An Efficient Convex Hull Algorithm for a Planer Set of Points

K. R. Wijeweera and U. A. J. Pinidiyaarachchi

Department of Statistics & Computer Science, University of Peradeniya, Sri Lanka
*(*Corresponding author's email: ajp@pdn.ac.lk)*

Received: 21November2012 / Accepted after revision: 21March2013

ABSTRACT

This paper contains a new efficient algorithm to construct the convex hull of a set of points in the plane. The proposed algorithm is able to find the points on the convex hull in boundary traversal order. When the convex hull has collinear points, the algorithm can detect all the collinear points on the hull without skipping the intermediate points. Furthermore it can deal with the data sets where coincident points appear. Two main methods have been used to make the algorithm efficient. First one is achieving parallelism which is done by partitioning the data set. Second one is data reduction which is done by removing unnecessary points at each step of processing. Further we have proved by experimental comparison, that the performance of the presented algorithm is better than interior points' algorithm.

INTRODUCTION

The convex hull of a given set of points can be defined as the smallest convex set containing all the points of that set (Chen, 2001). It is one of the most basic concepts in the fields like convex analysis, data mining, computational geometry, geometric modeling, linear programming, computer graphics and image processing. Practical applications of the convex hull of a set of points are multiple and diverse. This concept has been used to determine the shapes of objects (Kaasalainen, 2001). To fast collision detection, replacing the complex shapes of objects from their convex hulls can be used (Sherali, 2001). Further, convex hull concept has been used in finding the minimum bounding box (O'Rourke, 1993) and constructing spherical Voronoi diagram (Sugihara, 2000).

Finding the convex hull of a set of points is easy if the task is performed by intuition. But, in order to automate the process, some kind of a constructive approach is needed. Special attention should be given to the efficiency and memory consumption when dealing with huge data sets. Also the way a certain algorithm finds the points on the convex hull may be different in each of the existing algorithms. Depending on the application, a suitable algorithm should be selected. If the problem is collision detection (Sherali, 2001), then it is needed to draw the corresponding convex hull of the object. Therefore the set of points on the convex hull should be in boundary traversal order. The algorithm used here should be able to output the set of hull points in an order. In this case if there are collinear points on the convex hull, then the end points of those points on the line are enough to draw the convex polygon. Because of this reason the algorithm can ignore the intermediate points and perform faster. When it comes to outlier detection using convex polygons (Rousseau, 1996) the points on the convex hull are needed to be removed. Therefore the order of the points on the convex hull is not important, but all the points on the convex hull should be detected even in the collinear case.

The convex hull problem is a well-studied problem in the history of computer science and mathematics. Here the attention is restricted to planer convex hull algorithms. Before the invention of efficient algorithms, exhaustive approach has been used. The time complexity of each algorithm can be expressed in terms of input points n and the number of points on the hull h. In the worst case h may be equal to n.

The earliest efficient convex hull algorithm is known as gift wrapping algorithm (Chand, 1970). It has O(nh) time complexity, and in the worst case the complexity is $O(n^2)$. In this algorithm, the convex hull of a set of points is determined by continuing to find the straight line which makes the maximum interior angle with the adjacent boundary until all boundaries of the convex hull is found. This algorithm has been later improved to increase numerical robustness and topological consistency while maintaining the same time complexity (Sugihara, 1994).

A slightly more sophisticated, but more efficient Graham scan algorithm (Graham, 1972) was proposed with O(n log n) complexity. If the points are already sorted by one of the coordinates or by the angle to a fixed vector, then the algorithm takes O(n) time. This method has been improved by introducing a method to simplify the computation to check whether the middle point in three consecutive points is convex or not (Anderson, 1978). Later, quick methods have been invented by extending this algorithm (Koplowitz 1978, Atwah, 1995, Atwah, 2002).

Jarvis algorithm (Jarvis, 1973) is another approach to construct convex hull when the number of given points is relatively small. The behavior of this algorithm is very similar to that of selection sort algorithm where the item goes to the next slot is found repeatedly. This method has been improved to deal with huge data sets by incorporating the ideas of parallel computation (Atwah, 1996).

Another convex hull algorithm with O(n log n) complexity has been proposed by using divide and conquer paradigm (Preparata, 1977). This method is based on merging two non-intersecting convex hulls. First the data set is decomposed in equal parts as left and right. Then the sub problems are solved respectively on left and right. Finally both convex hulls are merged. An algorithm similar to quick sort algorithm called quick hull algorithm has been proposed with O(n log n) complexity (Eddy, 1977). But it may degenerate to O(nh) in the worst case. In this algorithm the set of points are successively partitioned into several regions by the use of binary trees. An algorithm (Akl, 1978) has been introduced using throw away principle integrated with quick hull algorithm. In that algorithm furthest outside point is selected for the selected edge of the current convex hull in each iterative routine and then the selected edge is replaced with two new edges. Since this algorithm does not output the set of hull points in an order, extra processing is needed to locate them in boundary traversal order.

In this paper, a new efficient algorithm to construct the convex hull of a set of points in the plane is introduced. This method is based on the gradient in coordinate geometry with some modifications. To deal with large data sets the algorithm has been fertilized with parallel computation and data reduction. The algorithm is able to find all the points in boundary traversal order. Therefore the proposed algorithm can be applied successfully in the applications like collision detection and outlier detection as mentioned above.

METHODOLOGY

In this section, a new efficient algorithm to determine the convex hull of a finite set of points in the plane is introduced. Directions of x and y axes are shown in Figure 1. These directions are used throughout the research paper.

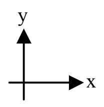


Figure 1: Directions of x and y axes

A gradient based technique has been used to detect points on the convex hull. Consider the straight line segment shown in

Figure 2 where end points are M(mx, my) and N(nx, ny).

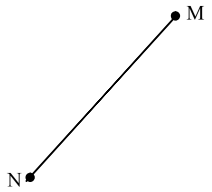


Figure 2: Gradient of a straight line

If the gradient of the straight line segment is m, it can be calculated as follows.

$$m = (ny - my) / (nx - mx)$$

Pseudo code of the first part of the proposed algorithm

```
Input:
  S = {$P_1, P_2, \dots, P_n$}  // a non-empty set of points

Output:
  H // a non-empty set that contains all the points
  of the convex hull of S

Steps:
  1. Calculate following values from the set S.
       a) minx  // minimum value of x coordinates
          y_minx  // y coordinate of the point corresponding to minx
       b) maxx  // maximum value of x coordinates
          y_maxx  // y coordinate of the point corresponding to maxx
       c) miny  // minimum value of y coordinates
          x_miny  // x coordinate of the point corresponding to miny
       d) maxy  // maximum value of y coordinates
          x_maxy  // x coordinate of the point corresponding to maxy

  2. Define following subsets of S.

       A = {(x, y) ∈ S | (x ≤ x_maxy) AND (y ≥ y_minx)}

       B = {(x, y) ∈ S | (x ≥ x_maxy) AND (y ≥ y_maxx)}

       C = {(x, y) ∈ S | (x ≥ x_miny) AND (y ≤ y_maxx)}

       D = {(x, y) ∈ S | (x ≤ x_miny) AND (y ≤ y_minx)}

  3. Find the parts of the convex hull belong to each of the sets A, B, C and D in
     parallel. Then merge those sequences of points to form the set **H**.
```

DISCUSSION ON INPUT AND OUTPUT

There should be a non-empty set of points to find the convex hull. If the given set is non-empty, the set of points belongs to convex hull of those points is also non empty.

Discussion on step 1

There can be more than one point corresponding to each of minx, maxx, miny, maxy values. Among them, one point per each value should be selected arbitrarily. (minx, y_minx), (maxx, y_maxx), (x_miny, miny), (x_maxy, maxy) is considered as those four points. It is obvious that those four points lie on the convex hull of the given set of points as shown in the Figure 3.

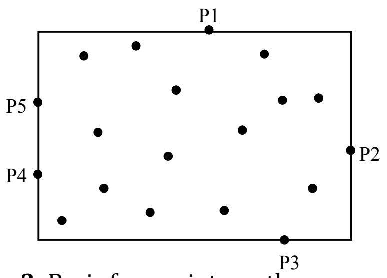


Figure 3: Basic four points on the convex hull

P1, P2, P3 are the points corresponding to the values maxy, maxx and miny. There are two points (P4 and P5) corresponding to the value of minx. In such cases, one point should be selected among them arbitrarily (P4 or P5). Then P4 is taken.

Discussion on step 2

If the data set is large, it will take a lot of time to determine the convex hull. Therefore it is better to partition the data set into regions and process each region in parallel. The process of partitioning the data set can also be done in parallel to speed up the execution. Figure 4 illustrates the separation of the given data points into subsets A, B, C and D.

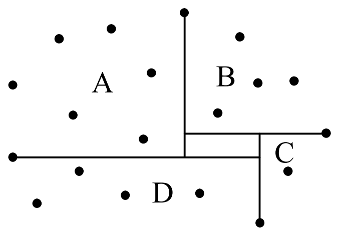


Figure 4: Separation of data set into four regions

Discussion on step 3

Once the given set of points is separated into four subsets, each of those parts can be processed in parallel. Finding the points of the convex hull is done in clockwise as shown in Figure 5.

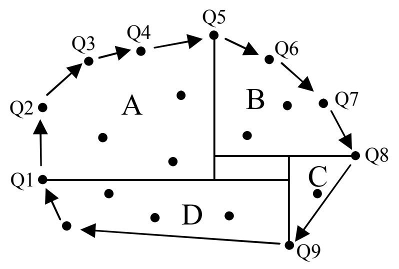


Figure 5: Finding the convex hull of the data set

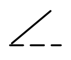
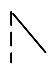
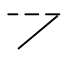
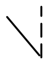
Here, set of points of the convex hull, H = {Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10};
Set of hull points belong to region A, HA = {Q1, Q2, Q3, Q4};
Set of hull points belong to region B, HB = {Q5, Q6, Q7};
Set of hull points belong to region C, HC = {Q8};
Set of hull points belong to region D, HD = {Q9, Q10};

Q5, Q8, Q9 and Q1 has been excluded from the A, B, C and D sets respectively in order to make HA, HB, HC and HD sets mutually exclusive. The points in those HA, HB, HC and HD sets is found in boundary traversal order. Therefore if those sets are merged sequentially, the set H with the points are in boundary traversal order of the convex hull can be formed. Because of this reason, that sequence of hull points can be used to draw the minimum convex polygon enclosing the given set of points. That is also an advantage of the proposed algorithm.

On the second part of the algorithm

After partitioning the data set into four regions, each region should be processed separately in parallel. The gradient with some modifications is used to process each region. Four separate algorithms have been defined to process each region though a single algorithm could have been used. However data sets in other three regions should then be adapted either from rotation or transformation to be compatible with that single algorithm. Therefore it is better to use four separate algorithms with slight modifications to process each region as shown in Table 1.

Table 1: Use of modified gradient for each of the region

Region	Diagram	Modified gradient	Begin point	End point
A		(ay[i]-hully)/(ax[i]-hullx)	x=minx y=y_minx	x=x_maxy y=maxy
B		(bx[i]-hullx)/(hully-by[i])	x=x_maxy y=maxy	x=maxx y=y_maxx
C		(hully-cy[i])/(hullx-cx[i])	x=maxx y=y_maxx	x=x_miny y=miny
D		(hullx-dx[i])/(dy[i]-hully)	x=x_miny y=miny	x=minx y=y_minx

Pseudo code for the part of algorithm to process region 'A'

Algorithm to draw hull relevant to 'A'

```
INPUT:
    (ax[], ay[]) // all the points in A
    a; // number of points in A
    minx, y_minx, x_maxy, maxy;
BEGIN

// To store the value of i
maxi = 0;

// Initialize the hull point
hullx = minx;
hully = y_minx;
```

```
REPEAT
    Save the hull point (hullx, hully);

    // initially, maximum value of gradient is zero
    maxm = 0;

        // minimum distance of y coordinates
    mindfy = maxy - y_minx;
    /*
     Initially, mindfy = maximum distance of y coordinates
    */

    // minimum distance of x coordinates
    mindfx = x_maxy - minx;
        /*
         Initially, mindfx = maximum distance of x coordinates
        */

        // Boolean variable is set into      false
    found = FALSE;

    // for each point in the set A
    FOR i = 0 TO i = a – 1
         // point is above the line y = hully and right of the line x = hullx
         IF [(ay[i] >= hully) AND (ax[i] >= hullx)] IS TRUE
              // point is vertically above the current hull point
              IF (ax[i] = hullx)
                   // point is not equal to current hull point
                   IF (ay[i] != hully)
                        // vertical distance from the point to current hull point
                        dfy = ay[i] - hully;
                        // selecting next closest hull point to current hull point
                        IF (mindfy>= dfy)
                             mindfy = dfy;
                             maxi = i;
                             found = TRUE;
                        END IF
                   END IF
              ELSE IF found IS FALSE
                   // calculating the gradient
                   m = (ay[i] - hully) / (ax[i] - hullx);
                   // selecting the point with maximum gradient
                   IF (maxm< m)
                        mindfx = ax[i] - hullx;
                        maxm = m;
                        maxi = i;
                   // point is on the y = hully line
                   ELSE IF (maxm = m)
                        dfx = ax[i] - hullx;
                        // selecting next closest hull point to current hull point
                        IF (mindfx>= dfx)
                             mindfx = dfx;
                             maxi = i;
                        END IF
                   END IF
              END IF
         END IF
    END FOR

// Assign new point to the hull point
hullx = ax[maxi];
hully = ay[maxi];
UNTIL [(hullx = x_maxy) AND (hully = maxy)] IS FALSE

END
```

NOTE: Similar approaches can be used to process regions B, C and D.

Implementation of the part of algorithm to process region 'A'

The C++ programming language has been used to implement the algorithm. The implementation is provided in order to improve the understandability.

```
void drawHullA(double ax[],double ay[],inta,doubleminx,doubley_minx,doublex_maxy,doublemaxy)
{
inti;
doublehullx,hully; // To store coordinates of a point on the convex hull
doublem,maxm; // To store gradient, maximum of gradients respectively
int maxi=0; // To store value of i corresponding to maxm
doubledfy,mindfy; // To store distance to point, minimum distance to point
doubledfx,mindfx; // To store distance to point, minimum distance to point
int found; // To store whether hull point found:indeterminate

hullx=minx;
hully=y_minx;

do
{
    setcolor(15);
    circle(hullx,hully,2);

    maxm=0;
    mindfy=maxy-y_minx;
    mindfx=x_maxy-minx;
    found=0;

    // Calculating the gradients
    for(i=0;i<a;i++)
    {
        if((ay[i]>=hully) && (ax[i]>=hullx)) //
Point is above the horizontal line y=hully
        {
            if(ax[i]==hullx)
            {
                if(ay[i]!=hully)
                {
                    dfy=ay[i]-hully;
                    if(mindfy>=dfy)
                    {
                        mindfy=dfy;
                        maxi=i;
                        found=1;
                    }

                }
            }
            else if(!found)
            {
                m=(ay[i]-hully)/(ax[i]-hullx);
                if(maxm<m)
                {
                    mindfx=ax[i]-hullx;
                    maxm=m;
                    maxi=i;
                }
                else if(maxm==m)
                {
                    dfx=ax[i]-hullx;
                    if(mindfx>=dfx)
                    {
                        mindfx=dfx;
                        maxi=i;
                    }
                }
            }
        }
    }

hullx=ax[maxi];
hully=ay[maxi];
}while(!((hullx==x_maxy) && (hully==maxy)));

}
```

Analysis of the part of algorithm to process region 'A'

In this section, the algorithm is analyzed for some example data sets. Further, some special cases are discussed which are encountered during processing of different data sets.

Example Data Set 1 (General Case)

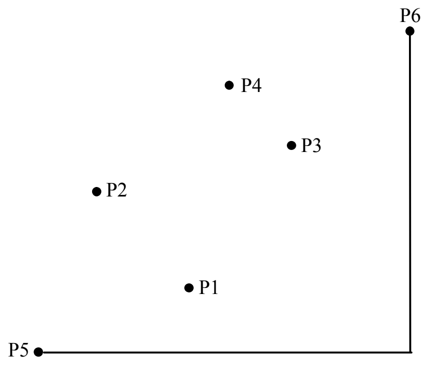


Figure 6: Situation 1

Initially as shown in Figure 6, A = {P1, P2, P3, P4, P5, P6}; Begin with P5. P5 is a hull point. Then with respect to P5, P2 has the maximum gradient. Therefore P2 is selected as a hull point. Now Only {P2, P3, P4, P6} should be considered. With respect to P2, P4 has the maximum gradient. Therefore P4 is selected as a hull point. Now only {P4, P6} should be considered. With respect to P4, P6 has the maximum gradient (Since it is the only point). Then P6 is selected as a hull point. But at this point the process is stopped. Actually P6 belongs to hull point set of region B.

Example Data Set 2 (Collinear Case)

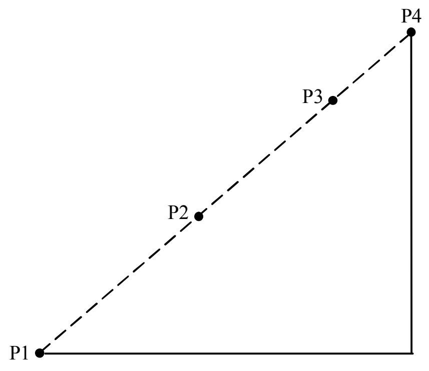


Figure 7: Situation 2

Initially as shown in Figure 7, A = {P1, P2, P3, P4}; Begin with P1. P1 is a hull point. Then with respect to P1, three points have the same gradient. They are P2, P3 and P4. Therefore the closest point out of those three points should be selected as the next hull point. Because of that reason, P2 is selected (Observe the use of *mindfx* variable when *maxm = m*). If P2 is considered, then P3 has to be selected as the next hull point as mentioned above. Then with respect to P3, P4 is the next hull point. So that point is ignored and stops processing.

Example Data Set 3 (Indeterminate Case)

Initially as shown in Figure 8, A = {P1, P2, P3, P4, P5}; Begin with P1. P1 is a hull point. Then with respect to P1, there are two pointswith indeterminate gradients (Other than P1). Priority should be given to that type of points (Observe the use of *found* variable). Those two points are P2 and P3. Out of them the closest point to P1 should be selected as the next hull point (Observe the use of *mindfy* variable). Therefore P2 is the next hull point. With respect to P2, P3 is the only point with indeterminate gradient. So the next hull point is P3. P4 and P5 points have the equal gradients with respect to P3. Out of those two points the closest point to P3 should be selected (Observe the use of *mindfx* variable). So the next hull point is P4. P5 is the only point with respect to P4 that should be considered. So P5 is selected. But P5 is the end point, so it ignored it and stops the process.

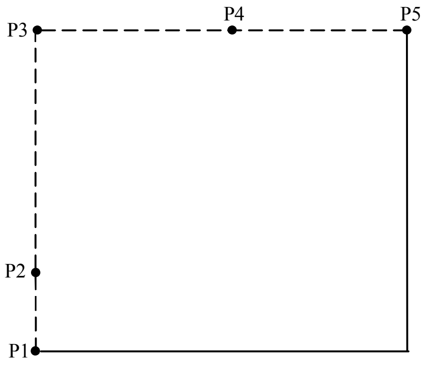


Figure 8: Situation 3

RESULTS AND DISCUSSION

The proposed algorithm has been implemented in C++ programming language with a computer with following hardware resources.Intel(R) Pentium(R) Dual CPU; E2180 @ 2.00 GHz; 2.00 GHz, 0.98 GB of RAM;

To compare the performance of the proposed algorithm an existing algorithm called interior points algorithm was implemented on the same machine using the same programming language. Interior points' algorithm is based on the following Lemma.

Lemma: A point is non-extreme if and only if it is inside some (closed) triangle whose vertices are points of the set and is not itself a corner of that triangle.

Then the algorithm is as follows (O'Rourke, 1997),

```
Algorithm: INTERIOR POINTS
for each $i$do
for each $j \neq i$do
for each $k \neq i \neq j$do
 for each $l \neq i \neq j \neq k$ do
       if$p_l \epsilon \Delta(p_i, p_j, p_k)$
             then$p_l$ is non-extreme
```

Following IDE was used to test two algorithms, Turbo C++; Version 3.0; Copyright(c) 1990, 1992 by Borland International, Inc.

First *n* number of random points were generated in the range 0-399 by using randomize() function(Kodituwakku 2012a, Kodituwakku 2012b) and written them to a text file. Such text files were created for different number of data points. Then the number of clock cycles taken to process each data file *k* times by each of the algorithms is counted using clock() function. The results are as shown in Tables 1-5. (A denotes the proposed algorithm, B denotes interior points' algorithm),

Average ratio is calculated for each of the five cases as follows,

$$Average\ Ratio = \frac{\sum Clock\ cycles\ for\ interior\ points\ algorithm}{\sum Clock\ cycles\ for\ the\ proposed\ algorithm}$$

To normalize the results, number of clock cycles was calculated when k=1000. X(1000) denotes the average number of clock cycles for X algorithm when k=1000.

A(1000) and B(1000) in Table 6 are shown as a graph in Figure 9.

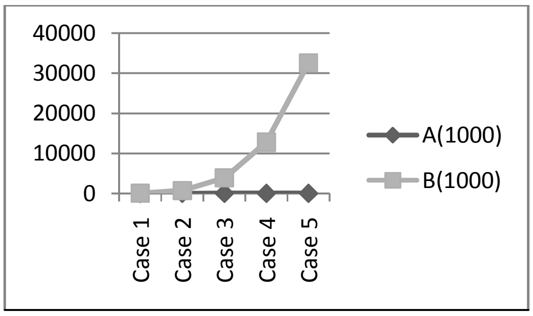


Figure 9: A(1000) and B(1000) for each of five cases

Results show that performance of the proposed algorithm is better than interior points' algorithm. Further when the number of data points is increased, the performance of the proposed algorithm becomes more prominent. Average ratio grows exponentially with the number of data points as shown in Table 6.

Table.6: Average ratios for each of five cases

Case	A(1000)	B(1000)	Average ratio
1	5.53	37.11	6.71
2	10.60	693.80	65.45
3	15.20	3881.6	255.36
4	17.00	12769.00	751.11
5	24.00	32554.00	1356.41

Table 1: n = 10, k = 32000

File	1	2	3	4	5	6	7	8	9	10
A	165	198	165	181	192	170	186	180	165	170
B	1197	1197	1182	1183	1182	1191	1192	1185	1184	1185

Table 2: n = 20, k = 1000

File	1	2	3	4	5	6	7	8	9	10
A	9	8	13	9	14	9	14	8	14	8
B	692	698	688	699	692	695	691	694	694	695

Table 3:n = 30, k = 500

File	1	2	3	4	5	6	7	8	9	10
A	5	6	11	6	11	5	11	6	10	5
B	1946	1942	1937	1936	1939	1950	1938	1942	1939	1939

Table 4: n = 40, k = 100

File	1	2	3	4	5	6	7	8	9	10
A	2	1	2	1	3	2	1	3	1	1
B	1275	1272	1272	1280	1269	1284	1276	1276	1272	1293

Table 5: n = 50, k = 50

File	1	2	3	4	5	6	7	8	9	10
A	1	1	1	1	1	2	1	1	1	2
B	1625	1626	1620	1624	1627	1626	1636	1627	1648	1618

CONCLUSION

In this paper a new efficient convex hull algorithm was introduced. The proposed algorithm can deal with the data files where coincident points appear. And also when there are collinear points on the convex hull, the proposed algorithm can find all of the data points on the hull without skipping intermediate hull points. Therefore the proposed algorithm can be successfully used in both kinds of applications like collision detection and outlier detection. Further this algorithm can be developed so that it can perform in higher dimensions than two.

REFERENCES

S. G. Akl and G. T. Toussaint (1978) A Fast Convex Hull Algorithm, Information Processing Letters, Vol. 7, No. 5, pp. 219-222.

K. R. Anderson (1978) A Reevaluation of an Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set, Information Processing Letters, Vol. 7, No. 1, pp. 53-55.

M. M. Atwah and J. W. Baker (2002) An Associative Static and Dynamic Convex Hull Algorithm, Proceedings of the International Parallel and Distributed Processing Symposium, Fort Lauderdale, FL, USA, pp. 15-19.

M. M. Atwah, J. W. Baker and S. G. Akl (1996) An Associative Implementation of Classical Convex Hull Algorithms, Proceedings of Eighth IASTED International Conference on Parallel and Distributed Computing and Systems, Chicago, IL, USA, pp. 435-438.

M. M. Atwah, J. W. Baker and S. G. Akl (1995) An Associative Implementation of Graham’s Convex Hull Algorithm, Proceedings of the Seventh IASTED International Conference on Parallel and Distributed Computing and Systems, Washington D.C., USA, pp. 273-276.

D. R. Chand and S. S. Kapur (1970) An Algorithm for Convex Polytopes, Journal of the Association for Computing Machinery, Vol. 17, No. 1, pp. 78-86.

Han-Ming Chen and Tzung-Han Lin (2006) An Algorithm to Build Convex Hulls for 3-D Objects. Journal of the Chinese Institute of Engineers, Vol. 29, No. 6, pp. 945-952.

W. F. Eddy (1977) A New Convex Hull Algorithm for Planar Sets, ACM Transactions on Mathematical Software, Vol. 3, No. 4, pp. 398-403.

R. L. Graham (1972) An Efficient Algorithm for Determining the Convex Hull of A Finite Planar Set, Information Processing Letters, Vol. 1, No. 4, pp. 132-133.

R. A. Jarvis (1973) On the Identification of The Convex Hull of a Finite Set of Points in The Plane, Information Processing Letters, Vol. 2, No. 1, pp. 18-21.

M. Kaasalainen and J. Torppa (2001) Optimization Methods for Asteroid Light Curve Inversion, ICARUS International Journal of Solar System Studies, Vol. 153, No. 1, pp. 24-36.

S. R. Kodituwakku, K. R. Wijeweera, M. A. P. Chamikara (2012) An efficient algorithm for line clipping in computer graphics programming; will appear in Ceylon Journal of Science (Physical Sciences.

S. R. Kodituwakku, K. R. Wijeweera, M. A. P. Chamikara (2012) An Efficient Line Clipping Algorithm for 3D Space, International Journal of Advanced Research in Computer Science and Software Engineering. Volume. 2. Issue. 5. pp. 96-101. May 2012. ISSN 2277 128X.

J. Koplowitz and D. Jouppi (1978) A More Efficient Convex Hull Algorithm, Information Processing Letters, Vol. 7, No. 1, pp. 56-57.

J. O’Rourke (1997) Computational Geometry in C (Second Edition), Cambridge University Press. New York, USA, pp. 66-67.

J. O’Rourke (1993) Computational Geometry in C, Cambridge University Press. New York, USA, pp. 113-167.

F. P. Preparata and S. J. Hong (1977) Convex Hull of A Finite Set of Points in Two and Three Dimensions, Communications of the ACM, Vol. 20, No. 2, pp. 87-93.

P. Rousseau and A. Leroy (1996) Robust Regression and Outlier Detection, John Wiley & Sons, 3rd edition.

H. D. Sherali, J. C. Smith and S. Z. Selim (2001) Convex Hull Representations of Models for Computing Collisions between Multiple Bodies, European Journal of Operational Research, Vol. 135, No. 3, pp. 514-526.

K. Sugihara (2000) Three-Dimensional Convex Hull as a Fruitful Source of Diagrams, Theoretical Computer Science, Vol. 235, No. 2, pp. 325-337.

K. Sugihara (1994) Robust Gift Wrapping for The Three-dimensional Convex Hull, Journal of Computer and System Sciences, Vol. 49, No. 2, pp. 391-407.