine whether a segment is entering or leaving there is a simple standard algorithm that, for a polygon with k segments, runs in time $O(k)$. Consider a point $P \equiv (x, y)$ and a segment s between two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$. The line containing s divides the plane in two parts that, taking as a reference the direction $p_1 \rightarrow p_2$, we will call the "left" and the "right" part of the plane. The position of p with respect to s can be determined by the sign of the expression

$$LR(p, s) = (x_2 - x_1)y + (y_2 - y_1)x + (x_1 y_2 - x_2 y_1) \tag{3}$$

so that

$LR(p, s) < 0 \Rightarrow p$ is on the left half-plane of s
$LR(p, s) = 0 \Rightarrow p$ is on the line through s
$LR(p, s) > 0 \Rightarrow p$ is on the right half-plane of s

The important observation here is the following: take two consecutive segments s_i and s_{i+1} on the contour. The status of the segment changes from entering to leaving or vice-versa if and only if the origin changes from one half-plane to another (i.e. if the origin is on the left of s_i but on the right of s_{i+1}; the case $LR(0, s) = 0$ is somewhat special). This gives us a way to trace *changes* in the orientation of each segment in constant time, that is, in $O(k)$ for all segments.

The following procedure takes one segment, and the values of LR and of the polarity for the previous one, and returns the orientation of the segment (+1 or -1) and its LR value:

```
polarity(lr, p, s)
    if LR(0, s) = 0 → return (lr, p);
    □  LR(0, s) · lr > 0 → return (LR(0, s), p);
    □  LR(0, s) · lr < 0 → return (LR(0, s), −p);
    fi end
```

To start the method, we observe that the segments that abut to the vertex with minimal X and Y are always entering.

If (p_i, p_{i+1}) is an edge of P, then the point p_i is labeled *positive* (resp. *negative*) if (p_i, p_{i+1}) is labeled *entering* and p_i is closer to (resp. farther from) the origin O that p_{i+1}, or if (p_i, p_j) is leaving and p_i is farther (resp. closer) to the origin O than p_{i+1}.

The following lemma shows that the property of being positive or negative can be assigned to vertices unambiguously, even without making reference to the edges to which they belong.

Lemma 2. *Let p_i be the vertex of a isothetic polygon P, belonging to the edges (p_{i-1}, p_i) and (p_i, p_{i+1}). Then p_i is positive (resp. negative) in (p_{i-1}, p_i) if and only if it is positive (resp. negative) in (p_i, p_{i+1}).*

Proof. (sketch) The two edges to which p belongs are either labeled *entering* or labeled *leaving*. The vertex in each edge might be closer or farther to the origin. This gives to eight possible topological relationships as shown in Figure 1. The proof is straightforward by checking each of them.

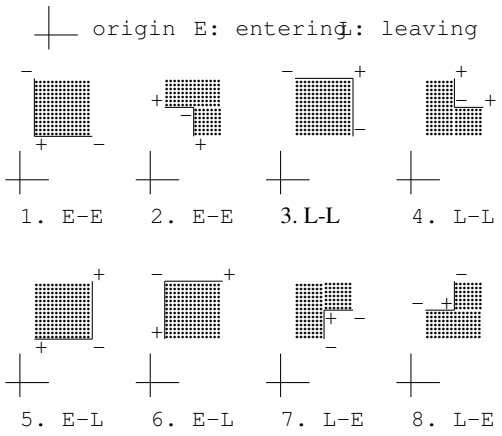

Fig. 1. Eight topological relationships formed by a vertex (positive or negative) and its edges (entering or leaving) of a isothetic polygon

The following lemma gives us a fast procedure for assigning a polarity to all the vertices of a straight polygon once we know the sign of one of them.

Lemma 3. *Let P be a isothetic polygon over tessellation **T**, $P = p_0, \cdots, p_{k-1}$. For every vertex $p_i, 0 \leq i \leq k-1$, if p_i is positive (resp. negative), then p_{i+1} (or p_0 if $i = k-1$) is negative (resp. positive).*

After we use Figure 1 to decide whether the first vertex is positive or negative, we can apply Lemma 3 to assign the polarity to all other vertices.

3.1 Computing Geometric Aggregation

With the edges and vertices duly labeled, we now proceed to the computation of the geometric aggregation. The first step in this is to introduce the concepts of *leaf edge* and *leaf rectangle*.

Let p_0, p_1, p_2, p_3 be four consecutive vertices of a isothetic polygon P. The edge $e = (p_1, p_2)$ is called a leaf edge if 1) edge (p_0, p_1) is equal to or shorter than edge (p_2, p_3); 2) the rectangle r formed by p_0, p_1, p_2 is contained in P, and r does not intersect any edge of P except at point p_0 and p_3. Figure 2 shows all the cases of leaf edges. In these cases, we call (p_0, p_1) and (p_2, p_3) the *side edges* of the *leaf edge* (p_1, p_2). For a leaf edge e, its leaf rectangle is the rectangle formed by p_0, p_1, p_2 with an open edge at p_0. For example in the case 1 of Figure 2, the leaf rectangle is formed by p'_0, p_1 and p_2, where $p'_0 = (p_0.x + 1, p_0.y)$.

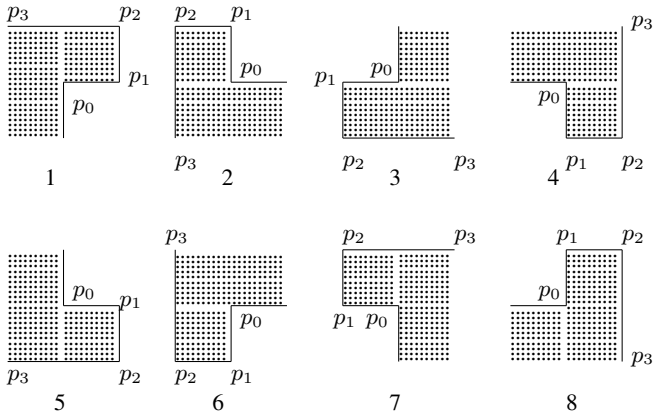

Fig. 2. Eight cases of leaf edge (p_1, p_2) of a isothetic polygon

Lemma 4. *Every isothetic polygon with more than four vertices has at least one leaf edge.*

Proof. Let P be a isothetic polygon with more than 4 vertices. No isothetic polygon with more than four edges is convex, so we can use the following procedure to split P into isothetic polygons with fewer vertices until all pieces are rectangles.

Consider a isothetic polygon P' with more than 4 vertices. Find a vertex p which has internal angles in P' greater than π, extend an edge that contains p inside P' until it reaches another edge of P'. Now P' can be split into two entities – a rectangle, and a isothetic polygon having fewer vertices than P'. The operation is repeated on the remaining straight polygon until all the pieces are rectangles. If we need m splits before all the pieces are rectangles, we have $m + 1$ rectangles. Every time we split a isothetic polygon, we add two "new" edges. An edge is new if it contains points which belongs the interior of the original isothetic polygon P. So we add at most $2m$ new edges. The total number of edges of $m + 1$ rectangles is $4m + 4$. So there must exist one rectangle that has three edges which are not new. These three edges must be consecutive in the rectangle, and the middle one satisfies our definition of a leaf edge.

The following theorem shows how to compute aggregation over a isothetic polygon using the prefix array F.

Theorem 3. *Let $P = p_0, \cdots, p_{k-1}$ be a isothetic polygon over tessellation **T**, F be the prefix array over the tessellation **T** and origin O. Then,*

$$\mathbf{\Sigma}(q) = \sum_{i=0}^{k} (-1)^{i_0+i} F[p_i.x + \delta_i^x, p_i.y + \delta_i^y] \tag{4}$$

where p_{i_0} is the vertex with minimal x and y coördinates, and δ_i^x (resp. δ_i^y) is -1 or 0 depending on whether the vertical (resp. horizontal) edge for p_i is entering or leaving.

Proof. First we show that $(-1)^{i_0+i}$ always correctly computes whether p_i is positive or negative. Because of Lemma 3, we only need to show that the factor is positive for p_{i_0}, which is the case because $(-1)^{i_0+i_0} = (-1)^{2i_0} = 1$.

Next we consider the simplest case of isothetic polygons, namely rectangles. Let $p_0.x = p_1.x$. From theorem 2,

$$\mathbf{\Sigma}(q) = F(p_0.x-1, p_0.y-1) - F(p_1.x-1, p_1.y) + F(p_2.x, p_2.y) - F(p_3.x, p_3.y-1), \tag{5}$$

which coincides with (4) for rectangles.

The case where $k > 4$ can be proved by induction over k. Suppose Equation 4 is true for any number of vertices less than k, we need to prove that any isothetic polygon P with k vertices ($k > 4$) can be reduced to two isothetic polygons P_1, P_2 with fewer vertices, and satisfy:

i) $\mathbf{\Sigma}(P) = \mathbf{\Sigma}(P_1) + \mathbf{\Sigma}(P_2)$;
ii) $P_1 \cup P_2 = P$, and $P_1 \cap P_2 = \emptyset$.

The idea is to find a leaf edge e, whose existence is guaranteed by Lemma 4. Then we decompose P into a isothetic polygon P_1 and a leaf rectangle P_2, where P_2 is formed by leaf edge e and the other shorter edge that connects to e.

There are eight cases for a leaf edge, we just show one case here. The proof for the others is similar. As shown in Figure 3, edge (p_{i+1}, p_{i+2}) is the leaf edge. We define three new points p_i', p and p', where $p_i'.x = p_i.x + 1$, $p_i'.y = p_i.y$, $p.x = p_i.x + 1$, $p.y = p_{i+2}.y$, $p'.x = p.x + 1$, $p'.y = p.y$.

Obviously $P_1 \cup P_2 = P$, $P_1 \cap P_2 = \emptyset$, P_1 is a isothetic polygon, and P_2 is a rectangle. According to Figure 1, we know p_i, p_{i+2} are positive in P, p is positive in P_1, p_{i+2}, p_i' are positive in P_2, p_{i+1} is negative in P, and p' and p_{i+1} are negative in P_2. It is easy to verify that

$$\begin{aligned} F(p_i'.x-1, p_i'.y-1) - F(p_{i+1}.x, p_{i+1}.y-1) + F(p_{i+2}.x, p_{i+2}.y) - F(p'.x-1, p'.y) + F(p.x, p.y) = \\ F(p_i.x, p_i.y-1) - F(p_{i+1}.x, p_{i+1}.y-1) + F(p_{i+2}.x, p_{i+2}.y) \end{aligned} \tag{6}$$

So we have $\mathbf{\Sigma}(P_1) + \mathbf{\Sigma}(P_2) = \mathbf{\Sigma}(P)$.

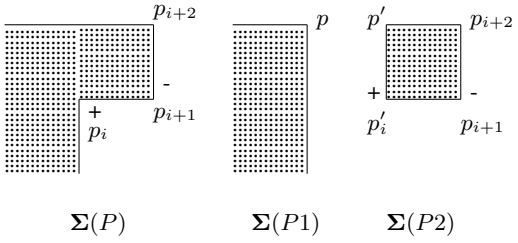

Fig. 3. Decomposition a isothetic polygon into a isothetic polygon and a rectangle for the first case in Figure 2

3.2 Aggregation Algorithm for Isothetic Polygons

The aggregation algorithm (Algorithm 3.2) is based on theorem 3. Here $P = \{\text{P[i]}\}, i = 1, \ldots, k$ is the list of points (indexed as an array) that define the query region, and $F = \{\text{F[i,j]}\}, i = 1, \ldots, n_x, j = 1, \ldots, n_y$ is the matrix with the prefixes. The function *min_coord* returns the index of the point with the minimal x coordinate and, if more than one such points exists, the point among them with the minimal y coordinate.

In the algorithm, line 1 determines the vertex with two entering edges, lines 12-16 compute whether an edge is entering or leaving, lines 18-21 and lines 22-25 compute δ_i^x and δ_i^y. The complexity of Algorithm 1 is given by the following theorem.

Theorem 4. *Algorithm 1 computes the aggregation for isothetic polygons correctly and its time complexity is $O(k)$, where k is the number of vertices in q. If the prefix array is stored on disk, Algorithm 1 needs at most k I/O reads.*

Proof. The correctness of Algorithm 1 is guaranteed by Theorem 3. The algorithm is independent of the size of tessellation **T**. Since F is only accessed k times in the algorithm, so if F is stored in the disk, Algorithm 1 needs at most k I/O reads.

4 Combining Prefix Sum Arrays with R-trees

So far, we have assumed that the prefix sum is stored in an array and that the aggregation algorithm would use all the elements of the array. This would be clearly wasteful if there are cells in the tessellation whose pre-aggregate value is the identity of the $+$ operation of the aggregation group. The aggregation would be more efficient if we could group clusters of non-zero cells into rectangles attached to prefix arrays. In this section, we outline a method that uses R-trees to speed up the computation of the prefix sum. What we are doing here is similar in spirit to [12], which keeps partial subtotals in the nodes of a BA-tree. The interested reader can find the details of the method in [11].

The idea here is to use several prefix arrays instead of one, and attach them to nodes of R-trees. However, the R-trees need to be constructed by first finding dense regions of the tessellation.

Given a tessellation **T**, let r be a rectangle whose vertices are from **T**. Let $\text{nz}(r)$ be the total number of cells in r with non-zero values, and $|r|$ the total number of cells in

```
Σ(P, F) →
      i_0 := min_coord(P);
α     δ_x := δ_y := −1
      sign := 1;
      prev := enter;
      i := i_0;
      sum := F[P[i].x + δ_x, P[i].y + δ_y];
      repeat
β       p := P[i];
        p_2 := P[i + 1 mod k ];
        p_3 := P[i − 1 mod k ];
        if LR(O, p_1, p_2) = LR(O, p_2, p_3) then
          current := prev;
        else
          if prev = enter then
            current := exit;
          else
            current := enter;
          fi
        fi
        sign := −sign;
        Δx := 0
        if (prev = enter and p_1.x = p_2.x) or (current = enter and p_3.x = p_2.x) then
          Δx := −1;
        fi
        Δy := 0
        if (prev = enter and p_1.y = p_2.y) or (current = enter and p_3.y = p_2.y) then
          Δy := −1;
        fi
        sum := sum + sign⋆F[p_2.x + Δx, p_2.y + Δy];
      until i = i_0;
      return sum;
```

Table 1. Algorithm to compute exact aggregations on isothetic polygons. Notes: in α, $P[i_0]$ has two entering edges; in β, p is the "previous" vertex, p_1 is the "current" vertex, and p_2 is the "next" vertex.

r. The *density* of r over **T** is defined as $\rho(r) = \text{nz}(r)/|r|$. Clearly $\rho(r)$ is the probability that a cell in r have non-zero value, and $0 \leq \rho(r) \leq 1$.

Our idea of space optimization is to find a set of disjoint rectangles that covers all the cells with non-zero values in **T**, then attach prefix arrays only to the rectangles with high density. The problem is how to increase the area of those rectangles while keeping the node density high.

We use the MX Quadtree to find rectangular regions with high density. The MX Quadtree recursively decomposes a raster into four blocks of equal area until the rectangles are decomposed into single cells. Empty cells are merged into larger cells. Four occupied cells are not merged. For details, see [10, 9].

To construct a MX Quadtree for a tessellation **T**, **T** is extended to a matrix with additional cells of zero value, so that the numbers of cells along X and Y are equal to n^4 for some n. Then we construct an MX Quadtree that contains all the cells with non-zero value using the standard insertion algorithm. For each node d in the tree, its *bounding box*, denoted by $\lfloor d \rfloor$, is the minimal rectangle containing all the cells in d with non-zero value. We denote the total number of cells in d with non-zero values as $\text{nz}(d) = \text{nz}(\lfloor d \rfloor)$, and the density of d as $\rho(d) = \rho(\langle d \rangle)$. All the leaf nodes represent cells in **T** and if d is a leaf node, then $\text{nz}(d) = 1$ and $\rho(d) = 1$.

Let $\mathbb{M}$ be a MX Quadtree for the tessellation **T**, let *desc* be the descendent relation on $\mathbb{M}$, and h a density threshold; an internal node d in $\mathbb{M}$ is said to be *selected by* h ($\text{sel}(h, d)$) if it satisfies the following two conditions:

i) $\rho(d) \geq h$ (the density of the node is no less than the density threshold);
ii) $\text{desc}(d', d) \Rightarrow \rho(d') < h$ (the density of any of the ancestor of d is less than the density threshold).

We have:

$$(d_1 \neq d_2 \wedge \text{sel}(h, d_1) \wedge \text{sel}(h, d_2)) \Rightarrow \langle d_1 \rangle \cap \langle d_2 \rangle = \emptyset \tag{7}$$

that is, if two distinct nodes are both selected by a same threshold, their bounding boxes are disjoint. This means that there will be no overlap if we build prefix arrays for them. Further, the node density of any ancestor node is lower than the given threshold, in a sense that the bounding box for any selected node has been maximally enlarged in $\mathbb{T}$.

We define the *total space* for prefix arrays over h, $S(h)$, as $\sum_{d:\text{sel}(h,d)} \lfloor d \rfloor$ and the *total number of data nodes* over h, $N(h)$, as the total number of nodes selected by h plus the number of cells which have non-zero values and are not descendent of any selected nodes.

Based on the MX Quadtree $\mathbb{M}$ and the density threshold h, we can construct a R-tree using two sets of data nodes. The first set contains only rectangles that are the minimal bounding boxes of all the selected internal nodes. The second set contains point data for all the remaining cells which are not descendent of any selected node. For each of the selected nodes, the prefix array is computed over its minimal bounding box.

For the R-tree thus constructed with hybrid data nodes over a given density threshold h, the total space used for prefix arrays is $S(h)$ and the size of the R-tree is $N(h)$.

In particular, if we choose the density threshold to be no greater than that of the root in the MX Quadtree, only the root will be selected, and the R-tree will be reduced to a single-node tree, which means that we will create a single prefix array for the whole tessellation.

With the increase of the density threshold, the selection of internal nodes will be moved from the root down the MX Quadtree tree. Since the minimal bounding box of any selected node is a region of any of its ancestor nodes, the total space used for prefix arrays will be decreased. Meanwhile the total number of data nodes is increasing. In selecting an appropriate threshold, we will get a R-tree which is a tradeoff between space and aggregation query time.

We define *the node density set* of $\mathbb{M}$, $\text{Den}(\mathbb{M})$, as the set of node density values of all the nodes in $\mathbb{M}$. Let $d_1, d_2 \in \text{Den}(\mathbb{M})$ be two consecutive density values, which

means there exists no value in Den($\mathbb{T}$) between d_1 and d_2. Then for any value $d_3, d_1 < d_3 < d_2$, d_1 and d_3 will select the same set of nodes. This is straightforward since there exists no node with density between d_1 and d_3.

So the node density set of $\mathbb{T}$ can be used as the list of candidate density thresholds. Besides, we can filter all those values which are less than the root density since they will produce the same single node R-tree.

To tradeoff between space for prefix arrays and the size of hybrid R-tree, we can define linear cost function such as $S(h) + k_1 N(h)$, where k_1 is a constant. For applications when aggregation query time is more critical, it may be more imperative to penalize the effect of having larger hybrid R-trees. In practice, we have found the cost function $S(h) + k_2 * N(h)^2$ to be adequate in achieving a balance between space and time.

Once the cost function is chosen, we can simply calculate the cost for every value in the node density set, identify the density threshold with the lowest cost, generate the prefix arrays, and construct the hybrid R-tree.

For a hybrid R-tree generated from tessellation **T**, set $\bar{h} = \text{minarg}_{h \in \text{Den}(\mathbb{M})} \text{cost}(h)$, then the total space for prefix arrays of **T** is $S(\bar{h}h$, and its total number of data nodes is $N(\bar{h})$

In [11] an algorithm is presented for the construction of the R-tree. We show there that the construction algorithm has time complexity $O(n^2 \log n)$ and space complexity $O(n)$. The complexity of the aggregate computation is that of the search in an R-tree which, although has a worst case performance of $O(n)$ (therefore it represents no improvement over the naïve algorithm, has an average case performance of $O(\log n)$.

5 Conclusions

In this paper, we present a generalization of the prefix array technique to deals with general isothetic polygons. For non-self intersecting polygons defined by the edges of a tessellation over which computed values of the aggregate are available, we show that the aggregation can be computed in a time linear in the number of vertices of the query polygon. This reduces to constant time for rectangles, in which the number of vertices becomes a constant four.

We further show how to apply the aggregation algorithm to polygons by combining it with concave polygon scan conversion and contour tracing algorithms. Optimal block I/O can be achieved based on a general assumption for storing the prefix arrays.

To avoid wasting space for subregions in the tessellation that have aggregation value zero, and expensive update operation to prefix arrays, we introduce a hybrid R-tree which combines the advantage of the efficiency of prefix arrays and flexibility of R-trees. Experiments have shown that the hybrid R-tree helps us achieve a good balance between the tree size and total space for prefix arrays.

References

1. P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In *B.Chazelle, J.E. Goodman, and R.Pollack, editors. Advances in Discrete and Computational Geometry*, pages 1–56. Contemporary Mathematics, 1999.

2. R. Beigel and E. Tanin. The geometry of browsing. In *Proceedings of the Latin American Symposium on Theoretical Informatics*, 1998.
3. R. Courant and H. Robbins. *What is Mathematics?* Oxford University Press, 1941.
4. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
5. S. Govindarajan, P. K. Agarwal, and L. Arge. CRB-tree: An efficient indexing scheme for range aggregate queries. In *Proceedings of International Conference on Database Theory*, pages 143–157, 2003.
6. C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in olap data cubes. In *Proceedings of SIGMOD*, pages 73–88, 1997.
7. J. Jin, N. An, and A. Sivasubramaniam. Analyzing range queries on spatial data. In *Proceedings of ICDE*, 2000.
8. A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams (Second Edition)*. John Wiley & Sons, Chichester, England, 1999.
9. H. Samet. *Applications of spatial data structures: Computer graphics, image processing, and GIS*. Addison-Wesley, Reading, MA, 1990.
10. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.
11. S. Santini, A. Gupta, and Y. Wang A fast algorithm for geometric aggregation *Cahiers d'informatique*, N. 11/2011, Escuela Politécnica Superior, Universidad Autónoma de Madrid.
12. D. Zhang, V. J. Tsotras, and D. Gunopulos. Efficient aggregation over objects with extent. In *Proceedings of PODS*, pages 121–132, 2002.