SDCEL

Scalable Overlay Operations over DCEL Polygon Layers

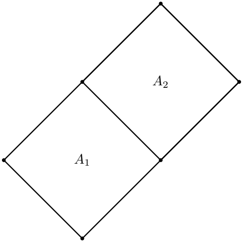
Andres Calderon · acald013@ucr.edu
Amr Magdy · amr@cs.ucr.edu
Vassilis J. Tsotras · tsotras@cs.ucr.edu

University of California, Riverside

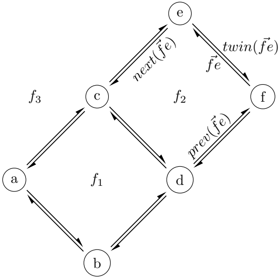
August 24, 2023

What is a DCEL?

- Doubly Connected Edge List - DCEL.
- A spatial data structure collecting topological and geometric information for vertices, edges and faces contained by a surface in the plane.
- Widely used to support polygon triangulation and its applications (art gallery problem, robot motion planing, circuit board printing, etc).



Planar subdivision



DCEL representation

- DCEL uses three tables: Vertices, Faces and Half-edges.

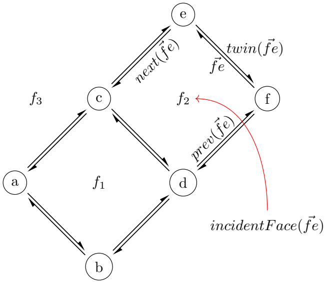


Table 1: **Vertex records.**

vertex	coordinates	incident edge
a	(0,2)	$\vec{ba}$
b	(2,0)	$\vec{db}$
⋮	⋮	⋮
e	(4,6)	$\vec{fe}$
f	(6,4)	$\vec{df}$
⋮	⋮	⋮

Table 2: **Face records.**

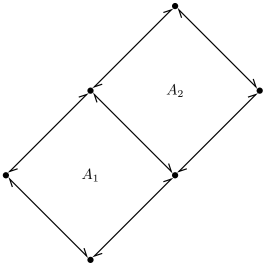
face	boundary edge	hole list
f_1	$\vec{ab}$	*nil*
f_2	$\vec{fe}$	*nil*
f_3	*nil*	*nil*

Table 3: **Half-edge records.**

half-edge	origin	face	twin	next	prev
$\vec{fe}$	f	f_2	$\vec{ef}$	$\vec{ec}$	$\vec{df}$
$\vec{ca}$	c	f_1	$\vec{ac}$	$\vec{ab}$	$\vec{dc}$
$\vec{db}$	d	f_3	$\vec{bd}$	$\vec{ba}$	$\vec{fd}$
⋮	⋮	⋮	⋮	⋮	⋮

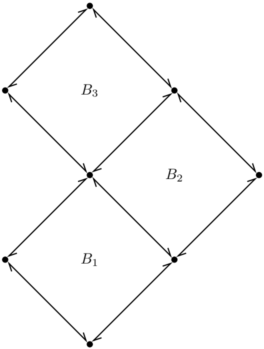
Advantages, challenges and contributions

- Very efficient for computation of *overlay operators*.
- Allows multiple operations over the same DCEL.
- The output of a DCEL operator can be input to another DCEL operator.



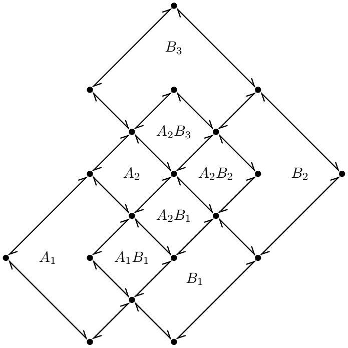
Advantages, challenges and contributions

- Very efficient for computation of *overlay operators*.
- Allows multiple operations over the same DCEL.
- The output of a DCEL operator can be input to another DCEL operator.

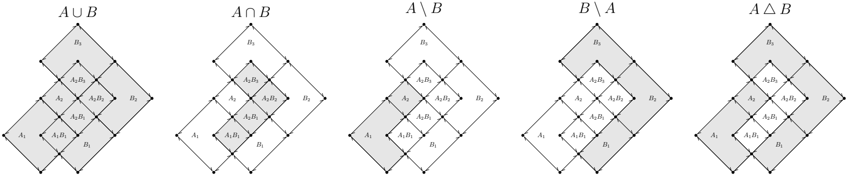


Advantages, challenges and contributions

- Very efficient for computation of *overlay operators*.
- Allows multiple operations over the same DCEL.
- The output of a DCEL operator can be input to another DCEL operator.

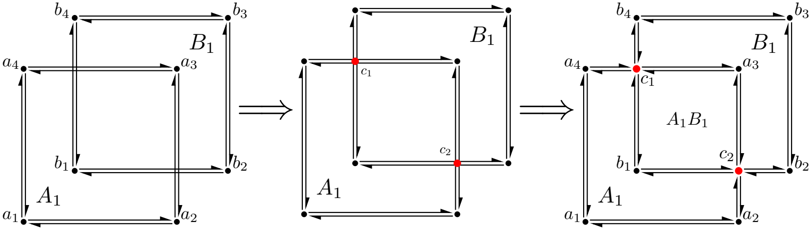


- Very efficient for computation of *overlay operators*.
- Allows multiple operations over the same DCEL.
- The output of a DCEL operator can be input to another DCEL operator.

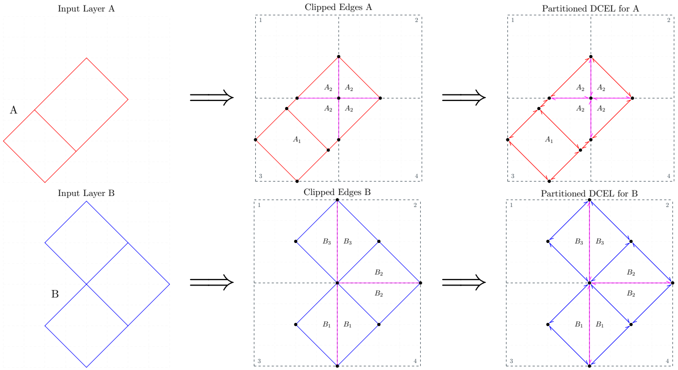


- Currently only sequential DCEL implementations exist.
- Unable to deal with large datasets (i.e. US Census tracks at national level).
- We propose a *scalable* and *distributed* approach to compute the overlay between two DCEL layers.
- Distribution enables scalability, but it also creates challenges: the ***orphan-cell*** problem and the ***orphan-hole*** problem.
- We also present *optimizations* that improve the overlay computation performance.

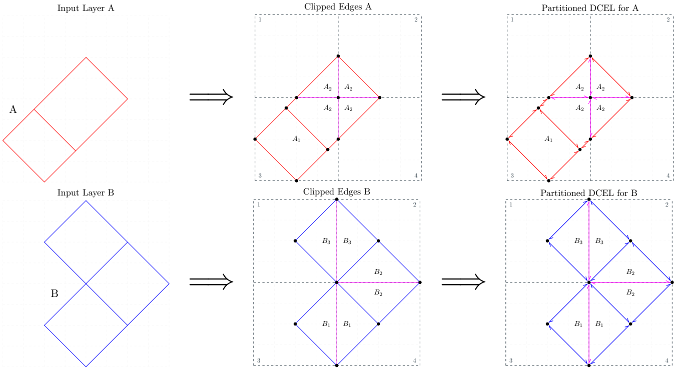
- Consider two (simple) input DCELs A_1 and B_1. The sequential algorithm first finds the intersections of half-edges.
- Then, new vertices (e.g. c_1, c_2) are created, half-edges are updated, new faces are added and labeled (e.g. A_1B_1).



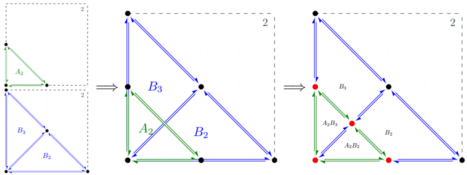
- Partition Strategy: we partition the space into cells using a spatial index (e.g. quadtree)
- Each input DCEL layer (e.g. A, B) is partitioned using the index

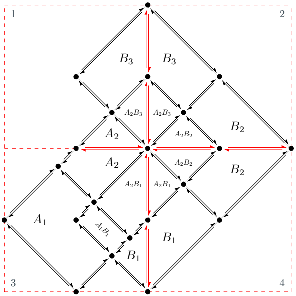


- Each cell should contain all information needed so that it can compute the overlay DCEL locally
- For each cell to be independent, we need to create "artificial" edges and vertices

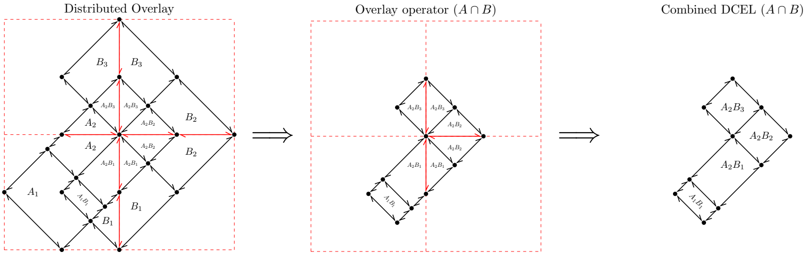


Distributed DCEL construction

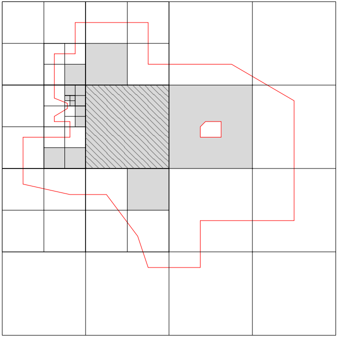




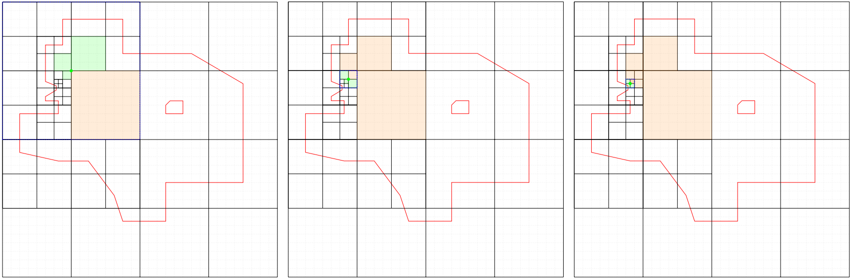
- Answering global overlay queries...
  - To compute a particular overlay operator, we query local DCELs.
  - This work is done independently at each cell (node).
  - SDCEL then collects back all local DCEL answers and computes the final answer (by removing artificial edges and concatenating the resulting faces).



- We next discuss the **orphan cell** problem (orphan holes are handled similarly).
- A large face (e.g. the red polygon in the figure) can contain cells that do not intersect with any of the face's boundary edges (called *regular edges*).
- Such cells do not contain any label and thus we do not know which face they belong to.



- We provide an algorithm to efficiently solve the orphan cell problem.



Labeling orphan cells and orphan holes

Algorithm 1: getNextCellWithEdges algorithm

```
Input: a quadtree Q and a list of cells M.
function getNextCellWithEdges(Q,M):
    C ← orphan cells in M
    foreach orphanCell in C do
        initialize cellList with orphanCell
        nextCellWithEdges ← nil
        referenceCorner ← nil
        done ← false
        while ¬done do
            c ← last cell in cellList
            cells, corner ← getCellsAtCorner(Q, c)
            foreach cell in cells do
                nedges ← get edge count of cell in M
                if nedges > 0 then
                    nextCellWithEdges ← cell
                    referenceCorner ← corner
                    done ← true
                else
                    add cell to cellList
                end
            end
        end
        foreach cell in cellList do
            output(cell,nextCellWithEdges,
             referenceCorner)
            remove cell from C
        end
    end
end
```

Algorithm 2: getCellsAtCorner algorithm

```
Input: a quadtree with cell envelopes Q and a cell c.
function getCellsAtCorner(Q, c):
    region ← quadrant region of c in c.parent
    switch region do
        case 'SW' do
            corner ← left bottom corner of c.envelope
        case 'SE' do
            corner ← right bottom corner of c.envelope
        case 'NW' do
            corner ← left upper corner of c.envelope
        case 'NE' do
            corner ← right upper corner of c.envelope
    end
    cells ← cells which intersect corner in Q
    cells ← cells − c
    cells ← sort cells on basis of their depth
    return (cells, corner)
end
```

Overlay optimizations

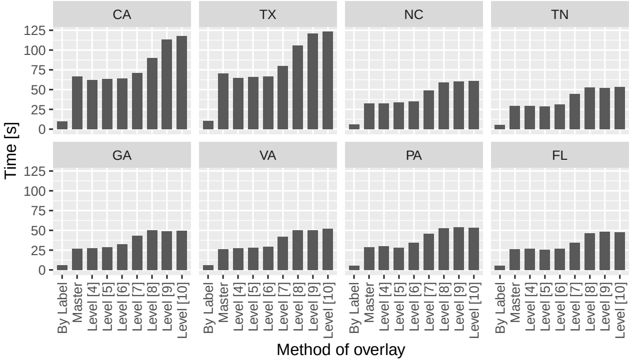
- Optimizing for faces overlapping many cells...
  - Naive approach sends all faces that overlap a cell to a master node (that will combine them).
  - We propose an intermediate reduce processing step.
    - The user provides a level in the quadtree structure and faces are evaluated at those intermediate reducers.
  - We also consider another approach that re-partitions such faces using their labels as the key.
    - It avoids the reduce phase but implies an additional shuffle.
    - However, as we show in the experiments this overhead is minimal.

- Optimizing for unbalanced layers...
  - Finding intersections is the most critical part of the overlay computation.
  - However, in many cases one of the layers has much more half-edges than the other.
  - Sweep-line algorithms to detect intersections run over all the edges.
  - Instead we scan the larger dataset only for the x-intervals where there are half-edges from the smaller dataset.

- Datasets.

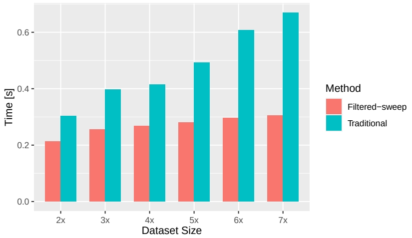
Dataset	Layer	Number of polygons	Number of edges
MainUS	Polygons for 2000	64983	35417146
	Polygons for 2010	72521	36764043
GADM	Polygons for Level 2	116995	32789444
	Polygons for Level 3	117891	37690256
CCT	Polygons for 2000	7028	2711639
	Polygons for 2010	8047	2917450

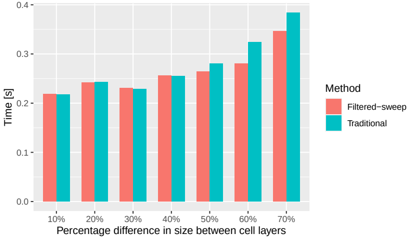
- Evaluation of the overlapping faces optimization.



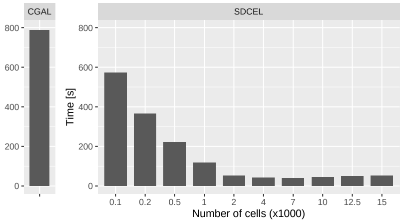
Experimental evaluation

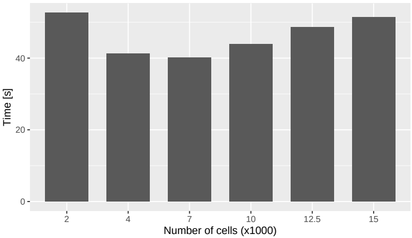
- Evaluation of the unbalanced layers optimization.



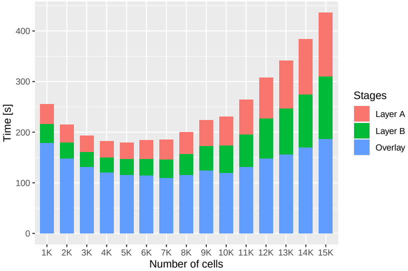


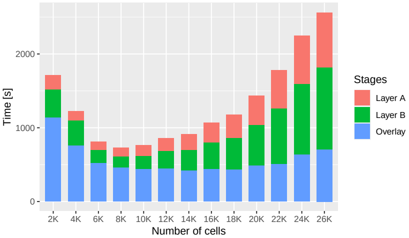
- Performance varying number of partition cells (CCT dataset).



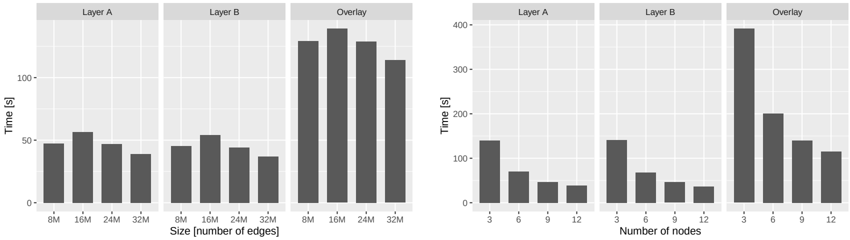


- Performance with MainUS and GADM datasets.

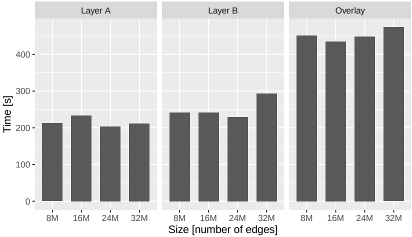


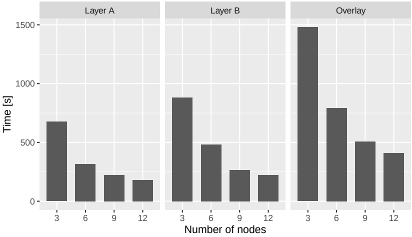


- MainUS scale-up and speed-up.



- GADM scale-up and speed-up.





- We introduced SDCEL, a scalable approach to compute the overlay operation among two layers that represent polygons from a planar subdivision of a surface.
- We use a partition strategy which guarantees that each partition (cell) has the data needed to work independently.
- We also proposed several optimizations to improve performance.
- Our experiments using real datasets show very good performance; we are able to compute overlays over very large layers (each with >35M edges) in few minutes.

Thank you!