OPTIMIZING RAY-TRACING FOR COMPLEX SOLIDS

J. J. Jiménez, R. J. Segura, F. R. Feito
Departamento de Informática. Universidad de Jaén
Escuela Politécnica Superior,
Avda. Madrid, 35, 23071 - JAÉN
SPAIN
e-mail: {juanjo,rsegura,ffeito@ujaen.es}

ABSTRACT

Ray-tracing algorithm is very used to produce realistic synthetic images in Computer Graphics. Yet, the time necessary to obtain images of high quality is very high. The bottleneck of this method of visualization appears when the intersection between the rays and the differents objects appearing in the scene, is computed. In this paper, an implementation of the ray-tracing method is presented, using a robust and efficient algorithm to determine the intersection between a ray and a polygon in 3D. We think that the use of this algorithm could decrease the time necessary to produce realistic synthetic images.

Keywords: visualization, ray-tracing, intersection, realistic images.

1. INTRODUCTION.

One of the main objectives in the field of Computer Graphics is the generation of synthetic realistic images. For that, the methods more commonly used nowadays are, on the one hand, the ray-tracing method and on the other one, the method based on radiosity.

The main disadvantage that both methods present is that the time necessary for the generation of the scene is high, so they are not usually useful when it is attempted to carry out a system that generates the graphics in real time.

Most research developed at the moment in the field of visualization are basically focused on reducing the times for different algorithms. For that, several alternatives are proposed: reducing the number of objects to be treated in the scene by the use of occluders, reducing the number of intersections by the use of detail levels, or reducing the time necessary to determine the intersections between rays and objects.

In this article we present the application of a robust and efficient algorithm for the generation of images by the ray-tracing method. The main advantage of this algorithm is that it is valid for any kind of objects represented by their boundaries: manifold or non-manifold, with or without holes, with concave or convex sides. For the implementation of the ray-tracing method, only improvement techniques based on the use of bounding box have been implemented. Despite that, the times necessary for the generation of scenes have been reduced against the necessary ones in other software for ray-tracing.

In the first section, a revision of the main technique of ray-tracing will be carried out. In the next section, the algorithm of test of intersection segment-polygon proposed in [Segura98], will be reviewed. Then, the most relevant aspects of the implementation of ray-tracing method using such algorithm, will be presented. Finally, some images obtained will be shown.

2. RAY-TRACING ALGORITHM.

Ray-tracing [Foley90][Wat00] determines the visibility of surfaces tracing imaginary light rays, from the observer to the objects of the scene. For that, a projection centre (the observer) and a window on an arbitrary view plane are selected. The window can be considered a regular mesh, the elements of which correspond to pixels with the required resolution. For each pixel of the window, a visual ray that goes from the projection centre to the scene is released. The colour of pixel would be the one of the nearest object to the projection centre. Besides, in order to calculate shadows, an additional ray is also released from the intersection point with the nearest object to each of the light sources. If one of these rays of shadow intersects an object, such object is under shadow in that point and the algorithm ignores the contribution of the light source of the shadow ray.

The illumination model developed by Whitted and Kay spreads basically ray-tracing to include the specular reflection and the refractive transparency. Apart from shadow rays, this algorithm generates reflection and refraction rays from the intersection point. In turn, each of these reflection and refraction rays can generate recursively shadow, reflection and refraction rays. In this algorithm a branch of the rays tree ends when the reflected and refracted rays do not intersect an object or when a maximum depth established by the user, has been reached.

Whitted found that about 75 and 95% of time needed to generate an image, was devoted to calculate intersections. Because of that, the strategies to improve the efficiency of algorithm of ray-tracing, try to accelerate the calculation of these intersections, or to avoid such calculations. For that, the use of bounding box, coherence techniques, or Binary Spacial Partitioning (BSP trees) are extensively used in this method. Likewise, new techniques to improve the times of calculations of intersections between rays and different objects that can be included in the scene, appear regularly in literature [Badouel90] [Moller97].

In this article, this algorithm of rays tracing has been implemented using new algorithms for the calculation of intersections [Segura98] and using bounding-box to avoid the calculation of some of these intersections. Next, we explain the fundamentals of the algorithms of intersection calculation used.

3. TEST OF INTERSECTION SEGMENT-POLYGON.

The formulation of the problem of the determination of intersection between polygons and straight line segments can be expressed as follows : let polygon be P, formed by n points, p_1, p_2,..., p_n, and a straight line segment S , the extremes of which are q_1 and q_2, it is pretended to determine the intersection between both objects.

In the case 3D, the classic solution consists in determining firstly the intersection of the straight line with the plane on which the polygon is. Once such intersection is determined, the inclusion of the point in the polygon is studied. For that, both the polygon and the point are projected to 2D (generally, rejecting one of the three coordinates will be sufficient); then, the inclusion of the point on the projected polygon is studied. Any test of inclusion such as the crossing-count one [Haines94] or Feito´s [Feito95] can be used. In the case that the point is inside the polygon, then the ray intersects the polygon.

The problem of the previous process mainly lies on that the whole process is threatened by problems of integrity and precision. Besides, it is necessary to calculate the point of intersection with the plane, though afterwards, it is determined that, actually, intersection does not exist, as such point is not included in the inner part of the polygon. This makes calculations increase unnecessarily.

The algorithm.

Next, it is presented the algorithm for ray-polygon intersection test in 3D [Segura98]. The algorithm returns whether intersection between a ray (known the initial and final extremes) and a polygon exists or not. The process of covering the polygon by means of triangles could be made as a preprocessing of the algorithm.

```
int testIntersSegment(point3D Q,point3D Q’)
{
int cut=0;
if Q y Q’ are coplanar
then return ( FALSE )
else
     Cover the polygon with triangles
     Foreach triangle Ti
        Switch Ti.testIntersSegment (Q,Q’){
         VERTEX:       return TRUE;
         EXTERNALEDGE: return  TRUE;
         INSIDE:       cut+= Ti.sign();
         INTERNALEDGE: cut+= ½*Ti.sign();
        }
return (cut==1);
}
```

Figure 1: Intersection test ray-polygon.

The calculation times of the previous algorithm improve up to a 16% the time necessary for the calculation of intersections for polygons with few sides. In the case the polygon is convex, then the polygon covering will be disjoint (in fact it will be a classic triangulation) so, once an intersection with any of the triangles is found, it will not be necessary to continue the calculation for the rest of them. This can also accelerate the calculations necessary for the intersection test. The testIntersectSegment algorithm for a triangle can be seen in [Segura01].

4. IMPLEMENTATION DETAILS.

For implementation of algorithm an object-oriented approach has been used. A ray-tracing method which belongs to scene class, has been defined. A scene stores information about objects and lights.

Tests have been done with complex polihedric pieces and using point lights. Lights follow Warn´s model, using lighting cones. C++ language has been used for the design of the system. Classes have been designed for storing 3D objects, as well as scenes, lights, and so on. 3D objects have a number of attributes: colour, material, specular and diffuse reflection coefficient.

As for image generation strictly speaking, no antialiasing techniques have been used, merely releasing a ray per pixel. In spite of that, and because no round-sided polyhedrons have been used, results are good. As far as depth of tree of rays is concerned, it has been set in four, though in figures 2.a and 2.b two scenes can be seen, generated with depth 1 and 4 respectively.

The method is also efficient in case figure has been built up starting from a triangles-mesh (figure 2.h). In such a case again, the intersection test is simplified considerably, as it is not necessary to cover the different faces of the solid with triangles. Efficiency improvement is at 16% approximately, as compared to other algorithms of intersection calculations with triangles, although when we have to calculate the point of intersection exactly, if it exists, times are quite close to those of other algorithms like the one proposed in [Moller97] or [Badouel90].

So, we have made a study about the number of intersections calculated to obtain figure 2.a. Objects contain 30 and 40 vertices respectively, with 18 and 28 faces. The size of the final scene is 300x300 pixels. It has been carried out 5,052,567 intersection tests, and only there has been intersection in 351,387 cases, that is, a 7% of cases. So, a 93% of calculations of intersection points has been avoided.

So, we have only had to compute the intersection point in 7% of cases; as the method used to test the intersection is a 16% better than other algorithms when the intersection does not exist (that is, a 93% of cases), we have a very interesting improvement with respect to other algorithms to compute the intersection point.

5. CONCLUSIONS AND FUTURE WORK.

In this article we have just presented an application of an algorithm for determining the intersection segment-polygon proposed in [Segura98]. For scene generating, complex pieces, more than showiness and scene realism, have been used, proving that the mentioned algorithm increases the efficiency of ray-tracing method, as it accelerates the calculation of intersections.

For future work, the application of texture to objects and the improvement of their materials, will be carried out. Likewise, we intend to undertake an exhaustive comparative of times with other algorithms of intersection test applied to ray-tracing.

REFERENCES

[Agrawal94] Agrawal, A., Requicha, A.: A paradigm for the robust design of algorithms for geometric modelling, *Eurographics'94, Computer Graphics Forum*, Vol.13, No.3, pp.33-44, 1994.

[Badouel90] Badouel, F.: An efficient Ray-Polygon intersection, *Graphic Gems*, *Academic Press*, pp:390-393, 1990.

[Feito95] Feito, F.R.: Orientation, Simplicity and inclusion test for general polygons. *Computer & Graphics*, Vol. 19, No. 4, pp595-600, 1995.

[Foley90] Foley, J.D. e.a., *Computer Graphics. Theory and Practice.* Addisson Wesley, 1990.

[Glassner89] An introduction to ray-tracing. Ed. Glassner, A. *Academic Press*, 1989.

[Haines94] Haines, E.: Point in Polygon strategies. Ed. Glassner, A. Graphics Gems IV, *Academic Press*, 1994.

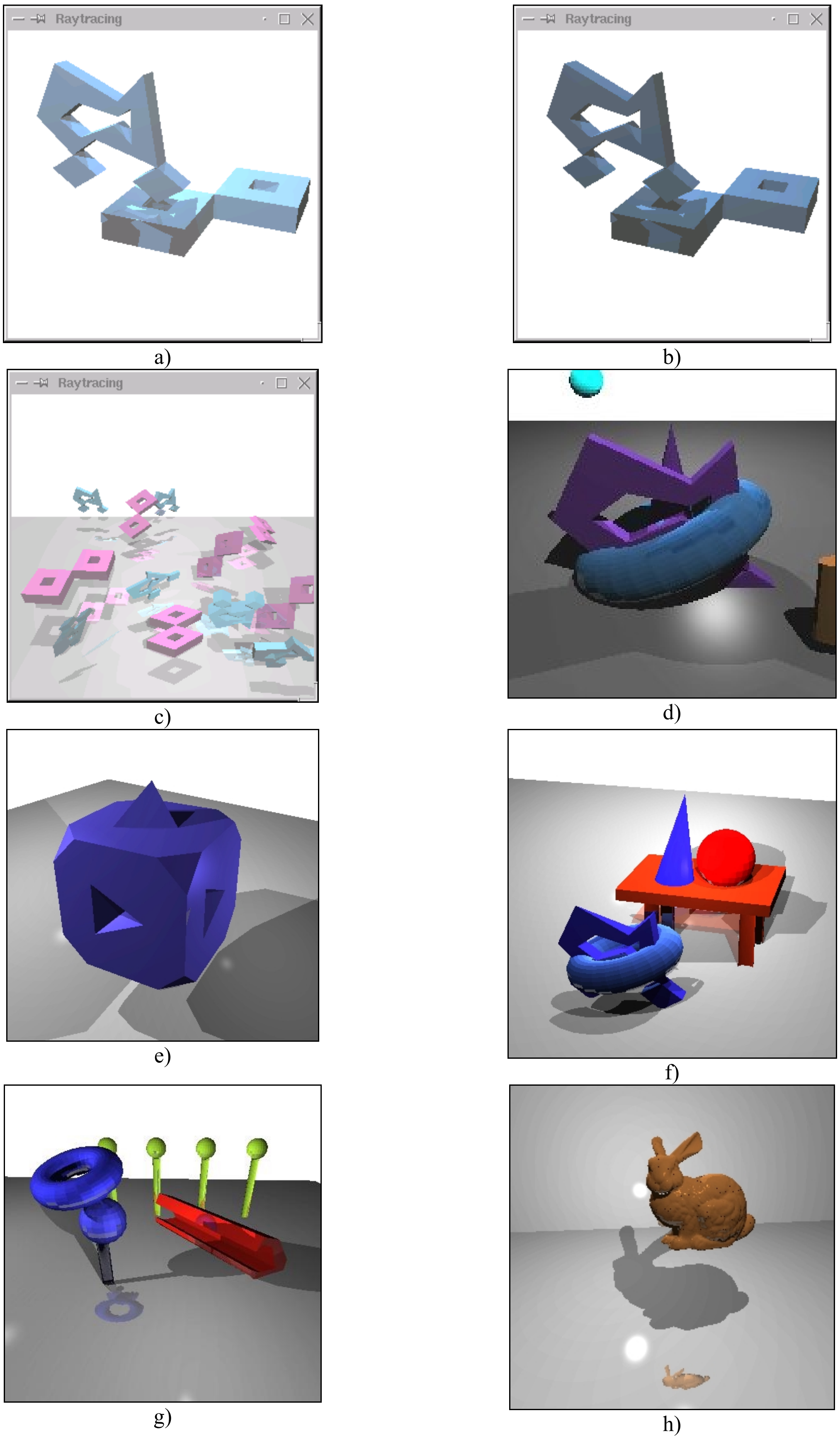
[Hoffman89] Hoffman, C.: Geometric and Solid Modelling. An introduction, *Morgan Kaufmann Publishers*, 1989.

[Moller97] Moller, T., Trumbore, B.: Fast, minimun storage ray-triangle intersection, *Journal on Graphic Tools*, Vol.2, No.1, pp.21-28, 1997.

[Segura98] Segura, R.J., Feito, F.R., An algorithm for determining intersection segment-polygon in 3D. *Computer & Graphics*, Vol. 23, No. 4, 1998.

[Segura01] Segura, R.J. Feito, F.R., Algorithms to test Ray-triangle intersection. Comparative study, *Proceedings of the 9th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2001. (Accepted).

[Watt00] Watt, A. 3D Computer Graphics, 3rd Ed. *Addisson Wesley*, 2000.



Scenes generated using the ray-tracer proposed in the paper
Figure 2.