Rectilinear Polygon Operations for Phsical Design

Yu-Cheng Lin
Dept. Electrical Engineering
National Taiwan University
Taipei, Taiwan
b05502145@ntu.edu.tw

Yu-Tsung Wu
Dept. Electrical Engineering
National Taiwan University
Taipei, Taiwan
b05901053@ntu.edu.tw

***Abstract*— In this paper we proposed a new rectilinear polygon merging and clipping algorithm which is modified by Weiler-Atherson's algorithm to solve more complex conditions. Also, we proposed an interval based algorithm to split the polygon in horizontal or vertical direction. Last, we implement the polynomial time algorithm, Hopcroft-Karp algorithm, on minimum polygon splitting.**

Keywords—Merging, Clipping, Minimum polygon splitting, Weiler-Atherson Algorithm, Hopcroft-Karp Algorithm

I. INTRODUCTION

Rectilinear polygon processing is a very common problem in Physical Design because there are many elements in the physical design flow that are represented by rectilinear polygons, such as macro blocks, cell pin shape, standard cell row regions, floorplan channels, ... and so on.

Therefore, in the physical design stage, it is often necessary to perform the merging, clipping, splitting, etc. processing on the rectilinear polygons, and obtain the results for further analysis or application. For example, "merge" operation is used to merge multiple rectilinear polygons connected to handle repeated descriptions of overlaps and reduce the number of rectilinear polygons to improve program execution efficiency. "Clip" operation is used to delete some special areas, and "split" operation is used to convert rectilinear polygons into rectangles to simplify the complexity of the problem.

There are some methods for the basic processing of rectilinear polygons. But the key is that when applied to physical design flow, the number of polygons is often very large, which makes the program implementation more difficult.

In section II, III and IV, we will introduce our merging/clipping, horizontal/vertical splitting and minimum polygon splitting algorithm in detail and analysis the time complexity separately. In section V, we will make the experiment on different size of numbers of polygon to see if the time-consuming is consistent with our complexity analysis. In section VI, we will make some discussion on our method and state what we can improve.

II. POLYGON MERGING AND CLIPPING

A. Algorithms

The algorithms we used is modified by Weiler-Atherton algorithm to handle complex conditions [1], such as overlapping angles or overlapping edges in Fig. 1. We utilize an intuitive concept for defining the in/out property of the intersection point. Also, under these conditions. The traversing path in Weiler's algorithm won't always be a cycle, it might not traverse back to the initial point, such as the condition shown in Fig. 2. Therefore, we apply a "crop" method to crop the path into cycle. The two difference in merge and clip is that the definition of in/out property of intersection point is not the same, and the traversing rule is a bit different.

B. Data structure

We used double-linked list structure with self-defined class point with various property, including "root", "next", "prev", "verti", "dir", "isclip", "angle", "pcolor", "iscolored", "has_intersect", which records the root of the polygon, the next or previous point in the linked-list, whether the edge is vertical, whether the edge is in positive direction, whether it is a point of the clipping polygon, what kind of angle is it, the color utilized in traversing and "check list", and determine if the polygon has intersect to delete in "check list".

C. Complexity analysis

The complexity of this algorithm is dominant by "Find intersect" and "Check list" algorithm. Both with complexity $O(n^2)$. This is because "Find intersect" function finds between any two polygons in worst case, and the check list function is $O(n)$ and we need to check n times in the whole program, where n represents the number of the polygon in the whole input file. Also, this dominant the whole program, we can clearly see this in section V.

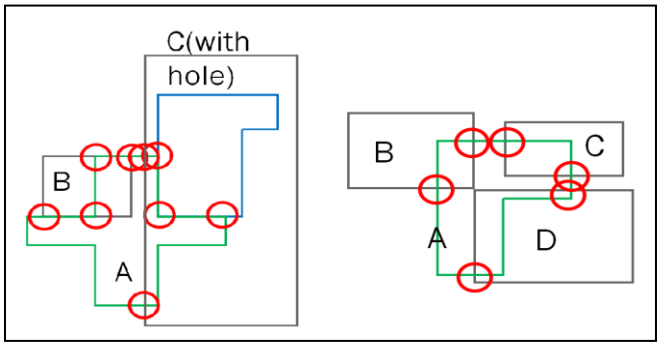

Fig. 1. Complex cases for merging or clipping

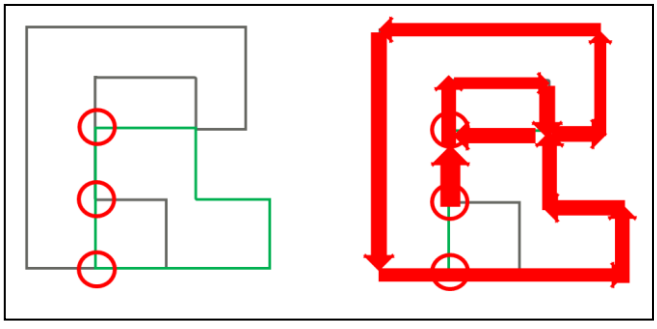

Fig. 2. Wrong traversing in Weiler-Atherson

D. Details

After reading the whole input file and construct all the polygon into double-linked list, we will have an Null vector called root list, denoted as R_t. For every polygon P_i, we insert it in to R_t and make operation merge or clip. The merge and clip operation only differs in the in/out property of the intersection point.

First, we will find all the intersections and determine its property between R_t and P_i. Also, in this process we can know that whether a polygon in R_t is inside of P_i, if it is inside P_i, we will delete the polygon and erase it from R_t. Then, we will connect intersect point into the double-linked list.

Next, we will randomly choose an out(in) point on polygon P_i and start traversing through the linked list for merging(clipping) operation. For merging, when traversing the linked list, if we walk on an intersection point, we cross to the other polygon's linked list if the property of the crossing intersect point is "out". For clipping, if we walk on a clipping, we need to see if we were walking on P_i, if it is, we cross to the other polygon's linked list if the property of the intersect point on P_i is "out"; if it is not, it cross to the other(P_j) polygon if the intersection on P_j ""is "in". This process will end if we traverse to the original point or a traversed point. Then, we will remove the dummy point in the path so that the path forms a cycle.

Next, we remove repeat points in this cycle, and also remove some points to allow the path to alternate between vertical and horizontal edge.

Last, the "Check list" function will check which polygon in R_t have been traversed in the "Walk" function and delete it. Then, we concatenate the new list and R_t. We repeat this procedure until all polygon are merged or clipped.

III. Horizontal or Vertical Splitting

A. Algorithms

First, we will build our interval class depend on the type of splitting. Then, we will sort the interval according to its coordinate. Last, we will see through the sorted interval and split the rectangle.

B. Data structure

We used a self-defined data structure: Interval, which contains member, "_left", "_right", "_position", "_reverse",

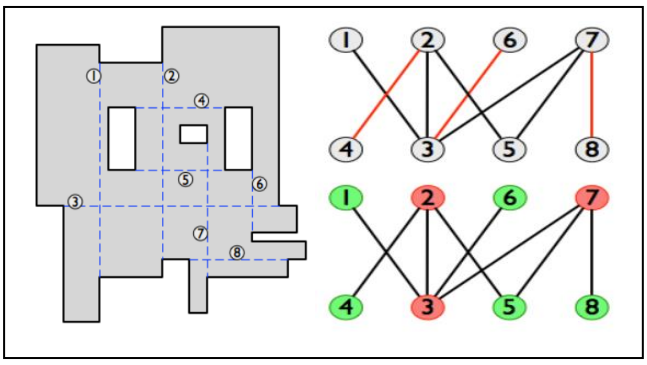

Fig. 3. An example for out mimimum rectilinear polygon splitting algorithm

each represents the coordinate of x(y) of end point of the edge, the y(x) coordinate of the edge and the direction of the edge if the splitting operation is Horizontal(Vertical).

First, we will build our interval class depend on the type of splitting. Then, we will sort the interval according to its coordinate. Last, we will see through the sorted interval and split the rectangle.

C. Complexity analysis

The complexity of splitting is dominant by the "find_intersections" function, which is related to the square of number of intervals. Since number of intervals is either number of horizontal edge or veritcal edge, the complexity can also be represent as $O(s^2)$, where s is the average number of vertices of the polygon.

D. Details

Since Horizontal and Vertical split are symmetric operations. We only demonstrate Horizontal split here.

First, we will get the reference of the vector of double-linked list after all merging and clipping operation. We traverse through the whole polygon, find the horizontal edge and build the Interval class with member LX, RX, Y, R. LX records the x-coordinate of the left side of the edge, RX records the x-coordinate of the right side of the edge, Y records the y-coordinate of the edge, R records the reverse property of the edge, where reverse is true for large to small x-coordinate. These Interval are store in a vector.

Second, we will sort these Interval by its y-coordinate in ascending order.

Last, we will make the "Union" or "Difference" operation on the sorted Interval. Initially, we will have a Null Interval called total Interval, denoted as I_t, if we get an Interval I with false reverse, we will "Union" these intervals, which means that the coordinate of x will be union, denoted as $I_t \leftarrow I_t \cup I$. Otherwise, we will "Difference" the Interval, denoted as $I_t \leftarrow I_t - I$. At each operation, we will check if the Interval I has intersected with the total Interval I_t, if it does, we will split a rectangle with bottom-left corner with coordinate($I_{t1}.LX, I_{t1}.Y$), and top-right corner with coordinate($I_{t1}.RX, I.Y$).

IV. Minimum Polygon Splitting

A. Algorithms

David Eppstein has proposed a method to spilt rectilinear polygon into minimum rectangles[2]. That is finding the maximum number of disjoint axis-parallel diagonals that have two concave vertices as endpoints, these diagonals are called "good diagonals", the diagonals dotted line in the left figure in Fig. 3 are all good diagonals.

We represent the problem by a graph data structure, and translate the problem in to finding the maximum independent set of the graph. Since, the diagonal only have intersects if one is vertical and the other is horizontal, so the graph is bipartite. Therefore, according to Konig's theorem this problem can be translate into a maximum matching problem on bipartite graph[3].

Maximum bipartite matching can be solved with Max-flow algorithm, but we choose to implement Hopcroft-Karp algorithm for greater performance[4].

B. Data structure

Each good diagonal is representing by a node, and the structure of the graph is stored by adjacency list. We also use queue for BFS in Hopcroft-Karp algorithm.

C. Complexity

The complexity is hard to calculate in this algorithm, since the number of "good diagonal" is hard to represent. Let *s* be the average number of vertices of the polygon. In worst case, the number of "good diagonal" may be $O(s)$. So, the Hopcroft-Karp will have complexity $O(s^{2.5})$. The translating between maximum independent set and maximum matching is $O(s)$. Additionally, the complexity of splitting the rectangle horizontally is $O(s^2)$. Therefore, we can get a very loose bound for this algorithm with complexity $O(s^{2.5})$.

D. Details

First we sorted the list of points in the polygon. Then, for every pair of points that have same x or y coordinates, we check if it is a good diagonal and save it. Next, we will apply the Hopcroft-Karp with BFS and DFS to find maximum matching in complexity of power of 2.5 of number of nodes. Last, by Konig's theorem, we will construct a node set K.

Let T and D represent the top and down side node, respectively. Let U be the set of the unmatched nodes in T. Let N be the set of node, which is connected to U by alternating path, which means every edge in the path should be alternating between matched edge and unmatched edge. K is constructed by union T/U and D ∩ U. Finally, we will get the minimum independent set K' by forming the complementary set of K. For example, refer to the right figure in Fig. 3, the bipartite graph has been construct by determine whether the good diagonal has intersections. After running the Hopcroft-Karp algorithm, we find the maximum matching {2, 4}, {6, 3}, {7, 8}, so the unmatched node is {1, 5} we start from node 1 and find the longest alternating path, that is {1, 3, 6}. Then we can find the complement of independent set by adding {3} from the bottom side and deleting {1, 6} on the top side to {1, 2, 6, 7}; therefore, we get the set {2, 3, 7} which is the node colored in red in Fig. 3, the other green node forms the maximum independent set.

After getting the minimum independent set, we cut the polygon into several polygons, and apply SH on every polygon to get the minimum number of rectangle.

V. Experiments and Results

A. Experiments

We make the experiment on OpenCase_2.txt provided by ICCAD Contest 2019 Taiwan to observe the time consuming in different operations. The principal operation in our methods are "Read file" for reading files and construct double-linked list, "Find Intersect" for finding out all the intersect, "Walk" for traversing through the new polygon under merging or clipping operation, "Construct poly" for constructing the new polygon after operation, "Check list" for determining whether the old polygon might be included in the new list, "Split" for splitting operation, "All" for time-consuming for the whole program. The result is shown in Table I.

Also, we plot the result of the horizontal splitting on Open case 1 in Fig. 8.

TABLE I. Time consuming for principle funcitons(in secs)

Functions	**Sizes of numbers of polygon**				
	100769	***200769***	***500769***	***656514***	***756514***
Read file	6	6	6	7	6
Find intersect	77	436	2721	5156	6,024
Walk	0	5	26	61	57
Construct poly	18	42	223	598	556
Check list	44	220	1719	3086	4,323
Split	51	143	1024	651	486
All	199	855	5727	9571	11,475

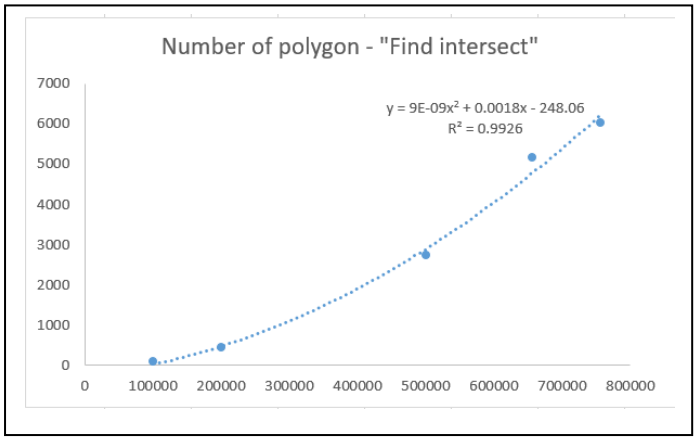

Fig. 4. Plot of number of polygon to time consuming of "Find intersct" funciton

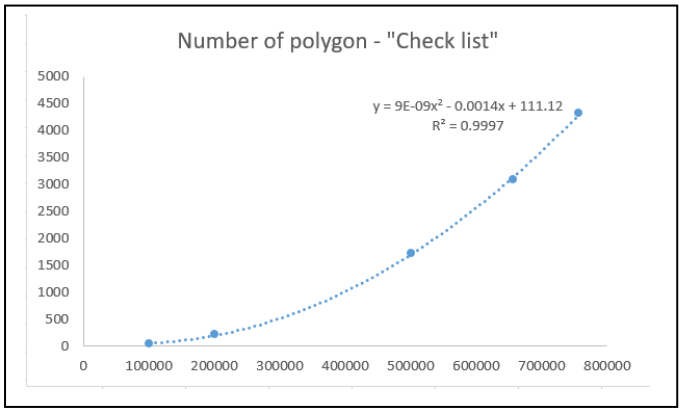

Fig. 5. Plot of number of polygon to time consuming of "Check list" funciton

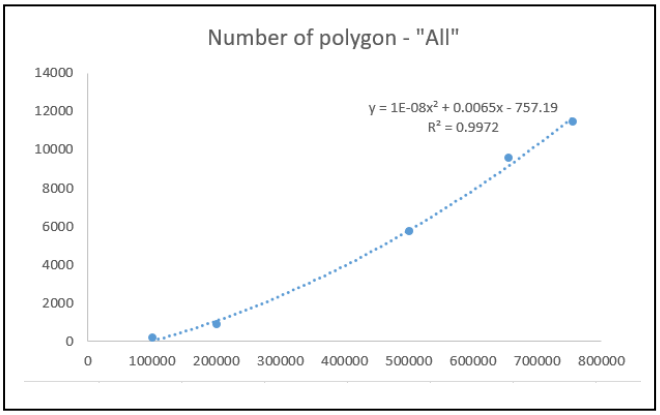

Fig. 6. Plot of number of polygon to time consuming of "All" funciton

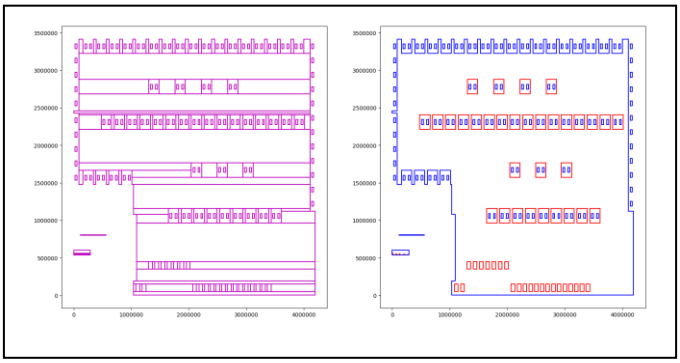

Fig. 7. Result of splitting horizontal in OpenCase_1

B. *Results and discussion*

From Fig. 4, 5 we can see that "Find intersect" and "Check list" function is $O(n^2)$ with high correlation. Also, as we have state in II, merging and clipping will dominate the whole program, so the overall runtime is also highly related to the square of number of polygon.

VI. Conclusion

In this paper, we modified the Weiler-Atherson algorithm to deal with complex conditions. Though the double-linked list data structure can be very efficient in memory, it is very inefficient in time due to finding the intersection. Also, we proposed an easy Interval-based method to split vertical or horizontal quickly. Last, we implement David Eppstein's method to split minimum polygon in polynomial time.

VII. Contribution

A. *Yu-Cheng Lin:*

Functions	Details
Read file	Read input filename and construct double linked list
Outside polygon	Check if a point is inside a polygon
Merge/Clip	Traversing the new polygon
Check list	Check if the old polygon need to be included in the new list
Check point	Check the in/out property of the intersection
New polygon	Construct new polygon by the point collected when traversing and delete dummy points
Split vertical/Horizontal	All split vertical/horizontal functions

B. *Yu-TSung Wu: Merge/Clip(Find intersect, Find cross, Insert intersect, New intersect, List construct), Split Minimum*

Functions	Details
Find intersect	Find the all intersects between two polygons
Find cross	Check two edge has intersection point
Insert intersect	Sort and insert intersect point after one point
New intersect	Construct two new intersection points and add each intersection points to the edge it belongs, connect two intersection points.
List construct	Construct edge property including the vertical, angle, direction. Specify the root, region of the polygon.
Split Minimum	Find max matching and min vertex cover and form maximum independent set.

References

[1] Weiler, Kevin and Atherton, Peter. "Hidden Surface Removal using Polygon Area Sorting", Computer Graphics, 11(2):214-222, 1977.

[2] David Eppstein, "Graph Theoretic Solutions to Computational Geometry Problems".

[3] Kőnig, Dénes, "Gráfok és alkalmazásuk a determinánsok és a halmazok elméletére", *Matematikai és Természettudományi Értesítő, 34: 104–119.*

[4] Hopcroft, John E.; Karp, Richard M., "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", SIAM Journal on Computing, 2 (4): 225–231, doi 10.1137/0202019.