Ceylon Journal of Science 47(2) 2018: 165-174
DOI: http://doi.org/10.4038/cjs.v47i2.7513

RESEARCH ARTICLE

On the convex layers of a planer dynamic set of points

K. R. Wijeweera[1, 3, *] and S. R. Kodituwakku[2, 3]

[1]*Department of Computer Science, Faculty of Science, University of Ruhuna, Sri Lanka*
[2]*Department of Statistics and Computer Science, Faculty of Science, University of Peradeniya, Sri Lanka*
[3]*Postgraduate Institute of Science, University of Peradeniya, Sri Lanka*

Received:12/12/2017; Accepted:12/05/2018

Abstract: The convex hull of a planer set of points can be defined as the set of vertices of the smallest convex polygon containing all the points. If S is a planer set of points then convex layers of S can be derived by iteratively computing the convex hull of S and then removing it from S. Computation of the convex layers has widely been studied in the static environment where the set of points is fixed. The theoretical lower bound of computing the convex layers of a fixed set of points is O (n log n). The static convex layers algorithms with the optimal time complexity have already been proposed in the literature. A set of points where the points can be inserted or deleted is called a dynamic set of points. The set of convex layers should be reconstructed from the scratch at each insertion or deletion if a static convex layers algorithm was used to handle a dynamic set of points. Therefore, it takes O (n log n) time to handle an insertion or a deletion even for an optimal static convex layers algorithm. A dynamic convex layers algorithm has been proposed recently that can perform an insertion or a deletion by doing a slight modification to the existing set of convex layers. It takes O (n) time for an insertion or a deletion. It assumes that the set of points does not contain collinear points and that assumption is not valid in practical applications. Furthermore, the notion of tangent used in that approach restricts the extension of the algorithm into higher dimensions. This paper proposes a novel dynamic convex layers algorithm to eliminate the drawbacks in the existing algorithm. Salient feature of this algorithm is that it represents each layer as a set of line segments. The layers are modified upon an insertion or a deletion of a point by adding some new line segments and deleting some existing line segments. The proposed algorithm takes O (n^3/k^2) time for an insertion or a deletion where k is the number of convex layers. A computer implementation is also presented. The proposed algorithm can work with set of points with collinear points and coincident points. The notion used in the algorithm can easily be extended to higher dimensions. Suppose the set of convex layers have already been found for a given set of points. Further, suppose that the layers are close to each other and new points are expected to fall within the region bounded by the outermost layer. This kind of situations widely occurs in practice and then the proposed algorithm takes O (n) time for an inclusion or a deletion.

Keywords: Computational Geometry, Convex Hull, Convex Layers, Robust Statistics, Data Structures.

INTRODUCTION

The convex hull of a planer set of points can be defined as the set of vertices of the smallest convex polygon containing all the points. The convex hull is one of the most important structures in computational geometry. In wide variety of situations, it is used as a basic tool to construct other structures. Also it is an amazing object in the field of pure mathematics (O'Rourke, 1997).

The convex layers can be seen as a natural extension of the idea of the convex hull. Let S be the set of points. The set of convex layers of S is denoted by the notation C(S). The set of convex layers is derived by applying following procedure iteratively on S: compute the convex hull of S and remove its vertices from S (Chazelle, 1985). An example for convex layers of a given set of points is shown in Figure 1.

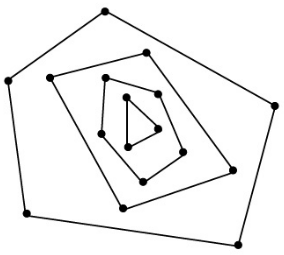

Figure 1: Convex layers of a point set.

The concept of convex layers has many applications in pattern recognition and statistics (Shamos, 1978). Evaluation of an unbiased estimator which is not too sensitive to outliers is a major problem in robust estimation. Here, the observations lying abnormally far from others should be identified. To deal with the two dimensional version of this problem, a method has been suggested that removes the outliers of a point set by peeling or shelling the set. The peeling process is iterated until only a prescribed fraction of the original points remain (Huber, 1972). In well-known retrieval problem, the idea of convex layers can be applied effectively. The half plane range search problem requires preprocessing a set of points in a plane so that for any query line L, the subset of points lying on a given side of L can be reported efficiently. An optimal

**Corresponding Author's Email:* krw19870829@gmail.com
http://orcid.org/0000-0002-8933-1687

solution to this problem has been proposed by the use of convex layers (Chazelle *et al.*, 1985).

The convex layers problem can be divided into two versions as static and dynamic. Constructing the convex layers for a set of points where all the points are available from the beginning is called static convex layers problem. The dynamic convex layers problem means the maintenance of convex layers of a set of points where points may be inserted or deleted.

A brute force approach can be used to find the set of convex layers of a given set of points using a suitable convex hull algorithm. This approach peels off a set of points in one layer at a time. The time complexity of the most efficient convex hull algorithm is O (n log n). Suppose there are k layers in the dataset. Therefore, it is possible to find convex layers using this approach in O (kn log n) time (Rufai *et al.*, 2017).

There are two main methods used by existing algorithms. The peeling process used in brute force approach is one of the widely used methods. The other general method is called plane sweep.

The algorithm proposed by Green and Silverman (Green *et al.*, 1979) is one of the earliest algorithms. Their algorithm uses quick hull algorithm to compute convex hull at each invocation. This algorithm takes O (n^3) time in worst case. But O (n^2 log n) time is expected in practice.

Overmars and Leeuwen (Overmars *et al.*, 1981) proposed an algorithm with O (n $\log^2$ n) time. A fully dynamic data structure is used to maintain the convex hull under arbitrary insertions and deletions of points. It takes O ($\log^2$ n) time for each update since construction of the convex layers can be reduced to insertion of all points into the data structure in O (n $\log^2$ n) time, marking points on the current convex hull, deleting them, and then repeating the process for next layer. It takes no more than O (n $\log^2$ n) time to these steps since each point is marked exactly once and deleted exactly once.

Chazelle (Chazelle, 1985) proposed an algorithm with O (n log n) time and O (n) space using balanced tree approach. Chazelle's algorithm achieved optimal time and space for the problem. However, this algorithm is extremely complicated and difficult to apply in practice (Rufai *et al.*, 2017).

Shamos (Preparata *et al.*, 1985) proposed an algorithm using plane sweep paradigm for the first time. This algorithm is a modification of the convex hull algorithm proposed by Jarvis and March. This algorithm performs a radial sweep changing the pivot along the way as in the Jarvis March algorithm. But it does not stop after processing all the points. It continues with another round excluding the points found in previous iteration. Thus, it takes O (n^2) time to find all convex layers.

Another optimal algorithm was proposed by Nielsen (Nielsen, 1996) using the grouping trick proposed by Chan (Chan, 1996). This algorithm is called output sensitive since its time complexity depends on the number of convex layers. The time complexity of the algorithm is O (n log H_L) where H_L is the number of points on the first L convex layers.

The convex layers algorithms mentioned above either do not achieve optimal O (n log n) time and O (n) space, or are extremely complicated and difficult to apply in practical situations. Recently, Raimi and Dana (Rufai et al., 2017) proposed a new algorithm that is both optimal and simple. Four sets of independent monotone chains are computed in O (n log n) time and O (n) space. These chains are merged in O (n log n) time.

All the convex layers algorithms discussed so far are static. That means they need the entire set of points at the beginning. Maintenance of the convex layers for a situation where points appear one by one on the plane is called incremental convex layers problem. In dynamic convex layers problem, points can be inserted or deleted and the set of convex layers should be maintained accordingly. A static convex layers algorithm can be used to handle both incremental and dynamic convex layers problems. Then the set of convex layers should be reconstructed from the scratch at each inclusion or a deletion of a point.

The optimal efficiency of static convex layers problem is O (n log n) time and O (n) space (Rufai *et al.*, 2017). Suppose there is already constructed set of convex layers for a given set of points. If a new point is inserted to the set of points then the set of convex layers also should be modified. The set of convex layers should be reconstructed for (n + 1) points. It takes log [(n + 1) log (n + 1)] = O (n log n) time and O (n + 1) = O (n) space. Similarly, if a point should be deleted then reconstruction of convex layers takes log [(n - 1) log (n - 1)] = O (n log n) time and O (n - 1) = O (n) space.

Suppose points appear one by one in the plane and convex layers should be maintained. It takes O (m log m) time and O (m) space to compute the convex layers of m points if an optimal static convex layers algorithm was used (Chazelle, 1985). Therefore, it takes O (n^2 log n) time and O (n) space to find convex layers of incremental n points.

Reconstruction of the set of convex layers from the scratch upon insertion or deletion of a single point is a waste of computational cost. The waste is significant when there are a large number of convex layers already available. Therefore, it is very important if there is an algorithm to insert or delete a point by doing only a slight modification to the existing set of convex layers. Such an algorithm is called a dynamic convex layers algorithm.

The first dynamic convex layers algorithm was proposed by Sanjib and Niraj (Sadhu *et al.*, 2015). Their algorithm requires O (n^2) time to compute convex layers in dynamic context. A single point can be inserted or deleted in O (n) time. It is the one and only algorithm available in literature to solve the dynamic convex layers problem. They assume that the set of points does not contain any collinear points. However, there are collinear points in practical datasets (O'Rourke, 1997). They proposed a theoretical algorithm and failed to provide a corresponding computer implementation. Furthermore, they use the notion of tangent to modify the convex layers. The notion of tangent is restricted to two dimensions and it is not available in higher dimensions. Therefore, the concept proposed by them cannot be extended to higher dimensions.

The algorithm proposed in this paper introduces a novel mechanism to modify convex layers instead of using tangents. The novel mechanism is applicable in any dimension. The algorithm is compatible with set of points with coincident points and collinear points. Therefore, the algorithm can be successfully used in practical applications. The proposed algorithm requires O (n^4/k^2) time to compute convex layers in dynamic context where k is the number of convex layers. A single point can be inserted or deleted in O (n^3/k^2) time. The algorithm was successfully implemented in C programming language. The implementation of the algorithm uses integer arithmetic only. Thus the precision error (Berg *et al.*, 2008) is avoided. Further, the concept of modifying a convex layer used in the algorithm can be extended to higher dimensions (Wijeweera *et al.*, 2018). That is another salient feature of the proposed algorithm which is not possessed by the existing algorithm.

A convex layer is represented by a convex polygon in the proposed algorithm. Therefore, the definitions of a polygon and a convex polygon are provided here.

A polygon can be defined as the region of a plane bounded by a finite collection of line segments forming a simple closed curve. Let v_0, v_1, v_2..., v_{n-1} be n points in the plane. Here and throughout the paper, all index arithmetic is mod n, conveying a cyclic ordering of the points, with v_0 following v_{n-1}, since (n - 1) + 1 ≡ n ≡ 0 (mod n). Let $e_0 = v_0v_1$, $e_1 = v_1v_2$..., $e_i = v_iv_{i+1}$..., $e_{n-1} = v_{n-1}v_0$ be n segments connecting the points. Then a polygon is bounded by these segments if and only if

1. The intersection of each pair of segments adjacent in the cyclic ordering is the single point shared between them: $e_i \cap e_{i+1} = v_{i+1}$, for all i = 0…, n – 1.
2. Non adjacent segments do not intersect: $e_i \cap e_j = Ø$, for all $j \neq i + 1$.
3. None of three consecutive vertices are collinear.

These line segments define a curve due to the fact that they are connected end to end. The curve is said to be closed since they form a cycle. Also this closed curve is simple since non adjacent segments do not intersect. The points v_i are called vertices of the polygon while the segments e_i are called its edges. According to the definition, a polygon is a closed region of a plane. If P denotes a polygon then ∂P is used to denote the boundary of the polygon. A polygon divides the plane into two mutually exclusive regions as interior and exterior. The interior region is bounded while the exterior region is unbounded (O'Rourke, 1997).

A polygon P is called a convex polygon if x in P and y in P implies that the segment xy is a subset of P. A vertex is called reflex if its internal angle is greater than π; otherwise the vertex is called convex. Note that all the internal angles of a convex polygon are convex (Wijeweera *et al.*, 2017).

METHODOLOGY

This section describes the methodology of the proposed algorithm. The pseudo code of the algorithm is available in the appendix.

Representation of a layer

The layers are numbered beginning from zero as shown in Figure 2. The very first point which arrives should always belong to the 0th layer.

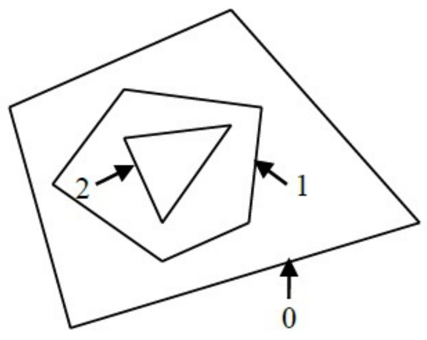

Figure 2: Numbering the layers.

A layer is represented as a set of edges. Edges are also numbered beginning from zeros in each layer. Figure 3 shows the representation of *j*th edge in *i*th layer.

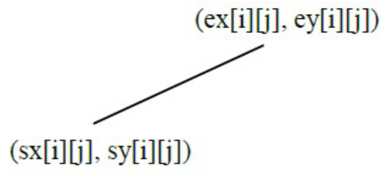

Figure 3: An edge.

A layer or an edge can have two states: dead or alive. Only alive layers and alive edges are considered as the convex layers of the point set. Those two states are stored in the *l_dead[i]* and *e_dead[i][j]* variables. These variables are of Boolean type. Being *l_dead[i]* equals to one means *i*th layer is dead. That also implies corresponding edges in that layer are also dead even though *e_dead[i][j]* values may not have set to one in that layer. Similarly *e_dead[i][j]* equals to one means *j*th edge of the *i*th layer is dead. These 'dead' variables are initialized to zero when an edge or a layer is created for the first time. In a particular layer, there may be both dead and alive edges available as discussed later. Once a layer or an edge is dead, they no longer participate in computations.

Insertion of a point

This subsection describes how a point can be inserted. The deletions are done from the existing points of the set. There should be at least one point in the set in order to delete.

Construction of the primary hull

Each layer begins with a single point and later they may evolve to a convex polygon. When a layer is a triangle, it is called the "primary hull". The very first point should be included to the 0th layer. Therefore, it is set as the *(sx[0][0],* *sy[0][0])*. Next point may coincide with the first point, if so, it is set as the *(sx[1][0], sy[1][0])*. If the next point also coincide with the first point then it is set as *(sx[2][0], sy[2][0])*. In this way, coincident points are propagated to inner layers in order to avoid coincident points in a particular layer.

The point which is distinct to the first point for the first time is set as *(ex[0][0], ey[0][0])*. This marks the first edge of the 0th layer. Then the next point can have two states as being collinear with the 0th edge of the 0th layer or not.

Suppose P_1 (x_1, y_1), P_2 (x_2, y_2) and P_3 (x_3, y_3) be three points on the plane. The area formed by (P_1, P_2, P_3) triangle is zero if and only if P_1, P_2 and P_3 are collinear. The area can be calculated using discrete version of Green's theorem as follows (Green, 1991):

$A = 0.5 * | x_1 * (y_2 - y_3) + x_2 * (y_3 - y_1) + x_3 * (y_1 - y_2)|;$

Let $txy = x_1 * (y_2 - y_3) + x_2 * (y_3 - y_1) + x_3 * (y_1 - y_2)$. Then $txy = 0$ if and only if $A = 0$. That means $txy = 0$, if and only if P_1, P_2 and P_3 are collinear.

Collinear with the 0th edge

If the next point is collinear with the 0th edge then there are three possibilities as shown in Figure 4. The new point is shown using a black dot.

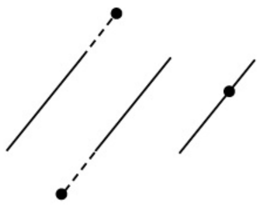

Figure 4: Positions of the point.

In first two situations shown in Figure 4, the convex layer is necessary to be extended by replacing closest end point to the new point by the new point. The collinear points are also not allowed in particular layer. Therefore, the replaced end point should be transferred to the closest inner layer of the current convex layer as a new point. In third situation shown in Figure 4, the convex layer does not need any extension. However the new point should not belong to the current layer. Instead it should be transferred to the closest inner layer as a new point since collinear points are not allowed in a particular layer.

Non-collinear with the 0th edge

If the next point is non-collinear with the 0th edge then the convex layer should be modified as shown in Figure 5. Let the coordinates of the new point be (x, y).

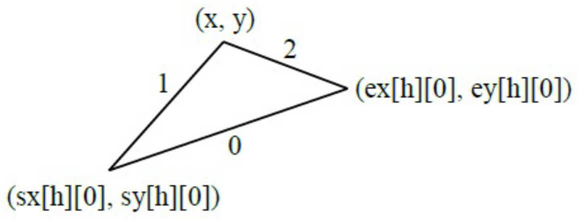

Figure 5: Non-collinear with the 0th edge.

The indices of the edges have been marked in Figure 5. Two new edges will be added as follows.

1st edge:
sx[h][1] = x;
sy[h][1] = y;
ex[h][1] = sx[h][0];
ey[h][1] = sy[h][0];

2nd edge:
sx[h][2] = x;
sy[h][2] = y;
ex[h][2] = ex[h][0];
ey[h][2] = ey[h][0];

After this was achieved, it is said that the layer *h* has become the "primary hull state".

Construction of the secondary hull

Once a layer has reached the primary hull state, it is a triangle. The mechanism used to extend the convex layer differs from this situation onwards. There should be a way to identify whether forthcoming points are inside or outside a particular layer. Let (xc/3, yc/3) be the centroid of the primary hull. Later the convex layer may grow to a polygon due to insertions of points. But (xc/3, yc/3) will always be inside the convex layer.

xc = sx[h][0] + ex[h][0] + sx[h][1];
yc = sy[h][0] + ey[h][0] + sy[h][1];

If the centroid and the new point are in two sides of a given edge of the convex layer then it is said that the new point is outside to that edge.

Identifying inside or outside to an edge

Suppose Figure 6 shows a particular situation of a convex layer from thick lines. And N appears as the new point. Here G is the centroid of the primary hull. As pointed out earlier, G should always lie inside the layer. With respect to other edges of the convex layer excluding AB, G and N are in the same side. That means N is outside the edge AB. And N is inside all the other edges except the edge AB.

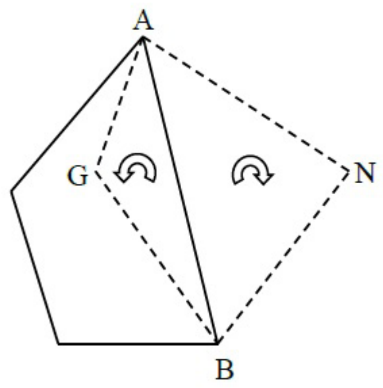

Figure 6: Inside or outside.

To detect whether two points are in the same side or not, a special method is used as shown in Figure 7. Let $A_1(x_1, y_1)$, A_2 (x_2, y_2) and A_3 (x_3, y_3) are vertices of the triangle.

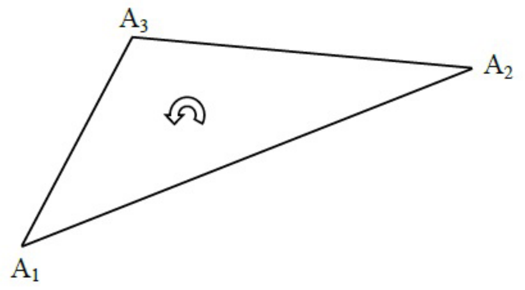

Figure 7: Sign of the expression.

The expression *txy* can be calculated as follows (See Section *Construction of the primary hull*).

$txy = x_1 * (y_2 - y_3) + x_2 * (y_3 - y_1) + x_3 * (y_1 - y_2);$

This expression is positive since it has been written in anti-clockwise direction ($A_1 \rightarrow A_2 \rightarrow A_3$) as shown in Figure 7. If this expression is written in clockwise direction ($A_3 \rightarrow A_2 \rightarrow A_1$) then it will be negative. This idea can be used to identify whether two points are in the same side or opposite sides. In Figure 6, BAG expression is in anti-clockwise direction and it is positive. But BAN expression is in clockwise direction and it is negative. If the multiplication of these two expressions is negative then G and N points are in the two sides of the edge. That means if the multiplication of two expressions is negative then it is decided that the new point is outside the corresponding edge.

The coordinates of the centroid is (xc/3, yc/3) and this can lead to the precision error because of the division operator. Therefore the entire expression should be multiplied by 3 before using it. Since multiplication is done by a positive number it does not change the sign of the expression.

Expansion a layer

Assume that the system consists of a set of convex layers as shown in Figure 8. When a new point appears, the outermost convex layer to which that point is outside should be selected first.

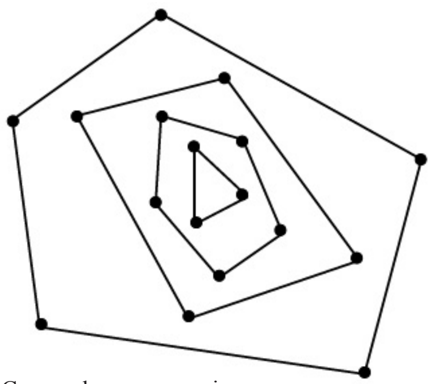

Figure 8: Convex layers expansion.

If the point is not outside none of the convex layers available then new layer should be created inside the innermost layer. If a layer got expanded then it may leave some of its existing points. Those points should be included to the adjacent inner layer of the current layer as new points. This procedure should be carried out inheriting points from outer layers to inner layers. Figure 9 shows a situation where a new point (circled one) has appeared outside the outermost layer of the set of layers shown in Figure 8.

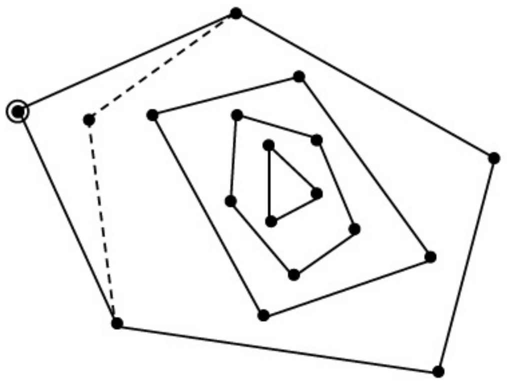

Figure 9: A new point appears.

Since outermost layer expanded, it has left one point and that point (which has two dotted edges in the Figure 9) should be transferred to the next outermost layer as shown in Figure 10.

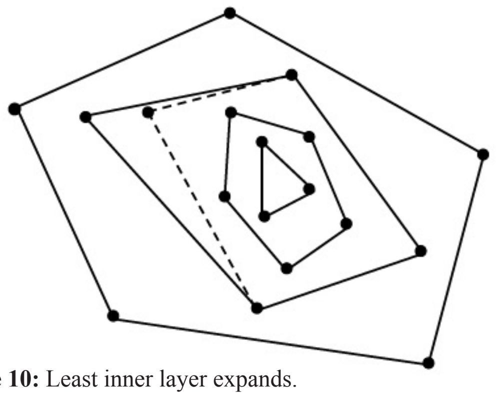

Figure 10: Least inner layer expands.

Again expansion of that layer has left another point; therefore it should also be transferred to the next outermost layer as shown in Figure 11.

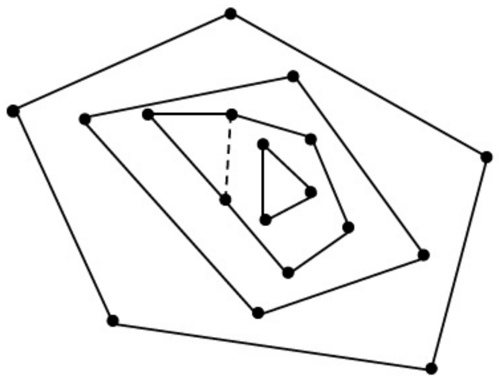

Figure 11: Least inner layer expands.

The last modified layer does not leave its existing points while the expansion. Therefore, inheritance of points concludes from that layer. And finally the modified convex layers can be seen as shown in Figure 12.

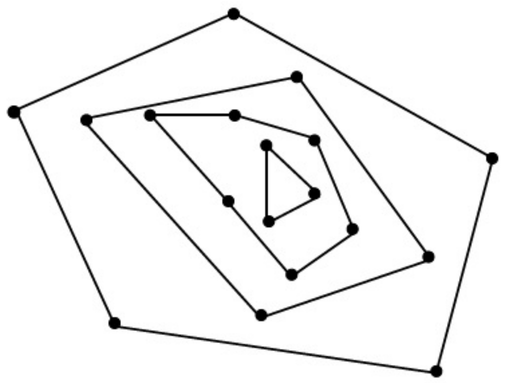

Figure 12: Final convex layers.

How to expand a layer

Expansion of a layer up to its primary hull has already been discussed. Figure 13 shows a layer in its primary hull state. Suppose a new point appears outside that convex layer as shown by a dot.

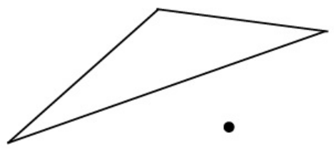

Figure 13: New point appears outside.

Then the new point is outside only to a single edge. That edge should be set into dead while two more alive edges should be added to the convex layer as shown in Figure 14.

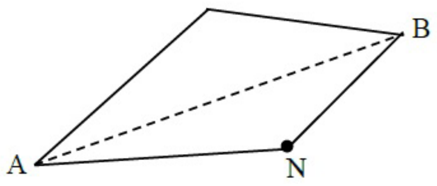

Figure 14: New point appears outside.

If A and B are the end points of an edge to which new point N is outside then AN and BN edges should be added to the edge list as alive edges. And AB edge should be set into dead. Figure 15 shows another point has appeared outside the convex layer shown in Figure 14.

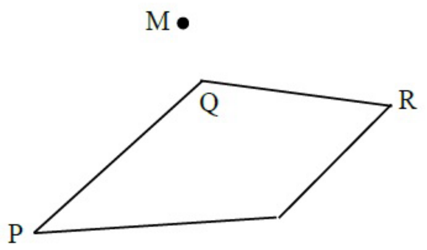

Figure 15: New point appears.

The new point is outside to PQ and QR edges only. Since new point M is outside PQ edge, the PQ edge is set into dead and MP and MQ edges should be included to the edge list as new two alive edges as shown in Figure 16.

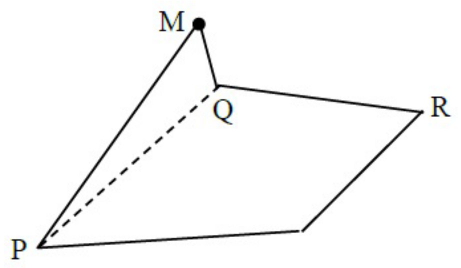

Figure 16: New point is outside PQ.

Since the new point M is outside QR edge also, MQ and MR edges should be added to the edge list as alive edges after setting QR edge dead as shown in Figure 17.

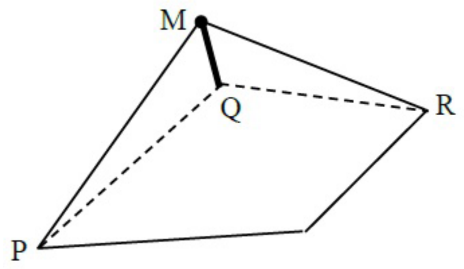

Figure 17: New point is outside QR.

The MQ edge appears twice in the edge list. Therefore it should be set into dead. Similarly if a particular edge appears in the edge list more than once then it should be set into dead. Then convex layer becomes as shown in Figure 18.

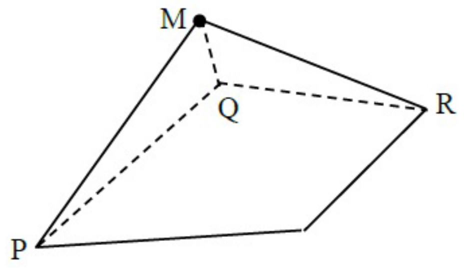

Figure 18: Duplicate edges are dead.

Then the final convex layer is shown in Figure 19.

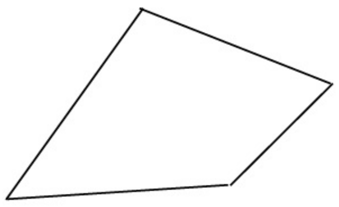

Figure 19: Final form of the layer.

When the convex layer was extended the point Q was left. That point which leaves the current layer should be transferred to the next outermost layer inside the current layer as a new point.

Deletion of a point

The algorithm should be compatible with both insertions and deletions of points. So far how to handle the insertion was discussed. In this section the mechanism used to handle deletion of points is discussed.

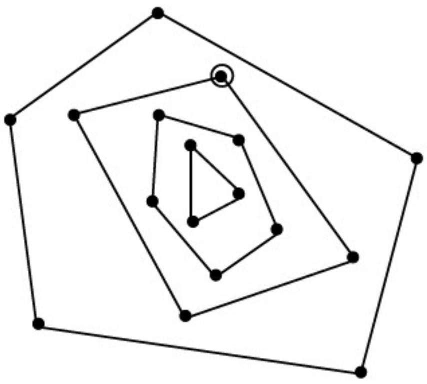

Figure 20: Deletion of a point.

Suppose the point circled in Figure 20 is necessary to be deleted. Then the corresponding layer should be set into dead first. Then the points in the deleted layer except the deleted point should be transferred to the adjacent inner layer of the deleted layer as new points. After setting the corresponding layer dead the set of layers looks like as in Figure 21.

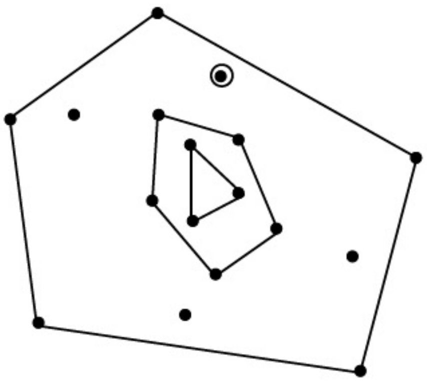

Figure 21: Deletion of a point.

Now the problem was reduced to point insertion. Except the point deleted, other points should be transferred as new points to the adjacent inner layer of the deleted layer. Henceforth the same mechanism used to insert points should be used.

Note that when inserting points to layers, collinear points and coincident points were inserted in separate layers. That was done to make the algorithm compatible with deletion. Suppose there is a set of coincident points. If it was deleted once, only the point in the largest layer is affected. Other coincident points remain as they are. When the inserting points are collinear with an edge of an existing layer, they are always transferred to adjacent inner layer. This process makes the deletion task easier.

Detailed explanation of the pseudo code

This section describes the pseudo code of the proposed algorithm. Only data structure used here is array.

Global variables

The variable *ml* stores the maximum possible number of convex layers. And each layer consists of a set of edges. The variable *me* stores the maximum possible number of edges in a particular layer. The variables *x* and *y* are used to store the coordinates of the new point to be inserted or deleted.

The two dimensional arrays *sx* and *sy* are used to store the coordinates of the starting points of the edges of the layers. And the two dimensional arrays *ex* and *ey* are used store the coordinates of the ending points of the edges of the layers. That means *(sx[i][j], sy[i][j])* stores the coordinates of the starting point of the *j*th edge of *i*th layer while *(ex[i][j], ey[i][j])* stores the coordinates of the ending point of the same edge.

The array *edges* is used to store the number of edges in a particular layer. That means *edges[i]* stores the number of edges (both dead and alive) in the *i*th layer. The variable *layers* stores the number of layers.

The variable *extended* can get two values: zero or one. Once a new point appears in the space as an insertion, the variable *extended* is initially set into zero. After finishing the processing of the new point, the variable *extended* is set into one.

The array *distinct* is used to store the number of distinct points in each layer. That means *distinct[i]* stores the number of distinct points in *i*th layer.

The two dimensional array *e_dead* is used to store the state (dead or alive) of an edge. That means *e_dead[i][j]* = 0 implies that the *j*th edge of *i*th layer is alive. Futher *e_dead[i][j]* = 1 implies that the same edge is dead. The variable *l_dead* is used to store the state (dead or alive) of a layer. That means *l_dead[i]* = 0 implies that the *i*th layer is alive. Futher *l_dead[i]* = 1 implies that the same layer is dead.

The function 'setLayers'

This function is used to decide whether the new point is an insertion or a deletion. The function takes a Boolean variable *ins* as the parameter. If variable *ins* is one then it is an insertion and the function *insLayers* is invoked. If the variable *ins* is zero then it is a deletion and the function *delLayers* is invoked.

The function 'insLayers'

This function finds the largest possible layer so that the new point exists outside it. The variable *layer* is initially zero. That means it initially stores the index of the outermost layer. Then inside the while loop, the variable *layer* is incremented one by one. That means it moves from outermost layer to innermost layer invoking the function *makeHull* until the variable *extended* becomes one. Note that the variable *extended* is initially zero.

The function 'makeHull'

This function takes the index of a particular layer as the parameter. And first it checks whether the given layer is dead or alive. If the layer is dead then it invokes the function *makeHull* considering the adjacent inner layer. If the given layer is alive then there are two possibilities: the layer has reached primary hull state or not. If the number of edges in the given layer is less than three then it has not yet reached primary hull state. Therefore, function *makePHull* should be invoked. If the given layer has already reached the primary hull state then the function *makeSHull* should be invoked.

The function 'makePHull'

This function has two main purposes. First purpose is to create new layers and develop them up to primary hull state. The second purpose is to transfer the new point to adjacent inner layer through function *makeHull* if the new point is not suitable for the current layer. The proposed algorithm does not keep coincident points or collinear points in a layer. Always if the new point is collinear or coincident with the existing points of the layer then the new point is transferred to the adjacent inner layer. This is useful when handling deletions.

The function *makePHull* takes the index of the layer (*h*) as the parameter. If a layer with that index (*h*) does not exist then it should be created. The creation of new layers is done only within this function.

The array element *distinct[h]* stores the number of distinct points in the layer. If the value of this array element is zero then it means that no such layer exists with index *h*. Therefore, the layer is created and the variable *layers* which stores the number of layers is incremented by one. The coordinates of the new point is assigned to the starting point of the 0th edge of the layer. The variable *extended* is set to one since the processing of the new point is over.

If the array element *distinct[h]* is one then that means there already exists a layer with index *h* with only one point. If the new point does not coincide with the existing point of the layer then the coordinates of the new point should be assigned to the ending point of the 0th edge of the layer. If the new point coincides with the existing point of the layer then the new point should be transferred to the adjacent inner layer using function *makeHull*.

Once the layer is a line segment then the new point can be either collinear or non-collinear with the line segment. The discrete version of Green's Theorem (Green, 1991) can be used to test this. In collinear situation, the new point can be on the line segment or outside the line segment. When the new point is on the line segment, the current layer is not extended and the new point is transferred to the adjacent inner layer using function *makeHull*. When the new point is outside the line segment, the current layer is expanded and one of the end points needs to be transferred to the adjacent inner layer. Note that the implementation should handle indeterminate case where line segment is parallel to the y-axis.

Arrival of non-collinear new point when the layer is a line segment forms a triangle. Therefore two more alive edges should be added to the list of edges of the layer.

The function 'makeSHull'

This function is invoked to consider a new point for insertion to a layer which has already reached the primary hull state. The function takes the index (*h*) of considering layer as a parameter. The variables *xc* and *yc* are computed such that *(xc/3, yc/3)* is the centroid of the primary hull.

Using a *for* loop, each edge of the existing layer is accessed. If an edge is not dead then it should be checked whether the new point and centroid of the primary hull are in the same side of the edge or two sides of the edge. In order to do that, variables *txy* and *nxy* are computed using the area of a triangle. The variable *txy* is computed using the edge and the new point. The variable *nxy* is computed using the edge and the centroid of the primary hull. The expression was multiplied by three in computing the variable *nxy* in order to avoid the division operator. Avoidance of the division operator keeps the computation fast and accurate.

If the new point is outside the edge then the expression *txy* * *nxy* is less than zero. Therefore the edge is set into dead and two alive edges joining the new point and each end point of the deleted edge are inserted to the list of edges of the layer.

Then pair of edges which coincide with each other in the list of edges of the layer should be set into dead. One of the end points of these coincident edge pair belongs to dead edges. Therefore that end point is a point which is left by the expanding layer. That end point should be transferred to the adjacent inner layer.

The function 'delLayers'

This function is used to delete a point from the set of points and rearrange the convex layers for the existing set of points. First the point to be deleted should be found. Therefore, using a *for* loop, each alive layers are accessed. In each layer, each alive edge should be accessed using another *for* loop. If the point to be deleted found in an end point of an edge then the layer which belongs that edge should be set into dead first. Then the variable *found* is set to one (Initially the variable *found* is zero) and go to line *L1*. If the variable *found* is still zero that means no such point could be found, therefore no deletion will occur.

After setting dead the entire layer in which point needs to be deleted existed, all the other points except the deleted point should be transferred to the adjacent inner layer of the dead layer. This is done by invoking the function *makeHull*.

RESULTS AND DISCUSSION

The time complexity and the space complexity of the proposed algorithm are discussed at the beginning.

Suppose that there are n points on the plane. The convex layers of the set of points have already been found and suppose that there are k convex layers. The convex layers should be modified upon insertion or deletion of a point from the set of points. The time complexity and the space complexity for an insertion and a deletion should be computed.

Let's consider an insertion of a point first. Each layer contains n/k average number of points. The worst case occurs when the new point appears outside all the layers since then all the layers require modifications. Suppose a point should be transferred to a layer. It takes O (n/k) time to search and set dead the existing edges of the layer in the worst case. Further, It takes O $[(n/k)^2]$ time to detect duplicate edges in the edge list. Altogether, it takes O (n/k) + O $[(n/k)^2]$ = O $[(n/k)^2]$ transfer a single point to a layer. A layer receives O (n/k) points in the worst case. Therefore, it takes O $[(n/k)^2]$ * O (n/k) = O $[(n/k)^3]$ time to transfer points from the outer layer to the current layer. Since there are k layers, it takes O $[(n/k)^3]$ * O (k) = O (n^3/k^2) time to insert a point in the worst case.

Let's consider a deletion of a point. The worst case occurs when a point on the outermost layer is deleted since then all the other layers require modifications. All the points except the point to be deleted should be transferred to inner layers. Thus the deletion problem becomes an insertion problem. Therefore, it takes O (n^3/k^2) time to delete a point in the worst case.

O (n/k) number of new edges should be added to the edge list in order to transfer a single point to a layer. A layer receives O (n/k) number of points in the worst case. Therefore, it takes additional O (n/k) * O (n/k) = O (n^2/k^2) space to process a layer. Since there are k layers, it takes O (n^2/k^2) * O (k) = O (n^2/k) space to insert or delete a point in the worst case.

A single point can be inserted or deleted in O (n^3/k^2) time using the proposed algorithm as shown above. Therefore, the algorithm takes n * O (n^3/k^2) = O (n^4/k^2) time to compute the convex layers in the dynamic context.

The proposed algorithm takes O (n^3/k^2) time in order to perform an insertion or a deletion of a point. The optimal time complexity of a static convex layers algorithm is O (n log n) (Chazelle, 1985). Suppose that an optimal static convex layers algorithm is used in handle an insertion or a deletion. Then the set of convex layers should be reconstructed from the scratch which takes O (n log n) time. If k > n / SQRT (log n) then n log n > n^3/k^2. That means if the number of convex layers is larger than n / SQRT (log n) then the proposed algorithm is faster than using an optimal static convex layers algorithm for an inclusion or a deletion of a point. The dynamic convex layers algorithm invented by Sanjib and Niraj can handle an inclusion or a deletion of a point in O (n) time (Sadhu *et al.*, 2015). It is clear that the number of convex layers of a set of points is always less than or equal to the number of points. Therefore, the Sanjib and Niraj algorithm is faster than the proposed algorithm for single point insertion or deletion.

The proposed algorithm takes O (n^4/k^2) time to compute the convex layers in the dynamic context. An optimal static convex layers algorithm takes O (n log n) time for an insertion or a deletion of a point (Chazelle, 1985). Therefore, an optimal convex layers algorithm takes n * O (n log n) = O $(n^2 \log n)$ time in the dynamic context. If k > n / SQRT (log n) then $n^2 \log n > n^4/k^2$. That means if the number of convex layers is larger than n / SQRT (log n) then the proposed algorithm is faster than using an optimal static convex layers algorithm in the dynamic context. The algorithm proposed by Sanjib and Niraj (Sadhu *et al.*, 2015) computes the convex layers in the dynamic context in O (n^2) time. It is clear that the number of convex layers of a set of points is always less than or equal to the number of points. Therefore, the Sanjib Niraj algorithm is faster than the proposed algorithm in the dynamic context.

In general, the proposed algorithm is slower than Sanjib Niraj algorithm as shown above. However, let's consider the performance of the proposed algorithm on following scenario that widely occurs in practice (O'Rourke, 1997). Suppose the set of convex layers have already been found for a given set of n points. Further, suppose that the layers are close to each other and new points are expected to fall within the region bounded by the outermost layer. In this kind of a situation, the new point is closer to one of the layers. That means only few points should be transferred to inner layers and the time taken by it is negligible. The worst case occurs when new point falls inside the innermost layer. Then all the layers should be searched and it takes O (n) time. The worst case of deletion of a point occurs when a point on the innermost layer should be deleted. It also takes O (n) time to search all layers until point to be deleted is found. Furthermore, it is obvious that the additional memory consumption for this kind of a situation is O (1). Therefore, the proposed algorithm achieves minimum possible time complexity and space complexity for this scenario.

Let's consider the scenario mentioned in the paragraph above for an optimal static convex layers algorithm. The set of convex layers should be constructed from the scratch upon an insertion or a deletion of a point. Therefore, it takes O (n log n) time and O (n) space as discussed above. Therefore, the proposed algorithm is better than using an optimal static convex layers algorithm both in time and space for the widely used scenario. Furthermore, the proposed algorithm achieves the same time complexity of the Sanjib Niraj algorithm for the widely used scenario discussed above.

Sanjib Niraj algorithm is the one and only dynamic convex layers algorithm available in literature. It has three major drawbacks. The algorithm assumes that the set of points does not contain any collinear points. But in practice, set of points contains collinear points (O'Rourke, 1997). The authors proposed a theoretical algorithm without providing a corresponding computer implementation. Therefore, a computer implementation to the algorithm is not available in literature. The algorithm uses the notion of tangent to extend a convex layer. The notion of tangent is not available in higher dimensions. Therefore the algorithm cannot be extended to higher dimensions.

The proposed algorithm is compatible with set of points with coincident points and collinear points. A computer implementation to the proposed algorithm is also provided. Furthermore, the concept of extending a convex layer has already been extended by the authors in their work on an incremental convex hull algorithm in three dimensions (Wijeweera *et al.*, 2018).

CONCLUSION

An algorithm to maintain convex layers for a dynamic set of points was proposed in this paper. The algorithm takes O (n^3/k^2) time and O (n^2/k) space to insert or delete a point.

Furthermore, the algorithm takes O (n^4/k^2) time and O (n^3/k) space to maintain convex layers in dynamic context.

If k > n / SQRT (log n) then the proposed algorithm is faster than using a static convex layers algorithm. The proposed algorithm is slower than Sanjib Niraj algorithm in general. Suppose the set of convex layers have already been found for a given set of n points. Further, suppose that the layers are close to each other and new points are expected to fall within the region bounded by the outermost layer. The proposed algorithm can insert or delete a point in this situation in O (n) time and O (1) space which is equivalent to the performance of Sanjib Niraj algorithm. Actually, that is the theoretical lower bound of the problem (O'Rourke, 1997).

The proposed algorithm is compatible with set of points where there are coincident points and collinear points. A computer implementation in C programming language is also provided. The concept of the proposed algorithm can be extended to higher dimensions as well. Improving the efficiency of the algorithm and extending it to higher dimensions are the future work.

REFERENCES

Berg, M. D., Cheng, O., Kreveld, M. V., Overmars M. (2008). *Computational Geometry Algorithms and Applications*. Springer-Verlag Berlin Heidelberg, Berlin, Germany.

Chan, T. M. (1996). Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions. *Discrete & Computational Geometry* **16(4):** 361-368.

Chazelle, B. (1985). On the Convex Layers of a Planer Set. *IEEE Transactions on Information Theory* **31(4):** 509-517.

Chazelle, B., Guibas L. J., Lee D. T. (1985). The Power of Geometric Duality. *BIT Numerical Mathematics* **25(1):** 76-90.

Green, P. J., Silverman, B. W. (1979). Constructing the Convex Hull of a Set of Points in the Plane. *The Computer Journal* **22(3):** 262-266.

Green, S. L. (1991). *Advanced Level Pure Mathematics*. University Tutorial Press, North Point, Hong Kong.

Huber, P. J. (1972). The 1972 Wald Lecture Robust Statistics: A Review. *The Annals of Mathematical Statistics* **43(4):** 1041-1067.

Nielsen, F. (1996). Output-Sensitive Peeling of Convex and Maximal Layers. *Information Processing Letters* **59(5):** 255-259.

O'Rourke, J. (1997). *Computational Geometry in C*. Cambridge University Press, England, United Kingdom.

Overmars, M. H., Leeuwen, J. V. (1981). Maintenance of Configurations in the Plane. *Journal of Computer and System Sciences* **23:** 166-204.

Preparata, F. P., Shamos M. (1985). *Computational Geometry: An Introduction*. Springer-Verlag New York, New York, United States.

Rufai, R. A., Richards, D. S. (2017). *A Simple Convex Layers Algorithm*. Available from: https://arxiv.org/pdf/1702.06829.pdf (Dec. 31, 2017).

Sadhu, S., Kumar, N. (2015). Computing Convex Layers of a Dynamic Point Set. *International Journal of Computer Theory and Engineering* **7(6):** 495-498.

Shamos, M. I. (1978). *Computational Geometry*. Yale University, United States.

Wijeweera, K. R., Kodituwakku S. R. (2017). Convex Partitioning of a Polygon into Minimized Number of Pieces with Lowest Memory Consumption. *Ceylon Journal of Science* **46(1):** 55-66.

Wijeweera, K. R., Kodituwakku, S. R. (2018). A Simple and Efficient Incremental Convex Hull Algorithm in 3D Space. *Proceedings of 5th Ruhuna International Science & Technology Conference*, Pp. 47.

APPENDIX

The pseudo code and the C programming implementation of the proposed algorithm are available from the following link:

https://www.academia.edu/36703644/On_the_Convex_Layers_of_a_Planer_Dynamic_Set_of_Points