Polygon-Polygon Collision Detection in 2D

Juan J. Jiménez Delgado, Rafael J. Segura Sánchez, Francisco R. Feito Higueruela

Departamento de Informática. Escuela Politécnica Superior. Universidad de Jaén. Jaén. Spain.

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[1] 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.

Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Geometric algorithms, languages and systems; Curve, surface, solid, and object representations.

Additional Keywords and Phrases: 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.

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 [2].

On the other hand, the barycentric coordinates of a point regarding a triangle are used in order to determine the point or polygon inclusion [3]. 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: First, 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 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 [1] 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. Previous Work

Previous work has carried out a characterisation of the collision detection problem and the strategies used to solve it [4]. Other authors have also made a revision of this problem [5, 6].

© The Eurographics Association 2003.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

To study the inclusion of a point in a polygon we used the algorithm proposed in [7] 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.*

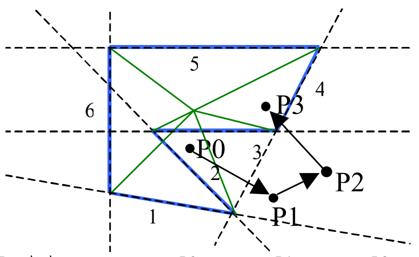
In [8] we can see different algorithms for the collision detection between a point and several types of figures (convex and non-convex, such as starred figures, contour maps and totally irregular figures). These algorithms are optimised according to the type of figure. Besides, the non-convex collision detection algorithm adapts to all the situations and offers quite good times.

```
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*

We have compared the point-polygon collision detection algorithm with the crossings test inclusion [9, 10] and the signed area [7, 11] 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 [8], 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.



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.

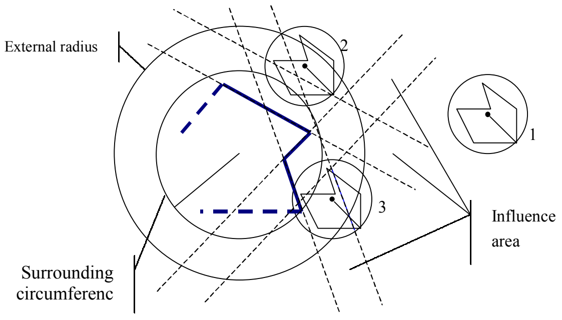
Figure 1*: Sample operation of 2D point-polygon collision detection algorithm. A covering of the polygon by triangles has been carried out and shows a division on zones.*

3. Developed algorithm

3.1. 2D polygon-polygon collision detection test

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.

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 2).



*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 2: *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).

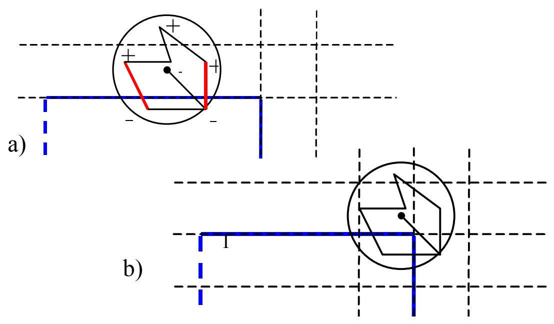


Figure 3: *a) Influence area. In red (vertices sign change), edges which can collide. b) Extended influence area*

In Figure 3.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). In Figure 3.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 3.a). This algorithm is shown underneath (Algorithm 3).

```
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 circum-
    ferences:
        - 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 in-
        tersecting 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.*

4. Time study

In order to verify the efficiency of this algorithm, the times for *different types of trajectories and polygons* have been measured (a *circular* one close to the static figure (Figure 4.a), and a *linear* one, so that it draws near the static polygon and collides with it (Figure 4.b). These times have been compared with those from the *PIVOT2D library* [1]. The times obtained in circular trajectory can be seen in Figure 5.

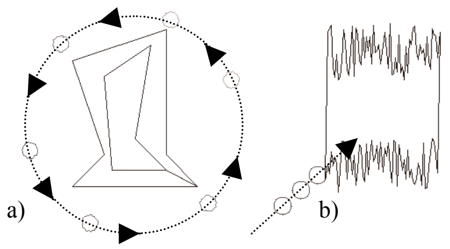
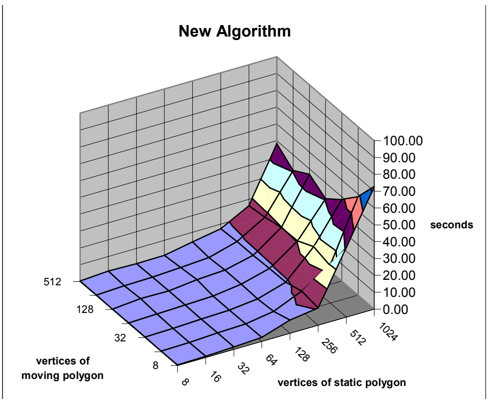


Figure 4: Trajectory types: a) circular. b) linear.

a)



b)

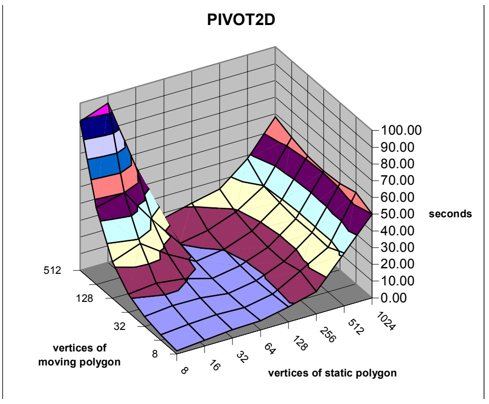


Figure 5: *Times obtained in tests with a) the new algorithm and b) PIVOT2D. X and Z axes show the number of vertices of the static and moving polygons respectively.*

5. 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.

References

1. 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), March, 2001. http://www.cs.unc.edu/~geom/PIVOT/
2. 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
3. Badouel, F. An efficient Ray-Polygon intersection. Graphics Gems. Academic Press, 390-393, 1990
4. 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
5. Jiménez, P.; Thomas, F.; Torras, C. 3D collision detection: a survey. Computer & Graphics 25 (2001) 269-285
6. Lin, M; Gottschalk, S. Collision Detection between Geometric Models: A Survey. IMA Conference on Mathematics of Surfaces, 1998.
7. Feito, F; Torres, J.C.; Ureña, L.A. Orientation, Simplicity and Inclusion Test for Planar Polygons. Computer & Graphics, Vol. 19, N 4, 1995
8. 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
9. Haines, E. Point in Polygon Strategies. Graphics Gems IV. Academic Press, 1994.
10. Laszlo, M. Computational Geometry and Computer Graphics in C++. Prentice Hall, 1996.
11. Hoffmann, C. Geometric and Solid Modelling. An Introduction. Morgan Kaufmann Publishers, 1989