1998 International Computer Symposium
Workshop on Computer Graphics and Virtual Reality
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

EFFICIENT DISCRETE-TIME COLLISION DETECTION AMONG POLYHEDRAL OBJECTS IN ARBITRARY MOTION

Chin-Shyurng Fahn and Jui-Lung Wang

Department of Electrical Engineering
National Taiwan University of Science and Technology
Taipei, Taiwan106, Republic of China

ABSTRACT

This paper presents an efficient discrete-time collision detection procedure for polyhedral objects that consist of convex polygons and perform arbitrary translating and/or rotating motions in a 3-D graphical environment. The discrete-time collision detection procedure can find the exact position of a collision event for a time step, which first localizes the object-to-object collision events with a "space cell" method. An "azimuth-elevation map" method is then proposed to rapidly select the polygons within or across the overlap region between two possibly collided objects by presorting their vertices in the spherical coordinates system. Subsequently, a divide-and-conquer method that takes advantage of bounding box representation is devised to moderate the number of polygons needed to be checked by a polygon-to-polygon intersection test. To deal with the discrete-time collision detection, a polygon-to-polygon intersection testing method based on a hierarchical scheme is developed to diminish unnecessary computation. So far, the experimental results from our proposed methods are very encouraging.

1. INTRODUCTION

Collision detection is widely employed in the fields of robot path planning, computer animation, and scientific simulation. It is also a crucial process in virtual reality to achieve the interaction between users and virtual worlds. To quickly response the input from users, efficient collision detection among polyhedral objects (called objects for short) must be accomplished in a 3-D graphical environment. Objects in the 3-D environment are frequently represented by polygons or parametric surfaces [1]. In general, a collision detection procedure can be roughly divided into two phases: an object-to-object overlap test and a polygon-to-polygon intersection test. The computational complexities of the two phases are $O(n^2)$ and $O(m_1m_2)$, where n is the number of objects and m_1 and m_2 are the numbers of polygons of the two objects possibly collided. To relieve the computation load, many methods are proposed in the literature [2-5].

In the object-to-object overlap test, bounding boxes and bounding spheres are commonly used. For each object in the 3-D environment, the bounding volume is generated to entirely surround the object. By this way, the test of each polygon of an object can be omitted when the bounding volume of the object does not collide with the others. However, the existing methods must check all pairs of objects in the environment [3, 4]. In fact, only few pairs of objects need to be detected. Accordingly, we can just trace the moving objects in the environment to reduce the computation load. Before the polygon-to-polygon intersection test, the polygons within or across the overlap region of two possibly collided objects must be picked out. The most common way is to test all the polygons of the two objects against the overlap region. It results in raising the computation load. Another improved method is to divide the bounding box of an object into smaller cells, each of which stores the associated constituting polygons [3]. During picking out the polygons within or across the overlap region, all the polygons of the cells associated with the overlap region are passed to the polygon-to-polygon intersection test. The drawback of this method is that the size of a cell can not be set perfectly. When the cell is larger than the overlap region, extra polygons are tested. If the cell is too small, conversely, a polygon may span several cells, and it requires multiple tests.

Spatial occupancy enumeration strategies, including octrees [6, 7], binary space partitioning (BSP) trees [8], and successive spherical approximation (SSA) representations [9], are also used to check the bounding volume overlap and pick out the polygons within or across the overlap region. Their common feature is that the space occupied by an object is decomposed into many subspaces in form of a hierarchical structure. Through a level-to-level test, the smallest collision region can be found. Nevertheless, constructing such a hierarchical structure is a time-consuming process, and the structure must be updated after the object performs geometrical transformations. Again, as mentioned above, the size of the smallest unit of the structure can not be decided well, and a polygon may span several subspaces [4]. It will increase the computation load of the polygon-to-polygon intersection test. Besides this, the aforementioned methods will enlarge the amounts of memory to store the hierarchical structures.

To accelerate the object-to-object overlap test, first of all, we divide the 3-D cyber space into equal cells, each of which is called a "space cell." When an object executes geometrical transformations (translations and/or rotations), the cells containing parts of the object will be made a sign. After all objects finish the transformations, only those cells with the sign will be checked. In consequence, the number of object pairs that need to be checked is diminished considerably. Because a point located in the spherical coordinates system remains its direction information, we transform the Cartesian coordinates of each vertex of an object into the spherical ones. When two bounding spheres

collide with each other, by calculating the cross angles of the two spheres, the polygons within or across the overlap region can be selected with an "azimuth-elevation (A-E) map" method, without testing all polygons against the overlap region.

However, the selected polygons of an object may not interfere with those of the other object, or there exist few polygons occurring in collision. To efficiently detect this, we propose a divide-and-conquer procedure of moderating the number of polygon pairs needed for test, which compares the bounding boxes composed of the two lists of the polygons within or across the overlap region of the two objects. This method will not generate the data structure of spatial occupancy enumeration, and avoids bringing the "span problem" of polygons. As for the polygon-to-polygon intersection test, we incorporate a hierarchical scheme [4] and extend the Cyrus-Beck algorithm [10]. The whole test procedure includes three steps: an overlap test of polygons using bounding boxes, a crossing test of an edge and a polygon, and an inside test of an intersection point and a convex polygon. After executing these steps, the crossing position of the two polygons can be obtained, if they intersect mutually.

2. THE OBJECT-TO-OBJECT OVERLAP TEST BASED ON SPACE CELLS

This section describes the first stage of our discrete-time collision detection procedure. Each object is represented as a bounding sphere, because its data structure is simple (the parameters of a sphere are only the center coordinates and the radius) and easy to check overlap (if the distance between the centers of two spheres is less than the sum of the radii of the two spheres). When objects are rotated or translated, all the bounding spheres merely update their center coordinates.

Because only moving objects have the chances to collide with the other objects, we can take note of the rotating and/or translating objects for each time step. In addition, only the objects near to the moving ones need to be checked, not all objects in the virtual world. A data structure that we call the space cell is used to record the statuses of objects in the world. To begin with, the world is partitioned into cubic cells. Then each cell can accept or cancel the registrations of objects when they move into or off the region of the cell. Following depicts our method:

1) If an object has been moved, cancel the registration of every cell in which the object was previously located.
2) Update the center coordinates of the bounding sphere of the object, then compute the serial numbers of the cells that cover the bounding sphere of the object, and register the object in the corresponding cells.
3) For all moving objects, check the registrations of the associated cells; if there are two or more objects in a cell, test their bounding spheres to find whether an overlap event occurs or not. Otherwise, no overlap event exists.
4) If the overlap event happens, pass the two possibly collided objects to the second stage for determining the pair of polygons that intersect.

With this method, every object keeps the data structure consisting of an array of eight items, and each item records the serial number of the cell which contains one of the eight vertices derived from the bounding sphere's center coordinates $(x_{center}, y_{center}, z_{center})$ and radius r as follows:

$$v_1 = (x_1, y_1, z_1) = (xcenter - r, ycenter - r, zcenter - r),$$
$$v_2 = (x_2, y_2, z_2) = (xcenter - r, ycenter - r, zcenter + r),$$
$$v_3 = (x_3, y_3, z_3) = (xcenter - r, ycenter + r, zcenter - r),$$
$$v_4 = (x_4, y_4, z_4) = (xcenter - r, ycenter + r, zcenter + r),$$
$$v_5 = (x_5, y_5, z_5) = (xcenter + r, ycenter - r, zcenter - r),$$
$$v_6 = (x_6, y_6, z_6) = (xcenter + r, ycenter - r, zcenter + r),$$
$$v_7 = (x_7, y_7, z_7) = (xcenter + r, ycenter + r, zcenter - r),$$
and $v_8 = (x_8, y_8, z_8) = (xcenter + r, ycenter + r, zcenter + r)$.

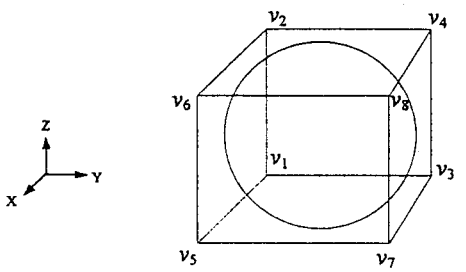


Fig. 1. The eight vertices of the cube surrounding a bounding sphere.

The eight vertices constitute a minimum cube surrounding the bounding sphere, as Fig. 1 shows. Using the coordinates of the vertices, we can easily compute the serial numbers of the eight cells in which the object falls. The formula of the serial number of the i-th cell is given below:

$$N_i = Iz_i \cdot n_x \cdot n_y + Iy_i \cdot n_x + Ix_i \qquad (1)$$

with $Ix_i = \frac{x_i - x_{min}}{2r}$, $Iy_i = \frac{y_i - y_{min}}{2r}$, and

$$Iz_i = \frac{z_i - z_{min}}{2r},$$

where x_{min}, y_{min}, and z_{min} are the coordinates of the minimum extent of the world; n_x, n_y, and n_z are the numbers of partitions in each dimension; r is the radius of the moving bounding sphere.

The size of the space cell is defined by the largest bounding sphere's radius of movable objects, and the edge length of the cell is two times of the object's radius. At the start of a realistic application, we can initialize an array of cells, whose size depends on that of the largest object. By this

way, we can calculate all the serial numbers of those cells which are covered by any object's bounding sphere through its eight vertices with little effort. Figure 2 illustrates the partition of a space in a 2-D representation, for example, *Object 1* is registered by *Cell 1* and *Cell 2*, whereas *Object 2* is registered by the four cells.

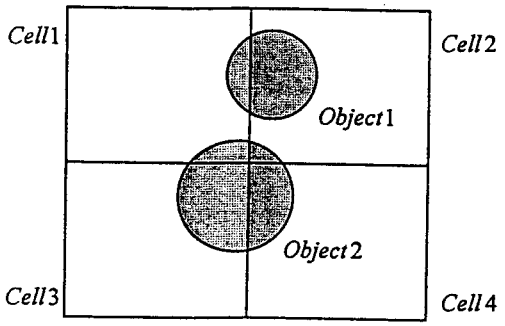


Fig. 2. An exemplary partition of a space with two objects.

3. SELECTING THE POLYGONS WITHIN OR ACROSS AN OVERLAP REGION

In the second stage of our discrete-time collision detection procedure, the pairs of possibly collided objects obtained from the first stage are checked where the collisions happen. The commonly used method often tests all the constituting m_1 and m_2 polygons of the pair of objects to filtrate those polygons across or within the overlap region, and further tests them in couples. Essentially, its computational complexity is $O(m_1 m_2)$ and can not be reduced. In this section, we will propose a method to economize the computing time of testing all the polygons across or within the overlap region. In order to relieve the computation load, we must take advantage of the basic data structures of computer graphics. That is, we must have the ability to acquire those polygons that share the same vertex. Besides, we also need an array to save some sorted information.

3.1 Azimuth-Elevation Mapping

As mentioned in the first stage, the bounding sphere is used as the minimum volume surrounding an object. It also plays an important role in this stage. At the beginning, the center coordinates $(x_{center}, y_{center}, z_{center})$ of an object are taken an average of the maximum and minimum coordinates of all the constituting vertices. Then the Cartesian coordinates of each vertex (x,y,z) are transformed into the spherical coordinates (ρ,θ,ϕ) relative to the center coordinates as:

$$\rho = \sqrt{x'^2 + y'^2 + z'^2}, \qquad (2a)$$

$$\theta = \tan^{-1}\frac{y'}{x'}, \qquad (2b)$$

$$\text{and } \phi = \tan^{-1}\frac{\sqrt{x'^2 + y'^2}}{z'}, \qquad (2c)$$

where $0^\circ \le \theta \le 360^\circ, 0^\circ \le \phi \le 180^\circ, x' = x - x_{center}$, $y' = y - y_{center}$, and $z' = z - z_{center}$.

After this transformation, the maximum value of ρ is selected to be the radius of the bounding sphere of the object. Subsequently, all the vertices are sorted according to both θ and ϕ values, and filled in a 2-D array called "azimuth-elevation (A-E) map." Figure 3 represents the A-E map, each element of which keeps the spherical coordinates of a vertex.

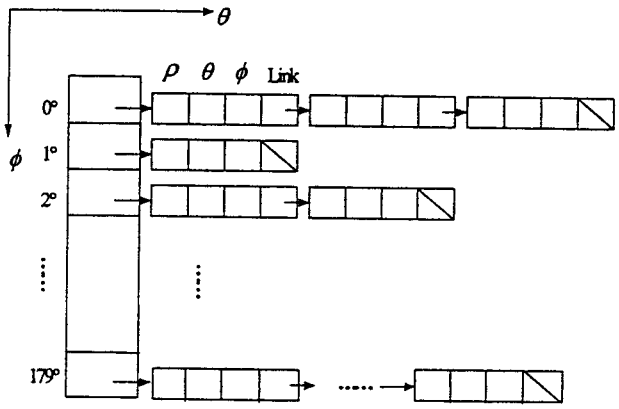


Fig. 3. An azimuth-elevation map in form of a 2-D array.

When the collision between two objects may happen, the following method is applied to pick out their constituting polygons that have the possibilities to collide with each other. Assume the two objects are denoted *Object 1* and *Object 2*.

1) Transform the center coordinates of *Object 2* into the spherical coordinates referring to that of *Object 1*, then the direction of *Object 2* related to *Object 1* is known.
2) From the sphere boundary, the cross angles $\psi_{1\theta}$ and $\psi_{1\phi}$ of *Object 1* are calculated as follows:

   If $\delta \ge r_2$ then $\psi_{1\theta} = \psi_{1\phi} = \cos^{-1}\frac{\delta - r_2}{r_1}$

   and $\eta = \delta - r_2$;

   if $\delta < r_2$ then $\psi_{1\theta} = 180^\circ$, $\psi_{1\phi} = 90^\circ$,

   and $\eta = 0$ (i.e., select all polygons),

   where r_1 and r_2 are the radii of the bounding spheres of *Object 1* and *Object 2*, respectively, and δ is the distance between the centers of the bounding spheres. Figure 4 shows an illustrative example for the case of the first if-statement.
3) Select those vertices on the A-E map of *Object 1*, whose azimuth and elevation angles satisfy: $\theta_{object2} - \psi_{1\theta} \le \theta \le \theta_{object2} + \psi_{1\theta}$ and $\phi_{object2} - \psi_{1\phi} \le \phi \le \phi_{object2} + \psi_{1\phi}$, as Fig. 5 illustrates. If ρ is less than η, then the associated vertex is discarded.

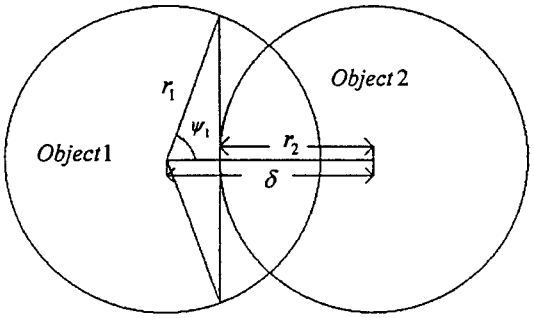


Fig. 4. Illustration of two overlapped bounding spheres.

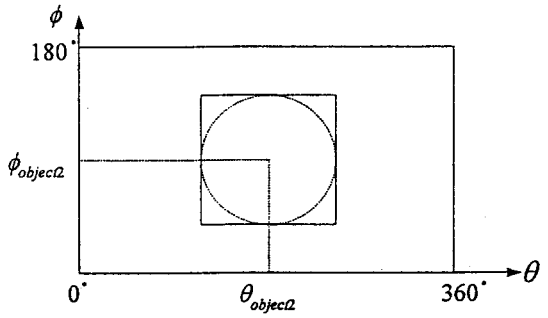


Fig. 5. An exemplary mapped overlap region between *Object1* and *Object 2* on the A-E map.

4) Put those polygons containing the selected vertices to a list.
5) Alternatively take *Object 2* as the center and repeat Step 1 to Step 4 to obtain another list.
6) Use these two lists to execute the polygon-to-polygon intersection test for finding the collision position.

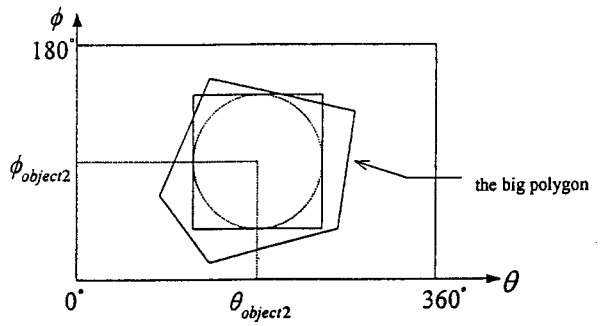


Fig. 6. The mapped overlap region across a big polygon.

Here a special case needs to be discussed. As shown in Fig. 6, the mapped overlap region is across a big polygon on the A-E map. If the big polygon crosses the overlap region of the two bounding spheres, all its vertices may be outside of the region. The equation of the first if-statement in Step 2 can solve this problem by enlarging the region to include all possible vertices, but it also increases the number of polygons managed in Step 6. If the object does not have big polygons, the following manipulation is preferred:

$$\psi_{1\theta} = \psi_{1\phi} = \cos^{-1}\frac{r_1^2 + \delta^2 - r_2^2}{2 \cdot r_1 \cdot \delta}. \qquad (3)$$

In Step 3, if the mapped region is across the boundary of ϕ, then the azimuth angle must be increased by 180° to acquire a correct region, as demonstrated in Fig. 7.

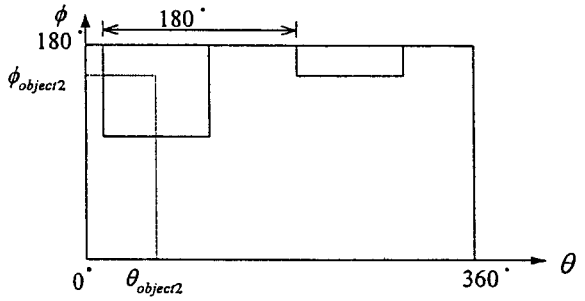


Fig. 7. The resulting overlap region across the boundary of ϕ.

3.2 The Solution to Rotated Objects

The method described in the previous subsection can only deal with the situation when objects move but do not rotate in the virtual world. If objects rotate, the direction we get can not generate the correct region on the A-E map to select the polygons that possibly collide with those of another object. The way to solve this problem is that each object must store the unit vector of three dimensions since the object was created. When an object performs rotations, its unit vector is also rotated to acquire the local axes of the coordinates of the object. For a rotated object, the transformation of its global Cartesian coordinates into the spherical ones is accomplished as follows. First, the global Cartesian coordinates are transformed into the local ones of the associated bounding sphere, and then transferred into the spherical coordinates, which preserves the correct position on the A-E map. By this way, the computing time of resorting every vertex on the A-E map is saved for the rotated objects.

4. SPEEDING UP THE POLYGON-TO-POLYGON INTERSECTION TEST

Prior to this, we obtain two lists of polygons, *list 1* and *list 2*, from two different objects possessing an overlap region. These two lists can compose $m_1 m_2$ polygon pairs, where m_1 is the number of polygons of *list 1* and m_2 is the number of polygons of *list 2*. In order to detect the occurrence of collision events, we must test each polygon pair for interference. In such an exhaustive manner, the computational complexity is $O(m_1 m_2)$. Following is our proposed method to reduce the computing time.

4.1 Overlap Test of Polygons Using Bounding Boxes

We use the bounding box as the auxiliary data structure to simplify the polygon-to-polygon intersection test. In order to economize the computing cost, we must choose the

polygon pairs that seem to be interfered, not all polygon pairs. To achieve this, we use the "minmax test" of a hidden-surface algorithm [11]. For each polygon, a data structure keeps both maximum and minimum of the three coordinates of the vertices of its bounding box. If the minmax test is true, then the polygons do not interfere with each other; but if the bounding boxes do overlap, we are still not confirmed whether the polygons within them have interference. In such a case, we must carry out a further test.

4.2 Crossing Test of an Edge and a Polygon

The idea of a hierarchical test described in [4] is used to check the interference event occurring in a pair of polygons. Given two polygons *Poly1* and *Poly2*. If the vertices of *Poly1* (*Poly2*) are all above or below the plane formed by *Poly2* (*Poly1*), there is no edge crossing the plane and no intersections exist. Otherwise, if the vertices of *Poly1* lie on the different sides of the plane formed by *Poly2* and vice versa, then an interference event is discovered. To determine the side on which a vertex lies, we substitute the coordinates of the vertex into the polygon's plane equation to yield the distance from the vertex to the plane, and check the sign of the result. Notice that the results of the same sign indicate the vertices on the same side of the plane. If the distances have different signs, then the vertices of a polygon lie on different sides of the plane. If there exist interference events, we will perform the following test to ensure whether the two polygons really intersect each other.

4.3 Inside Test of an Intersection Point and a Convex Polygon

Now we have all the distances from the vertices of a polygon to another polygon. Assume an edge is connected by two vertices p_i and p_j whose distances from the situated polygon to another are d_i and d_j, respectively. If d_i and d_j are of different signs, as shown in Fig. 8, then the intersection point p can be computed below:

$$p = p_i + |d_i|(p_j - p_i)/(|d_i| + |d_j|). \qquad (4)$$

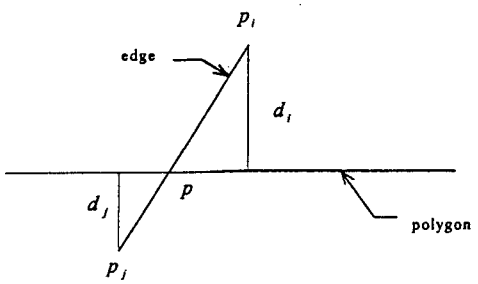


Fig. 8. The intersection point between an edge and a polygon.

Succeedingly, the 2-D Cyrus-Beck algorithm [10] is extended to a 3-D one to test whether the intersection point is inside a convex polygon. Because all the vertices are put counterclockwise to determine the visibility of a polygon, the outward normal vector of the edge $\overline{p_i p_j}$ is calculated by $\mathbf{N}_e = \mathbf{e}_{ij} \times \mathbf{N}$, referring to Fig. 9, where $\mathbf{e}_{ij}$ is the vector of the edge $\overline{p_i p_j}$ and $\mathbf{N}$ is the normal vector of the polygon. Then we take the dot product of the outward normal vector of each edge and the vector from a point p_e on the edge to the intersection point p. If each dot product is negative for all edges of the polygon through the above calculations, then the point p is inside the polygon; if not, it is outside. The crossing line of the two interfered polygons can be found by recording the cross points of the edges of the two polygons.

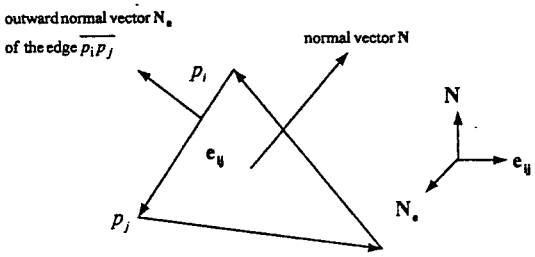


Fig. 9. Illustration of deriving the outward normal vector of an edge.

4.4 A Divide-and-Conquer Procedure

A divide-and-conquer procedure is devised to moderate the computing time of collision detection among objects. To begin with, the bounding boxes of the two lists of polygons are generated. Then the overlap test using bounding boxes is applied to check the overlap event. If no overlap occurs, then the procedure is terminated; otherwise, the minmax test method is employed to pick out those polygons within or across the overlap region to compose the other two lists of polygons. In the same manner, two new generated lists repeatedly perform the above procedure until the overlap region does not change.

5. EXPERIMENTAL RESULTS

To demonstrate the effectiveness of the proposed methods, many experiments are made on the discrete-time collision detection. All the testing programs have been implemented in C++ language by using the Microsoft Visual C/C++ 4.0 compiler under the operating system Microsoft Windows NT 4.0 of a personal computer with a Pentium Pro-180 CPU and 64M RAM. Because Windows NT is a multitasking operating system, we use the Win32 API function, GetThreadTimes, to get the CPU time during the execution of the programs [12]. The performance of the algorithms is tested without considering the time of screen I/O, so that all the programs are developed in the console mode of Windows NT to simplify coding.

5.1 Performance Test of Object-to-Object Collision Detection

In the following collision detection experiments, the object-to-object comparison is accomplished by two

simulation programs. The first program exhaustively detects all the bounding boxes of objects pair by pair, and the second program uses the space cell method, but not for each pair of bounding boxes.

First, the programs randomly generate both the center and radius of the bounding sphere of each object. The movable objects are animated along the predetermined path for 1,000 time steps. In these experiments, all the generated objects are set to be movable. Table 1 and Table 2 show the experimental results from the exhaustive method and the space cell method, respectively, on the same simulation condition. From the results of these two tables, we observe that the number of the pairs of objects required to compare with the latter method is much less than that with the former method. Consequently, the execution time needed for testing bounding boxes with the space cell method is considerably less than that with the exhaustive method.

TABLE 1
Object-to-Object Collision Detection by Use of the Exhaustive Method

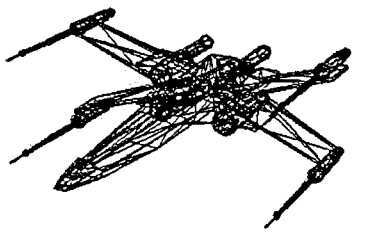
Object numbers	Compared pairs	Detected pairs	Execution time (100ns)
10	45,000	152	4,806,912
20	190,000	1,095	21,330,672
30	435,000	2,067	51,173,584
40	780,000	3,802	98,641,840
50	1,225,000	5,941	162,834,144
60	1,770,000	8,532	250,960,864
70	2,415,000	10,651	360,117,824
80	3,160,000	13,352	501,921,728
90	4,005,000	17,062	657,745,792
100	4,950,000	22,209	868,849,344

TABLE 2
Object-to-Object Collision Detection by Use of the Space Cell Method

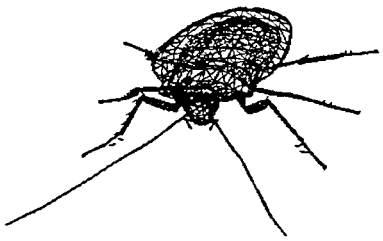
Object numbers	Compared pairs	Detected pairs	Execution time (100ns)
10	1,474	152	2,603,744
20	4,230	1,095	6,509,360
30	11,750	2,067	11,816,992
40	22,657	3,802	18,927,216
50	33,213	5,941	26,438,016
60	43,964	8,532	35,651,264
70	59,581	10,651	47,267,968
80	76,800	13,352	59,285,248
90	94,233	17,062	71,703,104
100	123,727	22,209	90,229,744

5.2 Performance Test of Selecting Polygons for Collision Detection

In the following experiments, a divide-and-conquer method of selecting polygons for collision detection will be tested, which is compared with a non-divide-and-conquer method. Figure 10 illustrates two of experimental objects that are retrieved from the web site: http://www.3dcafe.com. One is an X-wing, consisting of 1,293 vertices and 2,496 polygons, and another is a roach with 3,285 vertices and 5,491 polygons. During the testing, two same objects rotate about their centers and move toward each other to do the polygon-to-polygon collision detection. With the divide-and-conquer or non-divide-and-conquer method, the bounding boxes of the tested objects are updated by comparing all their vertices for each time step. Table 3 shows the number of the polygons of two roaches, denoted *Object 1* and *Object 2*, selected with the two methods around the occurrence of a collision event. It can be easily seen that the performance of the divide-and-conquer method is better than that of the non-divide-and-conquer one. All the selected polygons are finally passed to the polygon-to-polygon intersection test and the numbers of recursive calls are recorded. We can find that the pairs of polygons required to be checked are decreased or become zero after the recursive function is carried out. Table 4 shows the execution time of all detected pairs for each time step when the bounding boxes of the two objects have an overlap region.



(a)



(b)

Fig. 10. (a) A graphical X-wing with 1,293 vertices and 2,496 polygons; (b) a graphical roach with 3,285 vertices and 5,491 polygons.

5.3 Performance Test of Bounding Box and Azimuth-Elevation Map Methods

In this subsection, we compare the efficiency of filtering out the polygons within or across an overlap region of two bounding boxes with that of two bounding spheres. Besides the experimental objects mentioned before, Fig. 11 shows another experimental object- a ball that is composed of 1,986 vertices and 2,049 polygons. In the following experiments, two same objects perform rotating transformations about their centers and move close to each other. The bounding box and A-E map methods are compared by taking account of the amounts of execution time.

TABLE 3
Comparison of the Numbers of Polygons Selected with the Divide-and-Conquer and Non-Divide-and-Conquer Methods

Time step no.	Number of selected polygons				Number of recursive calls
	non-divide-and-conquer method		divide-and-conquer method		
	object 1	object 2	object 1	object 2	
65	0	0	0	0	0
66	0	0	0	0	0
67	0	0	0	0	0
68	0	0	0	0	0
69	72	0	72	0	1
70	461	0	461	0	1
71	653	53	653	53	1
72	785	570	178	0	2
73	1,112	1,235	56	1	6
74	1,346	1,725	8	2	6
75	1,410	2,037	0	20	5
76	1,618	2,196	0	102	5
77	1,641	2,674	16	0	6
78	1,348	2,759	0	140	8
79	1,241	2,600	432	517	5
80	1,044	2,358	541	725	5
81	764	1,946	709	843	3
82	651	1,421	468	905	4

TABLE 4
Comparison of the Execution Time of Divide-and-Conquer and Non-Divide-and-Conquer Methods

Time step no.	Execution time (100 ns)	
	non-divide-and-conquer method	divide-and-conquer method
65	1,602,304	1,502,160
66	1,402,016	1,402,016
67	1,502,160	1,602,304
68	1,502,160	1,502,160
69	5,007,200	4,907,056
70	5,207,488	5,107,344
71	8,111,664	5,608,064
72	39,456,763	5,407,776
73	104,550,336	6,108,784
74	185,266,400	6,108,784
75	229,630,192	6,108,784
76	280,903,920	6,409,216
77	354,109,184	6,709,648
78	282,506,224	6,409,216
79	251,261,296	23,233,408
80	197,584,112	36,852,992
81	121,074,096	51,073,440
82	72,303,968	37,754,288

Figures 12(a)~(c) demonstrate the comparison of the execution time required for choosing the polygons within or across the overlap region by use of the two methods for the three illustrative objects. Figure 12(a) shows the result

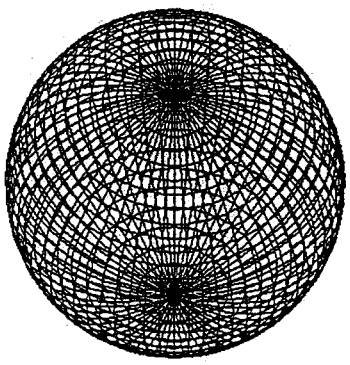


Fig. 11. A graphical ball with 1,986 vertices and 2,049 polygons.

of using the graphical ball as a tested object, where two graphical balls have the overlap region for the 70th time step. From this figure, we can find the efficiency of the A-E map method is better than that of the bounding box method. When using the X-wing as another tested object, as seen in Fig. 12(b), the overlap region detected with the A-E map method is earlier than that with the bounding box method; however, the efficiency of the former is less than that of the latter before the overlap region of the bounding boxes exists. Figure 12(c) exhibits the test result of two translating and rotating graphical roaches; since the roach is a very complicated and asymmetry object, the performance of the A-E map method is always worse than that of the bounding box method.

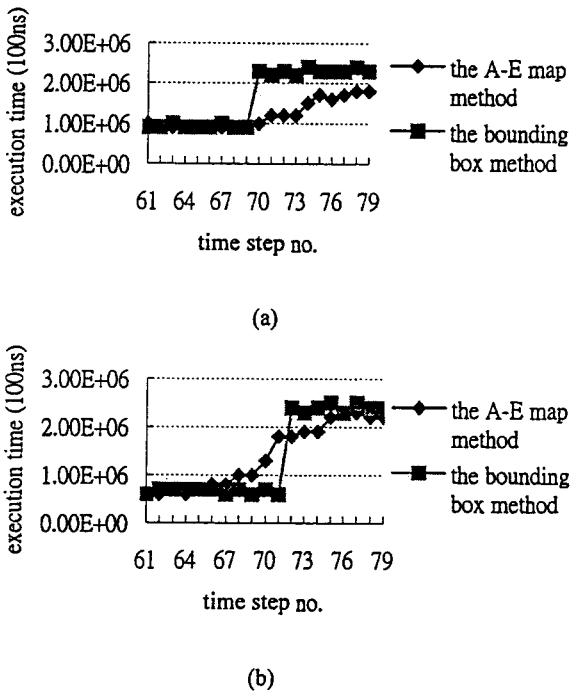


Fig. 12. Time required for picking out the polygons within or across the overlap region between (a) two balls; (b) two X-wings; (c) two roaches by use of the A-E map and bounding box methods.