SDCEL

Scalable Overlay Operations over DCEL Polygon Layers

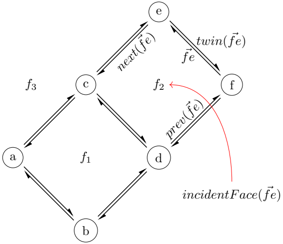
Andres Calderon · acald013@ucr.edu
Amr Magdy · amr@cs.ucr.edu
Vassilis Tsotras · tsotras@cs.ucr.edu

University of California, Riverside

August 22, 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 use to support polygon triangulation and its applications (art gallery problem, robot motion planing, circuit board printing, etc).



DCEL description

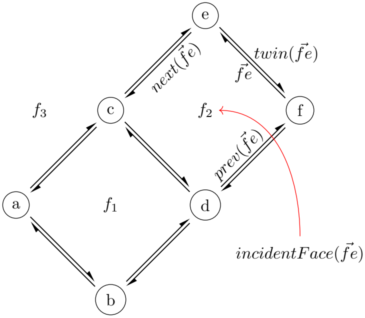


Table 1: Vertex records.

vertex	coordinates	incident edge
a	(0,2)	$\vec{ba}$
b	(2,0)	$\vec{db}$
c	(2,4)	$\vec{dc}$
$\vdots$	$\vdots$	$\vdots$

Table 2: Face records.

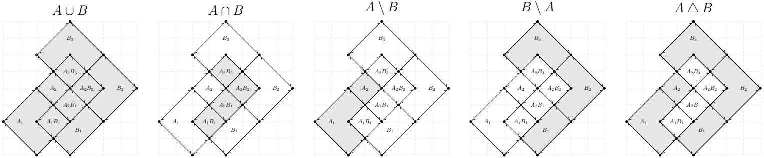
face	boundary edge	hole list
f_1	$\vec{ab}$	*nil*
f_2	$\vec{fe}$	*nil*
f_3	*nil*	*nil*

Table 3: Half-edge records.

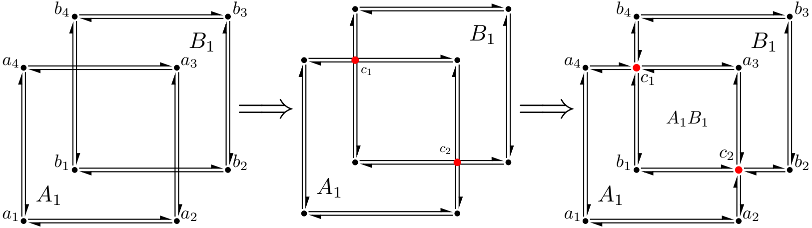
half-edge	origin	face	twin	next	prev
$\vec{fe}$	f	f_2	$\vec{ef}$	$\vec{ec}$	$\vec{df}$
$\vec{ca}$	c	f_1	$\vec{ac}$	$\vec{ab}$	$\vec{dc}$
$\vec{db}$	d	f_3	$\vec{bd}$	$\vec{ba}$	$\vec{fd}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

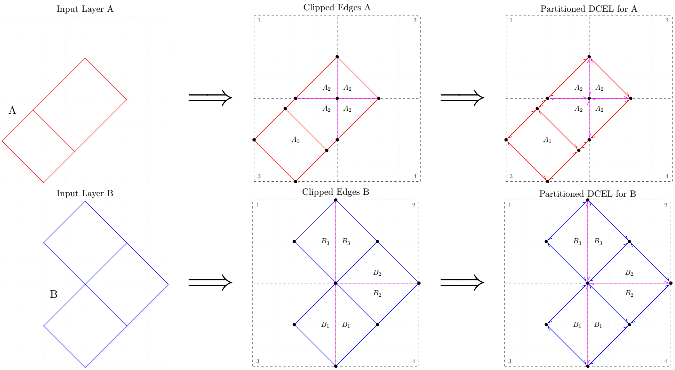
Advantages, challenges and contributions

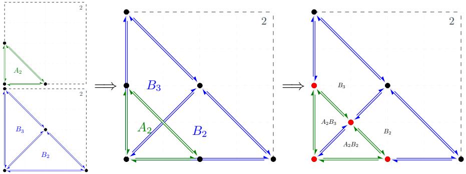
- Very efficient for computation of *overlay maps* and *overlay operators*.
- Allows multiple operations over the same DCEL and chaining.
- Currently only sequential implementations.
- Unable to deal with large datasets (i.e. US Census tracks at national level).
- We propose a scalable and distributed approach to compute the overlay between two DCEL layers.
- It solve issues related to holes and large empty areas in addition to optimizations during the overlay computations.

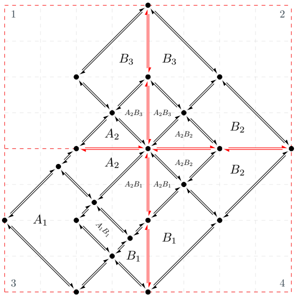


Sequential implementation

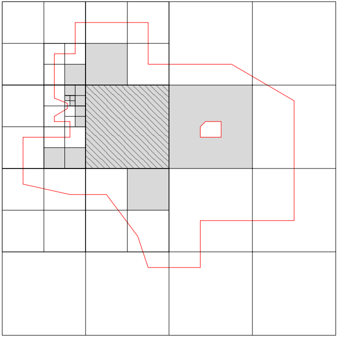



Input Layer A
A
Clipped Edges A
A_2 A_2
A_2 A_2
A_1
Partitioned DCEL for A
A_2 A_2
A_2 A_2
A_1
Input Layer B
B
Clipped Edges B
B_3 B_3
B_2
B_2
B_1 B_1
Partitioned DCEL for B
B_3 B_3
B_2
B_2
B_1 B_1
1 2 3 4

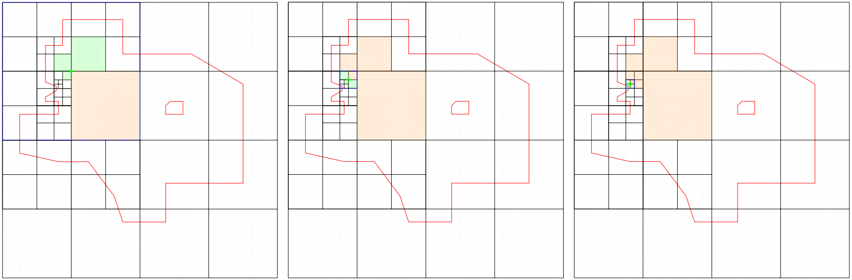

2
A_2
2
B_3
B_2
$\Longrightarrow$
2
B_3
A_2
B_2
$\Longrightarrow$
2
B_3
A_2B_3
B_2
A_2B_2


1
2
B_3
B_3
A_2B_3
A_2B_3
A_2
A_2B_2
B_2
A_2
A_2B_2
B_2
A_2B_1
A_2B_1
A_1
A_1B_1
B_1
B_1
3
4

- The **orphan cell** problem.
- When a cell is empty (it does not intersect or contain a *regular edge*).
- A regular edge is one that is not part of a hole.
- The label for the cell (or a hole if it is present) is lost. Thus we do not know which face it belongs to.
- We provide algorithms to solve the issue.



Labeling orphan cells and holes



Labeling orphan cells and holes

Algorithm 1: getNextCellWithEdges algorithm

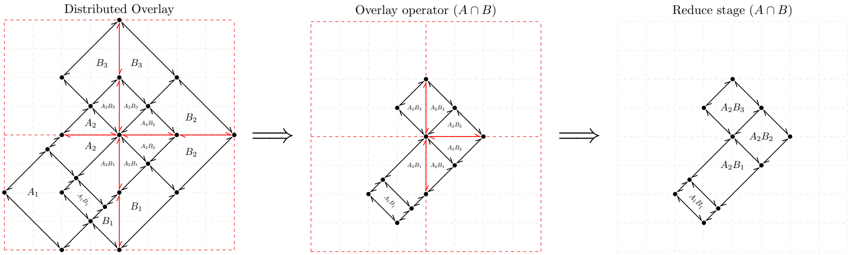
```
Input: a quadtree Q and a list of cells M.
function getNextCellWithEdges(Q,M):
    C ← orphan cells in M
    foreach orphanCell in C do
        initialize cellList with orphanCell
        nextCellWithEdges ← nil
        referenceCorner ← nil
        done ← false
        while ¬done do
            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 evaluation

- Answering global overlay queries...
  - We query local DCEL's for particular overlay operators.
  - It filters out at each worker.
  - SDCEL collects back and reduce the final answer (remove artificial edges and concatenate splits).



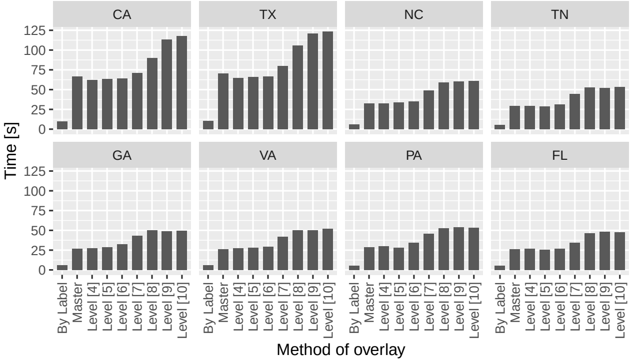
- Optimizations for faces expanding cells...
  - Naive approach send all faces to a master node.
  - We propose an intermediate reduce processing step.
  - The user provides a level in the quadtree structure and faces are evaluated at those intermediate reducers.
  - Another approach re-partitions the faces using its labels as the key.
  - It avoids the reduce phase but implies an additional shuffle.
  - However, SDCEL works with much smaller independent amounts of work.

- Optimizing for unbalanced layers...
  - Finding intersections is the most critical part of the overlay computation.
  - However, in many cases one of the layer has much more half-edges than the other.
  - Sweep-line algorithms to detect intersections runs over all the edges.
  - We scan the larger dataset only for the x-intervals where there are half-edges for the smaller dataset.
  - It avoids unnecessary scanning where there are few edge (or even none) from one of the layers.

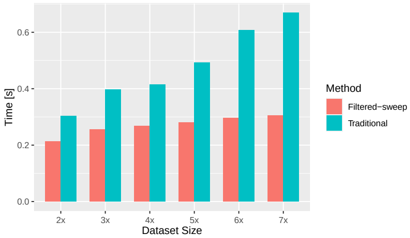
- Datasets.

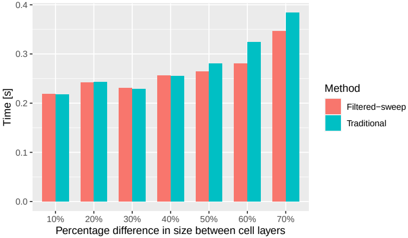
Dataset	Layer	Number of polygons	Number of edges
MainUS	Polygons for 2000	64983	35417146
	Polygons for 2010	72521	36764043
GADM	Polygons for Level 2	116995	32789444
	Polygons for Level 3	117891	37690256
CCT	Polygons for 2000	7028	2711639
	Polygons for 2010	8047	2917450

- Evaluation of overlay methods.

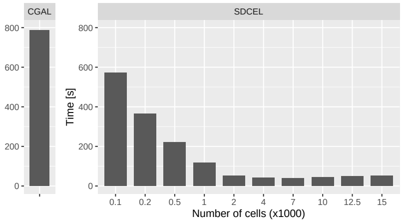


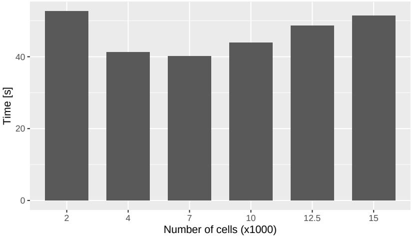
- Unbalance layers optimization.



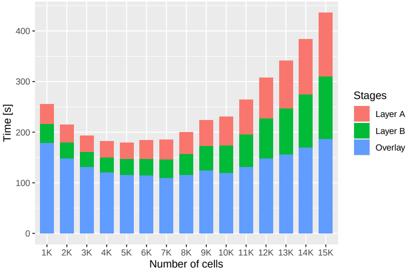


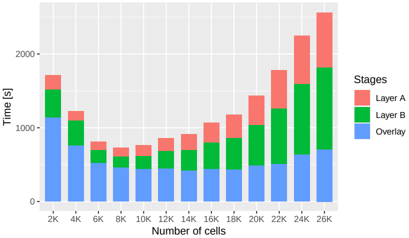
- Performance varying number of cells (CCT dataset).





- Performance with MainUS and GADM datasets.





- 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 first presented a partition strategy which guarantee each partition has the needed data to work independently.
- We also proposed several optimization to improve performance and ensure correct results.
- Our experiments in real datasets show very good performance and it is able to compute overlays over very large layers in few minutes.