EUROPEAN ACADEMIC RESEARCH
Vol. I, Issue 11/ February 2014

ISSN 2286-4822
www.euacademic.org

Impact Factor: 3.1 (UIF)
DRJI Value: 5.9 (B+)

Fast Gilbert-Johnson-Keerthi Method for Collision Detection between Two 2D Convex Hull Polygons Based on Chan's and Divide_Conquer Algorithms

ASIA MAHDI NASER ALZUBAIDI
Computer Science Department, College of Science
Karbala University, Karbala
Iraq

Abstract:

We can regard collision detection or intersection detection as a problem of determining whether two or more objects intersect, overlap, or collide at one or more vertices within an environment. Collision detection is fundamental in diverse set of applications such as physical simulation, computer simulated environments, computer animation, solid modeling and robotic motion planning and even in basic GUI applications. In this paper we present Gilbert-Johnson-Keerthi (GJK) technique in order to solve collision detection problem. One of the more attractive features of GJK is that it can only operate on convex shapes, so that we initially obtain the first convex polygon (A) using Chan's algorithm and the second convex (B) based on Divide_Conquer method. Then we apply GJK algorithm to check whether those hulls are collided or not. Originally the basic idea of the GJK is that it used to find the minimum Euclidean distance from one Convex Shape to another but it can be adapted to determine whether or not they intersect. GJK uses a mathematical construct called Minkowski difference (AϴB) through taking the difference of every point in A and every point in B then constructing the convex hull of resulting points. If the two Convex Shapes collide then the Minkowski difference operator will contain the origin. But this is not very practical as the number of points needed is $|A|*|B| = O(n^2)$ *assuming* $|A| \approx |B| = n$ *large, so computing convex hulls is an expensive operation. Therefore, instead of computing the Minkowski difference set points we will construct it implicitly through using a support mapping function to build a polygon inside the Minkowski Difference called the simplex. Then check if simplex contains the origin point which already found in Minkowski Difference then we can conclude that the Minkowski Difference contains the origin and collision happened.*

Key words: GJK algorithm, Common Tangent, Collision Detection, Minkowski Difference, Support Mapping Function, Convex hulls, Chan's & Divide_Conquer algorithms.

1. Introduction

The problem of collision detection of convex polygonal objects has been extensively studied in many fields such as robotics, computer animation and computational geometry (Kelvin 1996). In general the goal of collision detection is to report when and where a geometric contact occurs. This paper provides an overview and execution of GJK algorithm as a very efficient method of detecting the collision between two convex polygonal objects. The original idea of this algorithm was developed by E.G.Gilbert, D.W. Johnson, and S.S. Keerthi in 1988 for finding the minimum distance between convex polyhedral models in 3D environments (Gilbert et al. 1988). Then it enhanced and developed to provide collision information such as calculating the deeper penetration distance and contact point especially when supplemented by other algorithms that have improved performance, robustness, and versatility of GJK implementation (Jovanoski 2008). The attractive feature of GJK is that it can be applied to any shape in any m-dimensional space such as polytopes, quadrics, Minkowski sums of convex objects, and image under affine transformation (Lindemann 2009). The GJK algorithm relies heavily on the concept of Minkowski difference, as in equation (1) on calculating if there is collision or not between the concave polygonal objects because the most important feature of the Minkowski difference is the fact that if two shapes are intersecting, then the Minkowski Difference will actually contain the origin point (Bittle 2010). As showing in figure (1), the Minkowski difference of two convex shapes involves mirroring of one shape about the origin and then summing it to each boundary vertex of the other shape and taking the convex

hull of the result shape (Thomason 2012). In GJK method the Minkowski Difference needs to be explicitly constructed, this means that we just want to know whether the Minkowski Difference contains the origin point or not and this can be done by construct 2-simplex. The goal of GJK algorithm is to construct a polygon that contains just three vertices of Minkowski Difference and then check if it contains the origin or not (Eric 2011).

$$Minkowski\ point\ (x, y) = (Ax - Bx, Ay - By) \ldots \quad (1)$$

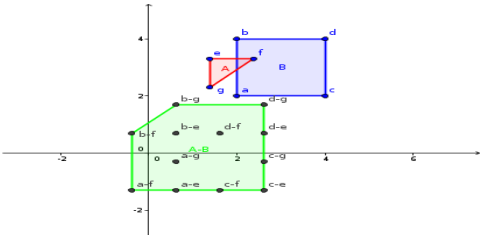


Figure (1) Minkowski differences of two convex hull

In the second section of this paper we will provide some information about the GJK algorithm's history. In section 3 we present the block diagram of the proposed collision detection system while in the next section we introduce a brief overview over the Convex hull algorithms that include Chan's Divide Conquer algorithms. Section 5 gives some preliminaries that are necessary to grasp the functionality of GJK algorithm and to explain some basic mathematical knowledge required to implement it. Section 6 gives an overview of GJK algorithm for collision detection problem. Finally, we introduce example for experimental result of the implementation of the proposed collision system and last sections provide the paper's conclusion and references.

2. Literature Survey

In 1988, E.G. Gilbert, D.W. Johnson and S.S. Keerthi presented

the original GJK algorithm which is restricted to calculate the Euclidean distance between two convex polyhedral (Kelvin 1996). The results show that the computational cost takes linear time in the total number of the two polytopes vertices. Then Mirtich illustrates collision detection algorithms for robot motion planning over both broad phase and narrow phase detection strategies and their alternative such as Lin-Canny, the algorithm using axes-aligned bounding boxes and a hierarchical spatial hash table. He concludes that line-canny is the fastest way to compute the distance between disjoint objects and to reduce the computational cost of intersection detection; it is prefer to independently schedule collision tests between pairs of objects instead of checking all pairs at fixed periods (Mirtich 1997).

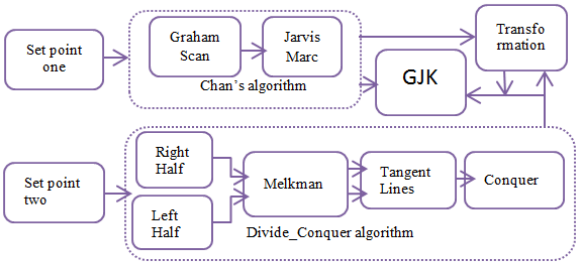
In 2003 there has been provided the algorithm for overlapping detection between significant geometric objects represented by polygonal shapes undergoing animation with deformable surfaces. The proposed method is based on the notion of using two kinds of interacting particles distributed on a surface for determining the minimum distance between two models. The experimental results show that the proposed algorithm can handle sufficiently complicated animation situations in real time (Senin et al. 2003). Then Kockara et al. (2007) provided a grasp classification of collision detection algorithms into two general parts: broad-phase and narrow-phase. They made a Survey to explain some of the existing collision detection algorithms such as sweep and prune, hierarchical hash tables, Cinder method for 3D space and GJK algorithm which was difficult to interpret. They found that Narrow-phase algorithms are usually given more detailed information and this information can be later used for the computation of time of impact, collision response and forces, and contact determination. Moreover, the thesis in 2010 gives a theoretical outline of the GJK algorithm for computing the distance between two convex objects in 2D & 3D space with

MPR based heuristic algorithm for computing penetration depth, intersection detection and manifold generation. The researcher shows how both SA-GJK and MPR algorithms behave in different situations and finds that these algorithms work well for collision detection purpose and they are only different in runtime; MPR wins in all implementation because GJK in each iteration finds the optimal search direction for the next iteration, this computation being significantly expensive. Moreover, he recommends to use both GJK and EPA for computing depth information in all cases (Olvång 2010). Then, in 2011 the researchers used cluster partitioning method to form convex hulls by split triangle meshes into convex hulls, then applied the GKJ algorithm to solve collision detection problem. They found that the Cluster number has a great influence on the precision of collision detection, so that if cluster number increased, then the precision could also increase (Liu et al. 2011).

3. Proposed Collision Detection System

In this section, we present an overview of the proposed system for collision detection. The block diagram of it is depicted in Figure 2. The first stage involves randomly generated of n of 2D points for two sets of points, since n is at least 3 points to construct the two convex hull. Secondly from the first point set we construct the first convex hull using Chan's algorithm which involved hybrid of graham scan and Jarvis March algorithms. The second points set are used to construct the second convex hull by Divide_Conquer algorithm which firstly involves sorting the points in increasing order then splitting the sorted points into two halves. From each we compute convex hull using Melkman algorithm and then merge the two convex hull to construct the sorted convex hull using left and right common tangent lines method. At this stage we have two convex hulls. Thirdly, we need to pass these two convex hull through a

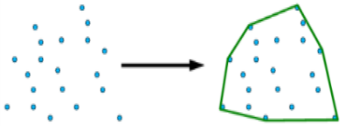
transformation process which involves translating and scaling to change the location and size of these hulls if it is necessary. Finally, the GJK algorithm is presented to detect if there is collision between hulls or not.



Figure(2) block diagram of Collision Detection system

4. The Convex Hull of a Planar Point Set

The convex hull of any 2D points set or polygon in the plane can be defined as the smallest subset of most extreme points on X-axis or Y-axis which is not included in any interior set points as indicated in Figure 3. Mathematically a polygon is convex if any line segment joining two points on the boundary stays within the polygon (Sunday 2012).



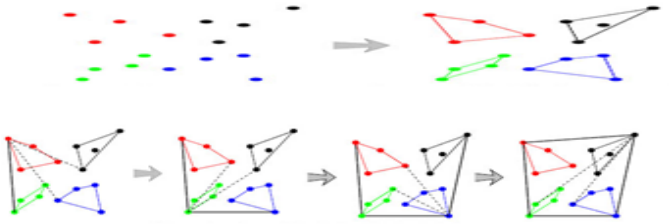
Figure(3)set of points and its corresponding convex hull

Frequently, the construction of Convex hull is used in many fields such as computational geometry, computer graphics, shape modeling and analysis, collision detection, interference computation, pattern recognition, statistics, GIS,… etc. (Tanga et al. 2012). GJK works with convex hull shapes only and approaches the collision of an actual object by computing 2-simplex function. There are many different convex hull finding algorithms, all of them taking a set of points and returning

decreasing ordered subset of those points that form the convex hull (CS255 Program2 2012). For this paper we need to construct two convex polygons so we will implement two of convex hull algorithm.

4.1 Chan's Algorithm

Chan's algorithm is one of an optimal output sensitive algorithms that is used to construct the convex hull of a set P of n points in 2D or 3D dimensional space (Wikipedia 2013). In the planar case, an algorithm known as Graham's scan achieves in O(nlogn) running time. There is another algorithm known as a Jarvis March or gift wrapping algorithm, which has a complexity of O(h n) where h is the number of points in the resulting hull (Hoffmann 2009). Chan's algorithm involves a very clever combination of the previous two algorithms, that are, Graham's scan and Jarvis's March, to form an algorithm that is faster than either one with O(n log h) time complexity. The main idea is smart and elegant based on Divide and conquer concept (Switzer 2010). Divide stage begins by splitting the input points n in to r = n/m groups, each of size m range from 4, 16, 32, 256, 512,…, then for each subset we compute the sub hull p(m) using Graham's scan algorithm. While in conquer or merge Stage we follow the general outline of Jarvis's march, it starts from finding the smallest point of p0, then successively finding the convex hull vertex that follows p0 in increasing order using binary search until we return back to the smallest point again (Bo et al. 2007), as in Figure(4).



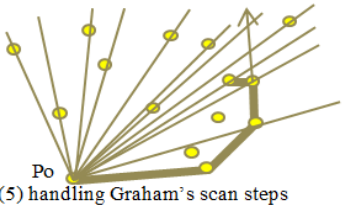
Figure(4) Execution of Graham Scan

Chan's algorithm(p)

1. For t = 1; 2; …
2. Let m = min(2^(2^t), n)
3. Let r = ceil(n/m)
4. **Divide stage**
5. For k = 1 to r do
6. For i = 1 to m do
7. Compute sub hull P(m)) using Graham's scan and store the vertices
   in an ordered array S in CCW oriented.
8. End
9. **Merge stage**
10. Find leftmost vertex S0 from all sub hulls that resulted from Graham
    scan algorithm
11. Compute final convex hull ch(h) using Jarvis March algorithm

4.1.1 Convex Hull by Graham's Scan Algorithm

The idea of algorithm is to find an extreme point p0 which is guaranteed to be in the resulting convex hull, then sort all other points as viewed from that vertex according to the angles in increasing order. Construct the convex hull by checking the direction(CCW) of point and adding the vertices when it makes a left turn direction, and back tracking when making a right turn (Bo et al. 2007). As depicted in Figure(5).



Figure(5) handling Graham's scan steps

Graham Scan Algorithm (p)

1. Compute polar angles as follow, then sort the points

according to their angle in CCW oriented.

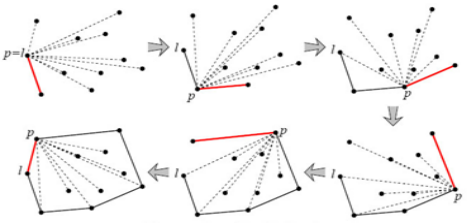
- Find angle between the current line (x1, y1, x2, y2) with X-Axis.
- Let DX = (x2-x1) and DY= (y2-y1)
- If (DX = 0) and (DY = 0) return 0.
- Else compute angle by using angle =ArcTan2(DY,DX)*57.30.
- If (angle < 0) return angle +360.

2. Let p0 be the point in the p with minimum.
3. Let p1, p2,...,pm be the remaining points in p, sorted by polar around p0.
4. Push(S,p0) , n=1
5. for i =2 to m do
6. find direction (CCW) of three points (p0,p1,p2) as follow:
   - dx1 = p1.x - p0.x
   - dy1 =p1.y - p0.y
   - dx2 = p2.x - p0.x
   - dy2 = p2.y - p0.y
   - if (dx1*dy2 > dy1*dx2) turn left
   - else if (dx1*dy2 < dy1*dx2) turn right
7. while (CCW (p [n-1],p [n], p [i]) > 0)do
   - if (n > 1)then n = n- 1
   - else if (i = m) break
   - Else
   - go to 13
8. End while
9. n = n+1
10. Swap(p [n] , p [i])
11. Push(S, p)
12. End for
13. return S

4.1.2 Convex Hull by Jarvis March Algorithm

This algorithm manipulates the points on the convex hull in the order in which they appear. It is started with an extreme

point p of a point set S which represents the leftmost point. At each step, we test each of the points, and find the one which makes the smallest turn right *l* and has to be the next one on the hull. Then we update *p* to *l* and repeat the process until we end up with the leftmost point (Mount 2007). As shown in Figure 6.



Figure(6) Execution of Jarvis March steps

Jarvis March(S)

1. Let S0 the point whose x-coordinate is the smallest, to be the first point in
   the output convex hull ch
2. Let L be the index of point S0 in S
3. Repeat
4. h= 0
5. ch[h] = S0
6. h=h+1
7. Compute farthest point q =(p+1)% n
8. For i = 0 to n - 1
9. if CCW (S[S0],S[i], P[q])= turn right
10. q = i
11. S0 = q
12. until S0 =L
13. Return ch

4.2 Convex Hull by Divide_Conquer Algorithm

The Divide-and-Conquer approach is an algorithm for solving convex hull problem for a planar point set P. This algorithm needs O(nlogn) time complexity and it is based on the principle of divide-and-conquer design technique or it is an extension of

merge sort algorithm (Felkel 2012). It begins if the number of points set P is at least three points, sorting all points by their x-coordinate in an increased order then the divide phase begins by partitioning the point set P into two halves L and R, where L consists of left half points with the lowest x coordinates p[0,..,n/2]and R consists of right half points with the highest x coordinates p[n/2+1,..,n]. Recursively compute convex hull for each half using Melkman algorithm. The merge phase begins by conquering the two hulls into a common convex hull using common tangent lines algorithm by finding the upper and lower tangents then discarded all points between these two tangents [15].

4.2.1 Convex Hull by Melkman Algorithm

This is used to produce a convex hull with a CCW orientation. It is probably the best convex hull algorithm, since it runs in linear time and take O(n) run time(Mitchell 2012). The strategy of the Melkman algorithm is straightforward, it uses a deque as a main structure to save incremental hull vertices that already processed. Actually deque D is a double-ended queue, so it can insert and remove from both top and bottom ended (Sunday, 2012).

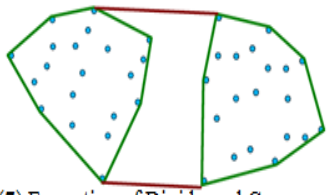
Melkman Algorithm (S)

1. Top=3 , bottom =0
2. D[top]=D[bottom]=S[2]
3. If CCW(S[0],S[1],S[2]> 0
4. D[bottom+1]=S[0]
5. D[bottom+2]=S[1]
6. Else
7. D[bottom+1]=S[1]
8. D[bottom+1]=S[0]
9. End
10. For i=3 to n-1
11. If CCW(D[bottom], D[bottom+1],S[i]) > 0 and

```
    CCW(D[top-1], D[top],S[i]) > continue
12. While CCW(D[bottom], D[bottom+1],S[i]) <= 0
13.  bottom=bot tom+ 1
14.  bottom = bottom – 1
15.  D[bottom] = S[i]
16. While CCW(D[top-1],D[top],S[i])<= 0
17. top =top - 1
18. top = top + 1
19. D[top] = S[i]
20. End for
21. for  h=0 to (top-bottom)
22. H[h] = D[bottom + h]
23.  Return H
```

4.2.2 Common Tangent Lines Algorithm

Finding the upper and lower Tangent lines between two disjoint convex polygons is a fundamental geometric operation and it resembles to find the maximum and minimum extreme vertices of the polygon (Sunday 2012). The algorithm uses to find independently the upper and lower tangent lines between the two convex hulls that resulted from Melkman algorithm in linear time. The right hull vertex is sorted in CCW ordered while the points of left hull are sorted in CW oriented. Then discard any points in the rectangular formed by the two lines. The remaining points can be easily added to merge hull in the form of a clockwise sequence (Vertlieb 2012), as depicted in Figure 7.



Figure(7) Execution of Divide and Conquer algorithm

Common Tangent Algorithm (LH, RH)

1. Let a the right most point in left hull and b the left most point in the right hull.
2. Let done is a Boolean number and with false as initial value
3. while done = 0
4. done = 1
5. while CCW(RH[b], LH[a], LH[a-1]) <= 0
6. a=a-1;
7. while CCW(LH[a], RH[b], RH[b+1]) >= 0
8. b=b+1
9. done = 0
10. End while
11. return a, b

5. Preliminaries

This includes a number of essentially mathematical issues that help us to understand the outlines of GJK algorithm such as 2-simplex, farthest point and support function.

5.1 Simplex

The GJK algorithm performs the idea of whether shapes are overlapping by testing if the Minkowski Difference contains the origin point or not. Instead of computing the Minkowski Difference in O(n^2)run time we just want to construct incrementally a polygon within the Minkowski boundary called simplex, which contains the origin and for 2D it is a triangle (Golgobot 2010). The simplex is the convex hull of independent set of points so that a point is a 0-simplex, a line is a 1-simplex, a triangle is a 2-simplex, and a tetrahedron is a 3-simplex. The goal of the GJK algorithm in 2D space is the ability to build a 2-simplex that contains the origin so collision happen or determine that such a simplex cannot be constructed (Zhang 2013).

5.2 Direction Vector

Choosing the initial value of direction vector can be calculated by subtracting the center point C1 of shape A from the center point C2 of shape B as in equation 2. Finding the center of a 2D polygon can be done in a number of methods. In this paper we just picked a point that looked like the center to make the mathematical easier using equation 3 (Ericson 2005).

$$D = C2 - C1 \qquad \ldots(2)$$

$$A = \frac{1}{2}\sum_{I=0}^{N-1}(XiYi + 1 - Xi + 1Yi)$$

$$Cx = \frac{1}{6A}\sum_{i=0}^{N-1}(Xi + Xi + 1)(XiYi + 1 - Xi + 1Yi)$$

$$Cy = \frac{1}{6A}\sum_{i=0}^{N-1}(Yi + Yi + 1)(XiYi + 1 - Xi + 1Yi) \qquad \ldots(3)$$

where

D: Direction Vector.

A: The Area Weighted Method.

Cx, Cy: Center Point of Convex Hull.

5.3 Farthest Point

The purpose of supporting mapping function is to create a simplex in the Minkowski differences and this can be done by choosing the farthest point according to direction value. This operation is significant because it creates a simplex that contains a maximum area therefore increasing the chance that the algorithm will exit quickly. Mathematically, we can obtain the farthest point of shape points with direction value by performing the dot product of the points with the value of direction then the point with maximum value will represent the farthest point of the shape (Bittle 2010).

5.4 Support Mapping Function

The purpose of support mapping function is to create a simplex

in the Minkowski differences and this can be done by choosing the farthest point according to the value of direction vector. This operation is significant because it creates a simplex that contains a maximum area and therefore it increases the chance that the algorithm will terminate quickly (Golgobot 2010). A support mapping is a function that takes a direction vector to search along and returns the vertex on a first shape A that is a farthest point in that direction and subtract it from the farthest point of the second shape B in the opposite direction; the result is point on the edge of the Minkowski Difference (Ericson 2005), as indicated in equation (4).

Point on Minkowski along D = (Farthest point on A in direction D)-
(Farthest point on B in direction -D) … (4)

5.5 Translation Function

A point in XY plane can be translated by adding translation amount to the coordinates of the point, for each point P(X,Y) which is to be moved by TX units parallel to the X-axis and TY units parallel to the Y-axis to get new point p (Hearn et al.2004), as in equation (5).

$$Px = X + TX$$
$$Py = Y + TY \qquad \ldots (5)$$

5.6 Scaling Function

A scaling transformation alters the size of an object. This operation can be carried out for convex polygon by multiplying the coordinate values (X, Y) of each vertex by scaling factors SX in X- axis and SY in Y-axis. We can control the location of a scaled object by choosing a position called the fixed point (XF, YF) that is to remain unchanged after the scaling operation (Hearn et al.2004), as in equation 6.

$$Px = (X - XF)SX + XF$$

$$Py = (Y - Yf)SY + YF \qquad \ldots(6)$$

6. Collision Detection by GJK Algorithm

The Gilbert-Johnson-Keerthi's GJK algorithm is an efficient method for 2D intersection test between two convex shapes. It works through a support mapping by constricting and updating a 2- simplex inside Minkowski difference (Zhang 2013). In each iteration GJK checks if the simplex polygon contains the origin point or not by dividing the area around the simplex into regions depending on the kind of simplex. GJK needs to check in which of these regions the origin lies, if so then the Minkowski difference must also contain the origin and GJK terminate with a distance of 0 and collision happens between the two shapes (Golgobot 2010). If the origin is not in the simplex polygon the GJK algorithm finds a new simplex with a new direction to expand the simplex in, it should be in towards the origin as the last simplex and still is inside the Minkowski differences. Finally, it removes any points from the simplex that are no longer necessary; this depends on the size of the simplex. Because we are working in 2D, we are concerned with the 0-simplex being a point, 1-simplex being a line, and 2-simplex - a triangle (Ericson 2005).

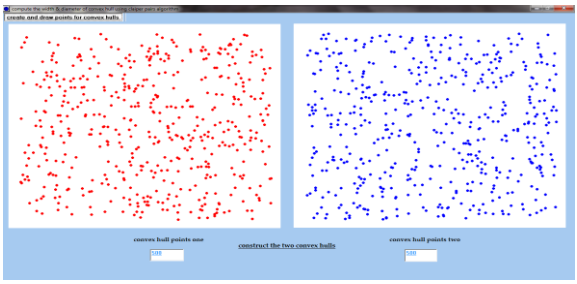
GJK Algorithm

1. Initialize the 0-simplex as follow
   - Compute the initial value of direction by equation (2).
   - n=0 such n don't increase of 2
   - simplex[n]=support (A,B,D)
   - compute the negative value of direction D
2. While(true)
3. n=n+1
4. simplex[n]=support(A,B,D)
5. if (simplex[n] dot product of D <=0) then

6. the last added point not pass the origin so return 0
7. Else
8. a = Simplex[n]
9. a0.x:= 0 - a.X
10. a0.Y:= 0 - a.Y
11. if n>=2 then
12. b = Simplex[n-1]
13. c = Simplex[n-2]
14. ab=b - a
15. ac=c - a
16. abPerp = tripleProduct(ab, ab, ac)
17. acPerp = tripleProduct(ac, ac, ab)
18. if (abPerp dot product a0 > 0) then remove c
19. set abPerp to the new direction D
20. Else
21. If (acPerp dot product a0 > 0) then remove b
22. set acPerp to the new direction D
23. Else Return 1
24. Else if n < 2
25. b = Simplex[n-1] , ab = b – a
26. abPerp = tripleProduct(ab, a0, ab)
27. set abPerp to the new direction D
28. return 0

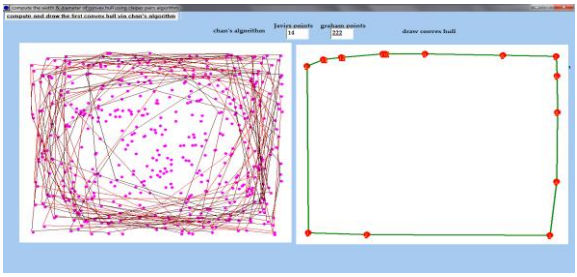
7. Experimental Results of System

More details can be found in this section of software implementation. The block diagram in figure (2) explains the GUI operations performed by system to implement collision detection between two convex hull objects. We explain the results with example of 500 points. Firstly we need to generate two sets of random points each set detected to construct one of convex hulls as shown in figure(8).



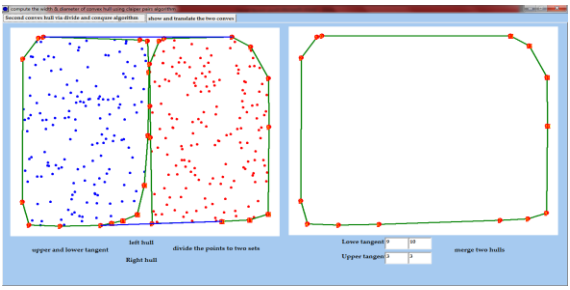
Figure(8) the GUI of randomly generated of two 2D set points

Then we build the Chan's convex hull by graham scan to construct n sub groups then applying Jarvis algorithms to obtain one group represent the final convex hull as depicted in the GUI Chan's algorithm in figure(9)



Figure(9) the GUI of Chan's Algorithm

The second convex hull is constructed by using divide-conquer with Melkman algorithm. The program system divides the set of sorting points, relative to x coordinate, into the left and right groups, then applying the Melkman algorithm at each group to obtain two convex hulls. Finally we obtain the final convex hull by merging the two previous hulls using upper and lower common tangent lines algorithm and discard any point between these two lines as shown in figure (10).



Figure(10) GUI of Divide_Conquer Algorithm and Tangent lines

Figure (11) shows the chart resulted of comparing the implementation of Chan's algorithm and Divide-Conquer algorithm for different numbers of polygon vertices.

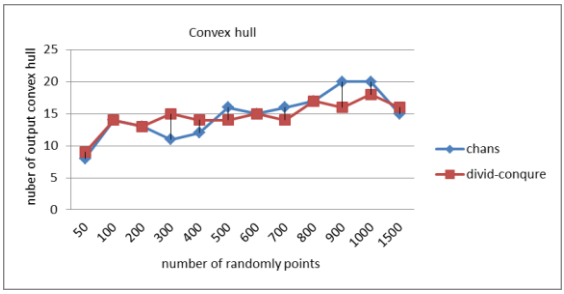


Figure (11) chart compares Chan's with divide-conquer

From this chart we can see that both algorithms have approximately the same behavior with 1500 points but from our test we found that Melkman algorithm deals just with simple number of points while Chan's algorithm could deal with any set of numbers. After having two convex hulls we need some time to change the size or location of them by using transformation on 2D set points. Then we apply GJK algorithm to check if these objects are intersected or not depending on the ability to build the simplex polygon in Minkowski difference as in figure (12)

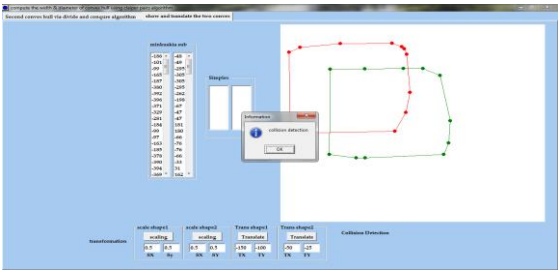


Figure (12) GUI of convex hulls after transformation and GJK algorithm

Figure (13) shows the GUI of two collided convex hulls with their Minkowski difference and simplex polygon.

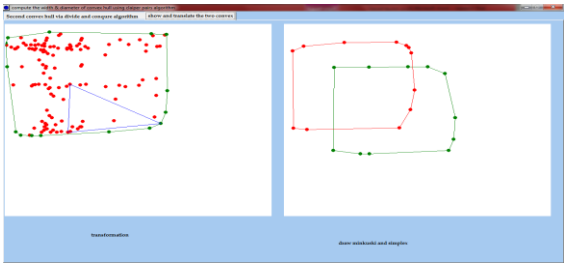


Figure (13) GUI of collided hulls and Minkowski difference with simplex polygon

8. Conclusion

In this paper we have provided the Gilbert-Johnson-Keerthi's procedure to collision detection between two convex objects in 2D space. At each iteration it is significantly expensive to find the optimal search direction for the next iteration to construct the simplex. Actually, GJK algorithm is very fast and versatile, easy to implement, can be used with many types of objects whether in 2D or 3D plane, popular and widely used in terms of collision detection routine with linear time. Moreover, GJK algorithm does not require a special structure to save geometry data like other algorithms because it depends on the support mapping function to read the geometry of the objects and detect whether there is collision or not. The algorithm was

implemented in Delphi programming language and tested on a PC under Microsoft Windows operating system.

BIBLIOGRAPHY:

_____. CS255 Program2. 2012. "Convex Hulls a Divide-and-Conquer Algorithm 15 points." http://hivemined.ir/wp-content/uploads/2012/04/convexHull.pdf

Bittle, W. 2010. "GJK (Gilbert–Johnson–Keerthi)." Available at http://www.codezealot.org/archives/88#gjk-intro

Bo, Z., H. Xisheng, and J.Mingxun. 2007. "Convex Hull in 2D." Available at http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg_cs_07w/Webprojects/Zhaobo_hull/

Eric. 2011. "GJK Algorithm." Available at http://entropyinteractive.com/tag/gjk.

Ericson, C. 2005. "The Gilbert-Johnson-Keerthi algorithm." Available at http://realtimecollisiondetection.net/pubs/SIGGRAPH04_Ericson_GJK_notes.pdf.

Felkel, P. 2012. "Convex Hulls." Available at https://cw.felk.cvut.cz/wiki/_media/misc/projects/oppa_oi english/courses/ae4m39vg/lectures/04-convexhull.pdf

Gilbert E. G., D. W. Johnson, and S. S. Keerthi. 1988. "A fast procedure for computing the distance between complex objects in three-dimensional space." *IEEE Trans. Robot. Autom* 4(2): 193–203.

Golgobot. 2010. "GJK Algorithm." Available at http://physics2d.com/content/gjk-algorithm

Hearn, D. and M. P. Baker. 2004. *Computer Graphics - C version*. Pearson Education.

Hoffmann, M. 2009. "Chan's Algorithm." Available at http://www.ti.inf.ethz.ch/ew/courses/CG09/materials/v2.pdf

Jovanoski, D. 2008. *Gilbert – Johnson – Keerthi's (GJK)*

Algorithm. Department of Computer Science University of Salzburg.

Kelvin C.T.L. 1996. *An Efficient Collision Detection Algorithm for Polytopes in Virtual Environments*. M. Phil thesis, The University of Hong Kong.

Kockara, S., T. Halic, and K. Iqbal. 2007. "Collision Detection: A Survey." IEEE. 1-4244-0991-8.

Lindemann, P. 2009. "The Gilbert-Johnson-Keerthi Distance Algorithm." The Media Informatics Proseminar on Algorithms in Media Informatics. University of Munich, Germany.

Liu, S. and Y. Zheng. 2011. "GKJ and clustering based algorithm for collision detection and proximity query on deformable body." *Multimedia Technology (ICMT)* International Conference. 3539 – 3542.

Mirtich, B. 1997. "Efficient Algorithms for Two-Phase Collision Detection." Available at http://www.merl.com

Mitchell, J. 2012. "Melkman's Convex Hull Algorithm." Available at www.ams.sunysb.edu/~jsbm/courses/345/melkman.pdf.

Mount, D. M. 2007. "CMSC754ComputationalGeometry." Department of Computer Science University of Maryland.

Olvång, L. 2010. *Real-time Collision Detection with Implicit Objects.* Master Thesis, Institute for Information Technology, Department of Information Technology. Uppsala University.

Senin, M., N. Kojekine, V. Savchenko, and I. Hagiwara. 2003. "Particle-based Collision Detection." The Euro Graphics Association.

Sunday, D. 2012. "The convex hull of a planer point set." available at http://geomalgorithms.com/a10-_hull-1.html

Sunday, D. 2012. "Fast convex hull of a 2D simple polyline." http://geomalgorithms.com/a12-_hull-3.html

Sunday, D. 2012. "Tangent to & between 2D polygons."

Available at http://geomalgorithms.com/a15-_tangents.html

Switzer, T. 2010. "2D Convex Hulls: Chan's Algorithm." online available at http://tomswitzer.net/2010/12/2d-convex-hulls-chans-algorithm/.

Tanga. M, J. Zhaoa, and R. Tonga. 2012, "GPU accelerated Convex Hull Computation." Zhejiang University, China, http://gamma.cs.unc.edu/volccd/ghull/paper.pdf

Thomason, A. 2012. "Physics for games." Available at http://www.andythomason.com/lecture_notes/physics/physics_collision.html#2.

Vertlieb, A. 2012. "Parallel Convex Hull using MPI." Available at http://www.cse.buffalo.edu/faculty/miller/Courses/CSE633/Vertlieb-Fall-2012-CSE633.pdf.

Zhang, X. 2013. *Gilbert-Johnson-Keerthi (GJK) Algorithm*. The University of North Carolina at Chapel Hill, USA.

Wikipedia, Timothy, M. 2013. "Chan's algorithm." Available at http://en.wikipedia.org/wiki/Chan's_algorithm.