Efficient Collision Detection between 2D Polygons

Juan José Jiménez, Rafael J. Segura, Francisco R. Feito
Departamento de Informática. E.P.S.J. Universidad de Jaén
Av. Madrid, 35
Spain (23071), Jaén, Jaén
{juanjo, rsegura, ffeito}@ujaen.es

ABSTRACT

Collision detection between moving objects is an open question which raises major problems concerning its algorithmic complexity. In this paper we present a polygon collision detection algorithm which uses polygon decomposition through triangle coverings and polygon influence areas (implemented by signs of barycentric coordinates). By using influence areas and the temporal and spatial coherence property, the amount of time needed to detect a collision between objects is reduced. By means of these techniques, a valid representation for any kind of polygon is obtained, whether concave or convex, manifold or non-manifold, with or without holes, as well as a collision detection algorithm for this type of figures. This detection algorithm has been compared with the well-known PIVOT2D [Hof01] one and better results have been achieved in most situations. This improvement together with its possible extension to 3D makes it an attractive method because pre-processing of the polygons is no longer necessary. Besides, since this method uses sign operations, it proves to be a simple, more efficient and robust method.

Keywords

Animation, Barycentric Coordinates, Coherence, Collision Detection, Triangle Cover.

1. INTRODUCTION

The problem of *collision detection* among objects in motion is essential in several application fields, such as in simulations of the physical world, robotics, animation, manufacturing, navigation in virtual worlds, etc. Apart from giving scenes a more realistic appearance, it is necessary for the objects belonging to it to interact, so that they do not collide, and if they do, a suitable response is obtained.

Due to this, collision detection is studied intensively by the scientific community. Most of the algorithms developed are based in heuristics that aim to reduce collision determination time, but they are not usually valid for some types of figure, such as non-convex polygons, non-manifold polygons or polygons with holes. At worst, given two polygons, it is necessary to check the intersection between all pairs of edges, with $O(n \cdot m)$ time, n and m being the number of edges of each figure.

Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972
WSCG'2004, February 2-6, 2003, Plzen, Czech Republic.

In this work, on the one hand, we try to use a formal 3D solid representation system, and on the other one, to use it for the collision detection among rigid solids (first among 2D polygons). This formal system is based on polygons coverings by means of triangles (in 2D) and operations with signs. This provides more efficient and robust operations according to Feito [Fei98].

On the other hand, the barycentric coordinates of a point regarding a triangle are used in order to determine the point or polygon inclusion [Bad90]. The use of barycentric coordinates can be seen computationally more intensive, but after the initial step, and once the sign is calculated, it is only needed to recalculate the coordinates sign when the point changes from some spatial zones to others. In addition, it provides a measure of the distance of the point to each triangle, and of course to the polygon, so that we can verify if a point or a polygon is to a given distance from the static polygon.

In order to check its efficiency, this algorithms have been compared with other ones, such us inclusion algorithms, and 2D collision detection ones, obtaining satisfactory results that induce us to develop and implement these techniques in 3D in the future.

There follows a brief scheme of the contents that are going to discussed in this paper: after the introduction we will define the collision detection term used by the authors, as well as its possible applications, especially in 2D. Then, we shall present a summary of the authors' previous work on which the development of this new algorithm of collision detection between 2D polygons is based. Next, we provide definitions of the basic concepts necessary to understand this algorithm. Then we present the new algorithm and its implementation. Later, a temporal study will be carried out, in which the new algorithm is compared with the one developed by Hoffman [Hof01] in the *PIVOT2D* library. Finally, in the conclusions section, we summarise the features of the new algorithm and future work to be undertaken by the authors.

2. COLLISION DETECTION DEFINITION AND APPLICATIONS

Collision detection is the interference or intersection determination between two moving objects. We can determine just whether collision occurs or not or else we can calculate the features of the objects involved in the collision. The response to the collision may imply distortion or a change in the trajectory of the objects, but this lies outside the scope of our study.

We define collision detection at several levels. At the first level, we deal with collision detection between two objects. The complexity is $O(n \cdot m)$. At the second level, we deal with collision detection of several moving objects. In this case, the complexity is $O(k^2)$, where k is the number of objects in the scene. In this paper we deal with collision detection between two objects.

We can see that collision detection is a problem similar to *inclusion detection*, so that we can regard collision detection as a problem of inclusion detection in consecutive time intervals. Nevertheless, we can somehow exploit the *space and temporal coherence* of the movement of the objects in order not to repeat calculations, or we may simply eliminate features of the objects on which the collision test is not to be made. This coherence is going to be used in the new algorithm.

On the other hand, we can see that a great number of applications which need collision detection may be reduced to objects moving on a flat surface, for example, a vehicle travelling around a city, a machine that must carry loads in an warehouse, or in virtual reality applications, in which an avatar moves about a scene in which it must avoid certain obstacles, etc. Other applications make use of collision detection between objects on a plane. For example, the design of integrated circuits may require determining *"paths"* for the various tracks connecting chips and electronic components; the automatic design of roads may require the determination of the best way of avoiding certain obstacles, etc.

3. PREVIOUS WORK

Previous work has carried out a characterisation of the collision detection problem and the strategies used to solve it [Jim02a]. Other authors have also made a revision of this problem [Jim01] [Lin98].

In this paper we present several techniques in order to solve the collision detection problem in 2D, so that if it proves to be effective, it may be extended to 3D in a future.

Then, we present some of the developed works by the authors; they will be a reference for the understanding of the collision detection algorithm between polygons presented in this paper.

To study the inclusion of a point in a polygon we used the algorithm proposed in [Fei95] adapted to use barycentric coordinates. In Algorithm 1 the result of this adaptation is shown.

```
int Polygon::inclusionTest(point p) {
    sum = 0
    i = 0
    while (i < triangleNumber) {
        is_in = Triangle[i]->inclusionTest(p)
        if (is_in==EDGE_EXTERNAL OR
              is_in==VERTEX_V1 OR is_in==VERTEX_V2)
            return IN
        else
            if (is_in==IN)
                sum += 2*Triangle[i]->sign()
            else
                if (is_in==EDGE_RIGHT OR
                      is_in==EDGE_LEFT)
                    sum += Triangle[i]->sign()
        i++
    }
    if (sum==2) return IN
    else return OUT //No inclusion
}
```

Algorithm 1. Point-Polygon inclusion test.

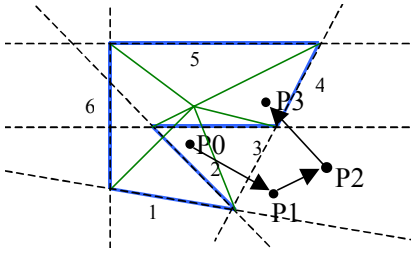
In [Jim02b] we can see different algorithms for the collision detection between a point and several types of figures (convex and non-convex -such as starred figures, random contour maps and totally irregular figures-). These algorithms are optimised according to the type of figure. Besides, the non-convex collision detection algorithm adapts to all the situations and offers quite good times.

We have compared the point-polygon collision detection algorithm with the crossings test inclusion [Hai94] [Las96] and the signed area [Fei95] [Hof89] ones. This algorithm is efficient in most situations, with higher execution times than crossings test algorithm, but quite near to it. Also, the times obtained are better than those ones in signed area algorithm.

The present work is based on the point - non-convex polygon algorithm [Jim02b], which has been extended so that it works with two polygons. Like its predecessor, time and space coherence is used to reduce the number of necessary calculations in collision determination. The summarised *point-polygon collision detection algorithm* (Algorithm 2), and an operation example (Figure 1) may be seen underneath.

1. Make a triangles covering of the polygon with origin in the centroid of the figure.
2. Make a space division through the sign of barycentric coordinate associated to the triangle edge that belongs to the polygon (see section 4.3.)
3. Calculate the sign of the moving point with respect each zone, keep it in a bit mask (in which value 1 means that the point is on the inner side, and value 0 on the outer side)
4. Move the point
5. Recalculate the sign with respect to each zone and compare it with the previous mask
6. If new mask is equal to the old mask return EQUAL_STATE. Go to step 3.
7. If one bit changes from 0 in the old mask to 1 in the new mask:
   7.1. Calculate barycentric coordinates t and u, only when the bit of the mask is 1
   7.2. If this change has not taken place, the point is in the same zone. Return OUT and go to step 3.
8. Check whether the point is inside each triangle. By using Algorithm 1
9. Return IN or OUT accordingly, and go to step 3.

Algorithm 2. 2D Point-polygon collision detection.



Position	P0	P1	P2	P3
Order	123456	123456	123456	123456
Sign	+-++++	+-++++	+-++++	+-++++
s coord. mask	110111	110011	110011	111111
t coord. mask	11-000	10--01	------	100100
u coord. mask	11----	1----0		0--100
state	OUT	OUT	EQUAL	IN

Point in position P1 is in the same zone as in P2. If the point changes from P2 to P3, it changes zones.

Figure 1. Sample operation of 2D point-polygon collision detection algorithm. A covering of the polygon by triangles has been carried out and shows a division on zones.

4. MATHEMATICAL AND GEOMETRIC FOUNDATIONS

Basic Definitions

Definition 1: Let x be a real number. We define the *sign function*, *sign (x)*, as follows :

$$\text{sign}(x) = (1, \text{if } x>0;\ 0, \text{if } x=0;\ -1, \text{if } x<0)$$

Definition 2 : Let three points be $A,B,C \in R^n$, for n=2, the coordinates of which on the plane are $A(x_A,y_A)$, $B(x_B,y_B)$, $C(x_C,y_C)$, the *signed area* of these three points is defined as [Hof89] :

$$|ABC| = \tfrac{1}{2} * \begin{vmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{vmatrix}$$

The signed area of these three points can be negative, depending on the points order. For counter-clockwise order the signed area is positive and it is zero if the three points are aligned.

Definition 3 : A triangle T is *positive* if sign (|T|)=1, and *negative* if sign (|T|)= -1.

Theorem : Let points be $A,B,C \in R^2$, and let's suppose they are not collinear. Let another point be $P \in R^2$. Then, there are unique $s,t,u \in R$, so that $s + t + u = 1$ and $P = sA + tB + uC$, which means:

$$s = |BCP| / |ABC|$$
$$t = |CAP| / |ABC|$$
$$u = |ABP| / |ABC|$$

The numbers *s,t,u* defined in the above theorem are the *barycentric coordinates* of P with regard to points A, B and C.

Lemma : Let a point be $P \in R^2$ with barycentric coordinates *(s, t, u)* in relation to points A, B, $C \in R^n$. It is said that point P is inside the triangle defined by points A, B, C, if and only if [Bad90] :

$$0 \le s, t, u \le 1$$

Corollary : Likewise, we define point P as outside the triangle ABC, if and only if :

$$s < 0 \vee t < 0 \vee u < 0$$

Geometric Interpretation of Barycentric Coordinates of a Point with Regard to a Triangle

If barycentric coordinates of a point P in relation to *S,T,U* are *s,t,u*, then a point with *sign(s)=+1* will be placed on the same side as S, with respect to the infinite line that goes through U and T. If *sign(s)=-1*, it will be on the opposite side; if *sign(s)=0*, it will be on the line (Figure 2).

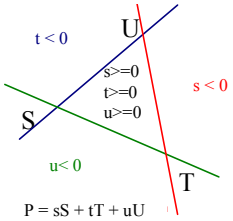


Figure 2. Geometric interpretation of barycentric coordinates of a point with regard to a triangle.

Coverings of the Polygons and Zones Division

A preliminary step in all algorithms consists in a *covering of the polygon by triangles* with origin at the centroid of the polygon (Figure 3). This covering is a *generator system* that is valid for every type of 2D polygon (as well as for its extension to 3D polyhedral solids), *manifold* or *non-manifold*, *with or without holes*, *convex* or *non-convex* [Fei95] [Fei97].

Let the centroid be S, and let T,U be the vertices of a triangle of the covering, the space can be divided into two zones, one for the points with barycentric coordinate *s<=0*, and another one for points with barycentric coordinate *s>0* (Figure 2). When a point changes the sign of its barycentric coordinate s with regard to some triangle of the covering, we say that it has *moved from one zone to another* (Figure 1).

Influence Area of an Edge

Given a 2D polygon, the *influence area with length "n" of an edge "e"* is defined as a 2D space zone which is external to the triangle formed by the centroid of the polygon and the edge *"e"*, and bounded by two infinite and parallel lines, the first one going through the vertices of the edge and the second one at a distance of *"n"* units from the first one (Figure 3). This influence area can be limited by the barycentric coordinate s.

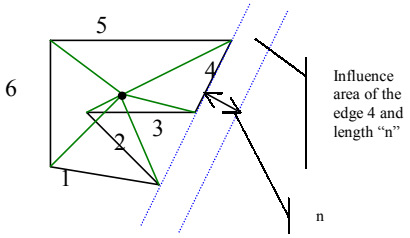


Figure 3. Influence area of an edge

Given a 2D polygon, the *influence area with length "n" of the polygon* is defined as the addition of the influence areas with length *"n"* of each of the edges forming the polygon.

Extended Influence Area of an Edge

Given a 2D polygon, the *extended influence area with length "n" of an edge "e"* is defined as a 2D space zone relative to the triangle formed by the centroid of the polygon and the edge *"e"*, and bounded by two infinite lines which are parallel to the line which passes through the vertices of the edge, both at a distance of *"n"* units from the last one (Figure 4). This influence area can be limited by barycentric coordinate s.

Given a 2D polygon, the *extended influence area with length "n" of the polygon* is defined as the addition of extended influence areas with length *"n"* of each of the edges forming the polygon.

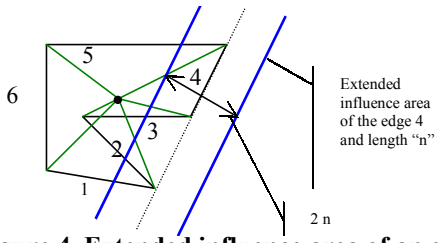


Figure 4. Extended influence area of an edge

5. DEVELOPED ALGORITHMS

Let's see the development of the *2D polygons collision detection algorithm* step by step. Firstly we will see the most simple (but least efficient) collision detection algorithm, so that we may gradually optimise it.

Point-Triangle Inclusion Test

In order to determine whether a point is included in a triangle or not, we just have to check whether the barycentric coordinates of the point with regard to the coordinates forming the triangle are all within the interval [0,1] or, we may just check whether any of the barycentric coordinates is negative.

Two 2D Segments Intersection Test

In order to check whether two segments intersect in 2D space, the barycentric coordinates of a point *P* with respect to the other three *S,T,U* are calculated (the points are the extremes of the two extreme segments) and the sign of these coordinates is calculated too. If an intersection occurs, then barycentric coordinates must be *s<=0, t>=0 and u>=0*, as can be seen in Figure 5:

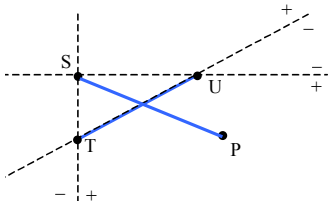
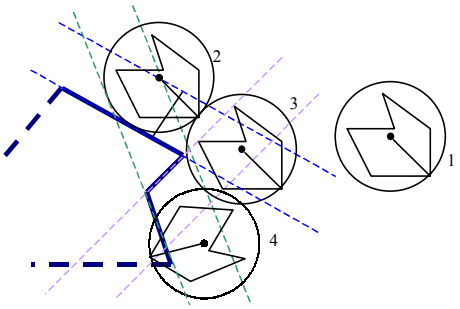


Figure 5. Edges intersection

2D Polygon-Polygon Intersection Test

Static polygon-polygon intersection test consists in calculating whether intersection occurs between two edges, one from each polygon. The segment-polygon inclusion test can be repeated for all the edges of both polygons. The run time of this algorithm is *O(n·m)*, *n* and *m* being the number of edges of each one of the polygons involved in the test. In order to reduce these times, an influence area with respect to one of the polygons can be created, outside which it is known that the polygons are not going to collide. The second polygon is surrounded by a circumference, whose centre is the c*entroid* or common point of the covering and whose radius is the maximum length from that point to each one of the vertices of the polygon.

Firstly, a *polygon influence area* with length equal to the circumference radius is used. For a collision to take place, the centroid of the second polygon must be inside the influence area of the first one. If it is not, collision will certainly not take place. The problem has been reduced to detecting the collision between a point and a polygon greater than the first one, bounded by its area of influence. When the centroid enters this area, a *detailed collision detection test* is applied (Figure 6).



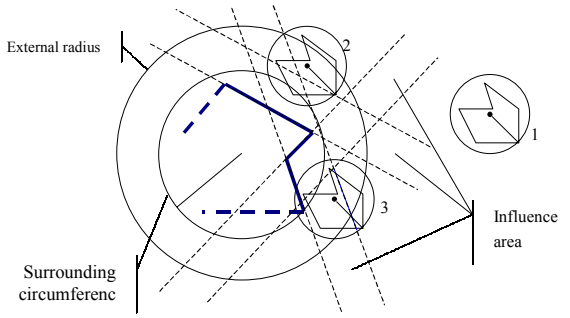
*In **1**, the centroid is inside the influence area: collision does not take place. In **2**, the centroid is at the limit of the influence area, the influence area length is equal to the radius of the circumference. In **3**, the centroid is inside the influence area: the detailed collision test is conducted. Collision does not take place. In **4**, the centroid is inside the influence area: the detailed collision test is conducted. Collision takes place.*

Figure 6. Influence areas with length equal to the radius of the bounding circumference of the polygon.

2D Polygon-Polygon Collision Detection Test

We have seen how to check the intersection between two static polygons. Let us suppose a fixed polygon and a moving polygon (this scheme is also valid for two polygons in movement). We can use a combination of the *point-polygon collision detection algorithm* and the *polygons intersection test* by means of *influence areas*. The purpose is to verify at initial time whether collision between the polygons takes place or not (by means of the static test of intersection) and, if it does not take place, to apply the *temporal coherence* together with *influence areas* to detect whether the moving polygon is inside the influence area of another polygon, so that the detailed collision detection test may be applied in that case.

Firstly, both polygons are surrounded by a circumference centered in the centroid. This way, if there is no intersection between the circumferences, a collision between polygons may be discarded. If a collision between circumferences should occur, we must check whether the moving polygon is inside the influence area or not (if the centroid is in the area). If it is not inside the influence area, the procedure is the same as for point-polygon collision detection but, instead of considering the side of the polygon, we must consider its extension, that is to say, the side of the corresponding influence area. If the point is inside the influence area, the polygon detailed collision detection is used (Figure 7).



*In **1**, we check whether intersection between circumferences occurs; no collision takes place. In **2**, there is intersection between circumferences, but the centroid is not in the influence area; intersection does not take place. This situation allows making use of temporal coherence. In **3**, there is intersection between circumferences, and the centroid is in the influence area. A detailed collision test between polygons is carried out.*

Figure 7. Collision detection with bounding circumferences and influence areas.

The number of intersection tests between the edges of both polygons may be reduced by calculating where the centroid of the moving polygon is situated, that is, under what edges' areas of influence. If the centroid is in one of these areas, it is likely to collide with the edge of that area (and probably with another one).

In Figure 8.a) we can see the centroid of the polygon in the influence area of an edge. It can only collide with this edge (if it were in more influence areas, it could collide with each of the edges involved with those areas).

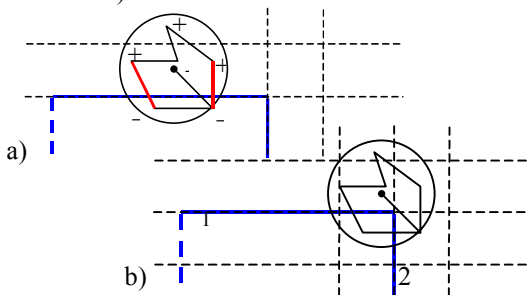


Figure 8. a) Influence area. In red, edges which can collide. b) Extended influence area.

In Figure 8.b) we can see that this reasoning is not altogether correct for, although it is still inside the same influence area (just one), edge 2 is also involved (and in fact it does collide with the polygon). This problem may arise in the vicinity of the vertices. In order to solve this, we have used the extended influence area of the polygon. If the centroid is in the extended influence area of an edge, that edge can collide with the polygon. Only the edges meeting this condition can collide with other edges of the polygon.

In addition, it is possible to reduce the number of edges of the polygon in movement that may be

involved in the collision. We need only check, with respect to each edge of the static polygon that can take part in the collision, the sign of *first barycentric coordinate s* of each one of the vertices of the polygon in movement. The edges that can collide will be those in which a change of sign in these barycentric coordinates takes place in the vertices (Figure 8.a). This algorithm is shown underneath (Algorithm 3).

Practical Implementation of Influence Areas

The *influence areas* are open zones, of infinite extension. They are somehow bounded by the circumference formed by the sum of the radiuses of the two bounding circumferences of both polygons (Figure 7). This bounding may not be suitable in certain circumstances, such as figures with edges far removed from the bounding circumference (in concavities or holes).

Make a triangles covering of the polygon with origin in the centroid of the figure.
Calculate the radius of the bounding circumferences
r = radius of the moving polygon bounding circumference
p = point that is the centroid of moving polygon
First step:
- Move the polygon
- If there is no intersection between bounding circumferences:
  - Return OUT
  - Go to the first step

Second step:
- Calculate the influence mask
- Compare with the previous influence mask
- If p moves out of some influence area, then go to the third step
- Else return OUT. Go to the first step

Third step:
- If p is in the influence area of length r of the polygon
  - Obtain the edges that may take part in the collision, using extended influence area of the polygon.
  - Make and return the polygon-polygon detailed intersecting test with edges calculated previously.
  - Go to the first step
- Else return OUT
- Go to the first step

Algorithm 3. 2D polygons collision detection test.

Ideally, the *extended influence area with length n of an edge* would be that with a distance to the edge smaller or equal to *n* (Figure 9.a). This ideal extended influence area is bounded by the definition of *Minkowski sum* of a segment and a circumference of radius n.[Lar00]

A point (the centroid of the moving polygon) outside this area ensures that the polygon it represents does not intersect with the static polygon. This space zone is difficult to implement. For this reason, we shall deal with a slightly greater zone and with a very small and efficient calculation time. This zone is bounded by the extended influence area and two lines parallel to axis X or Y (according to the slope), and at a distance *n* from the vertices (Figure 9.b).

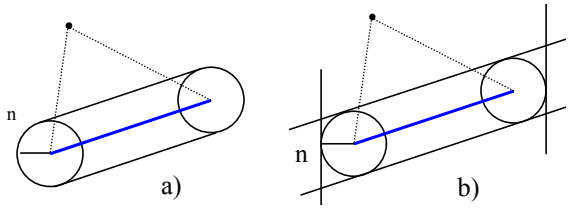


Figure 9. a) Ideal extended influence area with length n of an edge. b) Extended influence area implementation for –1<slope<1.

6. TIME STUDY

The *2D collision detection module* has been implemented as part of a *C++ 3D graphic library*, with an *object-oriented approach*. The graphics standard that has been used is *OpenGL* in a *Linux* platform.

In order to verify the efficiency of this algorithm, the times for *different types of trajectories and polygons* have been measured. These times have been compared with those from the *PIVOT2D library* [Hof01]. This library is one of the few that have been developed specifically for 2D objects. It uses *Voronoi regions* for its calculations of collision detection and makes use of *graphical hardware* in order to determine whether collision between polygons does take place or not and in order to obtain the pairs of features involved in it. This option has been deactivated for the calculations, so that it only detects whether collision takes place or not.

If necessary, the new algorithm developed allows directly obtaining the features involved in the collision in approximately the same time as the measured time. This measurement will be the object of a future study. The characteristics of the algorithms are shown underneath (Table 1).

Characteristic	PIVOT 2D	NEW
Uses graphic hardware	Yes	No
Uses Voronoi regions	Yes	No
Uses barycentric coordinates & coverings	No	Yes
Valid for non-convex polygons	Yes	Yes
Pre-processing	No	In construc-tion time
Uses hierarchical structures	When needed	No
Uses bounding volumes	Yes	No*
Tessellation	No	No
Returns involved features	Yes**	Yes**
Algorithm based in space of:	Hybrid geometry and image based	Geometry-based
Calculus error	Yes, it depends on resolution. Bounded	Yes, it depends on precision
** They have not been used for the tests.* *** These options have been deactivated in the tests.*		

Table 1. Main characteristics of PIVOT2D and the NEW algorithm.

For both algorithms the *collision detection time* has been studied, whether collision takes place or not. In order to measure these times, a 360 MHz Pentium-II processor has been used. The times of two types of trajectories have been measured, a *circular* one close to the static figure, composed of 90,000 movements, so that the moving figure returns to its starting point (Figure 10.a), and a *linear* one, composed of 9,000 movements, so that it draws near to the static polygon and collides with it (Figure 10.b).

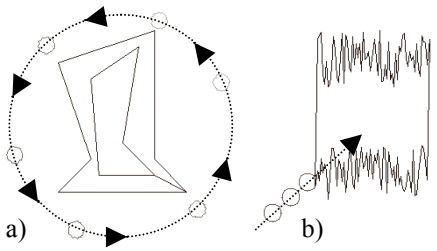


Figure 10. Trajectory types: a) circular. b) linear.

In order to perform the tests, different *types of polygons* have been used (Figure 11): *convex and non-convex* (the latter including *starred, contour, irregular, and with-holes polygons*). The times have been measured on the basis of the number of vertices of both figures. The following table (Table 2) summarises the characteristics measured.

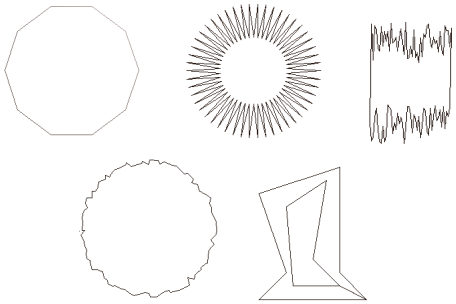


Figure 11. Types of polygons: Convex, starred, irregular, irregular contour, with hole.

Circular trajectory (Figure 10.a)	Rotatory movement of the moving polygon around the center of the static polygon. The movement is very close between the polygons and is made up of 90,000 positions, returning to the starting point. No collision takes place along the whole trajectory .
Linear trajectory (Figure 10.b)	Linear movement towards the static figure, composed of 9,000 positions. Collision takes place.
Number of Vertices	Vertices in both polygons between 8 & 1024 (powers of 2)
Type of polygons (Figure 11)	Variation of the type of static and moving polygon: Convex and non-convex (starred, contour, irregular)

Table 2. Main characteristics measured.

Different tests and time measurements in collision detection have been made. The reason why we have used circular trajectories and contour polygons in most cases is the difficulty the new algorithm faces in these situations, because the polygon in movement is continuously moving from one influence area to another and most of the time it is very near to the static polygon. We are considering one of the worst cases or situations; in this case, all the areas are crossed and the moving polygon is nearly always inside some influence area. We have chosen an initial distance between polygons of 9 units, because we did not want to penalize to the *PIVOT2D* algorithm, which offers worse results when the polygons are closer to one another.

Influence of the Number of Vertices

This test aims to measure the *influence of the number of edges in both polygons*. A circular trajectory is used and the starting situation positions the polygons are very close but not touching one another.

It can be seen (Figure 12) that the new algorithm uses less time in most situations, it being only slightly worse with very big static polygons (512-1024 vertices) and very small moving polygons (8-16 vertices).

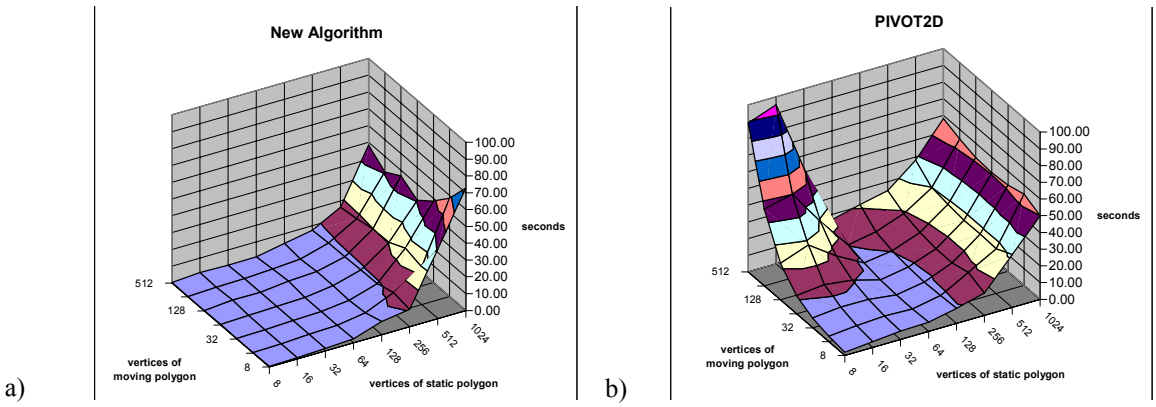


Figure 12. Times obtained in tests with a) the new algorithm and b) PIVOT2D. X and Z axes show the number of vertices of the static and moving polygons respectively.

This is due to the size of the influence areas (which depend on the radius of the bounding circumference of the moving polygon) and to the size of the edges of the static polygon. The greater the number of edges of the static polygon, the smaller the edges, because the figure is enclosed within the same volume. Therefore, both the size of the different influence areas and the degree of spatial coherence diminish, because a point moves from one area to others with greater frequency, thereby reducing the effectiveness of the algorithm.

The *PIVOT2D* behaviour becomes worse in the case of static polygons with few vertices (8-32 vertices) and of polygons in movement with many vertices (128-512 vertices), whereas the new algorithm achieves quite low times. In all the remaining performed tests (different types of polygons and linear trajectory), the results are similar or better to the obtained with contour polygons and circular trajectory.

7. CONCLUSIONS AND FUTURE WORK

We have obtained a *2D polygon-polygon collision detection algorithm* with better times than those provided by the *PIVOT2D* library in most situations. This algorithm is simple, more efficient and robust. Besides, it is suitable for any type of polygon, convex or non-convex, manifold or non-manifold, with or without holes, and, above all, it may be extended to 3D, which makes it especially attractive.

It uses a *triangles covering* of the polygons as pre-processing. This covering is made in a linear time based on the number of vertices, no type of complex data structure being necessary. The algorithm also uses the *geometric and temporal coherence*. Besides, once the collision is detected, we can obtain the edges taking part in it, almost at the same time. One final advantage is that it allows specifying a *distance* between objects.

The algorithm is being improved as far as its implementation is concerned. These improvements can offer us still better times than those reflected in this study. Some of these improvements would be: the efficient implementation of the operations between bit masks; the use of graphical hardware speeding up the operations; the use of techniques of space subdivision, invariants with rigid transformations; the use of the geometric coherence to calculate the edges that cross influence areas, so that it is not necessary to re-calculate them in the following movement; the extension of these techniques to several moving objects; and, finally, the use of bounding volumes hierarchies at different levels of detail. Extension to 3D is the most important work to be developed. It is also necessary to make a mathematical study of the speed of the algorithm based on the size of the influence areas and to obtain the times of effective calculation of the edges involved in the collision.

8. ACKNOWLEDGEMENTS

This work has been partially granted by the Ministry of Science and Technology of Spain and the European Union by means of the ERDF funds, under the research project TIC2001-2099-C03-03

9. REFERENCES

[Bad90] Badouel, F. An efficient Ray-Polygon intersection. Graphics Gems. Academic Press, 390-393, 1990

[Fei95] Feito, F; Torres, J.C.; et al; Orientation, Simplicity and Inclusion Test for Planar Polygons. Computer & Graphics, 19:4, 1995

[Fei97]Feito, F; Torres, J.C. Boundary representation of polyhedral heterogeneous solids in the context of a graphic object algebra. The Visual Computer, 13, pp. 64-77, Springer-Velag, 1997

[Fei98] Feito, F.R.; Segura, R.J.; Torres, J.C. Representing Polyhedral Solids by Simplicial Coverings. Set-Theoretic Solid Modelling, Techniques and Applications, CSG'98 Information Geometers, 203-219, 1998

[Hai94] Haines, E. Point in Polygon Strategies. Graphics Gems IV. Academic Press, 1994.

[Hof01] Hoff III, Kenneth E.; Zaferakis, A.; Lin M.; Manocha, D.; Fast and Simple 2D Geometric Proximity Queries Using Graphics Hardware. Symposium on Interactive 3D Graphics (I3D), 2001. http://www.cs.unc.edu/~geom/PIVOT/

[Hof89] Hoffmann, C. Geometric and Solid Modelling. An Introduction. Morgan Kaufmann Publishers, 1989.

[Jim01] Jiménez, P.; Thomas, F.; Torras, C. 3D collision detection: a survey. Computer & Graphics 25 (2001) 269-285

[Jim02a] Jiménez, J.J.; Segura, R.J.; Feito, F.R. Tutorial sobre Detección de Colisiones en Informática Gráfica. Novatica, Nº 157. May-June 2002, pp 55-58

[Jim02b] Jiménez, J.J., Segura, R.J., Feito, F.R.. Algorithms for Point-Polygon Collision Detection in 2D. 1st Ibero-American Symposium on Computer Graphics, Guimaraes-Portugal, pp. 253-261, 2002

[Lar00]Larsen, E.; Gottschalk, S.; Lin, M.; Manocha, D.; Fast distance queries with rectangular swept sphere volumes. IEEE International Conference on Robotics and Automation, 2000

[Las96] Laszlo, M. Computational Geometry and Computer Graphics in C++. Prentice Hall, 1996.

[Lin98] Lin, M; Gottschalk, S. Collision Detection between Geometric Models: A Survey. IMA Conference on Mathematics of Surfaces, 1998.