HAL
open science

# Stripped halfedge data structure for parallel computation of arrangements of segments

Guillaume Damiand, David Coeurjolly, Pierre Bourquat

## To cite this version:

Guillaume Damiand, David Coeurjolly, Pierre Bourquat. Stripped halfedge data structure for parallel computation of arrangements of segments. The Visual Computer, 2021, 37 (9), pp.2461-2472. 10.1007/s00371-021-02185-4 . hal-03268260

**HAL Id: hal-03268260**

**https://hal.science/hal-03268260v1**

Submitted on 23 Jun 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

**CGI2021 manuscript No.**
(will be inserted by the editor)

# Stripped halfedge data structure for parallel computation of arrangements of segments

**Guillaume Damiand · David Coeurjolly · Pierre Bourquat**

**Abstract** Computing an arrangement of segments with some geometrical and topological guarantees is a critical step in many geometry processing applications. In this paper, we propose a method to efficiently compute arrangements of segments using a strip based data structure. Thanks to this new data structure, the arrangement computation algorithm can easily be parallelized as the per strip computations are independent. Another interest of our approach is that we can propose an out-of-core and streamed construction for large datasets, while keeping a low memory footprint. We prove the correctness of our structure and provide a complete comparative evaluation with respect to state-of-the-art demonstrating the interest of our construction for the computation of an exact arrangement.

**Keywords** Arrangement of segments · Parallel algorithm · Out-of-core construction · Halfedge data structure

## 1 Introduction

Given a set of segments in the plane, the arrangement of these segments is a combinatorial structure that describes all vertices, edges and faces induced by the segments. It thus encodes all the topological information defined by the segments while providing operators to navigate through the planar partition. Computing such planar data structure is a classical problem in computational geometry with applications in many areas such as computer graphics (*e.g.* Boolean set operations between two objects in the plane, object clipping, shape offset, swept volumes...), robotics (*e.g.* the translational motion planning of a robot in a room cluttered with obstacles, fabrication (*e.g.* slicing 3d models for additive layering), or Geographic Information Systems (*e.g.* planar map representation of countries). The computation of combinatorial arrangements of geometrical objects have been intensively studied for several decades with books [21,1,18] or book chapters entirely dedicated to arrangements and their applications [17,11,10,8].

In this work, we focus on the specific case of the computation of the arrangement of straight line segments in the plane. Computing such an arrangement requires to be able to efficiently compute pairwise intersections between such segments, to numerically represent such intersections with exact arithmetic, and to build a combinatorial data structure to encode the adjacency relationships and topological information of the planar structure. Challenges in this context consist in certifying the computation (both geometrically and topologically), and to be able to handle large datasets that is required in Geographical Information Systems (GIS) applications for instance.

Related Works.

To compute an arrangement, we are facing an algorithmic problem to efficiently perform intersections of pairs of segments, an arithmetic issue to reliably decide if two segments intersect and compute the intersections, and finally we need a planar map data structure to represent the topology of the arrangement.

**Line segments arrangements.** First we need to efficiently compute intersections between segments. A naive algorithm would consider each pair of segments intersects and check their intersection. This would lead to

Université de Lyon, UCBL, INSALyon, CNRS, LIRIS, UMR5205, F-69622, Lyon, France
Corresponding author: Guillaume Damiand E-mail: guillaume.damiand@liris.cnrs.fr

a quadratic complexity that does not scale up for large datasets. Several improvements have been proposed. The method given in [7] is often considered as the state-of-the art with a computational cost in $O((n+k)\log n)$, where $n$ is the number of segments, and $k$ is the total number of intersection points. The main idea of [7] is to sweep a line through the segments, from left to right, keeping updated a list of active segments (*i.e.* segments that intersect the line). Thanks to this principle, intersections are only tested between consecutive segments in the sweep line. One drawback of this method is to require to process all segments in a total order, preventing from computing the arrangement in parallel. Some other works have better theoretical complexity [14,5] but the algorithms are more complex to implement.

**Parallel segment intersections.** Using the above-mentioned plane sweep approach on a strip-based decomposition of the domain, [27] and [26] proposed a parallel method to compute all intersections between a set of segments. In our context, these techniques suffer from critical drawbacks. First, the output is not the complete line segment arrangements, but just the segment-segment intersections. Furthermore, the geometry computations are achieved with floating point arithmetic, leading to numerical errors and inconsistency in the reported intersections. Lastly, strip positions cannot contain a vertex of the planar arrangement.

**Parallel arrangements.** To achieve best performances on multicore systems, or to handle large datasets in an *out-of-core* setting, parallel design of arrangement algorithms has been a challenging task in computational geometry [2]. More precisely, plane sweeping techniques [4,20], or arrangement computation algorithms [19,3] have gained some theoretical attention on very specific parallel models (*e.g.* massively parallel schemes). However, these theoretical algorithms do not apply for more real case scenarios, or on recent multicore models.

**Robust computations.** To decide whether two segments intersect or not, we need a robust predicate in order to avoid numerical errors coming from floating point calculations that would imply inconsistent information between the topological representation and the geometrical one (leading to errors, infinite loops or crashes when implementing geometrical algorithms) [24]. One solution is to use robust predicates (for example [31]). But a question remains: how to compute and store the intersection points in order to represent the arrangement? One solution would be to use a snap rounding technique [23] to produce a valid planar map partition with floating point coordinates. But the obtained representation is only a simplified representation of the arrangement at a certain level of precision, and does not correspond to the exact result. To represent the exact result of the arrangement without numerical errors, we need an exact representation of the geometrical objects.

**Planar map representation.** Beside resolving numerical intersection issues between segments, we need a data structure to describe the planar partition itself. This data structure should allow insertions of new segments, and modifications of existing segments when they are split. In the literature, many data structures have been proposed to represent such planar partitions: Winged edges [6], Halfedge data structure [34], Combinatorial Maps [25], Corner Table [30], Doubly Connected Edge List [28], Surface Mesh [32]. . . . The different solutions are close and vary in their storage cost, in the type of operators that they support, and in the type of objects they can describe (please refer to [12] and [15] for a more complete comparison).

Contributions.

In this article, we propose (1) the definition of a stripped halfedge data structure ensuring a global topological consistency of the overeall planar map; (2) a fast parallel arrangement construction based on this stripped representation, restricting the global arrangement computation to local ones, which can be performed using [7] for instance; (3) an out-of-core streamed construction that allows constructing an arrangement of a huge number of segments with a low memory footprint. Our approach is exact and outputs a valid planar map encoding whatever the input.

Our method answers the limitations of the related works: (1) we define a parallel algorithm that can be implemented on a multi-core model; (2) we represent the full topology of the arrangement through the stripped halfedge data structure; (3) our approach is robust, based on exact arithmetic construction and exact geometrical predicates provided by CGAL [33], a computational geometry algorithm library; (4) we have no constraint on the location of the strips.

## 2 Preliminaries

A *planar partition* is a subdivision of a 2D domain into open topological *cells*: *vertices* (0D cells), *edges* (1D cells) and *faces* (2D cells). Cells are equipped with some neighborhood relationships. Two cells are *incident* if one belongs to the boundary of the other. Two cells are adjacent if they have the same dimension and if they share a common cell incident to both. An oriented edge can be denoted by its two vertices $[AB]$, $A$ being the source endpoint of the edge and $B$ its target endpoint.

In this work, we use the well-known *halfedge data structure*, denoted *HDS* (see [34] for all precise definitions). Each oriented edge $[AB]$ of the planar partition is represented by two halfedges in the HDS, linked by an `opposite` relation, thus for each halfedge $h$, $\texttt{opposite}(h)$ gives the other halfedge describing the same edge. Four other relations exist between halfedges: $\texttt{next}(h)$ is the next halfedge around the face of $h$ (in clockwise order), $\texttt{prev}(h)$ is the previous halfedge around the face of $h$, $\texttt{next}_\text{v}(h)$ is the next halfedge around the vertex of $h$ (in clockwise order) and $\texttt{prev}_\text{v}(h)$ is the previous halfedge around the vertex of $h$. Lastly, $\texttt{vertex}(h) = a$ gives the vertex at the source of $h$, and $\texttt{face}(h)$ the face bounded by $h$.

Relations exist between operators around faces and operators around vertices: $\texttt{next}(h) = \texttt{prev}_\text{v}(\texttt{opposite}(h))$, and $\texttt{prev}(h) = \texttt{opposite}(\texttt{next}_\text{v}(h))$ (and reciprocally $\texttt{next}_\text{v}(h) = \texttt{opposite}(\texttt{prev}(h))$ and $\texttt{prev}_\text{v}(h) = \texttt{next}(\texttt{opposite}(h))$). Thus it is enough to store relations around vertices or relations around faces. Note that we use here clockwise order for next, some papers use counter-clockwise, both conventions are possible. These halfedges and operators describe all the cells of the planar partition, and all the incidence and adjacency relations. For example, starting from a halfedge, and iterating on `next`, we traverse all the halfedges of a boundary of a face. Each face is characterized by one outer boundary and possibly some inner boundaries (one per hole). Another example is the test if two edges $e$ and $e'$ are adjacent that can be done by testing if there exists two halfedges $h$ and $h'$ describing $e$ and $e'$ so that $\texttt{vertex}(h) = \texttt{vertex}(h')$. A complete example is given in Fig. 1.

An HDS can be constructed incrementally during the sweep-line algorithm of [7] for arrangement computations. In this algorithm, the sweep line stores the ordered list of active segments from bottom to top. Segments in the line are updated locally depending on the configurations (begin and end of a segment, and new intersection). When the sweep line is moved to a vertex $v$, we can create, in the HDS, the pairs of halfedges describing all segments incident to $v$. Moreover, segments in the active line being ordered, $\texttt{prev}_\text{v}$ and $\texttt{next}_\text{v}$ can be directly defined without an additional sort. This gives an algorithm that builds the HDS describing the arrangement of segments having the same complexity as the algorithm of [7]: $O((n + k) \log n)$, where $n$ is the number of segments, and $k$ is the total number of intersection points (see for example [18]).

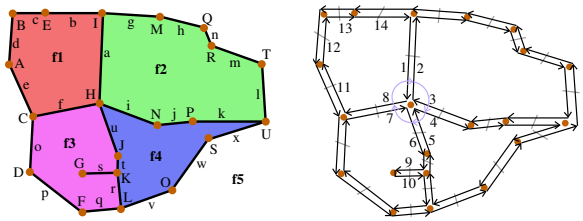



Fig. 1: **Example of planar partition and HDS**. (Left) An object with 4 bounded faces (`f1`, `f2`, `f3` and `f4`) plus an unbounded face (`f5`), 24 edges and 21 vertices. Faces `f1` and `f2` are adjacent since they share edge `a`; edges `a` and `f` are adjacent through vertex `H`. Edge `s` is *dangling*, i.e. attached to the border of its face by only one of its incident vertex (`K`). (Right) The HDS representing this planar partition with 48 halfedges (some being numbered). $\texttt{next}(8) = 11$, $\texttt{prev}(8) = 1$, $\texttt{next}_\text{v}(8) = 2$, $\texttt{prev}_\text{v}(8) = 6$, $\texttt{opposite}(8) = 7$ and $\texttt{vertex}(8) = H$. The sequence of halfedges (1,8,11,12,13,14) is the ordered halfedges around face $f1$, and the sequence (2,4,6,8) the ordered halfedges around vertex $H$. Edges $a$ and $f$ are adjacent because halfedge 2 describes edge $a$, halfedge 8 describes edge $f$ and $\texttt{vertex}(2) = \texttt{vertex}(8)$. Since edge `s` is dangling, $\texttt{next}_\text{v}(10) = \texttt{prev}(10) = 10$.

## 3 Stripped HDS

In this section we define our stripped version of the halfedge data structure.

### 3.1 Definitions

Strips are defined by vertical lines obtained by the decomposition of the real line into $s$ ($s \geq 1$) disjoint intervals $(-\infty, x^1[\cup[x^2, x^3[\cup \ldots \cup[x^{s-1}, +\infty)$, with $x^i < x^{i+1} \in \mathbb{R}$). Each interval defines a strip $S^i := [x^{i-1}, x^i[\times\mathbb{R}$ (with two special cases $S^1 := (-\infty, x^1[\times\mathbb{R}$ and $S^s := [x^{s-1}, \infty) \times \mathbb{R}$). The union of all strips is a partition of the plane.

Let us consider now a 2D planar partition $P = (V,E,F)$ with $V$ its vertices, $E$ its edges, and $F$ its faces. Each vertex $v \in V$ belongs exactly to one strip. Note that contrary to [27], strip boundaries may contain some vertices. An edge $e$ belongs to strip $S^i$ if $e \cap S^i \neq \emptyset$. Each edge $e \in E$ belongs to at least one strip, and at most to all strips. Edge $e$ is a *critical edge* if it belongs to more than one strip. Each critical edge $e$ is labeled with an unique id $\texttt{id}(e)$. Note that a critical edge cannot be vertical (indeed a vertical edge belongs necessarily to only one strip).

We define a stripped halfedge data structure as a sequence of independent local partial HDSs $\{H^i\}$, one

per strip, with unique identifiers associated with some halfedges allowing the retrieval of topological information between different strips. A *partial* HDS is an HDS in which we may have some undefined –null– links for some of its operators.

**Definition 1** Given a 2D planar partition $P = (V,E,F)$ and $s$ strips $\{S^i\}$, a *stripped halfedge data structure* encoding $P$, denoted *s-HDS*, is a set of partial HDS $\{H^1,\ldots,H^s\}$ and a set of faces $F$. $H^i$ describes the restriction of $P$ to strip $S^i$ $\forall i$, $1 \leq i \leq s$. More precisely:

- $H^i$ contains all vertices that belong to $S^i$;
- each edge that belongs to strip $S^i$ is described in $H^i$ by two halfedges linked by $\texttt{opposite}^i$;
- for each halfedge $h \in H^i$ describing the edge $[AB]$, we define $\texttt{vertex}^i(h) := A$ if $A \in S^i$, otherwise $\texttt{vertex}^i(h) :=$ null. In this case, $h$ is called an *external* halfedge.
- for each non-external halfedge $h \in H^i$, $\texttt{prev}^i_{\text{v}}(h)$ is the previous halfedge around its vertex (and $\texttt{next}^i_{\text{v}}(h)$ the next one);
- for each external halfedge $h \in H^i$, $\texttt{prev}^i_{\text{v}}(h)$ and $\texttt{next}^i_{\text{v}}(h)$ are set to null;
- for each halfedge $h \in H^i$ describing a critical edge $e$, we extend the identifier map using $\texttt{id}(h) := \texttt{id}(e)$ if $h$ is oriented from left to right, and $\texttt{id}(h) := -\texttt{id}(e)$ otherwise (*i.e.* $h$ is from right to left);
- for each halfedge $h \in H^i$, $\texttt{face}^i(h)$ is the face bounded by $h$. Contrary to vertices, a face may be shared by multiple strips, leading to a global index of faces.

The partial HDS $\{H^i\}$ are independent data structures which are made globally consistent using the unique identifier labeling. These links are implicit through the global labeling `id` of halfedges describing critical edges. Having no explicit links between halfedges of two different strips is of great interest to simplify the local computations on each strip, as illustrated in this paper for parallel and streamed arrangements. From the definition, we can remark the following important properties on a s-HDS:

1. for each halfedge $h \in H^i$, its opposite $\texttt{opposite}^i(h)$ belongs, by definition, to the same $H^i$;
2. for each non-external halfedge $h \in H^i$, $\texttt{prev}^i_{\text{v}}(h)$ and $\texttt{next}^i_{\text{v}}(h)$ are also non-external halfedges in the same $H^i$. Indeed by definition there are two halfedges associated with the same vertex in the strip $S^i$, and thus they are non-external;
3. each edge of the planar partition is described by exactly two non-external halfedges, and possibly any even number of external halfedges.

An external halfedge is called *left-external* (resp. *right-external*) if it is external, and oriented from left to right (resp. from right to left). A left-external halfedge (resp. right-external) traverses the left border of the strip (resp. right border). Note that, as critical edges, external halfedges cannot be vertical.

In the previous section, we have seen that $\texttt{next}(h) = \texttt{prev}_{\text{v}}(\texttt{opposite}(h))$, and $\texttt{prev}(h) = \texttt{opposite}(\texttt{next}_{\text{v}}(h))$. In an s-HDS, links between halfedges are stored using $\texttt{prev}^i_{\text{v}}$ and $\texttt{next}^i_{\text{v}}$ because these relations always stay inside one strip (contrary to $\texttt{next}^i$ and $\texttt{prev}^i$). All these notions are illustrated in Figures 2 and 3.

### 3.2 Global Topological Operators from an s-HDS

From the s-HDS definition, we can define global topological operators through the local links in each $H^i$ and the `id` of the halfedges describing critical edges.

**Definition 2** Let $SH$ be an s-HDS and $h$ a halfedge. We denote by $\texttt{ne}(h)$ the non-external halfedge describing edge $\texttt{id}(h)$ and having the same orientation as $h$.

**Algorithm 1:** Non-external halfedge.

**Input:** $HS = \{H^1,\ldots,H^s\}$: An s-HDS;
$(h,i)$: A halfedge in $H^i$.
**Output:** $\texttt{ne}(h)$.

```
while h is external do
    if h is left-external then  i ← i − 1 ;
    else  i ← i + 1 ;
    h ← halfedge in H^i having id(h) as id;
```

As seen in the previous section, a critical edge is described by exactly two non-external halfedges having opposite orientations, and some external halfedges. A non-critical edge is described by only two non-external halfedges with opposite orientations (and thus no external halfedges). This means that for any halfedge $h$, $\texttt{ne}(h)$ is unique. But two different halfedges $h$ and $h'$ can have equal `ne`, `ne` is a surjective function. Note that if $h$ is non-external, $\texttt{ne}(h) = h$.

Algorithm 1 allows computing $\texttt{ne}(h)$, by iteratively traversing the adjacent strips, identifying the halfedges thanks to their `id`. Given a left-external halfedge, finding the corresponding halfedge in the left strip is done directly by using the `id` associated with halfedges through an associative array giving for each `id` its corresponding halfedge. The complexity of Algorithm 1 is linear in number of strips traversed, when using an associative container with constant time access in average, such as hash maps.

We can combine $\texttt{opposite}^i$, $\texttt{next}^i_{\text{v}}$, $\texttt{prev}^i_{\text{v}}$, the local relationships between halfedges within strips, and `ne` to

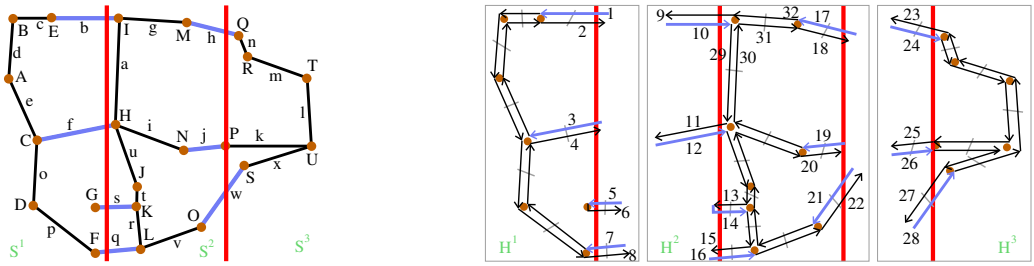



Fig. 2: **Example of s-HDS**. (Left) The same 2D object as in Fig. 1, but here cut in three strips. $S^1$ contains vertices $\{A,B,C,D,E,F,G\}$ and edges $\{b,c,d,e,f,o,p,q,s\}$; $S^2$ has vertices $\{H,I,J,K,L,M,N,O\}$ and edges $\{a,b,f,g,h,i,j,q,r,s,t,u,v,w\}$; and $S^3$ contains vertices $\{P,Q,R,S,T,U\}$ and edges $\{j,h,k,l,m,n,w,x\}$. Edges $\{b,f,h,j,q,s,w\}$ are critical (drawn in blue), each one belongs to two strips. Vertex $P$ belongs to the left border of strip $S^3$ and thus edge $j$ is critical since it intersects both strips $S^2$ and $S^3$. Edge $k$ is not critical since strip $S^2$ does not contain vertex $P$, thus intersection of $k$ and strip $S^2$ is empty. (Right) The stripped HDS representing this 2D object, with three HDS $\{H^1,H^2,H^3\}$. $H^1$ has four external halfedges $\{1,3,5,7\}$ (drawn in blue), $H^2$ has seven external halfedges $\{10,12,14,16,17,19,21\}$ and $H^3$ has three $\{24,26,28\}$. halfedge 10 is left-external and 17 is right-external. Let us consider that edges are labeled with the `id` given in the left part of the figure. We have for example $\texttt{id}(2) = b = \texttt{id}(10)$ and $\texttt{id}(1) = -b = \texttt{id}(9)$. Critical edge $b$ is described by 4 halfedges in the s-HDS (two non-external 2, 9 and two external 1, 10), and non-critical edge $a$ by two non-external halfedges. $\texttt{prev}_\text{v}^i$, $\texttt{next}_\text{v}^i$ and $\texttt{vertex}^i$ are defined in each strip for each non-external halfedges (*e.g.* 2 in $H^1$), and are null for external ones (e.g. 1 in $H^1$).

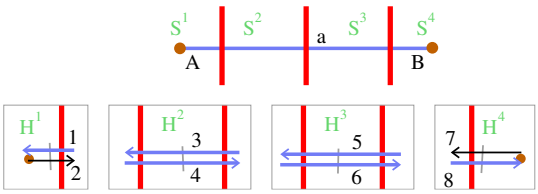



Fig. 3: **Example of a critical edge that belongs to more than two strips**. (*Top*) Edge $a$ belongs to four strips. (*Bottom*) $a$ is represented by four pairs of halfedges in the corresponding s-HDS, one pair in each $H^i$. Among these 8 halfedges, two are non-external (2 and 7, having the two extremities of the edge as vertex), the six others are external. Halfedges in strips $S^2$ and $S^3$ traverse the strip without having a vertex in these strips. `id` of halfedges 2, 4, 6 and 8 is the same ($a$).

retrieve directly the operators on a global topological representation through the different strips.

**Definition 3** Let $SH = \{H^1,\ldots,H^s\}$ an s-HDS. The HDS describing the same planar partition as $SH$, denoted $\texttt{global}(SH)$, is defined by:

1. The set of halfedges of $\texttt{global}(SH)$ is the union of all non-external halfedges of all $H^i$;
2. The set of vertices of $\texttt{global}(SH)$ is the union of all vertices of all $H^i$;
3. The set of faces of $\texttt{global}(SH)$ is the set of faces of $SH$;
4. For each halfedge $h$ of $\texttt{global}(SH)$:
   (a) $\texttt{opposite}(h) := \texttt{ne}(\texttt{opposite}^i(h))$;
   (b) $\texttt{next}_\text{v}(h) := \texttt{next}_\text{v}^i(h)$;
   (c) $\texttt{prev}_\text{v}(h) := \texttt{prev}_\text{v}^i(h)$;
   (d) $\texttt{vertex}(h) := \texttt{vertex}^i(h)$.
   (e) $\texttt{face}(h) := \texttt{face}^i(h)$.

Thanks to this definition, we can use an s-HDS to traverse the implicit corresponding global HDS. It is enough to consider only non-external halfedges, and to use the global operators defined above. Using these global operators, we can retrieve the global operators `next` and `prev` using the formula from Section 2.

In the s-HDS shown in Fig. 2, we can verify that $\texttt{ne}(2) = \texttt{ne}(10) = 2$. $\texttt{opposite}(9) = \texttt{ne}(\texttt{opposite}^1(9)) = \texttt{ne}(10) = 2$, $\texttt{next}_\text{v}(9) = \texttt{next}_\text{v}^1(9) = 31$ and $\texttt{prev}_\text{v}(9) = \texttt{prev}_\text{v}^1(9) = 29$. $\texttt{next}(2) = \texttt{prev}_\text{v}(\texttt{opposite}(2)) = \texttt{prev}_\text{v}(9) = 29$ and $\texttt{prev}(32) = \texttt{opposite}(\texttt{next}_\text{v}(32)) = \texttt{opposite}(18) = 17$.

### 3.3 Topological Equivalence Between the Different Structures

We now show that given a 2D planar partition $P$, the HDS representing $P$ is isomorphic to $\texttt{global}(SH)$, $SH$ being a stripped HDS representing $P$.

**Theorem 1** *Let $P = (V,E,F)$ a 2D planar partition, $H$ the HDS representing $P$, and $SH$ an s-HDS representing $P$. $H$ and $\texttt{global}(SH)$ are isomorphic.*

For the proof, we use the definition of HDS (Section 2), the definition of s-HDS (Def. 1) and the definition of $\texttt{global}(SH)$ (Def. 3).

1. The vertices and the faces are the same in $H$, $SH$ and $\texttt{global}(SH)$;
2. Each edge of $E$ is described by two halfedges in $H$ and two non-external halfedges in $SH$ (plus possibly some external halfedges). The set of halfedges of $\texttt{global}(SH)$ being the union of all non-external halfedges of $SH$, there is a one-to-one mapping $m$ between halfedges of $H$ and halfedges of $\texttt{global}(SH)$ that preserves sources and targets of halfedges in $H$ and halfedges in $SH$;
3. For each halfedge $h \in H$, $h_n := \texttt{next}_\text{v}(h)$ is the next halfedge around the source of $h$. If $h' := m(h)$ belongs to $H^i$ in $SH$, then $h'_n := \texttt{next}^i_\text{v}(h')$ in $SH$ is the next halfedge around the source of $h'$. Since $h'_n = \texttt{next}_\text{v}(h')$ is in $\texttt{global}(SH)$ (Def. 3-4b), we conclude that $h'_n = m(h_n)$, and thus $\texttt{next}_\text{v}(m(h)) = m(\texttt{next}_\text{v}(h))$ (a similar proof holds for $\texttt{prev}_\text{v}$ using Def. 3-4c);
4. For each halfedge $h \in H$, $h_o := \texttt{opposite}(h)$ is the other halfedge describing the same edge $e$ as the edge of $h$ but with reverse orientation. Edge $e$ is described in $SH$ by two non-external halfedges $h' := m(h)$ and $h'' := m(h_o)$, with opposite orientations, and possibly some external halfedges. By Def. 3-4a, $\texttt{opposite}(h') = \texttt{ne}(\texttt{opposite}^i(h'))$. Edge $\texttt{opposite}^i(h')$ gives a halfedge with other orientation than $h'$. $\texttt{ne}$ preserves the orientation. Thus $\texttt{opposite}(h')$ gives the non-external halfedge with other orientation than $h'$: this is $h''$. Thus we can conclude that $\texttt{opposite}(h') = h''$, and thus $\texttt{opposite}(m(h)) = m(\texttt{opposite}(h))$.

There is a one-to-one mapping between $H$ and $\texttt{global}(SH)$ that preserves the operators $\texttt{next}_\text{v}$, $\texttt{prev}_\text{v}$, $\texttt{opposite}$, $\texttt{vertex}$, $\texttt{face}$ (and thus $\texttt{next}$ and $\texttt{prev}$): this proves that $H$ and $\texttt{global}(SH)$ are isomorphic and thus that they represent the same planar partition. Note that this is true whatever the number of strips of $SH$. As a corollary of this theorem, $\texttt{global}(SH)$ is a valid HDS structure as described in Section 2. This shows that an s-HDS can represent all the specific configurations of planar subdivision, like HDS can (such as loops, degeneracies, isolated points, ...).

### 3.4 Transformations Between HDS and s-HDS

We have just proven that an HDS and its stripped version are equivalent. In this section we give the transformation algorithms allowing to convert an HDS into an s-HDS, and conversely.

Given an HDS $H$ and $s$ strips $S^1, \ldots, S^s$, we can build the s-HDS $SH := \{H^1, \ldots, H^s\}$ describing the same planar partition as $H$, but within $s$ independant strips, by using Algorithm 2. Its complexity is $O(sn)$, $n$ being the number of halfedges of $H$ (if we use an associative container with constant time access in average, such as hash maps). Reciprocally, we can build a global HDS from a stripped representation using the definition of global operators given in Section 3.2 (Algorithm 3 with the same complexity as Algorithm 2).

**Algorithm 2:** HDS to s-HDS.

**Input:** $H$: An HDS;
$S^1, \ldots, S^s$: $s$ strips.
**Output:** $SH = \{H^1, \ldots, H^s\}$: The s-HDS describing the same planar partition than $H$ on strips $S^i$.

```
n ← 1;
foreach non-vertical halfedge e of H oriented from left to right do
    id(h) ← n;  id(opposite(h)) ← −n;  n ← n + 1;
for i ← 1 to s do
    foreach vertex v in H that belongs to S^i do
        m^i_v(v) ← a new vertex in H^i;
    foreach halfedge h in H that intersects strip S^i do
        m^i(h) ← a new halfedge in H^i;
        id(m^i(h)) ← id(h);
        if vertex(h) belongs to S^i then
            vertex^i(m^i(h)) ← m^i_v(vertex(h));
    foreach halfedge h in H that intersects strip S^i do
        if vertex^i(h) ≠ null then
            prev^i_v(m^i(h)) ← m^i(prev_v(h));
            next^i_v(m^i(h)) ← m^i(next_v(h));
        opposite^i(m^i(h)) ← m^i(opposite(h));
```

**Algorithm 3:** s-HDS to HDS.

**Input:** $SH = \{H^1, \ldots, H^s\}$: An s-HDS.
**Output:** $H$: The HDS describing the same planar partition than $SH$.

```
for i ← 1 to s do
    foreach vertex v in H^i do
        m_v(v) ← a new vertex in H;
    foreach non-external halfedge h in H^i do
        m(h) ← a new halfedge in H;
        vertex(m(h)) ← m_v(vertex^i(h));
for i ← 1 to s do
    foreach non-external halfedge h in H^i do
        opposite(m(h)) ← m(ne(opposite^i(h)));
        next_v(m(h)) ← m(next^i_v(h));
        prev_v(m(h)) ← m(prev^i_v(h));
```

## 4 Parallel Computation of Arrangement of Segments

In this section, we use the s-HDS data structure to design a fast parallel algorithm to compute the arrangement of segments. Let us consider a set of segments $\Sigma := \{\sigma^1, \ldots, \sigma^n\}$, and $s$ vertical strips $S^1, \ldots, S^s$. A segment $\sigma^j$ is *concerned* by strip $S^i$ if $\sigma^j \cap S^i \neq \emptyset$. Note that a segment can be concerned by several/all strips, and each segment is concerned by at least one strip.

The main principle of our parallel method (cf. Algorithm 4) is to label each segment with a unique global id, then to extract in parallel each local HDS, one per strip, using the classical sweep-line algorithm (line 4), *e.g.* using [7]. Since these local HDS are fully independent, there is no critical section, nor any step after the parallel computation to gather the different parts.

**Algorithm 4:** Computation of Arrangement of Segments in Parallel.

**Input:** $\Sigma$: A set of segments;
$S^1$,...,$S^s$: $s$ strips.
**Output:** $HS$: The s-HDS representing the arrangement of $\Sigma$.

1 Let $HS$ be an empty s-HDS having with $S^1$,...,$S^s$ strips;
2 Label each segment in $\Sigma$ with a unique identifier;
3 **parallel for each** *strip* $S^i$ **do**
4 Compute in $H^i$, the arrangement of segments concerned by strip $S^i$;
5 Remove vertices and halfedges of $H^i$ with no intersection with $S^i$;
6 Compute faces;
7 **return** $HS$

When computing the local arrangement $H^i$ of segments concerned by strip $S^i$, it is possible to obtain vertices or halfedges outside the strip that need to be pruned (line 5). Indeed, let us consider for example two long edges that are concerned by strip $S^i$, but that intersect before its left border. The intersection point, and the two segments on its left are outside the strip, and thus will not be described in $H^i$.

When a vertex is outside the strip, it is not described in $H^i$, and thus the halfedge having this vertex as source has `vertex` equal to null, which means that the halfedge is external. The ids of the halfedges are set during the algorithm using the global ids of the edges, ensuring a consistent labeling of the different strips. Faces are computed after all local HDS have been obtained, iterating through all the halfedges and finding connected components and their inclusion.

After the computation of the s-HDS, we can use Algorithm 3 if we want to construct a global representation of the entire arrangement. Note that for traversal purposes for instance, we can directly rely on the s-HDS $SH$ without constructing `global`($SH$).

The complexity of our parallel algorithm is $O(s(n+k)\log n)$, $n$ being the total number of segments, $k$ the total number of intersection points and $s$ the number of strips. Indeed, in the worst case, each segment belongs to all strips. To improve the worst case, it is possible to crop segments before to add them in the local arrangement. This cropping avoid to compute intersections outside the current strip in the sweep line algorithm.

## 5 Out-of-core Streamed Construction of Arrangement of Segments

**Algorithm 5:** Streamed Computation of Arrangement of Segments.

**Input:** $\Sigma$: A stream containing an ordered sequence of segments;
$q$: Number of segments in a chunk.
**Result:** $HS$: The s-HDS representing the arrangement of segments stored to disk.

Let $HS$ be an empty s-HDS;
Let `ActiveSegments` be a list of segments;
$i \leftarrow 1$; $j \leftarrow 1$; $x_{\min} \leftarrow -\infty$;
**while** $\Sigma$ *is not empty* **do**
  $n \leftarrow 1$;
  **while** $n \leq q$ **and** $\Sigma$ *is not empty* **do**
    Read next segment $\sigma^j$ in S, label it $j$, and add $\sigma^j$ at the end of `ActiveSegments`;
    $n \leftarrow n+1$; $j \leftarrow j+1$;
  **if** $S$ *is empty* **then**
    $x_{\max} \leftarrow \infty$; `// last strip`
  **else**
    $x_{\max} \leftarrow$ maximum x-coordinates of source vertices in `ActiveSegments`;
  Compute in $H^i$, the arrangement of segments in `ActiveSegments`;
  Remove vertices and halfedges of $H^i$ with no intersection with $S^i$;
  Swap $H^i$ to disk;
  $i \leftarrow i+1$; $x_{\min} \leftarrow x_{\max}$;
  Remove from `ActiveSegments` all segments smaller than $x_{\min}$;

We can use an s-HDS to build an arrangement of segments in an out-of-core streamed algorithm when segments are given in an ordered way for their source vertices. The main principle of Algorithm 5 is to load from the disk a first chunk of segments, compute its local HDS, and swap it to disk before to load a second chunk. The size of the chunks (*i.e.* the number of segments read from the disk) is the parameter $q$ given by users. Thanks to the s-HDS definition, links between

the different HDS are done only through the global labeling, simplifying the swap to disk and the global consistency.

Before loading from disk the next set of segments, we remove from the list of active segments all segments that are entirely to the left of the beginning of the next strip. Segments having their target after the beginning of the next strip are kept. Edges, and its halfedges, are labeled with unique global id, ensuring the topological validity of the s-HDS. In this context, the computation of faces is performed as a post-process, from the resulting s-HDS (merging the local faces obtained per strip). Note that having the segments sorted is not an issue at all in our streamed and out-of-core approach. Segments can be already defined sorted for some specific applications, or sorting them, even in out-of-core scenarios, is a very small preliminary step in the overall arrangement computation problem.

## 6 Experiments

We have implemented our new method to compute arrangement of segments, both the parallel and the streamed versions. Our code uses exact arithmetic computations with real numbers provided by the CGAL kernel *Exact_predicates_exact_constructions_kernel* [13]. In this kernel, several mechanisms were developed to speedup evaluations and computations: efficient lazy evaluations based on interval arithmetic [29] and algebraic methods and arithmetic filtering to define exact predicates [16]. We compared our solution with the sweep line arrangement method provided in CGAL[1] [35]. To our knowledge, this is the only method publicly available that provides fully exact numerical calculations and thus guarantees no error during intersection computations. Note that we also rely on this implementation for the independent, per strip, local arrangement computations (Algorithm 4-line 4).

All experiments[2] were run on an AMD®Ryzen 3970X 32-Core processor with 126 GB RAM. Each computation time is the average of 5 runs on the same input (we do not count the I/O of segment loading).

### 6.1 Datasets

We have conduced our experiments using synthetic random data, input segments from a sketch and GIS data of countries. More precisely, we have considered:

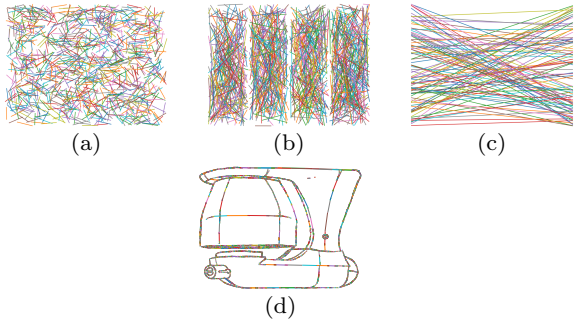



Fig. 4: **Input random segments and sketch**. (*a*) Random. 20000 segments, uniformly randomly drawn within the domain with random directions. Segment sizes follow Gaussian distribution ($\mu = 800$, $\sigma = 80$ for `Rand-short` and $\mu = 1600$, $\sigma = 160$ for `Rand-long`). (*b*) `Best-32`: best-case with 32 chunks of 50000 independent random segments (random size between 0 and 1600, making sure that each segment remains inside its strip). (*c*) `Worst`: all segments go from the left to the right side of the domain (3500 segments with random endpoints on the borders). ($a-c$) Domain size is $[0,10000]^2$. (*d*) A large sketch from [22].

1. three kind of random input segments: a pure random one (the segment size follows a normal distribution with uniform random orientation and location on a $[0,10000]^2$ domain, cf. Fig. 4a), a *best-case* for algorithms based on a stripped decomposition of the input (independent sets of random segments, each segment will be in a single strip, cf. Fig. 4b), and a *worst-case* in which all random segments are created from the left to the right border of the domain, and thus each one belongs to all strips (cf. Fig. 4c). Please refer to Fig. 4 for details on the different parameters used in the experiments, and Table 1 for some statistical properties of the segment distributions;
2. input segments from a sketch from [22][3] (cf. Fig. 4d), the arrangement computation can be seen as a preliminary step for more complex sketch processing;
3. GIS dataset consists of spatial databases of ten countries (Australia, Brazil, Canada, China, Germany, France, United Kingdom, Japan, Russia and USA) available at `http://www.gadm.org/` (cf. France in Fig. 5). They are in `shapefile` format, a popular geospatial vector data format for geographic information system (GIS) software. The different amount of input segments are given in Table 2 for each country; it is 1,683,488 in average. This table also gives the number of vertices, edges and faces in the final

[1] Thanks to the GeometryFactory company for its help to compare our method with the one in CGAL.

[2] The link to the code and the scripts used in this paper is `https://gitlab.liris.cnrs.fr/gdamiand/stripped-hds`.

[3] `https://repo-sam.inria.fr/d3/OpenSketch/`

arrangement computed from the input set of segments. Usually, GIS data suffers of many geometrical and topological errors, implying problems for algorithms that try to process these data [9]. When errors exist, the data must be repaired. This is often achieved manually implying a long and error-prone process. Using an arrangement of segments allows to automatically correct errors, producing a topologically valid planar partition.

For all these test-cases, Table 1, Fig. 5 and Table 2 summarize the size of the input segment sets and some empirical statistical measures (distribution of segment lengths, of segment orientation and heat map of the segment midpoint locations).

### 6.2 Parallel Computation Evaluation

In this first experiment, we have compared the computation time of the arrangement of segments construction for CGAL method and for our parallel algorithm based on the s-HDS data structure. We have used the parallel computation algorithm, with a number of threads equal to the number of strips, for an increasing number of strips from 1 to 32.

Table 1 presents the results for the stochastic datasets and the sketch. On the rightmost column, the first green bar corresponds to the CGAL method timing. A first observation is that, on a single thread, our approach is equivalent to the CGAL version (even slightly better). Furthermore, as we increase the number of threads, we considerably improve the overall timing thanks to the independent, strip-based, computation of the local arrangement of the s-HDS. For our method, the global part (in green) is the time spent for the final construction of faces.

For the GIS dataset, we can see in Table 2 that our method is always faster than CGAL method: on average, 3.5 times faster with 32 strips (2.83 seconds against 9.86 seconds). The timings of our method show that the computation time decreases while the number of strips, and thus the number of threads, increases.

When the number of threads increases, and so the number of strips, the speedup factor decreases due to the number of critical edges which increases. Since these edges belong to several strips, they are considered by several threads, explaining an overhead for the computation time. Note that there are also cache issues and memory bandwidth when accessing to the shared memory.

In these experiments, strip positions are regularly spaced in the bounding box of the input segments. This strategy has a visible impact on the thread workload (*e.g.* in the worst-case scenario with more intersections to be computed by threads dedicated to the central part of the domain). Designing a heuristic to balance the workload is a very challenging task as we need to have a fast estimate on the number of intersections that must work all datasets. The heuristics we tried did not impact significantly the overall timings. We leave this as an interesting future work.

In these experiments, we use the optional possibility to crop the critical segments. Indeed, when there are many long segments, using this option improves the computation times of our parallel arrangement method.

The number of critical edges, and thus the number of external halfedges, increases with the number of strips. This implies an overhead in memory since halfedges may be duplicated in several strips but such overhead is negligible. For example, for `Rand-long`, it increases from 2.46 GB for 1 strip to 2.96 GB for 32 strips (from 1.96 GB to 2.17 GB for the GIS data). Another drawback could be the overhead on computation times when traversing the s-HDS, for instance to visit each face of the arrangement. Again, this overhead is also negligible: for `Rand-long`, the complete traversal time increases from 0.52 seconds for 1 strip to 0.57 seconds for 32 strips (from 0.21 to 0.22 seconds for the GIS data).

### 6.3 Streamed Arrangement Computation

In a second experiment, we have studied the memory space consumption during the arrangement computation (which is one of the key features for streamed approaches). We have performed a comparison between CGAL, where all segments are inserted in one batch, our method with only one strip, and finally the streamed construction version introduced in Section 5: at each 10,000 new segments, one local HDS is computed then swapped on disk and thus removed from central memory.

Memory space consumption is computed using Heaptrack [36], a tool that traces and analyzes all memory allocations. One result is given in Fig. 6 for Australia. These graphs show the memory space consumed depending on the elapsed time of the software. Note that the timings observed here are much higher than the computation times taken by the computation of the arrangements due to the important overhead taken by Heaptrack to track memory space allocations/deallocations. As expected, the memory space consumption for CGAL and for our method with one strip increases progressively when segments are successively added in the arrangement. We can remark that our method occupied slightly less memory than CGAL: for

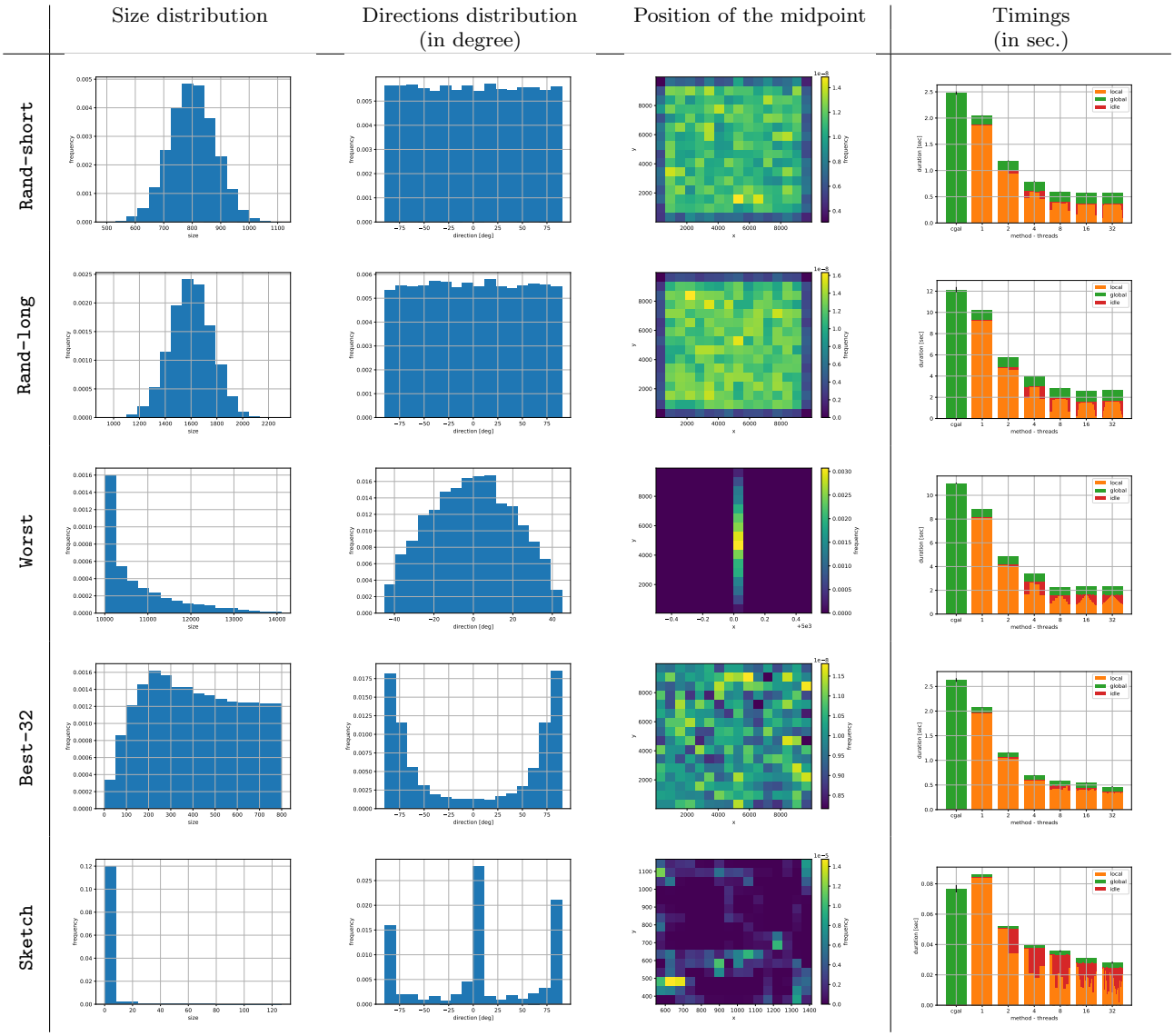



Table 1: **Input segment distributions and results**. For the data described in Fig. 4, we first detail some empirical statistical measures of the input segments (histograms of size distribution, direction distribution and position of the segment midpoints). The last column details the timings of our parallel arrangement computation algorithm for various numbers of threads. Each thread workload is represented by an orange bar, with possible idle time in red if its processing has been completed (note that threads are illustrated, from left to right, following the same left-right order of the strips). The first bar on the last column corresponds to the CGAL approach. Results for stochastic data are averaged on 50 realizations.

Australia, 1.5 GB instead of 2.3 GB. The most interesting result is the memory space consumption for the streamed construction method. Indeed, the memory space consumption stays approximately constant, and very low, showing that it is now possible with this method to compute an arrangement of a huge number of segments without any memory problem. For Australia, only 15 MB of memory is used as peak, and about 12 MB as average. Similar behaviors are observed for other data in our dataset.

About the timings, in average for the 10 GIS files, computing the arrangement of segments with the streamed method takes 18.6 seconds, against 9.7 seconds for our method (sequential version, one strip). The overhead is due to the time spent by the writing of the data to disk. Without this overhead, the streamed version takes only 7.1 seconds, which is faster than the sequiential one. This could be explained by the memory access which is much faster due to the small memory used.

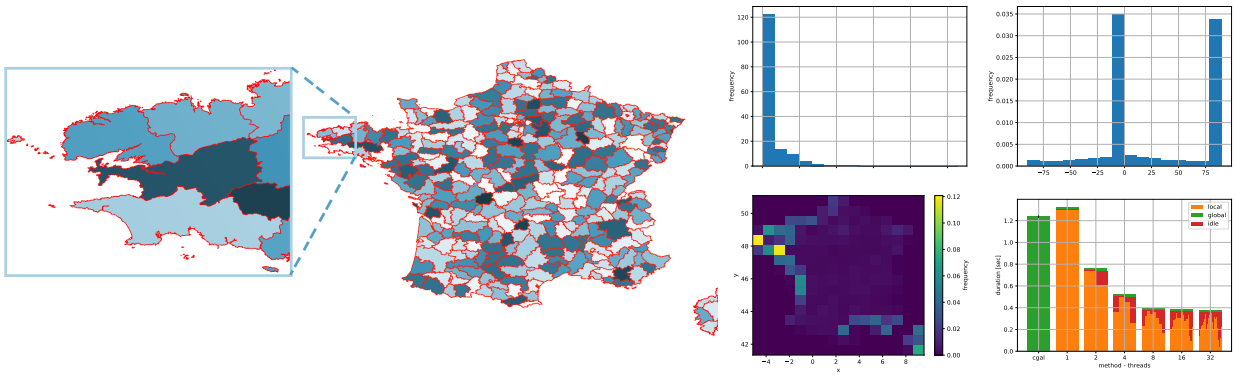
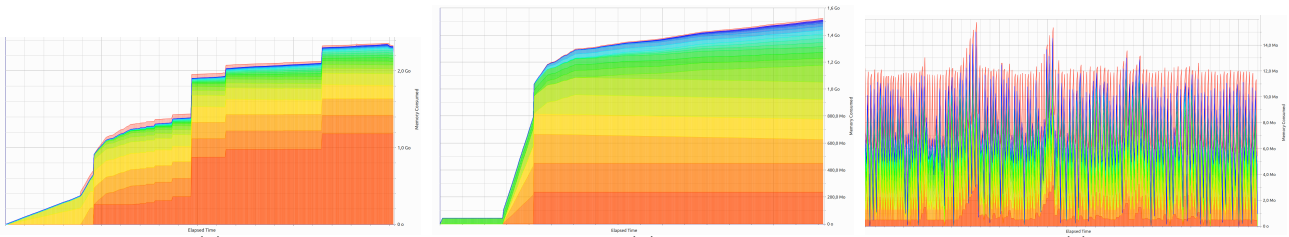



Fig. 5: **GIS experiments**. (Left) One of a GIS model used in the tests (FRA). (Right) Some statistical properties of the input set of segments as well as some detailed timings for the parallel arrangement computation.

| File | #segments | #vertices | #edges | #faces | CGAL | Ours | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | 1 | 2 | 4 | 8 | 16 | 32 |
| AUS | 1,339,516 | 1,307,542 | 1,307,803 | 4,685 | **5.05** | 5.08 | 3.15 | 1.98 | 1.78 | 1.28 | **1.14** |
| BRA | 1,134,931 | 898,552 | 949,550 | 51,808 | **8.55** | 8.52 | 6.04 | 5.11 | 3.30 | 2.33 | **2.23** |
| CAN | 5,195,304 | 4,577,948 | 4,594,771 | 41,404 | **23.47** | 23.89 | 16.39 | 9.98 | 9.23 | 6.96 | **6.70** |
| CHN | 1,758,983 | 1,244,209 | 1,246,654 | 4,455 | **16.28** | 15.59 | 11.91 | 8.69 | 5.40 | **3.86** | 4.06 |
| DEU | 1,453,505 | 897,406 | 973,585 | 76,573 | **16.77** | 16.39 | 9.16 | 6.74 | 4.87 | 4.75 | **4.73** |
| FRA | 278,021 | 246,449 | 246,884 | 800 | **1.28** | 1.32 | 0.74 | 0.47 | 0.33 | 0.33 | **0.31** |
| GBR | 431,807 | 423,714 | 423,905 | 1,110 | **1.15** | 1.17 | 0.96 | 0.71 | 0.48 | 0.39 | **0.33** |
| JPN | 866,161 | 712,056 | 713,879 | 3,223 | **5.49** | 5.47 | 3.31 | 2.66 | 2.22 | 1.75 | **1.57** |
| RUS | 1,842,592 | 1,735,465 | 1,737,761 | 8,443 | **7.12** | 7.45 | 7.37 | 4.17 | 2.65 | 2.50 | **2.48** |
| USA | 2,534,063 | 2,290,217 | 2,293,616 | 12,034 | **13.46** | 13.40 | 13.22 | 7.27 | 6.71 | 6.78 | **4.80** |
| Mean | 1,683,488 | 1,433,356 | 1,448,841 | 20,454 | **9.86** | 9.83 | 7.23 | 4.78 | 3.69 | 3.09 | **2.83** |

Table 2: **Number of elements and timings for GIS data**. Number of input segments, and number of cells of the final arrangement, for the ten countries used in our experiments. On the right part of the table, timings in seconds of CGAL arrangement and our method (for increasing number of threads and strips, from 1 to 32).



(a) (b) (c)

Fig. 6: **Memory space comparison for the streamed arrangement computation of Australia**. (*a*) CGAL: memory increases progressively to about 2.3 GB. (*b*) Our global method with 1 strip: memory increases progressively to about 1.5 GB, whereas the memory stays constant (around 12MB) in our streamed version (*c*).

## 7 Conclusion

In this paper, we have defined the stripped halfedge data structure, a sequence of independent partial HDS, to represent a planar partition which it is equivalent to a global HDS describing the same planar partition.

As the local HDS are fully independent with links between local and global operations, we have proposed a parallel algorithm to compute arrangement of segments, where different threads extract in parallel each local HDS, one per strip. Since these local HDS are fully independent, there is no critical section, nor any step after the parallel computation to gather the different parts (only the faces require a fast extra pruning step). Moreover, extra associated arrays used to store external halfedges lead to negligible memory and computational overhead. We also defined a method to build the arrangement in an out-of-core streamed way, allowing computing an arrangement of a huge number of segments with a limited amount of memory.

In future work, we plan to define topological operations directly on the s-HDS such as edge removal and edge contraction, and operations to split a strip in several strips, or to merge some adjacent strips in one. Combining these future operations with the ability of swapping some strips on disk will provide an entire robust framework allowing creating, traverse and modify huge planar partitions.

## References


1. Agarwal, P.K., Sharir, M.: Arrangements and their applications. In: J.R. Sack, J. Urrutia (eds.) Handbook of Computational Geometry, chap. 2, pp. 49–119. North-Holland, Amsterdam (2000)
2. Aggarwal, A., Chazelle, B., Guibas, L., Ó'Dúnlaing, C., Yap, C.: Parallel computational geometry. Algorithmica **3**(1-4), 293–327 (1988)
3. Anderson, R., Beanie, P., Brisson, E.: Parallel algorithms for arrangements. Algorithmica **15**(2), 104–125 (1996)
4. Atallah, M.J., Goodrich, M.T.: Efficient plane sweeping in parallel. In: Proc. of second annual symposium on Computational geometry, pp. 216–225 (1986)
5. Balaban, I.J.: An optimal algorithm for finding segments intersections. In: Proc. of Eleventh Annual Symposium on Computational Geometry, SCG'95, pp. 211–219. Association for Computing Machinery, New York, NY, USA (1995)
6. Baumgart, B.: A polyhedron representation for computer vision. In: Proc. of AFIPS National Computer Conference, vol. 44, pp. 589–596 (1975)
7. Bentley, J.L., Ottmann, T.A.: Algorithms for reporting and counting geometric intersections. IEEE Transactions on Computers **28**(9), 643–647 (1979)
8. de Berg, M., van Kreveld, M., Overmars, M.H., Cheong, O.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer-Verlag, Berlin, Germany (2008)
9. Biljecki, F., Ledoux, H., Du, X., Stoter, J., Soon, K.H., Khoo, V.H.S.: The most common geometric and semantic errors in CityGML datasets. ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci. **IV-2/W1**, 13–22 (2016)
10. Boissonnat, J.D., Teillaud, M. (eds.): Effective Computational Geometry for Curves and Surfaces. Springer-Verlag Berlin Heidelberg (2006)
11. Boissonnat, J.D., Yvinec, M.: Algorithmic Geometry. Cambridge University Press, Cambridge, UK (1998)
12. Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., Lévy, B.: Polygon Mesh Processing. AK Peters (2010)
13. Brönnimann, H., Fabri, A., Giezeman, G.J., Hert, S., Hoffmann, M., Kettner, L., Pion, S., Schirra, S.: 2D and 3D linear geometry kernel. In: CGAL User and Reference Manual, 5.0.2 edn. CGAL Editorial Board (2020). URL https://doc.cgal.org/5.0.2/Manual/packages.html#PkgKernel23
14. Chazelle, B., Edelsbrunner, H.: An optimal algorithm for intersecting line segments in the plane. J. ACM **39**(1), 1–54 (1992)
15. Damiand, G., Lienhardt, P.: Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing. A K Peters/CRC Press (2014)
16. Devillers, O., Fronville, A., Mourrain, B., Teillaud, M.: Algebraic methods and arithmetic filtering for exact predicates on circle arcs. Computational Geometry **22**, 119–142 (2002)
17. Edelsbrunner, H.: Algorithms in Combinatorial Geometry. Springer-Verlag Berlin Heidelberg, Berlin, Germany (1987)
18. Fogel, E., Halperin, D., Wein, R.: CGAL Arrangements and Their Applications - A Step-by-Step Guide., *Geometry and computing*, vol. 7. Springer (2012)
19. Goodrich, M.T.: Intersecting line segments in parallel with an output-sensitive number of processors. SIAM Journal on Computing **20**(4), 737–755 (1991)
20. Goodrich, M.T., Ghouse, M.R., Bright, J.: Sweep methods for parallel computational geometry. Algorithmica **15**(2), 126–153 (1996)
21. Grünbaum, B.: Convex Polytopes. New York, NY (1967)
22. Gryaditskaya, Y., Sypesteyn, M., Hoftijzer, J.W., Pont, S., Durand, F., Bousseau, A.: Opensketch: A richly-annotated dataset of product design sketches. ACM Transactions on Graphics (Proc. SIGGRAPH Asia) **38** (2019)
23. Hershberger, J.: Stable snap rounding. In: Proc. of twenty-seventh annual symposium on Computational geometry, pp. 197–206 (2011)
24. Hoffmann, C.M.: Geometric and Solid Modeling: An Introduction. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)
25. Lienhardt, P.: N-Dimensional generalized combinatorial maps and cellular quasi-manifolds. Inte. J. of Computational Geometry and Applications **4**(3), 275–324 (1994)
26. McKenney, M., Frye, R., Dellamano, M., Anderson, K., Harris, J.: Multi-core parallelism for plane sweep algorithms as a foundation for gis operations. GeoInformatica **21**, 151–174 (2017)
27. McKenney, M., McGuire, T.: A parallel plane sweep algorithm for multi-core systems. In: Proc. of 17th ACM SIGSPATIAL international conference on advances in geographic information systems, pp. 392–395 (2009)
28. Muller, D., Preparata, F.: Finding the intersection of two convex polyhedra. Theoretical Computer Science **7**(2), 217 – 236 (1978)
29. Pion, S., Fabri, A.: A Generic Lazy Evaluation Scheme for Exact Geometric Computations. Science of Computer Programming **76**(4), 307–323 (2011)
30. Rossignac, J.: 3D compression made simple: Edgebreaker with zipandwrap on a corner-table. In: Proc. of International Conference on Shape Modeling and Applications, pp. 278–283 (2001)
31. Shewchuk, J.R.: Robust adaptive floating-point geometric predicates. In: Proc. of Twelfth Annual Symposium on Computational Geometry, SCG '96, p. 141–150. Association for Computing Machinery, New York, NY, USA (1996)
32. Sieger, D., Botsch, M.: Design, implementation, and evaluation of the surface_mesh data structure. In: W.R. Quadros (ed.) Proc. of 20th International Meshing Roundtable, pp. 533–550. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
33. The CGAL Project: CGAL User and Reference Manual, 5.0.1 edn. CGAL Editorial Board (2020). URL https://doc.cgal.org/5.0.1/Manual/packages.html
34. Weiler, K.: Edge-based data structures for solid modelling in curved-surface environments. Computer Graphics and Applications **5**(1), 21–40 (1985)
35. Wein, R., Berberich, E., Fogel, E., Halperin, D., Hemmer, M., Salzman, O., Zukerman, B.: 2D arrangements. In: CGAL User and Reference Manual, 5.0.1 edn. CGAL Editorial Board (2020). URL https://doc.cgal.org/5.0.1/Manual/packages.html#PkgArrangementOnSurface2
36. Wolff, M.: Heaptrack: A heap memory profiler for linux (2017). URL https://github.com/KDE/heaptrack